|  |  |
| --- | --- |
| **Project Acronym:** | RLW Navigator |
| **Project Title:** | Remote Laser Welding System Navigator for Eco & Resilient Automotive Factories |

|  |  |
| --- | --- |
| **Theme FoF-ICT-2011.7:** | Digital factories: Manufacturing design and product lifecycle management |
| **Grant Agreement No:** | 285051 |
| **Work Package 3:** | RLW Process Design |
| **Task 3.2:** | Robust Fixture Design for RLW |
| **Deliverable D3.7:** | Software documentation: Fixture Optimiser description |
| **APPENDIX A** | **Variation Response Method (VRM) – USER GUIDE** |

**Responsible organization: Warwick**

|  |  |
| --- | --- |
| Document Type: | Prototype |
| Document Version: | 4.0 |
| Document Preparation Date: | 2013-11-18 |
| Document Due Date: | 2013-12-20 |

|  |  |  |
| --- | --- | --- |
| Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013) | | |
| Dissemination Level | | |
| PU | Public |  |
| PP | Restricted to other programme participants (including the Commission Services) |  |
| RE | Restricted to a group specified by the consortium (including the Commission Services) |  |
| CO | Confidential, only for members of the consortium (including the Commission Services) | X |

**Versioning and Contribution History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Organization** | **Comment** | **Date** |
| V1.0 | Warwick | Initial draft | 2012/10/19 |
| V1.1 | Warwick | Added functions for mapping | 2013/01/19 |
| V1.2 | Warwick | Added functions for sub-modelling | 2013/03/03 |
| V2.0 | Warwick | Added functions for reaction force calculation | 2013/07/25 |
| V3.0 | Warwick | First revision | 2013/09/20 |
| V3.1 | Warwick | Added additional examples | 2013/11/20 |
| V4.0 | Warwick | Final version | 2013/12/15 |

|  |  |  |
| --- | --- | --- |
| **Name** | **Organization** | **Email** |
| Dr. Pasquale Franciosa | Warwick | p.franciosa@warwick.ac.uk |
| Abhishek Das | Warwick | abhishek.das@warwick.ac.uk |
| Selim Yilmazer | Warwick | s.yilmazer@warwick.ac.uk |
| Prof. Salvatore Gerbino | UNIMOL | salvatore.gerbino@unimol.it |
|  |  |  |
|  |  |  |

# Contents

[Contents 3](#_Toc394953184)

[1 INTRODUCTION 5](#_Toc394953185)

[1.1 Goal of this document 5](#_Toc394953186)

[1.2 How to install VRM 5](#_Toc394953187)

[1.3 VRM script code 5](#_Toc394953188)

[2 INITIALISATION 7](#_Toc394953189)

[2.1 femInit 8](#_Toc394953190)

[3 FILE HANDLING 13](#_Toc394953191)

[3.1 importMesh 14](#_Toc394953192)

[3.2 exportStlFile 16](#_Toc394953193)

[3.3 exportInpFile 17](#_Toc394953194)

[3.4 exportBdfFile 18](#_Toc394953195)

[4 GEOMETRY SETTINGS 19](#_Toc394953196)

[4.1 boundary3Nodes 20](#_Toc394953197)

[4.2 createVariationalMesh 22](#_Toc394953198)

[4.3 DCT3Decomposition 23](#_Toc394953199)

[4.4 DCT3CreateGeometry 24](#_Toc394953200)

[4.5 getNormalDevPoints2Points 25](#_Toc394953201)

[4.6 meanFilterMesh 26](#_Toc394953202)

[4.7 point2Normal 28](#_Toc394953203)

[4.8 point2PointProjection 30](#_Toc394953204)

[4.9 point2PointNormalProjection 32](#_Toc394953205)

[4.10 pointNormal2PointProjection 34](#_Toc394953206)

[4.11 Set: fem.Dct 36](#_Toc394953207)

[4.12 Set: fem.Geometry 40](#_Toc394953208)

[5 BOUNDARY CONDITIONS 42](#_Toc394953209)

[5.1 Set: fem.Boundary.Constraint.Bilateral.Element 43](#_Toc394953210)

[5.2 Set: fem.Boundary.Constraint.Bilateral.Node 46](#_Toc394953211)

[5.3 Set: fem.Boundary.Constraint.PinHole 48](#_Toc394953212)

[5.4 Set: fem.Boundary.Constraint.PinSlot 50](#_Toc394953213)

[5.5 Set: fem.Boundary.Constraint.RigidLink 52](#_Toc394953214)

[5.6 Set: fem.Boundary.Constraint.Unilateral 54](#_Toc394953215)

[5.7 Set: fem.Boundary.ContactPair 58](#_Toc394953216)

[5.8 Set: fem.Boundary.DimplePair 62](#_Toc394953217)

[5.9 Set: fem.Boundary.Load.Element 65](#_Toc394953218)

[5.10 Set: fem.Boundary.Load.Node 67](#_Toc394953219)

[6 DOMAIN CONDITIONS 69](#_Toc394953220)

[6.1 Set: fem.Domain 70](#_Toc394953221)

[7 PROCESSING COMMANDS 72](#_Toc394953222)

[7.1 femLoad 73](#_Toc394953223)

[7.2 femMapping 75](#_Toc394953224)

[7.3 femPreProcessing 79](#_Toc394953225)

[7.4 femReactionForce 80](#_Toc394953226)

[7.5 femRefresh 83](#_Toc394953227)

[7.6 femReset 85](#_Toc394953228)

[7.7 femSetSelection 86](#_Toc394953229)

[7.8 femSolve 88](#_Toc394953230)

[7.9 femStore 91](#_Toc394953231)

[7.10 femSubModel 93](#_Toc394953232)

[7.11 femSwapComponents 97](#_Toc394953233)

[7.12 flipNormalComponent 98](#_Toc394953234)

[7.13 resetNormalComponent 100](#_Toc394953235)

[8 POST-PROCESSING COMMANDS 101](#_Toc394953236)

[8.1 annotationPlot 102](#_Toc394953237)

[8.2 bilateralElementBcPlot 104](#_Toc394953238)

[8.3 bilateralNodeBcPlot 106](#_Toc394953239)

[8.4 boundaryConstraintPlot 108](#_Toc394953240)

[8.5 contactPairPlot 110](#_Toc394953241)

[8.6 contourPlot 112](#_Toc394953242)

[8.7 dimpleBcPlot 115](#_Toc394953243)

[8.8 getInterpolationData 117](#_Toc394953244)

[8.9 getGapsVariable 119](#_Toc394953245)

[8.10 labelElementPlot 121](#_Toc394953246)

[8.11 labelNodePlot 123](#_Toc394953247)

[8.12 meshPlot 125](#_Toc394953248)

[8.13 meshComponentPlot 127](#_Toc394953249)

[8.14 normalElementPlot 129](#_Toc394953250)

[8.15 normalNodePlot 131](#_Toc394953251)

[8.16 pinholeBcPlot 133](#_Toc394953252)

[8.17 pinslotBcPlot 135](#_Toc394953253)

[8.18 rigilinkBcPlot 137](#_Toc394953254)

[8.19 selectPointComponent 139](#_Toc394953255)

[8.20 selectionTool 141](#_Toc394953256)

[8.21 unilateralBcPlot 143](#_Toc394953257)

# INTRODUCTION

## Goal of this document

The document intends to summarise the main features and commands of VRM toolbox and their implementation and to provide general guidelines on how to use the main capabilities of VRM.

## How to install VRM

VRM can be installed following two steps:

1. copy all source files on your computer (for example: "C:\VRM")
2. run the following lines from MatLAB script:

% define path

wdir="C:\VRM";

% add path to matlab preferences

path(path,wdir);

% initialise VRM

fem=femInit(wdir);

The main software architecture of VRM is MatLAB-based and integrates MEX-C++ routines. Indefinite linear systems are solved using the sparse UMFPACK solver developed by T. Davies (<http://www.cise.ufl.edu/research/sparse/umfpack/>), which takes advantages of multi-core CPUs capability and x64 platforms (x32 platforms are no longer supported). Definite linear systems can be handled through the CHOLMOD solver (<http://www.cise.ufl.edu/research/sparse/cholmod/).>

## VRM script code

VRM script is based on three main sections: (I) pre-processing; (II) solution; (III) post-processing. The general template for the script file is something like this:

**% PRE-PROCESSING**

***% INITIALISATION***

fem=femInit(wdir);

***% READING MESH FILE***

fem=importMesh(fem,'meshfile.inp');

***% DEFINE INPUTS:***

***% 1. DOMAIN PROPERTIES***

***% 2. BOUNDARY CONDITIONS***

***% 3. LOAD CONDITIONS***

***% ECT.***

***% GET MATRICES AND INDICES***

fem=femPreProcessing(fem);

**% SOLUTION**

***% COMPILE ALL EQUATIONS***

fem=femRefresh(fem);

***% SOLVE MODEL***

fem=femSolve(fem);

**% POST-PROCESSING**

***% 1. PLOT MESH***

***% 2. PLOT BOUNDARY CONSTRAINTS***

***% 3. PLOT GEOMETRIC FEATURES***

***% 4. CONTOUR PLOT***

***% ETC.***

# INITIALISATION

This Section describes the main commands for VRM initialisation.

## femInit

*Aim:*

Initialise "fem" structure.

*Syntax:*

fem=femInit(pwd)

*Input:*

* *pwd*: defines the location of VRM toolbox folders

*Output:*

* *fem*: fem structure

*Description:*

Initialise the main structure of VRM (both pre- and post-processing fields).

Pre-Processing fields:

* **Options**: options controlling the simulation workflow
  + **Solver**
    - **Method**: 'penalty' or 'lagrange'. It defines the method to manage constraint and boundary conditions (string)
    - **PenaltyStiffness**: penalty stiffness required for the penalty solver method (double)
    - **LinearSolver**: 'umfpack' or 'cholmod'. It defines the solver to solve linear systems (string)
    - **StoreAssembly**: is "true" than the assembly stiffness matrix is stored (true / false)
    - **MaxIter**: define the maximum number of iterations required to solve contact pairs and non-linear constraints (integer)
    - **Eps**: numerical tolerance (double)
    - **EpsCheck**: numerical tolerance used to check residuals after solving linear systems (double)
    - **CheckTol**: is "true" than the numerical solution is set to zero when the residual is less than "EpsCheck" (true / false)
  + **Eps**: global numerical tolerance (double)
  + **Max**: maximum number allowed (double)
  + **Min**: minimum number allowed (double)
  + **StiffnessUpdate**: if "true" than the stiffness matrix is re-calculated at every run (true / false)
  + **MassUpdate**: if "true" than the mass matrix is re-calculated at every run (true / false)
  + **UseActiveSelection**: if "true" than only active elements/nodes are considered (true / false)
  + **ConnectivityUpdate:** if "true" than the connectivity matrices for mesh de-noising are not re-calculated at every run (true / false)
  + **UseActiveSelection**: if "true" than only active elements/nodes are considered (true / false)
  + **GapFrame**: it define the geometry frame to calculate the gap distribution
    - “ref”: reference frame. It corresponds to the nominal geometry
    - “def”: deformed frame. It corresponds to the deformed geometry.
* **xMesh**: node coordinates and element connections
* **Geometry**: it defines the user coordinate systems
* **Denoise**: de-noise mesh options using mean filter operator
* **Dct**: DCT decomposition settings
* **Boundary**: boundary conditions
* **Domain**: domain conditions
* **Mapping**: mapping options

Solution fields:

* **Sol**: solution data

Post-processing fields:

* **Post**: post-processing options
  + **Options**: general options
    - **ParentAxes**: axes used to plot results
    - **ShowAxes**: if "true" than the 3D axis is visualised (true / false)
    - **ShowPatch**: if "true" than surfaces are shaded (true / false)
    - **ColorPatch**: define the surface colour ('g'=green; 'r'=red; 'k'=black; 'c'=cyan)
    - **ShowEdge**: if "true" edges are visualised (true / false)
    - **ColorEdge**: define the edge colour ('g'=green; 'r'=red; 'k'=black; 'c'=cyan)
    - **LengthAxis**: define the length of plotted 3D vectors (double)
    - **SymbolSize**: define the size of graphical objects (double)
    - **SubSampling**: define the percentage of 3D vectors to plot (double)
    - **LabelSize**: define the font size of 3D annotations (double)
    - **ShowProjection**: if "true" than projected points are visualised (true / false)
  + **ShowBoundary**: boundary visualisation options
    - **BilateralNode**: show bilateral constraints defined at node level (true / false)
    - **BilateralElement**:show bilateral constraints defined at element level (true / false)
    - **Unilateral**: show unilateral constraints (true / false)
    - **PinHole**: show pin-hole constraints (true / false)
    - **PinSlot**: show pin-slot constraints (true / false)
    - **RigidLink**: show rigid-link constraints (true / false)
    - **Dimple**: show dimple pairs (true / false)
    - **Contact**: show contact pairs (true / false)
  + **ShowAnnotation**: annotation options
    - **Domain**: domain identification number (integer)
    - **Node**: show node label (true / false)
    - **Element**: show element label (true / false)
    - **NormalNode**: show normal vectors of mesh nodes (true / false)
    - **NormalElement**: show normal vectors of mesh elements (true / false)
  + **Contour**: contour plot options
    - **Domain**: domain identification number (integer)
    - **ContourVariable**:variable to plot ('u'=displacement along X; 'v'=displacement along X; 'w'=displacement along Z; 'alfa'=rotation around X; 'beta'=rotation around Y; 'gamma'=rotation around Z; 'gap'=gap distribution; 'user'=user expression)
    - **ContactPair**: identification number (id) of contact pair used to plot gap distribution (integer)
    - **MaxRange**: maximum data range (double)
    - **MinRange**: minimum data range (double)
    - **Resolution**: graphical resolution (integer)
    - **Deformed**: if "true" than the deformed shape is visualised (true / false)
    - **ScaleFactor**: scaling magnitude used to plot deformed shape (double)
  + **Interp**: interpolation options
    - **Pm**: (x, y, z) coordinate of key points used for interpolation (double)
    - **Domain**: domain identification number (integer)
    - **SearchDist**: searching distance (double)
    - **InterpVariable**: variable to interpolate ('u'=displacement along X; 'v'=displacement along X; 'w'=displacement along Z; 'alfa'=rotation around X; 'beta'=rotation around Y; 'gamma'=rotation around Z; 'gap'=gap distribution; 'user'=user expression)
    - **ContactPair**: identification number (id) of contact pair used for interpolation (integer)

*Example:*

% define path

wdir="C:\VRM";

% add path to matlab preferences

path(path,wdir);

% initialise VRM

fem=femInit(wdir);

=>>fem =

Options: [1x1 struct]

xMesh: [1x1 struct]

Geometry: [1x1 struct]

Denoise: [1x1 struct]

Dct: [1x1 struct]

Mapping: [1x1 struct]

Boundary: [1x1 struct]

Domain: [1x2 struct]

Sol: [1x1 struct]

Post: [1x1 struct]

=>> fem.Options

Solver: [1x1 struct]

Eps: 1.0000e-006

Max: 1.0000e+009

Min: -1.0000e+009

StiffnessUpdate: 1

=>> fem.xMesh

Ucs: [1x858 double]

Reference: [1x858 logical]

Element: [1x768 struct]

Node: [1x1 struct]

=>> fem.Geometry

Ucs: {[3x3 double]}

=>> fem.Denoise

Options: [1x1 struct]

Tria: [1536x3 double]

Trianormal: [1536x3 double]

Connectivity: [1x1 struct]

=>> fem.Boundary

Constraint: [1x1 struct]

ContactPair: [1x1 struct]

DimplePair: [1x2 struct]

Load: [1x1 struct]

=>> fem.Domain

Element

Node

Material

Constant

Options

Load

SubModel

ElementTria

ElementQuad

=>> fem.Sol

Node2Element: {1x42510 cell}

nDoF: 255060

nLSPC: 0

nLMPC: 30

nLCt: 59

nDom: 2

Log: [1x1 struct]

Tria: [1x1 struct]

Quad: [1x1 struct]

Gauss: [1x1 struct]

Kast: [1x1 struct]

U: [255060x1 double]

res: [255060x1 double]

UserExp: []

Gap: [1x1 struct]

R: [255060x1 double]

DeformedFrame: [1x1 struct]

=>> fem.Post

Options: [1x1 struct]

ShowBoundary: [1x1 struct]

ShowAnnotation: [1x1 struct]

Contour: [1x1 struct]

Interp: [1x1 struct]

% define max. no. of iterations for iterative solver

fem.Options.Solver.MaxIter=30;

% check numerical residuals (fem.Sol.res):

fem.Options.Solver.CheckTol=true;

fem.Options.Solver.EpsCheck=1e-8;

%... if norm(fem.Sol.res) >= fem.Options.Solver.EpsCheck than fem.Sol.U=0

%.. finally, ask for stiffness matrix update:

fem.Options.StiffnessUpdate=true;

# FILE HANDLING

This Section describes the main commands for managing external files.

## importMesh

*Aim:*

Read and import mesh file. Supported formats are: ".inp", ".bdf" and ".stl". The first one is the native Abaqus mesh file, whereas the ".bdf" file has the standard Nastran 8-character fields. "Stl" is the standard file format native to the stereolithography CAD software.

*Syntax:*

fem=importMesh(fem, filename)

*Input:*

* *fem*: fem structure
* *filename*: input mesh file (string)

*Output:*

* *fem*: fem structure

*Description:*

Read and import mesh file and perform domain growing (group connected elements).

The ".inp" format has the following structure:

\*NODE

1, 149.999999987 , 49.906739943242, 3.0524272534592

2, 149.999999987 , -49.90673994343, -3.052427253471

3, 20.000000013 , 49.90673994357 , -3.052427253295

\*\*----------------

\*ELEMENT,TYPE=S4

1, 62, 63, 58, 59

2, 63, 54, 55, 58

3, 47, 48, 63, 62

4, 48, 49, 54, 63

\*\*----------------

The ".bdf" format appears:

BEGIN BULK

GRID 1 2491.739771.08471169.194

GRID 2 2490.567770.63851170.237

GRID 3 3157.465856.9986516.8795

$--------------

CQUAD4 1 0 11981 11193 11194 11189

CQUAD4 2 0 11191 11981 11189 11190

CQUAD4 3 0 11982 11196 11197 11198

CQUAD4 4 0 11199 11983 11982 11198

$--------------

ENDDATA

Supported elements are:

* **Abaqus**: S3 (TRIA 3-node) and S4 (QUAD 4-node) elements
* **Nastran**: CTRIA3 (TRIA 3-node) and CQUAD4 (QUAD 4-node) elements
* **STL**: 3-node triangular element.

*Example:*

% define path

wdir="C:\VRM";

% add path to matlab preferences

path(path,wdir);

% initialise VRM

fem=femInit(wdir);

% set filename

filename='mesh.inp';

%... import

fem=importMesh(fem,filename);

>> fem.xMesh

Ucs: [1x1342 double]

Reference: [1x1342 logical]

Element: [1x1260 struct]

Node: [1x1 struct]

## exportStlFile

*Aim:*

Write the mesh model into ASCII .stl format.

*Syntax:*

exportStlFile(filename, fem)

*Input:*

* *filename*: output .stl file (string)
* *fem*: fem structure

*Output:*

* *fem*: file

*Description:*

Export the mesh model in ASCII .stl format. Binary format is not supported yet.

*Example:*

% define path

wdir="C:\VRM";

% add path to matlab preferences

path(path,wdir);

% initialise VRM

fem=femInit(wdir);

% set filename

filename='mesh.inp';

%... import

fem=importMesh(fem,filename);

% pre-processing fem

fem=femPreProcessing(fem);

% export mesh model in ".stl" format

filename='stlfile.stl';

exportStlFile(filename,fem)

## exportInpFile

*Aim:*

Write mesh model into .inp Abaqus format.

*Syntax:*

exportInpFile(filename, fem)

*Input:*

* *filename*: output .stl file (string)
* *fem*: fem structure

*Output:*

* *fem*: file

*Description:*

Export the mesh model in .inp format.

*Example:*

% define path

wdir="C:\VRM";

% add path to matlab preferences

path(path,wdir);

% initialise VRM

fem=femInit(wdir);

% set filename

filename='mesh.inp';

%... import

fem=importMesh(fem,filename);

% pre-processing fem

fem=femPreProcessing(fem);

% export mesh model

filename='inpfile.inp';

exportInpFile(filename,fem)

## exportBdfFile

*Aim:*

Write mesh model into .bdf Nastran format.

*Syntax:*

exportBdfFile(filename, fem)

*Input:*

* *filename*: output .stl file (string)
* *fem*: fem structure

*Output:*

* *fem*: file

*Description:*

Export the mesh model in .bdf format.

*Example:*

% define path

wdir="C:\VRM;

% add path to matlab preferences

path(path,wdir);

% initialise VRM

fem=femInit(wdir);

% set filename

filename='mesh.inp';

%... import

fem=importMesh(fem,filename);

% pre-processing fem

fem=femPreProcessing(fem);

% export mesh model into bdf format

filename='bdffile.bdf';

exportBdfFile(filename,fem)

# GEOMETRY SETTINGS

This Section describes the main commands regarding geometry settings.

## boundary3Nodes

*Aim:*

Calculate nodes’ ids of a given boundary.

*Syntax:*

idbnd=boundary3Nodes(idin, element, nnode)

*Input:*

* *idin*: ids of 3 given nodes on the boundary (integer)
* *element*: element connection - 3-node TRIA elements (double)
* *nnode*: number of nodes

*Output:*

* *idbnd*: ids of nodes belonging to the given boundary

*Description:*

This command calculates the indices of all nodes belonging to a given boundary. The boundary is parsed in input through 3 consecutive nodes (*idin*). "Element" is the TRIA representation of the mesh model. It can be accessed from **fem.Denoise.Tria**. Notice that the first two selected nodes on the given boundary have to be adjacent to each-other (that is, "idin[1]" and "idin[2]" are on the same boundary edge - red edge in Figure below).

This command might be useful to automatically assign boundary load or constraints on given boundaries.



*Example:*

% import mesh

fem=importMesh(fem,filesource);

fem=femPreProcessing(fem);

% get triangular elements

element=fem.Denoise.Tria;

nnode=size(fem.xMesh.Node.Coordinate,1);

% pick initial nodes on the boundary (manual selection)

pb=[1 2 223];

% calculate boundary nodes

boundEdge=boundary3Nodes(pb, element, nnode);

Pi=fem.xMesh.Node.Coordinate(boundEdge,:);

% plot boundary points

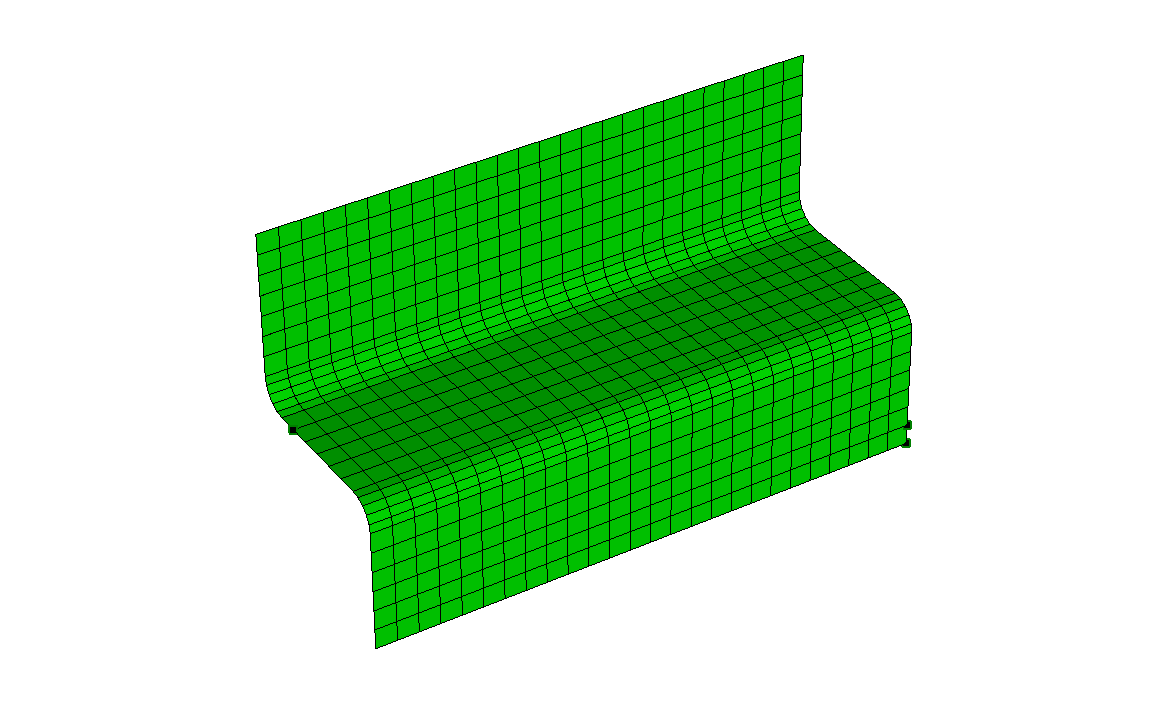
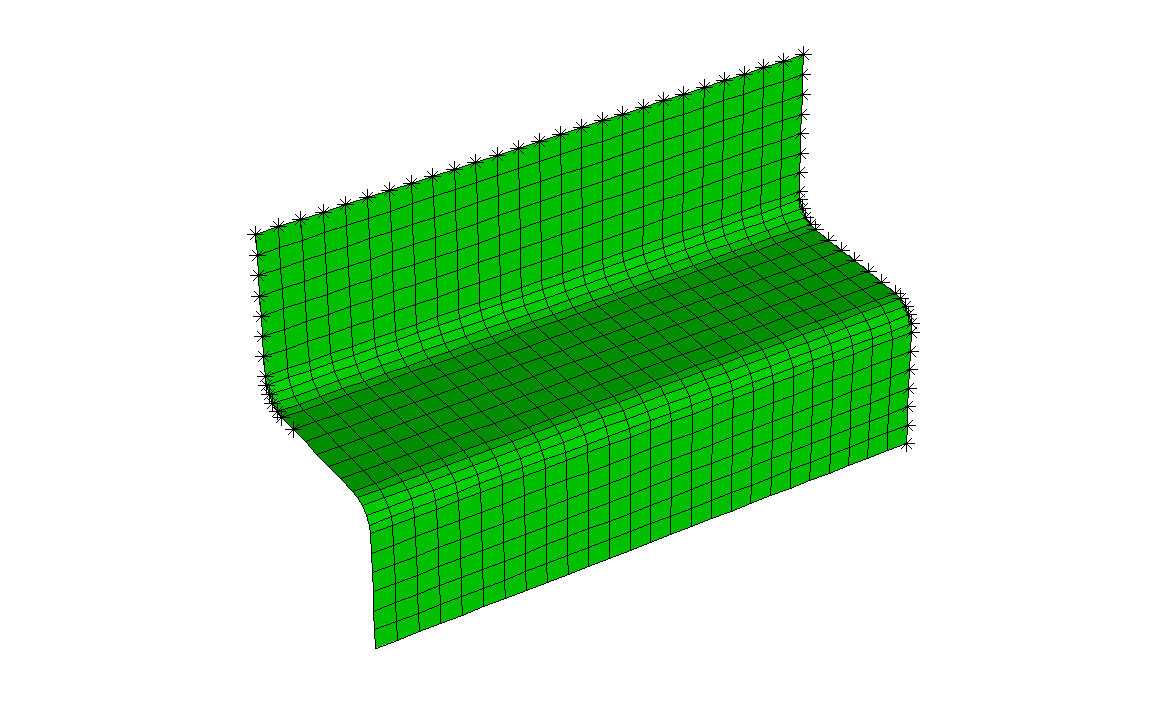
Pbi=fem.xMesh.Node.Coordinate(pb,:);

% plot mesh

meshPlot(fem)

plot3(Pi(:,1),Pi(:,2),Pi(:,3),'s')

plot3(Pbi(:,1),Pbi(:,2),Pbi(:,3),'.')

Initial boundary nodes (Pi) Recognised boundary nodes(Pbi)

## createVariationalMesh

*Aim:*

Create variational geometry.

*Syntax:*

fem=createVariationalMesh(fem, dev)

*Input:*

* *fem*: fem structure
* *dev*: deviation vector (double)

*Output:*

* *fem*: fem structure

*Description:*

This command creates variational geometry by translating mesh nodes along the normal direction at every node. The amount of variation depends on "dev".

*Example:*

% laod nominal mesh

fem=importMesh(fem,'mesh.inp');

% set properties

E=210e3;

nu=0.3;

th=1;

fem.Domain(1).Material.E=E;

fem.Domain(1).Material.ni=nu;

fem.Domain(1).Material.lamda=5/6;

fem.Domain(1).Constant.Th=th;

% add gaussian noise

nnode=size(fem.xMesh.Node.Coordinate,1);

dev=randn(nnode,1)\*1.5; % 1.5 is the standard deviation of the noise

fem=createVariationalMesh(fem, dev);

## DCT3Decomposition

*Aim:*

Run 3D DCT decomposition.

*Syntax:*

dataDCT=DCT3Decomposition(fem, cloud)

*Input:*

* *fem*: fem structure
* *cloud*: xyz coordinate of measured cloud of points (double)

*Output:*

* *dataDCT*: DCT data structure

*Description:*

This command calculates DCT modes.

*Example:*

% load cloud

filecop=’point.txt’;

cloud=importdata(filecop);

dataDCT=DCT3Decomposition(fem, cloud);

## DCT3CreateGeometry

*Aim:*

Create 3D geometry using DCT modes.

*Syntax:*

[fem, dctdata]=DCT3CreateGeometry(fem, dctdata)

*Input:*

* *fem*: fem structure
* *dataDCT*: DCT data structure

*Output:*

* *dataDCT*: DCT data structure

*Description:*

This command generate 3D geometry using DCT modes.

*Example:*

% load nominal product:

femnominalPath='femNominal';

fem=loadLargeFile(fem, femnominalPath);

% load DCT database

dctfile='dataDCT';

nd=fem.Sol.nDom;

% load DCT data structure

dataDCT=loadLargeFileDCT(dctfile, nd);

% generate variational geometry using DCT modes

femdct=fem;

[femdct,~]=DCT3CreateGeometry(femdct, dataDCT);

## getNormalDevPoints2Points

*Aim:*

Calculate normal deviations.

*Syntax:*

dev=getNormalDevPoints2Points(Pn, Nn, Pp, dsear)

*Input:*

* *Pn*: xyz coordinates of the template (double)
* *Nn*: normal vectors of the template (double)
* *Pp*: xyz coordinates of the cloud of points (double)
* *dsear*: searching distance (double)

*Output:*

* *dev*: list of deviations (double)

*Description:*

This command calculates the deviations of a cloud of points, Pp, with respect to the template, Pn. Deviations are computed along the normal vectors of the template, Nn.

*Example:*

% load template

fem=importMesh(fem,filemesh);

% import CoP

cloud=importdata(filecloud);

% get template references

Pn=fem.xMesh.Node.Coordinate;

Nn=fem.xMesh.Node.Normal;

% set searching distance

dsear=10;

% run calculation

dev=getNormalDevPoints2Points(Pn, Nn, Pp, dsear);

## meanFilterMesh

*Aim:*

Apply a mean filter on the mesh model.

*Syntax:*

fem=meanFilterMesh(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *fem*: fem structure

*Description:*

This command is useful to reduce noise of a given mesh model. The mean filter algorithm has been implemented.

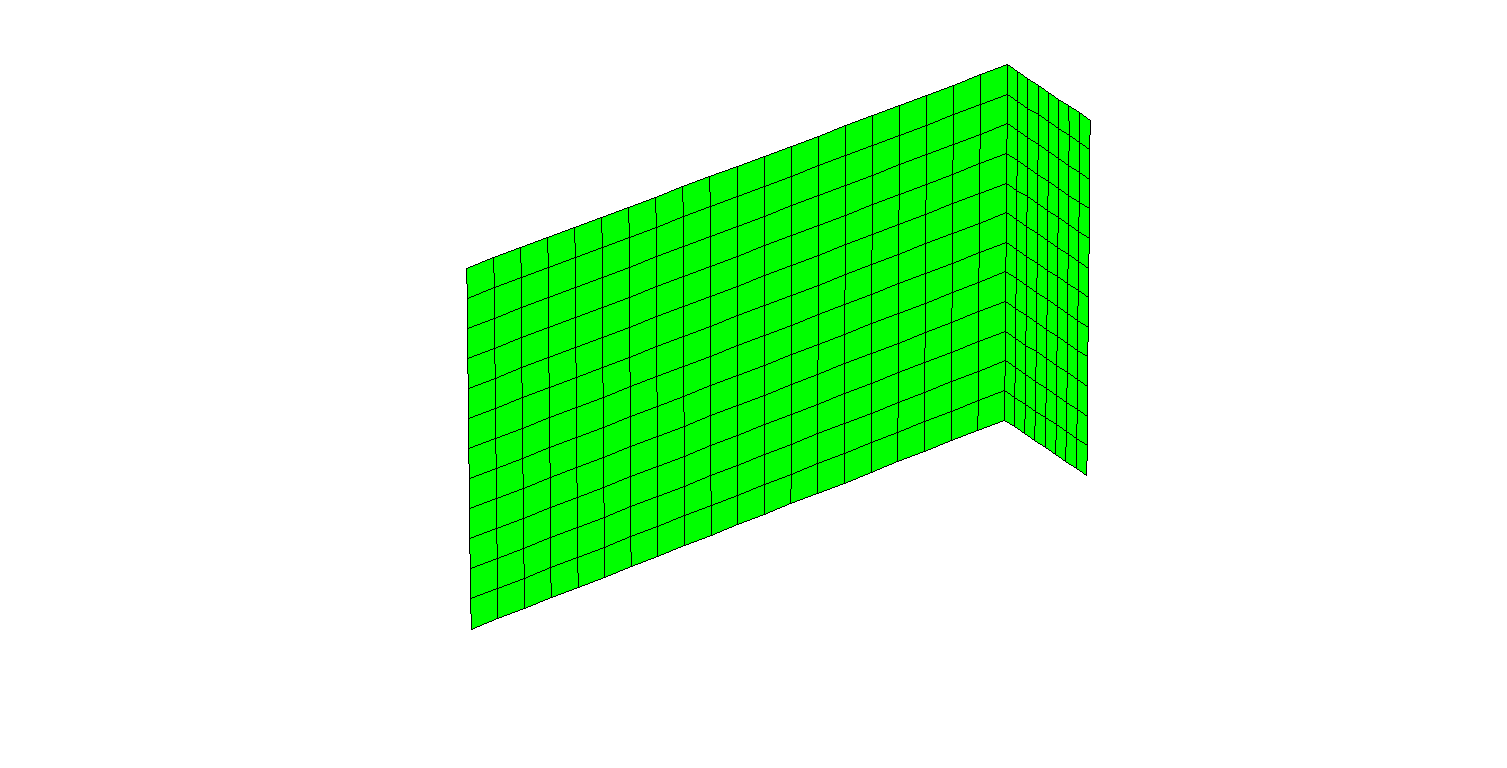
The number of iteration for mesh filtering is defined in **fem.Denoise.Options.MaxIter** (integer).

