Introduction to Truss Structures Optimization with Python

Ernesto Aranda *José Carlos Bellido*
e-mail ernesto.aranda@uclm.es josecarlos.bellido@uclm.es
Departmento de Matemáticas
Universidad de Castilla - La Mancha
Spain

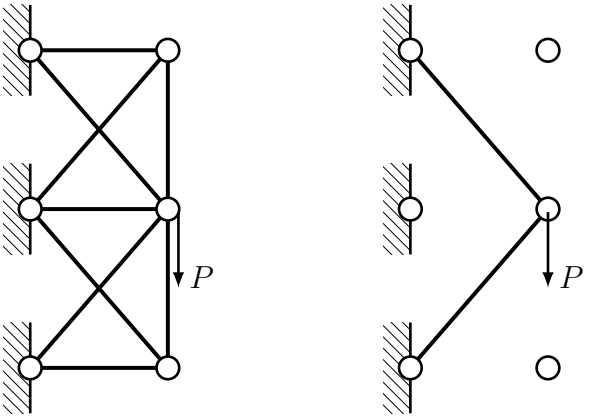
Abstract

In this note we introduce the classical problem of optimizing a truss structure in a pedagogical fashion. The truss will consist in a number of nodes (in our context two-dimensional points) and connections between those points (in our context elastic bars). Subject to certain supporting and loads conditions on the structure, our aim is to design the stiffest structure among those whose volume do not exceed a given tolerance. The design variables are the cross section areas of the bars connecting the different nodes of the structure. We introduce this nonlinear optimization problem, go a bit into its mathematical analysis, and propose a basic numerical algorithm for getting optimal solutions that is implemented in Python.

1 Introduction

In order to set the problem up we consider a set of fixed two-dimensional points, that we will call *nodes*, and a number of connections between those points. We will assume that those connections are *elastic bars*. The resulting mechanical structure is called a *truss*. This structure can be supported in a variety of ways, so that the displacements for some of the nodes may be restricted, and the rest of the nodes are free to displace in any direction. Further, loads may be considered to act on certain nodes of the structures. See Figure 1a.

Our aim is to design the best possible structure by choosing the cross section areas of the truss members. The cross section area of a elastic bar is a structural thickness nature variable, so that this is a structural optimization problem for the truss structure. In the literature this problem is included into the category of sizing optimization problems, since we are optimizing the size of the structural elements of the truss; however, if we allow those areas to take the value zero in the optimization problem, then the problem falls into the category of topology optimization problems, since when that happens, bars are removed form the structure, changing the connectivity between nodes and consequently the topology of the truss structure (see Figure 1b). Criteria to be optimized and constraints for this problem could be of any kind with physical meaning, and among the most used because of their interest in engineering is to maximize the stiffness of the structure subject to a constraint on the maximal volume of the truss. This is the problem that we consider in this paper. Having this in mind



(a) Initial truss (b) Optimized truss

Figure 1: Example of a truss structure before and after optimization

the result for the optimized structure in Figure 1 is mechanically intuitive, since separating the bars into the structure as much as possible increases its inertia as much as possible, thus making the truss stiffest. Of course the permitted maximal volume is small enough for getting a structure with only two bars as optimized design.

The history of truss optimization problems is long and goes back to 1904, when A. Michell published its pioneering work [7], and was able to obtained optimality conditions for minimal weight topologies in geometrically simple cases under punctual load conditions. In particular, in those optimal structures all the bars work under the same stress, either under tension or compression, and globally the truss is an orthogonal curves system. In Figure 2 two examples of Michell type structures are shown. There was a lack of interest on the subject until the sixties when the use of computers allowed to attack more realistic problems. There has been a great deal of work on the subject since then and we can affirm that it is now a very well developed subject. There are many references on it, and we just cite [3, 2, 8] as good accounts on the subject. [1] is a general basic reference on topology optimization.

