A simple Finite Element implementation for solving plane strain elasticity problem

xieyn

cexieyn@fzu.edu.cn

April, 2023

This notebook shows a simple finite element implementation to solve a plane strain biaxial compression problem with linear elastic material.

Preliminaries

First clear all the expressions in Global context and set the current working directory for potential file operations. It is not necessary, but it clears much of daily confusions in my daily use of Mathematica.

```
In[•]:= ClearAll["Global`*"];

In[•]:= SetDirectory[NotebookDirectory[]];
```

Problem statement

As shown in the **Fig. 1**, a linear elastic body is subjected to σ_x = 200 kPa on the right boundary and σ_y = 100 kPa on the top boundary. The left boundary is fixed in x direction, and the bottom in y direction.

The Young's modulus and Poisson's ratio for the medium are E = 200 MPa and v = 0.3, respectively.

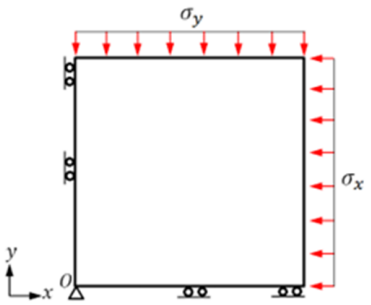


Fig. 1 Schematic of a loaded elastic body

In this problem, we seek to solve for the displacement and stress (strain) of the elastic body by Finite Element method. Note that the above region is just a quarter of a plane strain elastic body of

size 4m by 4m subjected to horizontal stress of 200 kPa and vertical stress of 100 kPa, respectively, so the analytical solutions can be easily obtained.

A first step simple discretization for the FEM is shown in **Fig. 2**, with nodes and denoted by red numbers and elements by blue circled numbers. Surely, the coarseness of discretization should be generalized in the code.

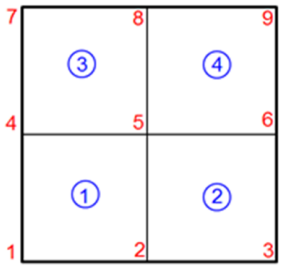


Fig. 2 Simple discretization for FEM

Further, we may want to find the stress and strain distributions if the lateral load is changed to triangle load.

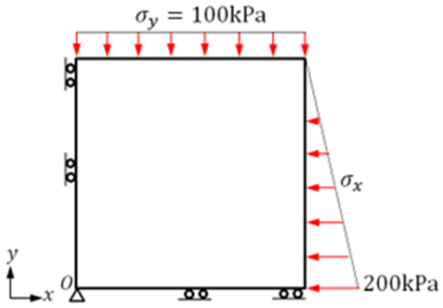


Fig. 3 Elastic body subjected to triangle load

Finite Element procedure

Overview

Here we directly present procedures in conducting a FE analysis. The mathematical background can be easily found in related textbooks.

A basic FEM may consist of:

1. Discretization of the domain of interest. Specifically, if the coordinates of nodes and connectivity list are determined, the discretization mesh for the problem is then determined.
2. Assigning material properties to the elements and formulating the generalized Hooke's law for plane strain condition. Here we only consider uniform and constant linearly elastic body, so the stiffness matrix for the whole simulation is fixed. Also a total form of stress-strain relationship is adopted. If nonlinear and inhomogeneous material is to be considered, a rate form with varied material properties should be implemented.
3. Formulating the stiffness matrix at element level. Get B matrix and Jacobian matrix. Also integrating to form stiffness matrix by Gaussian quadrature at element level.
4. Assembling global stiffness matrix according to the connectivity list.

5. Imposing essential and natural boundary conditions. Natural boundary stresses are 'naturally' integrated to boundary nodes. Essential boundary is realized by various techniques.
6. Solving the linear system for displacements. Post-processing the displacements to get stresses and strains.

Nodes

The nodes are presented as associations of NodeID → Coordinates, for example, 1 → {0.0,0.0}.

In this simple example, the nodes can be generated as the following(We don't need to sort the output nodes, but sort it with KeySort for better inspection, anyway.).

Function to generate nodes and corresponding coordinates.

```
In[•]:= ClearAll[genNodes]
genNodes[width_?NumericQ, height_?NumericQ, helem_Integer, velem_Integer] := Module[
  {oneNode},
  oneNode[i_Integer, j_Integer] := Module[
    {nodeid, x, y},
    nodeid = i + (j - 1) * (helem + 1);
    x = (i - 1) * (width / helem);
    y = (j - 1) * (height / velem);
    nodeid → {x, y}
   ];
  KeySort@Association[Array[oneNode, {helem + 1, velem + 1}]]
 ]
```

In the simple discretization as shown in Fig. 2, the data for the nodes is:

```
In[•]:= nodes = genNodes[1.0, 1.0, 2, 2]
Out[•]=
<|1 → {0., 0.}, 2 → {0.5, 0.}, 3 → {1., 0.}, 4 → {0., 0.5},
 5 → {0.5, 0.5}, 6 → {1., 0.5}, 7 → {0., 1.}, 8 → {0.5, 1.}, 9 → {1., 1.}|>
```

Elements

Building connectivity list, again, stored as associations.

Function to generate elements and associated nodes, in counter-clockwise direction.

```
In[•]:= ClearAll[genElems]
genElems[helem_Integer, velem_Integer] := Module[
  {getNode, oneElem},
  getNode[i_Integer, j_Integer] := i + (j - 1) * (helem + 1);
  oneElem[i_Integer, j_Integer] := Module[{elemid},
    elemid = i + (j - 1) * helem;
    elemid → {getNode[i, j], getNode[i + 1, j], getNode[i + 1, j + 1], getNode[i, j + 1]}
   ];
  KeySort@Association[Array[oneElem, {helem, velem}]]
 ]
```

In the Fig. 2, the elements are:

```
In[•]:= elems = genElems[2, 2]
```

```
Out[•]=
<|1 → {1, 2, 5, 4}, 2 → {2, 3, 6, 5}, 3 → {4, 5, 8, 7}, 4 → {5, 6, 9, 8}|>
```

Material

Considering plane strain case and uniform linear elasticity for now.

The stress-strain relationship is:

$$\begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \tau_{xy} \end{pmatrix} = \frac{E}{(1+\nu)\ (1-2\,\nu)} \begin{pmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \frac{1-2\,\nu}{2} \end{pmatrix} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \gamma_{xy} \end{pmatrix}$$

or in compact form:

$\sigma = \mathbf{D} : \varepsilon$

Linear elastic stiffness matrix in plane strain condition.

```
In[•]:= ClearAll[dMatrix]
dMatrix[youngs_?NumericQ, nu_?NumericQ] := Module[
  {pre},
  pre = youngs / (1.0 + nu) / (1.0 - 2.0 * nu);
  pre * {{1.0 - nu, nu, 0.0}, {nu, 1.0 - nu, 0.0}, {0.0, 0.0, (1.0 - 2.0 * nu) / 2.0}}
 ]
```

```
In[•]:= youngs = 200.0 × 10^6;
nu = 0.3;
dm = dMatrix[youngs, nu]
```

```
Out[•]=
{{2.69231 × 10^8, 1.15385 × 10^8, 0.},
 {1.15385 × 10^8, 2.69231 × 10^8, 0.}, {0., 0., 7.69231 × 10^7}}
```

Element stiffness matrix

Shape functions and its derivatives with respect to ξ and η

The shape function for 4-node quadrilateral element is

$$N_i\ (\xi, \eta) = \frac{1}{4}\ (1 + \xi_i\ \xi)\ (1 + \eta_i\ \eta),\ i = 1, 2, 3, 4$$

where

i	ξ_i	η_i
1	−1	−1
2	1	−1
3	1	1
4	−1	1

Shape function and its derivatives.