*Example:*

% plot nominal mesh

% plot

meshComponentPlot(fem,1)



**Nominal mesh**

% add gaussian noise

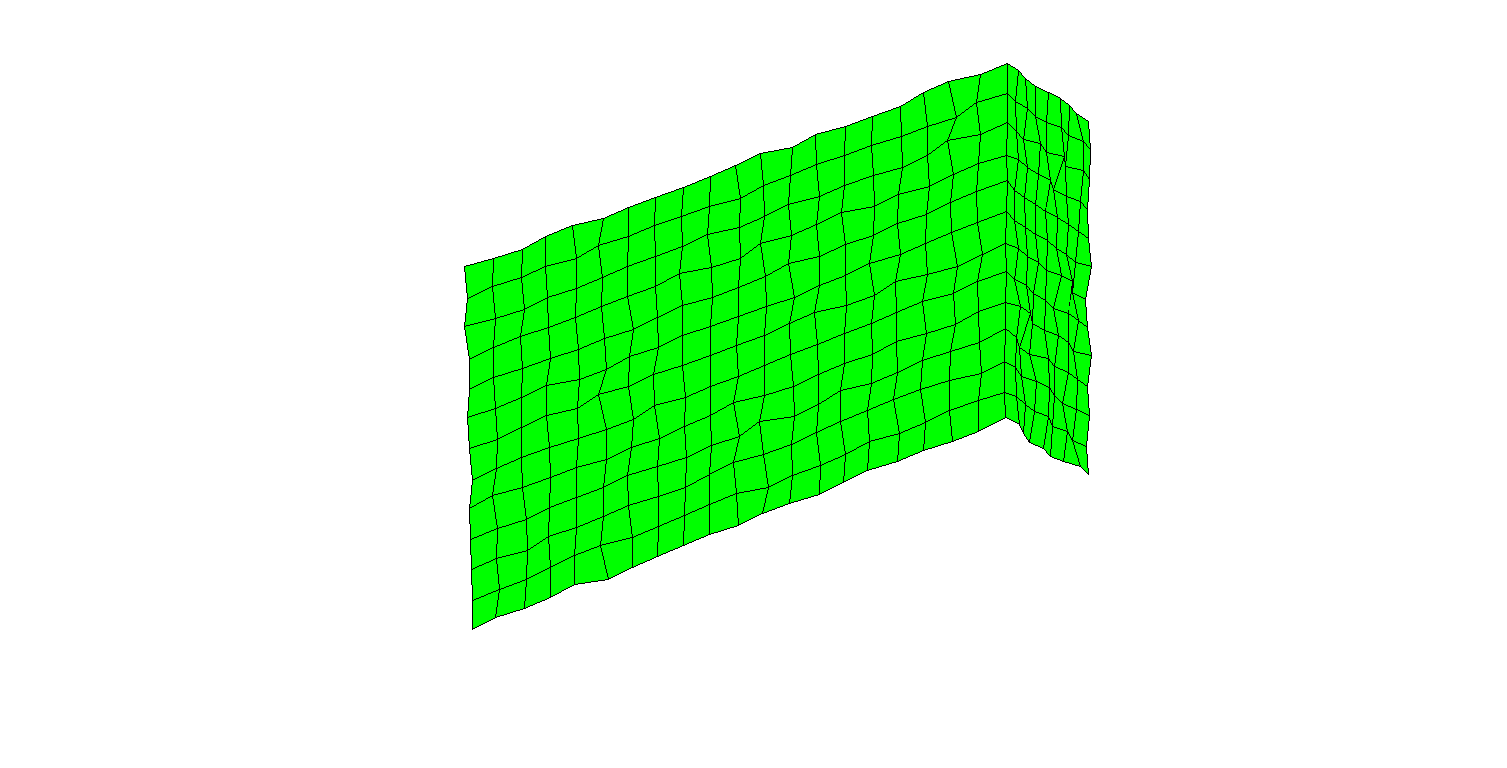
nnode=size(fem.xMesh.Node.Coordinate,1);

dev=randn(nnode,1)\*0.5; % 0.5 is the standard deviation

fem=createVariationalMesh(fem, dev);

% plot so generated geometry

meshComponentPlot(fem,1)



**Nominal mesh + noise**

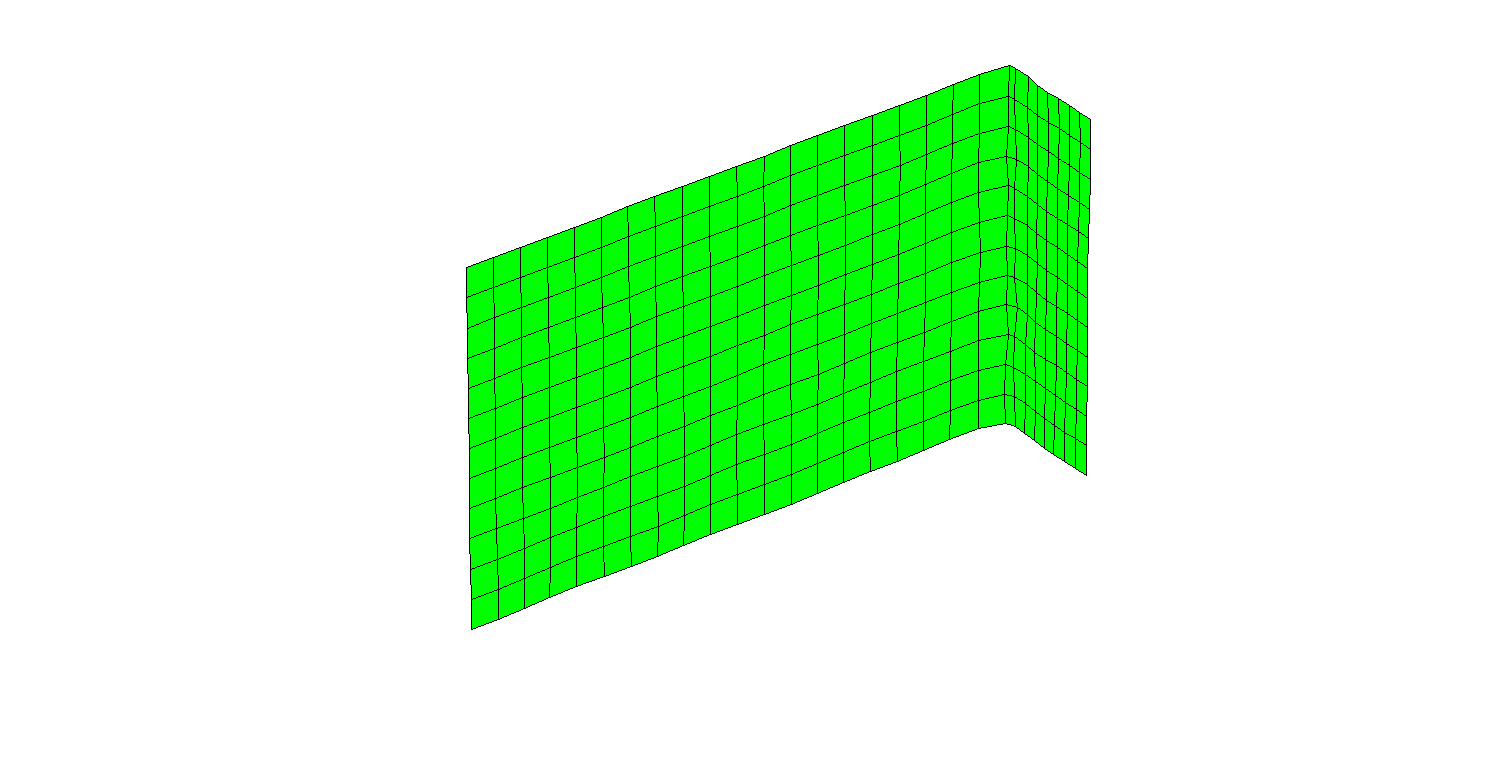
% apply filter (no. of iteration equal to 3)

fem.Denoise.Options.MaxIter=3;

fem=meanFilterMesh(fem);

% plot filtered geometry

meshComponentPlot(fem,1)



**Smoothed/filtered mesh**

## point2Normal

*Aim:*

Determine the normal vector of a given point being projected on a mesh surface.

*Syntax:*

[Nm, flag]=point2Normal(fem, Pm, idpart, seardist)

*Input:*

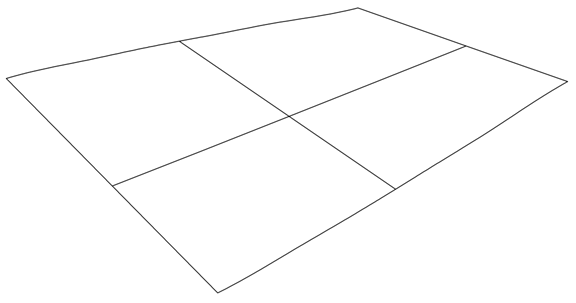
* *fem*: fem structure
* *Pm*: key points to be projected
* *idpart*: domain identification
* *searchdist*: searching distance. If the distance from the key point and the projected point (Pm,p) is greater than "searchdist" then the normal vector is not provided in output

*Output:*

* *Nm*: normal vector
* *flag*: it comes true when key point "Pm" is successfully projected on the given surface "idpart". If no projection is found than "flag" is false.

*Description:*

This command can be used to determine the normal vector corresponding to the projection of a given key point on a mesh surface.



Pm,1

Pm1,p

Nm,1

Pm,2

Pm2,p

Nm,2

*Example:*

%---

fem=femInit(wdir);

%

fem=importMesh(fem,'meshfile.inp');

%---

fem=femPreProcessing(fem);

% find normal vector for a given point

Ps=[6 0.1 0];

idpart=1;

seardist=10;

[Nm, flag]=point2Normal(fem,Ps,idpart,seardist);

## point2PointProjection

*Aim:*

Determine the projection of a given point on a given mesh surface.

*Syntax:*

[Pp, flag]=point2PointProjection(fem, Ps, idpart, seardist)

*Input:*

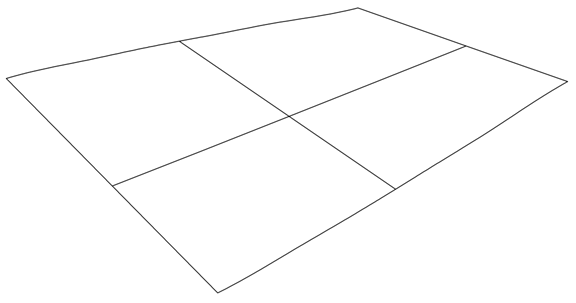
* *fem*: fem structure
* *Ps*: key points to be projected
* *idpart*: domain identification
* *searchdist:* searching distance. If the distance from the key point and the projected point (Pp) is greater than "searchdist" then the projected point is not provided in output

*Output:*

* *Pp*: projected points
* *flag*: it comes true when key point "Ps" is successfully projected on the given surface "idpart". If no projection is found than "flag" becomes false. In the latter case "Pp" is just equal to the "Ps" point.

*Description:*

This command can be used to determine the projection of a given key point on a mesh surface. The projected point is calculated along the normal vector (Nm) of the closest element to the key point Ps.



Ps,1

Pp,1

Nm,1

Ps,2

Pp,2

Nm,2

*Example:*

%---

fem=femInit(wdir);

%

fem=importMesh(fem,'meshfile.inp');

%---

fem=femPreProcessing(fem);

% create a regular grid of points being projected

[x,y]=meshgrid(0:10, -0.5:0.05:0.5);

z=ones(size(x))\*0.1;

% xyz coordinates

Ps=[x(:),y(:),z(:)];

% calculate projections

idpart=1;

seardist=10;

[Pp, flag]=point2PointProjection(fem, Ps, idpart, seardist);

% plot results

a=axes;

hold all

fem.Post.Options.ParentAxes=a;

fem.Post.Options.ShowAxes=false;

fem.Post.Options.ShowPatch=true;

fem.Post.Options.SymbolSize=1;

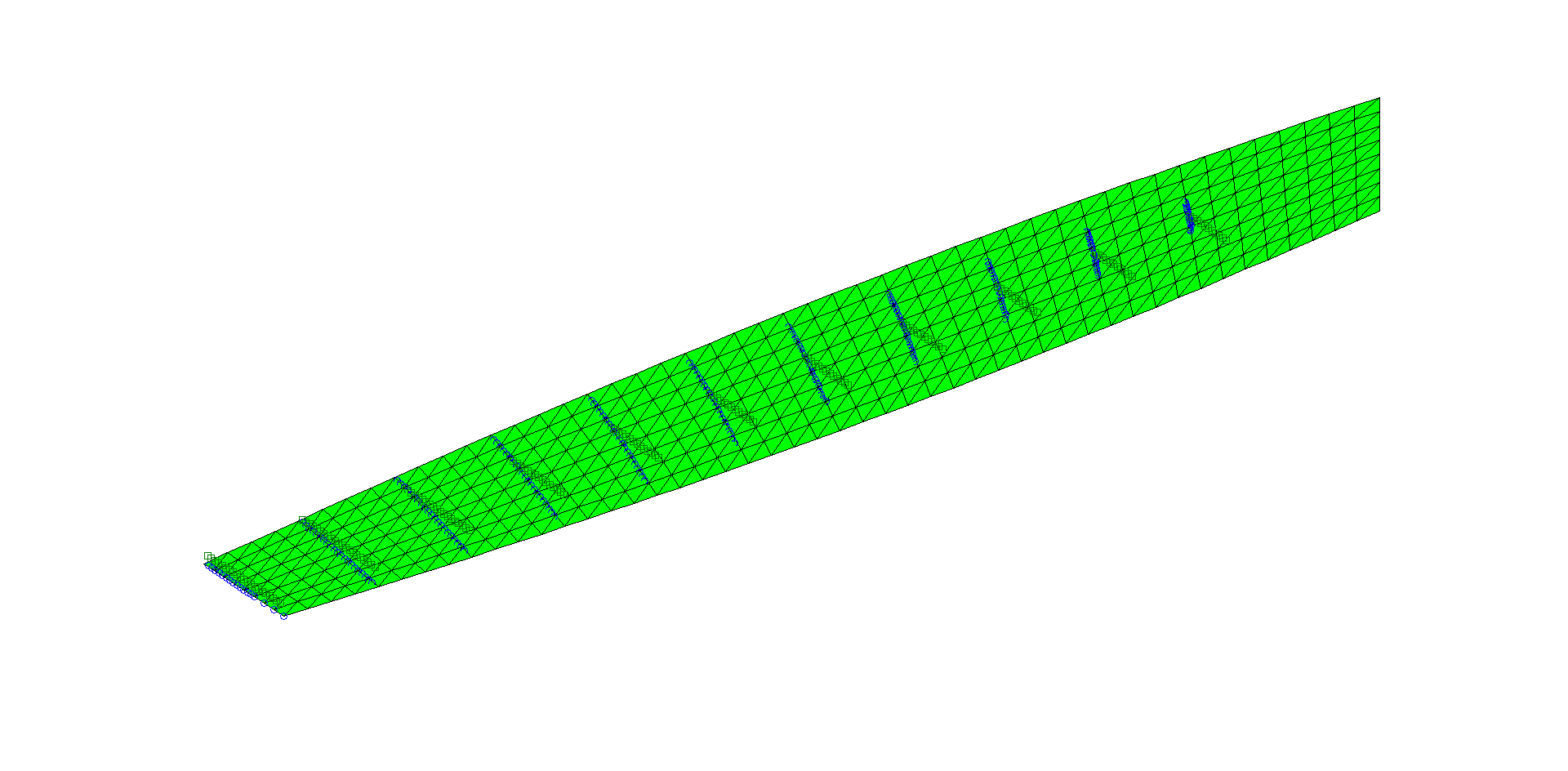
fem.Post.Options.LengthAxis=1;

meshComponentPlot(fem,1)

% plot projected points and key points

plot3(Pp(:,1),Pp(:,2),Pp(:,3),'o')

plot3(Ps(:,1),Ps(:,2),Ps(:,3),'s')

****

**Projected points on a given mesh surface**

## point2PointNormalProjection

*Aim:*

Determine the projection of a given point on a given mesh surface.

*Syntax:*

[Pp, Np, flag]=point2PointNormalProjection(fem, Ps, idpart, seardist)

*Input:*

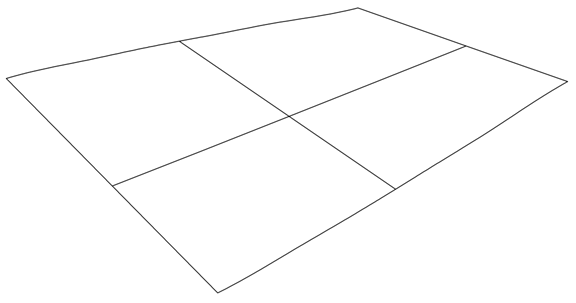
* *fem*: fem structure
* *Ps*: key points to be projected
* *idpart*: domain identification
* *searchdist*: searching distance. If the distance from the key point and the projected point (Pp) is greater than "searchdist" then the projected point is not provided in output

*Output:*

* *Pp*: projected points
* *Np*: normal vector of the projected points on the mesh surface
* *flag*: it comes true when key point "Ps" is successfully projected on the given surface "idpart". If no projection is found than "flag" becomes false. In the latter case "Pp" is just equal to the "Ps" point.

*Description:*

This command can be used to determine the projection of a given key point on a mesh surface. The projected point is calculated along the normal vector (Nm) of the closest element to the key point Ps. The, Nm corresponds to the Np vector.



Ps,1

Pp,1

Nm,1

Ps,2

Pp,2

Nm,2

*Example:*

%---

fem=femInit(wdir);

%

fem=importMesh(fem,'meshfile.inp');

%---

fem=femPreProcessing(fem);

% create a regular grid of points being projected

[x,y]=meshgrid(0:10, -0.5:0.05:0.5);

z=ones(size(x))\*0.1;

% xyz coordinates

Ps=[x(:),y(:),z(:)];

% calculate projections

idpart=1;

seardist=10;

[Pp, Np, flag]=point2PointNormalProjection(fem, Ps, idpart, seardist);

## pointNormal2PointProjection

*Aim:*

Determine the projection of a given point on a given mesh surface along a given direction.

*Syntax:*

[Pp, flag]=pointNormal2PointProjection(fem, Ps, Ns, idpart)

*Input:*

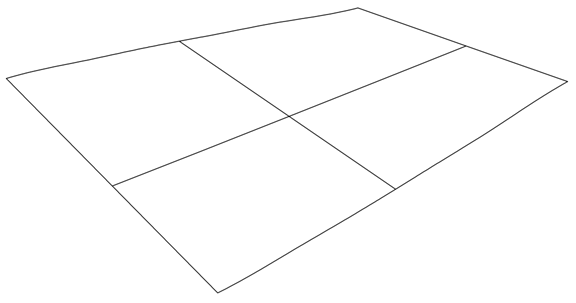
* *fem*: fem structure
* *Ps*: key points to be projected
* *Ns*: projection direction
* *idpart*: domain identification

*Output:*

* *Pp*: projected points
* *flag*: it comes true when key point "Ps" is successfully projected on the given surface "idpart". If no projection is found than "flag" becomes false. In the latter case "Pp" is just equal to the "Ps" point.

*Description:*

This command can be used to determine the projection of a given key point on a mesh surface. The projected point is calculated along the given vector Ns.



Ps,1

Pp,1

Ns

Ps,2

Pp,2

Ns

*Example:*

%---

fem=femInit(wdir);

%

fem=importMesh(fem,'meshfile.inp');

%---

fem=femPreProcessing(fem);

% create a regular grid of points being projected

[x,y]=meshgrid(0:10, -0.5:0.05:0.5);

z=ones(size(x))\*0.1;

% xyz coordinates

Ps=[x(:),y(:),z(:)];

% calculate projections

idpart=1;

Ns=[1 0 0];

[Pp, flag]=pointNormal2PointProjection(fem, Ps, Ns, idpart);

## Set: fem.Dct

*Aim:*

Define geometric features.

*Syntax:*

.Dct.Domain=domain id

.Dct.Option.Energy=energy compaction

.Dct.Option.CorrThr=correlation limit to keep

.Dct.Option.VoxX=voxel size along X

.Dct.Option.VoxY=voxel size along Y

.Dct.Option.VoxZ=voxel size along Z

.Dct.Option.VoxelPercentage=percentage for no. of voxels

.Dct.Option.VoxelSelection=1/2

.Dct.Option.ScaleBB=percentage of bb scale

.Dct.Option.SearchDist=searching distance

.Dct.Option.Offset=offset

.Dct.Option.LaplaceInterp=true/false

.Dct.Option.WeightCorrection=true/false

.Dct.Option.EnergyCoeffManual=true/false

.Dct.Option.NoEnergyCoeff=true/false

.Dct.Option.CorrCoeffManual=true/false;

.Dct.Option.NoCorrCoeff=true/false

*Input:*

* *fem*: fem structure

*Output:*

* *fem*: fem structure

*Description:*

DCT decomposition is performed based on the following inputs:

* **Domain**: it defines the domain identification id (integer).
* **Energy**: Default value for energy coefficient selection is 90%. The higher the energy compaction, the higher the number coefficients and model accuracy will increase accordingly (double).
* **CorrThr**: it defines the threshold for correlation selection. For example, 90% means that only those modes having 90% of correlation with respect to the original deviation data set are kept. Default value for correlation coefficient selection is 0.2. It will select all the coefficients whose correlation coefficients is more than 0.2. The lower the correlation threshold the higher the number of selected coefficients and higher model accuracy achieved (double).
* **VoxX/VoxY/VoxZ**: it defines the voxel size along X, Y and Z. Default value for the number of voxels are [100 100 100]. The geometry is divided into 100 equal sections in X, Y and Z directions (double).
* **VoxelPercentage**: it defines the percentage of voxel along the 3 coordinate directions. Default value for the number of voxels based upon the percentage selection is 0.5. The length (L), width (W) and height (H) of the geometry is calculated and voxel size has been calculated based as the following 0.5\*[L W H]. For example, if L=100, W=50 and H =20, then voxel size will be 0.5\*[100 50 20] = [50 25 10] (double).
* **VoxelSelection**: if it is “1” then the manual input as per **Vox/VoxY/VoxZ** is utilised. If it is “2” then the voxel percentage as per **VoxelPercentage** is used (double).
* **ScaleBB**: it defines the correction scaling factor of the bounding box. Please notice that the voxel grid is built around the bounding box frame. For flat surface one of the three dimensions of the bounding box is zero. To avoid such a singularity **ScaleBB** has to be bigger than zero. Suggested value is 0.1% (double).
* **SearchDist**: if defines the searching distance to calculate the deviation of the input cloud of points with respect to the surface mesh (double).
* **Offset**: it defines the offset between cloud of points and surface mesh (double).
* **LaplaceInterp**: Laplace interpolation is required to create a smooth model where the empty voxel elements are filled (true/false).
* **WeightCorrection**: Least square approach is applied on the modes to correct the coefficients value (true/false).
* **EnergyCoeffManual**: it defines the number of most energy compacted coefficients to include in the model (true/false).
* **NoEnergyCoeff**: If **EnergyCoeffManual** is true then “**NoEnergyCoeff**” modes are adopted (integer).
* **CorrCoeffManual**: it defines the most correlated coefficients to include in the model (true/false).
* **NoCorrCoeff**: If **CorrCoeffManual** is true then “**NoCorrCoeff**” modes are adopted (integer).

*Example:*

%--

% define user inputs

Nvox=[50 50 50];

offset=0.0;

corrlev=0.2;

fem.Dct.Option.Energy=energylev; % energy compaction

fem.Dct.Option.ScaleBB=0.001; % percentage of bb scale

fem.Dct.Option.SearchDist=10; % searching distance

fem.Dct.Option.VoxelSelection = 1; % use manual selection

fem.Dct.Option.VoxX=Nvox(1);

fem.Dct.Option.VoxY=Nvox(2);

fem.Dct.Option.VoxZ=Nvox(3);

fem.Dct.Domain=1; % domain id

fem.Dct.Option.Offset = offset;

fem.Dct.Option.CorrThr=corrlev;

fem.Dct.Option.LaplaceInterp=true;

fem.Dct.Option.WeightCorrection=false;

fem.Dct.Option.EnergyCoeffManual=true;

fem.Dct.Option.NoEnergyCoeff=10;

fem.Dct.Option.CorrCoeffManual=true;

fem.Dct.Option.NoCorrCoeff=10;

% run DCT decomposition

femdct=fem;

% no. of domains

nd=fem.Sol.nDom;

domainlist=[];

for i=1:nd

if ~isempty(dataDCT(i).BBox)

% load CoP file

filecloud=’pointfile.txt’;

cloud=importdata(filecloud);

% run DCT decomposition

dataDCT(i)=DCT3Decomposition(fem, cloud);

% generate variational geometry using DCT modes

[femdct,~]=DCT3CreateGeometry(femdct, dataDCT(i));

domainlist=[domainlist,i];

end

end

%% apply smoothing

% define options

femdct.Options.StiffnessUpdate=false;

femdct.Options.MassUpdate=false;

femdct.Options.ConnectivityUpdate=false;

femdct.Denoise.Options.MaxIter=maxsmooth;

%--

femdct=femPreProcessing(femdct);

femdct.Denoise.Options.Domain=domainlist;

femdct=meanFilterMesh(femdct);

%--

femdct=femPreProcessing(femdct);

## Set: fem.Geometry

*Aim:*

Define geometric features.

*Syntax:*

.Geometry.Ucs{id}=define user coordinate system

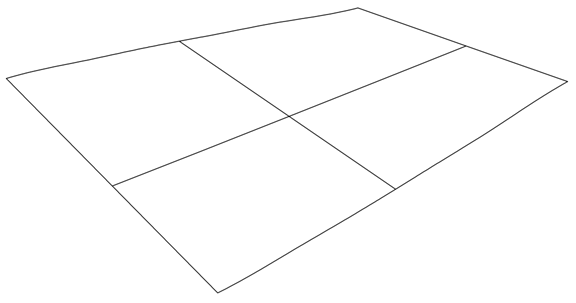
*Input:*

* *id*=UCS identification number (integer)

*Output:*

* *fem*: fem structure

*Description:*



ΩUcs

Ω0

Ω0

Xucs

Yucs

Zucs

This call can be used to define dedicated user coordinate systems. The coordinate system is expressed by means of the 3x3 rotation matrix, defining the local coordinate frame ΩUcs with respect to the global coordinate frame Ω0. User coordinate systems can be defined for any node of the mesh model.

User coordinate systems are useful to model boundary constraints acting along any direction not coincident with those of the global coordinate system.

Notice that user coordinate systems cannot be defined for those nodes employed to model unilateral constraints, contact pairs or dimple pairs. Moreover, the identification number has to be greater than one because “id=1” corresponds by default to the global reference frame.

*Example:*

% define a fixed constraint at nodes (1, 14)

fem.Boundary.Constraint.Bilateral.Node(1).Node=[1 14];

fem.Boundary.Constraint.Bilateral.Node(1).Reference='cartesian';

fem.Boundary.Constraint.Bilateral.Node(1).DoF=[1 2 3 4 5 6];

fem.Boundary.Constraint.Bilateral.Node(1).Value=[0 0 0 0 0 0];

fem.Boundary.Constraint.Bilateral.Node(1).Physic='shell';

% now add a prescribed displacement at node 9 along y (global coordinate system)

fem.Boundary.Constraint.Bilateral.Node(2).Node=9;

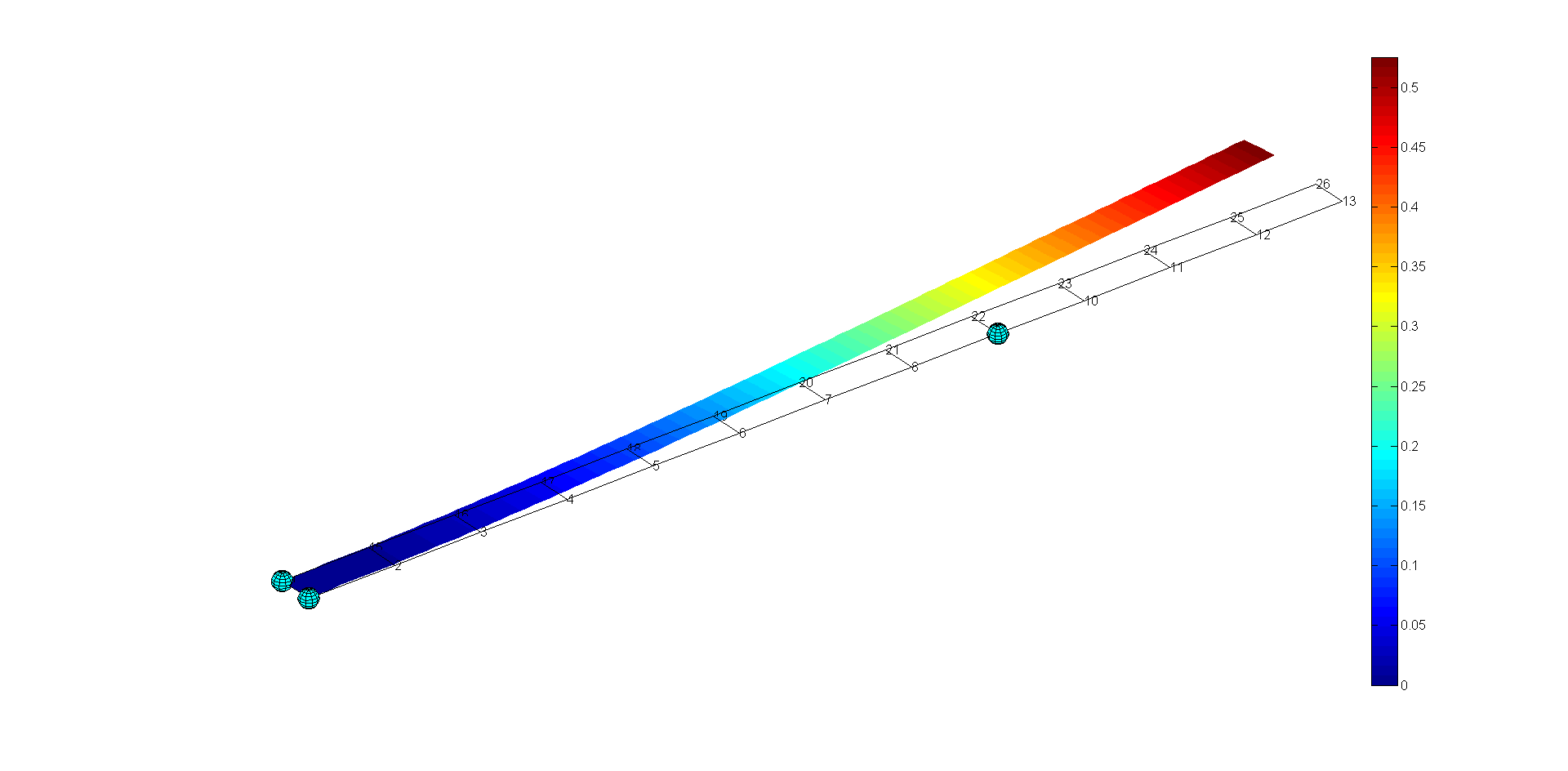
fem.Boundary.Constraint.Bilateral.Node(2).Reference='cartesian';

fem.Boundary.Constraint.Bilateral.Node(2).DoF=2;

fem.Boundary.Constraint.Bilateral.Node(2).Value=0.3;

fem.Boundary.Constraint.Bilateral.Node(2).Physic='shell';

% after running the simulation...



**DoFs of node "9" are defined with respect to the global coordinate system**

% ... the same constraint is defined with respect to a user coordinate system so that the x axis is parallel to the y axis of the global coordinate frame

fem.Boundary.Constraint.Bilateral.Node(2).Reference='cartesian';

fem.Boundary.Constraint.Bilateral.Node(2).DoF=1;

fem.Boundary.Constraint.Bilateral.Node(2).Value=0.3;

fem.Boundary.Constraint.Bilateral.Node(2).Physic='shell';

% define local UCS (3x3 rotation matrix)

R=[0 0 1

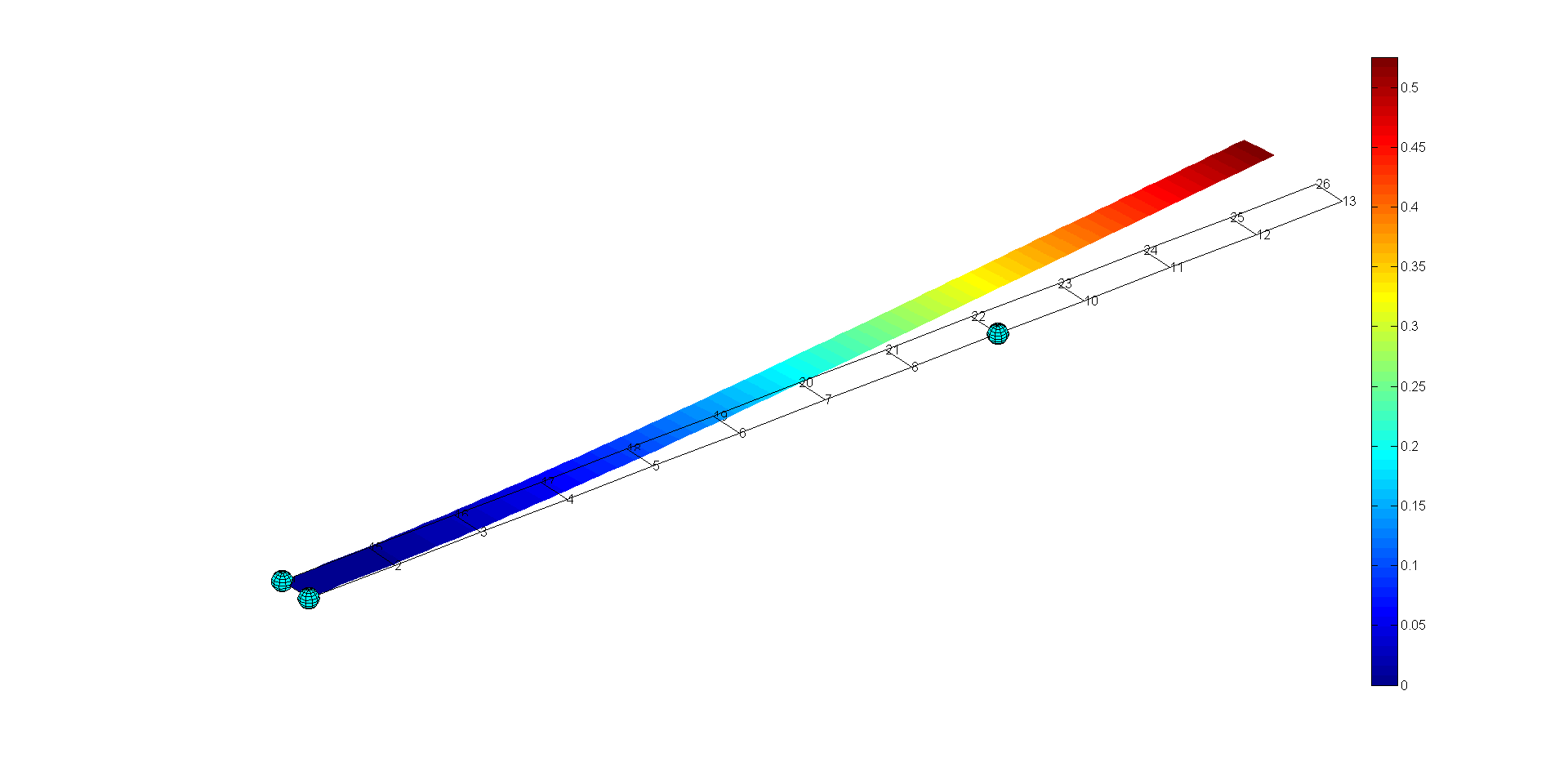
1 0 0

0 1 0];

% set UCS

fem.xMesh.Ucs(9)=2;

fem.Geometry.Ucs{2}=R; % notice that the new UCS has id=2



**DoFs of node "9" are defined with respect to the local coordinate system**

# BOUNDARY CONDITIONS

This Section describes the main options regarding boundary conditions and their options.

## Set: fem.Boundary.Constraint.Bilateral.Element

*Aim:*

Define bilateral constraints defined at element (domain) side.

*Syntax:*

.Element(id).Pm=(x, y, z) coordinates of key point to project

.Element(id).SearchDist=search distance

.Element(id).Reference="cartesian", "vectorTra", "vectorRot"

.Element(id).Nm=given unit vector

.Element(id).DoF=list of dofs

.Element(id).Value=prescribed displacement/rotation

.Element(id).Domain=domain id

.Element(id).Physic="shell"

.Element(id).UserExp.Tag=user string

.Element(id).UserExp.Matrix=user matrix

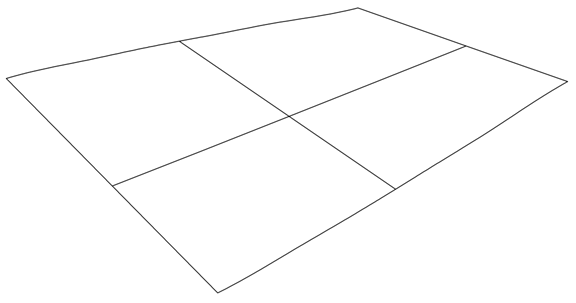
*Input:*

* *id*: constraint identification number (integer)

*Output:*

* *fem*: fem structure

*Description:*



Pm

U

Pm,d

Nm

* **Pm**: define the coordinates of the key point to project (Pm,d) on the specified domain. Hence, the constraint is applied to the projected point. Notice that this call is useful to define constraints for any key point, even not-coincident with existing mesh nodes (double).
* **SearchDist**: used to determine the projected point. If the normal distance from the key points to the specified domain is greater than "SearchDist" no constraint is assigned (double).
* **Reference**: it may contain the following options:
* *cartesian*: define constraint with respect to the cartesian coordinate frame (string)
* v*ectorTra*: define translation constraint along a given direction (defined into "Nm" field) (string)
* *vectorRot*: define rotation constraint around a given direction (defined into "Nm" field) (string).
* **Nm**: define a unit vector for the prescribed translation/rotation. This field has to be defined when "vectorTra" or "vectorRot" options are used (double).
* **DoF**: define the list of constrained degrees of freedom. The following syntax is adopted [u, v, w, α, β, γ] = [1 2 3 4 5 6]. This field has to be defined when "cartesian" option is employed (integer).
* **Value**: define the amount (U0) of displacement/rotation (double). The number of entries depends on:
* *cartesian*: [1x6] entries
* v*ectorTra*: scalar entry
* *vectorRot*: scalar entry.
* **Domain**: define domain identification number (integer).
* **Physic**: define the modelled physics. When modelling sheet-metal parts, this field has to be "shell" (string).
* **UserExp.Tag/UserExp.Matrix**: it allows to define user functions to run customised simulations (matrix double).

*Equation:*

**cartesian: U** = **U0**

**vectorTra / vectorRot: U⋅ Nm = U0**

*Example:*

%... prescribed displacement along Z (0, 0, 1) axis at point (5, 0.1, 0)

fem.Boundary.Constraint.Bilateral.Element(1).Pm=[5 0.1 0];

fem.Boundary.Constraint.Bilateral.Element(1).Reference='vectorTra';

fem.Boundary.Constraint.Bilateral.Element(1).SearchDist=5;

fem.Boundary.Constraint.Bilateral.Element(1).Nm=[0 0 1];

fem.Boundary.Constraint.Bilateral.Element(1).Value=1;

fem.Boundary.Constraint.Bilateral.Element(1).Domain=1;

fem.Boundary.Constraint.Bilateral.Element(1).Physic='shell';

% the above constraint can be also stated as:

fem.Boundary.Constraint.Bilateral.Element(1).Pm=[5 0.1 0];

fem.Boundary.Constraint.Bilateral.Element(1).Reference='cartesian';

fem.Boundary.Constraint.Bilateral.Element(1).SearchDist=5;

fem.Boundary.Constraint.Bilateral.Element(1).DoF=3;

fem.Boundary.Constraint.Bilateral.Element(1).Value=1;

fem.Boundary.Constraint.Bilateral.Element(1).Domain=1;

fem.Boundary.Constraint.Bilateral.Element(1).Physic='shell';

%... fixed constraint at point (2, 0.1, 0)

fem.Boundary.Constraint.Bilateral.Element(2).Pm=[2 0.1 0];

fem.Boundary.Constraint.Bilateral.Element(2).Reference='cartesian';

fem.Boundary.Constraint.Bilateral.Element(2).SearchDist=5;

fem.Boundary.Constraint.Bilateral.Element(2).DoF=[1 2 3 4 5 6];

fem.Boundary.Constraint.Bilateral.Element(2).Value=[0 0 0 0 0 0];

fem.Boundary.Constraint.Bilateral.Element(2).Domain=1;

fem.Boundary.Constraint.Bilateral.Element(2).Physic='shell';

## Set: fem.Boundary.Constraint.Bilateral.Node

*Aim:*

Define bilateral constraints defined at node side.

