CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Civil Engineering
Department of mechanics
Thákurova 7
166 29 Praha 6

Issue No. 1
Revision No. 0
Date 30.12.2023
Number of pages: 17

User manual

StableTrussOpt-MATLAB: Program of 3D truss structure optimization with global stability constraints in MATLAB.

MATĚJ LEPŠ
ŠIMON GLANC

Contents

StableTrussOpt-MATLAB is a program of 3D truss structure optimization with global stability constraints in MATLAB. For structural analysis, the frame elements are used to capture true 3D behavior. Then the problem is solved as a sizing optimization task formulated as a SDP program with polynomial constraints.

The software is available at `https://github.com/lepsmate/StableTrussOpt-MATLAB`. It is provided under LGPL-2.1 license.

Pre-requisites

Software needed to run the following functions:

- MATLAB (tested on MATLAB R2021a)
- YALMIP toolbox for Matlab (`https://yalmip.github.io/`)
- SGLIB toolbox for Matlab (`https://web.mat.bham.ac.uk/kocvara/penlab/`)

Acknowledgment

Financial support for this work was provided by the Technology Agency of the Czech Republic (TAČR), project number FW04020163 (Extension of the HiStruct platform by optimisation of global stability and analysis of design combinations).

Introduction

StableTrussOpt-MATLAB software is focused on the optimization of truss structures with regard to global stability. Although there are mathematical approaches that enable simultaneous analysis of stability and optimization (SAND), we will separately describe both topics in the introduction part and in the description of the conducted work. First, we will clarify the complexity of stability calculation compared to classical linear analysis. Then, we will describe the nonlinearity of structural optimization including conditions for global stability.

0.1 Functions

In this paragraph, the functions of StableTrussOpt-MATLAB software are briefly described. Following file folder tree is considered

```
Matlab
  framesFEM
    beamVectorXFn.m
    codeNumbersFn.m
    criticalLoadFn.m
    deformationGraphFn.m
    discretizationBeamsFn.m
    endForcesFn.m
    geometricMatrixFnV2.m
    sectionToElementFn.m
    sortValuesVectorFn.m
    stiffnessStabMatrixFn.m
    stiffnessMatrixFn.m
    transformationMatrixFn.m
    XYtoBeamsFn.m
    XYtoElementFn.m
  towerStabTorii_ex0_opt1.m
  towerStabTorii_ex0_stab1.m
```

Torii_ex0_stab1.m

The file provides critical load scale factor calculations using linear stability, plots images of the deformed structure, and displays the critical load scale factor, normal force and stress values from the reference load.

Important parts that can be appropriately changed are following:

Lines 13-30 of the Torii_ex0_stab1.m file

```
13 %% Sections
14
15     nSections = 6 ;
16
17     r0 = 10*10^-3 ;
18
19     sections.A = (15*pi*(ones(nSections,1)*r0).^2)/64 ;
20
21     for j = 1:nSections
22         sections.Iy(j,1) = sections.A(j)^2*113/(60*pi)   ;
23     end
24
25     sections.Iz = sections.Iy;
26     sections.Ix = sections.Iy*2;
27     sections.E  = ones(nSections,1) * 210*10^9;
28     sections.v  = ones(nSections,1) * 0.3;
29
30     ndisc = 8;
```

- Line 15: Number of cross-sections in the model.
- Line 17-28: Computation of cross-sectional characteristics for individual beams.
- Line 30: Definition of the number of FEM elements per beam

Lines 32-36 of the Torii_ex0_stab1.m file

```
32 %% Nodes
33     nodes.x = [0;2;0;2;4];
34     nodes.y = [0;0;0;0;0];
35     nodes.z = [0;0;2;2;2];
36     nodes.nnodes = numel(nodes.x);
```

- Line 33-35: Definition of node coordinates .
- Line 36: Number of nodes in the model.

