Topology Optimization

Department of Mechanical Engineering

I.I.T. Bombay

ME 308

Industrial Engineering and Operation Research

Under Guidance of

Prof. Avinash Bhardwaj

Prof. Makarand S Kulkarni

Members of Team Project Slayer

Tanyut Sharma (190100128)

Sumit Sureka (19D100024)

Shrihari Ravi Wattamwar (19D100021)

Aarush Billayia (190100002)

Aman Kumar (190100012)

Acknowledgement

We acknowledge with gratitude our Professors for giving us the opportunity to work on this project and gave the valuable guidance for preparing this project.

We would like to thank those people who directly or indirectly help us to enhance our practical knowledge in the field of Industrial Engineering and Optimization research and express our sincere gratitude to all those who share valuable thoughts with us.

This being our first effort, the possibilities of errors and omissions in its contents and presentation cannot be completely ruled out. We shall, however, be grateful to our colleagues and other readers for their suggestions for its improvement.

Introduction

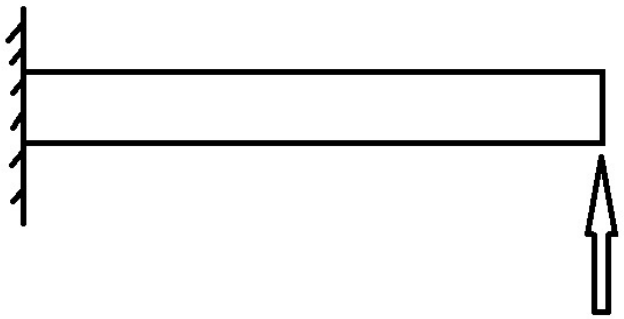
Topology optimization is a mathematical method that optimizes material layout within a given design space, for a given set of loads, boundary conditions and constraints with the goal of maximizing the performance of the system. It is different from shape optimization and sizing optimization in the sense that the design can attain any number of holes within the design space, instead of dealing with predefined configurations. There are a number of software available in the market, which can easily topologically optimize the problems. And manually or mathematically, we can even calculate the same by carrying out some calculations for stresses, strains and structural and solid mechanics. So, with the help of AMPL and python (for plotting), the problem solving has been initiated.

Problem Statement

We wish to divide a given region into two parts, one with material, one without, such that a set of equations governing the physics of the region are satisfied and a provided objective function is minimised. However, before this can be implemented, we need to solve the simpler problem of finding the deformation of a uniform region for some given loads and fixed surfaces.

Initially we have chosen to study a horizontal cantilever beam loaded vertically at its tip. We devised two methods of finding its deformation:

1) Strain – strain relationships
2) Grid of springs



Saint Venant-Kirchhoff Model:

The simplest hyper elastic material model is the Saint Venant–Kirchhoff model which is just an extension of the geometrically linear elastic material model to the geometrically nonlinear regime. It is thus a very good representation of the material characteristics in both the linear and nonlinear regime.

Parameters:

- λ & μ are elastic coefficients.
- L is Length, H is the height of the domain.
- n, m represents no. of finite elements in the domain

Decision variables:

2-D Strain tensor –

$$\varepsilon = \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{yx} & \varepsilon_{yy} \end{bmatrix}$$

- $\boldsymbol{e_{11,i,j}} = \varepsilon_{xx}$ (tensile strain in x-direction)
- $\boldsymbol{e_{12,i,j}} = \varepsilon_{xy}$ (shear strain in x-direction)

  $= \varepsilon_{yx}$ (shear strain in y-direction)
- $\boldsymbol{e_{22,i,j}} = \varepsilon_{yy}$ (tensile strain in y-direction)

  $\forall\, i \in (1, n-1)\ \&\ j \in (1, m-1)$

2-D Stress tensor –

$$\sigma = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy} \end{bmatrix}$$

- $\boldsymbol{s_{11,i,j}} = \sigma_{xx}$ (tensile stress in x direction)
- $\boldsymbol{s_{12,i,j}} = \sigma_{xy}$ (shear stress in x direction)

  $= \sigma_{yx}$ (shear stress in y direction)
- $\boldsymbol{s_{22,i,j}} = \sigma_{yy}$ (tensile stress in y direction)

  $\forall\, i \in (1, n-1)\ \&\ j \in (1, m-1)$

Deflections at each point:

$\boldsymbol{u_{x,i,j}}$ **and** $\boldsymbol{u_{y,i,j}}$ $\quad \forall\, i \in (1, n)\ \&\ j \in (1, m)$

Constraints:

Relations between the decision variables:

- $e_{11,i,j} = 0.5\ (u_{x,i+1,j} + u_{x,i+1,j+1} - u_{x,i,j} - u_{x,i,j+1})\ /\ dx$
- $e_{22,i,j} = 0.5\ (u_{y,i,j+1} + u_{y,i+1,j+1} - u_{y,i,j} - u_{y,i+1,j})\ /\ dy$
- $e_{12,i,j} = 0.25\ (u_{x,i,j+1} + u_{x,i+1,j+1} - u_{x,i,j} - u_{x,i+1,j})\ /\ dy + 0.25\ (u_{y,i+1,j} + u_{y,i+1,j+1} - u_{y,i,j} - u_{y,i,j+1})\ /\ dx$
- $s_{11,i,j} = \lambda\ (e_{11,i,j} + e_{22,i,j}) + 2\ \mu\ e_{11,i,j}$
- $s_{12,i,j} = 2\ \mu\ e_{12,i,j}$
- $s_{22,i,j} = \lambda\ (e_{11,i,j} + e_{22,i,j}) + 2\ \mu\ e_{22,i,j}$

$\forall\ i \in (1, n-1)\ \&\ j \in (1, m-1)$

Fixing the left edge:

- $u_{x,1,j} = 0;\ \forall\ j \in (1, m)$
- $u_{y,1,j} = 0;\ \forall\ j \in (1, m)$

Neumann boundary condition (stress):

- $s_{11,n-1,m/2} = 0$
- $s_{12,n-1,m/2} = 10$

These are the given boundary conditions, which we had applied to our model.

Dirichlet boundary condition (deflection):

- $u_{x,n,m/2} = 0$
- $u_{y,n,m/2} = 0.2$

These are the given boundary conditions, which we had applied to our model.

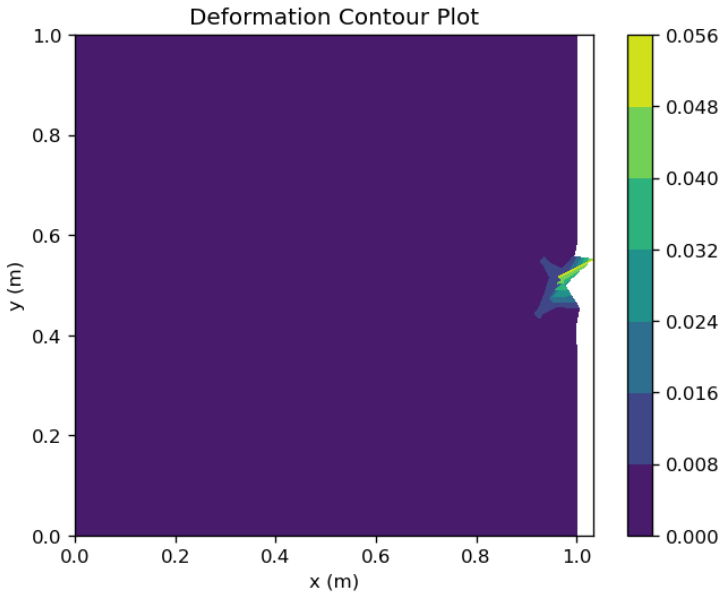
Objective function:

$$\boldsymbol{W} = \sum_{j=1}^{m-1} \sum_{i=1}^{n-1} s_{11,i,j}\ e_{11,i,j} + 2\ s_{12,i,j}\ \ e_{12,i,j} + s_{22,i,j}\ e_{22,i,j}$$

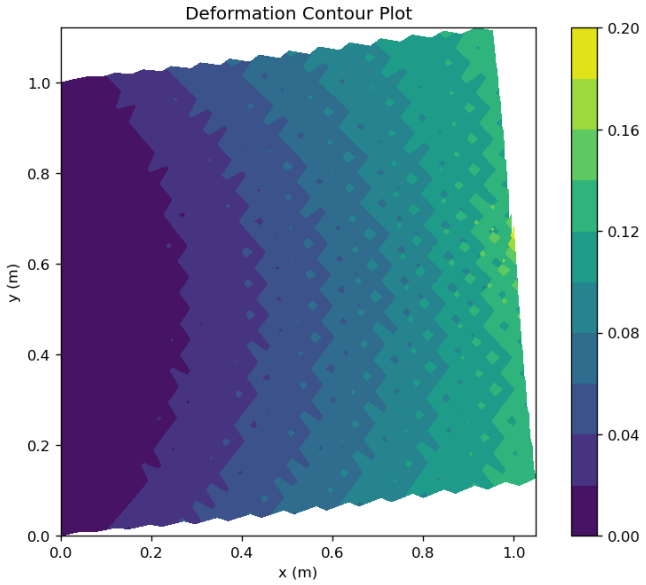
Minimize (W) for obtaining the most stable deformed state of the beam.

Results:

1) With the Neumann boundary conditions, this formulation does not work at all. Only the points near the application of force have any displacement at all. The results are clearly wrong as seen here:



2) With the Dirichlet boundary conditions, the results are much better, though still incorrect. The displacement looks reasonable, and gives very close results to analytical methods, but is incorrect as it gives a very 'wavy' result:



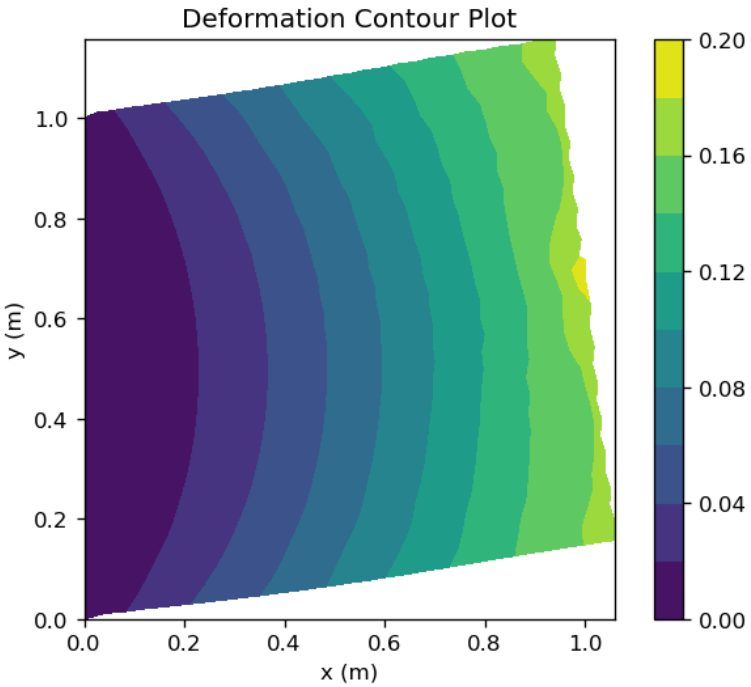
One can see the waviness best on the top and bottom edges. Performing topology optimisation is impossible for such inaccurate results.

Explanation for results:

We believe this result can be explained by a much simpler problem; 1D springs connected at N nodes, moving only along their length. Two boundary conditions are needed for this problem, let one of them be keeping the left end of the springs fixed. The remaining one can be a force applied on the right end, or a displacement specified at the same point.

