HiperLife Tutorial: Linear Elasticity

Arash Imani

LaCàN

November 10, 2024

1 Problem Definition

Elasticity is the part of solid mechanincs that deals with stress and deformation of solid continue Linearized elasticity is concerned with small deformations obeying Hooke's Law. There is a class of problems in elasticity whose solutions are not dependent on one of the coordinates because of their geometry, boundary conditions, and external applied loads. Such problems are called *plane elasticity* which contain plane strain and plane stress problems. Both classes are described by a set of two coupled partial differential equations expressed in terms of two dependent variables that represent the two components of the displacement vector.

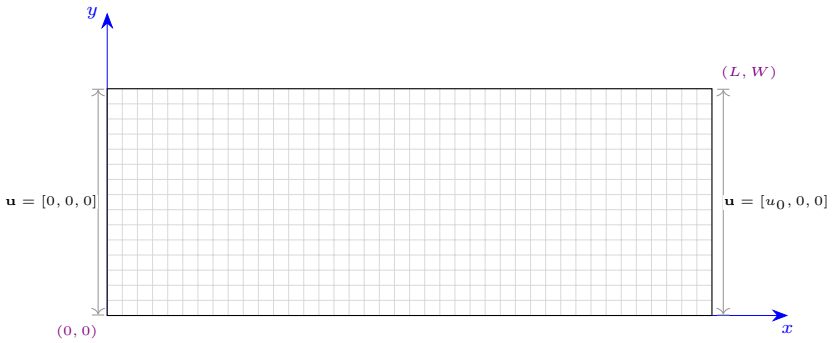

Figure 1: Geometry, BC and computational domain used for the analysis of linear elasticity.

For validating our result in this case because of the specific geometry, the deflection of cantilever beam with single fixed support gives a proper estimate. The solution can be retrieve by simple equation as

$$\delta = \frac{FL^3}{3EI} \tag{1}$$

where F is the end load, I denotes the moment of inertia and E is Young's Modulus. In our computational model, we have used Dirichlet BC on $x = 0$, and Neumann BC along the line $x = L$, as depicted in Figure 1. The domain with Length and width of L, W and uniform thickness of t. note that the material properties are as follows: Elasticity module and Poisson's ratio are E and ν.

2 Governing Equations

The governing equations for the plane elasticity problems discussed above are summarized below, Cauchy momentum equation for a domain $\Omega \subset \mathbb{R}^n$ with boundary $\partial\Omega$ is like:

$$\begin{aligned} \frac{D\boldsymbol{v}}{Dt} &= \frac{1}{\rho}\nabla \cdot \boldsymbol{\sigma} + \mathbf{f} \quad \text{in } \Omega\,, \\ \boldsymbol{\sigma} \cdot \mathbf{n} &= \hat{\mathbf{t}} \quad \text{on } \Gamma_N\,, \\ \mathbf{u} &= \hat{\mathbf{u}} \quad \text{on } \Gamma_D\,. \end{aligned} \tag{2}$$

where $\boldsymbol{v}$ is the velocity vector field , t is time, $\frac{D\boldsymbol{v}}{Dt}$ is the material derivative of $\boldsymbol{v}$, ρ is the density at a given point, $\boldsymbol{\sigma}$ is the stress tensor, and $\mathbf{f}$ is a vector containing all of the accelerations caused by body forces. So for the expanded form of equations of motion for 2D space we would have

$$\begin{aligned}\rho\frac{\partial^2 u_x}{\partial t^2} &= \frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} + f_x\,,\\ \rho\frac{\partial^2 u_y}{\partial t^2} &= \frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + f_y\,.\end{aligned} \tag{3}$$

where f_x and f_y denote the components of the body force vector measured per unit volume along the x and y directions. respectively. here for the sake of simplicity we consider static problem[1].

3 Weak Form

The starting point for the development of the finite element models of Eq. (1) is their weak forms. The variation formulation of our model problem can be introduced as find $\mathbf{u} \in V$ such that

$$\mathcal{F}(\mathbf{u}; v) = 0 \quad \forall v \in \hat{V}\,. \tag{4}$$

where

$$\mathcal{F}(\mathbf{u}; v) = \int_\Omega v\nabla\cdot\boldsymbol{\sigma} + v\rho\mathbf{f}\mathrm{d}\Omega\,. \tag{5}$$

and

$$\begin{aligned}\hat{V} &= \{v \in H^1(\Omega) : v = 0 \text{ on } \Gamma\},\\ V &= \{v \in H^1(\Omega) : v = 0 \text{ on } x = 0\}\,.\end{aligned} \tag{6}$$

where v is a test function, which will be equated, in the our FE model to the interpolation function used for $\mathbf{u}$.

