

AceFEM

Mathematica Finite
Element Environment

Jože Korelc

University of Ljubljana

Faculty of Civil and Geodetic
Engineering

2009

AceFEM Contents

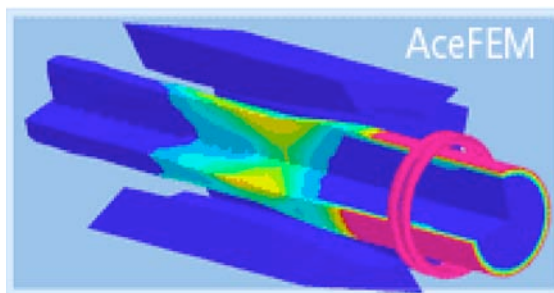
AceFEM Contents	3
AceFEM Tutorials	19
AceFEM Preface	19
• Acknowledgement	
AceFEM Structure	26
AceFEM Palettes	38
Standard AceFEM Procedure	43
Input Data	56
Analysis Phase	63
Iterative solution procedure	65
Data Base Manipulations	74
Selecting Nodes	76
Selecting Elements	80
Interactive Debugging	81
Code Profiling	96
Implementation Notes for Contact Elements	107
Semi-analytical solutions	123
User Defined Tasks	140
Parallel AceFEM computations	159
Summary of Examples	168
• Basic AceFEM Examples • Basic AceGen-AceFEM Examples • Advanced Examples • Examples of Contact Formulations • Implementation of Finite Elements in Alternative Numerical Environments	
Bibliography	174
Shared Finite Element Libraries	179
AceShare	179
Accessing elements from shared libraries	182
Unified Element Code	192
Simple AceShare library	194
Advanced AceShare library	202
Basic AceFEM Examples	222
Bending of the column (path following procedure, animations, 2D solids)	223
Boundary conditions (2D solid)	231
Solution 2: based on general node selector and calculated nodal forces	240
Standard 6-element benchmark test for distortion sensitivity (2D solids)	247
Solution Convergence Test	251
Postprocessing (3D heat conduction)	259
Basic AceGen-AceFEM Examples	274
Simple 2D Solid, Finite Strain Element	275
Mixed 3D Solid FE, Elimination of Local Unknowns	285
Mixed 3D Solid FE, Auxiliary Nodes	305
Cubic triangle, Additional nodes	329

Inflating the Tyre	342
Advanced Examples	357
Round-off Error Test	358
Solid, Finite Strain Element for Direct and Sensitivity Analysis	373
Parameter, Shape and Load Sensitivity Analysis of Multi-Domain Example	402
Three Dimensional, Elasto-Plastic Element	420
Axisymmetric, finite strain elasto-plastic element	453
Cyclic tension test, advanced post-processing , animations	476
Solid, Finite Strain Element for Dynamic Analysis	500
Elements that Call User External Subroutines	510
Examples of Contact Formulations	531
2D slave node, line master segment element	532
2D indentation problem	545
2D slave node, smooth master segment element	553
2D snooker simulation	575
3D slave node, triangle master segment element	585
3D slave node, quadrilateral master segment element	594
3D slave node, quadrilateral master segment and 2 neighboring nodes element	605
3D slave triangle and 2 neighboring nodes, triangle master segment element	616
3D slave triangle, triangle master segment and 2 neighboring nodes element	629
3D contact analysis	641
Troubleshooting and New in version	647
AceFEM Troubleshooting	647
New in version	650
Advanced User Documentation	653
Mesh Input Data Structures	653
Sensitivity Input Data Structures	666
Reference Guide	669
Description of Problem	669
SMTInputData	669
SMTAddDomain	673
SMTAddMesh	675
SMTAddElement	677
SMTAddNode	681
SMTAddEssentialBoundary ...	682
SMTAddNaturalBoundary	686
SMTAddInitialBoundary	687
SMTMesh	687
SMTAddSensitivity	708
Analysis	715
SMTAnalysis .	715
SMTNewtonIteration .	719
SMTNextStep	720
SMTStepBack	722

SMTConvergence	723
SMTDump	729
SMTRestart	742
SMTSensitivity	742
SMTTask	744
SMTStatusReport	744
SMTSessionTime	745
SMTErrorCheck	746
SMTSimulationReport	749
Postprocessing	750
SMTShowMesh	750
SMTMakeAnimation .	768
SMTResidual .	769
SMTPostData .	771
SMTData	772
SMTPut	776
SMTGet	787
SMTSave	788
Data Base Manipulations	789
SMTIData	789
SMTRData	791
SMTNodeData	791
• Interpreted nodal data	
SMTNodeSpecData	795
SMTElementData	796
SMTDomainData	798
SMTFindNodes	800
SMTFindElements	800
SMTSetSolver	801
• References	
Shared Finite Element Libraries	807
SMTSetLibrary	807
SMTAddToLibrary	809
SMTLibraryContents .	812
Utilities	813
SMTScanedDiagramToTable .	813
SMTPost	819
SMTPointValues	822
SMTMakeDll .	823

AceFEM Tutorials

AceFEM Preface



AceFEM

© **Prof. Dr. Jože Korelc**, 2006, 2007, 2008, 2009, 2010
Ravnikova 4, SI - 1000, Ljubljana, Slovenia
E-mail : AceProducts@fgg.uni-lj.si
www.fgg.uni-lj.si/Symech/

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 Ace-

Gen the AceFem package represents an ideal tool for a rapid development of new numerical models.

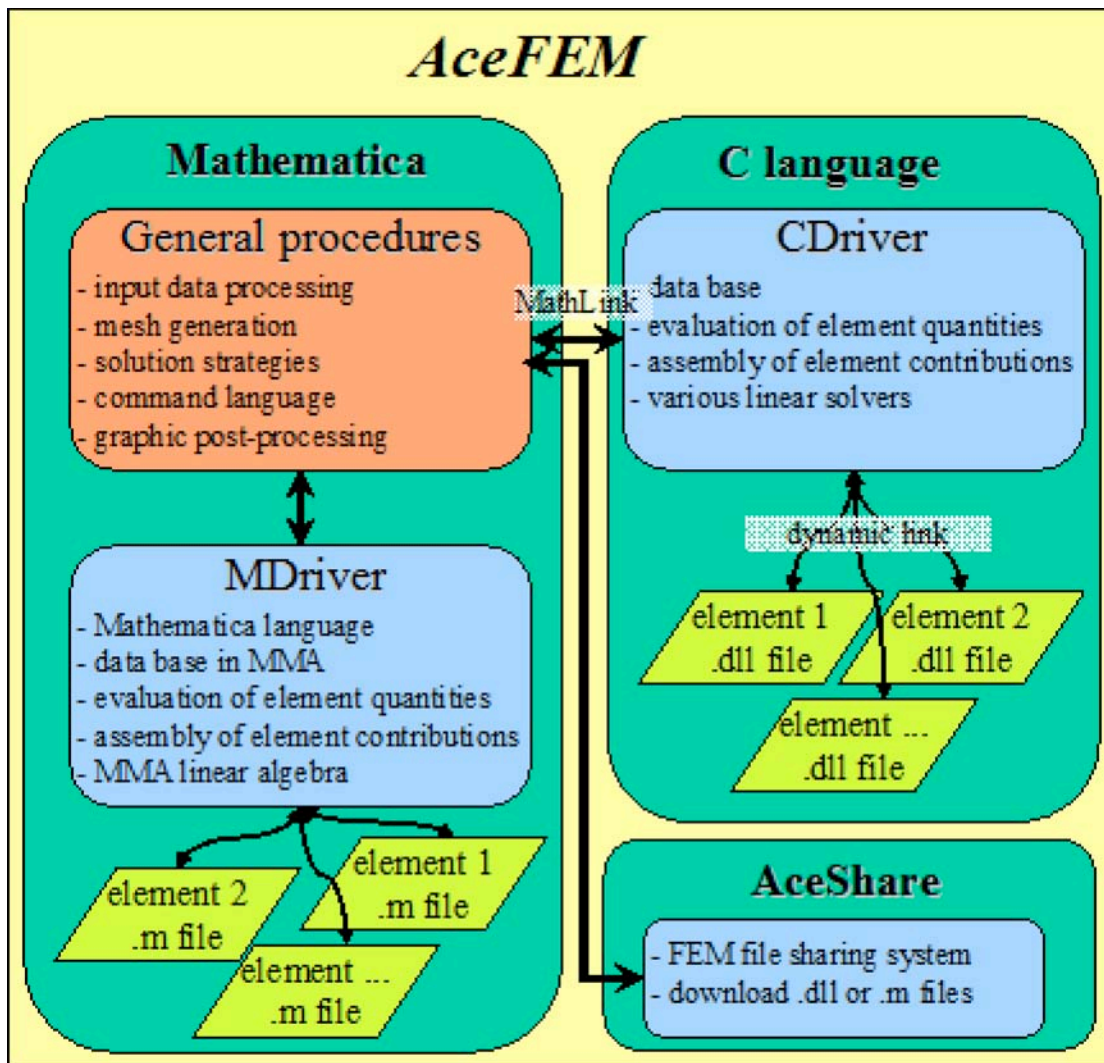
Acknowledgement

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.

AceFEM Structure

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 (Accessing elements from shared libraries).

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.

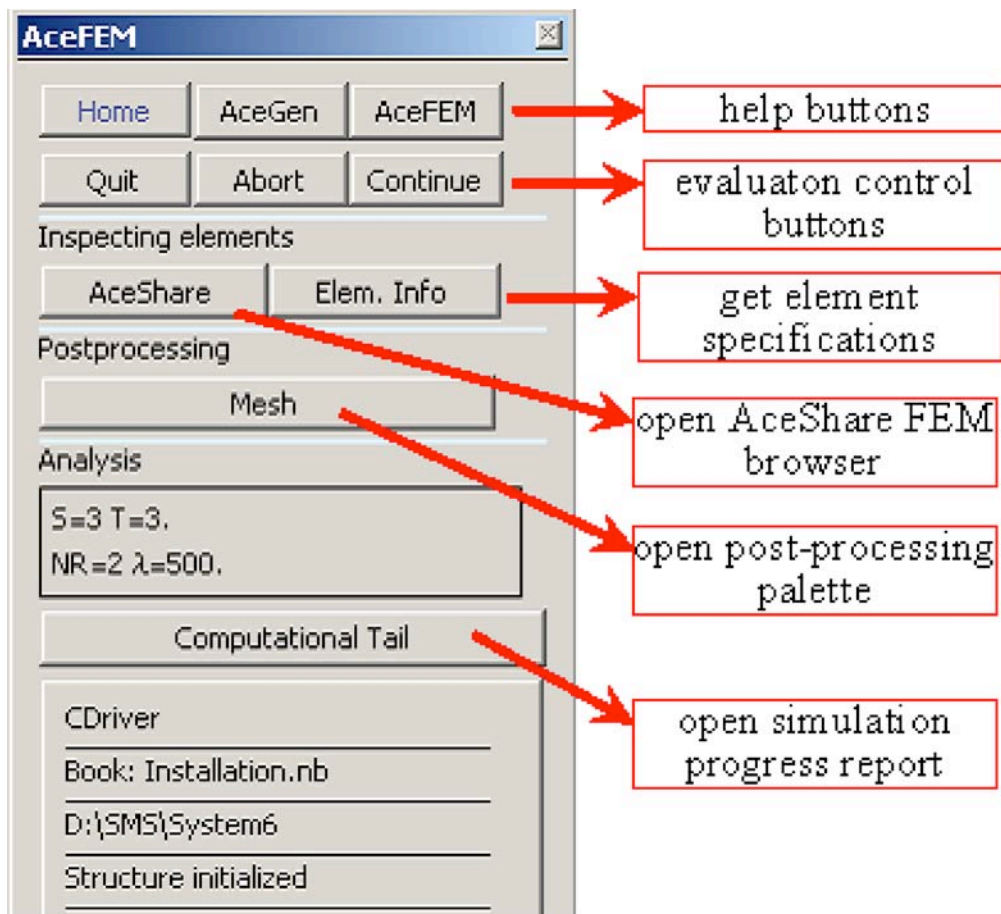
The *AceFEM* package has also some basic pre-processing and post-processing functions (*SMTMesh*, *SMTShowMesh*). It can be used for the geometries that can be discretized by the structured meshes of arbitrary shape. For a more complex geometries the commercial pre/post-processor has to be used. The *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 downloadable 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).

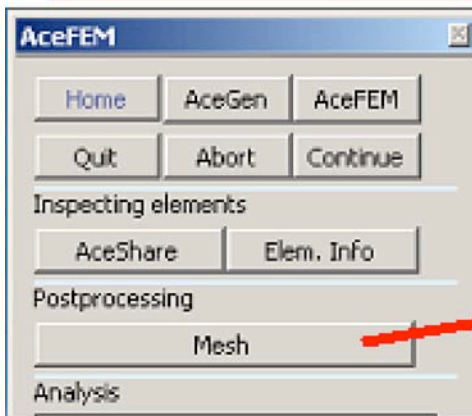
AceFEM Palettes

Main AceFEM palette.

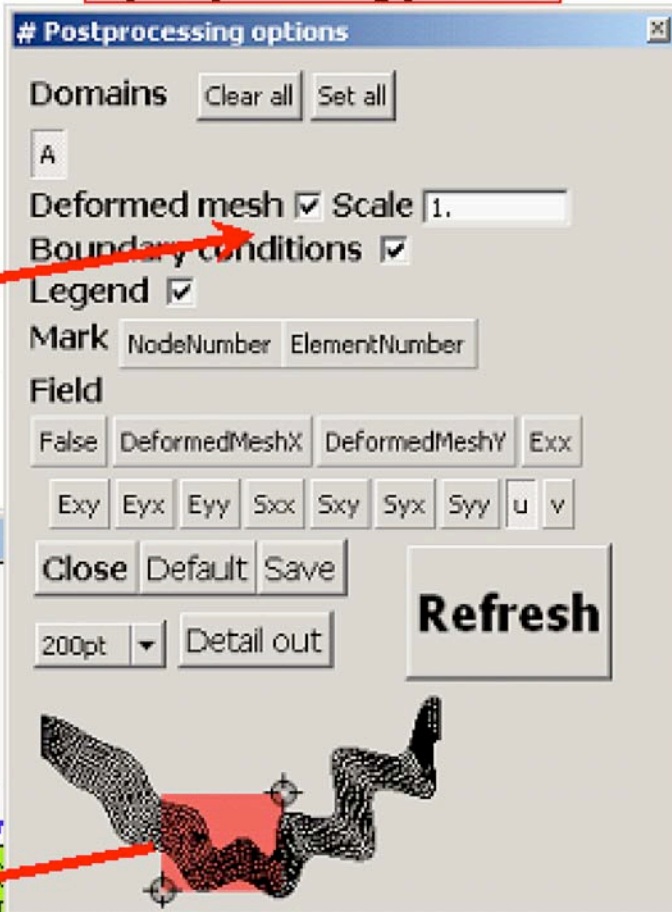


Post-processing palette and mesh display windows.

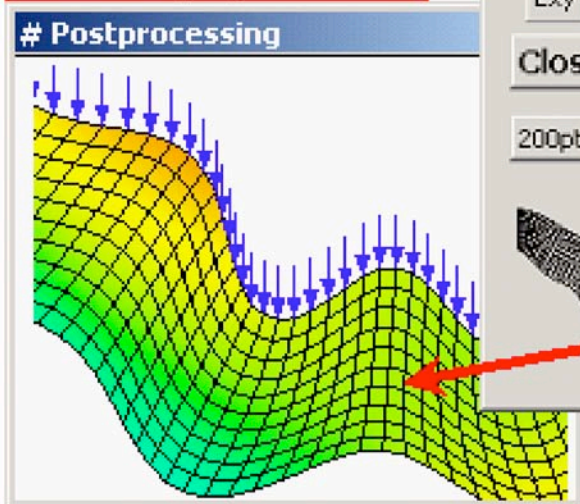
main AceFEM palette



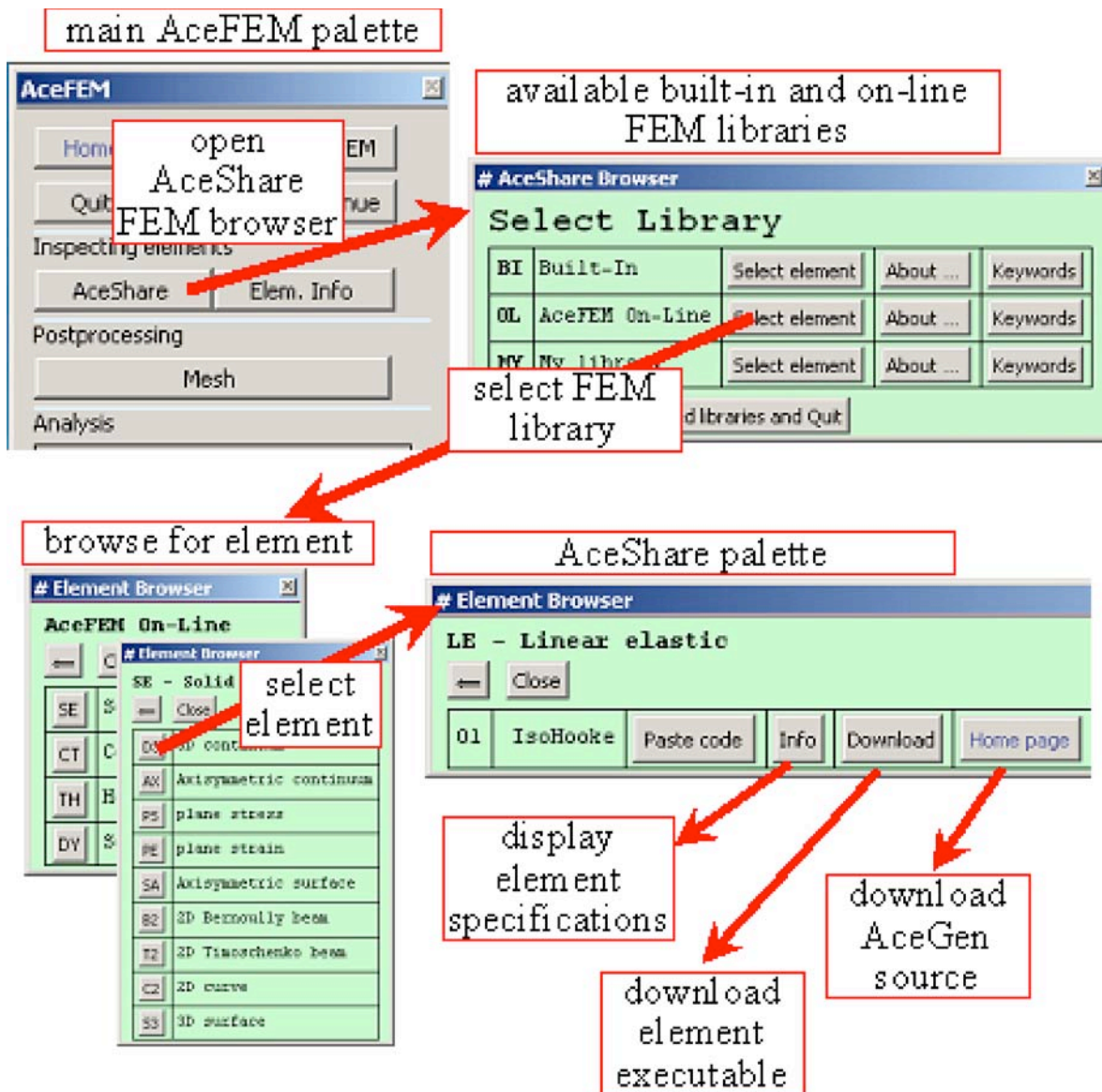
post-processing palette



mesh display window



AceShare FEM browser.



Standard AceFEM Procedure

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

A) Input data phase

- phase starts with `SMTInputData`
- mesh input data (Input Data)
- element description (Input Data, the actual element codes have to be generated before the analysis by *AceGen* code generator)
- sensitivity input data (`SMTAddSensitivity`)

B) Analysis phase

- phase starts with `SMTAnalysis`
- solution procedure is executed accordingly to the *Mathematica* input given by the user (`SMTConvergence`)
- *AceFEM* is designed to solve steady-state or transient finite element and related problems implicitly by means of Newton-Raphson type procedures
- post-processing of the results can be part of the analysis (`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.

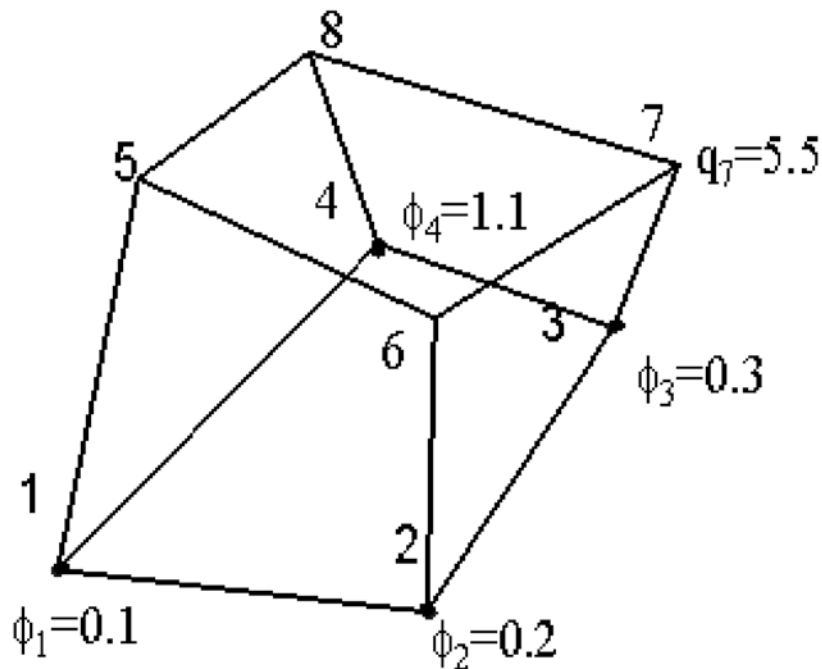
This loads the *AceFEM* package, and prepares input data structures and starts input data section

```
<< 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 $Q = 1$.

```
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.



```
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*.

```
SMTAnalysis[];
```

Here the real time and the value of the boundary condition and the heat source multiplier are prescribed. The problem is steady-state so that the real time in this case has no meaning.

```
SMTNextStep[1, 1];
```

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

```

While[SMTConvergence[10^-12, 10], SMTNewtonIteration[];
  SMTStatusReport[SMTPostData["Temperature", {-0.5, 0, -0.5}]]];

T/ΔT=1./1. λ/Δλ=1./1. ||Δa||/||Ψ||=1.10033
/3.66682 Iter/Total=1/1 Status=0/{ } Tag=0.1

T/ΔT=1./1. λ/Δλ=1./1. ||Δa||/||Ψ||=0.0370544
/0.142921 Iter/Total=2/2 Status=0/{ } Tag=0.1

T/ΔT=1./1. λ/Δλ=1./1. ||Δa||/||Ψ||=0.0000510625
/0.000203298 Iter/Total=3/3 Status=0/{ } Tag=0.1

T/ΔT=1./1. λ/Δλ=1./1. ||Δa||/||Ψ||=8.83286 × 10^-11
/3.5599 × 10^-10 Iter/Total=4/4 Status=0/{ } Tag=0.1

T/ΔT=1./1. λ/Δλ=1./1. ||Δa||/||Ψ||=1.14193 × 10^-16
/4.57125 × 10^-16 Iter/Total=5/5 Status=0/{ } Tag=0.1

```

During the analysis we have all the time full access to all environment, nodal and element data. They can be accessed and changed with the data manipulation commands (see Data Base Manipulations). This gives to *AceFEM* flexibility that is not shared by other FE environments.

Here additional step is made, however instead of increasing time or multiplier, the temperature in node 3 is set to 1.5.

```

SMTNextStep[0, 0];
SMTNodeData[3, "Bp", {1.5}];
While[SMTConvergence[10^-12, 10], SMTNewtonIteration[]];
SMTStatusReport[SMTPostData["Temperature", {-0.5, 0, -0.5}]]];

T/ΔT=1./0. λ/Δλ=1./0. ||Δa||/||Ψ||=7.86125 × 10^-15 /
3.91155 × 10^-14 Iter/Total=5/10 Status=0/{Convergence} Tag=0.1

```

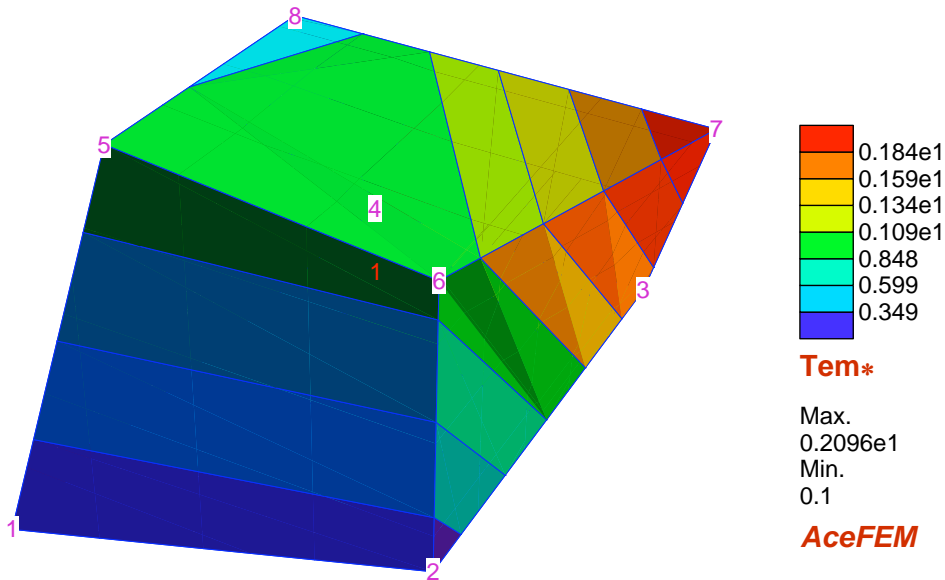
The SMTSimulationReport command displays vital characteristics of the analysis performed.

SMTSimulationReport [] ;

No. of nodes	8
No. of elements	1
No. of equations	4
Data memory (KBytes)	2
Number of threads used/max	8/8
Solver memory (KBytes)	16
No. of steps	2
No. of steps back	0
Step efficiency (%)	100.
Total no. of iterations	10
Average iterations/step	5.
Total time (s)	0.173
Total linear solver time (s)	0.
Total linear solver time (%)	0.
Total assembly time (s)	0.
Total assembly time (%)	0.
Average time/iteration (s)	0.0173
Average linear solver time (s)	0.
Average K and R time (s)	0.
Total Mathematica time (s)	0.032
Total Mathematica time (%)	18.4971
USER-IData	
USER-RData	

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.

```
SMTShowMesh["Marks" → True, "Field" → "Tem*", "Contour" → True]
```



Input Data

The obligatory parts of the input data phase are:

- A) The input phase starts with the initialization (see SMTInputData).
- B) 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 by the AceGen code generator or taken from the AceShare libraries.
- C) The node coordinates and the connectivity of nodes for topological mesh can be given by the SMTMesh , the SMTAddNode , and the SMTAddElement commands. 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.
- D) The essential (Dirichlet) and the natural (Neumann) boundary conditions of the problem are specified by the SMTAddEssentialBoundary , the SMTAddNaturalBoundary and the SMTAddInitialBoundary commands.
- E) If sensitivity analysis is required then the SMTAddSensitivity command specifies the type and the values of the sensitivity parameters.

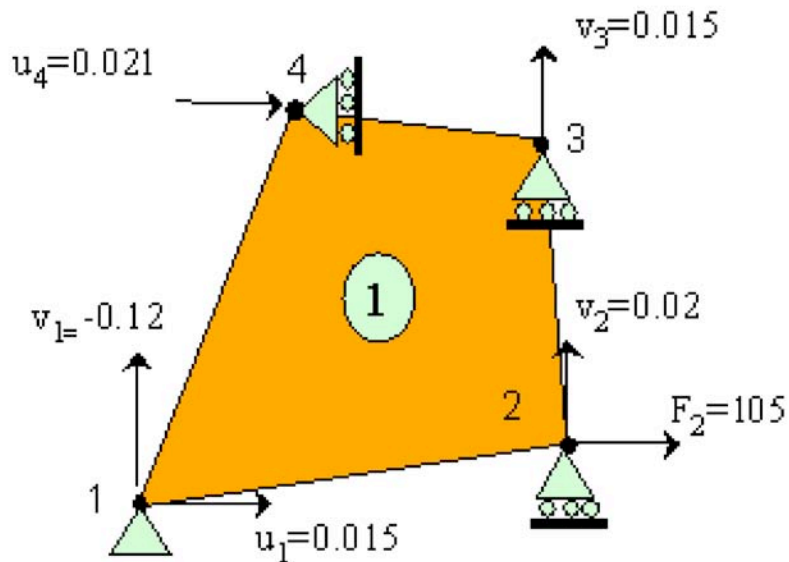
Note that the node numbering is changed after the input data phase due to the process of joining the nodes which have the coordinates and the node identification with the same value.

See also: Standard 6-element benchmark test for distortion sensitivity (2D solids)

Bending of the column (path following procedure, animations, 2D solids)

■ Example: One element

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



```
<< 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.

```
SMTAddDomain[
  {"patch", "SEPEQ1DFLEQ1Hooke", {"E *" -> 3000., "ν *" -> 0.3, "t *" -> 2.}}]
```

Definition of nodes.

```
SMTAddNode[{-0.5, 0.1}, {1.2, -0.3}, {1.1, 1.3}, {0.1, 1.4}]
{1, 2, 3, 4}
```

Definition of elements.

```
SMTAddElement[{"patch", {1, 2, 3, 4}}]
```

1

Definition of boundary conditions.

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

{{1, 1 -> 0.015, 2 -> -0.12}, {2, 2 -> 0.02}, {3, 2 -> 0.015}, {4, 1 -> 0.021}}

{{2, 1 -> 105}}
```

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

```
SMTAnalysis[]

True
```

Analysis Phase

The standard parts of the analysis phase are:

- A) The analysis phase starts with the initialization (see SMTAnalysis).
- B) 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 of the iterative solution procedure are given in Iterative solution procedure .
- C) The graphic post-processing of the results can be part of the analysis (SMTPostData , SMTShowMesh) or done later independently of the analysis (SMTPut).
- D) 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;
- 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 procedure).

Iterative solution procedure

Symbol table:

$\square t$ current iterative value

$\square p$ value at the end of previous time (load) step

Bt, Bp boundary conditions

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

$\overline{B}t$ 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 $\overline{B}t := Bt \setminus \tilde{B}t$. 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 used for the initialization of the Bp vector at the start of the analysis and discarded after.

ap, at global variables with unknown value

$\tilde{a}p, \tilde{a}t$ global variables with prescribed value (essential boundary condition)

ht, hp element specific (transient) variables

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

- 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 $Bt := Bp + \Delta\lambda dB$, where $\Delta\lambda$ is the multiplier increment
 - essential boundary conditions are set $\tilde{a}t := \tilde{B}t$

- natural boundary conditions are added to the global vector $\mathbf{R}:=\mathbf{R}+\overline{\mathbf{B}t}$
- user subroutine "*Tangent and residual*" is called for each element,
 - the element tangent matrix \mathbf{K}_e is added to the global matrix $\mathbf{K}:=\mathbf{K}+\mathbf{K}_e$,
 - element residual Ψ_e is taken from the global vector $\mathbf{R}:=\mathbf{R}-\Psi_e$
- set of linear equations is solved $\mathbf{K} \Delta \mathbf{a}=\mathbf{R}$,
- solution is incremented $\mathbf{a}t:=\mathbf{a}t+\Delta \mathbf{a}$.

See also: Standard 6-element benchmark test for distortion sensitivity (2D solids)

Bending of the column (path following procedure, animations, 2D solids)

■ Example: constant increment

This is a path following procedure with a constant multiplier increment (the multiplier increment runs from 0 to 1 in 10 steps).

```
Do[SMTNextStep[1, .1];
  While[SMTConvergence[10^-8, 10], SMTNewtonIteration[]];
  , {i, 1, 10}]
```

■ Example: adaptive increment

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

```
SMTNextStep[1, 0.1];
While[
  While[step = SMTConvergence[10^-8, 10, {"Adaptive", 8, 0.001, 0.2, 10.}],
    SMTNewtonIteration[]];
  step[[3]]
  , If[step[[1]], SMTStepBack[]];
  SMTNextStep[1, step[[2]]]
]
```

■ Example: adaptive increment with graphics

This is a path following procedure with an adaptive multiplier increment (the multiplier increment 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.

```
graph = {};
SMTNextStep[1, 0.1];
While[
  While[step = SMTConvergence[10^-8, 10, {"Adaptive", 8, 0.001, 0.2, 10.}],
    SMTNewtonIteration[]];
  If[step[[4]] == "MinBound", SMTStatusReport[" $\Delta\lambda < \Delta\lambda_{min}$ "]];];
  If[Not[step[[1]]],
    SMTShowMesh["DeformedMesh" -> True, "Show" -> "Window"];
    AppendTo[graph, {SMTData["Multiplier"], SMTPostData["Sxx", {.2, .5}]}];
  ];
  step[[3]]
  , If[step[[1]], SMTStepBack[]];];
  SMTNextStep[1, step[[2]]
];
ListLinePlot[graph]
```

Data Base Manipulations

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.

- environment data (see SMTIData , SMTRData ,);
- node data (see SMTNodeData , Node Data);
- node specification data (see SMTNodeSpecData , Node Specification Data);
- element data (see SMTElementData , Element Data);
- element specification data (see SMTDomainData , Domain Specification Data);
- tangent matrix nad residual related to the whole structure as well as to the particular element (see SMTData);
- method used to solve linear system of equations during the iterative procedure (see SMTSetSolver).

The user can interactively change during the analysis:

- all environment data;
- node coordinates;
- values of the essential nad natural boundary conditions and all unknowns of the problem;
- elements nodes;
- material data;
- elements history variables.

Selecting Nodes

<i>form</i>	<i>selected nodes</i>
$\{in_1, in_2, \dots, in_N\}$	N nodes with the node indeces in_1, in_2, \dots, in_N (note that node index can be changed due to the "Tie" command after the <i>SMTAnalysis</i> command)
<i>i_Integer</i>	$\equiv \{i\}$
<i>crit_Function</i>	nodes for which test $crit[x_i, y_i, z_i, nID_i]$ yields True
Line[{T ₁ , T ₂ }]	all nodes on line segment with end points T ₁ = {x ₁ , y ₁ , z ₁ } and T ₂ = {x ₂ , y ₂ , z ₂ }
Point[T]	all nodes with coordinates T={x, y, z}
{"Find", X, Y, Z, nID}	search for node with the coordinates {X, Y, Z} and node identification <i>nID</i>
{"Find", "ALL", Y, Z, nID}	search for all nodes with the coordinates $y_i \equiv Y, z_i \equiv Z,$ arbitrary x coordinate and node identification <i>nID</i> (other combinations are also possible e.g. {" Find ", X, "ALL", "ALL", nID})
{"Find", X, Y, Z}	search for all nodes with the coordinates $x_i \equiv X,$ $y_i \equiv Y, z_i \equiv Z$ and arbitrary identification $\equiv \{X, Y, Z, "ALL"\}$
{"Find", X, Y}	search for all nodes with the coordinates $x_i \equiv X, y_i \equiv Y$ $\equiv \{X, Y, "ALL", "ALL"\}$
{"Find", X}	search for all nodes with the coordinates $x_i \equiv X$ $\equiv \{X, "ALL", "ALL", "ALL"\}$
{"Find", <i>crit</i> }	$\equiv crit$

Selecting nodes.

Many functions require as input a list of nodes on which certain action is applied. The parameter that is used to select nodes can be a list of node indeces, coordinates of the nodes or a pure function. The simbol "ALL" or simply empty space can be used insted of the coordinates of the nodes. The string "ALL" means that boundary condition is prescribed for all nodes with the given coodinate or node identification.

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 "ID".

Examples: "X"==5 && "Z"==2 & \equiv {"Find",5,"ALL",2} \equiv {"Find",5, ,2} \equiv {"Find","X"==5 && "Z"==2 &} \equiv {"Find",#1==5 && #3==2&}

The number of coordinates is reduced for 2D and 1D problems as follows.

<i>form</i>	<i>explanation</i>
{" Find ", X, Y, nID}	sets essential or natural boundary condition for the node with the coordinates {X, Y} and node identification <i>nID</i>
{" Find ", <i>crit</i> }	nodes for which test $crit[x_i, y_i, nID_i]$ yields True

Selecting nodes for 2D problems.

<i>form</i>	<i>explanation</i>
<code>{"Find ", X, nID}</code>	sets essential or natural boundary condition for the node with the coordinates {X,Y} and node identification <i>nID</i>
<code>{"Find ", crit}</code>	nodes for which test <code>crit[x_i, nID_i]</code> yields True

Selecting nodes for 1D problems.

Selecting Elements

<i>form</i>	<i>selected elements</i>
<code>{ie₁, ie₂, ... ie_N}</code>	N elements with the element indeces <i>ie₁, ie₂, ... ie_N</i>
<i>i_Integer</i>	$\equiv \{i\}$
<i>dID</i>	all elements with domain identification <i>dID</i>
<i>crit_Function</i>	selection of elements with the nodes for which test <code>crit[x_i, y_i, z_i, nID_i]</code> yields True

Selecting nodes for 3D problems.

Many functions require as input a list of elements on which certain action is applied. The parameter that is used to select elements can be a list of element indeces, domain identification or pure function.

The parameter *crit* 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".

Examples: `"X"<5 && "Z">2 &`

Interactive Debugging

The procedures described in the section User Interface 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 may 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.

```

<< AceGen` ;
SMSInitialize["ExamplesDebugHeatConduction",
  "Environment" -> "AceFEM", "Mode" -> "Debug"];
SMSSTemplate["SMSTopology" -> "H1", "SMSDOFGlobal" -> 1,
  "SMSSymmetricTangent" -> False,
  "SMSGroupDataNames" ->
  {"k0 -conductivity parameter", "k1 -conductivity parameter",
  "k2 -conductivity parameter", "Q -heat source"},
  "SMSDefaultData" -> {1, 0, 0, 0}];
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]];
E = {ξ, η, ζ} + Table[SMSReal[es$$["IntPoints", i, Ig]], {i, 3}];
XI + Table[SMSReal[nd$$[i, "X", j]], {i, SMSNoNodes}, {j, SMSNoDimensions}];
En = {{-1, -1, -1}, {1, -1, -1}, {1, 1, -1}, {-1, 1, -1},
  {-1, -1, 1}, {1, -1, 1}, {1, 1, 1}, {-1, 1, 1}};
NI = Table[1 / 8 (1 + ξ En[[i, 1]]) (1 + η En[[i, 2]]) (1 + ζ En[[i, 3]]), {i, 1, 8}];
X + SMSFreeze[NI.XI]; Jg = SMSD[X, E]; Jgd = Det[Jg];
φI + SMSReal[Table[nd$$[i, "at", 1], {i, SMSNoNodes}]];
φ = NI.φI;
{k0, k1, k2, Q} + SMSReal[Table[es$$["Data", i], {i, Length[SMSGroupDataNames]}]];
k = k0 + k1 φ + k2 φ2;
SMSSetBreak["k"];
λ + SMSReal[rdata$$["Multiplier"]];
wgp + SMSReal[es$$["IntPoints", 4, Ig]];
SMSDo[
  Dφ = SMSD[φ, X, "Dependency" -> {E, X, SMSInverse[Jg]}];
  δφ = SMSD[φ, φI, i];
  Dδφ = SMSD[δφ, X, "Dependency" -> {E, X, SMSInverse[Jg]}];
  Rg = Jgd wgp (k Dδφ.Dφ - δφ λ Q);
  SMSExport[SMSResidualSign Rg, p$$[i], "AddIn" -> True];
  SMSDo[
    Kg = SMSD[Rg, φI, j];
    SMSExport[Kg, s$$[i, j], "AddIn" -> True];
    , {j, 1, 8}
  ];
  , {i, 1, 8}
];
SMSEndDo[];
SMSWrite[];

time=0 variable= 0 ≡ {}
time=1 variable= 100 ≡ {}

[2] Consistency check - global
[2] Consistency check - expressions
[3] Generate source code :

Events: 6 SMSDB-0

[3] Final formatting

```


File:	ExamplesDebugHeatConduction.c	Size:	20 382
Methods	No. Formulae	No. Leafs	
SKR	190	4099	

The *SMTMesh* 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*.

```
<< 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}];
SMTMesh["cube", "H1", {5, 5, 5}, {
{{{-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}}}
}];
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.

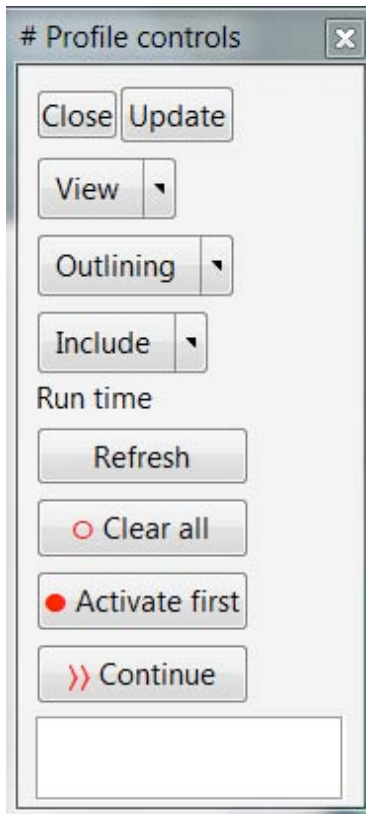
The program stops for each integration point where the program structure together with all the basic variables of the problem and the current values of the variables are presented.

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.

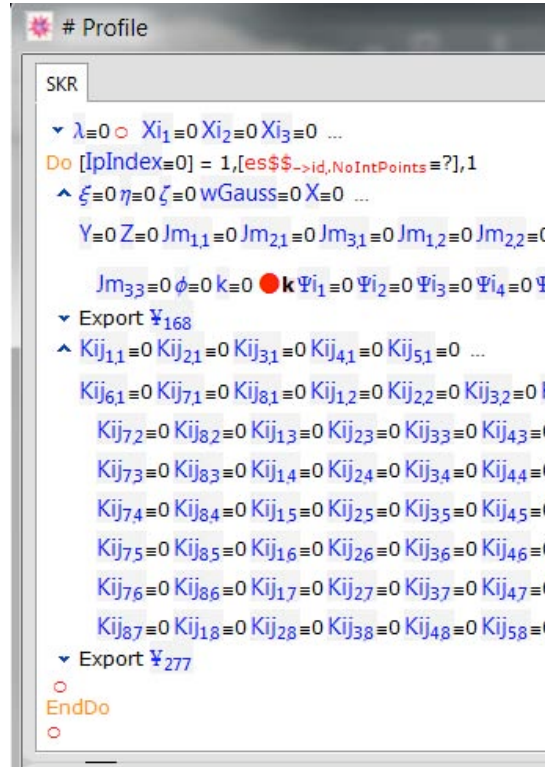
```
SMTIData["DebugElement", 1];

SMTNextStep[0, 0];
SMTNewtonIteration[];
```

Debugger palette



Display

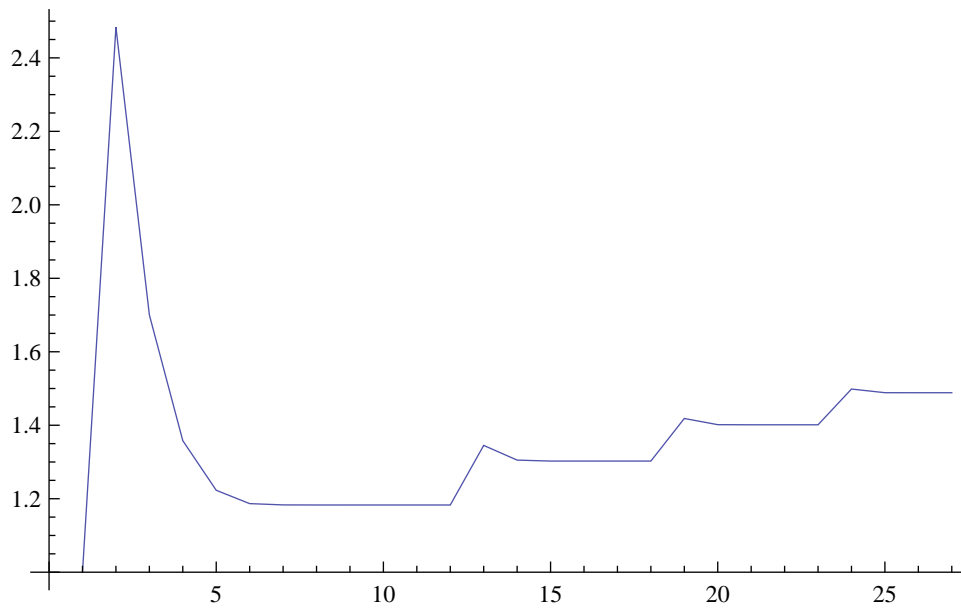


The break points can be used also to trace the values of arbitrary variables during the analysis. Here the value of the conductivity k in the 6-th integration point of the 512-th element is traced during the Newton-Raphson iterative procedure.

```
allk = {};
SMSActivateBreak["k", If[ Ig == 6, allk = {allk, k};] &];

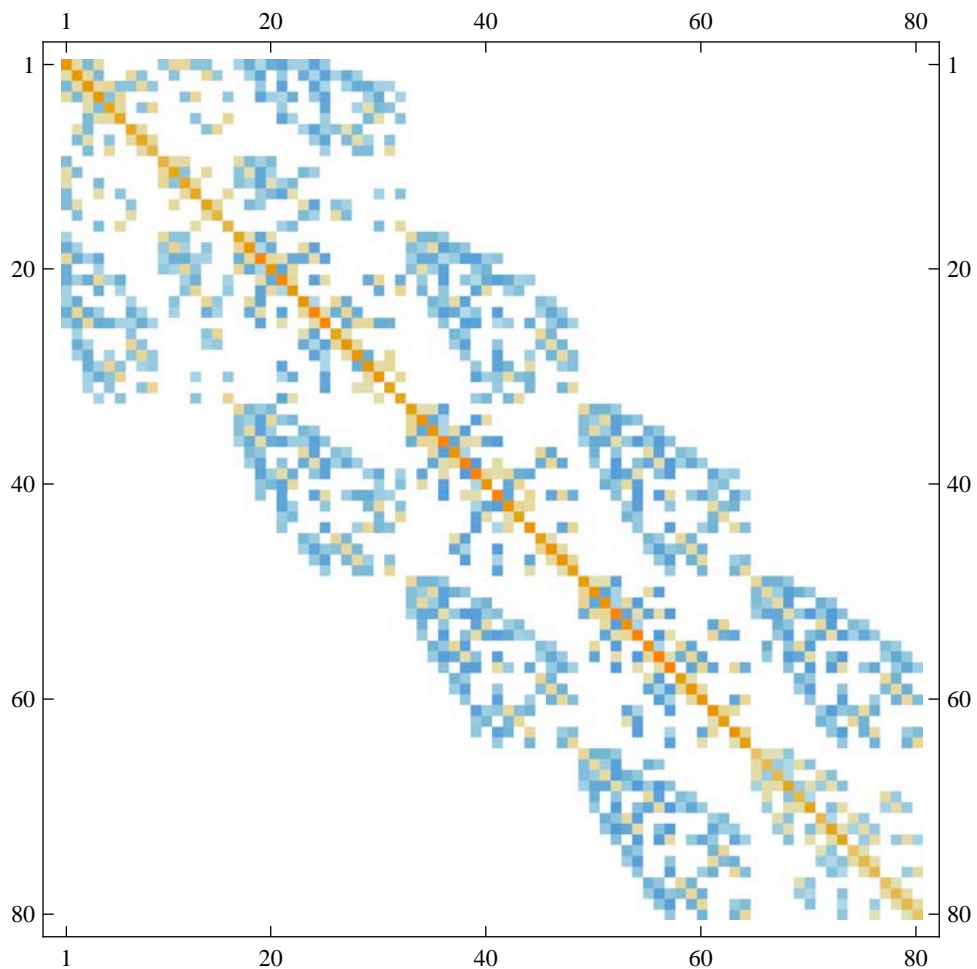
Do[
  SMTNextStep[1, 500];
  While[SMTConvergence[10^-12, 15], SMTNewtonIteration[]];
  , {i, 1, 4}]
```

```
ListLinePlot[allk // Flatten, PlotRange -> All]
```



The sparsity structure of the resulting tangent matrix can be graphically displayed by the ArrayPlot function.

```
MatrixPlot[SMTData["TangentMatrix"]]
```



Code Profiling

Code profiling is typically used to understand exactly where an application is spending its execution time. The SMSimulationReport 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.

```
<< AceGen` ;
SMSInitialize["ExamplesProfileHeatConduction", "Environment" -> "AceFEM"];
SMSTemplate["SMSTopology" -> "H1", "SMSDOFGlobal" -> 1
, "SMSSymmetricTangent" -> False
, "SMSIDataNames" -> {"NoEvaluations"}
, "SMSRDataNames" -> {"EvaluationTime"}
, "SMSGroupDataNames" ->
{"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.

```
time = SMSTime[];

SMSDo[Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]];
E = {ξ, η, ζ} + Table[SMSReal[es$$["IntPoints", i, Ig]], {i, 3}];
XI = Table[SMSReal[nd$$[i, "X", j]], {i, SMSNoNodes}, {j, SMSNoDimensions}];
En = {{-1, -1, -1}, {1, -1, -1}, {1, 1, -1}, {-1, 1, -1},
{-1, -1, 1}, {1, -1, 1}, {1, 1, 1}, {-1, 1, 1}};
NI = Table[1 / 8 (1 + ξ En[[i, 1]]) (1 + η En[[i, 2]]) (1 + ζ En[[i, 3]]), {i, 1, 8}];
X = SMSFreeze[NI.XI]; Jg = SMSD[X, E]; Jgd = Det[Jg];
φI = SMSReal[Table[nd$$[i, "at", 1], {i, SMSNoNodes}]];
φ = NI.φI;
{k0, k1, k2, Q} = SMSReal[Table[es$$["Data", i], {i, Length[SMSGroupDataNames]}]];
k = k0 + k1 φ + k2 φ2;
λ = SMSReal[rdata$$["Multiplier"]];
wgp = SMSReal[es$$["IntPoints", 4, Ig]];
SMSDo[
Dφ = SMSD[φ, X, "Dependency" -> {E, X, SMSInverse[Jg]}];
δφ = SMSD[φ, φI, i];
Dδφ = SMSD[δφ, X, "Dependency" -> {E, X, SMSInverse[Jg]}];
Rg = Jgd wgp (k Dδφ.Dφ - δφ λ Q);
SMSExport[SMSResidualSign Rg, p$$[i], "AddIn" -> True];
SMSDo[
Kg = SMSD[Rg, φI, j];
SMSExport[Kg, s$$[i, j], "AddIn" -> True];
, {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".

```

SMSExport[SMTTime[] - time, rdata$$["EvaluationTime"], "AddIn" -> True];
SMSExport[SMSInteger[idata$$["NoEvaluations"]] + 1, idata$$["NoEvaluations"]];

SMSWrite[];

```

File:	ExamplesProfileHeatConduction.c	Size:	10 687
Methods	No.Formulae	No.Leafs	
SKR	169	2583	

Here is presented an input for the analysis and the results of code profiling for a cube discretized by the 20×20×20 mesh.

```

<< 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];
SMTMesh["cube", "H1", {20, 20, 20}, {
  {{-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}}
}];
SMTAnalysis[];
SMTNextStep[0, 1];
While[SMTConvergence[10^-12, 10], SMTNewtonIteration[]];
SMTSimulationReport[{"NoEvaluations"}, {"EvaluationTime"}];

```

```

No. of nodes                9261
No. of elements             8000
No. of equations            7220
Data memory (KBytes)        2603
Number of threads used/max   8/8
Solver memory (KBytes)      29 517
No. of steps                 1
No. of steps back            0
Step efficiency (%)          100.
Total no. of iterations      4
Average iterations/step      4.
Total time (s)               2.855
Total linear solver time (s)  0.718
Total linear solver time (%)  25.1489
Total assembly time (s)      0.109
Total assembly time (%)      3.81786
Average time/iteration (s)   0.71375
Average linear solver time (s) 0.1795
Average K and R time (s)     3.40625 × 10-6
Total Mathematica time (s)   2.013
Total Mathematica time (%)   70.5079
USER-IData
Total NoEvaluations          31 986
Average NoEvaluations/element 0.999563
USER-RData
Total EvaluationTime (s)     0.232999
Total EvaluationTime (%)     8.1611
Average EvaluationTime/element (s) 7.28123 × 10-6

```

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)

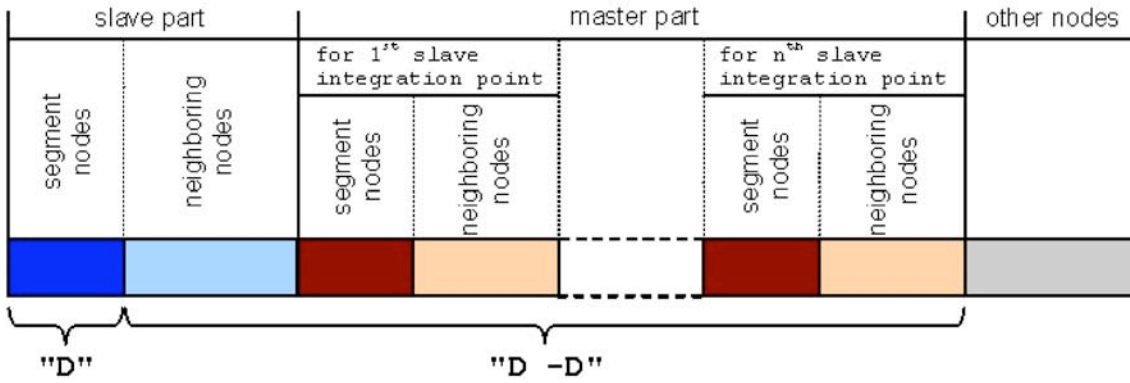
SMSTemplate[..., "SMSCharSwitch" -> { <i>slave part contact type, master part contact type</i> , "ContactElement"}, ...]	
slave part contact types	"CTD2N1 " - one 2 D node "CTD2L1 " - one 2 D line "CTD3V1 " - one 3 D node "CTD3P1 " - one 3 D triangle "CTD3S1 " - one 3 D quad
master part contact types	"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


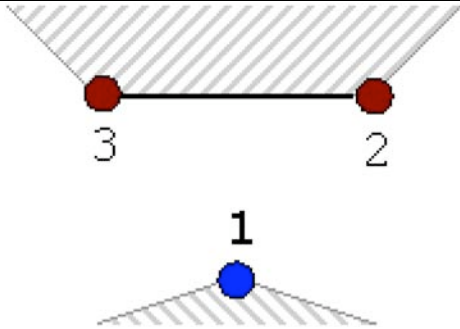
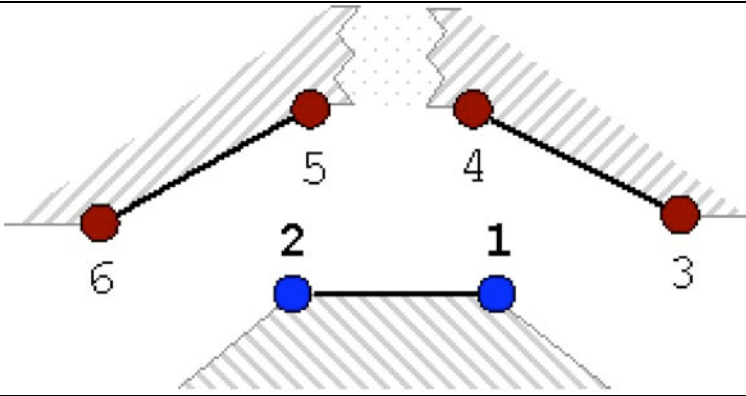
It is possible to extend each contact type (except CTNULL) to attach also neighboring nodes. You do this appending "DN<i>" string to the contact type, where <i> is a number of dummy slots prepared for each node's neighborhood. Thus you have to define additional (i*NumberOfSlaveNodes) slots if you extend the slave part or (i*NumberOfMasterNodes*NumberOfIntegrationPoints) slots in case of extending master part.

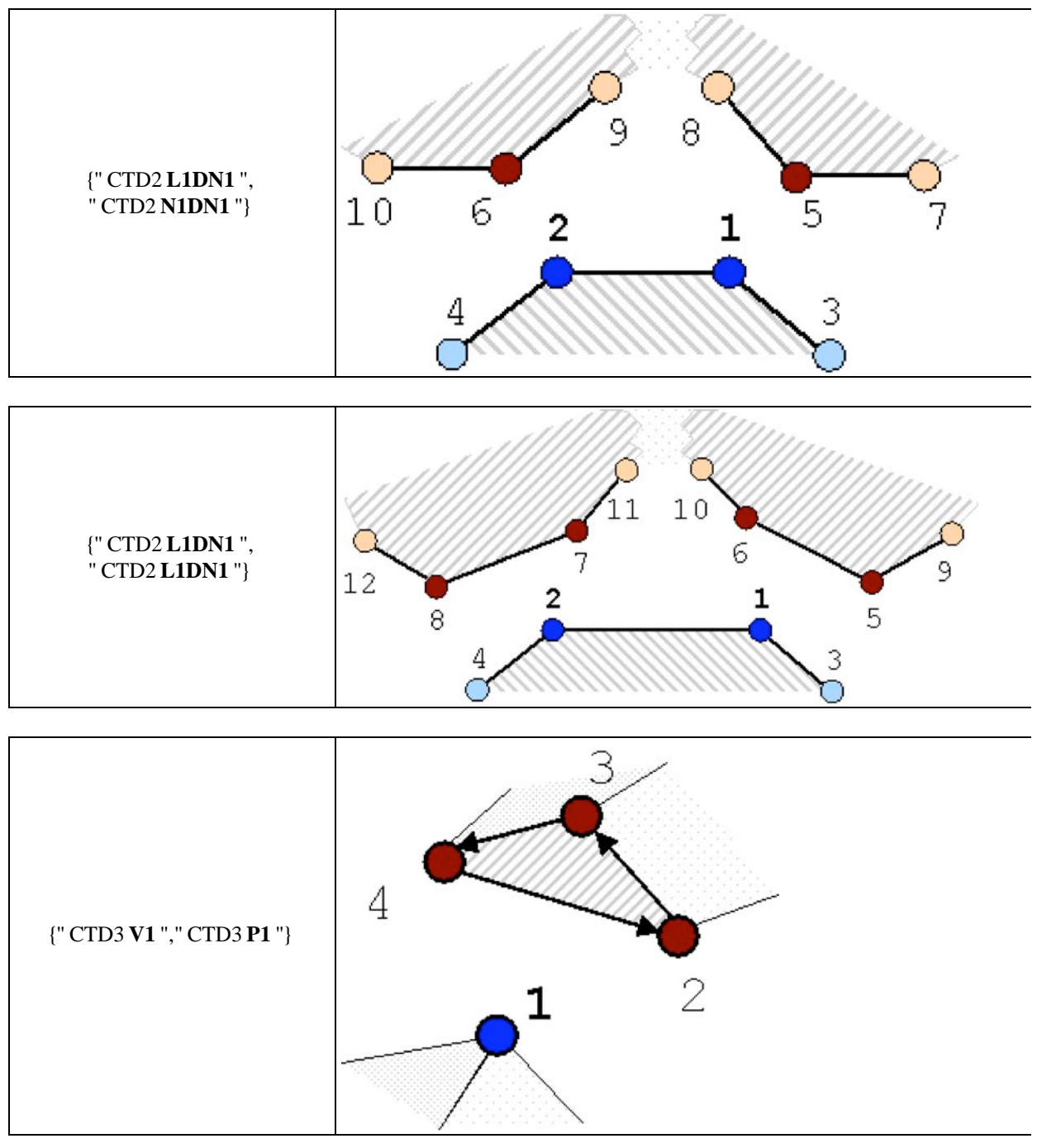
NOTE: the number of neighboring nodes may vary during analysis, while the number of slots in element is strictly defined by "DN<i>" flag. The question is: what the system puts into unused dummy slots. The answer is: the most recent neighbor found for given node. The reason is that the common use of neighboring 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.