Lines 38-52 of the Torii_ex0_stab1.m file

```
38 %% Supports
39     kinematic.x.nodes = [1;3];
40     kinematic.y.nodes = [1;3];
41     kinematic.z.nodes = [1;3];
42     kinematic.rx.nodes = [];
43     kinematic.ry.nodes = [];
44     kinematic.rz.nodes = [1;3];
45
46     nodes.dofs = true(nodes.nnodes,6);
47     nodes.dofs(kinematic.x.nodes,1) = false;
48     nodes.dofs(kinematic.y.nodes,2) = false;
49     nodes.dofs(kinematic.z.nodes,3) = false;
50     nodes.dofs(kinematic.rx.nodes,4) = false;
51     nodes.dofs(kinematic.ry.nodes,5) = false;
52     nodes.dofs(kinematic.rz.nodes,6) = false;
```

- Line 38-44: Nodes indices with restricted displacements/rotations.
- Line 46-52: Degrees of freedom according to boundary conditions.

Lines 54-60 of the Torii_ex0_stab1.m file

```
54 %% Beams
55     beams.nodesHead = [1;3;3;2;4;2];
56     beams.nodesEnd  = [2;2;4;4;5;5];
57     beams.nbeams = numel(beams.nodesHead);
58
59     beams.disc      = ones(beams.nbeams,1)*ndisc;
60     beams.sections  = 1:nSections;
```

- Line 55: Index of head nodes of beams.
- Line 56: Index of end nodes of beams.
- Line 57: Number of beams in the model.
- Line 59-60: In this step, the discretization and section index on the beams are set.

Lines 62-73 of the Torii_ex0_stab1.m file

```
62 %% Ground structure
63     plot3([nodes.x(beams.nodesHead) nodes.x(beams.nodesEnd)]', ...
64          [nodes.y(beams.nodesHead) nodes.y(beams.nodesEnd)]', ...
65          [nodes.z(beams.nodesHead) nodes.z(beams.nodesEnd)]', ...
66          'k','LineWidth',3);
67     hold on;
68     scatter3(nodes.x, nodes.y, nodes.z, 'black', 'filled', 'o');
69     axis equal;
70     xlim([min(nodes.x)-0.1,max(nodes.x)+0.1]);
71     ylim([min(nodes.y)-0.1,max(nodes.y)+0.1]);
72     zlim([min(nodes.z)-0.1,max(nodes.z)+0.1]);
73     grid on
```

- Line 63-73: These lines plot a graph showing the structure with nodes and beams.

Lines 75-88 of the Torii_ex0_stab1.m file

```
75 %% Loads
76     loads.y.nodes = [];
77     loads.y.value = [];
78     loads.x.nodes = [];
79     loads.x.value = [];
80     loads.z.nodes = [5];
81     loads.z.value = [-1000];
82
83     loads.rx.nodes = [];
84     loads.rx.value = [];
85     loads.ry.nodes = [];
86     loads.ry.value = [];
87     loads.rz.nodes = [];
88     loads.rz.value = [];
```

- Line 76-88: In this section load input is defined. *nodes* is the index of the node and *value* is the magnitude of the force for the specified load. In the same way the load moments are defined

Lines 90-95 of the Torii_ex0_stab1.m file

```
90 %% Force vector
91     forceVector = sparse([loads.x.nodes*6-5; loads.y.nodes*6-4; loads.z.nodes*6-3;...
92                           loads.rx.nodes*3-2; loads.ry.nodes*3-1; loads.rz.nodes*3],...
93                           1,[loads.x.value; loads.y.value; loads.z.value;loads.rx.value;...
94                           loads.ry.value; loads.rz.value],nodes.nnodes*6, 1);
95     f = forceVector(reshape(reshape(nodes.dofs.',[],1).', 1, [])');
```

- Line 91-95: The force vector for finite element method is defined.