*Syntax:*

.Node(id).Node=list of constrained nodes

.Node(id).Reference="cartesian", "vectorTra", "vectorRot"

.Node(id).Nm=given unit vector

.Node(id).DoF=list of dofs

.Node(id).Value=prescribed displacement/rotation

.Node(id).Physic="shell"

.Node(id).UserExp.Tag=user string

.Node(id).UserExp.Matrix=user matrix

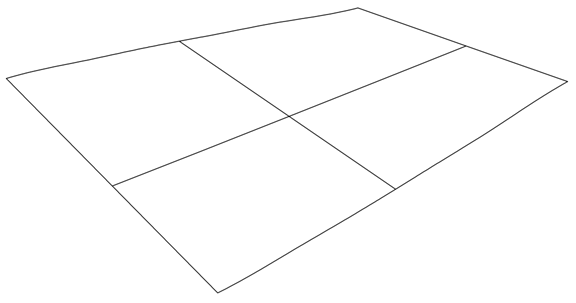
*Input:*

* *id*: constraint identification number (integer)

*Output:*

* *fem*: fem structure

*Description:*



U

Nm

* **Node**: define node identification number (integer).
* **Reference**: it may contain the following options:
* *cartesian*: define constraint with respect to the cartesian coordinate frame (string)
* v*ectorTra*: define translation constraint along a given direction (defined into "Nm" field) (string)
* *vectorRot*: define rotation constraint around a given direction (defined into "Nm" field) (string).
* **Nm**: define a unit vector for the prescribed translation/rotation. This field has to be defined when "vectorTra" or "vectorRot" options are used (double).
* **DoF**: define the list of constrained degrees of freedom. The following syntax is adopted [u, v, w, α, β, γ] = [1 2 3 4 5 6]. This field has to be defined when "cartesian" option is employed (integer).
* **Value**: define the amount (U0) of displacement/rotation (double). The number of entries depends on:
* *cartesian*: [1x6] entries
* v*ectorTra*: scalar entry
* *vectorRot*: scalar entry.
* **Physic**: define the modelled physics. When modelling sheet-metal parts, this field has to be "shell" (string).
* **UserExp.Tag/UserExp.Matrix**: it allows to define user functions to run customised simulations (matrix double).

*Equation:*

**cartesian: U** = **U0**

**vectorTra / vectorRot: U⋅ Nm = U0**

*Example:*

%... define a fixed constraint set at (1, 14) nodes

fem.Boundary.Constraint.Bilateral.Node(1).Node=[1 14];

fem.Boundary.Constraint.Bilateral.Node(1).Reference='cartesian';

fem.Boundary.Constraint.Bilateral.Node(1).DoF=[1 2 3 4 5 6];

fem.Boundary.Constraint.Bilateral.Node(1).Value=[0 0 0 0 0 0];

fem.Boundary.Constraint.Bilateral.Node(1).Physic='shell';

%... constraint along x at (4, 5, 6) nodes. Given displacement equal to "1.4"

fem.Boundary.Constraint.Bilateral.Node(1).Node=[4, 5, 6];

fem.Boundary.Constraint.Bilateral.Node(1).Reference='cartesian';

fem.Boundary.Constraint.Bilateral.Node(1).DoF=1;

fem.Boundary.Constraint.Bilateral.Node(1).Value=1.4;

fem.Boundary.Constraint.Bilateral.Node(1).Physic='shell';

% apply a constraint along the direction "V" at node "8"... that node cannot move along that direction

fem.Boundary.Constraint.Bilateral.Node(1).Node=8;

fem.Boundary.Constraint.Bilateral.Node(1).Reference='vectorTra';

fem.Boundary.Constraint.Bilateral.Node(1).Nm=V;

fem.Boundary.Constraint.Bilateral.Node(1).Value=0;

fem.Boundary.Constraint.Bilateral.Node(1).Physic='shell';

## Set: fem.Boundary.Constraint.PinHole

*Aim:*

Define pin-hole constraint

*Syntax:*

.PinHole(id).Pm=pin-hole location

.PinHole(id).Nm1=main axis

.PinHole(id).Nm2=secondary

.PinHole(id).Domain=domain identification

.PinHole(id).SearchDist=search distance

.PinHole(id).Value=constraint values (2 translations/2 rotations)

.PinHole(id).UserExp.Tag=user string

.PinHole(id).UserExp.Matrix=user matrix

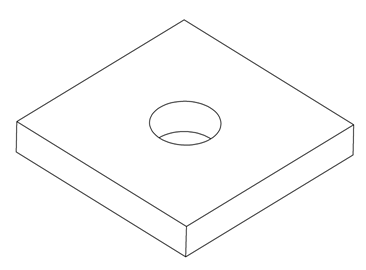
*Input:*

* *id*: identification number (integer)

*Output:*

* *fem*: fem structure

*Description:*



Nm2

Pm

Nm1

* **Pm**: define (x, y, z) coordinate of pin-hole (double).
* **Nm1**: define the main constraint axis of the pin-hole constraint (double).
* **Nm2**: define the secondary constraint axis of the pin-hole constraint (double).
* **Domain**: define the domain identification number (integer).
* **SearchDist**: searching distance (double).
* **Value**: constraint value [translation along Nm1, translation along Nm2, rotation around Nm1, rotation around Nm2] (double).
* **UserExp.Tag/UserExp.Matrix**: it allows to define user functions to run customised simulations (matrix double).

"Pin-Hole" is a macro command which can be used to model pin-hole constraints. The axis of the pin-hole joint, Nm, is calculated as cross product of Nm1 and Nm2. The rotation/translation around/along the Nm direction is free. On the contrary, the translations along Nm1 and Nm2 and the rotations around Nm1 and Nm2 are constrained

*Example:*

% define a pin-hole constraint at point (0, 0, 0) and acting along the Y axis

fem.Boundary.Constraint.PinHole(1).Pm=[0 0 0];

fem.Boundary.Constraint.PinHole(1).Nm1=[0 01];

fem.Boundary.Constraint.PinHole(1).Nm2=[10 0];

fem.Boundary.Constraint.PinHole(1).Domain=1;

fem.Boundary.Constraint.PinHole(1).SearchDist=3;

fem.Boundary.Constraint.PinHole(1).Value=[0 0 2.0\*pi/180 3.0\*pi/180]; % allow a small rotations around X

## Set: fem.Boundary.Constraint.PinSlot

*Aim:*

Define pin-slot constraint

*Syntax:*

.PinSlot(id).Pm=pin-slot location

.PinSlot(id).Nm1=main axis

.PinSlot(id).Nm2=secondary axis

.PinSlot(id).Domain=domain identification

.PinSlot(id).SearchDist=search distance

.PinSlot(id).Value=constraint values (1 traslations/1 rotations)

.PinSlot(id).UserExp.Tag=user string

.PinSlot(id).UserExp.Matrix=user matrix

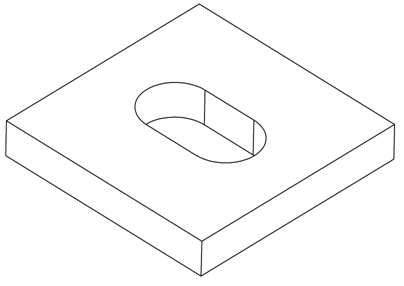
*Input:*

* *id*: identification number (integer)

*Output:*

* *fem*: fem structure

*Description:*



Nm2

Pm

Nm1

* **Pm**: define (x, y, z) coordinate of pin-slot (double).
* **Nm1**: define the main axis of the pin-slot constraint (double).
* **Nm2**: define the secondary axis of the pin-slot constraint (double).
* **Domain**: define the domain identification number (integer).
* **SearchDist**: searching distance (double).
* **Value**: constraint value [translation along Nm1, rotation around Nm2] (double).
* **UserExp.Tag/UserExp.Matrix**: it allows to define user functions to run customised simulations (matrix double).

"Pin-Slot" is a macro command which can be used to model pin-slot constraints: the translation along Nm1 and the rotation around Nm2 are constrained.

*Example:*

% define a pin-slot constraint at point (0, 0, 0) and acting along the Y axis

fem.Boundary.Constraint.PinSlot(1).Pm=[0 0 0];

fem.Boundary.Constraint.PinSlot(1).Nm1=[0 1 0]; % direction of main constraint

fem.Boundary.Constraint.PinSlot(1).Nm2=[1 0 0];

fem.Boundary.Constraint.PinSlot(1).Domain=1;

fem.Boundary.Constraint.PinSlot(1).SearchDist=3;

fem.Boundary.Constraint.PinSlot(1).Value=[0 10\*pi/180]; % allow a small rotation around Nm2

## Set: fem.Boundary.Constraint.RigidLink

*Aim:*

Define rigid link between slave and master part.

*Syntax:*

.RigidLink(id).Pm=(x, y, z) coordinates of key point to project

.RigidLink(id).SearchDist=search distance

.RigidLink(id).Nm=vector defining the link direction

.RigidLink(id).Master=master component

.RigidLink(id).Slave=slave component

.RigidLink(id).Frame= ref / def

.RigidLink(id).UserExp.Tag=user string

.RigidLink(id).UserExp.Matrix=user matrix

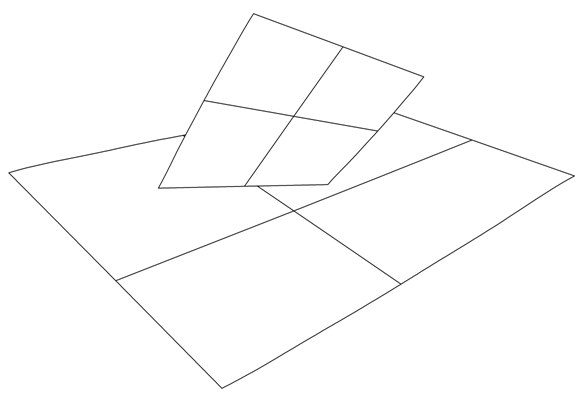
*Input:*

* *id*: constraint identification number (integer)

*Output:*

* *fem*: fem structure

*Description:*



Pm

Us

Um

Pm,s

Pm,m

Nm

* **Pm**: define the coordinates of the key point to project on the specified slave (Pm,s) and master (Pm,m) domains (double).
* **SearchDist**: used to determine the projected point. If the normal distance from the key points to the specified master and slave domains is greater than "SearchDist" no constraint is assigned (double).
* **Nm**: define a unit vector used to find the projection of key point Pm on the master and slave components (double).
* **Master**: define the master identification number (integer).
* **Slave**: define the slave identification number (integer).
* **Frame**: it defines the geometry frame used to calculate the constraint equations. Two options are available:
  + “ref”: reference frame. It corresponds to the nominal geometry
  + “def”: deformed frame. It corresponds to the deformed geometry.
* **UserExp.Tag/UserExp.Matrix**: it allows to define user functions to run customised simulations (matrix double).

This constraint can be used to simulate fastening tools (such as spot welds or rivets). Notice that all 6 DoFs are constrained.

*Equation:*

****

*Example:*

%... define a rigid link between components "1" and "2". Key point is (0, 0, 5)

Pm=[0 0 5];

V=[1 1 1];

V=V/norm(V); % normalise

fem.Boundary.Constraint.RigidLink(i).Pm=Pm;

fem.Boundary.Constraint.RigidLink(i).Nm=V;

fem.Boundary.Constraint.RigidLink(i).SearchDist=4;

fem.Boundary.Constraint.RigidLink(i).Master=2;

fem.Boundary.Constraint.RigidLink(i).Slave=1;

fem.Boundary.Constraint.RigidLink(i).Frame=’ref’;

## Set: fem.Boundary.Constraint.Unilateral

*Aim:*

Define unilateral constraint.

*Syntax:*

.Unilateral(id).Pm=(x, y, z) coordinates of key point to project

.Unilateral(id).SearchDist=search distance

.Unilateral(id).Nm=vector defining the constraint direction

.Unilateral(id).Size=true / false

.Unilateral(id).Nt=tangent direction

.Unilateral(id).SizeA=clamp size A (main axis)

.Unilateral(id).SizeB=clamp size B (secondary axis)

.Unilateral(id).Pmsize=it defines the clamp points

.Unilateral(id).Offset=offset

.Unilateral(id).Domain=domain id

.Unilateral(id).Constraint= free / lock

.Unilateral(i).Frame= ref / def

.Unilateral(i).UserExp.Tag=user string

.Unilateral(i).UserExp.Matrix=user matrix

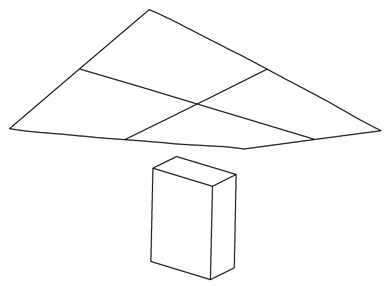
*Input:*

* *id*: constraint identification number (integer)

*Output:*

* *fem*: fem structure

*Description:*



U

Pm

Pm,d

Nm

Nt

A

B

* **Pm**: define the location of the unilateral constraint. This point is projected (Pm,d) on the specified domain along the "Nm" direction (double).
* **SearchDist**: used to determine the projected point. If the normal distance from the key points to the specified master and slave domains is greater than "SearchDist" no constraint is assigned (double).
* **Nm**: define a unit vector for the unilateral constraint direction (double).
* **Size**: if "false" than just 1-point is used to model the foot-print (true / false). If "true" than the foot-print is modelled using 5-point. The following sub-options are available:
  + **Nt**: define the orientation of the rectangular shape foot-print (double).
  + **SizeA**: define the main size of the foot-print along the "Nt" direction (double).
  + **SizeB**: define the secondary size of the foot-print (double).
* **Pmsize**: it collects the xyz coordinates of the clamp points according to the clamp size (1-point or 5-point) (double).
  + if “size=false” then clamp point is “Pm”.
  + If “size=true” and “Pmsize” is not empty, then clamp point is Pmsize.
  + If “size=true” and “Pmsize” is empty, then clamp points are calculated automatically depending on the “Nt” and “A/B” user inputs.
* **Offset**: define an initial gap between the constraint location and the specified domain. For example, if Pm and Pm,d are initially coincident, than "Offset" specifies an additional gap along the Nm direction. When no initial gap is provided than "Offset" is equal to zero (double).
* **Domain**: define the domain identification number (integer).
* **Constraint**: if "free" than the constraint is treated as unilateral; if "lock" than the constraint is treated as bilateral. In this latter case the point Pm,d is constrained to be coincident with the fixed Pm point (string).
* **Frame**: it define the geometry frame used to calculate the constraint equations. Two options are available:
  + “ref”: reference frame. It corresponds to the nominal geometry
  + “def”: deformed frame. It corresponds to the deformed geometry.
* **UserExp.Tag/UserExp.Matrix**: it allows to define user functions to run customised simulations (matrix double).

This constraint can be used to simulate clamps or NC-blocks.

*Equation:*

**Free: **

**Lock: **

*Example:*

%... unilateral constraint acting along Z axis and applied at (3, 0.1, -0.1).

% lock option

fem.Boundary.Constraint.Unilateral(1).Pm=[3 0.1 -0.1];

fem.Boundary.Constraint.Unilateral(1).SearchDist=3;

fem.Boundary.Constraint.Unilateral(1).Nm=[0 0 1];

fem.Boundary.Constraint.Unilateral(1).Size=false;

fem.Boundary.Constraint.Unilateral(1).Offset=0;

fem.Boundary.Constraint.Unilateral(1).Domain=1;

fem.Boundary.Constraint.Unilateral(1).Constraint='lock';

fem.Boundary.Constraint.Unilateral(1).Frame='ref';

% free option

fem.Boundary.Constraint.Unilateral(2).Pm=[5 0.1 -0.1];

fem.Boundary.Constraint.Unilateral(2).SearchDist=3;

fem.Boundary.Constraint.Unilateral(2).Nm=[0 0 -1];

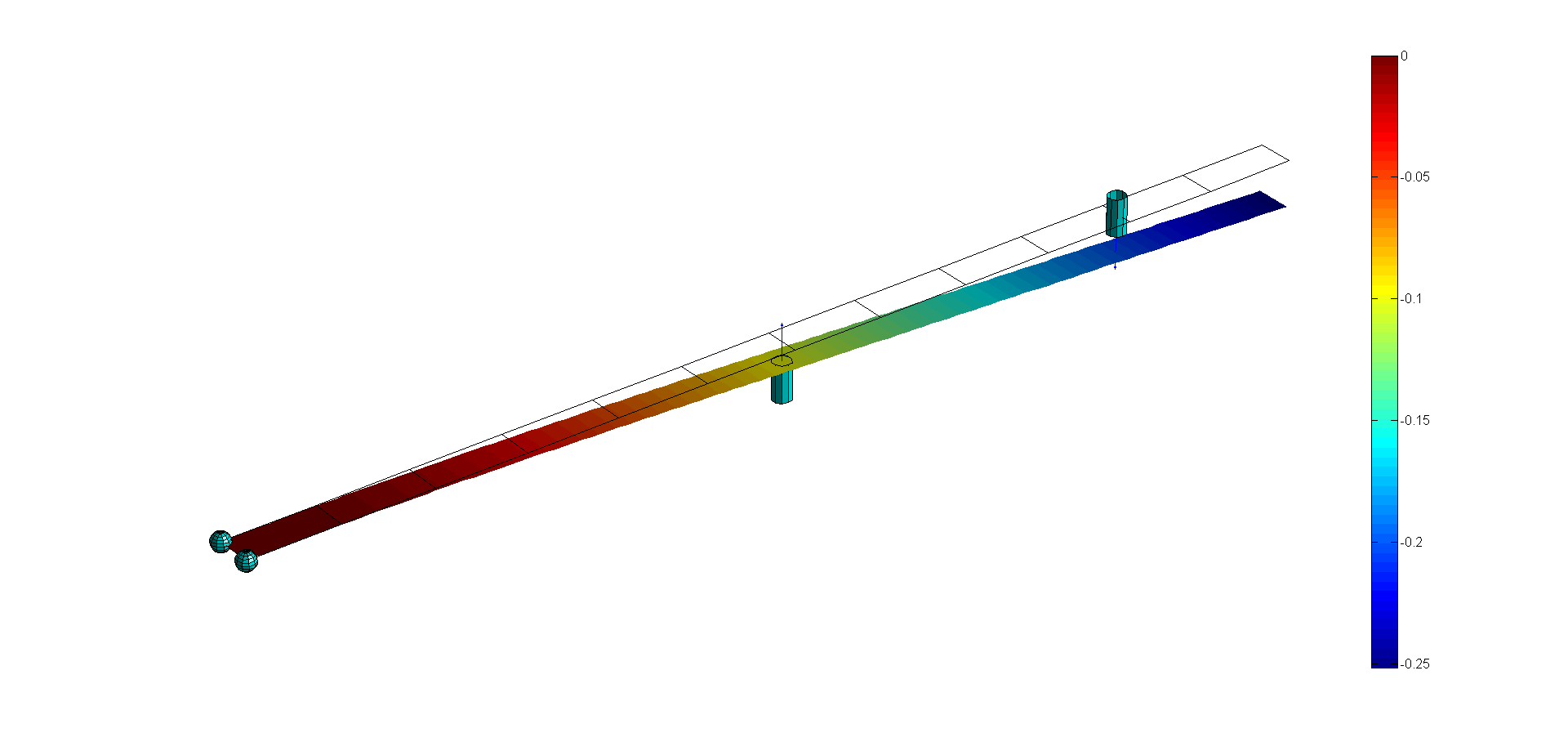
fem.Boundary.Constraint.Unilateral(2).Size=false;

fem.Boundary.Constraint.Unilateral(2).Offset=0;

fem.Boundary.Constraint.Unilateral(2).Domain=1;

fem.Boundary.Constraint.Unilateral(2).Constraint='free';

% Notice that no boundary or domain load has been applied!



**Constraint 1 (left)=lock; Constraint 2 (right)=free**

% now, the constraint id=2 (that one on the right) is assumed lock as well:

fem.Boundary.Constraint.Unilateral(2).Pm=[5 0.1 -0.1];

fem.Boundary.Constraint.Unilateral(2).SearchDist=3;

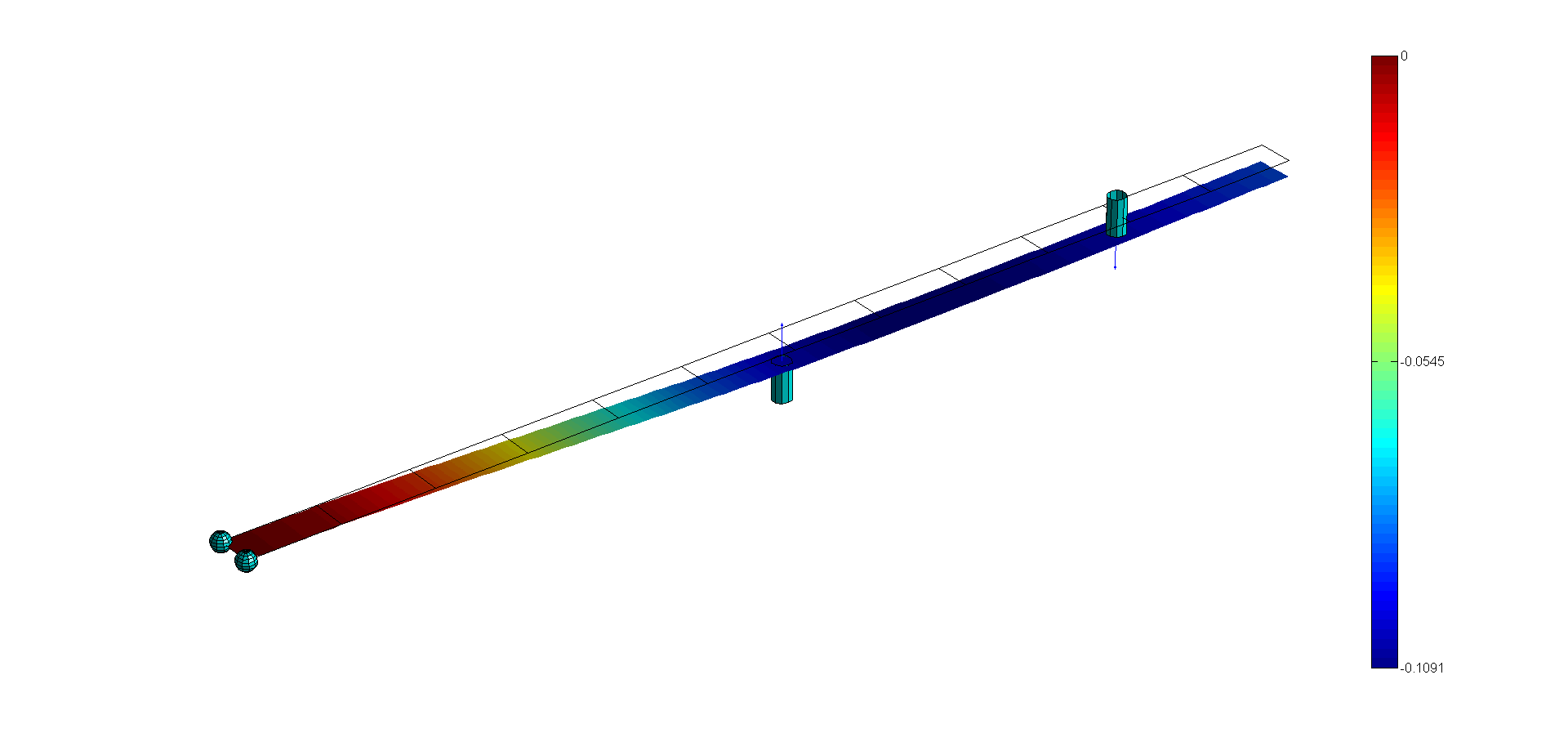
fem.Boundary.Constraint.Unilateral(2).Nm=[0 0 -1];

fem.Boundary.Constraint.Unilateral(2).Size=false;

fem.Boundary.Constraint.Unilateral(2).Offset=0;

fem.Boundary.Constraint.Unilateral(2).Domain=1;

fem.Boundary.Constraint.Unilateral(2).Constraint='lock';



**Constraint 1 (left)=lock; Constraint 2 (right)=lock**

% finally, an "offset=-0.1" is applied at constraint id=2:

fem.Boundary.Constraint.Unilateral(2).Pm=[5 0.1 -0.1];

fem.Boundary.Constraint.Unilateral(2).SearchDist=3;

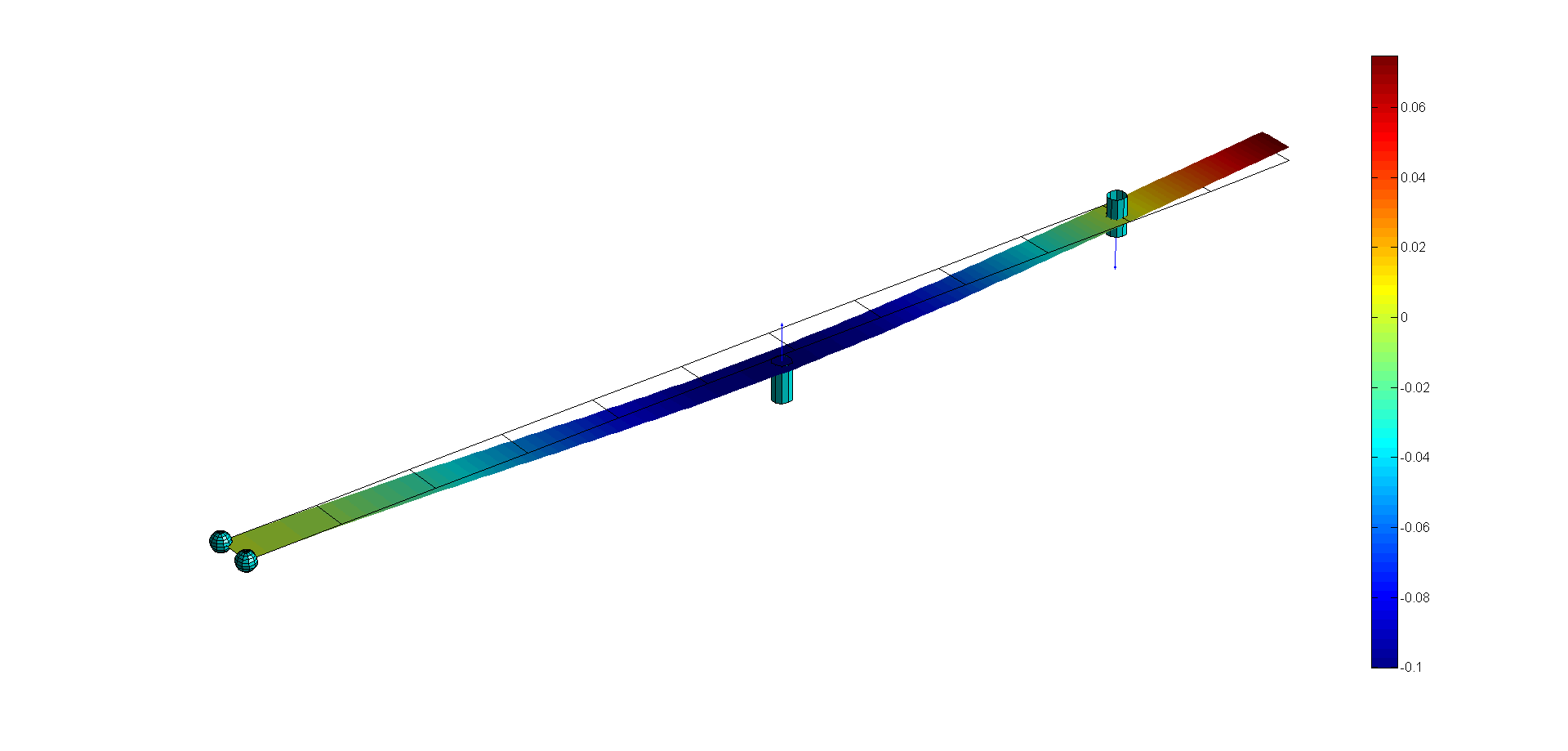
fem.Boundary.Constraint.Unilateral(2).Nm=[0 0 -1];

fem.Boundary.Constraint.Unilateral(2).Size=false;

fem.Boundary.Constraint.Unilateral(2).Offset=-0.1;

fem.Boundary.Constraint.Unilateral(2).Domain=1;

fem.Boundary.Constraint.Unilateral(2).Constraint='lock';



**Constraint 1 (left)=lock; Constraint 2 (right)=lock - offset=-0.1**

## Set: fem.Boundary.ContactPair

*Aim:*

Define contact pair between slave and master part.

*Syntax:*

.ContactPair(id).Master=master component

.ContactPair(id).MasterFlip=true / false

.ContactPair(id).Slave=slave component

.ContactPair(id).SearchDist=search distance

.ContactPair(id).Offset=offset

.ContactPair(id).Physic=shell

.ContactPair(id).Enable=true / false

.ContactPair(id).Frame=ref / def

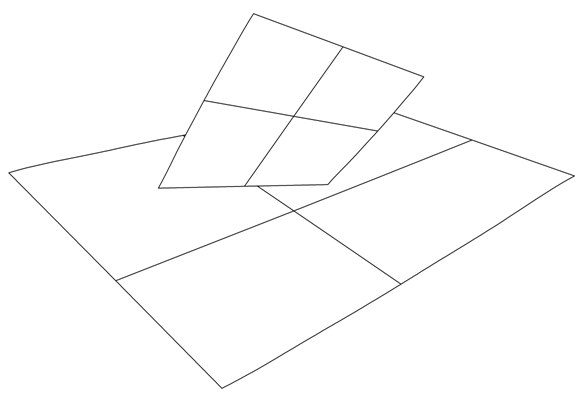
*Input:*

* *id*: contact pair identification number (integer)

*Output:*

* *fem*: fem structure

*Description:*



Us

Ps

Pm

Nm

Um

* **Master**: define the identification number of the master domain (integer).
* **MasterFlip**: if "true" than the normal vector (Nm) of every element belonging to the master domain is flipped (true / false).
* **Slave**: define the identification number of the slave domain (integer).
* **SearchDist**: used to determine the projected point of every point (Ps) belonging to the slave component. If the distance from Ps to the projected point on the master domain (Pm) is greater than "SearchDist" no contact pair is assigned (double).
* **Offset**: define an initial gap for the contact pair. For example, if Ps and Pm are initially coincident, than "Offset" specifies an additional gap along the Nm direction. When no initial gap is provided than "Offset" is equal to zero (double).
* **Physic**: define the modelled physics. When modelling sheet-metal parts, this field has to be "shell" (string).
* **Enable**: if "false" than the contact pair is used only for post-processing purpose. If "true" contact pairs are solved for to avoid slave-to-master penetration. When one is aimed to calculate only the gap distribution, without solving part-to-part penetration, this option can be "false" (true / false).
* **Frame**: it define the geometry frame used to calculate the constraint equations. Two options are available:
  + “ref”: reference frame. It corresponds to the nominal geometry
  + “def”: deformed frame. It corresponds to the deformed geometry.

When working with contact pairs the following guidelines should be followed up:

* The slave component should be the softest one.
* The mean mesh size on the slave component should be half of that one on the master domain.

*Equation:*

****

*Example:*

% define contact pair between component "1" and "2"

fem.Boundary.ContactPair(1).Master=2;

fem.Boundary.ContactPair(1).MasterFlip=false;

fem.Boundary.ContactPair(1).Slave=1;

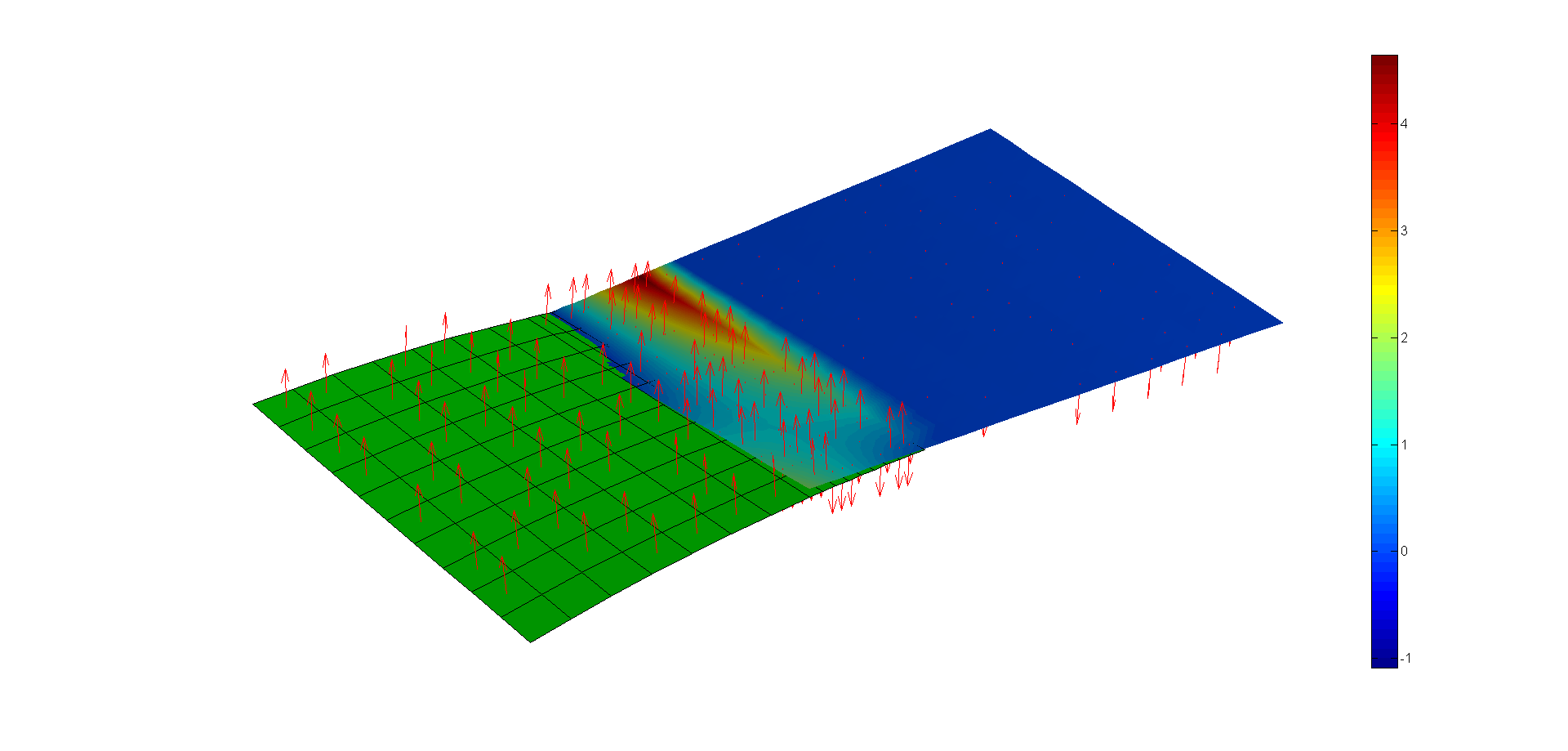
fem.Boundary.ContactPair(1).SearchDist=6;

fem.Boundary.ContactPair(1).Offset=0.0;

fem.Boundary.ContactPair(1).Physic='shell';

fem.Boundary.ContactPair(1).Enable=false;

fem.Boundary.ContactPair(1).Frame=’ref’;



**Parts penetrate each-other (initial configuration)**

% Since "Enable=false" the gap distribution is calculated without solving part-to-part penetration (negative gap is observed)

% ... now, "Enable=true"

fem.Boundary.ContactPair(1).Master=2;

fem.Boundary.ContactPair(1).MasterFlip=false;

fem.Boundary.ContactPair(1).Slave=1;