1) If we apply a force (Neumann BC), then the minimum energy will be given by keeping all nodes in their original position, and only moving the leftmost node to provide the force. This is clearly an incorrect solution. We see analogous result in our 2D domain, when only points near the applied force have any significant displacement.
2) For a given displacement (Dirichlet BC), the minimum energy of the springs will require the correct displacements of the springs, this can be easily proved mathematically. This reflects our 2D results which are much better when a displacement is given, however our results are not perfectly correct. This may be due to the fact that the top and bottom edges, and parts of the right edge are stress-free surfaces. The minimum energy condition should have internally applied this condition, but this is seen to not be the case. Manually forcing the surfaces to be stress-free runs into the problem discussed for the Neumann boundary conditions.

   Specifying the displacement at an additional point does alleviate the problem somewhat as the amount of free surfaces is reduced:



At this point, we believed that if displacements were specified at all points on the boundary, then the results would be correct. To test this, we devised a method the convert Neumann boundary conditions into Dirichlet ones. Our idea had two elements:

1) The domain would be extended into the free space (presumed vacuum) outside of the beam
2) The elastic coefficients would no longer be constant, but would vary to distinguish the beam from its surroundings

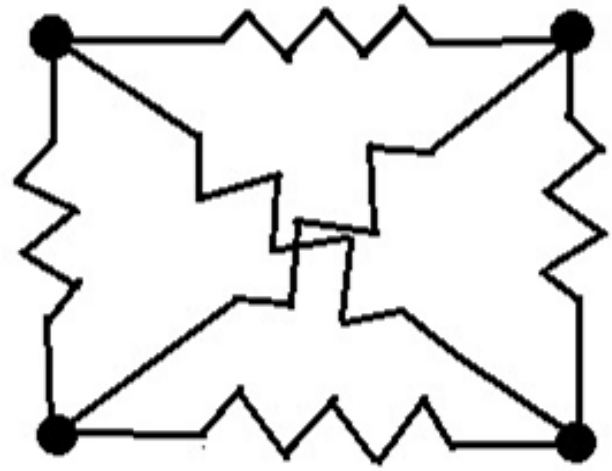
The premise was that we could specify displacements all along the outer boundary of the larger domain, and then the vacuum's negligible elastic coefficients would ensure that it could apply no force on the beam's surface, thus making them stress-free surfaces.

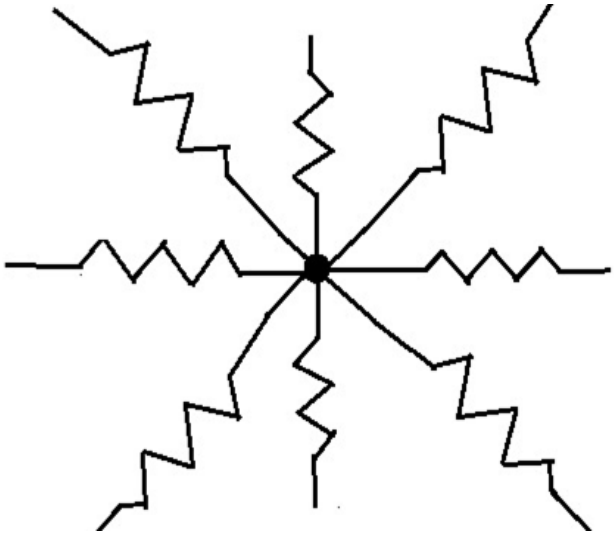
Initial results showed that this method suffered from the exact same errors as seen before, in light of this, we immediately rejected this method.

Using stress-strain relationships was not giving correct results, and optimizing the topology was impossible without accurate values. We had found the 1D arrangement of spring explanation to our difficulties very straightforward, so we believed the logical next step was to try to implement a 2D grid of springs.

Grid of Springs:

We consider a grid of nodes present on the region. The nodes are connected to each other through horizontal, vertical and diagonal springs. A maximum of 8 springs are connected to each node and each element of four adjoining nodes contains 6 total springs.





This is a highly non-linear model due to presence of quadratic and trigonometric components.

Decision variables:

Absolute position of each point: $u_{x,i,j}$ and $u_{y,i,j}$

Deformation in the springs: $l_{h,i,j}$, $l_{v,i,j}$, $l_{d1,i,j}$ and $l_{d2,i,j}$

Forces in the springs: $F_{h,i,j}$, $F_{v,i,j}$, $F_{d1,i,j}$ and $F_{d2,i,j}$

Net force at the nodes: $F_{hx,i,j}$, $F_{hy,i,j}$, $F_{vx,i,j}$, $F_{vy,i,j}$, $F_{d1x,i,j}$, $F_{d1y,i,j}$, $F_{d2x,i,j}$ and $F_{d2y,i,j}$

Parameters:

- L is Length, H is height of the domain.
- n, m represents no. of finite elements in the domain
- $k_{h,i,j}, k_{v,i,j}, k_{d1,i,j}, k_{d2,i,j}$ are spring constants
- dx = L/(n-1), dy = H/(m-1), $\mathrm{dl} = \sqrt{\mathrm{dx}^2 + \mathrm{dy}^2}$

Constraints:

Pythagorean Relations

- $\boldsymbol{l_{h,i,j}} = \sqrt{\left(\boldsymbol{u_{x,i,j}} - \boldsymbol{u_{x,i+1,j}}\right)^2 + \left(\boldsymbol{u_{y,i,j}} - \boldsymbol{u_{y,i+1,j}}\right)^2}$ $\forall\ i \in (1, n-1)\ \&\ j \in (1, m)$
- $\boldsymbol{l_{v,i,j}} = \sqrt{\left(\boldsymbol{u_{x,i,j}} - \boldsymbol{u_{x,i,j+1}}\right)^2 + \left(\boldsymbol{u_{y,i,j}} - \boldsymbol{u_{y,i,j+1}}\right)^2}$ $\forall\ i \in (1, n)\ \&\ j \in (1, m-1)$
- $\boldsymbol{l_{d1,i,j}} = \sqrt{\left(\boldsymbol{u_{x,i,j}} - \boldsymbol{u_{x,i+1,j+1}}\right)^2 + \left(\boldsymbol{u_{y,i,j}} - \boldsymbol{u_{y,i+1,j+1}}\right)^2}$ $\forall\ i \in (1, n-1)\ \&\ j \in (1, m-1)$
- $\boldsymbol{l_{d2,i,j}} = \sqrt{\left(\boldsymbol{u_{x,i+1,j}} - \boldsymbol{u_{x,i,j+1}}\right)^2 + \left(\boldsymbol{u_{y,i+1,j}} - \boldsymbol{u_{y,i,j+1}}\right)^2}$ $\forall\ i \in (1, n-1)\ \&\ j \in (1, m-1)$

F=K x for linear springs

- $\boldsymbol{F_{h,i,j}} = \boldsymbol{k_{h,i,j}}\left(\boldsymbol{l_{h,i,j}} - \boldsymbol{dx}\right)$ $\forall\ i \in (1, n-1)\ \&\ j \in (1, m)$
- $\boldsymbol{F_{v,i,j}} = \boldsymbol{k_{v,i,j}}\left(\boldsymbol{l_{v,i,j}} - \boldsymbol{dy}\right)$ $\forall\ i \in (1, n)\ \&\ j \in (1, m-1)$
- $\boldsymbol{F_{d1,i,j}} = \boldsymbol{k_{d1,i,j}}\left(\boldsymbol{l_{d1,i,j}} - \boldsymbol{dl}\right)$ $\forall\ i \in (1, n-1)\ \&\ j \in (1, m-1)$
- $\boldsymbol{F_{d2,i,j}} = \boldsymbol{k_{d2,i,j}}\left(\boldsymbol{l_{d2,i,j}} - \boldsymbol{dl}\right)$ $\forall\ i \in (1, n-1)\ \&\ j \in (1, m-1)$
- $\boldsymbol{F_{x,i,j}} = \boldsymbol{F_{hx,i,j}} + \boldsymbol{F_{vx,i,j}} + \boldsymbol{F_{d1x,i,j}} + \boldsymbol{F_{d2x,i,j}}$ $\forall\ i \in (1, n)\ \&\ j \in (1, m)$
- $\boldsymbol{F_{y,i,j}} = \boldsymbol{F_{hy,i,j}} + \boldsymbol{F_{vy,i,j}} + \boldsymbol{F_{d1y,i,j}} + \boldsymbol{F_{d2y,i,j}}$ $\forall\ i \in (1, n)\ \&\ j \in (1, m)$

Global components of Spring Forces

- $$F_{hx,i,j} = \begin{cases} F_{h,i,j}\frac{(u_{x,i+1,j}-u_{x,i,j})}{l_{h,i,j}} - F_{h,i-1,j}\frac{(u_{x,i,j}-u_{x,i-1,j})}{l_{h,i-1,j}}, & i \neq 1 \text{ and } i \neq n \\ F_{h,i,j}\frac{(u_{x,i+1,j}-u_{x,i,j})}{l_{h,i,j}}, & i = 1 \\ -F_{h,i-1,j}\frac{(u_{x,i,j}-u_{x,i-1,j})}{l_{h,i-1,j}}, & i = n \end{cases}$$

- $$F_{hy,i,j} = \begin{cases} F_{h,i,j}\frac{(u_{y,i+1,j}-u_{y,i,j})}{l_{h,i,j}} - F_{h,i-1,j}\frac{(u_{y,i,j}-u_{y,i-1,j})}{l_{h,i-1,j}}, & i \neq 1 \text{ and } i \neq n \\ F_{h,i,j}\frac{(u_{y,i+1,j}-u_{y,i,j})}{l_{h,i,j}}, & i = 1 \\ -F_{h,i-1,j}\frac{(u_{y,i,j}-u_{y,i-1,j})}{l_{h,i-1,j}}, & i = n \end{cases}$$

$\forall\ i \in (1, n)\ \&\ j \in (1, m)$

- $$F_{vx,i,j} = \begin{cases} F_{v,i,j}\frac{(u_{x,i,j+1}-u_{x,i,j})}{l_{v,i,j}} - F_{v,i,j-1}\frac{(u_{x,i,j}-u_{x,i,j-1})}{l_{v,i,j-1}}, & j \neq 1 \text{ and } j \neq m \\ F_{v,i,j}\frac{(u_{x,i,j+1}-u_{x,i,j})}{l_{v,i,j}}, & j = 1 \\ -F_{v,i,j-1}\frac{(u_{x,i,j}-u_{x,i,j-1})}{l_{v,i,j-1}}, & j = m \end{cases}$$

- $$F_{vy,i,j} = \begin{cases} F_{v,i,j}\frac{(u_{y,i,j+1}-u_{y,i,j})}{l_{v,i,j}} - F_{v,i,j-1}\frac{(u_{y,i,j}-u_{y,i,j-1})}{l_{v,i,j-1}}, & j \neq 1 \text{ and } j \neq m \\ F_{v,i,j}\frac{(u_{y,i,j+1}-u_{y,i,j})}{l_{v,i,j}}, & j = 1 \\ -F_{v,i,j-1}\frac{(u_{y,i,j}-u_{y,i,j-1})}{l_{v,i,j-1}}, & j = m \end{cases}$$