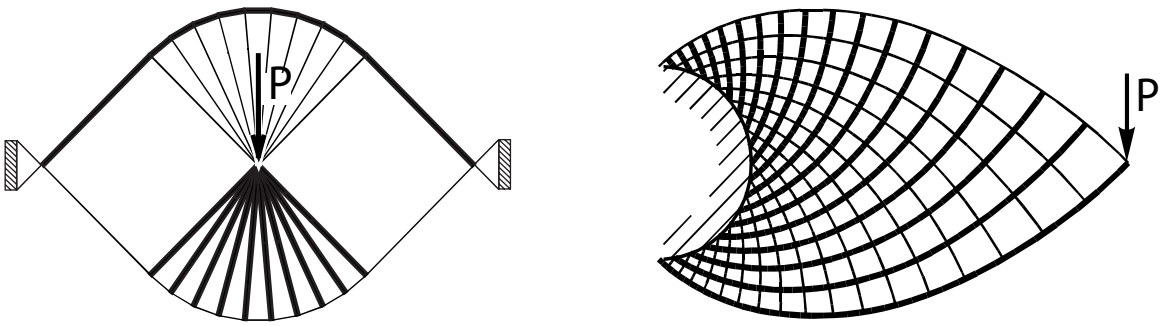


Figure 2: Michell type structures: simply supported beam-type truss (left) and cantilever-type truss (right)

In this introductory note on the subject we focus on the model problem of designing the stiffest truss structure such that its volume is below a prescribed value. The plan of the paper is the following: in section 2 we introduced the problem, going into the mechanical modeling and the mathematical optimization problem. The contents of this section are based on [2, Ch.5]. Section 3 is devoted to the computational implementation in Python of an algorithm for solving the problem. Some examples are shown in section 4 and codes are shown in section 5.

Finally, we remark that we work on the planar two-dimensional situation for simplicity in the exposition, but all the analysis can be extended without much effort to the three-dimensional case.

2 The optimization problem

In this section we obtain the optimization problem we are interested on. We assume elastic linear behavior for the truss bars. For the sake of simplicity, we will assume that our truss is made of bars of the same material with constant Young modulus E, although different materials bars can be considered in the algorithm in section 3.

If we collect together the cross section areas of all the bars to be optimized in the truss in a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$, (we will use boldface characters for vector and matrices) n being the number of such bars, then our optimization problem has the form

$$\underset{\mathbf{x}\in\mathcal{U}_{\text{ad}}}{\text{Minimize}}\ C(\mathbf{x}) = \mathbf{F}^T\mathbf{U} \tag{1}$$

subject to

$$\mathbf{K}(\mathbf{x})\mathbf{U} = \mathbf{F} \tag{2}$$

and the volume constraint

$$\sum_{j=1}^{n} l_j x_j \leq V_{\max}. \tag{3}$$

The feasible set for the optimization problem, $\mathcal{U}_{\text{ad}}$, is given by

$$\mathcal{U}_{\text{ad}} = \left\{\mathbf{x} \in \mathbb{R}^n :\ x_j^{\min} \leq x_j \leq x_j^{\max},\ j = 1, \ldots, n\right\},$$

where $x_j^{\min}$, $x_j^{\max}$ are respectively the maximal and minimal cross section areas allowed for the j-th bar. We call m the number of free nodes of the truss, that is to say, nodes whose displacement is not constrained by the boundary conditions. Then $\mathbf{U} \in \mathbb{R}^{2m}$ stands for the displacement vector and $\mathbf{F} \in \mathbb{R}^{2m}$ stands for the vector of loads applied on the nodes of the structure. Recall that our model is two-dimensional, thus if the number of free nodes is m then the number of components of the displacement and load vector must be $2m$. The cost functional $C(\mathbf{x}) = \mathbf{F}^T\mathbf{U}$, is the *compliance*, or the work made by the load to deform the structure. Minimizing the compliance is equivalent to maximizing the stiffness of the truss, since the smallest the compliance is, the stiffest the truss is. The equilibrium equation of the system is given by the linear system of equations (2), where $\mathbf{K}(\mathbf{x})$ stands for the stiffness matrix of the structure. In section 2.1 we show how the stiffness matrix is obtained and why it depends on the cross section areas of the bars of the truss. Finally, in the volume constraint (3), l_j stands for the length of the j-th bar, so that its volume is $l_j x_j$, and the total truss volume is

$$\sum_{j=1}^{n} l_j x_j.$$

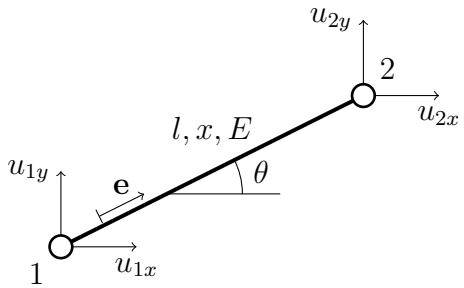


Figure 3: Single bar with notation

$V_{\max}$ in (3) is the maximal volume allowed for the structure.