Lines 96-101 of the Torii_ex0_stab1.m file

```
96      %% FEM
97      nodes.ndofs = sum(sum(nodes.dofs));
98
99      beams.vectorX = beamVertexFn(beams,nodes);
100     beams.codeNumbers = codeNumbersFn(beams,nodes);
101     beams.XY = XYtoBeamsFn(beams);
```

- Line 97: Number of unknown degrees of freedom.
- Line 99-101: In these lines, the direction vector is calculated, the code numbers from the degrees of freedom are plotted, and the vector in the XY plane is constructed.
- Line 103-106: In this part, the beams are divided into finite elements, assigned the cross-section, the vector in the XY plane and the new number of degrees of freedom is calculated.

Lines 108-115 of the Torii_ex0_stab1.m file

```
108 % Linear analysis
109     endForces.global = sparse(elements.ndofs,1);
110     endForces.global(1:max(max(beams.codeNumbers))) = f;
111     transformationMatrix = transformationMatrixFn(elements);
112     stiffnesMatrix = stiffnessMatrixStabFn(elements,transformationMatrix);
113
114     [endForces.local, displ] = endForcesFn(stiffnesMatrix,endForces,...
115                                 transformationMatrix,elements);
```

- Line 109-110: Create a force vector for the FEM elements
- Line 111: Preparation of the transformation matrix and calculating the lengths of the FEM elements contained in the structure.
- Line 112: Creation of local and global stiffness matrices that are contained in the structure.
- Line 114: Solution of the linear static analysis equation $\mathbf{K_e u} = \mathbf{f}$, which leads to the calculation of local end forces and element displacements.

Lines 117-124 of the Torii_ex0_stab1.m file

```
117 % Stability
118     geometricMatrix = geometricMatrixFnV2(elements,transformationMatrix,endForces);
119
120     volume = sum(elements.sections.A .* transformationMatrix.lengths);
121
122     Results = criticalLoadFn(stiffnesMatrix.global,geometricMatrix.global,100);
123
124     [sortedValues,sortedVectors]= sortValuesVectorFn(Results.values,Results.vectors);
```

- Line 118: Creation of geometric matrix $\mathbf{K}_\sigma$ with found end forces and element lengths.
- Line 120: Calculation of the volume of the structure, which is not important for stability.
- Line 122-124: Solving the eigenvalue problem, finding the critical load factors and ordering the positive eigenvalues from lowest to highest.

Lines 126-148 of the Torii_ex0_stab1.m file

```
126 %% Post-procesing
127     disp('= Section areas ========')
128     fprintf( 'A%d = %f\n', [(1:nSections)', sections.A]');
129     fprintf( 'Volume: %f\n', value(volume));
130 
131     disp( '= Five smallest critical loads =======')
132     fprintf( 'lambda%d = %f\n', [(1:numel(sortedValues(1:5)))', value(sortedValues(1:5))]');
133 
134     for i = 1:beams.nbeams
135         idR = 1:12;
136         idC = (i-1)*ndisc + 1 : i*ndisc;
137         endForces.beams{i} = endForces.local(idR, idC);
138         N(1,i)=endForces.beams{i}(7,1);
139     end
140 
141     N = full(N'/1000);
142     disp('= Normal forces ========')
143     fprintf('N%d = %f kN \n', [(1:numel(N))', value(N)]');
144     sigma = N./value(sections.A)/1000;
145     disp('= Normal stresses ======')
146     fprintf('sigma%d = %f MPa \n', [(1:numel(sigma))', value(sigma)]');
147 % Graph of deformed structure
148     graphValueId = 1;
149     graphScale = 3;
150     graph = deformationGraphFn(nodes,beams,sortedVectors(:,graphValueId),graphScale);
```

- Line 127-146: The critical load scale factors, normal forces and stresses on the beams are displayed.
- Line 148-150: A graph of the deformed structure corresponding to the coefficient of the lowest critical load scale factor is displayed. If you want to display a different value, you can change the index, or if you want to change the scale, this is also possible.

