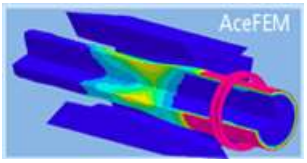
UNIVERSITY OF LJUBLJANA



FEM 7.0

University of Ljubljana, 2020
E–mail : AceProducts@fgg.uni-lj.si http://symech.fgg.uni-lj.si

Vertrieb durch:
ADDITIVE Soft- und Hardware für Technik und Wissenschaft GmbH
Max-Planck-Straße 22b • 61381 Friedrichsdorf
http://www.additive-mathematica.de • eShop: http://eshop.additive-net.de
Verkauf: +49-6172-5905-30 • mathematica@additive-net.de

ADDITIVE
SOFT- & HARDWARE FÜR TECHNIK & WISSENSCHAFT

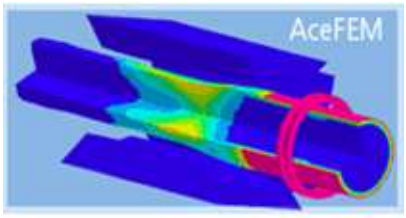
Contents

CHAPTER 1

Introduction

Introduction to AceFEM

Preface



© **Prof. Dr.-Ing. Jože Korelc**, 2006-2018
Ravnikova 4, SI - 1000, Ljubljana, Slovenia
E–mail : AceProducts@fgg.uni-lj.si
http://symech.fgg.uni-lj.si

The *AceFEM* package is a general finite element environment designed to solve multi-physics and multi-field problems. The *AceFEM* package explores advantages of symbolic capabilities of *Mathematica* while maintaining numerical efficiency of commercial finite element environments. The main part of the package includes procedures that are not numerically intensive, such as processing of the user input data, mesh generation, control of the solution procedures, graphic post-processing of the results, etc.. Those procedures are written in *Mathematica* language and executed inside *Mathematica*. The numerical module includes numerically intensive operations, such as evaluation and assembly of the finite element quantities (tangent matrix, residual, sensitivity vectors, etc.), solution of the linear system of equations, contact search procedures, etc.. The numerical module exists as *Mathematica* package as well as external program written in C language and is connected with *Mathematica* via the *MathLink* protocol. This unique capability gives the user the opportunity to solve industrial large-scale problems with several 100000 unknowns and to use advanced capabilities of *Mathematica* such as high precision arithmetic, interval arithmetic, or even symbolic evaluation of FE quantities to analyze various properties of the numerical procedures on relatively small examples. The *AceFEM* package comes with a large library of finite elements (solid, thermal, contact,... 2D, 3D,...) including full symbolic input for most of the elements. Additional elements can be accessed through the AceShare finite element file sharing system. The element oriented approach enables easy creation of customized finite element based applications in *Mathematica*. In combination with the automatic code generation package *AceGen* the *AceFem* package represents an ideal tool for a rapid development of new numerical models.

Acknowledgment

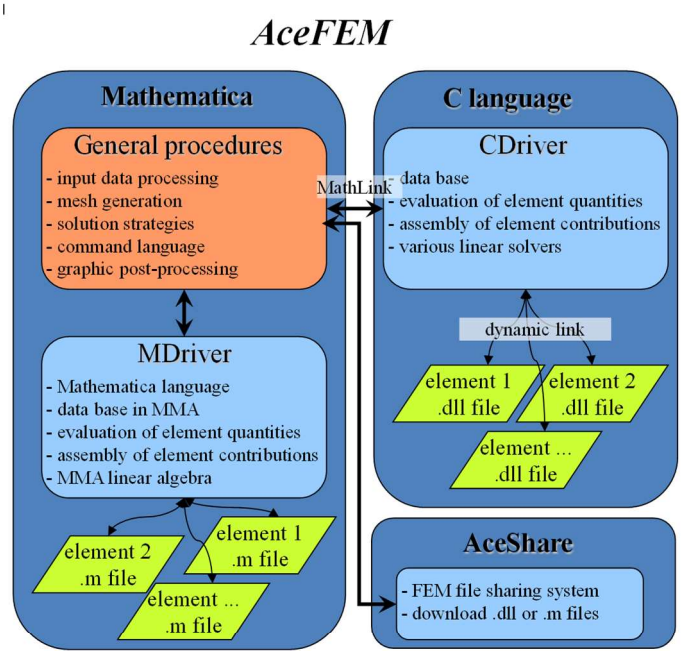
The *AceFEM* environment is, as it is the case with all complex environments, the result of scientific cooperation with my colleagues and former students. I would like to thank Tomaž Šuštar, Centre for Computational Continuum Mechanics d.o.o., Vandotova 55, Ljubljana, Slovenia for his work on *AceFEM* and especially implementation of various linear solver packages. I am also indebted to my friends Stanislaw Stupkiewicz and Jakub Lengiewicz, Institute of Fundamental Technological Research, Swietokrzyska 2, Warszawa, Poland for helpful discussions and implementation of contact search routines.

General

Commercial FE systems have incorporated ten to several hundred different element formulations. This of course can not be done manually without the use of reusable parts of the code (often element shape functions, material models, pre and post-processing procedures are written as reusable codes). The complexity of advanced numerical software arises also from other sources which include: necessity for realistic description of physical phenomena involved in industrial problems, requirements for highly efficient numerical procedures, and the complexity of the data structure. Normally the complete structure appears inside the FE environment

which is typically written in FORTRAN or C language. In the last decade or so the use of object oriented (OO) approach was considered as the main method of obtaining reusable and extensible numerical software. However, despite its undoubted success in many areas, the OO approach did not gain much popularity in the field of finite element methods, and all the main FE systems (ABAQUS, ANSYS, MARC, etc.) are still written in a standard way. Also most of the research work is still based on a traditional approach. One of the reasons for this is that only the shift of complexity of data management has been performed by utilizing OO methods, while the level of abstraction of the problem description remains the same. The symbolic approach can bypass this drawback of the OO formulation since only the basic functionality is provided at the global level of the finite element environment which is manually coded, while all the codes at the local level of the finite element are automatically generated. The *AceFEM* package has been designed in a way that explores advantages of this new approach to the design of finite element environments.

The *element oriented* concept is the basic concept behind the formulation of the *AceFEM* environment. The idea is to design a FE environment where code complexity will be shifted out of the finite element environment to a symbolic module, which will provide all the necessary formulation dependent codes by automatic code generation. The shift concerns the data structures (organization of environment, nodal and element data) as well as numerical algorithms at the local element level. The traditional definition of the finite element treats the chosen discretization of the unknown fields and a chosen variational formulation as the "definition of the finite element", while different material models then entail only different implementation of the same elements. This approach requires the creation of reusable code for material description and element description. In the present formulation an element will be identified by its discretization and material models, so no reusable code is needed at the local level. In principle each element has a separate source file with the element user subroutines. This is the way how the "*element oriented*" concept can be fully exploited in the case of multi-field, multi-physic, and multi-domain problems. Usually it is more convenient to have a single complex symbolic description and to generate several separate elements for various tasks, than to make a very general element which covers several tasks.



AceFEM organization scheme

The *AceFEM* package is a general finite element environment designed to solve multi-physics and multi-field problems. The *AceFEM* package explores advantages of symbolic capabilities of *Mathematica* while maintaining numerical efficiency of commercial finite element environment. The *AceFEM* package is designed to solve steady-state or transient finite element and similar type problems implicitly by means of Newton-Raphson type procedures.

The main part of the package includes procedures that are not numerically intensive such as processing of the user input data, mesh generation, control of the solution procedures, graphic post-processing of the results, etc.. Those procedures are written in *Mathematica* language and executed inside *Mathematica*. The second part includes numerically intensive operations such as evaluation and assembly of the finite element quantities (tangent matrix, residual, sensitivity vectors, etc.), solution of the linear system of equations, contact search procedures, etc.. The numerical module exists in two versions.

The basic version called *CDriver* is independent executable written in C language and is connected with *Mathematica* via the *MathLink* protocol. It is designed to solve industrial large-scale problems with several 1.000.000 unknowns. The element subroutines are not

linked directly with the *CDriver* but dynamically when they are needed. Consequently, there are as many dynamically linked library files (dll file) as is the number of different elements. The dll file is created automatically by the `SMTMakeDll` function. User can derive and use its own user defined finite elements or it can use standard elements from the extensive library of standard elements (`AceShare`).

The alternative version called *MDriver* is completely written in Mathematica's symbolic language. It has advantage that we can use advanced capabilities of *Mathematica*, such as high precision arithmetic, interval arithmetic, or even symbolic evaluation of FE quantities to analyze various properties of the numerical procedures on relatively small examples. The *MDriver* has the same data structures and command language as the *CDriver*, but due to the limited functionality and efficiency it should be primary used in element development phase for the problems with less than 10.000 unknowns. It also does not support advanced post processing, contact searches, etc..

The *AceGen* package represents suitable environment for debugging and testing of a new finite element before it is included into the commercial finite element environment. For example, the following tests can be performed directly in *Mathematica*:

⇒ convergence of iterative procedures,

⇒ different forms of the patch tests,

⇒ element distortion tests,

⇒ tests of the element eigenvalues,

⇒ test of objectivity.

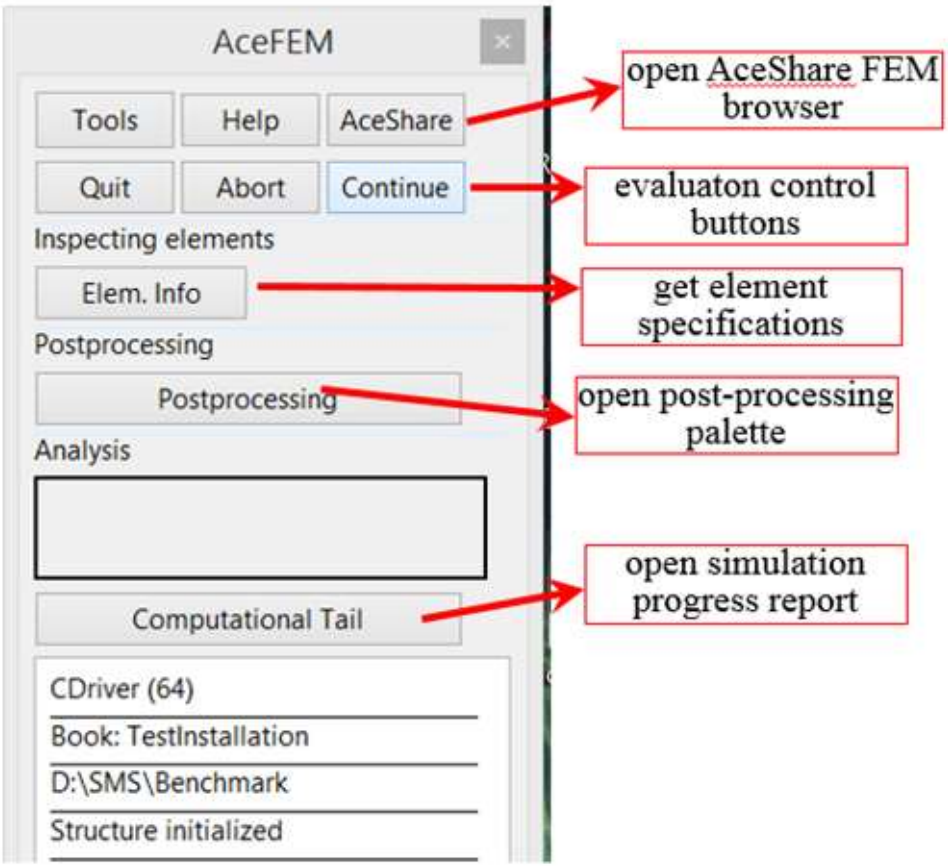
The *AceFEM* package has also some basic pre-processing and post-processing functions (`SMTAddMesh`, `SMTShowMesh`). It can be used for the geometries that can be discretized by the structured meshes of arbitrary shape. For a more complex geometries the commercial pre/post-processor has to be used. The *AceFEM* has built-in interface to commercial pre/post-processor GID developed by International Center for Numerical Methods in Engineering, Edificio C1, Campus Norte UPC, Gran Capitan, 08034 Barcelona, Spain, http://www.cimne.upc.es.

The *AceFEM* environment comes with a small built-in library including standard solid, structural, thermal and contact elements. Additional elements are accessed and automatically down-loadable through the *AceShare* system. The *AceShare* system is a finite element file sharing mechanism built in *AceFEM* that makes *AceGen* generated finite element source codes available for other users to download through the Internet. The *AceShare* system enables: browsing the on-line FEM libraries; downloading the finite elements from the on-line libraries; formation of the user defined library that can be posted on the internet to be used by other users of the *AceFEM* system. The *AceShare* system offers for each finite element included in the on-line library: the element home page with basic descriptions, links, authors data, etc.. , the *AceGen* template (Mathematica input) for the symbolic description of the element, the element source codes for all supported finite element environments (FEAP, AceFEM-MDriver, Abaqus, ...), the element Dynamic Link File (dll) used by *AceFEM*, an additional documentation and benchmark tests. The files are stored on and served by personal computers of the users.

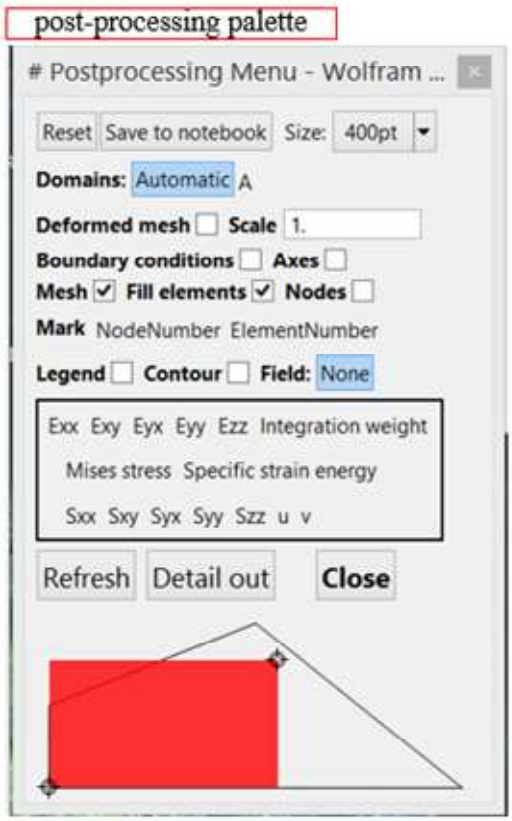
The already available *AceShare* on-line libraries include *AceGen* templates for the symbolic description of direct and sensitivity analysis of the most finite element formulations that appear in the description of problems by finite element method (steady state, transient, coupled and coupled transient problems). This large collection of prepared *Mathematica* inputs for a broad range of finite elements can be easily adjusted for users specific problem. The user can use the *Mathematica* input file as a template for the introduction of modifications to the available formulation (e.g. modified material model) or combine several *Mathematica* input files into one that would create a coupled finite element (e.g. the *AceGen* input files for solid and thermal conduction elements can be combined into new *AceGen* input file that would create a finite element for thermomechanical analysis).

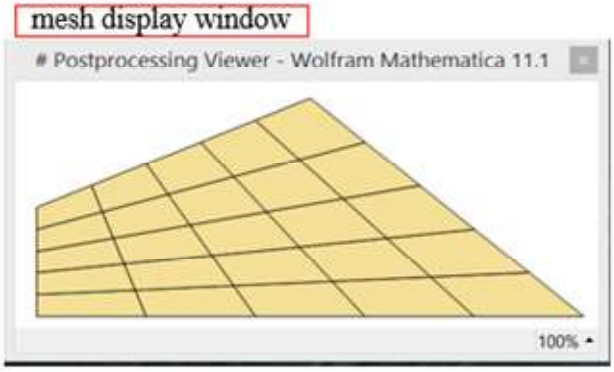
AceFEM Palettes

- Main AceFEM palette.

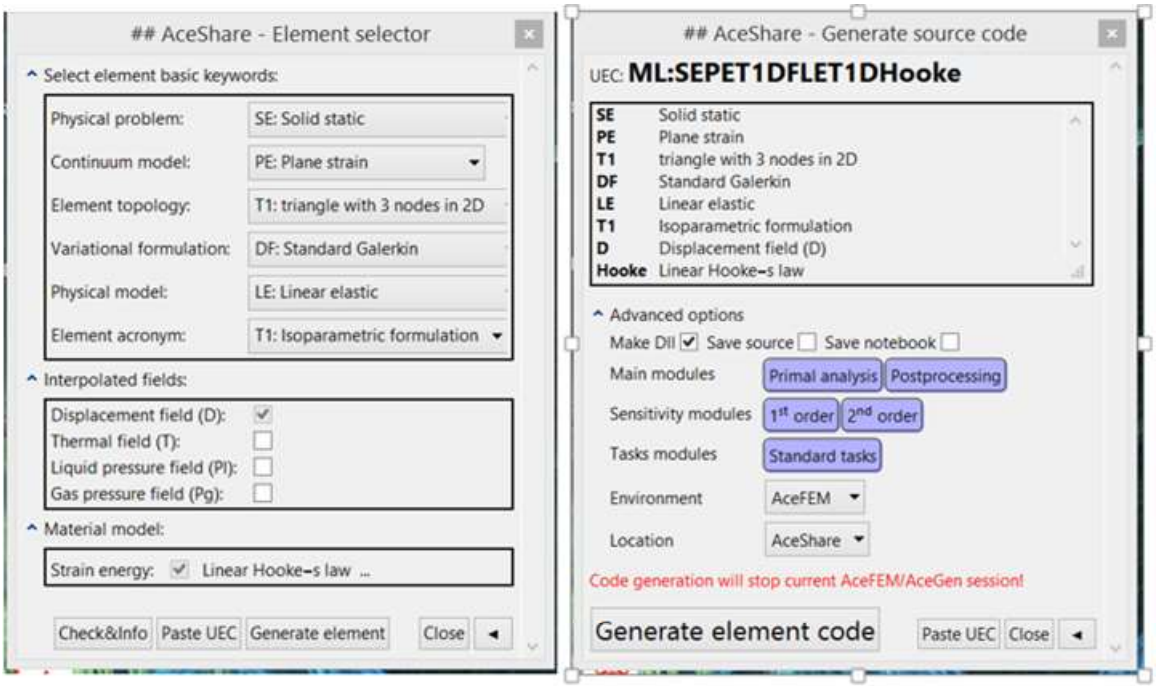


- Post-processing palette and mesh display windows.





- AceShare library browser and run-time element source code generation



AceFEM Overview

Input data phase

Input Data Phase

SMTInputData — start input data phase

SMTAnalysis — start analysis phase

SMTAddDomain — define element types

Element Mesh Generation

SMTAddMesh . SMTAddElement . SMTAddNode — mesh generation

Selecting nodes, elements and bodies

Boundary Conditions

SMTAddEssentialBoundary . SMTAddNaturalBoundary . SMTAddInitialBoundary. — define boundary conditions

Analysis phase

Analysis Phase

SMTNewtonIteration — perform one Newton type iteration

SMTConvergence . SMTNextStep . SMTStepBack — continuation procedures

Independent Batch Mode

SMTDump . SMTRestart — dump complete analysis to file and restart later

SMTStatusReport . SMTSessionTime . SMTErrorCheck . SMTSimulationReport — progress reports

User Defined Tasks

SMTTask — execute user defined tasks

SMTSetSolver — reset linear solver after the change of mesh, boundary conditions , etc.

Iterative Arc - length Solution Procedure

Linear Algebra

SMTSchurComplementOfEssentialBC - Schur complement of essential boundary conditions.

Post-processing and visualization

Visualization and Post - processing Phase

SMTShowMesh — show mesh and results

SMTResidual . SMTPostData . SMTData ....

SMTPut . SMTGet . SMTSave — save data to file and retrieve data from file for post-processing

SMTStatusReport . SMTSessionTime . SMTErrorCheck . SMTSimulationReport — progress reports

Complete data base control

Data Base Manipulations

SMTIData . SMTRData . SMTNodeData . SMTNodeSpecData . SMTElementData . SMTDomainData — manipulate node and element data

SMTFindNodes . SMTFindElements — select nodes and elements

Create shared finite element libraries

AceShare

SMTSetLibrary — initializes the library

SMTAddToLibrary — add new element to library

SMTLibraryContents — prepare library for posting

Advanced features

User Defined Tasks

Sensitivity Analysis

Contact Problems

Isogeometric Formulations

Summary of Examples

The examples given in examples section of the manual are meant to illustrate the general symbolic approach to computational problems and the use of AceGen and AceFEM in the process. They are NOT meant to represent the state of the art solution or formulation of particular numerical or physical problem.

All the examples come with a full AceGen input used to generate AceFEM source codes and dll files. All "dll" files are also included as a part of installation (at directory $BaseDirectory/Applications/AceFEM/Elements/), thus one does not have to create finite element codes with AceGen in order to run simulations that are part of the examples. More examples are available at http://symech.fgg.uni-lj.si/examples/ .

Standard performance evaluation tests

Basic examples how to use AceFEM

- Standard FE Procedure
- Simple bending of the column
- Bending of the column (path following procedure, animations, 2D solids)
- Boundary conditions (2D solid)
- Standard 6-element benchmark test for distortion sensitivity (2D solids)
- Cyclic tension test
- Solution Convergence Test
- Postprocessing (3D heat conduction)
- Advanced control of boundary conditions
- Round-off Error Test
- Houglassing test with animation
- Adaptive mesh refinement
- Analysis of Equilibrium Paths
- Analysis of cylindrical shell structure
- Global task that finds neighboring elements of the given set of elements
- Simulation of process of delamination
- Numerical efficiency of AceFEM data manipulations

Examples where AceGen is first used to generate elements

- Simple 2D Solid, Finite Strain Element
- Mixed 3D Solid FE, Elimination of Local Unknowns
- Mixed 3D Solid FE, Auxiliary Nodes
- Cubic triangle, Additional nodes
- Gas Pressure Element – Inflating the Tyre
- Sensitivity analysis
  - Finite Strain Element for Direct and Sensitivity Analysis
  - Parameter Shape and Load Sensitivity Analysis of Multi – Domain Example
- Elasto-plastic analysis
  - Three Dimensional, Elasto-Plastic Element

- Axisymmetric, finite strain elasto-plastic element
- Cyclic tension test, advanced post – processing, animations

- Dynamic analysis
  - Solid, Finite Strain Element for Dynamic Analysis
- Elements that Call User External Subroutines

Examples of Contact Formulations

- 2 D slave node – line master segment element
- 2 D indentation problem
- 2 D slave node – smooth master segment element
- 2 D snooker simulation
- 3 D slave node – triangle master segment element
- 3 D slave node – quadrilateral master segment element
- 3 D slave node – quadrilateral master segment and 2 neighboring nodes element
- 3 D slave triangle and 2 neighboring nodes – triangle master segment element
- 3 D slave triangle – triangle master segment and 2 neighboring nodes element
- 3 D contact analysis

Implementation of Finite Elements in Alternative Numerical Environments

- ABAQUS
- FEAP
- ELFEN
- User Defined Environment Interface

Summary of stochastic analysis examples

- Material parameter as stochastic variable, time independent, hyper-elastic problem, parameter is modeled as stochastic variable, perturbation approach;
- Stochastic analysis with Monte – Carlo method, time independent, hyper-elastic problem, parameter is modeled as stochastic variable, Mote-Carlo simulations;
- Material parameter as stochastic field, time independent, hyper-elastic problem, parameter is modeled as stochastic field, perturbation approach;
- Stochastic analysis of time dependent problems, time dependent elasto-plastic problem, parameter is modeled as stochastic field, perturbation approach.

CHAPTER 2

AceFEM Basics

Standard AceFEM Procedure

The standard *AceFEM* procedure is comprised of three major phases:

- **Input data phase (see `Input Data Phase`)**
  - Phase starts with `SMTInputData` followed by
  - element description (`SMTAddDomain`, actual element codes have to generated before the analysis by *AceGen* code generator or taken from `AceShare`,
  - definition of the mesh (`SMTAddMesh`, `SMTAddElement`...)
  - definition of boundary conditions (`Boundary Conditions`) and
  - sensitivity input data (`SMTSensitivityProblem`).
- **Analysis phase (see `Iterative Solution Procedures`)**
  - Phase starts with `SMTAnalysis` followed by
  - the solution procedure executed accordingly to the *Mathematica* input given by the user.
  - *AceFEM* is designed to solve time-independent or time-dependent finite element and related problems implicitly by the means of Newton-Raphson type procedures.
- **Visualization and post-processing phase (`SMTShowMesh`)**
  - Visualization of the results can be part of the analysis phase (`SMTShowMesh`) or done later independently of the analysis. (`SMTPut`)

Let us consider a simple one element example to illustrate the standard *AceFEM* procedure. The problem considered is steady-state heat conduction in a three-dimensional domain. The procedure to generate heat-conduction element that is used in this example is explained in *AceGen* manual section Standard FE Procedure. The element dll file (ExamplesHeatConduction.dll) is also included as a part of installation (in directory $BaseDirectory/Applications/AceFEM/Elements/), thus one does not have to create dll with AceGen in order to run the example.

Here the *AceFEM* is used to analyze simple one element example.

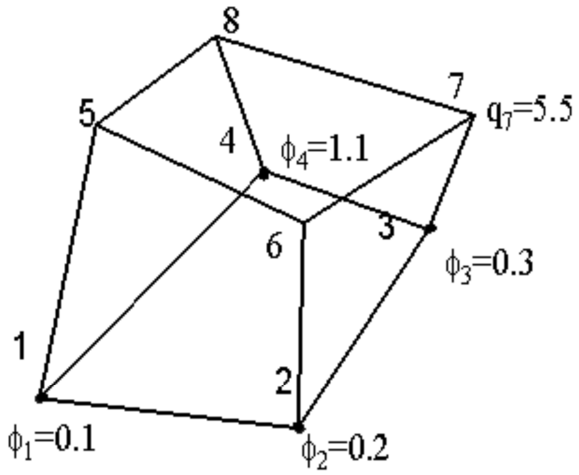
- This loads the *AceFEM* package, and prepares input data structures and starts input data section

In[9]:=
```
<< AceFEM`;
SMTInputData[];
```

- Here the domain description is given that defines the name of the element, the source code file with the element subroutines, the material data (in this case k_0, k_1, k_2) and the initial value of the heat source Q = 1.

In[11]:=
```
SMTAddDomain["A", "ExamplesHeatConduction",
  {"k0 *" -> 10., "k1 *" -> .5, "k2 *" -> .1, "Q *" -> 1.}];
```

- Here the element, the node coordinates and the boundary conditions of the problem depicted below.



In[12]:=
```
SMTAddElement["A", {{-0.5, 0, -0.5}, {1, 0, 0},
    {1, 1.5, 0}, {0, 1.5, 0}, {0, 0, 1.1}, {1, 0, 1}, {1.35, 1, 1}, {0, 1, 1}}];
SMTAddNaturalBoundary[{7, 1 -> 5.5}];
SMTAddEssentialBoundary[{1, 1 -> 0.1}, {2, 1 -> 0.2}, {3, 1 -> 0.3}, {4, 1 -> 1.1}];
```

- This checks the input data, creates data structures and starts the analysis. The SMTAnalysis also compiles the element source files and creates dynamic link library files (dll file) with the user subroutines (see also `SMTMakeDll`) or in the case of *MDriver* reads all the element source files into *Mathematica.*

In[15]:=
```
SMTAnalysis[];
```

- Here we define factor λ whit which the intensity of the boundary conditions and the heat source is multiplied. λ is traditionally refer as “load level”.

In[16]:=
```
SMTNextStep["λ" → 1];
```

- Here the problem is solved by the standard quadratically convergent Newton-Raphson iterative method. Observed quadratic convergence is also a partial confirmation that the problem was correctly linearized. This test can be used as one of the code verification tests.

In[17]:=
```
While[SMTConvergence[10^-12, 10], SMTNewtonIteration[];
 SMTStatusReport[SMTPostData["Temperature", Point[{-0.5, 0, -0.5}]]]];
```

```
Step/Iter=1/1 λ/Δλ=1./1. ||Δp||/||R||=1.10033/3.66682 Events=0 Status=0/{} Tag={0.1}

Step/Iter=1/2 λ/Δλ=1./1. ||Δp||/||R||=0.0370544/0.142921 Events=0 Status=0/{} Tag={0.1}

Step/Iter=1/3 λ/Δλ=1./1. ||Δp||/||R||=
 0.0000510625/0.000203298 Events=0 Status=0/{} Tag={0.1}

Step/Iter=1/4 λ/Δλ=1./1. ||Δp||/||R||=
 8.83286×10^-11/3.5599×10^-10 Events=0 Status=0/{} Tag={0.1}

Step/Iter=1/5 λ/Δλ=1./1. ||Δp||/||R||=
 1.40404×10^-17/8.88612×10^-17 Events=0 Status=0/{} Tag={0.1}
```

- During the analysis we have all the time full access to all environmental, nodal and element data. They can be accessed and changed with the data manipulation commands (see Data Base Manipulations). This gives *AceFEM* flexibility that is not shared by other FE environments as shown on the following example.
  Here an additional step is made, however instead of increasing the load level, the temperature in node 3 is directly set to 1.5. More about the advanced control of boundary conditions can be found in chapter Advanced control of boundary conditions.

In[18]:=
```
SMTNextStep["Δλ" → 0];
SMTNodeData[3, "Bp", {1.5}];
While[SMTConvergence[10^-12, 10], SMTNewtonIteration[];];
SMTStatusReport[SMTPostData["Temperature", Point[{1, 1.5, 0}]]];
```

```
Step/Iter=2/5 λ/Δλ=1./0. ||Δp||/||R||=7.94178×10^-15
 /3.98158×10^-14 Events=0 Status=0/{Convergence} Tag={1.5}
```

- The SMTSimulationReport command displays vital characteristics of the analysis performed.

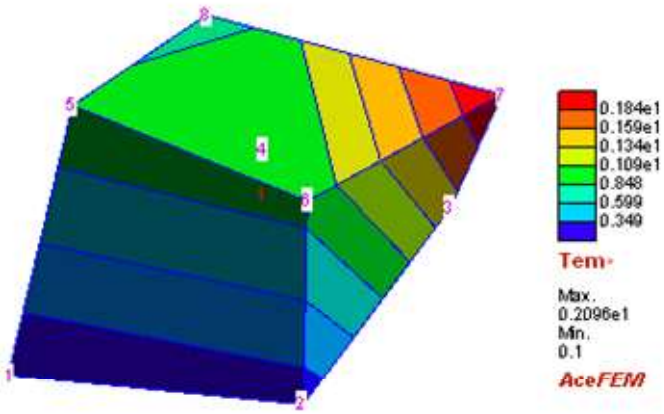
In[22]:=
```
SMTSimulationReport[];
```

No. of nodes	8
No. of elements	1
No. of equations	4
Number of threads used/max	8/8
Data memory (KBytes)	1
Tangent matrix (KBytes)	0
Solver memory (KBytes)	142
Total driver (KBytes)	143
MMA kernel memory (KBytes)	62 513
MMA front end memory (KBytes)	614 601
Total memory (KBytes)	677 257
Solver type	Pardiso
Matrix type	1
No. of steps	2
No. of steps back	0
Step efficiency (%)	100.
Total no. of iterations	10
Average iterations/step	5.
Terminal BC multiplier (λ)	1.
Terminal time (t)	0.
Terminal parameter (γ)	0.

Total absolute time (s)	0.3172100
Total driver time (s)	0.194
Total driver time (%)	61.1582
Total linear solver time (s)	0.101
Total linear solver time (%)	31.8401
Total K&R time (s)	0.000999928
Total K&R time (%)	0.315226
Average time/iteration (s)	0.0194
Average linear solver time (s)	0.0101
Average Ke&Re time (s)	0.0000999928
CPU Mathematica time (s)	0.078
CPU Mathematica time (%)	24.5894

- Here the SMTShowMesh function displays three-dimensional contour plot of the current temperature distribution. Additional examples how to post-process the results can be found in Solution Convergence Test.

In[23]:=
```
SMTShowMesh["Marks" → True, "Field" → "Tem*", "Contour" → True]
```



Input Data Phase

Contents

Overview

The obligatory parts of the input data phase are:

- The input phase starts with the initialization (see SMTInputData).
- The type of the elements and the data common to all elements of the specific type is specified by the SMTAddDomain command. The actual element codes have to be generated before the analysis (see Standard FE Procedure) or taken from the AceShare libraries.
- *AceFEM* is an element oriented environment that provides mechanisms for the elements to take active part in construction of the actual mesh. The topological mesh is a base on which the actual finite element mesh is constructed. If there are no elements that take active part in construction of the mesh, then are the actual mesh and the topological mesh identical.
  The node coordinates and the connectivity of nodes for topological mesh can be given by SMTAddNode, SMTAddMesh and SMTAddElement commands.
- The essential and the natural boundary conditions of the problem are specified by the SMTAddEssentialBoundary, the SMTAddNaturalBoundary and the SMTAddInitialBoundary commands. The imposed essential boundary conditions usually correspond to Dirichlet boundary conditions and the imposed natural boundary conditions to the weighted Neumann boundary conditions of the underlying partial differential equations. However, the actual relation between the imposed boundary conditions and the boundary conditions of the underlying boundary value problem has to be derived from the physical meaning of the nodal DOF. The physical meaning of the nodal DOF is implicitly defined by the algebraic equations associated with the nodal DOF. In AceFEM there are no predefined physical meanings of the nodal DOF. If the imposed essential and natural boundary conditions do not correspond to the requested boundary conditions of the underlying boundary value problem then true boundary conditions have to be imposed by the additional constraints. The additional constraints can be imposed by e.g. Lagrange multiplier method and implemented as separate finite elements.
- If sensitivity analysis is required then the SMTSensitivityProblem command specifies the type and the values of the sensitivity parameters.

Note that the node numbering can be changed after the input data phase due to the process of joining the nodes which have the coordinates and the node identification with the same value.

See also: Simple bending of the column

Input Data Phase Functions

SMTInputData

SMTInputData[]
initialize input data arrays and launch the numerical module (all the data from the previous AceFEM session is erased)

Initialize the input data phase.

The SMTInputData is the first command of every AceFEM session. See also: Standard AceFEM Procedure.

option	default	description
"LoadSession"	False	"*session_name*" ⇒ the data and definitions associated with the derivation of the element "*session_name*" are reloaded from the automatically generated file. The session name, the element source file name and the element name has to be the same for the proper run-time debugging. See also `Run Time Debugging`.
"NumericalModule"	"CDriver"	specifies numerical module: "CDriver" ⇒ C language "MDriver" ⇒ Mathematica language
"Precision"	$MachinePrecision	starts the MDriver numerical module with the numerical precision set to n (it has no effect on CDriver)
"Console"	False	starts the CDriver module as console application (or terminal application) and connect it with the Mathematica through MathLink protocol (it has no effect on MDriver)
"Threads"	All	sets the number of processors that are available for the parallel execution
"SeriesMethod"	"Lagrange"	specifies the power series expansion method (see also Semi-analytical Solutions)
"SeriesData"	False	specifies the power series expansion parameters, the expansion point and the order of the power series expansion $\{\{x,x_0,n_x\},\{y,y_0,n_y\},...\}$ (see also Semi-analytical Solutions)

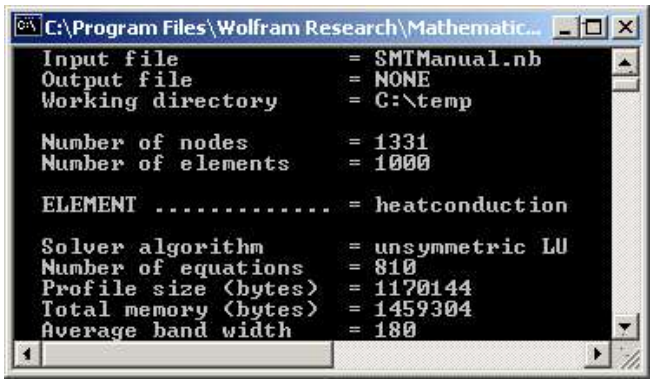
Options of the *SMTInputData* function.

Printing additional messages to terminal window

The *CDriver* numerical module is an executable written in C language and connected with the *Mathematica* through the *MathLink* protocol. Printing on standard output device (e.g. printf("hello")) from C language would **NOT** produce a print out to the *Mathematica* notebook but to the separate window depending on operating system used.

- Windows

  On Windows operating system the CDriver can be started in a separate window with the option "Console"->True. Additional messages are then printed to command window as shown below.



- Mac OS

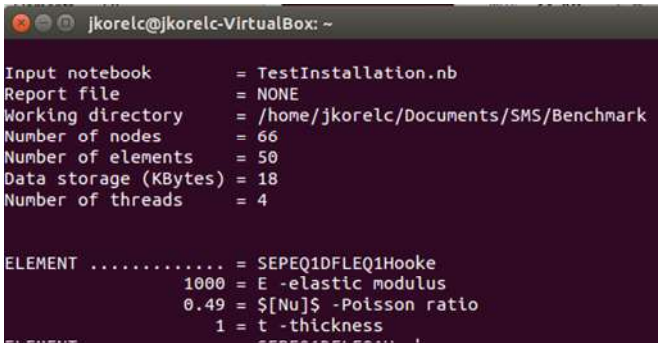
  In order to see printouts on Mac *Mathematica* has to be started from terminal as follows:

  - open Terminal (look under Applications-Utilities for Terminal.app),
  - start *Mathematica* form Terminal (e.g. /Applications/Mathematica.app/Contents/MacOS/*Mathematica*),
  - messages will now be printed to terminal window.

- Linux

In order to see printouts on Linux *Mathematica* has to be started from terminal as follows:

- open Terminal (e.g. search for Terminal),
- start *Mathematica* form Terminal (e.g.*Mathematica* &),
- messages will now be printed to terminal window as shown below.



SMTAddDomain

SMTAddDomain[*dID*,*UEC*,{*dkey*$_1$->*d*$_1$,*dkey*$_2$->*d*$_2$,...},*option*]
add domain data to the list of domains with input data given as a list of rules

The domain is identified by the unique string *dID* used within the session, the unified element code *UEC* (`Unified Element Code`) and the input data values that are common for all elements within the domain {d_1, d_2, ...}. The input data values are defined by the `SMSDomainDataNames` template constant and are specific for each element used. Input data values is given as a vector or a list of rules ($dkey_i \to d_i$). The keys of the data can be abbreviated (e.g. "Factor" can be given as "F*"). Only those values that are not equal to the default values (see `SMSDefaultData`) have to be given.

THe *UEC* is in the case of locally generated elements the name of the DLL file where the element definitions are stored. If the element is taken from the *AceShare* then the full `Unified Element Code` has to be given. For example, the "ML:SEPEQ1HRLEPianSumDHooke" represents *UEC* for the the well-known Pian-Sumihara element derived for isotropic Hook's material and stored in a main on-line AceFEM library "ML". The library UEC can also be given split to its constituent parts e.g. {"ML:","SE","PE","Q1","HR","LE","-PianSum","D","Hooke"}.

option	default	description
"Source"	Automatic	specification of the location of element's source code
"AdditionalData"	{}	additional data common for all the elements within a particular domain (e.g. flow curve)

Options for domain input data.

Several data sets can be added at the same time as well. For example:

```
In[24]:= SMTAddDomain[{{"A", "steel", {"E *" -> 21000, "ρ *" -> 2700}},
           {"A", "concrete", {"E *" → 3000, "ρ *" → 800}}}]
Out[24]= {Null, Null}
```

SMTMakeDll

SMTMakeDll[*source*]

Function creates dll file from the *source*. *source* parameter can be an AceShare element code or a name of c source file.

Function returns the following list:

{full dll file name

, *source*

, AceShare library index or 0 if the element is not from the AceShare}

SMTMakeDll[]
create dll file from the last generated *AceGen* code (if possible)

option	default	description
"OptimizeDll"	True	True ⇒ create dll file optimised by the compiler False ⇒ create dll file without additional compiler optimisation the data is not saved if the absolute difference in multiplier for two successive SMTPut calls is less than "MultiplierFrequency"
"AdditionalSourceFiles"	{}	list of additional source files (they are always recompiled)
"AdditionalLibraries"	{}	list of additional libraries
"AdditionalObjectFiles"	{}	list of additional object files
"Debug"	False	pause on exit and keep all temporary files

Options of the SMTMakeDll function.

The dll file is created automatically by the *SMTMakeDll* function if the C compiler is available. For details see installation instructions at http://symech.fgg.uni-lj.si/User/AceInstallation.htm .

SMTElementReport

SMTElementReport[*UEC*]
returns basic element characteristics as a list of rules.

In[77]:=
```
SMTElementReport[{"ML:", "SE", "PE", "Q1", "DF", "LE", "Q1", "D", "Hooke"}] // TableForm
```

```
UEC → {ML:, SE, PE, Q1, DF, LE, Q1, D, Hooke}
Code → SEPEQ1DFLEQ1DHooke
Topology → Q1
SpecIndex → 0
NoDimensions → 2
NoDOFGlobal → 8
NoDOFCondense → 0
NoNodes → 4
NoDomainData → 6
NoSegmentPoints → 10
IntCode → 2
NoTimeStorage → 0
NoElementData → 0
NoIntPoints → 4
NoGPostData → 13
NoNPostData → 4
SymmetricTangent → 1
NoIntPointsA → 4
NoIntPointsB → 0
NoIntPointsC → 0
NoSensNames → 0
ShapeSensitivity → 0
NoIData → 0
NoRData → 0
DefaultIntegrationCode → 2
LocalReKe → 0
NoAdditionalData → 0
NoCharSwitch → 0
NoIntSwitch → 1
NoDoubleSwitch → 0
dummy1 → -2 147 482 623
PostIterationCall → 0
Active → 1
DOFScaling → 0
EBCSensitivity → 0
SensitivityOrder → 0
WorkingVectorSize → 521
NoTopologyNodes → 4
SpatialDimensions → 2
ElementDimensions → 2
```

Example: One element

- Here follows the input data for a single element example depicted below.

In[270]:=
```
<< AceFEM`;
SMTInputData[];
```

- Here are the element material data, the integration code, and the type of the element. The element name is usually used also for the file name of the file where element subroutines are stored.

In[272]:=
```
SMTAddDomain[{"patch", {"ML:", "SE", "PE", "Q1", "DF", "LE", "Q1", "D", "Hooke"},
  {"E *" -> 3000., "ν *" -> 0.3, "t *" -> 2.}}]
```

Out[272]=
```
1
```

- Definition of nodes.

In[273]:=
```
SMTAddNode[{-0.5, 0.1}, {1.2, -0.3}, {1.1, 1.3}, {0.1, 1.4}]
```

Out[273]=
```
{1, 2, 3, 4}
```

- Definition of elements.

In[274]:=
```
SMTAddElement[{"patch", {1, 2, 3, 4}}]
```

Out[274]=
```
1
```

- Definition of boundary conditions.

In[275]:=
```
SMTAddEssentialBoundary[{1, 1 -> 0.015, 2 -> -0.12},
  {2, 2 -> 0.02}, {3, 2 -> 0.015}, {4, 1 -> 0.021}]
SMTAddNaturalBoundary[{2, 1 -> 105}]
```

Out[275]=
```
{4, 1 → 0.021}
```

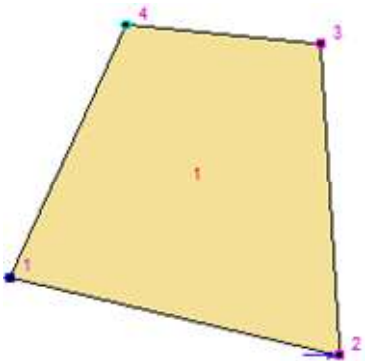
Out[275]=
```
{2, 1 → 105}
```

- Here the finite element data structures are established. **Note that the node numbering can be changed after the SMTAnalysis due to the process of joining the nodes which have the coordinates and the node identification with the same value.**

In[277]:=
```
SMTAnalysis[]
```

Out[277]=
```
True
```

In[278]:=
```
SMTShowMesh["Marks" → True, "BoundaryConditions" → True]
```



- 3 D mesh structured mesh types
- 2 D and 3 D surface structured mesh types with mesh refinement
- 3 D structured mesh types with mesh refinement

Introduction

Many numerical solution techniques work by replacing a region of interest with an approximation of that region. This approximation is called a discrete region. The discrete region is partitioned into a collection of smaller elements that, as a sum, make up the entire discrete region. This partitioned discrete region is called a mesh. Mesh is typically created as a part of input data phase by a pre-processor or mesher. *AceFEM* is an element oriented environment that provides mechanisms for the elements to take active part in construction of the actual mesh. The main mechanism is the *AceGen* option SMTAdditionalNodes that can be used to actively add additional nodes to elements generated by the mesher. The connection between the mesh and the elements is established by definitions of domains and associated domain identifications (see SMTAddDomain, Mixed Element Types, Element Topology).

Thus, within the AceFEM we can talk about two types of meshes:

- **Topological mesh** generated by the mesher
  - Topological mesh is defined by a list of node coordinates and corresponding connectivity table that defines a list of elements. The topological mesh is defined by the type of the mesh or **mesh type** and the type of the elements that form topological mesh or **mesh element type (see** Mixed Element Types).
- **Actual mesh** used to perform simulation
  - The topological mesh is a base on which the actual finite element mesh is constructed. The type of the elements of the actual mesh or **actual element type (**see Mixed Element Types, Element Topology**)** is specified by the domain identification *dID*.

Additionally we define:

- **mesh type**
  - Mesh type is the type of the mesh generated by the mesher (e.g. various structured meshes, unstructured meshes).
- **mesh element type**
  - Mesh element type is type of the elements that form topological mesh generated by mesher. Mesher can partition the region into elements of various types accordingly to the shape (line, triangle, quadrilateral, tetrahedron, hexahedron are the most usual shapes of elements generated by meshers) and various number of nodes that defines the order of interpolation (first order or linear, second order or quadratic are the most common orders).
- **actual element type**
  - The type of the elements of the actual mesh is defined by the element code (see Unified Element Code, Element Topology) and specified by the domain identification *dID* (see SMTAddDomain).

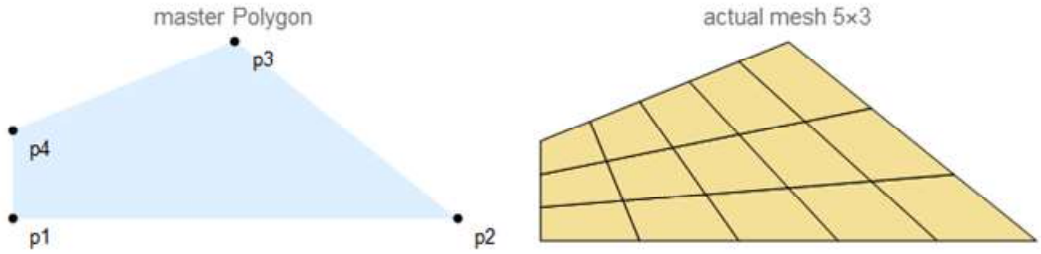
If there are no finite elements that take active part in construction of the mesh (self-generating meshes), then the actual mesh and the topological mesh are identical. Meshes are generated or imported by the SMTAddMesh command. The SMTAddMesh function returns a list of global node numbers of the nodes added and a list of global indexes of the elements added. The SMTAddMesh function can be repeated several times and used to combine several meshes generated by *Mathematica, AceFEM* or an arbitrary external mesher. In general it is assumed that the numbering of the nodes is done locally for each SMTAddMesh call. Thus, after the SMTAddMesh call the indexes of the nodes can be changed in rather unpredictable way.

Meshers

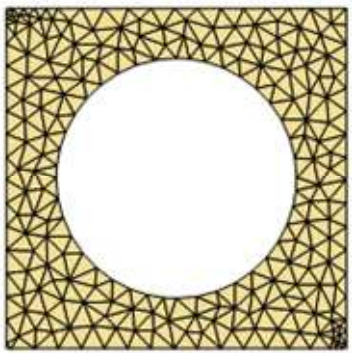
Withing the *AceFEM* meshes can be generated using three different meshers:

- built-in mesher that generates an arbitrary structured mesh (see Types of Structured Meshes) for an arbitrary shaped 2D quadrilateral or 3D hexahedron region and meshed with elements with an arbitrary topology (Element Topology).

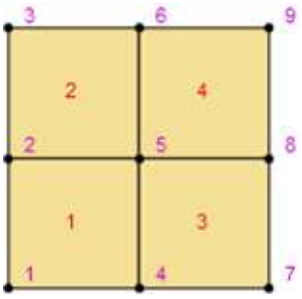
- For example SMTAddMesh[Polygon[{p_1,p_2,p_3,p_4}], *dID,* "Division"-> {n_x,n_x}] command generates structured mesh for an arbitrary quadrilateral region where {n_x,n_x} is the number of elements along local x and y axis. The domain identification *dID* defines the actual element type.
- SMTAddDomain["Ω", {"ML:","SE","PE","Q1","DF","LE","Q1","D","Hooke"}, {"E *" -> 1000, "*v* *" -> 0.3}]; SMTAddMesh[Polygon[{{0, 0}, {L, 0}, {L/2, 2 H}, {0, H}}], "Ω", "Division"-> {5, 3}]



- built-in mesher that generates unstructured mesh composed of triangular (2D) or tetrahedron (3D) elements of various orders,
  - For example SMTAddMesh[*mesh_ElementMesh*, *dID*] command adds mesh defined by *ElementMesh* data object generated by built-in *Mathematica* mesher (see `Element Mesh Generation`) and domain identification *dID* that defines the actual element type. The actual element type must correspond to the mesh element type.
  - SMTAddDomain["Ω",{"ML:","SE","PE","T1","DF","LE","T1","D","Hooke"}, {"E *" -> 1000, "*v* *" -> 0.3}]; SMTAddMesh[ToElementMesh[ImplicitRegion[x^2 + y^2 > 0.5, {x, y}], {{-1, 1}, {-1, 1}}, "MeshOrder" -> 1], "Ω"];



- mesh generated by an arbitrary third-party mesher that produce a list of nodes and element connectivity table in a format that can be imported into *Mathematica* and *AceFem* using the Import command.
  - For example SMTAddMesh[{{X_1,Y_1,Z_1},{X_2,Y_2,Z_2},…}, {dID_1->{{n_1^1,n_2^1,…},{n_1^2,n_2^2,…},…}, dID_2→{{n_1^1,n_2^1,…},{n_1^2,n_2^2,…},…}, …}] command imports mesh defined by the list of node coordinates {{X_1,Y_1,Z_1},{X_2,Y_2,Z_2},…} and a list of regions defined by domain identification dID_i and connectivity table, {{n_1^1,n_2^1,…},{n_1^2,n_2^2,…},…}.
  - SMTAddDomain[{"Ω",{"ML:","SE","PE","Q1","DF","LE","Q1","D","Hooke"}, {"E *" -> 1000, "*v* *" -> 0.3}}]; SMTAddMesh[{{0., 0.}, {0., 2.5}, {0., 5.}, {2.5, 0.}, {2.5, 2.5}, {2.5, 5.}, {5., 0.}, {5., 2.5}, {5., 5.}} , {"Ω" -> {{1, 4, 5, 2}, {2, 5, 6, 3}, {4, 7, 8, 5}, {5, 8, 9, 6}}}];



Bodies and boundary meshes

Elements can be grouped together to form a "body". Body is defined by its identification *bodyID*. *bodyID* is a string that uniquely defines the body and is defined as a part of mesh generation process (SMTAddMesh option "BodyID"). The primal purposes of defining bodies is to :

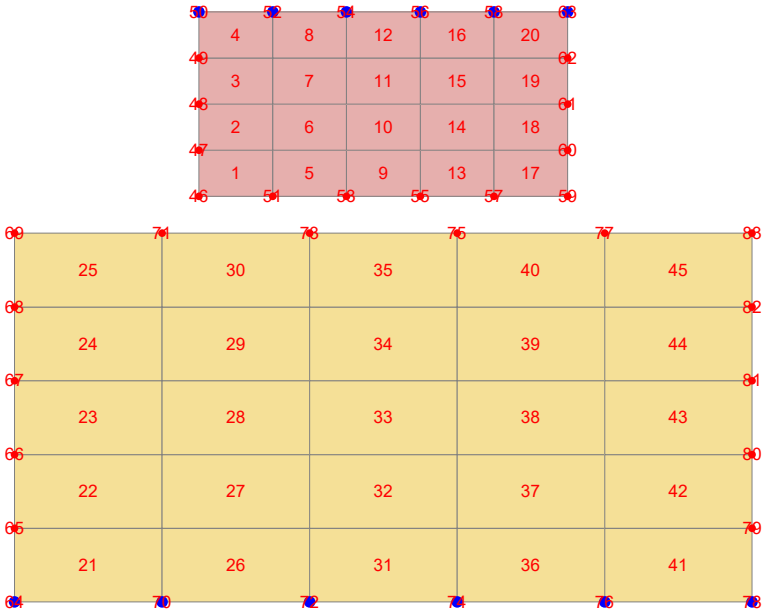
- select the elements/nodes that belong to the same body (e.g. SMTFindNodes[{"BodyID",bodyIDSelector}]),
- formulate contact problem (see `Implementation Notes for Contact Elements`),
- add elements at the body boundary (SMTAddMesh option "BoundaryDomainID").

Algorithm: The boundary of the body is first divided into line segments in 2D or triangular or quadrilateral segments in 3D. The surface segments are then used to create boundary mesh. Consequently, the boundary elements can only be first order elements ("L1", "P1" or "S1" topology) or points ("V2" or "V3" topology) on the surface.

Example 1:

Here two bodies are created ("B1" and "B2"). The boundary of the bodies is then enveloped into a layer of contact elements. Contact element is defined by its slave node (red dots).

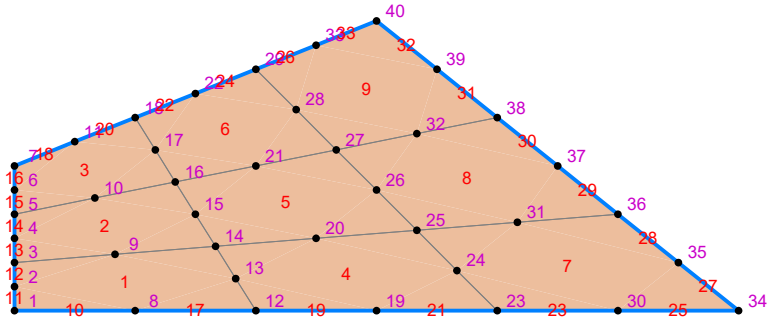
In[567]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[
   {"Solid1", "ExamplesHypersolid2D", {"E *" -> 70000, "ν *" -> 0.3}},
  {"Solid2", "ExamplesHypersolid2D", {"E *" -> 210000, "ν *" -> 0.3}},
  {"Contact", "ExamplesCTD2N1L1Pen", {"ρ *" -> 200000}}];
SMTAddMesh[Polygon[{{-1/2, 0.1}, {1/2, 0.1}, {1/2, 0.6}, {-1/2, 0.6}}],
  "Solid1", "Division" -> {5, 4},
 "BodyID" → "B1", "BoundaryDomainID" -> "Contact"];
SMTAddMesh[Polygon[{{-1, -1}, {1, -1}, {1, 0}, {-1, 0}}], "Solid2", "Division" -> {5, 5},
 "BodyID" → "B2", "BoundaryDomainID" -> "Contact"];
SMTAddEssentialBoundary[{"Y" == -1 &, 1 -> 0, 2 -> 0}, {"Y" == 0.6 &, 1 -> 0, 2 -> -1}];
SMTAnalysis["Output" -> "tmp1.out"];
SMTShowMesh["BoundaryConditions" → True, "Marks" → {"ElementNumber"}]
```



Example 2:

Here a heat transfer boundary condition elements are created on the surface of the domain. Although the solid is discretized with quadratic elements, the surface can only be discretized with first order elements.

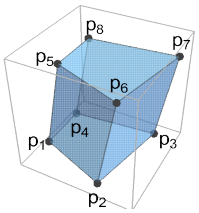
In[591]:=
```
SMTInputData[];
L = 100; H = 20; nx = 3; ny = 3;
points = {{0, 0}, {L, 0}, {L / 2, 2 H}, {0, H}};
SMTAddDomain["A", {"ML:", "TH", "D2", "Q2S", "DF", "ST", "Q2S", "T", "Fourier"}, {}];
SMTAddDomain["B", {"ML:", "TH", "S2", "L1", "DF", "FL", "L1", "T", "Const"}, {}];
SMTAddMesh[Polygon[points], "A",
  "Division" -> {nx, ny}, "BoundaryDomainID" → "B", "BodyID" → "B1"];
SMTAnalysis[];
SMTShowMesh["Marks" → True]
```



Basic mesh generation

SMTAddMesh[*geoObject*, *dID*]
Adds unstructured mesh defined by given geometric object *geoObject* and domain identification *dID*. Topology of the mesh is implicitly defined by the domain identification *dID*.

SMTAddMesh[*geoObject*, *dID*, "Division"->*div*]
Adds structured mesh defined by given geometric object *geoObject*, domain identification *dID* and mesh division *div*. Topology of the mesh is implicitly defined by the domain identification *dID*.

geoObject	description
Polygon[{p_1,p_2,p_3,p_4,...}]	polygon in 2 D or 3 D
Polygon[{{p_{11},p_{12},p_{13},...},{p_{21},p_{22},...}},...]	collection of polygons in 2 D or 3 D
Line[{p_1,p_2,...,p_m}]	mesh of $m-1$ 2 D or 3 D line elements
Line[{{p_{11},$p1_2$,...},{p_{21},p_{22},...}},...]	collection of lines elements
Hexahedron[{p_1,p_2,p_3,p_4,p_5,p_6,p_7,p_8}]	structured mesh for an arbitrary hexahedron region ()
Raster[*raster*]	structured mesh for an arbitrary shaped 2 D region with four curved edges (curved quadrilateral)
Raster3D[*raster*]	structured mesh for 3 D region defined by six arbitrary shaped surfaces (curved hexahedron)

Possible geometric objects.

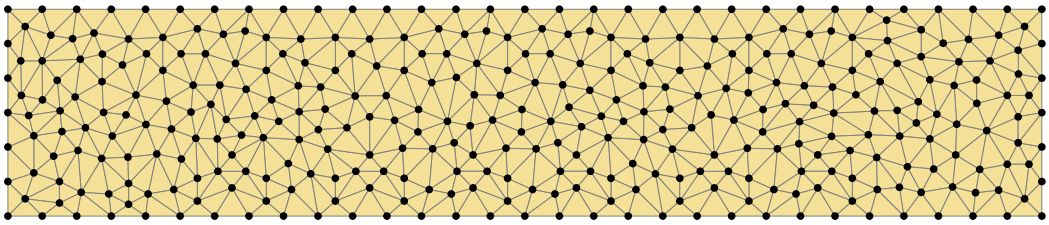
option	default	description
"BodyID"	None	body identification string
"BoundaryDomainID"	{}	domain identification (one or more) for the elements additionally generated on the outer surface of body generated by SMTAddMesh
"Division"	None	"Division" is relevant only for structured meshes: n_x ⇒ number of elements along the local x for line elements $\{n_x,n_y\}$ ⇒ number of elements along the local x and y axis for 2 D elements in 2 D or 3 D $\{n_x,n_y,n_z\}$ ⇒ number of elements along the local x, y and z axis for solid elements
all ToElementMesh options		for unstructured meshes all options that can be given to ToElementMesh command (e.g. "MaxCellMeasure", "MeshOrder",...)

General options of the SMTAddMesh function.

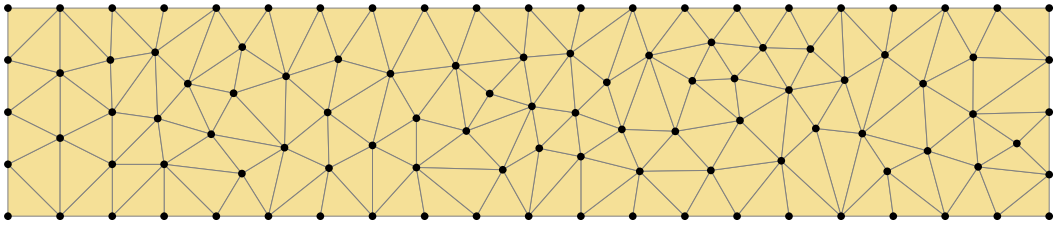
Examples of 2D meshes

In[313]:=
```
<< AceFEM`;
L = 100; H = 20; F = 1000; q = 10;
polygon = Polygon[{{0, 0}, {L, 0}, {L, H}, {0, H}}];
```

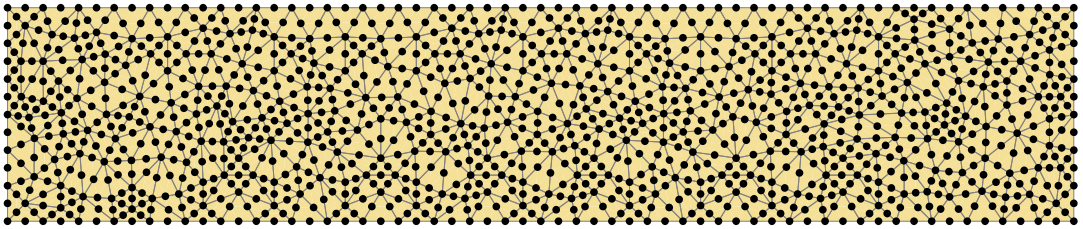
In[316]:=
```
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "PS", "T1", "DF", "HY", "T1", "D", {{"NeoHooke", "WA"}}}, {}];
SMTAddMesh[polygon, "A"];
SMTAnalysis[];
SMTShowMesh["NodeMarks" → True]
```



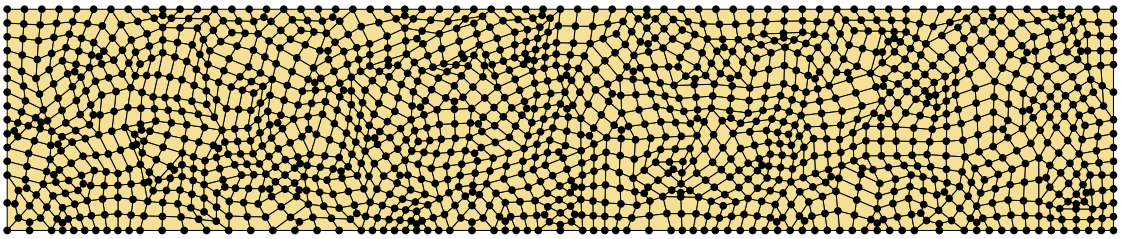
In[341]:=
```
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "PS", "T1", "DF", "HY", "T1", "D", {{"NeoHooke", "WA"}}}, {}];
SMTAddMesh[polygon, "A", "MaxCellMeasure" → 20];
SMTAnalysis[];
SMTShowMesh["NodeMarks" → True]
```



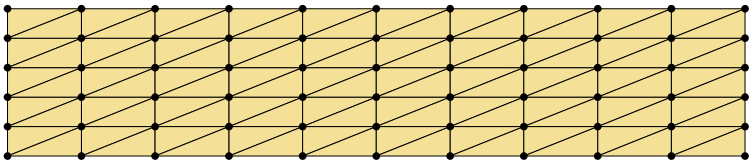
In[321]:=
```
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "PS", "T2", "DF", "HY", "T2", "D", {{"NeoHooke", "WA"}}}, {}];
SMTAddMesh[polygon, "A"];
SMTAnalysis[];
SMTShowMesh["NodeMarks" → True]
```



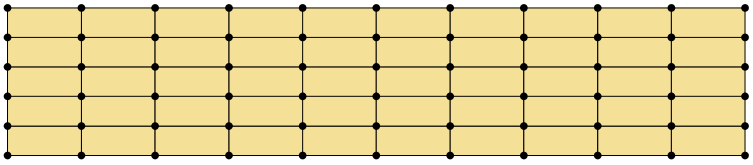
In[42]:=
```
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "PS", "Q1", "DF", "HY", "Q1", "D", {{"NeoHooke", "WA"}}}, {}];
SMTAddMesh[polygon, "A"];
SMTAnalysis[];
SMTShowMesh["NodeMarks" → True]
```



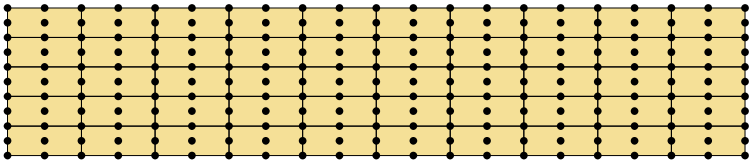
In[52]:=
```
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "PS", "T1", "DF", "HY", "T1", "D", {{"NeoHooke", "WA"}}}, {}];
SMTAddMesh[polygon, "A", "Division" → {10, 5}];
SMTAnalysis[];
SMTShowMesh["NodeMarks" → True]
```



In[57]:=
```
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "PS", "Q1", "DF", "HY", "Q1", "D", {{"NeoHooke", "WA"}}}, {}];
SMTAddMesh[polygon, "A", "Division" → {10, 5}];
SMTAnalysis[];
SMTShowMesh["NodeMarks" → True]
```



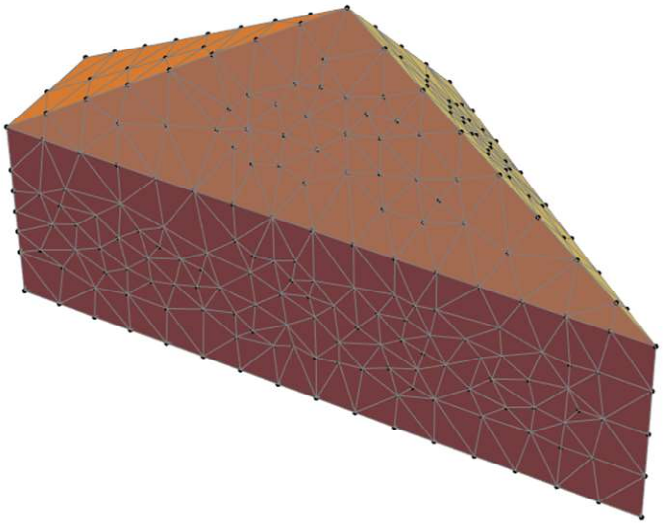
In[47]:=
```
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "PS", "Q2", "DF", "HY", "Q2", "D", {{"NeoHooke", "WA"}}}, {}];
SMTAddMesh[polygon, "A", "Division" → {10, 5}];
SMTAnalysis[];
SMTShowMesh["NodeMarks" → True]
```



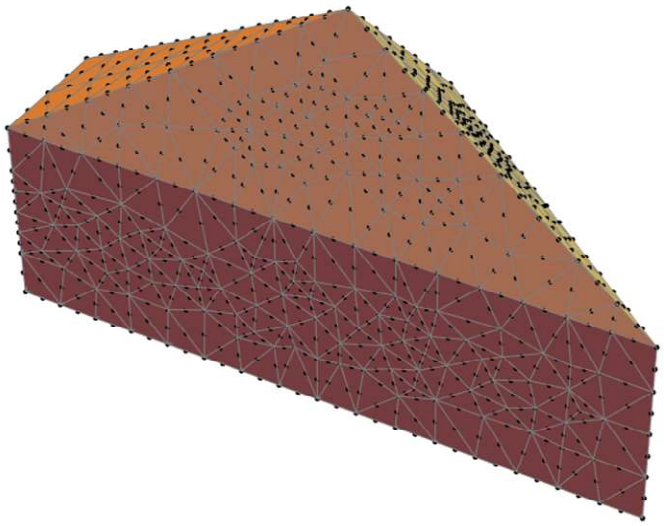
Examples of 3D solid meshes

In[1]:=
```
L = 100; H = 20; V = 30; nx = 5; ny = 3; nz = 4;
points = {{0, 0, 0}, {L, 0, 0}, {L / 2, 2 H, 0}, {0, H, 0},
    {0, 0, V}, {L, 0, V}, {L / 2, 2 H, V}, {0, H, V}, {L / 2, H / 2, 2 V}};
body = Polygon[{points[[{1, 2, 6, 5}]], points[[{1, 2, 3, 4}]], points[[{1, 4, 8, 5}]],
     points[[{4, 3, 7, 8}]], points[[{2, 6, 7, 3}]], points[[{5, 6, 9}]],
     points[[{6, 7, 9}]], points[[{7, 8, 9}]], points[[{8, 5, 9}]]}];
```

In[4]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "D3", "O1", "DF", "LE", "O1", "D", "Hooke"}, {}];
SMTAddMesh[body, "A"];
SMTAnalysis[];
SMTShowMesh["NodeMarks" → True]
```



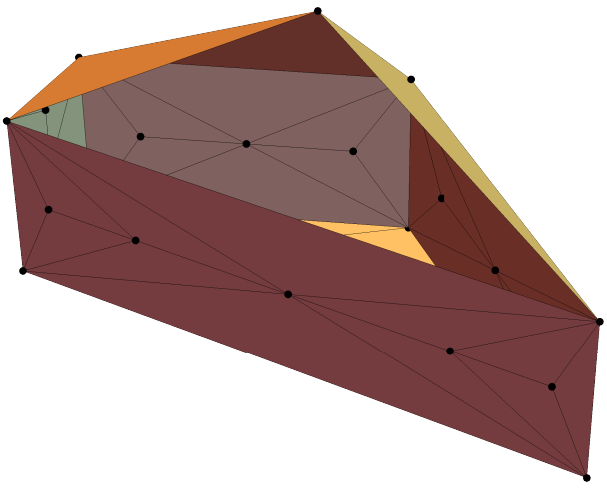
In[10]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "D3", "O2", "DF", "LE", "O2", "D", "Hooke"}, {}];
SMTAddMesh[body, "A"];
SMTAnalysis[];
SMTShowMesh["NodeMarks" → True]
```



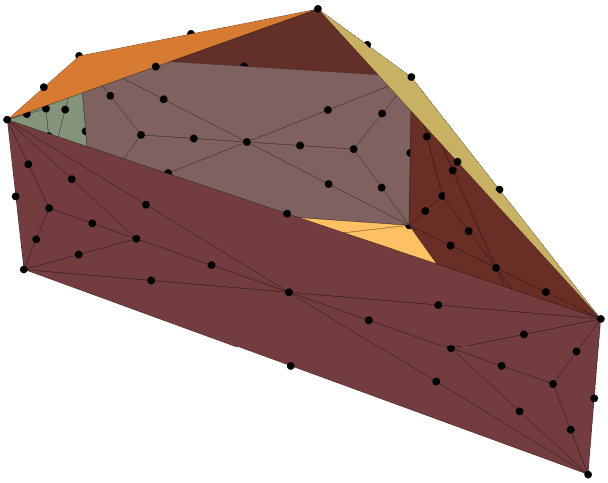
Examples of 3D shell meshes

In[217]:=
```
shellUnstructured = Polygon[{points[[{1, 2, 6, 5}]], points[[{1, 2, 3, 4}]],
    points[[{1, 4, 8, 5}]], points[[{4, 3, 7, 8}]], points[[{2, 6, 7, 3}]],
    points[[{6, 7, 9}]], points[[{7, 8, 9}]], points[[{8, 5, 9}]]}];
```

In[218]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "MS", "P1", "DF", "HY", "P1P6", {"D", "Fi"}, "SVenant"}, {}];
SMTAddMesh[shellUnstructured, "A"];
SMTAnalysis[];
SMTShowMesh["NodeMarks" → True]
```

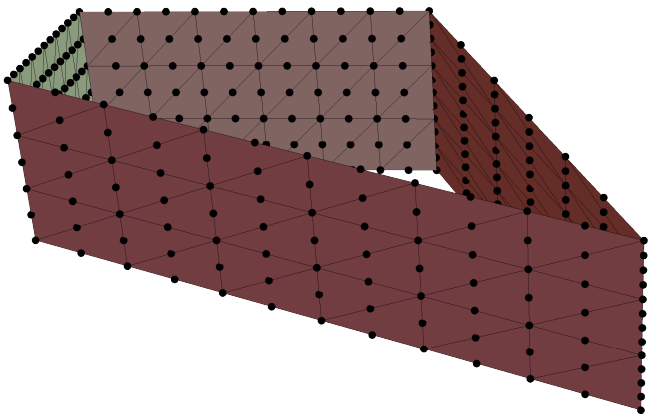


In[224]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "MS", "P2", "DF", "HY", "P2P6", {"D", "Fi"}, "SVenant"}, {}];
SMTAddMesh[shellUnstructured, "A"];
SMTAnalysis[];
SMTShowMesh["NodeMarks" → True]
```

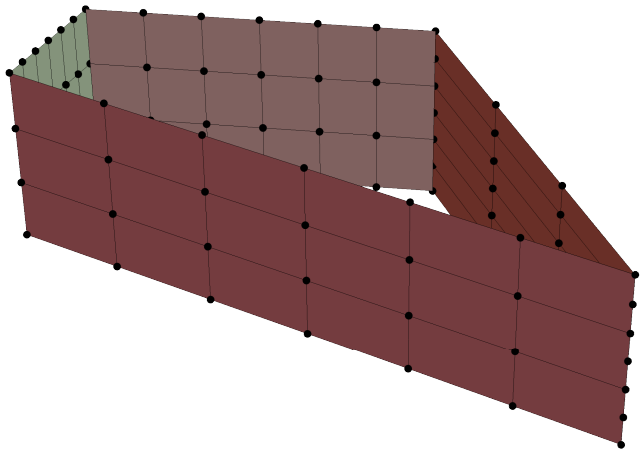


In[230]:=
```
shellStructured = Polygon[{points[[{1, 2, 6, 5}]],
    points[[{1, 4, 8, 5}]], points[[{2, 6, 7, 3}]], points[[{4, 3, 7, 8}]]}];
```

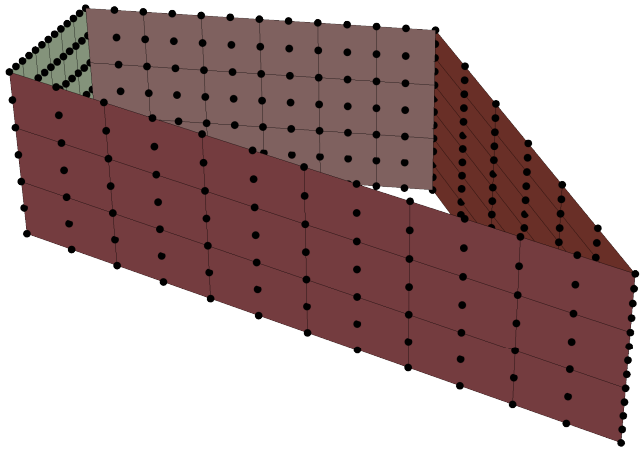
In[231]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "MS", "P2", "DF", "HY", "P2P6", {"D", "Fi"}, "SVenant"}, {}];
SMTAddMesh[shellStructured, "A", "Division" → {6, 3}];
SMTAnalysis[];
SMTShowMesh["NodeMarks" → True]
```



In[237]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "MS", "S1", "DF", "HY", "S1P6", {"D", "Fi"}, "SVenant"}, {}];
SMTAddMesh[shellStructured, "A", "Division" → {6, 3}];
SMTAnalysis[];
SMTShowMesh["NodeMarks" → True]
```



In[243]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "MS", "S2", "DF", "HY", "S2P6", {"D", "Fi"}, "SVenant"}, {}];
SMTAddMesh[shellStructured, "A", "Division" → {6, 3}];
SMTAnalysis[];
SMTShowMesh["NodeMarks" → True]
```



Advanced SMTAddMesh

Importing `ElementMesh` meshes

SMTAddMesh[*elementMesh*, *dID*]

Adds mesh defined by `ElementMesh` data object *elementMesh* generated by built-in *Mathematica* mesher (see `Element Mesh Generation`, ToElementMesh) and domain identification *dID* that defines the actual element type. Mesh must not be divided into sub-regions and have to be composed of only one type of mesh elements. The actual element type must correspond to the mesh element type.

SMTAddMesh[*elementMesh*, {*meshElementType*$_1$→*dID*$_1$,*meshElementType*$_2$→*dID*$_2$,…}]
Adds mesh defined by `ElementMesh` data object *elementMesh* generated by built-in *Mathematica* mesher (see `Element Mesh Generation`). Mesh must not be divided into sub-regions, but it can be composed of different types of mesh elements. Second argument defines the mapping between the mesh elements type (*meshElementType*) generated by mesher (see `Mixed Element Types`) and the actual element type defined by the domain identification *dID*.

SMTAddMesh[*elementMesh*,{{*meshElementType*$_1$,*regionMarker*$_1$}→
dID$_1$,{*meshElementType*$_1$,*regionMarker*$_2$}→dID$_2$,…,{*meshElementType*$_n$,*regionMarker*$_n$}→*dID*$_n$}]
Adds an arbitrary mesh defined by `ElementMesh` data object *elementMesh* generated by built-in *Mathematica* mesher (see ToElementMesh , `Mixed Element Types`).

option	default	description
"MeshElements"	True	True ⇒ import mesh elements False ⇒ skip mesh elements
"BoundaryElements"	Automatic	True ⇒ import boundary elements Automatic ⇒ import boundary elements only if mesh elements are not defined False ⇒ skip boundary elements
"PointElements"	False	True ⇒ import point elements False ⇒ skip point elements

Generation of an arbitrary unstructured mesh.

Importing MeshRegion meshes

SMTAddMesh[*meshRegion*,*dID*]
Adds an arbitrary mesh defined by MeshRegion data object generated by Mathematica (see `Mesh-Based Geometric Regions`) . Generated elements can have an arbitrary number of nodes in 2D and 3D depending on MeshRegion data object as described in a table below. Generated elements are associated with domain identification *dID*.

`Point` [i]	`0`	`point`
`Line` [{i_1,i_2,…}]	`1`	`line segments` {i_1,i_2}, {i_2,i_3}, …
`Triangle` [{i_1,i_2,i_3}]	`2`	`filled triangle`
`Polygon` [{i_1,i_2,…}]	`2`	`filled polygon`
`Tetrahedron` [{i_1,…,i_4}]	`3`	`filled tetrahedron`
`Hexahedron` [{i_1,…,i_8}]	`3`	`filled hexahedron`
`Pyramid` [{i_1,…,i_5}]	`3`	`filled pyramid`
`Prism` [{i_1,…,i_6}]	`3`	`filled prism`
`Simplex` [{i_1,…,i_k}]	`0, 1, 2, 3`	`filled simplex`

SMTAddMesh[*meshRegion*, {*cellType*$_1$→*dID*$_1$,*cellType*$_2$→*dID*$_2$,…}]
Adds an arbitrary mesh defined by MeshRegion data object generated by Mathematica (see `Mesh-Based Geometric Regions`), where *cellType* can be Point, Line,Triangle, Polygon,Tetrahedron,Hexahedron,Pyramid or Prism (see MeshRegion). Generated elements of different types are associated with different domain identifications.

Structured meshes

SMTAddMesh[Polygon[{p_1,p_2,p_3,p_4}], *dID, meshType,* {n$_x$,n$_x$}]
Generates structured mesh for an arbitrary quadrilateral region where {n$_x$,n$_x$} is the number of elements along local x and y axis. The domain identification *dID* defines the actual element type and the *meshType* mesh type
(`Types of Structured Meshes`).

SMTAddMesh[Hexahedron[{p_1,p_2,p_3,p_4,p_5,p_6,p_7,p_8}], *dID, meshType,* {n$_x$,n$_x$,n$_z$}]

Generates structured mesh for an arbitrary hexahedron region where $\{n_x,n_x,n_z\}$ is the number of elements along local x and y and z axis. The domain identification *dID* defines the actual element type and the *meshType* the mesh type (`Types of Structured Meshes`).

SMTAddMesh[Line[$\{p_1,p_2,\ldots,p_m\}$], *dID, meshType*]
Generates mesh of *m-1* 2D or 3D line elements with given coordinates of the nodes $\{p_1,p_2,\ldots,p_m\}$. The domain identification *dID* defines the actual element type and the *meshType* mesh type (see `1D structured mesh types`).

SMTAddMesh[Line[$\{p_1,p_2,\ldots,p_m\}$], *dID, meshType, n*]
Generates mesh of *n* 2D or 3D line elements. The coordinates of the nodes of the actual mesh are obtained by interpolation of given points Line[$\{p_1,p_2,\ldots,p_m\}$]. The order of interpolation is defined by the "InterpolationOrder" option with the default value 3. The domain identification *dID* defines the actual element type and the *meshType* mesh type (see `1D structured mesh types`).

Generation of structured mesh.

SMTAddMesh[Raster[*raster*], *dID, meshType,* $\{n_x,n_x\}$]
Generates structured mesh for an arbitrary shaped 2D region with four curved edges (curved quadrilateral) where $\{n_x,n_x\}$ is the number of elements along local x and y axis. The region is defined by the two-dimensional array of points or *raster*. The domain identification *dID* defines the actual element type and the *meshType* the mesh type (`Types of Structured Meshes`).

SMTAddMesh[Raster3D[*raster*], *dID, meshType,* $\{n_x,n_x,n_z\}$]
Generates structured mesh for 3D region defined by six arbitrary shaped surfaces (curved hexahedron) where $\{n_x,n_x,n_z\}$ is the number of elements along local x and y and z axis.The region is defined by the three-dimensional array of points or *raster*. The domain identification *dID* defines the actual element type and the *meshType* the mesh type (`Types of Structured Meshes`).

option	default	description
"InterpolationOrder"	3	The degree of raster interpolation is specified by the option "InterpolationOrder". By default the third-order (or less if the number of points in one direction is less than 4) polynomial interpolations of coordinates X,Y,Z is used. **Option applies only to structured meshes defined by a raster of points.**

Generation of an structured mesh defined by raster of points.

Importing meshes generated by an arbitrary external meshers

SMTAddMesh[$\{\{X_1,Y_1,Z_1\},\{X_2,Y_2,Z_2\},\ldots\}$, $\{dID_1\text{->}\{\{n_1^1,n_2^1,\ldots\},\{n_1^2,n_2^2,\ldots\},\ldots\}$, $dID_2\text{→}$ $\{\{n_1^1,n_2^1,\ldots\},\{n_1^2,n_2^2,\ldots\},\ldots\}$, …}]
Imports mesh defined by the list of node coordinates $\{\{X_1,Y_1,Z_1\},\{X_2,Y_2,Z_2\},\ldots\}$ and a list of regions defined by domain identification dID_i and connectivity table, $\{\{n_1^1,n_2^1,\ldots\},\{n_1^2,n_2^2,\ldots\},\ldots\}$. This form can be used to import mesh generated by an arbitrary external mesher. Node numbers in connectivity table refer to the successive indexes of the nodes given.

SMTAddMesh[$\{dID_1\text{->}\{\{n_1^1,n_2^1,\ldots\},\{n_1^2,n_2^2,\ldots\},\ldots\}$, $dID_2\text{->}\{\{n_1^1,n_2^1,\ldots\},\{n_1^2,n_2^2,\ldots\},\ldots\}$, …}]
Imports mesh defined by a list of regions defined by domain identification dID_i and connectivity table, $\{\{n_1^1,n_2^1,\ldots\},\{n_1^2,n_2^2,\ldots\},\ldots\}$. Node numbers in connectivity table refer to the global node numbers. This form can be used to import several sub-meshes imposed on the existing mesh of spatial nodal points.

SMTAddMesh[$\{gn_1,gn_2,\ldots\}$, $\{dID_1\text{->}\{\{n_1^1,n_2^1,\ldots\},\{n_1^2,n_2^2,\ldots\},\ldots\}$, $dID_2\text{->}\{\{n_1^1,n_2^1,\ldots\},\{n_1^2,n_2^2,\ldots\},\ldots\}$, …}]
Imports mesh defined by a list of regions defined by domain identification dID_i and connectivity table, $\{\{n_1^1,n_2^1,\ldots\},\{n_1^2,n_2^2,\ldots\},\ldots\}$. Node numbers in connectivity table refer to the successive elements in the list of global node numbers $\{gn_1,gn_1,\ldots\}$ where gn_i is a global node number of the *i*-th local node. This form can be used to import several sub-meshes imposed on the existing mesh of spatial nodal points where each sub-mesh has its own node numbering.

Importing meshes generated by an arbitrary external meshers into AceFEM.

Function SMTAddMesh returns the range of node indexes added and the range of element indexes ({{nodMin,nodeMax},{elementMin,-elementMax}}). If global node numbering is used it returns only the range of element indexes ({Null,{elementMin,elementMax}}).

Obsolete SMTMesh command has been incorporated into a more general SMTAddMesh command.

Unstructured Meshes

Simple unstructured meshes

SMTAddMesh[*elementMesh*, *dID*]
Adds mesh defined by `ElementMesh` data object generated by `ToElementMesh` or ToBoundaryMesh mesher (see `Mixed Element Types`) and domain identification *dID* that defines the actual element type. Mesh must not be divided into sub-regions and have to be composed of only one type of mesh elements.

See the `ToElementMesh` function documentation for the instructions how to generate mesh with *Mathematica* and `Element Mesh Generation` documentation for the general description of the *Mathematica's* meshing capabilities.

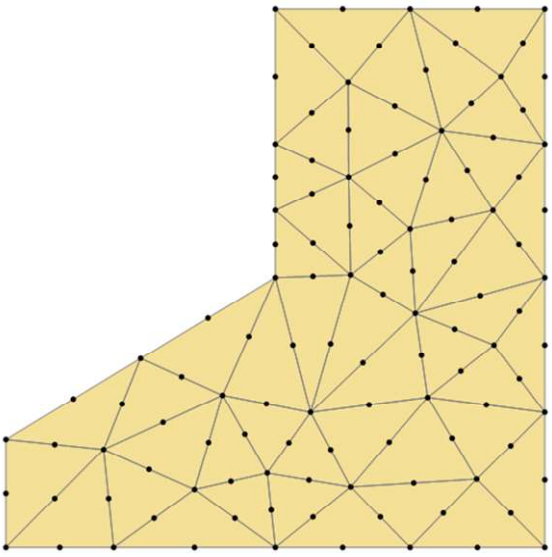
The actual element type must directly correspond to the mesh element type. The `ToElementMesh` mesher generates unstructured meshes composed of first order (topology T1) and second order (topology T2) `TriangleElement`, first order (topology Q1) and second order (topology Q2S) `QuadElement`, first order (topology O1) and second order (topology O2) `TetrahedronElement` and first order (topology H1) and second order (topology H2S) `HexahedronElement`. It is the user responsibility to choose an appropriate elements (see `Mixed Element Types`).

Arbitrary 2D polygon

In[78]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[{"Ω",
   {"ML:", "SE", "PE", "T2", "DF", "LE", "T2", "D", "Hooke"}, {"E *" -> 1000, "ν *" -> 0.3}}];
points = {{0, 0}, {10, 0}, {10, 10}, {5, 10}, {5, 5}, {0, 2}};
```

- Create a mesh made of quadratic triangles with the maximal area 2 (MaxCellMeasure).

In[82]:=
```
mesh = ToElementMesh[Polygon[points], MaxCellMeasure → 2];
SMTAddMesh[mesh, "Ω"];
SMTAnalysis[];
SMTShowMesh["NodeMarks" → True]
```



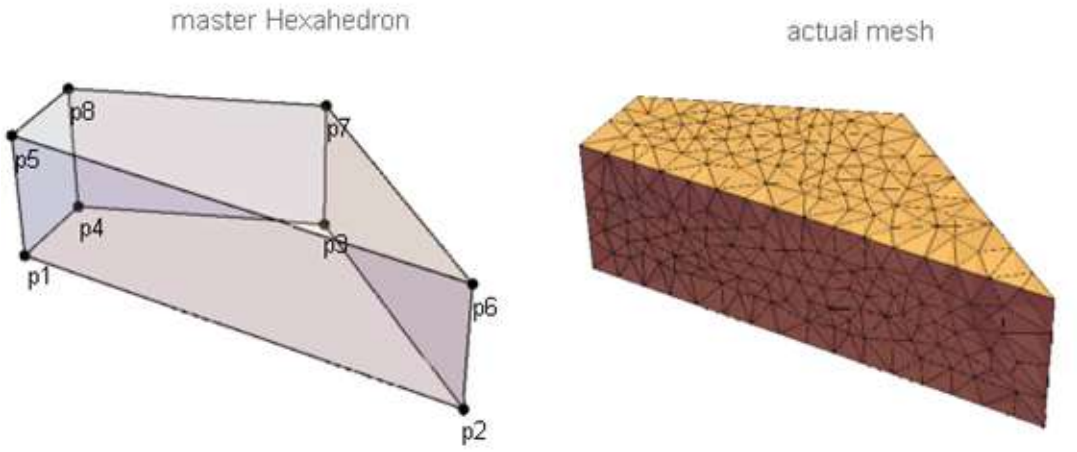
3D Hexahedron

In[58]:=
```
<< AceFEM`;
SMTInputData[];
L = 100; H = 20; V = 30; nx = 5; ny = 3; nz = 4;
points =
  {{0, 0, 0}, {L, 0, 0}, {L / 2, 2 H, 0}, {0, H, 0}, {0, 0, V}, {L, 0, V}, {L / 2, 2 H, V}, {0, H, V}};
SMTAddDomain["A", {"ML:", "SE", "D3", "O1", "DF", "LE", "O1", "D", "Hooke"}, {}];
```

- Create a mesh made of linear tetrahedral elements with the maximal volume 600 (MaxCellMeasure).

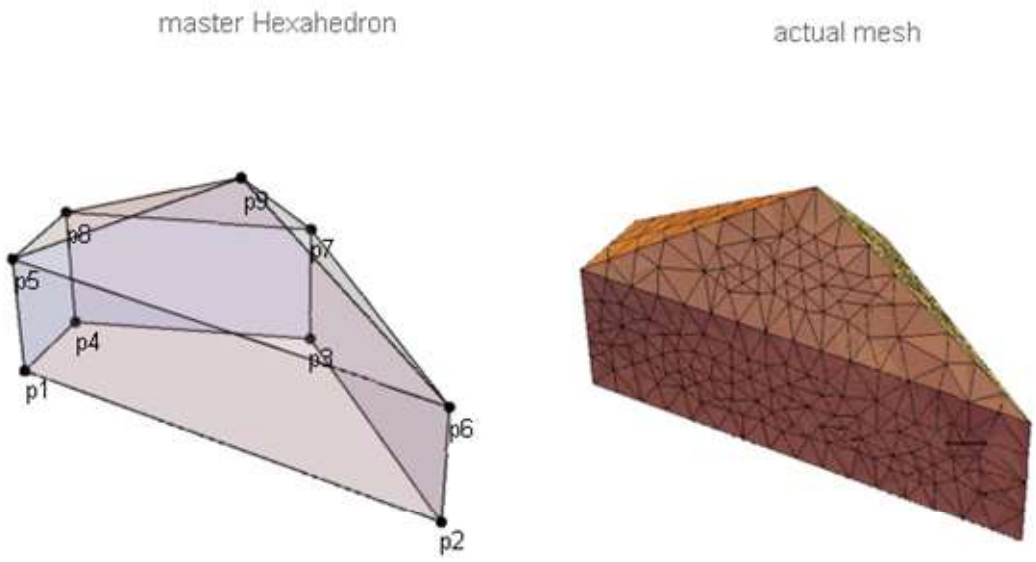
In[63]:=
```
mesh = ToElementMesh[Hexahedron[points], "MeshOrder" → 1, MaxCellMeasure → {"Volume" → 30}];
SMTAddMesh[mesh, {"O1" → "A"}];
SMTAnalysis[];
```

In[66]:=
```
GraphicsRow[
 {Graphics3D[{LightBlue, {Opacity[0.2], Hexahedron[points]}, Black, PointSize[0.02], Point[
     points], Black, MapIndexed[Text["p" <> ToString[#2[[1]]], #1 + 2 {1, 1, -3}] &, points]},
   PlotLabel → "master Hexahedron", Boxed → False],
  SMTShowMesh["Label" → "actual mesh", "Opacity" -> 0.9]}, -10, ImageSize → 500]
```



Arbitrary 3D Polyhedron

In[11]:=
```
<< AceFEM`;
SMTInputData[];
L = 100; H = 20; V = 30; nx = 5; ny = 3; nz = 4;
points = {{0, 0, 0}, {L, 0, 0}, {L / 2, 2 H, 0}, {0, H, 0},
    {0, 0, V}, {L, 0, V}, {L / 2, 2 H, V}, {0, H, V}, {L / 2, H / 2, 2 V}};
body = Polygon[{points[[{1, 2, 6, 5}]], points[[{1, 2, 3, 4}]], points[[{1, 4, 8, 5}]],
     points[[{4, 3, 7, 8}]], points[[{2, 6, 7, 3}]], points[[{5, 6, 9}]],
     points[[{6, 7, 9}]], points[[{7, 8, 9}]], points[[{8, 5, 9}]]}];
SMTAddDomain["A", {"ML:", "SE", "D3", "O1", "DF", "LE", "O1", "D", "Hooke"}, {}];
mesh = ToElementMesh[body, "MeshOrder" → 1, MaxCellMeasure → {"Volume" → 30}];
SMTAddMesh[mesh, {"O1" → "A"}];
SMTAnalysis[];
GraphicsRow[
 {Graphics3D[{LightBlue, {Opacity[0.2], body}, Black, PointSize[0.02], Point[points],
    Black, MapIndexed[Text["p" <> ToString[#2[[1]]], #1 + 2 {1, 1, -3}] &, points]},
   PlotLabel → "master Hexahedron", Boxed → False],
  SMTShowMesh["Label" → "actual mesh", "Opacity" -> 0.9]}, -10, ImageSize → 500]
```



Cylindrical shell

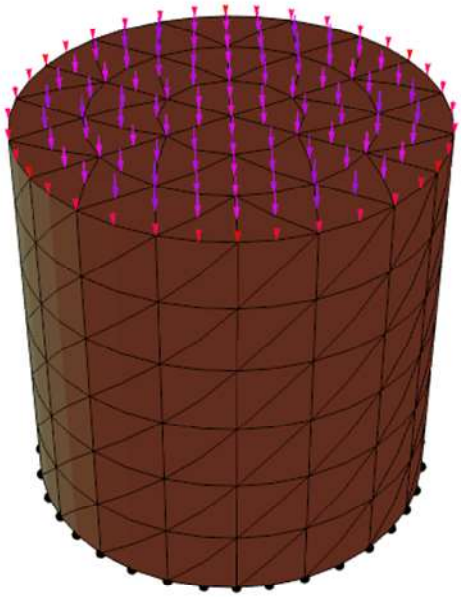
In[304]:=
```
<< AceFEM`;
R = 300; H = 600; t = 1;
SMTInputData[];
SMTAddDomain["shell",
  {"ML:", "SE", "MS", "P2", "DF", "HY", "P2P6", {"D", "Fi"}, "SVenant"}, {"t *" → t}];
```

- Create a mesh made of quadratic shell elements with the maximal area "total surface"/10.

In[308]:=
```
surface = 2 π R H;
mesh = ToBoundaryMesh[Cylinder[{{0, 0, 0}, {0, 0, H}}, R],
  "MeshOrder" → 2, MaxCellMeasure → ("Area" -> surface / 10)];
SMTAddMesh[mesh, "shell"];
```

- Cylinder is clamped at the bottom and subjected to the surface load at the top.

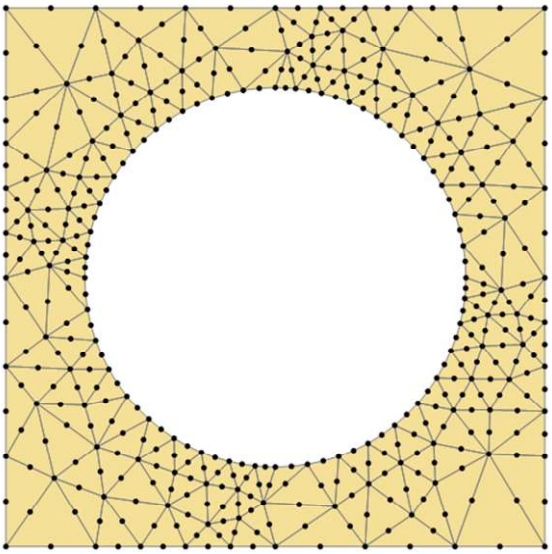
In[310]:=
```
SMTAddEssentialBoundary["Z" == 0 && "ID" == "D" &, 1 → 0, 2 → 0, 3 → 0];
SMTAddNaturalBoundary["Z" == H && "ID" == "D" &, 3 -> Polygon[{-0.01}]];
SMTAnalysis[];
SMTShowMesh["BoundaryConditions" → True]
```



2D region with circular hole

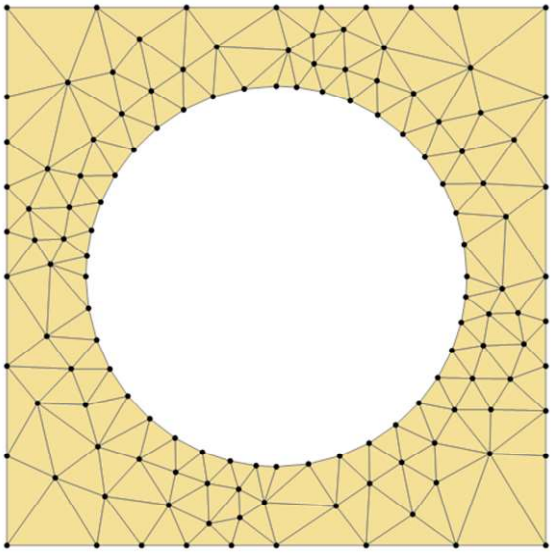
- Here presented is an example of rectangular region with the hole. The element code {"ML:","SE","PE","T2",-"DF","LE","T2","D","Hooke"} her defines the actual element type as six nodded triangle which corresponds with the default value of the option "MeshOrder"→2 of the ToElementMesh command.

In[85]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[{"Ω",
   {"ML:", "SE", "PE", "T2", "DF", "LE", "T2", "D", "Hooke"}, {"E *" -> 1000, "ν *" -> 0.3}}];
mesh = ToElementMesh[ImplicitRegion[x^2 + y^2 > 0.5, {x, y}],
   {{-1, 1}, {-1, 1}}, MaxCellMeasure → 1];
SMTAddMesh[mesh, "Ω"];
SMTAnalysis[];
SMTShowMesh["NodeMarks" → True]
```



- The element code "ML:SEPET1DFLET1DHooke" here defines the actual element type as three nodded triangle. In order to create corresponding mesh an option "MeshOrder"→1 has to be specified, since, by default, the 6 nodded triangles are generated by the ToElementMesh command.

In[91]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[{"Ω",
   {"ML:", "SE", "PE", "T1", "DF", "LE", "T1", "D", "Hooke"}, {"E *" -> 1000, "ν *" -> 0.3}}];
mesh = ToElementMesh[ImplicitRegion[x^2 + y^2 > 0.5, {x, y}],
   {{-1, 1}, {-1, 1}}, "MeshOrder" → 1, MaxCellMeasure → 1];
SMTAddMesh[mesh, "Ω"];
SMTAnalysis[];
SMTShowMesh["NodeMarks" → True]
```

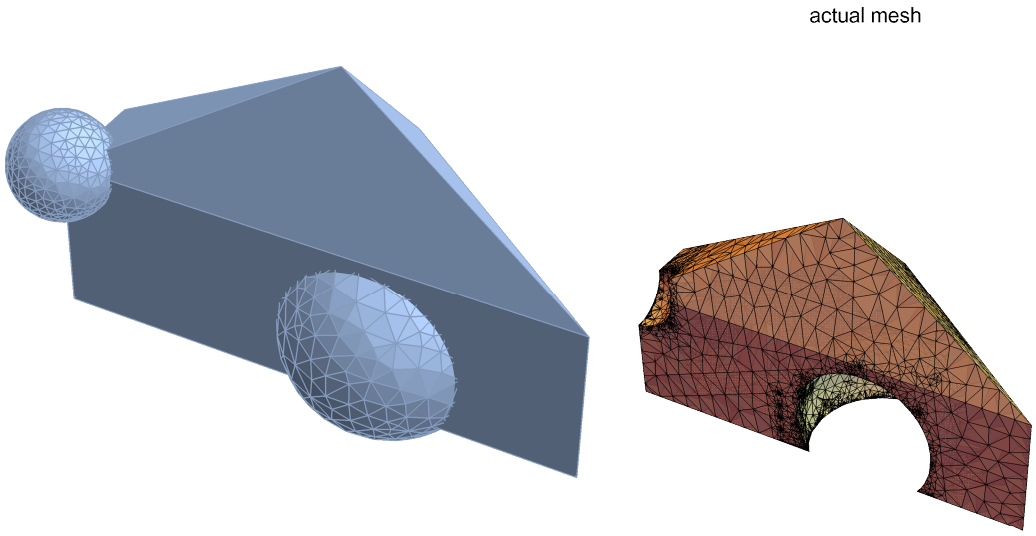


Arbitrary 3D Regions

- The Wolfram Language provides several ways of deriving new regions from existing ones, including combining them through Boolean operations and transforming them through a mapping. Here voids are inserted randomly into solid body.

In[78]:=
```
<< AceFEM`;
SMTInputData[];
L = 100; H = 20; V = 30; nx = 5; ny = 3; nz = 4;
points = {{0, 0, 0}, {L, 0, 0}, {L / 2, 2 H, 0}, {0, H, 0},
   {0, 0, V}, {L, 0, V}, {L / 2, 2 H, V}, {0, H, V}, {L / 2, H / 2, 2 V}};
balls = {{{0, 0, 30}, 10}, {{50, 10, 10}, 10}, {{60, 10, 10}, 20}};
SMTAddDomain["A", {"ML:", "SE", "D3", "O1", "DF", "LE", "O1", "D", "Hooke"}, {}];
body = BoundaryMeshRegion[points, Polygon[{{1, 2, 6, 5}, {1, 2, 3, 4}, {1, 4, 8, 5},
     {4, 3, 7, 8}, {2, 6, 7, 3}, {5, 6, 9}, {6, 7, 9}, {7, 8, 9}, {8, 5, 9}}]];
ballsmesh = Map[BoundaryDiscretizeRegion[Ball[#1[[1]], #1[[2]]],
    MaxCellMeasure → {"Length" → 10}, Method → "MarchingCubes"] &, balls];
regionholes = Pick[balls, Map[RegionWithin[body, #] &, ballsmesh]];
region = Fold[RegionDifference, body, ballsmesh];
mesh = ToElementMesh[region, "MeshOrder" → 1, "MaxCellMeasure" → {"Length" → 10},
   "MaxBoundaryCellMeasure" → {"Length" → 10}, "BoundaryMeshGenerator" → "RegionPlot",
   "ImproveBoundaryPosition" → False, "MeshElementType" → TetrahedronElement,
   "MeshOrder" → 1, "RegionHoles" -> regionholes[[All, 1]]];
SMTAddMesh[mesh, {"O1" → "A"}];
SMTAnalysis[];
```

In[91]:=
```
GraphicsRow[{Show[{body, ballsmesh}],
  SMTShowMesh["Label" → "actual mesh", "Opacity" -> 0.9]}, -10, ImageSize → 600]
```



Element Meshes with Subregions

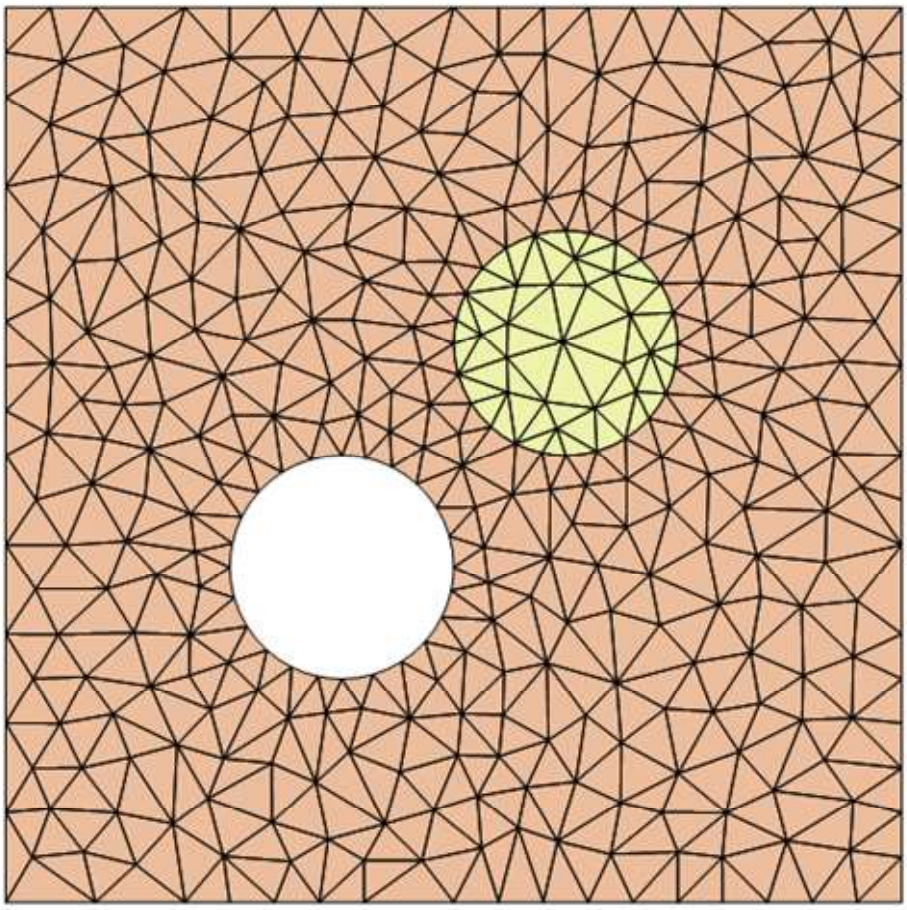
SMTAddMesh[*elementMesh*,{{*meshElementType*$_1$,*regionMarker*$_1$}→dID$_1$,{*meshElementType*$_1$,*regionMarker*$_2$}→dID$_2$,...,{*meshElementType*$_n$,*regionMarker*$_n$}→dID$_n$}]
Adds mesh divided into sub-regions defined by successive region markers *regionMarker*$_i$.

Generation of an arbitrary unstructured meshes composed of sub-regions.

2D region with circular hole and subregions

In[62]:=
```
Ω = ImplicitRegion[(x - 1 / 2) ^ 2 + (y - 1 / 2) ^ 2 ≥ (1 / 2) ^ 2 &&
     (x + 1 / 2) ^ 2 + (y + 1 / 2) ^ 2 ≥ (1 / 2) ^ 2, {{x, -2, 2}, {y, -2, 2}}];
mesh = ToElementMesh[Ω, "RegionHoles" → {{-1 / 2, -1 / 2}},
   "RegionMarker" → {{{1 / 2, 1 / 2}, 2}, {{1 / 2, -1 / 2}, 1}}, "MaxBoundaryCellMeasure" → 1];
```

In[64]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[
  {"T2 matrix",
   {"ML:", "SE", "PE", "T2", "DF", "JC", "T2", "D", {{"NeoHooke", "WA"}, {"Mises", "ExH"}}},
   {"E *" -> 1000, "ν *" -> 0.3, "σy *" → 10, "Kh *" → 100}},
  {"T2 inclusion", {"ML:", "SE", "PE", "T2", "DF", "JC", "T2", "D", {{"NeoHooke", "WA"},
     {"Mises", "ExH"}}}, {"E *" -> 100, "ν *" -> 0.3, "σy *" → 10, "Kh *" → 10}}
 ];
SMTAddMesh[mesh, {{"T2", 1} -> "T2 matrix", {"T2", 2} → "T2 inclusion"}];
SMTAnalysis[];
SMTShowMesh[]
```



Mixed Element Types

In most cases the `ToElementMesh` generated mesh can be composed of different types of elements. In that case the mapping between the mesh element type generated by the mesher (`Element Mesh Generation`) and the actual element type (`Element Topology`) defined by domain identification *dID* has to be provided explicitly as follows.

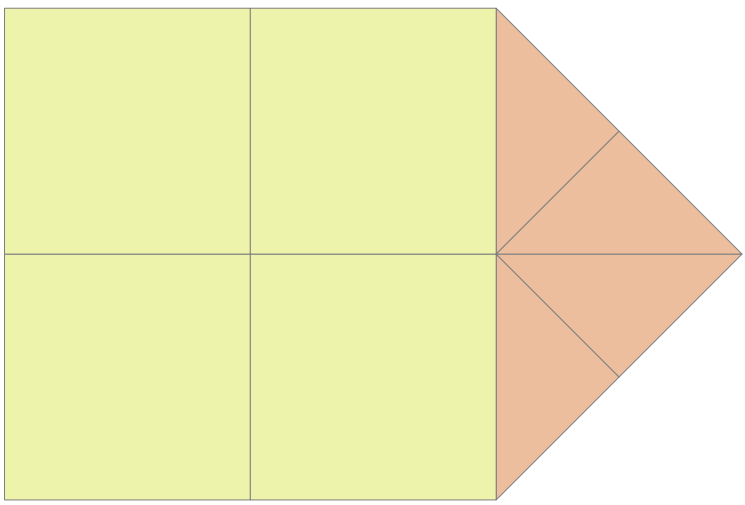
SMTAddMesh[*mesh_ElementMesh*, {$meshElementType_1 \to dID_1$,$meshElementType_2 \to dID_2$,...}]
If the mesh is composed of different types of elements then the second argument defines the mapping between the mesh element type of elements generated by mesher and the actual element type defined by domain identification *dID*.

meshElementType	"MeshOrder" option	Actual mesh type
"T2"	Automatic	"T2"
"T1"	1	"T1"
"T2"	2	"T2"
"Q2S"	Automatic	"Q2S" (serendipity)
"Q1"	1	"Q1"
"Q2S"	2	"Q2S" (serendipity)
"Q2"*	2	"Q2" (second order Lagrange)
"O2"	Automatic	"O2"
"O1"	1	"O1"
"O2"	2	"O2"
"H2S"	Automatic	"H2S" (serendipity)
"H1"	1	"H1"
"H2S"	2	"H2S" (serendipity)
"H2"	2	"H2" (second order Lagrange)

Note that second order `ToElementMesh` generated quadrilateral and hexahedron elements are of serendipity type elements (topology Q2S and H2S). If Lagrange type elements (Q2, H2) are required then each element of the mesh is automatically augmented with an additional center node. Quadrilateral and hexahedron elements are in general not generated by the `ToElementMesh` directly.

In[67]:=
```
coordinates = {{0., 0.}, {1., 0.}, {2., 0.}, {2.5, 0.5}, {0., 1.},
    {1., 1.}, {2., 1.}, {3., 1.}, {2.5, 1.5}, {0., 2.}, {1., 2.}, {2., 2.}};
e1 = QuadElement[{{1, 2, 6, 5}, {2, 3, 7, 6}, {5, 6, 11, 10}, {6, 7, 12, 11}}];
e2 = TriangleElement[{{3, 4, 7}, {4, 8, 7}, {7, 9, 12}, {7, 8, 9}}];
mesh = ToElementMesh["Coordinates" → coordinates, "MeshElements" → {e1, e2}];
```

In[71]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[{"trangle", {"ML:", "SE", "PE", "T1", "DF", "LE", "T1", "D", "Hooke"}, {}},
  {"quad", {"ML:", "SE", "PE", "Q1", "DF", "LE", "Q1", "D", "Hooke"}, {}}];
SMTAddMesh[mesh, {"T1" -> "trangle", "Q1" → "quad"}];
SMTAnalysis[];
SMTShowMesh[]
```



Structured Meshes

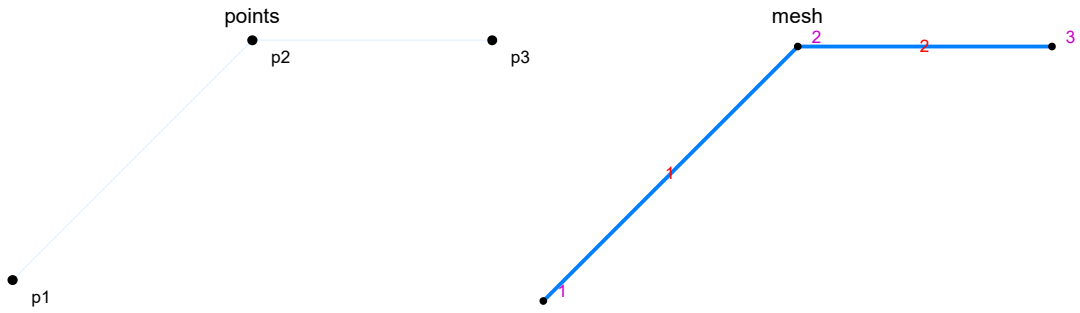
Line

SMTAddMesh[Line[{p_1,p_2,…,p_m}], *dID, meshType*]
Generates mesh of *m-1* 2D or 3D line elements with given coordinates of the nodes {p_1,p_2,…,p_m}. The domain identification *dID* defines the actual element type and the *meshType* mesh type (see `1 D structured mesh types`).

SMTAddMesh[Line[{p_1,p_2,…,p_m}], *dID, meshType, n*]
Generates mesh of *n* 2D or 3D line elements. The coordinates of the nodes of the actual mesh are obtained by interpolation of given points Line[{p_1,p_2,…,p_m}]. The order of interpolation is defined by the "InterpolationOrder" option with the default value 3. The domain identification *dID* defines the actual element type and the *meshType* mesh type (see `1 D structured mesh types`).

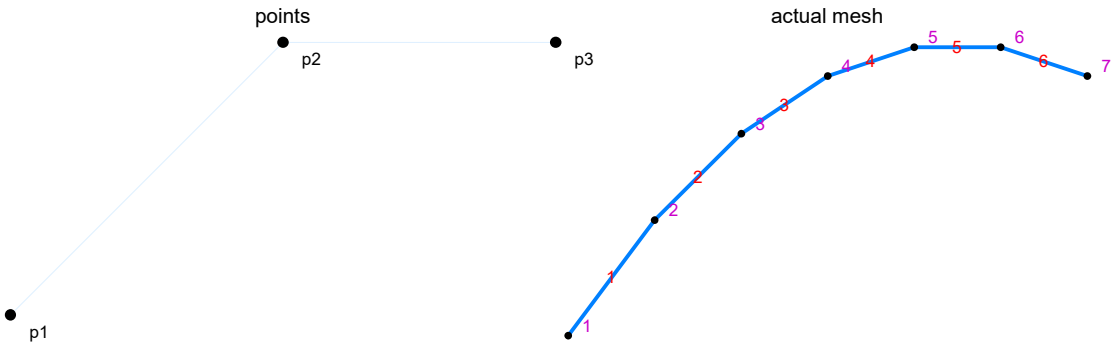
- Simple set of line elements.

In[79]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["A", "ExamplesGasPressure", {}];
points = {{0, 0}, {1, 1}, {2, 1}};
SMTAddMesh[Line[points], "A", "L1"];
SMTAnalysis[];
GraphicsRow[
 {Graphics[{LightBlue, Line[points], Black, PointSize[0.02], Point[points], Black, MapIndexed[
     Text["p" <> ToString[#2[[1]]], Offset[{15, -10}, #1]] &, points]}, PlotLabel → "points"],
  SMTShowMesh["Label" → "mesh", "Marks" → True]}, -50, ImageSize → 600]
```



- Higher order interpolation of given points.

In[86]:=
```
<< AceFEM`;
SMTInputData[]; nx = 6;
SMTAddDomain["A", "ExamplesGasPressure", {}];
points = {{0, 0}, {1, 1}, {2, 1}};
SMTAddMesh[Line[points], "A", "L1", nx, "InterpolationOrder" → 3];
SMTAnalysis[];
GraphicsRow[
 {Graphics[{LightBlue, Line[points], Black, PointSize[0.02], Point[points], Black, MapIndexed[
     Text["p" <> ToString[#2[[1]]], Offset[{15, -10}, #1]] &, points]}, PlotLabel → "points"],
  SMTShowMesh["Label" → "actual mesh", "Marks" → True]}, -50, ImageSize → 600]
```



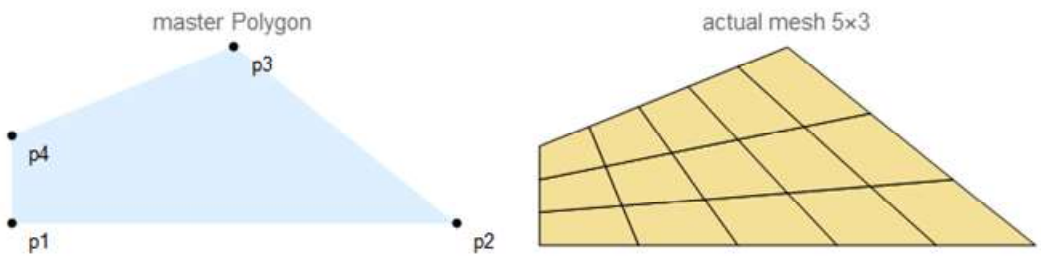
Quadrilateral

SMTAddMesh[Polygon[{p_1,p_2,p_3,p_4}], *dID, meshType,* {n_x,n_x}]

Generates structured mesh for an arbitrary quadrilateral region where {n_x,n_x} is the number of elements along local x and y axis. The domain identification *dID* defines the actual element type and the *meshType* mesh type (see `Types of Structured Meshes`). For meshing the polygons with more or less than 4 corners see `Simple unstructured meshes`.

In[93]:=
```
<< AceFEM`;
SMTInputData[];
L = 100; H = 20; nx = 5; ny = 3;
points = {{0, 0}, {L, 0}, {L / 2, 2 H}, {0, H}};
SMTAddDomain["A",
  {"ML:", "SE", "PE", "Q1", "DF", "JC", "Q1", "D", {{"NeoHooke", "WA"}, {"Mises", "ExH"}}},
  {"E *" → 21 000, "ν *" → 0.2, "σy *" → 24, "Kh *" → 200}];
SMTAddMesh[Polygon[points], "A", "Q1", {nx, ny}];
SMTAnalysis[];
```

In[100]:=
```
GraphicsRow[
 {Graphics[{LightBlue, Polygon[points], Black, PointSize[0.02], Point[points], Black,
    MapIndexed[Text["p" <> ToString[#2[[1]]], Offset[{15, -10}, #1]] &, points]},
   PlotLabel → "master Polygon"],
  SMTShowMesh["Label" → "actual mesh 5×3"]}, -50, ImageSize → 600]
```



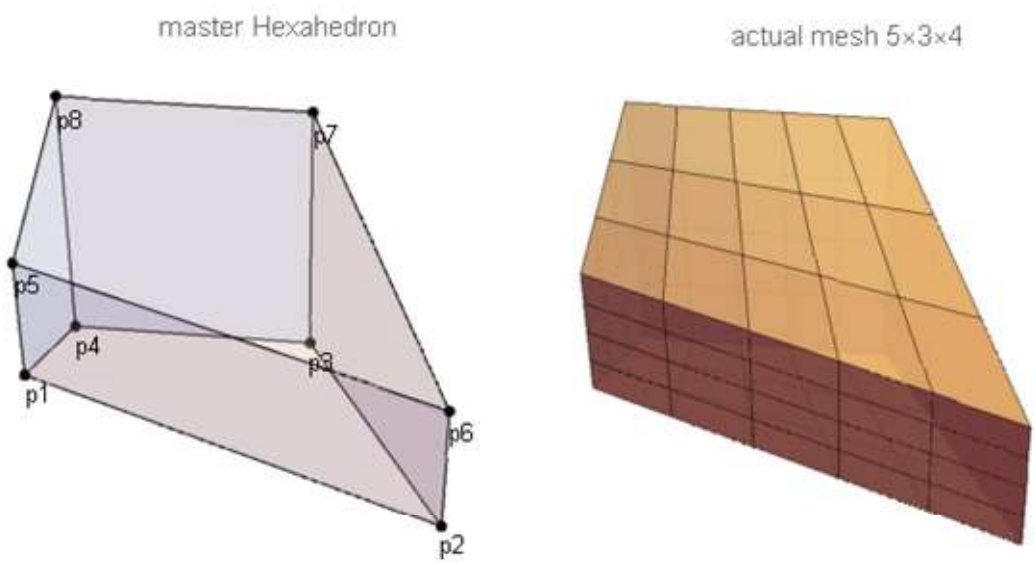
Hexahedron

SMTAddMesh[Hexahedron[{p_1,p_2,p_3,p_4,p_5,p_6,p_7,p_8}], *dID, meshType,* {n_x,n_x,n_z}]

Generates structured mesh for an arbitrary hexahedron region where {n_x,n_x,n_z} is the number of elements along local x and y and z axis. The domain identification *dID* defines the actual element type and the *meshType* the mesh type (see `Types of Structured Meshes`).

In[101]:=
```
<< AceFEM`;
SMTInputData[];
L = 100; H = 20; V = 30; nx = 5; ny = 3; nz = 4;
points = {{0, 0, 0}, {L, 0, 0}, {L / 2, 2 H, 0},
    {0, H, 0}, {0, 0, V}, {L, 0, V}, {L / 2, 2 H, 2 V}, {0, H, 2 V}};
SMTAddDomain["A", {"ML:", "SE", "D3", "H1", "DF", "LE", "H1", "D", "Hooke"}, {}];
SMTAddMesh[Hexahedron[points], "A", "H1", {nx, ny, nz}];
SMTAnalysis[];
```

In[108]:=
```
GraphicsRow[
 {Graphics3D[{LightBlue, {Opacity[0.2], Hexahedron[points]}, Black, PointSize[0.02], Point[
     points], Black, MapIndexed[Text["p" <> ToString[#2[[1]]], #1 + 2 {1, 1, -3}] &, points]},
   PlotLabel → "master Hexahedron", Boxed → False],
  SMTShowMesh["Label" → "actual mesh 5×3×4", "Opacity" -> 0.9]}, -10, ImageSize → 500]
```



Curved quadrilateral

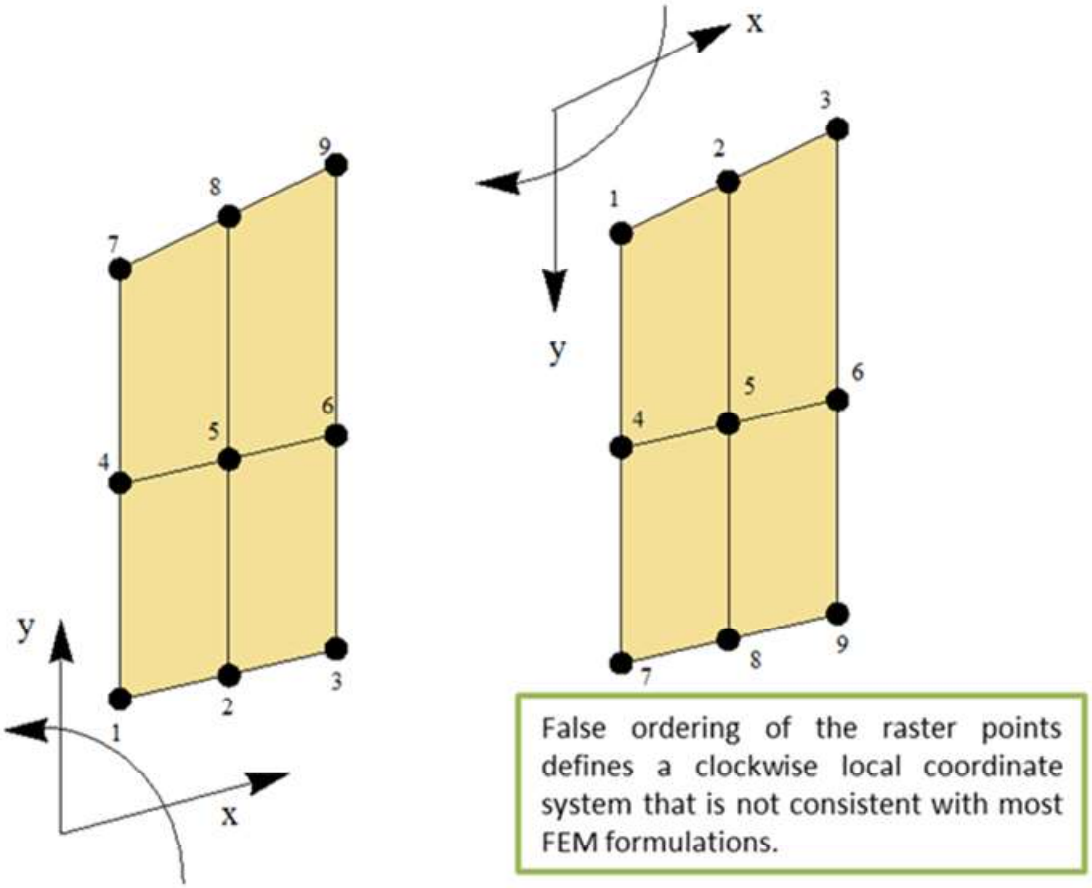
SMTAddMesh[Raster[*raster*], *dID, meshType,* {n_x,n_x}]

Generates structured mesh for an arbitrary shaped 2D region with four curved edges (curved quadrilateral) where {n_x,n_x} is the number of elements along local x and y axis. The region is defined by the two-dimensional array of points or *raster*. The domain identification *dID* defines the actual element type and the *meshType* the mesh type (see `Types of Structured Meshes`).

The *raster* parameter is a regular two dimensional array of arbitrary number of points that outlines the boundary of the problem domain. The coordinates of nodes of the actual mesh are obtained as multidimensional interpolation of raster points. The order of interpolation is defined by the "InterpolationOrder" option.

IMPORTANT: The *raster* points have to be given in a order that defines a counter-clockwise local coordinate system.

IMPORTANT: The "InterpolationOrder" is NOT related to the interpolation order of the elements used, the interpolation order of the elements is defined by the **actual element type**.



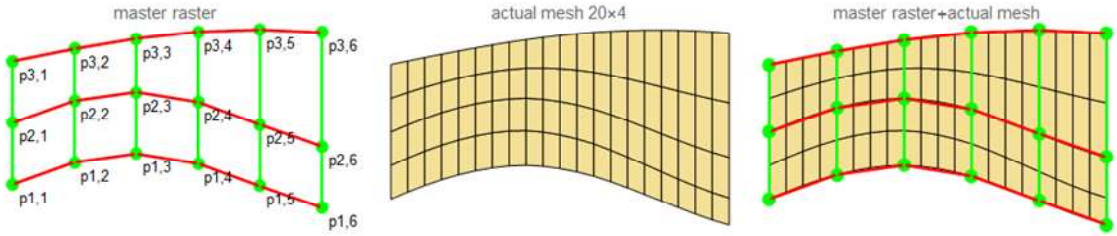
False ordering of the raster points defines a clockwise local coordinate system that is not consistent with most FEM formulations.

Correct ordering of the raster points defines counter-clockwise localcoordinate system.

2D mesh defined by 2D raster of points

In[109]:=
```
<< AceFEM`;
L = 10.;
raster = {Table[ {x, Sin[4 x / L]}, {x, 0, L, 2}],
    Table[ {x, Sin[4 x / L] + 2}, {x, 0, L, 2}],
    Table[ {x, Sin[2 x / L] + 4}, {x, 0, L, 2}]};
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "PS", "Q1", "DF", "HY", "Q1", "D", {{"NeoHooke", "WA"}}},
  {"E *" -> 1000., "ν *" -> .49}];
SMTAddMesh[Raster[raster], "A", "Q1", {20, 4}];
SMTAnalysis[];
```

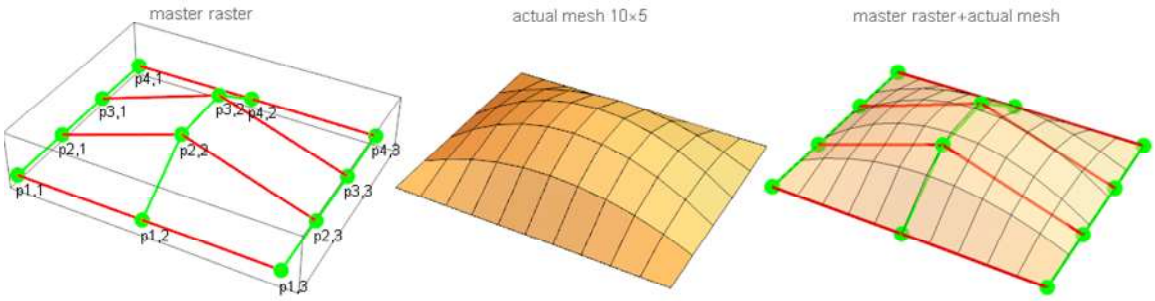
In[116]:=
```
GraphicsRow[{Graphics[{Green, PointSize[0.04], Point[Flatten[raster, 1]], Red, Thick,
     Map[Line, raster], Green, Map[Line, raster // Transpose], Black, MapIndexed[
      Text["p" <> ToString[#2[[1]]] <> "," <> ToString[#2[[2]]], Offset[{15, -10}, #1]] &,
      raster, {2}]}, PlotLabel → "master raster"],
   SMTShowMesh["Label" → "actual mesh 20×4"], Show[SMTShowMesh[],
    Graphics[{Green, PointSize[0.04], Point[Flatten[raster, 1]], Red,
      Thick, Map[Line, raster], Green, Map[Line, raster // Transpose]}],
    PlotLabel → "master raster+actual mesh"]}, -20, ImageSize → 800]
```



3D surface mesh defined by 2D raster of points.

In[117]:=
```
<< AceFEM`;
{a = 10, b = 8, h = 2};
geometry = {{{0, 0, 0}, {a, 0, 0}}, {{0, b, 0}, {a, b, 0}}} // N;
ShapeFunc[x_, y_] := (Sin[π x / a] Sin[π y / b]);
mastermeshdivision = {4, 3};
raster = Array[{#2, #1, h ShapeFunc[#2, #1]} &, mastermeshdivision, {{0, b}, {0, a}}] // N;
SMTInputData[];
SMTAddDomain["Shell1",
  {"ML:", "SE", "MS", "S1", "ES", "HY", "ANSE2P6", {"D", "Fi"}, "SVenant"}, {}];
SMTAddMesh[Raster[raster], "Shell1", "S1", {10, 5}];
SMTAnalysis[];
```

In[127]:=
```
GraphicsRow[{Graphics3D[{Green, PointSize[0.04], Point[Flatten[raster, 1]],
     Red, Thick, Map[Line, raster], Green, Map[Line, raster // Transpose], Black,
     MapIndexed[Text["p" <> ToString[#2[[1]]] <> "," <> ToString[#2[[2]]], #1 + {0.5, 0, -0.5}] &,
      raster, {2}]}, PlotLabel → "master raster"],
   SMTShowMesh["Label" → "actual mesh 10×5"], Show[SMTShowMesh["Opacity" -> 0.5],
    Graphics3D[{Green, PointSize[0.04], Point[Flatten[raster, 1]], Red,
      Thick, Map[Line, raster], Green, Map[Line, raster // Transpose]}],
    PlotLabel → "master raster+actual mesh"]}, -20, ImageSize → 800]
```



Curved Hexahedron

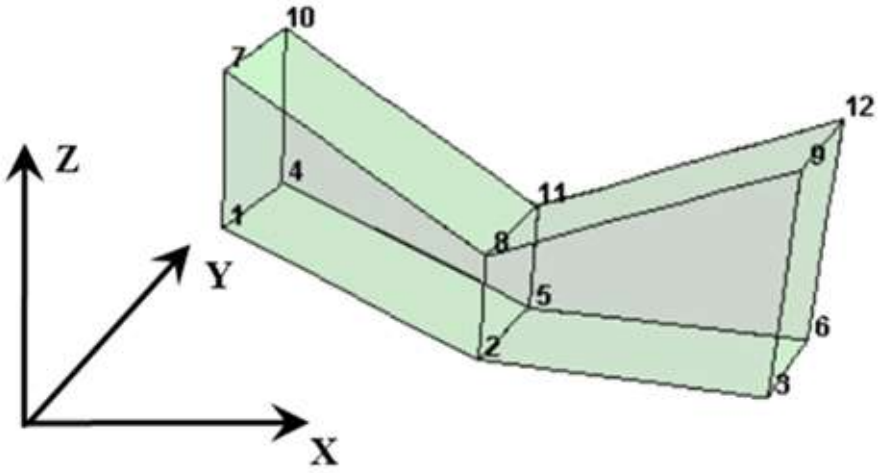
SMTAddMesh[Raster3D[*raster*], *dID, meshType,* {n_x,n_x,n_z}]

Generates structured mesh for 3D region defined by six arbitrary shaped surfaces (curved hexahedron) where {n_x,n_x,n_z} is the number of elements along local x and y and z axis.The region is defined by the three-dimensional array of points or *raster*. The domain identification *dID* defines the actual element type and the *meshType* the mesh type (see `Types of Structured Meshes`).

The *raster* parameter is a regular three dimensional array of arbitrary number of points that outlines the boundary of the problem domain. The coordinates of nodes of the actual mesh are obtained as multidimensional interpolation of raster points. The order of interpolation is defined by the "InterpolationOrder" option.

IMPORTANT: The *raster* points have to be given in a order that defines a counter-clockwise local coordinate system.

IMPORTANT: The "InterpolationOrder" is NOT related to the interpolation order of the elements used, the interpolation order of the elements is defined by the **actual element type**.

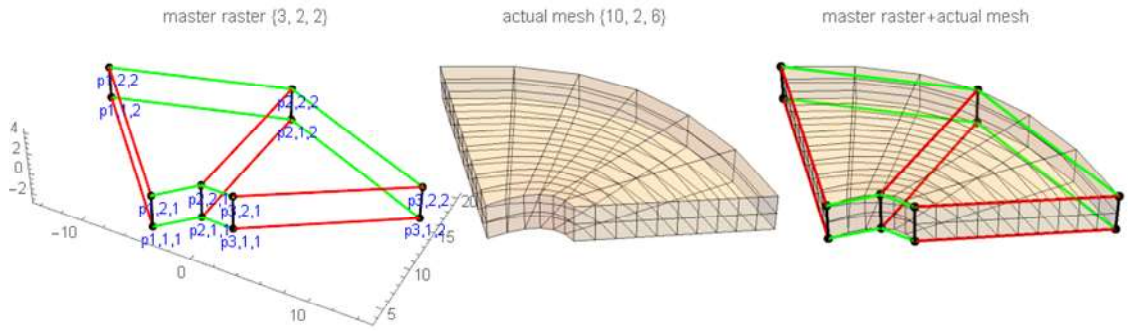


Correct ordering of the raster points defines 3D counter-clockwise local coordinate system.

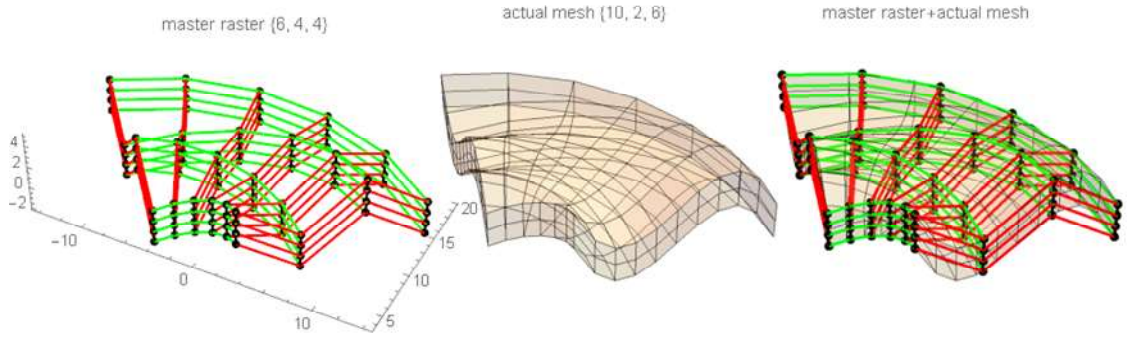
3D mesh defined by 3D raster of points

In[128]:=
```
<< AceFEM`;
a = 10; b = 8; h = 3;
geometry = {{{0, 0, 0}, {a, 0, 0}}, {{0, b, 0}, {a, b, 0}}} // N;
ninp = {0.5, 0.5};
ShapeFunc[x_, y_] := (Sin[ninp[[1]] x] Sin[2 ninp[[2]] y]);
mastermeshdivision = 10 {3, 2, 2};
raster = Array[{#3 Sin[#1] , #3 Cos[#1], #2 + h ShapeFunc[#3, #1]} &, mastermeshdivision,
    {{π / 4, 3 π / 4} - π / 2, {0, h}, {5, 20}}] // N;
SMTInputData[];
SMTAddDomain["Solid1",
  {"ML:", "SE", "D3", "H1", "ES", "HY", "H1E9", "D", {{"NeoHooke", "WA"}}}, {}];
SMTAddMesh[Raster3D[raster], "Solid1", "H1", div = 3 {10, 2, 6}];
SMTAnalysis[];
```

In[139]:=
```
GraphicsRow[{Graphics3D[{Black, PointSize[0.02], Point[Flatten[raster, 2]], Red, Thick,
     Map[Line, raster, {2}], Green, Map[Line, Transpose[raster, {3, 2, 1}], {2}],
     Black, Map[Line, Transpose[raster, {1, 3, 2}], {2}], Blue,
     MapIndexed[Text["p" <> ToString[#2[[1]]] <> "," <> ToString[#2[[2]]] <>
         "," <> ToString[#2[[3]]], #1 + {0.5, 0.5, -1.5}] &, raster, {3}]},
   PlotLabel → Row[{"master raster ", mastermeshdivision}], Axes → True, Boxed → False],
  SMTShowMesh["Opacity" → 0.2, "Label" → Row[{"actual mesh ", div}]],
  Show[SMTShowMesh["Opacity" → 0.2],
   Graphics3D[{Black, PointSize[0.02], Point[Flatten[raster, 2]], Red, Thick,
     Map[Line, raster, {2}], Green, Map[Line, Transpose[raster, {3, 2, 1}], {2}],
     Black, Map[Line, Transpose[raster, {1, 3, 2}], {2}]}],
   PlotLabel → "master raster+actual mesh"]}, -100, ImageSize → 800]
```



- Denser raster and mesh for the same problem



Importing Externally Generated Meshes

SMTAddMesh[{{X_1,Y_1,Z_1},{X_2,Y_2,Z_2},...}, {*dID*$_1$->{{n_1^1,n_2^1,...},{n_1^2,n_2^2,...},...}, *dID*$_2$→ {{n_1^1,n_2^1,...},{n_1^2,n_2^2,...},...}, ...}]

Imports mesh defined by the list of node coordinates {{X_1,Y_1,Z_1},{X_2,Y_2,Z_2},...} and a list of regions defined by domain identification *dID*$_i$ and connectivity table, {{n_1^1,n_2^1,...},{n_1^2,n_2^2,...},...}. This form can be used to import mesh generated by an arbitrary external mesher. Node numbers in connectivity table refer to the successive indexes of the nodes given.

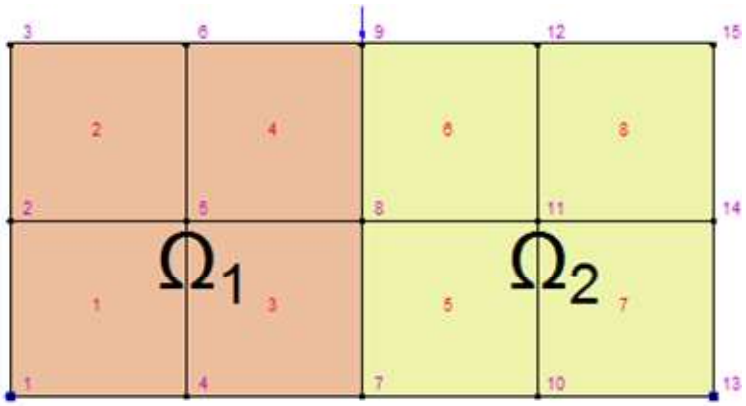
SMTAddMesh[{*dID*$_1$->{{n_1^1,n_2^1,...},{n_1^2,n_2^2,...},...}, *dID*$_2$->{{n_1^1,n_2^1,...},{n_1^2,n_2^2,...},...}, ...}]

Imports mesh defined by a list of regions defined by domain identification *dID*$_i$ and connectivity table, {{n_1^1,n_2^1,...},{n_1^2,n_2^2,...},...}. Node numbers in connectivity table refer to the global node numbers. This form can be used to import several sub-meshes imposed on the existing mesh of spatial nodal points.

SMTAddMesh[{gn_1,gn_2,...}, {*dID*$_1$->{{n_1^1,n_2^1,...},{n_1^2,n_2^2,...},...}, *dID*$_2$->{{n_1^1,n_2^1,...},{n_1^2,n_2^2,...},...}, ...}]

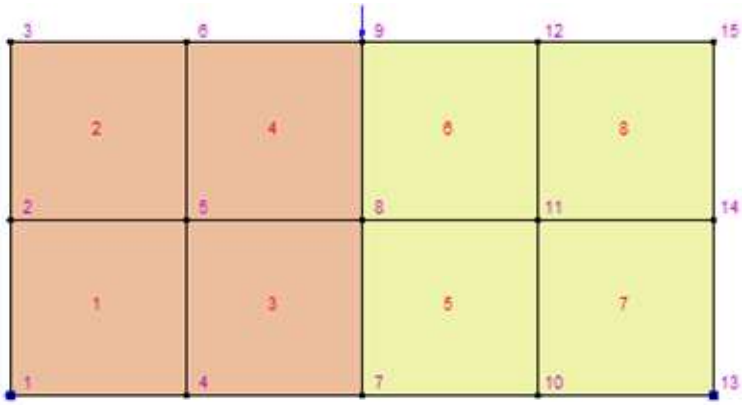
Imports mesh defined by a list of regions defined by domain identification *dID*$_i$ and connectivity table, {{n_1^1,n_2^1,...},{n_1^2,n_2^2,...},...}. Node numbers in connectivity table refer to the successive elements in the list of global node numbers {gn_1,gn_1,...} where gn_i is a global node number of the *i*-th local node. This form can be used to import several sub-meshes imposed on the existing mesh of spatial nodal points where each sub-mesh has its own node numbering.

Combine several meshes defined by the list of nodes and elements



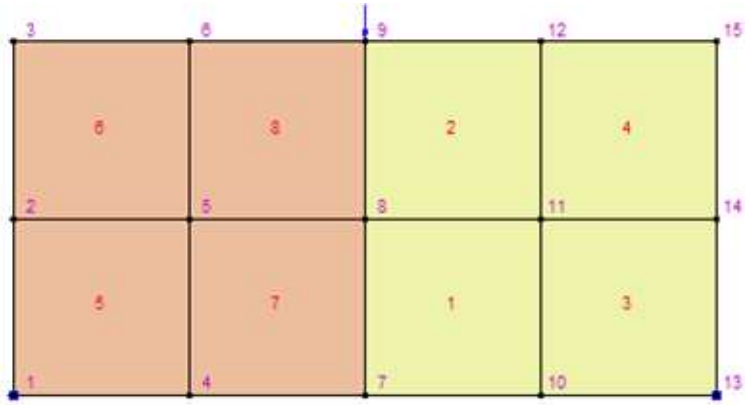
The input data for the example above is created using local (independent) node numbering for domains Ω_2 and Ω_1. The domains are tied together after SMTAnalysis command. Note that the boundary conditions should be here defined using geometric entities since the actual (global) node numbers are not known in advance.

In[140]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[
  {"Ω1", {"ML:", "SE", "PE", "Q1", "ES", "LE", "Q1E4", "D", "Hooke"},
   {"E *" -> 1000, "ν *" -> 0.3}},
  {"Ω2", {"ML:", "SE", "PE", "Q1", "ES", "LE", "Q1E4", "D", "Hooke"},
   {"E *" -> 5000, "ν *" -> 0.2}}
 ];
SMTAddMesh[
  {{0., 0.}, {0., 2.5}, {0., 5.}, {2.5, 0.}, {2.5, 2.5}, {2.5, 5.}, {5., 0.}, {5., 2.5}, {5., 5.}}
  , {"Ω1" -> {{1, 4, 5, 2}, {2, 5, 6, 3}, {4, 7, 8, 5}, {5, 8, 9, 6}}}
 ];
SMTAddMesh[
  {{5., 0.}, {5., 2.5}, {5., 5.}, {7.5, 0.},
   {7.5, 2.5}, {7.5, 5.}, {10., 0.}, {10., 2.5}, {10., 5.}}
  , {"Ω2" -> {{1, 4, 5, 2}, {2, 5, 6, 3}, {4, 7, 8, 5}, {5, 8, 9, 6}}}
 ];
SMTAddEssentialBoundary[Point[{0, 0}], 1 -> 0, 2 → 0];
SMTAddEssentialBoundary[Point[{10, 0}], 1 -> 0, 2 → 0];
SMTAddNaturalBoundary[Point[{5, 5}], 2 → -1];
SMTAnalysis[];
SMTShowMesh["Marks" -> True, "BoundaryConditions" -> True]
```



The input data for the above example is here created by first defining nodes and after that domains Ω_2 and Ω_1. The mapping from local to global node numbers is stored in list *nodes*. Note that the boundary conditions are here defined using actual (global) node numbers and the "Tie" is prevented.

In[150]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[
   {"Ω1", {"ML:", "SE", "PE", "Q1", "ES", "LE", "Q1E4", "D", "Hooke"},
    {"E *" -> 1000, "ν *" -> 0.3}},
   {"Ω2", {"ML:", "SE", "PE", "Q1", "ES", "LE", "Q1E4", "D", "Hooke"},
    {"E *" -> 5000, "ν *" -> 0.2}}
 ];
nodes = SMTAddMesh[
     {{0., 0.}, {0., 2.5}, {0., 5.}, {2.5, 0.}, {2.5, 2.5}, {2.5, 5.}, {5., 0.}, {5., 2.5},
      {5., 5.}, {7.5, 0.}, {7.5, 2.5}, {7.5, 5.}, {10., 0.}, {10., 2.5}, {10., 5.}}
     , {}
    ][[1]];
SMTAddMesh[Range @@ nodes
   , {"Ω2" -> {{7, 10, 11, 8}, {8, 11, 12, 9}, {10, 13, 14, 11}, {11, 14, 15, 12}},
    "Ω1" -> {{1, 4, 5, 2}, {2, 5, 6, 3}, {4, 7, 8, 5}, {5, 8, 9, 6}}}
 ];
SMTAddEssentialBoundary[{1, 13}, 1 -> 0, 2 → 0];
SMTAddNaturalBoundary[9, 2 → -1];
SMTAnalysis["Tie" -> False];
SMTShowMesh["Marks" -> True, "BoundaryConditions" -> True]
```



Adding Individual Elements and Nodes

SMTAddNode

SMTAddNode[{*X*,*Y*,*Z*}]
SMTAddNode[{*X*,*Y*}]
SMTAddNode[{*X*}]
appends a new (1D, 2D or 3D) node to the list of nodes and returns node indices of newly created nodes

SMTAddNode[{{X_1,Y_1,Z_1},{X_2,Y_2,Z_2},…}]
appends a collection of nodes to the list of nodes and returns indexes of newly created nodes

Several data sets can be added at the same time as well. For example: SMTAddNode[{{1,1},{2,2}}] adds 2 nodes.

The function returns the node index or the range of indexes of newly created nodes.

SMTAddElement

SMTAddElement[*dID*,{$node_1$, $node_2$,…}]

appends new element to the list of elements (element is defined by the list of nodes {$node_1$,$node_2$,...} and domain identification *dID* where $node_i$ can be either global node number or a coordinates of the node {X_i,Y_i,Z_i})

SMTAddElement[{ {dID_1,{$n_{1,1}$, $n_{1,2}$,...}}, {dID_2,{$n_{2,1}$, $n_{2,2}$,...}},... }]
appends a collection of new elements where dID_i is domain identification of *i*-th element and $n_{i,j}$ is *j*-th node of *i*-th element

option	default	description
"BodyID"	"None"	body identification string
"BoundaryDomainID"	{}	domain identification (one or more) for the elements additionally generated on the outer surface of elements with the same BodyID

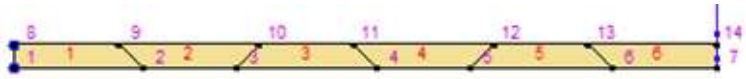
Options of the *SMTAddElement* function.

The function returns the element index or the range of indexes of new elements.

Several data sets can be added at the same time as well. For example: SMTAddElement[{{"A",{1,2}},{"A",{2,3}}}] adds 2 elements.

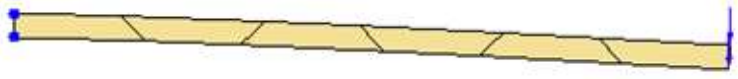
Example: Adding individual elements and nodes

In[159]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "PS", "Q1", "ES", "LE", "Q1E4", "D", "Hooke"},
  {"E *" -> 11.×10^7, "ν *" -> 0.3, "t *" -> 0.1}];
SMTAddNode[{{0.0, -0.2}, {1.1, -0.2}, {1.9, -0.2}, {3.1, -0.2}, {3.9, -0.2},
   {5.1, -0.2}, {6.0, -0.2}, {0.0, 0}, {0.9, 0}, {2.1, 0},
   {2.9, 0}, {4.1, 0}, {4.9, 0}, {6.0, 0}}];
SMTAddElement[{{"A", {1, 2, 9, 8}}, {"A", {2, 3, 10, 9}}, {"A", {3, 4, 11, 10}},
{"A", {4, 5, 12, 11}}, {"A", {5, 6, 13, 12}}, {"A", {6, 7, 14, 13}}}];
SMTAddNaturalBoundary["X" == 6 & , 2 -> -0.5];
SMTAddEssentialBoundary["X" == 0 &, 1 -> 0, 2 -> 0];
SMTAnalysis[];
SMTShowMesh["BoundaryConditions" -> True, "Marks" → True]
```



In[168]:=
```
SMTNextStep["λ" -> 1];
While[SMTConvergence[10^-12, 10], SMTNewtonIteration[]];
SMTPostData["v", Point[{6, 0}]]
SMTShowMesh["BoundaryConditions" -> True, "DeformedMesh" -> True, "Scale" -> 1000]
```

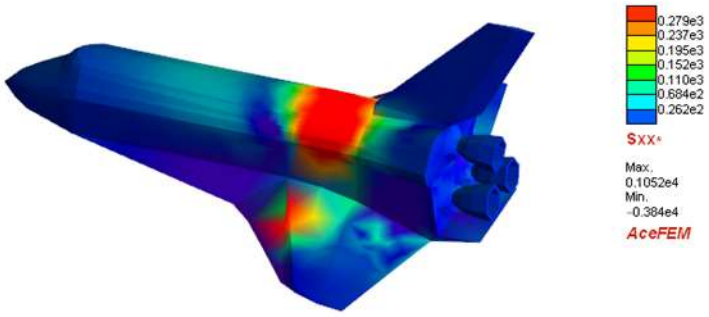
Out[170]=
```
-0.000264364
```



Advanced Examples

Space Shuttle

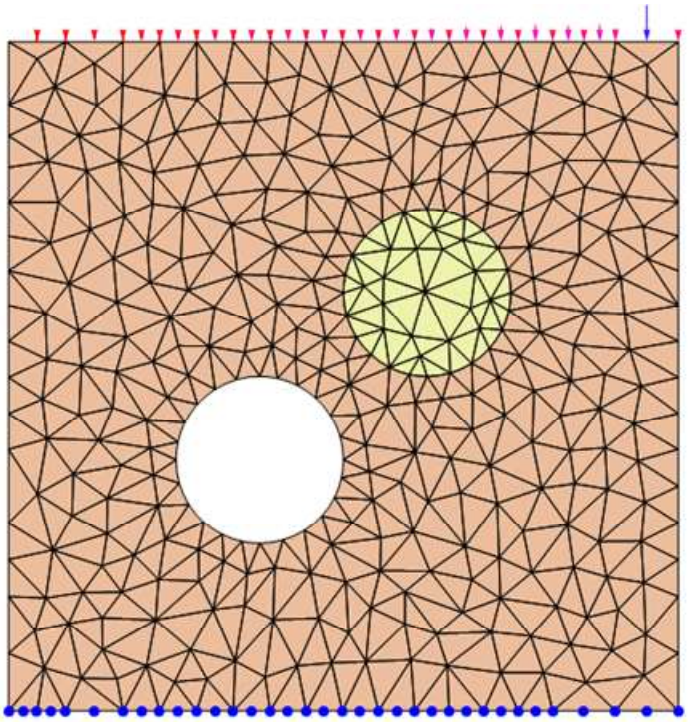
In[172]:=
```
<< AceFEM`;
gc = DiscretizeGraphics[ExampleData[{"Geometry3D", "SpaceShuttle"}, "GraphicsComplex"]];
mesh = ToElementMesh[gc, "MeshOrder" → 1, MeshQualityGoal → 1];
SMTInputData[];
SMTAddDomain["SpaceShuttle", {"ML:", "SE", "D3", "O1", "DF", "LE", "O1", "D", "Hooke"}, {}];
SMTAddMesh[mesh, "SpaceShuttle"];
SMTAnalysis[];
SMTAddNaturalBoundary[{"DistributedOver", "SpaceShuttle"}, 3 → -1];
SMTAddEssentialBoundary["Z" < -1.3 &, 1 → 0, 2 → 0, 3 → 0];
SMTNextStep["Δλ" → 1];
SMTNewtonIteration[];
SMTShowMesh["DeformedMesh" → True, "BoundaryConditions" → False, "Field" → "Sxx*", "Mesh" → False]
```



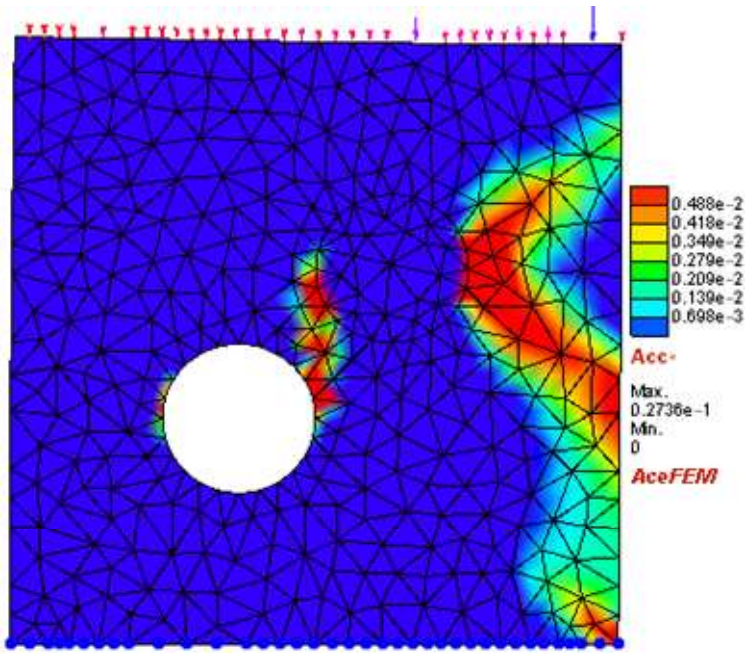
2D region with circular holes and subregions

In[184]:=
```
Ω = ImplicitRegion[(x - 1 / 2) ^ 2 + (y - 1 / 2) ^ 2 ≥ (1 / 2) ^ 2 &&
     (x + 1 / 2) ^ 2 + (y + 1 / 2) ^ 2 ≥ (1 / 2) ^ 2, {{x, -2, 2}, {y, -2, 2}}];
mesh = ToElementMesh[Ω, "RegionHoles" → {{-1 / 2, -1 / 2}},
   "RegionMarker" → {{{1 / 2, 1 / 2}, 2}, {{1 / 2, -1 / 2}, 1}}, "MaxBoundaryCellMeasure" → 1];
```

In[186]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[
  {"T2 matrix",
   {"ML:", "SE", "PS", "T2", "DF", "JC", "T2", "D", {{"NeoHooke", "WA"}, {"Mises", "ExH"}}},
   {"E *" -> 1000, "ν *" -> 0.3, "σy *" → 10, "Kh *" → 100}},
  {"T2 inclusion", {"ML:", "SE", "PS", "T2", "DF", "JC", "T2", "D", {{"NeoHooke", "WA"},
     {"Mises", "ExH"}}}, {"E *" -> 100, "ν *" -> 0.3, "σy *" → 10, "Kh *" → 10}}
 ];
SMTAddMesh[mesh, {{"T2", 1} -> "T2 matrix", {"T2", 2} → "T2 inclusion"}];
SMTAddEssentialBoundary[Line[{{-2, -2}, {2, -2}}], 1 → 0, 2 → 0];
SMTAddNaturalBoundary[Line[{{-2, 2}, {2, 2}}], 2 → Line[{0, -1}]];
SMTAnalysis[];
SMTShowMesh["BoundaryConditions" → True]
```

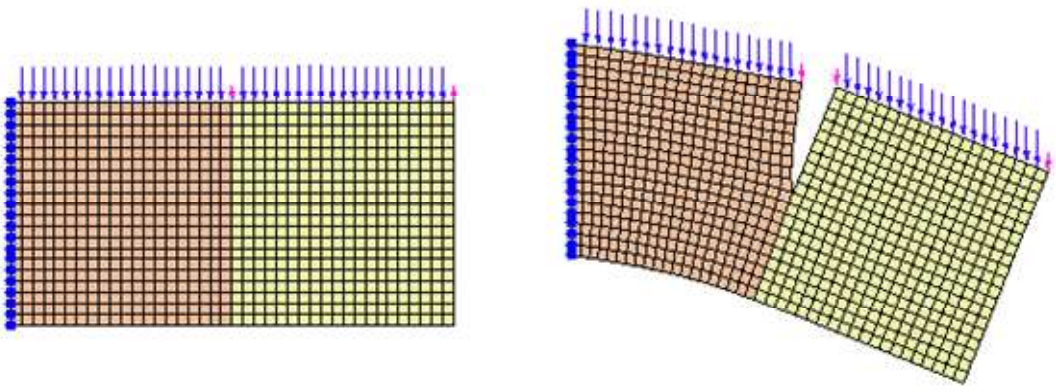


In[194]:=
```
λ0 = 0.1; ΔλMin = 0.001; ΔλMax = 0.2; λMax = 10;
tolNR = 10^-8; maxNR = 10; targetNR = 7;
SMTNextStep["Δλ" → λ0];
While[
  While[
   step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λMax}]
   , SMTNewtonIteration[];
  ];
  If[step[[4]] === "MinBound", SMTStatusReport["Δλ < ΔλMin"];];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" → step[[2]]]
 ];
SMTShowMesh["DeformedMesh" → True, "BoundaryConditions" → True, "Field" → "Acc*"]
```



Partial tie of subregions

In[199]:=
```
SMTInputData[];
L = 10; Q = 15; ν = 1;
SMTAddDomain[{"Ω1",
    {"ML:", "SE", "PS", "Q1", "DF", "LE", "Q1", "D", "Hooke"}, {"E *" -> 1000, "ν *" -> 0.3}},
  {"Ω2", {"ML:", "SE", "PS", "Q1", "DF", "LE", "Q1", "D", "Hooke"},
    {"E *" -> 5000, "ν *" -> 0.2}}];
SMTAddEssentialBoundary[ "X" == 0 &, 1 -> 0, 2 → 0];
SMTAddNaturalBoundary[ "Y" == L / 2 &, 2 → Line[{-Q}]];
SMTAddMesh[Polygon[{{0, 0}, {L / 2, 0}, {L / 2, L / 2}, {0, L / 2}}], "Ω1", "Q1", {20, 20}];
SMTAddMesh[Polygon[{{L / 2, 0}, {L, 0}, {L, L / 2}, {L / 2, L / 2}}], "Ω2", "Q1", {20, 20}];
SMTAnalysis["Tie" → {True, ("X" == L / 2 && "Y" > L / 4 &)}];
SMTNextStep["λ" -> 1];
SMTNewtonIteration[];
GraphicsRow[{SMTShowMesh["BoundaryConditions" → True],
  SMTShowMesh["DeformedMesh" → True, "BoundaryConditions" → True]}, ImageSize → 600]
```



In[210]:=

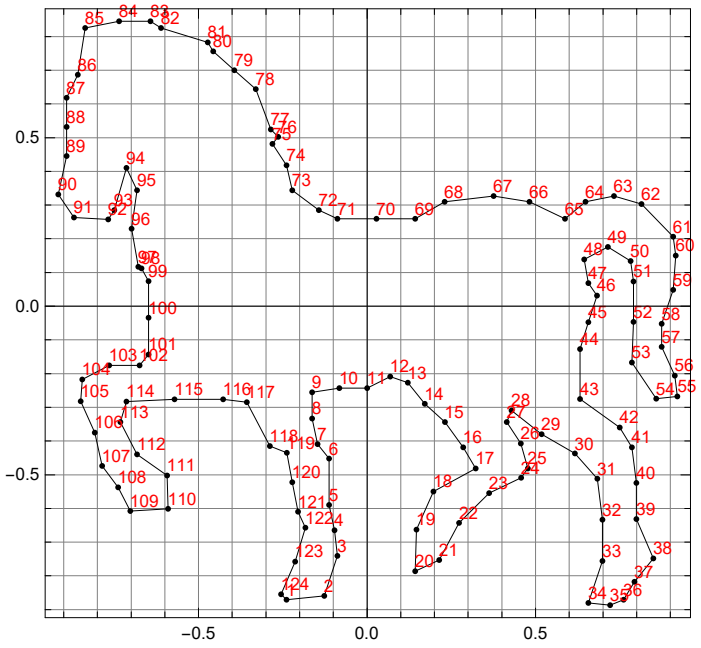
Convert an image to an element mesh

In[211]:=
```
<< AceFEM`;
```

```
shape = ColorConvert[ <image> , "Grayscale"];
```

In[213]:=
```
dist = DistanceTransform[Image[1 - ImageData[EdgeDetect[shape]]]];
data = Transpose[Reverse[-ImageData[dist] * (2 * ImageData[shape] - 1)]];
dataRange = Dimensions[data] / Max[Dimensions[data]] // N;
levelset =
  ListInterpolation[ data, Transpose[{-dataRange, dataRange}], InterpolationOrder → 1];
bundary = ToBoundaryMesh[ImplicitRegion[levelset[x, y] <= 0 , {x, y}],
   Transpose[{-dataRange, dataRange}], "BoundaryMeshGenerator" → "RegionPlot"];
```

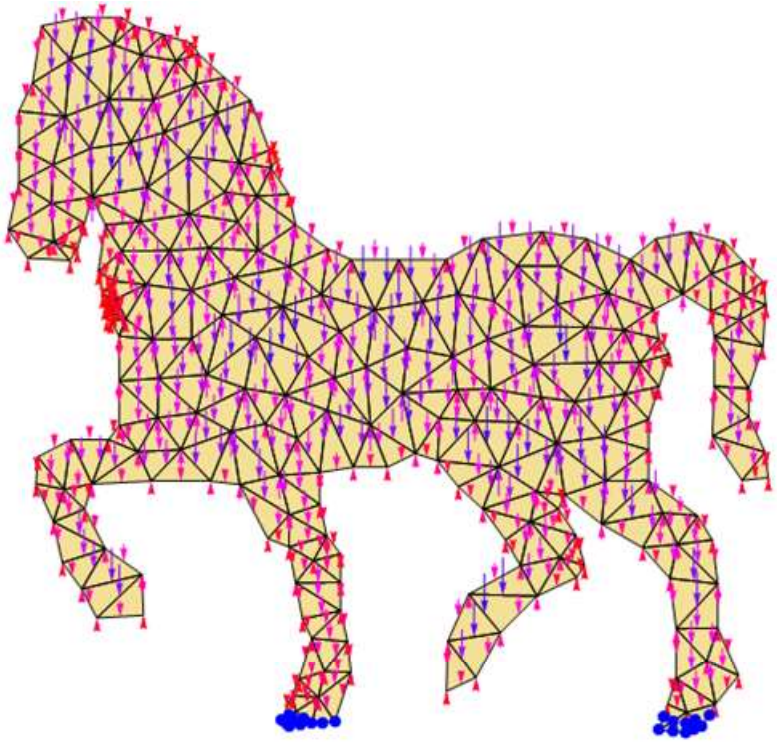
In[218]:=
```
Show[bundary["Wireframe"[Axes → True, GridLines → {Range[-1, 1, 0.1], Range[-1, 1, 0.1]}]],
 bundary["Wireframe"["MeshElement" → "PointElements", "MeshElementIDStyle" → Red]],
 AspectRatio → Automatic, Frame → True]
```



In[219]:=
```
mesh = ToElementMesh[bundary];
```

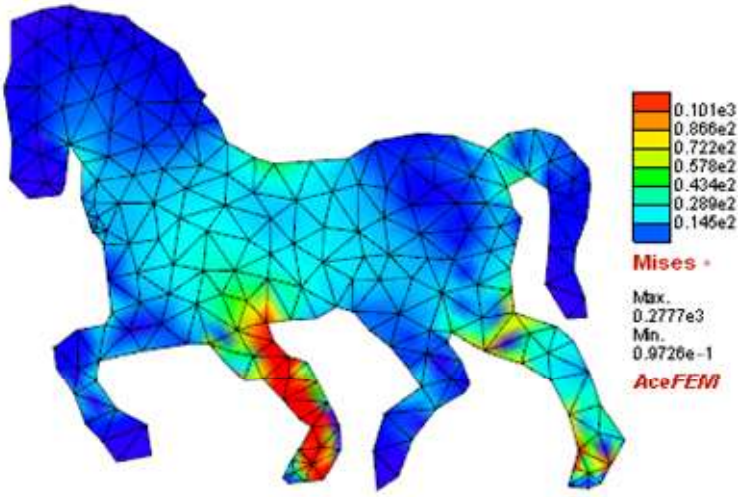
In[220]:=
```
SMTInputData[];
SMTAddDomain[
  {"Horse",
   {"ML:", "SE", "PS", "T2", "DF", "LE", "T2", "D", "Hooke"}, {"E *" -> 1000, "ν *" -> 0.3}}
 ];
SMTAddMesh[mesh, "Horse"];
SMTAddEssentialBoundary["Y" < -0.84 &, 1 → 0, 2 → 0];
SMTAddNaturalBoundary[{"DistributedOver", "Horse"}, 2 → -1];
SMTAnalysis[];
```

In[226]:=
```
SMTShowMesh["BoundaryConditions" → True]
```



In[227]:=
```
SMTNextStep["λ" → 10];
SMTNewtonIteration[];
```

In[229]:=
```
SMTShowMesh["DeformedMesh" → True, "Field" → "Mises *"]
```



Convert triangular mesh to quadrilateral mesh

SMTTriangularToQuad[*triangularElementMesh*]

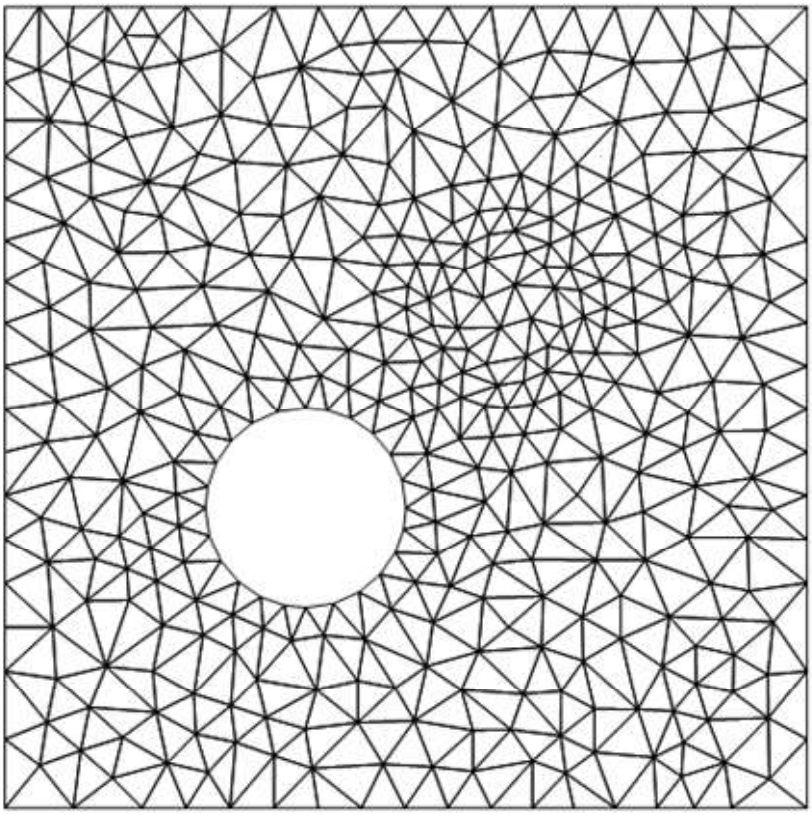
Converts triangular mesh defined by `ElementMesh` data object *triangularElementMesh* generated by built-in *Mathematica* mesher (see `Element Mesh Generation`) into quadrilateral only mesh data object.

ToElementMesh directly creates only triangular meshes. Unstructured mesh composed of linear triangles can be converted to unstructured mesh composed of quadrilateral with the help of an additional function defined here. Algorithm is base on procedures described in HOUMAN BOROUCHAKI AND PASCAL J. FRE, ADAPTIVE TRIANGULAR QUADRILATERAL MESH GENERATION, INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, VOL. 41, 915-934 (1998).

In[230]:=
```
<< AceFEM`;
```

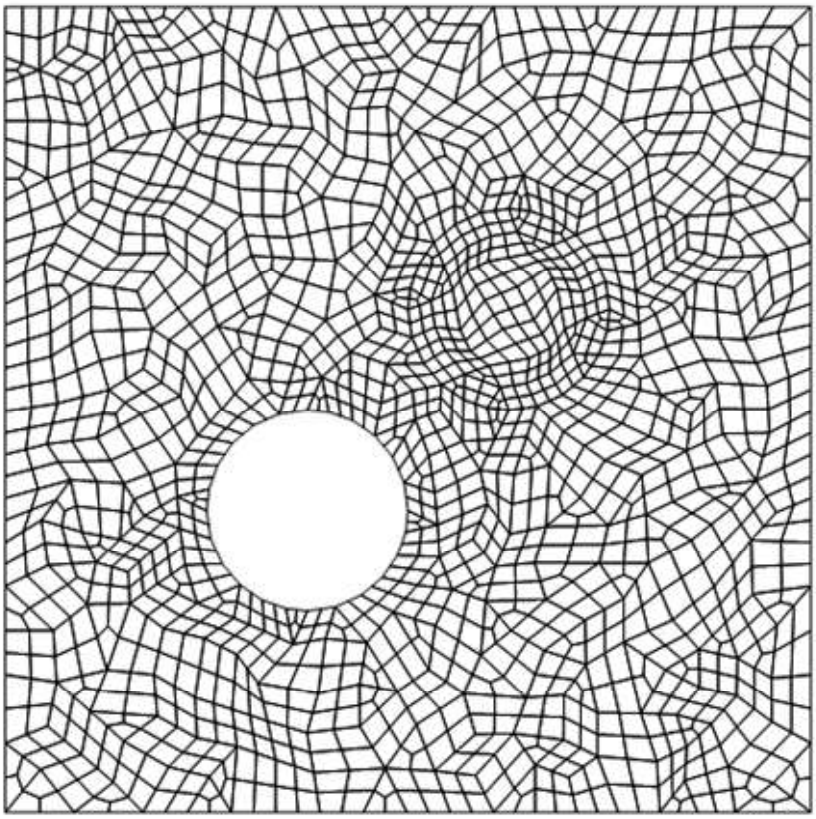
- 2D region with circular hole and subregions is here meshed with unstructured triangular mesh.

In[231]:=
```
Ω = ImplicitRegion[(x - 1/2)^2 + (y - 1/2)^2 ≥ (1/2)^2 &&
      (x + 1/2)^2 + (y + 1/2)^2 ≥ (1/2)^2, {{x, -2, 2}, {y, -2, 2}}];
mesh = ToElementMesh[Ω, "RegionHoles" → {{-1/2, -1/2}},
    "RegionMarker" → {{{1/2, 1/2}, 2}, {{1/2, -1/2}, 1}},
    "MaxBoundaryCellMeasure" → 1, "MeshOrder" → 1, MeshQualityGoal → 1];
mesh["Wireframe"]
```



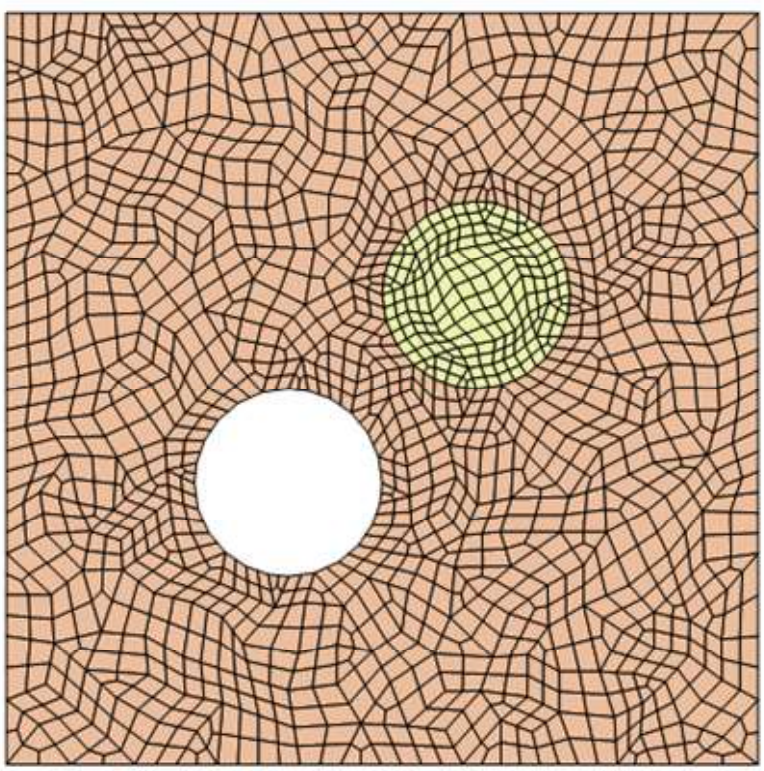
- Unstructured triangular mesh is here converted to unstructured quadrilateral mesh. The resulting mesh is twice denser than the original triangular mesh.

In[234]:=
```
mesh = SMTTriangularToQuad[mesh];
mesh["Wireframe"]
```

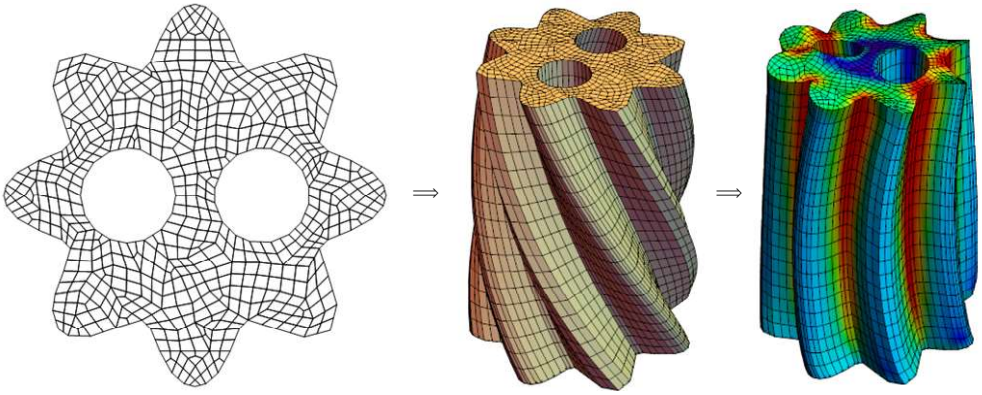


- Generated mesh is here imported into CDriver.

In[236]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[
  {"Q1 matrix",
   {"ML:", "SE", "PS", "Q1", "DF", "JC", "Q1", "D", {{"NeoHooke", "WA"}, {"Mises", "ExH"}}},
   {"E *" -> 1000, "ν *" -> 0.3, "σy *" → 10, "Kh *" → 100}},
  {"Q1 inclusion", {"ML:", "SE", "PS", "Q1", "DF", "JC", "Q1", "D", {{"NeoHooke", "WA"},
     {"Mises", "ExH"}}}, {"E *" -> 100, "ν *" -> 0.3, "σy *" → 10, "Kh *" → 10}}
 ];
SMTAddMesh[mesh, {{"Q1", 1} -> "Q1 matrix", {"Q1", 2} → "Q1 inclusion"}];
SMTAnalysis[];
SMTShowMesh[]
```



3D hexahedral unstructured/structured mesh

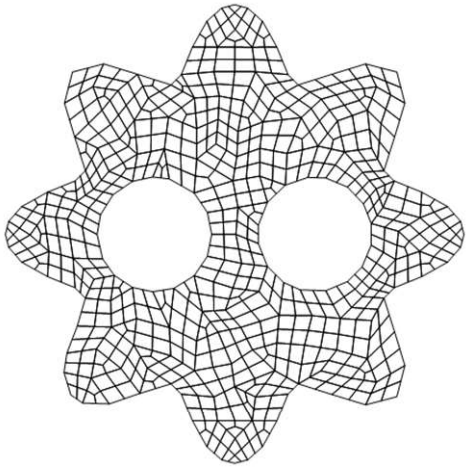


Two-dimensional triangular mesh can always be converted into two-dimensions unstructured quadrilateral mesh. However, the similar procedure that would convert three-dimensional tetrahedral mesh into three-dimensional hexahedral mesh does not exist. In the case that the shape of the structure is relatively simple in at least one direction, one can combine two-dimensional unstructured quadrilateral mesh in layers in order to produces three dimensional unstructured/structured hexahedral mesh as presented on an example below.

In[242]:=
```
<< AceFEM`
```

- 2D region with the holes is here meshed with two-dimensional quadrilateral mesh.

In[243]:=
```
Ω = ImplicitRegion[(x - 0.7)^2 + y^2 ≥ (1 / 2)^2 && (x + 0.7)^2 + y^2 ≥ (1 / 2)^2 &&
      x^2 + y^2 < 3 + Cos[8 ArcTan[x, y]], {{x, -2, 2}, {y, -2, 2}}];
mesht = ToElementMesh[Ω, "MeshOrder" → 1, "MaxCellMeasure" → .05,
    "BoundaryMeshGenerator" → "RegionPlot"];
mesh = SMTTriangularToQuad[mesht];
mesh["Wireframe"]
```



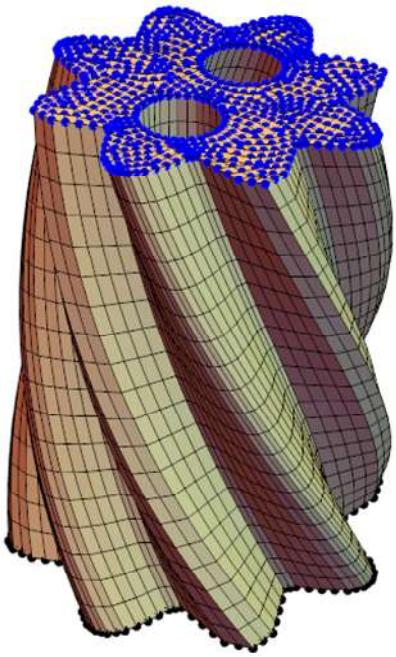
- 20 layers of 2D mesh is here combined into 3D hexahedral mesh. Each layer is additionally rotated for fixed angle and stretched in radial direction.

In[247]:=
```
H = 5; ϕ = -π / 2; stretch = -0.2;
nlay = 20;
dz = H / nlay; dϕ = ϕ / nlay; dstretch = stretch / nlay;
D2coord = mesh["Coordinates"];
D2elem = mesh["MeshElements"][[1, 1]];
D2nnodes = D2coord // Length;
nodes = Flatten[Table[rot = RotationTransform[(l - 1) dϕ];
    Map[Join[(1 + (l - 1) dstretch) rot[#], {(l - 1) dz}] &, D2coord] , {l, nlay + 1}], 1];
H1elem = Flatten[Table[Map[Join[D2nnodes (l - 1) + #, D2nnodes l + #] &, D2elem], {l, nlay}], 1];
```

- Problem is meshed with 3D elasto-plastic problems. The upper surface is then rotated for 360 degrees.

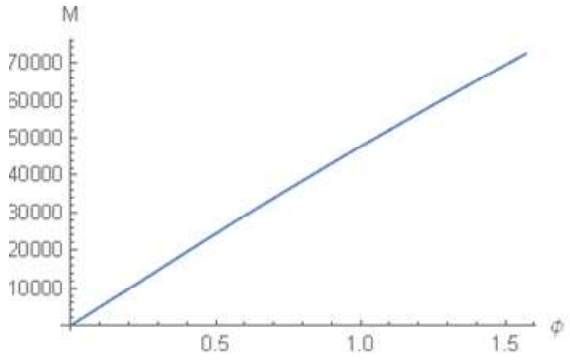
In[255]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["H1 body",
  {"ML:", "SE", "D3", "H1", "DF", "HY", "H1", "D", {{"NeoHooke", "WA"}}}, {}];
SMTAddMesh[nodes, {"H1 body" -> H1elem}];
SMTAddEssentialBoundary["Z" == 0 &, 1 -> 0, 2 → 0, 3 → 0];
SMTAddEssentialBoundary["Z" == H &, 1 -> 0, 2 → 0];
SMTAnalysis[];
```

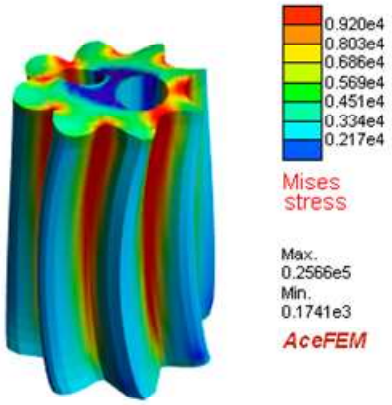
In[262]:=
```
SMTShowMesh["BoundaryConditions" → True]
```



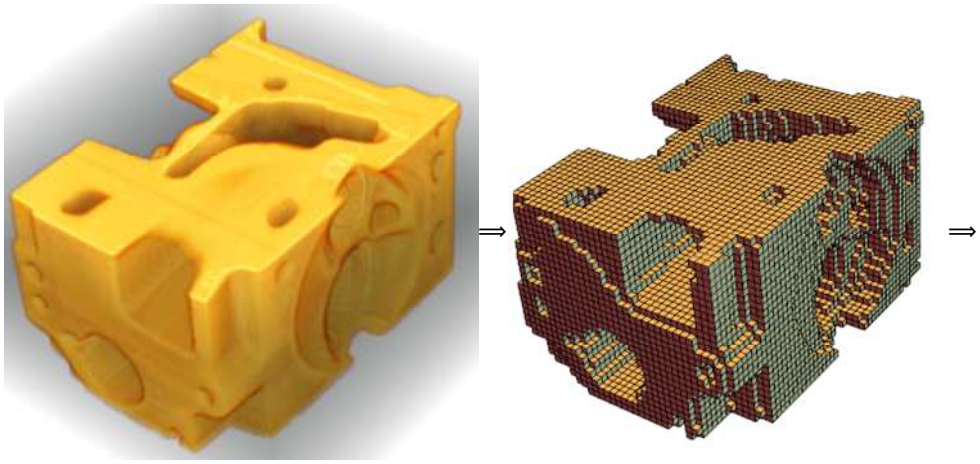
In[263]:=
```
ϕMax = π / 2 // N; ΔϕMin = ϕMax / 1000; Δϕ0 = ϕMax / 100; ΔϕMax = ϕMax / 30;
SMTNextStep["λ" → Δϕ0];
Mϕ = {{0, 0}};
UpperSurface = SMTFindNodes[ "Z" == H &];
Bp = SMTNodeData[UpperSurface, "Bp"];
While[
 ϕ = SMTRData["Multiplier"]; Δϕ = SMTRData["MultiplierIncrement"];
 Bt = Map[{- #[[1]] Cos[(Pi - ϕ) / 2] - #[[2]] Sin[(Pi - ϕ) / 2], #[[1]] Sin[(Pi - ϕ) / 2] -
       #[[2]] Cos[(Pi - ϕ) / 2], 0} 2 Sin[ϕ / 2] &, SMTNodeData[UpperSurface, "X"]];
 SMTNodeData[UpperSurface, "dB", (Bt - Bp) / Δϕ];
 While[step = SMTConvergence[10^-8, 16, {"Adaptive BC", 12, ΔϕMin, ΔϕMax, ϕMax}],
  SMTNewtonIteration[];];
 If[step⟦4⟧ === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
 If[Not[step⟦1⟧],
  Bp = Bt;
  M = Total[MapThread[Norm[Cross[#1, #2]] &, {SMTNodeData[UpperSurface, "X"] +
        SMTNodeData[UpperSurface, "at"], SMTResidual[UpperSurface]}]];
  AppendTo[Mϕ, {ϕ, M}];
  SMTShowMesh["DeformedMesh" → True, "Show" -> "Window"];
 ];
 step⟦3⟧
 , If[step⟦1⟧, SMTStepBack[];];
 SMTNextStep["Δλ" → step⟦2⟧];
]
```

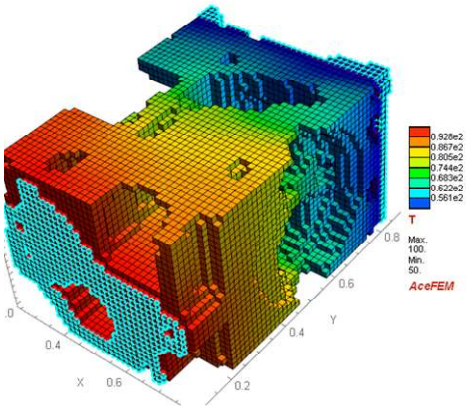
In[269]:=
```
SMTShowMesh["DeformedMesh" -> True, "Field" -> "Mises stress", "Mesh" -> False,
 "Combine" -> (GraphicsRow[{ListLinePlot[Mϕ, AxesLabel -> {"ϕ", "M"}], #}, ImageSize -> 600] &)]
```





Convert voxel image to 3D element mesh





A voxel represents a value on a regular grid in three-dimensional space. 3D image with pixel values given by the array (voxels) can be converted into 3D mesh composed of 3D hexahedral elements by the presented procedure.

- Here a 3D grayscale image is imported.

In[270]:=
```
image0 = Import["ExampleData/CTengine.tiff", "Image3D"]
```

In[272]:=
```
ImageColorSpace[image0]
```

```
Grayscale
```

- Original image is first resized to a desired mesh density.

In[273]:=
```
ImageDimensions[image0]
image1 = ImageResize[image0, Round[ImageDimensions[image0] / 1.5]];
```

```
{256, 256, 110}
```

- Image is here converted from 0-256 grayscale image to 0-1 bit image, where a threshold value of 50 is chosen for conversion.

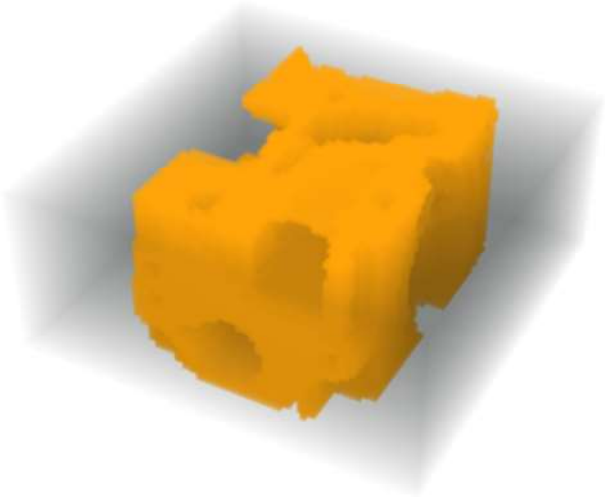
In[275]:=
```
treshhold = 50;
imagedata0 = Map[If[# > treshhold, 1, 0] &, ImageData[image1, "Byte"], {3}];
```

- Isolated cells are here deleted from the mesh.

In[277]:=
```
dim0 = imagedata0 // Dimensions;
imagedata1 = MapIndexed[
   If[# === 1
     , {i, j, k} = #2;
     If[If[i > 1, imagedata0[[i - 1, j, k]], 0] + If[i < dim0[[1]], imagedata0[[i + 1, j, k]], 0] +
        If[j > 1, imagedata0[[i, j - 1, k]], 0] + If[j < dim0[[2]], imagedata0[[i, j + 1, k]], 0] +
        If[k > 1, imagedata0[[i, j, k - 1]], 0] +
        If[k < dim0[[3]], imagedata0[[i, j, k + 1]], 0] > 0, 1, 0]
     , 0
    ] &, imagedata0, {3}];
```

- This is the final image that will be converted to the finite element mesh.

In[279]:=
```
Image3D[imagedata1, "Bit"]
```



- Voxel array is here converted to mesh nodes and element connectivity table.

In[280]:=
```
dim = imagedata1 // Dimensions;
nodesNeeded = Array[False &, dim + 1];
```

In[282]:=
```
elementsNeeded = MapIndexed[If[# === 1,
      nodesNeeded[[Sequence @@ (#2 + {0, 0, 0})]] = True;
      nodesNeeded[[Sequence @@ (#2 + {1, 0, 0})]] = True;
      nodesNeeded[[Sequence @@ (#2 + {1, 1, 0})]] = True;
      nodesNeeded[[Sequence @@ (#2 + {0, 1, 0})]] = True;
      nodesNeeded[[Sequence @@ (#2 + {0, 0, 1})]] = True;
      nodesNeeded[[Sequence @@ (#2 + {1, 0, 1})]] = True;
      nodesNeeded[[Sequence @@ (#2 + {1, 1, 1})]] = True;
      nodesNeeded[[Sequence @@ (#2 + {0, 1, 1})]] = True;
      True, False] &, imagedata1, {3}];
```

In[283]:=
```
i = 0; nodeNumbers = nodesNeeded /. {False → 0, True :→ (++i)};
```

In[284]:=
```
conectivity =
  Flatten[MapIndexed[If[#, Extract[nodeNumbers, {#2 + {0, 0, 0}, #2 + {1, 0, 0}, #2 + {1, 1, 0},
        #2 + {0, 1, 0}, #2 + {0, 0, 1}, #2 + {1, 0, 1}, #2 + {1, 1, 1},
        #2 + {0, 1, 1}}], Nothing] &, elementsNeeded, {3}], 2];
```

- Node coordinates are scaled in a way that the the largest dimension of the mesh bounding box is 1.

In[285]:=
```
maxdim = Max[dim];
dims = dim / maxdim // N;
```

In[287]:=
```
nodeX = Flatten[MapIndexed[If[#, {(#2[[3]] - 1) / maxdim,
        dims[[2]] - (#2[[2]] - 1) / maxdim, dims[[1]] - (#2[[1]] - 1) / maxdim} // N, Nothing] &,
    nodesNeeded, {3}], 2];
```

- Element mesh is generated here for the 3D thermal analysis of the engine part. The front part is heated to 100 degrees and the back part cooled to 50 degrees. Stationary temperature distribution within the engine is calculated and displayed.

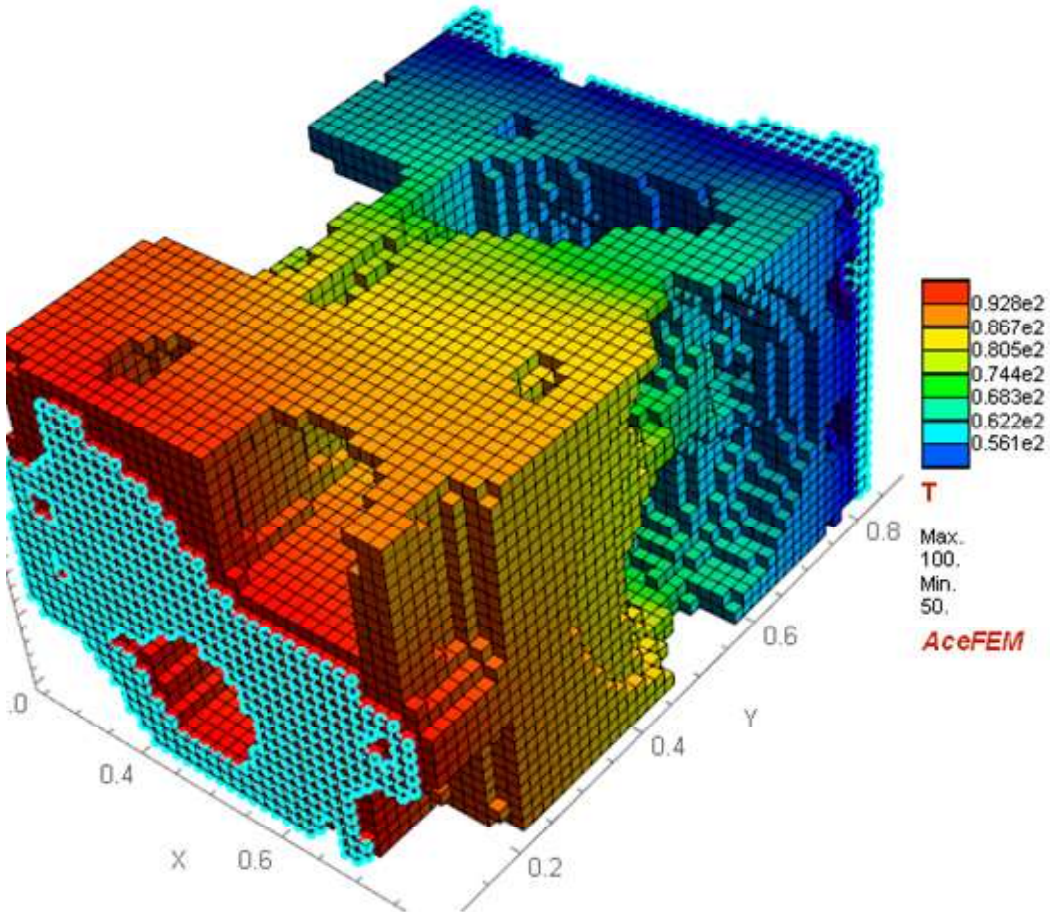
In[288]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["engine", {"ML:", "TH", "D3", "H1", "DF", "ST", "H1", "T", "Fourier"}, {}];
SMTAddMesh[nodeX, {"engine" -> conectivity}];
SMTAddEssentialBoundary["Y" <= 0.1 &, 1 → 100];
SMTAddEssentialBoundary["Y" > 0.8 &, 1 → 50];
SMTAnalysis[];
```

In[295]:=
```
SMTNextStep["λ" → 1];
SMTNewtonIteration[]
SMTNewtonIteration[]
```

```
77.604
```

$$2.70035 \times 10^{-11}$$

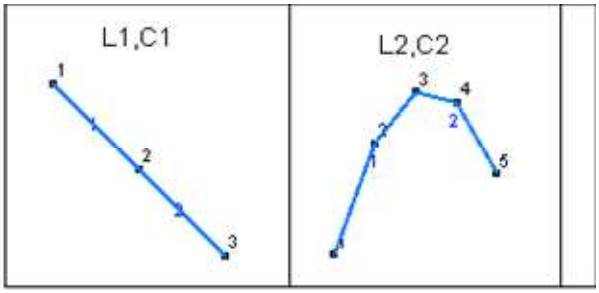
In[298]:=
```
SMTShowMesh[Axes → True, AxesLabel → {"X", "Y", "Z"}, "BoundaryConditions" → True, "Field" -> "T"]
```



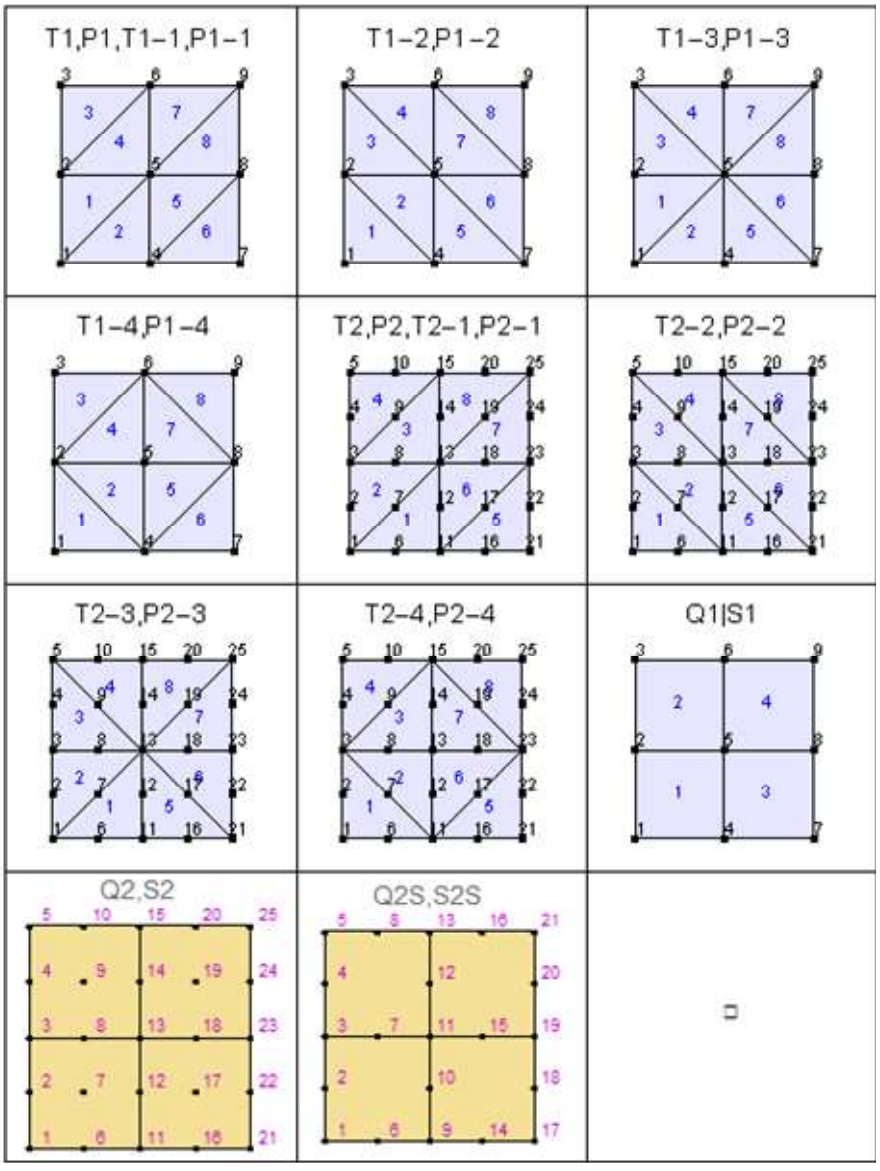
Types of Structured Meshes

- 1 D structured mesh types
- 2 D and 3 D surface structured mesh types
- 3 D mesh structured mesh types
- 2 D and 3 D surface structured mesh types with mesh refinement
- 3 D structured mesh types with mesh refinement

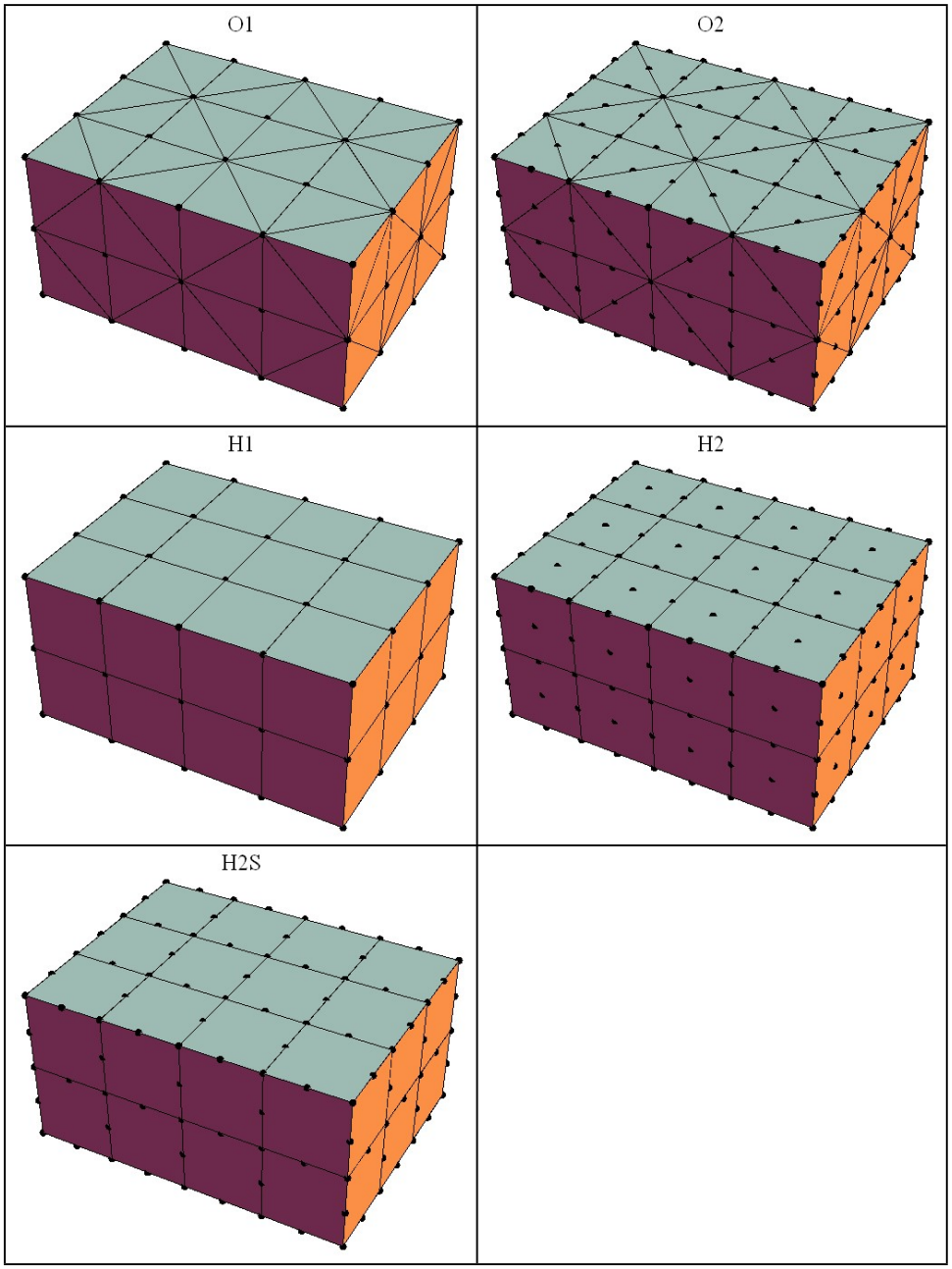
1D structured mesh types



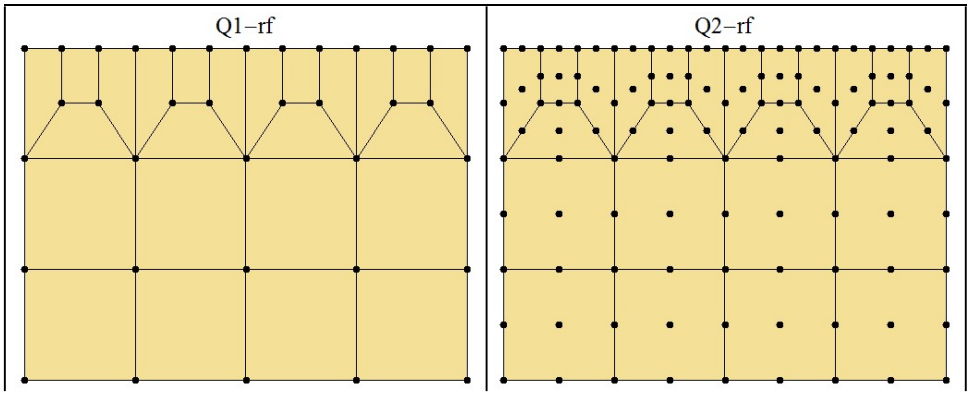
2D and 3D surface structured mesh types

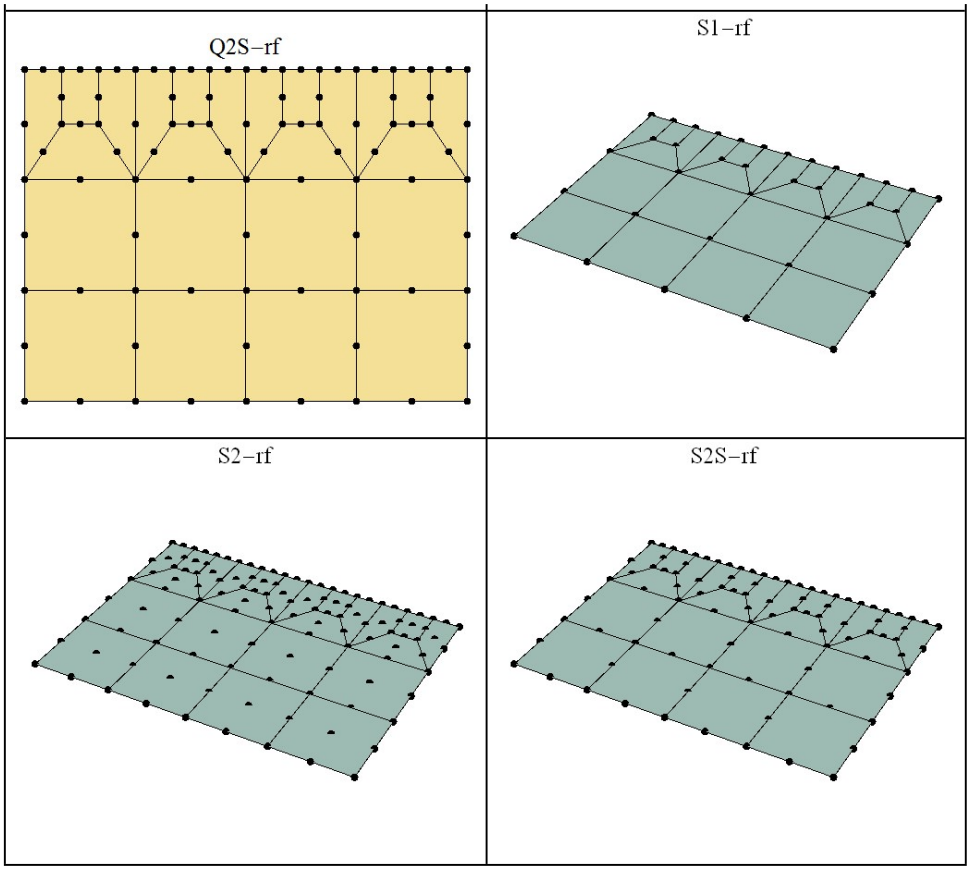


3D mesh structured mesh types



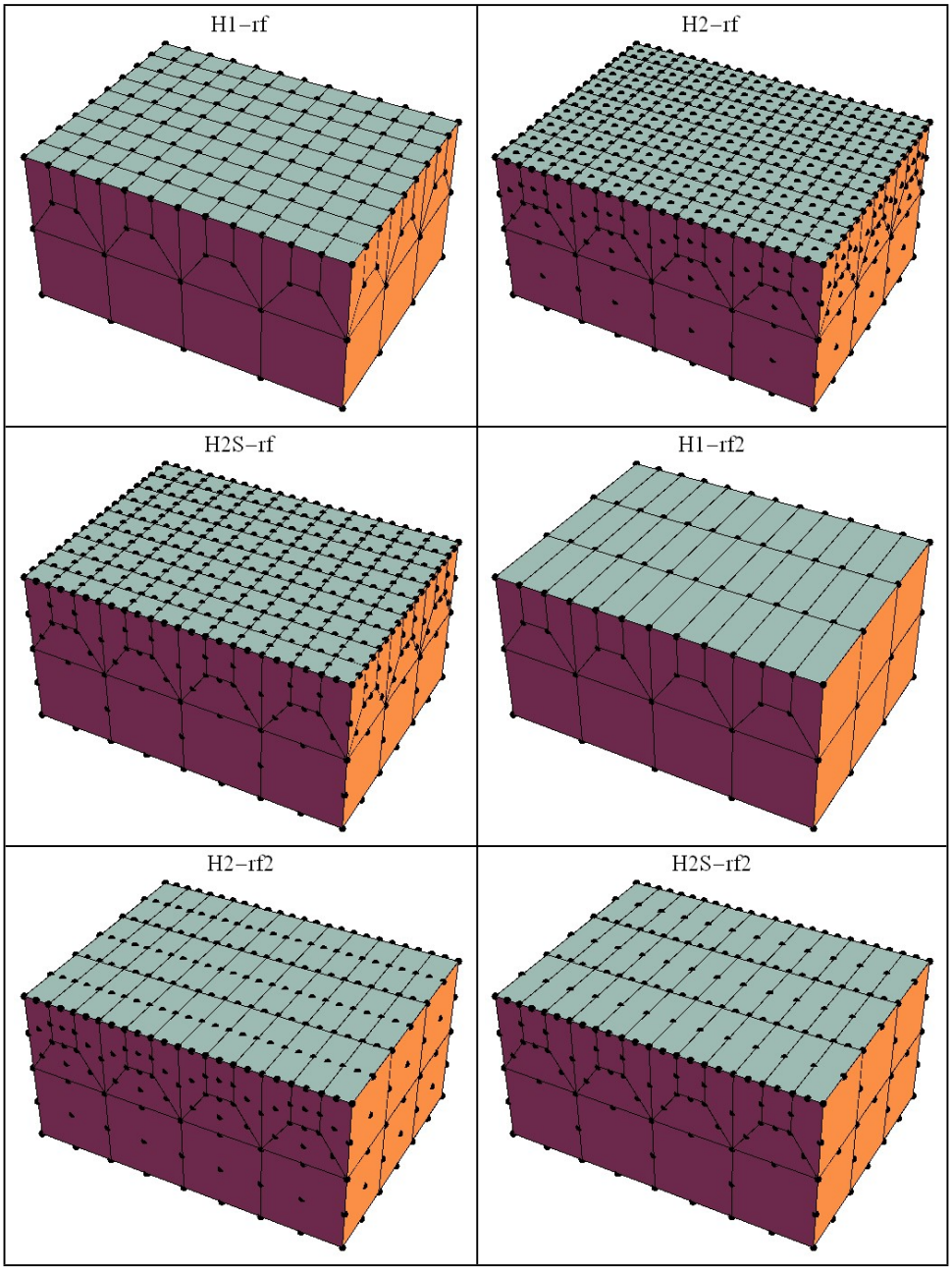
2D and 3D surface structured mesh types with mesh refinement



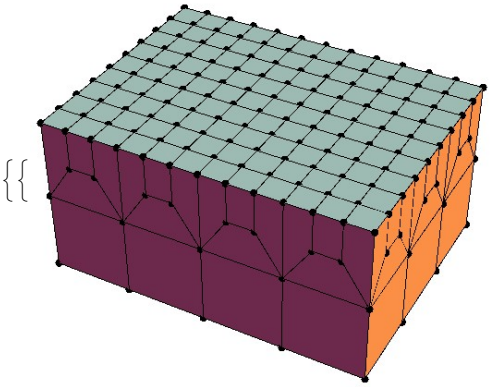


3D structured mesh types with mesh refinement

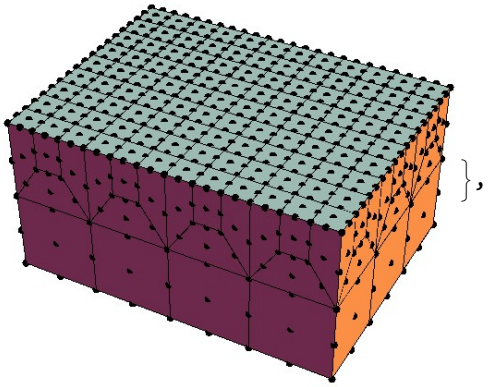
In[299]:=



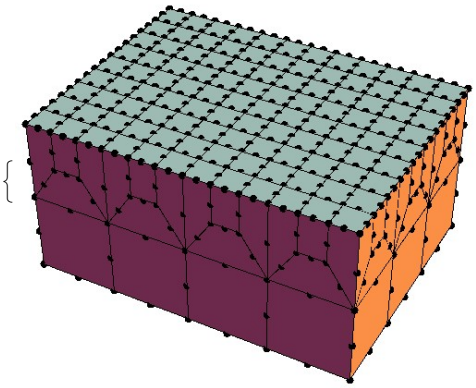
H1−rf



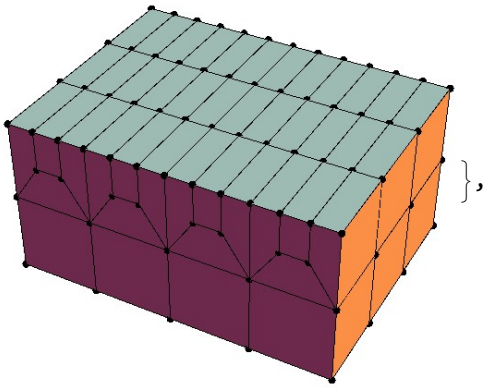
H2−rf



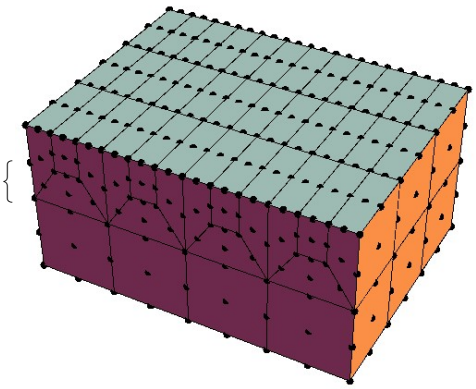
H2S−rf



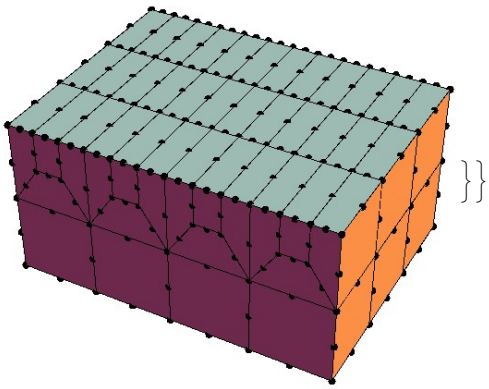
H1−rf2



H2−rf2



H2S−rf2



Selecting nodes, elements and bodies

Contents

Selecting Nodes

SMTFindNodes

SMTFindNodes[*nodeSelector*]
The parameter *nodeSelector* is a criteria used to select nodes. It has one of the forms described below.

nodeSelector	description
Point[T_1] or Point[{T_1,T_2,…,T_n}]	all nodes at given points {T_1,T_2,…,T_n}
Point[T_1,*NodeID*] or Point[{T_1,T_2,…,T_n},*NodeID*]	all nodes at given points and with node identification *NodeID*
Point[T_1,*NodeID*,*r*] or Point[{T_1,T_2,…,T_n},*NodeID*,*r*]	all nodes inside the circle (2 D case) or sphere (3 D case) with radius *r*, center T_i and node identification *NodeID*
Line[{T_1,T_2,…,T_n}, *NodeID*,*tolerance*]	all nodes within the distance less than *tolerance* from the line (2 D or 3 D) joining a sequence of points {T_1,T_2,…,T_n} and with the node identification *NodeID*
Line[{{T_1,T_2,…,T_n}, *NodeID*]	≡ Line[{T_1,T_2,…,T_n}, *NodeID*,Automatic]
Line[{T_1,T_2,…,T_n}]	≡ Line[{T_1,T_2,…,T_n}, All,Automatic]
Polygon[{T_1,T_2,…,T_n}, *NodeID*,*tolerance*]	all nodes inside the polygon {T_1,T_2,…,T_n} (2 D or 3 D) expanded for the *tolerance* and with node identification *NodeID* **In 3 D it is assumed that the polygon is planar with the plane defined by the first three points. FIRST THREE POINTS CANNOT BE COLINEAR!**
Polygon[{T_1,T_2,…,T_n}, *NodeID*]	≡ Polygon[{T_1,T_2,…,T_n}, *NodeID*,Automatic]
Polygon[{T_1,T_2,…,T_n}]	≡ Polygon[{T_1,T_2,…,T_n}, All,Automatic]
Tetrahedron[{T_1,T_2,T_3,T_4}, *NodeID*,*tolerance*]	all nodes inside the tetrahedron defined by four corner nodes {T_1,T_2,T_3,T_4} expanded for the *tolerance* and with node identification *NodeID*
Tetrahedron[{T_1,T_2,T_3,T_4}, *NodeID*]	≡ Tetrahedron[{T_1,T_2,T_3,T_4}, *NodeID*,Automatic]
Tetrahedron[{T_1,T_2,T_3,T_4}]	≡ Tetrahedron[{T_1,T_2,T_3,T_4}, All,Automatic]
Hexahedron[{T_1,T_2,T_3,T_4,T_5,T_6,T_7,T_8}, *NodeID*,*tolerance*]	all nodes inside the hexahedron defined by 8 corner nodes {T_1,T_2,T_3,T_4,T_5,T_6,T_7,T_8} expanded for the *tolerance* and with node identification *NodeID* **Only regular hexahedron (Cube) is supported!** If the given points are not vertices of a cube than the bounding box is used instead.
Hexahedron[{T_1,T_2,T_3,T_4,T_5,T_6,T_7,T_8}, *NodeID*]	≡ Hexahedron[{T_1,T_2,T_3,T_4,T_5,T_6,T_7,T_8}, *NodeID*,Automatic]
Hexahedron[{T_1,T_2,T_3,T_4,T_5,T_6,T_7,T_8}]	≡ Hexahedron[{T_1,T_2,T_3,T_4,T_5,T_6,T_7,T_8}, All,Automatic]

Nodes selector based on geometric regions.

nodeSelector	description
{"NodeID",*nodeIDSelector*}	all nodes with node identification *NodeID* (see Node Identification)
{"DomainID",*domainIDSelector*}	all nodes that a part of elements from domains defined by *domainIDSelector* (see `Selecting Domains`)
{"BodyID",*bodyIDSelector*}	all nodes that a part of elements from bodies defined by *bodyIDSelector*
{"BoundaryNodes", *domainIDSelector*, *bodyIDSelector*,*nodeIDSelector*}	all nodes that form the boundary of the selected bodies or the boundary between the subregions of the selected bodies with selected node identification
{"BoundaryNodes"}	≡ {"BoundaryNodes",All,All,All}
nodeID_String	≡ {"NodeID",n*odeID*}

Nodes selector based on mesh regions.

nodeSelector	description
crit_Function	nodes for which test $crit[x_i,y_i,z_i,nID_i]$ yields True
$\{in_1,in_2,\ldots in_N\}$	nodes with the node indices $in_1,in_2,\ldots in_N$ (note that node index can be changed due to the "Tie" command after the *SMTAnalysis* command)
i_Integer	≡ {i}
All	≡ $\{1,2,\ldots,n_{tn}\}$

Alternative ways of selecting nodes.

nodeSelector	description
nodeSelector$_1$ && *nodeSelector*$_2$ …	gives a sorted list of the node indexes that match all the given criteria
nodeSelector$_1$ || *nodeSelector*$_2$ …	gives a sorted list of all the distinct node indexes that match any of the given criteria

Boolean type expressions involving And, Or Boolean operators and arbitrary *nodeSelector-s*.

Geometric regions Point, Line and Polygon accept two-dimensional and three-dimensional points.

The parameter *crit* is a pure function applied to each node in turn. Nodes for which test function *crit* returns True are selected. The standard *Mathematica* symbols for the formal parameters of the pure function (#1,#2,#3,..) can be replaced by the strings representing coordinates "X","Y","Z", and the node identification with string "ID" (see `Node Identification`).

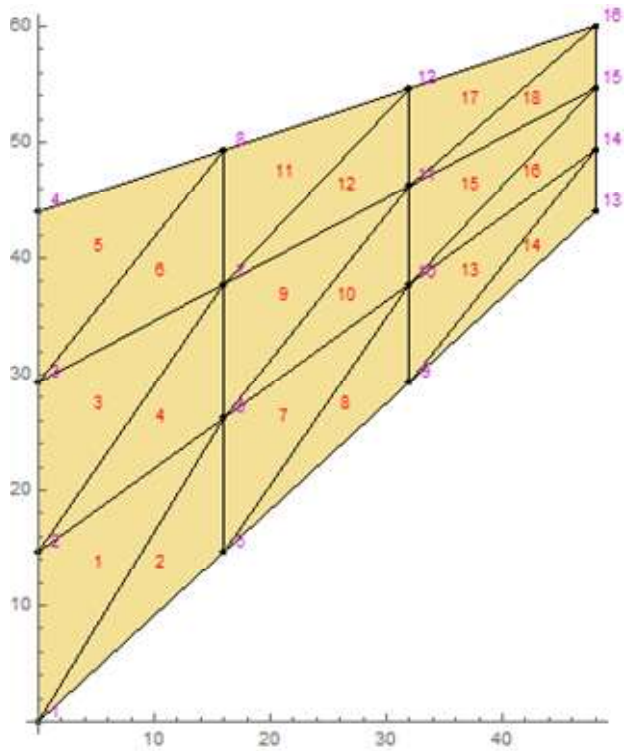
The Boolean type expressions can be nested.

Examples: Selecting nodes

- SMTFindNodes[Polygon[{{0,0},{1,0},{0,1}}]] returns a list of indexes of all nodes inside the triangle {{0,0},{1,0},{0,1}}.
- SMTFindNodes["Temp"] returns a list of indexes of all nodes with node identification "Temp".
- SMTFindNodes["X"<5 && "Z">2 &] returns a list of indexes of all nodes in region "X"<5 && "Z">2.
- SMTFindNodes[{1,2,200}] returns {1,2,200}.
- SMTFindNodes[(Line[{{0,0},{10,0}}] || Line[{{10,0},{10,5}}]) && ("Y"<2.5&)] returns a list of indexes of all nodes that lies either on line {{0,0},{10,0}} or on line {{10,0},{10,5}} and with Y coordinate less than 2.5

In[300]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[
  {"Test", {"ML:", "SE", "PE", "T1", "DF", "LE", "T1", "D", "Hooke"}, {"E *" -> 1, "ν *" -> 0}}];
SMTAddMesh[Polygon[{{0, 0}, {48, 44}, {48, 44 + 16}, {0, 44}}], "Test", "T1", {3, 3}];
SMTAddEssentialBoundary[ "X" == 0 &, 1 → 0, 2 → 0];
SMTAddNaturalBoundary[ "X" == 48 &, 2 → -.1];
SMTAnalysis[];
```

In[307]:=
```
SMTShowMesh["Marks" → True, Axes → True]
```



In[308]:=
```
SMTFindNodes["D"]
```
Out[308]=
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
```

In[309]:=
```
SMTFindNodes[Point[{48, 44}]]
```
Out[309]=
```
{13}
```

In[310]:=
```
SMTFindNodes[Point[{48, 44}, All, 20]]
```
Out[310]=
```
{10, 11, 12, 13, 14, 15, 16}
```

In[311]:=
```
SMTFindNodes[Line[{{0, 0}, {0, 50}}] || Line[{{0, 0}, {48, 44}}]]
```
Out[311]=
```
{1, 2, 3, 4, 5, 9, 13}
```

In[312]:=
```
SMTFindNodes[{"DomainID", "Test"}]
```
Out[312]=
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
```

In[313]:=
```
SMTFindNodes[{"NodeID", "D"}]
```
Out[313]=
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
```

In[314]:=
```
SMTFindNodes[{"BoundaryNodes"}]
```
Out[314]=
```
{1, 2, 3, 4, 5, 8, 9, 12, 13, 14, 15, 16}
```

In[315]:=
```
SMTFindNodes[{"BoundaryNodes", All, All, "D"}]
```
Out[315]=
```
{1, 2, 3, 4, 5, 8, 9, 12, 13, 14, 15, 16}
```

In[316]:=
```
SMTFindNodes[Polygon[{{0, 0}, {48, 44}, {0, 44}}]]
```
Out[316]=
```
{1, 2, 3, 4, 5, 6, 7, 9, 10, 13}
```

In[317]:=
```
SMTFindNodes["Y" > 20 &]
```
Out[317]=
```
{3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
```

Selecting Elements

SMTFindElements

SMTFindElements[*elementSelector*]
The parameter *elementSelector* is used to select elements. It has one of the forms described below.

elementSelector	description
{*nodeSelector, domainSelector*}	The parameter *nodeSelector* is first used to select nodes. It has one of the forms described in section `Selecting Nodes`. The parameter *domainSelector* is then used to select relevant domains or element types. It has one of the forms described in section `Selecting Domains`. After that the elements are selected with all nodes within the selected nodes and the domain identification within the selected domain identifications.
{*nodeSelector, domainSelector, bodyIDSelector*}	Find all elements that satisfy all three criteria *nodeSelector, domainSelector, bodyIDSelector.* (*see* Selecting Nodes, `Selecting Domains`, Selecting Bodies)
{"BoundaryElements", *domainIDSelector, bodyIDSelector*}	all segments that form the boundary of the selected bodies or the boundary between the subregions of the selected bodies and with domain identification *dID*
ncrit_Function	selection of elements with the nodes for which test $ncrit[x_i, y_i, z_i, nID_i]$ yields True
dID_String	≡ {All,*dID*} all elements with domain identification *dID*
$\{ie_1, ie_2, \ldots ie_N\}$	elements with the element indices $ie_1, ie_2, \ldots ie_N$
i_Integer	≡ {*i*}
All	≡ $\{1,2,\ldots,n_e\}$

Forms of input data for selecting elements.

Many functions require as input a list of elements on which certain action is applied. The parameter that is used to select elements can have various forms described above.

The parameter *ncrit* is a pure function applied to all nodes of the element in turn. Elements for which all nodes return True are selected. The standard *Mathematica* symbols for the formal parameters of the pure function (#1,#2,#3,..) can be replaced by the strings representing coordinates "X","Y","Z", and the node identification "ID".

The parameter *dcrit* is a pure function applied to all domain identifications in turn. Domains for which test *dcrit*[*dID*] yields True are selected.

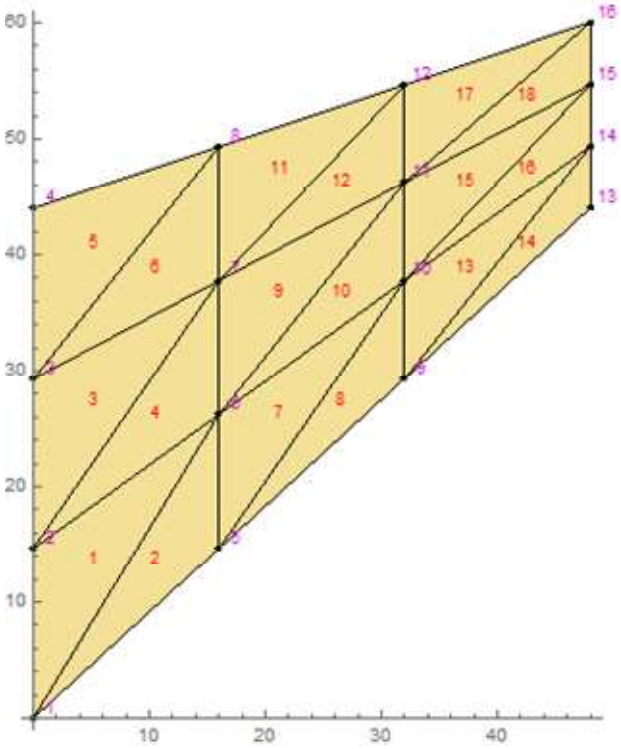
Examples:

- SMTFindElements[{Polygon[{{0,0},{1,0},{0,1}}], All}] returns a list of indexes of all elements inside the triangle {{0,0},{1,0},{0,1}}.
- SMTFindElements["bottom"] returns a list of indexes of all elements with domain identification "bottom".
- SMTFindElements[{"X"<5 && "Z">2 &,"bottom"}] returns a list of indexes of all elements in region "X"<5 && "Z">2 and with the domain identification "bottom".

Examples: Selecting elements

In[318]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[
  {"Test", {"ML:", "SE", "PE", "T1", "DF", "LE", "T1", "D", "Hooke"}, {"E *" -> 1, "ν *" -> 0}}];
SMTAddMesh[Polygon[{{0, 0}, {48, 44}, {48, 44 + 16}, {0, 44}}], "Test", "T1", {3, 3}];
SMTAddEssentialBoundary[ "X" == 0 &, 1 → 0, 2 → 0];
SMTAddNaturalBoundary[ "X" == 48 &, 2 → -.1];
SMTAnalysis[];
```

In[325]:=
```
SMTShowMesh["Marks" → True, Axes → True]
```



In[326]:=
```
SMTFindElements["Test"]
```

Out[326]=
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18}
```

In[327]:=
```
SMTFindElements[{Polygon[{{0, 0}, {48, 44}, {0, 44}}], All}]
```

Out[327]=
```
{1, 2, 3, 4, 7, 8}
```

In[328]:=
```
SMTFindElements[{"Y" > 20 &, "Test"}]
```

Out[328]=
```
{5, 6, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18}
```

In[329]:=
```
SMTFindElements[{"BoundaryElements", "Test", All}]
```

Out[329]=
```
{{2, 1}, {1, 5}, {3, 2}, {8, 4}, {4, 3}, {5, 9},
 {12, 8}, {9, 13}, {13, 14}, {14, 15}, {16, 12}, {15, 16}}
```

Selecting Domains

SMTFindDomains

SMTFindDomains[*domainSelector*]

The parameter *domainSelector* is used to select domains. It has one of the forms described below. The functions returns a list of indexes of selected domains.

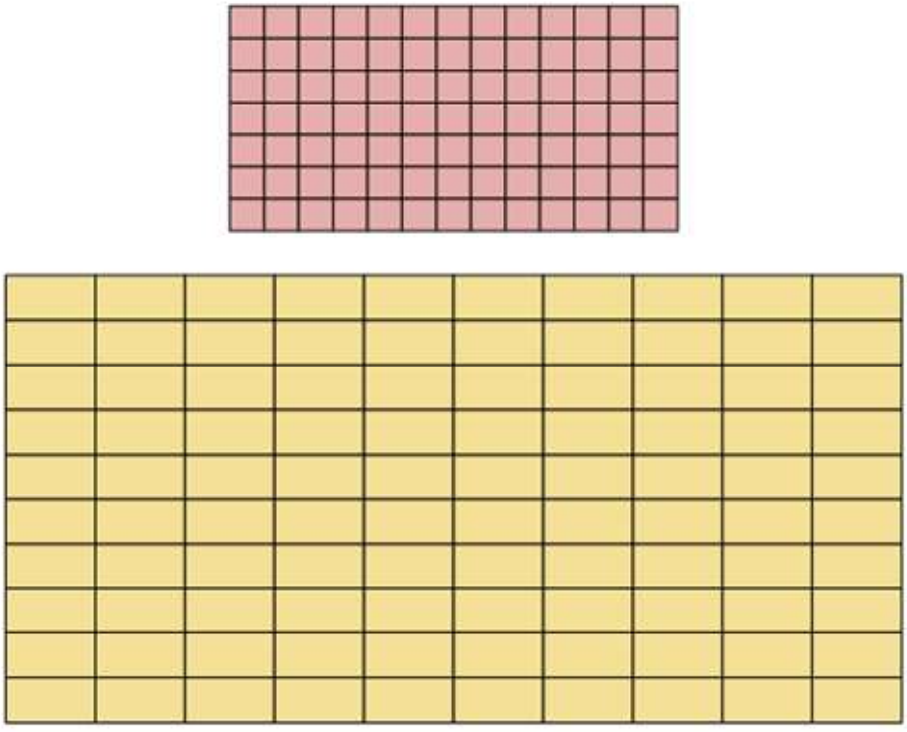
domainSelector	description
dID_String	domain with identification *dID*
{dID_1,dID_2, ...}	set of domains with given identifications dID_i
dcrit_Function	all domains for which test *dcrit*[dID_i] yields True
All	all domains
{id_1,id_2, ...}	domains with the indices id_1,id_2, ...

Input data for selecting domains.

Many functions require as input a list of domains on which certain action is applied. The parameter that is used to select domains can have various forms described above.

Examples: Selecting domains

In[330]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[
   {"Solid1", "ExamplesHypersolid2D", {"E *" -> 70000, "ν *" -> 0.3}},
  {"Solid2", "ExamplesHypersolid2D", {"E *" -> 210000, "ν *" -> 0.3}},
  {"Contact", "ExamplesCTD2N1L1Pen", {"ρ *" -> 200000}}];
SMTAddMesh[Polygon[{{-1/2, 0.1}, {1/2, 0.1}, {1/2, 0.6}, {-1/2, 0.6}}], "Solid1",
  "Q1", {13, 7}, "BodyID" → "B1", "BoundaryDomainID" -> "Contact"];
SMTAddMesh[Polygon[{{-1, -1}, {1, -1}, {1, 0}, {-1, 0}}], "Solid2",
  "Q1", {10, 10}, "BodyID" → "B2", "BoundaryDomainID" -> "Contact"];
SMTAddEssentialBoundary[{"Y" == -1 &, 1 -> 0, 2 -> 0}, {"Y" == 0.6 &, 1 -> 0, 2 -> -1}];
SMTAnalysis[];
SMTShowMesh[]
```



In[338]:=
```
SMTFindDomains[All]
```
Out[338]=
```
{1, 2, 3}
```

In[339]:=
```
SMTFindDomains[{"Contact"}]
```
Out[339]=
```
{3}
```

In[340]:=
```
SMTFindDomains[{1, 3}]
```
Out[340]=
```
{1, 3}
```

In[341]:=
```
SMTFindDomains[StringMatchQ[#, "Solid*"] &]
```
Out[341]=
```
{1, 2}
```

Selecting Bodies

SMTFindBodies

SMTFindBodies[*bodySelector*]
The parameter *bodySelector* is used to select bodies. It has one of the forms described below. The functions returns a list of indexes of selected bodies.

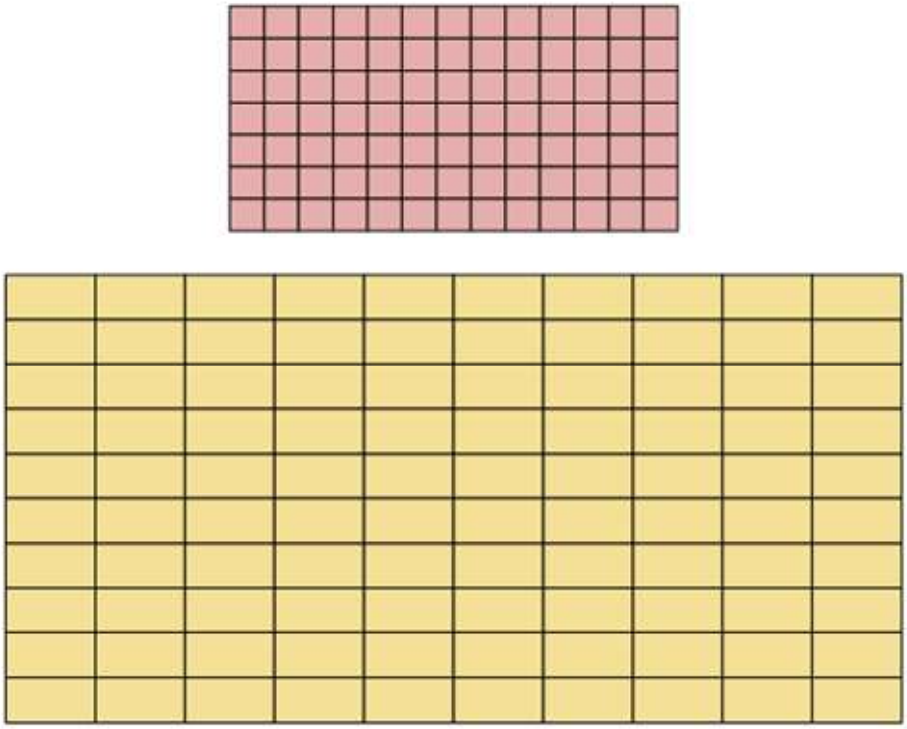
bodySelector	description
bID_String	body with identification *bID*
{bID_1,*bID*$_2$, …}	set of bodies with given identifications bID_i
dcrit_Function	all bodies for which test *dcrit*[bID_i] yields True
All	all bodies
{ib_1,ib_2, …}	bodies with the indices ib_1,ib_2, …

Input data for selecting bodies.

Many functions require as input a list of bodies on which certain action is applied. The parameter that is used to select bodies can have various forms described above.

Examples: Selecting bodies

In[342]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[
   {"Solid1", "ExamplesHypersolid2D", {"E *" -> 70000, "ν *" -> 0.3}},
   {"Solid2", "ExamplesHypersolid2D", {"E *" -> 210000, "ν *" -> 0.3}},
   {"Contact", "ExamplesCTD2N1L1Pen", {"ρ *" -> 200000}}];
SMTAddMesh[Polygon[{{-1/2, 0.1}, {1/2, 0.1}, {1/2, 0.6}, {-1/2, 0.6}}], "Solid1",
  "Q1", {13, 7}, "BodyID" → "B1", "BoundaryDomainID" -> "Contact"];
SMTAddMesh[Polygon[{{-1, -1}, {1, -1}, {1, 0}, {-1, 0}}], "Solid2",
  "Q1", {10, 10}, "BodyID" → "B2", "BoundaryDomainID" -> "Contact"];
SMTAddEssentialBoundary[{"Y" == -1 &, 1 -> 0, 2 -> 0}, {"Y" == 0.6 &, 1 -> 0, 2 -> -1}];
SMTAnalysis[];
SMTShowMesh[]
```



In[350]:=
```
SMTFindBodies[All]
```

Out[350]=
```
{1, 2}
```

In[351]:=
```
SMTFindBodies["B2"]
```

Out[351]=
```
{2}
```

In[352]:=
```
SMTFindBodies[{2, 1}]
```

Out[352]=
```
{2, 1}
```

In[353]:=
```
SMTFindBodies[StringMatchQ[#, "B*"] &]
```

Out[353]=
```
{1, 2}
```

Boundary Conditions

Contents

Introduction

In AceFEM one can directly apply two types of boundary conditions:

- Essential (Dirichlet) boundary conditions (SMTAddEssentialBoundary)
  Effectively the values of selected degree of freedom in selected node takes prescribed value.
- Natural (Neumann) boundary conditions (SMTAddNaturalBoundary)
  Effectively the prescribed value of the boundary condition is subtracted from the element of the global residual vector that corresponds to the selected degree of freedom in selected node.

The current value of the boundary condition (**B**t) is defined as $\mathbf{B}t = \mathbf{B}p + \Delta\lambda\, d\mathbf{B}$, where $\Delta\lambda$ is the boundary conditions multiplier increment (see Iterative Solution Procedures). AceFEM generally deals with the nonlinear problem and uses various path following procedures to solve it. By default, the state of the system at the beginning of the path-following procedure is zero state. Alternatively, an initial nonzero state of either essential or natural boundary conditions can be set by SMTAddInitialBoundary.

Boundary conditions are prescribed as a part of the input data phase. At the input data phase all the data related to the prescribed boundary conditions is collected and then applied in the following order:

- first the initial boundary conditions are applied,
- then the natural boundary conditions are applied,
- at the end the essential boundary conditions are applied.

Thus when both the essential and the natural boundary conditions are prescribed for the same DOF, then the precedence goes to the essential boundary conditions. Initial boundary condition is by default natural boundary condition.

Boundary conditions can be applied as a part of input data (before SMTAnalysis) or after the SMTAnalysis at the analysis phase. At the analysis phase the boundary conditions commands are executed immediately. In order to apply natural boundary condition at analysis phase, the eventual essential boundary condition have to be removed first with the SMTAddEssentialBoundary[*nodes*, *dof* → Null] command.

Essential Boundary Conditions

SMTAddEssentialBoundary[*nodeSelector*, $dof_1 \to dB_1$, $dof_2 \to dB_2$, ...]
increment or set reference value of essential (Dirichlet) boundary condition in selected nodes (see Selecting Nodes) where dB_i is the reference value of the intensity of essential boundary condition (support) for the dof_i-th unknown

intensity of essential boundary condition	description
dB_Number	value *dB* is set to all nodes that match *nodeSelector*
f_Function	The parameter $f[n,X,Y,Z]$ is a function applied to each node that match *nodeSelector* in turn where n is a node number. (e.g. 2→Function[{n,X,Y,Z},10 Y] would apply boundary condition with the value proportional to the Y coordinate of the node to second DOF in all selected nodes).
$\{dB_1,dB_2,\ldots,dB_N\}$	value dB_i is set to the i−th node that match *nodeSelector*
Null	remove the prescribed essential boundary condition if set by previous definitions (unconstrained DOF)

Forms of intensity of prescribed essential boundary conditions.

options	default	description
"Set"	False	override the previous defined boundary conditions for the chosen degrees of freedom with the newly defined values

Options for prescribed essential boundary conditions.

The value of boundary condition is by default incremented by the given value. With the "Set"->True option the new value overrides all the previous definitions. With the dof_i→Null input form all the previously defined boundary conditions are deleted for the dof_i-th degree of freedom.

The current value of the boundary condition (**B**t) is defined as $\mathbf{B}t = \mathbf{B}p + \Delta\lambda\, d\mathbf{B}$, where $\Delta\lambda$ is the boundary conditions multiplier increment (see `Iterative Solution Procedures`).

The *nodeSelector* parameter is defined in `Selecting Nodes`.

Examples:

- The third degree of freedom in all the nodes on line segment {(0,0),(1,1)} and with node identification "D" is incremented by 1.5.

```
In[354]:= SMTAddEssentialBoundary[Line[{{0, 0}, {1, 1}}, "D"], 3 → 1.5]
```

- The third degree of freedom in all the nodes on line segment {(0,0),(1,1)} and with node identification "D" is incremented by *10 x+2y*.

```
SMTAddEssentialBoundary[Line[{{0, 0}, {1, 1}}, "D"], 3 → Function[{n, X, Y, Z}, 10 X + 2 Y]]
```

- All the nodes on line segment {(0,0),(1,1)} and with node identification "D" get a prescribed value 1.5 for the third degree of freedom.