$\forall\ i \in (1, n)\ \&\ j \in (1, m)$

- $$F_{d1x,i,j} = \begin{cases} F_{d1,i,j}\frac{(u_{x,i+1,j+1}-u_{x,i,j})}{l_{d1,i,j}} - F_{d1,i-1,j-1}\frac{(u_{x,i,j}-u_{x,i-1,j-1})}{l_{d1,i-1,j-1}}, & i \neq 1, i \neq n,\ j \neq 1 \text{ and } j \neq m \\ F_{d1,i,j}\frac{(u_{x,i+1,j+1}-u_{x,i,j})}{l_{d1,i,j}}, & i = 1 \text{ or } j = 1 \\ -F_{d1,i-1,j-1}\frac{(u_{x,i,j}-u_{x,i-1,j-1})}{l_{d1,i-1,j-1}}, & i = n \text{ or } j = m \end{cases}$$

- $$F_{d1y,i,j} = \begin{cases} F_{d1,i,j}\frac{(u_{y,i+1,j+1}-u_{y,i,j})}{l_{d1,i,j}} - F_{d1,i-1,j-1}\frac{(u_{y,i,j}-u_{y,i-1,j-1})}{l_{d1,i-1,j-1}}, & i \neq 1, i \neq n,\ j \neq 1 \text{ and } j \neq m \\ F_{d1,i,j}\frac{(u_{y,i+1,j+1}-u_{y,i,j})}{l_{d1,i,j}}, & i = 1 \text{ or } j = 1 \\ -F_{d1,i-1,j-1}\frac{(u_{y,i,j}-u_{y,i-1,j-1})}{l_{d1,i-1,j-1}}, & i = n \text{ or } j = m \end{cases}$$

  ∀ *i* ϵ *(1, n)* & *j* ϵ *(1, m)*

- $$F_{d2x,i,j} = \begin{cases} F_{d2,i,j-1}\frac{(u_{x,i+1,j-1}-u_{x,i,j})}{l_{d2,i,j-1}} - F_{d2,i-1,j}\frac{(u_{x,i,j}-u_{x,i-1,j+1})}{l_{d2,i-1,j}}, & i \neq 1, i \neq n,\ j \neq 1 \text{ and } j \neq m \\ F_{d2,i,j-1}\frac{(u_{x,i+1,j-1}-u_{x,i,j})}{l_{d2,i,j-1}}, & i = 1 \text{ or } j = m \\ -F_{d2,i-1,j}\frac{(u_{x,i,j}-u_{x,i-1,j+1})}{l_{d2,i-1,j}}, & i = n \text{ or } j = 1 \end{cases}$$

- $$F_{d2y,i,j} = \begin{cases} F_{d2,i,j-1}\frac{(u_{y,i+1,j-1}-u_{y,i,j})}{l_{d2,i,j-1}} - F_{d2,i-1,j}\frac{(u_{y,i,j}-u_{y,i-1,j+1})}{l_{d2,i-1,j}}, & i \neq 1, i \neq n,\ j \neq 1 \text{ and } j \neq m \\ F_{d2,i,j-1}\frac{(u_{y,i+1,j-1}-u_{y,i,j})}{l_{d2,i,j-1}}, & i = 1 \text{ or } j = m \\ -F_{d2,i-1,j}\frac{(u_{y,i,j}-u_{y,i-1,j+1})}{l_{d2,i-1,j}}, & i = n \text{ or } j = 1 \end{cases}$$

  ∀ *i* ϵ *(1, n)* & *j* ϵ *(1, m)*

Fixing the left edge:

- $u_{x,\,1,\,j}$ = 0 ∀ j ϵ (1, m)
- $u_{y,\,1,\,j}$ = (j-1) dy ∀ j ϵ (1, m)

Deflection at the bottom-right corner:

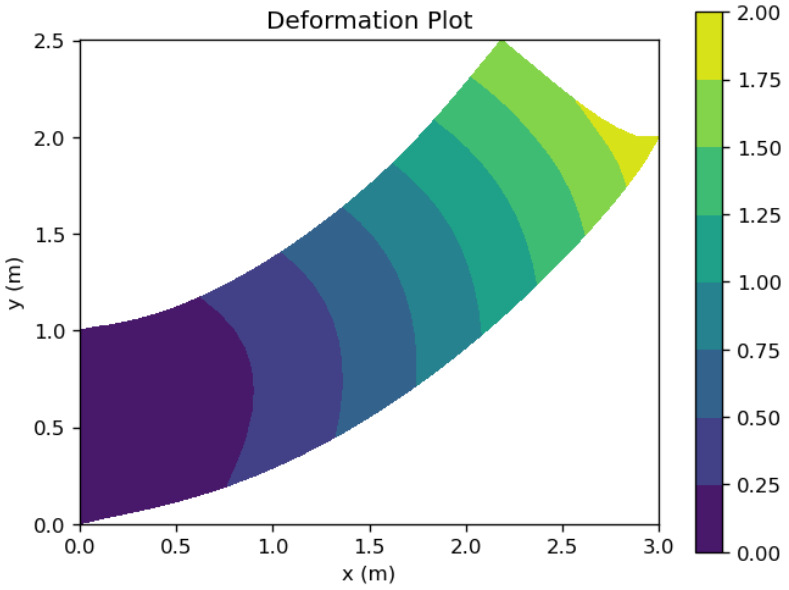
- $u_{x,\,n,\,1}$ = 0
- $u_{y,\,n,\,1}$ = 2

Net force zero at other nodes:

- $F_{x,\,i,\,j}$ = 0 ∀ i ϵ (2, n-1) & j ϵ (1, m)
- $F_{y,\,i,\,j}$ = 0 ∀ i ϵ (2, n-1) & j ϵ (1, m)
- $F_{x,\,n,\,j}$ = 0 ∀ j ϵ (2, m)
- $F_{y,\,n,\,j}$ = 0 ∀ j ϵ (2, m)

Results:

The obtained deflection looks very promising, and we have ascertained that it is indeed correct:



We now proceeded to implement topology optimization. However, there is a problem we saw immediately:

How do we define the presence or absence of material when only springs are present?

We worked on two approaches to solve this:

1) Isotropic method: Keep each spring constant variable and define presence of material as average of spring constants
2) Anisotropic method: Define a variable from 0 to 1 indicating presence or absence of material, and derive all spring constants from it

Topology Optimization:

NOTE: The following is in addition to the discussion for springs alone previously

Decision variables:

Spring Constants: $\boldsymbol{k_{h,i,j}}, \boldsymbol{k_{v,i,j}}, \boldsymbol{k_{d1,i,j}}$ **and** $\boldsymbol{k_{d2,i,j}}$

Normalised Material Density: $\boldsymbol{h_{i,j}}$

Parameters:

- *L* is Length, *H* is height of the domain.
- *n, m* represents no. of finite elements in the domain
- *dx = L/(n-1), dy = H/(m-1),* $dl = \sqrt{dx^2 + dy^2}$
- k_{min} and k_{max} – Constraint on Spring Stiffnesses
- Total Optimized Weight Constant, or Total Material Fraction *c* ($0 \leq c \leq 1$)

Constraints:

1) Isotropic Domain:

- $\boldsymbol{k_{h,i,j}} = \begin{cases} \boldsymbol{k_{min} + 0.5\,(k_{max} - k_{min})(h_{i,j-1} + h_{i,j}),} & \boldsymbol{j \neq 1 \text{ and } j \neq m} \\ \boldsymbol{k_{min} + 0.5\,(k_{max} - k_{min})h_{i,1},} & \boldsymbol{j = 1} \\ \boldsymbol{k_{min} + 0.5\,(k_{max} - k_{min})h_{i,m-1},} & \boldsymbol{j = m} \end{cases}$ ∀ *i* ϵ *(1, n-1) &* *j* ϵ *(1, m)*

- $\boldsymbol{k_{v,i,j}} = \begin{cases} \boldsymbol{k_{min} + 0.5\,(k_{max} - k_{min})(h_{i-1,j} + h_{i,j}),} & \boldsymbol{i \neq 1 \text{ and } i \neq n} \\ \boldsymbol{k_{min} + 0.5\,(k_{max} - k_{min})h_{1,j},} & \boldsymbol{i = 1} \\ \boldsymbol{k_{min} + 0.5\,(k_{max} - k_{min})h_{n-1,j},} & \boldsymbol{i = n} \end{cases}$ ∀ *i* ϵ *(1, n) &* *j* ϵ *(1, m-1)*

- $\boldsymbol{k_{d1,i,j} = k_{d2,i,j} = k_{min} + (k_{max} - k_{min})h_{i,j}}$ ∀ *i* ϵ *(1, n-1) &* *j* ϵ *(1, m-1)*

Normalised Material density:

- $0 \leq h_{i,j} \leq 1$ ∀ i ϵ (1, n-1) & j ϵ (1, m-1)

Limiting Total Material Mass:

- $\sum_{j=1}^{m-1} \sum_{i=1}^{n-1} h_{i,j} \leq c\,(m-1)\,(n-1)$

2) Anisotropic Domain:

- $\boldsymbol{h_{i,j} = 0.125\ (k_{h,i,j} + k_{v,i,j} + k_{h,i,j+1} + k_{v,i+1,j}) + 0.25\ (k_{d1,i,j} + k_{d2,i,j})}$

  ∀ *i* ϵ *(1, n-1)* & *j* ϵ *(1, m-1)*

Spring Stiffnesses:

- $k_{min} \leq k_{h,i,j} \leq k_{max}$ ∀ i ϵ (1, n-1) & j ϵ (1, m)
- $k_{min} \leq k_{v,i,j} \leq k_{max}$ ∀ i ϵ (1, n) & j ϵ (1, m-1)
- $k_{min} \leq k_{d1,i,j} \leq k_{max}$ ∀ i ϵ (1, n-1) & j ϵ (1, m-1)
- $k_{min} \leq k_{d2,i,j} \leq k_{max}$ ∀ i ϵ (1, n-1) & j ϵ (1, m-1)

Limiting Total Material Mass:

- $0.25\ (\sum_{j=1}^{m}\sum_{i=1}^{n-1}\frac{k_{h,i,j}}{m \text{ x } (n-1) \text{ x } k_{max}} + \sum_{j=1}^{m-1}\sum_{i=1}^{n}\frac{k_{v,i,j}}{(m-1) \text{ x } n \text{ x } k_{max}} + \sum_{j=1}^{m-1}\sum_{i=1}^{n-1}\frac{k_{d1,i,j}+k_{d2,i,j}}{(m-1) \text{ x } (n-1) \text{ x } k_{max}} \leq c$

Objective function:

Deflection is applied on the entire right edge– need to optimize for the deflection of the right edge (work done or energy stored is according to the right edge).

$\boldsymbol{W = \sum_{j=1}^{m} F_{y,n,j}}$

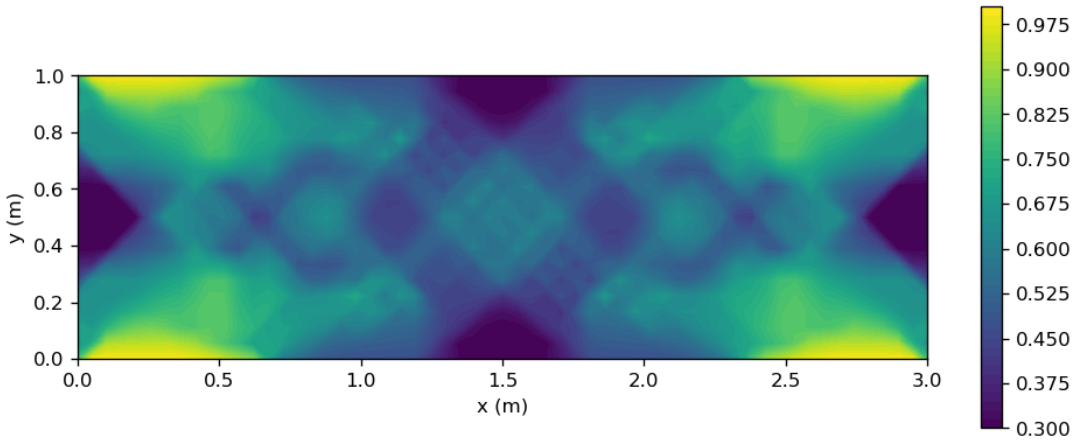
Maximize W (total vertical force on the right edge) for obtaining the optimized solution.