In practice $x_j^{\min}$ is a very small but strictly positive value, so that we assume that if this value is reached in the optimized design, then j-th bar is removed from the truss. This makes perfect mechanical sense since the stiffness contribution of such a bar to the truss will be nonsignificant. We do not consider $x_j^{\min} = 0$ because the stiffness matrix $\mathbf{K}(\mathbf{x})$ might become singular during the optimization process.

2.1 Equilibrium equations

In this section we obtain the truss equilibrium equations (2). First of all we focus on obtaining the equilibrium equation of a single homogeneous bar with constant cross section area. Let us take as a model the bar of Figure 3. Here E, ρx and l stand for, respectively, the Young modulus, the cross section area and the length of the bar. θ is the angle of the bar with the horizontal, and $\mathbf{e}$ is the unitary director vector of the bar, namely

$$\mathbf{e} = \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}. \tag{4}$$

The vectors

$$\mathbf{u}_1 = \begin{pmatrix} u_{1x} \\ u_{1y} \end{pmatrix}, \ \mathbf{u}_2 = \begin{pmatrix} u_{2x} \\ u_{2y} \end{pmatrix}$$

stand for the displacement vectors on the bar extremes, or nodes. It will be useful to collect these two vectors together in the column matrix

$$\mathbf{u} = \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix}.$$

We also introduce the column matrix $\mathbf{f} = \begin{pmatrix} f_{1x} & f_{1y} & f_{2x} & f_{2y} \end{pmatrix}^T$ where

$$\begin{pmatrix} f_{1x} \\ f_{1y} \end{pmatrix}, \ \begin{pmatrix} f_{2x} \\ f_{2y} \end{pmatrix}$$

are the force vectors acting respectively on nodes 1 and 2 of the bar. If s denotes the force in the bar, since in this model forces act longitudinally, either by compression ($s < 0$) or by tension ($s > 0$), then the relation

$$\mathbf{f} = \begin{pmatrix} -\mathbf{e} \\ \mathbf{e} \end{pmatrix} s \tag{5}$$

holds.

The elongation of the bar, δ, can be obtained as

$$\delta = (\mathbf{u}_2 - \mathbf{u}_1)^T \mathbf{e} = \begin{pmatrix} -\mathbf{e} \\ \mathbf{e} \end{pmatrix}^T \mathbf{u}. \tag{6}$$

By Hooke's law we have that force and elongation are related through the relation

$$s = D\delta, \tag{7}$$

where the stiffness is given by

$$D = \frac{Ex}{l}.$$

For later use, we introduce the stress σ, and the strain ε, of the bar, related in the following way

$$s = \sigma x = E\varepsilon x_j. \tag{8}$$

Recall that the strain verifies $\varepsilon = \frac{\delta}{l}$. Inserting (6) and (7) into (5) we arrive at

$$\mathbf{f} = D \begin{pmatrix} -\mathbf{e} \\ \mathbf{e} \end{pmatrix} \begin{pmatrix} -\mathbf{e} \\ \mathbf{e} \end{pmatrix}^T \mathbf{u},$$

and if we call $\mathbf{k}^0$ to the matrix

$$\mathbf{k}^0 = D \begin{pmatrix} -\mathbf{e} \\ \mathbf{e} \end{pmatrix} \begin{pmatrix} -\mathbf{e} \\ \mathbf{e} \end{pmatrix}^T = \frac{E}{l} \left(\begin{array}{c|c} \mathbf{e}\,\mathbf{e}^T & -\mathbf{e}\,\mathbf{e}^T \\ \hline -\mathbf{e}\,\mathbf{e}^T & \mathbf{e}\,\mathbf{e}^T \end{array} \right) \tag{9}$$

and $\mathbf{k}(x) = x\mathbf{k}^0$, then the equilibrium equations for the single bar are given by the linear system

$$\mathbf{k}(x)\mathbf{u} = \mathbf{f}. \tag{10}$$

The equilibrium equation for the whole truss structure is a linear system like (10), and the global stiffness matrix is obtained by a procedure called assembly. In practice, for a truss structure, assembly is actually very simple, and after a numeration of bars and nodes, we have just to sum over all bars the elemental stiffness matrix $\mathbf{k}(x)$ in right positions into the global stiffness matrix $\mathbf{K}(\mathbf{x})$. In section 3.2 we show how this is computationally implemented. For the example of Figure 1, the equilibrium equations can be computed by hand, and since we have six degrees of freedom for the displacements the stiffness matrix will be an order six matrix ($m = 6$). In this case vector $\mathbf{x} \in \mathbb{R}^{\mathbf{9}}$, $n = 9$. For examples on explicit computation on equilibrium equations for elementary trusses like this, and simpler, we refer the interested readers to standard textbooks like [2].