```
SMTAddEssentialBoundary[Line[{{0, 0}, {1, 1}}, "D"], 3 → 1.5, "Set" -> True]
```

- Remove the prescribed essential boundary condition for the third degree of freedom for all the nodes on line segment {(0,0),(1,1)} and with node identification "D".

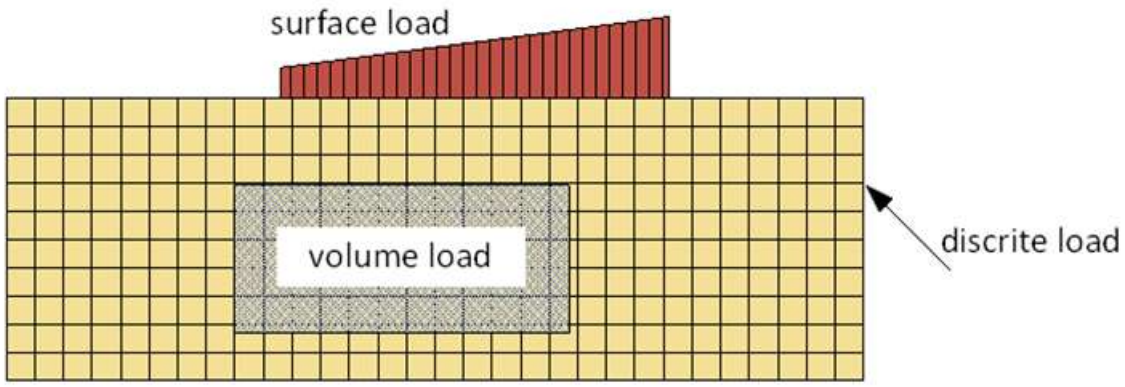
```
SMTAddEssentialBoundary[Line[{{0, 0}, {1, 1}}, "D"], 3 → Null]
```

Natural Boundary Conditions

Prescribed natural boundary conditions are of three types:

- discrete natural boundary condition prescribed in nodes of the mesh (concentrated load), see `Discrete Natural Boundary Conditions`
- distributed natural boundary condition prescribed on the volume of the domain (volume load), see `Distributed Natural Boundary Defined by Nodes` or `Distributed Natural Boundary Defined by Elements`

- distributed natural boundary condition prescribed on the surface of the domain (surface load), see `Distributed Natural Boundary Defined by Nodes`



In the case of distributed natural boundary conditions (surface or volume) the distributed load has to be transformed into the equivalent discrete loads applied in corresponding nodes of the mesh. Actual nodal values are calculated on the assumption of a standard izoparametric interpolation of all fields as

$$dB_i = \int_\Omega db\,(\mathbf{X})\,N_i\,(\mathbf{X})\,d\mathrm{x}$$

where N_i is a standard izoparametric shape function that belongs to the *i*-th node and *db*(**X**) is the intensity of the prescribed natural boundary condition as a function of the spatial coordinate **X.** The distributed natural boundary conditions can be applied on two ways depending on the definition of integration domain Ω. The integration domain Ω can be specified as:

- Ω is the volume (surface or line) of the actual selected elements (see `Distributed Natural Boundary Defined by Elements`)
- Ω is a volume (surface or line) of elements temporarily created from the selected nodes (`Distributed Natural Boundary Defined by Nodes`)

If the surface of the domain is created from the selected nodes then it is divided into triangular or quadrilateral segments regardless on topology of the underlying elements. The results is not exact for the higher order elements.

The topology of the underlying elements is exactly accounted for when the distributed natural boundary condition is applied on elements directly (`Distributed Natural Boundary Defined by Elements`).

Natural boundary conditions are prescribed by the SMTAddNaturalBoundary command.

SMTAddNaturalBoundary[*selectedNodesOrElements*, dof_1->$ints_1$, $ints_2$->$ints_2$,...]
increment or set reference value of natural (Neumann) boundary condition in selected nodes or elements (`Selecting Nodes, Selecting Elements`) where the parameters $ints_i$ is used to calculate the reference value of intensity of natural boundary condition for the dof_i-th nodal unknown

The actual form of parameters *selectedNodesOrElements* and $ints_i$ depends on the type of prescribed boundary condition and is given on examples throughout this chapter. The current value of the boundary condition (**B**t) is defined as $\boldsymbol{B}t = \boldsymbol{B}p + \Delta\lambda\, d\boldsymbol{B}$, where Δλ is the boundary conditions multiplier increment (see `Iterative Solution Procedures`). The total reference value of the boundary condition is for specific DOF in specific node for each SMTAddNaturalBoundary call incremented by the given value. With the "Set"->True option the new value overrides all the previous definitions.

option	default	description
"Set"	False	False ⇒ the total reference value of the boundary condition is for specific DOF in specific node for each SMTAddNaturalBoundary call incremented by the given value True ⇒ override the previous defined boundary conditions for the chosen degrees of freedom with the newly defined values

Options for the SMTAddNaturalBoundary command.

Discrete Natural Boundary Conditions

SMTAddNaturalBoundary[*nodeSelector*, dof_1->dB_1, dof_2->dB_2,...]
increment or set reference value of natural (Neumann) boundary condition in selected nodes (see `Selecting Nodes`) where dB_i is the reference value of intensity of discrete natural boundary condition (force) for the dof_i-th unknown as described in table below

Applying discrete natural boundary conditions.

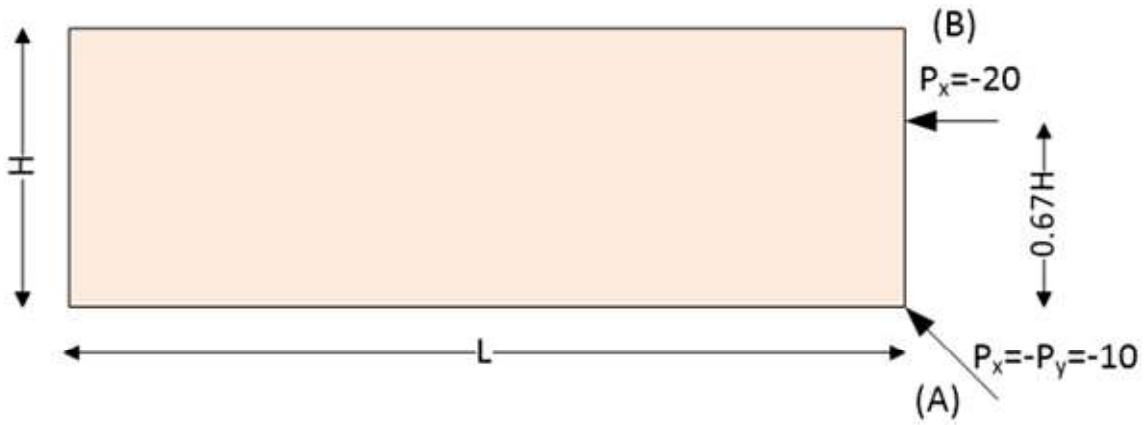
dB_i reference boundary condition intensity	description
dB_Number	value *dB* is set to all nodes that match *nodeSelector*
f_Function	The parameter $f[n,X,Y,Z]$ is a function applied to each node that match *nodeSelector* in turn where *n* is a node number. (e.g. 2→Function[{n,X,Y,Z},10 Y]).
$\{dB_1, dB_2, \ldots\}$	value dB_i is set to the i–th node that match *nodeSelector*
Point[{*arc_length*,*dB*}]	This form assumes that the nodes that match *nodeSelector* form line or curve segment. The discrite boundary condition with the intensity *dB* is applied at the given arc length distance from the start of the curve. Nodal values in effected nodes are then calculated on the assumption of a standard izoparametric interpolation of all fields as $dB_i = dB\, N_i(\mathbf{X})$ where N_i is a standard izoparametric shape function that belongs to the i–th node.
Null	remove the coresponding prescribed natural boundary conditions if set by previous definitions

Forms of intensity of prescribed discrete natural boundary conditions.

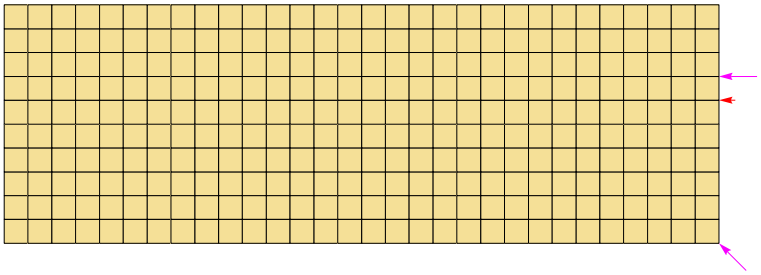
The *nodeSelector* parameter is defined in `Selecting Nodes`.

Examples:

- Generic example:
  - (A) concentrated load applied on the selected nodes
  - (B) concentrated load applied on an arbitrary spatial point



In[355]:=
```
<< AceFEM`;
SMTInputData[];
L = 9; H = 3;
SMTAddDomain["Ω", {"ML:", "SE", "PE", "Q1", "DF", "HY", "Q1", "D", {{"NeoHooke", "WA"}}}, {}];
SMTAddMesh[Polygon[{{0, 0}, {L, 0}, {L, H}, {0, H}}], "Ω", "Q1", {30, 10}];
(*A*)SMTAddNaturalBoundary[Line[{{L, 0}, {L, H}}], 1 -> Point[{0.67 H, -20}]];
(*B*)SMTAddNaturalBoundary[Point[{L, 0}], 1 → -10, 2 -> 10];
SMTAnalysis[];
SMTShowMesh["BoundaryConditions" → True]
```



- This would apply boundary condition with the intensity proportional to the Y coordinate of the node to second DOF in all selected nodes.

In[364]:=
```
SMTAddNaturalBoundary[Line[{{L, 0}, {L, H}}], 2 -> Function[{n, X, Y, Z}, 10 Y]]
```

Out[364]= 11

Distributed Natural Boundary Defined by Nodes

SMTAddNaturalBoundary[*nodeSelector*, dof_1->db_1, dof_2->db_2,...]

increment or set reference value of natural (Neumann) boundary condition in selected nodes (see `Selecting Nodes`) where db_i is the reference value of distributed natural boundary condition (force) for the dof_i-th unknown as described below

Applying distributed natural boundary conditions on volume (surface, line) defined by nodes.

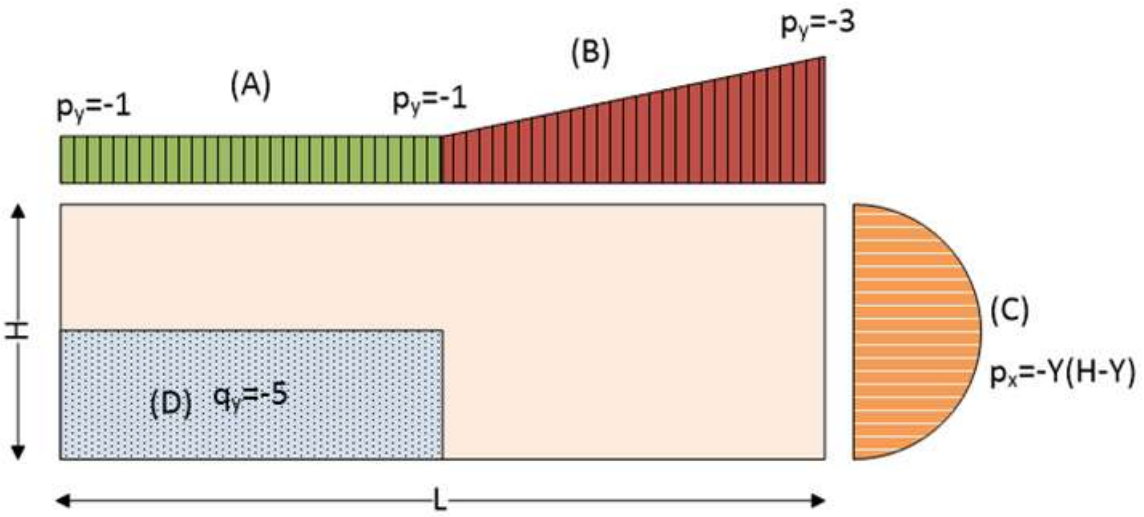
boundary condition intensity	description
Line[{*db*}]	This form assumes that the nodes that match *nodeSelector* form line or curve segment and that the uniformly distributed boundary condition with the intensity *db* is prescribed on the segment. The result is approximate for the elements with higher than quadratic approximation.
Line[{db_0, db_1}]	Form defines lineary distributed natural boundary condition. db_0 is intensity of distributed boundary condition in the first node that match *nodeSelector* and db_1 is intensity of distributed boundary condition in the last node that match *nodeSelector*. The result is approximate for the elements with higher than quadratic approximation.
Line[*f_Function*]	This form assumes that the nodes that match *nodeSelector* form line or curve segment and that the distributed boundary condition is prescribed on a segments of the curve. Parameter f is a function used to evaluate the intensity of the distributed boundary condition accordingly to the topology of the segments as follows: "L1" $\Rightarrow db=f\left[X,Y,t_x,t_y\right]$ "L2" $\Rightarrow db=f\left[X,Y,t_x,t_y\right]$ "C1" $\Rightarrow db=f\left[X,Y,Z,t_x,t_y,t_z\right]$ "C2" $\Rightarrow db=f\left[X,Y,Z,t_x,t_y,t_z\right]$ Vector $\left(t_x,t_y,t_z\right)$ is unit tangent vector in the direction of the curve in point (X,Y).
Polygon[{*db*}]	This form assumes that the nodes that match *nodeSelector* form a surface (2 D or 3 D) and that the uniformly distributed boundary condition with the intensity *db is* prescribed on the surface. The surface is first divided into triangular or quadrilateral segments accordingly to the surface triangulation option (see `SMSSegmentsTriangulation`). The nodal values are then calculated on the assumption of a standard linear izoparametric interpolation of all fields on resulting 3 D triangles or quadrilaterals. The result is approximate for the higher order elements.
Polygon[*f_Function*]	This form assumes that the nodes that match *nodeSelector* form a surface (2 D or 3 D) and that the distributed boundary condition is prescribed on the surface. Parameter f is a function used to evaluate the intensity of the distributed boundary condition accordingly to the topology of the surface segments as follows: "T1" $\Rightarrow db=f[X,Y]$ "Q1" $\Rightarrow db=f[X,Y]$ "P1" $\Rightarrow db=f\left[X,Y,Z,n_x,n_y,n_z\right]$ "S1" $\Rightarrow db=f\left[X,Y,Z,n_x,n_y,n_z\right]$ Vector $\left(n_x,n_y,n_z\right)$ is a unit normal vector on the surface in point (X,Y,Z). **The surface is divided into triangular or quadrilateral segments regardless on topology of the underlying elements.**
Null	remove the coresponding prescribed natural boundary conditions if set by previous definitions

Forms of intensity of distributed natural boundary conditions when domain is defined by nodes.

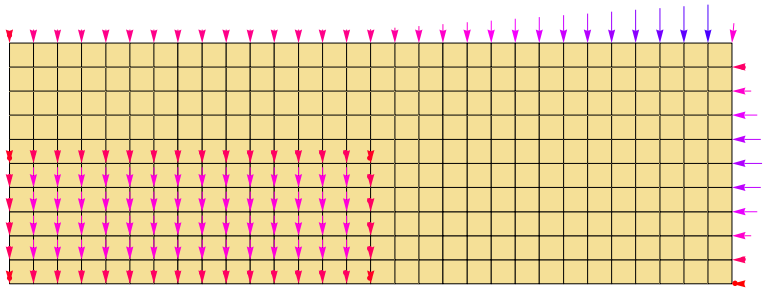
Examples:

- Generic example:
  - (A) constant distributed surface load
  - (B) distributed surface load with linear distribution of intensity
  - (C) distributed surface load with an arbitrary distribution of intensity

- (D) constant distributed volume load



In[365]:=
```
<< AceFEM`;
SMTInputData[];
L = 9; H = 3;
SMTAddDomain["Ω", {"ML:", "SE", "PE", "Q1", "DF", "HY", "Q1", "D", {{"NeoHooke", "WA"}}}, {}];
SMTAddMesh[Polygon[{{0, 0}, {L, 0}, {L, H}, {0, H}}], "Ω", "Q1", {30, 10}];
(*A*)SMTAddNaturalBoundary[Line[{{0, H}, {L / 2, H}}], 2 -> Line[{-1}]];
(*B*)SMTAddNaturalBoundary[Line[{{L / 2, H}, {L, H}}], 2 -> Line[{-1, -3}]];
(*C*)SMTAddNaturalBoundary[Line[{{L, 0}, {L, H}}], 1 -> Line[Function[{X, Y}, -Y (H - Y)]]];
(*G*)
SMTAddNaturalBoundary[Polygon[{{0, 0}, {L / 2, 0}, { L / 2, H / 2}, {0, H / 2}}], 2 -> Polygon[{-5}]];
SMTAnalysis[];
SMTShowMesh["BoundaryConditions" -> True]
```



Distributed Natural Boundary Defined by Elements

SMTAddNaturalBoundary[{"DistributedOver", *elementSelector*}, dof_1->db_1, dof_2->db_2,...]
increment or set reference value of natural (Neumann) boundary condition in nodes of selected elements (see `Selecting Elements`) where db_i is the reference value of the distributed natural boundary condition (force) for the dof_i-th unknown as described below

Applying distributed natural boundary conditions on volume (surface, line) defined by elements.

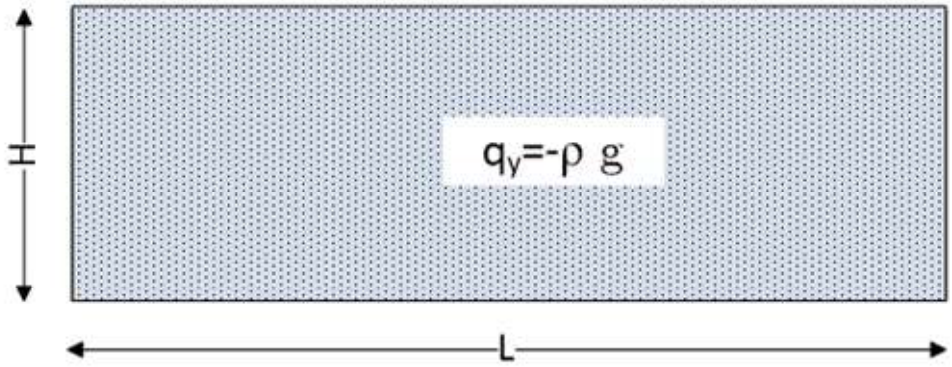
boundary condition intensity	description
db_Number	This form assumes that uniformly distributed boundary condition with the intensity *db is* prescribed on all selected elements.
f_Function	This form calculates the intensity of the distributed boundary condition accordingly to the topology of the elements as follows: "L1","L2" ⇒ $db=f\left[X,Y,t_x,t_y\right]$ "T1","T2","Q1","Q2","Q2S" ⇒ $db=f[X,Y]$ "C1" ,"C2" ⇒ $db=f\left[X,Y,Z,t_x,t_y,t_z\right]$ "P1","P2","S1","S2","S2S" ⇒ $db=f\left[X,Y,Z,n_x,n_y,n_z\right]$ "O1","O2","H1","H2","H2S" ⇒ $db=f[X,Y,Z]$ Vector (n_x,n_y,n_z) is a unit normal vector on the surface in point (X,Y,Z). Vector (t_x,t_y,t_z) is unit tangent vector in the direction of the curve in point (X,Y).
Null	remove the coresponding prescribed natural boundary conditions if set by previous definitions

Forms of intensity of distributed natural boundary conditions when domain is defined by elements.

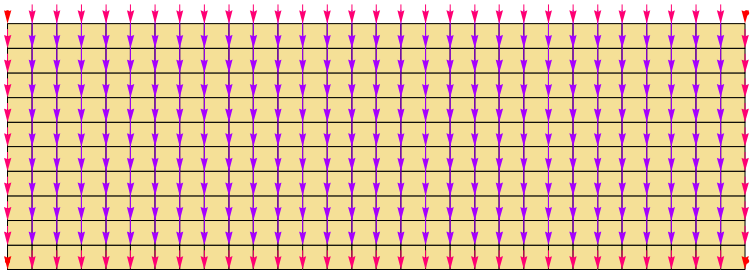
Only elements with topologies "L1", "L2", "C1", "C2", "T1", "T2", "P1", "P2", "Q1", "Q2", "Q2S", "S1", "S2", "H1", "H2", "H2S", "O1", "O2" are supported.

Examples:

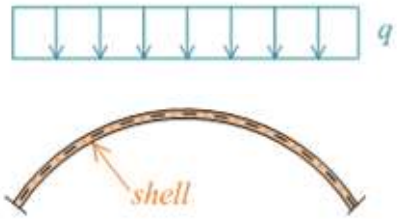
- Generic example:
  - add self weight to the domain "Ω" where ρ is the density and g is the gravitational acceleration.



In[383]:=
```
<< AceFEM`;
SMTInputData[];
L = 9; H = 3; ρ = 8100; g = 9.81;
SMTAddDomain["Ω", {"ML:", "SE", "PS", "Q1", "DF", "HY", "Q1", "D", {{"NeoHooke", "WA"}}}, {}];
SMTAddMesh[Polygon[{{0, 0}, {L, 0}, {L, H}, {0, H}}], "Ω", "Q1", {30, 10}];
SMTAddNaturalBoundary[{"DistributedOver", "Ω"}, 2 → -ρ g];
SMTAnalysis[];
SMTShowMesh["BoundaryConditions" -> True]
```

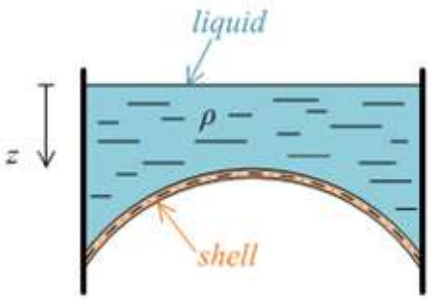


- Lets assume that we have modelled a shell like structure with shell finite elements. This would add a snow like distributed load with intensity q on the surface of the shell.



In[382]:=
```
SMTAddNaturalBoundary[{"DistributedOver", "shell"},
  2 → Function[{X, Y, Z, nx, ny, nz}, q Abs[nz]]]
```

- Lets assume that we have a shell structure filed with liquid and that the Z coordinate is the depth of the liquid with density ρ. This would add hydrostatic pressure on the surface of the shell.




```
SMTAddNaturalBoundary[{"DistributedOver", "shell"}, 1 → Function[{X, Y, Z, nx, ny, nz}, ρ g Z nx],
  2 → Function[{X, Y, Z, nx, ny, nz}, ρ g Z ny], 3 → Function[{X, Y, Z, nx, ny, nz}, ρ g Z nz]]
```

Initial Boundary Conditions

SMTAddInitialBoundary[*selector*, dof_1->v_1, dof_2->v_2,...]

increment or set initial state of the problem where v_i is the initial state for the dof_i-th unknown and *ibtype* the type of prescribed initial boundary condition accordingly to the table below where *selector* can be used to select nodes (see `Selecting Nodes`) or elements (see `Selecting Elements`) on which boundary condition is applied

option	default	description
"Set"	False	override the previous defined boundary conditions for the chosen degrees of freedom with the newly defined values
"Type"	*Automatic*	see table below

Options for prescribed boundary conditions.

AceFEM generally deals with the nonlinear problem and uses various path following procedures to solve it. By default, the state of the system at the beginning of the path-following procedure is zero state. Alternatively, an initial nonzero state of either essential or natural boundary conditions can be set by `SMTAddInitialBoundary`.

The initial boundary condition can be either initial essential or initial natural boundary condition or initial value of chosen DOF. Default type of initial boundary condition is initial natural unless the essential boundary condition is also defined for the same DOF by the *SMTAddEssentialBoundary* command. With the option "Type"->*ibtype* is the type of initial boundary condition explicitly defined. The parameter *selector* is by default used to select nodes (see `Selecting Nodes`), except for the "Type"->"Distributed" where the elements on which distributed load is applied have to be selected (see `Selecting Elements`).

"Type"	description
Automatic	If the chosen DOF is also constrained by the *SMTAddEssentialBoundary* command then this sets initial value ($Bp_i=Bt_i=v_i$) of essential boundary condition, otherwise the initial value ($Bp_i=Bt_i=v_i$) of the natural boundary condition is set for the chosen DOF in selected nodes.
"EssentialBoundary"	Constrain chosen DOF and set initial value of the chosen DOF ($Bp_i=Bt_i=v_i$) in selected nodes as described in `Essential Boundary Conditions`.
"InitialCondition"	Set initial value of chosen DOF ($ap=at=v$) for e.g. Initial value problems. Eventual boundary conditions previously attached to chosen DOF are left unchanged!
"Distributed"	Adds distributed initial natural boundary conditions ($Bp_i=Bt_i=v_i$) on selected elements as decribed in Distributed Natural Boundary Defined by Elements.

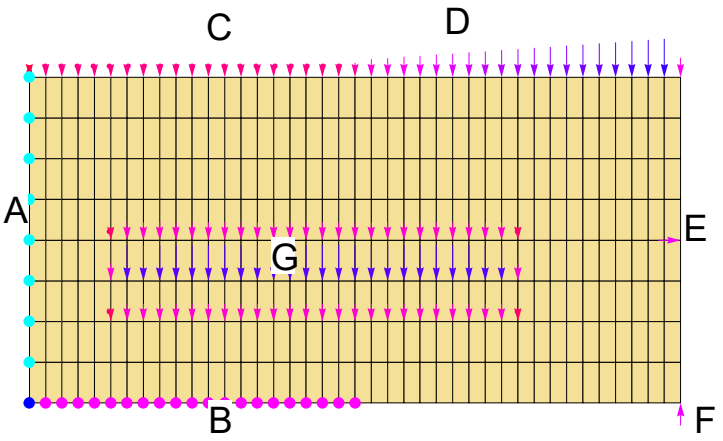
Option "Type" for SMTAddInitialBoundary command.

- Set the initial condition for first DOF in all nodes with node identification "D" to be 1/2.

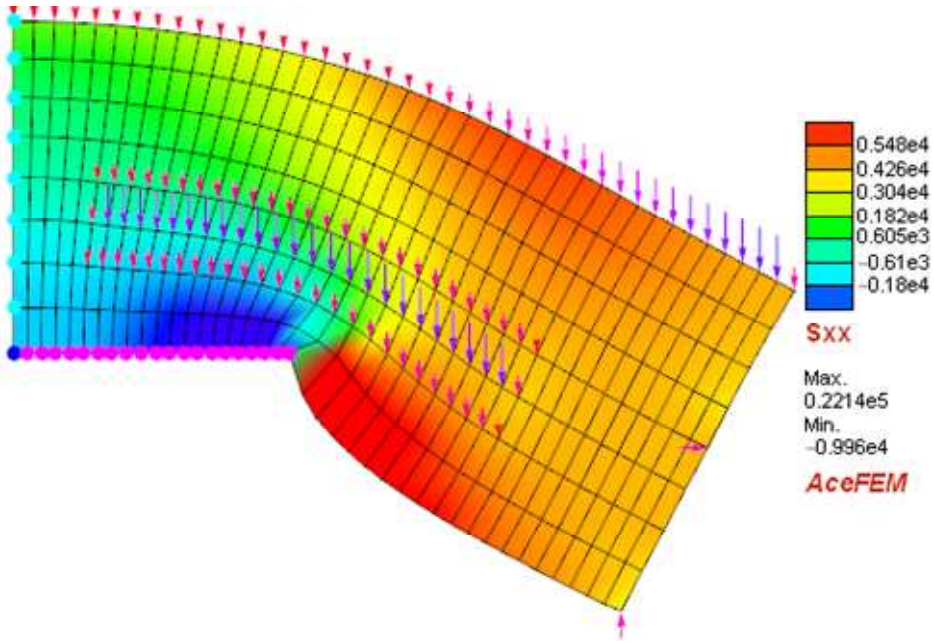
```
SMTAddInitialBoundary["D", 1 → 1 / 2, "Type" -> "InitialCondition"]
```

Examples of Prescribed Boundary Conditions

2D Example

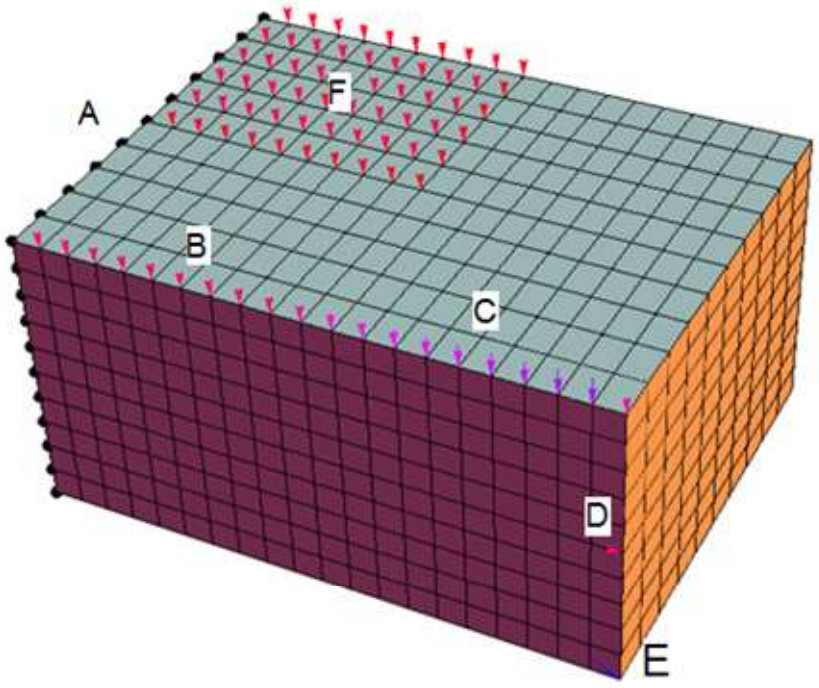


In[390]:=
```
<< AceFEM`;
SMTInputData[];
L = 100; H = 50;
q1 = 2; q2 = 1; q3 = 1;
nx = 40; ny = 10;
SMTAddDomain["A", {"ML:", "SE", "PS", "Q1", "DF", "HY", "Q1", "D", {{"NeoHooke", "WA"}}},
  {"E *" -> 1000., "ν *" -> .49, "t *" -> 1.}];
SMTAddMesh[Polygon[{{0, 0}, {L, 0}, {L, H}, {0, H}}], "A", "Q1", {nx, ny}];
(*A*)SMTAddEssentialBoundary[Line[{{0, 0}, {0, H}}], 1 -> 0];
(*B*)SMTAddEssentialBoundary[Line[{{0, 0}, {L / 2, 0}}], 2 -> 0];
(*C*)SMTAddNaturalBoundary[Line[{{0, H}, {L, H}}], 2 -> Line[{-q1}]];
(*D*)SMTAddNaturalBoundary[Line[{{L / 2, H}, {L, H}}], 2 -> Line[{-q2, -5 q2}]];
(*E*)SMTAddNaturalBoundary[Line[{{L, 0}, {L, H}}], 1 -> Point[{H / 2, 10}]];
(*F*)SMTAddNaturalBoundary[Point[{L, 0}], 2 -> 10];
(*G*)SMTAddNaturalBoundary[
  Polygon[{{H / 4, H / 4}, {3 L / 4, H / 4}, {3 L / 4, H / 2}, {H / 4, H / 2}}], 2 -> Polygon[{-1}]];
SMTAnalysis[];
SMTNextStep["λ" -> 10];
SMTNewtonIteration[];
SMTShowMesh["BoundaryConditions" -> True, "DeformedMesh" -> True, "Field" -> "Sxx"]
```



3D Example

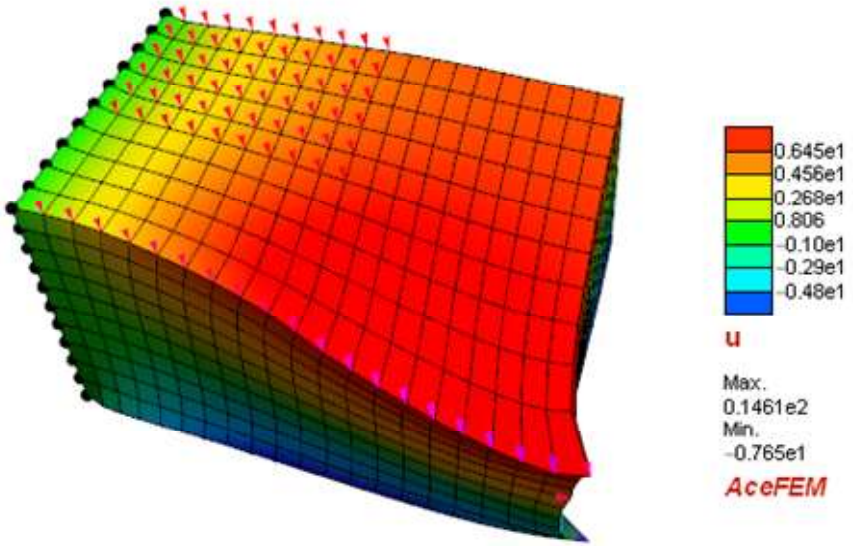
In[408]:=
```
<< AceFEM`;
SMTInputData[];
q1 = 20; q2 = 50; q3 = 10;
H = 50; A = 100; B = 75;
SMTAddDomain["A",
  {"ML:", "SE", "D3", "H1", "DF", "LE", "H1", "D", "Hooke"}, {"E *" -> 1000., "ν *" -> .49}];
{nx, ny, nz} = {20, 10, 10};
SMTAddMesh[Hexahedron[{{0, 0, 0}, {A, 0, 0}, {A, B, 0}, {0, B, 0},
    {0, 0, H}, {A, 0, H}, {A, B, H}, {0, B, H}}], "A", "H1", {nx, ny, nz}];
```



In[415]:=
```
(*A*)SMTAddEssentialBoundary[
  Polygon[{{0, 0, 0}, {0, B, 0}, {0, B, H}, {0, 0, H}}], 1 → 0, 2 -> 0, 3 -> 0];
(*B*)SMTAddNaturalBoundary[Line[{{0, 0, H}, {A, 0, H}}], 3 -> Line[{-q1}]];
(*C*)SMTAddNaturalBoundary[Line[{{A / 2, 0, H}, {A, 0, H}}], 3 -> Line[{-q1, -q2}]];
(*D*)SMTAddNaturalBoundary[Line[{{A, 0, 0}, {A, 0, H}}], 1 -> Point[{H / 2, 100}]];
(*E*)SMTAddNaturalBoundary[Point[{A, 0, 0}], 1 -> 400, 3 -> -200];
(*F*)SMTAddNaturalBoundary[
  Polygon[{{0, B / 2, H}, {A / 2, B / 2, H}, {A / 2, B, H}, {0, B, H}}], 3 → Polygon[{-2}]];
```

In[421]:=
```
SMTAnalysis[];
```

In[422]:=
```
SMTNextStep["λ" → 30];
While[SMTConvergence[10^-8, 10], SMTNewtonIteration[];]
SMTShowMesh["DeformedMesh" → True,
 "BoundaryConditions" → True, "DeformedMesh" -> True, "Field" → "u"]
```


0.645e1
0.456e1
0.268e1
0.806
-0.10e1
-0.29e1
-0.48e1
u
Max.
0.1461e2
Min.
-0.765e1
AceFEM

Analysis Phase

Contents

General Description

The standard parts of the analysis phase are:

- The analysis phase starts with the initialization (see SMTAnalysis).
- The *AceFEM* is designed to solve steady-state or transient finite element and related problems implicitly by means of Newton-Raphson type procedures. The actual solution procedure is executed accordingly to the *Mathematica* input given by the user. The details are given in Iterative Solution Procedures.

- The graphic post-processing of the results can be part of the analysis (SMTPostData, SMTShowMesh) or done later independently of the analysis (SMTPut).
- All the data in the data base can be directly accessed from *Mathematica* and most of the data can be also changed during the analysis using Data Base Manipulations .

The SMTAnalysis command does the following:

- checks the correctness of the input data structures;
- transcripts input data structures into analysis data base structures;
- correct mesh input data in a way that:
  - nodes which have coordinates and node identification with the same value are joined into single node (see "Tie" command of SMTAnalysis command);
  - for elements modified with SMSAdditionalNodes (see Self – transforming meshes) option given set of nodes is transformed into the true element nodes;
- compiles the element source files and creates dynamic link library files or in the case of MDriver numerical;
- module reads all the element source files into *Mathematica;*
- performs initialization of the data base structures (see Iterative Solution Procedures).

The numbering of the nodes is changed after the data base construction (SMTAnalysis). Thus, the node numbers used at the input data phase are not valid any more at the analysis phase. They have to be replaced by the true node numbers. If *n* is the index of the node at the input data phase then *SMTTrueNodeNumber[[n]]* is the index of the same node at the analysis phase (after SMTAnalysis). Note that nodes that had different indexes at the input data phase can share the same index after the SMTAnalysis command.

Main iterative loop

Symbol table:

□t current iterative value

□p value at the end of previous time (load) step

Bt current value of boundary conditions

Bp value of boundary conditions at the end of previous step

dB vector associated with the pattern of the applied boundary conditions

$\tilde{\boldsymbol{Bt}}$ part of the current boundary conditions vector (***Bt***) where essential boundary conditions are prescribed by SMTAddEssentialBoundary

$\overline{\boldsymbol{Bt}}$ part of the current boundary conditions vector (***Bt***) where natural boundary conditions are prescribed

By default all the unknowns have prescribed natural boundary condition (set to 0), thus $\overline{\boldsymbol{Bt}}$:= ***Bt***\$\tilde{\boldsymbol{Bt}}$. The nonzero value of the natural boundary condition can be prescribed by SMTAddNaturalBoundary.

Bi initial boundary conditions

The initial boundary conditions are not stored into the analysis data base. They are used for the initialization of the ***Bp*** vector at the start of the analysis and discarded after.

ap, at global variables with unknown value

$\tilde{\boldsymbol{ap}}$, $\tilde{\boldsymbol{at}}$ global variables with prescribed value (essential boundary condition)

ht, hp vector of time dependent variables that are local to specific element

The following steps are performed at the start at of the analysis:

- data is set to zero

  ***at=ap=ht=hp=*0**

- boundary conditions are set to initial boundary conditions (***Bi***) prescribed by `SMTAddInitialBoundary`

  Bt:=Bi

  Bp:=Bi

The following steps are performed at the start of time or multiplier increment:

- time and boundary conditions multiplier are incremented by `SMTNextStep`.

  $t:=t+\Delta t$

  $\lambda:=\lambda+\Delta\lambda$

  $\gamma:=\gamma+\Delta\gamma$

- global variables at the end of previous step are reset to the current values of global variables

  ap:=at

- boundary conditions at the end of previous step are reset to the current values of boundary conditions

  Bp:=Bt

The following steps are performed for each iteration:

- global vector ***R*** and global matrix ***K*** are initialized

  ***R:=*0, *K:=*0**

- boundary conditions are updated as follows:
  - the current boundary value is calculated as $\boldsymbol{Bt:=Bp+\Delta\lambda\, dB}$, where $\Delta\lambda$ is the multiplier increment
  - essential boundary conditions are set $\tilde{\boldsymbol{at}} := \tilde{\boldsymbol{Bt}}$
  - the global vector of natural boundary conditions is subtracted from the global vector $\boldsymbol{R:=R+\overline{Bt}}$
- user subroutine *"Tangent and residual"* is called for each element,
  - the element tangent matrix $\mathbf{K_e}$ is added to the global matrix $\boldsymbol{K:=K+}\mathbf{K_e}$,
  - element residual $\mathbf{R}_e$ is taken from the global vector $\mathbf{R} := \mathbf{R} - \mathbf{R}_e$
- set of linear equations is solved $\boldsymbol{K\,\Delta a=R}$,
- solution is incremented $\boldsymbol{at:=at+\Delta a}$.

Main Analysis Phase Functions

SMTAnalysis

SMTAnalysis[*options*]
check the input data, create data structures and start the analysis

The SMTAnalysis also compiles the element source files and creates dynamic link library files (dll file) with the user subroutines (see also `SMTMakeDll`) or in the case of MDriver numerical module reads all the element source files into *Mathematica.*

option	default	description
"Output"	False (no output file)	output file name (for comments, debugging, etc.)
"DumpInputTo"	False	file name for storing input data Input data is stored to files *name.m* and *name.h5* that are used by AceFEM for independent post- processing (see `Separate Visualization`) or to run batch mode analysis (see Independent Batch Mode). For full save/restart of the AceFEM session see `SMTDump`.
"BatchModeSteering"	False	the name of the C source file for the steering of the batch mode analysis (see Independent Batch Mode)
"Solver"	0	0 ⇒ appropriate linear solver is chosen automatically None ⇒ solver is not set automatically, use `SMTSetSolver` directly solverID ⇒ solver with linear solver identification number solverID {solverID,p_1,p_2,...} ⇒ solver with linear solver identification number *solverID* and set of parameters for initialization (see also `SMTSetSolver`)
"OptimizeDll"	Automatic	True ⇒ set compiler options for the fastest code False ⇒ set compiler options for debugging
"SearchFunction"	(#&)	function applied on coordinates of nodes before the search procedures if the coordinates are not numbers (e.g before "Tie", `SMTFindNodes`, etc.)
"Precision"	6	number of significant digits used within the search procedures (e.g for the "Tie" option, `SMTFindNodes`, etc.)
"Tolerance"	10^{-10}	the numbers smaller in absolute magnitude than *"Tolerance"* are replaced by 0 within the search procedures
"Tie"	True	True ⇒ join nodes which have coordinates and node identification with the same values {True, *nodeselctor*} ⇒ join nodes which have coordinates and node identification with the same values, except the nodes that match the *nodelector* (see also `Selecting Nodes`) False ⇒ suppress tie (the tie is by default suppresed for all nodes that have node identification switch -T, see Node Identification)
"Debug"	False	True ⇒ prevent closing of the CDriver console on exit
"NodeReordering"	Automatic	Reordering of the nodes effects the numerical efficiency of the linear solver used as well as memory consumption, however there is no assurance that the node reordering will actually have positive effect. False ⇒ no node reordering apart from the nodes with the node identification switch -E that are always positioned at the end "AdvancingFront" ⇒ Simple reordering scheme based on the element connectivity table. Additional nodes of the element are always positioned after the topological nodes. The nodes with the switch - E are always positioned at the end. Automatic ⇒ "AdvancingFront"
"ContactPairs"	Automatic	specify the pairs slave-body/master - body for the possible contact condition is checked {{slaveBodyID$_1$,masterBodyID$_1$},{slaveBodyID$_2$,masterBodyID$_2$},...}

Options of the *SMTAnalysis* function.

SMTNewtonIteration

SMTNewtonIteration[]

executes one iteration of the general Newton-Raphson iterative procedure.

Detailed description and examples are given in section `Iterative Solution Procedures`.

option	default	description
"AllNorms"	False	False ⇒ The function returns modified Euklid' s norm of the increment of global d.o.f $\|\|\mathbf{da}\|\|_S$. True ⇒ returns the error analysis table **E**
"EBCUpdate"	Default	the metod for the update of the essential BC (see table below)

Options of the SMTNewtonIteration function.

"EBCUpdate"	IData["LinearEstimate"]	description
Default	1	≡ "EBC to NBC"
"EBC to NBC"	1	i_{NR}=1 ⇒ in the first iteration of the NewtonRaphson iterative procedure the essential boundary conditions are not updated and the residual is evaluated by $\mathbf{at}^{(0)}$=**ap** && $\mathbf{R}^{(0)}$= $\mathbf{R}(\mathbf{ap})+\mathbf{K}(\mathbf{ap})*\mathbf{da}_{EBC}$ and IData["ResidualAndTangentTask"]=1; i_{NR}>1 ⇒ "EBC to Bt"
"EBC to Bt"	0	DOF`s with prescribed essential BC are updated to time n+1 and IData["ResidualAndTangentTask"]=0
"EBC to Bp"	2	DOF`s with prescribed essential BC are set to time n and IData["ResidualAndTangentTask"]=0
"No update"	4	DOF`s are not changed (user can set it`s own values!) and IData["ResidualAndTangentTask"]=0

The option "EBCUpdate" defines the way how the DOF`s with prescribed essential BC are updated before the execution of the NR iteration.

The "EBCUpdate" option can be used to implement various types of predictor-corrector methods for the solution of the resulting parameterized system of nonlinear equations.

The error analysis table is of the form:

$$\mathbf{E} = \{\{\|\mathbf{da}\|_S, \|\mathbf{R}\|_S\}, \text{Table}[\{\|\mathbf{da}_{i,j}\|_S, \|\mathbf{at}_{i,j}\|_S, \|\mathbf{R}_{i,j}\|_S, \|\mathbf{da}_{i,j}\|_S \alpha_{i,j}, \|\mathbf{R}_{i,j}\|_S / \alpha_{i,j}\}, \{i, 1, \text{No Node Specifications}\}, \{j, 1, \text{No DOF}_i\}]\}$$

where

$\alpha_{i,j}$ = SMTNodeSpecData["DOFScaling"][[i, j]] is a scaling factor that belongs to *i*-th unknown in nodes with *j*-th node identification,

$\mathbf{da}_{i,j}$ is a vector of increments of *i*-th unknown in all nodes with *j*-th node identification,

$\mathbf{at}_{i,j}$ is a vector of *i*-th unknown in all nodes with *j*-th node identification,

$\mathbf{R}_{i,j}$ is a vector of *i*-th component of residual in all nodes with *j*-th node identification,

$$\|\mathbf{da}\|_S = \sqrt{\frac{\sum_{i=1}^{\text{No node specs}} \sum_{j=1}^{\text{No DOF}_i} \mathbf{da}_{i,j}.\mathbf{da}_{i,j} / \alpha_{i,j}^2}{\text{NoEquations}}}, \quad \|\mathbf{R}\|_S = \sqrt{\frac{\sum_{i=1}^{\text{No node specs}} \sum_{j=1}^{\text{No DOF}_i} \mathbf{R}_{i,j}.\mathbf{R}_{i,j} \alpha_{i,j}^2}{\text{NoEquations}}}.$$

See Also

- Iterative Solution Procedures
  - Example : constant BC multiplier increment
  - Example : successive values of time
  - Example : adaptive BC multiplier
  - Example : adaptive time and constant BC multiplier
  - Example : simultaneous incrementation of time and BC multiplier
  - Example : adaptive BC multiplier with visualization
  - Arc - length examples
- Bending of the column (path following procedure, animations, 2 D solids)
- Iterative Arc - length Solution Procedure

SMTNextStep

SMTNextStep[*options*]
goes to the next time/load/parameter accordingly to the given options.

Detailed description and examples are given in section Iterative Solution Procedures.

option	default	description
"Δt"→r	0	time increment (t←t+r)
"Δλ"→r	0	increment of boundary conditions multiplier (λ←λ+r)
"Δγ"→r	0	increment of general parameter (γ←γ+r)
"t"→r	None	next time (t←r)
"λ"→r	None	next boundary conditions multiplier (λ←r)
"γ"→r	None	next general parameter (γ←r)
"λ[t]"→f	None	function that defines boundary conditions multiplier λ as a function of time t (λ←f[t]) (e.g. "λ[t]"->Function[{t},Sin[t]]] or "λ[t]"->Sin)
"λ[γ]"→f	None	function that defines boundary conditions multiplier λ as a function of general parameter γ (λ←f[γ])
"t[γ]"→f	None	function that defines time t as a function of general parameter γ (t←f[γ])

Options of the SMTNextStep function.

The values of all variables that define the state of the system at the end of the previous time step (**ap, sp, hp, Bp**) are after the SMTNextStep command set to be equal to the current values (**ap ≡ at, sp ≡ st, hp ≡ ht, Bp ≡ Bt**).

The problem can be parameterized either by time (t), boundary conditions multiplier (λ) or a general parameter (γ). Time, multiplier and a general parameter are stored in real type environmental variables SMTRData["Time"], SMTRData["Multiplier"] and SMTRData["-Parameter"]. The command SMTNextStep["Δt"->x, "Δλ"->y, "Δγ"->z] increments all parameters. However, within the adaptive path-following procedures only one parameter can be a leading parameter of the problem and the other two are the function of the leading one.

SMTNextStep[Δt, Δλ] ≡ SMTNextStep["Δt"→Δt, "Δλ"→Δλ]

SMTNextStep[Δt, Δλ, Δγ] ≡ SMTNextStep["Δt"→Δt, "Δλ"→Δλ, "Δγ"→Δγ]
goes to the next time/load/parameter step by updating the current time (t←t+Δt), boundary conditions multiplier (λ←λ+Δλ) and general parameter (γ←γ+Δγ) and time dependent data (ap≡at, sp≡st, hp≡ht, Bp≡Bt).

SMTNextStep[Δγ] ≡ SMTNextStep["Δγ"→Δγ]

SMTNextStep[Δt,λ_f]
updates current time (t←+Δt) and boundary conditions multiplier (λ←λ+λ_f(t+Δt)-λ_f(t))

where λ_f(t) is a boundary conditions multiplier as a function of time.

SMTNextStep[Δt,λ_f] ≡ SMTNextStep["Δt"->Δt, "Δλ"->λ_f[t+Δt]-λ_f[t]]

Legacy forms for compatibility with versions lower than 6.2.

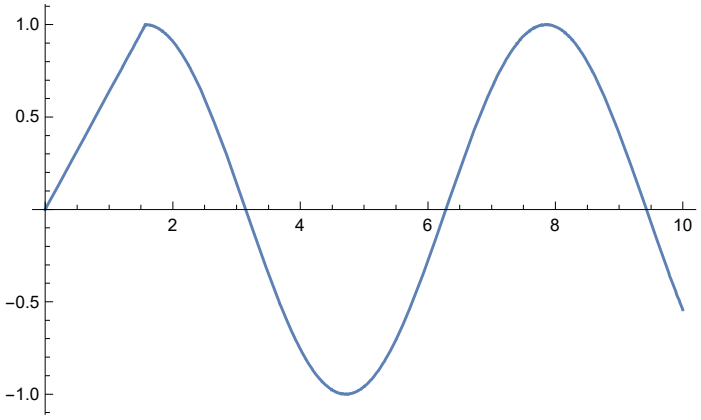
Examples

```
In[17]:= SMTNextStep["Δt" → 1]
```

```
In[17]:= SMTNextStep["t" → 1]
```

```
In[17]:= load[time_] := Piecewise[{{Sin[time], time > π / 2}, {2 / π time, time ≤ π / 2}}]
         SMTNextStep["t" → 1, "λ[t]" → load]
```

```
In[20]:= Plot[load[t], {t, 0, 10}]
```



See Also

- Iterative Solution Procedures, SMTNewtonIteration
- Bending of the column (path following procedure, animations, 2 D solids)
- Iterative Arc - length Solution Procedure

SMTStepBack

SMTStepBack[]
makes the current state of the system to be the same as the one at the end of the previous time step (***at*≡*ap***, ***st*≡*sp***, ***ht*≡*hp***, ***Bt*≡*Bp***).

Detailed description and examples are given in section Iterative Solution Procedures.

- Iterative Solution Procedures, SMTNewtonIteration
- Iterative Arc - length Solution Procedure

SMTConvergence

SMTConvergence[*tol*,*n*,*type*]
analyzes the current state of the Newton-Raphson iterative procedure and returns the result of the analysis.

Detailed description and examples are given in section Iterative Solution Procedures.

SMTConvergence[*tol*,*n*] ≡ SMTConvergence[*tol*,*n*,"Abort"]

SMTConvergence[] ≡ SMTConvergence[10^-7,15,"Abort"]

SMTConvergence["FullReset"]

SMTConvergence["Reset"]

The SMTConvergence["FullReset"] command completely clears the history of iterations.

The SMTConvergence["Reset"] command partially clears the history of iterations from the second one to the current one. With this a special treatment of the first iteration is omitted.

- Example : constant BC multiplier increment
- Example : successive values of time
- Example : adaptive BC multiplier
- Example : adaptive time and constant BC multiplier
- Example : simultaneous incrementation of time and BC multiplier
- Example : adaptive BC multiplier with visualization

The SMTConvergence command continuously stores data related to iterations and makes decisions accordingly to the history of iterations. Consequently, it only works properly when it is called **alternately with the SMTNewtonIteration command**.

The return value of the SMTConvergence function depends on the type of the path following procedure and the options given to the SMTConvergence function. The return value has to be properly interpreted by the user and an appropriate action has to be performed accordingly to the return value. In all cases the function returns *True* when one more iteration is required within the current time step.

type	return value	description
"Abort"	True	$\|\Delta a^t\|$>tol , one more iteration is required
	none	iterative process is aborted in the case of divergence
"Analyze"	True	$\|\Delta a^t\|$>tol , one more iteration is required
	False	$\|\Delta a^t\|$<tol, convergence condition for the iterative procedure has been satisfied
	divergence_type	divergence (the possible values of divergence_type are defined in a table below)
{"Adaptive Time", m, Δt_{min}, Δt_{max}, t_{max}}	True	$\|\Delta a^t\|$>tol , one more iteration is required within the current time step
	{*step_back*, Δt, *step_forward*, *report*}	this return value is used to steer the path following procedure of the problem parameterized with time t (see below)
{"Adaptive BC", m, $\Delta\lambda_{min}$, $\Delta\lambda_{max}$, λ_{max}}	True	$\|\Delta a^t\|$>tol , one more iteration is required within the current time step
	{*step_back*, Δλ, *step_forward*, *report*}	this return value is used to steer the path following procedure of the problem parameterized with boundary conditions multiplier λ (see below)
{"Adaptive γ", m, $\Delta\gamma_{min}$, $\Delta\gamma_{max}$, γ_{max}}	True	$\|\Delta a^t\|$>tol , one more iteration is required within the current time step
	{*step_back*, Δγ, *step_forward*, *report*}	this return value is used to steer the path following procedure of the problem parameterized with general parameter γ (see below)

Return values accordingly to the value of the parameter *type*.

The parameter *tol* is the tolerance, *n* is the maximum number of iterations, *m* is the desired number of iterations, Δt, Δλ or Δγ are the suggested values of increment of time, multiplier and general parameter, Δt_{min}, $\Delta\lambda_{min}$ or $\Delta\gamma_{min}$ are the minimum and Δt_{max}, $\Delta\lambda_{max}$ or $\Delta\gamma_{max}$ are the maximum values of the increments of time, multiplier or general parameter. The target or terminal value of the parameter used to parameterized the problem is defined by t_{max}, λ_{max} or γ_{max}. Note that within the adaptive path-following procedures only one parameter can be a leading parameter of the problem (t, λ,or γ) and the other two are the function of the leading one.

This function is implicitly defined by the `SMTNextStep` command.

path following parameters	description
step_back	If *step_back* is True then the iterative procedure in the current step has failed to converge, thus the values of all variables that define the state of the system has to be returned to the last converged state using the SMTStepBack command. A new, shorter step can be attempted using the SMTNextStep command.
step_forward	If *step_forward* is True then the target value of time (t_{max}) or boundary conditions multiplier (λ_{max}) has not yet been achieved, thus an additional step has to be made using the SMTNextStep command. Parameter *step_forward* is False in the case that the final goal of simulation have been reached and if the simulation has to be prematurely aborted (e.g. if $\|\Delta t\|<\Delta t_{min}$ or $\|\Delta\lambda\|<\Delta\lambda_{min}$).
Δt, $\Delta\lambda$ or $\Delta\gamma$	The suggested values of the increment of the parameter used to parameterized the problem is calculated on a basis of the history of iterations and steps.
report	The parameter *report* gives detailed analysis of the current state of the iterative path following procedure. The possible values of report are defined in a table below.

Return values accordingly to the value of the parameter *type*.

report	description
$\|\|\Delta a^t\|\|$	The norm of the increment of DOF–s in last iteration is returned when no special action si required.
"MinBound"	The proposed value of the increment of parameter is less than prescribed minimum ($\|\Delta t\|<\Delta t_{min}$, $\|\Delta\lambda\|<\Delta\lambda_{min}$ or $\|\Delta\gamma\|<\Delta\gamma_{min}$), thus the path following procedure has failed.
"MaxBound"	The target value of parameter has been reached ($\|t\|\geq\|t_{max}\|$, $\|\lambda\|\geq\|\lambda_{max}\|$ or $\|\gamma\|\geq\|\gamma_{max}\|$), thus the simulation has succeeded.
divergence_type	The value of *divergence_type* gives detailed analysis why the iterative procedure has failed to converge. The possible values of *divergence_type* are defined in a table below.

Detailed analysis of the current state of the iterative path following procedure.

divergence_type	description
"Divergence"	The divergence of the increments of DOF–s has been detected ($\|\|\Delta\mathbf{a}^t\|\| \to\infty$).
"N/A"	An error has been detected due to indetermined values during the evaluation of element tangent and residual (e.g 1/0 , Sqrt (–1), etc. situation).
"Alternate"	The norm of the increments of DOF–s alternates between two or more values. This condition is interpreted accordingly to the value of the option "Alternate" as described below.
"ErrorStatus"	During the evaluation of the element tangent and residual the error status flag has been set to 2 (see `Iterative Solution Procedures`).
"Unknown"	The reason for the lack of convergence of iterative procedure is unknown.

Detailed analysis of the reason for the divergence of the iterative procedure.

option	default value	description
"MaxTotalSteps"	Infinity	maximum total number of steps and back-steps
"MaxTime"	Infinity	maximum absolute time allowed in seconds
"Alternate"	"Divergence"	"Ignore" ⇒ detection of alternating solution is ignored "Divergence" ⇒ detection of alternating solution is activated and if the alternating solution is detected, the iterations are stopped and a new time or multiplier step is proposed True ≡ "Divergence" False≡"Ignore" Automatic ≡ $\{\{\Delta t_{max}/4, \Delta\lambda_{max}/4\}, \{\Delta t_{max}/10, \Delta\lambda_{max}/10\}, \{\infty,\infty\}\}$ {{Δtl,Δλl}, {Δtg,Δλg}, {Δtg1,Δλg1}} ⇒ if the alternating solution occurs then start the following procedure: 1) if Δt<Δtl ‖ Δλ<Δλl then set SMTIData["SubIterationMode",1]; 2) if solution still alternates and Δt<Δtg ‖ Δλ<Δλg then set SMTIData["GlobalIterationMode",1]; 3) if convergence conditions have been satisfied and Δt>Δtg1 ‖ Δλ> Δλg1 then set SMTIData["GlobalIterationMode",0] and repeat iterations
"PostIteration"	Automatic	enables an additional call to the SKR user subroutines after the convergence of the global solution (see `SMSPostIterationCall`) Automatic ⇒ perform post-iteration call accordingly to the value of the template constant of specific element False ⇒ prevent post-iteration call True ⇒ force post-iteration call *tolR* ⇒ tolerance for the residual (if $\|\mathbf{R}\|>tolR$ then "ErrorStatus" is set to 2)
"IgnoreMinBound"	False	If an alternating solution occurs at the minimum time or multiplier increment ($\|\Delta\lambda\|<\Delta\lambda_{min}$ or $\|\Delta t\|<\Delta t_{min}$ or $\|\Delta\gamma\|<\Delta\gamma_{min}$) then ignore the minimum increment condition and proceed for another time or multiplier step.
"AlternativeTarget"	(True&)	The primal target of the adaptive path following procedure is to reach λ_{max} or t_{max} or γ_{max}. The "AlternativeTarget" option sets an additional target for the adaptive path following procedure. Alternative target is a function that is evaluated at the end of each successfully completed time or multiplier increment. If the target function yields False then the divergence of the NR procedure for the given time increment is assumed and treated accordingly to the other options given to the SMTConvergence function.
"SecondStep"	{1/5.,1/2.}	Option accelerates steps after the small first step. In the case that the first step increment is small $\Delta s^{(1)} < \Delta s_{max} *$ "SecondStep"[[1]] then the second step is taken as $\Delta s^{(2)} = \Delta s_{max} *$ "SecondStep"[[2]]. Path parameter *s* can be t, λ, or γ.

Options of the SMTConvergence function.

- `Bending of the column (path following procedure, animations, 2D solids)`
- `Iterative Arc-length Solution Procedure`

Iterative Solution Procedures

AceFEM is designed to solve time-independent or time-dependent finite element and related problems implicitly by the means of Newton-Raphson type iterative solution procedures. *AceFEM* has no predefined commands to perform complete path-following procedure (implicit solution procedure of the parameterized system of nonlinear equations), however it provides a set of commands that can be used to write an arbitrary path-following procedure. The most essential commands needed to construct an iterative path-following solution procedure are:

SMTNewtonIteration[]	executes one iteration of the general Newton-Raphson iterative procedure (for details see SMTNewtonIteration).
SMTNextStep["Δt"→Δt, "Δλ"→Δλ,"Δγ"→Δγ]	goes to the next step by updating the parameters and time dependent data (***ap≡at***, ***sp≡st***, ***hp≡ht***, ***Bp≡Bt***) (for details see SMTNextStep).
SMTStepBack[]	makes the current state of the system to be the same as the one at the end of the previous time step (***at≡ap***, ***st≡sp***, ***ht≡hp***, ***Bt≡Bp***).
SMTConvergence[*tolerance*,*max_iterations*]	returns True if $\|\|\Delta \boldsymbol{at}\|\| > tolerance$ and thus one more iteration is required, False if $\|\|\Delta \boldsymbol{at}\|\| \le tolerance$ or aborts the calculation if the number of iterations has exceeded *max_iterations* (for details see SMTConvergence).
SMTStatusReport[*expr*]	prints out the report of the current status of the system tagged by the arbitrary expression *expr* (for details see SMTStatusReport).
SMTSimulationReport[]	prints out report identifying the percentage of time spent in specific tasks during the analysis (for details see SMTSimulationReport).

The SMTConvergence command performs an analysis of the current state of the Newton-Raphson iterative procedure and returns the results of the analysis. The user is then responsible to act accordingly to the results (e.g. to make an additional iteration or to stop the iterative procedure). The SMTConvergence function returns in the case when one more iteration is required within the current time/load step the value True. For other cases the return value depends on the type of path following procedure and the options given to the SMTConvergence function (for details see SMTConvergence). The return value has to be properly interpreted by the user and an appropriate action has to be performed accordingly to the return value.

The problem can be parameterized either by time (t), boundary conditions multiplier (λ) or a general parameter (γ). Time, multiplier and a general parameter are stored in real type environmental variables SMTRData["Time"], SMTRData["Multiplier"] and SMTRData["-Parameter"]. The command SMTNextStep["Δt"→Δt, "Δλ"→Δλ,"Δγ"→Δγ] increments all parameters. However, within the adaptive path-following procedures only one parameter can be a leading parameter of the problem and the other two are the function of the leading one.

See Also

- SMTNewtonIteration
- Bending of the column (path following procedure, animations, 2 D solids)
- Iterative Arc - length Solution Procedure

Example: constant BC multiplier increment

This is a path following procedure where the path is parameterized by boundary conditions multiplier (load level). The command SMTNextStep["Δλ"→Δλ] increments the multiplier by Δλ.

The iterative path following solution procedure is presented where the parameter is increased in constant BC multiplier increment Δλ = 0.1 (the multiplier runs from 0 to 1 in 10 steps). The iterative solution procedure is controlled using the return value of the SMTConvergence command as follows:

In[505]:=
```
(*do 10 steps*)
nstep = 10; Δλ = 0.1; tolNR = 10^-8; maxNR = 15;
Do[
 (* increment the load level λ and time dependent data *)
 SMTNextStep["Δλ" → Δλ];
 (* repeate Newton iterations in infinite loop *)
 While[
  (* SMTConvergence returns: True if ||Δat||>10^-8, False if ||Δat||≤10^-8 and
     aborts the cacluation if the number of iterations has exceeded 10 *)
  SMTConvergence[tolNR, maxNR]
  (* the actual Newton iteration *)
  , SMTNewtonIteration[];
 ];
 , {i, 1, nstep}]
```

or shortly as

In[507]:=
```
nstep = 10; Δλ = 0.1; tolNR = 10^-8; maxNR = 15;
Do[
 SMTNextStep["Δλ" → Δλ];
 While[SMTConvergence[tolNR, maxNR], SMTNewtonIteration[];];
 , {i, 1, nstep}]
```

or with the report about the reasons for the failure to reach the target λ as

In[509]:=
```
nstep = 10; Δλ = 0.1; tolNR = 10^-8; maxNR = 15;
Do[
 SMTNextStep["Δλ" → Δλ];
 While[step = SMTConvergence[tolNR, maxNR, "Analyze"], SMTNewtonIteration[];];
 If[step =!= False, SMTStatusReport["Analyze"]; Abort[];];
 , {i, 1, nstep}]
```

Example: successive values of time

This is a path following procedure where the path is parameterized by time variable. The command SMTNextStep[“t”→t] sets the current time.

This is an iterative path following solution procedure with *nstep* time steps.

In[431]:=
```
tolNR = 10^-8; maxNR = 15; tmax = 10; nstep = 100;
Do[
 SMTNextStep["t" → t];
 While[step = SMTConvergence[tolNR, maxNR, "Analyze"], SMTNewtonIteration[];];
 If[step =!= False, SMTStatusReport["Analyze"]; Abort[];];
 , {t, tmax / nstep, tmax, tmax / nstep}]
```

This is an iterative path following solution procedure time steps Δt.

In[433]:=
```
tolNR = 10^-8; maxNR = 15; tmax = 10; Δt = 0.1;
Do[
 SMTNextStep["t" → t];
 While[step = SMTConvergence[tolNR, maxNR, "Analyze"], SMTNewtonIteration[];];
 If[step =!= False, SMTStatusReport["Analyze"]; Abort[];];
 , {t, Δt, tmax, Δt}]
```

This is an iterative path following solution procedure where the time takes successive values of 0.1,1.,13,50.

In[435]:=
```
tolNR = 10^-8; maxNR = 15;
Do[
 SMTNextStep["t" → t];
 While[step = SMTConvergence[tolNR, maxNR, "Analyze"], SMTNewtonIteration[];];
 If[step =!= False, SMTStatusReport["Analyze"]; Abort[];];
 , {t, {0.1, 1., 13, 50}}]
```

Example: adaptive BC multiplier

Within the adaptive path-following procedures only one parameter (λ, t or γ) can be a leading parameter of the problem. The return value of the SMTConvergence command can be used to control the convergence of the NR-iterations within one time step as shown in the previous example. With an additional argument {"Adaptive Time", *targetNR*, Δt_{min}, Δt_{max}, t_{max}} or {"Adaptive BC", *targetNR*, $\Delta\lambda_{min}$, $\Delta\lambda_{max}$, λ_{max}} or {"Adaptive γ", targetNR, $\Delta\gamma_{min}$, $\Delta\gamma_{max}$, γ_{max}} the SMTConvergence is able to control also path following procedure with an adaptive time or boundary conditions multiplier stepping.

This is an example of the path following procedure with an adaptive boundary conditions multiplier λ as a leading parameter. The multiplier runs from 0 to $\lambda_{max} = 10$ with an initial value $\lambda_0 = 0.1$, maximal increment $\Delta\lambda_{max} = 0.2$ and minimal increment $\Delta\lambda_{min} = 0.001$.The tolerance of the NR procedure *tolNR* is set to 10^-8, the maximum number of iterations *maxNR* to 10 and the desired number of iterations *targetNR* to 7.

The SMTConvergence command in this case returns True if $||\Delta \boldsymbol{at}||$>*tolNR* and a vector of four parameters

{*step_back*, *inc*, *step_forward*, *report*} otherwise.

If the first parameter *step_back* ≡ True then the iterative procedure in the current step has failed to converge, thus the values of all variables that define the state of the system has to be returned to the last converged state using the SMTStepBack command. If the third parameter *step_forward* ≡ True then the target value λ_{max} has not been achieved yet, thus an additional step has to be made using the SMTNextStep[1,*inc*] command and the second parameter *inc* as the suggested value of the multiplier increment $\Delta\lambda$. If the fourth parameter *report* ≡ "MinBound" then the proposed increment *inc* is less than prescribed minimum *inc*<$\Delta\lambda_{min}$, thus the path following procedure has failed. Parameter *step_forward* ≡ False in the case that the final goal of simulation have been reached and if the simulation has to be prematurely aborted (*inc*<$\Delta\lambda_{min}$).

In[437]:=
```
λMax = 10; λ0 = λMax / 100; ΔλMin = λMax / 1000; ΔλMax = λMax / 10;
tolNR = 10^-8; maxNR = 15; targetNR = 8;
SMTNextStep["λ" → λ0];
While[
  While[
   step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λMax}]
   , SMTNewtonIteration[];
  ];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" → step[[2]]]
 ];
SMTSimulationReport[];
```

Example: adaptive time and constant BC multiplier

This is a path following procedure with an adaptive time increment (time runs from 0 to 10 with an initial increment 0.1, maximal increment 0.2 and minimal increment 0.001). BC multiplier parameter has in this case a constant value $\lambda = 1$ set by SMTNextStep[“t”→ t0,”λ”→1] command.

In[442]:=
```
tMax = 10.; t0 = tMax / 100.; ΔtMin = tMax / 1000.; ΔtMax = tMax / 10.;
tolNR = 10.^-8; maxNR = 15; targetNR = 8;
SMTNextStep["t" → t0, "λ" → 1];
While[
  While[step = SMTConvergence[tolNR, maxNR, {"Adaptive Time", targetNR, ΔtMin, ΔtMax, tMax}],
   SMTNewtonIteration[];];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δt" → step[[2]]]
 ];
SMTSimulationReport[];
```

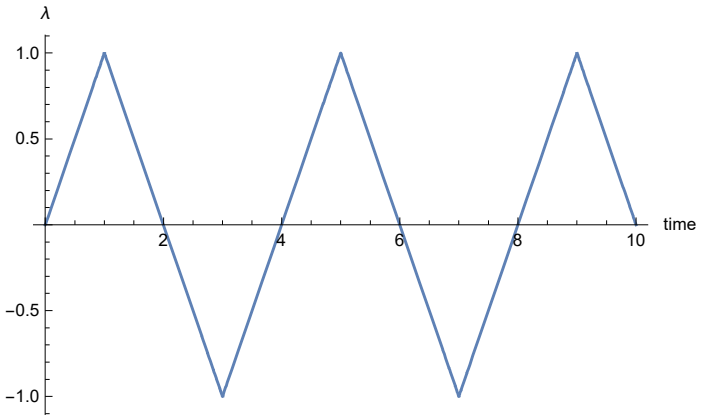
Example: simultaneous incrementation of time and BC multiplier

A special form of the SMTNextStep commands allows simultaneous incrementation of time and multiplier. The time parameter is in this case the leading parameter (independent parameter). Multiplier λ is defined as an explicit function of time $\lambda_f(t)$. The multiplier is automatically changed accordingly to the current value of the time parameter as follows:

SMTNextStep["t" → t, "λ[t]" → λ_f]

An example of a path following procedure with a zig-zag multiplier λ as a function of time is presented. The time runs from 0 to 10 in 100 steps while the multiplier follows the prescribed zig-zag pattern.

In[447]:=
```
λf[t_] := If[OddQ[Floor[(t + 1) / 2]], 1, -1] (2 Floor[(t + 1) / 2] - t);
Plot[λf[t], {t, 0, 10}, AxesLabel → {"time", "λ"}]
```



- with constant time stepping

In[449]:=
```
Do[
 SMTNextStep["Δt" → 0.1, "λ[t]" → λf];
 While[SMTConvergence[10^-8, 10], SMTNewtonIteration[];];
 , {i, 1, 100}]
```

- with adaptive time stepping

In[450]:=
```
tMax = 10.; t0 = tMax / 100.; ΔtMin = tMax / 1000.; ΔtMax = tMax / 10.;
tolNR = 10. ^ -8; maxNR = 15; targetNR = 8;
SMTNextStep["t" → t0, "λ[t]" → λf];
While[
  While[step = SMTConvergence[tolNR, maxNR, {"Adaptive Time", targetNR, ΔtMin, ΔtMax, tMax}],
   SMTNewtonIteration[];];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δt" → step[[2]], "λ[t]" → λf]
 ];
SMTSimulationReport[];
```

Example: adaptive BC multiplier with visualization

This is a path following procedure with an adaptive BC multiplier (the BC multiplier runs from 0 to 10 with an initial increment 0.1, maximal increment 0.2 and minimal increment 0.001).

- Deformed mesh is displayed after each completed increment into the post-processing window. The list of points *graph* is also collected during the analysis.

In[455]:=
```
λMax = 10.; λ0 = λMax / 100.; ΔλMin = λMax / 1000; ΔλMax = λMax / 10.;
tolNR = 10. ^ -8; maxNR = 15; targetNR = 8;
graph = {};
SMTNextStep["λ" → λ0];
While[
  While[step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λMax}],
   SMTNewtonIteration[];];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  If[Not[step[[1]]],
   SMTShowMesh["DeformedMesh" → True, "Show" -> "Window"];
   AppendTo[graph, {SMTData["Multiplier"], SMTPostData["Sxx", Point[{.2, .5}]]}];
  ];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" → step[[2]]]
 ];
ListLinePlot[graph]
```

- Simulation with automated animation of response (see also SMTAnimationOfResponse).

In[461]:=
```
λMax = 10.; λ0 = λMax / 100.; ΔλMin = λMax / 1000.; ΔλMax = λMax / 10.;
tolNR = 10. ^ -8; maxNR = 15; targetNR = 8;
SMTAnimationOfResponse["Initialize", {0, 0}];
SMTNextStep["λ" → λ0];
While[
  While[
   step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λMax}]
   , SMTNewtonIteration[];
  ];
  If[Not[step[[1]]], SMTAnimationOfResponse["LeadingNodePosition" → {0, L}]];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" → step[[2]]]
 ];
SMTAnimationOfResponse["Last frame"]
```

Example: Arc length procedure

First step of Arc-length procedure is using command SMTArcLengthSet[] which initializes properties of arc-length analysis. It returns an estimated arc-length for the target load level 3 on a assumption that the problem is linear. Terminal arc-length curve length *sMax*, maximum steps Δ*sMax,* minimum steps Δ*sMin* and initial increment of arc length *s0* are then chosen. Adaptive Time procedure with Arc-length iterations is used to solve described problem. Procedure is stopped when sMax is reached. After each successful step, SMTArcLengthNext[] must be called.

For more see: `Iterative Arc – length Solution Procedure`

In[466]:=
```
sMax = SMTArcLengthSet["λTarget" → 3];
ΔsMax = sMax / 20.; s0 = sMax / 100.; ΔsMin = sMax / 1000.;
tolNR = 10. ^ -8; maxNR = 15; targetNR = 8;
SMTNextStep["γ" → s0];
While[
  While[
   step = SMTConvergence[tolNR, maxNR, {"Adaptive γ", targetNR, ΔsMin, ΔsMax, sMax}]
   , SMTArcLengthIteration[];
  ];
  If[Not[step[[1]]], SMTShowMesh["DeformedMesh" → True, "Show" -> "Window"];];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  ,
  If[step[[1]], SMTStepBack[];, SMTArcLengthNext[];];
  SMTNextStep["Δγ" → step[[2]]]
 ];
SMTArcLengthFree[];
```

Analysis of Equilibrium Paths

SMTEquilibriumPaths

SMTEquilibriumPaths[*options*]
uses various iterative solution procedures to get response of parameterized system.

The `SMTEquilibriumPaths` function accepts also all options of the `SMTAnimationOfResponse` function.

The problem can be parameterized either by time (t), boundary conditions multiplier (*λ*) or a arc length parameter (*γ*). The terminal value of time (t) is given by option "tMax", boundary conditions multiplier (*λ*) by option "*λ*Max" or a arc length parameter(*γ*) by option "*γ*Max". Within the adaptive path-following procedures only one parameter can be a leading parameter of the problem, thus only one of options "tMax", "*λ*Max" or "*γ*Max" can be given.

option	default	description
"Method"	"Newton"	"Newton" ⇒ standard Newton-Raphson iterative method "Modified Newton" ⇒ modified Newton method (tangent matrix is evaluated and decomposed only in the first iteration) "Explicit" ⇒ incremental Euler one-step method "Arc Length" ⇒ arc length method has a separate set of options given in a table below (see also Implementation Notes for Arc-length solution procedure)
"Steps"	"Adaptive"	Option defines how the parameter is incremented. n ⇒ constant incrementation with n steps up to the terminal value "Adaptive" ⇒ if "λMax"=!=None then adaptive multiplier incrementation else if "tMax"=!= None adaptive time incrementation
problems parameterized with BC multiplier		
"λMax"	None	terminal BC multiplier
"Δλ Max"	λMax/10	the maximum increment of BC multiplier
"Δλ Min"	Δλ Max/100	minimum increment of BC multiplier
"Δλ0"	Δλ Max	initial increment of BC multiplier
problems parameterized with time		
"tMax"	None	terminal time
"ΔtMax"	tMax/10	the maximum increment time
"ΔtMin"	ΔtMax/100	minimum increment of time
"Δt0"	ΔtMax	initial increment of time
"λ[t]"	None	function that defines boundary conditions multiplier λ as a function of time t (λ←f[t]) (e.g. "λ[t]"->Function[{t},Sin[t]] or "λ[t]"->Sin)
options for iterative procedures		
"Tolerance"	10^-8	tolerance for the arc-length iterations
"MaxIterations"	15	maximum number of iterations
"OptimalIterations"	8	optimal number of iterations (step length is preserved constant)
"InitializeAnimationOfResponse"	False	False ⇒ Starts a new animation of response session. $\{x_0,y_0\}$ ⇒ Starts a new animation of response session with the first point of response curve set to $\{x_0,y_0\}$. (see SMTAnimationOfResponse)
SMTAnimationOfResponse options		

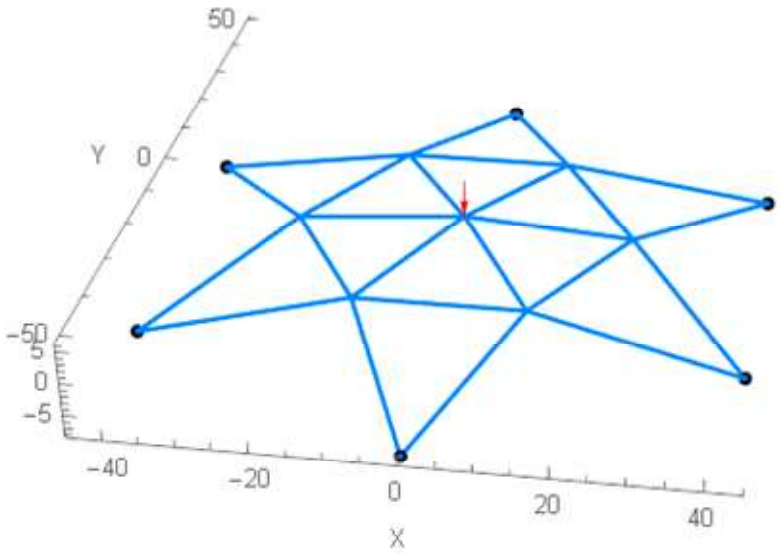
General options of the SMTEquilibriumPaths function for iterative solution algorithms.

option	default	description
"Method"->"Arc Length"		Algorithm follows all the branches of the response curve including bifurcations using arc length type of methods combined with effective branch switching algorithm.
"MaxPaths"	1	maximum number of equilibrium paths to follow (with "MaxPaths"->1 only the fundamental path is analysed, wit "MaxPaths"->2 the fundamental path and all the bifurcations that starts from the fundamental path are followed)
"γMax"	None	terminal arc-length curve length
"$\Delta\gamma$Max"	γMax/10	the maximum increment of the arc-length within a single time step
"$\Delta\gamma$Min"	$\Delta\gamma$Max/100	minimum increment of the arc-length within a single time step
"$\Delta\gamma$0"	$\Delta\gamma$Max	initial increment of the arc-length
"BifurcationPerturbation"	1	factor (α) used to get direction of an alternative branches after bifurcation (alternative direction $\Delta p=\alpha\,\Psi$)
"CriticalPointTolerance"	0.001	relative accuracy of the critical value (limit or bifurcation point) of the parameter of the response curve ($\Delta\gamma_{cr}/\gamma_{cr}<\epsilon$)
"BifurcationPointTolerance"	10^-5	tolerance for the detection of bifurcation point ($\|\Psi.P\|<\epsilon$)
"Tolerance"	10^-8	tolerance for the arc-length iterations
"MaxIterations"	15	maximum number of iterations
"OptimalIterations"	8	optimal number of iterations (step lenght is preserved constant)
"InitializeAnimationOfResponse"	False	False ⇒ Starts a new animation of response session. $\{x_0,y_0\}$ ⇒ Starts a new animation of response session with the first point of response curve set to $\{x_0,y_0\}$. (see `SMTAnimationOfResponse`)
SMTAnimationOfResponse options		

Options of the `SMTEquilibriumPaths` function for "Arc Length" solution algorithm.

Star dome truss example

Star dome truss example is the standard benchmark case for the assessment of the performance of the path following algorithms. Combination of the arc-length commands and the `SMTEquilibriumPaths` command is used here to get complete response curve of the structure. Stable segments of the curve are shown in blue color and unstable segments in red color. Additionally the limit and bifurcation points are marked in green and magenta color. Kinematically exact truss element is used made of elastic material.



Kinematically exact elastic truss element

In[91]:=
```
<< AceGen`;
SMSInitialize["ExamplesNonlinearTruss", "Environment" → "AceFEM"];
SMSTemplate["CDriver", "SMSTopology" → "C1", "SMSDOFGlobal" → 3, "SMSSymmetricTangent" → True,
  "SMSDomainDataNames" → {"E - elastic modulus", "A - area"}, "SMSDefaultData" → {21000, 1}];
SMSStandardModule["Tangent and residual"];
{rA, rB} ⊨ SMSIO["Nodal coordinates"];
{Em, A} ⊨ SMSIO["Domain data"];
{uA, uB} ⊨ SMSIO["Nodal DOFs"];
pe ⊨ Flatten[{uA, uB}];
d0 ⊨ rB - rA;
d ⊨ d0 + uB - uA;
l0 ⊨ SMSSqrt[d0.d0];
l ⊨ SMSSqrt[d.d];
De ⊨ (l - l0)/l0;
Π ⊨ A l0 1/2 Em De^2;
Rel ⊨ SMSD[Π, pe];
Kel ⊨ SMSD[Rel, pe];
SMSIO[Rel, "Add to", "Residual"];
SMSIO[Kel, "Add to", "Tangent"];
SMSStandardModule["Postprocessing"];
uIO ⊨ SMSIO["Nodal DOFs"];
SMSIO[{"DeformedMeshX" -> uIO[[All, 1]],
   "DeformedMeshY" -> uIO[[All, 2]], "DeformedMeshZ" -> uIO[[All, 3]], "u" -> uIO[[All, 1]],
   "v" -> uIO[[All, 2]], "w" -> uIO[[All, 3]]}, "Export to", "Nodal point post"];
SMSWrite[];
```

```
File: ExamplesNonlinearTruss.c  Size: 6088  Time: 1
```

Method	SKR	SPP
No.Formulae	25	7
No.Leafs	746	152

Star dome truss FE input data

In[562]:=
```
<< AceFEM`;
Acs = 0.1; (*cm^2*)
Em = 2.034*10^7*10^-3; (*kN/cm^2*)
PLoad = -1; a = 43.3; b = 25.;
c = 8.216; d = 2.; e = a/Cos[π/6];
nodes = {{1, 0., 0., 0.}, {2, b Cos[π/3], b Sin[π/3], -d}, {3, b, 0, -d},
    {4, b Cos[π/3], -b Sin[π/3], -d}, {5, -b Cos[π/3], -b Sin[π/3], -d},
    {6, -b, 0, -d}, {7, -b Cos[π/3], b Sin[π/3], -d}, {8, 0, e, -c},
    {9, a, e Sin[π/6], -c}, {10, a, -e Sin[π/6], -c}, {11, 0, -e, -c},
    {12, -a, -e Sin[π/6], -c}, {13, -a, e Sin[π/6], -c}};
elements = {{1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}, {1, 7}, {2, 7}, {2, 3},
    {3, 4}, {4, 5}, {5, 6}, {6, 7}, {7, 8}, {2, 8}, {2, 9}, {3, 9}, {3, 10},
    {4, 10}, {4, 11}, {5, 11}, {5, 12}, {6, 12}, {6, 13}, {7, 13}};
SMTInputData[];
SMTAddDomain["A", "ExamplesNonlinearTruss", {"E - elastic modulus" → Em, "A - area" → Acs}];
SMTAddMesh["A", nodes, elements];
SMTAddEssentialBoundary["Z" == -c &, 1 → 0, 2 → 0, 3 → 0];
SMTAddNaturalBoundary[Point[{0, 0, 0}], 3 → PLoad];
SMTAnalysis[];
```

Initialization of the arc-length procedure and iterative procedure with adaptive parameter incrementation

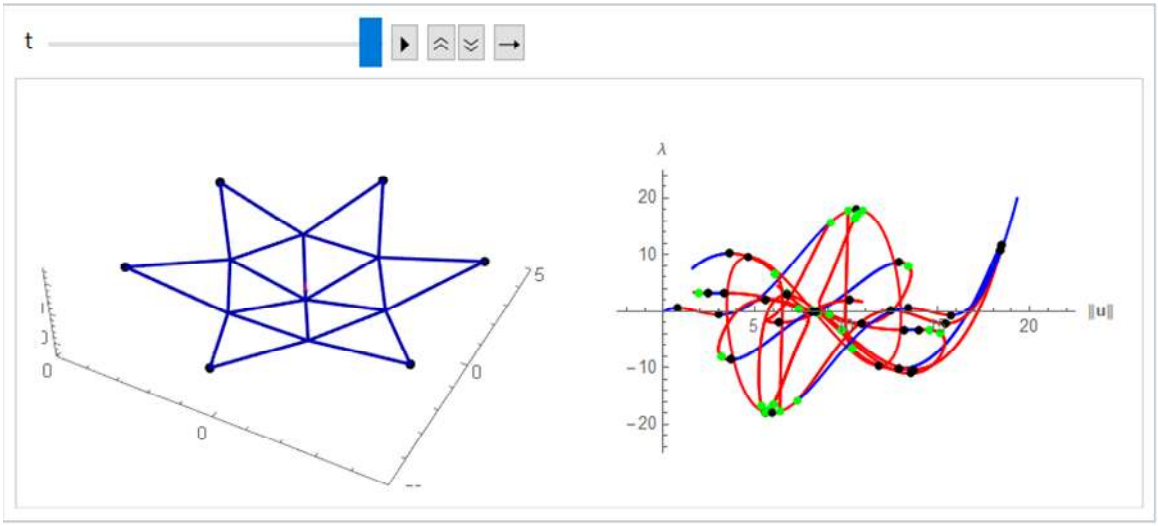
- Not that due to the simple example, the rendering the graphics takes the majority of the simulation time!!

In[575]:=
```
(* ocena *)
γMax = SMTArcLengthSet["λTarget" → 120];
```

In[576]:=
```
SMTEquilibriumPaths[
 "Method" -> "Arc Length"
 , "γMax" -> γMax
 , "Δγ0" -> γMax / 200
 , "ΔγMax" -> γMax / 200
 , "ΔγMin" -> γMax / 1000
 , "MaxPaths" → 2
 , "LeadingNodePosition" → {0., 0., 0.}
 , "ImageSize" → 300, "PlotMarkers" → False, "AnimationFrequency" → γMax / 200
 , "ShowMeshOptions" → {PlotRange → {{-50, 50}, {-50, 50}, {-25, 5}}}
 , "PlotOptions" → {PlotRange → {{0, 20}, {-20, 20}}}
 , "Show" → {"ExportFrames", "trussdome"}
 , "InitializeAnimationOfResponse" → {0, 0}
]
```

- Animation is displayed with the default 10 frames/second.

In[577]:=
```
SMTAnimationOfResponse["Animate"]
```



Utility Functions

SMTStatusReport

SMTStatusReport[]
prints out the report of the current status of the system

SMTStatusReport[*expr*]
prints out the report of the current status of the system tagged by the arbitrary expression *expr*

SMTStatusReport["Analyze"]
prints out extended report

- This prints out the report of the current status of the system and the current values of all degrees of freedom in node 5.

In[91]:=
```
SMTStatusReport[SMTNodeData[5, "at"]]
```

See Also

- `Iterative Solution Procedures, SMTErrorCheck`

SMTSessionTime

SMTSessionTime[]
gives the total number of seconds of real time that have elapsed since the beginning of your *AceFEM* session

SMTSimulationReport

SMTSimulationReport[]
prints out report and returns a list of rules identifying the percentage of time spent in specific tasks during the analysis

SMTSimulationReport[*comments*]
prints out report with additional *comments* and returns a list of rules identifying the percentage of time spent in specific tasks during the analysis

SMTSimulationReport[False]
returns only a list of rules identifying the percentage of time spent in specific tasks during the analysis

SMTSimulationReport[*idata_names*,*rdata_names*,*comments, options*]

prints out additional information stored in user defined integer and real type environment variables and arbitrary comments (for details see `Code Profiling`).

option	default	description
"Output"	{"Console","File"}	possible output devices: "Console" ⇒ current notebook "File" ⇒ current output file (if defined)

Options of the *SMTSimulationReport* function.

- Example

In[31]:=
```
SMTSimulationReport[];
```

No. of nodes	451
No. of elements	100
No. of equations	880
Number of threads used/max	8/8
Data memory (KBytes)	149
Tangent matrix (KBytes)	105
Solver memory (KBytes)	1040
Mathematica memory (KBytes)	63 902
Total memory (KBytes)	65 196
No. of steps	14
No. of steps back	0
Step efficiency (%)	100.
Total no. of iterations	104
Average iterations/step	7.42857
Terminal time (t)	0.
Terminal BC multiplier (λ)	10.
Terminal parameter (γ)	0.

Total absolute time (s)	529.2195632
Total driver time (s)	529.028
Total driver time (%)	99.9638
Total linear solver time (s)	0.0559988
Total linear solver time (%)	0.0105852
Total K&R time (s)	0.0640016
Total K&R time (%)	0.012098
Average time/iteration (s)	5.08681
Average linear solver time (s)	0.00053845
Average Ke&Re time (s)	6.154×10^{-6}
Solver type	Pardiso
Matrix type	-2
CPU Mathematica time (s)	1.14
CPU Mathematica time (%)	0.215412
USER-IData	
USER-RData	

See Also: `Simple bending of the column`, `SMTErrorCheck`

SMTSetSolver

SMTSetSolver[*solverID*]
update all structures related to the global tangent matrix and number of equations accordingly to the value of parameter *solverID*

SMTSetSolver[]
reset linear solver related structures using initialization from the last call

solverID	description
0	appropriate solver is chosen automatically
1	standard LU profile unsymmetrical solver (no user parameters defined)
2	standard LDL profile symmetric solver (no user parameters defined)
3	**NOT A PART OF THE STANDARD DISTRIBUTION !!!** **SuperLU** solver SMTSetSolver[4,{*ordering, work_allocation*},{*pivot_tresh, fill*}]
4	**NOT A PART OF THE STANDARD DISTRIBUTION !!!** **UMFPACK** solver
5	**MKL Direct Sparse Solver** **– PARDISO** solver from INTEL MKL (see MKL Direct Sparse Solver)
6	**MKL Iterative Sparse Solver** (see MKL Iterative Sparse Solver)
None	only basic, solver independent structures are updated

Linear solver sets the number of negative pivots (or - 1 if data is not available) to Integer Type Environment variable "NegativePivots" and the number of near-zero pivots to variable "ZeroPivots". The data can be accessed through the SMTIData["NegativePivots"] and SMTIData[""ZeroPivots"] commands. The the number of negative pivots (SMTIData["NegativePivots"]) is only available for the standard LU and LDL solvers and for the PARDISO mtype=-2 solver!!.

MKL Direct Sparse Solver - PARDISO

option	default value	description
"MatrixType"	Automatic	1 - real and structurally symmetric matrix (partial pivoting) 2 - real and symmetric positive defini matrix -2 - real and symmetric indefinite matrix (SMTIData["NegativePivots"] available) 3 - complex and structurally symmetric matrix 4 - complex and Hermitian positive definite matrix -4 - complex and Hermitian indefinite matrix 6 - complex and symmetric matrix 11 - real and unsymmetrical matrix (full pivoting) 13 - complex and unsymmetrical matrix The default matrix type depends on the element specification `SMSSymmetricTangent` and node identifications (Node Identification) -L and -AL.
"IntegerParameters "	Automatic	{{$index_1$,$value_1$},{$index_2$,$value_2$},...} where $index_i$ is an index of the parameter accordingly to the PARDISO documentation and $value_i$ is the choosen value of the parameter

Options of the INTEL MKL PARDISO direct solver.

For examples are and details see `MKL Direct Sparse Solver`.

MKL Iterative Sparse Solver

option	default value	description
"MatrixType"	Automatic	1 - real and structurally symmetric matrix (partial pivoting) 2 - real and symmetric positive definite matrix -2 - real and symmetric indefinite matrix (SMTIData["NegativePivots"] available) 3 - complex and structurally symmetric matrix 4 - complex and Hermitian positive definite matrix -4 - complex and Hermitian indefinite matrix 6 - complex and symmetric matrix 11 - real and unsymmetrical matrix (full pivoting) 13 - complex and unsymmetrical matrix The default matrix type depends on the element specification `SMSSymmetricTangent` and node identifications (Node Identification) -L and -AL.
"IterativeSolverType"	2	1 - FGMRES 2 - CG
"Preconditioner"	1	0 - no preconditioner 1 - ILUO 2 - ILUT 3 - Jacobi
"MaxNoIterations"	10^4	maximum number of iterations
"MaxFillInRatio"	2	maximum nonzero terms in a row of the incomplitly factorized matrix should be "MaxFillInRatio"*a*verage number of nonzero terms in a row* (ILUT only option)
"MaxFillInElements"	0	maximum nonzero terms in a row of the incompletely factorized matrix (when given the "MaxFillInRatio" option is ignored) (ILUT only option)
"ResidualErrorTolerance"	10^{-6}	tolerance
"VectorNormToleranceTol"	10^{-12}	tolerance (FGMRES only option)
"PreconditionerTol"	10^{-6}	tolerance (ILUT only option)
"IntegerParameters"	Automatic	$\{\{index_1,value_1\},\{index_2,value_2\},\ldots\}$ where $index_i$ is an index of the parameter accordingly to the Intel MKL documentation and $value_i$ is the choosen value of the parameter
"RealParameters"	Automatic	$\{\{index_1,value_1\},\{index_2,value_2\},\ldots\}$ where $index_i$ is an index of the parameter accordingly to the Intel MKL documentation and $value_i$ is the choosen value of the parameter

Options of the ITERATIVE INTEL MKL solver.

For examples are and details see `MKL Iterative Sparse Solver`.

SMTErrorCheck

A set of integer type environment variables:

"MissingSubroutine", "SubDivergence", "ElementState", "ElementShape", "MaterialState" and "ErrorStatus"

named *error state variables* give more information about the state of the simulation and its eventual failure (see also `Integer Type Environment Data`). The SMTErrorCheck function can be used during the *AceFEM* session to process the information stored in *error state variables*. In the case of error event the error message is produced and all *error state variables* are set back to zero value.

SMTErrorCheck[*console,file,report,notebook*]

check for error events and print error messages on:

console≠0 ⇒ console driver window (if opened)

file≠0 ⇒ output file (if opened)

report≠0 ⇒ report window

notebook≠0 ⇒ current notebook

SMTErrorCheck[] ≡ SMTErrorCheck[1,1,1,0]

The environment variable "ErrorStatus" identifies the general error status:

Error status	description
0	no special events were detected during the session
1	warnings were detected during the session, (evaluation is still performed in a regular way, time step cutting is recommended)
2	fatal errors were detected during the session (time step cutting is necessary)
3	fatal error (terminate the process)

Codes for the error status switch.

The *error state variables* are set or increased from the user element subroutines. The user can therefore control the iterative solution procedure by setting appropriate environment variables. The *error state variables* should be increased by 1 whenever the error condition that specifies specific event appears.

- Here is a part of the AceGen input where the error event is reported and the idata$$ variable "ElementShape" increased by 1 whenever the Jacobean of the nonlinear coordinate mapping (J) becomes negative.

```
In[472]:= SMSIf[J ≤ 10^-9];
            SMSExport[1, idata$$["ErrorStatus"]];
            SMSExport[SMSInteger[idata$$["ElementShape"]] + 1, idata$$["ElementShape"]];
          SMSEndIf[];
```

SMTPrintToConsole

SMTPrintToConsole[*input...*]

transforms an arbutrary *input* into string and prints the string to standard console window

- Windows

  On Windows application can be started in a separate window with the SMTInputData option "Console"->True.

```
In[142]:= SMTPrintToConsole["\n\n\nIt is working ", 100, "%"]
```

- Mac OS

  In order to see printouts on Mac *Mathematica* has to be started from the terminal as follows:

  - open Terminal (look under Applications-Utilities for Terminal.app),
  - start *Mathematica* form Terminal (e.g. /Applications/Mathematica.app/Contents/MacOS/*Mathematica*),
  - messages will now be printed to terminal window.

- Linux

  In order to see printouts on Linux *Mathematica* has to be started from the terminal as follows:

  - open Terminal (e.g. search for Terminal),
  - start *Mathematica* form Terminal (e.g.*Mathematica* &),
  - messages will now be printed to terminal window.

Visualization and Post-processing Phase

Contents

Visualization

SMTShowMesh

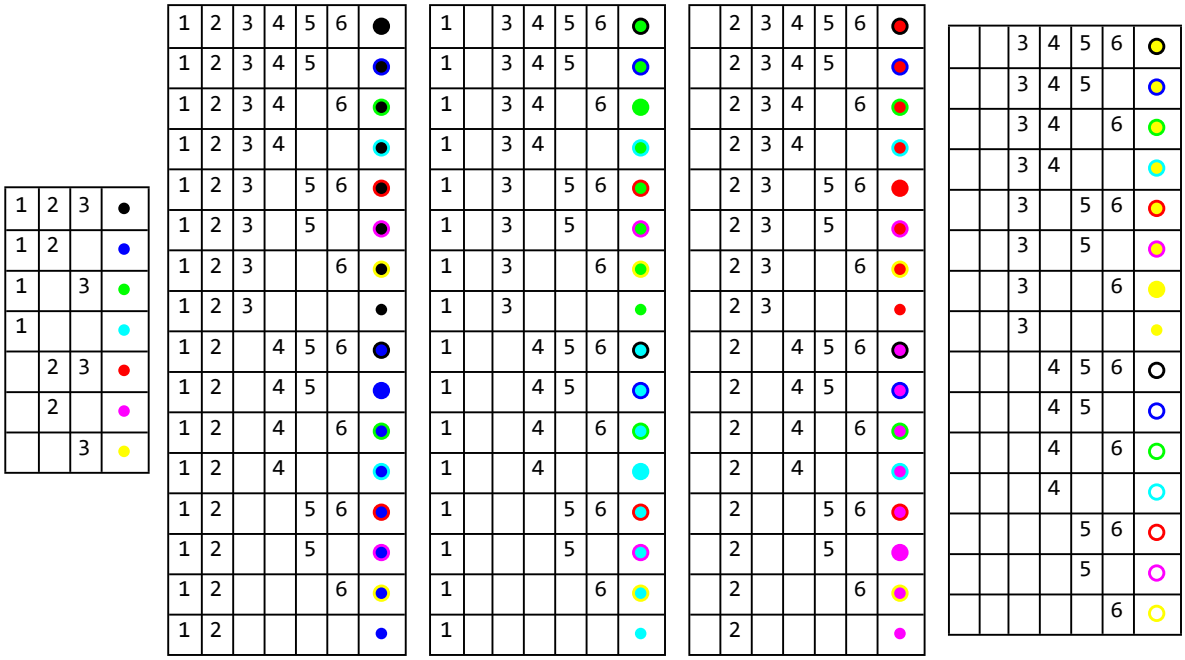
SMTShowMesh[*options*]
display two- or three-dimensional graphics using options specified and return graphics object

option	default	description
"Show"	True|False	specifies output device
"Label"	None	an expression or the list of expressions to be printed as a label for the plot
"Mesh"	True	display mesh as wire frame accordingly to the mesh type Possible mesh types include True, False, Automatic, RGBColor[*red,green,blue*] , GreyLevel[*level*].
"Marks"	False	display nodal and element numbers
"BoundaryConditio-ns"	False	mark nodes with the non-zero boundary conditions
"Domains"	Automatic	list of the domain identifications (only the domains with the domain identification included in the list are plotted) All ⇒ all domains $\{dID_1,\ldots,dID_n\}$ ⇒ list of domain identifications Automatic ⇒ if option "Field"-> *keyword* is given then only domains are selected that have defined given post-processing keyword
"FillElements"	True	fill in the element surfaces with the element surface color or with the contour lines (possible values True|False|RGBColor|Hue[i]|ColorData[i]| ColorDataFunction[i]×GrayLevel[i] where i=1,2,nespec)
"Field"	False	the vector of nodal values that defines scalar field used to calculate element surface color or contour lines
"Contour"	False	display contour lines of the scalar field $\boldsymbol{p}$ defined by the "Field" option
"Legend"	True	include legend specifying the colors and the range of the "Field" values
"DeformedMesh"	False	display deformed mesh by adding the vector field $\boldsymbol{u}$ multiplied by the "Scale" option to the nodal coordinates $\boldsymbol{X}$ ($\tilde{\boldsymbol{X}}:=\boldsymbol{X}+\boldsymbol{u}$)
"Scale"	1.	scaling factor for deformed mesh
"Opacity"	False	graphics directive which specifies that graphical objects which follow are to be displayed, if possible,with given opacity (number between 0 and 1)
"User"	{}	user supplied graphic primitives (e.g. {Circle[{0,0},1]}). The coordinate system for the user graphics is the same as the coordinate system of the structure!
"Combine"	(#&)	apply arbitrary function on the results of the SMTShowMesh command and return the results
"TimeFrequency"	0	the SMTShowMesh command is not executed if the absolute difference in time for two successive SMTShowMesh calls is less than "TimeFrequency"
"MultiplierFrequency"	0	the SMTShowMesh command is not executed if the absolute difference in multiplier for two successive SMTShowMesh calls is less than "MultiplierFrequency"
"StepFrequency"	0	the SMTShowMesh command is not executed if the absolute difference in step number for two successive SMTShowMesh calls is less than "StepFrequency"
"Rasterize"	False	False ⇒ original graphic is exported as video frames True ⇒ graphics is Rasterized before the video frames are stored (complex 3 D images can result in highe video frames files) {options} ⇒ options of the Rasterize command
"FullWireFrame"	False	show wire frame including interior elements

Options of the *SMTShowMesh* function.

The nodes with the non-zero essential boundary condition are marked with color points as presented in a table below.

The nodes with the non-zero natural boundary condition are marked with arrow for the nodes with the number of unknowns equal to the spatial dimension and with color point otherwise. Before the analysis the **Bp** and **dB** boundary values (see also `Input Data Phase`) are displayed separately, and during the analysis only **Bt** is displayed.



Coloring of the nodes accordingly to the applied essential boundary condition.

"Show" *option*	description
"Show"→True	displays graphics as a new notebook cell
"Show"→False	the graphics object is returned, but no display is generated
"Show"→"Window"	displays graphics in a separate post–processing notebook
"Show"→ {"ExportFrames", *keyword, options*}	creates HDF5 file with the name *keyword.h5* and exports current graphics as HDF5 datasets entry "frame_frameNumber" (frame numbers are counted automatically and *options* are the same as for command Export HDF5 files, result can be animated with SMTAnimationOfResponse["Animate", *keyword*], animated gif file can be created with SMTAnimationOfResponse["Export to","gif",*keyword*]) With an additional options "RealTime"→ *time* we specify real time of the frame used in animation. With an additional options "Data"→ *data* we specify an arbitrary data attached to the frame that can be rerived with the SMTAnimationOfResponse["Animate", *keyword*] command.
"Show"→ {"Export",*file*, *format, options*}	≡ Export[*file*,produced graphics, *format, options*] export data to a file, converting it to a specified format (see also Export command)
"Show"→"Window" | False	show options can be combined (e.g. "Window" | False displays graphics into post–processing notebook and returns graphics object)

Methods for output device.

"Contour" *option*	description
"Contour"–>False	color graph using vertex colors
"Contour"–>{False,*min*,*max*,*ndiv*}	color graph using vertex colors taken from the range {*min*,*max*} and show legend with *ndiv* divisions
"Contour"–>True	display 10 contour lines
"Contour"→*n*	display *n* contour lines
"Contour"→{*min*,*max*,*n*}	display *n* contour lines taken from the range {*min*,*max*}

Methods for setting up contour lines.

"Legend" *option*	description
"Legend"→True	display legend with default number of divisions
"Legend"→*ndiv*	display legend with *ndiv* divisions
"Legend"→"MinMax"	display only minimum and maximum values
"Legend"→False	no legend

Methods for setting up plot legend.

"Label" *option*	description
"Label"–>Automatic	display min. and max. nodal value
"Label"→None	no label
"Label"→expression	give an overall label for the plot

Methods for setting up plot label.

"DeformedMesh" *option*	description
"DeformedMesh"→True	Original finite element mesh is deformed as follows $\mathbf{X}_{\text{new}}=\mathbf{X}+\mathbf{u}_X$ $\mathbf{Y}_{\text{new}}=\mathbf{Y}+\mathbf{u}_Y$ $\mathbf{Z}_{\text{new}}=\mathbf{Z}+\mathbf{u}_Z$ where the displacement vectors $\mathbf{u}_X$, $\mathbf{u}_Y$, $\mathbf{u}_Z$ are by default obtained using the *SMTPostData* (see `SMTPostData`) command. If the "DeformedMeshX", "DeformedMeshY" and "DeformedMeshZ" post–processing codes are specified by the user then $\mathbf{u}_X$=SMTPostData["DeformedMeshX"], $\mathbf{u}_Y$=SMTPostData["DeformedMeshY"], $\mathbf{u}_Z$=SMTPostData["DeformedMeshZ"], else $\mathbf{u}_X$=SMTPostData[{"at",1}], $\mathbf{u}_Y$=SMTPostData[{"at",2}], $\mathbf{u}_Z$=SMTPostData[{"at",3}].
"DeformedMesh"→{$\mathbf{u}_X$, $\mathbf{u}_Y$, $\mathbf{u}_Z$}	user defined displacement vectors $\mathbf{u}_X$, $\mathbf{u}_Y$, $\mathbf{u}_Z$

Methods for setting up deformed mesh.

"Marks" *option*	description
"Marks"–>True	display nodal and element numbers ≡ "Marks"–>{"NodeNumber","ElementNumber"}
"Marks"→False	no numbers
"Marks"→"ElementNumber"	display element numbers
"Marks"→"NodeNumber"	display node numbers

Methods for setting up contour lines.

option	default	description
"TextStyle"	{FontSize→9, FontFamily→ "Arial"}	specifies the default style and font options with which text should be rendered
"NodeMarks"->*r*	False	mark all the nodes with the circle with radius *r*
"NodeTagOffset"	{.01,.01,.01}	position of the node number relative to the node
"NodeID"	Automatic	list of the node identifications (only the nodes with the node identification included in the list are plotted)
"ColorFunction"	Automatic	a function to apply to the values of scalar field ***p*** defined by the "Field" option to determine the color to use for a particular contour region (e.g. "ColorFunction"→Function[{x},ColorData["GrayTones"][x]])
"ZoomNodes"	False	the parameter is used to select nodes (it has one of the forms described in section `Selecting Nodes`, only the elements with all nodes selected are depicted)
"ZoomElements"	False	the parameter is used to select elements (it has one of the forms described in section `Selecting Elements`, only the selected elements are depicted)
"NodeMarksField"	False	the vector of nodal values that are used to calculate color for each nodal point mark

"RawOutput"	False	instead of the graphics return raw data as follows:
		{
		vertex coordinates,
		vertex field values,
		vertex colors that correspond to vertex values,
		legend graphics object,
		complete graphics object
		}

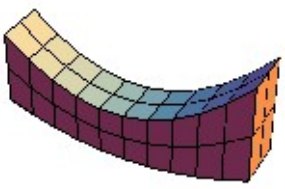
"ShowFor"	Automatic	specify a list of rules that transforms symbolic expressions (e.g. for nodal coordinates) into numerical values (option is *MDriver* specific, see also Semi-analytical Solutions)

Additional options for the *SMTShowMesh* function.

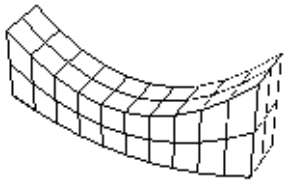
The *SMTShowMesh* function is the main post-processing function for displaying the element mesh, the boundary conditions and arbitrary post-processing quantities. The vectors of nodal values ***p***, ***n***, $\mathbf{u}_X$, $\mathbf{u}_Y$, $\mathbf{u}_Z$ are arbitrary vectors of *NoNodes* numbers (see also `SMTPostData`). Several examples are presented in Post processing (3D heat conduction).

examples

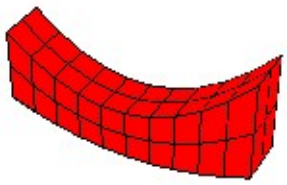
SMTShowMesh[]



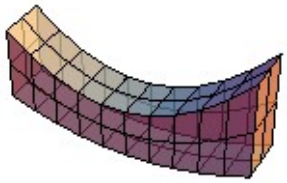
```
SMTShowMesh["FillElements"→White,
Lighting→{{"Ambient",White}}]
```



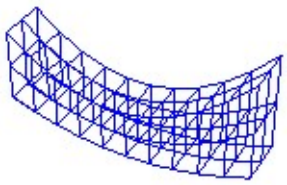
```
SMTShowMesh["FillElements"→Red,
Lighting→{{"Ambient",White}}]
```



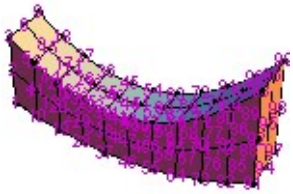
```
SMTShowMesh["Opacity"→0.7]
```



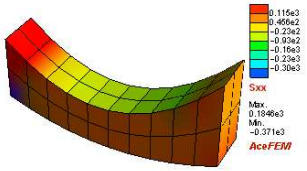
```
SMTShowMesh[
"FillElements"→False,"Mesh"→Blue]
```



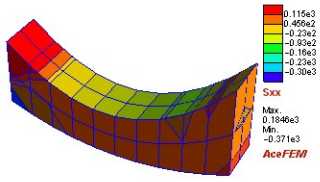
```
SMTShowMesh["Marks"→"NodeNumber"]
```



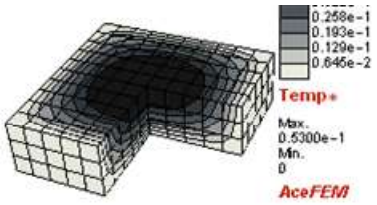
```
SMTShowMesh["Field"→"Sxx"]
```



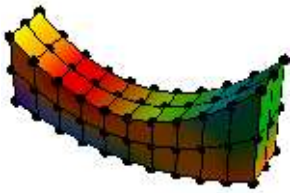

```
SMTShowMesh[
"Field"→"Sxx","Contour"→True]
```



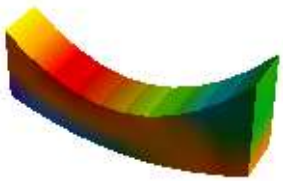
```
SMTShowMesh[
"Field"->"Temp*","Mesh"→Black,
"Contour"→4,"ZoomNodes"→
("Z"<=0.3 && ("X"<=0 || "Y"≥0)&),
"ColorFunction"→Function[{x},
 ColorData["GrayTones"][1-x]],
Lighting→{{"Ambient",White}}]
```



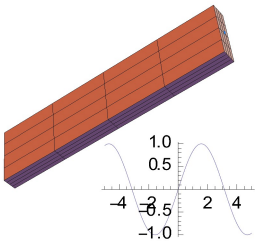

```
SMTShowMesh["Field"→"Szz",
"NodeMarks"→6,"Legend"→False]
```



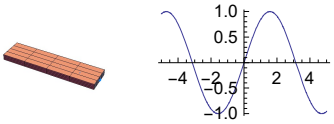
```
SMTShowMesh["Field"→"Szz",
"Mesh"→False,"Legend"→False]
```



```
SMTShowMesh[
Epilog->Inset[Plot[Sin[x],{x,-5,5}],
 {Right,Bottom},{Right,Bottom}] ]
```




```
SMTShowMesh[
"Combine"→(GraphicsRow[
 {#,Plot[Sin[x],{x,-5,5}]}]&)]
```



SMTUpdatePostData

SMTUpdatePostData[]

SMTUpdatePostData command transfers the mesh data stored in CDriver from CDriver to Mathematica.

The simulation data is shared among the CDriver and Mathematica. In order to save the memory only selected data that is needed to create the graphics is stored in Mathematica. This data has to be updated whenever the node coordinates or element connectivity have been changed during the simulation. Otherwise the graphics will not be correct.

SMTAnimationOfResponse

SMTAnimationOfResponse[*options*]

enables active visualization of the path following procedure accordingly to the given options

option	default	description
"x"	$\|\|\mathbf{u}\|\|$ in leading node	an expression for x axis of the response curve y(x) (e.g. Hold[SMTRData["Time"]] defines time for x-axis)
"y"	leading parameter (λ, t or γ)	an expression for y axis of the response curve y(x) (e.g. Hold[Total[SMTResidual[Line[{{0,L},{0.5,L}}]]][[2]]] defines the reaction force at line specified for y-axis)
"LeadingNodePositi-on"	None	the norm of DOF-s in leading node is the default value for the x-axis of the response curve y(x)
"PlotOptions"	{}	an additional options for the response curve plot (see ListLinePlot)
"ShowMeshOptions"	{}	an additional options for the mesh plot (see SMTShowMesh)
"Include"	Row[{"ShowMesh", "Plot"}," "]	defines what is actually displayed in a separate post-processing notebook. Two keywords can be used: "ShowMesh" is replaced by the results of the SMTShowMesh[*ShowMeshOptions*] command "Plot" is replaced by the response curve ListLinePlot[*points,PlotOptions*]
"Show"	"Window"	for all options see SMTShowMesh "Window" ⇒ displays graphics in a separate post-processing notebook {"ExportFrames", *framesFile*} ⇒ only frames are exported to framesFile.h5 file "Window"| {"ExportFrames", *framesFile*} ⇒ both
"PlotRange"	Automatic	plot range {{xmin,xmax},{ymin,ymax}}
"Report"	False	returns the last frame as graphic object
"AnimationFrequen-cy"	Automatic	The number of steps recoded can be less than the actual number of steps performed. Option "AnimationFrequency" defines with which frequency the output is generated. Automatic ⇒ all steps Hold[exp] ⇒ output is generated whenever *exp* evaluates to True *dp* ⇒ increment of the leading parameter for which the output is generated
"PlotFrequency"	All	All ⇒ response curve plot contains all calculated points "AnimationFrequency" ⇒ the response curve plot contains only points from recorded frames
"ImageSize"	Automatic	total size of the image (see also ImageSize)
"PlotMarkers"	Automatic	see PlotMarkers
"NewStartingPoint"	False	start a new response curve with the given starting point (final plot can can contain several response curves)
"MarkPointOnly"	False	include plot marker only, curve is left unchanged
"PlotStyle"	Automatic	see PlotStyle Automatic ⇒ Blue
"Locator"	(Locator[#]&)	include Locator at the end of the response curve False ⇒ locator is omitted
"RealTime"	Automatic	Automatic ⇒ real time is 10 frames/second Hold[*exp*] ⇒ real time in seconds is evaluated *exp*
"Data"	Null	store an arbitrary additional data for each frame
"CriticalPoint"	Null	idetify the nature of equilibrium point: Null ⇒ ordinary equilibrium point "Bifurcation" ⇒ bifurcation equilibrium point "Limit" ⇒ limit equilibrium point "Critical" ⇒ critical point of unknown type

Options of the *SMTAnimationOfResponse* function.

Specific command for post-processing of data created by path-following procedure

SMTAnimationOfResponse["Initialize"]

Starts a new animation of response session.

SMTAnimationOfResponse["Initialize",{x_0,y_0}]

Starts a new animation of response session with the first point of response curve set to $\{x_0,y_0\}$.

SMTAnimationOfResponse["CriticalPoints"]
SMTAnimationOfResponse["CriticalPoints","Report"]
SMTAnimationOfResponse["CriticalPoints","λ"]
SMTAnimationOfResponse["CriticalPoints","t"]
SMTAnimationOfResponse["CriticalPoints","γ"]
returns information about critical points

General commands for post-processing of data stored as frames

SMTAnimationOfResponse["Frame"]
SMTAnimationOfResponse["Frame", *framesFile*]
SMTAnimationOfResponse["Last frame"] ≡ SMTAnimationOfResponse["Frame","FrameSelection"->-1]
returns frames stored accordingly to given options

SMTAnimationOfResponse["Animate"]
SMTAnimationOfResponse["Animate", *framesFile*]
creates a cell with controls added to allow interactive manipulation of animation within the notebook

Animation is generated from the previously stored frames from the last simulation or from the frames stored in file *framesFile.h5*. Command accepts all options of Animate command.

SMTAnimationOfResponse["Additional data"]
SMTAnimationOfResponse["Additional data", *framesFile*]
returns additional data stored for each frame (SMTAnimationOfResponse or SMTMesh "Data" option) accordingly to options

SMTAnimationOfResponse["Response"]
SMTAnimationOfResponse["Response", *framesFile*]
returns a list of points forming the response curve (if stored)

SMTAnimationOfResponse["Frame data","Frame size", *framesFile*]
SMTAnimationOfResponse["Frame data","Frame time", *framesFile*]

SMTAnimationOfResponse["All data"]
SMTAnimationOfResponse["All data", *framesFile*]
returns all data stored for each frame accordingly to options as follows

frameData={

1-"Step",

2-"Iteration",

3-"Multiplier",

4-"Time",

5-"Parameter",

6-"MultiplierIncrement",

7-"TimeIncrement",

8-"ParameterIncrement",

9-{plot marker, point type "Bifurcation" "Limit" "Critical" or Null} for point on response curve if present,

10-{x,y} on response curve if present,

11-real time for animation,

12-additional data for each frame (SMTAnimationOfResponse or SMTMesh "Data" option)

13 - AbsoluteTime[]-sessionStartTime - clock on the wall time

13 - frame size

}

option	default	description
"RealTime"	None	returns the frame nearest to given real time
"FrameSelection"	All	-1 ⇒ returns last frame All ⇒ returns all frames {f1,f2,...} ⇒ returns selected frames

Options of SMTAnimationOfResponse["Frame",...] SMTAnimationOfResponse["Data",...] function.

SMTAnimationOfResponse["Export to", *format*]

SMTAnimationOfResponse["Export to", *framesFile, format*]

generates an animated GIF file from previously stored frames. *frames* can be a list of frames or the name of *framesFile*.h5 file with the frames.

The name of the animation file is *framesFile.gif*. See also `Export` for the detailed set of all available formats and options.

option	default	description
"TimeScaling"	1	Real time can be scaled for a given factor
"FirstFrame"	"LastFrame"	frame showed before the start of animation "LastFrame", "EmptyFrame", given frame
"FrameSelection"	All	animation from selected set of frames
all Export options		

Options of SMTAnimationOfResponse["Export to",...] function.

SMTAnimationOfResponse["Save frame to", *framesFile, graphics*]

Adds *graphics* to the *framesFile.h5* file as an additional frame. With an additional options "RealTime"→time we specify real time of the frame and with "Data"→*data* we specify an arbitrary data attached to the frame.

SMTAnimationOfResponse["Number of frames"]

SMTAnimationOfResponse["Number of frames", *framesFile*]

returns number of frames stored

SMTAnimationOfResponse["Show frames", *framesFile*]

choose frames interactively

SMTAnimationOfResponse["Combine", Function[{*frameIndex*,*animationFrame*,*beforeFrameData*},*exp*]]

SMTAnimationOfResponse["Combine", *framesFile*,
Function[{*frameIndex*,*animationFrame*,*beforeFrameData*},*exp*]]

uses given function to produce video frames and create Animation object. Function is called for each video frame of original animation.

Output of the SMTAnimationOfResponse depends on the leading parameter of the problem. The problem can be parameterized either by time (t), boundary conditions multiplier (λ) or a general parameter (γ). Leading parameter is γ if γ is defined, t if time is defined and λ if neither t nor γ are defined. set initial point for the response curve plot

Examples:

`Simple bending of the column`

`Cyclic tension test`

`Houglassing test with animation`

`Analysis of Equilibrium Paths`

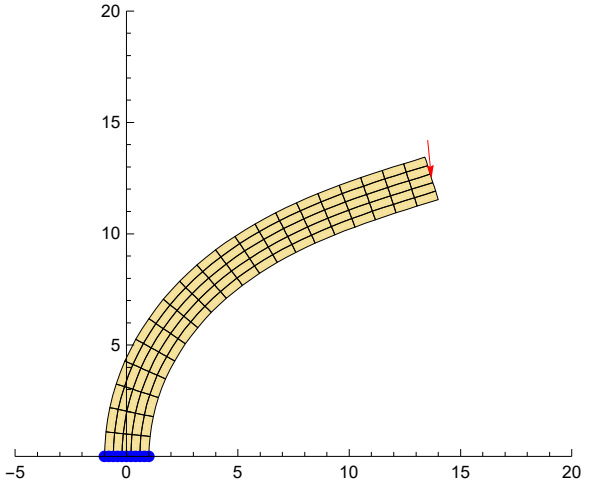
Example: Simple bending of the column

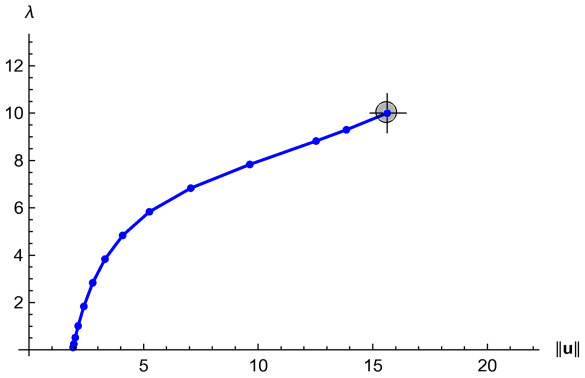
In[106]:=
```
<< AceFEM`;
SMTInputData[];
Vref = 10; H0 = 10; L = 20; B = 2;
SMTAddDomain["Ω", {"ML:", "SE", "PS", "Q2", "DF", "HY", "Q2", "D", {{"NeoHooke", "WA"}}},
  {"E *" -> 21000, "ν *" -> 0.3}];
SMTAddEssentialBoundary[Line[{{-B/2, 0}, {B/2, 0}}], 1 -> 0, 2 -> 0];
SMTAddNaturalBoundary[Point[{0, L}], 2 -> -Vref];
SMTAddInitialBoundary[Point[{0, L}], 1 -> H0];
SMTAddMesh[Polygon[{{B/2, 0}, {B/2, L}, {-B/2, L}, {-B/2, 0}}], "Ω", "Division" -> {20, 5}];
SMTAnalysis[];
```

In[10]:=
```
λMax = 10; λ0 = λMax/100; ΔλMin = λMax/1000; ΔλMax = λMax/10;
tolNR = 10^-8; maxNR = 15; targetNR = 8;
SMTNextStep["λ" → λ0];
SMTAnimationOfResponse["Initialize"];
While[
  While[
   step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λMax}]
   , SMTNewtonIteration[];
  ];
  If[Not[step[[1]]], SMTAnimationOfResponse[
    (*specify node for which response is ploted*)
    "LeadingNodePosition" → {0, L}
    (*coordinate range for response curve*)
    , "PlotOptions" → {PlotRange → {{0, 20}, {0, 12}}}
    (*coordinate range for animation*)
    , "ShowMeshOptions" → {PlotRange → {{-5, L}, {0, L}}}
    (*set image size in printer points*)
    , "ImageSize" → 300
    (*show the frames into separate window and save the frames into file responseFile.h5*)
    , "Show" → "Window" | {"ExportFrames", "responseFile"}
    (*save additional data Ry(λ) and σyy(λ) for all frames*)
    , "Data" → Hold[{{Total[SMTResidual["Y" == 0 &]][[2]], SMTRData["Multiplier"]},
       {SMTPostData["Syy", Point[{0, 0}]], SMTRData["Multiplier"]}}]
   ]];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" → step[[2]]]
 ];
```

- show the final frame again

In[45]:=
```
SMTAnimationOfResponse["Last frame"]
```





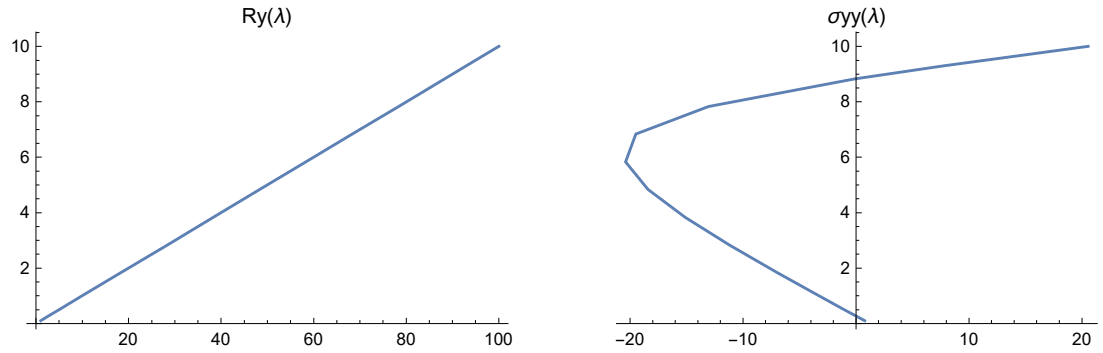
- get all points of response curve

In[44]:=
```
SMTAnimationOfResponse["Response"]
```

```
{{{1.92226, 0.1}, {1.95519, 0.24898}, {2.01799, 0.519575},
  {2.14299, 1.01107}, {2.39056, 1.83356}, {2.77964, 2.83356},
  {3.31449, 3.83356}, {4.08542, 4.83356}, {5.25316, 5.83356}, {7.06097, 6.83356},
  {9.63365, 7.83356}, {12.5265, 8.82335}, {13.8497, 9.29805}, {15.6272, 10.}}}
```

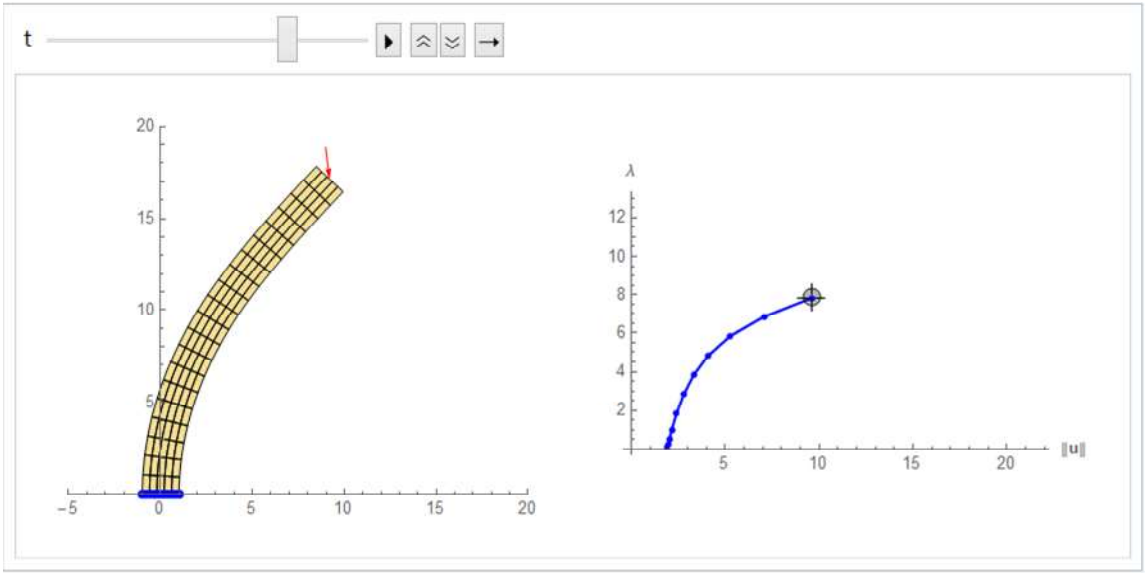
- use additional data stored for each frame ("Data" option) to produce plots Ry(λ) and σyy(λ)

In[50]:=
```
GraphicsRow[{
  ListLinePlot[SMTAnimationOfResponse["Additional data"][[All, 1]], PlotLabel → "Ry (λ) "]
 , ListLinePlot[SMTAnimationOfResponse["Additional data"][[All, 2]], PlotLabel → "σyy (λ) "]}
, ImageSize → 600]
```



- create animation object

In[51]:=
```
SMTAnimationOfResponse["Animate"]
```



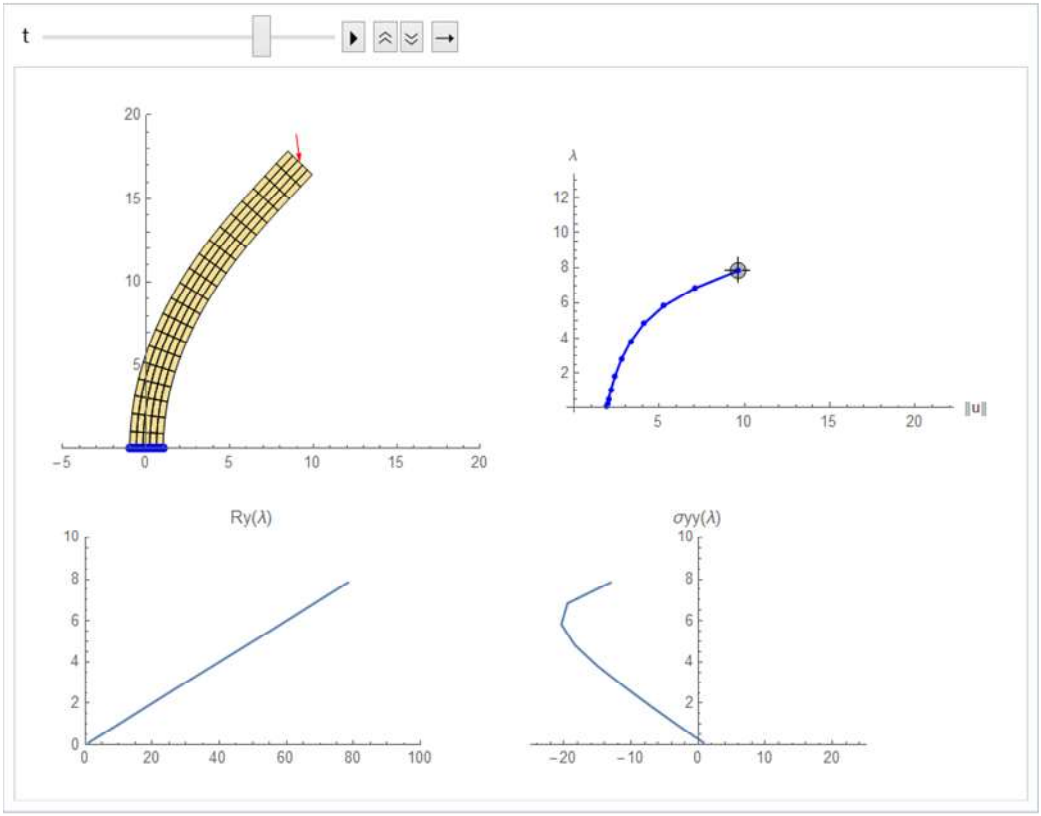
- export animation as animated GIF file

In[73]:=
```
SMTAnimationOfResponse["Export to", "gif", "FirstFrame" → "LastFrame"]
```

```
responseFile.gif
```

- combine frames and data stored into combined graphics

In[77]:=
```
combinedAnimation = SMTAnimationOfResponse["Combine",
  Function[{frameIndex, animationFrame, beforeFrameData}, Column[{
     animationFrame
     , GraphicsRow[{
       ListLinePlot[beforeFrameData[[All, 1]],
        PlotLabel → "Ry(λ)", PlotRange → {{0, 100}, {0, 10}}]
       , ListLinePlot[beforeFrameData[[All, 2]], PlotLabel → "σyy(λ)",
        PlotRange → {{-25, 25}, {0, 10}}]}
      , ImageSize → 600]
    }]]
 ]
```



- export combined graphics as animated GIF file

In[78]:=
```
Export["combinedAnimation.gif", combinedAnimation]
```

```
combinedAnimation.gif
```

Example: Combining several simulations

- Here bending of imperfect column is performed for angels of imperfection ϕ=0, 0.01 and 0.2. Animation of response is done at fixed load steps. Fixed steps are combined with adaptive time stepping inside the fixed steps. The number of exported load steps (frames) must be equal for all simulations.

In[79]:=
```
<< AceFEM`;
ϕAll = {0, 0.01, 0.2};
style = 1;
Do[
 SMTInputData[];
 Vref = 1; L = 20; B = 2;
 SMTAddDomain["Ω", {"ML:", "SE", "PS", "Q2", "DF", "HY", "Q2", "D", {{"NeoHooke", "WA"}}},
  {"E *" -> 21000, "ν *" -> 0.3}];
 SMTAddEssentialBoundary[Line[{{-B/2, 0}, {B/2, 0}}], 1 -> 0, 2 -> 0];
 SMTAddMesh[Polygon[{{-B/2, 0}, {B/2, 0}, {B/2 + ϕ L, L}, {-B/2 + ϕ L, L}}],
  "Ω", "Division" -> {5, 20}];
 SMTAddNaturalBoundary[Point[{0 + ϕ L, L}], 2 -> -Vref];
 SMTAnalysis[];
 SMTAnimationOfResponse["Initialize", {0, 0}];
 λMax = 100; nstep = 40; Δλ = λMax / nstep; tolNR = 10^-8; maxNR = 15;
 ΔλMin = λMax / 1000; ΔλMax = λMax / 10; targetNR = 8;
 realTimeFactor = 15. / λMax;
 (*perform simulation at fixed steps*)
 Do[
  SMTNextStep["λ" → λ];
  (*inside the step perform addaptive time stepping*)
  While[
   While[
    step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, Δλ, λ}]
    , SMTNewtonIteration[];
   ];
   If[step⟦4⟧ === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
   step⟦3⟧
   , If[step⟦1⟧, SMTStepBack[];];
   SMTNextStep["Δλ" → step⟦2⟧]
  ];
  SMTAnimationOfResponse[
   "LeadingNodePosition" → {B/2 + ϕ L, L}
   , "PlotOptions" → {PlotRange → {{0, L}, {0, λMax}}, PlotStyle → {Blue, Black}[[style]]}
   , "ShowMeshOptions" → {PlotRange → {{-B, L}, {0, L}}}
   , "ImageSize" → 400
   (*export frames to tmpϕ.h5 files*)
   , "Show" -> "Window" | {"ExportFrames", "tmp" <> ToString[ϕ]}
   , "Include" -> {"ShowMesh", "Plot"}
   , "RealTime" → realTimeFactor λ
   , "PlotMarkers" → {Automatic, False}[[style]]
  ];
  , {λ, Δλ, λMax, Δλ}];

 , {ϕ, ϕAll}]
```

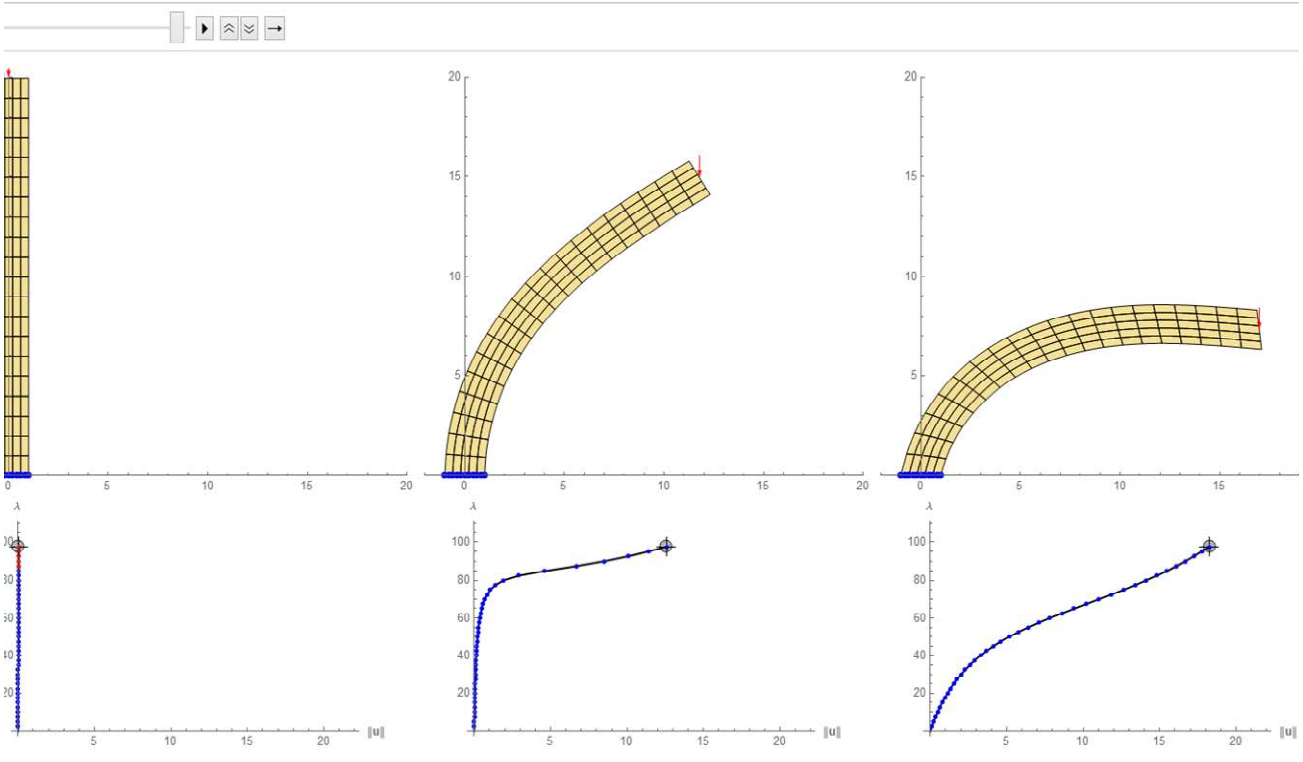
In[1]:=
```
<< AceFEM`;
ϕAll = {0, 0.01, 0.2};
```

- Read all reports from h5 files where frames were exported and create a list of grids containing frames.

In[91]:=
```
allFrames = Table[SMTAnimationOfResponse["Frame", "tmp" <> ToString[ϕ]], {ϕ, ϕAll}];
```

- Animation of load steps. Each simulation is presented separately but synchronized with all others.

In[84]:=
```
SMTAnimationOfResponse["Animate", Map[Grid[Transpose[#]] &, allFrames // Transpose]]
```



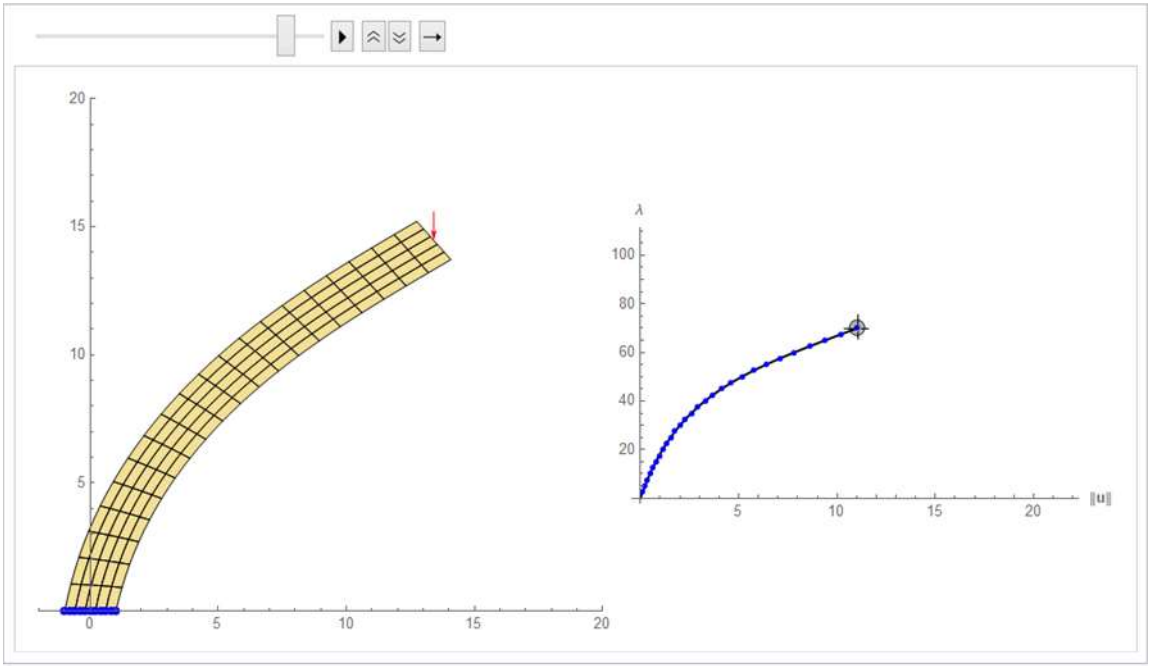
- Here an animated gif file is created. The initial frame is the final frame of the simulation.

In[85]:=
```
SMTAnimationOfResponse["Export to", "gif", Map[Grid[Transpose[#]] &, allFrames // Transpose],
 "FileName" → "tmpAnim", "FirstFrame" → "LastFrame"]
```

```
tmpAnim.gif
```

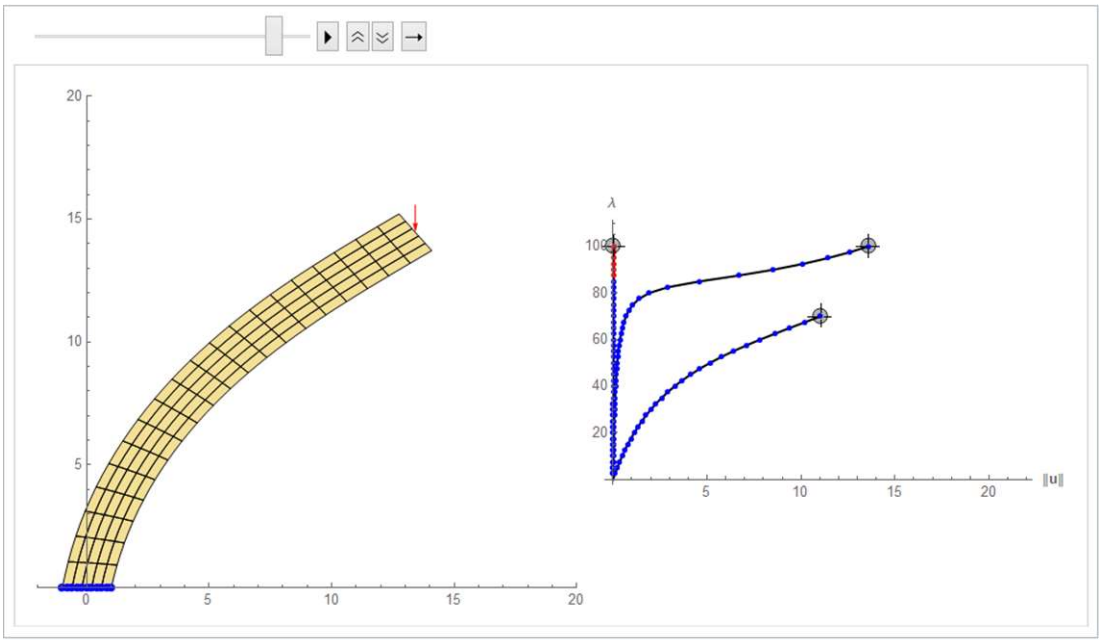
- Here a grid of graphs is created for all load step. However simulations are presented in sequence.

In[87]:=
```
ListAnimate[Map[Row, Flatten[allFrames, 1]]]
```



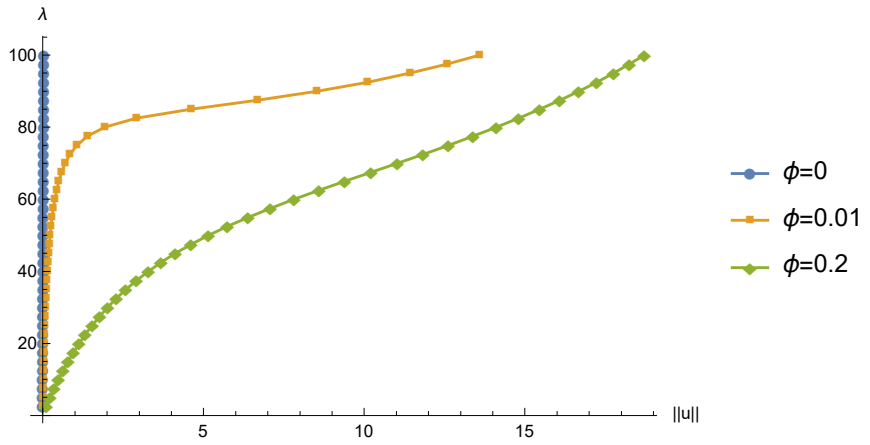
- Here a grid of graphs is created for all load step. The last frame of all previous simulations is used as background.

```
In[92]:= background = Graphics[];
         allFrames1 = Flatten[Table[
             frames = Table[
               plot = frame[[2]];
               Row[{frame[[1]], Show[plot, background]}]
               , {frame, simData}];
             (*The last frame of all previous simulations is used as background.*)
             background = {background, plot};
             frames
             , {simData, allFrames}]];

In[94]:= ListAnimate[allFrames1]
```



- Here all response curves a plotted separately.

```
In[102]:= allResponse = Table[SMTAnimationOfResponse["Response", "tmp" <> ToString[ϕ]], {ϕ, ϕAll}];
          ListLinePlot[allResponse, PlotLegends → Map[Row[{"ϕ=", #}] &, ϕAll],
           PlotMarkers → Automatic, AxesLabel → {"||u||", "λ"}]
```



SMTMakeAnimation

SMTMakeAnimation[*keyword*, *format*, *options*] ≡ SMTAnimationOfResponse["Export to", *format*, *keyword*, *options*]

Obsolete function, please use more general SMTAnimationOfResponse instead.

SMTShowEigenvectors

SMTShowEigenvectors[**K**, *n, options*]
evaluates first *n* eigenvectors of the global matrix **K** and presents them imposed on a mesh as displacements of the nodes.

option	default	description
"Scale"	1	scaling factor for the eigenvectors
"ShowMesh"	True	True - returns the graphics representation of eigenvectors False - returns the table of eigenvectors

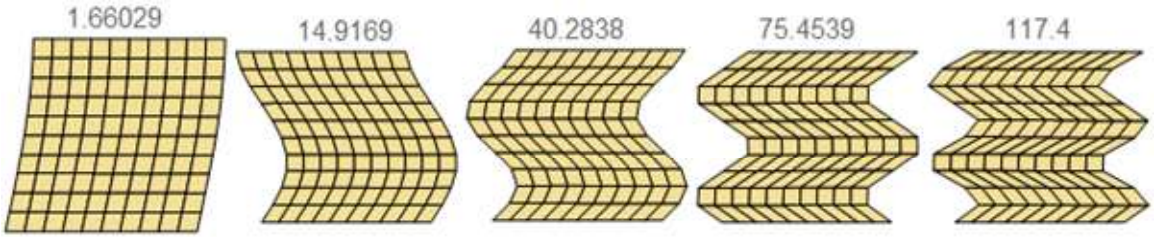
Options of the *SMTShowEigenvectors* function.

Example: Eigenvectors of block under pressure

In[115]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "PE", "Q1", "DF", "HY", "Q1", "D", {{"NeoHooke", "WA"}}},
  {"E *" -> 250, "ν *" → 0.4999}];
ne = 10; ΔH = -1; a = 1;
SMTAddMesh[Polygon[{{-a, 0}, {a, 0}, {a, 2 a}, {-a, 2 a}}], "A", "Division" -> {ne, ne}];
SMTAddEssentialBoundary[{ Point[{0, 0}], 1 -> 0, 2 → 0},
  { Line[{{-a, 0}, {a, 0}}], 2 -> 0}, { Line[{{-a, 2 a}, {a, 2 a}}], 2 -> ΔH }];
SMTAnalysis[];
```

- Display first 5 eigenvectors and eigenvalues.

In[429]:=
```
Row[SMTShowEigenvectors[SMTData["TangentMatrix"], 5, "Scale" → 3, ImageSize → 100]]
```



See also: Houglassing test with animation

Post-processing

SMTResidual

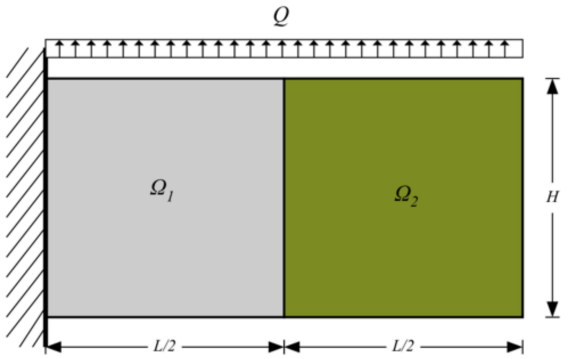
SMTResidual[*nodeSelector*]
evaluate residual (reaction) in nodes defined by *nodeSelector* (see Selecting Nodes) as a sum of residual vectors of all elements that contribute to the node. The result is a list of nodal residual vectors for all selected nodes.

SMTResidual[*nodeSelector,elementSelector*]
evaluate residual (reaction) in nodes defined by *nodeSelector* (see Selecting Nodes)

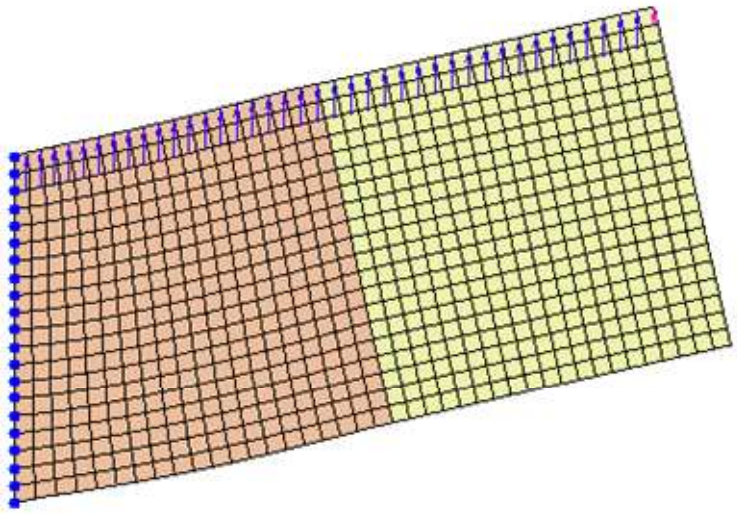
as a sum of residual vectors of all elements defined by *elementSelector* (see Selecting Elements) that contribute to the specific node

Example:

Calculate the total force at the clamped end and on the interface between the regions Ω_1 and Ω_2 of the cantilever beam depicted below.



In[520]:=
```
<< AceFEM`;
SMTInputData[];
L = 10; Q = 20; H = L / 2;
SMTAddDomain[
  {"Ω1", "ExamplesSensitivity2D", {"E *" -> 1000, "ν *" -> 0.3}},
  {"Ω2", "ExamplesSensitivity2D", {"E *" -> 5000, "ν *" -> 0.2}}];
SMTAddEssentialBoundary[Line[{{0, 0}, {0, H}}], 1 -> 0, 2 → 0];
SMTAddNaturalBoundary[ Line[{{0, H}, {L, H}}], 2 → Line[{Q}]];
SMTAddMesh[Polygon[{{0, 0}, {L/2, 0}, {L/2, H}, {0, H}}], "Ω1", "Q1", {20, 20}];
SMTAddMesh[Polygon[{{L/2, 0}, {L, 0}, {L, H}, {L/2, L/2}}], "Ω2", "Q1", {20, 20}];
SMTAnalysis[];
SMTNextStep["λ" -> 1];
While[SMTConvergence[10^-9, 10], SMTNewtonIteration[]];
SMTShowMesh["BoundaryConditions" → True, "DeformedMesh" → True]
```



First the total force at the clamped end is calculated:

In[532]:=
```
Total[SMTResidual[Line[{{0, 0}, {0, H}}]]]
```

$$\{-1.13687 \times 10^{-13}, -197.5\}$$

Note that the part of the distributed force Q that goes directly into the supported nodes is not accounted for in the calculation, thus the total vertical support force is not exactly $Q*L = 200$ as it theoretically should be. This represents a discretization error and is **NOT A**

BUG.

Second, calculate the total force on the interface between the regions Ω_1 and Ω_2. The total residual in all nodes from all elements is equal to the external load in that nodes, thus the command

In[533]:=
```
Total[SMTResidual[Line[{{L/2, 0}, {L/2, H}}]]]
```

$$\{-6.59028 \times 10^{-13},\ 5.\}$$

yields exactly what is should, the external force in the node at middle. In order to get the total force at the interface, elements of one of the region have to be excluded from the summation. Summation over the region Ω_1 yields correct result $Q*L/2 = 100$ (with again a small discretization error).

In[534]:=
```
Total[SMTResidual[Line[{{L/2, 0}, {L/2, H}}], "Ω1"]]
```

$$\{1.27898 \times 10^{-13},\ 102.5\}$$

SMTPostData

SMTPostData[*field_code*,Point[T]]
evaluate scalar *field/fields* at spatial point T

SMTPostData[*field_code*, Point[{T_1,T_2,...}]]
evaluate scalar *field/fields* at spatial points {T_1,T_2,...}

SMTPostData[*field_code*, Line[{T_1,T_2,...}, *N*_Integer]]
Spatial points {T_1,T_2,...} are first interpolated by B-spline interpolation of given degree. The scalar *field/fields* is then evaluated in *N* equidistant points positioned on given B-spline curve

SMTPostData[*field_code*]
Get a vector composed of the nodal values of scalar *field/fields*. Function returns a vector of *NoNodes* or a matrix of *NoNodes*×*NoFields* numbers that can be used for post-processing by the SMTShowMesh command or directly in Mathematica.

Post-processing of results at spatial points or nodes.

field_code	description
pcode_String	field defined by post−processing keyword *pcode* (definition of post−processing keywords is a part of element code, see `Subroutine: Postprocessing`)
{*ncode_String*, *dof_Integer*}	field is the *dof* −th component of the primal analysis related nodal quantity *ncode* ("at","ap" or "da") (see also `Node Data`)
{*scode_String*, *dof_Integer*, *sID_String*}	field is the *dof* − th component of the sensitivity analysis related nodal quantity *scode* ("st" or "sp") (see also `Node Data`) . The *sID* parameter is an identification or a keyword that defines specific design sensitivity parameter (see `Sensitivity Analysis`).
{*scode_String*, *dof_Integer*, {sID_1, sID_2}}	field is a second order sensitivity parameter defined by sensitivity parameters keywords {sID_1, sID_2}
$\{s_1, s_2, \ldots, s_{NoNodes}\}$	scalar field defined by the vector of nodal values s_i in all nodes (*NoNodes* real numbers)
{$field_1$,$field_2$,...}	set of scalar fields {$field_1$,$field_2$,...} where $field_i$ is any of the above scalar field descriptions
$\{\{s_{11}, s_{12}, \ldots, s_{1,NoFields}\}, \{s_{2,1}, s_{2,2}, \ldots, s_{2,NoFields}\}, \ldots, \{s_{NoNodes,1}, \ldots, s_{NoNodes,NoFields}\}\}$	set of scalar fields given node−wise (*NoNodes*×*NoFields* real numbers)

Various definitions of scalar field for post-processing.

option	description
"OutputForm"−>"Field"	function returns scalar *field*/*fields* evaluated at selected point or points (ordered point−wise) {{$field_1|_{T_1}$,$field_2|_{T_1}$,...}, {$field_1|_{T_2}$,$field_2|_{T_2}$,...} ,...}
"OutputForm"−>"Points"	function returns scalar *field*/*fields* evaluated at selected points together with the points {{{$field_1|_{T_1}$,$field_2|_{T_1}$,...},T_1}, {{$field_1|_{T_2}$,$field_2|_{T_2}$,...},T_2} ,...}
"OutputForm"−>"ColorLine"	function returns Line graphics object together with the legend where the value of given scalar *field* is used to get vertex colors of the given line (it can only be used in combination with Line[{T_1,T_2,...}, *N*_Integer] specification)
"OutputForm"−>"ColorPoints"	function returns Point graphics object together with the legend where the value of given scalar *field* is used to get vertex colors of a set of points lying on the given line (it can only be used in combination with Line[{T_1,T_2,...}, *N*_Integer] specification)

Option "OutputForm" defines the form of the output.

option	default	description
"Elements"→ *domains*	All	the post-processing is called only for elements at specified domains
"Tolerance"→_Real	10^{-10}	the numbers smaller in absolute magnitude than "Tolerance" are replaced by 0 within the search procedures
"SplineDegree"→ _Integer	1	degree of polynomials for B-spline interpolation
"Post"	None	function applied on the scalar *field*/*fields* after the evaluation of the *field*/*fields*
"ColorFunction"	ColorData["TemperatureMap"]	specifies a function to apply to determine vertex colors
"Style"	{}	graphics style specification for points and lines
"SaveHistory"	True	save data related to the position of the points within the mesh

Additional options for the SMTPostData command.

See also: Post-processing (3D heat conduction)

The given point T can lie outside the mesh. In that case, the value is obtained by extrapolation of the field from the element that lies closes to the given point.

The post-processing keywords (*pcode*) can only be used if the element user subroutine for data post-processing has been defined (see `Standard User Subroutines`). The post-processing code *pcode* can corresponds either to the integration point or to the nodal point post-processing quantity.

If the *pcode* code corresponds to the integration point quantity then the integration point values are first mapped to nodes accordingly to the type of extrapolation as follows:

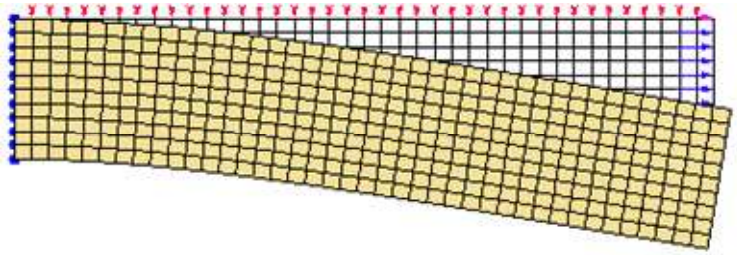
- Type 0: Least square extrapolation from integration points to nodal points is used. The nodal value is additionally multiplied by the user defined nodal weight factor that is stored in element specification data structure for each node (es$$["PostNodeWeights",nodenumber]. Default value of the nodal weight factor is 1 for nodes that appear in the list of segments and 0 for others.
- Type 1: The integration point value is multiplied by the shape function value $f_{Ni} = w_{Ni} \sum_{j=1}^{NoIntPoints} w_{ij} f_{Gj}$ i=1,2,…,NoNodes where f_{Gj} are an integration point values, w_{ij} are an integration point weight factors, w_{Ni} are the nodal wight factors andf_{Ni} are the values mapped to nodes. Integration point weight factor can be e.g the value of the shape functions at the integration point and have to be supplied by the user. By default the last NoNodes integration point quantities are taken for the weight factors.

The type of extrapolation is defined by the value of SMTIData["ExtrapolationType"] (`Integer Type Environment Data`) . By default the least square extrapolation is used. The smoothing over the patch of elements is then performed in order to get nodal values.

There are three post processing codes with the predetermined function: "DeformedMeshX", "DeformedMeshY" and "DeformedMeshZ". If the "DeformedMeshX", "DeformedMeshY" and "DeformedMeshZ" post-processing codes are specified by the user then `SMTShowMesh` use the corresponding data to produce deformed version of the original finite element mesh.

Example:

In[122]:=
```
<< AceFEM`;
SMTInputData[];
L = 1000; H = 200; q = 2;
SMTAddDomain[{"Ω", {"ML:", "SE", "PS", "Q1", "DF", "LE", "Q1", "D", "Hooke"}, {"E*" → 3000}}];
SMTAddMesh[Polygon[{{0, 0}, {L, 0}, {L, H}, {0, H}}], "Ω", "Division" -> {40, 10}];
SMTAddEssentialBoundary[ Line[{{0, 0}, {0, H}}], 1 -> 0, 2 → 0];
SMTAddNaturalBoundary[ Line[{{0, H}, {L, H}}], 2 → Line[{-q}]];
SMTAddNaturalBoundary[ Line[{{L, 0}, {L, H}}], 1 → Line[{10 q}]];
SMTAnalysis[];
SMTNextStep["λ" -> 1];
SMTNewtonIteration[];
Show[SMTShowMesh["BoundaryConditions" → True, "FillElements" → False],
 SMTShowMesh["DeformedMesh" → True]]
```



- displacement "u" in point (L,H)

In[547]:=
```
SMTPostData["u", Point[{L, H}]]
```

Out[547]=
```
23.3448
```

- displacement vector in point (L,H)

In[548]:=
```
SMTPostData[{"u", "v"}, Point[{L, H}]]
```

Out[548]=
```
{23.3448, -129.249}
```

- norm of displacement vector in point (L,H)

In[549]:=
```
SMTPostData[{"u", "v"}, Point[{L, H}], "Post" → Norm]
```

Out[549]=
```
131.34
```

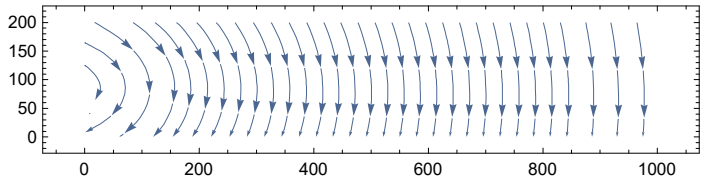
- displacement vector in points (L,0) and (L,H)

In[550]:=
```
SMTPostData[{"u", "v"}, Point[{{L, 0}, {L, H}}]]
```

Out[550]=
```
{{-9.85315, -128.782}, {23.3448, -129.249}}
```

- stream plot of displacements

In[551]:=
```
displacement[x_?NumberQ, y_?NumberQ] := SMTPostData[{"u", "v"}, Point[{x, y}]]
StreamPlot[displacement[x, y], {x, 0, L}, {y, 0, H}, AspectRatio → Automatic]
```



- norm of displacement vector in points (L,0) and (L,H)

In[553]:=
```
SMTPostData[{"u", "v"}, Point[{{L, 0}, {L, H}}], "Post" → Norm]
```

```
{129.159, 131.34}
```

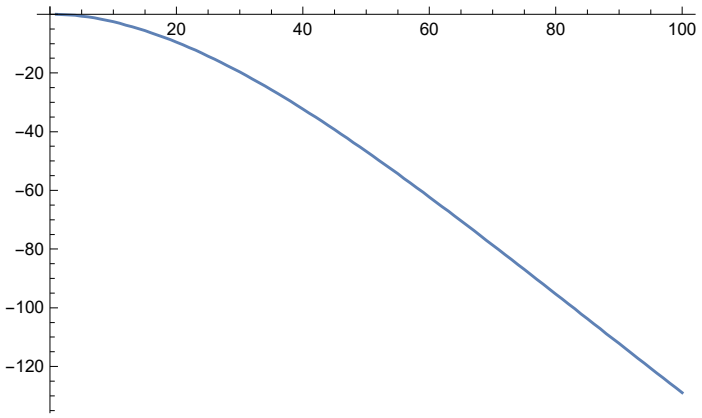
- displacement vector in points (L,0) and (L,H) together with the points

In[554]:=
```
SMTPostData[{"u", "v"}, Point[{{L, 0}, {L, H}}], "OutputForm" → "Points"]
```

```
{{{-9.85315, -128.782}, {1000, 0}}, {{23.3448, -129.249}, {1000, 200}}}
```

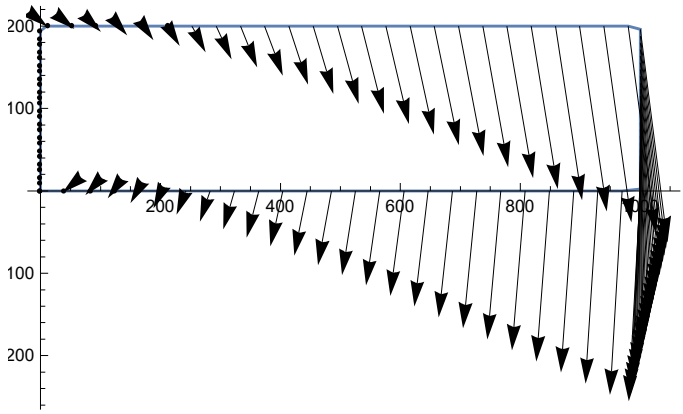
- displacement "v" along the central line (0,H/2) - (L,H/2) in 100 points.

In[555]:=
```
ListLinePlot[SMTPostData["v", Line[{{0, H/2}, {L, H/2}}, 100]]]
```



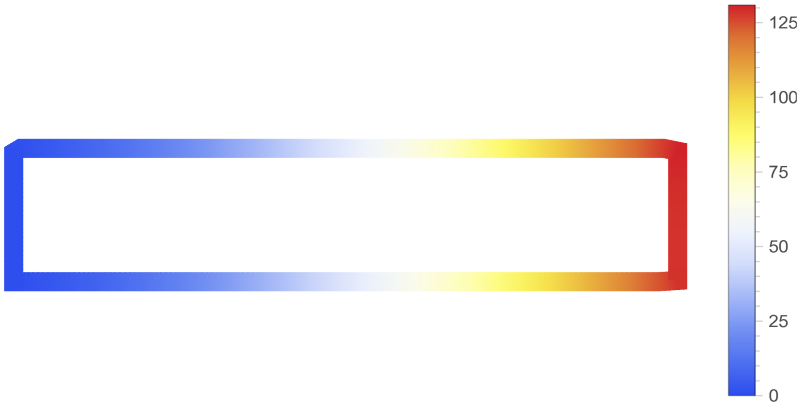
- plot displacement vectors at the outline

In[556]:=
```
data = SMTPostData[{"u", "v"},
    Line[{{0, 0}, {L, 0}, {L, H}, {0, H}, {0, 0}}, 100], "OutputForm" -> "Points"];
Show[ListLinePlot[data[[All, 2]]],
 Graphics[Map[Arrow[{#[[2]], #[[2]] + 2 #[[1]]}] &, data]], PlotRange → All, ImagePadding → 10]
```

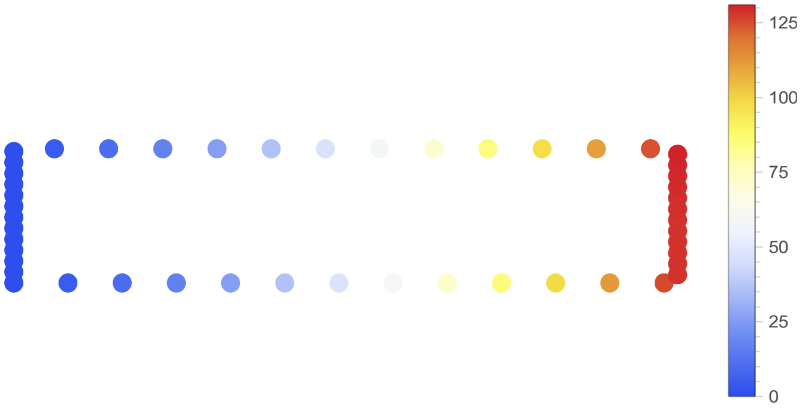


- plot norm of displacements at the outline as vertex color

In[558]:=
```
Apply[Legended, SMTPostData[{"u", "v"}, Line[{{0, 0}, {L, 0}, {L, H}, {0, H}, {0, 0}}, 100],
  "Post" → Norm, "OutputForm" -> "ColorLine", "Style" → AbsoluteThickness[10]]]
```



In[559]:=
```
Apply[Legended, SMTPostData[{"u", "v"}, Line[{{0, 0}, {L, 0}, {L, H}, {0, H}, {0, 0}}, 50],
  "Post" → Norm, "OutputForm" -> "ColorPoints", "Style" → AbsolutePointSize[10]]]
```



SMTData

SMTData[*GCode_String*]
get data accordingly to the keyword *gcode*

SMTData[*ie, ECode_String*]
get element data for the element with the index *ie* accordingly to the keyword *ecode*

SMTData[..., "File"->True]
append the result of the *SMTData* function to the output file

The SMTData function returns data according to the code *GCode* or *ECode* and element number *ie*.The data returned can be used for post-processing and debugging. An advanced user can use the Data Base Manipulations to access and change all the analysis data base structures during the analysis.

GCode	description
"Time"	real time (see `Iterative Solution Procedures`)
"Multiplier"	natural and essential boundary conditions multiplier (load level) (see `Iterative Solution Procedures`)
"Step"	total number of completed solution steps (see `Iterative Solution Procedures`)
"Parameter"	general parameter used to parameterize the problem (see `Iterative Solution Procedures`)
"NonPositivePivots"	number of negative or near zero pivots (or −1 if data is not available), (see also `SMTSetSolver`)
"MatrixGlobal"	global matrix according to the last completed action (command yields a SparseArray)
"VectorGlobal"	global vector according to the last completed action
"MatrixLocal"	contents of the local matrix received from the user subroutine in the last completed action (e.g. last evaluated element tangent matrix)
"VectorLocal"	contents of the local vector received from the user subroutine in the last completed action (e.g. last evaluated element residual vector)
"TangentMatrix"	global tangent matrix is obtained by executing one Newton−Raphson iteration (SMTNewtonIteration) without solving the linear system of equations), (command yields a SparseArray of dimensions NoEquations×NoEquations)
"Residual"	global residual vector of dimension NoEquations (includes contribution from the elements and the natural boundary conditions)
"DOFNode"	for all unknown parameters the index of the node where the parameter appears
"DOFNodeID"	for all unknown parameters the identification of the node where the parameter appears
"DOFElements"	for all unknown parameters the list of elements where the parameter appears

Various global data directly accessible from *Mathematica*.

ECode	*Return values*	Action
"All"	String	collect all the data associated with the element
"SKR"	tangent matrix and residual for element *ie*	call to the "SKR" user subroutine (see `SMSStandardModule`)
"CKR"	statically condensed (Schur complement) of tangent matrix and residual for element *ie*	call to the "SKR" user subroutine (see `SMSStandardModule`), after that perform `Elimination of Local Unknowns` and return statically condensed (Schur complement) of tangent matrix and residual
"SSE"	first order sensitivity pseudo-load vector	call to the "SSE" user subroutine for selected element and *i*-th sensitivity parameter defined by SMTIData["SensIndex",*i*] (see `Introduction to Sensitivity Analysis`)
"SS2"	second order sensitivity pseudo-load vector	call to the "SS2" user subroutine for selected element and *i*-th sensitivity parameter defined by SMTIData["SensIndex",*i*] (see `Introduction to Sensitivity Analysis`)
"SHI"	True	call to the "SHI" user subroutine or selected element and *i*-th sensitivity parameter defined by SMTIData["SensIndex",*i*]
"SRE"	residual for element *ie*	call to the "SKR" user subroutine
"SPP"	get arrays of integration point and nodal point post-processing quantities for element *ie*	call to the "SPP" user subroutine
"User*n*"	True	call to the user defined "User*n*" user subroutine where n=1,2,…,9

Various element data directly accessible from *Mathematica*.

Perform Visualization Phase Separate from Analysis Phase

Here a method how to separate the analysis and visualization of the results into two completely separated *Mathematica* sessions is demonstrated. However, one has to be aware that when the analysis and visualization are done separately, one can visualize only the data that have been stored during the analysis. The data stored during analysis includes original mesh and vectors of nodal values for all post-processing quantities defined in post-processing subroutines (see `Standard User Subroutines`) for all elements.

The data stored does not include the nodal DOF and history data or any modification of the mesh done during the analysis, thus complex post-processing has to be done in parallel with the analysis as presented in Bending of the column (path following procedure, animations, 2D solids) example or with the use of `SMTDump`/`SMTRestart`.

SMTPut

SMTPut[*pkey*,*uservector*,*options*]
save all visualization data together with the additional vector of arbitrary real numbers *uservector* to the file specified by the `SMTAnalysis` option "DumpInputTo" using the keyword *pkey*

SMTPut[pkey] ≡ SMTPut[*pkey*,{}]

SMTPut[] ≡ SMTPut[SMTIData["Step"]]

option	default	description
"TimeFrequency"	0	the data is not saved if the absolute difference in time for two successive SMTPut calls is less than "TimeFrequency"
"MultiplierFrequency"	0	the data is not saved if the absolute difference in multiplier for two successive SMTPut calls is less than "MultiplierFrequency"
"StepFrequency"	0	the data is not saved if the absolute difference in step number for two successive SMTPut calls is less than "StepFrequency"

Options of the SMTPut function.

The keyword *pkey* can be arbitrary integer or real number and it has to be unique for each SMTPut call. It is used later by the `SMTGet` command to locate the position of the visualization data. The SMTPut command creates two files. The .idx file is text file that contains all the input data, all keys and also eventual definitions of symbols stored by the `SMTSave` command. The .hdf file is a binary file that stores actual visualization data. Visualization data stored is composed from all scalar field defined by post-processing subroutines of all elements.

SMTGet

SMTGet[ ]
returns all the keywords and the names of the stored post-processing quantities.

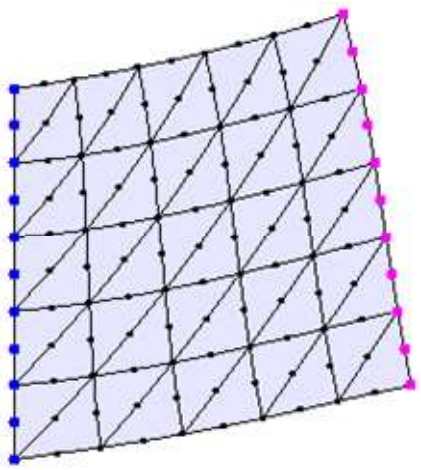
SMTGet[*pkey*]
get all post-processing data stored under the keyword *pkey*

SMTSave

SMTSave[*symbol*]
appends definitions associated with the specified symbol to a file specified by the SMTAnalysis's option *PostOutput*

Example: separate visualization

- Here is an example of the analysis of a rectangular structure (L*L). Structure is clamped at X=0 and it has prescribed vertical displacement at X=L. Session can be divided into analysis and visualization session.



Analysis session

In[140]:=
```
<< AceFEM`;
SMTInputData[];
Emodul = 1000; L = 5;
SMTAddDomain["Ω", {"ML:", "SE", "PE", "T2", "DF", "HY", "T2", "D", {{"NeoHooke", "WA"}}},
  {"E *" -> Emodul, "ν *" -> 0.3}];
SMTAddEssentialBoundary[{ "X" == 0 &, 1 -> 0, 2 -> 0}, { "X" == L &, 2 -> 1}];
SMTAddMesh[Polygon[{{0, 0}, {5, 0}, {5, 5}, {0, 5}}], "Ω", "Division" -> {5, 5}];
```

- Here the input data structures are saved into restart files (see SMTAnalysis).

In[566]:=
```
SMTAnalysis["DumpInputTo" -> "post"];
```

- Here are saved some general parameters of the problem .

In[567]:=
```
SMTSave[Emodul, L];
```

- During the analysis all the visualization data is stored for each converged step. Additionally, the total force is also stored into binary file post.hdf. The current step number (SMTIData["Step"]) is used as the keyword under which the data is stored.

In[568]:=
```
SMTNextStep["λ" → 0.1];
While[
  While[
   step = SMTConvergence[10^-8, 15, {"Adaptive BC", 8, .0001, 0.1, 1}], SMTNewtonIteration[];];
  If[Not[step[[1]]]
   , force = Plus @@ SMTResidual["X" == L &];
   SMTPut[SMTData["Step"], force];
   SMTShowMesh["DeformedMesh" → True, "Field" → "Sxx", "Contour" → 10, "Show" → "Window"];
  ];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" -> step[[2]]]
 ];
```

Visualization session

- Here the new, independent session starts by reading input data from the previously stored visualization data base files.

In[570]:=
```
<< AceFEM`;
SMTRestart["post"];
```

```
Visualisation session: post See also: SMTPut
```

- All the symbols stored by the SMTSave command are available at this point.

In[572]:=
```
Emodul
```

```
1000
```

- The SMTGet[] command returns all the keywords and the names of the stored post-processing quantities.

In[573]:=
```
{allkeywords, allpostnames} = SMTGet[];
allpostnames
allkeywords
```

```
{DeformedMeshX, DeformedMeshY, Exx, Exy, Eyx, Eyy, Ezz, Integration weight,
 Mises stress, Specific strain energy, Sxx, Sxy, Syx, Syy, Szz, u, v}
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

- The SMTGet[3] command reads the data stored under keyword 3 (step 3 for this example) back to AceFEM and returns the user defined vector of real numbers for step 3 (total force for this example).

In[576]:=
```
force = SMTGet[3]
```

$$\{3.31494 \times 10^{-13},\ 46.2523\}$$

- Here the post-processing quantity "Sxx" for the current step is evaluated at the center of the rectangle

In[577]:=
```
SMTPostData["Sxx", Point[{L / 2, L / 2}]]
```

```
0.306591
```

Example: cyclic tension test

See Cyclic tension test, advanced post-processing, animations.

Utility Post-processing Functions

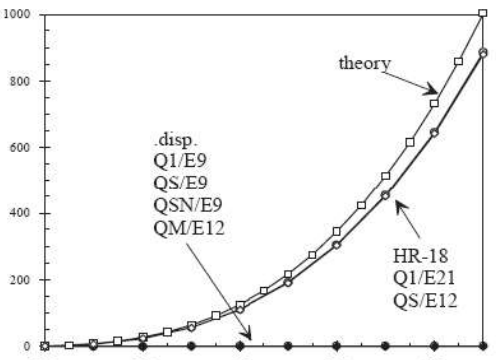
SMTScanedDiagramToTable

SMTScanedDiagramToTable[*scaned_points*,s_0,s_1,v_0,v_1]
transforms a table of points picked with *Mathematica* from the bitmap picture into the table of points in the actual coordinate system, where s_0 and s_1 are two points picked from the picture with the known coordinates v_0 and v_1

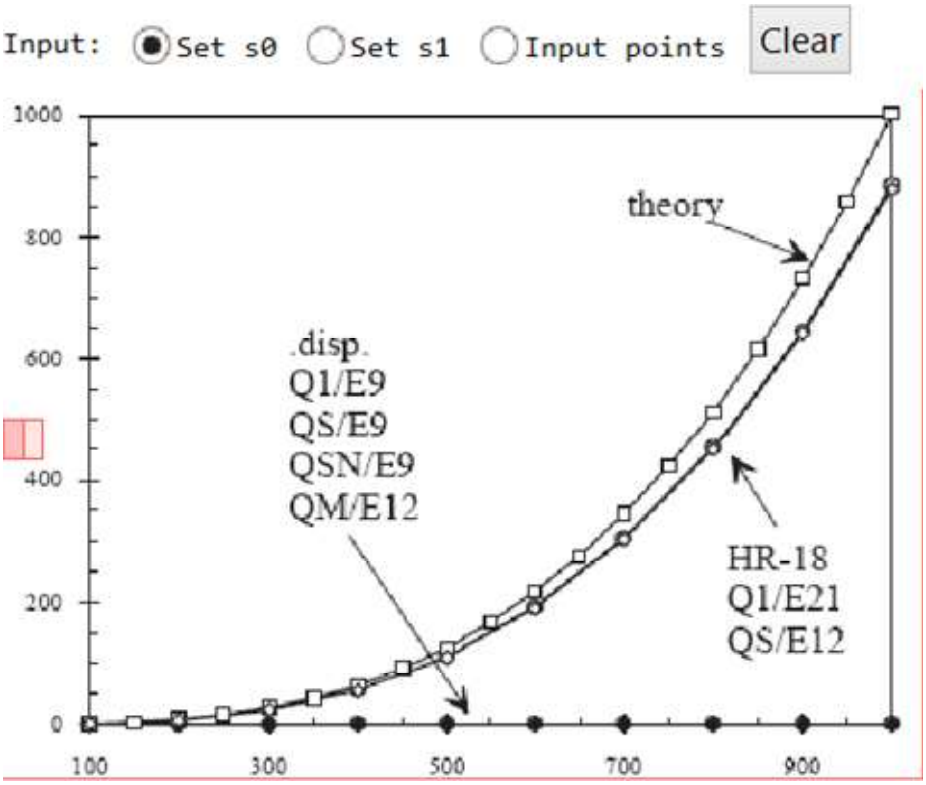
Example

In[583]:=
```
<< AceFEM`;
```

In[584]:=
```
pt = {{20, 20}, {100, 100}}; task = "s0";
Column[{
  Row[{"Input:", RadioButtonBar[Dynamic[task], {"Set s0", "Set s1", "Input points"}],
    Button["Clear", pt = Take[pt, 2]]}, "    "],
  ClickPane[
```

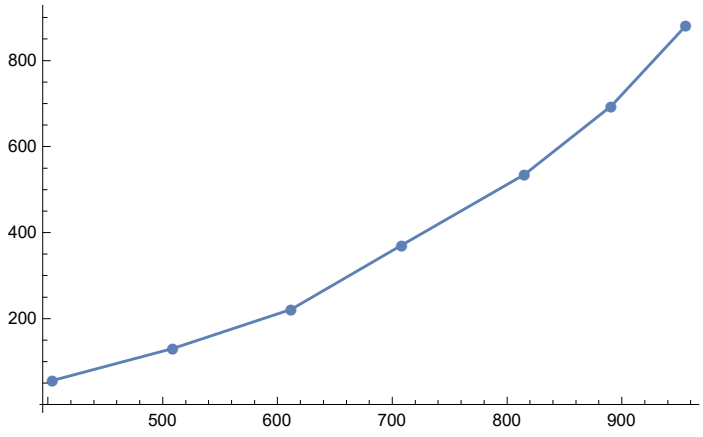



```
  Show[
   Graphics[
    Dynamic[{Red, Point[Drop[pt, 2]], Blue, Locator[pt[[1]]], Green, Locator[pt[[2]]]}]]
  , Switch[task, "Set s0", pt = {#, # + 100}, "Set s1", pt[[2]] = #,
    "Input points", AppendTo[pt, #]] &]
 }]
```



In[587]:=
```
graph = SMTScanedDiagramToTable[Drop[pt, 2], pt[[1]], pt[[2]], {100., 0}, {1000., 1000.}]
ListLinePlot[graph, PlotMarkers → Automatic]
```

```
{{404.196, 56.0748}, {509.091, 130.841}, {611.888, 221.184},
 {708.392, 370.717}, {815.385, 535.826}, {890.909, 694.704}, {955.944, 881.62}}
```



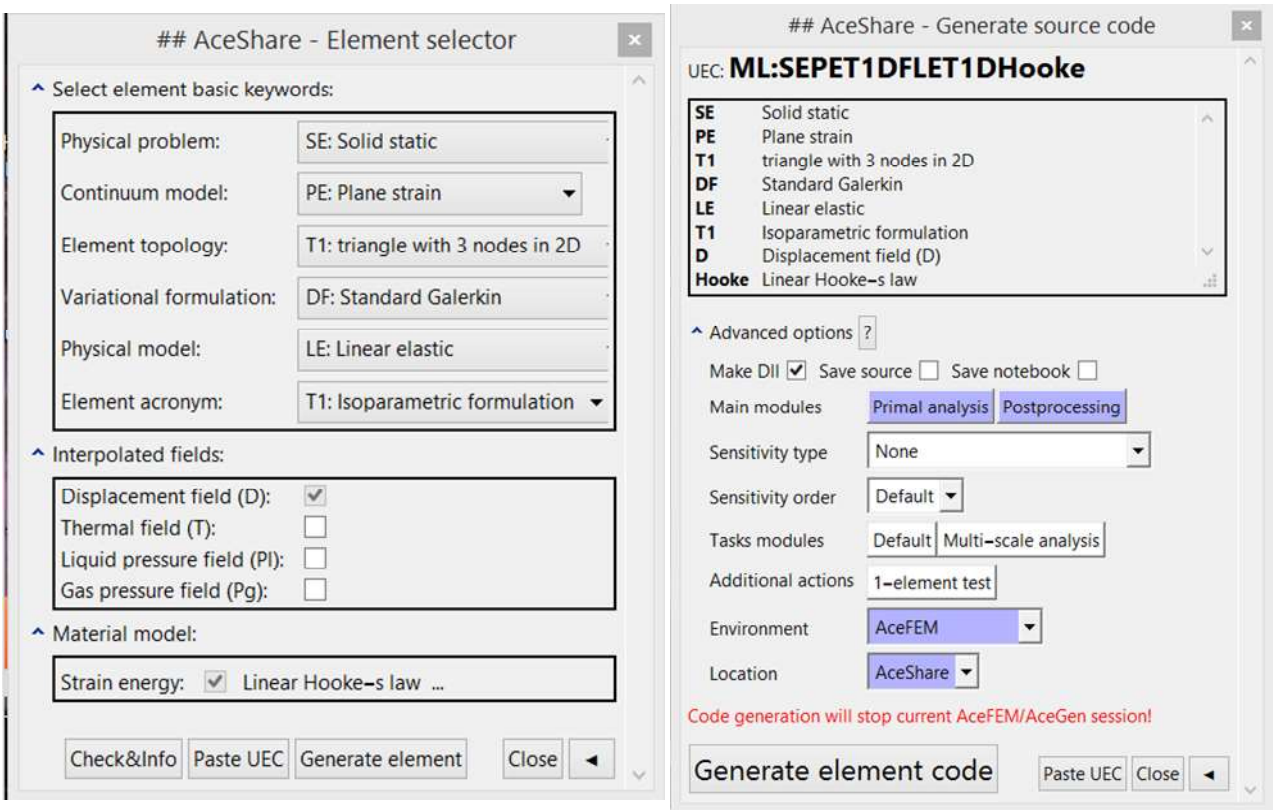
AceShare

Contents

Introduction

The AceFEM environment does not contain actual finite elements. Within the AceFEM environment the actual finite element definitions and/or source files are accessed through the *AceShare* file sharing system. The *AceShare* system is a file sharing mechanism built in AceFEM that allows:

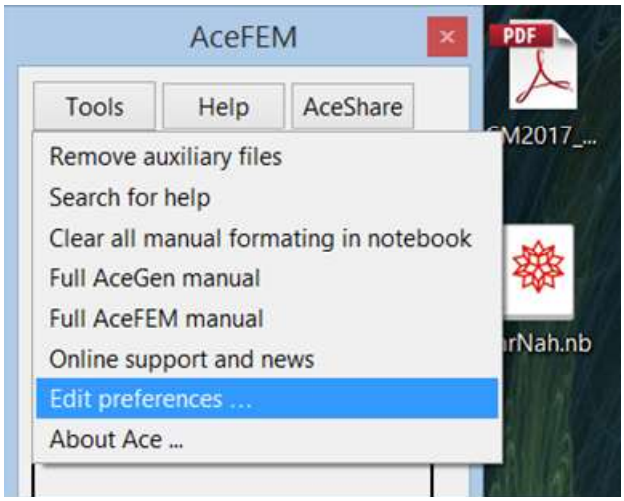
- browsing the on-line FEM libraries
- download of the precomputed finite elements from the on-line libraries
- run-time generation of an arbitrary elements using definitions stored in on-line libraries
- creation of the user defined library that can be posted on the Internet to be used by other users of the AceFEM system



AceShare palettes for selecting elements and run-time code generation.

Once the element is downloaded from *AceShare* it is stored into the local *AceShare* directory and used until the on-line version has been changed. The information about the available on-line libraries is automatically updated once per day and stored in *AceShare* directory. The downloaded dll files from the on-line libraries are also stored in AceShare directory.

The *AceShare* directory is automatically created as a subdirectory of the current notebook directory (working_notebook_directory/AceShare/). A global *AceShare* directory can be set in 'Edit preferences' menu as follows



Set up of the directory where the downloaded files are stored.

Some on-line libraries can contain an arbitrary number of pre-generated finite elements. For each pre-generated element the on-line library stores the following data:

- element dll file,
- home page file (HTML file with basic descriptions, links, etc..)
- symbolic *AceGen* input used to generate element,
- element C source code for AceFEM environment,
- optionally description and results of standard benchmark tests.

From the practical reasons, the number of pre-generated elements is limited to the most commonly used elements and to user subroutines for primal analysis and post-processing. Additionally to the pre-generated elements, the on-line contains also mechanism for the run-time selection and generation of finite element source codes that are then compiled and dynamically linked into *AceFEM* at the users home computer.

Unified Element Code

The `SMTAnalysis` command automatically locates the finite element codes in the AceShare directory or downloads the codes from the on-line libraries or generates the element source code at run-time. The elements in the libraries are identified by the unique codes given as an argument of the `SMTAddDomain` command. The code of the element is a list of strings of the form:

{"*libraryCode*:","*keyword1*","*keyword2*",…}

The chosen library is defined by the unique keyword *libraryCode* (e.g. "ML" stands for main AceFEM on-line library).

The concatenated keywords form and *elementCode*. If the specific element was precomputed and the generated source codes stored on-line then it can be identified also by concatenated UEC leading to "*libraryCode:elementCode*".

Run-time generation of the chosen element requires full UEC!

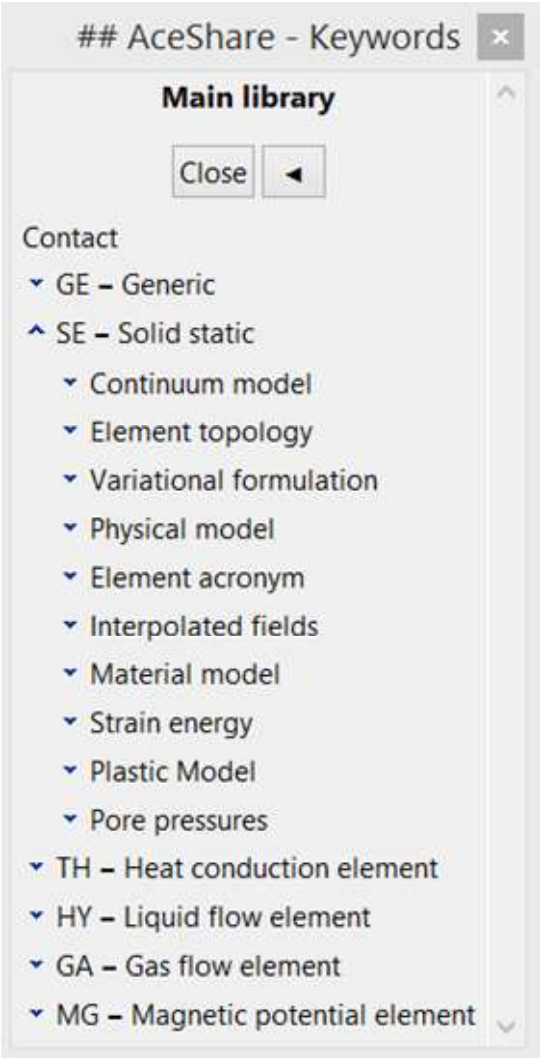
The *keywords* are in general arbitrary and are defined by the creator of the specific library (see `Create Your Own AceShare Library`).

For example, the codes of elements from the main on-line AceFEM library (library code is "ML") consist of the following keywords:

Description	*No. of characters in keyword*	*Examples of keywords*
Physical problem	2	SE ≡ Mechanical problems
Continuum model	2	PE ≡ 2 D solid plane strain problems
Topology	2	Q1 ≡ 2 D Quadrilateral with 4 nodes
Variational formulation	2	DF ≡ Full integrated displacement based elements
Physical model	2	LE == linear elastic solid
Acronym	arbitrary	Q1E4
…	arbitrary	…
…	arbitrary	…

Keywords for the standard on-line AceFEM library.

For example the {"ML:","SE","PE","Q1","HR","LE","PianSum","D","Hooke"} UEC represents the well-known Pian-Sumihara element derived for isotropic Hook's material and stored in a main on-line AceFEM library "ML". The UEC can also be given concatenated leading to "ML:SEPEQ1HRLEPianSumDHooke".



UEC browser is a part of each AceShare library.

The main AceFEM On-line library contains large number of finite elements (solid, thermal,... 2D, 3D,...) with AceGen input of most of the elements. The number of finite elements included in on-line libraries is growing daily. Please browse the available libraries to see if the finite element model for your specific problem is already available or order creation of specialized elements (www.fgg.uni-lj.si/consulting/).

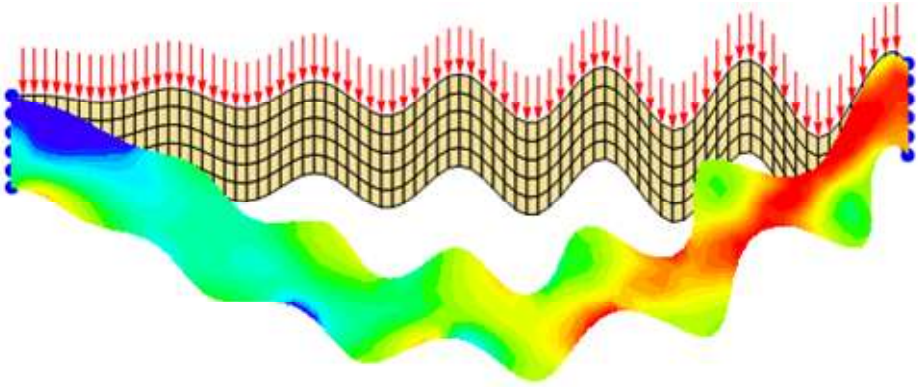
Example: Accessing elements from shared libraries

In[589]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "PS", "Q1", "ES", "HY", "Q1E4", "D", {{"NeoHooke", "WA"}}},
  {"E *" -> 1000., "ν *" -> .49, "t *" -> 1.}];
SMTAddNaturalBoundary[Abs["X" Sin["X" // N] / 20 + 8 - "Y"] < 0.1 &, 0, -.01];
SMTAddEssentialBoundary["X" == 1 &, 1 -> 0, 2 -> 0];
SMTAddEssentialBoundary["X" == 40 &, 1 -> 0, 2 -> 0];
SMTAddMesh[Raster[Array[ {#2, #2 Sin[#2 // N] / 20 + 4 #1 } &, {2, 40}]], "A", "Q1", {80, 5}];
SMTAnalysis[];
```

In[597]:=
```
SMTNextStep["λ" → 100];
While[
 While[step = SMTConvergence[10^-7, 15, {"Adaptive BC", 8, .01, 300, 500}] ,
  SMTNewtonIteration[]];
 SMTStatusReport[];
 If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
 If[Not[step[[1]]], SMTShowMesh["DeformedMesh" → True,
   "Field" → "Sxy", "Mesh" → False, "Contour" → 20, "Show" -> "Window"]];
 step[[3]] ,
 If[step[[1]], SMTStepBack[]];
 SMTNextStep["Δλ" → step[[2]]]
]
```

```
Step/Iter=1/11 λ/Δλ=100./100. ||Δp||/||R||=6.02552×10^-9/9.13936×10^-13 Events=0 Status=0/{}
Step/Iter=2/7 λ/Δλ=195.918/95.9184
  ||Δp||/||R||=6.3223×10^-13/1.02724×10^-12 Events=0 Status=0/{}
Step/Iter=3/6 λ/Δλ=338.817/142.899
  ||Δp||/||R||=4.48935×10^-8/9.83968×10^-13 Events=0 Status=0/{}
Step/Iter=4/6 λ/Δλ=500./161.183 ||Δp||/||R||=3.36622×10^-10/9.5251×10^-13 Events=0 Status=0/{}
```

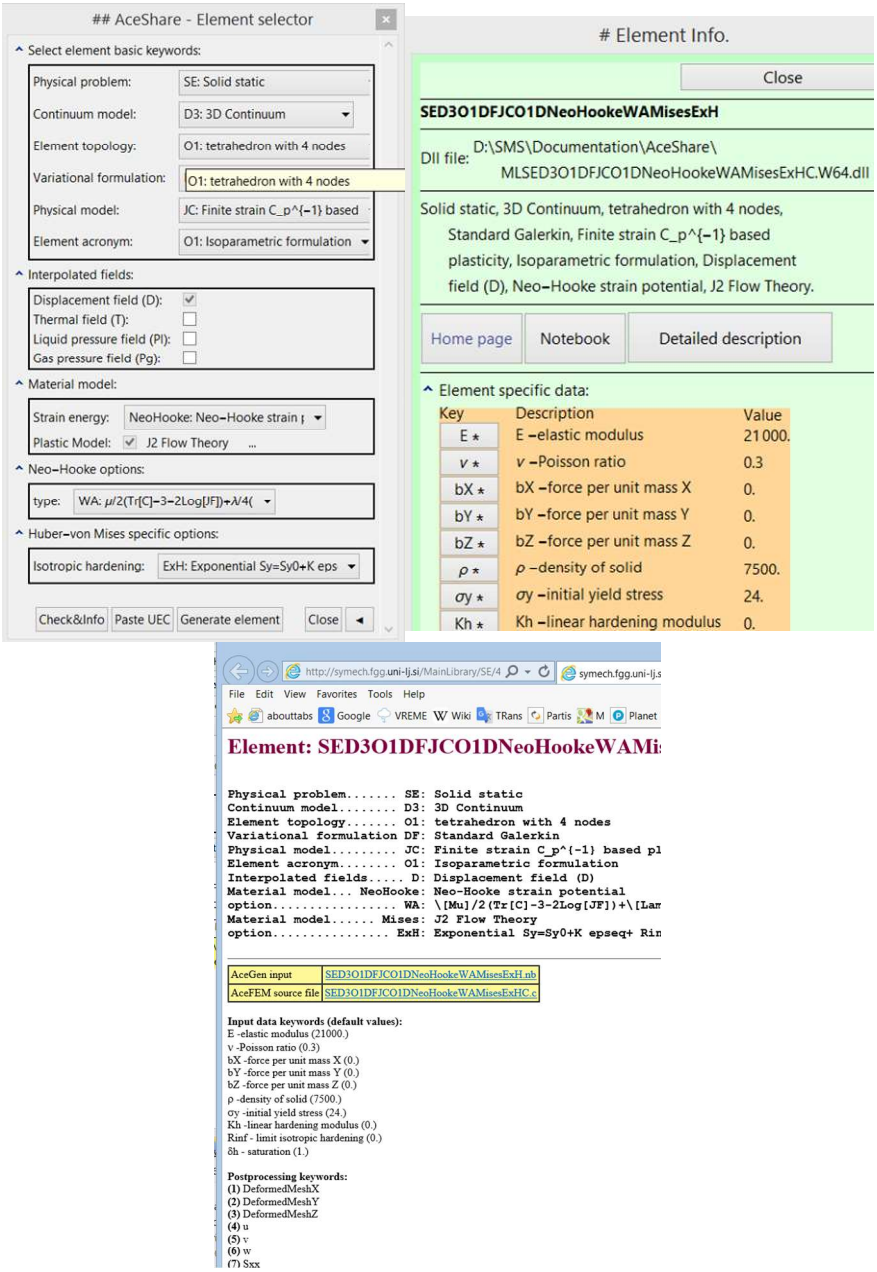
In[599]:=
```
Show[SMTShowMesh["Show" → False, "BoundaryConditions" → True],
 SMTShowMesh["DeformedMesh" → True, "Mesh" → False, "Field" → "Sxy",
  "Contour" → 20, "Show" → False, "Legend" → False], PlotRange → All]
```



Element selector menu

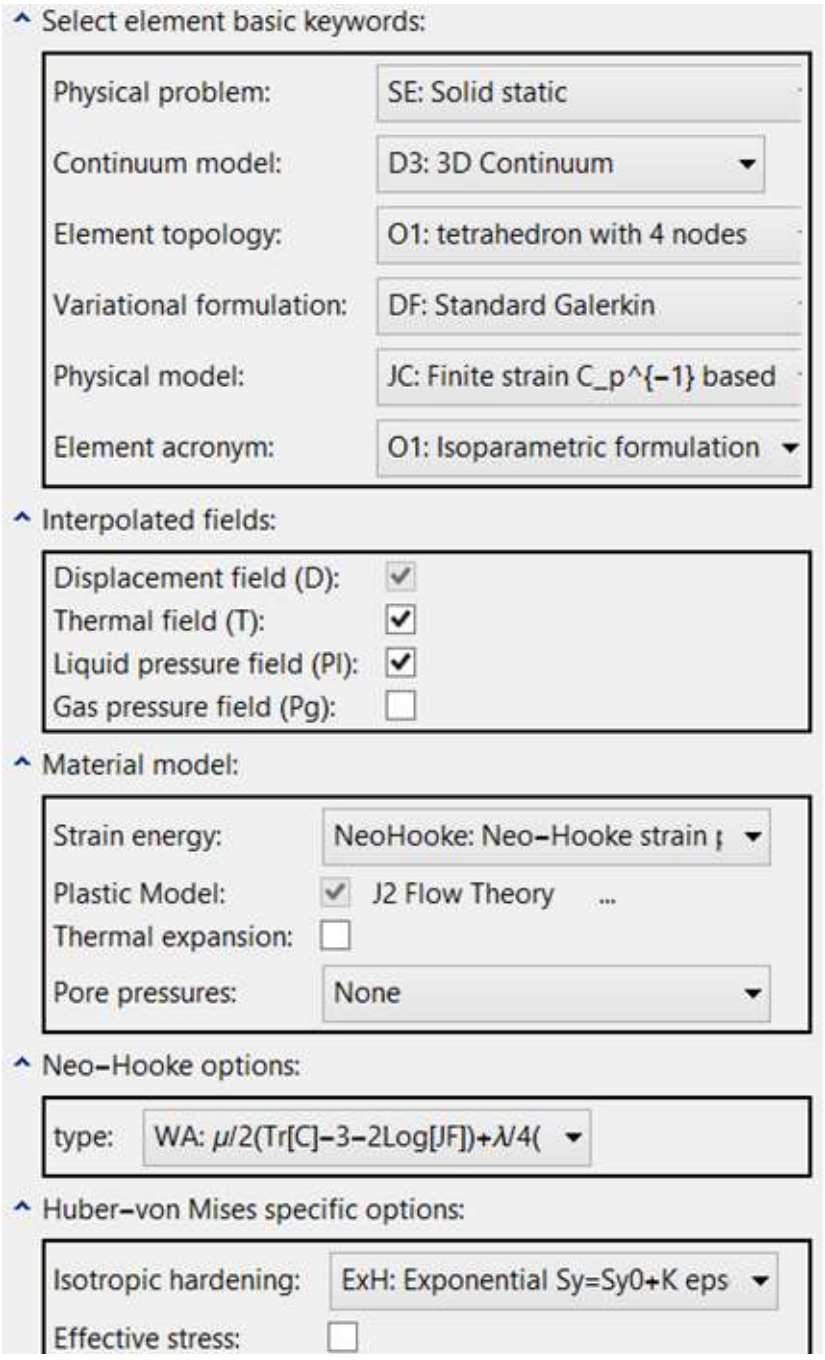
The *Element selector* menu can be used to locate the element, paste the element code into notebook, get information about material

constants and available post processing keywords, and also to access element's home page with the links to benchmark tests and source codes.

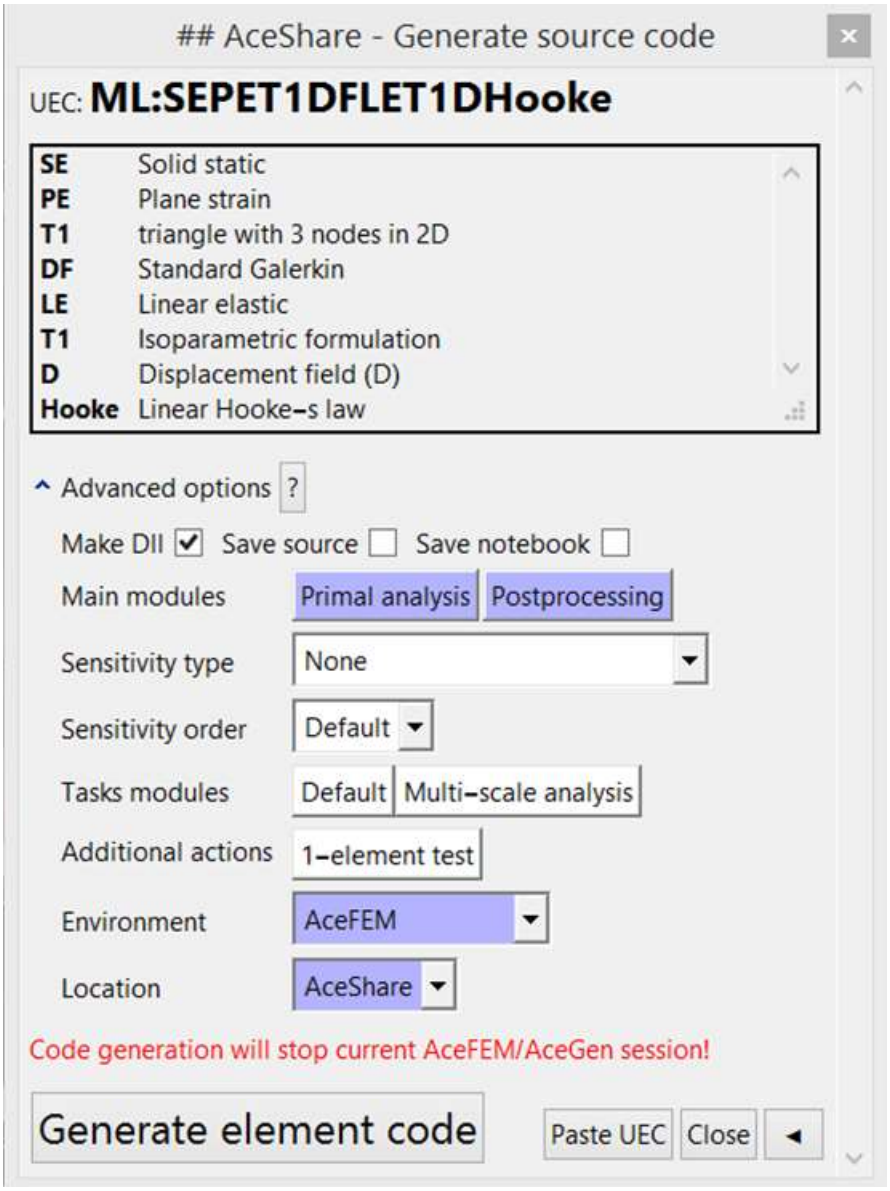


Searching the AceShare libraries for the precomputed finite element source codes.

From the practical reasons the number of precomputed elements is limited to the most commonly used elements. Additionally to the precomputed elements, the on-line library contains also mechanism for the run-time generation of finite elements as shown below for the ML library



Selecting the combination of keywords for the run-time generation of finite element source codes.



Run-time generation of the selected finite element.

Create Your Own AceShare Library

The standard procedure for creation of the third-party AceShare library is composed of following steps:

- first a system of keyword has to be established and the library has to be initialized ;
- the *libraryNotebook* is a notebook that contains the AceGen source code for all library elements;
- posting of the library on internet.

Library initialization

SMTSetLibrary[*libdir*]
creates a new local library at directory *libdir* or sets the active local library to be the one that exists at local directory *libdir*

Initialize the library.

option	default	description
"Code"	"UL"	two letter code that uniquely identifies the library
"Title"	"User Library"	the short name that describes the contents of the library (e.g. "large strain plasticity models")
"URL"	"xxx"	the Internet address where the library will be posted later
"Keywords"	{}	the system of keywords used to create palettes for run-time generation
"Contents"	"xxx"	the description of the contents of the library
"Contact"	"xxx"	the contact address of the corresponding author or institution
"RecreateNotebook"	Atomatic	True if automatic generation of the notebook with AceGen input that can be used to modify the element is supported

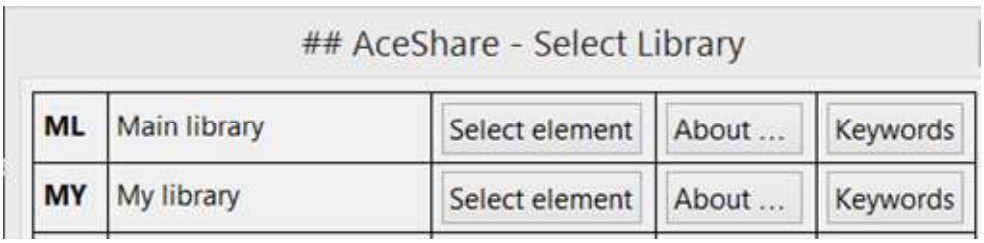
Options of the SMTSetLibrary function.

If the directory *libdir* exists and contains valid previously created *AceShare* library then the SMTSetLibrary command sets the active user library to be the one that already exists at *libdir*. If directory *libdir* does not exist, then the SMTSetLibrary command creates a new directory *libdir* and initializes its contents. The *SMTSetLibrary* command must be a part of the *libraryNotebook.* The *libraryNotebook* is automatically saved and copied to the *libdir* whenever the library is initialized. The *libdir* contains all the information needed for the posting the library on the internet.

A list of keywords has to be organized hierarchically. Each additional level has to be introduced by {…, "Description"}->"text" followed by alternative possibilities for the specified level. For example:

```
"Keywords" -> {
  (* a branch*)
  {"Description"} → "what first level means",
  {"a"} -> "what a means",
  {"a", "Description"} → "what second level means",
  {"a", "b1"} -> "what b1 means",
  {"a", "b2"} -> "what b2 means",
  {"a", _, "Description"} → "what third level means",
  {"a", _, "c1"} -> "what c1 means",
  {"a", _, "c2"} -> "what c2 means",

  (* b branch*)
  {"Description"} → "what first level means",
  {"b"} -> "what b means",
  {"b", "Description"} → "what second level means",
  {"b", "b1"} -> "what b1 means",
  {"b", "b2"} -> "what b2 means",
  {"b", _, "Description"} → "what third level means",
  {"b", _, "c1"} -> "what c1 means",
  {"b", _, "c2"} -> "what c2 means"
 }
```

The SMTSetLibrary command also adds the link to local library directory to the list of active AceShare libraries. An additional item now appears in the list of AceShare libraries under the keyword e.g. "MY".



User generated *AceShare* libraries.

Run-time generation of the element

The run-time generation of the element follows the following procedure:

- first the combination of the keywords is selected and the button "Generate element code" is pressed
- the run-time mechanism then downloads the *libraryNotebook* from the server (the *libraryNotebook* is kept private and is never stored locally on the users disc!)
- the run-time mechanism then sets the interface data and executes the Cell with the tag "Make element" that is obligatory part of the *libraryNotebook*

Run-time interface data

The run-time mechanism passes to the *libraryNotebook* two global variables:

- SMCKeywords variable contains a list of combination of keywords selected by the user;
- SMCLibData is the following data structure

```
SMCLibData = {
 elementCode,

 environment,

 {
  True if standard module "Tangent and residual" is needed,
  True if standard module "Postprocessing" is needed,
  listOfTasks that have to be supported by the standard module "Tasks",
  sensitivityAnalysisType,
  sensitivityAnalysisOrder,
  listOfAdditionalAction,
 sensitivityVelocityFieldOrder
  },

 True if automatic generation of the notebook with AceGen is required,

 element description (automatically generated from SMCKeywords),

 libraryCode,

 code optimisation level ("Prototype"...)
 }
```

sensitivityAnalysisType (see Introduction to Sensitivity Analysis)	description
None	sensitivity analysis is not supported
"All types"	all below
"Essential boundary conditions"	sensitivity with respect to the change of parameters used to parameterize the essential boundary conditions
"Shape sensitivity"	sensitivity with respect change of parameters used to parameterize the mesh
"Domain data"	sensitivity with respect to material data, etc. (see)
"Asymptotic numerical methods"	Asymptotic numerical method is a method where sensitivity analysis is performed with respect to the parameter of the nonlinear problem, thus we have only one sensitivity parameter
"Implicit"	only implicit sensitivity in the case of time/path dependent problems

sensitivityAnalysisOrder	description
Default	second order sensitivity analysis
1	first order sensitivity analysis
2	second order sensitivity analysis
>2	only supported in the case of asymptotic numerical methods

listOfTasks (see User Defined Tasks)	description
Default	library and/or element dependent
"Multi-scale analysis"	necessary tasks for FE^2 and MIEL multi-scale analysis (see Documentation for Multi-scale Computational Environment)

listOfAdditionalActions	description
"1-element test"	benchmark 1-element test of code correctness

The SMCKeywords and SMCLibData must be properly interpreted by the Cells with the tag "Make element" in order to make the whole mechanism of run-time element generation working properly.

The automatically generated notebook with AceGen input can be used to modify the element by the third party user to create new types of elements.

The element UEC can be obtained from the SMCKeywords as

```
UEC = StringJoin[libraryCode, ":", SMCKeywords]
```

Posting third-party library on AceShare

So far, the generated library is located at the local directory and accessible only from the current AceFEM session. In order to make the new library available at any time and for everybody copy the complete library directory *libdir* to the publicly accessible server exactly to the location given by the SMTSetLibrary option "URL"->library_*url*. After that, please send the URL to AceProducts@fgg.uni-lj.si to make third-party library available to all AceFEM users.

Example of simple AceShare library

Initialize library

Run initialization whenever the library notebook "MyLib.nb" has been changed or you want to use library locally.

For the public use of the library, copy created directory "C:/MYLib/" to server location "http://symech.fgg.uni-lj.si/MYLib/" and sent notification to AceProducts@fgg.uni-lj.si.

In[186]:=
```
<< AceFEM`; << AceGen`;
SMTSetLibrary[
 "C:/MYLib/"
 , "Code" → "MY"
 , "Title" → "My library"
 , "URL" → "http://symech.fgg.uni-lj.si/MYLib/"
 , "Keywords" → {
   {"Description"} -> "Continuum model",
   {"D3"} → "3D elements",
   {"D3", "Description"} -> "Interpolation",
   {"D3", "H1"} → "8 noded",
   {"D3", _, "Description"} -> "Physical problem",
   {"D3", _, "TH"} → "thermal",
   {"D3", _, _, "Description"} -> "Material",
   {"D3", _, "TH", "Lin"} → "Constant conductivity",
   {"D3", _, "TH", "Nonlin"} → "Nonlinear conductivity"
  }
 , "Contents" -> "Test library"
 , "Contact" -> "University of Ljubljana, xxx@yyy.si"
 , "RecreateNotebook" → True
]
```

Generation of elements

- By setting the global variables SMCKeywords and SMCLibData user can generate and test the elements locally without the use of the library mechanism. Only after the elements are fully developed and tested it is advisable to activate the run-time finite element generation mechanism.

In[247]:=
```
<< AceGen`;
```

Generate linear element

In[251]:=
```
SMCKeywords = {"D3", "H1", "TH", "Lin"};
SMCLibData = {
    (*element code*)StringJoin[SMCKeywords],
    (*target environment*)"AceFEM",
    {
     (*1 Tangent and residual*)True
     (*2 Postprocessing*), False
     (*3 Tasks*), {}
     (*4 Sensitivity type*), None
     (*5 Sensitivity order*), Default
    },
    (*automatic generation of notebook*)False,
    (*element description based on keywords*)"test run",
    (*library code*)"MY"
   };
```

- The generation of the element starts in the Cell with the cell tag "Make element"! Element is generated for the given values of the global variables SMCKeywords and SMCLibData.

In[253]:=
```
SMSEvaluateCellsWithTag["Make element"];
```

```
[1s e160] Include Tag : Make element (1 cells found, 1 evaluated)

[3]  Consistency check - expressions
```

File: D3H1THLin.c **Size:** 11 251 **Time:** 3

Method	SKR
No.Formulae	198
No.Leafs	2692

- The generated element is here included into library at C:\MYLib directory.

In[254]:=
```
SMTAddToLibrary[StringJoin[SMCKeywords], SMCKeywords]
```

Generate nolinear element

In[255]:=
```
SMCKeywords = {"D3", "H1", "TH", "Nonlin"};
SMCLibData = {
    (*element code*)StringJoin[SMCKeywords],
    (*target environment*)"AceFEM",
    {
     (*1 Tangent and residual*)True
     (*2 Postprocessing*), False
     (*3 Tasks*), {}
     (*4 Sensitivity type*), None
     (*5 Sensitivity order*), Default
    },
    (*automatic generation of notebook*)False,
    (*element description based on keywords*)"test run",
    (*library code*)"MY"
   };
```

- The generation of the element starts in the Cell with the cell tag "Make element"! Element is generated for the given values of the global variables SMCKeywords and SMCLibData.

In[257]:=
```
SMSEvaluateCellsWithTag["Make element"];
```

```
Include Tag : Make element (1 cells found, 1 evaluated)

[3]  Consistency check - expressions
```

File: D3H1THNonlin.c **Size:** 11 761 **Time:** 3

Method	SKR
No.Formulae	214
No.Leafs	2845

- The generated element is here included into library at C:\MYLib directory.

In[258]:=
```
SMTAddToLibrary[StringJoin[SMCKeywords], SMCKeywords]
```

Element definitions (cell tag "Make element")

Element is here generated for the given values of the global variables SMCKeywords and SMCLibData. DO NOT EVALUETE!

In[197]:=
```
SMSInitialize[SMSTagEvaluate[SMCLibData[[1]]],
  "Environment" → SMSTagEvaluate[SMCLibData[[2]]]];
SMSTemplate["SMSTopology" → "H1", "SMSDOFGlobal" → 1, "SMSSymmetricTangent" → False
  , "SMSDomainDataNames" → SMSTagSwitch[SMCKeywords[[4]]
    , "Lin", {"k0 -conductivity parameter k0", "Q -heat source"}
    , "Nonlin", {"k0 -conductivity parameter", "k1 -conductivity parameter",
     "k2 -conductivity parameter", "Q -heat source"}
   ]
  , "SMSDefaultData" → SMSTagSwitch[SMCKeywords[[4]]
    , "Lin", {1, 0}
    , "Nonlin", {1, 0, 0, 0}
   ]
 ];
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
XIO ⊨ SMSIO["Nodal coordinates"];
Ξn = {{-1, -1, -1}, {1, -1, -1}, {1, 1, -1}, {-1, 1, -1},
   {-1, -1, 1}, {1, -1, 1}, {1, 1, 1}, {-1, 1, 1}};
Nh ⊨ Table[1 / 8 (1 + ξ Ξn[[i, 1]]) (1 + η Ξn[[i, 2]]) (1 + ζ Ξn[[i, 3]]), {i, 1, 8}];
SMSFreeze[X, Nh.XIO]; Je ⊨ SMSD[X, Ξ]; Jed ⊨ Det[Je];
ϕI ⊨ Flatten[SMSIO["Nodal DOFs"]];
ϕ ⊨ Nh.ϕI;
SMSTagSwitch[SMCKeywords[[4]]
  , "Lin",
  {k0, Q} ⊨ SMSIO["Domain data"];
  k ⊨ k0;
  , "Nonlin",
  {k0, k1, k2, Q} ⊨ SMSIO["Domain data"];
  k ⊨ k0 + k1 ϕ + k2 ϕ^2;
 ];
λ ⊨ SMSIO["Multiplier"];
wgp ⊨ SMSIO["Integration weight"[Ig]];
Dϕ ⊨ SMSD[ϕ, X, "Dependency" -> {Ξ, X, SMSInverse[Je]}];
W ⊨ 1/2 k Dϕ.Dϕ - ϕ λ Q;
SMSDo[
  Rg ⊨ Jed SMSD[W, ϕI, i, "Constant" → k];
  SMSIO[wgp Rg, "Add to", "Residual"[i]];
  SMSDo[
   Kg ⊨ SMSD[Rg, ϕI, j];
   SMSIO[wgp Kg, "Add to", "Tangent"[i, j]];
   , {j, 1, 8}
  ];
  , {i, 1, 8}
 ];
SMSEndDo[];
SMSWrite[];
```

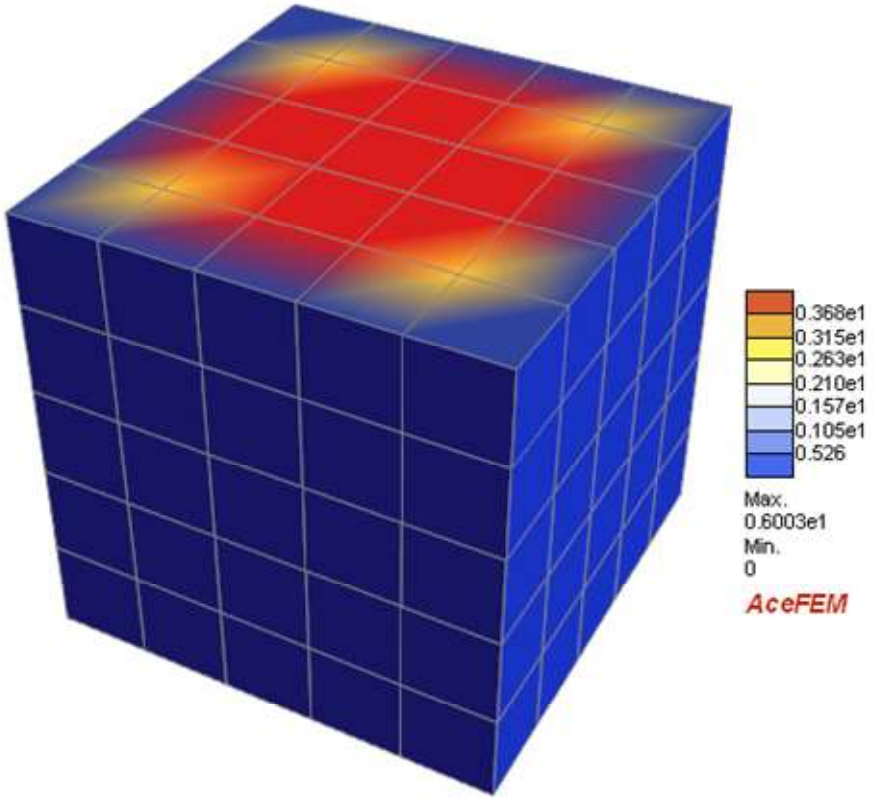
Benchmark test example

The elements from the active local library can be directly accessed by the element UEC of the form {"MY:","D3","H1","TH","Lin"} and used within the *AceFEM* session.

In[259]:=
```
<< AceFEM`;
SMTInputData[];
Q = 50;
nn = 5;
SMTAddDomain["cube", {"MY:", "D3", "H1", "TH", "Lin"}, {"k0 *" -> 0.58, "Q *" -> Q}];
SMTAddEssentialBoundary[
  {"X" == -0.5 || "X" == 0.5 || "Y" == -0.5 || "Y" == 0.5 || "Z" == 0. &, 1 -> 0}];
SMTAddMesh[Hexahedron[{{-0.5, -0.5, 0}, {0.5, -0.5, 0}, {0.5, 0.5, 0}, {-0.5, 0.5, 0},
    {-0.5, -0.5, 1}, {0.5, -0.5, 1}, {0.5, 0.5, 1}, {-0.5, 0.5, 1}}], "cube", "H1", {nn, nn, nn}];
SMTAnalysis[];
```

In[267]:=
```
SMTNextStep["λ" → 1];
While[step = SMTConvergence[10^-8, 10], SMTNewtonIteration[];];
```

In[269]:=
```
SMTShowMesh["Field" → SMTPostData[{"at", 1}]]
```



Debugging of AceGen-AceFEM Codes

Contents

Introduction

Several possibilities for debugging of automatically generated codes are already presented in AceGen manual section `Verification and Debugging`. However, the potential huge codes and large scale calculations involved in finite element solution procedures puts a lot of restrictions on how and what can be used for debugging of finite element codes.

Basic ways how to verify and debug the AceGen generated FE codes are:

- by creating the codes using the AceGen debug mode SMSInitialize[..."Mode"→"Debug"] the consistency of C code is checked,
- print out the chosen intermediate results while running the AceFEM to notebook, console or file to check intermediate numerical values,
- more sophisticated is interactive run-time debugging where AceGen and AceFEM run in parallel, thus generated formulas can be analyzed as well.

Additional suggestions:

- Printing from finite element user subroutines might be problematic when the program is parallelized. Please turn off parallelization with SMTInputData["Threads" -> 1] command.
- When the CDriver is run as console application (SMTInputData["Console"→True]) useful additional information is printed out to console window.
- Any C print statement (e.g. printf("hello")) inserted directly into the element C source file will appear in console window (SMTInputData["Console"→True]).
- Additional useful information is also printed out to output file when defined (SMTAnalysis["Output"→*filename*]).
- When the code is generated for user defined environment, always first check it in AceFEM. The sophisticated debugging options available in AceFEM are usually not available in standard FEM environments.

Running AceGen and AceFEM in debug mode

- The 2D, finite strain, hyper-elastic element is generated here. For more details see `Simple 2 D Solid, Finite Strain Element`.
- When the code is generated in "Debug" mode, an additional tests are performed during the code generati0n. Additional test are also inserted into the generated code and performed at run-time. Consequently, the code created in debug mode is VERY SLOW!

In[308]:=
```
<< "AceGen`";
SMSInitialize["Debug2D", "Environment" → "AceFEM", "Mode" → "Debug"];
SMSTemplate["SMSTopology" → "Q1", "SMSSymmetricTangent" → True,
  "SMSDomainDataNames" -> {"E -elastic modulus", "ν -poisson ratio", "t -thickness"},
  "SMSDefaultData" -> {21000, 0.3, 1}
 ];
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
XIO ⊨ SMSIO["Nodal coordinates"];
Nh ⊨ 1 / 4 {(1 - ξ) (1 - η), (1 + ξ) (1 - η), (1 + ξ) (1 + η), (1 - ξ) (1 + η)};
SMSFreeze[X, Append[Nh.XIO, ζ]];
Je ⊨ SMSD[X, Ξ]; Jed ⊨ Det[Je];
uIO ⊨ SMSIO["Nodal DOFs"];
pe = Flatten[uIO]; u ⊨ Append[Nh.uIO, 0];
H ⊨ SMSD[u, X, "Dependency" → {Ξ, X, SMSInverse[Je]}];
SMSFreeze[F, IdentityMatrix[3] + H, "Ignore" → PossibleZeroQ];
JF ⊨ Det[F]; Ct ⊨ Transpose[F].F;
{Em, ν, tζ} ⊨ SMSIO["Domain data"];
{λ, μ} ⊨ SMSHookeToLame[Em, ν];
W ⊨ 1 / 2 λ (JF - 1) ^ 2 + μ (1 / 2 (Tr[Ct] - 3) - Log[JF]);
wgp ⊨ SMSIO["Integration weight"[Ig]];
SMSDo[
  Rg ⊨ Jed tζ SMSD[W, pe, i];
  SMSIO[wgp Rg, "Add to", "Residual"[i]];
  SMSDo[
   Kg ⊨ SMSD[Rg, pe, j];
   SMSIO[wgp Kg, "Add to", "Tangent"[i, j]];
   , {j, i, 8}];
  (* lets make intentional mistake here, by index running to 9 instead to 8*)
  , {i, 1, 9}];
SMSEndDo[];
SMSWrite[];
```

```
 time=0  variable= 0

 time=1  variable= 100

 [1]  Consistency check - global

 [1]  Consistency check - expressions

 Events: 14 SMSDB-0

 [1]  Generate source code :

 [1] Final formating
```

File: Debug2D.c **Size:** 16071 **Time:** 1

Method	SKR
No.Formulae	116
No.Leafs	2173

In[330]:=
```
<< AceFEM`;
SMTInputData["Console" → True];
SMTAddDomain[{"Test", "Debug2D", {"E *" -> 1, "ν *" -> 0}}];
SMTAddMesh[Polygon[{{0, 0}, {48, 44}, {48, 60}, {0, 44}}], "Test", "Q1", {2, 2}];
SMTAddEssentialBoundary[Line[{{0, 0}, {0, 44}}], 1 → 0, 2 → 0];
SMTAddNaturalBoundary[ Line[{{48, 44}, {48, 60}}], 2 → Line[{-1}]];
SMTAnalysis[];
SMTNextStep["Δλ" → 0.1];
```

- Intentional error has created memory fault situation and caused the CDriver to crash. Due to the "Debug" option the reason for the crash is clearly identified.

In[338]:=
```
SMTNewtonIteration[];
```

Index out of bounds error while executing user subroutine SKR of element Debug2D (possible bug in the symbolic description of the element!!).

Description : Index in expression: p$$[i] has value: 9. Index bounds are min= 1 max= 8. Source code identification: $D48$

Version: 7.101 Windows (18 Feb 20) (MMA 12.) **Module:** SMTBoundCheck

See also: Debugging AceFEM Troubleshooting Continue

--- LinkObject: Unable to communicate with closed link LinkObject["C:\Users\jkorelc\AppData\Roaming\Mathematica\Applications\AceFEM\Include\CDriver\Win64\ WinConsoleDriver64.exe", 14969, 11].

Printing from FEM codes

Printing from the automatically generated codes is done by `SMSPrint` command. AceFEM related options of the `SMSPrint` command are described below and presented on examples in the subsequent sections.

Options	description
"Output"–>"Console"	create a source code sequence that prints out to Mathematica notebook
"Output"–>"File"	create a source code sequence that prints out to the output file defined by SMTAnalysis["Output"→*filename*] command
"Output"–>"AceFEM console"	create a source code sequence that prints out to CDriver console when CDriver is started as console application by SMTInputData["Console"→True] command
"Condition"–>"DebugElement"	With the "DebugElement" option (SMSPrint[…,"Condition"–>"DebugElement"]) the print out is executed accordingly to the value of the SMTIData["DebugElement"] environment variable: SMTIData["DebugElement",–1] ⇒ print outs are active for all elements SMTIData["DebugElement",0] ⇒ print outs are disabled (default value) SMTIData["DebugElement",*i*] ⇒ print out is active for *i*–th element
"Optimal"→True	By default the printing code is generated only in "Debug" and "Prototype" mode. With the option "Optimal"→ True the printing source code is always generated.

AceFEM related SMSPrint options.

Unconditional printing to notebook

- The 2D, finite strain, hyper-elastic element is generated here. For more details see `Simple 2 D Solid, Finite Strain Element`.
- By default the printing code is generated only in "Debug" and "Prototype" mode, thus "Mode"→"Prototype".

In[341]:=
```
<< "AceGen`";
SMSInitialize["Debug2D", "Environment" → "AceFEM", "Mode" → "Prototype"];
```

In[343]:=
```
ElementDefinitions[] := (
    SMSTemplate["SMSTopology" → "Q1", "SMSSymmetricTangent" → True,
     "SMSDomainDataNames" -> {"E -elastic modulus", "ν -poisson ratio", "t -thickness"},
     "SMSDefaultData" -> {21000, 0.3, 1}
    ];
    SMSStandardModule["Tangent and residual"];
    SMSDo[Ig, 1, SMSIO["No. integration points"]];
    Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
    XIO ⊨ SMSIO["Nodal coordinates"];
    Nh ⊨ 1 / 4 {(1 - ξ) (1 - η), (1 + ξ) (1 - η), (1 + ξ) (1 + η), (1 - ξ) (1 + η)};
    SMSFreeze[X, Append[Nh.XIO, ζ]];
    Je ⊨ SMSD[X, Ξ]; Jed ⊨ Det[Je];
    uIO ⊨ SMSIO["Nodal DOFs"];
    pe = Flatten[uIO]; u ⊨ Append[Nh.uIO, 0];
    H ⊨ SMSD[u, X, "Dependency" → {Ξ, X, SMSInverse[Je]}];
    SMSFreeze[F, IdentityMatrix[3] + H, "Ignore" → PossibleZeroQ];
    JF ⊨ Det[F]; Ct ⊨ Transpose[F].F;
    {Em, ν, tζ} ⊨ SMSIO["Domain data"];
    {λ, μ} ⊨ SMSHookeToLame[Em, ν];
    W ⊨ 1 / 2 λ (JF - 1) ^ 2 + μ (1 / 2 (Tr[Ct] - 3) - Log[JF]);
    wgp ⊨ SMSIO["Integration weight"[Ig]];
    SMSDo[
     Rg ⊨ Jed tζ SMSD[W, pe, i];
     SMSIO[wgp Rg, "Add to", "Residual"[i]];
     SMSDo[
      Kg ⊨ SMSD[Rg, pe, j];
      SMSIO[wgp Kg, "Add to", "Tangent"[i, j]];
      , {j, i, 8}];
     , {i, 1, 8}];
    SMSEndDo[];
   );
```

In[344]:=
```
ElementDefinitions[]
```

- Print out the element number and the diagonal elements of the element tangent matrix.

In[345]:=
```
SMSPrintMessage["elem", SMSInteger[ed$$["id", "ElemIndex"] + 1], "K", Table[s$$[i, i], {i, 8}]];
```

In[346]:=
```
SMSWrite[];
```

```
[1]  Consistency check - expressions
```

File: Debug2D.c Size: 14397 Time: 1

Method	SKR
No.Formulae	100
No.Leafs	1986

In[347]:=
```
<< AceFEM`;
```

- Printing from finite element user subroutines might be problematic when the program is parallelized. Please turn off parallelization with SMTInputData["Threads" -> 1] command.

In[348]:=
```
SMTInputData["Threads" → 1];
```

In[349]:=
```
SMTAddDomain[{"Test", "Debug2D", {"E *" -> 1, "ν *" -> 0}}];
SMTAddMesh[Polygon[{{0, 0}, {48, 44}, {48, 60}, {0, 44}}], "Test", "Q1", {2, 2}];
SMTAddEssentialBoundary[Line[{{0, 0}, {0, 44}}], 1 → 0, 2 → 0];
SMTAddNaturalBoundary[ Line[{{48, 44}, {48, 60}}], 2 → Line[{-1}]];
SMTAnalysis[];
SMTNextStep["Δλ" → 0.1];
```

- SMTNewtonIteration calls generated element user subroutines that print out the required data into notebook.

In[355]:=
```
SMTNewtonIteration[];
```

```
elem 1. K 0.283606 0.438994 1.19424 0.956408 0.423402 0.570991 1.05444 0.824411

elem 2. K 0.284972 0.439677 0.873747 0.796164 0.394581 0.556581 0.764139 0.67926

elem 3. K 0.420216 0.666545 1.48025 1.36041 0.709417 0.974995 1.19105 1.05196

elem 4. K 0.34417 0.628522 1.03289 1.13673 0.553722 0.897147 0.823337 0.868105
```

Printing to notebook from selected element

In[356]:=
```
<< "AceGen`";
SMSInitialize["Debug2D", "Environment" → "AceFEM", "Mode" → "Prototype"];
```

- Please execute the previously defined ElementDefinitions[] cell first.

In[358]:=
```
ElementDefinitions[]
```

- Print out the element number and the diagonal elements of the element tangent matrix, however only when the chosen "DebugElement" element is evaluated.

In[359]:=
```
SMSPrint["elem", SMSInteger[ed$$["id", "ElemIndex"] + 1],
  "K", Table[s$$[i, i], {i, 8}], "Condition" -> "DebugElement"];
SMSWrite[];
```

```
[1]  Consistency check - expressions

File: Debug2D.c  Size: 14397  Time: 1
```

Method	SKR
No.Formulae	100
No.Leafs	1986

In[361]:=
```
<< AceFEM`;
```

In[362]:=
```
SMTInputData["Threads" → 1];
```

In[363]:=
```
SMTAddDomain[{"Test", "Debug2D", {"E *" -> 1, "ν *" -> 0}}];
SMTAddMesh[Polygon[{{0, 0}, {48, 44}, {48, 60}, {0, 44}}], "Test", "Q1", {2, 2}];
SMTAddEssentialBoundary[Line[{{0, 0}, {0, 44}}], 1 → 0, 2 → 0];
SMTAddNaturalBoundary[ Line[{{48, 44}, {48, 60}}], 2 → Line[{-1}]];
SMTAnalysis[];
SMTNextStep["Δλ" → 0.1];
```

In[369]:=
```
SMTNewtonIteration[];
```

```
elem 1. K 0.283606 0.438994 1.19424 0.956408 0.423402 0.570991 1.05444 0.824411

elem 2. K 0.284972 0.439677 0.873747 0.796164 0.394581 0.556581 0.764139 0.67926

elem 3. K 0.420216 0.666545 1.48025 1.36041 0.709417 0.974995 1.19105 1.05196

elem 4. K 0.34417 0.628522 1.03289 1.13673 0.553722 0.897147 0.823337 0.868105
```

- The printing is done accordingly to the value of the SMTIData["DebugElement"] variable. The scope of the printing can be limited to one element by the command SMTIData["DebugElement",*elementnumber*], all elements by the command SMTIData["DebugElement",-1] or none by the command SMTIData["DebugElement",0]. Default value is SMTIData["DebugElement",0].
- Here the element 2 is specified as the element for which the break point is activated.

In[370]:=
```
SMTIData["DebugElement", 2];
```

In[371]:=
```
SMTNewtonIteration[];
```

```
elem 2. K 0.304057 0.426902 0.767671 0.889838 0.384382 0.54354 0.628767 0.711887
```

Printing to file

In[372]:=
```
<< "AceGen`";
SMSInitialize["Debug2D", "Environment" → "AceFEM", "Mode" → "Prototype"];
```

- Please execute the previously defined ElementDefinitions[] cell first.

In[374]:=
```
ElementDefinitions[]
```

- Create a source code sequence that prints out to the output file defined by SMTAnalysis["Output"→filename] command.

In[375]:=
```
SMSPrint["elem", SMSInteger[ed$$["id", "ElemIndex"] + 1],
  "K", Table[s$$[i, i], {i, 8}], "Output" → "File"];
SMSWrite[];
```

```
[1]  Consistency check - expressions

File: Debug2D.c  Size: 13208  Time: 1
```

Method	SKR
No.Formulae	91
No.Leafs	1847

In[377]:=
```
<< AceFEM`;
SMTInputData["Threads" → 1];
SMTAddDomain[{"Test", "Debug2D", {"E *" -> 1, "ν *" -> 0}}];
SMTAddMesh[Polygon[{{0, 0}, {48, 44}, {48, 60}, {0, 44}}], "Test", "Q1", {2, 2}];
SMTAddEssentialBoundary[Line[{{0, 0}, {0, 44}}], 1 → 0, 2 → 0];
SMTAddNaturalBoundary[ Line[{{48, 44}, {48, 60}}], 2 → Line[{-1}]];
```

- Definition of the file to which printing will be done.

In[383]:=
```
SMTAnalysis["Output" → "debug.txt"];
```

In[384]:=
```
SMTNextStep["Δλ" → 0.1];
SMTNewtonIteration[];
```

- Display the contents of the file.

In[386]:=
```
FilePrint["debug.txt"]
```

```
Input data file       = .h5
Input notebook        = AceFEMTutorials
Report file           = debug.txt
Working directory     = C:\Users\jkorelc\Desktop\SMS\Documentation
Steering dll file     =
AceShare directory     = C:\\Users\\jkorelc\\Desktop\\SMS\\Documentation\\AceShare\\
Number of nodes       = 10
Number of elements    = 4
Data storage (KBytes) = 3
Number of threads     = 1


ELEMENT ............ = Debug2D
                   1 = E -elastic modulus
                   0 = $[Nu]$ -poisson ratio
                   1 = t -thickness
Set solver            = PARDISO
Number of equations   = 12
Storage(KBytes)       = tangent=0 solver=0 total=0
Matrix type           = -2

14:10:44    3.44    3.44  Next step update

14:10:44    3.45    0.01  Predictor iteration
elem 1 K 0.283606 0.438994 1.19424 0.956408 0.423402 0.570991 1.05444 0.824411
elem 2 K 0.284972 0.439677 0.873747 0.796164 0.394581 0.556581 0.764139 0.67926
elem 3 K 0.420216 0.666545 1.48025 1.36041 0.709417 0.974995 1.19105 1.05196
elem 4 K 0.34417 0.628522 1.03289 1.13673 0.553722 0.897147 0.823337 0.868105
14:10:44    3.47    0.02  K and R loop
Solver memory (+filin)= 147 KBytes
14:10:44    3.58    0.11  Linear solution (12)
  Step=1 Iteration=1 Mode(l/g)=0/0 Events=0 |R|=0.282843 |da|=10.0294
  Time(t/dt)=0/0 Multiplier(l/dl)=0.1/0.1
```

Combined printing options

In[387]:=
```
<< "AceGen`";
SMSInitialize["Debug2D", "Environment" → "AceFEM", "Mode" → "Prototype"];
```

- Please execute the previously defined ElementDefinitions[] cell first.

In[389]:=
```
ElementDefinitions[]
```

- Options to SMSPrint command can be combined. Here the printing is done to CDriver console,notebook and output file.

In[390]:=
```
SMSPrint["elem", SMSInteger[ed$$["id", "ElemIndex"] + 1], "K", Table[s$$[i, i], {i, 8}]
  , "Condition" -> "DebugElement", "Output" → {"AceFEM console", "Console", "File"}];
SMSWrite[];
```

```
[1]  Consistency check - expressions
```

File: Debug2D.c **Size:** 15623 **Time:** 1

Method	SKR
No.Formulae	100
No.Leafs	1986

In[392]:=
```
<< AceFEM`;
SMTInputData["Threads" → 1, "Console" → True];
SMTAddDomain[{"Test", "Debug2D", {"E *" -> 1, "ν *" -> 0}}];
SMTAddMesh[Polygon[{{0, 0}, {48, 44}, {48, 60}, {0, 44}}], "Test", "Q1", {2, 2}];
SMTAddEssentialBoundary[Line[{{0, 0}, {0, 44}}], 1 → 0, 2 → 0];
SMTAddNaturalBoundary[ Line[{{48, 44}, {48, 60}}], 2 → Line[{-1}]];
SMTAnalysis["Output" → "debug.txt"];
SMTNextStep["Δλ" → 0.1];
SMTIData["DebugElement", 1];
SMTNewtonIteration[];
```

In[402]:=
```
FilePrint["debug.txt"]
```

```
Input data file       = .h5
Input notebook        = AceFEMTutorials
Report file           = debug.txt
Working directory     = C:\Users\jkorelc\Desktop\SMS\Documentation
Steering dll file     =
AceShare directory     = C:\\Users\\jkorelc\\Desktop\\SMS\\Documentation\\AceShare\\
Number of nodes       = 10
Number of elements    = 4
Data storage (KBytes) = 3
Number of threads     = 1


ELEMENT ............. = Debug2D
                    1 = E -elastic modulus
                    0 = $[Nu]$ -poisson ratio
                    1 = t -thickness
Set solver            = PARDISO
Number of equations   = 12
Storage(KBytes)       = tangent=0 solver=0 total=0
Matrix type           = -2

14:11:08    0.45    0.45  Next step update

14:11:08    0.46    0.01  Predictor iteration
elem 1 K 0.283606 0.438994 1.19424 0.956408 0.423402 0.570991 1.05444 0.824411
14:11:08    0.48    0.01  K and R loop
Solver memory (+filin)= 147 KBytes
14:11:08    0.59    0.11  Linear solution (12)
  Step=1 Iteration=1 Mode(l/g)=0/0 Events=0 |R|=0.282843 |da|=10.0294
  Time(t/dt)=0/0 Multiplier(l/dl)=0.1/0.1
```

Interactive Debugging

The procedures described in the section `Verification` and `Debugging` of the *AceGen* manual for the run-time debugging of automatically generated codes can be used within the AceFEM environment as well. For the interactive debugging procedures, the code has to be generated in "Debug" mode.

By default compiler compiles the code generated in "Debug" mode with the compiler options for debugging. For a large scale problems this my represent a drawback. Compiler can be forced to produce optimized program also for the code generated in "Debug" mode with the `SMTAnalysis` option "OptimizeDll"->True.

- Here the code for the steady state heat conduction problem (see Standard FE Procedure) is generated in "Debug" mode. The break point (see `SMSSetBreak`) with the identification "k" is inserted.

In[403]:=
```
<< AceGen`;
SMSInitialize["ExamplesDebugHeatConduction", "Environment" -> "AceFEM", "Mode" -> "Debug"];
SMSTemplate["SMSTopology" → "H1", "SMSDOFGlobal" → 1, "SMSSymmetricTangent" → False,
  "SMSDomainDataNames" -> {"k0 -conductivity parameter", "k1 -conductivity parameter",
    "k2 -conductivity parameter", "Q -heat source"},
  "SMSDefaultData" -> {1, 0, 0, 0}];
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
XIO ⊨ SMSIO["Nodal coordinates"];
Ξn = {{-1, -1, -1}, {1, -1, -1}, {1, 1, -1}, {-1, 1, -1},
   {-1, -1, 1}, {1, -1, 1}, {1, 1, 1}, {-1, 1, 1}};
Nh ⊨ Table[1 / 8 (1 + ξ Ξn[[i, 1]]) (1 + η Ξn[[i, 2]]) (1 + ζ Ξn[[i, 3]]), {i, 1, 8}];
SMSFreeze[X, Nh.XIO]; Je ⊨ SMSD[X, Ξ]; Jed ⊨ Det[Je];
ϕI ⊨ Flatten[SMSIO["Nodal DOFs"]];
ϕ ⊨ Nh.ϕI;
{k0, k1, k2, Q} ⊨ SMSIO["Domain data"];
k ⊨ k0 + k1 ϕ + k2 ϕ^2;
SMSSetBreak["k"];
λ ⊨ SMSIO["Multiplier"];
wgp ⊨ SMSIO["Integration weight"[Ig]];
SMSDo[
  Dϕ ⊨ SMSD[ϕ, X, "Dependency" -> {Ξ, X, SMSInverse[Je]}];
  δϕ ⊨ SMSD[ϕ, ϕI, i];
  Dδϕ ⊨ SMSD[δϕ, X, "Dependency" -> {Ξ, X, SMSInverse[Je]}];
  Rg ⊨ Jed (k Dδϕ.Dϕ - δϕ λ Q);
  SMSIO[wgp Rg, "Add to", "Residual"[i]];
  SMSDo[
   Kg ⊨ SMSD[Rg, ϕI, j];
   SMSIO[wgp Kg, "Add to", "Tangent"[i, j]];
   , {j, 1, 8}
  ];
  , {i, 1, 8}
 ];
SMSEndDo[];
SMSWrite[];
```

```
 time=0  variable= 0

 time=1  variable= 100

 time=1  variable= 200

 [2]  Consistency check - global

 [2]  Consistency check - expressions

 Events: 6 SMSDB-0

 [2]  Generate source code :

 [2] Final formating
```

File: ExamplesDebugHeatConduction.c **Size:** 26016 **Time:** 2

Method	SKR
No.Formulae	214
No.Leafs	4489

- The SMTAddMesh function generates nodes and elements for a cube discretized by the 10×10×10 mesh.
  The data and the definitions associated with the derivation of the element are reloaded from the automatically generated file DebugHeatConduction.*dbg*. See also SMSLoadSession.

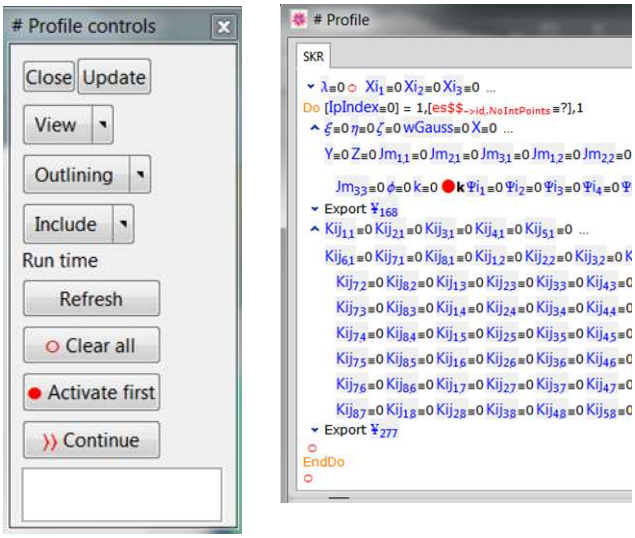
In[423]:=
```
<< AceFEM`;
SMTInputData["LoadSession" -> "ExamplesDebugHeatConduction"];
SMTAddDomain["cube", "ExamplesDebugHeatConduction",
  {"k0 *" -> 1., "k1 *" -> .1, "k2 *" -> .5, "Q *" -> 1.}];
SMTAddEssentialBoundary[
  {"X" == -0.5 || "X" == 0.5 || "Y" == -0.5 || "Y" == 0.5 || "Z" == 0. &, 1 -> 0}];
SMTAddMesh[Hexahedron[{{-0.5, -0.5, 0}, {0.5, -0.5, 0}, {0.5, 0.5, 0}, {-0.5, 0.5, 0},
    {-0.5, -0.5, 1}, {0.5, -0.5, 1}, {0.5, 0.5, 1}, {-0.5, 0.5, 1}}], "cube", "H1", {5, 5, 5}];
SMTAnalysis[];
```

- The break point stops the execution of the program accordingly to the value of the SMTIData["DebugElement"] variable. The scope of the break point can be limited to one element by the command SMTIData["DebugElement",*elementnumber*], all elements by the command SMTIData["DebugElement",-1] or none by the command SMTIData["DebugElement",0]. Default value is SMTIData["DebugElement",0]. Here the element 512 is specified as the element for which the break point is activated.
  The program stops for each integration point where the program structure together with all the basic variables of the problem and the current values of the variables are presented.
  The program can be stopped also when there are no user defined break points by activating the automatically generated break point at the beginning of the chosen module.

In[429]:=
```
SMTIData["DebugElement", 1];
```

In[432]:=
```
SMTNextStep["λ" → 0];
SMTNewtonIteration[];
```

Debugger palette Display

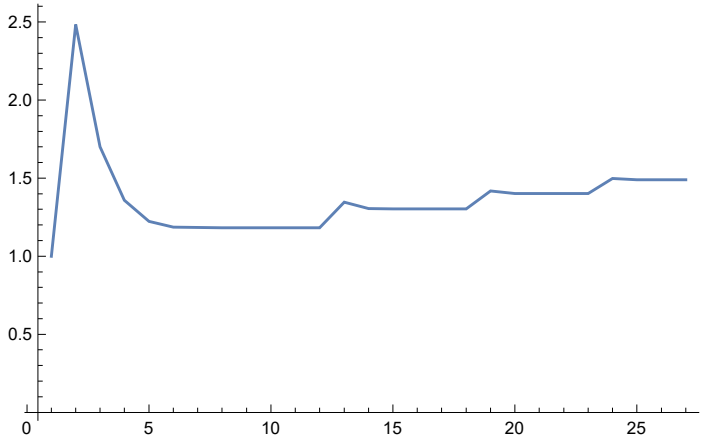


- The break points can be used also to trace the values of arbitrary variables during the analysis. Here the value of the conductivity *k* in the 6-th integration point of the 512-th element is traced during the Newton-Raphson iterative procedure.

In[432]:=
```
allk = {};
SMSActivateBreak["k", If[ Ig == 6, allk = {allk, k};] &];
```

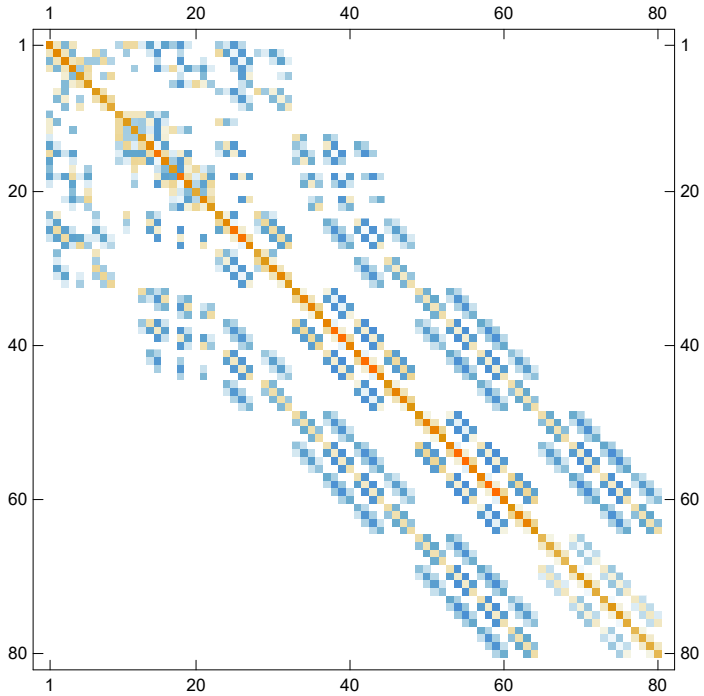
In[434]:=
```
Do[
 SMTNextStep[1, 500];
 While[SMTConvergence[10^-12, 15], SMTNewtonIteration[];];
 , {i, 1, 4}]
```

In[435]:=
```
ListLinePlot[allk // Flatten, PlotRange -> All]
```



- The sparsity structure of the resulting tangent matrix can be graphically displayed by the `ArrayPlot` function.

In[440]:=
```
MatrixPlot[SMTData["TangentMatrix"]]
```



Code Profiling

Code profiling is typically used to understand exactly where an application is spending its execution time.The `SMTSimulationReport` produce report identifying the percentage of time spent in specific tasks. However, by default you only see performance information at the global level rather than at the specific element level. For this additional user defined environment variables has to be defined.

- Here the code for the steady state heat conduction problem (see Standard FE Procedure) is generated. Two additional user defined environment variables are defined where the results of the code profiling will be stored. The real type environment variable "EvaluationTime" will store the total time spent for the evaluation of the element tangent matrix and residual during the analysis. The integer type environment variable "NoEvaluations" will count the number of evaluations.

In[446]:=
```
<< AceGen`;
SMSInitialize["ExamplesProfileHeatConduction", "Environment" -> "AceFEM"];
SMSTemplate["SMSTopology" → "H1", "SMSDOFGlobal" → 1
  , "SMSSymmetricTangent" → False
  , "SMSIDataNames" → {"NoEvaluations"}
  , "SMSRDataNames" → {"EvaluationTime"}
  , "SMSDomainDataNames" -> {"k0 -conductivity parameter", "k1 -conductivity parameter",
    "k2 -conductivity parameter", "Q -heat source"},
  "SMSDefaultData" -> {1, 0, 0, 0}];
SMSStandardModule["Tangent and residual"];
```

- Mark the starting time of the evaluation.