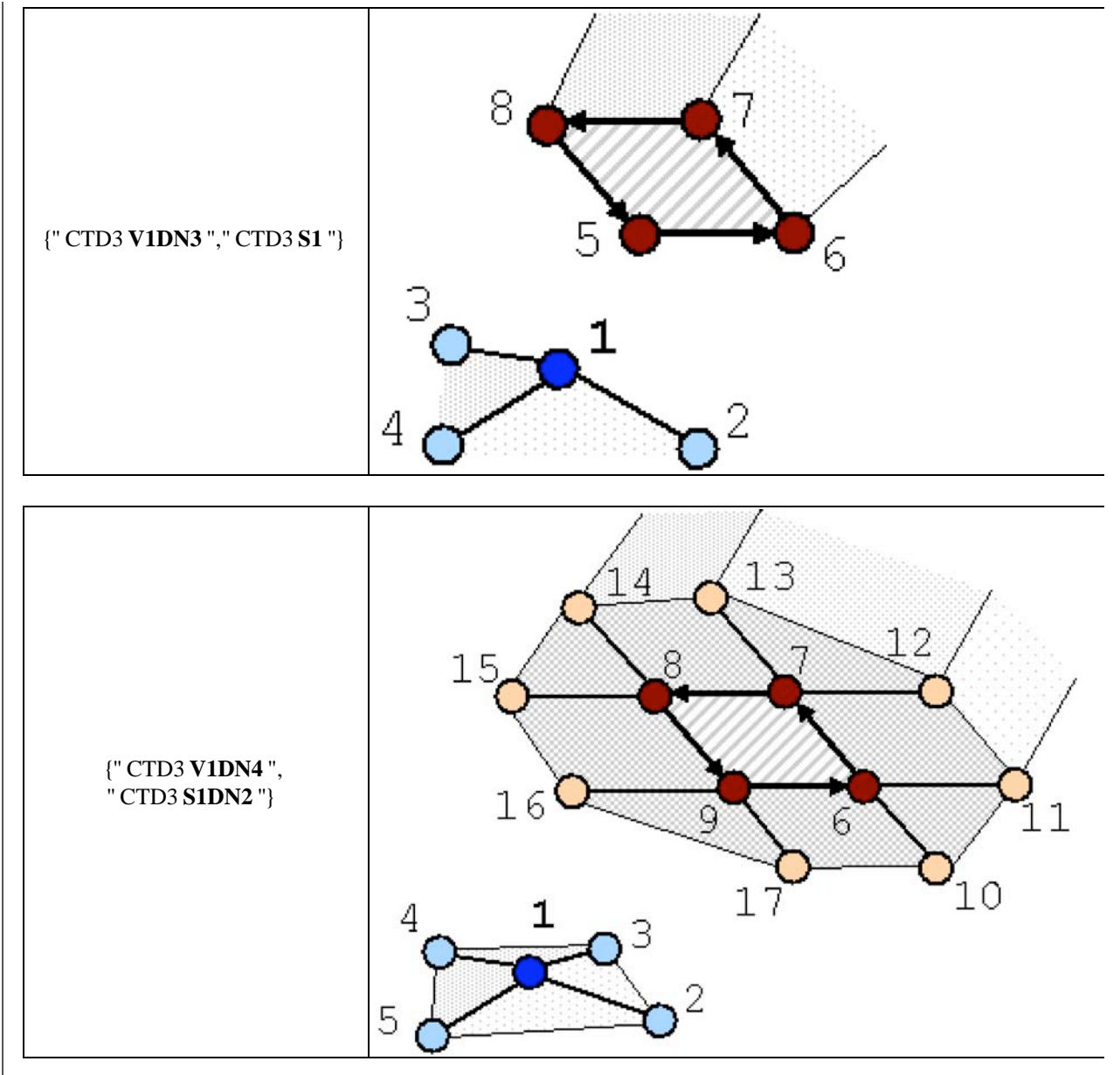
NOTE: from the same reasons as above, there might be the situation where there are more neighboring nodes found than the number of slots in contact element. In such a situation the slots are filled with a natural order until it is possible. There is only one exception to this procedure: the last slot 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
{" CTD2 N1 ", " CT NULL "	<p style="text-align: center;">1</p> 
{" CTD2 N1 ", " CTD2 L1 "	
{" CTD2 L1 ", " CTD2 L1 "	





Position of nodes in the contact element (detailed numbering scheme for choosen 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 SMTMesh 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 -> {{slaveBodyID1, masterBodyID1}, {slaveBodyID2, masterBodyID2}, ...}` 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) :

<i>Default form</i>	<i>Description</i>	<i>Default value</i>
SMTRData["ContactSearchTolerance", tolerance]	contact search tolerance for global search routine.	0.001 SMTRData["XYZRange"]

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.

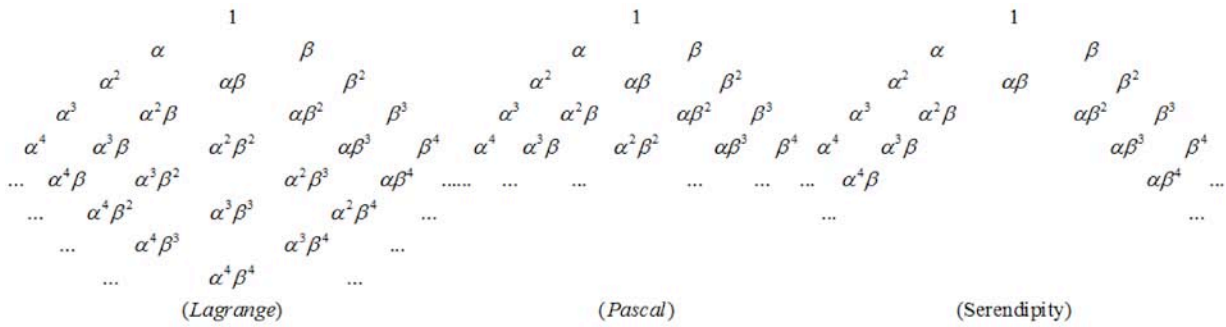
NOTE: it is important to cover with contact elements also the body, which is "master" to all the others. Such an elements are needed to calculate the actual positions of nodes (integration points) on the surface of this body. It is enough to have only "Nodal information" subroutine defined there. In this case we may set the master part contact type as "CTNULL".

<i>Function name</i>	<i>Description</i>
SMTContactData[]	prints out the actual contact state (node->segment mapping)
SMTContactSearch[]	manually runs the global contact search routine

Contact related functions.

Semi-analytical solutions

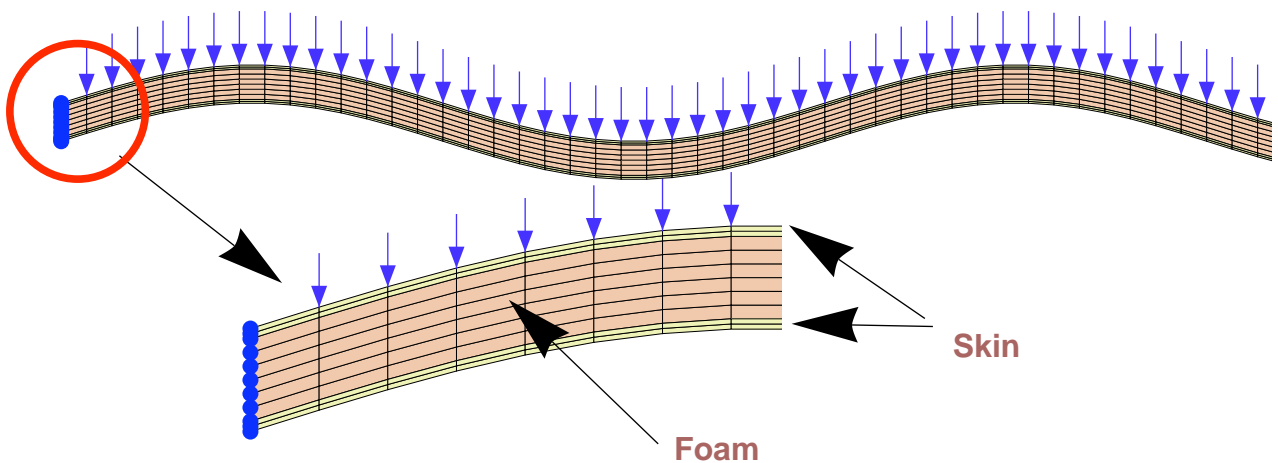
Semi-analytical solution of the finite element problem is a solution of the problem where the results are expressed as a power series expansion with respect to one or several parameters of the problem. The expansion parameters, the expansion point and the order of the power series expansion is specified 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. Three types of reduces expansion ("Lagrange", "Pascal", and "Serendipity") are available as depicted in a table below.



<i>expansion</i>	<i>description</i>
"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 employs 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.

```

<< 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", "SEPSQ1DFLEQ1Hooke",
  {"E *" → 500., "ν *" → 0.48, "t *" → b}];
SMTAddDomain["Steel", "SEPSQ1DFLEQ1Hooke",
  {"E *" → 21 000, "ν *" → 0.3, "t *" → b}];
SMTMesh["Foam", "Q1", {nx, ny},
  {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}]}];
SMTMesh["Steel", "Q1", {nx, 2},
  {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}]}];
SMTMesh["Steel", "Q1", {nx, 2},
  {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}]}];
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[];
SMTNextStep[1, 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.

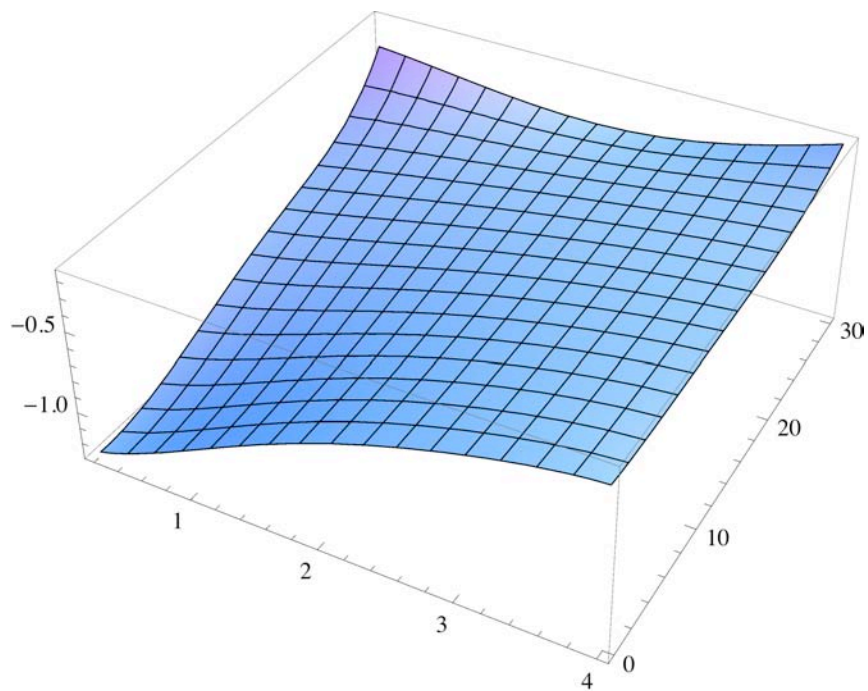
```

w = Normal[ SMTNodeData["X" == L / 2 &, "at"][[1, 2]]]
-0.632351 + 0.233877 (-2. + α) - 0.0364066 (-2. + α)2 -
0.00448821 (-2. + α)3 + 0.00435552 (-2. + α)4 - 0.0012384 (-2. + α)5 +
(0.00835702 - 0.00977376 (-2. + α) + 0.00342247 (-2. + α)2 - 0.000207154 (-2. + α)3 -
0.000314466 (-2. + α)4 + 0.000152023 (-2. + α)5) (-10. + β) +
(0.000352781 - 0.000126855 (-2. + α) - 0.0000726406 (-2. + α)2 + 0.000050776 (-2. + α)3 -
6.11087 × 10-6 (-2. + α)4 - 5.64683 × 10-6 (-2. + α)5) (-10. + β)2 +
(-9.40667 × 10-6 + 0.0000266261 (-2. + α) - 0.0000118708 (-2. + α)2) (-10. + β)3 +
(-4.12175 × 10-7 - 4.01097 × 10-7 (-2. + α) + 7.85323 × 10-7 (-2. + α)2) (-10. + β)4 +
(9.00983 × 10-9 - 4.28775 × 10-8 (-2. + α) + 1.10114 × 10-8 (-2. + α)2) (-10. + β)5

wc = w / . α -> T0foam / . β -> hwave0
-0.632351

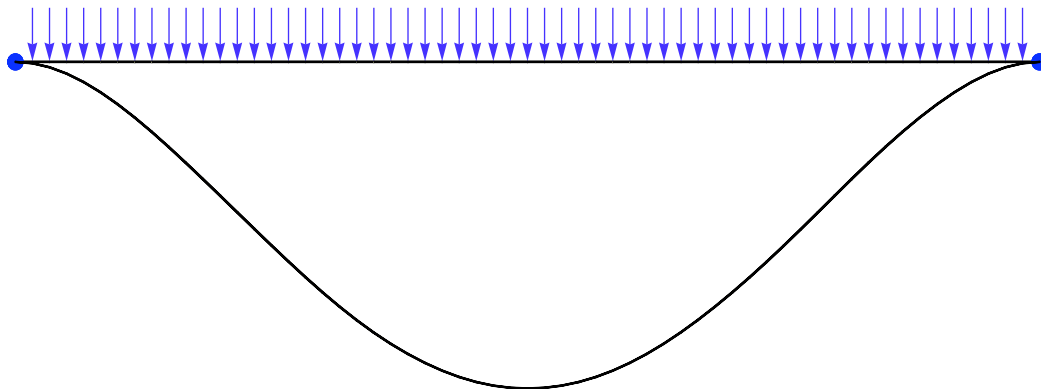
```

```
Plot3D[w, { $\alpha$ , 0.1 T0foam, 2 T0foam}, { $\beta$ , 0, 3 hwave0}]
```

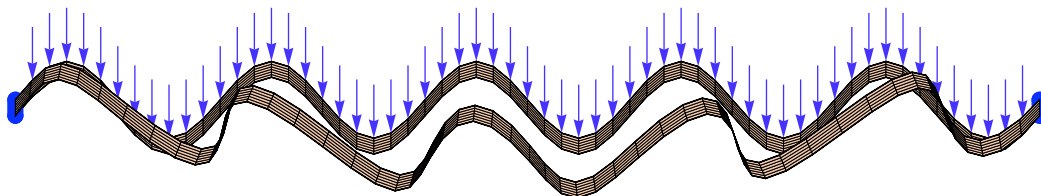


The "ShowFor" option of the SMTShowMesh command can be used to depict mesh and results for arbitrary values of parameters.

```
Show[SMTShowMesh["BoundaryConditions" -> True, "ShowFor" -> { $\alpha$  -> 0.1 T0foam,  $\beta$  -> 0}],  
SMTShowMesh["DeformedMesh" -> True,  
"Scale" -> 100, "ShowFor" -> { $\alpha$  -> 0.1 T0foam,  $\beta$  -> 0}]]
```



```
Show[SMTShowMesh["BoundaryConditions" -> True,  
"ShowFor" -> { $\alpha$  -> 3 T0foam,  $\beta$  -> 3 hwave0}], SMTShowMesh["DeformedMesh" -> True,  
"Scale" -> 100, "ShowFor" -> { $\alpha$  -> 3 T0foam,  $\beta$  -> 3 hwave0}]]
```



User Defined Tasks

The standard task performed by the finite element environment are evaluation of the global residual and tangent matrix, solution of the resulting system of linear equations and postprocessing of the results. Additionally to the standard tasks, user can also define and execute user defined tasks. The process of defining and executing the user defined tasks 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 SMTask command.

SMTask[*taskID*] execute task that corresponds to the task identification *taskID*

<i>option</i>	<i>description</i>	<i>default value</i>
"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" -> <i>elementselector</i>	the user subroutine "Tasks" is called only for the selected elements	All
"Point" -> {x,y,z}	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, see also)	False

Options for the SMTask function.

The user defined tasks can be used to perform various tasks that requires 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 SMTask command depends on the type of the task as well as on the number of output parameters. 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 SMSTasks command. If the number of IntegerOutput or RealOutput values is 1 then a scalar is returned rather than a vector.

<i>task type</i>	<i>task description</i>	return values
1	the user subroutine is called for all or selected elements, resulting integer and real output vectors are summarized and returned as the results of the task	{IntegerOutput,RealOutput}
2	see above	continuous field evaluated in all nodes or in a given spatial point
3	see above	continuous field evaluated in all nodes or in a given spatial point
4	the user subroutine is called for all elements and assembled global vector is returned together with the integer and real output vectors	{IntegerOutput,RealOutput,VectorGlobal}
5	the user subroutine is called for all elements and assembled global matrix is returned together with the integer and real output vectors	{IntegerOutput,RealOutput,MatrixGlobal}
6	the user subroutine is called for all elements and assembled global vector and global matrix are returned together with the integer and real output vectors	{IntegerOutput,RealOutput,VectorGlobal,MatrixGlobal}

Return values accordingly to the type of the task.

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 the 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.

■ 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:

- 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.
- Calculate the mesh distortion in an arbitrary point of the mesh. Mesh distortion is defined by

$$d = \frac{|\text{ArcTang}[r,\eta] - \text{ArcTang}[r,\xi] - \pi/2|}{\pi/2} \text{ where } \{\xi,\eta\} \text{ are the reference coordinates of the element.}$$

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.

```

<< "AceGen`";
SMSInitialize["ExamplesTasks2D", "Environment" -> "AceFEM"];
SMSTemplate["SMSTopology" -> "Q1",
  "SMSCharSwitch" -> {"Mass", "MeshDistortion"},
  "SMSGroupDataNames" -> {"ρ -density", "t -thickness"},
  "SMSDefaultData" -> {1, 1}];
SMSStandardModule["Tasks"];
task = SMSInteger[Task$$];

SMSIf[task < 0

, SMSSwitch[task
  , -1,
  SMSExport[{1, 0, 0, 0, 1}, TasksData$$];
  , -2,
  SMSExport[{3, 0, 0, 0, SMSInteger[es$$["id", "NoIntPoints"]}], TasksData$$];
];

, {ρ, tξ} = SMSReal[Table[es$$["Data", i], {i, Length[SMSGroupDataNames]}]];
{Xn, Yn} = Table[SMSReal[nd$$[j, "X", i]], {i, 2}, {j, SMSNoNodes}];
SMSDo[
  E = {ξ, η} = Table[SMSReal[es$$["IntPoints", i, Ig]], {i, 2}];
  XI = Table[SMSReal[nd$$[i, "X", j]], {i, SMSNoNodes}, {j, SMSNoDimensions}];
  NI = 1/4 {(1 - ξ) (1 - η), (1 + ξ) (1 - η), (1 + ξ) (1 + η), (1 - ξ) (1 + η)};
  X = SMSFreeze[NI.XI];
  Jg = SMSD[X, E]; Jgd = Det[Jg];
  SMSSwitch[task
    , 1,
    SMSExport[Jgd tξ ρ, RealOutput$$[1], "AddIn" -> True];
    , 2,
    rξ = SMSD[X, ξ]; rη = SMSD[X, η];
    dist = SMSAbs[(ArcTan@@rη) - (ArcTan@@rξ) - (π/2)] / (π/2);
    SMSExport[dist, RealOutput$$[Ig]];
  ];
  , {Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]}
];

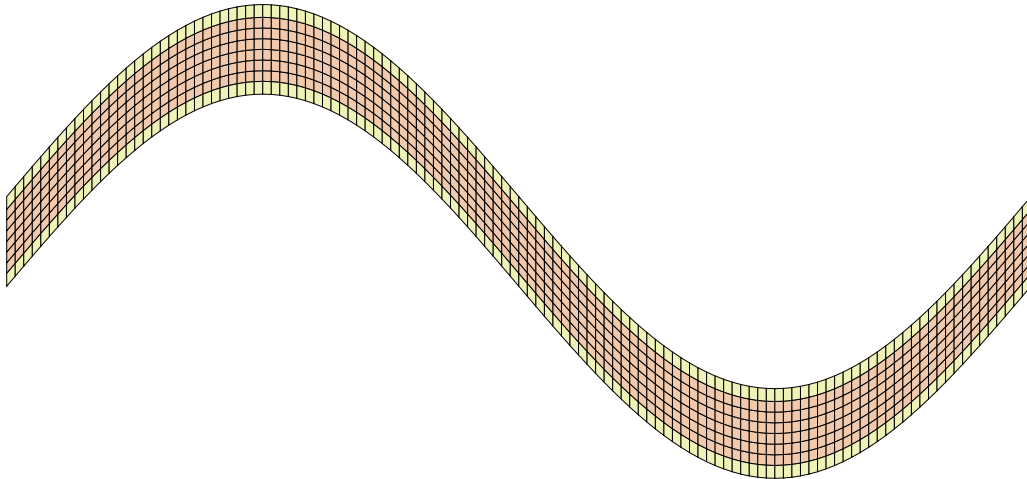
]
SMSWrite[];

```

File:	ExamplesTasks2D.c	Size:	5205
Methods	No.Formulae	No.Leafs	
Tasks	31	350	

Sinusoidal double skin cladding

Find the mass and mesh distortion of the sinusoidal double skin cladding.



```

<< AceFEM` ;
L = 80.; b = 100; hwave0 = 30; nwave = 2; T0foam = 5; T0steel = 1;
nx = 120; ny = 6;  $\delta h = 2 T0steel / ny 0.5$ ; dx = L / (10. (hwave0 + T0foam));
Tfoam = T0foam; Tsteel = T0steel; hwave = hwave0;
SMTInputData[];
SMTAddDomain["Foam", "ExamplesTasks2D", {" $\rho$  *" -> 10 10-6, "t *" -> b}];
SMTAddDomain["Steel", "ExamplesTasks2D", {" $\rho$  *" -> 7830 10-6, "t *" -> b}];
SMTMesh["Foam", "Q1", {nx, ny},
  { Table[{x, hwave / 2 Sin[nwave  $\pi$  x / L] - Tfoam / 2}, {x, 0, L, dx}],
    Table[{x, hwave / 2 Sin[nwave  $\pi$  x / L] + Tfoam / 2}, {x, 0, L, dx}]}];
SMTMesh["Steel", "Q1", {nx, 1},
  { Table[{x, hwave / 2 Sin[nwave  $\pi$  x / L] + Tfoam / 2}, {x, 0, L, dx}],
    Table[{x, hwave / 2 Sin[nwave  $\pi$  x / L] + Tfoam / 2 + Tsteel}, {x, 0, L, dx}]}];
SMTMesh["Steel", "Q1", {nx, 1},
  { Table[{x, hwave / 2 Sin[nwave  $\pi$  x / L] - Tfoam / 2 - Tsteel}, {x, 0, L, dx}],
    Table[{x, hwave / 2 Sin[nwave  $\pi$  x / L] - Tfoam / 2}, {x, 0, L, dx}]}];
SMTAnalysis[];

```

Here the mass if the structure is evaluated.

```
SMTTask["Mass"]
```

```
125.68
```

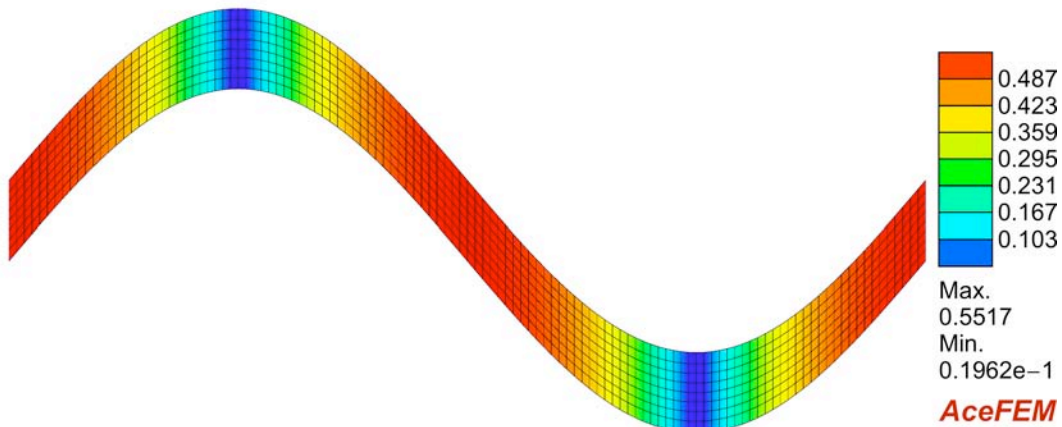
The mass if the structure can as well be calculated as a sum of the mass of the steel and the mass of the foam.

```
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.

```
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.

```
SMTTask["MeshDistortion", "Point" → {L / 2, 0}]
```

```
0.551796
```

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 subkernels (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

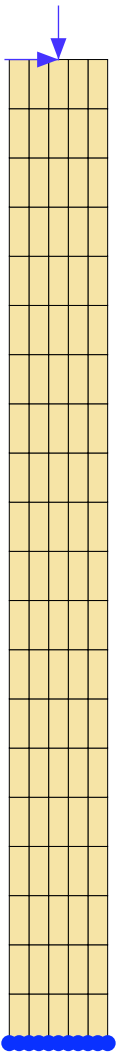
First one must launch the AceFEM on a master kernel.

```
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.

```
kernels = ParallelEvaluate[
  SetDirectory[$HomeDirectory];
  Get["AceFEM`Remote`"];
  $KernelID]
{1, 2, 3, 4, 5, 6, 7, 8}
```

Calculate the typical load/deflection curve of the $20 \times 2 \times 2$ column subjected to the constant horizontal force $H = 10$ and variable vertical force $V = -\lambda \cdot 10$ 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" \rightarrow 1 option for the `SMTInputData` command prevents the parallelization of the assembly procedure and parallelization of the linear solver.

```

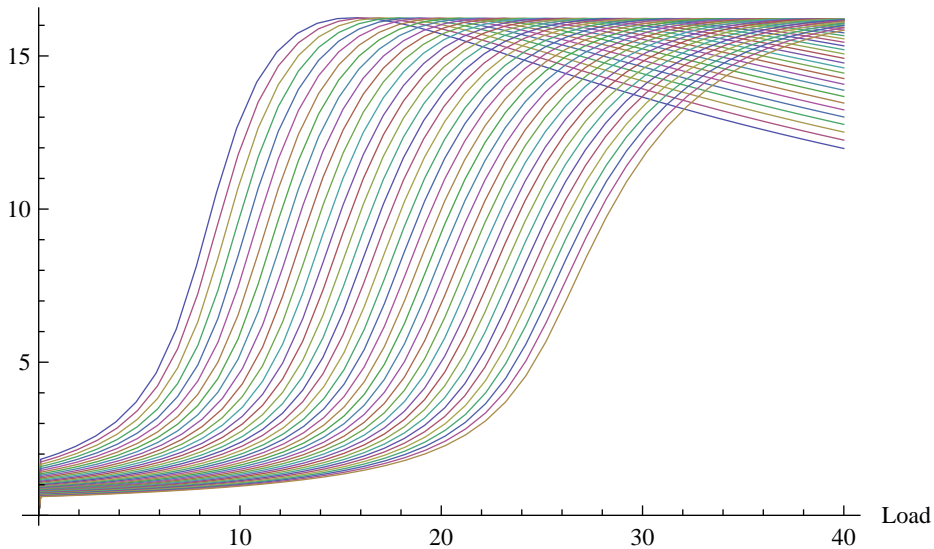
table = ParallelTable[
  SMTInputData["Threads" → 1];
  SMTAddDomain["Ω", "SEPEQ2DFHYQ2NeoHooke", {"E *" → emodule, "t *" → 2}];
  SMTAddEssentialBoundary[{ "Y" == 0 &, 1 -> 0, 2 -> 0}];
  SMTAddNaturalBoundary[ "X" == 0 && "Y" == 20 &, 2 -> -10];
  SMTAddInitialBoundary[ "X" == 0 && "Y" == 20 &, 1 -> 10];
  SMTMesh["Ω", "Q2", {20, 5}, {{1, 0}, {1, 20}}, {{-1, 0}, {-1, 20}}];
  SMTAnalysis[];
  λucurve = {{0, 0}};
  SMTNextStep[1, .1];
  While[
    While[step = SMTConvergence[10^-8, 15, {"Adaptive", 8, .0001, 1, 40}],
      SMTNewtonIteration[]];
    If[step[[4]] == "MinBound", SMTStatusReport["Error: Δλ < Δλmin"]; Abort[]];
    If[Not[step[[1]]],
      AppendTo[λucurve, {SMTData["Multiplier"], SMTPostData["u", {0, 20}]}];];
    step[[3]]
    , If[step[[1]], SMTStepBack[]];];
  SMTNextStep[1, step[[2]]
];
{emodule, λucurve}
, {emodule, 10 000, 30 000, 500}];

```

The result are the deflection curves of upper center node considering the changing elastic modulus.

```
ListLinePlot[table[[All, 2]], AxesLabel → {"Load", "Tip deflection"}]
```

Tip deflection



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 www.fgg.uni-lj.si/symech/examples/.

Basic AceFEM Examples

Standard FE Procedure

Bending of the column (path following procedure, animations, 2D solids)

Boundary conditions (2D solid)

Standard 6-element benchmark test for distortion sensitivity (2D solids)

Solution Convergence Test

Postprocessing (3D heat conduction)

Basic AceGen-AceFEM Examples

Simple 2D Solid, Finite Strain Element

Mixed 3D Solid FE, Elimination of Local Unknowns

Mixed 3D Solid FE, Auxiliary Nodes

Cubic triangle, Additional nodes

Inflating the Tyre

Advanced Examples

Round-off Error Test

Solid, Finite Strain Element for Direct and Sensitivity Analysis

Parameter, Shape and Load Sensitivity Analysis of Multi-Domain Example

Three Dimensional, Elasto-Plastic Element

Axisymmetric, finite strain elasto-plastic element

Cyclic tension test, advanced post-processing , animations

Solid, Finite Strain Element for Dynamic Analysis

Elements that Call User External Subroutines

Examples of Contact Formulations

3D contact analysis

2D slave node, line master segment element

2D indentation problem

2D slave node, smooth master segment element

2D snooker simulation

3D slave node, triangle master segment element

3D slave node, quadrilateral master segment element

3D slave node, quadrilateral master segment and 2 neighboring nodes element

3D slave triangle and 2 neighboring nodes, triangle master segment element

3D slave triangle, triangle master segment and 2 neighboring nodes element

3D contact analysis

Implementation of Finite Elements in Alternative Numerical Environments

ABAQUS

FEAP

ELFEN

User defined environment interface

Bibliography

- Korelc J. Automation of primal and sensitivity analysis of transient coupled problems. *Computational mechanics*, 44(5):631-649 (2009).
- Korelc J. Direct computation of critical points based on Crout's elimination and diagonal subset test function, *Computers and Structures*, 88:189-197 (2010).
- Korelc J. Automation of the finite element method. V: WRIGGERS, Peter. *Nonlinear finite element methods*. Springer, 483-508 (2008).
- 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-thickness 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

Shared Finite Element Libraries

AceShare

The AceFEM environment comes with the built-in library is a part of the installation and contains the element dll files for the most basic and standard elements. The standard built-in library of the elements is located at directory `$BaseDirectory/Applications/AceFEM/Library/`.

Additional elements can be accessed through the AceShare system. The AceShare system is a file sharing mechanism built in AceFEM that allows:

- browsing the on-line FEM libraries
- downloading 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

Each on-line library can contain:

- ⇒ element dll files,
- ⇒ home page file (HTML file with basic descriptions, links, etc..)
- ⇒ symbolic input used to generate element,
- ⇒ source codes for other environments (FEAP, AceFEM-MDriver, Abaqus, ...)
- ⇒ benchmark tests.

See also:

`SMTSetLibrary` — initializes the library

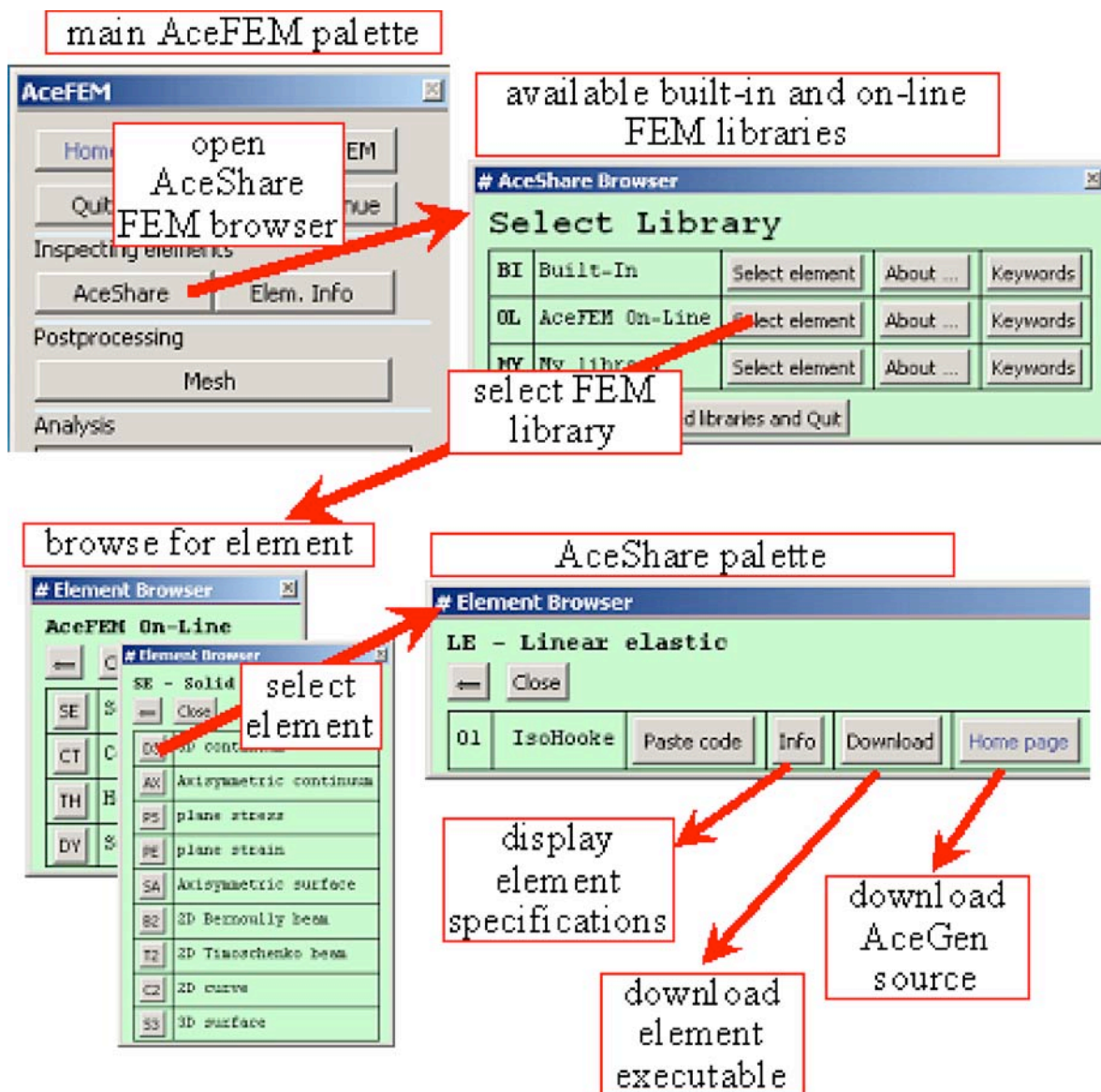
`SMTAddToLibrary` — add new element to library

`SMTLibraryContents` — prepare library for posting Unified Element Code

Unified Element Code — the elements in the libraries are located through unique codes

Accessing elements from shared libraries

The `SMTAnalysis` command automatically locates and downloads finite elements from the built-in and on-line libraries. The required element dll file is located on a base on an Unified Element Code. The libraries can be selected and searched by the use of the finite element browser. The finite element browser can be used to locate the element, paste the element code into problems input data, get informations about material constants, available postprocessing keywords, and also to access element's home page with the links to benchmark tests and source codes.



Selecting and browsing the AceShare libraries.

The information about the available on-line libraries is automatically updated once per day and stored in directory \$BaseDirectory/AceCommon/. The downloading dll files from the on-line libraries are also stored in the same directory \$BaseDirectory/AceCommon/.

The library is defined by the unique code (e.g. OL stands for standard AceFEM on-line library). The standard AceFEM On-line library contains large number of finite elements (solid, thermal,... 2D, 3D,...) with full symbolic input for most of the elements. The number of finite elements included in on-line libraries is a 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

```

<< AceFEM` ;
SMTInputData [ ] ;
SMTAddDomain [ "A", "OL:SEPSQ1ESHYQ1E4NeoHooke",
  { "E *" -> 1000., "v *" -> .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 ] ;
SMTMesh [ "A", "Q1", { 80, 5 }, Array [ { #2, #2 Sin [ #2 // N ] / 20 + 4 #1 } &, { 2, 40 } ] ] ;
SMTAnalysis [ "Output" -> "tmp.out" ] ;

SMTNextStep [ 1, 100 ] ;
While [
  While [ step = SMTConvergence [ 10^-7, 15, { "Adaptive", 8, .01, 300, 500 } ],
    SMTNewtonIteration [ ] ] ;
  SMTStatusReport [ ] ;
  If [ step[[4]] == "MinBound", Print [ "Error:  $\Delta\lambda < \Delta\lambda_{min}$ " ] ] ;
  If [ Not [ step[[1]] ], SMTShowMesh [ "DeformedMesh" -> True,
    "Field" -> "Sxy", "Mesh" -> False, "Contour" -> 20, "Show" -> "Window" ] ] ;
  step[[3]] ,
  If [ step[[1]], SMTStepBack [ ] ] ;
  SMTNextStep [ 1, step[[2]] ]
]

T/ $\Delta$ T=1./1.  $\lambda/\Delta\lambda=100./100.$   $\|\Delta a\|/\|\Psi\|=5.27759 \times 10^{-9}$ 
/  $7.88253 \times 10^{-13}$  Iter/Total=11/11 Status=0/{Convergence}

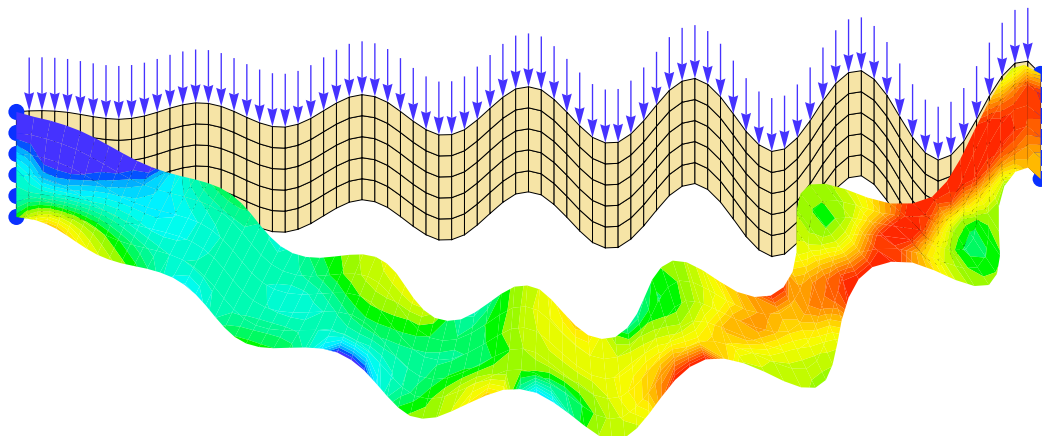
T/ $\Delta$ T=2./1.  $\lambda/\Delta\lambda=195.918/95.9184$   $\|\Delta a\|/\|\Psi\|=6.34201 \times 10^{-13}$ 
/  $8.53098 \times 10^{-13}$  Iter/Total=7/18 Status=0/{Convergence}

T/ $\Delta$ T=3./1.  $\lambda/\Delta\lambda=338.817/142.899$   $\|\Delta a\|/\|\Psi\|=4.49627 \times 10^{-8}$ 
/  $8.59231 \times 10^{-13}$  Iter/Total=6/24 Status=0/{Convergence}

T/ $\Delta$ T=4./1.  $\lambda/\Delta\lambda=500./161.183$   $\|\Delta a\|/\|\Psi\|=3.36995 \times 10^{-10}$ 
/  $9.49991 \times 10^{-13}$  Iter/Total=6/30 Status=0/{Convergence}

Show [ SMTShowMesh [ "Show" -> False, "BoundaryConditions" -> True ],
  SMTShowMesh [ "DeformedMesh" -> True, "Mesh" -> False, "Field" -> "Sxy",
  "Contour" -> 20, "Show" -> False, "Legend" -> False ], PlotRange -> All ]

```



Unified Element Code

The elements in the libraries are located through unique codes. The code is a string of the form:

`"library_code:element_code"` .

If the library code is omitted then the first element with the given *element_code* is selected.

The element code can be further composed from several parts with predefined meaning.

For example the codes for the elements from the standard on-line AceFEM library (OL) consist of the following keywords:

<i>Description</i>	<i>No. of characters in keyword</i>	<i>Examples of keywords</i>
Physical problem	2	SE ≡ Mechanical problems
Physical 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
General model	2	LE == linear elastic solid
Acronym	arbitrary	Q1E4
Sub-model	arbitrary	IsoHooke
Sub-sub-model	arbitrary	Missess
...	arbitrary	...

Keywords for the standard on-line AceFEM library.

For example the "OL:SEPEQ1HRLEPianSumHooke" code represents the well-known Pian-Sumihara element derived for isotropic Hook's material.

Simple AceShare library

Set up library

SMTSetLibrary — initializes the library

```
<< AceGen` ; << AceFEM` ;

SMTSetLibrary["D:/UserLib/"
, "Code" → "MY"
, "Title" → "My library"
, "URL" → "http://www.fgg.uni-lj.si/symech/UserLib/"
, "Keywords" → {
  {"D3"} → "3d elements",
  {"D3", "H1"} → "8 nodeed",
  {"D3", _, "SE"} → "solid",
  {"D3", _, "TH"} → "Thermal"
}
]
```

Create element

```

SMSInitialize["D3H1TH", "Environment" -> "AceFEM"];
SMSTemplate["SMSTopology" -> "H1",
  "SMSDOFGlobal" -> 1, "SMSSymmetricTangent" -> False,
  "SMSGroupDataNames" ->
    {"k0 -conductivity parameter", "k1 -conductivity parameter",
     "k2 -conductivity parameter", "Q -heat source"},
  "SMSDefaultData" -> {1, 0, 0, 0}];
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]];
E = {ξ, η, ζ} + Table[SMSReal[es$$["IntPoints", i, Ig]], {i, 3}];
XI + Table[SMSReal[nd$$[i, "X", j]], {i, SMSNoNodes}, {j, SMSNoDimensions}];
En = {{-1, -1, -1}, {1, -1, -1}, {1, 1, -1}, {-1, 1, -1},
      {-1, -1, 1}, {1, -1, 1}, {1, 1, 1}, {-1, 1, 1}};
NI = Table[1/8 (1 + ξ En[[i, 1]]) (1 + η En[[i, 2]]) (1 + ζ En[[i, 3]]), {i, 1, 8}];
X + SMSFreeze[NI.XI]; Jg = SMSD[X, E]; Jgd = Det[Jg];
φI + SMSReal[Table[nd$$[i, "at", 1], {i, SMSNoNodes}]];
φ = NI.φI;
{k0, k1, k2, Q} + SMSReal[Table[es$$["Data", i], {i, Length[SMSGroupDataNames]}]];
k + k0 + k1 φ + k2 φ2;
λ + SMSReal[rdata$$["Multiplier"]];
wgp + SMSReal[es$$["IntPoints", 4, Ig]];
Dφ = SMSD[φ, X, "Dependency" -> {E, X, SMSInverse[Jg]}];
W =  $\frac{1}{2}$  k Dφ.Dφ - φ λ Q;
SMSDo[
  Rg = Jgd wgp SMSD[W, φI, i, "Constant" -> k];
  SMSExport[SMSResidualSign Rg, p$$[i], "AddIn" -> True];
  SMSDo[
    Kg = SMSD[Rg, φI, j];
    SMSExport[Kg, s$$[i, j], "AddIn" -> True];
    , {j, 1, 8}
  ];
  , {i, 1, 8}
];
SMSEndDo[];
SMSWrite[];

```

File:	D3H1TH.c	Size:	11 024
Methods	No.Formulae	No.Leafs	
SKR	178	2665	

Add element to library

SMTAddToLibrary — add new element to library

```
SMTAddToLibrary[{"D3", "H1", "TH"}]
```

Prepare library for posting

SMTLibraryContents — prepare library for posting

```
SMTLibraryContents[]
```

Please send the URL to AceProducts@fgg.uni-lj.si to make third-party library available to all AceFEM users.

Advanced AceShare library

Set up library

```
<< AceGen` ; << AceFEM` ;

SMTSetLibrary["D:/UserLib/"
, "Code" → "MY"
, "Title" → "My library"
, "URL" → "http://www.fgg.uni-lj.si/symech/UserLib/"
, "Keywords" → {
  {"D3"} → "3d elements",
  {"D3", "H1"} → "8 nodeed",
  {"D3", _, "SE"} → "Solid",
  {"D3", _, "TH"} → "Thermal",
  {"D3", _, "TH", "Lin"} → "constant conductivity",
  {"D3", _, "TH", "Nonlin"} → "nonlinear conductivity"
}
];
```

Create first element

```
code = "D3H1THLin";
keywords = {"D3", "H1", "TH", "Lin"};
material = keywords[[4]];
SMSEvaluateCellsWithTag["AceShareElement",
  "CollectInputStart" → True, "RemoveTag" → True];
SMSEvaluateCellsWithTag["AceShareExample", "RemoveTag" → True];
SMSRecreateNotebook["File" → SMSSessionName, "Close" -> True,
  "Head" → {Cell[SMSSessionName, "Title"], Cell["<<AceGen`;<<AceFEM`";, "Input"]}];
```

Include Tag : AceShareElement (7 cells found, 5 evaluated)

File:	D3H1THLin.c	Size:	11 021
Methods	No.Formulae	No.Leafs	
SKR	194	2602	

Include Tag : AceShareExample (5 cells found, 3 evaluated)

```
SMTAddToLibrary[keywords
, "DeleteOriginal" → True
, "Source" -> SMSSessionName
, "Documentation" -> SMSSessionName
, "Examples" -> SMSSessionName
, "Author" → {"J. Korelc", "http://www.fgg.uni-lj.si/~/jkorelc/"}
];
```

Create second element

```
code = "D3H1THNonlin";
keywords = {"D3", "H1", "TH", "Nonlin"};
material = keywords[[4]];
SMSEvaluateCellsWithTag["AceShareElement",
  "CollectInputStart" → True, "RemoveTag" → True];
SMSEvaluateCellsWithTag["AceShareExample", "RemoveTag" → True];
SMSRecreateNotebook["File" → SMSSessionName, "Close" → True,
  "Head" → {Cell[SMSSessionName, "Title"], Cell["<<AceGen`;<<AceFEM`";, "Input"]}];

Include Tag : AceShareElement (7 cells found, 5 evaluated)
```

File:	D3H1THNonlin.c	Size:	11 040
Methods	No.Formulae	No.Leafs	
SKR	179	2668	

```
Include Tag : AceShareExample (5 cells found, 3 evaluated)
```

```
SMTAddToLibrary[keywords
, "DeleteOriginal" → True
, "Source" → SMSSessionName
, "Documentation" → SMSSessionName
, "Examples" → SMSSessionName
, "Author" → {"J. Korelc", "http://www.fgg.uni-lj.si/~/jkorelc/"}
];
```

Prepare library for posting

```
SMTLibraryContents[]
```

AceGen input

```
(* SMSTagEvaluate -
  evaluated variable code is included into recreated notebook*)
SMSInitialize[SMSTagEvaluate[code], "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "H1",
  "SMSDOFGlobal" → 1, "SMSSymmetricTangent" → False];
SMSStandardModule["Tangent and residual"];

(* SMSTagSwitch - only choosen option is included into recreated notebook*)
SMSTagSwitch[material
, "Lin",
  SMSGroupDataNames = {"k0 -conductivity parameter k0", "Q -heat source"};
  SMSDefaultData = {1, 0};
  {k0, Q} = SMSReal[{es$$["Data", 1], es$$["Data", 2]}];

, "Nonlin",
  SMSGroupDataNames = {"k0 -conductivity parameter", "k1 -conductivity parameter",
    "k2 -conductivity parameter", "Q -heat source"};
  SMSDefaultData = {1, 0, 0, 0};
  {k0, k1, k2, Q} =
    SMSReal[{es$$["Data", 1], es$$["Data", 2], es$$["Data", 3], es$$["Data", 4]}];
];
```

```

SMSDo[Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]];
E = {ξ, η, ζ} + Table[SMSReal[es$$["IntPoints", i, Ig]], {i, 3}];
XI + Table[SMSReal[nd$$[i, "X", j]], {i, SMSNoNodes}, {j, SMSNoDimensions}];
En = {{-1, -1, -1}, {1, -1, -1}, {1, 1, -1}, {-1, 1, -1},
      {-1, -1, 1}, {1, -1, 1}, {1, 1, 1}, {-1, 1, 1}};
NI = Table[1 / 8 (1 + ξ En[[i, 1]]) (1 + η En[[i, 2]]) (1 + ζ En[[i, 3]]), {i, 1, 8}];
X + SMSFreeze[NI.XI]; Jg = SMSD[X, E]; Jgd = Det[Jg];
φI + SMSReal[Table[nd$$[i, "at", 1], {i, SMSNoNodes}]];
φ = NI.φI;

SMSTagSwitch[material
  , "Lin",
  k + k0;
  , "Nonlin",
  k + k0 + k1 φ + k2 φ2;
];

λ + SMSReal[rdata$$["Multiplier"]];
wgp + SMSReal[es$$["IntPoints", 4, Ig]];
Dφ = SMSD[φ, X, "Dependency" -> {E, X, SMSInverse[Jg]}];
W =  $\frac{1}{2}$  k Dφ.Dφ - φ λ Q;
SMSDo[
  Rg = Jgd wgp SMSD[W, φI, i, "Constant" -> k];
  SMSExport[SMSResidualSign Rg, p$$[i], "AddIn" -> True];
  SMSDo[
    Kg = SMSD[Rg, φI, j];
    SMSExport[Kg, s$$[i, j], "AddIn" -> True];
    , {j, 1, 8}
  ];
  , {i, 1, 8}
];
SMSEndDo[];
SMSWrite[];

```

File:	D3H1THNonlin.c	Size:	15 022
Methods	No.Formulae	No.Leafs	
SKR	217	4407	

AceFEM example

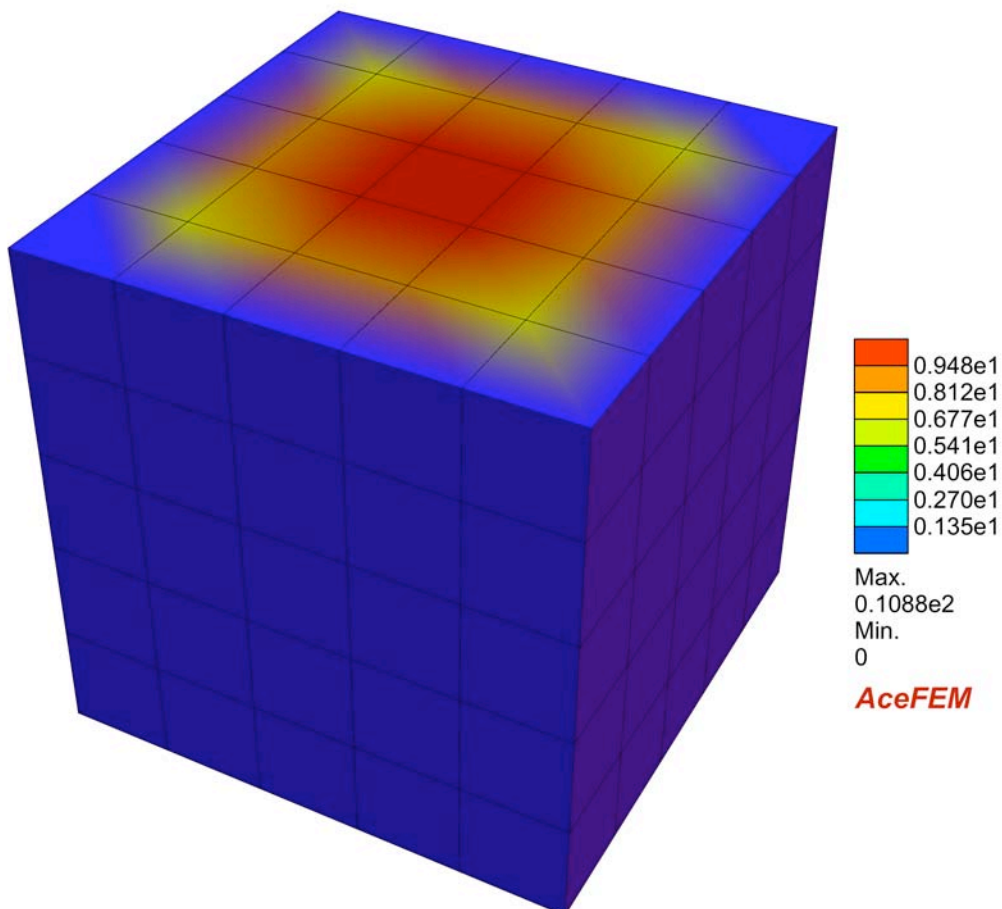
```

SMTInputData[];
Q = 50;
nn = 5;
SMSTagSwitch[material
, "Lin",
SMTAddDomain["cube", SMSTagEvaluate[code], {"k0 *" -> 0.58, "Q *" -> Q}];
, "Nonlin",
SMTAddDomain["cube", SMSTagEvaluate[code],
{"k0 *" -> 0.1, "k1 *" -> 0.02, "k2 *" -> 0.003, "Q *" -> Q}];
];
SMTAddEssentialBoundary[
{"X" == -0.5 || "X" == 0.5 || "Y" == -0.5 || "Y" == 0.5 || "Z" == 0. &, 1 -> 0}];
SMTMesh["cube", "H1", {nn, nn, nn}, {
{{{-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}}}
}];
SMTAnalysis[];

SMTNextStep[0, 1];
While[step = SMTConvergence[10^-8, 10], SMTNewtonIteration[]];

SMTShowMesh["Field" -> SMTPost[1]]

```



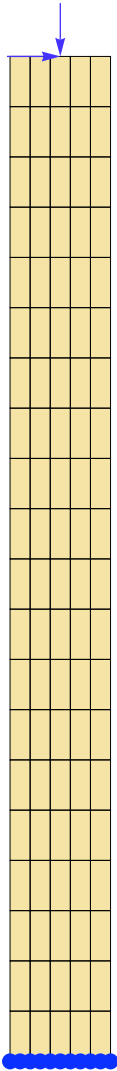
Basic AceFEM Examples

Bending of the column (path following procedure, animations, 2D solids)

Calculate the typical load/deflection curve of the column subjected to the constant horizontal force $H=10$ and variable vertical force $V=-\lambda 10$.

```
<< AceFEM`;  
SMTInputData[];  
SMTAddDomain["Ω", "SEPSQ2DFHYQ2NeoHooke", {"E *" -> 21 000, "ν *" -> 0.3}];  
SMTAddEssentialBoundary["Y" == 0 &, 1 -> 0, 2 -> 0];  
SMTAddNaturalBoundary["X" == 0 && "Y" == 20 &, 2 -> -10];  
SMTAddInitialBoundary["X" == 0 && "Y" == 20 &, 1 -> 10];  
SMTMesh["Ω", "Q2", {20, 5}, {{{1, 0}, {1, 20}}, {{-1, 0}, {-1, 20}}}];  
SMTAnalysis[];
```

```
SMTShowMesh["BoundaryConditions" → True]
```