Finally, we would like to remark that the global stiffness matrix $\mathbf{K}(\mathbf{x})$ is symmetric, which is straightforward due to the fact that $\mathbf{k}(x)$ is symmetric and the way in which elemental matrices are assembled (diagonal terms of elementary stiffness matrices sum on the diagonal terms of the global stiffness matrix). Also, stiffness matrix is positive semidefinite. In order to show this, we have to check that

$$\mathbf{U}^T \mathbf{K}(\mathbf{x}) \mathbf{U} \geq 0 \tag{11}$$

for any displacement vector $\mathbf{U}$ and any design vector $\mathbf{x} \in \mathcal{U}_{\text{ad}}$. An elementary computation shows that

$$\mathbf{U}^T\mathbf{K}(\mathbf{x})\mathbf{U} = \sum_{j=1}^{n} \mathbf{u}_j^T \mathbf{k}(x_j)\mathbf{u}_j,$$

where x_j and $\mathbf{u}_j$ are respectively the cross section area and the displacement vector of the j-th bar of the truss. Then, since $\mathbf{k}(x)$ is positive semidefinite for any $x > 0$, we conclude that (11) holds, or, in other words, $\mathbf{K}(\mathbf{x})$ is positive semidefinitive. If the structure is properly fixed with suitable boundary conditions, then the resulting global stiffness matrices will be positive definite. This can be shown easily with the following physical argument. Let SE_j be the j-th bar strain energy, given by

$$SE_j = \frac{1}{2}\sigma_j \varepsilon_j l_j, \tag{12}$$

where σ_j and ε_j are, respectively, the j-bar stress and strain. Operating, and δ_j being the elongation and $\mathbf{e}_j$ the unitary director vector of the j-th bar, we have that

$$SE_j = \frac{1}{2}E\varepsilon_j^2 x_j l = \frac{1}{2}E\frac{\delta_j^2}{l_j^2}x_j l_l = \frac{1}{2}E\frac{\left[\begin{pmatrix}-\mathbf{e}_j\\ \mathbf{e}_j\end{pmatrix}^T \mathbf{u}_j\right]^2}{l_j^2}x_j l_j$$

$$= \frac{1}{2}\mathbf{u}_j^T \frac{Ex_j}{l_j}\begin{pmatrix}-\mathbf{e}_j\\ \mathbf{e}_j\end{pmatrix}\begin{pmatrix}-\mathbf{e}_j\\ \mathbf{e}_j\end{pmatrix}^T \mathbf{u}_j = \frac{1}{2}\mathbf{u}_j^T \mathbf{k}_j(x_j)\mathbf{u}_j$$

For later use, from the last identity, and using (8) and (12), the following equality

$$\mathbf{u}_j^T \mathbf{k}_j(x_j)\mathbf{u}_j = \frac{\sigma_j^2}{E} l_j \tag{13}$$

holds.

Then, the strain energy of the whole bar is

$$SE = \sum_j SE_j = \frac{1}{2}\mathbf{U}^T\mathbf{K}(\mathbf{x})\mathbf{U}.$$

Now if the boundary conditions avoid rigid motion then there cannot be a displacement different of zero with zero strain energy, and this translates into mathematical terms by saying that $\mathbf{K}$ is positive definite.