In[450]:=
```
time ⊨ SMSTime[];
```

In[451]:=
```
SMSDo[Ig, 1, SMSIO["No. integration points"]];
Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
XIO ⊨ SMSIO["Nodal coordinates"];
Ξn = {{-1, -1, -1}, {1, -1, -1}, {1, 1, -1}, {-1, 1, -1},
    {-1, -1, 1}, {1, -1, 1}, {1, 1, 1}, {-1, 1, 1}};
Nh ⊨ Table[1 / 8 (1 + ξ Ξn⟦i, 1⟧) (1 + η Ξn⟦i, 2⟧) (1 + ζ Ξn⟦i, 3⟧), {i, 1, 8}];
SMSFreeze[X, Nh.XIO]; Je ⊨ SMSD[X, Ξ]; Jed ⊨ Det[Je];
ϕI ⊨ Flatten[SMSIO["Nodal DOFs"]];
ϕ ⊨ Nh.ϕI;
{k0, k1, k2, Q} ⊨ SMSIO["Domain data"];
k ⊨ k0 + k1 ϕ + k2 ϕ^2;
λ ⊨ SMSIO["Multiplier"];
wgp ⊨ SMSIO["Integration weight"[Ig]];
SMSDo[
  Dϕ ⊨ SMSD[ϕ, X, "Dependency" -> {Ξ, X, SMSInverse[Je]}];
  δϕ ⊨ SMSD[ϕ, ϕI, i];
  Dδϕ ⊨ SMSD[δϕ, X, "Dependency" -> {Ξ, X, SMSInverse[Je]}];
  Rg ⊨ Jed (k Dδϕ.Dϕ - δϕ λ Q);
  SMSIO[wgp Rg, "Add to", "Residual"[i]];
  SMSDo[
   Kg ⊨ SMSD[Rg, ϕI, j];
   SMSIO[wgp Kg, "Add to", "Tangent"[i, j]];
   , {j, 1, 8}
  ];
  , {i, 1, 8}
 ];
SMSEndDo[];
```

- Mark the end time of the evaluation and add the difference to the user defined real type environment variable "EvaluationTime". Increase also the value of the integer type environment variable "NoEvaluations".

In[465]:=
```
SMSExport[SMSTime[] - time, rdata$$["EvaluationTime"], "AddIn" → True];
SMSExport[SMSInteger[idata$$["NoEvaluations"]] + 1, idata$$["NoEvaluations"]];
```

In[467]:=
```
SMSWrite[];
```

```
[3]  Consistency check - expressions
```

File: ExamplesProfileHeatConduction.c Size: 11501 Time: 4

Method	SKR
No.Formulae	214
No.Leafs	2759

- Here is presented an input for the analysis and the results of code profiling for a cube discretized by the 20×20×20 mesh.

In[468]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["cube", "ExamplesProfileHeatConduction",
  {"k0 *" -> 1., "k1 *" -> .1, "k2 *" -> .5, "Q *" -> 1.}];
SMTAddEssentialBoundary["X" == -0.5 || "X" == 0.5 || "Y" == -0.5 || "Y" == 0.5 || "Z" == 0. &, 1 -> 0];
SMTAddMesh[Hexahedron[{{-0.5, -0.5, 0}, {0.5, -0.5, 0}, {0.5, 0.5, 0}, {-0.5, 0.5, 0},
    {-0.5, -0.5, 1}, {0.5, -0.5, 1}, {0.5, 0.5, 1}, {-0.5, 0.5, 1}}], "cube", "H1", {20, 20, 20}];
SMTAnalysis[];
SMTNextStep["Δλ" → 1];
While[SMTConvergence[10^-12, 10], SMTNewtonIteration[];];
SMTSimulationReport[{"NoEvaluations"}, {"EvaluationTime"}];
```

No. of nodes	9261	Direct analysis:	{}
No. of elements	8000	Total linear solver time (s)	0.358
No. of equations	7220	Total linear solver time (%)	16.411
Number of threads used/max	8/8	Total K&R time (s)	0.038
Data memory (KBytes)	2825	Total K&R time (%)	1.742
Tangent matrix (KBytes)	1403	Average time/iteration (s)	0.535
Solver memory (KBytes)	29679	Average linear solver time (s)	0.0895
Total driver (KBytes)	33907	General timing:	
MMA kernel memory (KBytes)	193755	Input data phase (s)	1.2939
MMA front end memory(KBytes)	427860	Input data phase (%)	59.312
Total memory (KBytes)	655522	Total visualization time (s)	0
Solver type	Pardiso	Total visualization time (%)	0
Matrix type	1	Total absolute time (s)	2.1814
No. of steps	1	Total driver time (s)	2.14
No. of steps back	0	CPU Mathematica time (s)	0.875
Step efficiency (%)	100.	CPU Mathematica time (%)	40.111
Total no. of iterations	4	Profile time	0.035
Average iterations/step	4.	USER-IData	
Terminal BC multiplier (λ)	1.	Total NoEvaluations	31892
Terminal time (t)	0.	Average NoEvaluations/element	0.99663
Terminal parameter (γ)	0.	USER-RData	
		Total EvaluationTime (s)	0.169
		Total EvaluationTime (%)	7.7472
		Average EvaluationTime/element (s)	5.2813×10^{-6}

Linear Algebra

Contents

MKL Direct Sparse Solver

The Intel MKL PARDISO package is a high-performance, robust, memory efficient, and easy to use software package for solving large sparse linear systems of equations on shared memory multiprocessors. The parallel pivoting methods allow complete supernode pivoting to compromise numerical stability and scalability during the factorization process. For sufficiently large problem sizes, numerical experiments demonstrate that the scalability of the parallel algorithm is nearly independent of the shared-memory multiprocessing architecture.

Documentation is available at `https://software.intel.com/en-us/intel-mkl/documentation`, `PARDISO`.

MKL Direct Sparse Solver options

The use of the PARDISO solver is initiated and controlled by the arguments of the `SMTSetSolver` command.

option	default value	description
"MatrixType"	Automatic	1 - real and structurally symmetric matrix (partial pivoting) 2 - real and symmetric positive defini matrix -2 - real and symmetric indefinite matrix (SMTIData["NegativePivots"] available) 3 - complex and structurally symmetric matrix 4 - complex and Hermitian positive definite matrix -4 - complex and Hermitian indefinite matrix 6 - complex and symmetric matrix 11 - real and unsymmetrical matrix (full pivoting) 13 - complex and unsymmetrical matrix The default matrix type depends on the element specification `SMSSymmetricTangent` and node identifications (Node Identification) -L and -AL.
"IntegerParameters"	Automatic	$\{\{index_1, value_1\}, \{index_2, value_2\}, \ldots\}$ where $index_i$ is an index of the parameter accordingly to the PARDISO documentation and $value_i$ is the choosen value of the parameter

SMTSetSolver options related to the INTEL MKL PARDISO direct solver.

Some of the more common parameters are given below. For a complete list of the parameters refer to the manual available at https://software.intel.com/en-us/intel-mkl/documentation.

{60, *pv* } - Out-of-core

Parameter 60 controls what version of PARDISO - out-of-core version (pv=2) or in-core version (pv=0) - is used. The current OOC version does not use threading.

{8,*nstep*} -Max number of iterative refinement steps

Maximum number of iterative refinement steps that the solver will perform. Iterative refinement will stop if a satisfactory level of accuracy of the solution in terms of backward error has been achieved. The solver will not perform more than the absolute value of nstep steps of iterative refinement and will stop the process if a satisfactory level of accuracy of the solution in terms of backward error has been achieved. The default value for nstep is 2.

{10,*pv*} -Small pivot threshold

On entry, pv instructs PARDISO how to handle small pivots or zero pivots for unsymmetrical matrices. The magnitude of the potential pivot is tested against a constant threshold of: $\epsilon = 10^{-pv}$ Small pivots are therefore perturbed with $\epsilon = 10^{-pv}$ The default value of pv is 13 and therefore $\epsilon = 10^{-13}$.

MKL Iterative Sparse Solver

Intel MKL supports iterative sparse solvers (ISS) based on the reverse communication interface (RCI), referred to here as the RCI ISS interface. The RCI ISS interface implements a group of user-callable routines that are used in the step-by-step solving process of a symmetric positive definite system (RCI conjugate gradient solver, or RCI CG), and of a non-symmetric indefinite (non-degenerate) system (RCI flexible generalized minimal residual solver, or RCI FGMRES) of linear algebraic equations.

Documentation is available at MKL Iterative Sparse Solver and Preconditioners based on Incomplete LU Factorization Technique .

MKL Iterative Sparse Solver options

The use of the iterative solver is initiated and controlled by the arguments of the SMTSetSolver command.

option	default value	description
"MatrixType"	Automatic	1 - real and structurally symmetric matrix (partial pivoting) 2 - real and symmetric positive definite matrix -2 - real and symmetric indefinite matrix (SMTIData["NegativePivots"] available) 3 - complex and structurally symmetric matrix 4 - complex and Hermitian positive definite matrix -4 - complex and Hermitian indefinite matrix 6 - complex and symmetric matrix 11 - real and unsymmetrical matrix (full pivoting) 13 - complex and unsymmetrical matrix The default matrix type depends on the element specification `SMSSymmetricTangent` and node identifications (Node Identification) -L and -AL.
"IterativeSolverType"	2	1 - FGMRES 2 - CG
"Preconditioner"	1	0 - no preconditioner 1 - ILUO 2 - ILUT 3 - Jacobi
"MaxNoIterations"	10^4	maximum number of iterations
"MaxFillInRatio"	2	maximum nonzero terms in a row of the incomplitly factorized matrix should be "MaxFillInRatio"*a*verage number of nonzero terms in a row* (ILUT only option)
"MaxFillInElements"	0	maximum nonzero terms in a row of the incompletely factorized matrix (when given the "MaxFillInRatio" option is ignored) (ILUT only option)
"ResidualErrorTolerance"	10^{-6}	tolerance
"VectorNormTolerance"	10^{-12}	tolerance (FGMRES only option)
"PreconditionerTol"	10^{-6}	tolerance (ILUT only option)
"IntegerParameters"	Automatic	$\{\{index_1,value_1\},\{index_2,value_2\},\ldots\}$ where $index_i$ is an index of the parameter accordingly to the Intel MKL documentation and $value_i$ is the choosen value of the parameter
"RealParameters"	Automatic	$\{\{index_1,value_1\},\{index_2,value_2\},\ldots\}$ where $index_i$ is an index of the parameter accordingly to the Intel MKL documentation and $value_i$ is the choosen value of the parameter

`SMTSetSolver` options related to the ITERATIVE INTEL MKL solver.

Some of the more common parameters are given below. For a complete list of the parameters refer to the manual available at `https://software.intel.com/en-us/intel-mkl/documentation`

Integer parameters

- **{8,val} - Residual stopping test**

  If the value val is not equal to 0, the stopping test for the residual is performed during iterations.

  {8,0} defines that the routine does not perform residual stopping test.

  The default value val is 0.

- **{9,val} - User-defined residual error stopping test**

  If the value val is not equal to 0, the stopping test is performed during iterations. This test checks if the current residual error is smaller than the residual error tolerance, which is defined as an option “ResidualErrorTolerance” in SMTSetSolver. If option “ResidualErrorTolerance” is not explicitly defined by the user, its default value 10^{-6} is used.

  {9,0} defines that the routine does not perform the user-defined stopping test.

  The default value is 1.

- **{14,val} - Number of non-restarted FGMRES iterations (only FGMRES)**

Value val specifies the number of non-restarted FGMRES iterations. If this value is higher than number of max iterations (option "MaxNoIterations" in SMTSetSolver) then the non-restarted version of FGMRES is used. The default value is min(MaxNoIterations,150).

Warning: option max number of iterations and option of preconditioner that could be defined with **{4,val}** and **{10,val}** are overwritten by the chosen or default values in SMTSetSolver options "MaxNoIterations" and "Preconditioner".

Real parameters

- **{0,val} - Relative tolerance**
  Value val specifies the relative tolerance. The default value is 10^{-6}.
- **{1,val} - Absolute tolerance**
  Value val specifies the absolute tolerance. The default value is 0.0.

Number of iterations, performed in last call of SMTNewtonIteration[] is stored as integer type environment variable with keyword "Solver6" (the value can be obtained with SMTIData["Solver6"]).

Iterative methods solve systems of linear equations A x = b, where $x \in \mathbb{R}^n$ is an unknown vector, $b \in \mathbb{R}^n$ is a known vector and A is a known square matrix: $A \in \mathbb{R}^{nxn}$ (with some further restrictions, as described below). By these methods, the system of linear equations is solved by repetitive vector - matrix multiplications. Therefore, iterative methods can be less time and space consuming in comparison to direct solvers, specially when dealing with large sparse, well-posed systems.

AceFEM allows use of two different iterative Intel MKL solvers:

- Conjugate Gradient (CG)
- Flexible Generalized Minimal Residual Solver (FGMRES)

Conjugate Gradient (CG) method

CG method starts from the quadratic form $f(x) = \frac{1}{2} x^T A x - b^T x + c$. If A is symmetric and positive definite, f(x) is minimized by the solution to A x = b. The minimum of f(x) can be found by setting gradient of f'(x) equal to zero: f'(x) =0. In iterative method one starts with an arbitrary initial solution approximation x_0 (usually x_0={0,0,...,0}) and for estimation of next approximations x_1, x_2,...x_i,... iteratively takes step in the direction in which f decreases most quickly.

In theory, the solution is obtained after n steps, where n is the dimension of the system. However, in practice it can take more than n steps or may fail to converge when matrix A is ill-conditioned. At the other hand, when problem is well-posed, the solution can be obtained in much less than n steps.

Numerical complexity :

- time complexity is $O(m \sqrt{\kappa(A)})$, where m is number of nonzero terms in A and κ is condition number
- space complexity is $O(m)$, where m is number of nonzero terms in A

Note: CG is suitable only when matrix A is:

- symmetric,
- positive definite.

In case that matrix A is not symmetric and positive definite, the CG method may fail to compute solution or compute the wrong solution.

Flexible Generalized Minimal Residual Solver (FGMRES)

FGMRES is a generalization of GMRES method in way that it performs better in case of ill-conditioned problems. In contrast to CG method, FGMRES can solve also non-symmetric indefinite (non-degenerate) system of linear algebraic equations.

Preconditioners

The main drawback of iterative methods compared to direct methods is that they are less predictable. Convergence of iterative methods depends greatly on the values of matrix A. Therefore, specially when dealing with ill-conditioned matrix, the iterative methods may require many steps to calculate the solution or may not converge at all.

To improve the convergence rate of iterative methods, the preconditioners can be used. Preconditioning transforms a problem A x = b into another system with more favorable properties for iterative solution. A preconditioner is a matrix (M) that effects such a transformation: M^{-1} A x = M^{-1} b.

AceFEM allows use of two classes of preconditioners:

- Incomplete LU factorization

  The first possibility is to obtain a preconditioner as the incomplete factorization of matrix A. In this case, matrix A is factorized into a product of lower and upper triangular matrices. As well known, this factorization leads to some fill-in in the resulting matrix in comparison with the original matrix. Regarding the amount of fill-in two possible preconditioners are:

  - ILU0
    - no-fill ILU factorization: it preserves the structure of matrix A.
    - less efficient for ill-conditioned systems
  - ILUT
    - it preserves some fill-in in comparison to matrix A. It calculates each element of the preconditioner and saves it only if it satisfies two conditions:
      1.) its value is greater than the product of the given tolerance (option "PreconditionerTol" in command SMTSetSolver) and matrix row norm
      2.) its value is in the given bandwidth of the resulting preconditioner matrix (option “MaxFillInElements” in command SMTSetSolver)
    - more efficient for ill-conditioned systems

  ILU preconditioners may produce non-symmetric resulting matrix even if the original matrix is symmetric. Therefore, when the analysis is performed with CG method and incomplete LU factorization is chosen as preconditioner, the sequence of multiplication on L and L^T is performed (instead of L and U) to assure symmetric preconditioner. Note: such kind of preconditioner improve system only for diagonal dominant matrices.

  Warning! When ILU preconditioners are chosen with CG method, the matrix should be defined as unsymmetric (Option “MatrixType”→11 in SMTSetSolver). Otherwise, the matrix A will be treated as upper triangular when calculating the preconditioner and obviously such preconditioner most likely will not improve the convergence rate of the actual system.

- diagonal scaling or Jacobi preconditioning

  Diagonal scaling uses preconditioner M=diag(A). It is one of the simplest preconditioners. This preconditioner is efficient for diagonally dominant matrices A.

Note: Preconditioners can be applied to any matrix. However, in some cases they can increase number of iterations or even ruin the convergence. Influence of a preconditioner on the convergence of the iterative method can be often determined only in practice. Therefore, proper preconditioner should be chosen carefully.

Complete documentation is available at `Preconditioners based on Incomplete LU Factorization Technique` .

References

[1] Intel Math Kernel Library Developer Reference, MKL 11.3, Revision : 005

[2] J. R. Shewchuk, An introduction to the Conjugate Gradient Method Without the Agonizing Pain, School of Computer Science, Carnegie Mellon University, Pittsburgh (1994).

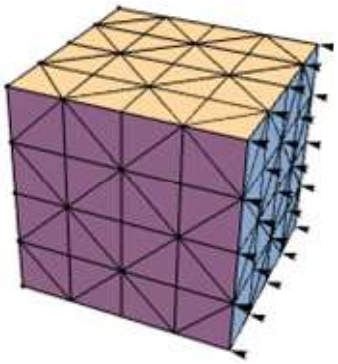
[3] H. R. Schwarz, The Method of Conjugate Gradients in Finite Element Applications, Journal of Applied Mathematics and Physics (ZAMP) 30 (1979)

[4] M. Benzi, Preconditioning Techniques for Large Linear System : A survey , Journal of Computational Physics 182 : 418 – 477 (2002) .

[5] K. Morikumi, L. Reichel, K. Hayami, FGMRES for linear discrete ill – posed problems, Applied Numerical Mathematics 75 : 175 – 187 (2014).

Example: Comparison of Direct solver and Iterative solvers with different preconditioners

To demonstrate calculation times and number of iterations of different iterative methods and preconditioners, a simple 3D linear-elastic example is chosen. A cube is clamped on one side and loaded with uniform axial force at the opposite side.



The analysis is performed in 10 steps, in each step 1 Newton Iteration is performed (since the problem is linear elastic). In each analysis step: calculation time and number of iterations that iterative solvers need to converge to the solution are put down. At the end of the analysis, average calculation time and average number of iterations per step are calculated. The results are compared for different sizes of the problem (the number of equations are changed through different mesh densities). Direct solver PARDISO and iterative solvers CG and FGMRES with different preconditioners are compared:

- PARDISO: direct solver PARDISO
- CG: iterative solver CG, no preconditioning
- CG-ILU0: iterative solver CG, ILU0 preconditioner
- CG-Jacobi: iterative solver CG, Jacobi preconditioner
- FGMRES: iterative solver FGMRES, no preconditioning
- FGMRES-ILU0: iterative solver FGMRES, ILU0 preconditioner
- FGMRES-ILUT: iterative solver FGMRES, ILUt preconditioner
- FGMRES-Jacobi: iterative solver FGMRES, Jacobi preconditioner

The mesh density is regulated through parameter nEl that defines number of elements along each axis.

Perform analysis for different solvers and different mesh densities:

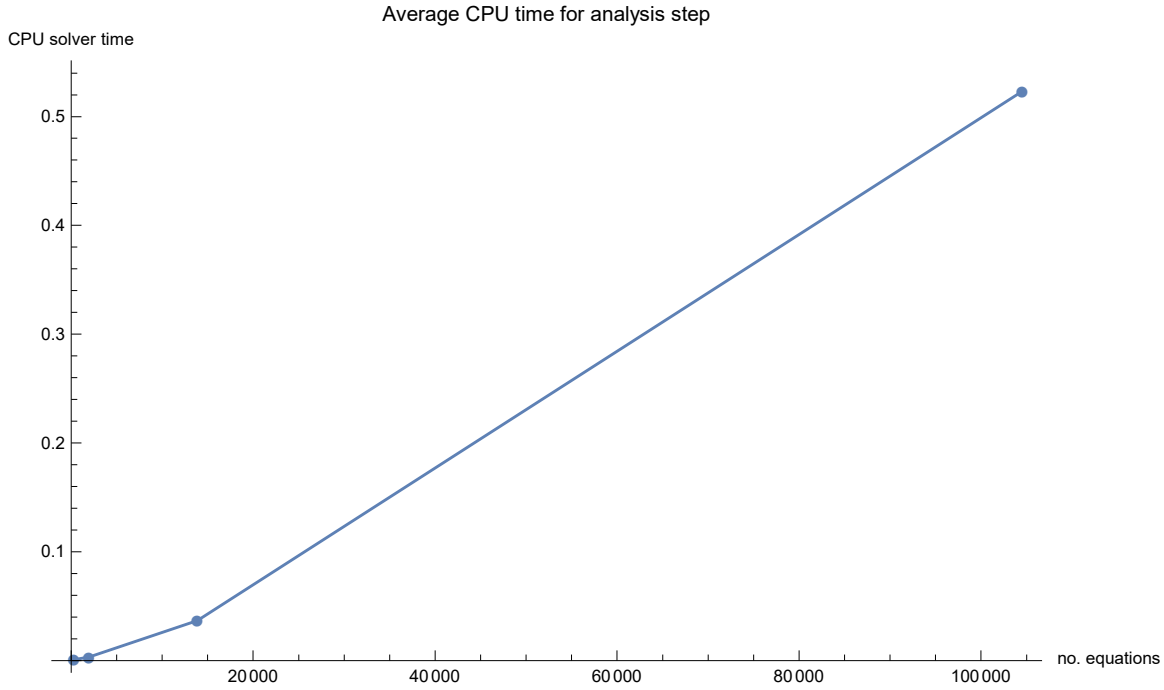
In[476]:=
```
time = {};
noEquations = {};
iterations = {};
```

In[479]:=
```
<< AceFEM`
```

In[480]:=
```
Do[
 SMTInputData["Console" → True];
 Vref = 50; H = 1; L = 1; λf = 1;
 SMTAddDomain["Ω", "OL:SED3O1DFLEO1Hooke", {"E *" → 1000, "ν *" -> 0.3}];
 SMTAddMesh[Hexahedron[{{0, 0, -H/2}, {0, L, -H/2}, {0, L, H/2}, {0, 0, H/2},
    {L, 0, -H/2}, {L, L, -H/2}, {L, L, H/2}, {L, 0, H/2}}], "Ω", "O1", {nEl, nEl, nEl}];
 SMTAddEssentialBoundary[Polygon[{{0, 0, -H/2}, {0, L, -H/2}, {0, L, H/2}, {0, 0, H/2}}],
  1 -> 0, 2 -> 0, 3 → 0];
 SMTAddNaturalBoundary[Polygon[{{L, 0, -H/2}, {L, L, -H/2}, {L, L, H/2}, {L, 0, H/2}}],
  1 → Polygon[{-Vref}]];
 SMTAnalysis[];
 AppendTo[noEquations, "No. of equations" /. SMTSimulationReport["Output" → {}]];
 nonZeroElements = "Tangent matrix (KBytes)" * 1000 / 8. /. SMTSimulationReport["Output" → {}];
 ilutMaxFill = Ceiling[nonZeroElements / noEquations * 2];
 (*max fill-in elements for ilut preconditioner is
   chosen as 2 times average nonZeroTerms in row of tangent matrix*)

 (*Solver is set with SMTSetSolver:*)
 Switch[1,
  1, solver = "CG";
  SMTSetSolver[6, "MatrixType" → 2, "IterativeSolverType" → 2, "Preconditioner" → 0];,
  2, solver = "CG-ILU0";
  SMTSetSolver[6, "MatrixType" → 11, "IterativeSolverType" → 2, "Preconditioner" → 1];,
  3, solver = "CG-Jacobi",
  SMTSetSolver[6, "MatrixType" → 2, "IterativeSolverType" → 2, "Preconditioner" → 3];,
  4, solver = "FGMRES", SMTSetSolver[6, "MatrixType" → 11,
    "IterativeSolverType" → 1, "Preconditioner" → 0];,
  5, solver = "FGMRES-ILU0", SMTSetSolver[6, "MatrixType" → 11,
    "IterativeSolverType" → 1, "Preconditioner" → 1];,
  6, solver = "FGMRES-ILUT", SMTSetSolver[6, "MatrixType" → 11, "IterativeSolverType" → 1,
    "Preconditioner" → 2, "MaxFillInElements" → ilutMaxFill];,
  7, solver = "FGMRES-Jacobi", SMTSetSolver[6, "MatrixType" → 11,
    "IterativeSolverType" → 1, "Preconditioner" → 3];,
  8, solver = "PARDISO", SMTSetSolver[];
 ];

 (*Ten steps of analysis are performed (notice: the problem is linear-elastic,
   therefore the result is calculated with only one Newton iteration:*)
 stepIterations = {};
 Do[
  SMTNextStep["Δλ" → 0.1];
  SMTNewtonIteration[];
  If[Not[solver == "PARDISO"], AppendTo[stepIterations, SMTIData["Solver6"]]];
  , {10}];

 AppendTo[time, "Average linear solver time (s)" /. SMTSimulationReport["Output" → {}]];
 If[Not[solver == "PARDISO"], AppendTo[iterations, Mean[stepIterations]]];

 , {nEl, {4, 8, 16, 32}}]
```

Diagram of CPU time:

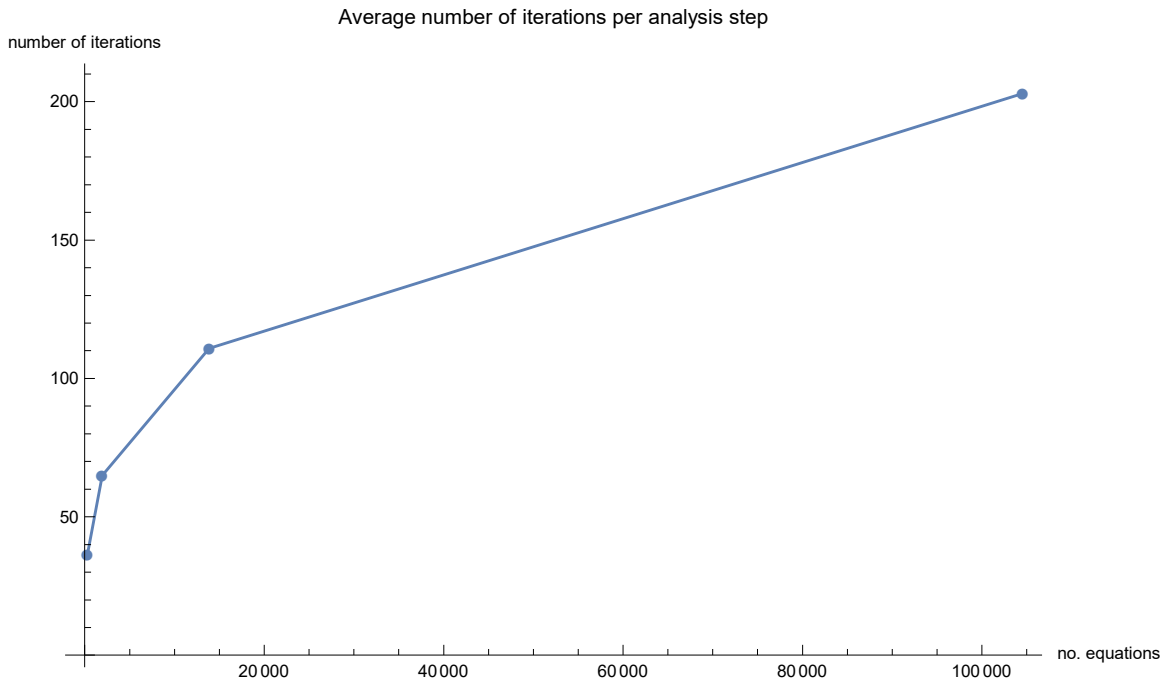
In[481]:=
```
ListLinePlot[
 Transpose[{noEquations, time}], PlotMarkers → Automatic, ImageSize → 600,
 PlotLegends → solver, AxesLabel -> {"no. equations", "CPU solver time"},
 PlotLabel → "Average CPU time for analysis step"]
```



CG

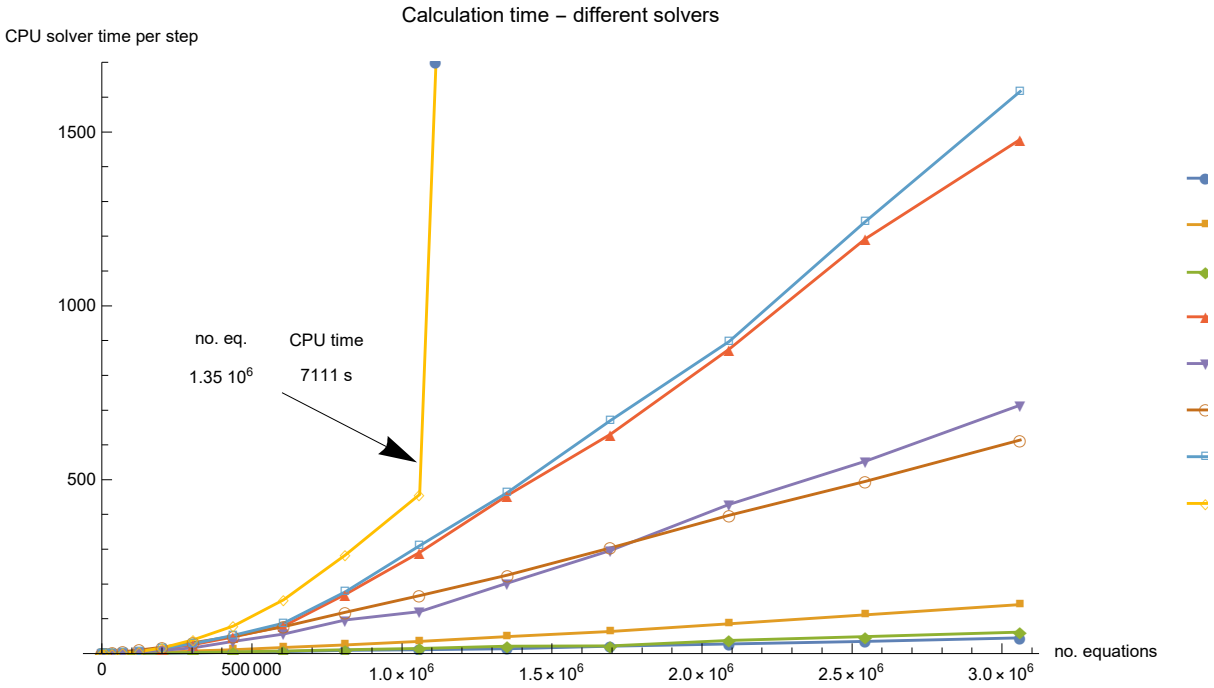
Average number of iterations per step:

In[482]:=
```
ListLinePlot[
 Transpose[{noEquations, iterations}], PlotMarkers → Automatic, ImageSize → 600,
 PlotLegends → solver, AxesLabel -> {"no. equations", "number of iterations"},
 PlotLabel → "Average number of iterations per analysis step"]
```



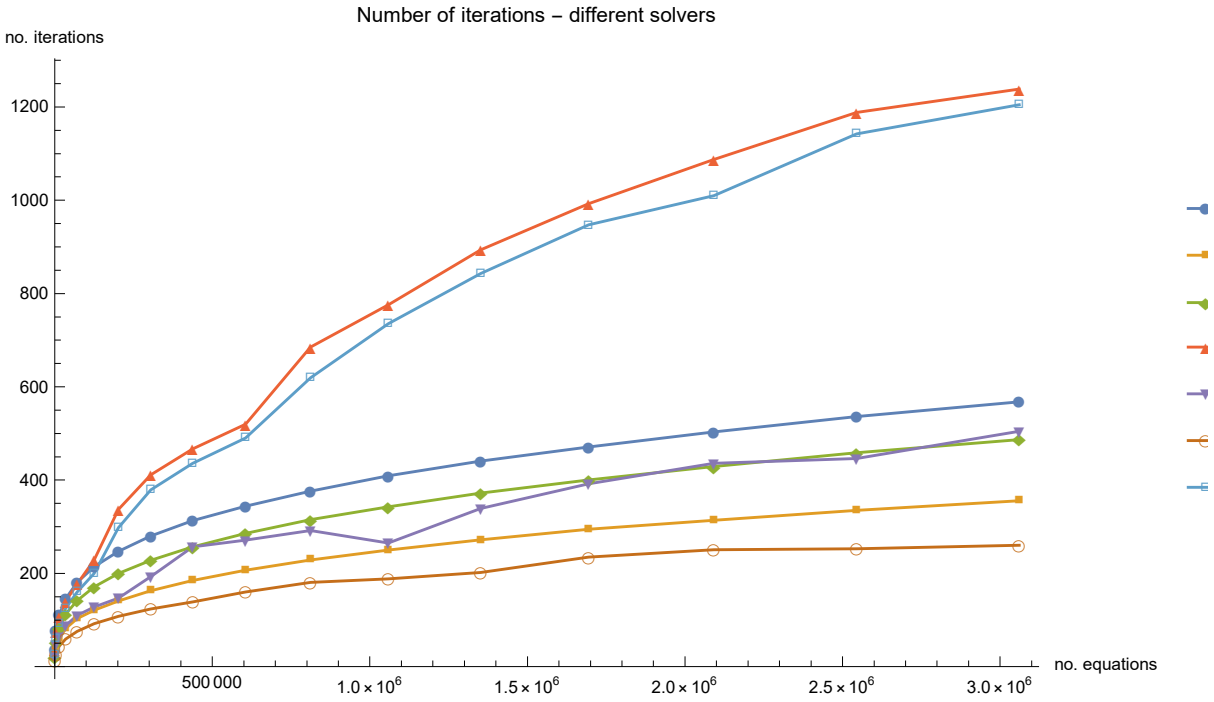
CG

Comparison of CPU time for different solvers and different number of equations. The analysis is performed on processor Intel Core(TM) i7-CPU Q720 1.60GHz, 16.0GB RAM, 4 cores, WIndows 7.



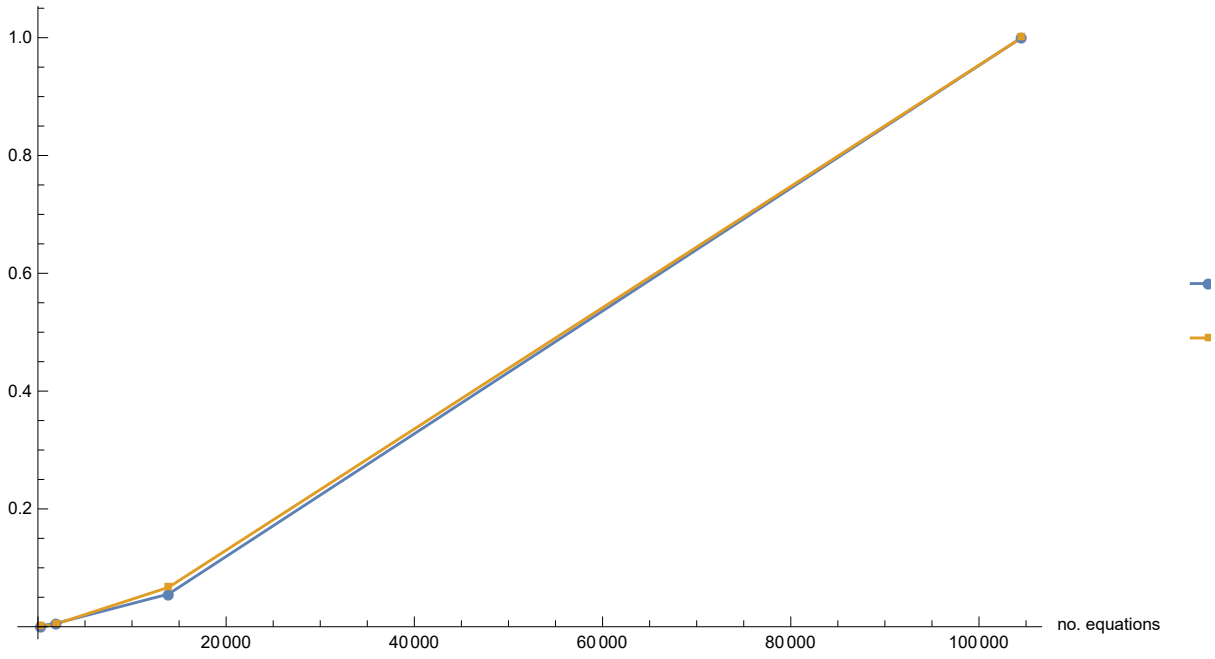
However, the difference of calculation times for different solvers and preconditioners can vary in comparison to the diagram shown, since the parallelization is not equally effective for different solvers.

Comparison of number of iterations for different solvers and different mesh densities:



Processor: Intel Core(TM) i7-CPU Q720 1.60GHz,16.0GB RAM,4 cores,WIndows 7					
no. equations	13 872	124 950	438 204	1 058 610	2 091 144
solver time [s]					
CG	0.0260999	0.5932	3.3609	11.1434	27.2
CG-ILU0	0.1469	2.1978	11.2086	35.3	85.8
CG-Jacobi	0.0263	0.8084	4.6273	15.069	37.5
FGMRES	0.1879	5.2628	47.5561	291.3	875.442
FGMRES-ILU0	0.2113	4.6512	34.6287	120.9	428.857
FGMRES-ILUT	0.6638	10.1945	49.7673	166.445	397.373
FGMRES-Jacobi	0.1997	5.6153	52.2	310.654	897.7
PARDISO	0.061	5.24	79.14	457.6	x
number of iterations					
CG	110.9	214.	313.	408.9	503
CG-ILU0	60.	121.	185.	250	314.
CG-Jacobi	82.8	170.9	256.9	342.9	429
FGMRES	107.8	229.	466.8	776	1087.
FGMRES-ILU0	65.	128.	257.	265	435.9
FGMRES-ILUT	44.	92.	139.	188.	251.
FGMRES-Jacobi	92.	199.6	436	735.	1010

Time complexity of CG is proportional to m $\sqrt{\kappa(A)}$), where m is number of nonzero terms in A and κ is condition number



Schur Complement

SMTSchurComplementOfEssentialBC

Task is to calculate Schur complement of the system

$$\begin{pmatrix} Kbb & Kbu \\ Kub & Kuu \end{pmatrix} \cdot \begin{pmatrix} \triangle b \\ \triangle u \end{pmatrix} = \begin{pmatrix} Rb \\ Ru \end{pmatrix}$$

where Ru=0 (converged state is assumed!) and b is set of DOF with prescribed essential boundary conditions (EBC). The result are

condensed matrix Kcc and vector Rc:

Kcc = Kbb – Kbu.Kuu^{-1}.Kub

Rc = Rb – Kbu.Kuu^{-1}.Ru = Rb

SMTSchurComplementOfEssentialBC is performed with the following procedure:

- essential boundary conditions are removed
- additional SMTNewtonIteration is called. During the step the solution of the system of linear equations suppressed and no update is performed. The result is full **K** and **R**.
  Additional SMTNewtonIteration will increment the SMTIData["Iteration"] counter!
- Schur complement is calculated using the MKL library functions

---

SMTSchurComplementOfEssentialBC[]
command returns data structure composed as follows:

{

{total number of DOF, dimension of Schur complement, number of eliminated DOF-s}

,Kcc

,Rc

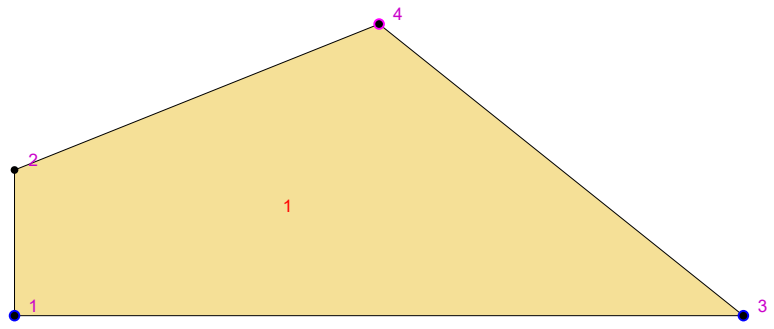
,a list indices of the nodes that belong to constrained DOF-s

,a list of positions of constrained DOFs within the nodes

}

In[146]:=
```
<< AceFEM`;
SMTInputData[];
L = 100; H = 20; nx = 1; ny = 1;
points = {{0, 0}, {L, 0}, {L / 2, 2 H}, {0, H}};
SMTAddDomain["A",
  {"ML:", "SE", "PE", "Q1", "DF", "LE", "Q1", "D", "Hooke"}, {"E *" → 21000, "ν *" → 0.2}];
SMTAddEssentialBoundary[Line[points[[{1, 2}]]], 1 → 0, 2 → 0];
SMTAddEssentialBoundary[Point[points[[3]]], 2 → 0.5];
SMTAddMesh[Polygon[points], "A", "Division" -> {nx, ny}];
SMTAnalysis[];
```

In[492]:=
```
SMTShowMesh["BoundaryConditions" → True, "Marks" → True]
```



- DOF-s with the number -1 have prescribed EBC and have to be eliminated

In[493]:=
```
SMTNodeData["X"]
```

```
{{0., 0.}, {0., 20.}, {100., 0.}, {50., 40.}}
```

In[494]:=
```
SMTNodeData["DOF"]
```

```
{{-1, -1}, {0, 1}, {-1, -1}, {2, -1}}
```

- calculate converged state

In[495]:=
```
SMTNextStep["λ" → 1];
SMTNewtonIteration[]
SMTNewtonIteration[]
```

```
0.111392
```

$$4.63879 \times 10^{-17}$$

In[498]:=
```
schur = SMTSchurComplementOfEssentialBC[];
```

- schur[[1]] - {total number of DOF, dimension of Schur complement, number of eliminated DOFs}

In[499]:=
```
schur[[1]]
```

```
{8, 5, 3}
```

- schur[[2]] - Kcc

In[500]:=
```
schur[[2]] // Chop // MatrixForm
```

$$\begin{pmatrix} 3774.69 & 867.123 & -3774.69 & 867.123 & -1734.25 \\ 867.123 & 6312.03 & -867.123 & 6312.03 & -12\,624.1 \\ -3774.69 & -867.123 & 3774.69 & -867.123 & 1734.25 \\ 867.123 & 6312.03 & -867.123 & 6312.03 & -12\,624.1 \\ -1734.25 & -12\,624.1 & 1734.25 & -12\,624.1 & 25\,248.1 \end{pmatrix}$$

- schur[[3]] - Rc

In[501]:=
```
schur[[3]]
```

```
{-867.123, -6312.03, 867.123, -6312.03, 12 624.1}
```

- schur[[4]] - a list indices of the nodes that belong to constrained DOF-s

In[502]:=
```
schur[[4]]
```

```
{1, 1, 3, 3, 4}
```

- schur[5]] - a list of positions of constrained DOFs within the nodes

In[503]:=
```
schur[[5]]
```

```
{1, 2, 1, 2, 2}
```

Eigenvalues and Eigenvectors

SMTPowerMethod

SMTPowerMethod[]
calculates the:

- greatest and the smallest eigenvalues of tangent matrix K or

- the smallest (in absolute value) eigenvalue of tangent matrix K or

- the eigenvalue nearest to some given value

- the smallest (in absolute value) eigenvalue of the generalized eigenvalue problem (initial stability problem: $K_0 + \lambda K_\sigma$)

- the smallest (in absolute value) eigenvalue of the generalized eigenvalue problem (linearized buckling analysis: $K=K_1+\gamma(K_2-K_1)$; $\lambda=\lambda_1+\gamma(\lambda_2-\lambda_1)$)

and corresponding eigenvectors.

Calculation is performed with power method and its variations: shifted power method, inverse power method and inverse shifted power method.

option	default value	description
"Method"	3	1 - calculation of largest and smallest eigenvalue: $K\,\Psi = \lambda\,\Psi$ 2 - calculation of largest (in absolute value) eigenvalue: $K\,\Psi = \lambda\,\Psi$ 3 - calculation of smallest (in absolute value) eigenvalue : $K\,\Psi = \lambda\,\Psi$ or eigenvalue nearest to some arbitrary value (option "Shift"→μ) $(K-\mu\,I)\,\Psi = (\lambda-\mu)\,\Psi$ 4 - calculation of the smallest (in absolute value) eigenvalue of the generalized eigenvalue problem (initial stability problem): $K_0\,\Psi = \lambda\,K_\sigma\,\Psi$ or eigenvalue nearest to some arbitrary value μ (option "Shift"→μ): $(K_0-\mu\,K_\sigma)\,\Psi = (\lambda-\mu)\,K_\sigma\,\Psi$ 5 - calculation of the smallest (in absolute value) eigenvalue of the generalized eigenvalue problem (linearized buckling analysis: $K=K_1+\gamma\,(K_2-K_1)$; $\lambda=\lambda_1+\gamma\,(\lambda_2-\lambda_1)$; $K_1\,\Psi = \gamma\,(K_2-K_1)\,\Psi$ or eigenvalue nearest to some arbitrary value (option "Shift"→μ)
"MaxNoIterations"	2000	Maximum number of iterations allowed.
"Tolerance"	10^(-6)	Tolerance for the error of calculated eigenvalue.
"ErrorMethod"	2	1 - error = $\frac{\lvert\lambda^{(i)}-\lambda^{(i-1)}\rvert}{\lvert\lambda^{(i)}\rvert}$ 2 - error = $\lVert\text{residual}\rVert_2/\lvert\lambda^{(i)}\rvert$, where residual is: $K\,\Psi^{(i)} - \lambda^{(i)}\,\Psi^{(i)}$ (standard eigenvalue problem) or $(K-\mu\,I)\,\Psi^{(i)} - (\lambda^{(i)}-\mu\,)\,\Psi^{(i)}$ (shifted eigenvalue problem, μ is shift) or $K_0\,\Psi^{(i)} - \lambda^{(i)}\,K_\sigma\,\Psi^{(i)}$ (generalized eigenvalue problem) or $(K_0-\mu\,K_\sigma)\,\Psi^{(i)} - (\lambda^{(i)}-\mu\,)\,K_\sigma\,\Psi^{(i)}$ (shifted generalized eigenvalue problem, μ is shift)
"InitialVectorEstimate"	1	1 - each term of the initial approximation of eigenvector is generated with pseudo random generator 2 - initial approximation of eigenvector is taken from the current value of the nodal d.o.f (Node Data "at"). 3 - initial approximation of eigenvector is taken from the current value of the nodal d.o.f (Node Data "at") so that the maximal term in node data "at" is replaced with 1 and all the other terms are taken as zero terms.
"Shift"	0	Shift of the method. The eigenvalue nearest to value of option "shift" is sought. Only possible with "Method" 3 and 4.
"Eigenvectors"	False	Defines if eigenvectors are printed out or not.
"Report"	False	Defines if some additional data of the analysis is printed out at the end of the list with the results. In case option "True" is chosen, the last list of the results is: {True/False,{*iterations*},{*relativeErrors*}}. First term evaluates to True if the result has converged, otherwise this term is False.
"Abort"	True	Specifies if the iterative process is aborted in case that the result has not converged (in case that the process is not aborted, first term of "Report" evaluates to False when the result has not converged).
"λ1"	Null	Natural and essential boundary conditions multiplier at lower load level when tangent matrix (K_1) was stored. Only in case of method 5 (linearized buckling analysis).

Options of the eigensystem calculation with Power method.

- Method 1: calculation of the largest and smallest eigenvalue

  The power method is performed in two steps:

  - step 1: The dominant eigenvalue $\lambda_{dominant}$ of standard eigenvalue $K\,\Psi = \lambda\,\Psi$ problem is calculated.

    Iterations:

    $X^{(k)} = K\,\Psi^{(k-1)}$

    $\Psi^{(k)} = \frac{X^{(k)}}{\lVert X^{(k)}\rVert}$

    $\lambda^{(k)} = \Psi^{(k)}.K.\Psi^{(k)}$

  - step 2: To obtain the opposite between the largest or the smallest eigenvalue, the one with smaller magnitude, the shifted power method is applied to calculate the dominant eigenvalue to matriΨ $K - \mu\,I$, where $\mu = \lambda_{dominant}$ is the dominant eigenvalue, obtained with power method applied in step 1 to matrix K. I is identity matrix: $(K-\mu\,I)\,x = (\lambda-\mu)\,x$.

Iterations:

$X^{(k)} = (K - \mu I)\, \Psi^{(k-1)}$

$\Psi^{(k)} = \frac{X^{(k)}}{\|X^{(k)}\|}$

$\lambda^{(k)} - \mu = \Psi^{(k)}.(K - \mu I).\Psi^{(k)}$

- Method 2: calculation of the largest (in absolute value) eigenvalue

  The power method is applied to calculate the dominant eigenvalue $\lambda_{dominant}$ of tangent matrix K (step 1 in method 1).

- Method 3: calculation of the smallest (in absolute value) eigenvalue or eigenvalue nearest to some arbitrary value

  The inverse power method is applied to calculate eigenvalue nearest to 0 in standard eigensystem $K\,\Psi = \lambda\,\Psi$.

  Iterations:

  $K.X^{(k)} = \Psi^{(k-1)}$

  $\Psi^{(k)} = \frac{X^{(k)}}{\|X^{(k)}\|}$

  $\lambda^{(k)} = \Psi^{(k)}.K.\Psi^{(k)}$

  or nearest to a chosen shift value μ (option "Shift") in shifted standard eigensystem $(K - \mu I)\,\Psi = (\lambda - \mu)\,\Psi$.

  Iterations:

  $(K - \mu I).X^{(k)} = \Psi^{(k-1)}$

  $\Psi^{(k)} = \frac{X^{(k)}}{\|X^{(k)}\|}$

  $\lambda^{(k)} - \mu = \Psi^{(k)}.(K - \mu I).\Psi^{(k)}$

- Method 4: calculation of the smallest (in absolute value) eigenvalue of the generalized eigenvalue problem

  The inverse power method is applied to calculate eigenvalue nearest to 0 for generalized eigenvalue problem $K_0\,\Psi = \lambda\,K_\sigma\,\Psi$.

  - K_0 should be stored by user during the analysis with call of SMTStoreTangentMatrix[] command (when the global tangent matrix, currently formed is in fact matrix K_0)
  - $K_0 + \lambda * K_\sigma$ is current tangent matrix

    Iterations:

    $K_0.X^{(k)} = K_\sigma\,\Psi^{(k-1)}$

    $\Psi^{(k)} = \frac{X^{(k)}}{\|X^{(k)}\|}$

    $\lambda^{(k)} = \frac{\Psi^{(k)}.K_0.\Psi^{(k)}}{\Psi^{(k)}.K_\sigma.\Psi^{(k)}}$

  or nearest to a chosen shift value μ (option "Shift") for generalized eigenvalue problem $(K_0 - \mu K_\sigma)\Psi = (\lambda - \mu)\,K_\sigma\Psi$.

  Iterations:

  $(K_0 - \mu K_\sigma).X^{(k)} = K_\sigma\,\Psi^{(k-1)}$

  $\Psi^{(k)} = \frac{X^{(k)}}{\|X^{(k)}\|}$

  $\lambda^{(k)} - \mu = \frac{\Psi^{(k)}.(K_0 - \mu K_\sigma).\Psi^{(k)}}{\Psi^{(k)}.K_\sigma.\Psi^{(k)}}$

- Method 5: calculation of the smallest (in absolute value) eigenvalue of the generalized eigenvalue problem

  The inverse power method is applied to calculate eigenvalue nearest to 0 for generalized eigenvalue problem $K_1\,\Psi = \gamma\,(K_1 - K_2)\,\Psi$, $\lambda = \lambda_1 + \gamma(\lambda_2 - \lambda_1)$, where K_1 and K_2 are tangent matrices at two different load levels at which boundary conditions multipliers are λ_1 and λ_2:

  - K_2 is tangent matrix at current load level λ_2
  - K_1 is tangent matrix at (arbitrary) lower load level (λ_1) and should be stored by user during the analysis, with call of SMTStore-TangentMatrix[] command. λ_1 should be given by user as option "λ1" in SMTPowerMethod command.

    Iterations:

    $K_1.X^{(k)} = (K_1 - K_2)\,\Psi^{(k-1)}$

    $\Psi^{(k)} = \frac{X^{(k)}}{\|X^{(k)}\|}$

    $\gamma^{(k)} = \frac{\Psi^{(k)}.K_1.\Psi^{(k)}}{\Psi^{(k)}.(K_1 - K_2).\Psi^{(k)}}$

    $\lambda^{(k)} = \lambda_1 + \gamma^{(k)}(\lambda_2 - \lambda_1)$

Caution: if the initial estimate of eigenvector is an eigenvector of matrix, the Power Method will converge to corresponding eigenvalue.

SMTStoreTangentMatrix[]

Stores current tangent matrix.

SMTFreeStoredTangentMatrix[]
deallocates memory that was used to store tangent matrix with SMTStoreTangentMatrix[].

SMTFEAST

SMTFEAST[*emin*,*emax*]
calculates eigenvalues and eigenvectors on interval [*emin*,*emax*] with MKL FEAST Algorithm

Calculation of eigensystem with SMTFEAST is done with Extended Eigensolver Routines from INTEL MKL. It is based on the FEAST Algorithm. The details of FEAST Algorithm can be found in https://software.intel.com/en-us/intel-mkl/documentation.

option	default value	description
"Method"	1	1 - standard eigenvalue problem with tangent matrix is calculated: $K\,\Psi = \lambda\,\Psi$ 2 - initial stability problem - generalized eigenvalue problem is calculated: $K_0\,\Psi = \lambda\,K_\sigma\,\Psi$ 3 - linearized buckling analysis - generalized eigenvalue problem is calculated: $K_1\,\Psi = \gamma\,(K_2 - K_1)\,\Psi$; $\lambda = \lambda_1 + \gamma\,(\lambda_2 - \lambda_1)$
"MaxNoEigenvalue-sOnInterval"	Automatic	Initial guess for subspace dimension to be used. Should be at least equal to the total number of eigenvalues located in the search interval. As a rule of the thumb, if number of searched eigenvalues is smaller than 1000, it is recommended to use subspace dimension = 1.5*total number of eigenvalues located in the search interval. However, subspace dimension cannot exceed number of equations. If "MaxNoEigenvaluesOnInterval" is not given by user, the Automatic default value is used: min (20, *n*), where *n* is number of equations.
"NoContourPoints"	8	The number of contour points used in contour integration (Gauss-Legendre quadrature). Must be one of: {3,4,5,6,8,10,12,16,20,24,32,40,48}
"ErrorTrace"	6	error trace double precision stopping criteria ϵ ($\epsilon = 10^{-\text{errorTrace}}$). Stopping test: - standard eigenvalue problem ($A\Psi=\lambda\Psi$): $\max_{i=1:mode} \frac{\|A\Psi_i - \lambda_i \Psi_i\|_1}{\max(\|e_{min}\|, \|e_{max}\|)\,\|\Psi_i\|_1} < \epsilon$ - generalized eigenvalue problem ($A\Psi=\lambda B\Psi$): $\max_{i=1:mode} \frac{\|A\Psi_i - \lambda_i B\,\Psi_i\|_1}{\max(\|e_{min}\|, \|e_{max}\|)\,\|B\,\Psi_i\|_1} < \epsilon$
"MaxNoIterations"	200	Maximum number of loops allowed in FEAST algorithm
"Eigenvectors"	False	Defines if eigenvectors are printed out or not.
"Report"	False	Defines if some additional data of the analysis is printed out at the end of the list with the results. In case option "True" is chosen, the last list of the results is: {True/False,*iterations*,*info*,{*residuals*}}. First term evaluates to True if the result has converged, otherwise this term is False. Term *info* is an integer variable that defines if errors or warnings were encountered during a run of FEAST. If *info* = 0, no warnings or errors were encountered. For other return values see mkl documentation.
"Abort"	True	Specifies if the iterative process is aborted in case that the result has not converged (in case that the process is not aborted, first term of "Report" evaluates to False when the result has not converged).
"λ1"	Null	Natural and essential boundary conditions multiplier at lower load level when tangent matrix (K_1) was stored. Only in case of method 5 (linearized buckling analysis).

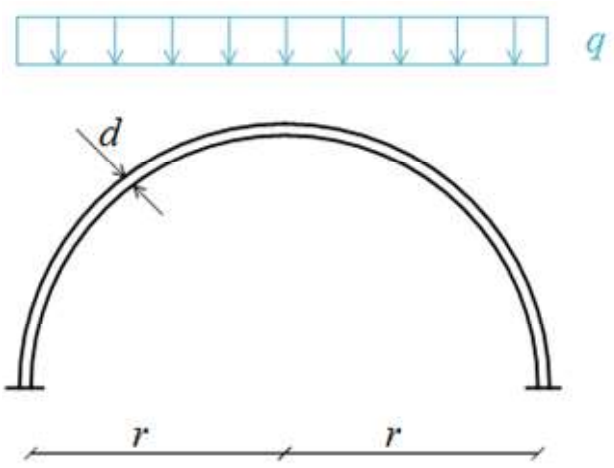
Options of SMTFEAST function.

Caution: In case of Method 2 (initial stability problem), K_0 should be stored by user during the analysis with call of SMTStoreTangentMatrix[] command. It is presumed that current tangent matrix is $K_0 + \lambda * K_\sigma$.

Note: In case of Method 3 (linearized buckling analysis) FEAST algorithm solves generalized eigenvalue problem $K_1 \Psi = \gamma (K_1\text{-}K_2)\Psi$, where K_2 is tangent matrix at current load level (λ_2) and K_1 is tangent matrix at (arbitrary) lower load level (λ_1) and should be stored by user during the analysis, with call of SMTStoreTangentMatrix[] command. λ_1 should be given by user as option "λ1" in SMTFEAST command.

λ is then calculated as $\lambda = \lambda_1 + \gamma (\lambda_2\text{-}\lambda_1)$. The result that AceFEM prints as output is λ (and not γ)! Search interval [emin,emax] should be given for λ (and not for γ).

Example: Arch bending - eigenvalues of tangent matrix

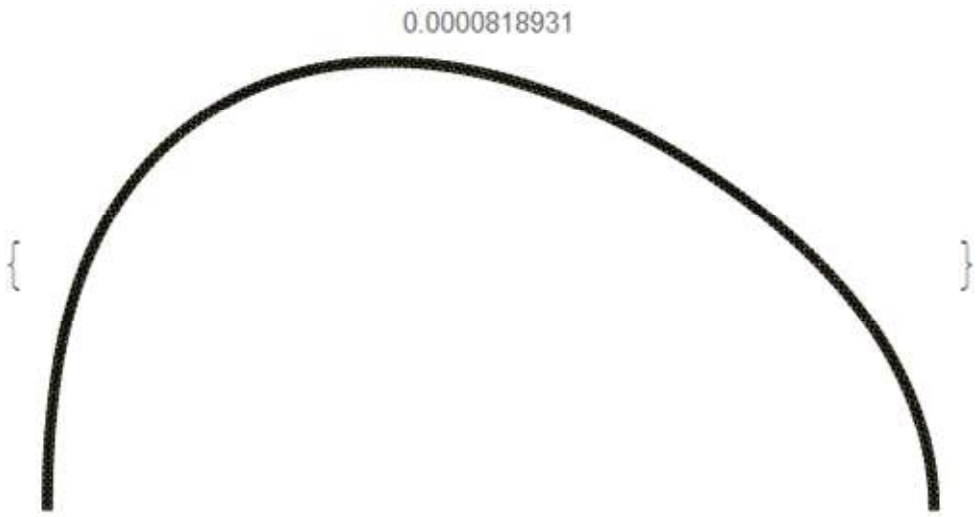



```
In[15]:= << AceFEM`;
         r = 200.; d = 4.; q = 0.03;
         SMTInputData[];
         SMTAddDomain["arch", {"ML:", "SE", "PE", "T2", "DF", "HY", "T2", "D", {{"NeoHooke", "WA"}}},
           {"E *" → 21000., "ν *" -> .3}];
         SMTAddMesh[ToElementMesh[
            Polygon[Join[Table[{(r + d) * Cos[ϕ], (r + d) * Sin[ϕ]}, {ϕ, 0, π, π / 100}],
              Table[{r * Cos[ϕ], r * Sin[ϕ]}, {ϕ, π, 0, -π / 100}]]], "MaxCellMeasure" → 5], "arch"];
         SMTAddEssentialBoundary[Line[{{r, 0}, {r + d, 0}}], 1 → 0, 2 → 0];
         SMTAddEssentialBoundary[Line[{{-r, 0}, {-r - d, 0}}], 1 → 0, 2 → 0];
         SMTAddNaturalBoundary[Line[Table[{(r + d) * Cos[ϕ], (r + d) * Sin[ϕ]}, {ϕ, 0, π, π / 100}]],
           2 → Line[Function[{X, Y, tx, ty}, -q Abs[tx]]]];
         SMTAnalysis[];

In[24]:= Do[
           SMTNextStep["Δλ" → 1];
           While[SMTConvergence[10^-8, 10], SMTNewtonIteration[]];
           , {i, 1, 4}];
```

Power method

This calculates the smallest (in absolute value) eigenvalue of tangent matrix and returns corresponding eigenvector.

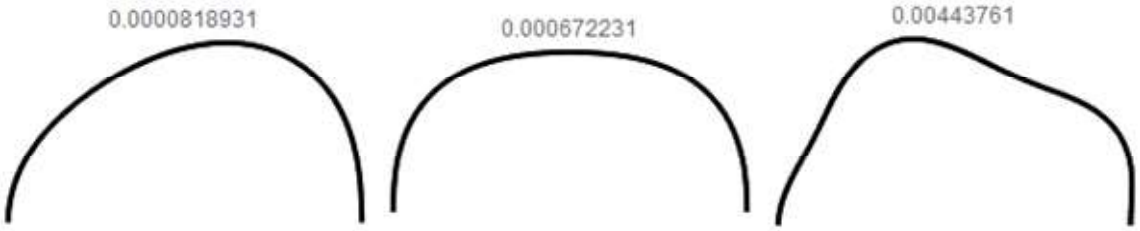
```
In[25]:= {λ, Ψ} = SMTPowerMethod["Method" → 3, "Eigenvectors" → True];

In[26]:= SMTShowEigenvectors[λ, Ψ]
```



FEAST algorithm

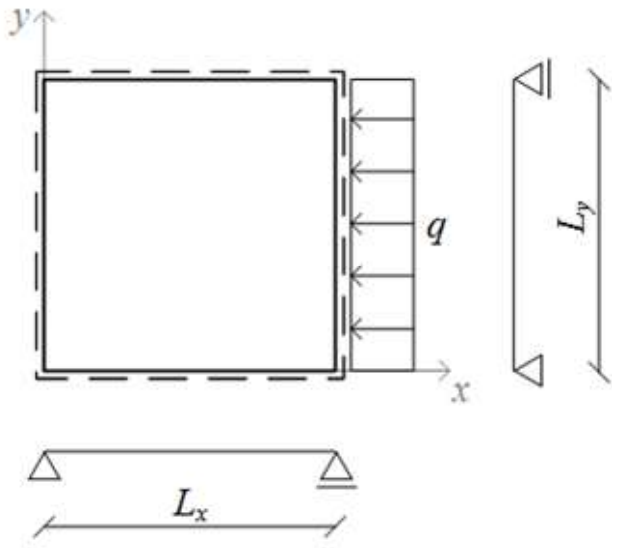
FEAST algorithm looks for eigenvalues of tangent matrix at interval [0,0.01] and returns corresponding eigenvectors.

```
In[27]:= {λ, Ψ} = SMTFEAST[0, 0.01, "Method" → 1,
            "MaxNoEigenvaluesOnInterval" → 15, "NoContourPoints" → 24, "Eigenvectors" → True];

In[28]:= Row[SMTShowEigenvectors[λ, Ψ, ImageSize → 200], "  "]
```



Example: Buckling of simply supported plate (initial stability problem)

In this example simply supported plate is loaded in one axial direction. Plate is modelled with linear elastic shell elements. The aim is to calculate critical buckling load. Initial stability problem method is used for this purpose. In this analysis one has to solve generalized eigenvalue problem: $(K_0 + \lambda_i K_\sigma^{ref}) \Psi_i = 0$, i = 1, ..., n , where K_σ^{ref} is geometric tangent matrix, obtained when reference load q^{ref} is applied, λ_i is eigenvalue and Ψ_i is corresponding eigenvector. The (absolutely) smallest λ_i is sought.



In[155]:=
```
<< AceFEM`
q = 20 000;
Lx = 10; Ly = 1;
SMTInputData[];
SMTAddDomain[{"plate", {"ML:", "SE", "MS", "S1", "DF", "LE", "S1P6", {"D", "Fi"}, "Hooke"},
    {"E *" → 210 000 000, "ν *" → 0.29, "t *" → 0.05}}];
SMTAddMesh[Polygon[{{0, 0, 0}, {Lx, 0, 0}, {Lx, Ly, 0}, {0, Ly, 0}}],
  "plate", "Division" -> {200, 20}];
SMTAddEssentialBoundary[
  {Line[{{0, 0, 0}, {Lx, 0, 0}}, "D"], 2 → 0, 3 → 0},
  {Line[{{0, 0, 0}, {Lx, 0, 0}}, "Fi"], 2 → 0},
  {Line[{{Lx, 0, 0}, {Lx, Ly, 0}}, "D"], 3 → 0},
  {Line[{{Lx, 0, 0}, {Lx, Ly, 0}}, "Fi"], 1 → 0},
  {Line[{{0, Ly, 0}, {Lx, Ly, 0}}, "D"], 3 → 0},
  {Line[{{0, Ly, 0}, {Lx, Ly, 0}}, "Fi"], 2 → 0},
  {Line[{{0, 0, 0}, {0, Ly, 0}}, "D"], 1 → 0, 3 → 0},
  {Line[{{0, 0, 0}, {0, Ly, 0}}, "Fi"], 1 → 0}
 ];
SMTAddNaturalBoundary[
  {Line[{{Lx, 0, 0}, {Lx, Ly, 0}}, "D"], 1 → Line[{-q}]}];
SMTAnalysis[];
```

- solve linear elastic problem

In[29]:=
```
SMTNextStep["Δλ" → 1];
SMTNewtonIteration[];
```

- store the last assembled tangent matrix that is in this case linear elastic matrix or K_0

In[29]:=
```
SMTStoreTangentMatrix[];
```

- the sum of elastic and geometric tangent matrix $K_0 + K_\sigma$ is assembled here

In[44]:=
```
SMTIData[{"GeometricTangentMatrix", "SkipSolver"}, {3, 1}];
SMTNewtonIteration[];
SMTIData[{"GeometricTangentMatrix", "SkipSolver"}, {0, 0}];
```

Power method

- This calculates the smallest buckling load and shape using Power method.

In[47]:=
```
{λ, Ψ} = SMTPowerMethod["Method" → 4, "InitialVectorEstimate" → 1, "Eigenvectors" → True];
```

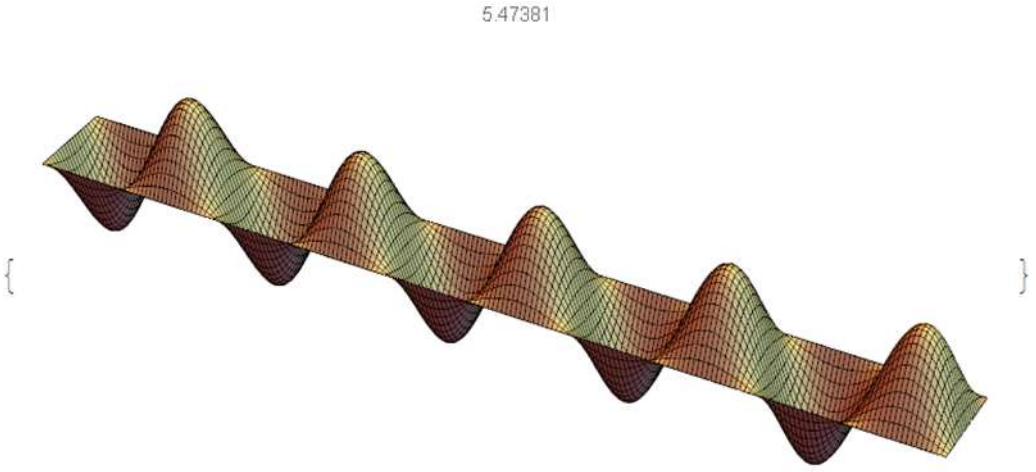
- Calculated eigenvalue:

In[48]:=
```
λ
```

Out[48]=
```
{5.47381}
```

- Show buckling mode:

In[49]:=
```
SMTShowEigenvectors[λ, Ψ, "Scale" → 2]
```



FEAST algorithm

- FEAST algorithm looks for buckling loads at interval [0,7] and returns corresponding buckling shapes.

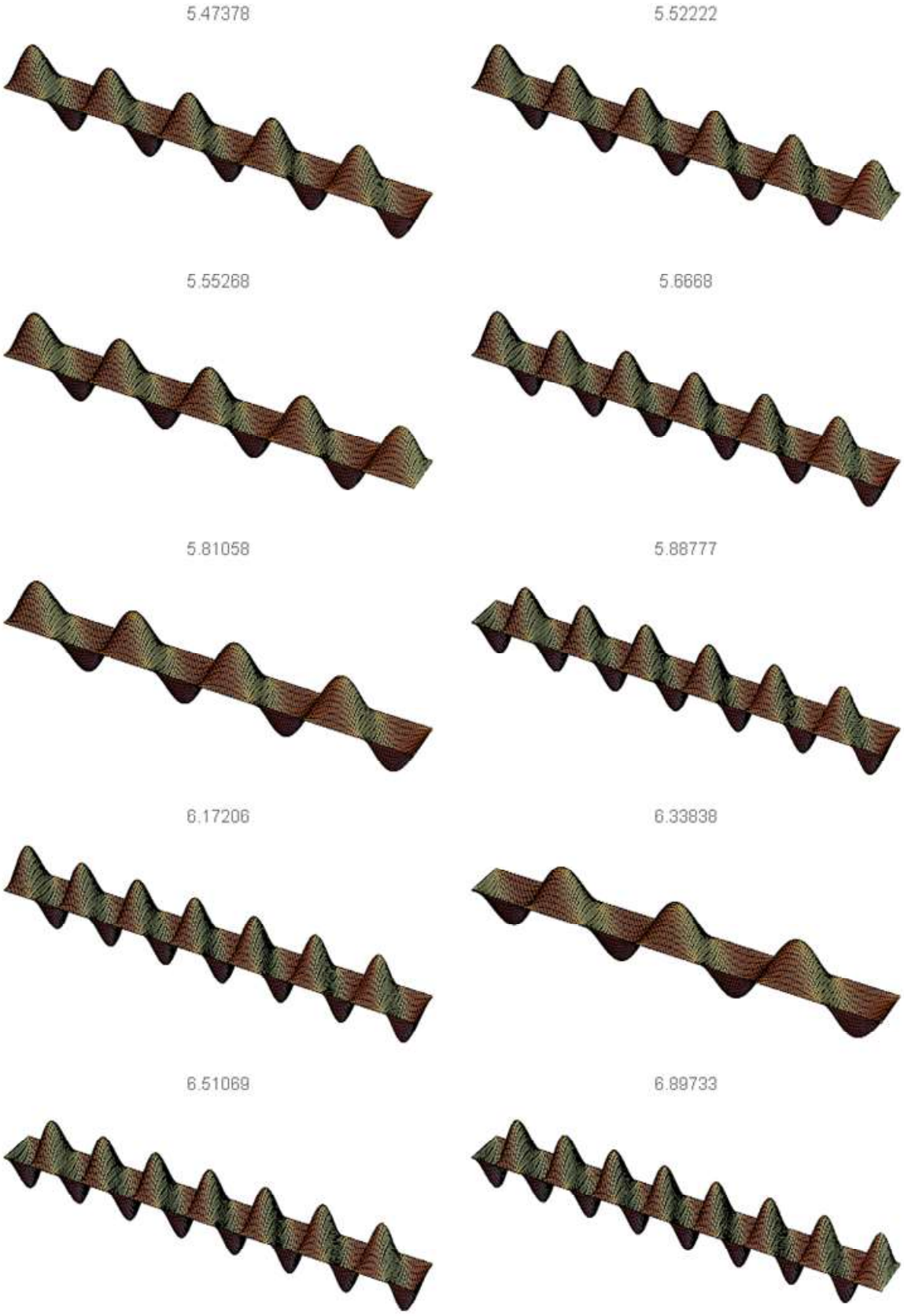
```
In[50]:= {λ, Ψ} = SMTFEAST[0, 7, "Method" → 2, "MaxNoEigenvaluesOnInterval" → 50,
            "NoContourPoints" → 24, "Eigenvectors" → True, "MaxNoIterations" → 1000];
```

- Calculated eigenvalues:

```
In[51]:= λ

Out[51]= {5.47378, 5.52222, 5.55268, 5.6668, 5.81058, 5.88777, 6.17206, 6.33838, 6.51069, 6.89733}
```

- Show buckling modes and loads

```
In[56]:= Grid[Partition[SMTShowEigenvectors[λ, Ψ, "Scale" → 2, ImageSize → 300], 2]]
```



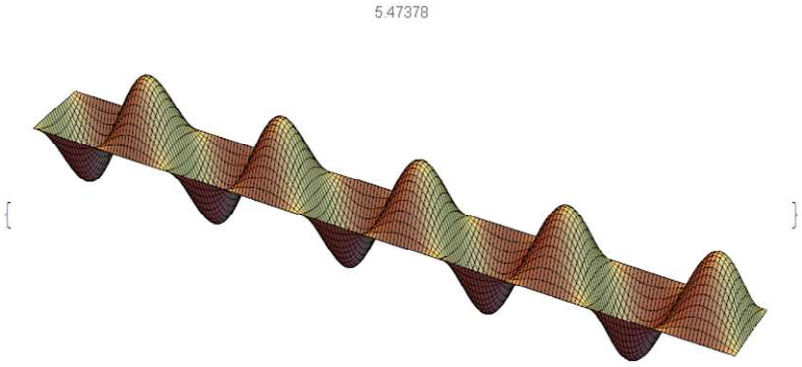
- memory used to store K_0 matrix is freed

In[120]:=
```
SMTFreeStoredTangentMatrix[];
```

Mathematica Eigensystem command

Buckling loads can be calculated also with the use of Mathematica's Eigensystem command. However, in that case the tangent matrix has to be transferred from CDriver to Mathematica with a substantial lost of memory space.

```
In[57]:= mp = SMTData["TangentMatrix"];
         SMTIData["GeometricTangentMatrix", 3];
         mt = SMTData["TangentMatrix"];
         SMTIData["GeometricTangentMatrix", 0];
         {γ, Φ} =
           Eigensystem[{mt, mp}, -1, Method → {"Arnoldi", MaxIterations → 800, "Tolerance" → 10^-8}];
         λ =
           1/(1 - γ);
In[63]:= SMTShowEigenvectors[λ, Φ, "Scale" → 2]
```



Example: Buckling of simply supported plate (linearized buckling analysis)

Simply supported plate is loaded in one axial direction. Plate is modelled with hyperelastic shell elements. Linearized buckling analysis is applied to calculate critical buckling load. In this analysis it is presumed that tangent matrix at arbitrary load level (λ) can be obtained as linear interpolation of tangent matrices (K_1 and K_2) at two different load levels (λ_1 and λ_2):

$K = K_1 + \gamma\,(K_2 - K_1)$

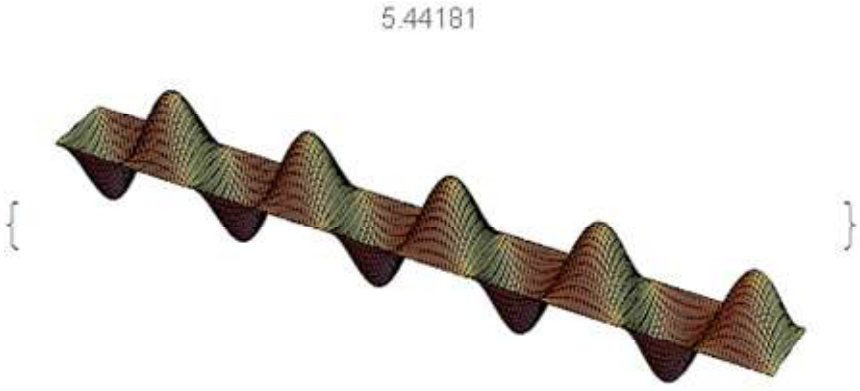
$\lambda = \lambda_1 + \gamma\,(\lambda_2 - \lambda_1)$

In AceFEM: K_2 is tangent matrix at current load level (λ_2). K_1 should be stored by user at arbitrary lower load level (λ_1), with call of SMTStoreTangentMatrix[] command. λ_1 should be given by user as option "λ1" in SMTPowerMethod and SMTFEAST command!

For calculation of critical buckling load one has to solve generalized eigenvalue problem: $(K_1 + \gamma_i\,(K_2 - K_1))\,\Psi_i = 0;\ i = 1, \ldots, n$ and then calculate critical load multipliers as $\lambda_i = \lambda_1 + \gamma_i\,(\lambda_2 - \lambda_1)$. Both calculations are performed in AceFEM and λ_i are the returned values. Ψ_i is corresponding eigenvector. The (absolutely) smallest λ_i is sought.

Power method

```
In[164]:= << AceFEM`
          q = 20 000;
          Lx = 10; Ly = 1;
          SMTInputData[];
          SMTAddDomain[{"plate", {"ML:", "SE", "MS", "S1", "DF", "HY", "S1P6", {"D", "Fi"}, "SVenant"},
              {"E *" → 210 000 000, "ν *" → 0.29, "t *" → 0.05}}];
          SMTAddMesh[Polygon[{{0, 0, 0}, {Lx, 0, 0}, {Lx, Ly, 0}, {0, Ly, 0}}],
            "plate", "Division" -> {200, 20}];
          SMTAddEssentialBoundary[
            {Line[{{0, 0, 0}, {Lx, 0, 0}}, "D"], 2 → 0, 3 → 0},
            {Line[{{0, 0, 0}, {Lx, 0, 0}}, "Fi"], 2 → 0},
            {Line[{{Lx, 0, 0}, {Lx, Ly, 0}}, "D"], 3 → 0},
            {Line[{{Lx, 0, 0}, {Lx, Ly, 0}}, "Fi"], 1 → 0},
            {Line[{{0, Ly, 0}, {Lx, Ly, 0}}, "D"], 3 → 0},
            {Line[{{0, Ly, 0}, {Lx, Ly, 0}}, "Fi"], 2 → 0},
            {Line[{{0, 0, 0}, {0, Ly, 0}}, "D"], 1 → 0, 3 → 0},
            {Line[{{0, 0, 0}, {0, Ly, 0}}, "Fi"], 1 → 0}
           ];
          SMTAddNaturalBoundary[{Line[{{Lx, 0, 0}, {Lx, Ly, 0}}, "D"], 1 → Line[{-q}]}];
          SMTAnalysis[];
          λmax = 1; nstep = 5;
          Δλ = λmax / nstep;
          tolNR = 10^-8; maxNR = 15;
          Do[
           SMTNextStep["Δλ" → Δλ];
           While[SMTConvergence[tolNR, maxNR], SMTNewtonIteration[];];
           If[i == 2, SMTStoreTangentMatrix[]; λ1 = SMTData["Multiplier"]];
           , {i, 1, nstep}]

In[78]:= {λ, Ψ} = SMTPowerMethod["Method" → 5, "MaxNoIterations" → 2000,
             "Eigenvectors" → True, "Tolerance" → 10^-8, "λ1" -> λ1];

In[79]:= λ

Out[79]= {5.44181}

In[80]:= SMTShowEigenvectors[λ, Ψ, "Scale" → 2]
```



FEAST algorithm

```
In[81]:= {λ, Ψ} = SMTFEAST[1, 7, "Method" → 3, "MaxNoEigenvaluesOnInterval" → 15,
             "NoContourPoints" → 24, "Eigenvectors" → True, "λ1" → λ1];
```

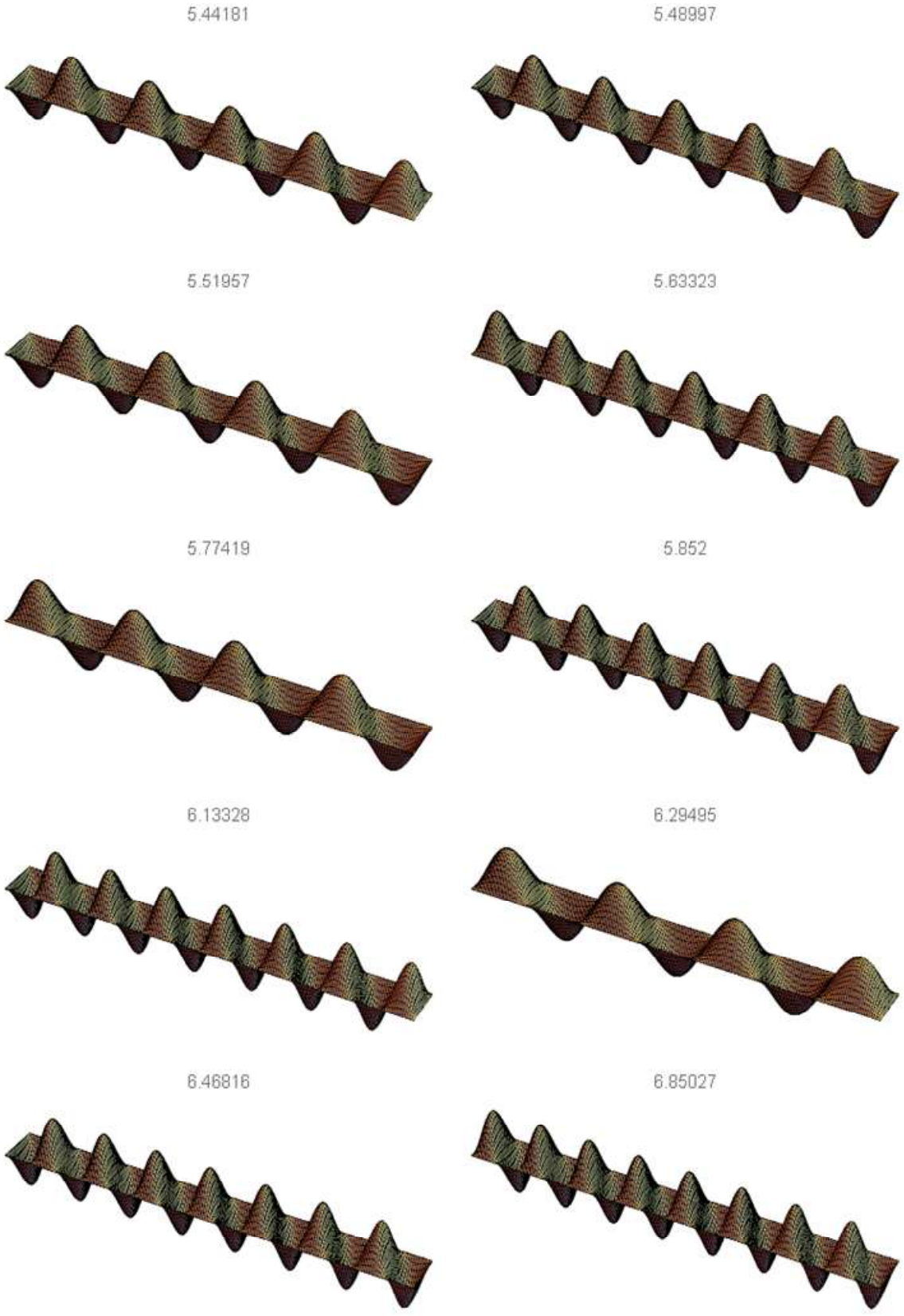
- Calculated eigenvalues:

In[82]:=
```
λ
```

```
{5.44181, 5.48997, 5.51957, 5.63323, 5.77419, 5.852, 6.13328, 6.29495, 6.46816, 6.85027}
```

- Show buckling modes:

In[83]:=
```
Grid[Partition[SMTShowEigenvectors[λ, Ψ, "Scale" → 2, ImageSize → 300], 2]]
```



- memory used to store K_1 matrix is freed

In[84]:=
```
SMTFreeStoredTangentMatrix[];
```

Mathematica Eigensystem command

Buckling loads can be calculated also with the use of Mathematica's Eigensystem command. In this the tangent matrix has to be transferred from CDriver to Mathematica with a substantial lost of memory space.

In[85]:=
```
<< AceFEM`
q = 20 000;
Lx = 10; Ly = 1;
SMTInputData[];
SMTAddDomain[{"plate", {"ML:", "SE", "MS", "S1", "DF", "HY", "S1P6", {"D", "Fi"}, "SVenant"},
    {"E *" → 210 000 000, "ν *" → 0.29, "t *" → 0.05}}];
SMTAddMesh[Polygon[{{0, 0, 0}, {Lx, 0, 0}, {Lx, Ly, 0}, {0, Ly, 0}}],
  "plate", "Division" -> {200, 20}];
SMTAddEssentialBoundary[
  {Line[{{0, 0, 0}, {Lx, 0, 0}}, "D"], 2 → 0, 3 → 0},
  {Line[{{0, 0, 0}, {Lx, 0, 0}}, "Fi"], 2 → 0},
  {Line[{{Lx, 0, 0}, {Lx, Ly, 0}}, "D"], 3 → 0},
  {Line[{{Lx, 0, 0}, {Lx, Ly, 0}}, "Fi"], 1 → 0},
  {Line[{{0, Ly, 0}, {Lx, Ly, 0}}, "D"], 3 → 0},
  {Line[{{0, Ly, 0}, {Lx, Ly, 0}}, "Fi"], 2 → 0},
  {Line[{{0, 0, 0}, {0, Ly, 0}}, "D"], 1 → 0, 3 → 0},
  {Line[{{0, 0, 0}, {0, Ly, 0}}, "Fi"], 1 → 0}
 ];
SMTAddNaturalBoundary[
  {Line[{{Lx, 0, 0}, {Lx, Ly, 0}}, "D"], 1 → Line[{-q}]}];
SMTAnalysis[];
λmax = 1; nstep = 5;
Δλ = λmax / nstep;
tolNR = 10^-8; maxNR = 15;
Do[
  SMTNextStep["Δλ" → Δλ];
  While[SMTConvergence[tolNR, maxNR], SMTNewtonIteration[];];
  If[i == 2, K1 = SMTData["TangentMatrix"]; λ1 = SMTData["Multiplier"]];
  , {i, 1, nstep}];
K2 = SMTData["TangentMatrix"];
λ2 = SMTData["Multiplier"];
```

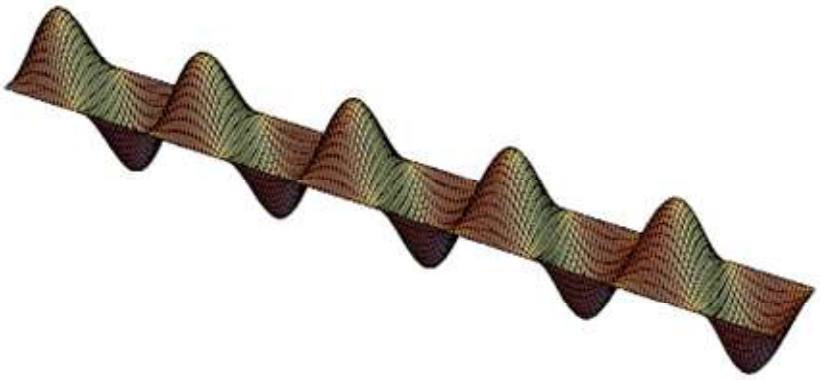
In[100]:=
```
{γ, Ψ} = Eigensystem[{K2, K1}, -1, Method → {Arnoldi, "MaxIterations" -> 100 000}];
λ = λ1 + 1/(1 - γ) (λ2 - λ1);
```

In[102]:=
```
λ
```

Out[102]=
```
{5.44181}
```

In[104]:=
```
SMTShowEigenvectors[λ, Ψ, "Scale" → 2]
```

5.44181

{  }

CHAPTER 3

Control of Solution Procedures

Data Base Manipulations

- General Description
  - SMTData
- Manipulate Integer Type Environment Data
  - SMTIData
  - General data
  - Mesh input related data
  - Iterative procedure related data
  - Debugging and errors related data
  - Linear solver related data
  - Sensitivity related data
  - Contact related data
- Manipulate Real Type Environment Data
  - SMTRData
  - Real Type Environment Data
- Manipulate Node Data
  - SMTNodeData
  - Node Data
  - Interpreted nodal data
  - Examples : SMTNodeData
- Manipulate Node Specification Data
  - SMTNodeSpecData
  - Node Specification Data
- Manipulate Element Data
  - SMTElementData
  - Element Data
- Manipulate Domain Specification Data
  - SMTDomainData
  - Domain Specification Data
  - Memory allocation
  - General Data
  - Run Mathematica from code
  - Mesh generation
  - Domain input data
  - Numerical integration
  - Graphics postprocessing
  - Sensitivity analysis

General Description

The SMTAnalysis command transcripts input data structures into analysis data base structures. The analysis data base structures can be accessed and changed during the analysis. The user can interactively change during the analysis all environment data, node coordinates, values of the essential and natural boundary conditions and all unknowns of the problem, elements nodes, material data and elements history variables.

- Integer Type Environment Data (in AceGen idata$$,in AceFEM SMTIData)
- Real Type Environment Data (in AceGen rdata$$, in AceFEM SMTRData)
- Domain Specification Data (in AceGen es$$, in AceFEM SMTDomainData)
- Element Data (in AceGen ed$$, in AceFEM SMTElementData)
- Node Specification Data (in AceGen ns$$, in AceFEM SMTNodeSpecData)
- Node Data (in AceGen nd$$, in AceFEM SMTNodeData)
- method used to solve linear system of equations during the iterative procedure (see SMTSetSolver).
- various data such as tangent matrix and residual related to the whole structure as well as to the particular element (see SMTData).

Manipulate Integer Type Environment Data

SMTIData

SMTIData[*code*]
get the value of the integer type environment variable with the associated keyword *code*

SMTIData[*code*,*value*]
set the value of the integer type environment variable with the associated keyword *code* to be equal *value*

SMTIData["All"]
get all names and values of integer type environment variables

Get or set the real or integer type environment variable.

The values of the *code* parameter are specified in:

- General data
- Mesh input related data
- Iterative procedure related data
- Debugging and errors related data
- Linear solver related data
- Sensitivity related data
- Contact related data

Example

- This returns the number of nodes.

In[31]:=
```
SMTIData["NoNodes"]
```

Out[31]=
```
9261
```

Manipulate Real Type Environment Data

SMTRData

SMTRData[*code*]
get the value of the real type environment variable with the code *code*

SMTRData[*code*,*value*]
set the value of the real type environment variable with the code *code* to be equal *value*

SMTRData["All"]
get all names and values of integer type environment variables

Get or set the real type environment variable.

The values of the *code* parameter are specified in `Real Type Environment Data`.

- This returns the current value of the boundary conditions multiplier λ.

```
In[32]:= SMTRData["Multiplier"]
Out[32]= 1.
```

Manipulate Node Data

SMTNodeData

SMTNodeData[*code*]
get the data with the code *code* for all nodes

SMTNodeData[*nodeSelector*,*code*]
get the data with the code *code* for all nodes selected by *nodeSelector*

SMTNodeData[*nodeSelector*,*code*,*value*]
set the data with the code *code* for all nodes selected by *nodeSelector* to be equal given *value*

SMTNodeData[*nodeSelector*,*code*,{v_1,,...,v_n}]
set the data with the code *code* for nodes selected by *nodeSelector* to be equal given values {v_1,,...,v_n}

SMTNodeData[*code*, *value*]
set the data with the code *code* for all nodes to be equal given *value*

SMTNodeData[*code*,{v_1,...,$v_{NoNodes}$}]
set the data with the code *code* for all nodes to be equal given values

Get or set the general nodal data.

The values of the *code* parameter are specified in `Node Data`. The *nodeSelector* can be a node number, a list of node numbers or a logical expression (see also `Selecting nodes, elements and bodies`).

IMPORTANT: The structure of the global tangent matrix has to be updated with the `SMTSetSolver` command in the case that the change of element or nodal data has effect on the structure of the global tangent matrix (e.g. if essential boundary conditions are added or removed during the analysis).

Interpreted nodal data

SMTNodeData["Unknowns"]
get a vector of current values of all unknowns of the problem

SMTNodeData["Unknowns",{a_1,a_2,...,$a_{NoEquations}$}]
set the current value of all unknowns of the problem

SMTNodeData[*nodeSelector*,"AllElements"]
a set of all elements topologically associated with the nodes for selected nodes

SMTNodeData[*nodeSelector*,"NoAllElements"]
length of a set of all elements topologically associated with the nodes for selected nodes

SMTNodeData["Coordinates"]
get coordinates of all nodes

Get an additional information related to the node that is not a part of the nodal data structure.

The vector of unknowns is composed of all nodal degrees of freedom that are not constrained. The ordering of the components is done accordingly to the current ordering of equations as defined by SMTNodeData["DOF"];

SMTNodeData[*nodeSelector*, "NodeID"]
get the node specification *NodeID* for nodes selected by *nodeSelector*

SMTNodeData["NodeID"]
get the node specification *NodeID* for all nodes

SMTNodeData[*nodeSelector*,All]
get all names and values of the data for nodes selected by *nodeSelector*

SMTNodeData[All]
get all names and values of the data for all nodes

Get an additional information related to the node that is not a part of the nodal data structure.

Examples: SMTNodeData

- This returns the current values of all unknowns in node 5.

```
In[33]:= SMTNodeData[5, "at"]
```

- This adds 1 to unknowns in first 10 nodes.

```
In[34]:= SMTNodeData[Range[10], "at", 1 + SMTNodeData[Range[10], "at"] ]
```

- This sets all unknowns to zero in all nodes (assuming that they all have 2 unknowns).

```
In[35]:= SMTNodeData["at", {0, 0} ]
```

- This sets the design velocity fields in node 5 to {1,0,1}.

```
In[233]:= SMTNodeData[5, "ADVF", {1, 0, 1} ]
```

Manipulate Node Specification Data

SMTNodeSpecData

SMTNodeSpecData[*code*]
get the data with the code *code* for all node specifications

SMTNodeSpecData[*NodeID*, *code*]
get the data with the code *code* for the node specification *NodeID*

SMTNodeSpecData[*NodeID*,"All"]
get all names and values of the data for the node specification *NodeID*

SMTNodeSpecData[*NodeID*, "DummyNode"]
returns an index of the dummy node associated with node specification *NodeID* (see `Node Identification`). The dummy nodes can not be selected directly by SMTFindNodes command. An index of the dummy node associated with nodes with node identification *NodeID* can be accessed by SMTNodeSpecData[*NodeID*,"DummyNode"] command.

Get node specification data.

The values of the *code* parameter are specified in `Node Specification Data`. Node specification data can not be changed.

Manipulate Element Data

SMTElementData

SMTElementData[*code*]
get the data with the code *code* for all elements

SMTElementData[*elementSelector*, *code*]
get the data with the code *code* for all elements selected by *elementSelector*

SMTElementData[*elementSelector, code, value*]
set the data with the code *code* for all elements selected by *elementSelector* to be equal given *value*

SMTElementData[*elementSelector*,*code*,{v_1,,...,v_n}]
set the data with the code *code* for elements selected by *elementSelector* to be equal given values {v_1,,...,v_n}

SMTElementData[*code*, *value*]
set the data with the code *code* for all elements to be equal given *value*

SMTElementData[*code*,{v_1,,...,$v_{NoElements}$}]
set the data with the code *code* for all elements to be equal given values

Get or set the element data.

SMTElementData["Domain"]
SMTElementData[*elementSelector*, "Domain"]
get the domain identification dID for all elements or for the selection of elements

SMTElementData["Body"]
SMTElementData[*elementSelector*, "Body"]
get the body identification bID for all elements or for the selection of elements

SMTElementData["Code"]
SMTElementData[*elementSelector*, "Code"]
get the element type for all elements or for the selection of elements

SMTElementData["Volume"]
SMTElementData[*elementSelector*, "Volume"]
get the volume/area/length of the element accordingly to the element topology

(volume is only **approximative** for the elements with curved edges !)

SMTElementData[All]
SMTElementData[*elementSelector*,All]
get all names and values for all elements or for the selection of elements

Get an additional information related to the element that is not a part of the element's data structure.

The values of the *code* parameter are specified in `Element Data`. The *elementSelector* can be an element number, a list of element numbers or a logical expression (see also `Selecting nodes, elements and bodies`).

- This returns a list of nodes of 35-th element.

In[234]:=
```
SMTElementData[35, "Nodes"]
```

Manipulate Domain Specification Data

SMTDomainData

SMTDomainData[*code*]
get the data with the code *code* for all domains

SMTDomainData[*dID*, *code*]
get the data with the code *code* for domain *dID*

SMTDomainData[*dID*, *code*, *value*]
set the data with the code *code* for domain *dID* to be equal *value*

SMTDomainData[*dID*, "Data", *keyword*]
get input data for domain dID that is stored under keyword *keyword* (keywords are defined by SMSDomainDataNames, see `Template Constants`)

SMTDomainData[*dID*, "Data", *keyword->value*]
set input data for domain *dID* that is stored under keyword *keyword*

SMTDomainData[*i_Integer*, *code*]
get the data with the code *code* for *i*-th domain

SMTDomainData[*i_Integer*, *code*, *value*]
set the data with the code *code* for *i*-th domain to be equal *value*

Get or set the element specification data.

SMTDomainData[*dID*,"All"]
get all names and values of the domain specific data for domain *dID*

SMTDomainData["DomainID"]
get a list of domain identifications for all domains

Get an additional information related to the domain that is not a part of the domain's data structure.

The values of the *code* parameter are specified in `Domain Specification Data`:

- `Memory allocation`
- `General Data`
- `Run Mathematica from code`
- `Mesh generation`
- `Domain input data`
- `Numerical integration`
- `Graphics post - processing`
- `Sensitivity analysis`

Examples

- This returns material data specified for the domain "solid".

In[235]:=
```
SMTDomainData["solid", "Data"]
```

- This sets all material data specified for the domain "solid" to new values.

In[27]:=
```
SMTDomainData["solid", "Data", {E, ν, σy, ...}]
```

- This sets only material data with the keyword "E" (see SMSDomainDataNames in chapter `Template Constants`) for the domain "solid" to new value.

In[27]:=
```
SMTDomainData["solid", "Data", "E" -> 1000]
```

Active Mesh Control

Contents

General Description

At the input data phase, the mesh, the boundary conditions and the element properties are defined. With the active mesh control commands we can change during the analysis the problem as defined by the input data. The presented commands are implemented directly within the CDriver, thus they are fast and can be called frequently. The changes of input data include:

- elements can be included or excluded from the global assembly procedures,
- elements can be excluded/included from visualisation,
- degrees of freedom of the nodes can be constrained/unconstrained.

Modify Elements and Nodes

SMTModifyElements

SMTModifyElements[*elementSelector, options*]
sets attributes of the selected elements accordingly to the given options

option	default	description
"Assembly"	Automatic	True ⇒ selected elements contributions (tangent, residual , pseudo-load vectors, post-processing) are assembled into global matrices and vectors False ⇒ selected elements are ignored during global assembly procedures
"Visualization"	Automatic	True ⇒ selected elements are used for visualization False ⇒ selected elements are ignored for visualization
"Update"	Automatic	After the change of elements attributes the CDriver database remains inconsistent. This is also true for the parts of the data stored in Mathematica. With the "Update" option we can regulate what is updated after the SMTModifyElements command. An argument is a list of keywords chosen from "Solver", "MMA", "Global surface", "Local surface".

Options of the *SMTModifyElements* function.

Simulation of process of delamination

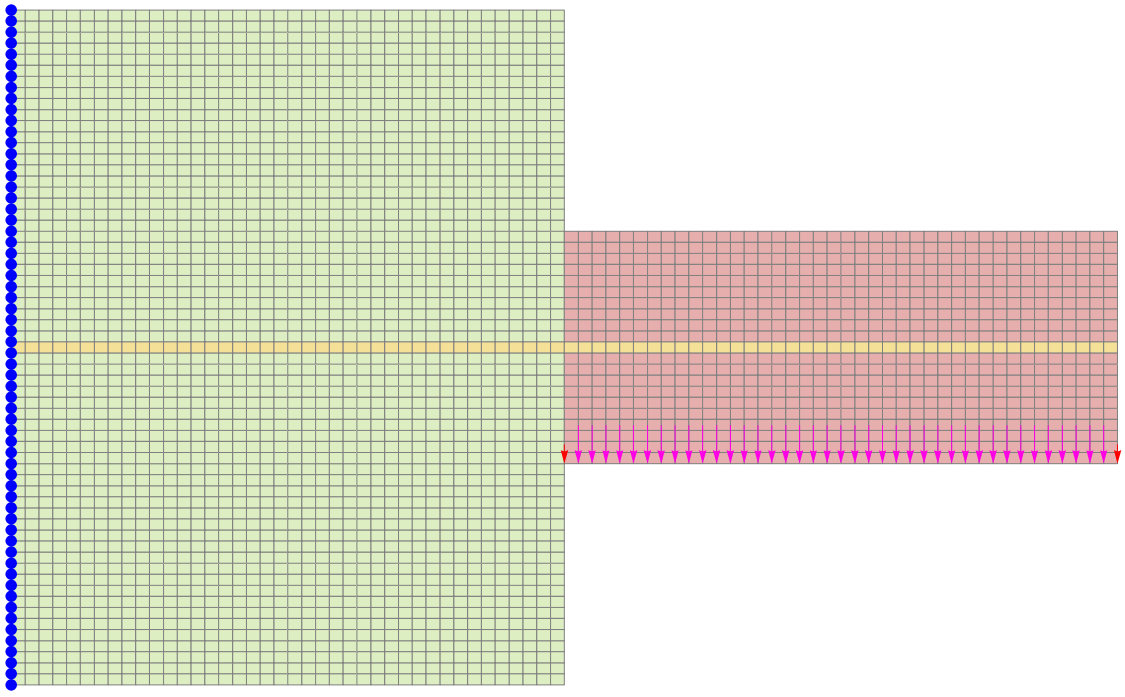
Let assume that parts of the structure are glued together and that the glue and the structures are meshed with finite elements. The process of delamination creates a split within the mesh, thus requires some elements of the glue to be removed from the analysis as well as from visualization procedures. If the load is cyclic some parts of the glue that have been damaged in tension can come into compression and thus become active again. The strategy adopted here is as follows:

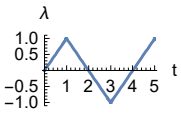
- element of the glue is removed from the analysis (made inactive for assembly and visualization) when:
  - element is undamaged, the normal stress in the glue (σyy) is tension and stress exceeds threshold value (pTreshold),
  - glue has been damaged already before and the normal stress in the glue is tension,

- element of the glue that are in compression are always active,
- the slip between the delaminated parts is ignored.

In[177]:=
```
<< AceFEM`;
SMTInputData["Console" → False];
L1 = 100; L2 = 100; H = 20; δ = H / 10; nx = 40; ny = 10; nx1 = 40;
SMTAddDomain["A", {"ML:", "SE", "PE", "Q1", "DF", "HY", "Q1", "D", {{"NeoHooke", "WA"}}},
  {"E *" → 20 000, "ν *" → 0.2}];
SMTAddDomain["glue", {"ML:", "SE", "PE", "Q1", "DF", "HY", "Q1", "D", {{"NeoHooke", "WA"}}},
  {"E *" → 100, "ν *" → 0.2}];
SMTAddDomain["C", {"ML:", "SE", "PE", "Q1", "DF", "HY", "Q1", "D", {{"NeoHooke", "WA"}}},
  {"E *" → 40 000, "ν *" → 0.2}];
SMTAddMesh[Polygon[{{0, -3 H}, {L1, -3 H}, {L1, 0}, {0, 0}}], "C", "Division" -> {nx, 3 ny}];
SMTAddMesh[Polygon[{{0, 0}, {L1, 0}, {L1, δ}, {0, δ}}], "glue", "Division" -> {nx, 1}];
SMTAddMesh[Polygon[{{0, δ}, {L1, δ}, {L1, δ + 3 H}, {0, δ + 3 H}}], "C", "Division" -> {nx, 3 ny}];
SMTAddMesh[Polygon[{{L1, -H}, {L1 + L2, -H}, {L1 + L2, 0}, {L1, 0}}],
  "A", "Division" -> {nx1, ny}];
SMTAddMesh[Polygon[{{L1, 0}, {L1 + L2, 0}, {L1 + L2, δ}, {L1, δ}}],
  "glue", "Division" -> {nx1, 1}];
SMTAddMesh[Polygon[{{L1, δ}, {L1 + L2, δ}, {L1 + L2, δ + H}, {L1, δ + H}}],
  "A", "Division" -> {nx1, ny}];
SMTAddEssentialBoundary[Line[{{0, -5 H}, {0, 5 H}}], 1 -> 0, 2 -> 0];
SMTAddNaturalBoundary[Line[{{L1, -H}, {L1 + L2, -H}}], 2 → Line[{-25}]];
SMTAnalysis[];
```

In[55]:=
```
λf[t_] := If[OddQ[Floor[(t + 1) / 2]], 1, -1] (2 Floor[(t + 1) / 2] - t);
Row[{SMTShowMesh["BoundaryConditions" → True], Plot[λf[t], {t, 0, 5}, AxesLabel → {"t", "λ"}]}]
```



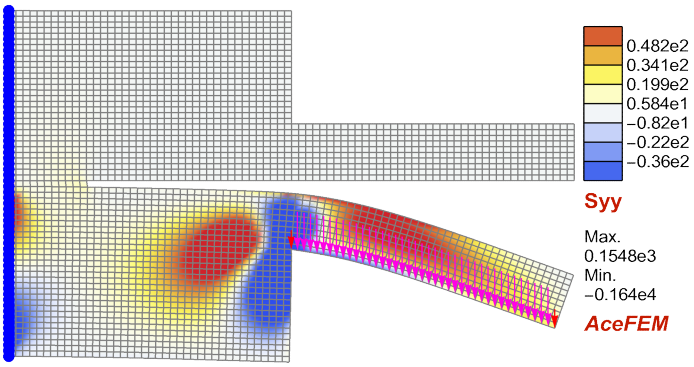


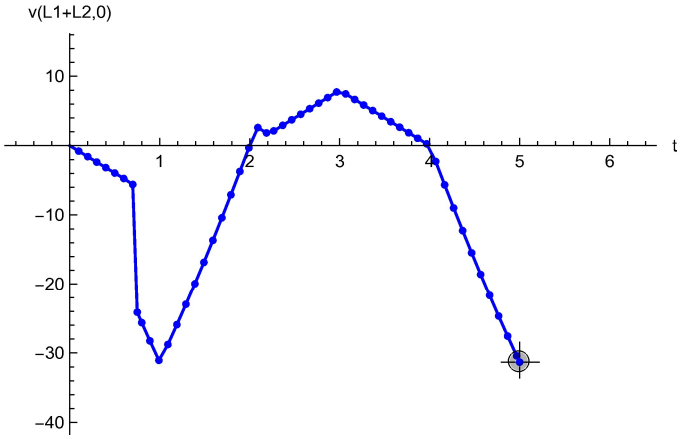
In[57]:=
```
glueElements = SMTElementData["glue", "ElemIndex"];
damagedElements = {};
pTreshold = 15;
```

- The normal stress (σyy) is measured at the center of the elements of the glue.

In[60]:=
```
X = SMTNodeData["X"];
midpoints = Table[Total[X[[e]]] / Length[e], {e, SMTElementData["glue", "Nodes"]}];
```

In[62]:=
```
tMax = 5; t0 = 0.1; ΔtMin = 1 / 1000.; ΔtMax = 0.1;
tolNR = 10. ^ -8; maxNR = 15; targetNR = 8;
SMTNextStep["t" → t0, "λ[t]" → λf];
SMTAnimationOfResponse["Initialize", {0, 0}];
While[
  While[
   step = SMTConvergence[tolNR, maxNR, {"Adaptive Time", targetNR, ΔtMin, ΔtMax, tMax}],
   If[SMTIData["Iteration"] < 5,
    σyy = SMTPostData["Syy", Point[midpoints]];
    tensionElements = Extract[glueElements, Position[σyy, _?(# > pTreshold &)]];
    SMTModifyElements[Union[tensionElements, damagedElements],
     "Assembly" → False, "Visualization" → False];
    compressionElements = Extract[glueElements, Position[σyy, _?(# <= pTreshold / 100. &)]];
    SMTModifyElements[compressionElements, "Assembly" → True, "Visualization" → True];
   ];
   SMTNewtonIteration[];
  ];
  If[Not[step[[1]]]
   , damagedElements = Union[tensionElements, damagedElements];
   SMTAnimationOfResponse["y" → Hold[SMTPostData["v", Point[{L1 + L2, 0}]]],
    "x" → Hold[SMTRData["Time"]], "ShowMeshOptions" → {"Field" → "Syy", Axes → False},
    "PlotOptions" → {AxesLabel → {"t", "v(L1+L2,0)"}}];
  ];
  If[step⟦4⟧ === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step⟦3⟧
  , If[step⟦1⟧, SMTStepBack[];];
  SMTNextStep["Δt" → step⟦2⟧, "λ[t]" → λf];
 ];
SMTAnimationOfResponse["Last frame"]
```


0.482e2
0.341e2
0.199e2
0.584e1
−0.82e1
−0.22e2
−0.36e2
Syy
Max.
0.1548e3
Min.
−0.164e4
AceFEM


v(L1+L2,0)
10
−10
−20
−30
−40
1
2
3
4
5
6
t

Iterative Arc-length Solution Procedure

Contents

Implementation Notes for Arc-length solution procedure

Theory

Standard arc length method with quadratic constraint

The arc-length method (also continuation method) is an iterative method used for tracing non-linear equilibrium path of solid, especially when equilibrium path contains critical points, e.g. limit point, bifurcation point, snap-back and snap-through. Iterative solution procedure solves consistently linearized augmented system which is composed of equilibrium equations $\mathbf{R}(\Delta\mathbf{p},\Delta\lambda)=\mathbf{0}$ and constraint equation $g(\Delta\mathbf{p},\Delta\lambda)=0$ for set of variables $\Delta\mathbf{p}$ and $\Delta\lambda$ with unknown values. Constraint equation

$$g(\Delta\boldsymbol{p},\Delta\lambda) = \Delta\boldsymbol{p}^T\Delta\boldsymbol{p} + \Delta\lambda^2\,\Psi_1^2\,\boldsymbol{R}_{ref}^T\boldsymbol{R}_{ref} + \Delta\lambda^2\,\Psi_2^2\,\boldsymbol{p}_{ref}^T\boldsymbol{p}_{ref} - \Delta s^2 = 0$$

considers increment of arc length of equilibrium path Δs to determine incremental vector of the variables $\Delta\mathbf{p}$ and $\Delta\lambda$ with unknown values. Table below shows the symbols used in constraint equation $g(\Delta\mathbf{p},\Delta\lambda)$, their notation in AceFEM and description.

Symbol in constraint equation	AceFEM	description
$\Delta\mathbf{p}$	**da**	increment of global vector of nodal degree of freedoms
$\Delta\lambda$	$\Delta\lambda$	incremental load multiplier
Δs	$\Delta\gamma$	increment of arc length of equilibrium path
Ψ_1, Ψ_2	"Ψ1", "Ψ2"	scaling parameters
$\mathbf{R}_{ref}$	$\overline{\boldsymbol{Bt}}$	reference load vector (see `SMTAddNaturalBoundary`)
$\mathbf{p}_{ref}$	$\tilde{\boldsymbol{Bt}}$	referenced vector of prescribed displacements (see `SMTAddEssentialBoundary`)

Constraint equation.

Generally, arc-length method is based on a two-step solution strategy. First step is predictor phase which executes first iteration and define course of analysis in current increment. For this reason it is important to choose a proper criterion to determine correct direction (for details see SMTArcLengthSet). Second step is called corrector and executes following Newton-Raphson iterations in current increment.

Path- following method with adaptive one DOF constraint

Reference: Stanic, A., Brank, B., Korelc, J. 2016. On path-following methods for structural failure problems. Computational Mechanics 58: 281-306.

Constraint equation:

$$g(\Delta\boldsymbol{p}) = \boldsymbol{w}_{n+1}^T \Delta\boldsymbol{p} - s_{n+1}\,\Delta l = 0$$

The constraint prescribes incremental change of chosen degree of freedom (the control DOF). The control DOF may change from one increment to another. Table below shows the symbols used in constraint equation g(Δ**p**), their notation in AceFEM and description.

Symbol in constraint equation	AceFEM	description
$\Delta\boldsymbol{p}$	**da**	increment of global vector of nodal degree of freedoms (DOFs)
Δl	$\Delta\gamma$	prescribed incremental change of chosen DOF
$\boldsymbol{w}_{n+1}$	$\boldsymbol{w}_{n+1}$	vector of zero entries, except for a single nonzero entry that is set to 1 and defines the controlled incremental DOF (see SMTArcLengthControlDOF)
s_{n+1}	s_{n+1}	$s_{n+1}=\text{Sign}[\mathbf{w}_{n+1}^T\,\Delta\mathbf{p}_{n-1}]$

Constraint equation.

Path- following method with constraint equation based on dissipation

Reference: Stanic, A., Brank, B., 2017. A path-following method for elasto-plastic solids and structures based on control of plastic dissipation and plastic work. Finite Elements in Analysis and Design 123: 1-8.

Constraint equation:

$$g(\Delta\boldsymbol{p}) = \Delta\boldsymbol{p}_n^T\left(\lambda_n\,\hat{\boldsymbol{f}}^{ext} - \boldsymbol{f}_n^*\right) - \Delta\tau = 0$$

The constraint prescribes incremental growth of dissipation. Table below shows the symbols used in constraint equation g(Δ**p**), their notation in AceFEM and description.

Symbol in constraint equation	AceFEM	description
$\Delta \boldsymbol{p}_n$	**da**	Increment of global vector of nodal degree of freedoms (DOFs)
$\Delta \tau$	$\Delta\gamma$	Prescribed incremental change of dissipation
$\boldsymbol{f}_n^*$	$\boldsymbol{f}_n^*$	The nodal force vector. The $\boldsymbol{f}_n^*$ is computed on finite element mesh level and it has the same form as the reference external load vector (see SMTTask). The command SMTArcLengthVectorGlobal imports the $\boldsymbol{f}_n^*$ into the constraint.
$\hat{\boldsymbol{f}}^{ext}$	**dB**	Reference external load vector
λ_n		Load multiplier from the last converged equilibrium state

Constraint equation.

References

[1] Stanic, A., Brank, B., Korelc, J. 2016.
On path – following methods for structural failure problems. C omputational Mechanics 58 : 281 – 306.

[2] Stanic, A., Brank, B., 2017.
A path – following method for elasto –
plastic solids and structures based on control of plastic dissipation and plastic work. Finite Elements in Analysis and Desi

[3] Crisfield M. A. 1991.
Non – linear Finite Element Analysis of Solids and Structures, Vol .1 : Essentials. Chichester, John Wiley & Sons : 345 p.

[4] de Souza Neto E. A. , Feng Y. T. 1999.
On the determination of the path direction for arc – length methods in the presence of bifurcations and ' snap –
backs '. *Computer Methods in Applied Mechanics and Engineering 179 : 81 – 89 .*

AceFEM implementation

Arc Length method can be used for time-independent problem. AceFEM has no predefined commands to perform complete path-following procedure (implicit solution procedure of the parameterized system of nonlinear equations), however it provides a set of commands that can be used to write an arbitrary path-following procedure. The most essential commands needed to construct an iterative arc-length solution procedure are:

SMTArcLengthSet[]	sets the properties of the arc-length analysis (for details see SMTArcLengthSet) and returns an estimated arc-length for given load level
SMTNextStep["$\Delta\gamma$"→$\Delta\gamma$]	goes to the next arc-length step by updating the current length of equilibrium path ($\gamma \leftarrow \gamma+\Delta\gamma$) and time dependent data (***ap*≡*at*, *sp*≡*st*, *hp*≡*ht*, *Bp*≡*Bt***) (for details see `SMTNextStep`).
SMTConvergence[*tolerance*, *max_iterations*, {"Adaptive γ", m, $\Delta\gamma_{min}$, $\Delta\gamma_{max}$, γ_{max}}]	returns True if $\|\|\Delta \boldsymbol{at}\|\| > tolerance$ and thus one more iteration is required, False if $\|\|\Delta \boldsymbol{at}\|\| \le tolerance$ or aborts the calculation if the number of iterations has exceeded *max_iterations* (for details see `SMTConvergence`).
SMTArcLengthIteration[]	executes one iteration of the Arc-length iterative procedure.
SMTStepBack[]	makes the current state of the system to be the same as the one at the end of the previous time step (***at*≡*ap*, *st*≡*sp*, *ht*≡*hp*, *Bt*≡*Bp***).
SMTArcLengthNext[]	Updates time dependent data (for details see SMTArcLengthNext).
SMTArcLengthFree[]	Deallocates memory blocks. Arc length analysis session is closed. (for details see SMTArcLengthFree).

Essential commands for arc-length analysis.

SMTIData["ArcLengthStatus"]	0 ⇒ arc length method is not activated 1 ⇒ arc length method is activated
SMTIData["ArcLengthPredictorSignCriterion"]	1 ⇒ the criterion proposed by Feng, Peric and Owen [1] 2 ⇒ the criterion based on the sign of the determinant of current stiffness matrix [2]
SMTIData["ArcLengthMethod"]	1 ⇒ ordinary arc length analysis 2 ⇒ arc length analysis by Pohl, i.e. arc length analysis for softening
SMTRData["ArcLengthPsi1"]	Ψ_1
SMTRData["ArcLengthPsi2"]	Ψ_2
SMTRData["ArcLengthDirectionCheck"]	If the wrong sign is predicted, the solution sequence ' doubles back' on the original load-deflection curve and the Arc-length method fails to trace the complete path. If wrong direction is detected, Arc-length procedure does SMTStepBack[] and halves next step. Since, the check is costly, the direction is checked only after increment error is less than prescribed ($\|\mathbf{\Delta a}\|<err$).
SMTRData["ArcLengthDirectionTolerance"]	Direction is found to be false when cos $\phi<-1+tol$, where ϕ is the angle between the increment vector in previous and the current step.

Data for arc-length analysis.

The SMTConvergence command performs an analysis of the current state of the Arc-length iterative procedure and returns the results of the analysis. The user is then responsible to act accordingly to the results (e.g. to make an additional iteration or to stop the iterative procedure). The SMTConvergence function returns in the case when one more iteration is required within the current time/load step the value True. For other cases the return value depends on the type of path following procedure and the options given to the SMTConvergence function (for details see `SMTConvergence`). The return value has to be properly interpreted by the user and an appropriate action has to be performed accordingly to the return value.

NOTE: In case of solving problems with Arc-length method in AceFEM, it is obligated to set type "Adaptive γ" within command SMTConvergence. General parameters ($\Delta\gamma_{min}$, $\Delta\gamma_{max}$, γ_{max}) are now arc-length parameters (Δs_{min}, Δs_{max}, s_{max}).

Basic example: Snap-back phenomena of shell

Example presents an isotropic shell which is loaded by reference force $\mathbf{R}_{ref}$. Geometrical and material properties are taken from the article: *Sze K. Y., Liu X. H., Lo S. H., Popular Benchmark problems for geometric nonlinear analysis of shells, Finite Elements in Analysis and Design, 40: 1551-1569 (2004)*.

Shell arch with given width L_y, radius R and circle sector central angle θ is loaded with point load applied in the middle of arch. All nodes on edges at $x = -L_x$ and $x = L_x$ are constrained by pinned constraints.

In[36]:=
```
<< AceFEM`;
```

In[192]:=
```
(*geometry*)
Ly = 508; R = 2540; θ = 0.1;
H = R - R Cos[θ]; Lx = R Sin[θ];
raster = Table[{R Sin[θi], bi , R Cos[θi] - R + H}, {θi, -θ, θ, θ/6}, {bi, 0, Ly, Ly}];
SMTInputData[];
SMTAddDomain[{"A", {"ML:", "SE", "MS", "S1", "ES", "HY", "ANSE4P6", {"D", "Fi"}, "SVenant"},
   {"E *" → 3102.75, "ν *" → 0.3, "t *" → 6.35, "μDrill *" → 1.}}];
SMTAddMesh[Raster[raster], "A", "Division" -> {24, 24}];
SMTAddEssentialBoundary[{
   {Line[{{-Lx, 0, 0}, {-Lx, Ly, 0}}, "D"], 1 → 0, 2 → 0, 3 → 0 },
   {Line[{{Lx, 0, 0}, {Lx, Ly, 0}}, "D"], 1 → 0, 2 → 0, 3 → 0}}];
SMTAddNaturalBoundary[{Point[{0, Ly/2, H}, "D"], 3 → -3000 }];
SMTAnalysis[];
graf = {};
```

Arc length procedure

First step of Arc-length procedure is using command SMTArcLengthSet[] which initializes properties of arc-length analysis. It returns an estimated arc-length for the target load level 3 on a assumption that the problem is linear.

In[47]:=
```
sMax = SMTArcLengthSet["λTarget" → 3];
```

Terminal arc-length curve length *sMax*, maximum steps Δ*sMax,* minimum steps Δ*sMin* and initial increment of arc length *s0* are chosen:

In[48]:=
```
ΔsMax = sMax / 20;
s0 = sMax / 100;
ΔsMin = sMax / 1000;
```

Adaptive Time procedure with Arc-length iterations is used to solve described problem. Procedure is stopped when sMax is reached. After each successful step, SMTArcLengthNext[] must be called.

In[51]:=
```
SMTNextStep["Δγ" → s0];
While[
  While[
   step = SMTConvergence[10^-10, 50, {"Adaptive γ", 16, ΔsMin, ΔsMax, sMax}]
   , SMTArcLengthIteration[];
  ];
  If[Not[step[[1]]],
   AppendTo[graf, {SMTNodeData[Point[{0, Ly / 2, H}], "at"][[1, 3]], SMTRData["Multiplier"]}]];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  ,
  If[step[[1]], SMTStepBack[];, SMTArcLengthNext[];];
  SMTNextStep["Δγ" → step[[2]]]
 ];
```

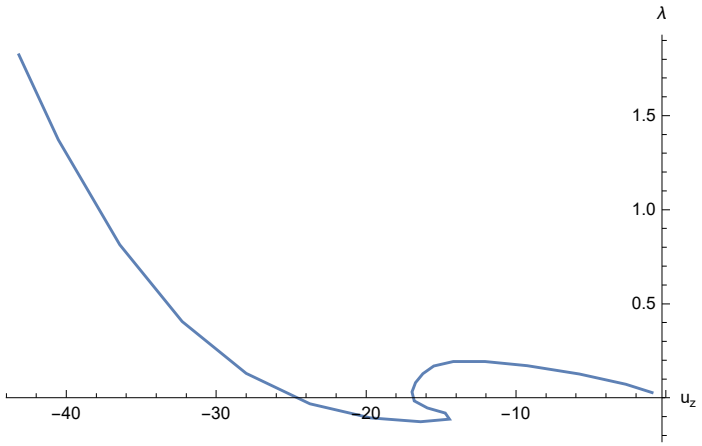
Finally, the arc-length analysis is concluded by command SMTArcLengthFree[].

In[53]:=
```
SMTArcLengthFree[];
```

Results

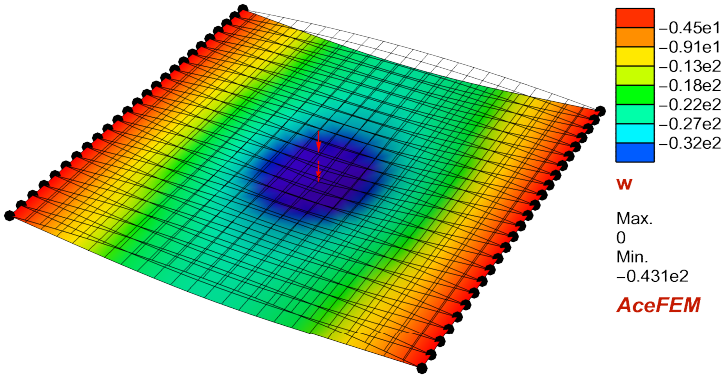
Equilibrium path $\lambda(u_z)$ of the isotropic shell:

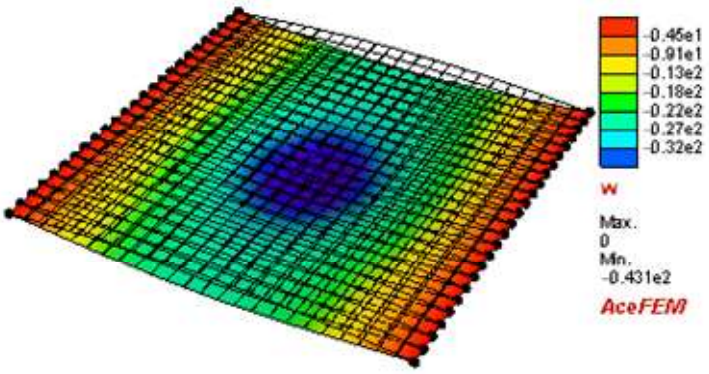
In[54]:=
```
ListLinePlot[graf, PlotRange → All, AxesLabel -> {"u_z", "λ"}]
```



The deformed mesh after procedure with field u_z is displayed:

In[55]:=
```
Show[SMTShowMesh["BoundaryConditions" → True, "DeformedMesh" → True, "Field" → "w"],
 SMTShowMesh["FillElements" → False, "Mesh" → Black, "BoundaryConditions" → True]]
```





Commands for Arc-length solution procedure

SMTArcLengthSet

SMTArcLengthSet[*options*]
initializes and sets the properties of the Arc-length iterative procedure

option	default	description
"Method"	"Crisfield"	"Crisfield" ⇒ standard arc length method with quadratic constraint (see [Stanic et al., 2016]) "OneDOFControl" ⇒ path-following method with the control one DOF constraint (see [Stanic et al., 2016]) "DissipationControl" ⇒ path-following method with constraint based on dissipation constraint (see [Stanic and Brank, 2017])
"λTarget"	False	λ ⇒ the value of BC multiplier for which the actual value of arc length of equilibrium path is estimated on a assumption that the problem is linear False ⇒ arc length of equilibrium path is estimated using the last calculated values of DOF and the last BC multiplier
"Report"	False	False ⇒ function returns an estimated arc length of equlibrium path for choosen BC multiplier True ⇒ a detailed report about the components of the constraint equation is returned
"PredictorSignCriterion"	"Feng"	Define criterion to determine the correct sign in the predictor phase of the Arc-length method "Feng" ⇒ the criterion proposed by Feng, Peric and Owen (see [de Souza Neto and Feng, 1999]) "DetK" ⇒ the criterion based on the sign of the determinant of current stiffness matrix (see [Crisfield, 1991]) . In this case it is required to set *"Solver"-> {5,-2}* within command *SMTAnalysis[ ]*.
"Ψ1"	Automatic	value of scaling parameter Ψ_1 in constraint equation
"Ψ2"	Automatic	value of scaling parameter Ψ_2 in constraint equation
"DirectionCheck"	10^{-8}	Direction is checked only after increment error is less than prescribed ($\|\Delta\mathbf{p}\|<err$). True ⇒ $err=10^{50}$,always check for false direction False ⇒ err=0., never check for false direction
"DirectionTolerance"	0.01	Direction is found to be false when cos ϕ<-1+tol, where ϕ is the angle between the increment vector in previous and the current step.

Options of the *SMTArcLengthSet* function.

[1] de Souza Neto E. A. , Feng Y. T. On the determination of the path direction for arc – length methods in the presence of bifurcations and ' snap – backs ', *Computer Methods in Applied Mechanics and Engineering,* 179 : 81 – 89 (1999) .

[2] Crisfield M. A. *Non –linear Finite Element Analysis of Solids and Structures*, Volume 1. Wiley, Chichester (1997).

- This sets scaling parameter Ψ_1 = 2 and turn off indicator of false direction arc-length analysis

In[235]:=
```
SMTArcLengthSet["λTarget" → 3, "Ψ1" → 2, "DirectionCheck" → False]
```

- This sets scaling parameter Ψ_2 = 1.5 and define “DetK” as the criterion in the predictor phase of arc-length analysis. It returns an estimated arc length of equlibrium path fot the value of BC multiplier 3 on a assumption that the problem is linear

In[27]:=
```
SMTAnalysis["Solver" → {5, -2}];
SMTArcLengthSet["Ψ2" → 1.5, "PredictorSignCriterion" → "DetK"];
```

SMTArcLengthIteration

SMTArcLengthIteration[]

executes one iteration of the Arc-length iterative procedure and returns the modified Euklid’s norm of the increment of global unknowns

option	default	description
"Method"	Automatic	Automatic ⇒ previously set method (see SMTArcLengthSet) "Crisfield" ⇒ standard arc length method with quadratic constraint (see [Stanic et al., 2016]) "OneDOFControl" ⇒ path-following method with the control one DOF constraint (see [Stanic et al., 2016]) "DissipationControl" ⇒ path-following method with constraint based on dissipation constraint (see [Stanic and Brank, 2017])

Options of the *SMTArcLengthIteration* function.

SMTArcLengthNext

SMTArcLengthNext[]
updates time dependent data (**app**≡**ap**).

SMTArcLengthNext[True]
updates time dependent data (**app**≡**ap**) and returns **app**

SMTArcLengthNext[*appInput*]
updates time dependent data (**app**≡*appInput*)

ap - global vector of degree of freedom at the end of previous time (load) step

app - global vector of degree of freedom at the end of the penultimate time (load) step

SMTArcLengthFree

SMTArcLengthFree[]
deallocates memory blocks. The arc-length analysis session is closed.

SMTArcLengthControlDOF

SMTArcLengthControlDOF[*setOfControlDOFs*]
setOfControlDOF is a list of integer numbers that represent the global indexes of DOFs included into constraint equation.

In case of standard arc length method (SMTArcLengthSet["Method"→"Crisfield"], see Theory), *setOfControlDOF* contains the indexes of diagonal elements that are set to 1 in the scaling matrix W. The remain diagonal elements in the scaling matrix W are set to zero.

In case of path-following method with adaptive one-DOF constraint (SMTArcLengthSet["Method"→"OneDOFControl"], see Theory). *setOfControlDOF* contains only global index of DOF that is controlled during the increment.

SMTArcLengthVectorGlobal

SMTArcLengthVectorGlobal[*RealVector*]
RealVector is a vector of real numbers. The command is used for importing a global nodal force vector into the constraint equation for path-following method with dissipation energy constraint (see Theory).

Arc-length examples

Description of the problem

Example presents an isotropic shell which is loaded by reference force $\mathbf{R}_{ref}$. Geometrical and material properties are taken from the article: *Sze K. Y., Liu X. H., Lo S. H., Popular Benchmark problems for geometric nonlinear analysis of shells, Finite Elements in Analysis and Design, 40: 1551-1569 (2004)*.

Shell arch with given width L_y, radius R and circle sector central angle θ is loaded with point load applied in the middle of arch. All nodes on edges at $x = -L_x$ and $x = L_x$ are constrained by pinned constraints.

See also `Implementation Notes for Arc - length solution procedure`.

```
In[202]:= << AceFEM`;
          ProblemDescription[] := (
            Ly = 508; R = 2540; θ = 0.1;
            H = R - R Cos[θ];
            Lx = R Sin[θ];
            raster = Table[{R Sin[θi], bi , R Cos[θi] - R + H}, {θi, -θ, θ, θ / 6}, {bi, 0, Ly, Ly}];
            SMTInputData[];
            SMTAddDomain[{"A", {"ML:", "SE", "MS", "S1", "ES", "HY", "ANSE4P6", {"D", "Fi"}, "SVenant"},
              {"E *" → 3102.75, "ν *" → 0.3, "t *" → 6.35, "μDrill *" → 1.}}];
            SMTAddMesh[Raster[raster], "A", "Division" -> {24, 24}];
            SMTAddEssentialBoundary[{
              {Line[{{-Lx, 0, 0}, {-Lx, Ly, 0}}, "D"], 1 → 0, 2 → 0, 3 → 0 },
              {Line[{{Lx, 0, 0}, {Lx, Ly, 0}}, "D"], 1 → 0, 2 → 0, 3 → 0}}];
            SMTAddNaturalBoundary[{Point[{0, Ly / 2, H}, "D"], 3 → -3000 }];
            SMTAnalysis[];
            graf = {};
            ResultsArticle =
             {{0, 0}, {-1.846`, 0.0517`}, {-5.271`, 0.1182`}, {-8.257`, 0.1583`}, {-10.799`, 0.1837`},
              {-11.904`, 0.1914`}, {-12.892`, 0.1953`}, {-13.752`, 0.195`}, {-14.472`, 0.1901`},
              {-15.05`, 0.1806`}, {-15.501`, 0.1671`}, {-16.145`, 0.1323`}, {-16.602`, 0.0923`},
              {-16.915`, 0.0504`}, {-17.008`, 0.0083`}, {-16.697`, -0.0312`}, {-15.78`, -0.0622`},
              {-15.206`, -0.0739`}, {-14.767`, -0.0861`}, {-14.52`, -0.1001`}, {-14.451`, -0.1142`},
              {-14.862`, -0.1247`}, {-15.778`, -0.1288`}, {-16.961`, -0.1271`}, {-18.32`, -0.1196`},
              {-19.817`, -0.1055`}, {-21.42`, -0.0825`}, {-23.1`, -0.0484`}, {-24.824`, -0.0006`},
              {-26.565`, 0.0626`}, {-28.302`, 0.1427`}, {-30.023`, 0.2403`}, {-31.72`, 0.3559`},
              {-33.388`, 0.4898`}, {-35.024`, 0.6417`}, {-36.626`, 0.8114`}, {-38.45`, 1.0313`}};
           )
          CollectResults[] := (
            AppendTo[graf, {SMTPostData["w", Point[{0, Ly / 2, H}]], SMTPostData["Sxx",
                Point[{0, Ly / 2, H}]], SMTRData["Multiplier"] , SMTRData["Parameter"]}];
           )
```

Arc-length procedure: solve for prescribed arc length

Description of the problem

For the description of the problem and necessary initializations first follow the hyperlink ArcLenghtExample and evaluate the initialization cell.

```
In[59]:= << AceFEM`;
         ProblemDescription[];
```

Arc-length procedure

Before arc-length procedure command **SMTArcLengthSet[]** must be used. The curve length is estimated for the target load level 3.

In[61]:=
```
sMax = SMTArcLengthSet["λTarget" → 3]
```

```
646.037
```

Terminal arc-length curve length *sMax*, maximum steps Δ*sMax,* minimum steps Δ*sMin* and initial increment of arc length *s0* are chosen:

In[62]:=
```
ΔsMax = sMax / 20;
s0 = sMax / 100;
ΔsMin = sMax / 1000;
```

Adaptive procedure with Arc-length iterations is used to solve described problem. Procedure is stopped when ΔsMax is reached. After each successful step, SMTArcLengthNext[] must be called.

In[65]:=
```
CollectResults[];
SMTNextStep["Δγ" → s0];
While[
  While[step = SMTConvergence[10^-10, 50, {"Adaptive γ", 16, ΔsMin, ΔsMax, sMax}],
   SMTArcLengthIteration[];];
  If[Not[step[[1]]], CollectResults[];];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  ,
  If[step[[1]], SMTStepBack[];, SMTArcLengthNext[];];
  SMTNextStep["Δγ" → step[[2]]]
 ];
SMTStatusReport[];
```

```
Step/Iter=22/5 λ/Δλ=1.82268/0.451257 γ/Δγ=646.037/21.0568
  ||Δp||/||R||=5.77574×10^-15/6.28132×10^-11 Events=0 Status=0/{Convergence}
```

After the analysis the SMTArcLengthFree[] must be called.

In[69]:=
```
SMTArcLengthFree[];
```

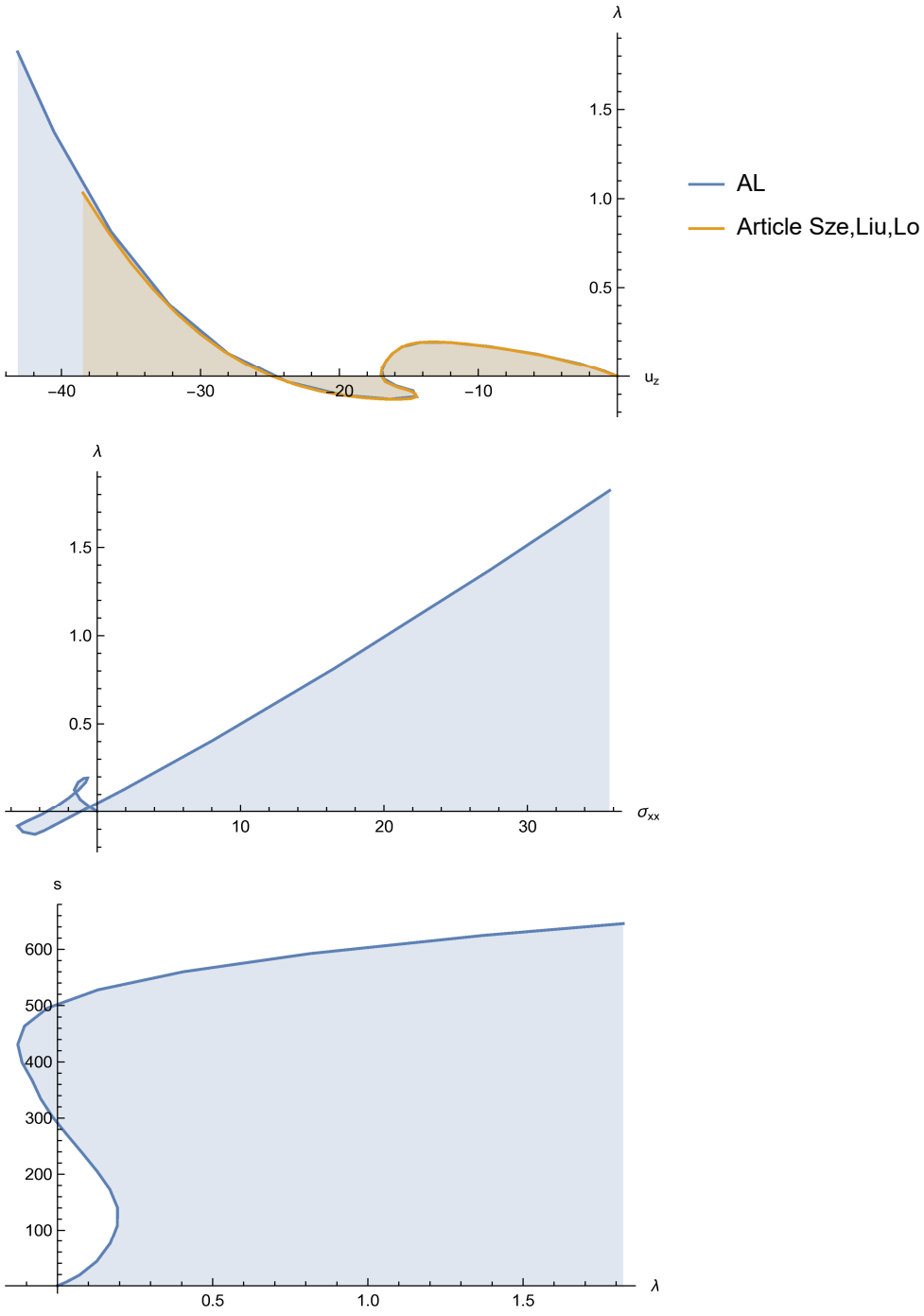
Analysis of the results

In[70]:=
```
grafAL = graf;
reportAL = SMTSimulationReport[];
```

No. of nodes	1250
No. of elements	576
No. of equations	3600
Number of threads used/max	8/8
Data memory (KBytes)	1871
Tangent matrix (KBytes)	744
Solver memory (KBytes)	9339
Total driver (KBytes)	11 954
MMA kernel memory (KBytes)	92 123
MMA front end memory(KBytes)	964 538
Total memory (KBytes)	1 068 615
Solver type	Pardiso
Matrix type	-2
No. of steps	22
No. of steps back	1
Step efficiency (%)	95.6522
Total no. of iterations	121
Average iterations/step	5.26087
Terminal BC multiplier (λ)	1.82268
Terminal time (t)	0.
Terminal parameter (γ)	646.037

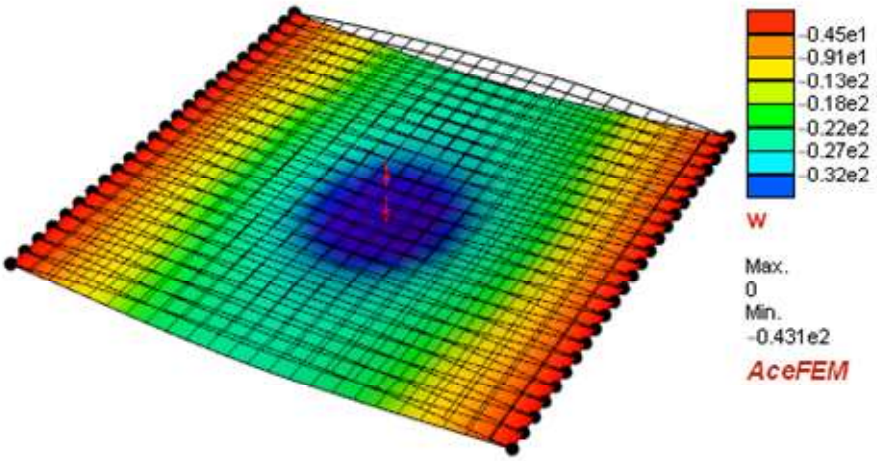
Total absolute time (s)	12.3722519
Total driver time (s)	12.186
Total driver time (%)	98.4946
Total linear solver time (s)	1.387
Total linear solver time (%)	11.2106
Total K&R time (s)	9.793
Total K&R time (%)	79.1529
Average time/iteration (s)	0.100711
Average linear solver time (s)	0.0114628
Average Ke&Re time (s)	0.00014051
CPU Mathematica time (s)	0.374
CPU Mathematica time (%)	3.02289

In[72]:=
```
ListLinePlot[{graf[[;; , {1, 3}]], ResultsArticle}, PlotLegends → {"AL", "Article Sze,Liu,Lo"},
 PlotRange → All, AxesLabel -> {"u_z", "λ"}, Filling → Axis]
ListLinePlot[graf[[;; , {2, 3}]], PlotRange → All, AxesLabel -> {"σ_xx", "λ"}, Filling → Axis]
ListLinePlot[graf[[;; , {3, 4}]], PlotRange → All, AxesLabel -> {"λ", "s"}, Filling → Axis]
```



The deformed mesh after procedure with field u_z is displayed:

In[75]:=
```
Show[SMTShowMesh["BoundaryConditions" → True, "DeformedMesh" → True, "Field" → "w"],
 SMTShowMesh["FillElements" → False, "Mesh" → Black, "BoundaryConditions" → True]]
```



Arc-length procedure: solve for target λ=1 by bisection

Description of the problem

For the description of the problem and necessary initializations follow the hyperlink ArcLenghtExample and evaluate the initialization cell.

In[76]:=
```
<< AceFEM`;
ProblemDescription[];
```

Arc-length procedure

Since multiplier cannot be directly controlled with Arc-Length procedure, option of SMTConvergence:

`"AlternativeTarget"→((undertarget=SMTRData["Multiplier"]<λtarget)&)`

can be used to terminate procedure when criterion `Abs[SMTRData["Multiplier"]-λtarget]/λtarget<λerror` is fulfilled.

In[78]:=
```
λtarget = 1;
λerror = 0.00005 ;
```

Before arc-length procedure command SMTArcLengthSet[]must be used. The curve length is estimated for the target load level 1.

In[80]:=
```
sEstimated = SMTArcLengthSet["λTarget" → 1]
```

```
215.346
```

Terminal arc-length curve length *sMax*, maximum steps Δ*sMax,* minimum steps Δ*sMin* and initial increment of arc length *s0* are chosen:

In[81]:=
```
sMax = Infinity;
ΔsMax = sEstimated / 4;
s0 = sEstimated / 100;
ΔsMin = 0;
```

Adaptive procedure with Arc-length iterations is used to solve described problem. Procedure is stopped when *ΔsMax* is reached. After each successful step, SMTArcLengthNext must be called.

```
In[85]:= CollectResults[];
         SMTNextStep["Δγ" → s0];
         While[
           While[step = SMTConvergence[10^-10, 50, {"Adaptive γ", 16, ΔsMin, ΔsMax, sMax},
               "AlternativeTarget" → (SMTRData["Multiplier"] < λtarget &)], SMTArcLengthIteration[];];
           If[Not[step[[1]]]
            , CollectResults[];
            If[Abs[SMTRData["Multiplier"] - λtarget] / λtarget < λerror
             , SMTStatusReport[{"Target λ: ", λtarget, " calculated λ: ", SMTRData["Multiplier"]}];
             step[[3]] = False;
            ];
           ];
           If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
           step[[3]]
           ,
           If[step[[1]], SMTStepBack[];, SMTArcLengthNext[];];
           SMTNextStep["Δγ" → step[[2]]]
          ];
```

```
Step/Iter=19/3 λ/Δλ=0.999965/0.0000518433 γ/Δγ=601.01/
 0.00312774 ||Δp||/||R||=4.20648×10^-15/5.91744×10^-11 Events=0 Status=
 0/{Convergence} Tag={{Target λ: , 1,  calculated λ: , 0.999965}}
```

```
In[88]:= SMTArcLengthFree[];
```

Analysis of the results

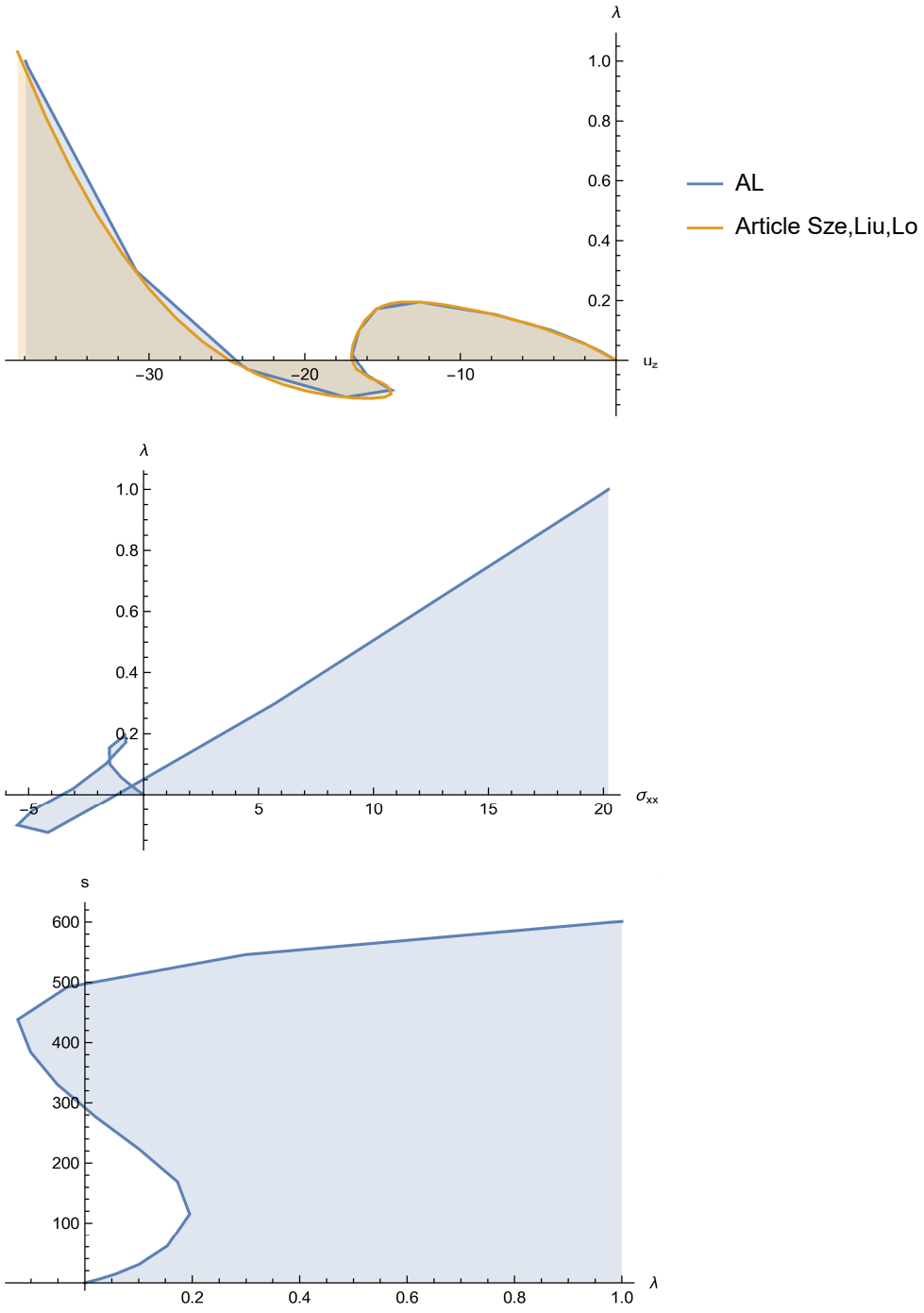
```
In[89]:= grafALB = graf;
         reportALB = SMTSimulationReport[];
```

No. of nodes	1250
No. of elements	576
No. of equations	3600
Number of threads used/max	8/8
Data memory (KBytes)	1871
Tangent matrix (KBytes)	744
Solver memory (KBytes)	9339
Total driver (KBytes)	11 954
MMA kernel memory (KBytes)	97 700
MMA front end memory(KBytes)	974 164
Total memory (KBytes)	1 083 818
Solver type	Pardiso
Matrix type	−2
No. of steps	19
No. of steps back	18
Step efficiency (%)	51.3514
Total no. of iterations	164
Average iterations/step	4.43243
Terminal BC multiplier (λ)	0.999965
Terminal time (t)	0.
Terminal parameter (γ)	601.01

Total absolute time (s)	16.3779248
Total driver time (s)	16.183
Total driver time (%)	98.8098
Total linear solver time (s)	1.868
Total linear solver time (%)	11.4056
Total K&R time (s)	13.342
Total K&R time (%)	81.4633
Average time/iteration (s)	0.0986768
Average linear solver time (s)	0.0113902
Average Ke&Re time (s)	0.000141239
CPU Mathematica time (s)	0.501
CPU Mathematica time (%)	3.059

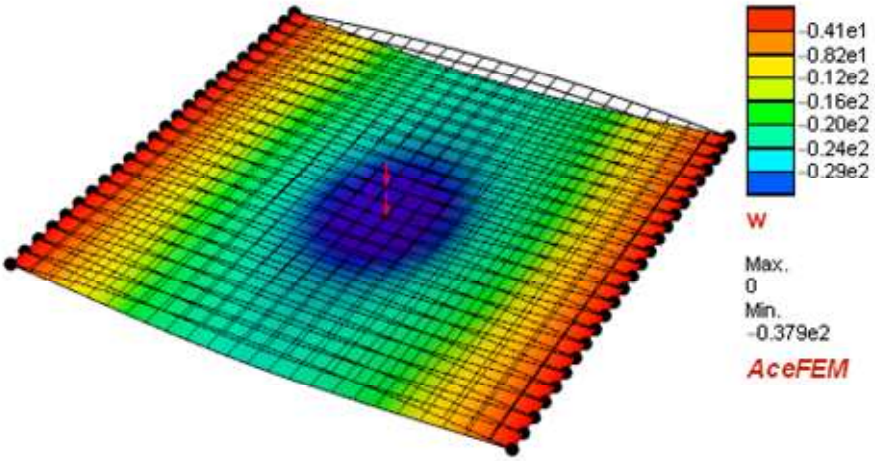
Here $\lambda - u_z$ graph is displayed:

In[91]:=
```
ListLinePlot[{graf[[;; , {1, 3}]], ResultsArticle}, PlotLegends → {"AL", "Article Sze,Liu,Lo"},
 PlotRange → All, AxesLabel -> {"u_z", "λ"}, Filling → Axis]
ListLinePlot[graf[[;; , {2, 3}]], PlotRange → All, AxesLabel -> {"σ_xx", "λ"}, Filling → Axis]
ListLinePlot[graf[[;; , {3, 4}]], PlotRange → All, AxesLabel -> {"λ", "s"}, Filling → Axis]
```



The deformed mesh after procedure with field u_z is displayed:

In[94]:=
```
Show[SMTShowMesh["BoundaryConditions" → True, "DeformedMesh" → True, "Field" → "w"],
 SMTShowMesh["FillElements" → False, "Mesh" → Black, "BoundaryConditions" → True]]
```



Arc-length procedure: solve for target λ=1 by interpolation

Description of the problem

For the description of the problem and necessary initializations follow the hyperlink ArcLenghtExample and evaluate the initialization cell.

In[95]:=
```
<< AceFEM`;
ProblemDescription[];
```

Arc-length procedure

Since multiplier is calculated quantity and it cannot be directly controlled with Arc-Length procedure, option of SMTConvergence

```
"AlternativeTarget"→((undertarget=SMTRData["Multiplier"]<λtarget)&)
```

can be used to terminate procedure when criterion

```
Abs[SMTRData["Multiplier"]-λtarget]/λtarget<λerror
```

is fulfilled.

In[97]:=
```
λtarget = 1;
λerror = 0.00005 ;
```

Before arc-length procedure command SMTArcLengthSet[]must be used. The curve length is estimated for the target load level 1.

In[99]:=
```
sEstimated = SMTArcLengthSet["λTarget" → 1]
```

Out[99]=
```
215.346
```

Terminal arc-length curve length *sMax*, maximum steps Δ*sMax,* minimum steps Δ*sMin* and initial increment of arc length *s0* are chosen:

In[100]:=
```
sMax = Infinity;
ΔsMax = sEstimated / 4;
s0 = sEstimated / 100;
ΔsMin = 0;
```

Adaptive procedure with Arc-length iterations is used to solve described problem. Procedure is stopped when *ΔsMax* is reached. After each successful step, SMTArcLengthNext must be called.

In[104]:=
```
CollectResults[];
SMTNextStep["Δγ" → s0];
interpolate = False;
While[
  While[step = SMTConvergence[10^-10, 50, {"Adaptive γ", 16, ΔsMin, ΔsMax, sMax},
     "AlternativeTarget" → (SMTRData["Multiplier"] < λtarget &)], SMTArcLengthIteration[];];
  interpolate = interpolate || Not[FreeQ[step, "AlternativeTarget"]];
  If[Not[step[[1]]]
   , CollectResults[];
   If[Abs[SMTRData["Multiplier"] - λtarget] / λtarget < λerror
    , SMTStatusReport[{"Target λ: ", λtarget, " calculated λ: ", SMTRData["Multiplier"]}];
    step[[3]] = False;
   ];
  ];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  ,
  If[interpolate
   , {γ, Δγ, λ, Δλ} =
    SMTRData[{"Parameter", "ParameterIncrement", "Multiplier", "MultiplierIncrement"}];
   Δs = Δγ (Δλ - λ + λtarget) / Δλ;
   , Δs = step[[2]]
  ];
  If[step[[1]], SMTStepBack[];, SMTArcLengthNext[];];
  SMTNextStep["Δγ" → Δs]
];
```

```
Step/Iter=17/3 λ/Δλ=0.999981/0.00417745 γ/Δγ=601.011
 /0.252308 ‖Δp‖/‖R‖=7.53745×10^-11/1.62748×10^-9 Events=0 Status=
 0/{Convergence} Tag={{Target λ: , 1,  calculated λ: , 0.999981}}
```

In[108]:=
```
SMTArcLengthFree[];
```

Analysis of the results

In[109]:=
```
grafALI = graf;
reportALI = SMTSimulationReport[];
```

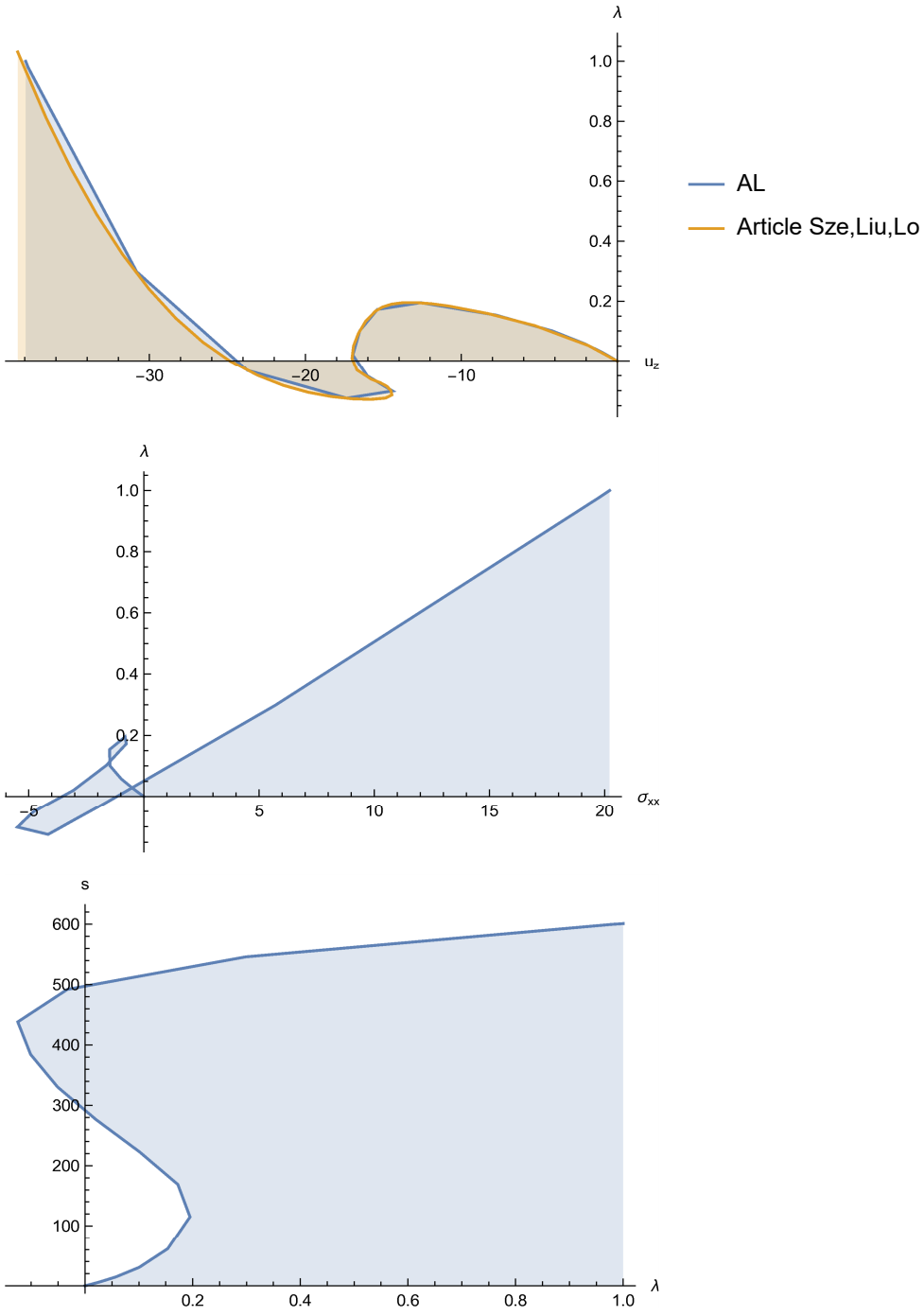
No. of nodes	1250
No. of elements	576
No. of equations	3600
Number of threads used/max	8/8
Data memory (KBytes)	1871
Tangent matrix (KBytes)	744
Solver memory (KBytes)	9339
Total driver (KBytes)	11 954
MMA kernel memory (KBytes)	97 700
MMA front end memory(KBytes)	979 456
Total memory (KBytes)	1 089 110
Solver type	Pardiso
Matrix type	−2
No. of steps	17
No. of steps back	3
Step efficiency (%)	85.
Total no. of iterations	102
Average iterations/step	5.1
Terminal BC multiplier (λ)	0.999981
Terminal time (t)	0.
Terminal parameter (γ)	601.011

Total absolute time (s)	10.8182149
Total driver time (s)	10.625
Total driver time (%)	98.214
Total linear solver time (s)	1.176
Total linear solver time (%)	10.8706
Total K&R time (s)	8.583
Total K&R time (%)	79.3384
Average time/iteration (s)	0.104167
Average linear solver time (s)	0.0115294
Average Ke&Re time (s)	0.000146089
CPU Mathematica time (s)	0.313
CPU Mathematica time (%)	2.89327

Here $\lambda - u_z$ graph is displayed:

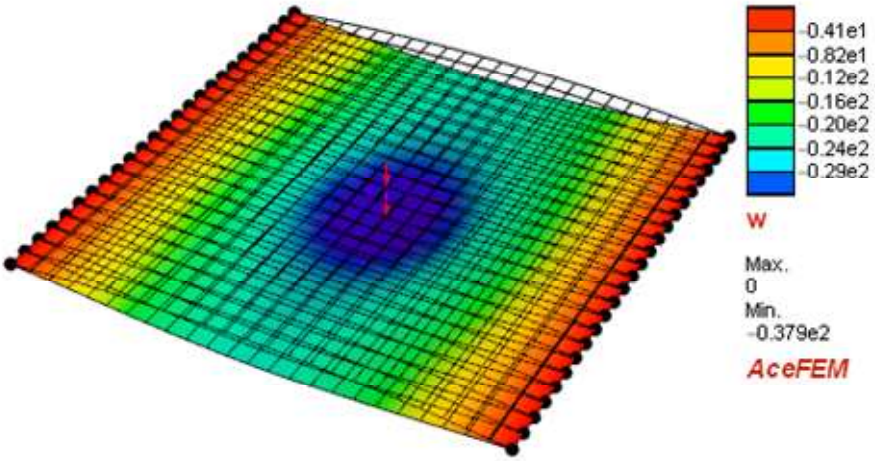
In[111]:=
```
ResultsArticle =
 {{0, 0}, {-1.846`, 0.0517`}, {-5.271`, 0.1182`}, {-8.257`, 0.1583`}, {-10.799`, 0.1837`},
  {-11.904`, 0.1914`}, {-12.892`, 0.1953`}, {-13.752`, 0.195`}, {-14.472`, 0.1901`},
  {-15.05`, 0.1806`}, {-15.501`, 0.1671`}, {-16.145`, 0.1323`}, {-16.602`, 0.0923`},
  {-16.915`, 0.0504`}, {-17.008`, 0.0083`}, {-16.697`, -0.0312`}, {-15.78`, -0.0622`},
  {-15.206`, -0.0739`}, {-14.767`, -0.0861`}, {-14.52`, -0.1001`}, {-14.451`, -0.1142`},
  {-14.862`, -0.1247`}, {-15.778`, -0.1288`}, {-16.961`, -0.1271`}, {-18.32`, -0.1196`},
  {-19.817`, -0.1055`}, {-21.42`, -0.0825`}, {-23.1`, -0.0484`}, {-24.824`, -0.0006`},
  {-26.565`, 0.0626`}, {-28.302`, 0.1427`}, {-30.023`, 0.2403`}, {-31.72`, 0.3559`},
  {-33.388`, 0.4898`}, {-35.024`, 0.6417`}, {-36.626`, 0.8114`}, {-38.45`, 1.0313`}};
```

In[112]:=
```
ListLinePlot[{graf[[;;, {1, 3}]], ResultsArticle}, PlotLegends → {"AL", "Article Sze,Liu,Lo"},
 PlotRange → All, AxesLabel -> {"u_z", "λ"}, Filling → Axis]
ListLinePlot[graf[[;;, {2, 3}]], PlotRange → All, AxesLabel -> {"σ_xx", "λ"}, Filling → Axis]
ListLinePlot[graf[[;;, {3, 4}]], PlotRange → All, AxesLabel -> {"λ", "s"}, Filling → Axis]
```



The deformed mesh after procedure with field u_z is displayed:

In[115]:=
```
Show[SMTShowMesh["BoundaryConditions" → True, "DeformedMesh" → True, "Field" → "w"],
 SMTShowMesh["FillElements" → False, "Mesh" → Black, "BoundaryConditions" → True]]
```



Standard Newton Raphson procedure for comparison

Description of the problem

For the description of the problem and necessary initializations first follow the hyperlink ArcLenghtExample and evaluate the initialization cell.

```
In[116]:= << AceFEM`;
          ProblemDescription[];
```

Newton Raphson procedure

Adaptive BC procedure with Newton-Raphson iterations is used to solve described problem. Procedure is stopped when λ reaches λMax or NR files to converge. In the present case NR procedure fails to converge in the vicinity of the limit point.

In[118]:=
```
λMax = 1;
λ0 = λMax / 100;
ΔλMax = λMax / 10;
ΔλMin = λ0 / 1000;
CollectResults[];
SMTNextStep["λ" → λ0];
While[
  While[step = SMTConvergence[10^-10, 50, {"Adaptive BC", 16, ΔλMin, ΔλMax, λMax}],
   SMTNewtonIteration[];];
  If[Not[step[[1]]], CollectResults[]];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  ,
  If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" → step[[2]]]
 ];
```

Iterative solution procedure has failed.

Error is alternating. Increment of parameter is below the minimum prescribed.

Step/Iter=9/8 λ/Δλ=0.196217/0.0000182311 ‖Δp‖/‖R‖=0.0158838/0.00736999 Events=0 Status=0

Tot.Step	Iter	|a|	|Ψ|	Error	Neg.Piv.	λ	Δλ
25	2	0.12062	0.0024089	0	2	0.196217	0.0000182311
25	3	0.0586375	0.156318	0	1	0.196217	0.0000182311
25	4	0.0317984	0.037856	0	1	0.196217	0.0000182311
25	5	0.0182781	0.0109309	0	1	0.196217	0.0000182311
25	6	0.0157018	0.0035678	0	1	0.196217	0.0000182311
25	7	0.0263755	0.00261395	0	2	0.196217	0.0000182311
25	8	0.0158838	0.00736999	0	1	0.196217	0.0000182311

Analysis of the results

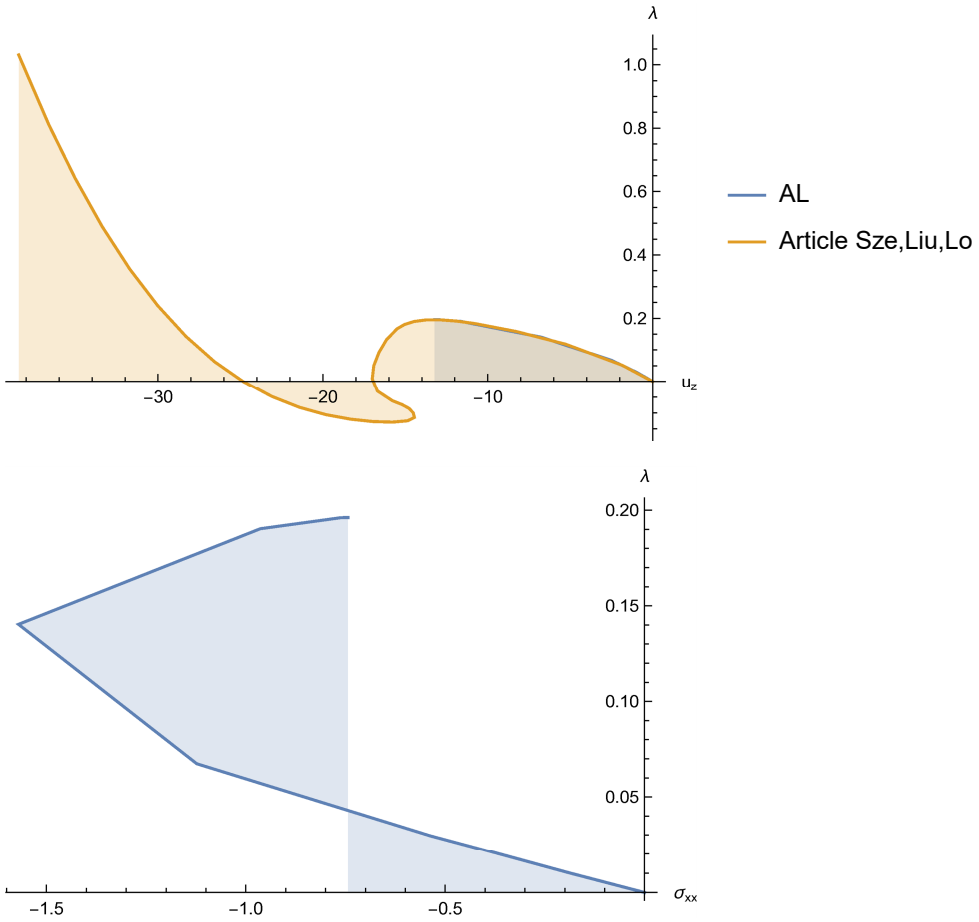
In[125]:=
```
grafNR = graf;
reportNR = SMTSimulationReport[];
```

No. of nodes	1250
No. of elements	576
No. of equations	3600
Number of threads used/max	8/8
Data memory (KBytes)	1871
Tangent matrix (KBytes)	744
Solver memory (KBytes)	9339
Total driver (KBytes)	11954
MMA kernel memory (KBytes)	97700
MMA front end memory(KBytes)	984412
Total memory (KBytes)	1094066
Solver type	Pardiso
Matrix type	-2
No. of steps	8
No. of steps back	17
Step efficiency (%)	32.
Total no. of iterations	187
Average iterations/step	7.48
Terminal BC multiplier (λ)	0.196199
Terminal time (t)	0.
Terminal parameter (γ)	0.

Total absolute time (s)	18.1360975
Total driver time (s)	17.967
Total driver time (%)	99.0676
Total linear solver time (s)	1.604
Total linear solver time (%)	8.84423
Total K&R time (s)	15.947
Total K&R time (%)	87.9296
Average time/iteration (s)	0.0960802
Average linear solver time (s)	0.00857753
Average Ke&Re time (s)	0.000148052
CPU Mathematica time (s)	0.594
CPU Mathematica time (%)	3.27524

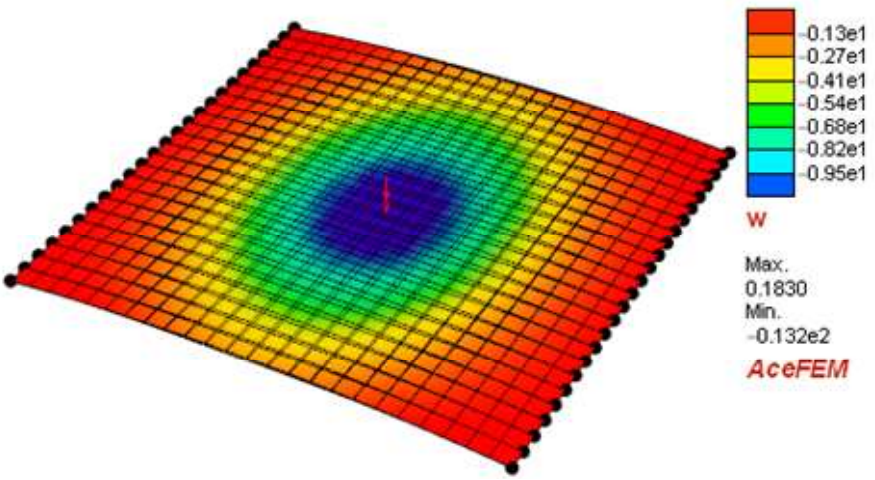
Here $\lambda - u_z$ graph is displayed:

In[127]:=
```
ListLinePlot[{graf[[;;, {1, 3}]], ResultsArticle}, PlotLegends → {"AL", "Article Sze,Liu,Lo"},
 PlotRange → All, AxesLabel -> {"u_z", "λ"}, Filling → Axis]
ListLinePlot[graf[[;;, {2, 3}]], PlotRange → All, AxesLabel -> {"σ_xx", "λ"}, Filling → Axis]
```



The deformed mesh after procedure with field u_z is displayed:

In[129]:=
```
Show[SMTShowMesh["BoundaryConditions" → True, "DeformedMesh" → True, "Field" → "w"],
 SMTShowMesh["FillElements" → False, "Mesh" → Black, "BoundaryConditions" → True]]
```

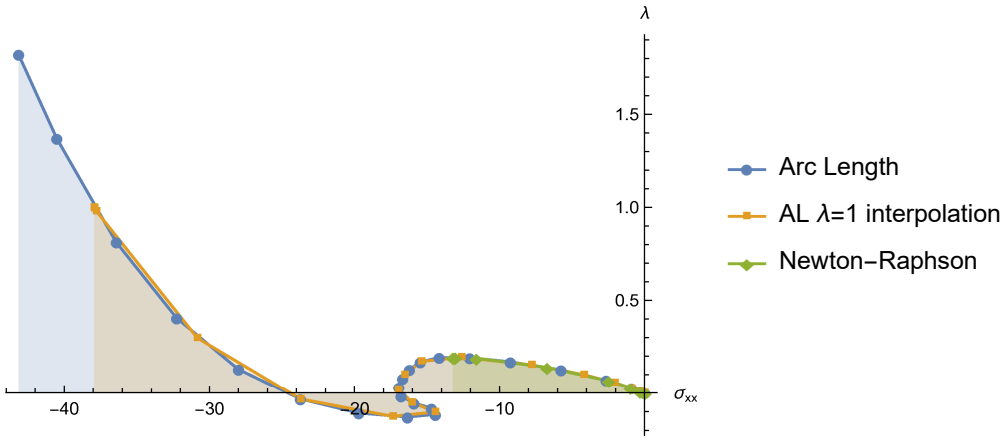


Comparison

In[130]:=
```
Join[{{"Data", "Newton-Raphson", "ArcLength", "AL λ=1 bisection", "AL λ=1 interpolation"}},
 {reportALB[[;;, 1]], reportNR[[;;, 2]], reportAL[[;;, 2]],
  reportALB[[;;, 2]], reportALI[[;;, 2]]}^T] // TableForm
```

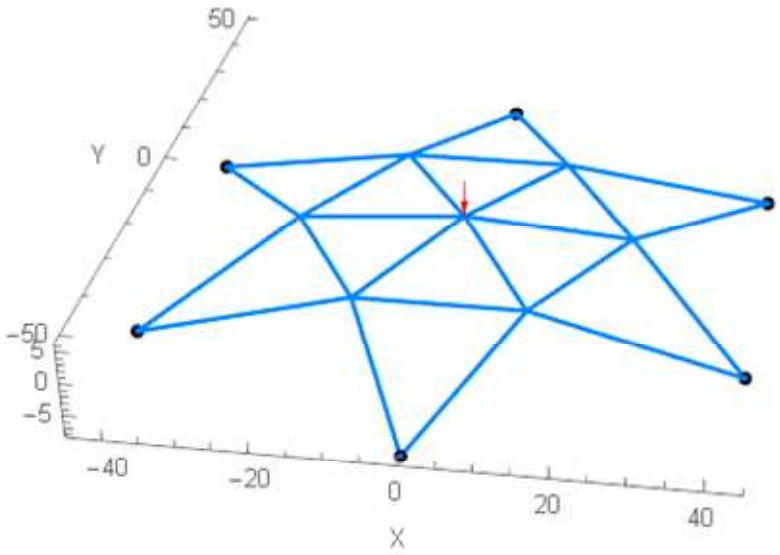
Data	Newton-Raphson	ArcLength	AL λ=1 bisection	Al
No. of nodes	1250	1250	1250	12
No. of elements	576	576	576	57
No. of equations	3600	3600	3600	36
Number of threads used/max	8/8	8/8	8/8	8/
Data memory (KBytes)	1871	1871	1871	18
Tangent matrix (KBytes)	744	744	744	74
Solver memory (KBytes)	9339	9339	9339	93
Total driver (KBytes)	11954	11954	11954	11
MMA kernel memory (KBytes)	97700	92123	97700	97
MMA front end memory (KBytes)	984412	964538	974164	97
Total memory (KBytes)	1094066	1068615	1083818	1(
Solver type	Pardiso	Pardiso	Pardiso	Pa
Matrix type	-2	-2	-2	-
No. of steps	8	22	19	17
No. of steps back	17	1	18	3
Step efficiency (%)	32.	95.6522	51.3514	85
Total no. of iterations	187	121	164	16
Average iterations/step	7.48	5.26087	4.43243	5.
Terminal BC multiplier (λ)	0.196199	1.82268	0.999965	0.
Terminal time (t)	0.	0.	0.	0.
Terminal parameter (γ)	0.	646.037	601.01	66
Total absolute time (s)	18.1360975	12.3722519	16.3779248	16
Total driver time (s)	17.967	12.186	16.183	16
Total driver time (%)	99.0676	98.4946	98.8098	98
Total linear solver time (s)	1.604	1.387	1.868	1.
Total linear solver time (%)	8.84423	11.2106	11.4056	16
Total K&R time (s)	15.947	9.793	13.342	8.
Total K&R time (%)	87.9296	79.1529	81.4633	79
Average time/iteration (s)	0.0960802	0.100711	0.0986768	0.
Average linear solver time (s)	0.00857753	0.0114628	0.0113902	0.
Average Ke&Re time (s)	0.000148052	0.00014051	0.000141239	0.
CPU Mathematica time (s)	0.594	0.374	0.501	0.
CPU Mathematica time (%)	3.27524	3.02289	3.059	2.

In[131]:=
```
ListLinePlot[{grafAL[[;;, {1, 3}]], grafALI[[;;, {1, 3}]], grafNR[[;;, {1, 3}]]},
 PlotLegends → {"Arc Length", "AL λ=1 interpolation", "Newton-Raphson"},
 PlotRange → All, AxesLabel -> {"σxx", "λ"}, Filling → Axis, PlotMarkers → Automatic]
```



Fundamental path of star dome truss example

Star dome truss example is the standard benchmark case for the assessment of the performance of the path following algorithms. Combination of the arc-length commands and the SMTAnimationOfResponse command is used here to get the fundamental path of the response curve. Stable segments of the curve are shown in blue color and unstable segments in red color. The algorithm cannot automatically follow the alternative branched of the response curve after the bifurcation. In the next chapter a more advanced procedure is presented that can follow all the branched of the response curve (see).



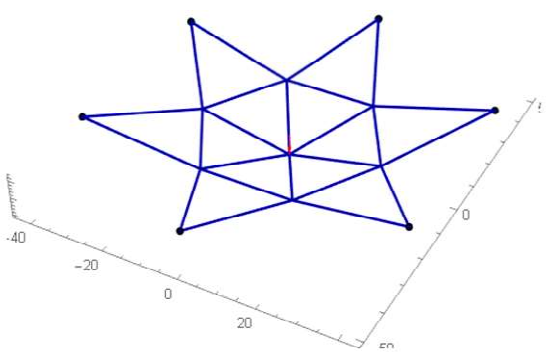
In[2]:=
```
<< AceFEM`;
```

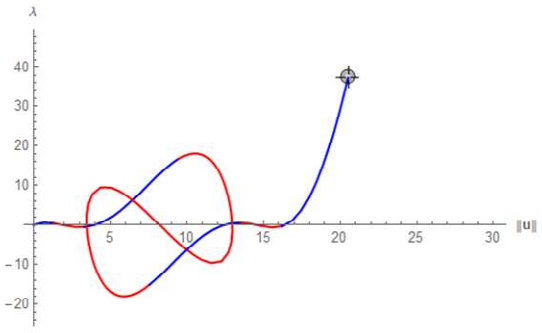
In[3]:=
```
Acs = 0.1; (*cm^2*)
Em = 2.034 * 10^7 * 10^-3; (*kN/cm^2*)
PLoad = -1; a = 43.3; b = 25.;
c = 8.216; d = 2.; e = a / Cos[π / 6];
nodes = {{1, 0., 0., 0.}, {2, b Cos[π / 3], b Sin[π / 3], -d}, {3, b, 0, -d},
    {4, b Cos[π / 3], -b Sin[π / 3], -d}, {5, -b Cos[π / 3], -b Sin[π / 3], -d},
    {6, -b, 0, -d}, {7, -b Cos[π / 3], b Sin[π / 3], -d}, {8, 0, e, -c},
    {9, a, e Sin[π / 6], -c}, {10, a, -e Sin[π / 6], -c}, {11, 0, -e, -c},
    {12, -a, -e Sin[π / 6], -c}, {13, -a, e Sin[π / 6], -c}};
elements = {{1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}, {1, 7}, {2, 7}, {2, 3},
    {3, 4}, {4, 5}, {5, 6}, {6, 7}, {7, 8}, {2, 8}, {2, 9}, {3, 9}, {3, 10},
    {4, 10}, {4, 11}, {5, 11}, {5, 12}, {6, 12}, {6, 13}, {7, 13}};
SMTInputData[];
SMTAddDomain["A", "ExamplesNonlinearTruss", {"E - elastic modulus" → Em, "A - area" → Acs}];
SMTAddMesh["A", nodes, elements];
SMTAddEssentialBoundary["Z" == -c &, 1 → 0, 2 → 0, 3 → 0];
SMTAddNaturalBoundary[Point[{0, 0, 0}], 3 → PLoad];
SMTAnalysis[];
```

In[15]:=
```
sMax = 120 SMTArcLengthSet["λTarget" → 1];
ΔsMax = sMax / 100;
ΔsMin = sMax / 1000;
```

In[18]:=
```
SMTNextStep["Δγ" → ΔsMax];
While[
  While[step = SMTConvergence[10^-8, 15, {"Adaptive γ", 10, ΔsMin, ΔsMax, sMax}],
   SMTArcLengthIteration[];
  ];
  If[step[[4]] === "MinBound", SMTStatusReport["Error: Δγ < Δγmin"]; Abort[];];
  If[Not[step[[1]]]
   , SMTAnimationOfResponse[
     "LeadingNodePosition" → {0, 0, 0}, "ImageSize" → 400, "PlotMarkers" → False];
  ];
  step[[3]]
  , If[step[[1]], SMTStepBack[];, SMTArcLengthNext[];];
  SMTNextStep["Δγ" → step[[2]]];
 ];
```

In[20]:=
```
SMTAnimationOfResponse["Last frame"]
```





User Defined Tasks

Contents

Introduction

The standard tasks performed by the finite element environment are evaluation of the global residual and tangent matrix, solution of the resulting system of linear equations and post-processing of the results. Additionally to the standard tasks, user can also define and execute user defined tasks. User defined tasks can be:

- associated with the element,
- globally defined user tasks.

User Tasks Associated With the Element

The process of defining and executing the user defined tasks associated with the element is composed of:

- definition of the standard user subroutine "Tasks" as a part of element definition with AceGen (see also Standard User Subroutines);
- generation of the element source file;
- definition of the AceFEM input data;
- at any point of the analysis the user defined tasks can be executed by the SMTTask command.

Task is identified by the task identification keyword *taskID*. The *taskID* keyword has to appear as an element of the vector of character type element switches *es$$["CharSwitch",i]* (see Domain Specification Data) for all elements that support given task.

Standard user subroutine: Tasks

The "Tasks" standard user element subroutine can be used to perform various tasks that requires the assembly of the results over the complete finite element mesh or over the part of the mesh. The structure and the standard input/output arguments of the standard user subroutines is described in Standard User Subroutines. "Tasks" user subroutine has a number of additional input/output arguments that are interpreted accordingly to the type of the task.

task type	*task description*	active output parameters
1	given integer and real output vectors are evaluated and optionally summarized (see SMTTask command) for selected elements	IntegerOutput$$,RealOutput$$
2	the tasks expects the values of the continuous field to be given for all element nodes and stored in the RealOutput$$ vector (the defined continuous field can then be smoothed and extrapolated to the given spatial point, etc.. , depending on the options given to the SMTTask command)	RealOutput$$
3	the tasks expects the values of the continuous field to be given for all element integration points and stored in the RealOutput$$ vector (the defined continuous field can then be smoothed, extrapolated to the given spatial point, etc.. , depending on the options given to the SMTTask command)	RealOutput$$
4	given local element vectors are assembled accordingly to the standard finite element assembly procedure and given integer and real output vectors are summarized for selected elements	IntegerOutput$$,RealOutput$$,p$$
5	given local element matrices are assembled accordingly to the standard finite element assembly procedure and given integer and real output vectors are summarized for selected elements	IntegerOutput$$,RealOutput$$,s$$
6	types 4 and 5 combined	IntegerOutput$$,RealOutput$$,p$$,s$$

Types of the tasks to be performed.

argument	*Description*	*Type*
Task$$	Task$$ is an input argument that should be interpreted as follows: -n ⇒ initialize the task (set TasksData$$) with task identification keyword *taskID* where *n* is position of *taskID* within the vector of character type element switches, thus *es$$["CharSwitch",n]==taskID* *n* ⇒ execute the task that corresponds to task *es$$["CharSwitch",n]==taskID*	Integer
TasksData$$[5]	TasksData$$[1] ⇒ task type (see table above) TasksData$$[2] ⇒ the length of the IntegerInput$$ vector TasksData$$[3] ⇒ the length of the RealInput$$ vector TasksData$$[4] ⇒ the length of the IntegerOutput$$ vector TasksData$$[5] ⇒ the length of the RealOutput$$ vector	Real
IntegerInput$$[TasksData$$[2]]	integer input vector (the same for all elements)	Integer
RealInput$$[TasksData$$[3]]	real input vector (the same for all elements)	Real
IntegerOutput$$[TasksData$$[4]]	integer output vector (the values returned from all selected elements are further processed accordingly to the options given to SMTTask command)	Integer
RealOutput$$[TasksData$$[5]]	real output vector (the values returned from all selected elements are further processed accordingly to the options given to SMTTask command)	Real

Additional arguments of the "Tasks" subroutine.

form	I/O parameter
SMSIO["Task index"]	Task$$
SMSIO["Task data"[*index*]]	TasksData$$[*index*]
SMSIO["Task real input"[*index*]]	RealInput$$[*index*]
SMSIO["Task integer input"[*index*]]	IntegerInput$$[*index*]
SMSIO[*value*, "Add to" or "Export to", "Task data"[*index*]]	TasksData$$[*index*]
SMSIO[*value*, "Add to" or "Export to", "Task real output"[*index*]]	RealOutput$$[*index*]
SMSIO[*value*, "Add to" or "Export to", "Task integer output"[*index*]]	IntegerOutput$$[*index*]
SMSIO[*value*, "Add to" or "Export to", "Task local vector"[*index*]]	p$$[*index*]
SMSIO[*value*, "Add to" or "Export to", "Task local matrix"[*indexi,indexj*]]	s$$[*indexi,indexj*]

SMSIO commands for I/O of data related to tasks.

Template of the Tasks user subroutine

Initialization

The SMSCharSwitch constant holds the identifications of the tasks.

```
SMSTemplate[
 ..., "SMSCharSwitch" →
  {"TaskType1", "TaskType2", "TaskType3", "TaskType4", "TaskType5", "TaskType6"}, ...
]
...
SMSStandardModule["Tasks"];
task ⊨ SMSIO["Task index"];
...
```

Task type 1

Initialization and execution of the type 1 task with the task identification "TaskType1" that will return 1 integer and 3 real values.

```
SMSIf[task == -1, SMSIO[{1, 0, 0, 1, 3}, "Export to", "Task data"];
  SMSReturn[];];
SMSIf[task == 1
  , SMSIO[ival, "Export to", "Task integer output"[1]];
  SMSIO[{rval1, rval2, rval3}, "Export to", "Task real output"[{1, 2, 3}]];
 ];
```

Task type 2

Initialization and execution of the type 2 task with the task identification "TaskType2" that will return SMTNoNodes real type values

```
SMSIf[task == -2, SMSIO[{2, 0, 0, 0, SMSNoNodes}, "Export to", "Task data"];
  SMSReturn[];];
SMSIf[task == 2
  ,
  SMSIO[{val2, val2, ..., valSMTNoNodes}, "Export to", "Task real output"[{1, 2, ... SMTNoNodes}]];
 ];
```

Task type 3

Initialization and execution of the type 3 task with the task identification "TaskType3" that will return the number of integration points real type values

```
SMSIf[task == -3, SMSIO[{3, 0, 0, 0, SMSIO["No. integration points"]}, "Export to", "Task data"];
  SMSReturn[];];
SMSIf[task == 3
  ,
  SMSIO[{val2, val2, ..., valSMTNoNodes}, "Export to", "Task real output"[{1, 2, ... NoIntPoints}]];
 ];
```

Task type 4

Initialization and execution of the type 4 task with the task identification "TaskType4" that will set the local element vector p$$. The local element vectors are assembled to form a global vector that is result of the SMTTask["TaskType4"] command.

```
SMSIf[task == -4, SMSIO[{4, 0, 0, 0, 0}, "Export to", "Task data"];
  SMSReturn[];];
SMSIf[task == 4
  , SMSIO[{val2, val2, ..., valSMTNoDofGlobal}, "Export to", "Task local vector"];
 ];
```

Task type 5

Initialization and execution of the type 5 task with the task identification "TaskType5" that will set the local element matrix s$$. The local element matrices are assembled to form a global matrix that is result of the SMTTask["TaskType5"] command.

```
SMSIf[task == -5, SMSIO[{5, 0, 0, 0, 0}, "Export to", "Task data"];
  SMSReturn[];];
SMSIf[task == 5
  , SMSIO[localmatrix, "Export to", "Task local matrix"];
 ];
```

Task type 6

Initialization and execution of the type 6 task with the task identification "TaskType6" that will set the local element matrix s$$ and the local element vector p$$. The local quantities are assembled to form a global quantities that are result of the SMTTask["TaskType6"] command.

```
SMSIf[task == -6, SMSIO[{6, 0, 0, 0, 0}, "Export to", "Task data"];
  SMSReturn[];];
SMSIf[task == 6
  , SMSIO[{val2, val2, ..., valSMTNoDofGlobal}, "Export to", "Task local vector"];
  SMSIO[localmatrix, "Export to", "Task local matrix"];
 ];
```

SMTTask

SMTTask[*taskID*]
execute task identified by keyword *taskID*

option	default	description
"IntegerInput"-> {i1,i2,...}	{}	vector of integer values; The length of the vector is specified by the TasksData$$[2] constant in the user subroutine "Tasks" (see Standard User Subroutines).
"RealInput"-> {r1,r2,...}	{}	vector of real values; The length of the vector is specified by the TasksData$$[3] constant in the user subroutine "Tasks".
"Elements"-> *elementSelector*	All	the user subroutine "Tasks" is called only for elements selected by *elementSelector* (see `Selecting Elements`)
"Point"->{x,y,z}	False	the user subroutine "Tasks" is called only for the patch of elements that surrounds the given spatial point and the results are then extrapolated into the given point (only valid for the task types 2 and 3). The "Point" option can be combined with the "Elements" option in the case of multi-field problems to evaluate only elements that are used to discretize specific field at the given point.
"Tolerance"	10^{-10}	the numbers smaller in absolute magnitude than *"Tolerance"* are replaced by 0 within the search procedures
"Summation"	True	True ⇒ the resulting integer and real output vectors are summarized False ⇒ result is a list of resulting integer and real output vectors for all or selected elements
"Averaging"	True	True ⇒ for tasks 2 and 3 the resulting integer and real output vectors are summarized and averaged to all or selected nodes False ⇒ the resulting integer and real output vectors are only summarized

Options of the SMTTask function.

The user defined tasks can be used to perform various tasks that require the assembly of the results over a complete finite element mesh or over a part of the mesh. The type of the task is defined at the code generation phase (see `Standard User Subroutines`) and cannot be changed later. The return value of the SMTTask command depends on the type of the task and the additional options described above.The complete list of the possible output parameters is *{IntegerOutput, RealOutput, VectorGlobal, MatrixGlobal}*, however the output parameters that are not actually used are not returned as the results of the *SMSTask* command. If the number of *IntegerOutput* or *RealOutput* values is 1 then a scalar is returned rather than a vector. The actual execution of the tasks type 2 and 3 depends on the value of the “Point” option. If the spatial point is not given then the user subroutine is called for all elements, the resulting continuous field is smoothed and evaluated for all nodes of the mesh and the value of the field for all nodes is then returned as the result of the task. In the case that spatial point is specified then the user subroutine is called only for the patch of the elements that surrounds the given spatial point, the resulting continuous field is then extrapolated to the given spatial point and the value of the field in the given point is then returned as the result of the task.

task type and options	*task description*	return values
type=1	The user subroutine is called for all elements and resulting integer and real output vectors are returned for all elements as the result of the task.	{{IntegerOutput1, RealOutput1},... for all elements}
type=1 "Summation"->True	The user subroutine is called for all elements and resulting integer and real output vectors are summarized.	{IntegerOutput, RealOutput}
type=1 "Elements"-> *elementSelector*	The user subroutine is called for selected elements (see `Selecting Elements`) and resulting integer and real output vectors are returned	{{IntegerOutput1, RealOutput1},...,for all selected elements}
type=1 "Summation"->True "Elements"-> *elementSelector*	The user subroutine is called for selected elements and resulting integer and real output vectors are summarized.	{IntegerOutput, RealOutput}
type=2	The user subroutine is called for all elements.The resulting vectors of element nodal values are then smoothed at the global level and they define a global continuous scalar field.	{v1, v2,... for all nodes}
type=2 "Point"->{x,y,z}	The user subroutine is called for a patch of elements that surround the given spatial point. The resulting vectors of element nodal values are then smoothed at the global level. The resulting continuous field is then extrapolated to the given spatial point.	v
type=3	The user subroutine is called for all elements. The resulting vectors of values defined for each element integration point are then smoothed at the global level and they define a global continuous scalar field.	{v1, v2,... for all nodes}
type=3 "Point"->{x,y,z}	The user subroutine is called for a patch of elements that surround the given spatial point. The resulting vectors of values defined for each element integration point are then smoothed at the global level. The resulting continuous scalar field is then extrapolated to the given spatial point.	v
type=4	The user subroutine is called for all elements and assembled global vector is returned together with summarized integer and real output vectors.	{IntegerOutput, RealOutput, VectorGlobal}
type=5	The user subroutine is called for all elements and assembled global matrix is returned together with summarized integer and real output vectors.	{IntegerOutput, RealOutput, MatrixGlobal}
type=6	The user subroutine is called for all elements and assembled global vector and global matrix are returned together with summarized integer and real output vectors.	{IntegerOutput, RealOutput, VectorGlobal, MatrixGlobal}

Return values for all tasks.

Example: Evaluating Mass and mesh distortion of arbitrary quadrilateral 2D mesh

Generation of the element source code

Create an 2D quadrilateral element that would perform two tasks:

a) Calculate the mass $m = \int_A \rho \, dA$ where A is a complete mesh or a part of the mesh and ρ is a density.

b) Calculate the mesh distortion in an arbitrary point of the mesh. Mesh distortion is defined by

$d = \frac{|\text{ArcTang}[\mathbf{r}_{,\eta}] - \text{ArcTang}[\mathbf{r}_{,\xi}] - \pi/2|}{\pi/2}$ where $\{\xi,\eta\}$ are the reference coordinates of the element.

The first task returns a real type scalar value and is, accordingly to the definitions defined in User Defined Tasks, "type 1" task. The second task calculates the distortion in a integration points of the element and is a "type 3" task.

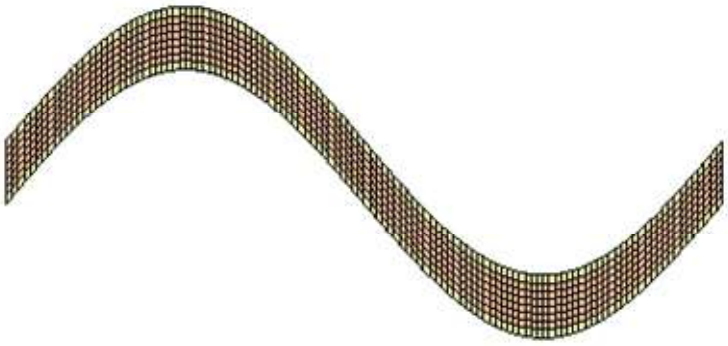
In[15]:=
```
<< "AceGen`";
SMSInitialize["ExamplesTasks2D", "Environment" -> "AceFEM"];
SMSTemplate["SMSTopology" → "Q1",
  "SMSCharSwitch" → {"Mass", "MeshDistortion"},
  "SMSDomainDataNames" -> {"ρ -density", "t -thickness"},
  "SMSDefaultData" -> {1, 1}];
SMSStandardModule["Tasks"];
task ⊨ SMSIO["Task index"];

SMSIf[task < 0

 , SMSSwitch[task
   , -1,
   SMSIO[{1, 0, 0, 0, 1}, "Export to", "Task data"];
   , -2,
   SMSIO[{3, 0, 0, 0, SMSIO["No. integration points"]}, "Export to", "Task data"];
  ];

 , {ρ, tξ} ⊨ SMSIO["Domain data"];
 SMSDo[
  Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
  XIO ⊨ SMSIO["Nodal coordinates"];
  Nh ⊨ 1 / 4 {(1 - ξ) (1 - η), (1 + ξ) (1 - η), (1 + ξ) (1 + η), (1 - ξ) (1 + η)};
  SMSFreeze[X, Nh.XIO];
  Je ⊨ SMSD[X, {ξ, η}]; Jed ⊨ Det[Je];
  SMSSwitch[task
   , 1,
   SMSIO[Jed tξ ρ, "Add to", "Task real output"[1]];
   , 2,
   rξ ⊨ SMSD[X, ξ]; rη ⊨ SMSD[X, η];
   dist ⊨ SMSAbs[(ArcTan @@ rη) - (ArcTan @@ rξ) - (π / 2)] / (π / 2);
   SMSIO[dist, "Export to", "Task real output"[Ig]];
  ];
  , {Ig, 1, SMSIO["No. integration points"]}
 ];

]
SMSWrite[];
```

```
File: ExamplesTasks2D.c  Size: 5553  Time: 1
```

Method	Tasks
No.Formulae	41
No.Leafs	486

Sinusoidal double skin cladding

Find the mass and mesh distortion of the sinusoidal double skin cladding.



In[22]:=
```
<< AceFEM`;
L = 80.; b = 100; hwave0 = 30; nwave = 2; T0foam = 5; T0steel = 1;
nx = 120; ny = 6; δh = 2 T0steel / ny 0.5; dx = L / (10. (hwave0 + T0foam));
Tfoam = T0foam; Tsteel = T0steel; hwave = hwave0;
SMTInputData[];
SMTAddDomain["Foam", "ExamplesTasks2D", {"ρ *" -> 10×10^-6, "t *" -> b}];
SMTAddDomain["Steel", "ExamplesTasks2D", {"ρ *" -> 7830×10^-6, "t *" -> b}];
SMTAddMesh[Raster[{ Table[{x, hwave / 2 Sin[nwave π x / L] - Tfoam / 2} , {x, 0, L, dx}], Table[
     {x, hwave / 2 Sin[nwave π x / L] + Tfoam / 2} , {x, 0, L, dx}]}], "Foam", "Q1", {nx, ny}];
SMTAddMesh[Raster[{Table[ {x, hwave / 2 Sin[nwave π x / L] + Tfoam / 2 }, {x, 0, L, dx}],
    Table[ {x, hwave / 2 Sin[nwave π x / L] + Tfoam / 2 + Tsteel },
     {x, 0, L, dx}]}], "Steel", "Q1", {nx, 1}];
SMTAddMesh[Raster[{Table[ {x, hwave / 2 Sin[nwave π x / L] - Tfoam / 2 - Tsteel }, {x, 0, L, dx}],
    Table[ {x, hwave / 2 Sin[nwave π x / L] - Tfoam / 2 },
     {x, 0, L, dx}]}], "Steel", "Q1", {nx, 1}];
SMTAnalysis[];
```

Here the mass if the structure is evaluated.

In[33]:=
```
SMTTask["Mass"]
```

```
125.68
```

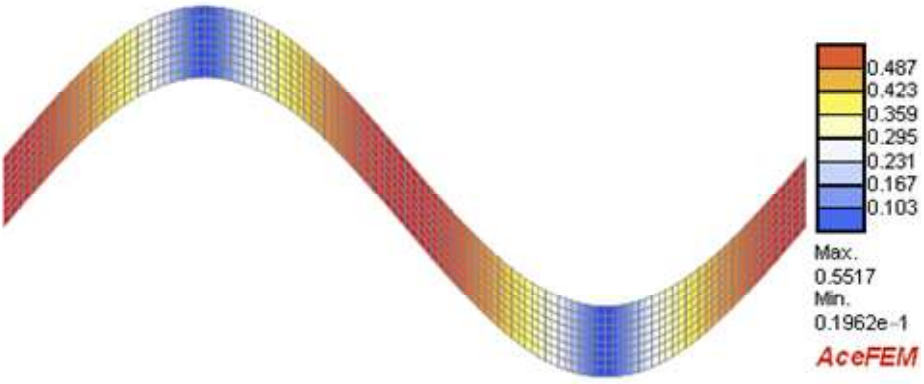
The mass if the structure can as well be calculated as a sum of the mass of the steel and the mass of the foam.

In[34]:=
```
SMTTask["Mass", "Elements" -> "Foam"] + SMTTask["Mass", "Elements" -> "Steel"]
```

```
125.68
```

Here the distortion of the mesh is calculated for all integration points, extrapolated to the nodes and depicted by the SMTShowMesh command.

In[35]:=
```
SMTShowMesh["Field" → SMTTask["MeshDistortion"]]
```



Here the distortion of the mesh is calculated for all integration points in the neighborhood of the given spatial position {L/2,0} and extrapolated to the given spatial position.

```
In[36]:= SMTTask["MeshDistortion", "Point" → {L / 2, 0}]
```

```
0.551796
```

Globally Defined User Tasks

The process of defining and executing globally defined tasks is composed of:

- definition of the `UserGlobalTasks` user subroutine in C language;
- C source code file with the definition of `UserGlobalTasks` **subroutine** can be compiled and dynamically linked into CDriver at any point of analysis (run time linking). `SMTLoadGlobalTasks` compiles and dynamically links the `UserGlobalTasks` subroutine into CDriver;
- at any point of the analysis user defined global task can be executed by the `SMTRunGlobalTask` command.

There can be only one active `UserGlobalTasks` subroutine defined at any given time!

UserGlobalTasks

User global task subroutine template:

```
#include \" sms.h\"

DLLEXPORT int UserGlobalTasks(

char *taskName,

SMTStructure *SMT,

ElementSpec** ElemSpecs,

ElementData** Elements,

NodeSpec**  NodeSpecs,

NodeData** Nodes,

int *IData,

double *RData)

{

/*body*/

return 1;

}
```

User global task subroutine has predefined input/output parameters as follows:

parameter	description
*char *taskName*	an arbitrary string interpreted in UserGlobalTasks
SMTStructure *SMT	pointer to SMTStructure data structure
ElementSpec** ElemSpecs	all element specification data - Domain Specification Data
ElementData** Elements	all element data - Element Data
NodeSpec** NodeSpecs	all node specification data - Node Specification Data
NodeData** Nodes	all nodes data - Node Data
int *IData	Integer Type Environment Data
double *RData	Real Type Environment Data

The UserGlobalTasks subroutine parameters.

SMTRunGlobalTask	UserGlobalTask
integerInputParameters	SMT->TasksStructure.IntegerInput[*i*]: *i*= 0,... SMT->TasksStructure.IntegerInputLength-1 where SMT->TasksStructure.IntegerInputLength= Length[*integerInputParameters*]
realInputParameters	SMT->TasksStructure.RealInput[*i*]: *i*= 0,... SMT->TasksStructure.RealInputLength-1 where SMT->TasksStructure.RealInputLength=Length[*realInputParameters*]
charSwitch	SMT->TasksStructure.CharInputField

The relation between the SMTRunGlobalTask input parameters and the UserGlobalTasks subroutine code.

Example: stretch X coordinate in all nodes for factor 2

In[16]:=
```
Export["stretch.c", "
#include \"sms.h\"
DLLEXPORT int UserGlobalTasks(char *taskName, SMTStructure \
*SMT, ElementSpec** ElemSpecs, ElementData** Elements, NodeSpec** \
NodeSpecs, NodeData** Nodes, int *IData, double *RData){
int i;
for(i=0;i<IData[ID_NoNodes];i++){
    Nodes[i]->X[0]=2*Nodes[i]->X[0];
};
return 1;
}
", "Text"];
```

SMTStructure

SMTStructure is CDriver data structure that can be used within the user defined subroutines to directly access various functions and data defined globally in CDriver.

Example: If the SMT variable is defined in C code as SMTStructure *SMT then we can make a time step with SMT->NextStep(dtime, dlambda, dgama).

```
typedef struct {
 char *SimulationDllFile;
 char *H5InputFile;
 void (*InitializeConvergence)();
 int (*NextStep)(double, double, double);
 int (*StepBack)();
```

```
 int (*Convergence)(double, double,char *);
 double (*NewtonIteration)();
 void (*FreeConvergence)();
 void (*Put)(int);
 int (*DumpState)(char *dumpkey,int *exc,int level);
 int (*RestartState)(char *dumpkey);
 double (*TimeStamp)(char *msg);
 void (*ExportVectorToH5File)(char *, char *, void *, int, char *);
 ConvergenceOptions ConvergenceOptions;
 ConvergenceReturn ConvergenceReturn;
 PutOptions PutOptions;
 int (*Task)(char *code); /*pointer to BatchTask function*/
 TasksStructure TasksStructure; /*user defined tasks data*/
 int (*Sensitivity)();
 /* nodes and elements selection functions */
 int (*FindNodes)(char *, int **);
 int (*FindNodesPoint)(double *, char *,double tol, int **);
 int (*FindNodesLine)(double **, int, char *,double, int **);
 int (*FindNodesPolygon)(double **, int, char *,double tol, int **);
 int (*FindNodesTetrahedron)(double **, char *,double tol, int **);
 int (*FindNodesHexahedron)(double **, char *,double tol, int **);
 int (*FindNodesFunction)(int (*)(double *, double), char *, int **);
 int (*FindElements)(int *, int ,int , int **);
 void (*Free)(void *);
 void (*ReallocateRealField)(double **f,int *fl,int *l, int nl);
 void (*ReallocateIntegerField)(int **f,int *fl,int *l, int nl);
 int *TempIntVector;
 double *TempRealVector;
 /*DllBackCall can be called from element DLL with SMT->ExportReKeToRK(ed,es,nd,ns,p,s);*/
 int (*DllBackCall)(ElementData *ce,ElementSpec *cs,NodeData **nd, NodeSpec** ns, double
vl[],double ml[]);
} SMTStructure;
```

SMTStructure data structure.

```
typedef struct
{
 double Tolerance;
 int Summation;
 int Averaging;
 int *IntegerInput;
 double *RealInput;
 int *Elements;
 int NoElements;
 int IntegerInputLength;
 int RealInputLength;
 int IntegerInputFieldLength;
 int RealInputFieldLength;
 int Failed;
```

```
  int Type;
  int IntegerOutputLength;
  int IntegerOutputFieldLength;
  int *IntegerOutput;
  int RealOutputLength;
  int RealOutputFieldLength;
  double *RealOutput;
  int *IntegerOutputForElements;
  double *RealOutputForElements;
  int IntegerOutputForElementsLength;
  int RealOutputForElementsLength;
  int IntegerOutputForElementsFieldLength;
  int RealOutputForElementsFieldLength;
  char *CharInputField;
  double *PostVector;         /* postprocessing vector length NoAllNodes */
  double *MatrixGlobal;       /* global  matrix length NoDOF*NoDOF*/
  double *VectorGlobal;       /* global  vector length NoDOF*/
  int *TasksData;
  int *TasksPosition;         /* position of the task keyword within the SMSCharSwitch field of the
element types 1,2,3*/
  char *userChar1;
  FILE *userFile1;
  int *userInt1;
  double *userReal1;
  char *userChar2;
  FILE *userFile2;
  int *userInt2;
  double *userReal2;
  char *userChar3;
  FILE *userFile3;
  int *userInt3;
  double *userReal3;
} TasksStructure;
```

TasksStructure data structure is part of SMTStructure and used for the IO within the user defined tasks.

```
typedef struct
{
  char Alternate[MAX_DATA_LEN];
  double AlternateList[6]; /* {{_,_},{_,_},{_,_}} */
  int PostIteration; /* T/F (-1:automatic) */
  int IgnoreMinBound; /* T/F */
  int AlternativeTarget; /* T/F */
  char Type[MAX_DATA_LEN];
  int TargetIterations;
  double MinIncrement; /* lambda or time */
  double MaxIncrement;
  double Target;
  int ConvFunctOpt; /* 1,2,3 */
```

```
  double LargeDrop;
 } ConvergenceOptions;
```

ConvergenceOptions data structure is part of SMTStructure.

```
 typedef struct
 {
  char Indeterminate[MAX_DATA_LEN];
  char MaxIterations[MAX_DATA_LEN];
  char Convergence[MAX_DATA_LEN];
  char ErrorStatus[MAX_DATA_LEN];
  char Alternate[MAX_DATA_LEN];
  char Divergence[MAX_DATA_LEN];
  char Other[MAX_DATA_LEN];
  int IsOtherDefined; /* T/F */
 } ConvergenceStatus;
```

ConvergenceStatus data structure is part of SMTStructure.

```
 typedef struct
 {
  char Indeterminate[MAX_DATA_LEN];
  char MaxIterations[MAX_DATA_LEN];
  char Convergence[MAX_DATA_LEN];
  char ErrorStatus[MAX_DATA_LEN];
  char Alternate[MAX_DATA_LEN];
  char Divergence[MAX_DATA_LEN];
  char Other[MAX_DATA_LEN];
  int IsOtherDefined; /* T/F */
 } ConvergenceReturn;
```

ConvergenceReturn data structure is part of SMTStructure.

SMTLoadGlobalTasks

SMTLoadGlobalTasks[*fileName*]

C source code file *fileName.c* with the definition of `UserGlobalTasks` subroutine is compiled and dynamically linked into *CDriver*.

SMTRunGlobalTask

SMTRunGlobalTask[*taskName, integerInputParameters, realInputParameters, charSwitch*]

SMTRunGlobalTask[*taskName*]≡SMTRunGlobalTask[*taskName*,{},{},""]

runs UserGlobalTasks subroutine with given parameters and returns vectors

{

SMT->TasksStructure.IntegerOutput[i]: i=0,...,SMT->TasksStructure.IntegerOutputLength-1

,SMT->TasksStructure.RealOutput[i]: i=0,...,SMT->TasksStructure.RealOutputLength-1

}

Input parameters of the SMTRunGlobalTask are copied into input parameters of the `UserGlobalTasks` subroutine accordingly to the table below.

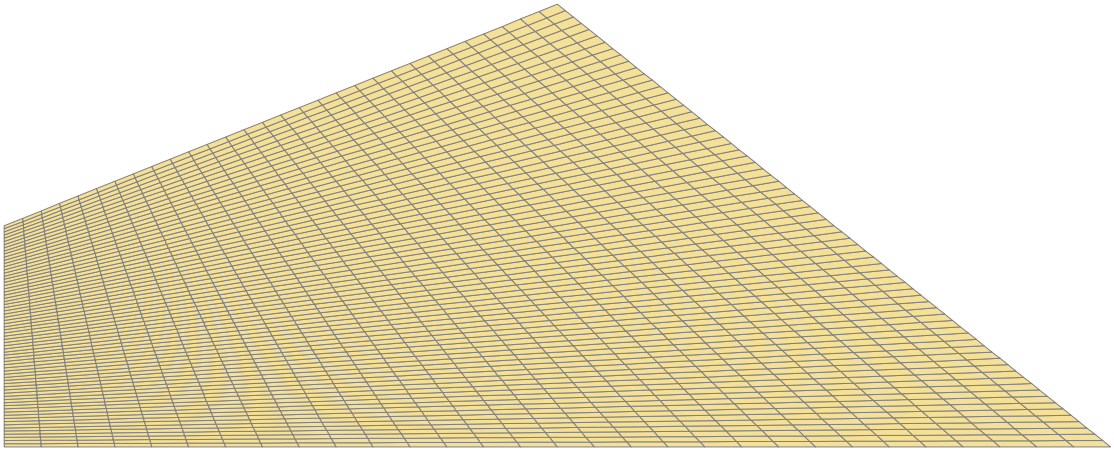
SMTRunGlobalTask	UserGlobalTask
taskName	*taskName*
integerInputParameters	SMT->TasksStructure.IntegerInput[*i*]: *i*= 0,... SMT->TasksStructure.IntegerInputLength-1 where SMT->TasksStructure.IntegerInputLength= Length[*integerInputParameters*]
realInputParameters	SMT->TasksStructure.RealInput[*i*]: *i*= 0,... SMT->TasksStructure.RealInputLength-1 where SMT->TasksStructure.RealInputLength=Length[*realInputParameters*]
charSwitch	SMT->TasksStructure.CharInputField

The relation between the SMTRunGlobalTask input parameters and the UserGlobalTasks subroutine code.

Global task that finds neighboring elements of the given set of elements

- Definition of the mesh.

In[205]:=
```
<< AceFEM`;
SMTInputData["Console" → False];
L = 100; H = 20; nx = 30; ny = 70;
points = {{0, 0}, {L, 0}, {L / 2, 2 H}, {0, H}};
SMTAddDomain["A",
  {"ML:", "SE", "PE", "Q1", "DF", "LE", "Q1", "D", "Hooke"}, {"E *" → 21000, "ν *" → 0.2}];
SMTAddMesh[Polygon[points], "A", "Division" -> {nx, ny}];
SMTAnalysis[];
SMTShowMesh[]
```



- UserGlobalTasks subroutine that gets a neighboring elements of the given set of elements.

In[45]:=
```
Export["userTasks10.c", "
#include \"sms.h\"
DLLEXPORT int UserGlobalTasks(char *task, SMTStructure \
*SMT,ElementSpec** ElemSpecs, ElementData** Elements, NodeSpec** \
NodeSpecs, NodeData** Nodes, int *IData, double *RData){
int i,j,k,n,m,nout,*work,ei,ni,iil;
ElementData *ce;ElementSpec *cs;
NodeData *cn;NodeSpec *cns;
work=SMT->TempIntVector;
if(strcmp(task,\"Neighborhood\")==0){
  for(i=0;i<IData[ID_NoElements];i++)work[i]=0;
  for(i=0;i<SMT->TasksStructure.IntegerInputLength;i++){
    ei=SMT->TasksStructure.IntegerInput[i]-1;
    ce=Elements[ei];
    cs=ElemSpecs[ce->id.SpecIndex];
    work[ei]=1;
    for(j=0;j<cs->id.NoNodes;j++){
      ni=ce->Nodes[j]-1;
      cn=Nodes[ni];
      cns=NodeSpecs[cn->id.SpecIndex];
      k=0;while(cn->Elements[k]){
        work[cn->Elements[k]-1]=1;
        k++;
      };
    }
  };
  nout=0;for(i=0;i<IData[ID_NoElements];i++) if(work[i]){work[nout]=i+1;nout++;};
  SMT->TasksStructure.IntegerOutputLength=nout;
  return 3;
};
return 0;
}
", "Text"];
```

- Compile and link C code.

In[46]:=
```
SMTLoadGlobalTasks["userTasks10"]
```

Out[46]=
```
True
```

- Choose random selection of 150 elements.

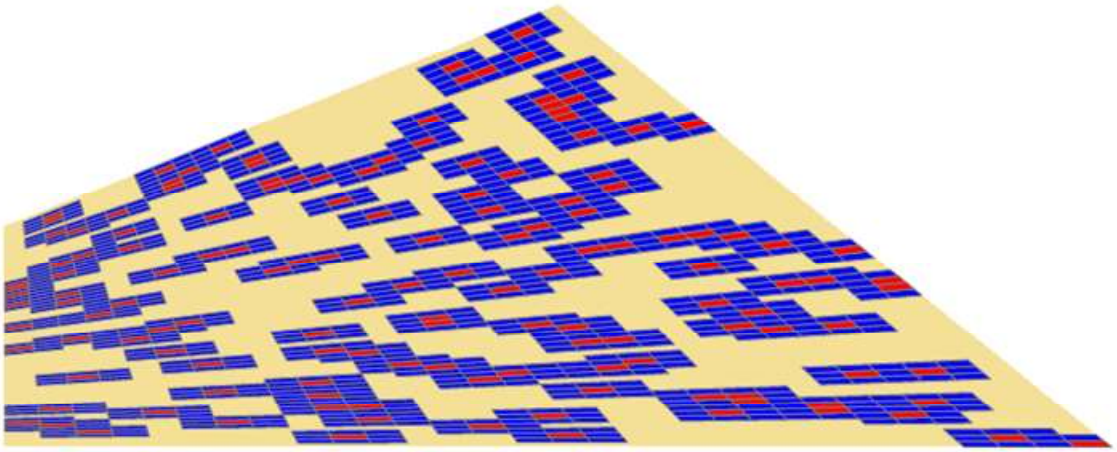
In[47]:=
```
base = RandomInteger[{1, nx ny}, 150]
```

Out[47]=
```
{1099, 1728, 742, 1163, 1807, 707, 1284, 1952, 207, 643, 813, 1918, 376, 904, 723,
 686, 1676, 608, 1008, 2057, 48, 1704, 145, 640, 1454, 710, 185, 1007, 45, 1462,
 646, 31, 1583, 408, 183, 208, 1177, 77, 995, 915, 1364, 1924, 1943, 727, 873, 1632,
 625, 608, 413, 1053, 404, 1521, 1848, 38, 697, 1276, 727, 8, 1856, 1281, 1903,
 1038, 1177, 1712, 2082, 1746, 1808, 265, 1479, 1347, 1507, 1057, 248, 50, 194,
 629, 1892, 172, 1959, 1618, 1066, 191, 792, 272, 899, 1407, 320, 1756, 904, 2031,
 1202, 1809, 1480, 1442, 883, 1376, 1410, 382, 839, 310, 1678, 928, 1371, 203, 161,
 1099, 1717, 626, 792, 2056, 1804, 435, 2062, 1015, 1351, 969, 1828, 31, 1787, 1389,
 1222, 1585, 290, 678, 1810, 900, 1379, 1577, 1033, 1636, 1248, 1156, 633, 82, 1879,
 1617, 1841, 904, 1726, 453, 1763, 905, 427, 51, 466, 1215, 1701, 859, 1886, 442}
```

- Get a neighboring elements of the given set of elements.

In[48]:=
```
neighborhood = SMTRunGlobalTask["Neighborhood", base, {}, ""];
```

In[49]:=
```
Show[{
  SMTShowMesh["Mesh" → False],
  SMTShowMesh["ZoomElements" → neighborhood, "FillElements" → Blue],
  SMTShowMesh["ZoomElements" → base, "FillElements" → Red]}]
```



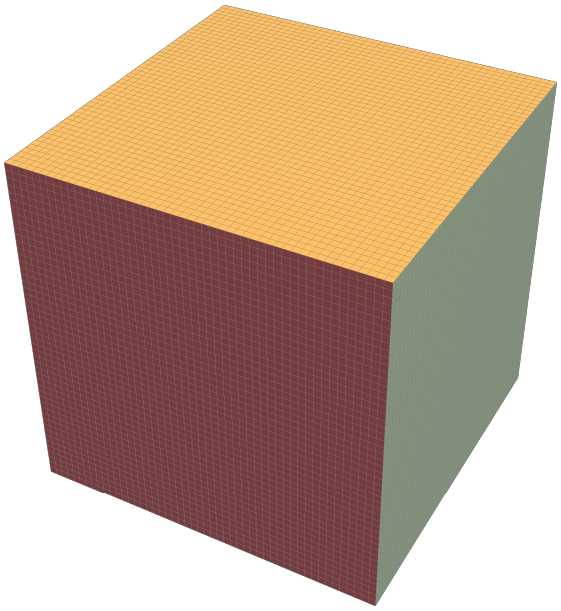
Numerical efficiency of AceFEM data manipulations

AceFEM is open environment where all the simulation data can be accessed and changed during the analysis. However, when the large scale finite element models have to be manipulated the numerical efficiency of the data manipulations plays a crucial role. In general data can be manipulated by:

- a) writing a procedural program in Mathematica language combined with `Data Base Manipulations` commands,
- b) writing a functional program in Mathematica language combined with `Data Base Manipulations` commands,
- c) writing a global tasks subroutine in C language.

The numerical efficiency of various approaches is assessed on a simple example. Let assume that we have 50×50×50 3D mesh of elements and we have to stretch the undeformed domain in X direction for factor 2. Thus X coordinate in all nodes has to be multiplied by 2.

In[43]:=
```
<< AceFEM`;
nx = ny = nz = 50;
B = 1.; L = 1.; H = 1.;
SMTInputData[];
points = {{0, 0, 0}, {B, 0, 0}, {B, L, 0}, {0, L, 0}, {0, 0, H}, {B, 0, H}, {B, L, H}, {0, L, H}};
SMTAddDomain["powder", {"ML:", "SE", "D3", "H1", "DF", "LE", "H1", "D", "Hooke"}, {}];
SMTAddMesh[Hexahedron[points], "powder", "Division" -> {nx, ny, nz}];
SMTAnalysis[];
SMTShowMesh[ImageSize → 300]
```



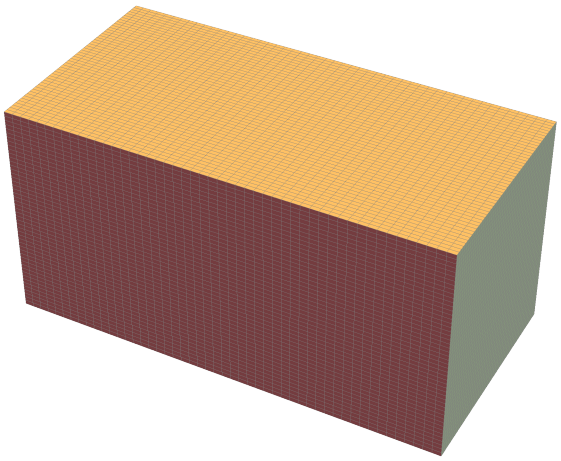
Solution 1: Procedural programing

- The problem with the procedural program is that the data has to be transferred between the Mathematica and the CDriver several times. The data transfer goes through MathLink protocol that has to be opened and closed each time, thus it has a certain fixed cost for each individual call.

In[52]:=
```
time1 = AbsoluteTiming[
  Do[
   SMTNodeData[i,
    "X",
    {2 SMTNodeData[i, "X"][[1]], SMTNodeData[i, "X"][[2]], SMTNodeData[i, "X"][[3]]}
   ]
   , {i, 1, SMTIData["NoNodes"]}
  ]
 ]
```

Out[52]=
```
{20.3233, Null}
```

In[53]:=
```
SMTUpdateDataBase[];
SMTShowMesh[ImageSize → 300]
```



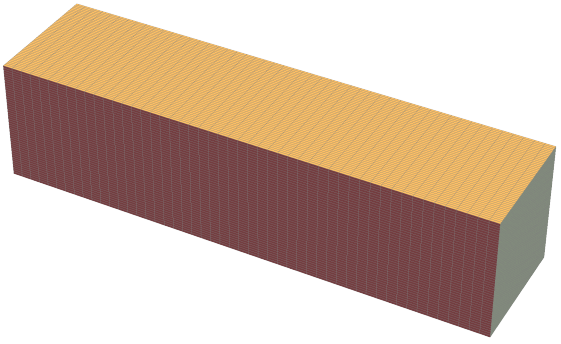
Solution 2: Functional programing

- Only 2 call to SMTNodeData results in only 2 MathLink transfers resulting in a significant reduction of time.

In[55]:=
```
time2 = AbsoluteTiming[
  SMTNodeData["X"
   , Map[
    {2 #[[1]], #[[2]], #[[3]]} &
    , SMTNodeData["X"]]
   ]
  ]
```

Out[55]=
```
{4.0336, True}
```

In[56]:=
```
SMTUpdateDataBase[];
SMTShowMesh[ImageSize → 300]
```



Solution 3: Global CDriver task

- Global task is evaluated in CDriver, thus it requires no data transfer resulting in 4000 times shorter time when compered to the procedural Mathematica program.

In[58]:=
```
Export["stretch.c", "
#include \"sms.h\"
DLLEXPORT int UserGlobalTasks(char *task, SMTStructure \
*SMT,ElementSpec** ElemSpecs, ElementData** Elements, NodeSpec** \
NodeSpecs, NodeData** Nodes, int *IData, double *RData){
int i;
for(i=0;i<IData[ID_NoNodes];i++){
    Nodes[i]->X[0]=2*Nodes[i]->X[0];
};
return 1;
}
", "Text"];
```

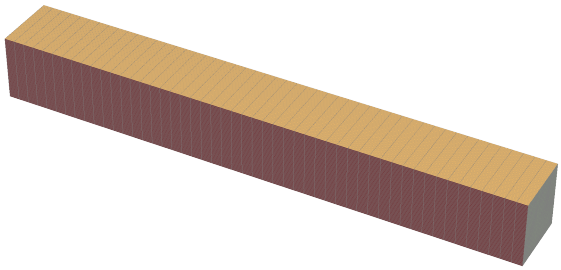
In[59]:=
```
SMTLoadGlobalTasks["stretch"]
```

Out[59]=
```
True
```

In[60]:=
```
time3 = AbsoluteTiming[SMTRunGlobalTask["XXX", {}, {}, ""]]
```

Out[60]=
```
{0.00531098, True}
```

In[61]:=
```
SMTUpdateDataBase[];
SMTShowMesh[ImageSize → 300]
```



Analysis: normalized time

In[63]:=
```
{time1[[1]], time2[[1]], time3[[1]]}/time3[[1]]
```

Out[63]=
```
{3826.66, 759.483, 1.}
```

Save and Restart Session

SMTDump[*name*]	saves restart data to the files *name.m, name.h5* and *name.bst.* Returns the number of bytes stored.
SMTDump[*name, symb1,symb2,...*]	saves restart data to the files *name.m, name.h5* and *name.bst* and appends definitions associated with the given symbols to file *name.m*
SMTRestart[*name*]	reads the restart data and continues the AceFEM session from the point of the corresponding SMTDump command
SMTRestart[*name, options*]	SMTRestart takes the same options as SMTInputData
SMTDumpState[s*name*]	saves only the state of the simulation (unknowns, history data, etc.) to the file s*name.bst.* Returns the number of bytes stored.
SMTRestartState[s*name*]	reads the state of the simulation (unknowns, history data, etc.) and continues the AceFEM session from the point of the corresponding SMTDumpState command

option	default	description
"LocalDll"	False	True ⇒ all element dll-s are copied to the current directory in order to make restart machine independent
"BatchModeSteering"	""	batch mode steering file (see `Standard batch mode procedure`)
"State"	True	True ⇒ the input data as well as the complete CDriver data structures are saved False ⇒ only input data is saved
"CompressionLevel"	2	-2 ⇒ plain binary data -1 ⇒ zlib default compression (https://www.zlib.net/) 0 - 9 ⇒ level of compression accordingly to zlib (0 minimum, 9 maximum compression)
"Exclude"	{{}, {"da", "tmp"}}	specification of the data fields that are set to zero before they are written to the *name.bst* file {*list of element data fileds*, *list of node data fields*} element data fields can be any of: {} node data fields can be any of: {"da","st","sp","Data","tmp","Bt","Bp","dB","ADVF"}
"AccumulateStatistics"	"Session"	"Session" ⇒ continue statistics across the same AceFEM simulation "Driver" ⇒ continue statistics across the same run of CDriver executable False ⇒ start from zero after SMTRestartState
"CreateBundle"	False	True ⇒ a directory with the name *name_AceFEM* is created where all the files are included for the Mathematica independent simulation (see Independent Batch Mode)

Options of the SMTDump function.

In order to restart the session on different machine than the one where simulation was started, all the necessary dll-s have to be copied and stored at the same directory as notebook. With the option "LocalDll"->True a local copy of all dll-s used is made in prepared for restart. For the operating system independent restart one needs also original C files placed and compiled at the same directory.

SMTDump is not available in M*Driver*!

The SMTDump commands saves all the data associated with the current state of the active AceFEM session so that it can be retrieved by the SMTRestart command.

The SMTDump command creates several files:

- *name.h5* file is a HDF5 (see HDF5) binary file that stores general input data
- *name.bst* file is a binary file that stores data that defines the state of the simulation (unknowns, history data, etc.). The data is stored in compressed format. zlib compression library is used for that purpose (https://www.zlib.net/).

The *SMTDump*/*SMTRestart* can be used unlimited times within the same session. However, the *name.h5* files remain in principle the same throughout the session (if the user does not use advanced data manipulations!!), while *name.bst* depends on the current state of simulations. In order to save time and space one can, after the first use of SMTDump command, store only the state of simulation at

various times with SMTDumpState command. The state of the simulation at the certain time is then retrieved by the SMTRestartState command.

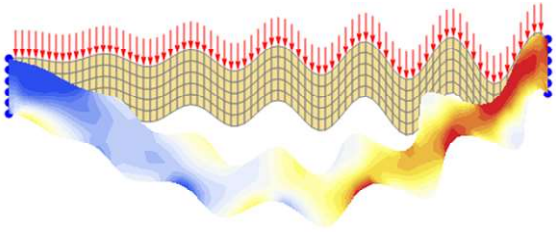
Example: Restarting the AceFEM session

- Here we first analyze the give structure up to the load level 500.

In[154]:=
```
<< AceFEM`;
```

In[213]:=
```
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "PS", "Q1", "DF", "HY", "Q1", "D", {{"NeoHooke", "WA"}}},
  {"E *" -> 1000., "ν *" -> .49}];
SMTAddNaturalBoundary[Abs["X" Sin["X" // N] / 20 + 8 - "Y"] < 0.1 &, 2 -> -.01];
SMTAddEssentialBoundary["X" == 1 &, 1 -> 0, 2 -> 0];
SMTAddEssentialBoundary["X" == 40 &, 1 -> 0, 2 -> 0];
SMTAddMesh[
  Raster[Array[ {#2, #2 Sin[#2 // N] / 20 + 4 #1 } &, {2, 40}]], "A", "Division" -> {80, 5}];
SMTAnalysis[];
SMTNextStep["λ" → 100];
While[
  While[
   step = SMTConvergence[10^-7, 15, {"Adaptive BC", 8, .01, 300, 500}] , SMTNewtonIteration[]];
  SMTStatusReport[];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  If[Not[step[[1]]], SMTShowMesh["DeformedMesh" → True,
    "Field" → "Sxy", "Mesh" → False, "Contour" → 20, "Show" -> "Window"]];
  step[[3]] ,
  If[step[[1]], SMTStepBack[]];
  SMTNextStep["Δλ" → step[[2]]]
 ];
Show[SMTShowMesh["Show" → False, "BoundaryConditions" → True],
 SMTShowMesh["DeformedMesh" → True, "Mesh" → False, "Field" → "Sxy",
  "Contour" → 20, "Show" → False, "Legend" → False], PlotRange → All]
```

```
Step/Iter=1/8 λ/Δλ=100./100. ||Δp||/||R||=0./0. Events=0 Status=0/{}
Step/Iter=2/7 λ/Δλ=226.531/126.531 ||Δp||/||R||=0./0. Events=0 Status=0/{}
Step/Iter=3/6 λ/Δλ=415.035/188.505 ||Δp||/||R||=0./0. Events=0 Status=0/{}
Step/Iter=4/5 λ/Δλ=500./84.9646 ||Δp||/||R||=0./0. Events=0 Status=0/{}
```



- The state of the analysis at the load level 500 is then stored in "dump1" restart files.

In[165]:=
```
SMTDump["dump1", "State" → True]
```

```
187 636
```

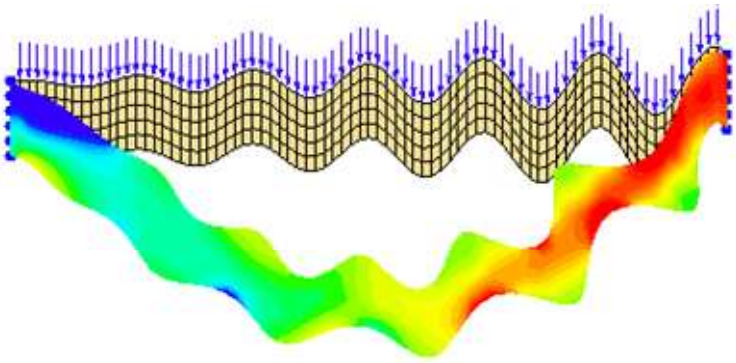
- Here the data associated with the state of the AceFEM session at the load level 500 is restored from the "dump1" restart files and the analysis then continues up to the load level 1000.

In[166]:=
```
<< AceFEM`;
```

In[167]:=
```
SMTRestart["dump1"];
```

In[168]:=
```
While[
  While[step = SMTConvergence[10^-7, 15, {"Adaptive BC", 8, .01, 300, 1000}] ,
   SMTNewtonIteration[]];
  SMTStatusReport[];
  If[step[[4]] === "MinBound", SMTStatusReport["Δλ < Δλmin"]];
  If[Not[step[[1]]], SMTShowMesh["DeformedMesh" → True,
    "Field" → "Sxy", "Mesh" → False, "Contour" → 20, "Show" -> "Window"]];
  step[[3]] ,
  If[step[[1]], SMTStepBack[]];
  SMTNextStep["Δλ" → step[[2]]]
 ];
Show[SMTShowMesh["Show" → False, "BoundaryConditions" → True],
 SMTShowMesh["DeformedMesh" → True, "Mesh" → False, "Field" → "Sxy",
  "Contour" → 20, "Show" → False, "Legend" → False], PlotRange → All]
```

```
Step/Iter=4/6 λ/Δλ=500./84.9646 ||Δp||/||R||=4.55187×10^-8/1.1537×10^-12 Events=0 Status=0/{}

Step/Iter=5/6 λ/Δλ=642.186/142.186 ||Δp||/||R||=
 3.19181×10^-13/1.13064×10^-12 Events=0 Status=0/{}

Step/Iter=6/6 λ/Δλ=880.129/237.943 ||Δp||/||R||=
 2.18564×10^-12/1.19812×10^-12 Events=0 Status=0/{}

Step/Iter=7/5 λ/Δλ=1000./119.871 ||Δp||/||R||=
 1.38741×10^-9/1.24281×10^-12 Events=0 Status=0/{}
```



Parallel AceFEM Computations

The AceFEM based finite element simulations can be accelerated by utilizing three types of parallelization:

A) the procedure used to collect the contributions of individual finite elements to the global matrices and vectors is fully parallelized for the multi-core environments (parallelization of the assembly procedure)

B) the solution to the system of linear equations performed by the PARDISO linear solver is parallelized for the multi-core environments (parallelization of the linear solver)

C) several finite elements simulations can be performed in parallel on multi-core or grid environments using *Mathematica* 7.0 parallel computing capabilities (parallelization of the FE simulations)

The user can control the type A and the type B parallelization by setting the SMTInputData option "Threads" (see `SMTInputData`). The "Threads" option limits the number of processors used for the parallel execution on multi-core systems.

The user controls the type C parallelization by launching a specified number of sub-kernels (see `LaunchKernels`).

Using all three types of parallelization on a single multi-core environments is obviously not an optimal parallelization strategy. The type A and the type B parallelization can be suppressed by setting SMTInputData option "Threads" to 1 (SMTInputData["Threads" → 1]).

The highest speedup is achieved when AceFEM is run on a grid of multi-core machines. The type C parallelization is then used to run several simulations in parallel on the nodes of the grid. Additionally, the assembly procedures and the linear solver of each simulation is also parallelized.

Example: Parallelization of complete FE simulations

- First one must lunch the AceFEM on a master kernel.

In[1]:=
```
Get["AceFEM`"];
```

- The user interface (palettes, menus, etc.) is not available on remote kernels. The Get["AceFEM`Remote`"] command loads the AceFEM package without the user interface on all available kernels. For more details how to select and set specific number of kernels see `Parallel Computing`.

In[2]:=
```
kernels = ParallelEvaluate[
    SetDirectory[$UserDocumentsDirectory];
    Get["AceFEM`Remote`"];
    $KernelID]
```

Out[2]=
```
{1, 2, 3, 4, 5, 6}
```

Calculate the typical load/deflection curve of the 20×2×2 column subjected to the constant horizontal force H = 10 and variable vertical force V = -λ 10 for the elastic modulus ranging from 10000 to 30000.

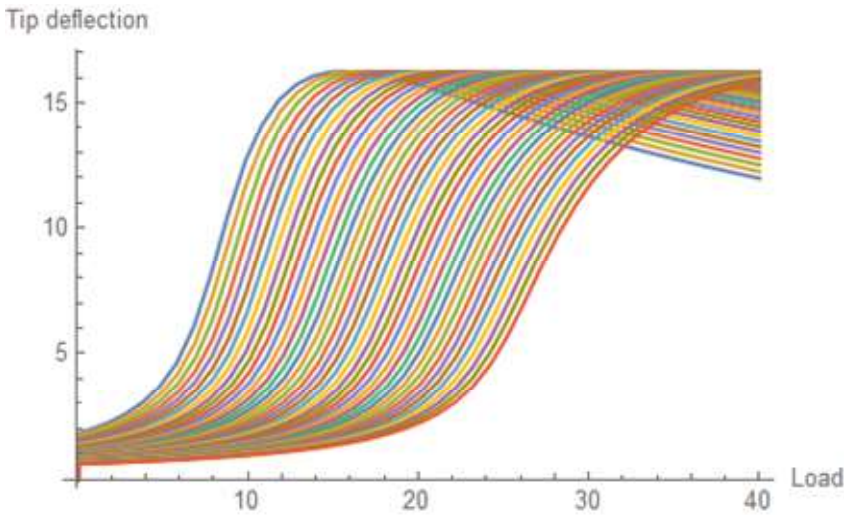
The ParallelTable commands executes the given AceFEM input data and the analysis procedure in parallel on all available kernels and generates a list of load/deflection curves.

The "Threads" → 1 option for the SMTInputData command prevents the parallelization of the assembly procedure and parallelization of the linear solver.

In[189]:=
```
table = ParallelTable[
    SMTInputData["Threads" → 1];
    SMTAddDomain["Ω", {"ML:", "SE", "PE", "Q2", "DF", "HY", "Q2", "D", {{"NeoHooke", "WA"}}},
     {"E *" -> emodule, "t *" → 2}];
    SMTAddEssentialBoundary[{ "Y" == 0 &, 1 -> 0, 2 -> 0}];
    SMTAddNaturalBoundary[ "X" == 0 && "Y" == 20 &, 2 -> -10];
    SMTAddInitialBoundary[ "X" == 0 && "Y" == 20 &, 1 -> 10];
    SMTAddMesh[Polygon[{{1, 0}, {1, 20}, {-1, 20}, {-1, 0}}], "Ω", "Division" -> {20, 5}];
    SMTAnalysis[];
    λucurve = {{0, 0}};
    SMTNextStep["λ" → .1];
    While[
     While[step = SMTConvergence[10^-8, 15, {"Adaptive BC", 8, .0001, 1, 40}],
      SMTNewtonIteration[];];
     If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
     If[Not[step[[1]]],
      AppendTo[λucurve, {SMTData["Multiplier"], SMTPostData["u", Point[{0, 20}]]}];];
     step[[3]]
     , If[step[[1]], SMTStepBack[];];
     SMTNextStep["Δλ" → step[[2]]]
    ];
    {emodule, λucurve}
    , {emodule, 10000, 30000, 500}];
```

- The result are the deflection curves of upper center node considering the changing elastic modulus.

In[190]:=
```
ListLinePlot[table[[All, 2]], AxesLabel → {"Load", "Tip deflection"}]
```


Tip deflection
15
10
5
10
20
30
40
Load

Independent Batch Mode

Contents

AceFEM can be also used without *Mathematica*. The *Mathematica* independent batch mode retains most of the capabilities of *AceFEM* except:

- mesh generation,
- higher level data base manipulations (advanced symbolic capabilities of *Mathematica* are not supported by C language!),
- visualization and post-processing of results.

In batch mode *AceFEM* reads input data from the input data file. Input data file is written in HDF5 format and it can be constructed either with *Mathematica* or any other HDF5 software.

The simulation procedure is controlled from the batch mode steering dynamic library (*steering-file.dll*). The steering DLL can be generated from the *steering-file.c* source code by AceFEM.

Standard batch mode procedure

The procedure presented is composed from three parts:

- preparation of the batch mode input data file and steering dll with *Mathematica*,
- running of simulation with remote Mathematica kernel or as Mathematica independent batch mode application,
- visualization of the results with *Mathematica*.

Preparation of the batch mode input data

- Here is the steering application written and exported to file Steering.c. A set of commands used in steering application is a direct translation of the commands used in *Mathematica* to control the `Iterative Solution Procedures`. Some restrictions are imposed due to the restrictions of the C language.
  The SMT->DumpState("batch") command at the end writes the complete simulation data base to files so that it can be restarted later in *Mathematica* for post-processing.

In[34]:=
```
<< AceFEM`;
```

In[35]:=
```
Export["Steering.c", "
#include \"sms.h\"
DLLEXPORT int Simulation(SMTStructure *SMT,ElementSpec** ElemSpecs, ElementData**
    Elements, NodeSpec** NodeSpecs, NodeData** Nodes, int *IData, double *RData){
int i;
SMT->ConvergenceOptions.MinIncrement = 0.0001;
SMT->ConvergenceOptions.MaxIncrement = 100;
SMT->ConvergenceOptions.Target = 500;
SMT->NextStep(1, 100,0);
while(SMT->ConvergenceReturn.StepForward){
    while(SMT->Convergence(1.e-9, 30,\"Adaptive BC\"))SMT->NewtonIteration();
    if(strcmp(SMT->ConvergenceReturn.Report,\"MinBound\")==0){
        SMT->TimeStamp(\"Divergence.\");
        break;
    };
    if(SMT->ConvergenceReturn.StepBack) SMT->StepBack();
    if(SMT->ConvergenceReturn.StepForward)
    SMT->NextStep(1,SMT->ConvergenceReturn.Increment,0);
};
SMT->DumpState(\"batch\",NULL,2);
return 1;
}
", "Text"];
```

- Here the problem is described in usual way.

In[223]:=
```
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "PS", "Q1", "DF", "HY", "Q1", "D", {{"NeoHooke", "WA"}}},
  {"E *" -> 1000., "ν *" -> .49}];
SMTAddNaturalBoundary[Abs["X" Sin["X" // N] / 20 + 8 - "Y"] < 0.1 &, 2 -> -.01];
SMTAddEssentialBoundary["X" == 1 &, 1 -> 0, 2 -> 0];
SMTAddEssentialBoundary["X" == 40 &, 1 -> 0, 2 -> 0];
SMTAddMesh[
  Raster[Array[ {#2, #2 Sin[#2 // N] / 20 + 4 #1 } &, {2, 40}]], "A", "Division" -> {8, 15}];
SMTAnalysis[];
```

- The SMTDump command will create *batch.h5* file that contains input data in a format required by the batch mode AceFEM executable program. The "Steering.dll" dynamic library is generated from the "Steering.c" file. “CreateBundle”→True options creates a directory *name_AceFEM* with all the files needed for the Mathematica independent execution of simulation.

In[43]:=
```
SMTDump["batch", "State" → False, "BatchModeSteering" → "Steering",
   "CreateBundle" → True, "Debug" → True]; // AbsoluteTiming
```

Out[43]=
```
{4.77273, Null}
```

- SMTQuit[] unloads the AceFEM and quits all kernels.

In[11]:=
```
SMTQuit[]
```

Run the batch mode simulation from Mathematica notebook

SMTRun[*dumpfile*]
runs the simulation in batch mode using the restart data saved under name *dumpfile*

SMTRun[*dumpfile*,"Debug"->True]
runs the simulation in batch mode using the restart data saved under name *dumpfile* in debug mode

- A remote mode version of AceFEM loaded here does not support notebook interface. Consequently, batch mode simulation does not print an error report into notebook, but to the console window.

In[1]:=
```
SetDirectory[NotebookDirectory[]];
Get["AceFEM`Remote`"];
```

- Run the batch mode session. When 0 is returned this indicates successful simulation.

In[3]:=
```
SMTRun["batch", "Debug" → False]
```

```
0
```

- Run the batch mode session again in debug mode. The console window with the error messages can be opened with the option "Debug"->True.

In[4]:=
```
SMTRun["batch", "Debug" → True]
```

```
0
```

- SMTQuit[] unloads the AceFEM and quits all kernels.

In[5]:=
```
SMTQuit[]
```

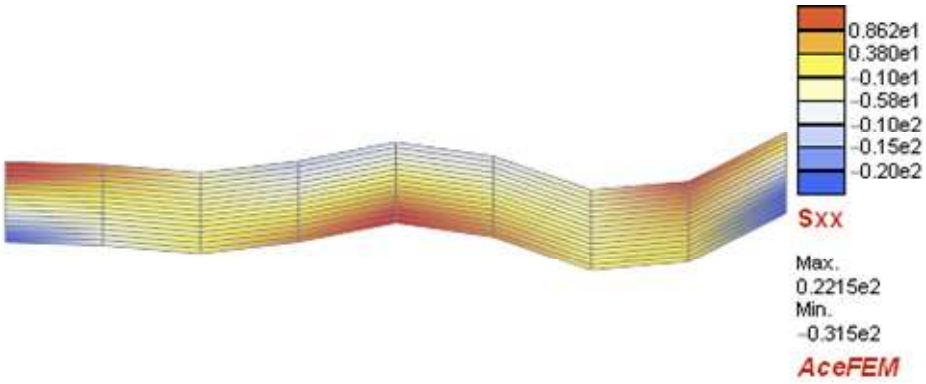
Visualization of the results

- Here the simulation is restarted from the restart files stored at the end of simulation.

In[45]:=
```
<< AceFEM`;
SMTRestart["batch"];
```

- The simulation can continue from the point when it has been stored.

In[47]:=
```
SMTShowMesh["Field" -> "Sxx"]
```



In[5]:=
```
SMTQuit[]
```

Run the batch mode simulation from Mathematica kernel

- The active core or kernel of Mathematical can be called directly without the notebook interface. This option is convenient when we have a grid of computers with Mathematica and AceFEM installed on each node of the grid.
- "batch.wl" generated hare is an ordinary Mathematica script file that can be executed directly by Mathematica kernel.

In[1]:=
```
Export[FileNameJoin[{NotebookDirectory[], "batch.wl"}],
  "SetDirectory[" <> ToString[NotebookDirectory[] // InputForm] <> "];
Get[\"AceFEM`Remote`\"];
SMTRun[\"batch\",\"Debug\"->False];", "Text"];
```

- "batch.h5", "Steering.dll" and "batch.wl" files have to be copied to the node first. After that we can run the simulation by

In[2]:=
```
mathcommand = StringReplace[First[$CommandLine], "WolframKernel" → "wolfram"]
```

```
wolfram
```

In[3]:=
```
RunThrough[mathcommand <> " -noprompt", Get[FileNameJoin[{NotebookDirectory[], "batch.wl"}]]]
```

In[4]:=
```
SMTQuit[]
```

Run the batch mode simulation independently of Mathematica

- Alternatively one can also run the session by running the batch mode simulation from the command line directly. Note that the batch file commands and the name and the location of the AceFEM executable depends on the operating system used.
- *batch_AceFEM* directory contains in the case of Windows OS the folowing files:

In[46]:=
```
FileNames["*.*", FileNameJoin[{NotebookDirectory[], "batch_AceFEM"}]] // TableForm
```