TOPOLOGY OPTIMIZATION RESULTS:

On solving the problem, we have the following results for our different cases and inputs. We found that the converged result depends heavily on the particular material properties and constraints chosen, much more than we expected.

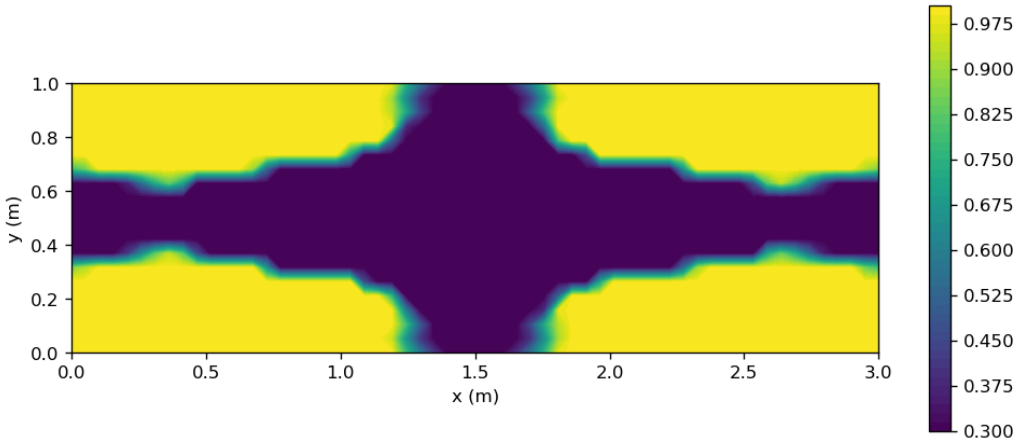
1) Isotropic method:

h:

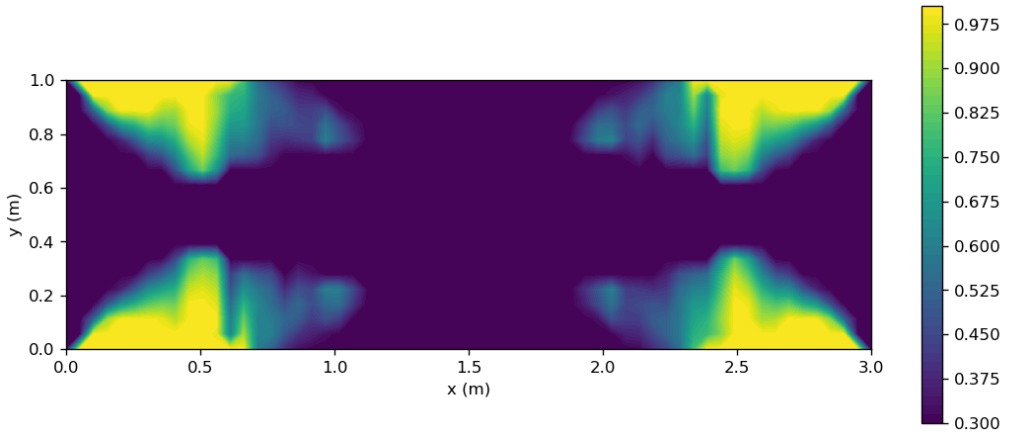


The distributions of particular springs are as follows-

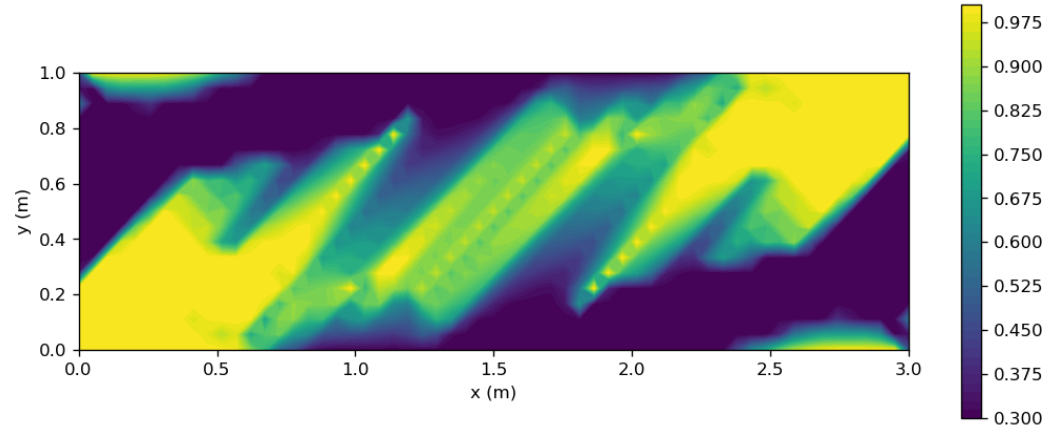
kx:

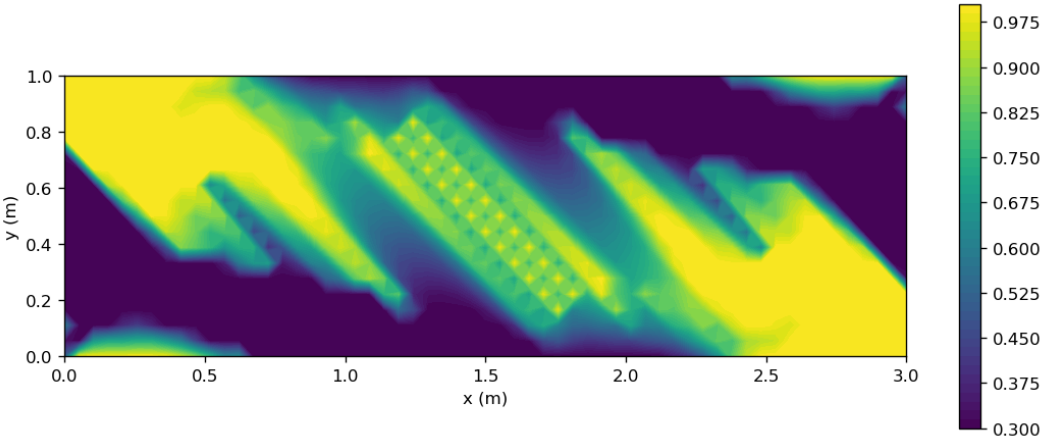


ky:



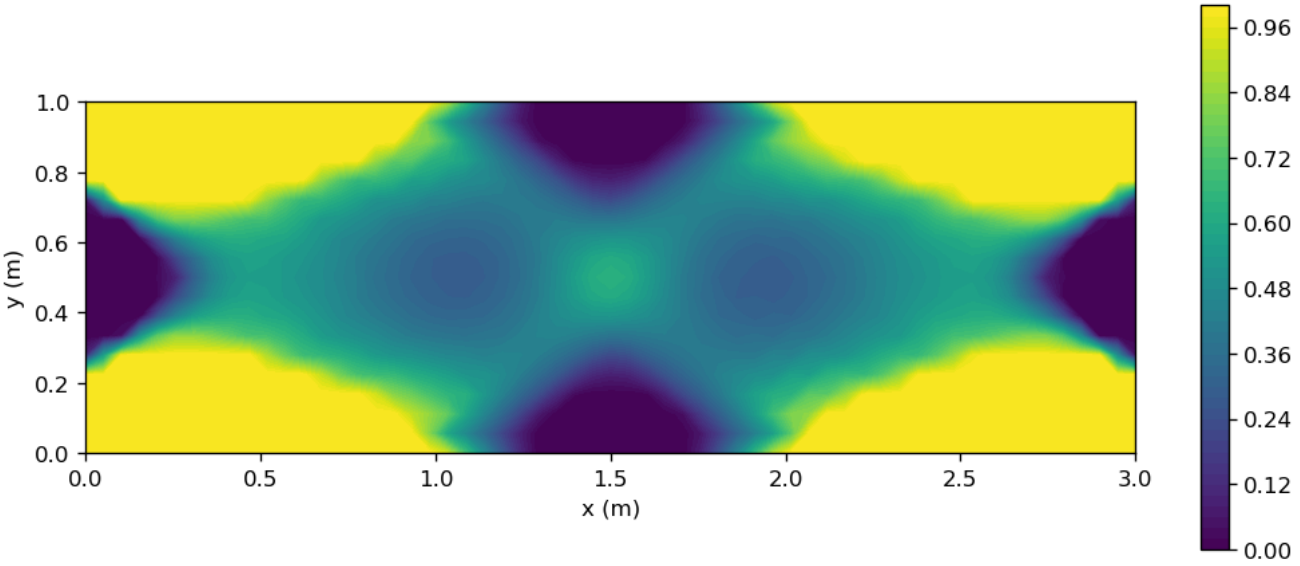
kd1 and kd2:





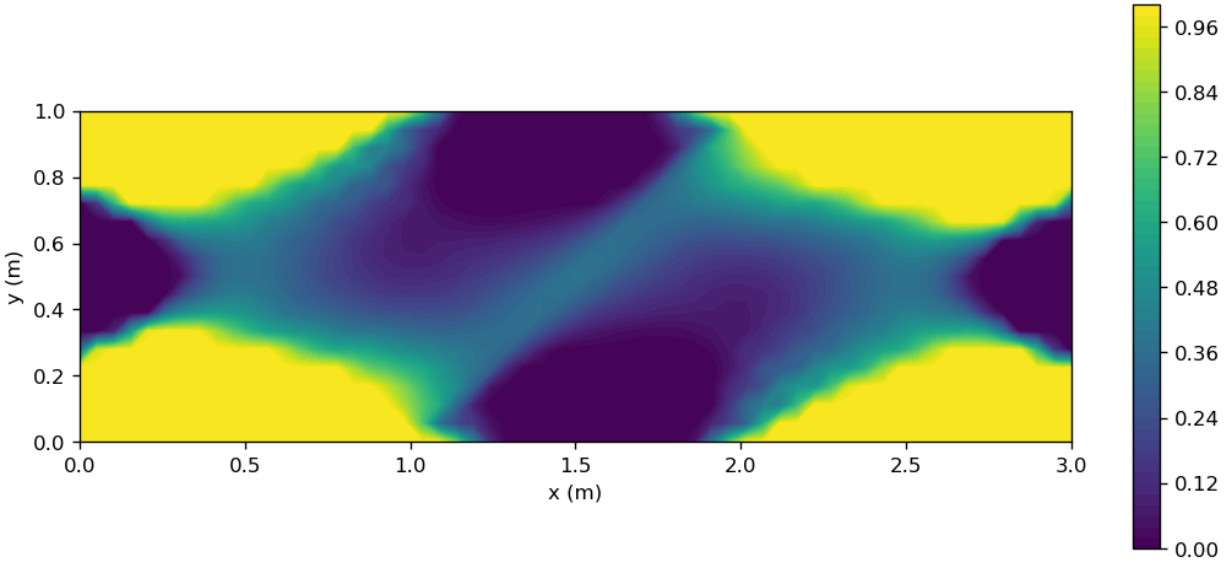
2) Anisotropic method:

h:



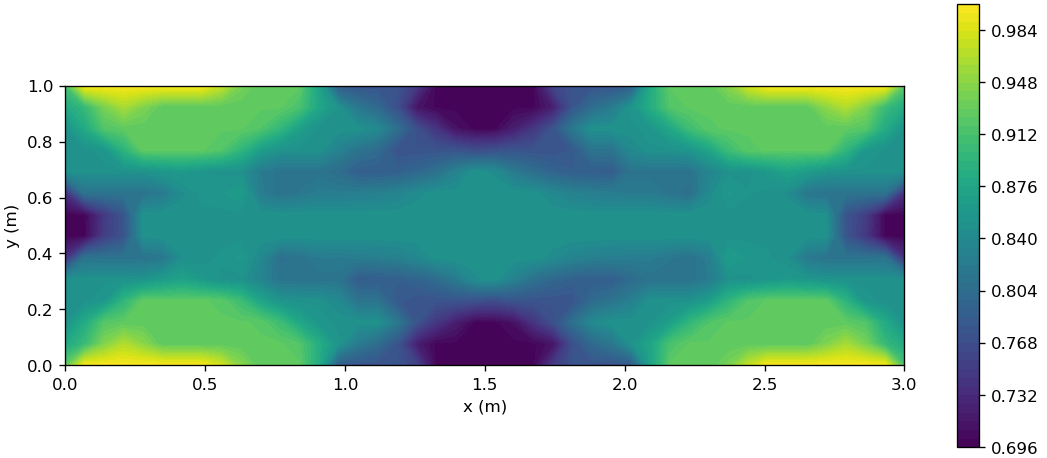
This structure appears to incorporate many of the elements seen in the isotropic results. Overall, this seems to be a smoother, better result.

Next, we have increased the deflection:

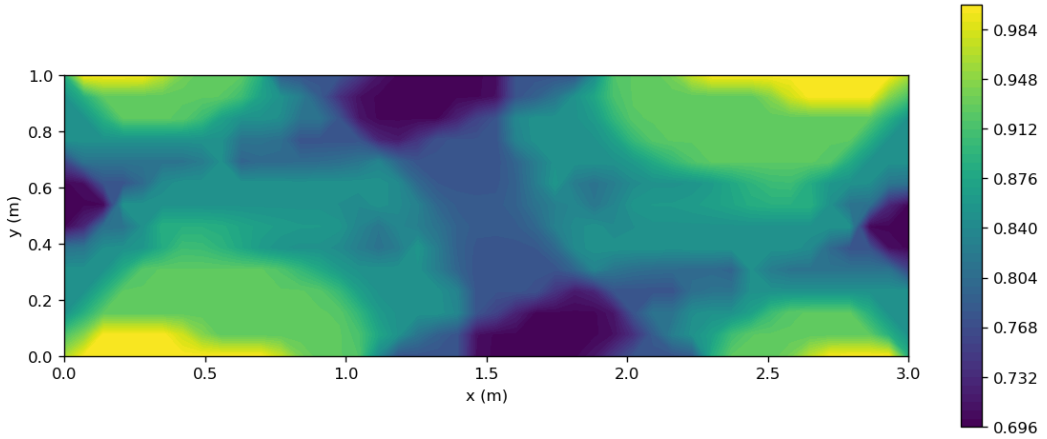


We can see that on increasing the deflection the result loses its symmetricity. This is because the springs behave linearly only for small deflections.

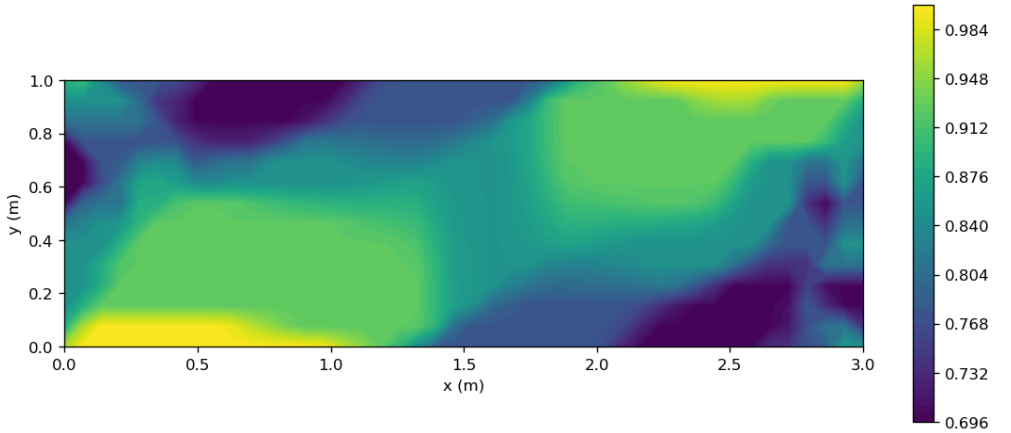
With the isotropic method:



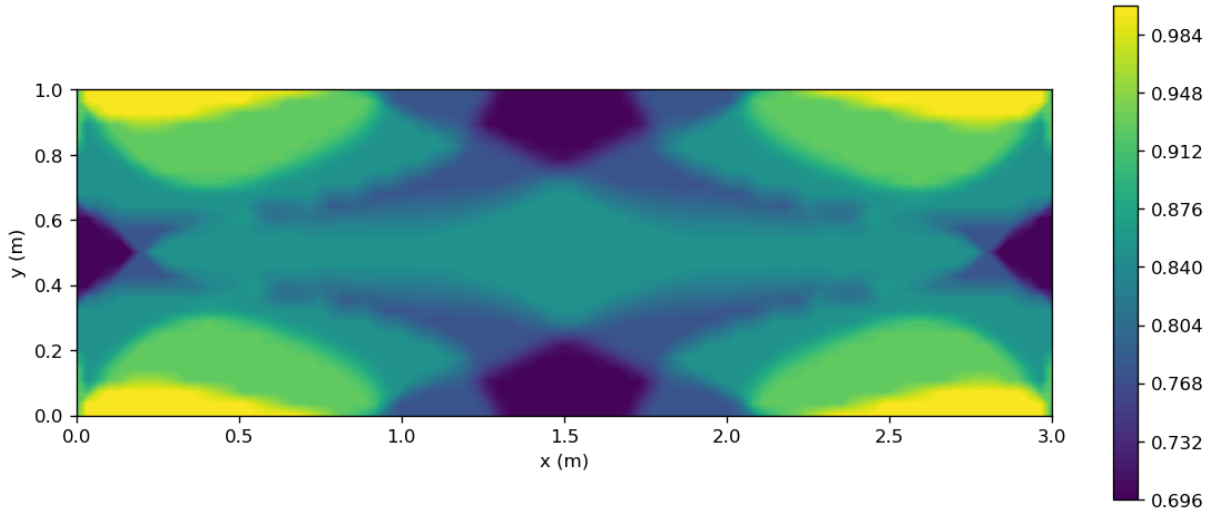
This is for the small deflection of the beam and we can clearly see the symmetricity again and for higher deflection, it is shown below.



The symmetry is again lost. We increased the deflection even further, and the remaining symmetry is lost as well:



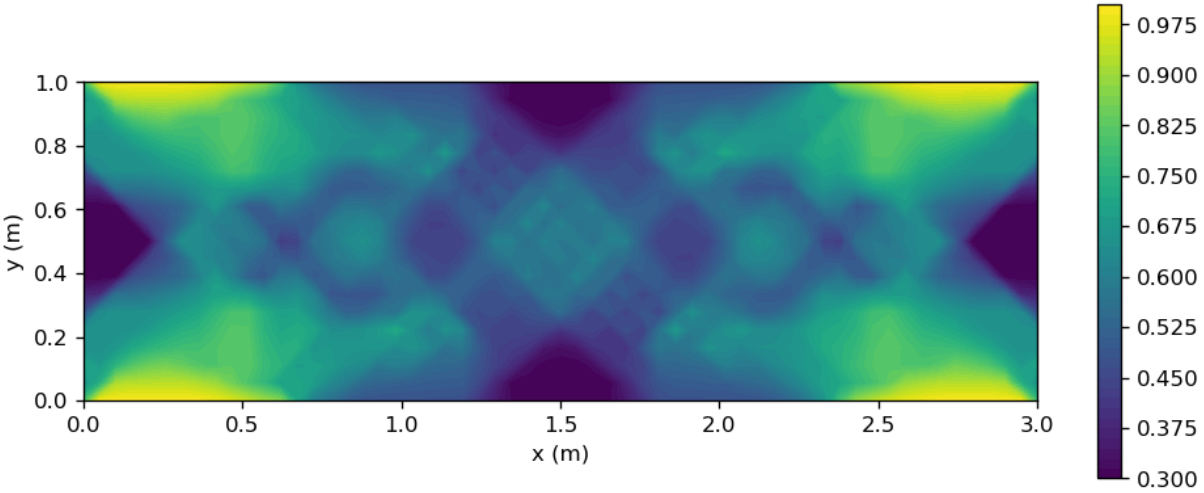
Then we thought about increasing the number of nodes (resolution of the beam structure). That has given the following result.



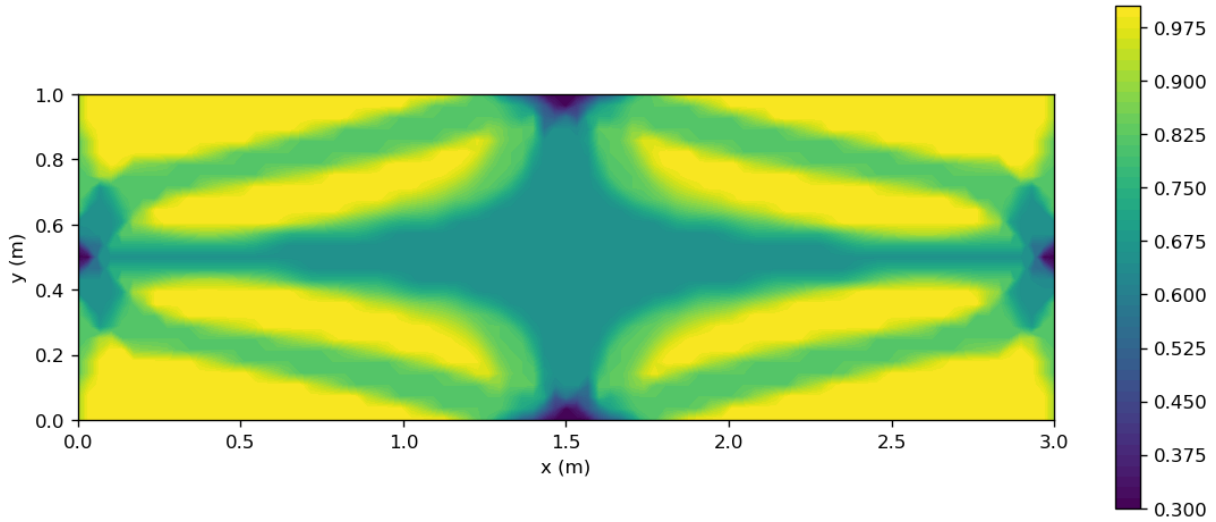
It can be seen clearly that the boundaries are more sharply defined than before.

Then it was thought, how about increasing the material fraction of the structure.