Torii_ex0_opt1.m

The procedure used in this file is almost identical to the file Torii_ex0_stab1.m, therefore only the parts different to this file are listed here.

Lines 16-27 of the Torii_ex0_opt1.m file

```
16 %% Sections
17 
18     nSections = 6 ; % number of cross-sections
19 
20     r0 = 10*10^-3
21 
22     startingA = (15*pi*(ones(nSections,1)*r0).^2)/64 ; % cross section area
23 
24     outA = sdpvar(nSections,1);
25     assign(outA, startingA);
26 
27     sections.A = outA ;
28     for j = 1:nSections
29         sections.Iy(j,1) = outA(j)^2*113/(60*pi)  ;
30     end
31 
32     sections.Iz = sections.Iy;
33     sections.Ix = sections.Iy*2;
34     sections.E  = ones(nSections,1) * 210*10^9;
35     sections.v  = ones(nSections,1) * 0.3;
```

- Line 20: Initial radius on beams

- Line 22-35: Define the radius as a *sdpvar* variable (from YALMIP library) and calculate the cross-sectional characteristics for each beam. *sdpvar* variable overloads the original one by its symbolic representation, but the object itself can contain a real value.

Lines 118-128 of the `Torii_ex0_opt1.m` file

```
118 % Linear analysis
119     endForces.global = sparse(elements.ndofs,1);
120     endForces.global(1:max(max(beams.codeNumbers))) = f;
121     transformationMatrix = transformationMatrixFn(elements);
122     stiffnesMatrix = stiffnessMatrixFn(elements,transformationMatrix);
123 
124     assign(outA, startingA);
125     realMatrix.global = value(stiffnesMatrix.global) ;
126     for i=1:size(stiffnesMatrix.local,2)
127         realMatrix.local{i} = value(stiffnesMatrix.local{i}) ;
128     end
```

- Line 122: Creation of local and global stiffness matrices with *sdpvar* variables that are contained in the structure.
- Line 124: Assign the *sdpvar* variables with starting radius.
- Line 125-128: Since the *stiffnesMatrix* is dependent on *outA*, the Line 124 fills the current values of areas into *outA* and Lines 125-128 recomputes current real values of *stiffnesMatrix* and stores these numbers in *realMatrix*.

Lines 132-144 of the `Torii_ex0_opt1.m` file

```
132 %% Optimization
133     geometricMatrix = geometricMatrixFnV2(elements,transformationMatrix,endForces);
134 
135     nonnegativeCrossSections = outA>=0.00001;
136 
137     Lambda = 10.0 ;
138     LMIconstraint = stiffnesMatrix.global+Lambda*geometricMatrix.global >= 0;
139 
140     volume = sum(elements.sections.A .* transformationMatrix.lengths);
141 
142     ops = sdpsettings('solver','penlab','verbose',1,'debug',1);
143     optimize([nonnegativeCrossSections, LMIconstraint], ...
144              volume, ops);
```

- Line 133: Creation of geometric matrix $\mathbf{K}_\sigma$ with found end forces and element lengths.
- Line 135: Setting of minimal (non-zero) areas $A_i \geq A_{min}, \forall i$
- Line 137: Setting of target value of critical multiplier $\bar{\lambda}$
- Line 138: SDP condition $\left(\mathbf{K_e} + \bar{\lambda}\mathbf{K_g}\right) \succeq \mathbf{0}$
- Line 140: Calculate the volume of the structure.
- Line 142: Select the optimization method, solver and settings.
- Line 143: Optimize with a defined objective function and constraints.