In[•]:=
```
ClearAll[shapeFun, dNdξ, dNdη];
shapeFun[ξ_?NumericQ, η_?NumericQ] :=
  Module[
   {xi = {-1, 1, 1, -1}, eta = {-1, -1, 1, 1}},
   1.0 / 4.0 (1 + xi * ξ) (1 + eta * η)
  ];
dNdξ[ξ_?NumericQ, η_?NumericQ] :=
  {-1.0/4.0 (1 - η), 1.0/4.0 (1 - η), 1.0/4.0 (1 + η), -1.0/4.0 (1 + η)};
dNdη[ξ_?NumericQ, η_?NumericQ] :=
  {-1.0/4.0 (1 - ξ), -1.0/4.0 (1 + ξ), 1.0/4.0 (1 + ξ), 1.0/4.0 (1 - ξ)};
```

B matrix and Jacobian matrix

B matrix relates the strain in the element to the nodal displacements, specifically, $\boldsymbol{\varepsilon} = \mathbf{B}\,\mathbf{u}$, where $\boldsymbol{\varepsilon}$ are the strains at any point in the element, $\mathbf{u}$ are the nodal displacements, and $\mathbf{B}$ is given in terms of the derivative of shape functions with respect to global coordinates {x,y} as:

$$\mathbf{B}(x, y) = \begin{pmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} & 0 & \frac{\partial N_4}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & 0 & \frac{\partial N_3}{\partial y} & 0 & \frac{\partial N_4}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} & \frac{\partial N_4}{\partial y} & \frac{\partial N_4}{\partial x} \end{pmatrix}$$

For convenience of Gaussian integration in a later step, we need to transform the above entries to {ξ,η} coordinates system, which is related to {x, y} coordinates system by so-called Jacobian matrix (**J**),

$$\begin{pmatrix} \frac{\partial N}{\partial \xi} \\ \frac{\partial N}{\partial \eta} \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{pmatrix}}_{\mathbf{J}} \begin{pmatrix} \frac{\partial N}{\partial x} \\ \frac{\partial N}{\partial y} \end{pmatrix}$$

$$\mathbf{J}(\xi, \eta) = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{4} \frac{\partial N_i}{\partial \xi} x_i & \sum_{i=1}^{4} \frac{\partial N_i}{\partial \xi} y_i \\ \sum_{i=1}^{4} \frac{\partial N_i}{\partial \eta} x_i & \sum_{i=1}^{4} \frac{\partial N_i}{\partial \eta} y_i \end{pmatrix} = \begin{pmatrix} \frac{\partial N_1}{\partial \xi} & \frac{\partial N_2}{\partial \xi} & \frac{\partial N_3}{\partial \xi} & \frac{\partial N_4}{\partial \xi} \\ \frac{\partial N_1}{\partial \eta} & \frac{\partial N_2}{\partial \eta} & \frac{\partial N_3}{\partial \eta} & \frac{\partial N_4}{\partial \eta} \end{pmatrix} \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{pmatrix}$$

Therefore, the B matrix can be collectively obtained by

$$\mathbf{B}(\xi, \eta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{J}^{-1} & 0 \\ 0 & \mathbf{J}^{-1} \end{pmatrix} \begin{pmatrix} \frac{\partial N_1}{\partial \xi} & 0 & \frac{\partial N_2}{\partial \xi} & 0 & \frac{\partial N_3}{\partial \xi} & 0 & \frac{\partial N_4}{\partial \xi} & 0 \\ \frac{\partial N_1}{\partial \eta} & 0 & \frac{\partial N_2}{\partial \eta} & 0 & \frac{\partial N_3}{\partial \eta} & 0 & \frac{\partial N_4}{\partial \eta} & 0 \\ 0 & \frac{\partial N_1}{\partial \xi} & 0 & \frac{\partial N_2}{\partial \xi} & 0 & \frac{\partial N_3}{\partial \xi} & 0 & \frac{\partial N_4}{\partial \xi} \\ 0 & \frac{\partial N_1}{\partial \eta} & 0 & \frac{\partial N_2}{\partial \eta} & 0 & \frac{\partial N_3}{\partial \eta} & 0 & \frac{\partial N_4}{\partial \eta} \end{pmatrix}$$

Function to get B matrix and determinant of Jacobian matrix of an element at local coordinates (ξ,η).

```
In[•]:= ClearAll[getBandJ];
getBandJ[ξ_?NumericQ, η_?NumericQ, nodesxy_List] := Module[
  {jacob, detJ, invJ, indmat, invJmat, dNmat, bmat},
  jacob = {dNdξ[ξ, η], dNdη[ξ, η]}.nodesxy;
  detJ = Det[jacob];
  invJ = Inverse[jacob];

  indmat = {{1.0, 0, 0, 0}, {0, 0, 0, 1.0}, {0, 1.0, 1.0, 0}};
  invJmat = SparseArray[Band[{1, 1}] -> {invJ, invJ}];
  dNmat = {Riffle[dNdξ[ξ, η], {0, 0, 0, 0}], Riffle[dNdη[ξ, η], {0, 0, 0, 0}],
    Riffle[{0, 0, 0, 0}, dNdξ[ξ, η]], Riffle[{0, 0, 0, 0}, dNdη[ξ, η]]};

  bmat = indmat.invJmat.dNmat;
  {bmat, detJ}
 ]
```

Element stiffness matrix

The element stiffness matrix is $\mathbf{k}^e = \int_{\Omega^e} \mathbf{B}^\intercal \mathbf{D}\, \mathbf{B}\, d\Omega^e = \int_{\Omega^e} \mathbf{f}(x, y)\, d\Omega^e = \int_{\Omega^e} f(\xi, \eta) \left| \mathbf{J}(\xi, \eta) \right| d\xi\, d\eta$ in integral form. It is approximated by Gaussian quadrature as $\mathbf{k}^e \approx \sum_i^{\text{jpt}} \sum_j^{\text{jpt}} W_i W_j \mathbf{f}(\xi_i, \eta_j) \left| \mathbf{J}(\xi_i, \eta_j) \right|$. Here we use two quadrature points in both directions.

Function to construct stiffness matrix at element level.

```
In[•]:= ClearAll[stiffElem]
stiffElem[nodesxy_List, dm_List] := Module[
  {weightposx, weightposy, gauss},
  weightposx = N[{{1, -√3 / 3}, {1, √3 / 3}}];
  weightposy = N[{{1, -√3 / 3}, {1, √3 / 3}}];
  gauss[{w1_, x1_}, {w2_, x2_}] := Module[{bm, dj},
    {bm, dj} = getBandJ[x1, x2, nodesxy];
    w1 * w2 * Transpose[bm].dm.bm * dj
   ];
  (* Outer may be slow here*)
  Plus @@ Flatten[Outer[gauss[#1, #2] &, weightposx, weightposy, 1], 1]
 ]
```

Global stiffness matrix

Assemble the global stiffness matrix according to the connectivity list.

NOTE: It is worth trying to represent the entries in the stiffness matrix by Associations, in which case, we only need to deal with the entry indices and associated values. Global stiffness matrix assembling and essential boundary imposing should also be much easier, since it is easier to manipulate associations.

Function to assemble stiffness matrix for one element and all element.

```
In[•]:= ClearAll[assemble1, assemble2, assembleall];
(* assemble with Table *)
```

```
assemble1[nodes_Association, connectivity_List, dm_List] :=
 Module[{nodesid, nodesxy, ke, ig, jg, n},
  nodesid = connectivity;
  nodesxy = Lookup[nodes, nodesid];
  ke = stiffElem[nodesxy, dm];
  n = Length[nodes];

  Table[
   ig = nodesid[[ie]];
   jg = nodesid[[je]];
   SparseArray[
    {
     {2 * ig - 1, 2 * jg - 1} -> ke[[2 * ie - 1, 2 * je - 1]],
     {2 * ig, 2 * jg - 1} -> ke[[2 * ie, 2 * je - 1]],
     {2 * ig - 1, 2 * jg} -> ke[[2 * ie - 1, 2 * je]],
     {2 * ig, 2 * jg} -> ke[[2 * ie, 2 * je]]
    }
    , {2 * n, 2 * n}]
   ,
   {ie, 1, 4},
   {je, 1, 4}]
 ]

(* assemble with Array, seems to be slower *)
assemble2[nodes_Association, connectivity_List, dm_List] :=
 Module[{nodesid, nodesxy, ke, n, assem},
  nodesid = connectivity;
  nodesxy = Lookup[nodes, nodesid];
  ke = stiffElem[nodesxy, dm];
  n = Length[nodes];

  assem[ie_, je_] := Module[
    {ig, jg},
    ig = nodesid[[ie]];
    jg = nodesid[[je]];
    SparseArray[
     {
      {2 * ig - 1, 2 * jg - 1} -> ke[[2 * ie - 1, 2 * je - 1]],
      {2 * ig, 2 * jg - 1} -> ke[[2 * ie, 2 * je - 1]],
      {2 * ig - 1, 2 * jg} -> ke[[2 * ie - 1, 2 * je]],
      {2 * ig, 2 * jg} -> ke[[2 * ie, 2 * je]]
     }
     , {2 * n, 2 * n}]
   ];
  Array[assem, {4, 4}]
 ]

(* assemble across all elements *)
assembleall[nodes_Association, elems_Association, dm_List] :=
 Plus @@ Flatten[Map[assemble1[nodes, #, dm] &, Values[elems]], 2]
```

We may check the stiffness matrix after assembling the first element:

```
In[•]:= Plus @@ Flatten[assemble1[nodes, First@elems, dm], 1]
```