Applying integration by part, and also considering boundary condition, the weak form takes the following form

$$\begin{aligned}\mathcal{F}(\mathbf{u}; v) &= \int_\Omega \nabla\cdot[v\boldsymbol{\sigma}]\mathrm{d}\Omega - \int_\Omega \boldsymbol{\sigma}\nabla v\mathrm{d}\Omega + v\rho\mathbf{f}\mathrm{d}\Omega\\ &= \int_\Gamma v\boldsymbol{\sigma}\cdot\mathbf{n}\mathrm{d}\Gamma - \int_\Omega \boldsymbol{\sigma}\nabla v\mathrm{d}\Omega + \int_\Omega v\rho\mathbf{f}\mathrm{d}\Omega\\ &= \int_\Gamma v\hat{\mathbf{t}}\mathrm{d}\Gamma - \int_\Omega \boldsymbol{\sigma}\nabla v\mathrm{d}\Omega + \int_\Omega v\rho\mathbf{f}\mathrm{d}\Omega\,.\end{aligned} \tag{7}$$

In general, the stress-strain relation for a linear elastic material can be considered as

$$\sigma_{ij} = C_{ijkl}\varepsilon_{kl} \tag{8}$$

where σ_{ij} and C_{ijkl} are the components of the Cauchy stress tensor and the elasticity tensor ($C_{ijkl} = \lambda\delta_{ij}\delta_{kl} + \mu[\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}]$).The $\boldsymbol{\varepsilon}_{kl}$ are the components of the strain tensor, which by definition is $\boldsymbol{\varepsilon} = \frac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^T)$, where $\mathbf{u}$ is displacement vector. In simple plane stress case and for isotropic elastic materials the Eq. (8) takes following format

$$\begin{aligned}\sigma_i &= \mathbf{c}_{ij}\varepsilon_j\\ \begin{Bmatrix}\sigma_{xx}\\ \sigma_{yy}\\ \sigma_{xy}\end{Bmatrix} &= \begin{bmatrix}c_{11} & c_{12} & 0\\ c_{12} & c_{22} & 0\\ 0 & 0 & c_{33}\end{bmatrix}\begin{Bmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{xy}\end{Bmatrix}\end{aligned} \tag{9}$$

where $c_{11} = c_{22} = \frac{E}{1-\nu^2}$ and $c_{12} = \nu c_{11}$ and $c_{33} = \frac{1-\nu}{2}c_{11}$. Considering ($hdA = dV$) and $\mathbf{u} = \{u_x, u_y\}$ we can rewrite The Eq. (6) in an explicit form as [1]

$$\begin{aligned}\int_\Omega h\left[\frac{\partial v}{\partial x}\left(c_{11}\frac{\partial u_x}{\partial x} + c_{12}\frac{\partial u_y}{\partial y}\right) + c_{33}\frac{\partial v}{\partial y}\left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\right)\right]\,dA &= \int_\Omega hvf_x\,dA + \int_\Gamma hvt_x\,dS\,,\\ \int_\Omega h\left[c_{33}\frac{\partial v}{\partial x}\left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\right) + \frac{\partial v}{\partial y}\left(c_{12}\frac{\partial u_x}{\partial x} + c_{22}\frac{\partial u_y}{\partial y}\right)\right]\,dA &= \int_\Omega hvf_y\,dA + \int_\Gamma hvt_y\,dS\,.\end{aligned} \tag{10}$$

[1] Applying time discritization into HiperLife is explained in the transient heat transfer tutorial.