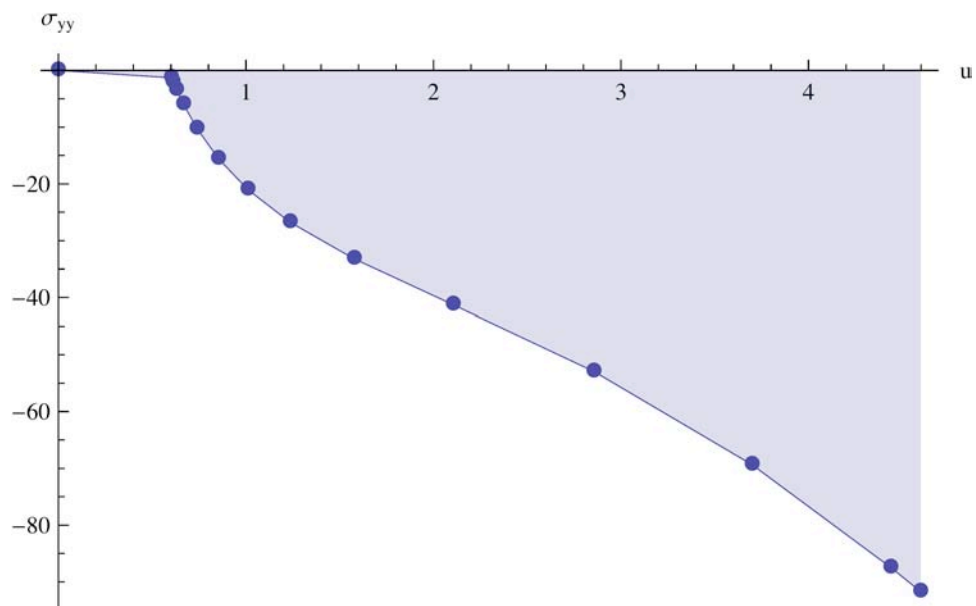
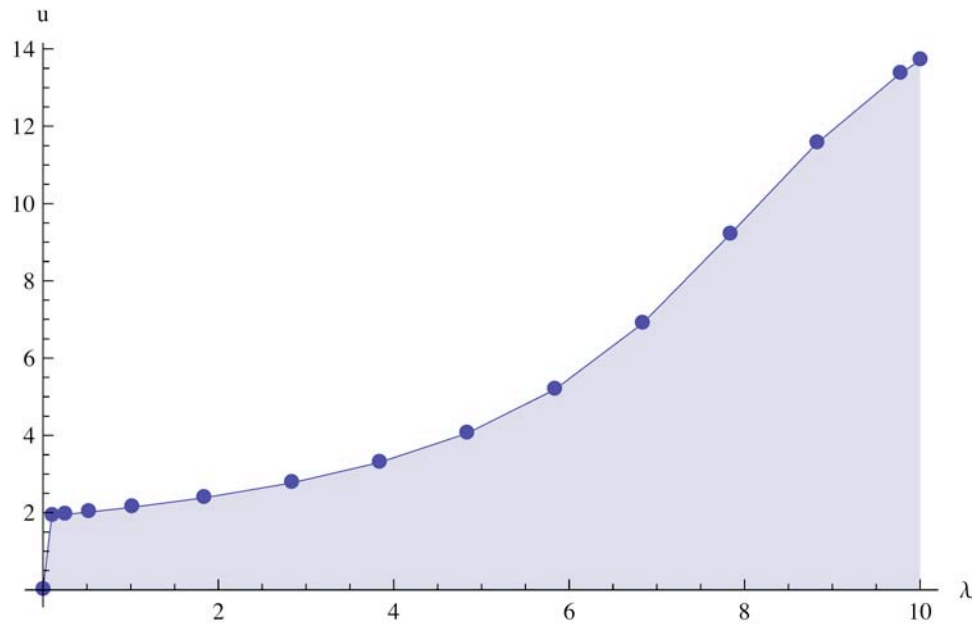


```

λucurve = {{0, 0}};
uσcurve = {{0, 0}};
SMTNextStep[1, .1];
While[
  While[step = SMTConvergence[10^-8, 15, {"Adaptive", 8, .0001, 1, 10}],
    SMTNewtonIteration[]];
  If[Not[step[[1]]],
    SMTShowMesh["DeformedMesh" → True, "Field" → "Syy",
      "Show" → "Window" | {"Animation", "column"}, "BoundaryConditions" → True];
    AppendTo[λucurve, {SMTData["Multiplier"], SMTPostData["u", {0, 20}]}];
    AppendTo[uσcurve, SMTPostData[{"u", "Syy"}, {0, 10}]];
  ];
  If[step[[4]] === "MinBound", SMTStatusReport["Error: Δλ < Δλmin"];];
  step[[3]]
  , If[step[[1]], SMTStepBack[]];];
SMTNextStep[1, step[[2]]
];

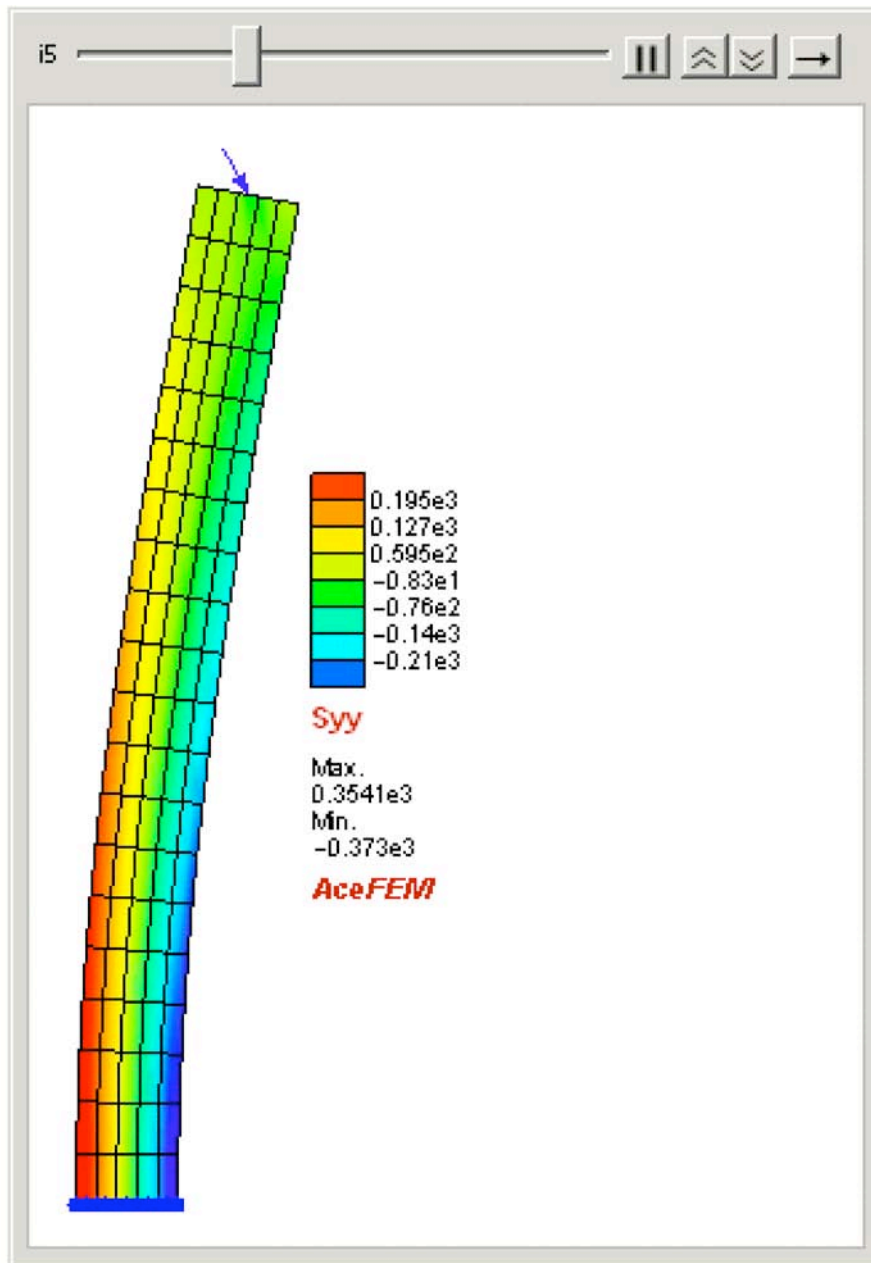
```

```
ListLinePlot[ $\lambda$ curve, PlotRange  $\rightarrow$  All, Filling  $\rightarrow$  Axis,  
  AxesLabel  $\rightarrow$  {" $\lambda$ ", "u"}, PlotMarkers  $\rightarrow$  Automatic]  
ListLinePlot[u $\sigma$ curve, PlotRange  $\rightarrow$  All, Filling  $\rightarrow$  Axis,  
  AxesLabel  $\rightarrow$  {"u", " $\sigma_{yy}$ "}, PlotMarkers  $\rightarrow$  Automatic]
```

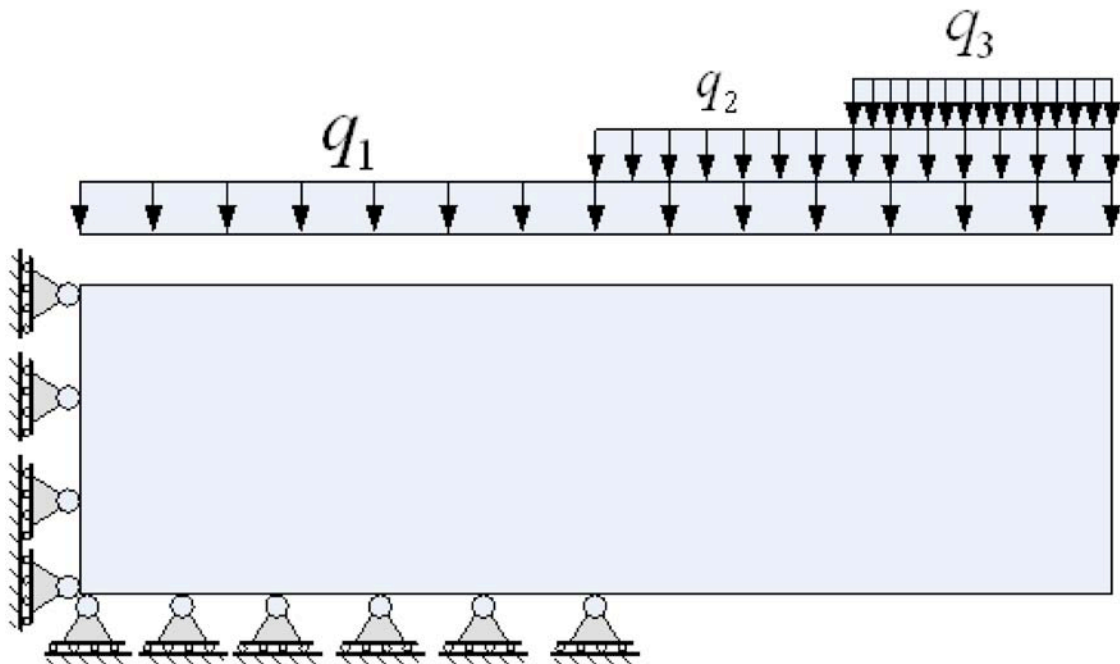


```
SMTMakeAnimation["column"]
```

Out [50]=



Boundary conditions (2D solid)



■ Solution 1: based on line segment node selector and continuous loads

```

<< AceFEM` ;
SMTInputData[];
L = 100; H = 20;
q1 = 2; q2 = 1; q3 = 1;
nx = 40; ny = 10;
SMTAddDomain["A", "BI:SEPSQ1DFHYQ1NeoHooke",
  {"E *" -> 1000., "v *" -> .49, "t *" -> 1.}];
SMTMesh["A", "Q1", {40, 10}, {{{0, 0}, {L, 0}}, {{0, H}, {L, H}}];
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[];

```

```

SMTNextStep[1, 0.1];
While[
  While[step = SMTConvergence[10^-7, 15, {"Adaptive", 8, .01, 0.5, 1}],
    SMTNewtonIteration[]];
  SMTStatusReport[];
  If[step[[4]] == "MinBound", Print["Error:  $\Delta\lambda < \Delta\lambda_{min}$ "]];
  If[Not[step[[1]]], SMTShowMesh["DeformedMesh" → True,
    "Field" → "Sxy", "Mesh" → False, "Contour" → 20, "Show" → "Window"]];
  step[[3]],
  If[step[[1]], SMTStepBack[]];
  SMTNextStep[1, step[[2]]]
]

```

```

T/ $\Delta$ T=1./1.  $\lambda/\Delta\lambda=0.1/0.1$   $\|\Delta a\|/\|\Psi\|=2.2186 \times 10^{-9}$ 
/2.89266  $\times 10^{-12}$  Iter/Total=5/5 Status=0/{Convergence}

```

```

T/ $\Delta$ T=2./1.  $\lambda/\Delta\lambda=0.281633/0.181633$   $\|\Delta a\|/\|\Psi\|=7.93853 \times 10^{-13}$ 
/2.66669  $\times 10^{-12}$  Iter/Total=6/11 Status=0/{Convergence}

```

```

T/ $\Delta$ T=3./1.  $\lambda/\Delta\lambda=0.585589/0.303957$   $\|\Delta a\|/\|\Psi\|=4.3351 \times 10^{-9}$ 
/2.78672  $\times 10^{-12}$  Iter/Total=6/17 Status=0/{Convergence}

```

```

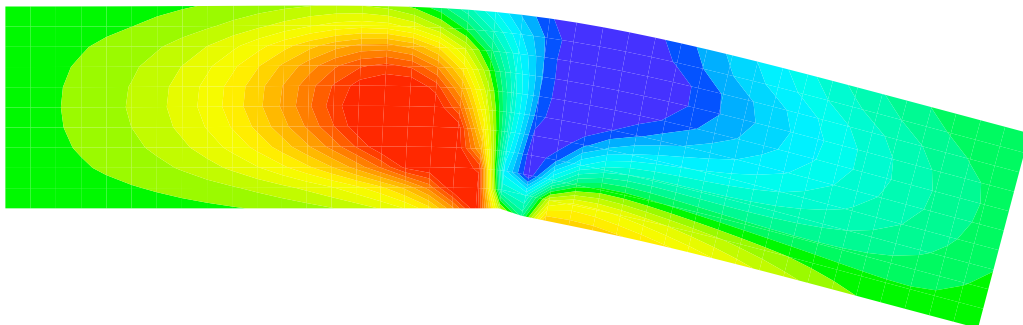
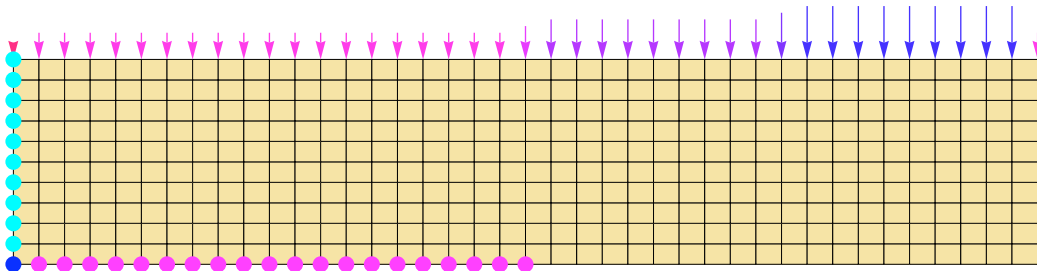
T/ $\Delta$ T=4./1.  $\lambda/\Delta\lambda=1./0.414411$   $\|\Delta a\|/\|\Psi\|=2.2437 \times 10^{-13}$ 
/2.75836  $\times 10^{-12}$  Iter/Total=7/24 Status=0/{Convergence}

```

```

Column[{SMTShowMesh["Show" → False, "BoundaryConditions" → True],
  SMTShowMesh["DeformedMesh" → True, "Mesh" → False,
    "Field" → "Sxy", "Contour" → 20, "Show" → False, "Legend" → False]}]

```



Solution 2: based on general node selector and calculated nodal forces

```

<< AceFEM` ;
SMTInputData[];
L = 100; H = 20;
q1 = 2; q2 = 1; q3 = 1;
nx = 40; ny = 10;
SMTAddDomain["A", "OL:SEPSQ1ESHYQ1E4NeoHooke",
  {"E *" -> 1000., "v *" -> .49, "t *" -> 1.}];
SMTMesh["A", "Q1", {40, 10}, {{{0, 0}, {L, 0}}, {{0, H}, {L, H}}];
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[];

SMTNextStep[1, 0.1];
While[
  While[step = SMTConvergence[10^-7, 15, {"Adaptive", 8, .01, 0.5, 1}],
    SMTNewtonIteration[]];
  SMTStatusReport[];
  If[step[[4]] == "MinBound", Print["Error:  $\Delta\lambda < \Delta\lambda_{min}$ "]];
  If[Not[step[[1]]], SMTShowMesh["DeformedMesh" -> True,
    "Field" -> "Sxy", "Mesh" -> False, "Contour" -> 20, "Show" -> "Window"]];
  step[[3]],
  If[step[[1]], SMTStepBack[]];
  SMTNextStep[1, step[[2]]]
]

T/ $\Delta$ T=1./1.  $\lambda/\Delta\lambda=0.1/0.1$   $\|\Delta a\|/\|\Phi\|=3.82764 \times 10^{-9}$ 
/2.47179  $\times 10^{-12}$  Iter/Total=5/5 Status=0/{Convergence}

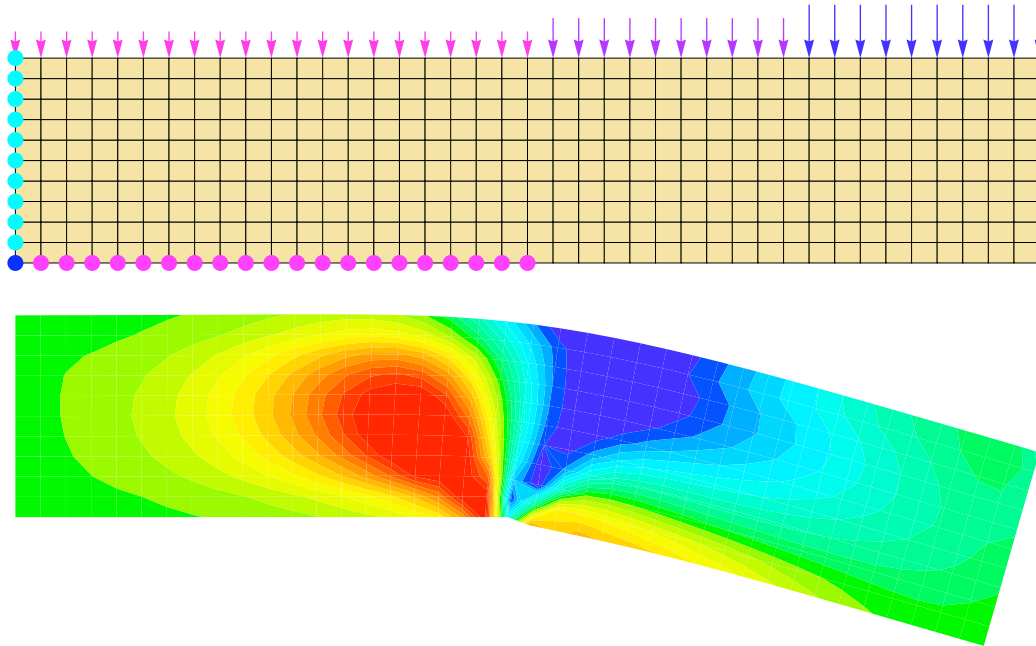
T/ $\Delta$ T=2./1.  $\lambda/\Delta\lambda=0.281633/0.181633$   $\|\Delta a\|/\|\Phi\|=2.39379 \times 10^{-12}$ 
/2.46301  $\times 10^{-12}$  Iter/Total=6/11 Status=0/{Convergence}

T/ $\Delta$ T=3./1.  $\lambda/\Delta\lambda=0.585589/0.303957$   $\|\Delta a\|/\|\Phi\|=1.30189 \times 10^{-8}$ 
/2.68259  $\times 10^{-12}$  Iter/Total=6/17 Status=0/{Convergence}

T/ $\Delta$ T=4./1.  $\lambda/\Delta\lambda=1./0.414411$   $\|\Delta a\|/\|\Phi\|=2.38454 \times 10^{-12}$ 
/2.55997  $\times 10^{-12}$  Iter/Total=7/24 Status=0/{Convergence}

```

```
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)

```
<< AceFEM` ;

SMTInputData[];
SMTAddDomain["A", "SEPSQ1DFLEQ1Hooke",
  {"E *" -> 11. 107, "ν *" -> 0.3, "τ *" -> 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}}
  ];
SMTAddNaturalBoundary["X" == 6 &, 2 -> -0.5];
SMTAddEssentialBoundary["X" == 0 &, 1 -> 0, 2 -> 0];
```

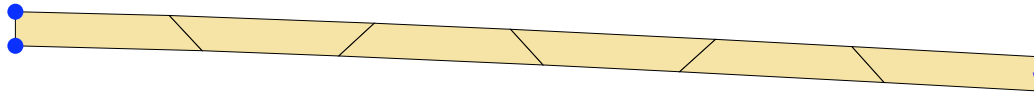


```

SMTAnalysis[];
SMTNextStep[1, 1];
While[SMTConvergence[10^-12, 10], SMTNewtonIteration[]];
SMTNodeData["X" == 6 &, "at"]
SMTShowMesh["BoundaryConditions" -> True, "DeformedMesh" -> True, "Scale" -> 1000]

{{{-5.77117 × 10-6, -0.000264499}, {5.15829 × 10-6, -0.000264364}}

```

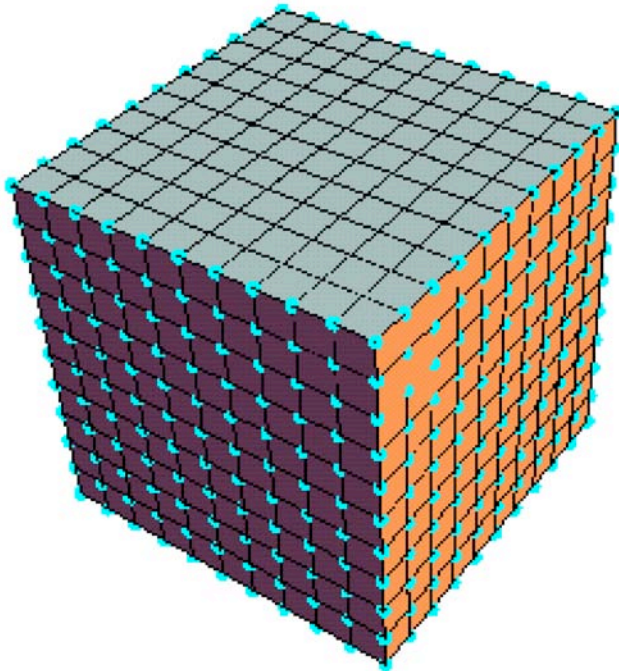


Solution Convergence Test

The domain of the problem is $[-0.5, 0.5] \times [-0.5, 0.5] \times [0, 1]$ cube. On all sides, apart from the upper surface, the constant temperature $\phi=0$ 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 $10 \times 10 \times 10$ 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 .



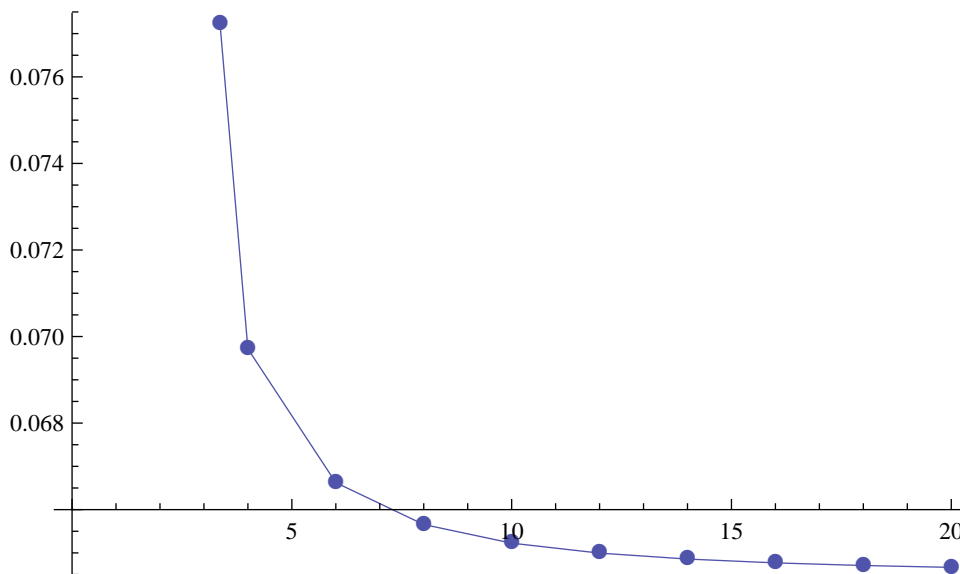
In order to assess the convergence properties the problem is analyzed with the set of meshes from $2 \times 2 \times 2$ to $20 \times 20 \times 20$. The solution at the center of the cube (0,0,0.5) is observed.

```
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}];
  SMTMesh["cube", "H1", {i, i, i},
    {{{{-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}}}}];
  SMTAnalysis["Solver" -> 5];
  SMTNextStep[1, 1];
  While[SMTConvergence[10^-12, 10], SMTNewtonIteration[]];
  {i, SMTPostData["Temp*", {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}}
```

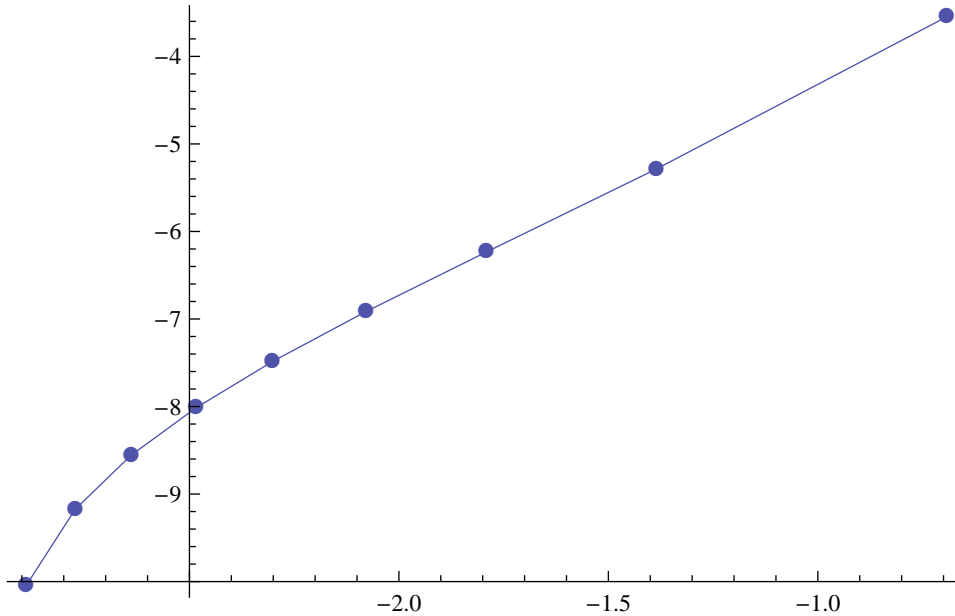
The figure shows rapid convergence of the solution in the centre of the cube with the mesh refinement.

```
ListLinePlot[s, PlotMarkers -> Automatic]
```



More about the convergence properties of the element can be observed from the $\text{Log}[\text{characteristic mesh size}]/\text{Log}[\text{error}]$ plot. The solution obtained with the $20 \times 20 \times 20$ 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.

```
ListLinePlot[Map[Log[{1/#[[1]], Abs[s[[-1, 2]] - #[[2]]}]] &, Drop[s, -1]],
  PlotMarkers -> Automatic]
```



Postprocessing (3D heat conduction)

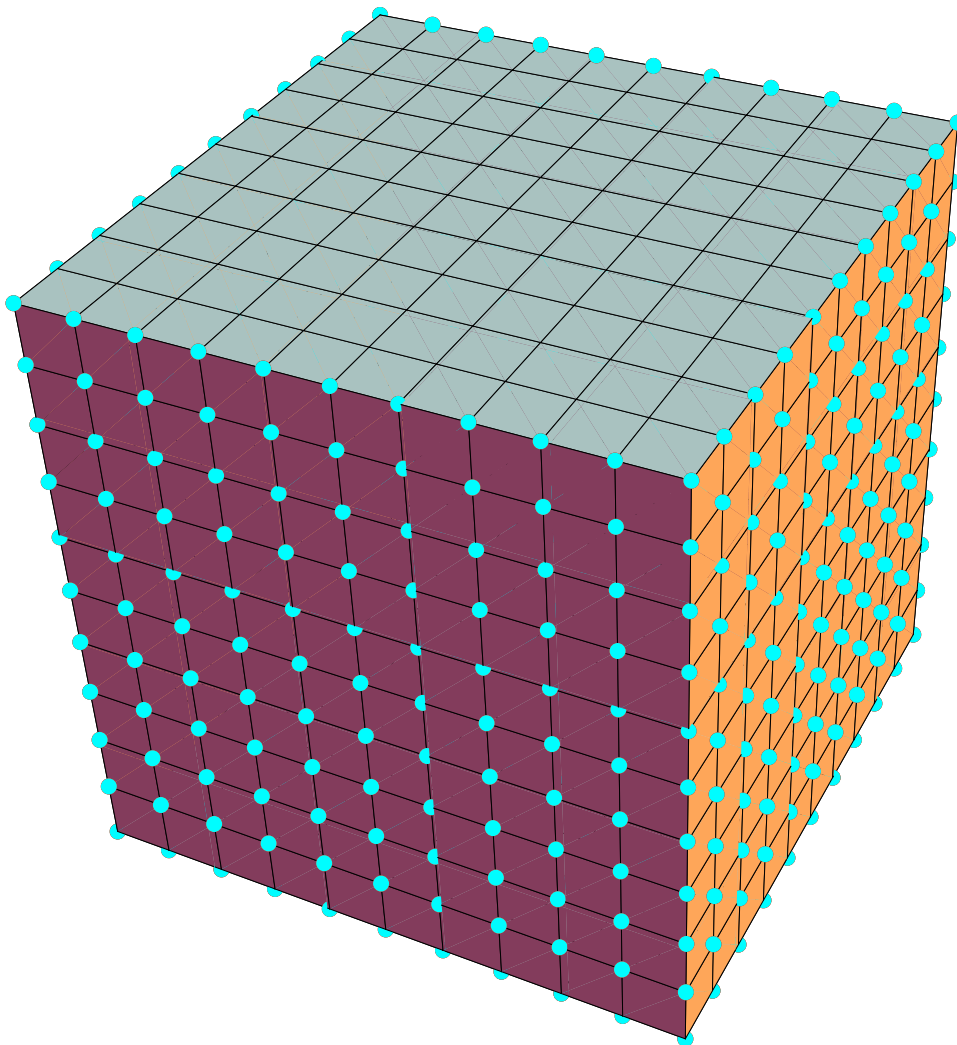
The domain of the problem is $[-0.5, 0.5] \times [-0.5, 0.5] \times [0, 1]$ cube. On all sides, apart from the upper surface, the constant temperature $\phi=0$ is prescribed. The upper surface is isolated so that there is no heat flow over the boundary ($\bar{q}=0$). There exists a constant heat source $Q=1$ 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 $10 \times 10 \times 10$ 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 `SMTMesh` function generates nodes and elements for a cube discretized by the $10 \times 10 \times 10$ mesh. The mesh and the boundary conditions are also depicted.

```
<< 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}];
SMTMesh["cube", "H1", {10, 10, 10}, {
  {{-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}}
}];
SMTAnalysis[];
SMTShowMesh["BoundaryConditions" -> True, "Mesh" -> True, "Elements" -> True]
```



Here the problem is analyzed and the contour plot of the temperature distribution is depicted.

```
SMTNextStep[0, 1];
While[SMTConvergence[10^-12, 10], SMTNewtonIteration[]; SMTStatusReport[]];
SMTShowMesh["Field" -> "Temperature", "Mesh" -> False, "Contour" -> True]

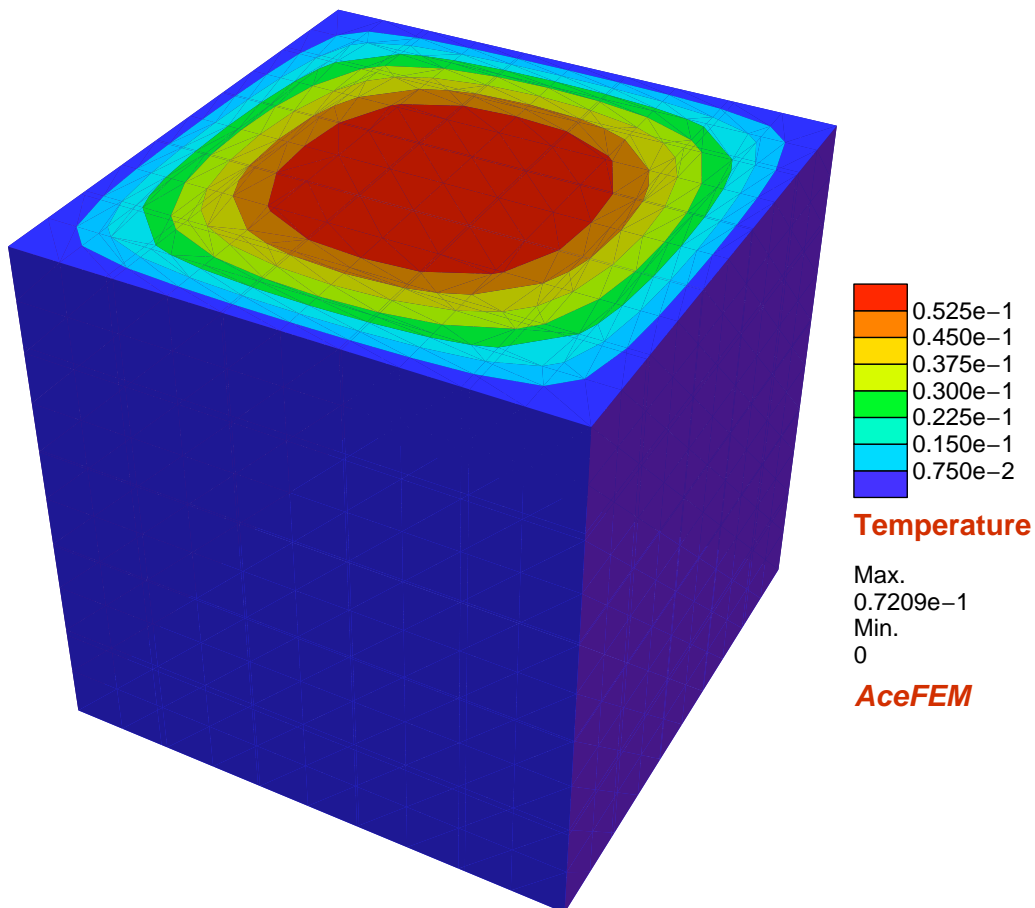
T/ΔT=0./0. λ/Δλ=1./1. ||Δa||/||Ψ||=
0.039126/0.000961769 Iter/Total=1/1 Status=0/{}

T/ΔT=0./0. λ/Δλ=1./1. ||Δa||/||Ψ||=
0.000117723/3.304 × 10-6 Iter/Total=2/2 Status=0/{}

T/ΔT=0./0. λ/Δλ=1./1. ||Δa||/||Ψ||=
2.01035 × 10-9/9.4527 × 10-11 Iter/Total=3/3 Status=0/{}

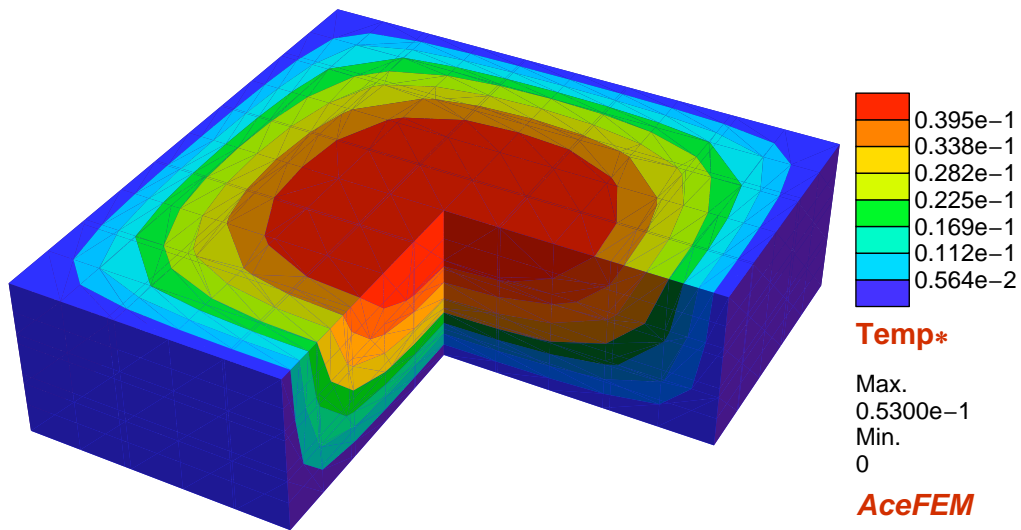
T/ΔT=0./0. λ/Δλ=1./1. ||Δa||/||Ψ||=
2.95684 × 10-18/6.9094 × 10-19 Iter/Total=4/4 Status=0/{}

```

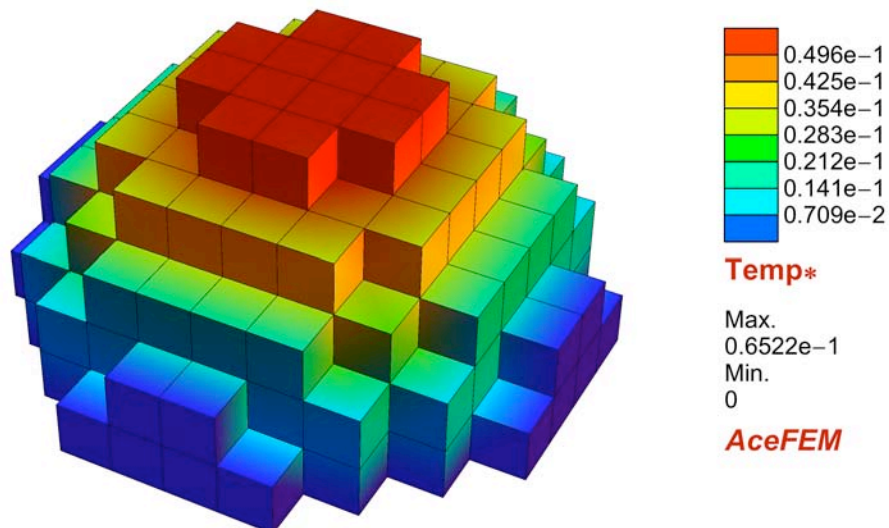


Various views on the results can be obtained by the "Zoom" option.

```
SMTShowMesh["Field" -> "Temp*", "Mesh" -> False,
  "Contour" -> True, "Zoom" -> ("Z" <= 0.3 && ("X" <= 0 || "Y" ≥ 0) & ) ]
```

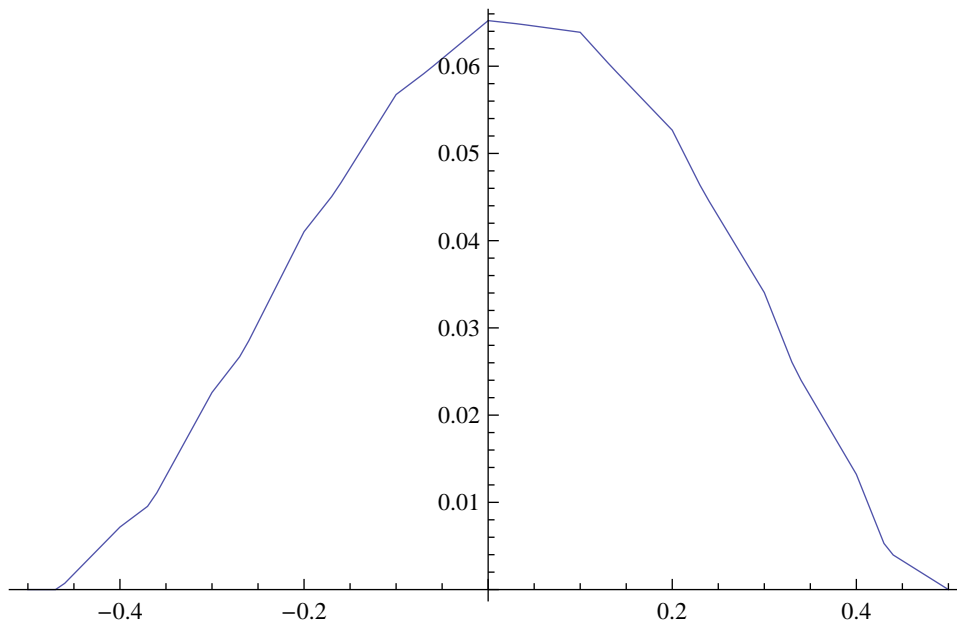


```
SMTShowMesh["Field" -> "Temp*", "Zoom" -> ("X" ^ 2 + "Y" ^ 2 + "Z" ^ 2 ≤ .3 & ) ]
```



Here the temperature distribution along the central diagonal of the cube.

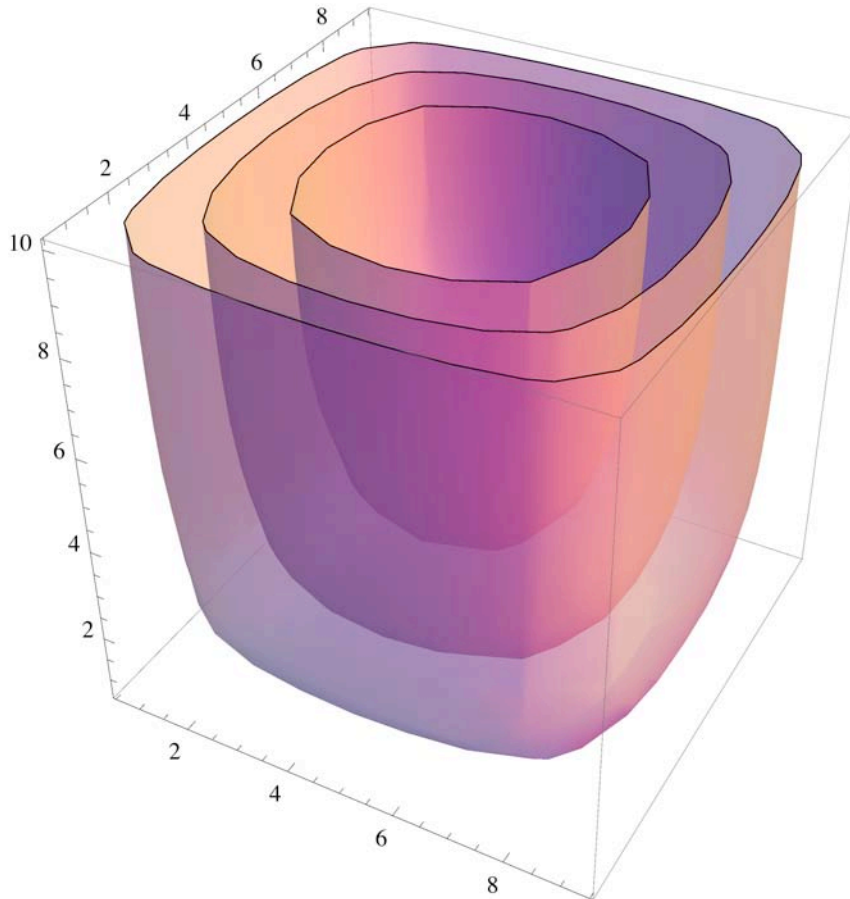
```
ListLinePlot[{Table[ $\xi$ , { $\xi$ , -0.5, 0.5, .01}],  
SMTPostData["Temp*", Table[{ $\xi$ ,  $\xi$ ,  $\xi + 0.5$ }, { $\xi$ , -0.5, 0.5, .01}]]] // Transpose]
```



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.

```
ListContourPlot3D[
  Transpose[Partition[Partition[SMTPost["Temp*"], 11], 11], {3, 2, 1}],
  Contours -> 3, ContourStyle -> Opacity[0.6], Mesh -> False]
```



Basic 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,
- ⇒ 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 hyperelastic Neo-Hooke type strain energy potential,

$$W = \frac{\lambda}{2} (\det \mathbf{F} - 1)^2 + \mu \left(\frac{\text{Tr}[\mathbf{C}] - 3}{2} - \text{Log}[\det \mathbf{F}] \right),$$

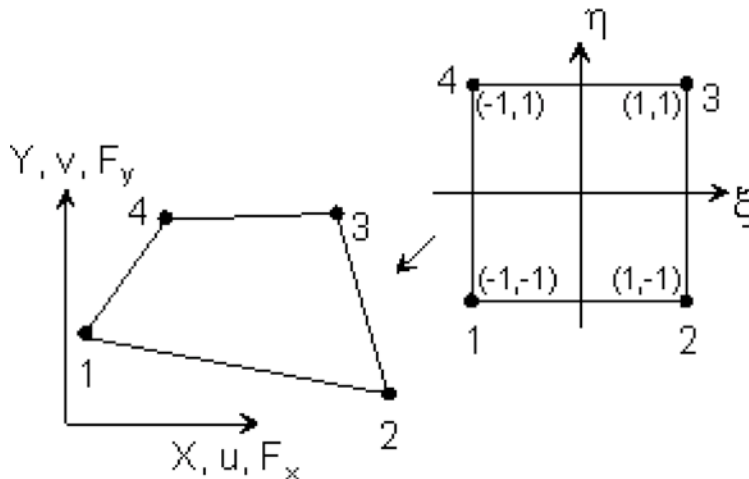
where $\mathbf{C} = \mathbf{F}^T \mathbf{F}$ is right Cauchy-Green tensor, $\mathbf{F} = \mathbf{I} + \nabla \mathbf{u}$ is deformation gradient,

\mathbf{u} is displacements field, Ω_0 is the initial domain of the problem and λ, μ are the first and the second Lamé's material constants.

- ⇒ $W^{\text{ext}} = -\rho_0 \mathbf{u} \cdot \bar{\mathbf{b}}$ is potential of body forces where $\bar{\mathbf{b}}$ is force per unit mass nad ρ_0 density in initial configuration.

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.



```
<< "AceGen`";
SMSInitialize["ExamplesHypersolid2D", "Environment" -> "AceFEM"];
SMSTemplate["SMSTopology" -> "Q1", "SMSSymmetricTangent" -> True,
  "SMSGroupDataNames" -> {"E -elastic modulus", "v -poisson ratio", "t -thickness",
    "rho0 -density", "bX -force per unit mass X", "bY -force per unit mass Y"},
  "SMSDefaultData" -> {21 000, 0.3, 1, 1, 0, 0}
];
```

Definitions of geometry, kinematics, strain energy ...

```

ElementDefinitions[] := (
  E = {ξ, η, ζ} ⊢ Table[SMSReal[es$$["IntPoints", i, Ig]], {i, 3}];
  XI ⊢ Table[SMSReal[nd$$[i, "X", j]], {i, SMSNoNodes}, {j, SMSNoDimensions}];
  NI = 1 / 4 {(1 - ξ) (1 - η), (1 + ξ) (1 - η), (1 + ξ) (1 + η), (1 - ξ) (1 + η)};
  X ⊢ SMSFreeze[Append[NI.XI, ζ]];
  Jg ⊢ SMSD[X, E]; Jgd ⊢ Det[Jg];
  uI ⊢ SMSReal[Table[nd$$[i, "at", j], {i, SMSNoNodes}, {j, SMSNoDimensions}]];
  pe = Flatten[uI]; u ⊢ Append[NI.uI, 0];
  Dg ⊢ SMSD[u, X, "Dependency" → {E, X, SMSInverse[Jg]}];
  SMSFreeze[F, IdentityMatrix[3] + Dg, "KeepStructure" → True];
  JF ⊢ Det[F]; Cg ⊢ Transpose[F].F;
  {Em, ν, tξ, ρ0, bX, bY} ⊢
    SMSReal[Table[es$$["Data", i], {i, Length[SMSGGroupDataNames]}]];
  {λ, μ} ⊢ SMSHookeToLame[Em, ν]; bb ⊢ {bX, bY, 0};
  W = 1 / 2 λ (JF - 1) ^ 2 + μ (1 / 2 (Tr[Cg] - 3) - Log[JF]);
)

```

"Tangent and residual" user subroutine

```

SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]];
ElementDefinitions[];
wgp ⊢ SMSReal[es$$["IntPoints", 4, Ig]];
SMSDo[
  Rg ⊢ Jgd tξ wgp SMSD[W - ρ0 u.bb, pe, i];
  SMSExport[SMSResidualSign Rg, p$$[i], "AddIn" → True];
  SMSDo[
    Kg ⊢ SMSD[Rg, pe, j];
    SMSExport[Kg, s$$[i, j], "AddIn" → True];
    , {j, i, 8}];
    , {i, 1, 8}];
  SMSEndDo[];

```

"Postprocessing" user subroutine

```

SMSStandardModule["Postprocessing"];
SMSDo[Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]];
ElementDefinitions[];
SMSNPostNames = {"DeformedMeshX", "DeformedMeshY", "u", "v"};
SMSExport[Table[Join[uI[[i]], uI[[i]], {i, SMSNoNodes}], npost$$];
Eg ⊢ 1 / 2 (Cg - IdentityMatrix[3]);
σ ⊢ (1 / JF) * SMSD[W, F, "IgnoreNumbers" -> True].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[];

```

Code generation

```
SMSwrite[];
```

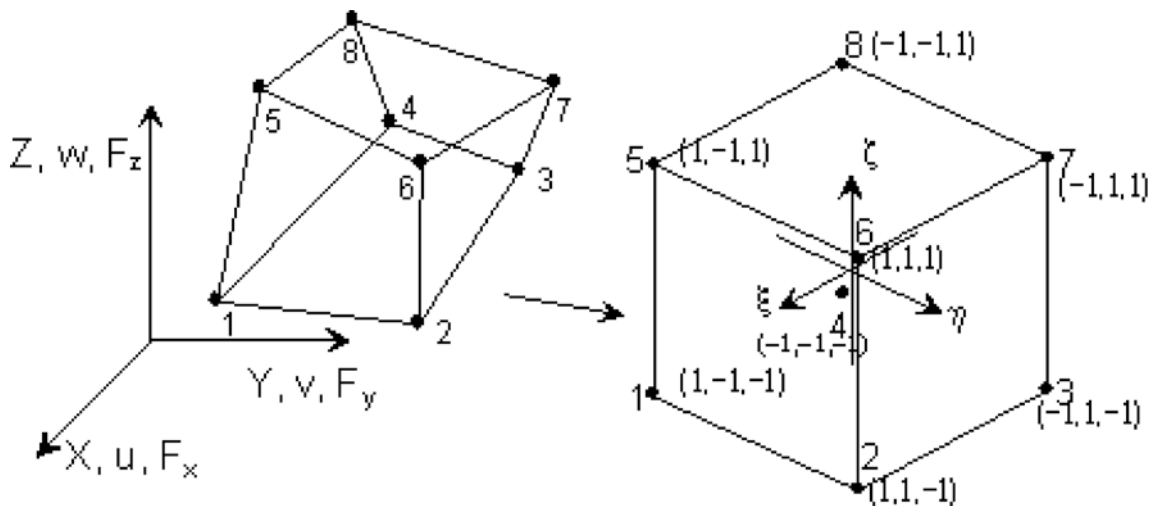
File:	ExamplesHypersolid2D.c	Size:	11 064
Methods	No. Formulae	No. Leafs	
SKR	92	1439	
SPP	68	967	

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,
- ⇒ isoparametric mapping from the reference to the actual frame,



- ⇒ global unknowns are displacements of the nodes,

$$u = u_i N_i, \quad v = v_i N_i, \quad w = w_i N_i$$

- ⇒ enhanced strain formulation to improve shear and volumetric locking response,

$$\Delta \mathbf{u} = \begin{pmatrix} u_{,X} & u_{,Y} & u_{,Z} \\ v_{,X} & v_{,Y} & v_{,Z} \\ w_{,X} & w_{,Y} & w_{,Z} \end{pmatrix}$$

$$\mathbf{D} = \Delta \mathbf{u} + \frac{\text{Det}[J_0]}{\text{Det}[J]} \begin{pmatrix} \xi \alpha_1 & \eta \alpha_2 & \zeta \alpha_3 \\ \xi \alpha_4 & \eta \alpha_5 & \zeta \alpha_6 \\ \xi \alpha_7 & \eta \alpha_8 & \zeta \alpha_9 \end{pmatrix} \cdot \mathbf{J}_0^{-1}$$

where $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_9\}$ are internal degrees of freedom

- ⇒ 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 $C = F^T F$ is right Cauchy-Green tensor, $F = I + D$ is enhanced deformation gradient.

⇒ $W^{\text{ext}} = -\rho_0 \mathbf{u} \cdot \bar{\mathbf{b}}$ is potential of body forces where $\bar{\mathbf{b}}$ is force per unit mass nad ρ_0 density in initial configuration.

⇒ eliminate internal degrees of freedom at the element level by the static condensation procedure (see Elimination of local unknowns)

■ Solution

```
<< "AceGen`";
SMSInitialize["ExamplesHypersolid3D", "Environment" -> "AceFEM"];
SMSTemplate["SMSTopology" -> "H1",
  "SMSNoDOFCondense" -> 9, "SMSGroupDataNames" -> {"E -elastic modulus",
  "v -poisson ratio", "rho0 -density", "bX -force per unit mass X",
  "bY -force per unit mass Y", "bZ -force per unit mass Z"},
  "SMSDefaultData" -> {21 000, 0.3, 1, 0, 0, 0}];
```

TestExamples

```
ElementDefinitions[] := (
  E = {ξ, η, ζ} Table[SMSReal[es$$["IntPoints", i, Ig]], {i, 3}];
  XI Table[SMSReal[nd$$[i, "X", j]], {i, 8}, {j, 3}];
  En = {{-1, -1, -1}, {1, -1, -1}, {1, 1, -1}, {-1, 1, -1},
  {-1, -1, 1}, {1, -1, 1}, {1, 1, 1}, {-1, 1, 1}};
  NI Table[1 / 8 (1 + ξ En[i, 1]) (1 + η En[i, 2]) (1 + ζ En[i, 3]), {i, 1, 8}];
  X SMSFreeze[NI.XI]; Jg SMSD[X, E]; Jgd Det[Jg];
  uI SMSReal[Table[nd$$[i, "at", j], {i, 8}, {j, 3}]];
  pe Flatten[uI]; u NI.uI;
  Dg SMSD[u, X, "Dependency" -> {E, X, SMSInverse[Jg]}];
  J0 SMSReplaceAll[Jg, {ξ -> 0, η -> 0, ζ -> 0}]; J0d Det[J0];
  ae Table[SMSReal[ed$$["ht", i]], {i, SMSNoDOFCondense}];
  ph Join[pe, ae];
  HbE = (


|           |           |           |
|-----------|-----------|-----------|
| ξ ae[[1]] | η ae[[2]] | ξ ae[[3]] |
| ξ ae[[4]] | η ae[[5]] | ξ ae[[6]] |
| ξ ae[[7]] | η ae[[8]] | ξ ae[[9]] |


); Hb =  $\frac{J0d}{Jgd}$  HbE.SMSInverse[J0];
  F SMSFreeze[IdentityMatrix[3] + Dg + Hb];
  JF Det[F]; Cg Transpose[F].F; {Em, v, ρ0, bX, bY, bZ} SMSReal[Table[es$$["Data", i], {i, Length[SMSGroupDataNames]}]];
  {λ, μ} SMSHookeToLame[Em, v]; bb {bX, bY, bZ};
  W = 1 / 2 λ (JF - 1) ^ 2 + μ (1 / 2 (Tr[Cg] - 3) - Log[JF]);
  wgp SMSReal[es$$["IntPoints", 4, Ig]];
)
```

```

SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]];
ElementDefinitions[];
SMSDo[
  Rg = Jgd wgp SMSD[W - ρ0 u.bb, ph, i];
  SMSEExport[SMSResidualSign Rg, p$$[i], "AddIn" → True];
  SMSDo[
    Kg = SMSD[Rg, ph, j];
    SMSEExport[Kg, s$$[i, j], "AddIn" → True];
    , {j, i, SMSNoAllDOF}];
    , {i, 1, SMSNoAllDOF}];
  SMSEndDo[];

SMSStandardModule["Postprocessing"];
SMSDo[Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]];
ElementDefinitions[];
SMSNPostNames = {"DeformedMeshX", "DeformedMeshY", "DeformedMeshZ", "u", "v", "w"};
SMSEExport[Table[Join[uI[i], uI[i]], {i, SMSNoNodes}], npost$$];
Eg = 1 / 2 (Cg - IdentityMatrix[3]);
σ = (1 / JF) * SMSD[W, F] . Transpose[F];
SMSGPostNames = {"Sxx", "Sxy", "Sxz", "Syx", "Syy", "Syz", "Szx", "Szy", "Szz",
  "Exx", "Exy", "Exz", "Eyx", "Eyy", "Eyz", "Ezx", "Ezy", "Ezz"};
SMSEExport[{σ, Eg} // Flatten, gpost$$[Ig, #] &];
SMSEndDo[];

SMSWrite[];

```

Elimination of local unknowns requires additional memory. Corresponding constants are set to:

```

SMSCondensationData = {ed$$[ht, 1], ed$$[ht, 10],
  ed$$[ht, 19], ed$$[ht, 235]}
SMSNoTimeStorage = 234 + 9 idata$$[NoSensParameters]

```

See also: [Elimination of local unknowns](#)

File:	ExamplesHypersolid3D.c	Size:	35 793
Methods	No.Formulae	No.Leafs	
SKR	298	6683	
SPP	232	4036	

■ Cantilever beam example

```

<< AceFEM` ;
SMTInputData[];
SMTAddDomain["A", "ExamplesHypersolid3D", {"E *" -> 1000., "ν *" -> .3}];
SMTAddEssentialBoundary[{ "X" == 0 &, 1 -> 0, 2 -> 0, 3 -> 0}, {"X" == 10 &, 3 -> -1}];
SMTMesh["A", "H1", {15, 6, 6}, {{{{0, 0, 0}, {10, 0, 0}}, {{0, 2, 0}, {10, 2, 0}}},
  {{{0, 0, 3}, {10, 0, 2}}, {{0, 2, 3}, {10, 2, 2}}}}];
SMTAnalysis[];

```

```

SMTNextStep[1, 1];
While[
  While[step = SMTConvergence[10^-8, 10, {"Adaptive", 8, .001, 1, 5}],
    SMTNewtonIteration[]];
  SMTStatusReport[];
  If[step[[4]] == "MinBound", Print["Error:  $\Delta\lambda < \Delta\lambda_{min}$ "]];
  step[[3]]
  , If[step[[1]], SMTStepBack[]];
  SMTNextStep[1, step[[2]]]
];

T/ $\Delta T$ =1./1.  $\lambda/\Delta\lambda$ =1./1.  $\| \Delta a \| / \| \Psi \|$ = $1.52949 \times 10^{-13}$ 
/2.07976  $\times 10^{-11}$  Iter/Total=5/5 Status=0/{Convergence}

T/ $\Delta T$ =2./1.  $\lambda/\Delta\lambda$ =2./1.  $\| \Delta a \| / \| \Psi \|$ = $4.39114 \times 10^{-11}$ 
/5.04681  $\times 10^{-10}$  Iter/Total=5/10 Status=0/{Convergence}

T/ $\Delta T$ =3./1.  $\lambda/\Delta\lambda$ =3./1.  $\| \Delta a \| / \| \Psi \|$ = $6.5379 \times 10^{-11}$ 
/7.42823  $\times 10^{-10}$  Iter/Total=5/15 Status=0/{Convergence}

T/ $\Delta T$ =4./1.  $\lambda/\Delta\lambda$ =4./1.  $\| \Delta a \| / \| \Psi \|$ = $1.49454 \times 10^{-11}$ 
/2.18176  $\times 10^{-10}$  Iter/Total=5/20 Status=0/{Convergence}