```
D:\SMS\Documentation\batch_AceFEM\batch.bat
D:\SMS\Documentation\batch_AceFEM\batch.h5
D:\SMS\Documentation\batch_AceFEM\hdf5.dll
D:\SMS\Documentation\batch_AceFEM\hdf5_hl_cpp.dll
D:\SMS\Documentation\batch_AceFEM\hdf5_hl.dll
D:\SMS\Documentation\batch_AceFEM\libaceutility.a
D:\SMS\Documentation\batch_AceFEM\libiomp5md.dll
D:\SMS\Documentation\batch_AceFEM\ml64i3.dll
D:\SMS\Documentation\batch_AceFEM\MLSEPSQ1DFHYQ1DNeoHookeWAC.W64.dll
D:\SMS\Documentation\batch_AceFEM\Steering.W64.dll
D:\SMS\Documentation\batch_AceFEM\szip.dll
D:\SMS\Documentation\batch_AceFEM\vcruntime140.dll
D:\SMS\Documentation\batch_AceFEM\WinBatchDriver64.exe
D:\SMS\Documentation\batch_AceFEM\zlib.dll
```

- the *batch.bat* file is a command file that executes the simulation.
- It is important to set the working directory to be the directory containing the bundle of files created by SMTDump command. Otherwise the AceFEM executable cannot locate dll files!

In[1]:=
```
SetDirectory[FileNameJoin[{NotebookDirectory[], "batch_AceFEM"}]];
Run["batch.bat"]
```

```
0
```

Batch mode iterative solution procedure commands

SMTStructure is CDriver data structure that can be used within the user defined subroutines to directly access various functions and data defined globaly in CDriver. If the SMT variable is defined in C code as SMTStructure *SMT then we can make a time step with SMT->NextStep(dtime, dlambda, dgama). A complete SMTStructure data structure is given in `SMTStructure`.

MathLink mode	batch mode	*help*
SMTNextStep[Δt, Δλ, Δγ]	SMT-> NextStep (double Δt, double Δλ, double Δγ)	`SMTNextStep`
SMTStepBack[]	SMT->StepBack ()	`SMTStepBack`
SMTConvergence[*tolerance, maxsteps, type*]	SMT->Convergence (double *tol*,int *maxsteps*, char **type*)	`SMTConvergence`
SMTNewtonIteration[]	SMT->NewtonIteration ()	`SMTNewtonIteration`
SMTDumpState[*keyword*]	SMT->DumpState (char **keyword*)	`SMTDump`
SMTRestartState[*keyword*]	SMT->RestartState (char **keyword*)	`SMTDump`
SMTTask[*keyword*]	SMT->Task (char **keyword*)	User Defined Tasks
SMTForwardSensitivity[]	SMT->Sensitivity ()	`SMTForwardSensitivity`

See also: Analysis Phase Functions

Example:

- This is a path following procedure with an adaptive boundary conditions multiplier increment (the multiplier runs from 0 to 10 with an initial increment 0.1, maximal increment 0.2 and minimal increment 0.001). Time parameter in this case counts the number of successful steps.

In[183]:=
```
(*Mathematica - AceFEM*)
SMTNextStep["λ" → 0.1];
While[
 While[step = SMTConvergence[10^-8, 10, {"Adaptive BC", 8, 0.001, 0.2, 10.}],
  SMTNewtonIteration[];];
 If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
 step[[3]]
 , If[step[[1]], SMTStepBack[];];
 SMTNextStep["Δλ" → step[[2]]]
]
```

```
/*batch mode steering C source code*/
# include "sms.h"
DLLEXPORT int Simulation(SMTStructure *SMT,ElementSpec** ElemSpecs, ElementData** Elements,
NodeSpec** NodeSpecs, NodeData** Nodes, int *IData, double *RData){
int i;
SMT->ConvergenceOptions.MinIncrement = 0.001;
SMT->ConvergenceOptions.MaxIncrement = 0.2;
SMT->ConvergenceOptions.Target = 10;
SMT->NextStep(0, 0.1,0);
while(SMT->ConvergenceReturn.StepForward){
    while(SMT->Convergence(1.e-8, 10,"Adaptive BC"))SMT->NewtonIteration();
    if(strcmp(SMT->ConvergenceReturn.Report,"MinBound")==0){
        SMT->TimeStamp("Divergence dl<dlmin");
        break;
    };
    if(SMT->ConvergenceReturn.StepBack) SMT->StepBack();
    if(SMT->ConvergenceReturn.StepForward) SMT->NextStep(0,SMT-
>ConvergenceReturn.Increment,0);
};
SMT->DumpState("batch"); /*dump the state for visialisation*/
return 1;
```

Selecting nodes and elements in batch mode

MathLink mode	batch mode
SMTFindNodes[*NodeID*]	*NNodesFound*=SMT−>FindNodes (char ∗*NodeID*, int ∗∗*NodesFound*)
SMTFindNodes[Polygon[{T_1,T_2,…,T_n}]]	*NNodesFound*=SMT−> FindNodesPolygon (double ∗∗*PolygonPoints*, int *NPolygonPoints*, "", int ∗∗*NodesFound*)
SMTFindNodes[Polygon[{T_1,T_2,…,T_n},*NodeID*]]	*NNodesFound*=SMT−>FindNodesPolygon (double ∗∗*PolygonPoints*, int *NPolygonPoints*, char ∗*NodeID*, int ∗∗*NodesFound*)

Example: see also Selecting Nodes

- Create a list of indices of all nodes inside the triangle {{0,0},{1,0},{0,1}} in MathLink and batch mode.

In[185]:=
```
(*Mathematica - AceFEM*)
SMTFindNodes[Polygon[{{0, 0}, {1, 0}, {0, 1}}]]
```

```
/*batch mode C code*/
double p1[]={0,0},p2[]={1,0},p3[]={0,1};
```

```
double *PolygonPoints[]={p1,p2,p3};
int NNodesFound,*NodesFound;
NNodesFound=SMT->FindNodesPolygon[PolygonPoints,3,"",&NodesFound];
...
SMT->Free(NodesFound);
```

MathLink mode	batch mode
SMTFindElements[{*Nodes_List, dID_String*}]	*NElementsFound*= SMT−>FindElements (int ∗*Nodes*,int *NNodes*, char ∗*dID*, int ∗∗*ElementsFound*)
SMTFindElements[{*Nodes_List*, All}]	*NElementsFound*=SMT−>FindElements (int ∗*Nodes*, int *NNodes*, "", int ∗∗*ElementsFound*)
SMTFindElements[*dID_String*]	*NElementsFound*=SMT−>FindElements (NULL, 0, char ∗ *dID*, int ∗∗*ElementsFound*)

Example: see also `Selecting Elements`

- Create a list of indexes of all elements inside the triangle {{0,0},{1,0},{0,1}} in MathLink and batch mode.

In[186]:=
```
(*Mathematica - AceFEM*)
SMTFindElements[{Polygon[{{0, 0}, {1, 0}, {0, 1}}], All}]
```

```
/*batch mode C code*/
double p1[]={0,0},p2[]={1,0},p3[]={0,1};
double *PolygonPoints[]={p1,p2,p3};
int NNodesFound,*NodesFound,NElementsFound,*ElementsFound;
NNodesFound=SMT->FindNodesPolygon[PolygonPoints,3,"",&NodesFound];
NElementsFound=SMT->FindElements[NodesFound,NNodesFound,"",&ElementsFound];
...
SMT->Free(NodesFound);
SMT->Free(ElementsFound);
```

CHAPTER 4

AceFEM Examples

Standard performance evaluation tests

```
processorType = "Intel Core(TM) i7-CPU Q720 1.60GHz, 16.0GB RAM, 4 cores, WIndows 7";

<< AceFEM`;
```

Test A: General analysis

```
SetAttributes[getMemoryAndTime, HoldAll];
getMemoryAndTime[i_, k_, j_] := i → StringForm["Time: ``s, MemoryInUse: ``KBytes",
   NumberForm[AbsoluteTiming[j][[1]] / k, 3], (MemoryInUse[] - memoryStart) / 10^3. // Round]
memoryStart = MemoryInUse[];

SMTInputData["Threads" → 1];
meshStat = getMemoryAndTime["Mesh generation", 1,
   L = 100; B = 20; H = 40;
   points =
    {{0, 0, 0}, {L, 0, 0}, {L, B, 0}, {0, B, 0}, {0, 0, H}, {L, 0, H}, {L, B, H}, {0, B, H}};
   SMTAddDomain["A", element = StringJoin[{"ML:", "SE", "D3", "H1",
       "ES", "HY", "H1E9", "D", {{"NeoHooke", "WA"}}}], {}];
   SMTAddMesh[Hexahedron[points], "A", "Division" -> 8 {10, 2, 4}];
   SMTAddEssentialBoundary[
    Polygon[{{0, 0, 0}, {0, B, 0}, {0, B, H}, {0, 0, H}}], 1 → 0, 2 → 0, 3 → 0];
   SMTAddEssentialBoundary[Polygon[{{L, 0, 0}, {L, B, 0}, {L, B, H}, {L, 0, H}}], 3 → -1];
];

setDriverStat = getMemoryAndTime["Set driver", 1, SMTAnalysis["Solver" → 5]];
SMTNextStep[1, 1];
iterationStat = getMemoryAndTime["NR Iteration", 10, Table[SMTNewtonIteration[], {10}];];
visualization1Stat = getMemoryAndTime["ShowMesh - first time", 1, SMTShowMesh[]];
visualization2Stat = getMemoryAndTime["ShowMesh", 1, SMTShowMesh[]];

TestResultsA = Join[
   {"MMA version" -> $Version, "Ace version" -> SMCSession[[3]], "Processor" -> processorType,
    "No. cores" -> $ProcessorCount, "Element" → StringJoin[element],
    meshStat, setDriverStat, iterationStat, visualization1Stat, visualization2Stat},
   SMTSimulationReport["Output" → {}]];

SMCPrintReport[TestResultsA,
 Grid[Apply[List, TestResultsA, {1}], Frame → All, Alignment → Left], True]
```

TestResultsA

MMA version	11.0.1 for Microsoft Windows (64-bit) (September 20, 2016)
Ace version	6.816 Windows (8 May 17)
Processor	Intel Core(TM) i7-CPU Q720 1.60GHz, 16.0GB RAM, 4 cores, WIndows 7
No. cores	4
Element	ML:SED3H1ESHYH1E9DNeoHookeWA
Mesh generation	Time: 2.97s, MemoryInUse: 17982KBytes
Set driver	Time: 13.7s, MemoryInUse: 45060KBytes
NR Iteration	Time: 39.8s, MemoryInUse: 45102KBytes
ShowMesh - first time	Time: 4.94s, MemoryInUse: 50487KBytes
ShowMesh	Time: 0.216s, MemoryInUse: 50465KBytes
No. of nodes	45 441
No. of elements	40 960
No. of equations	134 079
Number of threads used/max	1/8
Data memory (KBytes)	172 080
Tangent matrix (KBytes)	41 050
Solver memory (KBytes)	1 106 492
Total driver (KBytes)	1 319 622
MMA kernel memory (KBytes)	120 791
MMA front end memory(KBytes)	112 640
Total memory (KBytes)	1 553 053
Solver type	Pardiso
Matrix type	-2
No. of steps	1
No. of steps back	0
Step efficiency (%)	100.
Total no. of iterations	10
Average iterations/step	10.
Terminal BC multiplier (λ)	1.
Terminal time (t)	1.
Terminal parameter (γ)	0.
Total absolute time (s)	420.9163394
Total driver time (s)	403.93
Total driver time (%)	95.9644
Total linear solver time (s)	336.505
Total linear solver time (%)	79.9458
Total K&R time (s)	62.062
Total K&R time (%)	14.7445
Average time/iteration (s)	40.393
Average linear solver time (s)	33.6505
Average Ke&Re time (s)	0.000151519
CPU Mathematica time (s)	17.284
CPU Mathematica time (%)	4.10628

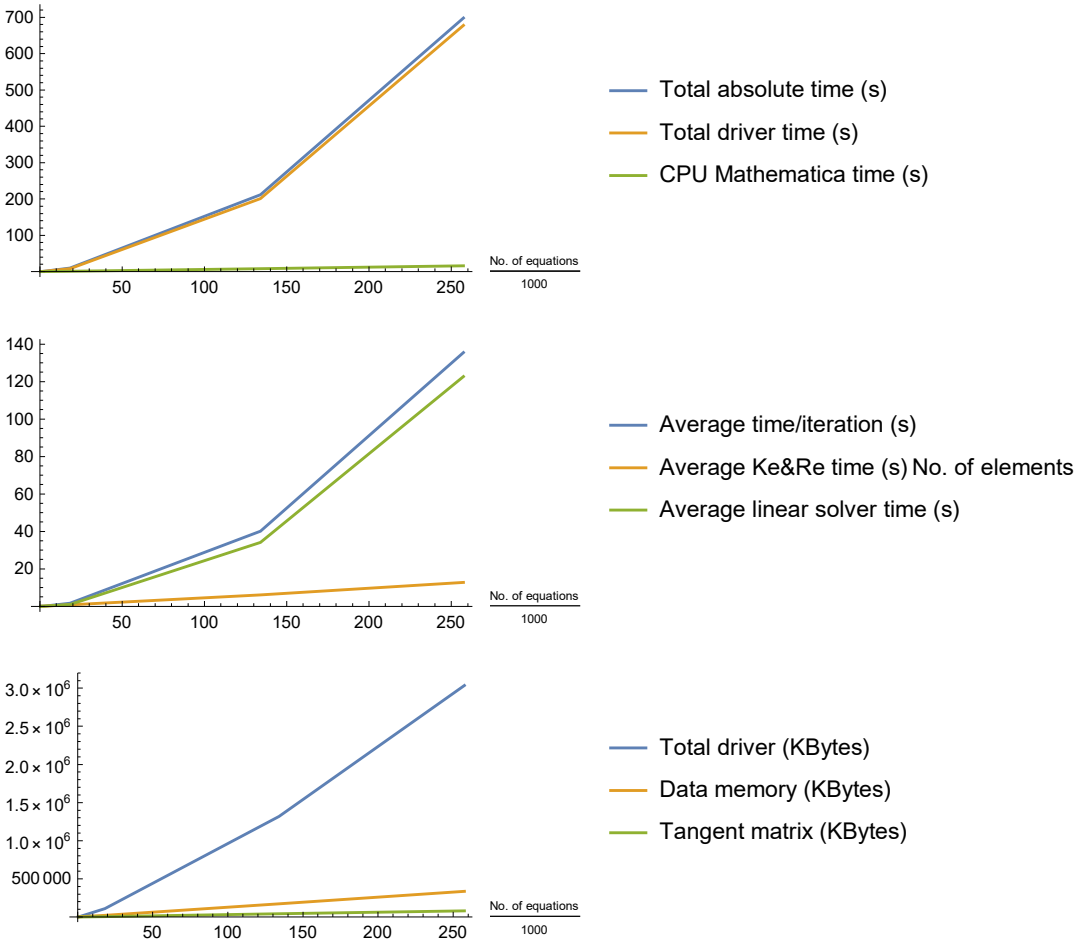
Test B: Mesh size benchmark

```
testDataB = {"No. of equations"/1000, "No. of elements", "Total absolute time (s)",
   "Total driver time (s)", "CPU Mathematica time (s)", "Average time/iteration (s)",
   "No. of elements" "Average Ke&Re time (s)", "Average linear solver time (s)",
   "Total driver (KBytes)", "Data memory (KBytes)", "Tangent matrix (KBytes)"};
```

```
TestResultsB = Table[
   SMTInputData["Threads" → 1];
   L = 100; B = 20; H = 40;
   points =
    {{0, 0, 0}, {L, 0, 0}, {L, B, 0}, {0, B, 0}, {0, 0, H}, {L, 0, H}, {L, B, H}, {0, B, H}};
   SMTAddDomain["A", element = StringJoin[{"ML:", "SE", "D3", "H1",
       "ES", "HY", "H1E9", "D", {{"NeoHooke", "WA"}}}], {}];
   SMTAddMesh[Hexahedron[points], "A", "Division" -> meshDensity {10, 2, 4}];
   SMTAddEssentialBoundary[
    Polygon[{{0, 0, 0}, {0, B, 0}, {0, B, H}, {0, 0, H}}], 1 → 0, 2 → 0, 3 → 0];
   SMTAddEssentialBoundary[Polygon[{{L, 0, 0}, {L, B, 0}, {L, B, H}, {L, 0, H}}], 3 → -1];
   SMTAnalysis["Solver" → 5];
   SMTNextStep[1, 1];
   Table[SMTNewtonIteration[], {nNRIterations = 5}];
   simReport = SMTSimulationReport["Output" → {}];
   {Map[# /. simReport &, testDataB], simReport}
   , {meshDensity, {1, 2, 4, 8, 10}}
  ];
SMCPrintReport[
 TestResultsB,
 Column[{
   Grid[{{"Element", StringJoin[element]}, {"MMA version", $Version},
     {"Ace version", SMCSession[[3]]}, {"Processor", processorType}},
    Frame → All, Alignment → Left],
   ListLinePlot[Table[TestResultsB[[All, 1, {1, i}]], {i, {3, 4, 5}}],
    PlotLegends → testDataB[[{3, 4, 5}]],
    AxesLabel → {testDataB[[1]], ""}, ImageSize → 300, PlotRange → All],
   ListLinePlot[Table[TestResultsB[[All, 1, {1, i}]], {i, {6, 7, 8}}],
    PlotLegends → testDataB[[{6, 7, 8}]],
    AxesLabel → {testDataB[[1]], ""}, ImageSize → 300, PlotRange → All],
   ListLinePlot[Table[TestResultsB[[All, 1, {1, i}]], {i, {9, 10, 11}}],
    PlotLegends → testDataB[[{9, 10, 11}]],
    AxesLabel → {testDataB[[1]], ""}, ImageSize → 300, PlotRange → All]
  }],
 True]
```

Element	ML:SED3H1ESHYH1E9DNeoHookeWA
MMA version	11.0.1 for Microsoft Windows (64-bit) (September 20, 2016)
Ace version	6.816 Windows (8 May 17)
Processor	Intel Core(TM) i7-CPU Q720 1.60GHz, 16.0GB RAM, 4 cores, WIndows 7

TestResultsB



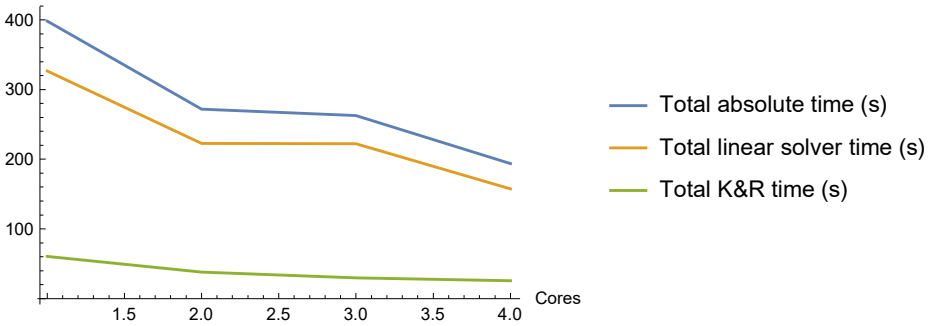
Test C: Parallelization benchmark

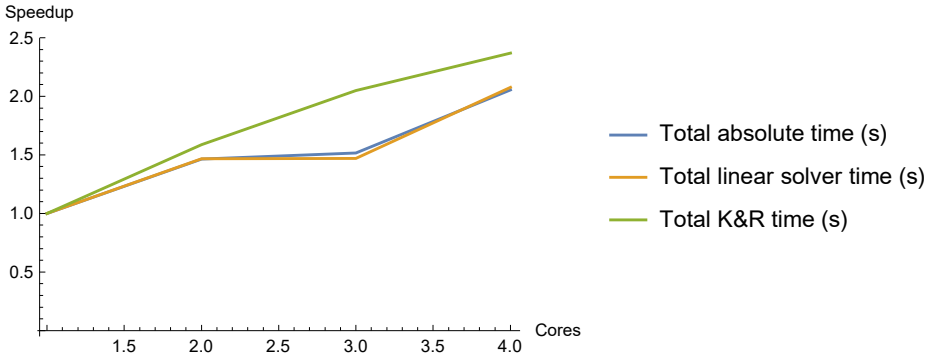
```
testDataC =
  {"Cores", "Total absolute time (s)", "Total linear solver time (s)", "Total K&R time (s)"};
```

```
TestResultsC = Table[
   SMTInputData["Threads" → cores];
   L = 100; B = 20; H = 40;
   points =
    {{0, 0, 0}, {L, 0, 0}, {L, B, 0}, {0, B, 0}, {0, 0, H}, {L, 0, H}, {L, B, H}, {0, B, H}};
   SMTAddDomain["A", element = StringJoin[{"ML:", "SE", "D3", "H1",
       "ES", "HY", "H1E9", "D", {{"NeoHooke", "WA"}}}], {}];
   SMTAddMesh[Hexahedron[points], "A", "Division" -> 8 {10, 2, 4}];
   SMTAddEssentialBoundary[
    Polygon[{{0, 0, 0}, {0, B, 0}, {0, B, H}, {0, 0, H}}], 1 → 0, 2 → 0, 3 → 0];
   SMTAddEssentialBoundary[Polygon[{{L, 0, 0}, {L, B, 0}, {L, B, H}, {L, 0, H}}], 3 → -1];
   SMTAnalysis["Solver" → 5];
   SMTNextStep[1, 1];
   Table[SMTNewtonIteration[], {10}];
   simReport = SMTSimulationReport["Output" → {}];
   {Map[# /. simReport /. {"Cores" -> cores} &, testDataC], simReport}
   , {cores, $ProcessorCount}
  ];
SMCPrintReport[TestResultsC,
 Column[{
   Grid[{{"Element", StringJoin[element]}, {"MMA version", $Version},
     {"Ace version", SMCSession[[3]]}, {"Processor", processorType},
     {"No. cores", $ProcessorCount}}, Frame → All, Alignment → Left],
   ListLinePlot[Table[TestResultsC[[All, 1, {1, i}]], {i, {2, 3, 4}}],
    PlotLegends → testDataC[[{2, 3, 4}]],
    AxesLabel → {testDataC[[1]], ""}, ImageSize → 300, PlotRange → All],
   ListLinePlot[Table[{TestResultsC[[All, 1, 1]],
       TestResultsC[[1, 1, i]] / TestResultsC[[All, 1, i]]} // Transpose, {i, {2, 3, 4}}],
    PlotLegends → testDataC[[{2, 3, 4}]], AxesLabel → {testDataC[[1]], "Speedup"},
    ImageSize → 300, PlotRange → All]
  }],
 True]
```

Element	ML:SED3H1ESHYH1E9DNeoHookeWA
MMA version	11.0.1 for Microsoft Windows (64-bit) (September 20, 2016)
Ace version	6.816 Windows (8 May 17)
Processor	Intel Core(TM) i7-CPU Q720 1.60GHz, 16.0GB RAM, 4 cores, WIndows 7
No. cores	4

TestResultsC





Summary of benchmark tests

```
Print[Grid[{
    {"MMA version", $Version},
    {"Ace version", SMCSession[[3]]}, {"Processor", processorType},
    {"Test B:", Row[{"CDriver(KBytes):", TestResultsB[[-1, 1, 9]], ", K&R(s):",
       TestResultsB[[-1, 1, 7]], ", Solver(s):", TestResultsB[[-1, 1, 8]]}]},
    {"Test C: ", Row[{"Speedup of total time:", Max[TestResultsC[[1, 1, 2]] / TestResultsC[[
          All, 1, 2]]], ", solver:", Max[TestResultsC[[1, 1, 3]] / TestResultsC[[All, 1, 3]]],
       ", K&R assembly:", Max[TestResultsC[[1, 1, 4]] / TestResultsC[[All, 1, 4]]]}]}
   }
   , Frame → All, Alignment → Left]];
```

Typical results

MMA version	11.1.1 for Microsoft Windows (64-bit) (April 18, 2017)
Ace version	6.808 Windows (5 Sep 16)
Processor	Intel Xeon E5-2650 2GHz 2 processors, 64.0 GB RAM, 16 cores, WIndows 10
Test B:	CDriver(KBytes):Total driver (KBytes), K&R(s):12.6252, Solver(s):74.9332
Test C:	Speedup of total time:6.68745571, solver:8.17603, K&R assembly:10.0056

MMA version	11.1.1 for Microsoft Windows (64-bit) (April 18, 2017)
Ace version	6.816 Windows (8 May 17)
Processor	Intel Xeon E5-2650 2GHz 2 processors, 64.0 GB RAM, 16 cores, WIndows 10
Test B:	CDriver(KBytes):3036718, K&R(s):12.7362, Solver(s):74.3608
Test C:	Speedup of total time:6.88599086, solver:8.48472, K&R assembly:10.131

MMA version	11.1.0 for Microsoft Windows (64-bit) (March 13, 2017)
Ace version	6.817 Windows (10 May 17)
Processor	Intel Core(TM) i7-3610QM CPU @ 2.30GHz 2.30 GHz, 8.0 GB RAM, 4 cores
Test B:	CDriver(KBytes):3036718, K&R(s):9.569, Solver(s):52.809
Test C:	Speedup of total time:2.25387642, solver:2.2337, K&R assembly:2.86931

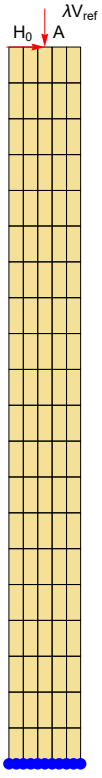
MMA version	11.0.1 for Microsoft Windows (64-bit) (September 20, 2016)
Ace version	6.816 Windows (8 May 17)
Processor	Intel Core(TM) i7-CPU Q720 1.60GHz, 16.0GB RAM, 4 cores, WIndows 7
Test B:	CDriver(KBytes):3036718, K&R(s):12.7898, Solver(s):122.694
Test C:	Speedup of total time:2.053319518, solver:2.07504, K&R assembly:2.36883

Simple bending of the column

Calculate the response of the clamped column dimensions B × L = 2 × 10 subjected to the constant horizontal force $H_0 = 10$ and variable vertical force $V = -\lambda V_{ref}$ in point A = (0, L) at the top center of the column where $\lambda \in [0, \lambda_{Max}]$ is the load multiplier (parameter of the problem) and $V_{ref} = 10$ is the reference force. Due to the high non-linearity of the problem an adaptive load stepping strategy is used.

In[230]:=
```
<< AceFEM`;
SMTInputData[];
Vref = 10; H0 = 10; L = 20; B = 2;
SMTAddDomain["Ω", {"ML:", "SE", "PS", "Q2", "DF", "HY", "Q2", "D", {{"NeoHooke", "WA"}}},
  {"E *" -> 21000, "ν *" -> 0.3}];
SMTAddEssentialBoundary[Line[{{-B/2, 0}, {B/2, 0}}], 1 -> 0, 2 -> 0];
SMTAddNaturalBoundary[Point[{0, L}], 2 -> -Vref];
SMTAddInitialBoundary[Point[{0, L}], 1 -> H0];
SMTAddMesh[Polygon[{{B/2, 0}, {B/2, L}, {-B/2, L}, {-B/2, 0}}], "Ω", "Division" -> {20, 5}];
SMTAnalysis[];
```

In[200]:=
```
Show[SMTShowMesh["BoundaryConditions" → True],
 Graphics[{Text["A", {0.4, L + .4}], Text["H0", {-0.6, L + .4}], Text["λVref", {1., L + 1}]}]]
```



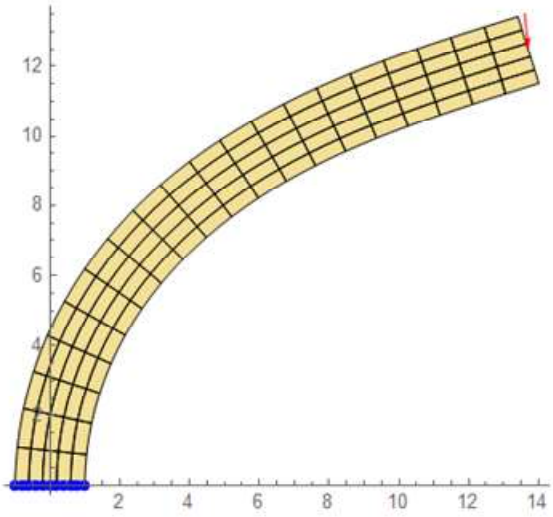
In[201]:=
```
λMax = 10; λ0 = λMax / 100; ΔλMin = λMax / 1000; ΔλMax = λMax / 10;
tolNR = 10^-8; maxNR = 15; targetNR = 8;
SMTNextStep["λ" → λ0];
While[
  While[
   step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λMax}]
   , SMTNewtonIteration[];
  ];
  If[Not[step[[1]]], SMTAnimationOfResponse["LeadingNodePosition" → {0, L}]];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" → step[[2]]]
 ];
```

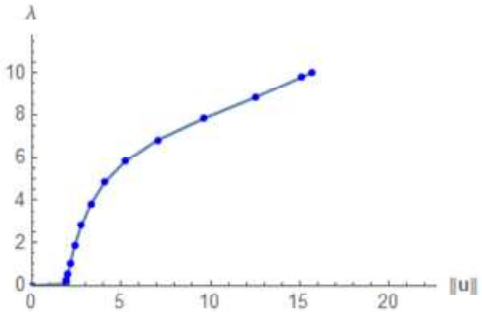
In[205]:=
```
SMTSimulationReport[];
```

No. of nodes	451
No. of elements	100
No. of equations	880
Number of threads used/max	8/8
Data memory (KBytes)	163
Tangent matrix (KBytes)	105
Solver memory (KBytes)	1328
Total driver (KBytes)	1596
MMA kernel memory (KBytes)	109 578
MMA front end memory (KBytes)	993 829
Total memory (KBytes)	1 105 003
Solver type	Pardiso
Matrix type	−2
No. of steps	14
No. of steps back	1
Step efficiency (%)	93.3333
Total no. of iterations	105
Average iterations/step	7.
Terminal BC multiplier (λ)	10.
Terminal time (t)	0.
Terminal parameter (γ)	0.

Total absolute time (s)	3.3602399
Total driver time (s)	2.631
Total driver time (%)	78.298
Total linear solver time (s)	0.107
Total linear solver time (%)	3.18429
Total K&R time (s)	0.0549996
Total K&R time (%)	1.63678
Average time/iteration (s)	0.0250571
Average linear solver time (s)	0.00101905
Average Ke&Re time (s)	5.23806×10^{-6}
CPU Mathematica time (s)	1.562
CPU Mathematica time (%)	46.4848

In[206]:=
```
SMTAnimationOfResponse["Last frame"]
```





Bending of the column (path following procedure, animations, 2D solids)

Calculate the response of the clamped column dimensions B × L = 2 × 10 subjected to the constant horizontal force $H_0 = 10$ and variable vertical force $V = -\lambda V_{ref}$ in point A = (0, L) at the top center of the column where $\lambda \in [0, \lambda_{Max}]$ is the load multiplier (parameter of the problem) and $V_{ref} = 10$ is the reference force. Due to the high non-linearity of the problem an adaptive load stepping strategy is used.

In[239]:=
```
<< AceFEM`;
SMTInputData[];
Vref = 10; H0 = 10; L = 20; B = 2;
SMTAddDomain["Ω", {"ML:", "SE", "PS", "Q2", "DF", "HY", "Q2", "D", {{"NeoHooke", "WA"}}},
  {"E *" -> 21000, "ν *" -> 0.3}];
SMTAddEssentialBoundary[Line[{{-B/2, 0}, {B/2, 0}}], 1 -> 0, 2 -> 0];
SMTAddNaturalBoundary[Point[{0, L}], 2 -> -Vref];
SMTAddInitialBoundary[Point[{0, L}], 1 -> H0];
SMTAddMesh[Polygon[{{B/2, 0}, {B/2, L}, {-B/2, L}, {-B/2, 0}}], "Ω", "Division" -> {20, 5}];
SMTAnalysis[];
```

In order to access the response of column the following response curves are calculated and collected during the simulation:

- λucurve ≡ $u_{(0,20)}(\lambda)$
  The *λucurve* curve represents a horizontal displacement in point (0,20) as a function of load level λ. The points on a curve are calculated by the following command
  AppendTo[λucurve, { SMTPostData["u", Point[{0, 20}]],SMTData["Multiplier"]}].
- σEcurve ≡ $\sigma_{yy(0,10)}(\epsilon_{yy(0,10)})$
  The σEcurve curve represents σ_{yy} stress in point (0,10) as a function of a ϵ_{yy} strain in point (0,10). The points on a curve are calculated by the following command
  AppendTo[σEcurve, SMTPostData[{"Eyy", "Syy"}, Point[{0, 10}]]]
- λRcurve ≡ $R_{y\,|\,y==0}(\lambda)$
  The λRcurve curve represents the total reaction force in Y direction as a function of load level λ. The points on a curve are calculated by the following command
  AppendTo[λRcurve, {Total[SMTResidual[Line[{{-B/2,0},{B/2,0}}]]][[2]], SMTData["Multiplier"]}]
- λMcurve ≡ $M_z(\lambda)$
  The λMcurve curve represents the bending moment at the clamped end of the column as a function of load level λ. The moment is evaluated in three different ways resulting in three curves λMcurve1, λMcurve2 and λMcurve3 as follows:
- λMcurve1
  The bending moment at the clamped end of the column is by definition
  $M_z = \int_{-B/2}^{B/2} \sigma_{yy}(x)\, x\, dx$.
  This integral can be numerically evaluated directly using the *Mathematica* function for the numerical integration NIntegrate of an arbitrary function and the AceFEM function SMTPostData["Syy", Point[{x, y}]] that evaluates the stress S_{yy} in an arbitrary point (x, y). The points on the λMcurve1 curve are then calculated by the following command
  AppendTo[λMcurve1, {-NIntegrate[σx[x] x, {x, -1, 1}, MaxPoints -> 10], SMTData["Multiplier"]}].
  The reason for an additional definition of the function σx[x] as σx[x_?NumericQ] := SMTPostData["Syy", Point[{x, 0}]] is to prevent symbolic evaluation of the function SMTPostData that obviously has only numerical values.
- λMcurve2
  The bending moment at the clamped end of the column can also be evaluated as a sum of moments of reaction forces $R_{y,i}$ in the nodes at the clamped and of the column
  $M_z = \sum_{i=1}^{n} R_{y,i}\, x_i$
  where *n* is the number of nodes at "Y"==0, $\mathbf{R}_{y,i}$=SMTResidual["Y" == 0 &][[i, 2]] and x_i = SMTNodeData["Y" == 0 &, "X"][[i, 1]]. The points on the λMcurve2 curve are calculated by the following command
  AppendTo[λMcurve2,{Total[SMTResidual[Line[{{-B/2,0},{B/2,0}}]]][[All,2]] SMTNodeData[Line[{{-B/2,0},{B/2,0}}],"X"][[All,1]],SMTData["Multiplier"]}].

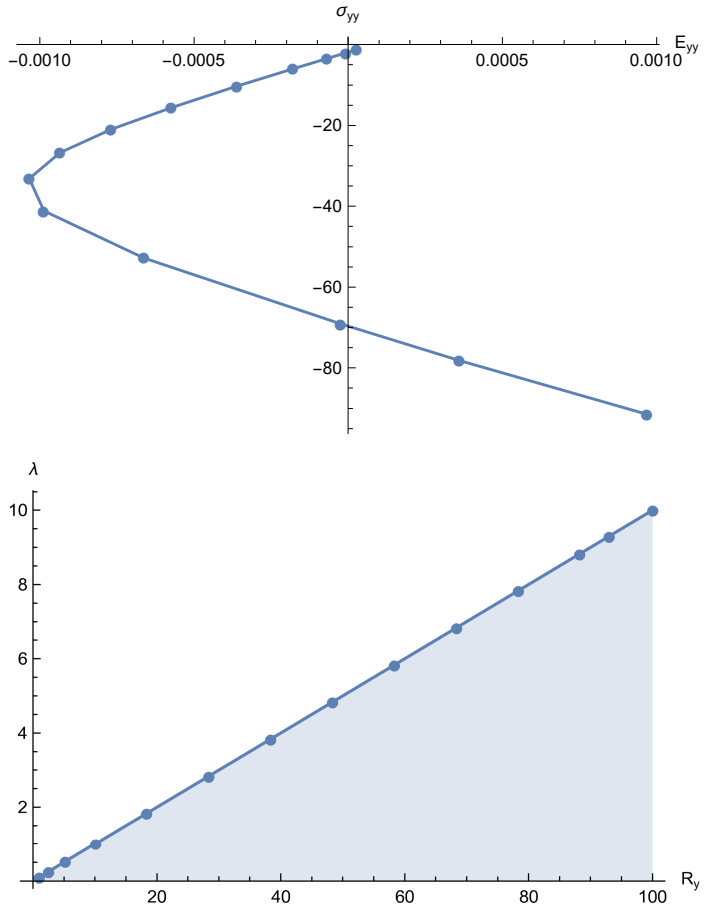
- λMcurve3
  The bending moment can also be obtained from the global equilibrium as
  $M_z = u_A\, \lambda\, V_{ref} + (L + v_A)\, H_0$.
  The points on the λMcurve3 curve are then calculated by the following command
  AppendTo[λMcurve3,{SMTPostData["u",Point[{0,20}]] SMTData["Multiplier"] Vref +(L+SMTPostData["v",Point[{0,20}]]) H0,SMTData["Multiplier"]}].

Here an adaptive load stepping procedure is employed. The process is controlled by the SMTConvergence command. The points on the response curves are evaluated at the end of each successfully completed step and added to the lists by SMTAnimationOfResponse command.

In[390]:=
```
σx[x_?NumericQ] := SMTPostData["Syy", Point[{x, 0}]]; Off[NIntegrate::"maxp"];
λMax = 10; λ0 = λMax / 100; ΔλMin = λMax / 1000; ΔλMax = λMax / 10;
tolNR = 10^-8; maxNR = 15; targetNR = 8;
SMTNextStep["λ" → λ0];
SMTAnimationOfResponse["Initialize"];
While[
  While[
   step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λMax}]
   , SMTNewtonIteration[];
  ];
  If[Not[step[[1]]],
   SMTAnimationOfResponse[
    (*specify node for which response curve ||u||(λ) is taken*)
    "LeadingNodePosition" → {0, L}
    (*coordinate range for response curve*)
    , "PlotOptions" → {PlotRange → {{0, 20}, {0, 12}}}
    (*coordinate range for animation*)
    , "ShowMeshOptions" → {PlotRange → {{-0.1 L, L}, {-0.1 L, 1.1 L}},
      "Field" → "Syy", "Contour" -> {False, -1000, 1000, 5}}
    (*set image size in printer points*)
    , "ImageSize" → 300
    (*show the frames into separate window and save the frames into file responseFile.h5*)
    , "Show" → "Window" | {"ExportFrames", "responseFile"}
    (*save additional response curves points for all frames*)
    , "Data" → Hold[{
       SMTPostData[{"Eyy", "Syy"}, Point[{0, 10}]]
       (* σEcurve *)
       , {Total[SMTResidual[Line[{{-B / 2, 0}, {B / 2, 0}}]]][[2]], SMTData["Multiplier"]}
       (* λRcurve *)
       , {-NIntegrate[ σx[x] x, {x, -1, 1}, MaxPoints -> 10], SMTData["Multiplier"]}
       (* λMcurve1 *)
       , {-NIntegrate[ σx[x] x, {x, -1, 1}, MaxPoints -> 10], SMTData["Multiplier"]}
       (* λMcurve2 *)
       , {Total[SMTResidual[Line[{{-B / 2, 0}, {B / 2, 0}}]]][[All, 2]]×
          SMTNodeData[Line[{{-B / 2, 0}, {B / 2, 0}}], "X"][[All, 1]], SMTData["Multiplier"]}
       (* λMcurve3 *)
       , {SMTPostData["u", Point[{0, 20}]]×SMTData["Multiplier"] Vref +
         (L + SMTPostData["v", Point[{0, 20}]]) H0, SMTData["Multiplier"]}
      }]
   ]];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" → step[[2]]]
];
```

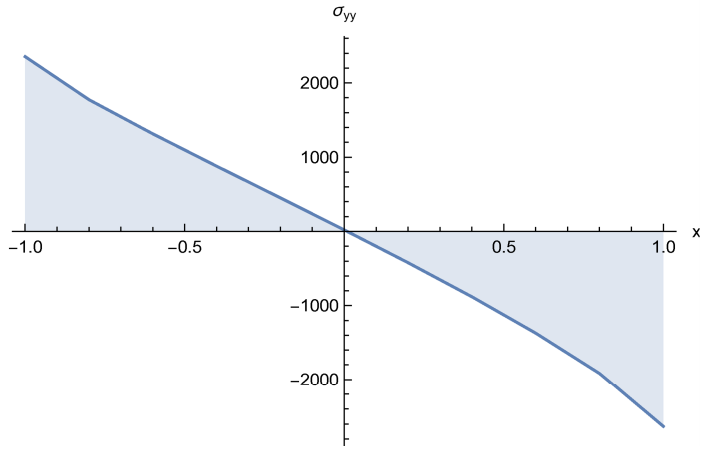
Here the λucurve, σEcurve and λRcurve curves are displayed using the ListLinePlot command.

```
In[396]:= ListLinePlot[SMTAnimationOfResponse["Data"][[All, 1]],
           PlotRange → All, AxesLabel -> {"Eyy", "σyy"}, PlotMarkers → Automatic]
          ListLinePlot[SMTAnimationOfResponse["Data"][[All, 2]], PlotRange → All,
           Filling → Axis, AxesLabel -> {"Ry", "λ"}, PlotMarkers → Automatic]
```



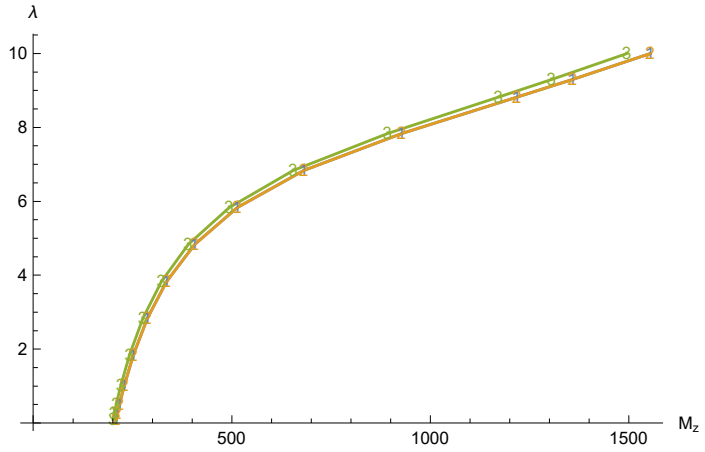
Here is given the distribution of the stress σ_{yy} in cross section Y==0 at the end of simulation ($\lambda = \lambda_u$).

```
In[398]:= ListLinePlot[Table[{x, σx[x]}, {x, -B / 2, B / 2, .01}], Filling → Axis, AxesLabel -> {"x", "σyy"}]
```



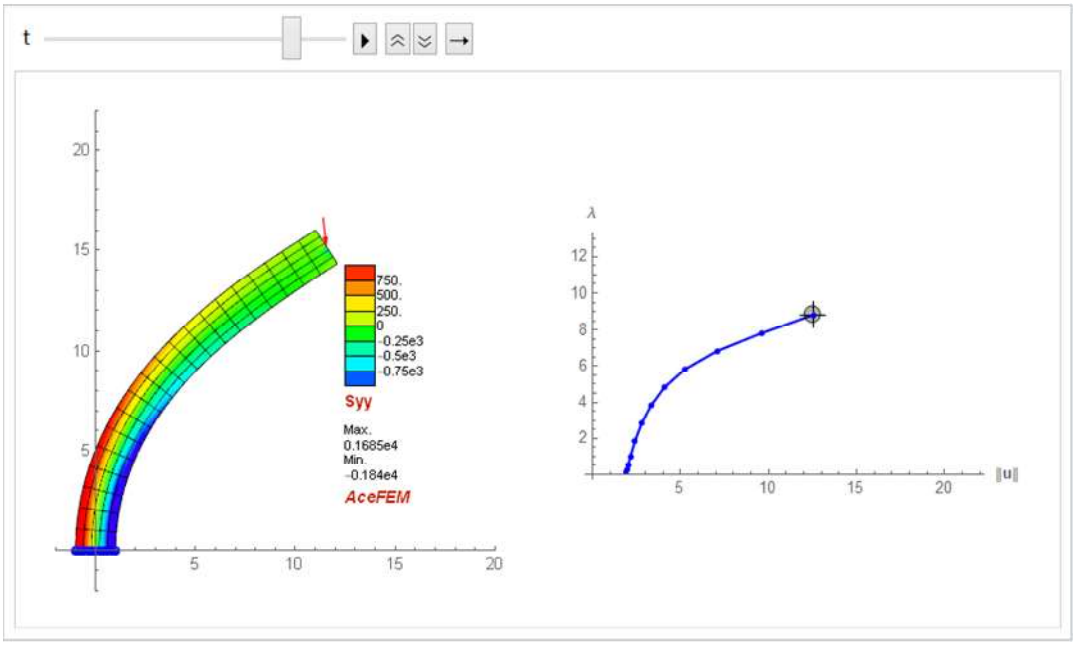
The bending moment at the clamped and of the column was evaluates using three different strategies. The following plot shows that the second and the third strategy yield identical results and the first strategy only slightly different.

In[399]:=
```
ListLinePlot[{SMTAnimationOfResponse["Data"][[All, 3]],
  SMTAnimationOfResponse["Data"][[All, 4]], SMTAnimationOfResponse["Data"][[All, 5]]},
 PlotRange → All, AxesOrigin → {0, 0}, AxesLabel -> {"Mz", "λ"}, PlotMarkers → {"1", "2", "3"}]
```

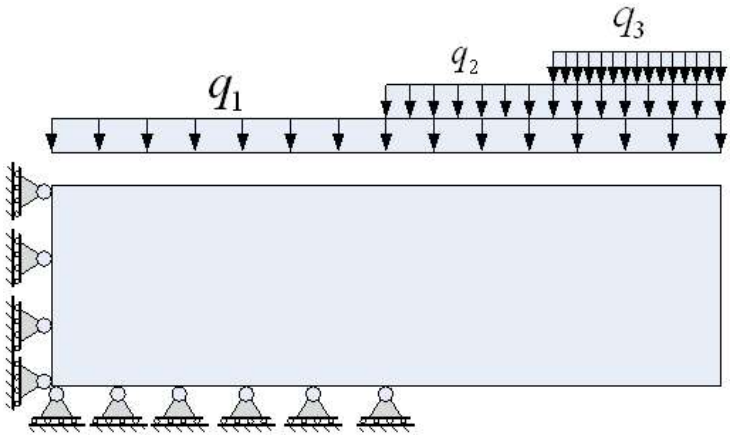


The previously stored frames to "*coulmn.h5*" file are here combined to create animation of the bending of the column.

In[400]:=
```
SMTAnimationOfResponse["Animate"]
```



Boundary conditions (2D solid)



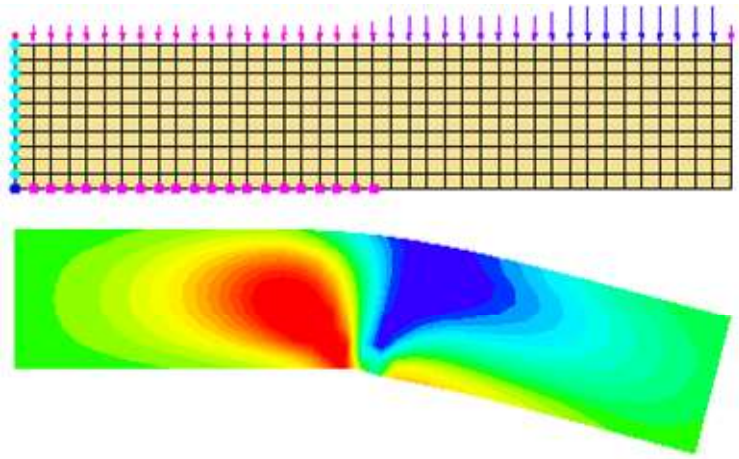
Solution 1 is based on line segment node selector and continuous loads

In[248]:=
```
<< AceFEM`;
SMTInputData[];
L = 100; H = 20;
q1 = 2; q2 = 1; q3 = 1;
nx = 40; ny = 10;
SMTAddDomain["A", {"ML:", "SE", "PS", "Q1", "DF", "HY", "Q1", "D", {{"NeoHooke", "WA"}}},
  {"E *" -> 1000., "ν *" -> .49, "t *" -> 1.}];
SMTAddMesh[Polygon[{{0, 0}, {L, 0}, {L, H}, {0, H}}], "A", "Division" -> {40, 10}];
SMTAddEssentialBoundary[Line[{{0, 0}, {0, H}}], 1 -> 0];
SMTAddEssentialBoundary[Line[{{0, 0}, {L / 2, 0}}], 2 -> 0];
SMTAddNaturalBoundary[Line[{{0, H}, {L, H}}], 2 -> Line[{-q1}]];
SMTAddNaturalBoundary[Line[{{L / 2, H}, {L, H}}], 2 -> Line[{-q2}]];
SMTAddNaturalBoundary[Line[{{3 L / 4, H}, {L, H}}], 2 -> Line[{-q3}]];
SMTAnalysis[];
```

In[247]:=
```
SMTNextStep["λ" → 0.1];
While[
 While[step = SMTConvergence[10^-7, 15, {"Adaptive BC", 8, .01, 0.5, 1}] ,
  SMTNewtonIteration[]];
 SMTStatusReport[];
 If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
 If[Not[step[[1]]], SMTShowMesh["DeformedMesh" → True,
   "Field" → "Sxy", "Mesh" → False, "Contour" → 20, "Show" -> "Window"]];
 step[[3]] ,
 If[step[[1]], SMTStepBack[]];
 SMTNextStep["Δλ" → step[[2]]]
]
```

```
Step/Iter=1/5 λ/Δλ=0.1/0.1 ||Δp||/||R||=2.22119×10^-9/3.19596×10^-12 Events=0 Status=0/{}
Step/Iter=2/6 λ/Δλ=0.281633/0.181633
  ||Δp||/||R||=7.88439×10^-13/2.95438×10^-12 Events=0 Status=0/{}
Step/Iter=3/6 λ/Δλ=0.585589/0.303957
  ||Δp||/||R||=4.3367×10^-9/3.2744×10^-12 Events=0 Status=0/{}
Step/Iter=4/7 λ/Δλ=1./0.414411 ||Δp||/||R||=2.23985×10^-13/3.16998×10^-12 Events=0 Status=0/{}
```

In[249]:=
```
Column[{SMTShowMesh["Show" → False, "BoundaryConditions" → True],
  SMTShowMesh["DeformedMesh" → True, "Mesh" → False,
   "Field" → "Sxy", "Contour" → 20, "Show" → False, "Legend" → False]}]
```



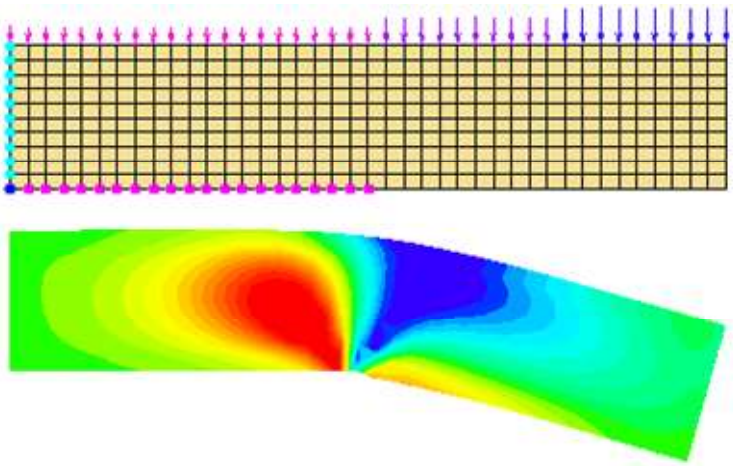
Solution 2 is based on general node selector and calculated nodal forces

In[261]:=
```
<< AceFEM`;
SMTInputData[];
L = 100; H = 20;
q1 = 2; q2 = 1; q3 = 1;
nx = 40; ny = 10;
SMTAddDomain["A", {"ML:", "SE", "PS", "Q1", "ES", "HY", "Q1E4", "D", {{"NeoHooke", "WA"}}},
  {"E *" -> 1000., "ν *" -> .49, "t *" -> 1.}];
SMTAddMesh[Polygon[{{0, 0}, {L, 0}, {L, H}, {0, H}}], "A", "Division" -> {40, 10}];
SMTAddEssentialBoundary["X" == 0 &, 1 -> 0];
SMTAddEssentialBoundary["Y" == 0 && "X" <= L / 2 &, 2 -> 0];
SMTAddNaturalBoundary["Y" == H &, 2 -> -q1 L / nx];
SMTAddNaturalBoundary["Y" == H && "X" > L / 2 &, 2 -> -q2 L / nx];
SMTAddNaturalBoundary["Y" == H && "X" > 3 L / 4 &, 2 -> -q3 L / nx];
SMTAnalysis[];
```

In[263]:=
```
SMTNextStep["λ" → 0.1];
While[
 While[step = SMTConvergence[10^-7, 15, {"Adaptive BC", 8, .01, 0.5, 1}] ,
  SMTNewtonIteration[]];
 SMTStatusReport[];
 If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
 If[Not[step[[1]]], SMTShowMesh["DeformedMesh" → True,
   "Field" → "Sxy", "Mesh" → False, "Contour" → 20, "Show" -> "Window"]];
 step[[3]] ,
 If[step[[1]], SMTStepBack[]];
 SMTNextStep["Δλ" → step[[2]]]
]
```

```
Step/Iter=1/5 λ/Δλ=0.1/0.1 ||Δp||/||R||=3.83173×10^-9/2.80021×10^-12 Events=0 Status=0/{}

Step/Iter=2/6 λ/Δλ=0.281633/0.181633
  ||Δp||/||R||=2.38464×10^-12/2.7859×10^-12 Events=0 Status=0/{}

Step/Iter=3/6 λ/Δλ=0.585589/0.303957
  ||Δp||/||R||=1.30238×10^-8/2.87459×10^-12 Events=0 Status=0/{}

Step/Iter=4/7 λ/Δλ=1./0.414411 ||Δp||/||R||=2.39675×10^-12/2.89293×10^-12 Events=0 Status=0/{}
```

In[265]:=
```
Column[{SMTShowMesh["Show" → False, "BoundaryConditions" → True],
  SMTShowMesh["DeformedMesh" → True, "Mesh" → False,
   "Field" → "Sxy", "Contour" → 20, "Show" → False, "Legend" → False]}]
```



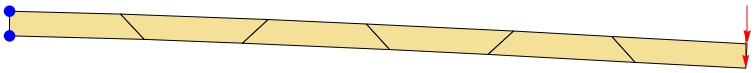
Standard 6-element benchmark test for distortion sensitivity (2D solids)

In[294]:=
```
<< AceFEM`;
```

In[295]:=
```
SMTInputData[];
SMTAddDomain["A", {"ML:", "SE", "PS", "Q1", "DF", "LE", "Q1", "D", "Hooke"},
  {"E *" -> 11. × 10^7, "ν *" -> 0.3, "t *" -> 0.1}];
SMTAddMesh["A"
  , {{1, 0.0, -0.2}, {2, 1.1, -0.2}, {3, 1.9, -0.2}
   , {4, 3.1, -0.2}, {5, 3.9, -0.2}, {6, 5.1, -0.2}
   , {7, 6.0, -0.2}, {8, 0.0, 0}, {9, 0.9, 0}
   , {10, 2.1, 0}, {11, 2.9, 0}, {12, 4.1, 0}
   , {13, 4.9, 0}, {14, 6.0, 0}}
  , {{1, 2, 9, 8}, {2, 3, 10, 9}, {3, 4, 11, 10},
   {4, 5, 12, 11}, {5, 6, 13, 12}, {6, 7, 14, 13}}
 ];
SMTAddEssentialBoundary[Line[{{0, -0.2}, {0, 0}}], 1 -> 0, 2 -> 0];
SMTAddNaturalBoundary[Line[{{6, -0.2}, {6, 0}}] , 2 -> -0.5];
```

In[300]:=
```
SMTAnalysis[];
SMTNextStep["λ" → 1];
While[SMTConvergence[10^-12, 10], SMTNewtonIteration[]];
SMTPostData["v", Point[{6, -0.1}]]
SMTShowMesh["BoundaryConditions" -> True, "DeformedMesh" -> True, "Scale" -> 1000]
```

```
-0.000264431
```



Cyclic tension test

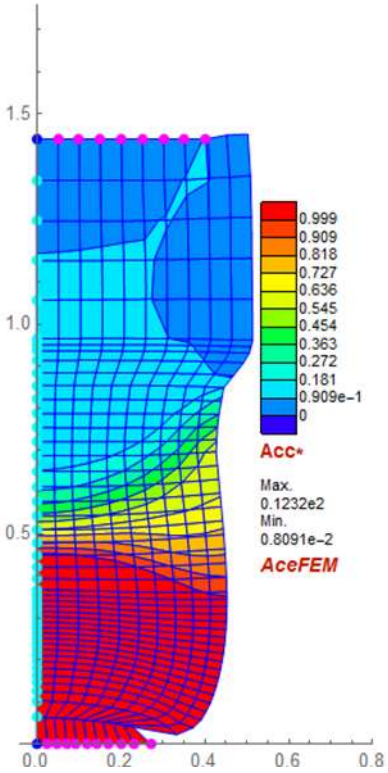
In[287]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["A", "ExamplesFiniteStrain", {"E *" -> 206.9, "ν *" -> .29,
    "σy *" → 0.45, "K *" -> 0.12924, "σyInf *" -> 0.715, "δ *" -> 16.93}];
L = 1.6; R = 0.4; R1 = 0.5; ne = 10;
ΔL = 0.16;
SMTAddMesh[Polygon[{{R, 0}, {R, 0.4}, {0, 0.4}, {0, 0}}], "A", "Q1", {2*ne, ne}];
SMTAddMesh[Polygon[{{R, 0.4}, {R, 0.8}, {0, 0.8}, {0, 0.4}}], "A", "Q1", {ne, ne}];
SMTAddMesh[Raster[{{{0.4, 0.8}, {0.41, 0.89}, {0.43, 0.98}, {0.49, 1.06}, {0.5, 1.1}},
     {{0, 0.8}, {0, 0.89}, {0, 0.98}, {0, 1.06}, {0, 1.1}}}],
  "A", "Q1", {ne, ne}, "InterpolationOrder" → 1];
SMTAddMesh[Polygon[{{R1, 1.1}, {R1, L}, {0, L}, {0, 1.1}}], "A", "Q1", {ne/2, ne}];

SMTAddEssentialBoundary[{Line[{{0, 0}, {0, L}}], 1 -> 0},
  {Line[{{0, 0}, {R, 0}}], 2 -> 0}, { Line[{{0, L}, {R, L}}], 2 -> ΔL }];
SMTAnalysis[];
```

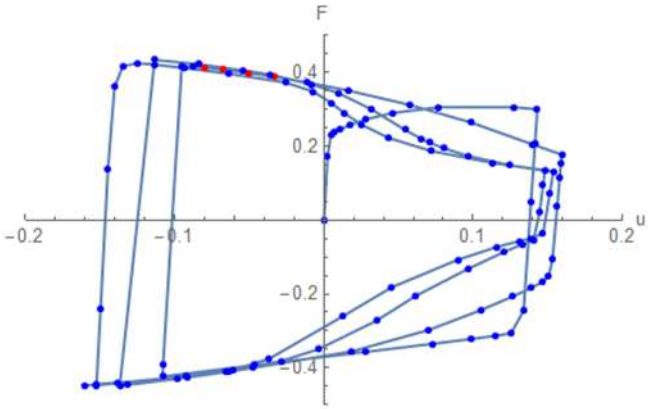
In[298]:=
```
Clear[λ]; λ[t_] := If[OddQ[Floor[(t + 1)/2]], 1, -1] (2 Floor[(t + 1)/2] - t);
```

In[299]:=
```
tMax = 15.; t0 = tMax/500.; ΔtMin = tMax/10000.; ΔtMax = tMax/10.;
tolNR = 10.^-8; maxNR = 15; targetNR = 8;
SMTNextStep["t" → t0, "λ[t]" → λ];
While[
  While[step = SMTConvergence[tolNR, maxNR, {"Adaptive Time", targetNR, ΔtMin, ΔtMax, tMax}],
   SMTNewtonIteration[];];
  If[Not[step[[1]]]
   , SMTAnimationOfResponse["x" -> Hold[SMTRData["Multiplier"] ΔL],
     "y" → Hold[Total[ SMTResidual[Line[{{0, L}, {0.5, L}}]]][[2]]],
     "Show" → "Window" | {"ExportFrames", "cyclic"},
     "ShowMeshOptions" → {"Field" → "Acc*", "Contour" → {0, 1, 10},
       PlotRange → {{-.1 R, 2 R}, {0, L + ΔL}}},
     "PlotOptions" → {AxesLabel → {"u", "F"}, PlotRange → {{-0.2, 0.2}, {-0.5, 0.5}}},
     "RealTime" → Hold[SMTRData["Time"]]
    ];
  ];
  If[step[[4]] === "MinBound", SMTStatusReport["ΔT<ΔT_min"]; SMTStepBack[];];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δt" → step[[2]], "λ[t]" → λ]
 ];
```

- This shows the latest frame evaluated.

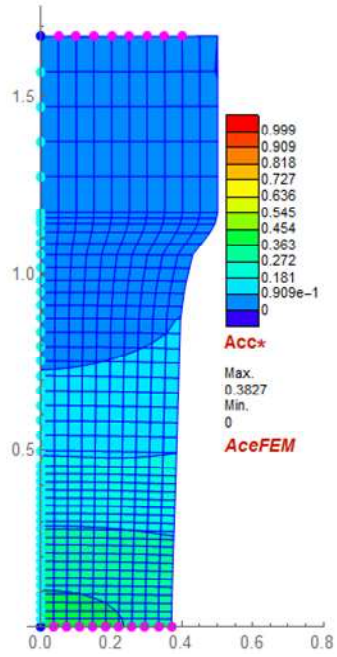
In[284]:=
```
SMTAnimationOfResponse["Last frame"]
```

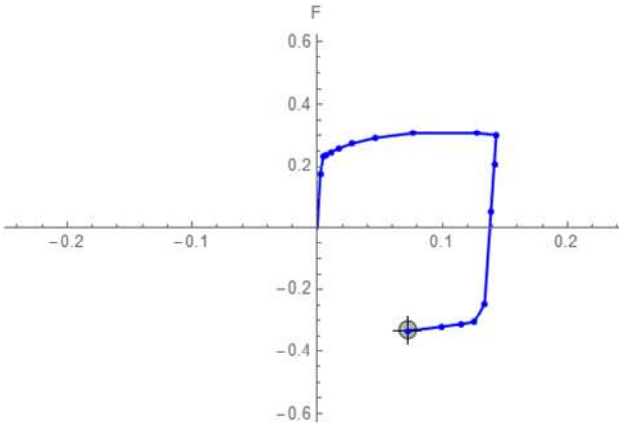




- This shows the frame nearest to real time 1.5.

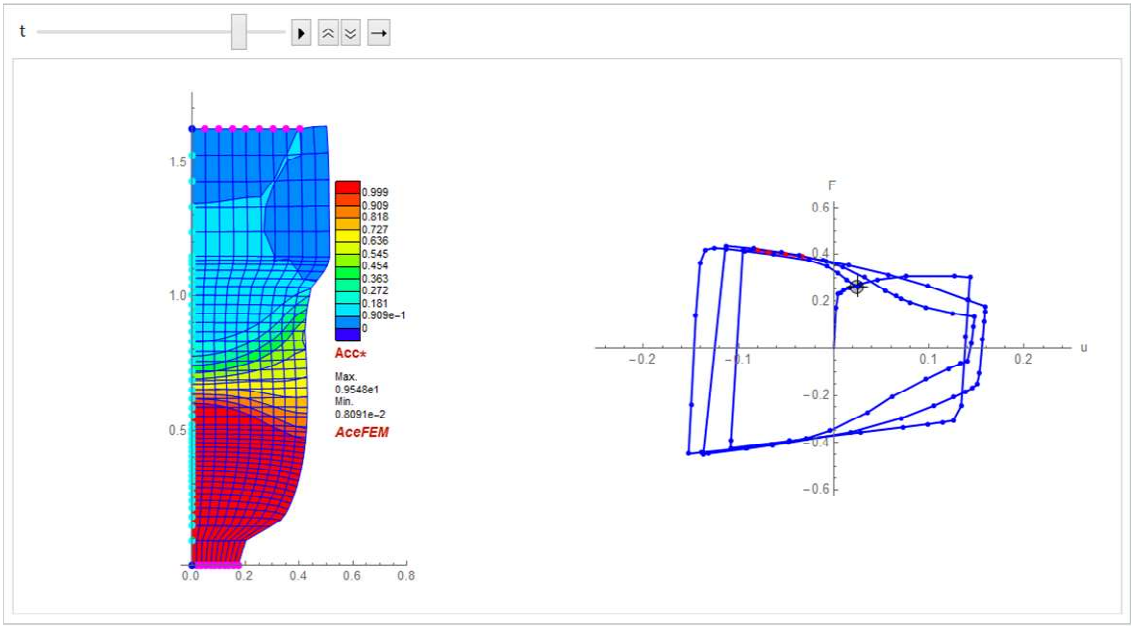
In[303]:=
```
SMTAnimationOfResponse["Report", 1.5]
```





- This creates a cell with controls added to allow interactive manipulation of animation within the notebook.

In[304]:=
```
SMTAnimationOfResponse["Animate"]
```



- This creates the animated GIF file from the frames stored during simulation.

In[305]:=
```
SMTAnimationOfResponse["Export to", "gif"]
```

```
cyclic.gif
```

- This runs animation using the operating systems default player for GIF files if there is any available.

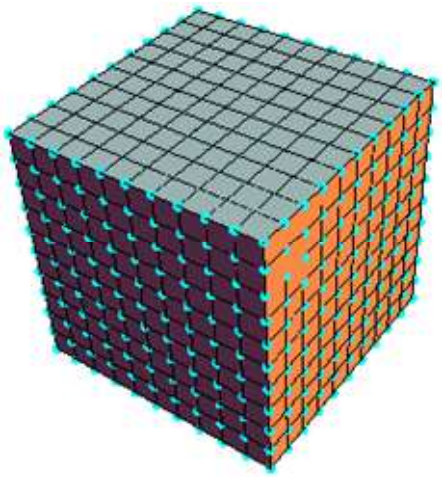
In[306]:=
```
Run["cyclic.gif"]
```

Solution Convergence Test

The domain of the problem is [-.0.5,0.5]×[-0.5,0.5]×[0,1] cube. On all sides, apart from the upper surface, the constant temperature ϕ=0 is prescribed. The upper surface is isolated so that there is no heat flow over the boundary ($\bar{q}$=0). There exists a constant heat source Q=1 inside the cube. The thermal conductivity of the material is temperature dependent as follows:

$k = 1. + 0.1\,\phi + 0.5\,\phi^2$.

The task is to calculate the temperature distribution inside the cube. First the example with the mesh 10×10×10 is tested in order to assess convergence properties of the Newton-Raphson iterative procedure for large meshes. The procedure to generate heat-conduction element that is used in this example is explained in *AceGen* manual section Standard FE Procedure .



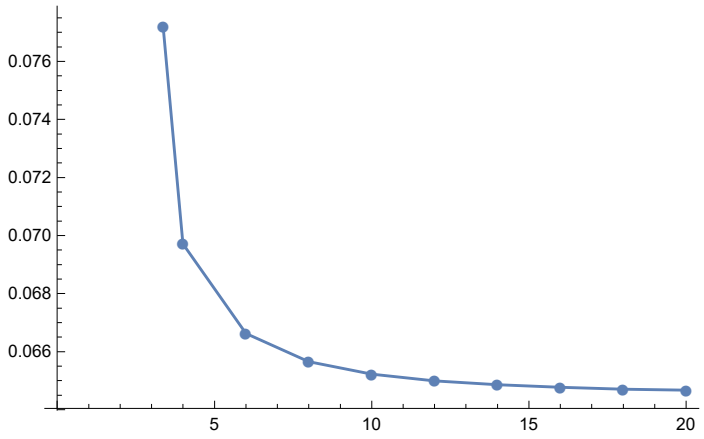
- In order to assess the convergence properties the problem is analyzed with the set of meshes from 2×2×2 to 20×20×20. The solution at the center of the cube (0.,0.,0.5) is observed.

In[1]:=
```
<< AceFEM`;
s = Table[
  SMTInputData[];
  SMTAddDomain["cube", "ExamplesHeatConduction",
   {"k0 *" -> 1., "k1 *" -> .1, "k2 *" -> .5, "Q *" -> 1.}];
  SMTAddEssentialBoundary[{#1 == -0.5 || #1 == 0.5 || #2 == -0.5 || #2 == 0.5 || #3 == 0. &, 1 -> 0}];
  SMTAddMesh[Hexahedron[{{-0.5, -0.5, 0}, {0.5, -0.5, 0}, {0.5, 0.5, 0}, {-0.5, 0.5, 0},
     {-0.5, -0.5, 1}, {0.5, -0.5, 1}, {0.5, 0.5, 1}, {-0.5, 0.5, 1}}], "cube", "H1", {i, i, i}];
  SMTAnalysis["Solver" → 5];
  SMTNextStep["λ" → 1];
  While[SMTConvergence[10^-12, 10], SMTNewtonIteration[];];
  {i, SMTPostData["Temp*", Point[{0, 0, 0.5}]]}
  , {i, 2, 20, 2}]
```

```
{{2, 0.0934011}, {4, 0.0697145}, {6, 0.0666232}, {8, 0.0656559}, {10, 0.0652253},
 {12, 0.0649954}, {14, 0.064858}, {16, 0.0647693}, {18, 0.0647088}, {20, 0.0646656}}
```

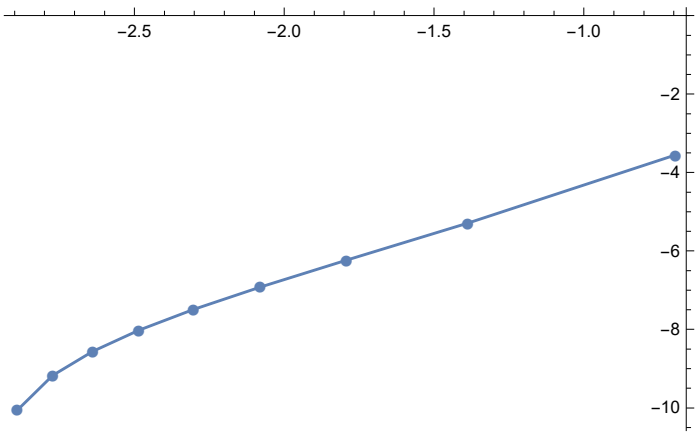
- The figure shows rapid convergence of the solution in the center of the cube with the mesh refinement.

In[3]:=
```
ListLinePlot[s, PlotMarkers -> Automatic]
```



- More about the convergence properties of the element can be observed form the *Log[characteristic mesh size]/Log[error]* plot. The solution obtained with the 20×20×20 mesh is assumed to be a converged solution. For a finite element method it is characteristic that the dependence between the characteristic mesh size and the error in a logarithmic scale is linear.

In[4]:=
```
ListLinePlot[Map[Log[{1/#[[1]], Abs[s[[-1, 2]] - #[[2]]]}] &, Drop[s, -1]],
 PlotMarkers -> Automatic]
```



Postprocessing (3D heat conduction)

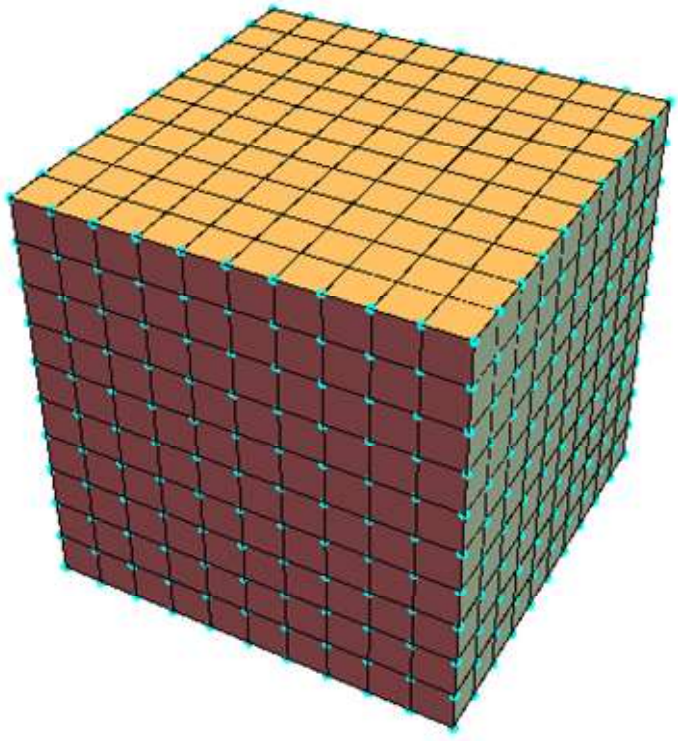
The domain of the problem is [-.0.5,0.5]×[-0.5,0.5]×[0,1] cube. On all sides, apart from the upper surface, the constant temperature ϕ=0 is prescribed. The upper surface is isolated so that there is no heat flow over the boundary ($\bar{q}$=0). There exists a constant heat source Q=1 inside the cube. The thermal conductivity of the material is temperature dependent as follows:

$k = 1. + 0.1\,\phi + 0.5\,\phi^2$.

The task is to calculate the temperature distribution inside the cube. First the example with the mesh 10×10×10 is tested in order to assess convergence properties of the Newton-Raphson iterative procedure for large meshes. The procedure to generate heat-conduction element that is used in this example is explained in *AceGen* manual section Standard FE Procedure

- The SMTAddMesh function generates nodes and elements for a cube discretized by the 10×10×10 mesh. The mesh and the boundary conditions are also depicted.

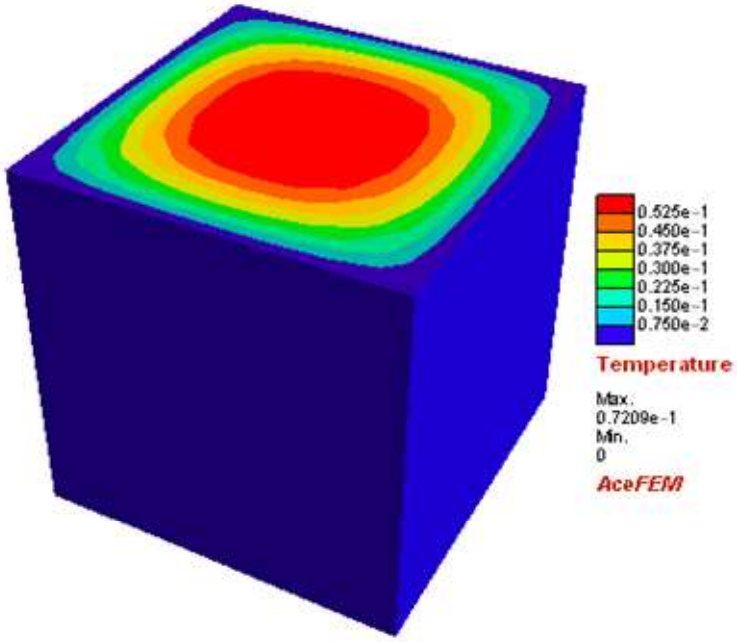
In[5]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["cube", "ExamplesHeatConduction",
  {"k0 *" -> 1., "k1 *" -> .1, "k2 *" -> .5, "Q *" -> 1.}];
SMTAddEssentialBoundary[{"X" == -0.5 || "X" == 0.5 || "Y" == -0.5 || "Y" == 0.5 || "Z" == 0. &, 1 -> 0}];
SMTAddMesh[Hexahedron[{{-0.5, -0.5, 0}, {0.5, -0.5, 0}, {0.5, 0.5, 0}, {-0.5, 0.5, 0},
    {-0.5, -0.5, 1}, {0.5, -0.5, 1}, {0.5, 0.5, 1}, {-0.5, 0.5, 1}}], "cube", "H1", {10, 10, 10}];
SMTAnalysis[];
SMTShowMesh["BoundaryConditions" → True, "Mesh" → True]
```



- Here the problem is analyzed and the contour plot of the temperature distribution is depicted.

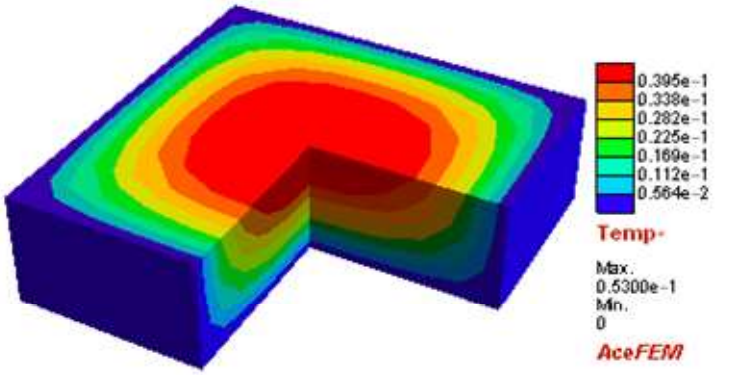
In[12]:=
```
SMTNextStep["λ" → 1];
While[SMTConvergence[10^-12, 10], SMTNewtonIteration[]; SMTStatusReport[];];
SMTShowMesh["Field" -> "Temperature", "Mesh" → False, "Contour" → True]
```

```
Step/Iter=1/1 λ/Δλ=1./1. ||Δp||/||R||=0.039126/0.000961769 Events=0 Status=0/{}
Step/Iter=1/2 λ/Δλ=1./1. ||Δp||/||R||=0.000117723/3.304×10^-6 Events=0 Status=0/{}
Step/Iter=1/3 λ/Δλ=1./1. ||Δp||/||R||=2.01035×10^-9/9.4527×10^-11 Events=0 Status=0/{}
Step/Iter=1/4 λ/Δλ=1./1. ||Δp||/||R||=2.82612×10^-18/6.53546×10^-19 Events=0 Status=0/{}
```

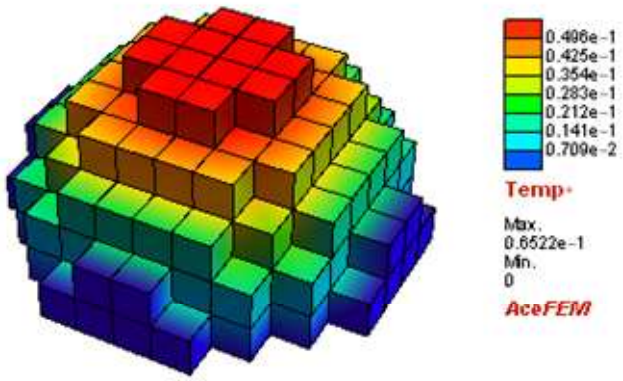


- Various views on the results can be obtained by the "Zoom" option.

In[15]:=
```
SMTShowMesh["Field" -> "Temp*", "Mesh" → False,
  "Contour" → True, "ZoomElements" → ("Z" <= 0.3 && ("X" <= 0 || "Y" ≥ 0) &) ]
```

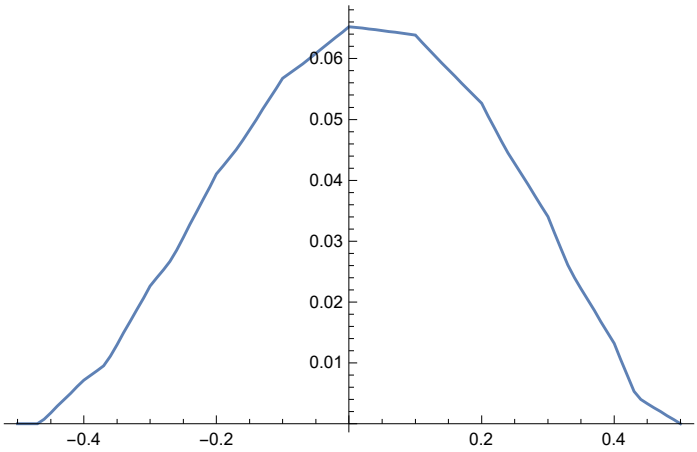


In[16]:=
```
SMTShowMesh["Field" -> "Temp*", "ZoomElements" → ("X"^2 + "Y"^2 + "Z"^2 ≤ .3 &) ]
```



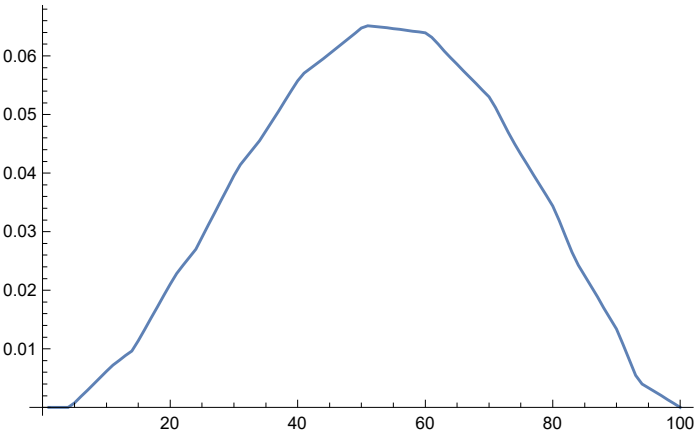
- Command depicts the temperature distribution along the central diagonal of the cube.

```
In[17]:= ListLinePlot[Table[{ξ, SMTPostData["Temp*", Point[{ξ, ξ, ξ + 0.5}]]}, {ξ, -0.5, 0.5, .01}]]
```



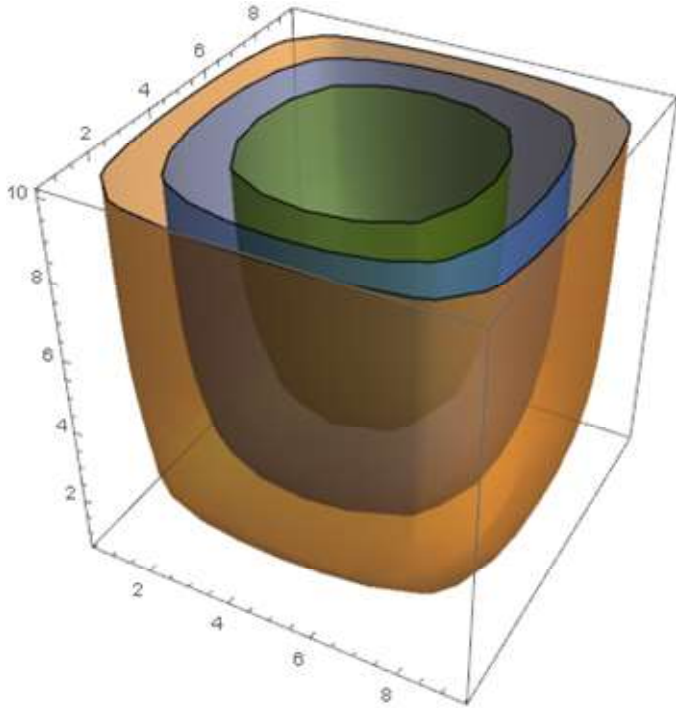
- The same graph can be produced by the following command.

```
In[18]:= ListLinePlot[SMTPostData["Temp*", Line[{{-0.5, -0.5, 0}, {0.5, 0.5, 1}}, 100]]]
```



- *AceFEM* is fully incorporated into *Mathematica* so that the various built-in *Mathematica*'s functions can be used for the analysis of the results. One should observe that *Mathematica*'s built-in functions operate mostly on regular domains, while *SMTShowMesh* displays results for an arbitrary mesh.
  Here the *ListContourPlot3D* is used to get surfaces with constant temperature.

In[19]:=
```
ListContourPlot3D[Transpose[Partition[Partition[SMTPostData["Temp*"], 11], 11], {3, 2, 1}],
 Contours → 3, ContourStyle → Opacity[0.6], Mesh -> False]
```

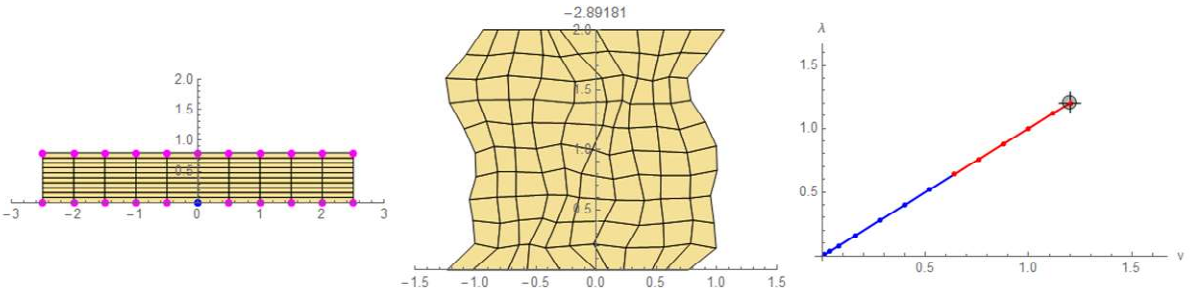


Houglassing test with animation

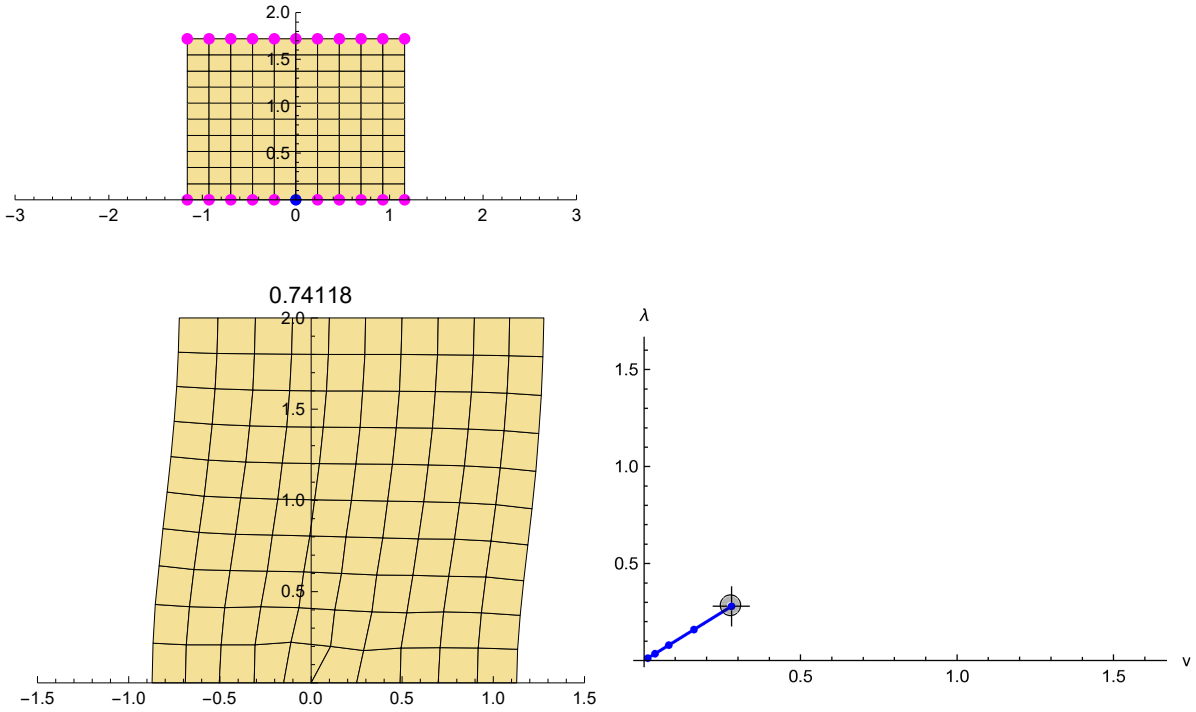
In[274]:=
```
<< AceFEM`;
SMTInputData[];
{element, topology} = Switch["Q1E4"
   , "Q1", {{"ML:", "SE", "PE", "Q1", "DF", "HY", "Q1", "D", {{"NeoHooke", "WA"}}}, "Q1"}
   , "Q1E4", {{"ML:", "SE", "PE", "Q1", "ES", "HY", "Q1E4", "D", {{"NeoHooke", "WA"}}}, "Q1"}
   , "CG4", {{"ML:", "SE", "PE", "Q1", "ES", "HY", "CG4", "D", {{"NeoHooke", "WA"}}}, "Q1"}
   , "Q2", {{"ML:", "SE", "PE", "Q2", "DF", "HY", "Q2", "D", {{"NeoHooke", "WA"}}}, "Q2"}
  ];
SMTAddDomain["A", element, {"E *" -> 250, "ν *" → 0.4999}];
ne = 10; ΔH = -1; a = 1;
SMTAddMesh[Polygon[{{-a, 0}, {a, 0}, {a, 2 * a}, {-a, 2 * a}}], "A", "Division" -> {ne, ne}];
SMTAddEssentialBoundary[{ Point[{0, 0}], 1 -> 0, 2 → 0},
  { Line[{{-a, 0}, {a, 0}}], 2 -> 0}, { Line[{{-a, 2 a}, {a, 2 a}}], 2 -> ΔH }];
SMTAnalysis[];
```

In[9]:=
```
λMax = 1.2; λ0 = λMax / 100; ΔλMin = λMax / 1000; ΔλMax = λMax / 10;
tolNR = 10^-8; maxNR = 15; targetNR = 8;
SMTNextStep["λ" → λ0];
While[
  While[
   step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λMax}]
   , SMTNewtonIteration[];
  ];
  If[Not[step[[1]]]
   , SMTAnimationOfResponse[
    "LeadingNodePosition" → {0, 2 a},
    "PlotOptions" →
     {AxesLabel → {"v", "λ"}, PlotRange → {{0, -ΔH 1.5}, {0, 1.5}}, ImageSize → 300},
    "ShowMeshOptions" → {PlotRange → {{-3 a, 3 a }, {0, 2 a}}, ImageSize → 300},
    "Include" → Row[{"ShowMesh"
       , SMTShowEigenvectors[SMTData["TangentMatrix"], 1, "Scale" → 3,
         PlotRange → {{-1.5 a, 1.5 a}, {0, 2 a}}, Axes → True, ImageSize → 300][[1]]
       , "Plot"}, "    "]
    , "Show" → "Window" | {"ExportFrames", "tmp"}
   ]
  ];
  If[step[[4]] === "MinBound", SMTStatusReport["ΔT<ΔT_min"]; SMTStepBack[];];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" → step[[2]]]
 ];
```

In[17]:=
```
SMTAnimationOfResponse["Last frame"]
```

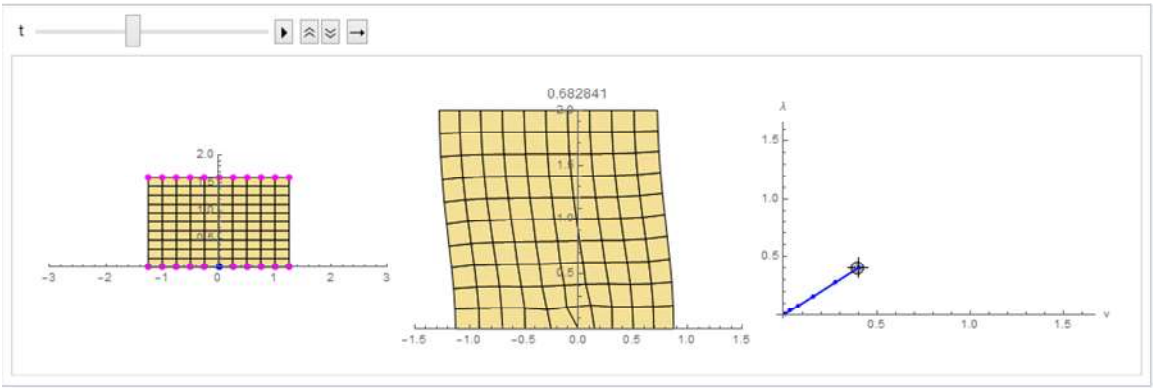


In[36]:= **SMTAnimationOfResponse["Frame", "FrameSelection" → 5]**



- This creates a cell with controls added to allow interactive manipulation of animation within the notebook.

In[15]:= **SMTAnimationOfResponse["Animate", DefaultDuration → 10]**



Round-off Error Test

With the AceFEM-MDriver module the advantages of the *Mathematica*'s high precision arithmetic, interval arithmetic, or even symbolic evaluation can be used. In this section the number of significant digits lost due to the round-off error is calculated.

In the previous section the *C* language code for the steady state heat conduction problem was generated. Due to the arbitrary precision arithmetic required, the C code can not be used any more and *Mathematica* code has to be generated.

- Here the *Mathematica* code for the steady state heat conduction problem (see Standard FE Procedure) for AceFEM-MDriver is generated.

In[43]:=
```
<< AceGen`;
SMSInitialize["PrecisionHeatConduction", "Environment" -> "AceFEM-MDriver"];
SMSTemplate["SMSTopology" → "H1", "SMSDOFGlobal" → 1, "SMSSymmetricTangent" → False,
  "SMSDomainDataNames" -> {"k0 -conductivity parameter", "k1 -conductivity parameter",
    "k2 -conductivity parameter", "Q -heat source"},
  "SMSDefaultData" -> {1, 0, 0, 0}];
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
XIO ⊨ SMSIO["Nodal coordinates"];
Ξn = {{-1, -1, -1}, {1, -1, -1}, {1, 1, -1}, {-1, 1, -1},
   {-1, -1, 1}, {1, -1, 1}, {1, 1, 1}, {-1, 1, 1}};
Nh ⊨ Table[1 / 8 (1 + ξ Ξn[[i, 1]]) (1 + η Ξn[[i, 2]]) (1 + ζ Ξn[[i, 3]]), {i, 1, 8}];
SMSFreeze[X, Nh.XIO]; Je ⊨ SMSD[X, Ξ]; Jed ⊨ Det[Je];
ϕI ⊨ Flatten[SMSIO["Nodal DOFs"]];
ϕ ⊨ Nh.ϕI;
{k0, k1, k2, Q} ⊨ SMSIO["Domain data"];
k ⊢ k0 + k1 ϕ + k2 ϕ^2;
λ ⊨ SMSIO["Multiplier"];
wgp ⊨ SMSIO["Integration weight"[Ig]];
Dϕ ⊨ SMSD[ϕ, X, "Dependency" -> {Ξ, X, SMSInverse[Je]}];
W ⊨ 1/2 k Dϕ.Dϕ - ϕ λ Q;
SMSDo[
  Rg ⊨ Jed SMSD[W, ϕI, i, "Constant" → k];
  SMSIO[wgp Rg, "Add to", "Residual"[i]];
  SMSDo[
   Kg ⊨ SMSD[Rg, ϕI, j];
   SMSIO[wgp Kg, "Add to", "Tangent"[i, j]];
   , {j, 1, 8}
  ];
  , {i, 1, 8}
 ];
SMSEndDo[];
SMSWrite[];
```

```
File: PrecisionHeatConduction.m  Size: 11142  Time: 3
```

Method	SMT`SKR
No.Formulae	139
No.Leafs	2895

The domain of the problem is sphere with radius 1. On the lower surface of the sphere the constant temperature of ϕ=0 is prescribed. The upper surface is isolated, so that there is no heat flow over the boundary ($\bar{q}$=0). There exists a constant heat source Q=1 inside the sphere. The task is to evaluate the number of significant digits lost when the sphere is discretized with the 125 hexahedra thermal conductivity elements.

- This starts symbolic *MDriver* with all the numerical constants set to have 40 digit precision, thus during the analysis the 40 digit precision real numbers are used.

In[63]:=
```
<< AceFEM`;
SMTInputData["NumericalModule" → "MDriver", "Precision" → 40]
SMTAddDomain["sphere", "PrecisionHeatConduction",
  {"k0 *" -> 100, "k1 *" -> 10, "k2 *" -> 5, "Q *" -> 1}];
SMTAddEssentialBoundary[ "Z" <= 0.01 && "X"^2 + "Y"^2 + "Z"^2 > 0.98 &, 1 -> 0];
```

```
True
```

- The SMTAddMesh function uses general interpolation of coordinates with the arbitrary number of interpolation points. Here the mesh of 11×11×11 interpolation points is obtained by mapping the cube into the sphere. The volume is then divided into 5×5×5 elements.

In[67]:=
```
points = N[Table[ Max[Abs[{ξ, η, ζ}]] {ξ, η, ζ} / Sqrt[{ξ, η, ζ}.{ξ, η, ζ}],
    {ξ, -1, 1, 2/11}, {η, -1, 1, 2/11}, {ζ, -1, 1, 2/11}], 40];
SMTAddMesh[Raster3D[points], "sphere", "H1", {5, 5, 5}];
```

In[69]:=
```
SMTAnalysis[];
SMTNextStep["λ" -> 1];
```

In[71]:=
```
$MinPrecision = 0; $MaxPrecision = Infinity;
```

- This gives the temperature in the node 100 and the numerical precision of the result for 5 Newton-Raphson iterations.

In[72]:=
```
res = Table[
   SMTNewtonIteration[];
   {i, SMTNodeData[100, "at"][[1]], SMTNodeData[100, "at"][[1]] // Precision}
   , {i, 5}];
```

In[73]:=
```
TableForm[res, TableHeadings → {None, {"Iteration", "Temperature", "Precision"}}]
```

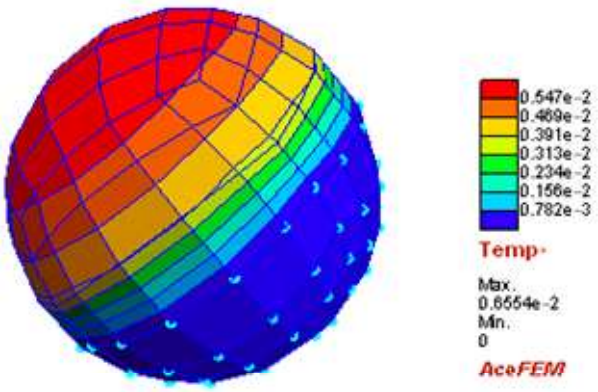
Iteration	Temperature	Precision
1	0.00431367779157925174656753608684058 60	35.6105
2	0.00431268955724400112376415971040235	33.0778
3	0.00431268955719375064016726409861	30.485
4	0.0043126895571937506401672639 69	28.2958
5	0.0043126895571937506401672639 69	28.2958

It can be seen that 7 significant digits have been lost during the computation. Although this seems a lot, one should be aware that when a high precision arithmetic is used, *Mathematica* takes "save bound" for the precision of the intermediate results and that the actual number of significant digits lost is usually much lower. This can be verified by running the same example with the AceFEM-CDriver, where double precision real numbers are used.

In[74]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["sphere", "ExamplesHeatConduction",
  {"k0 *" -> 100, "k1 *" -> 10, "k2 *" -> 5, "Q *" -> 1}];
SMTAddEssentialBoundary[ "Z" <= 0.01 && "X"^2 + "Y"^2 + "Z"^2 > 0.98 &, 1 -> 0];
SMTAddMesh[Raster3D[Table[ Max[Abs[{ξ, η, ζ}]] {ξ, η, ζ} / Sqrt[{ξ, η, ζ}.{ξ, η, ζ}],
    {ξ, -1., 1., 2./11}, {η, -1., 1., 2./11}, {ζ, -1., 1., 2./11}]], "sphere", "H1", {5, 5, 5}];
SMTAnalysis[]; SMTNextStep["λ" -> 1];
Table[SMTNewtonIteration[], {5}]
```

$$\{0.00468568,\ 1.27035\times 10^{-6},\ 1.0465\times 10^{-13},\ 2.75474\times 10^{-19},\ 2.54215\times 10^{-19}\}$$

In[81]:=
```
SMTShowMesh["Field" -> "Temp*", "BoundaryConditions" → True, "Contour" -> True]
```



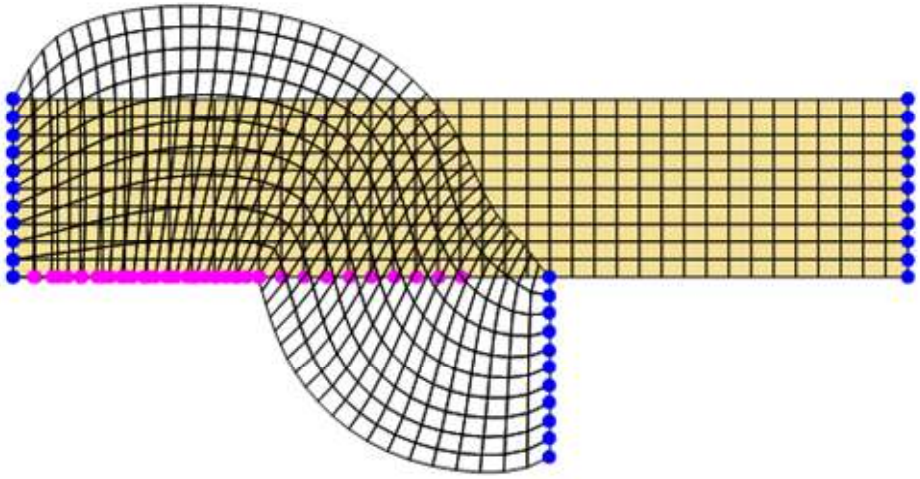
- The numerical precision of the machine precision numbers is always 16. The number of significant digits lost can be obtained by comparing the results with the high precision results.

In[82]:=
```
SMTNodeData[100, "at"] // InputForm
```

```
{0.004312689557193752}
```

Simulation of residual stresses and spring-back effects

Elasto-plastic block of steel is pressed back and pushed down as depicted. The tool is after the action removed. Due to the high plastic and elastic deformations we get significant residual stresses as well as spring-back effect.

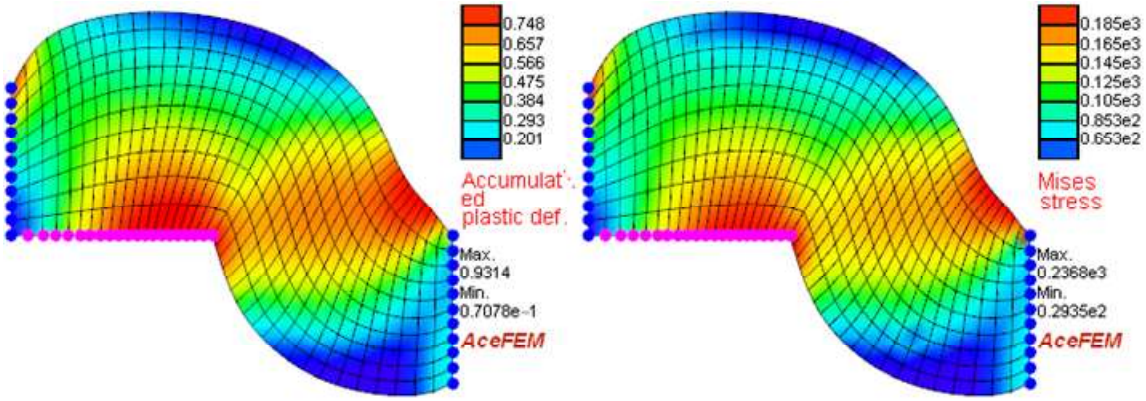


Simulations of forming process

In[282]:=
```
<< AceFEM`;
SMTInputData[];
L = 100; H = 20;
nx = 40; ny = 10;
SMTAddDomain["A",
  {"ML:", "SE", "PE", "Q1", "DF", "JC", "Q1", "D", {{"NeoHooke", "WA"}, {"Mises", "ExH"}}},
  {"E *" → 21000, "ν *" → 0.2, "σy *" → 24, "Kh *" → 200}];
SMTAddMesh[Polygon[{{0, 0}, {L, 0}, {L, H}, {0, H}}], "A", "Division" -> {40, 10}];
SMTAddEssentialBoundary[Line[{{0, 0}, {0, H}}], 1 -> 0, 2 → 0];
SMTAddEssentialBoundary[Line[{{0, 0}, {L / 2, 0}}], 2 -> 0];
SMTAddEssentialBoundary[Line[{{L, 0}, {L, H}}], 2 → -H, 1 → -2 H];
SMTAnalysis[];
```

In[93]:=
```
SMTNextStep["λ" → 0.1];
While[
 While[step = SMTConvergence[10^-7, 15, {"Adaptive BC", 8, .01, 0.2, 1}] ,
  SMTNewtonIteration[]];
 If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
 If[Not[step[[1]]],
  SMTShowMesh["DeformedMesh" → True, "Field" → "Mises*", "Mesh" → False, "Show" -> "Window"]];
 step[[3]] ,
 If[step[[1]], SMTStepBack[]];
 SMTNextStep["Δλ" → step[[2]]]
]
```

In[95]:=
```
defmesh = SMTShowMesh["DeformedMesh" → True, "FillElements" → False];
Row[{SMTShowMesh["DeformedMesh" → True,
    "Field" → "Accumulated plastic def.", "BoundaryConditions" → True, ImageSize → 300],
  SMTShowMesh["DeformedMesh" → True, "Field" → "Mises stress",
   "BoundaryConditions" → True, ImageSize → 300]}]
```



Removing the prescribed boundary conditions

- Check that the body is at the end of simulation in perfect equilibrium.

In[96]:=
```
SMTNextStep["Δλ" → 0];
SMTNewtonIteration[]
```

$$4.64388 \times 10^{-14}$$

- Prescribed displacements at the end are removed.

In[98]:=
```
SMTAddEssentialBoundary[Line[{{L, 0}, {L, H}}], 1 → Null, 2 → Null];
```

- After the removal of the prescribed displacement the body is no longer in equilibrium. For the successful continuation of the simulation, the equilibrium have to be established again. Thus, the prescribed displacement are her replaced by the forces in the same nodes.
- Here the equilibrating forces are calculated.

In[99]:=
```
force = SMTResidual[Line[{{L, 0}, {L, H}}]]
```

```
{{84.2582, -0.039973}, {59.1034, 125.279}, {6.44076, 154.475}, {-51.6629, 167.814},
 {-114.07, 174.577}, {-168.379, 173.472}, {-199.361, 166.744}, {-209.909, 205.443},
 {-109.588, 209.756}, {646.522, 1261.19}, {-4032.11, -1646.59}}
```

- Here the current value of nodal forces (**Bt**) is set to equilibrating forces.

In[100]:=
```
SMTAddInitialBoundary[Line[{{L, 0}, {L, H}}],
  1 → force[[All, 1]], 2 → force[[All, 2]], "Set" → True];
```

- The structure is again in perfect equilibrium.

In[101]:=
```
SMTNewtonIteration[]
```

$$1.15697 \times 10^{-12}$$

- Here the increment of nodal forces (**dB**) is set to negative equilibrating forces in order to counterbalance the equilibrating forces at the end of simulation.

In[102]:=
```
SMTAddNaturalBoundary[Line[{{L, 0}, {L, H}}],
  1 → -force[[All, 1]], 2 → -force[[All, 2]], "Set" → True];
```

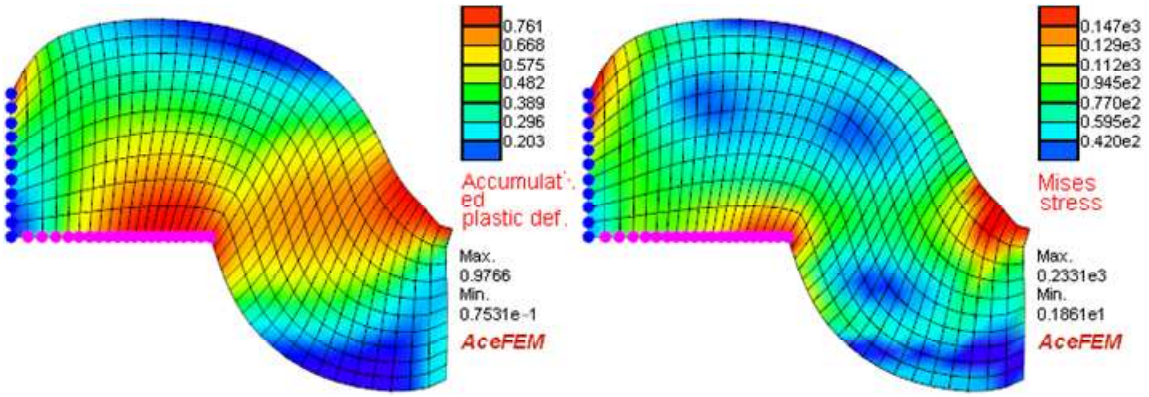
- At the end of path following procedure the forces in nodes at the end are brought to zero.

In[103]:=
```
SMTRData["Multiplier", 0];
SMTNextStep["Δλ" → 0.01];
While[
 While[
  step = SMTConvergence[10^-7, 15, {"Adaptive BC", 8, .0001, 0.2, 1}, "Alternate" → "Ignore"] ,
  SMTNewtonIteration[]];
 SMTStatusReport[];
 If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
 If[Not[step[[1]]], SMTShowMesh["DeformedMesh" → True,
   "Field" → "Sxy", "Mesh" → False, "Contour" → 20, "Show" -> "Window"]];
 step[[3]] ,
 If[step[[1]], SMTStepBack[]];
 SMTNextStep["Δλ" → step[[2]]]
]
```

```
Step/Iter=14/5 λ/Δλ=0.01/0.01 ||Δp||/||R||=1.8419×10^-9/7.86191×10^-11 Events=78 Status=0/{}

Step/Iter=15/4 λ/Δλ=0.0281633/0.0181633
  ||Δp||/||R||=5.12843×10^-10/9.88013×10^-11 Events=2 Status=0/{}

Step/Iter=16/5 λ/Δλ=0.0630071/0.0348438
  ||Δp||/||R||=1.1009×10^-11/8.73348×10^-11 Events=3 Status=0/{}

Step/Iter=17/7 λ/Δλ=0.126295/0.0632877
  ||Δp||/||R||=2.46144×10^-10/8.56714×10^-11 Events=24 Status=0/{}

Step/Iter=18/5 λ/Δλ=0.220581/0.0942858
  ||Δp||/||R||=5.90578×10^-8/1.14567×10^-9 Events=4 Status=0/{}

Step/Iter=19/6 λ/Δλ=0.391834/0.171254
  ||Δp||/||R||=2.65121×10^-11/8.82218×10^-11 Events=35 Status=0/{}

Step/Iter=20/6 λ/Δλ=0.591834/0.2 ||Δp||/||R||=
 5.12039×10^-8/3.73423×10^-9 Events=44 Status=0/{}

Step/Iter=21/7 λ/Δλ=0.791834/0.2 ||Δp||/||R||=
 2.35996×10^-11/7.89108×10^-11 Events=42 Status=0/{}

Step/Iter=22/7 λ/Δλ=0.991834/0.2 ||Δp||/||R||=
 2.3419×10^-10/8.83089×10^-11 Events=85 Status=0/{}

Step/Iter=23/5 λ/Δλ=1./0.0081655 ||Δp||/||R||=
 6.00339×10^-11/8.98739×10^-11 Events=6 Status=0/{}
```

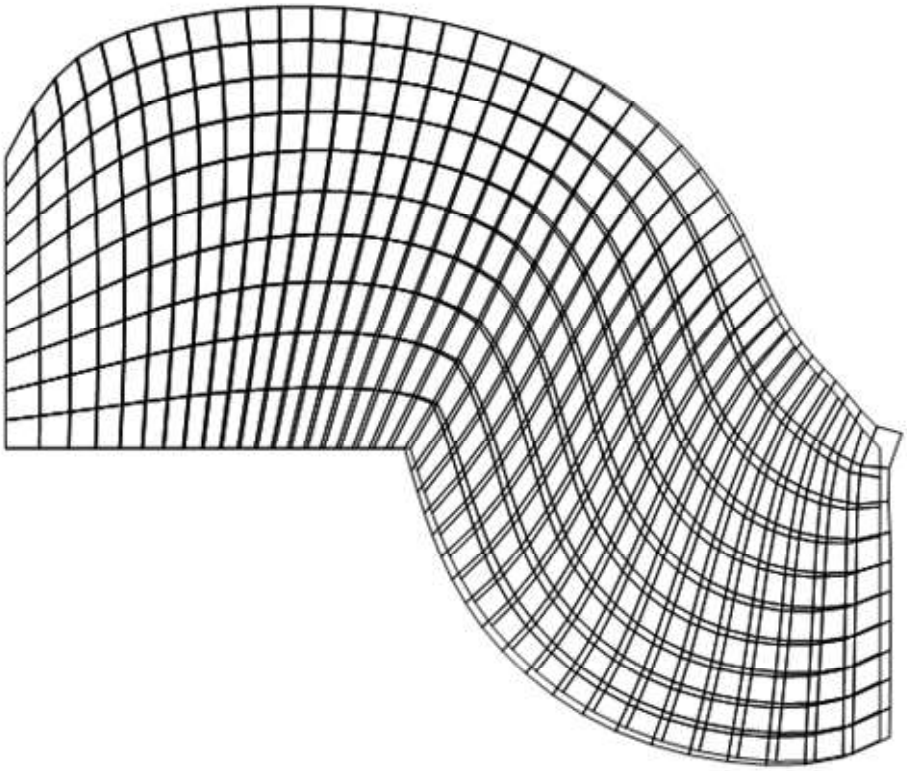
- We can see that accumulated plastic deformation remains the same, while the stress patterns is completely changed. The residual Mises stress is shown here.

In[106]:=
```
Row[{SMTShowMesh["DeformedMesh" → True,
   "Field" → "Accumulated plastic def.", "BoundaryConditions" → True, ImageSize → 300],
  SMTShowMesh["DeformedMesh" → True, "Field" → "Mises stress",
   "BoundaryConditions" → True, ImageSize → 300]}]
```



- The amount of spring-back is shown here.

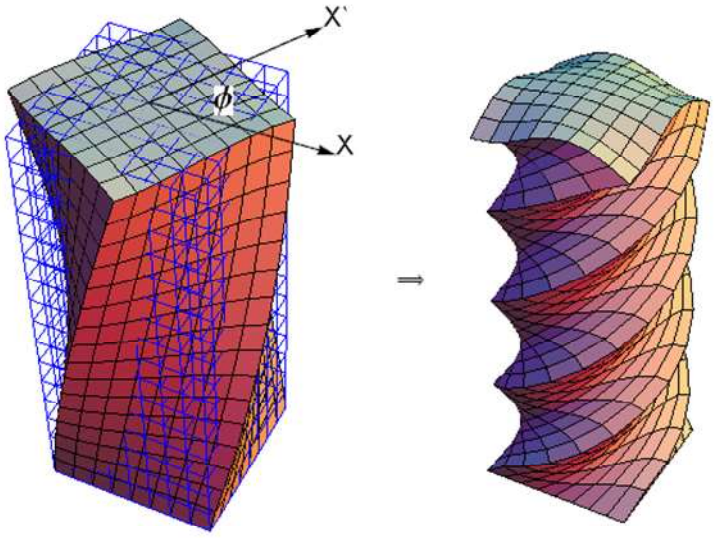
In[107]:=
```
Show[defmesh, SMTShowMesh["DeformedMesh" → True, "FillElements" → False]]
```



Advanced control of boundary conditions

Torsion test

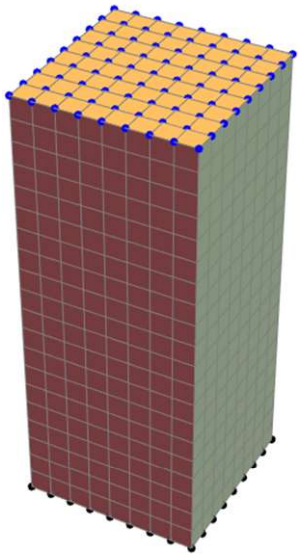
The domain of the problem is [-.a,a]×[-a,a]×[0,L] cube. The bottom surface is fully constrained. The upper surface is rotated around the center point for angle ϕ. Deplaning of the upper surface in is not restricted. The task is to calculate curve *torsion_moment*(ϕ) for $\phi \in [0, \phi_M]$. Since the rotation of the upper surface cannot be described simply by linearly increasing essential boundary conditions, the prescribed displacement of each node on upper surface has to be prescribed explicitly as a function of the angle of rotation ϕ in a way that follows the circular path. Let $\mathbf{B}_t$ be the current value of prescribed displacements of the nodes forming the upper surface, $\Delta\mathbf{B}$ a vector associated with the current pattern of the prescribed displacements and $\mathbf{B}_p$ the value of prescribed displacements at the end of previous step. In *AceFEM* the current value of boundary conditions (BC) is calculated as $\mathbf{B}_t = \mathbf{B}_p + \Delta\lambda\,\Delta\mathbf{B}$, where $\Delta\lambda$ is an increment of BC multiplier. Additionally, the path following procedure can be controlled (parameterized) either by time t or BC multiplier λ. Both, time and BC multiplier can be used instead of angle ϕ leading to two different solutions.



Solution 1: Path parameterized by BC multiplier

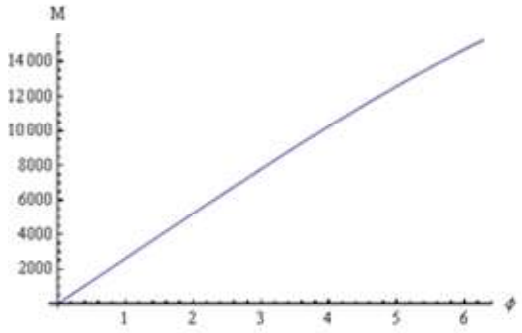
First, the boundary condition (BC) multiplier λ is used as a parameter (angle) to parameterized the path of each node, thus $\phi \equiv \lambda$ and $\Delta\phi = \Delta\lambda$. All the upper surface nodes are constrained and set to 0 at the input data phase. With the known relation $\mathbf{B}_t(\phi)$ the pattern $\Delta\mathbf{B}$ can be calculated as $\Delta\mathbf{B} = \frac{\mathbf{B}_t(\phi)-\mathbf{B}_p}{\Delta\lambda}$ and set to all nodes at the upper surface at given λ.

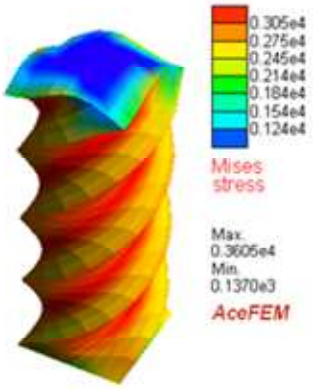
In[292]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[{"A", {"ML:", "SE", "D3", "H1", "DF", "HY", "H1", "D", {{"NeoHooke", "WA"}}},
   {"E *" → 2500, "ν *" -> 0.35}}];
L = 5; a = 1; ϕM = 2 Pi // N;
SMTAddMesh[Hexahedron[{{-a, -a, 0}, {a, -a, 0}, {a, a, 0}, {-a, a, 0},
    {-a, -a, L}, {a, -a, L}, {a, a, L}, {-a, a, L}}], "A", "Division" -> {8, 8, 20}];
SMTAddEssentialBoundary[Polygon[{{-a, -a, 0}, {a, -a, 0}, {a, a, 0}, {-a, a, 0}}],
  1 -> 0, 2 → 0, 3 → 0];
SMTAddEssentialBoundary[Polygon[{{-a, -a, L}, {a, -a, L}, {a, a, L}, {-a, a, L}}], 1 -> 0, 2 → 0];
SMTAnalysis[];
SMTShowMesh["BoundaryConditions" → True]
```



In[117]:=
```
SMTNextStep["λ" → Pi / 100.];
Mϕ = {{0, 0}};
UpperSurface = SMTFindNodes[ "Z" == L &];
Bp = SMTNodeData[UpperSurface, "Bp"];
While[
 ϕ = SMTRData["Multiplier"]; Δϕ = SMTRData["MultiplierIncrement"];
 Bt = Map[{- #[[1]] Cos[(Pi - ϕ) / 2] - #[[2]] Sin[(Pi - ϕ) / 2], #[[1]] Sin[(Pi - ϕ) / 2] -
       #[[2]] Cos[(Pi - ϕ) / 2], 0} 2 Sin[ϕ / 2] &, SMTNodeData[UpperSurface, "X"]];
 SMTNodeData[UpperSurface, "dB", (Bt - Bp) / Δϕ];
 While[step = SMTConvergence[10^-8, 16, {"Adaptive BC", 12, 0.0001, ϕM / 100, ϕM}],
  SMTNewtonIteration[];];
 If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
 If[Not[step[[1]]],
  Bp = Bt;
  M = Total[MapThread[Norm[Cross[#1, #2]] &, {SMTNodeData[UpperSurface, "X"] +
        SMTNodeData[UpperSurface, "at"], SMTResidual[UpperSurface]}]];
  AppendTo[Mϕ, {ϕ, M}];
  SMTShowMesh["DeformedMesh" → True, "Show" -> "Window"];
 ];
 step[[3]]
 , If[step[[1]], SMTStepBack[];];
 SMTNextStep["Δλ" → step[[2]]];
]
```

In[122]:=
```
SMTShowMesh["DeformedMesh" -> True, "Field" -> "Mises stress", "Mesh" -> False,
 "Combine" -> (GraphicsRow[{ListLinePlot[Mϕ, AxesLabel -> {"ϕ", "M"}], #}, ImageSize -> 600] &)]
```





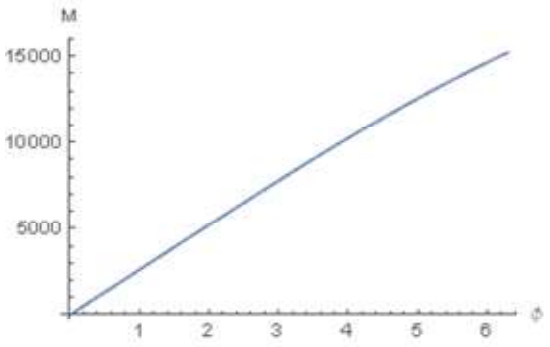
Solution 2: Path parameterized by general parameter

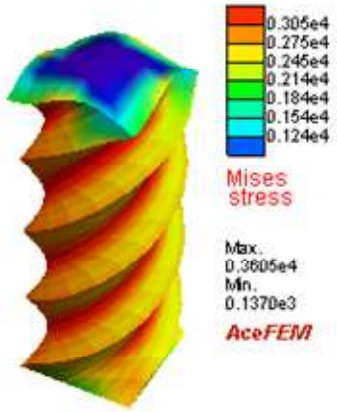
Here, general parameter γ is used as a parameter (angle) to parameterized the path of each node, thus $\phi \equiv \gamma$ and $\Delta\phi = \Delta\gamma$. At the input data phase all the upper surface nodes are constrained and set to 0. With the known relation $\mathbf{B}_t(\phi)$ and the multiplier increment $\Delta\lambda$ always set to 1 by SMTNextStep["Δλ"->1], the pattern $\Delta\mathbf{B}$ can be calculated as $\Delta\mathbf{B} = \mathbf{B}_t(\phi) - \mathbf{B}_p$ and set to all nodes at the upper surface at given time.

In[301]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[{"A", {"ML:", "SE", "D3", "H1", "DF", "HY", "H1", "D", {{"NeoHooke", "WA"}}},
    {"E *" → 2500, "ν *" -> 0.35}}];
L = 5; a = 1; ϕM = 2 Pi // N;
SMTAddMesh[Hexahedron[{{-a, -a, 0}, {a, -a, 0}, {a, a, 0}, {-a, a, 0},
     {-a, -a, L}, {a, -a, L}, {a, a, L}, {-a, a, L}}], "A", "Division" -> {8, 8, 20}];
SMTAddEssentialBoundary[Polygon[{{-a, -a, 0}, {a, -a, 0}, {a, a, 0}, {-a, a, 0}}],
  1 -> 0, 2 → 0, 3 → 0];
SMTAddEssentialBoundary[Polygon[{{-a, -a, L}, {a, -a, L}, {a, a, L}, {-a, a, L}}], 1 -> 0, 2 → 0];
SMTAnalysis[];
```

In[131]:=
```
SMTNextStep["γ" -> 0.1, "Δλ" -> 1];
Mϕ = {{0, 0}};
UpperSurface = SMTFindNodes[ "Z" == L &];
Bp = SMTNodeData[UpperSurface, "Bp"];
While[
 ϕ = SMTRData["Parameter"];
 Bt = Map[{- #[[1]] Cos[(Pi - ϕ) / 2] - #[[2]] Sin[(Pi - ϕ) / 2], #[[1]] Sin[(Pi - ϕ) / 2] -
         #[[2]] Cos[(Pi - ϕ) / 2], 0} 2 Sin[ϕ / 2] &, SMTNodeData[UpperSurface, "X"]] // N;
 SMTNodeData[UpperSurface, "dB", Bt - Bp];
 While[step = SMTConvergence[10^-8, 16, {"Adaptive γ", 12, 0.0001, ϕM / 100, ϕM}],
  SMTNewtonIteration[];];
 If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
 If[Not[step[[1]]],
  Bp = Bt;
  M = Total[MapThread[Norm[Cross[#1, #2]] &, {SMTNodeData[UpperSurface, "X"] +
        SMTNodeData[UpperSurface, "at"], SMTResidual[UpperSurface]}]];
  AppendTo[Mϕ, {ϕ, M}];
  SMTShowMesh["DeformedMesh" → True, "Show" -> "Window"];
 ];
 step[[3]]
 , If[step[[1]], SMTStepBack[];];
 SMTNextStep["Δγ" -> step[[2]], "Δλ" -> 1];
]
```

In[136]:=
```
SMTShowMesh["DeformedMesh" -> True, "Field" -> "Mises stress", "Mesh" -> False,
 "Combine" -> (GraphicsRow[{ListLinePlot[Mϕ, AxesLabel -> {"ϕ", "M"}], #}, ImageSize -> 600] &)]
```





Adaptive mesh refinement

Adaptive finite element refinement algorithm

Adaptive finite element refinement algorithm presented here makes a global rifinement of the whole mesh and is based on a ***posteriori recovery based error estimator*** as decribed in Zienkiewicz & Taylor, The finite element method, Vol 1, 2000.

Steps of the finite element refinement algorithm:

1. simulation before remeshing using coarse mesh
2. adaptive finite element refinement
3. data transfer from old to new mesh
4. established new equilibrium after remeshing
5. simulation after remeshing using fine mesh

Algorithm has the following limitations:

- the procedure assumes the all the nodes have the same number of unknowns and that all unknowns approximate the same vector field
- problem has to be time independent and without history variables

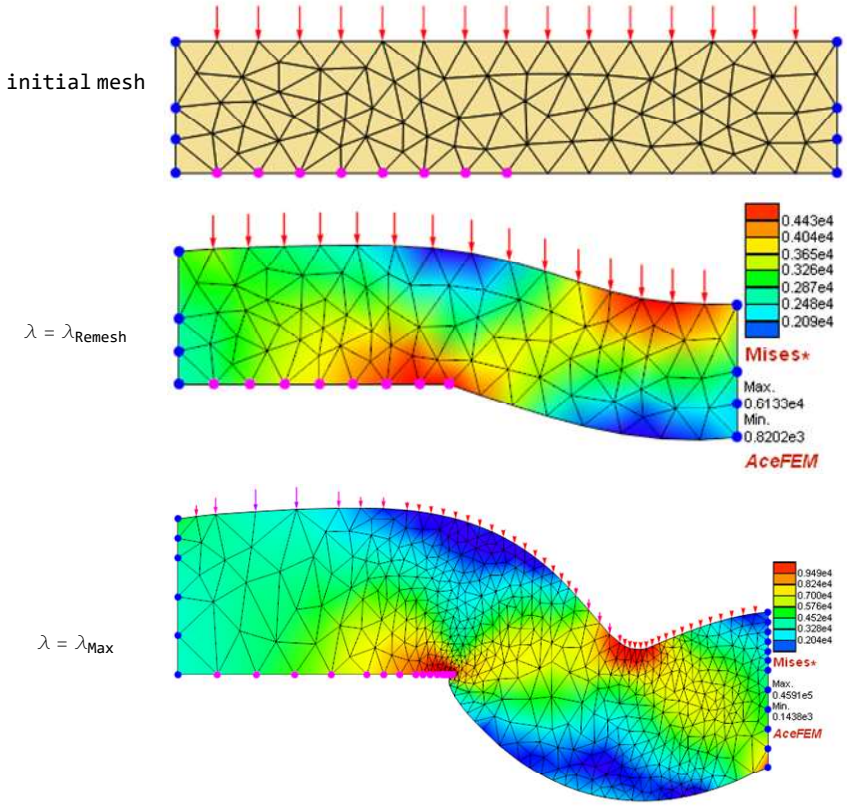
Key AceFEM functionality used is SMTPostData function.

SMTPostData[*field_code*, Point[{T_1,T_2,...}]]
evaluate scalar *field/fields* at spatial points {T_1,T_2,...}

field_code	description
pcode_String	field defined by post−processing keyword *pcode* (definition of post−processing keywords is a part of element code, see Subroutine: Postprocessing)
{*ncode_String*, *dof_Integer*}	field is the *dof*−th component of the primal analysis related nodal quantity *ncode* ("at", "ap" or "da") (see also Node Data)

Simple example

- hyper-elastic material
- constant triangle elements
- rectangular domain L×H
- boundary conditions
  - x=0: **u**=(0,0) , Y=0 ∧ X<L/2: v=0
  - prescribed proportional displacement boundary conditions at the and of the beam x=L: **u** = λ (up, vp)
  - constant distributed load at the top of the beam y=H: **t** = (0, q)
- path-following procedures with adaptive BC multiplier $\lambda \in [0, \lambda_{Max}]$ is used to solve nonlinear problem
- remeshing is done at the predefined load level $\lambda = \lambda_{Remesh}$



Step 1. Run simulation with coarse mesh until $\lambda = \lambda_{Remesh}$

In[137]:=
```
<< AceFEM`;
```

- Geometry, load and mesh parameters.

In[138]:=
```
L = 100; H = 20;
qConstant = -100; up = -H / 2; vp = -H;
λMax = 1.2;
fullArea = L H;
(* maximal element area*)
maxArea = fullArea / 100;
```

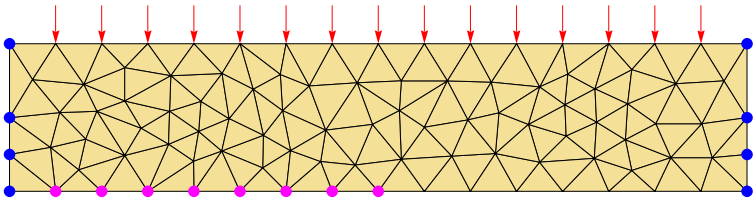
- Generation of original unstructured mesh.

In[143]:=
```
meshCoarse = ToElementMesh[
    Polygon[{{0, 0}, {L / 2, 0}, {L, 0}, {L, H}, {0, H}}]
    , MaxCellMeasure → maxArea
    , "MeshOrder" → 1
   ];
```

In[144]:=
```
startAceFEM[mesh_] := (
  SMTInputData[];
  SMTAddDomain["A", {"ML:", "SE", "PE", "T1", "DF", "HY", "T1", "D", {{"NeoHooke", "WA"}}},
   {"E *" → 21000, "ν *" → 0.2}];
  SMTAddMesh[mesh, "A"];
  SMTAddInitialBoundary[Point[{0, H}] || Point[{L, H}], 2 → 0, "Type" → "EssentialBoundary"];
  SMTAddInitialBoundary[Line[{{0, H}, {L, H}}], 2 → Line[{qConstant}]];
  SMTAddEssentialBoundary[Line[{{0, 0}, {0, H}}], 1 -> 0, 2 → 0];
  SMTAddEssentialBoundary[Line[{{0, 0}, {L / 2, 0}}], 2 -> 0];
  SMTAddEssentialBoundary[Line[{{L, 0}, {L, H}}], 2 → up, 1 → vp];
  SMTAnalysis[];
 )
```

In[145]:=
```
startAceFEM[meshCoarse]
```

In[146]:=
```
SMTShowMesh["BoundaryConditions" → True]
```



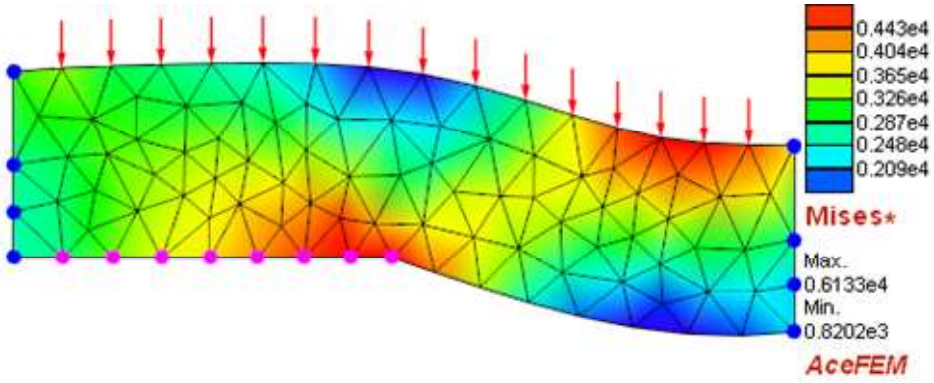
- run the path-following procedures with adaptive BC multiplier with $\lambda \in [0, \lambda_{\text{Remesh}}]$

In[147]:=
```
λRemesh = 0.8;
λ0 = λMax / 100.; ΔλMin = λMax / 1000.;
ΔλMax = λMax / 10.; tolNR = 10. ^ -8;
maxNR = 15;
targetNR = 8;
```

In[151]:=
```
SMTNextStep["λ" → λ0];
While[
  While[step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λRemesh}]
   , SMTNewtonIteration[]
  ];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  If[Not[step[[1]]],
   SMTShowMesh["DeformedMesh" → True, "Field" → "Mises*", "Mesh" → True, "Show" -> "Window"]];
  step[[3]] ,
  If[step[[1]], SMTStepBack[]];
  SMTNextStep["Δλ" → step[[2]]]
 ];
ΔλRemesh = SMTRData["MultiplierIncrement"];
```

- show distribution of the Mises stress

In[154]:=
```
SMTShowMesh["DeformedMesh" → True, "Field" → "Mises*", "Mesh" → True, "BoundaryConditions" → True]
```



Step 2. Adaptive finite element refinement

- Mesh refinement is in Mathematica governed by the refinement function specific to the *ToElementMesh* mesh generator. Refinement function is of the form *refine[vertices,area]* and has the following properties:
  - receives vertices and the surface area of the the potential element;
  - must return **False** when potential element is acceptable accordingly to the chosen error estimator;
  - must return **True** when mesh should be refined at the area defined by the given vertices.
- Mesh refinement is global and will produce completely new mesh, refined and coarsened accordingly to the output of the refinement function. *ToElementMesh* function builds a new mesh solely based on the output of refinement function without the knowledge of the previous mesh. In order to accommodate the *ToElementMesh* function a scalar field $A_{max}(\boldsymbol{X})$ is constructed that defines a maximum are of the element as a function of spatial coordinates. $A_{max}(\boldsymbol{X})$ is constructed based on a ***posteriori recovery based error estimator*** as decribed in Zienkiewicz & Taylor, The finite element method, Vol 1, 2000. $A_{max}(\boldsymbol{X})$ field is discretized on the old mesh, but evaluated at the vertices of the elements of a new mesh.
- Definitions (see Zienkiewicz & Taylor, The finite element method, Vol 1, 2000)

  Let the exact solution of the problem be defined by the field $\mathbf{u}(\mathbf{X})$, $\hat{\mathbf{u}}(\mathbf{X})$ is an approximated solution calculated by the old mesh and $\mathbf{u}^*$ is a solution obtained by the patch recovery procedure. The recovered field $\mathbf{u}^*$ is smooth over the domain of the problem and it can be used as an approximation of the exact solution, thus $\mathbf{u}(\mathbf{X}) \approx \mathbf{u}^*(\mathbf{X})$ within a posteriori error estimator.
  Let also define:
  - $\overline{\eta}$ desired maximum relative error (e.g. 0.05)
  - $(||\mathbf{e}||)_k^2$ squared norm of the error in *k*-th element and $||\mathbf{e}||^2 = \sum (||\mathbf{e}||)_k^2$, where error is $\mathbf{e} = \mathbf{u}^* - \hat{\mathbf{u}}$ $((||\mathbf{e}||)_k^2$ (≡ek2))
  - $\overline{e}_m = \overline{\eta}\left(\frac{||\hat{u}||^2+||e||^2}{n_e}\right)^{\frac{1}{2}}$ is desired averaged error
  - $\xi_k = \frac{||\mathbf{e}||_k}{\overline{e}_m}$ is the error ratio of the *k*-th element. The element shall be refined whenever $\xi_k > 1$.
  - Assuming that $||\mathbf{e}||_k \propto h_k^p$, where h_k is the current element size and p the polynomial order of approximation, then $A_k^{max} = \xi_k^{-\frac{2}{p}} A_k$ where A_k is the area of the current element and A_k^{max} the maximum area of the refined element.
- The old AceFEM session is still active, thus AceFEM functions are evaluated at the nodes and elements of the old mesh. The following AceFEM functions are used:
  - u*=SMTPostData[*u_code*] returns nodal values of the field defined by keyword *u_code* and obtained by the patch recovery procedure.
  - û=SMTPost[*u_code*,"Smooth"->False] returns values of the field defined by keyword *u_code* evaluated at the nodes of the element for all elements
  - SMTPostData[A_{max},Point[*vertices*]] returns A_{max} evaluated at an arbitrary set of *vertices* of new mesh

Step 2.1 Recovery based error estimator

- For the sake of simplicity, the Mises stress is chosen as for the error estimator quantity. Mises stress is defined for all solid elements in Main library (ML), thus the presented algorithm is element formulation independent and should work for all elements from the library. More advanced error estimators would require addition element dependent functionality.
- Here the quantities u*≡uS, $\hat{u}$≡uh, $(\|\mathbf{e}\|)_k^2$ ≡ ek2 , $\|\mathbf{e}\|^2$ ≡ e2, $\|\hat{u}\|^2$ ≡ uh2, $\overline{\eta}$ ≡ηb, $\overline{e}_m$≡emb, ξ_k = ξk and A_k^{max} = Amxk are evaluated accordingly to the definitions above.

In[155]:=
```
uCode = "Mises*";
uS = SMTPostData[uCode];
uh = SMTPost[uCode, "Smooth" -> False];
```

In[158]:=
```
ek2 = MapThread[(#1 - uS[[#2]]).(#1 - uS[[#2]]) / Length[#2] &, {uh, SMTElementData["Nodes"]}];
```

In[159]:=
```
e2 = Total[ek2];
uh2 = Total[Map[#.# / Length[#] &, uh]];
```

In[161]:=
```
ηb = 0.05;
emb = ηb Sqrt[(uh2 + e2) / SMTIData["NoElements"]];
```

In[163]:=
```
ξk = Sqrt[ek2] / emb;
```

In[164]:=
```
Amaxk = ξk^(-2) SMTElementData["Volume"];
```

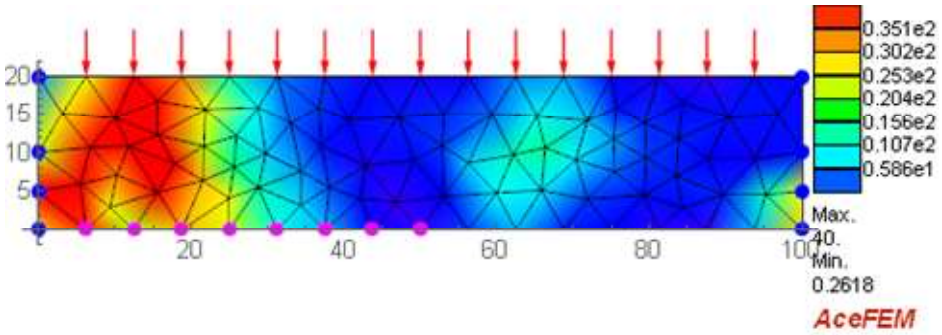
- Here the element based estimate of the maximum area of refined elements (A_k^{max}) is transform into smooth field over the nodes of the domain by averaging nodal values. The resulting field A_{max} is then further limited by the user chosen minimum and maximum element area.

In[165]:=
```
Amax = ConstantArray[0, SMTIData["NoNodes"]];
MapThread[ ( Amax[[#1]] += #2;) &, {SMTElementData["Nodes"], Amaxk}];
Amax = Amax / SMTNodeData["NoElements"];
```

In[168]:=
```
minArea = fullArea / 10000.;
maxArea = fullArea / 50.;
Amax = Map[Min[Max[#, minArea], maxArea] &, Amax];
```

- A_{max} field can be presented and manipulated in AceFEM as any other nodal based field.

In[171]:=
```
SMTShowMesh["DeformedMesh" → False, "Field" → Amax,
 "Mesh" → True, "BoundaryConditions" → True, Axes → True]
```



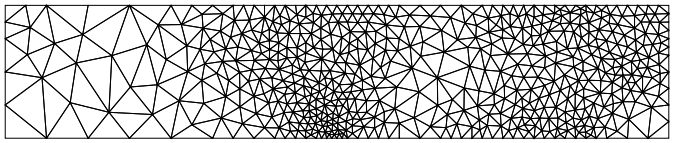
- With the A_{max} field defined the refinement function takes a very simple form.

In[172]:=
```
refine[vertices_, area_] := area > Min[SMTPostData[Amax, Point[vertices]]]
```

Step 2.2 Mew mesh generation with controlled refinement

In[173]:=
```
meshFine = ToElementMesh[
    Polygon[{{0, 0}, {L / 2, 0}, {L, 0}, {L, H}, {0, H}}]
    , MaxCellMeasure → maxArea
    , "MeshOrder" → 1
    , MeshRefinementFunction → refine
   ];
```

In[174]:=
```
meshFine["Wireframe"]
```



Step 3. Data transfer from old to new mesh

- new nodal data must be calculated while the old simulation is still active!
- apNew - nodal unknowns of the old mesh evaluated at the nodes of the new mesh in time *n+1*
- the procedure here assumes the all the nodes have the same number of unknowns and that all unknowns approximate the same vector field.

In[175]:=
```
ndof = SMTNodeSpecData["NoDOF"][[1]];
atNew = SMTPostData[
    Table[{"at", i}, {i, ndof}]
    , Point[meshFine["Coordinates"]]
   ];
```

- This starts a new simulation with new mesh and original material data and boundary conditions. Old simulation data does not exist beyond this point.

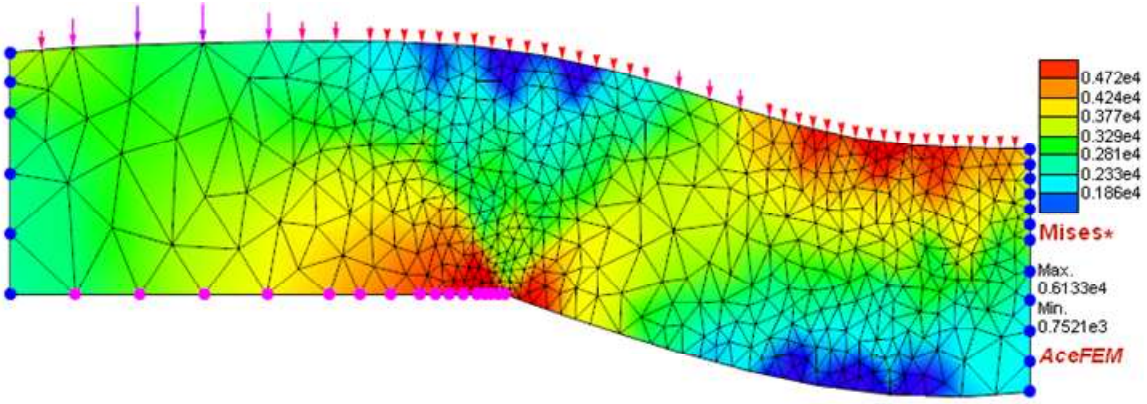
In[177]:=
```
startAceFEM[meshFine]
```

- This makes initial state of the new simulation to be the state at the end of the old simulation $\lambda = \lambda_{\text{Remesh}}$. For the unknowns with the prescribed essential boundary condition is the interpolated value wrong and it has to be replaced by the exact value.

In[178]:=
```
SMTNextStep["λ" → λRemesh];
```

In[179]:=
```
atBC = MapThread[If[#2 < 0, λRemesh #3, #1] &, {atNew, SMTNodeData["DOF"], SMTNodeData["dB"]}, 2];
SMTNodeData["at", atBC];
```

In[181]:=
```
SMTShowMesh["DeformedMesh" → True, "Field" → "Mises*", "Mesh" → True, "BoundaryConditions" → True]
```



Step 4. Establish new equilibrium after remeshing

The structure is after the refinement in general not in equilibrium. To make situation worse, the transition from the equilibrium to nonequilibrium is not continuous, thus the usual path following procedure can not be used to achieve the new equilibrium after remeshing. Here the algorithm is used where the first a new equilibrium is achieved by adding additional discrete nodal forces. Additional nodal forces can then be slowly removed using the path following procedure.

Calculate nodal forces that are not in equilibrium

- New equilibrium will be achieved for constant $\lambda = \lambda_{\text{Remesh}}$, thus $\Delta\lambda$=0.

In[182]:=
```
SMTNextStep["Δλ" → 0];
```

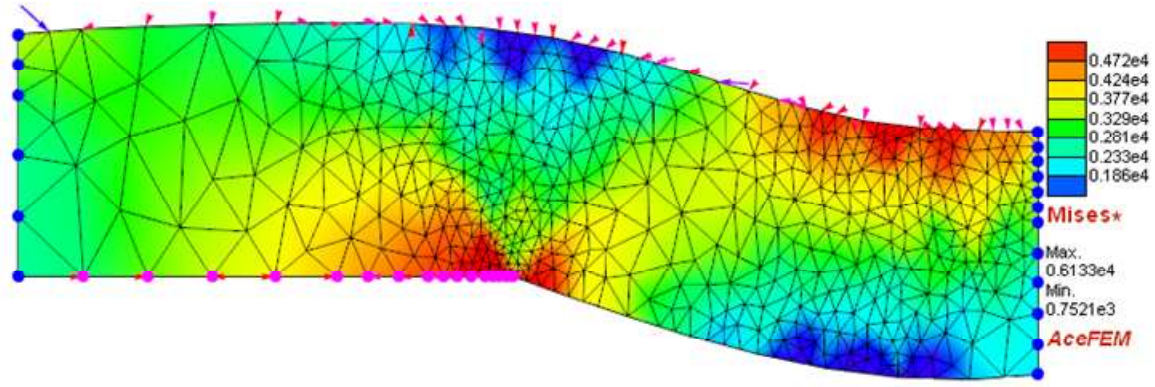
- BCEqui are nodal forces that has to be added to existing nodal forces to reestablish the equilibrium where BCRemesh = Bt $|_{\lambda=\lambda_{\text{Remesh}}}$.

In[183]:=
```
BCRemesh = SMTNodeData["Bt"];
BCEqui =
  MapThread[If[#2 < 0, 0, #1 - #3] &, {SMTResidual[All], SMTNodeData["DOF"], BCRemesh}, 2];
```

- Here the new natural boundary conditions are set. It is sufficient to set Bp, since Bt = Bp + $\Delta\lambda$ dB = Bp = BCRemesh + BCEqui.

In[185]:=
```
SMTNodeData["Bp", BCRemesh + BCEqui];
```

In[186]:=
```
SMTShowMesh["DeformedMesh" → True, "Field" → "Mises*", "Mesh" → True, "BoundaryConditions" → True]
```



- The structure is in perfect equilibrium for the modified natural BC.

In[187]:=
```
SMTNewtonIteration["EBCUpdate" -> "EBC to Bt"]
```

Out[187]=
$$2.76936 \times 10^{-16}$$

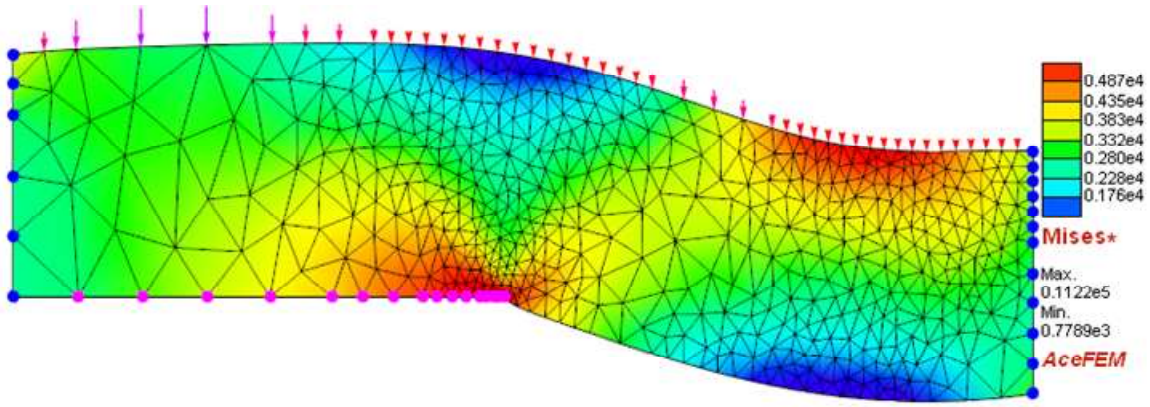
Remove additional nodal forces

- An alternative path following parameter $\gamma \in [0,1]$ is used here to create an alternative path following procedure that will remove additional nodal forces. Natural boundary condition are here calculates as
Bt = BCRemesh + (1 – γ) BCEqui + $\Delta\lambda$ dB = BCRemesh + (1 – γ) BCEqui = Bp, thus Bt $|_{\gamma=1}$ = Bt $|_{\lambda=\lambda_{\text{Remesh}}}$.

In[188]:=
```
SMTNextStep["Δγ" → 1];
While[
 γ = SMTRData["Parameter"];
 SMTNodeData["Bp", BCRemesh + (1 - γ) BCEqui];
 While[step = SMTConvergence[tolNR, maxNR, {"Adaptive γ", targetNR, .0001, 1, 1},
    "Alternate" → "Ignore"] , SMTNewtonIteration["EBCUpdate" -> "EBC to Bt"]];
 SMTStatusReport[];
 If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
 step[[3]] ,
 If[step[[1]], SMTStepBack[]];
 SMTNextStep["Δγ" → step[[2]]]
]
```

```
Step/Iter=3/5 λ/Δλ=0.8/0. γ/Δγ=1./1. ||Δp||/||R||=
 1.98426×10^-12/2.37274×10^-9 Events=0 Status=0/{Convergence}
```

In[190]:=
```
SMTShowMesh["DeformedMesh" → True, "Field" → "Mises*", "Mesh" → True, "BoundaryConditions" → True]
```



- The structure is again in perfect equilibrium.

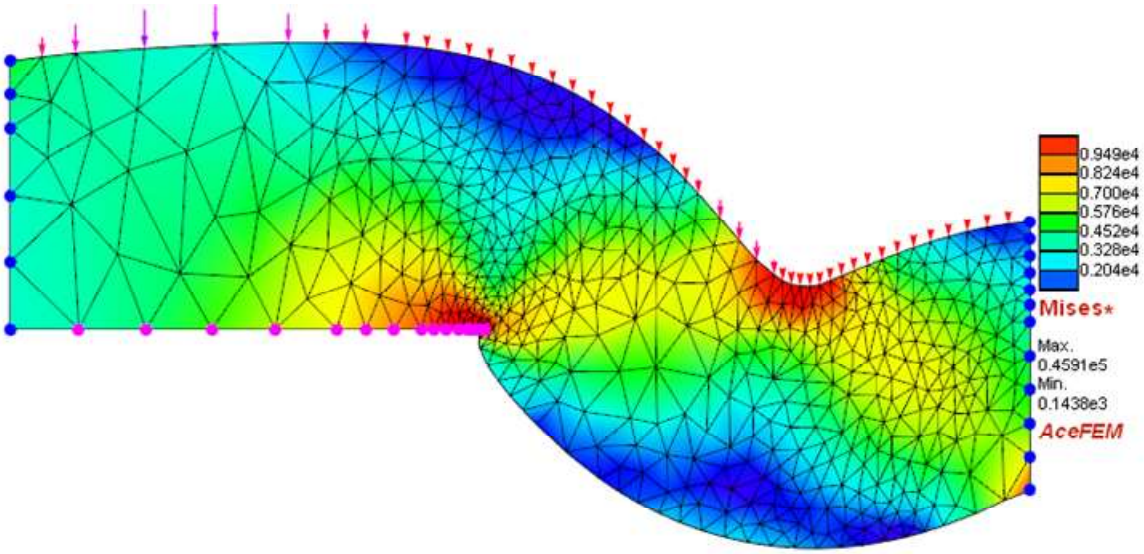
In[191]:=
```
SMTNewtonIteration[]
```

$$1.01187 \times 10^{-15}$$

Step 5. Run simulation after remeshing until $\lambda = \lambda_{Max}$

In[192]:=
```
SMTNextStep["Δλ" → ΔλMax];
While[
 While[step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λMax}]
  , SMTNewtonIteration[]
 ];
 If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
 If[Not[step[[1]]], SMTShowMesh["DeformedMesh" → True, "Field" → "Mises*",
   "Mesh" → True, "BoundaryConditions" → True, "Show" -> "Window"]];
 step[[3]] ,
 If[step[[1]], SMTStepBack[]];
 SMTNextStep["Δλ" → step[[2]]]
]
```

In[194]:=
```
SMTShowMesh["DeformedMesh" → True, "Field" → "Mises*", "Mesh" → True, "BoundaryConditions" → True]
```


0.949e4
0.824e4
0.700e4
0.576e4
0.452e4
0.328e4
0.204e4
Mises*
Max.
0.4591e5
Min.
0.1438e3
AceFEM

Analysis of cylindrical shell structure

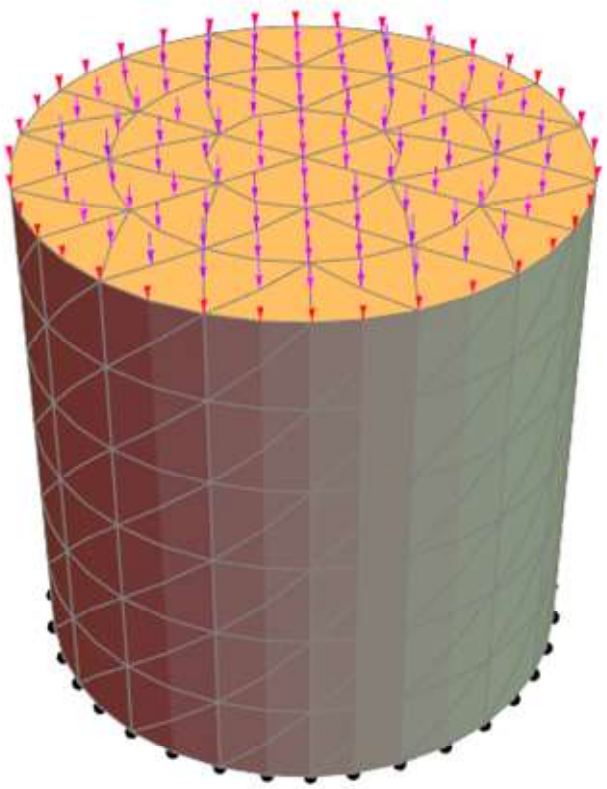
In[448]:=
```
<< AceFEM`;
R = 300; H = 600; t = 1;
surface = 2 π R H;
SMTInputData[];
SMTAddDomain[
  {"shell", {"ML:", "SE", "MS", "P2", "DF", "HY", "P2P6", {"D", "Fi"}, "SVenant"}, {"t *" → t}}];
```

- Create a mesh made of quadratic shell elements with the maximal element area "total surface"/10.

In[453]:=
```
SMTAddMesh[ToBoundaryMesh[Cylinder[{{0, 0, 0}, {0, 0, H}}, R],
   "MeshOrder" → 2, MaxCellMeasure → ("Area" -> surface / 10)], "shell"];
```

- Cylinder is clamped at the bottom and subjected to the surface load at the top.

In[454]:=
```
SMTAddEssentialBoundary["Z" == 0 && "ID" == "D" &, 1 → 0, 2 → 0, 3 → 0];
SMTAddNaturalBoundary["Z" == H && "ID" == "D" &, 3 -> Polygon[{-1}]];
SMTAnalysis[];
SMTShowMesh["BoundaryConditions" → True];
```

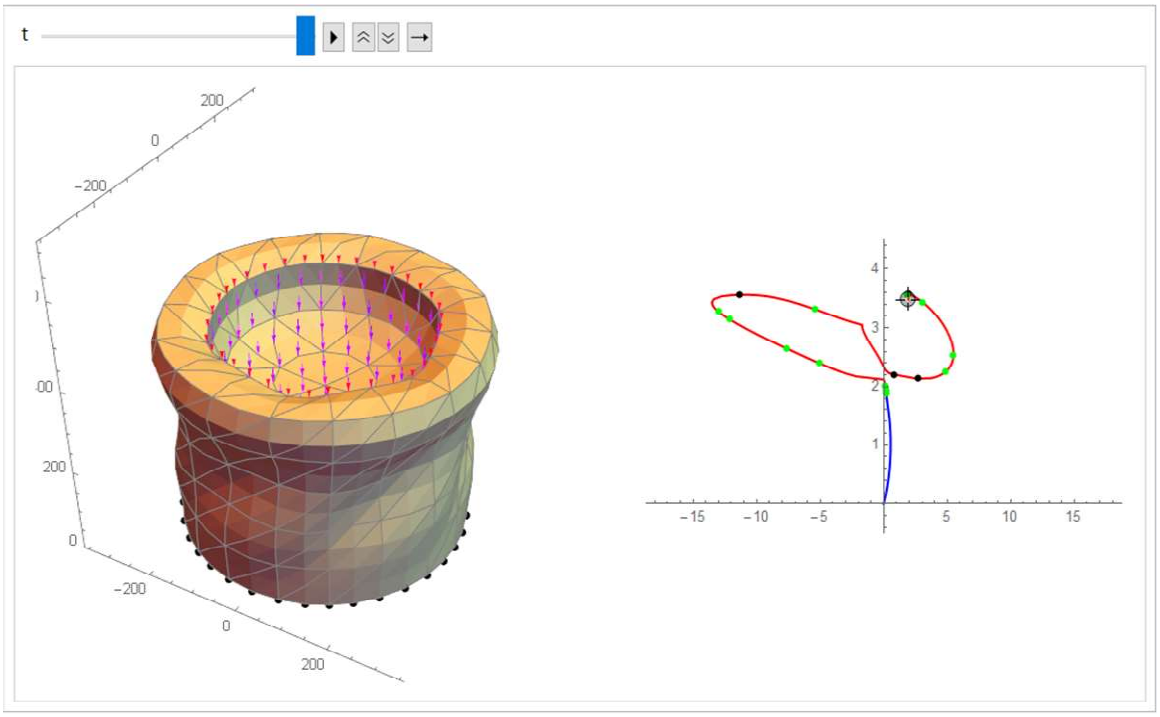


Initialization of the arc-length procedure and iterative procedure with adaptive parameter incrementation

In[458]:=
```
γMax = SMTArcLengthSet["λTarget" → 5];
```

In[459]:=
```
SMTEquilibriumPaths[
 "Method" -> "Arc Length"
 , "γMax" -> γMax
 , "Δγ0" -> γMax / 500
 , "ΔγMax" -> γMax / 500
 , "ΔγMin" -> γMax / 50 000
 , "MaxPaths" → 1
 , "OptimalIterations" → 10
 , "x" → Hold[SMTPostData["u", Point[{R, 0, H / 2}]]]
 , "ImageSize" → 400, "PlotMarkers" → False, "AnimationFrequency" → γMax / 100
 , "ShowMeshOptions" → {PlotRange → 1.2 {{-R, R}, {-R, R}, {0, H}}}
 , "PlotOptions" → {PlotRange → {{-15, 15}, {0, 4}}}
 , "Show" → {"ExportFrames", "tmpCylinder"}
 , "InitializeAnimationOfResponse" → {0, 0}
]
```

In[467]:=
```
SMTAnimationOfResponse["Animate"]
```



CHAPTER 5

AceGen-AceFEM Examples

Simple 2D Solid, Finite Strain Element

Generate two-dimensional, four node finite element for the analysis of the steady state problems in the mechanics of solids. The element has the following characteristics:

⇒ quadrilateral topology,

⇒ 4 node element,

⇒ izoparametric mapping from the reference to the actual frame,

⇒ global unknowns are displacements of the nodes,

⇒ the element should allow arbitrary large displacements and rotations,

⇒ the problem is defined by the hyperelastic Neo-Hooke type strain energy potential,

$W = \frac{\lambda}{2}(\det \mathbf{F} - 1)^2 + \mu\left(\frac{\mathrm{Tr}[\mathbf{C}]-3}{2} - \mathrm{Log}[\det \mathbf{F}]\right)$,

where $\mathbf{C} = \mathbf{F}^T \mathbf{F}$ is right Cauchy-Green tensor, $\mathbf{F} = \mathbf{I} + \nabla \mathbf{u}$ is deformation gradient,

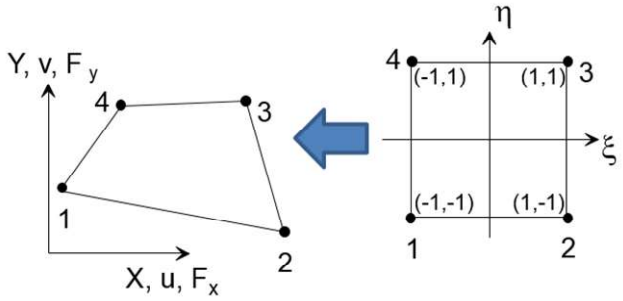
$\boldsymbol{u}$ is displacements field, Ω_0 is the initial domain of the problem and λ, μ are the first and the second Lame's material constants.

The following user subroutines have to be generated:

⇒ user subroutine for the direct implicit analysis,

⇒ user subroutine for the post-processing that returns the Green-Lagrange strain tensor and

the Cauchy stress tensor and main stresses.



In[220]:=
```
<< "AceGen`";
SMSInitialize["ExamplesHypersolid2D", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "Q1", "SMSSymmetricTangent" → True,
  "SMSDomainDataNames" -> {"E -elastic modulus", "ν -poisson ratio", "t -thickness"},
  "SMSDefaultData" -> {21000, 0.3, 1}
 ];
```

Definitions of geometry, kinematics, strain energy ...

In[223]:=
```
ElementDefinitions[] := (
  Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
  {XIO, uIO} ⊨ SMSIO["All coordinates and DOFs"];
  Nh ⊨ 1 / 4 {(1 - ξ) (1 - η), (1 + ξ) (1 - η), (1 + ξ) (1 + η), (1 - ξ) (1 + η)};
  SMSFreeze[X, Append[Nh.XIO, ζ]];
  Je ⊨ SMSD[X, Ξ]; Jed ⊨ Det[Je];
  u ⊨ Append[Nh.uIO, 0];
  H ⊨ SMSD[u, X, "Dependency" → {Ξ, X, SMSInverse[Je]}];
  SMSFreeze[F, IdentityMatrix[3] + H, "Ignore" → PossibleZeroQ];
  JF ⊨ Det[F]; Ct ⊨ Transpose[F].F;
  {Em, ν, tζ} ⊨ SMSIO["Domain data"];
  {λ, μ} ⊨ SMSHookeToLame[Em, ν];
  W ⊨ 1 / 2 λ (JF - 1)^2 + μ (1 / 2 (Tr[Ct] - 3) - Log[JF]);
  wgp ⊨ SMSIO["Integration weight"[Ig]];
  pe ⊨ Flatten[uIO];
  fGauss ⊨ Jed tζ;
 )
```

"Tangent and residual" user subroutine

In[224]:=
```
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
SMSDo[
  Rg ⊨ fGauss SMSD[W, pe, i];
  SMSIO[wgp Rg, "Add to", "Residual"[i]];
  SMSDo[
   Kg ⊨ SMSD[Rg, pe, j];
   SMSIO[wgp Kg, "Add to", "Tangent"[i, j]];
   , {j, i, SMSNoDOFGlobal}];
  , {i, 1, SMSNoDOFGlobal}];
SMSEndDo[];
```

"Postprocessing" user subroutine

In[229]:=
```
SMSStandardModule["Postprocessing"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
SMSIO[{"DeformedMeshX" -> uIO[[All, 1]], "DeformedMeshY" -> uIO[[All, 2]],
    "u" -> uIO[[All, 1]], "v" -> uIO[[All, 2]]}, "Export to", "Nodal point post"];
Eg ⊨ 1 / 2 (Ct - IdentityMatrix[3]);
σ ⊨ (1 / JF) * SMSD[W, F, "Ignore" -> NumberQ] . Transpose[F];
SMSIO[{"Exx" -> Eg[[1, 1]], "Exy" -> Eg[[1, 2]], "Eyy" -> Eg[[2, 2]]
    , "Sxx" → σ[[1, 1]], "Sxy" → σ[[1, 2]], "Syy" → σ[[2, 2]], "Szz" → σ[[3, 3]]}
  , "Export to", "Integration point post"[Ig]];


SMSEndDo[];
```

Code generation

In[238]:=
```
SMSWrite[];
```

```
File: ExamplesHypersolid2D.c  Size: 11729  Time: 2
```

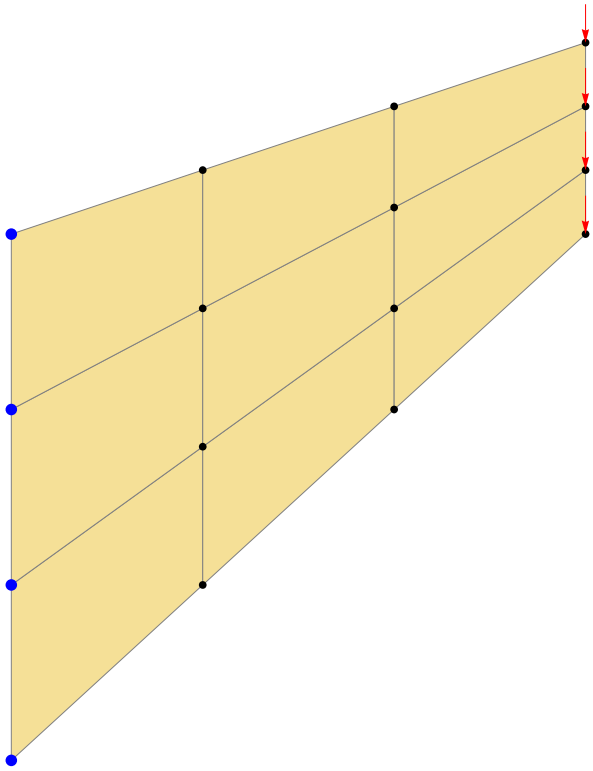
Method	SKR	SPP
No.Formulae	95	85
No.Leafs	1402	1344

Cook`s membrane test

In[239]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[{"Test", "ExamplesHypersolid2D", {"E *" -> 1, "ν *" -> 0}}];
SMTAddMesh[Polygon[{{0, 0}, {48, 44}, {48, 60}, {0, 44}}], "Test", "Division" -> {3, 3}];
SMTAddEssentialBoundary[ "X" == 0 &, 1 → 0, 2 → 0];
SMTAddNaturalBoundary[ "X" == 48 &, 2 → -.1];
SMTAnalysis[];
```

In[246]:=
```
Do[SMTNextStep["Δλ" → 0.1];
  While[SMTConvergence[10^-8, 10], SMTNewtonIteration[];];
  , {i, 1, 10}];
```

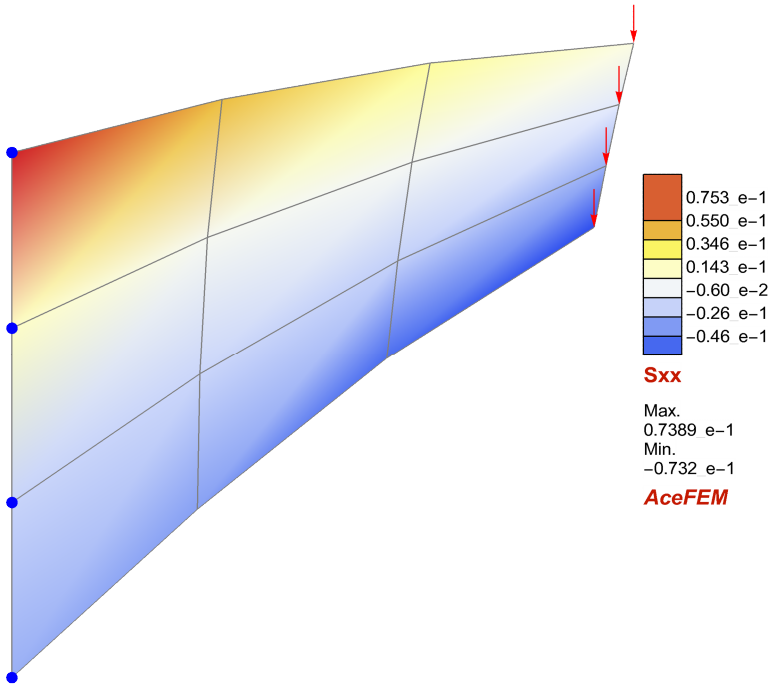
In[247]:=
```
SMTShowMesh["NodeMarks" → True, "BoundaryConditions" -> True]
```



In[248]:=
```
SMTNodeData[Point[{48, 44 + 16}], "at"]
```

```
{{4.06515, -6.87855}}
```

In[249]:= `SMTShowMesh["DeformedMesh" → True, "BoundaryConditions" → True, "Field" → "Sxx"]`

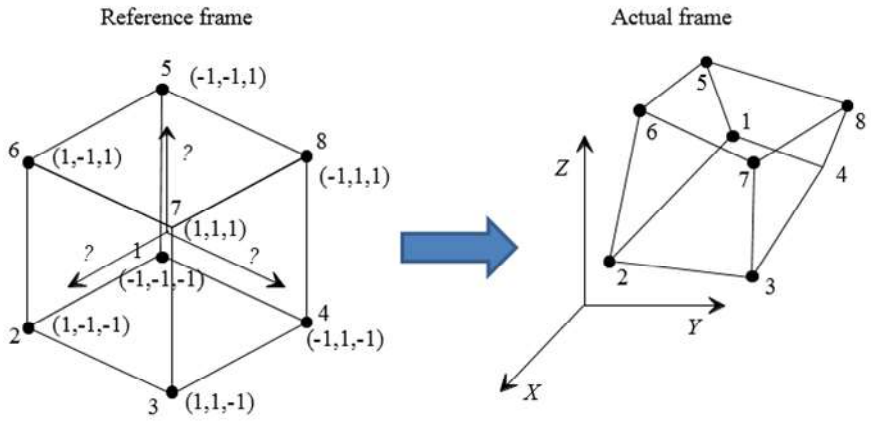


Mixed 3D Solid FE, Elimination of Local Unknowns

Description

Generate the three-dimensional, eight node finite element for the analysis of hyperelastic solid problems. The element has the following characteristics:

- hexahedral topology,
- 8 nodes,
- izoparametric mapping from the reference to the actual frame,



- global unknowns are displacements of the nodes,
  $u = u_i N_i,\ v = v_i N_i,\ w = w_i N_i$
- enhanced strain formulation to improve shear and volumetric locking response,

$$\mathbf{H} = \begin{pmatrix} u_{,X} & u_{,Y} & u_{,Z} \\ v_{,X} & v_{,Y} & v_{,Z} \\ w_{,X} & w_{,Y} & w_{,Z} \end{pmatrix}$$

$$\tilde{\mathbf{H}} = \mathbf{H} + \frac{\text{Det}[\mathbf{J}_0]}{\text{Det}[\mathbf{J}]} \begin{pmatrix} \xi\alpha_1 & \eta\alpha_2 & \zeta\alpha_3 \\ \xi\alpha_4 & \eta\alpha_5 & \zeta\alpha_6 \\ \xi\alpha_7 & \eta\alpha_8 & \zeta\alpha_9 \end{pmatrix} . \mathbf{J}_0^{-1}$$

  where $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_9\}$ are internal degrees of freedom
  $\tilde{\mathbf{H}}$ is incompatible
- the classical hyperelastic Neo-Hooke's potential energy,
  $W = \frac{\lambda}{2}(\det \mathbf{F} - 1)^2 + \mu\left(\frac{\text{Tr}[\mathbf{C}]-3}{2} - \text{Log}[\det \mathbf{F}]\right)$,
  where $\mathbf{C} = \mathbf{F}^T \mathbf{F}$ is right Cauchy-Green tensor, $\mathbf{F} = \mathbf{I} + \tilde{\mathbf{H}}$ is enhanced deformation gradient.
- eliminate internal degrees of freedom at the element level by the static condensation procedure (see `Elimination of Local Unknowns`).
- internal degrees of freedom at the element level and condensation procedure data are stored in data structure ed$$["ht",i] ("Element time dependent data" keyword) specified by template constants "SMSNoDOFCondense"→9,"SMSNoTimeStorage"→9,"SMSCondensationData"→ed$$["ht",1] (see `Elimination of Local Unknowns`)

Solution

In[1]:=
```
<< "AceGen`";
SMSInitialize["ExamplesHypersolid3D", "Environment" → "AceFEM"];
SMSTemplate[
  "SMSTopology" → "H1",
  "SMSNoDOFCondense" → 9, "SMSNoTimeStorage" → 9, "SMSCondensationData" → ed$$["ht", 1],
  "SMSDomainDataNames" -> {"E -elastic modulus", "ν -poisson ratio"},
  "SMSDefaultData" -> {21000, 0.3}
 ];
```

TestExamples

In[4]:=
```
ElementDefinitions[] := (
  Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
  {XIO, uIO} ⊨ SMSIO["All coordinates and DOFs"];
  Ξn = {{-1, -1, -1}, {1, -1, -1}, {1, 1, -1}, {-1, 1, -1},
    {-1, -1, 1}, {1, -1, 1}, {1, 1, 1}, {-1, 1, 1}};
  Nh ⊨ Table[1 / 8 (1 + ξ Ξn⟦i, 1⟧) (1 + η Ξn⟦i, 2⟧) (1 + ζ Ξn⟦i, 3⟧), {i, 1, 8}];
  SMSFreeze[X, Nh.XIO]; Je ⊨ SMSD[X, Ξ]; Jed ⊨ Det[Je];
  pe = Flatten[uIO]; u ⊨ Nh.uIO;
  H ⊨ SMSD[u, X, "Dependency" → {Ξ, X, SMSInverse[Je]}];
  J0 ⊨ SMSReplaceAll[Je, {ξ → 0, η → 0, ζ → 0}]; J0d ⊨ Det[J0];
  αe ⊨ Table[SMSIO["Element time dependent data"[i]], {i, SMSNoDOFCondense}];
  ph = Join[pe, αe];
  HbΞ = ( | ξ αe⟦1⟧ | η αe⟦2⟧ | ζ αe⟦3⟧ |
          | ξ αe⟦4⟧ | η αe⟦5⟧ | ζ αe⟦6⟧ |
          | ξ αe⟦7⟧ | η αe⟦8⟧ | ζ αe⟦9⟧ | ); Hb ⊨ J0d / Jed HbΞ.SMSInverse[J0];
  SMSFreeze[F, IdentityMatrix[3] + H + Hb]; JF ⊨ Det[F]; Ct ⊨ Transpose[F].F;
  {Em, ν} ⊨ SMSIO["Domain data"];
  {λ, μ} ⊨ SMSHookeToLame[Em, ν];
  W ⊨ 1 / 2 λ (JF - 1) ^ 2 + μ (1 / 2 (Tr[Ct] - 3) - Log[JF]);
  wgp ⊨ SMSIO["Integration weight"[Ig]];
)
```

In[5]:=
```
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
SMSDo[
  Rg ⊨ Jed  SMSD[W, ph, i];
  SMSIO[wgp Rg, "Add to", "Residual"[i]];
  SMSDo[
   Kg ⊨ SMSD[Rg, ph, j];
   SMSIO[wgp Kg, "Add to", "Tangent"[i, j]];
   , {j, i, SMSNoAllDOF}];
  , {i, 1, SMSNoAllDOF}];
SMSEndDo[];
```

In[10]:=
```
SMSStandardModule["Postprocessing"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
SMSIO[{"DeformedMeshX" -> uIO[[All, 1]],
    "DeformedMeshY" -> uIO[[All, 2]], "DeformedMeshZ" -> uIO[[All, 3]], "u" -> uIO[[All, 1]],
    "v" -> uIO[[All, 2]], "w" -> uIO[[All, 3]}, "Export to", "Nodal point post"];
Eg := 1 / 2 (Ct - IdentityMatrix[3]);
σ := (1 / JF) * SMSD[W, F, "Ignore" -> NumberQ] . Transpose[F];
SMSIO[{"Exx" -> Eg[[1, 1]], "Exy" -> Eg[[1, 2]], "Exz" -> Eg[[1, 3]], "Eyx" -> Eg[[2, 1]],
    "Eyy" -> Eg[[2, 2]], "Eyz" -> Eg[[2, 3]], "Ezx" -> Eg[[3, 1]], "Ezy" -> Eg[[3, 2]], "Ezz" -> Eg[[3, 3]]
    , "Sxx" → σ[[1, 1]], "Sxy" → σ[[1, 2]], "Sxz" → σ[[1, 3]], "Syx" → σ[[2, 1]], "Syy" → σ[[2, 2]],
    "Syz" → σ[[2, 3]], "Szx" → σ[[3, 1]], "Szy" → σ[[3, 2]], "Szz" → σ[[3, 3]]},
  "Export to", "Integration point post"[Ig]];
SMSEndDo[];
```

In[18]:=
```
SMSWrite[];
```

File: ExamplesHypersolid3D.c Size: 35 934 Time: 13

Method	SKR	SPP
No.Formulae	321	268
No.Leafs	6565	4583

Cantilever beam example H1E9

In[19]:=
```
<< AceFEM`;
SMTInputData["Console" → True];
SMTAddDomain["A", "ExamplesHypersolid3D", {"E *" -> 1000., "ν *" -> .499}];
SMTAddEssentialBoundary[{ "X" == 0 &, 1 -> 0, 2 -> 0, 3 -> 0}, { "X" == 10 &, 3 -> -1}];
SMTAddMesh[
  Hexahedron[{{0, 0, 0}, {10, 0, 0}, {10, 2, 0}, {0, 2, 0},
    {0, 0, 3}, {10, 0, 2}, {10, 2, 2}, {0, 2, 3}}], "A", "H1", {15, 6, 6}];
SMTAnalysis[];
```

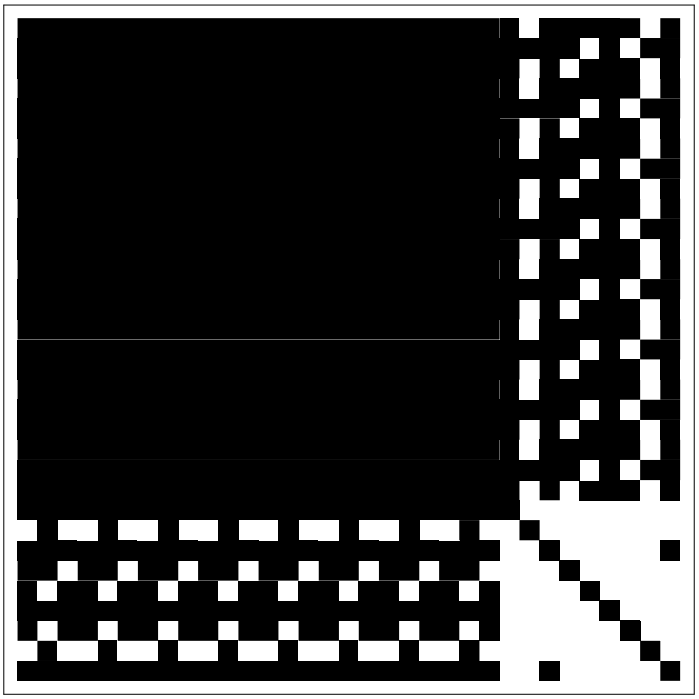
Analysis of condensation procedures

- Analyse full (not condensed) tangent matrix of first element.

In[25]:=
```
K = SMTData[1, "SKR"][[1]];
K // Dimensions
K // Eigenvalues // Chop
ArrayPlot[K, ColorFunction -> (If[Chop[#] == 0, White, Black] &), ColorFunctionScaling → False]
```

```
{33, 33}
```

```
{239 613., 140 525., 134 372., 94 940.6, 9194.64, 4196.41, 2321.89, 470.765, 453.775,
 280.202, 239.896, 217.947, 215.632, 206.161, 185.328, 172.379, 145.628, 136.451, 118.9,
 116.18, 100.794, 72.9595, 58.2749, 57.4526, 29.4289, 21.9266, 11.0545, 0, 0, 0, 0, 0, 0}
```

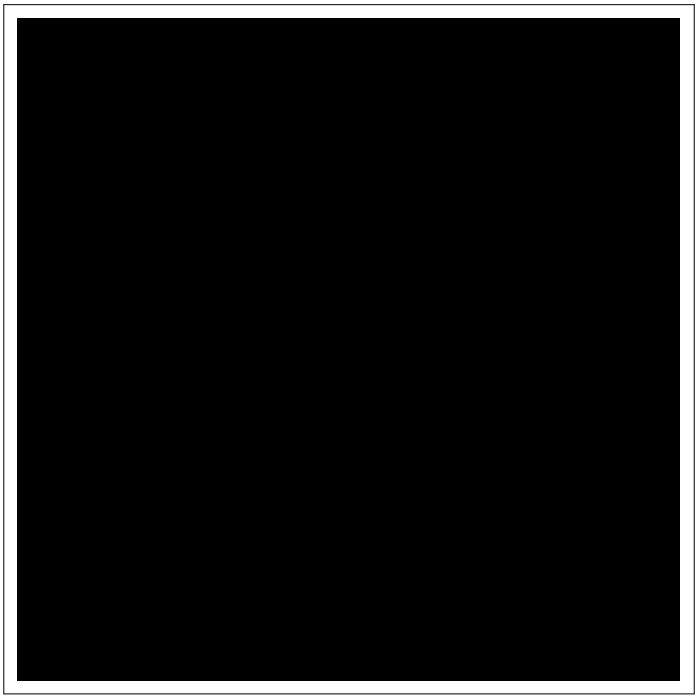


- Analyse statically condensed tangent matrix of first element.

In[29]:=
```
KCond = SMTData[1, "CKR"][[1]];
KCond // Dimensions
KCond // Eigenvalues // Chop
ArrayPlot[KCond, ColorFunction -> (If[Chop[#] == 0, White, Black] &),
 ColorFunctionScaling → False]
```

```
{24, 24}
```

```
{140 524., 9194.91, 4196.41, 2321.89, 254.696, 239.896,
 235.978, 215.738, 206.153, 120.556, 118.863, 116.195, 100.796,
 64.5585, 57.4551, 38.3077, 34.1006, 11.0548, 0, 0, 0, 0, 0, 0}
```



- Manually statically condensed tangent matrix of first element.

In[33]:=
```
KK = SMTData[1, "SKR"][[1]];
Kuu = Take[KK, 24, 24];
nModes = 9; Kqiqi = Take[KK, -nModes, -nModes]; L = Take[KK, 24, -nModes];
KCond = Kuu - L.Inverse[Kqiqi].Transpose[L];
KCond // Eigenvalues // Chop
```

```
{140 524., 9194.91, 4196.41, 2321.89, 254.696, 239.896,
 235.978, 215.738, 206.153, 120.556, 118.863, 116.195, 100.796,
 64.5585, 57.4551, 38.3077, 34.1006, 11.0548, 0, 0, 0, 0, 0, 0}
```

Simulation of response.

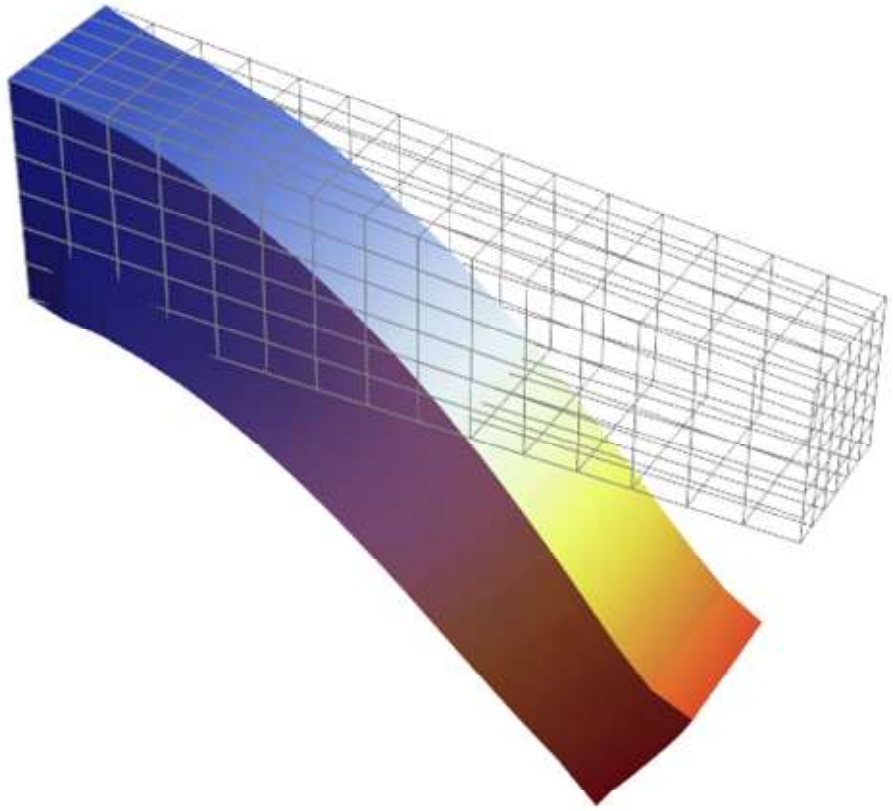
In[38]:=
```
SMTNextStep["λ" → 0.5];
While[
  While[
   step = SMTConvergence[10^-8, 10, {"Adaptive BC", 8, .001, 1, 5}], SMTNewtonIteration[];];
  SMTStatusReport[];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" → step[[2]]]
 ];
```

```
Step/Iter=1/7 λ/Δλ=0.5/0.5 ||Δp||/||R||=
 1.25179×10^-12/4.47919×10^-10 Events=0 Status=0/{Convergence}

Step/Iter=2/7 λ/Δλ=1.13265/0.632653 ||Δp||/||R||=
 6.0188×10^-10/1.35321×10^-6 Events=0 Status=0/{Convergence}

Step/Iter=3/4 λ/Δλ=1.93315/0.8005 ||Δp||/||R||=
 12.5469/2.33383×10^10 Events=0 Status=0/{Global divergence}

Step/Iter=3/6 λ/Δλ=1.5329/0.40025 ||Δp||/||R||=
 102.107/1.77954×10^10 Events=0 Status=0/{Global divergence}

Step/Iter=3/5 λ/Δλ=1.33278/0.200125 ||Δp||/||R||=
 6.63423×10^-12/4.67695×10^-10 Events=0 Status=0/{Convergence}

Step/Iter=4/6 λ/Δλ=1.66768/0.334903 ||Δp||/||R||=
 6.42061×10^-15/8.30036×10^-11 Events=0 Status=0/{Convergence}

Step/Iter=5/5 λ/Δλ=2.16662/0.498937 ||Δp||/||R||=
 117.951/16983.1 Events=0 Status=0/{Global divergence}

Step/Iter=5/5 λ/Δλ=1.91715/0.249469 ||Δp||/||R||=
 1.38182×10^-10/8.51874×10^-9 Events=0 Status=0/{Convergence}

Step/Iter=6/11 λ/Δλ=2.33463/0.417478 ||Δp||/||R||=
 7.65364/8.41869×10^16 Events=0 Status=0/{MaxIterations}

Step/Iter=6/5 λ/Δλ=2.12589/0.208739 ||Δp||/||R||=
 3.12864×10^-11/9.42173×10^-10 Events=0 Status=0/{Convergence}

Step/Iter=7/6 λ/Δλ=2.47521/0.349318 ||Δp||/||R||=
 4.80618×10^-16/6.52611×10^-12 Events=0 Status=0/{Convergence}

Step/Iter=8/7 λ/Δλ=2.99562/0.520413 ||Δp||/||R||=
 1.15158×10^-11/5.63641×10^-9 Events=0 Status=0/{Convergence}

Step/Iter=9/5 λ/Δλ=3.6541/0.658482 ||Δp||/||R||=
 13.7771/2.6943×10^8 Events=0 Status=0/{Global divergence}

Step/Iter=9/5 λ/Δλ=3.32486/0.329241 ||Δp||/||R||=
 2.78856×10^-10/2.65832×10^-8 Events=0 Status=0/{Convergence}

Step/Iter=10/9 λ/Δλ=3.87584/0.550975 ||Δp||/||R||=
 6.68859×10^-15/1.17487×10^-11 Events=0 Status=0/{Convergence}

Step/Iter=11/5 λ/Δλ=4.35794/0.482103 ||Δp||/||R||=
 57.2863/2.91167×10^12 Events=0 Status=0/{Global divergence}

Step/Iter=11/5 λ/Δλ=4.11689/0.241051 ||Δp||/||R||=
 1.05182×10^-11/3.8243×10^-10 Events=0 Status=0/{Convergence}

Step/Iter=12/6 λ/Δλ=4.52028/0.403392 ||Δp||/||R||=
 1.69747×10^-12/3.29004×10^-9 Events=0 Status=0/{Convergence}

Step/Iter=13/7 λ/Δλ=5./0.479721 ||Δp||/||R||=
 7.71464×10^-10/5.08123×10^-7 Events=0 Status=0/{Convergence}
```

In[40]:=
```
SMTPostData[{"u", "v", "w", "Sxx", "Exx"}, Point[{10, 1, 1}]]
```

$$\{-1.70042,\ 1.10773\times10^{-15},\ -5.,\ 49.6655,\ -0.0967131\}$$

In[41]:=
```
Show[SMTShowMesh["FillElements" -> False],
 SMTShowMesh["DeformedMesh" → True, "Mesh" -> False, "Field" -> Map[Norm, SMTNodeData["at"]]]]
```



Mixed 3D Solid FE, Auxiliary Nodes

Description

Generate the three-dimensional, eight node finite element for the analysis of hyperelastic solid problems as described in example Mixed 3D Solid FE. However, instead of eliminating internal degrees of freedom at the element level, create for each element an additional auxiliary node where the additional unknowns are stored.

Solution

Created element has 8 topological (topology H1) and 1 auxiliary node, thus all together 9 nodes. For each element an auxiliary node (specified by the -LP switch as described in Node Identification) is created with the following characteristics:

- node has node identification "EAS"
- node has no coordinates and is not shown on graphs
- node holds all 9 local unknowns

In[1]:=
```
<< "AceGen`";
SMSInitialize["ExamplesHypersolid3DB", "Environment" → "AceFEM"];
SMSTemplate[
  "SMSTopology" → "H1",
  "SMSNoNodes" → 9,
  "SMSDOFGlobal" → {3, 3, 3, 3, 3, 3, 3, 3, 9},
  "SMSNodeID" → {"D", "D", "D", "D", "D", "D", "D", "D", "EAS -LP"},
  "SMSAdditionalNodes" → Function[{}, {Null}],
  "SMSSymmetricTangent" → True,
   "SMSDomainDataNames" -> {"E -elastic modulus", "ν -poisson ratio"},
  "SMSDefaultData" -> {21000, 0.3}
 ];
```

TestExamples

In[4]:=
```
ElementDefinitions[] := (
  Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
  XIO ⊨ SMSIO["Nodal coordinates"][[1 ;; 8]];
  Ξn = {{-1, -1, -1}, {1, -1, -1}, {1, 1, -1}, {-1, 1, -1},
    {-1, -1, 1}, {1, -1, 1}, {1, 1, 1}, {-1, 1, 1}};
  Nh ⊨ Table[1 / 8 (1 + ξ Ξn[[i, 1]]) (1 + η Ξn[[i, 2]]) (1 + ζ Ξn[[i, 3]]), {i, 1, 8}];
  SMSFreeze[X, Nh.XIO]; Je ⊨ SMSD[X, Ξ]; Jed ⊨ Det[Je];
  ph ⊨ SMSIO["Nodal DOFs"];
  pe = Flatten[ph];
  uIO = ph[[1 ;; 8]]; u ⊨ Nh.uIO;
  H ⊨ SMSD[u, X, "Dependency" → {Ξ, X, SMSInverse[Je]}];
  J0 ⊨ SMSReplaceAll[Je, {ξ → 0, η → 0, ζ → 0}]; J0d ⊨ Det[J0];
  αe = ph[[9]];
  HbΞ = ( {{ξ αe[[1]], η αe[[2]], ζ αe[[3]]},
           {ξ αe[[4]], η αe[[5]], ζ αe[[6]]},
           {ξ αe[[7]], η αe[[8]], ζ αe[[9]]}} ); Hb ⊨ J0d / Jed HbΞ.SMSInverse[J0];
  SMSFreeze[F, IdentityMatrix[3] + H + Hb];
  JF ⊨ Det[F];
  Ct ⊨ Transpose[F].F;
  {Em, ν} ⊨ SMSIO["Domain data"];
  {λ, μ} ⊨ SMSHookeToLame[Em, ν];
  W ⊨ 1 / 2 λ (JF - 1) ^ 2 + μ (1 / 2 (Tr[Ct] - 3) - Log[JF]);
  wgp ⊨ SMSIO["Integration weight"[Ig]];
)
```

In[5]:=
```
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
SMSDo[
  Rg ⊨ Jed SMSD[W, pe, i];
  SMSIO[wgp Rg, "Add to", "Residual"[i]];
  SMSDo[
   Kg ⊨ SMSD[Rg, pe, j];
   SMSIO[wgp Kg, "Add to", "Tangent"[i, j]];
   , {j, i, SMSNoDOFGlobal}];
  , {i, 1, SMSNoDOFGlobal}];
SMSEndDo[];
```

In[10]:=
```
SMSStandardModule["Postprocessing"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
SMSIO[{"DeformedMeshX" -> uIO[[All, 1]],
    "DeformedMeshY" -> uIO[[All, 2]], "DeformedMeshZ" -> uIO[[All, 3]], "u" -> uIO[[All, 1]],
    "v" -> uIO[[All, 2]], "w" -> uIO[[All, 3]]}, "Export to", "Nodal point post"];
Eg = 1 / 2 (Ct - IdentityMatrix[3]);
σ = (1 / JF) * SMSD[W, F, "Ignore" -> NumberQ] . Transpose[F];
SMSIO[{"Exx" -> Eg[[1, 1]], "Exy" -> Eg[[1, 2]], "Exz" -> Eg[[1, 3]], "Eyx" -> Eg[[2, 1]],
    "Eyy" -> Eg[[2, 2]], "Eyz" -> Eg[[2, 3]], "Ezx" -> Eg[[3, 1]], "Ezy" -> Eg[[3, 2]], "Ezz" -> Eg[[3, 3]]
    , "Sxx" → σ[[1, 1]], "Sxy" → σ[[1, 2]], "Sxz" → σ[[1, 3]], "Syx" → σ[[2, 1]], "Syy" → σ[[2, 2]],
    "Syz" → σ[[2, 3]], "Szx" → σ[[3, 1]], "Szy" → σ[[3, 2]], "Szz" → σ[[3, 3]]},
  "Export to", "Integration point post"[Ig]];
SMSEndDo[];
```

In[18]:=
```
SMSWrite[];
```

File: ExamplesHypersolid3DB.c Size: 35 639 Time: 13

Method	SKR	SPP
No.Formulae	321	268
No.Leafs	6574	4592

Simple test example

In[19]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["A", "ExamplesHypersolid3DB", {"E *" -> 1000., "ν *" -> .3}];
SMTAddEssentialBoundary[{ "X" == 0 &, 1 -> 0, 2 -> 0, 3 -> 0}, { "X" == 10 &, 3 -> -1}];
SMTAddMesh[
  Hexahedron[{{0, 0, 0}, {10, 0, 0}, {10, 2, 0}, {0, 2, 0},
    {0, 0, 3}, {10, 0, 2}, {10, 2, 2}, {0, 2, 3}}], "A", "H1", {2, 1, 1}];
SMTAnalysis[];
SMTNextStep["λ" → 1];
SMTNewtonIteration[];
```

Example has:

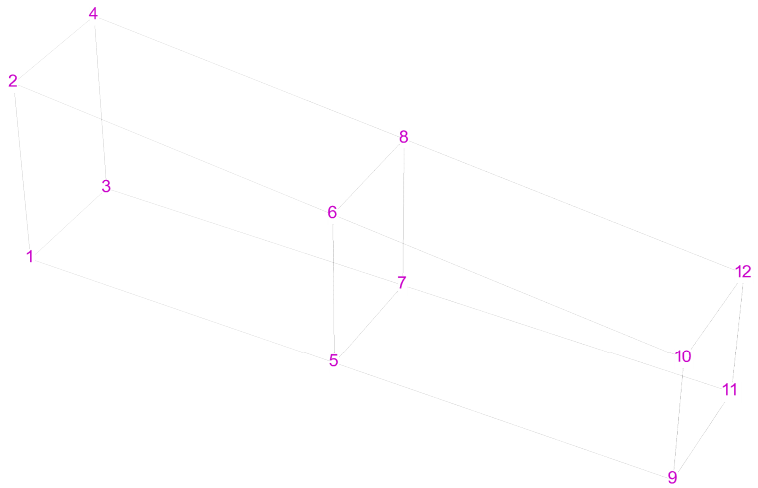
- 2 elements
- 12 topological nodes (node identification "D") with 3 do.o.f. each
- 2 auxiliary nodes (node identification "EAS") with 9 d.o.f each

In[27]:=
```
{SMTNodeData["NodeIndex"], SMTNodeData["NodeID"],
  SMTNodeData["X"], SMTNodeData["DOF"]} // Transpose // MatrixForm
```

1	D	{0., 0., 0.}	{−1, −1, −1}
2	D	{0., 0., 3.}	{−1, −1, −1}
3	D	{0., 2., 0.}	{−1, −1, −1}
4	D	{0., 2., 3.}	{−1, −1, −1}
5	D	{5., 0., 0.}	{0, 1, 2}
6	D	{5., 0., 2.5}	{6, 7, 8}
7	D	{5., 2., 0.}	{3, 4, 5}
8	D	{5., 2., 2.5}	{9, 10, 11}
9	D	{10., 0., 0.}	{21, 22, −1}
10	D	{10., 0., 2.}	{25, 26, −1}
11	D	{10., 2., 0.}	{23, 24, −1}
12	D	{10., 2., 2.}	{27, 28, −1}
13	EAS	{0., 0., 0.}	{12, 13, 14, 15, 16, 17, 18, 19, 20}
14	EAS	{0., 0., 0.}	{29, 30, 31, 32, 33, 34, 35, 36, 37}

Node identification switch -LP prevents post-processing of auxiliary nodes.

In[28]:=
```
SMTShowMesh["FillElements" -> False, "NodeMarks" → True, "Marks" -> "NodeNumber"]
```



Node identification switch -P prevents also selection of the auxiliary nodes by coordinates of the nodes alone.

In[29]:=
```
SMTNodeData["X" == 0 &, "NodeIndex"]
```

Out[29]=
```
{1, 2, 3, 4}
```

Auxiliary nodes can be selected if the node identification is gives as a part of the search criteria.

In[30]:=
```
SMTNodeData["ID" == "EAS" &, "NodeIndex"]
```

Out[30]=
```
{13, 14}
```

Here the index of auxiliary nodes for all elements is extracted.

In[31]:=
```
SMTElementData["Nodes"][[All, 9]]
```

Out[31]=
```
{13, 14}
```

Cantilever beam example

In[32]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["A", "ExamplesHypersolid3DB", {"E *" -> 1000., "ν *" -> .3}];
SMTAddEssentialBoundary[{ "X" == 0 &, 1 -> 0, 2 -> 0, 3 -> 0}, { "X" == 10 &, 3 -> -1}];
SMTAddMesh[
  Hexahedron[{{0, 0, 0}, {10, 0, 0}, {10, 2, 0}, {0, 2, 0},
    {0, 0, 3}, {10, 0, 2}, {10, 2, 2}, {0, 2, 3}}], "A", "H1", {15, 6, 6}];
SMTAnalysis[];
```

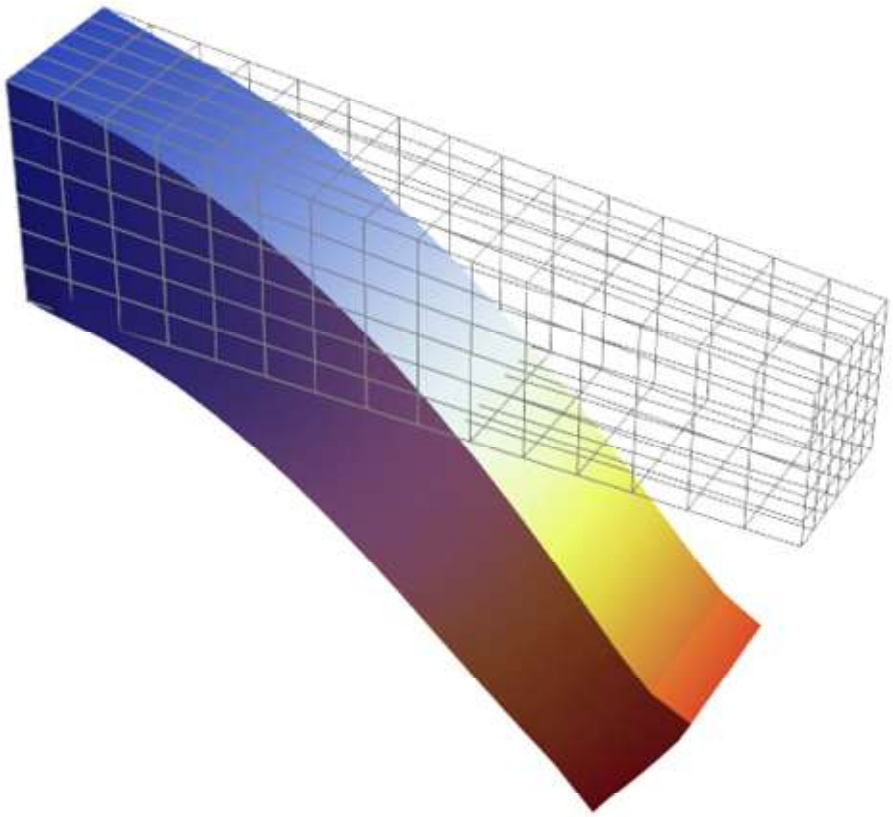
In[38]:=
```
SMTNextStep["λ" → 1];
While[
  While[
   step = SMTConvergence[10^-8, 10, {"Adaptive BC", 8, .001, 1, 5}], SMTNewtonIteration[];];
  SMTStatusReport[];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" → step[[2]]]
 ];
```

```
Step/Iter=1/5 λ/Δλ=1./1. ||Δp||/||R|| =
 1.16528×10^-13/1.19582×10^-11 Events=0 Status=0/{Convergence}

Step/Iter=2/5 λ/Δλ=2./1. ||Δp||/||R|| =
 2.43509×10^-11/2.80439×10^-10 Events=0 Status=0/{Convergence}

Step/Iter=3/5 λ/Δλ=3./1. ||Δp||/||R|| =
 3.6261×10^-11/4.24019×10^-10 Events=0 Status=0/{Convergence}

Step/Iter=4/5 λ/Δλ=4./1. ||Δp||/||R|| =
 8.29826×10^-12/1.27478×10^-10 Events=0 Status=0/{Convergence}

Step/Iter=5/5 λ/Δλ=5./1. ||Δp||/||R|| =
 2.29262×10^-12/3.84229×10^-11 Events=0 Status=0/{Convergence}
```

In[40]:=
```
SMTPostData[{"u", "v", "w", "Sxx", "Exx"}, Point[{10, 1, 1}]]
```

$$\{-1.69498,\ 5.48784\times10^{-16},\ -5.,\ 61.145,\ -0.0500509\}$$

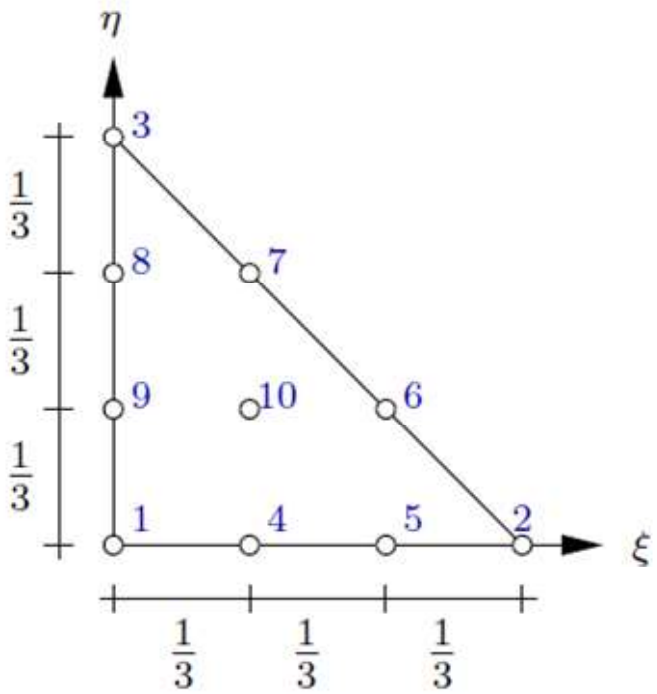
In[41]:=
```
Show[SMTShowMesh["FillElements" -> False],
 SMTShowMesh["DeformedMesh" → True, "Mesh" -> False, "Field" -> Map[Norm, SMTNodeData["at"]]]]
```



Cubic triangle, Additional nodes

Description

Generate the two-dimensional, triangular plane strain element with cubic interpolation of displacements (see also Simple 2D Solid, Finite Strain Element). Standard build-in topologies cover only liner (3 node) and quadratic (6 node) triangular elements, thus the proper template constants (see Template Constants) have to be provided by the user.



Solution

The element is based on a 3 node triangle topology (T1), enhanced by two additional nodes on a edges of the triangle and one additional node in the middle of the element.

In[42]:=
```
<< "AceGen`"
SMSInitialize["ExamplesT3", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "T1",
  "SMSNoNodes" → 10,
  "SMSAdditionalNodes" → Function[{x1, x2, x3}, {
     x1 + 1 / 3 (x2 - x1), x1 + 2 / 3 (x2 - x1),
     x2 + 1 / 3 (x3 - x2), x2 + 2 / 3 (x3 - x2),
     x3 + 1 / 3 (x1 - x3), x3 + 2 / 3 (x1 - x3),
     (x1 + x2 + x3) / 3
    }],
  "SMSSegments" → {{1, 4, 5, 2, 6, 7, 3, 8, 9}},
  "SMSSegmentsTriangulation" → {{{1, 4, 9}, {4, 5, 10}, {5, 2, 6},
     {4, 10, 9}, {5, 6, 10}, {9, 10, 8}, {10, 7, 8}, {10, 6, 7}, {8, 7, 3}}},
  "SMSReferenceNodes" → {{0, 0}, {1, 0}, {0, 1}, {1 / 3, 0}, {2 / 3, 0},
    {2 / 3, 1 / 3}, {1 / 3, 2 / 3}, {0, 2 / 3}, {0, 1 / 3}, {1 / 3, 1 / 3}},
  "SMSDefaultIntegrationCode" → 42,
  "SMSSymmetricTangent" → True,
  "SMSDomainDataNames" -> {"E -elastic modulus", "ν -poisson ratio", "t -thickness"},
  "SMSDefaultData" -> {21000, 0.3, 1}
 ];
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
XIO ⊨ SMSIO["Nodal coordinates"];
κ ⊨ 1 - ξ - η;
Nh ⊨ {κ (3 κ - 1) (3 κ - 2) / 2, ξ (3 ξ - 1) (3 ξ - 2) / 2,
    η (3 η - 1) (3 η - 2) / 2, 9 κ ξ (3 κ - 1) / 2, 9 κ ξ (3 ξ - 1) / 2, 9 η ξ (3 ξ - 1) / 2,
    9 η ξ (3 η - 1) / 2, 9 κ η (3 η - 1) / 2, 9 κ η (3 κ - 1) / 2, 27 η ξ κ};
SMSFreeze[X, Append[Nh.XIO, ζ]];
Je ⊨ SMSD[X, Ξ]; Jed ⊨ Det[Je];
uIO ⊨ SMSIO["Nodal DOFs"];
pe = Flatten[uIO]; u ⊨ Append[Nh.uIO, 0];
H ⊨ SMSD[u, X, "Dependency" → {Ξ, X, SMSInverse[Je]}];
F ⊨ IdentityMatrix[3] + H; JF ⊨ Det[F]; Ct ⊨ Transpose[F].F;
{Em, ν, tζ} ⊨ SMSIO["Domain data"];
{λ, μ} ⊨ SMSHookeToLame[Em, ν];
W ⊨ 1 / 2 λ (JF - 1) ^ 2 + μ (1 / 2 (Tr[Ct] - 3) - Log[JF]);
wgp ⊨ SMSIO["Integration weight"[Ig]];
SMSDo[
  Rg ⊨ Jed tζ SMSD[W, pe, i];
  SMSIO[wgp Rg, "Add to", "Residual"[i]];
  SMSDo[
   Kg ⊨ SMSD[Rg, pe, j];
   SMSIO[wgp Kg, "Add to", "Tangent"[i, j]];
   , {j, i, SMSNoDOFGlobal}];
  , {i, 1, SMSNoDOFGlobal}];
SMSEndDo[];
SMSWrite[];
```

File: ExamplesT3.c Size: 12939 Time: 3

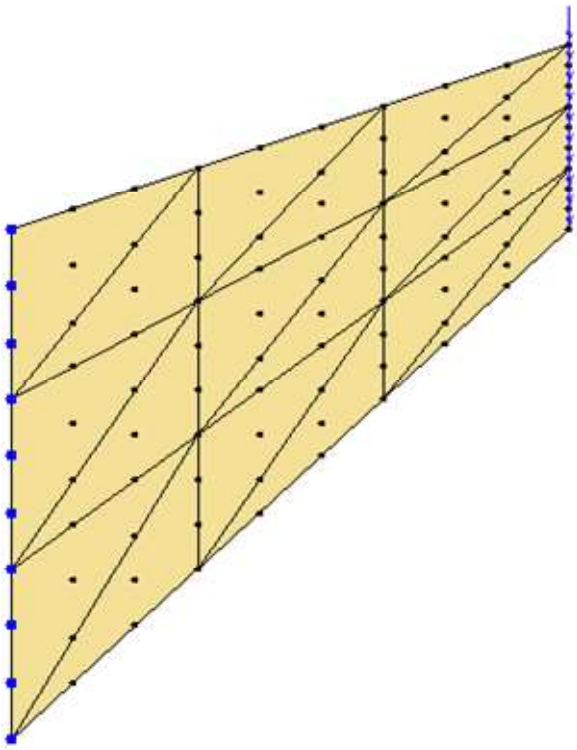
Method	SKR
No.Formulae	174
No.Leafs	2997

Cook`s membrane test

In[64]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[{"Test", "ExamplesT3", {"E *" -> 1, "ν *" -> 0}}];
SMTAddMesh[Polygon[{{0, 0}, {48, 44}, {48, 60}, {0, 44}}], "Test", "T1", {3, 3}];
SMTAddEssentialBoundary[ "X" == 0 &, 1 → 0, 2 → 0];
SMTAddNaturalBoundary[ "X" == 48 &, 2 → -.1];
SMTAnalysis[];
```

In[71]:=
```
Do[SMTNextStep["Δλ" → 0.1];
  While[SMTConvergence[10^-8, 10], SMTNewtonIteration[];];
  , {i, 1, 10}];
```

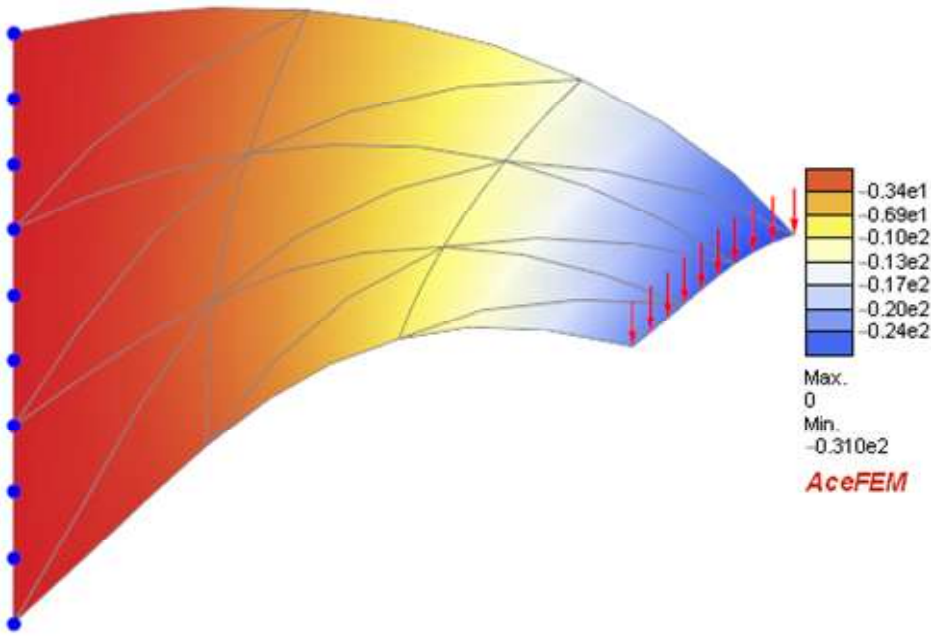
In[72]:=
```
SMTShowMesh["NodeMarks" → True, "BoundaryConditions" -> True]
```



In[73]:=
```
SMTNodeData[Point[{48, 44 + 16}], "at"]
```

Out[73]=
```
{{10.1018, -31.057}}
```

In[74]:=
```
SMTShowMesh["DeformedMesh" → True, "BoundaryConditions" → True, "Field" → SMTPostData[{"at", 2}]]
```


-0.34e1
-0.69e1
-0.10e2
-0.13e2
-0.17e2
-0.20e2
-0.24e2
Max.
0
Min.
-0.310e2
AceFEM

Gas Pressure Element - Inflating the Tyre

Description

The load such as gas pressure acts perpendicular to the actual deformed surface. Generate 2D surface traction element with the following characteristics:

⇒ surface is composed of linear segments,

⇒ traction is perpendicular to the deformed surface of the domain,

⇒ global unknowns are displacements of the nodes,

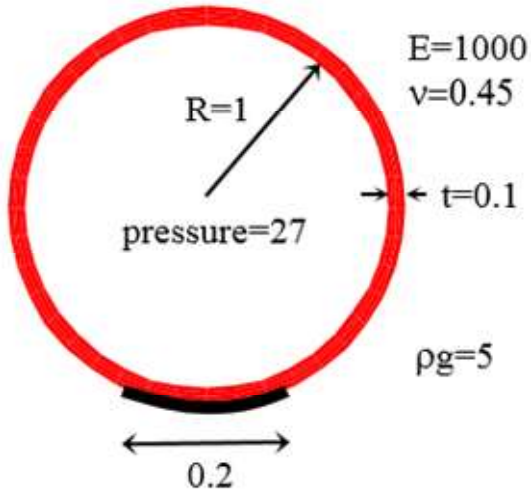
⇒ contribution of the surface pressure to the virtual work of the problem is defined by

$\int_\Gamma \mathbf{t}\delta u \, d\Gamma$

where $\delta\boldsymbol{u}$ is variation of the displacement field, $\boldsymbol{t}$ *is* prescribed surface traction and Γ is the

deformed boundary of the problem.

With the derived code and the element from the previous examples analyze the problem of inflating the tyre. Calculate the shape of the tyre when the pressure inside is 0 and when the pressure is 27.



Gas pressure element

Due to the surface pressure acting perpendicular to the deformed mesh the surface traction element contributes also to the global tangent matrix and makes the global tangent unsymmetrical.

In[75]:=
```
<< AceGen`;
SMSInitialize["ExamplesGasPressure", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "L1"
   , "SMSSymmetricTangent" → False
   , "SMSDefaultIntegrationCode" → 0
   , "SMSDomainDataNames" -> {"p -pressure", "t -thickness"}
   , "SMSDefaultData" -> {0, 1}];
SMSStandardModule["Tangent and residual"];
XIO ⊨ SMSIO["Nodal coordinates"];
uIO ⊨ SMSIO["Nodal DOFs"];
pe = Flatten[uIO];
{p, tξ} ⊨ SMSIO["Domain data"];
{xi, xj} ⊨ XIO + uIO;
L ⊨ SMSSqrt[(xj - xi).(xj - xi)];
{cosϕ, sinϕ} ⊨ (xj - xi) / L;
ξ ⊢ SMSFictive[];
u = 1/2 {(1 - ξ), (1 + ξ)}.uIO;
p = {{cosϕ, -sinϕ}, {sinϕ, cosϕ}}.{0, p};
Rel ⊨ tξ Integrate[-L/2 p.D[u, {pe}], {ξ, -1, 1}];
Ke ⊨ SMSD[Rel, pe];
SMSIO[Rel, "Add to", "Residual"];
SMSIO[Ke, "Add to", "Tangent"];
SMSWrite[];
```

```
File: ExamplesGasPressure.c  Size: 4268  Time: 1
```

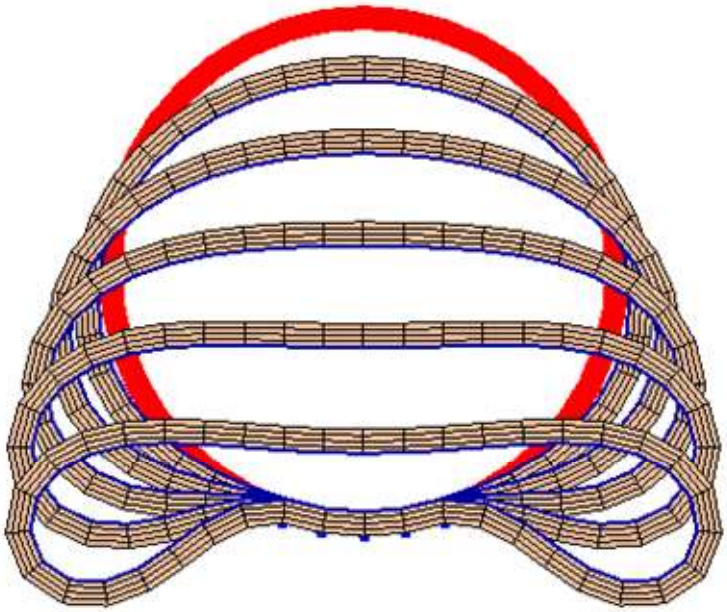
Method	SKR
No.Formulae	7
No.Leafs	269

Analysis

In[94]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[{"tyre", "ExamplesHypersolid2D", {"E *" -> 1000., "ν *" -> .3}},
  {"gas", "ExamplesGasPressure", {}}];
ne = 40;
SMTAddEssentialBoundary["Y" < -1.02 &, 1 -> 0, 2 -> 0];
SMTAddMesh[
  Raster[Table[{1.1 {Cos[ϕ], Sin[ϕ]}, {Cos[ϕ], Sin[ϕ]}}, {ϕ, 0, 2. π, 2. π / 20}] // Transpose],
  "tyre", "Q1", {ne, 4}];
SMTAddMesh[Line[Table[{Cos[ϕ], Sin[ϕ]}, {ϕ, 0, 2. π, 2. π / 20}] // Reverse], "gas", "L1", ne];
SMTAnalysis[];
undef = SMTShowMesh["BoundaryConditions" → True, "Marks" → False, "Show" -> False,
 "Mesh" → False, "FillElements" → RGBColor[1, 0, 0]];
```

The first task is to find out the shape of the empty tire. We only know that the shape of the weightless tire is circular. The shape of the empty tyre can be obtained by starting with the weightless tire and then increasing the volume force to its final value.

In[103]:=
```
SMTAddNaturalBoundary[{"DistributedOver", "tyre"}, 2 → -5];
nstep = 5;
graph = Table[
    SMTNextStep["Δλ" → 1. / nstep];
    While[SMTConvergence[10^-9, 10], SMTNewtonIteration[]];
    SMTShowMesh["DeformedMesh" → True,
     "Show" → "Window" | False, PlotRange → {{-1.5, 1.5}, {-1.4, 1.2}}]
    , {i, 1, nstep}];
Show[undef, graph]
```



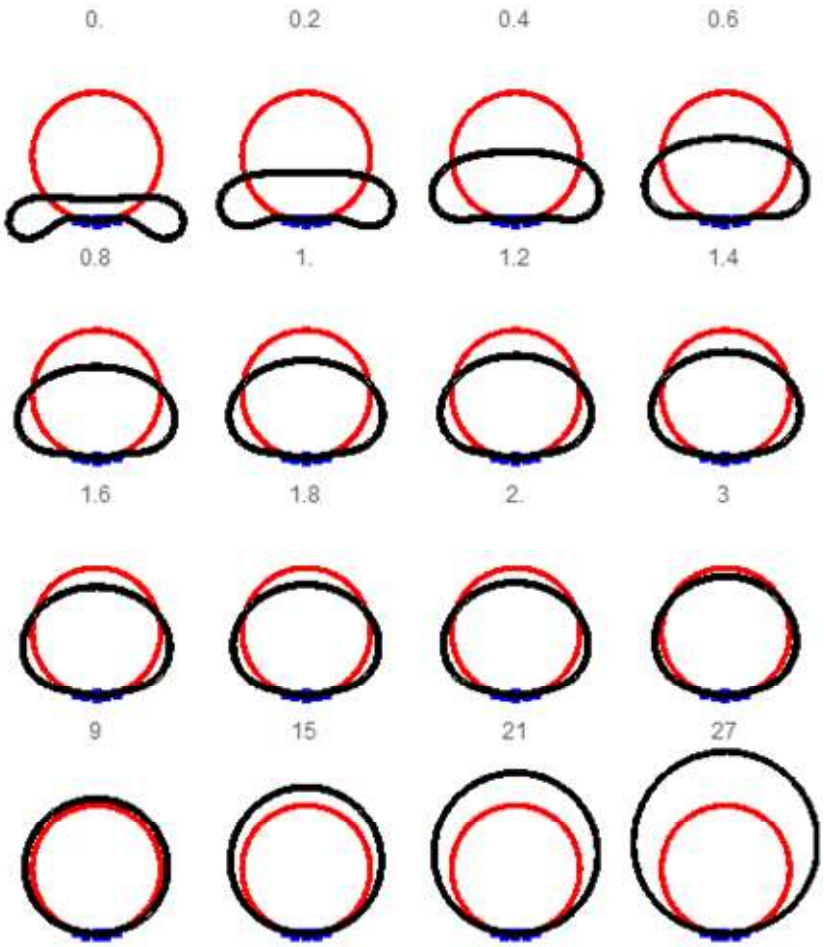
From here further the volume force will be kept constant.

In[107]:=
```
SMTNextStep[];
SMTAddNaturalBoundary[{"DistributedOver", "tyre"}, 2 → 0, "Set" → True];
```

The gas pressure is then increased in order to obtain the shape of the tyre for various gas pressure levels.

In[109]:=
```
graph = {};
Map[(SMTNextStep[];
     SMTDomainData["gas", "Data", "p *" -> #];
     While[SMTConvergence[10^-9, 10], SMTNewtonIteration[]];
     AppendTo[graph,
      SMTShowMesh["DeformedMesh" → True, "Show" → "Window" | False, "Domains" → "tyre"]];
    ) &, label = Join[Range[0, 2, .2], Range[3, 30, 6]]];
```

In[111]:=
```
GraphicsGrid[Partition[MapThread[Show[undef, #,
       DisplayFunction → Identity, PlotLabel → #2,
       PlotRange → {{-1.5, 1.5}, {-1.4, 2}}] &, {graph, label}], 4]
  , Spacings -> 0, ImageSize -> 450]
```


0.
0.2
0.4
0.6
0.8
1.
1.2
1.4
1.6
1.8
2.
3
9
15
21
27

Three Dimensional, Elasto-Plastic Element

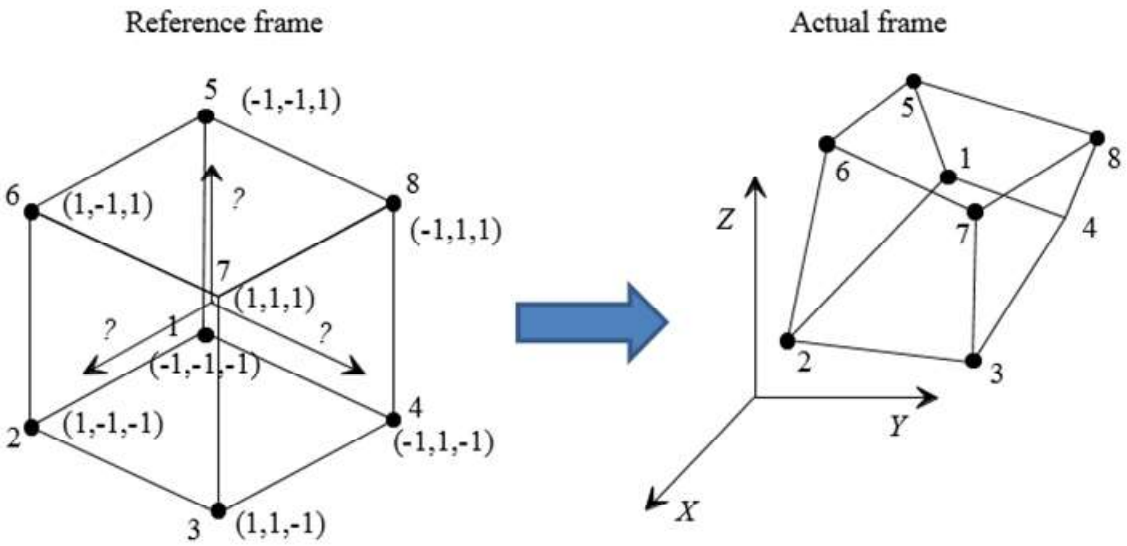
Description

Generate the three-dimensional, eight noded finite element for the analysis of the coupled, path-dependent problems in mechanics of solids. The element has the following characteristics:

⇒ hexahedral topology,

⇒ 8 nodes,

⇒ izoparametric mapping from the reference to the actual frame,



⇒ global unknowns are displacements of the nodes,

⇒ the element should take into account only small strains,

⇒ surface tractions and body forces are neglected,

⇒ the classical Hooke's law for the elastic response and an ideal elasto-plastic material law for the plastic response.

Three Dimensional, Elasto-Plastic Element

- Here the *AceGen* and the *AceFEM* interface variables are initialized.

In[20]:=
```
<< AceGen`;
SMSInitialize["ExamplesElastoPlastic3D", "Environment" -> "AceFEM"];
nhg = 7; (* 7 state variables per integration point*)
lhg = nhg + 1; (* add an additional history data for the elasto/plastic state indicator*)
ldhe = lhg es$$["id", "NoIntPoints"];
lgd = 6 ; ldae = 6 es$$["id", "NoIntPoints"];
SMSTemplate["SMSTopology" → "H1", "SMSSymmetricTangent" → True,
  "SMSNoTimeStorage" → ldhe,
  (*store the components of strain tensor for postprocessing*)
  "SMSNoElementData" → ldae,
  (* force one additional call of the SKR user subroutines
   after the convergence of the global solution has been archived in
   order to solve the evolution equations with the same accuracy*)
  "SMSPostIterationCall" → True,
  "SMSDomainDataNames" -> {"E -elastic modulus", "ν -poisson ration", "σy -yield stress"},
  "SMSDefaultData" -> {21000, 0.3, 24}
 ];
SMSStandardModule["Tangent and residual"];
```

- Element is numerically integrated by one of the built-in standard numerical integration rules (see Numerical Integration). This starts the loop over the integration points, where ξ, η, ζ are coordinates of the current integration point.

In[28]:=
```
SMSDo[Ig, 1, SMSIO["No. integration points"]];
Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
```

- Standard izoparametric interpolation of coordinates and displacements within the element domain is performed here. The $N_i = 1/8\,(1+\xi\,\xi_i)\,(1+\eta\,\eta_i)\,(1+\zeta\,\zeta_i)$ is the shape function for ith node where $\{\xi_i, \eta_i, \zeta_i\}$ are the coordinates of the ith node.

In[30]:=
```
XIO ⊨ SMSIO["Nodal coordinates"];
Ξn = {{-1, -1, -1}, {1, -1, -1}, {1, 1, -1}, {-1, 1, -1},
    {-1, -1, 1}, {1, -1, 1}, {1, 1, 1}, {-1, 1, 1}};
Nh ⊨ Table[1 / 8 (1 + ξ Ξn[[i, 1]]) (1 + η Ξn[[i, 2]]) (1 + ζ Ξn[[i, 3]]), {i, 1, 8}];
SMSFreeze[X, Nh.XIO];
Je ⊨ SMSD[X, Ξ];
Jed ⊨ Det[Je];
```

- Approximation of displacements and definition of displacement gradient $\mathbb{D}$
  $\mathbf{u} = \{u_i N_i,\ v_i N_i,\ w_i N_i\}$
  $\mathbb{D} = \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$

In[34]:=
```
uIO ⊨ SMSIO["Nodal DOFs"]; pe = Flatten[uIO];
u ⊨ Nh.uIO;
H ⊨ SMSD[u, X, "Dependency" → {Ξ, X, SMSInverse[Je]}];
```

- Definition of the small strain tensor $\epsilon = \frac{1}{2}\left(\mathbb{H} + \mathbb{H}^T\right)$.
  The *SMSFreeze* function is used here in order to enable differentiation with respect to the elements of the strain tensor later on when consistent linearization is performed ($\frac{\partial \mathbf{Q}_\Xi}{\partial \epsilon}$). The "Symmetric"->True option is necessary here to always preserve the symmetry of ϵ.

In[36]:=
```
SMSFreeze[ϵ, 1 / 2 (H + Transpose[H]), "Symmetric" → True];
```

- The total strain tensor is stored in element "Data" field for the later post-processing purposes.

In[37]:=
```
SMSIO[{ϵ[[1, 1]], ϵ[[2, 2]], ϵ[[3, 3]], ϵ[[1, 2]], ϵ[[1, 3]], ϵ[[2, 3]]},
  "Export to", "Element data"[Table[(Ig - 1) lgd + i, {i, 6}]]];
```

- The $\mathbf{h}_{g,n}$ is the vector of the state variables for the *Ig*-th integration point at the end of the previous time step. It contains the components of the plastic strain tensor $\boldsymbol{\epsilon}_n^p$ and the plastic multiplier $\lambda_{m,n}$ ($\mathbf{h}_{g,n} = \{\epsilon^p_{n_{1,1}}, \epsilon^p_{n_{2,2}}, \epsilon^p_{n_{3,3}}, \epsilon^p_{n_{1,2}}, \epsilon^p_{n_{1,3}}, \epsilon^p_{n_{2,3}}, \lambda_{m,n}\}$). The location of the $\mathbf{h}_{g,n}$ vector within the elements history variables field ed$$["hp",i] is calculated and stored into the *Ihg* variable. An additional history variable "*state*" holds information whether the material point is in elastic regime (*state=0*) or in plastic regime (*state=1000+*$\mathcal{F}$).

SMSIO["Element time dependent data n"[Ihg+ i] ≡ ed$$["hp",Ihg+i]
i-th component of the history variables at the end of the previous time step in current Gauss point

SMSIO["Element time dependent data"[Ihg+ i] ≡ ed$$["ht",Ihg+i]
i-th component of the current history variables in current Gauss point

In[38]:=
```
Ihg ⊢ SMSInteger[(Ig - 1) lhg];
dhgnIO ⊨ Table[SMSIO["Element time dependent data n"[Ihg + i]], {i, lhg}];
hgnIO ⊨ dhgnIO[[1 ;; nhg]];
state ⊨ dhgnIO[[nhg + 1]];
```

- Here the material constants are defined.

In[41]:=
```
{Em, ν, σy} ⊨ SMSIO["Domain data"];
{λ, μ} ⊨ SMSHookeToLame[Em, ν];
```

- The W$\mathcal{F}$Q[*task_*,*hgt_*] function calculates accordingly to the task required the yield function $\mathcal{F}$, the system of local equations $\mathbf{Q}_g$ or elastic strain energy W as follows:

$\boldsymbol{\epsilon}^e = \boldsymbol{\epsilon} - \boldsymbol{\epsilon}^p$

$W = W(\boldsymbol{\epsilon}^e)$

$\boldsymbol{\sigma} = \boldsymbol{\sigma}(\boldsymbol{\epsilon}^e)$

$\mathbf{s} = \boldsymbol{\sigma} - \frac{1}{3}\mathbf{I}\,\mathrm{tr}\,\boldsymbol{\sigma}$

$\mathcal{F} = \sqrt{\frac{3}{2}\mathbf{s}:\mathbf{s}} - \sigma_y$

$\mathcal{A} = \frac{\partial \mathcal{F}}{\partial \boldsymbol{\sigma}}$

$\mathcal{Z} = \boldsymbol{\epsilon}^p - \boldsymbol{\epsilon}^p_n - (\lambda_m - \lambda_{m,n})\,\mathcal{A}$

$\mathbf{Q}_g = \{\mathcal{Z}_{1,1}, \mathcal{Z}_{2,2}, \mathcal{Z}_{3,3}, \mathcal{Z}_{1,2}, \mathcal{Z}_{1,3}, \mathcal{Z}_{2,3}, \mathcal{F}\} = \mathbf{0}$

In[43]:=
```
WFQ[task_, hgt_] :=
 (ep = {{hgt[[1]], hgt[[4]], hgt[[5]]},
        {hgt[[4]], hgt[[2]], hgt[[6]]},
        {hgt[[5]], hgt[[6]], hgt[[3]]}; λm = hgt[[7]];
  SMSFreeze[ee, ϵ - ep, "Symmetric" → True];
  W ⊨ Simplify[λ / 2 Tr[ee]^2 + μ Tr[ee.ee]];
  If[task == "W", Return[]];
  SMSFreeze[σ, SMSD[W, ee, "Symmetric" → True], "Symmetric" → True];
  s = σ - 1 / 3 IdentityMatrix[3] × Tr[σ];
  F ⊨ SMSSqrt[3 / 2 Tr[s.s]] - σy;
  If[task == "F", Return[]];
  R = Simplify[SMSD[F, σ, "Symmetric" → True]];
  epn = {{hgnIO[[1]], hgnIO[[4]], hgnIO[[5]]},
         {hgnIO[[4]], hgnIO[[2]], hgnIO[[6]]},
         {hgnIO[[5]], hgnIO[[6]], hgnIO[[3]]}};
  λmn = hgnIO[[7]];
  Z = ep - epn - (λm - λmn) R;
  Qg ⊨ Append[{Z[[1, 1]], Z[[2, 2]], Z[[3, 3]], Z[[1, 2]], Z[[1, 3]], Z[[2, 3]]}, F];
  If[task == "FQ", Return[]];)
```

In[44]:=
```
WFQ["F", hgnIO];
iNR ⊨ SMSIO["Iteration"];
SMSIf[ (iNR == 1 && state == 0) || (iNR > 1 && F < 1/10^8)];
```

- This is elastic part of the elasto-plastic formulation. The state variables are set to be the same as at the end of the previous time step.

In[47]:=
```
hgIO ⊰ hgnIO;
SMSIO[Join[hgnIO, {0}], "Export to", "Element time dependent data"[Table[Ihg + i, {i, lhg}]]];
```

In[49]:=
```
SMSElse[];
```

- This is plastic part of the elasto-plastic formulation.

  Independent vector of state variables $\mathbf{h}_g^{(j)}$ is first introduced. The trial value for $\mathbf{h}_g^{(j)}$ is taken to be the one at the end of the previous time step ($\mathbf{h}_{g,n}$). The local system of equations $\mathbf{Q}_g$ is evaluated, linearized and solve using the Newton-Raphson procedure. Consistent linearization of the global system of equations requires evaluation of the implicit dependencies among the state variables and the components of the total strain tensor ($\frac{D\mathbf{h}_g}{D\boldsymbol{\epsilon}}$). Implicit dependencies are obtained by definition $\mathbf{A}_g \frac{D\mathbf{h}_g}{D\boldsymbol{\epsilon}} = -\frac{\partial \mathbf{Q}_g}{\partial \boldsymbol{\epsilon}} \Longrightarrow \frac{D\mathbf{h}_g}{D\boldsymbol{\epsilon}}$.

  The p_g = SMSVariables[$\boldsymbol{\epsilon}$] function returns the true independent variables in $\boldsymbol{\epsilon}$ with the symmetry of $\boldsymbol{\epsilon}$ correctly considered. In order to obtain $\frac{D\mathbf{h}_g}{D\boldsymbol{\epsilon}}$ it is sufficient to calculate the implicit dependencies only for true independent variables in $\boldsymbol{\epsilon}$.

In[50]:=
```
hgj = hgnIO;
pg = SMSVariables[ϵ];
SMSDo[jNR, 1, 30, 1, hgj];
WℱQ["ℱQ", hgj];
Ag = SMSD[Qg, hgj];
LU = SMSLUFactor[Ag];
Δh = SMSLUSolve[LU, -Qg];
hgj = hgj + Δh;
SMSIf[Sqrt[Δh.Δh] < SMSIO["SubIterationTolerance"],
  (*the opearator = is neceserry here because the DhgDϵ will be exported from the loop *)
  DhgDϵ = SMSLUSolve[LU, -SMSD[Qg, pg, "Constant" → hgj]];
  (*The values of the state variables are stored back to the history data of the element.*)
  SMSIO[Join[hgj, {1000 + SMSAbs[ℱ]}], "Export to",
   "Element time dependent data"[Table[Ihg + i, {i, lhg}]]];
  (*exit the sub-iterative loop when the local equations are satisfied*)
  SMSBreak[];
 ];
SMSIf[jNR == "29"
  , SMSIO[1, "Export to", "SubDivergence"]; SMSIO[2, "Export to", "ErrorStatus"];
  (*exit the sub-iterative loop if the convergence was not reached*)
  SMSBreak[];
 ];
SMSEndDo[hgj, DhgDϵ];
```

- Get new, independent state variables and derive strain potential for them again. In this case, the automatic differentiation procedure will ignore the sub-iteration loop. The type D exception is used to specify partial derivatives $\frac{\mathrm{D}\mathbf{h}_g}{\mathrm{D}\boldsymbol{\epsilon}}$ (Exceptions in Differentiation).

In[61]:=
```
hgIO = SMSFreeze[hgj, "Dependency" → {pg, DhgDϵ}];
```

In[62]:=
```
SMSEndIf[hgIO];
```

- Here the Gauss point contribution to the global residual $\mathbf{R}_g$ and the global tangent matrix $\mathbf{K}_g$ are evaluated and exported to the output parameters of the user subroutine. The strain energy function is first evaluated for the final value (valid for both elastic and plastic case) of the state variables $\mathbf{h}_g$.

In[63]:=
```
WℱQ["W", hgIO];
wgp = SMSIO["Integration weight"[Ig]];
SMSDo[
  Rg = Jed SMSD[W, pe, i, "Constant" → hgIO];
  SMSIO[wgp Rg, "Add to", "Residual"[i]];
  SMSDo[
   Kg = SMSD[Rg, pe, j];
   SMSIO[wgp Kg, "Add to", "Tangent"[i, j]];
   , {j, i, SMSNoDOFGlobal}];
  , {i, 1, SMSNoDOFGlobal}];
```

- This is the end of the numerical integration loop.

In[66]:=
```
SMSEndDo[];
```

In[67]:=
```
SMSStandardModule["Postprocessing"];
SMSDo[
  Ihg ⊨ SMSInteger[(Ig - 1) lhg];
  dhgIO ⊨ Table[SMSIO["Element time dependent data"[Ihg + i]], {i, lhg}];
  hgIO = dhgIO[[1 ;; nhg]]; state = dhgIO[[nhg + 1]]; hgnIO = hgIO;
  {Em, ν, σy} ⊨ SMSIO["Domain data"];
  {λ, μ} ⊨ SMSHookeToLame[Em, ν];
  ϵi ⊨ Table[SMSIO["Element data"[(Ig - 1) lgd + i]], {i, lhg}];
  ϵ ⊨ {{ϵi[[1]], ϵi[[4]], ϵi[[5]]},
       {ϵi[[4]], ϵi[[2]], ϵi[[6]]},
       {ϵi[[5]], ϵi[[6]], ϵi[[3]]}};
  WℱQ["ℱ", hgIO];
  SMSIO[{"Exx" → ϵ[[1, 1]], "Exy" → ϵ[[1, 2]], "Exz" → ϵ[[1, 3]], "Eyx" → ϵ[[2, 1]],
    "Eyy" → ϵ[[2, 2]], "Eyz" → ϵ[[2, 3]], "Ezx" → ϵ[[3, 1]], "Ezy" → ϵ[[3, 2]], "Ezz" → ϵ[[3, 3]]
    , "Sxx" → σ[[1, 1]], "Sxy" → σ[[1, 2]], "Sxz" → σ[[1, 3]], "Syx" → σ[[2, 1]],
    "Syy" → σ[[2, 2]], "Syz" → σ[[2, 3]], "Szx" → σ[[3, 1]], "Szy" → σ[[3, 2]], "Szz" → σ[[3, 3]]
    , "Exxp" → ϵp[[1, 1]], "Exyp" → ϵp[[1, 2]], "Exzp" → ϵp[[1, 3]], "Eyxp" → ϵp[[2, 1]],
    "Eyyp" → ϵp[[2, 2]], "Eyzp" → ϵp[[2, 3]], "Ezxp" → ϵp[[3, 1]], "Ezyp" → ϵp[[3, 2]], "Ezzp" → ϵp[[3, 3]],
    "Accumulated plastic deformation" -> λm, "State 0-elastic 1000+f -plastic" -> state
   }, "Export to", "Integration point post"[Ig]];
, {Ig, 1, SMSIO["No. integration points"]}];
uIO ⊨ SMSIO["Nodal DOFs"];
SMSIO[{"DeformedMeshX" -> uIO[[All, 1]],
   "DeformedMeshY" -> uIO[[All, 2]], "DeformedMeshZ" -> uIO[[All, 3]], "u" -> uIO[[All, 1]],
   "v" -> uIO[[All, 2]], "w" -> uIO[[All, 3]]}, "Export to", "Nodal point post"];
```

- Here the element source code is written in "ExamplesElastoPlastic3D.c" file.

In[71]:=
```
SMSWrite[];
```

File: ExamplesElastoPlastic3D.c Size: 35175 Time: 13

Method	SKR	SPP
No.Formulae	555	52
No.Leafs	9884	1321

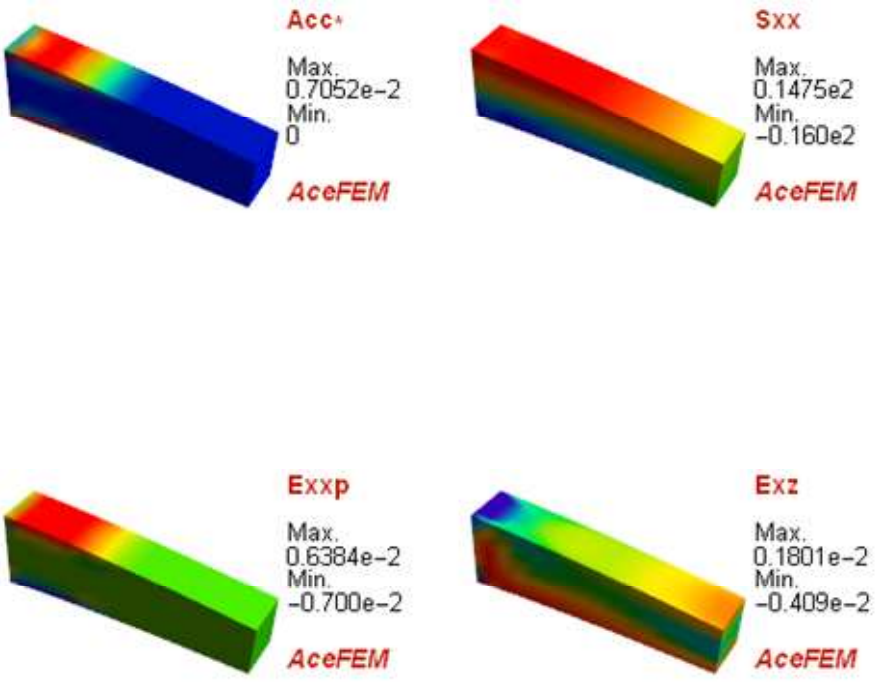
Analysis

In[89]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["A", "ExamplesElastoPlastic3D", {"E *" -> 1000., "ν *" -> .3, "σy *" -> 10.}];
SMTAddEssentialBoundary[{"X" == 0 &, 1 -> 0, 2 -> 0, 3 -> 0}, {"X" == 10 &, 3 -> -1}];
SMTAddMesh[
  Hexahedron[{{0, 0, 0}, {10, 0, 0}, {10, 2, 0}, {0, 2, 0},
    {0, 0, 3}, {10, 0, 2}, {10, 2, 2}, {0, 2, 3}}], "A", "H1", {15, 6, 6}];
SMTAnalysis[];
```

In[95]:=
```
SMTNextStep["λ" -> 0.1];
While[
  While[
   step = SMTConvergence[10^-8, 10, {"Adaptive BC", 8, .001, .1, 0.5}], SMTNewtonIteration[];];
  SMTStatusReport[];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" -> step[[2]]]
 ];
```

```
Step/Iter=1/3 λ/Δλ=0.1/0.1 ‖Δp‖/‖R‖=0./0. Events=0 Status=0/{}

Step/Iter=2/3 λ/Δλ=0.2/0.1 ‖Δp‖/‖R‖=0./0. Events=0 Status=0/{}

Step/Iter=3/3 λ/Δλ=0.3/0.1 ‖Δp‖/‖R‖=0./0. Events=0 Status=0/{}

Step/Iter=4/5 λ/Δλ=0.4/0.1 ‖Δp‖/‖R‖=0./0. Events=0 Status=0/{}

Step/Iter=5/6 λ/Δλ=0.5/0.1 ‖Δp‖/‖R‖=0./0. Events=0 Status=0/{}
```

In[97]:=
```
GraphicsGrid[ Partition[
  Table[
   SMTShowMesh["Field" -> {"Acc*", "Sxx", "Exxp", "Exz"}[[i]]
    , "DeformedMesh" → True, "Mesh" → False, "Legend" -> "MinMax"]
   , {i, 1, 4}]
  , 2], Spacings -> 0, ImageSize → {400, Automatic}]
```



Axisymmetric, finite strain elasto-plastic element

Description

Generate axisymmetric four node finite element for the analysis of the steady state problems in the mechanics of solids. The element has the following characteristics:

⇒ quadrilateral topology,

⇒ 4 node element,

⇒ izoparametric mapping from the reference to the actual frame

⇒ global unknowns are displacements of the nodes,

⇒ volumetric/deviatoric split

⇒ Taylor expansion of shape functions

⇒ J2 finite strain plasticity

⇒ isotropic hardening of the form: $\sigma_y = \sigma_{y0} + K\,\alpha + (Y_\infty - \sigma_{y0})\,(1 - \text{Exp}[-\delta\,\alpha])$

The detailed description of the steps is given in Three Dimensional, Elasto-Plastic Element .The only difference is in kinematic equations and the definition of the $\mathcal{F}\Phi\Pi$ function. The state variables in this case are the components of an inverse plastic right Cauchy strain tensor $\mathbf{C}_p^{-1}$, a plastic multiplier γ_m ($\mathbf{h_g} = \{C_{p_{11}}^{-1}, C_{p_{22}}^{-1}, C_{p_{33}}^{-1}, C_{p_{12}}^{-1}, \gamma_m\}$). The state variables are stored as history dependent real type values per each integration point of each element (SMSNoTimeStorage). Additionally to the state variables, the component of the deformation tensor ($\mathbf{F}_{1,1}$, $\mathbf{F}_{1,2}$, $\mathbf{F}_{2,1}$, $\mathbf{F}_{2,2}$, $\mathbf{F}_{3,3}$) are also stored for post-processing as arbitrary real values per element (SMSNoElementData).