Out[•]=

SparseArray [ Specified elements: 64
Dimensions: {18, 18}]

```
In[•]:= Plus @@ Flatten[assemble1[nodes, First@elems, dm], 1] // MatrixForm
```

Out[•]//MatrixForm=

$$\begin{pmatrix}
1.15385\times10^{8} & 4.80769\times10^{7} & -7.69231\times10^{7} & 9.61538\times10^{6} & 0. & 0. & 1.92308\times10^{7} & -9.61538 \\
4.80769\times10^{7} & 1.15385\times10^{8} & -9.61538\times10^{6} & 1.92308\times10^{7} & 0. & 0. & 9.61538\times10^{6} & -7.69231 \\
-7.69231\times10^{7} & -9.61538\times10^{6} & 1.15385\times10^{8} & -4.80769\times10^{7} & 0. & 0. & -5.76923\times10^{7} & 4.80769 \\
9.61538\times10^{6} & 1.92308\times10^{7} & -4.80769\times10^{7} & 1.15385\times10^{8} & 0. & 0. & 4.80769\times10^{7} & -5.76923 \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\
1.92308\times10^{7} & 9.61538\times10^{6} & -5.76923\times10^{7} & 4.80769\times10^{7} & 0. & 0. & 1.15385\times10^{8} & -4.80769 \\
-9.61538\times10^{6} & -7.69231\times10^{7} & 4.80769\times10^{7} & -5.76923\times10^{7} & 0. & 0. & -4.80769\times10^{7} & 1.15385 \\
-5.76923\times10^{7} & -4.80769\times10^{7} & 1.92308\times10^{7} & -9.61538\times10^{6} & 0. & 0. & -7.69231\times10^{7} & 9.61538 \\
-4.80769\times10^{7} & -5.76923\times10^{7} & 9.61538\times10^{6} & -7.69231\times10^{7} & 0. & 0. & -9.61538\times10^{6} & 1.92308 \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0.
\end{pmatrix}$$

Note that positions associated with {1,2,5,4} are filled with numbers.

Global stiffness matrix:

```
In[•]:= assembleall[nodes, elems, dm]
```

Out[•]=

SparseArray [ Specified elements: 196
Dimensions: {18, 18}]

```
In[•]:= assembleall[nodes, elems, dm] // MatrixForm
```

Out[•]//MatrixForm=

$$\begin{pmatrix}
1.15385\times10^{8} & 4.80769\times10^{7} & -7.69231\times10^{7} & 9.61538\times10^{6} & 0. & 0. & 1\\
4.80769\times10^{7} & 1.15385\times10^{8} & -9.61538\times10^{6} & 1.92308\times10^{7} & 0. & 0. & 9\\
-7.69231\times10^{7} & -9.61538\times10^{6} & 2.30769\times10^{8} & 7.45058\times10^{-9} & -7.69231\times10^{7} & 9.61538\times10^{6} & -\\
9.61538\times10^{6} & 1.92308\times10^{7} & 0. & 2.30769\times10^{8} & -9.61538\times10^{6} & 1.92308\times10^{7} & 4\\
0. & 0. & -7.69231\times10^{7} & -9.61538\times10^{6} & 1.15385\times10^{8} & -4.80769\times10^{7} & \\
0. & 0. & 9.61538\times10^{6} & 1.92308\times10^{7} & -4.80769\times10^{7} & 1.15385\times10^{8} & \\
1.92308\times10^{7} & 9.61538\times10^{6} & -5.76923\times10^{7} & 4.80769\times10^{7} & 0. & 0. & 1\\
-9.61538\times10^{6} & -7.69231\times10^{7} & 4.80769\times10^{7} & -5.76923\times10^{7} & 0. & 0. & 7\\
-5.76923\times10^{7} & -4.80769\times10^{7} & 3.84615\times10^{7} & 0. & -5.76923\times10^{7} & 4.80769\times10^{7} & -\\
-4.80769\times10^{7} & -5.76923\times10^{7} & 0. & -1.53846\times10^{8} & 4.80769\times10^{7} & -5.76923\times10^{7} & 7\\
0. & 0. & -5.76923\times10^{7} & -4.80769\times10^{7} & 1.92308\times10^{7} & -9.61538\times10^{6} & \\
0. & 0. & -4.80769\times10^{7} & -5.76923\times10^{7} & 9.61538\times10^{6} & -7.69231\times10^{7} & \\
0. & 0. & 0. & 0. & 0. & 0. & 1\\
0. & 0. & 0. & 0. & 0. & 0. & -\\
0. & 0. & 0. & 0. & 0. & 0. & -\\
0. & 0. & 0. & 0. & 0. & 0. & -\\
0. & 0. & 0. & 0. & 0. & 0. & \\
0. & 0. & 0. & 0. & 0. & 0. &
\end{pmatrix}$$

Boundary conditions

Natural boundary

The force boundary is specified as {node1, node2, stress direction(1 for x, 2 for y), stress1, stress2}. Stress is linearly interpolated between the two nodes.

Function to set natural boundary.

```
In[•]:= ClearAll[setNaturalBoundary];
setNaturalBoundary[bcnodes_List, stsdir_Integer,
  sts1_Real, sts2_Real, nodes_Association] := Module[
  {stsf},
  stsf = Interpolation[{{Lookup[nodes, First[bcnodes]], sts1},
     {Lookup[nodes, Last[bcnodes]], sts2}}, InterpolationOrder → {1, 1}];
  ({#1[[1]], #1[[2]], stsdir, stsf @@ nodes[#1[[1]]], stsf @@ nodes[#1[[2]]]} &) /@
   Partition[bcnodes, 2, 1]
 ]
```

The natural boundary on the top is:

```
In[•]:= ftop = setNaturalBoundary[Range[7, 9], 2, -100.0 × 10^3, -100.0 × 10^3, nodes]
```

••• Interpolation: Requested order is too high; order has been reduced to {1, 0}.

Out[•]=

```
{{7, 8, 2, -100 000., -100 000.}, {8, 9, 2, -100 000., -100 000.}}
```

The natural boundary on the right in Fig.1 and Fig.3 are:

```
In[•]:= fright1 = setNaturalBoundary[3 * Range[3], 1, -200.0 × 10^3, -200.0 × 10^3, nodes]
```

Interpolation: Requested order is too high; order has been reduced to {0, 1}.

```
Out[•]= {{3, 6, 1, -200 000., -200 000.}, {6, 9, 1, -200 000., -200 000.}}
```

```
In[•]:= fright2 = setNaturalBoundary[3 * Range[3], 1, -200.0 × 10^3, 0.0 × 10^3, nodes]
```

Interpolation: Requested order is too high; order has been reduced to {0, 1}.

```
Out[•]= {{3, 6, 1, -200 000., -100 000.}, {6, 9, 1, -100 000., 0.}}
```

Take the natural boundary and collect the forces on the nodes segment by segment.

Function to assemble the natural boundary for one segment and all segments.

```
In[•]:= ClearAll[getNaturalBoundary1, getNaturalBoundary]
getNaturalBoundary1[seg_List, nodes_Association] := Module[
  {node1, node2, dir, t1, t2, a, b, len, weightpos, txi, shape1, shape2, fnode1, fnode2},
  {node1, node2, dir, t1, t2} = seg;
  a = (t1 + t2) / 2.0;
  b = (t2 - t1) / 2.0;
  len = EuclideanDistance[nodes[node1], nodes[node2]];

  weightpos = N[{{1, -√3 / 3}, {1, √3 / 3}}];
  txi[ξ_] := a + b ξ;
  shape1[ξ_] := 0.5 * (1 - ξ);
  shape2[ξ_] := 0.5 * (1 + ξ);

  fnode1 = Plus @@ (shape1[#2] * txi[#2] * #1 * len / 2.0 & @@@ weightpos);
  fnode2 = Plus @@ (shape2[#2] * txi[#2] * #1 * len / 2.0 & @@@ weightpos);

  SparseArray[{2 * (node1 - 1) + dir -> fnode1,
    2 * (node2 - 1) + dir -> fnode2},
   2 * Length@nodes]
 ]
getNaturalBoundary[segs_List, nodes_Association] :=
 Plus @@ (getNaturalBoundary1[#, nodes] & /@ segs)
```

In cases in Fig.1 and Fig.3, the force vector are (stored as SparseArray, but expanded here):

```
In[•]:= getNaturalBoundary[Join[ftop, fright1], nodes] // MatrixForm
```

```
Out[•]//MatrixForm=
```

$$\begin{pmatrix} 0. \\ 0. \\ 0. \\ 0. \\ -50\,000. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ -100\,000. \\ 0. \\ 0. \\ -25\,000. \\ 0. \\ -50\,000. \\ -50\,000. \\ -25\,000. \end{pmatrix}$$

```
In[•]:= getNaturalBoundary[Join[ftop, fright2], nodes] // MatrixForm
```

```
Out[•]//MatrixForm=
```

$$\begin{pmatrix} 0. \\ 0. \\ 0. \\ 0. \\ -41\,666.7 \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ -50\,000. \\ 0. \\ 0. \\ -25\,000. \\ 0. \\ -50\,000. \\ -8333.33 \\ -25\,000. \end{pmatrix}$$

Essential boundary

Essential boundary are specified as {nodeid, dir, disp}.