2.2 Mathematical analysis

Now we go into some mathematical questions on the optimal design problem. In order to do this, we assume that the global stiffness matrix is positive definitive for any admissible design $\mathbf{x}$. Then, for any $\mathbf{x} \in \mathcal{U}_{\text{ad}}$ there exists a unique solution of the system

$$\mathbf{K}(\mathbf{x})\mathbf{U} = \mathbf{F}, \tag{14}$$

which we denote $\mathbf{U}(\mathbf{x})$. Then, the nested formulation of the problem is

$$\min_{\mathbf{x}\in\mathcal{U}_{\text{ad}}} C(\mathbf{x}) = \mathbf{F}^T\mathbf{U}(\mathbf{x})$$

subject to

$$\sum_{j=1}^{n} x_j l_j \leq V_{\max}.$$

This is a non-linear programming problem. The constraints now are just the box constraints (those defining the admissible set) and a linear constraint (the volume constraint). Existence of optimal solutions can be shown in many ways, but for a finite dimensional problem like this, probably the easiest one is just by realizing that the feasible set for the optimization problem, $\mathcal{U}_{\text{ad}}$ is a compact subset of $\mathbb{R}^n$ and that the cost functional, C, is continuous on this set.

For the numerical algorithm we will need the derivatives of the cost functional C with respect to the design variables x_j. This is obtain in a straightforward manner

$$\frac{\partial C(\mathbf{x})}{\partial x_j} = \mathbf{F}^T\frac{\partial \mathbf{U}(\mathbf{x})}{\partial x_j} = (\mathbf{K}(\mathbf{x})\mathbf{U}(\mathbf{x}))^T\frac{\partial \mathbf{U}(\mathbf{x})}{\partial x_j} = \mathbf{U}(\mathbf{x})^T\mathbf{K}(\mathbf{x})\frac{\partial \mathbf{U}(\mathbf{x})}{\partial x_j}. \tag{15}$$

If we now derive the state equation, we get

$$\frac{\partial \mathbf{K}(\mathbf{x})}{\partial x_j}\mathbf{U}(\mathbf{x}) + \mathbf{K}(\mathbf{x})\frac{\partial \mathbf{U}(\mathbf{x})}{\partial x_j} = 0,$$

and then

$$\frac{\partial \mathbf{U}(\mathbf{x})}{\partial x_j} = -\mathbf{K}(\mathbf{x})^{-1}\frac{\partial \mathbf{K}(\mathbf{x})}{\partial x_j}\mathbf{U}(\mathbf{x}). \tag{16}$$

Finally, substituting (16) into (15) we find out

$$\frac{\partial C(\mathbf{x})}{\partial x_j} = -\mathbf{U}(\mathbf{x})^T\frac{\partial \mathbf{K}(\mathbf{x})}{\partial x_j}\mathbf{U}(\mathbf{x}).$$

The derivative of the stiffness matrix with respect to x_j is very easy to obtain, since x_j only appears in $\mathbf{K}(\mathbf{x})$ multiplying elemental stiffness $\mathbf{k}_j$ (in the positions where this elemental matrix occupies when $\mathbf{K}(\mathbf{x})$ is assembled), so that

$$\frac{\partial C(\mathbf{x})}{\partial x_j} = -\mathbf{u}_j^T\mathbf{k}_j\mathbf{u}_j.$$

Recall that $\mathbf{u}_j$ is the displacement vector of the j-th bar. In terms of strain and stress, and according to (13), the derivative can be written as

$$\frac{\partial C(\mathbf{x})}{\partial x_j} = -\frac{\sigma_j^2 l_j}{E}. \tag{17}$$

We could go further in our analysis obtaining first-order optimality (sufficient) conditions for this problem, like the Karusk-Kuhn-Tucker conditions (KKT). This problem can be shown to be convex, and KKT become also necessary conditions, so any critical point verifying KKT is also an optimal design. For the simple example of Figure 1 the optimal design could be obtained by hand, and after a bit of tedious calculations, from the KKT conditions. To Rather to delve in these issues, the aim of this paper is to get into the computational implementation of this simple, but real-world engineering, optimization problem, so that we refer the interested readers in this more mathematical questions to the books and accounts already referenced in the introduction of the paper.