HUDOBIVNIK, Blaž, KORELC, Jože. Closed-form representation of matrix functions in the formulation of nonlinear material models. Finite Elements in Analysis and Design, 2016, 111:19-32

KORELC, Jože, STUPKIEWICZ, Stanislaw. Closed-form matrix exponential and its application in finite-strain plasticity. International journal for numerical methods in engineering, ISSN 0029-5981, 2014, 98(13):960-987, ilustr., doi: 10.1002/nme.4653.

Solution

- All steps are described in detail in example Three Dimensional, Elasto-Plastic Element.

In[183]:=
```
<< AceGen`;
```

In[184]:=
```
WFQ[task_, hgt_] :=
 (Cgpi = IdentityMatrix[3] + {{hgt[[1]], hgt[[4]], 0}, {hgt[[4]], hgt[[2]], 0}, {0, 0, hgt[[3]]}};
  γ = hgt[[5]];
  SMSFreeze[be, F.Cgpi.Transpose[F], "Symmetric" -> True, "Ignore" → NumberQ];
  Jbe ⊨ Det[be];
  W = κ / 2 (1 / 2 (Jbe - 1) - 1 / 2 Log[Jbe]) + μ / 2 (Tr[Jbe^(-1/3) be] - 3);
  If[task == "W", Return[]];
  SMSFreeze[τ, Simplify[2 be.SMSD[W, be, "Symmetric" -> True, "Ignore" → NumberQ]],
   "Symmetric" -> True, "Ignore" → NumberQ];
  s = τ - 1/3 IdentityMatrix[3] × Tr[τ];
  α = √(2 / 3) γ;
  σy ⊨ √(2 / 3) (σy0 + Kf α + (Yinf - σy0) (1 - Exp[-δ α]));
  F ⊨ SMSSqrt[Total[s s, 2]] - σy;
  If[task == "F", Return[]];
  Cgpin = IdentityMatrix[3] + {{hgnIO[[1]], hgnIO[[4]], 0}, {hgnIO[[4]], hgnIO[[2]], 0}, {0, 0, hgnIO[[3]]}};
  γn = hgnIO[[5]];
  A = SMSD[F, τ, "Symmetric" -> True, "Ignore" → NumberQ];
  M ⊨ -2 (γ - γn) A;
  Z ⊨ Simplify[F.Cgpi - SMSMatrixExp[M, 1].F.Cgpin];
  Qg ⊨ {Z[[1, 1]], Z[[2, 2]], Z[[3, 3]], Z[[1, 2]], F};
  If[task == "FQ", Return[]];)
```

In[185]:=
```
<< AceGen`;
SMSInitialize["ExamplesFiniteStrain", "Environment" -> "AceFEM"];
nhg = 5; lhg = nhg + 1; ldhe = lhg es$$["id", "NoIntPoints"];
SMSTemplate["SMSTopology" → "Q1", "SMSSymmetricTangent" → True,
  "SMSNoTimeStorage" → ldhe,
  "SMSNoElementData" → 5 es$$["id", "NoIntPoints"],
  "SMSPostIterationCall" → True,
  "SMSDomainDataNames" ->
   {"E -elastic modulus", "ν -poisson ration", "σy -initial yield stress",
    "K -hardening coefficient", "σyInf -residual flow stress", "δ -saturation exponent"},
  "SMSDefaultData" -> {21000, 0.3, 24, 0, 24, 0}
 ];
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
XIO ⊨ SMSIO["Nodal coordinates"];
Nh ⊨ 1 / 4 {(1 - ξ) (1 - η), (1 + ξ) (1 - η), (1 + ξ) (1 + η), (1 - ξ) (1 + η)};
SMSFreeze[X, Append[Nh.XIO, ζ]];
Je ⊨ SMSD[X, Ξ];
Jed ⊨ Det[Je];
```

In[195]:=
```
{X0, J0d} ⊨ SMSReplaceAll[{X⟦1⟧, Jed}, ξ → 0, η → 0];
{Nx, Ny} = Table[SMSD[Nh, X⟦i⟧, "Dependency" → {Ξ, X, SMSInverse[Je]}], {i, 2}];
{NXξ, NXη, NYξ, NYη} = {SMSD[Nx, ξ], SMSD[Nx, η], SMSD[Ny, ξ], SMSD[Ny, η]};
{NXξ0, NXη0, NYξ0, NYη0} ⊨ SMSReplaceAll[{NXξ, NXη, NYξ, NYη}, ξ → 0, η → 0];
{V, V0} ⊨ Simplify[Integrate[
    SMSRestore[{ X⟦1⟧ Jed, X0 J0d}, ξ | η] // Normal // Simplify, {ξ, -1, 1}, {η, -1, 1}]];
NX0 ⊨ 1/V Simplify[Integrate[SMSRestore[Jed X⟦1⟧ Nx, ξ | η] // Normal // Simplify,
    {ξ, -1, 1}, {η, -1, 1}]];
NY0 ⊨ 1/V Simplify[Integrate[SMSRestore[Jed X⟦1⟧ Ny, ξ | η] // Normal // Simplify,
    {ξ, -1, 1}, {η, -1, 1}]];
Nϕ ⊨ 1/V Simplify[Integrate[SMSRestore[Jed Nh, ξ | η] // Normal // Simplify,
    {ξ, -1, 1}, {η, -1, 1}]];
uIO ⊨ SMSIO["Nodal DOFs"];
pe = Flatten[uIO]; {un, vn} = Transpose[uIO];
u ⊨ Append[Nh.uIO, 0];
Dv = 1/3 (NX0.un + NY0.vn + Nϕ.un) IdentityMatrix[3];
dv = (NX0 + NXξ0 ξ + NXη0 η).un + (NY0 + NYξ0 ξ + NYη0 η).vn + Nϕ.un;
Dd = {{(NX0 + NXξ0 ξ + NXη0 η).un - 1/3 dv, NY0.un, 0},
      {NX0.vn, (NY0 + NYξ0 ξ + NYη0 η).vn - 1/3 dv, 0},
      {0, 0, Nϕ.un - 1/3 dv}};
H ⊨ Simplify[Dv + Dd];
SMSFreeze[F, IdentityMatrix[3] + H, "Ignore" -> NumberQ];
SMSIO[{F⟦1, 1⟧, F⟦1, 2⟧, F⟦2, 1⟧, F⟦2, 2⟧, F⟦3, 3⟧},
  "Export to", "Element data"[Table[(Ig - 1) 5 + i, {i, 5}]]];
Ihg ⊨ SMSInteger[(Ig - 1) lhg];
dhgnIO ⊨ Table[SMSIO["Element time dependent data n"[Ihg + i]], {i, lhg}];
hgnIO ⊨ dhgnIO⟦1 ;; nhg⟧; state ⊨ dhgnIO⟦nhg + 1⟧;
{Em, ν, σy0, Kf, Yinf, δ} ⊨ SMSIO["Domain data"];
{μ, κ} ⊨ SMSHookeToBulk[Em, ν];
```

In[216]:=
```
WFQ["F", hgnIO];
iNR ⊨ SMSIO["Iteration"];
SMSIf[ (iNR == 1 && state == 0) || (iNR > 1 && F < 1/10^8)];
```

In[219]:=
```
hgIO ⊐ hgnIO;
SMSIO[Join[hgnIO, {0}], "Export to", "Element time dependent data"[Table[Ihg + i, {i, lhg}]]];
```

In[221]:=
```
SMSElse[];
```

In[222]:=
```
hgj = hgnIO;
SMSDo[jNR, 1, 30, 1, hgj];
WℱQ["ℱQ", hgj];
Ag = SMSD[Qg, hgj];
LU = SMSLUFactor[Ag];
Δh = SMSLUSolve[LU, -Qg];
hgj = hgj + Δh;
SMSIf[Sqrt[Δh.Δh] < SMSIO["SubIterationTolerance"] ,
  (*the opearator = is neceserry here because the δh∈ will be exported out from the loop *)
  DhDF = SMSLUSolve[LU, -SMSD[Qg, SMSVariables[F], "Constant" → hgj]];
  (*The values of the state variables are stored back to the history data of the element.*)
  SMSIO[Join[hgj, {1000 + SMSAbs[ℱ]}], "Export to",
   "Element time dependent data"[Table[Ihg + i, {i, lhg}]]];
  (*exit the sub-iterative loop when the local equations are satisfied*)
  SMSBreak[];
 ];
SMSIf[jNR == "29"
  , SMSIO[1, "Export to", "SubDivergence"]; SMSIO[2, "Export to", "ErrorStatus"];
  (*exit the sub-iterative loop if the convergence was not reached*)
  SMSBreak[];
 ];
SMSEndDo[hgj, DhDF];
hgIO = SMSFreeze[hgj, "Dependency" → {SMSVariables[F], DhDF}];
```

In[233]:=
```
SMSEndIf[hgIO];
```

In[234]:=
```
WℱQ["W", hgIO];
wgp = SMSIO["Integration weight"[Ig]];
SMSDo[
  Rg = 2 π V/V0 X0 J0d SMSD[W, pe, i, "Constant" → hgIO];
  SMSIO[wgp Rg, "Add to", "Residual"[i]];
  SMSDo[
   Kg = SMSD[Rg, pe, j];
   SMSIO[wgp Kg, "Add to", "Tangent"[i, j]];
   , {j, i, SMSNoDOFGlobal}];
  , {i, 1, SMSNoDOFGlobal}];
```

In[237]:=
```
SMSEndDo[];
```

In[238]:=
```
SMSStandardModule["Postprocessing"];
SMSDo[
  Ihg ⊨ SMSInteger[(Ig - 1) lhg];
  dhgIO ⊨ Table[SMSIO["Element time dependent data"[Ihg + i]], {i, lhg}];
  hgIO ⊨ dhgIO[[1 ;; nhg]]; state ⊨ dhgIO[[nhg + 1]]; hgnIO = hgIO;
  {Em, ν, σy0, Kf, Yinf, δ} ⊨ SMSIO["Domain data"];
  {μ, κ} ⊨ SMSHookeToBulk[Em, ν];
  Fi ⊨ Table[SMSIO["Element data"[(Ig - 1) 5 + i]], {i, 5}];
  F ⊨ {{Fi[[1]], Fi[[2]], 0}, {Fi[[3]], Fi[[4]], 0}, {0, 0, Fi[[5]]}};
  WFQ["F", hgIO];
  Eg ⊨ 1 / 2 (Transpose[F].F - IdentityMatrix[3]);
  σ ⊨ τ / Det[F];
  Egp = 1 / 2 (Inverse[Cgpi] - IdentityMatrix[3]);
  SMSIO[{"Exx" → Eg[[1, 1]], "Exy" → Eg[[1, 2]], "Exz" → Eg[[1, 3]], "Eyx" → Eg[[2, 1]],
    "Eyy" → Eg[[2, 2]], "Eyz" → Eg[[2, 3]], "Ezx" → Eg[[3, 1]], "Ezy" → Eg[[3, 2]], "Ezz" → Eg[[3, 3]],
    "Sxx" → σ[[1, 1]], "Sxy" → σ[[1, 2]], "Sxz" → σ[[1, 3]], "Syx" → σ[[2, 1]], "Syy" → σ[[2, 2]],
    "Syz" → σ[[2, 3]], "Szx" → σ[[3, 1]], "Szy" → σ[[3, 2]], "Szz" → σ[[3, 3]], "Exxp" → Egp[[1, 1]],
    "Exyp" → Egp[[1, 2]], "Exzp" → Egp[[1, 3]], "Eyxp" → Egp[[2, 1]], "Eyyp" → Egp[[2, 2]],
    "Eyzp" → Egp[[2, 3]], "Ezxp" → Egp[[3, 1]], "Ezyp" → Egp[[3, 2]], "Ezzp" → Egp[[3, 3]],
    "Accumulated plastic deformation" -> Sqrt[2 / 3] γ, "State 0-elastic 1000+f -plastic" -> state
   }, "Export to", "Integration point post"[Ig]];
, {Ig, 1, SMSIO["No. integration points"]}];
uIO ⊨ SMSIO["Nodal DOFs"];
SMSIO[{"DeformedMeshX" -> uIO[[All, 1]], "DeformedMeshY" -> uIO[[All, 2]],
   "u" -> uIO[[All, 1]], "v" -> uIO[[All, 2]]}, "Export to", "Nodal point post"];
```

In[242]:=
```
SMSWrite[];
```

File: ExamplesFiniteStrain.c Size: 49 937 Time: 220

Method	SKR	SPP
No.Formulae	809	59
No.Leafs	15 839	1339

Cyclic tension test, advanced post-processing, animations

In this example a method how to separate the analysis and post-processing of the results into two completely separated *Mathematica* sessions is demonstrated. However, one has to be aware that when the analysis and post-processing are done separately, one can post-process only the data that have been stored during the analysis. The data stored during analysis includes mesh and vectors of nodal values for all post-processing quantities defined in post-processing subroutines (see `Standard User Subroutines`) for all elements. The data does not include the nodal DOF and history data, thus complex post-processing has to be done in parallel with the analysis as presented in Bending of the column (path following procedure, animations, 2D solids) example. For more see `Separate Visualization`.

Analysis Session

With the use of the elements generated in example Axisymmetric, finite strain elasto-plastic element the following cyclic plasticity example is analyzed.

In[243]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain["A", "ExamplesFiniteStrain", {"E *" -> 206.9, "ν *" -> .29,
   "σy *" -> 0.45, "K *" -> 0.12924, "σyInf *" -> 0.715, "δ *" -> 16.93}];
L = 1.6; R = 0.4; R1 = 0.5; ne = 10;
ΔL = 0.16;
SMTAddMesh[Polygon[{{R, 0}, {R, 0.4}, {0, 0.4}, {0, 0}}], "A", "Q1", {2*ne, ne}];
SMTAddMesh[Polygon[{{R, 0.4}, {R, 0.8}, {0, 0.8}, {0, 0.4}}], "A", "Q1", {ne, ne}];
SMTAddMesh[Raster[{{{0.4, 0.8}, {0.41, 0.89}, {0.43, 0.98}, {0.49, 1.06}, {0.5, 1.1}},
    {{0, 0.8}, {0, 0.89}, {0, 0.98}, {0, 1.06}, {0, 1.1}}}],
  "A", "Q1", {ne, ne}, "InterpolationOrder" → 1];
SMTAddMesh[Polygon[{{R1, 1.1}, {R1, L}, {0, L}, {0, 1.1}}], "A", "Q1", {ne/2, ne}];
SMTAddEssentialBoundary[{Line[{{0, 0}, {0, L}}], 1 -> 0},
  {Line[{{0, 0}, {R, 0}}], 2 -> 0}, { Line[{{0, L}, {R1, L}}], 2 -> ΔL }];
```

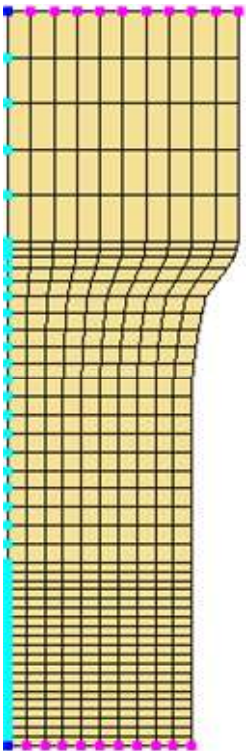
- Here the post-processing data base file name ("cyclic") is specified.

In[253]:=
```
SMTAnalysis["Output" -> "Log.txt", "DumpInputTo" → "cyclic"];
```

- With the SMTSave command definitions of arbitrary number of symbols can be stored into the post-processing data base file cyclic.idx.

In[254]:=
```
SMTSave[ΔL, L, R];
```

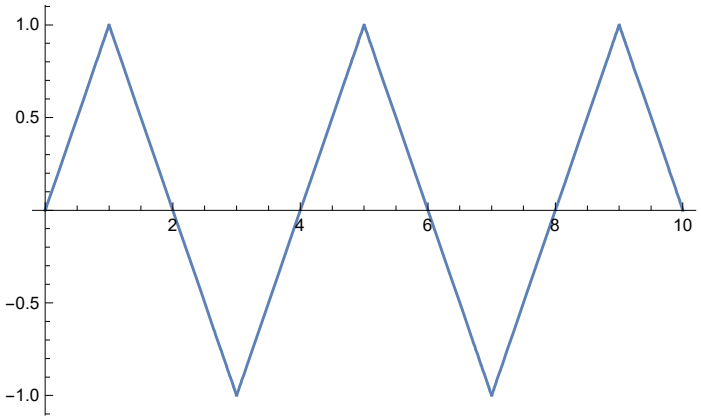
In[255]:=
```
SMTShowMesh["BoundaryConditions" → True]
```



- Here the cyclic loading simulation is performed. Function λ(t) defines the load history with λ as the load multiplier. The SMTPut command writes current values of post-processing data into the data base. Current time is used as an keyword that will be used later to retrieve information from data base. Additionally the resulting force at the top of the specimen is olso stored into data base.

In[256]:=
```
Clear[λ]; λ[t_] := If[OddQ[Floor[(t + 1) / 2]], 1, -1] (2 Floor[(t + 1) / 2] - t) ;
```

In[257]:=
```
Plot[λ[t], {t, 0, 10}]
```



In[258]:=
```
SMTNextStep["Δt" → 0.01, "λ[t]" → λ];
While[
  While[step = SMTConvergence[10^-8, 15, {"Adaptive Time", 8, .0001, 0.1, 10}],
   SMTNewtonIteration[];];
  If[Not[step[[1]]]
   , force = (Plus @@ SMTResidual["Y" == L &])[[2]];
   SMTPut[SMTRData["Time"], force, "TimeFrequency" → 0.1];
   SMTShowMesh["DeformedMesh" → True, "Field" → "Acc*", "Show" → "Window"];
  ];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δt" → step[[2]], "λ[t]" → λ]
 ];
```

In[260]:=
```
SMTSimulationReport[];
```

No. of nodes	506	Direct analysis:	{}
No. of elements	450	Total linear solver time (s)	7.512
No. of equations	944	Total linear solver time (%)	9.7399
Number of threads used/max	8/8	Total K&R time (s)	9.525
Data memory (KBytes)	415	Total K&R time (%)	12.35
Tangent matrix (KBytes)	65	Average time/iteration (s)	0.082418
Solver memory (KBytes)	975	Average linear solver time (s)	0.0080342
Total driver (KBytes)	1455	General timing:	
MMA kernel memory (KBytes)	256437	Input data phase (s)	2.4927
MMA front end memory(KBytes)	622498	Input data phase (%)	3.2319
Total memory (KBytes)	880390	Total visualization time (s)	12.167
Solver type	Pardiso	Total visualization time (%)	15.776
Matrix type	-2	Total absolute time (s)	77.126
No. of steps	128	Total driver time (s)	77.061
No. of steps back	26	CPU Mathematica time (s)	11.345
Step efficiency (%)	83.117	CPU Mathematica time (%)	14.71
Total no. of iterations	935	Profile time	2.787
Average iterations/step	6.0714		
Terminal BC multiplier (λ)	0.		
Terminal time (t)	10.		
Terminal parameter (γ)	0.		

Postprocessing Session

- Here the new, independent session starts by reading data from previously stored files.

In[261]:=
```
<< AceFEM`;
SMTRestart["cyclic"];
```

```
Visualisation session: cyclic See also: SMTPut
```

- The SMTGet[] command returns all keywords and the names of stored post-processing quantities.

In[263]:=
```
{allkeys, allpost} = SMTGet[];
allpost
allkeys // Length
```

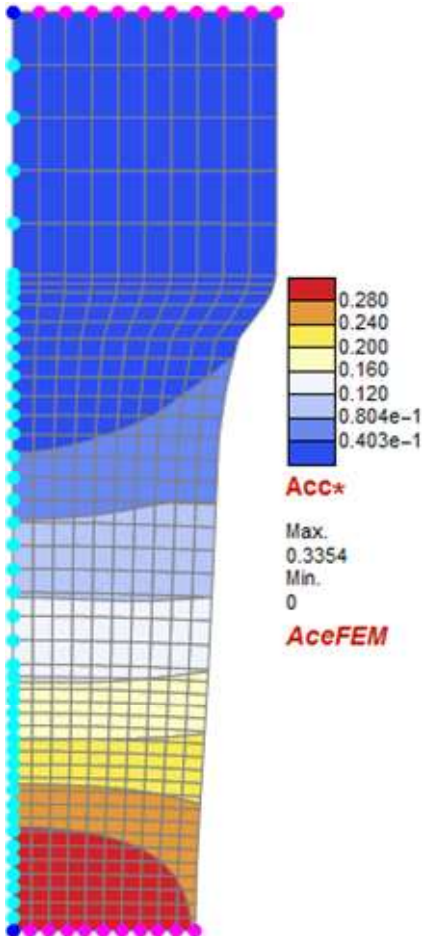
```
{Accumulated plastic deformation, DeformedMeshX, DeformedMeshY, Exx, Exxp, Exy,
 Exyp, Exz, Exzp, Eyx, Eyxp, Eyy, Eyyp, Eyz, Eyzp, Ezx, Ezxp, Ezy, Ezyp, Ezz, Ezzp,
 State 0-elastic 1000+f -plastic, Sxx, Sxy, Sxz, Syx, Syy, Syz, Szx, Szy, Szz, u, v}

95
```

- The SMTGet[key] command reads the information stored under the keyword allkeys[[10]].

In[266]:=
```
force = SMTGet[allkeys[[10]]]
allkeys[[10]]
SMTShowMesh["BoundaryConditions" → True, "DeformedMesh" → True,
 "Field" -> "Acc*", "Marks" → False, "Contour" → True]
```

```
{-0.295677}
```

```
1.08743
```



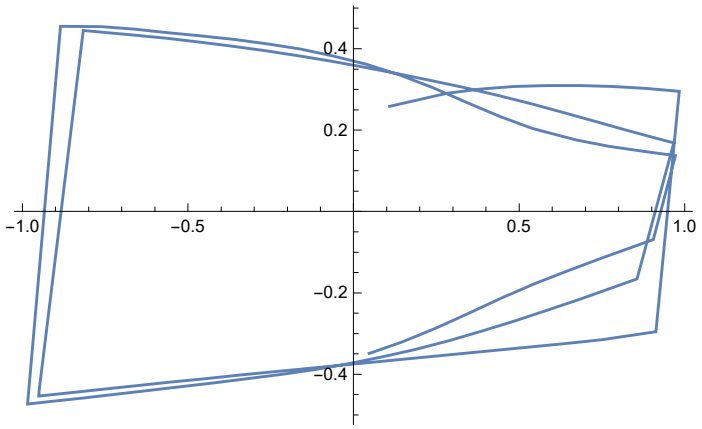
- Here all post-processing quantities are evaluated at point {R,0}. Note that the value of "R" was stored by the SMTSave command at the analysis time and restored by the SMTRestart[“cyclic”] command.

In[269]:=
```
{Range[allpost // Length], allpost, SMTPostData[allpost, Point[{R, 0}]]} // Transpose // TableForm
```

1	Accumulated plastic deformation	0.278257
2	DeformedMeshX	−0.0533813
3	DeformedMeshY	0.
4	Exx	−0.102829
5	Exxp	−0.104138
6	Exy	0.000222381
7	Exyp	0.000223274
8	Exz	0.
9	Exzp	0.
10	Eyx	0.000222381
11	Eyxp	0.000223274
12	Eyy	0.335853
13	Eyyp	0.340869
14	Eyz	0.
15	Eyzp	0.
16	Ezx	0.
17	Ezxp	0.
18	Ezy	0.
19	Ezyp	0.
20	Ezz	−0.123209
21	Ezzp	−0.124491
22	State 0−elastic 1000+f −plastic	1000.
23	Sxx	0.304881
24	Sxy	−0.00341876
25	Sxz	0.
26	Syx	−0.00341876
27	Syy	−0.439097
28	Syz	0.
29	Szx	0.
30	Szy	0.
31	Szz	0.313423
32	u	−0.0533813
33	v	0.

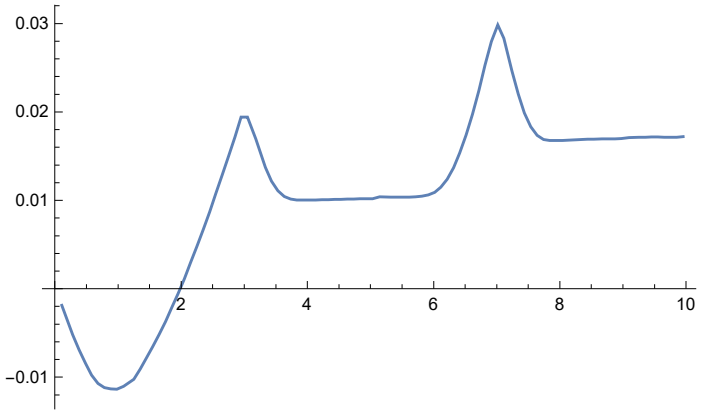
- Here the f(λ) diagram is presented.

In[270]:=
```
ListLinePlot[Map[(force = SMTGet[#][[1]];
    {SMTRData["Multiplier"], force}) &, allkeys]]
```

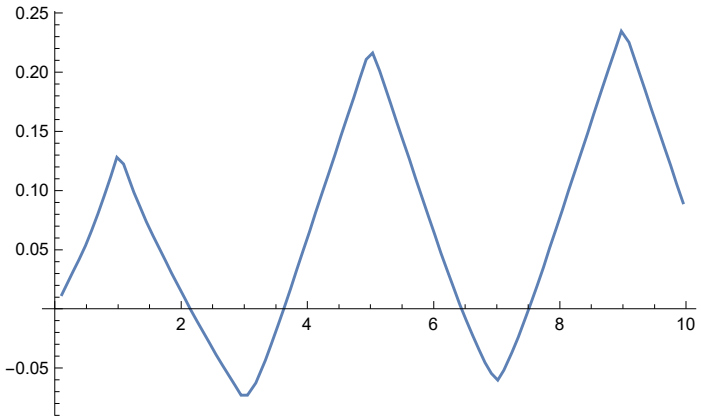


- Here the evolution of some (u, v, plastic deformation, σ_{yy}) post-processing quantities in point {R/2,L/3} is presented.

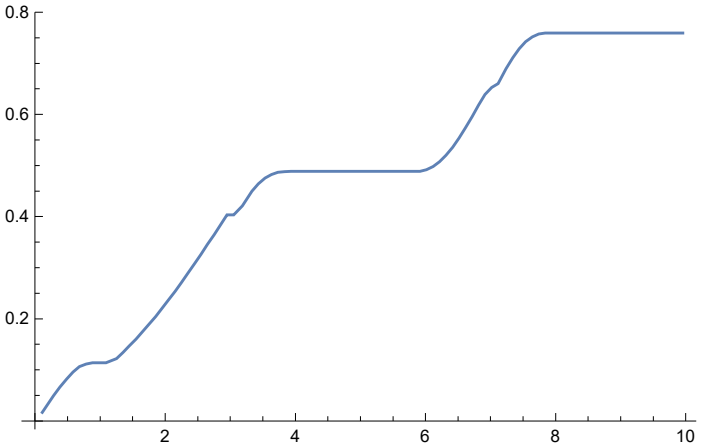
In[271]:=
```
ListLinePlot[Map[{#, SMTGet[#];
    SMTPostData["u", Point[{R/2, L/3.}]]}&, allkeys]]
```



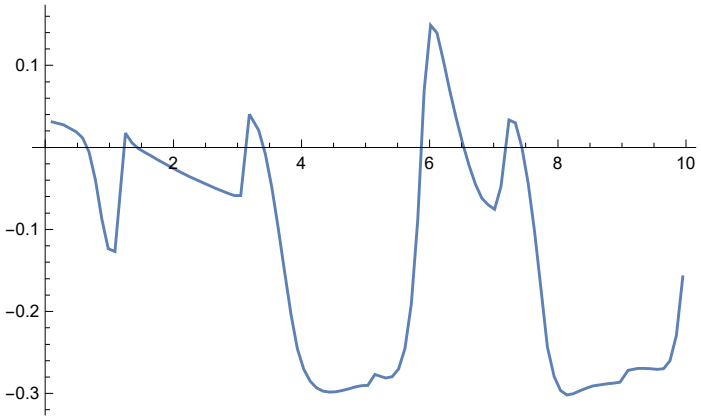
In[272]:=
```
ListLinePlot[Map[{#, SMTGet[#];
    SMTPostData["v", Point[{R/2, L/3.}]]}&, allkeys]]
```



In[273]:=
```
ListLinePlot[Map[{#, SMTGet[#];
    SMTPostData["Acc*", Point[{R/2, L/3.}]]}&, allkeys]]
```



In[274]:=
```
ListLinePlot[Map[{#, SMTGet[#];
    SMTPostData["Sxx", Point[{R / 2, L / 3.}]]} &, allkeys]]
```

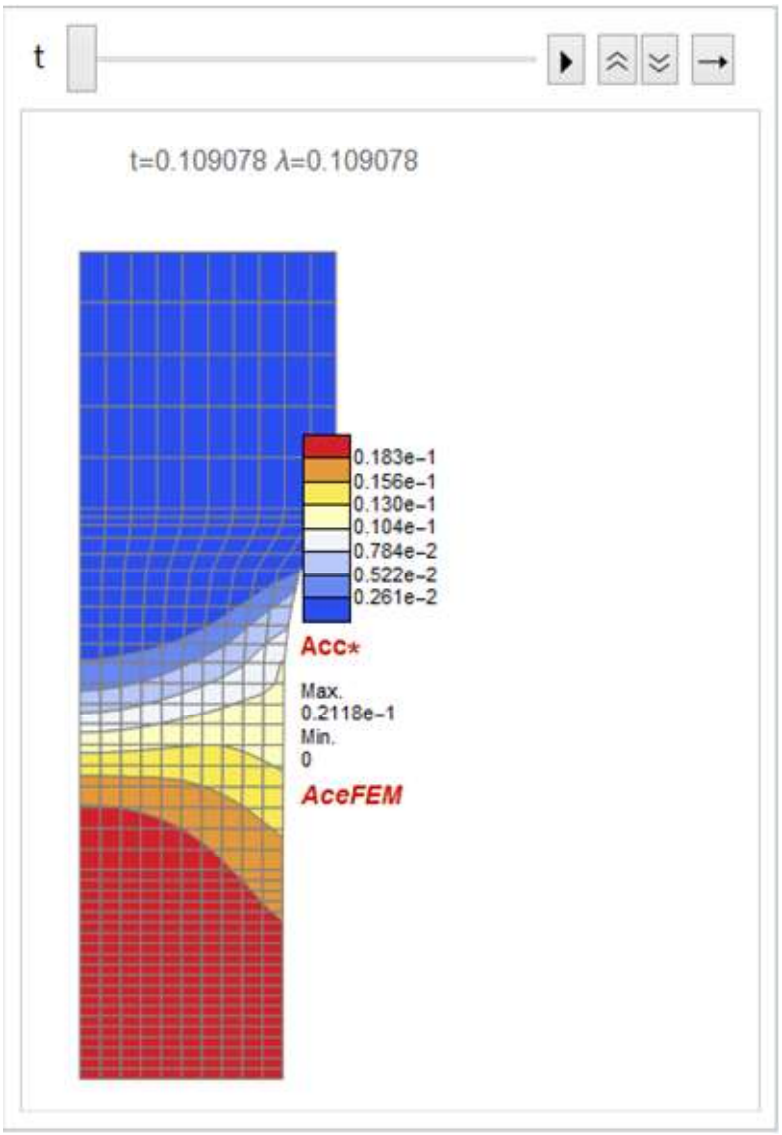


- Here the sequence of pictures is generated and stored in HDF5 data base file pldef.h5.

In[275]:=
```
Map[(SMTGet[#];
    SMTShowMesh["DeformedMesh" → True, "Field" → "Acc*", "Contour" -> True
     , PlotRange → {{-.1 R, 2 R}, {0, L + ΔL}}
     , "Label" → {"t=", SMTRData["Time"], " λ=", SMTRData["Multiplier"]}
     , "Show" → {"ExportFrames", "pldef"}];
   ) &, allkeys];
```

- The sequence of pictures is here transformed into the Mathematica type animation.

In[276]:=
```
SMTAnimationOfResponse["Animate", "pldef"]
```



- The sequence of pictures is here transformed into various video formats.

In[277]:=
```
SMTAnimationOfResponse["Export to", "gif", "pldef"]
```

pldef.gif

In[278]:=
```
SMTAnimationOfResponse["Export to", "SWF", "pldef"]
```

pldef.SWF

In[279]:=
```
SMTAnimationOfResponse["Export to", "mov", "pldef"]
```

pldef.mov

Solid, Finite Strain Element for Dynamic Analysis

See also 2 D snooker simulation.

In[1]:=
```
<< "AceGen`";
lhg = 4;
SMSInitialize["ExamplesHypersolidDynNewmark", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "Q1", "SMSSymmetricTangent" → True
  , "SMSNoTimeStorage" → lhg es$$["id", "NoIntPoints"]
  , "SMSDomainDataNames" -> {"E -elastic modulus", "ν -poisson ratio",
    "t -thickness of the element", "ρ0 -density", "β -Newmark beta", "δ -Newmark delta"}
  , "SMSDefaultData" -> {21000, 0.3, 1, 2700, 0.25, 0.5}];
```

TestExamples

In[5]:=
```
ElementDefinitions[] := (
  Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
  XIO ⊨ SMSIO["Nodal coordinates"];
  Nh ⊨ 1 / 4 {(1 - ξ) (1 - η), (1 + ξ) (1 - η), (1 + ξ) (1 + η), (1 - ξ) (1 + η)};
  SMSFreeze[X, Append[Nh.XIO, ζ]];
  Je ⊨ SMSD[X, Ξ]; Jed ⊨ Det[Je];
  uIO ⊨ SMSIO["Nodal DOFs"];
  unIO ⊨ SMSIO["Nodal DOFs n"];
  pe = Flatten[uIO]; u ⊨ Append[Nh.uIO, 0]; un ⊨ Append[Nh.unIO, 0];
  H ⊨ SMSD[u, X, "Dependency" → {Ξ, X, SMSInverse[Je]}];
  SMSFreeze[F, IdentityMatrix[3] + H, "Ignore" → NumberQ];
  JF ⊨ Det[F]; Ct ⊨ Transpose[F].F;
  {Em, ν, tζ, ρ0, α, δ} ⊨ SMSIO["Domain data"];
  Ihg ⊢ SMSInteger[(Ig - 1) lhg];
  dhgnIO ⊨ Table[SMSIO["Element time dependent data n"[Ihg + i]], {i, lhg}];
  vn ⊨ {dhgnIO[[1]], dhgnIO[[2]], 0}; an ⊨ {dhgnIO[[3]], dhgnIO[[4]], 0};
  Δt ⊨ SMSIO["TimeIncrement" ];
  v ⊨ δ / (α Δt) (u - un) + (1 - δ / α) vn + Δt (1 - δ / (2 α)) an;
  SMSFreeze[a, 1 / (α Δt^2) (u - un - Δt vn - Δt^2 (1 / 2 - α) an), "Ignore" → NumberQ];
  {λ, μ} ⊨ SMSHookeToLame[Em, ν];
  W ⊨ 1/2 λ (JF - 1)^2 + μ (1/2 (Tr[Ct] - 3) - Log[JF]);
  T ⊨ ρ0 u.a;
)
```

In[6]:=
```
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
SMSIf[JF ≤ 10^-9, SMSIO[2, "Export to", "ErrorStatus"]];
SMSExport[{v[[1]], v[[2]], a[[1]], a[[2]]}, Table[ed$$["ht", Ihg + i], {i, 4}]];
wgp ⊨ SMSIO["Integration weight"[Ig]];
SMSDo[
  Rg ⊨ Jed tζ SMSD[W + T, pe, i, "Constant" → SMSVariables[a]];
  SMSIO[wgp Rg, "Add to", "Residual"[i]];
  SMSDo[
   Kg ⊨ SMSD[Rg, pe, j];
   SMSIO[wgp Kg, "Add to", "Tangent"[i, j]];
   , {j, i, 8}];
  , {i, 1, 8}];
SMSEndDo[];
```

In[14]:=
```
SMSStandardModule["Postprocessing"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
SMSNPostNames = {"DeformedMeshX", "DeformedMeshY", "u", "v"};
SMSExport[Table[{uIO[[i, 1]], uIO[[i, 2]], uIO[[i, 1]], uIO[[i, 2]]}, {i, SMSNoNodes}], npost$$];
Eg := 1 / 2 (Ct - IdentityMatrix[3]);
σ := (1 / JF) * SMSD[W, F, "Ignore" → NumberQ] . Transpose[F];
SMSGPostNames = {"Exx", "Eyy", "Exy", "Sxx", "Syy", "Sxy", "Szz"};
SMSExport[Join[Extract[Eg, {{1, 1}, {2, 2}, {1, 2}}],
   Extract[σ, {{1, 1}, {2, 2}, {1, 2}, {3, 3}}]], gpost$$[Ig, #1] &];
SMSEndDo[];
```

In[24]:=
```
SMSWrite[];
```

File: ExamplesHypersolidDynNewmark.c **Size:** 13 324 **Time:** 5

Method	SKR	SPP
No.Formulae	138	74
No.Leafs	2136	1153

Elements that Call User External Subroutines

The methods for definition and use of user defined external functions within the automatically generated codes is described in detail in section User Defined Functions.

In[25]:=
```
<< AceGen`;
```

This will create a C code file "Energy.c" with the following contents

In[93]:=
```
Export["Energy.c",
 "void Energy(double *ICp, double *IICp,double
   *IIICp, double c[5], double *W,double dW[3], double ddW[3][3])
{
    double I1,I3,C1,C2,C3;
    int i,j;
    I1=*ICp;I3=*IIICp;C1=c[0];C2=c[1];
    *W=(C2*(-3 + I1))/2. + (C1*(-1 + I3 - log(I3)))/4. - (C2*log(I3))/2.;
    dW[0]=C2/2.;
    dW[1]=0.;
    dW[2]=(C1*(1 - 1/I3))/4. - C2/(2.*I3);
    for(i=0;i<3;i++){for(j=0;j<3;j++)ddW[i][j]=0.;};
    ddW[2][2]=C1/(4.*I3*I3) + C2/(2.*I3*I3);
}", "Text"]
```

Energy.c

and the C header file "Energy.h" with the following contents

In[94]:=
```
Export["Energy.h",
 "void Energy(double *ICp, double *IICp,double *IIICp,
   double c[5], double *W,double dW[3], double ddW[3][3]);", "Text"]
```

Energy.h

Subroutine Energy calculates the strain energy W(IC,IIC,IIIC) where IC and IIC and IIIC are invariants of the right Cauchy-Green tensor **C** and first and second derivative of the strain energy with respect to the input parameters IC and IIC and IIIC. 2D, plain strain, izoparameteric, quadrilateral element is generated (for details see Simple 2D Solid, Finite Strain Element).

Two solution will be presented:

Solution 1: Generated element C code file and user supplied "Energy.c" are compiled separately and then linked together to produce elements dll file

In[95]:=
```
<< AceGen`;
SMSInitialize["ExamplesUserSub1", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "Q1", "SMSSymmetricTangent" → True
  , "SMSDomainDataNames" -> {"c1 -constant 1", "c2 -constant 2", "c3 -constant 3",
    "c4 -constant 4", "c5 -constant 5"}
  , "SMSDefaultData" -> {600, 300, 100, 50, 10}
 ];
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
XIO ⊨ SMSIO["Nodal coordinates"];
Nh ⊨ 1 / 4 {(1 - ξ) (1 - η), (1 + ξ) (1 - η), (1 + ξ) (1 + η), (1 - ξ) (1 + η)};
SMSFreeze[X, Append[Nh.XIO, ζ]];
Je ⊨ SMSD[X, Ξ]; Jed ⊨ Det[Je];
uIO ⊨ SMSIO["Nodal DOFs"];
pe = Flatten[uIO]; u ⊨ Append[Nh.uIO, 0];
H ⊨ SMSD[u, X, "Dependency" → {Ξ, X, SMSInverse[Je]}];
F ⊨ IdentityMatrix[3] + H; JF ⊨ Det[F]; Ct ⊨ Transpose[F].F;
SMSFreeze[{IC, IIC, IIIC}, {Tr[Ct], 0, Det[Ct]}];
𝔠 ⊨ SMSIO["Domain data"];
WCall ⊨ SMSCall["Energy", IC, IIC, IIIC, 𝔠,
   Real[W$$], Real[dW$$[3]], Real[ddW$$[3, 3]], "System" → False];
δδW ⊢ SMSReal[Array[ddW$$[#1, #2] &, {3, 3}], "Subordinate" → WCall];
δW ⊢
  SMSReal[Array[dW$$[#1] &, {3}], "Subordinate" → WCall, "Dependency" → {{IC, IIC, IIIC}, δδW}];
W ⊢ SMSReal[W$$, "Subordinate" → WCall, "Dependency" → Transpose[{{IC, IIC, IIIC}, δW}]];
wgp ⊨ SMSIO["Integration weight"[Ig]];
Rg ⊨ Jed SMSD[W, pe];
SMSIO[wgp Rg, "Add to", "Residual"];
Kg ⊨ SMSD[Rg, pe];
SMSIO[wgp Kg, "Add to", "Tangent"];
SMSEndDo[];
SMSWrite["IncludeHeaders" → {"Energy.h"}];
```

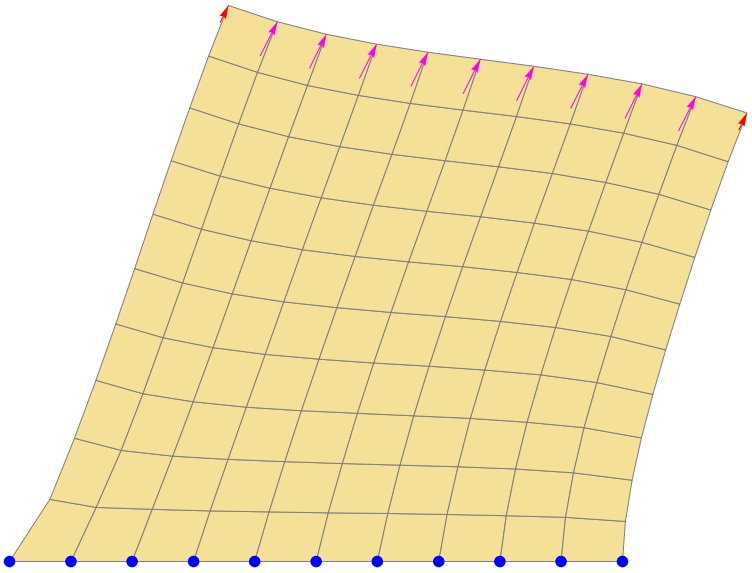
```
File: ExamplesUserSub1.c  Size: 14478  Time: 3
```

Method	SKR
No.Formulae	214
No.Leafs	4639

In[122]:=
```
SMTMakeDll["ExamplesUserSub1", "AdditionalSourceFiles" → {"Energy.c"}];
```

In[123]:=
```
<< AceFEM`;
```

In[124]:=
```
SMTInputData[];
SMTAddDomain[{"1", "ExamplesUserSub1", {}}];
SMTAddMesh[Polygon[{{0, 0}, {3, 0}, {3, 2}, {0, 2}}], "1", "Q1", {10, 10}];
SMTAddNaturalBoundary[Line[{{0, 2}, {3, 2}}] , 1 -> Line[{1}], 2 -> Line[{2}]];
SMTAddEssentialBoundary[Line[{{0, 0}, {3, 0}}], 1 -> 0, 2 -> 0];
SMTAnalysis[];
SMTNextStep["λ" -> 100];
While[SMTConvergence[10^-12, 10], SMTNewtonIteration[]];
SMTShowMesh["BoundaryConditions" -> True, "DeformedMesh" -> True]
```



Solution 2: User supplied source code file is included into the generated source code

In[66]:=

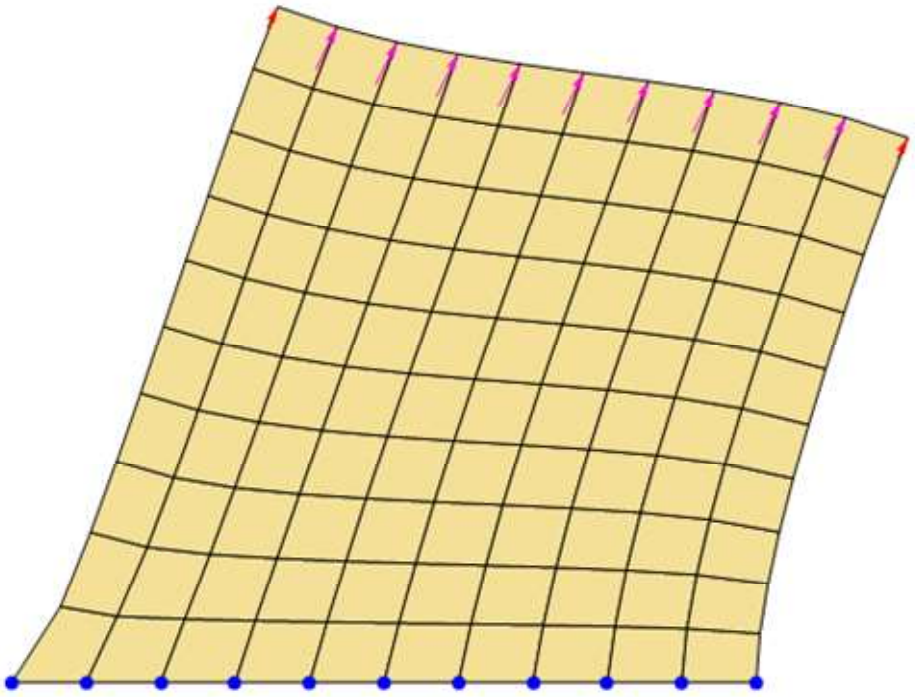
In[67]:=
```
<< AceGen`;
SMSInitialize["ExamplesUserSub2", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "Q1", "SMSSymmetricTangent" → True
  , "SMSDomainDataNames" -> {"c1 -constant 1", "c2 -constant 2", "c3 -constant 3",
    "c4 -constant 4", "c5 -constant 5"}
  , "SMSDefaultData" -> {600, 300, 100, 50, 10}
 ];
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
XIO ⊨ SMSIO["Nodal coordinates"];
Nh ⊨ 1 / 4 {(1 - ξ) (1 - η), (1 + ξ) (1 - η), (1 + ξ) (1 + η), (1 - ξ) (1 + η)};
SMSFreeze[X, Append[Nh.XIO, ζ]];
Je ⊨ SMSD[X, Ξ]; Jed ⊨ Det[Je];
uIO ⊨ SMSIO["Nodal DOFs"];
pe = Flatten[uIO]; u ⊨ Append[Nh.uIO, 0];
H ⊨ SMSD[u, X, "Dependency" → {Ξ, X, SMSInverse[Je]}];
F ⊨ IdentityMatrix[3] + H; JF ⊨ Det[F]; Ct ⊨ Transpose[F].F;
SMSFreeze[{IC, IIC, IIIC}, {Tr[Ct], 0, Det[Ct]}];
𝔠 ⊨ SMSIO["Domain data"];
WCall ⊢ SMSCall["Energy", IC, IIC, IIIC, 𝔠,
   Real[W$$], Real[dW$$[3]], Real[ddW$$[3, 3]], "System" → False];
δδW ⊢ SMSReal[Array[ddW$$[#1, #2] &, {3, 3}], "Subordinate" → WCall];
δW ⊢
  SMSReal[Array[dW$$[#1] &, {3}], "Subordinate" → WCall, "Dependency" → {{IC, IIC, IIIC}, δδW}];
W ⊢ SMSReal[W$$, "Subordinate" → WCall, "Dependency" → Transpose[{{IC, IIC, IIIC}, δW}]];
wgp ⊨ SMSIO["Integration weight"[Ig]];
Rg ⊨ Jed SMSD[W, pe];
SMSIO[wgp Rg, "Add to", "Residual"];
Kg ⊨ SMSD[Rg, pe];
SMSIO[wgp Kg, "Add to", "Tangent"];
SMSEndDo[];
SMSWrite["IncludeHeaders" → {"Energy.h"}, "Splice" -> {"Energy.c"}];
```

```
File: ExamplesUserSub2.c  Size: 14927  Time: 5
```

Method	SKR
No.Formulae	214
No.Leafs	4639

In[94]:=
```
<< AceFEM`;
```

In[95]:=
```
SMTInputData[];
SMTAddDomain[{"1", "ExamplesUserSub2", {}}];
SMTAddMesh[Polygon[{{0, 0}, {3, 0}, {3, 2}, {0, 2}}], "1", "Q1", {10, 10}];
SMTAddNaturalBoundary[Line[{{0, 2}, {3, 2}}] , 1 -> Line[{1}], 2 -> Line[{2}]];
SMTAddEssentialBoundary[Line[{{0, 0}, {3, 0}}], 1 -> 0, 2 -> 0];
SMTAnalysis[];
SMTNextStep["λ" -> 100];
While[SMTConvergence[10^-12, 10], SMTNewtonIteration[]];
SMTShowMesh["BoundaryConditions" -> True, "DeformedMesh" -> True]
```



CHAPTER 6

Advanced Applications

Sensitivity Analysis

Contents

- General Backward Mode Sensitivity Analysis Example
  - Description of backward mode example
  - First order backward mode sensitivity analysis of integral of Mises stress
  - Second order backward mode sensitivity analysis of integral of Mises stress
  - First order backward mode sensitivity analysis of L2 norm of displacement vector

Introduction to Sensitivity Analysis

The aim of the sensitivity analysis is to calculate derivatives of an arbitrary response functional with respect to chosen parameters. The response functional can depend on arbitrary analysis model inputs (material constants, load intensity and distribution, shape parameters, etc.) as well as on arbitrary intermediate or final results of the analysis (solution vectors, derived quantities such as stress tensor, integrated quantities such as damage or ductility factors, etc.). The complete automation of the sensitivity analysis is thus only possible if the automatic differentiation is applied on the complete simulation code. However, a large variety of sensitivity problems, important for practical applications, can still be solved following the classical finite element procedure where all problem dependent quantities are evaluated at the individual element level while the global level procedures remain largely problem independent. The automation of sensitivity analysis enables efficient evaluation of sensitivities that are exact except for the round off errors.

For the sensitivity problem, we define the residuals, vectors of unknowns and response functional as a function of a vector of design parameters **Φ**. The solution algorithm of the sensitivity problem can then be obtained from the primal problem by differentiating the response functional and the residuals with respect to design parameters.

Sensitivity analysis is closely related to automatic differentiation. Similar to automatic differentiation one can perform sensitivity analysis in forward mode (also called direct differentiation approach) or in backward mode (also called adjoint sensitivity approach). Forward (direct differentiation) mode is appropriate for the problem with relatively small number of sensitivity parameters and an arbitrary number of functionals, while backward (adjoint) mode is more efficient for the problems with small number of functionals and an arbitrary number of sensitivity parameters.

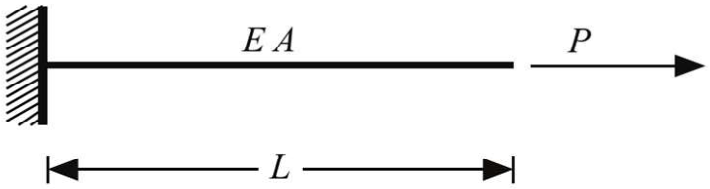
command	description
SMTSensitivityProblem[*options*]	Command that is used to define a set of sensitivity parameters **Φ** and the relation between the sensitivity parameters and finite elements (so called design and functional velocity fields (VF)).
SMTForwardSensitivity[]	calls sensitivity related user subroutines and calculates the solution of sensitivity problem with forward mode sensitivity analysis
SMTBackwardSensitivity[]	calls sensitivity related user subroutines and calculates the solution of sensitivity problem with backward mode sensitivity analysis
SMTSetVelocityFields[{ *VF_ID*–> {*nodeSelector*, *field_definition*}, …}]	Design or functional velocity fields are defined by the nodal values in selected nodes or a function applied on the selected nodes.
SMTPostData[{"st", *dof*, *sID*}, Point[{X,Y,Z}]	in forward mode returns first order sensitivity of *dof* –th component of nodal unknowns with respect to sensitivity parameter *sID* in point (X,Y,Z).
SMTPostData[{"st", *dof*,{ sID_i, sID_j}}, Point[{X,Y,Z}]	in forward mode returns second order sensitivity of *dof* – th component of nodal unknowns with respect to sensitivity parameters sID_i, sID_j in point (X,Y,Z)

The most essential AceFEM commands needed to perform sensitivity analysis.

Sensitivity analysis introductory example

Description of introductory example

The sensitivity of the displacement field **u** = {u, v} with respect to the length L is analyzed in this simple, stretching of the beam, example. The corresponding 2D, plane stress finite element "ExamplesSEPSQ1DFLEQ1DHooke" has been prepared in advance.



In[21]:=
```
<< AceFEM`;
SMTInputData[];
L = 10; H = 1; nL = 10; nH = 5; P = 1.; Em = 1000; t = 1;
SMTAddDomain[{"Ω", "ExamplesSEPSQ1DFLEQ1DHooke", {"E *" → Em, "ν *" -> 0, "t *" → t}}];
SMTAddEssentialBoundary[Line[{{0, 0}, {0, H}}], 1 -> 0, 2 → 0];
SMTAddNaturalBoundary[Line[{{L, 0}, {L, H}}], 1 → Line[{P / H}]];
SMTAddMesh[Polygon[{{0, 0}, {L, 0}, {L, H}, {0, H}}], "Ω", "Q1", {nL, nH}];
```

Definition of forward mode problem

Command that is used to define the vector of sensitivity parameters **Φ** and the relation between the input data of the finite element codes and sensitivity parameters are `SMTSensitivityProblem` and `SMTSetVelocityFields`.

In[28]:=
```
SMTSensitivityProblem[
  "Mode" → "Forward"
  , "FirstOrderVelocityFields" → {
    (* coordinate X is the parameter in finite element code that depends on
     sensitivity parameter L, therefore shape velocity field has to be defined *)
    {"L", {"S", "∂X/∂L", All -> (*X*)1}}
   }
 ];
```

In[29]:=
```
SMTAnalysis[];
```

- This sets shape velocity field (sensitivity parameter has influence on node coordinates) with respect to L: $\frac{\partial X}{\partial \phi_i} = \frac{\partial X(L)}{\partial L} = \frac{X}{L}$ (since X is linearly dependent on L)

In[30]:=
```
SMTSetVelocityFields[{"∂X/∂L" → {All, Function[{n, X, Y}, X / L]}}];
```

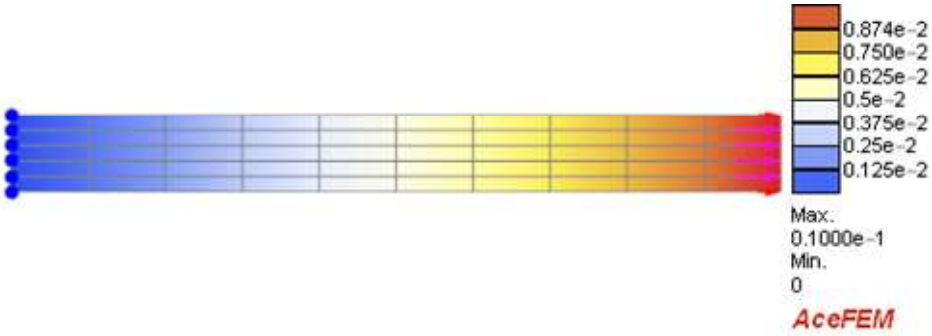
Primal analysis

- Primal analysis has to be completed before sensitivity analysis.

In[31]:=
```
SMTNextStep["Δλ" → 1];
While[SMTConvergence[10^-9, 10], SMTNewtonIteration[]];
SMTStatusReport[];
```

```
Step/Iter=1/2 λ/Δλ=1./1. ||Δp|| / ||R|| =
 7.73591×10^-15/1.8754×10^-15 Events=0 Status=0/{Convergence}
```

In[34]:=
```
SMTShowMesh["BoundaryConditions" → True, "DeformedMesh" → True, "Field" → SMTPostData[{"at", 1}]]
```



- Displacement u in point $\left(L, \frac{H}{2}\right)$.

In[35]:=
```
SMTPostData[{"at", 1}, Point[{L, H/2}]]
```

Out[35]=
```
0.01
```

- Displacement u in point $\left(L, \frac{H}{2}\right)$ accordingly to the beam theory.

In[36]:=
```
P L
---
Em t H
```

Out[36]=
```
0.01
```

Sensitivity analysis

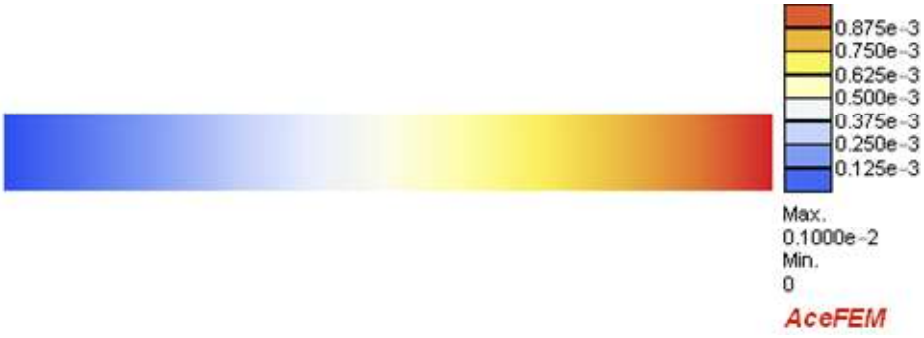
- This performs the sensitivity analysis with respect to all parameters in forward mode (`SMTForwardSensitivity`).

In[37]:=
```
SMTForwardSensitivity[];
```

Visualization of the Results of Sensitivity Analysis

- This plots sensitivity of the first nodal unknown (e.g. u) with respect to sensitivity parameter "L".

In[38]:=
```
SMTShowMesh["DeformedMesh" → True, "Mesh" → False, "Field" → SMTPostData[{"st", 1, "L"}]]
```



- Here the value of $\frac{\partial u}{\partial L}$ is taken in point $\left(L, \frac{H}{2}\right)$.

In[39]:=
```
SMTPostData[{"st", 1, "L"}, Point[{L, H/2}]]
```

Out[39]=
```
0.001
```

- Here the value of $\frac{\partial u}{\partial L}$ is calculated in point $\left(L, \frac{H}{2}\right)$ accordingly to the beam theory.

In[40]:=
```
P
---
Em t H
```

Out[40]=
```
0.001
```

General organisation of sensitivity analysis

data	description
idata$$["SensitivityScheme"]	general sensitivity analysis scheme 1 – first order forward mode sensitivity analysis 2 – first order forward mode sensitivity, second order forward mode sensitivity 3 – first order backward mode sensitivity analysis 4 – first order forward mode sensitivity, second order backward mode sensitivity
idata$$["Task"]	Value coresponds to the call of user subroutine for: 6 – global part of first order forward mode sensitivity 7 – local part of first order forward mode sensitivity 8 – global part of second order forward mode sensitivity 9 – local part of second order forward mode sensitivity 10 – first order backward mode sensitivity 11 – second order backward mode sensitivity
idata$$["NoSensParameters"]	number of sensitivity parameters of the problem (**Φ**)

Definition of relation between sensitivity parameters and sensitivity dependent finite element data

The relation between the finite element input data parameters and the actual sensitivity parameters on which functional depends is an essential part of sensitivity analysis. Forward mode is implemented in a way that this relation can be defined through design velocity fields by SMTSetVelocityFields command. Since backward method is generally preferred when the problem has high number of sensitivity parameters, the required memory to store all the velocity fields (in way that it is implemented for forward sensitivity analysis) can become prohibitively large. Therefore, the implementation of backward method was made in way that it minimizes required memory for storage of input data about sensitivity parameters. Velocity fields that relate sensitivity parameters with finite element input data are not given as input data with SMTSetVelocityFields, but the relation between proper parameters should be defined and calculated directly in finite elements!

Thus the backward mode sensitivity analysis code is problem (functional) dependent, while the forward mode sensitivity code is in general problem (functional) independent.

Organisation of derivatives

Currently AceFEM supports first and second order sensitivity analysis. A set of first order derivatives is

gradient $\nabla = \left[\frac{\partial}{\phi_1}, \frac{\partial}{\phi_2}, \ldots, \frac{\partial}{\phi_{ns}}\right]$

Second order derivatives matrix is always symmetric, thus only total of $\frac{n\phi\,(n\phi+1)}{2}$ second order derivatives is stored row by row leading to

ϕ_{11}	ϕ_{12}	ϕ_{13}	…	$\phi_{1\,n\phi}$
	ϕ_{22}	ϕ_{23}	…	$\phi_{2\,n\phi}$
		ϕ_{33}	…	$\phi_{3\,n\phi}$
			…	…
				$\phi_{n\phi\,n\phi}$

with vectorized Hessian defined by $\nabla\nabla = \left[\frac{\partial^2}{\phi_1\,\phi_{n\phi}}, \frac{\partial^2}{\phi_2\,\phi_2}, \ldots, \frac{\partial^2}{\phi_2\,\phi_{n\phi}}, \ldots, \frac{\partial^2}{\phi_{n\phi}\,\phi_{n\phi}}\right]$.

Let i_s be an index of sensitivity parameter ϕ_{i_s} and j_s be an index of sensitivity parameter ϕ_{j_s} then

$dIndex\,(i_s, j_s) = (i_s - 1)\,n_\phi + (i_s - 1)\,i_s/2 + j_s$ is a position of derivative $\frac{\partial^2}{\partial\phi_{i_s}\,\partial\phi_{j_s}}$ within $\nabla\nabla$ and $n_\phi + dIndex\,(i_s, j_s)$ is a position of derivative $\frac{\partial^2}{\partial\phi_{i_s}\,\partial\phi_{j_s}}$ within the $\nabla \cup \nabla\nabla$, a set of all derivatives of an arbitrary functions composed of all first order derivatives and symmetric part of second order derivatives.

Forward Mode Implementation Notes

General description of forward mode

In forward mode the sensitivity of basic unknowns (**p**) is calculated first. Once the sensitivity of the basic unknowns of the problem is established, the derivatives of the response functional can be evaluated for an arbitrary number of response functionals. At the moment, the sensitivity analysis up to the second-order is possible.

The following steps are performed for all sensitivity parameters:

- user subroutine "Forward mode first order pseudo-load" is called for each element,
- sensitivity pseudo-load vector is added to the matrix of global pseudo-load vectors $^{i}\tilde{\mathbf{R}}$,
- system of linear equations with multiple right hand sides is solved $\mathbf{K}\frac{\partial \mathbf{p}}{\partial \mathbf{\Phi}}={}^{i}\tilde{\mathbf{R}}$ that yields sensitivities of global (nodal) variables: $\frac{\partial \mathbf{p}}{\partial \mathbf{\Phi}}$,
- user subroutine "Forward mode dependent sensitivity" is called for each element to resolve sensitivity of local (element) variables $\frac{\partial \mathbf{h}_e}{\partial \mathbf{\Phi}}$ in the case of locally coupled problems.

When sensitivity analysis of second-order is performed, additional steps have to be added to the steps above in order to calculate the second-order derivatives. After all first-order derivatives are calculated, the following steps are performed:

- user subroutine “Forward mode second order pseudo-load" is called for each element,
- second-order sensitivity pseudo-load vector is added to the matrix of global pseudo-load vectors $^{ij}\tilde{\mathbf{R}}$
- system of linear equations with multiple right hand sides is solved $\mathbf{K}\frac{\partial^2 \mathbf{p}}{\partial \Phi_i \partial \Phi_j}={}^{ij}\tilde{\mathbf{R}}$ that yields second-order sensitivities of global (nodal) variables: $\frac{\partial^2 \mathbf{p}}{\partial \Phi_i \partial \Phi_j}$
- user subroutine "Forward mode dependent sensitivity" is called for each element to resolve second-order sensitivity of local (element) variables $\frac{\partial^2 \mathbf{h}_e}{\partial \Phi_i \partial \Phi_j}$ in the case of locally coupled problems.

Thus, in order to perform first-order sensitivity analysis two additional user subroutines are required “Forward mode first order pseudo-load and “Forward mode dependent sensitivity” user subroutines. In order to perform second-order sensitivity analysis, another additional user subroutine is required: “Forward mode second order pseudo-load”. The user subroutine “Forward mode dependent sensitivity” is independent of the order of sensitivity analysis and is therefore the same for an arbitrary order of sensitivity analysis. Full example of the element that supports sensitivity analysis is presented in section `Finite Strain Element for Direct and Sensitivity Analysis`.

The necessary input data that defines the sensitivity problem and the command that actually solves the sensitivity problem are described in section `All Sensitivity Analysis Commands` and `General Forward Mode Sensitivity Analysis Example`.

Sensitivity of the nodal unknowns

Sensitivity of the nodal unknowns in time *n+1* and *n* is stored into nodal data fields nd$$["st"] and nd$$["sp"]. The data is for *i*-th node organized as follows (nϕ = NoSensParameters)

$\{\frac{\partial p_{i,1}}{\phi_1},\frac{\partial p_{i,2}}{\phi_1},\ldots,\frac{\partial p_{i,2}}{\phi_{is}}\ldots,\frac{\partial p_{i,NoDOF}}{\phi_1},\ldots,\frac{\partial^2 p_{i,1}}{\phi_1\,\phi_1},\ldots\frac{\partial^2 p_{i,2}}{\phi_{is}\,\phi_{js}},\ldots\}$ and obtained for first derivatives by SMSIO["Sensitivity DOFs"[is][node, dof]] and for second derivatives by SMSIO["Sensitivity DOFs"[is, js][node, dof]]

Sensitivity of the internal unknowns

Sensitivity of the internal unknowns in time *n+1* and *n* is stored in ed$$["ht"] and ed$$["hp"] fields. Organization of the data is left to the user.

Implementation of design velocity fields

Velocity field can be time independent (parameter "P" and shape "S") or time dependent (essential boundary "EBC" and natural boundary "NBC"). The time dependent velocity field is related to the prescribed boundary conditions and is composed of three separate values:

$\Delta\mathbf{\Psi}_t$ or $\Delta\mathbf{\Psi}$ - current value of design velocity field

$\Delta\boldsymbol{\Psi}_p$ - value at the end of the previous time step

$\boldsymbol{\Delta\Psi_{ref}}$ - vector associated with the pattern of the applied velocity field

Time dependent velocity field is updated in a same way as prescribed boundary conditions at the beginning of each time step (see `Main iterative loop`)

$\Delta\boldsymbol{\Psi}_t := \Delta\boldsymbol{\Psi}_p + \Delta\lambda\ \boldsymbol{\Delta\Psi_{ref}}$.

Design velocity fields are stored in a nodal data field nd$$["ADVF"] in compressed form. Only non-zero velocity fields are stored. The time dependent velocity field is composed of three separate values ($\Delta\boldsymbol{\Psi}_t$, $\Delta\boldsymbol{\Psi}_p$, $\boldsymbol{\Delta\Psi_{ref}}$ in this order) which are counted as three separate fields. The data field is organized as follows

$\{\Delta\Psi_1,\Delta\Psi_2,\ldots,\Delta\Psi_{NoDesignVelocityFields},0,1\}$

The last two values represent zero and Identity velocity fields which are always present.

For the use within the elements codes is the "ADVF" field translated into "SDVF" field with the help of es$$["SensPVFIndex", i] data structure. The proper translation of "ADVF" field into boundary condition related sensitivity data is done with the help of ns$$[i,"SensB-VFIndex", j,k] data structure.

Data structures relevant for forward mode

form	I/O parameter	description
SMSIO["Shape velocity field"[*is*][*node, coor*]]	nd$$[*node*, "SDVF", *index (is), index (coor)*]	returns value of first order shape velocity field for *is*-th sensitivity parameter in *node*-th element node for *coor* -th coordinate ($X_1 \equiv X$, $X_2 \equiv Y$, $X_3 \equiv Z$) ($\frac{\partial X_{e,coor}}{\partial \phi_{is}}$)
SMSIO["Shape velocity field"[*is, js*][*node, coor*]]	nd$$[*node*, "SDVF", *index (is, js), index (coor)*]	returns value of second order shape velocity field for *is, js*-th sensitivity parameters in *node*-th element node for *coor* -th coordinate ($\frac{\partial^2 X_{e,coor}}{\partial \phi_{is} \partial \phi_{js}}$)
SMSIO["Parameter velocity field"[*is*][*node, iparm*]]	nd$$[*node*, "SDVF", *index (is), index (iparm)*]	returns the value of first order parameter velocity fields for *is*-th sensitivity parameter in *node*-th element node for *iparm*-th constant defined by SMSSensitivityNames ($\frac{\partial d_{e,iparm}}{\partial \phi_{is}}$)
SMSIO["Parameter velocity field"[*is, js*][*node, iparm*]]	nd$$[*node*, "SDVF", *index (is,js), index (iparm)*]	returns the value of second order parameter velocity field for *is, js*-th sensitivity parameters in *node*-th element node for *iparm*-th constant defined by SMSSensitivityNames ($\frac{\partial^2 d_{e,par}}{\partial \phi_{is} \partial \phi_{js}}$)
SMSIO["Sensitivity DOFs"[*is*][*node, dof*]]	nd$$[*node*,"st", *index (is), dof*]	returns first derivative of *dof*-th nodal DOF for *is*-th sensitivity parameter in *node*-th element node ($\frac{\partial p_{e,dof}}{\partial \phi_{is}}$)
SMSIO["Sensitivity DOFs"[*is, js*][*node, dof*]]	nd$$[*node*,"st", *index (is, js), dof*]	returns second derivative of *dof*-th nodal DOF for *is, js*-th sensitivity parameters in *node*-th element node ($\frac{\partial^2 p_{e,dof}}{\partial \phi_{is} \partial \phi_{js}}$)
SMSIO["Sensitivity DOFs n"[*is*][*node, dof*]] SMSIO["Sensitivity DOFs n"[*is, js*][*node, dof*]]	nd$$[*node*, "sp", *dIndex, dof*]	returns sensitivity of nodal DOFs at time n
SMSIO[*value*, "Add to" or "Export to", "First order pseudo load"[*is*][*dof*]]	s$$[*index (is), dof*]	adds or exports to first order sensitivity pseudo load *value* for *is-th* sensitvity parameter and *dof -th* nodal
SMSIO[*value*, "Add to" or "Export to", "Second order pseudo load"[*is,js*][*dof*]]	s$$[*index (is,js), dof*]	adds or exports to second order sensitivity pseudo load *value* for *is, js -th* sensitvity parameters and *dof -th* nodal

`IO Data Management` commands related to forward mode.

IData	description
idata$$["SensitivityScheme"]	general sensitivity analysis scheme 1 – first order forward mode sensitivity analysis 2 – first order forward mode sensitivity, second order forward mode sensitivity
idata$$["Task"]	6 – global part of first order forward mode sensitivity 7 – local part of first order forward mode sensitivity 8 – global part of second order forward mode sensitivity 9 – local part of second order forward mode sensitivity
idata$$["NoSensParameters"]	number of sensitivity parameters of the problem (Φ)
idata$$["NoSensDerivatives"]	NoFirstOrderDerivatives+NoSecondOrderDerivatives+NoThirdOrderDerivatives
idata$$["SensIndexStart"]	– an index of the first sensitivity parameter within the currently calculated subset of first order sensitivity parameters – in the case of second order sensitivity the index to derivative that corresponds to diagonal element $\{\phi_{\text{SensRowStart}},\phi_{\text{SensRowStart}}\}$ at row data$$["SensRowStart"]
idata$$["SensIndexEnd"]	– an index of the last sensitivity parameter within the currently calculated subset of first order sensitivity parameters – in the case of second order sensitivity the index to derivative that correcponds diagonal element at row data$$["SensRowEnd"]
data$$["SensRowStart"]	an index of the first row in the currently calculated band of matrix of second order derivatives
idata$$["SensRowEnd"]	an index of the last row in the currently calculated band of matrix of second order derivatives
idata$$["SensMaxGroupLength"]	maximum length of the subset of sensitivity derivatives
idata$$["NoFirstOrderDerivatives"]≡ idata$$["NoSensParameters"]	number of first order derivatives
idata$$["NoSecondOrderDerivatives"]= NoSensParameters* (NoSensParameters+1)/2	number of second order derivatives (symmetric matrix)
idata$$["NoThirdOrderDerivatives"]	number of third order derivatives (not yet used)
idata$$["SensIndex"]	index of the currently processed sensitivity derivative for post– processing and debugging
idata$$["NoDesignVelocityFields"]	total number of user defined design velocity fields + 2 (Identity and zero field)
idata$$["NoBCFields"]	number of nonzero boundary conditions related design velocity fields (time dependent velocity fields)

Forward sensitivity related integer type environmental data (in AceFEM: SMTIData["keyword"])

Element specifications	description
es$$["id","NoSensNames"]	number of element input data fields (except the nodal unknowns!) **AceFEM**: SMTIDomainData[*dID*,"NoSensNames"]
es$$["id","ShapeSensitivity"]	1 ⇒ shape sensitivity is enabled 0 ⇒ shape sensitivity is not enabled **AceFEM**: SMTIDomainData[*dID*, "ShapeSensitivity"]
es$$["id","EBCSensitivity"]	1 ⇒ essential boundary condition sensitivity is enabled 0 ⇒ essential boundary condition sensitivity is not enabled **AceFEM**: SMTIDomainData[*dID*, "EBCSensitivity"]
es$$["id","SensitivityOrder"]	Order of sensitivity analysis that is supported by the element: 0 ⇒ sensitivity analysis is not supported 1 ⇒ first order sensitivity analysis is supported 2 ⇒ second order sensitivity analysis is supported **AceFEM**: SMTIDomainData[*dID*, "SensitivityOrder"]
es$$["SensitivityNames",*i*]	set of element input data fields identifiers (e.g. "X", "Y","u",..., except the nodal unknowns!) $\{\psi_{e,1},\ldots,\psi_{e,\text{NoSensNames}}\}$ **AceFEM**: $\psi_{e,i} \equiv$ SMTIDomainData[*dID*,"SensitivityNames"]⟦i⟧
es$$["SensPVFIndex", *i*]	is a vector of indeces that point to the "ADVF" node data field where the element general design velocity fields are stored $\left\{\frac{d\psi_{e,1}}{d\phi_1}, \frac{d\psi_{e,2}}{d\phi_1}, \ldots, \frac{d\psi_{e,\text{NoSensNames}}}{d\phi_1},\ldots, \frac{d^2\psi_{e,\text{NoSensNames}}}{d^2\phi_{n\phi}}\right\}$ Data is used to interpret the "SDVF" keyword. **AceFEM**: $index\left(\partial\psi_{e,k}/\partial\phi_j\right) \equiv$ SMTIDomainData[*dID*,"SensPVFIndex"]⟦(j−1) NoSensNames+k⟧ $index\left(\partial^2\psi_{e,k}/\partial\phi_i\,\partial\phi_j\right) \equiv$ SMTIDomainData[*dID*,"SensPVFIndex"]⟦ (sensPos (i,j)−1) NoSensNames+k⟧
es$$["SensLowerOrderIndex", *i*]	for all higher order sensitivity derivatives indeces of the sensitvity parameters (or 0 if the lower order derivatives does not exist) $\{\ldots,$ sensPos (ϕ_i^n), sensPos (ϕ_j^n), sensPos (ϕ_k^n), sensPos (ϕ_i^n,ϕ_j^n), sensPos (ϕ_i^n,ϕ_k^n), sensPos (ϕ_j^n,ϕ_k^n), ... n=1,...,NoSensDerivatives$\}$
es$$["SensIndexi",*i*]	≡ sensPos (ϕ_i^n) (taken from es$$["SensLowerOrderIndex"] vector)
es$$["SensIndexj",j]	≡ sensPos (ϕ_j^n) (taken from es$$["SensLowerOrderIndex"] vector)

Element specification data related to sensitivity analysis.

Nodal data	descripion
nd$$[*i*,"st", *j*, *k*]	current $j-$th derivative of the $k-$th nodal d.o.f in $i-th$ node **AceFEM**: $\frac{\partial \hat{p}_{i,k}}{\phi_j} \equiv$ SMTNodeData[i,"st"]⟦(j−1) NoDOF+k⟧ $\frac{\partial^2 \hat{p}_{i,k}}{\phi_j \phi_m} \equiv$ SMTNodeData[i,"st"]⟦(sensPos (j,m)−1) NoDOF+k⟧
nd$$[*i*,"sp", *j*, *k*]	$j-$th derivative of the $k-$th nodal d.o.f in $i-th$ node in time n
nd$$[*i*, "SDVF", *j*, *k*]	first or second order sensitivity of $k-$ th element general input data field with respect to the $j-$th sensitivity derivative in i $-th$ node (components of the element general design velocity matrix $D_{\phi} \Psi_e$) input structure is translated into an appropriate position in the "ADVF" data field (nd$$[*i*, "SDVF", *j*, *k*]≡nd$$[i, "ADVF",*position* (*j*,*k*)]) **AceFEM**: no ekvivalent (interpreted, not an actual data)
nd$$[*i*, "ADVF", *j*]	the $j-$th nonzero velocity field in $i-th$ node **AceFEM**: SMTNodeData[i,"ADVF"]⟦j⟧
ns$$[i,"SensBVFIndex", *j*,*k*]	is an index of the boundary conditions velocity field that corresponds to the $j-$ th nodal unknown and $k-$th sensitivity derivative in i $-th$ node specification. Index points to the "ADVF" node data field. It is used to construct boundary conditions related sensitvity data structures. **AceFEM**: *index* $(\partial p_j / \partial \phi_k) \equiv$ SMTNodeSpecData[*NodeID*,"SensBVFIndex"]⟦(k−1) NoDOF+j⟧

Element level nodal data related to sensitivity analysis.

AceFEM	descripion
SMTNodeData[*i*,"st"]	current sensitivities in $i-th$ node organized as $\left\{\frac{\partial \hat{p}_{i,1}}{\phi_1}, \frac{\partial \hat{p}_{i,2}}{\phi_1}, \ldots, \frac{\partial \hat{p}_{i,\text{NoDOF}}}{\phi_1}, \ldots, \frac{\partial^2 \hat{p}_{i,1}}{\phi_{n\phi} \phi_{n\phi}}, \frac{\partial^2 \hat{p}_{i,2}}{\phi_{n\phi} \phi_{n\phi}}, \ldots, \frac{\partial^2 \hat{p}_{i,\text{NoDOF}}}{\phi_{n\phi} \phi_{n\phi}}\right\}$
SMTNodeData[*i*,"sp"]	sensitivities in $i-th$ node in previous step
SMTNodeData[*i*, "ADVF"]	all non−zero velocity fields in $i-th$ node

AceFEM nodal data related to sensitivity analysis.

"Forward mode first order pseudo-load" user subroutine

The "Forward mode first order pseudo-load" user subroutine returns a matrix of pseudo-load vectors used in direct implicit analysis to get sensitivities of the global unknowns with respect to arbitrary parameter.

Each design parameter ϕ_{is} for sensitivity analysis is classified according to its actual appearance (or lack of it) in the formulation of a finite element code. Input parameters of finite element user subroutines (e.g. material constants, nodal unknowns, nodal coordinates) that depend on sensitivity parameters are called sensitivity dependent input parameters and denoted with ψ_i. Therefore, the derivatives of ψ_i with respect to ϕ_{is} have to be defined. These derivatives are called design velocity fields and denoted with $d\psi_i$.

- Here is a schematic example of the "Forward mode first order pseudo-load" subroutine.

```
(*
 - keywords used to identify general parameters
 - switch that indicated availability of
   the shape and essential boundary conditions sensitivity
*)
SMSTemplate[
 ...,
 "SMSSensitivityNames" → {"X -spatial coordinate", "Y -spatial coordinate",
    "Z -spatial coordinate", "E -elastic modulus", "ν -poisson ratio", "t -thickness"}
 , "SMSShapeSensitivity" -> True, "SMSEBCSensitivity" -> True,
 ...
]
```

In[635]:=
```
SMSStandardModule["Forward mode first order pseudo-load"];

(*... element IO interface defined by IO data management ...*)
{XIO, uIO} ⊨ SMSIO["All coordinates and DOFs"];
domainData ⊨ SMSIO["Domain data"];

(*... body of the subroutine that evaluates residual Rg ... *)

(*is - index of the current sensitivity
  parameter that runs from SensIndexStart to SensIndexEnd *)
SMSDo[is, SMSIO["SensIndexStart"], SMSIO["SensIndexEnd"]];
(* ϕis -current sensitivity parameter is introduced
  as fictitious parameter that can represent arbitrary parameter*)
ϕis ⊨ SMSFictive[];

(*add shape velocity field ∂X/∂ϕis to coordinates*)
SMSDefineDerivative[SMSIO["Nodal coordinates"], ϕis, SMSIO["Shape velocity field"[is]]];

(*add parameter velocity field ∂d/∂ϕis to all parameters defined by SMSDomainDataNames,
∂d/∂ϕi is interpolated over the element*)
SMSDefineDerivative[SMSIO["Domain data"], ϕis, Nh.SMSIO["Parameter velocity field"[is]]];

(*add first order essential boundary conditions velocity field ∂pe/∂ϕis to nodal DOFs"*)
SMSDefineDerivative[SMSIO["Nodal DOFs"], ϕis, SMSIO["Sensitivity DOFs"[is]]];

(*evaluate sensitivity pseudo-load vector for current sensitivity parameter ϕ
  and export to IO parameters of subroutine*)
SMSIO[SMSD[Rg, ϕi], "Add to", "First order pseudo load"[is]];

SMSEndDo[];
```

"Forward mode dependent sensitivity" user subroutine

The "Forward mode dependent sensitivity" user subroutine has the same structure as "Forward mode first order pseudo-load" user subroutine. It has no return values.

"Forward mode second order pseudo-load" user subroutine

The "Forward mode second order pseudo-load" user subroutine has the same structure as "Forward mode first order pseudo-load" user subroutine with additional data of second-order velocity fields of sensitivity parameters. The subroutine returns a matrix of pseudo-load vectors used within the direct implicit analysis to get sensitivities of the global unknowns with respect to arbitrary parameter.

- Here is a schematic example how the second-order sensitivity pseudo-load vector can be evaluated.

```
SMSStandardModule["Forward mode second order pseudo-load"];

(*... element IO interface defined by IO data management ...*)
{XIO, uIO} ⊨ SMSIO["All coordinates and DOFs"];
domainData ⊨ SMSIO["Domain data"];

(*... body of the subroutine that evaluates residual Rg ... *)

(*two nested loops are generated here where:
  is - is index of ϕis, js - is index of ϕjs *)

(*loop over i-th parameters for row SensRowStart to row SensRowEnd*)
SMSDo[is, SMSIO["SensRowStart"], SMSIO["SensRowEnd"]];
(* ϕi -i-th sensitivity parameter*)
ϕis ⊨ SMSFictive[];

(*add first order shape velocity field ∂X/∂ϕis  to coordinates*)
SMSDefineDerivative[SMSIO["Nodal coordinates"], ϕis, SMSIO["Shape velocity field"[is]]];
(*add first order parameter velocity field ∂d/∂ϕis to all parmeters
 defined by SMSDomainDataNames, ∂d/∂ϕi is interpolated over the element*)
SMSDefineDerivative[SMSIO["Domain data"], ϕis, Nh.SMSIO["Parameter velocity field"[is]]];
(*add first order first order sensitivities ∂pe/∂ϕis to nodal DOFs*)
SMSDefineDerivative[SMSIO["Nodal DOFs"], ϕis, SMSIO["Sensitivity DOFs"[is]]];

(*loop over js-th parameters,
due to the symmetry of second order derivatives only upper triangle is calculated*)
SMSDo[js, is, SMSIO["NoSensParameters"]];
(* ϕjs -j-th sensitivity parameter*)
ϕjs ⊨ SMSFictive[];

(*add first order shape velocity field ∂X/∂ϕjs  to coordinates*)
SMSDefineDerivative[SMSIO["Nodal coordinates"], ϕjs, SMSIO["Shape velocity field"[js]]];

(*add second order shape velocity field ∂(∂X/∂ϕi)/∂ϕj to first order shape velocity field*)
SMSDefineDerivative[SMSIO["Shape velocity field"[is]],
  ϕjs, SMSIO["Shape velocity field"[is, js]]];

(*add first order parameter velocity field ∂d/∂ϕj to all parameters
 defined by SMSDomainDataNames, ∂d/∂ϕj is interpolated over the element*)
SMSDefineDerivative[SMSIO["Domain data"], ϕjs, Nh.SMSIO["Parameter velocity field"[js]]];

(*add second order parameter velocity field
 ∂(∂d/∂ϕis)/∂ϕjs to first order parameter velocity field*)
SMSDefineDerivative[SMSIO["Parameter velocity field"[is]],
  ϕjs, SMSIO["Parameter velocity field"[is, js]]];

(*add first order first order sensitivities ∂pe/∂ϕjs to nodal DOFs*)
SMSDefineDerivative[SMSIO["Nodal DOFs"], ϕjs, SMSIO["Sensitivity DOFs"[js]]];

(*add second order essential boundary conditions
 velocity field ∂(∂pe/∂ϕi)/∂ϕj to first order sensitivityes*)
SMSDefineDerivative[SMSIO["Sensitivity DOFs"[is]], ϕjs, SMSIO["Sensitivity DOFs"[is, js]]];

(*evaluate sensitivity pseudo-load vector for current sensitivity parameters ϕi and ϕj*)
Rtg ⊨ SMSD[SMSD[Rg, ϕis, "Mode" → "Forward"], ϕjs, "Mode" → "Backward"];
```

```
SMSIO[Rtg, "Add to", "Second order pseudo load"[is, js]];

SMSEndDo[];
```

Backward Mode Implementation Notes

General description of backward mode

In backward or adjoint method of sensitivity analysis the derivatives (also called sensitivities) of response functionals F with respect to sensitivity parameters $\boldsymbol{\phi} = \{\phi_i, i = 1, \ldots, n_\phi\}$ are sought:

$\frac{DF}{D\phi_i}$...in case of first order sensitivity analysis or

$\frac{D^2 F}{D\phi_i D\phi_j}$...in case of second order sensitivity analysis

Response functional F can be arbitrary for example displacement or stress in one point, norm of displacements, volume change, plastic work... The aim of backward method of sensitivity analysis is to eliminate the expressions that contain derivatives of solution vectors with intention to calculate derivatives of response functionals directly. Lagrange multiplier method is used for this purpose. Final expressions, obtained with this procedure, strongly depend on the nature of mechanical problem. For better understanding of the procedure, backward method for two classical mechanical problems will be presented in following:

- steady-state, uncoupled class of mechanical problems (e.g. hyperelastic material)
- transient, Gauss-point coupled class of mechanical problems (e.g. elasto-plastic material)

Since the procedure of backward method of sensitivity analysis differs substantially from forward method of sensitivity analysis, also the procedure in AceFEM has some considerable differences in comparison to forward method procedure in AceFEM. The differences are even more pronounced in case of transient coupled class of mechanical problems. In general the procedure of backward sensitivity analysis in AceFEM is:

- Input data phase:
  - Definition of sensitivity problem with **SMTSensitivityProblem**. The command SMTSensitivityProblem is due to the needs of defining backward sensitivity problem expanded with some additional options (see `SMTSensitivityProblem`).
- Analysis phase:
  - Execution of primal analysis.
  - In case of second order backward sensitivity analysis, first order forward sensitivity analysis has to be performed (with **SMTForwardSensitivity**[]).*
  - State of the simulation that contains all the necessary data for backward sensitivity analysis (explained more in detail in following sections) has to be stored (with **SMTDumpState**).
- Backward sensitivity analysis phase:
  - Execution of backward sensitivity analysis with **SMTBackwardSensitivity *command*.**

* For calculation of second order derivatives $\frac{D^2 F}{D\phi_i D\phi_j}$, all first order derivatives of solution vector ($\frac{Dp}{D\phi_i}$ and $\frac{Dp}{D\phi_j}$) have to be known . Therefore, for the second order backward sensitivity analysis, firstly first order **forward** sensitivity analysis has to be performed first!!!

Data structures relevant for backward mode

form	I/O parameter	description
SMSIO["Sensitivity DOFs"[*is*][*node, dof*]]	nd$$[*node*,"st", *index (is), dof*]	returns first derivative of *dof*-th nodal DOF for *is*-th sensitivity parameter in *node*-th element node ($\frac{\partial p_{e,dof}}{\partial \phi_{is}}$)
SMSIO["Sensitivity DOFs"[*is, js*][*node, dof*]]	nd$$[*node*,"st", *index (is, js), dof*]	returns second derivative of *dof*-th nodal DOF for *is, js*-th sensitivity parameters in *node*-th element node ($\frac{\partial^2 p_{e,dof}}{\partial \phi_{is} \partial \phi_{js}}$)
SMSIO["Sensitivity DOFs n"[*is*][*node, dof*]] SMSIO["Sensitivity DOFs n"[*is, js*][*node, dof*]]	nd$$[*node*, "sp", *dIndex, dof*]	returns sensitivity of nodal DOFs at time n
SMSIO[*value*, "Add to" or "Export to", "Adjoint pseudo–load"[*ifunc*][*dof*]]	s$$[*ifunc, dof*]	adds or exports adjoint vector of *sof -th* nodal DOF for *ifunc -th* functional
SMSIO[*value*, "Add to" or "Export to", "Backward sensitivity element storage"[*pos*]]	ed$$["BSETS", *pos*]	adds or exports *value* at position *pos* on the allocated "BSETS" data filed
SMSIO[*value*, "Add to" or "Export to", "Functional derivative"[*ifunc, is*]]	fd$$[*ifunc, index (is)*]	adds or exports *value* of first derivative of *ifunc*-*th* functional for *is*-th parameter ($\frac{\partial F_{ifunc}}{\partial \phi_{is}}$)
SMSIO[*value*, "Add to" or "Export to", "Functional derivative"[*ifunc,is,js*]]	fd$$[*ifunc, index (is,js)*]	adds or exports *value* of second derivative of *ifunc -th* functional for *is, js*-th parameter ($\frac{\partial^2 F_{ifunc}}{\partial \phi_{is} \partial \phi_{js}}$)
SMSIO["Integer input vector"[*i*]]	SMT$$["->TasksStructure.IntegerInput",i]	*i*-th component of integer type user supplied vector by "IntegerInput" option of SMTBackwardSensitivity command
SMSIO["Integer input vector length"]	SMT$$["->TasksStructure.IntegerInputLength"]	length of user defined integer type vector
SMSIO["Real input vector"[*i*]]	SMT$$["->TasksStructure.RealInput",i]	*i*-th component of real type user supplied vector by "RealInput" option of SMTBackwardSensitivity command
SMSIO["Real input vector length"]	SMT$$["->TasksStructure.RealInputLength"]	length of user defined real type vector

`IO Data Management` commands related to backward mode.

data	description
idata$$["SensitivityScheme"]	general sensitivity analysis scheme 3 – first order backward mode sensitivity analysis 4 – first order forward mode sensitivity, second order backward mode sensitivity
idata$$["Task"]	code: 10 – first order backward mode sensitivity 11 – second order backward mode sensitivity
idata$$["NoSensParameters"]	number of sensitivity parameters of the problem (**Φ**)
idata$$["NoResponseFunctionals"]	number of response functionals
idata$$["NoBackwardSteps"]	total number of backward mode steps
idata$$["BackwardStepIndex"]	index of current bacward mode step

SMSExtraSensitivityData	AceGen template constat is vector with the following data:
	{
	1 – Defines the length of additional data storage per element
	needed for the execution of backward mode sensitivity analysis int he
	case of localy coupled problems. The data is s attached at the and of
	ed$$["Data",_] and can be written and read by SMSIO commands.
	}

ed$$["BSETS",i]	Additional backward sensitivity related element storage, it has length SMSExtraSensitivityData[[1]] real numbers.
nd$$[*i*, "BSAV", *j*,*k*]	Value of backward sensitivity adjoint vector for the j $-th$ functional in i $-th$ node and k $-th$ dof. Data is stored into nodal Data field (nd$$[node,"Data"]) and has length NoDOF∗NoResponseFunctionals.
nd$$[*i*, "BSFVF", *j*,*k*]	Value of backward sensitivity velocity field for the j $-th$ functional in i $-th$ node and k $-th$ dof $\frac{\partial F_j}{\partial p_{i\,k}}$. Data is stored into nodal Data field (nd$$[node,"Data"]) and has length NoDOF∗NoResponseFunctional and is stored for n–th node as folows $\{\frac{\partial F_1}{\partial p_{n,1}}, \frac{\partial F_1}{\partial p_{n,2}}, \ldots \frac{\partial F_2}{\partial p_{n,1}}, \frac{\partial F_2}{\partial p_{n,2}}, \ldots \frac{\partial F_{nf}}{\partial p_{n,NoDOF}}\}$

Data structures related to backward mode.

Backward mode for steady-state uncoupled problems

A steady state, uncoupled mechanical problem is described with residual **R** that depends on solution vector **p**. The problem is time-independent, however, due to nonlinearity, the primal analysis may have to be solved in more than one (pseudo) time steps to reach the final state.

$\mathbf{R}(\mathbf{p}) = \mathbf{0}$

Let assume that response functional F in general depends only on the solution **p** in the final (last) time step of primal analysis

$F = F(\mathbf{p})$

Furthermore, the response functional can in general be defined as a sum of function that depends directly on nodal unknowns F^{ext} (e.g. displacement or norm of displacements) and/or Gauss point quantities F^{int} (e.g. strain energy) and is therefore written as:

$F = F^{ext} + F^{int} = F^{ext}(\mathbf{p}) + \sum_e \sum_{g=1}^{n_g} w_g F_g(\mathbf{p}_e), (1)$

In backward method of sensitivity analysis, the implicit derivatives of solution vector $\frac{D\mathbf{p}}{D\phi_i}$ (in case of first order) or $\frac{D^2\mathbf{p}}{D\phi_i D\phi_j}$ (in case of

second order) are eliminated from the equation of calculating derivatives $\frac{DF}{D\phi_i}$ or $\frac{D^2 F}{D\phi_i D\phi_j}$, respectively.

In Lagrange multiplier method, the equation of $\frac{DF}{D\phi_i}$ or $\frac{D^2 F}{D\phi_i D\phi_j}$ is augmented:

$$\frac{D\hat{F}}{D\phi_i} = \frac{DF}{D\phi_i} - \boldsymbol{\lambda}.\frac{D\mathbf{R}}{D\phi_i}, \qquad (2)$$

$$\frac{D^2\hat{F}}{D\phi_i D\phi_j} = \frac{D^2 F}{D\phi_i D\phi_j} - \boldsymbol{\lambda}.\frac{D^2\mathbf{R}}{D\phi_i D\phi_j}, \quad (3)$$

where $\boldsymbol{\lambda}$ is Lagrange multiplier. As can be seen from this equation in finite element method Lagrange multiplier $\boldsymbol{\lambda}$ is a vector of dimensions equal to number of degrees of freedom of the (discretized) mechanical system. Note that $\frac{D\hat{F}}{D\phi_i} = \frac{DF}{D\phi_i}$ since $\frac{D\mathbf{R}}{D\phi_i} = \mathbf{0}$ and also $\frac{D^2\hat{F}}{D\phi_i D\phi_j} = \frac{D^2 F}{D\phi_i D\phi_j}$ since $\frac{D^2\mathbf{R}}{D\phi_i D\phi_j} = \mathbf{0}$. By this method, the proper $\boldsymbol{\lambda}$ is sought so that the parts of the equations (2) and (3) that contain unknown sensitivities $\frac{D\mathbf{p}}{D\phi_i}$ (in case of first order) or $\frac{D^2\mathbf{p}}{D\phi_i D\phi_j}$ (in case of second order) are eliminated from the equation.

To derive the expressions for first and second order sensitivity analysis we write F and $\mathbf{R}$ in dependence on sensitivity parameters ϕ_i: $F = F(\mathbf{p}(\phi_i), \phi_i)$ and $\mathbf{R} = \mathbf{R}(\mathbf{p}(\phi_i), \phi_i)$ (in case of first order) or in dependence on sensitivity parameters ϕ_i and ϕ_j: $F = F(\mathbf{p}(\phi_i, \phi_j), \phi_i, \phi_j)$ and $\mathbf{R} = \mathbf{R}(\mathbf{p}(\phi_i, \phi_j), \phi_i, \phi_j)$ (in case of second order). Then the Eq. (2) and Eq. (3) become:

$$\frac{D\hat{F}}{D\phi_i} = \left(\frac{\partial F^{ext}}{\partial \mathbf{p}} + \underset{e}{A}\sum_{g=1}^{n_g} w_g \frac{\partial F_g}{\partial \mathbf{p}_e} - \underset{e}{A}\sum_{g=1}^{n_g} w_g \boldsymbol{\lambda}_e \cdot \frac{\partial \mathbf{R}_g}{\partial \mathbf{p}_e}\right)\frac{D\mathbf{p}}{D\phi_i} + \frac{\partial F^{ext}}{\partial \phi_i} + \sum_e \sum_{g=1}^{n_g} w_g \frac{\partial F_g}{\partial \phi_i} - \sum_e \sum_{g=1}^{n_g} w_g \boldsymbol{\lambda}_e \cdot \frac{\partial \mathbf{R}_g}{\partial \phi_i}, \qquad (4)$$

$$\frac{D^2\hat{F}}{D\phi_i D\phi_j} = \left(\frac{\partial F^{ext}}{\partial \mathbf{p}} + \underset{e}{A}\sum_{g=1}^{n_g} w_g \frac{\partial F_g}{\partial \mathbf{p}_e} - \underset{e}{A}\sum_{g=1}^{n_g} w_g \boldsymbol{\lambda}_e \cdot \frac{\partial \mathbf{R}_g}{\partial \mathbf{p}_e}\right)\frac{D^2\mathbf{p}}{D\phi_i D\phi_j} + \frac{\partial^2\left(F^{ext} + \sum_e \sum_{g=1}^{n_g} w_g F_g\right)}{\partial \mathbf{p}\,\partial \phi_j}\frac{D\mathbf{p}}{D\phi_i} + \frac{\partial^2\left(F^{ext} + \sum_e \sum_{g=1}^{n_g} w_g F_g\right)}{\partial \mathbf{p}\,\partial \phi_i}\frac{D\mathbf{p}}{D\phi_j} + \qquad (5)$$

$$\frac{\partial^2\left(F^{ext} + \sum_e \sum_{g=1}^{n_g} w_g F_g\right)}{\partial \mathbf{p}^2}\frac{D\mathbf{p}}{D\phi_i}\frac{D\mathbf{p}}{D\phi_j} + \frac{\partial^2\left(F^{ext} + \sum_e \sum_{g=1}^{n_g} w_g F_g\right)}{\partial \phi_i\,\partial \phi_j} - \sum_e \sum_{g=1}^{n_g} w_g \boldsymbol{\lambda}_e \cdot \left(\frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e\,\partial \phi_j}\frac{D\mathbf{p}_e}{D\phi_i} + \frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e\,\partial \phi_i}\frac{D\mathbf{p}_e}{D\phi_j} + \frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e{}^2}\frac{D\mathbf{p}_e}{D\phi_i}\frac{D\mathbf{p}_e}{D\phi_j} + \frac{\partial^2 \mathbf{R}_g}{\partial \phi_i\,\partial \phi_j}\right)$$

The equations (4) and (5) are written in a manner as the equations of mechanical problems are usually discretized and solved in finite element method, where the individual finite element contribution to global residual $\mathbf{R}$ is usually calculated with numerical integration. $\mathbf{R}_g$ is the Gauss point contribution to the residual vector, w_g is Gauss point weight, n_g is the number of Gauss points and $\underset{e}{A}$ stands for the assembly procedure.

To eliminate terms that contain unknown derivatives $\frac{D\mathbf{p}}{D\phi_i}$ or in case of second order $\frac{D^2\mathbf{p}}{D\phi_i D\phi_j}$, we have to determine Lagrange multiplier $\boldsymbol{\lambda}$ so that the terms in first bracket in Eq. (4) and Eq. (5) is equal to 0:

$$\frac{\partial F^{ext}}{\partial \mathbf{p}} + \underset{e}{A}\sum_{g=1}^{n_g} w_g \frac{\partial F_g}{\partial \mathbf{p}_e} - \underset{e}{A}\sum_{g=1}^{n_g} w_g \boldsymbol{\lambda}_e \cdot \frac{D\mathbf{R}_g}{D\mathbf{p}_e} = \mathbf{0}, \quad (6)$$

Considering that $\underset{e}{A}\sum_{g=1}^{n_g} w_g \frac{D\mathbf{R}_g}{D\mathbf{p}_e}$ is tangent matrix, the expression (6) can be rewritten

$$\mathbf{K}^T.\boldsymbol{\lambda} = -\frac{\partial F^{ext}}{\partial \mathbf{p}} + \underset{e}{A}\sum_{g=1}^{n_g} w_g \frac{\partial F_g}{\partial \mathbf{p}_e}, \quad (7)$$

This expression represents linear system of equations, similar to the system of equations that has to be solved in each step of primal analysis. The right-hand side can therefore be treated as **backward sensitivity pseudo-load vector $\tilde{\mathbf{R}}$**

$$\mathbf{K}^T.\boldsymbol{\lambda} = -\tilde{\mathbf{R}} = -\left(\tilde{\mathbf{R}}^{ext} + \tilde{\mathbf{R}}^{int}\right), \quad (8)$$

$$\tilde{\mathbf{R}}^{ext} = \frac{\partial F^{ext}}{\partial \mathbf{p}}$$

$$\tilde{\mathbf{R}}^{int} = \underset{e}{A}\tilde{\mathbf{R}}_e = \underset{e}{A}\sum_{g=1}^{n_g} w_g \tilde{\mathbf{R}}_g$$

$$\tilde{\mathbf{R}}_g = \frac{\partial F_g}{\partial \mathbf{p}_e}$$

The solution of Eq. (8) is $\boldsymbol{\lambda}$. In next step, derivatives of response functional $\frac{DF}{D\phi_i}$ or $\frac{D^2 F}{D\phi_i D\phi_j}$ can be calculated from (the rest terms) of Eq. (4) or Eq. (5):

$$\frac{\mathrm{D}\hat{\mathrm{F}}}{\mathrm{D}\phi_i} = \frac{\partial \mathrm{F}^{\mathrm{ext}}}{\partial \phi_i} + \frac{\partial \mathrm{F}^{\mathrm{int}}}{\partial \phi_i} - \boldsymbol{\lambda}\,\frac{\mathrm{D}\mathbf{R}}{\mathrm{D}\phi_i} = \frac{\partial \mathrm{F}^{\mathrm{ext}}}{\partial \phi_i} + \mathop{\mathrm{A}}_{\mathrm{e}} \sum_{g=1}^{n_g} w_g \frac{\partial \mathrm{F}_g}{\partial \phi_i} - \mathop{\mathrm{A}}_{\mathrm{e}} \sum_{g=1}^{n_g} w_g\, \boldsymbol{\lambda}\, \frac{\partial \mathbf{R}_g}{\partial \phi_i} = \frac{\partial \mathrm{F}^{\mathrm{ext}}}{\partial \phi_i} + \sum_e \sum_{g=1}^{n_g} \left(\frac{\partial \mathrm{F}_g}{\partial \phi_i} - w_g\, \boldsymbol{\lambda}_e\, \frac{\partial \mathbf{R}_g}{\partial \phi_i} \right) \quad (9)$$

$$\frac{\mathrm{D}^2\hat{\mathrm{F}}}{\mathrm{D}\phi_i\,\mathrm{D}\phi_j} = \frac{\partial^2\left(\mathrm{F}^{\mathrm{ext}} + \sum_e \sum_{g=1}^{n_g} w_g \mathrm{F}_g\right)}{\partial \mathbf{p}\,\partial \phi_j} \frac{\mathrm{D}\mathbf{p}}{\mathrm{D}\phi_i} + \frac{\partial^2\left(\mathrm{F}^{\mathrm{ext}} + \sum_e \sum_{g=1}^{n_g} w_g \mathrm{F}_g\right)}{\partial \mathbf{p}\,\partial \phi_i} \frac{\mathrm{D}\mathbf{p}}{\mathrm{D}\phi_j} + \frac{\partial^2\left(\mathrm{F}^{\mathrm{ext}} + \sum_e \sum_{g=1}^{n_g} w_g \mathrm{F}_g\right)}{\partial \mathbf{p}^2} \frac{\mathrm{D}\mathbf{p}}{\mathrm{D}\phi_i} \frac{\mathrm{D}\mathbf{p}}{\mathrm{D}\phi_j} + \quad (10)$$

$$\frac{\partial^2\left(\mathrm{F}^{\mathrm{ext}} + \sum_e \sum_{g=1}^{n_g} w_g \mathrm{F}_g\right)}{\partial \phi_i\,\partial \phi_j} - \sum_e \sum_{g=1}^{n_g} w_g\, \boldsymbol{\lambda}_e \cdot \left(\frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e\,\partial \phi_j} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_i} + \frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e\,\partial \phi_i} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_j} + \frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e{}^2} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_i} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_j} + \frac{\partial^2 \mathbf{R}_g}{\partial \phi_i\,\partial \phi_j} \right)$$

$$= \left(\frac{\partial^2 \mathrm{F}^{\mathrm{ext}}}{\partial \mathbf{p}\,\partial \phi_j} \frac{\mathrm{D}\mathbf{p}}{\mathrm{D}\phi_i} + \frac{\partial^2 \mathrm{F}^{\mathrm{ext}}}{\partial \mathbf{p}\,\partial \phi_i} \frac{\mathrm{D}\mathbf{p}}{\mathrm{D}\phi_j} + \frac{\partial^2 \mathrm{F}^{\mathrm{ext}}}{\partial \mathbf{p}^2} \frac{\mathrm{D}\mathbf{p}}{\mathrm{D}\phi_i} \frac{\mathrm{D}\mathbf{p}}{\mathrm{D}\phi_j} + \frac{\partial^2 \mathrm{F}^{\mathrm{ext}}}{\partial \phi_i\,\partial \phi_j} \right) +$$

$$\sum_e \sum_{g=1}^{n_g} w_g \left(\left(\frac{\partial^2 \mathrm{F}_g}{\partial \mathbf{p}_e\,\partial \phi_j} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_i} + \frac{\partial^2 \mathrm{F}_g}{\partial \mathbf{p}_e\,\partial \phi_i} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_j} + \frac{\partial^2 \mathrm{F}_g}{\partial \mathbf{p}_e{}^2} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_i} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_j} + \frac{\partial^2 \mathrm{F}_g}{\partial \phi_i\,\partial \phi_j} \right) \right.$$

$$\left. - \boldsymbol{\lambda}_e \cdot \left(\frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e\,\partial \phi_j} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_i} + \frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e\,\partial \phi_i} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_j} + \frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e{}^2} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_i} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_j} + \frac{\partial^2 \mathbf{R}_g}{\partial \phi_i\,\partial \phi_j} \right) \right)$$

If individual element contribution explicitly depends only on a subset of sensitivity parameters $\boldsymbol{\phi}_e$ then a more optimized form of (10) leads to

$$\frac{\mathrm{D}\hat{\mathrm{F}}}{\mathrm{D}\phi_i} = \frac{\partial \mathrm{F}^{\mathrm{ext}}}{\partial \phi_i} + \sum_e \sum_{g=1}^{n_g} w_g \begin{cases} \left(\frac{\partial \mathrm{F}_g}{\partial \phi_i} - \boldsymbol{\lambda}_e \frac{\partial \mathbf{R}_g}{\partial \phi_i} \right) & i \in \boldsymbol{\phi}_e \\ 0 & i \notin \boldsymbol{\phi}_e \end{cases}, \quad (11)$$

$$\frac{\mathrm{D}^2\hat{\mathrm{F}}}{\mathrm{D}\phi_i\,\mathrm{D}\phi_j} = \left(\frac{\partial^2 \mathrm{F}^{\mathrm{ext}}}{\partial \mathbf{p}\,\partial \phi_j} \frac{\mathrm{D}\mathbf{p}}{\mathrm{D}\phi_i} + \frac{\partial^2 \mathrm{F}^{\mathrm{ext}}}{\partial \mathbf{p}\,\partial \phi_i} \frac{\mathrm{D}\mathbf{p}}{\mathrm{D}\phi_j} + \frac{\partial^2 \mathrm{F}^{\mathrm{ext}}}{\partial \mathbf{p}^2} \frac{\mathrm{D}\mathbf{p}}{\mathrm{D}\phi_i} \frac{\mathrm{D}\mathbf{p}}{\mathrm{D}\phi_j} + \frac{\partial^2 \mathrm{F}^{\mathrm{ext}}}{\partial \phi_i\,\partial \phi_j} \right) + \quad (13)$$

$$\sum_e \sum_{g=1}^{n_g} w_g \left(\left(\frac{\partial^2 \mathrm{F}_g}{\partial \mathbf{p}_e{}^2} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_i} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_j} - \boldsymbol{\lambda}_e \cdot \frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e{}^2} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_i} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_j} \right) + \right.$$

$$+ \begin{cases} \frac{\partial^2 \mathrm{F}_g}{\partial \mathbf{p}_e\,\partial \phi_i} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_j} - \boldsymbol{\lambda}_e \frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e\,\partial \phi_i} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_j} & i \in \boldsymbol{\phi}_e \\ 0 & i \notin \boldsymbol{\phi}_e \end{cases}$$

$$+ \begin{cases} \frac{\partial^2 \mathrm{F}_g}{\partial \mathbf{p}_e\,\partial \phi_j} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_i} - \boldsymbol{\lambda}_e \frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e\,\partial \phi_j} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_i} & j \in \boldsymbol{\phi}_e \\ 0 & j \notin \boldsymbol{\phi}_e \end{cases}$$

$$\left. + \begin{cases} \frac{\partial^2 \mathrm{F}_g}{\partial \phi_i\,\partial \phi_j} - \boldsymbol{\lambda}_e \frac{\partial^2 \mathbf{R}_g}{\partial \phi_i\,\partial \phi_j} & i \text{ and } j \in \boldsymbol{\phi}_e \\ 0 & i \text{ and } j \notin \boldsymbol{\phi}_e \end{cases} \right)$$

, however with backward mode AD the evaluation time of (13) is not necessarily reduced when compared to (10). Form (10) can be alternatively rewritten to

$$\frac{\mathrm{D}^2\hat{\mathrm{F}}}{\mathrm{D}\phi_i\,\mathrm{D}\phi_j} = \left(\frac{\partial^2 \mathrm{F}^{\mathrm{ext}}}{\partial \mathbf{p}\,\partial \phi_j} \frac{\mathrm{D}\mathbf{p}}{\mathrm{D}\phi_i} + \frac{\partial^2 \mathrm{F}^{\mathrm{ext}}}{\partial \mathbf{p}\,\partial \phi_i} \frac{\mathrm{D}\mathbf{p}}{\mathrm{D}\phi_j} + \frac{\partial^2 \mathrm{F}^{\mathrm{ext}}}{\partial \mathbf{p}^2} \frac{\mathrm{D}\mathbf{p}}{\mathrm{D}\phi_i} \frac{\mathrm{D}\mathbf{p}}{\mathrm{D}\phi_j} + \frac{\partial^2 \mathrm{F}^{\mathrm{ext}}}{\partial \phi_i\,\partial \phi_j} \right) + \quad (14)$$

$$\sum_e \sum_{g=1}^{n_g} w_g \left(\left(\frac{\partial^2 \mathrm{F}_g}{\partial \mathbf{p}_e{}^2} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_i} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_j} - \boldsymbol{\lambda}_e \cdot \frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e{}^2} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_i} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_j} \right) + \right.$$

$$\left. + \begin{cases} \frac{\partial^2 \mathrm{F}_g}{\partial \mathbf{p}_e\,\partial \phi_j} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_i} + \frac{\partial^2 \mathrm{F}_g}{\partial \mathbf{p}_e\,\partial \phi_i} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_j} + \frac{\partial^2 \mathrm{F}_g}{\partial \phi_i\,\partial \phi_j} & i \text{ or } j \in \boldsymbol{\phi}_e \\ -\boldsymbol{\lambda}_e \left(\frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e\,\partial \phi_j} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_i} + \frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e\,\partial \phi_i} \frac{\mathrm{D}\mathbf{p}_e}{\mathrm{D}\phi_j} + \frac{\partial^2 \mathbf{R}_g}{\partial \phi_i\,\partial \phi_j} \right) & \\ 0 & i \text{ and } j \notin \boldsymbol{\phi}_e \end{cases} \right)$$

or

$$\frac{D^2\hat{F}}{D\phi_i D\phi_j} = \left(\frac{\partial^2 F^{ext}}{\partial \mathbf{p}\,\partial\phi_j}\frac{D\mathbf{p}}{D\phi_i} + \frac{\partial^2 F^{ext}}{\partial \mathbf{p}\,\partial\phi_i}\frac{D\mathbf{p}}{D\phi_j} + \frac{\partial^2 F^{ext}}{\partial \mathbf{p}^2}\frac{D\mathbf{p}}{D\phi_i}\frac{D\mathbf{p}}{D\phi_j} + \frac{\partial^2 F^{ext}}{\partial\phi_i\,\partial\phi_j}\right) + \tag{15}$$

$$\sum_e \sum_{g=1}^{n_g} w_g \left(\left(\frac{\partial^2 F_g}{\partial \mathbf{p}_e{}^2}\frac{D\mathbf{p}_e}{D\phi_i}\frac{D\mathbf{p}_e}{D\phi_j} + \frac{\partial^2 F_g}{\partial \mathbf{p}_e\,\partial\phi_i}\frac{D\mathbf{p}_e}{D\phi_j} - \boldsymbol{\lambda}_e\left(\frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e{}^2}\frac{D\mathbf{p}_e}{D\phi_i}\frac{D\mathbf{p}_e}{D\phi_j} + \frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e\,\partial\phi_i}\frac{D\mathbf{p}_e}{D\phi_j}\right)\right) + \right.$$

$$\left. + \begin{cases} \frac{\partial^2 F_g}{\partial \mathbf{p}_e\,\partial\phi_j}\frac{D\mathbf{p}_e}{D\phi_i} + \frac{\partial^2 F_g}{\partial\phi_i\,\partial\phi_j} - \boldsymbol{\lambda}_e\left(\frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e\,\partial\phi_j}\frac{D\mathbf{p}_e}{D\phi_i} + \frac{\partial^2 \mathbf{R}_g}{\partial\phi_i\,\partial\phi_j}\right) & j \in \boldsymbol{\phi}_e \\ 0 & j \notin \boldsymbol{\phi}_e \end{cases}\right)$$

with the form (15) especially appropriate for automatic derivation with AceGen.

In case of more than one response functional $\mathbf{F} = \{F_k\}$, k = 1, ..., n_F, the above described procedure is similar, only the Eq.(8) becomes the linear system of equations with multiple right-hand sides and the result of this system are Lagrange multipliers $\boldsymbol{\Lambda} = \{\boldsymbol{\lambda}_k\}$, k = 1, ..., n_F, where $\boldsymbol{\lambda}_k$ belongs to the response functional F_k.

Another important remark should be made regarding second order backward sensitivity analysis. As can be seen from Eq. (5) and (10), for calculation of second order derivatives $\frac{D^2 F}{D\phi_i D\phi_j}$, all first order derivatives of solution vector have to be known ($\frac{D\mathbf{p}}{D\phi_i}$ and $\frac{D\mathbf{p}}{D\phi_j}$). The most effective way to calculate $\frac{D\mathbf{p}}{D\phi_i}$ and $\frac{D\mathbf{p}}{D\phi_j}$ is first order forward analysis. The procedure to perform second order backward sensitivity analysis therefore includes evaluation of first order forward sensitivity analysis.

Procedure for first or second order backward sensitivity analysis for steady-state uncoupled problems

In order to perform the backward sensitivity analysis for a steady-state uncoupled problems with AceFEM, three additional finite element user subroutines are required: "Backward mode pseudo-load"" and “Backward mode first order derivatives” and "Backward mode second order derivatives" user subroutine. In first subroutine, derivative $\frac{\partial F_g}{\partial \mathbf{p}}$ is calculated in second subroutine, Eq. (9) and in third Eq. (10) is calculated. In case that the response functional is a quantity that is defined only in nodes F^{int}=0 (e.g. displacement or norm of displacements), the subroutine “"Backward mode pseudo-load” is not required.

The AceFEM procedure for backward sensitivity analysis method for steady-state uncoupled problems:

Primal analysis:

- primal problem is solved
- first order forward sensitivity analysis is performed with **SMTForwardSensitivity[]** (only in case of second order backward sensitivity analysis!).
- $\tilde{\mathbf{R}}^{ext} = \frac{\partial F^{ext}}{\partial \mathbf{p}}$ is calculated by user and stored in **SMTNodeData**["BSFVF"] with the `SMTSetVelocityFields` command for the later use.
- state of simulation has to be stored with **SMTDumpState**[myFileName,”BackwardSensitivity”→True].

Backward sensitivity analysis is performed with command **SMTBackwardSensitivity**[myFileName] as follows:

- $\tilde{\mathbf{R}}^{int} = \underset{e}{A}\,\tilde{\mathbf{R}}_e = \underset{e}{A}\sum_{g=1}^{n_g} w_g\,\tilde{\mathbf{R}}_g$ is assembled by calling user subroutine “Backward mode pseudo-load”
- Backward sensitivity pseudo-load vector $\tilde{\mathbf{R}} = \left(\tilde{\mathbf{R}}^{ext} + \tilde{\mathbf{R}}^{int}\right)$ is formed where $\tilde{\mathbf{R}}^{ext}$ was stored in nd$$[*node*,"BSFVF",*ifunc*,*dof*] (SMTNodeData["BSFVF"]) during primal analysis.
- $\mathbf{K}^T$ is calculated and assembled using "Tangent and residual" user subroutine.
- system of linear equations is solved (Eq. (8)). The result is Lagrange multiplier $\boldsymbol{\lambda}$.
  $\mathbf{K}^T.\boldsymbol{\lambda} = -\tilde{\mathbf{R}}$
- user subroutine “Backward mode first order derivatives” or "Backward mode second order derivatives" is called for each element in which finite element contribution to derivatives of response functional $\frac{DF}{D\phi_i}$ or $\frac{D^2 F}{D\phi_i D\phi_j}$ are calculated from Eq. (9) or Eq. (10), respectively.

- The result of the command SMTBackwardSensitivity gives the derivatives of all response functionals with respect to all sensitivity parameters $\frac{DF}{D\phi_i}$ or $\frac{D^2 F}{D\phi_i D\phi_j}$.

"Backward mode pseudo-load" user subroutine for steady-state uncoupled problems

Let us suppose that the problem has n_F response functionals F_1, F_2, ..., F_{nF} . The element contributions to the response functionals is calculated in Gauss point (e.g. strain energy): $\sum_{g=1}^{n_g} w_g F_{g,1}$, $\sum_{g=1}^{n_g} w_g F_{g,2}$, ... , $\sum_{g=1}^{n_g} w_g F_{g,n_F}$. User subroutine "Backward mode pseudo-load" has for steady-state uncoupled problems the following additional arguments:

I/O argument	IO data	description
s$$[*NoFunctionals*, *NoDOFGlobal*]	SMSIO[$\tilde{\boldsymbol{R}}_g$, "Add to", "Adjoint pseudo–load"[*iF*]]	Backward sensitivity pseudo load vector contains: On input: $\mathbf{0}$ On output: $\tilde{\mathbf{R}}_{e,i_F} = \sum_{g=1}^{n_g} w_g \frac{\partial F_{g,i_F}}{\partial \mathbf{p}_e}$

Specific I/O for "Backward mode pseudo-load" user subroutine.

```
In[1259]:= SMSStandardModule["Backward mode pseudo-load"];
           SMSDo[Ig, 1, SMSIO["No. integration points"]];
           FgiF ⊨ ...; (* iF-th response functional iF=1,2,...nF*);
           DF1gDpe ⊨ -SMSD[FgiF, pe];
           SMSIO[DF1gDpe, "Add to", "Adjoint pseudo-load"[iF]];
           SMSEndDo[];
```

"Backward mode first order derivatives" user subroutine for steady-state uncoupled problems

In this subroutine derivatives element contribution to $\frac{DF}{D\phi_i}$ is calculated. User subroutine "Backward mode first order derivatives" has for steady-state uncoupled problems the following additional arguments:

argument	IO data	description
fd$$[*NoFunctionals*, *NoDerivatives*]	SMSIO[$\frac{DG}{D\phi_{is}}$,"Add to", "Functional derivative"[i_F,i_s]]	On output add to input: $\sum_{g=1}^{n_g}(\frac{\partial F_{g,i_F}}{\partial \phi_{i_s}} - w_g \lambda_{e,i_F} \cdot \frac{\partial \mathbf{R}_g}{\partial \phi_{i_s}})$

Specific I/O for "Backward mode first order derivatives" user subroutine.

```
In[1259]:= SMSStandardModule["Backward mode first order derivatives"];
           NoIp ⊨ SMSIO["No. integration points"];
           SMSDo[Ig, 1, NoIp];
           Rg =. .. (* integration point residual *)
           FgiF ⊨ ... (* iF-th response functional *);
           λeiF ⊨ Flatten[SMSIO["Adjoint vector DOFs"[iF]]];(*adjoin vector for iF-th functional*)
           SMSDo[is, 1, SMSIO["NoSensParameters"]];
           ϕis ⊨ SMSFictive[];
           (*Definition of derivatives of all relevant input parameters
            of subroutine with respect to sensitivity parameter ϕis *)
           SMSDefineDerivative[ψj, ϕis, DψjDϕis];
           ...
           G = FgiF - λeiF.Rg (* ADB pseudo potential *)
           DGDϕis ⊨ SMSD[G, ϕis];
           SMSIO[DGDϕis, "Add to", "Functional derivative"[iF, is]];
           SMSEndDo[];
           SMSEndDo[];
```

"Backward mode second order derivatives" user subroutine for steady-state uncoupled problems

In this subroutine derivatives element contribution to $\frac{D^2 F_{i_F}}{D\phi_{i_s} D\phi_{j_s}}$ is calculated. User subroutine "Backward mode second order deriva-

tives" has for steady-state uncoupled problems the following additional arguments:

argument	IO data	description
fd$$[*NoFunctionals, NoDerivatives*]	SMSIO[$\frac{D^2 G}{D\phi_{is} D\phi_{js}}$,"Add to", "Functional derivative"[i_F,]]	On output add to input: $\sum_{g=1}^{n_g} W_g ((\frac{\partial^2 F_{g,i_F}}{\partial \mathbf{p}_e \partial \phi_{j_s}} \frac{D\mathbf{p}_e}{D\phi_{i_s}} + \frac{\partial^2 F_{g,i_F}}{\partial \mathbf{p}_e \partial \phi_{i_s}} \frac{D\mathbf{p}_e}{D\phi_{j_s}} + \frac{\partial^2 F_{g,i_F}}{\partial \mathbf{p}_e{}^2} \frac{D\mathbf{p}_e}{D\phi_{i_s}} \frac{D\mathbf{p}_e}{D\phi_{j_s}} + \frac{\partial^2 F_{g,i_F}}{\partial \phi_{i_s} \partial \phi_{j_s}}) - \lambda_e .(\frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e \partial \phi_{j_s}} \frac{D\mathbf{p}_e}{D\phi_{i_s}} + \frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e \partial \phi_{i_s}} \frac{D\mathbf{p}_e}{D\phi_{j_s}} + \frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e{}^2} \frac{D\mathbf{p}_e}{D\phi_{i_s}} \frac{D\mathbf{p}_e}{D\phi_{j_s}} + \frac{\partial^2 \mathbf{R}_g}{\partial \phi_{i_s} \partial \phi_{j_s}}))$

Specific I/O for "Backward mode second order derivatives" user subroutine.

```
In[1259]:= SMSStandardModule["Backward mode second order derivatives"];
           SMSDo[Ig, 1, SMSIO["No. integration points"]];
           Rg =. .. (* integration point residual *)
           FgiF ⊨ ... (* iF-th response functional iF=1,2,...nF*);
           λeiF ⊨ Flatten[SMSIO["Adjoint vector DOFs"[iF]]];(*adjoin vector for iF-th functional*)

           SMSDo[is, 1, SMSIO["NoSensParameters"]];
           ϕis ⊨ SMSFictive[];
           SMSDefineDerivative[SMSIO["Nodal DOFs"], ϕis, SMSIO["Sensitivity DOFs"[is]]];
           (*Definition of derivatives of all relevant input parameters ψj of subroutine with respect
            to sensitivity parameter ϕis were already defined in forward mode first order analysis*)
           SMSDefineDerivative[SMSIO["Nodal coordinates"], ϕis, SMSIO["Shape velocity field"[is]]];
           SMSDefineDerivative[SMSIO["Domain data"], ϕis, Nh.SMSIO["Parameter velocity field"[is]]];

           G = FgiF - λeiF.Rg (* ADB pseudo potential *)
           DGDϕi ⊨ SMSD[G, ϕis, "Mode" → "Forward"];

           SMSDo[js, is, SMSIO["NoSensParameters"]];
           ϕjs ⊨ SMSFictive[];
           SMSDefineDerivative[SMSIO["Nodal DOFs"], ϕjs, SMSIO["Sensitivity DOFs"[js]]];
           SMSDefineDerivative[SMSIO["Nodal coordinates"], ϕjs, SMSIO["Shape velocity field"[js]]];
           SMSDefineDerivative[SMSIO["Domain data"], ϕjs, Nh.SMSIO["Parameter velocity field"[js]]];
           DGDϕiϕj ⊨ SMSD[DGDϕi, ϕj, "Mode" → "Backward"];
           SMSIO[DGDϕiϕj, "Add to", "Functional derivative"[iF, is, js]];
           SMSEndDo[];

           SMSEndDo[];
           SMSEndDo[];
```

All Sensitivity Analysis Commands

SMTSensitivityProblem

SMTSensitivityProblem[*options*]

SMTSensitivityProblem[*firstOrderVelocityFields*,*options*] ≡
SMTSensitivityProblem["FirstOrderVelocityFields"->*firstOrderVelocityFields*, "Mode"->"Forward",*options*]

Definition of sensitivity problem.

option	default	description
"Mode"	"Forward"	Method of sensitivity analysis. Possible methods are "Forward" and "Backward".
"Order"	1	Order of sensitivity analysis. Currently the sensitivity analysis up to second-order is supported.

"FirstOrderVelocity-Fields"	{}	Definition of design velocity fields for first-order sensitivity analysis:
		{
		{sID_1, {$VelocityFieldType_{1,1}$, $VelocityFieldID_{1,1}$, $RulesForField_{1,1}$},
		{$VelocityFieldType_{1,2}$, $VelocityFieldID_{1,2}$, $RulesForField_{1,2}$},...},
		{sID_2,{$VelocityFieldType_{2,1}$, $VelocityFieldID_{2,1}$, $RulesForField_{2,1}$},...}},
		....
		}

"SecondOrderVeloc-ityFields"	{}	Definition of design velocity fields for second-order sensitivity analysis: {{{sID_i,sID_j}, {$VelocityFieldType_{ij\,1}$, $VelocityFieldID_{ij\,1}$, $RulesForField_{ij\,1}$},...},...}} sID_i and sID_j are the identifications of the two sensitivity parameters for which the design velocity field of second-order is defined. The sID_i and sID_j can be different (mixed derivative: $\frac{D\psi_k^2}{D\phi_i D\phi_j}$) or the same (second derivative: $\frac{D\psi_k^2}{D\phi_i^2}$).
"ResponseFunction-als"	{}	Definition of response functional identifications (**only in the case of backward mode!**). {$FunctionalID_1$,$FunctionalID_2$,} The $FunctionalID_i$ is a keyword that marks specific response functional with index i.
"NoSensitivityPara-meters"	Automatic	Number of sensitivity parameters (**only in the case of first order backward mode!**).
"MaxGroupLength"	Automatic	Maximum number of derivatives for which the sensitivity analysis is solved together using the solution to the system of linear equation with multiple right hand sides. If the number is lower the sensitivity analysis requires less computer memory, however it will require longer computational time.

Options for the SMTSensitivityProblem command.

Design velocity fields

Design velocity fields are needed for forward mode sensitivity analysis. The *sID* parameter is a keyword that marks specific design sensitivity parameter. The *VelocityFieldType* parameter defines the type of design velocity field accordingly to its actual appearance in the finite element solution procedure (e.g., material parameter ("P"), coordinates ("S"), nodal forces ("NBC", ..). The *VelocityFieldID* is a keyword that marks specific velocity field. *RulesForField* is a sequence of rules that defines more in detail which input data field of the finite element user subroutines or which nodal DOF depends on specific sensitivity parameter. Note that, the *VelocityFieldType* is defined by the type of the input data field (e.g., nodal coordinates, surface traction,...) and not by the type of sensitivity parameter. **Only the non-zero velocity fields have to be defined.**

The relation between the specific sensitivity parameter ϕ_j and the input data field ψ_j is defined by design velocity field DΨiDϕj. Design velocity field can be defined explicitly and given as an input data or it can be coded directly into sensitivity related user subroutine. In general, the design velocity fields for forward mode are given explicitly since the number of sensitivity parameters is low. In the case of backward mode we have large number of parameters and consequently the design velocity fields have to be coded into the backward mode related sensitivity user subroutines.

Each input data field ψ_i of the finite procedure that depend on specific sensitivity parameter ϕ_j has nonzero design velocity field assigned $\left(\frac{D\psi_i}{D\phi_j}\right)$. Explicitly given design velocity fields have to be defined and classified according to the rules given below:

VelocityFieldType	Description	*RulesForField*
"P"	material parameter velocity field (also general design velocity field)	$dID_1 \to p_1$, $dID_2 \to p_2$,… time independent velocity field that belongs to the p_i-th general input data field on domain dID_i
"S"	shape velocity field (also general design velocity field)	$dID_1 \to p_1$, $dID_2 \to p_2$,… time independent velocity field that belongs to the chosen spatial coordinate implemented in the element as the p_i-th general input data field on domain dID_i. By default (but not necessary!) the coordinates are the first general parameters of the element thus p_i= 1 ⇒ X coordinate, p_i=2 ⇒ Y coordinate, p_i=3 ⇒ Z coordinate.
"EBC"	essential boundary condition velocity field	$NodeID_1 \to dof_1$, $NodeID_2 \to dof_2$,… time dependent velocity field that belongs to the dof_i-th component of the vector of unknowns in nodes with node identification $NodeID_i$
"NBC"	natural boundary condition velocity field	$NodeID_1 \to dof_1$, $NodeID_2 \to dof_2$,… time dependent velocity field that belongs to the dof_i-th component of the vector of reference values of natural (Neumann) boundary condition in nodes with node identification $NodeID_i$

Codes for the types of velocity fields and corresponding range of elements or nodes where the velocity field is applied.

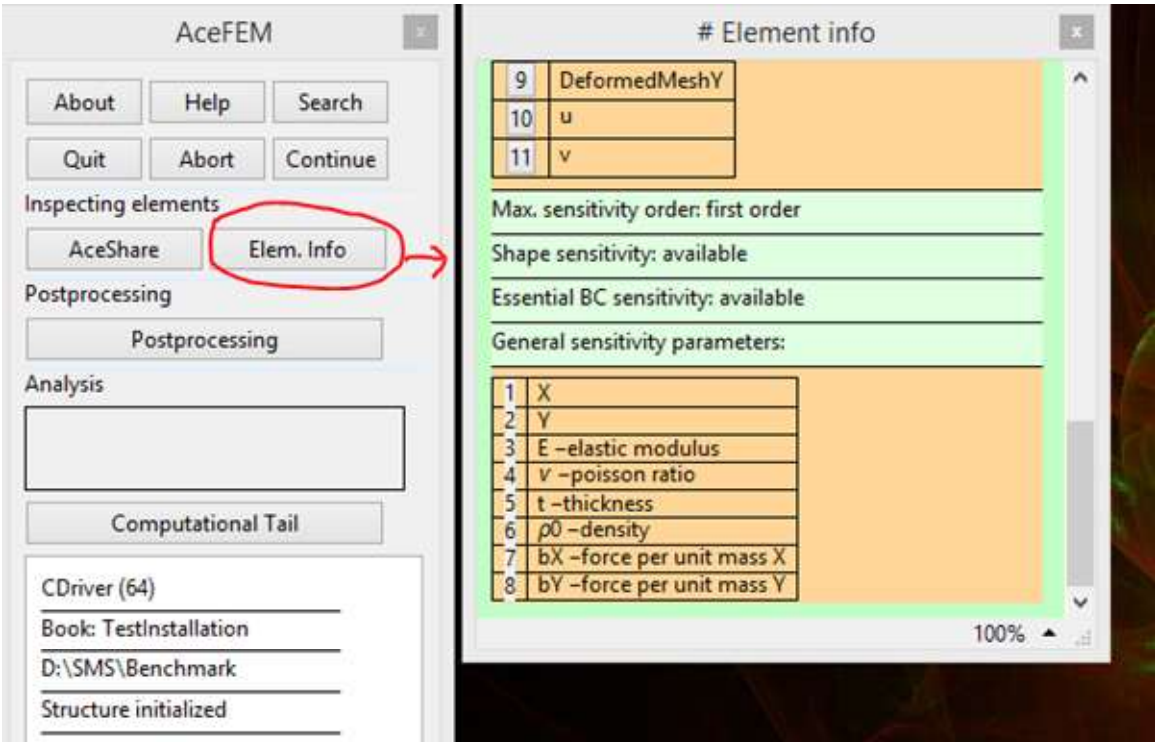
VelocityFieldID	description
Identity	identifies constant velocity field with nodal value 1 ({1:i=1,…,NoNodes})
Null	identifies constant velocity field with nodal value 0 ({0:i=1,…,NoNodes})
arbitrary	unique identification (the nodal values of the corresponding velocity field are defined later by the SMTSetVelocityFields command)

Velocity field identification.

For example, if the sensitivity parameter is an elastic modulus (E) that is constant over the domain of the problem then the corresponding velocity filed that relates the sensitivity parameter elastic modulus and the material constant elastic modulus that appears as a part of finite element formulation (E) is identity vector since $\frac{\partial E}{\partial E} = 1$ (Identity velocity field). The zero velocity field signifies that specific element quantity does not depend on the specific sensitivity parameter. Zero velocity field is default value for velocity field unless nonzero velocity field is defined.

General design velocity fields relate arbitrary sensitivity parameters with the general input data fields of the finite element user subroutine that is used to evaluate the residual. Typically, this includes nodal coordinates, material constants, cross section characteristics, geometrical characteristics, etc. In fact this includes all the input parameters of the "*Tangent and residual*" subroutine, except the nodal unknowns.

General input data is indexed as follows: firstly spatial nodal coordinates are given, followed by the material parameters. E.g.: Ψ_e={"X","Y","E","*v*",...}. All the general fields and their indexes are listed in *Element Info* (which is accessible by clicking on the main AceFEM palette):



SMTSetVelocityFields

SMTSetVelocityFields[{
ID$_1$->{*nodeSelector*, *field_definition*},
…}]

Velocity field is defined by its nodal values in selected nodes (see `Selecting Nodes`) or the function applied on the selected nodes. The function has to be of the form f[node number, X, Y, Z, node specification index]. The *ID* can define either design velocity field (*VelocityFieldID*) or a functional (*FunctionalID*). The *field_definition* depends on the *ID* type as defined below.

field_definitions	description
{…}	list of nodal values of $\Delta\Psi$ for time independent velocity fields ("P" and "S") or a pattern of the applied velocity field $\Delta\Psi_{ref}$ for time dependent velocity fields ("EBC" and "NBC")
{{…}, {…}}	list of nodal values of $\Delta\Psi_{ref}$ and Ψ_p and $\Delta\Psi_t$=Ψ_p
{{…}, {…}, {…}}	list of nodal values of $\Delta\Psi_{ref}$ and Ψ_p and Ψ_t (SMTNextStep overwrites Ψ_t!)
Function[{n,X,Y,Z,*IDIndex*},…]	function applied on selected nodes. The result is a list of nodal values of $\Delta\Psi$ for time independent velocity fields or pattern of the applied velocity field $\Delta\Psi_{ref}$ for time dependent velocity fields. Here n is global node number, X, Y, Z (or X, Y in the case of 2 D problems) node spatial coordinates and *IDIndex* an index of node identification (see Node Identification). *IDIndex* can be skipped.
{Function[{n,X,Y,Z,*IDIndex*},…], Function[{n,X,Y,Z,*IDIndex*},…]}	functions applied on selected nodes to get $\Delta\Psi_{ref}$ and Ψ_p in selected nodes
{Function[{n,X,Y,Z,*IDIndex*},…], Function[{n,X,Y,Z,*IDIndex*},…], Function[{n,X,Y,Z,*IDIndex*},…]}	functions applied on selected nodes to get $\Delta\Psi_{ref}$ and Ψ_p and Ψ_t in selected nodes

Design velocity field definitions (*VelocityFieldID*).

field_definitions	description
$\left\{\frac{\partial F(FunctionalID)}{\partial \mathbf{p}_1}, \frac{\partial F(FunctionalID)}{\partial \mathbf{p}_2}, \ldots\right\}$	list of derivatives of selected functional with respect to DOF' s of selected nodes
Function[{n,X,Y,Z,*IDIndex*},...]	function applied on selected nodes. The result must be vector of derivatives of selected functional with respect to DOF' s of selected node.

Functional velocity field definitions (*FunctionalID)*.

option	default	description
"Set"	False	New definition overwrites all previous given definitions.

Options for the SMTSetVelocityFields command.

The design velocity fields are stored in a nodal data field "ADVF" (see `AceFEM implementation`). Time dependent velocity field is updated in a same way as prescribed boundary conditions at the beginning of each time step (see `Main iterative loop`)

$\Delta\boldsymbol{\Psi}_t := \Delta\boldsymbol{\Psi}_p + \Delta\lambda\ \boldsymbol{\Delta\Psi}_{\mathbf{ref}}$.

SMTNodeData["ADVF"]
returns all design velocity fields

SMTNodeData["BSFVF"]
returns all functional velocity fields

SMTNodeData["ADVF",Table[0,SMTIData["NoDesignVelocityFields"]-2]]
resets all design velocity fields to 0 (the last two fields are Identity and zero fields that can not be changed, thus that is where -2 comes)

SMTNodeData["BSFVF", 0*SMTNodeData["BSFVF"]]
resets all functional velocity fields

Some useful commands for the manipulation of the velocity fields.

SMTForwardSensitivity

SMTForwardSensitivity[]
calls sensitivity related user subroutines and calculates the solution of sensitivity problem

First the primal problem has to be solved and then sensitivity problem. It is important that the primal problem is solved first since within the solution of the sensitivity problem the decomposed tangent matrix from the last iteration of the iterative solution of the primal problem is used. For time independent problems the `SMTForwardSensitivity` command has to be use only at the point where the results of sensitivity analysis are required. In the case of time dependent problem, the sensitivity is also time dependent, the `SMTForwardSensitivity` command has to be run at the end of each completed time step.

SMTBackwardSensitivity

SMTBackwardSensitivity[*fileName,options*]
calls backward method sensitivity related user subroutines and calculates derivatives of response functionals **F** with respect to sensitivity parameters. The *fileName* is a name, by which the state of simulation was stored during primal analysis (with SMTDumpState[*fileName*,"BackwardSensitivity"→True]). Note: the state of simulation is saved to the files *fileName_i.bst*, where *i* is the successive number of current time step (in case of time independent problems, only the final state of simulation has to be stored).

option	default value	description
"Output"	Default	defines the format of results: Default ⇒ gives the result as gradient vector for first order or Hessian matrix for second order analysis. "Association" ⇒ results are presented as association, where *FunctionalIDs are* used to make keys.
"IntegerInput"	{}	vector of integer constants passed to backward mode user subroutines and accessed by SMSIO["Integer input vector"[i]]
"RealInput"	{}	vector of integer constants passed to backward mode user subroutines and accessed by SMSIO["Real input vector"[i]]

Options of the backward sensitivity method.

The result that is returned by SMTBackwardSensitivity command is represented as:

- for "Output"->Default option:
  - First order: gradient $\left\{\frac{DF}{D\phi_i}\right\}$ or a list of gradients $\{\left\{\frac{DF_1}{D\phi_i}\right\},\left\{\frac{DF_2}{D\phi_i}\right\},...\}$ if more than one functional is defined
  - Second order: Hessian $\left\{\frac{D^2 F}{D\phi_i D\phi_j}\right\}$ or a list of Hessians $\{\left\{\frac{D^2 F_1}{D\phi_i D\phi_j}\right\},\left\{\frac{D^2 F_2}{D\phi_i D\phi_j}\right\},...\}$ if more than one functional is defined
- **for** "Output"→"Association" option an Association in which key a is associated with value of derivative, for example:
  - First order backward sensitivity with two sensitivity parameters (ϕ1 and ϕ2) and two response functionals (F1 and F2) will give e.g. result:
    <|{F1, ϕ1} → 0.3254, {F1, ϕ2} → 800.3, {F2, ϕ1} → 0.8621, {F2, ϕ2} → 53.38|> ...that means that $\frac{DF_1}{D\phi_1}$ = 0.3254, $\frac{DF_1}{D\phi_2}$ = 800.3...
  - Second order backward sensitivity analysis with two sensitivity parameters (ϕ1 and ϕ2) and two response functionals (F1 and F2) will give e.g. result:
    <|{F1, ϕ1, ϕ1} → −5.388, {F1, ϕ1, ϕ2} → 501.3, {F1, ϕ2, ϕ2} → 8077, {F2, ϕ1, ϕ1} → −0.008, {F2, ϕ1, ϕ2} → 212.9, {F2, ϕ2, ϕ2} → 883.3|>
    ...that means that $\frac{D^2 F_1}{D\phi_1 D\phi_1}$ = −5.388, $\frac{D^2 F_1}{D\phi_1 D\phi_2}$ = 501.3...

SMTFindSensitivityParameters

Forward mode only!

SMTFindSensitivityParameters[{sID_i,sID_j,{sID_k,sID_l},...}]
returns an index that corresponds to the given sensitivity parameter identification (first or higher order)

{ i_1, i_2,... } /. SMTFindSensitivityParameters["ID"]
would transform a list of sensitivity parameter indexes into the corresponding list of sensitivity parameter identifications

{sID_i,sID_j,{sID_k,sID_l},...} /. SMTFindSensitivityParameters["Index"]
would transform a list of sensitivity parameter identifications into the corresponding list of sensitivity parameter indexes

SMTFindSensitivityParameters["First order"]
returns all first order sensitivity parameters

SMTFindSensitivityParameters["Second order"]
returns all second order sensitivity parameters

SMTFindVelocityFields

Forward mode only!

SMTFindVelocityFields[{$VelocityFieldID_1$,$VelocityFieldID_2$,...}]
returns an index that corresponds to the given velocity field identification

SMTFindVelocityFields[]
returns a list of velocity field identifications (without Null and Identity fields)

{ i_1, i_2,... } /. SMTFindVelocityFields["ID"]
would transform a list of velocity field indexes into the corresponding list of velocity field identifications

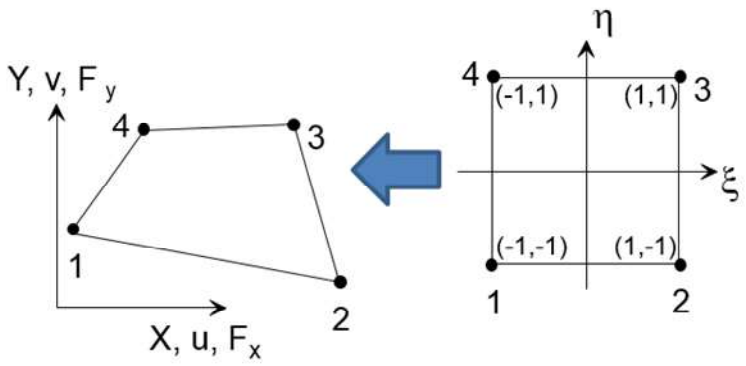
{sID_i,sID_j,{sID_k,sID_l},...} /. SMTFindVelocityFields["Index"]
would transform a list of velocity field identifications into the corresponding list of velocity field indexes

Finite Strain Element for Direct and Sensitivity Analysis

Description of FE

Generate two-dimensional, four node finite element for the analysis of the steady state problems in the mechanics of solids. The element has the following characteristics:

- quadrilateral topology;
- 4 node element;
- isoparametric mapping from the reference to the actual frame;



- global unknowns are displacements of the node;
- thickness of the element is interpolated over the domain as:
  $t(\xi, \eta) = t_0 + N_i(\xi, \eta)\,\Delta t_i$
  where t_0 constant over the domain and variable part is approximated in a standard way $N_i(\xi, \eta)\,\Delta t_i$ where Δt_i is variable part of the thickness in element nodes (Δt_i is stored as an additional nodal data, see SMSNoNodeData) ,
- the element should allow arbitrary large displacements and rotations;
- The problem is defined by the hyper-elastic Neo-Hooke type strain energy potential
  $W = \frac{\lambda}{2}(\det \mathbf{F} - 1)^2 + \mu\left(\frac{\mathrm{Tr}[\mathbf{C}]-3}{2} - \mathrm{Log}[\det \mathbf{F}]\right)$
  and total potential energy of the problem
  $\Pi = \int_{\Omega_0}(W -)\,d\Omega_0 - \int_{\Omega_0}\boldsymbol{\rho_0}\,\mathbf{u}.\bar{\mathbf{b}}\,d\Omega_0 - \sum_i \mathbf{u}_i.\mathbf{P}_i$
  where $\mathbf{C} = \mathbf{F}^T\mathbf{F}$ is right Cauchy-Green tensor, $\mathbf{F} = \mathbf{I} + \nabla\mathbf{u}$ is deformation gradient, $\boldsymbol{u}$ is displacements field, $\bar{\mathbf{b}}$ is force per unit mass, ρ_0 density in initial configuration and $\mathbf{P}_i$ are discrete forces;
- The contribution of the strain energy is incorporated into finite element, while the contribution of the external forces is handled by the SMTAddDictributedBoundary and SMTAddNaturalBoundary commands. Ω_0 is the initial domain of the problem and λ, μ are the first and the second Lame's material constants.

The following general user subroutines have to be generated

- user subroutine for the direct implicit analysis,
- user subroutine for the post-processing that returns the Green-Lagrange strain tensor and, the Cauchy stress tensor.

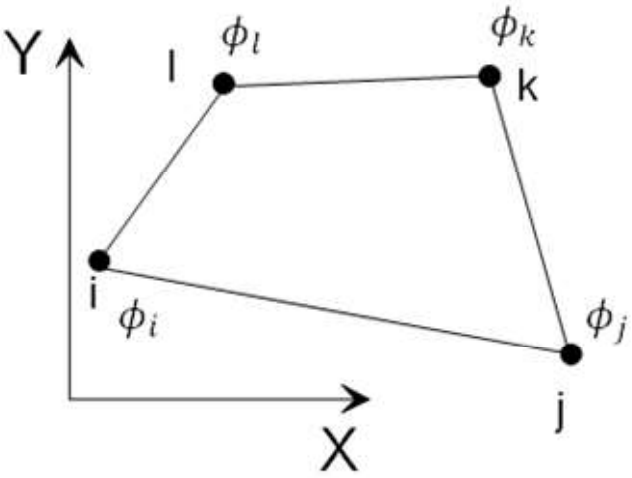
Forward mode sensitivity analysis user subroutines

Forward mode sensitivity analysis user subroutines are general and allow evaluation of sensitivity of an arbitrary response functional with respect to an arbitrary parameters:

- user subroutine for the forward mode first order sensitivity analysis with respect to the material constants, an arbitrary shape parameter and prescribed boundary conditions;
- user subroutine for the the forward mode second order sensitivity analysis with respect to the material constants, an arbitrary shape parameter and prescribed boundary conditions.

Backward mode sensitivity analysis user subroutines

Backward mode sensitivity analysis user subroutines are problem dependent and thus have to be written separately for a specific response functionals and sensitivity parameters. To demonstrate the use of backward sensitivity analysis for steady-state problem we will derive sensitivity analysis user subroutines for a selected set of functionals and sensitivity parameters which are defined as **thicknesses of the domain in each individual node** $\phi_i = t_0 + \Delta t_i$. Number of sensitivity parameters n_ϕ of the problem is therefore always equal to number of nodes. Sensitivity parameters we denote with letter ϕ and corresponding index, e.g. ϕ_1, $\phi_{2,\ldots}$. For the convenience it is chosen that indexes of sensitivity parameters are equivalent to global indexes of nodes $\phi_i = t_0 + \Delta t_i$.



The following user subroutines are derived:

- user subroutine for calculation of finite element contribution to backward sensitivity pseudo-load vector for for selected set of response functionals with respect to sensitivity parameters that define thickness in each node;
- user subroutine for calculation of derivatives with backward sensitivity analysis for selected set of response functionals with respect to sensitivity parameters that define thickness in each node.

The response functionals of interest are:

- 1 - integrated strain energy

$F_1 = F_1^{ext} + F_1^{int}$

$F_1^{ext} = 0$

$F_1^{int} = \int_V W\, dV \approx \sum_e \sum_{g=1}^{n_g} w_g\, W_g\, (\mathbf{p}_e, \boldsymbol{\phi}_e)$ where $\boldsymbol{\phi}_e = (\phi_i, \phi_j, \phi_k, \phi_l)$

$$\tilde{\mathbf{R}} = \mathop{A}_{e} \sum_{g=1}^{n_g} w_g \frac{\partial W}{\partial \mathbf{p}_e}$$

$$\frac{D\hat{F}}{D\phi_i} = \sum_e \sum_{g=1}^{n_g} w_g \begin{cases} \left(\dfrac{\partial W}{\partial \phi_i} - \boldsymbol{\lambda}_e \cdot \dfrac{\partial \mathbf{R}_g}{\partial \phi_i}\right) & i \in \boldsymbol{\phi}_e \\ 0 & i \notin \boldsymbol{\phi}_e \end{cases}$$

$$\frac{D^2\hat{F}}{D\phi_i\, D\phi_j} = \sum_e \sum_{g=1}^{n_g} w_g \left(\left(\frac{\partial^2 W}{\partial \mathbf{p}_e{}^2} \frac{D\mathbf{p}_e}{D\phi_i} \frac{D\mathbf{p}_e}{D\phi_j} + \frac{\partial^2 W}{\partial \mathbf{p}_e\, \partial \phi_i} \frac{D\mathbf{p}_e}{D\phi_j} - \boldsymbol{\lambda}_e \left(\frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e{}^2} \frac{D\mathbf{p}_e}{D\phi_i} \frac{D\mathbf{p}_e}{D\phi_j} + \frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e\, \partial \phi_i} \frac{D\mathbf{p}_e}{D\phi_j} \right) \right) + \right.$$

$$\left. + \begin{cases} \dfrac{\partial^2 W}{\partial \mathbf{p}_e\, \partial \phi_j} \dfrac{D\mathbf{p}_e}{D\phi_i} + \dfrac{\partial^2 W}{\partial \phi_i\, \partial \phi_j} - \boldsymbol{\lambda}_e \left(\dfrac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e\, \partial \phi_j} \dfrac{D\mathbf{p}_e}{D\phi_i} + \dfrac{\partial^2 \mathbf{R}_g}{\partial \phi_i\, \partial \phi_j} \right) & j \in \boldsymbol{\phi}_e \\ 0 & j \notin \boldsymbol{\phi}_e \end{cases} \right)$$

- 2 - Mises stresses to an arbitrary power n

$F_2 = F_2^{ext} + F_2^{int}$

$F_2^{ext} = 0$

$F_2^{int} = \int_V \tilde{\sigma}^n \, dV \approx \sum_e \sum_{g=1}^{n_g} w_g \, \tilde{\sigma}^n (\mathbf{p}_e, \boldsymbol{\phi}_e)$

$\tilde{\mathbf{R}} = \underset{e}{A} \sum_{g=1}^{n_g} w_g \frac{\partial \tilde{\sigma}^n}{\partial \mathbf{p}_e}$

$$\frac{D\hat{F}}{D\phi_i} = \sum_e \sum_{g=1}^{n_g} w_g \begin{cases} \left(\frac{\partial \tilde{\sigma}^n}{\partial \phi_i} - \boldsymbol{\lambda}_e \cdot \frac{\partial \mathbf{R}_g}{\partial \phi_i} \right) & i \in \boldsymbol{\phi}_e \\ 0 & i \notin \boldsymbol{\phi}_e \end{cases}$$

$$\frac{D^2\hat{F}}{D\phi_i D\phi_j} = \sum_e \sum_{g=1}^{n_g} w_g \left(\left(\frac{\partial^2 \tilde{\sigma}^n}{\partial \mathbf{p}_e{}^2} \frac{D\mathbf{p}_e}{D\phi_i} \frac{D\mathbf{p}_e}{D\phi_j} + \frac{\partial^2 \tilde{\sigma}^n}{\partial \mathbf{p}_e \partial \phi_i} \frac{D\mathbf{p}_e}{D\phi_j} - \boldsymbol{\lambda}_e \left(\frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e{}^2} \frac{D\mathbf{p}_e}{D\phi_i} \frac{D\mathbf{p}_e}{D\phi_j} + \frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e \partial \phi_i} \frac{D\mathbf{p}_e}{D\phi_j} \right) \right) + \right.$$

$$\left. + \begin{cases} \frac{\partial^2 \tilde{\sigma}^n}{\partial \mathbf{p}_e \partial \phi_j} \frac{D\mathbf{p}_e}{D\phi_i} + \frac{\partial^2 \tilde{\sigma}^n}{\partial \phi_i \partial \phi_j} - \boldsymbol{\lambda}_e \left(\frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e \partial \phi_j} \frac{D\mathbf{p}_e}{D\phi_i} + \frac{\partial^2 \mathbf{R}_g}{\partial \phi_i \partial \phi_j} \right) & j \in \boldsymbol{\phi}_e \\ 0 & j \notin \boldsymbol{\phi}_e \end{cases} \right)$$

- 3 - displacement v in nv-th node

$F_3 = F_3^{ext} + F_3^{int}$

$F_3^{ext} = v_{nv}$

$F_3^{int} = 0$

$\tilde{\mathbf{R}} = \frac{\partial F^{ext}}{\partial \mathbf{p}} = \{\delta_{i\,nv} : i = 1, \ldots, n_{nodes}\}$

$$\frac{D\hat{F}}{D\phi_i} = \sum_e \sum_{g=1}^{n_g} w_g \begin{cases} -\boldsymbol{\lambda}_e \cdot \frac{\partial \mathbf{R}_g}{\partial \phi_i} & i \in \boldsymbol{\phi}_e \\ 0 & i \notin \boldsymbol{\phi}_e \end{cases}$$

$$\frac{D^2\hat{F}}{D\phi_i D\phi_j} = \sum_e \sum_{g=1}^{n_g} w_g \left(-\boldsymbol{\lambda}_e \left(\frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e{}^2} \frac{D\mathbf{p}_e}{D\phi_i} \frac{D\mathbf{p}_e}{D\phi_j} + \frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e \partial \phi_i} \frac{D\mathbf{p}_e}{D\phi_j} \right) + \begin{cases} -\boldsymbol{\lambda}_e \left(\frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e \partial \phi_j} \frac{D\mathbf{p}_e}{D\phi_i} + \frac{\partial^2 \mathbf{R}_g}{\partial \phi_i \partial \phi_j} \right) & j \in \boldsymbol{\phi}_e \\ 0 & j \notin \boldsymbol{\phi}_e \end{cases} \right)$$

- 4 - L2 norm of displacement vector

$F_4 = F_4^{ext} + F_4^{int}$

$F_4^{ext} = \|\mathbf{p}\|_2 = \sqrt{\mathbf{p}.\mathbf{p}}$

$F_4^{int} = 0$

$\tilde{\mathbf{R}} = \frac{\partial F^{ext}}{\partial \mathbf{p}} = \left\{ \frac{p_i}{\sqrt{\mathbf{p}.\mathbf{p}}} : i = 1, \ldots, n_{nodes} \right\}$

$$\frac{D\hat{F}}{D\phi_i} = \sum_e \sum_{g=1}^{n_g} w_g \begin{cases} -\boldsymbol{\lambda}_e \cdot \frac{\partial \mathbf{R}_g}{\partial \phi_i} & i \in \boldsymbol{\phi}_e \\ 0 & i \notin \boldsymbol{\phi}_e \end{cases}$$

$$\frac{D^2\hat{F}}{D\phi_i D\phi_j} = \sum_e \sum_{g=1}^{n_g} w_g \left(-\boldsymbol{\lambda}_e \left(\frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e{}^2} \frac{D\mathbf{p}_e}{D\phi_i} \frac{D\mathbf{p}_e}{D\phi_j} + \frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e \partial \phi_i} \frac{D\mathbf{p}_e}{D\phi_j} \right) + \begin{cases} -\boldsymbol{\lambda}_e \left(\frac{\partial^2 \mathbf{R}_g}{\partial \mathbf{p}_e \partial \phi_j} \frac{D\mathbf{p}_e}{D\phi_i} + \frac{\partial^2 \mathbf{R}_g}{\partial \phi_i \partial \phi_j} \right) & j \in \boldsymbol{\phi}_e \\ 0 & j \notin \boldsymbol{\phi}_e \end{cases} \right)$$

User subroutine Task performs the following tasks

- 1 - "Volume and sensitivity"
  - returns $\int_{\Omega_e} \text{Flatten}\left[\left\{V, \frac{\partial V}{\partial \boldsymbol{\phi}}, \text{vec}\left(\frac{\partial^2 V}{\partial \boldsymbol{\phi}^2}\right)\right\}\right] dV$. Forward mode sensitivity $\frac{\partial^2 V}{\partial \boldsymbol{\phi}^2}$ is symmetric, thus only upper triangular matrix is returned.
- 2 - "Integrated stress and sensitivity (P, plane strain)"
  - returns integral of components of stress tensor $\text{vec}(\boldsymbol{P}) = \{P_{11}, P_{12}, P_{21}, P_{22}, P_{33}\}$ over domain of the problem and forward mode sensitivity of integrated stress with respect to all sensitivity parameters $\int_{\Omega_e} \text{Flatten}\left[\left\{\text{vec}(P), \frac{\partial \text{vec}(P)}{\partial \boldsymbol{\phi}}\right\}\right] dV$.
  - If sensitivity parameters $\boldsymbol{\phi}$ are components of deformation gradient $\boldsymbol{\phi} = \text{vec}(F_M)$ then returns $\int_{\Omega_e} \text{Flatten}\left[\left\{\text{vec}(P), \frac{\partial \text{vec}(P)}{\partial \text{vec}(F_M)}\right\}\right] dV$. The term $\int_{\Omega_e} \frac{\partial \text{vec}(P)}{\partial \text{vec}(F_M)} dV$ represents homogenized constitutive matrix for FE^2 multi-scale simulations.
- 3 - "Integrated strain energy and derivatives (plane strain)"

- returns integral of strain energy over domain of the problem and first and second order forward mode derivatives of strain energy with respect to all sensitivity parameters $\int_{\Omega_e} \text{Flatten}\left[\left\{W, \frac{\partial \mathbf{W}}{\partial \boldsymbol{\phi}}, \text{vec}\left(\frac{\partial^2 \mathbf{W}}{\partial \boldsymbol{\phi}^2}\right)\right\}\right] dV$.
- If sensitivity parameters are components of nodal degrees of freedom $\boldsymbol{\phi} = \text{vec}(\mathbf{u}_M)$ of macro element then term $\int_{\Omega_e} \frac{\partial \mathbf{W}}{\partial \mathbf{vec}(\mathbf{u}_M)} dV$ represents contribution of the micro element to macro residual and $\int_{\Omega_e} \frac{\partial^2 \mathbf{W}}{\partial \mathbf{vec}(\mathbf{u}_M)^2} dV$ contribution of the micro element to condensed tangent matrix for sub-structuring multi-scale simulations. $\frac{\partial^2 \mathbf{W}}{\partial \mathbf{vec}(\mathbf{u}_M)^2}$ is symmetric, thus only upper triangular matrix is returned.

- 4 - "Reset sensitivity data"
  - All sensitivity related data is set to 0.
- 5 - "Mises stress"
  - Mises stress is calculated using Cauchy stress tensor $\boldsymbol{\sigma}$.
- 6 - "Integrated Mises^n and forward mode sensitivity"
  - returns $\int_V \text{Flatten}\left[\left\{\tilde{\sigma}^n, \frac{\partial \tilde{\sigma}^n}{\partial \boldsymbol{\phi}}, \text{vec}\left(\frac{\partial^2 \tilde{\sigma}^n}{\partial \boldsymbol{\phi}^2}\right)\right\}\right] dV$. $\frac{\partial^2 \tilde{\sigma}^n}{\partial \boldsymbol{\phi}^2}$ is symmetric, thus only upper part is returned.

Direct analysis user subroutines

In[1]:=
```
<< "AceGen`";
SMSInitialize["ExamplesSensitivity2D", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "Q1", "SMSSymmetricTangent" → True,
  "SMSDomainDataNames" -> {"E -elastic modulus", "ν -poisson ratio", "t -thickness"},
  "SMSShapeSensitivity" -> True, "SMSEBCSensitivity" -> True,
  "SMSDefaultData" -> {21000, 0.3, 1},
  "SMSNoNodeData" → {1, 1, 1, 1},
  "SMSCharSwitch" →
   {"Volume and sensitivity", "Integrated stress and sensitivity (P, plane strain)",
    "Integrated strain energy and derivatives (plane strain)",
    "Reset sensitivity data", "Mises stress", "Integrated Mises^n and sensitivity"}
];
SMSSensitivityNames =
  Join[{"X -spatial coordinate", "Y -spatial coordinate"}, SMSDomainDataNames];
```

Definitions of geometry, kinematics, strain energy ...

In[5]:=
```
ElementDefinitions[] := Block[{},
  Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
  {XIO, uIO} ⊨ SMSIO["All coordinates and DOFs"];
  Nh ⊨ 1 / 4 {(1 - ξ) (1 - η), (1 + ξ) (1 - η), (1 + ξ) (1 + η), (1 - ξ) (1 + η)};
  SMSFreeze[X, Append[Nh.XIO, ζ]];
  Je ⊨ SMSD[X, Ξ];
  Jed ⊨ Det[Je];
  u ⊨ Append[Nh.uIO, 0];
  H ⊨ SMSD[u, X, "Dependency" → {Ξ, X, SMSInverse[Je]}];
  SMSFreeze[F, IdentityMatrix[3] + H, "Ignore" → PossibleZeroQ ];
  JF ⊨ Det[F]; Ct ⊨ Transpose[F].F;
  {Em, ν, tζ0} ⊨ SMSIO["Domain data"];
  {λ, μ} ⊨ SMSHookeToLame[Em, ν];
  W ⊨ 1 / 2 λ (JF - 1) ^ 2 + μ (1 / 2 (Tr[Ct] - 3) - Log[JF]);
  wgp ⊨ SMSIO["Integration weight"[Ig]];
  pe = Flatten[uIO];
  P ⊨ SMSD[W, F, "Ignore" -> NumberQ];
  σ ⊨ (1 / JF) * P . Transpose[F];
  s ⊨ σ - 1/3 IdentityMatrix[3] * Tr[σ];
  Mises ⊨ SMSSqrt[(3 / 2) Total[s s, 2]];
  Δtζi ⊨ SMSIO["Nodal data"][[All, 1]];
  tζ ⊨ tζ0 + Nh.Δtζi;
  ΔV ⊨ Jed tζ;
 ]
```

"Tangent and residual" user subroutine

In[6]:=
```
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
SMSDo[
  Rg ⊨ ΔV SMSD[W, pe, i];
  SMSIO[wgp Rg, "Add to", "Residual"[i]];
  SMSDo[
   Kg ⊨ SMSD[Rg, pe, j];
   SMSIO[wgp Kg, "Add to", "Tangent"[i, j]];
   , {j, i, SMSNoDOFGlobal}];
  , {i, 1, SMSNoDOFGlobal}];
SMSEndDo[];
```

"Postprocessing" user subroutine

In[11]:=
```
SMSStandardModule["Postprocessing"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
SMSIO[{"DeformedMeshX" -> uIO[[All, 1]], "DeformedMeshY" -> uIO[[All, 2]],
    "u" -> uIO[[All, 1]], "v" -> uIO[[All, 2]]}, "Export to", "Nodal point post"];
Eg ⊨ 1 / 2 (Ct - IdentityMatrix[3]);
SMSIO[{"Exx" -> Eg[[1, 1]], "Exy" -> Eg[[1, 2]], "Eyy" -> Eg[[2, 2]], "Sxx" → σ[[1, 1]], "Sxy" → σ[[1, 2]],
    "Syy" → σ[[2, 2]], "Szz" → σ[[3, 3]]}, "Export to", "Integration point post"[Ig]];
SMSEndDo[];
```

Forward mode sensitivity user subroutines

"Forward mode first order pseudo-load" user subroutine

```
In[261]:= SMSStandardModule["Forward mode first order pseudo-load"];
          SMSDo[Ig, 1, SMSIO["No. integration points"]];

In[263]:= ElementDefinitions[];
          Rg ⊨ ΔV SMSD[W, pe];
          SMSDo[is, SMSIO["SensIndexStart"], SMSIO["SensIndexEnd"]];
          ϕis ⊢ SMSFictive[];
          "add first order shape velocity field ∂X/∂ϕi to coordinates";
          SMSDefineDerivative[SMSIO["Nodal coordinates"], ϕis, SMSIO["Shape velocity field"[is]]];
          "add parameter velocity field ∂d/∂ϕi to all parmeters
             defined by SMSDomainDataNames, ∂d/∂ϕi is interpolated over the element";
          SMSDefineDerivative[SMSIO["Domain data"], ϕis, Nh.SMSIO["Parameter velocity field"[is]]];
          "add first order essential boundary conditions velocity field ∂pe/∂ϕi to nodal DOFs";
          SMSDefineDerivative[SMSIO["Nodal DOFs"], ϕis, SMSIO["Sensitivity DOFs"[is]]];
          Rtg ⊨ SMSD[Rg, ϕis, "Mode" → "Backward"];
          SMSIO[wgp Rtg, "Add to", "First order pseudo load"[is]];
          SMSEndDo[];

In[276]:= SMSEndDo[];
```

"Forward mode second order pseudo-load" user subroutine

```
In[277]:= SMSStandardModule["Forward mode second order pseudo-load"];
          SMSDo[Ig, 1, SMSIO["No. integration points"]];

In[279]:= ElementDefinitions[];
          Rg ⊨ Jed tξ SMSD[W, pe];

In[281]:= SMSDo[is, SMSIO["SensRowStart"], SMSIO["SensRowEnd"]];
          ϕis ⊢ SMSFictive[];
          "add first order shape velocity field ∂X/∂ϕi to coordinates";
          SMSDefineDerivative[SMSIO["Nodal coordinates"], ϕis, SMSIO["Shape velocity field"[is]]];
          "add first order parameter velocity field ∂d/∂ϕi to all parmeters
             defined by SMSDomainDataNames, ∂d/∂ϕi is interpolated over the element";
          SMSDefineDerivative[SMSIO["Domain data"], ϕis, Nh.SMSIO["Parameter velocity field"[is]]];
          "add first order first order sensitivities ∂pe/∂ϕi to nodal DOFs";
          SMSDefineDerivative[SMSIO["Nodal DOFs"], ϕis, SMSIO["Sensitivity DOFs"[is]]];

In[289]:= SMSDo[js, is, SMSIO["NoSensParameters"]];
          ϕjs ⊢ SMSFictive[];
          "add first order shape velocity field ∂X/∂ϕj to coordinates";
```

```
In[292]:= SMSDefineDerivative[SMSIO["Nodal coordinates"], ϕjs, SMSIO["Shape velocity field"[js]]];
          "add second order shape velocity field ∂(∂X/∂ϕi)/∂ϕj to first order shape velocity field";
          SMSDefineDerivative[SMSIO["Shape velocity field"[is]],
            ϕjs, SMSIO["Shape velocity field"[is, js]]];
          "add first order parameter velocity field ∂d/∂ϕj to all parmeters defined
             by SMSDomainDataNames, ∂d/∂ϕj is interpolated over the element";
          SMSDefineDerivative[SMSIO["Domain data"], ϕjs, Nh.SMSIO["Parameter velocity field"[js]]];
          "add second order parameter velocity
             field ∂(∂d/∂ϕi)/∂ϕj to first order parameter velocity field";
          SMSDefineDerivative[SMSIO["Parameter velocity field"[is]], ϕjs,
            SMSIO["Parameter velocity field"[is, js]]];
          "add first order first order sensitivities ∂pe/∂ϕj to nodal DOFs";
          SMSDefineDerivative[SMSIO["Nodal DOFs"], ϕjs, SMSIO["Sensitivity DOFs"[js]]];
          "add second order essential boundary conditions
             velocity field ∂(∂pe/∂ϕi)/∂ϕj to first order sensitivityes";
          SMSDefineDerivative[SMSIO["Sensitivity DOFs"[is]], ϕjs, SMSIO["Sensitivity DOFs"[is, js]]];
          Rtg ⊨ SMSD[SMSD[Rg, ϕis, "Mode" → "Forward"], ϕjs, "Mode" → "Backward"];
          SMSIO[wgp Rtg, "Add to", "Second order pseudo load"[is, js]];

In[305]:= SMSEndDo[];(*js*)
          SMSEndDo[];(*is*)

In[307]:= SMSEndDo[];(*Ig*)
```

Backward mode sensitivity user subroutines

Backward mode sensitivity user subroutines are based on the following assumptions:

- Subroutines support all four described functionals, however user can at any time calculate only derivatives of one selected functional, thus $i_F = 1$.

"Backward mode pseudo-load" user subroutine

```
In[308]:= SMSStandardModule["Backward mode pseudo-load"];
          functionalType ⊨ SMSIO["Integer input vector"[1]];
          SMSIf[functionalType == 3 || functionalType == 4 ,
            (*Backward sensitivity pseudo-load load is given as functional velocity field*)
            SMSReturn[];
           ];
          SMSDo[Ig, 1, SMSIO["No. integration points"]];
          ElementDefinitions[];
          SMSSwitch[functionalType
            (*integral of strain energy*)
            , 1, F ⊨ ΔV W;
            SMSIO[-SMSD[F, pe], "Add to", "Adjoint pseudo-load"[1]];
            (*integral of Mises stress^r*)
            , 2, n ⊨ SMSIO["Real input vector"[1]];
            F ⊨ ΔV Mises^n;
            SMSIO[-SMSD[F, pe], "Add to", "Adjoint pseudo-load"[1]];
           ];
          SMSEndDo[];
```

"Backward mode first order derivatives" user subroutine

In[315]:=
```
SMSStandardModule["Backward mode first order derivatives"];
functionalType ⊨ SMSIO["Integer input vector"[1]];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
Rg ⊨ ∆V SMSD[W, pe];
λadj ⊨ Flatten[SMSIO["Adjoint vector DOFs"[1]]];
nodes ⊢ SMSInteger[Table[nd$$[i, "id", "NodeIndex"] + 1, {i, SMSNoNodes}]];

SMSDo[kn, 1, SMSNoNodes];
is ⊢ SMSPart[nodes, kn];
ϕis ⊢ SMSFictive[];
SMSDefineDerivative[tξ, ϕis, SMSPart[Nh, kn]];
DRgDϕi ⊨ SMSD[Rg, ϕis, "Mode" → "Forward"];
SMSIO[-wgp DRgDϕi.λadj, "Add to", "Functional derivative"[1, is]];
SMSSwitch[functionalType
  (*integral of strain energy*)
  , 1, F ⊨ ∆V W;
  SMSIO[wgp SMSD[F, ϕis], "Add to", "Functional derivative"[1, is]];
  (*integral of Mises stress^r*)
  , 2, n ⊨ SMSIO["Real input vector"[1]];
  F ⊨ ∆V Mises^n;
  SMSIO[wgp SMSD[F, ϕis], "Add to", "Functional derivative"[1, is]];
 ];
SMSEndDo[];

SMSEndDo[];
```

"Backward mode second order derivatives" user subroutine

In[331]:=
```
SMSStandardModule["Backward mode second order derivatives"];
```

In[332]:=
```
functionalType ⊨ SMSIO["Integer input vector"[1]];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
Rg ⊨ ΔV SMSD[W, pe];
λadj ⊨ Flatten[SMSIO["Adjoint vector DOFs"[1]]];
nodes ⊢ SMSInteger[Table[nd$$[i, "id", "NodeIndex"] + 1, {i, SMSNoNodes}]];

SMSDo[is, 1, SMSIO["NoSensParameters"]];
ϕis ⊢ SMSFictive[];
SMSDefineDerivative[SMSIO["Domain data"][[3]],
 ϕis, Nh.SMSIO["Parameter velocity field"[is][All, 3]]];
SMSDefineDerivative[SMSIO["Nodal DOFs"], ϕis, SMSIO["Sensitivity DOFs"[is]]];
F ⊨ SMSSwitch[functionalType
    , 1, ΔV W
    , 2, ΔV Mises^SMSIO["Real input vector"[1]]
    , _, 0
    , "Inline" → True];
G ⊨ F - λadj.Rg;
DGDϕi ⊨ SMSD[G, ϕis, "Mode" → "Forward"];

SMSDo[js, is, SMSIO["NoSensParameters"]];
ϕjall ⊢ SMSFictive[];
SMSDefineDerivative[SMSIO["Nodal DOFs"], ϕjall, SMSIO["Sensitivity DOFs"[js]]];
DGDϕiϕja ⊨ SMSD[DGDϕi, ϕjall, "Mode" → "Backward"];
SMSIO[wgp * DGDϕiϕja, "Add to", "Functional derivative"[1, is, js]];
SMSIf[(js == nodes[[1]] || js == nodes[[2]] || js == nodes[[3]] || js == nodes[[4]])];
ϕjlocal ⊢ SMSFictive[];
SMSDefineDerivative[SMSIO["Domain data"][[3]],
 ϕjlocal, Nh.SMSIO["Parameter velocity field"[js][All, 3]]];
DGDϕiϕjl ⊨ SMSD[DGDϕi, ϕjlocal, "Mode" → "Backward"];
SMSIO[wgp * DGDϕiϕjl, "Add to", "Functional derivative"[1, is, js]];
SMSEndIf[];
SMSEndDo[];

SMSEndDo[];

SMSEndDo[];
```

Tasks user subroutine

In[355]:=
```
SMSStandardModule["Tasks"];
```

In[356]:=
```
task ⊨ SMSIO["Task index"];
(*noSensDerivatives=noSensParameters + size of upper triangle of second order derivatives *)
NoIp ⊨ SMSIO["No. integration points"];
StessComponents = {{1, 1}, {1, 2}, {2, 1}, {2, 2}, {3, 3}};
NoStressComponents = Length[StessComponents];
```

In[360]:=
```
SMSIf[task < 0
  , SMSSwitch[task
   , -1,
   SMSIO[{1, 0, 0, 0, 1 + SMSIO["NoSensDerivatives"]}, "Export to", "Task data"];
   , -2,
   SMSIO[{1, 0, 0, 0, NoStressComponents + NoStressComponents SMSIO["NoSensParameters"]},
     "Export to", "Task data"];
   , -3,
   SMSIO[{1, 0, 0, 0, 1 + SMSIO["NoSensDerivatives"]}, "Export to", "Task data"];
   , -4,
   SMSIO[{1, 0, 0, 0, 0}, "Export to", "Task data"];
   , -5,
   SMSIO[{3, 0, 0, 0, NoIp}, "Export to", "Task data"];
   , -6,
   SMSIO[{1, 0, 1, 0, 1 + SMSIO["NoSensDerivatives"]}, "Export to", "Task data"];
  ];
  SMSReturn[];
 ];
```

In[361]:=
```
SMSIf[task == 4,
  SMSDo[
   SMSExport[Table[0, {i, SMSNoNodes}, {j, SMSDOFGlobal[[i]]}],
    Table[nd$$[i, "st", is, j], {i, SMSNoNodes}, {j, SMSDOFGlobal[[i]]}]];
   SMSExport[Table[0, {i, SMSNoNodes}, {j, SMSDOFGlobal[[i]]}], Table[nd$$[i, "sp", is, j],
     {i, SMSNoNodes}, {j, SMSDOFGlobal[[i]]}]];, {is, 1, SMSIO["NoSensDerivatives"]}];
  SMSReturn[];
 ];
```

In[362]:=
```
SMSDo[Ig, 1, NoIp];
ElementDefinitions[];
"primal anaysis quantities";
Wg ⊨ ΔV W;
PI ⊨ ΔV Extract[P, StessComponents];
IntMises ⊨ ΔV Mises^SMSIO["Task real input"[1]];
```

In[368]:=
```
SMSSwitch[task
  , 1, SMSIO[ wgp ΔV, "Add to", "Task real output"[1]];
  , 2, SMSIO[ wgp PI, "Add to", "Task real output"];
  , 3, SMSIO[wgp Wg, "Add to", "Task real output"[1]];
  , 5, SMSIO[Mises, "Export to", "Task real output"[Ig]];
  , 6, SMSIO[ wgp IntMises, "Add to", "Task real output"[1]];
 ];
```

Here it is assumed that sensitivity analysis has been performed in forward mode, thus all sensitivities of all unknowns are known.

In[369]:=
```
SMSDo[is, 1, SMSIO["NoSensParameters"]];
ϕis ⊢ SMSFictive[];
SMSDefineDerivative[SMSIO["Nodal coordinates"], ϕis, SMSIO["Shape velocity field"[is]]];
SMSDefineDerivative[SMSIO["Domain data"], ϕis, Nh.SMSIO["Parameter velocity field"[is]]];
SMSDefineDerivative[SMSIO["Nodal DOFs"], ϕis, SMSIO["Sensitivity DOFs"[is]]];
```

In[374]:=
```
"first order sensitivity anaysis quantities";
{DΔVDϕi, DPDϕi, DWDϕi, DMisesDϕi} = SMSD[{ΔV, PI, Wg, IntMises}, ϕis];
SMSSwitch[task
  , 1, SMSIO[wgp DΔVDϕi, "Add to", "Task real output"[1 + is]];
  , 2, SMSIO[wgp DPDϕi, "Add to", "Task real output"[
     Table[NoStressComponents + (is - 1) NoStressComponents + i, {i, NoStressComponents}]]];
  , 3, SMSIO[wgp DWDϕi, "Add to", "Task real output"[1 + is]];
  , 6, SMSIO[wgp DMisesDϕi, "Add to", "Task real output"[1 + is]];
 ];
```

In[377]:=
```
SMSDo[js, is, SMSIf[SMSIO["NoSecondOrderDerivatives"] == 0, 0, SMSIO["NoSensParameters"]]];
ϕjs ⊢ SMSFictive[];
```

In[379]:=
```
SMSDefineDerivative[SMSIO["Nodal coordinates"], ϕjs, SMSIO["Shape velocity field"[js]]];
SMSDefineDerivative[SMSIO["Shape velocity field"[is]],
  ϕjs, SMSIO["Shape velocity field"[is, js]]];
SMSDefineDerivative[SMSIO["Domain data"], ϕjs, Nh.SMSIO["Parameter velocity field"[js]]];
SMSDefineDerivative[SMSIO["Parameter velocity field"[is]],
  ϕjs, SMSIO["Parameter velocity field"[is, js]]];
SMSDefineDerivative[SMSIO["Nodal DOFs"], ϕjs, SMSIO["Sensitivity DOFs"[js]]];
SMSDefineDerivative[SMSIO["Sensitivity DOFs"[is]], ϕjs, SMSIO["Sensitivity DOFs"[is, js]]];
```

In[385]:=
```
"second order sensitivity anaysis quantities, upper tringle only";
ijs ⊨ SMSInteger[1 + is + (is - 1) SMSIO["NoSensParameters"] - (is - 1) is / 2 + js];
SMSSwitch[task
  , 1,
  DΔVDϕiϕj = SMSD[DΔVDϕi, ϕjs];
  SMSIO[ wgp DΔVDϕiϕj, "Add to", "Task real output"[ijs]];
  , 3,
  DWDϕiϕj ⊨ wgp SMSD[DWDϕi, ϕjs];
  SMSIO[DWDϕiϕj, "Add to", "Task real output"[ijs]];
  , 6,
  DMisesDϕiϕj ⊨ SMSD[DMisesDϕi, ϕjs];
  SMSIO[ wgp DMisesDϕiϕj, "Add to", "Task real output"[ijs]];
 ];
```

In[388]:=
```
SMSEndDo[];(*sensitivity j*)
SMSEndDo[];(*sensitivity i*)
```

In[390]:=
```
SMSEndDo[];(*integration points*)
```

Code generation

In[391]:=
```
SMSWrite[];
```

```
[136]  Consistency check - expressions
```

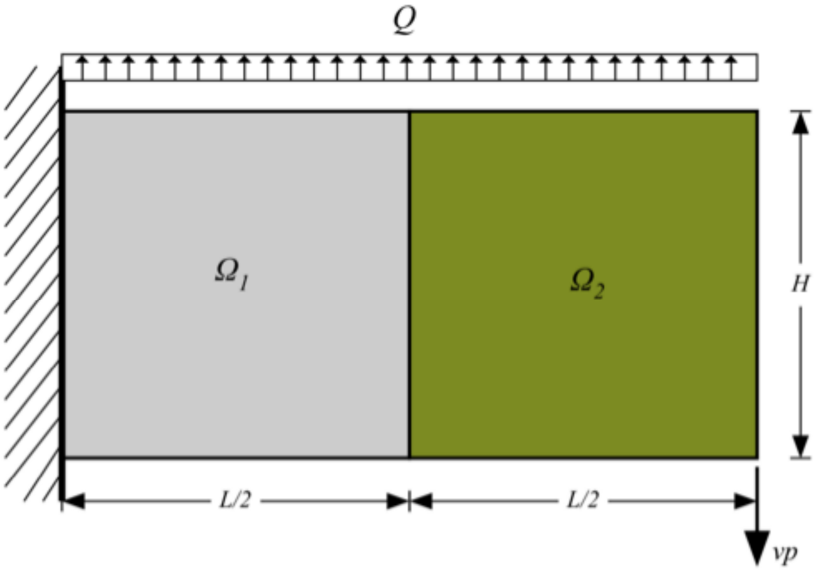
File: ExamplesSensitivity2D.c **Size:** 140 291 **Time:** 139

Method	SKR	SPP	SSE	SS2	SBL	SBG	SBH	Tasks
No.Formulae	100	74	557	905	125	106	259	547
No.Leafs	1519	1155	8580	18 002	1851	1539	4613	11 620

General Forward Mode Sensitivity Analysis Example

Description of forward mode example

With the use of the element "ExamplesSensitivity2D" generated in previous section analyze the following example composed of two-domains Ω_1 and Ω_2.



The first and second order sensitivity of the displacement field **u** = {u, v} with respect to the following parameters has to be calculated with respect to:

$\phi_1 = E_1$ ⇒ E_1 is elastic modulus on domain Ω_1 ,

$\phi_2 = t$ ⇒ thickness t on all domains,

$\phi_3 = L$ ⇒ length L,

$\phi_4 = Q$ ⇒ distributed force Q,

$\phi_5 = v_p$ ⇒ prescribed displacement v_p.

First order forward mode sensitivity analysis

In[191]:=
```
<< AceFEM`;
SMTInputData[];
L = 10; Q = 50; vp = 1; H = L / 2; ne = 20;
SMTAddDomain[
  {"Ω1", "ExamplesSensitivity2D", {"E *" -> 1000, "ν *" -> 0.3, "t *" -> 1}},
  {"Ω2", "ExamplesSensitivity2D", {"E *" -> 5000, "ν *" -> 0.2, "t *" -> 1}}
 ];
SMTAddEssentialBoundary[Line[{{0, 0}, {0, H}}], 1 -> 0, 2 → 0];
SMTAddNaturalBoundary[ Line[{{0, H}, {L, H}}], 2 → Line[{Q}]];
SMTAddEssentialBoundary[ Point[{L, 0}], 2 → -vp];
SMTAddMesh[Polygon[{{0, 0}, {L / 2, 0}, {L / 2, H}, {0, H}}], "Ω1", "Q1", {ne, ne}];
SMTAddMesh[Polygon[{{L / 2, 0}, {L, 0}, {L, H}, {L / 2, H}}], "Ω2", "Q1", {ne, ne}];
```

In[200]:=
```
SMTSensitivityProblem[{
    (* E is the third general parameter on "Ω1" domain *)
    {"E", {"P", (*∂E/∂E=1*)Identity, "Ω1" → 3(*E*)}},
    (* t is the fifth general parameter on all domains*)
    {"t", {"P", (*∂t/∂t=1*)Identity, All → 5(*t*)}},
    (* L is the parameter on which coordinates
     X and Y depends and also the nodal forces in Y direction*)
    {"L", {"S", "∂X/∂L", All -> (*X*)1}, {"S", "∂Y/∂L", All -> (*Y*)2},
     {"NBC", "∂Fy/∂L", "D" → 2}},
    (* Q is natural boundary condition parameter*)
    {"Q", {"NBC", "∂Fy/∂Q", "D" → 2}},
    (* vp is essential boundary condition parameter *)
    {"vp", {"EBC", "∂v/∂vp", "D" → 2}}
   }];
```

In[201]:=
```
SMTAnalysis[];
```

- This sets velocity fields

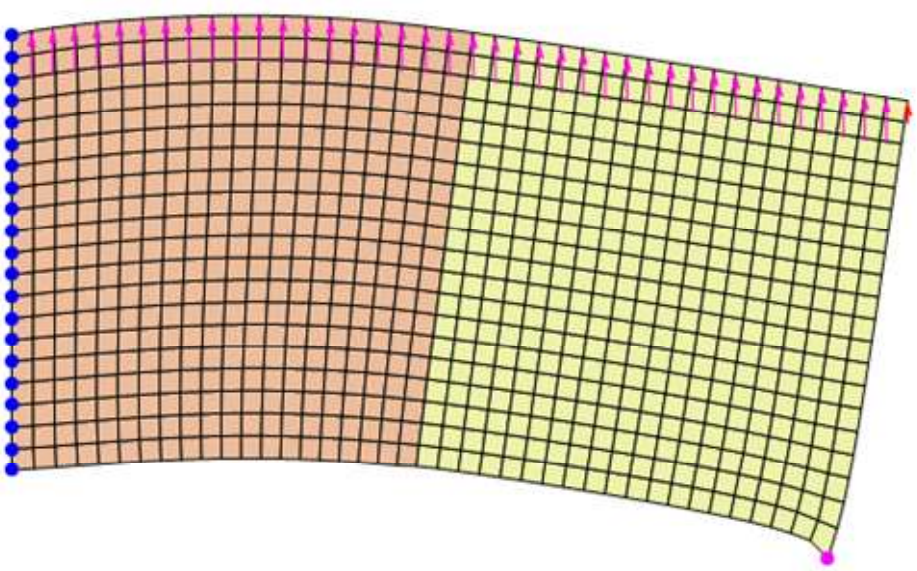
In[202]:=
```
SMTSetVelocityFields[{
    "∂X/∂L" → {All, Function[{n, X, Y}, X / L]},
    "∂Y/∂L" → {All, Function[{n, X, Y}, Y / L]},
    "∂Fy/∂L" → {Line[{{0, H}, {L, H}}], Function[{n, X, Y}, Q / (2 ne)]},
    "∂Fy/∂L" → {Point[{0, H}] || Point[{L, H}], Function[{n, X, Y}, Q / (4 ne)]},
    "∂Fy/∂Q" → {Line[{{0, H}, {L, H}}], Function[{n, X, Y}, L / (2 ne)]},
    "∂Fy/∂Q" → {Point[{0, H}] || Point[{L, H}], Function[{n, X, Y}, L / (4 ne)]},
    "∂v/∂vp" → {Point[{L, 0}], {-1}}
   }];
```

- Here is the primal analysis executed.

In[203]:=
```
SMTNextStep["λ" → 1];
While[SMTConvergence[10^-9, 10], SMTNewtonIteration[]];
SMTStatusReport[];
```

```
Step/Iter=1/6 λ/Δλ=1./1. ||Δp|| / ||R|| =
 1.93709×10^-13/5.33088×10^-11 Events=0 Status=0/{Convergence}
```

In[206]:=
```
SMTShowMesh["BoundaryConditions" → True, "DeformedMesh" → True]
```

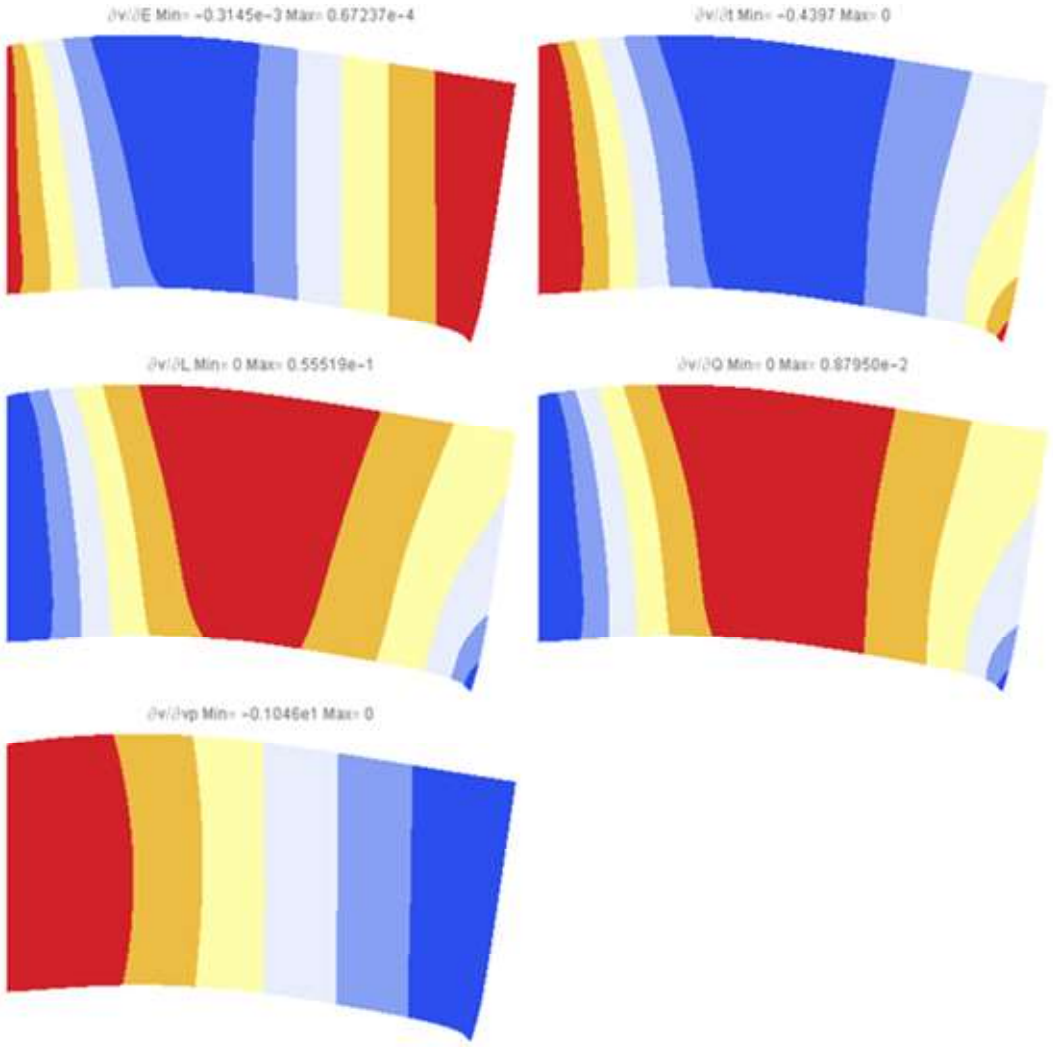


- This performs the sensitivity analysis with respect to all parameters.

In[207]:=
```
SMTForwardSensitivity[];
```

Visualization and post-processing

In[231]:=
```
ImageCollage[
 Table[SMTShowMesh["DeformedMesh" → True, "Mesh" → False, "Field" → SMTPostData[{"st", 2, ϕ}],
   "Legend" → False, "Contour" → 5, "Label" → {"∂v/∂", ϕ, " ", Automatic}]
  , {ϕ, {"E", "t", "L", "Q", "vp"}}], Background → White]
```



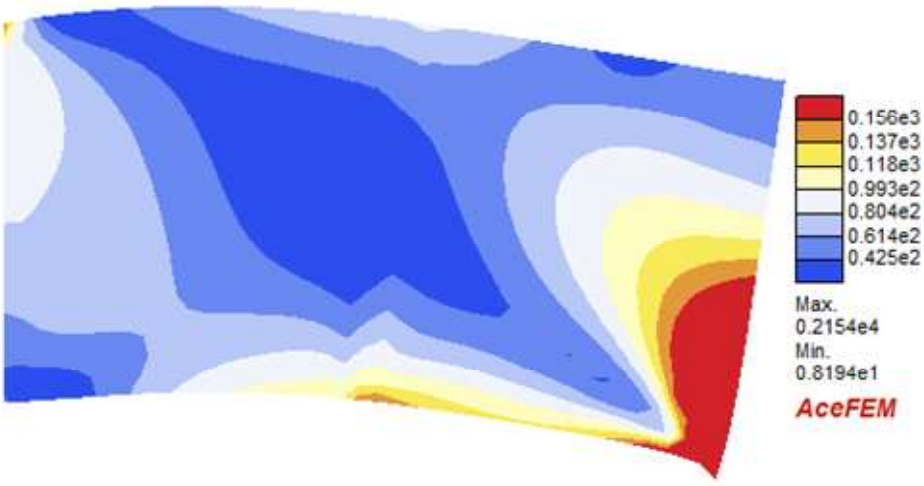
- This evaluates the volume of the mesh and sensitivity of the volume with respect to all parameters: {V, $\frac{DV}{DE}$, $\frac{DV}{Dt}$, $\frac{DV}{DL}$, $\frac{DV}{DQ}$, $\frac{DV}{Dv_p}$}. As expected, only the thickness t and length L effect the volume.

In[209]:=
```
SMTTask["Volume and sensitivity"]
```

Out[209]=
```
{50., 0., 50., 10., 0., 0.}
```

- This shows the distribution of the Misses stress.

In[210]:=
```
SMTShowMesh["DeformedMesh" → True, "Mesh" → False,
 "Field" → SMTTask["Mises stress"], "Contour" → True]
```



Second order forward mode sensitivity analysis

In[232]:=
```
<< AceFEM`;
SMTInputData[];
L = 10; Q = 50; vp = 1; H = L / 2; ne = 20;
SMTAddDomain[
  {"Ω1", "ExamplesSensitivity2D", {"E *" -> 1000, "ν *" -> 0.3, "t *" -> 1}},
  {"Ω2", "ExamplesSensitivity2D", {"E *" -> 5000, "ν *" -> 0.2, "t *" -> 1}}
 ];
SMTAddEssentialBoundary[Line[{{0, 0}, {0, H}}], 1 -> 0, 2 → 0];
SMTAddNaturalBoundary[ Line[{{0, H}, {L, H}}], 2 → Line[{Q}]];
SMTAddEssentialBoundary[ Point[{L, 0}], 2 → -vp];
SMTAddMesh[Polygon[{{0, 0}, {L / 2, 0}, {L / 2, H}, {0, H}}], "Ω1", "Q1", {ne, ne}];
SMTAddMesh[Polygon[{{L / 2, 0}, {L, 0}, {L, H}, {L / 2, H}}], "Ω2", "Q1", {ne, ne}];
```

In[241]:=
```
SMTSensitivityProblem[{
   (* E is the third general parameter on "Ω1" domain *)
   {"E", {"P", (*∂E/∂E=1*)Identity, "Ω1" → 3(*E*)}},
   (* t is the fifth general parameter on all domains*)
   {"t", {"P", (*∂t/∂t=1*)Identity, All → 5(*t*)}},
   (* L is the parameter on which
    coordinates X and Y depends and the total force in Y direction*)
   {"L", {"S", "∂X/∂L", All -> (*X*)1}, {"S", "∂Y/∂L", All -> (*Y*)2},
    {"NBC", "∂Fy/∂L", "D" → 2}},
   (* Q is natural boundary condition parameter*)
   {"Q", {"NBC", "∂Fy/∂Q", "D" → 2}},
   (* vp is essential boundary condition parameter *)
   {"vp", {"EBC", "∂v/∂vp", "D" → 2}}
  }
  , "Order" → 2
  , "SecondOrderVelocityFields" → {{"Q", "L"}, {"NBC", "∂²Fy/∂L∂Q", "D" → 2}}
 ];
```

In[242]:=
```
SMTAnalysis[];
```

- The velocity fields of first-order have to be define in the same manner as shown in previous example. In this example, the only non-zero second-order velocity field is the derivative of vertical force (acting on the top of the domain) with respect to length L and distributed force Q: $\frac{\partial^2 F_y}{\partial L \partial Q}$

In[243]:=
```
SMTSetVelocityFields[{
    "∂X/∂L" → {All, Function[{n, X, Y}, X / L]},
    "∂Y/∂L" → {All, Function[{n, X, Y}, Y / L]},
    "∂Fy/∂L" → {Line[{{0, H}, {L, H}}], Function[{n, X, Y}, Q / (2 ne)]},
    "∂Fy/∂L" → {Point[{0, H}] || Point[{L, H}], Function[{n, X, Y}, Q / (4 ne)]},
    "∂Fy/∂Q" → {Line[{{0, H}, {L, H}}], Function[{n, X, Y}, L / (2 ne)]},
    "∂Fy/∂Q" → {Point[{0, H}] || Point[{L, H}], Function[{n, X, Y}, L / (4 ne)]},
    "∂v/∂vp" → {Point[{L, 0}], {-1}},
    "∂²Fy/∂L∂Q" → {Line[{{0, H}, {L, H}}], Function[{n, X, Y}, 1 / (2 ne)]},
    "∂²Fy/∂L∂Q" → {Point[{0, H}] || Point[{L, H}], Function[{n, X, Y}, 1 / (4 ne)]}
   }];
```

In[244]:=
```
SMTNextStep["λ" → 1];
While[SMTConvergence[10^-9, 10], SMTNewtonIteration[]];
SMTStatusReport[];
```

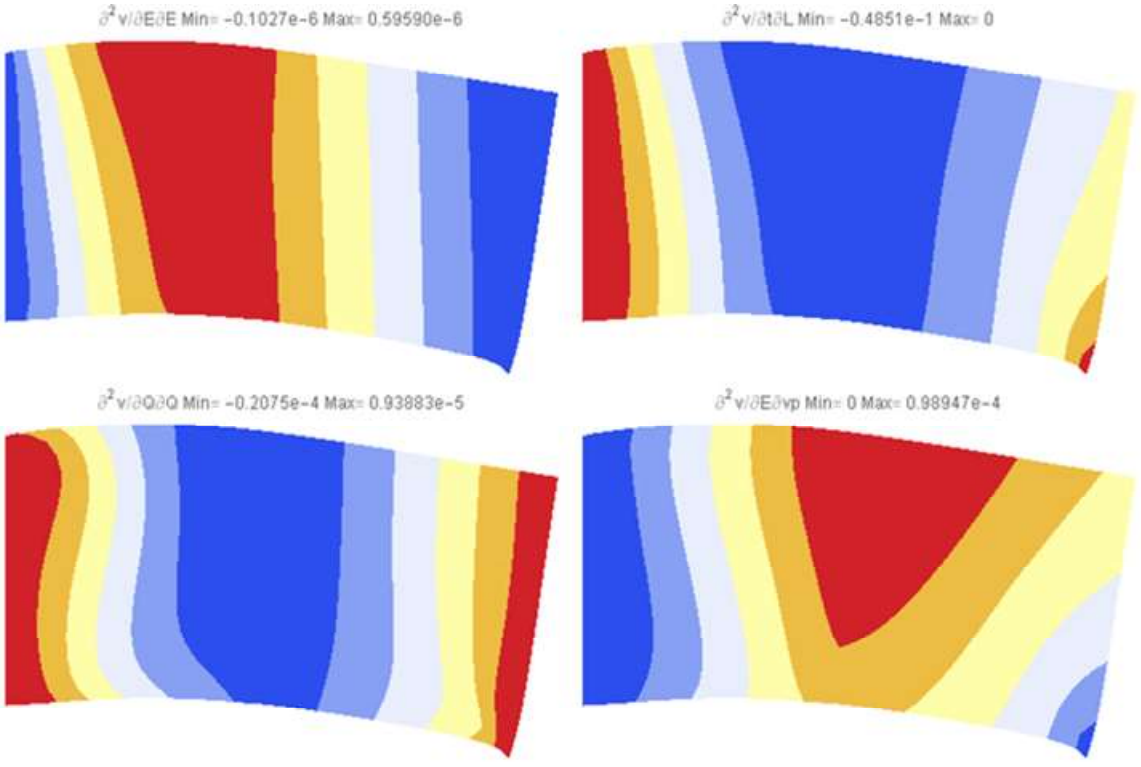
```
Step/Iter=1/6 λ/Δλ=1./1. ||Δp|| / ||R|| =
 1.93756×10^-13/5.33687×10^-11 Events=0 Status=0/{Convergence}
```

- The command SMTForwardSensitivity in this case performs first- and second-order sensitivity analysis with respect to all parameters.

In[247]:=
```
SMTForwardSensitivity[];
```

- The second-order sensitivities:

In[248]:=
```
ImageCollage[
 Table[SMTShowMesh["DeformedMesh" → True, "Mesh" → False, "Field" → SMTPostData[{"st", 2, p}],
   "Legend" → False, "Contour" → 5, "Label" → {"∂²v/∂", p[[1]], "∂", p[[2]], " ", Automatic}]
  , {p, {{"E", "E"}, {"t", "L"}, {"Q", "Q"}, {"E", "vp"}}}], Background → White]
```



- The results of second-order sensitivity analysis e.g. of the vertical displacement in point A in the center of the domain (point {L/2,H/2}).

$\frac{D^2 v_A}{D E^2}$:

In[249]:=
```
SMTPostData[{"st", 2, {"Q", "Q"}}, Point[{L / 2, H / 2}]]
```

```
-0.0000205545
```

$\frac{D^2 v_A}{D L^2}$:

In[250]:=
```
SMTPostData[{"st", 2, {"L", "L"}}, Point[{L / 2, H / 2}]]
```

```
-0.000310233
```

$\frac{D^2 v_A}{DE\, DL}$:

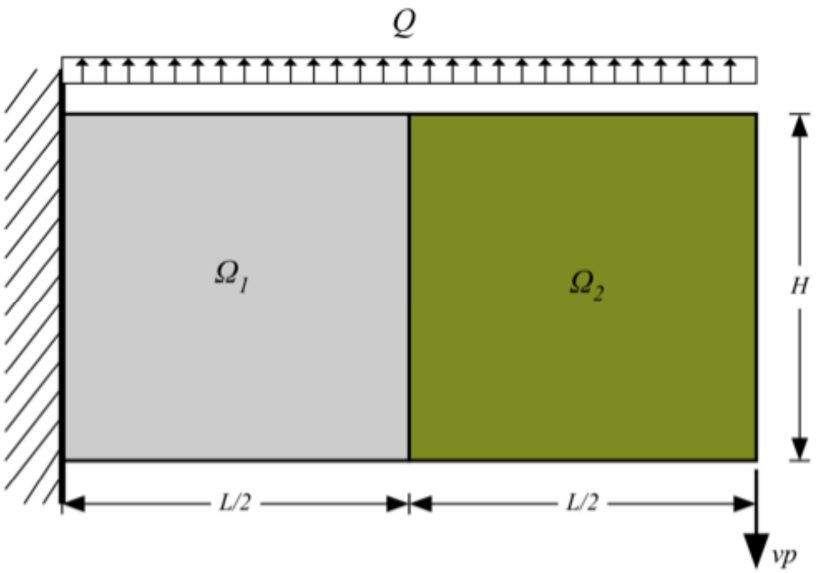
In[251]:=
```
SMTPostData[{"st", 2, {"E", "L"}}, Point[{L / 2, H / 2}]]
```

```
-0.000032868
```

General Backward Mode Sensitivity Analysis Example

Description of backward mode example

To demonstrate the use of backward sensitivity analysis for steady-state problem we will analyze the same two-domain example Ω_1 and Ω_2 as for forward mode. For backward mode we will assume different sensitivity parameters which are defined as **thicknesses of the domain in each individual node**. Number of sensitivity parameters n_ϕ is therefore equal to number of nodes. Sensitivity parameters we denote with letter ϕ and corresponding index, e.g. ϕ_1, $\phi_{2,...}$. For the convenience it is chosen that indexes of sensitivity parameters are equivalent to global indexes of nodes. Thickness of the domain is approximated by $t(\xi, \eta) = t_0 + N_i(\xi, \eta)\,\Delta t_i$, with initial, default value $\Delta t_i = 0$.



The possible response functionals coded in the ExamplesSensitivity2D element of are:

F1 = $\int_V W\, dV$ ⇒ integrated strain energy

F2 = $\int_V \tilde{\sigma}^n\, dV$ ⇒ integral of Mises stress^n

F3 = v_{nv} ⇒ displacement v in nv-th node

F4 = $||\mathbf{p}||_2$ ⇒ L2 norm of displacement vector

The actual response functional is defined by "IntegerInput"→{func_type} option of SMTBackwardSensitivity command.

The finite element code with the subroutines for backward sensitivity analysis for the chosen sensitivity parameters and response functionals is presented in `Finite Strain Element for Direct and Sensitivity Analysis`.

Input parameters:

In[580]:=
```
<< AceFEM`;
L1 = 10.;(*Length of the domain*)
H1 = 5.;(*Height of the domain*)
Q = 50.;(*Continuous vertical load along the top of the whole domain.*)
vp = 1.;(*Prescribed vertical displacement in point A*)
ne = 20;(*number of finite elements along y axis and
 number of finite elements along x axis on each domain: Ω1 and Ω2*)
t0 = 1;(* average initial thickness *)
```

First order backward mode sensitivity analysis of integral of Mises stress

In[583]:=
```
SMTInputData[];
nϕ = (ne + 1) (2 ne + 1);
SMTAddDomain[
  {"Ω1", "ExamplesSensitivity2D", {"E *" -> 1000, "ν *" -> 0.3, "t *" -> t0}},
  {"Ω2", "ExamplesSensitivity2D", {"E *" -> 5000, "ν *" -> 0.2, "t *" -> t0}}
 ];
SMTAddEssentialBoundary[Line[{{0, 0}, {0, H1}}], 1 -> 0, 2 → 0];
SMTAddNaturalBoundary[ Line[{{0, H1}, {L1, H1}}], 2 → Line[{Q}]];
SMTAddEssentialBoundary[ Point[{L1, 0}], 2 → -vp];
SMTAddMesh[Polygon[{{0, 0}, {L1 / 2, 0}, {L1 / 2, H1}, {0, H1}}], "Ω1", "Q1", {ne, ne}];
SMTAddMesh[Polygon[{{L1 / 2, 0}, {L1, 0}, {L1, H1}, {L1 / 2, H1}}], "Ω2", "Q1", {ne, ne}];
SMTSensitivityProblem[
  "NoSensitivityParameters" → nϕ
  , "Mode" → "Backward"
  , "ResponseFunctionals" → {"F2"}
  , "Order" → 1
 ];
SMTAnalysis[];
```

- Set $\Delta t_i = 0$.

In[593]:=
```
SMTNodeData["Data", {0}];
```

- Here is the primal analysis executed.

In[594]:=
```
SMTNextStep["λ" → 1.0];
While[SMTConvergence[10^-9, 10], SMTNewtonIteration[]];
```

- This stores the state of simulation that will be needed for backward sensitivity analysis. The file name of the stored state is "tmpBackward_1.bst".

In[596]:=
```
SMTDumpState["tmpBackward", "BackwardSensitivity" → True]
```

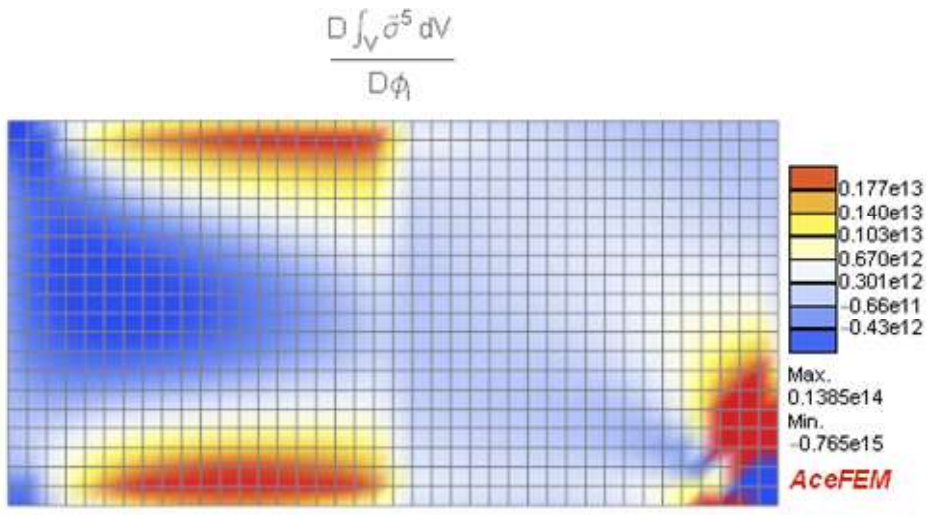
Out[596]= {tmpBackward_1.bst, 80 738}

- This calls backward method sensitivity related user subroutines and calculates the equation (9), that calculates $\frac{D\hat{F}_k}{D\phi_i}$. "IntegerInput"→{2} defines that the response functional is $\tilde{\sigma}^n$ and exponent n = 5 of $\tilde{\sigma}^n$ is defined by "RealInput"→{5.}.

In[597]:=
```
ΔMises = SMTBackwardSensitivity["tmpBackward", "IntegerInput" → {2}, "RealInput" → {5.}];
```

- Graphical display of $\frac{D\int_V \tilde{\sigma}^5 \, dV}{D\phi_i}$ where ϕ_i is thickness in *i*-th node.

In[598]:=
```
SMTShowMesh["Field" → ΔMises, "Label" → "D∫V σ̃^5 dV / Dϕi"]
```



Second order backward mode sensitivity analysis of integral of Mises stress

For second order backward sensitivity analysis, the sensitivity parameters are given in the same manner as in forward sensitivity analysis - with definition of velocity fields.

```
In[599]:= << AceFEM`;
          SMTInputData[];
          L1 = 10.; Q = 50.; vp = 1.; H1 = 5.; ne = 20; nϕ = (ne + 1) (2 ne + 1);
          SMTAddDomain[
            {"Ω1", "ExamplesSensitivity2D", {"E *" -> 1000, "ν *" -> 0.3, "t *" -> 1}},
            {"Ω2", "ExamplesSensitivity2D", {"E *" -> 5000, "ν *" -> 0.2, "t *" -> 1}}
           ];
          SMTAddEssentialBoundary[Line[{{0, 0}, {0, H1}}], 1 -> 0, 2 → 0];
          SMTAddNaturalBoundary[ Line[{{0, H1}, {L1, H1}}], 2 → Line[{Q}]];
          SMTAddEssentialBoundary[ Point[{L1, 0}], 2 → -vp];
          SMTAddMesh[Polygon[{{0, 0}, {L1 / 2, 0}, {L1 / 2, H1}, {0, H1}}], "Ω1", "Q1", {ne, ne}];
          SMTAddMesh[Polygon[{{L1 / 2, 0}, {L1, 0}, {L1, H1}, {L1 / 2, H1}}], "Ω2", "Q1", {ne, ne}];
          SMTSensitivityProblem[
            (* t is the fifth general sensitivity
             parameter on all domains as defined by SMSSensitvityNames *)
            Table[{"ϕ" <> ToString[i], {"P", "∂t/∂ϕ" <> ToString[i], All → 5}}, {i, nϕ}]
            , "Mode" → "Backward"
            , "ResponseFunctionals" → {"F2"}
            , "Order" → 2
           ];
          SMTAnalysis[];
          SMTNodeData["Data", {0}];
```

- Definition of velocity fields: sensitivity parameters are thicknesses in nodes, therefore corresponding velocity field is a field with all zeros except in node for which sensitivity parameter is defined.

```
In[611]:= SMTSetVelocityFields[Table["∂t/∂ϕ" <> ToString[ip] → {{ip}, {1}}, {ip, nϕ}]];
```

- Here is the primal analysis executed in two steps.

```
In[612]:= SMTNextStep["λ" → 1.0];
          While[SMTConvergence[10^-9, 10], SMTNewtonIteration[]];
```

- Here is the first order forward sensitivity analysis is executed.

```
In[614]:= SMTForwardSensitivity[];
```

- This stores the state of simulation that will be needed for backward sensitivity analysis. The file name of the stored state is "tmpBackward.bst".

In[615]:=
```
SMTDumpState["tmpBackward", "BackwardSensitivity" → True];
```

- This calls backward method sensitivity related user subroutines and calculates the equation (10), that calculates $\frac{D^2 \hat{F}_k}{D\phi_i\, D\phi_j}$.

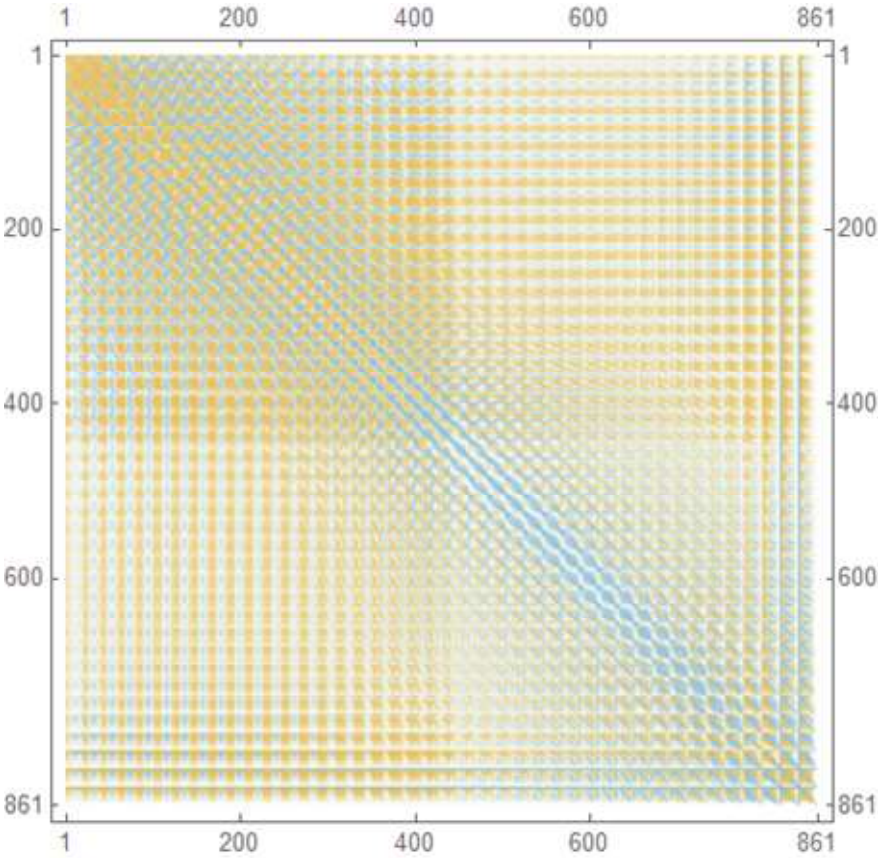
In[616]:=
```
ΔΔMises = SMTBackwardSensitivity["tmpBackward", "IntegerInput" → {2}, "RealInput" → {5.}];
```

- Hessian matrix has dimensions 861×861, thus a total of $\frac{861\,(861+1)}{2}$ = 371 091 derivatives has been calculated. It would be impossible to calculate that in forward mode, due to the fact that in forward mode that leads to the solution of the linear system of equations with 371 091 right hand sides.

In[617]:=
```
ΔΔMises // Dimensions
```

Out[617]=
```
{861, 861}
```

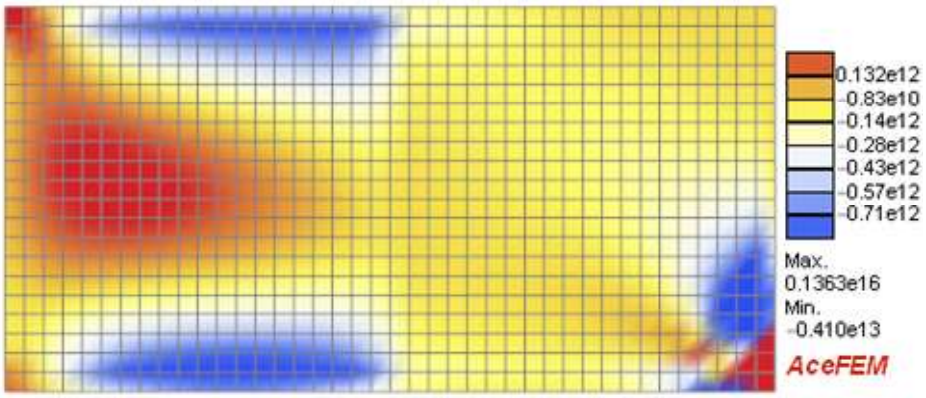
In[618]:=
```
MatrixPlot[ΔΔMises]
```



- This plot leading ($\frac{D \int_V \tilde{\sigma}^5\, dV}{D\phi_i\, D\phi_i}$) elements of the matrix.

In[619]:=
```
SMTShowMesh["Field" → Diagonal[ΔΔMises], "Label" → "D ∫V σ̃^5 dV / Dϕi Dϕi"]
```

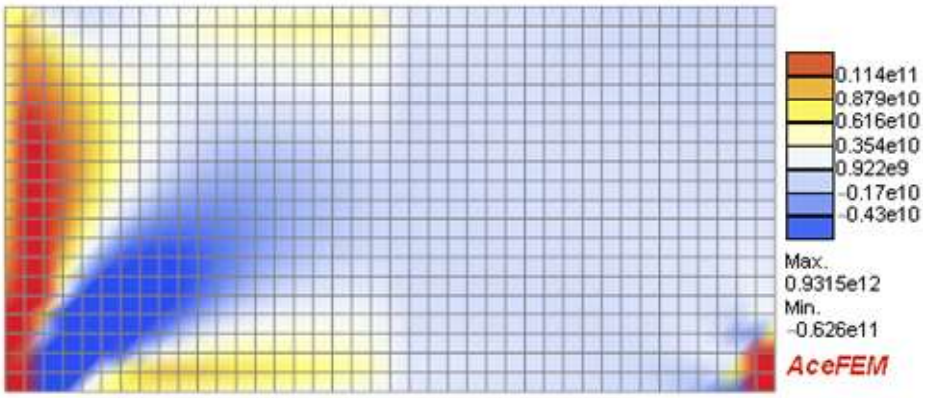
$$\frac{D\int_V \tilde{\sigma}^5\, dV}{D\phi_1\, D\phi_1}$$



- This plot first row ($\frac{D\int_V \tilde{\sigma}^5\, dV}{D\phi_1\, D\phi_i}$) elements of the matrix.

In[620]:=
```
SMTShowMesh["Field" → ΔΔMises[[1]], "Label" → "D∫V σ̃^5 dV / Dϕ1 Dϕi"]
```

$$\frac{D\int_V \tilde{\sigma}^5\, dV}{D\phi_1\, D\phi_i}$$



First order backward mode sensitivity analysis of L2 norm of displacement vector

In[488]:=
```
SMTInputData[];
nϕ = (ne + 1) (2 ne + 1);
SMTAddDomain[
  {"Ω1", "ExamplesSensitivity2D", {"E *" -> 1000, "ν *" -> 0.3, "t *" -> t0}},
  {"Ω2", "ExamplesSensitivity2D", {"E *" -> 5000, "ν *" -> 0.2, "t *" -> t0}}
 ];
SMTAddEssentialBoundary[Line[{{0, 0}, {0, H1}}], 1 -> 0, 2 → 0];
SMTAddNaturalBoundary[ Line[{{0, H1}, {L1, H1}}], 2 → Line[{Q}]];
SMTAddEssentialBoundary[ Point[{L1, 0}], 2 → -vp];
SMTAddMesh[Polygon[{{0, 0}, {L1 / 2, 0}, {L1 / 2, H1}, {0, H1}}], "Ω1", "Q1", {ne, ne}];
SMTAddMesh[Polygon[{{L1 / 2, 0}, {L1, 0}, {L1, H1}, {L1 / 2, H1}}], "Ω2", "Q1", {ne, ne}];
SMTSensitivityProblem[
  "NoSensitivityParameters" → nϕ
  , "Mode" → "Backward"
  , "ResponseFunctionals" → {"F4"}
  , "Order" → 1
 ];
SMTAnalysis[];
SMTNodeData["Data", {0}];
```

- Here is the primal analysis executed.

In[499]:=
```
SMTNextStep["λ" → 1.0];
While[SMTConvergence[10^-9, 10], SMTNewtonIteration[]];
```

- Calculation of derivatives of response functional F2 with respect to solution vector $\frac{\partial F2}{\partial \mathbf{p}}$. In current example, the solution vector is a vector that contains horizontal (u) and vertical (v) displacements in all nodes: $\mathbf{p} = \{u_1,v_1,u_2,v_2,...,u_{nNodes},v_{nNodes}\}$. The derivative $\frac{\partial F2}{\partial \mathbf{p}}$ is: $\frac{\partial F2}{\partial \mathbf{p}}=\{\frac{u_1}{||\mathbf{u}||_2},\frac{v_1}{||\mathbf{u}||_2},\frac{u_2}{||\mathbf{u}||_2},\frac{v_2}{||\mathbf{u}||_2},...,\frac{u_{nNodes}}{||\mathbf{u}||_2},\frac{v_{nNodes}}{||\mathbf{u}||_2}\}$.

In[501]:=
```
displacementsNorm = Norm[Flatten[SMTNodeData["at"]], 2];
DF4Du = SMTPostData["u"] / displacementsNorm;
DF4Dv = SMTPostData["v"] / displacementsNorm;
```

- This sets data of derivatives of response functional with respect to degrees of freedom ($\frac{\partial F}{\partial \mathbf{p}}$).

In[504]:=
```
SMTNodeData["BSFVF", Transpose[{DF4Du, DF4Dv}]];
```

- This stores the state of simulation that will be needed for backward sensitivity analysis. The file name of the stored state is "tmpBackward_1.bst".

In[505]:=
```
SMTDumpState["tmpBackward", "BackwardSensitivity" → True]
```

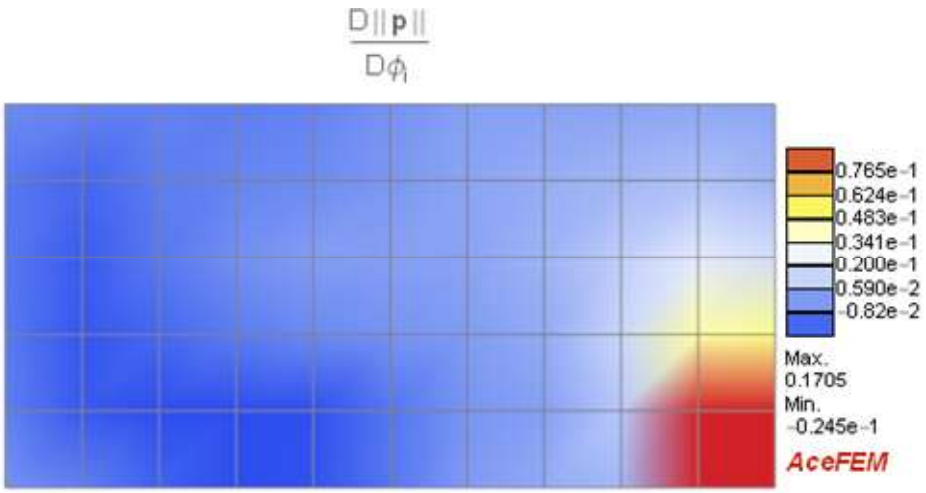
Out[505]=
```
{tmpBackward_1.bst, 11 408}
```

- "IntegerInput"→{4} defines that the response functional is L2 norm of displacement vector.

In[506]:=
```
ΔL2 = SMTBackwardSensitivity["tmpBackward", "IntegerInput" → {4}];
```

- Graphical display of $\frac{D||\mathbf{p}||}{D\phi_i}$ where ϕ_i is thickness in *i*-th node.

In[507]:=
```
SMTShowMesh["Field" → ΔL2, "Label" → "D||p|| / Dϕi"]
```


D||p||
Dϕ1
0.765e-1
0.624e-1
0.483e-1
0.341e-1
0.200e-1
0.590e-2
-0.82e-2
Max.
0.1705
Min.
-0.245e-1
AceFEM

Contact Problems

Contents

Implementation Notes for Contact Elements

Contact section has been contributed by Jakub Lengiewicz, Institute of Fundamental Technological Research, Warszawa, Poland.

The contact support in AceGen is based on a particular scheme, which will be presented below. The basic part of the scheme is an element. We use the special kind of elements in AceGen (a *contact elements*) to formulate the contact laws. We make a distinction there in the contact elements for slave and master part. The slave part is defined directly as the system covers the bodies surfaces with contact elements. The master part is a group of special kind of nodes (Node Specification Data) which are considered as empty slots where another nodes are placed during the evaluation (due to a *global search routine*).

Such an element must provide additional information required for *global search routine* purposes:

- contact type for slave and master part separately (see table below) put into SMSCharSwitch array in SMSTemplate procedure,
- information, that the element should be considered as contact element -- "ContactElement" string put into SMSCharSwitch array,
- SMSStandardModule["Nodal information"] describing the actual and previous positions of all integration points (slave nodes).

SMSTemplate[...., "SMSCharSwitch"->{*slave part contact type, master part contact type*,"ContactElement"}, ...]
information, that the element should be considered as contact element -- “ContactElement” string put into SMSCharSwitch array

slave part contact types	description
"CTD2N1"	one 2 D node
"CTD2L1"	one 2 D line
" CTD3V1 "	one 3 D node
" CTD3P1 "	one 3 D triangle
" CTD3S1 "	one 3 D quad

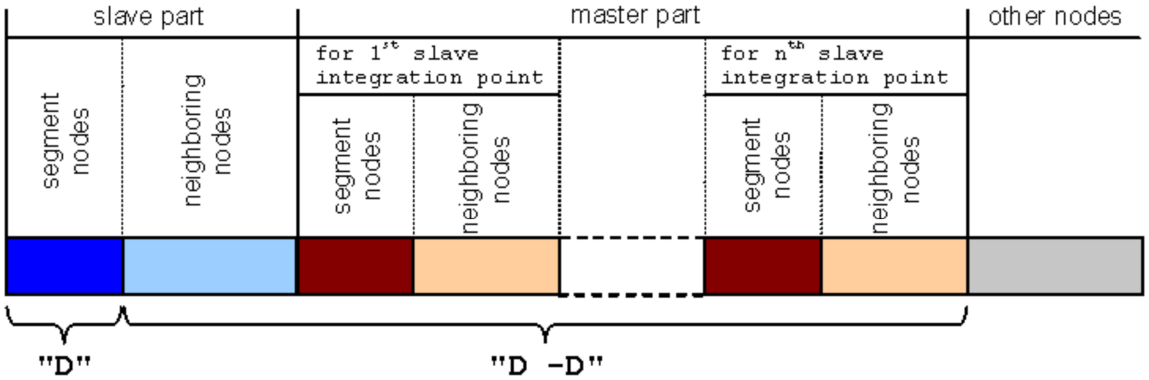
master part contact types	description
"CTNULL"	no nodes
" CTD2N1 "	one 2 D node
" CTD2L1 "	one 2 D line
" CTD3V1 "	one 3 D node (vertex)
" CTD3P1 "	one 3 D triangle
" CTD3S1 "	one 3 D quad

Contact types supported by the system

It is possible to extend each contact type (except CTNULL) to attach also neighboring nodes. You do this appending "DN<i>" string to the contact type, where <i> is a number of dummy slots prepared for each node's neighborhood. Thus you have to define additional (i*NumberOfSlaveNodes) slots if you extend the slave part or (i*NomberOfMasterNodes*NumberOfIntegrationPoints) slots in case of extending master part.

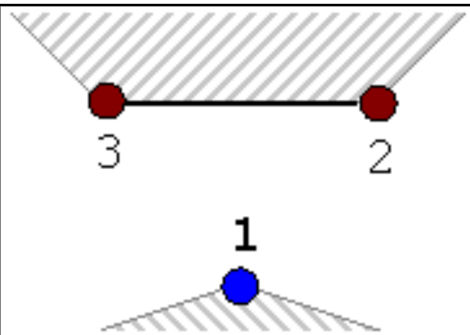
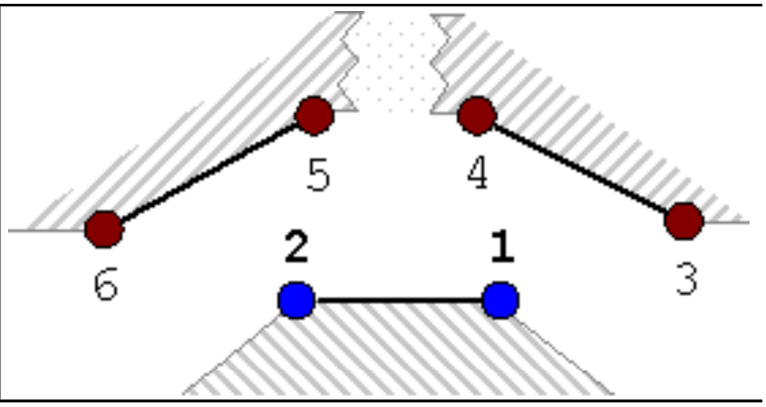
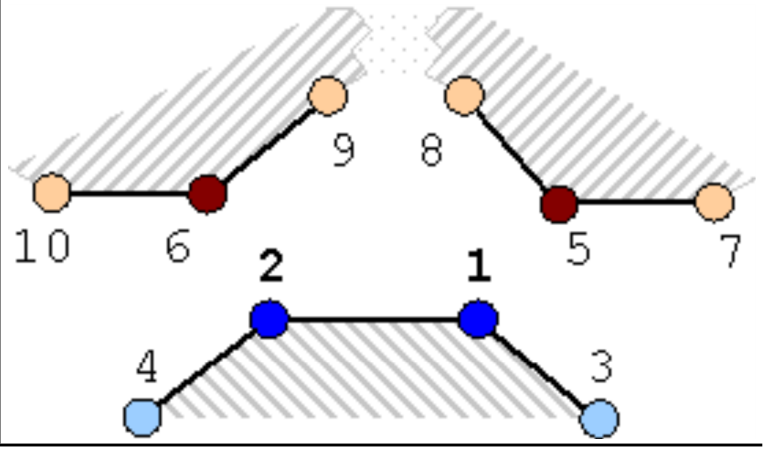
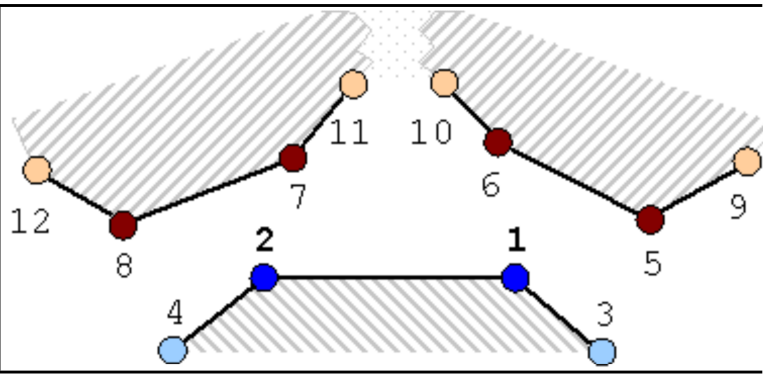
NOTE: the number of neighboring nodes may vary during analysis, while the number of slots in element is strictly defined by "DN<i>" flag. The question is: what the system puts into unused dummy slots. The answer is: the most recent neighbor found for given node. The reason is that the common use of neighboring nodes is to calculate the normal vectors. If we put dummy node instead of the last found, the element code would be more complicated because of if/else statements.

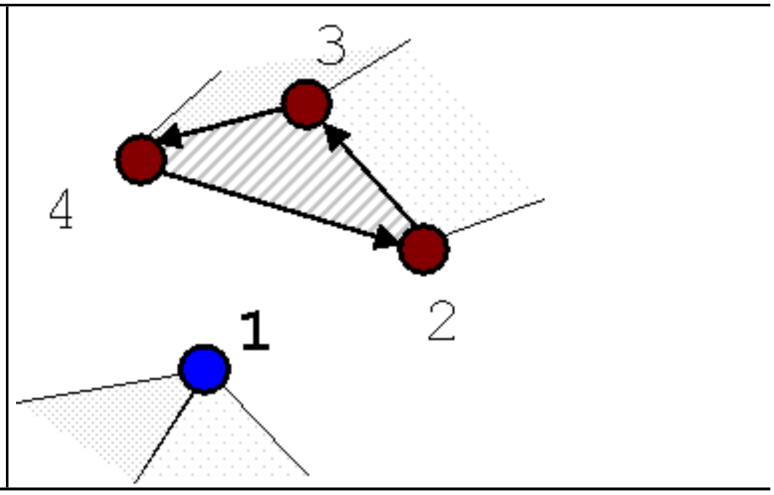
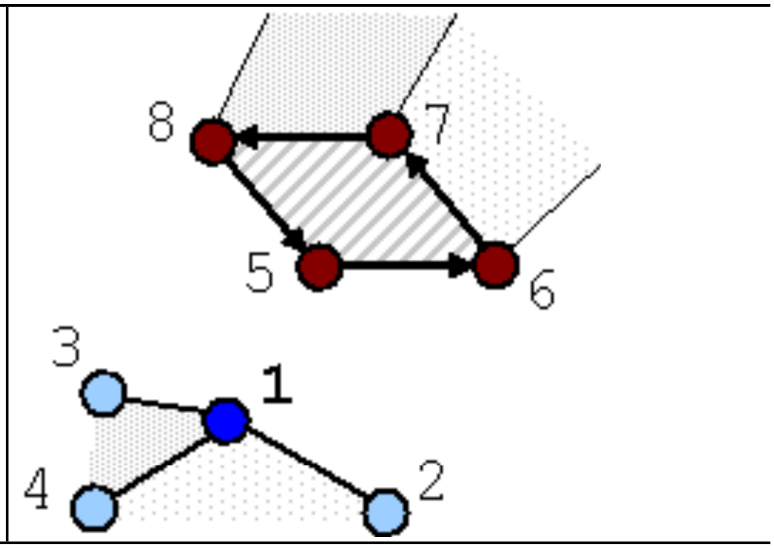
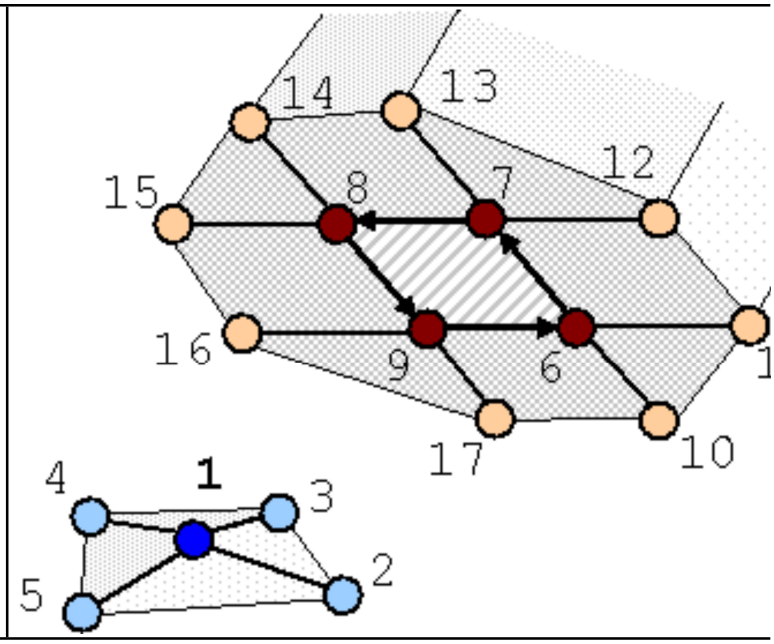
NOTE: from the same reasons as above, there might be the situation where there are more neighboring nodes found than the number of slots in contact element. In such a situation the slots are filled with a natural order until it is possible. There is only one exception to this procedure: the last slot is always reserved for the last found neighboring node.



Nodes position in the contact element (general grouping scheme.)

{type of slave, type of master}	Detailed numbering scheme
{"CTD2N1","CTNULL"}	

{"CTD2N1","CTD2L1"}	
{"CTD2L1","CTD2L1"}	
{"CTD2L1DN1","CTD2N1DN1"}	
{"CTD2L1DN1","CTD2L1DN1"}	

{"CTD3V1","CTD3P1"}	
{"CTD3V1DN3","CTD3S1"}	
{"CTD3V1DN4","CTD3S1DN2"}	

Position of nodes in the contact element (detailed numbering scheme for chosen examples.)

After definition of the contact element we may proceed with the analysis. In particular contact system, there are several bodies which may come into the contact. Each of them has its own boundary (boundaries) where the contact elements must be placed. We need to

apply some extra parameters to SMTAddMesh procedure to specify the body name to which the mesh belongs and the list of domains (contact domains in our case) which will be used to cover the surface of the body.

By default ("ContactPairs"->Automatic) the lexicographical order of BodyID's is used by the *global search routine* (in master-slave approach): for each surface node taken from particular body it searches for the closest segment on the surface of bodies which ID's are "greater". So slave bodies have "lower" ID's than master bodies. All possible combinations of bodies are checked.

The SMTAnalysis option "ContactPairs" –> {{slaveBodyID$_1$, masterBodyID$_1$}, {slaveBodyID$_2$, masterBodyID$_2$}, ...} specifies the pairs slave-body/master -body for which the possible contact condition is checked.

After SMTAnalysis one may need to provide some additional coefficients for *global search routine* purpose. We setup them as (see Real Type Environment Data) :

Default form	default	description
SMTRData["ContactSearchTol-erance",*tolerance*]	0.001 SMTRData["XYZRange"]	contact search tolerance for global search routine.

Additional environmental data for contact search purposes.

Then, calling the SMTNewtonIteration[] procedure, the *global search routine* is executed to search for contact pairs and, eventually, update the information in elements' dummy slots.

NOTE: it is important to cover with contact elements also the body, which is "master" to all the others. Such an elements are needed to calculate the actual positions of nodes (integration points) on the surface of this body. It is enough to have only "Nodal information" subroutine defined there. In this case we may set the master part contact type as "CTNULL".

Function name	description
SMTContactData[]	prints out the actual contact state (node–>segment mapping)
SMTContactSearch[]	manually runs the global contact search routine

Contact related functions.

2D slave node - line master segment element

The support for the analysis of contact problems in AceFEM and implementation of contact finite elements in AceGen is described in section Implementation Notes for Contact Elements.

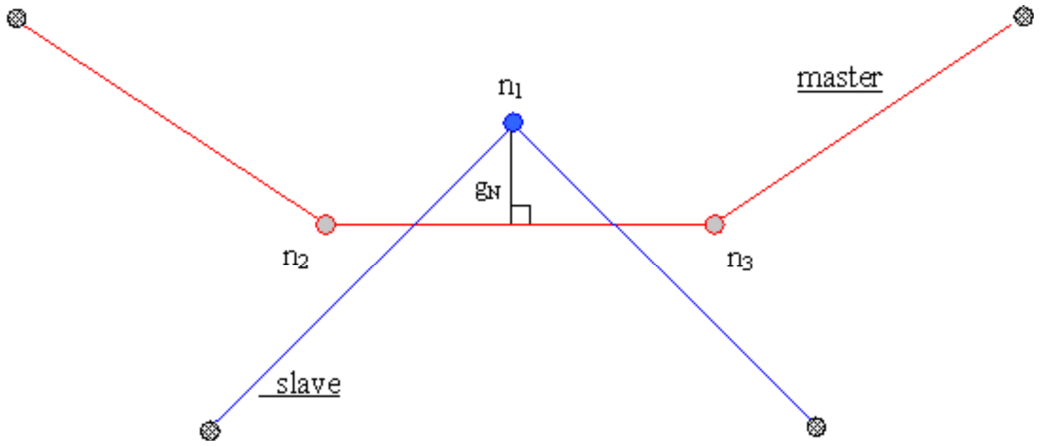
Description

Generate the node-to-segment element for analysis of the 2-D frictionless contact problems. The element has the following characteristics:

- 3 node element: one slave node + two dummy nodes (for linear master segment)
- global unknowns are displacements of the nodes,
  the element should allow arbitrary large displacements,
- the impenetrability condition is regularized with penalty method and formulated as energy potential:

$$\Pi = \begin{cases} \sum_{i=1}^{N} \frac{1}{2} \rho \, g_N^{\,2} & \text{for } g_N \le 0 \\ 0 & \text{for } g_N > 0 \end{cases}$$

where ρ is penalty parameter, and g_N is normal gap (at the point of orthogonal projection of slave point on master boundary). N is a number of nodes on slave contact surface.



The following user subroutines have to be generated:

- user subroutine for specification of nodal positions,
- user subroutine for the direct implicit analysis.

Solution

In[1]:=
```
<< AceGen`
SMSInitialize["ExamplesCTD2N1L1Pen", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "V2", "SMSNoDimensions" → 2, "SMSNoNodes" → 3,
  "SMSSymmetricTangent" → True, "SMSDOFGlobal" → {2, 2, 2}, "SMSSegments" → {{1}},
  "SMSAdditionalNodes" → Null, "SMSNodeID" → {"D", "D -D", "D -D"},
  "SMSCharSwitch" → {"CTD2N1", "CTD2L1", "ContactElement"}
  , "SMSDomainDataNames" -> {"ρ -penalty parameter"}, "SMSDefaultData" → {1000}];

SMSStandardModule["Nodal information"];
Xi ⊨ Table[SMSIO["Nodal coordinates"[i, 1]], {i, 1}];
Yi ⊨ Table[SMSIO["Nodal coordinates"[i, 2]], {i, 1}];
ui ⊨ Table[SMSIO["Nodal DOFs"[i, 1]], {i, 1}];
vi ⊨ Table[SMSIO["Nodal DOFs"[i, 2]], {i, 1}];
uiP ⊨ Table[SMSIO["Nodal DOFs n"[i, 1]], {i, 1}];
viP ⊨ Table[SMSIO["Nodal DOFs n"[i, 2]], {i, 1}];
x ⊨ {Xi + ui, Yi + vi};
xP ⊨ {Xi + uiP, Yi + viP};
SMSExport[{Flatten[{x, xP}]}, d$$];

SMSStandardModule["Tangent and residual"];
{Xs, X1, X2} ⊨ SMSIO["Nodal coordinates"];
{us, u1, u2} ⊨ SMSIO["Nodal DOFs"];
u ⊨ Flatten[{us, u1, u2}];
{ρ} ⊨ SMSIO["Domain data"];
 (* check if contact is not detected by the global search *)
SMSIf[SMSInteger[ns$$[2, "id", "Dummy"]] == 1];
 SMSReturn[];
SMSElse[];
 xs = Xs + us; x1 = X1 + u1; x2 = X2 + u2;
 l ⊨ SMSSqrt[(x2 - x1).(x2 - x1)];
 t ⊨ x2 - x1;
 eT ⊨ t/Sqrt[t.t];
 eN ⊨ ({0, 0, 1} × {eT[[1]], eT[[2]], 0})[[{1, 2}]];
```

```
ξ ⊨ ((xs - x1).eT)/l;
xξ ⊨ x1 + (x2 - x1) ξ;
r ⊨ -xs + xξ;
gN ⊨ r.eN;
SMSIf[gN > 0];
 SMSReturn[];
SMSEndIf[];
Π ⊨ 1/2 ρ gN^2;
SMSDo[i, 1, 6];
 Ri ⊨ SMSD[Π, u, i];
 SMSIO[Ri, "Export to", "Residual"[i]];
 SMSDo[j, i, 6];
  Kij = SMSD[Ri, u, j];
  SMSIO[Kij, "Export to", "Tangent"[i, j]];
 SMSEndDo[];
SMSEndDo[];
SMSEndIf[];
SMSWrite[];
```

File: ExamplesCTD2N1L1Pen.c Size: 6887 Time: 2

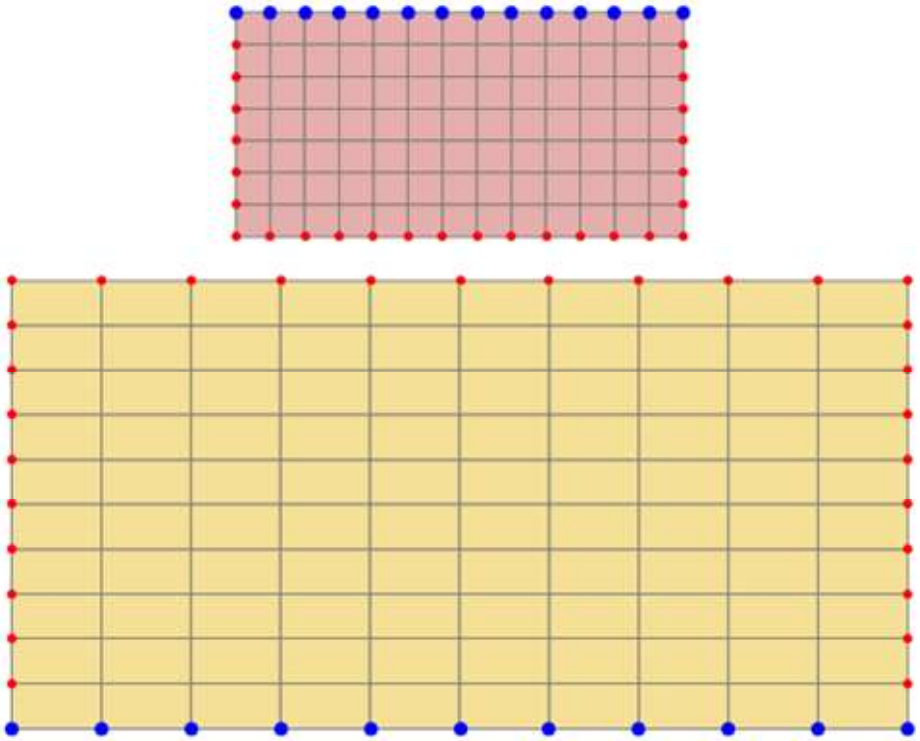
Method	PAN	SKR
No.Formulae	3	55
No.Leafs	84	889

2D indentation problem

The support for the analysis of contact problems in AceFEM and implementation of contact finite elements in AceGen is described in section Implementation Notes for Contact Elements.

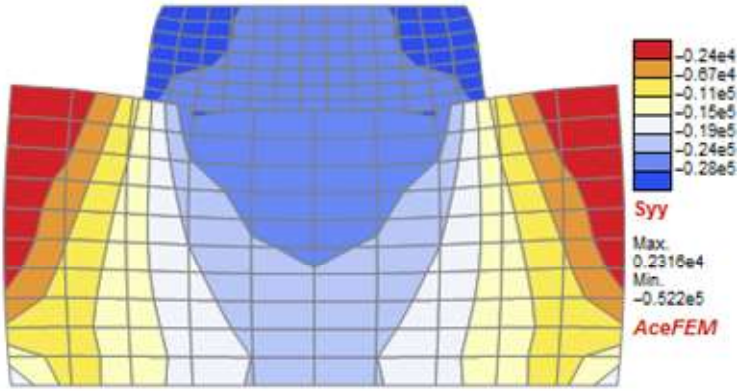
Description

Use the contact element from `2D slave node` and `line master segment element` section and hypersolid element from Solid, Finite Strain Element for Direct and Sensitivity Analysis section to analyze the simple 2D indentation problem: small elastic box pressed down into large elastic box.



Solution

In[62]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[
   {"Solid1", "ExamplesHypersolid2D", {"E *" -> 70000, "ν *" -> 0.3}},
   {"Solid2", "ExamplesHypersolid2D", {"E *" -> 210000, "ν *" -> 0.3}},
   {"Contact", "ExamplesCTD2N1L1Pen", {"ρ *" -> 200000}}];
SMTAddMesh[
  Polygon[{{-1/2, 0.1}, {1/2, 0.1}, {1/2, 0.6}, {-1/2, 0.6}}], "Solid1", "Q1", {13, 7},
  "BodyID" → "B1", "BoundaryDomainID" -> "Contact"];
SMTAddMesh[Polygon[{{-1, -1}, {1, -1}, {1, 0}, {-1, 0}}], "Solid2", "Q1", {10, 10},
  "BodyID" → "B2", "BoundaryDomainID" -> "Contact"];
SMTAddEssentialBoundary[{"Y" == -1 &, 1 -> 0, 2 -> 0}, {"Y" == 0.6 &, 1 -> 0, 2 -> -1}];
SMTAnalysis["Output" -> "tmp1.out"];
SMTRData["ContactSearchTolerance", 0.01];
SMTNextStep["λ" → 0.1];
While[
  While[step = SMTConvergence[10^-8, 10, {"Adaptive BC", 7, 0.0001, 0.2, 0.35}],
   SMTNewtonIteration[];];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]],
  If[step[[1]], SMTStepBack[], SMTShowMesh["DeformedMesh" → True, "Marks" → False,
     "Domains" → {"Solid1", "Solid2"}, "Show" → "Window", "Field" -> "Syy", "Contour" → True];];
  SMTNextStep["Δλ" → step[[2]]]
 ];
SMTShowMesh["DeformedMesh" → True, "Field" -> "Syy", "Contour" → True]
```



2D slave node - smooth master segment element

The support for the analysis of contact problems in AceFEM and implementation of contact finite elements in AceGen is described in section Implementation Notes for Contact Elements.