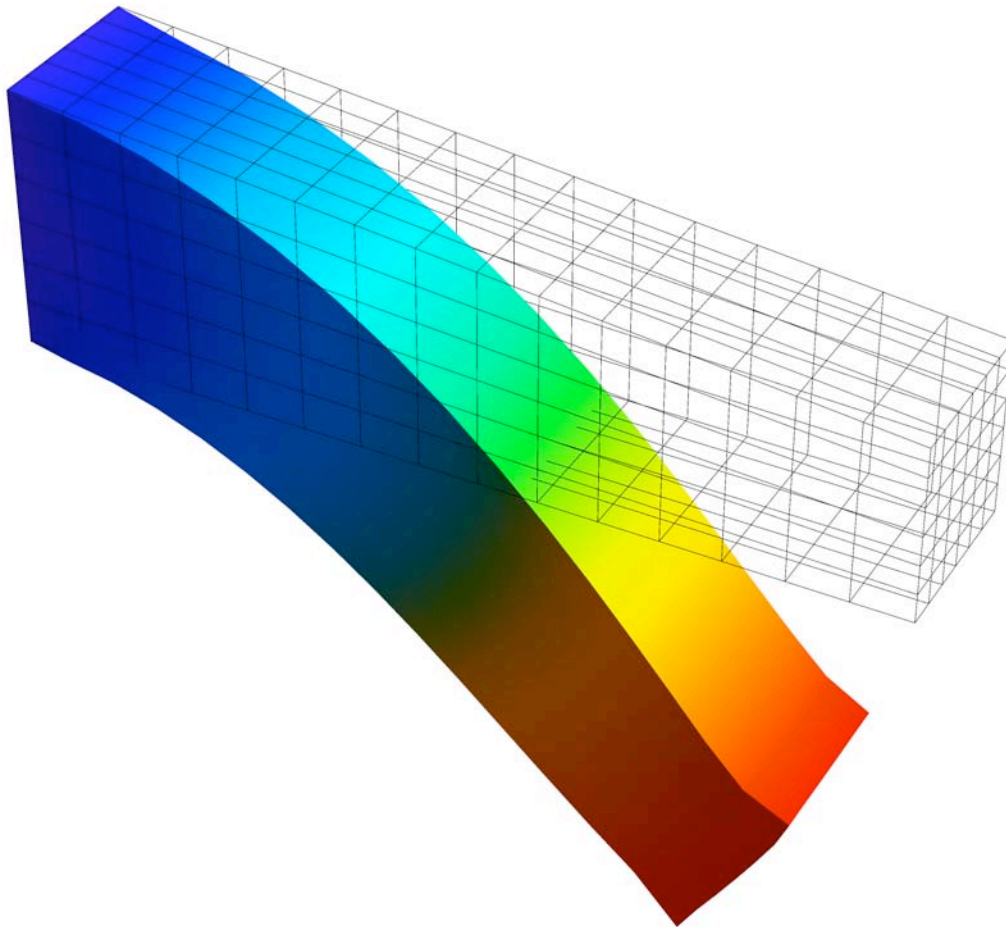
T/ $\Delta T$ =5./1.  $\lambda/\Delta\lambda$ =5./1.  $\| \Delta a \| / \| \Psi \|$ = $4.13176 \times 10^{-12}$ 
/6.68321  $\times 10^{-11}$  Iter/Total=5/25 Status=0/{Convergence}

SMTPostData[{"u", "v", "w", "Sxx", "Exx"}, {10, 1, 1}]

{-1.69498, -7.99336  $\times 10^{-17}$ , -5., 61.145, -0.0500509}

```

```
Show[SMTShowMesh["Elements" -> 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

```

<< "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" -> Hold[{Null} &],
  "SMSSymmetricTangent" -> True,
  "SMSGroupDataNames" -> {"E -elastic modulus",
    "v -poisson ratio", "rho0 -density", "bX -force per unit mass X",
    "bY -force per unit mass Y", "bZ -force per unit mass Z"},
  "SMSDefaultData" -> {21 000, 0.3, 1, 0, 0, 0}
];

```

TestExamples

```

ElementDefinitions[] := (
  E = {xi, eta, zeta} + Table[SMSReal[es$$["IntPoints", i, Ig]], {i, 3}];
  XI = Table[SMSReal[nd$$[i, "X", j]], {i, 8}, {j, 3}];
  En = {{-1, -1, -1}, {1, -1, -1}, {1, 1, -1}, {-1, 1, -1},
    {-1, -1, 1}, {1, -1, 1}, {1, 1, 1}, {-1, 1, 1}};
  NI = Table[1 / 8 (1 + xi En[[i, 1]]) (1 + eta En[[i, 2]]) (1 + zeta En[[i, 3]]), {i, 1, 8}];
  X = SMSFreeze[NI.XI]; Jg = SMSD[X, E]; Jgd = Det[Jg];
  pI = SMSReal[Table[nd$$[i, "at", j], {i, SMSNoNodes}, {j, SMSDOFGlobal[[i]]}]];
  pe = Flatten[pI];
  uI = pI[[1 ;; 8]]; u = NI.uI;
  Dg = SMSD[u, X, "Dependency" -> {E, X, SMSInverse[Jg]}];
  J0 = SMSReplaceAll[Jg, {xi -> 0, eta -> 0, zeta -> 0}]; J0d = Det[J0];
  ae = pI[[9]];

  HbE = (


|            |             |              |
|------------|-------------|--------------|
| xi ae[[1]] | eta ae[[2]] | zeta ae[[3]] |
| xi ae[[4]] | eta ae[[5]] | zeta ae[[6]] |
| xi ae[[7]] | eta ae[[8]] | zeta ae[[9]] |


); Hb =  $\frac{J0d}{Jgd}$  HbE.SMSInverse[J0];

  F = SMSFreeze[IdentityMatrix[3] + Dg + Hb];
  JF = Det[F]; Cg = Transpose[F].F; {Em, v, rho0, bX, bY, bZ} =
    SMSReal[Table[es$$["Data", i], {i, Length[SMSGroupDataNames]}]];
  {lambda, mu} = SMSHookeToLame[Em, v]; bb = {bX, bY, bZ};
  W = 1 / 2 lambda (JF - 1) ^ 2 + mu (1 / 2 (Tr[Cg] - 3) - Log[JF]);
  wgp = SMSReal[es$$["IntPoints", 4, Ig]];
)

```



```

SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]];
ElementDefinitions[];
SMSDo[
  Rg = Jgd wgp SMSD[W - ρ0 u.bb, pe, i];
  SMSExport[SMSResidualSign Rg, p$$[i], "AddIn" → True];
  SMSDo[
    Kg = SMSD[Rg, pe, j];
    SMSExport[Kg, s$$[i, j], "AddIn" → True];
    , {j, i, SMSNoDOFGlobal}}];
    , {i, 1, SMSNoDOFGlobal}}];
  SMSEndDo[]];

SMSStandardModule["Postprocessing"];
SMSDo[Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]];
ElementDefinitions[];
SMSNPostNames = {"DeformedMeshX", "DeformedMeshY", "DeformedMeshZ", "u", "v", "w"};
SMSExport[Table[Join[uI[[i]], uI[[i]], {i, 8}], npost$$];
Eg = 1 / 2 (Cg - IdentityMatrix[3]);
σ = (1 / JF) * SMSD[W, F] . Transpose[F];
SMSGPostNames = {"Sxx", "Sxy", "Sxz", "Syx", "Syy", "Syz", "Szx", "Szy", "Szz",
  "Exx", "Exy", "Exz", "Eyx", "Eyy", "Eyz", "Ezx", "Ezy", "Ezz"};
SMSExport[{σ, Eg} // Flatten, gpost$$[Ig, #] &];
SMSEndDo[]];

SMSWrite[]];

```

File:	ExamplesHypersolid3DB.c	Size:	35 506
Methods	No.Formulae	No.Leafs	
SKR	298	6692	
SPP	232	4045	

■ Simple test example

```

<< AceFEM` ;
SMTInputData[];
SMTAddDomain["A", "ExamplesHypersolid3DB", {"E *" -> 1000., "v *" -> .3}];
SMTAddEssentialBoundary[{"X" == 0 &, 1 -> 0, 2 -> 0, 3 -> 0}, {"X" == 10 &, 3 -> -1}];
SMTMesh["A", "H1", {2, 1, 1}, {{{{0, 0, 0}, {10, 0, 0}}, {{0, 2, 0}, {10, 2, 0}},
  {{{0, 0, 3}, {10, 0, 2}}, {{0, 2, 3}, {10, 2, 2}}}}];
SMTAnalysis[];
SMTNextStep[1, 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

```

{SMTNodeData["NodeIndex"], SMTNodeData["NodeID"],
  SMTNodeData["X"], SMTNodeData["DOF"]} // Transpose // MatrixForm

```

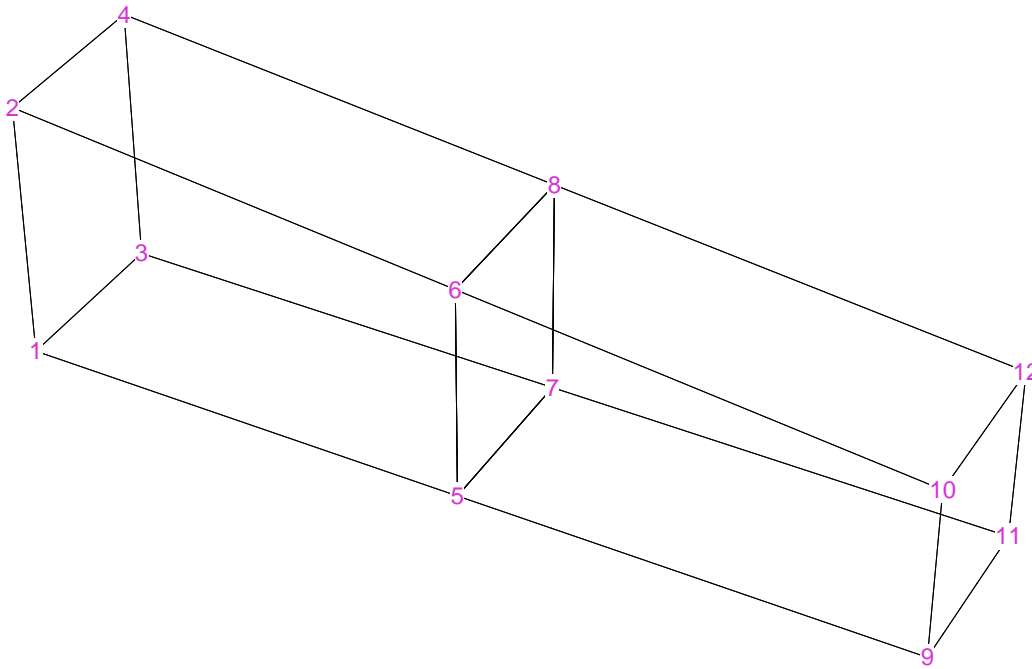
1	D	{0., 0., 0.}	{-1, -1, -1}
2	D	{0., 0., 3.}	{-1, -1, -1}
3	D	{0., 2., 0.}	{-1, -1, -1}
4	D	{0., 2., 3.}	{-1, -1, -1}
5	D	{5., 0., 0.}	{0, 1, 2}
6	D	{5., 0., 2.5}	{6, 7, 8}
7	D	{5., 2., 0.}	{3, 4, 5}
8	D	{5., 2., 2.5}	{9, 10, 11}
9	D	{10., 0., 0.}	{21, 22, -1}
10	D	{10., 0., 2.}	{25, 26, -1}
11	D	{10., 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.

```

SMTShowMesh["Elements" -> False, "NodeMarks" -> True, "Marks" -> "NodeNumber"]

```



Node identification switch -P prevents also selection of the auxiliary nodes by coordinates of the nodes alone.

```

SMTNodeData["X" == 0 &, "NodeIndex"]

```

{1, 2, 3, 4}

Auxiliary nodes can be selected if the node identification is gives as a part of the search criteria.

```

SMTNodeData["ID" == "EAS" &, "NodeIndex"]

```

{13, 14}

Here the index of auxiliary nodes for all elements is extracted.

```
SMTElementData["Nodes"][[All, 9]]
{13, 14}
```

■ Cantilever beam example

```
<< AceFEM` ;
SMTInputData[];
SMTAddDomain["A", "ExamplesHypersolid3DB", {"E *" -> 1000., "v *" -> .3}];
SMTAddEssentialBoundary[{ "X" == 0 &, 1 -> 0, 2 -> 0, 3 -> 0}, {"X" == 10 &, 3 -> -1}];
SMTMesh["A", "H1", {15, 6, 6}, {{{{0, 0, 0}, {10, 0, 0}}, {{0, 2, 0}, {10, 2, 0}}},
  {{{0, 0, 3}, {10, 0, 2}}, {{0, 2, 3}, {10, 2, 2}}}}];
SMTAnalysis[];

SMTNextStep[1, 1];
While[
  While[step = SMTConvergence[10^-8, 10, {"Adaptive", 8, .001, 1, 5}],
    SMTNewtonIteration[]];
  SMTStatusReport[];
  If[step[[4]] === "MinBound", Print["Error:  $\Delta\lambda < \Delta\lambda_{\min}$ "]];
  step[[3]]
  , If[step[[1]], SMTStepBack[]];
  SMTNextStep[1, step[[2]]]
];

T/ $\Delta T$ =1./1.  $\lambda/\Delta\lambda$ =1./1.  $\| \Delta a \| / \| \Psi \|$ = $1.16526 \times 10^{-13}$ 
/1.19557  $\times 10^{-11}$  Iter/Total=5/5 Status=0/{Convergence}

T/ $\Delta T$ =2./1.  $\lambda/\Delta\lambda$ =2./1.  $\| \Delta a \| / \| \Psi \|$ = $2.43509 \times 10^{-11}$ 
/2.80438  $\times 10^{-10}$  Iter/Total=5/10 Status=0/{Convergence}

T/ $\Delta T$ =3./1.  $\lambda/\Delta\lambda$ =3./1.  $\| \Delta a \| / \| \Psi \|$ = $3.6261 \times 10^{-11}$ 
/4.24018  $\times 10^{-10}$  Iter/Total=5/15 Status=0/{Convergence}

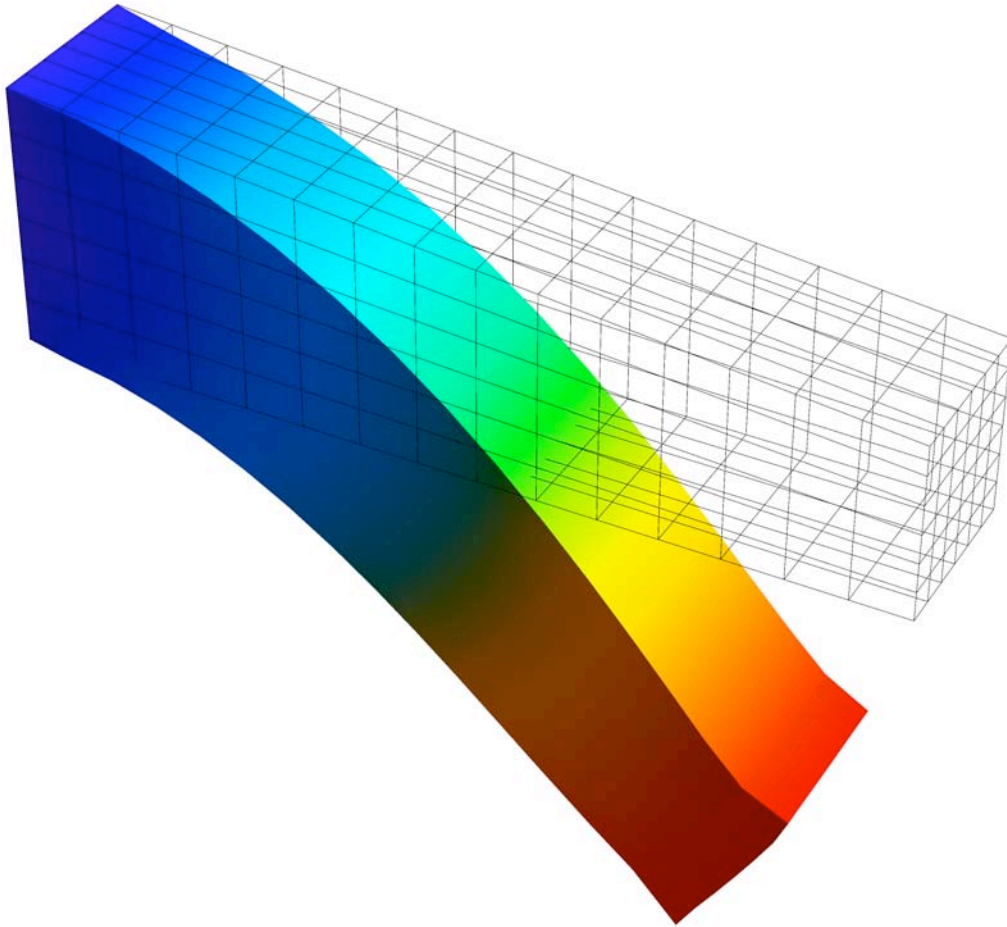
T/ $\Delta T$ =4./1.  $\lambda/\Delta\lambda$ =4./1.  $\| \Delta a \| / \| \Psi \|$ = $8.29827 \times 10^{-12}$ 
/1.2748  $\times 10^{-10}$  Iter/Total=5/20 Status=0/{Convergence}

T/ $\Delta T$ =5./1.  $\lambda/\Delta\lambda$ =5./1.  $\| \Delta a \| / \| \Psi \|$ = $2.29267 \times 10^{-12}$ 
/3.84212  $\times 10^{-11}$  Iter/Total=5/25 Status=0/{Convergence}

SMTPostData[{"u", "v", "w", "Sxx", "Exx"}, {10, 1, 1}]

{-1.69498,  $7.12243 \times 10^{-18}$ , -5., 61.145, -0.0500509}
```

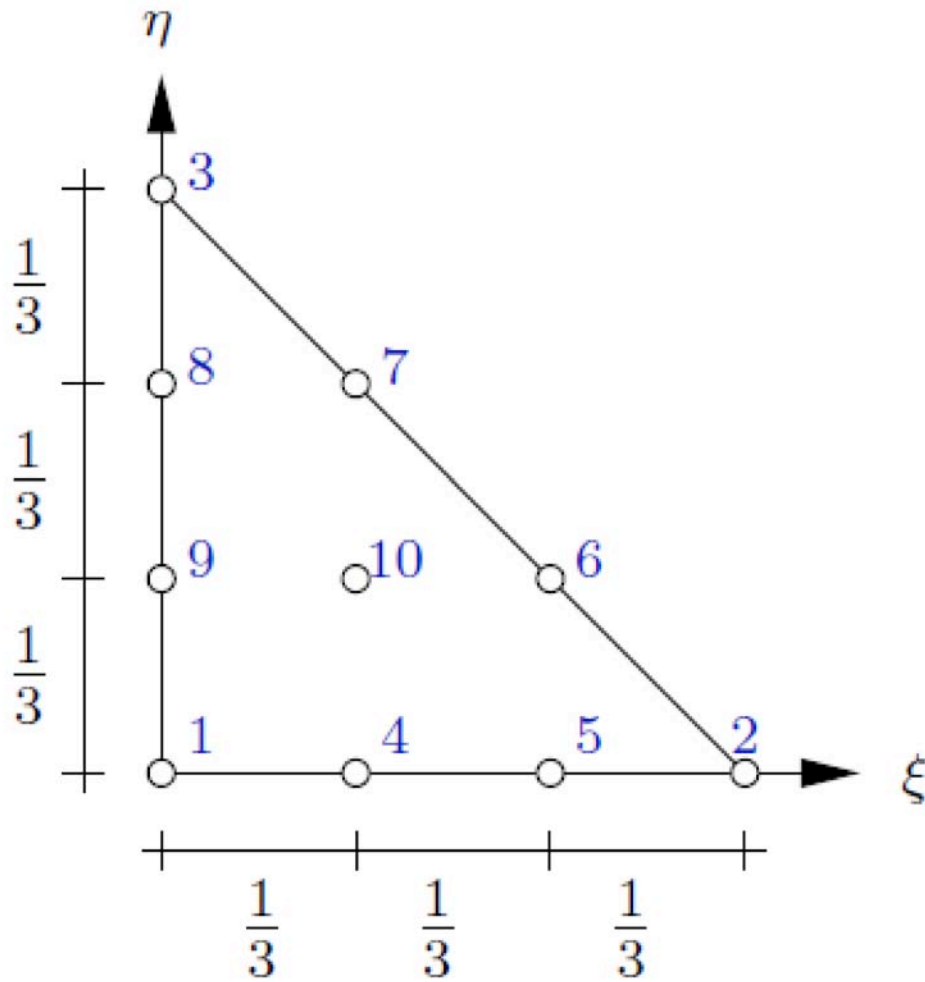
```
Show[SMTShowMesh["Elements" -> False],  
SMTShowMesh["DeformedMesh" -> True,  
"Mesh" -> False, "Field" -> Map[Norm, SMTNodeData["at"]]]]
```



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 linear (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.

```

<< "AceGen`"
SMSInitialize["ExamplesT3", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "T1",
  "SMSNoNodes" → 10,
  "SMSAdditionalNodes" → Hold[{{#1 + 1 / 3 (#2 - #1), #1 + 2 / 3 (#2 - #1),
    #2 + 1 / 3 (#3 - #2), #2 + 2 / 3 (#3 - #2),
    #3 + 1 / 3 (#1 - #3), #3 + 2 / 3 (#1 - #3),
    (#1 + #2 + #3) / 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" → 17,
  "SMSSymmetricTangent" → True,
  "SMSGroupDataNames" ->
    {"E -elastic modulus", "ν -poisson ratio", "t -thickness", "ρ0 -density",
     "bX -force per unit mass X", "bY -force per unit mass Y"},
  "SMSDefaultData" -> {21000, 0.3, 1, 1, 0, 0}
];
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]];
E = {ξ, η, ζ} + Table[SMSReal[es$$["IntPoints", i, Ig]], {i, 3}];
XI + Table[SMSReal[nd$$[i, "X", j]], {i, SMSNoNodes}, {j, 2}];
κ = 1 - ξ - η;
NI = {κ (3 κ - 1) (3 κ - 2) / 2, ξ (3 ξ - 1) (3 ξ - 2) / 2,
  η (3 η - 1) (3 η - 2) / 2, 9 κ ξ (3 κ - 1) / 2, 9 κ ξ (3 ξ - 1) / 2, 9 η ξ (3 ξ - 1) / 2,
  9 η ξ (3 η - 1) / 2, 9 κ η (3 η - 1) / 2, 9 κ η (3 κ - 1) / 2, 27 η ξ κ};
X + SMSFreeze[Append[NI, XI, ζ]];
Jg = SMSD[X, E]; Jgd = Det[Jg];
uI + SMSReal[Table[nd$$[i, "at", j], {i, SMSNoNodes}, {j, 2}]];
pe = Flatten[uI]; u = Append[NI, uI, 0];
Dg = SMSD[u, X, "Dependency" → {E, X, SMSInverse[Jg]}];
F = IdentityMatrix[3] + Dg; JF = Det[F]; Cg = Transpose[F].F;
{Em, ν, tξ, ρ0, bX, bY} +
  SMSReal[Table[es$$["Data", i], {i, Length[SMSGroupDataNames]}]];
{λ, μ} = SMSHookeToLame[Em, ν]; bb = {bX, bY, 0};
W = 1 / 2 λ (JF - 1) ^ 2 + μ (1 / 2 (Tr[Cg] - 3) - Log[JF]);
wgp + SMSReal[es$$["IntPoints", 4, Ig]];
SMSDo[
  Rg = Jgd tξ wgp SMSD[W - ρ0 u.bb, pe, i];
  SMSExport[SMSResidualSign Rg, p$$[i], "AddIn" → True];
  SMSDo[
    Kg = SMSD[Rg, pe, j];
    SMSExport[Kg, s$$[i, j], "AddIn" → True];
    , {j, i, SMSNoDOFGlobal}];
    , {i, 1, SMSNoDOFGlobal}];
  SMSEndDo[]];
SMSWrite[]];

```

File:	ExamplesT3.c	Size:	13 446
Methods	No.Formulae	No.Leafs	
SKR	178	3200	

Cook's membrane test

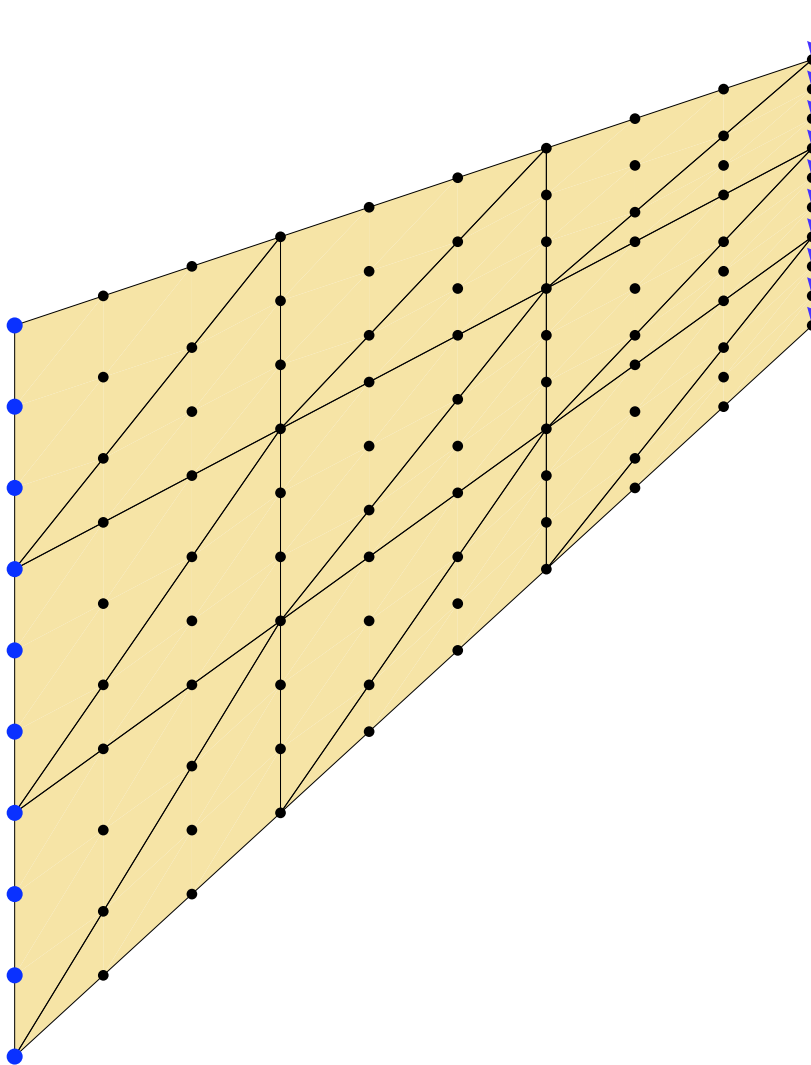
```

<< AceFEM` ;
SMTInputData[];
SMTAddDomain[{"Test", "ExamplesT3", {"E *" -> 1, "ν *" -> 0}}];
SMTMesh["Test", SMSTopology, {3, 3}, {{{0, 0}, {48, 44}}, {{0, 44}, {48, 44 + 16}}]];
SMTAddEssentialBoundary["X" == 0 &, 1 -> 0, 2 -> 0];
SMTAddNaturalBoundary["X" == 48 &, 2 -> -.1];
SMTAnalysis[];

Do[SMTNextStep[1, 0.1];
  While[SMTConvergence[10^-8, 10], SMTNewtonIteration[]];
  , {i, 1, 10}];

SMTShowMesh["NodeMarks" -> True, "BoundaryConditions" -> True]

```

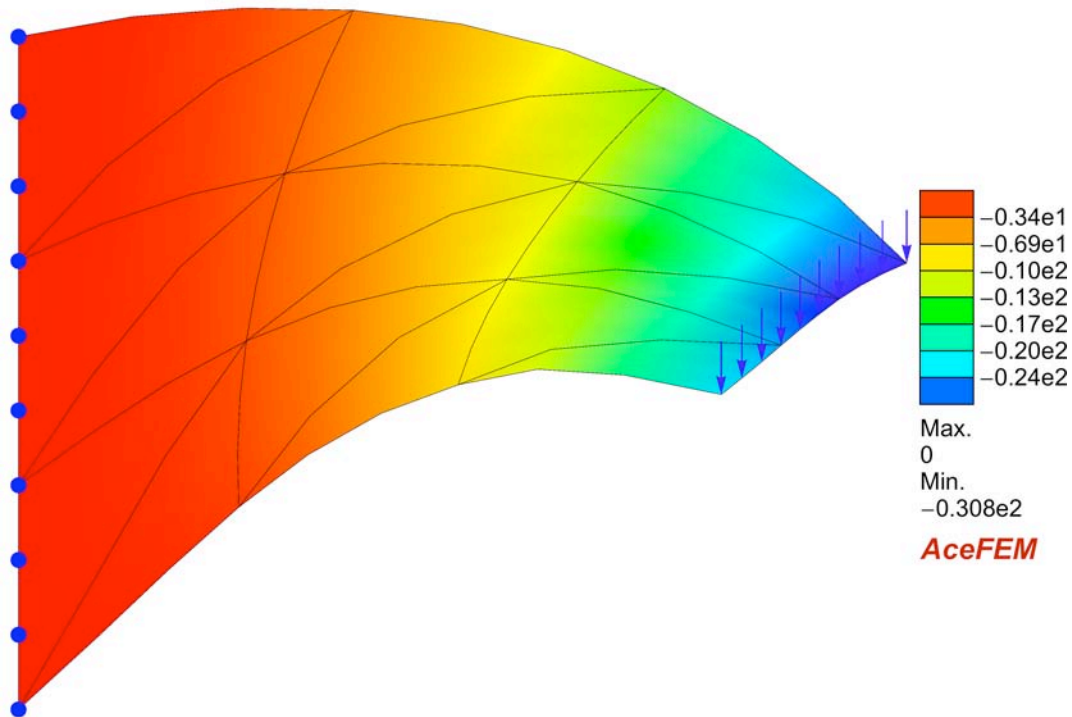


```

SMTNodeData["X" == 48 && "Y" == 44 + 16 &, "at"]
{{10.1092, -30.8017}}

```

```
SMTShowMesh["DeformedMesh" → True, "BoundaryConditions" → True, "Field" → SMTPost[2]]
```



Inflating the Tyre

■ Description

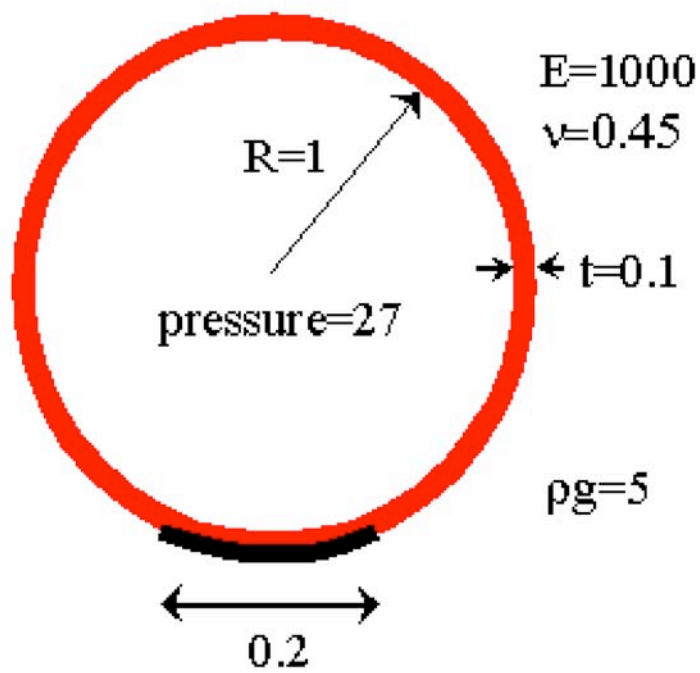
The load such as gas pressure acts perpendicular to the actual deformed surface. Generate 2D surface traction element with the following characteristics:

- ⇒ 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

$$\int_{\Gamma} \mathbf{t} \delta \mathbf{u} \, d\Gamma$$

where $\delta \mathbf{u}$ is variation of the displacement field, \mathbf{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.

```

<< AceGen` ;
SMSInitialize["ExamplesGasPressure", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "L1"
, "SMSSymmetricTangent" → False
, "SMSDefaultIntegrationCode" → 0
, "SMSGroupDataNames" → {"p -pressure", "t -thickness"}
, "SMSDefaultData" → {0, 1}];
SMSStandardModule["Tangent and residual"];
XI = Table[SMSReal[nd$$[i, "X", j]], {i, SMSNoNodes}, {j, SMSNoDimensions}];
uI = SMSReal[Table[nd$$[i, "at", j], {i, SMSNoNodes}, {j, SMSNoDimensions}]];
pe = Flatten[uI];
{p, tξ} = SMSReal[Table[es$$["Data", i], {i, Length[SMSGroupDataNames]}]];
{xi, xj} = XI + uI;
L = SMSSqrt[(xj - xi) . (xj - xi)];
{cosφ, sinφ} = (xj - xi) / L;
ξ = SMSFictive[];
u =  $\frac{1}{2}$  { (1 - ξ) , (1 + ξ) }.uI;
p = {{cosφ, -sinφ}, {sinφ, cosφ}} . {0, p};
Rel = tξ Integrate[- $\frac{L}{2}$  p.D[u, {pe}], {ξ, -1, 1}];
Ke = SMSD[Rel, pe];
SMSExport[Rel, p$$, "AddIn" → True];
SMSExport[Ke, s$$, "AddIn" → True];
SMSWrite[];

```

File:	ExamplesGasPressure.c	Size:	4126
Methods	No.Formulae	No.Leafs	
SKR	10	182	

■ Analysis

```

<< AceFEM` ;
SMTInputData[];
SMTAddDomain[{"tyre", "ExamplesHypersolid2D", {"E *" → 1000., "ν *" → .3}},
{"gas", "ExamplesGasPressure", {}}];
ne = 40;
SMTAddEssentialBoundary["Y" < -1.02 &, 1 → 0, 2 → 0];
SMTMesh["tyre", "Q1", {ne, 4}, Table[
{1.1 {Cos[φ], Sin[φ]}, {Cos[φ], Sin[φ]}}, {φ, 0, 2. π, 2. π / 20}] // Transpose];
SMTMesh["gas", "L1", {ne}, Table[{Cos[φ], Sin[φ]}, {φ, 0, 2. π, 2. π / 20}] //
Reverse];
SMTAnalysis[];
undef = SMTShowMesh["BoundaryConditions" → True, "Marks" → False, "Show" → False,
"Mesh" → False, "Elements" → RGBColor[1, 0, 0]];
SMTNextStep[1, 1];

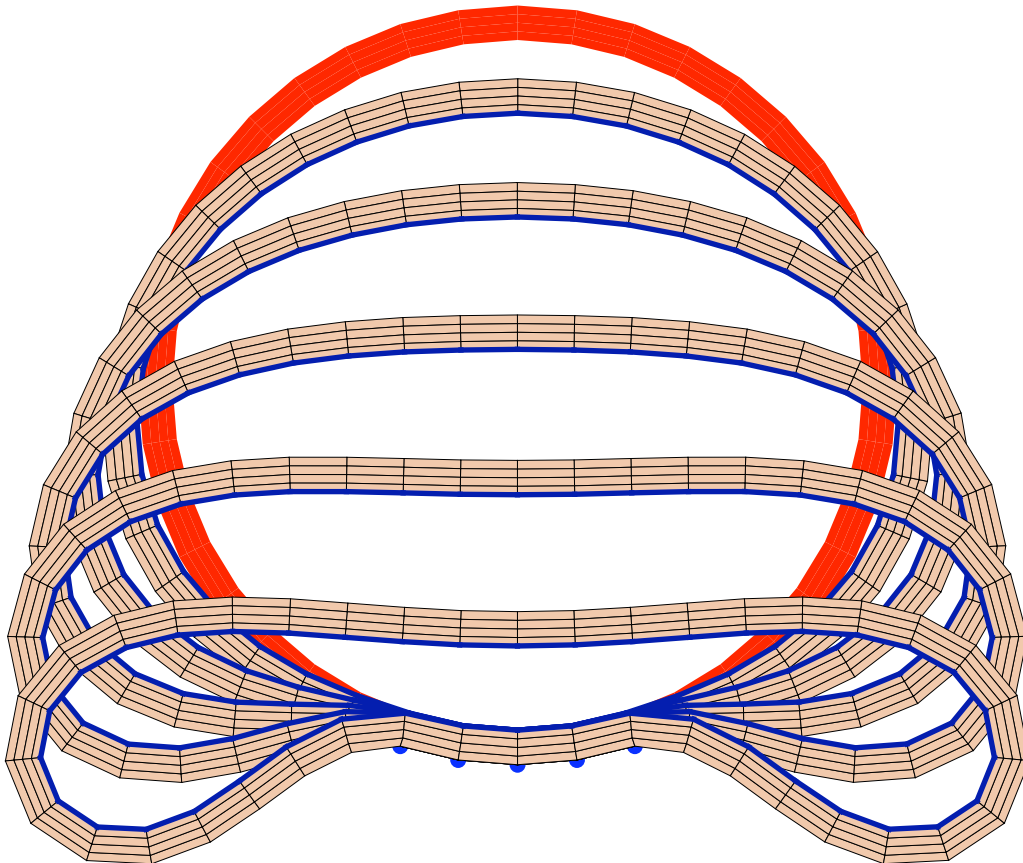
```

The first task is to find out the shape of the empty tyre. We only know that the shape of the weightless tyre is circular. The shape of the empty tyre can be obtained by starting with the weightless tyre and then increasing the volume force to its final value.

```

graph = Table[
  SMTNextStep[0, 0];
  SMTDomainData["tyre", "Data", "bY *" -> -force];
  While[SMTConvergence[10^-9, 10], SMTNewtonIteration[]];
  SMTShowMesh["DeformedMesh" -> True,
    "Show" -> "Window" | False, PlotRange -> {{-1.5, 1.5}, {-1.4, 1.2}}]
  , {force, 1, 5, 1}];
Show[undef, graph]

```



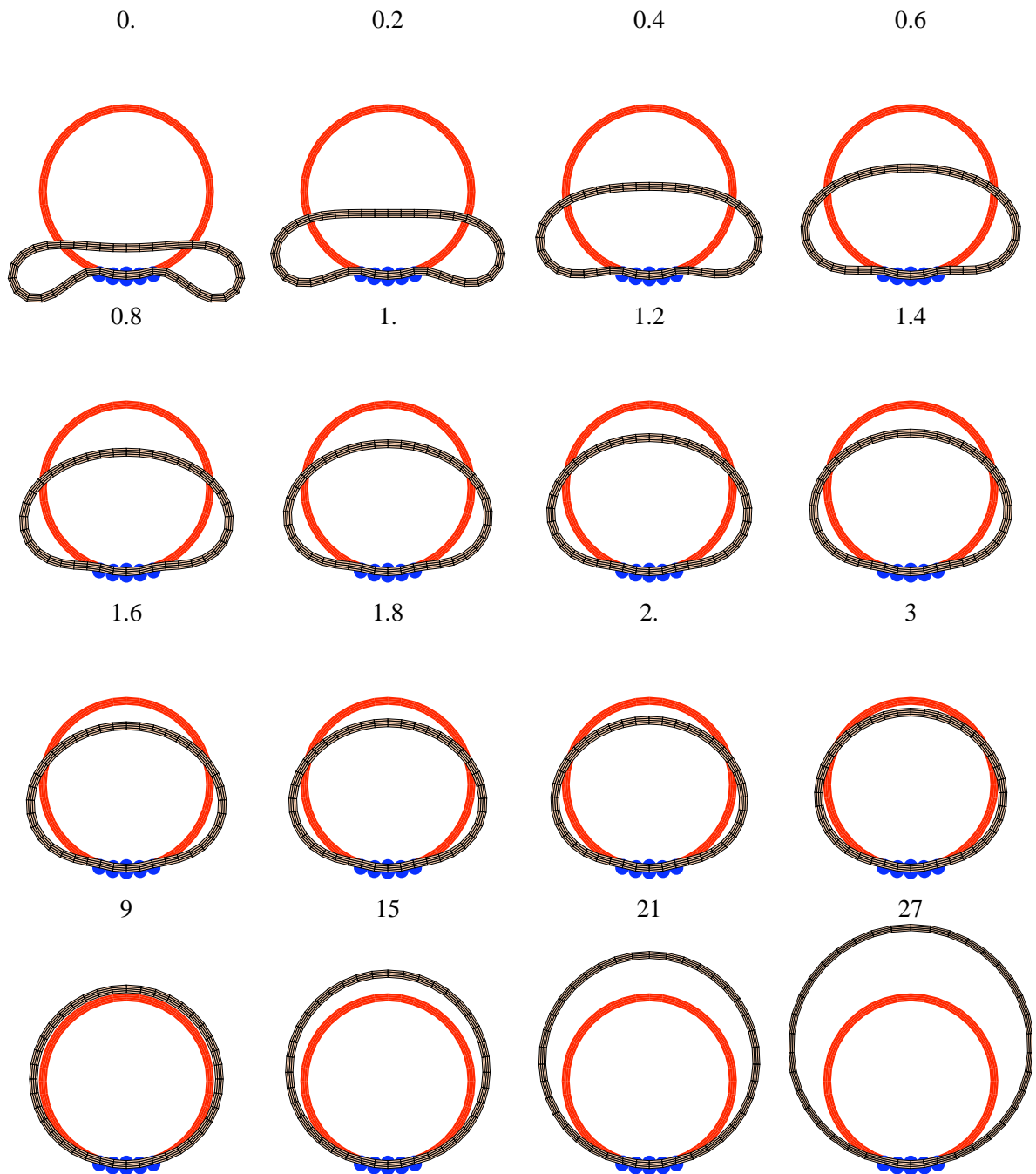
The gas pressure is then increased in order to obtain the shape of the tyre for various gas pressure levels.

```

graph = {};
Map[ (SMTNextStep[0, 0];
  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]]];

```

```
GraphicsGrid[Partition[MapThread[Show[undef, #,
  DisplayFunction -> Identity, PlotLabel -> #2,
  PlotRange -> {{-1.5, 1.5}, {-1.4, 2}}] &, {graph, label}], 4]
, Spacings -> 0, ImageSize -> 450]
```



Advanced Examples

Round-off Error Test

With the AceFEM-MDriver (see AceFEM Structure) 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.

```
<< AceGen` ;
SMSInitialize["PrecisionHeatConduction", "Environment" -> "AceFEM-MDriver"];
SMSTemplate["SMSTopology" -> "H1",
  "SMSDOFGlobal" -> 1, "SMSSymmetricTangent" -> False,
  "SMSGroupDataNames" ->
  {"k0 -conductivity parameter", "k1 -conductivity parameter",
   "k2 -conductivity parameter", "Q -heat source"},
  "SMSDefaultData" -> {1, 0, 0, 0}];
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]];
E = {ξ, η, ζ} + Table[SMSReal[es$$["IntPoints", i, Ig]], {i, 3}];
XI + Table[SMSReal[nd$$[i, "X", j]], {i, SMSNoNodes}, {j, SMSNoDimensions}];
En = {{-1, -1, -1}, {1, -1, -1}, {1, 1, -1}, {-1, 1, -1},
  {-1, -1, 1}, {1, -1, 1}, {1, 1, 1}, {-1, 1, 1}};
NI = Table[1/8 (1 + ξ En[[i, 1]]) (1 + η En[[i, 2]]) (1 + ζ En[[i, 3]]), {i, 1, 8}];
X + SMSFreeze[NI.XI]; Jg = SMSD[X, E]; Jgd = Det[Jg];
φI + SMSReal[Table[nd$$[i, "at", 1], {i, SMSNoNodes}]];
φ = NI.φI;
{k0, k1, k2, Q} + SMSReal[Table[es$$["Data", i], {i, Length[SMSGroupDataNames]}]];
k + k0 + k1 φ + k2 φ2;
λ + SMSReal[rdata$$["Multiplier"]];
wgp + SMSReal[es$$["IntPoints", 4, Ig]];
Dφ = SMSD[φ, X, "Dependency" -> {E, X, SMSInverse[Jg]}];
W =  $\frac{1}{2}$  k Dφ.Dφ - φ λ Q;
SMSDo[
  Rg = Jgd wgp SMSD[W, φI, i, "Constant" -> k];
  SMSExport[SMSResidualSign Rg, p$$[i], "AddIn" -> True];
  SMSDo[
    Kg = SMSD[Rg, φI, j];
    SMSExport[Kg, s$$[i, j], "AddIn" -> True];
    , {j, 1, 8}
  ];
  , {i, 1, 8}
];
SMSEndDo[];
SMSWrite[];
```

File:	PrecisionHeatConduction.m	Size:	11387
Methods	No.Formulae	No.Leafs	
SMT`SKR	178	2645	

The domain of the problem is sphere with radius 1. On the lower surface of the sphere the constant temperature of $\phi=0$ 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.

```
<< 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 *SMTMesh* function uses general interpolation of coordinates with the arbitrary number of interpolation points. Here the mesh of $11 \times 11 \times 11$ interpolation points is obtained by mapping the cube into the sphere. The volume is then divided into $5 \times 5 \times 5$ elements.

```
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];
SMTMesh["sphere", "H1", {5, 5, 5}, points];

SMTAnalysis[];
SMTNextStep[1, 1];

$MinPrecision = 0; $MaxPrecision = Infinity;
```

This gives the temperature in the node 100 and the numerical precision of the result for 5 Newton-Raphson iterations.

```
res = Table[
  SMTNewtonIteration[];
  {i, SMTNodeData[100, "at"][[1]], SMTNodeData[100, "at"][[1]] // Precision}
  , {i, 5}];

TableForm[res, TableHeadings -> {None, {"Iteration", "Temperature", "Precision"}}]
```

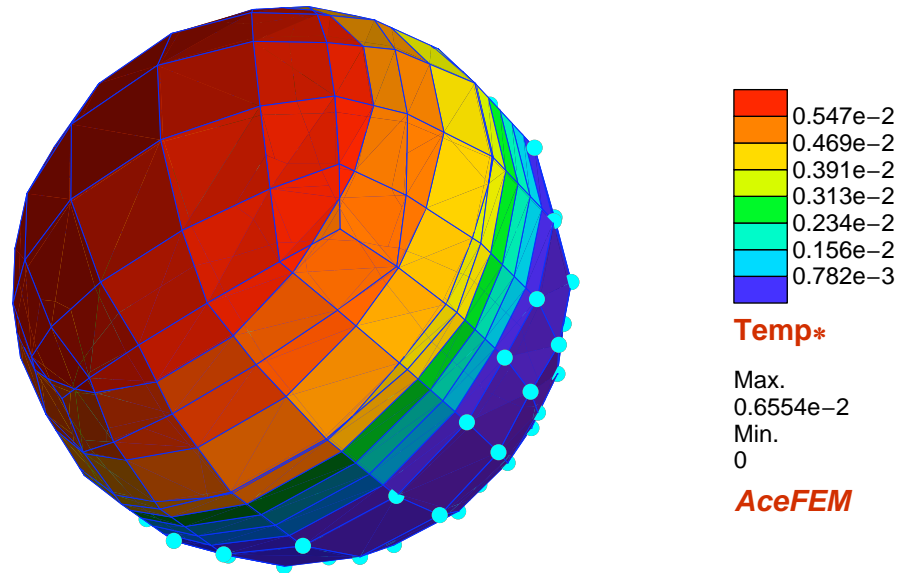
Iteration	Temperature	Precision
1	0.0043136777915792517465675360868405860	35.6102
2	0.00431268955724400112376415971040235	33.0774
3	0.00431268955719375064016726409861	30.4846
4	0.004312689557193750640167263969	28.2954
5	0.004312689557193750640167263969	28.2954

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.

```
<< 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];
SMTMesh["sphere", "H1", {5, 5, 5},
  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}]];
SMTAnalysis[]; SMTNextStep[1, 1];
Table[SMTNewtonIteration[], {5}]

{0.00468568, 1.27035 × 10-6, 1.0465 × 10-13, 3.01546 × 10-19, 2.44699 × 10-19}
```

```
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.

```
SMTNodeData[100, "at"] // InputForm  
  
{0.004312689557193752}
```

Solid, Finite Strain Element for Direct and Sensitivity Analysis

■ Description

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 nodes,
- ⇒ the element should allow arbitrary large displacements and rotations,

⇒ the problem is defined by the hyperelastic Neo-Hooke type strain energy potential

$$W = \frac{\lambda}{2} (\det \mathbf{F} - 1)^2 + \mu \left(\frac{\text{Tr}[\mathbf{C}] - 3}{2} - \text{Log}[\det \mathbf{F}] \right)$$

and total potential energy of the problem

$$\Pi = \int_{\Omega_0} (W - \rho_0 \mathbf{u} \cdot \bar{\mathbf{b}}) d\Omega_0$$

where $\mathbf{C} = \mathbf{F}^T \mathbf{F}$ is right Cauchy-Green tensor, $\mathbf{F} = \mathbf{I} + \nabla \mathbf{u}$ is deformation gradient,

\mathbf{u} is displacements field, \mathbf{Q} is prescribed body force, $\bar{\mathbf{b}}$ is force per unit mass nad ρ_0 density in initial configuration,

Ω_0 is the initial domain of the problem and λ, μ are the first and the second Lamé's material constants.

The following user subroutines have to be generated:

⇒ user subroutine for the direct implicit analysis,

⇒ user subroutine for the sensitivity analysis with respect to the material constants, arbitrary shape parameter and prescribed boundary condition.

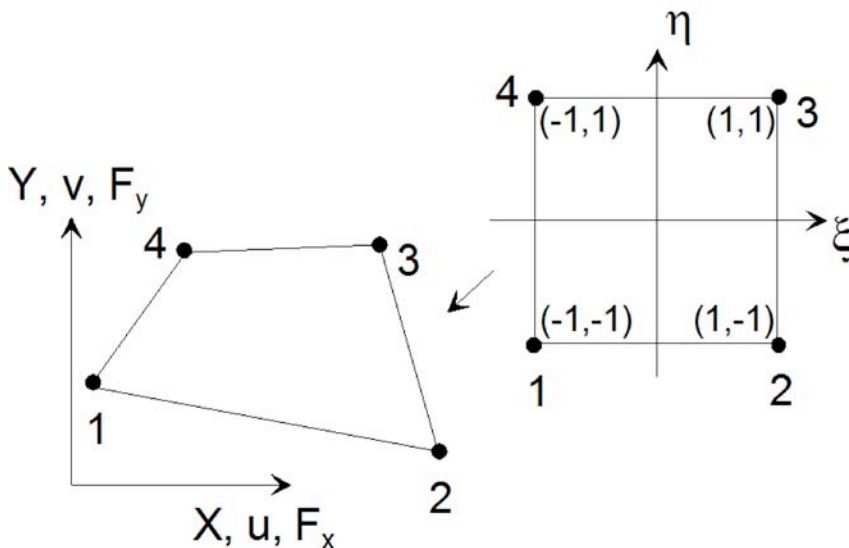
⇒ user subroutine for the post-processing that returns the Green-Lagrange strain tensor and the Cauchy stress tensor.

⇒ user subroutine Task that performs two tasks:

1 task "Volume" that returns the volume and the sensitivity of the volume with respect to all sensitivity parameters

2 task "Misses" that returns the Misses stress in all integration points

3 task "MissesSensitivity" that returns the sensitivity of the Misses stress with respect to given sensitivity parameter in all integration points



■ Solution

```
<< "AceGen`";
SMSInitialize["ExamplesSensitivity2D", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "Q1", "SMSSymmetricTangent" → True,
  "SMSGroupDataNames" -> {"E -elastic modulus", "ν -poisson ratio", "t -thickness",
    "ρ0 -density", "bX -force per unit mass X", "bY -force per unit mass Y"},
  "SMSSensitivityNames" → {"E -elastic modulus", "ν -poisson ratio",
    "t -thickness", "ρ0 -density", "bX -force per unit mass X",
    "bY -force per unit mass Y", "BC"},
  "SMSShapeSensitivity" -> True,
  "SMSDefaultData" -> {21000, 0.3, 1, 1, 0, 0},
  "SMSCharSwitch" → {"Volume", "Misses", "MissesSensitivity"}
];
```

Definitions of geometry, kinematics, strain energy ...

```

ElementDefinitions[sensitivity_] := (
  E = {ξ, η, ζ} ⊢ Table[SMSReal[es$$["IntPoints", i, Ig]], {i, 3}];
  If[sensitivity
    , (*shape sensitivity*)
    DXIDφ = SMSIf[SensType == 2
      , SMSReal[Table[
        nd$$[i, "sX", SensTypeIndex, j], {i, SMSNoNodes}, {j, SMSNoDimensions}]]
      , Table[0, {i, SMSNoNodes}, {j, SMSNoDimensions}]]
    ];
  XI ⊢ Table[SMSReal[nd$$[i, "X", j], "Dependency" → {φ, DXIDφ[[i, j]]}],
    {i, SMSNoNodes}, {j, SMSNoDimensions}];

  (*DOF sensitivity - essential boundary sensitivity included*)
  DuIDφ = SMSReal[
    Table[nd$$[i, "st", SensIndex, j], {i, SMSNoNodes}, {j, SMSNoDimensions}]];
  uI ⊢ Table[SMSReal[nd$$[i, "at", j], "Dependency" → {φ, DuIDφ[[i, j]]}],
    {i, SMSNoNodes}, {j, SMSNoDimensions}];

  (*parameter sensitivity*)
  {Em, ν, tξ, ρ0, bX, bY} = Table[
    SMSReal[es$$["Data", i], "Dependency" →
      {φ, SMSKroneckerDelta[1, SensType] SMSKroneckerDelta[i, SensTypeIndex]}]
    , {i, Length[SMSGGroupDataNames]};

  , XI ⊢ Table[SMSReal[nd$$[i, "X", j]], {i, SMSNoNodes}, {j, SMSNoDimensions}];
  uI ⊢ SMSReal[Table[nd$$[i, "at", j], {i, SMSNoNodes}, {j, SMSNoDimensions}]];
  {Em, ν, tξ, ρ0, bX, bY} ⊢
    SMSReal[Table[es$$["Data", i], {i, Length[SMSGGroupDataNames]}]];
  ];
  bb = {bX, bY, 0};
  NI = 1 / 4 { (1 - ξ) (1 - η), (1 + ξ) (1 - η), (1 + ξ) (1 + η), (1 - ξ) (1 + η) };
  X ⊢ SMSFreeze[Append[NI.XI, ζ]];
  Jg = SMSD[X, E]; Jgd = Det[Jg];
  pe = Flatten[uI]; u = Append[NI.uI, 0];
  Dg = SMSD[u, X, "Dependency" → {E, X, SMSInverse[Jg]}];
  SMSFreeze[F, IdentityMatrix[3] + Dg, "KeepStructure" → True];
  JF = Det[F]; Cg = Transpose[F].F;
  {λ, μ} = SMSHookeToLame[Em, ν];
  W = 1 / 2 λ (JF - 1) ^ 2 + μ (1 / 2 (Tr[Cg] - 3) - Log[JF]);
  wgp ⊢ SMSReal[es$$["IntPoints", 4, Ig]];
)

```

"Tangent and residual" user subroutine

```

SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]];
ElementDefinitions[False];
SMSDo[
  Rg = Jgd tξ wgp SMSD[W - ρ0 u.bb, pe, i];
  SMSExport[SMSResidualSign Rg, p$$[i], "AddIn" → True];
  SMSDo[
    Kg = SMSD[Rg, pe, j];
    SMSExport[Kg, s$$[i, j], "AddIn" → True];
    , {j, i, 8}];
    , {i, 1, 8}];
  SMSEndDo[];

```

"Postprocessing" user subroutine

```

SMSStandardModule["Postprocessing"];
SMSDo[Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]];
ElementDefinitions[False];
SMSNPostNames = {"DeformedMeshX", "DeformedMeshY", "u", "v"};
SMSExport[Table[Join[uI[[i]], uI[[i]], {i, SMSNoNodes}], npost$$];
Eg = 1 / 2 (Cg - IdentityMatrix[3]);
σ = (1 / JF) * SMSD[W, F, "IgnoreNumbers" -> True].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[];

```

"Sensitivity pseudo-load" user subroutine

```

SMSStandardModule["Sensitivity pseudo-load"];
ϕ = SMSFictive[];
SensIndex = SMSInteger[idata$$["SensIndex"]];
SensType = SMSInteger[es$$["SensType", SensIndex]];
SensTypeIndex = SMSInteger[es$$["SensTypeIndex", SensIndex]];
SMSDo[
  ElementDefinitions[True];
  SMSDo[
    Rg = Jgd tξ wgp SMSD[W - ρ0 u.bb, pe, i];
    SMSExport[SMSResidualSign SMSD[Rg, ϕ], p$$[i], "AddIn" → True];
    , {i, 1, 8}];
    , {Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]]];

```

"Tasks" user subroutine

```

SMSStandardModule["Tasks"];

SMSIf[SMSInteger[Task$$] == -1
,
  SMSExport[{1, 0, 0, 0, 1 + SMSInteger[idata$$["NoSensParameters"]]}, TasksData$$];
];

```

```

SMSIf [SMSInteger [Task$$] == 1
, SMSDo [
  ElementDefinitions [False];
  SMSEExport [Jgd tζ wgp, RealOutput$$[1], "AddIn" → True];
  SMSDo [
    φ ← SMSFictive [];
    SensType ← SMSInteger [es$$["SensType", SensIndex]];
    SensTypeIndex ← SMSInteger [es$$["SensTypeIndex", SensIndex]];
    ElementDefinitions [True];
    SMSEExport [SMSD [Jgd tζ wgp, φ], RealOutput$$[1 + SensIndex], "AddIn" → True];
    , {SensIndex, 1, SMSInteger [idata$$["NoSensParameters"]]}
  ];
  , {Ig, 1, SMSInteger [es$$["id", "NoIntPoints"]]}
];

SMSIf [SMSInteger [Task$$] == -2
, SMSEExport [{3, 0, 0, 0, SMSInteger [es$$["id", "NoIntPoints"]]}, TasksData$$];
];

SMSIf [SMSInteger [Task$$] == 2
, SMSDo [
  ElementDefinitions [False];
  σ = (1 / JF) * SMSD [W, F, "IgnoreNumbers" → True] . Transpose [F];
  s = σ -  $\frac{1}{3}$  IdentityMatrix [3] Tr [σ];
  Misses =  $\sqrt{(3 / 2) \text{Total} [s s, 2]}$  ;
  SMSEExport [Misses, RealOutput$$[Ig]];
  , {Ig, 1, SMSInteger [es$$["id", "NoIntPoints"]]}
];

SMSIf [SMSInteger [Task$$] == -3
, SMSEExport [{3, 1, 0, 0, SMSInteger [es$$["id", "NoIntPoints"]]}, TasksData$$];
];

```

Please, consider using `SMSSqrt` instead of `Sqrt`.

See also: [Expression Optimization](#)

```

SMSIf [SMSInteger[Task$$] == 3
, SMSDo [
  ϕ ← SMSFictive[];
  SensIndex ← SMSInteger[IntegerInput$$[1]];
  SensType ← SMSInteger[es$$["SensType", SensIndex]];
  SensTypeIndex ← SMSInteger[es$$["SensTypeIndex", SensIndex]];
  ElementDefinitions[True];
  σ ← (1 / JF) * SMSD[W, F, "IgnoreNumbers" -> True] . Transpose[F];
  s = σ -  $\frac{1}{3}$  IdentityMatrix[3] Tr[σ];

  Misses =  $\sqrt{(3 / 2) \text{Total}[s s, 2]}$  ;
  SMSExport[SMSD[Misses, ϕ], RealOutput$$[Ig]];
  , {Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]}
];
]

```

Please, consider using `SMSSqrt` instead of `Sqrt`.

See also: [Expression Optimization](#)

Code generation

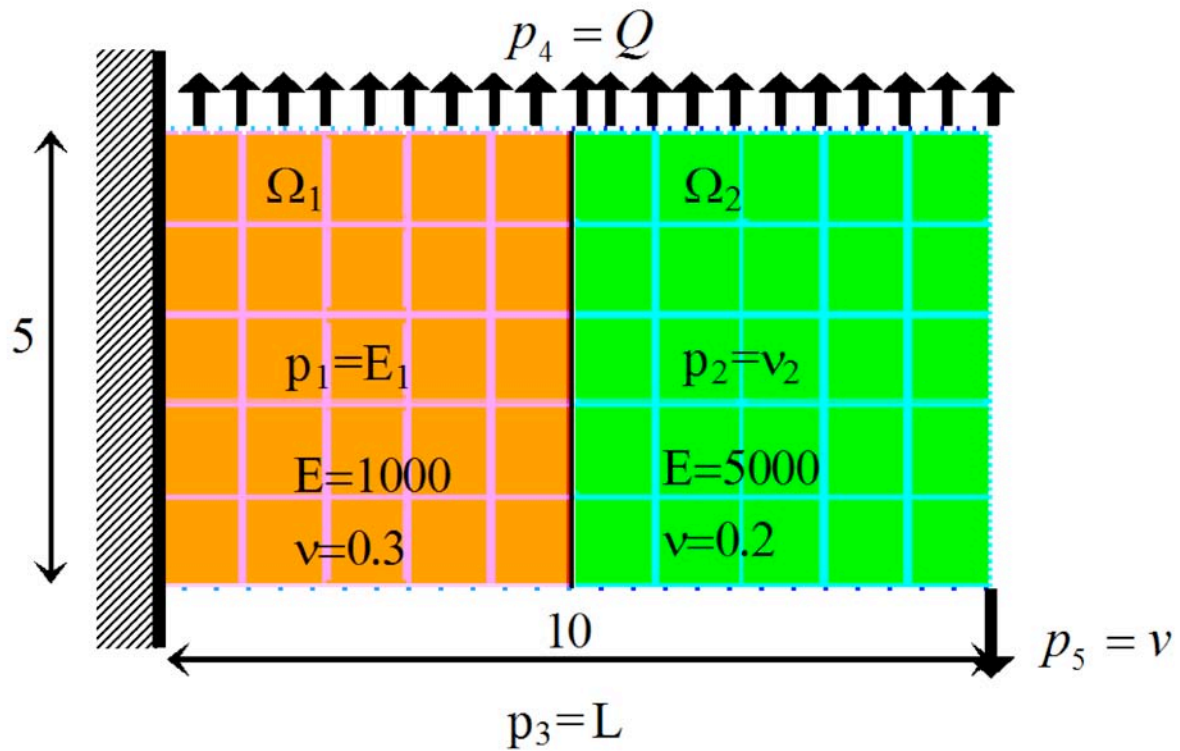
```
SMSWrite[];
```

File:	ExamplesSensitivity2D.c	Size:	31 839
Methods	No.Formulae	No.Leafs	
SKR	92	1439	
SPP	68	967	
SSE	162	2638	
Tasks	297	4224	

Parameter, Shape and Load Sensitivity Analysis of Multi-Domain Example

■ Description

With the use of the element generated in previous example analyze the following two-domain example.



The sensitivity of the displacement field $\mathbf{u} = \{u, v\}$ with respect to the following parameters has to be analyzed:

- $p_1 \Rightarrow$ elastic modulus of the first domain E_1 ,
- $p_2 \Rightarrow$ Poisson ration of the second domain ν_2 ,
- $p_3 \Rightarrow$ length L ,
- $p_4 \Rightarrow$ distributed force Q ,
- $p_5 \Rightarrow$ prescribed displacement v .

■ Solution

```

<< AceFEM` ;
SMTInputData[];
L = 10; Q = 500; v = 1;
SMTAddDomain[{"Ω1", "ExamplesSensitivity2D", {"E *" -> 1000, "v *" -> 0.3}},
  {"Ω2", "ExamplesSensitivity2D", {"E *" -> 5000, "v *" -> 0.2}}];
SMTAddEssentialBoundary["X" == 0 &, 1 -> 0, 2 -> 0];
SMTAddNaturalBoundary["Y" == L / 2 &, 2 -> Q / 40];
SMTAddEssentialBoundary["X" == L && "Y" == 0 &, 2 -> -v];
SMTMesh["Ω1", "Q1", {20, 20}, {{{0, 0}, {L / 2, 0}}, {{0, L / 2}, {L / 2, L / 2}}];
SMTMesh["Ω2", "Q1", {20, 20}, {{{L / 2, 0}, {L, 0}}, {{L / 2, L / 2}, {L, L / 2}}];
SMTAddSensitivity[{
  (* E is the first material parameter on "Ω1" domain*)
  {"E", 1000, "Ω1" -> {1, 1}},
  (* v is the second material parameter on "Ω2" domain*)
  {"v", 0.2, "Ω2" -> {1, 2}},
  (* L is first shape parameter for all domains*)
  {"L", L, _ -> {2, 1}},
  (* Q is the first boundary condition parameter - prescribed force*)
  {"Q", 1, _ -> {5, 1}},
  (* v is the second boundary condition parameter - prescribed displacement *)
  {"v", 1, _ -> {4, 2}}
}];

SMTAnalysis[];

```

This sets an initial sensitivity of node coordinates (shape velocity field) with respect to L for shape sensitivity analysis.