In this particular case, the nodes on the bottom and left boundaries are zero.

Function to generate essential boundary for the simple case.

```
In[•]:= ClearAll[essentialBoundary]
essentialBoundary[helem_Integer, velem_Integer] := Module[{bottom, left},
  bottom = Table[{inode, 2, 0.0}, {inode, 1, helem + 1}];
  left = Table[{(inode - 1) * (helem + 1) + 1, 1, 0.0}, {inode, 1, velem + 1}];
  Join[bottom, left]
 ]

In[•]:= essentialBoundary[2, 2]
Out[•]=
{{1, 2, 0.}, {2, 2, 0.}, {3, 2, 0.}, {1, 1, 0.}, {4, 1, 0.}, {7, 1, 0.}}
```

A utility function replace part of a SparseArray, taken from https://mathematica.stackexchange.com/a/165010/46781.
Although straightforward, replace elements in a SparseArray with ReplacePart would be extremely slow.

```
In[•]:= ClearAll[inject]
inject[sa_SparseArray, replacelist_List -> valuelist_] := sa + SparseArray[
   replacelist -> Subtract[valuelist, Extract[sa, replacelist]], Dimensions[sa]]
```

The essential boundary is imposed by: (1) suppose that the imposed displacement $\overline{u}$ corresponds to the i-th DOF in the u vectors; (2) the entries of i-th row of the global stiffness matrix are set to zero except at (i,i) position; (3) the i-th force is set to be $K_{ii}\,\overline{u}$. So in imposing essential boundary, the global stiffness matrix and the force vector are modified.

Function to set the essential boundary.

```
In[•]:= ClearAll[getEssentialBoundary]
getEssentialBoundary[am_SparseArray, bvec_SparseArray, bound_List] :=
 Module[{indx, u, apos, aval, newam, newbvec},
  indx = 2 * (bound[[All, 1]] - 1) + bound[[All, 2]];
  u = bound[[All, 3]];
  apos = Flatten[
    Table[If[i ≠ j, {i, j}, Nothing], {i, indx}, {j, 1, Dimensions[am][[2]]}], 1];
  aval = 0.0;
  newam = inject[am, apos -> aval];
  newbvec = inject[bvec, (List /@ indx) -> Diagonal[am][[indx]] * u];
  {newam, newbvec}
 ]
```

Solve

Using LinearSolve to solve for displacements at the nodes.

Solutions

Overall

Further define a function to get stress and strain at Gaussian points after the nodal displacements are solved.

Function to get strain and stress at one Gaussian point.

In[•]:=
```
ClearAll[getStrainStress]
getStrainStress[{ξ_?NumericQ, η_?NumericQ}, nodes_Association,
  connectivity_List, disp_Association, dm_List] := Module[
  {nodesid, nodesxy, nodesdisp, bmat, detJ, gpstn, gpsts, gpxy},
  nodesid = connectivity;
  nodesxy = Lookup[nodes, nodesid];
  nodesdisp = Flatten[Lookup[disp, nodesid]];
  {bmat, detJ} = getBandJ[ξ, η, nodesxy];
  gpstn = bmat.nodesdisp;
  gpsts = dm.gpstn;
  gpxy = shapeFun[ξ, η].nodesxy;
  {gpxy, gpstn, gpsts}
 ]
```

Collectively, the whole procedure can be put in one function. Since we are dealing with different natural boundaries here, so it is set as an input.

Function to conduct the whole FEM.

```
In[•]:= ClearAll[femProcedure]
femProcedure[width_?NumericQ, height_?NumericQ, helem_Integer,
  velem_Integer, youngs_?NumericQ, nu_?NumericQ, natural_List] := Module[
  {nodes, elems, dm, am, fright, ftop, bvec, essential, newam,
   newbvec, uvec, disp, gpos, ss, stnx, stny, stnxy, stsx, stsy, stsxy},
  nodes = genNodes[width, height, helem, velem];
  elems = genElems[helem, velem];
  dm = dMatrix[youngs, nu];
  am = assembleall[nodes, elems, dm];
  bvec = getNaturalBoundary[natural, nodes];
  essential = essentialBoundary[helem, velem];
  {newam, newbvec} = getEssentialBoundary[am, bvec, essential];
  uvec = LinearSolve[newam, newbvec];
  disp = AssociationThread[Range[1, Length@nodes], Partition[uvec, 2]];
  gpos = {{-√3 / 3.0, -√3 / 3.0},
    {√3 / 3.0, -√3 / 3.0}, {√3 / 3.0, √3 / 3.0}, {-√3 / 3.0, √3 / 3.0}};
  ss = Flatten[Map[Function[gpt, getStrainStress[gpt, nodes, #, disp, dm]], gpos] & /@
     Values[elems], 1];
  stnx = Append[#[[1]], #[[2]][[1]]] & /@ ss;
  stny = Append[#[[1]], #[[2]][[2]]] & /@ ss;
  stnxy = Append[#[[1]], #[[2]][[3]]] & /@ ss;
  stsx = Append[#[[1]], #[[3]][[1]]] & /@ ss;
  stsy = Append[#[[1]], #[[3]][[2]]] & /@ ss;
  stsxy = Append[#[[1]], #[[3]][[3]]] & /@ ss;


  Association[
   "Nodes" -> nodes,
   "Elements" -> elems,
   "Displacement" -> disp,
   "StrainXX" -> stnx,
   "StrainYY" -> stny,
   "StrainXY" -> stnxy,
   "StressXX" -> stsx,
   "StressYY" -> stsy,
   "StressXY" -> stsxy
  ]
 ]
```

Function to plot the displacements, strains and stresses.

```
In[•]:= ClearAll[plots];
plots[case_Association] := Module[
  {plotux, plotuy, plotstnx, plotstny, plotstnxy, plotstsx, plotstsy, plotstsxy},
  plotux = ListContourPlot[
    Transpose@Append[Transpose@Values[case["Nodes"]],
      Values[case["Displacement"]][[All, 1]]],
    PlotLegends ->
     Placed[BarLegend[Automatic, LegendLabel → "X-Displacement"], Right],
```

```
  PlotRangePadding -> None,
  ImageSize -> Medium
 ];
plotuy = ListContourPlot[
  Transpose@Append[Transpose@Values[case["Nodes"]],
    Values[case["Displacement"]][[All, 2]]],
  PlotLegends ->
   Placed[BarLegend[Automatic, LegendLabel → "Y-Displacement"], Right],
  PlotRangePadding -> None,
  ImageSize -> Medium
 ];
plotstnx = ListDensityPlot[
  case["StrainXX"],
  PlotLegends -> Placed[BarLegend[Automatic, LegendLabel → "εx"], Right],
  PlotRangePadding -> None,
  PlotRange -> All,
  ImageSize -> Medium
 ];
plotstny = ListDensityPlot[
  case["StrainYY"],
  PlotLegends -> Placed[BarLegend[Automatic, LegendLabel → "εy"], Right],
  PlotRangePadding -> None,
  PlotRange -> All,
  ImageSize -> Medium
 ];
plotstnxy = ListDensityPlot[
  case["StrainXY"],
  PlotLegends -> Placed[BarLegend[Automatic, LegendLabel → "γxy"], Right],
  PlotRangePadding -> None,
  PlotRange -> All,
  ImageSize -> Medium
 ];
plotstsx = ListDensityPlot[
  case["StressXX"],
  PlotLegends -> Placed[BarLegend[Automatic, LegendLabel → "σx"], Right],
  PlotRangePadding -> None,
  PlotRange -> All,
  ImageSize -> Medium
 ];
plotstsy = ListDensityPlot[
  case["StressYY"],
  PlotLegends -> Placed[BarLegend[Automatic, LegendLabel → "σy"], Right],
  PlotRangePadding -> None,
  PlotRange -> All,
  ImageSize -> Medium
 ];
plotstsxy = ListDensityPlot[
  case["StressXY"],
  PlotLegends -> Placed[BarLegend[Automatic, LegendLabel → "τxy"], Right],
```

```
    PlotRangePadding -> None,
    PlotRange -> All,
    ImageSize -> Medium
   ];
  Column[{{plotux, plotuy},
    {plotstnx, plotstny, plotstnxy}, {plotstsx, plotstsy, plotstsxy}}]
 ]
```