Description

Generate the node-to-segment smooth element for analysis of the 2-D contact problems (Coulomb friction, augmented Lagrange multipliers method). The element has the following characteristics:

⇒ 8 node element: one slave node + two dummy slave nodes (referential area) + four master nodes (3rd order Bezier curve) + Lagrange multipliers node,

⇒ global unknowns are displacements of the nodes + Lagrange multipliers,

⇒ the element should allow arbitrary large displacements,

⇒ the impenetrability condition is regularized with augmented Lagrange multipliers method and formulated as energy potential (for each node on slave contact surface):

The augmented Lagrangian functional $\mathcal{L}$ for two solid bodies in contact is constructed as

$$\mathcal{L}(\boldsymbol{\varphi}, \boldsymbol{\lambda}; \hat{T}_N) = \Pi_1(\boldsymbol{\varphi}^1) + \Pi_2(\boldsymbol{\varphi}^2) + \Pi_c(\boldsymbol{\varphi}, \boldsymbol{\lambda}; \hat{T}_N),$$

where contact contribution leads to

$$\Pi_c(\boldsymbol{\varphi}, \boldsymbol{\lambda}; \hat{T}_N) = \int_{\Gamma_c^1} l_N(g_N, \lambda_N)\, dS + \int_{\Gamma_c^1} l_T(\Delta \boldsymbol{g}_T, \boldsymbol{\lambda}_T; \hat{T}_N)\, dS,$$

with contribution of normal tractions

$$l_N(g_N, \lambda_N) = \begin{cases} \left(\lambda_N + \dfrac{\varrho}{2} g_N\right) g_N, & \hat{\lambda}_N \leqslant 0 \quad \text{(contact)} \\ -\dfrac{1}{2\varrho} \lambda_N^2, & \hat{\lambda}_N > 0 \quad \text{(separation)} \end{cases}$$

and contribution of tangential tractions

$$l_T(\Delta \boldsymbol{g}_T, \boldsymbol{\lambda}_T; \hat{T}_N) = \begin{cases} \left\{ \begin{array}{ll} \left(\boldsymbol{\lambda}_T + \dfrac{\varrho}{2} \Delta \boldsymbol{g}_T\right) \cdot \Delta \boldsymbol{g}_T, & \|\hat{\boldsymbol{\lambda}}_T\| \leqslant \hat{k} \text{ (stick)} \\ -\dfrac{1}{2\varrho}(\|\boldsymbol{\lambda}_T\|^2 - 2\hat{k}\|\hat{\boldsymbol{\lambda}}_T\| + \hat{k}^2), & \|\hat{\boldsymbol{\lambda}}_T\| > \hat{k} \text{ (slip)} \end{array} \right\} & \hat{\lambda}_N \leqslant 0 \\ -\dfrac{1}{2\varrho} \|\boldsymbol{\lambda}_T\|^2, & \hat{\lambda}_N > 0 \end{cases}.$$

Here $\boldsymbol{\lambda}$ is the field of Lagrange multipliers defined along Γ_c^1, such that its components in the local basis,

$$\lambda_N = \boldsymbol{\lambda} \cdot \boldsymbol{n}, \qquad \boldsymbol{\lambda}_T = \lambda_{T\alpha} \boldsymbol{\tau}^\alpha, \qquad \lambda_{T\alpha} = \boldsymbol{\lambda} \cdot \boldsymbol{\tau}_\alpha,$$

can be interpreted as the nominal contact tractions T_N and $\boldsymbol{T}_T = T_{T\alpha} \boldsymbol{\tau}^\alpha$. Further, $\hat{\lambda}_N = \lambda_N + \varrho g_N$ and $\hat{\boldsymbol{\lambda}}_T = \boldsymbol{\lambda}_T + \varrho \Delta \boldsymbol{g}_T$ are the augmented Lagrange multipliers and $\varrho > 0$ is a regularization parameter. Finally, $\hat{T}_N = \hat{\lambda}_N$ is the augmented normal contact traction which defines the radius $\hat{k}$ of the Coulomb cone in l_T according to $\hat{k} = \max(0, -\mu \hat{T}_N)$. Importantly, the augmented Lagrangian $\mathcal{L}$ is C^1 continuous.

The following user subroutines have to be generated:

⇒ user subroutine for specification of nodal positions,

⇒ user subroutine for the direct implicit analysis.

Solution

In[45]:=
```
<< AceGen`;
driver = 2;
SMSInitialize["ExamplesCTD2N1DN2L1DN1AL", "Environment" → "AceFEM"];
SMSTemplate[
  "SMSTopology" → "V2",
  "SMSNoDimensions" → 2,
  "SMSNoNodes" → 8,
  "SMSDOFGlobal" → {2, 2, 2, 2, 2, 2, 2, 2},
  "SMSSegments" → {},
  "SMSAdditionalNodes" → Function[{x}, {Null, Null, Null, Null, Null, Null, x}],
  "SMSNodeID" → {"D", "D -D", "D -D", "D -D", "D -D", "D -D", "D -D", "Lagr -AL -S"},
  "SMSCharSwitch" → {"CTD2N1DN2", "CTD2L1DN1", "ContactElement"},
  "SMSNoTimeStorage" → 4,
  "SMSSymmetricTangent" → False,
  "SMSDefaultIntegrationCode" → 0 ,
  "SMSDomainDataNames" ->
   {"μ -friction coefficient", "ρ -regularization parameter", "t -thickness"},
  "SMSDefaultData" → {0, 1, 1}
 ];
```

In[49]:=
```
(* ============================================================ *)
SMSStandardModule["Nodal information"];
x ⊨ SMSIO["Nodal coordinates"[1, {1, 2}]] + SMSIO["Nodal DOFs"[1, {1, 2}]];
xP ⊨ SMSIO["Nodal coordinates"[1, {1, 2}]] + SMSIO["Nodal DOFs n"[1, {1, 2}]];
SMSExport[{Flatten[{x, xP}]} , d$$];
(* ============================================================ *)
SMSStandardModule["Tangent and residual"];
{μ, ρ, thick} ⊨ SMSIO["Domain data"];
Xiaug ⊨ Table[SMSIO["Nodal coordinates"[i, 1]], {i, 8}];
Yiaug ⊨ Table[SMSIO["Nodal coordinates"[i, 2]], {i, 8}];
uiaug ⊨ Table[SMSIO["Nodal DOFs"[i, 1]], {i, 8}];
viaug ⊨ Table[SMSIO["Nodal DOFs"[i, 2]], {i, 8}];
uiaugP ⊨ Table[SMSIO["Nodal DOFs n"[i, 1]], {i, 8}];
viaugP ⊨ Table[SMSIO["Nodal DOFs n"[i, 2]], {i, 8}];
{ui, vi, uiP, viP, Xi, Yi} ⊨
  Map[Join[{#[[1]]}, Take[#, {4, 7}]] &, {uiaug, viaug, uiaugP, viaugP, Xiaug, Yiaug}];
{λN, λT} ⊨ {uiaug[[8]], viaug[[8]]};
basicVars ⊨ Join[({ui, vi} // Transpose // Flatten), {λN, λT}];
basicVarsP ⊨ Join[({uiP, viP} // Transpose // Flatten), {0, 0}];
index ⊢ SMSArray[{1, 2, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}, "Type" → Integer];
{slaveS1, slaveS2} ⊨
  {{Xiaug⟦2⟧ - Xiaug⟦1⟧, Yiaug⟦2⟧ - Yiaug⟦1⟧}, {Xiaug⟦1⟧ - Xiaug⟦3⟧, Yiaug⟦1⟧ - Yiaug⟦3⟧}};
Area0 ⊨ (SMSSqrt[slaveS1.slaveS1] + SMSSqrt[slaveS2.slaveS2]) thick / 2;
(* If no contact *)
SMSIf[SMSInteger[ns$$[4, "id", "Dummy"]] == 1];
 LagrNNC ⊨ Area0 (-λN^2 / (2 ρ));
 LagrTNC ⊨ Area0 (-λT^2 / (2 ρ));
 LagrNC ⊨ LagrNNC + LagrTNC;
 SMSDo[i, 1, 2];
  ii ⊢ SMSInteger[SMSPart[index, 10 + i]];
   RNC ⊨ SMSD[LagrNC, {λN, λT}, i];
   SMSIO[RNC, "Export to", "Residual"[ii]];
   SMSDo[j, 1, 2];
```

```
   jj ⊢ SMSInteger[SMSPart[index, 10 + j]];
   dRdaNC ⊨ SMSD[ RNC, {λN, λT}, j ];
   SMSIO[dRdaNC, "Export to", "Tangent"[ii, jj]];
  SMSEndDo[];
 SMSEndDo[];
 SMSIO[{0, 0}, "Export to", "Element time dependent data"[{1, 2}]];
SMSElse[];
{ξP, gNP} ⊨ SMSIO["Element time dependent data n"[{1, 2}]];
(* We have to zero the ξP because of local convergence problems in case of contact switch *)
ξP ⊢ SMSReal[0.];
{xS, xM2, xM3, xM1, xM4} ⊨ Transpose[ {Xi + ui, Yi + vi} ];
{L, L31, L24} ⊨ Map[ SMSSqrt[ #.# ] &, {xM3 - xM2, xM3 - xM1, xM2 - xM4} ];
{t31, t24} ⊨ {(xM3 - xM1) / L31, (xM2 - xM4) / L24};
{Bxi, Byi} ⊨ Transpose[ {xM2, xM2 + (L / 3) t31, xM3 + (L / 3) t24, xM3} ];
Clear[ξ, b];
(* Normal and tangential gap *)
localSearch :=
  (fBi = Module[ {m = 4}, Table[ 1/8 Binomial[m - 1, i - 1] (1 + ξ)^(i-1) (1 - ξ)^(m-i), {i, m} ] ];
   ξA ⊢ SMSReal[-1];
   fBiA = Module[ {m = 4}, Table[ 1/8 Binomial[m - 1, i - 1] (1 + ξA)^(i-1) (1 - ξA)^(m-i), {i, m} ] ];
   ξB ⊢ SMSReal[1];
   fBiB = Module[ {m = 4}, Table[ 1/8 Binomial[m - 1, i - 1] (1 + ξB)^(i-1) (1 - ξB)^(m-i), {i, m} ] ];
   SMSIf[ξ < -1];
    xP ⊣ Simplify[{Bxi[[1]], Byi[[1]]} + SMSD[{Bxi, Byi}.fBiA, ξA] (ξ + 1)];
   SMSElse[];
    SMSIf[ξ > 1];
     xP1 ⊣ Simplify[{Bxi[[4]], Byi[[4]]} + SMSD[{Bxi, Byi}.fBiB, ξB] (ξ - 1)];
    SMSElse[];
     xP1 ⊣ Simplify[{Bxi, Byi}.fBi];
    SMSEndIf[xP1];
    xP ⊣ xP1;
   SMSEndIf[xP];
   t ⊨ SMSD[xP, ξ];
   n ⊨ {t[[2]], -t[[1]]} / SMSSqrt[t.t];
   H ⊨ xP + gN n - xS;
   dHdb ⊨ SMSD[H, {ξ, gN}];
   dHdbInv ⊨ SMSInverse[dHdb];
  );
b ⊣ {ξP, gNP};
SMSDo[ iter, 1, 30, 1, b ];
 {ξ, gN} ⊨ b;
 localSearch;
 Δb ⊨ -dHdbInv.H;
 b ⊣ b + Δb;
 SMSIf[Sqrt[Δb.Δb] < 1 / 10^12 || iter == 29];
  SMSBreak[];
 SMSEndIf[];
SMSEndDo[b];
b ⊨ SMSReal[b];
(* Remember to define "b" before using this tag! *)
```

```
{ξ, gN} ⊨ b;
localSearch;
dHda ⊨ SMSD[H, basicVars];
dbda ⊨ -dHdbInv.dHda;
SMSDefineDerivative[b, basicVars, dbda];
(* Augmented multipliers and friction cone radius *)
(* Δξ⊨ξ-ξP; *)
Δξ ⊨ SMSSqrt[t.t] × SMSD[ ξ, basicVars ].(basicVars - basicVarsP);
λNaug ⊨ λN + ρ gN;
λTaug ⊨ λT + ρ Δξ;

SMSIf[SMSIO["Iteration"] == 1];
 {stateN, stateT} ⇉ SMSInteger[SMSIO["Element time dependent data n"[{3, 4}]]];
SMSElse[];
 {stateN, stateT} ↾ SMSInteger[{0, 0}];
SMSEndIf[stateN, stateT];

SMSIf[λNaug ≤ 10^-8 || stateN == 1];
 lagrN ⇉ λNaug;
 SMSIO[1, "Export to", "Element time dependent data"[3]];
SMSElse[];
 lagrN ↾ 0;
 SMSIO[0, "Export to", "Element time dependent data"[3]];
SMSEndIf[lagrN];

kaug ⊢ -μ lagrN;
SMSIf[SMSAbs[λTaug] - kaug ≥ -10^-8 || stateT == 1];
 λTaugkaug ⇉ SMSAbs[λTaug] - kaug;
  SMSIO[1, "Export to", "Element time dependent data"[4]];
SMSElse[];
 λTaugkaug ↾ 0;
  SMSIO[0, "Export to", "Element time dependent data"[4]];
SMSEndIf[λTaugkaug];

(* Augmented Lagrangian of contact *)
LagrN ⊨ Area0 (lagrN^2 - λN^2) / (2 ρ);
LagrT ⊨ Area0 (λT Δξ + ρ Δξ^2 / 2 - (λTaugkaug^2) / (2 ρ));
Lagr ⊨ LagrN + LagrT;
SMSDo[i, 1, 12];
(* Residual *)
Ri ⊨ SMSD[Lagr, basicVars, i, "Constant" → kaug];
(* Tangent and export *)
ii ⊢ SMSInteger[SMSPart[index, i]];
SMSIO[Ri, "Export to", "Residual"[ii]];
SMSDo[ j, 1, 12];
dRidaj ⊨ SMSD[Ri, basicVars, j];
jj ⊢ SMSInteger[SMSPart[index, j]];
SMSIO[dRidaj, "Export to", "Tangent"[ii, jj]];
SMSEndDo[];
SMSEndDo[];
SMSIO[{ξ, gN}, "Export to", "Element time dependent data"[{1, 2}]];
SMSEndIf[];

SMSWrite[];
```

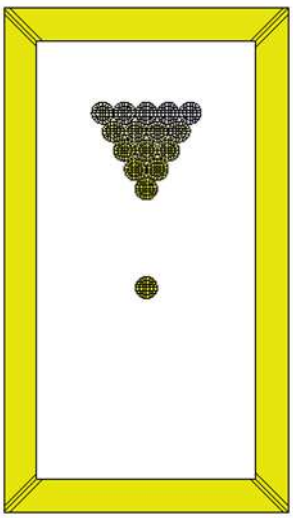
File: ExamplesCTD2N1DN2L1DN1AL.c Size: 58450 Time: 36

Method	PAN	SKR
No.Formulae	3	1548
No.Leafs	84	16987

2D snooker simulation

Description

Use the contact element from 2 D slave node and smooth master segment element and hyper-solid dynamic element from Solid, Finite Strain Element for Dynamic Analysis and analyze the single snooker shoot.



Analysis

In[224]:=
```
<< AceFEM`;
δ = 0.60; (* 0.5 for non-dissipative model *)
β = (δ + 0.5)/2)^2;
bx = 1;
by = 2 bx;
bx2 = bx + 0.3 bx;
by2 = by + 0.3 bx;
db = 0.05 bx;
R = 0.1 bx;
v0x = -0.14;
v0y = 1;
Nb = 6;
TN = 5;
For[BN = 0; i = 1, i <= TN, i++, BN = BN + i];
ball[X0_, Y0_, R_, Nb_, domain_, body_, boundary_] :=
  SMTAddMesh[Raster[Table[{X0, Y0} + R {i, j}/Sqrt[1 + Min[Abs[{i, j}]]^2/Max[Abs[{i, j}]]^2],
      {j, -1, 1, 2/5}, {i, -1, 1, 2/5}]], domain,
   "Q1", {Nb, Nb}, "BodyID" → body, "BoundaryDomainID" → boundary];
SMTInputData[];
SMTAddDomain[
  {"b", "ExamplesHypersolidDynNewmark",
   {"E *" -> 400., "ν *" -> 0.3, "ρ0 *" -> 2, "β *" -> β, "δ *" -> δ}},
  {"c", "ExamplesCTD2N1DN2L1DN1AL", {"μ *" -> 0.1, "ρ *" -> 100.}}];
```

```
Table[
  SMTAddDomain["s" <> ToString[i - 1], "ExamplesHypersolidDynNewmark",
   {"E *" -> 50000., "ν *" -> 0.3, "ρ0 *" -> 2, "β *" -> β, "δ *" -> δ}]
  , {i, BN + 1}];
SMTAddMesh[
  Raster[{{{-bx2, -by2}, {-bx2 + db, -by2}, {bx2 - db, -by2}, {bx2, -by2}, {bx2, -by2 + db},
     {bx2, by2 - db}, {bx2, by2}, {bx2 - db, by2}, {-bx2 + db, by2}, {-bx2, by2},
     {-bx2, by2 - db}, {-bx2, -by2 + db}, {-bx2, -by2}},
    {{-bx, -by}, {-bx + db, -by}, {bx - db, -by}, {bx, -by}, {bx, -by + db}, {bx, by - db}, {bx, by},
     {bx - db, by}, {-bx + db, by}, {-bx, by}, {-bx, by - db}, {-bx, -by + db}, {-bx, -by}}}], "b",
  "Q1", {12, 1}, "BodyID" → "C", "BoundaryDomainID" -> "c", "InterpolationOrder" → 1];
ball[0, -by/8, R, Nb, "s0", "B0", "c"];
For[i = 1;
 bnum = 0, i ≤ TN, i++, For[j = 0, j < i, j++, bnum += 1;
  ball[-R (i - 1) + 2 j R, by/2 + (i - (TN + 1)/2) R √3,
   R, Nb, "s" <> ToString[bnum], "B" <> ToString[bnum], "c"]]]
SMTAddEssentialBoundary[Abs["Y"] >= by || Abs["X"] ≥ bx &, 1 -> 0., 2 -> 0.];
SMTAnalysis[];
SMTRData["ContactSearchTolerance", 2 R (1 - Cos[π/(4 Nb)])];
SMTIData["Contact1", 0];
SMTElementData[SMTFindElements["s0"], "ht",
  Table[{v0x, v0y, 0, 0, v0x, v0y, 0, 0, v0x, v0y, 0, 0, v0x, v0y, 0, 0},
   {Length[SMTFindElements["s0"]]}]];
SMTShowMesh["Show" → "Window", "Domains" → Table["s" <> ToString[i - 1], {i, BN + 1}],
  "User" → {Line[{{bx, by}, {bx, -by}, {-bx, -by}, {-bx, by}, {bx, by}}]}];
SMTNextStep["t" -> 0.0025];
While[
  While[step = SMTConvergence[10^-8, 15,
     {"Adaptive Time", 8, .0000001, 0.05, 5}, "Alternate" → True], SMTNewtonIteration[]; ];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  , If[step[[1]], SMTStepBack[], SMTShowMesh["DeformedMesh" → True, "Show" → "Window",
     "Domains" → Table["s" <> ToString[i - 1], {i, BN + 1}],
     "User" → {Line[{{bx, by}, {bx, -by}, {-bx, -by}, {-bx, by}, {bx, by}}]}];
  ];
  SMTNextStep["Δt" -> step[[2]]]
 ];
```

3D slave node - triangle master segment element

The support for the analysis of contact problems in AceFEM and implementation of contact finite elements in AceGen is described in section Implementation Notes for Contact Elements.

In[147]:=
```
<< "AceGen`";
SMSInitialize["ExamplesCTD3V1P1", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "V3", "SMSNoNodes" → 4,
  "SMSSymmetricTangent" → True, "SMSDOFGlobal" → {3, 3, 3, 3}, "SMSSegments" → {},
  "SMSAdditionalNodes" → Null, "SMSNodeID" → {"D", "D -D", "D -D", "D -D"},
  "SMSCharSwitch" → {"CTD3V1", "CTD3P1", "ContactElement"},
  "SMSDomainDataNames" -> {"ρ -penalty parameter"},
  "SMSDefaultData" -> {1000}
 ];

SMSStandardModule["Nodal information"];
x ⊨ SMSIO["Nodal coordinates"[1, {1, 2, 3}]] + SMSIO["Nodal DOFs"[1, {1, 2, 3}]];
xP ⊨ SMSIO["Nodal coordinates"[1, {1, 2, 3}]] + SMSIO["Nodal DOFs n"[1, {1, 2, 3}]];
SMSExport[{Flatten[{x, xP}]}, d$$];
```

In[154]:=
```
SMSStandardModule["Tangent and residual"];
(*Read and set element values*)
{ρ} ⊨ SMSIO["Domain data"];
Xi ⊨ Table[SMSIO["Nodal coordinates"[i, 1]], {i, 4}];
Yi ⊨ Table[SMSIO["Nodal coordinates"[i, 2]], {i, 4}];
Zi ⊨ Table[SMSIO["Nodal coordinates"[i, 3]], {i, 4}];
ui ⊨ Table[SMSIO["Nodal DOFs"[i, 1]], {i, 4}];
vi ⊨ Table[SMSIO["Nodal DOFs"[i, 2]], {i, 4}];
wi ⊨ Table[SMSIO["Nodal DOFs"[i, 3]], {i, 4}];
uiP ⊨ Table[SMSIO["Nodal DOFs n"[i, 1]], {i, 4}];
viP ⊨ Table[SMSIO["Nodal DOFs n"[i, 2]], {i, 4}];
wiP ⊨ Table[SMSIO["Nodal DOFs n"[i, 3]], {i, 4}];
basicVars ⊨ ({ui, vi, wi} // Transpose // Flatten);
{xS, x1, x2, x3} ⊨ Transpose[{Xi + ui, Yi + vi, Zi + wi}];
(* If contact possible *)
SMSIf[SMSInteger[ns$$[2, "id", "Dummy"]] ≠ 1];
(* projection point and gap distance *)
localSearch := (
    xP ⊨ {ξ, η, 1 - ξ - η}.{x1, x2, x3};
    τξ ⊨ SMSD[xP, ξ];
    τη ⊨ SMSD[xP, η];
    τn ⊨ Cross[τξ, τη];
    n ⊨ τn / SMSSqrt[τn.τn];
    H ⊨ xP + gN n - xS;
    dHdb ⊨ SMSD[H, b];
    dHdbInv ⊨ SMSInverse[dHdb]
   );

b ⊣ SMSReal[{0, 0, 0}];
SMSDo[ iter, 1, 30, 1, b ];
 {ξ, η, gN} ⊨ b;
 localSearch;
 Δb ⊨ -dHdbInv.H;
 b ⊣ b + Δb;
 SMSIf[Sqrt[Δb.Δb] < 1 / 10^12 || iter == 29];
  SMSBreak[];
 SMSEndIf[];
SMSEndDo[b];

b ⊢ SMSReal[b];
```

```
(* Remember to define "b" before using this tag! *)
{ξ, η, gN} ⊨ b;
localSearch;
dHda ⊨ SMSD[H, basicVars];
dbda ⊨ -dHdbInv.dHda;
SMSDefineDerivative[b, basicVars, dbda];
SMSIf[gN ≤ 0];
 Π ⊰ 1/2 ρ gN^2;
SMSElse[];
 Π ⊰ 0;
SMSEndIf[Π];

SMSDo[i, 1, 12];
 (* Residual *)
 Ri ⊨ SMSD[Π, basicVars, i];
 SMSIO[Ri, "Export to", "Residual"[i]];
 SMSDo[ j, i, 12];
  dRidaj ⊨ SMSD[Ri, basicVars, j];
  SMSIO[dRidaj, "Export to", "Tangent"[i, j]];
 SMSEndDo[];
SMSEndDo[];

SMSEndIf[];
SMSWrite[];
```

File: ExamplesCTD3V1P1.c Size: 15347 Time: 9

Method	PAN	SKR
No.Formulae	4	231
No.Leafs	125	4498

3D slave node - quadrilateral master segment element

The support for the analysis of contact problems in *AceFEM* and implementation of contact finite elements in *AceGen* is described in section Implementation Notes for Contact Elements .

In[200]:=
```
<< "AceGen`";
SMSInitialize["ExamplesCTD3V1S1", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "V3", "SMSNoNodes" → 5,
  "SMSSymmetricTangent" → True, "SMSDOFGlobal" → {3, 3, 3, 3, 3}, "SMSSegments" → {},
  "SMSAdditionalNodes" → Null, "SMSNodeID" → {"D", "D -D", "D -D", "D -D", "D -D"},
  "SMSCharSwitch" → {"CTD3V1", "CTD3S1", "ContactElement"},
  "SMSDomainDataNames" -> {"ρ -penalty parameter"},
  "SMSDefaultData" -> {1000}];

SMSStandardModule["Nodal information"];
x ⊨ SMSIO["Nodal coordinates"[1, {1, 2, 3}]] + SMSIO["Nodal DOFs"[1, {1, 2, 3}]];
xP ⊨ SMSIO["Nodal coordinates"[1, {1, 2, 3}]] + SMSIO["Nodal DOFs n"[1, {1, 2, 3}]];
SMSExport[{Flatten[{x, xP}]} , d$$];

SMSStandardModule["Tangent and residual"];
(*Read and set element values*)
{ρ} ⊨ SMSIO["Domain data"];
Xi ⊨ Table[SMSIO["Nodal coordinates"[i, 1]], {i, 5}];
Yi ⊨ Table[SMSIO["Nodal coordinates"[i, 2]], {i, 5}];
```

```
Zi = Table[SMSIO["Nodal coordinates"[i, 3]], {i, 5}];
ui = Table[SMSIO["Nodal DOFs"[i, 1]], {i, 5}];
vi = Table[SMSIO["Nodal DOFs"[i, 2]], {i, 5}];
wi = Table[SMSIO["Nodal DOFs"[i, 3]], {i, 5}];
uiP = Table[SMSIO["Nodal DOFs n"[i, 1]], {i, 5}];
viP = Table[SMSIO["Nodal DOFs n"[i, 2]], {i, 5}];
wiP = Table[SMSIO["Nodal DOFs n"[i, 3]], {i, 5}];
basicVars = ({ui, vi, wi} // Transpose // Flatten);
{xS, x1, x2, x3, x4} = Transpose[{Xi + ui, Yi + vi, Zi + wi}];
(* If contact possible *)
SMSIf[SMSInteger[ns$$[2, "id", "Dummy"]] ≠ 1];

(* projection point and gap distance *)
localSearch := (
   xP = {ξ η, (1 - ξ) η, (1 - ξ) (1 - η), ξ (1 - η)}.{x1, x2, x3, x4};
   τξ = SMSD[xP, ξ];
   τη = SMSD[xP, η];
   τn = Cross[τξ, τη];
   n = τn / SMSSqrt[τn.τn];
   H = xP + gN n - xS;
   dHdb = SMSD[H, b];
   dHdbInv = SMSInverse[dHdb]
  );

b := SMSReal[{0, 0, 0}];
SMSDo[ iter, 1, 30, 1, b ];
{ξ, η, gN} = b;
localSearch;
Δb = -dHdbInv.H;
b = b + Δb;
SMSIf[Sqrt[Δb.Δb] < 1 / 10^12 || iter == 29];
SMSBreak[];
SMSEndIf[];
SMSEndDo[b];

b = SMSReal[b];
(* Remember to define "b" before using this tag! *)
{ξ, η, gN} = b;
localSearch;
dHda = SMSD[H, basicVars];
dbda = -dHdbInv.dHda;
SMSDefineDerivative[b, basicVars, dbda];

SMSIf[gN ≤ 0];
   1
Π := - ρ gN^2;
   2
SMSElse[];
Π = 0;
SMSEndIf[Π];

SMSDo[i, 1, 15];
(* Residual *)
Ri = SMSD[Π, basicVars, i];
SMSIO[Ri, "Export to", "Residual"[i]];
SMSDo[ j, i, 15];
```

```
dRidaj = SMSD[Ri, basicVars, j];
SMSIO[dRidaj, "Export to", "Tangent"[i, j]];
SMSEndDo[];
SMSEndDo[];

SMSEndIf[];

SMSWrite[];
```

File: ExamplesCTD3V1S1.c Size: 23613 Time: 17

Method	PAN	SKR
No.Formulae	4	407
No.Leafs	125	7764

3D slave node - quadrilateral master segment and 2 neighboring nodes element

The support for the analysis of contact problems in AceFEM and implementation of contact finite elements in AceGen is described in section Implementation Notes for Contact Elements.

Neighboring nodes are not used in the element.

In[253]:=
```
<< "AceGen`";
SMSInitialize["ExamplesCTD3V1S1DN2", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "V3", "SMSNoNodes" → 13,
  "SMSDOFGlobal" → {3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3},
  "SMSSegments" → {}, "SMSAdditionalNodes" → Null,
  "SMSNodeID" → {"D", "D -D", "D -D", "D -D", "D -D", "D -D",
    "D -D", "D -D", "D -D", "D -D", "D -D", "D -D", "D -D"},
  "SMSCharSwitch" → {"CTD3V1", "CTD3S1DN2", "ContactElement"}, "SMSSymmetricTangent" → True,
  "SMSDomainDataNames" -> {"ρ -penalty parameter"},
  "SMSDefaultData" -> {1000}];

SMSStandardModule["Nodal information"];
x = SMSIO["Nodal coordinates"[1, {1, 2, 3}]] + SMSIO["Nodal DOFs"[1, {1, 2, 3}]];
xP = SMSIO["Nodal coordinates"[1, {1, 2, 3}]] + SMSIO["Nodal DOFs n"[1, {1, 2, 3}]];
SMSExport[{Flatten[{x, xP}]}, d$$];

SMSStandardModule["Tangent and residual"];
(*Read and set element values*)
{ρ} = SMSIO["Domain data"];
Xi = Table[SMSIO["Nodal coordinates"[i, 1]], {i, 5}];
Yi = Table[SMSIO["Nodal coordinates"[i, 2]], {i, 5}];
Zi = Table[SMSIO["Nodal coordinates"[i, 3]], {i, 5}];
ui = Table[SMSIO["Nodal DOFs"[i, 1]], {i, 5}];
vi = Table[SMSIO["Nodal DOFs"[i, 2]], {i, 5}];
wi = Table[SMSIO["Nodal DOFs"[i, 3]], {i, 5}];
uiP = Table[SMSIO["Nodal DOFs n"[i, 1]], {i, 5}];
viP = Table[SMSIO["Nodal DOFs n"[i, 2]], {i, 5}];
wiP = Table[SMSIO["Nodal DOFs n"[i, 3]], {i, 5}];
basicVars = ({ui, vi, wi} // Transpose // Flatten);
{xS, x1, x2, x3, x4} = Transpose[{Xi + ui, Yi + vi, Zi + wi}];
(* If contact possible *)
SMSIf[SMSInteger[ns$$[2, "id", "Dummy"]] ≠ 1];

(* projection point and gap distance *)
localSearch := (
```

```
    xP ⊨ {ξ η, (1 - ξ) η, (1 - ξ) (1 - η), ξ (1 - η)}.{x1, x2, x3, x4};
    τξ ⊨ SMSD[xP, ξ];
    τη ⊨ SMSD[xP, η];
    τn ⊨ Cross[τξ, τη];
    n ⊨ τn / SMSSqrt[τn.τn];
    H ⊨ xP + gN n - xS;
    dHdb ⊨ SMSD[H, b];
    dHdbInv ⊨ SMSInverse[dHdb]
   );

b ⊣ SMSReal[{0, 0, 0}];
SMSDo[ iter, 1, 30, 1, b ];
{ξ, η, gN} ⊨ b;
localSearch;
Δb ⊨ -dHdbInv.H;
b ⊣ b + Δb;
SMSIf[Sqrt[Δb.Δb] < 1 / 10^12 || iter == 29];
SMSBreak[];
SMSEndIf[];
SMSEndDo[b];

b ⊢ SMSReal[b];
(* Remember to define "b" before using this tag! *)
{ξ, η, gN} ⊨ b;
localSearch;
dHda ⊨ SMSD[H, basicVars];
dbda ⊨ -dHdbInv.dHda;
SMSDefineDerivative[b, basicVars, dbda];

SMSIf[gN ≤ 0];
Π ⊣ 1/2 ρ gN^2;
SMSElse[];
Π ⊣ 0;
SMSEndIf[Π];

SMSDo[i, 1, 15];
(* Residual *)
Ri ⊨ SMSD[Π, basicVars, i];
SMSIO[Ri, "Export to", "Residual"[i]];
SMSDo[ j, i, 15];
dRidaj ⊨ SMSD[Ri, basicVars, j];
SMSIO[dRidaj, "Export to", "Tangent"[i, j]];
SMSEndDo[];
SMSEndDo[];

SMSEndIf[];

SMSWrite[];
```

File: ExamplesCTD3V1S1DN2.c Size: 23852 Time: 17

Method	PAN	SKR
No.Formulae	4	407
No.Leafs	125	7764

3D slave triangle and 2 neighboring nodes - triangle master segment element

The support for the analysis of contact problems in *AceFEM* and implementation of contact finite elements in *AceGen* is described in section Implementation Notes for Contact Elements.

Neighboring nodes are not used in the element.

In[306]:=
```
<< "AceGen`";
SMSInitialize["ExamplesCTD3P1DN2P1", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "P1", "SMSNoNodes" → 18,
  "SMSDOFGlobal" → {3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3}, "SMSSegments" → {},
  "SMSAdditionalNodes" → Null, "SMSNodeID" → {"D", "D", "D", "D -D", "D -D", "D -D", "D -D",
    "D -D", "D -D", "D -D", "D -D", "D -D", "D -D", "D -D", "D -D", "D -D", "D -D"},
  "SMSCharSwitch" → {"CTD3P1DN2", "CTD3P1", "ContactElement"}, "SMSSymmetricTangent" → True,
  "SMSDomainDataNames" -> {"ρ -penalty parameter"},
  "SMSDefaultData" -> {1000}];

SMSStandardModule["Nodal information"];
For[i = 1, i ≤ 3, i++,
  x ⊨ SMSIO["Nodal coordinates"[i, {1, 2, 3}]] + SMSIO["Nodal DOFs"[i, {1, 2, 3}]];
  xP ⊨ SMSIO["Nodal coordinates"[i, {1, 2, 3}]] + SMSIO["Nodal DOFs n"[i, {1, 2, 3}]];
  SMSExport[Flatten[{x, xP}] , d$$[i, #] &]
 ];

SMSStandardModule["Tangent and residual"];
(*Read and set element values*)
{ρ} ⊨ SMSIO["Domain data"];
Xi ⊨ Table[SMSIO["Nodal coordinates"[i, 1]], {i, SMSNoNodes}];
Yi ⊨ Table[SMSIO["Nodal coordinates"[i, 2]], {i, SMSNoNodes}];
Zi ⊨ Table[SMSIO["Nodal coordinates"[i, 3]], {i, SMSNoNodes}];
ui ⊨ Table[SMSIO["Nodal DOFs"[i, 1]], {i, SMSNoNodes}];
vi ⊨ Table[SMSIO["Nodal DOFs"[i, 2]], {i, SMSNoNodes}];
wi ⊨ Table[SMSIO["Nodal DOFs"[i, 3]], {i, SMSNoNodes}];
uiP ⊨ Table[SMSIO["Nodal DOFs n"[i, 1]], {i, SMSNoNodes}];
viP ⊨ Table[SMSIO["Nodal DOFs n"[i, 2]], {i, SMSNoNodes}];
wiP ⊨ Table[SMSIO["Nodal DOFs n"[i, 3]], {i, SMSNoNodes}];
dis ⊨ Transpose[{ui, vi, wi}];
basicVars ⊨ (dis // Flatten);
allX ⊨ Transpose[{Xi + ui, Yi + vi, Zi + wi}];
index = {Join[{1, 2, 3}, Range[28, 36]],
   Join[{4, 5, 6}, Range[37, 45]], Join[{7, 8, 9}, Range[46, 54]]};
For[id = 0, id ≤ 2, id++,
  {xS, x1, x2, x3} ⊨ Map[allX[[#]] &, {1 + id, 10 + 3 id, 11 + 3 id, 12 + 3 id}];
  localBasic ⊨ Flatten[Map[dis[[#]] &, {1 + id, 10 + 3 id, 11 + 3 id, 12 + 3 id}]];
  localIndex ⊢ SMSInteger[index[[id + 1]]];
  (* If contact possible *)
  SMSIf[SMSInteger[ns$$[10 + 3 id, "id", "Dummy"]] ≠ 1];
  (* projection point and gap distance *)
  localSearch :=
   (
    xP ⊨ {ξ, η, 1 - ξ - η}.{x1, x2, x3};
    τξ ⊨ SMSD[xP, ξ];
    τη ⊨ SMSD[xP, η];
    τn ⊨ Cross[τξ, τη];
    n ⊨ τn / SMSSqrt[τn.τn];
```

```
    H ⊨ xP + gN n - xS;
    dHdb ⊨ SMSD[H, b];
    dHdbInv ⊨ SMSInverse[dHdb]
   );
  b ⊣ SMSReal[{0, 0, 0}];
  SMSDo[iter, 1, 30, 1, b];
   {ξ, η, gN} ⊨ b;
   localSearch;
   Δb ⊨ -dHdbInv.H;
   b ⊣ b + Δb;
   SMSIf[Sqrt[Δb.Δb] < 1/10^12 || iter == 29];
    SMSBreak[];
   SMSEndIf[];
  SMSEndDo[b];
  b ⊢ SMSReal[b];
  (* Remember to define "b" before using this tag! *)
  {ξ, η, gN} ⊨ b;
  localSearch;
  dHda ⊨ SMSD[H, localBasic];
  dbda ⊨ -dHdbInv.dHda;
  SMSDefineDerivative[b, localBasic, dbda];
  SMSIf[gN ≤ 0];
   Π ⊣ 1/2 ρ gN^2;
  SMSElse[];
   Π ⊣ 0;
  SMSEndIf[Π];
  SMSDo[i, 1, Length[localBasic]];
   (* Residual *)
   Ri ⊨ SMSD[Π, localBasic, i];
   ii ⊢ SMSInteger[SMSPart[localIndex, i]];
   SMSIO[Ri, "Add to", "Residual"[ii]];
   SMSDo[j, i, Length[localBasic]];
    dRidaj ⊨ SMSD[Ri, localBasic, j];
    jj ⊢ SMSInteger[SMSPart[localIndex, j]];
    SMSIO[dRidaj, "Add to", "Tangent"[ii, jj]];
   SMSEndDo[];
  SMSEndDo[];

  SMSEndIf[]
];
SMSWrite[];
```

```
File: ExamplesCTD3P1DN2P1.c  Size: 41438  Time: 34
```

Method	PAN	SKR
No.Formulae	12	697
No.Leafs	375	13716

3D slave triangle - triangle master segment and 2 neighboring nodes element

The support for the analysis of contact problems in *AceFEM* and implementation of contact finite elements in *AceGen* is described in section Implementation Notes for Contact Elements.

Neighboring nodes are not used in the element.

In[328]:=
```
<< "AceGen`";
SMSInitialize["ExamplesCTD3P1P1DN2", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "P1", "SMSNoNodes" → 30,
  "SMSDOFGlobal" → {3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
    3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3}, "SMSSegments" → {},
  "SMSAdditionalNodes" → Null,
  "SMSNodeID" → {"D", "D", "D", "D -D", "D -D", "D -D", "D -D", "D -D", "D -D", "D -D",
    "D -D", "D -D", "D -D", "D -D", "D -D", "D -D", "D -D", "D -D", "D -D", "D -D",
    "D -D", "D -D", "D -D", "D -D", "D -D", "D -D", "D -D", "D -D", "D -D", "D -D"},
  "SMSCharSwitch" → {"CTD3P1", "CTD3P1DN2", "ContactElement"}, "SMSSymmetricTangent" → True,
  "SMSDomainDataNames" -> {"ρ -penalty parameter"}, "SMSDefaultData" -> {1000}];

SMSStandardModule["Nodal information"];
For[i = 1, i ≤ 3, i++,
  x ⊨ SMSIO["Nodal coordinates"[i, {1, 2, 3}]] + SMSIO["Nodal DOFs"[i, {1, 2, 3}]];
  xP ⊨ SMSIO["Nodal coordinates"[i, {1, 2, 3}]] + SMSIO["Nodal DOFs n"[i, {1, 2, 3}]];
  SMSExport[Flatten[{x, xP}], d$$[i, #] &];
 ];

SMSStandardModule["Tangent and residual"];
(*Read and set element values*)
{ρ} ⊨ SMSIO["Domain data"];
Xi ⊨ Table[SMSIO["Nodal coordinates"[i, 1]], {i, SMSNoNodes}];
Yi ⊨ Table[SMSIO["Nodal coordinates"[i, 2]], {i, SMSNoNodes}];
Zi ⊨ Table[SMSIO["Nodal coordinates"[i, 3]], {i, SMSNoNodes}];
ui ⊨ Table[SMSIO["Nodal DOFs"[i, 1]], {i, SMSNoNodes}];
vi ⊨ Table[SMSIO["Nodal DOFs"[i, 2]], {i, SMSNoNodes}];
wi ⊨ Table[SMSIO["Nodal DOFs"[i, 3]], {i, SMSNoNodes}];
uiP ⊨ Table[SMSIO["Nodal DOFs n"[i, 1]], {i, SMSNoNodes}];
viP ⊨ Table[SMSIO["Nodal DOFs n"[i, 2]], {i, SMSNoNodes}];
wiP ⊨ Table[SMSIO["Nodal DOFs n"[i, 3]], {i, SMSNoNodes}];
dis ⊨ Transpose[{ui, vi, wi}];
basicVars ⊨ (dis // Flatten);
allX ⊨ Transpose[{Xi + ui, Yi + vi, Zi + wi}];
index = {Join[{1, 2, 3}, Range[10, 18]],
    Join[{4, 5, 6}, Range[37, 45]], Join[{7, 8, 9}, Range[64, 72]]};

For[id = 0, id ≤ 2, id++,
  {xS, x1, x2, x3} ⊨ Map[allX[[#]] &, {1 + id, 4 + 9 id, 5 + 9 id, 6 + 9 id}];
  localBasic ⊨ Flatten[Map[dis[[#]] &, {1 + id, 4 + 9 id, 5 + 9 id, 6 + 9 id}]];
  localIndex ⊢ SMSInteger[index[[id + 1]]];
  (* If contact possible *)
  SMSIf[SMSInteger[ns$$[4 + 9 id, "id", "Dummy"]] ≠ 1];

  (* projection point and gap distance *)
  localSearch := (
    xP ⊨ {ξ, η, 1 - ξ - η}.{x1, x2, x3};
    τξ ⊨ SMSD[xP, ξ];
    τη ⊨ SMSD[xP, η];
    τn ⊨ Cross[τξ, τη];
    n ⊨ τn / SMSSqrt[τn.τn];
    H ⊨ xP + gN n - xS;
    dHdb ⊨ SMSD[H, b];
    dHdbInv ⊨ SMSInverse[dHdb]
```

```
  );

 b = SMSReal[{0, 0, 0}];
 SMSDo[ iter, 1, 30, 1, b];
  {ξ, η, gN} = b;
  localSearch;
  Δb = -dHdbInv.H;
  b = b + Δb;
  SMSIf[Sqrt[Δb.Δb] < 1 / 10^12 || iter == 29];
   SMSBreak[];
  SMSEndIf[];
 SMSEndDo[b];
 b = SMSReal[b];
 {ξ, η, gN} = b; (* Remember to define "b" before using this tag! *)
 localSearch;
 dHda = SMSD[H, localBasic];
 dbda = -dHdbInv.dHda;
 SMSDefineDerivative[b, localBasic, dbda];
 SMSIf[gN ≤ 0];
  Π = 1/2 ρ gN^2;
 SMSElse[];
  Π = 0;
 SMSEndIf[Π];
 SMSDo[i, 1, Length[localBasic]];
  Ri = SMSD[Π, localBasic, i];
  (* Residual *)
  ii = SMSInteger[SMSPart[localIndex, i]];
  SMSIO[Ri, "Add to", "Residual"[ii]];
  SMSDo[j, i, Length[localBasic]];
   dRidaj = SMSD[Ri, localBasic, j];
   jj = SMSInteger[SMSPart[localIndex, j]];
   SMSIO[dRidaj, "Add to", "Tangent"[ii, jj]];
  SMSEndDo[];
 SMSEndDo[];

 SMSEndIf[];
];

SMSWrite[];
```

```
File: ExamplesCTD3P1P1DN2.c  Size: 42605  Time: 35
```

Method	PAN	SKR
No.Formulae	12	697
No.Leafs	375	13716

3D contact analysis

The use quadrilateral 3D contact element to analyze the 3D indentation problem: small elastic box pressed down into large elastic box.

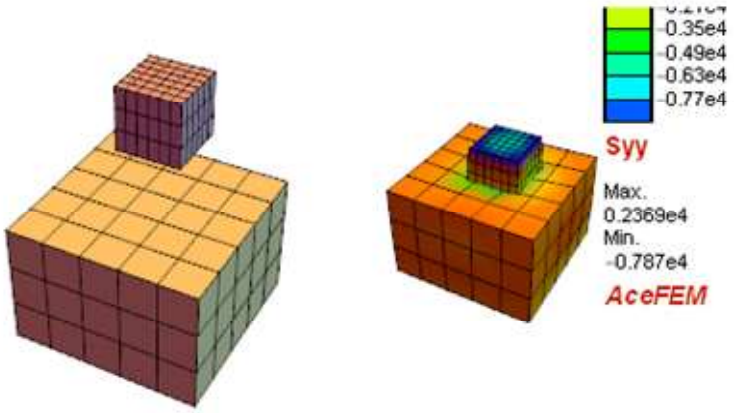
The support for the analysis of contact problems in *AceFEM* and implementation of contact finite elements in *AceGen* is described in section Implementation Notes for Contact Elements .

Simulation

In[309]:=
```
<< AceFEM`;
SMTInputData[];
SMTAddDomain[{"Dom1", {"ML:", "SE", "D3", "H1", "DF", "HY", "H1", "D", {{"NeoHooke", "WA"}}},
    {"E *" -> 70000., "ν *" -> 0.3}},
  {"Dom2", {"ML:", "SE", "D3", "H1", "DF", "HY", "H1", "D", {{"NeoHooke", "WA"}}},
    {"E *" -> 70000., "ν *" -> 0.3}},
  {"c", "ExamplesCTD3V1S1DN2", {"ρ *" -> 10000}}];
SMTAddMesh[Hexahedron[{{0.1, 0.2, 0}, {1.1, 0.2, 0}, {1.1, 1.2, 0},
    {0.1, 1.2, 0}, {0.1, 0.2, 1}, {1.1, 0.2, 1}, {1.1, 1.2, 1}, {0.1, 1.2, 1}}],
  "Dom1", "Division" -> {6, 6, 3}, "BodyID" → "B1", "BoundaryDomainID" → "c"];
SMTAddMesh[Hexahedron[{{-1, -1, -3}, {2, -1, -3}, {2, 2, -3},
    {-1, 2, -3}, {-1, -1, -1}, {2, -1, -1}, {2, 2, -1}, {-1, 2, -1}}],
  "Dom2", "Division" -> {5, 5, 3}, "BodyID" → "B2", "BoundaryDomainID" → "c"];
SMTAddEssentialBoundary[{"Z" == 1 &, 1 -> 0, 2 -> 0, 3 -> -1}, {"Z" ≤ -3 &, 1 -> 0, 2 -> 0, 3 -> 0}];
SMTAnalysis[];
SMTRData["ContactSearchTolerance", 0.1];
```

In[1167]:=
```
Map[(SMTNextStep["Δλ" → #];
    While[SMTConvergence[], SMTNewtonIteration[];]) &, {1, 0.1, 0.1, 0.1, 0.1}];
```

In[1168]:=
```
GraphicsRow[{SMTShowMesh[], SMTShowMesh["DeformedMesh" → True, "Field" -> "Syy"]}]
```



Isogeometric Formulations

Contents

Implementation Notes for Isogeometric Elements

Isogeometric analysis bridges the gap between fields of Computer Aided Design (CAD) and Computer Aided Engineering (CAE) by unifying geometrical description. Isogeometric analysis is based on most widely used CAD representation, the NURBS (Non-Uniform Rational B-Splines) functions. Hence, the same basis functions are used for the design and the analysis model. In NURBS-based analysis there are two notions of meshes, the control mesh and the physical mesh. The control points define the control mesh, which is the convex hull of the actual geometry. Hence, the control mesh controls the actual geometry. The control variables are the degrees of freedom and they are located at the control points. The physical mesh is a decomposition of the actual geometry. It has two representations, one in parent space and one in physical space. In two dimensions is the parent space a rectangle, in three dimensions it is a cuboid. Elements in the physical mesh are defined by knot spans. Knots are points, lines, and surfaces in one-, two-, and three-dimensional topologies, respectively. Knot spans define element domains, where NURBS-basis functions are smooth (*i.e.,* C^{∞}). Across knots, NURBS basis functions will be C^{p-m}, where p is the base function order and m is the multiplicity of the knot in question. NURBS functions are built from B-splines that are local to parameter space (not to the elements like in FEA-"Finite Element Analysis") and defined by knot vectors.

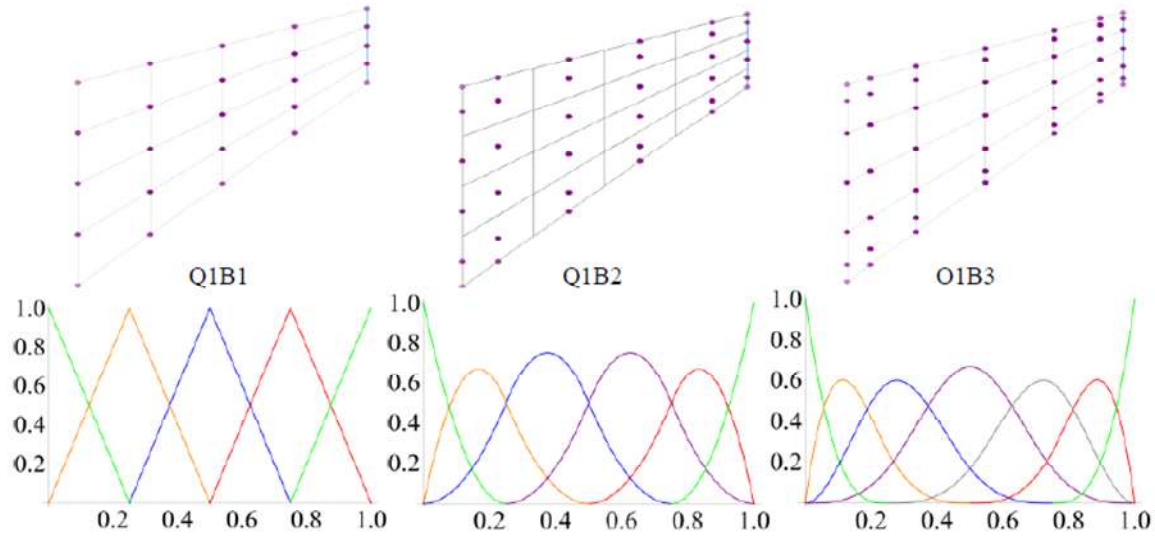
Recursive formula to determine the basis functions

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi)$$

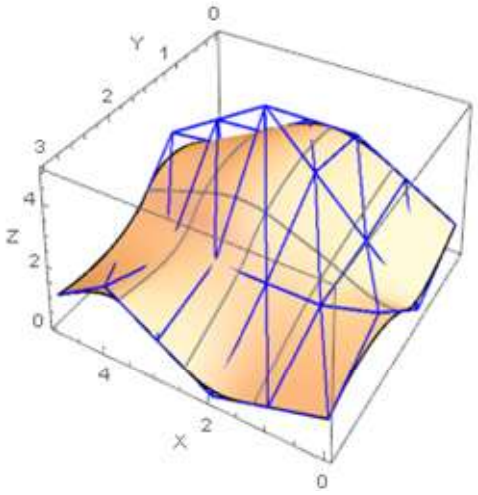
Parametric representation of a NURBS surface

$$\mathbf{S}(\xi,\eta) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} N_{i,p}(\xi) M_{j,q}(\eta) w_{i,j}}{\sum_{k=1}^{n}\sum_{l=1}^{m} N_{k,p}(\xi) M_{l,q}(\eta) w_{k,l}} \mathbf{P}_{i,j}$$

Plot of basis functions



NURBS surface with its control polygon



Input Data for Isogeometric Analysis

SMTIsogeometricMesh

SMTIsogeometricMesh["dID", *basis_function_order*, *knot_vectors*, *refined_knot_vectors*, *nodes_of_the_control_poligon*, *weights_in_the_nodes*]
construct structured isogeometric mesh of elements with the domain identification *dID*

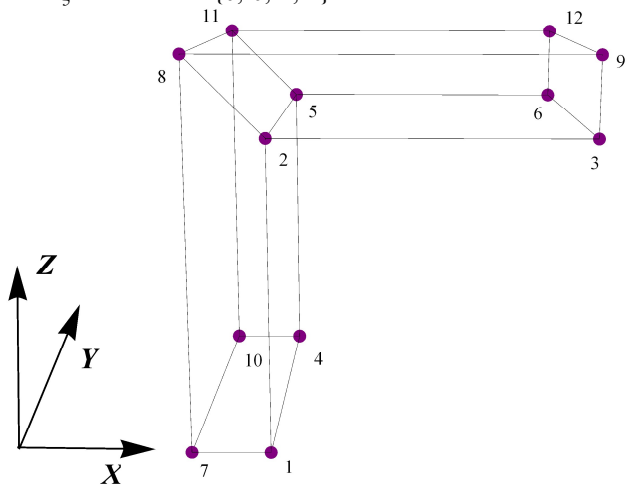
See also: `Implementation Notes for Isogeometric Elements`

The function creates mesh for isogeometric analysis and adds new nodes and elements to the *SMTNodes* and *SMTElements* input data arrays (see `Input Data Phase`). The *basis_function_order* parameter defines the order of the Bezier base functions for modelling the geometry of the problem with NURBS. The *nodes_of_the_control_poligon* refers to the control nodes of the control mesh for the NURBS geometry model, together with *knot_vector_in_ξ_direction, knot_vector_in_η_direction* and *knot_vector_in_ζ_direction* and *weights_in_the_nodes* of the control polygon. For correct insertion of the control nodes remember to put first the nodes in ξ direction followed by the nodes in η direction and the nodes in ζ direction (see examples below). The control mesh is refined with knot insertion. *Refined_knot_vector_in_ξ_direction, refined_knot_vector_in_η_direction, refined_knot_vector_in_ζ_direction* refer to the refined knot vectors of the refined control polygon. The function returns node indexes and weights of the new nodes and element indexes and knot

spans of the elements in the refined control polygon.

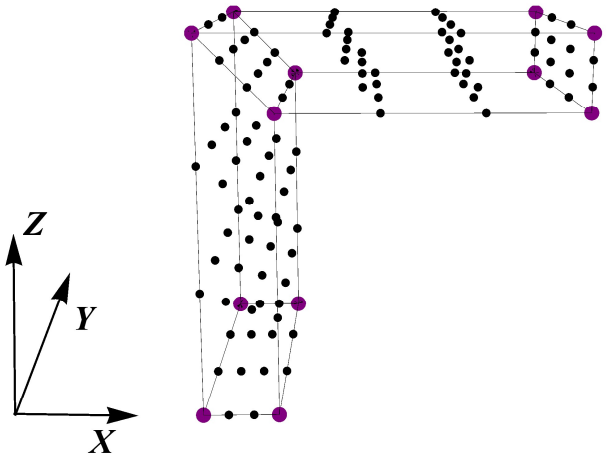
Function also calculates additional nodes for each element (4 for 1D, 16 for 2D, 64 for 3D) that represent points on the exact geometry of the problem. This additional nodes have zero degrees of freedom and are connected with lines when the element mesh is drawn. This means that linear approximation of the geometry is used for graphical representation, however in isogeometric analysis the exact geometry is used.

Nodes of the control poligon for two element mesh and basis function order
knots in ξ direction = {0, 0, 0.5, 1, 1},
knots in η direction = {0, 0, 1, 1},
knots in ζ direction = {0, 0, 1, 1}

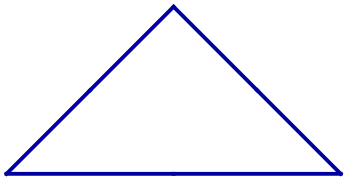
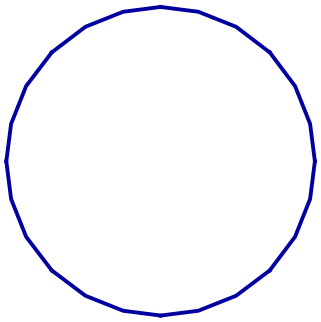
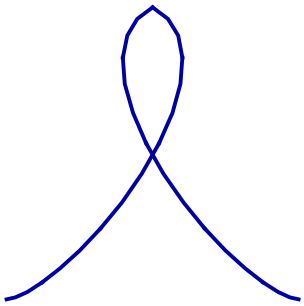


Mesh generated with

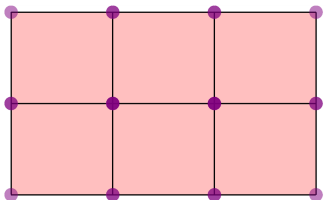
```
In[1169]:= SMTIsogeometricMesh["A", 1, {{0, 0, 0.5, 1, 1}, {0, 0, 1, 1}, {0, 0, 1, 1}},
            {{0, 0, 0.5, 1, 1}, {0, 0, 1, 1}, {0, 0, 1, 1}},
            {{0.2, 0, 0}, {0.2, 0, 0.8}, {1, 0, 0.8}, {0.2, 1, 0}, {0.2, 1, 0.8}, {1, 1, 0.8},
             {0, 0, 0}, {0, 0, 1}, {1, 0, 1}, {0, 1, 0}, {0, 1, 1}, {1, 1, 1}}, ConstantArray[1, 12]]
```



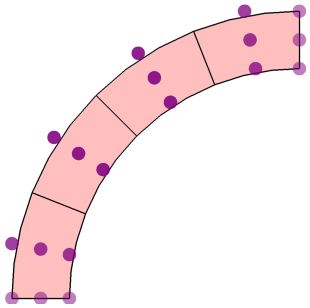
1D isogeometric mesh topology

	SMTIsogeometricMesh["A", 1, {{0, 0, 1, 2, 3, 3}}, {{0, 0, 0.5, 1, 1.5, 2, 2.5, 3, 3}}, {{0, 0}, {1, 1}, {2, 0}, {0, 0}}, {1, 1, 1, 1}]
	SMTIsogeometricMesh["A", 2, {{0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 4}}, {{0, 0, 0, .5, 1, 1, 1.5, 2, 2, 2.5, 3, 3, 3.5, 4, 4, 4}}, {{0, 1}, {1, 1}, {1, 0}, {1, -1}, {0, -1}, {-1, -1}, {-1, 0}, {-1, 1}, {0, 1}}, {1, 1 / Sqrt[2], 1, 1 / Sqrt[2], 1, 1 / Sqrt[2], 1, 1 / Sqrt[2], 1} // N]
	SMTIsogeometricMesh["A", 3, {{0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 2}}, {{0, 0, 0, 0, 0.2, 0.4, 0.6, 0.8, 1, 1, 1, 1.2, 1.4, 1.6, 1.8, 2, 2, 2, 2}}, {{0, 0}, {0.3, 0}, {2, 1.5}, {1, 2}, {0, 1.5}, {1.7, 0}, {2, 0}}, {1, 0.8, 0.6, 1, 0.6, 0.8, 1}];

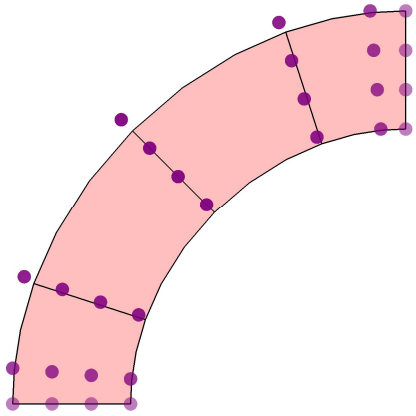
2D isogeometric mesh topology



```
SMTIsogeometricMesh["A", 1,
 {{0, 0, 1, 1}, {0, 0, 1, 1}},
 {{0, 0, 1/3, 2/3, 1, 1},
  {0, 0, 0.5, 1, 1}},
 {{0, 0}, {50, 0}, {0, 30},
  {50, 30}}, {1, 1, 1, 1}]
```

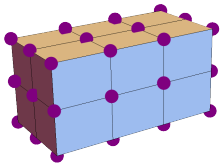


```
SMTIsogeometricMesh[
 "A", 2, {{0, 0, 0, 1, 1, 1},
  {0, 0, 0, 1, 1, 1}},
 {{0, 0, 0, .25, .5, .75, 1, 1, 1},
  {0, 0, 0, 1, 1, 1}},
 {{0.2, 0}, {0.2, 0.8}, {1, 0.8},
  {0.1, 0}, {0.1, 0.9}, {1, 0.9},
  {0, 0}, {0, 1}, {1, 1}},
 {1, 1/Sqrt[2], 1, 1, 1/Sqrt[2],
  1, 1, 1/Sqrt[2], 1} // N]
```

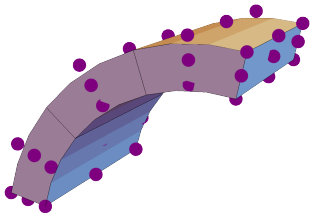


```
SMTIsogeometricMesh[
 "A", 3, {{0, 0, 0, 1, 1, 1},
  {0, 0, 0, 1, 1, 1}},
 {{0, 0, 0, .25, .5, .75, 1, 1, 1},
  {0, 0, 0, 1, 1, 1}},
 {{.3, 0}, {0.3, 0.7×2/3},
  {0.3 + 1/3×0.7, 0.7}, {1, 0.7},
  {.2, 0}, {0.2, 0.8×2/3},
  {0.2 + .8/3, 0.8}, {1, 0.8},
  {.1, 0}, {0.1, 0.9×2/3},
  {0.1 + .9/3, 0.9},
  {1, 0.9}, {0, 0}, {0, 2/3},
  {1/3, 1}, {1, 1}},
 {1, Sqrt[2]/3, Sqrt[2]/3, 1,
  1, Sqrt[2]/3, Sqrt[2]/3, 1,
  1, Sqrt[2]/3, Sqrt[2]/3, 1,
  1, Sqrt[2]/3, Sqrt[2]/3, 1}];
```

3D isogeometric mesh topology

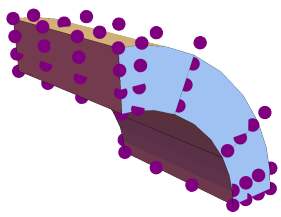


```
SMTIsogeometricMesh["A", 1,
 {{0, 0, 1, 1}, {0, 0, 1, 1}, {0, 0, 1, 1}},
 {{0, 0, 1/3, 2/3, 1, 1},
  {0, 0, 1/2, 1, 1}, {0, 0, 1/2, 1, 1}},
 {{0, 0, 0}, {10, 0, 0}, {0, 5, 0}, {10, 5, 0},
  {0, 0, 5}, {10, 0, 5}, {0, 5, 5}, {10, 5, 5}},
 {1, 1, 1, 1, 1, 1, 1, 1}]
```



```
SMTIsogeometricMesh["A", 2,
 {{0, 0, 0, 1, 1, 1}, {0, 0, 0, 1, 1, 1},
  {0, 0, 0, 1, 1, 1}}, {{0, 0, 0, 1/3, 2/3, 1, 1, 1},
  {0, 0, 0, 1, 1, 1}, {0, 0, 0, 1, 1, 1}},
 {{.2, 0, 0}, {0.2, 0, 0.8}, {1, 0, 0.8}, {0.2, .5, 0},
  {0.2, .5, 0.8}, {1, .5, 0.8}, {0.2, 1, 0},
  {0.2, 1, 0.8}, {1, 1, 0.8}, {0.1, 0, 0}, {0.1, 0, 0.9},
  {1, 0, 0.9}, {.1, 0.5, 0}, {0.1, 0.5, 0.9},
  {1, .5, 0.9}, {.1, 1, 0}, {0.1, 1, 0.9}, {1, 1, 0.9},
  {0, 0, 0}, {0, 0, 1}, {1, 0, 1}, {0, .5, 0}, {0, .5, 1},
  {1, .5, 1}, {0, 1, 0}, {0, 1, 1}, {1, 1, 1}},
 {1, 1/Sqrt[2], 1, 1, 1/Sqrt[2], 1, 1,
```

```
    1/Sqrt[2], 1, 1, 1/Sqrt[2], 1, 1, 1/Sqrt[2],
    1, 1, 1/Sqrt[2], 1, 1, 1/Sqrt[2], 1,
    1, 1/Sqrt[2], 1, 1, 1/Sqrt[2], 1} // N]
```



```
SMTIsogeometricMesh["A", 3,
    {{0, 0, 0, 0, 1, 1, 1, 1}, {0, 0, 0, 0, 1, 1, 1, 1}},
    {0, 0, 0, 0, 1, 1, 1, 1}},
 {{0, 0, 0, 0, 1/3, 2/3, 1, 1, 1, 1},
  {0, 0, 0, 0, 1, 1, 1, 1}, {0, 0, 0, 0, 1, 1, 1, 1}},
 {{.3, 0, 0}, {0.3, 0, 0.7×2/3},
  {0.3 + 1/3×0.7, 0, 0.7}, {1, 0, 0.7}, {.3, 1/3, 0},
  {0.3, 1/3, 0.7×2/3}, {0.3 + .7/3, 1/3, 0.7},
  {1, 1/3, 0.7}, {.3, 2/3, 0}, {0.3, 2/3, 0.7×2/3},
  {0.3 + .7/3, 2/3, 0.7}, {1, 2/3, 0.7}, {.3, 1, 0},
  {0.3, 1, 0.7×2/3}, {0.3 + .7/3, 1, 0.7}, {1, 1, 0.7},
  {.2, 0, 0}, {0.2, 0, 0.8×2/3}, {0.2 + .8/3, 0, 0.8},
  {1, 0, 0.8}, {.2, 1/3, 0}, {0.2, 1/3, 0.8×2/3},
  {0.2 + .8/3, 1/3, 0.8}, {1, 1/3, 0.8}, {.2, 2/3, 0},
  {0.2, 2/3, 0.8×2/3}, {0.2 + .8/3, 2/3, 0.8},
  {1, 2/3, 0.8}, {.2, 1, 0}, {0.2, 1, 0.8×2/3},
  {0.2 + .8/3, 1, 0.8}, {1, 1, 0.8}, {.1, 0, 0},
  {0.1, 0, 0.9×2/3}, {0.1 + .9/3, 0, 0.9},
  {1, 0, 0.9}, {.1, 1/3, 0}, {0.1, 1/3, 0.9×2/3},
  {0.1 + .9/3, 1/3, 0.9}, {1, 1/3, 0.9}, {.1, 2/3, 0},
  {0.1, 2/3, 0.9×2/3}, {0.1 + .9/3, 2/3, 0.9},
  {1, 2/3, 0.9}, {.1, 1, 0}, {0.1, 1, 0.9×2/3},
  {0.1 + .9/3, 1, 0.9}, {1, 1, 0.9},
  {0, 0, 0}, {0, 0, 2/3}, {1/3, 0, 1},
  {1, 0, 1}, {0, 1/3, 0}, {0, 1/3, 2/3},
  {1/3, 1/3, 1}, {1, 1/3, 1}, {0, 2/3, 0},
  {0, 2/3, 2/3}, {1/3, 2/3, 1}, {1, 2/3, 1},
  {0, 1, 0}, {0, 1, 2/3}, {1/3, 1, 1}, {1, 1, 1}},
 {1, 2/(3 Sqrt[2]), 2/(3 Sqrt[2]), 1, 1,
   2/(3 Sqrt[2]), 2/(3 Sqrt[2]), 1, 1, 2/(3 Sqrt[2]),
   2/(3 Sqrt[2]), 1, 1, 2/(3 Sqrt[2]), 2/(3 Sqrt[2]),
   1, 1, 2/(3 Sqrt[2]), 2/(3 Sqrt[2]), 1,
   1, 2/(3 Sqrt[2]), 2/(3 Sqrt[2]), 1, 1,
   2/(3 Sqrt[2]), 2/(3 Sqrt[2]), 1, 1, 2/(3 Sqrt[2]),
   2/(3 Sqrt[2]), 1, 1, 2/(3 Sqrt[2]), 2/(3 Sqrt[2]),
   1, 1, 2/(3 Sqrt[2]), 2/(3 Sqrt[2]), 1,
   1, 2/(3 Sqrt[2]), 2/(3 Sqrt[2]), 1, 1,
   2/(3 Sqrt[2]), 2/(3 Sqrt[2]), 1, 1, 2/(3 Sqrt[2]),
   2/(3 Sqrt[2]), 1, 1, 2/(3 Sqrt[2]), 2/(3 Sqrt[2]),
   1, 1, 2/(3 Sqrt[2]), 2/(3 Sqrt[2]), 1,
   1, 2/(3 Sqrt[2]), 2/(3 Sqrt[2]), 1} // N]
```

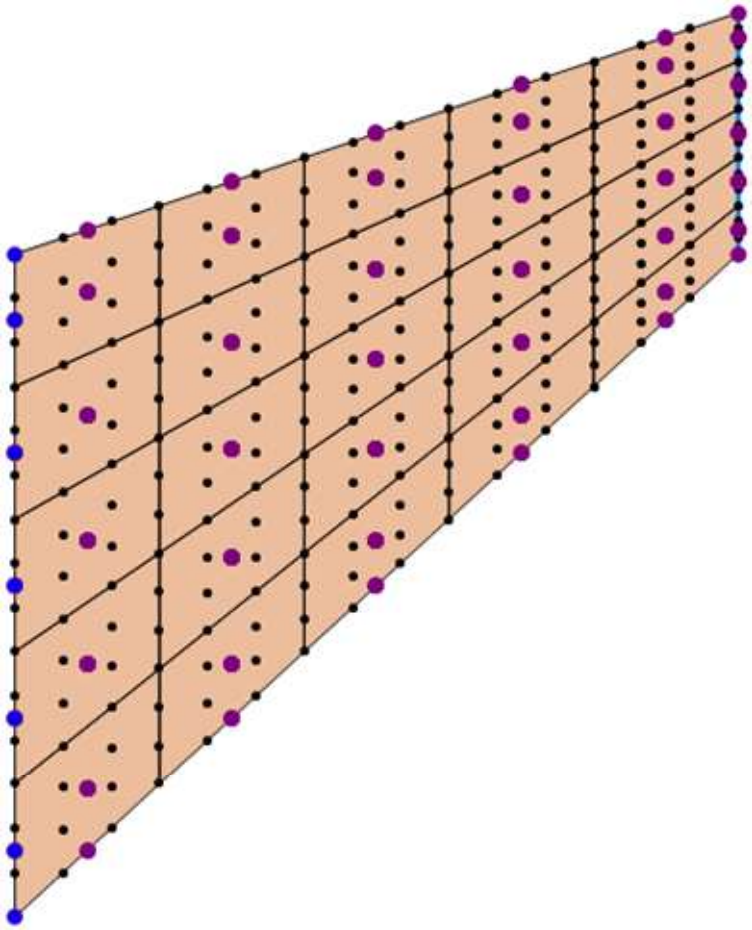
Isogeometric examples

Cooke's Membrane (Isogeometric)

In[1170]:=
```
<< AceFEM`;
```

In[1171]:=
```
SMTInputData[];
p = 2; (*base function order input is valid for 1/2/3*)
nr = 5; (*number of elements in X direction*)
nv = 5; (*number of elements in Y direction*)

SMTAddDomain[{"A", Switch[p, 1, "IGA:SEPEB1HY", 2, "IGA:SEPEB2HY", 3, "IGA:SEPEB3HY"],
   {"E *" -> 1, "ν *" -> 0, "t *" → 1}}];

knotsXi = Switch[p, 1, {0, 0, 1, 1}, 2, {0, 0, 0, 1, 1, 1}, 3, {0, 0, 0, 0, 1, 1, 1, 1}];
knotsXiRefined = Flatten[{ConstantArray[0, p], Table[i / nr, {i, 0, nr}], ConstantArray[1, p]}];
knotsEta = Switch[p, 1, {0, 0, 1, 1}, 2, {0, 0, 0, 1, 1, 1}, 3, {0, 0, 0, 0, 1, 1, 1, 1}];
knotsEtaRefined = Flatten[{ConstantArray[0, p], Table[i / nv, {i, 0, nv}], ConstantArray[1, p]}];
contrlpnts = Switch[p
   , 1, {{0, 0}, {48, 44}, {0, 44}, {48, 60}}
   , 2, {{0, 0}, {24, 22}, {48, 44}, {0, 22}, {24, 37}, {48, 52}, {0, 44}, {24, 52}, {48, 44 + 16}}
   , 3, {{0, 0}, {48 / 3, 44 / 3}, {2×48 / 3, 2×44 / 3}, {48, 44}, {0, 44 / 3},
    {48 / 3, 5×44 / 9 + 16 / 9}, {2×48 / 3, 7×44 / 9 + 32 / 9}, {48, 44 + 16 / 3}, {0, 2×44 / 3},
    {48 / 3, 7×44 / 9 + 32 / 9}, {2×48 / 3, 8×44 / 9 + 2×32 / 9}, {48, 44 + 32 / 3},
    {0, 44}, {48 / 3, 44 + 16 / 3}, {2×48 / 3, 44 + 32 / 3}, {48, 44 + 16}}];
weights = ConstantArray[1, (p + 1) ^ 2];
SMTIsogeometricMesh["A", p, {knotsXi, knotsEta},
  {knotsXiRefined, knotsEtaRefined}, contrlpnts, weights];

SMTAddDomain["B", Switch[p, 1, "IGA:SELoL2B1", 2, "IGA:SELoL2B2", 3, "IGA:SELoL2B3"],
  {"qx *" → 0, "qy *" → -0.1}];
contrlpnts2 = Switch[p
   , 1, {{48, 44}, {48, 60}}
   , 2, {{48, 44}, {48, 52}, {48, 60}}
   , 3, {{48, 44}, {48, 44 + 16 / 3}, {48, 44 + 32 / 3}, {48, 60}}];
weights2 = ConstantArray[1, (p + 1)];
SMTIsogeometricMesh["B", p, {knotsEta}, {knotsEtaRefined}, contrlpnts2, weights2];

SMTAddEssentialBoundary[Line[{{0, 0}, {0, 44}}, "D"], 1 -> 0, 2 -> 0];
SMTAnalysis[];
```

In[1189]:=
```
SMTShowMesh["BoundaryConditions" -> True, "NodeMarks" → True]
```

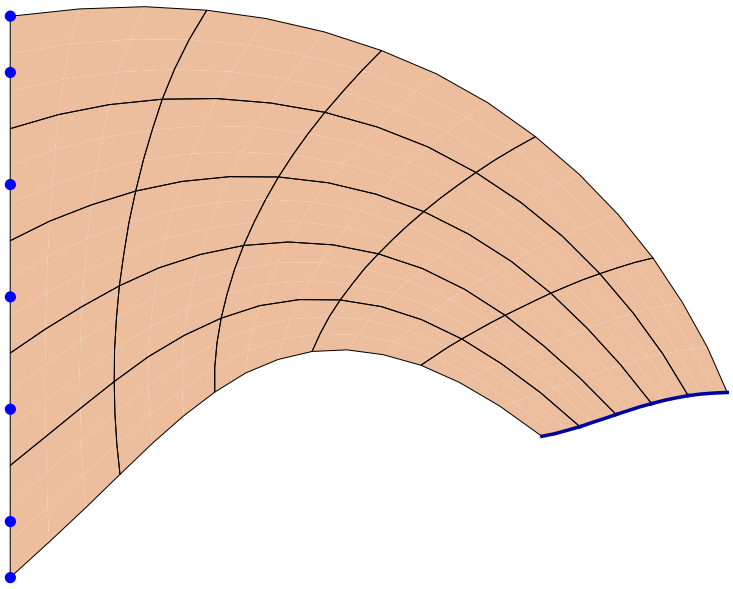


In[1190]:=
```
SMTNextStep["λ" → 0.1];
While[
  While[step = SMTConvergence[10^-8, 10, {"Adaptive BC", 8, 0.001, 0.2, 1}],
   SMTNewtonIteration[];];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" → step[[2]]]
 ];
```

In[1192]:=
```
SMTNodeData[Point[{48, 60}, "D"], "at"]
```

Out[1192]=
```
{{8.20404, -45.4867}}
```

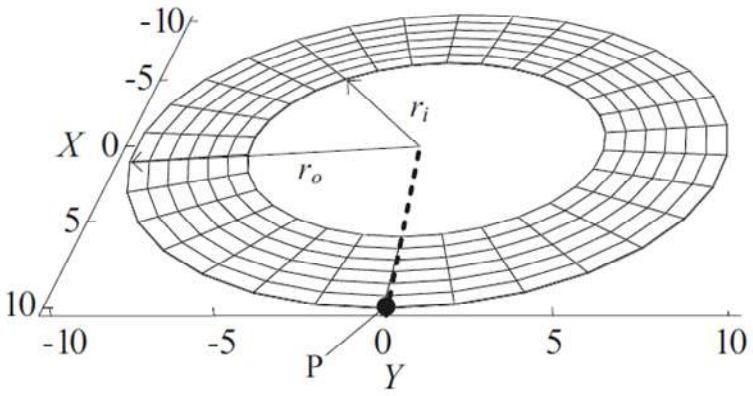
In[1193]:=
```
SMTShowMesh["BoundaryConditions" -> True, "DeformedMesh" -> True]
```

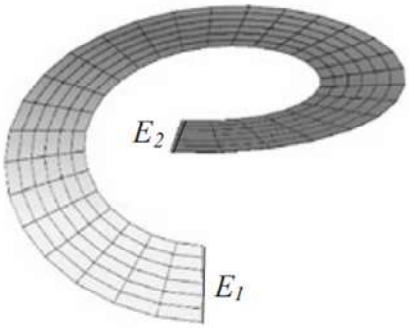


Thin Plate Ring (Isogeometric)

A thin circular plate ring is loaded by an equally distributed surface load at the free end and, in the opposite direction, at the clamped end to avoid stress singularities.

Geometry	Material	Load	Constraints
$r_i = 6$ mm	$K = 7000000$ MPa	$q = 6.67$ MPa	$Y=0 \wedge X>0 \wedge Z=0:\quad v=w=0$
$r_o = 10$ mm	$\mu = 10500000$ MPa		$Y=0 \wedge X>0 \wedge Z>0:\quad v=0$
$t = 0.03$ mm	$\beta = -2$		$(r_i,0,0):\quad u=v=w=0$

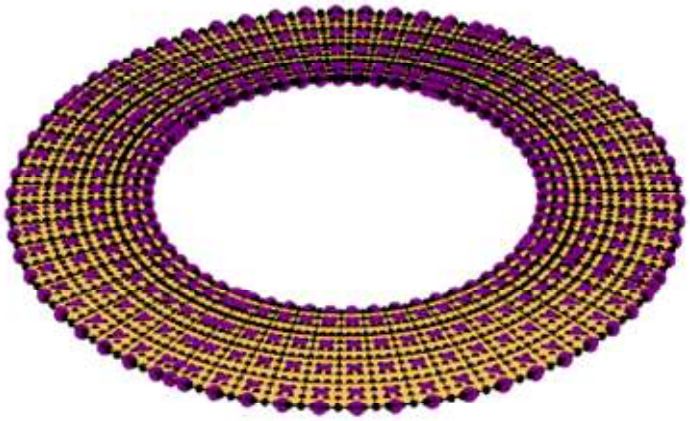




In[1194]:= `<< AceFEM`;`

In[1195]:=
```
SMTInputData[];
q = 6.67 ×4×0.03; (*surface load on the free end*)
p = 2; (*base function order*)
ns = 16; (*number of elements in tangential direction*)
nr = 4; (*number of elements in radial direction*)
nv = 1; (*number of elements in Z direction*)
K = 7 000 000;
G = 10 500 000;
SMTAddDomain[
  {"A", "IGA:SED3B2HY", {"E *" -> 9 K G / (3 K + G), "ν *" -> (3 K - 2 G) / (2 (3 K + G))}}];
knotsXi = {0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 4};
knotsXiRefined =
  Flatten[{ConstantArray[0, 2], Table[i / ns, {i, 0, ns}], 1 + Table[i / ns, {i, 0, ns}],
    2 + Table[i / ns, {i, 0, ns}], 3 + Table[i / ns, {i, 0, ns}], ConstantArray[4, 2]}];
knotsEta = {0, 0, 0, 1, 1, 1};
knotsEtaRefined =
  Flatten[{ConstantArray[0, p], Table[i / nr, {i, 0, nr}], ConstantArray[1, p]}];
knotsZeta = {0, 0, 0, 1, 1, 1};

contrlpnts = {{0, 10, 0}, {10, 10, 0}, {10, 0, 0}, {10, -10, 0}, {0, -10, 0}, {-10, -10, 0},
   {-10, 0, 0}, {-10, 10, 0}, {0, 10, 0}, {0, 8, 0}, {8, 8, 0}, {8, 0, 0}, {8, -8, 0}, {0, -8, 0},
   {-8, -8, 0}, {-8, 0, 0}, {-8, 8, 0}, {0, 8, 0}, {0, 6, 0}, {6, 6, 0}, {6, 0, 0}, {6, -6, 0},
   {0, -6, 0}, {-6, -6, 0}, {-6, 0, 0}, {-6, 6, 0}, {0, 6, 0}, {0, 10, -0.015}, {10, 10, -0.015},
   {10, 0, -0.015}, {10, -10, -0.015}, {0, -10, -0.015}, {-10, -10, -0.015}, {-10, 0, -0.015},
   {-10, 10, -0.015}, {0, 10, -0.015}, {0, 8, -0.015}, {8, 8, -0.015}, {8, 0, -0.015},
   {8, -8, -0.015}, {0, -8, -0.015}, {-8, -8, -0.015}, {-8, 0, -0.015}, {-8, 8, -0.015},
   {0, 8, -0.015}, {0, 6, -0.015}, {6, 6, -0.015}, {6, 0, -0.015}, {6, -6, -0.015},
   {0, -6, -0.015}, {-6, -6, -0.015}, {-6, 0, -0.015}, {-6, 6, -0.015}, {0, 6, -0.015},
   {0, 10, -0.03}, {10, 10, -0.03}, {10, 0, -0.03}, {10, -10, -0.03}, {0, -10, -0.03},
   {-10, -10, -0.03}, {-10, 0, -0.03}, {-10, 10, -0.03}, {0, 10, -0.03}, {0, 8, -0.03},
   {8, 8, -0.03}, {8, 0, -0.03}, {8, -8, -0.03}, {0, -8, -0.03}, {-8, -8, -0.03}, {-8, 0, -0.03},
   {-8, 8, -0.03}, {0, 8, -0.03}, {0, 6, -0.03}, {6, 6, -0.03}, {6, 0, -0.03}, {6, -6, -0.03},
   {0, -6, -0.03}, {-6, -6, -0.03}, {-6, 0, -0.03}, {-6, 6, -0.03}, {0, 6, -0.03}};
weights = {1, 1 / Sqrt[2], 1 , 1 / Sqrt[2], 1, 1 / Sqrt[2], 1, 1 / Sqrt[2], 1, 1, 1 / Sqrt[2],
    1 , 1 / Sqrt[2], 1, 1 / Sqrt[2], 1, 1 / Sqrt[2], 1, 1, 1 / Sqrt[2], 1 , 1 / Sqrt[2], 1,
    1 / Sqrt[2], 1, 1 / Sqrt[2], 1, 1, 1 / Sqrt[2], 1 , 1 / Sqrt[2], 1, 1 / Sqrt[2], 1, 1 / Sqrt[2],
    1, 1, 1 / Sqrt[2], 1 , 1 / Sqrt[2], 1, 1 / Sqrt[2], 1, 1 / Sqrt[2], 1, 1, 1 / Sqrt[2],
    1 , 1 / Sqrt[2], 1, 1 / Sqrt[2], 1, 1 / Sqrt[2], 1, 1, 1 / Sqrt[2], 1 , 1 / Sqrt[2],
    1, 1 / Sqrt[2], 1, 1 / Sqrt[2], 1, 1, 1 / Sqrt[2], 1 , 1 / Sqrt[2], 1, 1 / Sqrt[2], 1,
    1 / Sqrt[2], 1, 1, 1 / Sqrt[2], 1 , 1 / Sqrt[2], 1, 1 / Sqrt[2], 1, 1 / Sqrt[2], 1} // N;
SMTIsogeometricMesh["A", p, {knotsXi, knotsEta, knotsZeta},
  {knotsXiRefined, knotsEtaRefined, knotsZeta}, contrlpnts, weights];
SMTAnalysis["Tie" → False];
```

In[1213]:=
```
SMTShowMesh["NodeMarks" → True]
```



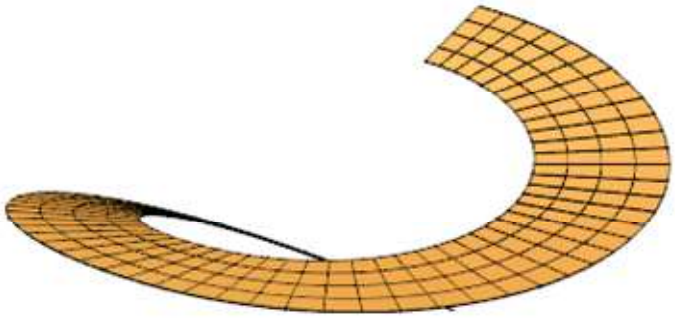
In[1214]:=
```
bz0 = SMTFindNodes["X" == 0 && "Y" ≥ 0 && "Z" == 0 && "ID" == "D" &];
bz1 = SMTFindNodes["X" == 0 && "Y" ≥ 0 && "Z" < 0 && "ID" == "D" &];
bz2 = SMTFindNodes["X" == 0 && "Y" ≥ 0 && "ID" == "D" &];
SMTNodeData[Table[bz0[[i]], {i, 1, 2 (nr + p), 2}], "DOF", Table[{-1, 1, -1}, {nr + p}]];

 (*adding essential boundary conditions to the nodes in radial direction*)
SMTNodeData[SMTFindNodes["Y" == 6 && "X" == 0 && "Z" == 0 && "ID" == "D" &][[1]], "DOF", {-1, -1, -1}];
SMTNodeData[Table[bz1[[i]], {i, 1, 2 (nr + p) (nv + p - 1), 2}],
 "DOF", Table[{-1, 1, 1}, {(nr + p) (nv + p - 1)}]];
SMTNodeData[Table[bz2[[i]], {i, 2, 2 (nr + p) (nv + p), 2}],
  "dB", Table[{0, 0, -q / ((nr + p) (nv + p))}, {(nr + p) (nv + p)}]];
SMTNodeData[Table[bz2[[i]], {i, 1, 2 (nr + p) (nv + p), 2}], "dB",
  Table[{0, 0, q / ((nr + p) (nv + p))}, {(nr + p) (nv + p)}]];
```

In[1221]:=
```
SMTSetSolver[];
```

In[1222]:=
```
SMTNextStep["λ" → 0.1];
While[
  While[step = SMTConvergence[10^-8, 10, {"Adaptive BC", 8, 0.001, 0.2, 1.}],
   SMTNewtonIteration[];];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" → step[[2]]]
 ];
SMTNodeData["X" == 0 && "Y" == 6 && "Z" == -0.03 && "ID" == "D" &, "at"]
```

$$\left\{\left\{0.,\ -1.17757\times 10^{-7},\ 0.0444666\right\},\ \left\{0.833168,\ -2.57672,\ -7.31602\right\}\right\}$$

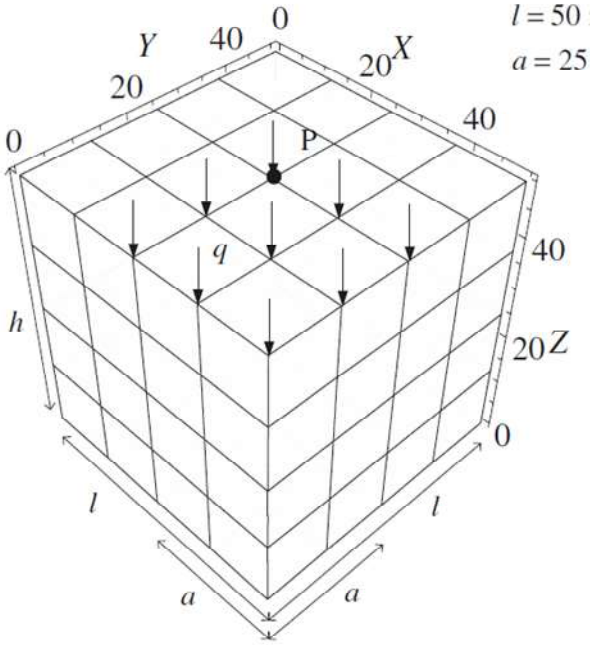
In[1225]:=
```
SMTShowMesh["DeformedMesh" -> True]
```

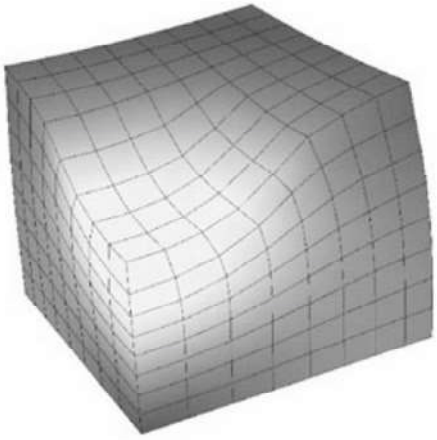


Incompressible Block (Isogeometric)

A nearly incompressible block is loaded by an equally distributed surface load in its top center. Only a quarter of the block is discretized due to symmetry.

Load	Geometry	Material	Constraints
$q = 3$ MPa	$h = 50$ mm	$K = 501$ MPa	$Z = 0$: $w = 0$
	$l = 50$ mm	$\mu = 1.61148$ MPa	$Z = h$: $u = v = 0$
	$a = 25$ mm	$\beta = -2$	$X = l$: $u = 0$
			$Y = l$: $v = 0$



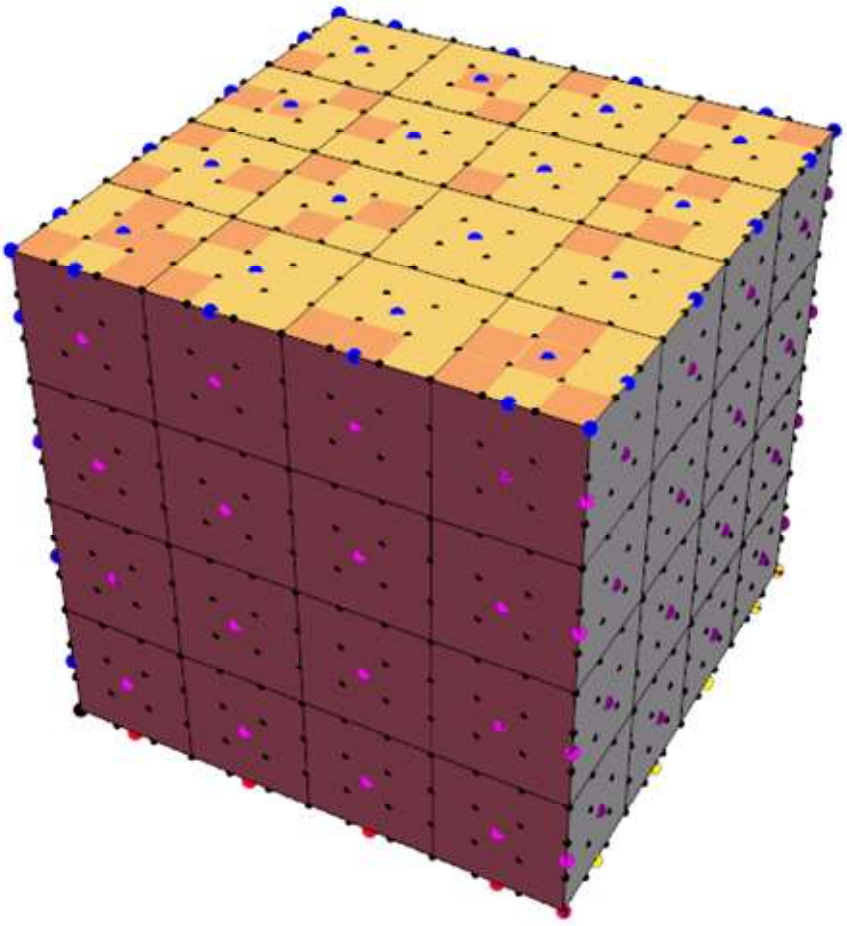


In[1226]:=
```
<< AceFEM`;
```

In[1227]:=
```
SMTInputData[];
q = -3; (*surface load in the top center top of the block*)
l = 50; (*length and width of the block*)
h = 50; (*height of the block*)
p = 2; (*base function order 1/2/3*)
nr = 4; (*number of elements in X direction*)
ns = 4; (*number of elements in Y direction*)
nv = 4; (*number of elements in Z direction*)
K = 501;
G = 1.61148;

SMTAddDomain[{"A", Switch[p, 1, "IGA:SED3B1HY", 2, "IGA:SED3B2HY", 3, "IGA:SED3B3HY"],
   {"E *" -> 9 K G / (3 K + G), "ν *" -> (3 K - 2 G) / (2 (3 K + G))}}];
```

```
knotsXi = Switch[p, 1, {0, 0, 1, 1}, 2, {0, 0, 0, 1, 1, 1}, 3, {0, 0, 0, 0, 1, 1, 1, 1}];
knotsXiRefined = Flatten[{ConstantArray[0, p], Table[i / nr, {i, 0, nr}], ConstantArray[1, p]}];
knotsEta = Switch[p, 1, {0, 0, 1, 1}, 2, {0, 0, 0, 1, 1, 1}, 3, {0, 0, 0, 0, 1, 1, 1, 1}];
knotsEtaRefined = Flatten[{ConstantArray[0, p], Table[i / ns, {i, 0, ns}], ConstantArray[1, p]}];
knotsZeta = Switch[p, 1, {0, 0, 1, 1}, 2, {0, 0, 0, 1, 1, 1}, 3, {0, 0, 0, 0, 1, 1, 1, 1}];
knotsZetaRefined =
  Flatten[{ConstantArray[0, p], Table[i / nv, {i, 0, nv}], ConstantArray[1, p]}];
contrlpnts = Switch[p
   , 1, {{0, 0, 0}, {l, 0, 0}, {0, l, 0}, {l, l, 0}, {0, 0, h}, {l, 0, h}, {0, l, h}, {l, l, h}}
   , 2, {{0, 0, 0}, {0.5 l, 0, 0}, {l, 0, 0}, {0, l / 2, 0},
    {0.5 l, l / 2, 0}, {l, l / 2, 0}, {0, l, 0}, {0.5 l, l, 0}, {l, l, 0},
    {0, 0, 0.5 l}, {0.5 l, 0, 0.5 l}, {l, 0, 0.5 l}, {0, l / 2, 0.5 l}, {0.5 l, l / 2, 0.5 l},
    {l, l / 2, 0.5 l}, {0, l, 0.5 l}, {0.5 l, l, 0.5 l}, {l, l, 0.5 l}, {0, 0, l}, {0.5 l, 0, l},
    {l, 0, l}, {0, l / 2, l}, {0.5 l, l / 2, l}, {l, l / 2, l}, {0, l, l}, {0.5 l, l, l}, {l, l, l}}
   , 3, {{0, 0, 0}, {1 / 3 l, 0, 0}, {2 / 3 l, 0, 0}, {l, 0, 0}, {0, l / 3 , 0},
    {1 / 3 l, l / 3 , 0}, {2 / 3 l, l / 3 , 0}, {l, l / 3 , 0}, {0, 2 l / 3, 0}, {1 / 3 l, 2 l / 3, 0},
    {2 / 3 l, 2 l / 3, 0}, {l, 2 l / 3, 0}, {0, l, 0}, {l / 3, l, 0}, {2 / 3 l, l, 0}, {l, l, 0},
    {0, 0, 1 / 3 h}, {1 / 3 l, 0, 1 / 3 h}, {2 / 3 l, 0, 1 / 3 h}, {l, 0, 1 / 3 h},
    {0, l / 3, h / 3}, {l / 3, l / 3, h / 3}, {2 l / 3, l / 3, h / 3}, {l, l / 3, h / 3},
    {0, 2 l / 3, h / 3}, {l / 3, 2 l / 3, h / 3}, {2 l / 3, 2 l / 3, h / 3}, {l, 2 l / 3, h / 3},
    {0, l, h / 3}, {l / 3, l, h / 3}, {2 l / 3, l, h / 3}, {l, l, h / 3},
    {0, 0, 2 h / 3}, {l / 3, 0, 2 h / 3}, {2 l / 3, 0, 2 h / 3}, {l, 0, 2 h / 3}, {0, l / 3, 2 h / 3},
    {l / 3, l / 3, 2 h / 3}, {2 l / 3, l / 3, 2 h / 3}, {l, l / 3, 2 h / 3}, {0, 2 l / 3, 2 h / 3},
    {l / 3, 2 l / 3, 2 h / 3}, {2 l / 3, 2 l / 3, 2 h / 3}, {l, 2 l / 3, 2 h / 3},
    {0, l, 2 h / 3}, {l / 3, l, 2 h / 3}, {2 l / 3, l, 2 h / 3}, {l, l, 2 h / 3},
    {0, 0, h}, {l / 3, 0, h}, {2 l / 3, 0, h}, {l, 0, h}, {0, l / 3, h}, {l / 3, l / 3, h},
    {2 l / 3, l / 3, h}, {l, l / 3, h}, {0, 2 l / 3, h}, {l / 3, 2 l / 3, h}, {2 l / 3, 2 l / 3, h},
    {l, 2 l / 3, h}, {0, l, h}, {l / 3, l, h}, {2 l / 3, l, h}, {l, l, h}}];
weights = ConstantArray[1, (p + 1) ^ 3];
SMTIsogeometricMesh["A", p, {knotsXi, knotsEta, knotsZeta},
  {knotsXiRefined, knotsEtaRefined, knotsZetaRefined}, contrlpnts, weights];

contrlpnts2 = Switch[p
   , 1, {{0, 0, h}, {l, 0, h}, {0, l, h}, {l, l, h}}
   , 2, {{0, 0, h}, {0.5 l, 0, h}, {l, 0, h}, {0, l / 2, h},
    {0.5 l, l / 2, h}, {l, l / 2, h}, {0, l, h}, {0.5 l, l, h}, {l, l, h}}
   , 3, {{0, 0, h}, {1 / 3 l, 0, h}, {2 / 3 l, 0, h}, {l, 0, h}, {0, l / 3, h}, { l / 3, l / 3, h},
    {2 l / 3, l / 3, h}, {l, l / 3, h}, {0, 2 l / 3, h}, { l / 3, 2 l / 3, h}, {2 l / 3, 2 l / 3, h},
    {l, 2 l / 3, h}, {0, l, h}, {l / 3, l, h}, {2 l / 3, l, h}, {l, l, h}}];
weights2 = ConstantArray[1, (p + 1) ^ 2];
SMTAddDomain["B", Switch[p, 1, "IGA:SELoLsB1", 2, "IGA:SELoLsB2", 3, "IGA:SELoLsB3"],
  {"qx *" → 0, "qy *" → 0, "qz *" → -3}];
SMTIsogeometricMesh["B", p, {knotsXi, knotsEta},
  {knotsXiRefined, knotsEtaRefined}, contrlpnts2, weights2];

SMTAddEssentialBoundary[
  {"Z" == 0 && "ID" == "D" &, 3 → 0}, { "Z" == h && "ID" == "D" &, 1 -> 0, 2 -> 0}];
SMTAddEssentialBoundary["Y" == 0 && "ID" == "D" &, 2 -> 0];
SMTAddEssentialBoundary["X" == 0 && "ID" == "D" &, 1 -> 0];

SMTAnalysis[];
```

In[1255]:=
```
SMTShowMesh["BoundaryConditions" -> True, "NodeMarks" → True]
```

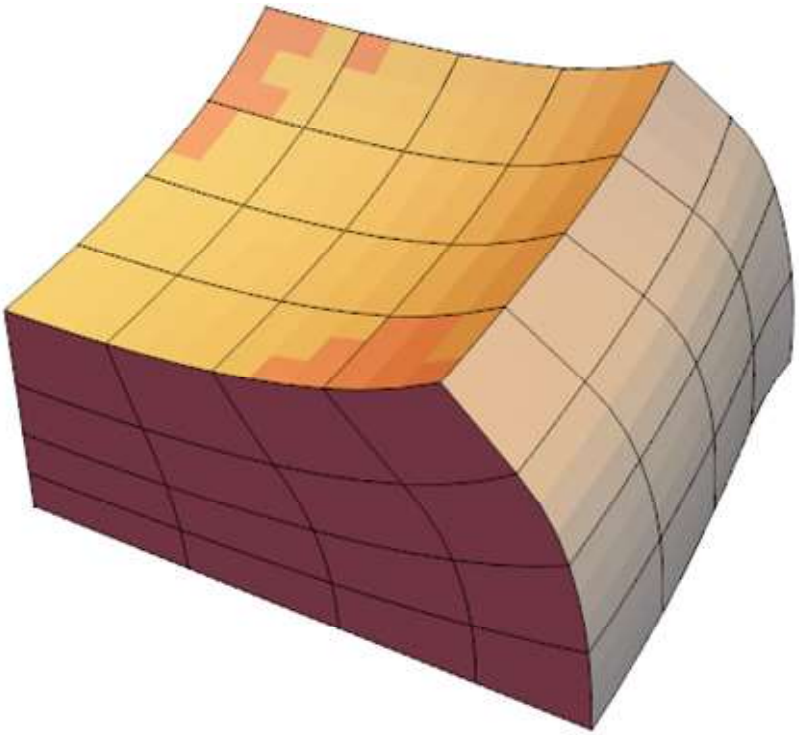


In[1256]:=
```
SMTNextStep["λ" → .1];
While[
  While[
   step = SMTConvergence[10^-8, 20, {"Adaptive BC", 8, .0001, 1, 1}], SMTNewtonIteration[]];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" → step[[2]]]
 ];
```

In[1258]:=
```
SMTNodeData[SMTFindNodes[Point[{0, 0, h}, "D"]], "at"][[1, 3]]
```

Out[1258]=
```
-23.3765
```

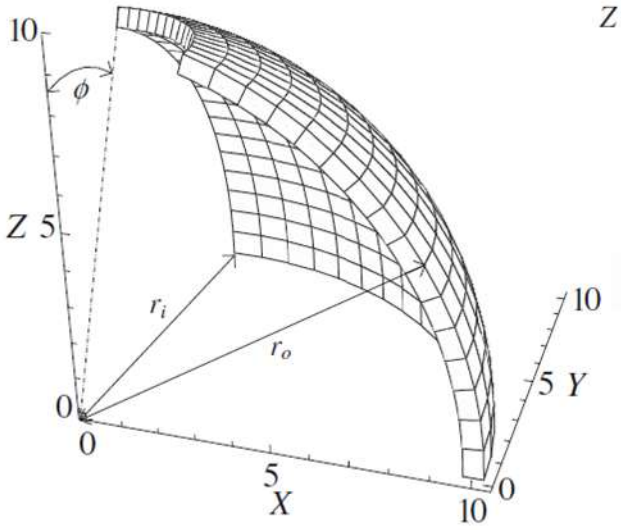
In[1259]:=
```
SMTShowMesh["DeformedMesh" -> True]
```

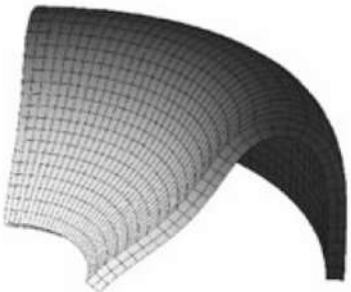


Spherical Shell (Isogeometric)

A spherical shell with a hole at its top and bottom is loaded with an uniform displacement at the edge of the hole. Only an 8 th of the sphere is discretized due to the symmetry.

Geometry	Material	Boundary conditions
$r_i = 9.75$ mm	$K = 175000$ MPa	$X = 0$: $u = 0$
$r_o = 10.25$ mm	$\mu = 80769.23$ MPa	$Y = 0$: $v = 0$
$\phi = 18°$	$\beta = -2$	$Z = 0$: $w = 0$
		$Z = \cos(\phi)\, r_i$: $w = 10$ mm

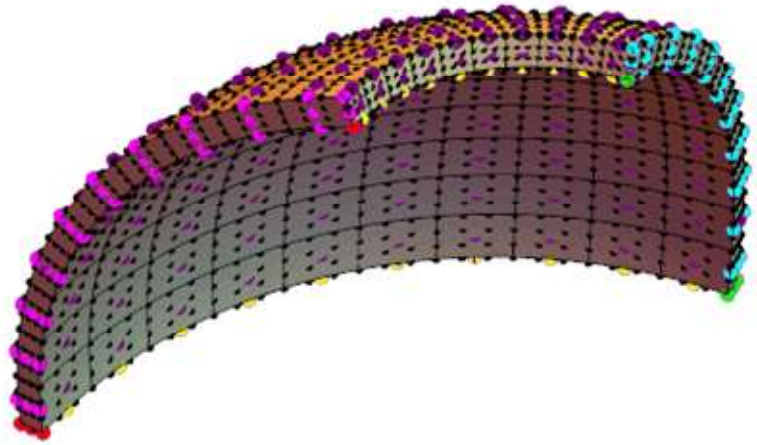




In[1260]:= `<< AceFEM`;`

In[1261]:=
```
SMTInputData[];
r = 10.25 ;(*outer radius of the sphere*)
d = 0.5; (*thickness of the sphere*)
p = 2; (*base function order*)
nr = 10; (*number of elements in X direction*)
ns = 10; (*number of elements in Y direction*)
nv = 1; (*number of elements in Z direction*)
K = 175 000;
G = 80 769.23;

SMTAddDomain[
  {"A", "IGA:SED3B2HY", {"E *" -> 9 K G / (3 K + G), "ν *" -> (3 K - 2 G) / (2 (3 K + G))}}];
knotsXi = {0, 0, 0, 1, 1, 1};
knotsXiRefined =
  Flatten[{ConstantArray[0, p], Table[i / nr, {i, 0, nr}], ConstantArray[1, p]}];
knotsEta = {0, 0, 0, 1, 1, 1};
knotsEtaRefined =
  Flatten[{ConstantArray[0, p], Table[i / ns, {i, 0, ns}], ConstantArray[1, p]}];
knotsZeta = {0, 0, 0, 1, 1, 1};
knotsZetaRefined =
  Flatten[{ConstantArray[0, p], Table[i / nv, {i, 0, nv}], ConstantArray[1, p]}];

contrlpnts =
  {{0.5, 0, 0}, {0.5, 0, 7.083789648052267`}, {7.237084304844262`, 0, 9.272801033877746`},
   {0.5, 9.75, 0}, {0.5, 9.75, 7.083789648052267`},
   {7.237084304844262`, 10.25 - 7.237084304844262`, 9.272801033877746`}, {10.25, 9.75, 0},
   {10.25, 9.75, 7.083789648052267`}, {10.25, 10.25 - 7.237084304844262`, 9.272801033877746`},
   {0.25, 0, 0}, {0.25, 0, 7.265425280053607`}, {7.1598300562505255`, 0, 9.510565162951535`},
   {0.25, 10, 0}, {0.25, 10, 7.265425280053607`},
   {7.1598300562505255`, 10.25 - 7.1598300562505255`, 9.510565162951535`}, {10.25, 10, 0},
   {10.25, 10, 7.265425280053607`}, {10.25, 10.25 - 7.1598300562505255`, 9.510565162951535`},
   {0, 0, 0}, {0, 0, 7.44706091205495`}, {7.082575807656789`, 0, 9.748329292025323`},
   {0, 10.25, 0}, {0, 10.25, 7.44706091205495`},
   {7.082575807656789`, 10.25 - 7.082575807656789`, 9.748329292025323`}, {10.25, 10.25, 0},
   {10.25, 10.25, 7.44706091205495`}, {10.25, 10.25 - 7.082575807656789`, 9.748329292025323`}};
weights = {1, Cos[2 Pi / 10], 1, 1 / Sqrt[2], Cos[2 Pi / 10] 1 / Sqrt[2],
    1 / Sqrt[2], 1, Cos[2 Pi / 10], 1, 1, Cos[2 Pi / 10], 1, 1 / Sqrt[2],
    Cos[2 Pi / 10] 1 / Sqrt[2], 1 / Sqrt[2], 1, Cos[2 Pi / 10], 1, 1, Cos[2 Pi / 10], 1,
    1 / Sqrt[2], Cos[2 Pi / 10] 1 / Sqrt[2], 1 / Sqrt[2], 1, Cos[2 Pi / 10], 1} // N;

SMTIsogeometricMesh["A", p, {knotsXi, knotsEta, knotsZeta},
  {knotsXiRefined, knotsEtaRefined, knotsZetaRefined}, contrlpnts, weights];
SMTAddEssentialBoundary["Z" == 0 && "ID" == "D" &, 3 → 0];
SMTAddEssentialBoundary["Y" == 0 && "ID" == "D" &, 2 -> 0];
SMTAddEssentialBoundary["X" == 10.25 && "ID" == "D" &, 1 -> 0];
SMTAddEssentialBoundary["Z" == 9.75 Sin[72 Pi / 180] && "ID" == "D" &, 3 → -10];
SMTAnalysis[];
```

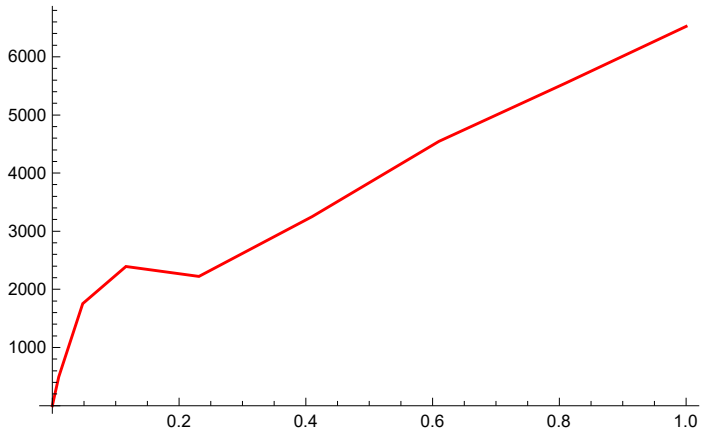
In[1285]:=
```
SMTShowMesh["BoundaryConditions" -> True, "NodeMarks" → True]
```



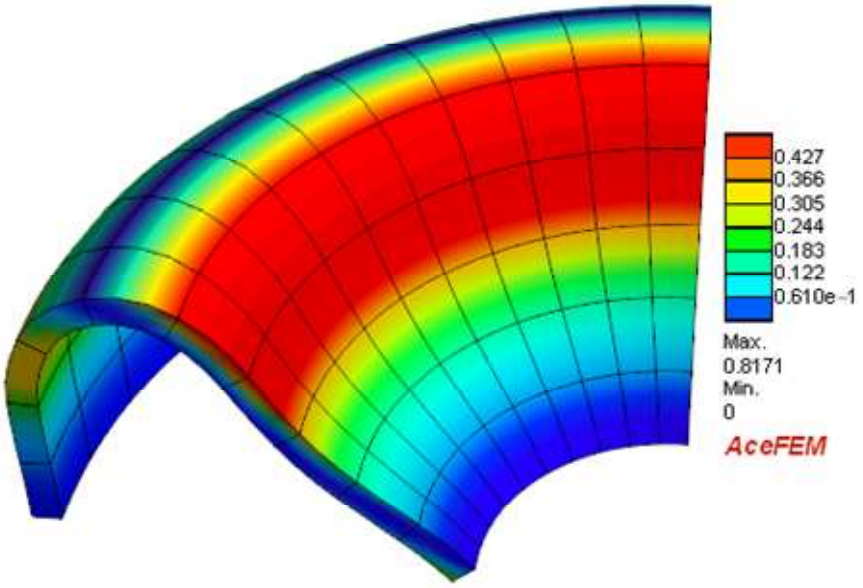
In[1286]:=
```
path = {{0, 0}};
```

In[1287]:=
```
SMTNextStep["λ" → .01];
While[
  While[
   step = SMTConvergence[10^-8, 15, {"Adaptive", 10, .001, .2, 1}],
   SMTNewtonIteration[];
  ];
  If[Not[step[[1]]],
   AppendTo[path, {SMTData["Multiplier"], Total[SMTResidual["Z" == 0 && "ID" == "D" &]][[3]]}]];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  ,
  If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" → step[[2]]]
 ];
```

In[1289]:=
```
ListLinePlot[path, PlotStyle → Red]
```



In[1290]:=
```
SMTShowMesh["DeformedMesh" -> True, "Field" → SMTPostData["u"]^2 + SMTPostData["v"]^2]
```

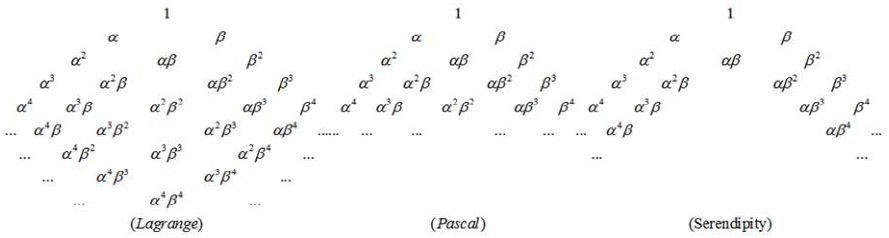

0.427
0.366
0.305
0.244
0.183
0.122
0.610e-1
Max.
0.8171
Min.
0
AceFEM

Semi-analytical Solutions

Semi-analytical solution of the finite element problem is a solution of the problem where the results are expressed as a power series expansion with respect to one or several parameters of the problem. The expansion parameters, the expansion point and the order of the power series expansion is specified by the SMTInputData options "SeriesData" and "SeriesMethod" (see `SMTInputData`). In the case of the full multivariate power series expansion the number of terms grows exponentially. The number of terms can be reduced by the reduced expansion.

KORELC, J. Semi-analytical solution of path-independent nonlinear finite element models. Finite elem. anal. des., 2011, 47:281-287.

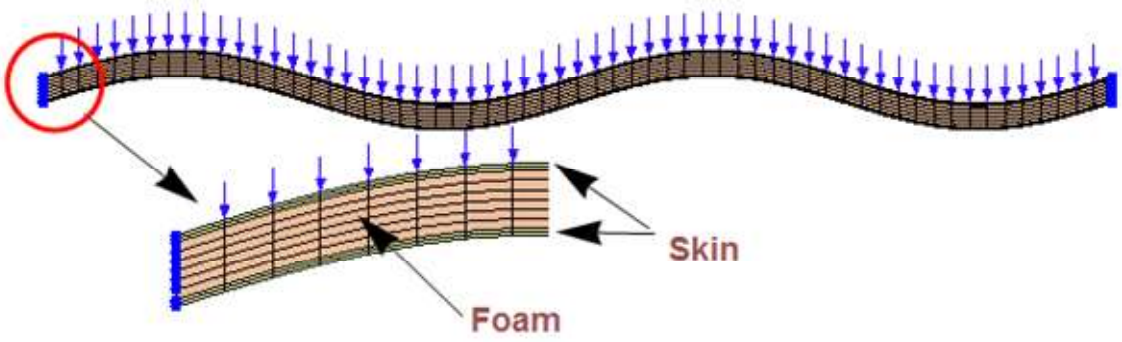
Three types of reduces expansion ("Lagrange", "Pascal", and "Serendipity") are available as depicted in a table below.



expansion	description
"Lagrange"	full multivariate series expansion
"Pascal"	multivariate series expansion in a form of Pascal triangle
{"Serendipity",*n*}	multivariate series expansion with the secondary terms up to the *n*th power

Possible values for the "SeriesMethod" option of the SMTInputData command.

Example: Bending of the sinusoidal double skin cladding



The goal of the presented example is to find a semi-analytical linear elastic solution for the bending of the sinusoidal double skin cladding. The solution employes the fifth order power series expansion with respect to thickness of the foam and the amplitude of the waves. The "Serendipity" type multivariate series expansion with the secondary terms up to the second power is used. The symbol α is used for the foam thickness and the symbol β for the amplitude of the waves .

In[330]:=
```
<< AceFEM`;
L = 400.;
b = 100; hwave0 = 10;
nwave = 10; T0foam = L / 200;
T0steel = 0.2;
qz0 = b 2.4 × 10^-4;
nx = 60; ny = 6; δh = 2 T0steel / ny 0.5; dx = L / (10. (hwave0 + T0foam));
Clear[α, β];
Tfoam = Series[α, {α, T0foam, 5}];
Tsteel = T0steel;
hwave = Series[β, {β, hwave0, 5}];
qz = qz0;
SMTInputData["NumericalModule" → "MDriver",
  "SeriesData" → {{α, T0foam, 5}, {β, hwave0, 5}}, "SeriesMethod" → {"Serendipity", 2}];
SMTAddDomain["Foam", {"ML:", "SE", "PS", "Q1", "DF", "LE", "Q1", "D", "Hooke"},
  {"E *" -> 500., "ν *" -> 0.48, "t *" -> b}];
SMTAddDomain["Steel", {"ML:", "SE", "PS", "Q1", "DF", "LE", "Q1", "D", "Hooke"},
  {"E *" -> 21000, "ν *" -> 0.3, "t *" -> b}];
SMTAddMesh[Raster[{Table[{x, hwave / 2 Sin[nwave π x / L] - Tfoam / 2}, {x, 0, L, dx}], Table[
    {x, hwave / 2 Sin[nwave π x / L] + Tfoam / 2}, {x, 0, L, dx}]}], "Foam", "Q1", {nx, ny}];
SMTAddMesh[Raster[{Table[{x, hwave / 2 Sin[nwave π x / L] + Tfoam / 2}, {x, 0, L, dx}],
    Table[{x, hwave / 2 Sin[nwave π x / L] + Tfoam / 2 + Tsteel},
     {x, 0, L, dx}]}], "Steel", "Q1", {nx, 2}];
SMTAddMesh[Raster[{Table[{x, hwave / 2 Sin[nwave π x / L] - Tfoam / 2 - Tsteel}, {x, 0, L, dx}],
    Table[{x, hwave / 2 Sin[nwave π x / L] - Tfoam / 2},
     {x, 0, L, dx}]}], "Steel", "Q1", {nx, 2}];
SMTAddNaturalBoundary[Abs[hwave / 2 Sin[nwave π "X" / L] + Tfoam / 2 + Tsteel - "Y"] <= δh &,
  2 -> -qz L / nx];
SMTAddEssentialBoundary[("X" == 0 || "X" == L) &, 1 -> 0., 2 -> 0.];
SMTAnalysis[];
```

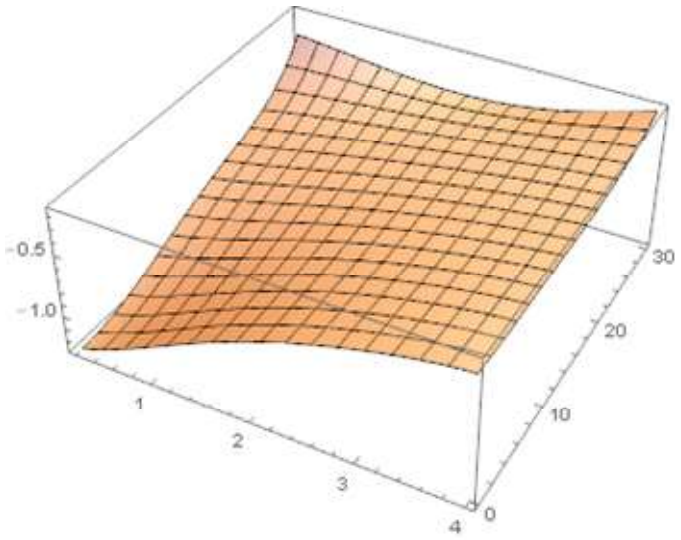
In[2]:=
```
SMTNextStep["λ" → 1];
SMTNewtonIteration[];
```

Here is the deflection in the middle of the beam retrieved from the data based and transformed into normal form. The transformation to the normal form (Normal[i]) is necessary for the symbolic manipulations later.

In[369]:=
```
w = Normal[SMTNodeData["X" == L / 2 &, "at"][[1, 2]]]
```

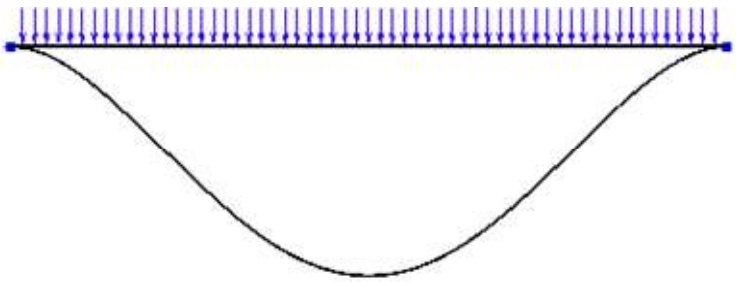
In[370]:=
```
wc = w /. α -> T0foam /. β -> hwave0
```

In[371]:=
```
Plot3D[w, {α, 0.1 T0foam, 2 T0foam}, {β, 0, 3 hwave0}]
```

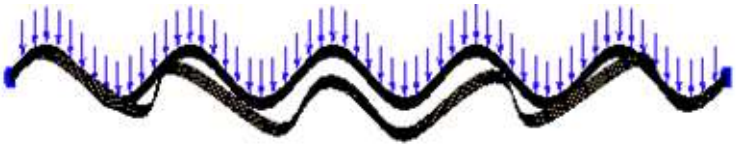


The "ShowFor" option of the SMTShowMesh command can be used to depict mesh and results for arbitrary values of parameters.

In[372]:=
```
Show[SMTShowMesh["BoundaryConditions" -> True, "ShowFor" -> {α -> 0.1 T0foam, β -> 0}],
 SMTShowMesh["DeformedMesh" -> True, "Scale" -> 100, "ShowFor" -> {α -> 0.1 T0foam, β -> 0}]]
```



In[373]:=
```
Show[SMTShowMesh["BoundaryConditions" -> True, "ShowFor" -> {α -> 3 T0foam, β -> 3 hwave0}],
 SMTShowMesh["DeformedMesh" -> True, "Scale" → 100, "ShowFor" -> {α -> 3 T0foam, β -> 3 hwave0}]]
```



Stochastic Analysis

Contents

Theory

Stochastic analysis

With AceFEM/AceGen two types of stochastic analysis can be performed:

- when an input parameter of the problem is random, it can be modelled as stochastic variable,
- when an input parameter of the problem is random and it also randomly varies over the domain, it can be modelled as stochastic field.

Modeling of stochastic fields requires additional discretization of stochastic field.

In both cases the statistics of the response (mean value and standard deviation) has to be evaluated at the end. Two approaches will be presented:

- standard Monte-Carlo method,
- perturbation method based on second order sensitivity analysis.

Advantage of the perturbation method based on second order sensitivity analysis is that only one direct simulation of the global FE model is needed. Monte-Carlo method typically requires several 1000 direct simulations.

Summary of examples

Several examples are presented at the end to demonstrate implementation of different approaches to stochastic analysis. First examples are simple and problem specific. At the end the stochastic analysis environment is introduced which is fully parameterized and general and it can be used as a template for the user defined stochastic analysis formulations.

- Material parameter as stochastic variable, time independent, hyper-elastic problem, parameter is modeled as stochastic variable, perturbation approach;
- Material parameter as stochastic field, time independent, hyper-elastic problem, parameter is modeled as stochastic field, perturbation approach;
- Stochastic analysis with Monte – Carlo method, time independent, hyper-elastic problem, parameter is modeled as stochastic variable, Mote-Carlo simulations;

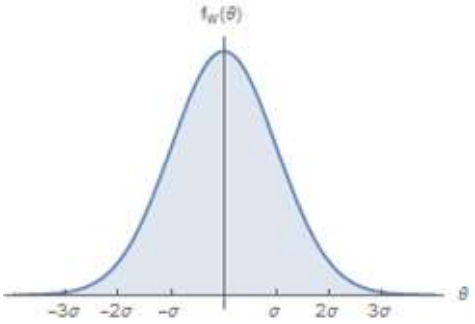
- `Environment for stochastic analysis of general time dependent problems` contains functions for the generation of KL mesh and solution of KL decomposition
- `Stochastic analysis of general time dependent problem`, time dependent elasto-plastic problem, parameter is modeled as stochastic field, perturbation approach.

Stochastic fields

When an input parameter of the problem is random and it also randomly varies over the domain, it can be modelled as stochastic field. A stochastic field is defined with:

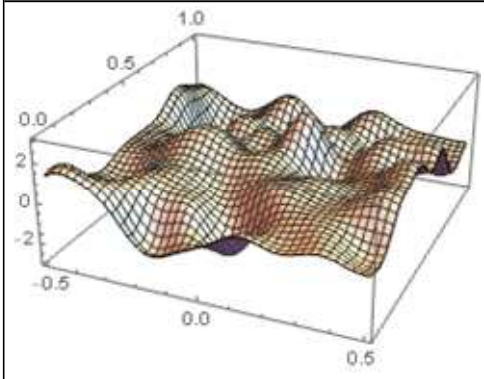
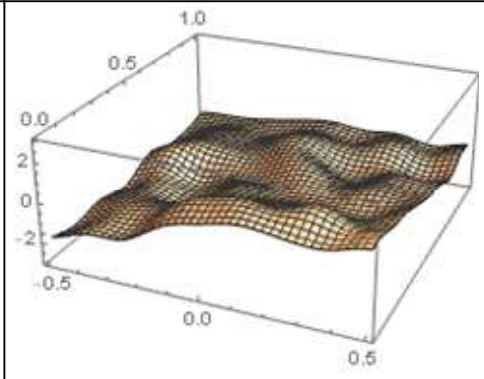
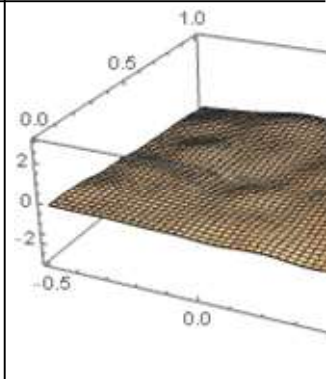
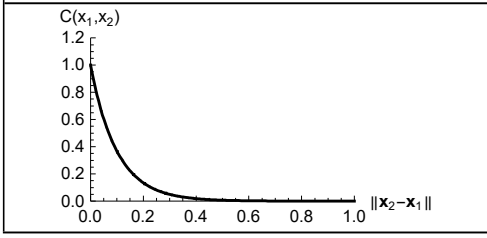
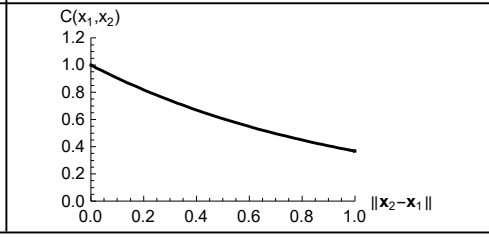
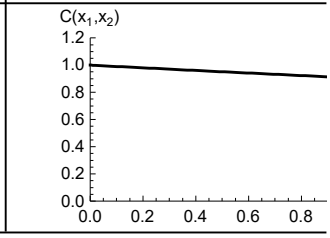
- Probability density function

  Probability density function specifies the probability of the random variable falling within a particular range of values. Most often used is Gaussian (or Normal):



- Covariance function

  Covariance function describes how much a variable changes along the domain. In mechanical problems, most often used is exponential covariance function $C(x_1, x_2) = \sigma^2 e^{\frac{\|x_2 - x_1\|}{l_c}}$, where x_i is a position vector over the physical domain, σ is standard deviation and l_c is correlation length. The bigger l_c is, the higher correlated is stochastic field.

"Low-correlated stochastic field"	"Moderately-correlated stochastic field"	"Highly-correlated stocha
		
		

The representation of the Gaussian stochastic field can be done with Karhunen-Loève expansion, which is truncated after first M terms:

$$Em(x,\theta) = \overline{Em}(x) + \sum_{k=1}^{M} \sqrt{\lambda_k}\, f_k(x)\, \xi_k(\theta), \qquad (1)$$

where x is a position vector over the physical domain D, θ is an event of the space of random events, $\overline{Em}(x)$ is expected value of the

stochastic field and $\xi_k(\theta)$ are normalized uncorrelated Gaussian random variables with zero mean and unit variance. λ_k and $f_k(x)$ are the eigenvalues and eigenvectors, respectively, obtained as the solution of the homogeneous Fredholm integral equation of the second kind with covariance function $C(x_1, x_2)$ as kernel:

$$\int_D C(x_1, x_2) f_k(x_1) dl x_1 = \lambda_k f_k(x_2) \qquad (2)$$

Galerkin procedure is used to solve this equation numerically. By this procedure, k-th eigenfunction is approximated as linear combination of J shape functions $N_j(x)$

$$f_k(x)=\sum_{j=1}^{J} f_{jk} N_j(x) \qquad (3)$$

Introducing Eq.(3) in Eq.(2) and fulfillment of the requirement of error to be orthogonal to the space spanned by the shape functions leads to a generalized eigenvalue problem

$$\mathbf{C f} = \mathbf{\Lambda N f}, \qquad (4)$$

where C is covariance matrix, N is matrix of eigenfunctions:

$$C_{ij} = \int_D \int_D C(x_1, x_2) N_j(x_1) N_i(x_2) dl x_1 dl x_2 \qquad (5)$$

$$N_{ij} = \int_D N_j(x_2) N_i(x_2) dl x_2 \qquad (6)$$

$$\Lambda_{lk} = \delta_k \lambda_k \qquad (7)$$

The eigenfunctions $f_k(x)$ are then obtained from the eigenvectors according toEq. (3) and should be normalized as follows

$$\int_D f_k(x) f_l(x) dl x = \delta_{kl} \qquad (8)$$

For details see e.g. MELINK, Teja, KORELC, Jože. Stability of Karhunen- Loève expansion for the simulation of Gaussian stochastic fields using Galerkin scheme. Probabilistic Engineering Mechanics, 2014, 37:7-15, doi: 10.1016/j.probengmech.2014.03.006.

Approximation of the response

When at least one of the input parameters is random, the response of the system is also random. In presented stochastic approach, the response of the problem is approximated with a finite number of its Taylor series around expected values of random variables ${}^0\xi=\{{}^0\xi_1, {}^0\xi_2, \ldots, {}^0\xi_M\}$, which resembles perturbation method. In case of Gaussian stochastic field ${}^0\xi=\{0,0,\ldots,0\}$ and second-order perturbation method:

$$p(\xi_1, \xi_2, \ldots, \xi_M)=p(0, 0, \ldots, 0) +\sum_{i=1}^{n} \frac{\partial p}{\partial \xi_i} \xi_i +\frac{1}{2}\sum_{i=1}^{M} \sum_{j=1}^{M} \frac{\partial^2 p}{\partial \xi_i \partial \xi_j} \xi_i \xi_j$$

where p is solution vector (in mechanics, p is usually vector of displacements). Derivatives of solution vector p with respect to random variables are calculated with sensitivity analysis. The order of derivatives represents the order of perturbation method.

Statistics of the response

Expected value:

$$E(p(\xi_1, \xi_2, \ldots, \xi_M)) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \ldots \int_{-\infty}^{\infty} p(\xi_1, \xi_2, \ldots, \xi_M) f(\xi_1) f(\xi_2) \ldots f(\xi_M) dl \xi_M \ldots dl \xi_2 dl \xi_1$$

$f(\xi_i)$ is probability density function of variable ξ_i (in this case Gaussian distribution with zero mean and unit variance).

Variance:

$$\mathrm{var}(p(\xi_1, \xi_2, \ldots, \xi_M)) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \ldots \int_{-\infty}^{\infty} (p(\xi_1, \xi_2, \ldots, \xi_M) - E(p(\xi_1, \xi_2, \ldots, \xi_M)))^2 f(\xi_1) f(\xi_2) \ldots f(\xi_M) dl \xi_M \ldots dl \xi_2 dl \xi_1$$

Standard deviation:

$$\sigma(p(\xi_1, \xi_2, \ldots, \xi_M)) = \sqrt{\mathrm{var}(p(\xi_1, \xi_2, \ldots, \xi_M))}$$

Implementation of KL Decomposition

Discretization of stochastic field

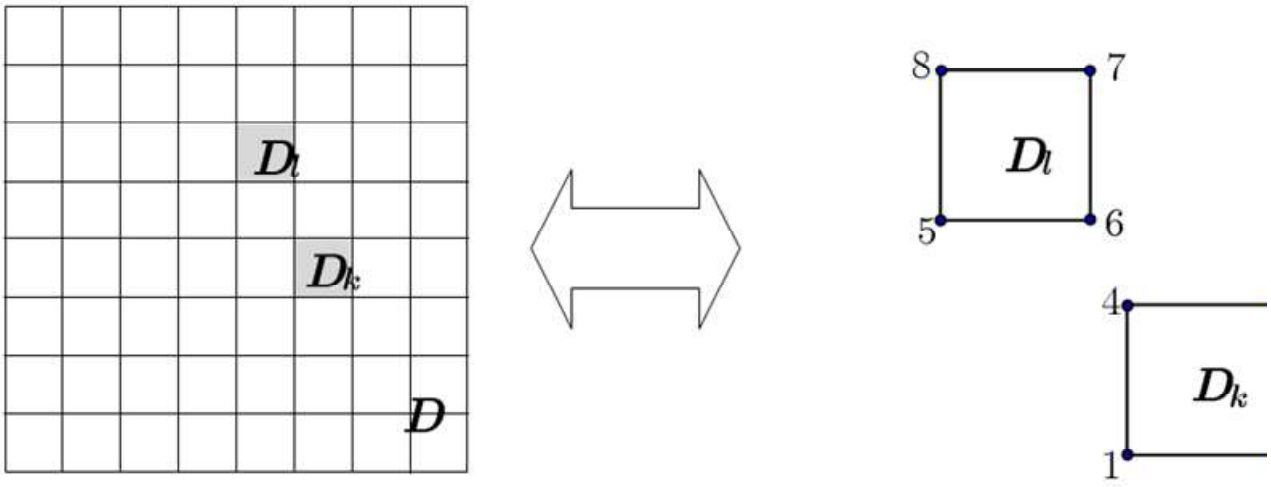
To solve Fredholm integral equation applying Galerkin procedure (Eq. (4)), one needs to calculate matrices C and N. For this purpose, finite elements are formulated, that can perform three tasks:

- Task 1: Calculation of matrix **N** - Eq. (5)
- Task 2: Calculation of matrix **C** - Eq. (6)
- Task 3: Calculation of integral $\int_D f_k(x) f_k(x) dx$ - Eq. (8)

Finite element for calculation of stochastic field

The finite element implementation is not straightforward as in usual mechanical problems. The reason is that in calculation of the contribution of single finite element to the global covariance matrix C, the distance between finite elements needs to be considered. To solve this problem, the stochastic finite elements are formulated as follows.

The physical domain D is firstly divided into n_D sub-elements D_k. In next step, each sub-element D_k is joined with all sub-elements $D_l \in D$ that are within an effective correlation length l_{eff}, thus $||\mathbf{t}_k - \mathbf{t}_l|| < l_{eff}$ where $\mathbf{t}_k$ and $\mathbf{t}_l$ are the centers of gravity of sub-elements. The matrix **C** is symmetric and therefore only upper triangle of this matrix is calculated which leads to another restriction $l \geq k$ on the selection of sub-elements. Each combination of master subdomains D_k with $D_l : l \geq k \wedge ||\mathbf{t}_k - \mathbf{t}_l|| < l_{eff}$ represents a stochastic finite element. An example of such element for two - dimensional domain D, discretized with linear, quadrilateral elements with topology "Q1":



The stochastic element nodes are composed of nodes of leading sub-element (D_k) and nodes of all its associated sub-elements, thus elementNodes = nodes $(D_k) \cup$ nodes $(D_l) : l \geq k \wedge ||\mathbf{t}_k - \mathbf{t}_l|| < l_{eff}$. The number of nodes is not constants, thus a dummy nodes approach is used to formulate the elements with variable number of nodes. Within the dummy nodes approach the maximum number of nodes is prescribed. Nodes are then divided into true nodes and dummy nodes. The maximum number of nodes depends on the mesh density and l_{eff}, thus the element can be generated at run-time, after the mesh of the problem has been generated. Alternatively, the maximum number of nodes can be fixed and the element is split into several elements such that the actual number of nodes does not exceeds he prescribed maximum number of nodes. Corresponding algorithm is implemented in `Environment for stochastic analysis of general time dependent problems`.

Description of tasks:

Task 1 : Calculation of matrix N

Matrix of eigenfunctions N is obtained by a numerical integration rule:

$N_{ij}^e = \sum_{g=1}^{n_g} w_g N_{ij}^g$

$N_{ij}^g = J_g N_i(x_g) N_j(x_g)$

Task 2 : Calculation of matrix C

Covariance matrix **C** is obtained by a numerical integration rule. The matrix **C** is symmetric and therefore only upper triangle of this matrix is calculated.

$C_{gh} = J_g\, J_h\, \sigma^2\, C\left(\mathbf{X}_g^k, \mathbf{X}_h^l\right) w_g^k\, w_h^l$

$C_{ij}^{gh} = \sum_{g=1}^{n_g} \sum_{h=1}^{n_g} C_{gh}\, N_j^k\left(\boldsymbol{\xi}_g\right) N_i^l\left(\boldsymbol{\xi}_h\right) \;\; \forall\, l : l \geq k \;\wedge\; \|\mathbf{t}_k - \mathbf{t}_l\| < l_{eff}$

where n_g is number of integration points, w_g and w_h are the weights of Gauss integration points g and h, respectively and J_g and J_h are determinants of the Jacobian matrices for the *k*-th and *l*-th sub-elements. Since the sequence of the sub-elements includes only half of the effective area ($l \geq k$), the symmetric part C_{ji}^e is also calculated, but only when the sub-elements D_k and D_l are not the same ($k \neq l$).

$C_{ji}^{gh} = \sum_{g=1}^{n_g} \sum_{h=1}^{n_g} C_{gh}\, N_i^k\left(\boldsymbol{\xi}_g\right) N_j^l\left(\boldsymbol{\xi}_h\right) \;\; \forall\, l : l > k \;\wedge\; \|\mathbf{t}_k - \mathbf{t}_l\| < l_{eff}$

Task 3 : Calculation of integral I_{f_k}

The Gaussian quadrature is performed for the calculation of the integral on the left side of Equation (8). Due to the orthogonality of eigenfunctions, only eigenfunctions with the same indexes have to be coupled

$I_{f_k}^e = \sum_{g=1}^{n_g} w_g\, I_{f_k}^g$

$I_{f_k}^g = J_g \left(\sum_{j=1}^{J} f_{jk}\, N_j(x_g)\right)^2$

The eigenfunctions are then normalized:

$\overline{f_k} = \frac{f_k}{\sqrt{I_{f_k}}}$

Discretization of KL and physical problem

In general KL decomposition and physical response represents solution of entirely different type of equations. Consequently, in general different meshes are required for the optimal discretization of the same domain. Response of solid mechanics problems is governed by very stiff partial differentiation equations, that requires dense meshes in order to avoid various locking phenomena. Denser mesh for the response will be used here than used for the KL decomposition. This is also convenient from the computational point of view since covariance matrix **C** is less sparse than tangent matrix **K**.

Stochastic finite element

In[2]:=
```
<< AceGen`
```

In[3]:=
```
NoTopologicalNodes = 4;
```

- Maximum number of nodes in stochastic element is set to 300*NoTopologicalNodes, thus max 300 subelements is joined into one macro element.

In[4]:=
```
maxNodes = 300 NoTopologicalNodes;
```

In[5]:=
```
SMSInitialize["ExamplesStochasticFE", "Environment" → "AceFEM", "Mode" → "Plain"];
SMSTemplate["SMSTopology" → "Q1"
  , "SMSNoNodes" → maxNodes
  , "SMSAdditionalNodes" → Null
  , "SMSNodeID" → "S -D"
  , "SMSDOFGlobal" → 1
  , "SMSSymmetricTangent" → True
  , "SMSCharSwitch" → {"matrix_N", "matrix_C", "integral_Ifk"}
  , "SMSDomainDataNames" ->
   {"σ -standard deviation", "lc -correlation length", "lceff -effective correlation length"}
  (*ed$$["Data",1]: 0- full integration ≠0-integration only over joined domains*)
  , "SMSNoElementData" → 1
  , "SMSDefaultData" → {1, 1, 1}
 ];
```

In[7]:=
```
SMSStandardModule["Tasks"];
task ▹ SMSIO["Task index"];
```

In[9]:=
```
SMSIf[task < 0];
SMSSwitch[task
  , -1,
  SMSIO[{5, 0, 0, 0, 0}, "Export to", "Task data"];
  , -2,
  SMSIO[{5, 0, 0, 0, 0}, "Export to", "Task data"];
  , -3,
  SMSIO[{1, 0, 0, 0, 1}, "Export to", "Task data"];
 ];
SMSReturn[];
SMSEndIf[];
```

- If switch==1 then current element is one of the elements produced by splitting the element nodes into several parts shorter than maxNodes. This elements are skipped for tasks 1 and 3 in order to prevent double integration over the same domain.

In[13]:=
```
switch ⊨ SMSInteger[SMSIO["Element data"[1]]];
SMSIf[switch == 1 && (task == 1 || task == 3), SMSReturn[]];
```

- Integration over *k*-th sub-element

In[15]:=
```
SMSDo[Ig, 1, SMSIO["No. integration points"]];
{ξg, ηg, ζg} ⊨ SMSIO["Integration point"[Ig]];
wg ⊨ SMSIO["Integration weight"[Ig]];
Nig ⊨ 1 / 4 {(1 - ξg) (1 - ηg), (1 + ξg) (1 - ηg), (1 + ξg) (1 + ηg), (1 - ξg) (1 + ηg)};
Xk ⊨ Table[SMSIO["Nodal coordinates"[i, 1]], {i, NoTopologicalNodes}];
Yk ⊨ Table[SMSIO["Nodal coordinates"[i, 2]], {i, NoTopologicalNodes}];
SMSFreeze[Xg, Nig.Xk];
SMSFreeze[Yg, Nig.Yk];
Jmg ⊨ SMSD[{Xg, Yg}, {ξg, ηg}];
Jdg ⊨ Det[Jmg];
```

- Task 1: Calculation of finite element contribution to matrix N

In[25]:=
```
SMSIf[task == 1];


SMSDo[i, 1, NoTopologicalNodes];
SMSDo[j, 1, NoTopologicalNodes];
Nij ⊨ SMSPart[Nig, j] * SMSPart[Nig, i] * wg * Jdg;
SMSExport[Nij, s$$[i, j], "AddIn" → True];
SMSEndDo[];
SMSEndDo[];
SMSEndIf[];
```

- Task 2: Calculation of finite element contribution to matrix C

In[33]:=
```
SMSIf[task == 2];
```

In[34]:=
```
nNodes ⊨ SMSLastTrueNode[];
nSubelements ⊨ SMSInteger[nNodes / NoTopologicalNodes];
{σS, lc, lceff} ⊨ SMSIO["Domain data"];
```

  - Loop over *l*-th sub-elements (l ≥ k)

In[37]:=
```
SMSDo[lSubelement, switch + 1, nSubelements];
```

  - Integration over *l*-th joined sub-element

In[38]:=
```
joinedNodes ⊨ SMSInteger[(lSubelement - 1) NoTopologicalNodes];
```

In[39]:=
```
SMSDo[Ih, 1, SMSIO["No. integration points"]];
```

In[40]:=
```
{ξh, ηh, ζh} ⊨ SMSIO["Integration point"[Ih]];
wh ⊨ SMSIO["Integration weight"[Ih]];
Nih ⊨ 1 / 4 {(1 - ξh) (1 - ηh), (1 + ξh) (1 - ηh), (1 + ξh) (1 + ηh), (1 - ξh) (1 + ηh)};
Xl ⊨ Table[SMSIO["Nodal coordinates"[joinedNodes + i, 1]], {i, NoTopologicalNodes}];
Yl ⊨ Table[SMSIO["Nodal coordinates"[joinedNodes + i, 2]], {i, NoTopologicalNodes}];
SMSFreeze[Xh, Nih.Xl];
SMSFreeze[Yh, Nih.Yl];
Jmh ⊨ SMSD[{Xh, Yh}, {ξh, ηh}];
Jdh ⊨ Det[Jmh];
```

- Calculate and store contribution of finite element to matrix C:

$$C_{gh} = J_g\, J_h\, \sigma^2\, C\left(\mathbf{X}_g^k, \mathbf{X}_h^l\right) w_g^k\, w_h^l$$

$$C_{ij}^{gh} = \sum_{g=1}^{n_g} \sum_{h=1}^{n_g} C_{gh}\, N_j^k(\xi_g)\, N_i^l(\xi_h) \;\; \forall\, l : l \geq k \;\wedge\; \|\mathbf{t}_k - \mathbf{t}_l\| < l_{eff}$$

$$C_{ji}^{gh} = \sum_{g=1}^{n_g} \sum_{h=1}^{n_g} C_{gh}\, N_i^k(\xi_g)\, N_j^l(\xi_h) \;\; \forall\, l : l > k \;\wedge\; \|\mathbf{t}_k - \mathbf{t}_l\| < l_{eff}$$

In[48]:=
```
Cgh ⊨ σS^2 Exp[-(SMSSqrt[(Xh - Xg)^2 + (Yh - Yg)^2] / lc)] * wg * wh * Jdg * Jdh;
```

In[49]:=
```
SMSDo[i, 1, NoTopologicalNodes];
SMSDo[j, 1, NoTopologicalNodes];
Cij ⊨ Cgh * SMSPart[Nig, i] * SMSPart[Nih, j];
SMSIO[Cij, "Add to", "Task local matrix"[i, joinedNodes + j]];
SMSIf[ lSubelement ≠ SMSInteger[1],
  Cji ⊨ Cgh * SMSPart[Nig, j] * SMSPart[Nih, i];
  SMSIO[Cji, "Add to", "Task local matrix"[joinedNodes + i, j]];
 ];
SMSEndDo[];
SMSEndDo[];
```

In[56]:=
```
SMSEndDo[];(*l-th subelement Gauss loop*)
```

In[57]:=
```
SMSEndDo[];(*l-th subelements loop*)
```

In[58]:=
```
SMSEndIf[];
```

- Task 3: Calculation of integral I_{f_k}

In[59]:=
```
SMSIf[task == 3];
fjk ⊨ Table[SMSIO["Nodal DOFs"[i, 1]], {i, NoTopologicalNodes}];
SMSFreeze[fkg, Nig.fjk];
Ifkg = fkg * fkg * wg * Jdg;
SMSIO[Ifkg, "Add to", "Task real output"[1]]
SMSEndIf[];
```

Export[fkg^2 Jdg wg, RealOutput$$1]

- End of k-th subelement Gauss loop

In[65]:=
```
SMSEndDo[];
```

- Write source code

In[66]:=
```
SMSWrite[];
```

File: ExamplesStochasticFE.c Size: 56709 Time: 2

Method	Tasks
No.Formulae	105
No.Leafs	1679

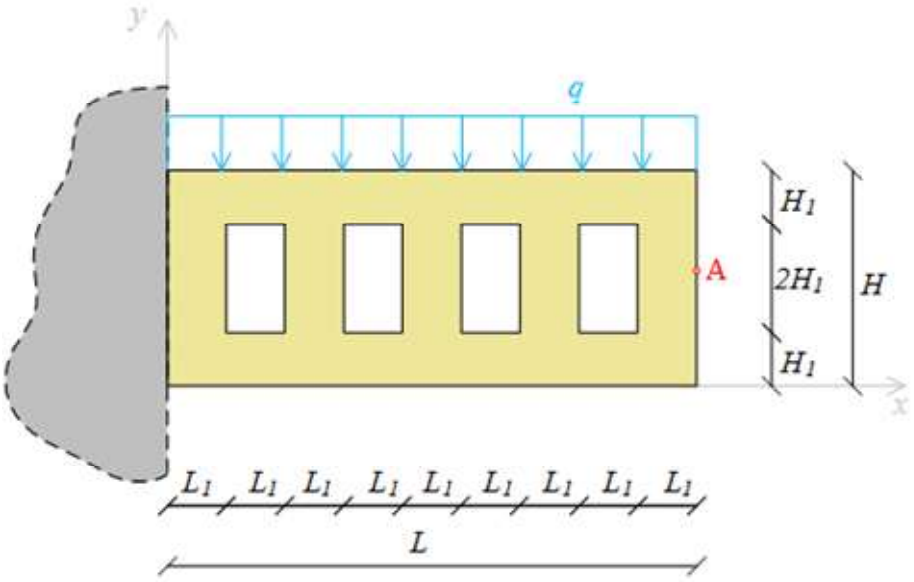
Material parameter as stochastic variable

In[67]:=
```
<< AceFEM`
```

Cantilever beam with material parameters as stochastic variables

A cantilever beam with holes is exposed to vertical load. In this example, the randomness of Young's modulus is considered. Young's modulus is modelled as stochastic variable with mean Em0 and standard deviation σEm. Since the material is hyper-elastic, the problem is time independent.

Statistics of the response is calculated using the perturbation method. To calculate Taylor series of response around expected values of random variable, derivatives of solution vector with respect to random variable is calculated. The derivatives are obtained with sensitivity analysis. The velocity fields of sensitivity problem are Identity fields.



Input parameters

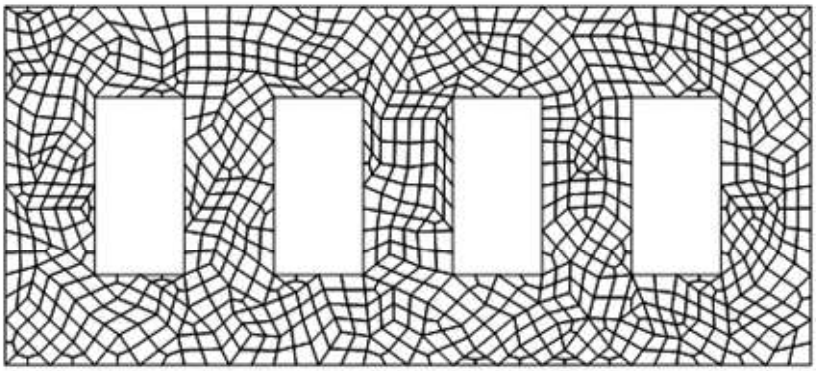
In[68]:=
```
L = 90;
H = 40;
q = 1;
Em0 = 1000;
σEm = 100;
```

Evaluation of response

Create mesh for evaluation of response

In[73]:=
```
mesh = SMTTriangularToQuad[
    ToElementMesh[Polygon[{{{0, 0}, {90, 0}, {90, 20}, {80, 20}, {80, 10}, {70, 10},
        {70, 20}, {60, 20}, {60, 10}, {50, 10}, {50, 20}, {40, 20}, {40, 10},
        {30, 10}, {30, 20}, {20, 20}, {20, 10}, {10, 10}, {10, 20}, {0, 20}},
       {{0, 40}, {90, 40}, {90, 20}, {80, 20}, {80, 30}, {70, 30}, {70, 20}, {60, 20},
        {60, 30}, {50, 30}, {50, 20}, {40, 20}, {40, 30}, {30, 30}, {30, 20}, {20, 20},
        {20, 30}, {10, 30}, {10, 20}, {0, 20}}}], "MaxCellMeasure" → 10, "MeshOrder" → 1]];
```

In[74]:=
```
mesh["Wireframe"]
```



Second order Sensitivity analysis

In[75]:=
```
SMTInputData[];
SMTAddDomain[{"CantileverBeam",
   "ExamplesStochaticSEPEQ1DFHYQ1DNeoHookeWA", {"E *" → Em0, "ν *" → 0.3, "t *" → 1}}];
SMTAddMesh[mesh, "CantileverBeam"];
SMTAddEssentialBoundary[{Line[{{0, 0}, {0, H}}], 1 → 0, 2 → 0}];
SMTAddNaturalBoundary[{Line[{{0, H}, {L, H}}], 2 → Line[{-q}]}];
```

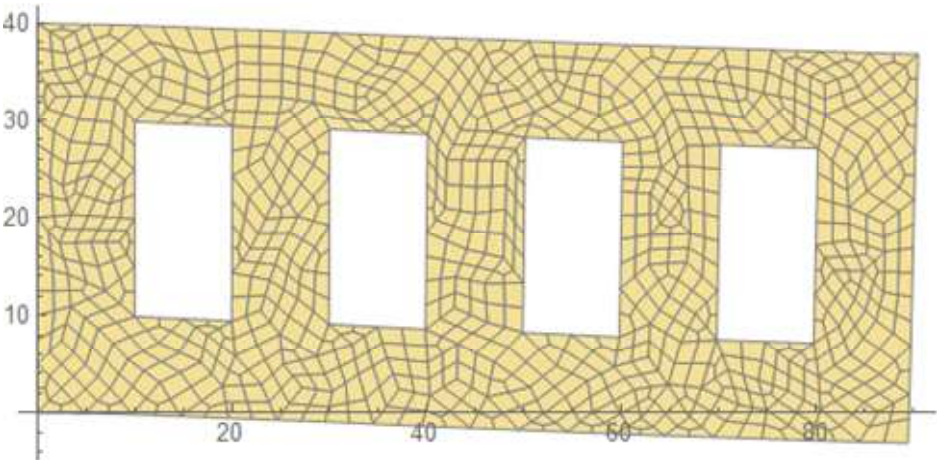
- Definition of second order parameter sensitivity analysis. Elastic modulus appears as 4-th parameter in a list of all sensitivity parameters. This is element specific and it should be checked for every element used!

In[80]:=
```
SMTSensitivityProblem[{{"Em", {"P", Identity, All → 4}}}, "Order" → 2]
SMTAnalysis[];
```

- Solution of primal problem.

In[82]:=
```
SMTNextStep["λ" → 1];
While[SMTConvergence[10^-12, 10], SMTNewtonIteration[]];
```

In[84]:=
```
SMTShowMesh["DeformedMesh" → True, Axes → True]
```



- Solution of sensitivity problem. The problem is time independent, therefore, `SMTForwardSensitivity` command has to be used only at the point where the results of sensitivity analysis are required.

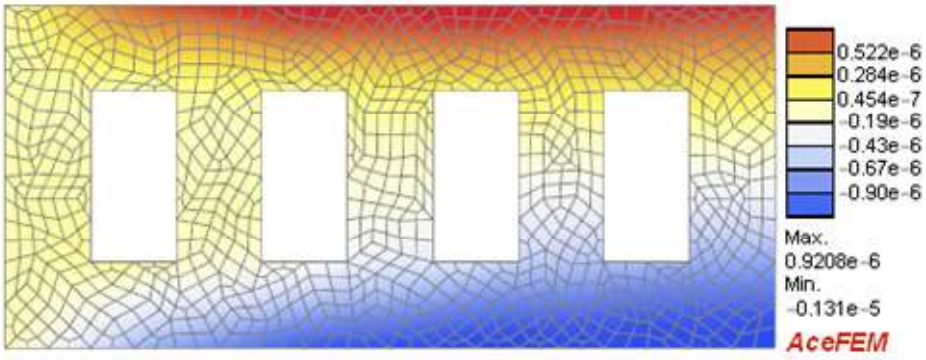
In[85]:=
```
SMTForwardSensitivity[];
```

- Graphical representation of selected first- and second-order sensitivities:
  - Sensitivity of vertical displacement with respect to Em ($\frac{\partial v}{\partial \mathrm{Em}}$):

In[86]:=
```
SMTShowMesh["Field" → SMTPostData[{"st", 2, "Em"}], ImageSize → 400]
```

- Second-order sensitivity of horizontal displacement with respect to Em ($\frac{\partial^2 u}{\partial Em^2}$):

In[87]:=
```
SMTShowMesh["Field" → SMTPostData[{"st", 1, {"Em", "Em"}}], ImageSize → 400]
```



Statistics of response

- Taylor expansion of second order

  Response of interest is vertical displacement v_A in central point at the free end of cantilever beam (point {L, H/2}) and the statistics: mean, variance and standard deviation.

  Taylor expansion of second order of the response of vertical displacement v_A is:

  $v_A(Em) = v_A(Em0) + \frac{\partial v_A}{\partial Em}\Delta Em + \frac{1}{2}\frac{\partial^2 v_A}{\partial Em\,\partial Em}\Delta Em\,\Delta Em$

In[88]:=
```
Clear[Em];
vA = SMTPostData["v", Point[{L, H/2}]] +
  SMTPostData[{"st", 2, "Em"}, Point[{L, H/2}]] (Em - Em0) +
  1/2 SMTPostData[{"st", 2, {"Em", "Em"}}, Point[{L, H/2}]] (Em - Em0) (Em - Em0)
```

Out[88]=
$$-3.10474 + 0.00313519\,(-1000 + Em) - 3.16511\times 10^{-6}\,(-1000 + Em)^2$$

- Expected value of v_A

In[90]:=
```
EvA =
 NExpectation[vA, {Em ≈ NormalDistribution[Em0, σEm]}, PrecisionGoal → 3, Method → "MonteCarlo"]
```

Out[90]=
$$-3.13659$$

- Variance of v_A

In[91]:=
```
VarvA = NExpectation[(vA - EvA)^2,
  {Em ≈ NormalDistribution[Em0, σEm]}, PrecisionGoal → 3, Method → "MonteCarlo"]
```

Out[91]=
$$0.100448$$

- Standard deviation of v_A

In[92]:=
```
σvA = √VarvA
```
```
0.316935
```

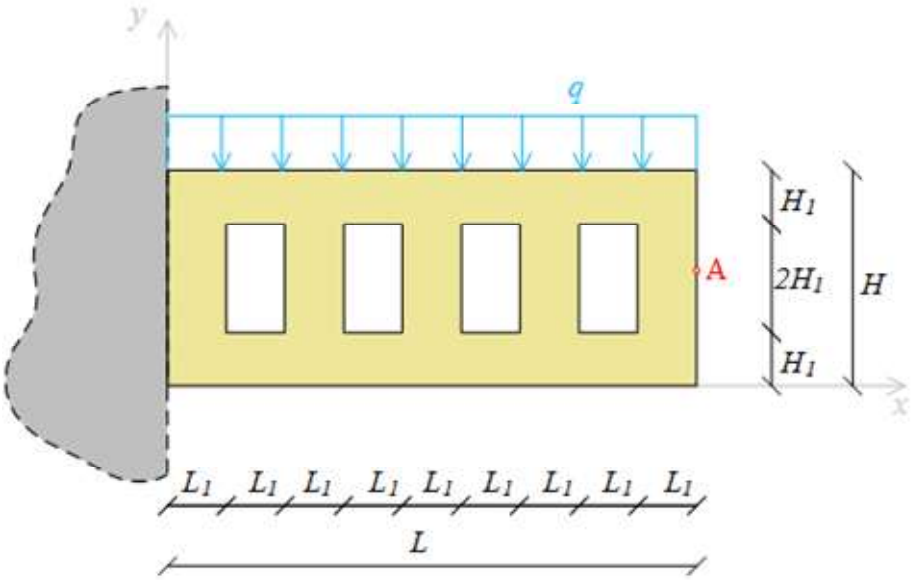
Material parameter as stochastic field

In[93]:=
```
<< AceFEM`
```

Cantilever beam with Young's modulus as stochastic field

In this example, the randomness of Young's modulus is considered. Therefore, Young's modulus is modelled with Gaussian stochastic field with mean Em0 and standard deviation σEm. Stochastic field is represented with Karhunen - Loève expansion. The values of Young's modulus along the domain are fairly correlated, therefore only first four terms of Karhunen - Loève expansion are retained (M=4). Since the material is hyper-elastic, the problem is time independent. The same mesh is used for KL decomposition and the evaluation of response, thus the input is relatively simple.

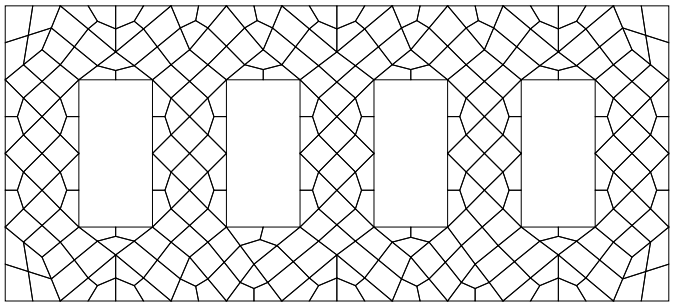
A cantilever beam with holes is exposed to vertical load.



Input parameters

In[94]:=
```
L = 90;
H = 40;
q = 1;
Em0 = 1000;
σEm = 100;
lc = L / 2;
lceff = 4 lc;
M = 4;
```

In[102]:=
```
mesh = SMTTriangularToQuad[
    ToElementMesh[Polygon[{{{0, 0}, {90, 0}, {90, 20}, {80, 20}, {80, 10}, {70, 10},
        {70, 20}, {60, 20}, {60, 10}, {50, 10}, {50, 20}, {40, 20}, {40, 10},
        {30, 10}, {30, 20}, {20, 20}, {20, 10}, {10, 10}, {10, 20}, {0, 20}},
       {{0, 40}, {90, 40}, {90, 20}, {80, 20}, {80, 30}, {70, 30}, {70, 20}, {60, 20},
        {60, 30}, {50, 30}, {50, 20}, {40, 20}, {40, 30}, {30, 30}, {30, 20}, {20, 20},
        {20, 30}, {10, 30}, {10, 20}, {0, 20}}}], "MaxCellMeasure" → 40, "MeshOrder" → 1]];
mesh[
 "Wireframe"]
```



Discretization of stochastic field

- *meshElements* is a list of node indices of each sub-element D_k.

In[104]:=
```
X = mesh["Coordinates"];
meshElements = Level[mesh["MeshElements"], {3}];
```

- *massCenter* is a list of mass centers ($\mathbf{t}_k$) of all D_k.

In[106]:=
```
massCenter = Table[RegionCentroid[Polygon[X[[e]]]], {e, meshElements}];
```

- For each D_k find all D_l: $||\mathbf{t}_k - \mathbf{t}_l|| < l_{eff}$.

In[107]:=
```
nearestElements = Nearest[massCenter → Automatic, massCenter, {Infinity, lceff}];
```

- Delete all $D_l : l < k$.

In[108]:=
```
nearestElements = Table[DeleteCases[e, _?(# < e[[1]] &)], {e, nearestElements}];
```

- Get true nodes of all stochastic elements.

In[109]:=
```
nodesStochasticFE = Table[Flatten[meshElements[[e]]], {e, nearestElements}];
```

- Here, the true maximum number of nodes should not exceed the maximum number of nodes set in element. See `Environment for stochastic analysis of general time dependent problems` for a more general approach.

In[110]:=
```
trueMaxNodes = Max[Table[Length[e], {e, nodesStochasticFE}]]
```

Out[110]=
```
1136
```

Calculation of Karhunen-Loève expansion

In[111]:=
```
SMTInputData[];
SMTAddDomain["CantileverBeam", "ExamplesStochasticFE", {"σ*" → σEm, "lc*" → lc}];
SMTAddMesh[mesh["Coordinates"], {"CantileverBeam" → nodesStochasticFE}];
SMTAnalysis[];
```

- Execution of Task 1 : calculation of matrix N:

In[115]:=
```
Nij = SMTTask["matrix_N"];
```

- Execution of Task 2 : calculation of matrix C:

In[116]:=
```
Cij = SMTTask["matrix_C"];
```

- Solution of generalized eigenvalue problem C f = Λ N f:

In[117]:=
```
{λ, fDOF} = Eigensystem[{Cij, Nij}, M];
```

- Map solution vector into nodes. The SMTNodeData["DOF"] returns a map between nodal DOF indices and nodal indices. DOF indices start with 0.

In[118]:=
```
fNode = Table[Extract[fDOF[[i]], SMTNodeData["DOF"] + 1], {i, 1, M}];
```

- Execution of Task 3 : calculation of integrals $\int_D f_k(x)\, f_k(x)\, dx$, k=1,...,M.

In[119]:=
```
Ifk = Table[
    SMTNodeData["at", Partition[fNode[[i]], 1]];
    SMTTask["integral_Ifk"]
    , {i, 1, M}];
```

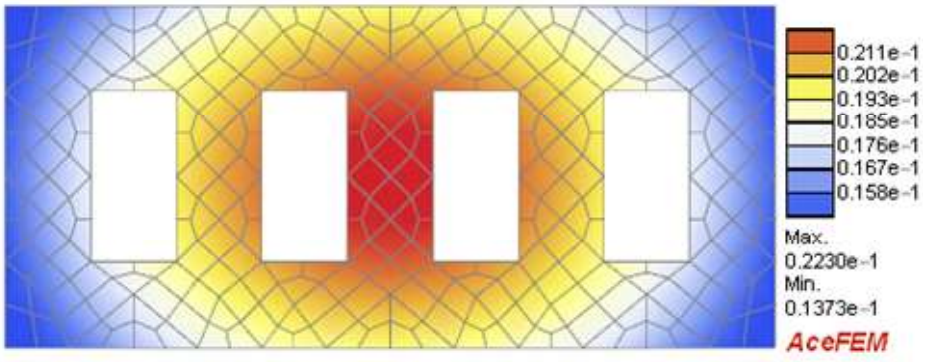
- Normalization of discretized eigenfunctions:

In[120]:=
```
fk = fNode / Sqrt[Ifk];
```

Visualize Karhunen-Loève eigenfunctions

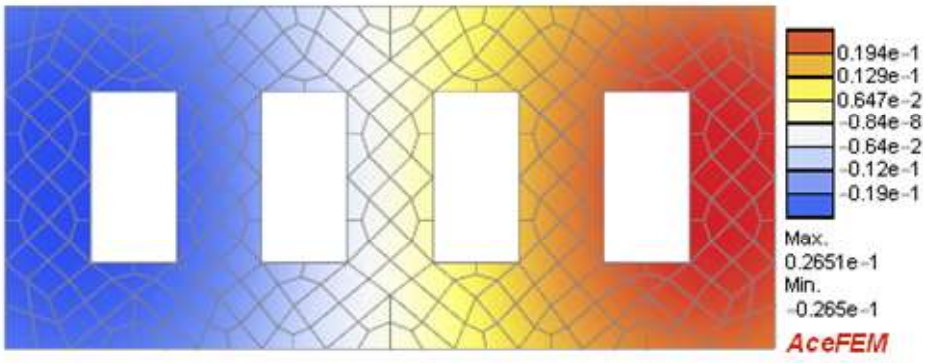
- Eigenfunction f_1:

In[121]:=
```
SMTShowMesh["Field" → fk[[1]], ImageSize → 400]
```



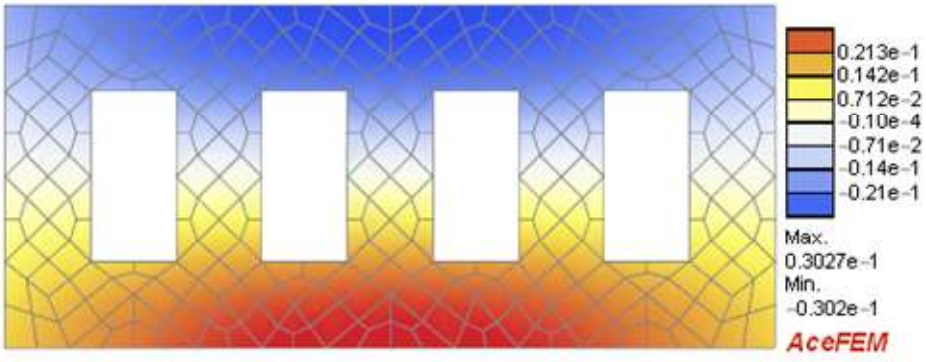
- Eigenfunction f_2:

In[122]:=
```
SMTShowMesh["Field" → fk[[2]], ImageSize → 400]
```



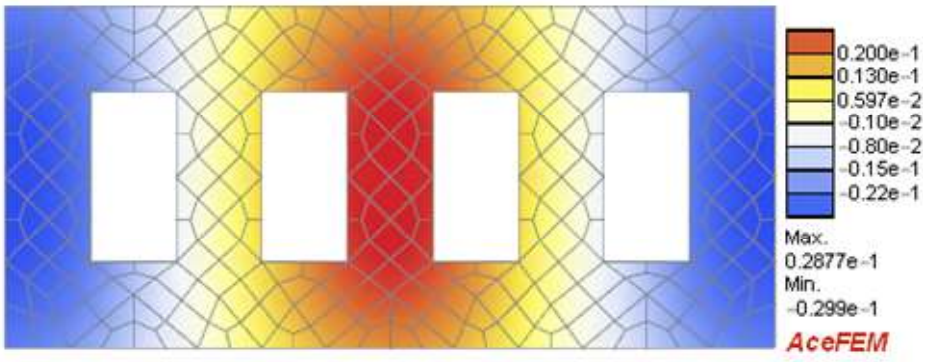
- Eigenfunction f_3:

In[123]:=
```
SMTShowMesh["Field" → fk[[3]], ImageSize → 400]
```



- Eigenfunction f_4:

In[124]:=
```
SMTShowMesh["Field" → fk[[4]], ImageSize → 400]
```



Evaluation of response

To evaluate Taylor series of response around expected values of random variables, derivatives of solution vector with respect to random variables have to be calculated. The derivatives are obtained with sensitivity analysis.

To define velocity fields, equation (1) has to be differentiated with respect to random variables ξ_i:

$$\frac{\partial \mathrm{Em}(x,\theta)}{\partial \xi_i} = \sqrt{\lambda_i}\, f_i(x)$$

$$\frac{\partial^2 \mathrm{Em}(x,\theta)}{\partial \xi_i \partial \xi_j} = 0$$

Thus, weighted eigenfunctions of the KL decomposition are exactly the first order velocity fields of the sensitivity problem.

- AceFEM session where KL decomposition was calculated is still active, thus the eigenfunctions can be easily evaluated for the new mesh simply by using *SMTPostData* command. *SMTPostData* command is using least square patch recovery procedure in order to get accurate transition from KL mesh to response mesh.

In[125]:=
```
velocityFields =
    Table[SMTPostData[fk[[f]]*Sqrt[λ[[f]]], Point[mesh["Coordinates"]]], {f, 1, M}];
```

Primal and second order sensitivity analysis

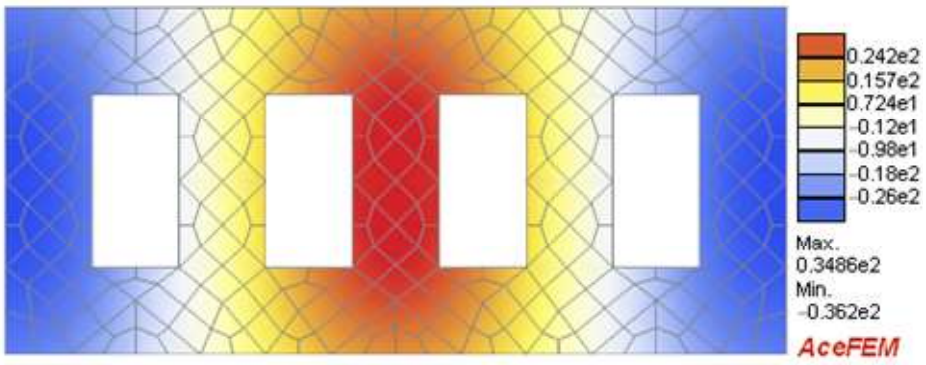
To perform second-order perturbation method, derivatives up to second order have to be calculated.

In[126]:=
```
SMTInputData[];
SMTAddDomain[{"CantileverBeam",
   "ExamplesStochaticSEPEQ1DFHYQ1DNeoHookeWA", {"E *" → Em0, "ν *" → 0.3, "t *" → 1}}];
SMTAddMesh[mesh, "CantileverBeam"];
SMTAddEssentialBoundary[{Line[{{0, 0}, {0, H}}], 1 → 0, 2 → 0}];
SMTAddNaturalBoundary[{Line[{{0, H}, {L, H}}], 2 → Line[{-q}]}];
SMTSensitivityProblem[{
   {"ξ1", {"P", "∂Em/∂ξ1", All → 4}},
   {"ξ2", {"P", "∂Em/∂ξ2", All → 4}},
   {"ξ3", {"P", "∂Em/∂ξ3", All → 4}},
   {"ξ4", {"P", "∂Em/∂ξ4", All → 4}}
  }
  , "Order" → 2];
SMTAnalysis["NodeReordering" → False];
```

In[133]:=
```
SMTSetVelocityFields[{
   "∂Em/∂ξ1" → {All, fk[[1]] * Sqrt[λ[[1]]]},
   "∂Em/∂ξ2" → {All, fk[[2]] * Sqrt[λ[[2]]]},
   "∂Em/∂ξ3" → {All, fk[[3]] * Sqrt[λ[[3]]]},
   "∂Em/∂ξ4" → {All, fk[[4]] * Sqrt[λ[[4]]]}
  }];
```

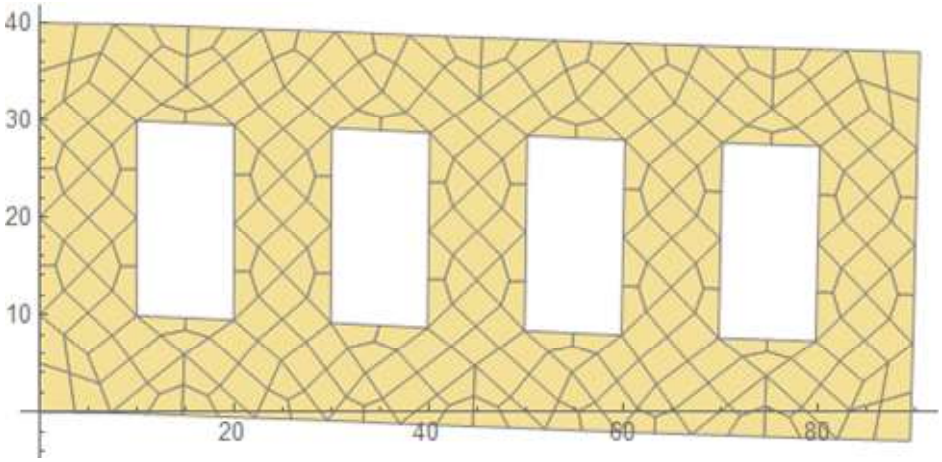
- 4-th eigenfunction is shown here again for denser mesh.

In[134]:=
```
SMTShowMesh["Field" → fk[[4]] * Sqrt[λ[[4]]], ImageSize → 400]
```



In[135]:=
```
SMTNextStep["λ" → 1];
While[SMTConvergence[10^-12, 10], SMTNewtonIteration[]];
```

In[137]:=
```
SMTShowMesh["DeformedMesh" → True, Axes → True]
```

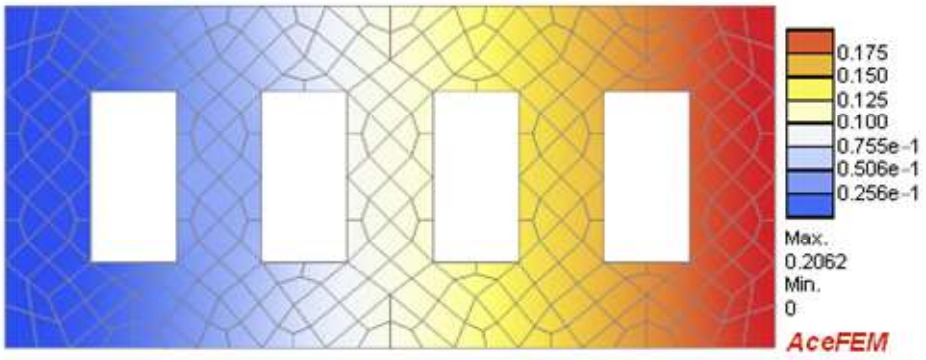


- Solution of sensitivity problem. The problem is time independent, therefore, `SMTForwardSensitivity` command has to be used only at the point where the results of sensitivity analysis are required.

In[138]:=
```
SMTForwardSensitivity[];
```

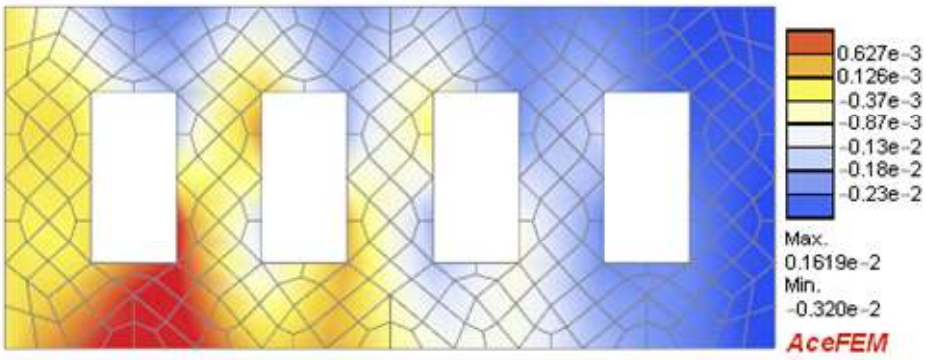
- Graphical representation of some first- and second-order calculated sensitivities:
  - Sensitivity of vertical displacement with respect to $\xi 1$ ($\frac{\partial v}{\partial \xi_1}$):

In[139]:=
```
SMTShowMesh["Field" → SMTPostData[{"st", 2, "ξ1"}], ImageSize → 400]
```



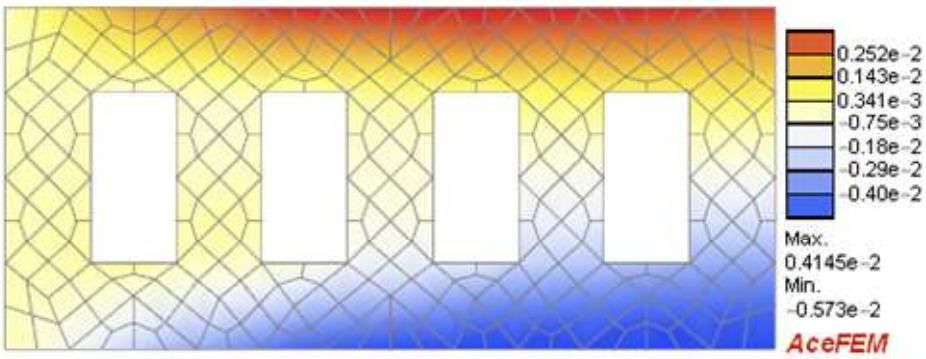
  - Sensitivity of vertical displacement with respect to $\xi 3$ ($\frac{\partial v}{\partial \xi_3}$):

In[140]:=
```
SMTShowMesh["Field" → SMTPostData[{"st", 2, "ξ3"}], ImageSize → 400]
```



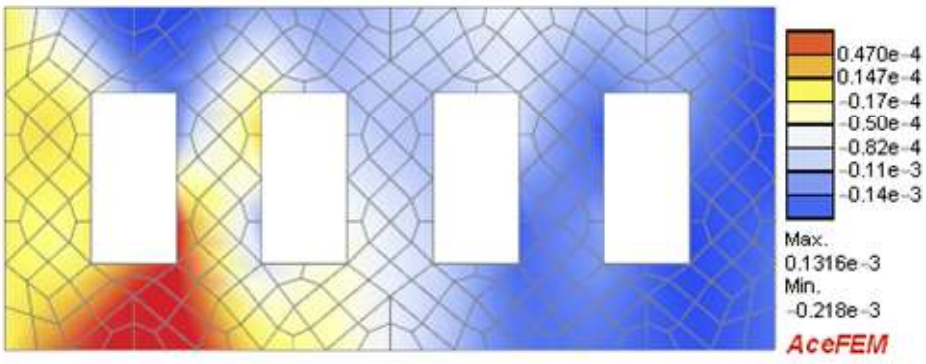
  - Second-order sensitivity of horizontal displacement with respect to $\xi 1$ ($\frac{\partial^2 u}{\partial \xi_1^2}$):

In[141]:=
```
SMTShowMesh["Field" → SMTPostData[{"st", 1, {"ξ1", "ξ1"}}], ImageSize → 400]
```



  - Second-order sensitivity of vertical displacement with respect to $\xi 2$ and $\xi 3$ ($\frac{\partial^2 v}{\partial \xi_2 \partial \xi_3}$):

In[142]:=
```
SMTShowMesh["Field" → SMTPostData[{"st", 2, {"ξ2", "ξ3"}}], ImageSize → 400]
```



Statistics of response

- Taylor expansion of second order

  Response of interest is vertical displacement v_A in central point at the free end of cantilever beam (point {L, H/2}) and the statistics: mean, variance and standard deviation.

  Taylor expansion of second order of the response of vertical displacement v_A is:

  $v_A(\xi_1, \xi_2, \xi_3, \xi_4) = v_A(0, 0, 0, 0) + \sum_{i=1}^{M} \frac{\partial v_A}{\partial \xi_i} \xi_i + \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{M} \frac{\partial^2 v_A}{\partial \xi_i \partial \xi_j} \xi_i \xi_j$

In[143]:=
```
ξKL = {ξ1, ξ2, ξ3, ξ4};
```

In[144]:=
```
vA = SMTPostData["v", Point[{L, H/2}]] +
  Sum[ SMTPostData[{"st", 2, ToString[p]}, Point[{L, H/2}]] p, {p, ξKL}] +
  1/2 Sum[ SMTPostData[{"st", 2, {ToString[pi], ToString[pj]}}, Point[{L, H/2}]] pi pj,
    {pi, ξKL}, {pj, ξKL}]
```

$$-2.9574 + 0.203169\,\xi1 - 0.0790047\,\xi2 - 0.00297129\,\xi3 - 0.0100366\,\xi4 + \frac{1}{2}\left(-0.0282671\,\xi1^2 + 0.0201738\,\xi1\,\xi2 - 0.0085641\,\xi2^2 + 0.000985807\,\xi1\,\xi3 - 0.000351651\,\xi2\,\xi3 - 0.00286044\,\xi3^2 + 0.000986487\,\xi1\,\xi4 - 0.00492093\,\xi2\,\xi4 - 0.0000163255\,\xi3\,\xi4 - 0.00245471\,\xi4^2\right)$$

- Expected value of v_A

In[145]:=
```
EvA = NExpectation[vA,
  {ξ1 ≈ NormalDistribution[0, 1], ξ2 ≈ NormalDistribution[0, 1], ξ3 ≈ NormalDistribution[0, 1],
   ξ4 ≈ NormalDistribution[0, 1]}, PrecisionGoal → 3, Method → "MonteCarlo"]
```

```
-2.97864
```

- Variance of v_A

In[146]:=
```
VarvA = NExpectation[(vA - EvA)^2,
  {ξ1 ≈ NormalDistribution[0, 1], ξ2 ≈ NormalDistribution[0, 1], ξ3 ≈ NormalDistribution[0, 1],
   ξ4 ≈ NormalDistribution[0, 1]}, PrecisionGoal → 3, Method → "MonteCarlo"]
```

```
0.0480075
```

- Standard deviation of v_A

In[147]:=
```
σvA = √VarvA
```

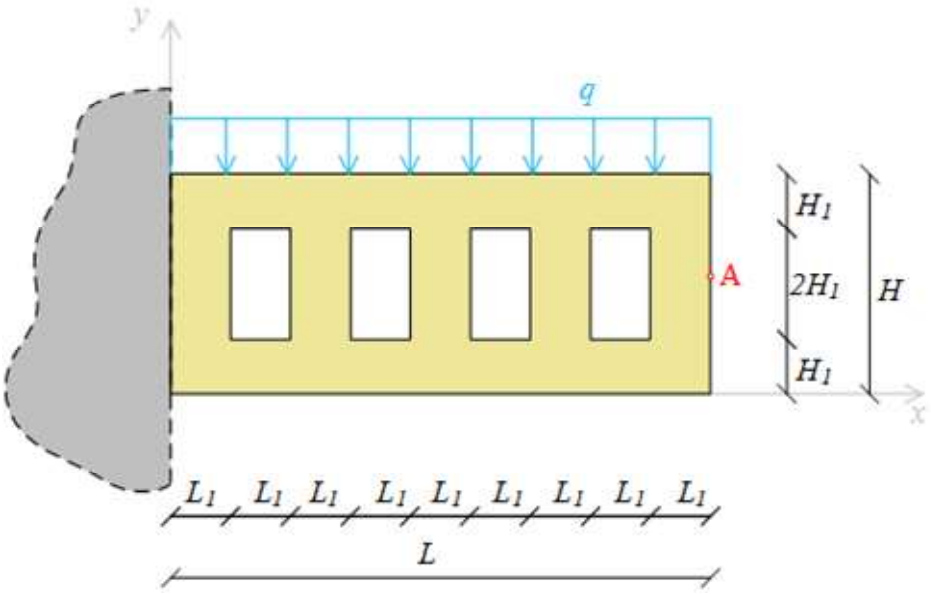
```
0.219106
```

Stochastic analysis with Monte-Carlo method

In[148]:=
```
<< AceFEM`
```

Cantilever beam with material parameters as stochastic variables

A cantilever beam with holes is exposed to vertical load. In this example, the randomness of Young's modulus is considered. Young's modulus is modelled as stochastic variable with mean Em0 and standard deviation σEm. Since the material is hyper-elastic, the problem is time independent.



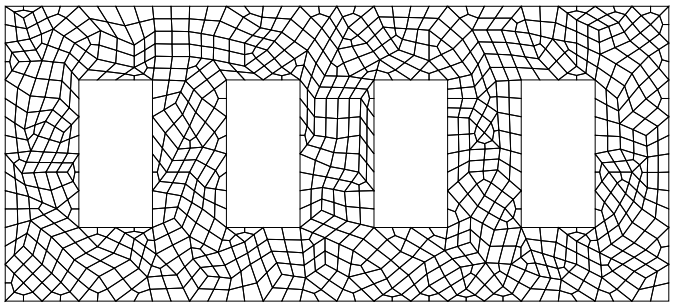
Input parameters

In[149]:=
```
L = 90;
H = 40;
q = 1;
Em0 = 1000;
σEm = 100;
```

Response function

In[154]:=
```
mesh = SMTTriangularToQuad[
    ToElementMesh[Polygon[{{{0, 0}, {90, 0}, {90, 20}, {80, 20}, {80, 10}, {70, 10},
        {70, 20}, {60, 20}, {60, 10}, {50, 10}, {50, 20}, {40, 20}, {40, 10},
        {30, 10}, {30, 20}, {20, 20}, {20, 10}, {10, 10}, {10, 20}, {0, 20}},
       {{0, 40}, {90, 40}, {90, 20}, {80, 20}, {80, 30}, {70, 30}, {70, 20}, {60, 20},
        {60, 30}, {50, 30}, {50, 20}, {40, 20}, {40, 30}, {30, 30}, {30, 20}, {20, 20},
        {20, 30}, {10, 30}, {10, 20}, {0, 20}}}], "MaxCellMeasure" → 10, "MeshOrder" → 1]];
```

In[155]:=
```
mesh["Wireframe"]
```



- MCEvaluation={number of MC evaluations, total time}

In[156]:=
```
response[par_?NumberQ] := (
  ++MCEvaluation[[1]];
  MCEvaluation[[2]] += AbsoluteTiming[
     SMTInputData[];
     SMTAddDomain[{"CantileverBeam", "ExamplesStochaticSEPEQ1DFHYQ1DNeoHookeWA",
       {"E *" → par, "ν *" → 0.3, "t *" → 1}}];
     SMTAddMesh[mesh, "CantileverBeam"];
     SMTAddEssentialBoundary[{Line[{{0, 0}, {0, H}}], 1 → 0, 2 → 0}];
     SMTAddNaturalBoundary[{Line[{{0, H}, {L, H}}], 2 → Line[{-q}]}];
     SMTAnalysis[];
     SMTNextStep["λ" → 1];
     While[SMTConvergence[10^-12, 10], SMTNewtonIteration[]];
    ][[1]];
  If[Mod[MCEvaluation[[1]], maxMCEvaluations / 10 // Floor] == 0,
   Print["MC simulation :", MCEvaluation];];
  If[MCEvaluation[[1]] > maxMCEvaluations,
   Print["MCEvaluation > maxMCEvaluations : ", MCEvaluation];
   Abort[]];
  SMTPostData["v", Point[{L, H / 2}]]
 )
```

Statistics of response

- RandomVariate gives a list of *maxMCEvaluations* pseudorandom variates from the symbolic distribution. Note that Monte-Carlo method is **very** time consuming method. Typically more than 10^4 evaluations are needed for accurate results. Number 500 is far to low!

In[157]:=
```
MCEvaluation = {0, 0};
maxMCEvaluations = 500;
data = RandomVariate[NormalDistribution[Em0, σEm], maxMCEvaluations];
```

- Here pseudorandom variates are used to perform MC simulations.

In[160]:=
```
MCSamples = Table[response[par], {par, data}];
```

```
MC simulation :{50, 11.5059}
MC simulation :{100, 23.4314}
MC simulation :{150, 36.0474}
MC simulation :{200, 48.0541}
MC simulation :{250, 60.5337}
MC simulation :{300, 73.1591}
MC simulation :{350, 85.0692}
MC simulation :{400, 97.0276}
MC simulation :{450, 108.834}
MC simulation :{500, 120.793}
```

- Calculate expected value of v_A using Mote Carlo method.

In[161]:=
```
EvA = Mean[MCSamples]
```

```
-3.12509
```

- Calculate Variance of v_A using Mote Carlo method.

In[162]:=
```
VarvA = Variance[MCSamples]
```

```
0.108511
```

- Standard deviation of v_A

In[163]:=
```
σvA = √VarvA
```
```
0.32941
```

Environment for stochastic analysis of general time dependent problems

In[164]:=
```
<< AceFEM`
SMTStochasticEnvironmentLoaded = True;
```

Documentation for computational environment

SMTMeshToKLMeshTransformation[]
command transforms original mesh composed of triangles or quadrilaterals into mesh of elements appropriate for Karhunen-Loève expansion. The stochastic element nodes are composed of nodes of leading sub-element (D_k) and nodes of all its associated sub-elements, thus elementNodes=nodes (D_k) ∪ nodes (D_l):$l \geq k \wedge ||\mathbf{t}_k - \mathbf{t}_l|| < l_{eff}$. Command must be called after the original mesh is defined and before the SMTAnalysis command.

SMTKLDecomposition[*M*]
Based on the generated mesh, the SMTKLDecomposition evaluates matrix N, matrix C, solves the corresponding eigenvalue problem and returns a list of eigenvalues and eigenvectors. *M* is the number of terms of Karhunen - Loève expansion.

SMTProjectEigenfunctions[λ,fk]
command performs least square projection of $\sqrt{\lambda_i}\, f_i(x)$ fields from KL mesh to response mesh.

SMTMeshToKLMeshTransformation

In[166]:=

In[167]:=
```
SMTMeshToKLMeshTransformation[] := Module[
  {d, dll, elinfo, ce, tnodes, setdata, tn, allsplit, splite, en, X, meshElements, massCenter,
   nearestElements, stochasticFE, trueMaxNodes, e, maxNodes, p1, celm, cdom, domi, lceff},
  If[SMTDomains === {}
   , SMCError = {};
   SMCAbort["Domains must be defined with SMTAddDomain command.",
    "SMTMeshToKLMeshTransformation", "Stochastic analysis"];
   , elinfo = Map[Check[
       p1 = Cases[SMTDomains[[1, 3]], Rule[_?(StringMatchQ[#, "lceff*"] &), i_] :> i];
       If[p1 === {}
        , SMCError = {"Domain: ", #};
        SMCAbort["lceff is an obligatory input data!",
         "SMTMeshToKLMeshTransformation", "Input Data Phase"];
       ];
       Join[SMTElementReport[#[[2]]], {"lceff" → p1[[1]]}]
       , SMCError = {"Domain: ", #};
       SMCAbort["Error in element.", "SMTMeshToKLMeshTransformation", "Input Data Phase"];
      ] &, SMTDomains]
  ];
  (*meshElements is a list of node indices of each sub-element Dk.*)
  X = SMTNodes[[All, 2 ;;]];
  allsplit = {};
  SMTNoElements = 0;
  SMTElements = Flatten[Map[(
       celm = #; cdom = celm[[1, 2]];
       domi = FirstPosition[SMTDomains[[All, 1]], cdom];
       If[domi === {}
        , SMCError = {"Domain: ", cdom};
```

```
        SMCAbort["Unknown element domain.",
         "SMTMeshToKLMeshTransformation", "Input Data Phase"];
        , domi = domi[[1]];
       ];
       (*massCenter is a list of mass centers (tk) of all Dk.*)
       massCenter = Table[RegionCentroid[Polygon[X[[e[[3]]]]]], {e, celm}];
       (* For each Dk find  all D1: ||tk-t1||<leff within the same domain*)
       lceff = "lceff" /. elinfo[[domi]];
       nearestElements =
        Map[celm[[#, 1]] &, Nearest[massCenter → Automatic, massCenter, {Infinity, lceff}]];
       (* Delete all D1:l<k .*)
       nearestElements = Table[DeleteCases[e, _?(# < e[[1]] &)], {e, nearestElements}];
       (*Get true nodes of all stochastic elements*)
       stochasticFE = Table[Flatten[SMTElements[[e, 3]]], {e, nearestElements}];
       (*True maximum number of nodes should not exceed the maximum number of nodes
        set in element.*)trueMaxNodes = Max[Table[Length[e], {e, stochasticFE}]];
       (*if trueMaxNodes>MaxNodes split elements to several elements
          and set ed$$["Data",1]=1 to prevent double integration*)
       maxNodes = "NoNodes" /. elinfo[[domi]];
       tnodes = "NoTopologyNodes" /. elinfo[[domi]];
       If [trueMaxNodes > maxNodes
        , Flatten[
         Map[(ce = #;
            If[Length[ce] > maxNodes
             , tn = Take[ce, tnodes];
             splite = Map[Join[tn, #] &, Partition[Drop[ce, tnodes], UpTo[maxNodes - tnodes]]];
             allsplit = {allsplit, Range[SMTNoElements + 2, SMTNoElements + Length[splite]]};
             Map[{++SMTNoElements, cdom, #} &, splite]
             , {{++SMTNoElements, cdom, ce}}
            ]) &
           , stochasticFE]
         , 1]
        , Map[{++SMTNoElements, cdom, #} &, stochasticFE]
       ]
      ) &, Gather[SMTElements, #1[[2]] === #2[[2]] &]], 1];
  SMTKLDecompositionData = {Flatten[allsplit]};
 ]
```

SMTKLDecomposition

In[168]:=
```
SMTKLDecomposition[M_] := Module[{Nij, Cij, λ, fDOF, fNode, Ifk, fk},
  (*ste switch to splitted elements*)
  SMTElementData[SMTKLDecompositionData[[1]], "Data", {1}];
  (*Execution of Task 1 : calculation of matrix N:*)
  Nij = SMTTask["matrix_N"];
  (*Execution of Task 2 : calculation of matrix C:*)
  Cij = SMTTask["matrix_C"];
  (*Solution of generalized eigenvalue problem C f = Λ N f:*)
  {λ, fDOF} = Eigensystem[{Cij, Nij}, M];
  (* Map solution vector into nodes. The SMTNodeData["DOF"] returns a map
   between nodal DOF indices and nodal indices. DOF indices start with 0.*)
  fNode = Table[Extract[fDOF[[i]], SMTNodeData["DOF"] + 1], {i, 1, M}];
  (*Execution of Task 3 : calculation of integrals ∫D fk(x) fk(x)dx, k=1,...,M.*)
  Ifk = Table[
    SMTNodeData["at", Partition[fNode[[i]], 1]];
    SMTTask["integral_Ifk"]
    , {i, 1, M}];
  (*Normalization of discretized eigenfunctions:*)
  fk = fNode / Sqrt[Ifk];
  (*KL session is stored for the later use in SMTProjectEigenfunctions*)
  SMTDump["KLSession"];
  {λ, fk}
 ]
```

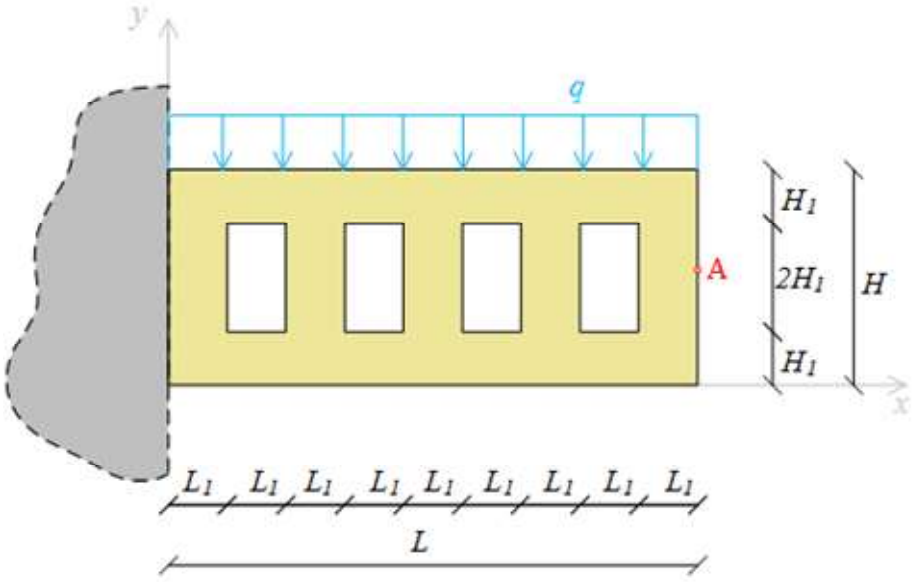
SMTProjectEigenfunctions

In[169]:=
```
SMTProjectEigenfunctions[λ_, fk_] := Module[{X, proj},
  X = SMTNodes[[1 ;; SMTNoNodes, 2 ;; -2]];
  SMTDump["ResponseSession"];
  SMTRestart["KLSession"];
  (* eigenfunctions can be easily evaluated for the new mesh simply by using
   SMTPostDatacommand.SMTPostDatacommand is using least square patch recovery
   procedure in order to get accurate transition from KL mesh to response mesh.*)
  proj = Table[SMTPostData[fk[[f]] * Sqrt[λ[[f]]], Point[X]], {f, 1, λ // Length}];
  SMTRestart["ResponseSession"];
  proj
 ]
```

Stochastic analysis of general time dependent problem

Cantilever beam with yield strength as stochastic field

In this example, the randomness of yield strength is considered and is modelled with stochastic field via Karhunen - Loève expansion. The values of yield strength along the domain are less correlated than in previous example, therefore more terms of Karhunen - Loève expansion are retained to be able to describe higher variability of stochastic field along the domain. Since the material is elasto-plastic, the problem is time-dependent.



In[170]:=
```
If[SMTStochasticEnvironmentLoaded =!= True,
  MessageDialog["Load stochastic computational environment first!"];
  Quit[]];
```

Input parameters

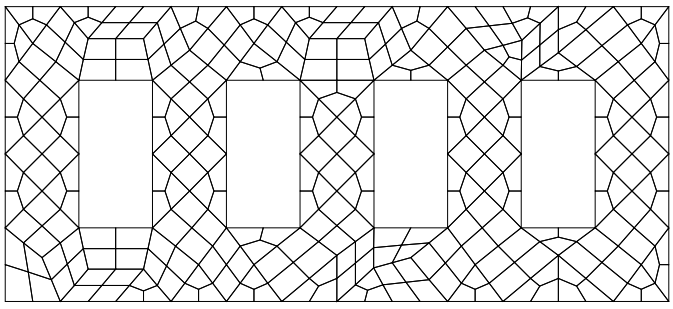
In[171]:=
```
L = 90;
H = 40;
q = 4;
Em = 21000;
σy0 = 44.8;
σσy = σKL = 2;
lc = 0.1 L;
lceff = 4 lc;
M = 9;
```

In[180]:=
```
polygons = {{{0, 0}, {90, 0}, {90, 20}, {80, 20}, {80, 10}, {70, 10}, {70, 20}, {60, 20},
    {60, 10}, {50, 10}, {50, 20}, {40, 20}, {40, 10}, {30, 10}, {30, 20}, {20, 20},
    {20, 10}, {10, 10}, {10, 20}, {0, 20}}, {{0, 40}, {90, 40}, {90, 20}, {80, 20},
    {80, 30}, {70, 30}, {70, 20}, {60, 20}, {60, 30}, {50, 30}, {50, 20}, {40, 20},
    {40, 30}, {30, 30}, {30, 20}, {20, 20}, {20, 30}, {10, 30}, {10, 20}, {0, 20}}};
```

Discretization of stochastic field

- Discretization for calculation of Karhunen-Loève expansion

In[181]:=
```
mesh = SMTTriangularToQuad[
    ToElementMesh[Polygon[polygons], "MaxCellMeasure" → 30, "MeshOrder" → 1]];
mesh["Wireframe"]
```



In[183]:=
```
SMTInputData[];
SMTAddDomain["KLBeam", "ExamplesStochasticFE", {"σ *" → σKL, "lc *" → lc, "lceff *" -> lceff}];
SMTAddMesh[mesh, "KLBeam"];
```

- Here SMTTransformToKLMesh transforms original mesh into KL mesh. Command must be called before the SMTAnalysis command.

In[186]:=
```
SMTMeshToKLMeshTransformation[];
SMTAnalysis[];
```

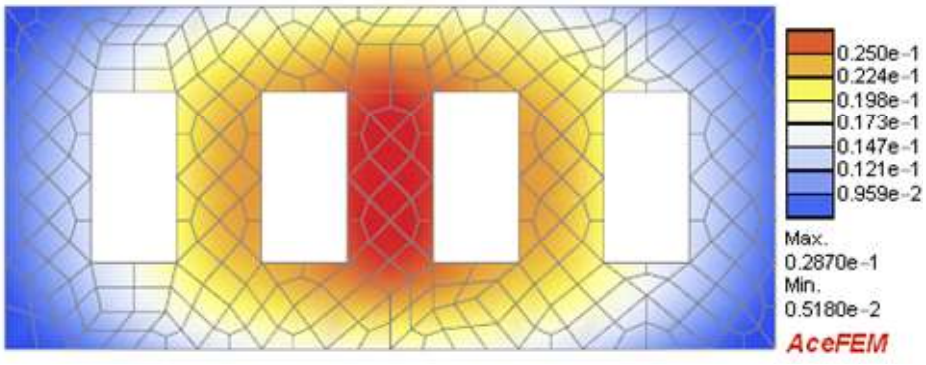
- Evaluation of Karhunen-Loève expansion.

In[188]:=
```
{λ, fk} = SMTKLDecomposition[M];
```

Visualize selected Karhunen-Loève eigenfunctions

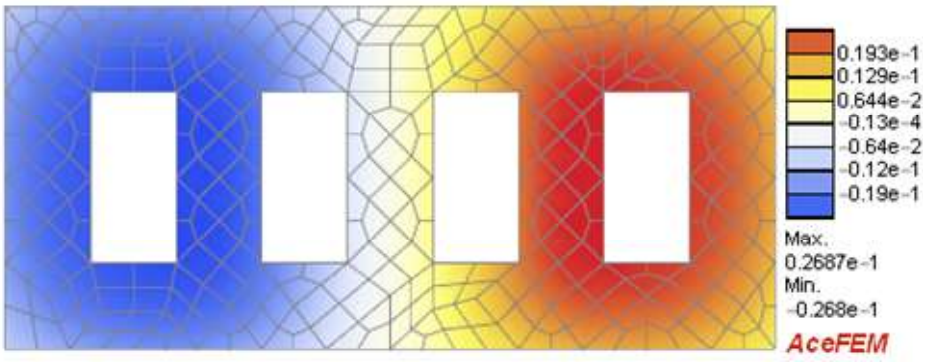
- Eigenfunction f_1:

In[189]:=
```
SMTShowMesh["Field" → fk[[1]], ImageSize → 400]
```



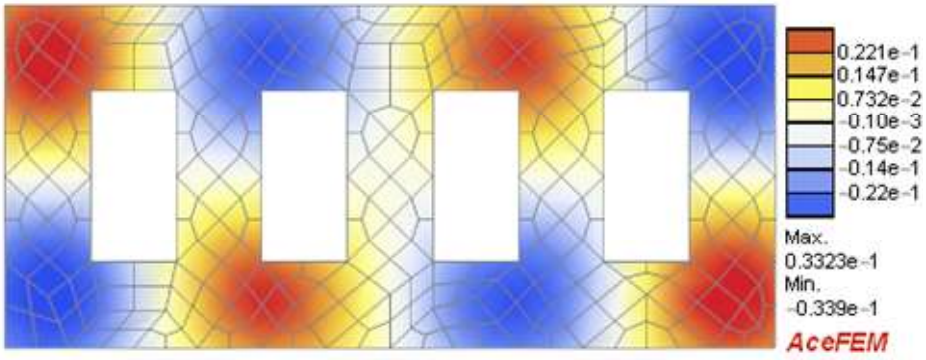
- Eigenfunction f_2:

In[190]:=
```
SMTShowMesh["Field" → fk[[2]], ImageSize → 400]
```



- Eigenfunction f_M:

In[191]:=
```
SMTShowMesh["Field" → fk[[M]], ImageSize → 400]
```



Evaluation of response

To evaluate Taylor series of response around expected values of random variables, derivatives of solution vector with respect to random variables have to be calculated. The derivatives are obtained with sensitivity analysis.

To define velocity fields, equation (1) has to be differentiated with respect to random variables ξ_i:

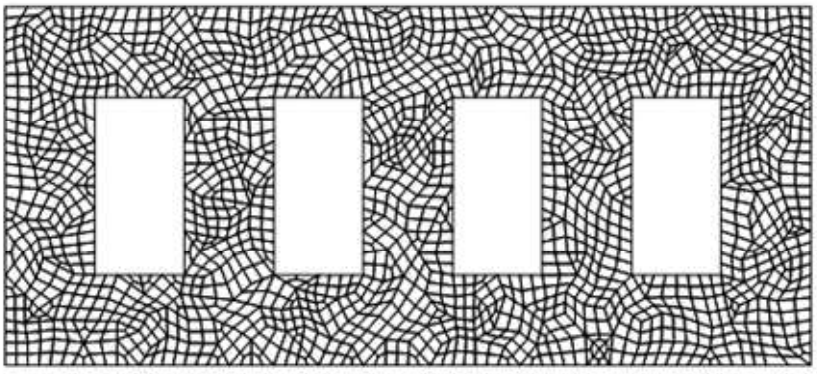
$$\frac{\partial \mathrm{Em}(x,\theta)}{\partial \xi_i} = \sqrt{\lambda_i}\, f_i(x)$$

$$\frac{\partial^2 \mathrm{Em}(x,\theta)}{\partial \xi_i \partial \xi_j} = 0$$

Thus, weighted eigenfunctions of the KL decomposition are exactly the first order velocity fields of the sensitivity problem.

Mesh for the evaluation of response

In[192]:=
```
mesh =
  SMTTriangularToQuad[ToElementMesh[Polygon[polygons], "MaxCellMeasure" → 5, "MeshOrder" → 1]];
mesh["Wireframe"]
```



Primal and second order sensitivity analysis

In[194]:=
```
SMTInputData[];
SMTAddDomain[
  {"beam", "ExamplesStochaticSEPEQ1DFJCQ1DNeoHookeWAMisesExH",
   {"E *" → Em, "σy *" → σy0, "ν *" -> 0.3, "t *" → 1}}
 ];
SMTAddMesh[mesh, "beam"];
SMTAddEssentialBoundary[{Line[{{0, 0}, {0, H}}], 1 → 0, 2 → 0}];
SMTAddNaturalBoundary[{Line[{{0, H}, {L, H}}], 2 → Line[{-q}]}];
```

- Definition of second order parameter sensitivity analysis. Parameter is yield stress that appears as 9-th parameter in a list of all sensitivity parameters. This is element specific and it should be checked for every element used! Uncorrelated random variables $\boldsymbol{\xi}$ represent sensitivity parameters.

In[199]:=
```
ξKL = Table[ToExpression["ξ" <> ToString[i]], {i, M}]
SMTSensitivityProblem[Table[
   {ToString[ξKL[[i]]], {"P", "∂σy/∂" <> ToString[ξKL[[i]]], All → 9}}, {i, M}], "Order" → 2];
```

Out[199]=
```
{ξ1, ξ2, ξ3, ξ4, ξ5, ξ6, ξ7, ξ8, ξ9}
```

In[201]:=
```
SMTAnalysis[];
```

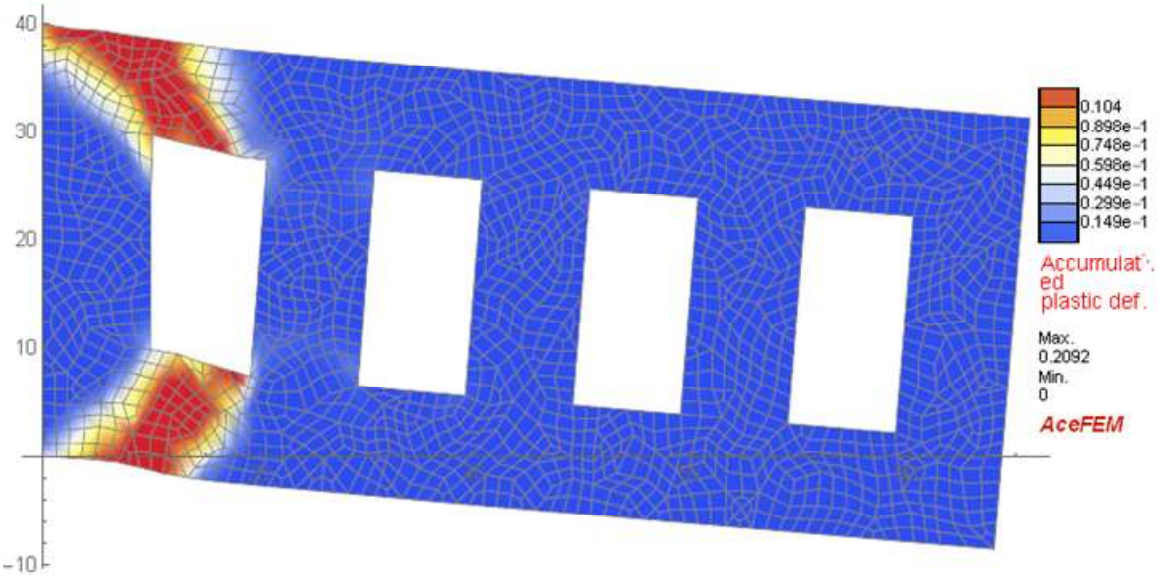
- Definition of velocity fields. They are obtained by least square projection of $\sqrt{\lambda_i}\, f_i(x)$ fields from KL mesh to response mesh.

In[202]:=
```
velocityFields = SMTProjectEigenfunctions[λ, fk];
SMTSetVelocityFields[
  Table["∂σy/∂" <> ToString[ξKL[[i]]] → {All, velocityFields[[i]]}, {i, M}]];
```

- Solution of primal and sensitivity problem.

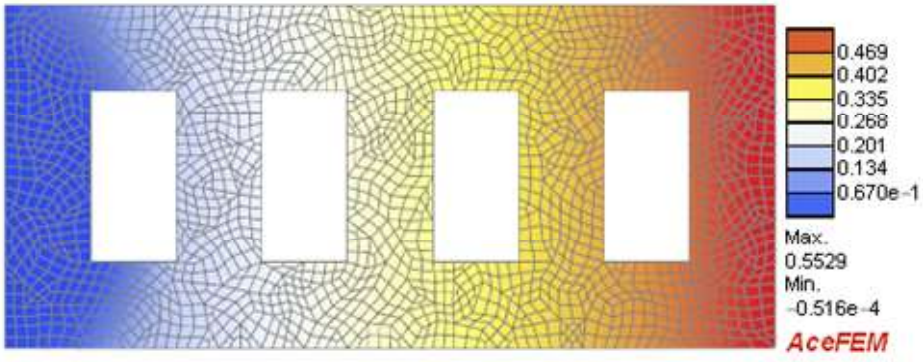
In[204]:=
```
λMax = 1; λ0 = λMax / 100; ΔλMin = λMax / 1000; ΔλMax = λMax / 10;
tolNR = 10^-10; maxNR = 15; targetNR = 8;
SMTNextStep["λ" → λ0];
While[
  While[
   step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λMax}]
   , SMTNewtonIteration[];
  ];
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; SMTStepBack[];];
  step[[3]]
  , If[Not[step[[1]]]
   , SMTNewtonIteration[];
   SMTForwardSensitivity[];
   , SMTStepBack[];];
  SMTNextStep["Δλ" → step[[2]]]
 ];
```

In[208]:=
```
SMTShowMesh["DeformedMesh" → True, Axes → True, "Field" → "Accumulated plastic def."]
```



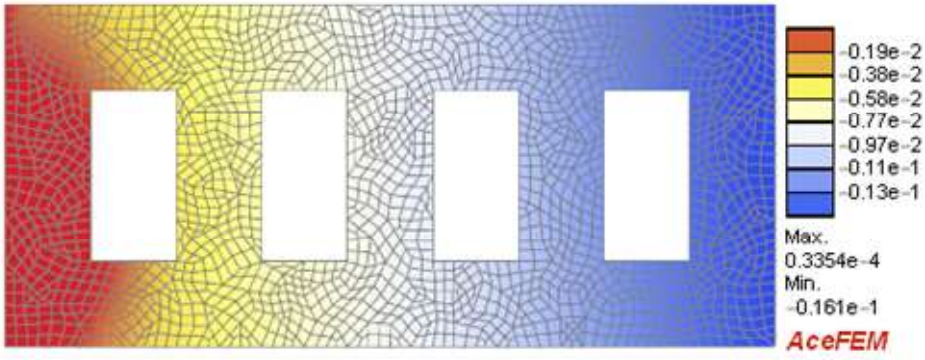
- Graphical representation of selected first- and second-order sensitivities:
  - Sensitivity of vertical displacement with respect to ξ1 ($\frac{\partial v}{\partial \xi_1}$):

In[209]:=
```
SMTShowMesh["Field" → SMTPostData[{"st", 2, ξKL[[1]]}], ImageSize → 400]
```



- Second-order sensitivity of vertical displacement with respect to ξ_1 and ξ_M ($\frac{\partial^2 v}{\partial \xi_2 \partial \xi_M}$):

In[210]:=
```
SMTShowMesh["Field" → SMTPostData[{"st", 2, {ξKL[[1]], ξKL[[M]]}}], ImageSize → 400]
```



Statistics of response

- Taylor expansion of second order

  Response of interest is vertical displacement v_A in central point at the free end of cantilever beam (point {L, H/2}) and the statistics: mean, variance and standard deviation.

  Taylor expansion of second order of the response of vertical displacement v_A is:

  $v_A(\boldsymbol{\xi}) = v_A(\mathbf{0}) + \sum_{i=1}^{M} \frac{\partial v_A}{\partial \xi_i} \xi_i + \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{M} \frac{\partial^2 v_A}{\partial \xi_i \partial \xi_j} \xi_i \xi_j$

In[211]:=
```
vA = SMTPostData["v", Point[{L, H/2}]] +
  Sum[ SMTPostData[{"st", 2, p}, Point[{L, H/2}]] p, {p, ξKL}] +
  1/2 Sum[ SMTPostData[{"st", 2, {pi, pj}}, Point[{L, H/2}]] pi pj, {pi, ξKL}, {pj, ξKL}]
```

$$
\begin{aligned}
&-8.63663 + 0.548515\,\xi1 - 0.804253\,\xi2 - 0.112787\,\xi3 + 0.680724\,\xi4 + \\
&0.0178241\,\xi5 - 0.353438\,\xi6 - 0.0566747\,\xi7 + 0.0330162\,\xi8 + 0.0941019\,\xi9 + \\
&\frac{1}{2}\Big(-0.0477814\,\xi1^2 + 0.140603\,\xi1\,\xi2 - 0.103793\,\xi2^2 + 0.0234501\,\xi1\,\xi3 - 0.0351305\,\xi2\,\xi3 - \\
&\quad 0.00519778\,\xi3^2 - 0.118817\,\xi1\,\xi4 + 0.176455\,\xi2\,\xi4 + 0.03\,\xi3\,\xi4 - 0.0758389\,\xi4^2 - \\
&\quad 0.0112865\,\xi1\,\xi5 + 0.0172472\,\xi2\,\xi5 + 0.00933698\,\xi3\,\xi5 - 0.0146418\,\xi4\,\xi5 - \\
&\quad 0.00545314\,\xi5^2 + 0.0622558\,\xi1\,\xi6 - 0.0936804\,\xi2\,\xi6 - 0.0173594\,\xi3\,\xi6 + 0.0824459\,\xi4\,\xi6 + \\
&\quad 0.00963282\,\xi5\,\xi6 - 0.0237677\,\xi6^2 + 0.0225353\,\xi1\,\xi7 - 0.0339667\,\xi2\,\xi7 - \\
&\quad 0.0113151\,\xi3\,\xi7 + 0.0292322\,\xi4\,\xi7 + 0.0113328\,\xi5\,\xi7 - 0.0176395\,\xi6\,\xi7 - \\
&\quad 0.00696651\,\xi7^2 - 0.00620003\,\xi1\,\xi8 + 0.0105408\,\xi2\,\xi8 + 0.00309029\,\xi3\,\xi8 - \\
&\quad 0.0112761\,\xi4\,\xi8 - 0.00252927\,\xi5\,\xi8 + 0.00928514\,\xi6\,\xi8 + 0.0038293\,\xi7\,\xi8 - \\
&\quad 0.00230461\,\xi8^2 - 0.0318061\,\xi1\,\xi9 + 0.0476068\,\xi2\,\xi9 + 0.011081\,\xi3\,\xi9 - 0.0412187\,\xi4\,\xi9 - \\
&\quad 0.00897386\,\xi5\,\xi9 + 0.0238198\,\xi6\,\xi9 + 0.0137355\,\xi7\,\xi9 - 0.0046198\,\xi8\,\xi9 - 0.00822911\,\xi9^2\Big)
\end{aligned}
$$

- Expected value of v_A

In[212]:=
```
ξDistributions = Table[i ≈ NormalDistribution[0, 1], {i, ξKL}]
```

```
{ξ1 ≈ NormalDistribution[0, 1],
 ξ2 ≈ NormalDistribution[0, 1], ξ3 ≈ NormalDistribution[0, 1],
 ξ4 ≈ NormalDistribution[0, 1], ξ5 ≈ NormalDistribution[0, 1],
 ξ6 ≈ NormalDistribution[0, 1], ξ7 ≈ NormalDistribution[0, 1],
 ξ8 ≈ NormalDistribution[0, 1], ξ9 ≈ NormalDistribution[0, 1]}
```

In[213]:=
```
EvA = NExpectation[vA, ξDistributions, PrecisionGoal → 3, Method → "MonteCarlo"]
```

```
-8.77526
```

- Variance of v_A

In[214]:=
```
VarvA = NExpectation[(vA - EvA)^2, ξDistributions, PrecisionGoal → 3, Method → "MonteCarlo"]
```

```
1.59607
```

- Standard deviation of v_A

In[215]:=
```
σvA = √VarvA
```

```
1.26335
```

CHAPTER 7

Multi-scale Analysis

This is work in progress. The final implementation of multi - scale environment might be different!!

Contents

- Solve an arbitrary FE^2 micro problem
- Solve an arbitrary MIEL micro problem

- Generation of selected micro structure meshes
  - FE22DMicroMeshVoids – Generate RVE mesh for 2 D perforated continuum
  - MIEL2DMicroMeshVoids – Generate MIEL mesh for 2 D perforated continuum
  - FE23DMicroMeshVoids – Generate RVE mesh for 3 D perforated continuum
  - MIEL3DMicroMeshVoids – Generate MIEL mesh for 3 D perforated continuum
- Examples of FE^2 multi – scale modeling
  - FE^2 modeling of 2 D uniformly distributed micro – structure
  - FE^2 modeling of 2 D functionally graded material
  - FE^2 modeling of 3 D uniformly distributed micro – structure
- Examples of MIEL multi – scale modeling
  - MIEL modeling of 2 D uniformly distributed micro – structure
  - MIEL modeling of 3 D uniformly distributed micro – structure
- Mixed single – scale / FE^2 / MIEL Modeling
  - Mixed FE^2 / MIEL modeling of uniformly distributed micro – structure
- Test of single micro structure at main kernel
  - 2 D FE^2 micro structure
  - 3 D FE^2 micro structure – unstructured mesh
  - 3 D FE^2 micro structure – structured mesh
  - 2 D MIEL micro structure
  - 3 D MIEL micro structure
- Comparison of multi – scale simulation with single scale simulation
  - Comparison of 2 D multi – scale simulation with 2 D single scale simulation
  - Comparison of 3 D multi – scale simulation with 3 D single scale simulation

Introduction to Multi-scale Analysis

Multi-scale modeling and the algorithms on which the present package are based is in detail described in paper" Sensitivity analysis based multi-scale methods of coupled path-dependent problems" available at https://link.springer.com/article/10.1007/s00466-019-01762-8 .

Documentation for Multi-scale Computational Environment

Multi - scale computational environment is fully parallelized for multi - core processors.

Multi-scale computational environment functions

SMTMultiScaleSet[*options*]

command sets up multi-scale problem accordingly to given options. For each kernel a directory is generated with the name "Kernel_"<>index. All micro meshes are created and saved at specific directory. Possible options are given below.

option	default	description
"FE^2"	{}	{{*micro_mesh _module*, *micro_solve _module*}-> {a list of element indexes for which micro problems should be generated using the given module},…}
"MIEL"	{}	{{*micro_mesh _module, micro_solve _module*}-> {a list of element indexes for which substructure should be generated using the given module},…}
"SpecificMicroData"	{}	is a list of rules provided as a second argument to *micro_mesh _modules*, e.g. {"FE^2"->{E,*v*},"MIEL"->{mesh_density},…} data should be checked and properly interprated by the *micro_mesh _module*
"PartialRestart"	False	True ⇒ only the CDriver data base of the micro problem is refreshed False (default) ⇒ complete restart is needed when the micro problems have structurally different meshes (only the actual coordinates of the nodes can be different).
"PartialDump"	True	True (default) ⇒ only the CDriver data base of the micro problem is stored on disc while the problem input data remains unchanged False ⇒ complete dump is needed only when during the solution of the local problem the mesh has been changed as well and it should be used only when it is necessary
"TestSingleScale"	False	At the development step it is beneficial to test the micro-structure independently at the main kernel. With the option "TestSingleScale"→True the environment is initialized for the testing a single micro structure at main kernel as shown in examples section.
"Debug"	0	In debug mode system prints out additional information for the *n*- th micro problem. If n= -1 all macro problems are included (True≡-1, False≡0).
"CompressionLevel"	2	-1 ⇒ plain data (0-9) ⇒ zlib compression level of dump file 0 - no compression 9 - maximum compression
"MaxKernels"	Automatic	number of parallel kernels lunched on the current computer
"Threads"	1	number of threads used by each parallel micro problem (SMTInputData["Threads"->...]) **Note that "MaxKernels"×"Threads" must be less that the maximum number of threads of the computer.**
"Console"	False	Cdriver at each micro problem is opened in a separate window

Options of the SMTMultiScaleSet function.

SMTMicroSolve[dump]

command solves all micro problems that belong to specific kernel and refreshes the stored state of micro problems when dump is True.

SMTMicroRestart[p]

command restarts micro problem closest to point p.

SMTMicroEvaluate[expression]

command evaluates expression remotely using the last restarted micro problem.

SMTMicroEvaluate["Continue"]
command does the same as pressing Continue button at the last restarted micro problem.

SMTMultiScalePostData[code]
command does the same as SMTPostData[code] and returns required quantity for all nodes of macro mesh.

SMTToSymmetricOrUnsymmetric[*symmMacro,ndofMacro,microMatrixData*]
command returns upper triangular matrix (*symmMacro*=True) or full matrix (*symmMacro*=False) where *ndofMacro* is size of matrix and *microMatrixData* is data provided by micro element (*microMatrixData* can corresponds to symmetric or unsymmetric matrix!)

SMTMultiScaleSimulationReport[]
prints out report identifying the percentage of time spent in specific task during the analysis and returns a list of rules

SMTMultiScaleSimulationReport[*comments*]
prints out report with additional *comments* and returns a list of rules identifying the percentage of time spent in specific tasks during the analysis

SMTMultiScaleSimulationReport[False]
returns only a list of rules identifying the percentage of time spent in specific tasks during the analysis

SMTMicroSet[]
command iInitializes and prepares all micro problems that belong to specific kernel (defined by SMTMaterialPointsForKernel) for restart operations.

Returns {local_problem_data, ... for all local problems at specific kernel}.

SMTSendLocalProblemsToKernels[]
command partitions the SMTAllLocalProblems into the number of kernels subsets and assigns each subset to be the value of symbol SMTLocalProblems on each kernel.

Multi-scale computational environment functions.

General flowchart

Computationally the multi - scale formulations represent locally coupled system of nonlinear equations solved implicitly by multi - level Newton - Raphson type of iterative method. Global equations or macro problem is formulated and solved at the main kernel and coupled systems or micro problems are formulated and solved at parallel kernels as independent finite element problems independently from the main kernel. Since the global problem and local problems are formulated within separate finite element formulations, the levels communicate by transferring the data. This can be done in the case of shared memory multi - core machine or in the case of Mathematica supported cluster of machines using MathLink protocol. In more general case the data between the nodes of the cluster can be passed using intermediate files.

The actual data transfer is not prescribed in advanced, but is defined by elements used to discretize macro and micro problem. That is done by the user defined tasks (see User Defined Tasks) that have to be supported by all macro and micro elements. The linking is a two stage process. Macro element has to have defined macro_micro_initialization task with the name "FE^2" or "MIEL" depending on the type of multi - scale formulation that returns macro_micro_initialization_data. Additionally the AceGen template constant SMSCharSwitch (see Template Constants) have to be defined in micro and macro elements that contains a series of character type user defined words or sentences.

Consistent linearization of the macro - micro coupled system of equations requires formulation of an algorithmic local tangent matrix.

Data structures

Notation

- n_m is the number of micro problems associated with the macro element
- n_g is the number of integration points of the macro element
  - $n_m = n_g$ in the case of FE^2 method

- n_m = 1 in the case of MIEL method

- $d_M = \{d_{M,1}, d_{M,2}, \ldots, d_{M,l_{dM}}\}$ is data returned by macro_data_task of the macro finite element
- l_{dM} is the length of macro data set d_M
- $d_m = \{d_{m,1}, d_{m,2}, \ldots, d_{m,l_{dm}}\}$ is data returned by micro_data_task of the micro finite element
- l_{dm} is the length of micro data set d_m

All data structures

- **macro_micro_initialization_data**
- **macro_data** returned by macro_data_task of the macro finite element
- **micro_data** returned by micro_data_task of the micro finite element
- **macro_micro_data** is composed of micro_data of all micro problems associated with specific macro problem
  - $d_{Mm} = \{d_{Mm,1}, d_{Mm,2}, \ldots, d_{Mm,l_{dMm}}\}$ where $l_{dMm} = n_m\, l_{dm}$
  - macro_micro_data is stored in a elements data field "Data" (see **Element Data**)
  - $l_{dMm} = n_g\, l_{dm}$ for FE^2 and $l_{dMm} = l_{dm}$ for MIEL
- **local_problem_data** is a general data for FE^2 formulations
- **local_problem_data** is a general data for MIEL formulations
- **specific_micro _data** is micro problem specific data related to the elements used, mesh, material, solution tolerances, etc.

macro_micro_initialization_data

The macro_micro _initialization _data task ("FE^2" or "MIEL") of macro element has to return the following list of integer numbers :

- 1 - number of micro problems associated with the element (n_m)
- 2 - index of character switch of macro element that defines the name of the task that returns multi-scale related macro data or macro_data_task
- 3 - length of multi-scale related macro data per micro problem (l_{dM})
- 4 - index of character switch of macro element that defines the name of the micro problem task that returns multi-scale related micro data or micro_data_task
- 5 - length of multi-scale related micro data per micro problem (n_m)

macro_data (d_M)

macro_data is returned by *macro_data _task* defined as a part of macro element formulation.

- FE^2
  The typical macro_data for FE^2 formulations is composed of the macro data for all integration points
  - $d_M = \text{Flatten}[\{d_{gM} : g = 1, 2, \ldots, n_g\}]$ where
  - $d_{gM} = \{d_{gM,1}, d_{gM,2}, \ldots, d_{gM,n_{dgM}}\}$ and $n_{dM} = n_g\, n_{dgM}$
  - $d_{gM} = \text{vec}(\epsilon_M)$ for small strain problems
  - $d_{gM} = \text{vec}(F_M)$ for finite strain problems
- MIEL
  The typical macro_data for MIEL formulations contains components of nodal degrees of freedom vec (u_M) of macro element.
  - $d_M = \text{vec}(u_M)$.

micro_data (d_m)

micro_data is returned by *micro_data _task* defined as a part of micro element formulation.

- FE^2

The typical micro_data for FE^2 formulations is composed of integral of flux rates of all physical fields ($a_m = \{a_{m_1}, a_{m_1}, \ldots, a_{m_{n_a}}\}$) togeher with their first order sensitivity with respect to the integration point macro_data d_{gM}.

- micro_data = Flatten[{a_M, K_M}] = $\int_{\Omega_e}$ Flatten[{a_m, K_m}] dV = $\int_{\Omega_e}$ Flatten$\left[\left\{a_m, \frac{\partial a_m}{\partial d_{gM_1}}, \frac{\partial a_m}{\partial d_{gM_2}}, \ldots, \frac{\partial a_m}{\partial d_{gM_n}}\right\}\right]$ dV
- For small strain problems a_m = vec (σ) and d_{gM} = vec (ϵ_M), thus micro_data_task task returns micro_data = $\int_{\Omega_e}$ Flatten$\left[\left\{\text{vec}(\sigma), \frac{\partial \text{vec}(\sigma)}{\partial \text{vec}(\epsilon_M)}\right\}\right]$ dV where vec (σ) are components of the macro stress tensor and vec (ϵ_M) components of macro small strain tensor accordingly to the kinematics of the problem. The term $\int_{\Omega_e} \frac{\partial \text{vec}(\sigma)}{\partial \text{vec}(\epsilon_M)}$ dV represents homogenized constitutive matrix for FE^2 multi-scale simulations.
- For finite strain problems a_m = vec (P) and d_{gM} = vec (F_M), thus *micro_data _task* task returns *micro_data* = $\int_{\Omega_e}$ Flatten$\left[\left\{\text{vec}(P), \frac{\partial \text{vec}(P)}{\partial \text{vec}(F_M)}\right\}\right]$ dV where vec (P) are components of the first Piola-Kirchoff stress tensor and vec (F_M) components of deformation gradient accordingly to the kinematics of the problem. The term $\int_{\Omega_e} \frac{\partial \text{vec}(P)}{\partial \text{vec}(F_M)}$ dV represents homogenized constitutive matrix for FE^2 multi-scale simulations.

- MIEL

  In the case of MIEL multi-scale formulation *micro_data_task* returns integral of strain energy over domain of the problem and sensitivity of strain energy with respect to the components of nodal degrees of freedom vec (u_M) of macro element, thus micro_data = $\int_{\Omega_e}$ Flatten$\left[\left\{W, \frac{\partial W}{\partial u_M} \mid_{\mathbf{h}=\text{const.}}, \text{vec}\left(\frac{\partial}{\partial u_M}\left(\frac{\partial W}{\partial u_M} \mid_{\mathbf{h}=\text{const.}}\right)\right)\right\}\right]$ dV. u_M are degrees of freedom used to parameterize prescribed essential boundary conditions of the macro problem. The term $\int_{\Omega_e} \frac{\partial W}{\partial u_M} \mid_{\mathbf{h}=\text{const.}}$ dV represents contribution of the micro element to macro residual and $\int_{\Omega_e} \text{vec}\left(\frac{\partial}{\partial u_M}\left(\frac{\partial W}{\partial u_M} \mid_{\mathbf{h}=\text{const.}}\right)\right)$ dV contribution of the micro element to condensed tangent matrix for MIEL multi-scale simulations. If second order directional derivatives vec$\left(\frac{\partial}{\partial u_M}\left(\frac{\partial W}{\partial u_M} \mid_{\mathbf{h}=\text{const.}}\right)\right)$ are symmetric then only upper triangular matrix is returned.

micro_macro_data

- *micro_data* appears in the macro element stored in a element "Data" data field for all macro problems associated with specific macro element
  - for FE^2 problems: micro_macro _data = Flatten[{micro_data$_1$, micro_data$_2$, … micro_data$_{n_g}$}] where n_g is the number of integration points
  - for MIEL problems: micro_macro _data = micro_data

local_problem_data for FE^2 formulation

local_problem _data data structure contains all the data needed for proper restart and solution of (FE^2) local problem.

- local_problem_data = {
  - 1 description is a list of 8 values:
  - {
  - 1 - "FE^2",
  - 2 - micro_mesh_module - the name of module that initializes the micro problem for given local_problem_data and returns updated local_problem_data
  - 3 - micro_solve_module - the name of module that solves micro problem
  - 4 - the number of micro problems associated with the corresponding macro element (e.g. the number of integration points)
  - 5 - macro_data_task - the name of the task that returns multi-scale related macro data
  - 6 - length of multi-scale related macro data
  - 7 - micro_data_task - the name of the micro problem task that returns multi-scale related micro data
  - 8 - length of multi-scale related micro data
  - }
  - 2 material point coordinates,

- 3 an arbitrary user defined data set at micro mesh generation phase (e.g. **specific_micro_data**) in *micro_mesh_module*
- 4 global index of local problem
- 5
- 6 corresponding macro element
- 7 data received from the corresponding macro element by executing macro_data_task per integration point (e.g. components of global strain measure from macro element integration point)
- 8 constrained_nodes - a list of indexes of corner nodes
- 9 RVE volume
- 10 solution procedure options {
- "MinSubSteps"->*n* ⇒ defines a minimum number of micro sub-steps per macro step (default 1)
- "MaxSubSteps"->*n* ⇒ defines a maximum number of micro sub-steps per macro step (default 100)
- }
- 11 macro sensitivity analysis data{
- number of macro design sensitivity parameters
- macro design velocity fields
- }
- }

local_problem_data for MIEL formulation

local_problem _data data structure contains all the data needed for proper restart and solution of MIEL local problem.

- local_problem_data = {
  - 1 description is a list of 8 values:
  - {
  - 1 - "MIEL"
  - 2 - micro_mesh_module - the name of module that initializes the micro problem for given local_problem_data and returns updated local_problem_data
  - 3 - micro_solve_module - the name of module that solves micro problem
  - 4 - 1 (the number of micro problems associated with the corresponding macro element)
  - 5 - macro_data_task - the name of the task that returns multi-scale related macro data
  - 6 - length of multi-scale related macro data
  - 7 - micro_data_task - the name of the micro problem task that returns multi-scale related micro data
  - 8 - length of multi-scale related micro data
  - }
  - 2 coordinates at the center of macro element (needed for restart)
  - 3 an arbitrary user defined data set at micro mesh generation phase (e.g. **specific_micro_data**) in *micro_mesh_module*
  - 4 global index of local problem
  - 5
  - 6 corresponding macro element
  - 7 data received from the corresponding macro element by executing macro_data_task (e.g. macro element nodal displacements - Flatten[u_M])
  - 8 macro element nodal coordinates - X_M

- 9 macro element topology
- 10 for all macro element nodes a set of coordinates of points for which result of SMTPost[code] is averaged to get nodal post-processing value (default $\{X_M\}^T$) ... important in the case where there are no corner nodes presents due to voids
- 11 minimum division of edge for accurate description of curved edges
- 12 solution procedure options {
- "MinSubSteps"->*n* ⇒ defines a minimum number of micro sub-steps per macro step (default 1)
- "MaxSubSteps"->*n* ⇒ defines a maximum number of micro sub-steps per macro step (default 100)
- }
- 13 True/False is corresponding macro element tangent matrix symmetric or not
- 14 macro sensitivity analysis data {
- number of macro design sensitivity parameters
- macro design velocity fields
- }
- } contains all the data needed for proper restart and solution of local problem

specific_micro_data

Data has no predefined form.

Variables

Shared variables - global at macro and micro level

- SMTSharedStatus={
  - 1 error type
  - 0 - no error
  - 1 - warnings were detected during the session, (evaluation is still performed in a regular way, time step cutting is recommended)
  - 2 - stop all micro problems and preform macro time step cutting
  - 3 - fatal error (terminate the process)
  - 2 additional data
  - integer counter of progress
  - error description if SMTSharedStatus[[1]]>0
  - }
- SMTMacroProblemStatus={
  - 1 spatial dimension of the problem,
  - 2 {"FE^2" , "MIEL"} possible types of multi scale analysis
  - 3 restart type
  - True ⇒ only the CDriver data base of the micro problem is refreshed from disc
  - False ⇒ complete restart is needed when the micro problems have structurally different mesh (only the actual coordinates of the nodes can be different).
  - 4 number of micro problems
  - 5 multi-scale palette notebook
  - 6 number of parallel kernels

- 7 dump type
- True ⇒ only the CDriver data base of the micro problem is stored on disc
- False ⇒ complete dump is needed only when during the solution of the local problem the mesh has been changed as well and it should be used only when it is necessary
- 8 start time
- 9 root directory of macro problem
- 10 {$KernelID 1,...} IDs of all parallel kernels
- 11 AdditionalMicroData (value of option to SMTMultiScaleSet function)
- 12 total micro solution time
- 13 total micro problems generation time
- 14 debug elements for which system prints out additional info -1⇒all, 0⇒none, i_Integer⇒ selected
- 15 zlib compression level of dump file (0-9) with 0-no compression 9-maximum compression
- 16 debugging function called at the end of micro solution *dbf[local_problem_data,True converged state, message]*
- 17 number of CDriver threads
- 18 CDriver console window True/False
- 19
- 20 index of current global NR iteration
- 21 SMTMicroSolve call index
- }

- SMTCurrentLocalProblem - current local problem restarted for post-processing

Global variables at macro level

- SMTMacroElements = {
  - 1 FE^2 {element index->{list of indices of local problems} , ... for all FE2 macro elements},
  - 2 MIEL {element index->{index of MIEL problem} , ... for all MIEL macro elements},
  - 3 {indices of single scale elements ... },
  - }
- SMTAllLocalProblems = {local_problem_data, ... for all micro problems}

Global variables at micro level (local to the parallel kernel)

- SMTLocalProblems = {local_problem_data, ... for all local problems at specific kernel}
- SMTMicroProfile ={
  - 1 restart time
  - 2 micro solution time
  - 3 dump time
  - 4 No of dumps
  - 5 No. of restarts
  - 6 No. of micro solutions (total)
  - 7 No. elements - set at SMTMicroSet
  - 8 No. nodes - set at SMTMicroSet
  - 9 No. equations - set at SMTMicroSet

- 10 counter of processed micro problems(set or solve)
- 11 Schur complement time
- 12 *macro_data_task* time
- 13 NoDiscreteEvents
- 14 max actual no. sub-steps
- 15 kernel index
- 16 no. sub-steps in last macro step (last calculated micro)
- 17 no. back-steps (last calculated micro)
- 18 no. sub-steps in last macro step (last calculated micro)
- 19 micro problem SMTSharedStatus
- 20 dump solution True/False
- }

Problem dependent user defined functions

- micro_mesh_module[local_problem_data, additional_micro_data]
  initializes the micro problem for given local_problem_data and additional_micro_data and returns updated local_problem_data.
- micro_solve_module [local_problem_data, dump (True or False)]
  solves selected micro problem and returns data micro_data depending on MS type

Finite elements for multi-scale analysis

All the elements from the "Main Library" AceShare library of solid and heat conduction finite elements are supporting the first and second order essential boundary conditions sensitivity analysis and thus can be used as the micro elements. An example of the element that supports multi - scale analysis is given in `Finite Strain Element for Direct and Sensitivity Analysis`.The element supports first and second order essential boundary conditions sensitivity analysis, thus it can be used for FE^2 and MIEL multi - scale simulations.For the discretization of the macro problem user has to built specialized finite elements.Some are given in `Examples of Specilized Elements for Multi – scale Analysis` section.

Finite elements for FE^2 analysis

FE^2 micro element

FE^2 micro element is any standard element that supports first order sensitivity analysis. Micro element must have user subroutine "Tasks" defined with the support for the following tasks:

- *micro_data_task* task returns *micro_data* data structure as described in section Data structures. The *micro_data_task* is identified as such by the corresponding character switch constant (see SMSCharSwitch). The same character switch constant is also used in the macro element. It represents the link between the micro and macro element.
- "Reset sensitivity data" task resets sensitivity problem by deleting all sensitivity related data.

FE^2 macro element

FE^2 macro element is specific finite element created for the FE^2 simulations. Macro element calculates macro tangent and residual. Macro element must have user subroutine "Tasks" defined with the support for the following tasks:

- "FE^2" task returns a list
  - {
  - 1 - number of micro problems associated with the element (this is in the case of FE^2 method typically the number of integration points of macro element)

- 2 - index of character switch constant of macro element that defines the name of the task *macro_data_task* that returns multi-scale related macro data
- 3 - length of multi-scale related macro data
- 4 - index of character switch constant of macro element that defines the name of the micro problem task that returns multi-scale related micro data or *micro_data_task*
- 5 - length of multi-scale related micro data
- }

- "Material points" task returns position vector of all integration points Join$[X_1, X_2, \ldots, X_{ng}]$
- *macro_data_task* task returns *macro_data* as described in section Data structures

element that imposes periodic boundary conditions

Element must support at least first order sensitivity analysis.

Finite elements for MIEL analysis

MIEL micro element

MIEL macro element is any standard element that supports first and second order sensitivity analysis. Micro element must have user subroutine "Tasks" defined with the support for the following tasks :

- *micro_data_task* task returns *micro_data* as described in section Data structures. The *micro_data_task* is identified as such by the corresponding character switch constant (see SMSCharSwitch). The same character switch constant is also used in the macro element. It represents the link between the micro and macro element.
- "Reset sensitivity data" task resets sensitivity problem by deleting all sensitivity related data.

MIEL macro element

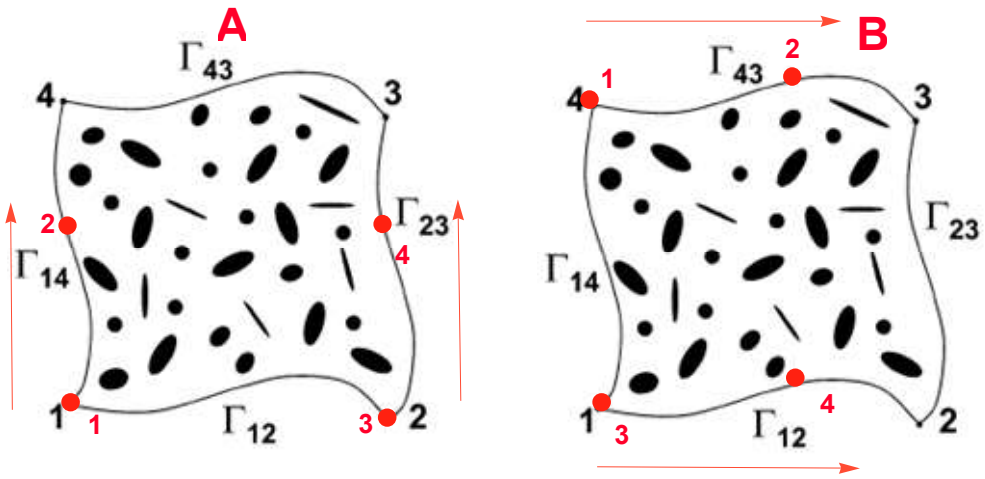
MIEL macro element is specific finite element created for the MIEL simulations. Macro element calculates macro tangent and residual. Macro element must have user subroutine "Tasks" defined with the support for the following tasks :

- "MIEL" task returns a list
  - {
  - 1 - number of micro problems associated with the element which is in the case of MIEL formulation 1
  - 2 - index of character switch constant of macro element that defines the name of the task *macro_data_task* that returns multi-scale related macro data
  - 3 - length of multi-scale related macro data
  - 4 -index of character switch constant of macro element that defines the name of the micro problem task that returns multi-scale related micro data or *micro_data_task*
  - 5 - length of multi-scale related micro data
  - }
- *macro_data_task* task returns *macro_data* as described in section Data structures

Examples of Specialized Elements for Multi-scale Analysis

Periodic boundary constraint element for 2D solid analysis - structured mesh

Description



- Periodicity condition: $u_4 - u_2 = u_3 - u_1$

AceGen input for periodic BC element

In[98]:=
```
<< AceGen`;
SMSInitialize["ExamplesPeriodic2DStructured", "Environment" → "AceFEM"];
SMSTemplate[
  (*element is point in 2D*)
  "SMSTopology" → "V2",
  "SMSDefaultIntegrationCode" → 0,
  "SMSNoNodes" → 5,
  "SMSDOFGlobal" → {2, 2, 2, 2, 2},
  (*no default postprocessing*)
  "SMSSegments" → {{2}, {4}},
  "SMSSegmentsTriangulation" → {{}, {}},
  (*define additional local auxiliary node that holds Lagrange multipliers*)
  "SMSNodeID" → {"D", "D", "D", "D", "PerConst -LP -L"},
  "SMSAdditionalNodes" → Function[{}, {Null}],
  "SMSSymmetricTangent" → True,
  "SMSDomainDataNames" -> {"ρ -augmented Lagrange penalty parameter"},
  "SMSDefaultData" → {10},
  "SMSEBCSensitivity" -> True
 ];
```

In[101]:=
```
ElementDefinitions[] := (
  uIO ⊨ SMSIO["All DOFs"];
  {ρ} ⊨ SMSIO["All domain data"];
  {λx, λy} ⊨ uIO[[5]];
  {gx, gy} ⊨ {(uIO[[3, 1]] - uIO[[1, 1]]) - (uIO[[4, 1]] - uIO[[2, 1]]),
    (uIO[[3, 2]] - uIO[[1, 2]]) - (uIO[[4, 2]] - uIO[[2, 2]])};
  Π = (λx gx + ρ / 2 gx^2) + (λy gy + ρ / 2 gy^2);
  pe = Flatten[uIO];
 )
```

In[102]:=
```
SMSStandardModule["Tangent and residual"];
ElementDefinitions[];
Rel ⊨ SMSD[Π, pe];
SMSIO[Rel, "Export to", "Residual"];
Ke ⊨ SMSD[Rel, pe];
SMSIO[Ke, "Export to", "Tangent"];
```

In[108]:=
```
SMSStandardModule["Sensitivity pseudo-load"];
SMSDo[is, SMSIO["SensIndexStart"], SMSIO["SensIndexEnd"]];
ϕis ⊨ SMSFictive[];
ElementDefinitions[];
"∂pe/∂ϕi here is first order essential boundary conditions velocity field";
SMSDefineDerivative[SMSIO["Nodal DOFs"], ϕis, SMSIO["Sensitivity DOFs"[is]]];
Rel ⊨ SMSD[Π, pe];
SMSIO[SMSD[Rel, ϕis], "Add to", "First order pseudo load"[is]];
SMSEndDo[];(*SensIndex*)
```

In[117]:=
```
SMSWrite[];
SMTMakeDll[];
```

```
File: ExamplesPeriodic2DStructured.c  Size: 6164  Time: 2
```

Method	SKR	SSE
No.Formulae	8	12
No.Leafs	637	329

Macro element for 2D solid FE^2 analysis - Q1 - σ/ϵ stress/strain pair - symmetric tangent

Description

Generate two - dimensional, four node macro finite element for the analysis of the micro - macro problems. The element has the following characteristics :

⇒ quadrilateral topology,

⇒ 4 node element,

⇒ isoparametric mapping from the reference to the actual frame,

⇒ global unknowns are displacements of the nodes,

⇒ the element should allow arbitrary large displacements and rotations,

⇒ the problem is defined by symmetric small strain and stress tensors and pseudo potential

$$\Pi = \int_{\Omega_0} \boldsymbol{\sigma} : \boldsymbol{\epsilon} \, \mathrm{d}\Omega_0$$

Ω_0 is the initial domain of the problem.