```
SMTNodeData["sX", Map[{{#[[2]] / L, 0} &, SMTNodes]]];
```

This sets an initial sensitivity of prescribed force (BC velocity field) with respect to the intensity of the distributed force Q .

```
SMTNodeData["Y" == L / 2 &, "sdB", {0, 1 / 40., 0, 0}];
```

This sets an initial sensitivity of prescribed displacement (BC velocity field) with respect to prescribed displacement v .

```
SMTNodeData["X" == L && "Y" == 0 &, "sdB", {0, 0, 0, -1}];
```

Here is the primal analysis executed.

```

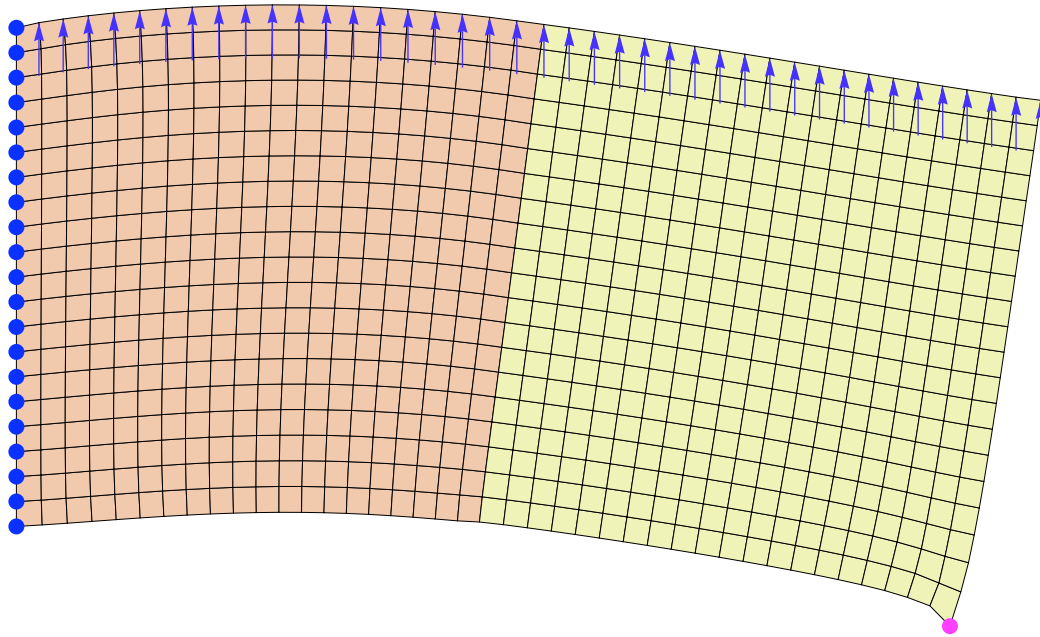
SMTNextStep[1, 1];
While[SMTConvergence[10^-9, 10], SMTNewtonIteration[]];
SMTStatusReport[];

T/ΔT=1./1. λ/Δλ=1./1. ||Δa||/||Ψ||=2.84978 × 10-13
/7.89774 × 10-11 Iter/Total=6/6 Status=0/{Convergence}

```



```
SMTShowMesh["BoundaryConditions" → True, "DeformedMesh" → True]
```

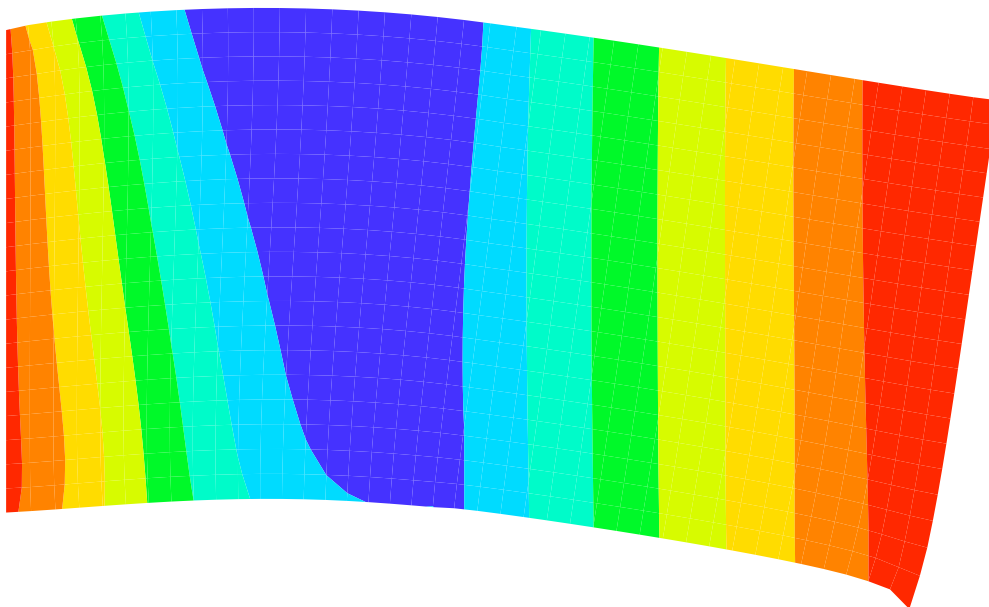


This performs the sensitivity analysis with respect to all parameters.

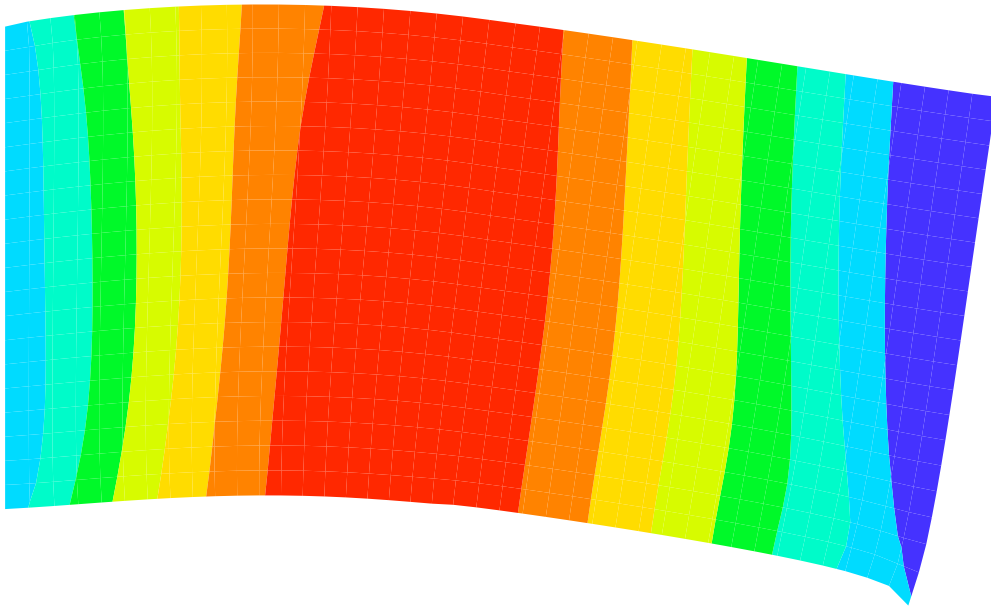
```
SMTSensitivity[];
```

```
GraphicsGrid[Partition[
  Join[Array[
    (SMTIData["SensIndex", #];
    SMTShowMesh["DeformedMesh" → True,
      "Mesh" → False, "Field" → SMTPost[2, "st"],
      "Legend" → False, "Contour" → 5, "Label" →
        {"∂v/∂", {"E", "ν", "L", "Q", "v"}[[#]], " ", Automatic}]] &, 5], {""}, 2]]
```

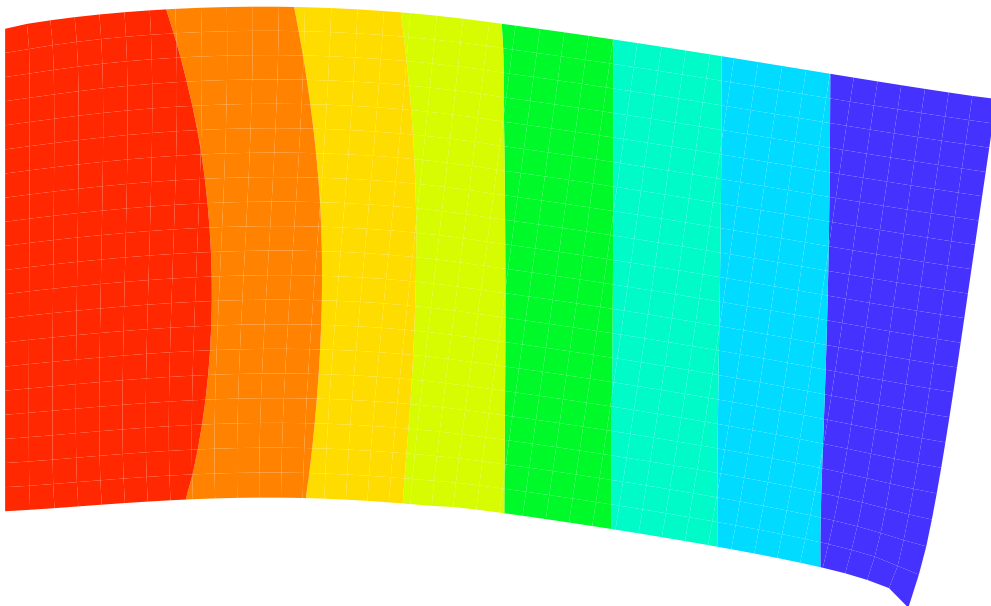
$\partial v / \partial E$ Min= $-0.3136e-3$ Max= $0.65908e-4$



$\partial v / \partial L$ Min= $-0.2577e-1$ Max= $0.74954e-1$



$\partial v / \partial v$ Min= $-0.1045e1$ Max= 0



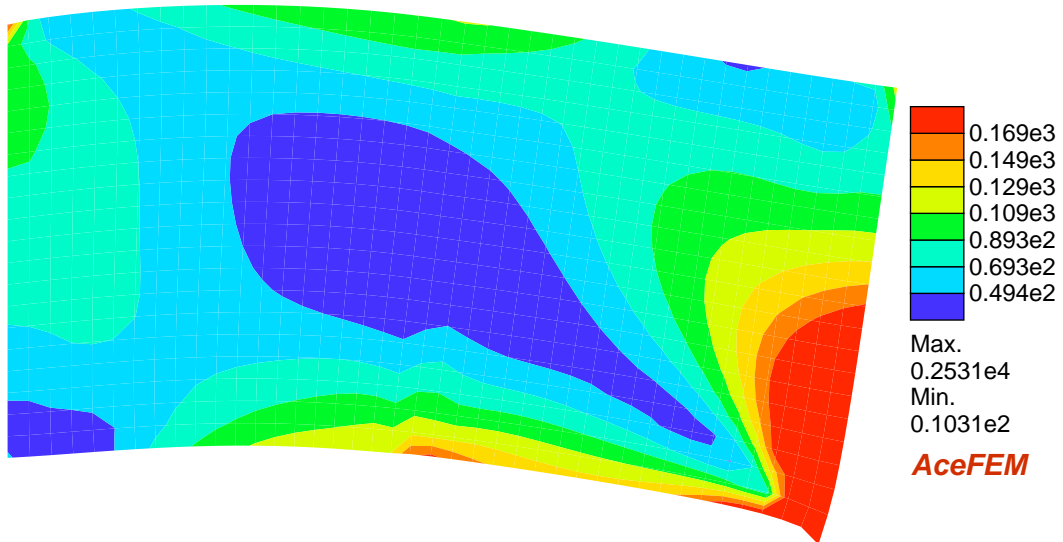
This evaluates the volume of the mesh and sensitivity of the volume with respect to all parameters. As expected, only the third parameter (length L) effects the volume.

```
SMTTask["Volume"]
```

```
{50., 0., 0., 5., 0., 0.}
```

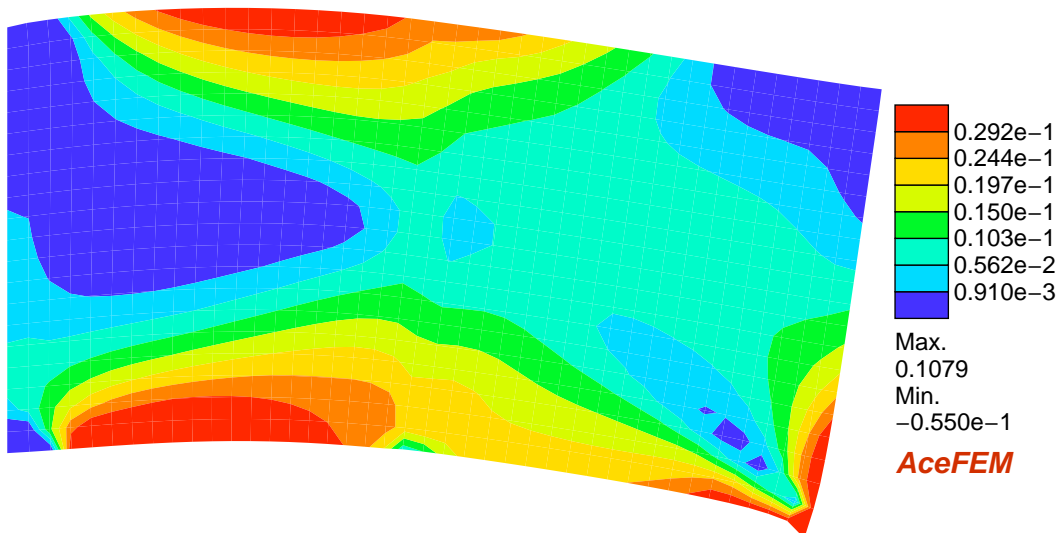
This shows the distribution of the Misses stress.

```
SMTShowMesh["DeformedMesh" → True, "Mesh" → False,  
"Field" → SMTTask["Misses"], "Contour" → True]
```



This shows the distribution of the sensitivity of Misses stress with respect to elastic modulus of the first domain E_1 .

```
SMTShowMesh["DeformedMesh" → True, "Mesh" → False,  
"Field" → SMTTask["MissesSensitivity", "IntegerInput" → {1}], "Contour" → True]
```

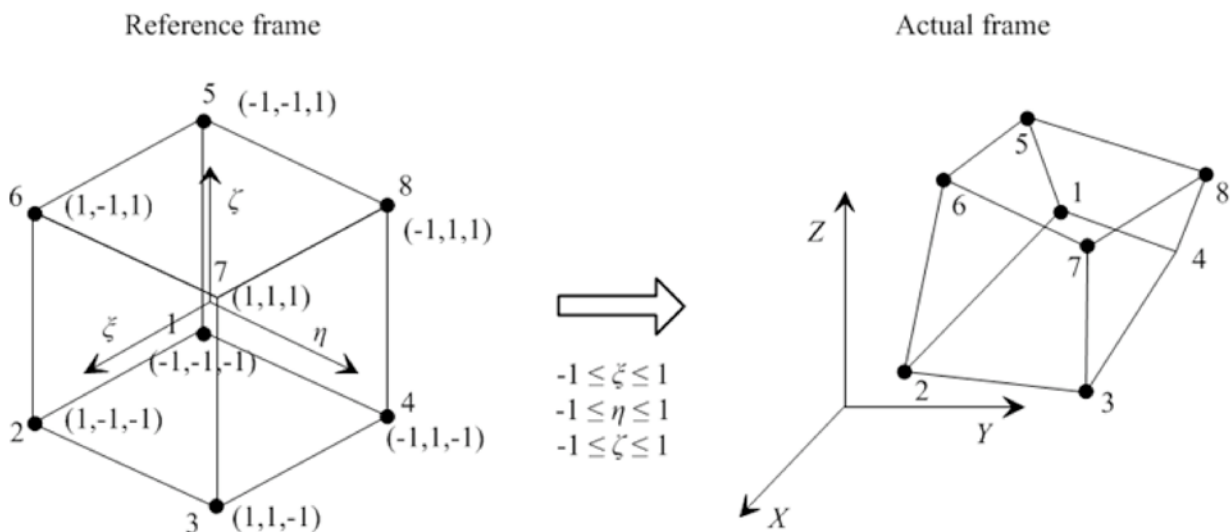


Three Dimensional, Elasto-Plastic Element

■ 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,
- ⇒ isoparametric 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.

```

<< AceGen` ;
SMSInitialize["ExamplesElastoPlastic3D",
  "VectorLength" → 1000, "Environment" -> "AceFEM"];
Ngh = 7; (* 7 state variables per integration point*)
Lgh = Ngh + 1;
(* add an additional history data for the elasto/plastic state indicator*)
Lhe = Lgh es$$["id", "NoIntPoints"];
SMSTemplate["SMSTopology" → "H1", "SMSSymmetricTangent" → True,
  "SMSNoTimeStorage" → Lhe,
  (*store the components of strain tensor for postprocessing*)
  "SMSNoElementData" → 6 es$$["id", "NoIntPoints"],
  "SMSPostIterationCall" → True,
  "SMSGroupDataNames" -> {"E -elastic modulus", "ν -poisson ration",
    "σy -yield stress", "ρ0 -density", "bX -force per unit mass X",
    "bY -force per unit mass Y", "bZ -force per unit mass Z"},
  "SMSDefaultData" -> {21 000, 0.3, 24, 1, 0, 0, 0}
];
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.

```

SMSDo[Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]];
Ξ = {ξ, η, ζ} + Table[SMSReal[es$$["IntPoints", i, Ig]], {i, 3}];

```

- Standard isoparametric 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 i -th node where $\{\xi_i, \eta_i, \zeta_i\}$ are the coordinates of the node.

```

XI + Table[SMSReal[nd$$[i, "X", j]], {i, SMSNoNodes}, {j, SMSNoDimensions}];
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}};
NI = Table[1 / 8 (1 + ξ Ξn[[i, 1]]) (1 + η Ξn[[i, 2]]) (1 + ζ Ξn[[i, 3]]), {i, 1, 8}];
X + SMSFreeze[NI.XI]; Jg = SMSD[X, Ξ]; Jgd = Det[Jg];

```

- Definition of the small strain tensor ϵ :

$$\mathbf{u} = \{u_i N_i, v_i N_i, w_i N_i\}$$

$$\mathbb{D} = \frac{\partial \mathbf{u}}{\partial \mathbf{X}}$$

$$\epsilon = \frac{1}{2} (\mathbb{D} + \mathbb{D}^T)$$

```

uI + SMSReal[Table[nd$$[i, "at", j], {i, SMSNoNodes}, {j, SMSNoDimensions}]];
pe = Flatten[uI]; u = NI.uI;
Dg = SMSD[u, X, "Dependency" → {Ξ, X, SMSInverse[Jg]}];

```

- 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 ($\frac{\partial Q_g}{\partial \epsilon}$). The "KeepStructure"→True option is necessary here to preserve the symmetry of ϵ .

```

SMSFreeze[ε, 1 / 2 (Dg + Transpose[Dg]), "KeepStructure" → True];

```

- The total strain tensor is stored in element "Data" filed for the post processing purposes.

```

SMSExport[{ε[[1, 1]], ε[[2, 2]], ε[[3, 3]], ε[[1, 2]], ε[[1, 3]], ε[[2, 3]]},
  Table[ed$$["Data", (Ig - 1) 6 + i], {i, 6}]];

```

- The \mathbf{hgn} 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 ${}^n \epsilon_p$ and the plastic multiplier $\lambda_{m,p}$ ($\mathbf{hgn} = \{{}^n \epsilon_{p,1,1}, {}^n \epsilon_{p,2,2}, {}^n \epsilon_{p,3,3}, {}^n \epsilon_{p,1,2}, {}^n \epsilon_{p,1,3}, {}^n \epsilon_{p,2,3}, {}^n \lambda_m\}$). The location of the \mathbf{hgn} vector within the

elements history variables field $ed\$\$[hp,i]$ is calculated and stored into the Igh variable. An additional history variable "state" stores information whether the material point is in elastic regime ($state=0$) or in plastic regime ($state=1000+\mathcal{F}$).

$ed\$\$[hp ",Igh+ i]$	i – th component of the history variables at the end of the previous time step in current Gauss point
$ed\$\$[ht, Igh + i]$	i – th component of the current history variables in current Gauss point

```
Igh = SMSInteger [ (Ig - 1) Lgh ] ;
hhgn = Table [ SMSReal [ ed$$ [ "hp" , Igh + i ] ] , { i , Lgh } ] ;
hgn = hhgn [ 1 ; ; Ngh ] ; state = hhgn [ Ngh + 1 ] ;
```

- Here the material constants are defined.

```
{Em, ν, σy, ρ0, bX, bY, bZ} =
SMSReal [ Table [ es$$ [ "Data" , i ] , { i , Length [ SMSGroupDataNames ] } ] ] ;
{λ, μ} = SMSHookeToLame [ Em, ν ] ;
bb = {bX, bY, bZ} ;
```

- The $WFQ[task_hgt_]$ function calculates accordingly to the task required the yield function \mathcal{F} , the system of local equations \mathbf{Q} or elastic strain energy W as follows:

$$\epsilon_e = \epsilon - \epsilon_p$$

$$W = W(\epsilon_e)$$

$$\sigma = \sigma(\epsilon_e)$$

$$s = \sigma - \frac{1}{3} I \text{tr } \sigma$$

$$\mathcal{F} = \sqrt{\frac{3}{2} s : s} - \sigma_y$$

$$\mathcal{A} = \frac{\partial \mathcal{F}}{\partial \sigma}$$

$$\mathcal{Z} = \epsilon_p - \epsilon_p - (\lambda_m - \lambda_m) \mathcal{A}$$

$$\mathbf{Q}_g = \{\mathcal{Z}[1, 1], \mathcal{Z}[2, 2], \mathcal{Z}[3, 3], \mathcal{Z}[1, 2], \mathcal{Z}[1, 3], \mathcal{Z}[2, 3], \mathcal{F}\} = \mathbf{0}$$

```
W $\mathcal{F}$ Q[task_, hgt_] :=
```

```
(
  

|          |          |          |
|----------|----------|----------|
| hgt[[1]] | hgt[[4]] | hgt[[5]] |
| hgt[[4]] | hgt[[2]] | hgt[[6]] |
| hgt[[5]] | hgt[[6]] | hgt[[3]] |


  ;  $\lambda m = \text{hgt}[[7]]$ ;

  SMSFreeze[ee,  $\epsilon - \text{ep}$ , "Symmetric"  $\rightarrow$  True];
  W = Simplify[ $\lambda / 2 \text{Tr}[\text{ee}]^2 + \mu \text{Tr}[\text{ee}.\text{ee}]$ ];
  If[task == "W", Return[]];
  SMSFreeze[ $\sigma$ , SMSD[W, ee, "Symmetric"  $\rightarrow$  True], "Symmetric"  $\rightarrow$  True];
  s =  $\sigma - 1 / 3 \text{IdentityMatrix}[3] \text{Tr}[\sigma]$ ;
   $\mathcal{F} = \text{SMSSqrt}[3 / 2 \text{Tr}[s.s]] - \sigma$ ;
  If[task == " $\mathcal{F}$ ", Return[]];
   $\mathcal{A} = \text{Simplify}[\text{SMSD}[\mathcal{F}, \sigma, "Symmetric" \rightarrow \text{True}]]$ ;

|          |          |          |
|----------|----------|----------|
| hgn[[1]] | hgn[[4]] | hgn[[5]] |
| hgn[[4]] | hgn[[2]] | hgn[[6]] |
| hgn[[5]] | hgn[[6]] | hgn[[3]] |


  ;  $\lambda mn = \text{hgn}[[7]]$ ;

  Z =  $\text{ep} - \text{epn} - (\lambda m - \lambda mn) \mathcal{A}$ ;
  Qg = Append[ $\{\text{Z}[[1, 1]], \text{Z}[[2, 2]], \text{Z}[[3, 3]], \text{Z}[[1, 2]], \text{Z}[[1, 3]], \text{Z}[[2, 3]]\}$ ,  $\mathcal{F}$ ];

  If[task == " $\mathcal{F}$ Q", Return[]];
)
```

```
W $\mathcal{F}$ Q[" $\mathcal{F}$ ", hgn];
```

```
iNR = SMSInteger[idata$$["Iteration"]];
```

```
SMSIf[ (iNR == 1 && state == 0) || (iNR > 1 &&  $\mathcal{F} < \frac{1}{10^8}$ ) ];
```

- 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.

```
hg = hgn;
```

```
SMSExport[Join[hgn, {0}], Table[ed$$["ht", Igh + i], {i, Lgh}]];
```

```
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} 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 ($D\mathbf{h}_g D\epsilon := \frac{\mathbf{h}_g^{(j)}}{\partial \epsilon}$). Implicit dependencies are obtained by definition $A \frac{\mathbf{h}_g^{(j)}}{\partial \epsilon} = -\frac{\partial \mathbf{Q}}{\partial \epsilon} \implies \frac{\mathbf{h}_g^{(j)}}{\partial \epsilon}$.

The SMSVariables[ϵ] function returns the true independent variables in ϵ with the symmetry of ϵ correctly considered.

```

hgj ← hgn;
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] < 1 / 10^9,
  (*the operator ←
  is necessary here because the δh will be exported out from the loop *)
  DhDe ← SMSLUSolve[LU, -SMSD[Qg, SMSVariables[ε], "Constant" → hgj]];
  (*The values of the state variables
  are stored back to the history data of the element.*)
  SMSExport[Join[hgj, {1000 + SMSAbs[f]}], Table[ed$$["ht", Igh + i], {i, Lgh}]];
  (*exit the sub-iterative loop when the local equations are satisfied*)
  SMSBreak[];
];
SMSIf[jNR == "29"
  , SMSExport[{1, 2}, {idata$$["SubDivergence"], idata$$["ErrorStatus"]}];
  (*exit the sub-iterative loop if the convergence was not reached*)
  SMSBreak[];
];
SMSEndDo[hgj, DhDe];

```

- 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 $\frac{\partial h_g}{\partial \epsilon}$ (Exceptions in Differentiation).

```
hg ← SMSFreeze[hgj, "Dependency" → {SMSVariables[ε], DhDe}];
```

```
SMSEndIf[hg];
```

- 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 .

```

WfQ["W", hg];
wgp ← SMSReal[es$$["IntPoints", 4, Ig]];
SMSDo [
  Rg ← Jgd wgp SMSD[W - ρ0 u.bb, pe, i, "Constant" → hg];
  SMSExport[SMSResidualSign Rg, p$$[i], "AddIn" → True];
  SMSDo [
    Kg ← SMSD[Rg, pe, j];
    SMSExport[Kg, s$$[i, j], "AddIn" → True];
    , {j, i, SMSNoDOFGlobal}];
    , {i, 1, SMSNoDOFGlobal}];

```

- This is the end of the numerical integration loop.

```
SMSEndDo[];
```



```

SMSStandardModule["Postprocessing"];
SMSDo [
  Igh = SMSInteger[(Ig - 1) Lgh];
  hhg = Table[SMSReal[ed$$["ht", Igh + i]], {i, Lgh}];
  hg = hhg[[1 ;; Ngh]]; state = hhg[[Ngh + 1]]; hgn = hg;
  {Em, ν, σy, ρ0, bX, bY, bZ} =
    SMSReal[Table[es$$["Data", i], {i, Length[SMSGGroupDataNames]}]];
  {λ, μ} = SMSHookeToLame[Em, ν];
  ei = Table[SMSReal[ed$$["Data", (Ig - 1) 6 + i]], {i, 6}];

```

	ei[[1]]	ei[[4]]	ei[[5]]
ε =	ei[[4]]	ei[[2]]	ei[[6]]
	ei[[5]]	ei[[6]]	ei[[3]]

```

  WFQ["f", hg];
  SMSGPostNames = {"Sxx", "Sxy", "Sxz", "Syx", "Syy", "Syz", "Szx", "Szy", "Szz",
    "Exx", "Exy", "Exz", "Eyx", "Eyy", "Eyz", "Ezx", "Ezy", "Ezz",
    "Exxp", "Exyp", "Exzp", "Eyxp", "Eyyp", "Eyzp", "Ezxp", "Ezyp", "Ezzp",
    "Accumulated plastic deformation", "State 0-elastic 1000+f -plastic"};
  SMSExport[Flatten[{σ, ε, ep, λm, state}], gpost$$[Ig, #1] &];
, {Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]}];

```

```

SMSNPostNames =
  {"DeformedMeshX", "DeformedMeshY", "DeformedMeshZ", "u", "v", "w"};
uI = SMSReal[Table[nd$$[i, "at", j], {i, SMSNoNodes}, {j, SMSNoDimensions}]];
SMSExport[Table[Join[uI[[i]], uI[[i]], {i, SMSNoNodes}], npost$$];

```

- Here the element source code is written in "ExamplesElastoPlastic3D.c" file.

```
SMSWrite[];
```

File:	ExamplesElastoPlastic3D.c	Size:	33 170
Methods	No. Formulae	No. Leafs	
SKR	538	8905	
SPP	54	820	

■ Analysis

```

<< AceFEM` ;
SMTInputData[];
SMTAddDomain["A", "ExamplesElastoPlastic3D",
  {"E *" -> 1000., "ν *" -> .3, "σy *" -> 10.}];
SMTAddEssentialBoundary[{"X" == 0 &, 1 -> 0, 2 -> 0, 3 -> 0}, {"X" == 10 &, 3 -> -1}];
SMTMesh["A", "H1", {15, 6, 6}, {{{{0, 0, 0}, {10, 0, 0}}, {{0, 2, 0}, {10, 2, 0}}},
  {{{{0, 0, 3}, {10, 0, 2}}, {{0, 2, 3}, {10, 2, 2}}}}];
SMTAnalysis[];

```

```

SMTNextStep[1, 0.1];
While[
  While[step = SMTConvergence[10^-8, 10, {"Adaptive", 8, .001, .1, 0.5}],
    SMTNewtonIteration[]];
  SMTStatusReport[];
  If[step[[4]] == "MinBound", SMTStatusReport[]];
  step[[3]]
  , If[step[[1]], SMTStepBack[]];
  SMTNextStep[1, step[[2]]]
];

T/ΔT=1./1. λ/Δλ=0.1/0.1 ||Δa||/||Ψ||=2.07431 × 10-15
/1.77281 × 10-15 Iter/Total=3/3 Status=0/{Convergence}

T/ΔT=2./1. λ/Δλ=0.2/0.1 ||Δa||/||Ψ||=1.02853 × 10-15
/3.41942 × 10-15 Iter/Total=3/6 Status=0/{Convergence}

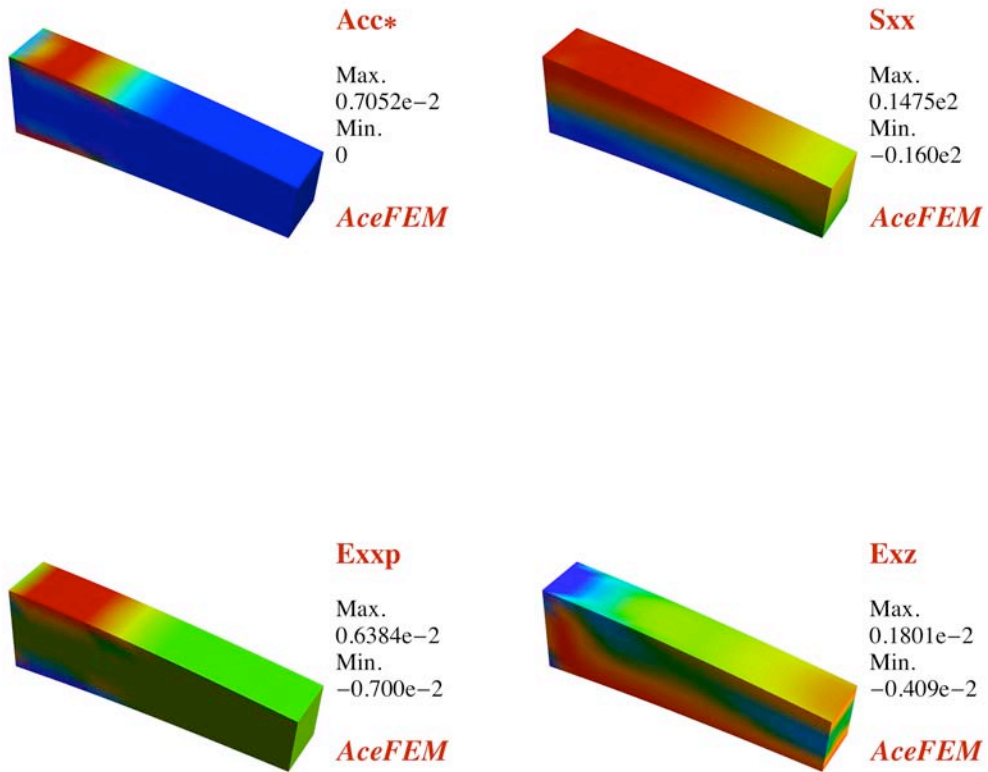
T/ΔT=3./1. λ/Δλ=0.3/0.1 ||Δa||/||Ψ||=1.85154 × 10-15
/5.08353 × 10-15 Iter/Total=3/9 Status=0/{Convergence}

T/ΔT=4./1. λ/Δλ=0.4/0.1 ||Δa||/||Ψ||=3.14954 × 10-9
/2.50188 × 10-13 Iter/Total=5/14 Status=0/{Convergence}

T/ΔT=5./1. λ/Δλ=0.5/0.1 ||Δa||/||Ψ||=2.03555 × 10-11
/8.1646 × 10-15 Iter/Total=6/20 Status=0/{Convergence}

```

```
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,
- ⇒ isoparametric 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 kinematical 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 C_p^{-1} , a plastic multiplier γ_m ($\mathbf{h}_g = \{C_{p11}^{-1}, C_{p22}^{-1}, C_{p33}^{-1}, C_{p12}^{-1}, \gamma_m\}$). 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 ($F_{1,1}, F_{1,2}, F_{2,1}, F_{2,2}, F_{3,3}$) are also stored for post-processing as arbitrary real values per element (SMSNoElementData).

■ Solution

- All steps are described in detail in example Three Dimensional, Elasto-Plastic Element .

<< AceGen` ;

```

WFQ[task_, hgt_] :=
(
  Cgpi = IdentityMatrix[3] + 

|          |          |          |
|----------|----------|----------|
| hgt[[1]] | hgt[[4]] | 0        |
| hgt[[4]] | hgt[[2]] | 0        |
| 0        | 0        | hgt[[3]] |

;

   $\gamma$  = hgt[[5]];
  SMSFreeze[be, F.Cgpi.Transpose[F], "Ignore" -> (# === 0 &), "Symmetric" -> True];
  Jbe = Det[be];
  W =  $\kappa / 2 (1 / 2 (Jbe - 1) - 1 / 2 \text{Log}[Jbe]) + \mu / 2 (\text{Tr}[Jbe^{-1/3} be] - 3)$ ;
  If[task == "W", Return[]];
  SMSFreeze[ $\tau$ ,
    Simplify[2 be.SMSD[W, be, "IgnoreNumbers" -> True, "Symmetric" -> True]],
    "KeepStructure" -> True];
   $s = \tau - \frac{1}{3} \text{IdentityMatrix}[3] \text{Tr}[\tau]$ ;
   $\alpha = \sqrt{2/3} \gamma$ ;
   $\sigma_Y = \sqrt{2/3} (\sigma_{Y0} + K_f \alpha + (Y_{inf} - \sigma_{Y0}) (1 - \text{Exp}[-\delta \alpha]))$ ;
   $\mathcal{F} = \text{SMS} \sqrt{\text{Total}[s s, 2]} - \sigma_Y$ ;
  If[task == "F", Return[]];

  Cgpin = IdentityMatrix[3] + 

|          |          |          |
|----------|----------|----------|
| hgn[[1]] | hgn[[4]] | 0        |
| hgn[[4]] | hgn[[2]] | 0        |
| 0        | 0        | hgn[[3]] |

;

   $\gamma_n = \text{hgn}[[5]$ ;
   $\mathcal{A} = \text{SMSD}[\mathcal{F}, \tau, "IgnoreNumbers" -> True, "Symmetric" -> True]$ ;
   $M = -2 (\gamma - \gamma_n) \mathcal{A}$ ;
   $Z = \text{Simplify}[F.Cgpi - \text{SMSMatrixExp}[M].F.Cgpin]$ ;
  Qg = {Z[[1, 1]], Z[[2, 2]], Z[[3, 3]], Z[[1, 2]],  $\mathcal{F}$ };

  If[task == "FQ", Return[]];
)

<< AceGen` ;
SMSInitialize["ExamplesFiniteStrain",
  "VectorLength" -> 3000, "Environment" -> "AceFEM"];
Ngh = 5; Lgh = Ngh + 1; Lhe = Lgh es$$["id", "NoIntPoints"];
SMSTemplate["SMSTopology" -> "Q1", "SMSSymmetricTangent" -> True,
  "SMSNoTimeStorage" -> Lhe,
  "SMSNoElementData" -> 5 es$$["id", "NoIntPoints"],
  "SMSPostIterationCall" -> True,
  "SMSGroupDataNames" -> {"E -elastic modulus", " $\nu$  -poisson ration",
    " $\sigma_Y$  -initial yield stress", "K -hardening coefficient",
    " $\sigma_{Yinf}$  -residual flow stress", " $\delta$  -saturation exponent", " $\rho_0$  -density",
    "bX -force per unit mass X", "bY -force per unit mass Y"},
  "SMSDefaultData" -> {21 000, 0.3, 24, 0, 24, 0, 1, 0, 0}
];
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]];
 $\Xi = \{\xi, \eta, \zeta\} \vdash \text{Table}[\text{SMSReal}[es$$["IntPoints", i, Ig]], \{i, 3\}]$ ;
XI = Table[SMSReal[nd$$[i, "X", j]], {i, SMSNoNodes}, {j, SMSNoDimensions}];
NI = 1/4 {(1 -  $\xi$ ) (1 -  $\eta$ ), (1 +  $\xi$ ) (1 -  $\eta$ ), (1 +  $\xi$ ) (1 +  $\eta$ ), (1 -  $\xi$ ) (1 +  $\eta$ )};
X = SMSFreeze[Append[NI.XI,  $\zeta$ ]]; Jg = SMSD[X,  $\Xi$ ]; Jgd = Det[Jg];

```

```

{X0, J0d} = SMSReplaceAll[{X[[1]], Jgd},  $\xi \rightarrow 0, \eta \rightarrow 0$ ];
{Nx, Ny} = Table[SMSD[NI, X[[i]], "Dependency"  $\rightarrow$  { $\Xi$ , X, SMSInverse[Jg]}], {i, 2}];
{NX $\xi$ , NX $\eta$ , NY $\xi$ , NY $\eta$ } = {SMSD[Nx,  $\xi$ ], SMSD[Nx,  $\eta$ ], SMSD[Ny,  $\xi$ ], SMSD[Ny,  $\eta$ ]};
{NX $\xi$ 0, NX $\eta$ 0, NY $\xi$ 0, NY $\eta$ 0} = SMSReplaceAll[{NX $\xi$ , NX $\eta$ , NY $\xi$ , NY $\eta$ },  $\xi \rightarrow 0, \eta \rightarrow 0$ ];
{V, V0} =
  Simplify[Integrate[SMSRestore[{X[[1]] Jgd, X0 J0d},  $\xi | \eta$ ] // Normal // Simplify,
    { $\xi$ , -1, 1}, { $\eta$ , -1, 1}]];
NX0 =  $\frac{1}{V}$  Simplify[Integrate[SMSRestore[Jgd X[[1]] Nx,  $\xi | \eta$ ] // Normal // Simplify,
  { $\xi$ , -1, 1}, { $\eta$ , -1, 1}]];
NY0 =  $\frac{1}{V}$  Simplify[Integrate[SMSRestore[Jgd X[[1]] Ny,  $\xi | \eta$ ] // Normal // Simplify,
  { $\xi$ , -1, 1}, { $\eta$ , -1, 1}]]; N $\phi$  =  $\frac{1}{V}$  Simplify[
  Integrate[SMSRestore[Jgd NI,  $\xi | \eta$ ] // Normal // Simplify, { $\xi$ , -1, 1}, { $\eta$ , -1, 1}]];
uI = SMSReal[Table[nd $\xi\xi$ [i, "at", j], {i, SMSNoNodes}, {j, SMSNoDimensions}]];
pe = Flatten[uI]; {un, vn} = Transpose[uI];
u = Append[NI.uI, 0];
Dv =  $\frac{1}{3}$  (NX0.un + NY0.vn + N $\phi$ .un) IdentityMatrix[3];
dv = (NX0 + NX $\xi$ 0  $\xi$  + NX $\eta$ 0  $\eta$ ).un + (NY0 + NY $\xi$ 0  $\xi$  + NY $\eta$ 0  $\eta$ ).vn + N $\phi$ .un;
Dd =  $\left( \begin{array}{ccc} (NX0 + NX\mathit{\xi}0 \mathit{\xi} + NX\mathit{\eta}0 \mathit{\eta}).un - \frac{1}{3} dv & NY0.un & 0 \\ NX0.vn & (NY0 + NY\mathit{\xi}0 \mathit{\xi} + NY\mathit{\eta}0 \mathit{\eta}).vn - \frac{1}{3} dv & 0 \\ 0 & 0 & N\mathit{\phi}.un - \frac{1}{3} dv \end{array} \right)$ ;
Dg = Simplify[Dv + Dd];
SMSFreeze[F, IdentityMatrix[3] + Dg, "KeepStructure"  $\rightarrow$  True];
SMSExport[{F[[1, 1]], F[[1, 2]], F[[2, 1]], F[[2, 2]], F[[3, 3]]},
  Table[ed $\xi\xi$ ["Data", (Ig - 1) 5 + i], {i, 5}]];
Igh = SMSInteger[(Ig - 1) Lgh];
hhgn = Table[SMSReal[ed $\xi\xi$ ["hp", Igh + i]], {i, Lgh}];
hgn = hhgn[[1 ;; Ngh]]; state = hhgn[[Ngh + 1]]; {Em,  $\nu$ ,  $\sigma$ y0, Kf, Yin $\mathit{f}$ ,  $\delta$ ,  $\rho$ 0, bX, bY} =
  Array[SMSReal[es $\xi\xi$ ["Data", #1]] &, SMSGroupDataNames // Length];
{ $\mu$ ,  $\kappa$ } = SMSHookeToBulk[Em,  $\nu$ ];
bb = {bX, bY, 0};

W $\mathcal{F}$ Q[" $\mathcal{F}$ ", hgn];
iNR = SMSInteger[idata $\xi\xi$ ["Iteration"]];
SMSIf[(iNR == 1 && state == 0) || (iNR > 1 &&  $\mathcal{F} < \frac{1}{10^8}$ )];

hg = hgn;
SMSExport[Join[hgn, {0}], Table[ed $\xi\xi$ ["ht", Igh + i], {i, Lgh}]];
SMSElse[];

```

```

hgj ← hgn;
SMSDo [jNR, 1, 30, 1, hgj];
W $\mathcal{F}$ Q[" $\mathcal{F}$ Q", hgj];
A ← SMSD[Qg, hgj];
LU ← SMSLUFactor[A];
 $\Delta$ h ← SMSLUSolve[LU, -Qg];
hgj ← hgj +  $\Delta$ h;
SMSIf[Sqrt[ $\Delta$ h. $\Delta$ h] < 1 / 109,
  (*the opearator ←
  is neceserry here because the  $\delta$ he 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.*)
  SMSExport[Join[hgj, {1000 + SMSAbs[ $\mathcal{F}$ ]}], Table[ed$$["ht", Igh + i], {i, Lgh}]];
  (*exit the sub-iterative loop when the local equations are satisfied*)
  SMSBreak[];
];
SMSIf[jNR == "29"
  , SMSExport[{1, 2}, {idata$$["SubDivergence"], idata$$["ErrorStatus"]}];
  (*exit the sub-iterative loop if the convergence was not reached*)
  SMSBreak[];
];
SMSEndDo[hgj, DhDF];
hg ← SMSFreeze[hgj, "Dependency" → {SMSVariables[F], DhDF}];

  Closed form solution of matrix exponent.
  See also: SMSMatrixExp

SMSEndIf[hg];

W $\mathcal{F}$ Q["W", hg];
wgp ← SMSReal[es$$["IntPoints", 4, Ig]];
SMSDo [
  
$$Rg = 2\pi \frac{v}{v_0} X_0 J_0 d \text{ wgp SMSD}[W - \rho_0 \text{ u.bb, pe, i, "Constant" } \rightarrow \text{hg}];$$

  SMSExport[SMSResidualSign Rg, p$$[i], "AddIn" → True];
  SMSDo [
    Kg ← SMSD[Rg, pe, j];
    SMSExport[Kg, s$$[i, j], "AddIn" → True];
    , {j, i, SMSNoDOFGlobal}];
    , {i, 1, SMSNoDOFGlobal}];
];
SMSEndDo[];

```

```

SMSStandardModule["Postprocessing"];
SMSDo[
  Igh = SMSInteger[(Ig - 1) Lgh];
  hhg = Table[SMSReal[ed$$["ht", Igh + i]], {i, Lgh}];
  hg = hhg[[1 ;; Ngh]]; state = hhg[[Ngh + 1]]; hgn = hg;
  {Em, v,  $\sigma_0$ , Kf, Yin,  $\delta$ ,  $\rho_0$ , bX, bY} =
    Array[SMSReal[es$$["Data", #1]] &, SMSGroupDataNames // Length];
  { $\mu$ ,  $\kappa$ } = SMSHookeToBulk[Em, v];
  Fi = Table[SMSReal[ed$$["Data", (Ig - 1) 5 + i]], {i, 5}];
  F = 

|         |         |         |
|---------|---------|---------|
| Fi[[1]] | Fi[[2]] | 0       |
| Fi[[3]] | Fi[[4]] | 0       |
| 0       | 0       | Fi[[5]] |

;
  WFQ[" $\mathcal{F}$ ", hg];
  Eg = 1 / 2 (Transpose[F].F - IdentityMatrix[3]);
   $\sigma$  =  $\tau$  / Det[F];
  Egp = 1 / 2 (Inverse[Cgpi] - IdentityMatrix[3]);
  SMSGPostNames = {"Sxx", "Sxy", "Sxz", "Syx", "Syy", "Syz", "Szx", "Szy", "Szz",
    "Exx", "Exy", "Exz", "Eyx", "Eyy", "Eyz", "Ezx", "Ezy", "Ezz",
    "Exxp", "Exyp", "Exzp", "Eyxp", "Eyyp", "Eyzp", "Ezxp", "Ezyp", "Ezzp",
    "Accumulated plastic deformation", "State 0-elastic 1000+f -plastic"};
  SMSExport[Flatten[{ $\sigma$ , Eg, Egp,  $\sqrt{2/3}$   $\gamma$ , state}], gpost$$[Ig, #1] &];
, {Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]}];
SMSNPostNames = {"DeformedMeshX", "DeformedMeshY", "u", "v"};
uI = SMSReal[Table[nd$$[i, "at", j], {i, SMSNoNodes}, {j, SMSNoDimensions}]];
SMSExport[Table[Join[uI[[i]], uI[[i]], {i, SMSNoNodes}], npost$$];

SMSWrite[];

```

File:	ExamplesFiniteStrain.c	Size:	71 142
Methods	No.Formulae	No.Leafs	
SKR	1365	21 061	
SPP	64	1067	

Cyclic tension test, advanced post-processing , animations

With the use of the elements generated in Problem 8 the following cyclic plasticity example is analysed.

■ Analysis Session

```

<< 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; ne = 10;
ΔL = 0.16;
SMTMesh["A", "Q1", {2 ne, ne},
  {{{0.4, 0}, {0.4, 0.4}}, {{0, 0}, {0, 0.4}}}, "InterpolationOrder" -> 1];
SMTMesh["A", "Q1", {ne, ne}, {{{0.4, 0.4}, {0.4, 0.8}}, {{0, 0.4}, {0, 0.8}}},
  "InterpolationOrder" -> 1];
SMTMesh["A", "Q1", {ne, ne},
  {{{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}}}, "InterpolationOrder" -> 1];
SMTMesh["A", "Q1", {ne / 2, ne}, {{{0.5, 1.1}, {0.5, 1.6}}, {{0, 1.1}, {0, 1.6}}},
  "InterpolationOrder" -> 1];
SMTAddEssentialBoundary[{ "X" == 0. &, 1 -> 0},
  { "Y" == 0. &, 2 -> 0}, { "Y" == L &, 2 -> ΔL}];

```

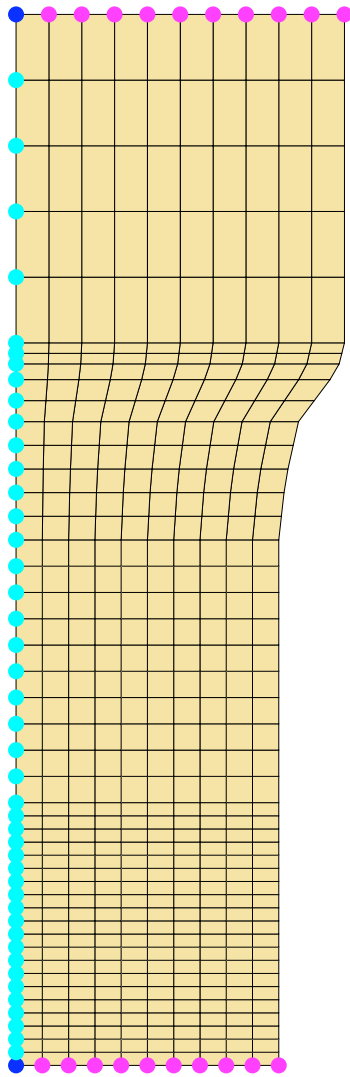
Here the post-processing data base file name ("cyclic") is specified.

```
SMTAnalysis["Output" -> "tmp.out", "PostOutput" -> "cyclic"];
```

With the SMTSave command definitions of arbitrary number of symbols can be storder into the post-processing data base.

```
SMTSave[ΔL, L, R];
```

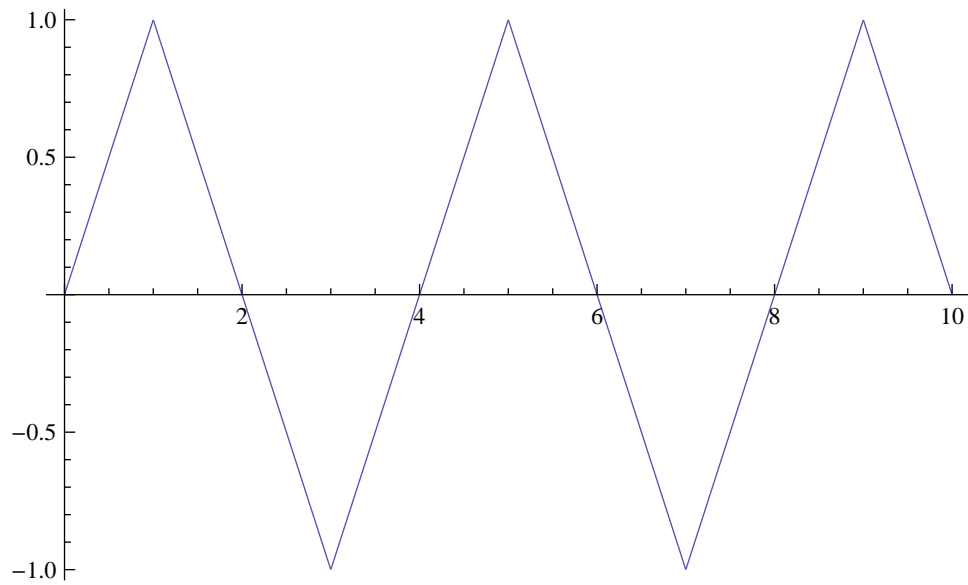
```
SMTShowMesh["BoundaryConditions" → True]
```



Here the cyclic loading simulation is performed. Function $\lambda(t)$ 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 informations from data base. Additionally the resulting force at the top of the specimen is also stored into data base.

```
Clear[λ]; λ[t_] := If[OddQ[Floor[(t + 1) / 2]], 1, -1] (2 Floor[(t + 1) / 2] - t);
```

```
Plot[λ[t], {t, 0, 10}]
```



```
SMTNextStep[0.01, 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[SMTData["Time"], force, "TimeFrequency" → 0.1];
```

```
    SMTShowMesh["DeformedMesh" → True, "Field" → "Acc*", "Show" → "Window"];
```

```
  ];
```

```
  If[step[[4]] === "MinBound", SMTStatusReport["ΔT<ΔTmin"];];
```

```
  step[[3]]
```

```
  , If[step[[1]], SMTStepBack[]];];
```

```
  SMTNextStep[step[[2]], λ]
```

```
];
```

```

SMTSimulationReport[];

No. of nodes           506
No. of elements       450
No. of equations      944
Data memory (KBytes)  391
Number of threads used/max  8/8
Solver memory (KBytes) 11 699
No. of steps          115
No. of steps back     23
Step efficiency (%)   83.3333
Total no. of iterations 864
Average iterations/step 6.26087
Total time (s)        84.1
Total linear solver time (s) 3.607
Total linear solver time (%) 4.28894
Total assembly time (s) 13.166
Total assembly time (%) 15.6552
Average time/iteration (s) 0.097338
Average linear solver time (s) 0.00417477
Average K and R time (s) 0.0000338632
Total Mathematica time (s) 16.521
Total Mathematica time (%) 19.6445
USER-IData
USER-RData

```

■ Postprocessing Session

Here the new, independent session starts by reading input data from previously stored post-processing data base file.

```

<< AceFEM` ;
SMTInputData[];
SMTAnalysis["PostInput" -> "cyclic"];

```

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

```

{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}

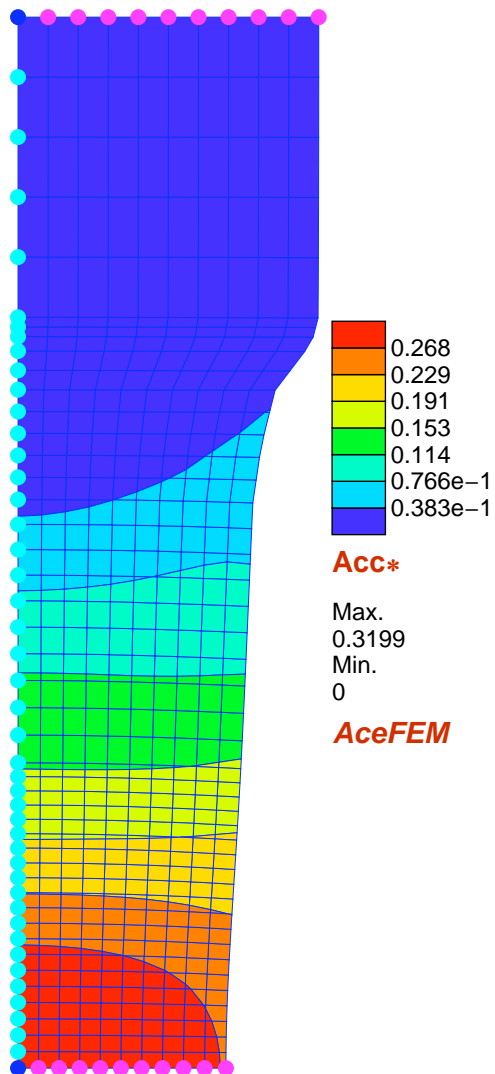
```

The SMTGet[key] command reads the information storder under the keyword allkeys[[10]].

```
force = SMTGet[allkeys[[10]]]
allkeys[[10]]
SMTShowMesh["BoundaryConditions" → True, "DeformedMesh" → True,
  "Field" -> "Acc*", "Marks" → False, "Contour" → True]

{-0.242828}

1.07279
```



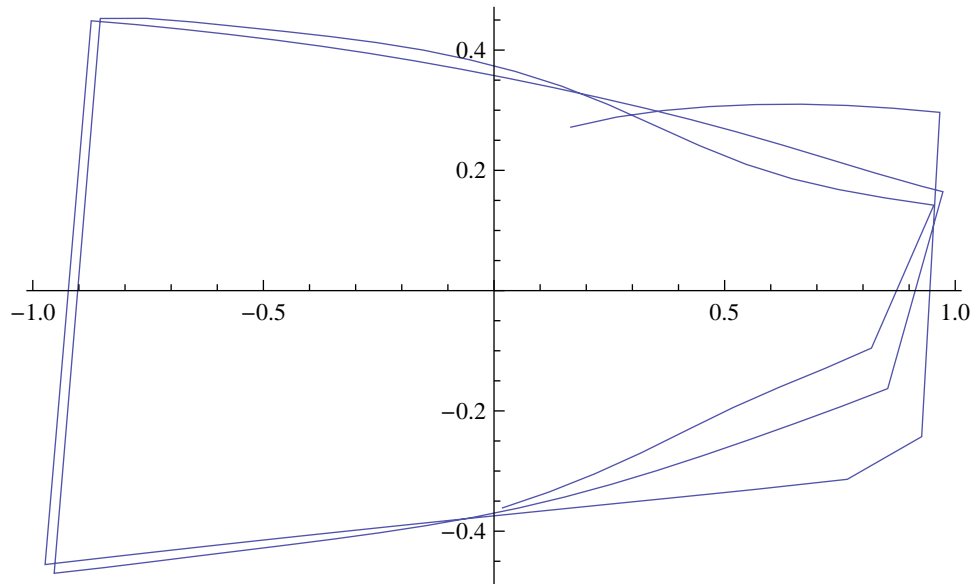
Here all post-processing quantities are evaluated at point {R,0}. Note that the value of "R" was stored by the SMTSave command at the analysis time and restored by the SMTAnalysis["PostInput"->"cyclic"] command.

```
{Range[allpost // Length], allpost, SMTPostData[allpost, {R, 0}]} // Transpose //
TableForm
```

1	Accumulated plastic deformation	0.265322
2	DeformedMeshX	-0.0541688
3	DeformedMeshY	0.
4	E _{xx}	-0.259427
5	E _{xxp}	-0.106615
6	E _{xy}	-0.394906
7	E _{xyp}	0.000202251
8	E _{xz}	0.
9	E _{xzp}	0.
10	E _{yx}	-0.394906
11	E _{yxp}	0.000202251
12	E _{yy}	0.319531
13	E _{ryp}	0.348849
14	E _{yz}	0.
15	E _{yzp}	0.
16	E _{zx}	0.
17	E _{zxp}	0.
18	E _{zy}	0.
19	E _{ryp}	0.
20	E _{zz}	0.00266447
21	E _{zrp}	-0.12568
22	State 0-elastic 1000+f -plastic	0.
23	S _{xx}	0.
24	S _{xy}	0.
25	S _{xz}	0.
26	S _{yx}	0.
27	S _{yy}	0.
28	S _{yz}	0.
29	S _{zx}	0.
30	S _{zy}	0.
31	S _{zz}	0.
32	u	-0.0541688
33	v	0.