Computation

Model Parameters

In[•]:=
```
width = 1.0;
height = 1.0;
helem = 20;
velem = 20;
youngs = 200.0 × 10^6;
nu = 0.3;
nodes = genNodes[width, height, helem, velem];
```

Case 1: Uniform right boundary stress

In[•]:=
```
ftop = setNaturalBoundary[
    (helem + 1) * velem + Range[1, helem + 1], 2, -100.0 × 10^3, -100.0 × 10^3, nodes];
fright1 = setNaturalBoundary[
    (helem + 1) * Range[1, velem + 1], 1, -200.0 × 10^3, -200.0 × 10^3, nodes];
natural1 = Join[fright1, ftop];
```

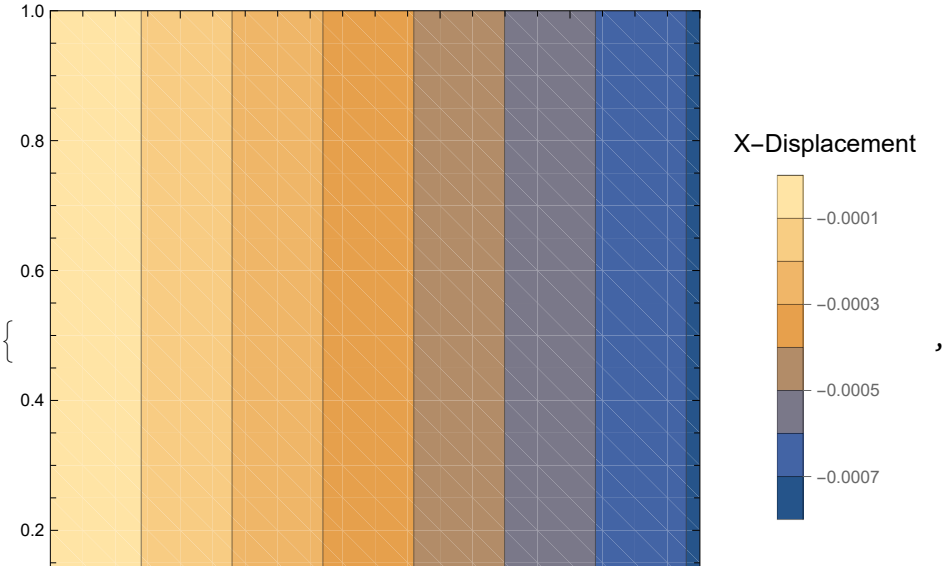
••• Interpolation: Requested order is too high; order has been reduced to {1, 0}.

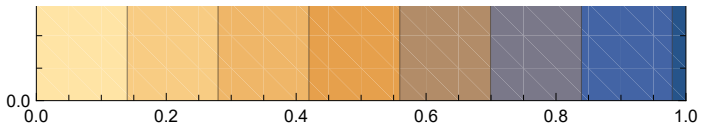
••• Interpolation: Requested order is too high; order has been reduced to {0, 1}.

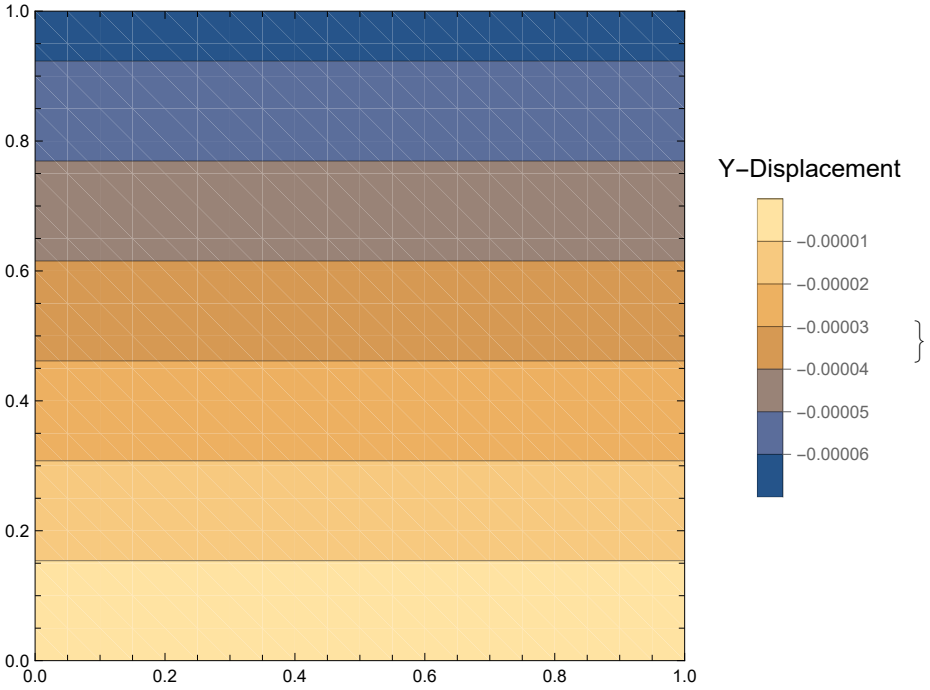
In[•]:=
```
case1 = femProcedure[width, height, helem, velem, youngs, nu, natural1];
```