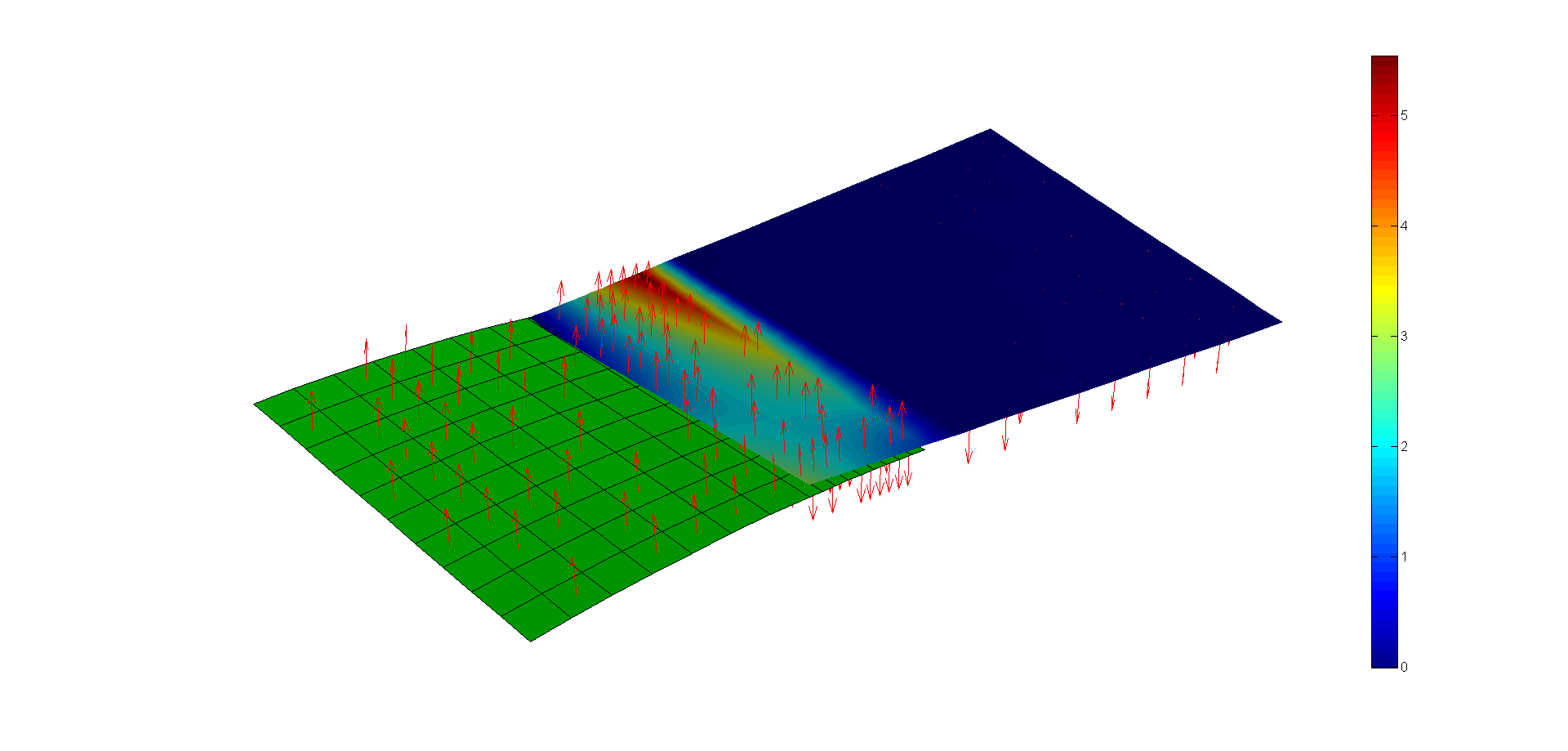
fem.Boundary.ContactPair(1).SearchDist=6;

fem.Boundary.ContactPair(1).Offset=0.0;

fem.Boundary.ContactPair(1).Physic='shell';

fem.Boundary.ContactPair(1).Enable=true;

fem.Boundary.ContactPair(1).Frame=’ref’;



**Parts do not penetrate each-other**

% use "Offset=3". The searching distance is equal to 20 to find all candidate contact pairs

fem.Boundary.ContactPair(1).Master=2;

fem.Boundary.ContactPair(1).MasterFlip=false;

fem.Boundary.ContactPair(1).Slave=1;

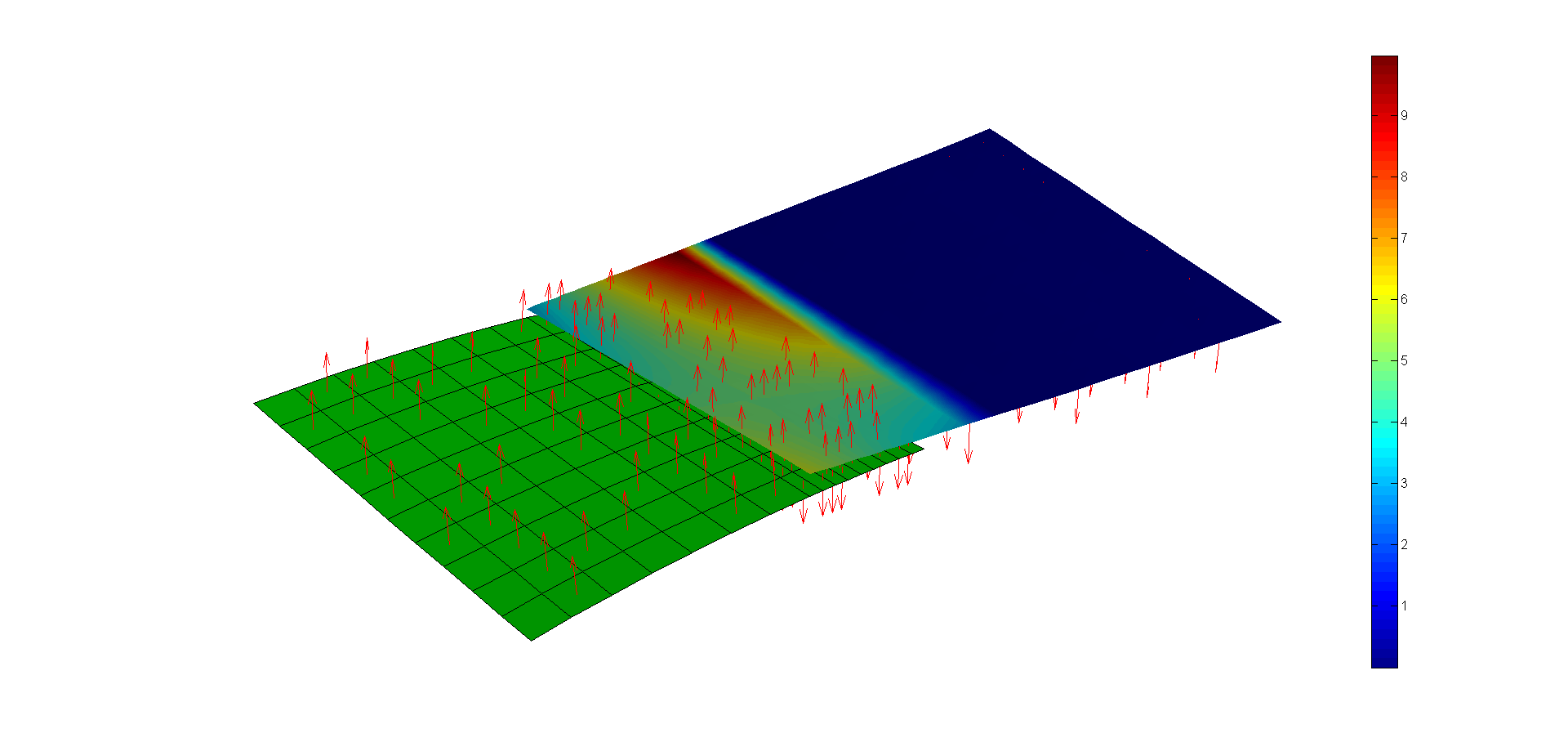
fem.Boundary.ContactPair(1).SearchDist=20;

fem.Boundary.ContactPair(1).Offset=3.0;

fem.Boundary.ContactPair(1).Physic='shell';

fem.Boundary.ContactPair(1).Enable=true;

fem.Boundary.ContactPair(1).Frame=’ref’;



**Gap distribution using "Offset=3"**

## Set: fem.Boundary.DimplePair

*Aim:*

Define dimple pair between slave and master part.

*Syntax:*

.DimplePair(id).Pm=(x, y, z) location of dimple

.DimplePair(id).Master=master component

.DimplePair(id).MasterFlip=true / false

.DimplePair(id).Slave=slave component

.DimplePair(id).SearchDist=search distance

.DimplePair(id).Offset=dimple offset

.DimplePair(id).Height=dimple height

.DimplePair(id).Physic=shell

.DimplePair(id).Frame=ref / def

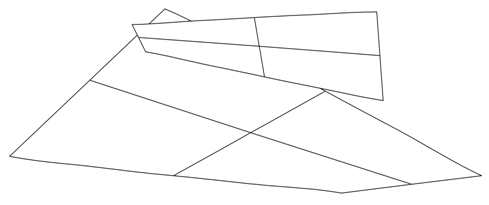
*Input:*

* *id*: dimple pair identification number (integer)

*Output:*

* *fem*: fem structure

*Description:*



Us

Ps,p

Pm,p

Nm

Um

Pm

Ps

Δd

Nm

* **Pm**: define the location of the dimple. This point is projected on the specified slave (Ps,p) and master (Pm,p) domains along the Nm direction. Nm is the unit normal vector to the master domain (double).
* **Master**: define the identification number of the master domain (integer).
* **MasterFlip**: if "true" than the normal vector (Nm) of every element belonging to the master domain is flipped (true / false).
* **Slave**: define the identification number of the slave domain (integer).
* **SearchDist**: used to determine the projected points (Ps,p and Pm,p). If the distance from Pm to Ps,p (or to Pm,p) is greater than "SearchDist" no dimple is assigned.
* **Height**: define the dimple height (Δd) (double).
* **Offset**: define an initial gap for the dimple pair. For example, if Ps and Pm are initially coincident, than "Offset" specifies an additional gap along the Nm direction. When no initial gap is provided than "Offset" is equal to zero (double).
* **Physic**: define the modelled physics. When modelling sheet-metal parts, this field has to be "shell" (string).
* **Frame**: it define the geometry frame used to calculate the constraint equations. Two options are available:
  + “ref”: reference frame. It corresponds to the nominal geometry
  + “def”: deformed frame. It corresponds to the deformed geometry.

*Equation:*

****

*Example:*

% define contact pairs:

fem.Boundary.ContactPair(1).Master=2;

fem.Boundary.ContactPair(1).MasterFlip=false;

fem.Boundary.ContactPair(1).Slave=1;

fem.Boundary.ContactPair(1).SearchDist=6;

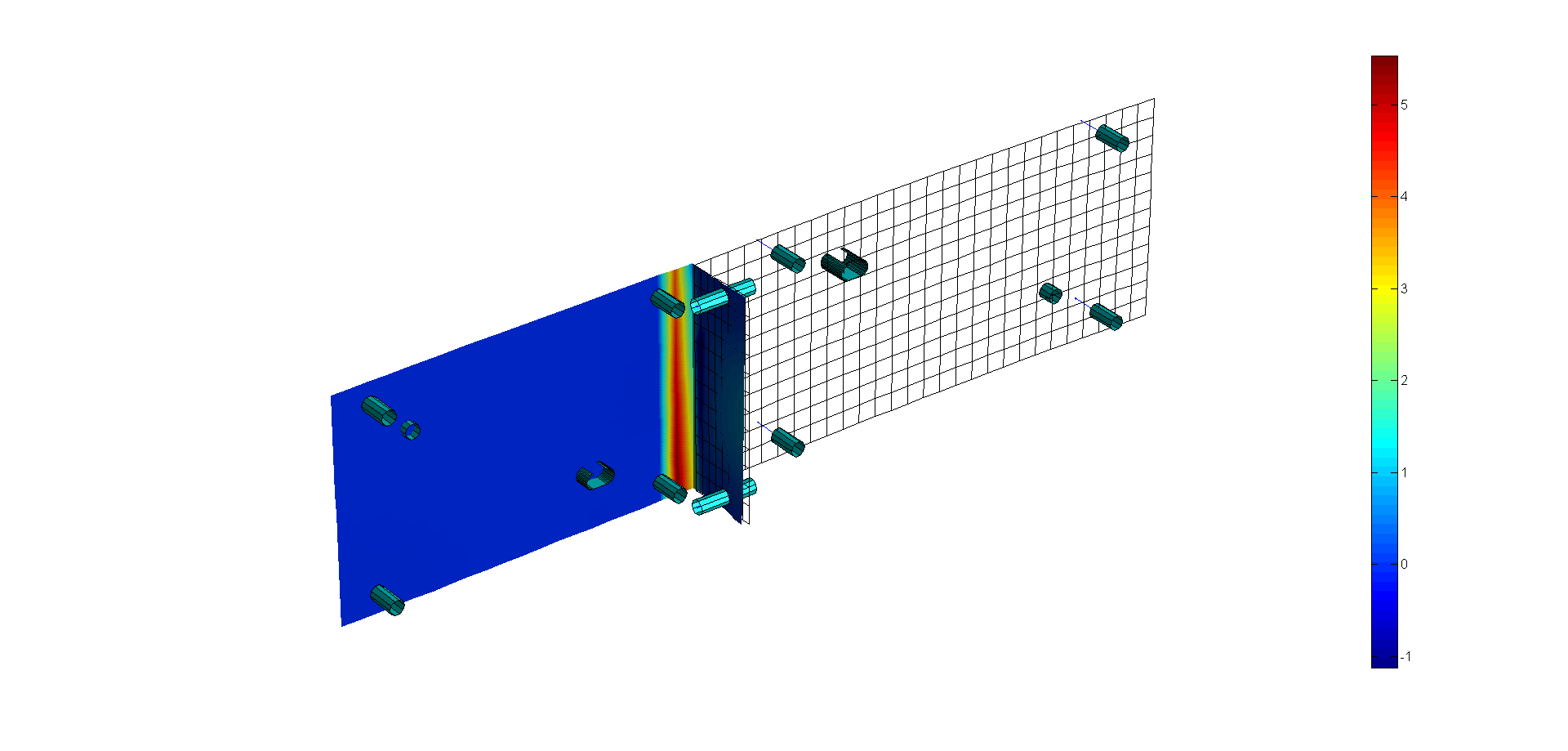
fem.Boundary.ContactPair(1).Offset=0.0;

fem.Boundary.ContactPair(1).Physic='shell';

fem.Boundary.ContactPair(1).Enable=true;

fem.Boundary.ContactPair(1).Frame=’ref’;

% run simulation without dimple



**Gap distribution - no dimple**

% ... now use dimple pairs (dimple height=0.1)

D=[0 -12 14;0 -12 -14];

for i=1:size(D,1)

fem.Boundary.DimplePair(i).Pm=D(i,:);

fem.Boundary.DimplePair(i).Master=1;

fem.Boundary.DimplePair(i).MasterFlip=false;

fem.Boundary.DimplePair(i).Slave=2;

fem.Boundary.DimplePair(i).SearchDist=5;

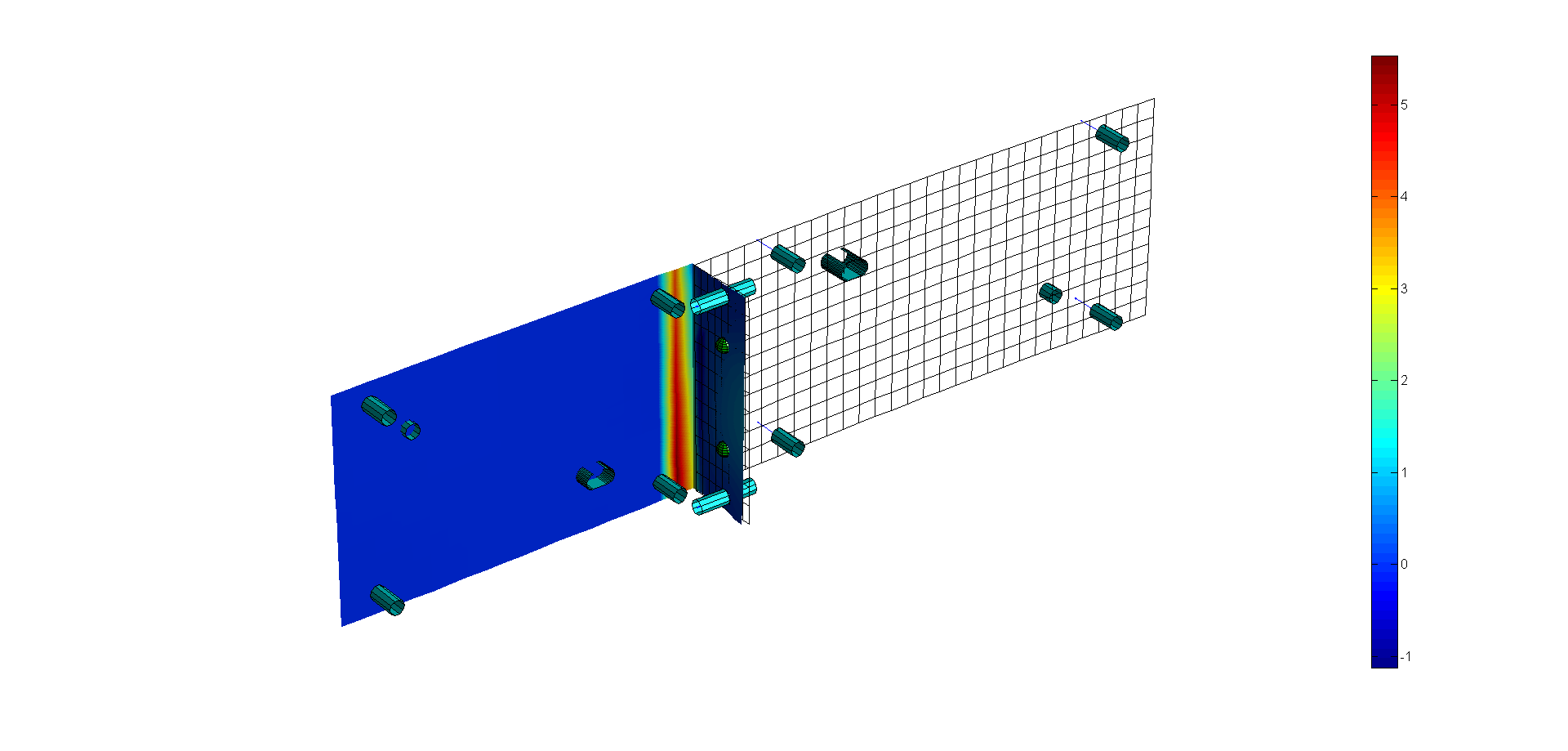
fem.Boundary.DimplePair(i).Height=0.1;

fem.Boundary.DimplePair(i).Offset=0.0;

fem.Boundary.DimplePair(i).Physic='shell';

fem.Boundary.DimplePair(i).Frame='ref';

end



**Gap distribution - with dimple**

## Set: fem.Boundary.Load.Element

*Aim:*

Define load at element (domain) level.

*Syntax:*

.Element(id).Pm=(x, y, z) coordinates of key point to project

.Element(id).SearchDist=search distance

.Element(id).Reference="cartesian", "vectorTra", "vectorRot"

.Element(id).Nm=given unit vector

.Element(id).DoF=list of dofs

.Element(id).Value=prescribed displacement/rotation

.Element(id).Domain=domain id

.Element(id).Physic="shell"

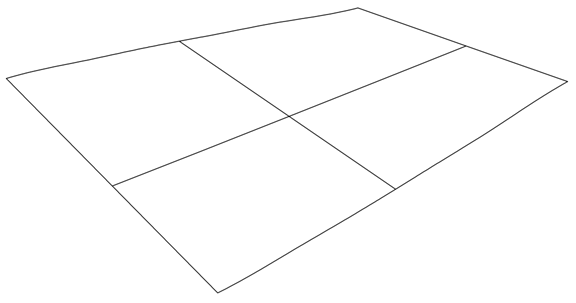
*Input:*

* *id*: constraint identification number (integer)

*Output:*

* *fem*: fem structure

*Description:*



Pm

F0

Pm,d

Nm

* **Pm**: define the coordinates of the key point to project (Pm,d) on the specified domain. Hence, the load is applied to the projected point. Notice that this call is useful to define loads for any key point, even not-coincident with existing mesh nodes (double).
* **SearchDist**: used to determine the projected point. If the normal distance from the key points to the specified domain is greater than "SearchDist" no load is assigned (double).
* **Reference**: it may contain the following options:
* *cartesian*: define load with respect to the cartesian coordinate frame (string)
* v*ectorTra*: define translation load along a given direction (defined into "Nm" field) (string)
* *vectorRot*: define rotation load (moment/wrench) around a given direction (defined into "Nm" field) (string).
* **Nm**: define a unit vector for the prescribed load. This field has to be defined when "vectorTra" or "vectorRot" options are used (double).
* **DoF**: define the list of degrees of freedom. The following syntax is adopted [u, v, w, α, β, γ] = [1 2 3 4 5 6]. This field has to be defined when "cartesian" option is employed (integer).
* **Value**: define the amount (F0) of load (double). The number of entries depends on:
* *cartesian*: [1x6] entries
* v*ectorTra*: scalar entry
* *vectorRot*: scalar entry.
* **Domain**: define domain identification number (integer).
* **Physi**c: define the modelled physics. When modelling sheet-metal parts, this field has to be "shell" (string).

*Equation:*

**cartesian: F** = **F0**

**vectorTra / vectorRot: F⋅ Nm = F0**

*Example:*

%... force along Z (0, 0, 1) axis at point (5, 0.1, 0)

fem.Boundary.Load.Element(1).Pm=[5 0.1 0];

fem.Boundary.Load.Element(1).Reference='vectorTra';

fem.Boundary.Load.Element(1).SearchDist=10;

fem.Boundary.Load.Element(1).Nm=[0 0 1];

fem.Boundary.Load.Element(1).Value=-100;

fem.Boundary.Load.Element(1).Domain=1;

fem.Boundary.Load.Element(1).Physic='shell';

% the above load condition can be also stated as:

fem.Boundary.Load.Element(1).Pm=[5 0.1 0];

fem.Boundary.Load.Element(1).Reference='cartesian';

fem.Boundary.Load.Element(1).SearchDist=5;

fem.Boundary.Load.Element(1).DoF=3;

fem.Boundary.Load.Element(1).Value=1;

fem.Boundary.Load.Element(1).Domain=1;

fem.Boundary.Load.Element(1).Physic='shell';

## Set: fem.Boundary.Load.Node

*Aim:*

Define load condition at node level.

*Syntax:*

.Node(id).Node=list of loaded nodes

.Node(id).Reference="cartesian", "vectorTra", "vectorRot"

.Node(id).Nm=given unit vector

.Node(id).DoF=list of dofs

.Node(id).Value=prescribed load

.Node(id).Physic="shell"

*Input:*

* *id*: load identification number (integer)

*Output:*

* *fem*: fem structure

*Description:*

* **Node**: define node identification number (integer).
* **Reference**: it may contain the following options:
* *cartesian*: define load with respect to the cartesian coordinate frame (string)
* v*ectorTra*: define translation load along a given direction (defined into "Nm" field) (string)
* *vectorRot*: define rotation load (moment) around a given direction (defined into "Nm" field) (string).
* **Nm**: define a unit vector for the prescribed load. This field has to be defined when "vectorTra" or "vectorRot" options are used (double).
* **DoF**: define the list of constrained degrees of freedom. The following syntax is adopted [u, v, w, α, β, γ] = [1 2 3 4 5 6]. This field has to be defined when "cartesian" option is employed (integer).
* **Value**: define the amount of load (double). The number of entries depends on:
* *cartesian*: [1x6] entries
* v*ectorTra*: scalar entry
* *vectorRot*: scalar entry.
* **Physic**: define the modelled physics. When modelling sheet-metal parts, this field has to be "shell" (string).

*Example:*

% define constant

F=100;

% define load condition acting along x and z axis applied at node (1, 2, 3)

fem.Boundary.Load.Node(1).Node=[1 2 3];

fem.Boundary.Load.Node(1).Reference='cartesian';

fem.Boundary.Load.Node(1).DoF=[1 3];

fem.Boundary.Load.Node(1).Value=[F F];

fem.Boundary.Load.Node(1).Physic="shell"

% the same load can be defined as:

fem.Boundary.Load.Node(1).Node=[1 2 3];

fem.Boundary.Load.Node(1).Reference='vectorTra';

fem.Boundary.Load.Node(1).Nm=[1 0 1]/norm[1 0 1];

fem.Boundary.Load.Node(1).Value=F \* norm[1 0 1];

fem.Boundary.Load.Node(1).Physic="shell"

# DOMAIN CONDITIONS

This Section describes the main options regarding domain properties and their options.

## Set: fem.Domain

*Aim:*

Define material, constant parameters and load condition for the specified component.

*Syntax:*

.Domain(id).Material.E=Young Modulus

.Domain(id).Material.ni=Poisson ratio

.Domain(id).Material.lamda=shear correction factor

.Domain(id).Constant.Th=thickness

.Domain(id).Load.Flag=true / false

.Domain(id).Load.Value=[Fx Fy Fz Mx My Mz]

.Domain(1).Status=true / false

*Input:*

* *id*: component identification number (integer)

*Output:*

* *fem*: fem structure

*Description:*

This call can be used to define material properties or load conditions for the specified component.

* **E**: define Young's Modulus (double).
* **ni**: define Poisson's ratio (double).
* **lamda**: define the shear correction factor. The suggested value for rectangular cross-sections is 5/6 (double).
* **Th**: thickness of shell elements. It is assumed constant over all elements belonging to that domain (double).
* **Load.Value**: define load condition applied at domain side (double). It is assumed as force per unit of surface area.
* **Load.Flag**: is "true" then domain loads are calculated (true/false).
* **Status**: is "false" then domain is de-activated and not considered in the calculation (true/false).

*Example:*

%... import

fem=importMesh(fem,filename);

% define constants

th=0.1;

E=1e7;

nu=0.3;

lamda=5/6;

F=0.5;

% component "5"

% material

fem.Domain(5).Material.E=E;

fem.Domain(5).Material.ni=nu;

fem.Domain(5).Material.lamda=lamda;

% constants

fem.Domain(5).Constant.Th=th;

% define load (force per unit area) along z direction

fem.Domain(5).Load.Value=[0 0 F 0 0 0];

fem.Domain(5).Load.Flag=true;

# PROCESSING COMMANDS

This Section describes the main commands to process input data and run simulation.

## femLoad

*Aim:*

Load fem solution.

*Syntax:*

fem=femLoad(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *fem*: fem structure

*Description:*

Load fem solution in terms of:

* output variables (displacements and reaction forces) - **fem.Sol.USet.U** and **fem.Sol.USet.R**
* boundary and domain conditions - **fem.Sol.ModelSet.Boundary** and **fem.Sol.ModelSet.Domain.**

The identification number of loaded solution is set through **fem.Sol.SolId**.

*Dependency:*

* **femPreProcessing**

*Example:*

% import mesh file

fem=importMesh(fem, filename);

% compile and solve model

fem=femRefresh(fem);

fem=femSolve(fem);

% store model: 1

fem.Sol.SolId=1;

fem=femStore(fem);

% reset model

fem=femReset(fem);

% run new calculation here...

fem=femRefresh(fem);

fem=femSolve(fem);

% store model: 2

fem.Sol.SolId=2;

fem=femStore(fem);

% reset model

fem=femReset(fem);

% now load model "1"

fem.Sol.SolId=1;

fem=femLoad(fem);

% post-process results

a=axes; hold all;

fem.Post.Options.ParentAxes=a;

fem.Post.Options.ColorPatch='b';

fem.Post.Options.ShowPatch=false;

fem.Post.Contour.Domain=1;

fem.Post.Contour.ContourVariable='v';

% plot nominal mesh

meshPlot(fem)

contourPlot(fem)

## femMapping

*Aim:*

Map the source domain to the destination domain.

*Syntax:*

fem=femMapping(femSrc, femDst)

*Input:*

* *femSrc*: source structure
* *femDst*: destination structure

*Output:*

* *femDst*: updated destination structure

*Description:*

This command maps the source domain on the destination domain.



Inputs are provided as follows:

* **femDst.Mapping.Source**: source domain
* **femDst.Mapping.Destination**: destination domain
* **femDst.Mapping.SearchDistance:** searching distance. This value is used to find the closest projected point of the destination domain (Pdst) on the source domain (Psrc).
* **femDst.Mapping.MapVariable**: mapping variable. The variable on the source domain (Usrc) is assigned to the related point on the destination domain (Udst). The allowed options are:
  + *u*: displacement along X
  + *v*: displacement along Y
  + *w*: displacement along Z
  + *alfa*: rotation around X
  + *beta*: rotation around Y
  + *gamma*: rotation around Z
  + *user*: user variable. This variable has to be stored in **femSrc.Sol.UserExp**.

Notice that gap variables are not supported.

*Dependency:*

* **femPreProcessing**
* **femRefresh**

*Example:*

% import mesh file

fem=importMesh(fem, filename);

% this model is made of two domains (domain 1=source; domain 2=destination)

fem=femPreProcessing(fem);

% define some constraints and loads here

%.-------------------

% solver model

fem.Options.Solver.Method='penalty';

fem.Options.Solver.MaxIter=100;

fem.Options.Solver.Eps=1e-8;

fem=femRefresh(fem);

fem=femSolve(fem);

% create destination structure

femDst=fem;

% map solution on the destination domain (domain 2)

femDst.Mapping.Source=1;

femDst.Mapping.Destination=2;

femDst.Mapping.SearchDist=10;

femDst.Mapping.MapVariable='u';

femDst=femMapping(fem, femDst);

% create even a random user field and map it

nnode=size(fem.xMesh.Node.Coordinate,1); % no. of nodes

dev=randn(1,nnode)\*5;

fem.Sol.UserExp=dev;

femDst.Mapping.MapVariable='user';

femDst=femMapping(fem, femDst);

%% plot results

% define current axis for plot purpose

ax=axes;

hold all

fem.Post.Options.ParentAxes=ax;

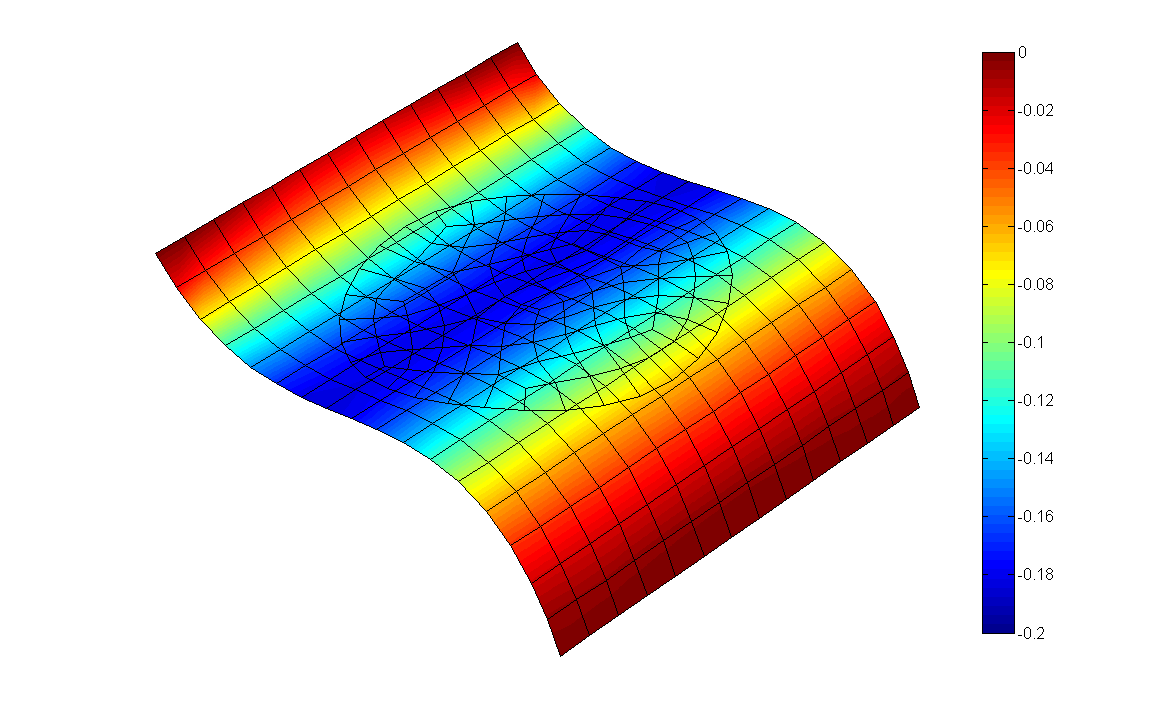
% plot source

meshPlot(fem)

fem.Post.Contour.Domain=1;

fem.Post.Contour.ContourVariable='v';

contourPlot(fem)



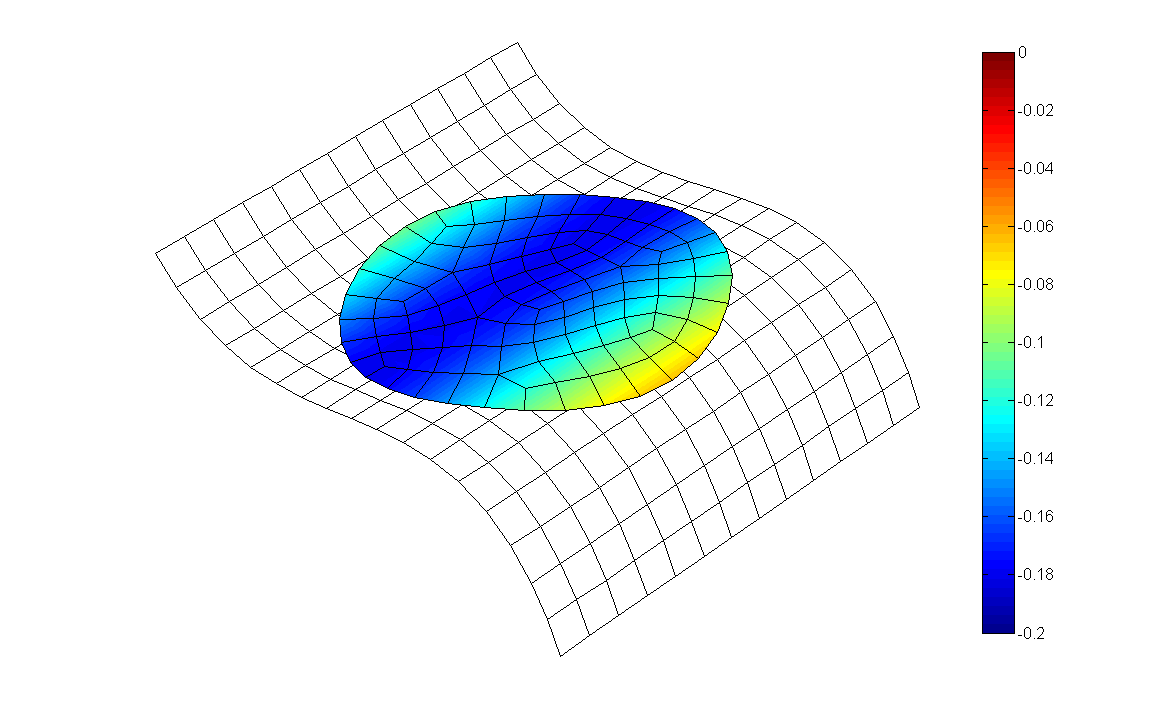
% plot destination

meshPlot(femDst)

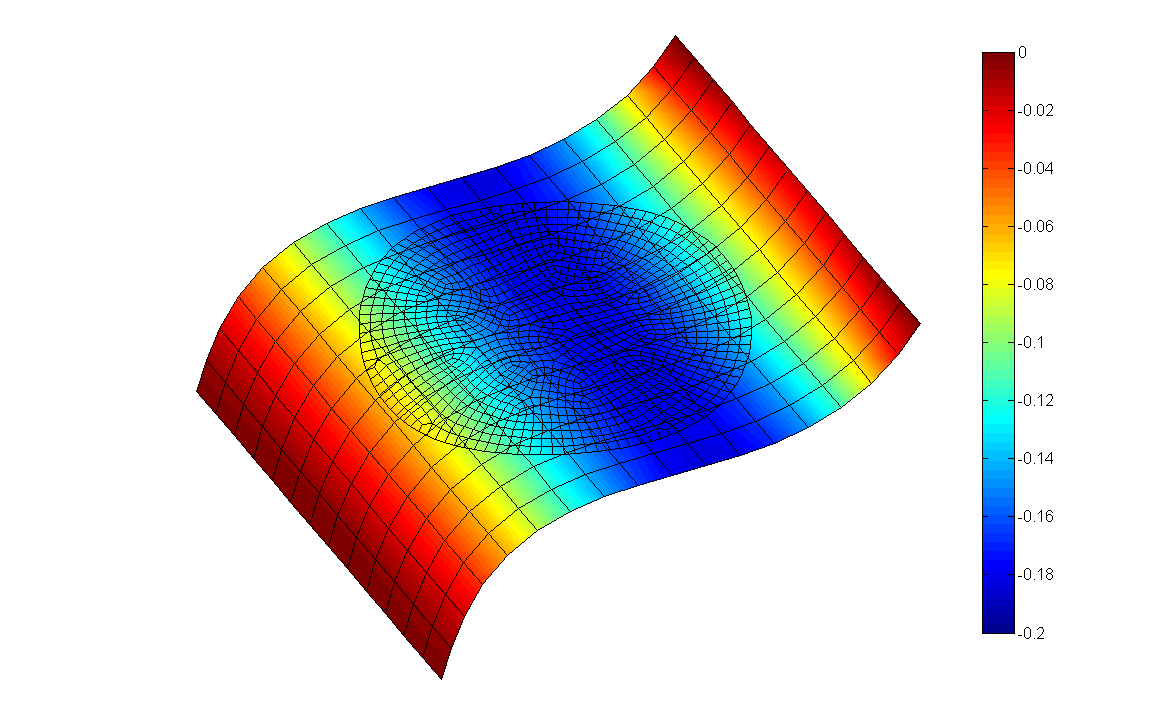
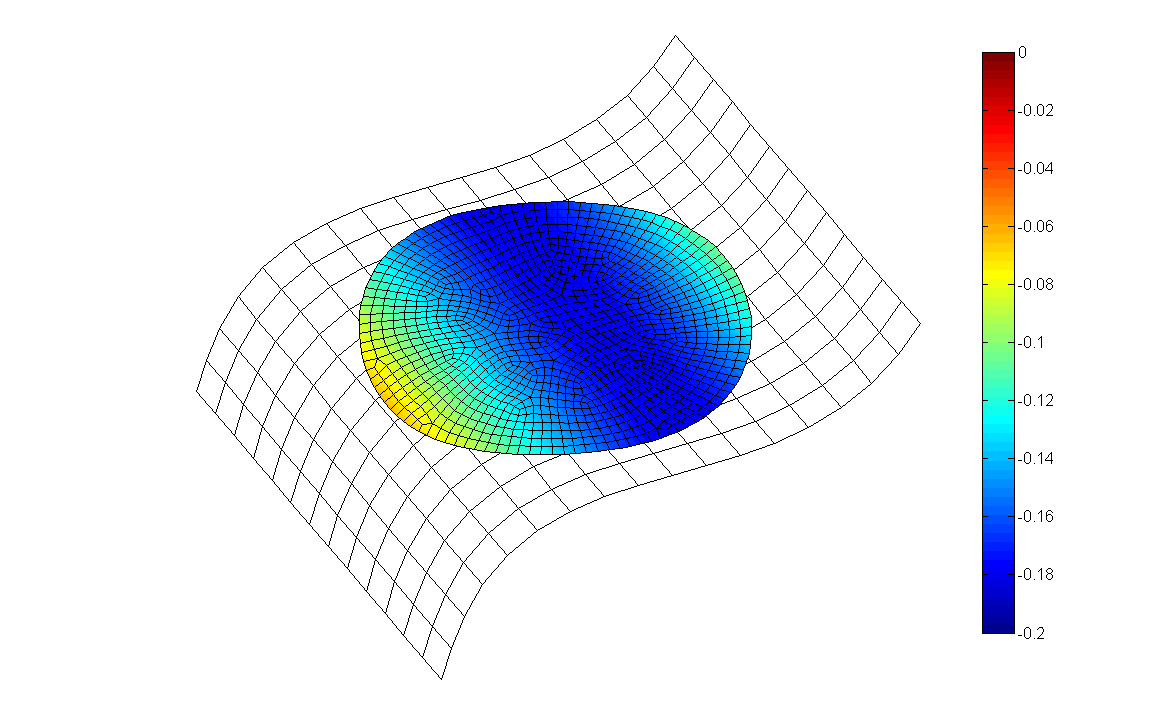
femDst.Post.Contour.Domain=2;

femDst.Post.Contour.ContourVariable='v';

contourPlot(femDst)



Notice that the mapping approach can be applied on any destination mesh model, even finer than the source model.