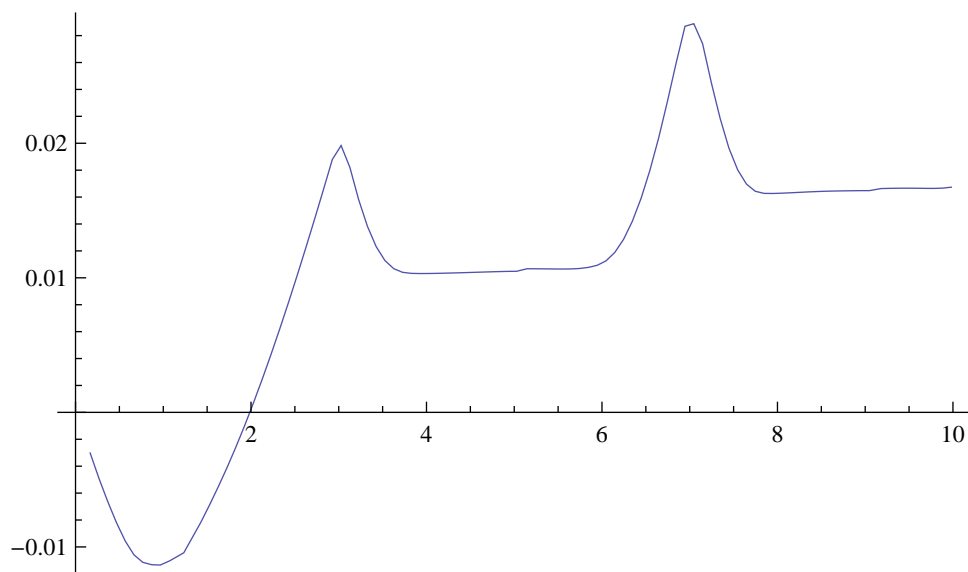
Here the $f(\lambda)$ diagram is presented.

```
ListLinePlot[
  Map[(force = SMTGet[#][[1]]; {SMTRData["Multiplier"], force}) &, allkeys]]
```

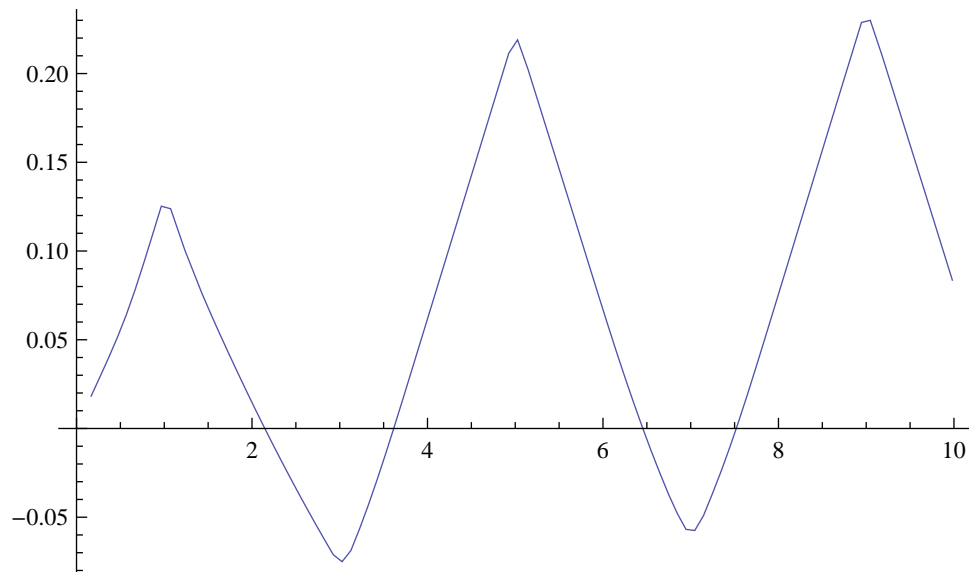


Here the evolution of some (u , v , plastic deformation, σ_{yy}) postprocessing quantities in point $\{R/2, L/3\}$ is presented.

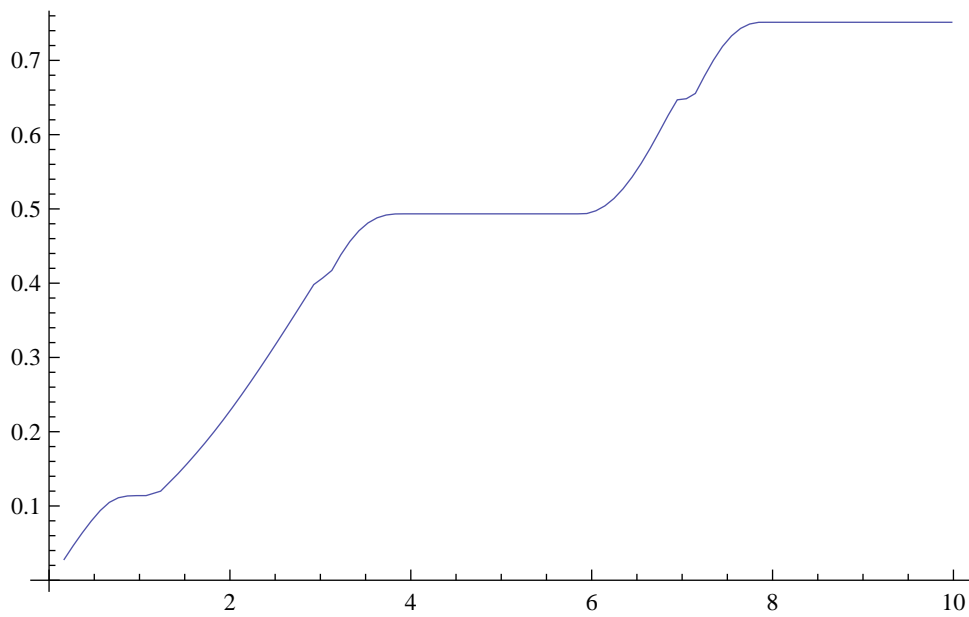
```
ListLinePlot[Map[{#, SMTGet[#]; SMPPostData["u", {R / 2, L / 3.}]} &, allkeys]]
```



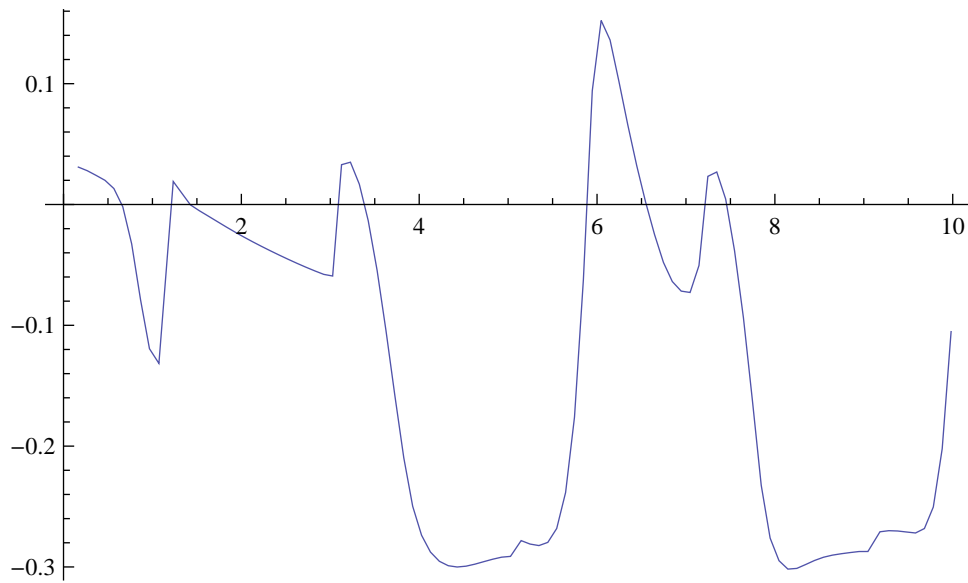
```
ListLinePlot[Map[#, SMTGet[#]; SMTPostData["v", {R/2, L/3.}]] &, allkeys]]
```



```
ListLinePlot[Map[#, SMTGet[#]; SMTPostData["Acc*", {R/2, L/3.}]] &, allkeys]]
```




```
ListLinePlot[Map[#, SMTGet[#]; SMTPostData["Sxx", {R/2, L/3.}]] &, allkeys]]
```

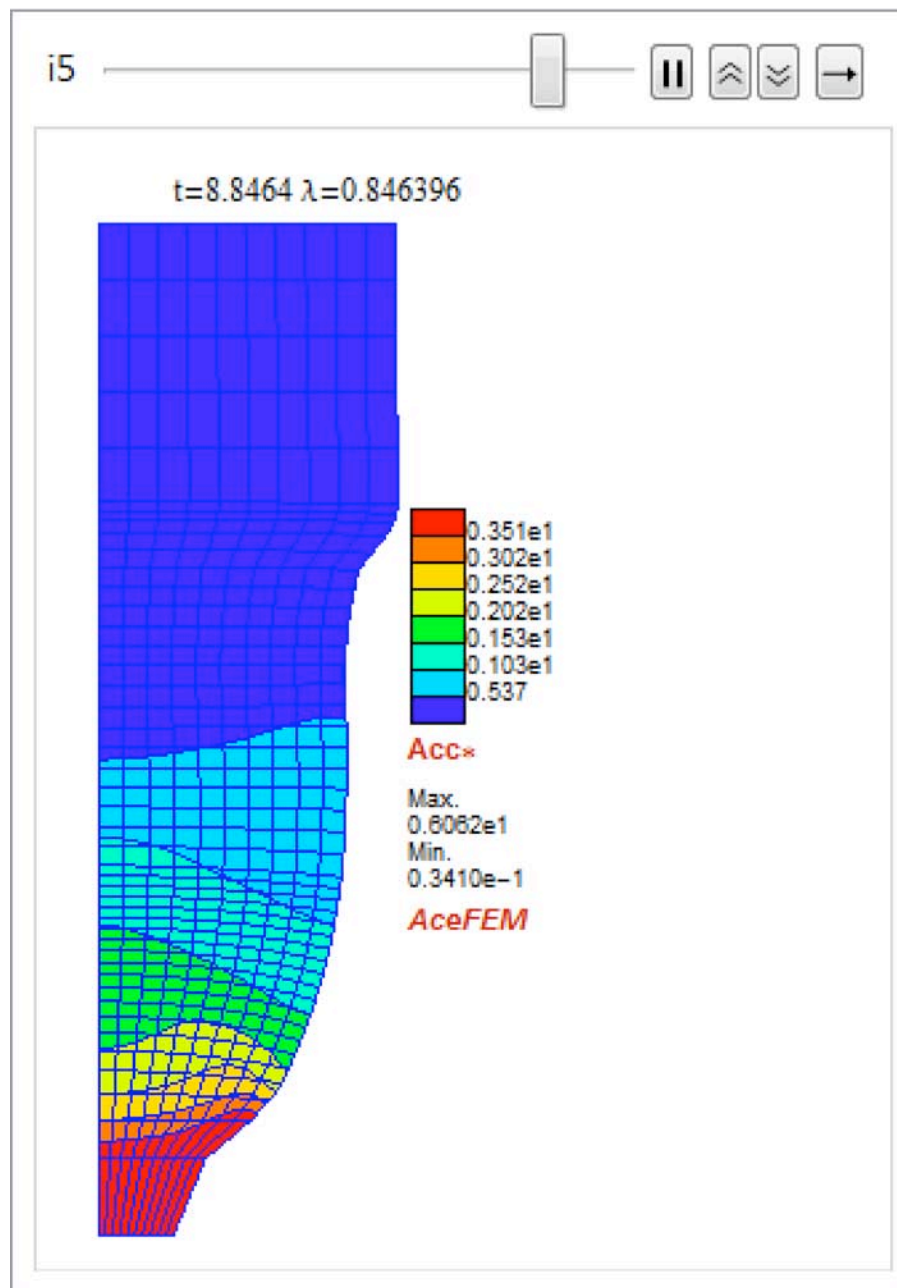


Here the sequence of pictures in GIF format is generated and stored in a subdirectory of the current directory with the name pldef.

```
If[FileType["pldef"] === Directory,
  DeleteDirectory["pldef", DeleteContents -> True]];
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" -> {"Animation", "pldef"}];
) &, allkeys];
```

The sequence of pictures is here transformed into the Mathematica type animation.

```
SMTMakeAnimation["pldef"]
```



The sequence of pictures is here transformed into Flash file.

```
SMTMakeAnimation["pldef", "Flash"]
```

```
$Aborted
```

Please follow the link to the animation gallery <http://www.fgg.uni-lj.si/symech/> .

Solid, Finite Strain Element for Dynamic Analysis

See also 2D snooker simulation .

```
<< "AceGen`";
Lgh = 4;
SMSInitialize["ExamplesHypersolidDynNewmark", "Environment" -> "AceFEM"];
SMSTemplate["SMSTopology" -> "Q1", "SMSSymmetricTangent" -> True
, "SMSNoTimeStorage" -> Lgh es$$["id", "NoIntPoints"]
, "SMSGroupDataNames" ->
{"E -elastic modulus", "v -poisson ratio", "t -thickness of the element",
"bX -force per unit mass X", "bY -force per unit mass Y",
"ρ0 -density", "β -Newmark beta", "δ -Newmark delta"}
, "SMSDefaultData" -> {21 000, 0.3, 1, 0, 0, 2700, 0.25, 0.5}];
```

TestExamples

```
ElementDefinitions[] := (
E = {ξ, η, ζ} ⊢ Table[SMSReal[es$$["IntPoints", i, Ig]], {i, 3}];
XI ⊢ Table[SMSReal[nd$$[i, "X", j]], {i, SMSNoNodes}, {j, SMSNoDimensions}];
NI = 1/4 {(1 - ξ) (1 - η), (1 + ξ) (1 - η), (1 + ξ) (1 + η), (1 - ξ) (1 + η)};
X ⊢ SMSFreeze[Append[NI.XI, ζ]];
Jg = SMSD[X, E]; Jgd = Det[Jg];
uI ⊢ SMSReal[Table[nd$$[i, "at", j], {i, SMSNoNodes}, {j, SMSNoDimensions}]];
uIn ⊢ SMSReal[Table[nd$$[i, "ap", j], {i, SMSNoNodes}, {j, SMSNoDimensions}]];
pe = Flatten[uI]; u = Append[NI.uI, 0]; un = Append[NI.uIn, 0];
Dg = SMSD[u, X, "Dependency" -> {E, X, SMSInverse[Jg]}];
SMSFreeze[F, IdentityMatrix[3] + Dg, "KeepStructure" -> True];
JF = Det[F]; Cg = Transpose[F].F;
{Em, v, tξ, bX, bY, ρ0, α, δ} ⊢
SMSReal[Table[es$$["Data", i], {i, Length[SMSGroupDataNames]}]];
bb = {bX, bY, 0};
Igh ⊢ SMSInteger[(Ig - 1) Lgh];
hhgn ⊢ Table[SMSReal[ed$$["hp", Igh + i], {i, Lgh}];
vn ⊢ {hhgn[[1]], hhgn[[2]], 0}; an ⊢ {hhgn[[3]], hhgn[[4]], 0};
Δt ⊢ SMSReal[rdata$$["TimeIncrement"]];
v = δ / (α Δt) (u - un) + (1 - δ / α) vn + Δt (1 - δ / (2 α)) an;
a = 1 / (α Δt^2) (u - un - Δt vn - Δt^2 (1 / 2 - α) an);
{λ, μ} = SMSHookeToLame[Em, v];
W =  $\frac{1}{2} \lambda (\mathbf{JF} - 1)^2 + \mu \left( \frac{1}{2} (\text{Tr}[\mathbf{Cg}] - 2) - \text{Log}[\mathbf{JF}] \right)$ ;
T = ρ0 u.a;
)
```

```

SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]];
ElementDefinitions[];
SMSIf[JF < 10-9, SMSEExport[2, idata$$["ErrorStatus"]]];
SMSEExport[{v[[1]], v[[2]], a[[1]], a[[2]]}, Table[ed$$["ht", Igh + i], {i, 4}]];
wgp = SMSReal[es$$["IntPoints", 4, Ig]];
SMSDo[
  Rg = Jgd tζ wgp SMSD[W + T - ρ0 u.bb, pe, i, "Constant" → a];
  SMSEExport[SMSResidualSign Rg, p$$[i], "AddIn" → True];
  SMSDo[
    Kg = SMSD[Rg, pe, j];
    SMSEExport[Kg, s$$[i, j], "AddIn" → True];
    , {j, i, 8}];
    , {i, 1, 8}];
  SMSEndDo[]];

SMSStandardModule["Postprocessing"];
SMSDo[Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]];
ElementDefinitions[];
SMSNPostNames = {"DeformedMeshX", "DeformedMeshY", "u", "v"};
SMSEExport[Table[Join[uI[[i]], uI[[i]]], {i, SMSNoNodes}], npost$$];
Eg = 1 / 2 (Cg - IdentityMatrix[3]);
σ = (1 / JF) * SMSD[W, F, "IgnoreNumbers" -> True] . Transpose[F];
SMSGPostNames = {"Exx", "Eyy", "Exy", "Sxx", "Syy", "Sxy", "Szz"};
SMSEExport[Join[Extract[Eg, {{1, 1}, {2, 2}, {1, 2}}],
  Extract[σ, {{1, 1}, {2, 2}, {1, 2}, {3, 3}}]], gpost$$[Ig, #1] &];
SMSEndDo[]];

SMSWrite[]];

```

File:	ExamplesHypersolidDynNewmark.c	Size: 12 965
Methods	No.Formulae	No.Leafs
SKR	125	2019
SPP	68	967

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 .

```
<< AceGen` ;
```

This will create a C source file "Energy.c" with the following contents

```

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

```

Export["Energy.h",
  "void Energy(double *ICp, double *IICp,double *IIICp, double
    c[5], double *W,double dW[3], double ddW[3][3]);","Text"];

```

Subroutine Energy calculates the strain energy $W(IC,IIC,IIIC)$ where IC and IIC and IIIC are invariants of the right Cauchy-Green tensor \mathbf{C} and first and second derivative of the strain energy with respect to the input parameters IC and IIC and IIIC. 2D, plain strain, izoparametric, quadrilateral element is generated (for details see Simple 2D Solid, Finite Strain Element).

Two solution will be presented:

■ Solution 1: Generated source and user supplied "Energy.c" are compiled separately and then linked together to produce elements dll file

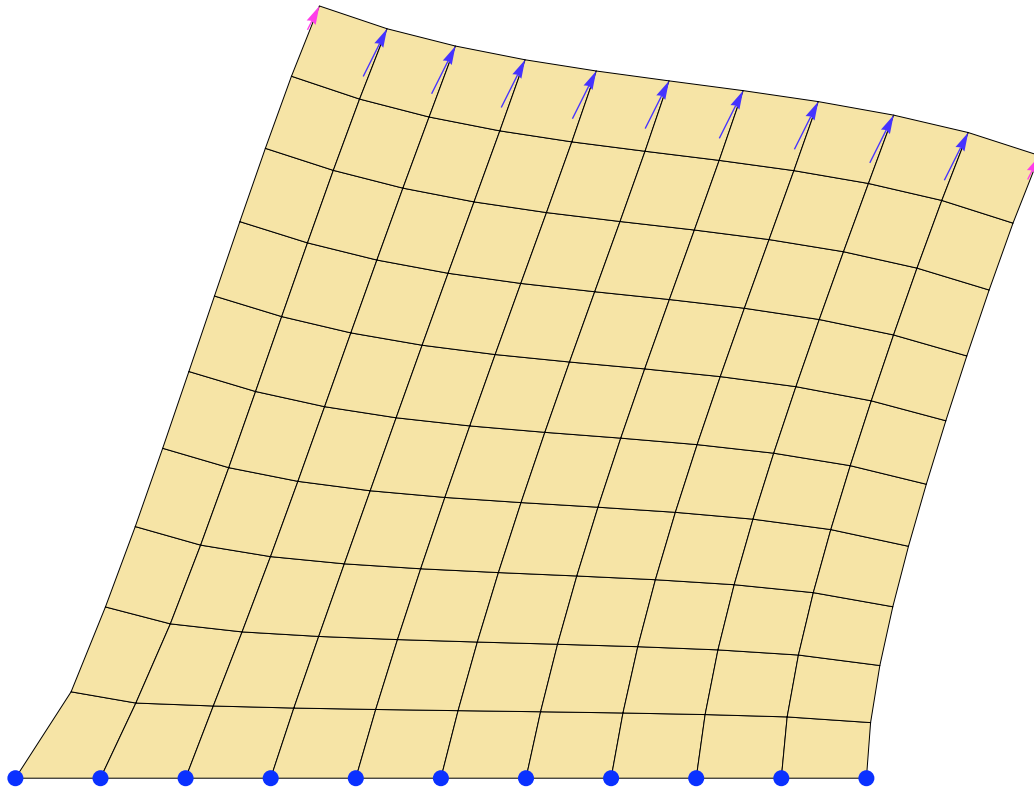
```
<< AceGen` ;
SMSInitialize["ExamplesUserSub1", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "Q1", "SMSSymmetricTangent" → True
, "SMSGroupDataNames" → {"c1 -constant 1", "c2 -constant 2", "c3 -constant 3",
" c4 -constant 4", "c5 -constant 5"}
, "SMSDefaultData" → {600, 300, 100, 50, 10}
];
SMSStandardModule["Tangent and residual"];
SMSDo[Ig, 1, SMSInteger[es$$["id", "NoIntPoints"]]];
E = {ξ, η, ζ} ⊢ Table[SMSReal[es$$["IntPoints", i, Ig]], {i, 3}];
XI ⊢ Table[SMSReal[nd$$[i, "X", j]], {i, SMSNoNodes}, {j, SMSNoDimensions}];
NI = 1/4 {(1 - ξ) (1 - η), (1 + ξ) (1 - η), (1 + ξ) (1 + η), (1 - ξ) (1 + η)};
X ⊢ SMSFreeze[Append[NI.XI, ζ]];
Jg = SMSD[X, E]; Jgd = Det[Jg];
uI ⊢ SMSReal[Table[nd$$[i, "at", j], {i, SMSNoNodes}, {j, SMSNoDimensions}]];
pe = Flatten[uI]; u = Append[NI.uI, 0];
Dg = SMSD[u, X, "Dependency" → {E, X, SMSInverse[Jg]}];
F = IdentityMatrix[3] + Dg; JF = Det[F]; Cg = Transpose[F].F;
{IC, IIC, IIIC} ⊢ SMSFreeze[{Tr[Cg], 0, Det[Cg]}];
c = SMSReal[Table[es$$["Data", i], {i, Length[SMSGroupDataNames]}]];
WCall = SMSCall["Energy", IC, IIC, IIIC, c,
Real[W$$], 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$$, "Subordinate" → WCall,
"Dependency" → Transpose[{{IC, IIC, IIIC}, δW}]];
wgp ⊢ SMSReal[es$$["IntPoints", 4, Ig]];
Rg = Jgd wgp SMSD[W, pe];
SMSExport[SMSResidualSign Rg, p$$, "AddIn" → True];
Kg = SMSD[Rg, pe];
SMSExport[Kg, s$$, "AddIn" → True];
SMSEndDo[];
SMSWrite["IncludeHeaders" → {"Energy.h"}];
```

File:	ExamplesUserSub1.c	Size:	14 281
Methods	No. Formulae	No. Leafs	
SKR	229	4129	

```
SMTMakeDll["ExamplesUserSub1", "AdditionalSourceFiles" → {"Energy.c"}];
```

```
<< AceFEM` ;
```

```
SMTInputData[];
SMTAddDomain[{"1", "ExamplesUserSub1", {}}];
SMTMesh["1", "Q1", {10, 10}, {{{0, 0}, {3, 0}}, {{0, 2}, {3, 2}}];
SMTAddNaturalBoundary[Line[{{0, 2}, {3, 2}}], 1 -> Line[{1}], 2 -> Line[{2}]];
SMTAddEssentialBoundary["Y" == 0 &, 1 -> 0, 2 -> 0];
SMTAnalysis[];
SMTNextStep[100, 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**

```

<< AceGen` ;
SMSInitialize["ExamplesUserSub2", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "Q1", "SMSSymmetricTangent" → True
, "SMSGroupDataNames" → {"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, SMSInteger[es$$["id", "NoIntPoints"]]];
E = {ξ, η, ζ} ⊢ Table[SMSReal[es$$["IntPoints", i, Ig]], {i, 3}];
XI ⊢ Table[SMSReal[nd$$[i, "X", j]], {i, SMSNoNodes}, {j, SMSNoDimensions}];
NI = 1/4 {(1 - ξ) (1 - η), (1 + ξ) (1 - η), (1 + ξ) (1 + η), (1 - ξ) (1 + η)};
X ⊢ SMSFreeze[Append[NI.XI, ζ]];
Jg ⊢ SMSD[X, E]; Jgd ⊢ Det[Jg];
uI ⊢ SMSReal[Table[nd$$[i, "at", j], {i, SMSNoNodes}, {j, SMSNoDimensions}]];
pe = Flatten[uI]; u ⊢ Append[NI.uI, 0];
Dg ⊢ SMSD[u, X, "Dependency" → {E, X, SMSInverse[Jg]}];
F ⊢ IdentityMatrix[3] + Dg; JF ⊢ Det[F]; Cg ⊢ Transpose[F].F;
{IC, IIC, IIIC} ⊢ SMSFreeze[{Tr[Cg], 0, Det[Cg]}];
c ⊢ SMSReal[Table[es$$["Data", i], {i, Length[SMSGroupDataNames]}]];
WCall ⊢ SMSCall["Energy", IC, IIC, IIIC, c,
Real[W$$], 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$$, "Subordinate" → WCall,
"Dependency" → Transpose[{{IC, IIC, IIIC}, δW}]];
wgp ⊢ SMSReal[es$$["IntPoints", 4, Ig]];
Rg ⊢ Jgd wgp SMSD[W, pe];
SMSExport[SMSResidualSign Rg, p$$, "AddIn" → True];
Kg ⊢ SMSD[Rg, pe];
SMSExport[Kg, s$$, "AddIn" → True];
SMSEndDo[];
SMSWrite["IncludeHeaders" → {"Energy.h"}, "Splice" → {"Energy.c"}];

```

File:	ExamplesUserSub2.c	Size:	14 705
Methods	No. Formulae	No. Leafs	
SKR	229	4129	

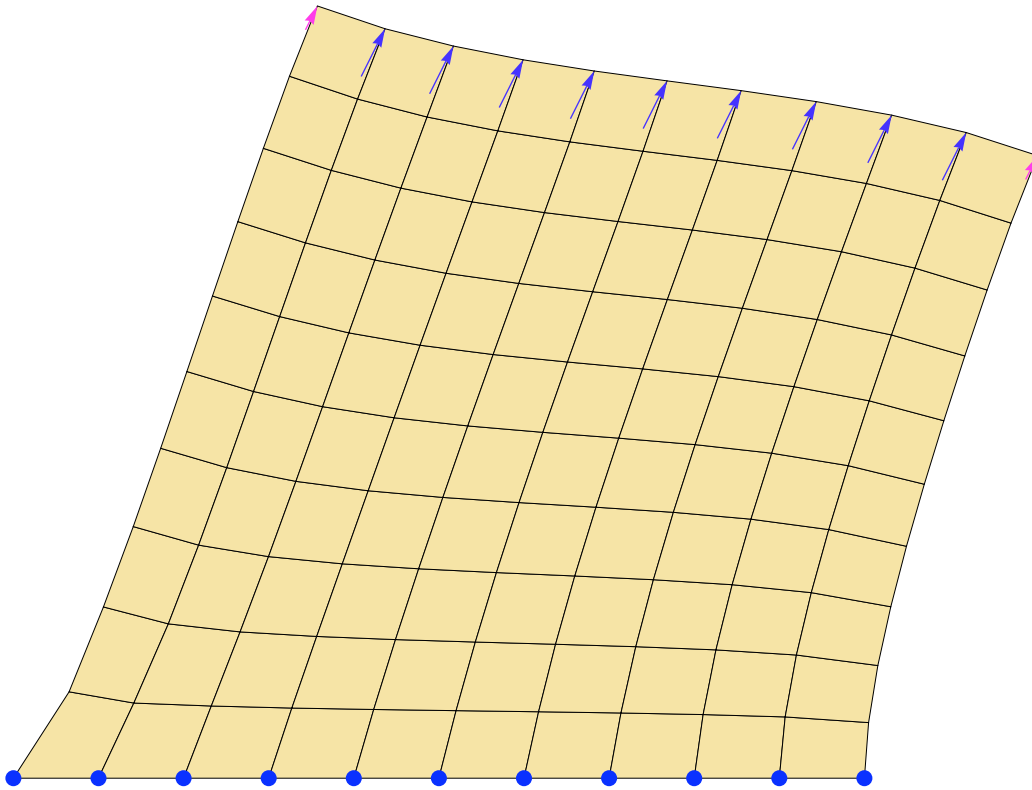
```

<< AceFEM` ;

```



```
SMTInputData[];
SMTAddDomain[{"1", "ExamplesUserSub2", {}}];
SMTMesh["1", "Q1", {10, 10}, {{{0, 0}, {3, 0}}, {{0, 2}, {3, 2}}]];
SMTAddNaturalBoundary[Line[{{0, 2}, {3, 2}}], 1 -> Line[{1}], 2 -> Line[{2}]];
SMTAddEssentialBoundary["Y" == 0 &, 1 -> 0, 2 -> 0];
SMTAnalysis[];
SMTNextStep[100, 100];
While[SMTConvergence[10^-12, 10], SMTNewtonIteration[]];
SMTShowMesh["BoundaryConditions" -> True, "DeformedMesh" -> True]
```



Examples of Contact Formulations

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 .

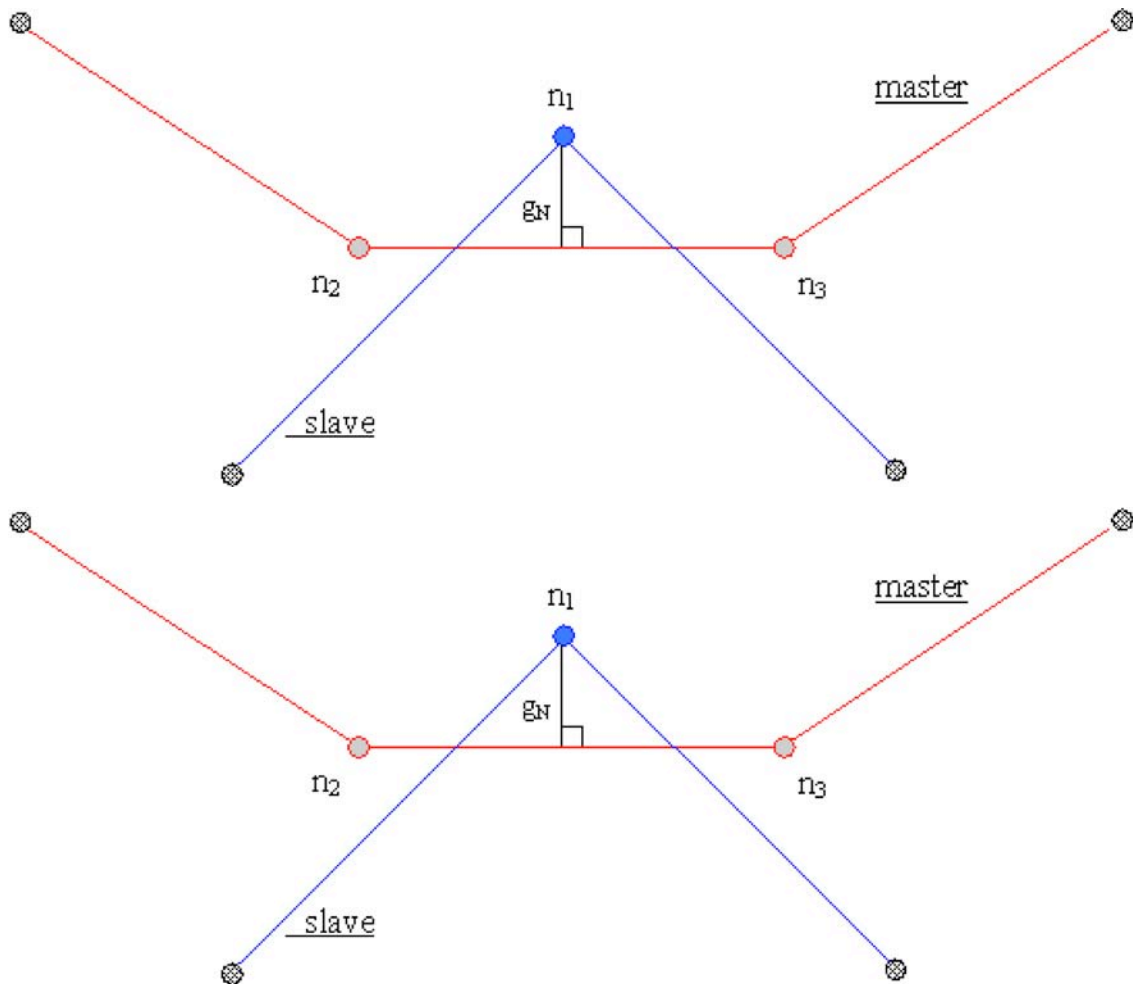
Description

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:

$$\Pi = \begin{cases} \sum_{i=1}^N \frac{1}{2} \rho g_N^2 & \text{for } g_N \leq 0 \\ 0 & \text{for } g_N > 0 \end{cases}$$

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

```

<< AceGen`
SMSInitialize["ExamplesCTD2N1L1Pen", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "LX", "SMSNoDimensions" → 2, "SMSNoNodes" → 3,
  "SMSSymmetricTangent" → True, "SMSDOFGlobal" → {2, 2, 2}, "SMSSegments" → {},
  "SMSAdditionalNodes" → "{Null,Null}&", "SMSNodeID" → {"D", "D -D", "D -D"},
  "SMSCharSwitch" → {"CTD2N1", "CTD2L1", "ContactElement"}];

SMSStandardModule["Nodal information"];
Xi = Array[SMSReal[nd$$[#1, "X", 1]] &, 1];
Yi = Array[SMSReal[nd$$[#1, "X", 2]] &, 1];
ui = Array[SMSReal[nd$$[#1, "at", 1]] &, 1];
vi = Array[SMSReal[nd$$[#1, "at", 2]] &, 1];
uiP = Array[SMSReal[nd$$[#1, "ap", 1]] &, 1];
viP = Array[SMSReal[nd$$[#1, "ap", 2]] &, 1];
x = {Xi + ui, Yi + vi};
xP = {Xi + uiP, Yi + viP};
SMSExport[{Flatten[{x, xP}], d$$}];

SMSStandardModule["Tangent and residual"];
{Xs, X1, X2} = Array[SMSReal[nd$$[#1, "X", #2]] &, {SMSNoNodes, SMSNoDimensions}];
{us, u1, u2} = Array[SMSReal[nd$$[#1, "at", #2]] &, {SMSNoNodes, SMSNoDimensions}];
u = Flatten[{us, u1, u2}];
SMSGroupDataNames = {"ρ -penalty parameter"};
SMSDefaultData = {1000};
{ρ} = Array[SMSReal[es$$["Data", #1]] &, 1];
(* check if contact is not detected by the global search *)
SMSIf[SMSInteger[ns$$[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 = 
$$\frac{t}{\sqrt{t.t}}$$
;
  eN = {{0, 0, 1} × {eT[[1]], eT[[2]], 0}} [[{1, 2}]];
  ξ = 
$$\frac{(xs - x1).eT}{l}$$
;
  xξ = x1 + (x2 - x1) ξ;
  r = -xs + xξ;
  gN = r.eN;
  SMSIf[gN > 0];
  SMSReturn[];
SMSEndIf[];
  Π = 
$$\frac{1}{2} \rho gN^2$$
;
  SMSDo[i, 1, 6];
  Ri = SMSD[Π, u, i];
  SMSExport[SMSResidualSign Ri, p$$[i]];
  SMSDo[j, i, 6];
  Kij = SMSD[Ri, u, j];
  SMSExport[Kij, s$$[i, j]];

```

```

SMSEndDo[];
SMSEndDo[];
SMSEndIf[];
SMSWrite[];

```

Postprocessing of the integration point quantities is suspended for the element.

See also: [SMSReferenceNodes](#)

Default value for the SMSSegmentsTriangulation constant is not available for the given topology :

LX See also: [Template Constants](#)

Default integration code is set to

21 . See also: [SMSDefaultIntegrationCode](#)

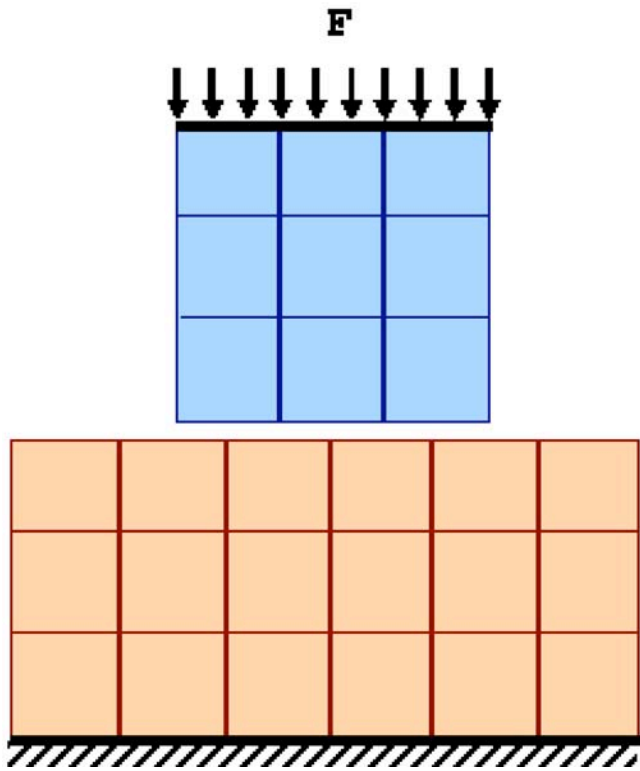
File:	ExamplesCTD2N1L1Pen.c	Size:	6677
Methods	No.Formulae	No.Leafs	
PAN	3	60	
SKR	63	894	

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, line master segment element section and hypersolid element from Solid, Finite Strain Element for Direct and Sensitivity Analysis section to analyse the simple 2D indentation problem: small elastic box pressed down into large elastic box.

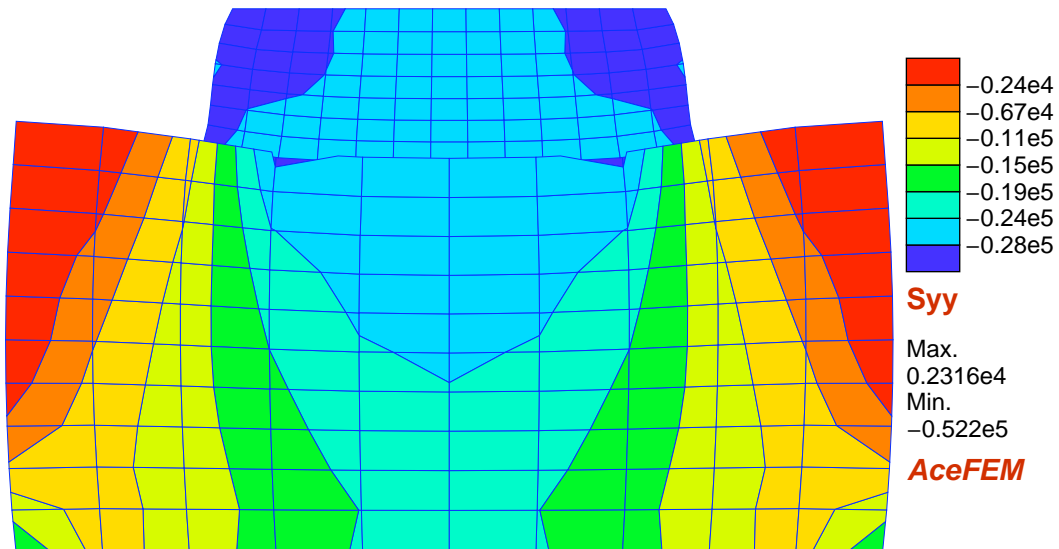


Solution

```

<< AceFEM` ;
SMTInputData[];
SMTAddDomain[
  {"Solid1", "ExamplesHypersolid2D", {"E *" -> 70 000, "ν *" -> 0.3}},
  {"Solid2", "ExamplesHypersolid2D", {"E *" -> 210 000, "ν *" -> 0.3}},
  {"Contact", "ExamplesCTD2N1L1Pen", {"ρ *" -> 200 000}}];
SMTMesh["Solid1",
  "Q1", {13, 7}, {{{-1/2, 0.1}, {1/2, 0.1}}, {{-1/2, 0.6}, {1/2, 0.6}}},
  "BodyID" -> "B1", "BoundaryDomainID" -> "Contact"];
SMTMesh["Solid2",
  "Q1", {10, 10}, {{{-1, -1}, {1, -1}}, {{-1, 0}, {1, 0}}},
  "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[0, 0.1];
While[
  While[step = SMTConvergence[10^-8, 10, {"Adaptive", 7, 0.0001, 0.2, 0.35}],
    SMTNewtonIteration[]];
  If[step[[4]] === "MinBound", SMTStatusReport["Δλ < Δλmin"]];
  step[[3]],
  If[step[[1]], SMTStepBack[], SMTShowMesh[
    "DeformedMesh" -> True, "Marks" -> False, "Domains" -> {"Solid1", "Solid2"},
    "Show" -> "Window", "Field" -> "Syy", "Contour" -> True];];
  SMTNextStep[1, 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):

$$\Pi = \Pi_N + \Pi_T$$

where:

$$\Pi_N = \begin{cases} \frac{1}{2\rho} (\langle g_N + \lambda_N \rangle_-^2 - \lambda_N^2) & \text{for } g_N \leq 0 \\ 0 & \text{for } g_N > 0 \end{cases}$$

and

$$\Pi_T = \begin{cases} \frac{1}{2\rho} (\langle g_T + \lambda_T \rangle_-^2 - \lambda_T^2) & \text{for } g_T \leq 0 \\ 0 & \text{for } g_T > 0 \end{cases}$$

and

$$\langle x \rangle_- = \begin{cases} x & \text{for } x \leq 0 \\ 0 & \text{for } x > 0 \end{cases}$$

and ρ is regularisation parameter, g_N is normal gap (at the point of orthogonal projection of slave point on master boundary), g_T is tangential slip.

λ_N and λ_T are lagrange multipliers and have a meaning of normal and tangential nominal pressure respectively.

The following user subroutines have to be generated:

- ⇒ user subroutine for specification of nodal positions,
- ⇒ user subroutine for the direct implicit analysis,

Solution

```

<< AceGen`;
driver = 2;
SMSInitialize["ExamplesCTD2N1DN2L1DN1AL",
  "Environment" → "AceFEM", "VectorLength" → 5000];
SMSTemplate[
  "SMSTopology" → "LX",
  "SMSNoDimensions" → 2,
  "SMSNoNodes" → 8,
  "SMSDOFGlobal" → {2, 2, 2, 2, 2, 2, 2, 2},
  "SMSSegments" → {},
  "SMSAdditionalNodes" → Hold[{Null, Null, Null, Null, Null, Null, #1} &],
  "SMSNodeID" → {"D", "D -D", "D -D", "D -D", "D -D", "D -D", "D -D", "Lagr -AL -S"},
  "SMSCharSwitch" → {"CTD2N1DN2", "CTD2L1DN1", "ContactElement"},
  "SMSNoTimeStorage" → 2 (2 + idata$$["NoSensParameters"]),
  "SMSSymmetricTangent" → False,
  "SMSDefaultIntegrationCode" → 0];
(* ===== *)
SMSStandardModule["Nodal information"];
x = Array[SMSReal[nd$$[1, "X", #]] &, 2] + Array[SMSReal[nd$$[1, "at", #]] &, 2];
xP = Array[SMSReal[nd$$[1, "X", #]] &, 2] + Array[SMSReal[nd$$[1, "ap", #]] &, 2];
SMSExport[{Flatten[{x, xP}], d$$];
(* ===== *)
SMSStandardModule["Tangent and residual"];
(*Element data*)
SMSGroupDataNames =
  {"μ -friction coefficient", "ρ -regularization parameter", "t -thickness"};
SMSDefaultData = {0, 1, 1};
{μ, ρ, thick} = Array[SMSReal[es$$["Data", #]] &, 3];

(*Read and set element values*)
Xiaug = Array[SMSReal[nd$$[#, "X", 1]] &, 8];
Yiaug = Array[SMSReal[nd$$[#, "X", 2]] &, 8];
uiaug = Array[SMSReal[nd$$[#, "at", 1]] &, 8];
viaug = Array[SMSReal[nd$$[#, "at", 2]] &, 8];
uiaugP = Array[SMSReal[nd$$[#, "ap", 1]] &, 8];
viaugP = Array[SMSReal[nd$$[#, "ap", 2]] &, 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 = {1, 2, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16};
{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];
SMSExport[SMSResidualSign RNC, p$$[ii]];
SMSDo[j, 1, 2];
  jj = SMSInteger[SMSPart[index, 10 + j]];
  dRdaNC = SMSD[ RNC, {λN, λT}, j];
  SMSExport[dRdaNC, s$$[ii, jj]];
SMSEndDo[];
SMSEndDo[];
SMSExport[{0, 0}, Array[ed$$["ht", #] &, 2]];
SMSElse[];
{ξP, gNP} = Array[SMSReal[ed$$["hp", #]] &, 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[  $\frac{1}{8}$  Binomial[m - 1, i - 1] (1 + ξ)i-1 (1 - ξ)m-i, {i, m} ] ];
ξA = SMSReal[-1];
fBiA = Module[ {m = 4}, Table[  $\frac{1}{8}$  Binomial[m - 1, i - 1] (1 + ξA)i-1 (1 - ξA)m-i, {i, m} ] ];
ξB = SMSReal[1];
fBiB = Module[ {m = 4}, Table[  $\frac{1}{8}$  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[SMSInteger[idata$$["Iteration"]] == 1];
  {stateN, stateT} = Array[SMSInteger[ed$$["hp", # + 2]] &, 2];
  SMSElse[];
  {stateN, stateT} = SMSInteger[{0, 0}];
  SMSEndIf[stateN, stateT];

  SMSIf[λNaug ≤ 10-8 || stateN == 1];
  lagrN = λNaug;
  SMSExport[1, ed$$["ht", 3]];
  SMSElse[];
  lagrN = 0;
  SMSExport[0, ed$$["ht", 3]];
  SMSEndIf[lagrN];

  kaug = -μ lagrN;

  SMSIf[SMSAbs[λTaug] - kaug ≥ -10-8 || stateT == 1];
  λTaugkaug = SMSAbs[λTaug] - kaug;
  SMSExport[1, ed$$["ht", 4]];
  SMSElse[];
  λTaugkaug = 0;
  SMSExport[0, ed$$["ht", 4]];
  SMSEndIf[λTaugkaug];

  (* Augmented Lagrangian of contact *)
  LagrN = Area0 (lagrN2 - λN2) / (2 ρ);
  LagrT = Area0 (λT Δξ + ρ Δξ2 / 2 - (λTaugkaug2) / (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]];
  SMSExport[SMSResidualSign Ri, p$$[ii]];
  SMSDo[j, 1, 12];
  dRidaj = SMSD[Ri, basicVars, j];
  jj = SMSInteger[SMSPart[index, j]];
  SMSExport[dRidaj, s$$[ii, jj]];
  SMSEndDo[];
  SMSEndDo[];

```

```
SMSExport[{ξ, gN}, Array[ed$$["ht", #] &, 2] ];
SMSEndIf[];
```

```
SMSWrite[];
```

Postprocessing of the integration point quantities is suspended for the element.

See also: `SMSReferenceNodes`

Default value for the `SMSSegmentsTriangulation` constant is not available for the given topology :

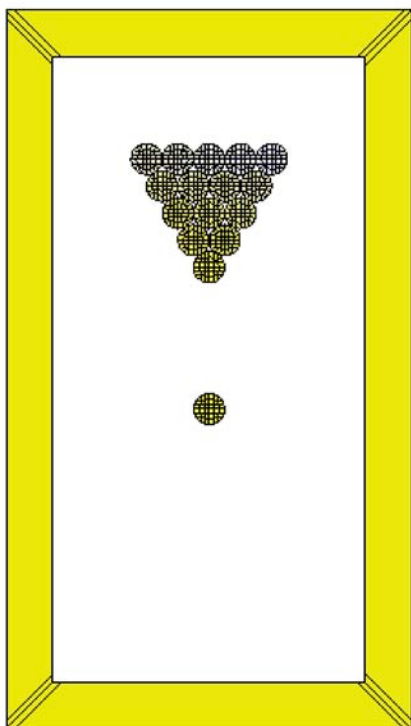
LX See also: `Template Constants`

File:	ExamplesCTD2N1DN2L1DN1AL.c	Size:	57 485
Methods	No. Formulae	No. Leafs	
PAN	1	62	
SKR	1503	16 568	

2D snooker simulation

■ Description

Use the contact element from 2D slave node, smooth master segment element and hypersolid dynamic element from Solid, Finite Strain Element for Dynamic Analysis and analyze the single snooker shoot.



■ Analysis

```

<< AceFEM` ;
δ = 0.60; (* 0.5 for non-dissipative model *)
β =  $\left(\frac{\delta + 0.5}{2}\right)^2$ ;
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_] :=
  SMTMesh[domain, "Q1", {Nb, Nb},
    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}], "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 *" -> 50 000., "ν *" -> 0.3, "ρ0 *" -> 2, "β *" -> β, "δ *" -> δ}]
  , {i, BN + 1}];
SMTMesh["b", "Q1", {12, 1},
  {{{-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}}},
  "BodyID" → "C", "BoundaryDomainID" -> "c", "InterpolationOrder" -> 1];
ball[0, - $\frac{by}{8}$ , 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,  $\frac{by}{2} + \left(i - \frac{(TN + 1)}{2}\right) R \sqrt{3}$ ,
    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  $\left(1 - \cos\left[\frac{\pi}{4 Nb}\right]\right)$ ];
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[0.0025, 0.01];
While[
  While[step = SMTConvergence[10^-8, 15, {"Adaptive Time", 8, .0000001, 0.05, 5},
    "Alternate" → True], SMTNewtonIteration[]];
  If[step[[4]] == "MinBound", SMTStatusReport[" $\Delta\lambda < \Delta\lambda_{min}$ "]];
  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[step[[2]], 0.01]
];

```

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 .

```

<< "AceGen`";
SMSInitialize["ExamplesCTD3V1P1", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "PX", "SMSNoNodes" → 4,
  "SMSSymmetricTangent" → True, "SMSDOFGlobal" → {3, 3, 3, 3},
  "SMSSegments" → {}, "SMSAdditionalNodes" → {"Null,Null,Null}&",
  "SMSNodeID" → {"D", "D -D", "D -D", "D -D"},
  "SMSCharSwitch" → {"CTD3V1", "CTD3P1", "ContactElement"}];

SMSStandardModule["Nodal information"];
x = Array[SMSReal[nd$$[1, "X", #]] &, 3] + Array[SMSReal[nd$$[1, "at", #]] &, 3];
xP = Array[SMSReal[nd$$[1, "X", #]] &, 3] + Array[SMSReal[nd$$[1, "ap", #]] &, 3];
SMSExport[{Flatten[{x, xP}], d$$];

SMSStandardModule["Tangent and residual"];
SMSGroupDataNames = {" $\rho$  -penalty parameter"};
SMSDefaultData = {1000};
{ $\rho$ } = Array[SMSReal[es$$["Data", #]] &, 1];

(*Read and set element values*)
Xi = Array[SMSReal[nd$$[#, "X", 1]] &, 4];
Yi = Array[SMSReal[nd$$[#, "X", 2]] &, 4];
Wi = Array[SMSReal[nd$$[#, "X", 3]] &, 4];
ui = Array[SMSReal[nd$$[#, "at", 1]] &, 4];
vi = Array[SMSReal[nd$$[#, "at", 2]] &, 4];
wi = Array[SMSReal[nd$$[#, "at", 3]] &, 4];
basicVars = ({ui, vi, wi} // Transpose // Flatten);
{xS, x1, x2, x3} = Transpose[{Xi + ui, Yi + vi, Wi + wi}];
(* If contact possible *)
SMSIf[SMSInteger[ns$$[2, "id", "Dummy"]] ≠ 1];
(* projection point and gap distance *)
localSearch := (
  xP = { $\xi$ ,  $\eta$ , 1 -  $\xi$  -  $\eta$ }.{x1, x2, x3};
   $\tau\xi$  = SMSD[xP,  $\xi$ ];
   $\tau\eta$  = SMSD[xP,  $\eta$ ];

```

```

    rn = Cross[ $\tau\xi$ ,  $\tau\eta$ ];
    n = rn / SMSqrt[rn.rn];
    H = xP + gN n - xS;
    dHdb = SMSD[H, b];
    dHdbInv = SMSInverse[dHdb]
  );

b = SMSReal[{0, 0, 0}];
SMSDo[iter, 1, 30, 1, b];
  { $\xi$ ,  $\eta$ , gN} = b;
  localSearch;
   $\Delta b$  = -dHdbInv.H;
  b = b +  $\Delta b$ ;
  SMSIf[Sqrt[ $\Delta b$ . $\Delta b$ ] < 1 / 10^12 || iter == 29];
    SMSBreak[];
  SMSEndIf[];
SMSEndDo[b];

b = SMSReal[b];
(* Remember to define "b" before using this tag! *)
{ $\xi$ ,  $\eta$ , gN} = b;
localSearch;
dHda = SMSD[H, basicVars];
dbda = -dHdbInv.dHda;
SMSDefineDerivative[b, basicVars, dbda];
SMSIf[gN ≤ 0];
   $\Pi = \frac{1}{2} \rho gN^2$ ;
SMSElse[];
   $\Pi = 0$ ;
SMSEndIf[ $\Pi$ ];

SMSDo[i, 1, 12];
  (* Residual *)
  Ri = SMSD[ $\Pi$ , basicVars, i];
  SMSExport[SMSResidualSign Ri, p$$[i]];
  SMSDo[j, i, 12];
    dRidaj = SMSD[Ri, basicVars, j];
    SMSExport[dRidaj, s$$[i, j]];
  SMSEndDo[];
SMSEndDo[];

SMSEndIf[];
SMSWrite[];

```

Postprocessing of the integration point quantities is suspended for the element.

See also: `SMSReferenceNodes`

Default value for the SMSSegmentsTriangulation constant is not available for the given topology :

PX See also: `Template Constants`

Default integration code is set to 17 . See also: `SMSDefaultIntegrationCode`

File:	ExamplesCTD3V1P1.c	Size:	19 217
Methods	No. Formulae	No. Leafs	
PAN	1	92	
SKR	265	4550	

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 .