Lines 146-154 of the Torii_ex0_opt1.m file

```
146 %% Post-procesing
147     fprintf( 'Optimized objective: %f\n', value(volume));
148     disp( '= Section areas ========')
149     fprintf( 'A%d = %f\n', [(1:numel(outA))', value(outA)]');
150 
151     realMatrix.global = value(stiffnesMatrix.global) ;
152     Results = criticalLoadFn(realMatrix.global,geometricMatrix.global,100);
153 
154     [sortedValues,sortedVectors]= sortValuesVectorFn(Results.values,Results.vectors);
```

- Line 147-149: The optimized objective and optimized cross-section areas are displayed.
- Line 151-154: Solving the eigenvalue problem, finding the critical load factors and ordering the positive eigenvalues from lowest to highest.

framesFEM/

function [vector] = beamVectorXFn (beams, nodes)

InputVar:
`beams.nbeams` (number of beams), `beams.nodesHead` (indexes of initial nodes), `beams.nodesEnd` (indexes of ending nodes), `nodes.x` (x node coordinates), `nodes.y` (y node coordinates), `nodes.z` (z node coordinates)
OutVar:
`vector` (directional vector for individual beams).

function [codes] = codeNumbersFn (beams, nodes)

InputVar:
`beams.nbeams` (number of beams), `beams.nodesHead` (indexes of initial nodes), `beams.nodesEnd` (indexes of ending nodes), `nodes.dofs` (degrees of freedom)
OutVar:
`codes` (code numbers of beams).

function [Results] = criticalLoadFn (stiffnesMatrix, geometricMatrix, n)

InputVar:
`beams.nbeams` (number of beams), `beams.nodesHead` (indexes of initial nodes), `beams.nodesEnd` (indexes of ending nodes), `nodes.dofs` (degrees of freedom)
OutVar:
`Results.values` (critical load scale factor), `Results.vectors` (buckling shape vector)

graph [graph] = deformationGraphFn (nodes, beams, eigenVector, scale)

InputVar:
`beams.nbeams` (number of beams), `beams.nodesHead` (indexes of initial nodes), `beams.nodesEnd` (indexes of ending nodes), `beams.disc` (number of elements per beam),

`nodes.x` (x node coordinates), `nodes.y` (y node coordinates), `nodes.z` (z node coordinates), `nodes.dofs` (degrees of freedom), `eigenVector` (eigenmode vector), `scale` (deformation scale parameter)
OutVar:
`graph` (buckling shape of the structure in graph).

function [elements] = discretizationBeamsFn (beams, nodes)

InputVar:
`beams.nodesHead` (indexes of initial nodes), `beams.nodesEnd` (indexes of ending nodes), `beams.disc` (number of elements per beam), `beams.codeNumbers` (code numbers of beams), `beams.vectorX` (directional vector of beams), `beams.nbeams` (number of beams), `nodes.dofs` (degrees of freedom)
OutVar:
`elements` (elements with code numbers, directional vector and number of elements).

function [localEndForces, displacements] = endForcesFn (stiffnesMatrix, endForces, transformationMatrix, elements)

InputVar:
`stiffnesMatrix.global` (global stiffness matrix), `stiffnesMatrix.local` (local stiffness matrix), `endForces.global` (global loads at nodes), `transformationMatrix.matrices` (transformation matrices for individual elements), `elements.codeNumbers` (code numbers of elements), `elements.vectorX` (directional vector of elements), `elements.nelement` (number of elements), `elements.ndofs` (number of unknown DoFs)
OutVar:
`localEndForces` (local end forces on elements), `displacements.local` (local displacements on elements).

function [geometricMatrix] = geometricMatrixFnV2 (elements, transformationMatrix, endForces)

InputVar:
`elements.codeNumbers` (code numbers of elements), `elements.vectorX` (directional vector for individual elements), `elements.nelement` (number of elements), `elements.ndofs` (number of unknown DoFs), `elements.sections.E` (Young's modulus of the element), `elements.sections.v` (Poisson's ratio of the element)
OutVar:
`geometricMatrix.local` (local matrixes of initial stresses), `geometricMatrix.global` (global matrix of initial stresses).