Source domain Destination domain

## femPreProcessing

*Aim:*

Pre-process fem structure.

*Syntax:*

fem=femPreProcessing(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *fem*: fem structure

*Description:*

This command performs data pre-processing:

* compile degrees of freedom
* calculate element/node normal vectors
* calculate element reference frame
* calculate stiffness matrix for every element
* initialise data for mesh denoising.

Notice that the required user-input are the mesh model and the domain properties.

*Dependency:*

* **mesh model has been loaded**

*Example:*

% import mesh file

fem=importMesh(fem, filename);

% calculate stiffness matrix

fem=femPreProcessing(fem);

## femReactionForce

*Aim:*

Recover reaction forces.

*Syntax:*

fem=femReactionForce(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *fem*: fem structure

*Description:*

This command recovers reaction forces of all constraints defined in the current model. Reaction forces are stored in:

* **fem.Boundary.Constraint.SPC.Reaction(idc)**: "Single-Point-Constraints" boundary conditions
* **fem.Boundary.Constraint.MPC(idc).Reaction**: "Multi-Point-Constraints" boundary conditions
* **fem.Boundary.Constraint.ContactWset(idc).Reaction.R**: Active unilateral constraints (including contact pairs) conditions

"idc" can be obtained as (where "id" is the high level constraint identification number):

* **Single point constraint (element-based)**  
  "idc" = fem.Boundary.Constraint.Bilateral.Element(id).Reaction.Id
* **Single point constraint (node-based)**  
  "idc" = fem.Boundary.Constraint.Bilateral.Node(id).Reaction.Id
* **Multi point constraint (unilateral constraint in "lock" mode)**  
  "idc" = fem.Boundary.Constraint.Unilateral(id).Reaction.Id
* **Multi point constraint (rigid link)**  
  "idc" = fem.Boundary.Constraint.RigidLink(id).Reaction.Id
* **Contact point constraint (unilateral constraint in "free" mode)**  
  "idct" = fem.Boundary.Constraint.Unilateral(id).Reaction.Id  
  ids=[fem.Boundary.Constraint.ContactWset.Reaction.Id]  
  idc=find(ids==idct)

*Dependency:*

* **femPreProcessing**
* **femRefresh**
* **femSolve**

*Equation:*

**Penalty Method** (ε=penalty stiffness)**:** 

**Lagrange Method** (λ=lagrangian multiplier)**:** 

*Example:*

% import mesh file

fem=importMesh(fem,filemesh);

% define bilateral constraints

fem.Boundary.Constraint.Bilateral.Node(1).Node=[1 21];

fem.Boundary.Constraint.Bilateral.Node(1).Reference='cartesian';

fem.Boundary.Constraint.Bilateral.Node(1).DoF=1:6;

fem.Boundary.Constraint.Bilateral.Node(1).Value=[0 0 1 0 0 0];

fem.Boundary.Constraint.Bilateral.Node(1).Physic='shell';

% define unilateral constraints

fem.Boundary.Constraint.Unilateral(1).Pm=[10 -40 1];

fem.Boundary.Constraint.Unilateral(1).SearchDist=10;

fem.Boundary.Constraint.Unilateral(1).Nm=-[0 0 1];

fem.Boundary.Constraint.Unilateral(1).Size=false;

fem.Boundary.Constraint.Unilateral(1).Offset=0;

fem.Boundary.Constraint.Unilateral(1).Domain=1;

fem.Boundary.Constraint.Unilateral(1).Constraint='free';

% solve model

fem=femPreProcessing(fem);

fem=femRefresh(fem);

fem=femSolve(fem);

% get reaction forces

fem=femReactionForce(fem);

%%----

>> fem.Boundary.Constraint.SPC.Reaction

0

-0.4659

-17.9829

-3.6205

0

0

0

-0.7412

0.0339

-18.0717

0

>> fem.Boundary.Constraint.ContactWset.Reaction

0.7928

## femRefresh

*Aim:*

Compile equations.

*Syntax:*

fem=femRefresh(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *fem*: fem structure

*Description:*

This command compiles equations regarding:

* boundary constraint
* boundary load
* domain load
* contact and dimple pairs

*Dependency:*

* **femPreProcessing**

*Example:*

% import mesh file

fem=importMesh(fem, filename);

% calculate stiffness matrix

fem=femPreProcessing(fem);

% define inputs

% LOAD

F=100;

fem.Boundary.Load.Node(1).Node=[1 2 3];

fem.Boundary.Load.Node(1).DoF=[1 3];

fem.Boundary.Load.Node(1).Value=[F F];

fem.Boundary.Load.Node(1).Physic="shell"

% CONSTRAINT

fem.Boundary.Constraint.Bilateral.Node(1).Node=[10 14];

fem.Boundary.Constraint.Bilateral.Node(1).Reference='cartesian';

fem.Boundary.Constraint.Bilateral.Node(1).DoF=[1 2 3 4 5 6];

fem.Boundary.Constraint.Bilateral.Node(1).Value=[0 0 0 0 0 0];

fem.Boundary.Constraint.Bilateral.Node(1).Physic='shell';

% compile equations

fem=femRefresh(fem);

## femReset

*Aim:*

Reset fem model.

*Syntax:*

fem=femReset(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *fem*: fem structure

*Description:*

Reset current fem model in terms of:

* output variables and counters.
* boundary conditions.

*Dependency:*

* **mesh model has been initialised**

*Example:*

% import mesh file

fem=importMesh(fem, filename);

% compile and solve model

fem=femRefresh(fem);

fem=femSolve(fem);

% save a copy for back-up

fem1=fem;

% reset model

fem=femReset(fem);

% do more stuffs from this point...

## femSetSelection

*Aim:*

Set user selection.

*Syntax:*

fem=femSetSelection(fem, idnode)

*Input:*

* *fem*: fem structure
* *idnode*: list of selected nodes

*Output:*

* *fem*: fem structure

*Description:*

This command defines a selection region based on the "idnode" list parsed in input. The following fields are updated:

* **fem.Selection.Element**: contains the details of the selected elements
* **fem.Selection.Node**: contains the details of the selected nodes.

The boundary nodes of the selected region are stored in **fem.Selection.Node.Boundary**. The list of selected nodes, "idnode" can be parsed manually or interactively, through the graphical user interface. To do that, the command **selectionTool** can be evocated.

All elements not belonging to the selected region will be discarded when solving the model. If the option **fem.Options.UseActiveSelection** is "true" than only selected elements/nodes are computed. If it is "false", than the selection is extended to all elements/nodes of the current model.

*Dependency:*

* **mesh model has been initialised**

*Example:*

% import mesh file

fem=importMesh(fem, filename);

% define list of nodes:

idnode=1:200;

% run selection routine

fem=femSetSelection(fem, idnode);

% plot selection

a=axes;

fem.Post.Options.ParentAxes=a;

fem.Post.Options.ShowPatch=false;

fem.Post.Options.ShowAxes=false;

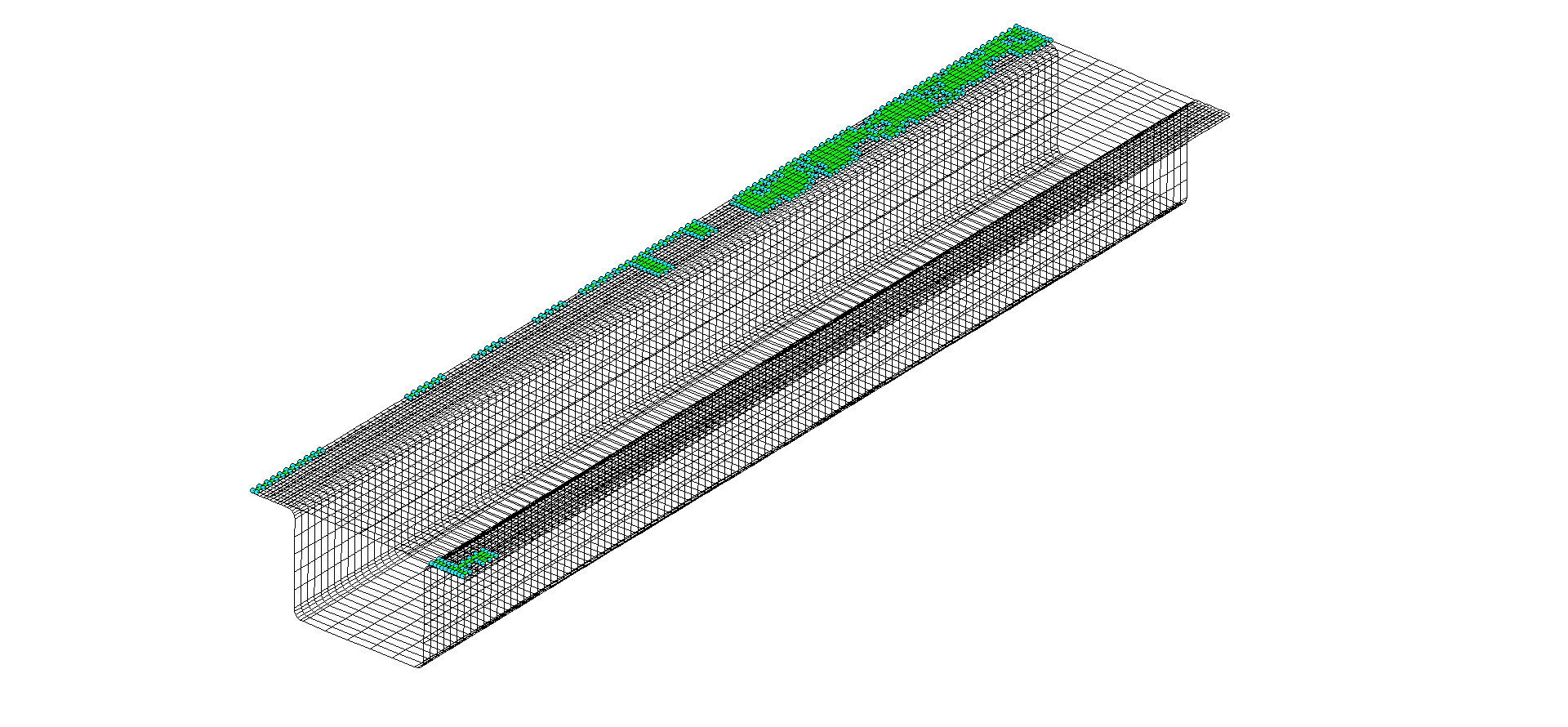
% plot model

meshPlot(fem)

% plot selection

fem.Post.Options.ShowPatch=true;

selectionPlot(fem)



**Selected elements (green patches)**

## femSolve

*Aim:*

Solve fem model.

*Syntax:*

fem=femSolve(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *fem*: fem structure

*Description:*

Solve model and calculate:

* output variables (displacement and gap variables)
* calculate deformed shape
* transform output variables into global reference frame

Two algorithms are available: penalty method and Lagrange multipliers. Solver settings are managed through **fem.Options.Solver**. The following options are available:

* **Method**: 'penalty' or 'lagrange'. It defines the method to manage constraint equations (string)
* **PenaltyStiffness**: penalty stiffness required for the penalty solver method (double)
* **LinearSolver**: 'umfpack' or 'cholmod'. It defines the solver type employed to solve linear systems (string)
* **StoreAssembly**: is "true" than the assembly stiffness matrix is stored (true / false)
* **MaxIter**: define the maximum number of iterations required to solve non-linear constraints (integer)
* **Eps**: numerical tolerance (double)
* **EpsCheck**: numerical tolerance used to check residuals after solving linear systems (double)
* **CheckTol**: is "true" than the numerical solution is set to zero when the residuals are less than "EpsCheck" (true / false)

Generally speaking, the penalty method is less accurate then Lagrange multipliers but it is more stable. This means that the penalty method should be adopted for solving optimisation problems, where a high automation level is required. The convergence rate of the penalty method depends on the penalty stiffness. It is known that the higher the penalty stiffness the smoother the convergence rate. However, if that value is too high the stiffness matrix may become ill-conditioned, so that numerical results are unreliable.

To avoid ill-conditioning problems the penalty might be chosen equal to: , where "E" is the Young's Modulus, "Ndof" is the number of degrees of freedom solved for, and "t" is the round-off error, typically equal to 10-17.

*Dependency:*

* **femPreProcessing**

*Example:*

% import mesh file

fem=importMesh(fem, filename);

% define domain properties

fem.Domain(1).Material.E=E;

fem.Domain(1).Material.ni=nu;

fem.Domain(1).Material.lamda=lamda;

fem.Domain(1).Constant.Th=th;

% calculate stiffness matrix

fem=femPreProcessing(fem);

% define inputs

% LOAD

F=100;

fem.Boundary.Load.Node(1).Node=[1 2 3];

fem.Boundary.Load.Node(1).DoF=[1 3];

fem.Boundary.Load.Node(1).Value=[F F];

fem.Boundary.Load.Node(1).Physic="shell"

% CONSTRAINT

fem.Boundary.Constraint.Bilateral.Node(1).Node=[10 14];

fem.Boundary.Constraint.Bilateral.Node(1).Reference='cartesian';

fem.Boundary.Constraint.Bilateral.Node(1).DoF=[1 2 3 4 5 6];

fem.Boundary.Constraint.Bilateral.Node(1).Value=[0 0 0 0 0 0];

fem.Boundary.Constraint.Bilateral.Node(1).Physic='shell';

%%--define solver options

% use lagrange method

fem.Options.Solver.Method='lagrange';

% use umfpack

fem.Options.Solver.LinearSolver='umfpack';

% do not store assembly stiffness matrix

fem.Options.Solver.StoreAssembly=false;

% max.iterations allowed

fem.Options.Solver.MaxIter=100;

% numerical tolerance allowed

fem.Options.Solver.Eps=1e-8;

%%--------------------------------

% compile equations

fem=femRefresh(fem);

% solve model

fem=femSolve(fem);

## femStore

*Aim:*

Storecurrent solution.

*Syntax:*

fem=femStore(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *fem*: fem structure

*Description:*

Store current solution in terms of:

* output variables (displacements and reaction forces) - **fem.Sol.USet.U** and **fem.Sol.USet.R**
* boundary and domain conditions - **fem.Sol.ModelSet.Boundary** and **fem.Sol.ModelSet.Domain.**

The identification number of stored solution can be handled through **fem.Sol.SolId**.

*Dependency:*

* **mesh model has been loaded**

*Example:*

% import mesh file

fem=importMesh(fem, filename);

% pre-process

fem=femPreProcessing(fem);

% run 10 simulation changing the boundary conditions

for k=1:10

%define NC-block

NC=[-262.46 85 -150

-362.72 85 -150

-262.46 85 240

-362.72 85 240];

r=randn(size(NC,1),3)\*10+10;

NC=NC+r; % random selection

% add boundary conditions

for i=1:size(NC,1)

fem.Boundary.Constraint.Unilateral(i).Pm=NC(i,:);

fem.Boundary.Constraint.Unilateral(i).SearchDist=300;

fem.Boundary.Constraint.Unilateral(i).Nm=[0 1 0];

fem.Boundary.Constraint.Unilateral(i).Size=false;

fem.Boundary.Constraint.Unilateral(i).Offset=0;

fem.Boundary.Constraint.Unilateral(i).Domain=1;

fem.Boundary.Constraint.Unilateral(i).Constraint='lock';

end

% compile and solve model

fem=femRefresh(fem);

fem=femSolve(fem);

% store model

fem.Sol.SolId=k;

fem=femStore(fem);

% reset model

fem=femReset(fem);

end

% check stored data structure

>> fem.Sol.USet

U: {1x10 cell}

R: {1x10 cell}

>> fem.Sol.ModelSet

Boundary: {1x10 cell}

Domain: {1x10 cell}

## femSubModel

*Aim:*

Create a sub-model.

*Syntax:*

femDst=femSubModel(femSrc, femDst)

*Input:*

* *femSrc*: source structure
* *femDst*: destination structure

*Output:*

* *femDst*: updated destination structure

*Description:*

Sub-modelling is a finite element technique used to get more accurate results in a region of the model.It is a way to “zoom in” on specific regions of a previously-analyzed model, create a fine mesh, and obtain highly accurate results just for that region.



In order to generate a sub-model, the following steps should be followed:

* **Create and analyze the initial (coarse) model (SOURCE)**
  + This follows the same steps as in a classical analysis.
  + The geometry does not need to include local details such as fillet radii.
* **Create the sub-model (DESTINATION)**
  + The sub-model is an independent, more finely meshed model of a region of interest within the initial model.
  + Typically, the sub-model will include details — such as a fillet radius — that were omitted in the initial model.
  + Some rules for the sub-model:
    - It must be in the same location with respect to the global origin as the corresponding portion in the initial model.
    - It must have the same loading and constraint conditions as the initial model.
* **Perform cut-boundary interpolation** 
  + This is the key step in sub-modelling, where displacements from the initial model are mapped to the cut boundaries of the sub-model.
* **Analyze the sub-model**

The command "femSubModel" allows to automatically generate the cut-boundary interpolation.

The cutting boundary is declared in **femDst.Domain(id).SubModel.CuttingSt**, where "id" is the domain identification id. The implemented algorithm uses the same strategy adopted to select a given boundary based on 3 given node ids (see "boundary3Nodes" command). Calculated nodes belonging to the cutting boundary are stored in **femDst.Domain(id).SubModel.CuttingId**.

Then, the command automatically assigns the prescribed displacements from the initial model to the cut boundaries of the sub-model (constraints are treated as bilateral defined ad cut boundary nodes).

Notice that the domain identification between source and destination are supposed to be the same. To swap domain identification use the "femSwapComponent" command.

*Dependency:*

* **femPreProcessing**
* **femRefresh**

*Example:*

% import source

fem=importMesh(fem,filesource);

fem=femPreProcessing(fem);

% define load and constraints

%.----------

% solve source model

fem=femRefresh(fem);

fem=femSolve(fem);

% import destination

femDst=importMesh(femDst,filedest);

femDst=femPreProcessing(femDst);

femDst=femRefresh(femDst);

% get cut boundaries

femDst.Domain(1).SubModel.CuttingSt=[408 409 45];

femDst=femSubModel(fem,femDst);

% reset model to avoid compilation errors

femDst=femReset(femDst);

% set loads as required

%.----------

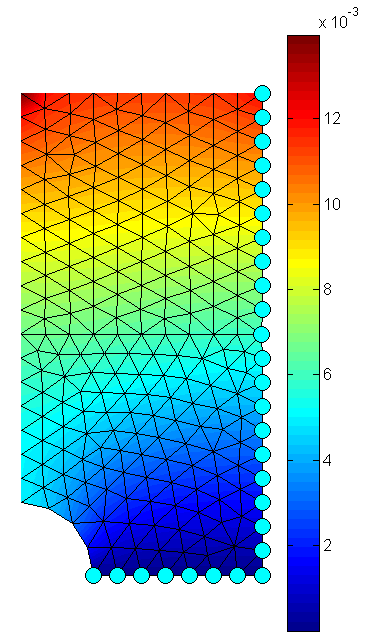
femDst=femRefresh(femDst);

femDst=femSolve(femDst);

% plot source solution

fem.Post.Contour.ContourVariable='u';

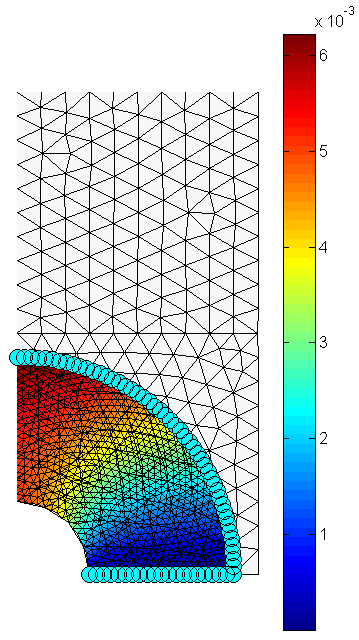
contourPlot(fem)



% plot destination solution

femDst.Post.Contour.ContourVariable='u';

contourPlot(femDst)



## femSwapComponents

*Aim:*

Swap two selected components.

*Syntax:*

fem=femSwapComponents(fem, idcmp1, idcmp2)

*Input:*

* *fem*: fem structure
* idcmp1: first component
* idcmp2: second component

*Output:*

* *fem*: fem structure

*Description:*

Swap identification numbers of two given components. This means that "idcmp1" will have the identification number of "idcmp2" and vice versa.

*Dependency:*

* **mesh model has been loaded**

*Example:*

% import mesh file

fem=importMesh(fem, filename);

% swap components 1 and 2

fem=femSwapComponents(fem, 1, 2);

## flipNormalComponent

*Aim:*

Flip normal vectors of selected component.

*Syntax:*

fem=flipNormalComponent(fem, idcmp)

*Input:*

* *fem*: fem structure
* *idcmp*: component identification number (integer)

*Output:*

* *fem*: fem structure

*Description:*

Flip normal vectors (both element and node vectors are flipped) of selected component. This command is useful to properly set the normal vectors of the master component when working with contact pairs.

*Dependency:*

* **femPreProcessing**

*Example:*

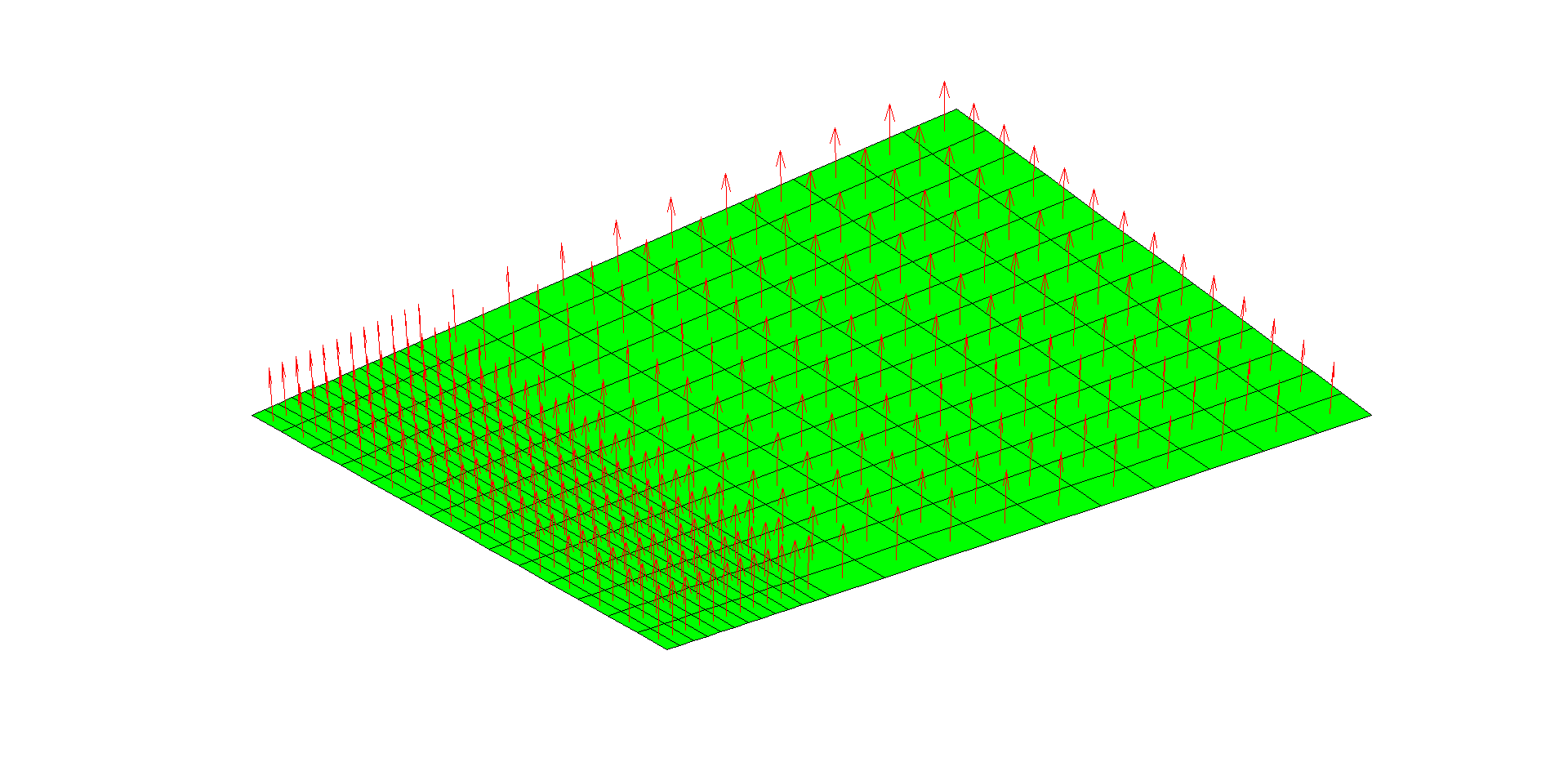
% pre-process model

fem=femPreProcessing(fem);

% show element normal vectors (component=1)

fem.Post.Options.SubSampling=1.0; % 100% percentage of vectors are visualised

normalElementPlot(fem,1)



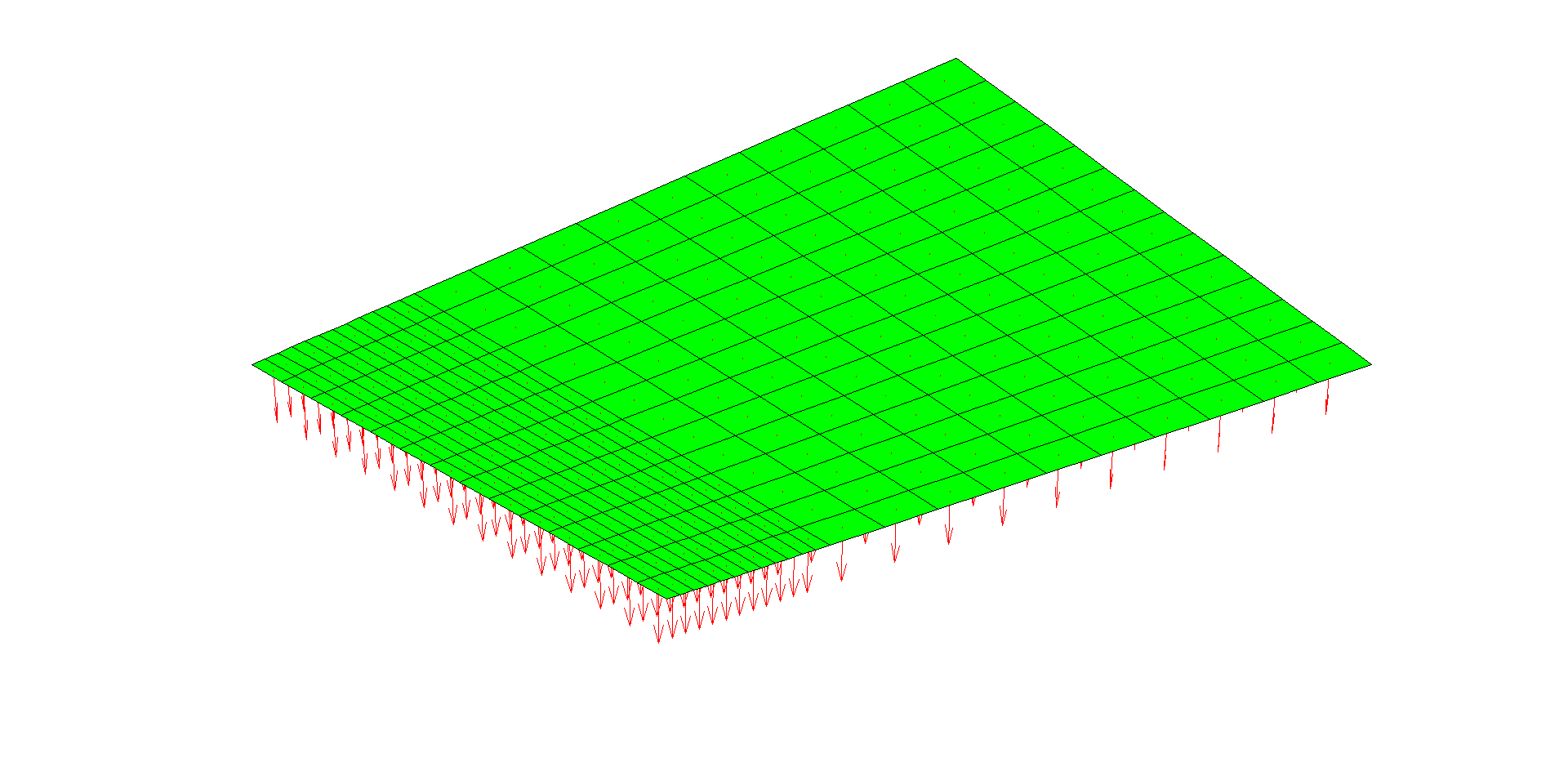
**Normal vectors: before flipping**

% now flip normal vectors

fem=flipNormalComponent(fem, 1);

% plot again:

normalElementPlot(fem,1)



**Normal vectors: after flipping**

## resetNormalComponent

*Aim:*

Reset normal vectors of selected component.

*Syntax:*

fem=resetNormalComponent(fem, idcmp)

*Input:*

* *fem*: fem structure
* *idcmp*: component identification number (integer)

*Output:*

* *fem*: fem structure

*Description:*

Reset normal vectors (both element and node vectors are reset) of selected component to the initial configuration.

*Dependency:*

* **femPreProcessing**

*Example:*

% pre-process model

fem=femPreProcessing(fem);

% flip component several times

for i=1:11

fem=flipNormalComponent(fem, 1);

end

% reset all

fem=resetNormalComponent(fem, 1);

# POST-PROCESSING COMMANDS

This Section describes the main commands to post-process simulation results. The post-processing window can be accessed typing the following lines:

% define current axis for plot purpose

ax=axes;

% set property

fem.Post.Options.ParentAxes=a;

The current axis can be shown depending on the property in **fem.Post.Options.ShowAxes**: if "true" than the current axis is visualised.

## annotationPlot

*Aim:*

Plot 3D annotations.

*Syntax:*

annotationPlot(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *graphical visualisation*

*Description:*

Plot 3D annotations. The following graphical properties can be set:

* **fem.Post.Options.LabelSize**: define font size (double)
* **fem.Post.Options.LengthAxis**: define the length of the arrow (double)
* **fem.Post.Options.SubSampling**: define the percentage of vectors to plot (if "1" than all vectors are plotted). Vectors are randomly selected (double).

Annotations can be visualised based on the following settings:

* **fem.Post.ShowAnnotation.IdDomain**: domain identification number (integer)
* **fem.Post.ShowAnnotation.Node**: show node label (true / false)
* **fem.Post.ShowAnnotation.Element**: show element label (true / false)
* **fem.Post.ShowAnnotation.NormalNode**: show normal vectors of mesh nodes (true / false)
* **fem.Post.ShowAnnotation.NormalElement**: show normal vectors of mesh elements (true / false)

*Dependency:*

* **femPreProcessing**

*Example:*

% pre-process model

fem=femPreProcessing(fem);

% define current axis for plot purpose

ax=axes;

fem.Post.Options.ParentAxes=a;

% define graphical options

fem.Post.Options.LabelSize=5;

fem.Post.Options.LengthAxis=1;

fem.Post.Options.SubSampling=1;

% plot annotations:

% domain id=1

fem.Post.ShowAnnotation.Node=true;

fem.Post.ShowAnnotation.Element=false;

fem.Post.ShowAnnotation.NormalNode=true;

fem.Post.ShowAnnotation.NormalElement=false;

fem.Post.ShowAnnotation.IdDomain=1;

annotationPlot(fem)

% domain id=2

fem.Post.ShowAnnotation.Node=false;

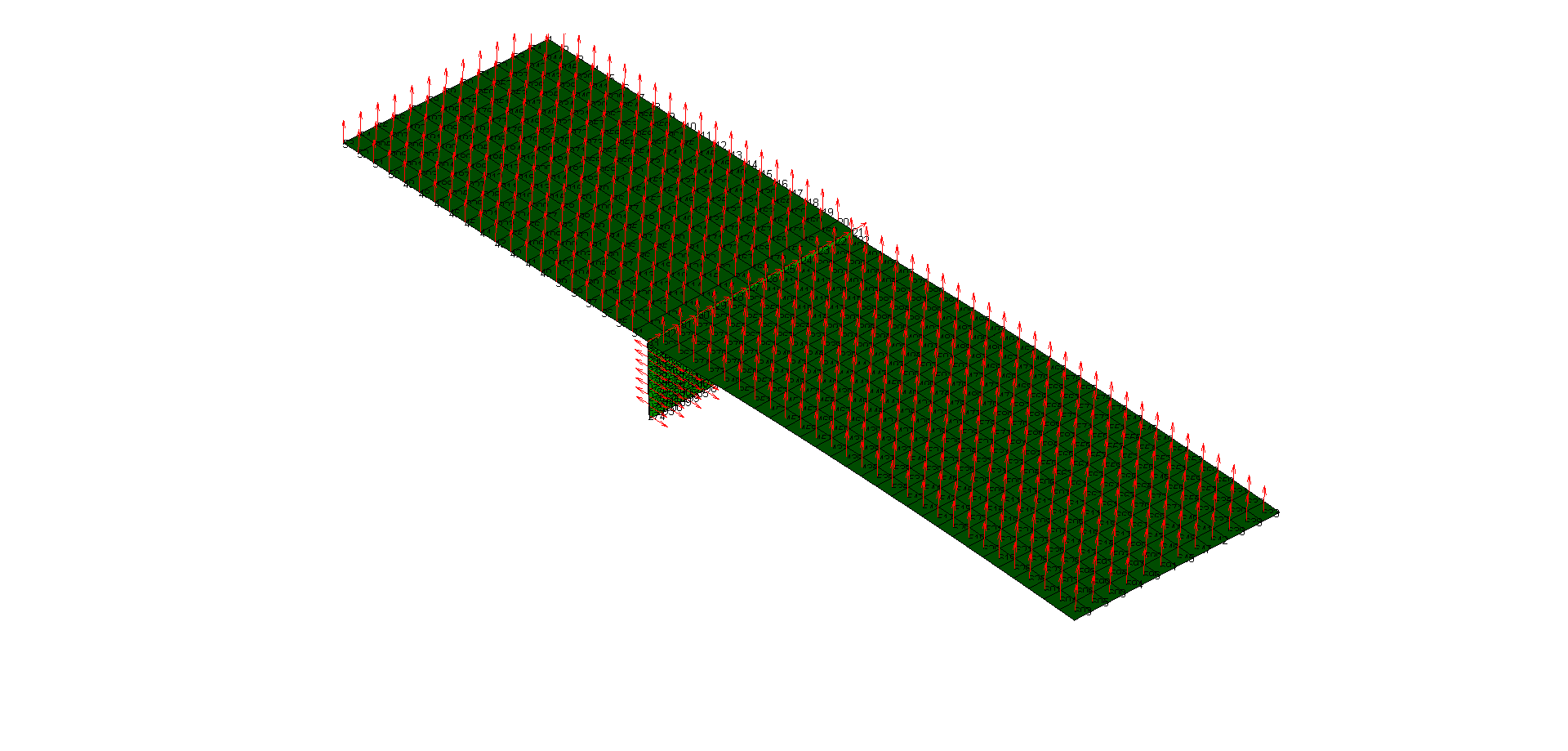
fem.Post.ShowAnnotation.Element=true;

fem.Post.ShowAnnotation.NormalNode=false;

fem.Post.ShowAnnotation.NormalElement=true;

fem.Post.ShowAnnotation.IdDomain=2;

annotationPlot(fem)



**Plotting annotations**

## bilateralElementBcPlot

*Aim:*

Plot bilateral constraints defined at element level.