⇒ ADB form of the problem leads to the element contribution to the governing equations of the problem in a form :

$$\mathbf{R}_e = \frac{\hat{\delta}\,\Pi}{\hat{\delta}\,\mathbf{p}_e} \Big|_{\sigma=\text{const.}}$$

AceGen input for macro element

Initialization

In[119]:=
```
<< "AceGen`";
NoStressComponents = 4; NoKinComponents = 3;
lgd = (NoStressComponents + NoStressComponents NoKinComponents);
Led = lgd es$$["id", "NoIntPoints"];
SMSInitialize["ExamplesQ1seMacroSymm", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "Q1"
  , "SMSSymmetricTangent" → True
  , "SMSDomainDataNames" -> {"t -thickness"}, "SMSDefaultData" -> {1}
  , "SMSNoElementData" → Led
  , "SMSCharSwitch" → {"FE^2", "Material points", "Small strain tensor (plane strain)",
    "Integrated stress and sensitivity (s, plane strain)"}
 ];
```

Definitions of geometry, kinematics, strain energy ...

In[125]:=
```
ElementDefinitions[] := (
  Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
  wgp ⊨ SMSIO["Integration weight"[Ig]];
  {XIO, uIO} ⊨ SMSIO["All coordinates and DOFs"];
  Nh ⊨ 1 / 4 {(1 - ξ) (1 - η), (1 + ξ) (1 - η), (1 + ξ) (1 + η), (1 - ξ) (1 + η)};
  SMSFreeze[X, Append[Nh.XIO, ζ]];
  Je ⊨ SMSD[X, Ξ]; Jed ⊨ Det[Je];
  u ⊨ Append[Nh.uIO, 0];
  H ⊨ SMSD[u, X, "Dependency" → {Ξ, X, SMSInverse[Je]}];
  SMSFreeze[ϵ, 1 / 2 (H + Transpose[H]), "Ignore" -> NumberQ, "Symmetric" → True];
  ϵI ⊨ Extract[ϵ, {{1, 1}, {1, 2}, {2, 2}}];
  Igd ⊨ SMSInteger[(Ig - 1) lgd];
  DσDϵ ⊨ Table[SMSIO["Element data"[Igd + NoStressComponents + (j - 1) NoStressComponents + i]],
    {i, NoStressComponents}, {j, NoKinComponents}];
  σI ⊨ Table[SMSIO["Element data"[Igd + i]], {i, NoStressComponents}];
  SMSDefineDerivative[σI, ϵI, DσDϵ];
  {tζ} ⊨ SMSIO["All domain data"];
  σ ⊨ {{σI[[1]], σI[[2]], 0}, {σI[[2]], σI[[3]], 0}, {0, 0, σI[[4]]}};
  W ⊨ Tr[σ.Transpose[ϵ]];
  pe = Flatten[uIO];
  fGauss ⊨ Jed tζ;
 )
```

"Tangent and residual" user subroutine

In[126]:=
```
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
SMSDo[
  Rg ⊨ fGauss SMSD[W, pe, i, "Constant" → σI];
  SMSIO[wgp Rg, "Add to", "Residual"[i]];
  SMSDo[
   Kg ⊨ SMSD[Rg, pe, j];
   SMSIO[wgp Kg, "Add to", "Tangent"[i, j]];
   , {j, i, SMSNoDOFGlobal}];
  , {i, 1, SMSNoDOFGlobal}];
SMSEndDo[];
```

"Postprocessing" user subroutine

In[131]:=
```
SMSStandardModule["Postprocessing"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
SMSIO[{"DeformedMeshX" -> uIO[[All, 1]], "DeformedMeshY" -> uIO[[All, 2]],
   "u" -> uIO[[All, 1]], "v" -> uIO[[All, 2]]}, "Export to", "Nodal point post"];
SMSIO[{"Exx" → ϵ[[1, 1]], "Exy" → ϵ[[1, 2]], "Eyy" → ϵ[[2, 2]], "Sxx" → σ[[1, 1]], "Sxy" → σ[[1, 2]],
   "Syy" → σ[[2, 2]], "Szz" → σ[[3, 3]]}, "Export to", "Integration point post"[Ig]];
SMSEndDo[];
```

"Tasks" user subroutine

- 1 "FE^2" initialization data
  - 1 - number of micro problems associated with the element
  - 2 - index of character switch of macro element that defines the name of the task that returns multi-scale related macro data
  - 3 - length of multi-scale related macro data
  - 4 - index of character switch of macro element that defines the name of the micro problem task that returns multi-scale related micro data
  - 5 - length of multi-scale related micro data
- 2 "Material points"
- 3 "Small strain tensor (plane strain)"
  - vec $(\epsilon) = \{\epsilon_{11}, \epsilon_{12}, \epsilon_{22}\}$

In[137]:=
```
SMSStandardModule["Tasks"];
ng = SMSIO["No. integration points"];
task = SMSIO["Task index"];
SMSIf[task < 0
  , SMSSwitch[task
   , -1,
   SMSIO[{1, 0, 0, 5, 0}, "Export to", "Task data"];
   , -2,
   SMSIO[{1, 0, 0, 0, SMSNoDimensions * ng}, "Export to", "Task data"];
   , -3,
   SMSIO[{1, 0, 0, 0, NoKinComponents * ng}, "Export to", "Task data"];
  ];
  SMSReturn[];
 ];
SMSIf[task == 1
  , SMSIO[{ng, 3, NoKinComponents, 4, lgd}, "Export to", "Task integer output"];
 ];
SMSDo[
  ElementDefinitions[];
  SMSSwitch[task
   , 2,
   SMSIO[X[[{1, 2}]], "Export to",
     "Task real output"[Table[(Ig - 1) * SMSNoDimensions + i, {i, SMSNoDimensions}]]];
   , 3,
   SMSIO[∈I, "Export to",
     "Task real output"[Table[(Ig - 1) NoKinComponents + i, {i, NoKinComponents}]]];
  ];
  , {Ig, 1, ng}
 ];
```

Code generation

In[143]:=
```
SMSWrite[];
```

```
File: ExamplesQ1seMacroSymm.c  Size: 13565  Time: 3
```

Method	SKR	SPP	Tasks
No.Formulae	83	64	83
No.Leafs	1051	936	1087

In[144]:=
```
SMTMakeDll[];
```

Macro element for 2D solid FE^2 analysis - Q1 - P/F stress/strain pair - symmetric tangent

Description

Generate two - dimensional, four node macro finite element for the analysis of the micro - macro problems. The element has the following characteristics :

- quadrilateral topology,
- 4 node element,
- isoparametric mapping from the reference to the actual frame,
- global unknowns are displacements of the nodes,
- the element should allow arbitrary large displacements and rotations,

- the problem is defined by the deformation gradient **F**, the first Piola - Kirchoff stress tensor **P** and pseudo potential W^P = **P** : **F**. Integration point contribution to element residual is then $\mathbf{R}_g = \frac{\hat{\delta} W^P}{\hat{\delta}\, \mathbf{p}_e} \Big|_{\mathbf{P}=\text{const.}}$.

AceGen input for macro element

Initialization

```
In[145]:= << "AceGen`";
          NoStressComponents = 5; NoKinComponents = 4;
          lgd = (NoStressComponents + NoStressComponents NoKinComponents);
          Led = lgd es$$["id", "NoIntPoints"];
          SMSInitialize["ExamplesQ1PFMacroSymm", "Environment" → "AceFEM"];
          SMSTemplate["SMSTopology" → "Q1"
            , "SMSSymmetricTangent" → True
            , "SMSDomainDataNames" -> {"t -thickness"}
            , "SMSDefaultData" -> {1}
            , "SMSNoElementData" → Led
            , "SMSCharSwitch" → {"FE^2", "Material points", "Deformation gradient (plane strain)",
              "Integrated stress and sensitivity (P, plane strain)"}
           ];
```

Definitions of geometry, kinematics, strain energy ...

- the "Data" field is organized as
- Flatten$\left[\left\{\left\{\text{vec}(P), \frac{\partial \text{vec}(P)}{\partial \text{vec}(F)_1}, \frac{\partial \text{vec}(P)}{\partial \text{vec}(F)_2}, \ldots\right\}, \ldots \text{for all intergation points}\right\}\right]$ where
- vec (P) = {P_{11}, P_{12}, P_{21}, P_{22}, P_{33}} and vec (F) = {F_{11}, F_{12}, F_{21}, F_{22}}

```
In[151]:= ElementDefinitions[] := (
            Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
            {XIO, uIO} ⊨ SMSIO["All coordinates and DOFs"];
            Nh ⊨ 1 / 4 {(1 - ξ) (1 - η), (1 + ξ) (1 - η), (1 + ξ) (1 + η), (1 - ξ) (1 + η)};
            SMSFreeze[X, Append[Nh.XIO, ζ]];
            Je ⊨ SMSD[X, Ξ]; Jed ⊨ Det[Je];
            u ⊨ Append[Nh.uIO, 0];
            H ⊨ SMSD[u, X, "Dependency" → {Ξ, X, SMSInverse[Je]}];
            SMSFreeze[F, IdentityMatrix[3] + H, "Ignore" → PossibleZeroQ];
            FI = Extract[F, {{1, 1}, {1, 2}, {2, 1}, {2, 2}}];
            Igd ⊨ SMSInteger[(Ig - 1) lgd];
            DPDF ⊨ Table[SMSIO["Element data"[Igd + NoStressComponents + (j - 1) NoStressComponents + i]],
              {i, NoStressComponents}, {j, NoKinComponents}];
            PI ⊨ Table[SMSIO["Element data"[Igd + i]], {i, NoStressComponents}];
            SMSDefineDerivative[PI, FI, DPDF];
            P ⊨ {{PI[[1]], PI[[2]], 0}, {PI[[3]], PI[[4]], 0}, {0, 0, PI[[5]]}};
            {tζ} ⊨ SMSIO["All domain data"];
            W ⊨ Tr[P.Transpose[F]];
            pe = Flatten[uIO];
            fGauss ⊨ Jed tζ;
            wgp ⊨ SMSIO["Integration weight"[Ig]];
           )
```

"Tangent and residual" user subroutine

- ADB form of the problem leads to the element contribution to the governing equations of the problem in a form :

In[152]:=
```
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
SMSDo[
  Rg ⊨ fGauss SMSD[W, pe, i, "Constant" -> PI];
  SMSIO[wgp Rg, "Add to", "Residual"[i]];
  SMSDo[
   Kg ⊨ SMSD[Rg, pe, j];
   SMSIO[wgp Kg, "Add to", "Tangent"[i, j]];
   , {j, i, SMSNoDOFGlobal}];
  , {i, 1, SMSNoDOFGlobal}];
SMSEndDo[];
```

"Postprocessing" user subroutine

In[157]:=
```
SMSStandardModule["Postprocessing"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
SMSIO[{"DeformedMeshX" -> uIO[[All, 1]], "DeformedMeshY" -> uIO[[All, 2]],
    "u" -> uIO[[All, 1]], "v" -> uIO[[All, 2]]}, "Export to", "Nodal point post"];
Eg ⊨ 1 / 2 (Transpose[F].F - IdentityMatrix[3]);
σ ⊨ (1 / Det[F]) * P . Transpose[F];
SMSIO[{"Exx" → Eg[[1, 1]], "Exy" → Eg[[1, 2]], "Eyy" → Eg[[2, 2]], "Sxx" → σ[[1, 1]], "Sxy" → σ[[1, 2]],
    "Syy" → σ[[2, 2]], "Szz" → σ[[3, 3]]}, "Export to", "Integration point post"[Ig]];
SMSEndDo[];
```

"Tasks" user subroutine

- 1 "FE^2" initialization data
  - 1 - number of micro problems associated with the element
  - 2 - index of character switch of macro element that defines the name of the task that returns multi-scale related macro data
  - 3 - length of multi-scale related macro data
  - 4 - index of character switch of macro element that defines the name of the micro problem task that returns multi-scale related micro data
  - 5 - length of multi-scale related micro data
- 2 "Material points"
  - Join$[X_1, X_2, \ldots, X_{ng}]$
- 3 "Deformation gradient (plane strain)"
  - vec (F) = $\{F_{11}, F_{12}, F_{21}, F_{22}\}$

In[165]:=
```
SMSStandardModule["Tasks"];
ng = SMSIO["No. integration points"];
task = SMSIO["Task index"];
SMSIf[task < 0
  , SMSSwitch[task
   , -1,
   SMSIO[{1, 0, 0, 5, 0}, "Export to", "Task data"];
   , -2,
   SMSIO[{1, 0, 0, 0, SMSNoDimensions * ng}, "Export to", "Task data"];
   , -3,
   SMSIO[{1, 0, 0, 0, NoKinComponents * ng}, "Export to", "Task data"];
  ];
  SMSReturn[];
 ];
SMSIf[task == 1
  , SMSIO[{ng, 3, NoKinComponents, 4, lgd}, "Export to", "Task integer output"];
 ];
SMSDo[
  ElementDefinitions[];
  SMSSwitch[task
   , 2,
   SMSIO[X[[{1, 2}]], "Export to",
     "Task real output"[Table[(Ig - 1) * SMSNoDimensions + i, {i, SMSNoDimensions}]]];
   , 3,
   SMSIO[FI, "Export to",
     "Task real output"[Table[(Ig - 1) NoKinComponents + i, {i, NoKinComponents}]]];
  ];
  , {Ig, 1, ng}
 ];
```

Code generation

In[171]:=
```
SMSWrite[];
```

```
File: ExamplesQ1PFMacroSymm.c  Size: 14566  Time: 4
```

Method	SKR	SPP	Tasks
No.Formulae	93	71	84
No.Leafs	1318	1071	1094

In[172]:=
```
SMTMakeDll[];
```

Macro element for 2D solid FE^2 analysis - Q1 - P/F stress/strain pair - unsymmetric tangent

Description

Generate two - dimensional, four node macro finite element for the analysis of the micro - macro problems. The element has the following characteristics :

- quadrilateral topology,
- 4 node element,
- isoparametric mapping from the reference to the actual frame,
- global unknowns are displacements of the nodes,
- the element should allow arbitrary large displacements and rotations,

- the problem is defined by the deformation gradient **F**, the first Piola - Kirchoff stress tensor **P** and pseudo potential $W^P = \mathbf{P} : \mathbf{F}$. Integration point contribution to element residual is then $\mathbf{R}_g = \frac{\hat{\delta} W^P}{\hat{\delta}\, \mathbf{p}_e} \Big|_{\mathbf{P}=\text{const.}}$.

AceGen input for macro element

Initialization

In[173]:=
```
<< "AceGen`";
NoStressComponents = 5; NoKinComponents = 4;
lgd = (NoStressComponents + NoStressComponents NoKinComponents);
Led = lgd es$$["id", "NoIntPoints"];
SMSInitialize["ExamplesQ1PFMacroUnsymm", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "Q1"
  , "SMSSymmetricTangent" → False
  , "SMSDomainDataNames" -> {"t -thickness"}
  , "SMSDefaultData" -> {1}
  , "SMSNoElementData" → Led
  , "SMSCharSwitch" → {"FE^2", "Material points", "Deformation gradient (plane strain)",
    "Integrated stress and sensitivity (P, plane strain)"}
 ];
```

Definitions of geometry, kinematics, strain energy ...

- the "Data" field is organized as
- Flatten$\left[\left\{\left\{\text{vec}(P), \frac{\partial \text{vec}(P)}{\partial \text{vec}(F)_1}, \frac{\partial \text{vec}(P)}{\partial \text{vec}(F)_2}, \ldots\right\}, \ldots \text{for all intergation points}\right\}\right]$ where
- vec (P) = {P_{11}, P_{12}, P_{21}, P_{22}, P_{33}} and vec (F) = {F_{11}, F_{12}, F_{21}, F_{22}}

In[179]:=
```
ElementDefinitions[] := (
  Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
  {XIO, uIO} ⊨ SMSIO["All coordinates and DOFs"];
  Nh ⊨ 1 / 4 {(1 - ξ) (1 - η), (1 + ξ) (1 - η), (1 + ξ) (1 + η), (1 - ξ) (1 + η)};
  SMSFreeze[X, Append[Nh.XIO, ζ]];
  Je ⊨ SMSD[X, Ξ]; Jed ⊨ Det[Je];
  u ⊨ Append[Nh.uIO, 0];
  H ⊨ SMSD[u, X, "Dependency" → {Ξ, X, SMSInverse[Je]}];
  SMSFreeze[F, IdentityMatrix[3] + H, "Ignore" → PossibleZeroQ];
  FI = Extract[F, {{1, 1}, {1, 2}, {2, 1}, {2, 2}}];
  Igd ⊨ SMSInteger[(Ig - 1) lgd];
  DPDF ⊨ Table[SMSIO["Element data"[Igd + NoStressComponents + (j - 1) NoStressComponents + i]],
    {i, NoStressComponents}, {j, NoKinComponents}];
  PI ⊨ Table[SMSIO["Element data"[Igd + i]], {i, NoStressComponents}];
  SMSDefineDerivative[PI, FI, DPDF];
  P ⊨ {{PI[[1]], PI[[2]], 0}, {PI[[3]], PI[[4]], 0}, {0, 0, PI[[5]]}};
  {tζ} ⊨ SMSIO["All domain data"];
  W ⊨ Tr[P.Transpose[F]];
  pe = Flatten[uIO];
  fGauss ⊨ Jed tζ;
  wgp ⊨ SMSIO["Integration weight"[Ig]];
 )
```

"Tangent and residual" user subroutine

- ADB form of the problem leads to the element contribution to the governing equations of the problem in a form :

In[180]:=
```
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
SMSDo[
  Rg ⊨ fGauss SMSD[W, pe, i, "Constant" -> PI];
  SMSIO[wgp Rg, "Add to", "Residual"[i]];
  SMSDo[
   Kg ⊨ SMSD[Rg, pe, j];
   SMSIO[wgp Kg, "Add to", "Tangent"[i, j]];
   , {j, 1, SMSNoDOFGlobal}];
  , {i, 1, SMSNoDOFGlobal}];
SMSEndDo[];
```

"Postprocessing" user subroutine

In[185]:=
```
SMSStandardModule["Postprocessing"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
SMSIO[{"DeformedMeshX" -> uIO[[All, 1]], "DeformedMeshY" -> uIO[[All, 2]],
    "u" -> uIO[[All, 1]], "v" -> uIO[[All, 2]]}, "Export to", "Nodal point post"];
Eg ⊨ 1 / 2 (Transpose[F].F - IdentityMatrix[3]);
σ ⊨ (1 / Det[F]) * P . Transpose[F];
SMSIO[{"Exx" → Eg[[1, 1]], "Exy" → Eg[[1, 2]], "Eyy" → Eg[[2, 2]], "Sxx" → σ[[1, 1]], "Sxy" → σ[[1, 2]],
    "Syy" → σ[[2, 2]], "Szz" → σ[[3, 3]]}, "Export to", "Integration point post"[Ig]];
SMSEndDo[];
```

"Tasks" user subroutine

- 1 "FE^2" initialization data
  - 1 - number of micro problems associated with the element
  - 2 - index of character switch of macro element that defines the name of the task that returns multi-scale related macro data
  - 3 - length of multi-scale related macro data
  - 4 - index of character switch of macro element that defines the name of the micro problem task that returns multi-scale related micro data
  - 5 - length of multi-scale related micro data
- 2 "Material points"
  - Join$[X_1, X_2, \ldots, X_{ng}]$
- 3 "Deformation gradient (plane strain)"
  - vec (F) = $\{F_{11}, F_{12}, F_{21}, F_{22}\}$

In[193]:=
```
SMSStandardModule["Tasks"];
ng = SMSIO["No. integration points"];
task = SMSIO["Task index"];
SMSIf[task < 0
  , SMSSwitch[task
   , -1,
   SMSIO[{1, 0, 0, 5, 0}, "Export to", "Task data"];
   , -2,
   SMSIO[{1, 0, 0, 0, SMSNoDimensions * ng}, "Export to", "Task data"];
   , -3,
   SMSIO[{1, 0, 0, 0, NoKinComponents * ng}, "Export to", "Task data"];
  ];
  SMSReturn[];
 ];
SMSIf[task == 1
  , SMSIO[{ng, 3, NoKinComponents, 4, lgd}, "Export to", "Task integer output"];
 ];
SMSDo[
  ElementDefinitions[];
  SMSSwitch[task
   , 2,
   SMSIO[X[[{1, 2}]], "Export to",
     "Task real output"[Table[(Ig - 1) * SMSNoDimensions + i, {i, SMSNoDimensions}]]];
   , 3,
   SMSIO[FI, "Export to",
     "Task real output"[Table[(Ig - 1) NoKinComponents + i, {i, NoKinComponents}]]];
  ];
  , {Ig, 1, ng}
 ];
```

Code generation

In[199]:=
```
SMSWrite[];
```

```
File: ExamplesQ1PFMacroUnsymm.c  Size: 14565  Time: 4
```

Method	SKR	SPP	Tasks
No.Formulae	93	71	84
No.Leafs	1318	1071	1094

In[200]:=
```
SMTMakeDll[];
```

Macro element for 2D MIEL analysis - Q1 - symmetric tangent

Description

Generate two - dimensional, four node macro finite element for the analysis of the MIEL formulation of multi - scale formulation of steady state problems that result in symmetric tangent matrix.

The element has the following characteristics :

- quadrilateral topology,
- 4 node element,
- global unknowns are displacements of the nodes,
- elements expects components of the residual and the symmetric tangent matrix to be available in element data field "Data" , thus micro_data = $\int_{\Omega_e} \text{Flatten}\left[\left\{W, \frac{\partial W}{\partial u_M} \mid_{\mathbf{h}=\text{const.}}, \text{vec}\left(\frac{\partial}{\partial u_M}\left(\frac{\partial W}{\partial u_M} \mid_{\mathbf{h}=\text{const.}}\right)\right)\right\}\right] dV$. u_M are degrees of freedom used to parameterize

prescribed essential boundary conditions of the macro problem. The term $\int_{\Omega_e} \frac{\partial W}{\partial u_M} \,|_{\mathbf{h}=\text{const.}}$ dV represents contribution of the micro element to macro residual and $\int_{\Omega_e} \text{vec}\left(\frac{\partial}{\partial u_M}\left(\frac{\partial W}{\partial u_M} \,|_{\mathbf{h}=\text{const.}}\right)\right)$ dV contribution of the micro element to condensed tangent matrix for MIEL multi-scale simulations. If second order directional derivatives $\text{vec}\left(\frac{\partial}{\partial u_M}\left(\frac{\partial W}{\partial u_M} \,|_{\mathbf{h}=\text{const.}}\right)\right)$ are symmetric then only upper triangular matrix is calculated and stored.

- Tangent matrix has $n(n+1)/2$ components that are stored in data field "Data" by rows as follows:
$\{K_{11}, K_{12}, \ldots, K_{1n}, K_{22}, \ldots, K_{2n}, \ldots, K_{nn}\}$

AceGen input for MIEL macro element

In[201]:=
```
Get["AceGen`"];
```

In[202]:=
```
SMSInitialize["ExamplesQ1MIELMacroSymm", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "Q1"
  , "SMSSymmetricTangent" → True
  , "SMSCharSwitch" → {"MIEL", "Nodal displacements (2D)",
    "Integrated strain energy and derivatives (plane strain)"}];
```

In[204]:=
```
SMSNoElementData = 1 + SMSNoDOFGlobal + SMSNoDOFGlobal (SMSNoDOFGlobal + 1) / 2;
```

"Tangent and residual" user subroutine

In[205]:=
```
SMSStandardModule["Tangent and residual"];
```

In[206]:=
```
row ⫤ SMSInteger[SMSNoDOFGlobal + 2];
SMSDo[
  SMSIO[SMSIO["Element data"[1 + i]], "Export to", "Residual"[i]];
  SMSDo[
   SMSIO[SMSIO["Element data"[row + j - i]], "Export to", "Tangent"[i, j]];
   , {j, i, SMSNoDOFGlobal}];
  row ⫣ row + SMSNoDOFGlobal - i + 1;
  , {i, 1, SMSNoDOFGlobal, 1, row}];
```

"Postprocessing" user subroutine

In[208]:=
```
SMSStandardModule["Postprocessing"];
uIO ⫦ SMSIO["All DOFs"];
SMSIO[{"DeformedMeshX" -> uIO[[All, 1]], "DeformedMeshY" -> uIO[[All, 2]],
    "u" -> uIO[[All, 1]], "v" -> uIO[[All, 2]]}, "Export to", "Nodal point post"];
```

"Tasks" user subroutine

- 1 "MIEL" initialization data
  - 1 - number of micro problems associated with the element
  - 2 - index of character switch of macro element that defines the name of the task that returns multi-scale related macro data
  - 3 - length of multi-scale related macro data
  - 4 - index of character switch of macro element that defines the name of the micro problem task that returns multi-scale related micro data
  - 5 - length of multi-scale related micro data
- 2 "Nodal displacements (2D)" returns Flatten[p_e]

In[211]:=
```
SMSStandardModule["Tasks"];
task ⫦ SMSIO["Task index"];
```

In[213]:=
```
SMSIf[task < 0
 , SMSSwitch[task
  , -1,
  SMSIO[{1, 0, 0, 5, 0}, "Export to", "Task data"];
  , -2,
  SMSIO[{1, 0, 0, 0, SMSNoDOFGlobal}, "Export to", "Task data"];
 ];
 SMSReturn[];
];
```

In[214]:=
```
SMSSwitch[task
 , 1,
 SMSIO[{1, 2, SMSNoDOFGlobal, 3, SMSNoElementData}, "Export to", "Task integer output"];
 , 2,
 SMSIO[SMSIO["All DOFs"] // Flatten,
   "Export to", "Task real output"[Table[i, {i, SMSNoDOFGlobal}]]];
];
```

In[215]:=
```
SMSWrite[];
```

File: ExamplesQ1MIELMacroSymm.c Size: 6503 Time: 1

Method	SKR	SPP	Tasks
No.Formulae	5	9	11
No.Leafs	71	202	280

In[216]:=
```
SMTMakeDll[];
```

Macro element for 2D MIEL analysis - Q1 - unsymmetric tangent

Description

Generate two - dimensional, four node macro finite element for the analysis of the MIEL formulation of multi - scale formulation of coupled transient problems thet results in unsymmetric tangent matrix.

The element has the following characteristics :

- quadrilateral topology,
- 4 node element,
- global unknowns are displacements of the nodes,
- elements expects components of the residual and the symmetric tangent matrix to be available in element data field "Data" , thus micro_data = $\int_{\Omega_e}$ Flatten$\left[\left\{W, \frac{\partial W}{\partial u_M}\,|_{\mathbf{h}=\text{const.}}, \text{vec}\left(\frac{\partial}{\partial u_M}\left(\frac{\partial W}{\partial u_M}\,|_{\mathbf{h}=\text{const.}}\right)\right)\right\}\right]$ dV. u_M are degrees of freedom used to parameterize prescribed essential boundary conditions of the macro problem. The term $\int_{\Omega_e} \frac{\partial W}{\partial u_M}\,|_{\mathbf{h}=\text{const.}}$ dV represents contribution of the micro element to macro residual and $\int_{\Omega_e} \text{vec}\left(\frac{\partial}{\partial u_M}\left(\frac{\partial W}{\partial u_M}\,|_{\mathbf{h}=\text{const.}}\right)\right)$ dV contribution of the micro element to condensed tangent matrix for MIEL multi-scale simulations. If second order directional derivatives $\text{vec}\left(\frac{\partial}{\partial u_M}\left(\frac{\partial W}{\partial u_M}\,|_{\mathbf{h}=\text{const.}}\right)\right)$ are symmetric then only upper triangular matrix is calculated and stored.
- Tangent matrix has n^2 components that are stored in data field "Data" by rows as follows: $\{K_{11}, K_{12}, \ldots, K_{1n}, K_{21}, \ldots, K_{2n}, \ldots, K_{nn}\}$

AceGen input for MIEL macro element

In[217]:=
```
Get["AceGen`"];
```

In[218]:=
```
SMSInitialize["ExamplesQ1MIELMacroUnsymm", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "Q1"
  , "SMSSymmetricTangent" → False
  , "SMSCharSwitch" → {"MIEL", "Nodal displacements (2D)",
    "Integrated strain energy and derivatives (plane strain)"}];
```

In[220]:=
```
SMSNoElementData = 1 + SMSNoDOFGlobal + SMSNoDOFGlobal SMSNoDOFGlobal;
```

"Tangent and residual" user subroutine

In[221]:=
```
SMSStandardModule["Tangent and residual"];
```

In[222]:=
```
row = SMSInteger[SMSNoDOFGlobal + 2];
SMSDo[
  SMSIO[SMSIO["Element data"[1 + i]], "Export to", "Residual"[i]];
  SMSDo[
   SMSIO[SMSIO["Element data"[row + j - 1]], "Export to", "Tangent"[i, j]];
   , {j, 1, SMSNoDOFGlobal}];
  row = row + SMSNoDOFGlobal;
  , {i, 1, SMSNoDOFGlobal, 1, row}];
```

"Postprocessing" user subroutine

In[224]:=
```
SMSStandardModule["Postprocessing"];
uIO = SMSIO["All DOFs"];
SMSIO[{"DeformedMeshX" -> uIO[[All, 1]], "DeformedMeshY" -> uIO[[All, 2]],
    "u" -> uIO[[All, 1]], "v" -> uIO[[All, 2]]}, "Export to", "Nodal point post"];
```

"Tasks" user subroutine

- 1 "MIEL" initialization data
  - 1 - number of micro problems associated with the element
  - 2 - index of character switch of macro element that defines the name of the task that returns multi-scale related macro data
  - 3 - length of multi-scale related macro data
  - 4 - index of character switch of macro element that defines the name of the micro problem task that returns multi-scale related micro data
  - 5 - length of multi-scale related micro data
- 2 "Nodal displacements (2D)" returns Flatten[ph]

In[227]:=
```
SMSStandardModule["Tasks"];
task = SMSIO["Task index"];
SMSIf[task < 0
  , SMSSwitch[task
   , -1,
   SMSIO[{1, 0, 0, 5, 0}, "Export to", "Task data"];
   , -2,
   SMSIO[{1, 0, 0, 0, SMSNoDOFGlobal}, "Export to", "Task data"];
  ];
  SMSReturn[];
 ];
SMSSwitch[task
  , 1,
  SMSIO[{1, 2, SMSNoDOFGlobal, 3, SMSNoElementData}, "Export to", "Task integer output"];
  , 2,
  SMSIO[SMSIO["All DOFs"] // Flatten,
    "Export to", "Task real output"[Table[i, {i, SMSNoDOFGlobal}]]];
 ];
```

In[231]:= **SMSWrite[];**

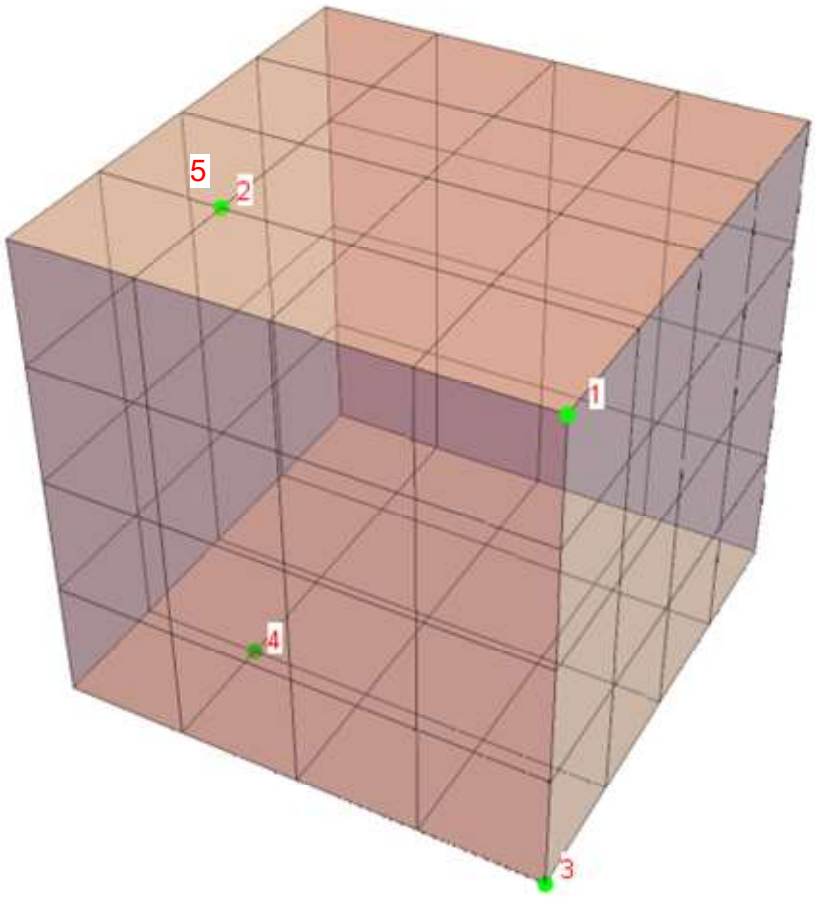
File: ExamplesQ1MIELMacroUnsymm.c Size: 6485 Time: 1

Method	SKR	SPP	Tasks
No.Formulae	4	9	11
No.Leafs	63	202	280

In[232]:= **SMTMakeDll[];**

Periodic boundary constraint element for 3D solid analysis - structured mesh

Description



Element has 5 nodes:

- nodes 1 and 3 are corner nodes
- node 2 is a node on the master surface
- nodes 4 represent slave node on oposite surface
- node 5 holds Lagrange multipliers. It is positioned at the position of the master node 2. Master nodes at the edges can have several periodic elements associated, but they all share the same Lagrange multipliers.

Periodicity condition: $u_4 - u_2 = u_3 - u_1$.

RVE mesh is composed of solid mesh and mesh of elements that are used to impose perodicity condition.

AceGen input for periodic BC element

In[233]:=
```
<< AceGen`;
SMSInitialize["ExamplesPeriodic3DStructured", "Environment" → "AceFEM"];
SMSTemplate[
  (*element is point in 3D*)
  "SMSTopology" → "V3",
  "SMSDefaultIntegrationCode" → 0,
  "SMSNoNodes" → 5,
  "SMSDOFGlobal" → {3, 3, 3, 3, 3},
  (*no default postprocessing*)
  "SMSSegments" → {{2}, {4}},
  "SMSSegmentsTriangulation" → {{}, {}},
  (*define additional local auxiliary node that holds Lagrange multipliers*)
  "SMSNodeID" → {"D", "D", "D", "D", "PerConst -F -E -L"},
  "SMSAdditionalNodes" → Function[{X1, X2, X3, X4}, {X2}],
  "SMSSymmetricTangent" → True,
  "SMSDomainDataNames" -> {"ρ -augmented Lagrange penalty parameter"},
  "SMSDefaultData" → {10},
  "SMSShapeSensitivity" -> True, "SMSEBCSensitivity" -> True
 ];
```

In[236]:=
```
ElementDefinitions[] := (
  uIO ⊨ SMSIO["All DOFs"];
  {ρ} ⊨ SMSIO["All domain data"];
  pe = Flatten[uIO];
  {λx, λy, λz} ⊨ uIO[[5]];
  {gx, gy, gz} ⊨ {
    (uIO[[3, 1]] - uIO[[1, 1]]) - (uIO[[4, 1]] - uIO[[2, 1]]),
    (uIO[[3, 2]] - uIO[[1, 2]]) - (uIO[[4, 2]] - uIO[[2, 2]]),
    (uIO[[3, 3]] - uIO[[1, 3]]) - (uIO[[4, 3]] - uIO[[2, 3]])
   };
  Π = (λx gx + ρ / 2 gx^2) + (λy gy + ρ / 2 gy^2) + (λz gz + ρ / 2 gz^2);
 )
```

In[237]:=
```
SMSStandardModule["Tangent and residual"];
ElementDefinitions[];
Rel ⊨ SMSD[Π, pe];
SMSIO[Rel, "Export to", "Residual"];
Ke ⊨ SMSD[Rel, pe];
SMSIO[Ke, "Export to", "Tangent"];
```

In[243]:=
```
SMSStandardModule["Sensitivity pseudo-load"];
SMSDo[is, SMSIO["SensIndexStart"], SMSIO["SensIndexEnd"]];
ϕis ⊨ SMSFictive[];
ElementDefinitions[];
"∂pe/∂ϕi here is first order essential boundary conditions velocity field";
SMSDefineDerivative[SMSIO["Nodal DOFs"], ϕis, SMSIO["Sensitivity DOFs"[is]]];
Rel ⊨ SMSD[Π, pe];
SMSIO[SMSD[Rel, ϕis], "Add to", "First order pseudo load"[is]];
SMSEndDo[];
```

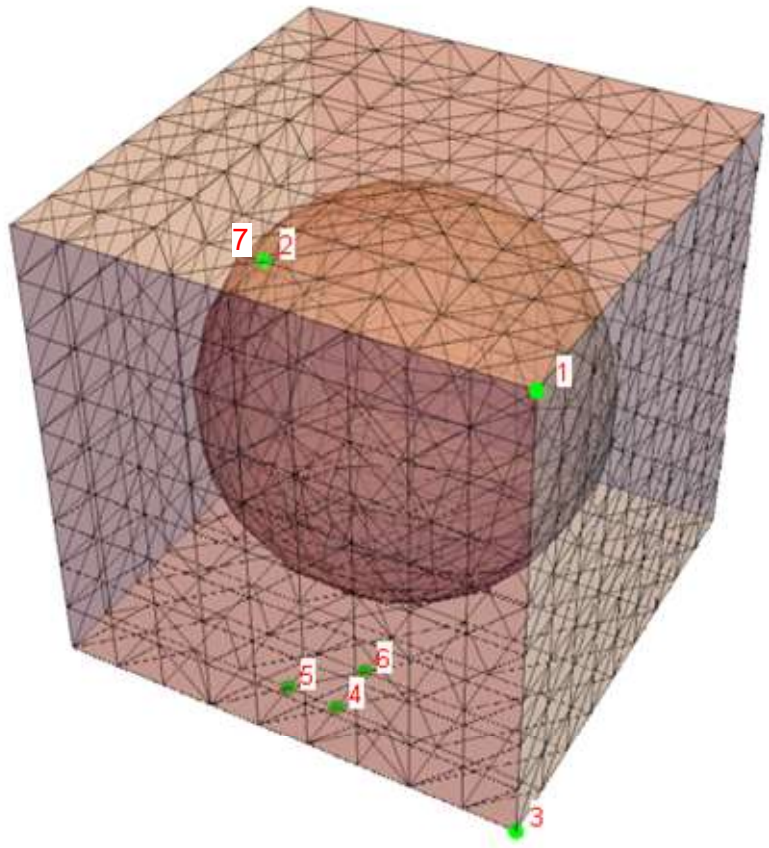
In[252]:=
```
SMSWrite[];
SMTMakeDll[];
```

```
File: ExamplesPeriodic3DStructured.c  Size: 7602  Time: 1
```

Method	SKR	SSE
No.Formulae	10	15
No.Leafs	1212	471

Periodic boundary constraint element for 3D solid analysis - unstructured mesh

Description

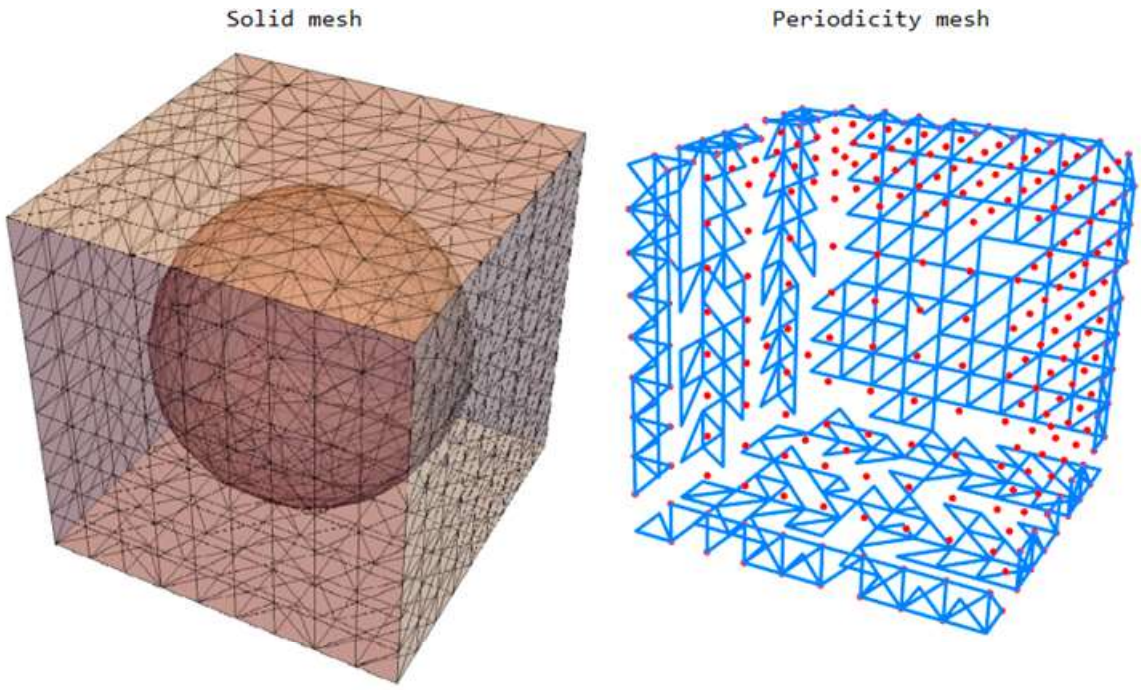


Element has 7 nodes:

- nodes 1 and 3 are corner nodes
- node 2 is a node on the master surface
- nodes 4, 5, 6 represent slave triangle on oposite surface
- node 7 holds Lagrange multipliers. It is positioned at the position of the master node 2. Master nodes at the edges can have several periodic elements associated, but they all share the same Lagrange multipliers.

Periodicity condition: $u_p(u_4, u_5, u_6) - u_2 = u_3 - u_1$ where $u_p(u_4, u_5, u_6)$ is displacement at the projection point of master node 2 obtained by iterpolating displacements of nodes 4, 5 and 6.

RVE mesh is composed of solid mesh and mesh of elements that are used to impose perodicity condition.



AceGen input for periodic BC element

In[254]:=
```
<< AceGen`;
SMSInitialize["ExamplesPeriodic3DUnstructured", "Environment" → "AceFEM"];
SMSTemplate[
  (*element is point in 3D*)
  "SMSTopology" → "V3",
  "SMSDefaultIntegrationCode" → 0,
  "SMSNoNodes" → 7,
  "SMSDOFGlobal" → {3, 3, 3, 3, 3, 3, 3},
  (*no default postprocessing*)
  "SMSSegments" → {{2}, {4, 5, 6, 4}},
  (*define additional local auxiliary node that holds Lagrange multipliers*)
  "SMSNodeID" → {"D", "D", "D", "D", "D", "D", "PerConst -F -E -L"},
  "SMSAdditionalNodes" → Hold[{#2} &], (*Function[{X1,X2,X3,X4,X5,X6},{X2}],*)
  "SMSSymmetricTangent" → True,
  "SMSDomainDataNames" -> {"ρ -augmented Lagrange penalty parameter"},
  "SMSDefaultData" → {10},
  "SMSShapeSensitivity" -> True, "SMSEBCSensitivity" -> True
 ];
```

In[257]:=
```
ElementDefinitions[] := (
  {XIO, uIO} ⊨ SMSIO["All coordinates and DOFs"];
  {ρ} ⊨ SMSIO["All domain data"];
  pe = Flatten[uIO];
  {x, y, z} ⊨ SMSWhich[
    XIO[[1, 1]] + XIO[[3, 1]] == 0, {XIO[[3, 1]], XIO[[2, 2]], XIO[[2, 3]]},
    XIO[[1, 2]] + XIO[[3, 2]] == 0, {XIO[[2, 1]], XIO[[3, 2]], XIO[[2, 3]]},
    XIO[[1, 3]] + XIO[[3, 3]] == 0, {XIO[[2, 1]], XIO[[2, 2]], XIO[[3, 3]]}
   ];
  d1 = -((-XIO[[4, 1]] + XIO[[5, 1]])^2 + (-XIO[[4, 2]] + XIO[[5, 2]])^2 + (-XIO[[4, 3]] + XIO[[5, 3]])^2);
  d2 =
   -((-XIO[[4, 1]] + XIO[[6, 1]])^2 + (-XIO[[4, 2]] + XIO[[6, 2]])^2 + (-XIO[[4, 3]] + XIO[[6, 3]])^2);
  d3 = -(-XIO[[4, 1]] + XIO[[5, 1]]) (-XIO[[4, 1]] + XIO[[6, 1]]) - (-XIO[[4, 2]] + XIO[[5, 2]])
      (-XIO[[4, 2]] + XIO[[6, 2]]) - (-XIO[[4, 3]] + XIO[[5, 3]]) (-XIO[[4, 3]] + XIO[[6, 3]]) ;
  α = (d2 * ((x - XIO[[4, 1]]) * (XIO[[4, 1]] - XIO[[5, 1]]) + (y - XIO[[4, 2]]) *
            (XIO[[4, 2]] - XIO[[5, 2]]) + (z - XIO[[4, 3]]) * (XIO[[4, 3]] - XIO[[5, 3]])) +
       d3 * (-((x - XIO[[4, 1]]) * (XIO[[4, 1]] - XIO[[6, 1]])) - (y - XIO[[4, 2]]) * (XIO[[4, 2]] -
             XIO[[6, 2]]) - (z - XIO[[4, 3]]) * (XIO[[4, 3]] - XIO[[6, 3]]))) / (d1 * d2 - d3^2);
  β = (d3 * (-((x - XIO[[4, 1]]) * (XIO[[4, 1]] - XIO[[5, 1]])) - (y - XIO[[4, 2]]) *
            (XIO[[4, 2]] - XIO[[5, 2]]) - (z - XIO[[4, 3]]) * (XIO[[4, 3]] - XIO[[5, 3]])) + d1 *
        ((x - XIO[[4, 1]]) * (XIO[[4, 1]] - XIO[[6, 1]]) + (y - XIO[[4, 2]]) * (XIO[[4, 2]] - XIO[[6, 2]]) +
          (z - XIO[[4, 3]]) * (XIO[[4, 3]] - XIO[[6, 3]]))) / (d1 * d2 - d3^2);
  {i1, i2, i3} ⊨ {1 - α - β, α, β};
  {λx, λy, λz} ⊨ uIO[[7]];
  {gx, gy, gz} ⊨ {
    (uIO[[3, 1]] - uIO[[1, 1]]) - ((i1 uIO[[4, 1]] + i2 uIO[[5, 1]] + i3 uIO[[6, 1]]) - uIO[[2, 1]]),
    (uIO[[3, 2]] - uIO[[1, 2]]) - ((i1 uIO[[4, 2]] + i2 uIO[[5, 2]] + i3 uIO[[6, 2]]) - uIO[[2, 2]]),
    (uIO[[3, 3]] - uIO[[1, 3]]) - ((i1 uIO[[4, 3]] + i2 uIO[[5, 3]] + i3 uIO[[6, 3]]) - uIO[[2, 3]])
   };
  Π = (λx gx + ρ / 2 gx^2) + (λy gy + ρ / 2 gy^2) + (λz gz + ρ / 2 gz^2);
 )
```

In[258]:=
```
SMSStandardModule["Tangent and residual"];
ElementDefinitions[];
Rel ⊨ SMSD[Π, pe];
SMSIO[Rel, "Export to", "Residual"];
Ke ⊨ SMSD[Rel, pe];
SMSIO[Ke, "Export to", "Tangent"];
```

In[264]:=
```
SMSStandardModule["Sensitivity pseudo-load"];
SMSDo[is, SMSIO["SensIndexStart"], SMSIO["SensIndexEnd"]];
ϕis ⊨ SMSFictive[];
ElementDefinitions[];
"∂pe/∂ϕi here is first order essential boundary conditions velocity field";
SMSDefineDerivative[SMSIO["Nodal DOFs"], ϕis, SMSIO["Sensitivity DOFs"[is]]];
Rel ⊨ SMSD[Π, pe];
SMSIO[SMSD[Rel, ϕis], "Add to", "First order pseudo load"[is]];
SMSEndDo[];(*SensIndex*)
```

In[273]:=
```
SMSWrite[];
SMTMakeDll[];
```

```
File: ExamplesPeriodic3DUnstructured.c  Size: 12861  Time: 3
```

Method	SKR	SSE
No.Formulae	59	58
No.Leafs	2702	1109

Dummy surface element for triangulation of higher order surfaces

Description

This is utility element. It will create a layer of triangels covering the surface of the body. With the command SMTPointLocation[{x,y,z},"boundarysurface",tolerance] we can then locate the closest triangle to the given point.

AceGen input

In[275]:=
```
<< AceGen`;
SMSInitialize["ExamplesSurfaceTriangulationP1", "Environment" → "AceFEM"];
SMSTemplate[
  "SMSTopology" → "P1",
  "SMSNoNodes" → 3,
  "SMSDOFGlobal" → {3, 3, 3}
 ];
```

In[278]:=
```
SMSStandardModule["Tangent and residual"];
```

In[279]:=
```
SMSWrite[];
SMTMakeDll[];
```

```
File: ExamplesSurfaceTriangulationP1.c  Size: 3774  Time: 0
```

Method	SKR
No.Formulae	0
No.Leafs	0

Macro element for 3D solid FE^2 analysis - H1 - P/F stress/strain pair - symmetric tangent

Description

Generate three-dimensional, eight node macro finite element for the analysis of the micro-macro problems. The element has the following characteristics:

⇒ 3D hexahedral element,

⇒ 8 node element,

⇒ isoparametric mapping from the reference to the actual frame,

⇒ global unknowns are displacements of the nodes,

⇒ the element should allow arbitrary large displacements and rotations,

⇒ the problem is defined by the deformation gradient $F = I + \nabla u$ and the first Piola-Kirchoff stress tensor P and pseudo potential

$$\Pi = \int_{\Omega_0} \mathbf{P} : \mathbf{F}\, d\Omega_0$$

Ω_0 is the initial domain of the problem.

⇒ ADB form of the problem leads to the element contribution to the governing equations of the problem in a form:

$$\mathbf{R}_e = \frac{\hat{\delta}\Pi}{\hat{\delta}\mathbf{p}_e} \Big|_{\frac{DP}{Dp_e}=0}$$

AceGen input for macro element

Initialization

In[281]:=
```
Get["AceGen`"];
NoStressComponents = 9; NoKinComponents = 9;
lgd = (NoStressComponents + NoStressComponents NoKinComponents);
Led = lgd es$$["id", "NoIntPoints"];
```

In[285]:=
```
SMSInitialize["ExamplesH1PFMacroSymm", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "H1"
  , "SMSSymmetricTangent" → True
  , "SMSDomainDataNames" -> {"ρ0 -density", "bX -force per unit mass X",
    "bY -force per unit mass Y", "bZ -force per unit mass Z"}
  , "SMSDefaultData" -> {1, 0, 0, 0}
  , "SMSNoElementData" → Led
  , "SMSCharSwitch" → {"FE^2", "Material points",
    "Deformation gradient (3D)", "Integrated stress and sensitivity (P, 3D)"}
 ];
```

Definitions of geometry, kinematics, strain energy ...

- the "Data" field is organized as
- Flatten$\left[\left\{\left\{\text{vec (P)}, \frac{\partial \text{vec (P)}}{\partial \text{vec (F)}_1}, \frac{\partial \text{vec (P)}}{\partial \text{vec (F)}_2}, \ldots\right\}, \ldots \text{for all intergation points}\right\}\right]$ where
- vec (P) = {P_{11}, P_{12}, P_{13}, P_{21}, P_{22}, P_{23}, P_{13}, P_{23}, P_{33}} and vec (F) = {F_{11}, F_{12}, F_{13}, F_{21}, $F_{22,}$, F_{23}, F_{31}, F_{32}, F_{33}}

In[287]:=
```
ElementDefinitions[] := (
  Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
  {XIO, uIO} ⊨ SMSIO["All coordinates and DOFs"];
  Ξn = {{-1, -1, -1}, {1, -1, -1}, {1, 1, -1}, {-1, 1, -1},
    {-1, -1, 1}, {1, -1, 1}, {1, 1, 1}, {-1, 1, 1}};
  Nh ⊨ Table[1 / 8 (1 + ξ Ξn⟦i, 1⟧) (1 + η Ξn⟦i, 2⟧) (1 + ζ Ξn⟦i, 3⟧), {i, 1, 8}];
  SMSFreeze[X, Nh.XIO];
  Je ⊨ SMSD[X, Ξ]; Jed ⊨ Det[Je];
  pe = Flatten[uIO]; u ⊨ Nh.uIO;
  Hg ⊨ SMSD[u, X, "Dependency" → {Ξ, X, SMSInverse[Je]}];
  SMSFreeze[F, IdentityMatrix[3] + Hg, "Ignore" -> NumberQ];
  FI = Extract[F, {{1, 1}, {1, 2}, {1, 3}, {2, 1}, {2, 2}, {2, 3}, {3, 1}, {3, 2}, {3, 3}}];
  Igd ⊨ SMSInteger[(Ig - 1) lgd];
  DPDF ⊨ Table[SMSIO["Element data"[Igd + NoStressComponents + (j - 1) NoStressComponents + i]],
    {i, NoStressComponents}, {j, NoKinComponents}];
  PI ⊨ Table[SMSIO["Element data"[Igd + i]], {i, NoStressComponents}];
  SMSDefineDerivative[PI, FI, DPDF];
  P ⊨ {{PI⟦1⟧, PI⟦2⟧, PI⟦3⟧}, {PI⟦4⟧, PI⟦5⟧, PI⟦6⟧}, {PI⟦7⟧, PI⟦8⟧, PI⟦9⟧}};
  {ρ0, bX, bY, bZ} ⊨ SMSIO["All domain data"];
  bb ⊨ {bX, bY, bZ};
  W ⊨ Tr[P.Transpose[F]];
  wgp ⊨ SMSIO["Integration weight"[Ig]];
 )
```

"Tangent and residual" user subroutine

In[288]:=
```
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
SMSDo[
  Rg ⊨ Jed SMSD[W - ρ0 u.bb, pe, i, "Constant" -> PI];
  SMSIO[wgp Rg, "Add to", "Residual"[i]];
  SMSDo[
   Kg ⊨ SMSD[Rg, pe, j];
   SMSIO[wgp Kg, "Add to", "Tangent"[i, j]];
   , {j, i, SMSNoAllDOF}];
  , {i, 1, SMSNoAllDOF}];
SMSEndDo[];
```

"Postprocessing" user subroutine

In[293]:=
```
SMSStandardModule["Postprocessing"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
SMSIO[{"DeformedMeshX" -> uIO[[All, 1]],
    "DeformedMeshY" -> uIO[[All, 2]], "DeformedMeshZ" -> uIO[[All, 3]], "u" -> uIO[[All, 1]],
    "v" -> uIO[[All, 2]], "w" -> uIO[[All, 3]}, "Export to", "Nodal point post"];
Eg = 1 / 2 (Transpose[F].F - IdentityMatrix[3]);
σ = P.Transpose[F] / Det[F];
s = σ - 1/3 Tr[σ] × IdentityMatrix[3]; MisesPost = SMSSqrt[3/2 Tr[s.s]];
SMSIO[{"Exx" -> Eg[[1, 1]], "Exy" -> Eg[[1, 2]], "Exz" -> Eg[[1, 3]], "Eyx" -> Eg[[2, 1]],
    "Eyy" -> Eg[[2, 2]], "Eyz" -> Eg[[2, 3]], "Ezx" -> Eg[[3, 1]], "Ezy" -> Eg[[3, 2]], "Ezz" -> Eg[[3, 3]]
    , "Sxx" → σ[[1, 1]], "Sxy" → σ[[1, 2]], "Sxz" → σ[[1, 3]], "Syx" → σ[[2, 1]], "Syy" → σ[[2, 2]],
    "Syz" → σ[[2, 3]], "Szx" → σ[[3, 1]], "Szy" → σ[[3, 2]], "Szz" → σ[[3, 3]], "Mises stress" -> MisesPost},
  "Export to", "Integration point post"[Ig]];
SMSEndDo[];
```

"Tasks" user subroutine

- 1 "FE^2" initialization data
  - 1 - number of micro problems associated with the element
  - 2 - index of character switch of macro element that defines the name of the task that returns multi-scale related macro data
  - 3 - length of multi-scale related macro data
  - 4 - index of character switch of macro element that defines the name of the micro problem task that returns multi-scale related micro data
  - 5 - length of multi-scale related micro data
- 2 "Material points"
  - Join$[X_1, X_2, \ldots, X_{ng}]$
- 3 "Deformation gradient (plane strain)"
  - vec (F) = $\{F_{11}, F_{12}, F_{13}, F_{21}, F_{22,}, F_{23}, F_{31}, F_{32}, F_{33}\}$

In[302]:=
```
SMSStandardModule["Tasks"];
ng = SMSIO["No. integration points"];
task = SMSIO["Task index"];
SMSIf[task < 0
  , SMSSwitch[task
   , -1,
   SMSIO[{1, 0, 0, 5, 0}, "Export to", "Task data"];
   , -2,
   SMSIO[{1, 0, 0, 0, SMSNoDimensions * ng}, "Export to", "Task data"];
   , -3,
   SMSIO[{1, 0, 0, 0, NoKinComponents * ng}, "Export to", "Task data"];
  ];
  SMSReturn[];
 ];
SMSIf[task == 1
  , SMSIO[{ng, 3, NoKinComponents, 4, lgd}, "Export to", "Task integer output"];
 ];
SMSDo[
  ElementDefinitions[];
  SMSSwitch[task
   , 2,
   SMSIO[X[[{1, 2, 3}]], "Export to",
     "Task real output"[Table[(Ig - 1) * SMSNoDimensions + i, {i, SMSNoDimensions}]]];
   , 3,
   SMSIO[FI, "Export to",
     "Task real output"[Table[(Ig - 1) NoKinComponents + i, {i, NoKinComponents}]]];
  ];
  , {Ig, 1, ng}
 ];
```

Code generation

In[308]:=
```
SMSWrite[];
SMTMakeDll[];
```

File: ExamplesH1PFMacroSymm.c Size: 39 272 Time: 9

Method	SKR	SPP	Tasks
No.Formulae	304	228	219
No.Leafs	5392	3796	3124

Macro element for 3D solid FE^2 analysis - H1 - P/F stress/strain pair - unsymmetric tangent

Description

Generate three-dimensional, eight node macro finite element for the analysis of the micro-macro problems. The element has the following characteristics:

⇒ 3D hexahedral element,

⇒ 8 node element,

⇒ isoparametric mapping from the reference to the actual frame,

⇒ global unknowns are displacements of the nodes,

⇒ the element should allow arbitrary large displacements and rotations,

⇒ the problem is defined by the deformation gradient $F = I + \nabla u$ and the first Piola-Kirchoff stress tensor P and pseudo potential

$$\Pi = \int_{\Omega_0} \mathbf{P} : \mathbf{F}\, d\Omega_0$$

Ω_0 is the initial domain of the problem.

⇒ ADB form of the problem leads to the element contribution to the governing equations of the problem in a form:

$$\mathbf{R}_e = \frac{\hat{\delta}\Pi}{\hat{\delta}\mathbf{p}_e} \Big| \frac{\mathrm{D}\mathbf{P}}{\mathrm{D}\mathbf{p}_e} = 0$$

AceGen input for macro element

Initialization

In[310]:=
```
Get["AceGen`"];
NoStressComponents = 9; NoKinComponents = 9;
lgd = (NoStressComponents + NoStressComponents NoKinComponents);
Led = lgd es$$["id", "NoIntPoints"];
```

In[314]:=
```
SMSInitialize["ExamplesH1PFMacroUnsymm", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "H1"
  , "SMSSymmetricTangent" → False
  , "SMSDomainDataNames" -> {"ρ0 -density", "bX -force per unit mass X",
    "bY -force per unit mass Y", "bZ -force per unit mass Z"}
  , "SMSDefaultData" -> {1, 0, 0, 0}
  , "SMSNoElementData" → Led
  , "SMSCharSwitch" → {"FE^2", "Material points",
    "Deformation gradient (3D)", "Integrated stress and sensitivity (P, 3D)"}
 ];
```

Definitions of geometry, kinematics, strain energy ...

- the "Data" field is organized as
- $\mathrm{Flatten}\left[\left\{\left\{\mathrm{vec}(P), \frac{\partial \mathrm{vec}(P)}{\partial \mathrm{vec}(F)_1}, \frac{\partial \mathrm{vec}(P)}{\partial \mathrm{vec}(F)_2}, \ldots\right\}, \ldots \text{for all intergation points}\right\}\right]$ where
- $\mathrm{vec}(P) = \{P_{11}, P_{12}, P_{13}, P_{21}, P_{22}, P_{23}, P_{13}, P_{23}, P_{33}\}$ and $\mathrm{vec}(F) = \{F_{11}, F_{12}, F_{13}, F_{21}, F_{22,}, F_{23}, F_{31}, F_{32}, F_{33}\}$

In[316]:=
```
ElementDefinitions[] := (
  Ξ = {ξ, η, ζ} ⊨ SMSIO["Integration point"[Ig]];
  {XIO, uIO} ⊨ SMSIO["All coordinates and DOFs"];
  Ξn = {{-1, -1, -1}, {1, -1, -1}, {1, 1, -1}, {-1, 1, -1},
    {-1, -1, 1}, {1, -1, 1}, {1, 1, 1}, {-1, 1, 1}};
  Nh ⊨ Table[1 / 8 (1 + ξ Ξn⟦i, 1⟧) (1 + η Ξn⟦i, 2⟧) (1 + ζ Ξn⟦i, 3⟧), {i, 1, 8}];
  SMSFreeze[X, Nh.XIO];
  Je ⊨ SMSD[X, Ξ]; Jed ⊨ Det[Je];
  pe = Flatten[uIO]; u ⊨ Nh.uIO;
  Hg ⊨ SMSD[u, X, "Dependency" → {Ξ, X, SMSInverse[Je]}];
  SMSFreeze[F, IdentityMatrix[3] + Hg, "Ignore" -> NumberQ];
  FI = Extract[F, {{1, 1}, {1, 2}, {1, 3}, {2, 1}, {2, 2}, {2, 3}, {3, 1}, {3, 2}, {3, 3}}];
  Igd ⊨ SMSInteger[(Ig - 1) lgd];
  DPDF ⊨ Table[SMSIO["Element data"[Igd + NoStressComponents + (j - 1) NoStressComponents + i]],
    {i, NoStressComponents}, {j, NoKinComponents}];
  PI ⊨ Table[SMSIO["Element data"[Igd + i]], {i, NoStressComponents}];
  SMSDefineDerivative[PI, FI, DPDF];
  P ⊨ {{PI⟦1⟧, PI⟦2⟧, PI⟦3⟧}, {PI⟦4⟧, PI⟦5⟧, PI⟦6⟧}, {PI⟦7⟧, PI⟦8⟧, PI⟦9⟧}};
  {ρ0, bX, bY, bZ} ⊨ SMSIO["All domain data"];
  bb ⊨ {bX, bY, bZ};
  W ⊨ Tr[P.Transpose[F]];
  wgp ⊨ SMSIO["Integration weight"[Ig]];
 )
```

"Tangent and residual" user subroutine

In[317]:=
```
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
SMSDo[
  Rg ⊨ Jed SMSD[W - ρ0 u.bb, pe, i, "Constant" -> PI];
  SMSIO[wgp Rg, "Add to", "Residual"[i]];
  SMSDo[
   Kg ⊨ SMSD[Rg, pe, j];
   SMSIO[wgp Kg, "Add to", "Tangent"[i, j]];
   , {j, 1, SMSNoDOFGlobal}];
  , {i, 1, SMSNoDOFGlobal}];
SMSEndDo[];
```

"Postprocessing" user subroutine

In[322]:=
```
SMSStandardModule["Postprocessing"];
SMSDo[Ig, 1, SMSIO["No. integration points"]];
ElementDefinitions[];
SMSIO[{"DeformedMeshX" -> uIO[[All, 1]],
    "DeformedMeshY" -> uIO[[All, 2]], "DeformedMeshZ" -> uIO[[All, 3]], "u" -> uIO[[All, 1]],
    "v" -> uIO[[All, 2]], "w" -> uIO[[All, 3]]}, "Export to", "Nodal point post"];
Eg := 1 / 2 (Transpose[F].F - IdentityMatrix[3]);
σ := P.Transpose[F] / Det[F];
s := σ - 1/3 Tr[σ] × IdentityMatrix[3]; MisesPost = SMSSqrt[3/2 Tr[s.s]];
SMSIO[{"Exx" -> Eg[[1, 1]], "Exy" -> Eg[[1, 2]], "Exz" -> Eg[[1, 3]], "Eyx" -> Eg[[2, 1]],
    "Eyy" -> Eg[[2, 2]], "Eyz" -> Eg[[2, 3]], "Ezx" -> Eg[[3, 1]], "Ezy" -> Eg[[3, 2]], "Ezz" -> Eg[[3, 3]]
    , "Sxx" → σ[[1, 1]], "Sxy" → σ[[1, 2]], "Sxz" → σ[[1, 3]], "Syx" → σ[[2, 1]], "Syy" → σ[[2, 2]],
    "Syz" → σ[[2, 3]], "Szx" → σ[[3, 1]], "Szy" → σ[[3, 2]], "Szz" → σ[[3, 3]], "Mises stress" -> MisesPost},
  "Export to", "Integration point post"[Ig]];
SMSEndDo[];
```

"Tasks" user subroutine

- 1 "FE^2" initialization data
  - 1 - number of micro problems associated with the element
  - 2 - index of character switch of macro element that defines the name of the task that returns multi-scale related macro data
  - 3 - length of multi-scale related macro data
  - 4 - index of character switch of macro element that defines the name of the micro problem task that returns multi-scale related micro data
  - 5 - length of multi-scale related micro data
- 2 "Material points"
  - Join$[X_1, X_2, \ldots, X_{ng}]$
- 3 "Deformation gradient (plane strain)"
  - vec (F) = $\{F_{11}, F_{12}, F_{13}, F_{21}, F_{22,}, F_{23}, F_{31}, F_{32}, F_{33}\}$

In[331]:=
```
SMSStandardModule["Tasks"];
ng = SMSIO["No. integration points"];
task = SMSIO["Task index"];
SMSIf[task < 0
  , SMSSwitch[task
   , -1,
   SMSIO[{1, 0, 0, 5, 0}, "Export to", "Task data"];
   , -2,
   SMSIO[{1, 0, 0, 0, SMSNoDimensions * ng}, "Export to", "Task data"];
   , -3,
   SMSIO[{1, 0, 0, 0, NoKinComponents * ng}, "Export to", "Task data"];
  ];
  SMSReturn[];
 ];
SMSIf[task == 1
  , SMSIO[{ng, 3, NoKinComponents, 4, lgd}, "Export to", "Task integer output"];
 ];
SMSDo[
  ElementDefinitions[];
  SMSSwitch[task
   , 2,
   SMSIO[X[[{1, 2, 3}]], "Export to",
     "Task real output"[Table[(Ig - 1) * SMSNoDimensions + i, {i, SMSNoDimensions}]]];
   , 3,
   SMSIO[FI, "Export to",
     "Task real output"[Table[(Ig - 1) NoKinComponents + i, {i, NoKinComponents}]]];
  ];
  , {Ig, 1, ng}
 ];
```

Code generation

In[337]:=
```
SMSWrite[];
SMTMakeDll[];
```

```
File: ExamplesH1PFMacroUnsymm.c  Size: 39 271  Time: 9
```

Method	SKR	SPP	Tasks
No.Formulae	304	228	219
No.Leafs	5392	3796	3124

Macro element for 3D MIEL analysis - H1 - symmetric tangent

Description

Generate three-dimensional, eight node macro finite element for the analysis of the MIEL formulation of multi-scale formulation. The element has the following characteristics:

- 3D hexahedral element,
- 8 node element,
- global unknowns are displacements of the nodes,
- elements expects components of the residual and the symmetric tangent matrix to be available in element data field "Data" , thus micro_data = $\int_{\Omega_e} \text{Flatten}\left[\left\{W, \frac{\partial W}{\partial u_M} \mid_{\mathbf{h}=\text{const.}}, \text{vec}\left(\frac{\partial}{\partial u_M}\left(\frac{\partial W}{\partial u_M} \mid_{\mathbf{h}=\text{const.}}\right)\right)\right\}\right] dV$. u_M are degrees of freedom used to parameterize prescribed essential boundary conditions of the macro problem. The term $\int_{\Omega_e} \frac{\partial W}{\partial u_M} \mid_{\mathbf{h}=\text{const.}} dV$ represents contribution of the micro

element to macro residual and $\int_{\Omega_e} \text{vec}\left(\frac{\partial}{\partial u_M}\left(\frac{\partial W}{\partial u_M} \mid_{\mathbf{h}=\text{const.}}\right)\right) dV$ contribution of the micro element to condensed tangent matrix for MIEL multi-scale simulations. If second order directional derivatives $\text{vec}\left(\frac{\partial}{\partial u_M}\left(\frac{\partial W}{\partial u_M} \mid_{\mathbf{h}=\text{const.}}\right)\right)$ are symmetric then only upper triangular matrix is calculated and stored.

- Tangent matrix has $n(n+1)/2$ components that are stored in data field "Data" by rows as follows:
$\{K_{11}, K_{12}, \ldots, K_{1n}, K_{22}, \ldots, K_{2n}, \ldots, K_{nn}\}$

AceGen input for MIEL macro element

In[339]:=
```
Get["AceGen`"];
```

In[340]:=
```
SMSInitialize["ExamplesH1MIELMacroSymm", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "H1"
  , "SMSSymmetricTangent" → True
  , "SMSCharSwitch" →
   {"MIEL", "Nodal displacements (3D)", "Integrated strain energy and derivatives (3D)"}];
```

In[342]:=
```
SMSNoElementData = 1 + SMSNoDOFGlobal + SMSNoDOFGlobal (SMSNoDOFGlobal + 1) / 2
```

Out[342]=
```
325
```

"Tangent and residual" user subroutine

In[343]:=
```
SMSStandardModule["Tangent and residual"];
```

In[344]:=
```
row = SMSInteger[SMSNoDOFGlobal + 2];
SMSDo[
  SMSIO[SMSIO["Element data"[1 + i]], "Export to", "Residual"[i]];
  SMSDo[
   SMSIO[SMSIO["Element data"[row + j - i]], "Export to", "Tangent"[i, j]];
   , {j, i, SMSNoDOFGlobal}];
  row = row + SMSNoDOFGlobal - i + 1;
  , {i, 1, SMSNoDOFGlobal, 1, row}];
```

"Postprocessing" user subroutine

In[346]:=
```
SMSStandardModule["Postprocessing"];
uIO = SMSIO["All DOFs"];
SMSIO[{"DeformedMeshX" -> uIO[[All, 1]],
    "DeformedMeshY" -> uIO[[All, 2]], "DeformedMeshZ" -> uIO[[All, 3]], "u" -> uIO[[All, 1]],
    "v" -> uIO[[All, 2]], "w" -> uIO[[All, 3]]}, "Export to", "Nodal point post"];
```

"Tasks" user subroutine

- 1 "MIEL" initialization data
  - 1 - number of micro problems associated with the element
  - 2 - index of character switch of macro element that defines the name of the task that returns multi-scale related macro data
  - 3 - length of multi-scale related macro data
  - 4 - index of character switch of macro element that defines the name of the micro problem task that returns multi-scale related micro data
  - 5 - length of multi-scale related micro data
- 2 "Nodal displacements (3D)" returns Flatten[ph]

In[349]:=
```
SMSStandardModule["Tasks"];
task = SMSIO["Task index"];
SMSIf[task < 0
  , SMSSwitch[task
   , -1,
   SMSIO[{1, 0, 0, 5, 0}, "Export to", "Task data"];
   , -2,
   SMSIO[{1, 0, 0, 0, SMSNoDOFGlobal}, "Export to", "Task data"];
  ];
  SMSReturn[];
 ];
SMSSwitch[task
  , 1,
  SMSIO[{1, 2, SMSNoDOFGlobal, 3, SMSNoElementData}, "Export to", "Task integer output"];
  , 2,
  SMSIO[SMSIO["All DOFs"] // Flatten,
    "Export to", "Task real output"[Table[i, {i, SMSNoDOFGlobal}]]];
 ];
```

In[353]:=
```
SMSWrite[];
```

File: ExamplesH1MIELMacroSymm.c Size: 8308 Time: 1

Method	SKR	SPP	Tasks
No.Formulae	5	25	11
No.Leafs	71	602	472

In[354]:=
```
SMTMakeDll[];
```

Macro element for 3D MIEL analysis - H1 - unsymmetric tangent

Description

Generate three-dimensional, eight node macro finite element for the analysis of the MIEL formulation of multi-scale formulation. The element has the following characteristics:

- 3D hexahedral element,
- 8 node element,
- global unknowns are displacements of the nodes,
- elements expects components of the residual and the symmetric tangent matrix to be available in element data field "Data" , thus micro_data = $\int_{\Omega_e}$ Flatten$\left[\left\{W, \frac{\partial W}{\partial u_M}\,|_{\mathbf{h}=\text{const.}}, \text{vec}\left(\frac{\partial}{\partial u_M}\left(\frac{\partial W}{\partial u_M}\,|_{\mathbf{h}=\text{const.}}\right)\right)\right\}\right]$ dV. u_M are degrees of freedom used to parameterize prescribed essential boundary conditions of the macro problem. The term $\int_{\Omega_e}\frac{\partial W}{\partial u_M}\,|_{\mathbf{h}=\text{const.}}$ dV represents contribution of the micro element to macro residual and $\int_{\Omega_e}\text{vec}\left(\frac{\partial}{\partial u_M}\left(\frac{\partial W}{\partial u_M}\,|_{\mathbf{h}=\text{const.}}\right)\right)$ dV contribution of the micro element to condensed tangent matrix for MIEL multi-scale simulations. If second order directional derivatives $\text{vec}\left(\frac{\partial}{\partial u_M}\left(\frac{\partial W}{\partial u_M}\,|_{\mathbf{h}=\text{const.}}\right)\right)$ are symmetric then only upper triangular matrix is calculated and stored.
- Tangent matrix has $n(n+1)/2$ components that are stored in data field "Data" by rows as follows:
  $\{K_{11}, K_{12}, \ldots, K_{1n}, K_{21}, \ldots, K_{2n}, \ldots, K_{nn}\}$

AceGen input for MIEL macro element

In[355]:=
```
Get["AceGen`"];
```

In[356]:=
```
SMSInitialize["ExamplesH1MIELMacroUnsymm", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "H1"
  , "SMSSymmetricTangent" → False
  , "SMSCharSwitch" →
   {"MIEL", "Nodal displacements (3D)", "Integrated strain energy and derivatives (3D)"}];
SMSNoElementData = 1 + SMSNoDOFGlobal + SMSNoDOFGlobal SMSNoDOFGlobal;
```

"Tangent and residual" user subroutine

In[359]:=
```
SMSStandardModule["Tangent and residual"];
row = SMSInteger[SMSNoDOFGlobal + 2];
SMSDo[
  SMSIO[SMSIO["Element data"[1 + i]], "Export to", "Residual"[i]];
  SMSDo[
   SMSIO[SMSIO["Element data"[row + j - 1]], "Export to", "Tangent"[i, j]];
   , {j, 1, SMSNoDOFGlobal}];
  row = row + SMSNoDOFGlobal;
  , {i, 1, SMSNoDOFGlobal, 1, row}];
```

"Postprocessing" user subroutine

In[362]:=
```
SMSStandardModule["Postprocessing"];
uIO = SMSIO["All DOFs"];
SMSIO[{"DeformedMeshX" -> uIO[[All, 1]],
    "DeformedMeshY" -> uIO[[All, 2]], "DeformedMeshZ" -> uIO[[All, 3]], "u" -> uIO[[All, 1]],
    "v" -> uIO[[All, 2]], "w" -> uIO[[All, 3]]}, "Export to", "Nodal point post"];
```

"Tasks" user subroutine

- 1 "MIEL" initialization data
  - 1 - number of micro problems associated with the element
  - 2 - index of character switch of macro element that defines the name of the task that returns multi-scale related macro data
  - 3 - length of multi-scale related macro data
  - 4 - index of character switch of macro element that defines the name of the micro problem task that returns multi-scale related micro data
  - 5 - length of multi-scale related micro data
- 2 "Nodal displacements (3D)" returns Flatten[ph]

In[365]:=
```
SMSStandardModule["Tasks"];
task = SMSIO["Task index"];
SMSIf[task < 0
  , SMSSwitch[task
   , -1,
   SMSIO[{1, 0, 0, 5, 0}, "Export to", "Task data"];
   , -2,
   SMSIO[{1, 0, 0, 0, SMSNoDOFGlobal}, "Export to", "Task data"];
  ];
  SMSReturn[];
 ];
SMSSwitch[task
  , 1,
  SMSIO[{1, 2, SMSNoDOFGlobal, 3, SMSNoElementData}, "Export to", "Task integer output"];
  , 2,
  SMSIO[SMSIO["All DOFs"] // Flatten,
    "Export to", "Task real output"[Table[i, {i, SMSNoDOFGlobal}]]];
 ];
```

In[369]:= **SMSWrite[];**

File: ExamplesH1MIELMacroUnsymm.c **Size:** 8290 **Time:** 1

Method	SKR	SPP	Tasks
No.Formulae	4	25	11
No.Leafs	63	602	472

In[370]:= **SMTMakeDll[];**

Multi-scale Computational Environment

In[1]:=
```
<< AceFEM`;
```

Problem independent utility functions

Global variables

In[2]:=
```
SMTMultiScaleLoaded = True;
SMTSharedStatus;
SMTMacroProblemStatus;
SMTMultiScaleTask;
SMTCurrentLocalProblem;
SMTMacroElements;
SMTAllLocalProblems;
SMTCriticalSectionA;
SMTUpdateSharedStatus;
```

SMTMultiScaleSet

In[11]:=
```
Options[SMTMultiScaleSet] =
  {"PartialRestart" → False, "MaxKernels" → Automatic, "Threads" → 1, "Console" → False,
   "PartialDump" → True, "SpecificMicroData" → {}, "FE^2" → {}, "MIEL" → {},
   "TestSingleScale" → False, "Debug" → False, "CompressionLevel" → 2, "RestartKernels" → True
  };
```

In[12]:=
```
SMTMultiScaleSet[opt__Rule] :=
 Module[{dir, q1, p1, p2, p3, p4, p5, p6, p7, pal, microprofiles, time},
  SMCCheckOptions[{SMTMultiScaleSet}, "SMTMultiScaleSet",
  {"Debug" → True | _Integer | False | {_Integer, _Function}, "PartialRestart" → True | False,
    "PartialDump" → True | False, "MaxKernels" → Automatic | _Integer,
    "SpecificMicroData" → {__Rule}, "FE^2" → {___Rule}, "MIEL" → {___Rule},
    "TestSingleScale" → True | False, "CompressionLevel" → _Integer, "Threads" → _Integer,
    "Console" → True | False, "RestartKernels" → True | False}, opt];

  q1 = Replace[{
      "TestSingleScale", "Debug", "PartialRestart", "MaxKernels", "PartialDump"
      , "SpecificMicroData", "FE^2", "CompressionLevel", "MIEL", "Threads"
      , "Console", "RestartKernels"}, Join[{opt}, Options[SMTMultiScaleSet]], {1}];
  SMCProgressBar["Action" -> "", "Report" → "", "Progress" → ""];
  SMTMultiScaleTask = "Initialization";
  If[ValueQ[SMTMacroProblemStatus], NotebookClose[SMTMacroProblemStatus[[5]]]];
  SMTMacroProblemStatus = {SMTSpatialDimension, {"FE^2", "MIEL"}, q1[[3]], Null ,
    Null, q1[[4]], q1[[5]], AbsoluteTime[], Directory[], 10, q1[[6]], 0, 0
    , q1[[2]] /. {True → -1, False → 0, {i_, _} :↦ i}
    , q1[[8]]
    , Switch[q1[[2]]
     , False, Function[{}, Null]
     , {_, _}, q1[[2, 2]]
     , _, Function[{localdata, conv, mess},
      If[SMTMacroProblemStatus[[14]] == -1 || SMTMacroProblemStatus[[14]] == localdata[[4]]
       , If[conv
        , SMTStatusReport[Row[{"Micro problem:", localdata[[4]], " ",
            SMTShowMesh["DeformedMesh" → True, ImageSize → 100], " ,", mess}]]
        , SMTStatusReport[Row[{"Micro problem:", localdata[[4]], " ,", mess}]]
       ]
      ]
```

```
    ]
   ]
   , q1[[10]], q1[[11]]
   , 0, 0, 0};
SMTCurrentLocalProblem = Null;
SMTMacroElements = {{}, {}, {}};
SMTAllLocalProblems = {};

If[q1[[1]]
 , SMTMicroProfile = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, {0, 0}, False};
 Return[True, Module]
];

Check[
 If[q1[[12]] || (Not[Kernels[] =!= {} && SMTMacroProblemStatus[[6]] === $KernelCount])
  , If[Kernels[] =!= {}
   , CloseKernels[];
  ];
  If[SMTMacroProblemStatus[[6]] === Automatic
   , LaunchKernels[];
   SMTMacroProblemStatus[[6]] = $KernelCount;
   , LaunchKernels[SMTMacroProblemStatus[[6]]];
  ];
 ];
 SMTMacroProblemStatus[[10]] = ParallelEvaluate[$KernelID];
 , SMCError = {"Kernels: ", SMTMacroProblemStatus[[6]]};
 SMCAbort["Cannot open parallel kernels."];
];
With[{dir0 = Directory[]}, ParallelEvaluate[SetDirectory[dir0]]];
(*lunch HDF5.exe not in parallel - bug in MMA*)
Do[
 ParallelEvaluate[
   Export[
     "tmpHDF5TestExport.h5",
     Table[RandomInteger[], {1000}] // Compress, {"Datasets", "Mathematica definitions"}
    ];
   , i
   , DistributedContexts → None];
 , {i, Kernels[]}];
DeleteFile["tmpHDF5TestExport.h5"];

Check[
 p1 = FileNameJoin[{SMTMacroProblemStatus[[9]], "tmpMS_MicroData"}];
 If[FileType[p1] === Directory, DeleteDirectory[p1, DeleteContents → True];];
 DeleteFile[FileNames["tmpMS_*"]];
 CreateDirectory[p1];
 , SMCError = {"Directory: ", p1};
 SMCAbort["Cannot delete old simulation files."];
];

With[
 {p1 = SMCSession[[37]], p2 = SMTMacroProblemStatus[[17]], p3 = SMTMacroProblemStatus[[18]]},
 ParallelEvaluate[
  Get["AceFEM`Remote`"];
```

```
  (*set AceShare directory to be the same for macro and micro problems*)
  CriticalSection[{SMTCriticalSectionA}
   , SMCSetAceShare[p1];
   SMTInputData["Threads" → p2, "Console" → p3];
  ];
  SMTMicroProfile = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, {0, 0}, False};
  , DistributedContexts → None]
];

DistributeDefinitions[SMTMicroSet, SMTMicroRestart, SMTMicroSolve,
 SMTMicroEvaluate, SMTMultiScalePostData, SMTMultiScaleSimulationReport,
 SMTMultiScaleStatusReport, SMTToSymmetricOrUnsymmetric, SMTUpdateSharedStatus];
SMTUpdateSharedStatus["Reset"];

(*start absolute time - free from kernels and AceFEM lunch time - only simulation time *)
SMTMacroProblemStatus[[8]] = AbsoluteTime[];

(* FE^2 *)
If[q1[[7]] =!= {},
 DistributeDefinitions @@ Flatten[q1[[7, All, 1]]];
 p3 = 0;
 Check[
  Scan[(
       If[Not[MatchQ[#, {_Symbol, _Symbol} → {_Integer ...}]]
        , SMCError = {"FE^2 elements: ", #};
        SMCAbort["Incorrect definition of FE^2 elements."];
       ];
       p4 = #[[1]]; (*micro mesh and solve module*)
       p7 = #[[2]];
       (*randomly reshuffle the elements p7=
         p7[[RandomSample[Range[p7//Length],p7//Length]]];*)
       p1 = MapThread[(
            p2 = #1; p6 = SMTDomainData[#2, "CharSwitch"];
            If[FreeQ[p6, "Material points"]
             , SMCError = {"FE^2 element: ", p2, " - ", SMTDomainData[#2, "Code"]};
             SMCAbort["Missing material points task."];
            ];
            p1 = SMTTask["Material points", "Elements" → p2];
            If[OddQ[Length[p1]]
             , SMCError = {"FE^2 element: ", p2, " - ", SMTDomainData[#2, "Code"], "\n", p1};
             SMCAbort["Vector must be a list of spatial points."];
            ];
            If[FreeQ[p6, "FE^2"]
             , SMCError = {"FE^2 element: ", p2, " - ", SMTDomainData[#2, "Code"]};
             SMCAbort["Missing FE^2 task."];
            ];
            p5 = SMTTask["FE^2", "Elements" → p2];
            If[Length[p5] =!= 5 || p5[[2]] > Length[p6] || p5[[4]] > Length[p6]
             , SMCError = {"FE^2 element: ", p2, " - ", SMTDomainData[#2, "Code"], "\n", p5};
             SMCAbort["Error in FE^2 task"];
            ];
            p5[[{2, 4}]] = p6[[p5[[{2, 4}]]]];
            Map[{Join[{"FE^2"}, p4, p5], #1, 3, ++p3, 5, p2, ConstantArray[0, p5[[3]]], 8, 9,
                {}, Null} &, Partition[p1, SMTSpatialDimension]]
           ) &, {p7, SMTElementData[p7, "SpecIndex"]}];
```

```
      SMTAllLocalProblems = Join[SMTAllLocalProblems, Flatten[p1, 1]];
      SMTMacroElements[[1]] =
       Join[SMTMacroElements[[1]], MapThread[Rule, {p7, p1[[All, All, 4]]}]];
     ) &, q1[[7]]
   ];
  , SMCError = {"FE^2: ", q1[[7]]};
  SMCAbort["Incorrect definition of FE^2 micro problems."];
 ];
];

(* MIEL *)
If[q1[[9]] =!= {},
 DistributeDefinitions @@ Flatten[q1[[9, All, 1]]];
 Check[
  p3 = Length[SMTAllLocalProblems];
  Scan[(
     If[Not[MatchQ[#, {_Symbol, _Symbol} → {_Integer ...}]]
      , SMCError = {"FE^2 elements: ", #};
      SMCAbort["Incorrect definition of FE^2 elements."];
     ];
     p7 = #[[2]];
     (*randomly reshuffle the elements p7=
       p7[[RandomSample[Range[p7//Length],p7//Length]]];*)
     p1 = Map[SMTNodeData[#, "X"] &, SMTElementData[p7, "Nodes"]];
     p4 = #[[1]];
     p1 = MapThread[(
          p6 = SMTDomainData[#3, "CharSwitch"];
          If[FreeQ[p6, "MIEL"]
           , SMCError = {"MIEL element: ", #2, " - ", SMTDomainData[#3, "Code"]};
           SMCAbort["Missing MIEL task."];
          ];
          p5 = SMTTask["MIEL", "Elements" → #2];
          If[Length[p5] =!= 5  || p5[[4]] > Length[p6]
           , SMCError = {"MIEL element: ", #2, " - ", SMTDomainData[#3, "Code"], "\n", p5};
           SMCAbort["Error in MIEL task"];
          ];
          p5[[4]] = p6[[p5[[4]]]]; p5[[2]] = p6[[p5[[2]]]];
          {Join[{"MIEL"}, p4, p5], Total[#1] / Length[#1], 3, ++p3, 5,
           #2, ConstantArray[0, p5[[3]]],
           #1, SMTDomainData[#3, "Topology"], {#1} // Transpose, 1, {},
            If[SMTDomainData[#3, "SymmetricTangent"] == 1, True, False], Null}
         ) &, {p1, p7, SMTElementData[p7, "SpecIndex"]}];
     SMTAllLocalProblems = Join[SMTAllLocalProblems, p1];
     SMTMacroElements[[2]] =
      Join[SMTMacroElements[[2]], MapThread[Rule, {p7, p1[[All, {4}]]}]];
    ) &, q1[[9]]
  ];
  , SMCError = {"MIEL: ", q1[[9]]};
  SMCAbort["Incorrect definition of MIEL micro problems."];
 ];
];

If[SMTAllLocalProblems === {}
 , SMCError = {"At least one of the options FE^2, MIEL must be defined."};
 SMCAbort["Multi-scale anaysis not defined."];
```

```
];

SMTMacroProblemStatus[[4]] = Length[SMTAllLocalProblems];
SMTMacroElements[[3]] = Complement[Range[SMTIData["NoElements"]],
  SMTMacroElements[[1, All, 1]], SMTMacroElements[[2, All, 1]]];

p1 = {
  {"Number of macro elements", SMTIData["NoElements"]}
  , {"No. of FE^2 macro elements", Length[SMTMacroElements[[1]]]}
  , {"Number of MIEL macro elements", Length[SMTMacroElements[[2]]]}
  , {"Number of single-scale elements", Length[SMTMacroElements[[3]]]}
  , {"Number of kernels", SMTMacroProblemStatus[[6]]}
 };
pal = Column[{
    Grid[p1, Alignment → Left, Frame → {False, All},
     ItemStyle -> Directive[FontSize → 11], Spacings → {1, 0}]
    , Pane[Column[{Dynamic[SMTMultiScaleTask],
       Dynamic[Refresh[SMTUpdateSharedStatus["Report"], UpdateInterval → 3]]
      }], ImageSize -> {1.2 SMCSession[[22]], 15 SMCSession[[23]]}]
  }];
Map[PutAppend[#, "tmpMS_Monitor.txt"] &, p1];
MapIndexed[PutAppend[{"Kernel: ", #2[[1]], "$ProcessID: ", #}, "tmpMS_Monitor.txt"] &,
 ParallelEvaluate[$ProcessID]];
SMTMacroProblemStatus[[5]] = CreatePalette[pal, WindowTitle → "# Multi-scale monitor",
  Saveable → False, WindowFloating → False];

DistributeDefinitions[SMTMacroProblemStatus];
Do[
 With[{p1 = i},
   ParallelEvaluate[
     SMTMicroProfile[[15]] = p1;
     , SMTMacroProblemStatus[[10, i]]
     , DistributedContexts → None
    ];
  ];
 , {i, 1, SMTMacroProblemStatus[[6]]}
];
(*set up all micro problems*)
SMTMacroProblemStatus[[13]] = AbsoluteTiming[SMTMicroSet[];][[1]];
SMTUpdateSharedStatus["Global"];

If[SMTSharedStatus[[1]] == 1 || SMTSharedStatus[[1]] == 2,
 Print[SMTSharedStatus[[2]]];
];
If[SMTSharedStatus[[1]] > 2
 , SMCError = SMTSharedStatus[[2]];
 SMCAbort["Error during initilization of micro problems."];
];
If[Not[FreeQ[SMTAllLocalProblems, $Aborted]]
 , SMCError = "Aborted micro problem.";
 SMCAbort["Error during initilization of micro problems."];
];
SMTMultiScaleTask = Style["Initialization done.", Green];
time = Row[{
```

```
     Floor[#/3600.], ":"
     , Floor[(# - 3600. Floor[#/3600.])/60.] , ":"
     , NumberForm[# - Floor[#/3600.] 3600 - 60 Floor[(# - 3600 Floor[#/3600.])/60], {4, 2}]
    }] &;
  microprofiles = ParallelEvaluate[SMTMicroProfile, DistributedContexts → None];
  SMTMacroProblemStatus[[5]] = CreatePalette[Column[{
     Grid[{
       {"Number of macro elements", SMTIData["NoElements"]}
       , {"Number of FE^2 macro elements", Length[SMTMacroElements[[1]]]}
       , {"No. MIEL macro elements", Length[SMTMacroElements[[2]]]}
       , {"Number of single-scale elements", Length[SMTMacroElements[[3]]]}
       , {"Number of kernels", SMTMacroProblemStatus[[6]]}
       , {"Total number of micro problems", Length[SMTAllLocalProblems]}
       , {"Total number of micro elements", Total[microprofiles[[All, 7]]]}
       , {"Total number of micro nodes", Total[microprofiles[[All, 8]]]}
       , {"Total number of micro equations", Total[microprofiles[[All, 9]]]}
       , {"Real micro mesh generation time", SMTMacroProblemStatus[[13]] // time}
      }, Alignment → Left, Frame → {False, All},
      ItemStyle -> Directive[FontSize → 11], Spacings → {1, 0}]
     , Pane[Column[{Dynamic[SMTMultiScaleTask],
        Dynamic[Refresh[SMTUpdateSharedStatus["Report"], UpdateInterval → 3]]
       }], ImageSize -> {1.2 SMCSession[[22]], 15 SMCSession[[23]]}]
     , Button["Refresh", Print[SMTUpdateSharedStatus["Report"]], Method -> "Queued"]
    }]
   , SMTMacroProblemStatus[[5]],
   WindowTitle → "# Multi-scale monitor", Saveable → False, WindowFloating → False];

  True
 ]
```

SMTUpdateSharedStatus

In[13]:=
```
SMTUpdateSharedStatus[local_, st_ : Null] := Module[{all, ff, gc},
 Switch[local

  (*at micro*)
  , "Local",
  Quiet[
   Check[
    gc = Get[FileNameJoin[{SMTMacroProblemStatus[[9]], "tmpMS_Kernel_All.txt"}]];
    , gc = {0, 0};]
  ];
  If[MatchQ[gc, {_?IntegerQ, _}],
   SMTMicroProfile[[19]] = gc;
   If[SMTMicroProfile[[19, 1]] < st[[1]]
    , SMTMicroProfile[[19]] = st;
    CriticalSection[{SMTCriticalSectionA}
     , Put[SMTMicroProfile[[19]],
      FileNameJoin[{SMTMacroProblemStatus[[9]], "tmpMS_Kernel_All.txt"}]];
     PutAppend["SharedStatus Local event: ", SMTMicroProfile[[19]], "tmpMS_Monitor.txt"];
    ];
   ];
   Put[st, FileNameJoin[{SMTMacroProblemStatus[[9]],
      "tmpMS_Kernel_" <> ToString[SMTMicroProfile[[15]]] <> ".txt"}]];
 ];
```

```
 (*at macro*)
 , "Global",
 Quiet[
  Check[
   SMTSharedStatus =
     Get[FileNameJoin[{SMTMacroProblemStatus[[9]], "tmpMS_Kernel_All.txt"}]];
   , SMTSharedStatus = {0, 0};]
 ];
 If[SMTSharedStatus[[1]] > 0
  , CriticalSection[{SMTCriticalSectionB}
    , PutAppend["SharedStatus Global event: ", SMTSharedStatus[[2]], "tmpMS_Monitor.txt"];
   ];
 ];

 , "Reset",
 Table[
  Put[{0, 0},
   FileNameJoin[{SMTMacroProblemStatus[[9]], "tmpMS_Kernel_" <> ToString[i] <> ".txt"}]],
  {i, SMTMacroProblemStatus[[6]]}];
 Put[{0, 0}, FileNameJoin[{SMTMacroProblemStatus[[9]], "tmpMS_Kernel_All.txt"}]];
 SMTSharedStatus = {0, 0};
 ParallelEvaluate[SMTMicroProfile[[19]] = {0, 0}, DistributedContexts → None];

 , "Report",
 all = Table[
   Get[FileNameJoin[{SMTMacroProblemStatus[[9]], "tmpMS_Kernel_" <> ToString[i] <> ".txt"}]],
    {i, SMTMacroProblemStatus[[6]]}];
 If[MatchQ[all, {{_?IntegerQ, _?IntegerQ} ...}]
  , Column[{
     Row[{"Total: ", Total[all[[All, 2]]],
       " Average: ", Round[Total[all[[All, 2]]] / SMTMacroProblemStatus[[6]]],
       " Time: ", Round[AbsoluteTime[] - SMTMacroProblemStatus[[8]], 0.1]}]
    , Row[{"Kernels: ", Row[all[[All, 2]], "■"]}]
   }]
  , Row[all, "■"]
 ]

 , "Debug",
 Do[
   Print["Kernel: ", i, " Status: ", Get[
     FileNameJoin[{SMTMacroProblemStatus[[9]], "tmpMS_Kernel_" <> ToString[i] <> ".txt"}]]];
   ff = FileNameJoin[{SMTMacroProblemStatus[[9]],
      "tmpMS_Kernel_Debug_" <> ToString[i] <> ".txt"}];
   If[FileExistsQ[ff], FilePrint[ff]];
   , {i, SMTMacroProblemStatus[[6]]}];
 ]
]
```

SMTMicroSet

In[14]:=
```
SMTMicroSet[] := Module[{},
 (*elements,nodes,equations*)
 SMTMultiScaleTask =
  "Initialization of " <> ToString[SMTMacroProblemStatus[[4]]] <> " micro problems.";
 PutAppend[SMTMultiScaleTask, "tmpMS_Monitor.txt"];
 SMTAllLocalProblems =
```

```
ParallelMap[
 Module[{LocalProblem, dumpfile, ret},
   LocalProblem = #[[4]];
   PutAppend[{"Ini start:", LocalProblem, $ProcessID}, "tmpMS_Monitor.txt"];
   SMTMicroProfile[[10]] ++;
   SMTUpdateSharedStatus["Local", {0, SMTMicroProfile[[10]]}];
   If[SMTMicroProfile[[19, 1]] == 3, Return[$Aborted, Module]];
   Check[
    (* call micro set subroutine *)
    ret = #[[1, 2]][#, SMTMacroProblemStatus[[11]]];
    , SMTUpdateSharedStatus["Local", {3, Row[{"Micro problem: ", #[[3]],
        "\nInitialization: ", #[[1, 2]], " returned:", ret}]}];
    Return[$Aborted, Module];
   ];
   PutAppend[{"Ini data", LocalProblem, $ProcessID, ret[[ ;; 10]]}, "tmpMS_Monitor.txt"];
   If[Head[ret] =!= List ||
     (#[[1, 1]] === "FE^2" && Not[MatchQ[ret,
         {{"FE^2", _Symbol, _Symbol, _Integer, _String, _Integer, _String, _Integer},
          _List, _, _Integer, _Integer, _Integer, _, _?(VectorQ[#, IntegerQ] &),
          _?NumberQ, _List, _}]]) ||
     (#[[1, 1]] === "MIEL" && Not[MatchQ[ret, {{"MIEL", _Symbol, _Symbol,
           _Integer, _String, _Integer, _String, _Integer},
          _List, _, _Integer, _Integer, _Integer, _, _?(MatrixQ[#, NumberQ] &),
          _String,
          _?(VectorQ[#, MatrixQ[#, NumberQ] &] &), _Integer, _List, True | False, _}]])
    , If[SMTMicroProfile[[19, 1]] =!= 3,
     SMTUpdateSharedStatus["Local", {3, Row[{"Micro problem: ", #[[3]],
          "\nInitialization routine: ", #[[1, 2]],
          " has returned incorrect local macro data structure:\n", ret}]}];
    ];
    Return[$Aborted, Module];
   ];
   SMTMicroProfile[[{7, 8, 9}]] += SMTIData[{"NoElements", "NoNodes", "NoEquations"}];
   dumpfile = FileNameJoin[
     {SMTMacroProblemStatus[[9]], "tmpMS_MicroData", "dump_" <> ToString[LocalProblem]}];
   SMTMicroProfile[[4]] ++;
   PutAppend[{"Ini dump", LocalProblem, $ProcessID,
     dumpfile, SMTMacroProblemStatus[[15]]}, "tmpMS_Monitor.txt"];
   SMTMicroProfile[[3]] += AbsoluteTiming[SMTDump[dumpfile, "Exclude" → {{}, {"da", "tmp",
         "st", "sp", "ADVF"}}, "CompressionLevel" -> SMTMacroProblemStatus[[15]]]][[1]];
   PutAppend[{"End dump", LocalProblem, $ProcessID}, "tmpMS_Monitor.txt"];
   (*check consistency of the micro and macro element pair*)
   If[FreeQ[SMTDomainData["CharSwitch"], #[[1, 7]]]
    , SMTUpdateSharedStatus["Local", {3, Row[{"\nMissing keyword:", #[[1, 7]]
       , "\nIncompatible macro and micro elements for multi-scale formulation."
       , "\nMacro MS formulation definitions: ", #[[1]],
       "\nMicro elements CharSwitch: ", SMTDomainData["CharSwitch"],
       "\nMicro elements: ", SMTDomains[[All, {1, 3}]] // TableForm
      }]}];
    Return[$Aborted, Module];
   ];
   ret
```

```
    ] &, SMTAllLocalProblems, DistributedContexts → None];
  PutAppend["SMTMicroSet - end", "tmpMS_Monitor.txt"];
  True
 ]
```

SMTMicroSolve

In[15]:=
```
Options[SMTMicroSolve] = {"Dump" → False};
SMTMicroSolve[opt___Rule] := Module[{retall, tr, task, dump, ii, locp},
  SMCCheckOptions[{SMTMicroSolve}, "SMTMicroSolve", {"Dump" → True | False}, opt];
  {dump} = {"Dump"} /. {opt} /. Options[SMTMicroSolve];
  SMTMacroProblemStatus[[21]] += 1;
  SMTMacroProblemStatus[[12]] += AbsoluteTiming[

    SMTMultiScaleTask =
     If[dump, "Solve and dump ", "Solve "] <> ToString[SMTMacroProblemStatus[[4]]] <>
      " micro problems (" <> ToString[SMTMacroProblemStatus[[21]]] <> "-th time).";
    SMTMacroProblemStatus[[20]] = SMTIData["Iteration"] + 1;
    DistributeDefinitions[SMTMacroProblemStatus];
    (* transfere macro data from macro elements to LocalProblems data structure *)
    Do[
     If[SMTMacroElements[[ii]] =!= {}
       , SMTAllLocalProblems[[Flatten[SMTMacroElements[[ii, All, 2]]], 7]] =
         Flatten[
          Map[(
              (*assume that all local problems of the same element are of the same type*)
              task = SMTAllLocalProblems[[#[[2, 1]], 1, 5]];
              tr = SMTTask[task, "Elements" → #[[1]]];
              If[Head[tr] =!= List
               , locp = SMTAllLocalProblems[[#[[2, 1]]]];
               SMCError = {
                 "Error during evaluation of macro element task:", task
                 , "\nMacro element:", #[[1]], " - ", SMTElementData[#[[1]], "Code"]
                 , "\nCorresponding local problem:", {locp[[1, 1]], "problem at Point",
                  locp[[2]], " Index:", locp[[4]], "Macro element:", locp[[6]]}
                };
               SMCAbort[
                "Error during solution of micro problem.", "SMTMicroSolve", "SMTMicroSolve"];
              ];
              Partition[tr, Length[tr] / Length[#[[2]]]]
             ) &, SMTMacroElements[[ii]]]
          , 1];
      ];
     , {ii, 2}];

    SMTUpdateSharedStatus["Reset"];
    With[{dd = dump}, ParallelEvaluate[SMTMicroProfile[[{10, 13, 14, 18, 19, 20}]] =
       {0, 0, 0, 0, {0, 0}, dd}, DistributedContexts → None]];

    retall = ParallelMap[
      Module[{RVEResponse, dumpfile, LocalProblem, retone, step, bstep, sr, p1, dumpp},
        Check[
         dumpp = SMTMicroProfile[[20]];
         LocalProblem = #[[4]];
         SMTMicroProfile[[10]] ++;
```

```
SMTUpdateSharedStatus["Local", {0, SMTMicroProfile[[10]]}];
If[SMTMicroProfile[[19, 1]] === 3, Return[$Aborted, Module];];
SMTMicroProfile[[6]] ++;
dumpfile = FileNameJoin[{SMTMacroProblemStatus[[9]],
   "tmpMS_MicroData", "dump_" <> ToString[LocalProblem]}];
SMTMicroProfile[[5]] ++;
SMTMicroProfile[[1]] += AbsoluteTiming[
   If[SMTMacroProblemStatus[[3]],
    SMTRestartState[dumpfile, "AccumulateStatistics" → "Driver"],
    SMTRestart[dumpfile, "Threads" → SMTMacroProblemStatus[[17]],
     "AccumulateStatistics" → "Driver", "Console" → SMTMacroProblemStatus[[18]]]]
  ][[1]];
(*solve one micro problem*)
step = SMTIData["Step"];
bstep = SMTIData["NoBackStep"];
SMTMicroProfile[[2]] += AbsoluteTiming[(retone = #[[1, 3]][#, dumpp]);][[1]];
If[retone === False
 , If[SMTMicroProfile[[19, 1]] === 0,
  SMTUpdateSharedStatus["Local",
    {3, {"Error during solution of micro problem: ", LocalProblem}}];
 ];
 Return[$Aborted, Module]
];
If[Length[retone] =!= #[[1, 8]],
 SMTUpdateSharedStatus["Local",
  {3, {"Micro problem: ", LocalProblem, "\nIncompatible micro and macro elements",
    "\nLength of data needed by macro:", #[[1, 8]],
    "\nMicro evaluation function:", #[[1, 3]],
    "\nLength of data provided by function:", Length[retone],
    "\n Data returned:", retone}}];
 Return[$Aborted, Module];
];
p1 = SMTIData["Step"] - step;
SMTMicroProfile[[13]] += SMTIData["NoDiscreteEvents"];
SMTMicroProfile[[14]] = Max[SMTMicroProfile[[14]], p1 - If[dumpp, 1, 0]];
SMTMicroProfile[[16]] += p1;
SMTMicroProfile[[17]] += SMTIData["NoBackStep"] - bstep;
SMTMicroProfile[[18]] += p1;
If[dumpp
 , SMTMicroProfile[[4]] ++;
 (*delete sensitivity history => relative sensitivity*)
 SMTTask["Reset sensitivity data"];
 SMTMicroProfile[[3]] += AbsoluteTiming[
    If[SMTMacroProblemStatus[[7]]
     , SMTDumpState[dumpfile, "Exclude" → {{}, {"da", "tmp", "st", "sp", "ADVF"}},
      "CompressionLevel" -> SMTMacroProblemStatus[[15]]]
     , SMTDump[dumpfile, "Exclude" → {{}, {"da", "tmp", "st", "sp", "ADVF"}},
      "CompressionLevel" -> SMTMacroProblemStatus[[15]]]
    ]
   ][[1]];
];
retone

, SMTUpdateSharedStatus["Local", {3,
```

```
            {"Unrecognized error during the solution of micro problem: ", LocalProblem}}];
           Return[$Aborted, Module];
          ]
         ] &, SMTAllLocalProblems, DistributedContexts → None]

    ][[1]]; (*timing*)

  SMTMultiScaleTask = Style["Micro problems solved.", Green];
  SMTUpdateSharedStatus["Global"];
  If[SMTSharedStatus[[1]] == 1 || SMTSharedStatus[[1]] == 2,
   If[SMTMacroProblemStatus[[14]] =!= 0, Print[SMTSharedStatus[[2]]]];
   Return[False, Module];
  ];

  If[SMTSharedStatus[[1]] > 2
   , SMCError = SMTSharedStatus[[2]];
   SMCAbort["Error during solution of micro problems.", "SMTMicroSolve", "SMTMicroSolve"];
  ];

  If[Not[MatchQ[retall, {{_?NumberQ ...} ...}]],
   SMCError = {"\nReturn values: ",
     Select[Flatten[retall], Not[NumberQ[#]] &, 10], "\nStatus: ", SMTSharedStatus};
   SMCAbort["Error during solution of micro problems. Return values are
       not a list of real numbers.", "SMTMicroSolve", "SMTMicroSolve"];
  ];
  (* transfere RVE response from micro to macro elements *)
  SMTElementData[
   Flatten[SMTMacroElements[[{1, 2}, All, 1]]]
   , "Data"
   , Flatten[Map[Flatten[retall[[#[[2]]]]] &, SMTMacroElements[[{1, 2}]], {2}], 1]
  ];
  True
 ]
```

SMTMicroRestart

In[17]:=
```
SMTMicroRestart[p_List] := (
  If[SMTSharedStatus[[1]] === 3, Abort[];];
  If[Not[MatchQ[p, ConstantArray[_?NumberQ, SMTMacroProblemStatus[[1]]]]]
   , SMCError = {"Input: ", p};
   SMCAbort["Input must be a list of NoSpatialDimensions numbers."];
  ];
  SMTMicroRestart[Nearest[SMTAllLocalProblems[[All, 2]] → Automatic, p][[1]]]
 )
```

In[18]:=
```
SMTMicroRestart[p_Integer] := (
  If[SMTSharedStatus[[1]] === 3, Abort[];];
  SMTCurrentLocalProblem = SMTAllLocalProblems[[p]];
  With[{i = SMTCurrentLocalProblem},
   ParallelEvaluate[
    SMTRestart[FileNameJoin[
       {SMTMacroProblemStatus[[9]], "tmpMS_MicroData", "dump_" <> ToString[i[[4]]]}]
     , "Threads" → SMTMacroProblemStatus[[17]], "Console" → SMTMacroProblemStatus[[18]]];
    SMTCurrentLocalProblem = i;
    , SMTMacroProblemStatus[[10, 1]], DistributedContexts → None]
  ];
  Take[SMTCurrentLocalProblem, 6]
 )
```

SMTMicroEvaluate[expression]

In[19]:=
```
SetAttributes[SMTMicroEvaluate, HoldFirst]
```

In[20]:=
```
SMTMicroEvaluate["Continue"] := SMTMicroEvaluate[SMCSession[[6]] = False;
  SMCAction = {}]
```

In[21]:=
```
SMTMicroEvaluate[c_] := If[SMTCurrentLocalProblem =!= Null
  , ParallelEvaluate[c, SMTMacroProblemStatus[[10, 1]], DistributedContexts → None]
 ]
```

SMTMultiScalePostData[code]

This is a generalized SMTPostData[field_code] command. It returns a nodal values of field with the given keyword in all nodes. The mesh can be composed of an arbitrary mixture of FE^2, mesoscale and ordinary elements.

In[22]:=
```
SMTMultiScalePostData[code_] := Module[{field, nodes, nfield, post, p1},
  field = Table[{0, 0}, SMTIData["NoNodes"]];
  Scan[(nodes = SMTElementData[#[[6]], "Nodes"];
     Switch[#[[1, 1]]
      , "FE^2",
      nfield = SMTPostData[code][[nodes]];
      field[[nodes, 1]] += nfield;
      field[[nodes, 2]] += 1;
      , "MIEL",
      SMTMicroRestart[#[[2]]];
      nfield = SMTMicroEvaluate[SMTPostData[code, Point[Flatten[#[[10]], 1]]]];
      p1 = 0;
      MapThread[(
         Do[
           p1++;
           field[[#2, 1]] += nfield[[p1]];
           field[[#2, 2]] += 1;
           , Length[#1]];
        ) &, {#[[10]], nodes}];
     ];
    ) &, SMTAllLocalProblems];
  Scan[
   (nodes = SMTElementData[#, "Nodes"];
     nfield = SMTPostData[code][[nodes]];
     field[[nodes, 1]] += nfield;
     field[[nodes, 2]] += 1;
    ) &, Complement[Range[SMTNoElements], SMTAllLocalProblems[[All, 6]]]
  ];
  post = Map[#[[1]] / #[[2]] &, field];
  post
 ];
```

SMTMultiScaleSimulationReport

In[23]:=
```
SMTMultiScaleSimulationReport[tab_?(# === True || # === False &), rm___] := Module[
  {microprofiles, avt, rep, time, nms, rdata, everage, macro},
  microprofiles = ParallelEvaluate[SMTMicroProfile, DistributedContexts → None];
  rdata = ParallelEvaluate[
    If[SMTMicroProfile[[10]] > 0
     , Join[
      SMTRData[{"KAndRTime", "SolverTime", "SensRTime", "SensSolTime"}], {SMTSessionTime[]}]
     , {0, 0, 0, 0, 0}
    ]
    , DistributedContexts → None];
  macro = SMTSimulationReport[False];
  avt = SMTMacroProblemStatus[[12]] + SMTMacroProblemStatus[[13]] +
    Total[{"Total linear solver time (s)", "Total K&R time (s)"} /. macro];
  nms = Max[Total[microprofiles[[All, 6]]], 1];
  everage =
   Total[MapThread[#1 / #2 &, {#, microprofiles[[All, 6]]}]] / SMTMacroProblemStatus[[6]] &;
  rep[] := {
    {"MACRO", ""},
    {"Number of macro elements", SMTIData["NoElements"]},
    {"Number of macro equations", SMTIData["NoEquations"]},
    {#, # /. macro} &["No. of steps"],
    {#, # /. macro} &["No. of steps back"],
```

```
 {#, # /. macro} &["Total no. of iterations"],
 {#, # /. macro // time} &["Total driver time (s)"],
 {#, # /. macro // time} &["Total linear solver time (s)"],
 {#, # /. macro // time} &["Total K&R time (s)"],
 {"MICRO", ""},
 {"Number of FE^2 elements", Length[SMTMacroElements[[1]]]},
 {"Number of MIEL elements", Length[SMTMacroElements[[2]]]},
 {"Number of single-scale elements", Length[SMTMacroElements[[3]]]},
 {"Parallelization kernels/threads",
  ToString[SMTMacroProblemStatus[[6]]] <> "/" <> ToString[SMTMacroProblemStatus[[17]]]},
 {"Total number of micro problems", Length[SMTAllLocalProblems]},
 {"Total number of micro elements", Total[microprofiles[[All, 7]]]},
 {"Total number of micro nodes", Total[microprofiles[[All, 8]]]},
 {"Total number of micro equations", Total[microprofiles[[All, 9]]]},
 {"Average number of micro equations",
  Total[microprofiles[[All, 9]]] / Length[SMTAllLocalProblems] // N},
 {"Real micro mesh generation time", SMTMacroProblemStatus[[13]] // time},
 {"Number of MicroSolve calls", SMTMacroProblemStatus[[21]]},
 {"Total MicroSolve time", SMTMacroProblemStatus[[12]] // time},
 {"Total number of MP equations solved",
  Total[microprofiles[[All, 9]]] × SMTMacroProblemStatus[[21]]},
 {"Total number of steps", Total[microprofiles[[All, 16]]]},
 {"Total number of back steps", Total[microprofiles[[All, 17]]]},
 {"Average number of micro problems/kernel",
  Round[Length[SMTAllLocalProblems] / SMTMacroProblemStatus[[6]], 0.1] // ToString},
 {"Total number of MP solved", Total[microprofiles[[All, 6]]]},
 {"Average MP time", Total[microprofiles[[All, 2]]] / nms // time},
 {"Average MP K&R time", everage[rdata[[All, 1]]] // time},
 {"Average MP linear solver time", everage[rdata[[All, 2]]] // time},
 {"Average MP sens.p.load time", everage[rdata[[All, 3]]] // time},
 {"Average MP sens.l.solver time", everage[rdata[[All, 4]]] // time},
 {"Average MP Schur complement time", everage[microprofiles[[All, 11]]] // time},
 {"Average MP task time", everage[microprofiles[[All, 12]]] // time},
 {"Average MP Driver time", everage[rdata[[All, 5]]] // time},
 {"Average real Driver time",
  Total[rdata[[All, 5]]] / SMTMacroProblemStatus[[6]] // time},
 {"Total parallel MP time", Total[microprofiles[[All, 2]]] // time},
 {"Number of restarts", Total[microprofiles[[All, 5]]]},
 {"Total parallel restart time", Total[microprofiles[[All, 1]]] // time},
 {"Average restart time",
  Total[microprofiles[[All, 1]]] / Max[Total[microprofiles[[All, 5]]], 1] // time},
 {"Number of dumps", Total[microprofiles[[All, 4]]]},
 {"Total parallel dump time", Total[microprofiles[[All, 3]]] // time},
 {"Average dump time",
  Total[microprofiles[[All, 3]]] / Max[Total[microprofiles[[All, 4]]], 1] // time},
 {"Total dump files size (byte)", Total[FileByteCount[#] & /@
    FileNames[{"*.bst", "*.h5"}, "tmpMS_MicroData"]]},
 {"zlib compression level", SMTMacroProblemStatus[[15]]},
 {"Macro + micro analysis time", avt // time},
 {"Aministrative time", AbsoluteTime[] - SMTMacroProblemStatus[[8]] - avt // time},
 {"Lost time: ", SMTMacroProblemStatus[[12]] - Total[microprofiles[[All, 6]]] /
     Length[SMTAllLocalProblems] × Total[microprofiles[[All, 2]]] / nms ×
    Length[SMTAllLocalProblems] / SMTMacroProblemStatus[[6]] // time},
 {"Total time", AbsoluteTime[] - SMTMacroProblemStatus[[8]] // time}
};
```

```
    time = Which[
       Not[NumberQ[#]], #
       , # < 60, NumberForm[#, {4, 2}]
       , # < 3600, Row[{
         Floor[# / 60.] , ":"
         , NumberForm[# - 60 Floor[# / 60], {4, 2}]
        }]
       , True, Row[{
         Floor[# / 3600.], ":"
         , Floor[(# - 3600. Floor[# / 3600.]) / 60.] , ":"
         , NumberForm[# - Floor[# / 3600.] 3600 - 60 Floor[(# - 3600 Floor[# / 3600.]) / 60], {4, 2}]
        }]
      ] &;
    SMTMacroProblemStatus[[5]] = CreatePalette[Column[{
        Grid[rep[], Frame → {False, All},
         Alignment → Left, ItemStyle → Directive[FontSize → 11], Spacings → {1, 0}]
       }], SMTMacroProblemStatus[[5]], WindowTitle → "# Multi-scale monitor",
      Saveable → False, WindowFloating → False];
    If[tab
     , Print[TextCell[Style[
       Column[{
         Row[{rm}, " ■ "]
         , TableForm[rep[], TableSpacing -> {1, 2}]
        }]
       , Plain, ShowStringCharacters → False]]]
    ];
    time = # &;
    Map[#[[1]] → #[[2]] &, rep[]]
   ];
SMTMultiScaleSimulationReport[rm___] := SMTMultiScaleSimulationReport[True, rm]
```

SMTMultiScaleStatusReport

In[25]:=
```
SMTMultiScaleStatusReport[i___] := Block[
  {microprofiles},
  microprofiles = ParallelEvaluate[SMTMicroProfile, DistributedContexts → None];
  If[SMTSharedStatus[[1]] === 0
   , SMTStatusReport[
    Row[{i, " Events:",
      microprofiles[[All, 13]] // Total,
      " Max./Ave.subs.:", microprofiles[[All, 14]] // Max, "/",
       , Total[MapThread[#1 / #2[[1]] &, {microprofiles[[All, 18]], microprofiles[[All, 6]]}]] /
        SMTMacroProblemStatus[[6]] // N
     }]
   ]
   , SMTStatusReport[
    Row[{"Micro:", i, " Error:", SMTSharedStatus[[1]], "/", SMTSharedStatus[[2]]}]
   ]
  ]
 ]
```

SMTToSymmetricOrUnsymmetric

```
In[26]:= SMTToSymmetricOrUnsymmetric[macsymm_, ndof_, micro_] :=
 Switch[Length[micro]
  , ndof ndof,
  If[macsymm
   , Table[(micro[[(i - 1) ndof + j]] + micro[[(j - 1) ndof + i]]) / 2, {i, 1, ndof}, {j, i, ndof}]
   , micro
  ]
  , ndof (ndof + 1) / 2,
  If[macsymm
   , micro
   , Block[{p1, p2},
    p1 = 0; p2 = Table[0, {i, 1, ndof}, {j, 1, ndof}];
    Do[
     If[i <= j, p2[[i, j]] = micro[[++p1]], p2[[i, j]] = p2[[j, i]]], {i, 1, ndof}, {j, 1, ndof}];
    p2
   ]
  ]
  , _, SMCError = {"Number of macro DOF:", ndof, "\nLength of micro data:", micro // Length};
  SMCAbort["Error during solution of micro problem. Incompatible
     data size. Data should represent either full matrix or upper matrix."];
 ]
```

Multi-scale formulation dependent functions

FE2SolveOne - Solve an arbitrary FE^2 micro problem

- FE2SolveOne[*local_problem_data*, dump] - solves micro problem for given local_problem_data and returns
  - micro_macro _data = Flatten$\left[\{\text{micro_data}_1, \text{micro_data}_2, \ldots \text{micro_data}_{n_g}\}\right]$ where n_g is the number of integration points and
  - micro_data = $\int_{\Omega_e} \text{Flatten}\left[\left\{a_m, \frac{\partial a_m}{\partial d_{BM_1}}, \frac{\partial a_m}{\partial d_{BM_2}}, \ldots, \frac{\partial a_m}{\partial d_{BM_{n_M}}}\right\}\right] dV$.

```
In[27]:= FE2SolveOne[locd_, dump_] := Module[
  {localdata = locd, ConstrainedNodes, X1, Y1, Z1, X2, Y2,
   Z2, X3, Y3, Z3, X4, Y4, Z4, X5, Y5, Z5, X6, Y6, Z6, X7, Y7, Z7, X8, Y8, Z8,
   bc, Δλ, F, λMax, ΔλMin, Δλ0, Δλstep, minss, maxss, method, microtask,
   ΔλMax, tolNR, maxNR, targetNR, step, dB, first, err, ϵ, vecF, vecϵ, Dϑ},

  ConstrainedNodes = localdata[[8]];
  (* new velocity fields *)
  Switch[localdata[[1, 5]]
   , "Deformation gradient (plane strain)",
   vecF = localdata[[7]];
   F = {{vecF[[1]], vecF[[2]]}, {vecF[[3]], vecF[[4]]}};
   {{X1, Y1}, {X2, Y2}, {X4, Y4}} = SMTNodeData[ConstrainedNodes, "X"];
   (*dbc=bc-bc_p=(F-1).X-bc_p*)
   bc = {{F[[1]].{X1, Y1} - X1, F[[2]].{X1, Y1} - Y1}
     , {F[[1]].{X2, Y2} - X2, F[[2]].{X2, Y2} - Y2}
     , {F[[1]].{X4, Y4} - X4, F[[2]].{X4, Y4} - Y4}
    };
   (*set the pattern for the current
    EBC velocity (dΨ) field and clear the histrory (Ψp,Ψt) *)
   SMTSetVelocityFields[{
     "∂u/∂F11" -> {ConstrainedNodes , {X1, X2, X4}},
```

```
  "∂u/∂F12" -> {ConstrainedNodes , {Y1, Y2, Y4}},
  "∂v/∂F21" -> {ConstrainedNodes , {X1, X2, X4}},
  "∂v/∂F22" -> {ConstrainedNodes , {Y1, Y2, Y4}}
 }];

, "Small strain tensor (plane strain)",
vecϵ = localdata[[7]];
ϵ = {{vecϵ[[1]], vecϵ[[2]]}, {vecϵ[[2]], vecϵ[[3]]}};
{{X1, Y1}, {X2, Y2}, {X4, Y4}} = SMTNodeData[ConstrainedNodes, "X"];
(*dbc=bc-bc_p=ϵ.X-bc_p*)
bc = {{ϵ[[1]].{X1, Y1}, ϵ[[2]].{X1, Y1}}
  , {ϵ[[1]].{X2, Y2}, ϵ[[2]].{X2, Y2}}
  , {ϵ[[1]].{X4, Y4}, ϵ[[2]].{X4, Y4}}
 };
(*set the pattern for the current
 EBC velocity (dΨ) field and clear the histrory (Ψp,Ψt) *)
SMTSetVelocityFields[{
  "∂u/∂e11" -> {ConstrainedNodes , {X1, X2, X4}},
  "∂u/∂e12" -> {ConstrainedNodes , {Y1, Y2, Y4}},
  "∂v/∂e12" -> {ConstrainedNodes , {X1, X2, X4}},
  "∂v/∂e22" -> {ConstrainedNodes , {Y1, Y2, Y4}}
 }];

, "Temperature gradient (2D)",
Dθ = localdata[[7]];
{{X1, Y1}, {X2, Y2}, {X4, Y4}} = SMTNodeData[ConstrainedNodes, "X"];
(*dbc=bc-bc_p=Dθ.X-bc_p*)
bc = {{Dθ.{X1, Y1}}
  , {Dθ.{X2, Y2}}
  , {Dθ.{X4, Y4}}
 };
(*set the pattern for the current
 EBC velocity (dΨ) field and clear the histrory (Ψp,Ψt) *)
SMTSetVelocityFields[{
  "∂θ/∂Dθ1" -> {ConstrainedNodes , {X1, X2, X4}},
  "∂θ/∂Dθ2" -> {ConstrainedNodes , {Y1, Y2, Y4}}
 }];

, "Deformation gradient (plane strain)+temperature gradient",
F = {{localdata[[7, 1]], localdata[[7, 2]]}, {localdata[[7, 3]], localdata[[7, 4]]}};
Dθ = localdata[[7, {5, 6}]];
{{X1, Y1}, {X2, Y2}, {X4, Y4}} = SMTNodeData[ConstrainedNodes, "X"];
(*dbc=bc-bc_p=(F-1).X-bc_p, dbc=bc-bc_p=Dθ.X-bc_p*)
bc = {{F[[1]].{X1, Y1} - X1, F[[2]].{X1, Y1} - Y1, Dθ.{X1, Y1}}
  , {F[[1]].{X2, Y2} - X2, F[[2]].{X2, Y2} - Y2, Dθ.{X2, Y2}}
  , {F[[1]].{X4, Y4} - X4, F[[2]].{X4, Y4} - Y4, Dθ.{X4, Y4}}
 };
(*set the pattern for the current
 EBC velocity (dΨ) field and clear the histrory (Ψp,Ψt) *)
SMTSetVelocityFields[{
  "∂u/∂F11" -> {ConstrainedNodes , {X1, X2, X4}},
  "∂u/∂F12" -> {ConstrainedNodes , {Y1, Y2, Y4}},
  "∂v/∂F21" -> {ConstrainedNodes , {X1, X2, X4}},
  "∂v/∂F22" -> {ConstrainedNodes , {Y1, Y2, Y4}},
  "∂θ/∂Dθ1" -> {ConstrainedNodes , {X1, X2, X4}},
```

```
  "∂θ/∂Dθ2" -> {ConstrainedNodes , {Y1, Y2, Y4}}
 }];

, "Deformation gradient (3D)",
vecF = localdata[[7]];
F = {{vecF[[1]], vecF[[2]], vecF[[3]]},
  {vecF[[4]], vecF[[5]], vecF[[6]]}, {vecF[[7]], vecF[[8]], vecF[[9]]}};
{{X1, Y1, Z1}, {X2, Y2, Z2}, {X3, Y3, Z3}, {X4, Y4, Z4}, {X5, Y5, Z5}, {X6, Y6, Z6},
  {X7, Y7, Z7}, {X8, Y8, Z8}} = SMTNodeData[ConstrainedNodes, "X"];
(*dbc=bc-bc_p=(F-1).X-bc_p*)
bc = {
  {F[[1]].{X1, Y1, Z1} - X1, F[[2]].{X1, Y1, Z1} - Y1, F[[3]].{X1, Y1, Z1} - Z1},
  {F[[1]].{X2, Y2, Z2} - X2, F[[2]].{X2, Y2, Z2} - Y2, F[[3]].{X2, Y2, Z2} - Z2},
  {F[[1]].{X3, Y3, Z3} - X3, F[[2]].{X3, Y3, Z3} - Y3, F[[3]].{X3, Y3, Z3} - Z3},
  {F[[1]].{X4, Y4, Z4} - X4, F[[2]].{X4, Y4, Z4} - Y4, F[[3]].{X4, Y4, Z4} - Z4},
  {F[[1]].{X5, Y5, Z5} - X5, F[[2]].{X5, Y5, Z5} - Y5, F[[3]].{X5, Y5, Z5} - Z5},
  {F[[1]].{X6, Y6, Z6} - X6, F[[2]].{X6, Y6, Z6} - Y6, F[[3]].{X6, Y6, Z6} - Z6},
  {F[[1]].{X7, Y7, Z7} - X7, F[[2]].{X7, Y7, Z7} - Y7, F[[3]].{X7, Y7, Z7} - Z7},
  {F[[1]].{X8, Y8, Z8} - X8, F[[2]].{X8, Y8, Z8} - Y8, F[[3]].{X8, Y8, Z8} - Z8}
 };

(*set the pattern for the current
 EBC velocity (dΨ) field and clear the histrory (Ψp,Ψt) *)
SMTSetVelocityFields[{
  "∂u/∂F11" -> {ConstrainedNodes , {X1, X2, X3, X4, X5, X6, X7, X8}},
  "∂u/∂F12" -> {ConstrainedNodes , {Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8}},
  "∂u/∂F13" -> {ConstrainedNodes , {Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8}},

  "∂v/∂F21" -> {ConstrainedNodes , {X1, X2, X3, X4, X5, X6, X7, X8}},
  "∂v/∂F22" -> {ConstrainedNodes , {Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8}},
  "∂v/∂F23" -> {ConstrainedNodes , {Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8}},

  "∂w/∂F31" -> {ConstrainedNodes , {X1, X2, X3, X4, X5, X6, X7, X8}},
  "∂w/∂F32" -> {ConstrainedNodes , {Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8}},
  "∂w/∂F33" -> {ConstrainedNodes , {Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8}}
 }];

, "Small strain tensor (3D)",
vecϵ = localdata[[7]];
ϵ = {{vecϵ[[1]], vecϵ[[2]], vecϵ[[3]]},
  {vecϵ[[2]], vecϵ[[4]], vecϵ[[5]]}, {vecϵ[[3]], vecϵ[[5]], vecϵ[[6]]}};
{{X1, Y1, Z1}, {X2, Y2, Z2}, {X3, Y3, Z3}, {X4, Y4, Z4}, {X5, Y5, Z5}, {X6, Y6, Z6},
  {X7, Y7, Z7}, {X8, Y8, Z8}} = SMTNodeData[ConstrainedNodes, "X"];
(*dbc=bc-bc_p=ϵ.X-bc_p*)
bc = {
  {ϵ[[1]].{X1, Y1, Z1}, ϵ[[2]].{X1, Y1, Z1}, ϵ[[3]].{X1, Y1, Z1}},
  {ϵ[[1]].{X2, Y2, Z2}, ϵ[[2]].{X2, Y2, Z2}, ϵ[[3]].{X2, Y2, Z2}},
  {ϵ[[1]].{X3, Y3, Z3}, ϵ[[2]].{X3, Y3, Z3}, ϵ[[3]].{X3, Y3, Z3}},
  {ϵ[[1]].{X4, Y4, Z4}, ϵ[[2]].{X4, Y4, Z4}, ϵ[[3]].{X4, Y4, Z4}},
  {ϵ[[1]].{X5, Y5, Z5}, ϵ[[2]].{X5, Y5, Z5}, ϵ[[3]].{X5, Y5, Z5}},
  {ϵ[[1]].{X6, Y6, Z6}, ϵ[[2]].{X6, Y6, Z6}, ϵ[[3]].{X6, Y6, Z6}},
  {ϵ[[1]].{X7, Y7, Z7}, ϵ[[2]].{X7, Y7, Z7}, ϵ[[3]].{X7, Y7, Z7}},
  {ϵ[[1]].{X8, Y8, Z8}, ϵ[[2]].{X8, Y8, Z8}, ϵ[[3]].{X8, Y8, Z8}}
 };
```

```
 (*set the pattern for the current
  EBC velocity (dΨ) field and clear the histrory (Ψp,Ψt) *)
 SMTSetVelocityFields[{
   "∂u/∂e11" -> {ConstrainedNodes , {X1, X2, X3, X4, X5, X6, X7, X8}},
   "∂u/∂e12" -> {ConstrainedNodes , {Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8}},
   "∂u/∂e13" -> {ConstrainedNodes , {Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8}},

   "∂v/∂e12" -> {ConstrainedNodes , {X1, X2, X3, X4, X5, X6, X7, X8}},
   "∂v/∂e22" -> {ConstrainedNodes , {Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8}},
   "∂v/∂e23" -> {ConstrainedNodes , {Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8}},

   "∂w/∂e13" -> {ConstrainedNodes , {X1, X2, X3, X4, X5, X6, X7, X8}},
   "∂w/∂e23" -> {ConstrainedNodes , {Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8}},
   "∂w/∂e33" -> {ConstrainedNodes , {Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8}}
  }];

];

(*set increment of boundary conditions relative to the last solution*)
dB = bc - SMTNodeData[ConstrainedNodes, "Bp"];
SMTNodeData[ConstrainedNodes, "dB", dB];

{minss, maxss, method, tolNR} =
 ReplaceAll[{"MinSubSteps", "MaxSubSteps", "Method", "tolNR"} /. localdata[[10]],
   {"MinSubSteps" → 1, "MaxSubSteps" → 100, "Method" → "Sensitivity", "tolNR" -> 10.^-10}];

(*solve micro problem for incremental boundary conditions using adaptive load stepping*)
Δλstep = 1.;
(*first iteration of global NR method should keep the tangent
  direction from the end of the last step - e.g.  elastic/plastic state frozen*)
first = Norm[dB // Flatten] < tolNR && Not[dump] && SMTMacroProblemStatus[[20]] == 1 ;
If[first
 , SMTNextStep["Δλ" → Δλstep];
 err = SMTNewtonIteration[];
 If[err < tolNR
  , If[method === "SchurMKL" || method === "SchurMMA"
   , SMTMacroProblemStatus[[16]][localdata, True, "MICRO first global iteration"];
   SMTStepBack[];
   SMTNextStep["Δλ" → Δλstep];
    Goto[schur];
  ];
  SMTSensitivity[];
  SMTMacroProblemStatus[[16]][localdata, True, "MICRO first global iteration"];
  Return[SMTTask[localdata[[1, 7]]] / localdata[[9]] , Module];
  , SMTStepBack[];
 ];
];
Δλ0 = Δλstep / minss; ΔλMax = Δλstep / minss; ΔλMin = Δλstep / maxss;
λMax = SMTRData["Multiplier"] + Δλstep;
maxNR = 30; targetNR = 8;
SMTNextStep["Δλ" → Δλ0];
While[
 While[step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λMax}]
  , SMTNewtonIteration[];
  SMTMacroProblemStatus[[16]][localdata, False, "MICRO iteration"];
```

```
 ];
 If[step[[4]] === "MinBound"
  , SMTUpdateSharedStatus["Local", {2, Row[{"Error in local problem: Δλ < Δλmin",
      "\n", localdata[[1, 1]], " problem at point=", localdata[[2]],
      " loc. problem=", localdata[[4]], " macro element=", localdata[[6]],
      , " |dB|=", Total[dB // Abs // Flatten]
     }]}];
  Return[False, Module];
 ];
 If[Not[step[[1]] && method === "Sensitivity", SMTSensitivity[];];
 step[[3]]
 , If[step[[1]], SMTStepBack[];];
 SMTNextStep["Δλ" → step[[2]]]
];
If[ method === "Quasi-sensitivity", SMTSensitivity[];];
SMTMacroProblemStatus[[16]][localdata, True, "MICRO converged"];

Label[schur];
If[dump
 , SMTNextStep["Δλ" → 0];
 SMTNewtonIteration[];
];
Switch[method
 , "Sensitivity" | "Quasi-sensitivity",
 SMTMicroProfile[[12]] +=
   AbsoluteTiming[microtask = SMTTask[localdata[[1, 7]]] / localdata[[9]]][[1]];
 , "SchurMKL" | "SchurMMA",
 SMTMicroProfile[[11]] += AbsoluteTiming[microtask = FE2Schur[localdata, method]][[1]];
];
microtask
]
```

FE2Schur - calculate Schur complement of constrained nodes and transformation for FE2 method

In[28]:=

```
FE2Schur[localdata_, method_] :=
 Module[{addinfo, Kc, Rc, nodesindex, nodesdof, index,
   ConstrainedNodes, constraineddof, el, sigmamacro, S4macro, X0, Xd, vecF, F,
   dof, ebc, nbc, mat},

  {addinfo, Kc, Rc, nodesindex, nodesdof} = SMTSchurComplementOfEssentialBC[];
  If[method === "SchurMMA",
   (*Kcc without MKL - ill conditioning*)
   dof = SMTNodeData["DOF"];
   ebc = 1 + MapThread[ dof[[#1, #2]] &, {nodesindex, nodesdof}];
   nbc = Complement[Range[SMTIData["NoEquations"]], ebc];
   mat = SMTData["TangentMatrix"];
   Kc =
    Part[mat, ebc, ebc] - Part[mat, ebc, nbc].Inverse[Part[mat, nbc, nbc]].Part[mat, nbc, ebc];
  ];
  index = MapThread[{#1, #2} &, {nodesindex, nodesdof}];
  ConstrainedNodes = localdata[[8]];
  X0 = SMTNodeData[ConstrainedNodes, "X"];
  Switch[localdata[[1, 5]]
   , "Deformation gradient (3D)",
   vecF = localdata[[7]];
   F = {{vecF[[1]], vecF[[2]], vecF[[3]]},
```

```
    {vecF[[4]], vecF[[5]], vecF[[6]]}, {vecF[[7]], vecF[[8]], vecF[[9]]}};
  Xd = Map[F.# &, X0];
  , "Small strain tensor (3D)",
  Xd = X0;
 ];
 constraineddof = Flatten[Map[{{#, 1}, {#, 2}, {#, 3}} &, ConstrainedNodes], 1];
 el = Partition[Flatten[Map[Position[index, #] &, constraineddof]], 3];
 sigmamacro = Sum[ArrayFlatten[TensorProduct[Rc[[el[[i]]]], X0[[i]]]],
    {i, 1, Length[ConstrainedNodes]}] / localdata[[9]];
 Switch[localdata[[1, 5]]
  , "Deformation gradient (3D)",
  S4macro = (
     Sum[ArrayFlatten[
       TensorProduct[
        Sum[
         TensorProduct[{{Kc[[el[[i, 1]], el[[j, 1]]]], Kc[[el[[i, 1]], el[[j, 2]]]],
            Kc[[el[[i, 1]], el[[j, 3]]]]}, {Kc[[el[[i, 2]], el[[j, 1]]]],
            Kc[[el[[i, 2]], el[[j, 2]]]], Kc[[el[[i, 2]], el[[j, 3]]]]},
           {Kc[[el[[i, 3]], el[[j, 1]]]], Kc[[el[[i, 3]], el[[j, 2]]]],
            Kc[[el[[i, 3]], el[[j, 3]]]]}}, X0[[i]]]
         , {i, 1, Length[ConstrainedNodes]}
        ]
        , X0[[j]]]]
      , {j, 1, Length[ConstrainedNodes]}]
    ) / localdata[[9]];
  Flatten[{sigmamacro, S4macro}]

  , "Small strain tensor (3D)",
  S4macro = (Sum[ArrayFlatten[TensorProduct[X0[[i]],
         {{Kc[[el[[i, 1]], el[[j, 1]]]], Kc[[el[[i, 1]], el[[j, 2]]]],
           Kc[[el[[i, 1]], el[[j, 3]]]]}, {Kc[[el[[i, 2]], el[[j, 1]]]],
           Kc[[el[[i, 2]], el[[j, 2]]]], Kc[[el[[i, 2]], el[[j, 3]]]]},
          {Kc[[el[[i, 3]], el[[j, 1]]]], Kc[[el[[i, 3]], el[[j, 2]]]],
           Kc[[el[[i, 3]], el[[j, 3]]]]}},
         X0[[j]]]], {i, 1, Length[ConstrainedNodes]}, {j, 1, Length[ConstrainedNodes]}]
     + Sum[ArrayFlatten[TensorProduct[Rc[[el[[i]]]], {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}},
         X0[[i]]]], {i, 1, Length[ConstrainedNodes]}]) / localdata[[9]];
  Flatten[Join[sigmamacro[[1, 1 ;; 3]], sigmamacro[[2, 2 ;; 3]], {sigmamacro[[3, 3]]},
    {S4macro[[1, 1]], S4macro[[1, 2]] + S4macro[[2, 1]], S4macro[[1, 3]] + S4macro[[3, 1]],
     S4macro[[2, 2]], S4macro[[2, 3]] + S4macro[[3, 2]], S4macro[[3, 3]],
     S4macro[[1, 4]], S4macro[[1, 5]] + S4macro[[2, 4]], S4macro[[1, 6]] + S4macro[[3, 4]],
     S4macro[[2, 5]], S4macro[[2, 6]] + S4macro[[3, 5]], S4macro[[3, 6]],
     S4macro[[1, 7]], S4macro[[1, 8]] + S4macro[[2, 7]], S4macro[[1, 9]] + S4macro[[3, 7]],
     S4macro[[2, 8]], S4macro[[2, 9]] + S4macro[[3, 8]], S4macro[[3, 9]],
     S4macro[[4, 4]], S4macro[[4, 5]] + S4macro[[5, 4]], S4macro[[4, 6]] + S4macro[[6, 4]],
     S4macro[[5, 5]], S4macro[[5, 6]] + S4macro[[6, 5]], S4macro[[6, 6]],
     S4macro[[4, 7]], S4macro[[4, 8]] + S4macro[[5, 7]], S4macro[[4, 9]] + S4macro[[6, 7]],
     S4macro[[5, 8]], S4macro[[5, 9]] + S4macro[[6, 8]], S4macro[[6, 9]],
     S4macro[[7, 7]], S4macro[[7, 8]] + S4macro[[8, 7]], S4macro[[7, 9]] + S4macro[[9, 7]],
     S4macro[[8, 8]], S4macro[[8, 9]] + S4macro[[9, 8]], S4macro[[9, 9]]}]]
 ]

];
```

MIELSolveOne - Solve an arbitrary MIEL micro problem

MIELSolveOne[local_problem_data, dump] - solve micro problem for given local_problem_data

In[29]:=

```
ξT1Q1[x1_, y1_, x2_, y2_, X_, Y_] := (X x1 - x1^2 - X x2 + x1 x2 + Y y1 - y1^2 - Y y2 + y1 y2)/(-x1^2 + 2 x1 x2 - x2^2 - y1^2 + 2 y1 y2 - y2^2);

ξT2Q2S[x1_, y1_, x2_, y2_, x3_, y3_, X_, Y_, tolP_] := Module[{a, b, c, d},
  Switch[Abs[2 x1 - 4 x2 + 2 x3] < tolP && Abs[2 y1 - 4 y2 + 2 y3] < tolP
   , True,
   (3 X x1 - 3 x1^2 - 4 X x2 + 4 x1 x2 + X x3 - x1 x3 + 3 Y y1 - 3 y1^2 - 4 Y y2 + 4 y1 y2 + Y y3 - y1 y3)/
    (-9 x1^2 + 24 x1 x2 - 16 x2^2 - 6 x1 x3 + 8 x2 x3 - x3^2 - 9 y1^2 + 24 y1 y2 - 16 y2^2 - 6 y1 y3 + 8 y2 y3 - y3^2)
   , False,
   a = -(X - x1) (3 x1 - 4 x2 + x3) - (Y - y1) (3 y1 - 4 y2 + y3);
   b = -13 x1^2 + 2 x1 (16 x2 - 5 x3) - (-4 x2 + x3)^2 +
     4 X (x1 - 2 x2 + x3) - 13 y1^2 + 2 y1 (16 y2 - 5 y3) - (-4 y2 + y3)^2 + 4 Y (y1 - 2 y2 + y3);
   c = 6 ((3 x1 - 4 x2 + x3) (x1 - 2 x2 + x3) + (3 y1 - 4 y2 + y3) (y1 - 2 y2 + y3));
   d = -8 ((x1 - 2 x2 + x3)^2 + (y1 - 2 y2 + y3)^2);
   -c / (3*d) - (2^(1/3) * (-c^2 + 3*b*d)) / (3*d*(-2*c^3 + 9*b*c*d - 27*a*d^2 +
         Sqrt[-4*(c^2 - 3*b*d)^3 + (2*c^3 - 9*b*c*d + 27*a*d^2)^2])^(1/3)) +
    (-2*c^3 + 9*b*c*d - 27*a*d^2 + Sqrt[-4*(c^2 - 3*b*d)^3 +
         (2*c^3 - 9*b*c*d + 27*a*d^2)^2])^(1/3) / (3*2^(1/3)*d)
  ]
 ];
```

In[31]:=

```
MIELSolveOne[locd_, dump_] := Module[
  {localdata = locd, microtopology, bc, Δλstep, Δλ0, λMax, ΔλMin, minss, maxss,
   ΔλMax, tolNR, maxNR, targetNR, step, dB, first, tol, microtask,
   err, XIO, uIO, edge, ndim, ndof, dof, index, us, ret, points,
   ξ, equations, all, mmesh, Xξ, xy, xyz, nodes, uξ, η, ξ, uξη, xyξ, xyzξ,
   xyzη, xyzξ, dis, nodes1, nodes2, xyz1, xyz2, Q1ξ, Q1η, edgeH, p1, method},

  microtopology = localdata[[9]];
  ndim = SMTMacroProblemStatus[[1]];
  (* transform p_e→p̂_e*)
  uIO = Partition[localdata[[7]], ndim];
  XIO = localdata[[8]];
  tol = SMTRData["MinElementSize"] / 10.;

  edge = Switch[microtopology
    , "T1", {{1, 2}, {2, 3}, {3, 1}}
    , "Q1", {{1, 2}, {2, 3}, {3, 4}, {4, 1}}
    , "T2", {{1, 4, 2}, {2, 5, 3}, {3, 6, 1}}
    , "Q2S", {{1, 5, 2}, {2, 6, 3}, {3, 7, 4}, {4, 8, 1}}
    , "O1", {{1, 4, 2}, {1, 2, 3}, {2, 4, 3}, {4, 1, 3}}
    , "O2", {{1, 4, 2, 8, 9, 5}, {1, 2, 3, 5, 6, 7}, {2, 4, 3, 9, 10, 6}, {4, 1, 3, 8, 7, 10}}
    , "H1", {{4, 3, 2, 1}, {1, 2, 6, 5}, {2, 3, 7, 6}, {3, 4, 8, 7}, {4, 1, 5, 8}, {5, 6, 7, 8}}
    , "H2S", {{4, 3, 2, 1, 11, 10, 9, 12}, {1, 2, 6, 5, 9, 14, 17, 13}, {2, 3, 7, 6, 10, 15, 18, 14},
     {3, 4, 8, 7, 11, 16, 19, 15}, {4, 1, 5, 8, 12, 13, 20, 16}, {5, 6, 7, 8, 17, 18, 19, 20}}
   ];
  ndof = Switch[microtopology, "T1", 6, "Q1",
    8, "T2", 12, "Q2S", 16, "O1", 12, "O2", 30, "H1", 24, "H2S", 60];
  equations = Switch[microtopology
    , "T1" | "Q1", {1 - ξ, ξ}
```

```
  , "T2" | "Q2S", {2 ξ^2 - 3 ξ + 1, -4 ξ^2 + 4 ξ, 2 ξ^2 - ξ}
  , "O1", {ξ, η, 1 - ξ - η}
  , "O2", {(2 ξ - 1) ξ, (2 η - 1) η, (2 (1 - ξ - η) - 1) (1 - ξ - η), 4 ξ η, 4 η (1 - ξ - η), 4 (1 - ξ - η) ξ}
  , "H1", {1/4 (1 - ξ) (1 - η), 1/4 (1 + ξ) (1 - η), 1/4 (1 + ξ) (1 + η), 1/4 (1 - ξ) (1 + η)}
  , "H2S", {1/4 (1 - ξ) (1 - η) (-ξ - η - 1),
   1/4 (1 + ξ) (1 - η) (ξ - η - 1), 1/4 (1 + ξ) (1 + η) (ξ + η - 1), 1/4 (1 - ξ) (1 + η) (-ξ + η - 1),
   1/2 (1 - ξ^2) (1 - η), 1/2 (1 + ξ) (1 - η^2), 1/2 (1 - ξ^2) (1 + η), 1/2 (1 - ξ) (1 - η^2)}
 ];
(*all={ {{node index, nodal displacement, nodal shape functions} ..
      for all nodes on boundary surface}
    ... for all surfaces}*)
all = Switch[microtopology
  , "T1" | "Q1" | "T2" | "Q2S",
  Map[(
    mmesh = #;
    Xξ = equations.XIO[[mmesh]];
    (* sufficiently dense points on edge *)
    points = Table[Xξ, {ξ, 0., 1., 1./localdata[[11]]}];
    nodes = SMTFindNodes[Line[points, "D", tol]];
    xy = SMTNodeData[nodes, "X"];
    uξ = equations.uIO[[mmesh]];
    MapThread[(
      xyξ = Switch[microtopology
        , "T1" | "Q1", ξT1Q1[XIO[[mmesh[[1]], 1]],
         XIO[[mmesh[[1]], 2]], XIO[[mmesh[[2]], 1]], XIO[[mmesh[[2]], 2]], #[[1]], #[[2]]]
        , "T2" | "Q2S", ξT2Q2S[XIO[[mmesh[[1]], 1]], XIO[[mmesh[[1]], 2]], XIO[[mmesh[[2]], 1]],
         XIO[[mmesh[[2]], 2]], XIO[[mmesh[[3]], 1]], XIO[[mmesh[[3]], 2]], #[[1]], #[[2]], 10.^-8]
       ];
      {#2, uξ, equations} /. ξ → xyξ
     ) &, {xy, nodes}]
   ) &, edge]
  , "O1" | "O2",
  Map[(
    mmesh = #;
    nodes = SMTFindNodes[Polygon[XIO[[mmesh[[1 ;; 3]]]], "D", 10^-10(*tol*)]];
    xyz = SMTNodeData[nodes, "X"];
    uξη = equations.uIO[[mmesh]];
    MapThread[(
      {xyzξ, xyzη, xyzζ, dis} =
       SMTInverseArea["P1"][#[[1]], #[[2]], #[[3]]
        , XIO[[mmesh[[1]], 1]], XIO[[mmesh[[1]], 2]], XIO[[mmesh[[1]], 3]]
        , XIO[[mmesh[[2]], 1]], XIO[[mmesh[[2]], 2]], XIO[[mmesh[[2]], 3]]
        , XIO[[mmesh[[3]], 1]], XIO[[mmesh[[3]], 2]], XIO[[mmesh[[3]], 3]]
        , 10^-10];
      {#2, uξη, equations} /. {ξ → xyzξ, η → xyzη}
     ) &, {xyz, nodes}]
   ) &, edge]
  , "H1" | "H2S",
  Map[(
```

```
    mmesh = #;
    nodes = SMTFindNodes[Polygon[XIO[[mmesh[[{1, 2, 3, 4}]]]], "D", 10^-10(*tol*)]];
    nodes1 = SMTFindNodes[Polygon[XIO[[mmesh[[{1, 2, 4}]]]], "D", 10^-10(*tol*)]];
    nodes2 = Complement[nodes, nodes1];
    xyz1 = SMTNodeData[nodes1, "X"];
    xyz2 = SMTNodeData[nodes2, "X"];
    uξη = equations.uIO[[mmesh]];
    Join[MapThread[(
        {xyzξ, xyzξ, xyzη, dis} =
         SMTInverseArea["P1"][#1[[1]], #1[[2]], #1[[3]]
          , XIO[[mmesh[[1]], 1]], XIO[[mmesh[[1]], 2]], XIO[[mmesh[[1]], 3]]
          , XIO[[mmesh[[2]], 1]], XIO[[mmesh[[2]], 2]], XIO[[mmesh[[2]], 3]]
          , XIO[[mmesh[[4]], 1]], XIO[[mmesh[[4]], 2]], XIO[[mmesh[[4]], 3]]
          , 10^-10];
        {Q1ξ, Q1η} = {-1 + 2 xyzξ, -1 + 2 xyzη};
        {#2, uξη, equations} /. {ξ → Q1ξ, η → Q1η}
       ) &, {xyz1, nodes1}],
     MapThread[(
        {xyzξ, xyzξ, xyzη, dis} =
         SMTInverseArea["P1"][#1[[1]], #1[[2]], #1[[3]]
          , XIO[[mmesh[[3]], 1]], XIO[[mmesh[[3]], 2]], XIO[[mmesh[[3]], 3]]
          , XIO[[mmesh[[4]], 1]], XIO[[mmesh[[4]], 2]], XIO[[mmesh[[4]], 3]]
          , XIO[[mmesh[[2]], 1]], XIO[[mmesh[[2]], 2]], XIO[[mmesh[[2]], 3]]
          , 10^-10];
        {Q1ξ, Q1η} = {1 - 2 xyzξ, 1 - 2 xyzη};
        {#2, uξη, equations} /. {ξ → Q1ξ, η → Q1η}
       ) &, {xyz2, nodes2}]]
   ) &, edge]
 ];

dB = Flatten[all[[All, All, 2]], 1] - SMTNodeData[all[[All, All, 1]] // Flatten, "Bp"];
SMTNodeData[all[[All, All, 1]] // Flatten, "dB", dB];

(*set new velocity fields*)
SMTSetVelocityFields[
 MapThread[(
     Table[
      us = Switch[u, 1, "u", 2, "v", 3, "w"];
      "∂" <> us <> "/∂" <> us <> "M" <> ToString[#1⟦j⟧] → {#2[[All, 1]], #2[[All, 3, j]]}
      , {u, 1, ndim}
      , {j, 1, Length[#1]}
     ]
    ) &, {edge, all}] // Flatten
];

{minss, maxss, method, tolNR} =
 ReplaceAll[{"MinSubSteps", "MaxSubSteps", "Method", "tolNR"} /. localdata[[12]],
  {"MinSubSteps" → 1, "MaxSubSteps" → 100, "Method" → "Sensitivity", "tolNR" -> 10.^-10}];
Δλstep = 1.;
(*first iteration of global NR method should keep the tangent
 direction from the end of the last step*)
first = Norm[dB // Flatten] < tolNR && Not[dump] && SMTMacroProblemStatus[[20]] == 1 ;
If[first
 , SMTNextStep["Δλ" → Δλstep];
 err = SMTNewtonIteration[];
```

```
 If[err < tolNR
  , If[method === "SchurMKL"
   , SMTMacroProblemStatus[[16]][localdata, True, "MICRO first global iteration"];
   SMTStepBack[];
   SMTNextStep["Δλ" → Δλstep];
    Goto[schur];
  ];
  SMTSensitivity[];
  SMTMacroProblemStatus[[16]][localdata, True, "MICRO first global iteration"];
  microtask = SMTTask[localdata[[1, 7]]];
  ret = Flatten[{
     microtask[[1]],
     microtask[[2 ;; ndof + 1]],
     SMTToSymmetricOrUnsymmetric[localdata[[13]], ndof, Drop[microtask, ndof + 1]]
    }];
  Return[ret, Module];
  , SMTStepBack[];
 ];
];
Δλ0 = Δλstep / minss; ΔλMax = Δλstep / minss; ΔλMin = Δλstep / maxss;
λMax = SMTRData["Multiplier"] + Δλstep;
maxNR = 30; targetNR = 8;
SMTNextStep["Δλ" → Δλ0];
While[
 While[step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λMax}]
  , SMTNewtonIteration[];
  SMTMacroProblemStatus[[16]][localdata, False, "MICRO iteration"];
 ];
 If[step[[4]] === "MinBound"
  , SMTUpdateSharedStatus["Local", {2, Row[{"Error in local problem: Δλ < Δλmin",
      "\n", localdata[[1, 1]], " problem at point=", localdata[[2]],
      " loc. problem=", localdata[[4]], " macro element=", localdata[[6]],
      , " |dB|=", Total[dB // Abs // Flatten]
     }]}];
  Return[False, Module];
 ];
 If[Not[step[[1]] && method === "Sensitivity", SMTSensitivity[];];
 step[[3]]
 , If[step[[1]], SMTStepBack[];];
 SMTNextStep["Δλ" → step[[2]]]
];
If[ method === "Quasi-sensitivity", SMTSensitivity[];];
SMTMacroProblemStatus[[16]][localdata, True, "MICRO converged"];

Label[schur];
If[dump
 , SMTNextStep["Δλ" → 0];
 SMTNewtonIteration[];
];
Switch[method
 , "Sensitivity" | "Quasi-sensitivity",
 SMTMicroProfile[[12]] += AbsoluteTiming[microtask = SMTTask[localdata[[1, 7]]]][[1]];
 , "SchurMKL",
 SMTMicroProfile[[11]] +=
   AbsoluteTiming[microtask = Join[{0}, MIELSchur[microtopology, all]] // Flatten][[1]];
```

```
];
ret = Flatten[{
   microtask[[1]],
   microtask[[2 ;; ndof + 1]],
   SMTToSymmetricOrUnsymmetric[localdata[[13]], ndof, Drop[microtask, ndof + 1]]
  }];
ret
]
```

MIELSchur - calculate Schur complement of constrained nodes and transformation for MIEL method

In[32]:=
```
MIELSchur[microtopology_, all_] := Module[
 {addinfo, Kc, Rc, nodesindex, nodesdof, indexen1, indexen2, indexi, indexj, factors, LM,
  Kfinal, Rfinal, i},
 Switch[microtopology
  , "O1",
  {addinfo, Kc, Rc, nodesindex, nodesdof} = SMTSchurComplementOfEssentialBC[];
  (*Transformation matrix*)
  indexen1 = SparseArray[MapIndexed[#[[1]] → #2[[1]] &, all, {2}] // Flatten, {SMTNoNodes}];
  indexen2 = SparseArray[MapIndexed[#[[1]] → #2[[2]] &, all, {2}] // Flatten, {SMTNoNodes}];
  indexi = Flatten[Table[{i, i, i}, {i, 1, Length[nodesindex]}]];
  indexj = Flatten[MapThread[(Switch[indexen1[[#1]]
        , 0, Nothing
        , 1, {{1, 10, 4}, {2, 11, 5}, {3, 12, 6}}[[#2]]
        , 2, {{1, 4, 7}, {2, 5, 8}, {3, 6, 9}}[[#2]]
        , 3, {{4, 10, 7}, {5, 11, 8}, {6, 12, 9}}[[#2]]
        , 4, {{10, 1, 7}, {11, 2, 8}, {12, 3, 9}}[[#2]]]) &, {nodesindex, nodesdof}], 1];
  factors = Flatten[Table[all[[indexen1[[nodesindex[[i]]]], indexen2[[nodesindex[[i]]]]]][[
      3]], {i, 1, Length[nodesindex]}]];
  LM = SparseArray[MapThread[({#1, #2} → #3) &, {indexi, indexj, factors}]];

  , "O2",
  {addinfo, Kc, Rc, nodesindex, nodesdof} = SMTSchurComplementOfEssentialBC[];
  (*Transformation matrix*)
  indexen1 = SparseArray[MapIndexed[#[[1]] → #2[[1]] &, all, {2}] // Flatten, {SMTNoNodes}];
  indexen2 = SparseArray[MapIndexed[#[[1]] → #2[[2]] &, all, {2}] // Flatten, {SMTNoNodes}];
  indexi = Flatten[Table[{i, i, i, i, i, i}, {i, 1, Length[nodesindex]}]];
  indexj = Flatten[MapThread[(Switch[indexen1[[#1]]
        , 0, Nothing
        , 1, {{1, 10, 4, 22, 25, 13}, {2, 11, 5, 23, 26, 14}, {3, 12, 6, 24, 27, 15}}[[#2]]
        , 2, {{1, 4, 7, 13, 16, 19}, {2, 5, 8, 14, 17, 20}, {3, 6, 9, 15, 18, 21}}[[#2]]
        , 3, {{4, 10, 7, 25, 28, 16}, {5, 11, 8, 26, 29, 17}, {6, 12, 9, 27, 30, 18}}[[#2]]
        , 4, {{10, 1, 7, 22, 19, 28}, {11, 2, 8, 23, 20, 29}, {12, 3, 9, 24, 21, 30}}[[#2]]]) &,
    {nodesindex, nodesdof}], 1];
  factors = Flatten[Table[all[[indexen1[[nodesindex[[i]]]], indexen2[[nodesindex[[i]]]]]][[
      3]], {i, 1, Length[nodesindex]}]];
  LM = SparseArray[MapThread[({#1, #2} → #3) &, {indexi, indexj, factors}]];

  , "H1",
  {addinfo, Kc, Rc, nodesindex, nodesdof} = SMTSchurComplementOfEssentialBC[];
  (*Transformation matrix*)
  indexen1 = SparseArray[MapIndexed[#[[1]] → #2[[1]] &, all, {2}] // Flatten, {SMTNoNodes}];
  indexen2 = SparseArray[MapIndexed[#[[1]] → #2[[2]] &, all, {2}] // Flatten, {SMTNoNodes}];
  indexi = Flatten[Table[{i, i, i, i}, {i, 1, Length[nodesindex]}]];
  indexj = Flatten[MapThread[(Switch[indexen1[[#1]]
```

```
        , 0, Nothing
        , 1, {{10, 7, 4, 1}, {11, 8, 5, 2}, {12, 9, 6, 3}}[[#2]]
        , 2, {{1, 4, 16, 13}, {2, 5, 17, 14}, {3, 6, 18, 15}}[[#2]]
        , 3, {{4, 7, 19, 16}, {5, 8, 20, 17}, {6, 9, 21, 18}}[[#2]]
        , 4, {{7, 10, 22, 19}, {8, 11, 23, 20}, {9, 12, 24, 21}}[[#2]]
        , 5, {{10, 1, 13, 22}, {11, 2, 14, 23}, {12, 3, 15, 24}}[[#2]]
        , 6, {{13, 16, 19, 22}, {14, 17, 20, 23}, {15, 18, 21, 24}}[[#2]]]) &,
    {nodesindex, nodesdof}], 1];
  factors = Flatten[Table[all[[indexen1[[nodesindex[[i]]]], indexen2[[nodesindex[[i]]]]][[
      3]], {i, 1, Length[nodesindex]}]];
  LM = SparseArray[MapThread[({#1, #2} → #3) &, {indexi, indexj, factors}]];

  , "H2S",
  {addinfo, Kc, Rc, nodesindex, nodesdof} = SMTSchurComplementOfEssentialBC[];
  (*Transformation matrix*)
  indexen1 = SparseArray[MapIndexed[#[[1]] → #2[[1]] &, all, {2}] // Flatten, {SMTNoNodes}];
  indexen2 = SparseArray[MapIndexed[#[[1]] → #2[[2]] &, all, {2}] // Flatten, {SMTNoNodes}];
  indexi = Flatten[Table[{i, i, i, i, i, i, i, i}, {i, 1, Length[nodesindex]}]];
  indexj = Flatten[MapThread[(Switch[indexen1[[#1]]
        , 0, Nothing
        , 1, {{10, 7, 4, 1, 31, 28, 25, 34},
          {11, 8, 5, 2, 32, 29, 26, 35}, {12, 9, 6, 3, 33, 30, 27, 36}}[[#2]]
        , 2, {{1, 4, 16, 13, 25, 40, 49, 37}, {2, 5, 17, 14, 26, 41, 50, 38},
          {3, 6, 18, 15, 27, 42, 51, 39}}[[#2]]
        , 3, {{4, 7, 19, 16, 28, 43, 52, 40}, {5, 8, 20, 17, 29, 44, 53, 41},
          {6, 9, 21, 18, 30, 45, 54, 42}}[[#2]]
        , 4, {{7, 10, 22, 19, 31, 46, 55, 43}, {8, 11, 23, 20, 32, 47, 56, 44},
          {9, 12, 24, 21, 33, 48, 57, 45}}[[#2]]
        , 5, {{10, 1, 13, 22, 34, 37, 58, 46}, {11, 2, 14, 23, 35, 38, 59, 47},
          {12, 3, 15, 24, 36, 39, 60, 48}}[[#2]]
        , 6, {{13, 16, 19, 22, 49, 52, 55, 58}, {14, 17, 20, 23, 50, 53, 56, 59},
          {15, 18, 21, 24, 51, 54, 57, 60}}[[#2]]]) &, {nodesindex, nodesdof}], 1];
  factors = Flatten[Table[all[[indexen1[[nodesindex[[i]]]], indexen2[[nodesindex[[i]]]]][[
      3]], {i, 1, Length[nodesindex]}]];
  LM = SparseArray[MapThread[({#1, #2} → #3) &, {indexi, indexj, factors}]];
 ];
 Kfinal = Transpose[LM].Kc.LM;
 Rfinal = Transpose[LM].Rc;
 {Rfinal, Kfinal}
];
```

Generation of selected micro structure meshes

FE22DMicroMeshVoids - Generate RVE mesh for 2D perforated continuum

- FE22DMicroMeshVoids[*local_problem_data*,*specific_micro_data*] initializes micro problem accordingly to the given *local_problem_data* data structure (see documentation) and *specific_micro_data* data structure. The contents of the *specific_micro_data* data structure is defined here.
- specific_micro_data={
  - 1 aRVE
  - 2 bRVE
  - 3 tRVE - RVE thickness
  - 4 rRVE - radius of perforations

- 5 NrRVE - number of elements in radial direction
- 6 NabRVE - number of elements per side of RVE
- 7 micro_element - SMTMakeDll[micro_element_UEC] - element used to discretize the solid domain
- 8 micro_material - material data related to micro element (see `SMTAddDomain`)
- 9 topology - micro mesh type (see `Element Topology`)
- 10 periodic_element - SMTMakeDll[periodic_element_UEC] - element used to impose periodic boundary conditions
- 11 list of solution procedure options - given options are stored in *local_problem_data[[10]]* and interpreted later by the FE2SolveOne function. Current options are:
- "MinSubSteps"->*n* - minimum number of micro sub-steps per one macro step (default *n*=1)
- "MaxSubSteps"->*n* ⇒ defines a maximum number of micro sub-steps per macro step (default 100)
- "Method"->
  "Sensitivity" - full sensitivity analysis
  "Quasi-sensitivity" - sensitivity analysis at the and of macro step
  "SchurMKL" -linearization based on Schur complement evaluated in MKL
  "SchurMMA" -linearization based on Schur complement evaluated in Mathematica
- }

In[33]:=
```
FE22DMicroMeshVoids[locd_, adddata_] := Module[{localdata = locd,
   ConstrainedNodes, aRVE, bRVE, tRVE, NrRVE , NabRVE, rRVE, aRVE2, bRVE2, npx, nodes,
   solidelements, nodes14, nodes23, nodes12, nodes43, X, microelem,
   micromat, periodicelem, topology, p1, sopt},

  X = localdata[[2]];
  p1 = "FE22DMicroMeshVoids" /. adddata;
  p1 = If[MemberQ[{Function, Symbol}, Head[p1]], p1[X], p1];
  localdata[[3]] = p1;
  If[Length[p1] =!= 11
   , SMTUpdateSharedStatus["Local", {3,
     Row[{"specific_micro_data must be a list of length 11.",
       "\nspecific_micro_data: ", adddata}]}];
   Return[$Aborted, Module];
  ];
  If[Cases[p1, _Symbol, {0, Infinity}, 1] =!= {}
   , SMTUpdateSharedStatus["Local", {3,
     Row[{"undefined symbols: ", Cases[p1, _Symbol, {0, Infinity}, 10],
       "\nspecific_micro_data: ", adddata}]}];
   Return[$Aborted, Module];
  ];

  {aRVE, bRVE, tRVE, rRVE, NrRVE ,
    NabRVE, microelem, micromat, topology, periodicelem, sopt} = p1;
  (* micro element can exist at the main directory or taken from the AceShare *)

  (*first create solid mesh only*)
  SMTInputData["Threads" → SMTMacroProblemStatus[[17]],
   "Console" → SMTMacroProblemStatus[[18]]];
  SMTAddDomain[{"micro", microelem[[2]], micromat, "Source" → microelem[[1]]}];

  If[OddQ[NabRVE], ++NabRVE];
  aRVE2 = aRVE / 2; bRVE2 = bRVE / 2;
```

```
Which[
 (* full square if radius is 0 *)
 Chop[rRVE] == 0,
 SMTAddMesh[Polygon[{{-aRVE2, -bRVE2}, {aRVE2, -bRVE2}, {aRVE2, bRVE2}, {-aRVE2, bRVE2}}],
   "micro", topology, {NabRVE, NabRVE}];

 (* square with opening with given radius *)
 , True,
 npx = 100;
 SMTAddMesh[Raster[{
    Table[{
      Interpolation[
        {{0, 0}, {0.25, -aRVE2}, {0.75, -aRVE2}, {1, 0}}, InterpolationOrder → 1][ξ],
      Interpolation[{{0, bRVE2}, {0.25, bRVE2}, {0.75, -bRVE2}, {1, -bRVE2}},
        InterpolationOrder → 1][ξ]
     }, {ξ, 0, 1, 1/(8 npx)}]
    , Table[ rRVE {Cos[ϕ], Sin[ϕ]}, {ϕ, π/2, 3π/2, π/(8 npx)}]}],
  "micro", topology, {2 NabRVE, NrRVE}, "InterpolationOrder" → 1];
 SMTAddMesh[Raster[{
    Table[{
      Interpolation[
        {{0, 0}, {0.25, aRVE2}, {0.75, aRVE2}, {1, 0}}, InterpolationOrder → 1][ξ],
      Interpolation[{{0, -bRVE2}, {0.25, -bRVE2}, {0.75, bRVE2}, {1, bRVE2}},
        InterpolationOrder → 1][ξ]},
     {ξ, 0, 1, 1/(8 npx)}]
    , Table[rRVE {Cos[ϕ], Sin[ϕ]}, {ϕ, -π/2, π/2, π/(8 npx)}]}],
  "micro", topology, {2 NabRVE, NrRVE}, "InterpolationOrder" → 1];
];
SMTAnalysis[];
nodes = SMTNodes[[All, {1, 2, 3}]];
solidelements = SMTElements[[All, 3]];

(*find boundary nodes*)
nodes14 = SMTFindNodes[Line[{{-aRVE2, -bRVE2}, {-aRVE2, bRVE2}}]];
nodes23 = SMTFindNodes[Line[{{aRVE2, -bRVE2}, {aRVE2, bRVE2}}]];
nodes12 = SMTFindNodes[Line[{{-aRVE2, -bRVE2}, {aRVE2, -bRVE2}}]];
nodes43 = SMTFindNodes[Line[{{-aRVE2, bRVE2}, {aRVE2, bRVE2}}]];

ConstrainedNodes = {SMTFindNodes[Point[{-aRVE2, -bRVE2}]][[1]],
  SMTFindNodes[Point[{aRVE2, -bRVE2}]][[1]], SMTFindNodes[Point[{-aRVE2, bRVE2}]][[1]]};

(*create real mesh solid elements + periodic constraint elements*)
SMTInputData["Threads" → SMTMacroProblemStatus[[17]],
 "Console" → SMTMacroProblemStatus[[18]]];

SMTAddDomain[{
```

```
  {"micro", microelem[[2]], micromat, "Source" → microelem[[1]]}
  , {"periodic", periodicelem[[2]], {}, "Source" → periodicelem[[1]]}
 }];

SMTAddMesh["micro", nodes, solidelements];

(* multi-scale elements dependent part *)
Switch[localdata[[1, 5]]

 , "Deformation gradient (plane strain)",
 SMTAddEssentialBoundary[ConstrainedNodes, 1 → 0, 2 → 0];
 SMTSensitivityProblem[{
   {"F11", {"EBC", "∂u/∂F11", "D" → 1}},
   {"F12", {"EBC", "∂u/∂F12", "D" → 1}},
   {"F21", {"EBC", "∂v/∂F21", "D" → 2}},
   {"F22", {"EBC", "∂v/∂F22", "D" → 2}}
  }];

 , "Small strain tensor (plane strain)",
 SMTAddEssentialBoundary[ConstrainedNodes, 1 → 0, 2 → 0];
 SMTSensitivityProblem[{
   {"e11", {"EBC", "∂u/∂e11", "D" → 1}},
   {"e12", {"EBC", "∂u/∂e12", "D" → 1}, {"EBC", "∂v/∂e12", "D" → 2}},
   {"e22", {"EBC", "∂v/∂e22", "D" → 2}}
  }];

 , "Temperature gradient (2D)",
 SMTAddEssentialBoundary[ConstrainedNodes, 1 → 0];
 SMTSensitivityProblem[{
   {"Dθ1", {"EBC", "∂θ/∂Dθ1", "T" → 1}},
   {"Dθ2", {"EBC", "∂θ/∂Dθ2", "T" → 1}}
  }];

 , "Deformation gradient (plane strain)+temperature gradient",
 SMTAddEssentialBoundary[ConstrainedNodes, 1 → 0, 2 → 0, 3 → 0];
 SMTSensitivityProblem[{
   {"F11", {"EBC", "∂u/∂F11", "DT" → 1}},
   {"F12", {"EBC", "∂u/∂F12", "DT" → 1}},
   {"F21", {"EBC", "∂v/∂F21", "DT" → 2}},
   {"F22", {"EBC", "∂v/∂F22", "DT" → 2}},
   {"Dθ1", {"EBC", "∂θ/∂Dθ1", "DT" → 3}},
   {"Dθ2", {"EBC", "∂θ/∂Dθ2", "DT" → 3}}
  }];

];
(*constrains on line 1-4 2-3*)
SMTAddElement[MapThread[{"periodic", {ConstrainedNodes⟦1⟧, #1, ConstrainedNodes⟦2⟧, #2}} &,
  {nodes14 // Rest, nodes23 // Rest}]];
(*constrains on line 1-2 4-3 *)
SMTAddElement[MapThread[{ "periodic", {ConstrainedNodes⟦3⟧, #1, ConstrainedNodes⟦1⟧, #2}} &,
  {nodes43[[2 ;; -2]], nodes12[[2 ;; -2]]}]];

SMTAnalysis[];
SMTSetSolver[5, "IntegerParameters" → {{8, 0}}];
```

```
  localdata[[{8, 9, 10}]] = {ConstrainedNodes, aRVE bRVE tRVE, sopt};
  localdata
 ]
```

MIEL2DMicroMeshVoids - Generate MIEL mesh for 2D perforated continuum

- MIEL2DMicroMeshVoids[*local_problem_data*,*specific_micro_data*] initializes micro problem accordingly to the given *local_problem_data* data structure (see documentation) and *specific_micro_data* data structure. The contents of the *specific_micro_data* data structure is defined here.
- specific_micro_data={
  - 1 {{{x1,y1},r1},{{x2,y2},r2},…} - all discs where {xi,yi} is a center of the i-th disc and ri is the radius of the i-th disc
  - 2 division - approximate number of elements per side
  - 3 micro_element - SMTMakeDll[micro_element_UEC] - element used to discretize the solid domain
  - 4 micro_material - material data related to micro element (see `SMTAddDomain`)
  - 5 topology - micro mesh type (see `Element Topology`)
  - 6 list of solution procedure options - given options are stored in *local_problem_data[[10]]* and interpreted later by the MIELSolveOne function. Current options are:
  - "MinSubSteps"->*n* - minimum number of micro sub-steps per one macro step (default *n*=1)
  - "MaxSubSteps"->*n* ⇒ defines a maximum number of micro sub-steps per macro step (default 100)
  - "Method"->
    "Sensitivity" - full sensitivity analysis
    "Quasi-sensitivity" - sensitivity analysis at the and of macro step
    "SchurMKL" -linearization based on Schur complement evaluated in MKL
  - }

In[34]:=
```
MIEL2DMicroMeshVoids[locd_, adddata_] :=
 Module[{disks, tol, disksdata, ddata, xmin, xmax, ymin, ymax, reg, bmesh, mesh, ,
   MicroElement, meshtype, X, localdata = locd, XIO, uIO, edge, edgepoints,
   postedge, ξ, ndim, senspr, macrotopology, microtopolgy, npx, x, y, us, ns, regcond,
   X1X2, X2X3, X3X4, X4X1, X3X1, n12, n23, n34, n41, n31, microelem, micromat,
   microtopology, meshorder, ne, division, p1, p2, p3, edgedivision, sopt},

  p1 = "MIEL2DMicroMeshVoids" /. adddata;
  p1 = If[MemberQ[{Function, Symbol}, Head[p1]], p1[X], p1];
  localdata[[3]] = p1;
  If[Length[p1] =!= 6
   , SMTUpdateSharedStatus["Local", {3,
     Row[{"specific_micro_data must be a list of length 6.",
       "\nspecific_micro_data: ", adddata}]}];
   Return[$Aborted, Module];
  ];
  If[Cases[p1, _Symbol, {0, Infinity}, 1] =!= {}
   , SMTUpdateSharedStatus["Local", {3,
     Row[{"udefined symbols: ", Cases[p1, _Symbol, {0, Infinity}, 10],
       "\nspecific_micro_data: ", adddata}]}];
   Return[$Aborted, Module];
  ];
  {disksdata, division, microelem, micromat, microtopology, sopt} = p1;
  localdata[[12]] = sopt;

  X = localdata[[2]];
```

```
macrotopology = localdata[[9]];
XIO = localdata[[8]];
ndim = SMTMacroProblemStatus[[1]];
edge = Switch[macrotopology
  , "T1", {{1, 2}, {2, 3}, {3, 1}}
  , "Q1", {{1, 2}, {2, 3}, {3, 4}, {4, 1}}
  , "T2", {{1, 4, 2}, {2, 5, 3}, {3, 6, 1}}
  , "Q2S", {{1, 5, 2}, {2, 6, 3}, {3, 7, 4}, {4, 8, 1}}
  , _, SMTUpdateSharedStatus["Local",
   {3, Row[{"Unsupported macro topology: ", microtopology}]}];
  Return[$Aborted, Module];
 ];
{xmin, xmax, ymin, ymax} =
 {Min[XIO[[All, 1]]], Max[XIO[[All, 1]]], Min[XIO[[All, 2]]], Max[XIO[[All, 2]]]} // N;

(*delete disks with 0 radius and out of element bounding box*)
disksdata = DeleteCases[disksdata,
  _?(Chop[#[[2]]] == 0 || #[[1, 1]] - #[[2]] > xmax || #[[1, 1]] + #[[2]] < xmin ||
      #[[1, 2]] - #[[2]] > ymax || #[[1, 2]] + #[[2]] < ymin &)];
disks = And @@ Map[(x - #[[1, 1]])^2 + (y - #[[1, 2]])^2 ≥ #[[2]]^2 &, disksdata];

SMTInputData["Threads" → SMTMacroProblemStatus[[17]],
 "Console" → SMTMacroProblemStatus[[18]]];
SMTAddDomain[{"micro", microelem[[2]], micromat, "Source" → microelem[[1]]}];

If[disksdata === {} && (macrotopology === "Q1" || macrotopology === "Q2S")

 (*homogenious microstructure - structured mesh*)
 , SMTAddMesh[Raster[Switch[macrotopology
     , "Q1", {XIO[[{1, 2}]], XIO[[{4, 3}]]}
     , "Q2S", {XIO[[{1, 5, 2}]], {XIO[[8]], Total[XIO]/8, XIO[[6]]}, XIO[[{4, 7, 3}]]}
    ]], "micro", microtopology, {division, division}
  ];

 (*nonhomogenious microstructure - unstructured mesh*)
 , regcond = Switch[macrotopology
   , "T1" | "T2",
   X1X2 = XIO[[2]] - XIO[[1]]; n12 = {X1X2[[2]], -X1X2[[1]]};
   X2X3 = XIO[[3]] - XIO[[2]]; n23 = {X2X3[[2]], -X2X3[[1]]};
   X3X1 = XIO[[1]] - XIO[[3]]; n31 = {X3X1[[2]], -X3X1[[1]]};
   disks && ({x, y} - XIO[[1]]).n12 ≤ 0 && ({x, y} - XIO[[2]]).n23 ≤ 0 && ({x, y} - XIO[[3]]).n31 ≤ 0
   , "Q1" | "Q2S",
   X1X2 = XIO[[2]] - XIO[[1]]; n12 = {X1X2[[2]], -X1X2[[1]]};
   X2X3 = XIO[[3]] - XIO[[2]]; n23 = {X2X3[[2]], -X2X3[[1]]};
   X3X4 = XIO[[4]] - XIO[[3]]; n34 = {X3X4[[2]], -X3X4[[1]]};
   X4X1 = XIO[[1]] - XIO[[4]]; n41 = {X4X1[[2]], -X4X1[[1]]};
   disks && ({x, y} - XIO[[1]]).n12 ≤ 0 &&
    ({x, y} - XIO[[2]]).n23 ≤ 0 && ({x, y} - XIO[[3]]).n34 ≤ 0 && ({x, y} - XIO[[4]]).n41 ≤ 0
  ];
 reg = ImplicitRegion[regcond, {x, y}];
 bmesh = Check[ToBoundaryMesh[reg, "BoundaryMeshGenerator" → "RegionPlot",
    "MaxBoundaryCellMeasure" → Min[xmax - xmin, ymax - ymin]/division]
   , SMTUpdateSharedStatus["Local", {3, Row[{
       "Regions not valid for meshing with ToBoundaryMesh or to low max mesh density.",
```

```
        "\nMacro element=", localdata[[6]], " Macro element nodes=", XIO,
        " MaxBoundaryCellMeasure=", Min[xmax - xmin, ymax - ymin] / division}]}];
    Return[$Aborted, Module];
   ];
 meshorder = Switch[
    microtopology
    , "T1", 1
    , "T2", 2
    , _,
    SMTUpdateSharedStatus["Local",
     {3, {"Unsupported micro topology for unstructured micro mesh: ", microtopology}}];
    Return[$Aborted, Module];
   ];
 mesh = ToElementMesh[bmesh, "MeshElementType" → TriangleElement,
    "MeshOrder" → meshorder, MaxCellMeasure → Min[xmax - xmin, ymax - ymin] / division];
 SMTAddMesh[mesh, {microtopology -> "micro"}];

];

senspr = Flatten[Table[
    {us = ToString[Switch[u, 1, "u", 2, "v", 3, "w"]];
     ns = ToString[j];
     us <> "M" <> ns, {"EBC", "∂" <> us <> "/∂" <> us <> "M" <> ns, "D" → u}}
    , {j, 1, Length[XIO]}
    , {u, 1, 2}
   ], 1];
SMTSensitivityProblem[senspr, "Order" → 2];

SMTAnalysis[];
SMTSetSolver[5, "IntegerParameters" → {{8, 0}}];

tol = SMTRData["MinElementSize"] / 10.;
edgepoints = Switch[macrotopology
   , "T1" | "Q1"
       , edgedivision = 1;
       Map[XIO[[#]] &, edge]
   , "T2" | "Q2S"
       (*in the case of quadratic macro element make edge with linear segments*)
       , p1 = {2 ξ^2 - 3 ξ + 1, -4 ξ^2 + 4 ξ, 2 ξ^2 - ξ};
       p2 = Map[p1.XIO[[#]] &, edge];
       (*p3 -
    the largest step on edge so that there is at lest 10 points per smallest element*)
       edgedivision = Ceiling[Min[Map[Norm[XIO[[#[[3]]]] - XIO[[#[[1]]]]] &, edge]] / tol];
       Map[Table[#, {ξ, 0., 1., 1. / edgedivision}] &, p2]
 ];
localdata[[11]] = edgedivision;
SMTAddEssentialBoundary[Or @@ Map[Line[#, "D", tol] &, edgepoints], 1 → 0, 2 → 0];

(*correction of postprocessing for the case when corners are inside voids*)
postedge = Switch[macrotopology
   , "T1", {{1, -3}, {-1, 2}, {-2, 3}}
   , "Q1", {{1, -4}, {-1, 2}, {-2, 3}, {-3, 4}}
   , "T2", {{1, -3}, {-1, 2}, {-2, 3}, 1, 2, 3}
   , "Q2S", {{1, -4}, {-1, 2}, {-2, 3}, {-3, 4}, 1, 2, 3, 4}
```

```
 ];
MapIndexed[(
   If[SMTFindNodes[Point[#]] === {}
    , p1 = #2[[1]];
    p3 = postedge[[p1]];
    If[IntegerQ[p3]
     (*middle point*)
     , p2 = SMTFindNodes[edgepoints[[p3]]];
     If[Length[p2] < 4
      , SMTUpdateSharedStatus["Local", {1, Row[{"To dense macro mesh. Not enought
             macro nodes on edge to support macro middle node.",
          "\nFound nodes on edge=", p2,
          "\nMacro element=",
          localdata[[6]], " Macro element nodes=", XIO}]}];
      p2 = Null;
      , p2 = Nearest[SMTNodeData[p2, "X"], #];
      p2 = {p2, # + (# - p2)};
     ];
     (*corner point*)
     , p2 = {SMTFindNodes[Line[If[p3[[1]] > 0, edgepoints[[p3[[1]]]]
          , edgepoints[[-p3[[1]]]] // Reverse]]]
       , SMTFindNodes[Line[If[p3[[2]] > 0, edgepoints[[p3[[2]]]]
          , edgepoints[[-p3[[2]]]] // Reverse]]]
      };
     If[Length[p2[[1]]] < 2 || Length[p2[[2]]] < 2
      , SMTUpdateSharedStatus["Local", {1, Row[{"To dense macro mesh. Not enought
             macro nodes on edge to support macro corner node.",
          "\nFound nodes on edges=", p2,
          "\nMacro element=",
          localdata[[6]], " Macro element nodes=", XIO}]}];
      p2 = Null;
      , p2 = SMTNodeData[p2[[All, 1]], "X"]
     ];
    ];
    If[p2 =!= Null, localdata[[10, p1]] = p2];
   ]) &
 , XIO];

 localdata
]
```

FE23DMicroMeshVoids - Generate RVE mesh for 3D perforated continuum

- FE23DMicroMeshVoids[*local_problem_data*,*specific_micro_data*] initializes micro problem accordingly to the given *local_problem_data* data structure (see documentation) and *specific_micro_data* data structure. The contents of the *specific_micro_data* data structure is defined here.
- specific_micro_data={
  - 1 aRVE
  - 2 bRVE
  - 3 cRVE
  - 4 rRVE - radius of perforations
  - 5 NrRVE - number of elements in radial direction

- 6 NabRVE - number of elements per side of RVE
- 7 micro_element - SMTMakeDll[micro_element_UEC] - element used to discretize the solid domain
- 8 micro_material - material data related to micro element (see SMTAddDomain)
- 9 topology - micro mesh type (see Element Topology)
- 10 periodic_element - SMTMakeDll[periodic_element_UEC] - element used to impose periodic boundary conditions
- for structured meshes perodic elements connect master node with slave node on oposite surface
- for unstructured meshes the periodic element connects master node with slave triangle on oposite surface
- 11:
- "" - mesh is structured, thus each surface node has a matching node at the oposide surface
- SMTMakeDll[surface_element_UEC] - dummy element for surface triangulation used to impose perodic boundary
- 12 list of solution procedure options - given options are stored in *local_problem_data[[10]]* and interpreted later by the FE2SolveOne function. Current options are:
- "MinSubSteps"->*n* - minimum number of micro sub-steps per one macro step (default *n*=1)
- "MaxSubSteps"->*n* ⇒ defines a maximum number of micro sub-steps per macro step (default 100)
- "Method"->

"Sensitivity"	- full sensitivity analysis
"Quasi-sensitivity"	- sensitivity analysis at the and of macro step
"SchurMKL"	-linearization based on Schur complement evaluated in MKL
"SchurMMA"	-linearization based on Schur complement evaluated in Mathematica

- }

Shared variables - global at macro and micro level

- MicroMeshData = {*aRVE*, *bRVE*, *mesh_type*,...} (the value can be also a function of spatial coordinates)
- MicroDomainData = {*micro_element_code*, *micro_element_material_data*} (the value can be also a function of spatial coordinates)
- ConstrainDomainData = {*constrain_element_code*, *constrain_element_material_data*} (the value can be also a function of spatial coordinates)

In[35]:=
```
FE23DMicroMeshVoids[locd_, adddata_] :=
 Module[{localdata = locd, ConstrainedNodes, aRVE, bRVE, cRVE,
   tRVE, NrRVE , NabRVE, rRVE, aRVE2, bRVE2, cRVE2, npx, nodes,
   solidelements, nodes14, nodes23, nodes12, nodes43, X, microelem, macromat,
   periodicelem, topology, p1, nodes4321, nodes1265, nodes2376, nodes3487,
   nodes4158, nodes5678, x, y, z, mesh, cond, reg, bmesh, meshorder, boundaryelem,
   nodes1265X, nodes2376X, nodes5678X, projection1265X, projection2376X,
   projection5678X, nodes4321X, nodes3487X, nodes4158X, sopt, structured, maxlength},

  X = localdata[[2]];
  p1 = "FE23DMicroMeshVoids" /. adddata;
  p1 = If[MemberQ[{Function, Symbol}, Head[p1]], p1[X], p1];
  localdata[[3]] = p1;
  If[Length[p1] =!= 12
   , SMTUpdateSharedStatus["Local", {3,
     Row[{"specific_micro_data must be a list of length 12.",
       "\nspecific_micro_data: ", adddata}]}];
   Return[$Aborted, Module];
  ];
  If[Cases[p1, _Symbol, {0, Infinity}, 1] =!= {}
   , SMTUpdateSharedStatus["Local", {3,
     Row[{"udefined symbols: ", Cases[p1, _Symbol, {0, Infinity}, 10],
```

```
     "\nspecific_micro_data: ", adddata}]}];
 Return[$Aborted, Module];
];

{aRVE, bRVE, cRVE, rRVE, NrRVE , NabRVE,
  microelem, macromat, topology, periodicelem, boundaryelem, sopt} = p1;
structured = boundaryelem === "";

(*first create solid mesh only*)
SMTInputData["Threads" → SMTMacroProblemStatus[[17]],
 "Console" → SMTMacroProblemStatus[[18]]];
If[structured
 , SMTAddDomain[{"micro", microelem[[2]], macromat, "Source" → microelem[[1]]}];
 , SMTAddDomain[
   {"micro", microelem[[2]], macromat, "Source" → microelem[[1]]},
   {"boundarysurface", boundaryelem[[2]], {}, "Source" → boundaryelem[[1]]}
  ];
];

If[OddQ[NabRVE], ++NabRVE];
aRVE2 = aRVE / 2; bRVE2 = bRVE / 2; cRVE2 = cRVE / 2;

Which[
 structured,
 (* full square - homogenious*)
 SMTAddMesh[Hexahedron[{{-aRVE2, -bRVE2, -cRVE2}, {aRVE2, -bRVE2, -cRVE2},
     {aRVE2, bRVE2, -cRVE2}, {-aRVE2, bRVE2, -cRVE2}, {-aRVE2, -bRVE2, cRVE2},
     {aRVE2, -bRVE2, cRVE2}, {aRVE2, bRVE2, cRVE2}, {-aRVE2, bRVE2, cRVE2}}],
   "micro", topology, {NabRVE, NabRVE, NabRVE}];

 (* full square if radius is 0 *)
 , Not[structured] && Chop[rRVE] == 0,
 SMTAddMesh[Hexahedron[{{-aRVE2, -bRVE2, -cRVE2}, {aRVE2, -bRVE2, -cRVE2},
     {aRVE2, bRVE2, -cRVE2}, {-aRVE2, bRVE2, -cRVE2}, {-aRVE2, -bRVE2, cRVE2},
     {aRVE2, -bRVE2, cRVE2}, {aRVE2, bRVE2, cRVE2}, {-aRVE2, bRVE2, cRVE2}}],
   "micro", topology, {NabRVE, NabRVE, NabRVE}, "BodyID" → "body",
   "BoundaryDomainID" → "boundarysurface"];

 (* cube with opening sphere with given radius *)
 , Not[structured],
 cond =
  {-aRVE2 ≤ x ≤ aRVE2 && -bRVE2 ≤ y ≤ bRVE2 && -cRVE2 ≤ z ≤ cRVE2 && x^2 + y^2 + z^2 ≥ rRVE^2};
 reg = ImplicitRegion[cond, {x, y, z}];
 bmesh = Check[ToBoundaryMesh[reg, "MaxCellMeasure" → {"Area" → aRVE bRVE / NabRVE^2}]
   , SMTUpdateSharedStatus["Local",
    {3, Row[{"Regions not valid for meshing with ToBoundaryMesh.",
       "\nMacro element=", localdata[[6]]}]}];
   Return[$Aborted, Module];
  ];
 meshorder = Switch[
   topology
   , "O1", 1
   , "O2", 2
   , _,
```

```
    SMTUpdateSharedStatus["Local",
     {3, {"Unsupported micro topology for unstructured micro mesh: ", topology}}];
    Return[$Aborted, Module];
   ];
 mesh = ToElementMesh[bmesh, "MeshElementType" → TetrahedronElement,
    "MeshOrder" → meshorder, "MaxCellMeasure" → {"Volume" → aRVE bRVE cRVE / NabRVE^3}];
 SMTAddMesh[mesh, {topology -> "micro"}, "BodyID" → "body",
  "BoundaryDomainID" → "boundarysurface"];

 , True, SMTUpdateSharedStatus["Local",
  {3, {"Unsupported combination of mesh and perodic boundary elements: ", topology}}];
 Return[$Aborted, Module];
];

SMTAnalysis[];

nodes = SMTNodes[[All, {1, 2, 3, 4}]];
solidelements = Select[SMTElements[[All, 3]], Length[#] > 3 &];

ConstrainedNodes = {
  SMTFindNodes[Point[{aRVE2, -bRVE2, -cRVE2}]][[1]],
  SMTFindNodes[Point[{aRVE2, bRVE2, -cRVE2}]][[1]],
  SMTFindNodes[Point[{-aRVE2, bRVE2, -cRVE2}]][[1]],
  SMTFindNodes[Point[{-aRVE2, -bRVE2, -cRVE2}]][[1]],
  SMTFindNodes[Point[{aRVE2, -bRVE2, cRVE2}]][[1]],
  SMTFindNodes[Point[{aRVE2, bRVE2, cRVE2}]][[1]],
  SMTFindNodes[Point[{-aRVE2, bRVE2, cRVE2}]][[1]],
  SMTFindNodes[Point[{-aRVE2, -bRVE2, cRVE2}]][[1]]};

(*find boundary nodes on master surfaces*)
nodes1265 = Select[SMTFindNodes["X" == aRVE2 &], FreeQ[ConstrainedNodes, #] &];
nodes2376 = Select[SMTFindNodes["Y" == bRVE2 &], FreeQ[ConstrainedNodes, #] &];
nodes5678 = Select[SMTFindNodes["Z" == cRVE2 &], FreeQ[ConstrainedNodes, #] &];
nodes1265X = SMTNodeData[nodes1265, "X"];
nodes2376X = SMTNodeData[nodes2376, "X"];
nodes5678X = SMTNodeData[nodes5678, "X"];

(*projection of master nodes to slave surface*)
projection1265X = Map[{-aRVE2, #[[2]], #[[3]]} &, nodes1265X];
projection2376X = Map[{#[[1]], -bRVE2, #[[3]]} &, nodes2376X];
projection5678X = Map[{#[[1]], #[[2]], -cRVE2} &, nodes5678X];

If[structured
 (*find nodes on slave side *)
 , nodes3487 = Flatten[Map[SMTFindNodes[Point[#]] &, projection1265X]];
 nodes4158 = Flatten[Map[SMTFindNodes[Point[#]] &, projection2376X]];
 nodes4321 = Flatten[Map[SMTFindNodes[Point[#]] &, projection5678X]];
 (*find trangle on slave side *)
 , nodes3487 = Map[SMTPointLocation[#, "boundarysurface", 10^-8][[2]] &, projection1265X];
 nodes4158 = Map[SMTPointLocation[#, "boundarysurface", 10^-8][[2]] &, projection2376X];
 nodes4321 = Map[SMTPointLocation[#, "boundarysurface", 10^-8][[2]] &, projection5678X];
];

(*create real mesh solid elements + periodic constraint elements*)
```

```
SMTInputData["Threads" → SMTMacroProblemStatus[[17]],
 "Console" → SMTMacroProblemStatus[[18]]];
SMTAddDomain[{
  {"micro", microelem[[2]], macromat, "Source" → microelem[[1]]}
  , {"periodic", periodicelem[[2]], {}, "Source" → periodicelem[[1]]}
 }];

SMTAddMesh["micro", nodes, solidelements];

(* multi-scale elements dependent part *)
Switch[localdata[[1, 5]]
 , "Deformation gradient (3D)",
 SMTAddEssentialBoundary[ConstrainedNodes, 1 → 0, 2 → 0, 3 → 0];
 SMTSensitivityProblem[{
   {"F11", {"EBC", "∂u/∂F11", "D" → 1}},
   {"F12", {"EBC", "∂u/∂F12", "D" → 1}},
   {"F13", {"EBC", "∂u/∂F13", "D" → 1}},
   {"F21", {"EBC", "∂v/∂F21", "D" → 2}},
   {"F22", {"EBC", "∂v/∂F22", "D" → 2}},
   {"F23", {"EBC", "∂v/∂F23", "D" → 2}},
   {"F31", {"EBC", "∂w/∂F31", "D" → 3}},
   {"F32", {"EBC", "∂w/∂F32", "D" → 3}},
   {"F33", {"EBC", "∂w/∂F33", "D" → 3}}
  }];
 , "Small strain tensor (3D)",
 SMTAddEssentialBoundary[ConstrainedNodes, 1 → 0, 2 → 0, 3 → 0];
 SMTSensitivityProblem[{
   {"e11", {"EBC", "∂u/∂e11", "D" → 1}},
   {"e12", {"EBC", "∂u/∂e12", "D" → 1}, {"EBC", "∂v/∂e12", "D" → 2}},
   {"e13", {"EBC", "∂u/∂e13", "D" → 1}, {"EBC", "∂w/∂e13", "D" → 3}},
   {"e22", {"EBC", "∂v/∂e22", "D" → 2}},
   {"e23", {"EBC", "∂v/∂e23", "D" → 2}, {"EBC", "∂w/∂e23", "D" → 3}},
   {"e33", {"EBC", "∂w/∂e33", "D" → 3}}
  }];
];

(*constrained node is removed from list of all points on considered surface*)
SMTAddElement[
 MapThread[{"periodic", {ConstrainedNodes⟦1⟧, #1, ConstrainedNodes⟦4⟧, #2} // Flatten} &,
  {nodes1265, nodes3487}]];
SMTAddElement[MapThread[{"periodic", {ConstrainedNodes⟦3⟧, #1, ConstrainedNodes⟦4⟧, #2} //
     Flatten} &, {nodes2376, nodes4158}]];
SMTAddElement[MapThread[{"periodic", {ConstrainedNodes⟦5⟧, #1, ConstrainedNodes⟦1⟧, #2} //
     Flatten} &, {nodes5678, nodes4321}]];

SMTAnalysis[];
SMTSetSolver[5, "IntegerParameters" → {{8, 0}}];

localdata[[{8, 9, 10}]] = {ConstrainedNodes, aRVE bRVE cRVE, sopt};
localdata
]
```

MIEL3DMicroMeshVoids - Generate MIEL mesh for 3D perforated continuum

- MIEL3DMicroMeshVoids[*local_problem_data, local_problem_data, additional_micro_data*] - initialize micro problem for given *local_problem_data* and *additional_micro_data*

- specific_micro_data={
  - 1 {{{x1,y1,z1},r1},{{x2,y2,z2},r2},…} - all spheres where {xi,yi,zi} is a center of the i-th voids and ri is the radius of the i-th void
  - 2 division - approximate number of elements per side
  - 3 micro_element - SMTMakeDll[micro_element_UEC] - element used to discretize the solid domain
  - 4 micro_material - material data related to micro element (see `SMTAddDomain`)
  - 5 topology - micro mesh type (see `Element Topology`)
  - 6 list of solution procedure options - given options are stored in *local_problem_data[[10]]* and interpreted later by the MIELSolveOne function. Current options are:
  - "MinSubSteps"->*n* - minimum number of micro sub-steps per one macro step (default *n*=1)
  - "MaxSubSteps"->*n* ⇒ defines a maximum number of micro sub-steps per macro step (default 100)
  - "Method"->
    "Sensitivity" - full sensitivity analysis
    "Quasi-sensitivity" - sensitivity analysis at the and of macro step
    "SchurMKL" -linearization based on Schur complement evaluated in MKL
  - }

In[36]:=
```
MIEL3DMicroMeshVoids[locd_, adddata_] :=
 Module[{disks, tol, disksdata, ddata, xmin, xmax, ymin, ymax, reg, bmesh, mesh, ,
   MicroElement, meshtype, X, localdata = locd, XIO, uIO, edge, edgepoints, postedge, ξ,
   ndim, senspr, macrotopology, microtopolgy, npx, x, y, us, ns, regcond, X1X2, X2X3, X3X4,
   X4X1, X3X1, n12, n23, n34, n41, n31, microelem, micromat, microtopology, meshorder,
   ne, division, p1, p2, p3, edgedivision, mesharea, surfacenodes, allnodes, zmin, zmax,
   z, n1, n2, n3, n4, n5, n6, tol1, bm, s1, s2, s3, s4, XIObord, bmesh2, pointcircle,
   preg, pdisc, S, r, diskaround, diskRM, pins, lineelem, newbound, ap, api, bmap,
   newboundap, n, XIOc, XIO1, XIO2, XIO3, XIO4, sopt, maxlength, balls, regionholes},
  PutAppend[locd, adddata, "analysis.txt"];
  p1 = "MIEL3DMicroMeshVoids" /. adddata;
  p1 = If[MemberQ[{Function, Symbol}, Head[p1]], p1[X], p1];
  localdata[[3]] = p1;
  If[Length[p1] =!= 6
   , SMTUpdateSharedStatus["Local", {3,
     Row[{"specific_micro_data must be a list of length 6.",
       "\nspecific_micro_data: ", adddata}]}];
   Return[$Aborted, Module];
  ];
  If[Cases[p1, _Symbol, {0, Infinity}, 1] =!= {}
   , SMTUpdateSharedStatus["Local", {3,
     Row[{"udefined symbols: ", Cases[p1, _Symbol, {0, Infinity}, 10],
       "\nspecific_micro_data: ", adddata}]}];
   Return[$Aborted, Module];
  ];
  PutAppend[1, "analysis.txt"];
  {disksdata, division, microelem, micromat, microtopology, sopt} = p1;
  localdata[[12]] = sopt;

  X = localdata[[2]];
  macrotopology = localdata[[9]];
  XIO = localdata[[8]];
  ndim = SMTMacroProblemStatus[[1]];
  edge = Switch[macrotopology
```

```
  , "O1", {{1, 4, 2}, {1, 2, 3}, {2, 4, 3}, {4, 1, 3}}
  , "O2", {{1, 4, 2}, {1, 2, 3}, {2, 4, 3}, {4, 1, 3}}
  , "H1", {{4, 3, 2, 1}, {1, 2, 6, 5}, {2, 3, 7, 6}, {3, 4, 8, 7}, {4, 1, 5, 8}, {5, 6, 7, 8}}
  , "H2S", {{4, 3, 2, 1}, {1, 2, 6, 5}, {2, 3, 7, 6}, {3, 4, 8, 7}, {4, 1, 5, 8}, {5, 6, 7, 8}}
  , _, SMTUpdateSharedStatus["Local",
   {3, Row[{"Unsupported macro topology: ", microtopology}]}];
  Return[$Aborted, Module];
 ];
{xmin, xmax, ymin, ymax, zmin, zmax} = {Min[XIO[[All, 1]]], Max[XIO[[All, 1]]],
   Min[XIO[[All, 2]]], Max[XIO[[All, 2]]], Min[XIO[[All, 3]]], Max[XIO[[All, 3]]]} // N;
maxlength = Min[xmax - xmin, ymax - ymin, zmax - zmin] / division;

(*delete sphere with 0 radius and out of element bounding box*)
disksdata = DeleteCases[disksdata,
  _?(Chop[#[[2]]] == 0 ||
       #[[1, 1]] - #[[2]] > xmax || #[[1, 1]] + #[[2]] < xmin ||
       #[[1, 2]] - #[[2]] > ymax || #[[1, 2]] + #[[2]] < ymin ||
       #[[1, 3]] - #[[2]] > zmax || #[[1, 3]] + #[[2]] < zmin &)];
disks =
 And @@ Map[(x - #[[1, 1]])^2 + (y - #[[1, 2]])^2 + (z - #[[1, 3]])^2 ≥ #[[2]]^2 &, disksdata];

PutAppend[{2, SMTMacroProblemStatus[[17]], SMTMacroProblemStatus[[18]]}, "analysis.txt"];
SMTInputData["Threads" → SMTMacroProblemStatus[[17]],
 "Console" → SMTMacroProblemStatus[[18]]];
PutAppend[3, "analysis.txt"];
(*create a mesh*)
If[disksdata === {}

 (*homogenious microstructure*)
 , SMTAddDomain[{"micro", microelem[[2]], micromat, "Source" → microelem[[1]]}];
 meshorder = Switch[microtopology, "O1" | "H1", 1, "O2S" | "H2S", 2];
 Switch[macrotopology
  , "H1" | "H2S",
  SMTAddMesh[Hexahedron[XIO[[1 ;; 8]]],
   "micro", microtopology, {division, division, division}]
  , "O1" | "O2",
  mesh = ToElementMesh[Tetrahedron[XIO[[1 ;; 4]]], "MeshElementType" → TetrahedronElement,
    "MeshOrder" → meshorder, MaxCellMeasure → {"Length" -> maxlength}];
  SMTAddMesh[mesh, {microtopology -> "micro"}]
 ];

 (* nonhomogenious microstructure*)
 , reg = Switch[macrotopology
   , "O1",
   BoundaryMeshRegion[XIO, Polygon[{{1, 2, 3}, {2, 4, 3}, {4, 1, 3}, {1, 4, 2}}]]
   , "O1",
   BoundaryMeshRegion[XIO,
    Polygon[{{1, 5, 2, 6, 3, 7}, {2, 9, 4, 10, 3, 6}, {4, 8, 1, 7, 3, 10}, {1, 8, 4, 9, 2, 5}}]]
   , "H1",
   BoundaryMeshRegion[XIO,
    Polygon[{{1, 4, 3, 2}, {2, 3, 7, 6}, {3, 4, 8, 7}, {4, 1, 5, 8}, {1, 2, 6, 5}, {5, 6, 7, 8}}]]
   , "H2S",
   BoundaryMeshRegion[XIO,
    Polygon[{{1, 12, 4, 11, 3, 10, 2, 9}, {2, 10, 3, 15, 7, 18, 6}, {3, 11, 4, 16, 8, 19, 7, 15},
      {4, 12, 1, 13, 5, 20, 8, 16}, {1, 9, 2, 14, 6, 17, 5, 13}, {5, 17, 6, 18, 7, 19, 8, 20}}]]
```

```
  ];
 balls = Map[BoundaryDiscretizeRegion[Ball[#[[1]], #[[2]]],
     MaxCellMeasure → {"Length" → maxlength}, Method → "MarchingCubes"] &, disksdata];
 regionholes = Pick[disksdata, Map[RegionWithin[reg, #] &, balls]];
 reg = Fold[RegionDifference, reg, balls];
 mesh = ToElementMesh[reg, "MeshOrder" → 1, "MaxCellMeasure" → {"Length" → maxlength},
   "MaxBoundaryCellMeasure" → {"Length" → maxlength}, "BoundaryMeshGenerator" → "RegionPlot",
   "ImproveBoundaryPosition" → False, "MeshElementType" → TetrahedronElement,
   "MeshOrder" → meshorder, "RegionHoles" -> regionholes[[All, 1]]];
 SMTAddDomain[{"micro", microelem[[2]], micromat, "Source" → microelem[[1]]}];
 SMTAddMesh[mesh, {microtopology -> "micro"}];
];

senspr = Flatten[Table[
   {us = ToString[Switch[u, 1, "u", 2, "v", 3, "w"]];
    ns = ToString[j];
    us <> "M" <> ns, {"EBC", "∂" <> us <> "/∂" <> us <> "M" <> ns, "D" → u}}
   , {j, 1, Length[XIO]}
   , {u, 1, 3}
  ], 1];
SMTSensitivityProblem[senspr, "Order" → 2];
PutAppend[4, "analysis.txt"];
SMTAnalysis[];
PutAppend[5, "analysis.txt"];
SMTSetSolver[5, "IntegerParameters" → {{8, 0}}];
tol = SMTRData["MinElementSize"] / 100.;
allnodes = SMTNodeData[All, "NodeIndex"];
surfacenodes =
 DeleteDuplicates[Flatten[Map[SMTFindNodes[Polygon[XIO[[#]], "D", tol]] &, edge], 1]];
SMTAddEssentialBoundary[Or @@ surfacenodes, 1 → 0, 2 → 0, 3 → 0];
PutAppend[{6, localdata}, "analysis.txt"];
localdata
]
```

FE22DMicroMeshVoids2materials - Generate RVE mesh for 2D two-material continuum

- FE22DMicroMeshVoids[*local_problem_data*,*specific_micro_data*] initializes micro problem accordingly to the given *local_problem_data* data structure (see documentation) and *specific_micro_data* data structure. The contents of the *specific_micro_data* data structure is defined here.
- specific_micro_data={
  - 1 aRVE
  - 2 bRVE
  - 3 tRVE - RVE thickness
  - 4 rRVE - radius of perforations
  - 5 NrRVE - number of elements in radial direction
  - 6 NabRVE - number of elements per side of RVE
  - 7 micro_element - SMTMakeDll[micro_element_UEC] - element used to discretize the solid domain
  - 8 micro_material - material data related to micro element (see `SMTAddDomain`)
  - 9 topology - micro mesh type (see `Element Topology`)
  - 10 periodic_element - SMTMakeDll[periodic_element_UEC] - element used to impose periodic boundary conditions

- 11 list of solution procedure options - given options are stored in *local_problem_data[[10]]* and interpreted later by the FE2SolveOne function. Current options are:
- "MinSubSteps"->*n* - minimum number of micro sub-steps per one macro step (default *n*=1)
- "MaxSubSteps"->*n* ⇒ defines a maximum number of micro sub-steps per macro step (default 100)
- "Method"->
  - "Sensitivity" - full sensitivity analysis
  - "Quasi-sensitivity" - sensitivity analysis at the and of macro step
  - "SchurMKL" -linearization based on Schur complement evaluated in MKL
  - "SchurMMA" -linearization based on Schur complement evaluated in Mathematica
- }

In[37]:=
```
FE22DMicroMeshVoids2materials[locd_, adddata_] :=
 Module[{localdata = locd, ConstrainedNodes, aRVE,
   bRVE, tRVE, NrRVE , NabRVE, rRVE, aRVE2, bRVE2, npx, nodes,
   solidelements, nodes14, nodes23, nodes12, nodes43, X, microelem, micromat,
   periodicelem, topology, p1, sopt, a1, b1, imp, nx , ny, microelemA, micromatA,
   microelemB, micromatB, nodesA, nodesB, solidelementsA, solidelementsB},

  X = localdata[[2]];
  p1 = "FE22DMicroMeshVoids2materials" /. adddata;
  p1 = If[MemberQ[{Function, Symbol}, Head[p1]], p1[X], p1];
  localdata[[3]] = p1;
  If[Length[p1] =!= 15
   , SMTUpdateSharedStatus["Local", {3,
     Row[{"specific_micro_data must be a list of length 15.",
       "\nspecific_micro_data: ", adddata}]}];
   Return[$Aborted, Module];
  ];
  If[Cases[p1, _Symbol, {0, Infinity}, 1] =!= {}
   , SMTUpdateSharedStatus["Local", {3,
     Row[{"undefined symbols: ", Cases[p1, _Symbol, {0, Infinity}, 10],
       "\nspecific_micro_data: ", adddata}]}];
   Return[$Aborted, Module];
  ];

  {aRVE, bRVE, tRVE, a1, b1, imp, nx , ny, microelemA,
    micromatA, microelemB, micromatB, topology, periodicelem, sopt} = p1;
  (* micro element can exist at the main directory or taken from the AceShare *)

  (*first create solid mesh only*)
  SMTInputData["Threads" → SMTMacroProblemStatus[[17]],
   "Console" → SMTMacroProblemStatus[[18]]];
  SMTAddDomain[{{"microA", microelemA[[2]], micromatA, "Source" → microelemA[[1]]},
    {"microB", microelemB[[2]], micromatB, "Source" → microelemB[[1]]}}];

  aRVE2 = aRVE / 2; bRVE2 = bRVE / 2;

  SMTAddMesh[
   Polygon[{{-aRVE2, -bRVE2}, {aRVE2, -bRVE2}, {aRVE2, -bRVE2 + a1}, {-aRVE2, -bRVE2 + a1}}],
   "microA", topology, {nx, ny}];
  SMTAddMesh[Polygon[{{-aRVE2, bRVE2 - a1}, {aRVE2, bRVE2 - a1},
     {aRVE2, bRVE2}, {-aRVE2, bRVE2}}], "microA", topology, {nx, ny}];
  SMTAddMesh[Polygon[{{-aRVE2, -bRVE2 + a1}, {-aRVE2 + a1, -bRVE2 + a1},
```

```
    {-aRVE2 + a1, bRVE2 - a1}, {-aRVE2, bRVE2 - a1}}], "microA", topology, {ny, nx - ny * 2}];
SMTAddMesh[Polygon[{{aRVE2 - a1, -bRVE2 + a1}, {aRVE2, -bRVE2 + a1}, {aRVE2, bRVE2 - a1},
    {aRVE2 - a1, bRVE2 - a1}}], "microA", topology, {ny, nx - ny * 2}];
SMTAddMesh[Polygon[{{-aRVE2 + a1, -b1 / 2}, {0, -b1 / 2 + imp}, {0, b1 / 2 - imp},
    {-aRVE2 + a1, b1 / 2}}], "microB", topology, {(nx - ny * 2) / 2, ny * 2}];
SMTAddMesh[Polygon[{{0, -b1 / 2 + imp}, {aRVE2 - a1, -b1 / 2}, {aRVE2 - a1, b1 / 2},
    {0, b1 / 2 - imp}}], "microB", topology, {(nx - ny * 2) / 2, ny * 2}];

SMTAnalysis[];

nodes = SMTNodes[[All, {1, 2, 3}]];
solidelements = SMTElements;
solidelementsA = TakeWhile [solidelements, #[[2]] == 1 &][[All, 3]];
solidelementsB = solidelements[[Length[solidelementsA] + 1 ;;, 3]];

(*find boundary nodes*)
nodes14 = SMTFindNodes[Line[{{-aRVE2, -bRVE2}, {-aRVE2, bRVE2}}]];
nodes23 = SMTFindNodes[Line[{{aRVE2, -bRVE2}, {aRVE2, bRVE2}}]];
nodes12 = SMTFindNodes[Line[{{-aRVE2, -bRVE2}, {aRVE2, -bRVE2}}]];
nodes43 = SMTFindNodes[Line[{{-aRVE2, bRVE2}, {aRVE2, bRVE2}}]];

ConstrainedNodes = {SMTFindNodes[Point[{-aRVE2, -bRVE2}]][[1]],
  SMTFindNodes[Point[{aRVE2, -bRVE2}]][[1]], SMTFindNodes[Point[{-aRVE2, bRVE2}]][[1]]};

(*create real mesh solid elements + periodic constraint elements*)
SMTInputData["Threads" → SMTMacroProblemStatus[[17]],
 "Console" → SMTMacroProblemStatus[[18]]];

SMTAddDomain[{
  {"microA", microelemA[[2]], micromatA, "Source" → microelemA[[1]]},
  {"microB", microelemB[[2]], micromatB, "Source" → microelemB[[1]]},
  {"periodic", periodicelem[[2]], {}, "Source" → periodicelem[[1]]}
 }];

SMTAddMesh["microA", nodes, solidelementsA];
SMTAddMesh["microB", nodes, solidelementsB];

(* multi-scale elements dependent part *)
Switch[localdata[[1, 5]]

 , "Deformation gradient (plane strain)",
 SMTAddEssentialBoundary[ConstrainedNodes, 1 → 0, 2 → 0];
 SMTSensitivityProblem[{
   {"F11", {"EBC", "∂u/∂F11", "D" → 1}},
   {"F12", {"EBC", "∂u/∂F12", "D" → 1}},
   {"F21", {"EBC", "∂v/∂F21", "D" → 2}},
   {"F22", {"EBC", "∂v/∂F22", "D" → 2}}
  }];

 , "Small strain tensor (plane strain)",
 SMTAddEssentialBoundary[ConstrainedNodes, 1 → 0, 2 → 0];
 SMTSensitivityProblem[{
   {"e11", {"EBC", "∂u/∂e11", "D" → 1}},
   {"e12", {"EBC", "∂u/∂e12", "D" → 1}, {"EBC", "∂v/∂e12", "D" → 2}},
   {"e22", {"EBC", "∂v/∂e22", "D" → 2}}
```

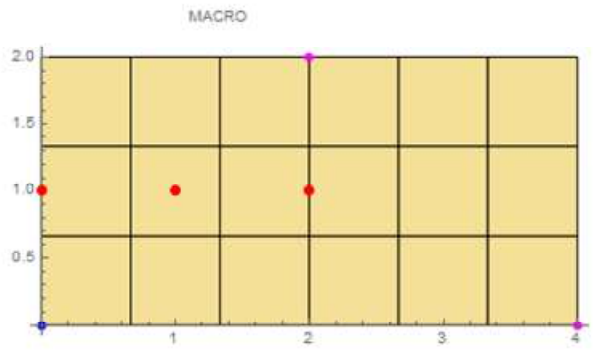
```
  }];

 , "Temperature gradient (2D)",
 SMTAddEssentialBoundary[ConstrainedNodes, 1 → 0];
 SMTSensitivityProblem[{
   {"Dθ1", {"EBC", "∂θ/∂Dθ1", "T" → 1}},
   {"Dθ2", {"EBC", "∂θ/∂Dθ2", "T" → 1}}
  }];

 , "Deformation gradient (plane strain)+temperature gradient",
 SMTAddEssentialBoundary[ConstrainedNodes, 1 → 0, 2 → 0, 3 → 0];
 SMTSensitivityProblem[{
   {"F11", {"EBC", "∂u/∂F11", "DT" → 1}},
   {"F12", {"EBC", "∂u/∂F12", "DT" → 1}},
   {"F21", {"EBC", "∂v/∂F21", "DT" → 2}},
   {"F22", {"EBC", "∂v/∂F22", "DT" → 2}},
   {"Dθ1", {"EBC", "∂θ/∂Dθ1", "DT" → 3}},
   {"Dθ2", {"EBC", "∂θ/∂Dθ2", "DT" → 3}}
  }];

];
(*constrains on line 1-4 2-3*)
SMTAddElement[MapThread[{"periodic", {ConstrainedNodes[[1]], #1, ConstrainedNodes[[2]], #2}} &,
  {nodes14 // Rest, nodes23 // Rest}]];
(*constrains on line 1-2 4-3 *)
SMTAddElement[MapThread[{ "periodic", {ConstrainedNodes[[3]], #1, ConstrainedNodes[[1]], #2}} &,
  {nodes43[[2 ;; -2]], nodes12[[2 ;; -2]]}]];

SMTAnalysis[];

SMTSetSolver[5, "IntegerParameters" → {{8, 0}}];

localdata[[{8, 9, 10}]] = {ConstrainedNodes, aRVE bRVE tRVE, sopt};
localdata
]
```

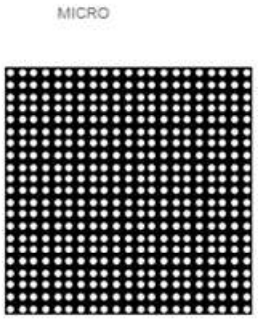
Examples of FE^2 multi-scale modeling

FE^2 modeling of 2D uniformly distributed micro-structure

Description

Perform simulation of the beam like structure made of uniformly perforated material with the given percentage of perforations. Calculate the effect of the percentage of perforations on the stiffness of the beam.





Macro problem

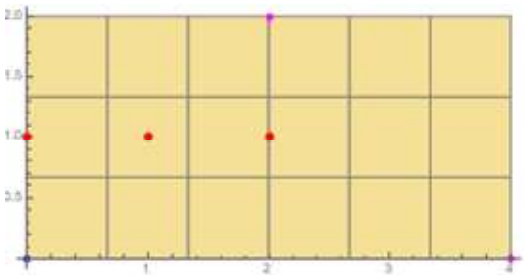
In[38]:=
```
If[SMTMultiScaleLoaded =!= True,
  MessageDialog["Load Multi-scale computational environment first!"];
  Quit[]];
```

In[39]:=
```
L = 4.; (*length cm/KN*)
H = 2.; (*height*)
Nx = 6; (*mesh density in x direction*)
Ny = 3; (*mesh density in y direction*)
vA = 1;(* prescribed displacement at L/2*)
SMTInputData["Threads" → 1];
SMTAddDomain[{"beam", "ExamplesQ1PFMacroSymm", {}}];
SMTAddEssentialBoundary[
  {Point[{0, 0}], 1 -> 0, 2 -> 0}
  , {Point[{L, 0}], 2 -> 0}
  , { Point[{L / 2, H}] , 2 → -vA}];
SMTAddMesh[Polygon[{{0, 0}, {L, 0}, {L, H}, {0, H}}], "beam", "Q1", {Nx, Ny}];
SMTAnalysis[];
FE2Elements = SMTFindElements["beam"];
Length[FE2Elements]
```

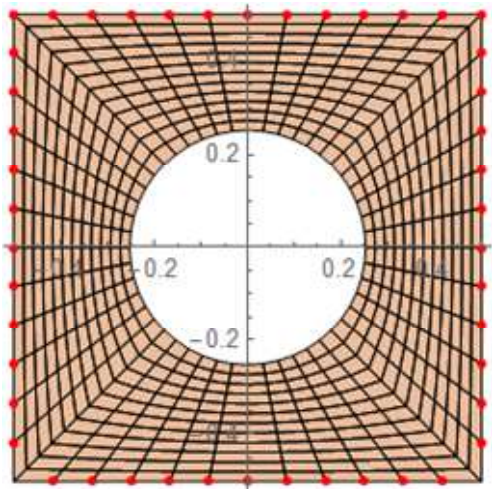
Out[39]=
```
18
```

- Macro mesh and selected points for post-processing

In[51]:=
```
monitorpoints = Table[{x, H / 2}, {x, {0., 0.25 L, 0.5 L}}];
SMTShowMesh["BoundaryConditions" → True,
 Epilog → {PointSize[Large], Red, Point[monitorpoints]}, Axes → True]
```



Micro problem



In[53]:=
```
aRVE = 1; (* RVE length *)
bRVE = 1; (*RVE height *)
tRVE = 1;(*RVE thickeness *)
pRVE = 0.3;(* percentage of perforation*)
rRVE = Sqrt[pRVE aRVE bRVE / π]; (* radius of perforation*)
NrRVE = 12; (* RVE mesh densitiy in radial direction*)
NabRVE = 12; (* RVE mesh densitiy per side*)
```

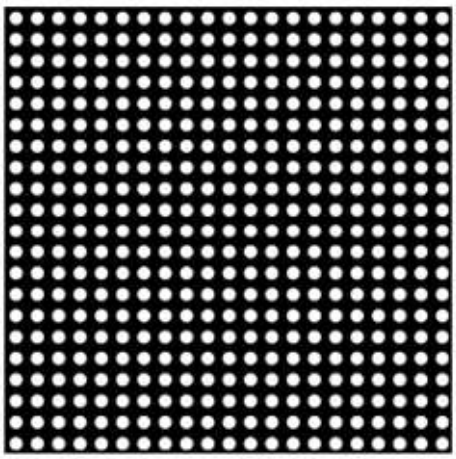
- The actual micro mesh is generated by the user defined FE22DMicroMeshVoids[localProblemData, specificMacroData] function where localProblemData is a standard local data structure (see documentation) and specificMacroData is an arbitrary user defined data needed by the FE22DMicroMeshVoids function in order to create micro problem mesh.
- specific_micro_data={
  - 1 aRVE
  - 2 bRVE
  - 3 cRVE
  - 4 rRVE - radius of perforations
  - 5 NrRVE - number of elements in radial direction
  - 6 NabRVE - number of elements per side of RVE
  - 7 micro_element - SMTMakeDll[micro_element_UEC] - element used to discretize the solid domain
  - 8 micro_material - material data related to micro element (see SMTAddDomain)
  - 9 topology - micro mesh type (see Element Topology)
  - 10 periodic_element - SMTMakeDll[periodic_element_UEC] - element used to impose periodic boundary conditions
  - for structured meshes periodic elements connect master node with slave node on opposite surface
  - for unstructured meshes the periodic element connects master node with slave triangle on opposite surface

- 11:
- "" - mesh is structured, thus each surface node has a matching node at the opposite surface
- SMTMakeDll[surface_element_UEC] - dummy element for surface triangulation used to impose periodic boundary
- 12 list of solution procedure options - given options are stored in *local_problem_data[[10]]* and interpreted later by the FE2SolveOne function. Current options are:
- "MinSubSteps"->*n* - minimum number of micro sub-steps per one macro step (default *n*=1)
- "MaxSubSteps"->*n* ⇒ defines a maximum number of micro sub-steps per macro step (default 100)
- "Method"->
  "Sensitivity" - full sensitivity analysis
  "Quasi-sensitivity" - sensitivity analysis at the and of macro step
  "Schur" -linearization based on Schur complement
- }

In[60]:=
```
Clear[specificMacroData]
specificMacroData = {aRVE , bRVE, tRVE, rRVE, NrRVE , NabRVE,
   SMTMakeDll["ExamplesSEPEQ1DFHYQ1DNeoHookeWA"], {"E *" → 21000, "ν *" → 0.3, "t *" → 2},
   "Q1", SMTMakeDll["ExamplesPeriodic2DStructured"], {"MinSubSteps" → 1, "MaxSubSteps" → 10}};
```

- Micro structure is infinitely periodic

In[62]:=
```
n = 20;
Graphics[{Rectangle[{0, 0}, {(n + 1) aRVE, (n + 1) bRVE}], White,
  Table[Disk[{x + aRVE / 2, y + bRVE / 2}, rRVE], {x, 0, n aRVE, aRVE}, {y, 0, n bRVE, bRVE}]}]
```



Set up multi-scale environment

- The "PartialRestart" option can be True only when the local problems have structurally the same mesh (only the actual coordinates of the nodes can be different).
- The "PartialDump" must be False only when during the solution of the local problem the mesh has been changed as well.

In[63]:=
```
SMTMultiScaleSet[
 "FE^2" → {{FE22DMicroMeshVoids, FE2SolveOne} -> FE2Elements},
 "SpecificMicroData" → {"FE22DMicroMeshVoids" -> specificMacroData},
"MaxKernels" → Automatic, "Threads" → 1,
 "Console" → False, "PartialDump" → True, "PartialRestart" → True]
```

Out[63]=
```
True
```

Visualize selected micro problem

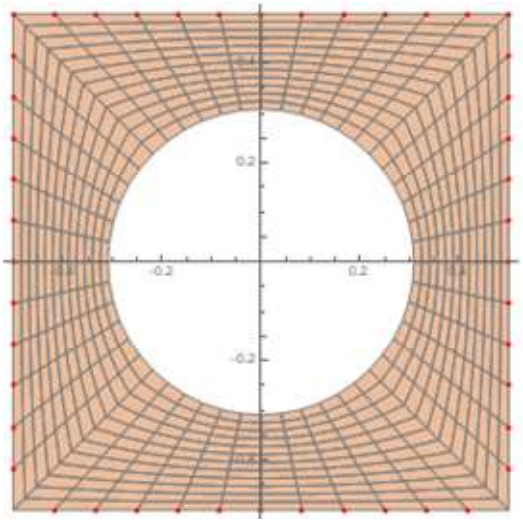
- SMTMicroRestart[{x, y}] ... restart micro problem that is nearest to the given spatial position (position is given in initial coordinates) and set the selected micro problem as the current micro problem

In[64]:=
```
SMTMicroRestart[{2.5, 1}]
```

```
{{FE^2, FE22DMicroMeshVoids, FE2SolveOne, 4, Deformation gradient (plane strain),
  4, Integrated stress and sensitivity (P, plane strain), 25},
 {2.52578, 0.80755}, {1, 1, 1, 0.309019, 12, 12,
  {C:\Users\jkorelc\AppData\Roaming\Mathematica\Applications\AceFEM\Elements\
     ExamplesSEPEQ1DFHYQ1DNeoHookeWA.W64.dll,
   ExamplesSEPEQ1DFHYQ1DNeoHookeWA, 0}, {E * → 21 000, ν * → 0.3, t * → 2}, Q1,
  {C:\Users\jkorelc\Desktop\SMS\Documentation\ExamplesPeriodic2DStructured.W64.dll,
   ExamplesPeriodic2DStructured, 0}, {MinSubSteps → 1, MaxSubSteps → 10}}, 42, 5, 11}
```

- SMTMicroEvaluate[exp] ... evaluates exp for the restarted micro problem

In[65]:=
```
SMTMicroEvaluate[SMTShowMesh[Axes → True]]
```



- local_problem_data structure of selected micro problem

In[66]:=
```
SMTCurrentLocalProblem
```

```
{{FE^2, FE22DMicroMeshVoids, FE2SolveOne, 4, Deformation gradient (plane strain),
  4, Integrated stress and sensitivity (P, plane strain), 25},
 {2.52578, 0.80755}, {1, 1, 1, 0.309019, 12, 12,
  {C:\Users\jkorelc\AppData\Roaming\Mathematica\Applications\AceFEM\Elements\
     ExamplesSEPEQ1DFHYQ1DNeoHookeWA.W64.dll,
   ExamplesSEPEQ1DFHYQ1DNeoHookeWA, 0}, {E * → 21 000, ν * → 0.3, t * → 2}, Q1,
  {C:\Users\jkorelc\Desktop\SMS\Documentation\ExamplesPeriodic2DStructured.W64.dll,
   ExamplesPeriodic2DStructured, 0}, {MinSubSteps → 1, MaxSubSteps → 10}}, 42, 5,
 11, {0, 0, 0, 0}, {235, 391, 79}, 1, {MinSubSteps → 1, MaxSubSteps → 10}, Null}
```

Perform and analyze single Newton-Raphson iteration

In[67]:=
```
NoStressComponents = 5; NoKinComponents = 4;
```

- SMTNextStep["λ"-> 1] - increment global load level

In[68]:=
```
SMTNextStep["λ" -> 1];
```

In[69]:=
```
testelem = 1;
```

- Task "FE^2" returns initialization data for chosen macro element
  - 1 - number of micro problems associated with the element
  - 2 - index of character switch of macro element that defines the name of the task that returns multi-scale related macro data

- 3 - length of multi-scale related macro data (≡NoKinComponents)
- 4 - index of character switch of macro element that defines the name of the micro problem task that returns multi-scale related micro data
- 5 - length of multi-scale related micro data (≡NoStressComponents+NoStressComponents*NoKinComponents)

In[70]:=
```
init = SMTTask["FE^2", "Elements" → testelem]
```

Out[70]=
```
{4, 3, 4, 4, 25}
```

- Check the position of micro problems within the chosen macro element

In[71]:=
```
Partition[SMTTask["Material points", "Elements" → testelem], SMTSpatialDimension]
```

Out[71]=
```
{{0.140883, 0.140883}, {0.525783, 0.140883}, {0.140883, 0.525783}, {0.525783, 0.525783}}
```

Multilevel-Newton iteration

- This is the data send by chosen macro element to micro problems associated with the chosen macro element
  - "Deformation gradient (plane strain)" task returns vec(F) = {F_{11}, F_{12}, F_{21}, F_{22}} in all integration points of chosen macro element

In[72]:=
```
Partition[SMTTask["Deformation gradient (plane strain)", "Elements" → testelem], init[[3]]] //
 MatrixForm
```

Out[72]=
$$\begin{pmatrix} 1. & 0. & 0. & 1. \\ 1. & 0. & 0. & 1. \\ 1. & 0. & 0. & 1. \\ 1. & 0. & 0. & 1. \end{pmatrix}$$

- SMTMicroSolve
  - transfers F from macro to micro problems
  - solves all micro problems and return average stress and constitutive tangent (inner Newton method)
  - transfers average stress and constitutive tangent from micro to macro elements
  - "Dump"→False states that the state of the micro problem stored on disk is not updated!
  - debugging can be turned on here with SMTMacroProblemStatus[[14]]=element_index command

In[73]:=
```
SMTMicroSolve["Dump" → False]
```

Out[73]=
```
True
```

- SMTNewtonIteration - standard Newton iteration at macro level

In[74]:=
```
SMTNewtonIteration[]
```

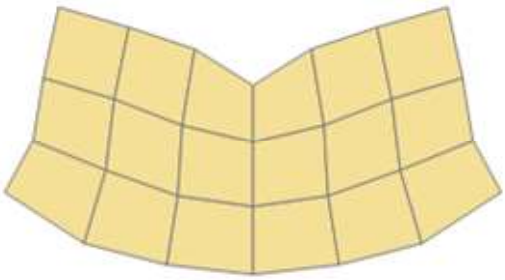
Out[74]=
```
0.462068
```

In[75]:=
```
SMTStatusReport[]
```

Out[75]=
```
Step/Iter=1/1 λ/Δλ=1./1. ||Δp||/||R||=0.462068/2423.14 Events=0 Status=0/{}
```

- Visualization of deformed macro problem after the first Newton iteration.