function [elemSections] = sectionToElementFn (sections, beams)

InputVar:
`beams.sections` (section id from variable sections), `beams.disc` (number of elements per

beam), `sections.A` (cross-sectional area), `sections.Iy` (moment of inertia about Y axis), `sections.Iz` (moment of inertia about Z axis), `sections.Ix` (torsional moment of inertia), `sections.E` (Young's modulus of the material), `sections.v` (Poisson's ratio of the material)
OutVar:
`elemSections.A` (cross-sectional area), `elemSections.Iy` (moment of inertia about Y axis), `elemSections.Iz` (moment of inertia about Z axis), `elemSections.Ix` (moment of inertia about X axis), `elemSections.E` (Young's modulus of the material), `elemSections.v` (Poisson's ratio of the material),

function [sorted_values, sorted_vectors] = sortValuesVectorFn (values, vectors)

InputVar:
`values` (eigenvalues), `vectors` (eigenvectors)
OutVar:
`sorted_values` (sorted positive eigenvalues), `sorted_vectors` (sorted eigenvectors according to the eigenvalues)

function [stiffnesMatrix] = stiffnessMatrixStabFn (elements, transformationMatrix)

InputVar:
`elements.sections.A` (cross-sectional area), `elements.sections.Iy` (moment of inertia about Y axis), `elements.sections.Iz` (moment of inertia about Z axis), `elements.sections.Ix` (torsional moment of inertia), `elements.sections.v` (Poisson's ratio), `elements.sections.E` (Young's modulus), `elements.nelement` (number of elements), `elements.ndofs` (number of unknown DoFs), `elements.codeNumbers` (element code numbers), `transformationMatrix.matrices` (transformation matrices for individual elements), `transformationMatrix.lengths` (lengths of elements)
OutVar:
`stiffnesMatrix.local` (local stiffness matrices), `stiffnesMatrix.global` (global stiffness matrix)

function [stiffnesMatrix] = stiffnessMatrixFn (elements, transformationMatrix)

InputVar:
`elements.sections.A` (cross-sectional area), `elements.sections.Iy` (moment of inertia about Y axis), `elements.sections.Iz` (moment of inertia about Z axis), `elements.sections.Ix` (torsional moment of inertia), `elements.sections.v` (Poisson's ratio), `elements.sections.E` (Young's modulus), `elements.nelement` (number of elements), `elements.ndofs` (number of unknown DoFs), `elements.codeNumbers` (element code numbers), `transformationMatrix.matrices` (transformation matrices for individual elements), `transformationMatrix.lengths` (lengths of elements)
OutVar:

`stiffnesMatrix.local` (local stiffness matrices for optimization),
`stiffnesMatrix.global` (global stiffness matrix for optimization)

function [transformationMatrix] = transformationMatrixFn (elements)

InputVar:
`elements.vectorX` (directional vector of individual elements), `elements.XY` (vector determining the XY plane of the element), `elements.nelement` (number of elements)
OutVar:
`transformationMatrix.matrices` (transformation matrices for individual elements),
`transformationMatrix.lengths` (lengths of elements)

function [XY] = XYtoBeamsFn (beams)

InputVar:
`beams.vectorX` (directional vector of individual beams), `beams.nbeams` (number of beams)
OutVar:
`XY` (vector determining the XY plane of the beams)

function [XY] = XYtoElementFn (beams)

InputVar:
`beams.XY` (vector determining the XY plane of the beams), `beams.nbeams` (number of beams),
`beams.disc` (number of elements per beam)
OutVar:
`XY` (vector determining the XY plane of the elements)