*Syntax:*

bilateralElementBcPlot(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *graphical visualisation*

*Description:*

Plot bilateral constraints defined at element level. For each constraint a sphere is shown. The following properties can be set:

* **fem.Post.Options.SymbolSize**: define the size of the 3D sphere (double)
* **fem.Post.Options.ShowProjection**: if "true" than projected points are visualised (true / false). This option can be set only after compiling all equations (**femRefresh**)

*Dependency:*

* **femPreProcessing**
* **femRefresh**

*Example:*

% define constraint

fem.Boundary.Constraint.Bilateral.Element(1).Pm=[2 0.1 0.3];

fem.Boundary.Constraint.Bilateral.Element(1).Reference='cartesian';

fem.Boundary.Constraint.Bilateral.Element(1).SearchDist=5;

fem.Boundary.Constraint.Bilateral.Element(1).DoF=[1 2 3 4 5 6];

fem.Boundary.Constraint.Bilateral.Element(1).Value=[0 0 0 0 0 0];

fem.Boundary.Constraint.Bilateral.Element(1).Domain=1;

fem.Boundary.Constraint.Bilateral.Element(1).Physic='shell';

% pre-process model

fem=femPreProcessing(fem);

% define current axis for plot purpose

ax=axes;

fem.Post.Options.ParentAxes=a;

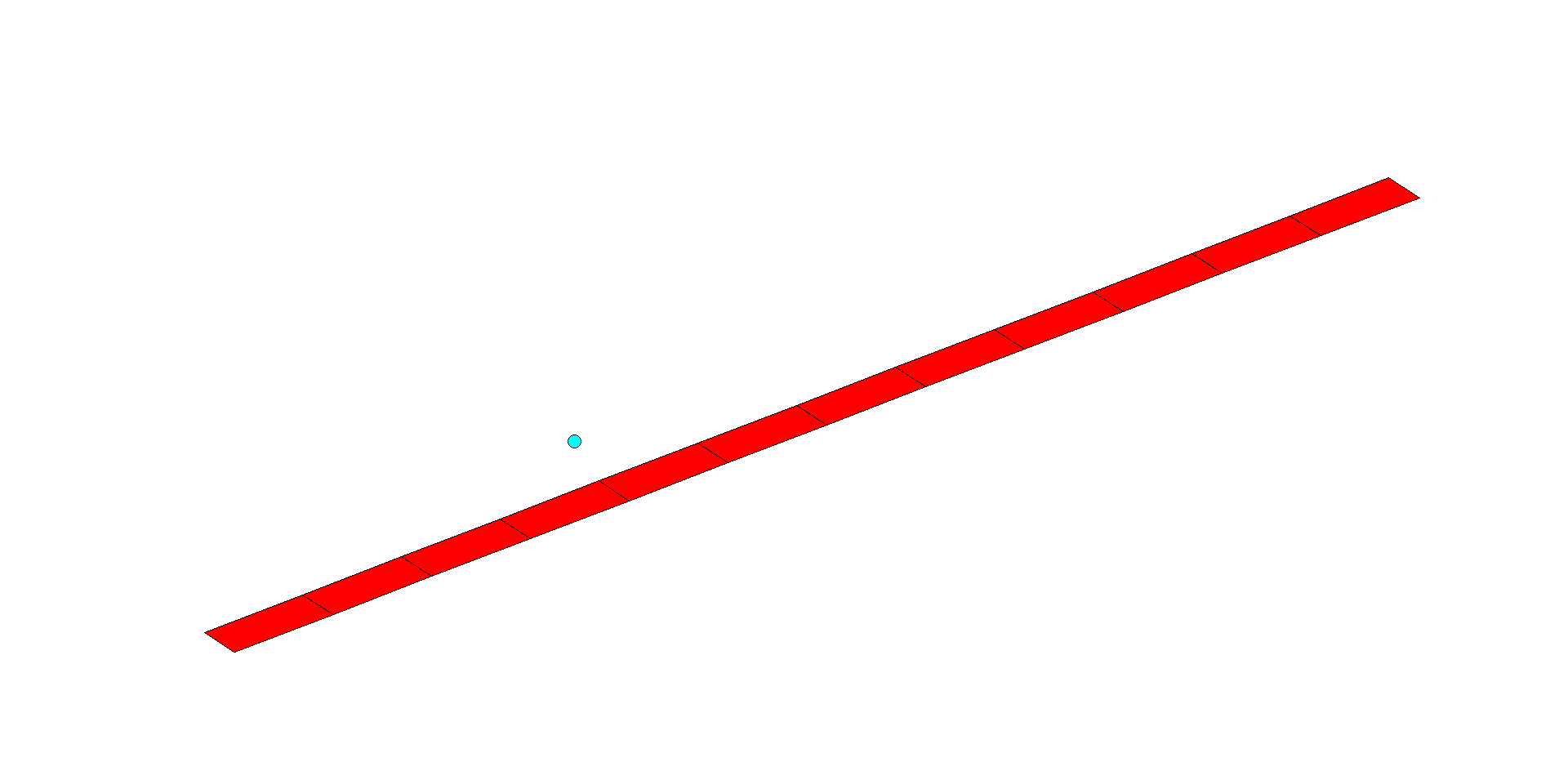
% plot mesh

meshPlot(fem)

% plot constraints

fem.Post.Options.SymbolSize=1;

bilateralElementBcPlot(fem)



**Plotting bilateral constraint at element level**

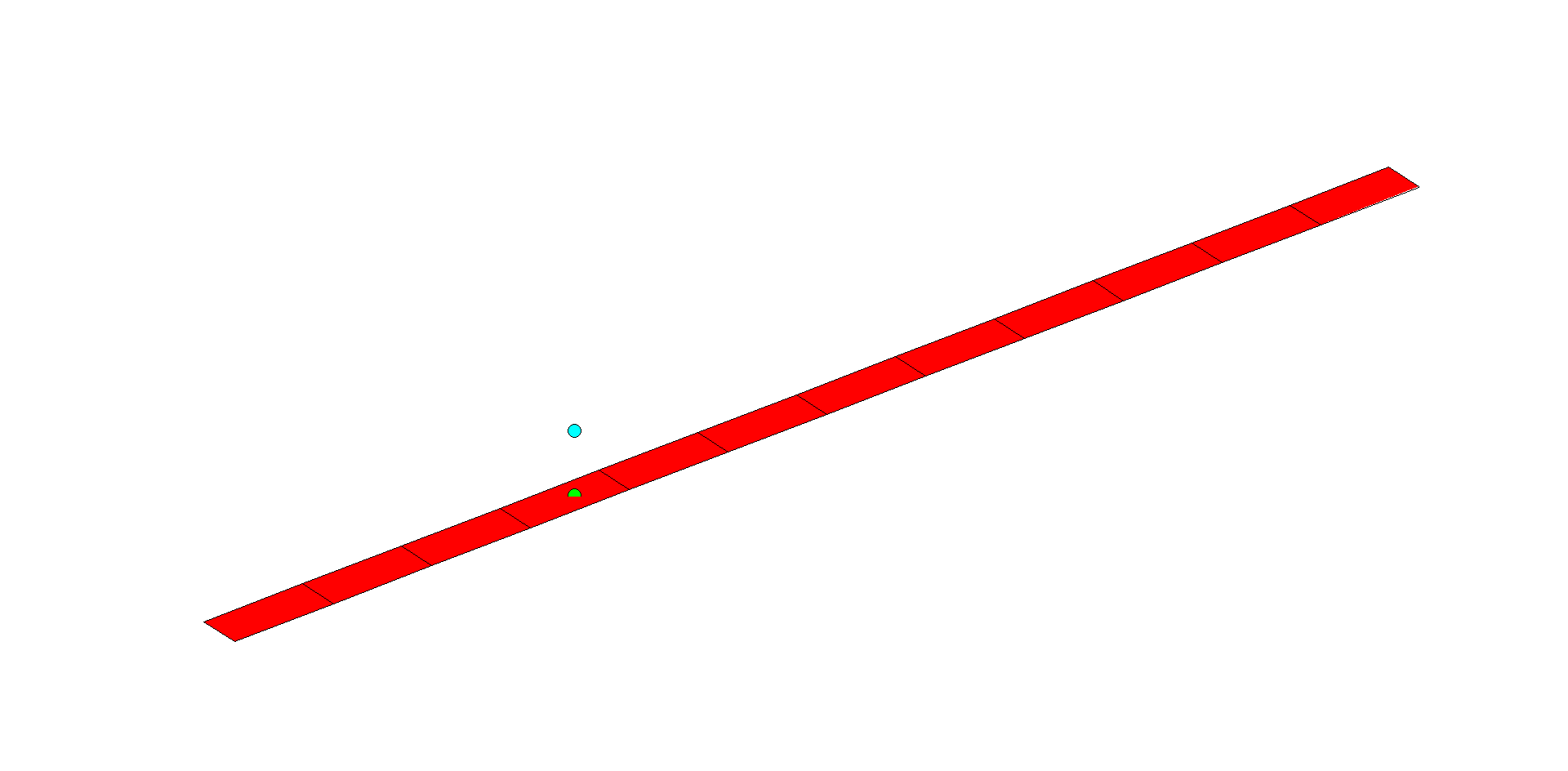
% now compile equations

fem=femRefresh(fem);

% plot projected points

fem.Post.Options.ShowProjection=true;

bilateralElementBcPlot(fem)



**Plotting bilateral constraint at element level - show projected points**

## bilateralNodeBcPlot

*Aim:*

Plot bilateral constraints defined at node level.

*Syntax:*

bilateralNodeBcPlot(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *graphical visualisation*

*Description:*

Plot bilateral constraints defined at node level. For each constraint a sphere is shown. The following property can be set:

* **fem.Post.Options.SymbolSize**: define the size of the 3D sphere (double)

*Dependency:*

* **femPreProcessing**

*Example:*

% define constraint

fem.Boundary.Constraint.Bilateral.Node(1).Node=[7 20];

fem.Boundary.Constraint.Bilateral.Node(1).Reference='cartesian';

fem.Boundary.Constraint.Bilateral.Node(1).DoF=[1 2 3 4 5 6];

fem.Boundary.Constraint.Bilateral.Node(1).Value=[0 0 0 0 0 0];

fem.Boundary.Constraint.Bilateral.Node(1).Physic='shell';

% pre-process model

fem=femPreProcessing(fem);

% define current axis for plot purpose

ax=axes;

fem.Post.Options.ParentAxes=a;

% plot mesh

meshPlot(fem)

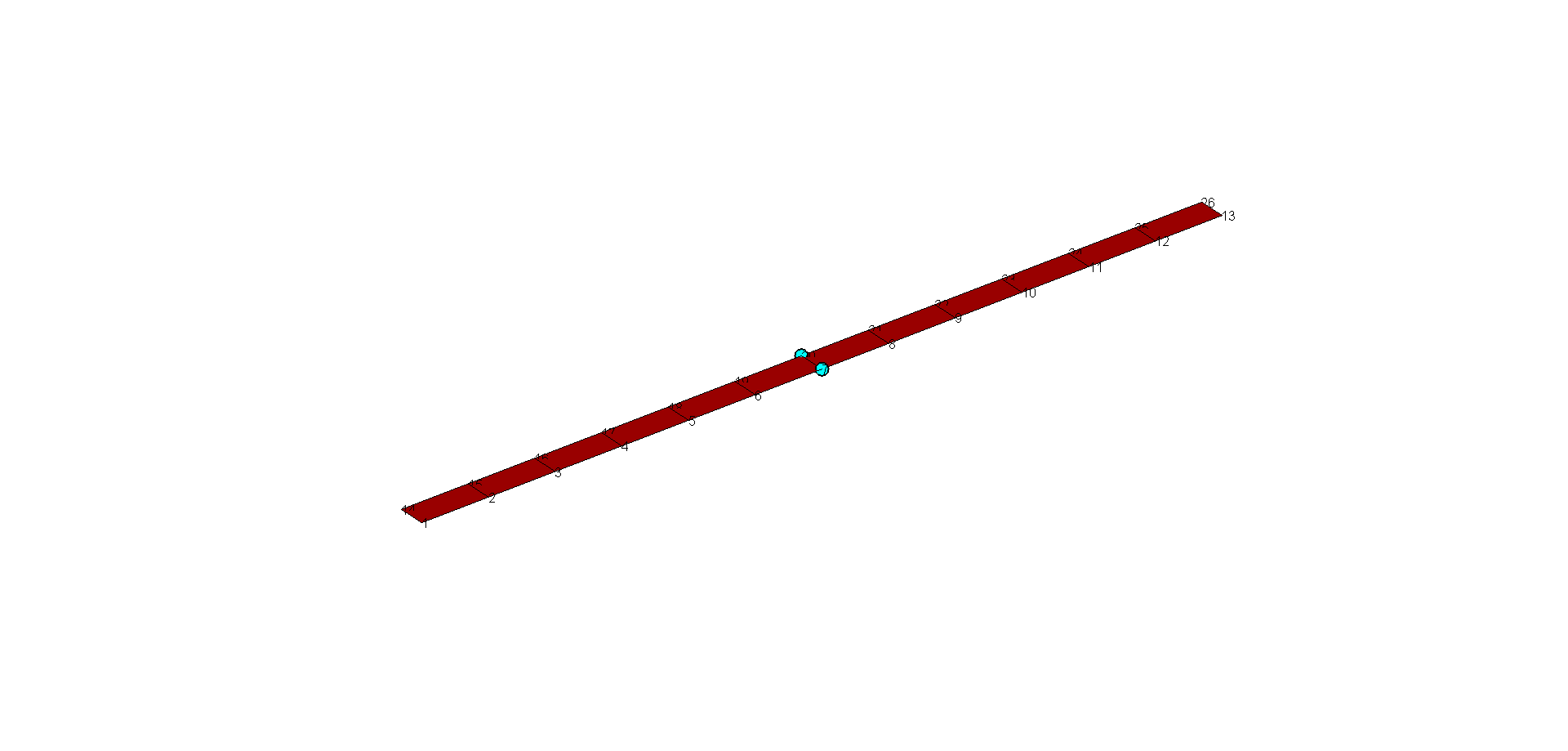
% plot label

labelNodePlot(fem,1)

% plot constraints

fem.Post.Options.SymbolSize=5;

bilateralNodeBcPlot(fem)



**Plotting bilateral constraint at node level**

## boundaryConstraintPlot

*Aim:*

Plot boundary constraint annotations.

*Syntax:*

boundaryConstraintPlot(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *graphical visualisation*

*Description:*

Plot boundary constraint annotations. The following graphical properties can be set:

* **fem.Post.Options.SymbolSize**: define the size of the 3D sphere (double)
* **fem.Post.Options.LengthAxis**: define the length of the arrow (double)
* **fem.Post.Options.ShowProjection**: if "true" than projected points are visualised (true / false). This option can be set only after compiling all equations (**femRefresh**)

Boundary constraints can be visualised based on the following settings:

* **fem.Post.ShowBoundary.BilateralNode**: show bilateral constraints defined at node level (true / false)
* **fem.Post.ShowBoundary.BilateralElement**:show bilateral constraints defined at element level (true / false)
* **fem.Post.ShowBoundary.Unilateral**: show unilateral constraints (true / false)
* **fem.Post.ShowBoundary.PinHole**: show pin-hole constraints (true / false)
* **fem.Post.ShowBoundary.PinSlot**: show pin-slot constraints (true / false)
* **fem.Post.ShowBoundary.RigidLink**: show rigid-link constraints (true / false)
* **fem.Post.ShowBoundary.Dimple**: show dimple pairs (true / false)
* **fem.Post.ShowBoundary.Contact**: show contact pairs (true / false)

*Dependency:*

* **femPreProcessing**
* **femRefresh**

*Example:*

% pre-process model

fem=femPreProcessing(fem);

% compile equations

fem=femRefresh(fem);

% define current axis for plot purpose

ax=axes;

fem.Post.Options.ParentAxes=a;

% set options

fem.Post.Options.SymbolSize=2;

fem.Post.Options.LengthAxis=0.1;

fem.Post.Options.ShowProjection=false;

% show all types of boundary constraints

fem.Post.ShowBoundary.BilateralElement=true;

fem.Post.ShowBoundary.BilateralNode=true;

fem.Post.ShowBoundary.Unilateral=true;

fem.Post.ShowBoundary.Dimple=true;

fem.Post.ShowBoundary.Contact=true;

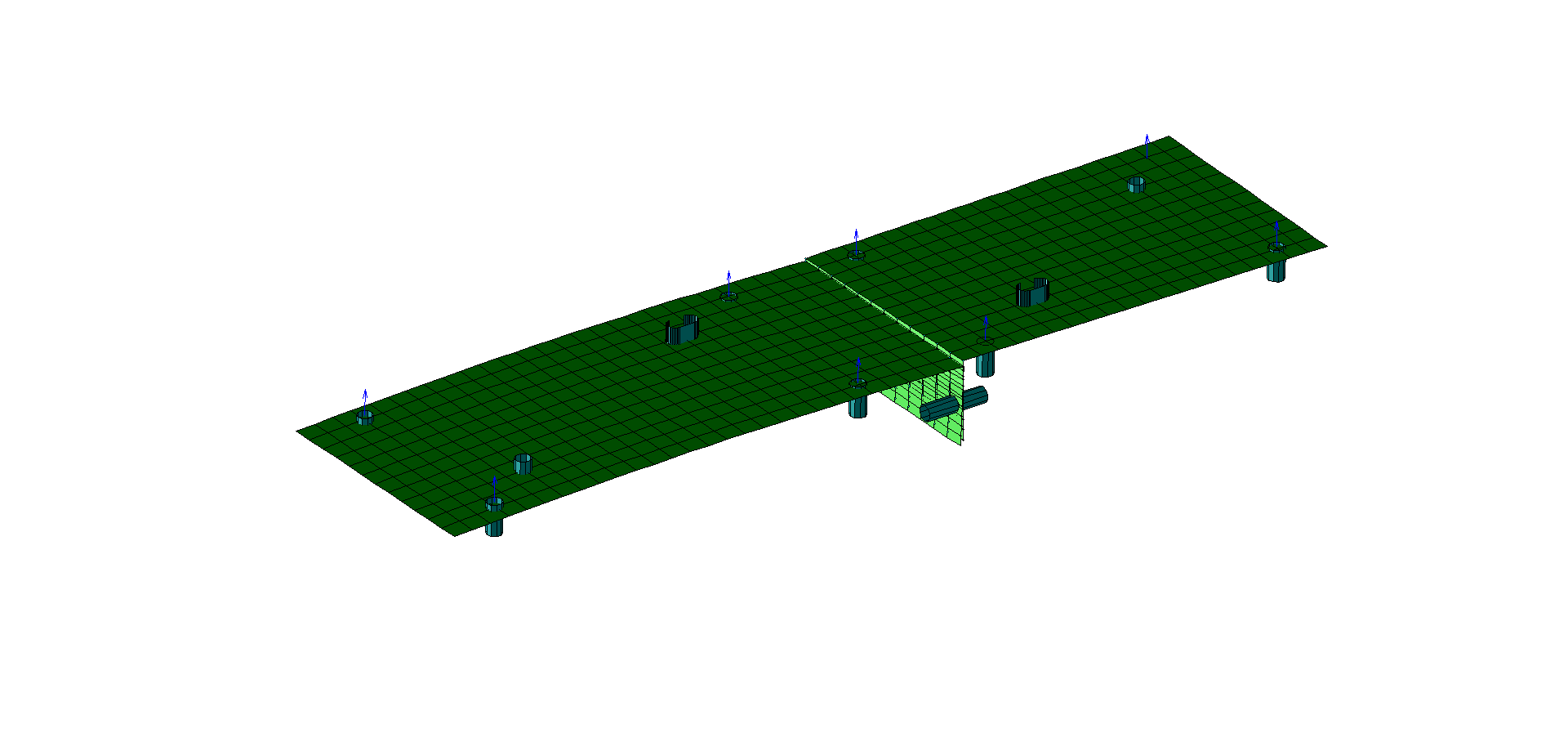
fem.Post.ShowBoundary.RigidLink=true;

fem.Post.ShowBoundary.PinHole=true;

fem.Post.ShowBoundary.PinSlot=true;

% plot boundary

boundaryConstraintPlot(fem)



**Plotting boundary constraints**

## contactPairPlot

*Aim:*

Plot contact pairs.

*Syntax:*

contactPairPlot(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *graphical visualisation*

*Description:*

Plot contact pairs. For each pair a sphere is shown. The following property can be set:

* **fem.Post.Options.SymbolSize**: define the size of the 3D sphere (double)

*Dependency:*

* **femPreProcessing**
* **femRefresh**

*Example:*

% define pair

fem.Boundary.ContactPair(1).Master=2;

fem.Boundary.ContactPair(1).MasterFlip=false;

fem.Boundary.ContactPair(1).Slave=1;

fem.Boundary.ContactPair(1).SearchDist=3;

fem.Boundary.ContactPair(1).Offset=0.0;

fem.Boundary.ContactPair(1).Physic='shell';

fem.Boundary.ContactPair(1).Enable=true;

% pre-process model

fem=femPreProcessing(fem);

% compile equations

fem=femRefresh(fem);

% define current axis for plot purpose

ax=axes;

fem.Post.Options.ParentAxes=a;

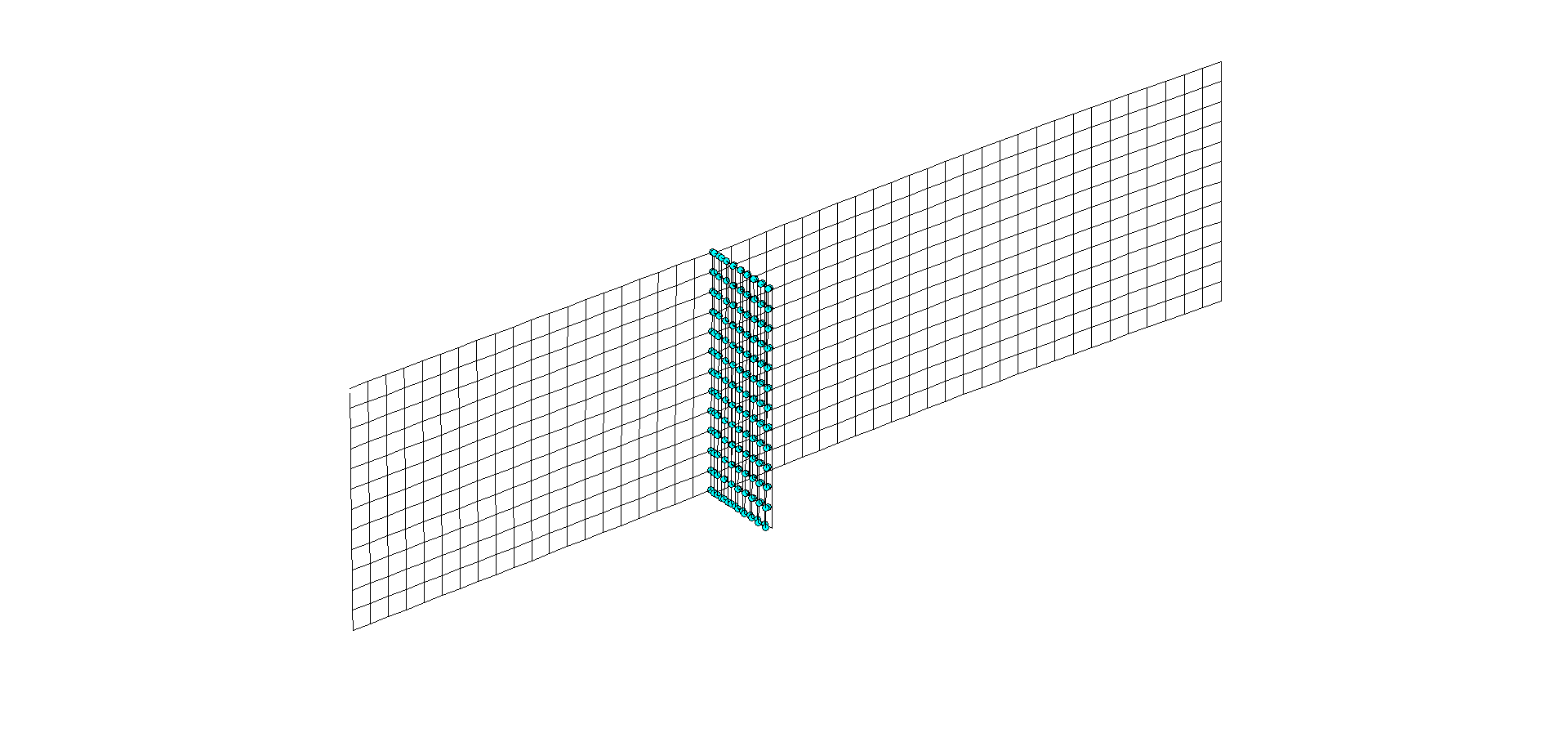
% plot mesh

meshPlot(fem)

% define options and plot

fem.Post.Options.SymbolSize=10;

contactPairPlot(fem)



**Plotting contact pairs**

## contourPlot

*Aim:*

Contour plot.

*Syntax:*

contourPlot(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *graphical visualisation*

*Description:*

Contour plot. The following properties can be set:

* **fem.Post.Contour.Domain**: domain identification number (integer)
* **fem.Post.Contour.ContourVariable**:variable to plot ('u'=displacement along X; 'v'=displacement along Y; 'w'=displacement along Z; 'alfa'=rotation around X; 'beta'=rotation around Y; 'gamma'=rotation around Z; 'gap'=gap distribution; 'user'=user expression)
* **fem.Post.Contour.ContactPair**: identification number of contact pair used to plot gap distribution (integer)
* **fem.Post.Contour.MaxRange**: maximum data range (double)
* **fem.Post.Contour.MinRange**: minimum data range (double)
* **fem.Post.Contour.Resolution**: graphical resolution (integer)
* **fem.Post.Contour.Deformed**: if "true" than the deformed shape is visualised (true / false)
* **fem.Post.Contour.ScaleFactor**: scaling magnitude used to plot deformed shape (double)

When using the "user" option, contour data has to be stored in the field **fem.Sol.UserExp**.

*Dependency:*

* **femPreProcessing**
* **femRefresh**
* **femSolve**

*Example:*

% pre-process model

fem=femPreProcessing(fem);

% compile equations

fem=femRefresh(fem);

% solve model

fem=femSolve(fem);

% define current axis for plot purpose

ax=axes;

fem.Post.Options.ParentAxes=a;

% plot nominal mesh

meshPlot(fem)

% define options for domain id=2

fem.Post.Contour.Domain=2;

fem.Post.Contour.ContourVariable='gap'; % gap distribution

fem.Post.Contour.ContactPair=1;

fem.Post.Contour.MaxRange=100;

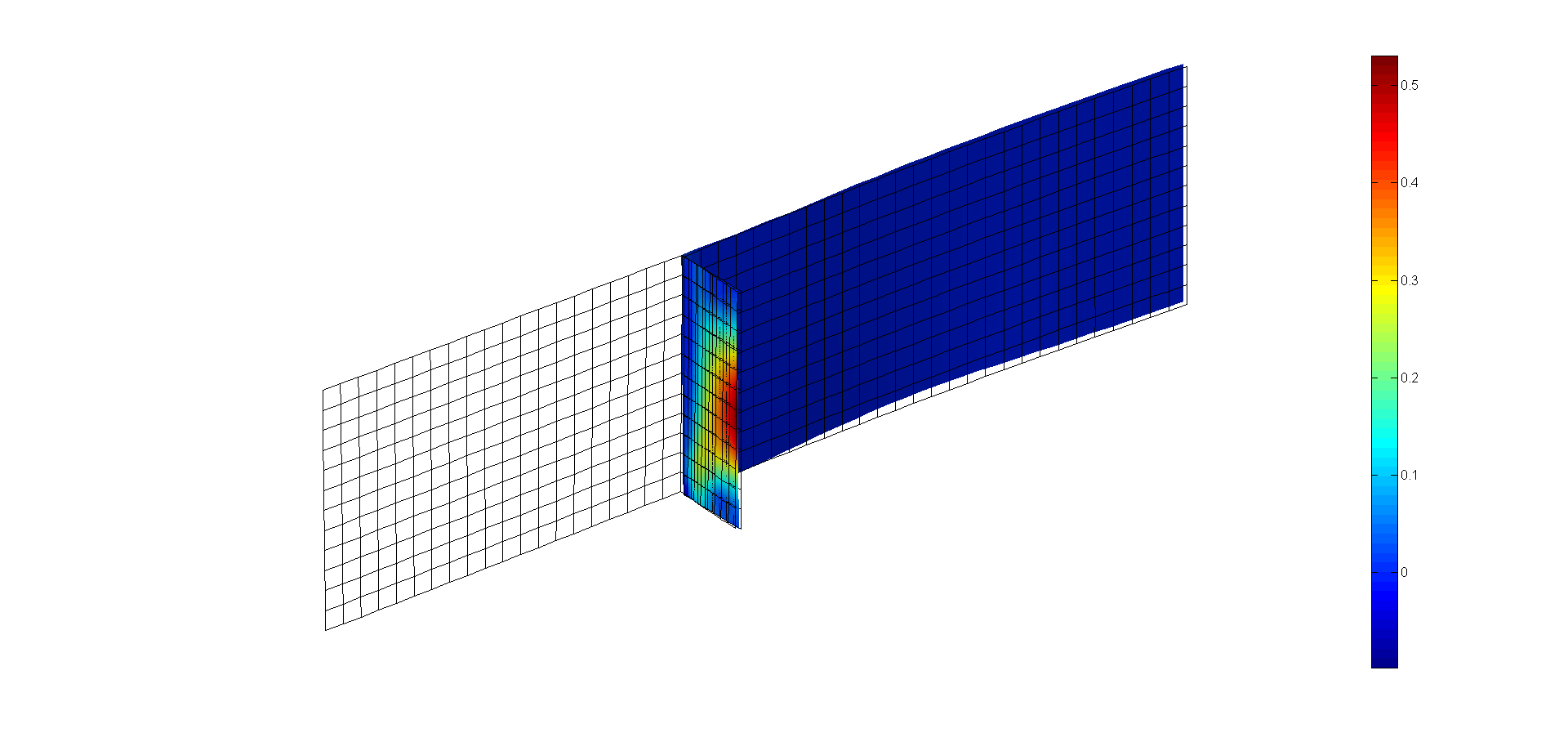
fem.Post.Contour.MinRange=-100;

fem.Post.Contour.Deformed=true;

fem.Post.Contour.ScaleFactor=1;

fem.Post.Contour.Resolution=1;

contourPlot(fem)



**Gap distribution**

% now, plot displacement along y (scale magnitude=10)

fem.Post.Contour.Domain=2;

fem.Post.Contour.ContourVariable='v';

fem.Post.Contour.MaxRange=100;

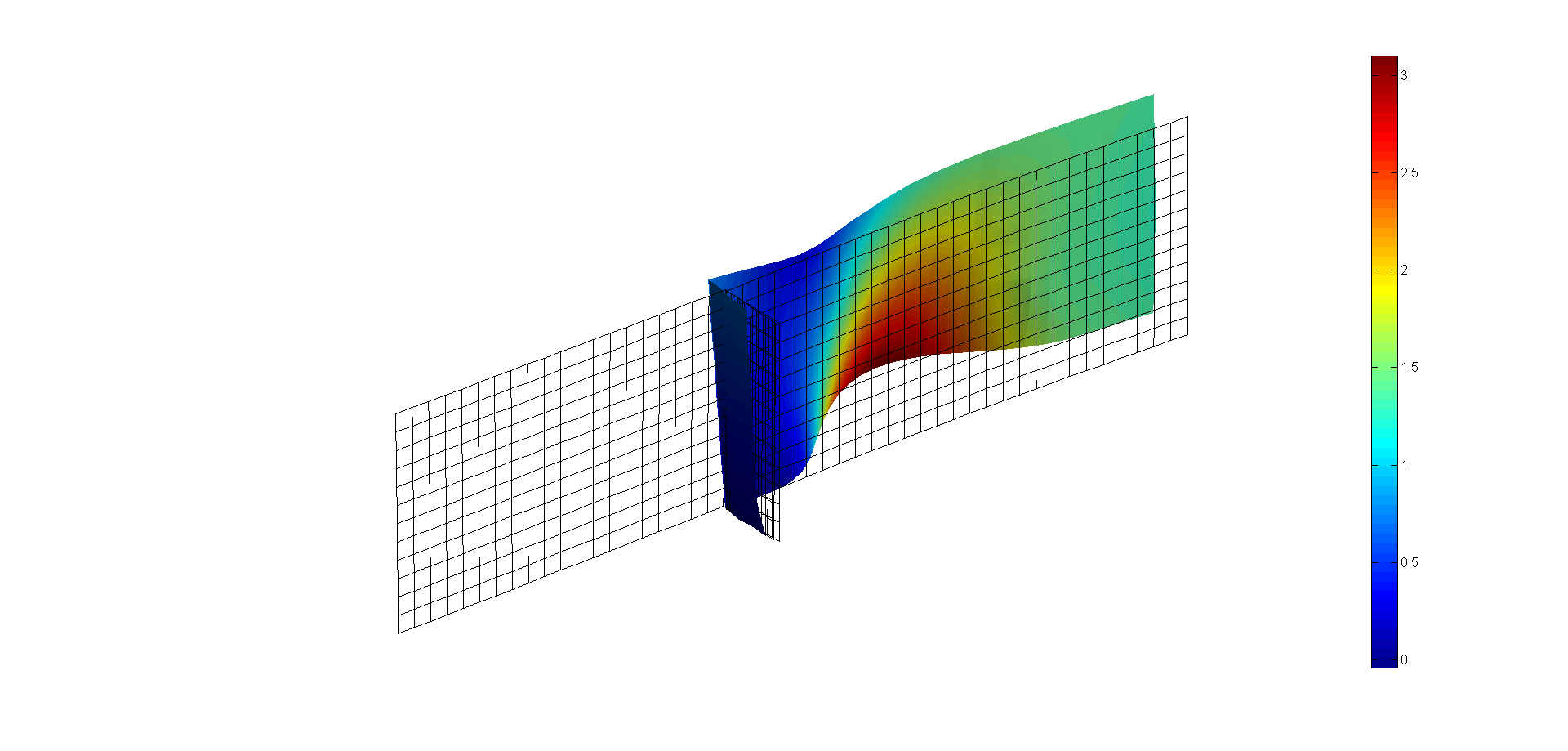
fem.Post.Contour.MinRange=-100;

fem.Post.Contour.Deformed=true;

fem.Post.Contour.ScaleFactor=10;

fem.Post.Contour.Resolution=1;

contourPlot(fem)



**Displacement along Y - magnified 10 times**

% generate a local noise for every node and plot the deviation field

nnode=size(fem.xMesh.Node.Coordinate,1); % no. of nodes

dev=randn(1,nnode)\*5;

% plot

fem.Post.Contour.Domain=2;

fem.Post.Contour.ContourVariable='user';

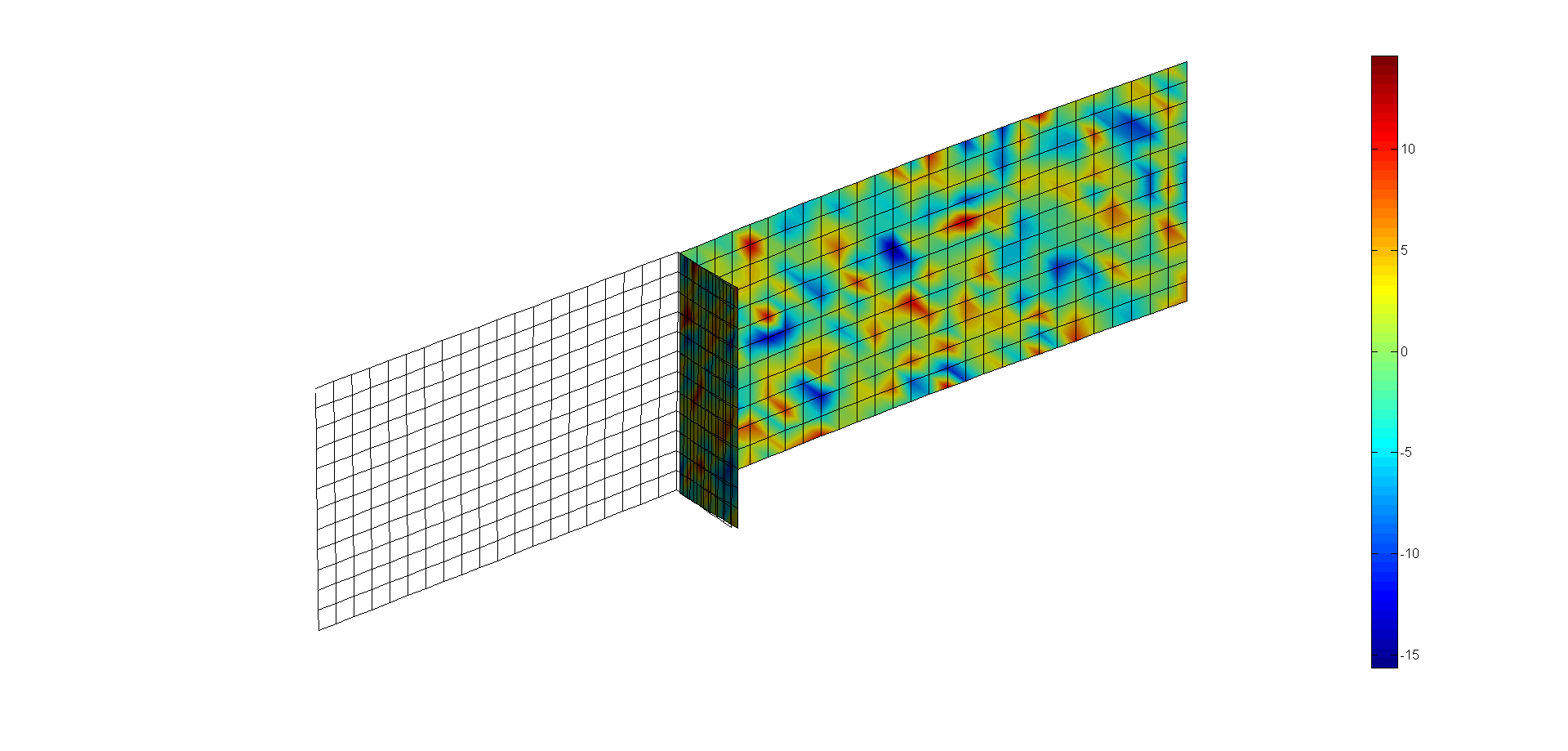
fem.Post.Contour.MaxRange=100;

fem.Post.Contour.MinRange=-100;

fem.Post.Contour.Deformed=false;

fem.Post.Contour.Resolution=1;

contourPlot(fem)



**Plotting user data field**

## dimpleBcPlot

*Aim:*

Plot dimple pairs.