4 Finite Element Model

Since we are developing the Ritz-Galerkin finite element model, the choice of the weight functions is restricted to the spaces of approximation functions used for the solution field. Suppose that the displacement approximated by expansions of the form

$$u(x,y) = \sum_{m=1}^{M} \phi_m(x,y)\mathbf{u}^m = \mathbf{\Phi}^T\mathbf{u}\,, \tag{11}$$

By substitution of Eq. (15) to Eq. (10)and writing the resulting algebraic equations in matrix form, we obtain

$$\begin{bmatrix}[K^{11}] & [K^{12}]\\ [K^{21}] & [K^{22}]\end{bmatrix}\begin{Bmatrix}\{u_x\}\\ \{u_y\}\end{Bmatrix} = \begin{Bmatrix}\{F^1\}\\ \{F^2\}\end{Bmatrix} \tag{12}$$

where

$$\begin{aligned}
K^{11}_{ij} &= \int_{\Omega^e} h\left[c_{11}\frac{\partial\phi_i}{\partial x}\frac{\partial\phi_j}{\partial x} + c_{33}\frac{\partial\phi_i}{\partial y}\frac{\partial\phi_j}{\partial y}\right]\,dxdy\\
K^{22}_{ij} &= \int_{\Omega^e} h\left[c_{33}\frac{\partial\phi_i}{\partial x}\frac{\partial\phi_j}{\partial x} + c_{22}\frac{\partial\phi_i}{\partial y}\frac{\partial\phi_j}{\partial y}\right]\,dxdy\\
K^{12}_{ij} = K^{21}_{ji} &= \int_{\Omega^e} h\left[c_{12}\frac{\partial\phi_i}{\partial x}\frac{\partial\phi_j}{\partial y} + c_{33}\frac{\partial\phi_i}{\partial y}\frac{\partial\phi_j}{\partial x}\right]\,dxdy\\
F^1_i &= \int_{\Omega^e} h\phi_i f_x\,dxdy + \int_{\Gamma^e} h\phi_i t_x\,ds\\
F^2_i &= \int_{\Omega^e} h\phi_i f_y\,dxdy + \int_{\Gamma^e} h\phi_i t_y\,ds
\end{aligned} \tag{13}$$

The elemental representation of the vector and matrix required for implementation in the Hiperlife would be like

$$\begin{aligned}
Ak(i,0,j,0) = K^{11}_{ij} \quad &, \quad Ak(i,0,j,1) = K^{12}_{ij}\\
Ak(i,1,j,0) = K^{21}_{ij} \quad &, \quad Ak(i,1,j,1) = K^{22}_{ij}\\
Bk(i,0) = F^1_i \quad &, \quad Bk(i,1) = F^2_i
\end{aligned} \tag{14}$$

5 Choice of Elements

Thus, for this simple problem every Lagrange and serendipity family of interpolation functions are admissible for the interpolation of the temperature field, our choice is would be linear quadrilateral element.Linear Quadrilateral Elements is the simplest quadrilateral element consists of four nodes. The associated interpolation functions for geometry and field variables are bilinear.

$$\mathbf{\Phi}_I(\xi,\eta) = \frac{1}{4}(1+\xi_I\xi)(1+\eta_I\eta) \quad (I \text{ from 1 to 4}) \tag{15}$$

where ξ_I and η_I are the corner coordinates at element T in domain of $\Omega_T \in (-1,1)^2$. As it shown in Figure 2 we are using 2×2 Gauss–Legendre quadrature integration.

We also choose a uniform mesh of 100×12 to model our domain $[0,2]\times[0,0.12]$.

6 Implementation

In this section, we present the implementation of our solution in the Hiperlife. The program is divided into three separate files, main part which we create our problem by the Hiperlife headers, auxiliary header where we introduce parameters and declare defined functions, and at last auxiliary file, where we define some functions which provide required matrices like Jacobian and Hessian.

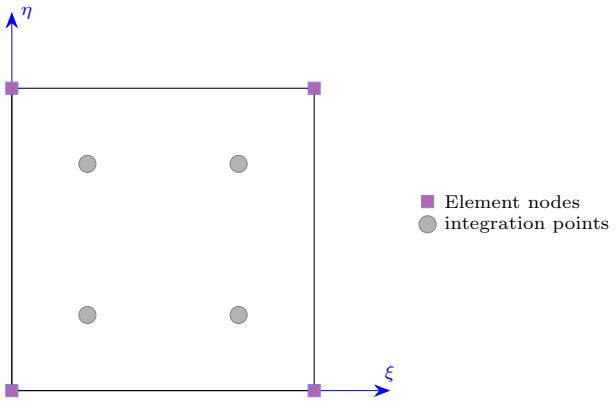

Figure 2: Linear quadrilateral element used for finite element model.