Low material fraction:

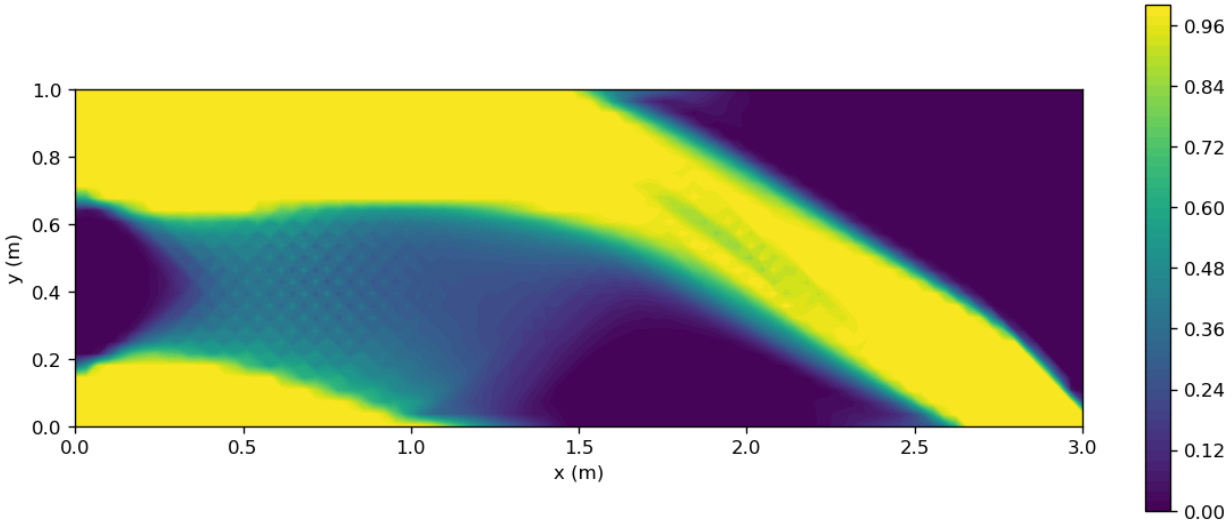


High material fraction:



The result is very different from previous ones. In all the following results, we have used the anisotropic method. We have never seen results that look like these before in our study of solid mechanics.

Next, instead of moving the entire right edge, we only move the bottom-right corner:



The results appear to be close to what we would expect.

This concludes the cantilever beam results. Next, we worked on bridge-like boundary conditions.

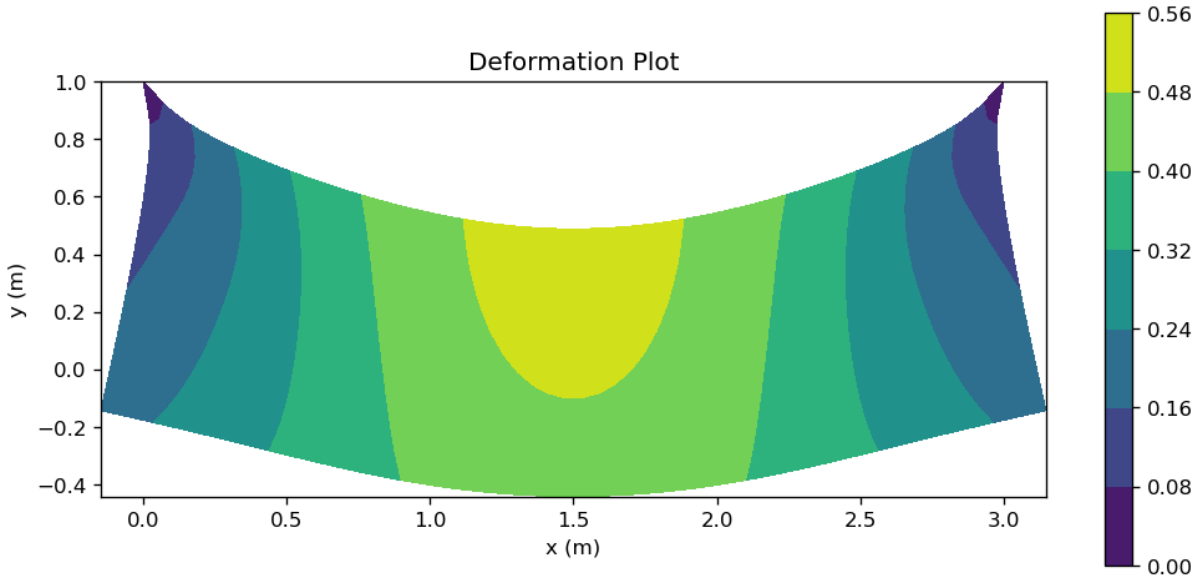
We have taken the same boundary condition which normally a bridge does in a real-world scenario. Nodes at the bottom left and right corner are fixed. And the nodes of the bottom of the bridge have been given a vertically downward force.

Objective function:

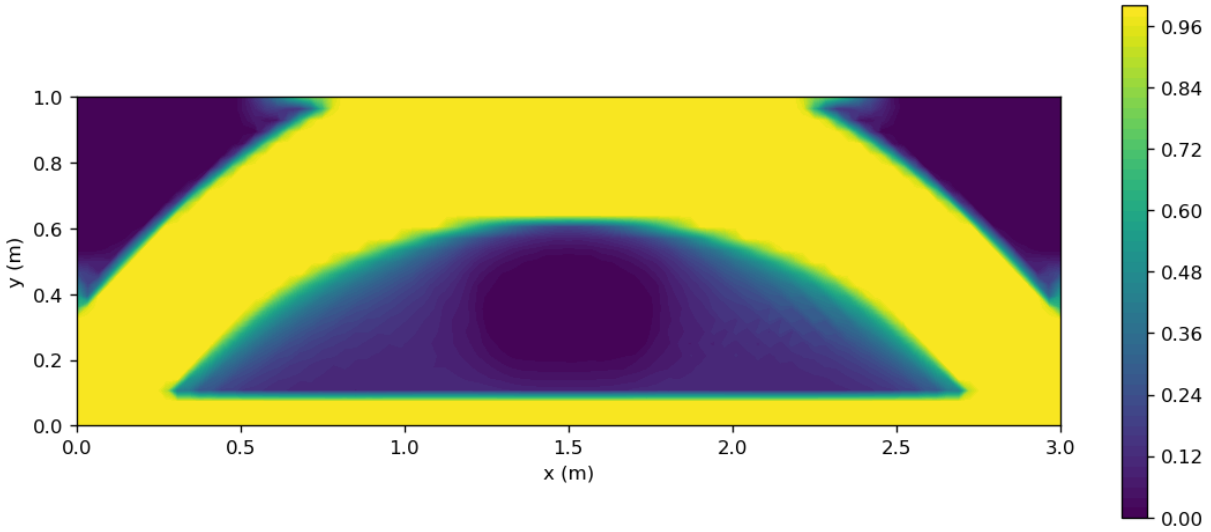
$$W = \sum_{i=2}^{n-1} u_{y,i,1}$$

Minimize W (total vertical deflection of the bottom edge) for obtaining the optimized solution.

Without optimization, the deflection is:



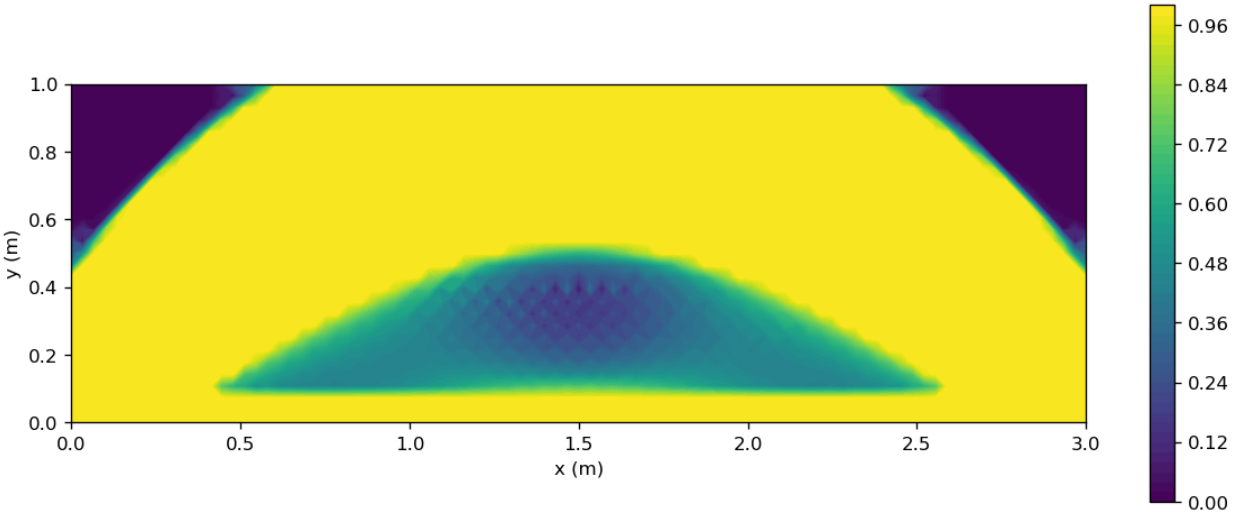
After optimizing, the result is:



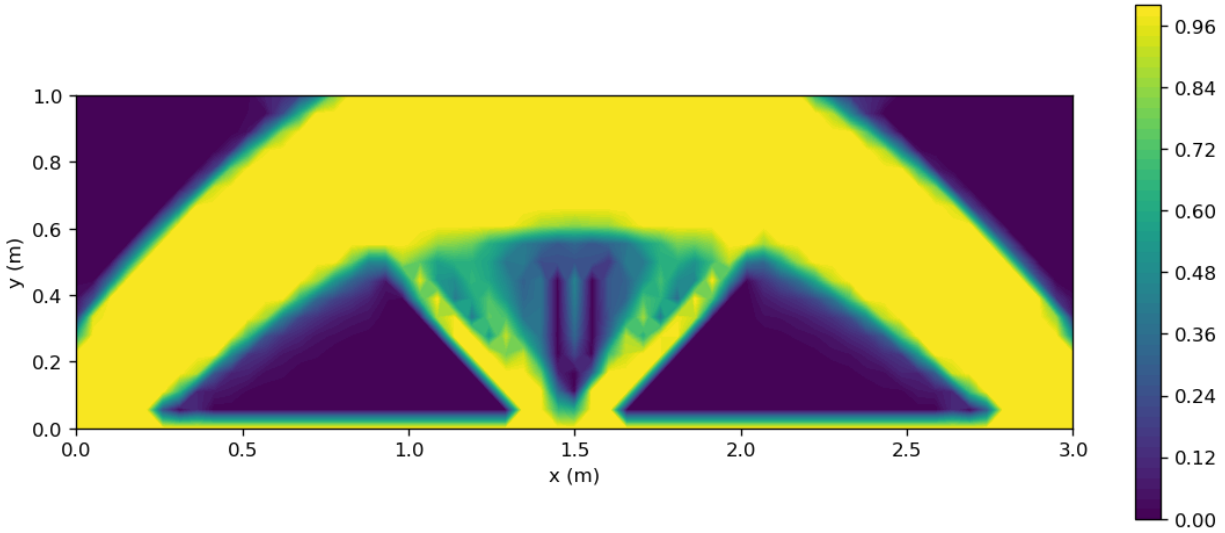
Here we have forced a layer on the bottom of the bridge (where the force was applied) to be always present. The solver does not converge without it.

The arc-like result corroborates our common-sense understanding.