*Syntax:*

dimpleBcPlot(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *graphical visualisation*

*Description:*

Plot dimple pairs. For each dimple a sphere is shown. The following properties can be set:

* **fem.Post.Options.SymbolSize**: define the size of the 3D sphere (double)
* **fem.Post.Options.ShowProjection**: if "true" than projected points are visualised (true / false). This option can be set only after compiling all equations (**femRefresh**)

*Dependency:*

* **femPreProcessing**
* **femRefresh**

*Example:*

% define constraint

D=[0 -12 14;0 -12 -14];

for i=1:size(D,1)

fem.Boundary.DimplePair(i).Pm=D(i,:);

fem.Boundary.DimplePair(i).Master=1;

fem.Boundary.DimplePair(i).MasterFlip=false;

fem.Boundary.DimplePair(i).Slave=2;

fem.Boundary.DimplePair(i).SearchDist=5;

fem.Boundary.DimplePair(i).Height=0.1;

fem.Boundary.DimplePair(i).Physic='shell';

end

% pre-process model

fem=femPreProcessing(fem);

% compile equations

fem=femRefresh(fem);

% define current axis for plot purpose

ax=axes;

fem.Post.Options.ParentAxes=a;

% plot mesh

meshPlot(fem)

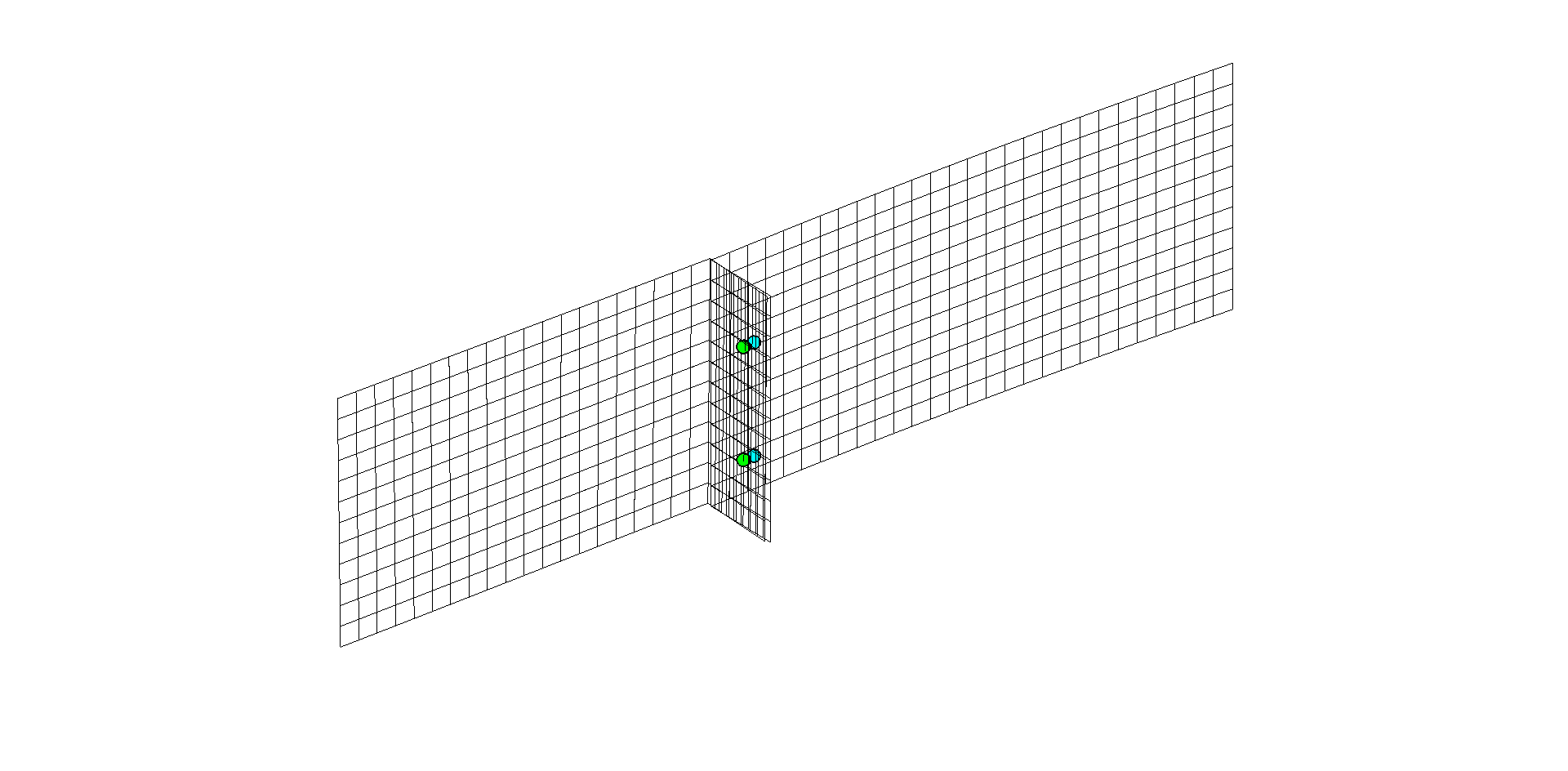
% define options

fem.Post.Options.SymbolSize=10;

fem.Post.Options.ShowProjection=true;

% plot

dimpleBcPlot(fem)



**Plotting dimple pairs**

## getInterpolationData

*Aim:*

Interpolate model based on key points.

*Syntax:*

fem=getInterpolationData(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *fem*: fem structure

*Description:*

Interpolate model based on key points. The following properties can be set:

* **fem.Post.Interp.Pm**: (x, y, z) coordinate of key points used for interpolation (double)
* **fem.Post.Interp.Domain**: domain identification number (integer)
* **fem.Post.Interp.SearchDist**: searching distance (double). If the distance from any key point to the selected domain is greater than "SearchDist" than no output data is saved.
* **fem.Post.Interp.InterpVariable**: variable to interpolate ('u'=displacement along X; 'v'=displacement along Y; 'w'=displacement along Z; 'alfa'=rotation around X; 'beta'=rotation around Y; 'gamma'=rotation around Z; 'gap'=gap distribution; 'user'=user expression)
* **fem.Post.Interp.ContactPair**: identification number of contact pair used for interpolation (integer)

This command is useful to extract output data based on the (x, y, z) location of a set of key points. Interpolation data are automatically stored in the field **fem.Post.Interp.Data**. Moreover, the field **fem.Post.Interp.Flag** contains "true" when the key point is properly interpolated (based on the "SearchDist" distance) and "false" otherwise.

*Dependency:*

* **femPreProcessing**
* **femRefresh**
* **femSolve**

*Example:*

% pre-process model

fem=femPreProcessing(fem);

% define options

fem.Post.Interp.InterpVariable='gap';

fem.Post.Interp.Domain=1;

fem.Post.Interp.ContactPair=1;

fem.Post.Interp.SearchDist=5;

% define key points (Pm)

%------------

.

fem.Post Interp.Pm=Pm;

% get data

fem=getInterpolationData(fem);

==> fem.Post.Interp.Data

==> fem.Post.Interp.Flag

## getGapsVariable

*Aim:*

Calculate gap variables on active contact pairs.

*Syntax:*

fem=getGapsVariables(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *fem*: fem structure

*Description:*

This commands calculates the gap variables using available contact pairs. The following properties are automatically obtained from contact pairs:

* **Master**: define the identification number of the master domain (integer).
* **MasterFlip**: if "true" than the normal vector (Nm) of every element belonging to the master domain is flipped (true / false).
* **Slave**: define the identification number of the slave domain (integer).
* **SearchDist**: used to calculate the closest surface for any point on the slave surface (double).
* **Offset**: define an initial gap for the contact pair (double).

Gap variables might be calculated both on the reference geometry or on the deformed geometry. To do that, please set the following option **fem.Options.GapFrame**:

* “ref”: reference frame.
* “def”: deformed frame.

Output data are stored in **fem.Sol.Gap.Gap**.

*Dependency:*

* **femPreProcessing**
* **femRefresh**

*Example:*

wdir='… \Source Files';

path(path,wdir);

fem=femInit(fem);

% define option

fem.Options.GapFrame='ref'; % used ref frame

% define contact pair

fem.Boundary.ContactPair(1).Master=1;

fem.Boundary.ContactPair(1).MasterFlip=false;

fem.Boundary.ContactPair(1).Slave=2;

fem.Boundary.ContactPair(1).SearchDist=3.0;

fem.Boundary.ContactPair(1).Offset=0.0;

fem.Boundary.ContactPair(1).Physic='shell';

fem.Boundary.ContactPair(1).Enable=true;

fem.Boundary.ContactPair(1).Frame='ref';

% pre-process model

fem=femPreProcessing(fem);

% refresh equations

fem=femRefresh(fem);

% calculate gap variables

fem=getGapsVariable(fem);

## labelElementPlot

*Aim:*

Plot labels for all elements belonging to the selected component.

*Syntax:*

labelElementPlot(fem, idcmp)

*Input:*

* *fem*: fem structure
* *idcmp*: component identification number (integer)

*Output:*

* *graphical visualisation*

*Description:*

Plot label annotations of all elements belonging to the selected component. The following properties can be set:

* **fem.Post.Options.LabelSize**: define the font size (double)

*Dependency:*

* **mesh model has been loaded**

*Example:*

% pre-process model

fem=femPreProcessing(fem);

% define current axis for plot purpose

ax=axes;

fem.Post.Options.ParentAxes=a;

% plot mesh

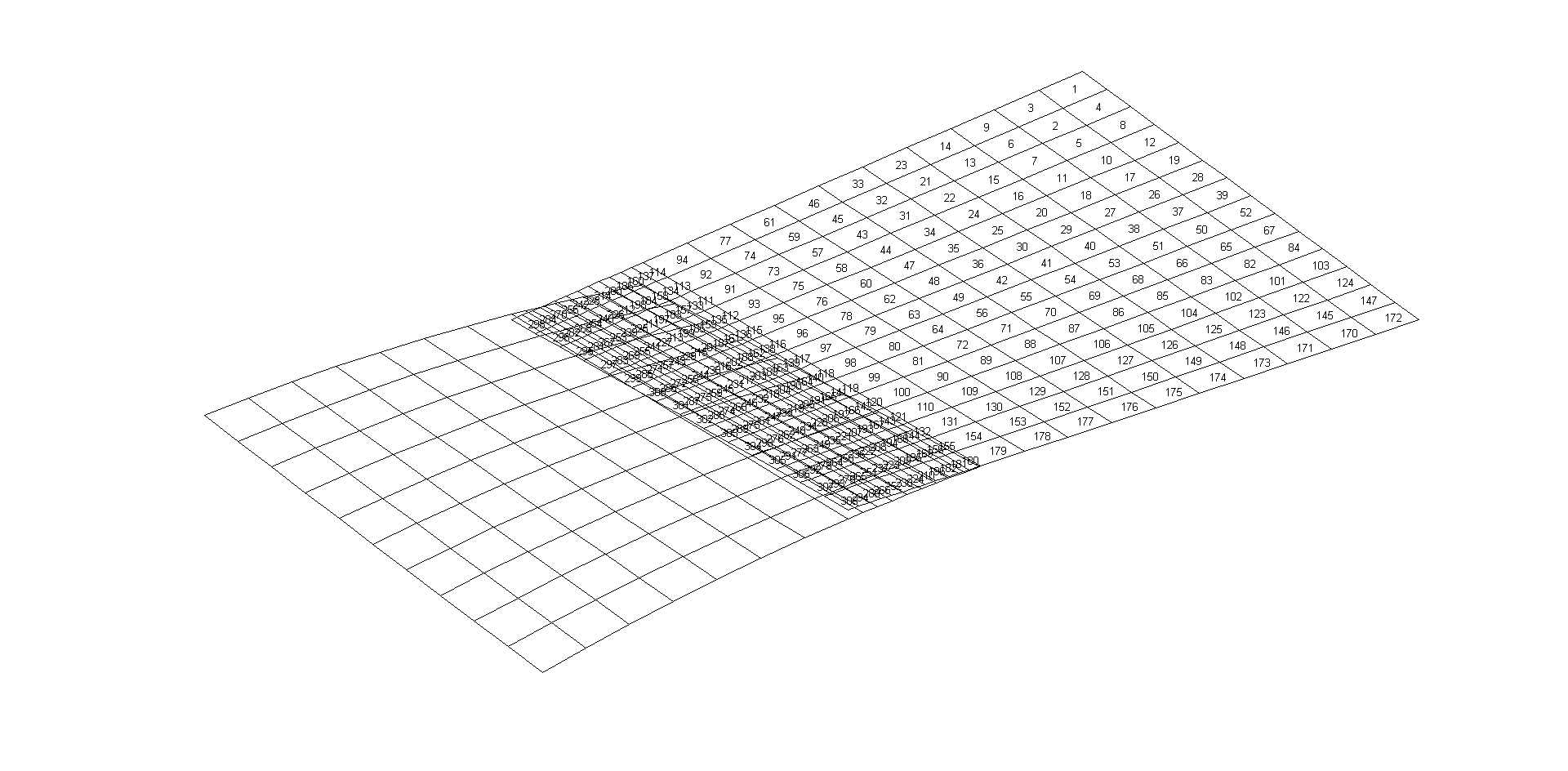
fem.Post.Options.ShowPatch=false;

meshPlot(fem)

% plot element labels for component=1

fem.Post.Options.LabelSize=8;

labelElementPlot(fem, 1)



**Plotting label annotation**

## labelNodePlot

*Aim:*

Plot labels for all nodes belonging to the selected component.

*Syntax:*

labelNodePlot(fem, idcmp)

*Input:*

* *fem*: fem structure
* *idcmp*: component identification number (integer)

*Output:*

* *graphical visualisation*

*Description:*

Plot label annotations of all nodes belonging to the selected component. The following properties can be set:

* **fem.Post.Options.LabelSize**: define the font size (double)

*Dependency:*

* **mesh model has been loaded**

*Example:*

% pre-process model

fem=femPreProcessing(fem);

% define current axis for plot purpose

ax=axes;

fem.Post.Options.ParentAxes=a;

% plot mesh

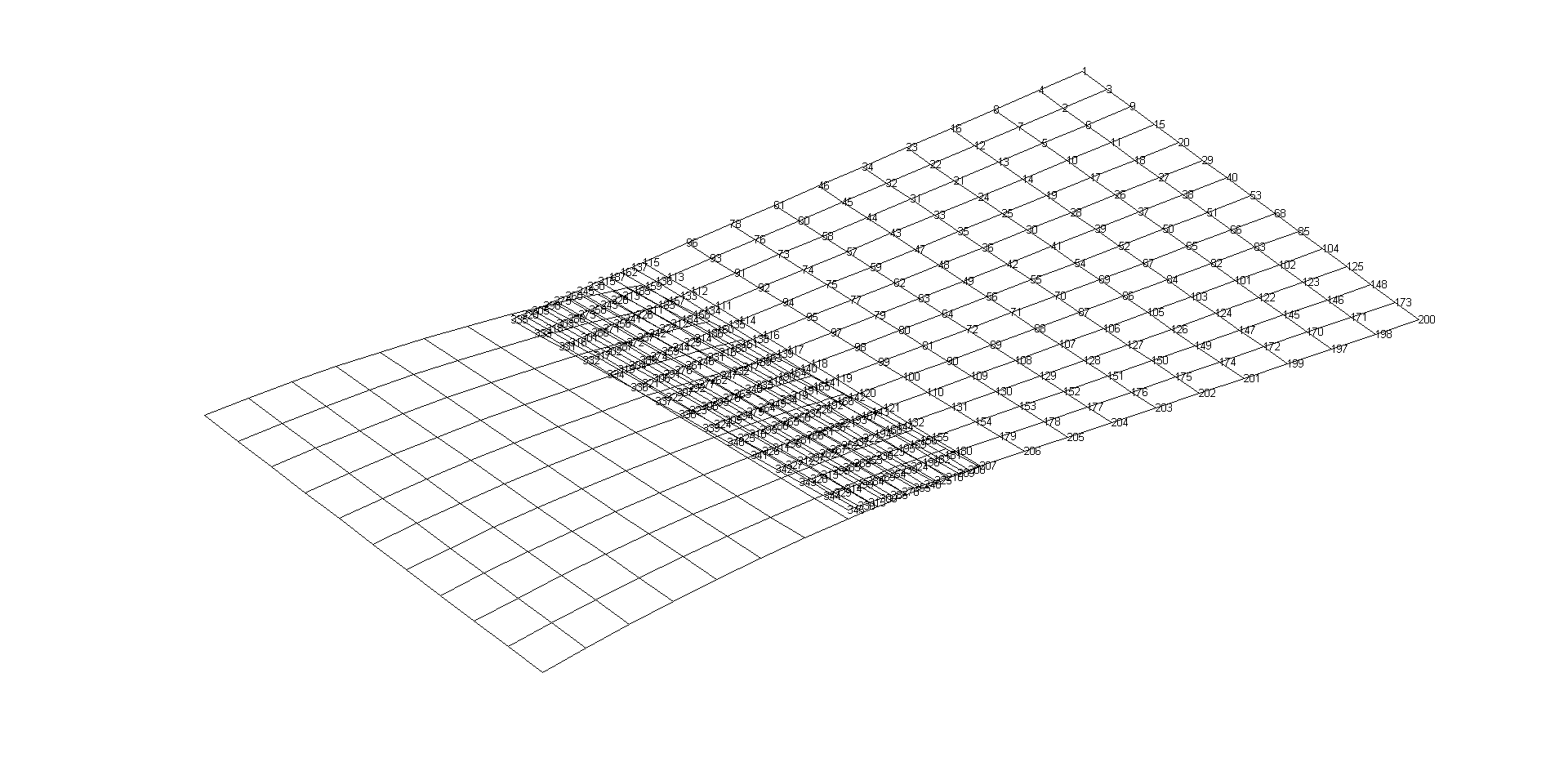
fem.Post.Options.ShowPatch=false;

meshPlot(fem)

% plot node labels for component=1

fem.Post.Options.LabelSize=8;

labelNodePlot(fem, 1)



**Plotting label annotation**

## meshPlot

*Aim:*

Plot nominal mesh model.

*Syntax:*

meshPlot(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *graphical visualisation*

*Description:*

Plot the nominal mesh model. The following properties can be set:

* **fem.Post.Options.ShowPatch**: if "true" than the mesh is visualised (true / false)
* **fem.Post.Options.ColorPatch**: define the patch colour ('g'=green; 'r'=red; 'k'=black; 'c'=cyan)
* **fem.Post.Options.ShowEdge**: if "true" than the edges of the mesh model are visualised (true / false)
* **fem.Post.Options.ColorEdge**: define the edge colour ('g'=green; 'r'=red; 'k'=black; 'c'=cyan)

*Dependency:*

* **mesh model has been loaded**

*Example:*

% pre-process model

fem=femPreProcessing

% define current axis for plot purpose

ax=axes;

fem.Post.Options.ParentAxes=ax;

% set patch properties

fem.Post.Options.ShowPatch=true;

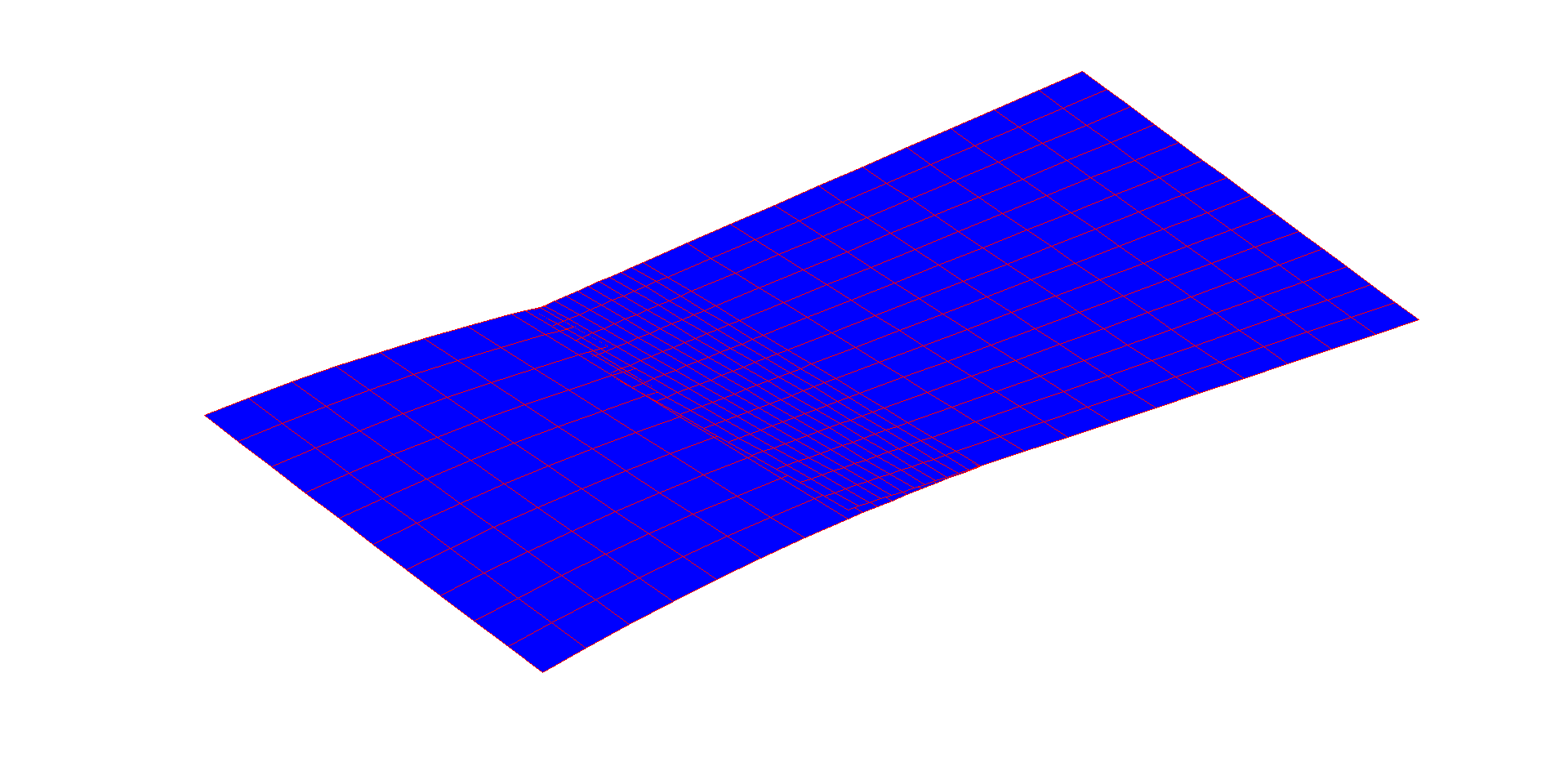
fem.Post.Options.ColorPatch='b';

fem.Post.Options.ShowEdge=true;

fem.Post.Options.ColorEdge='r';

% plot mesh

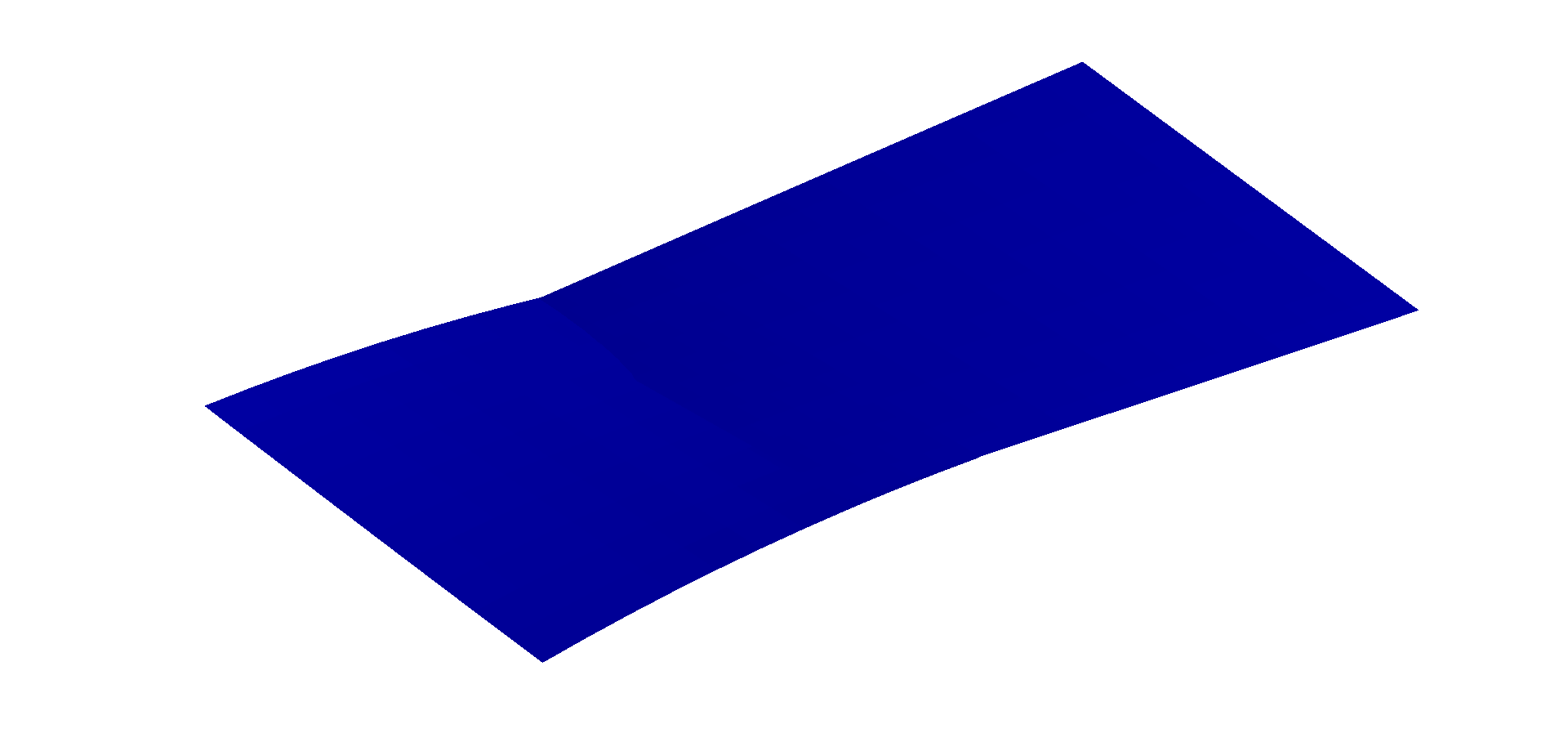
meshPlot(fem)



**Mesh visualisation with edges**

% now, hide edges

fem.Post.Options.ShowEdge=false;



**Mesh visualisation without edges**

## meshComponentPlot

*Aim:*

Plot nominal mesh model of selected component.

*Syntax:*

meshComponentPlot(fem, idcmp)

*Input:*

* *fem*: fem structure
* *idcmp*: component identification number (integer)

*Output:*

* *graphical visualisation*

*Description:*

Plot the nominal mesh model of the selected component. The following properties can be set:

* **fem.Post.Options.ShowPatch**: if "true" than the mesh is visualised (true / false)
* **fem.Post.Options.ColorPatch**: define the patch colour ('g'=green; 'r'=red; 'k'=black; 'c'=cyan)
* **fem.Post.Options.ShowEdge**: if "true" than the edges of the mesh model are visualised (true / false)
* **fem.Post.Options.ColorEdge**: define the edge colour ('g'=green; 'r'=red; 'k'=black; 'c'=cyan)

*Dependency:*

* **mesh model has been loaded**

*Example:*

% pre-process model

fem=femPreProcessing(fem);

% define current axis for plot purpose

ax=axes;

fem.Post.Options.ParentAxes=a;

% set property for component=1

fem.Post.Options.ShowPatch=true;

fem.Post.Options.ColorPatch='b';

fem.Post.Options.ShowEdge=false;

fem.Post.Options.ColorEdge='r';

meshComponentPlot(fem,1)

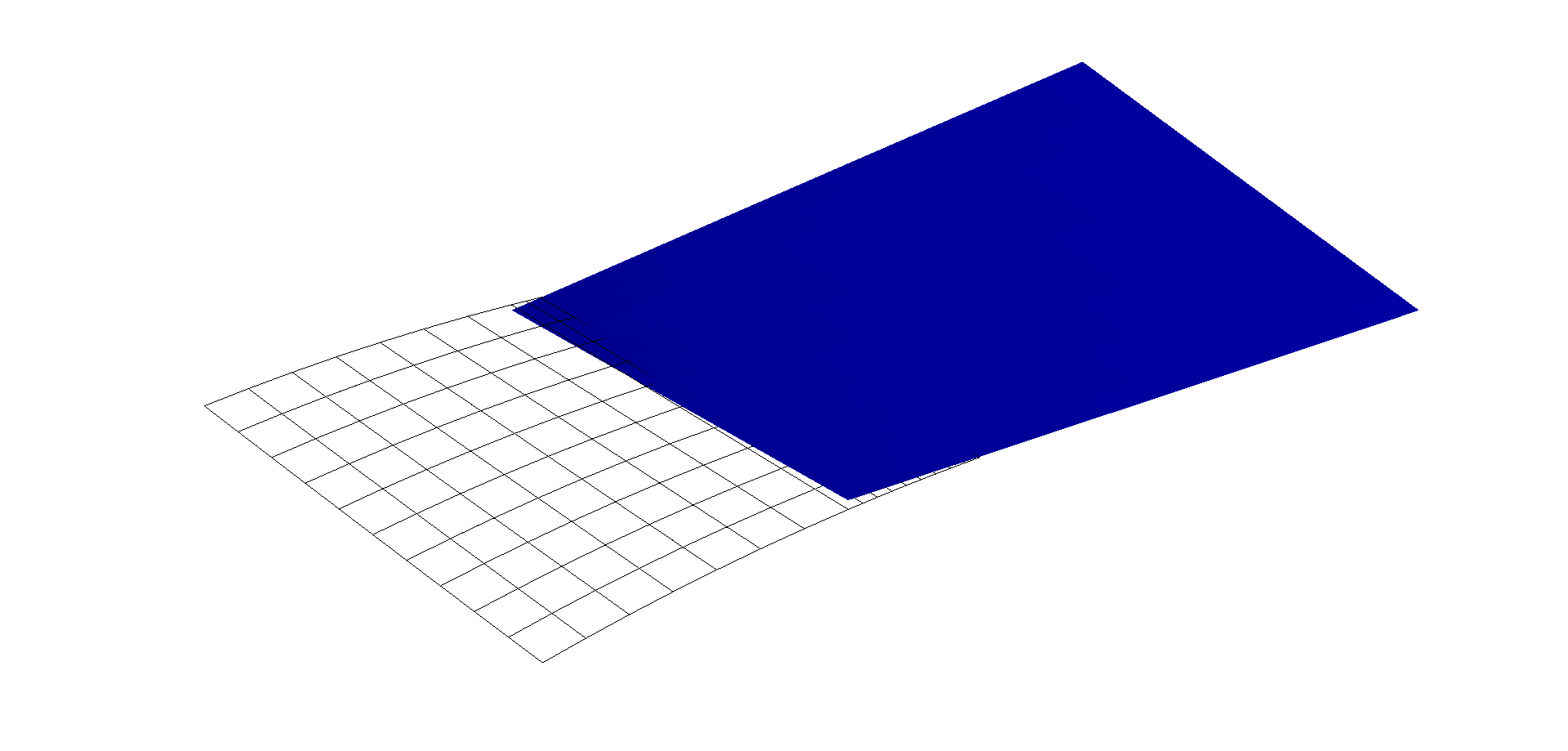
% set property for component=2

fem.Post.Options.ShowPatch=false;

fem.Post.Options.ShowEdge=true;

fem.Post.Options.ColorEdge='k';

meshComponentPlot(fem,2)



**Mesh visualisation of selected components**

## normalElementPlot

*Aim:*

Plot normal vectorsof elements belonging to the selected component.

*Syntax:*

normalElementPlot(fem, idcmp)

*Input:*

* *fem*: fem structure
* *idcmp*: component identification number (integer)

*Output:*

* *graphical visualisation*

*Description:*

Plot the normal vectors of elements belonging to selected components. The following properties can be set:

* **fem.Post.Options.LengthAxis**: define the length of visualised vectors (double)
* **fem.Post.Options.SubSampling**: define the percentage of vectors to plot (if "1" than all vectors are plotted). Vectors are randomly selected (double).

*Dependency:*

* **femPreProcessing**

*Example:*

% pre-process model

fem=femPreProcessing(fem);

% define current axis for plot purpose

ax=axes;

fem.Post.Options.ParentAxes=ax;

% plot normal vectors of 40% elements

fem.Post.Options.LengthAxis=2;

fem.Post.Options.SubSampling=0.4;

normalElementPlot(fem,1)

normalElementPlot(fem,2)

% plot component 1

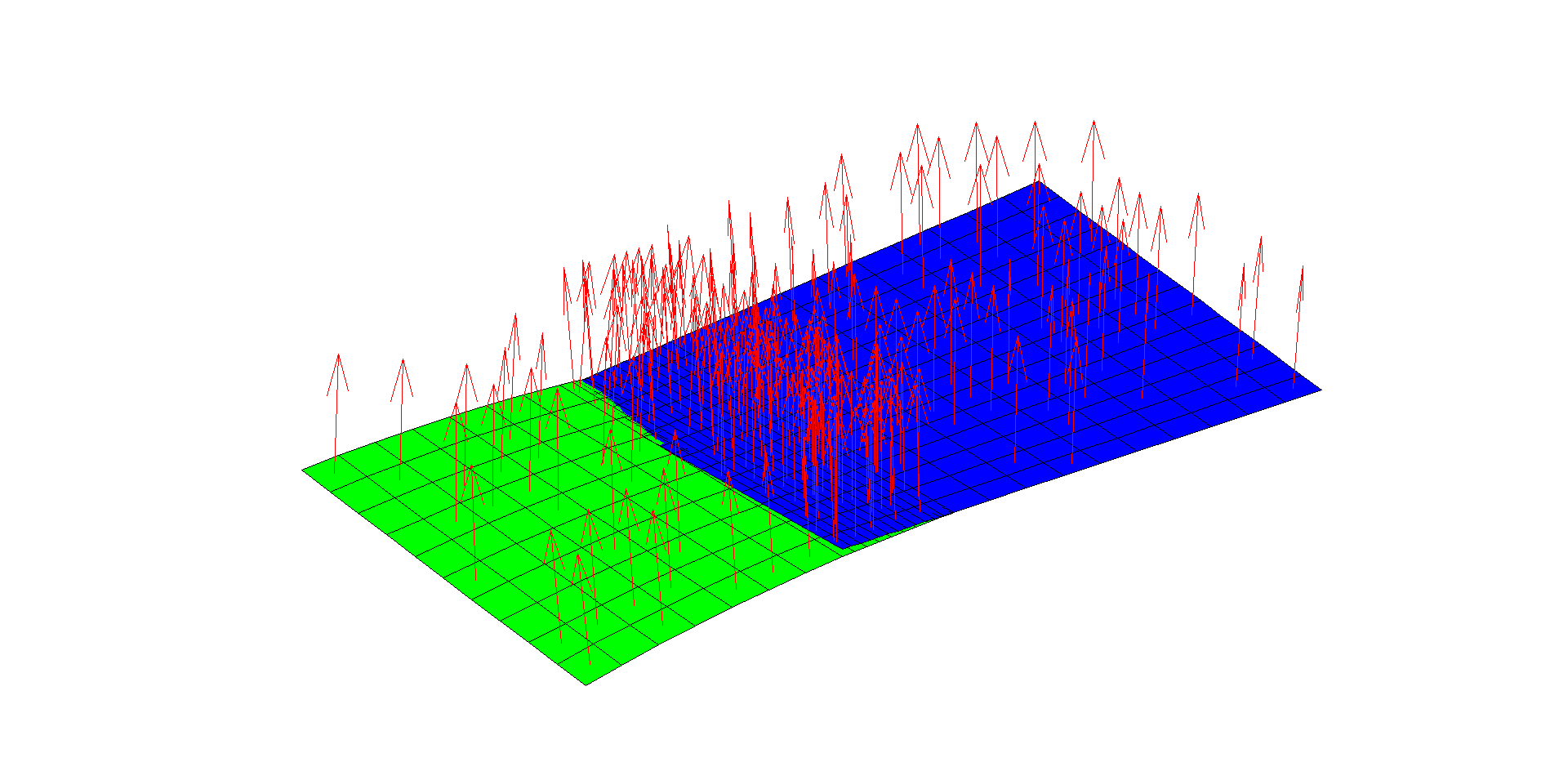
fem.Post.Options.ColorPatch='b';

meshComponentPlot(fem,1)

% plot component 2

fem.Post.Options.ColorPatch='g';

meshComponentPlot(fem,2)



**Plotting normal vectors**

## normalNodePlot

*Aim:*

Plot normal vectors of nodes belonging to the selected component.