```
<< "AceGen`";
SMSInitialize["ExamplesCTD3V1S1", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "SX", "SMSNoNodes" → 5,
  "SMSSymmetricTangent" → True, "SMSDOFGlobal" → {3, 3, 3, 3, 3},
  "SMSSegments" → {}, "SMSAdditionalNodes" → "{Null,Null,Null,Null}&",
  "SMSNodeID" → {"D", "D -D", "D -D", "D -D", "D -D"},
  "SMSCharSwitch" → {"CTD3V1", "CTD3S1", "ContactElement"}];

SMSStandardModule["Nodal information"];
x = Array[SMSReal[nd$$[1, "X", #]] &, 3] + Array[SMSReal[nd$$[1, "at", #]] &, 3];
xP = Array[SMSReal[nd$$[1, "X", #]] &, 3] + Array[SMSReal[nd$$[1, "ap", #]] &, 3];
SMSExport[{Flatten[{x, xP}]} , d$$];

SMSStandardModule["Tangent and residual"];
(*Element data*)
SMSGroupDataNames = {"ρ -penalty parameter"};
SMSDefaultData = {1000};
{ρ} = Array[SMSReal[es$$["Data", #]] &, 1];

(*Read and set element values*)
Xi = Array[SMSReal[nd$$[#, "X", 1]] &, 5];
Yi = Array[SMSReal[nd$$[#, "X", 2]] &, 5];
Wi = Array[SMSReal[nd$$[#, "X", 3]] &, 5];
ui = Array[SMSReal[nd$$[#, "at", 1]] &, 5];
vi = Array[SMSReal[nd$$[#, "at", 2]] &, 5];
wi = Array[SMSReal[nd$$[#, "at", 3]] &, 5];
basicVars = ({ui, vi, wi} // Transpose // Flatten);
{xS, x1, x2, x3, x4} = Transpose[{Xi + ui, Yi + vi, Wi + 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];
SMSExport[SMSResidualSign Ri, p$$[i]];
SMSDo[j, i, 15];
dRida[j] = SMSD[Ri, basicVars, j];
SMSExport[dRida[j], s$$[i, j]];
SMSEndDo[];
SMSEndDo[];

SMSEndIf[];

SMSWrite[];

```

Postprocessing of the integration point quantities is suspended for the element.

See also: [SMSReferenceNodes](#)

Default value for the SMSSegmentsTriangulation constant is not available for the given topology :

SX See also: [Template Constants](#)

Default integration code is set to

3 . See also: [SMSDefaultIntegrationCode](#)

File:	ExamplesCTD3V1S1.c	Size:	29 620
Methods	No. Formulae	No. Leafs	
PAN	1	92	
SKR	435	7863	

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.

```

<< "AceGen`";
SMSInitialize["ExamplesCTD3V1S1DN2", "Environment" → "AceFEM"];
SMSTemplate["SMSTopology" → "SX", "SMSNoNodes" → 13,
  "SMSDOFGlobal" → {3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3},
  "SMSSegments" → {}, "SMSAdditionalNodes" →
  "{Null,Null,Null,Null,Null,Null,Null,Null,Null,Null,Null,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"},
  "SMSCharSwitch" → {"CTD3V1", "CTD3S1DN2", "ContactElement"},
  "SMSSymmetricTangent" → True];

SMSStandardModule["Nodal information"];
x = Array[ SMSReal[nd$$[1, "X", #]] &, 3] + Array[ SMSReal[nd$$[1, "at", #]] &, 3];
xP = Array[ SMSReal[nd$$[1, "X", #]] &, 3] + Array[ SMSReal[nd$$[1, "ap", #]] &, 3];
SMSExport[{Flatten[{x, xP}]} , d$$];

SMSStandardModule["Tangent and residual"];
SMSGroupDataNames = {"ρ -penalty parameter"};
SMSDefaultData = {1000};
{ρ} = Array[ SMSReal[es$$["Data", #]] &, 1];

(*Read and set element values*)
Xi = Array[ SMSReal[nd$$[#, "X", 1]] &, 5];
Yi = Array[ SMSReal[nd$$[#, "X", 2]] &, 5];
Wi = Array[ SMSReal[nd$$[#, "X", 3]] &, 5];
ui = Array[ SMSReal[nd$$[#, "at", 1]] &, 5];
vi = Array[ SMSReal[nd$$[#, "at", 2]] &, 5];
wi = Array[ SMSReal[nd$$[#, "at", 3]] &, 5];
basicVars = ({ui, vi, wi} // Transpose // Flatten);
{xS, x1, x2, x3, x4} = Transpose[{Xi + ui, Yi + vi, Wi + 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];
SMSExport[SMSResidualSign Ri, p$$[i]];
SMSDo[j, i, 15];
dRida[j] = SMSD[Ri, basicVars, j];
SMSExport[dRida[j], s$$[i, j]];
SMSEndDo[];
SMSEndDo[];

SMSEndIf[];

SMSWrite[];

```

Postprocessing of the integration point quantities is suspended for the element.

See also: [SMSReferenceNodes](#)

Default value for the SMSSegmentsTriangulation constant is not available for the given topology :

SX See also: [Template Constants](#)

Default integration code is set to

3 . See also: [SMSDefaultIntegrationCode](#)

File:	ExamplesCTD3V1S1DN2.c	Size:	29 867
Methods	No. Formulae	No. Leafs	
PAN	1	92	
SKR	435	7863	

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.

```
<< "AceGen`";
SMSInitialize["ExamplesCTD3P1DN2P1", "Environment" → "AceFEM"];
SMSSTemplate["SMSSTopology" → "PX", "SMSNoNodes" → 18,
  "SMSDOFGlobal" → {3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3},
  "SMSSegments" → {}, "SMSAdditionalNodes" →
  "{Null,Null,Null,Null,Null,Null,Null,Null,Null,Null,Null,Null,Null,Null,Null,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"},
  "SMSCharSwitch" → {"CTD3P1DN2", "CTD3P1", "ContactElement"},
  "SMSSymmetricTangent" → True];

SMSStandardModule["Nodal information"];
For[i = 1, i ≤ 3, i++,
  x = Array[ SMSReal[nd$$[i, "X", #]] &, 3] + Array[ SMSReal[nd$$[i, "at", #]] &, 3];
  xP = Array[ SMSReal[nd$$[i, "X", #]] &, 3] + Array[ SMSReal[nd$$[i, "ap", #]] &, 3];
  SMSExport[Flatten[{x, xP}], d$$[i, #] &
];

SMSStandardModule["Tangent and residual"];
SMSGroupDataNames = {"ρ -penalty parameter"};
SMSDefaultData = {1000};
{ρ} = Array[ SMSReal[es$$["Data", #]] &, 1];

(*Read and set element values*)
Xi = Array[ SMSReal[nd$$[#, "X", 1]] &, SMSNoNodes];
Yi = Array[ SMSReal[nd$$[#, "X", 2]] &, SMSNoNodes];
Wi = Array[ SMSReal[nd$$[#, "X", 3]] &, SMSNoNodes];
ui = Array[ SMSReal[nd$$[#, "at", 1]] &, SMSNoNodes];
vi = Array[ SMSReal[nd$$[#, "at", 2]] &, SMSNoNodes];
wi = Array[ SMSReal[nd$$[#, "at", 3]] &, SMSNoNodes];
dis = Transpose[{ui, vi, wi}];
basicVars = (dis // Flatten);
allX = Transpose[{Xi + ui, Yi + vi, Wi + 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 =  $\frac{1}{2} \rho gN^2$ ;
SMSElse[];
  Π = 0;
SMSEndIf[Π];
SMSDo[i, 1, Length[localBasic]];
  (* Residual *)
  Ri = SMSD[Π, localBasic, i];
  ii = SMSInteger[SMSPart[localIndex, i]];
  SMSExport[SMSResidualSign Ri, p$$[ii], "AddIn" → True];
  SMSDo[j, i, Length[localBasic]];
    dRidaj = SMSD[Ri, localBasic, j];
    jj = SMSInteger[SMSPart[localIndex, j]];
    SMSExport[dRidaj, s$$[ii, jj], "AddIn" → True];
  SMSEndDo[];
SMSEndDo[];

SMSEndIf[]
];
SMSWrite[];

```

Postprocessing of the integration point quantities is suspended for the element.

See also: [SMSReferenceNodes](#)

Default value for the SMSSegmentsTriangulation constant is not available for the given topology :

PX See also: [Template Constants](#)


```

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 =  $\frac{1}{2} \rho gN^2$ ;
  SMSElse[];
  Π = 0;
  SMSEndIf[Π];
  SMSDo[i, 1, Length[localBasic]];
  Ri = SMSD[Π, localBasic, i];
  (* Residual *)
  ii = SMSInteger[SMSPart[localIndex, i]];
  SMSExport[SMSResidualSign Ri, p$$[ii], "AddIn" → True];
  SMSDo[j, i, Length[localBasic]];
  dRidaj = SMSD[Ri, localBasic, j];
  jj = SMSInteger[SMSPart[localIndex, j]];
  SMSExport[dRidaj, s$$[ii, jj], "AddIn" → True];
  SMSEndDo[];
  SMSEndDo[];

  SMSEndIf[];

```

```
];
```

```
SMSWrite[];
```

Postprocessing of the integration point quantities is suspended for the element.

See also: `SMSReferenceNodes`

Default value for the `SMSSegmentsTriangulation` constant is not available for the given topology :

PX See also: `Template Constants`

Default integration code is set to

17 . See also: `SMSDefaultIntegrationCode`

File:	ExamplesCTD3P1P1DN2.c	Size:	54 607
Methods	No.Formulae	No.Leafs	
<code>PAN</code>	3	276	
<code>SKR</code>	827	14 100	

3D contact analysis

The use quadrilateral 3D contact element to analyze the 3D indentation problem: small elastic box pressed down into large elastic box.

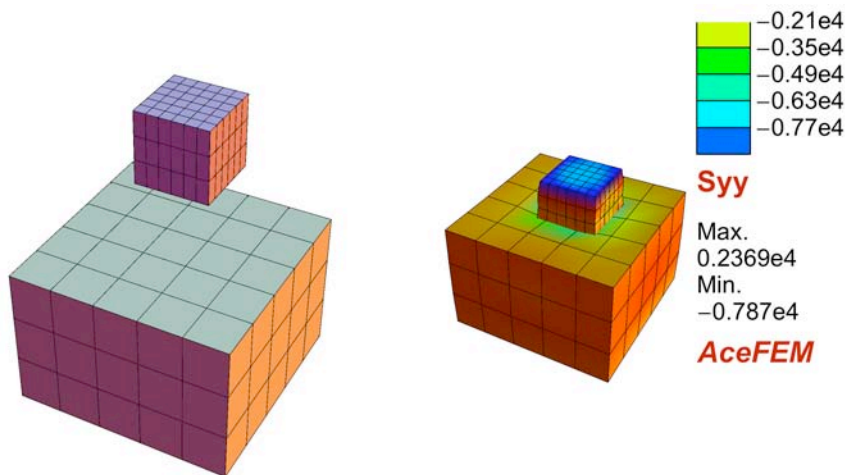
The support for the analysis of contact problems in AceFEM and implementation of contact finite elements in AceGen is described in section Implementation Notes for Contact Elements .

Simulation

```
<< AceFEM` ;
SMTInputData[];
SMTAddDomain[{"Dom1", "SED3H1DFHYH1NeoHooke", {"E *" -> 70 000., "v *" -> 0.3}},
{"Dom2", "SED3H1DFHYH1NeoHooke", {"E *" -> 70 000., "v *" -> 0.3}},
{"c", "ExamplesCTD3V1S1DN2", {"rho *" -> 10 000}}];
SMTMesh["Dom1", "H1", {6, 6, 3},
{{{0.1, 0.2, 0}, {1.1, 0.2, 0}}, {{0.1, 1.2, 0}, {1.1, 1.2, 0}}},
{{{0.1, 0.2, 1}, {1.1, 0.2, 1}}, {{0.1, 1.2, 1}, {1.1, 1.2, 1}}},
"BodyID" -> "B1", "BoundaryDomainID" -> "c"];
SMTMesh["Dom2", "H1", {5, 5, 3},
{{{ -1, -1, -3}, {2, -1, -3}}, {{ -1, 2, -3}, {2, 2, -3}}},
{{{ -1, -1, -1}, {2, -1, -1}}, {{ -1, 2, -1}, {2, 2, -1}}},
"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];

Map[{SMTNextStep[0, #]; While[SMTConvergence[], SMTNewtonIteration[];]} &,
{1, 0.1, 0.1, 0.1, 0.1}];
```

```
GraphicsRow[{SMTShowMesh[], SMTShowMesh["DeformedMesh" -> True, "Field" -> "Syy"]}]
```



Troubleshooting and New in version

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 too slow then restrict compiler optimization with SMSAnalysis["OptimizeDII" -> 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 \Rightarrow try out-of-core solution (SMTAnalyze[..., Solver -> {5, 11, {{60, 2}}}]
 - b) zero pivot, numerical factorization problem \Rightarrow try full pivoting (SMTAnalyze[..., Solver -> {5, 11}])
- Check the information given at www.fgg.uni-lj.si/symech/FAQ/.

■ 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.

New in version

- new boundary conditions input - (`SMTAddEssentialBoundary`)
- new command for generation and importing mesh - (`SMTAddMesh`)
- automatic node renumbering - (`SMTAnalysis` - "NodeReordering" option)
- new postprocessing command - `SMTPostData`
- material constants can now be specified as a list of rules - `SMTAddDomain` , Input Data
- parallel execution of K and R assembly (`SMTInputData` - "Threads")
- complete control over PARDISO solver options (`SMTSetSolver`)
- user definition of contact pairs (`SMTAnalysis`- "ContactPairs" option)
- PARDISO symmetric matrix type added (`SMTSetSolver` , `SMTAnalysis`)
- new Integer Type Environment Data (`idata$$`) data `SMTIData["ZeroPivots"]` and `SMTIData["NegativePivots"]` (replaces old `SMTIData["DiagonalSign"]` data)
- support for semi-analytical solutions (`Semi - analytical solutions`)
- support for 64-bit operating systems (`SMTInputData` - "Platform" option)
- support for user defined tasks (User Defined Tasks, `SMTTask`, Standard user subroutines)
- new boundary conditions sensitivity types (See also: `SMTSensitivity`, `SMTAddSensitivity`, Standard user subroutines, Solid, Finite Strain Element for Direct and Sensitivity Analysis, Parameter, Shape and Load Sensitivity Analysis of Multi-Domain Example .)
- parallelization of FE simulations using *Mathematica* 7.0 (see Parallel AceFEM computations)
- `Elements that Call User External Subroutines`

Advanced User Documentation

Mesh Input Data Structures

An experience user can manipulate the input data arrays directly. In that case is the users responsibility that the data provided is correct. It is advisable to use commands described in Input Data instead. This data structures are subject to change without notice! Directly changing the data after `SMTAnalysis` is forbidden. The data can be changed after the `SMTAnalysis` using the Data Base Manipulations .

input data

SMTNodes	node coordinates for all nodes
SMTElements	element nodes and domain identification for all elements
SMTDomains	description of domains
SMTEssentialBoundary	reference values of the essential boundary conditions
SMTNaturalBoundary	reference values of the natural boundary conditions
SMTInitialBoundary	initial values of boundary conditions (either essential or natural)
SMTBodies	description of bodies

Basic input data arrays.

$SMTNodes = \{node_1, node_2, \dots, node_N\}$

node_i

$\{inode, X, Y, Z\}$	3 D node with coordinates $\{X, Y, Z\}$
$\{inode, X, Y\}$	2 D node with coordinates $\{X, Y\}$
$\{inode, X\}$	1 D node with coordinates $\{X\}$

Node data for 3D, 2D and 1D problems.

$SMTElements = \{elem_1, elem_2, \dots, elem_M\}$

element_i

$\{ielem, dID, \{inode_1, inode_2, \dots\}\}$	element with the element index <i>ielem</i> , the list of nodes <i>inode₁</i> , <i>inode₂</i> ,... and domain identification <i>dID</i>
---	---

Element nodes and domain identification for all elements.

$SMTDomains = \{domain_1, domain_2, \dots, domain_K\}$

domain_i

$\{dID, etype, \{d_1, \dots, d_{ndata}\}\}$	specifies that the domain identification <i>dID</i> represents the element identified by the string <i>etype</i> and the material data <i>d_i</i> (the element source code or dll file and the appropriate integration code are selected automatically)
$\{dID, etype, \{d_1, d_2, \dots, d_{ndata}\}, opt\}$	specifies that the domain identification <i>dID</i> represents the element identified by the string <i>etype</i> , the material data <i>d_i</i> , and set of options <i>opt</i> (see table below)

<i>option</i>	<i>description</i>	<i>default</i>
"Source"	specification of the location of element' s source code (see table below)	Automatic
"IntegrationCode"	numerical integration code <i>intcode</i> (see <code>Numerical integration</code>)	<code>SMSDefaultIntegrationCode</code>
"AdditionalData"	additional data common for all the elements within a particular domain (e.g. flow curve)	{}

Options for domain input data.

Element source code location, group data, integration codes,... for all domains of the problem.

$$\text{SMTEssentialBoundary} = \{pnode_1, pnode_2, \dots, pnode_p\}$$

$pnode_i$

$$\{nodeselector, v_1, v_2, \dots, v_{N_{dof}}\}$$

v_i – reference value of the essential boundary condition (support) for the i -th unknown in all nodes that match *nodeselector* (see Selecting Nodes)

Essential boundary conditions.

$$\text{SMTNaturalBoundary} = \{nnode_1, nnode_2, \dots, nnode_R\}$$

$nnode_i$

$$\{nodeselector, f_1, f_2, \dots, f_{N_{dof}}\}$$

f_i – reference value of the natural boundary condition (force) for i -th unknown in all nodes that match *nodeselector* (see Selecting Nodes)

Natural boundary conditions.

$$\text{SMTInitialBoundary} = \{nnode_1, nnode_2, \dots, nnode_R\}$$

$nnode_i$

$$\{nodeselector, f_1, f_2, \dots, f_{N_{dof}}\}$$

f_i – initial boundary condition (force or support) for i -th unknown in all nodes that match *nodeselector* (see Selecting Nodes)

Initial boundary conditions.

$$\text{SMTBodies} = \{ \text{bodyspec}_1, \text{bodyspec}_2, \dots, \text{bodyspec}_R \}$$

$$\text{bodyspec}_i$$

$\{\{ielem_1, ielem_2, \dots\}, BID\}$	specifies that the elements with the index $\{ielem_1, ielem_2, \dots\}$ belongs to the body (topologically connected region of elements) with the body identification BID
$\{\{ielem_1, ielem_2, \dots\}, BID, \{dID_1, dID_2, \dots\}\}$	orders creation of additional elements with the domain identifications $\{dID_1, dID_2, \dots\}$ on the boundary of the body defined by the elements $\{ielem_1, ielem_2, \dots\}$
$\{\dots, dID\}$	$\equiv \{\dots, \{dID\}\}$

Definition of the bodies.

The parameter $inode$ is a node number, $ielem$ is an element number, dID is a domain identification, BID is a body identification and $inode_i$ are the indices of nodes. The domain identification dID and the body identification BID can be arbitrary strings.

Instead of the value of essential boundary condition v_i or natural boundary condition f_i we can give "FREE" string or simply empty space. The "FREE" string means that the boundary value is not prescribed for that parameter. If both essential and natural boundary condition are specified for a particular unknown then the essential boundary condition has precedence.

The parameter $nodeselector$ is used to select nodes. It has one of the forms described in section Selecting Nodes.

The SMTNodes and SMTElements input arrays are automatically generated by the SMTMesh command (see SMTMesh).

The element user subroutines can be stored in a file in the *Mathematica* or *C* language, or in a dynamic link library. Dynamic link library file is generated automatically when the problem is run with the *C* source file as input for the first time. After that the dll library is regenerated if "dll" file is older than the current *C* source file.

$$scode$$

$source_file$	file with the element source (proper extension is obligatory .m, or .c)
Automatic	for the standard elements that are part of the standard library system the driver automatically finds the source file of the element according to the element name
$executable_file$	dynamic link library with the element subroutines (name.dll)

Specification of the location of the element's source code.

Sensitivity Input Data Structures

An experience user can manipulate the input data arrays directly. In that case is the users responsibility that the data provided is correct. It is advisable to use commands described in Input Data instead. This data structures are subject to change without notice! Directly changing the data after SMTAnalysis is forbidden. The data can be changed after the SMTAnalysis using the Data Base Manipulations .

$$\text{SMTSensitivityData} = \{param_1, param_2, \dots, param_{ns}\}$$

$param_i$

 $\{sID, value, \{st_1, \dots, st_K\}, \{sti_1, \dots, sti_K\}\}$

specifies that the sensitivity parameter identification sID represents the sensitivity parameter identified by the current value $value$, the the sensitivity types (*SensType*) st_i for all domains of the problem and the indices in a type group (*SensTypeIndex*) sti_i for all domains of the problem

Sensitivity input data structure.

Reference Guide

Description of Problem

SMTInputData

SMTInputData[] erase data base from the previous AceFEM session (if any) initialize input data arrays and load the numerical module

Initialize the input data phase.

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

<i>option</i>	<i>description</i>	<i>default value</i>
"LoadSession"	" <i>session_name</i> " ⇒ the data and definitions associated with the derivation of the element " <i>session_name</i> " 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 SMSLoadSession.	False
"Platform"	"32" ⇒ 32 bit operating system (all operating systems Windows, Unix, Mac) "64" ⇒ 64 bit operating systems (Windows and Mac only)	Automatic
"NumericalModule"	specifys numerical module: "CDriver" ⇒ C language "MDriver" ⇒ <i>Mathematica</i> language	"CDriver"
"Precision"	starts the MDriver numerical module with the numerical precision set to <i>n</i> (it has no effect on CDriver)	\$MachinePrecision
"Console"	starts the CDriver module as console application and connect it with the <i>Mathematica</i> through <i>MathLink</i> protocol (it has no effect on MDriver)	False
"Threads"	sets the number of processors that are available for the parallel execution	All
"SeriesMethod"	specifys the power series expansion method (see also Semi–analytical solutions)	"Lagrange"
"SeriesData"	specifys the power series expansion parameters, the expansion point and the order of the power series expansion $\{x, x_0, n_x\}, \{y, y_0, n_y\}, \dots$ (see also Semi–analytical solutions)	False

Options for the SMTInputData function.

The *CDriver* numerical module is an executable and connected with the *Mathematica* through the *MathLink* protocol. The *CDriver* can be started in a separate window with the option "Console"->True. The option can be useful during the debugging.

```

C:\Program Files\Wolfram Research\Mathematica...
Input file           = SMTManual.nb
Output file          = NONE
Working directory    = C:\temp

Number of nodes      = 1331
Number of elements   = 1000

ELEMENT .....       = heatconduction

Solver algorithm     = unsymmetric LU
Number of equations  = 810
Profile size (bytes) = 1170144
Total memory (bytes) = 1459304
Average band width   = 180

```

SMTAddDomain

<p>SMTAddDomain[<i>dID</i>, <i>etype</i>, add domain data to the list of {<i>dcode</i>₁-><i>d</i>₁,<i>dcode</i>₂-><i>d</i>₂,...}, opt] domains with input data given as a list of rules</p>

The domain is identified by the unique string *dID* used within the session, the element code *etype* 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 `SMSGROUPDATA` constant and are specific for each element used. Input data values is given as a vector or a list of rules (*dcode*_{*i*} → *d*_{*i*}). The codes of the data can be abbreviated (e.g. "Factor" can be given as "F*"). Only those values the are not equal to the default values (see `SMSDEFAULTDATA`) have to be given.

<i>opt</i>	<i>description</i>	<i>default</i>
"Source"	specification of the location of element's source code	Automatic
"IntegrationCode"	numerical integration code <i>intcode</i> (see Numerical Integration)	see <code>SMSDEFAULTINTEGRATIONCODE</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:

```
SMTAddDomain[{"A", "solid", {"E *"->21000,"ρ *"->2700}}]
```

See also: Input Data

SMTAddMesh

<pre>SMTAddMesh[dID ,{{n1,X1,Y1,Z1},{n2,X2,Y2,Z2},...} ,{{n1^l,n2^l,...},{n1^e,n2^e,...} ...}]</pre>	<p>add nodes defined by the list of node coordinates $\{\{n_1, X_1, Y_1, Z_1\}, \{n_2, X_2, Y_2, Z_2\}, \dots\}$ and elements defined by the connectivity table $\{\{n_1^l, n_2^l, \dots\}, \{n_1^e, n_2^e, \dots\} \dots\}$ and domain identification dID to the existing mesh and return a list of global node and element numbers (n_i is the local node number and n_i^e is a local node number of the l-th node of the e-th element)</p>
<pre>SMTAddMesh["GID",mfile]</pre>	<p>import mesh from GID mesh file .msh, add nodes and elements to the existing mesh and return a list of element numbers</p>

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

Options for SMTAddMesh.

The SMTAddMesh function can be used to import a mesh generated by various existing mesh generators into AceFEM.

See also: Input Data

SMTAddElement

<pre>SMTAddElement[dID,{node1, node2, ...}]</pre>	<p>appends new element to the list of elements (element is defined by the list of nodes $node_1, node_2, \dots$ and domain identification dID where $node_i$ can be either global node number or coordinates of the node $\{X_i, Y_i, Z_i\}$)</p>
---	---

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

Options for SMTAddElement.

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

See also: Element Data , Input Data

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

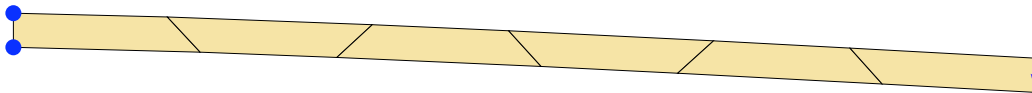
```

<< AceFEM` ;

SMTInputData[];
SMTAddDomain["A", "SEPSQ1DFLEQ1Hooke",
  {"E *" -> 11. 107, "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[];
SMTNextStep[1, 1];
While[SMTConvergence[10-12, 10], SMTNewtonIteration[]];
SMTNodeData["X" == 6 &, "at"]
SMTShowMesh["BoundaryConditions" -> True, "DeformedMesh" -> True, "Scale" -> 1000]
{{{-5.77117 × 10-6, -0.000264499}, {5.15829 × 10-6, -0.000264364}}}

```



SMTAddNode

SMTAddNode[X_i, Y_i, Z_i] appends new node to the list of nodes
 SMTAddNode[X_i, Y_i]
 SMTAddNode[X_i]

See also: Node Data , Input Data ,

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 new nodes.

SMTAddEssentialBoundary

SMTAddEssentialBoundary[<i>nodeselector</i> , <i>dof₁</i> -> <i>dB₁</i> , <i>dof₂</i> -> <i>dB₂</i> ,...]	increment or set reference value of essential (Dirichlet) boundary condition (<i>dB_i</i> is the reference value of the essential boundary condition (support) for the <i>dof_i</i> -th)
SMTAddNaturalBoundary[<i>nodeselector</i> , <i>dof₁</i> -> <i>dB₁</i> , <i>dof₂</i> -> <i>dB₂</i> ,...]	increment or set reference value of natural (Neumann) boundary condition (<i>dB_i</i> is the reference value of the natural boundary condition (force) for the <i>dof_i</i> -th unknown)
SMTAddInitialBoundary[<i>nodeselector</i> , <i>dof₁</i> -> <i>B0₁</i> , <i>dof₂</i> -> <i>B0₂</i> ,...]	increment or set initial boundary condition (<i>B0_i</i> is the initial boundary condition (essential or natural) for the <i>dof_i</i> -th unknown)

boundary condition form	description
<i>B_Number</i>	value <i>B</i> is set to all nodes that match <i>nodeselector</i>
{ <i>B₁</i> , <i>B₂</i> ,..., <i>B_N</i> }	value <i>B_i</i> is set to the <i>i</i> -th node that match <i>nodeselector</i>
Line[{ <i>b</i> }]	This form assumes that the nodes that match <i>nodeselector</i> form line or curve segment and that the constant continuous boundary condition with the intensity <i>b</i> is prescribed on the segment. The nodal values are calculated with the assumption of isoparametric interpolation of boundary between the nodes.
Line[{ <i>b₀</i> , <i>b₁</i> }]	<i>b₀</i> is intensity of continuous boundary condition in the first node that match <i>nodeselector</i> and <i>b₁</i> is intensity of continuous boundary condition in the last node that match <i>nodeselector</i> . The nodal values are calculated with the assumption of linear interpolation of intensity between <i>b₀</i> and <i>b₁</i> and an isoparametric interpolation of boundary between the nodes.

Forms of prescribed boundary conditions.

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

Options for the boundary conditions functions.

The value of boundary condition is by default incremented by the given value or set to the given value if the boundary condition is prescribed for the chosen degree of freedom for the first time. With the "Set"->True option the new value overrides all the previous definitions. With the *dof_i*->Null input all the previously defined boundary conditions are deleted for the *dof_i* degree of freedom.

The current value of the boundary condition (Bt) is defined as $B_t = B_p + \Delta\lambda dB$, where $\Delta\lambda$ is the boundary conditions multiplier increment (see Iterative solution procedure).

The *nodeselector* parameter is defined in Selecting Nodes.

Examples:

```
SMTAddEssentialBoundary["X"==5 && "Z"==2 && "ID"=="D" &, 3 -> 1.5 ]
```

The third degree of freedom in all the nodes with the coordinates X=5 and Z=2 and node identification "D" is either

incremented by 1.5 or set to 1.5 if the boundary condition is prescribed for the chosen degree of freedom for the first time.

```
SMTAddEssentialBoundary["X"==5 && "Z"==2 && "ID"=="D" &, 3 → 1.5,"Set"->True]
```

All the nodes with the coordinates $X=5$ and $Z=2$ and node identification "D" get a prescribed value 1.5 for the third degree of freedom.

```
SMTAddEssentialBoundary["X"==5 && "Z"==2 && "ID"=="D" &, 3 → Null]
```

Remove the prescribed essential boundary condition for the third degree of freedom for all the nodes with the coordinates $X=5$ and $Z=2$ and node identification "D".

See also: Selecting Nodes, Input Data , Node Data

Example: Bending of the column (path following procedure, animations, 2D solids), Boundary conditions (2D solid).

SMTAddNaturalBoundary

See also: SMTAddEssentialBoundary

Example: Bending of the column (path following procedure, animations, 2D solids), Boundary conditions (2D solid).

SMTAddInitialBoundary

See also: SMTAddEssentialBoundary

Example: Bending of the column (path following procedure, animations, 2D solids), Boundary conditions (2D solid).

SMTMesh

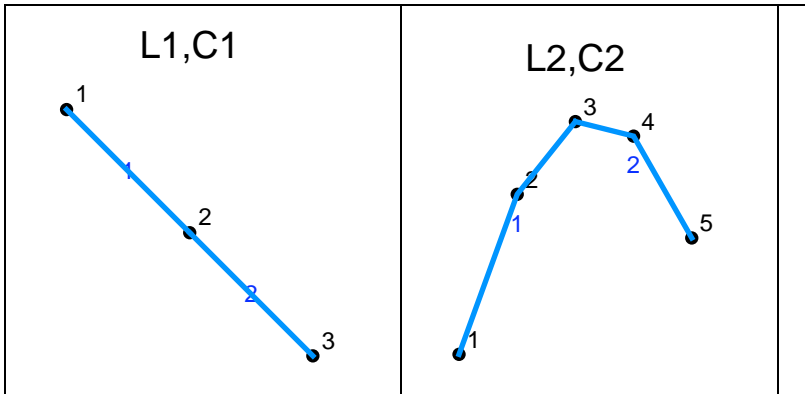
`SMTMesh[dID,topology,division,mastermesh]` construct structured mesh of elements with the domain identification *dID*, and append it to the previously defined nodes (*SMTNodes*) and elements (*SMTElements*)

<i>option</i>	<i>description</i>	<i>default value</i>
"InterpolationOrder"	The degree of master mesh 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.	3
"BodyID"	body identification string	None
"BoundaryDomainID"	domain identification (one or more) for the elements additionally generated on the outer surface of elements generated by <i>SMTMesh</i>	{}

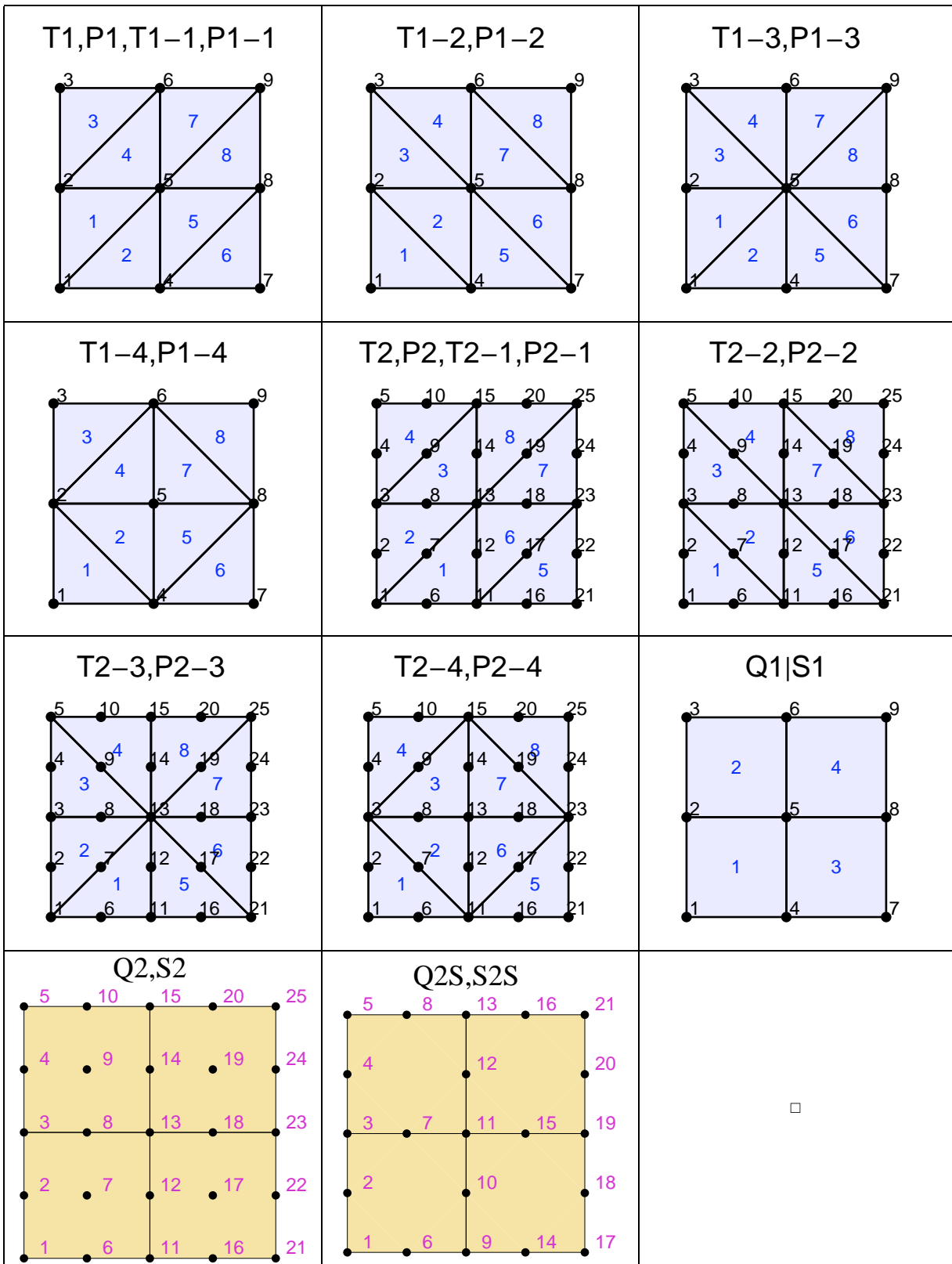
Options for SMTMesh.

The function adds new nodes and elements to the *SMTNodes* and *SMTElements* input data arrays (see Input Data). The *topology* parameter defines the type of the elements according to the types defined in section Template Constants. The function returns the range of indexes of new nodes and elements .

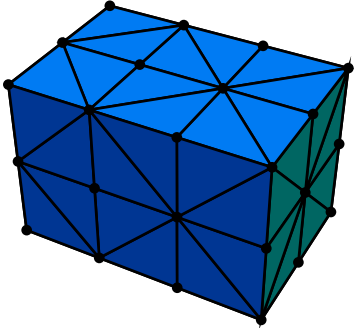
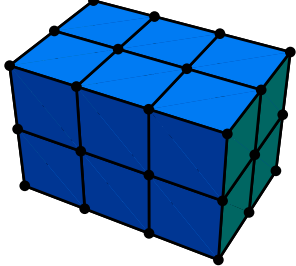
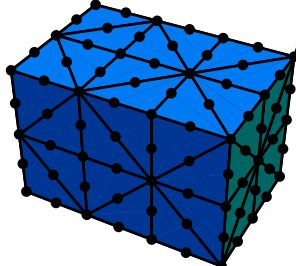
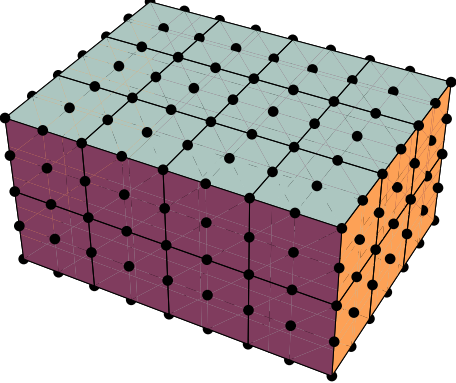
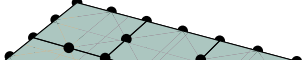
1D mesh topology

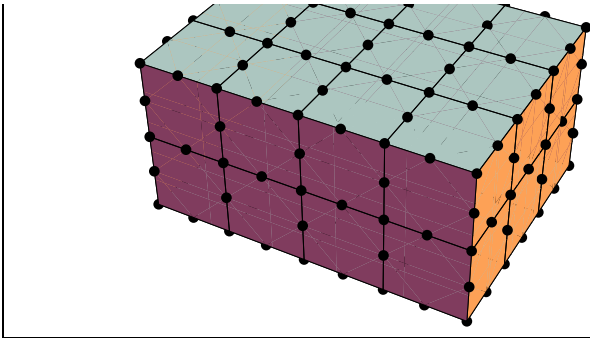


2D mesh topology



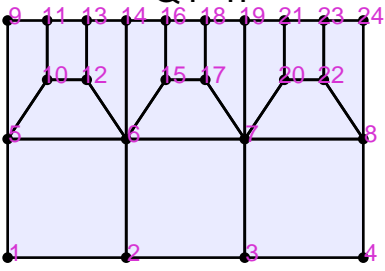
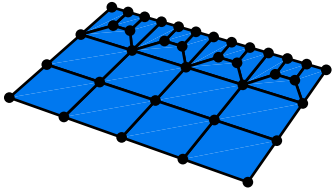
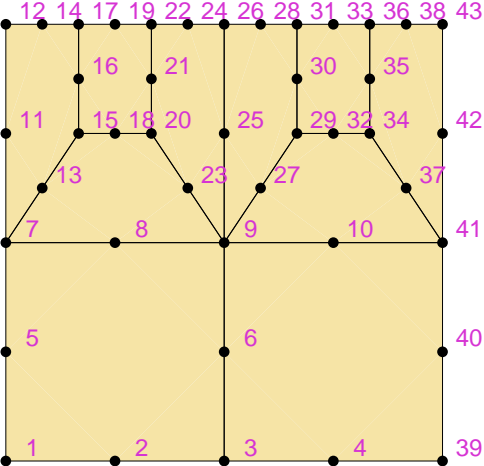
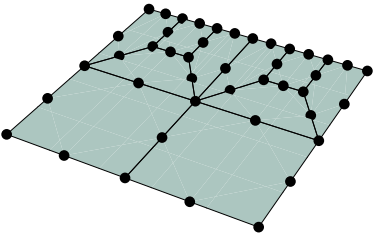
3D mesh topology

<p>O1 div={3,2,2}</p> 	<pre>SMTMesh["Solid", "O1", {3, {{{{0, 0, 0}, {3, 0, 0}}, {{0, 2, 0}, {3, 2, 0}}}}, {{{{0, 0, 2}, {3, 0, 2}}, {{0, 2, 2}, {3, 2, 2}}}}}]</pre>
<p>H1 div={3,2,2}</p> 	<pre>SMTMesh["Solid", "H1", {3, {{{{0, 0, 0}, {3, 0, 0}}, {{0, 2, 0}, {3, 2, 0}}}}, {{{{0, 0, 2}, {3, 0, 2}}, {{0, 2, 2}, {3, 2, 2}}}}}]</pre>
<p>O2 div={3,2,2}</p> 	<pre>SMTMesh["Solid", "O2", {3, {{{{0, 0, 0}, {3, 0, 0}}, {{0, 2, 0}, {3, 2, 0}}}}, {{{{0, 0, 2}, {3, 0, 2}}, {{0, 2, 2}, {3, 2, 2}}}}}]</pre>
<p>H2 div={4,3,2}</p> 	<pre>SMTMesh["Solid", "H2", {4, {{{{0, 0, 0}, {4, 0, 0}}, {{0, 3, 0}, {4, 3, 0}}}}, {{{{0, 0, 2}, {4, 0, 2}}, {{0, 3, 2}, {4, 3, 2}}}}}]</pre>
<p>H2S div={4,3,2}</p> 	

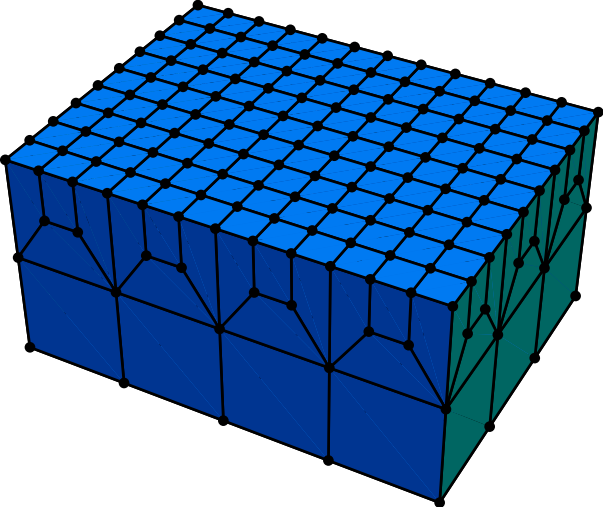
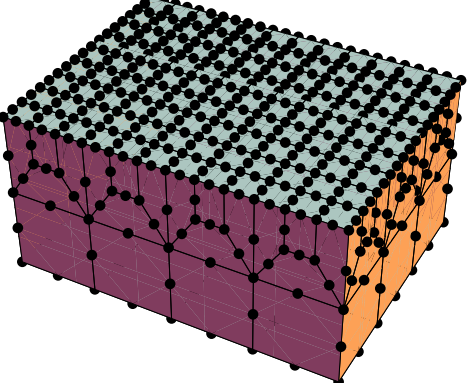


```
SMTMesh["Solid", "H2S",  
  {4, 3, 2}, {{{{0, 0, 0}, {4,  
    {0, 3, 0}, {4, 3, 0}}},  
  {{{0, 0, 2}, {4, 0, 2}},  
    {{0, 3, 2}, {4, 3, 2}}}]}
```

2D mesh topology with mesh refinement

<p style="text-align: center;">Q1-rf</p> 	<pre>SMTMesh["2D", "Q1-rf", {3, 2} {{{0, 0}, {3, 0}}, {{0, 2}, {</pre>
<p style="text-align: center;">S1-rf</p> 	<pre>SMTMesh["Shell", "S1-rf", {4, 3}, {{{0, 0, 0}, {4, 0 {0, 3, 0}, {4, 3, 0}}}]</pre>
<p style="text-align: center;">Q2S-rf</p> 	<pre>SMTMesh["2D", "Q2S-rf", {2, 2} {{{0, 0}, {3, 0}}, {{0, 2}, {</pre>
<p style="text-align: center;">S2S-rf</p> 	<pre>SMTMesh["Shell", "S2S-rf" {2, 2}, {{{0, 0, 0}, {3, 0 {0, 2, 0}, {3, 2, 0}}}]</pre>

3D mesh topology with mesh refinement

<p style="text-align: center;">H1-rf</p> 	<pre>SMTMesh["Solid", "H1-rf", {4, 3, 2}, {{{{0, 0, 0}, {4, {0, 3, 0}, {4, 3, 0}}}, {{{0, 0, 2}, {4, 0, 2}}, {{0, 3, 2}, {4, 3, 2}}}}}]</pre>
<p style="text-align: center;">H2S-rf</p> 	<pre>SMTMesh["Solid", "H2S-rf", {4, 3, 2}, {{{{0, 0, 0}, {3, {0, 2, 0}, {3, 2, 0}}}, {{{0, 0, 1}, {3, 0, 1}}, {{0, 2, 1}, {3, 2, 1}}}}}]</pre>

Mastermesh

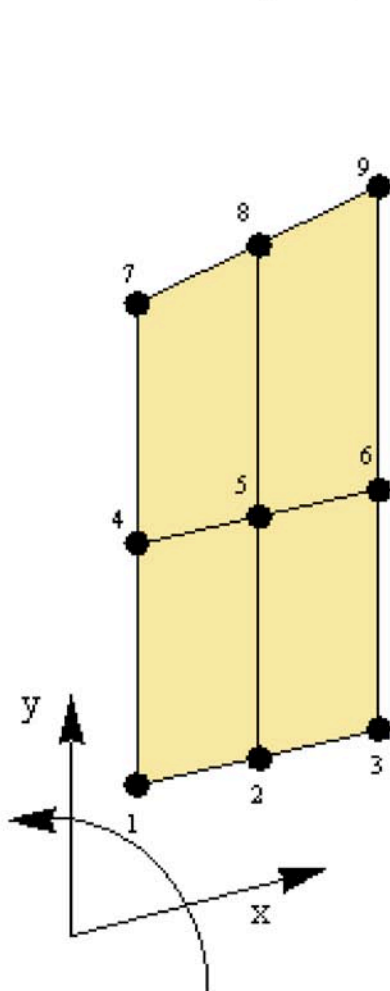
The *mastermesh* parameter is a regular two or three dimensional array of arbitrary number of points that outlines the boundary of the problem domain. The *mastermesh* argument defines nodes of a master mesh that is used to create actual mesh. The coordinates of nodes of actual mesh are obtained as multidimensional interpolation of master mesh nodes. The order of master mesh interpolation is defined by the "InterpolationOrder" option.

The parameter *division* is a number of elements of actual mesh in each direction (e.g. {nx×ny×nz}).

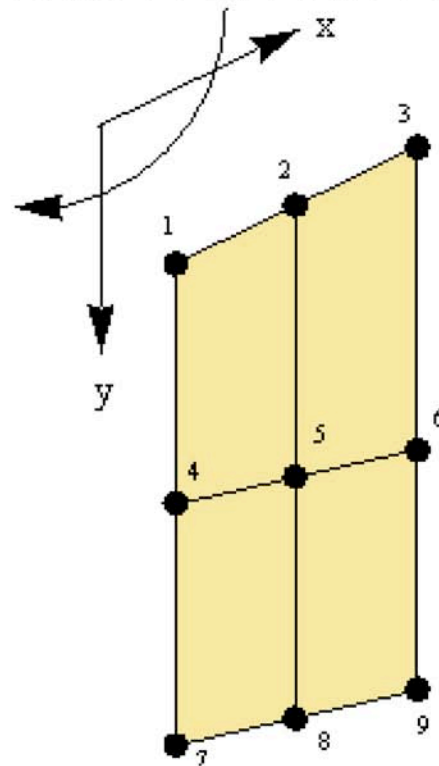
IMPORTANT: The *mastermesh* points have to be given in a order that defines a counter-clockwise local coordinate system of the master mesh.

IMPORTANT : The "InterpolationOrder" is NOT related to the interpolation order of the elements used, the interpolation order of the elements is defined by the elements topology.

```
SMTMesh["A","Q1",{nx,ny},{{1,2,3},{4,5,6},{7,8,9}}]
```

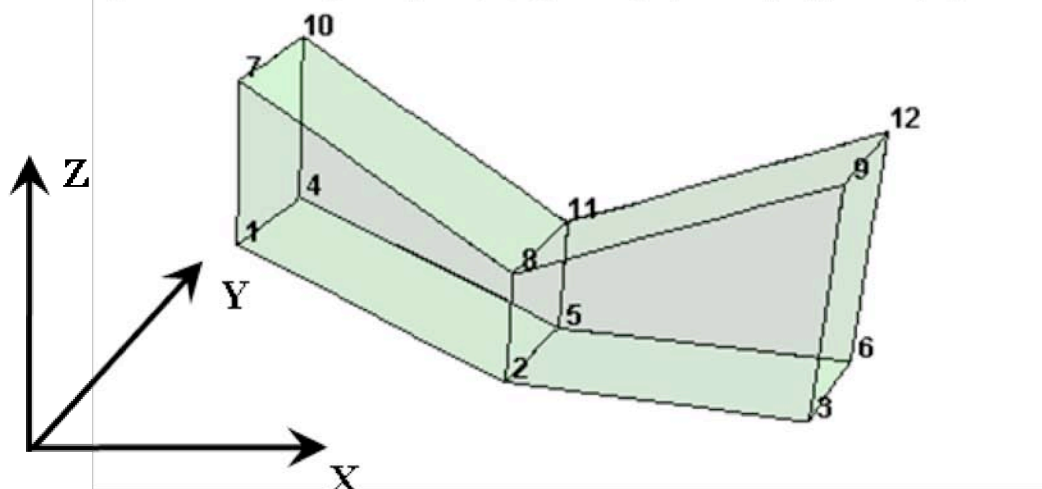


Correct ordering of the master mesh nodes defines counter-clockwise local coordinate system.



False ordering of the master mesh nodes defines a clockwise local coordinate system that is not consistent with most FEM formulations.

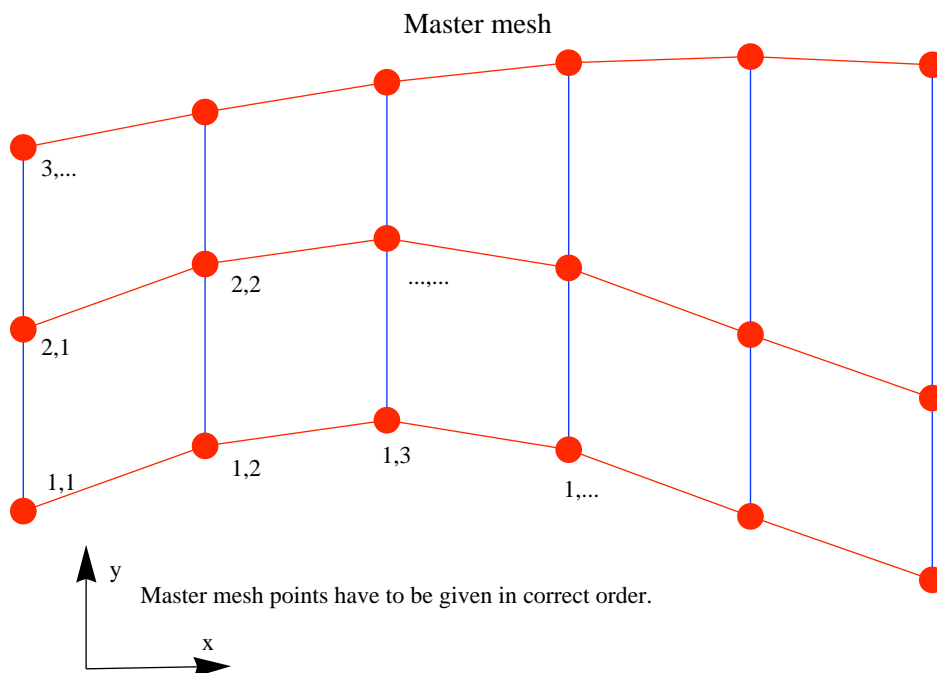
```
SMTMesh["A","H1",{nx,ny,nz},{{{1,2,3},{4,5,6}},{{7,8,9},{10,11,12}}}]
```



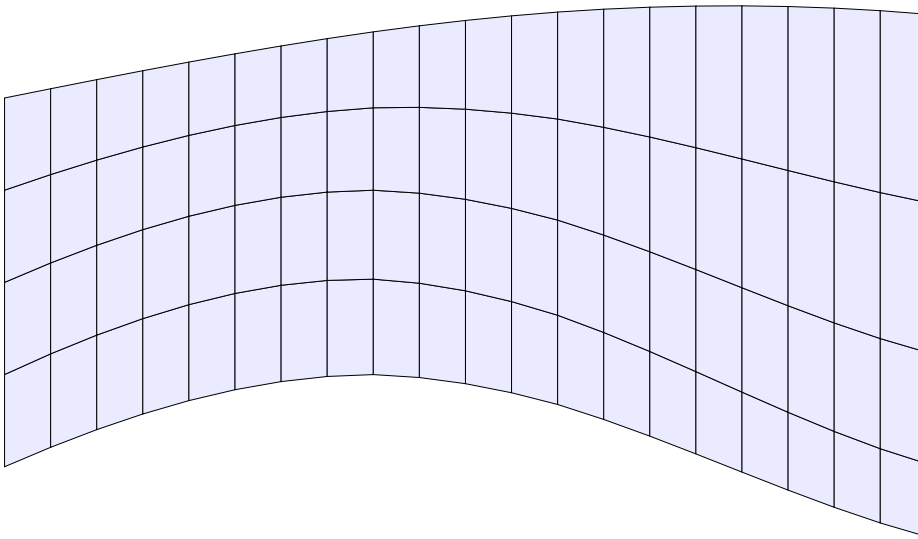
Correct ordering of the master mesh nodes defines 3D counter-clockwise local coordinate system.

Example

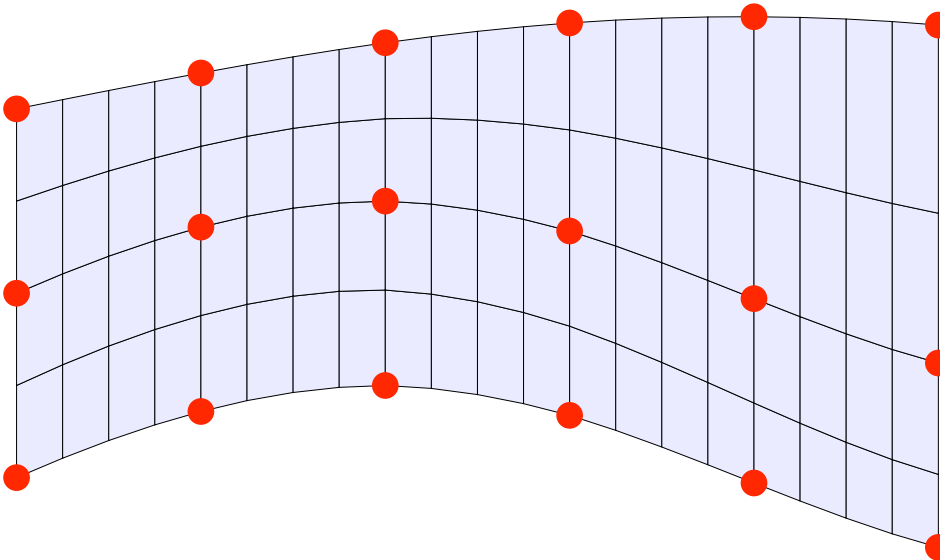
```
<< AceFEM` ;
L = 10.;
mastermesh = {Table[ {x, Sin[4 x / L]}, {x, 0, L, 2}],
  Table[ {x, Sin[4 x / L] + 2}, {x, 0, L, 2}],
  Table[ {x, Sin[2 x / L] + 4}, {x, 0, L, 2}]}];
SMTInputData[];
SMTAddDomain["A", "SEPSQ1DFHYQ1NHookeA", {"E *" -> 1000., "v *" -> .49}];
SMTMesh["A", "Q1", {20, 4}, mastermesh, "BodyID" -> "a"];
SMTAnalysis[];
```



Actual mesh 20*4



Actual mesh + master mesh



SMTAddSensitivity

SMTAddSensitivity[*sID*, *current_value*,
*dID*₁ → {*SensType*, *SensTypeIndex*}, *dID*₂ → ...]

add sensitivity parameter to the list of sensitivity parameter (sID represents the sensitivity parameter identified by its c value *value* and a set of rules that identifies SensType and SensTypeIndex of the parameter for selected domains, all remaining domains by default depend on parameter only implicitly)

<i>SensType code</i>	<i>Description</i>	<i>SensTypeIndex parameter</i>
1	parameter sensitivity	an index of the current material parameter as specified in the description of the material models (the value in general depends on an element type)
2	shape sensitivity	an index of the current shape parameter (the value must be the same for all domains, the shape velocity field has to be additionally defined after the SMSAnalysis)
3	implicit sensitivity	parameter has no meaning
4	essential boundary condition sensitivity	an index of the current boundary condition sensitivity parameter (essential or natural) (the value must be the same for all domains, the BC velocity field has to be additionally defined after the SMSAnalysis)
5	natural boundary condition sensitivity	an index of the current boundary condition sensitivity parameter (essential or natural) (the value must be the same for all domains, the BC velocity field has to be additionally defined after the SMSAnalysis)

Codes for the "SensType" and "SensTypeIndex" switches.

See also: SMTSensitivity, SMTAddSensitivity, Standard user subroutines, Solid, Finite Strain Element for Direct and Sensitivity Analysis, Parameter, Shape and Load Sensitivity Analysis of Multi-Domain Example .

One of the input data in the case of shape sensitivity analysis is also the derivation of the nodal coordinates with respect to the shape parameters (shape velocity field) and derivation of the boundary conditions with respect to the boundary conditions parameters (BC velocity field). The shape velocity field is by default stored in a nodal data field sX ($nd\$\$[i,"sX",j,k]$) and should be initialized by the user. The BC velocity field is by default stored in a nodal data field sBt and is calculated as $sBt = sBp + \Delta\lambda \text{sdB}$, where sBp is the BC velocity field at the end of previous step and the sdB is the reference value of the BC velocity field. The sBp ($nd\$\$[i,"sBp",j,k]$) and the sdB ($nd\$\$[i,"sdB",j,k]$) fields have to be initialized by the user after the SMSAnalysis command.

Example

```

<< AceFEM` ;
SMTInputData[];
L = 10; Q = 500; v = 1;
SMTAddDomain[{"Ω1", "ExamplesHypersolid2D", {"E *" -> 1000, "v *" -> 0.3}},
  {"Ω2", "ExamplesHypersolid2D", {"E *" -> 5000, "v *" -> 0.2}}];
SMTAddEssentialBoundary["X" == 0 &, 1 -> 0, 2 -> 0];
SMTAddNaturalBoundary["Y" == L / 2 &, 2 -> Q / 40];
SMTAddEssentialBoundary["X" == L && "Y" == 0 &, 2 -> -v];
SMTMesh["Ω1", "Q1", {20, 20}, {{{0, 0}, {L / 2, 0}}, {{0, L / 2}, {L / 2, L / 2}}];
SMTMesh["Ω2", "Q1", {20, 20}, {{{L / 2, 0}, {L, 0}}, {{L / 2, L / 2}, {L, L / 2}}];
SMTAddSensitivity[{
  (* E is the first material parameter on "Ω1" domain*)
  {"E", 1000, "Ω1" -> {1, 1}},
  (* v is the second material parameter on "Ω2" domain*)
  {"v", 0.2, "Ω2" -> {1, 2}},
  (* L is first shape parameter for all domains*)
  {"L", L, _ -> {2, 1}},
  (* Q is the first boundary condition parameter - prescribed force*)
  {"Q", 1, _ -> {5, 1}},
  (* v is the second boundary condition parameter - prescribed displacement *)
  {"v", 1, _ -> {4, 2}}
}];
SMTAnalysis[];

```

This sets an initial sensitivity of node coordinates (shape velocity field) with respect to L for shape sensitivity analysis.