6.1 PlaneStress2D.cpp

```cpp
/*
*****************************************************************************
linear elasticity 2-D Plane stress
*****************************************************************************
*/

// cpp headers
#include <iostream>
#include <fstream>
#include <time.h>

// hiperlife headers
#include "hl_Core.h"
#include "hl_ParamStructure.h"
#include "hl_Parser.h"
#include "hl_TypeDefs.h"
#include "hl_MeshLoader.h"
#include "hl_StructMeshGenerator.h"
#include "hl_DistributedMesh.h"
#include "hl_FillStructure.h"
#include "hl_DOFsHandler.h"
#include "hl_HiPerProblem.h"
#include "hl_SurfLagrParam.h"
#include "hl_LinearSolver_Iterative_AztecOO.h"
#include "hl_LinearSolver_Direct_MUMPS.h"
#include "hl_GlobalBasisFunctions.h"

// Header to auxiliary functions
#include "Aux2D.h"

// ------------------------- MAIN FUNCTION ---------------------------------//
int main(int argc, char** argv)
{
        using namespace std;
        using namespace hiperlife;
        using hiperlife::Tensor::tensor;

        //-------------------------INITIALIZATION-----------------------------//
        hiperlife::Init(argc, argv);

        //-----------------------------STRUCT---------------------------------//
        SmartPtr<ParamStructure> paramStr = CreateParamStructure<PlaneStressParams>();
        const double traction = paramStr->getRealParameter(PlaneStressParams::traction);
        //-------------------------MESH CREATION------------------------------//
        SmartPtr<StructMeshGenerator> structMesh = Create<StructMeshGenerator>();
```

```
46
47          structMesh->setNDim(3);
48          structMesh->setBasisFuncType(BasisFuncType::Lagrangian);
49          structMesh->setBasisFuncOrder(1);
50          structMesh->setElemType(ElemType::Square);
51          structMesh->genRectangle(100, 12, 2.0, 0.12);
52          SmartPtr<DistributedMesh> disMesh = Create<DistributedMesh>();
53          disMesh->setMesh(structMesh);
54          disMesh->setBalanceMesh(true);
55          disMesh->setElementLocatorEngine(ElementLocatorEngine::BoundingVolumeHierarchy);
56          disMesh->Update();
57          //-------------------------DOFsHANDLER CREATION-------------------------//
58          // ------------------------ Displacement------------------------------ //
59          SmartPtr<DOFsHandler> dofHand = Create<DOFsHandler>(disMesh);
60          dofHand->setNameTag("dofHand");
61          dofHand->setNumDOFs(2);
62          dofHand->setDOFs({"u","v"});
63          dofHand->Update();
64          //------------------------------ B.C ------------------------------------//
65          //--------------------- Dirichlet on the left side ---------------------//
66          dofHand->setBoundaryCondition(0, MAxis::Xmin, 0.0);
67          dofHand->setBoundaryCondition(1, MAxis::Xmin, 0.0);
68
69          //dofHand->setBoundaryCondition(1, MAxis::Xmax, 0.01);
70
71          dofHand->UpdateGhosts();
72          //------------------------------------------------------------------------ //
73          //-----------------------HIPERPROBLEM CREATION----------------------------//
74          // ------------------------------ Displacement ------------------------- //
75          SmartPtr<HiPerProblem> hiperProbl = Create<HiPerProblem>();
76
77          hiperProbl->setParameterStructure(paramStr);
78          hiperProbl->setDOFsHandlers({dofHand});
79
80          hiperProbl->setIntegration("Integ", {"dofHand"});
81          hiperProbl->setCubatureGauss("Integ", 4);
82          hiperProbl->setElementFillings("Integ", ElementFilling);
83
84          // Nuemann boundary condition: Set integration on the border
85          hiperProbl->setIntegration("BorderInteg", {"dofHand"});
86          hiperProbl->setCubatureBorderGauss("BorderInteg",1,{MAxis::Xmax});
87          hiperProbl->setElementFillings("BorderInteg", nullptr, nullptr, RHS_Border);
88
89          hiperProbl->Update();
90
91          //-------------------------SOLVER CREATION------------------------------//
92          // ------------------------ Displacement ------------------------------ //
93          SmartPtr<MUMPSDirectLinearSolver> solver = Create<MUMPSDirectLinearSolver>();
94          solver->setHiPerProblem(hiperProbl);
95          solver->setVerbosity(MUMPSDirectLinearSolver::Verbosity::Extreme);
96          solver->setDefaultParameters();
97          solver->Update();
98
99          solver->solve();
100         hiperProbl->UpdateSolution();
101
102         dofHand->printFileLegacyVtk("PlaneStress2D", true);
103
104         hiperlife::Finalize();
105         return 0;
106 }
```