*Syntax:*

normalNodePlot(fem, idcmp)

*Input:*

* *fem*: fem structure
* *idcmp*: component identification number (integer)

*Output:*

* *graphical visualisation*

*Description:*

Plot normal vectors of nodes belonging to the selected component. The following properties can be set:

* **fem.Post.Options.LengthAxis**: define the length of visualised vectors (double)
* **fem.Post.Options.SubSampling**: define the percentage of vectors to plot (if "1" than all vectors are plotted). Vectors are randomly selected (double).

*Dependency:*

* **femPreProcessing**

*Example:*

% pre-process model

fem=femPreProcessing(fem);

% define current axis for plot purpose

ax=axes;

fem.Post.Options.ParentAxes=a;

% plot normal vectors of 100% elements

fem.Post.Options.LengthAxis=2;

fem.Post.Options.SubSampling=1;

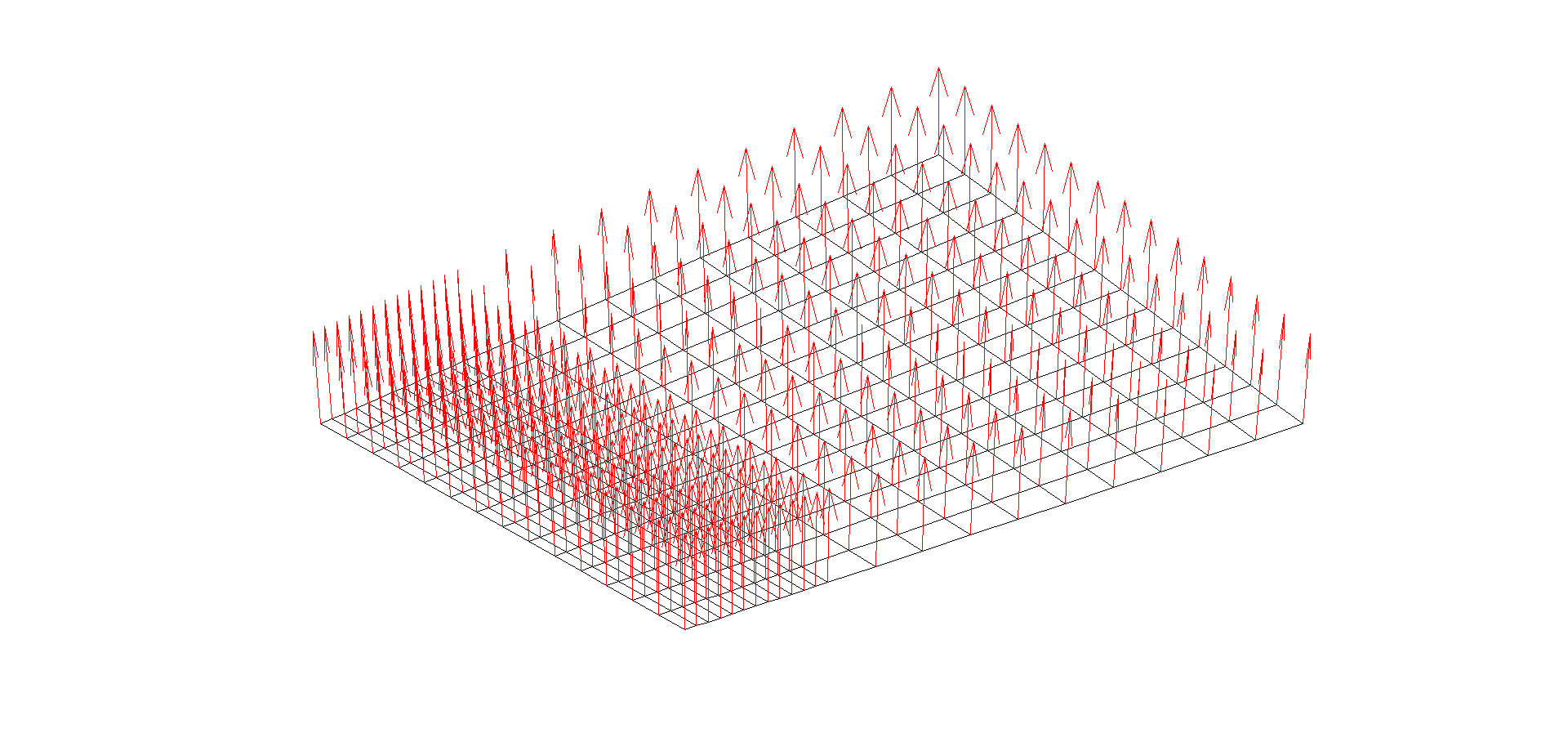
fem.Post.Options.ShowPatch=false;

fem.Post.Options.ShowEdge=true;

fem.Post.Options.ColorEdge='k';

normalNodePlot(fem,1)

meshComponentPlot(fem,1)



**Plotting normal vectors**

## pinholeBcPlot

*Aim:*

Plot pin-hole constraint.

*Syntax:*

pinholeBcPlot(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *graphical visualisation*

*Description:*

Plot pin-hole constraints. For each constraint a cylinder is shown. The following property can be set:

* **fem.Post.Options.SymbolSize**: define the size of the cylinder (double)

*Dependency:*

* **femPreProcessing**

*Example:*

% define constraint

Ph=[-80 0 15; 110 0 -15];

count=1;

for i=1:size(Ph,1)

fem.Boundary.Constraint.PinHole(count).Pm=Ph(i,:);

fem.Boundary.Constraint.PinHole(count).Nm=[0 1 0];

fem.Boundary.Constraint.PinHole(count).Domain=i;

fem.Boundary.Constraint.PinHole(count).SearchDist=3;

count=count+1;

end

% pre-process model

fem=femPreProcessing(fem);

% define current axis for plot purpose

ax=axes;

fem.Post.Options.ParentAxes=a;

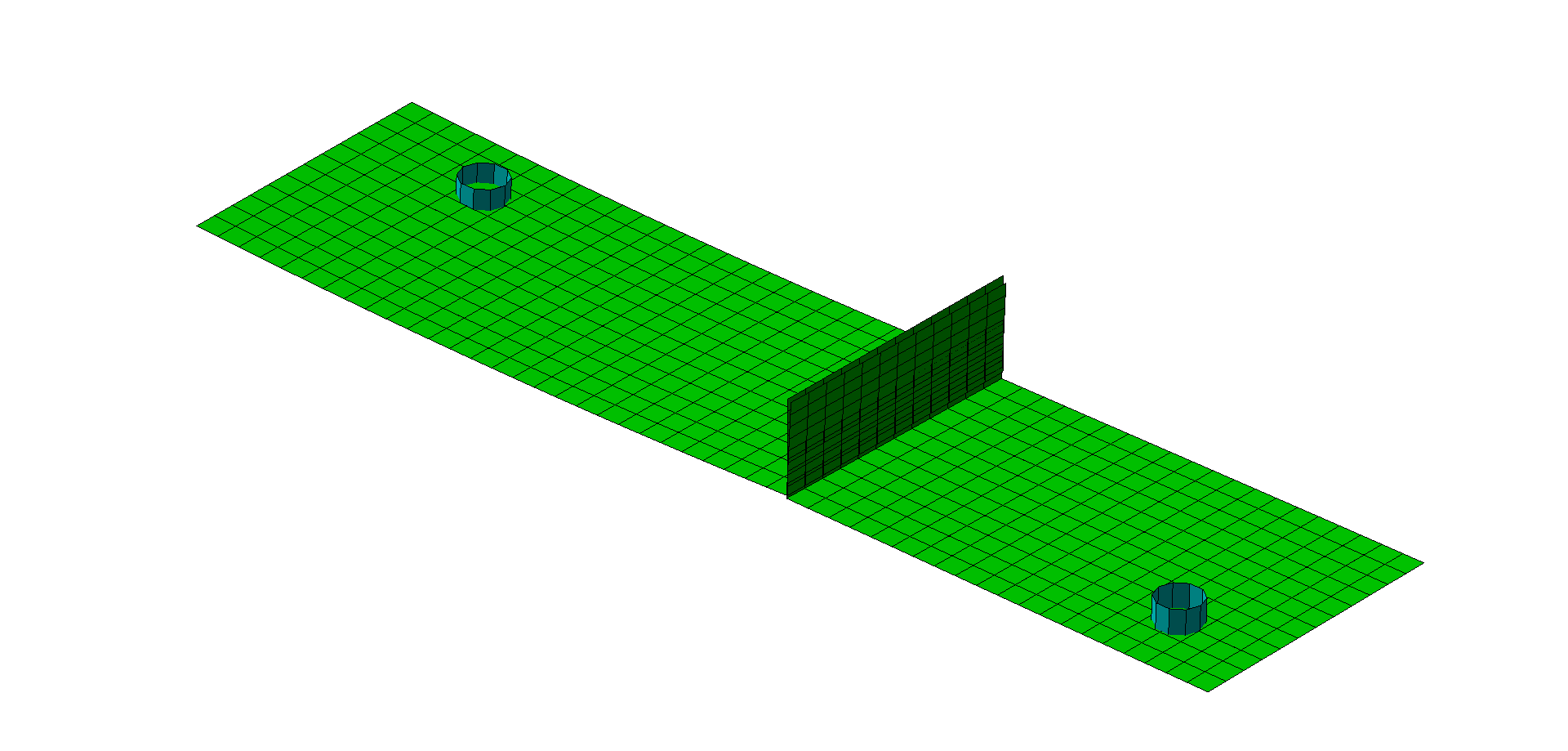
% plot mesh

meshPlot(fem)

% plot pin-hole

fem.Post.Options.SymbolSize=5;

pinholeBcPlot(fem)



**Plotting pin-hole constraints**

## pinslotBcPlot

*Aim:*

Plot pin-slot constraint.

*Syntax:*

pinslotBcPlot(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *graphical visualisation*

*Description:*

Plot pin-slot constraints. For each constraint a slot is shown. The following property can be set:

* **fem.Post.Options.SymbolSize**: define the size of the slot (double)

*Dependency:*

* **femPreProcessing**

*Example:*

% define constraint

Ps=[-30 0 -15;45 0 15];

count=1;

for i=1:size(Ps,1)

fem.Boundary.Constraint.PinSlot(count).Pm=Ps(i,:);

fem.Boundary.Constraint.PinSlot(count).Nm1=[0 0 1];

fem.Boundary.Constraint.PinSlot(count).Nm2=[1 0 0];

fem.Boundary.Constraint.PinSlot(count).Domain=i;

fem.Boundary.Constraint.PinSlot(count).SearchDist=3;

count=count+1;

end

% pre-process model

fem=femPreProcessing(fem);

% define current axis for plot purpose

ax=axes;

fem.Post.Options.ParentAxes=a;

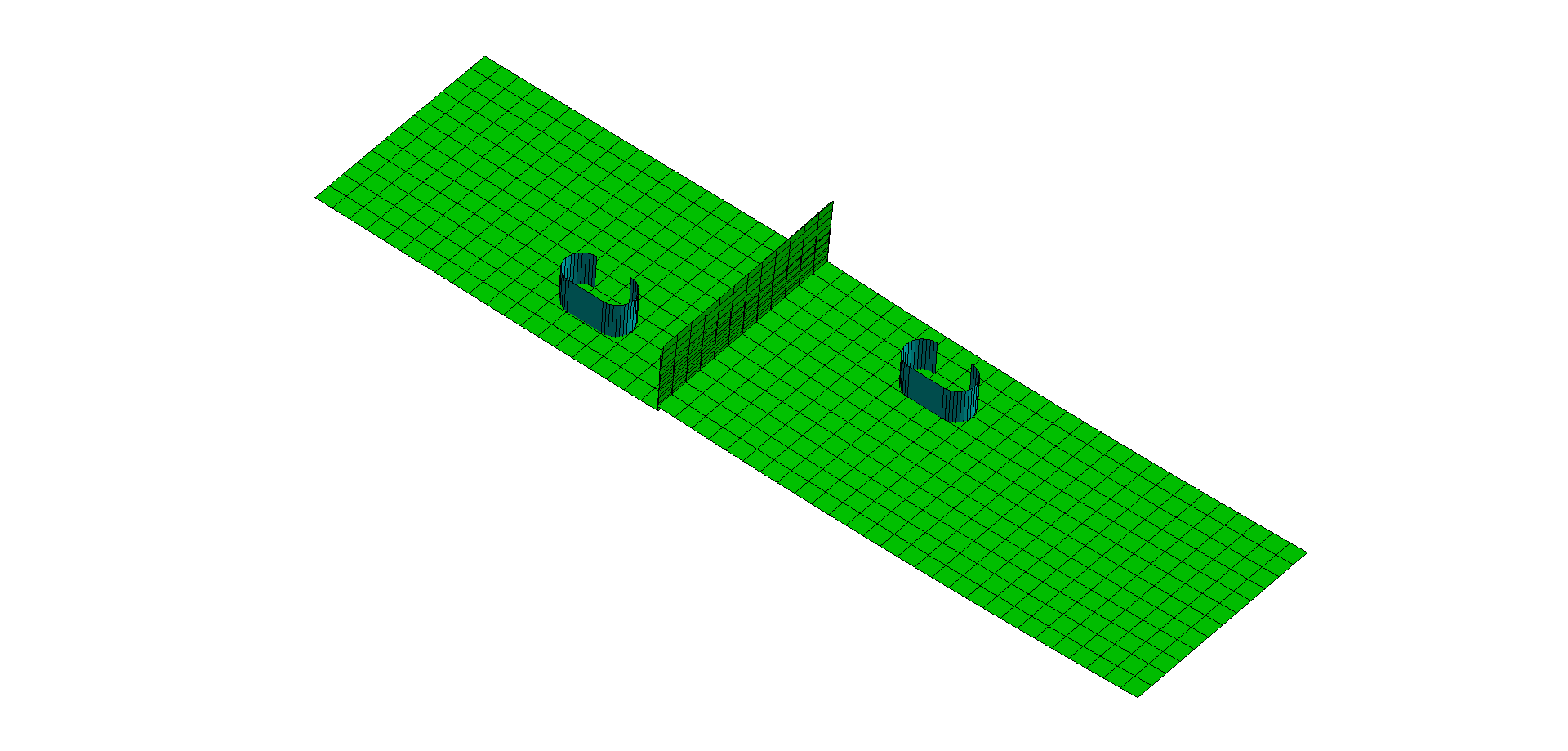
% plot mesh

meshPlot(fem)

% plot pin-slot

fem.Post.Options.SymbolSize=5;

pinslotBcPlot(fem)



**Plotting pin-slot constraints**

## rigilinkBcPlot

*Aim:*

Plot rigid-link constraint.

*Syntax:*

rigidlinkBcPlot(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *graphical visualisation*

*Description:*

Plot rigid-link constraints. For each constraint, a cylinder with one arrow pointing in the constraint direction is shown. The following properties can be set:

* **fem.Post.Options.SymbolSize**: define the size of the cylinder (double)
* **fem.Post.Options.LengthAxis**: define the length of the arrow (double)
* **fem.Post.Options.ShowProjection**: if "true" than projected points are visualised (true / false). This option can be set only after compiling all equations (**femRefresh**)

*Dependency:*

* **femPreProcessing**
* **femRefresh**

*Example:*

% define constraint

C=[0 -12 27;0 -12 -27];

for i=1:size(C,1)

fem.Boundary.Constraint.RigidLink(i).Pm=C(i,:);

fem.Boundary.Constraint.RigidLink(i).Nm=[-1 0 0];

fem.Boundary.Constraint.RigidLink(i).SearchDist=3;

fem.Boundary.Constraint.RigidLink(i).Master=2;

fem.Boundary.Constraint.RigidLink(i).Slave=1;

end

% pre-process model

fem=femPreProcessing(fem);

% compile equations

fem=femRefresh(fem);

% define current axis for plot purpose

ax=axes;

fem.Post.Options.ParentAxes=a;

% plot mesh

meshPlot(fem)

% define options

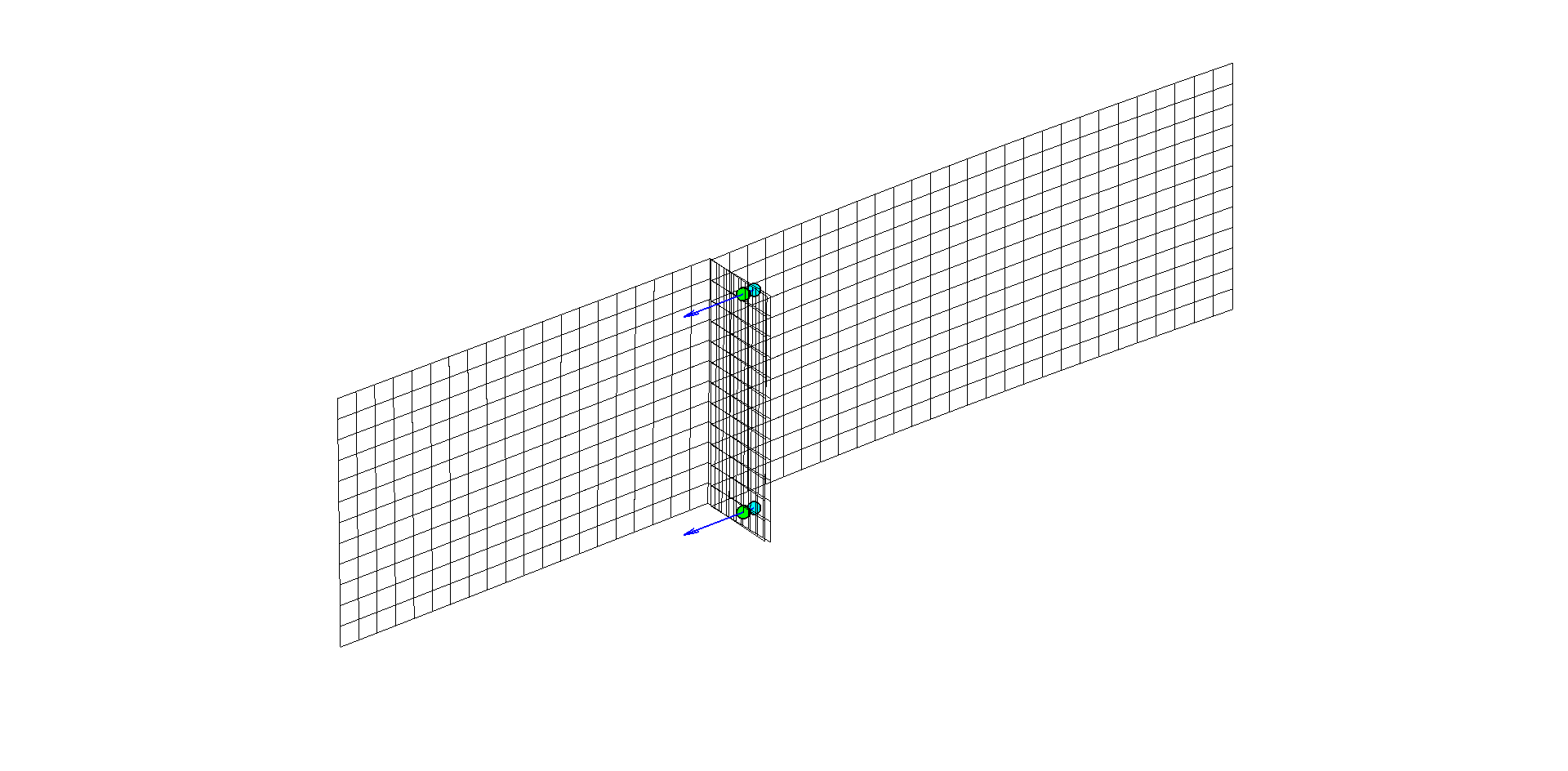
fem.Post.Options.SymbolSize=10;

fem.Post.Options.LengthAxis=0.5;

fem.Post.Options.ShowProjection=true;

% plot

rigidlinkBcPlot(fem)



**Plotting rigid-link constraints**

## selectPointComponent

*Aim:*

Allow user to select interactively a point.

*Syntax:*

selectPointComponent(fem, idpart)

*Input:*

* *fem*: fem structure
* *idpart*: list of components to be selected

*Output:*

* *graphical visualisation*

*Description:*

Allow user to select interactively a point.

*Dependency:*

* **mesh model has been loaded**

*Example:*

% import mesh model

fem=importMesh(fem,filemesh);

% run interactive tool

a=axes; hold all

fem.Post.Options.ParentAxes=a;

fem.Post.Options.ShowPatch=false;

meshPlot(fem)

% work on component "1"

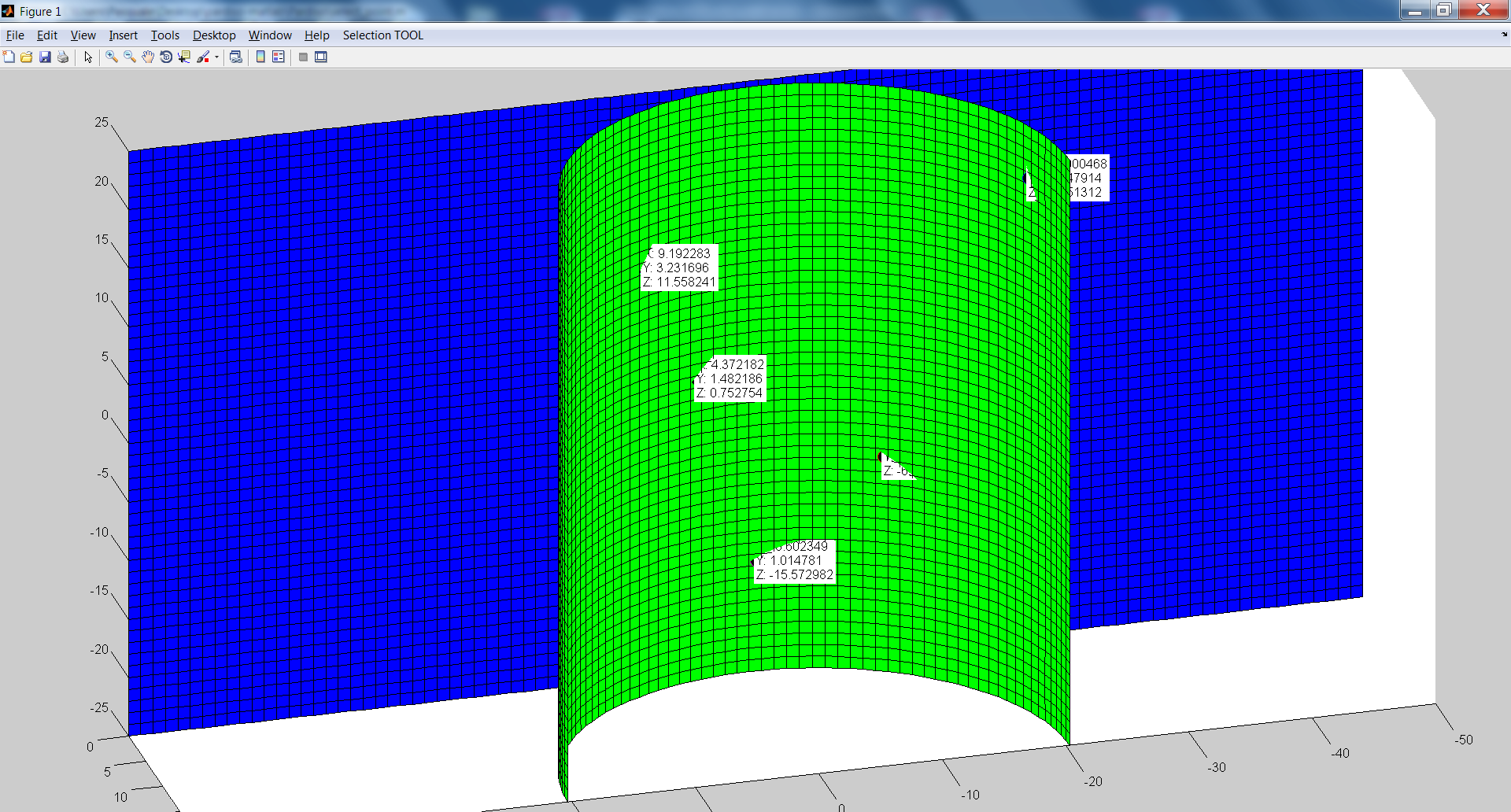
idpart=1;

fem.Post.Options.ShowPatch=true;

selectPointComponent(fem, idpart);

****

**Interactive tool for point selection**

****

**Interactive selection of a 3D point**

## selectionTool

*Aim:*

Allow user to select interactively a region of interest (ROI).

*Syntax:*

selectionTool(fem, idparts)

*Input:*

* *fem*: fem structure
* *idparts*: list of components to be selected

*Output:*

* *graphical visualisation*

*Description:*

Allow user to select interactively a region of interest. The "Select" tool allows to draw a rectangle in the graphical area.

*Dependency:*

* **mesh model has been loaded**

*Example:*

% import mesh model

fem=importMesh(fem,filemesh);

% run interactive tool

a=axes; hold all

fem.Post.Options.ParentAxes=a;

fem.Post.Options.ShowPatch=false;

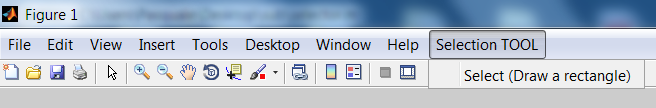
meshPlot(fem)

% work on components "1" and "2"

idpart=[1 2];

fem.Post.Options.ShowPatch=true;

selectionTool(fem, idpart);

****

**Interactive tool for RoI selection**

fprintf('Press any key to continue...\n')

pause

% get seleciont

handle=guidata(gcf);

fem=handle.fem;

idnode=handle.Selection;

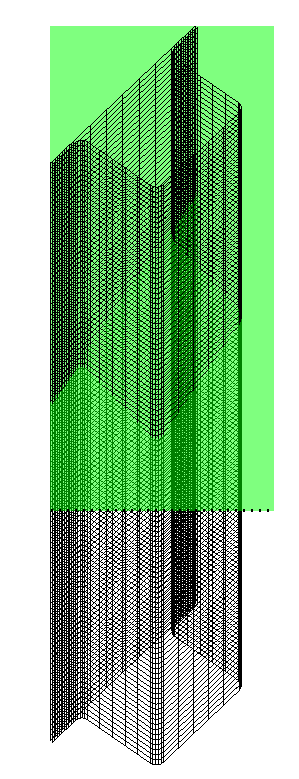
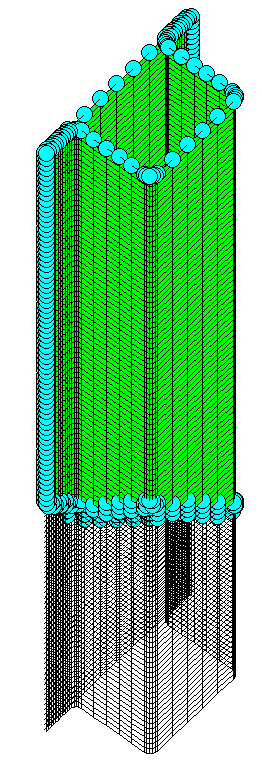
fem=femSetSelection(fem,idnode);

pre=findobj(gcf,'tag','slt');

delete(pre);

fem.Post.Options.ShowPatch=true;

selectionPlot(fem)

** **

**Interactive selection (left - rectangular window; right - selected elements)**

## unilateralBcPlot

*Aim:*

Plot unilateral constraints.

*Syntax:*

unilateralBcPlot(fem)

*Input:*

* *fem*: fem structure

*Output:*

* *graphical visualisation*

*Description:*

Plot unilateral constraints. For each constraint, a cylinder with one arrow pointing in the constraint direction is shown. The following properties can be set:

* **fem.Post.Options.SymbolSize**: define the size of the cylinder (double)
* **fem.Post.Options.LengthAxis**: define the length of the arrow (double)
* **fem.Post.Options.ShowProjection**: if "true" than projected points are visualised (true / false). This option can be set only after compiling all equations (**femRefresh**)

*Dependency:*

* **femPreProcessing**
* **femRefresh**

*Example:*

% define constraint

fem.Boundary.Constraint.Unilateral(1).Pm=[3 0.1 -0.3]; % point

fem.Boundary.Constraint.Unilateral(1).SearchDist=3; % search distance

fem.Boundary.Constraint.Unilateral(1).Nm=[0 0 1]; % normal direction

fem.Boundary.Constraint.Unilateral(1).Size=false; % true/false

fem.Boundary.Constraint.Unilateral(1).Offset=0; % offset

fem.Boundary.Constraint.Unilateral(1).Domain=1;

fem.Boundary.Constraint.Unilateral(1).Constraint='lock';

fem.Boundary.Constraint.Unilateral(2).Pm=[4 0.1 0.3]; % point

fem.Boundary.Constraint.Unilateral(2).SearchDist=3; % search distance

fem.Boundary.Constraint.Unilateral(2).Nm=[0 0 -1]; % normal direction

fem.Boundary.Constraint.Unilateral(2).Size=false; % true/false

fem.Boundary.Constraint.Unilateral(2).Offset=0; % offset

fem.Boundary.Constraint.Unilateral(2).Domain=1;

fem.Boundary.Constraint.Unilateral(2).Constraint='lock';

% pre-process model

fem=femPreProcessing(fem);

% define current axis for plot purpose

ax=axes;

fem.Post.Options.ParentAxes=a;

% plot mesh

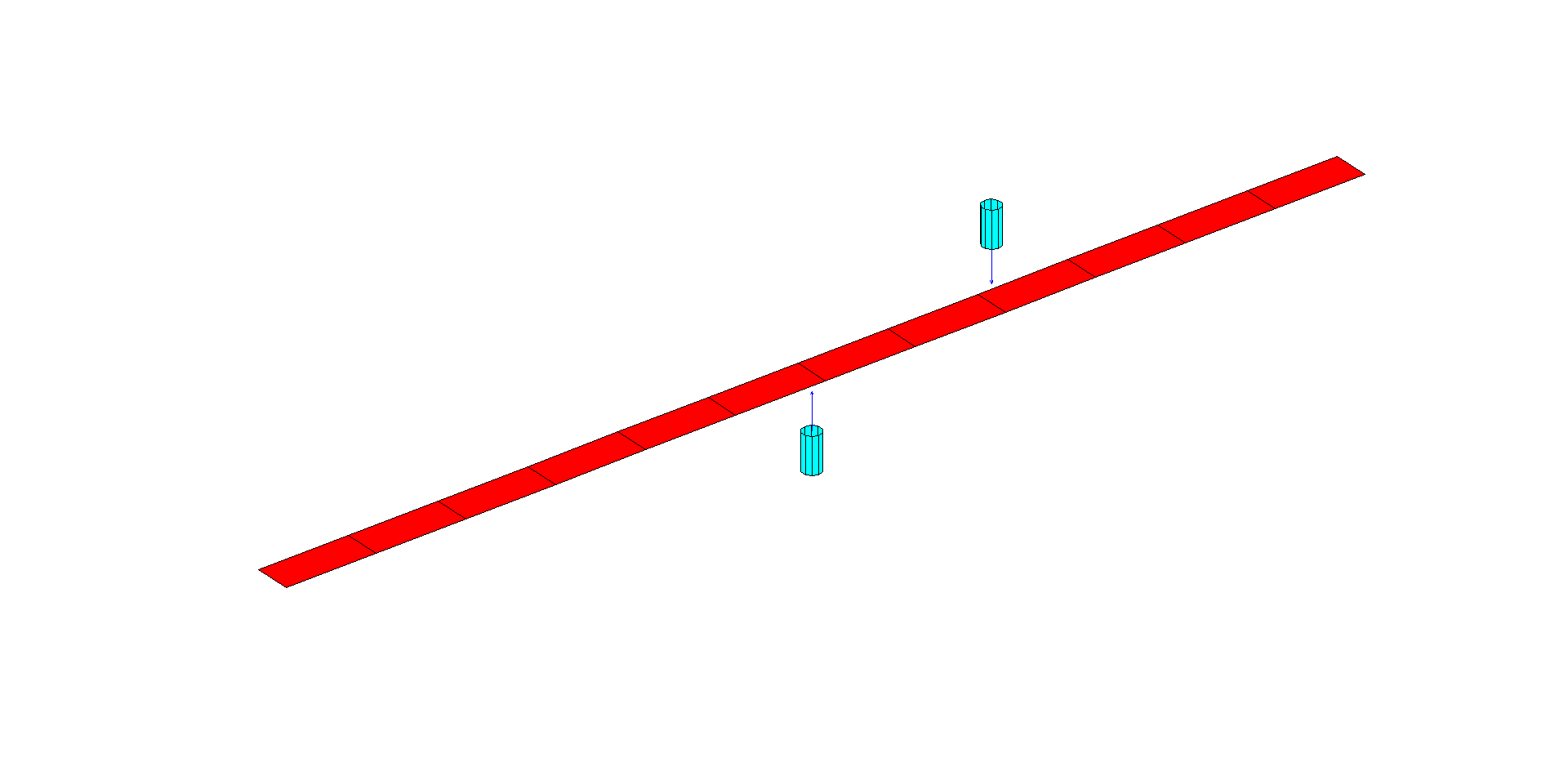
meshPlot(fem)

% plot constraints

fem.Post.Options.SymbolSize=0.05;

fem.Post.Options.LengthAxis=0.05;

unilateralBcPlot(fem)



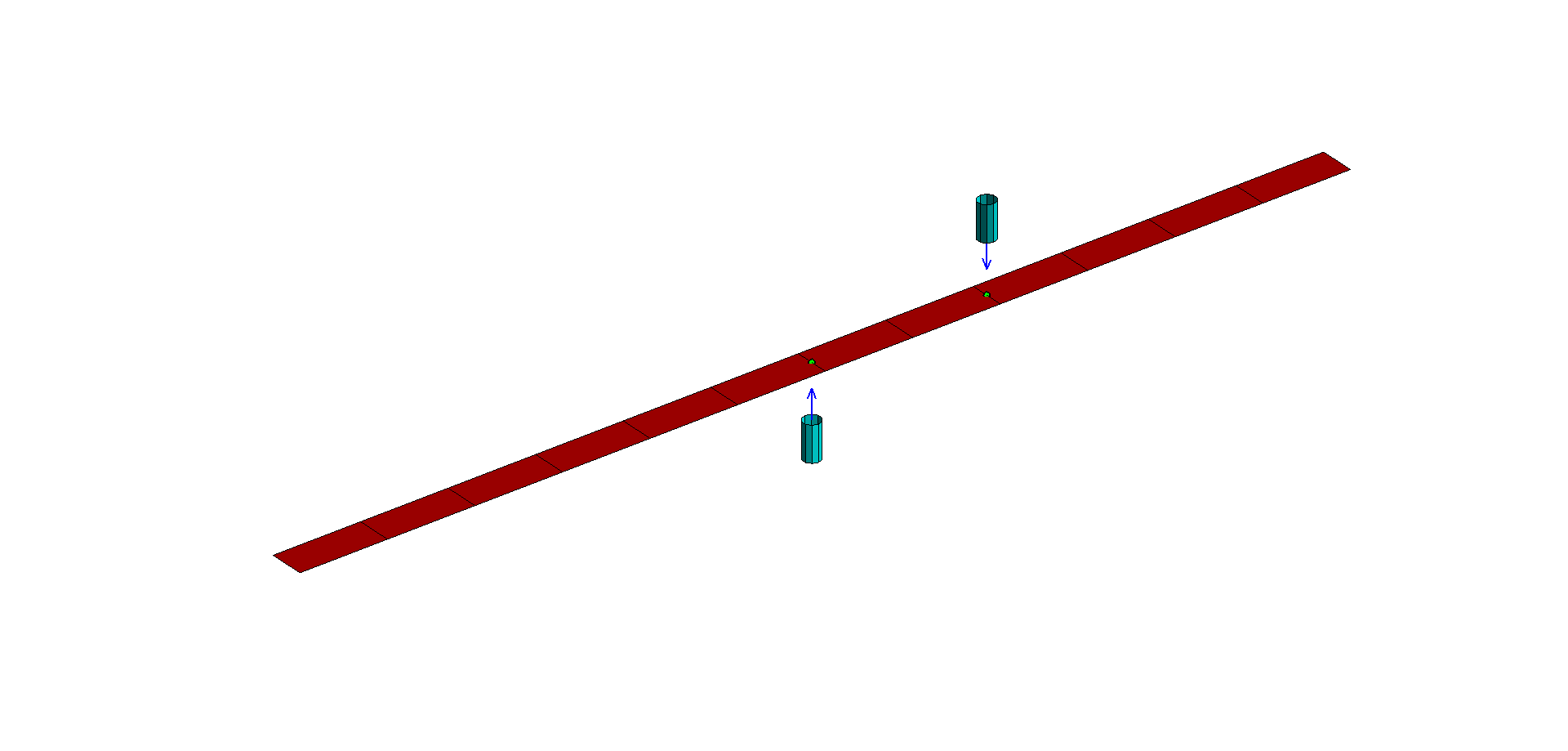
**Plotting unilateral constraint**

% now plot projection points as well

fem=femRefresh(fem); % compile equations

fem.Post.Options.ShowProjection=true;

unilateralBcPlot(fem)



**Plotting unilateral constraint - show projected points**