In[76]:=
```
SMTShowMesh["DeformedMesh" → True]
```



- Some useful tests of a single macro element
  - This is the data set by micro problems associated with the chosen macro element.
  - Flatten$\left[\left\{\left\{\text{vec (P)}, \frac{\partial \text{vec (P)}}{\partial \text{vec (F)}_1}, \frac{\partial \text{vec (P)}}{\partial \text{vec (F)}_2}, \ldots\right\} \ldots \text{for all intergation points}\right\}\right]$ where
  - vec (P) = {P_{11}, P_{12}, P_{21}, P_{22}, P_{33}} and vec (F) = {F_{11}, F_{12}, F_{21}, F_{22}}

In[77]:=
```
(allmicro = Partition[SMTElementData[testelem, "Data"], init[[5]]]) // Chop // MatrixForm
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 25\,000. & 0 & 0 & 7338.82 & 9701.65 & 0 & 5257.73 & 5257.73 & 0 & 0 & 0 & 5257.73 & 5257.73 & 0 \\ 0 & 0 & 0 & 0 & 0 & 25\,000. & 0 & 0 & 7338.82 & 9701.65 & 0 & 5257.73 & 5257.73 & 0 & 0 & 0 & 5257.73 & 5257.73 & 0 \\ 0 & 0 & 0 & 0 & 0 & 25\,000. & 0 & 0 & 7338.82 & 9701.65 & 0 & 5257.73 & 5257.73 & 0 & 0 & 0 & 5257.73 & 5257.73 & 0 \\ 0 & 0 & 0 & 0 & 0 & 25\,000. & 0 & 0 & 7338.82 & 9701.65 & 0 & 5257.73 & 5257.73 & 0 & 0 & 0 & 5257.73 & 5257.73 & 0 \end{pmatrix}$$

  - Those are stress tensor vec (P) and constitutive matrix D = $\frac{\partial \text{vec (P)}}{\partial \text{vec (F)}}$ in first integration point of the chosen element.

In[78]:=
```
allmicro[[1, 1 ;; NoStressComponents]]
```

```
{0., 0., 0., 0., 0.}
```

In[79]:=
```
(𝔻 = Partition[allmicro[[1, NoStressComponents + 1 ;;]], NoStressComponents] // Transpose) // Chop //
 MatrixForm
```

$$\begin{pmatrix} 25\,000. & 0 & 0 & 7338.82 \\ 0 & 5257.73 & 5257.73 & 0 \\ 0 & 5257.73 & 5257.73 & 0 \\ 7338.82 & 0 & 0 & 25\,000. \\ 9701.65 & 0 & 0 & 9701.65 \end{pmatrix}$$

  - This is the tangent matrix of the chosen macro element.

In[80]:=
```
SMTData[testelem, "SKR"][[1]] // MatrixForm
```

$$\begin{pmatrix} 10\,085.9 & 3149.14 & -7457.04 & 520.271 & -5042.96 & -3149.14 & 2414.09 & -520.271 \\ 3149.14 & 10\,085.9 & -520.271 & 2414.09 & -3149.14 & -5042.96 & 520.271 & -7457.04 \\ -7457.04 & -520.271 & 10\,085.9 & -3149.14 & 2414.09 & 520.271 & -5042.96 & 3149.14 \\ 520.271 & 2414.09 & -3149.14 & 10\,085.9 & -520.271 & -7457.04 & 3149.14 & -5042.96 \\ -5042.96 & -3149.14 & 2414.09 & -520.271 & 10\,085.9 & 3149.14 & -7457.04 & 520.271 \\ -3149.14 & -5042.96 & 520.271 & -7457.04 & 3149.14 & 10\,085.9 & -520.271 & 2414.09 \\ 2414.09 & 520.271 & -5042.96 & 3149.14 & -7457.04 & -520.271 & 10\,085.9 & -3149.14 \\ -520.271 & -7457.04 & 3149.14 & -5042.96 & 520.271 & 2414.09 & -3149.14 & 10\,085.9 \end{pmatrix}$$

  - Those are eigenvalues of the element tangent matrix of chosen macro element. In the fist iteration the number of zero eigenvalues should be equal to the number of rigid body motions.

In[81]:=
```
SMTData[testelem, "SKR"][[1]] // Eigenvalues // Chop
```

```
{32 338.8, 17 661.2, 10 515.5, 10 085.9, 10 085.9, 0, 0, 0}
```

  - This is residual vector of the chosen macro element. It should be zero in the first iteration because the macro load has no effect on micro problems in the first Newton iteration.

In[82]:=
```
SMTData[testelem, "SKR"][[2]]
```

```
{0., 0., 0., 0., 0., 0., 0., 0.}
```

- This is test of the positive definiteness of the global tangent matrix.

In[83]:=
```
PositiveDefiniteMatrixQ[SMTData["TangentMatrix"]]
```

```
True
```

Return the state of the macro problem back to initial state

In[84]:=
```
SMTStepBack[];
```

Evaluate complete response using an adaptive path following procedure

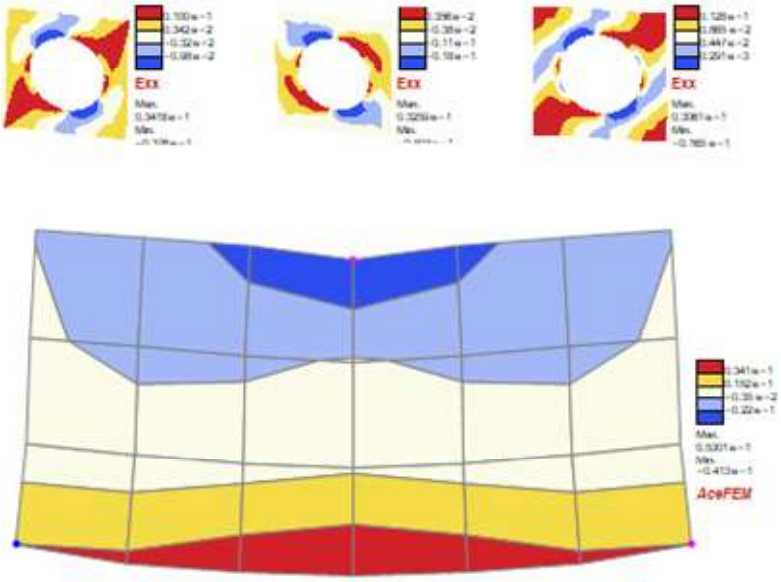
In[85]:=
```
response = {{0, 0}};
λMax = 1; λ0 = λMax / 4; ΔλMin = λMax / 1000; ΔλMax = λMax / 4;
tolNR = 10^-8; maxNR = 15; targetNR = 8;
SMTNextStep["λ" -> λ0];
While[

  While[
   step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λMax}]

   , SMTMicroSolve["Dump" → False];

   (* react on events at RVE *)
   Switch[SMTSharedStatus[[1]]
    , 0 | 1,
    SMTNewtonIteration[];
    SMTStatusReport[SMTSharedStatus[[1]]];
    , 2,
    (* force step back due to the failed micro simulation *)
    SMTIData[{"Iteration", "TotalIteration"}, SMTIData[{"Iteration", "TotalIteration"}] + 1];
    SMTIData["ErrorStatus", 2];
    SMTStatusReport[SMTSharedStatus[[2]]];
    , 3,
    (*abort due to the fatal error *)
    SMTStatusReport[SMTSharedStatus[[2]]]; Abort[];
   ]
  ]; (* end of NR loop *)

  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; Abort[];];

  (* successful step *)
  If[Not[step[[1]]],
   (*dump the state of micro problems to disc before the next step*)
   SMTMicroSolve["Dump" → True];
   (* vizualization and post-processing*)
   macromesh = SMTShowMesh["BoundaryConditions" → True, "DeformedMesh" → True,
     "Field" → SMTMultiScalePostData["Exx"], "Legend" → True, "Contour" → 4];
   micromesh = Table[
     SMTMicroRestart[x];
     SMTMicroEvaluate[SMTShowMesh["Mesh" → False,
       "DeformedMesh" → True, "Field" → "Exx", "Legend" → True, "Contour" → 4]]
     , {x, monitorpoints}];
   Print[Column[{GraphicsRow[micromesh, ImageSize → 600], Show[macromesh, ImageSize → 600]}]];
   AppendTo[response, {SMTRData["Multiplier"] vA, -SMTResidual[Point[{L / 2, H}]][[1, 2]]}];
  ];

  step[[3]]

  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" -> step[[2]]]
];
```

Step/Iter=1/1 λ/Δλ=0.25/0.25 ‖Δp‖/‖R‖=0.115517/605.785 Events=0 Status=0/{} Tag=0

Step/Iter=1/2 λ/Δλ=0.25/0.25 ‖Δp‖/‖R‖=0.00498499/40.8534 Events=0 Status=0/{} Tag=0

Step/Iter=1/3 λ/Δλ=0.25/0.25 ‖Δp‖/‖R‖=0.0000338016/0.169297 Events=0 Status=0/{} Tag=0

Step/Iter=1/4 λ/Δλ=0.25/0.25 ‖Δp‖/‖R‖=
$5.08573\times10^{-10}/4.51852\times10^{-6}$ Events=0 Status=0/{} Tag=0

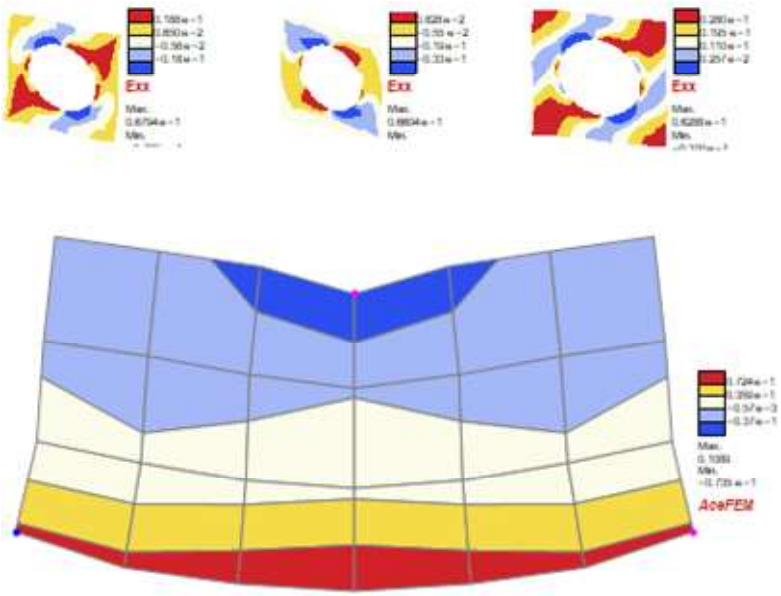


Step/Iter=2/1 λ/Δλ=0.5/0.25 ‖Δp‖/‖R‖=0.110658/623.443 Events=0 Status=0/{} Tag=0

Step/Iter=2/2 λ/Δλ=0.5/0.25 ‖Δp‖/‖R‖=0.00554694/41.6278 Events=0 Status=0/{} Tag=0

Step/Iter=2/3 λ/Δλ=0.5/0.25 ‖Δp‖/‖R‖=0.0000282586/0.183549 Events=0 Status=0/{} Tag=0

Step/Iter=2/4 λ/Δλ=0.5/0.25 ‖Δp‖/‖R‖=
$1.98442\times10^{-9}/8.40637\times10^{-6}$ Events=0 Status=0/{} Tag=0



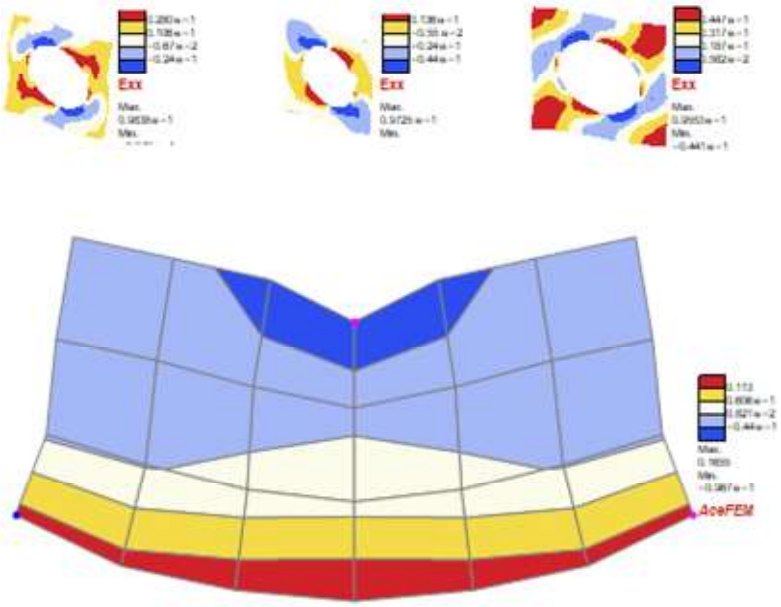
Step/Iter=3/1 λ/Δλ=0.75/0.25 ‖Δp‖/‖R‖=0.104509/622.271 Events=0 Status=0/{} Tag=0

Step/Iter=3/2 λ/Δλ=0.75/0.25 ‖Δp‖/‖R‖=0.00678936/43.0537 Events=0 Status=0/{} Tag=0

Step/Iter=3/3 λ/Δλ=0.75/0.25 ‖Δp‖/‖R‖=0.000103171/0.25543 Events=0 Status=0/{} Tag=0

Step/Iter=3/4 λ/Δλ=0.75/0.25 ‖Δp‖/‖R‖=
1.07983×10^{-8}/0.000127864 Events=0 Status=0/{} Tag=0

Step/Iter=3/5 λ/Δλ=0.75/0.25 ‖Δp‖/‖R‖=
6.37554×10^{-16}/2.94187×10^{-12} Events=0 Status=0/{} Tag=0



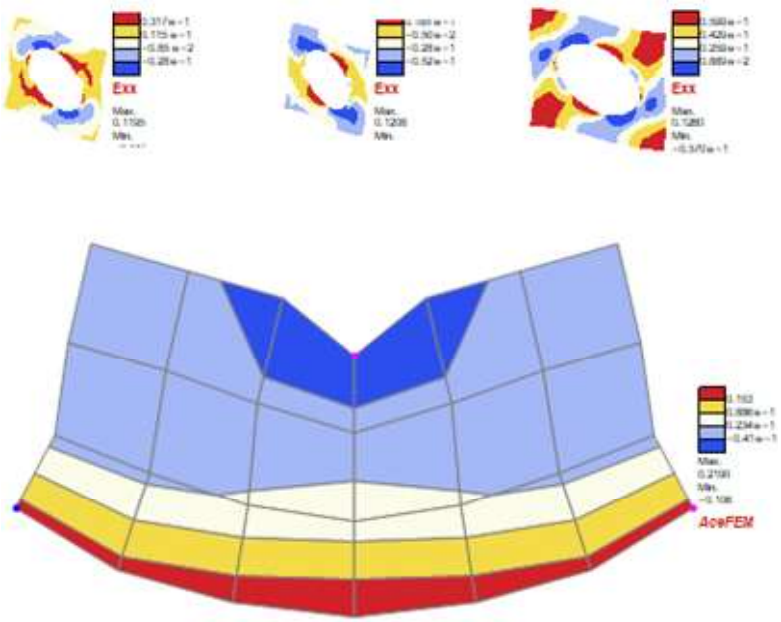
Step/Iter=4/1 λ/Δλ=1./0.25 ‖Δp‖/‖R‖=0.0967447/587.861 Events=0 Status=0/{} Tag=0

Step/Iter=4/2 λ/Δλ=1./0.25 ‖Δp‖/‖R‖=0.0107445/51.4503 Events=0 Status=0/{} Tag=0

Step/Iter=4/3 λ/Δλ=1./0.25 ‖Δp‖/‖R‖=0.000673112/1.28894 Events=0 Status=0/{} Tag=0

Step/Iter=4/4 λ/Δλ=1./0.25 ‖Δp‖/‖R‖=7.25209×10^{-7}/0.0051803 Events=0 Status=0/{} Tag=0

Step/Iter=4/5 λ/Δλ=1./0.25 ‖Δp‖/‖R‖=
4.74444×10^{-12}/7.51121×10^{-9} Events=0 Status=0/{} Tag=0



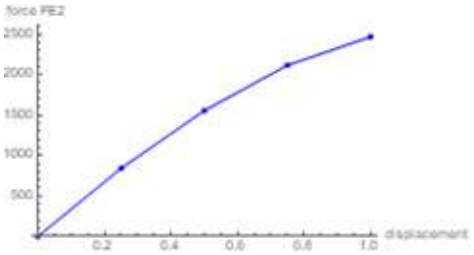
In[90]:= **SMTMultiScaleSimulationReport[];**

MACRO	
Number of macro elements	18
Number of macro equations	52
No. of steps	4
No. of steps back	1
Total no. of iterations	19
Total driver time (s)	30.92
Total linear solver time (s)	0.10
Total K&R time (s)	0.00
MICRO	
Number of FE^2 elements	18
Number of MIEL elements	0
Number of single-scale elements	0
Parallelization kernels/threads	16/1
Total number of micro problems	72
Total number of micro elements	43 128
Total number of micro nodes	46 584
Total number of micro equations	92 736
Average number of micro equations	1288.
Real micro mesh generation time	2.59
Number of MicroSolve calls	23
Total MicroSolve time	7.63
Total number of MP equations solved	2 132 928
Total number of steps	1946
Total number of back steps	2
Average number of micro problems/kernel	4.5
Total number of MP solved	1656
Average MP time	0.04
Average MP K&R time	0.01
Average MP linear solver time	0.02
Average MP sens.p.load time	0.00
Average MP sens.l.solver time	0.00
Average MP Schur complement time	0
Average MP task time	0.00
Average MP Driver time	0.15
Average real Driver time	15.45
Total parallel MP time	1:0.30
Number of restarts	1656
Total parallel restart time	6.55
Average restart time	0.00
Number of dumps	360
Total parallel dump time	19.93
Average dump time	0.06
Total dump files size (byte)	12 244 119
zlib compression level	2
Macro + micro analysis time	10.33
Aministrative time	8.02
Lost time:	3.86
Total time	18.35

Analysis of the results

Response curve

In[91]:=
```
ListLinePlot[response, AxesLabel → {"displacement", "force FE2"},
 PlotMarkers → Automatic, PlotStyle → Blue]
```



Calculate stiffness (KN/cm)

In[92]:=
```
response[[2, 2]] / response[[2, 1]]
```

```
3364.95
```

Visualization of the chosen micro problem

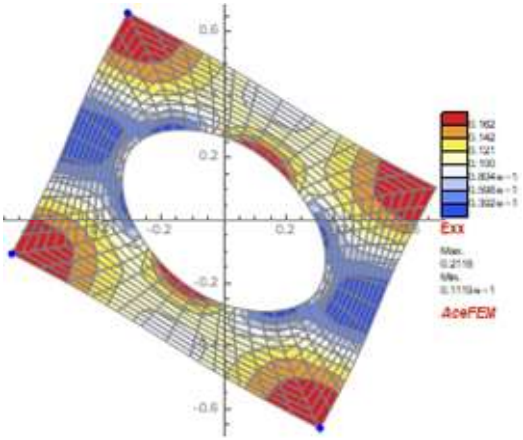
- SMTMicroRestart[{x, y}] ... restart micro problem that is nearest to the given spatial position (position is given in initial coordinates) and set the selected micro problem as the current micro problem

In[93]:=
```
SMTMicroRestart[{0, 0}]
```

```
{{FE^2, FE22DMicroMeshVoids, FE2SolveOne, 4, Deformation gradient (plane strain),
  4, Integrated stress and sensitivity (P, plane strain), 25},
 {0.140883, 0.140883}, {1, 1, 1, 0.309019, 12, 12,
  {C:\Users\jkorelc\AppData\Roaming\Mathematica\Applications\AceFEM\Elements\
     ExamplesSEPEQ1DFHYQ1DNeoHookeWA.W64.dll,
   ExamplesSEPEQ1DFHYQ1DNeoHookeWA, 0}, {E * → 21000, ν * → 0.3, t * → 2}, Q1,
  {C:\Users\jkorelc\Desktop\SMS\Documentation\ExamplesPeriodic2DStructured.W64.dll,
   ExamplesPeriodic2DStructured, 0}, {MinSubSteps → 1, MaxSubSteps → 10}}, 1, 5, 1}
```

- SMTMicroEvaluate[exp] ... evaluate exp for the current micro problem

In[94]:=
```
SMTMicroEvaluate[SMTShowMesh[Axes → True, "DeformedMesh" → True,
  "BoundaryConditions" → True, "Field" -> "Exx", "Mesh" → True, "Contour" -> True]]
```



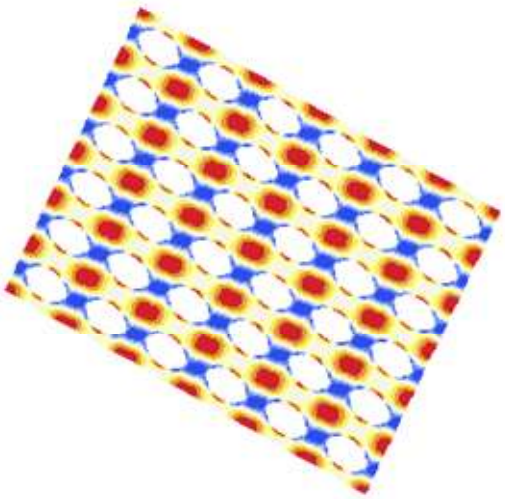
- Representation of periodic micro-structure

In[95]:=
```
SMTMicroEvaluate[SMTCurrentLocalProblem]
```

```
{{FE^2, FE22DMicroMeshVoids, FE2SolveOne, 4, Deformation gradient (plane strain),
  4, Integrated stress and sensitivity (P, plane strain), 25},
 {0.140883, 0.140883}, {1, 1, 1, 0.309019, 12, 12,
  {C:\Users\jkorelc\AppData\Roaming\Mathematica\Applications\AceFEM\Elements\
     ExamplesSEPEQ1DFHYQ1DNeoHookeWA.W64.dll,
   ExamplesSEPEQ1DFHYQ1DNeoHookeWA, 0}, {E * → 21000, ν * → 0.3, t * → 2}, Q1,
  {C:\Users\jkorelc\Desktop\SMS\Documentation\ExamplesPeriodic2DStructured.W64.dll,
   ExamplesPeriodic2DStructured, 0}, {MinSubSteps → 1, MaxSubSteps → 10}},
 1, 5, 1, {0.982788, 0.369815, -0.555631, 0.769222}, {235, 391, 79},
 1, {MinSubSteps → 1, MaxSubSteps → 10}, Null}
```

In[96]:=
```
infper = SMTMicroEvaluate[
  {aRVE, bRVE} = SMTCurrentLocalProblem[[3, {1, 2}]];
  corner =
   Map[SMTNodeData[Point[#], "at"][[1]] &,
    {{-aRVE / 2, -bRVE / 2}, {aRVE / 2, -bRVE / 2}, {aRVE / 2, bRVE / 2}, {-aRVE / 2, bRVE / 2}}];
  Table[SMTShowMesh["DeformedMesh" → {i, j} + i (corner[[3]] - corner[[4]]) +
      j (corner[[4]] - corner[[1]]) + {SMTPostData["u"], SMTPostData["v"]},
    "Mesh" → False, "Field" → "Exx", "Legend" → False, "Contour" -> True]
   , {i, 0, 5}, {j, 0, 5}]
 ];
```

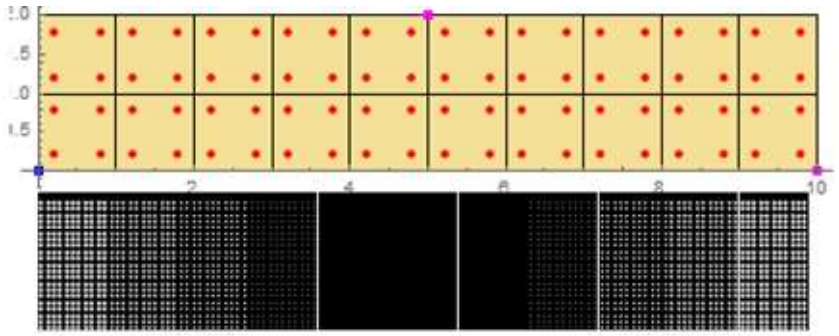
In[97]:=
```
Show[infper]
```



FE^2 modeling of 2D functionally graded material

Description

Perform simulation of the beam like structure made of perforated material. The percentage of perforations follows the distribution of the bending moments. Consequently the percentage is changing quadratically with the length of the beam.



Macro problem

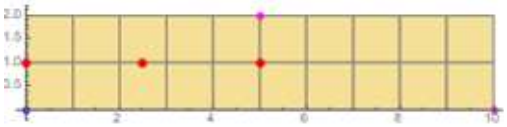
In[98]:=
```
If[SMTMultiScaleLoaded =!= True,
  MessageDialog["Load Multi-scale computational environment first!"];
  Quit[]];
```

In[99]:=
```
L = 10.; (*length cm/KN*)
H = 2.; (*height*)
Nx = 10; (*mesh density in x direction*)
Ny = 2; (*mesh density in y direction*)
vA = 3;(* prescribed displacement at L/2*)
SMTInputData["Threads" → 1];
SMTAddDomain[{"beam", "ExamplesQ1PFMacroSymm", {}}];
SMTAddEssentialBoundary[
  {Point[{0, 0}], 1 -> 0, 2 -> 0}
  , {Point[{L, 0}], 2 -> 0}
  , { Point[{L / 2, H}] , 2 → -vA}];
SMTAddMesh[Polygon[{{0, 0}, {L, 0}, {L, H}, {0, H}}], "beam", "Q1", {Nx, Ny}];
SMTAnalysis[];
FE2Elements = SMTFindElements["beam"];
Length[FE2Elements]
```

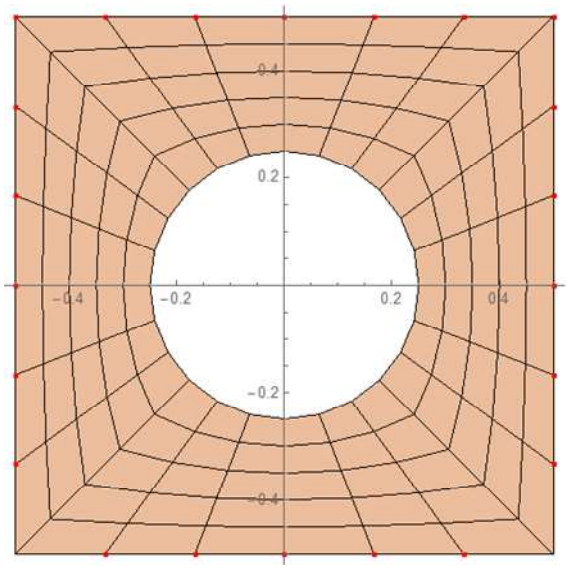
```
20
```

- Macro mesh and selected points for later post-processing

In[111]:=
```
monitorpoints = Table[{x, H / 2}, {x, {0., 0.25 L, 0.5 L}}];
SMTShowMesh["BoundaryConditions" → True,
 Epilog → {PointSize[Large], Red, Point[monitorpoints]}, Axes → True]
```



Micro problem

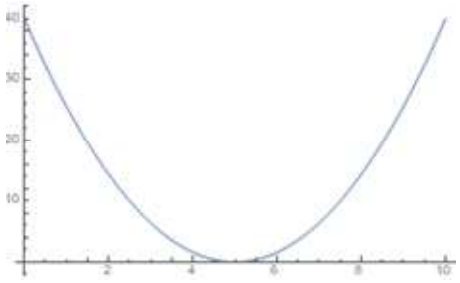


- The actual micro mesh is generated by the user defined FE22DMicroMeshVoids[localProblemData, specificMacroData] function where localProblemData is a standard local data structure (see documentation) and specificMacroData is an arbitrary user defined data neded by the FE22DMicroMeshVoids function in order to create micro problem mesh.
- micro mesh is a function of spatial coordinates

In[113]:=
```
specificMacroDataFunction[{X_, Y_}] := (
    aRVE = 1; (* RVE length *)
    bRVE = 1; (*RVE height *)
    tRVE = 1; (*RVE thickeness *)
    (* percentage of perforation is quadratic function with the maximum 40% *)
    pRVE = Interpolation[{{0, 0.4}, {L / 2, 0}, {L, 0.4}}, InterpolationOrder → 2];
    NrRVE = 5; (* RVE mesh densitiy in radial direction*)
    NabRVE = 6; (* RVE mesh densitiy per side*)
    {aRVE , bRVE, tRVE, Sqrt[pRVE[X] aRVE bRVE / π], NrRVE , NabRVE,
     SMTMakeDll["ExamplesSEPEQ1DFHYQ1DNeoHookeWA"], {"E *" → 21000, "ν *" → 0.3}, "Q1",
     SMTMakeDll["ExamplesPeriodic2DStructured"], {"MinSubSteps" → 1, "MaxSubSteps" → 10}}
   );
```

- Percentage of perforations

In[114]:=
```
pRVE = Interpolation[{{0, 0.4}, {L / 2, 0}, {L, 0.4}}, InterpolationOrder → 2];
Plot[100 pRVE[x], {x, 0, L}]
```



- Micro structure is infinitely periodic

In[116]:=
```
n = 10;
GraphicsRow[Table[
  specificMacroDataFunction[{xL, 0}];
  Graphics[{Rectangle[{0, 0}, {(n + 1) aRVE, 2 (n + 1) bRVE}], White, Table[Disk[{x + aRVE / 2,
       y + bRVE / 2}, Sqrt[pRVE[xL] aRVE bRVE / π]], {x, 0, n aRVE, aRVE}, {y, 0, 2 n bRVE, bRVE}]}]
  ,
  {xL,
   0,
   L,
   L /
    10}], -20]
```

Set up multi-scale environment

- The "PartialRestart" option can be True only when the local problems have structurally the same mesh (only the actual coordinates of the nodes can be different).
- The "PartialDump" must be False only when during the solution of the local problem the mesh has been changed as well.

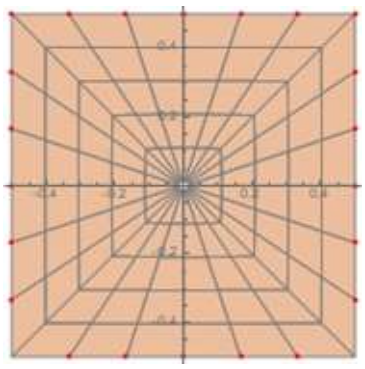
In[118]:=
```
SMTMultiScaleSet[
 "FE^2" → {{FE22DMicroMeshVoids, FE2SolveOne} -> FE2Elements},
 "SpecificMicroData" → {"FE22DMicroMeshVoids" -> specificMacroDataFunction},
"MaxKernels" → Automatic, "Threads" → 1,
 "Console" → False, "PartialDump" → True, "PartialRestart" → False]
```

Out[118]=
```
True
```

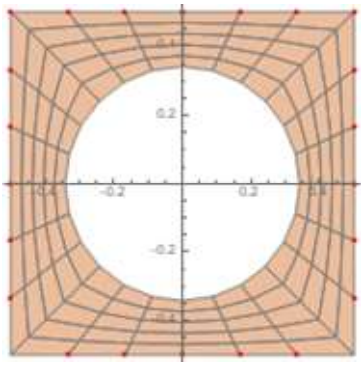
Visualize selected micro problem

- MicroRestart[{x, y}] ... restart micro problem that is nearest to the given spatial position (position is given in initial coordinates) and set the selected micro problem as the current micro problem
- MicroEvaluate[exp] ... evaluate exp for the restarted micro problem

In[119]:=
```
SMTMicroRestart[{L / 2, 0}];
SMTMicroEvaluate[SMTShowMesh[Axes → True]]
```



In[121]:=
```
SMTMicroRestart[{0, 0}];
SMTMicroEvaluate[SMTShowMesh[Axes → True]]
```



Evaluate response

In[123]:=
```
response = {{0, 0}};
λMax = 1; λ0 = λMax / 4; ΔλMin = λMax / 1000; ΔλMax = λMax / 4;
tolNR = 10^-8; maxNR = 15; targetNR = 8;
SMTNextStep["λ" -> λ0];
While[

  While[
   step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λMax}]

   , SMTMicroSolve["Dump" → False];
   (* react on events at RVE *)
   Switch[SMTSharedStatus[[1]]
    , 0 | 1,
    SMTNewtonIteration[];
    SMTStatusReport[];
    , 2,
    (* force step back due to the failed micro simulation *)
    SMTIData[{"Iteration", "TotalIteration"}, SMTIData[{"Iteration", "TotalIteration"}] + 1];
    SMTIData["ErrorStatus", 2];
    SMTStatusReport[SMTSharedStatus[[2]]];
    , 3,
    (*abort due to the fatal error *)
    SMTStatusReport[SMTSharedStatus[[2]]]; Abort[];
   ]
  ]; (* end of NR loop *)

  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; Abort[];];

  (* successful step *)
  If[Not[step[[1]]],
   (*dump the state of micro problems to disc before the next step*)
   SMTMicroSolve["Dump" → True];
   (* vizualization and post-processing*)
   macromesh = SMTShowMesh["BoundaryConditions" → True, "DeformedMesh" → True,
     "Field" → SMTMultiScalePostData["Exx"], "Legend" → True, "Contour" → 4];
   micromesh = Table[
     SMTMicroRestart[x];
     SMTMicroEvaluate[SMTShowMesh["Mesh" → False,
       "DeformedMesh" → True, "Field" → "Exx", "Legend" → True, "Contour" → 4]]
     , {x, monitorpoints}];
   Print[Column[{GraphicsRow[micromesh, ImageSize → 600], Show[macromesh, ImageSize → 600]}]];
   AppendTo[response, {SMTRData["Multiplier"] vA, -SMTResidual[Point[{L / 2, H}]][[1, 2]]}];
  ];

  step[[3]]

  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" -> step[[2]]]
];
```

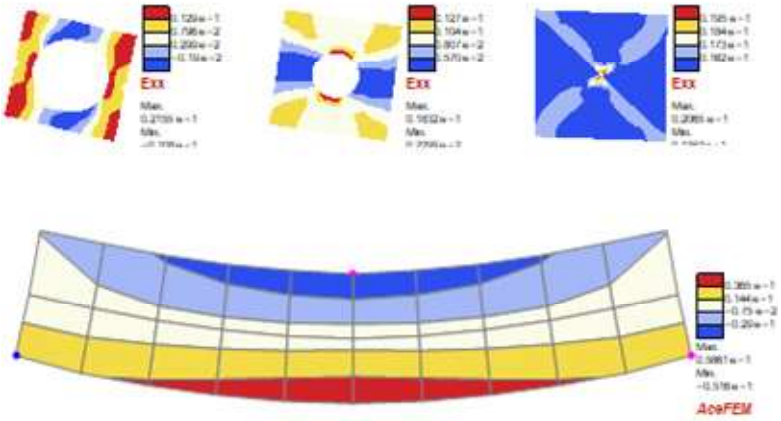
Step/Iter=1/1 λ/Δλ=0.25/0.25 ‖Δp‖/‖R‖=0.395349/1883.78 Events=0 Status=0/{}

Step/Iter=1/2 λ/Δλ=0.25/0.25 ‖Δp‖/‖R‖=0.0448592/155.725 Events=0 Status=0/{}

Step/Iter=1/3 λ/Δλ=0.25/0.25 ‖Δp‖/‖R‖=0.00071819/2.31169 Events=0 Status=0/{}

Step/Iter=1/4 λ/Δλ=0.25/0.25 ‖Δp‖/‖R‖=2.13775×10^{-7}/0.000886281 Events=0 Status=0/{}

Step/Iter=1/5 λ/Δλ=0.25/0.25 ‖Δp‖/‖R‖=1.09084×10^{-13}/2.51883×10^{-10} Events=0 Status=0/{}



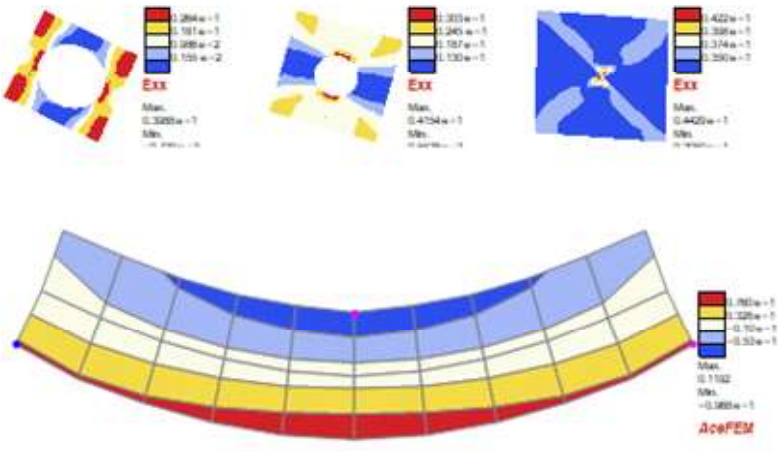
Step/Iter=2/1 λ/Δλ=0.5/0.25 ‖Δp‖/‖R‖=0.379167/1849.14 Events=0 Status=0/{}

Step/Iter=2/2 λ/Δλ=0.5/0.25 ‖Δp‖/‖R‖=0.0419593/147.062 Events=0 Status=0/{}

Step/Iter=2/3 λ/Δλ=0.5/0.25 ‖Δp‖/‖R‖=0.000618394/2.11076 Events=0 Status=0/{}

Step/Iter=2/4 λ/Δλ=0.5/0.25 ‖Δp‖/‖R‖=1.90083×10^{-7}/0.00068374 Events=0 Status=0/{}

Step/Iter=2/5 λ/Δλ=0.5/0.25 ‖Δp‖/‖R‖=9.32244×10^{-14}/2.2049×10^{-10} Events=0 Status=0/{}



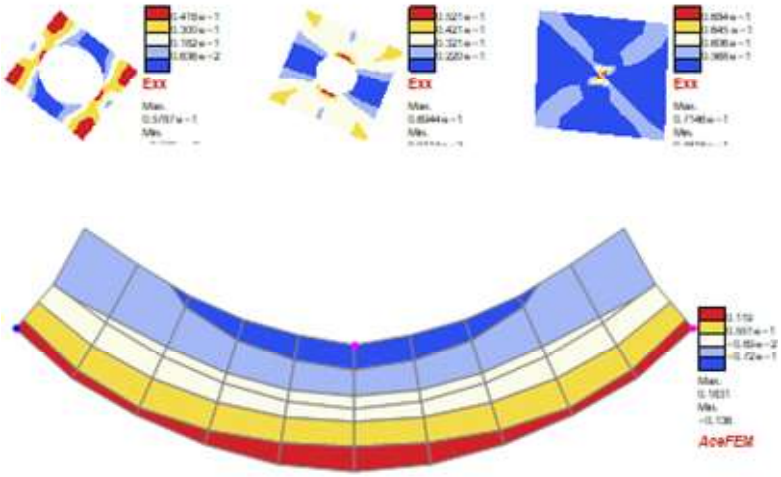
Step/Iter=3/1 λ/Δλ=0.75/0.25 ‖Δp‖/‖R‖=0.378675/1821.18 Events=0 Status=0/{}

Step/Iter=3/2 λ/Δλ=0.75/0.25 ‖Δp‖/‖R‖=0.0403591/141.758 Events=0 Status=0/{}

Step/Iter=3/3 λ/Δλ=0.75/0.25 ‖Δp‖/‖R‖=0.000546069/1.95341 Events=0 Status=0/{}

Step/Iter=3/4 λ/Δλ=0.75/0.25 ‖Δp‖/‖R‖=2.22737×10^{-7}/0.000538025 Events=0 Status=0/{}

Step/Iter=3/5 λ/Δλ=0.75/0.25 ‖Δp‖/‖R‖=8.62102×10^{-14}/2.33455×10^{-10} Events=0 Status=0/{}



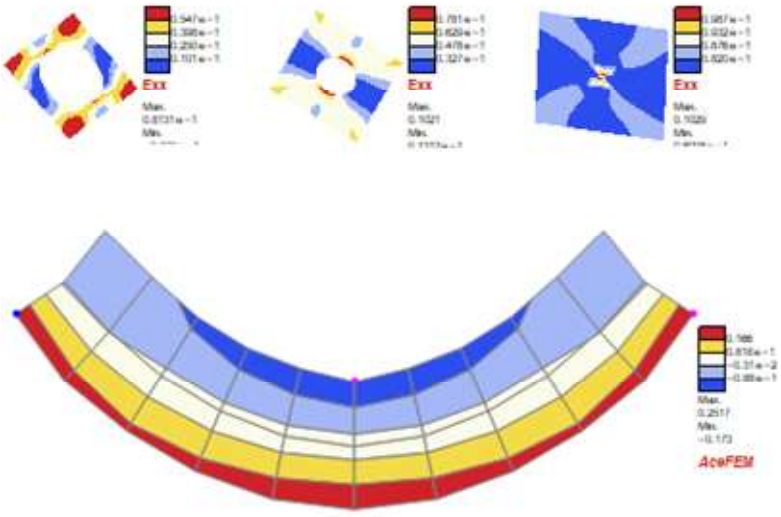
Step/Iter=4/1 λ/Δλ=1./0.25 ‖Δp‖/‖R‖=0.392019/1800.31 Events=0 Status=0/{}

Step/Iter=4/2 λ/Δλ=1./0.25 ‖Δp‖/‖R‖=0.0402269/138.897 Events=0 Status=0/{}

Step/Iter=4/3 λ/Δλ=1./0.25 ‖Δp‖/‖R‖=0.000506927/1.8372 Events=0 Status=0/{}

Step/Iter=4/4 λ/Δλ=1./0.25 ‖Δp‖/‖R‖=2.52495×10^{-7}/0.000456598 Events=0 Status=0/{}

Step/Iter=4/5 λ/Δλ=1./0.25 ‖Δp‖/‖R‖=8.56746×10^{-14}/2.52677×10^{-10} Events=0 Status=0/{}



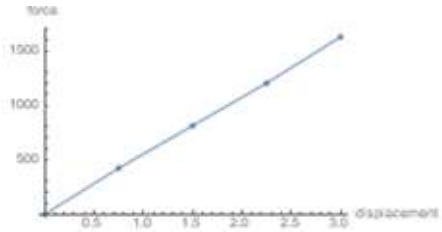
In[128]:=
```
SMTMultiScaleSimulationReport[];
```

MACRO	
Number of macro elements	20
Number of macro equations	62
No. of steps	4
No. of steps back	0
Total no. of iterations	20
Total driver time (s)	47.78
Total linear solver time (s)	0.00
Total K&R time (s)	0.00
MICRO	
Number of FE^2 elements	20
Number of MIEL elements	0
Number of single-scale elements	0
Parallelization kernels/threads	16/1
Total number of micro problems	80
Total number of micro elements	10 480
Total number of micro nodes	12 400
Total number of micro equations	24 320
Average number of micro equations	304.
Real micro mesh generation time	2.34
Number of MicroSolve calls	24
Total MicroSolve time	24.68
Total number of MP equations solved	583 680
Total number of steps	2240
Total number of back steps	0
Average number of micro problems/kernel	5.
Total number of MP solved	1920
Average MP time	0.01
Average MP K&R time	0.00
Average MP linear solver time	0.00
Average MP sens.p.load time	0.00
Average MP sens.l.solver time	0.00
Average MP Schur complement time	0
Average MP task time	0.00
Average MP Driver time	0.25
Average real Driver time	30.26
Total parallel MP time	25.73
Number of restarts	1920
Total parallel restart time	3:40.25
Average restart time	0.11
Number of dumps	400
Total parallel dump time	18.28
Average dump time	0.05
Total dump files size (byte)	6 473 011
zlib compression level	2
Macro + micro analysis time	27.03
Aministrative time	6.30
Lost time:	23.07
Total time	33.32

Analysis of the results

Response curve

In[129]:=
```
ListLinePlot[response, AxesLabel → {"displacement", "force"}, PlotMarkers → Automatic]
```



Calculate stiffness (KN/cm)

In[130]:=
```
response[[2, 2]] / response[[2, 1]]
```

```
563.81
```

Visualization of the chosen micro problem

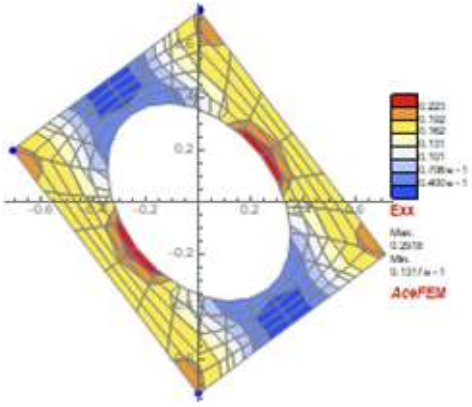
- SMTMicroRestart[{x, y}] ... restart micro problem that is nearest to the given spatial position (position is given in initial coordinates) and set the selected micro problem as the current micro problem

In[131]:=
```
SMTMicroRestart[{0, 0}]
```

```
{{FE^2, FE22DMicroMeshVoids, FE2SolveOne, 4, Deformation gradient (plane strain),
  4, Integrated stress and sensitivity (P, plane strain), 25},
 {0.211325, 0.211325}, {1, 1, 1, 0.341744, 5, 6,
  {C:\Users\jkorelc\AppData\Roaming\Mathematica\Applications\AceFEM\Elements\
     ExamplesSEPEQ1DFHYQ1DNeoHookeWA.W64.dll,
   ExamplesSEPEQ1DFHYQ1DNeoHookeWA, 0}, {E * → 21000, ν * → 0.3}, Q1,
  {C:\Users\jkorelc\Desktop\SMS\Documentation\ExamplesPeriodic2DStructured.W64.dll,
   ExamplesPeriodic2DStructured, 0}, {MinSubSteps → 1, MaxSubSteps → 10}}, 1, 5, 1}
```

- SMTMicroEvaluate[exp] ... evaluate exp for the current micro problem

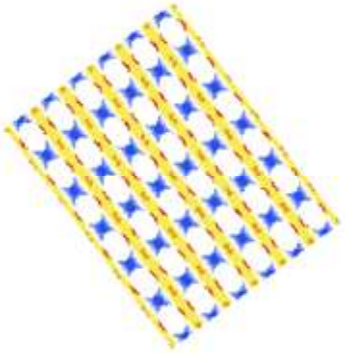
In[132]:=
```
SMTMicroEvaluate[SMTShowMesh[Axes → True, "DeformedMesh" → True,
  "BoundaryConditions" → True, "Field" -> "Exx", "Mesh" → True, "Contour" -> True]]
```



- Representation of periodic micro-structure

In[133]:=
```
infper = SMTMicroEvaluate[
   {aRVE, bRVE} = SMTCurrentLocalProblem[[3, {1, 2}]];
   corner =
    Map[SMTNodeData[Point[#], "at"][[1]] &,
     {{-aRVE / 2, -bRVE / 2}, {aRVE / 2, -bRVE / 2}, {aRVE / 2, bRVE / 2}, {-aRVE / 2, bRVE / 2}}];
   Table[SMTShowMesh["DeformedMesh" →  {i, j} + i (corner[[3]] - corner[[4]]) +
       j (corner[[4]] - corner[[1]]) + {SMTPostData["u"], SMTPostData["v"]},
     "Mesh" → False, "Field" → "Exx", "Legend" → False, "Contour" -> True]
    , {i, 0, 5}, {j, 0, 5}]
  ];
```

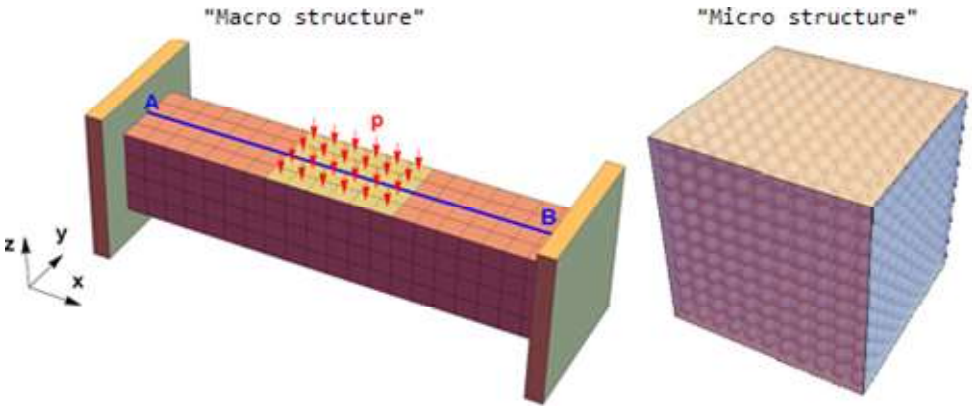
In[134]:=
```
Show[infper]
```



FE^2 modeling of 3D uniformly distributed micro-structure

Description

Perform simulation of the beam like structure made of perforated material.



Macro problem

In[135]:=
```
If[SMTMultiScaleLoaded =!= True,
  MessageDialog["Load Multi-scale computational environment first!"];
  Quit[]];
```

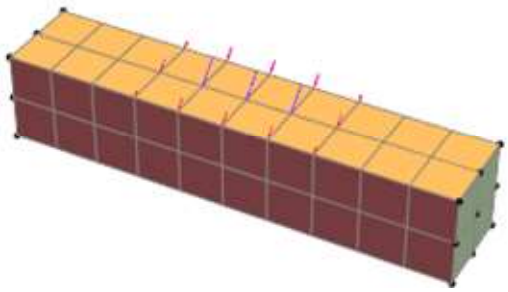
In[136]:=
```
largeStains = True;
pathDependent = False;
```

In[138]:=
```
macroElement =
  "ExamplesH1" <> If[largeStains, "PF", "se"] <> "Macro" <> If[pathDependent, "Unsymm", "Symm"];
```

In[139]:=
```
L = 12; B = 2.4; H = 2.4;
n = 2; nx = 5 * n; ny = 1 * n; nz = 1 * n;
p = 50;
SMTInputData["Threads" → 1];
SMTAddDomain[{"beam", macroElement, {}}];
points = {{0, 0, 0}, {L, 0, 0}, {L, B, 0}, {0, B, 0}, {0, 0, H}, {L, 0, H}, {L, B, H}, {0, B, H}};
SMTAddMesh[Hexahedron[points], "beam", "H1", {nx, ny, nz}];
SMTAddEssentialBoundary[
  Polygon[{{0, 0, 0}, {0, 0, H}, {0, B, H}, {0, B, 0}}], 1 → 0, 2 → 0, 3 → 0];
SMTAddEssentialBoundary[Polygon[{{L, 0, 0}, {L, 0, H}, {L, B, H}, {L, B, 0}}], 1 → 0, 2 → 0, 3 → 0];
SMTAddNaturalBoundary[
  Polygon[{{L / 10 * 3, 0, H}, {L / 10 * 7, 0, H}, {L / 10 * 7, B, H}, {L / 10 * 3, B, H}}],
  2 → Polygon[{-p}], 3 → Polygon[{-p}]];
SMTAnalysis[];
FE2Elements = SMTFindElements["beam"];
Length[FE2Elements]
```

```
40
```

In[152]:=
```
SMTShowMesh["BoundaryConditions" → True]
```



Micro problem

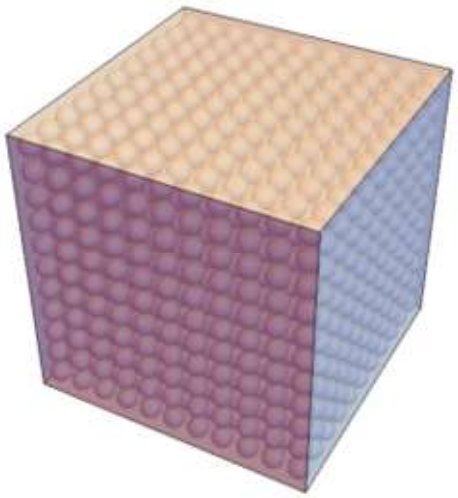
In[153]:=
```
nn = 2; (*micro devision*)
aRVE = 1; (*RVE length *)
bRVE = 1; (*RVE height *)
cRVE = 1; (*RVE height *)
pRVE = 0.3; (* percentage of perforation*)
rRVE = CubeRoot[(pRVE aRVE bRVE cRVE 3) / (π 4)]; (* radius of perforation*)
NrRVE = nn; (* RVE mesh densitiy in radial direction*)
NabRVE = nn;
```

- The actual micro mesh is generated by the user defined FE23DMicroMeshVoids[*localdata*, *additionalmicrodata*] function where *localdata* is a standard local data structure (see documentation) and *additionalmicrodata* is an arbitrary user defined data needed by the FE22DMicroMeshVoids function in order to create micro problem mesh.

In[161]:=
```
Clear[specificMacroData]
specificMacroData =
  {aRVE , bRVE, cRVE, rRVE, NrRVE , NabRVE, SMTMakeDll["ExamplesSED3O1DFHYO1DNeoHookeWA"],
   {"E *" → 2000, "ν *" → 0.3}, "O1", SMTMakeDll["ExamplesPeriodic3DUnstructured"],
   SMTMakeDll["ExamplesSurfaceTriangulationP1"], {"MinSubSteps" → 1, "MaxSubSteps" → 10}};
```

- Micro structure is infinitely periodic

In[163]:=
```
n = 10;
Graphics3D[{Opacity[0.6], Cuboid[{0, 0, 0}, {(n + 1) aRVE, (n + 1) bRVE, (n + 1) cRVE}],
  Table[Sphere[{x + aRVE / 2, y + bRVE / 2, z + cRVE / 2}, rRVE],
   {x, 0, n aRVE, aRVE}, {y, 0, n bRVE, bRVE}, {z, 0, n cRVE, cRVE}]}, Boxed → False]
```



Set up multi-scale environment

In[164]:=
```
SMTMultiScaleSet[
 "FE^2" → {{FE23DMicroMeshVoids, FE2SolveOne} -> FE2Elements},
 "SpecificMicroData" → {"FE23DMicroMeshVoids" -> specificMacroData},
"MaxKernels" → Automatic, "Threads" → 1,
 "Console" → False, "PartialDump" → True, "PartialRestart" → True]
```

```
True
```

Visualize selected micro problem

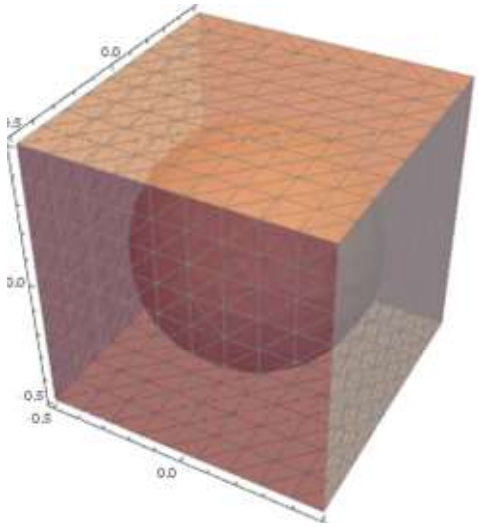
- SMTMicroRestart[{x, y,z}] ... restart micro problem that is nearest to the given spatial position (position is given in initial coordinates) and set the selected micro problem as the current micro problem

In[165]:=
```
SMTMicroRestart[{0, 0, 0}]
```

```
{{FE^2, FE23DMicroMeshVoids, FE2SolveOne, 8,
  Deformation gradient (3D), 9, Integrated stress and sensitivity (P, 3D), 90},
 {0.25359, 0.25359, 0.25359}, {1, 1, 1, 0.415283, 2, 2,
  {C:\Users\jkorelc\AppData\Roaming\Mathematica\Applications\AceFEM\Elements\
     ExamplesSED3O1DFHYO1DNeoHookeWA.W64.dll,
   ExamplesSED3O1DFHYO1DNeoHookeWA, 0}, {E * → 2000, ν * → 0.3}, O1,
  {C:\Users\jkorelc\Desktop\SMS\Documentation\ExamplesPeriodic3DUnstructured.W64.dll,
   ExamplesPeriodic3DUnstructured, 0},
  {C:\Users\jkorelc\Desktop\SMS\Documentation\ExamplesSurfaceTriangulationP1.W64.dll,
   ExamplesSurfaceTriangulationP1, 0}, {MinSubSteps → 1, MaxSubSteps → 10}}, 1, 5, 1}
```

- SMTMicroEvaluate[exp] ... evaluates exp for the restarted micro problem

In[166]:=
```
SMTMicroEvaluate[SMTShowMesh[Axes → True, "Opacity" → 0.6, "ZoomElements" → "micro"]]
```



- local_problem_data structure of selected micro problem

In[167]:= **SMTCurrentLocalProblem**

```
{{FE^2, FE23DMicroMeshVoids, FE2SolveOne, 8,
  Deformation gradient (3D), 9, Integrated stress and sensitivity (P, 3D), 90},
 {0.25359, 0.25359, 0.25359}, {1, 1, 1, 0.415283, 2, 2,
  {C:\Users\jkorelc\AppData\Roaming\Mathematica\Applications\AceFEM\Elements\
     ExamplesSED3O1DFHYO1DNeoHookeWA.W64.dll,
   ExamplesSED3O1DFHYO1DNeoHookeWA, 0}, {E * → 2000, ν * → 0.3}, O1,
  {C:\Users\jkorelc\Desktop\SMS\Documentation\ExamplesPeriodic3DUnstructured.W64.dll,
   ExamplesPeriodic3DUnstructured, 0},
  {C:\Users\jkorelc\Desktop\SMS\Documentation\ExamplesSurfaceTriangulationP1.W64.dll,
   ExamplesSurfaceTriangulationP1, 0}, {MinSubSteps → 1, MaxSubSteps → 10}},
 1, 5, 1, {0, 0, 0, 0, 0, 0, 0, 0, 0}, {147, 29, 667, 752, 129, 3, 650, 743},
 1, {MinSubSteps → 1, MaxSubSteps → 10}, Null}
```

Evaluate complete response using an adaptive path following procedure

In[168]:=
```
SMTAnimationOfResponse["Initialize", {0, 0}];
λMax = 1; λ0 = λMax / 10; ΔλMin = λMax / 1000; ΔλMax = λMax / 10;
tolNR = 10^-8; maxNR = 15; targetNR = 8;
SMTNextStep["λ" -> λ0];
While[
  While[
   step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λMax}]
   , SMTMicroSolve["Dump" → False];
   (* react on events at RVE *)
   Switch[SMTSharedStatus[[1]]
    , 0 | 1,
    SMTNewtonIteration[];
    SMTStatusReport[SMTSharedStatus[[1]]];
    , 2,
    (* force step back due to the failed micro simulation *)
    SMTIData[{"Iteration", "TotalIteration"}, SMTIData[{"Iteration", "TotalIteration"}] + 1];
    SMTIData["ErrorStatus", 2];
    SMTStatusReport[SMTSharedStatus[[2]]];
    , 3,
    (*abort due to the fatal error *)
    SMTStatusReport[SMTSharedStatus[[2]]]; Abort[];
   ]
  ]; (* end of NR loop *)
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; Abort[];];
  (* successful step *)
  If[Not[step[[1]]],
   (*dump the state of micro problems to disc before the next step*)
   SMTMicroSolve["Dump" → True];
   SMTAnimationOfResponse[
    "LeadingNodePosition" → {L / 2, 0, 0}, "x" → Hold[-SMTPostData["v", Point[{L / 2, 0, 0}]]]
    , "Show" → "Window" | {"ExportFrames", "responseFile"}
   ]
  ];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" -> step[[2]]]
];
```

```
Step/Iter=1/1 λ/Δλ=0.1/0.1 ‖Δp‖/‖R‖=0.101024/1.49666 Events=0 Status=0/{} Tag=0

Step/Iter=1/2 λ/Δλ=0.1/0.1 ‖Δp‖/‖R‖=0.00279489/2.65007 Events=0 Status=0/{} Tag=0

Step/Iter=1/3 λ/Δλ=0.1/0.1 ‖Δp‖/‖R‖=0.000728749/0.00910695 Events=0 Status=0/{} Tag=0

Step/Iter=1/4 λ/Δλ=0.1/0.1 ‖Δp‖/‖R‖=4.36707×10^-7/0.00066179 Events=0 Status=0/{} Tag=0

Step/Iter=1/5 λ/Δλ=0.1/0.1 ‖Δp‖/‖R‖=
 1.75696×10^-11/2.86164×10^-10 Events=0 Status=0/{} Tag=0

Step/Iter=2/1 λ/Δλ=0.2/0.1 ‖Δp‖/‖R‖=0.103466/1.49666 Events=0 Status=0/{} Tag=0

Step/Iter=2/2 λ/Δλ=0.2/0.1 ‖Δp‖/‖R‖=0.00308689/3.27434 Events=0 Status=0/{} Tag=0

Step/Iter=2/3 λ/Δλ=0.2/0.1 ‖Δp‖/‖R‖=0.000908646/0.0106008 Events=0 Status=0/{} Tag=0

Step/Iter=2/4 λ/Δλ=0.2/0.1 ‖Δp‖/‖R‖=6.17495×10^-7/0.00103647 Events=0 Status=0/{} Tag=0

Step/Iter=2/5 λ/Δλ=0.2/0.1 ‖Δp‖/‖R‖=
 1.78087×10^-11/6.29273×10^-10 Events=0 Status=0/{} Tag=0
```

Step/Iter=3/1 λ/Δλ=0.3/0.1 ‖Δp‖/‖R‖=0.10436/1.49666 Events=0 Status=0/{} Tag=0

Step/Iter=3/2 λ/Δλ=0.3/0.1 ‖Δp‖/‖R‖=0.00350527/3.96394 Events=0 Status=0/{} Tag=0

Step/Iter=3/3 λ/Δλ=0.3/0.1 ‖Δp‖/‖R‖=0.000974952/0.0123289 Events=0 Status=0/{} Tag=0

Step/Iter=3/4 λ/Δλ=0.3/0.1 ‖Δp‖/‖R‖=1.06928×10^{-6}/0.00119745 Events=0 Status=0/{} Tag=0

Step/Iter=3/5 λ/Δλ=0.3/0.1 ‖Δp‖/‖R‖=
1.39546×10^{-10}/1.52366×10^{-9} Events=0 Status=0/{} Tag=0

Step/Iter=4/1 λ/Δλ=0.4/0.1 ‖Δp‖/‖R‖=0.103857/1.49666 Events=0 Status=0/{} Tag=0

Step/Iter=4/2 λ/Δλ=0.4/0.1 ‖Δp‖/‖R‖=0.00383852/4.62819 Events=0 Status=0/{} Tag=0

Step/Iter=4/3 λ/Δλ=0.4/0.1 ‖Δp‖/‖R‖=0.000827077/0.0144727 Events=0 Status=0/{} Tag=0

Step/Iter=4/4 λ/Δλ=0.4/0.1 ‖Δp‖/‖R‖=1.42543×10^{-6}/0.000859255 Events=0 Status=0/{} Tag=0

Step/Iter=4/5 λ/Δλ=0.4/0.1 ‖Δp‖/‖R‖=
1.67833×10^{-10}/2.59929×10^{-9} Events=0 Status=0/{} Tag=0

Step/Iter=5/1 λ/Δλ=0.5/0.1 ‖Δp‖/‖R‖=0.102044/1.49666 Events=0 Status=0/{} Tag=0

Step/Iter=5/2 λ/Δλ=0.5/0.1 ‖Δp‖/‖R‖=0.00400931/5.12037 Events=0 Status=0/{} Tag=0

Step/Iter=5/3 λ/Δλ=0.5/0.1 ‖Δp‖/‖R‖=0.000456002/0.0165208 Events=0 Status=0/{} Tag=0

Step/Iter=5/4 λ/Δλ=0.5/0.1 ‖Δp‖/‖R‖=9.53574×10^{-7}/0.000250768 Events=0 Status=0/{} Tag=0

Step/Iter=5/5 λ/Δλ=0.5/0.1 ‖Δp‖/‖R‖=
3.44706×10^{-11}/1.17143×10^{-9} Events=0 Status=0/{} Tag=0

Step/Iter=6/1 λ/Δλ=0.6/0.1 ‖Δp‖/‖R‖=0.0988993/1.49666 Events=0 Status=0/{} Tag=0

Step/Iter=6/2 λ/Δλ=0.6/0.1 ‖Δp‖/‖R‖=0.00401604/5.30233 Events=0 Status=0/{} Tag=0

Step/Iter=6/3 λ/Δλ=0.6/0.1 ‖Δp‖/‖R‖=0.000118885/0.0174085 Events=0 Status=0/{} Tag=0

Step/Iter=6/4 λ/Δλ=0.6/0.1 ‖Δp‖/‖R‖=
8.25432×10^{-8}/3.38804×10^{-6} Events=0 Status=0/{} Tag=0

Step/Iter=6/5 λ/Δλ=0.6/0.1 ‖Δp‖/‖R‖=
2.75314×10^{-14}/8.74254×10^{-12} Events=0 Status=0/{} Tag=0

Step/Iter=7/1 λ/Δλ=0.7/0.1 ‖Δp‖/‖R‖=0.0945168/1.49666 Events=0 Status=0/{} Tag=0

Step/Iter=7/2 λ/Δλ=0.7/0.1 ‖Δp‖/‖R‖=0.0038812/5.13263 Events=0 Status=0/{} Tag=0

Step/Iter=7/3 λ/Δλ=0.7/0.1 ‖Δp‖/‖R‖=0.000439628/0.0165862 Events=0 Status=0/{} Tag=0

Step/Iter=7/4 λ/Δλ=0.7/0.1 ‖Δp‖/‖R‖=7.02314×10^{-7}/0.000236251 Events=0 Status=0/{} Tag=0

Step/Iter=7/5 λ/Δλ=0.7/0.1 ‖Δp‖/‖R‖=
2.13431×10^{-11}/6.36194×10^{-10} Events=0 Status=0/{} Tag=0

Step/Iter=8/1 λ/Δλ=0.8/0.1 ‖Δp‖/‖R‖=0.0892406/1.49666 Events=0 Status=0/{} Tag=0

Step/Iter=8/2 λ/Δλ=0.8/0.1 ‖Δp‖/‖R‖=0.00364627/4.68776 Events=0 Status=0/{} Tag=0

Step/Iter=8/3 λ/Δλ=0.8/0.1 ‖Δp‖/‖R‖=0.00063487/0.0144967 Events=0 Status=0/{} Tag=0

Step/Iter=8/4 λ/Δλ=0.8/0.1 ‖Δp‖/‖R‖=6.62306×10^{-7}/0.000510254 Events=0 Status=0/{} Tag=0

Step/Iter=8/5 λ/Δλ=0.8/0.1 ‖Δp‖/‖R‖=
3.76588×10^{-11}/5.66168×10^{-10} Events=0 Status=0/{} Tag=0

Step/Iter=9/1 λ/Δλ=0.9/0.1 ‖Δp‖/‖R‖=0.0835598/1.49666 Events=0 Status=0/{} Tag=0

Step/Iter=9/2 λ/Δλ=0.9/0.1 ‖Δp‖/‖R‖=0.00335824/4.10019 Events=0 Status=0/{} Tag=0

Step/Iter=9/3 λ/Δλ=0.9/0.1 ‖Δp‖/‖R‖=0.000659085/0.011985 Events=0 Status=0/{} Tag=0

Step/Iter=9/4 λ/Δλ=0.9/0.1 ‖Δp‖/‖R‖=4.09348×10^{-7}/0.000554306 Events=0 Status=0/{} Tag=0

Step/Iter=9/5 $\lambda/\Delta\lambda$=0.9/0.1 ‖Δp‖/‖R‖=
1.65945×10^-11/2.4218×10^-10 Events=0 Status=0/{} Tag=0

Step/Iter=10/1 $\lambda/\Delta\lambda$=1./0.1 ‖Δp‖/‖R‖=0.0779172/1.49666 Events=0 Status=0/{} Tag=0

Step/Iter=10/2 $\lambda/\Delta\lambda$=1./0.1 ‖Δp‖/‖R‖=0.00305211/3.48656 Events=0 Status=0/{} Tag=0

Step/Iter=10/3 $\lambda/\Delta\lambda$=1./0.1 ‖Δp‖/‖R‖=0.000580862/0.00963367 Events=0 Status=0/{} Tag=0

Step/Iter=10/4 $\lambda/\Delta\lambda$=1./0.1 ‖Δp‖/‖R‖=2.50268×10^-7/0.000431873 Events=0 Status=0/{} Tag=0

Step/Iter=10/5 $\lambda/\Delta\lambda$=1./0.1 ‖Δp‖/‖R‖=
1.18984×10^-12/1.09503×10^-10 Events=0 Status=0/{} Tag=0

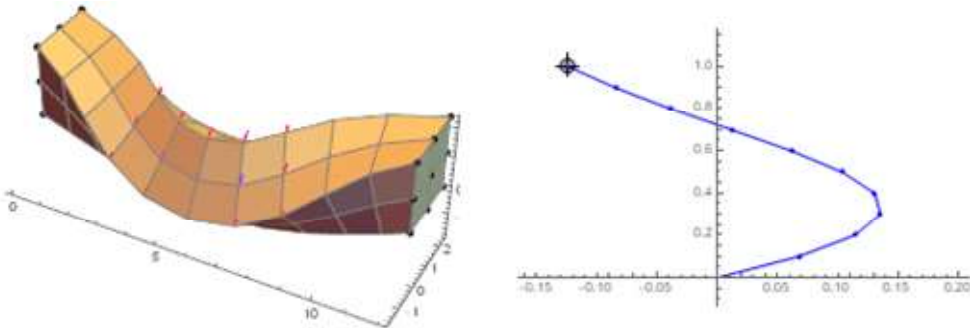
In[173]:= **SMTMultiScaleSimulationReport[];**

MACRO	
Number of macro elements	40
Number of macro equations	243
No. of steps	10
No. of steps back	0
Total no. of iterations	50
Total driver time (s)	8:14.28
Total linear solver time (s)	0.04
Total K&R time (s)	0.10
MICRO	
Number of FE^2 elements	40
Number of MIEL elements	0
Number of single-scale elements	0
Parallelization kernels/threads	16/1
Total number of micro problems	320
Total number of micro elements	901 440
Total number of micro nodes	340 160
Total number of micro equations	1 012 800
Average number of micro equations	3165.
Real micro mesh generation time	19.83
Number of MicroSolve calls	60
Total MicroSolve time	7:32.63
Total number of MP equations solved	60 768 000
Total number of steps	22 400
Total number of back steps	0
Average number of micro problems/kernel	20.
Total number of MP solved	19 200
Average MP time	0.33
Average MP K&R time	0.05
Average MP linear solver time	0.26
Average MP sens.p.load time	0.01
Average MP sens.l.solver time	0.01
Average MP Schur complement time	0
Average MP task time	0.00
Average MP Driver time	0.38
Average real Driver time	7:40.03
Total parallel MP time	1:44:20.17
Number of restarts	19 200
Total parallel restart time	5:26.37
Average restart time	0.02
Number of dumps	3520
Total parallel dump time	2:41.27
Average dump time	0.05
Total dump files size (byte)	140 995 187
zlib compression level	2
Macro + micro analysis time	7:52.60
Aministrative time	7.63
Lost time:	1:1.37
Total time	8:0.23

Analysis of the results

Response curve

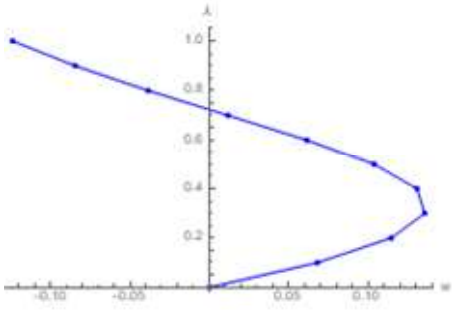
In[174]:= **SMTAnimationOfResponse["Report"]**



In[175]:=
```
response = SMTAnimationOfResponse["Response"]
```

```
{{{0, 0}, {0.0678648, 0.1}, {0.114334, 0.2}, {0.135371, 0.3},
  {0.130531, 0.4}, {0.103549, 0.5}, {0.0613196, 0.6}, {0.0117219, 0.7},
  {-0.0384485, 0.8}, {-0.0844831, 0.9}, {-0.123738, 1.}}}
```

In[176]:=
```
ListLinePlot[response, AxesLabel → {"w", "λ"}, PlotMarkers → Automatic, PlotStyle → Blue]
```



Calculate stiffness (KN/cm)

In[177]:=
```
response[[1, 2, 2]] / response[[1, 2, 1]]
```

```
1.47352
```

Visualization of the chosen micro problem

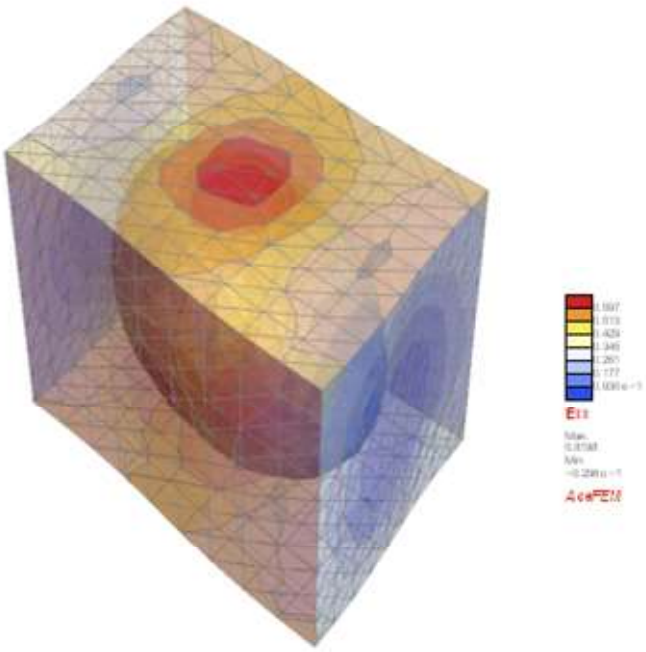
- SMTMicroRestart[{x, y}] ... restart micro problem that is nearest to the given spatial position (position is given in initial coordinates) and set the selected micro problem as the current micro problem

In[178]:=
```
SMTMicroRestart[{0, B, H}]
```

```
{{FE^2, FE23DMicroMeshVoids, FE2SolveOne, 8,
  Deformation gradient (3D), 9, Integrated stress and sensitivity (P, 3D), 90},
 {0.25359, 2.14641, 2.14641}, {1, 1, 1, 0.415283, 2, 2,
  {C:\Users\jkorelc\AppData\Roaming\Mathematica\Applications\AceFEM\Elements\
     ExamplesSED3O1DFHYO1DNeoHookeWA.W64.dll,
   ExamplesSED3O1DFHYO1DNeoHookeWA, 0}, {E * → 2000, ν * → 0.3}, O1,
  {C:\Users\jkorelc\Desktop\SMS\Documentation\ExamplesPeriodic3DUnstructured.W64.dll,
   ExamplesPeriodic3DUnstructured, 0},
  {C:\Users\jkorelc\Desktop\SMS\Documentation\ExamplesSurfaceTriangulationP1.W64.dll,
   ExamplesSurfaceTriangulationP1, 0}, {MinSubSteps → 1, MaxSubSteps → 10}}, 31, 5, 4}
```

- SMTMicroEvaluate[exp] ... evaluate exp for the current micro problem

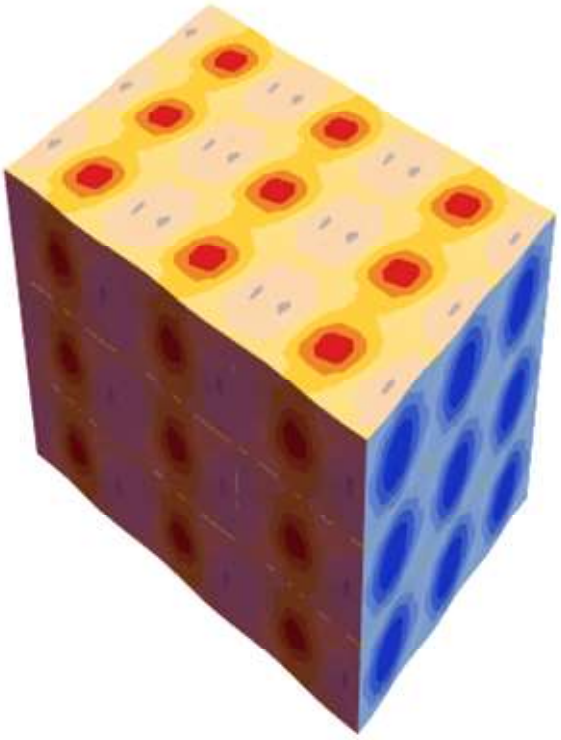
In[179]:=
```
mesh = SMTMicroEvaluate[SMTShowMesh["DeformedMesh" → True, "Field" -> "Exx", "Mesh" → True,
    "Contour" → True, "Legend" → True, "ZoomElements" → "micro", "Opacity" → 0.5, ImageSize → 600]]
```



- Representation of 3D periodic micro-structure.

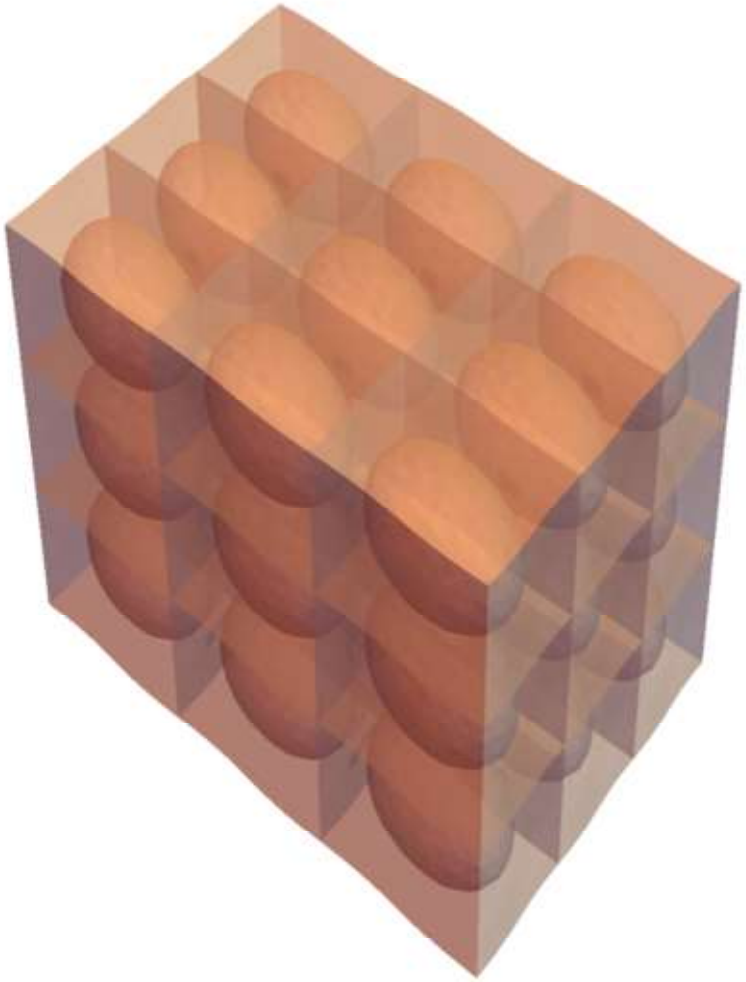
In[180]:=
```
infper = SMTMicroEvaluate[
  {aRVE, bRVE, cRVE} = SMTCurrentLocalProblem[[3, {1, 2, 3}]];
  corner =
   Map[SMTNodeData[Point[#], "at"][[1]] &,
    {{-aRVE / 2, -bRVE / 2, -cRVE / 2}, {aRVE / 2, -bRVE / 2, -cRVE / 2},
     {aRVE / 2, bRVE / 2, -cRVE / 2}, {-aRVE / 2, bRVE / 2, -cRVE / 2}, {-aRVE / 2, -bRVE / 2, cRVE / 2},
     {aRVE / 2, -bRVE / 2, cRVE / 2}, {aRVE / 2, bRVE / 2, cRVE / 2}, {-aRVE / 2, bRVE / 2, cRVE / 2}}];
  Table[SMTShowMesh["DeformedMesh" → {i aRVE, j bRVE, k cRVE} + i (corner[[2]] - corner[[1]]) +
      j (corner[[4]] - corner[[1]]) + k (corner[[7]] - corner[[3]]) +
      {SMTPostData["u"], SMTPostData["v"], SMTPostData["w"]},
    "Mesh" → False, "Field" → "Exx", "Legend" → False, "Contour" -> True]
   , {i, 0, 2}, {j, 0, 2}, {k, 0, 2}]
 ];
```

In[181]:=
```
Show[infper, ImageSize → 600]
```



In[182]:=
```
infper = SMTMicroEvaluate[
    {aRVE, bRVE, cRVE} = SMTCurrentLocalProblem[[3, {1, 2, 3}]];
    corner =
     Map[SMTNodeData[Point[#], "at"][[1]] &,
      {{-aRVE / 2, -bRVE / 2, -cRVE / 2}, {aRVE / 2, -bRVE / 2, -cRVE / 2},
       {aRVE / 2, bRVE / 2, -cRVE / 2}, {-aRVE / 2, bRVE / 2, -cRVE / 2}, {-aRVE / 2, -bRVE / 2, cRVE / 2},
       {aRVE / 2, -bRVE / 2, cRVE / 2}, {aRVE / 2, bRVE / 2, cRVE / 2}, {-aRVE / 2, bRVE / 2, cRVE / 2}}];
    Table[SMTShowMesh["DeformedMesh" →  {i aRVE, j bRVE, k cRVE} + i (corner[[3]] - corner[[4]]) +
        j (corner[[4]] - corner[[1]]) + k (corner[[7]] - corner[[3]]) +
        {SMTPostData["u"], SMTPostData["v"], SMTPostData["w"]}, "Mesh" → False,
      "Legend" → False, "Contour" → False, "ZoomElements" → "micro", "Opacity" → 0.5]
     , {i, 0, 2}, {j, 0, 2}, {k, 0, 2}]
   ];
```

In[183]:=
```
Show[infper, ImageSize → 800]
```

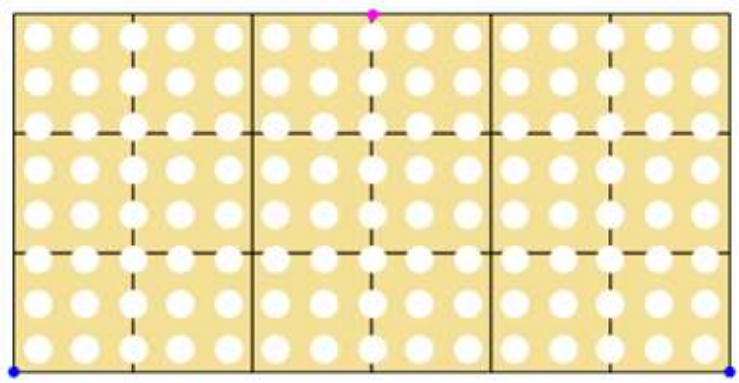


Examples of MIEL multi-scale modeling

MIEL modeling of 2D uniformly distributed micro-structure

Description

Perform simulation of the beam like structure made of perforated material. The real size of perforations is comparable with dimensions of the problem, thus the FE^2 approach can not give accurate structural response. The MIEL multi scale approach is used instead.



Macro problem

In[184]:=
```
<< AceFEM`;
```

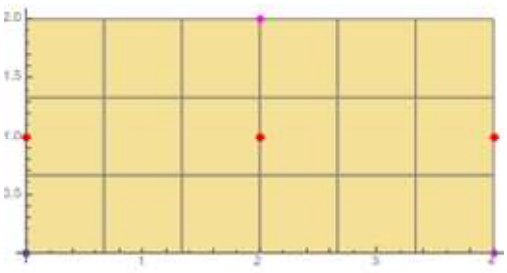
In[185]:=
```
If[SMTMultiScaleLoaded =!= True,
  MessageDialog["Load Multi-scale computational environment first!"];
  Quit[]];
```

In[186]:=
```
L = 4.; (*length cm/KN*)
H = 2.; (*height*)
vA = 1;(* prescribed displacement at L/2*)
Nx = 6; (*mesh density in x direction*)
Ny = 3; (*mesh density in y direction*)
SMTInputData["Threads" → 1];
SMTAddDomain[{"beam", "ExamplesQ1MIELMacroSymm", {}}];
SMTAddEssentialBoundary[
  {Point[{0, 0}], 1 -> 0, 2 -> 0}
  , {Point[{L, 0}], 2 → 0}
  , { Point[{L / 2, H}] , 2 → -vA}];
SMTAddMesh[Polygon[{{0, 0}, {L, 0}, {L, H}, {0, H}}], "beam", "Q1", {Nx, Ny}];
SMTAnalysis[];
MIELElements = SMTFindElements["beam"];
Length[MIELElements]
```

```
18
```

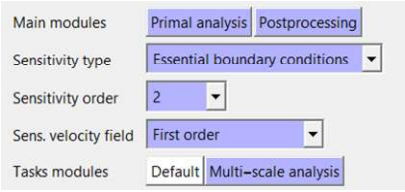
- Macro mesh and selected points for later post-processing

In[198]:=
```
monitorpoints = Table[{x, H / 2}, {x, {0., 0.5 L, L}}];
SMTShowMesh["BoundaryConditions" → True,
 Epilog → {PointSize[Large], Red, Point[monitorpoints]}, Axes → True]
```



Micro problem

Micro problem is meshed with unstructured mesh. Thus three noded element SEPET1DFHYT1DNeoHookeWA must be first generated using AceShare (check task Multi-scale analysis!).



- specific_micro_data={
  - 1 {{{x1,y1},r1},{{x2,y2},r2},…} - all disks where {xi,yi} is a center of the i-th voids and ri is the radius of the i-th void
  - 2 division - approximate number of elements per side
  - 3 micro_element - SMTMakeDll[micro_element_UEC] - element used to discretize the solid domain
  - 4 micro_material - material data related to micro element (see `SMTAddDomain`)
  - 5 topology - micro mesh type (see `Element Topology`)
  - 6 list of solution procedure options - given options are stored in *local_problem_data[[10]]* and interpreted later by the MIELSolveOne function. Current options are:
  - "MinSubSteps"->*n* - minimum number of micro sub-steps per one macro step (default *n*=1)
  - "MaxSubSteps"->*n* ⇒ defines a maximum number of micro sub-steps per macro step (default 100)
  - }

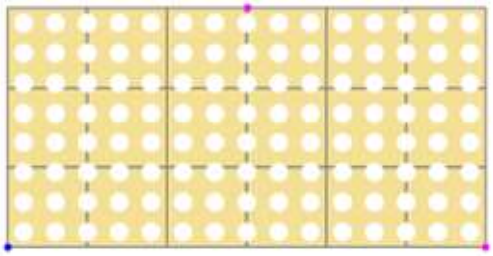
In[200]:=
```
n1 = 15; n2 = 8; r = 0.08;
disks =
  Flatten[Table[{{L / n1 / 2 + L / n1 i, H / n2 / 2 + H / n2 j}, r}, {i, 0, n1 - 1}, {j, 0, n2 - 1}], 1];
specificMacroData = {disks, 20, SMTMakeDll["ExamplesSEPET1DFHYT1DNeoHookeWA"],
   {"E *" → 21000, "ν *" → 0.3}, "T1", {"MinSubSteps" → 1, "MaxSubSteps" → 10}};
```

- percentage of perforation

In[203]:=
```
n1 n2 π r^2 / (L H)
```

Out[203]=
```
0.301593
```

In[204]:=
```
Show[SMTShowMesh["BoundaryConditions" → True],
 Graphics[{White, Map[Disk[#[[1]], #[[2]]] &, disks]}]]
```



Set up multi-scale environment

- The "PartialRestart" option can be True only when the local problems are structurally the same mesh (only the actual coordinates of the nodes can be different).
- The "PartialDump" must be False only when during the solution of the local problem the mesh has been changed as well.

In[205]:=
```
SMTMultiScaleSet[
  "MIEL" → {{MIEL2DMicroMeshVoids, MIELSolveOne} -> MIELElements},
  "SpecificMicroData" → {"MIEL2DMicroMeshVoids" -> specificMacroData},
 "MaxKernels" → Automatic, "Threads" → 1,
  "Console" → False, "PartialDump" → True, "PartialRestart" → False]
```

```
True
```

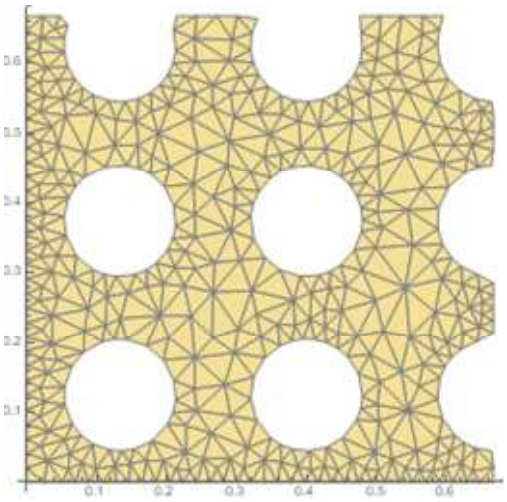
Visualize selected micro problem

- SMTMicroRestart[{x, y}] ... restart micro problem that is nearest to the given spatial position (position is given in initial coordinates) and set the selected micro problem as the current micro problem

In[206]:=
```
SMTMicroRestart[{0, 0}];
```

- SMTMicroEvaluate[exp] ... evaluate exp for the restarted micro problem

In[207]:=
```
SMTMicroEvaluate[SMTShowMesh[Axes → True, "Marks" → False]]
```



- local_problem_data structure of selected micro problem

In[208]:=
```
SMTCurrentLocalProblem
```

```
{{MIEL, MIEL2DMicroMeshVoids, MIELSolveOne, 1, Nodal displacements (2D), 8,
  Integrated strain energy and derivatives (plane strain), 45}, {0.333333, 0.333333},
 {{{{0.133333, 0.125}, 0.08}, {{0.133333, 0.375}, 0.08}, {{0.133333, 0.625}, 0.08},
   {{0.133333, 0.875}, 0.08}, {{0.133333, 1.125}, 0.08}, {{0.133333, 1.375}, 0.08},
   {{0.133333, 1.625}, 0.08}, {{0.133333, 1.875}, 0.08}, {{0.4, 0.125}, 0.08},
   {{0.4, 0.375}, 0.08}, {{0.4, 0.625}, 0.08}, {{0.4, 0.875}, 0.08},
   {{0.4, 1.125}, 0.08}, {{0.4, 1.375}, 0.08}, {{0.4, 1.625}, 0.08},
   {{0.4, 1.875}, 0.08}, {{0.666667, 0.125}, 0.08}, {{0.666667, 0.375}, 0.08},
   {{0.666667, 0.625}, 0.08}, {{0.666667, 0.875}, 0.08}, {{0.666667, 1.125}, 0.08},
   {{0.666667, 1.375}, 0.08}, {{0.666667, 1.625}, 0.08}, {{0.666667, 1.875}, 0.08},
   {{0.933333, 0.125}, 0.08}, {{0.933333, 0.375}, 0.08}, {{0.933333, 0.625}, 0.08},
   {{0.933333, 0.875}, 0.08}, {{0.933333, 1.125}, 0.08}, {{0.933333, 1.375}, 0.08},
   {{0.933333, 1.625}, 0.08}, {{0.933333, 1.875}, 0.08}, {{1.2, 0.125}, 0.08},
   {{1.2, 0.375}, 0.08}, {{1.2, 0.625}, 0.08}, {{1.2, 0.875}, 0.08},
   {{1.2, 1.125}, 0.08}, {{1.2, 1.375}, 0.08}, {{1.2, 1.625}, 0.08},
   {{1.2, 1.875}, 0.08}, {{1.46667, 0.125}, 0.08}, {{1.46667, 0.375}, 0.08},
   {{1.46667, 0.625}, 0.08}, {{1.46667, 0.875}, 0.08}, {{1.46667, 1.125}, 0.08},
   {{1.46667, 1.375}, 0.08}, {{1.46667, 1.625}, 0.08}, {{1.46667, 1.875}, 0.08},
   {{1.73333, 0.125}, 0.08}, {{1.73333, 0.375}, 0.08}, {{1.73333, 0.625}, 0.08},
   {{1.73333, 0.875}, 0.08}, {{1.73333, 1.125}, 0.08}, {{1.73333, 1.375}, 0.08},
   {{1.73333, 1.625}, 0.08}, {{1.73333, 1.875}, 0.08}, {{2., 0.125}, 0.08},
   {{2., 0.375}, 0.08}, {{2., 0.625}, 0.08}, {{2., 0.875}, 0.08},
   {{2., 1.125}, 0.08}, {{2., 1.375}, 0.08}, {{2., 1.625}, 0.08},
   {{2., 1.875}, 0.08}, {{2.26667, 0.125}, 0.08}, {{2.26667, 0.375}, 0.08},
   {{2.26667, 0.625}, 0.08}, {{2.26667, 0.875}, 0.08}, {{2.26667, 1.125}, 0.08},
   {{2.26667, 1.375}, 0.08}, {{2.26667, 1.625}, 0.08}, {{2.26667, 1.875}, 0.08},
   {{2.53333, 0.125}, 0.08}, {{2.53333, 0.375}, 0.08}, {{2.53333, 0.625}, 0.08},
   {{2.53333, 0.875}, 0.08}, {{2.53333, 1.125}, 0.08}, {{2.53333, 1.375}, 0.08},
   {{2.53333, 1.625}, 0.08}, {{2.53333, 1.875}, 0.08}, {{2.8, 0.125}, 0.08},
   {{2.8, 0.375}, 0.08}, {{2.8, 0.625}, 0.08}, {{2.8, 0.875}, 0.08},
   {{2.8, 1.125}, 0.08}, {{2.8, 1.375}, 0.08}, {{2.8, 1.625}, 0.08},
   {{2.8, 1.875}, 0.08}, {{3.06667, 0.125}, 0.08}, {{3.06667, 0.375}, 0.08},
   {{3.06667, 0.625}, 0.08}, {{3.06667, 0.875}, 0.08}, {{3.06667, 1.125}, 0.08},
   {{3.06667, 1.375}, 0.08}, {{3.06667, 1.625}, 0.08}, {{3.06667, 1.875}, 0.08},
   {{3.33333, 0.125}, 0.08}, {{3.33333, 0.375}, 0.08}, {{3.33333, 0.625}, 0.08},
   {{3.33333, 0.875}, 0.08}, {{3.33333, 1.125}, 0.08}, {{3.33333, 1.375}, 0.08},
   {{3.33333, 1.625}, 0.08}, {{3.33333, 1.875}, 0.08}, {{3.6, 0.125}, 0.08},
   {{3.6, 0.375}, 0.08}, {{3.6, 0.625}, 0.08}, {{3.6, 0.875}, 0.08},
   {{3.6, 1.125}, 0.08}, {{3.6, 1.375}, 0.08}, {{3.6, 1.625}, 0.08},
   {{3.6, 1.875}, 0.08}, {{3.86667, 0.125}, 0.08}, {{3.86667, 0.375}, 0.08},
   {{3.86667, 0.625}, 0.08}, {{3.86667, 0.875}, 0.08}, {{3.86667, 1.125}, 0.08},
   {{3.86667, 1.375}, 0.08}, {{3.86667, 1.625}, 0.08}, {{3.86667, 1.875}, 0.08}}, 20,
  {C:\Users\jkorelc\AppData\Roaming\Mathematica\Applications\AceFEM\Elements\
     ExamplesSEPET1DFHYT1DNeoHookeWA.W64.dll,
   ExamplesSEPET1DFHYT1DNeoHookeWA, 0}, {E * → 21000, ν * → 0.3}, T1,
  {MinSubSteps → 1, MaxSubSteps → 10}}, 1, 5, 1, {0, 0, 0, 0, 0, 0, 0, 0},
 {{0., 0.}, {0.666667, 0.}, {0.666667, 0.666667}, {0., 0.666667}}, Q1,
 {{{0., 0.}}, {{0.666667, 0.}}, {{0.666667, 0.527901}, {0.595238, 0.666667}},
  {{0., 0.666667}}}, 1, {MinSubSteps → 1, MaxSubSteps → 10}, True, Null}
```

Perform and analyze single Newton-Raphson iteration

In[209]:=
```
NoDOF = 8
```

```
8
```

- SMTNextStep["λ"-> 1] - increment global load level

In[210]:=
```
SMTNextStep["λ" -> 1];
```

In[211]:=
```
testelem = 1;
```

- "MIEL" task returns a list
  - {
  - 1 - number of micro problems associated with the element which is in the case of MIEL formulation 1
  - 2 - index of character switch constant of macro element that defines the name of the task *macro_data_task* that returns multi-scale related macro data
  - 3 - length of multi-scale related macro data
  - 4 -index of character switch constant of macro element that defines the name of the micro problem task that returns multi-scale related micro data or *micro_data_task*
  - 5 - length of multi-scale related micro data
  - }

In[212]:=
```
init = SMTTask["MIEL", "Elements" → testelem]
```

```
{1, 2, 8, 3, 45}
```

In[213]:=
```
1 + NoDOF + NoDOF * (NoDOF + 1) / 2
```

```
45
```

Multilevel-Newton iteration

- This is the data send by first macro element to micro problem associated with the chosen macro element
  - u_M - nodal displacements of macro element

In[214]:=
```
SMTNodeData[SMTElementData[1, "Nodes"], "at"] // Chop
```

```
{{0, 0}, {0, 0}, {0, 0}, {0, 0}}
```

- SMTMicroSolve
  - transfers u_M from macro to micro problem
  - solves all micro problems and returns macro residual and tangent matrix
  - "Dump"→False states that the state of the micro problem stored on disk is not updated!
  - debugging can be turned on here with SMTMacroProblemStatus[[14]]=element_index command

In[215]:=
```
SMTMicroSolve["Dump" → False]
```

```
True
```

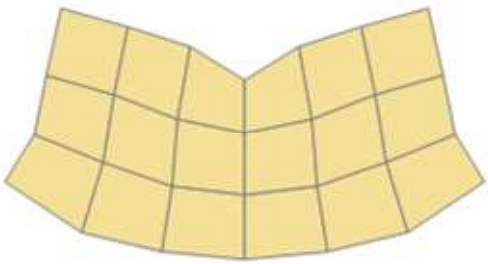
- SMTNewtonIteration - standard Newton iteration at macro level

In[216]:=
```
SMTNewtonIteration[]
```

```
0.482452
```

- Visualization of deformed macro problem after the first Newton iteration.

In[217]:=
```
SMTShowMesh["DeformedMesh" → True]
```



- Some useful tests of a single macro element
  - This is the data set by micro problem associated with the chosen macro element.
  - micro_data = $\int_{\Omega_e}$ Flatten$\left[\left\{W, \frac{\partial W}{\partial u_M}\Big|_{\mathbf{h}=\text{const.}}, \text{vec}\left(\frac{\partial}{\partial u_M}\left(\frac{\partial W}{\partial u_M}\Big|_{\mathbf{h}=\text{const.}}\right)\right)\right\}\right]$ dV

In[218]:=
```
micro = SMTElementData[testelem, "Data"]
```

```
{0., 0., 0., 0., 0., 0., 0., 0., 0., 5111.15, 1671.08, -3524.6, 141.738,
 -2780.38, -1728.28, 1193.84, -84.5386, 5341.15, -40.0743, 1279.89, -1710.04,
 -2910.89, 79.0297, -3710.14, 5262.92, -1896., 1175.33, 157.681, -2913.65,
 1778.4, 5393.42, -67.1605, -3639.16, 1821.42, -3034.15, 4548.71, 1672.21,
 -2943.66, 104.989, 5209.76, -101.608, 1340.29, 4663.48, -1798.85, 5404.}
```

In[219]:=
```
micro // Length
```

```
45
```

  - This is residual vector and tangent matrix as set by micro element. Only symmetric part was stored!

In[220]:=
```
micro[[2 ;; NoDOF + 1]]
```

```
{0., 0., 0., 0., 0., 0., 0., 0.}
```

In[221]:=
```
K = Table[If[j < i, micro[[1 + NoDOF + -j^2/2 + i - NoDOF + j (1/2 + NoDOF)]]
    , micro[[1 + NoDOF + -i^2/2 + j - NoDOF + i (1/2 + NoDOF)]]], {i, NoDOF}, {j, 1, NoDOF}];
K // Chop // MatrixForm
```

$$\begin{pmatrix}
5111.15 & 1671.08 & -3524.6 & 141.738 & -2780.38 & -1728.28 & 1193.84 & -84.5386 \\
1671.08 & 5341.15 & -40.0743 & 1279.89 & -1710.04 & -2910.89 & 79.0297 & -3710.14 \\
-3524.6 & -40.0743 & 5262.92 & -1896. & 1175.33 & 157.681 & -2913.65 & 1778.4 \\
141.738 & 1279.89 & -1896. & 5393.42 & -67.1605 & -3639.16 & 1821.42 & -3034.15 \\
-2780.38 & -1710.04 & 1175.33 & -67.1605 & 4548.71 & 1672.21 & -2943.66 & 104.989 \\
-1728.28 & -2910.89 & 157.681 & -3639.16 & 1672.21 & 5209.76 & -101.608 & 1340.29 \\
1193.84 & 79.0297 & -2913.65 & 1821.42 & -2943.66 & -101.608 & 4663.48 & -1798.85 \\
-84.5386 & -3710.14 & 1778.4 & -3034.15 & 104.989 & 1340.29 & -1798.85 & 5404.
\end{pmatrix}$$

  - This is the tangent matrix of the chosen macro element as set by macro element. It should be the same as the one set by micro element!

In[223]:=
```
SMTData[testelem, "SKR"][[1]] // MatrixForm
```

$$\begin{pmatrix} 5111.15 & 1671.08 & -3524.6 & 141.738 & -2780.38 & -1728.28 & 1193.84 & -84.5386 \\ 1671.08 & 5341.15 & -40.0743 & 1279.89 & -1710.04 & -2910.89 & 79.0297 & -3710.14 \\ -3524.6 & -40.0743 & 5262.92 & -1896. & 1175.33 & 157.681 & -2913.65 & 1778.4 \\ 141.738 & 1279.89 & -1896. & 5393.42 & -67.1605 & -3639.16 & 1821.42 & -3034.15 \\ -2780.38 & -1710.04 & 1175.33 & -67.1605 & 4548.71 & 1672.21 & -2943.66 & 104.989 \\ -1728.28 & -2910.89 & 157.681 & -3639.16 & 1672.21 & 5209.76 & -101.608 & 1340.29 \\ 1193.84 & 79.0297 & -2913.65 & 1821.42 & -2943.66 & -101.608 & 4663.48 & -1798.85 \\ -84.5386 & -3710.14 & 1778.4 & -3034.15 & 104.989 & 1340.29 & -1798.85 & 5404. \end{pmatrix}$$

- Those are eigenvalues of the element tangent matrix of chosen macro element. In the fist iteration the number of zero eigenvalues should be equal to the number of rigid body motions.

In[224]:=
```
SMTData[testelem, "SKR"][[1]] // Eigenvalues // Chop
```

Out[224]=
```
{16504.4, 9011.01, 6646.5, 4732.9, 4039.76, 0, 0, 0}
```

- This is residual vector of the chosen macro element. It should be zero in the first iteration because the macro load has no effect on micro problems in the first Newton iteration.

In[225]:=
```
SMTData[testelem, "SKR"][[2]]
```

Out[225]=
```
{0., 0., 0., 0., 0., 0., 0., 0.}
```

- This is test of the positive definiteness of the global tangent matrix.

In[226]:=
```
PositiveDefiniteMatrixQ[SMTData["TangentMatrix"]]
```

Out[226]=
```
True
```

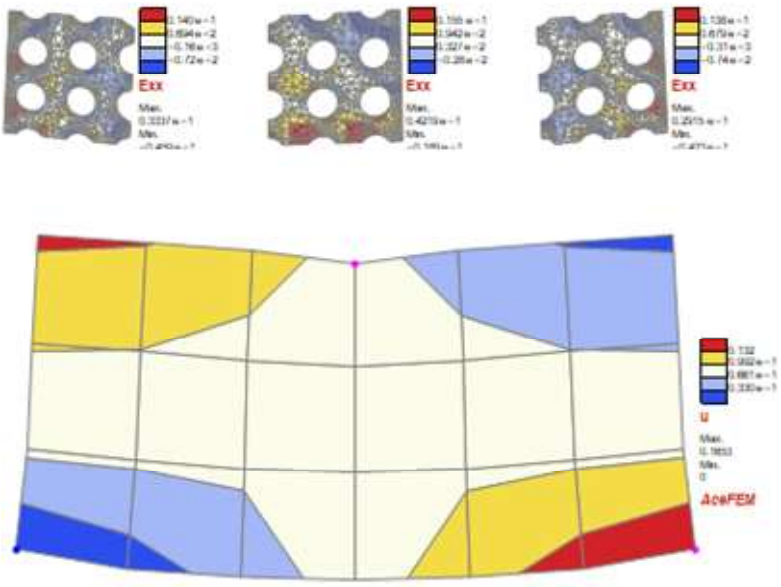
Return the state of the macro problem back to initial state

In[227]:=
```
SMTStepBack[];
```

Evaluate complete response using an adaptive path following procedure

In[228]:=
```
response = {{0, 0}};
λMax = 1; λ0 = λMax / 4; ΔλMin = λMax / 1000; ΔλMax = λMax / 4;
tolNR = 10^-8; maxNR = 15; targetNR = 8;
SMTNextStep["λ" -> λ0];
While[
  While[
   step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λMax}]
   , SMTMicroSolve["Dump" → False];
   (* react on events at RVE *)
   Switch[SMTSharedStatus[[1]]
    , 0 | 1,
    SMTNewtonIteration[];
    SMTStatusReport[SMTSharedStatus[[1]]];
    , 2,
    (* force step back due to the failed micro simulation *)
    SMTIData[{"Iteration", "TotalIteration"}, SMTIData[{"Iteration", "TotalIteration"}] + 1];
    SMTIData["ErrorStatus", 2];
    SMTStatusReport[SMTSharedStatus[[2]]];
    , 3,
    (*abort due to the fatal error *)
    SMTStatusReport[SMTSharedStatus[[2]]]; Abort[];
   ]
  ]; (* end of NR loop *)
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; Abort[];];
  (* successful step *)
  If[Not[step[[1]]],
   (*dump the state of micro problems to disc before the next step*)
   SMTMicroSolve["Dump" → True];
   (* vizualization and post-processing*)
   macromesh = SMTShowMesh["BoundaryConditions" → True,
     "DeformedMesh" → True, "Field" → "u", "Legend" → True, "Contour" → 4];
   micromesh = Table[
     SMTMicroRestart[x];
     SMTMicroEvaluate[
      SMTShowMesh["DeformedMesh" → True, "Field" → "Exx", "Legend" → True, "Contour" → 4]]
     , {x, monitorpoints}];
   Print[Column[{GraphicsRow[micromesh, ImageSize → 600], Show[macromesh, ImageSize → 600]}]];
   AppendTo[response, {SMTRData["Multiplier"] vA, -SMTResidual[Point[{L / 2, H}]][[1, 2]]}];
  ];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" -> step[[2]]]
];
```

```
Step/Iter=1/1 λ/Δλ=0.25/0.25 ‖Δp‖/‖R‖=0.120613/320.593 Events=0 Status=0/{} Tag=0

Step/Iter=1/2 λ/Δλ=0.25/0.25 ‖Δp‖/‖R‖=0.00393845/21.7813 Events=0 Status=0/{} Tag=0

Step/Iter=1/3 λ/Δλ=0.25/0.25 ‖Δp‖/‖R‖=0.0000327391/0.0764147 Events=0 Status=0/{} Tag=0

Step/Iter=1/4 λ/Δλ=0.25/0.25 ‖Δp‖/‖R‖=
 1.20742×10^-9/2.01077×10^-6 Events=0 Status=0/{} Tag=0
```

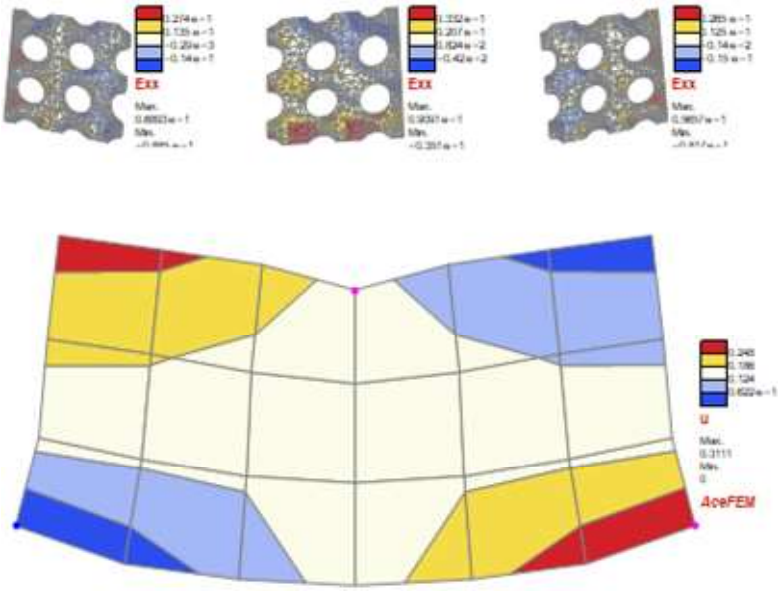


Step/Iter=2/1 λ/Δλ=0.5/0.25 ‖Δp‖/‖R‖=0.118026/330.617 Events=0 Status=0/{} Tag=0

Step/Iter=2/2 λ/Δλ=0.5/0.25 ‖Δp‖/‖R‖=0.0044162/22.8222 Events=0 Status=0/{} Tag=0

Step/Iter=2/3 λ/Δλ=0.5/0.25 ‖Δp‖/‖R‖=0.0000193958/0.0837705 Events=0 Status=0/{} Tag=0

Step/Iter=2/4 λ/Δλ=0.5/0.25 ‖Δp‖/‖R‖=
$1.91054\times10^{-10}/1.52163\times10^{-6}$ Events=0 Status=0/{} Tag=0

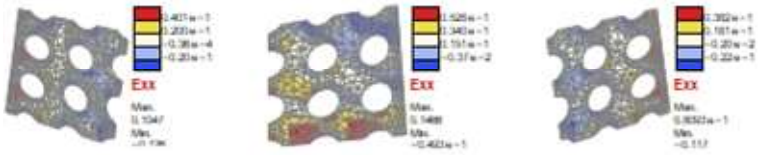


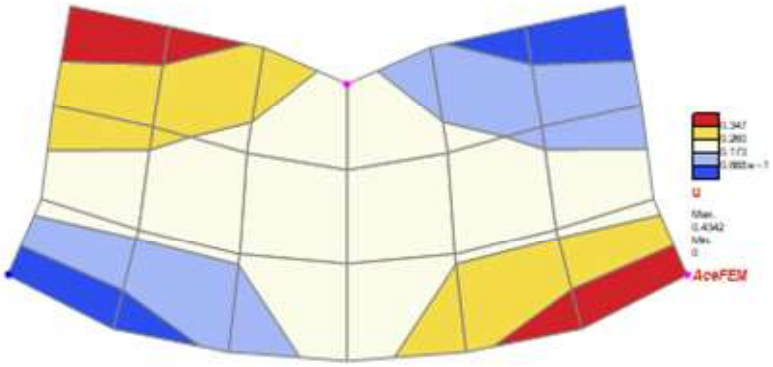
Step/Iter=3/1 λ/Δλ=0.75/0.25 ‖Δp‖/‖R‖=0.114687/336.785 Events=0 Status=0/{} Tag=0

Step/Iter=3/2 λ/Δλ=0.75/0.25 ‖Δp‖/‖R‖=0.0050298/23.1792 Events=0 Status=0/{} Tag=0

Step/Iter=3/3 λ/Δλ=0.75/0.25 ‖Δp‖/‖R‖=0.0000136038/0.0842996 Events=0 Status=0/{} Tag=0

Step/Iter=3/4 λ/Δλ=0.75/0.25 ‖Δp‖/‖R‖=
$1.08461\times10^{-9}/1.30259\times10^{-6}$ Events=0 Status=0/{} Tag=0



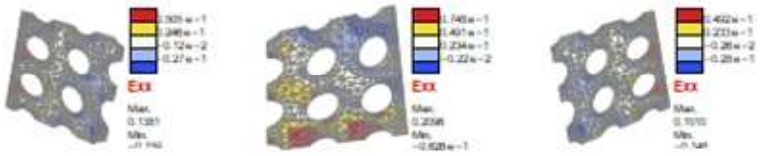


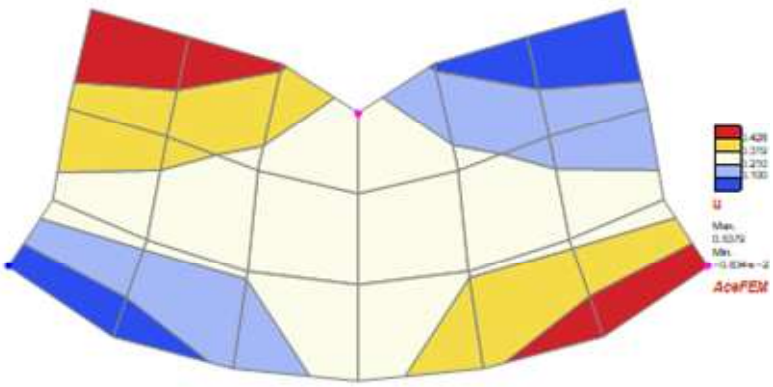
Step/Iter=4/1 λ/Δλ=1./0.25 ‖Δp‖/‖R‖=0.11069/337.874 Events=0 Status=0/{} Tag=0

Step/Iter=4/2 λ/Δλ=1./0.25 ‖Δp‖/‖R‖=0.00597158/23.1736 Events=0 Status=0/{} Tag=0

Step/Iter=4/3 λ/Δλ=1./0.25 ‖Δp‖/‖R‖=0.0000341847/0.0828817 Events=0 Status=0/{} Tag=0

Step/Iter=4/4 λ/Δλ=1./0.25 ‖Δp‖/‖R‖=
$2.57282\times10^{-9}/6.50608\times10^{-6}$ Events=0 Status=0/{} Tag=0





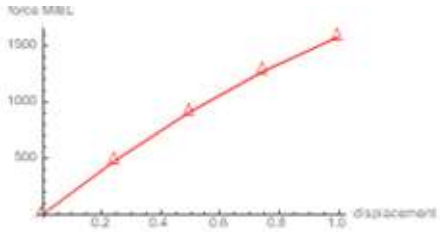
In[233]:=
```
SMTMultiScaleSimulationReport[];
```

MACRO	
Number of macro elements	18
Number of macro equations	52
No. of steps	4
No. of steps back	1
Total no. of iterations	17
Total driver time (s)	35.13
Total linear solver time (s)	0.01
Total K&R time (s)	0.00
MICRO	
Number of FE^2 elements	0
Number of MIEL elements	18
Number of single-scale elements	0
Parallelization kernels/threads	16/1
Total number of micro problems	18
Total number of micro elements	12 526
Total number of micro nodes	8230
Total number of micro equations	16 460
Average number of micro equations	914.444
Real micro mesh generation time	2.19
Number of MicroSolve calls	21
Total MicroSolve time	10.38
Total number of MP equations solved	345 660
Total number of steps	450
Total number of back steps	0
Average number of micro problems/kernel	1.1
Total number of MP solved	378
Average MP time	0.03
Average MP K&R time	0.00
Average MP linear solver time	0.01
Average MP sens.p.load time	0.00
Average MP sens.l.solver time	0.00
Average MP Schur complement time	0
Average MP task time	0.00
Average MP Driver time	0.94
Average real Driver time	20.96
Total parallel MP time	13.01
Number of restarts	378
Total parallel restart time	51.38
Average restart time	0.14
Number of dumps	90
Total parallel dump time	5.25
Average dump time	0.06
Total dump files size (byte)	2 807 886
zlib compression level	2
Macro + micro analysis time	12.58
Aministrative time	9.60
Lost time:	9.56
Total time	22.18

Analysis of the results

Response curve

In[234]:=
```
ListLinePlot[response, AxesLabel → {"displacement", "force MIEL"},
 PlotMarkers → "△", PlotStyle → Red]
```



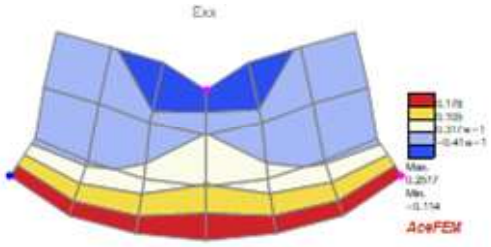
Calculate stiffness (KN/cm)

In[235]:=
```
response[[2, 2]] / response[[2, 1]]
```

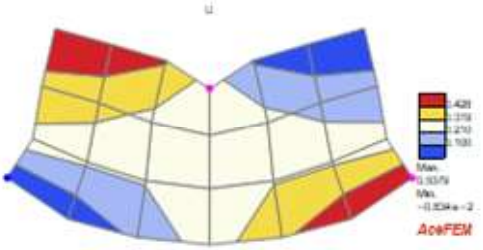
```
1957.06
```

Project micro solution to macro problem

In[236]:=
```
SMTShowMesh["BoundaryConditions" → True, "DeformedMesh" → True,
 "Field" → SMTMultiScalePostData["Exx"] // Chop, "Legend" → True, "Contour" → 4, "Label" → "Exx"]
```



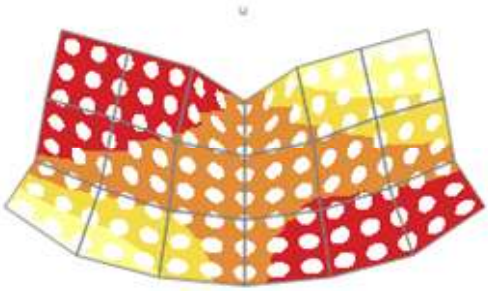
In[237]:=
```
SMTShowMesh["BoundaryConditions" → True, "DeformedMesh" → True,
 "Field" → SMTMultiScalePostData["u"], "Legend" → True, "Contour" → 4, "Label" → "u"]
```



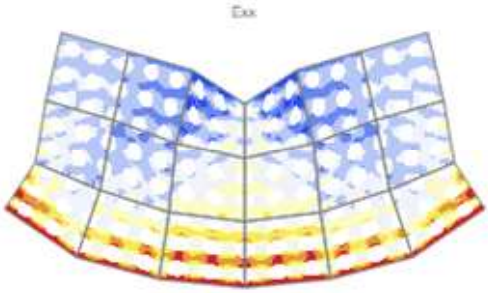
Visualize true micro solution

In[238]:=
```
Show[Map[(
    SMTMicroRestart[#[[2]]];
    SMTMicroEvaluate[SMTShowMesh["DeformedMesh" → True, "Mesh" → True]]
  ) &, SMTAllLocalProblems]
 , SMTShowMesh["FillElements" → False, "DeformedMesh" → True]]
```

In[239]:=
```
Show[Map[(
    SMTMicroRestart[#[[2]]];
    SMTMicroEvaluate[SMTShowMesh["DeformedMesh" → True,
      "Field" -> "u", "Contour" → {-0.3, 0.3, 6}, "Legend" → False, "Mesh" → False]]
  ) &, SMTAllLocalProblems]
 , SMTShowMesh["FillElements" → False, "DeformedMesh" → True], PlotLabel → "u"]
```



In[240]:=
```
Show[Map[(
    SMTMicroRestart[#[[2]]];
    SMTMicroEvaluate[SMTShowMesh["DeformedMesh" → True,
      "Field" -> "Exx", "Contour" → {-0.1, 0.2, 5}, "Legend" → False, "Mesh" → False]]
  ) &, SMTAllLocalProblems]
 , SMTShowMesh["FillElements" → False, "DeformedMesh" → True], PlotLabel → "Exx"]
```



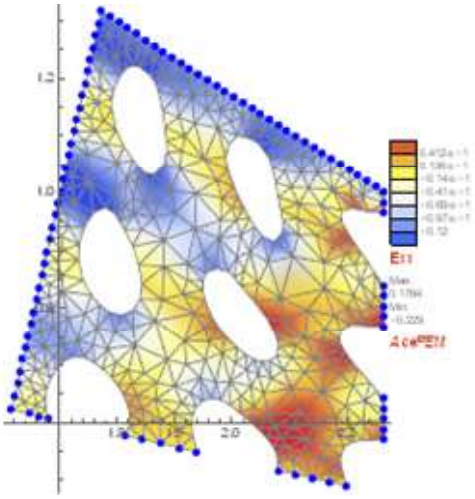
Visualization of the chosen micro problem

- SMTMicroRestart[{x, y}] ... restart micro problem that is nearest to the given spatial position (position is given in initial coordinates) and set the selected micro problem as the current micro problem

In[241]:=
```
SMTMicroRestart[{1.75, 1.75}];
```

- SMTMicroEvaluate[exp] ... evaluate exp for the current micro problem

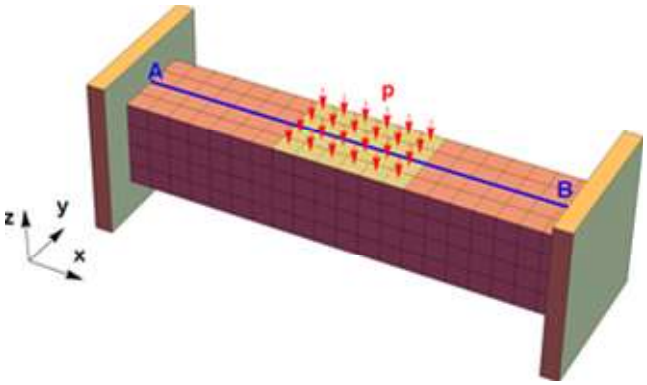
In[242]:=
```
SMTMicroEvaluate[SMTShowMesh[Axes → True, "DeformedMesh" → True,
  "BoundaryConditions" → True, "Field" -> "Exx", "Legend" → True]]
```

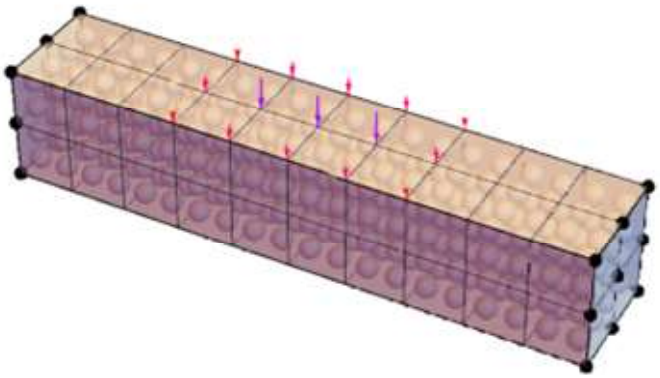


MIEL modeling of 3D uniformly distributed micro-structure

Description

Perform simulation of the beam like structure made of perforated material. The real size of perforations is comparable with dimensions of the problem, thus the FE^2 approach can not give accurate structural response. The MIEL multi scale approach is used instead.





Macro problem

In[38]:=
```
<< AceFEM`;
```

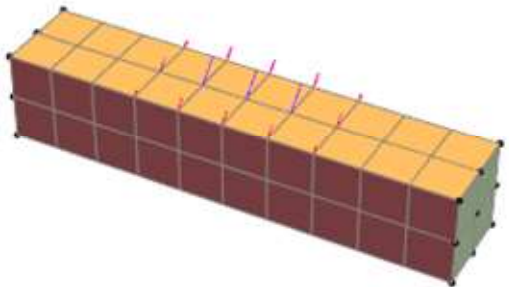
In[39]:=
```
If[SMTMultiScaleLoaded =!= True,
  MessageDialog["Load Multi-scale computational environment first!"];
  Quit[]];
```

In[40]:=
```
pathDependent = True;
topology = "H1";
macroElement = "Examples" <> topology <> "MIELMacro" <> If[pathDependent, "Unsymm", "Symm"];
```

In[43]:=
```
L = 12.; B = 2.4; H = 2.4;
n = 2; nx = 5 * n; ny = 1 * n; nz = 1 * n;
p = 50;
SMTInputData["Threads" → 1];
SMTAddDomain[{"beam", macroElement, {}}];
points = {{0, 0, 0}, {L, 0, 0}, {L, B, 0}, {0, B, 0}, {0, 0, H}, {L, 0, H}, {L, B, H}, {0, B, H}};
SMTAddMesh[Hexahedron[points], "beam", "H1", {nx, ny, nz}];
SMTAddEssentialBoundary[
  Polygon[{{0, 0, 0}, {0, 0, H}, {0, B, H}, {0, B, 0}}], 1 → 0, 2 → 0, 3 → 0];
SMTAddEssentialBoundary[Polygon[{{L, 0, 0}, {L, 0, H}, {L, B, H}, {L, B, 0}}], 1 → 0, 2 → 0, 3 → 0];
SMTAddNaturalBoundary[
  Polygon[{{L / 10 * 3, 0, H}, {L / 10 * 7, 0, H}, {L / 10 * 7, B, H}, {L / 10 * 3, B, H}}],
  2 → Polygon[{-p}], 3 → Polygon[{-p}]];
SMTAnalysis[];
MIELElements = SMTFindElements["beam"];
Length[MIELElements]
```

```
40
```

In[56]:=
```
SMTShowMesh["BoundaryConditions" → True]
```



Micro problem

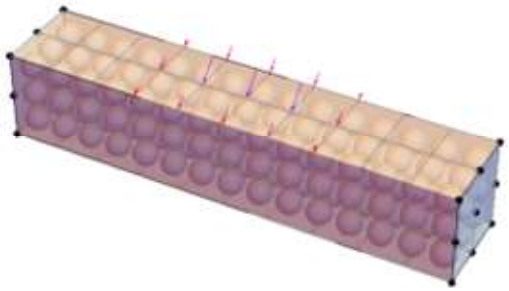
In[57]:=
```
(* number of spheres in each direction *)
nxC = 15; nyC = 3; nzC = 3;
aCell = L / nxC; bCell = B / nyC; cCell = H / nzC;
pVoids = 0.3; (* percentage of voids*)
rCell = CubeRoot[(pVoids aCell bCell cCell 3) / (π 4)]; (* radius of spheres*)
spheres = If[pVoids === 0, {},
   Flatten[Table[{{aCell / 2 + aCell i, bCell / 2 + bCell j, cCell / 2 + cCell k}, rCell},
      {i, 0, nxC - 1}, {j, 0, nyC - 1}, {k, 0, nzC - 1}], 2] // N];
meshDensity = 20; (*number of micro elements per side of macro eleemnt *)
```

- specific_micro_data={
  - 1 {{{x1,y1,z1},r1},{{x2,y2,z2},r2},…} - all spheres where {xi,yi,zi} is a center of the i-th voids and ri is the radius of the i-th void
  - 2 division - approximate number of elements per side
  - 3 micro_element - SMTMakeDll[micro_element_UEC] - element used to discretize the solid domain
  - 4 micro_material - material data related to micro element (see `SMTAddDomain`)
  - 5 topology - micro mesh type (see `Element Topology`)
  - 6 list of solution procedure options - given options are stored in *local_problem_data[[10]]* and interpreted later by the MIELSolveOne function. Current options are:

- "MinSubSteps"->*n* - minimum number of micro sub-steps per one macro step (default *n*=1)
- "MaxSubSteps"->*n* ⇒ defines a maximum number of micro sub-steps per macro step (default 100)
- }

In[63]:=
```
Clear[specificMacroData]
specificMacroData = {spheres, meshDensity, SMTMakeDll["ExamplesSED3O1DFHYO1DNeoHookeWA"],
   {"E *" → 2000, "ν *" → 0.3}, "O1", {"MinSubSteps" → 1, "MaxSubSteps" → 10}};
```

In[65]:=
```
Show[SMTShowMesh["FillElements" → False, "BoundaryConditions" → True],
 Graphics3D[{Opacity[0.6], Cuboid[{0, 0, 0}, {L, B, H}],
   Map[Sphere[#[[1]], #[[2]]] &, spheres]}, Boxed → False]]
```



Set up multi-scale environment

- The "PartialRestart" option can be True only when the local problems are structurally the same mesh (only the actual coordinates of the nodes can be different).
- The "PartialDump" must be False only when during the solution of the local problem the mesh has been changed as well.

In[66]:=
```
SMTMultiScaleSet[
 "MIEL" → {{MIEL3DMicroMeshVoids, MIELSolveOne} -> MIELElements},
 "SpecificMicroData" → {"MIEL3DMicroMeshVoids" -> specificMacroData},
"MaxKernels" → Automatic, "Threads" → 1,
 "Console" → False, "PartialDump" → True, "PartialRestart" → False]
```

Out[66]=
```
True
```

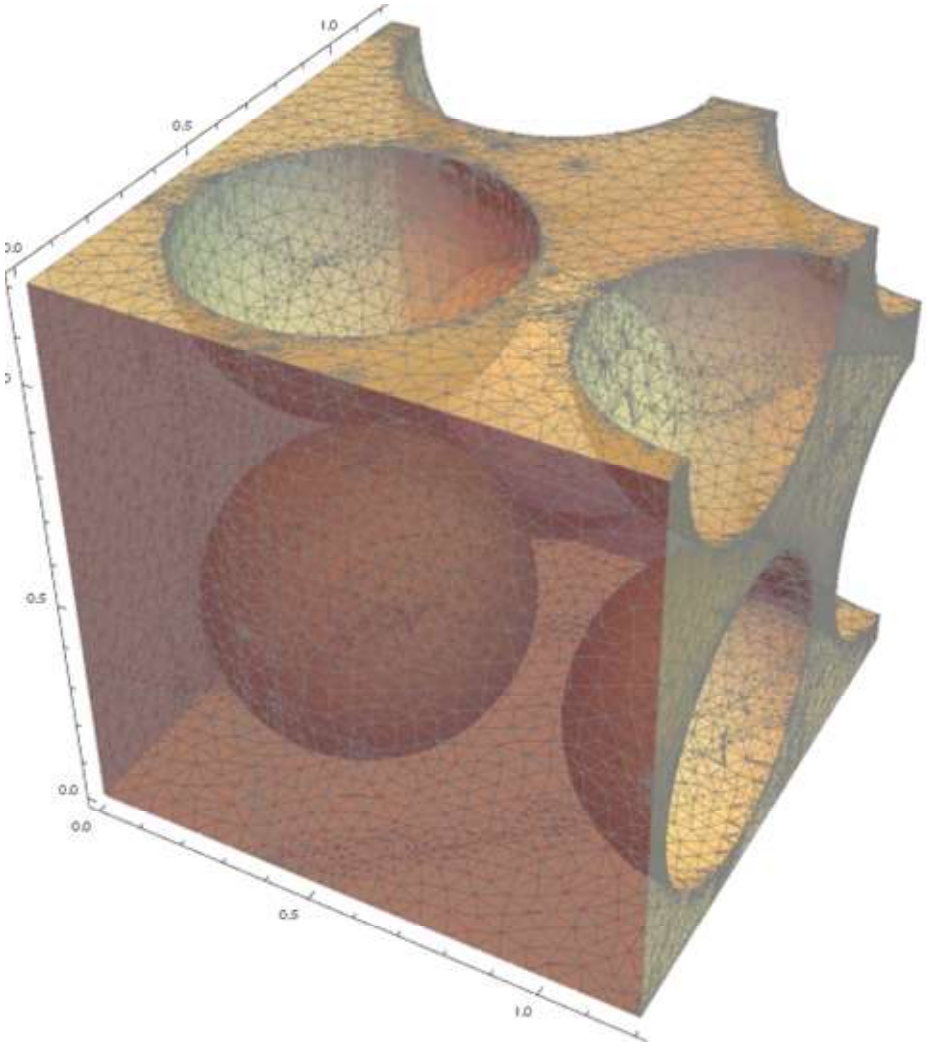
Visualize selected micro problem

- SMTMicroRestart[{x, y,z}] ... restart micro problem that is nearest to the given spatial position (position is given in initial coordinates) and set the selected micro problem as the current micro problem

In[67]:=
```
SMTMicroRestart[{0, 0, 0}];
```

- SMTMicroEvaluate[exp] ... evaluates exp for the restarted micro problem

In[68]:=
```
SMTMicroEvaluate[SMTShowMesh[Axes → True, "Opacity" → 0.6, "ZoomElements" → "micro"]]
```



Evaluate complete response using an adaptive path following procedure

In[70]:=
```
SMTAnimationOfResponse["Initialize", {0, 0}];
λMax = 1; λ0 = λMax / 10; ΔλMin = λMax / 1000; ΔλMax = λMax / 10;
tolNR = 10^-8; maxNR = 15; targetNR = 8;
SMTNextStep["λ" -> λ0];
While[
  While[
   step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λMax}]
   , SMTMicroSolve["Dump" → False];
   (* react on events at RVE *)
   Switch[SMTSharedStatus[[1]]
    , 0 | 1,
    SMTNewtonIteration[];
    SMTStatusReport[SMTSharedStatus[[1]]];
    , 2,
    (* force step back due to the failed micro simulation *)
    SMTIData[{"Iteration", "TotalIteration"}, SMTIData[{"Iteration", "TotalIteration"}] + 1];
    SMTIData["ErrorStatus", 2];
    SMTStatusReport[SMTSharedStatus[[2]]];
    , 3,
    (*abort due to the fatal error *)
    SMTStatusReport[SMTSharedStatus[[2]]]; Abort[];
   ]
  ]; (* end of NR loop *)
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; Abort[];];
  (* successful step *)
  If[Not[step[[1]]],
   (*dump the state of micro problems to disc before the next step*)
   SMTMicroSolve["Dump" → True];
   SMTAnimationOfResponse[
    "LeadingNodePosition" → {L / 2, 0, 0}, "x" → Hold[-SMTPostData["v", Point[{L / 2, 0, 0}]]]
    , "Show" → "Window" | {"ExportFrames", "responseFile"}
   ]
  ];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" -> step[[2]]]
];
```

```
Step/Iter=1/1 λ/Δλ=0.1/0.1 ‖Δp‖/‖R‖=0.0993261/1.49666 Events=0 Status=0/{} Tag=0

Step/Iter=1/2 λ/Δλ=0.1/0.1 ‖Δp‖/‖R‖=0.00246198/2.30975 Events=0 Status=0/{} Tag=0

Step/Iter=1/3 λ/Δλ=0.1/0.1 ‖Δp‖/‖R‖=0.000496976/0.00697385 Events=0 Status=0/{} Tag=0

Step/Iter=1/4 λ/Δλ=0.1/0.1 ‖Δp‖/‖R‖=2.22056×10^-7/0.000303734 Events=0 Status=0/{} Tag=0

Step/Iter=1/5 λ/Δλ=0.1/0.1 ‖Δp‖/‖R‖=
 4.73462×10^-12/6.45855×10^-11 Events=0 Status=0/{} Tag=0

Step/Iter=2/1 λ/Δλ=0.2/0.1 ‖Δp‖/‖R‖=0.101238/1.49666 Events=0 Status=0/{} Tag=0

Step/Iter=2/2 λ/Δλ=0.2/0.1 ‖Δp‖/‖R‖=0.00272906/2.77092 Events=0 Status=0/{} Tag=0

Step/Iter=2/3 λ/Δλ=0.2/0.1 ‖Δp‖/‖R‖=0.000615398/0.00792087 Events=0 Status=0/{} Tag=0

Step/Iter=2/4 λ/Δλ=0.2/0.1 ‖Δp‖/‖R‖=2.81678×10^-7/0.000469695 Events=0 Status=0/{} Tag=0

Step/Iter=2/5 λ/Δλ=0.2/0.1 ‖Δp‖/‖R‖=
 3.19546×10^-12/1.03794×10^-10 Events=0 Status=0/{} Tag=0

Step/Iter=3/1 λ/Δλ=0.3/0.1 ‖Δp‖/‖R‖=0.101589/1.49666 Events=0 Status=0/{} Tag=0
```

Step/Iter=3/2 λ/Δλ=0.3/0.1 ‖Δp‖/‖R‖=0.00313422/3.27425 Events=0 Status=0/{} Tag=0

Step/Iter=3/3 λ/Δλ=0.3/0.1 ‖Δp‖/‖R‖=0.000680902/0.00868214 Events=0 Status=0/{} Tag=0

Step/Iter=3/4 λ/Δλ=0.3/0.1 ‖Δp‖/‖R‖=3.7325×10^{-7}/0.000577059 Events=0 Status=0/{} Tag=0

Step/Iter=3/5 λ/Δλ=0.3/0.1 ‖Δp‖/‖R‖=
1.16941×10^{-11}/1.85803×10^{-10} Events=0 Status=0/{} Tag=0

Step/Iter=4/1 λ/Δλ=0.4/0.1 ‖Δp‖/‖R‖=0.100655/1.49666 Events=0 Status=0/{} Tag=0

Step/Iter=4/2 λ/Δλ=0.4/0.1 ‖Δp‖/‖R‖=0.00344403/3.77651 Events=0 Status=0/{} Tag=0

Step/Iter=4/3 λ/Δλ=0.4/0.1 ‖Δp‖/‖R‖=0.000635185/0.0092547 Events=0 Status=0/{} Tag=0

Step/Iter=4/4 λ/Δλ=0.4/0.1 ‖Δp‖/‖R‖=4.99599×10^{-7}/0.00050141 Events=0 Status=0/{} Tag=0

Step/Iter=4/5 λ/Δλ=0.4/0.1 ‖Δp‖/‖R‖=
2.71673×10^{-11}/3.36933×10^{-10} Events=0 Status=0/{} Tag=0

Step/Iter=5/1 λ/Δλ=0.5/0.1 ‖Δp‖/‖R‖=0.0987149/1.49666 Events=0 Status=0/{} Tag=0

Step/Iter=5/2 λ/Δλ=0.5/0.1 ‖Δp‖/‖R‖=0.00358751/4.1974 Events=0 Status=0/{} Tag=0

Step/Iter=5/3 λ/Δλ=0.5/0.1 ‖Δp‖/‖R‖=0.000447872/0.00971167 Events=0 Status=0/{} Tag=0

Step/Iter=5/4 λ/Δλ=0.5/0.1 ‖Δp‖/‖R‖=4.54158×10^{-7}/0.00024455 Events=0 Status=0/{} Tag=0

Step/Iter=5/5 λ/Δλ=0.5/0.1 ‖Δp‖/‖R‖=
1.46606×10^{-11}/3.16302×10^{-10} Events=0 Status=0/{} Tag=0

Step/Iter=6/1 λ/Δλ=0.6/0.1 ‖Δp‖/‖R‖=0.0958864/1.49666 Events=0 Status=0/{} Tag=0

Step/Iter=6/2 λ/Δλ=0.6/0.1 ‖Δp‖/‖R‖=0.00359327/4.44105 Events=0 Status=0/{} Tag=0

Step/Iter=6/3 λ/Δλ=0.6/0.1 ‖Δp‖/‖R‖=0.000169359/0.0100188 Events=0 Status=0/{} Tag=0

Step/Iter=6/4 λ/Δλ=0.6/0.1 ‖Δp‖/‖R‖=
1.47882×10^{-7}/0.0000264961 Events=0 Status=0/{} Tag=0

Step/Iter=6/5 λ/Δλ=0.6/0.1 ‖Δp‖/‖R‖=
4.67231×10^{-12}/7.84998×10^{-10} Events=0 Status=0/{} Tag=0

Step/Iter=7/1 λ/Δλ=0.7/0.1 ‖Δp‖/‖R‖=0.0922028/1.49666 Events=0 Status=0/{} Tag=0

Step/Iter=7/2 λ/Δλ=0.7/0.1 ‖Δp‖/‖R‖=0.00350216/4.44499 Events=0 Status=0/{} Tag=0

Step/Iter=7/3 λ/Δλ=0.7/0.1 ‖Δp‖/‖R‖=0.000182926/0.0100149 Events=0 Status=0/{} Tag=0

Step/Iter=7/4 λ/Δλ=0.7/0.1 ‖Δp‖/‖R‖=
1.50405×10^{-7}/0.0000339551 Events=0 Status=0/{} Tag=0

Step/Iter=7/5 λ/Δλ=0.7/0.1 ‖Δp‖/‖R‖=
2.05049×10^{-12}/1.68401×10^{-10} Events=0 Status=0/{} Tag=0

Step/Iter=8/1 λ/Δλ=0.8/0.1 ‖Δp‖/‖R‖=0.0877852/1.49666 Events=0 Status=0/{} Tag=0

Step/Iter=8/2 λ/Δλ=0.8/0.1 ‖Δp‖/‖R‖=0.00334262/4.21691 Events=0 Status=0/{} Tag=0

Step/Iter=8/3 λ/Δλ=0.8/0.1 ‖Δp‖/‖R‖=0.000381921/0.00960055 Events=0 Status=0/{} Tag=0

Step/Iter=8/4 λ/Δλ=0.8/0.1 ‖Δp‖/‖R‖=2.92705×10^{-7}/0.000192458 Events=0 Status=0/{} Tag=0

Step/Iter=8/5 λ/Δλ=0.8/0.1 ‖Δp‖/‖R‖=
2.78842×10^{-9}/5.32975×10^{-7} Events=0 Status=0/{} Tag=0

Step/Iter=9/1 λ/Δλ=0.9/0.1 ‖Δp‖/‖R‖=0.082897/1.49666 Events=0 Status=0/{} Tag=0

Step/Iter=9/2 λ/Δλ=0.9/0.1 ‖Δp‖/‖R‖=0.00313674/3.8249 Events=0 Status=0/{} Tag=0

Step/Iter=9/3 λ/Δλ=0.9/0.1 ‖Δp‖/‖R‖=0.000470656/0.00883516 Events=0 Status=0/{} Tag=0

Step/Iter=9/4 λ/Δλ=0.9/0.1 ‖Δp‖/‖R‖=1.83029×10^{-7}/0.000279798 Events=0 Status=0/{} Tag=0

Step/Iter=9/5 λ/Δλ=0.9/0.1 ‖Δp‖/‖R‖=5.22745×10^{-11}/1.0446×10^{-8} Events=0 Status=0/{} Tag=0

Step/Iter=10/1 λ/△λ=1./0.1 ‖△p‖/‖R‖=0.0778608/1.49666 Events=0 Status=0/{} Tag=0

Step/Iter=10/2 λ/△λ=1./0.1 ‖△p‖/‖R‖=0.00290308/3.35588 Events=0 Status=0/{} Tag=0

Step/Iter=10/3 λ/△λ=1./0.1 ‖△p‖/‖R‖=0.00046326/0.00787438 Events=0 Status=0/{} Tag=0

Step/Iter=10/4 λ/△λ=1./0.1 ‖△p‖/‖R‖=2.10385×10^{-7}/0.0002743 Events=0 Status=0/{} Tag=0

Step/Iter=10/5 λ/△λ=1./0.1 ‖△p‖/‖R‖=
2.67137×10^{-10}/5.03164×10^{-8} Events=0 Status=0/{} Tag=0

In[75]:= **SMTMultiScaleSimulationReport[];**

MACRO	
Number of macro elements	40
Number of macro equations	243
No. of steps	10
No. of steps back	0
Total no. of iterations	50
Total driver time (s)	2:8:47.95
Total linear solver time (s)	0.18
Total K&R time (s)	0.06
MICRO	
Number of FE^2 elements	0
Number of MIEL elements	40
Number of single-scale elements	0
Parallelization kernels/threads	16/1
Total number of micro problems	40
Total number of micro elements	5 611 082
Total number of micro nodes	1 475 235
Total number of micro equations	4 425 705
Average number of micro equations	110 643.
Real micro mesh generation time	1:31.89
Number of MicroSolve calls	60
Total MicroSolve time	2:6:48.43
Total number of MP equations solved	265 542 300
Total number of steps	2805
Total number of back steps	5
Average number of micro problems/kernel	2.5
Total number of MP solved	2400
Average MP time	35.82
Average MP K&R time	2.48
Average MP linear solver time	2.26
Average MP sens.p.load time	20.41
Average MP sens.l.solver time	3.49
Average MP Schur complement time	0
Average MP task time	4.22
Average MP Driver time	52.33
Average real Driver time	2:7:46.12
Total parallel MP time	23:52:37.73
Number of restarts	2400
Total parallel restart time	2:23:34.83
Average restart time	3.59
Number of dumps	440
Total parallel dump time	2:6:52.71
Average dump time	17.30
Total dump files size (byte)	832 934 586
zlib compression level	2
Macro + micro analysis time	2:8:20.55
Aministrative time	14.00
Lost time:	37:16.07
Total time	2:8:34.55

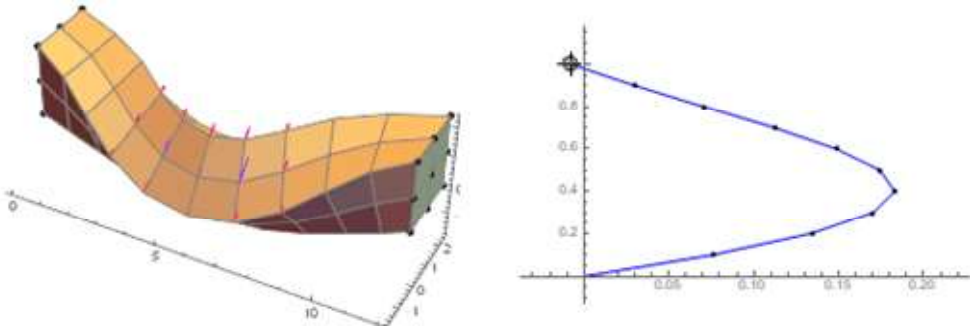
In[76]:=
```
SMTSimulationReport[];
```

No. of nodes	99	Direct analysis:	{}
No. of elements	40	Total linear solver time (s)	0.176
No. of equations	243	Total linear solver time (%)	0.0022773
Number of threads used/max	1/32	Total K&R time (s)	0.055
Data memory (KBytes)	229	Total K&R time (%)	0.00071167
Tangent matrix (KBytes)	88	Average time/iteration (s)	154.56
Solver memory (KBytes)	741	Average linear solver time (s)	0.00352
Total driver (KBytes)	1058	General timing:	
MMA kernel memory (KBytes)	178 205	Input data phase (s)	0.015633
MMA front end memory (KBytes)	269 132	Input data phase (%)	0.00020228
Total memory (KBytes)	448 395	Total visualization time (s)	4.0622
Solver type	Pardiso	Total visualization time (%)	0.052563
Matrix type	1	Total absolute time (s)	7728.3
No. of steps	10	Total driver time (s)	7728.2
No. of steps back	0	CPU Mathematica time (s)	27.407
Step efficiency (%)	100.	CPU Mathematica time (%)	0.35463
Total no. of iterations	50	Profile time	0.053
Average iterations/step	5.		
Terminal BC multiplier (λ)	1.		
Terminal time (t)	0.		
Terminal parameter (γ)	0.		

Analysis of the results

Response curve

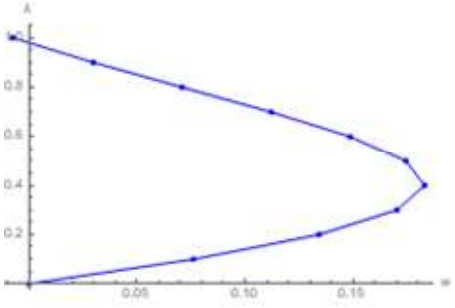
In[77]:=
```
SMTAnimationOfResponse["Report"]
```



In[78]:=
```
response = SMTAnimationOfResponse["Response"]
```

```
{{{0, 0}, {0.0760884, 0.1}, {0.134153, 0.2},
  {0.17025, 0.3}, {0.183024, 0.4}, {0.174277, 0.5}, {0.148589, 0.6},
  {0.112072, 0.7}, {0.070837, 0.8}, {0.0298449, 0.9}, {-0.00752171, 1.}}}
```

In[79]:=
```
ListLinePlot[response, AxesLabel → {"w", "λ"}, PlotMarkers → Automatic, PlotStyle → Blue]
```



Calculate stiffness (KN/cm)

In[80]:=
```
response[[1, 2, 2]] / response[[1, 2, 1]]
```

```
1.31426
```

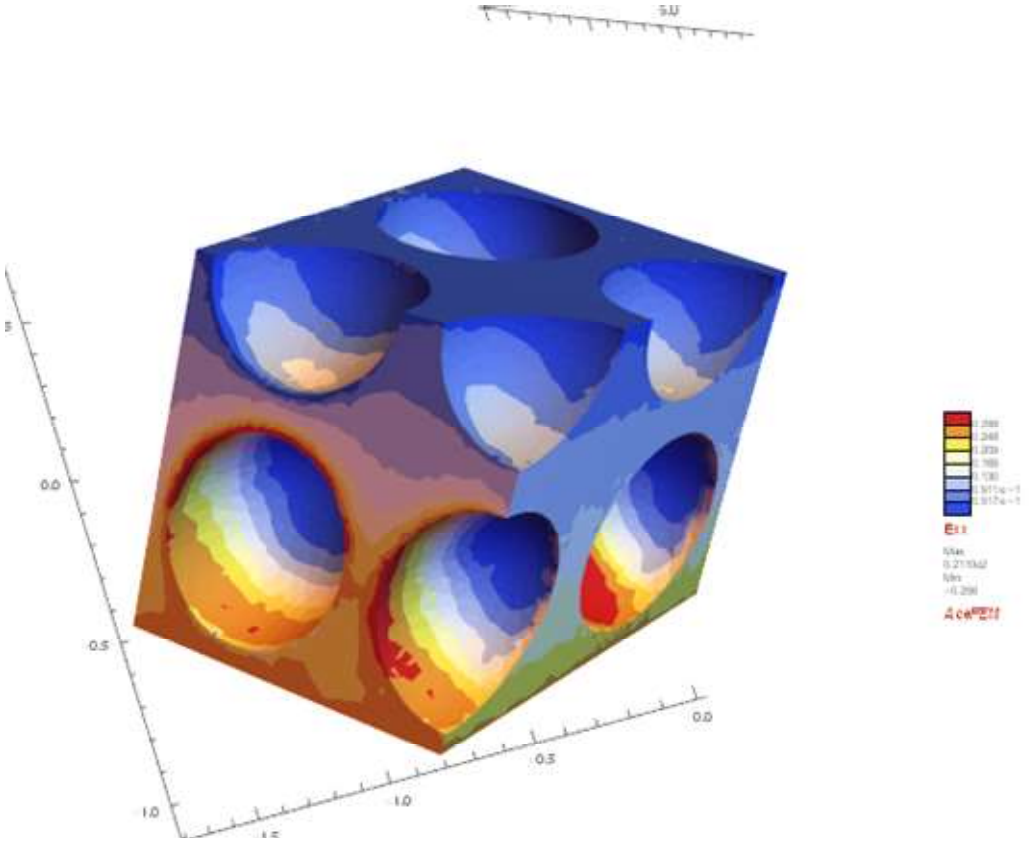
Visualization of the chosen micro problem

- SMTMicroRestart[{x, y}] ... restart micro problem that is nearest to the given spatial position (position is given in initial coordinates) and set the selected micro problem as the current micro problem

In[284]:=
```
SMTMicroRestart[{L / 2, 0, H}];
```

- SMTMicroEvaluate[exp] ... evaluate exp for the current micro problem

In[82]:=
```
SMTMicroEvaluate[SMTShowMesh[Axes → True, "DeformedMesh" → True,
  "BoundaryConditions" → False, "Field" -> "Exx", "Mesh" → False, "Contour" -> True]]
```

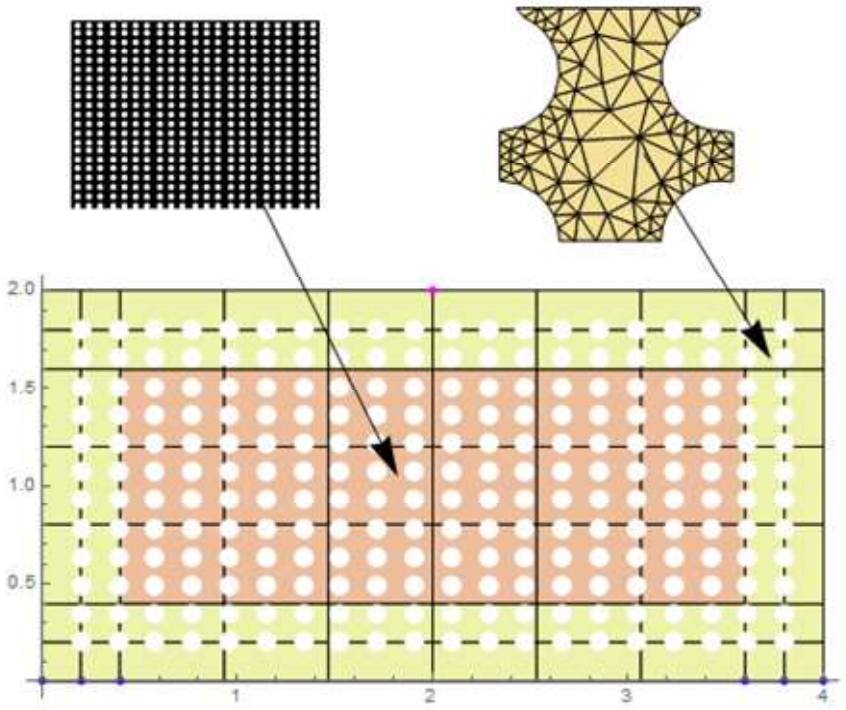


Mixed single-scale/FE^2/MIEL Modeling

Mixed FE^2/MIEL modeling of uniformly distributed micro-structure

Description

Perform simulation of the beam like structure made of perforated material. The real size of perforations is comparable with dimensions of the problem, thus the FE^2 approach can not give accurate structural response ner the boundary. The MIEL multi scale approach is used instead near the boundary.



In[83]:=
```
L = 4.; (*length cm/KN*)
H = 2.; (*height*)
vA = 1;(* prescribed displacement at L/2*)
pVoids = 0.3;(* percentage of perforation*)
Bsolid = .2(* solid boundary zone needed at the points of concentrated load*);
BMIEL = 0.4(* boundary layer where MIEL is applied *);
```

In[89]:=
```
nx = 20; ny = 12;(* number of voids in each direction *)
aCell = (L - 2 Bsolid) / (nx - 1); (* cell length *)
bCell = (H - 2 Bsolid) / (ny - 1); (* cell height *)
rVoid = Sqrt[pVoids aCell bCell /π];(* radius of perforation*)
```

Macro problem

In[93]:=
```
If[SMTMultiScaleLoaded =!= True,
  MessageDialog["Load Multi-scale computational environment first!"];
  Quit[]];
```

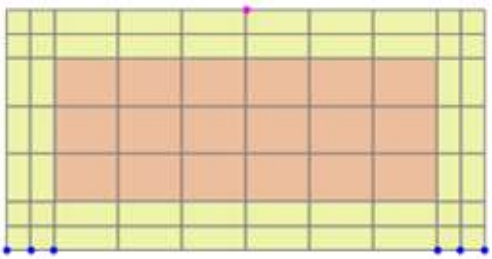
In[94]:=
```
Nx = 6; (*mesh density in x direction*)
Ny = 3; (*mesh density in y direction*)
Nb = 2;(*mesh density in boundary layer*)
SMTInputData["Threads" → 1];
SMTAddDomain[{"beam", "ExamplesQ1PFMacroSymm", {}}];
SMTAddDomain[{"layer", "ExamplesQ1MIELMacroSymm", {}}];
SMTAddEssentialBoundary[
  {Line[{{0, 0}, {L / 5, 0}}], 1 -> 0, 2 -> 0}
  , {Line[{{4 / 5 L, 0}, {L, 0}}], 1 -> 0, 2 → 0}
  , { Point[{L / 2, H}] , 2 → -vA}];
```

In[101]:=
```
SMTAddMesh[
  Polygon[{{BMIEL, BMIEL}, {L - BMIEL, BMIEL}, {L - BMIEL, H - BMIEL}, {BMIEL, H - BMIEL}}],
  "beam", "Q1", {Nx, Ny}];
SMTAddMesh[Polygon[{{0, 0}, {BMIEL, 0}, {BMIEL, BMIEL}, {0, BMIEL}}], "layer", "Q1", {Nb, Nb}];
SMTAddMesh[Polygon[{{BMIEL, 0}, {-BMIEL + L, 0}, {-BMIEL + L, BMIEL}, {BMIEL, BMIEL}}],
  "layer", "Q1", {Nx, Nb}];
SMTAddMesh[Polygon[{{-BMIEL + L, 0}, {L, 0}, {L, BMIEL}, {-BMIEL + L, BMIEL}}],
  "layer", "Q1", {Nb, Nb}];
SMTAddMesh[Polygon[{{-BMIEL + L, BMIEL}, {L, BMIEL}, {L, -BMIEL + H}, {-BMIEL + L, -BMIEL + H}}],
  "layer", "Q1", {Nb, Ny}];
SMTAddMesh[Polygon[{{-BMIEL + L, -BMIEL + H}, {L, -BMIEL + H}, {L, H}, {-BMIEL + L, H}}],
  "layer", "Q1", {Nb, Nb}];
SMTAddMesh[Polygon[{{BMIEL, -BMIEL + H}, {-BMIEL + L, -BMIEL + H}, {-BMIEL + L, H}, {BMIEL, H}}],
  "layer", "Q1", {Nx, Nb}];
SMTAddMesh[Polygon[{{0, -BMIEL + H}, {BMIEL, -BMIEL + H}, {BMIEL, H}, {0, H}}],
  "layer", "Q1", {Nb, Nb}];
SMTAddMesh[Polygon[{{0, BMIEL}, {BMIEL, BMIEL}, {BMIEL, -BMIEL + H}, {0, -BMIEL + H}}],
  "layer", "Q1", {Nb, Ny}];
```

In[110]:=
```
SMTAnalysis[];
MIELElements = SMTFindElements["layer"];
Length[MIELElements];
FE2Elements = SMTFindElements["beam"];
Length[FE2Elements];
```

- Macro mesh and selected points for later post-processing

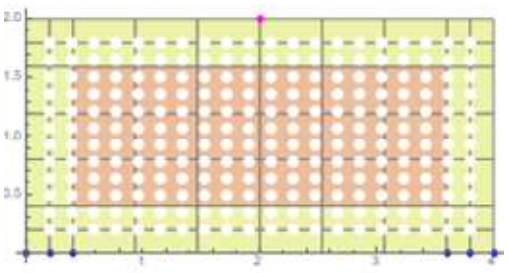
In[115]:=
```
SMTShowMesh["BoundaryConditions" → True]
```



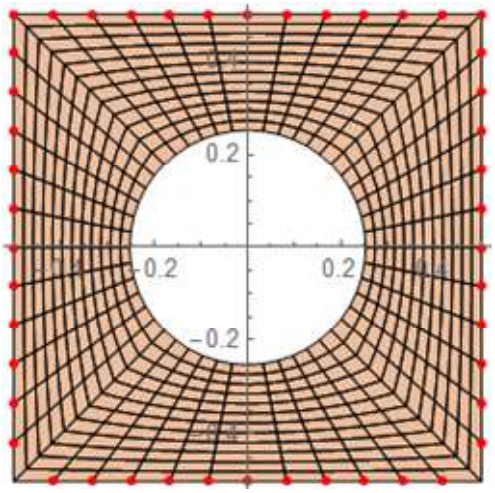
MIEL micro problem

In[116]:=
```
disks =
  Flatten[Table[{{Bsolid + aCell i, Bsolid + bCell j}, rVoid}, {i, 0, nx - 1}, {j, 0, ny - 1}], 1];
substructmicrodata = {disks, 10, SMTMakeDll["ExamplesSEPET1DFHYT1DNeoHookeWA"],
   {"E *" → 21000, "ν *" → 0.3}, "T1", {}};
```

In[118]:=
```
Show[SMTShowMesh["BoundaryConditions" → True, Axes → True],
 Graphics[{White, Map[Disk[#[[1]], #[[2]]] &, disks]}]]
```



FE^2 Micro problem



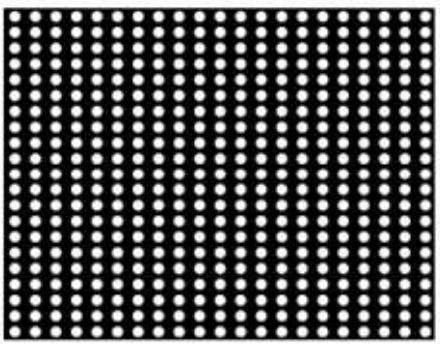
In[119]:=
```
aRVE = aCell; (* RVE length *)
bRVE = bCell; (*RVE height *)
tRVE = 1; (*RVE thickeness *)
NrRVE = 12; (* RVE mesh densitiy in radial direction*)
NabRVE = 12; (* RVE mesh densitiy per side*)
```

- The actual micro mesh is generated by the user defined FE22DMicroMeshVoids[localProblemData, specificMacroData] function where localProblemData is a standard local data structure (see documentation) and specificMacroData is an arbitrary user defined data neded by the FE22DMicroMeshVoids function in order to create micro problem mesh.

In[124]:=
```
FE2microdata =
  {aRVE , bRVE, tRVE, rVoid, NrRVE , NabRVE, SMTMakeDll["ExamplesSEPEQ1DFHYQ1DNeoHookeWA"],
   {"E *" → 21000, "ν *" → 0.3}, "Q1", SMTMakeDll["ExamplesPeriodic2DStructured"], {}};
```

- Micro structure is infinitely periodic

In[125]:=
```
n = 20;
Graphics[{Rectangle[{0, 0}, {(n + 1) aRVE, (n + 1) bRVE}], White,
  Table[Disk[{x + aRVE / 2, y + bRVE / 2}, rVoid], {x, 0, n aRVE, aRVE}, {y, 0, n bRVE, bRVE}]}]
```



Set up multi-scale environment

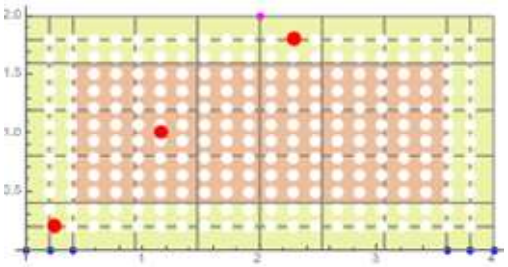
- The "PartialRestart" option can be True only when the local problems are structurally the same mesh (only the actual coordinates of the nodes can be different).
- The "PartialDump" must be False only when during the solution of the local problem the mesh has been changed as well.

In[126]:=
```
SMTMultiScaleSet[
 "FE^2" → {{FE22DMicroMeshVoids, FE2SolveOne} -> FE2Elements},
 "MIEL" → {{MIEL2DMicroMeshVoids, MIELSolveOne} -> MIELElements},
 "SpecificMicroData" →
  {"FE22DMicroMeshVoids" -> FE2microdata, "MIEL2DMicroMeshVoids" -> substructmicrodata},
"MaxKernels" → Automatic, "Threads" → 1, "Console" → False,
 "PartialDump" → True, "PartialRestart" → False]
```

```
True
```

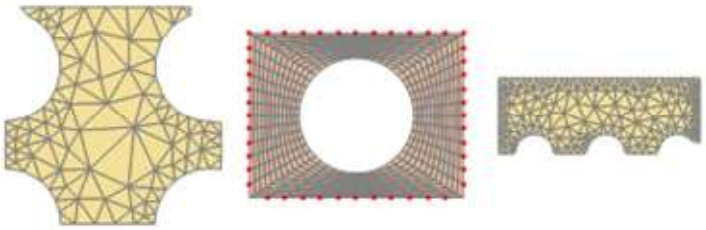
Visualize selected micro problem for post-processing

In[127]:=
```
monitorpoints = {{0.2335, 0.2142}, {1.159, 1.006}, {2.29, 1.803}};
SMTShowMesh["BoundaryConditions" → True,
 Epilog → {{White, Map[Disk[#[[1]], #[[2]]] &, disks]},
   PointSize[.03], Red, Point[monitorpoints]}, Axes → True]
```



- SMTMicroRestart[{x, y}] ... restart micro problem that is nearest to the given spatial position (position is given in initial coordinates) and set the selected micro problem as the current micro problem
- SMTMicroEvaluate[exp] ... evaluate exp for the restarted micro problem

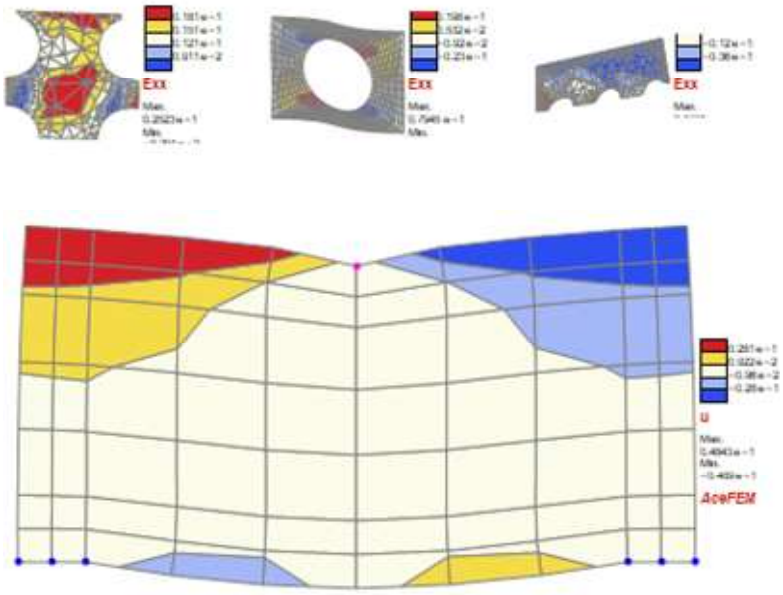
In[129]:=
```
GraphicsRow[Table[SMTMicroRestart[p];
   SMTMicroEvaluate[SMTShowMesh[ImageSize → 200]], {p, monitorpoints}]]
```



Evaluate complete response using an adaptive path following procedure

In[130]:=
```
response = {{0, 0}};
λMax = 1; λ0 = λMax / 4; ΔλMin = λMax / 1000; ΔλMax = λMax / 4;
tolNR = 10^-8; maxNR = 15; targetNR = 8;
SMTNextStep["λ" -> λ0];
While[
  While[
   step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λMax}]
   , SMTMicroSolve["Dump" → False];
   (* react on events at RVE *)
   Switch[SMTSharedStatus[[1]]
    , 0 | 1,
    SMTNewtonIteration[];
    SMTStatusReport[SMTSharedStatus[[1]]];
    , 2,
    (* force step back due to the failed micro simulation *)
    SMTIData[{"Iteration", "TotalIteration"}, SMTIData[{"Iteration", "TotalIteration"}] + 1];
    SMTIData["ErrorStatus", 2];
    SMTStatusReport[SMTSharedStatus[[2]]];
    , 3,
    (*abort due to the fatal error *)
    SMTStatusReport[SMTSharedStatus[[2]]]; Abort[];
   ]
  ]; (* end of NR loop *)
  If[step[[4]] === "MinBound", SMTStatusReport["Analyze"]; Abort[];];
  (* successful step *)
  If[Not[step[[1]]],
   (*dump the state of micro problems to disc before the next step*)
   SMTMicroSolve["Dump" → True];
   (* vizualization and post-processing*)
   macromesh = SMTShowMesh["BoundaryConditions" → True,
     "DeformedMesh" → True, "Field" → "u", "Legend" → True, "Contour" → 4];
   micromesh = Table[
     SMTMicroRestart[x];
     SMTMicroEvaluate[
      SMTShowMesh["DeformedMesh" → True, "Field" → "Exx", "Legend" → True, "Contour" → 4]]
     , {x, monitorpoints}];
   Print[Column[{GraphicsRow[micromesh, ImageSize → 600], Show[macromesh, ImageSize → 600]}]];
   AppendTo[response, {SMTRData["Multiplier"] vA, -SMTResidual[Point[{L / 2, H}]][[1, 2]]}];
  ];
  step[[3]]
  , If[step[[1]], SMTStepBack[];];
  SMTNextStep["Δλ" -> step[[2]]]
];
```

```
Step/Iter=1/1 λ/Δλ=0.25/0.25 ||Δp||/||R||=0.0644127/762.733 Events=0 Status=0/{} Tag=0

Step/Iter=1/2 λ/Δλ=0.25/0.25 ||Δp||/||R||=0.00415726/25.2689 Events=0 Status=0/{} Tag=0

Step/Iter=1/3 λ/Δλ=0.25/0.25 ||Δp||/||R||=0.00001334/0.0658864 Events=0 Status=0/{} Tag=0

Step/Iter=1/4 λ/Δλ=0.25/0.25 ||Δp||/||R||=
 3.15731×10^-10/4.76478×10^-6 Events=0 Status=0/{} Tag=0
```

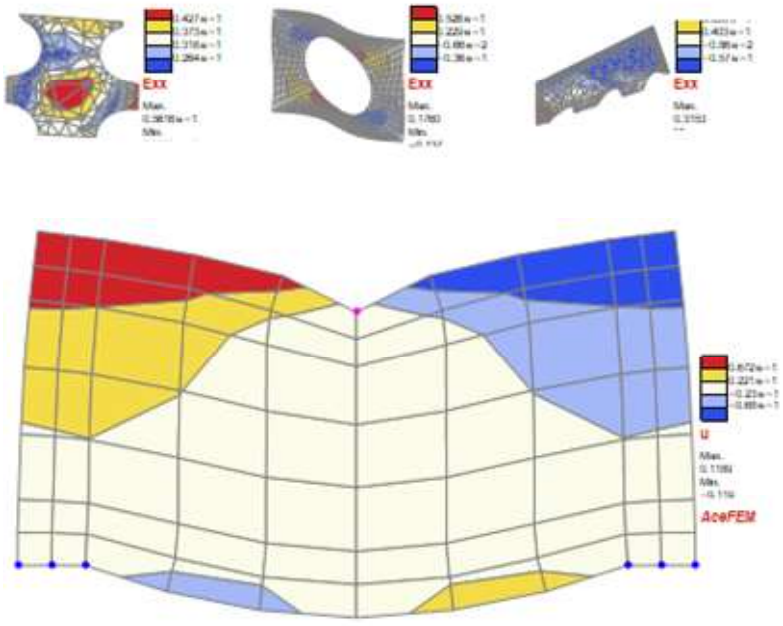


Step/Iter=2/1 λ/Δλ=0.5/0.25 ‖Δp‖/‖R‖=0.066917/788.194 Events=0 Status=0/{} Tag=0

Step/Iter=2/2 λ/Δλ=0.5/0.25 ‖Δp‖/‖R‖=0.00495369/30.0409 Events=0 Status=0/{} Tag=0

Step/Iter=2/3 λ/Δλ=0.5/0.25 ‖Δp‖/‖R‖=0.0000213985/0.0933037 Events=0 Status=0/{} Tag=0

Step/Iter=2/4 λ/Δλ=0.5/0.25 ‖Δp‖/‖R‖=
$7.35113\times10^{-10}/7.77588\times10^{-6}$ Events=0 Status=0/{} Tag=0



Step/Iter=3/1 λ/Δλ=0.75/0.25 ‖Δp‖/‖R‖=0.0704001/787.313 Events=0 Status=0/{} Tag=0

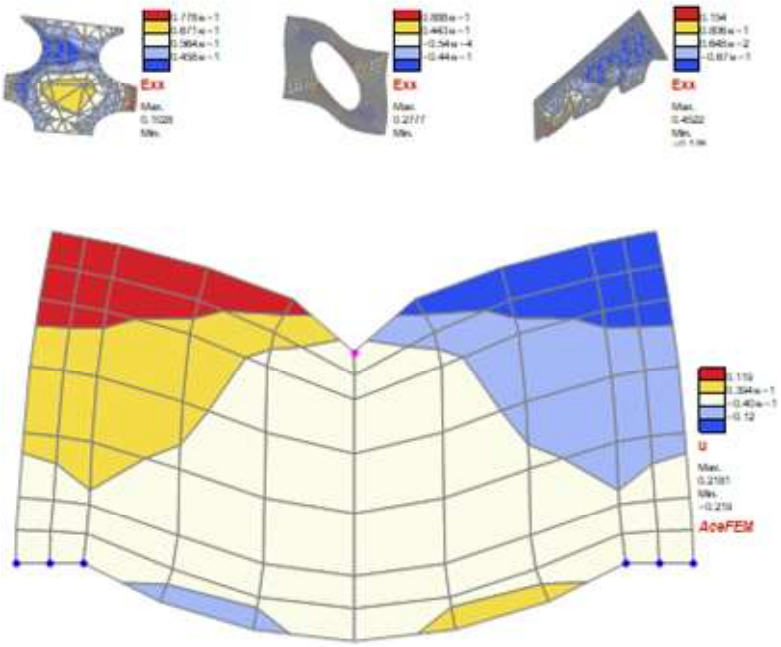
Step/Iter=3/2 λ/Δλ=0.75/0.25 ‖Δp‖/‖R‖=0.00581684/36.4386 Events=0 Status=0/{} Tag=0

Step/Iter=3/3 λ/Δλ=0.75/0.25 ‖Δp‖/‖R‖=0.000680626/6.12976 Events=0 Status=0/{} Tag=0

Step/Iter=3/4 λ/Δλ=0.75/0.25 ‖Δp‖/‖R‖=0.000615885/6.1083 Events=0 Status=0/{} Tag=0

Step/Iter=3/5 λ/Δλ=0.75/0.25 ‖Δp‖/‖R‖=3.82688×10^{-6}/0.022299 Events=0 Status=0/{} Tag=0

Step/Iter=3/6 λ/Δλ=0.75/0.25 ‖Δp‖/‖R‖=
6.216×10^{-11}/3.00593×10^{-7} Events=0 Status=0/{} Tag=0



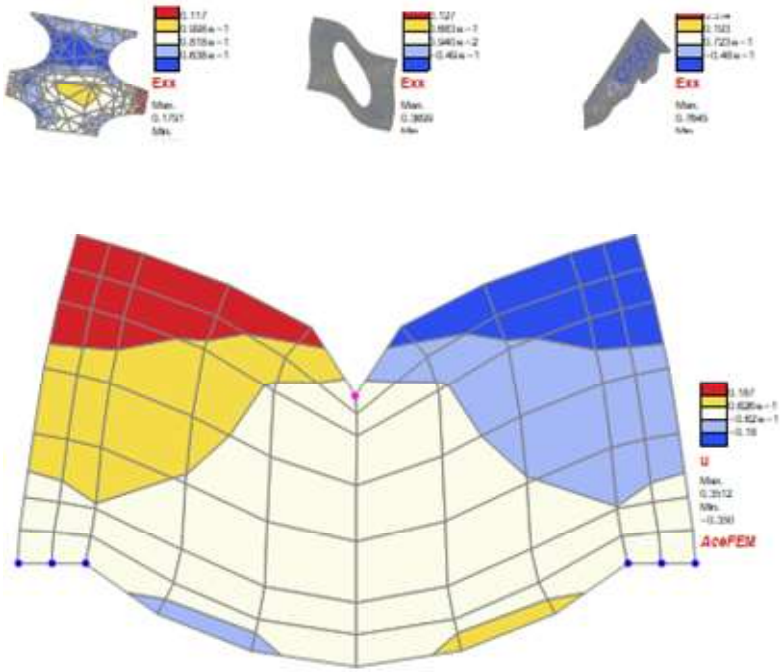
Step/Iter=4/1 λ/Δλ=1./0.25 ‖Δp‖/‖R‖=0.0751182/745.604 Events=0 Status=0/{} Tag=0

Step/Iter=4/2 λ/Δλ=1./0.25 ‖Δp‖/‖R‖=0.00687825/41.6589 Events=0 Status=0/{} Tag=0

Step/Iter=4/3 λ/Δλ=1./0.25 ‖Δp‖/‖R‖=0.000158968/0.243693 Events=0 Status=0/{} Tag=0

Step/Iter=4/4 λ/Δλ=1./0.25 ‖Δp‖/‖R‖=1.07263×10^{-7}/0.000511082 Events=0 Status=0/{} Tag=0

Step/Iter=4/5 λ/Δλ=1./0.25 ‖Δp‖/‖R‖=
6.57494×10^{-13}/1.68876×10^{-9} Events=0 Status=0/{} Tag=0



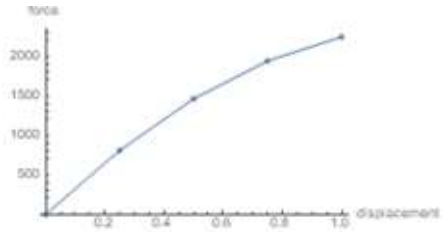
In[135]:= **SMTMultiScaleSimulationReport[];**

MACRO	
Number of macro elements	70
Number of macro equations	163
No. of steps	4
No. of steps back	0
Total no. of iterations	19
Total driver time (s)	1:1.44
Total linear solver time (s)	0.01
Total K&R time (s)	0.00
MICRO	
Number of FE^2 elements	18
Number of MIEL elements	52
Number of single-scale elements	0
Parallelization kernels/threads	16/1
Total number of micro problems	124
Total number of micro elements	59 518
Total number of micro nodes	57 462
Total number of micro equations	114 492
Average number of micro equations	923.323
Real micro mesh generation time	4.97
Number of MicroSolve calls	23
Total MicroSolve time	33.74
Total number of MP equations solved	2 633 316
Total number of steps	3354
Total number of back steps	6
Average number of micro problems/kernel	7.8
Total number of MP solved	2852
Average MP time	0.03
Average MP K&R time	0.00
Average MP linear solver time	0.01
Average MP sens.p.load time	0.00
Average MP sens.l.solver time	0.00
Average MP Schur complement time	0
Average MP task time	0.00
Average MP Driver time	0.24
Average real Driver time	42.21
Total parallel MP time	1:18.87
Number of restarts	2852
Total parallel restart time	5:38.88
Average restart time	0.12
Number of dumps	620
Total parallel dump time	30.41
Average dump time	0.05
Total dump files size (byte)	17 450 394
zlib compression level	2
Macro + micro analysis time	38.72
Aministrative time	8.68
Lost time:	28.81
Total time	47.41

Analysis of the results

Response curve

In[136]:=
```
ListLinePlot[response, AxesLabel → {"displacement", "force"}, PlotMarkers → Automatic]
```



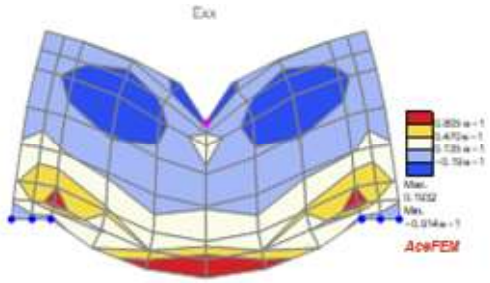
Calculate stiffness (KN/cm)

In[137]:=
```
response[[2, 2]] / response[[2, 1]]
```

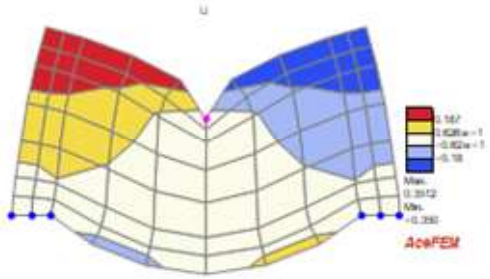
Out[137]=
```
3244.92
```

Project micro solution to macro problem

In[138]:=
```
SMTShowMesh["BoundaryConditions" → True, "DeformedMesh" → True,
 "Field" → SMTMultiScalePostData["Exx"] // Chop, "Legend" → True, "Contour" → 4, "Label" → "Exx"]
```

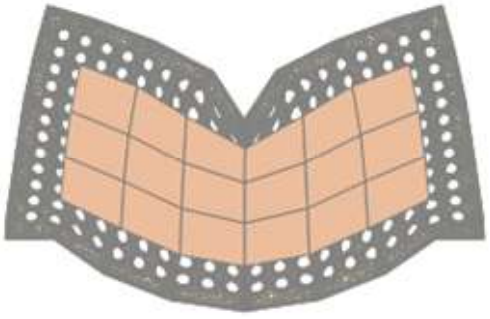


In[139]:=
```
SMTShowMesh["BoundaryConditions" → True, "DeformedMesh" → True,
 "Field" → SMTMultiScalePostData["u"], "Legend" → True, "Contour" → 4, "Label" → "u"]
```



Visualize true micro solution

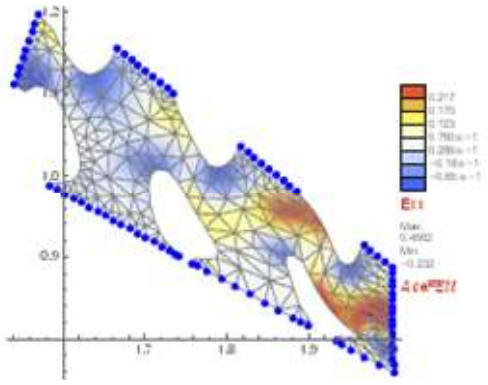
In[140]:=
```
Show[Map[(
    SMTMicroRestart[#[[2]]];
    SMTMicroEvaluate[SMTShowMesh["DeformedMesh" → True]]
  ) &, Cases[SMTAllLocalProblems, {{"MIEL", __}, __}]],
 SMTShowMesh["DeformedMesh" → True, "Domains" → "beam"]]
```



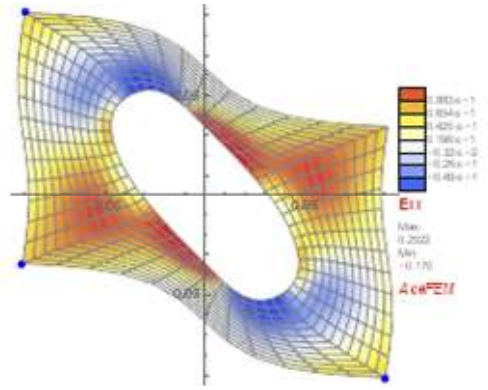
In[141]:=
```
Show[Map[(
    SMTMicroRestart[#[[2]]];
    SMTMicroEvaluate[SMTShowMesh["DeformedMesh" → True,
      "Field" -> "Exx", "Contour" → {-0.02, 0.03, 10}, "Legend" → False, "Mesh" → False]]
  ) &, Cases[SMTAllLocalProblems, {{"MIEL", __}, __}]],
 SMTShowMesh["DeformedMesh" → True, "Domains" → "beam",
  "Field" -> "Exx", "Contour" → {-0.02, 0.03, 10}, "Legend" → False, "Mesh" → False]]
```

Visualization of the chosen micro problem

In[142]:=
```
SMTMicroRestart[{1.75, 1.75}];
SMTMicroEvaluate[SMTShowMesh[Axes → True, "DeformedMesh" → True,
  "BoundaryConditions" → True, "Field" -> "Exx", "Legend" → True]]
```



In[143]:=
```
SMTMicroRestart[{2, 1}];
SMTMicroEvaluate[SMTShowMesh[Axes → True, "DeformedMesh" → True,
  "BoundaryConditions" → True, "Field" -> "Exx", "Legend" → True]]
```



Benchmark Tests and Debugging Procedures

Comparison of multi-scale simulation with single scale simulation

If the F^2 and MIEL formulations are correctly implemented then the results of the macro level simulation (using the element for the micro level simulation) should yield the same results as:

- F^2 simulation in the case of homogeneous materials. Homogeneous material has percentage of perforation set to zero. Micro mesh must be symmetric and structured.
- MIEL simulation in the case of homogeneous materials. Homogeneous material has no voids. Micro mesh must be composed of a single element.

Comparison of 2D multi-scale simulation with 2D single scale simulation

Calculate response curve of macro problem meshed with micro elements.

In[144]:=
```
<< AceFEM`;
```

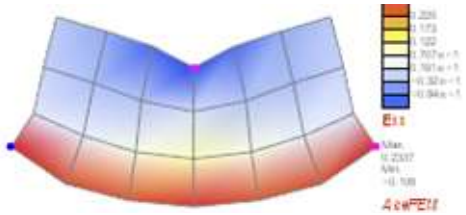
In[145]:=
```
L = 4.; (*length cm/KN*)
H = 2.; (*height*)
Nx = 6; (*mesh density in x direction*)
Ny = 3; (*mesh density in y direction*)
vA = 1;(* prescribed displacement at L/2*)
```

In[150]:=
```
SMTInputData["Threads" → 1];
SMTAddDomain["beam", "ExamplesSEPEQ1DFHYQ1DNeoHookeWA", {"E *" → 21000, "ν *" -> .3}];
SMTAddEssentialBoundary[
  {Point[{0, 0}], 1 -> 0, 2 -> 0}
  , {Point[{L, 0}], 2 -> 0}
  , { Point[{L / 2, H}] , 2 → -vA}];
SMTAddMesh[Polygon[{{0, 0}, {L, 0}, {L, H}, {0, H}}], "beam", "Q1", {Nx, Ny}];
SMTAnalysis[];
```

In[155]:=
```
response = {{0, 0}};
λMax = 1; λ0 = λMax / 4; ΔλMin = λMax / 1000; ΔλMax = λMax / 4;
tolNR = 10^-8; maxNR = 15; targetNR = 8;
SMTNextStep["λ" → λ0];
While[
 While[step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λMax}],
  SMTNewtonIteration[];
  SMTStatusReport[]];
 If[step[[4]] === "MinBound", SMTStatusReport[" Error: Δλ < Δλmin"]; Abort[];];
 If[Not[step[[1]]],
  AppendTo[response, {SMTRData["Multiplier"] vA, -SMTResidual[Point[{L / 2, H}]][[1, 2]]}]];
 step[[3]]
 , If[step[[1]], SMTStepBack[];];
 SMTNextStep["Δλ" -> step[[2]]]
]
```

```
Step/Iter=1/1 λ/Δλ=0.25/0.25 ||Δp||/||R||=0.125286/689.462 Events=0 Status=0/{}
Step/Iter=1/2 λ/Δλ=0.25/0.25 ||Δp||/||R||=0.003791/51.6797 Events=0 Status=0/{}
Step/Iter=1/3 λ/Δλ=0.25/0.25 ||Δp||/||R||=0.0000301258/0.0692578 Events=0 Status=0/{}
Step/Iter=1/4 λ/Δλ=0.25/0.25 ||Δp||/||R||=6.41202×10^-10/4.70166×10^-6 Events=0 Status=0/{}
Step/Iter=2/1 λ/Δλ=0.5/0.25 ||Δp||/||R||=0.124446/697.57 Events=0 Status=0/{}
Step/Iter=2/2 λ/Δλ=0.5/0.25 ||Δp||/||R||=0.00471548/55.0041 Events=0 Status=0/{}
Step/Iter=2/3 λ/Δλ=0.5/0.25 ||Δp||/||R||=0.0000131103/0.0633285 Events=0 Status=0/{}
Step/Iter=2/4 λ/Δλ=0.5/0.25 ||Δp||/||R||=7.44948×10^-11/7.09361×10^-7 Events=0 Status=0/{}
Step/Iter=3/1 λ/Δλ=0.75/0.25 ||Δp||/||R||=0.122165/704.868 Events=0 Status=0/{}
Step/Iter=3/2 λ/Δλ=0.75/0.25 ||Δp||/||R||=0.00529845/56.2095 Events=0 Status=0/{}
Step/Iter=3/3 λ/Δλ=0.75/0.25 ||Δp||/||R||=5.12668×10^-6/0.0676367 Events=0 Status=0/{}
Step/Iter=3/4 λ/Δλ=0.75/0.25 ||Δp||/||R||=3.51072×10^-11/4.12285×10^-7 Events=0 Status=0/{}
Step/Iter=4/1 λ/Δλ=1./0.25 ||Δp||/||R||=0.119198/712.17 Events=0 Status=0/{}
Step/Iter=4/2 λ/Δλ=1./0.25 ||Δp||/||R||=0.00558112/55.8779 Events=0 Status=0/{}
Step/Iter=4/3 λ/Δλ=1./0.25 ||Δp||/||R||=9.60951×10^-6/0.0717627 Events=0 Status=0/{}
Step/Iter=4/4 λ/Δλ=1./0.25 ||Δp||/||R||=9.44213×10^-11/1.1683×10^-6 Events=0 Status=0/{}
```

In[160]:=
```
SMTShowMesh["BoundaryConditions" → True, "DeformedMesh" → True, "Field" → "Exx"]
```



Here the single scale result is compared to the results of FE2 and MIEL simulations

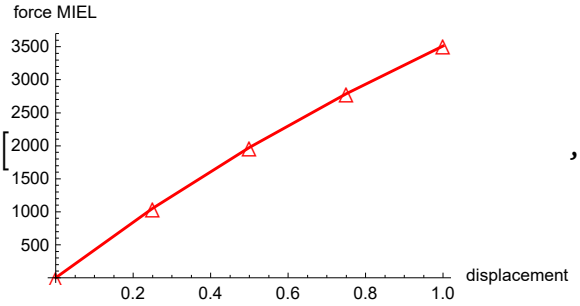
- The FE2 response curve was obtained by running the example "Example: FE^2 Modeling of Uniformly Distributed Micro-structure" with percentage of perforation set to zero thus

```
pRVE = 0.;
```

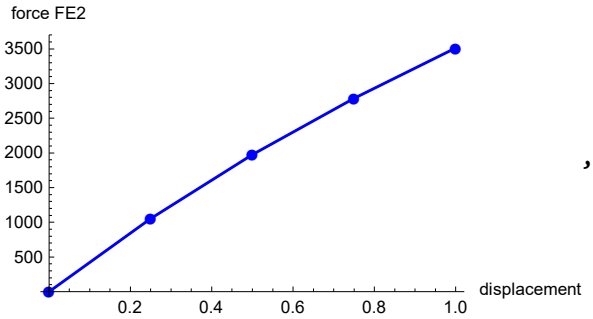
- The MIEL response curve was obtained by running the example "Example: MIEL Modeling of Uniformly Distributed Micro-structure" without voids and micro mesh composed of a single Q1 element, thus

In[623]:=
```
specificMacroData = {{}, 1,
    SMTMakeDll["ExamplesSEPEQ1DFHYQ1DNeoHookeWA"], {"E *" → 21000, "ν *" → 0.3}, "Q1", {}};
```

In[161]:=
```
Show[
```

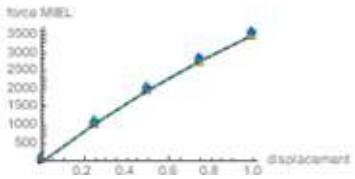


,



,

```
ListLinePlot[response, PlotMarkers → "X", PlotStyle → {Green, Dashed}], PlotRange → All]
```



Comparison of 3D multi-scale simulation with 3D single scale simulation

Calculate response curve of macro problem meshed with micro elements.

In[162]:=
```
<< AceFEM`;
```

In[163]:=
```
L = 12; B = 2.4; H = 2.4;
n = 2; nx = 5 * n;  ny = 1 * n;  nz = 1 * n;
p = 50;
```

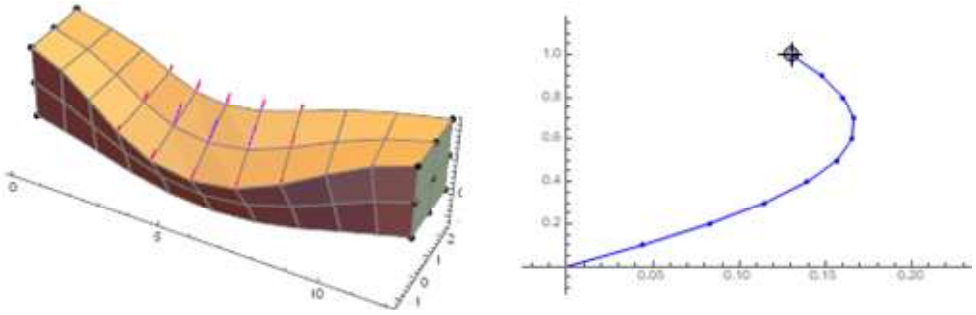
In[166]:=
```
SMTInputData["Threads" → 1];
SMTAddDomain[{"beam", "ExamplesSED3H1DFHYH1DNeoHookeWA", {"E *" → 2000, "ν *" → 0.3}}];
points = {{0, 0, 0}, {L, 0, 0}, {L, B, 0}, {0, B, 0}, {0, 0, H}, {L, 0, H}, {L, B, H}, {0, B, H}};
SMTAddMesh[Hexahedron[points], "beam", "H1", {nx, ny, nz}];
SMTAddEssentialBoundary[
  Polygon[{{0, 0, 0}, {0, 0, H}, {0, B, H}, {0, B, 0}}], 1 → 0, 2 → 0, 3 → 0];
SMTAddEssentialBoundary[Polygon[{{L, 0, 0}, {L, 0, H}, {L, B, H}, {L, B, 0}}], 1 → 0, 2 → 0, 3 → 0];
SMTAddNaturalBoundary[
  Polygon[{{L / 10 * 3, 0, H}, {L / 10 * 7, 0, H}, {L / 10 * 7, B, H}, {L / 10 * 3, B, H}}],
  2 → Polygon[{-p}], 3 → Polygon[{-p}]];
SMTAnalysis[];
```

In[174]:=
```
SMTAnimationOfResponse["Initialize", {0, 0}];
λMax = 1; λ0 = λMax / 10; ΔλMin = λMax / 1000; ΔλMax = λMax / 10;
tolNR = 10^-8; maxNR = 15; targetNR = 8;
SMTNextStep["λ" → λ0];
While[
 While[step = SMTConvergence[tolNR, maxNR, {"Adaptive BC", targetNR, ΔλMin, ΔλMax, λMax}],
  SMTNewtonIteration[];
  SMTStatusReport[]];
 If[step[[4]] === "MinBound", SMTStatusReport[" Error: Δλ < Δλmin"]; Abort[];];
 SMTAnimationOfResponse[
  "LeadingNodePosition" → {L / 2, 0, 0}, "x" → Hold[-SMTPostData["v", Point[{L / 2, 0, 0}]]]
  , "Show" → "Window" | {"ExportFrames", "responseFile"}
 ];
 step[[3]]
 , If[step[[1]], SMTStepBack[];];
 SMTNextStep["Δλ" -> step[[2]]]
]
```

```
Step/Iter=1/1 λ/Δλ=0.1/0.1 ‖Δp‖/‖R‖=0.0578997/1.49666 Events=0 Status=0/{}
Step/Iter=1/2 λ/Δλ=0.1/0.1 ‖Δp‖/‖R‖=0.000964583/1.60258 Events=0 Status=0/{}
Step/Iter=1/3 λ/Δλ=0.1/0.1 ‖Δp‖/‖R‖=0.000082916/0.00240267 Events=0 Status=0/{}
Step/Iter=1/4 λ/Δλ=0.1/0.1 ‖Δp‖/‖R‖=9.68535×10^-9/0.000016283 Events=0 Status=0/{}
Step/Iter=2/1 λ/Δλ=0.2/0.1 ‖Δp‖/‖R‖=0.0587716/1.49666 Events=0 Status=0/{}
Step/Iter=2/2 λ/Δλ=0.2/0.1 ‖Δp‖/‖R‖=0.00100677/1.79403 Events=0 Status=0/{}
Step/Iter=2/3 λ/Δλ=0.2/0.1 ‖Δp‖/‖R‖=0.0000984578/0.00267846 Events=0 Status=0/{}
Step/Iter=2/4 λ/Δλ=0.2/0.1 ‖Δp‖/‖R‖=1.25787×10^-8/0.0000230852 Events=0 Status=0/{}
Step/Iter=2/5 λ/Δλ=0.2/0.1 ‖Δp‖/‖R‖=1.72252×10^-14/6.67044×10^-13 Events=0 Status=0/{}
Step/Iter=3/1 λ/Δλ=0.3/0.1 ‖Δp‖/‖R‖=0.0593828/1.49666 Events=0 Status=0/{}
Step/Iter=3/2 λ/Δλ=0.3/0.1 ‖Δp‖/‖R‖=0.00106736/2.0022 Events=0 Status=0/{}
Step/Iter=3/3 λ/Δλ=0.3/0.1 ‖Δp‖/‖R‖=0.000113484/0.00291988 Events=0 Status=0/{}
Step/Iter=3/4 λ/Δλ=0.3/0.1 ‖Δp‖/‖R‖=1.52678×10^-8/0.0000307997 Events=0 Status=0/{}
Step/Iter=3/5 λ/Δλ=0.3/0.1 ‖Δp‖/‖R‖=2.74269×10^-14/8.10218×10^-13 Events=0 Status=0/{}
Step/Iter=4/1 λ/Δλ=0.4/0.1 ‖Δp‖/‖R‖=0.0597412/1.49666 Events=0 Status=0/{}
Step/Iter=4/2 λ/Δλ=0.4/0.1 ‖Δp‖/‖R‖=0.0011337/2.22415 Events=0 Status=0/{}
Step/Iter=4/3 λ/Δλ=0.4/0.1 ‖Δp‖/‖R‖=0.000125721/0.0030903 Events=0 Status=0/{}
Step/Iter=4/4 λ/Δλ=0.4/0.1 ‖Δp‖/‖R‖=1.69978×10^-8/0.0000379259 Events=0 Status=0/{}
Step/Iter=4/5 λ/Δλ=0.4/0.1 ‖Δp‖/‖R‖=3.57598×10^-14/9.26059×10^-13 Events=0 Status=0/{}
Step/Iter=5/1 λ/Δλ=0.5/0.1 ‖Δp‖/‖R‖=0.0598652/1.49666 Events=0 Status=0/{}
Step/Iter=5/2 λ/Δλ=0.5/0.1 ‖Δp‖/‖R‖=0.00119281/2.45367 Events=0 Status=0/{}
Step/Iter=5/3 λ/Δλ=0.5/0.1 ‖Δp‖/‖R‖=0.000132157/0.00315028 Events=0 Status=0/{}
Step/Iter=5/4 λ/Δλ=0.5/0.1 ‖Δp‖/‖R‖=1.69982×10^-8/0.0000420142 Events=0 Status=0/{}
Step/Iter=5/5 λ/Δλ=0.5/0.1 ‖Δp‖/‖R‖=3.63068×10^-14/1.00459×10^-12 Events=0 Status=0/{}
Step/Iter=6/1 λ/Δλ=0.6/0.1 ‖Δp‖/‖R‖=0.0597734/1.49666 Events=0 Status=0/{}
Step/Iter=6/2 λ/Δλ=0.6/0.1 ‖Δp‖/‖R‖=0.00123635/2.68046 Events=0 Status=0/{}
```

```
Step/Iter=6/3 λ/Δλ=0.6/0.1 ||Δp||/||R||=0.000129566/0.00306797 Events=0 Status=0/{}
Step/Iter=6/4 λ/Δλ=0.6/0.1 ||Δp||/||R||=1.48997×10^-8/0.0000404379 Events=0 Status=0/{}
Step/Iter=6/5 λ/Δλ=0.6/0.1 ||Δp||/||R||=2.64601×10^-14/7.60168×10^-13 Events=0 Status=0/{}
Step/Iter=7/1 λ/Δλ=0.7/0.1 ||Δp||/||R||=0.0594756/1.49666 Events=0 Status=0/{}
Step/Iter=7/2 λ/Δλ=0.7/0.1 ||Δp||/||R||=0.00126209/2.89024 Events=0 Status=0/{}
Step/Iter=7/3 λ/Δλ=0.7/0.1 ||Δp||/||R||=0.000115517/0.00283461 Events=0 Status=0/{}
Step/Iter=7/4 λ/Δλ=0.7/0.1 ||Δp||/||R||=1.10423×10^-8/0.0000321059 Events=0 Status=0/{}
Step/Iter=7/5 λ/Δλ=0.7/0.1 ||Δp||/||R||=1.30229×10^-14/5.03935×10^-13 Events=0 Status=0/{}
Step/Iter=8/1 λ/Δλ=0.8/0.1 ||Δp||/||R||=0.0589693/1.49666 Events=0 Status=0/{}
Step/Iter=8/2 λ/Δλ=0.8/0.1 ||Δp||/||R||=0.00127277/3.06614 Events=0 Status=0/{}
Step/Iter=8/3 λ/Δλ=0.8/0.1 ||Δp||/||R||=0.0000896424/0.00248547 Events=0 Status=0/{}
Step/Iter=8/4 λ/Δλ=0.8/0.1 ||Δp||/||R||=6.44753×10^-9/0.0000191615 Events=0 Status=0/{}
Step/Iter=9/1 λ/Δλ=0.9/0.1 ||Δp||/||R||=0.0582428/1.49666 Events=0 Status=0/{}
Step/Iter=9/2 λ/Δλ=0.9/0.1 ||Δp||/||R||=0.00127337/3.19163 Events=0 Status=0/{}
Step/Iter=9/3 λ/Δλ=0.9/0.1 ||Δp||/||R||=0.0000547113/0.0021244 Events=0 Status=0/{}
Step/Iter=9/4 λ/Δλ=0.9/0.1 ||Δp||/||R||=2.59363×10^-9/6.80134×10^-6 Events=0 Status=0/{}
Step/Iter=10/1 λ/Δλ=1./0.1 ||Δp||/||R||=0.0572853/1.49666 Events=0 Status=0/{}
Step/Iter=10/2 λ/Δλ=1./0.1 ||Δp||/||R||=0.00126835/3.25407 Events=0 Status=0/{}
Step/Iter=10/3 λ/Δλ=1./0.1 ||Δp||/||R||=0.0000198214/0.00192438 Events=0 Status=0/{}
Step/Iter=10/4 λ/Δλ=1./0.1 ||Δp||/||R||=5.9089×10^-10/4.00246×10^-7 Events=0 Status=0/{}
```

In[179]:=
```
SMTAnimationOfResponse["Report"]
```



In[180]:=
```
SMTAnimationOfResponse["Response"]
```

```
{{{0, 0}, {0.0441321, 0.1}, {0.0827114, 0.2},
  {0.114769, 0.3}, {0.139511, 0.4}, {0.156413, 0.5}, {0.165294, 0.6},
  {0.166367, 0.7}, {0.160239, 0.8}, {0.14787, 0.9}, {0.13048, 1.}}}
```

Here the single scale result is compared to the results of FE2 and MIEL simulations

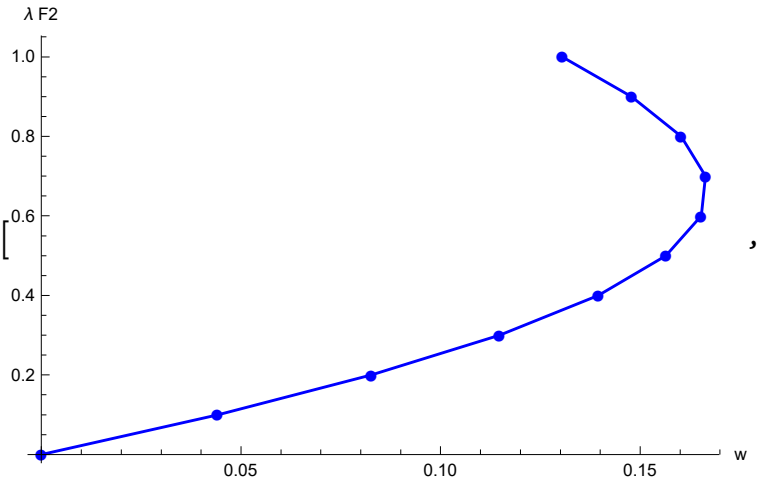
- The FE2 response curve was obtained by running the example "Example: FE^2 Modeling of Uniformly Distributed Micro-structure" with percentage of perforation set to zero thus

```
pRVE = 0.;
```

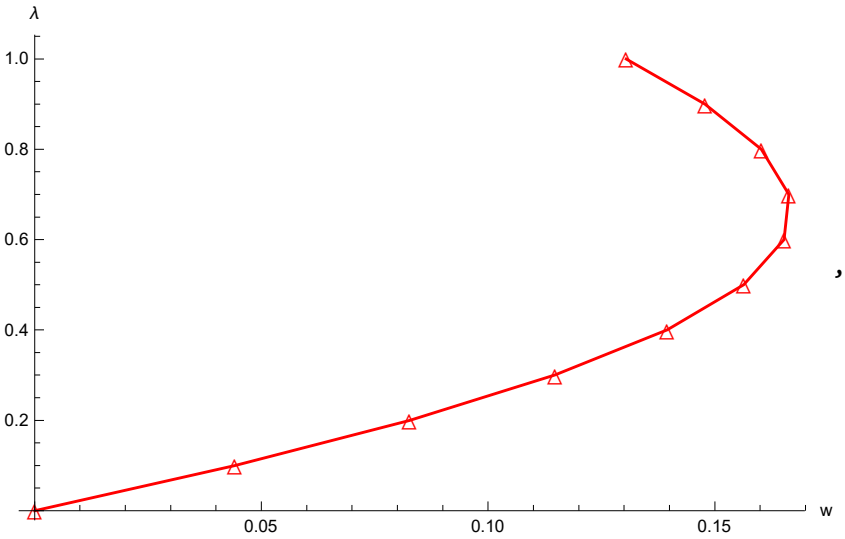
- The MIEL response curve was obtained by running the example "Example: MIEL Modeling of Uniformly Distributed Micro-structure" without voids and micro mesh composed of a single Q1 element, thus

In[623]:=
```
specificMacroData = {{}, 1,
   SMTMakeDll["ExamplesSEPEQ1DFHYQ1DNeoHookeWA"], {"E *" → 21000, "ν *" → 0.3}, "H1", {}};
```

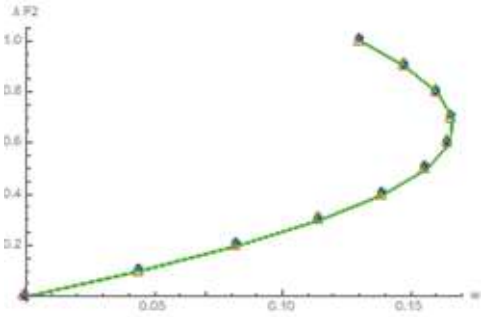
In[181]:=
```
Show[
```




```
,
```




```
, ListLinePlot[
SMTAnimationOfResponse["Response"], PlotMarkers → "X", PlotStyle → {Green, Dashed}]]
```



Test of single micro structure at main kernel

2D FE^2 micro structure

- Solve micro structure for given macro deformation gradient.
  - $F = \begin{pmatrix} 1.5 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
- Initialize multi-scale anvironment.

In[182]:=
```
SMTSpatialDimension = 2;
SMTMultiScaleSet["TestSingleScale" → True, "Threads" → 1, "Console" → True]
```

```
True
```

- The actual micro mesh is generated by the user defined FE22DMicroMeshVoids[localProblemData, specificMacroData] function where localProblemData is a standard local data structure (see documentation) and specificMacroData is an arbitrary user defined data needed by the FE22DMicroMeshVoids function in order to create micro problem mesh.
- local_data data structure
  - Prepare local_data data structure accordingly to documentation.
  - Proper local_data data structure of selected micro problem is also stored after the SMTMicroRestart[selected_micro_point] command in SMTCurrentLocalProblem global variable (see also Visualize selected micro problem section).

In[184]:=
```
localProblemData = {
    {"FE^2", FE22DMicroMeshVoids, FE2SolveOne, 4, "Deformation gradient (plane strain)",
     0, "Integrated stress and sensitivity (P, plane strain)", 0}
    , (*X*){0, 0}
    , 1, 1, 1, 1
    , (*vec(F)*){1.5, 0.5, 0, 1}
    , {}, (*RVE volume*)1, {}};
```

- specificMacroData data structure

In[185]:=
```
specificMacroData = {(*aRVE*)1 , (*bRVE*)1, (*tRVE*) 0.1, (*rRVE*) 0.1, (*NrRVE*)2 ,
   (*NabRVE*) 2 , SMTMakeDll["ExamplesSEPEQ1DFHYQ1DNeoHookeWA"], {"E *" → 21 000, "ν *" → 0.3},
  "Q1", SMTMakeDll["ExamplesPeriodic2DStructured"], {"MinSubSteps" → 1, "MaxSubSteps" → 10}}
```

```
{1, 1, 0.1, 0.1, 2, 2,
 {C:\Users\jkorelc\AppData\Roaming\Mathematica\Applications\AceFEM\Elements\
    ExamplesSEPEQ1DFHYQ1DNeoHookeWA.W64.dll,
  ExamplesSEPEQ1DFHYQ1DNeoHookeWA, 0}, {E * → 21 000, ν * → 0.3}, Q1,
 {C:\Users\jkorelc\Desktop\SMS\Documentation\ExamplesPeriodic2DStructured.W64.dll,
  ExamplesPeriodic2DStructured, 0}, {MinSubSteps → 1, MaxSubSteps → 10}}
```

- Create micro mesh and return augmented local_data data structure.

In[186]:=
```
localProblemData =
 FE22DMicroMeshVoids[localProblemData, {"FE22DMicroMeshVoids" -> specificMacroData}]
```

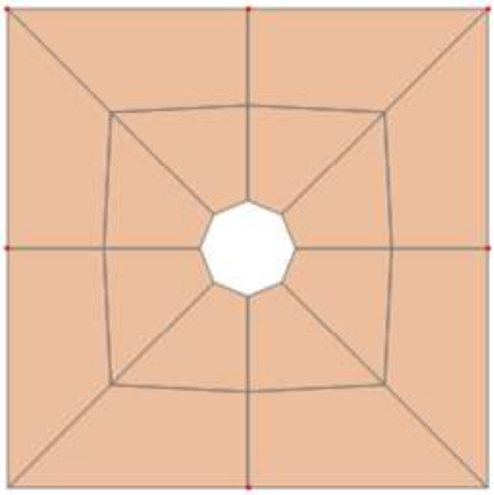
```
{{FE^2, FE22DMicroMeshVoids, FE2SolveOne, 4, Deformation gradient (plane strain), 0,
  Integrated stress and sensitivity (P, plane strain), 0}, {0, 0}, {1, 1, 0.1, 0.1, 2,
  2, {C:\Users\jkorelc\AppData\Roaming\Mathematica\Applications\AceFEM\Elements\
     ExamplesSEPEQ1DFHYQ1DNeoHookeWA.W64.dll,
   ExamplesSEPEQ1DFHYQ1DNeoHookeWA, 0}, {E * → 21 000, ν * → 0.3}, Q1,
  {C:\Users\jkorelc\Desktop\SMS\Documentation\ExamplesPeriodic2DStructured.W64.dll,
   ExamplesPeriodic2DStructured, 0}, {MinSubSteps → 1, MaxSubSteps → 10}}, 1,
 1, 1, {1.5, 0.5, 0, 1}, {10, 16, 4}, 0.1, {MinSubSteps → 1, MaxSubSteps → 10}}
```

- Check for message created during mesh generation phase.

In[187]:=
```
SMTSharedStatus
```

```
{0, 0}
```

In[188]:=
```
SMTShowMesh[]
```



- Solve given macro problem for given macro boundary conditions.
- turn on debugging

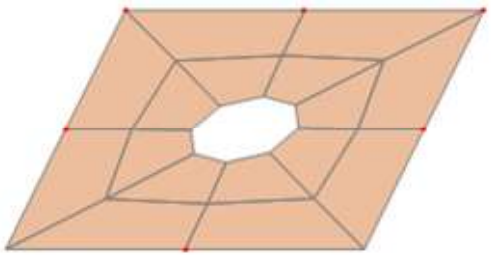
In[189]:=
```
SMTMacroProblemStatus[[14]] = -1
```

```
-1
```

In[190]:=
```
FE2SolveOne[localProblemData, False]
```

```
{106961., 38493.2, 4237.35, 64274.3, 65569.3, 177308., -152.318, -50375.6, 142348.,
 144303., -152.318, 76961.6, 8548.54, -270.982, -178.343, -50375.6, 8548.54,
 102220., -75642.5, -72329.2, 142348., -270.982, -75642.5, 303372., 216631.}
```

In[191]:=
```
SMTShowMesh["DeformedMesh" → True]
```



3D FE^2 micro structure - unstructured mesh

- Solve micro structure for given macro deformation gradient.

In[192]:=
```
F = ( 1.5  0.5  0 )
    (  0    1   0 );
    (  0    0   1 )
```

- Initialize multi-scale anvironment.

In[193]:=
```
SMTSpatialDimension = 3;
SMTMultiScaleSet["TestSingleScale" → True, "Threads" → 1, "Console" → True];
```

- The actual micro mesh is generated by the user defined FE23DMicroMeshVoids[localProblemData, specificMacroData] function where localProblemData is a standard local data structure (see documentation) and specificMacroData is an arbitrary user defined data needed by the FE23DMicroMeshVoids function in order to create micro problem mesh.
- *local_problem_data* data structure accordingly to documentation

In[195]:=
```
localProblemData = {{"FE^2", FE23DMicroMeshVoids, FE2SolveOne, 8, "Deformation gradient (3D)",
    9, "Integrated stress and sensitivity (P, 3D)", 90}, {0.25358983848622446`,
    0.2535898384862246`, 0.2535898384862246`}, 1, 1, 1, 1, F // Flatten, {}, 1, {}};
```

- specificMacroData data structure

In[196]:=
```
nn = 4; (*micro devision*)
aRVE = 1; (*RVE length *)
bRVE = 1; (*RVE height *)
cRVE = 1; (*RVE height *)
pRVE = 0.3;(* percentage of perforation*)
rRVE = CubeRoot[(pRVE aRVE bRVE cRVE 3) / (π 4)]; (* radius of perforation*)
NrRVE = nn; (* RVE mesh densitiy in radial direction*)
NabRVE = nn;
```

In[204]:=
```
specificMacroData =
  {aRVE , bRVE, cRVE, rRVE, NrRVE , NabRVE, SMTMakeDll["ExamplesSED3O1DFHYO1DNeoHookeWA"],
   {"E *" → 21000, "ν *" → 0.3}, "O1", SMTMakeDll["ExamplesPeriodic3DUnstructured"],
   SMTMakeDll["ExamplesSurfaceTriangulationP1"], {"MinSubSteps" → 1, "MaxSubSteps" → 10}};
```

- Create micro mesh and return augmented *local_problem_data* structure.

In[205]:=
```
localProblemData =
 FE23DMicroMeshVoids[localProblemData, {"FE23DMicroMeshVoids" -> specificMacroData}]
```

```
{{FE^2, FE23DMicroMeshVoids, FE2SolveOne, 8,
   Deformation gradient (3D), 9, Integrated stress and sensitivity (P, 3D), 90},
  {0.25359, 0.25359, 0.25359}, {1, 1, 1, 0.415283, 4, 4,
   {C:\Users\jkorelc\AppData\Roaming\Mathematica\Applications\AceFEM\Elements\
      ExamplesSED3O1DFHYO1DNeoHookeWA.W64.dll,
    ExamplesSED3O1DFHYO1DNeoHookeWA, 0}, {E * → 21000, ν * → 0.3}, O1,
   {C:\Users\jkorelc\Desktop\SMS\Documentation\ExamplesPeriodic3DUnstructured.W64.dll,
    ExamplesPeriodic3DUnstructured, 0},
   {C:\Users\jkorelc\Desktop\SMS\Documentation\ExamplesSurfaceTriangulationP1.W64.dll,
    ExamplesSurfaceTriangulationP1, 0}, {MinSubSteps → 1, MaxSubSteps → 10}}, 1,
  1, 1, {1.5, 0.5, 0, 0, 1, 0, 0, 0, 1}, {731, 613, 283, 368, 713, 587, 266, 359},
  1, {MinSubSteps → 1, MaxSubSteps → 10}}
```

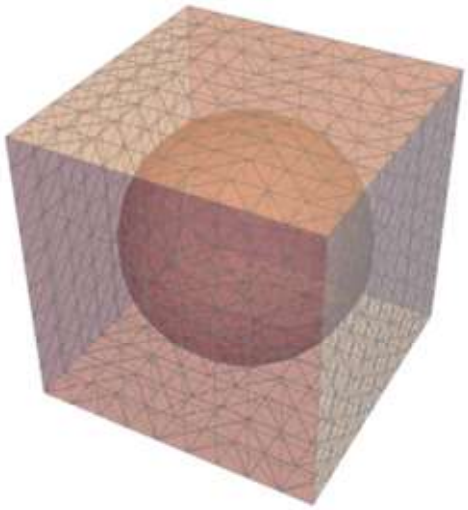
- Check for message created during mesh generation phase.

In[206]:=
```
SMTSharedStatus
```

```
{0, 0}
```

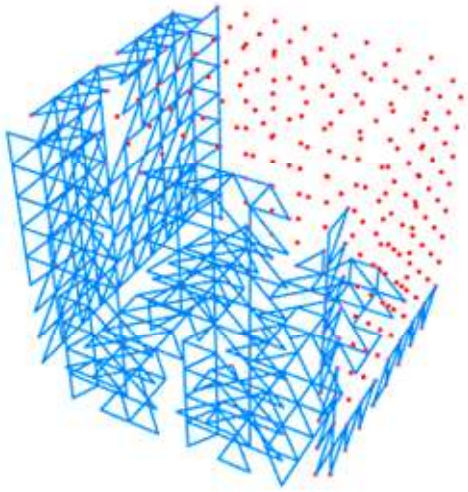
- Solid micro mesh.

In[207]:=
```
SMTShowMesh["Opacity" → 0.4, "ZoomElements" → "micro"]
```



- Mesh of elements used to enforce periodic boundary conditions.

In[208]:=
```
SMTShowMesh["ZoomElements" → "periodic"]
```



- turn on debugging

In[209]:=
```
SMTMacroProblemStatus[[14]] = -1;
```

- check first and last 10 eigenvalues of global tangent matrix at the zero state

In[210]:=
```
mat = SMTData["TangentMatrix"];
{Eigenvalues[mat, 10], Eigenvalues[mat, -10]} // Chop
```

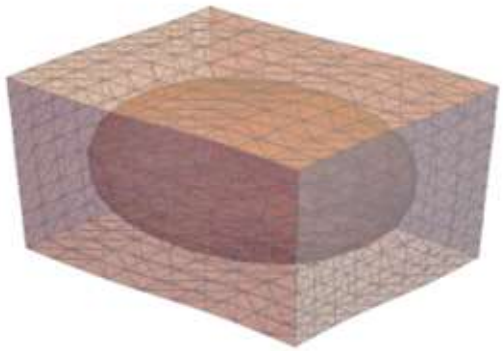
```
{{37 869.7, 37 827.3, 34 803., 34 431.5, 34 160.7, 33 843.2, 33 312.5, 33 283.8, 32 745.9,
  32 685.}, {-0.000119654, -0.000119329, -0.000115766, -0.000115716, -0.000114028,
  -0.000113596, -0.000110782, -0.000109718, -0.000108216, -0.000106504}}
```

- Solve micro problem for given **F** and given mesh. Function returns *micro_data/Volume* where *micro_data* = $\int_{\Omega_e} \text{Flatten}\left[\left\{\text{vec}(P), \frac{\partial \text{vec}(P)}{\partial \text{vec}(F_M)}\right\}\right] dV$.

In[212]:=
```
FE2SolveOne[localProblemData, False] // Chop
```

```
{5950.92, 2437.38, 0.38541, 772.314, 2557.81, 0.622298, 0.0495072, 0.622298,
 2509.56, 8871.55, 210.349, -0.484139, -1658.64, 3852., 1.61881, -0.895368,
 1.61881, 3949.21, 210.349, 4976.64, 1.17396, 1432.32, 540.7, 0.965734, 0.0458633,
 0.965734, 394.583, -0.484139, 1.17396, 4871.3, 1.27915, -2.73412, 91.6635,
 1543.94, 91.6635, -2.58428, -1658.64, 1432.32, 1.27915, 5754.61, -2497.34,
 -0.0958337, 0.901211, -0.145341, -1686.6, 3852., 540.7, -2.73412, -2497.34,
 13448.2, 2.98979, -2.61191, 2.36749, 5848.96, 1.61881, 0.965734, 91.6635,
 -0.0958337, 2.98979, 4852.54, -719.884, 2342.98, 0.835701, -0.895368, 0.0458633,
 1543.94, 0.901211, -2.61191, -719.884, 4979.1, 52.4296, -1.95191, 1.61881,
 0.965734, 91.6635, -0.145341, 2.36749, 2342.98, 52.4296, 4900.79, 1.458, 3949.21,
 394.583, -2.58428, -1686.6, 5848.96, 0.835701, -1.95191, 1.458, 13918.4}
```

In[213]:=
```
SMTShowMesh["Opacity" → 0.4, "DeformedMesh" → True, "ZoomElements" → "micro"]
```



- check first and last 10 eigenvalues of global tangent matrix at the end of first step

In[214]:=
```
mat = SMTData["TangentMatrix"];
{Eigenvalues[mat, 10], Eigenvalues[mat, -10]} // Chop
```

```
{{42866.2, 42121.3, 40835.5, 40729., 37845.9, 37758.7, 36547.5, 36528.3, 35874.1,
  35853.7}, {-0.000113007, -0.000111252, -0.000109181, -0.000105723, -0.00010563,
  -0.00010402, -0.000102546, -0.000101309, -0.0000973125, -0.0000963169}}
```

3D FE^2 micro structure - structured mesh

- Solve micro structure for given macro deformation gradient.

In[216]:=
$$F = \begin{pmatrix} 1.5 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix};$$

- Initialize multi-scale anvironment.

In[217]:=
```
SMTSpatialDimension = 3;
SMTMultiScaleSet["TestSingleScale" → True, "Threads" → 1, "Console" → True];
```

- The actual micro mesh is generated by the user defined FE23DMicroMeshVoids[localProblemData, specificMacroData] function where localProblemData is a standard local data structure (see documentation) and specificMacroData is an arbitrary user defined data needed by the FE23DMicroMeshVoids function in order to create micro problem mesh.
- *local_problem_data* data structure
  - Prepare *local_problem_data* structure accordingly to documentation.

In[219]:=
```
localProblemData = {{"FE^2", FE23DMicroMeshVoids, FE2SolveOne, 8, "Deformation gradient (3D)",
    9, "Integrated stress and sensitivity (P, 3D)", 90}, {0.25358983848622446`,
    0.2535898384862246`, 0.2535898384862246`}, 1, 1, 1, 1, F // Flatten, {}, 1, {}};
```

- specificMacroData data structure

In[220]:=
```
nn = 4; (*micro devision*)
aRVE = 1; (*RVE length *)
bRVE = 1; (*RVE height *)
cRVE = 1; (*RVE height *)
pRVE = 0; (* percentage of perforation*)
rRVE = CubeRoot[(pRVE aRVE bRVE cRVE 3) / (π 4)]; (* radius of perforation*)
NrRVE = nn; (* RVE mesh densitiy in radial direction*)
NabRVE = nn;
```

In[228]:=
```
specificMacroData = {aRVE , bRVE, cRVE, rRVE, NrRVE , NabRVE,
  SMTMakeDll["ExamplesSED3H1DFHYH1DNeoHookeWA"], {"E *" → 21000, "ν *" → 0.3}, "H1",
  SMTMakeDll["ExamplesPeriodic3DStructured"], "", {"MinSubSteps" → 1, "MaxSubSteps" → 10}}
```

```
{1, 1, 1, 0, 4, 4,
 {C:\Users\jkorelc\AppData\Roaming\Mathematica\Applications\AceFEM\Elements\
    ExamplesSED3H1DFHYH1DNeoHookeWA.W64.dll,
  ExamplesSED3H1DFHYH1DNeoHookeWA, 0}, {E * → 21000, ν * → 0.3}, H1,
 {C:\Users\jkorelc\Desktop\SMS\Documentation\ExamplesPeriodic3DStructured.W64.dll,
  ExamplesPeriodic3DStructured, 0}, , {MinSubSteps → 1, MaxSubSteps → 10}}
```

- Create micro mesh and return augmented *local_problem_data* structure.

In[229]:=
```
localProblemData =
 FE23DMicroMeshVoids[localProblemData, {"FE23DMicroMeshVoids" -> specificMacroData}]
```

```
{{FE^2, FE23DMicroMeshVoids, FE2SolveOne, 8,
   Deformation gradient (3D), 9, Integrated stress and sensitivity (P, 3D), 90},
 {0.25359, 0.25359, 0.25359}, {1, 1, 1, 0, 4, 4,
  {C:\Users\jkorelc\AppData\Roaming\Mathematica\Applications\AceFEM\Elements\
     ExamplesSED3H1DFHYH1DNeoHookeWA.W64.dll,
   ExamplesSED3H1DFHYH1DNeoHookeWA, 0}, {E * → 21000, ν * → 0.3}, H1,
  {C:\Users\jkorelc\Desktop\SMS\Documentation\ExamplesPeriodic3DStructured.W64.dll,
   ExamplesPeriodic3DStructured, 0}, , {MinSubSteps → 1, MaxSubSteps → 10}},
 1, 1, 1, {1.5, 0.5, 0, 0, 1, 0, 0, 0, 1}, {101, 121, 21, 1, 105, 125, 25, 5},
 1, {MinSubSteps → 1, MaxSubSteps → 10}}
```

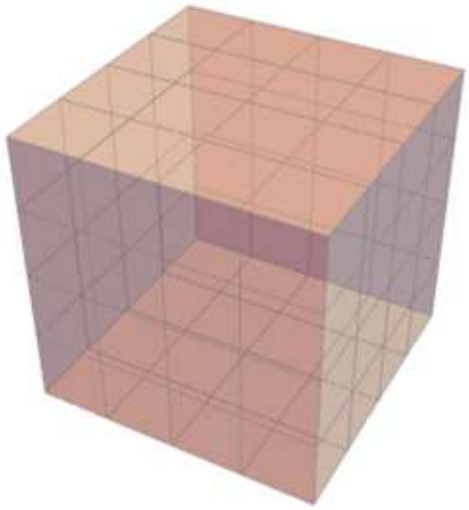
- Check for message created during mesh generation phase.

In[230]:=
```
SMTSharedStatus
```

```
{0, 0}
```

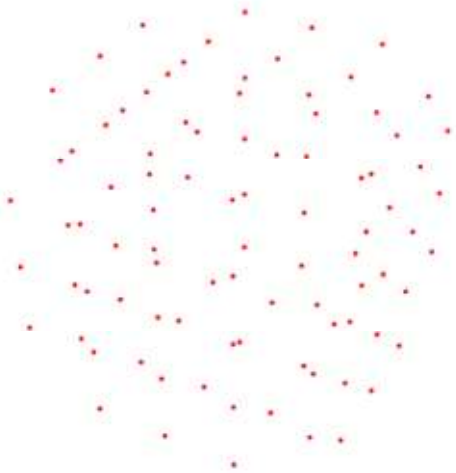
- Solid micro mesh.

In[231]:=
```
SMTShowMesh["Opacity" → 0.4, "ZoomElements" → "micro"]
```



- Mesh of elements used to enforce periodic boundary conditions.

In[232]:=
```
SMTShowMesh["ZoomElements" → "periodic"]
```



- turn on debugging

In[233]:=
```
SMTMacroProblemStatus[[14]] = -1;
```

- check first and last 10 eigenvalues of global tangent matrix at the zero state

In[234]:=
```
mat = SMTData["TangentMatrix"];
{Eigenvalues[mat, 10], Eigenvalues[mat, -10]} // Chop
```

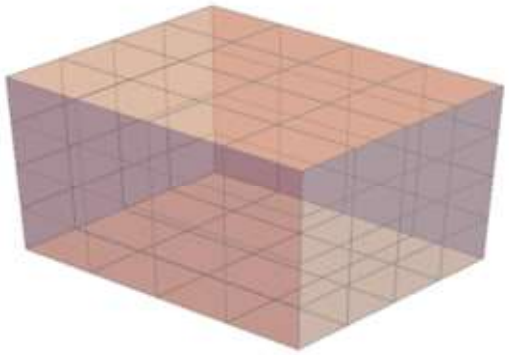
```
{{22 613.7, 22 613.7, 22 613.7, 19 147.4, 19 147.4, 18 614.1, 18 614.1, 17 772.2, 17 111.1,
  16 399.8}, {-0.000298076, -0.000272372, -0.000272372, -0.000269283, -0.000267699,
  -0.000267699, -0.000265043, -0.000210902, -0.000204179, -0.000204179}}
```

- Solve micro problem for given **F** and given mesh. Function returns *micro_data/Volume* where *micro_data* = $\int_{\Omega_e} \text{Flatten}\left[\left\{\text{vec}(P), \frac{\partial \text{vec}(P)}{\partial \text{vec}(F_M)}\right\}\right] dV$.

In[236]:=
```
FE2SolveOne[localProblemData, False] // Chop
```

```
{11778.8, 4038.46, 0, 168.269, 7572.12, 0, 0, 0, 7572.12, 20416.7, 0, 0, -6169.87,
 18173.1, 0, 0, 0, 18173.1, 0, 8076.92, 0, 336.538, 0, 0, 0, 0, 0, 0, 0, 8076.92, 0,
 0, 0, 336.538, 0, 0, -6169.87, 336.538, 0, 11161.9, -9254.81, 0, 0, 0, -9086.54,
 18173.1, 0, 0, -9254.81, 35841.3, 0, 0, 0, 27259.6, 0, 0, 0, 0, 0, 8076.92,
 -168.269, 504.808, 0, 0, 0, 336.538, 0, 0, -168.269, 8076.92, 0, 0, 0, 0, 0, 0,
 0, 504.808, 0, 8076.92, 0, 18173.1, 0, 0, -9086.54, 27259.6, 0, 0, 0, 35841.3}
```

In[237]:=
```
SMTShowMesh["Opacity" → 0.4, "DeformedMesh" → True, "ZoomElements" → "micro"]
```



- check first and last 10 eigenvalues of global tangent matrix at the end of first step

In[238]:=
```
mat = SMTData["TangentMatrix"];
{Eigenvalues[mat, 10], Eigenvalues[mat, -10]} // Chop
```

```
{{28814.4, 28608.6, 24737.2, 24238.2, 23811., 23581.2, 22107.6, 21649.1, 21567.8,
  21484.1}, {-0.000258607, -0.000238905, -0.000230419, -0.000228938, -0.000225994,
  -0.000216494, -0.000200395, -0.000175547, -0.000164501, -0.000152355}}
```

2D MIEL micro structure

- Solve micro structure for given nodal displacements

In[240]:=
```
SMTSpatialDimension = 2;
SMTMultiScaleSet["TestSingleScale" → True, "Threads" → 1, "Console" → True];
```

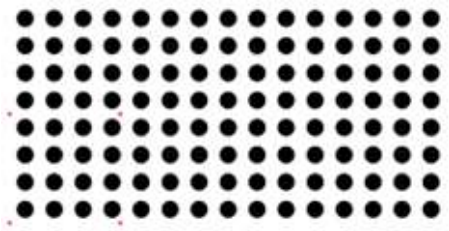
- The actual micro mesh is generated by the user defined MIEL2DMicroMeshVoids[localProblemData, specificMacroData] function where localProblemData is a standard local data structure (see documentation) and specificMacroData is an arbitrary user defined data needed by the MIEL2DMicroMeshVoids function in order to create micro problem mesh.
- local_data data structure
  - Prepare local_data data structure accordingly to documentation.
  - Proper local_data data structure of selected micro problem is also stored after the SMTMicroRestart[selected_micro_point] command in SMTCurrentLocalProblem global variable (see also Visualize selected micro problem section).

In[242]:=
```
XM = {{0, 0}, {1, 0}, {1, 1}, {0, 1}};
localProblemData = {
   {"MIEL", MIEL2DMicroMeshVoids, MIELSolveOne, 1, "Nodal displacements (2D)",
    0, "Integrated strain energy and derivatives (plane strain)", 0}
   , (*X*){0, 0}
   , 1, 1, 1, 1
   , (*Flatten[u_M]*){0, 0, 0, 0, .1, 0, 0, 0.5}
   , (*X_M*)XM
   , "Q1", {XM} // Transpose, 1, {}, True};
```

- specificMacroData data structure

In[244]:=
```
L = 4.; H = 2.; n1 = 15; n2 = 8; r = 0.08;
disks =
  Flatten[Table[{{L / n1 / 2 + L / n1 i, H / n2 / 2 + H / n2 j}, r}, {i, 0, n1 - 1}, {j, 0, n2 - 1}], 1];
specificMacroData = {disks, 30, SMTMakeDll["ExamplesSEPET1DFHYT1DNeoHookeWA"],
   {"E *" → 21000, "ν *" → 0.3}, "T1", {"MinSubSteps" → 1, "MaxSubSteps" → 10}};
```

In[247]:=
```
Show[Graphics[{Black, Map[Disk[#[[1]], #[[2]]] &, disks], Red, Point[XM]}]]
```



- Create micro mesh and return augmented local_data data structure.

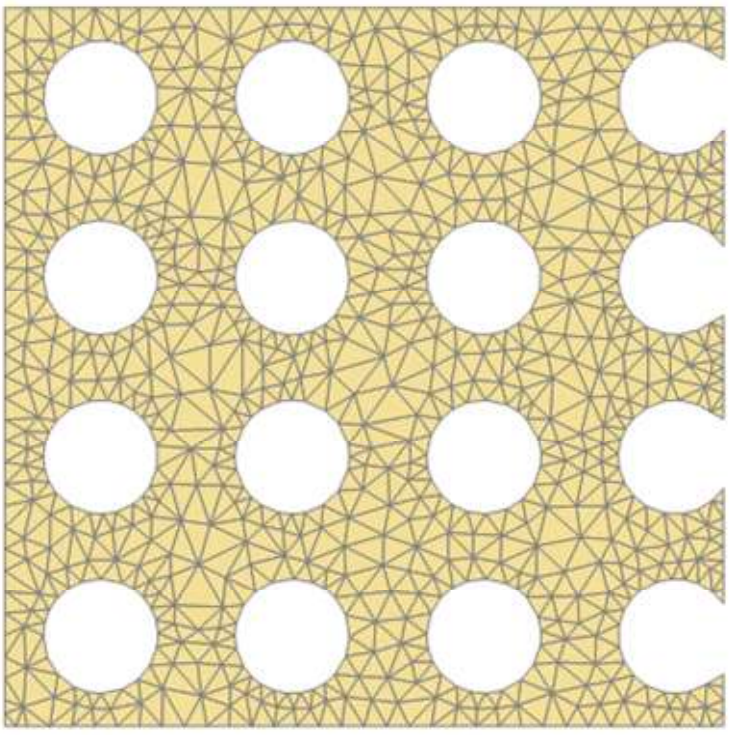
In[248]:=
```
localProblemData =
  MIEL2DMicroMeshVoids[localProblemData, {"MIEL2DMicroMeshVoids" -> specificMacroData}];
```

- Check for message created during mesh generation phase.

In[249]:=
```
SMTSharedStatus
```

Out[249]=
```
{0, 0}
```

In[250]:=
```
SMTShowMesh[]
```



- Solve given macro problem for given macro boundary conditions. Function returns micro_data = $\int_{\Omega_e} \text{Flatten}\left[\left\{W, \frac{\partial W}{\partial u_M}\Big|_{\mathbf{h}=\text{const.}}, \text{vec}\left(\frac{\partial}{\partial u_M}\left(\frac{\partial W}{\partial u_M}\Big|_{\mathbf{h}=\text{const.}}\right)\right)\right\}\right] dV$.

In[251]:=
```
microData = MIELSolveOne[localProblemData, False];
```

In[252]:=
```
NoDOF = 8;
```

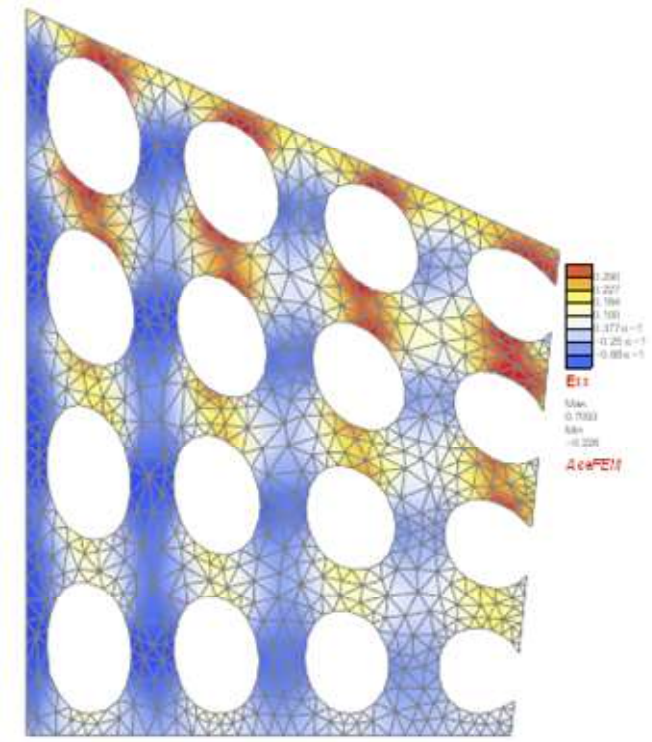
- Get macro strain energy, macro element residual and eigenvalues of macro element tangent matrix

In[253]:=
```
W = microData[[1]]
```
```
674.082
```

In[254]:=
```
R = microData[[2 ;; NoDOF + 1]]
```
```
{-530.218, -1599.68, 822.938, -1419.78, 777.275, 538.235, -1069.99, 2481.22}
```

In[255]:=
```
K = SMTToSymmetricOrUnsymmetric[False, NoDOF, Drop[microData, NoDOF + 1]] // Eigenvalues // Chop
```
```
{13256.8, 7068.64, 5974.3, 4602.19, 3730.55, 2005.1, 0, 0}
```

In[256]:=
```
SMTShowMesh["DeformedMesh" → True, "Field" -> "Exx"]
```



3D MIEL micro structure

- Solve micro structure for given nodal displacements

In[257]:=
```
SMTSpatialDimension = 3;
SMTMultiScaleSet["TestSingleScale" → True, "Threads" → 1, "Console" → True];
```

- The actual micro mesh is generated by the user defined MIEL3DMicroMeshVoids[localProblemData, specificMacroData] function where localProblemData is a standard local data structure (see documentation) and specificMacroData is an arbitrary user defined data needed by the MIEL2DMicroMeshVoids function in order to create micro problem mesh.

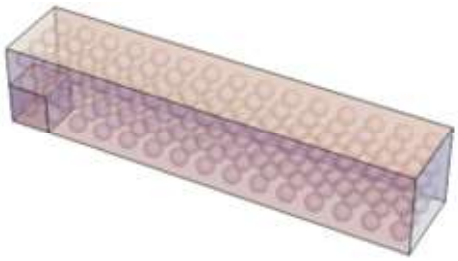
In[259]:=
```
XM = {{0.`, 0.`, 0.`}, {1.1999999999999993`, 0.`, 0.`},
    {1.1999999999999993`, 1.2`, 0.`}, {0.`, 1.2`, 0.`}, {0.`, 0.`, 1.2`},
    {1.1999999999999993`, 0.`, 1.2`}, {1.1999999999999993`, 1.2`, 1.2`}, {0.`, 1.2`, 1.2`}};
localProblemData = {
    {"MIEL", MIEL3DMicroMeshVoids, MIELSolveOne, 1,
     "Nodal displacements (3D)", 0, "Integrated strain energy and derivatives (3D)", 0}
    , (*X*){0, 0, 0}
    , 1, 1, 1, 1
    , (*Flatten[u_M]*)
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.2, 0, -0.1, 0, 0.2, 0, 0, 0.2, 0, 0, 0, 0}
    , (*X_M*)XM
    , "H1", {XM} // Transpose, 1, {}, True};
```

- specific_micro_data={
  - 1 {{{x1,y1,z1},r1},{{x2,y2,z2},r2},…} - all spheres where {xi,yi,zi} is a center of the i-th voids and ri is the radius of the i-th void
  - 2 division - approximate number of elements per side
  - 3 micro_element - SMTMakeDll[micro_element_UEC] - element used to discretize the solid domain
  - 4 micro_material - material data related to micro element (see `SMTAddDomain`)
  - 5 topology - micro mesh type (see `Element Topology`)
  - 6 list of solution procedure options - given options are stored in *local_problem_data[[10]]* and interpreted later by the MIELSolveOne function. Current options are:
  - "MinSubSteps"->*n* - minimum number of micro sub-steps per one macro step (default *n*=1)
  - "MaxSubSteps"->*n* ⇒ defines a maximum number of micro sub-steps per macro step (default 100)
  - }

In[261]:=
```
L = 12; B = 2.4; H = 2.4;
```

In[262]:=
```
(* number of spheres in each direction *)
nxC = 15; nyC = 4; nzC = 3;
aCell = L / nxC; bCell = B / nyC; cCell = H / nzC;
pVoids = 0.2;(* percentage of voids*)
rCell = CubeRoot[(pVoids aCell bCell cCell 3) / (π 4)];(* radius of spheres*)
spheres = If[pVoids === 0, {},
   Flatten[Table[{{aCell / 2 + aCell i, bCell / 2 + bCell j, cCell / 2 + cCell k}, rCell},
      {i, 0, nxC - 1}, {j, 0, nyC - 1}, {k, 0, nzC - 1}], 2] // N];
meshDensity = 5;(*number of micro elements per side of macro eleemnt *)
```

In[268]:=
```
Graphics3D[{Opacity[0.3], Hexahedron[XM], Cuboid[{0, 0, 0}, {L, B, H}],
  Map[Sphere[#[[1]], #[[2]]] &, spheres]}, Boxed → False]
```



In[269]:=
```
Clear[specificMacroData]
specificMacroData = {spheres, meshDensity, SMTMakeDll["ExamplesSED3O1DFHYO1DNeoHookeWA"],
   {"E *" → 21000, "ν *" → 0.3}, "O1", {"MinSubSteps" → 1, "MaxSubSteps" → 10}};
```

- Create micro mesh and return augmented local_data data structure.

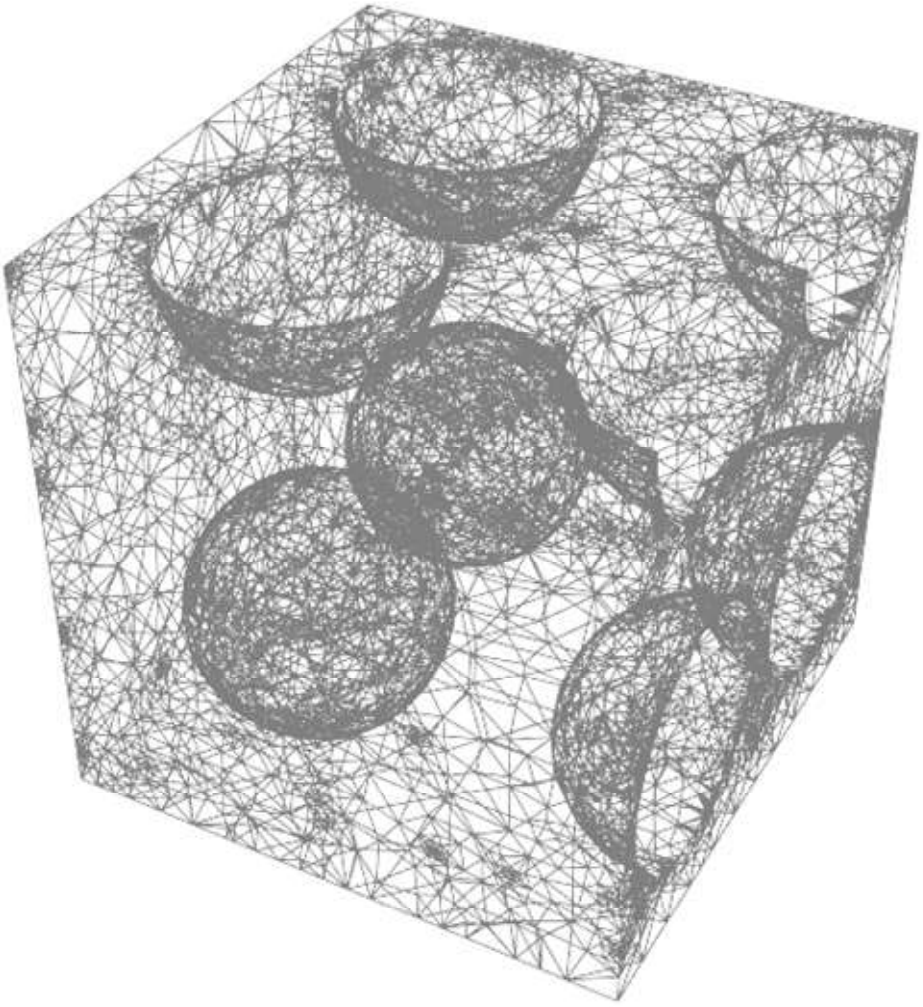
In[271]:=
```
localProblemData =
  MIEL3DMicroMeshVoids[localProblemData, {"MIEL3DMicroMeshVoids" -> specificMacroData}];
```

- Check for message created during mesh generation phase.

In[272]:=
```
SMTSharedStatus
```

Out[272]=
```
{0, 0}
```

In[273]:=
```
SMTShowMesh["FillElements" → False]
```



- Solve given macro problem for given macro boundary conditions. Function returns micro_data = $\int_{\Omega_e} \text{Flatten}\left[\left\{W, \frac{\partial W}{\partial u_M}\Big|_{\mathbf{h}=\text{const.}}, \text{vec}\left(\frac{\partial}{\partial u_M}\left(\frac{\partial W}{\partial u_M}\Big|_{\mathbf{h}=\text{const.}}\right)\right)\right\}\right] dV$.

In[274]:=
```
microData = MIELSolveOne[localProblemData, False];
```

In[275]:=
```
NoDOF = 24;
```

- Get macro strain energy, macro element residual and eigenvalues of macro element tangent matrix

In[276]:=
```
W = microData[[1]]
```

Out[276]=
```
177.831
```

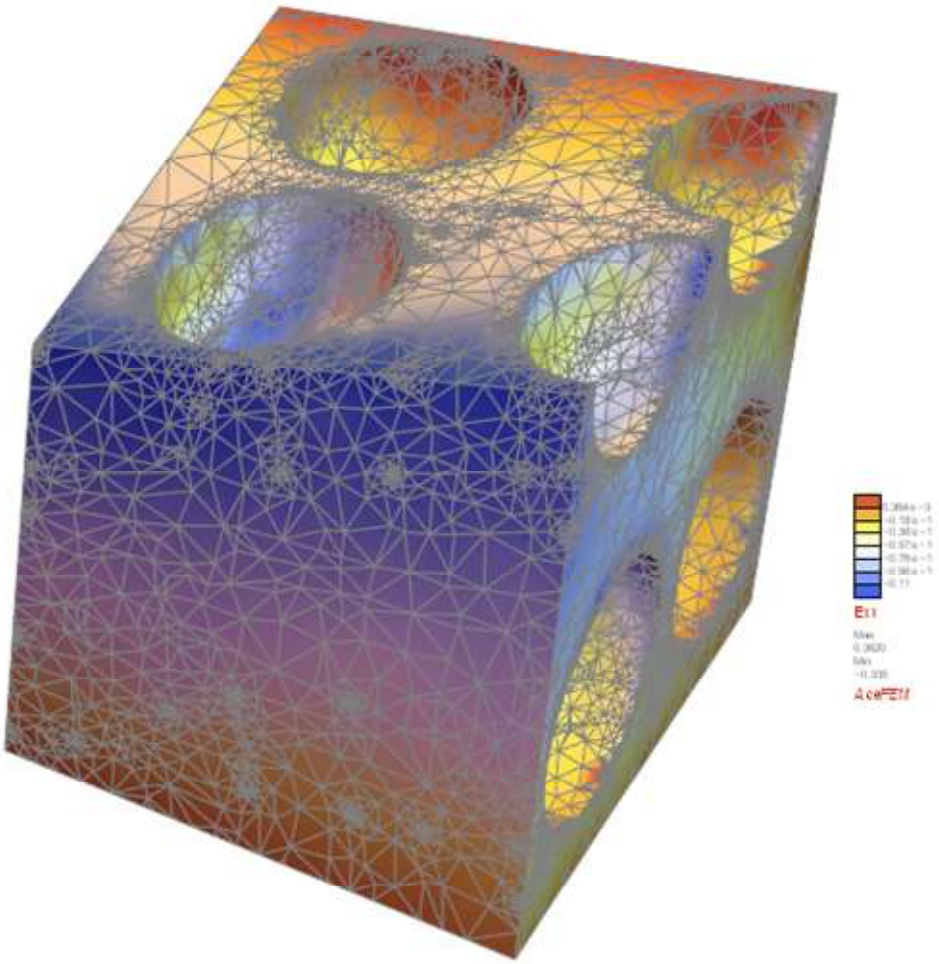
In[277]:=
```
R = microData[[2 ;; NoDOF + 1]]
```

```
{54.497, -86.4403, 37.7966, -561.366, -86.0462, 169.581, -161.156, -344.794,
 508.748, 143.578, -377.834, 280.082, 638.718, 240.595, -792.681, -563.617,
 501.03, -302.26, -33.7823, 221.913, 227.264, 483.128, -68.4237, -128.531}
```

In[278]:=
```
K = SMTToSymmetricOrUnsymmetric[False, NoDOF, Drop[microData, NoDOF + 1]] // Eigenvalues // Chop
```

```
{20362.5, 7844.54, 7727.52, 7414.86, 7263.71, 7038.65, 5401.06, 5102.44,
 4956.43, 4754.96, 3981.69, 3817.35, 3730.13, 2115.21, 1976.96, 1884.43, 1154.37,
 1083.83, -726.516, -317.729, -217.141, 1.00321 × 10^-8, 1.38992 × 10^-9, 0}
```

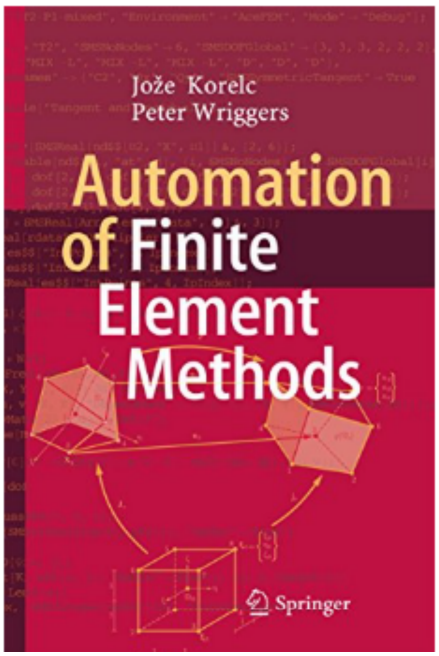
In[279]:=
```
SMTShowMesh["DeformedMesh" → True, "Field" -> "Exx"]
```



CHAPTER 8

Appendix

Bibliography



AceGen

KORELC, J. Automation of primal and sensitivity analysis of transient coupled problems. Computational mechanics, 44(5):631-649 (2009).

KORELC, Jože, Multi-language and Multi-environment Generation of Nonlinear Finite Element Codes, Engineering with Computers, 2002, 18(4):312-327

KORELC, Jože. Automatic generation of finite-element code by simultaneous optimization of expressions. Theor. comput. sci.. , 1997, 187:231-248.

HUDOBIVNIK, Blaž, KORELC, Jože. Closed-form representation o fmatrix functions in the formulation of nonlinear material models. Finite Elements in Analysis and Design, 2016, 111:19-32

KORELC, J. Semi-analytical solution of path-independent nonlinear finite element models. Finite elem. anal. des., 2011, 47:281-287.

KORELC, J. Automation of the finite element method. V: WRIGGERS, Peter. Nonlinear finite element methods. Springer, 483-508 (2008).

Applications

ŠOLINC, Urša, KORELC, Jože. A simple way to improved formulation of FE2 analysis. Computational mechanics, 2015, 56:905-915, doi: 10.1007/s00466-015-1208-4.

MELINK, Teja, KORELC, Jože. Stability of Karhunen- Loève expansion for the simulation of Gaussian stochastic fields using Galerkin scheme. Probabilistic Engineering Mechanics, 2014, 37:7-15, doi: 10.1016/j.probengmech.2014.03.006.

KORELC, Jože, STUPKIEWICZ, Stanislaw. Closed-form matrix exponential and its application in finite-strain plasticity. International journal for numerical methods in engineering, ISSN 0029-5981, 2014, 98(13):960-987, ilustr., doi: 10.1002/nme.4653.

CLASEN, Heiko, HIRSCHBERGER, C. Britta, KORELC, Jože, WRIGGERS, Peter, FE2-homogenization of micromorphic elasto-plastic materials, XII International Conference on Computational Plasticity. Fundamentals and Applications, COMPLAS XII, 2013

LENGIEWICZ, Jakub, KORELC, Joze, STUPKIEWICZ, Stanislaw., Automation of finite element formulations for large deformation contact problems. Int. j. numer. methods eng., 2011, 85: 1252-1279.

KORELC, J. Direct computation of critical points based on Crout`s elimination and diagonal subset test function, Computers and Structures, 88:189-197 (2010).

WRIGGERS, Peter, KRSTULOVIC-OPARA, Lovre, KORELC, Jože.(2001), Smooth C1-interpolations for two-dimensional frictional contact problems. *Int. j. numer. methods eng.*, 2001, vol. 51, issue 12, str. 1469-1495

KRSTULOVIC-OPARA, Lovre, WRIGGERS, Peter, KORELC, Jože. (2002), A C1-continuous formulation for 3D finite deformation frictional contact. *Comput. mech.*, vol. 29, issue 1, 27-42

STUPKIEWICZ, Stanislaw, KORELC, Jože, DUTKO, Martin, RODIC, Tomaž. (2002), Shape sensitivity analysis of large deformation frictional contact problems. *Comput. methods appl. mech. eng.*, 2002, vol. 191, issue 33, 3555-3581

BRANK, Boštjan, KORELC, Jože, IBRAHIMBEGOVIC, Adnan. (2002), Nonlinear shell problem formulation accounting for through-the-tickness stretching and its finite element implementation. *Comput. struct.*. vol. 80, n. 9/10, 699-717

BRANK, Boštjan, KORELC, Jože, IBRAHIMBEGOVIC, Adnan. (2003),Dynamic and time-stepping schemes for elastic shells undergoing finite rotations. *Comput. struct.*, vol. 81, issue 12, 1193-1210

STADLER, Michael, HOLZAPFEL, Gerhard A., KORELC, Jože. (2003) Cn continuous modelling of smooth contact surfaces using NURBS and application to 2D problems. *Int. j. numer. methods eng.*, 2177-2203

KUNC, Robert, PREBIL, Ivan, RODIC, Tomaž, KORELC, Jože. (2002),Low cycle elastoplastic properties of normalised and tempered 42CrMo4 steel. *Mater. sci. technol.*, Vol. 18, 1363-1368.

Bialas M, Majerus P, Herzog R, Mroz Z, Numerical simulation of segmentation cracking in thermal barrier coatings by means of cohesive zone elements, MATERIALS SCIENCE AND ENGINEERING A-STRUCTURAL MATERIALS PROPERTIES MICROSTRUCTURE AND PROCESSING 412 (1-2): 241-251 Sp. Iss. SI, DEC 5 2005

Maciejewski G, Kret S, Ruterana P, Piezoelectric field around threading dislocation in GaN determined on the basis of high-resolution transmission electron microscopy image , JOURNAL OF MICROSCOPY-OXFORD 223: 212-215 Part 3 SEP 2006

Wisniewski K, Turska E, Enhanced Allman quadrilateral for finite drilling rotations, COMPUTER METHODS IN APPLIED MECHANICS AND ENGINEERING 195 (44-47): 6086-6109 2006

Maciejewski G, Stupkiewicz S, Petryk H, Elastic micro-strain energy at the austenite-twinned martensite interface, ARCHIVES OF MECHANICS 57 (4): 277-297 2005

Stupkiewicz S, The effect of stacking fault energy on the formation of stress-induced internally faulted martensite plates, EUROPEAN JOURNAL OF MECHANICS A-SOLIDS 23 (1): 107-126 JAN-FEB 2004

Symbolic methods

Korelc J. (1997b), A symbolic system for cooperative problem solving in computational mechanics, *Computational Plasticity Fundamentals and Applications*, (Owen D.R.J., Oñate E. and Hinton E., editors), CIMNE, Barcelona, 447-451.

Korelc J., and Wriggers P. (1997c), Symbolic approach in computational mechanics, *Computational Plasticity Fundamentals and Applications*, (Owen D.R.J., Oñate E. and Hinton E., editors), CIMNE, Barcelona, 286-304.

Korelc J., (2001), Hybrid system for multi-language and multi-environment generation of numerical codes, Proceedings of the ISSAC'2001 Symposium on Symbolic and Algebraic Computation, New York, ACM:Press, 209-216

Korelc, J. (2003) Automatic Generation of Numerical Code. MITIC, Peter. Challenging the boundaries of symbolic computation : proceedings of the 5th International Mathematica Symposium. London: Imperial College Press, 9-16.

Gonnet G. (1986), New results for random determination of equivalence of expression, Proc. of 1986 ACM Symp. on Symbolic and Algebraic Comp, (Char B.W., editor), Waterloo, July 1986, 127-131.

Griewank A. (1989), On Automatic Differentiation, *Mathematical Programming: Recent Developments and Applications*, Kluwer Academic Publisher, Amsterdam, 83-108.

Hulzen J.A. (1983), Code optimization of multivariate polynomial schemes: A pragmatic approach. Proc. of IEUROCAL'83, (Hulzen J.A., editor), Springer-Verlag LNCS Series Nr. 162.

Kant E. (1993), Synthesis of Mathematical Modeling Software, *IEEE Software*, May 1993.

Leff L. and Yun D.Y.Y. (1991), The symbolic finite element analysis system. *Computers & Structures*, **41**, 227-231.

Noor A.K. (1994), Computerized Symbolic Manipulation in Structural Mechanics, *Computerized symbolic manipulation in mechanics*, (Kreuzer E., editor), Springer-Verlag, New York, 149-200.

Schwartz J.T. (1980), Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, **27**(4), 701-717.

Sofroniou M. (1993), An efficient symbolic-numeric environment by extending mathematica's format rules. Proceedings of Workshop on Symbolic and Numerical Computation, (Apiola H., editor), University of Helsinki, Technical Report Series, 69-83.

Wang P.S. (1986), Finger: A symbolic system for automatic generation of numerical programs in finite element analysis, *J. Symb. Comput,* **2**, 305-316.

Wang P.S. (1991), Symbolic computation and parallel software, Technical Report ICM-9109-12, Department of Mathematics and Computer Science, Kent State University, USA.

AceFEM Troubleshooting

General

- If the use of `SMSPrint` does not produce any printout or the execution does not stop at the break point (`SMSSetBreak`) check that:
  a) the code was generated in "Debug" mode (SMSInitialize["Mode"->"Debug"]),
  b) debug element has been set (SMTIData["DebugElement",element_number])
  c) the file is opened for printing (SMTAnalysis["Output"->filename]).
- If the compilation is to slow then restrict compiler optimization with SMTAnalysis["OptimizeDll"->False].
- If the quadratic convergence is not achieved check that:
  a) matrix is symmetric or unsymmetrical (by default all elements are assumed to have symmetric tangent matrix, use SMSTemplate["SMSSymmetricMatrix"->False] to specify unsymmetrical matrix),
  b) the tangent matrix is not singular or near singular.
- Problems with linear solver (PARDISO):
  a) not enough memory ⇒ try out-of-core solution (SMTAnalyze[...,Solver->{5,11,{{60,2}}}]
  b) zero pivot, numerical factorization problem ⇒ try full pivoting (SMTAnalyze[...,Solver->{5,11}])
- Check the information given at http://symech.fgg.uni-lj.si/FAQ/.
- Printing from finite element user subroutines might be problematic when the program is parallelized. Please turn off parallelization with SMTInputData["Threads" -> 1] command.
- When the CDriver is run as console application (SMTInputData["Console"→True]) useful additional information is printed out to console window.
- Any C print statement (e.g. printf("hello")) inserted directly into the element C source file will appear in console window (SMTInputData["Console"→True]).
- Additional useful information is also printed out to output file when defined (SMTAnalysis["Output"→*filename*]).
- When the code is generated for user defined environment, always first check it in AceFEM. The sophisticated debugging options available in AceFEM are usually not available in standard FEM environments.
- `Slow code, check Numerical efficiency of AceFEM data manipulations.`

Crash of the CDriver module

- Block parallelization (SMTInputData["Threads"->1])
- Recreate elements in Debug mode (SMSInitialize["Debug"->True]) and run the example again.
- Run CDriver module in console mode (SMTInputData["Console"->True]) and examine the output.