6.2 Aux2D.h

```
1 #ifndef AUXFLUID_H
2 #define AUXFLUID_H
```

```
3
4
5  #include <iostream>
6  #include <fstream>
7  #include <time.h>
8
9  // Hiperlife headers
10 #include "hl_Core.h"
11 #include "hl_ParamStructure.h"
12 #include "hl_Parser.h"
13 #include "hl_TypeDefs.h"
14 #include "hl_MeshLoader.h"
15 #include "hl_StructMeshGenerator.h"
16 #include "hl_DistributedMesh.h"
17 #include "hl_FillStructure.h"
18 #include "hl_DOFsHandler.h"
19 #include "hl_HiPerProblem.h"
20 #include "hl_SurfLagrParam.h"
21 #include "hl_LinearSolver_Iterative_AztecOO.h"
22 #include "hl_GlobalBasisFunctions.h"
23
24 using namespace std;
25 using namespace hiperlife;
26 using hiperlife::Tensor::tensor;
27
28 struct PlaneStressParams
29 {
30         enum RealParameters
31         {
32                 thickness,
33                 E,
34                 nu,
35                 traction
36         };
37
38         HL_PARAMETER_LIST DefaultValues
39         {
40                 {"thickness", 0.02},
41                 {"E", 200000000000.0},
42                 {"nu", 0.3},
43                 {"traction", 2000000000.0},
44         };
45 };
46
47 void ElementFilling(hiperlife::FillStructure& fillStr);
48 void RHS_Border(hiperlife::FillStructure& fillStr);
49
50
51 #endif
```