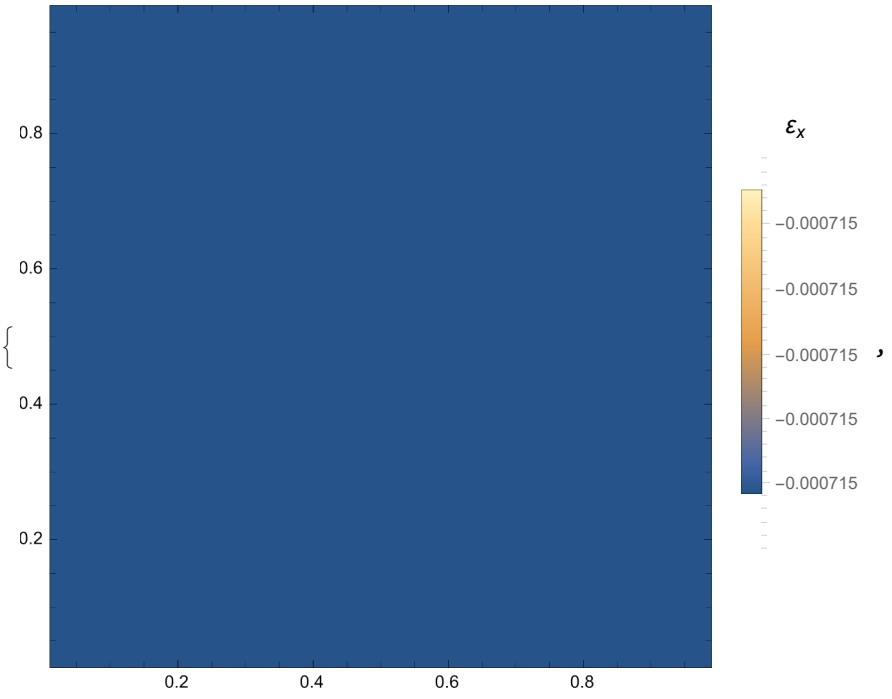
In[•]:=
```
plots[case1]
```

Out[•]=

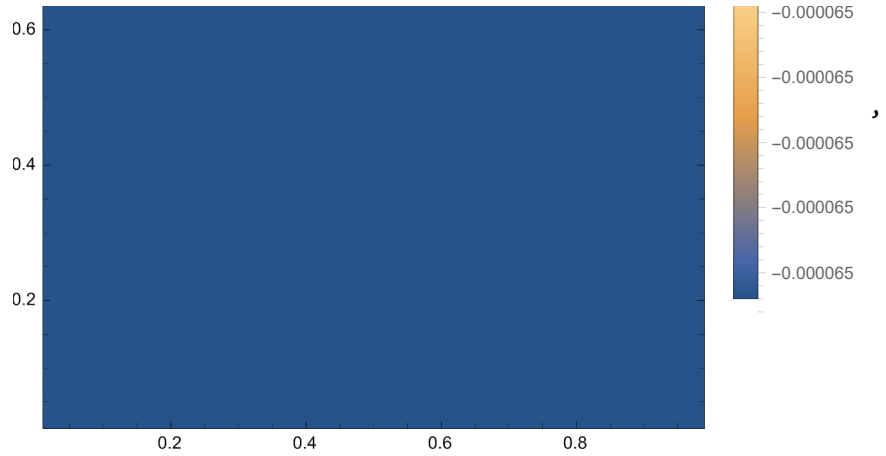



0.0
0.0
0.2
0.4
0.6
0.8
1.0

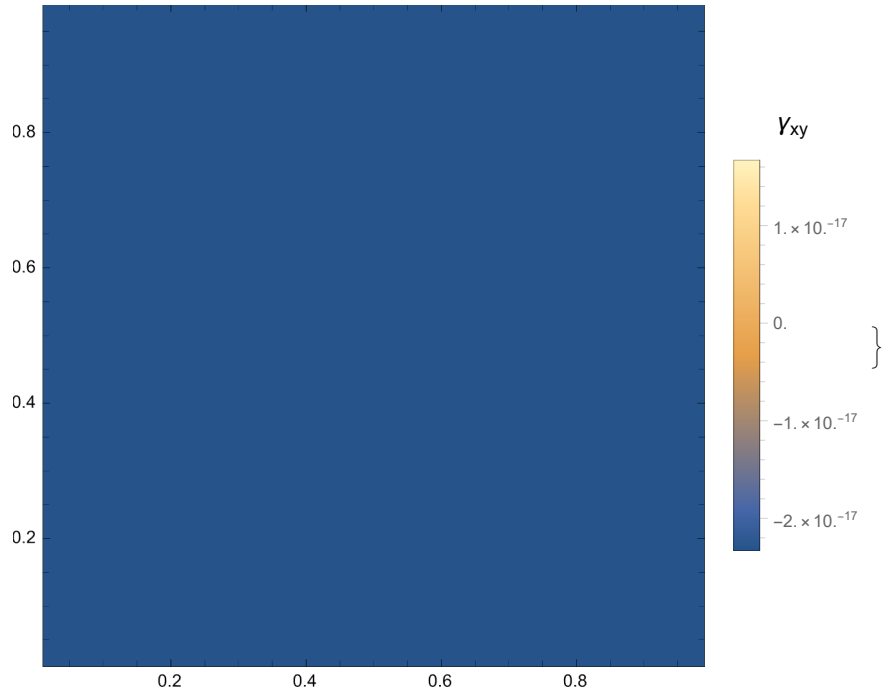

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Y−Displacement
−0.00001
−0.00002
−0.00003
−0.00004
−0.00005
−0.00006


0.8
0.6
0.4
0.2
0.2
0.4
0.6
0.8
ε_x
−0.000715
−0.000715
−0.000715
−0.000715
−0.000715

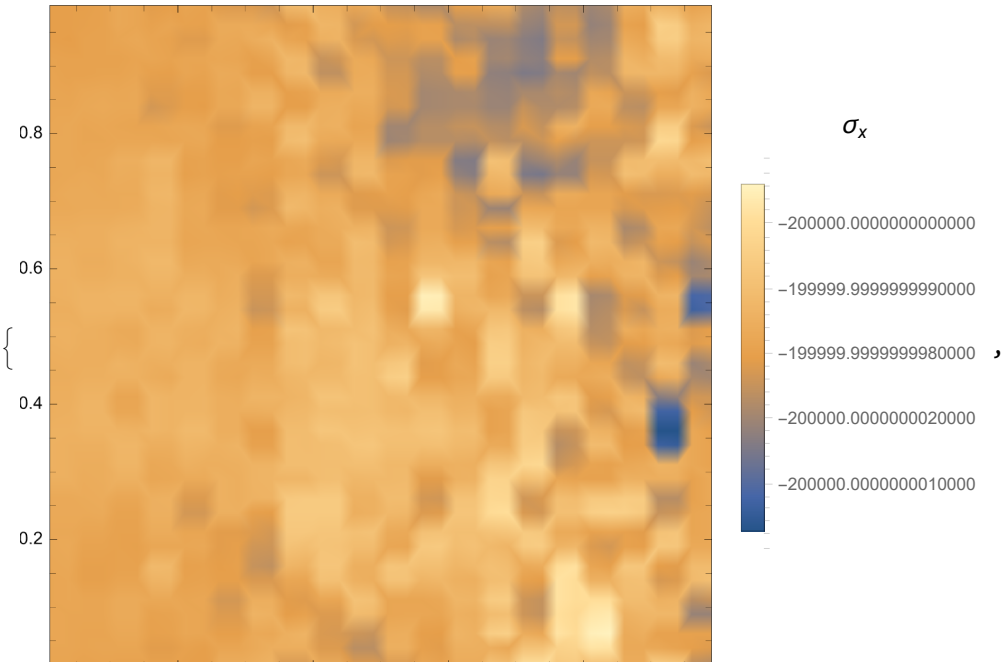

0.8
ε_y
−0.000065


0.6
0.4
0.2
0.2
0.4
0.6
0.8
−0.000065
−0.000065
−0.000065
−0.000065
−0.000065

,

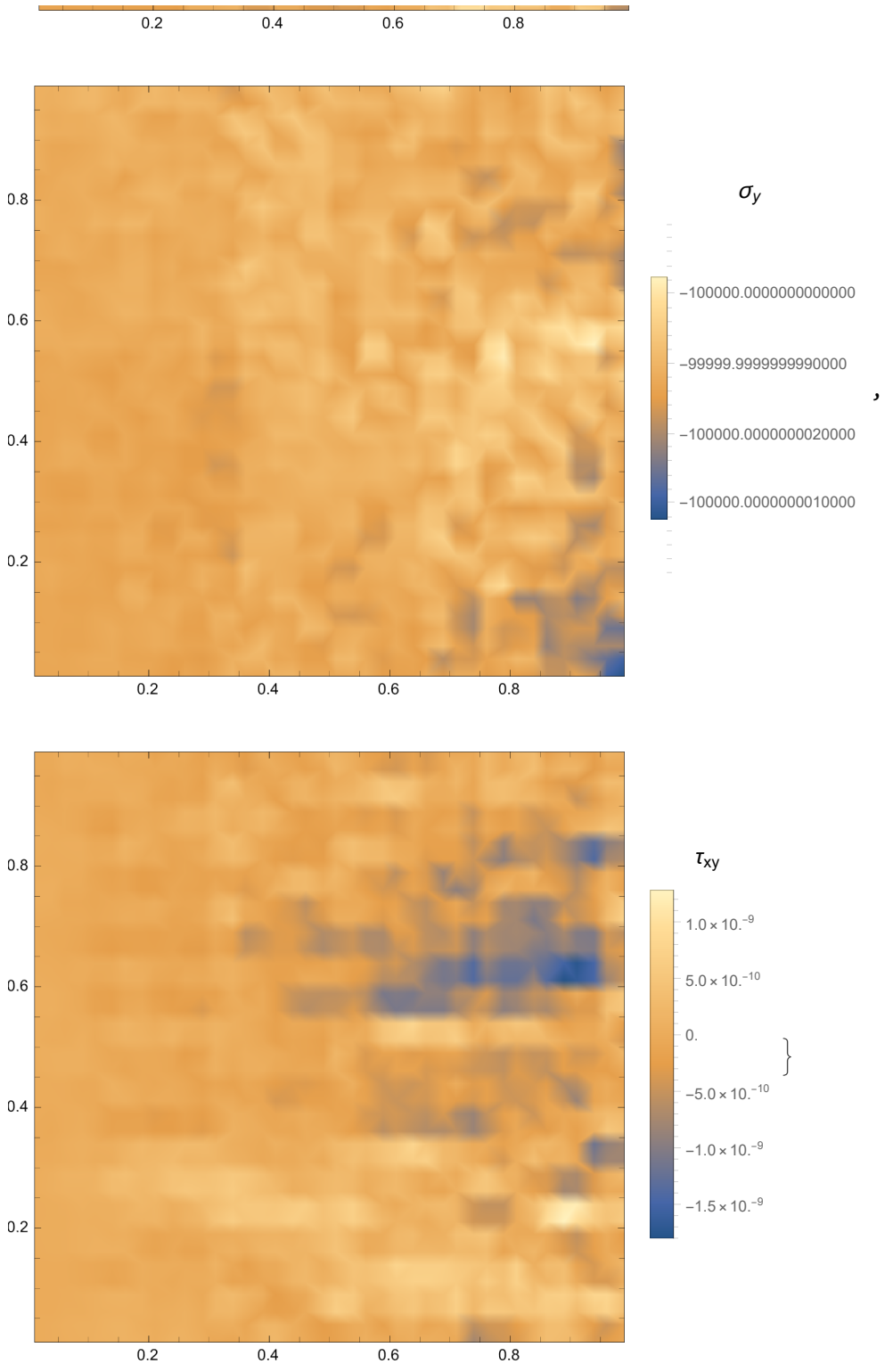

0.8
0.6
0.4
0.2
0.2
0.4
0.6
0.8
γ_{xy}
1. × 10.^−17
0.
−1. × 10.^−17
−2. × 10.^−17