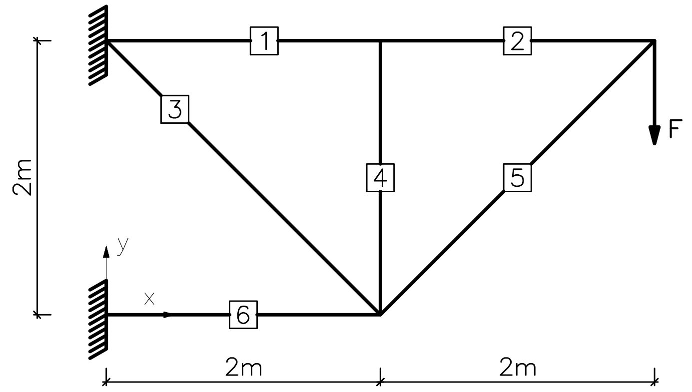

Figure 1: Example Setup.

0.2 Example of usage - stability

The studied example is taken from the introduction of the article [1]. The cross-section of the structure in Figure 1 is a tubular section with an outer radius of $r_o = 4$ cm and an inner radius of $r_i = 3.5$ cm. The material parameters are the modulus of elasticity $E = 210$ GPa and Poisson's ratio $\nu = 0.3$. Each member is divided into 16 frame elements. Running the file `Torii_ex0_stab1.m` with the implementation of this example yields $P_{crit} = 83.22$ kN. For comparison and validation, results from the OOFEM software package (`http://oofem.org`) are provided. The plot of the corresponding first mode shape is shown in Figure 2.

λ_i	**OOFEM**	**StableTrussOpt-MATLAB**	**[%]**
1	83.21	83.22	0.01
2	288.91	288.95	0.01
3	344.47	344.52	0.01
4	471.79	471.93	0.03
5	521.11	521.18	0.01
6	814.43	814.95	0.06
7	942.06	943.18	0.12
8	1208.39	1209.99	0.13
9	1276.09	1278.20	0.16
10	1729.81	1735.48	0.33

Table 1: Comparison of critical load factors for the first 10 eigenmodes.

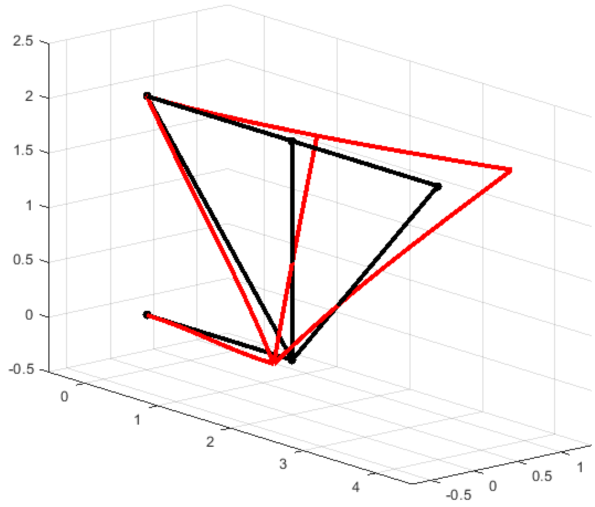

Figure 2: First eigenmode shape, showing deflection out of plane.

0.3 Example of usage - optimization

By running the file `Torii_ex0_opt1.m`, the optimization of the example is performed, yielding the targeted $P_{crit} = 10.00$ kN. For comparison and validation, the results from the OOFEM software package (`http://oofem.org`) are provided in the following table.

λ_i	**OOFEM**	**StableTrussOpt-MATLAB**	**[%]**
1	10.00	10.00	0.04
2	37.53	37.50	0.06
3	41.95	41.95	0.02
4	76.01	76.02	0.01
5	84.42	84.34	0.09
6	119.62	119.38	0.20
7	143.41	143.45	0.03
8	229.50	216.41	5.70
9	283.64	229.80	18.98
10	314.50	283.08	9.99

Table 2: Comparison of critical load factors

REFERENCES

[1] André Torii, Rafael Lopez, and Leandro Miguel. Modeling of global and local stability in optimization of truss-like structures using frame elements. *Structural and Multidisciplinary Optimization*, 51, 12 2014.