6.3 Aux2D.cpp

```
1  // Header to auxiliary functions
2  #include "Aux2D.h"
3
4  #include <iostream>
5  #include <fstream>
6  #include <time.h>
7
8  // Hiperlife headers
9  #include "hl_Core.h"
10 #include "hl_ParamStructure.h"
11 #include "hl_Parser.h"
12 #include "hl_TypeDefs.h"
13 #include "hl_MeshLoader.h"
14 #include "hl_StructMeshGenerator.h"
```

```
15 #include "hl_DistributedMesh.h"
16 #include "hl_FillStructure.h"
17 #include "hl_DOFsHandler.h"
18 #include "hl_HiPerProblem.h"
19 #include "hl_SurfLagrParam.h"
20 #include "hl_LinearSolver_Iterative_AztecOO.h"
21 #include "hl_GlobalBasisFunctions.h"
22
23 //-------------------------------------------------------------------------//
24 //------------------------------ FUNCTIONS --------------------------------//
25 //-------------------------------------------------------------------------//
26
27 void ElementFilling(hiperlife::FillStructure& fillStr)
28 {
29         using namespace std;
30         using namespace hiperlife;
31         using hiperlife::Tensor::tensor;
32
33
34         const long double E = fillStr.getRealParameter(PlaneStressParams::E);
35         const double nu = fillStr.getRealParameter(PlaneStressParams::nu);
36         const double thickness = fillStr.getRealParameter(PlaneStressParams::thickness);
37
38
39         //-------------------------------Velocity-related--------------------------//
40         // Dimensions
41         SubFillStructure& subFill = fillStr["dofHand"];
42         int eNN  = subFill.eNN;
43         int pDim = subFill.pDim;
44         int numDOFs = subFill.numDOFs; // numDOFs = 2
45
46         // Shape functions and derivatives at Gauss points
47         tensor<double,1,false>  bf(subFill.nborBFs(),eNN);
48         tensor<double,2> Dbf_g(eNN, pDim);
49         double jac;
50         GlobalBasisFunctions::gradients(Dbf_g, jac, subFill);
51
52         ttl:: wrapper<double,4> Ak(fillStr.Ak(0, 0).data(),eNN, numDOFs, eNN, numDOFs);
53         ttl:: wrapper<double,2> Bk(fillStr.Bk(0).data(), eNN, numDOFs);
54
55
56         // Elasticity Tensor
57         ttl::tensor<long double,2> C{{E/(1.0 -nu*nu), E*nu/(1.0-nu*nu), 0.0},
58                 {E*nu/(1.0-nu*nu), E/(1.0-nu*nu), 0.0},{0.0, 0.0, E/(1.0+nu)}};
59         ttl::tensor<double,1> F{0.0,0.0};
60
61         for (int i=0;i < eNN;i++)
62         {
63                 for (int j=0;j < eNN;j++)
64                 {
65                         Ak(i,0,j,0) += thickness * jac *
66                         (C(0,0)*Dbf_g(i,0)*Dbf_g(j,0) + C(2,2)*Dbf_g(i,1)*Dbf_g(j,1));
67                         Ak(i,0,j,1) += thickness * jac *
68                         (C(0,1)*Dbf_g(i,0)*Dbf_g(j,1) + C(2,2)*Dbf_g(i,1)*Dbf_g(j,0));
69                         Ak(i,1,j,0) += thickness * jac *
70                         (C(0,1)*Dbf_g(i,1)*Dbf_g(j,0) + C(2,2)*Dbf_g(i,0)*Dbf_g(j,1));
71                         Ak(i,1,j,1) += thickness * jac *
72                         (C(1,1)*Dbf_g(i,0)*Dbf_g(j,0) + C(2,2)*Dbf_g(i,1)*Dbf_g(j,1));
73                 }
74                 Bk(i,0) += thickness * jac * bf(i) * F(0);
75                 Bk(i,1) += thickness * jac * bf(i) * F(1);
76         }
77
78 }
79 void RHS_Border(hiperlife::FillStructure& fillStr)
80 {
81         using namespace std;
82         using namespace hiperlife;
```

```
83          using hiperlife::Tensor::tensor;
84
85          const double thickness = fillStr.getRealParameter(PlaneStressParams::thickness);
86          const double traction = fillStr.getRealParameter(PlaneStressParams::traction);
87          // Variables
88          SubFillStructure& subFill = fillStr["dofHand"];
89          int DOF  = subFill.numDOFs;
90          int eNN  = subFill.eNN;
91          int nDim = subFill.nDim;
92          vector<double>& nborCoords = subFill.nborCoords;
93          double *bf  = subFill.nborBFs();
94          double *dbf_l = subFill.nborBFsGrads();
95
96          ttl:: wrapper<double,2> Bk(fillStr.Bk(0).data(), eNN, DOF);
97
98          double Ip[4], xu[3], xv[3];
99          SurfLagrParam::MetricTensor(Ip, xu, xv, eNN, nborCoords.data(), dbf_l);
100
101         // Compute jacobian
102         auto bTangentRef = subFill.tangentsBoundaryRef();
103         double bTangent[3]={};
104         for (int n = 0; n < nDim; n++)
105         bTangent[n] = bTangentRef[0] * xu[n] + bTangentRef[1]*xv[n];
106         double jac = Math::Norm3D(bTangent);
107
108         // Fill RHS
109         for (int i = 0; i < eNN; i++)
110         fillStr.Bk(0)[i*DOF+1] += jac*bf[i]*traction * thickness;
111 }
```

7 Results

In this section we present our solution. Figures 3 and 3 show the displacement contour in deformed configuration.

References

[1] Junuthula Narasimha Reddy. *An introduction to the finite element method*, volume 3. McGraw-Hill New York, 2005.