}


0.8
0.6
0.4
0.2
σ_x
−200000.0000000000000
−199999.9999999990000
−199999.9999999980000
−200000.0000000020000
−200000.0000000010000

{

,



Case 2: Triangular right boundary stress

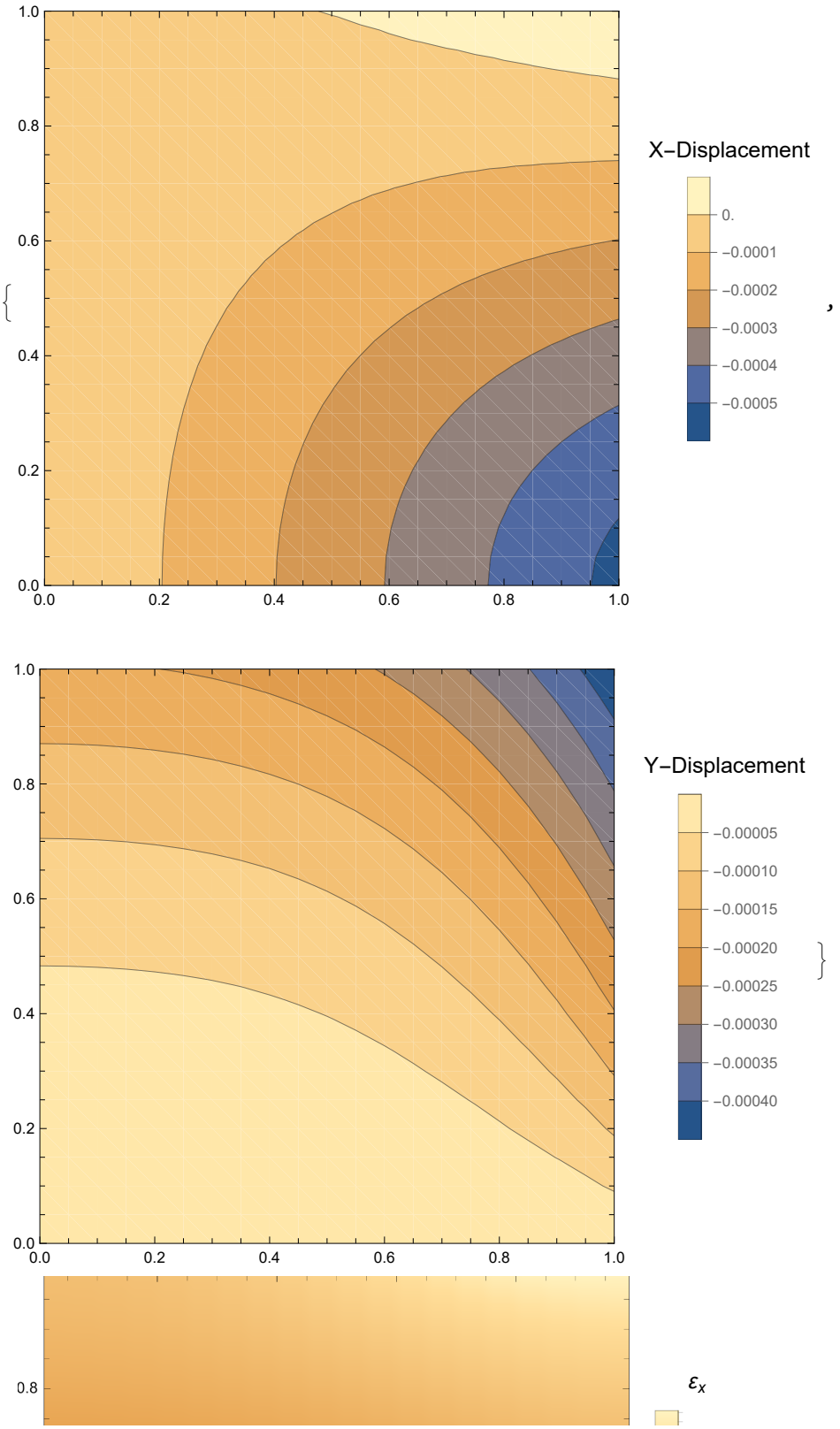
In[ ]:=
```
ftop = setNaturalBoundary[
    (helem + 1) * velem + Range[1, helem + 1], 2, -100.0 × 10^3, -100.0 × 10^3, nodes];
fright2 =
  setNaturalBoundary[(helem + 1) * Range[1, velem + 1], 1, -200.0 × 10^3, -0.0 × 10^3, nodes];
natural2 = Join[fright2, ftop];
```

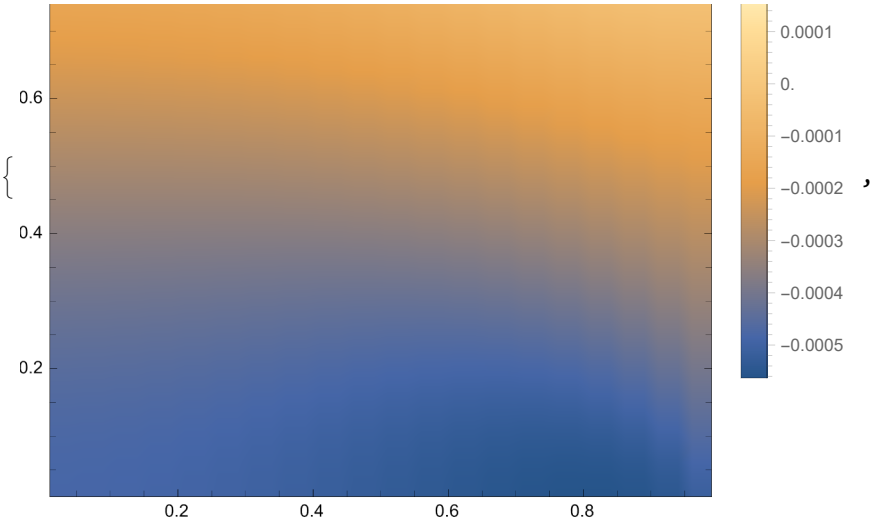
Interpolation: Requested order is too high; order has been reduced to {1, 0}.

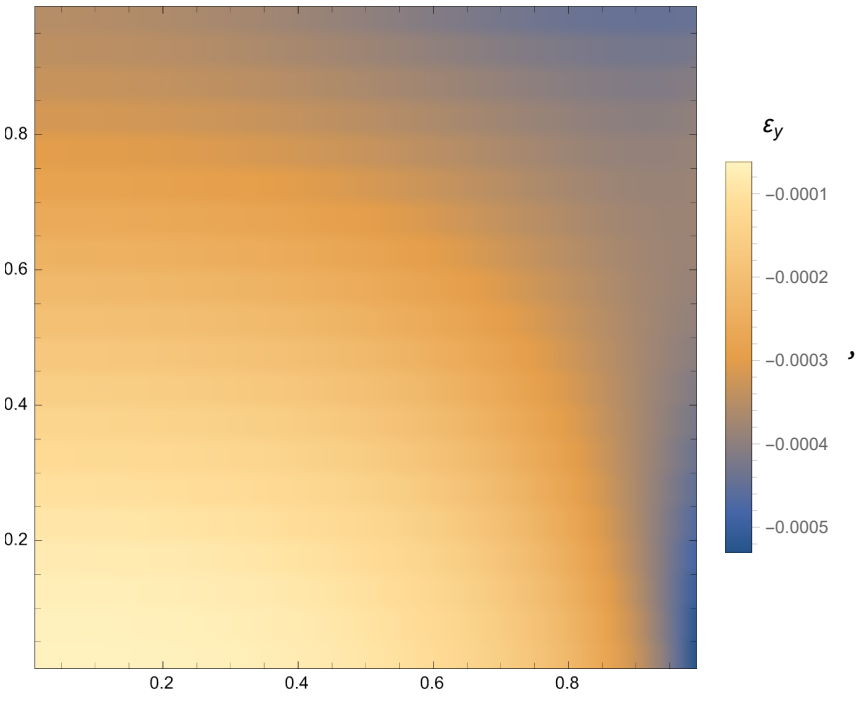
Interpolation: Requested order is too high; order has been reduced to {0, 1}.