```
SMTNodeData["sX", Map[{{#[[2]] / L, 0} &, SMTNodes]]];
```

This sets an initial sensitivity of prescribed force (BC velocity field) with respect to the intensity of the distributed force Q .

```
SMTNodeData["Y" == L / 2 &, "sdB", {0, 1 / 40., 0, 0}];
```

This sets an initial sensitivity of prescribed displacement (BC velocity field) with respect to prescribed displacement v .

```
SMTNodeData["X" == L && "Y" == 0 &, "sdB", {0, 0, 0, -1}];
```

Analysis

SMTAnalysis

SMTAnalysis[options] check the input data, create data structures and start the an:

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*.

<i>option</i>	<i>description</i>	<i>default value</i>
"Output"	output file name (for comments, debugging, etc.)	False (no output file)
"PostOutput"	post-processing data file name (see also SMTPut)	False
"PostInput"	input data can be imported from the previously stored post-processing data file (see also SMTGet)	False
"Solver"	0 ⇒ appropriate linear solver is chosen automatically <i>solverID</i> ⇒ solver with linear solver identification number <i>solverID</i> { <i>solverID</i> , <i>p</i> ₁ , <i>p</i> ₂ ,...} ⇒ solver with linear solver identification number <i>solverID</i> and set of parameters for initialisation (see also SMTSetSolver)	0
"OptimizeDll"	True ⇒ set compiler options for the fastest code False ⇒ set compiler options for debugging	Automatic
"SearchFunction"	function applied on coordinates of nodes before the search procedures if the coordinates are not numbers (e.g before "Tie" , SMTFindNodes, etc.)	(#&)
"Precision"	number of significant digits used within the search procedures (e.g for the "Tie" option, SMTFindNodes, etc.)	6
"Tolerance"	the numbers smaller in absolute magnitude than " <i>Tolerance</i> " are replaced by 0 within the search procedures	10 ⁻¹⁰
"Tie"	True ⇒ join nodes which have coordinates and node identification with the same values {True, <i>nodeselector</i> } ⇒ join nodes which have coordinates and node identification with the same values, except the nodes that match the <i>nodeselector</i> (see also Selecting Nodes) False ⇒ suppress tie (the tie is by default suppressed for all nodes that have node identification switch -T, see Node Identification)	True
"Debug"	True ⇒ prevent closing of the CDriver console on exit	False
"NodeReordering"	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"	Automatic
"ContactPairs"	specify the the pairs slave-body/master - body for which the possible contact condition is checked {{ <i>slaveBodyID</i> ₁ , <i>masterBodyID</i> ₁ }, { <i>slaveBodyID</i> ₂ , <i>masterBodyID</i> ₂ },...}	Automatic

Options for the `SMTAnalysis` function.

SMTNewtonIteration

`SMTNewtonIteration[]` one iteration of the general Newton–Raphson iterative procedure

See also: Iterative solution procedure, Bending of the column (path following procedure, animations, 2D solids).

SMTNextStep

`SMTNextStep[Δt, Δλ]` reset environment variables to the new time increment Δt ,
boundary conditions multiplier increment to $\Delta\lambda$,
update time and multiplier,
and make the solution at the end of the previous time step to
be equal to current solution (**ap≐at, sp≐st, hp≐ht, Bp≐Bt**)

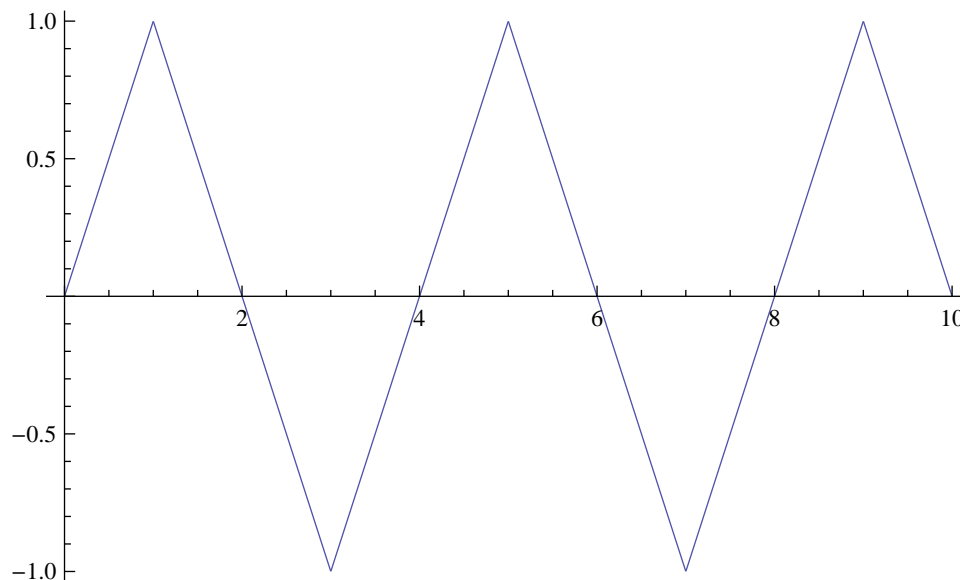
`SMTNextStep[Δt, λ]` ≐ `SMTNextStep[Δt, λ[t+Δt]−λ[t]`
where λ is a multiplier function

See also: Iterative solution procedure, Bending of the column (path following procedure, animations, 2D solids)

Example: Zig-zag load

This is a path following procedure with a zig-zag multiplier increment λ as a function of time.

```
λ[t_] := If[OddQ[Floor[(t + 1) / 2]], 1, -1] (2 Floor[(t + 1) / 2] - t);
Plot[λ[t], {t, 0, 10}]
```



```
Do[SMTNextStep[0.1, λ];
  While[SMTConvergence[10^-8, 10], SMTNewtonIteration[]];
  , {i, 1, 100}]
```

SMTStepBack

SMTStepBack[] make 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**)

See also: Iterative solution procedure, Bending of the column (path following procedure, animations, 2D solids)

SMTConvergence

SMTConvergence[*tol, n, type, options*] check if the convergence conditions for the iterative procedure have been satisfied

SMTConvergence[] \equiv SMTConvergence[$10^{-7}, 15, \text{"Abort"}$]

SMTConvergence["Reset"] The SMTConvergence command continuously stores data related to iterations and makes decisions accordingly to the history of iterations. The SMTConvergence["Reset"] command clears the history of iterations.

See also: Iterative solution procedure, Bending of the column (path following procedure, animations, 2D solids)

<i>type</i>	<i>return value</i>	<i>description</i>
"Abort"	True	$\ \Delta \mathbf{a}^i\ > tol$, one more iteration is required
	none	iterative process is aborted in the case of divergence
"Analyze"	True	$\ \Delta \mathbf{a}^i\ > tol$, one more iteration is required
	False	$\ \Delta \mathbf{a}^i\ < tol$, convergence condition for the iterative procedure has been satisfied
	<i>ctype</i>	divergence
{"Adaptive", <i>m</i> , $\Delta \lambda_{min}, \Delta \lambda_{max}, \lambda_{max}$ }	True	$\ \Delta \mathbf{a}^i\ > tol$, one more iteration is required
	{False, $\Delta \lambda$, True, $\ \Delta \mathbf{a}^i\ $ }	convergence conditions for the iterative procedure have been satisfied
	{False, 0, False, "MaxBound"}	the value of the boundary conditions multiplier is larger or equal λ_{max} ($ \lambda \geq \lambda_{max}$)
	{True, $\Delta \lambda$, True, <i>ctype</i> }	divergence
"Adaptive"	{True, 0, False, "MinBound"}	the value of the boundary conditions multiplier increment is less than prescribed minimum ($ \Delta \lambda < \Delta \lambda_{min}$)
	\equiv	{"Adaptive", $2/3 n, 0, \infty, \infty$ }
{"Adaptive Time", <i>m</i> , $\Delta t_{min}, \Delta t_{max}, t_{max}$ }	\equiv	adaptive time increment that is equivalent to the adaptive multiplier increment

Return values accordingly to the *type*.

<i>ctype</i>	<i>description</i>
"N/A"	error in numerical procedures
"Divergence"	$\ \Delta \mathbf{a}^i\ \rightarrow \infty$
"Alternate"	alternating solution
"ErrorStatus"	the error status flag has been set to 2
"Unknown"	the reason for the lack of convergence of iterative procedure is unknown

Divergence type accordingly to the *ctype*.

<i>option</i>	<i>description</i>	<i>default value</i>
"Alternate"	<p>False \Rightarrow alternating solution is ignored "Divergence" \Rightarrow alternating solution means divergence True \equiv "Divergence"</p> <p>Automatic \equiv $\{\{\Delta t_{max}/4, \Delta \lambda_{max}/4\}, \{\Delta t_{max}/10, \Delta \lambda_{max}/10\}, \{\infty, \infty\}\}$</p> <p>$\{\{\Delta t_l, \Delta \lambda_l\}, \{\Delta t_g, \Delta \lambda_g\}, \{\Delta t_{g1}, \Delta \lambda_{g1}\}\} \Rightarrow$ if alternating solution occurs start the following procedure: first if $\Delta t < \Delta t_l$ $\Delta \lambda < \Delta \lambda_l$ then set SMTIData["SubIterationMode",1]; if solution still alternate and $\Delta t < \Delta t_g$ $\Delta \lambda < \Delta \lambda_g$ then set SMTIData["GlobalIterationMode",1]; convergence conditions have been satisfied and $\Delta t > \Delta t_{g1}$ $\Delta \lambda > \Delta \lambda_{g1}$ then set SMTIData["GlobalIterationMode",0] and repeat iterations</p>	"Divergence"
"PostIteration"	<p>enables an additional call to the SKR user subroutines after the convergence of the global solution (see SMSTemplate) Automatic \Rightarrow perform post-iteration call accordingly to the value of the template constant of specific element (see SMSTemplate) False \Rightarrow prevent post-iteration call True \Rightarrow force post-iteration call</p>	Automatic
"IgnoreMinBound"	<p>If an alternating solution occurs at the minimum time/multiplier increment ($\Delta \lambda < \Delta \lambda_{min}$ or $\Delta t < \Delta t_{min}$) then ignore the minimum increment condition and proceed for another time/multiplier step.</p>	False
"AlternativeTarget"	<p>The primal target of the adaptive path following procedure is to reach λ_{max}/t_{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/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.</p>	(True&)

Options for the SMTConvergence function.

Where *tol* is the tolerance, *n* is the maximum number of iterations, *m* is the desired number of iterations, $\Delta \lambda$ is the proposed multiplier (or time) increment, $\Delta \lambda_{min}$ is the minimum value of the boundary conditions multiplier increment (or time increment), $\Delta \lambda_{max}$ is the maximum value of the boundary conditions multiplier increment (or time increment),

λ_{max} is the minimum value of the boundary conditions multiplier (or time), and *ctype* is the character constant that explains the reason for the lack of convergence of the iterative procedure.

SMTDump

<p>SMTDump[<i>name</i>] saves restart data to the files <i>name.rst</i> and <i>name.rsb</i></p> <p>SMTDump[<i>name</i>, <i>syml</i>,...] saves restart data to the files <i>name.rst</i> and <i>name.rsb</i> and appends definitions associated with the given symbols</p>
--

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 two files. The *name.rst* file is a text file that contains input data and eventual definitions of specified symbols. The *name.rsb* file is a binary file that stores nodal and element data.

*C*Driver specific!

See also: SMTRestart

Example: Restarting the AceFEM session

Here we first analyze the give structure up to the load level 500.

```
<< AceFEM` ;
```

```

SMTInputData[];
SMTAddDomain["A", "SEPSQ1DFHYQ1NeoHooke", {"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];
SMTMesh["A", "Q1", {80, 5}, Array[{#2, #2 Sin[#2 // N] / 20 + 4 #1} &, {2, 40}]];
SMTAnalysis[];
SMTNextStep[1, 100];
While[
  While[step = SMTConvergence[10^-7, 15, {"Adaptive", 8, .01, 300, 500}],
    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[1, step[[2]]
];
Show[SMTShowMesh["Show" -> False, "BoundaryConditions" -> True],
SMTShowMesh["DeformedMesh" -> True, "Mesh" -> False, "Field" -> "Sxy",
"Contour" -> 20, "Show" -> False, "Legend" -> False], PlotRange -> All]

```

```

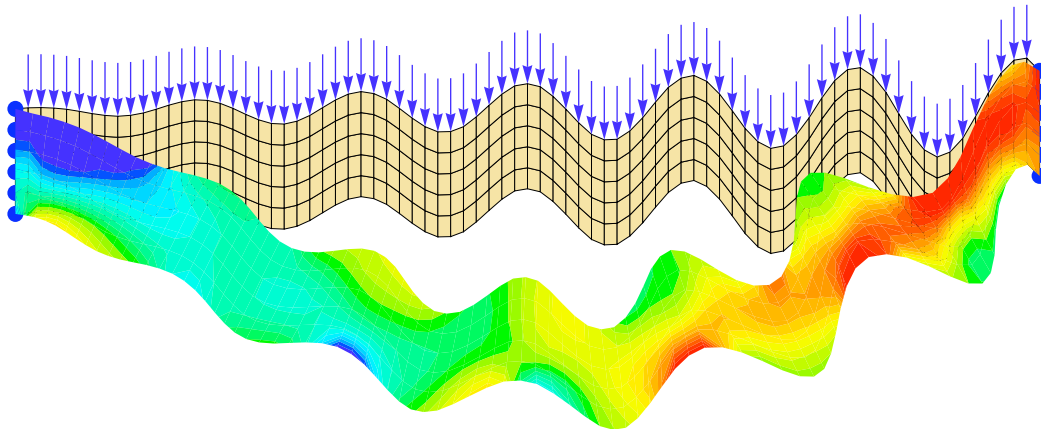
T/ΔT=1./1. λ/Δλ=100./100. ||Δa||/||Ψ||=8.69452 × 10-9
/9.03293 × 10-13 Iter/Total=8/8 Status=0/{Convergence}

T/ΔT=2./1. λ/Δλ=226.531/126.531 ||Δa||/||Ψ||=4.28439 × 10-11
/9.26182 × 10-13 Iter/Total=7/15 Status=0/{Convergence}

T/ΔT=3./1. λ/Δλ=415.035/188.505 ||Δa||/||Ψ||=6.93678 × 10-8
/1.00227 × 10-12 Iter/Total=6/21 Status=0/{Convergence}

T/ΔT=4./1. λ/Δλ=500./84.9646 ||Δa||/||Ψ||=4.55315 × 10-8
/9.74084 × 10-13 Iter/Total=5/26 Status=0/{Convergence}

```



The state of the analysis at the load level 500 is then stored in "dump1" restart files.

```
SMTDump["dump1"];
```

Old restart files dump1
are replaced by new. See also: `SMTDump`

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.

```
<< AceFEM` ;

SMTRestart["dump1"];

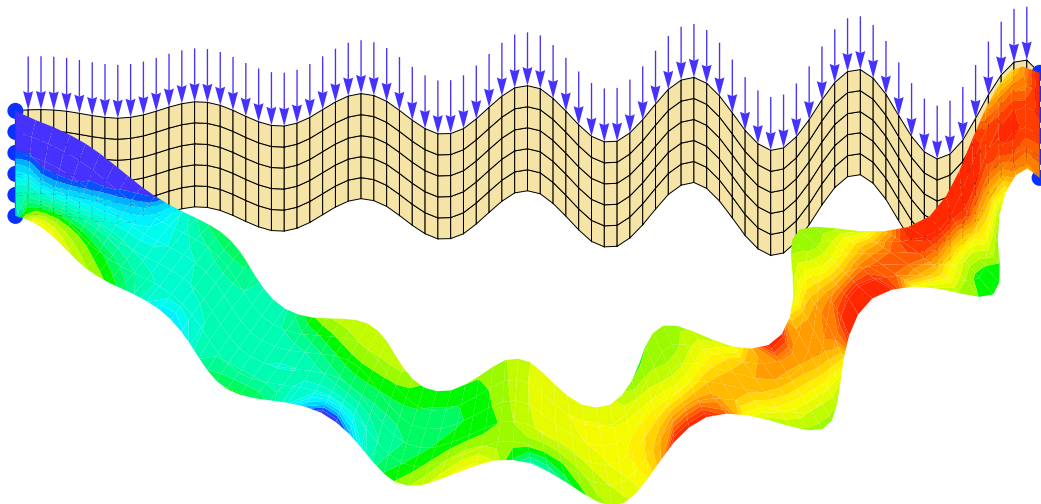
While[
  While[step = SMTConvergence[10^-7, 15, {"Adaptive", 8, .01, 300, 1000}] ,
    SMTNewtonIteration[]];
  SMTStatusReport[];
  If[step[[4]] == "MinBound", SMTStatusReport[" $\Delta\lambda < \Delta\lambda_{min}$ "]];
  If[Not[step[[1]]], SMTShowMesh["DeformedMesh" -> True,
    "Field" -> "Sxy", "Mesh" -> False, "Contour" -> 20, "Show" -> "Window"]];
  step[[3]] ,
  If[step[[1]], SMTStepBack[]];
  SMTNextStep[1, step[[2]]]
];
Show[SMTShowMesh["Show" -> False, "BoundaryConditions" -> True],
SMTShowMesh["DeformedMesh" -> True, "Mesh" -> False, "Field" -> "Sxy",
"Contour" -> 20, "Show" -> False, "Legend" -> False], PlotRange -> All]

T/ $\Delta$ T=4./1.  $\lambda/\Delta\lambda=500./84.9646$   $\|\Delta a\|/\|\Psi\|=4.55315 \times 10^{-8}$ 
/9.74084  $\times 10^{-13}$  Iter/Total=6/27 Status=0/{Convergence}

T/ $\Delta$ T=5./1.  $\lambda/\Delta\lambda=642.186/142.186$   $\|\Delta a\|/\|\Psi\|=3.18359 \times 10^{-13}$ 
/1.01415  $\times 10^{-12}$  Iter/Total=6/33 Status=0/{Convergence}

T/ $\Delta$ T=6./1.  $\lambda/\Delta\lambda=880.129/237.943$   $\|\Delta a\|/\|\Psi\|=2.19246 \times 10^{-12}$ 
/1.13948  $\times 10^{-12}$  Iter/Total=6/39 Status=0/{Convergence}

T/ $\Delta$ T=7./1.  $\lambda/\Delta\lambda=1000./119.871$   $\|\Delta a\|/\|\Psi\|=1.38806 \times 10^{-9}$ 
/1.237  $\times 10^{-12}$  Iter/Total=5/44 Status=0/{Convergence}
```



SMTRestart

SMTRestart[*name*] reads the restart data from the *name.rst* and *name.rsb* files and continues the AceFEM session from the point of the corresponding SMTDump command

CDriver specific!

See also: SMTDump

SMTSensitivity

SMTSensitivity[] solve the sensitivity problem for all sensitivity parameters

The following steps are performed for all sensitivity parameters:

- ⇒ user subroutine "*Sensitivity pseudo-load*" is called for each element,
- ⇒ sensitivity pseudo-load vector is added to the global pseudo-load vector Ψ ,
- ⇒ set of linear equations is solved $\mathbf{K} \frac{\partial \mathbf{a}}{\partial \phi_i} = \Psi$ that yields sensitivity of global (nodal) variables
- ⇒ user subroutine "*Dependent sensitivity*" is called for each element to resolve sensitivity of local (element) variables

See also: SMTSensitivity, SMTAddSensitivity, Standard user subroutines, Solid, Finite Strain Element for Direct and Sensitivity Analysis, Parameter, Shape and Load Sensitivity Analysis of Multi-Domain Example .

SMTTask

See: User Defined Tasks

See also: Standard user subroutines

SMTStatusReport

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

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

```
SMTStatusReport[SMTNodeData[5, "at"]]
```

SMTSessionTime

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

SMTErrCheck

SMTErrCheck[
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
 SMTErrCheck[] ≡ SMTErrCheck[1,1,1,0]

There are two types of environment variables identifying various error events. The environment variable *idata\$\$["ErrorStatus"]* (see table below) identifies the general error status.

<i>Error status</i>	<i>Description</i>
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.

Additional environment variables "MissingSubroutine", "SubDivergence", "ElementState", "ElementShape" and "MaterialState" (see [Environment Data](#)) then give more information about the error. Error event variables are set in a user subroutine. They should be increased by 1 whenever the error condition that specifies specific event appears. The *SMTErrCheck* function is used to process the error events. In the case of error event the error message is produced and all monitored variables are set back to zero value.

Here is the part of the symbolic input where the error event is reported if the Jacobean of the nonlinear coordinate mapping (*J*) becomes negative.

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

SMTSimulationReport

SMTSimulationReport[] produce report identifying the percentage of time spent in specific tasks during the analysis
 SMTSimulationReport[
idata_names, rdata_names, comment] print out additional information stored in user defined integer and real type environment variables and arbitrary comments (for details see Code Profiling)

<i>option</i>	<i>description</i>	<i>default value</i>
"Output"	a list of output devices: "Console" ⇒ current notebook "File" ⇒ current output file (if defined)	{"Console","File"}

Options for the *SMTSimulationReport* function.

Postprocessing

SMTShowMesh

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

<i>option</i>	<i>description</i>	<i>default value</i>
"Show"	specifies output device	True False
"Label"	an expression or the list of expressions to be printed as a label for the plot	None
"Mesh"	display mesh as wire frame accordingly to the mesh type Possible mesh types include True, False, Automatic, RGBColor[red,green,blue] , GreyLevel[level].	True
"Marks"	display nodal and element numbers	False
"BoundaryConditions"	mark nodes with the non-zero boundary conditions	False
"Domains"	list of the domain identifications (only the domains with the domain identification included in the list are plotted)	All
"Elements"	fill in the element surfaces with the element surface color or with the contour lines	True
"Field"	the vector of nodal values that defines scalar field used to calculate element surface color or contour lines	False
"Contour"	display contour lines of the scalar field p defined by the "Field" option	False
"Legend"	include legend specifying the colors and the range of the "Field" values	True
"DeformedMesh"	display deformed mesh by adding the vector field u multiplied by the "Scale" option to the nodal coordinates X ($\bar{X}:=X+u$)	False
"Scale"	scaling factor for deformed mesh	1.
"Opacity"	graphics directive which specifies that graphical objects which follow are to be displayed,if possible, with given opacity (number between 0 and 1) (New in <i>Mathematica</i> 6)	False
"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"	the SMTShowMesh command is not executed if the absolute difference in time for two successive SMTShowMesh calls is less than "TimeFrequency" (New in <i>Mathematica</i> 6)	0
"MultiplierFrequency"	the SMTShowMesh command is not executed if the absolute difference in multiplier for two successive SMTShowMesh calls is less than "MultiplierFrequency" (New in <i>Mathematica</i> 6)	0
"StepFrequency"	the SMTShowMesh command is not executed if the absolute difference in step number for two successive SMTShowMesh calls is less than "StepFrequency" (New in <i>Mathematica</i> 6)	0

Main options for the *SMTShowMesh* function.

The nodes with the non-zero essential boundary condition are marked with color points. The nodes with the non-zero natural boundary condition are marked with arrow for the nodes with two unknowns and with color point otherwise. Before the analysis the Bp and dB boundary values (see also Input Data) are displayed separately, and during the analysis only Bt is displayed.

<i>"Show" option</i>	<i>description</i>
"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"→{"Export", <i>file, format, options</i> }	≡ Export[<i>file, produced graphics, format, options</i>] export data to a file, converting it to a specified format (see also Export commad)
"Show"→{"Animation", <i>keyword, options</i> }	creates subdirectory with the name <i>keyword</i> and exports current graphics as GIF file with the name <i>frame_number.gif</i> (frame numbers are counted automatically and <i>options</i> are the same as for command Export)
"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.

<i>"Contour" option</i>	<i>description</i>
"Contour"→True	display 10 contour lines
"Contour"→n	display <i>n</i> contour lines
"Contour"→{ <i>min,max,n</i> }	display <i>n</i> contour lines taken from the range { <i>min,max</i> }

Methods for setting up contour lines.

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

Methods for setting up contour lines.

<i>"Label" option</i>	<i>description</i>
"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.

<code>"DeformedMesh"</code> <i>option</i>	<i>description</i>
<code>"DeformedMesh" → True</code>	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 <i>SMTPost</i> (see <i>SMTPost</i>) command. If the "DeformedMeshX", "DeformedMeshY" and "DeformedMeshZ" postprocessing codes are specified by the user then $\mathbf{u}_X = \text{SMTPost}["\text{DeformedMeshX}"],$ $\mathbf{u}_Y = \text{SMTPost}["\text{DeformedMeshY}"],$ $\mathbf{u}_Z = \text{SMTPost}["\text{DeformedMeshZ}"],$ else $\mathbf{u}_X = \text{SMTPost}[1],$ $\mathbf{u}_Y = \text{SMTPost}[2],$ $\mathbf{u}_Z = \text{SMTPost}[3].$
<code>"DeformedMesh" → {\mathbf{u}_X, \mathbf{u}_Y, \mathbf{u}_Z}</code>	user defined displacement vectors \mathbf{u}_X , \mathbf{u}_Y , \mathbf{u}_Z

Methods for setting up deformed mesh.

<code>"Marks"</code> <i>option</i>	<i>description</i>
<code>"Marks" → True</code>	display nodal and element numbers $\equiv \text{"Marks"} \rightarrow \{\text{"NodeNumber"}, \text{"ElementNumber"}\}$
<code>"Marks" → False</code>	no numbers
<code>"Marks" → "ElementNumber"</code>	display element numbers
<code>"Marks" → "NodeNumber"</code>	display node numbers

Methods for setting up contour lines.

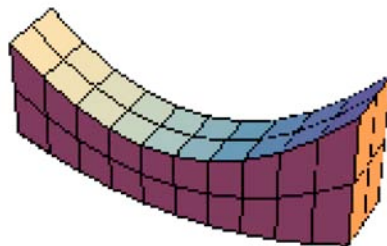
<i>option</i>	<i>description</i>	<i>default value</i>
"TextStyle"	specifies the default style and font options with which text should be rendered	{FontSize→9, FontFamily→"Arial"}
"NodeMarks"	mark all the nodes with the circle	False
"NodeTagOffset"	position of the node number relative to the node	{.01,.01,.01}
"NodeID"	list of the node identifications (only the nodes with the node identification included in the list are plotted)	Automatic
"ColorFunction"	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])	Automatic
"ZoomNodes"	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)	False
"ZoomElements"	the parameter is used to select elements (it has one of the forms described in section Selecting Elements, only the selected elements are depicted)	False
"NodeMarksField"	the vector of nodal values that are used to calculate color for each nodal point mark	False
"RawOutput"	instead of the graphics return raw data as follows: { vertex coordinates, vertex field values, vertex collors that correspond to vertex values, legend graphics object, complete graphics object } (New in <i>Mathematica</i> 6)	False
"ShowFor"	specify a list of rules that transforms symbolic expressions (e.g. for nodal coordinates) into numerical values (option is <i>MDriver</i> specific, see also Semi-analytical solutions)	Automatic

Additional options for the *SMTShowMesh* function.

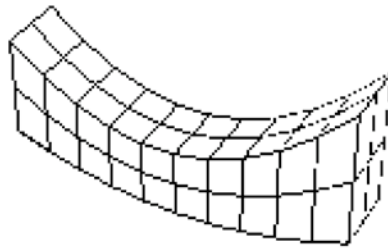
The *SMTShowMesh* function is the main postprocessing function for displaying the element mesh, the boundary conditions and arbitrary postprocessing quantities. The vectors of nodal values p , n , u_x , u_y , u_z are arbitrary vectors of *NoNodes* numbers (see also *SMTPost*). Several examples are presented in Postprocessing (3D heat conduction).

examples

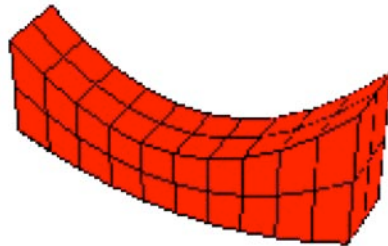
`SMTShowMesh[]`



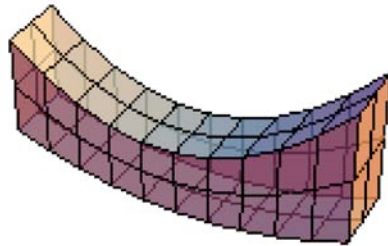
```
SMTShowMesh["Elements"→White,  
Lighting→{{"Ambient",White}}]
```



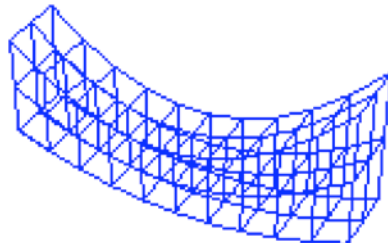
```
SMTShowMesh["Elements"→Red,  
Lighting→{{"Ambient",White}}]
```



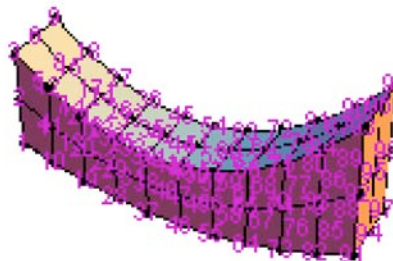
```
SMTShowMesh["Opacity"→0.7]
```



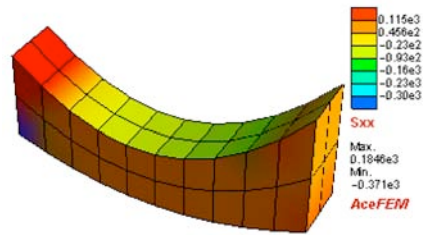
```
SMTShowMesh[  
"Elements"→False,"Mesh"→Blue]
```



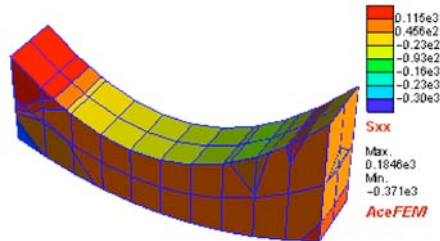
```
SMTShowMesh["Marks"→"NodeNumber"]
```



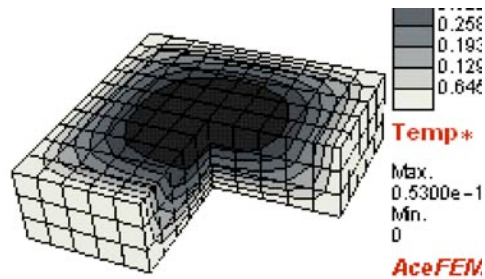
```
SMTShowMesh["Field"→"Sxx"]
```



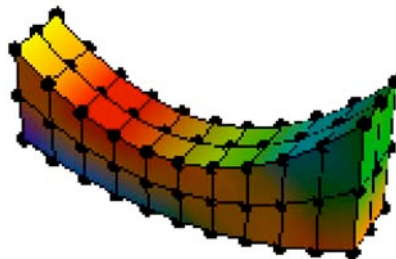
```
SMTShowMesh["Field"→"Sxx","Contour"→True]
```



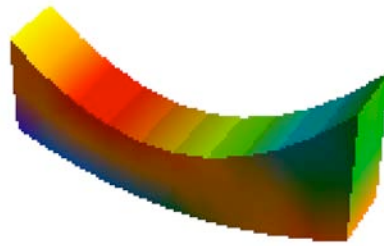
```
SMTShowMesh["Field"→"Temp*","Mesh"→Black,"Contour"→4,"Zoom"→("Z"<=0.3 && ("X"<=0 || "Y">=0)&),"ColorFunction"→Function[{x},ColorData["GrayTones"]][1-x]],Lighting→{"Ambient",White}]
```



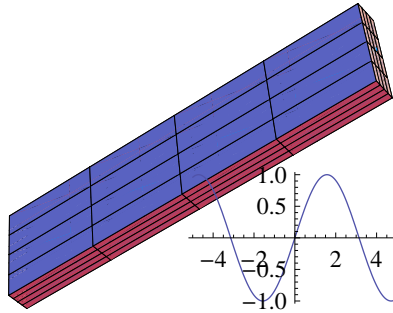
```
SMTShowMesh["Field"→"Szz","NodeMarks"→6,"Legend"→False]
```



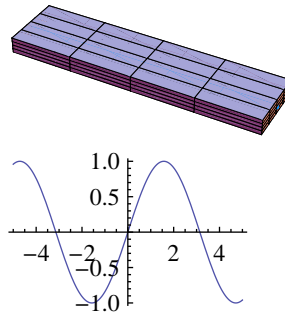

```
SMTShowMesh["Field"→"Szz",
"Mesh"→False,"Legend"→False]
```



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



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



SMTMakeAnimation

`SMTMakeAnimation[keyword]` generates a notebook animation from frames stored by the `SMTShowMesh` command under given *keyword* (new in Mathematica 6)

`SMTMakeAnimation[keyword,"Flash"]` generates a Shockwave Flash animation from frames stored by the `SMTShowMesh` command under given *keyword* (new in Mathematica 6)

`SMTMakeAnimation[keyword,"GIF"]` generates an animated GIF animation from frames stored by the `SMTShowMesh` command under given *keyword*

Animation frames are stored into the *keyword* subdirectory of the current directory. **The existing *keyword* subdirectory is automatically deleted.**

The corresponding `SMSShowMesh` option is `"Show"→{"Animation", keyword}`. The actual size of the frame can vary during the animation resulting in non-smooth animation. To prevent this the size of the frames has to be explicitly prescribed by an additional option `"Show"→{"Animation", keyword, ImageSize->{wspec, hspec}}`. The width and the

height of the animation are specified in printer's points. See also Bending of the column (path following procedure, animations, 2D solids).

SMTResidual

SMTResidual[<i>nodeselector</i>]	evaluate residual (reaction) in nodes defined by <i>nodeselector</i> (see Selecting Nodes) as a sum of residual vectors of all elements that contribute to the node
SMTResidual[<i>nodeselector,elementselector</i>]	evaluate residual (reaction) in nodes defined by <i>nodeselector</i> (see Selecting Nodes) as a sum of residual vectors of all elements defined by <i>elementselector</i> (see Selecting Elements) that contribute to the node

SMTPostData

SMTPostData[<i>pcode,pt</i>]	get a value of post-processing quantity defined by post-processing keyword <i>pcode</i> in point <i>pt</i>
SMTPostData[{ <i>pcode</i> ₁ , <i>pcode</i> ₂ ,...}, <i>pt</i>]	≡ get the values of post-processing quantities { <i>pcode</i> ₁ , <i>pcode</i> ₁ ,...} in point <i>pt</i>
SMTPostData[<i>pcode</i> ,{ <i>pt</i> ₁ , <i>pt</i> ₂ ,...}]	≡ get the values of post-processing quantity <i>pcode</i> in points { <i>pt</i> ₁ , <i>pt</i> ₂ ,...}

The SMTPostData function can only be used if the element user subroutine for data post-processing has been defined (Standard user subroutines). The post-processing code *pcode* can correspond either to the integration point or to the nodal point post-processing quantity.

The given point can lie outside the mesh. The value is obtained in that case by extrapolation of the field from the element that lies closest to the given point.

See also: SMTPost, SMTPointValues, Postprocessing (3D heat conduction).

SMTData

SMTData[<i>gcode_String</i>]	get data accordingly to the code <i>gcode</i>
SMTData[<i>ie,ecode_String</i>]	get element data for the element with the index <i>ie</i> accordingly to the code <i>ecode</i>
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 postprocessing and debugging. An advanced user can use the Data Base Manipulations to access and change all the analysis data base structures during the analysis.

<i>gcode</i>	<i>Description</i>
"Time"	real time
"Multiplier"	natural and essential boundary conditions multiplier (load level)
"Step"	total number of completed solution steps

Various analysis data directly accessible from *Mathematica*.

<i>gcode</i>	<i>Description</i>
"MatrixGlobal"	global matrix according to the last completed action (command yields a <code>SparseArray</code>)
"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 (matrix is obtained by executing one Newton–Raphson iteration (SMTNewtonIteration) without solving the linear system of equations), (command yields a <code>SparseArray</code>)
"Residual"	global residual vector (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*.

<i>ecode</i>	<i>Action</i>	<i>Return values</i>
"All"	collect all the data associated with the element	String
"SKR"	call to the "SKR" user subroutine (see <code>SMSStandardModule</code>)	tangent matrix and residual for element <i>ie</i>
"SSE"	call to the "SSE" user subroutine (see <code>SMSStandardModule</code>)	sensitivity pseudo–load vector for element <i>ie</i>
"SHI"	call to the "SHI" user subroutine	True
"SRE"	call to the "SKR" user subroutine	residual for element <i>ie</i>
"SPP"	call to the "SPP" user subroutine	get arrays of integration point and nodal point postprocessing quantities for element <i>ie</i>
"User <i>n</i> "	call to the user defined "User <i>n</i> " user subroutine where $n=1,2,\dots,9$	True

Various element data directly accessible from *Mathematica*.

SMTPut

SMTPut[*pkey*,
uservector,options] save all post-processing data together with the additional vector of arbitrary real numbers *uservector* to the file specified by the SMTAnalysis option *PostOutput* (see SMTAnalysis) using the keyword *pkey*

SMTPut[*pkey*] \equiv SMTPut[*pkey*,{}]

SMTPut[] \equiv SMTPut[SMTIData["Step"]]

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

Options for the SMTPut function.

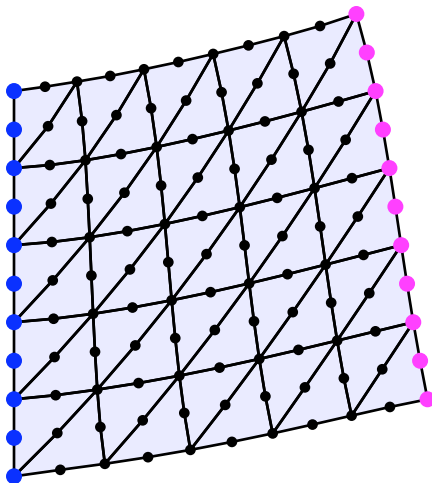
The keyword *pkey* can be arbitrary integer or real number and it has to be unique for each SMTPut call. It is used later by the SMTGet command to locate the position of post-processing data. The SMTPut command creates two files. The .idx file is text file that contains all the input data, all keys and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

CDriver specific!

See also: Standard user subroutines

Example

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 post-processing session.



Analysis session

```
<< AceFEM` ;
SMTInputData [ ] ;
Emodul = 1000 ; L = 5 ;
SMTAddDomain ["Ω1", "SEPET2DFHYT2NeoHooke", {"E *" -> Emodul, "ν *" -> 0.3}] ;
SMTAddEssentialBoundary [{"X" == 0 &, 1 -> 0, 2 -> 0}, {"X" == L &, 2 -> 1}] ;
SMTMesh ["Ω1", "T2", {5, 5}, {{{0, 0}, {5, 0}}, {{0, 5}, {5, 5}}] ;
```

Here the input data structures are saved into the file post.hdf.

```
SMTAnalysis ["PostOutput" -> "post", "Output" -> "tmp.txt"] ;
```

Here are saved some general parameters of the problem .

```
SMTSave [Emodul, L] ;
```

During the analysis all the post-processing 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.

```
SMTNextStep [1, 0.1] ;
While [
  While [step = SMTConvergence [10^-8, 15, {"Adaptive", 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 ["Error: Δλ < Δλmin"] ; Abort [] ;] ;
  step [[3]]
  , If [step [[1]], SMTStepBack []] ;] ;
  SMTNextStep [1, step [[2]]]
];
```

Post-processing session

Here the new, independent session starts by reading input data from the previously stored post-processing data base file.

```
<< AceFEM` ;
SMTInputData [ ] ;
SMTAnalysis ["PostInput" -> "post"] ;
```

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

```
Emodul
1000
```

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

```
{allkeywords, allpostnames} = SMTGet [ ];
allpostnames
allkeywords

{DeformedMeshX, DeformedMeshY, Exx, Exy, Eyx,
 Eyy, Ezz, Mises stress, 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).

```
force = SMTGet [3]

{-3.59259 × 10-13, 46.2459}
```

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

```
SMTPostData ["Sxx", {L / 2, L / 2}]

0.284164
```

SMTGet

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

*C*Driver specific!

See also `SMTPut`

SMTSave

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

*C*Driver specific!

See also `SMTPut`

Data Base Manipulations

SMTIData

<code>SMTIData["code"]</code>	get the value of the integer type environment variable with the code <i>code</i>
<code>SMTIData["code", value]</code>	set the value of the integer type environment variable with the code <i>code</i> to be equal <i>value</i>
<code>SMTIData["All"]</code>	get all names and values of integer type environment variables
<code>SMTRData["code"]</code>	get the value of the real type environment variable with the code <i>code</i>
<code>SMTRData["code", value]</code>	set the value of the real type environment variable with the code <i>code</i> to be equal <i>value</i>
<code>SMTRData["All"]</code>	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 Integer Type Environment Data and Real Type Environment Data.

This returns the number of nodes.

```
SMTIData [ "NoNodes" ]
```

SMTRData

See also: SMTIData

SMTNodeData

<code>SMTNodeData[nodeselector, "code"]</code>	get the data with the code <i>code</i> for nodes selected by <i>nodeselector</i>
<code>SMTNodeData[nodeselector, "code", value]</code>	set the data with the code <i>code</i> for nodes selected by <i>nodeselector</i> to be equal <i>value</i>
<code>SMTNodeData["code"]</code>	get the data with the code <i>code</i> for all nodes
<code>SMTNodeData["code", {v₁,v₂,...,v_{NoNodes}}]</code>	set the data with the code <i>code</i> 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).

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 ", { $a_1, a_2, \dots, a_{\text{NoEquations}}$ }] set the current value of all unknowns of the problem

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.

This returns the current values of all unknowns in node 5.

```
SMTNodeData[5, "at" ]
```

This adds 1 to unknowns in first 10 nodes.

```
SMTNodeData[Range[10], "at", 1 + SMTNodeData[Range[10], "at" ] ]
```

This sets all unknowns to zero in all nodes.

```
SMTNodeData["at", 0 * SMTNodeData["at" ] ]
```

This sets the shape velocity field in node 5 to {1,0}.

```
SMTNodeData[5, "sX", {1, 0} ]
```

This sets the BC velocity field in node 5 to {1,0}

```
SMTNodeData[5, "sdB", {1, 0} ]
```

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*

Get node specification data.

The values of the *code* parameter are specified in Node Specification Data. Node specification data can not be changed.

SMTElementData

SMTElementData[<i>elementselector</i> , <i>code</i>]	get the data with the code <i>code</i> for elements selected by <i>elementselect</i>
SMTElementData[<i>elementselector</i> , <i>code</i> , <i>value</i>]	set the data with the code <i>code</i> for elements selected by <i>elementselector</i> to be equal <i>value</i>
SMTElementData[" <i>code</i> "]	get the data with the code <i>code</i> for all elements

Get or set the element data.

SMTElementData[" Domain "]	
SMTElementData[<i>elementselector</i> , " Domain "]	get the domain identification <i>dID</i> for all elements or for the selection of elements
SMTElementData[" Body "]	
SMTElementData[<i>elementselector</i> , " Body "]	get the body identification <i>b ID</i> for all elements or for the selection of elements
SMTElementData[" Code "]	
SMTElementData[<i>elementselector</i> , " Code "]	get the element type <i>etype</i> for all elements or for the selection of elem
SMTElementData["All"]	
SMTElementData[<i>elementselector</i> , "All"]	get all names and values of for all elements or for the selection of eler

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 Elements).

This returns a list of nodes for 35-th element.

```
SMTElementData [ 35 , "Nodes" ]
```

SMTDomainData

SMTDomainData[<i>dID</i> , " <i>code</i> "]	get the data with the code <i>code</i> for domain <i>dID</i>
SMTDomainData[<i>dID</i> , " <i>code</i> ", <i>value</i>]	set the data with the code <i>code</i> for domain <i>dID</i> to be equal <i>value</i>
SMTDomainData[" <i>code</i> "]	get the data with the code <i>code</i> for all domains
SMTDomainData[<i>dID</i> , "All"]	get all names and values of the data for domain <i>dID</i>
SMTDomainData["DomainID"]	get domain identifications <i>dID</i> for all domains
SMTDomainData[<i>i_Integer</i> , " <i>code</i> "]	get the data with the code <i>code</i> for <i>i</i> –th domain
SMTDomainData[<i>i_Integer</i> , " <i>code</i> ", <i>value</i>]	set the data with the code <i>code</i> for <i>i</i> –th domain to be equal <i>value</i>

Get or set the element specification data.

The values of the *code* parameter are specified in Domain Specification Data.

This returns material data specified for the domain "solid".

```
SMTDomainData [ "solid" , "Data" ]
```

This sets material data specified for the domain "solid" to new values.

```
SMTDomainData["solid", "Data", {E,  $\nu$ ,  $\sigma_y$ , ...}]
```

SMTFindNodes

SMTFindNodes[*nodeselector*] The parameter *nodeselector* is used to select nodes. It has one of the forms described in section Selecting Nodes.

SMTFindElements

SMTFindElements[*elementselector*] The parameter *elementselector* is used to select elements. It has one of the forms described in section Selecting Elements.

SMTSetSolver

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

SMTSetSolver[*solverID*, *p*₁, *p*₂, ...] initialize solver with solver identification number *solverID* and a set of parameters depending on the solver type

SMTSetSolver[] reset solver structures

<i>solverID</i>	<i>Description</i>
0	appropriate solver is chosen automatically
1	standard LU profile unsymmetric solver (no user parameters defined)
2	standard LDL profile symmetric solver (no user parameters defined)
3	NOT A PART OF THE STANDARD DISTRIBUTION !!! SuperLU unsymmetric solver SMTSetSolver[4, { <i>ordering</i> , <i>work_allocation</i> }, { <i>pivot_tresh</i> , <i>fill</i> }]
4	NOT A PART OF THE STANDARD DISTRIBUTION !!! UMFPACK solver
5	PARDISO solver from INTEL MKL (manual is available at http://www.intel.com/software/products/mkl/docs/manuals.htm)

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!!.

Intel MKL Solver - PARDISO initialization

$SMTSetSolver[5,$ $mttype,\{\{pi_1,pv_1\},\{pi_2,pv_2\},\dots\}]$	initialize PARDISO solver $mttype$ -type of matrix pi_i - index of the i -th parameter in the $iparm$ vector accordingly to the PARDISO manu pv_i - value of the i -th parameter
--	--

Some of the more common parameters are given below. For a complete list of the parameters refer to the manual available at [http : // www.intel.com/software/products/mkl/docs/manuals.htm](http://www.intel.com/software/products/mkl/docs/manuals.htm).

mttype - Type of matrix

type of matrix

- 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 unsymmetric matrix (full pivoting)
 - 13 - complex and unsymmetric matrix
- Complex matrices are not yet supported!

The default matrix type depends on the element specification SMSSymmetricTangent and node identifications (Node Identification) -L and -AL.

{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 unsymmetric 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 $\epsilon = 10^{-13}$.

References

Intel® Math Kernel Library : *Reference Manual*

Shared Finite Element Libraries

SMTSetLibrary

SMTSetLibrary[*path*] sets the path to the users library and initializes the library

Initialize the library.

<i>option</i>	<i>description</i>	<i>Default</i>
"Code"	two letter code that uniquely identifies the library	"UL"
"Title"	the short name that describes the contents of the library (e.g. "large strain plasticity models")	"User Library"
"URL"	the internet address where the library will be posted	"xxx"
"Keywords"	the list of keywords used to create the elements code (see Unified Element Code)	the keywords used by the default built-in library
" Contents "	the description of the contents of the library	"xxx"
"Contact"	the contact address of the corresponding author or institution	"xxx"

Options for SMTSetLibrary command.

See also: AceShare, Simple AceShare library

SMTAddToLibrary

SMTAddToLibrary[{*key*₁,*key*₂,...}] adds the element with the unified element code (*key*₁<>*key*₂<>...) composed from the given keywords to the current library

Add elements to the library.

The keywords used to compose the element code must be the same as defined by the SMTSetLibrary (see Unified Element Code) command.

The AceGen session name used to create the element must be the same as the element code.

<i>option</i>	<i>description</i>	<i>Default</i>
"Author"	{"name","address"}	{"",""}
"Source"	the name of the notebook with the AceGen source used to create the element (the source notebook can be also automatically created by the SMSRecreateNotebook[] command)	""
"Documentation"	the name of the notebook with the documentation	""
"Examples"	the name of the notebook with the practical examples	""
"DeleteOriginal"	delete the original "Source", "Documentation" and "Examples" notebooks after they are copied to the library	False
"DescriptionTable"	2 D array included into home page (home page item)	{}
"Main"	arbitrary data stored (data can be retrived by "Main"/.SMTGetData["element code"])	□
"Data"	list of rules where arbitrary data can be stored under the keywords {data_key1->data_contents1,...} (data can be retrived by data_key/.SMTGetData["element code"])	{}
"Benchmark"	list of rules (home page item) {key1 -> "HTML source code printed into home page verbatim 1",...}	{}
"Figure"	a list of HTML form hyperlinks to the figures that appears as a part of the home page {"figure hyperlink 1",...}	{}
"Links"	a list of HTML form hyperlinks that are included into the home page {{ "link description 1", "hyperlink 1"},...}	{}

Options for SMTAddToLibrary command.

See also: AceShare, Simple AceShare library, SMTSetLibrary

SMTLibraryContents

SMTLibraryContents[] prepares the contents of the library in a way that can be posted on internet (it should be called once before posting)

See also: Simple AceShare library

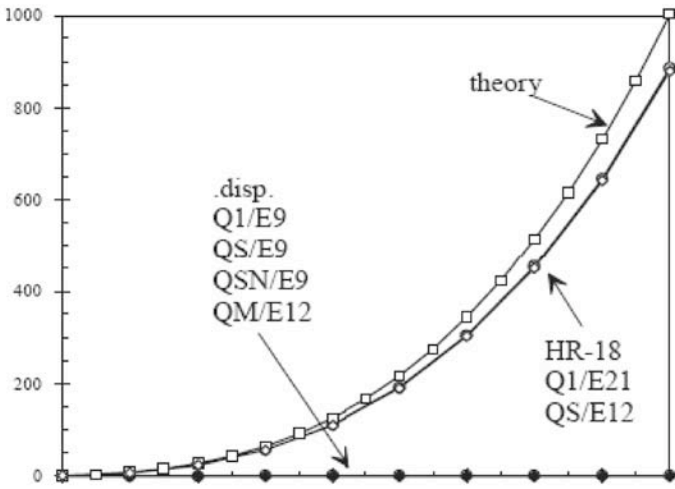
Utilities

SMTScannedDiagramToTable

SMTScannedDiagramToTable[*scanned_points*, *s0*, *s1*, *v0*, *v1*] transforms a table of points picked with *Mathematica* from the bitmap picture into the table of points in the actual coordinate system, where *s0* and *s1* are two points picked from the picture with the known coordinates *v0* and *v1*

Example

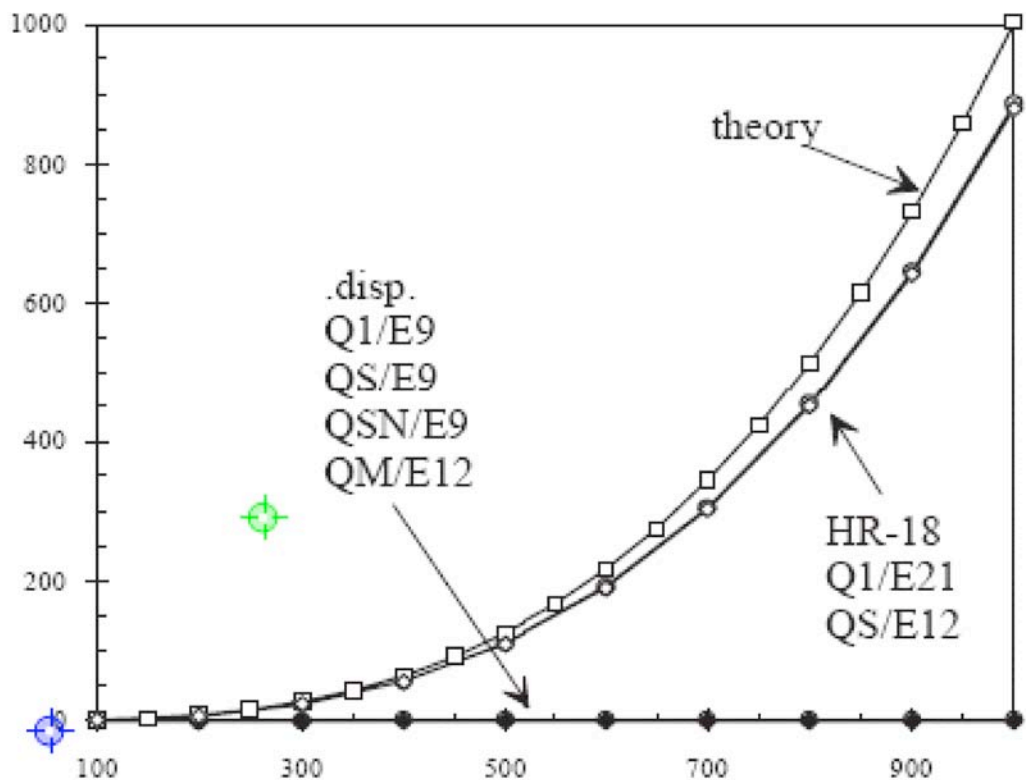
```
<< AceFEM` ;

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[


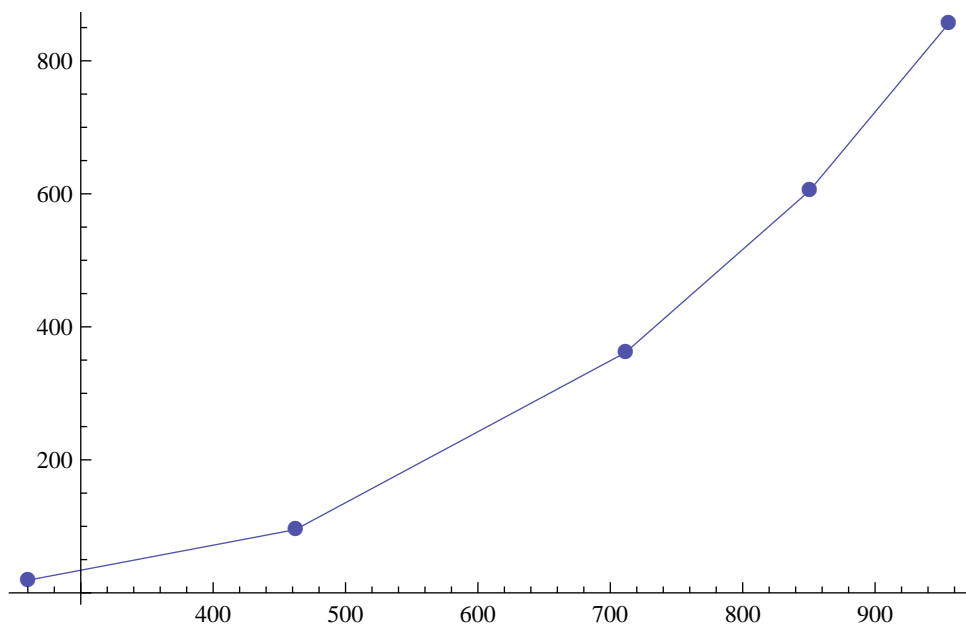
```

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, #] &]
]
}

Input: Set s0 Set s1 Input points



```
graph = SMTScannedDiagramToTable[
  Drop[pt, 2], pt[[1]], pt[[2]], {100., 0}, {1000., 1000.}]
ListLinePlot[graph, PlotMarkers -> Automatic]
{{260.058, 19.0114}, {462.099, 95.057},
 {711.37, 361.217}, {850.437, 604.563}, {955.394, 855.513}}
```



SMTPost

SMTPost[<i>pcode_String</i>]	get a vector composed of the nodal values according to element post-processing code <i>pcode</i> (CDriver specific)
SMTPost[<i>i_Integer, ncode_String</i>]	get a vector composed of the <i>i</i> -th component of the nodal quantity <i>ncode</i> ("at", "ap", "da", "st", "sp") from all nodes
SMTPost[<i>i_Integer</i>]	≡ SMTPost[<i>i</i> , "at"]

The *SMTPost* function returns the vector of *NoNodes* numbers that can be used for postprocessing by the SMT-ShowMesh function or directly in *Mathematica*.

The first form of the function 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 correspond 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.

Type 1: The integration point value is multiplied by the shape function value $f_{Ni} = w_{Ni} \sum_{j=1}^{NoIntPoints} w_{ij} f_{Gj}$ $i=1,2,\dots,NoNodes$ where f_{Gj} are an integration point values, w_{ij} are an integration point weight factors, w_{Ni} are the nodal weight factors and f_{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.

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

See also: SMTPostData, SMTPointValues .

SMTPointValues

SMTPointValues[<i>p, nv</i>]	evaluate scalar field defined by the vector of nodal values <i>nv</i> at point <i>p</i>
SMTPointValues[{ <i>p1, p2, ...</i> }, { <i>nv1, nv2, ..., nvN</i> }]	evaluate <i>N</i> -dimensional vector field defined by the vectors of nodal values <i>nv_i</i> at points <i>p_i</i>

The vectors of nodal values *nv_i* are arbitrary vectors of *NoNodes* numbers (see also SMTPost). The *SMTPointValues* function returns scalar fields evaluated at the specific points of the problem domain.

See also: SMTPostData , SMTPost

SMTMakeDll

SMTMakeDll[*source file*] create dll file

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

<i>option</i>	<i>description</i>	<i>default value</i>
"Platform"	"32" or "64" bit platform	Automatic
"OptimizeDll"	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"	True
"AdditionalSourceFiles"	list of additional source files (they are always recompiled)	{}
"AdditionalLibraries"	list of additional libraries	{}
"AdditionalObjectFiles"	list of additional object files	{}
"Debug"	pause on exit and keep all temporary files	False

Options for the SMTMakeDll function.

The dll file is created automatically by the *SMTMakeDll* function if the command line Visual studio C compiler and linker MinGW C compiler and linker are available. How to set necessary paths is described in the Install.txt file. For other C compilers, the user should write his own *SMTMakeDll* function that creates .dll file on a basis of the element source file, the sms.h header file and the SMSUtility.c file. Files can be found at the Mathematica directory \$BaseDirectory/Applications/AceFEM/Include/CDriver/).