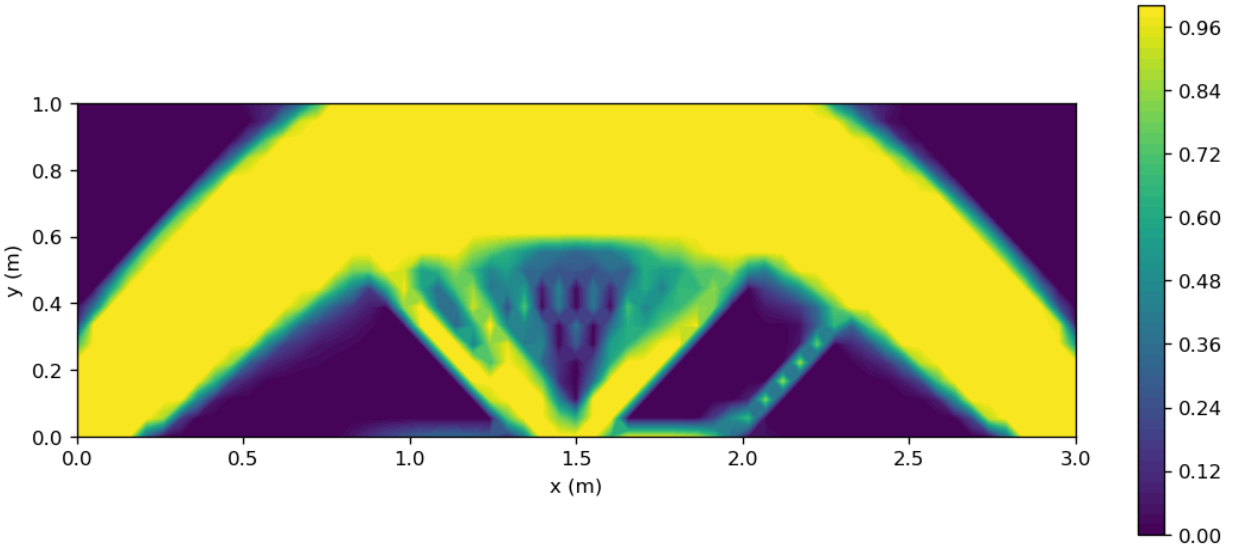
Then after this, we have increased the material fraction of the structure, then the result is:



Till now, for the bridge we have been optimizing the energy for all the nodes of the bottom of the bridge, but now we thought to optimize the energy of the centre of the bottom (max deflection is there), keeping all other things the same. So, the result for that is:



The deflection is kept small, but it had been increased slowly then something interesting came forward:



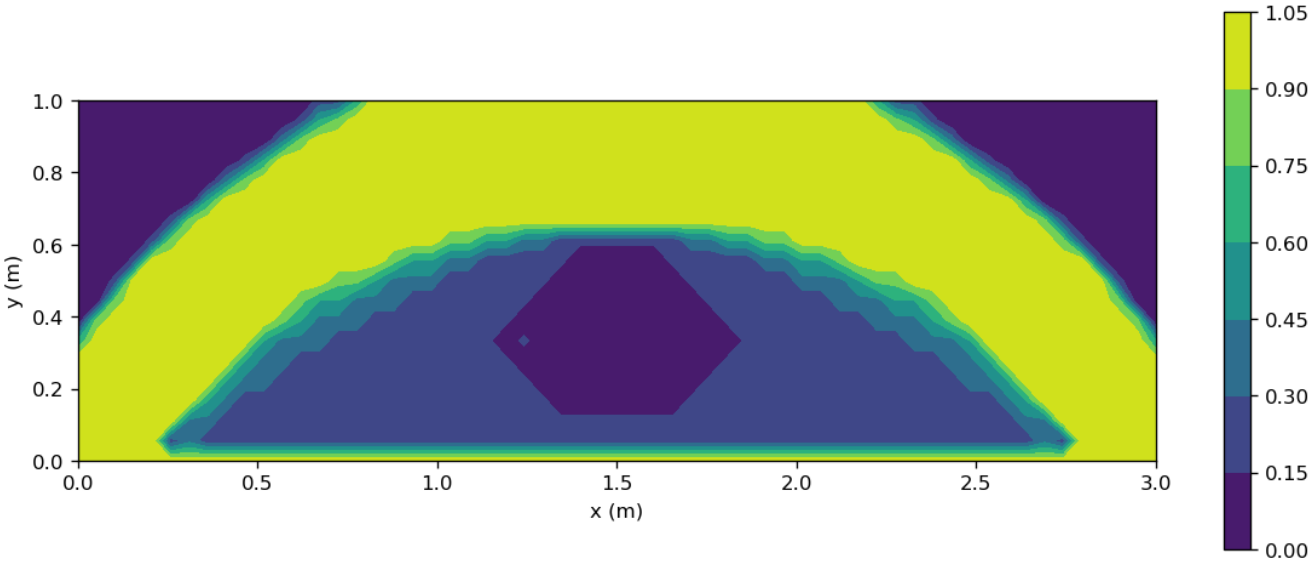
On increasing the deflection, it was found that there is buckling in the structure. That's why the symmetricity was lost. This is similar to the loss of symmetry seen for the cantilever beam.

Another interesting observation is that in the above case the material going from bottom-centre to the arc is making 45 degrees, this is because of the diagonal springs between the nodes.

Uncertainty analysis

In the real world, we do not have springs available at all possible sizes. In our results, we assumed a continuous distribution, which would never be real. Now we attempted to make our results more realistic with the following steps:

1) The springs can only take 6 discrete values between kmin and kmax:



2) The stiffness of each individual spring will have a uniform distribution around the discretised value. This was implemented in AMPL itself using the following function:

```
Uniform(-kerr, kerr)
```

3) The random seed in AMPL is changed before each run by:

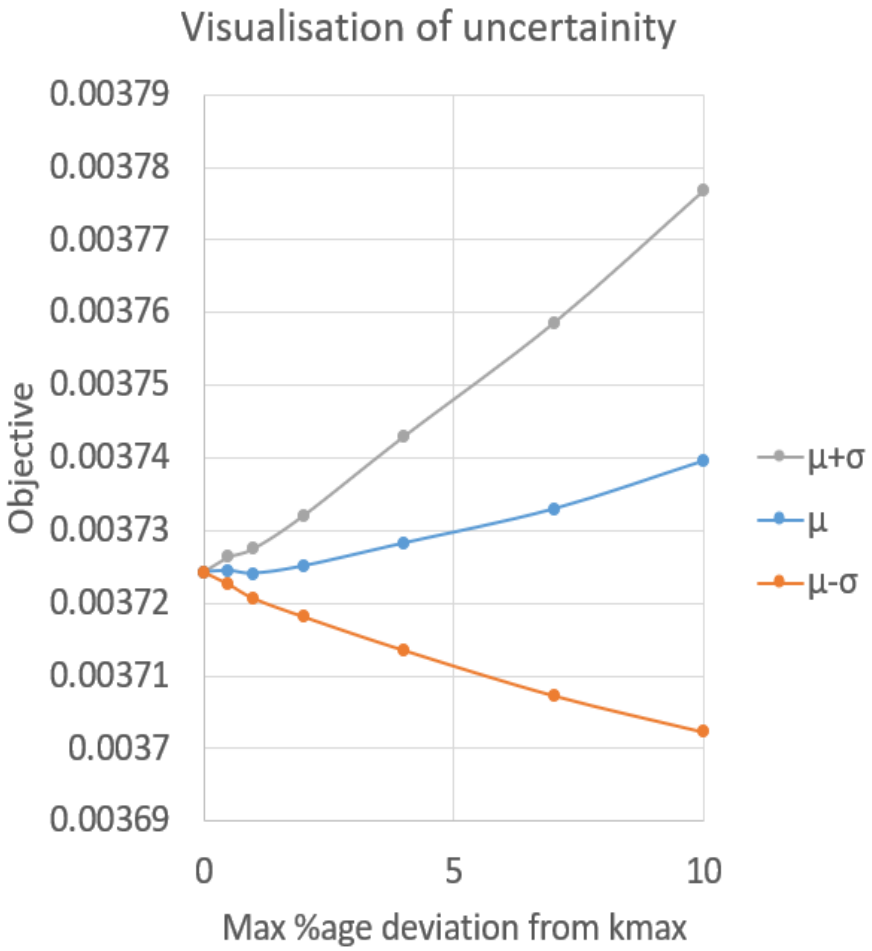
```
option randseed 7;
```

we were unable to automate this process, and hence were limited by manual readings only.

Results:

Value of objective with continuous springs: 0.00362112 (lower is better)
with discrete springs: 0.00372425

The discretisation gives a worse result as expected (3% worse). The results for introducing randomness are summarised in the following figure:



The y-axis denotes the objective, the x-axis the uncertainty in the spring constants.

We can clearly see that the mean of the result is moving upwards as we increase the uncertainty. This indicates that the structure is weakened more easily than it is strengthened, which we expect intuitively as well.

Difficulties

We ran into a large number of problems throughout this project, both due to our mistakes and the solver's issues. They are listed below:

1. The spring constants must remain at a small, finite value in the regions where material is not present. Ideally for a vacuum, we would like these to be 0. As we have verified the physics of the results for other cases, we believe this issue is caused entirely by the solver, which does not converge when kmin is lowered.
2. Large deformations or forces caused the solver to not converge.
3. Due to memory limitations, we often could not use more than ~2000 points for the domain.
4. A large number of points slowed down the program significantly.
5. Certain boundary conditions simply do not converge. This has no apparent cause.
6. Apply compressive loads on springs can cause them to 'flip over' and switch the positions of two neighbouring nodes, resulting in non-physical results. This is caused by the formulation itself and all our attempts at solving this were fruitless.
7. Real materials are not made of springs, and our attempts to use STVK model failed.

Conclusion

There have been a number of methods, accurate or inaccurate, that have been used for the optimization. Those methods yielded different results. So finally, following are the conclusions that can be drawn from the work and this report.

- Minimization of energy method does not work when solving using stress and strain relationships.
- Spring method used here, can alternatively be used in place of the material properties for getting topology optimization.
- This method of solving (using nodes and nodal elements) resembles the method used in Finite Element Analysis, so the results can be verified easily using any FEM software like Abaqus, Ansys, etc.

From the topology optimization results:

a. For small loads, the distribution for cantilever beam tends to be symmetric.
b. As the resolution increases, more fine plots can be seen, which means more finely distributed materials.
c. As the material amount increases, some beautiful patterns can be drawn as there is enough material which can handle the forces.
d. Different objectives can easily influence the final structure, as we saw when many supports appeared in the bridge optimization going from the bottom-centre to the arc.
e. Similarly, varying the loads at different locations yields different pattern results.
f. Applying uncertainty to the spring constants weakens the structure on average.

Finally, we would like to say that we really enjoyed working on this project as the beautiful final results we obtained were derived from very basic principles, which makes us appreciate the study of optimization.

THANK YOU!