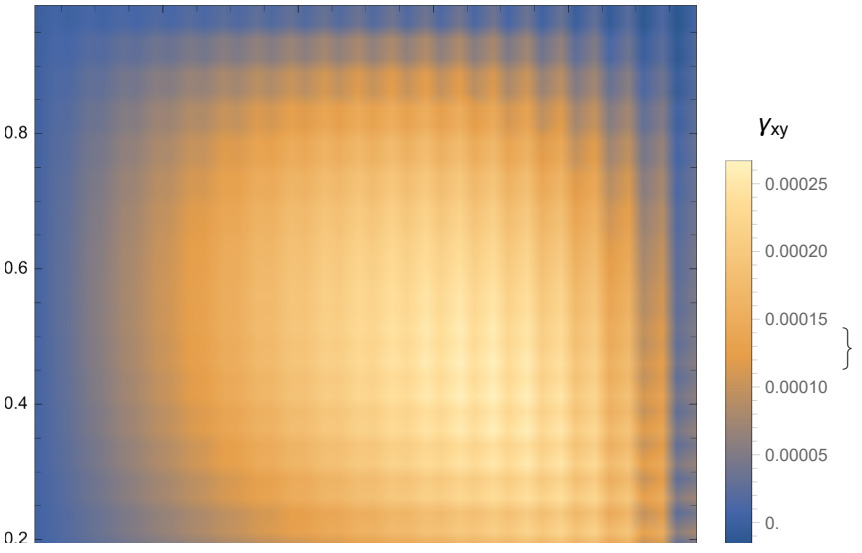
```
In[•]:= case2 = femProcedure[width, height, helem, velem, youngs, nu, natural2];

In[•]:= plots[case2]
```

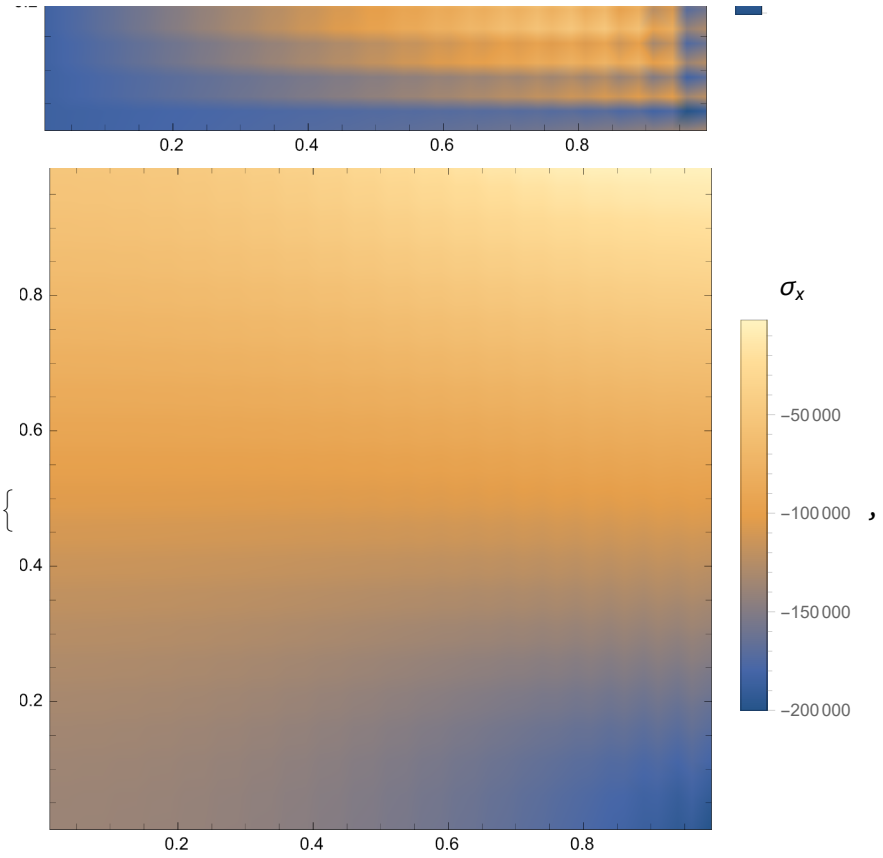
Out[•]=



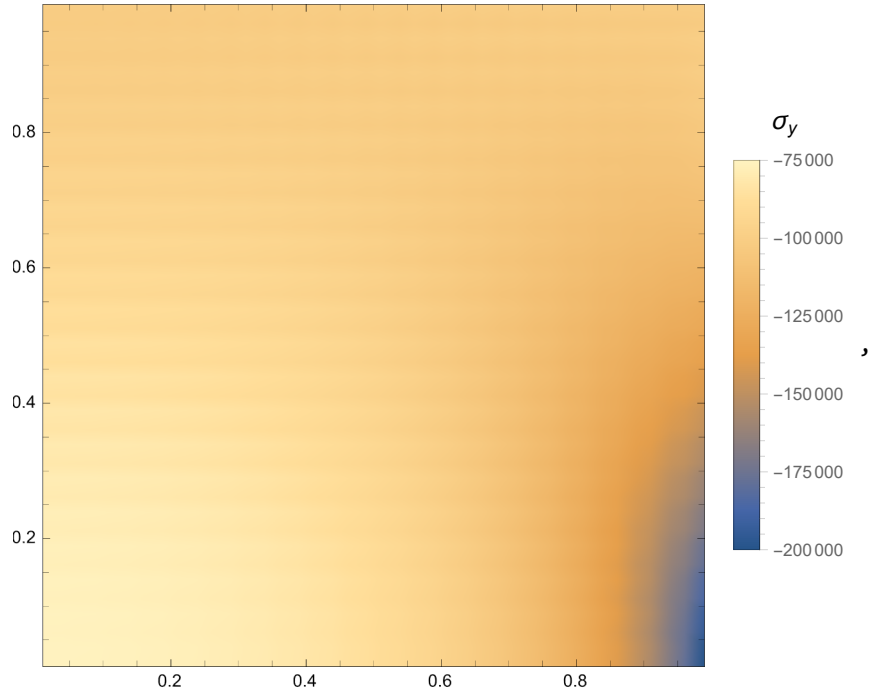

0.6
0.4
0.2
0.2
0.4
0.6
0.8
0.0001
0.
−0.0001
−0.0002
−0.0003
−0.0004
−0.0005


0.8
0.6
0.4
0.2
0.2
0.4
0.6
0.8
ε_y
−0.0001
−0.0002
−0.0003
−0.0004
−0.0005

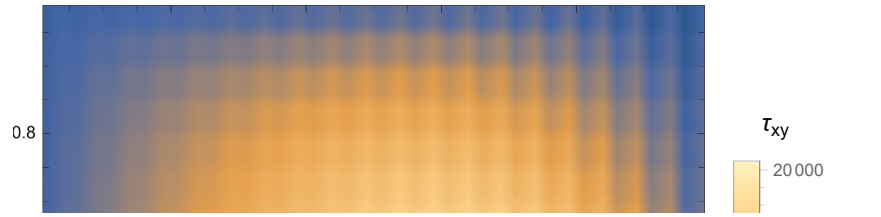

0.8
0.6
0.4
0.2
γ_{xy}
0.00025
0.00020
0.00015
0.00010
0.00005
0.

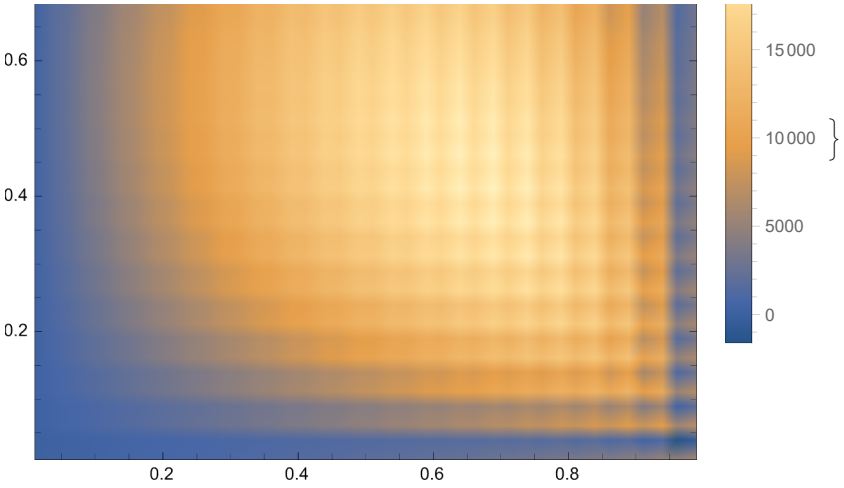

0.2
0.4
0.6
0.8
0.8
0.6
0.4
0.2
σ_x
−50 000
−100 000
−150 000
−200 000

,


0.8
0.6
0.4
0.2
0.2
0.4
0.6
0.8
σ_y
−75 000
−100 000
−125 000
−150 000
−175 000
−200 000

,


0.8
τ_{xy}
20 000


0.6
0.4
0.2
0.2
0.4
0.6
0.8
15000
10000
5000
0