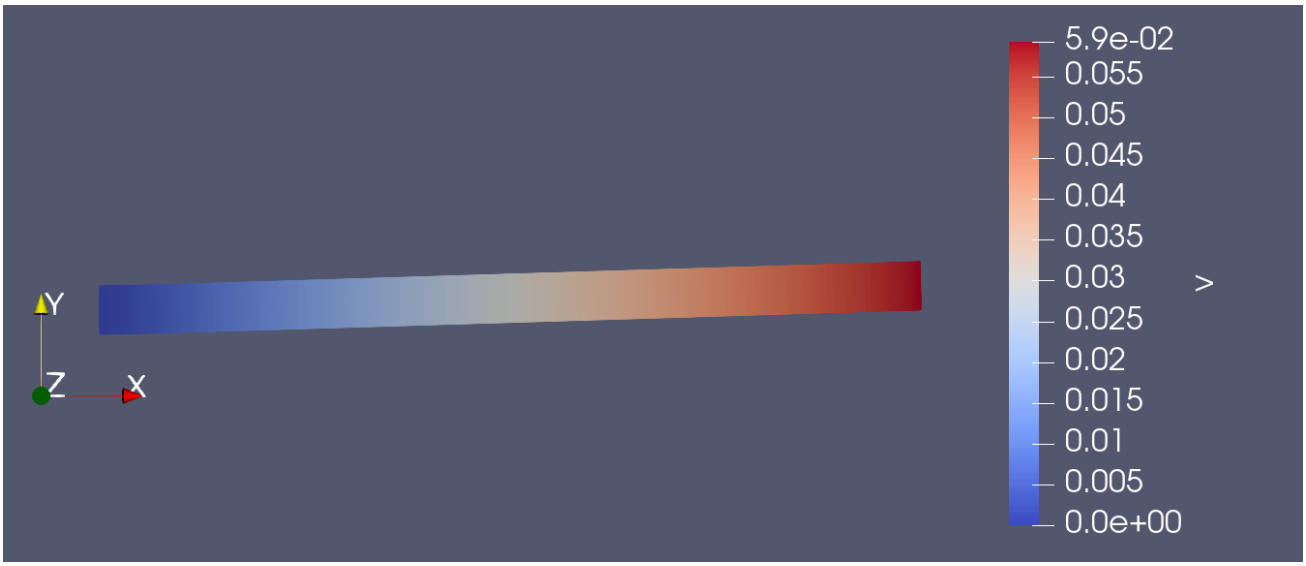

Figure 3: Displacement in y-direction.

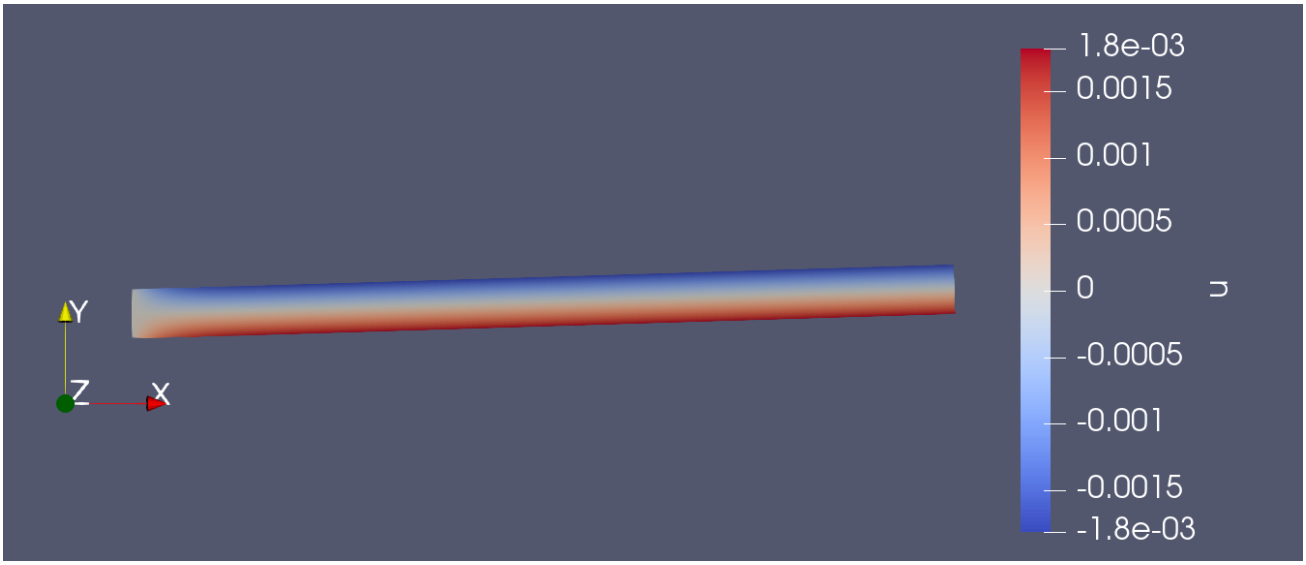

Figure 4: Displacement in x-direction.