Appendix

AMPL code for Stress-Strain

We present the formulation in terms of AMPL code snippets for ease of use:

These parameters define the length and width of the domain and the number of discrete points. The test domain has been chosen to be square with equally spaced points:

```
param L = 1;
param H = 1;

param n = 30;
param m = 30;

param dx = L/(n-1);
param dy = H/(m-1);
```

Elastic coefficients for the STVK stress:

```
param lambda = 15;
param mu = 3.6;
```

Deflections at each point:

```
var ux{1..n,1..m};
var uy{1..n,1..m};
```

A simple linear approximation is used to find the symmetric strain tensor from the deflections:

```
var e11{i in 1..(n-1),j in 1..(m-1)} =
     0.5 * (ux[i+1,j] + ux[i+1,j+1] - ux[i,j] - ux[i,j+1]) / dx;
var e12{i in 1..(n-1),j in 1..(m-1)} =
     0.25 * (ux[i,j+1] + ux[i+1,j+1] - ux[i,j] - ux[i+1,j]) / dy
+
     0.25 * (uy[i+1,j] + uy[i+1,j+1] - uy[i,j] - uy[i,j+1]) / dx;
var e22{i in 1..(n-1),j in 1..(m-1)} =
     0.5 * (uy[i,j+1] + uy[i+1,j+1] - uy[i,j] - uy[i+1,j]) / dy;
```

The stress tensor is obtained from the strains:

```
var s11{i in 1..(n-1),j in 1..(m-1)} =
     (lambda * (e11[i,j] + e22[i,j]) + (2 * mu * e11[i,j]));
var s12{i in 1..(n-1),j in 1..(m-1)} =
     (2 * mu * e12[i,j]);
```

```
var s22{i in 1..(n-1),j in 1..(m-1)} =
     (lambda * (e11[i,j] + e22[i,j]) + (2 * mu * e22[i,j]));
```

At this step, the boundary conditions need to be imposed. This fixes the left edge of the domain:

```
subject to fixedx {j in 1..m}: ux[1,j] = 0;
subject to fixedy {j in 1..m}: uy[1,j] = 0;
```

Now we need to ensure the domain will bend. There are two different ways to do so:

1) Neumann boundary condition-
   Here, the force applied at a particular point is known, and hence the strains are specified. This applies a vertical force on the centre of the right edge of the domain:

   ```
   subject to forcex: s11[n-1,m/2] = 0;
   subject to forcey: s12[n-1,m/2] = 10;
   ```

2) Dirichlet boundary condition-
   Here, the displacement at a given point is known. This moves one point vertically up from the centre of the right edge of the beam:

   ```
   subject to fixed2x: ux[n,m/2] = 0;
   subject to fixed2y: uy[n,m/2] = 0.2;
   ```

For the final step, the energy stored in the beam should be the least possible for the correct deformation. The energy density is given by the inner product of the stress and strain tensors:

```
minimize energy: sum{i in 1..(n-1),j in 1..(m-1)}
     ((s11[i,j]*e11[i,j]) + (2*s12[i,j]*e12[i,j]) +
(s22[i,j]*e22[i,j]));
```

AMPL code for grid of springs

Going through the AMPL code formulation:

Defining domain size and geometry as before:

```
param L = 3;
param H = 1;

param n = 30;
param m = 10;

param dx = L/(n-1);
param dy = H/(m-1);
param dl = sqrt(dx^2 + dy^2);
```

We have used the absolute position instead of relative displacement in this method. They are initialised to a regular grid , otherwise the solver will not converge:

```
var ux{i in 1..n,j in 1..m} := (i - 1) * dx;
var uy{i in 1..n,j in 1..m} := (j - 1) * dy;
```

Defining the spring constants for each type of spring (in this code, the suffixes x/h, y/v, d1 and d2 represent horizontal, vertical, upward diagonal and downward diagonal respectively):

```
param kx{i in 1..(n-1),j in 1..m} := 1;
param ky{i in 1..n,j in 1..(m-1)} := 1;
param kd1{i in 1..(n-1),j in 1..(m-1)} := 1;
param kd2{i in 1..(n-1),j in 1..(m-1)} := 1;
```

Defining the length of each spring after deformation, found from basic Pythagorean relation:

```
var lh{i in 1..(n-1),j in 1..m} = sqrt((ux[i,j] - ux[i+1,j])^2 +
(uy[i,j] –
uy[i+1,j])^2);

var lv{i in 1..n,j in 1..(m-1)} = sqrt((ux[i,j] - ux[i,j+1])^2 +
(uy[i,j] –
uy[i,j+1])^2);

var ld1{i in 1..(n-1),j in 1..(m-1)} = sqrt((ux[i,j] -
ux[i+1,j+1])^2 + (uy[i,j] –
uy[i+1,j+1])^2);
```

```
var ld2{i in 1..(n-1),j in 1..(m-1)} = sqrt((ux[i+1,j] -
ux[i,j+1])^2 + (uy[i+1,j] - uy[i,j+1])^2);
```

F = ks for linear springs:

```
var fh{i in 1..(n-1),j in 1..m} = kx[i,j] * (lh[i,j]-dx);
var fv{i in 1..n,j in 1..(m-1)} = ky[i,j] * (lv[i,j]-dy);
var fd1{i in 1..(n-1),j in 1..(m-1)} = kd1[i,j] * (ld1[i,j]-dl);
var fd2{i in 1..(n-1),j in 1..(m-1)} = kd2[i,j] * (ld2[i,j]-dl);
```

Now we define the spring forces acting on each point. For points along the border, some of these have been set to 0 using if else conditions since certain types of springs will be missing for them. In the following expressions, the components of the forces have been resolved along the global x and y coordinates (x and y suffix indicate global x and global y direction here):

```
var fhx{i in 1..n,j in 1..m} =
     (if i == n then 0 else (fh[i,j] * (ux[i+1,j] - ux[i,j]) /
lh[i,j])) -
     (if i == 1 then 0 else (fh[i-1,j] * (ux[i,j] - ux[i-1,j]) /
lh[i-1,j]));

var fhy{i in 1..n,j in 1..m} =
     (if i == n then 0 else (fh[i,j] * (uy[i+1,j] - uy[i,j]) /
lh[i,j])) -
     (if i == 1 then 0 else (fh[i-1,j] * (uy[i,j] - uy[i-1,j]) /
lh[i-1,j]));
```

```
var fvx{i in 1..n,j in 1..m} =
     (if j == m then 0 else (fv[i,j] * (ux[i,j+1] - ux[i,j]) /
lv[i,j])) -
     (if j == 1 then 0 else (fv[i,j-1] * (ux[i,j] - ux[i,j-1]) /
lv[i,j-1]));

var fvy{i in 1..n,j in 1..m} =
```

```
     (if j == m then 0 else (fv[i,j] * (uy[i,j+1] - uy[i,j]) /
lv[i,j])) -
     (if j == 1 then 0 else (fv[i,j-1] * (uy[i,j] - uy[i,j-1]) /
lv[i,j-1]));



var fd1x{i in 1..n,j in 1..m} =
     (if (j == m or i == n) then 0 else (fd1[i,j] * (ux[i+1,j+1]
- ux[i,j]) / ld1[i,j])) -
     (if (j == 1 or i == 1) then 0 else (fd1[i-1,j-1] * (ux[i,j]
- ux[i-1,j-1]) / ld1[i-1,j-1]));

var fd1y{i in 1..n,j in 1..m} =
     (if (j == m or i == n) then 0 else (fd1[i,j] * (uy[i+1,j+1]
- uy[i,j]) / ld1[i,j])) -
     (if (j == 1 or i == 1) then 0 else (fd1[i-1,j-1] * (uy[i,j]
- uy[i-1,j-1]) / ld1[i-1,j-1]));



var fd2x{i in 1..n,j in 1..m} =

     (if (j == 1 or i == n) then 0 else (fd2[i,j-1] * (ux[i+1,j-
1] - ux[i,j]) / ld2[i,j-1])) -
     (if (j == m or i == 1) then 0 else (fd2[i-1,j] * (ux[i,j] -
ux[i-1,j+1]) / ld2[i-1,j]));

var fd2y{i in 1..n,j in 1..m} =
     (if (j == 1 or i == n) then 0 else (fd2[i,j-1] * (uy[i+1,j-
1] - uy[i,j]) / ld2[i,j-1])) -
     (if (j == m or i == 1) then 0 else (fd2[i-1,j] * (uy[i,j] -
uy[i-1,j+1]) / ld2[i-1,j]));
```

Finally, we obtain the net force on each node:

```
var fx{i in 1..n,j in 1..m} = fhx[i,j] + fvx[i,j] + fd1x[i,j] +
fd2x[i,j];
```

```
var fy{i in 1..n,j in 1..m} = fhy[i,j] + fvy[i,j] + fd1y[i,j] +
fd2y[i,j];
```

Now we can impose constraints on the nodes. We have a choice to either specify a displacement at a node, or a force, as along as the number of provided constraints matches the number of free variables, AMPL will find the correct, and only feasible solution.

Currently, we have chosen to keep the left edge fixed, and move the bottom-right corner upwards (all other nodes receive the noforce condition):

```
subject to fixedx {j in 1..m}: ux[1,j] = 0;
subject to fixedy {j in 1..m}: uy[1,j] = (j - 1) * dy;

subject to fixed2x: ux[n,1] = L;
subject to fixed2y: uy[n,1] = 2;

subject to noforce2x {j in 2..m}: fx[n,j] = 0;
subject to noforce2y {j in 2..m}: fy[n,j] = 0;

subject to noforcex {i in 2..(n-1), j in 1..m}: fx[i,j] = 0;
subject to noforcey {i in 2..(n-1), j in 1..m}: fy[i,j] = 0;
```

1) Isotropic method:

h is a measure of the amount of material present:

```
var h{i in 1..(n-1), j in 1..(m-1)} =
    (kx[i,j] + ky[i,j] + kx[i,j+1] + ky[i+1,j])/8 + (kd1[i,j] +
kd2[i,j])/4;
```

The spring constants are constrained by fixed parameters:

```
param kmin = 0.3;
param kmax = 1;

subject to kxlimit {i in 1..(n-1),j in 1..m}: kmin <= kx[i,j] <=
kmax;
subject to kylimit {i in 1..n,j in 1..(m-1)}: kmin <= ky[i,j] <=
kmax;
```

```
subject to kd1limit {i in 1..(n-1),j in 1..(m-1)}: kmin <=
kd1[i,j] <= kmax;
subject to kd2limit {i in 1..(n-1),j in 1..(m-1)}: kmin <=
kd2[i,j] <= kmax;
```

The total mass of springs is also limited:

```
subject to matlim:
     0.25 * (((sum{i in 1..(n-1),j in 1..m} kx[i,j]) / (m * (n-1)
* kmax)) +
     ((sum{i in 1..n,j in 1..(m-1)} ky[i,j]) / (n * (m-1) *
kmax)) +
     ((sum{i in 1..(n-1),j in 1..(m-1)} kd1[i,j]) / ((n-1) * (m-
1) * kmax)) +
     ((sum{i in 1..(n-1),j in 1..(m-1)} kd2[i,j]) / ((n-1) * (m-
1) * kmax))) = 0.6;
```

2) Anisotropic method:

h is defined as a free variable:

```
var h{i in 1..(n-1), j in 1..(m-1)};
```

and the spring constants are derived from it:

```
var kx{i in 1..(n-1),j in 1..m} = kmin + 0.5 * (kmax – kmin) * (
     (if j == 1 then 0 else h[i,j-1]) + (if j == m then 0 else
h[i,j]));
var ky{i in 1..n,j in 1..(m-1)} = kmin + 0.5 * (kmax – kmin) * (
     (if i == 1 then 0 else h[i-1,j]) + (if i == n then 0 else
h[i,j]));
var kd1{i in 1..(n-1),j in 1..(m-1)} = kmin + (kmax – kmin) *
h[i,j];
var kd2{i in 1..(n-1),j in 1..(m-1)} = kmin + (kmax – kmin) *
h[i,j];
```

The total mass is limited through h:

```
subject to matlim:
     ((sum{i in 1..(n-1),j in 1..(m-1)} h[i,j]) / ((m-1) * (n-
1))) = 0.6;
```