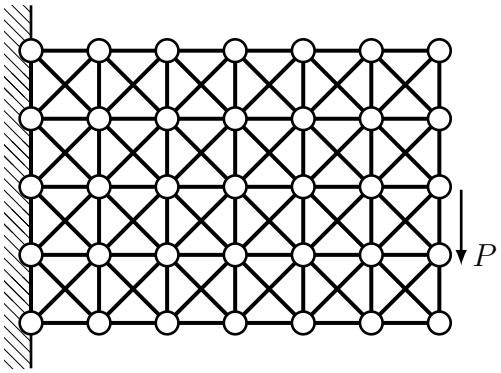


Figure 4: Truss structure and loads for the examples

3 Implementation

We have used the Python language for the implementation. Although Python is still behind C/C++ in the (scientific) programming language usage statistics, it is one of the languages with the faster spreading rate, a fact probably due to the combination of its simplicity, power and readability. Statistics available at different sites[1] show that it consistently ranks among the first five languages of choice with respect to several criteria (and for some of them, it is ranked number one). Finally, the availability of several specialized libraries for optimization problems, makes it well suited for use in the classroom. Also, there many interesting books on Mathematics with Python, as for example, [5].

We have divided our implementation in two parts. In the first one, we build a uniform truss structure with nodes and bars, and in the second one, we implement the code for the structural optimization. We will show in the examples the user interface for setting up and solving a particular problem.

3.1 Creation of the truss structure

The relevant information in a truss structure is given by the coordinates of the nodes and the bars connecting the nodes. To build a uniform structure as in Figure 4, we define a rectangle given by their lower left and upper right corners, (each of them is a tuple of two elements for the coordinates), and the number of nodes used in each direction (given by two integers); those are the parameters used in the function `meshtruss` (see Listing 1). The function returns two arrays: one with the coordinates of the nodes of the structure and the other one with the connecting bars.

In Listing 1, equally space nodes are created in the X and Y directions in lines 5–9. The nodes are numbered from left to right and from top to bottom. Then, for each node, the bars given in Figure 5 are created in lines 10–16. Of course, this is not valid for nodes on the right or on the top sides of the rectangle where only vertical or horizontal bars are necessary. That is done in line 17 (vertical) and lines 19–20 (horizontal).

For example, the simple structure of Figure 6 corresponds to the following called:

```
>>> coord, connec = meshtruss((0,0), (1,1), 1, 1)
```

[1]See for example `http://redmonk.com/sogrady/2015/01/14/language-rankings-1-15/` or `http://langpop.com`

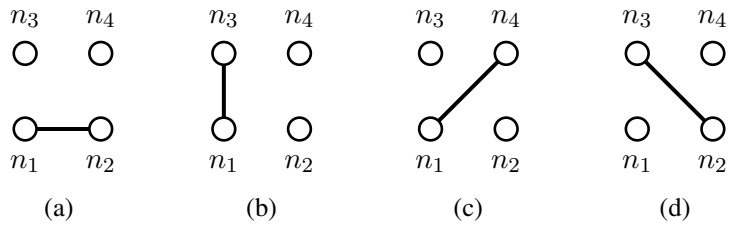


Figure 5: Creation of bars in a uniform structure

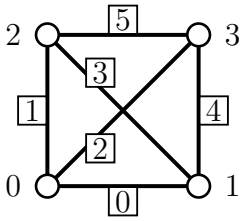


Figure 6: Basic structure

and the obtained arrays are:

```
>>> coord
array([[ 0.,  0.],
       [ 1.,  0.],
       [ 0.,  1.],
       [ 1.,  1.]])
>>> connec
array([[0, 1],
       [0, 2],
       [0, 3],
       [1, 2],
       [1, 3],
       [2, 3]])
```

so that `[coord[i,0],coord[i,1]]` are the coordinates of the i-th node and `connec[j,0]` and `connec[j,1]` are the indexes of the nodes for the j-th bar ends.

3.2 Structure optimization

The main code for the structure optimization has been implemented through a function `opttruss` (see Listing 2), which allows to solve the problem (1) using the method of moving asymptotes algorithm (MMA) developed in [9], which needs that the user supplies the cost functional, the constraints and their corresponding derivatives.

The entry parameters of this function are the structure information, given by the coordinates and connections of the nodes and bars as it has been computed in section 3.1, and some additional information about the structure conditions: namely, the Young modulus of each bar, the nodes that are

(partially) fixed, the external forces acting on each node, and a parameter corresponding to the normalization of $V_{\max}$ in (3). There is also a default boolean parameter which establishes whether the displacements in the structure must be plotted.

In Listing 2, the Young modulus of each bar (`E`) has to be defined through an array of length equal to the number of bars, such that `E[j]` gives the Young modulus of the j-th bar. On the other hand, `freenode` is a 2-dimensional array such that `freenode[i,:]` is a 2-length integer array which determines the freedom of the i-th node in the directions X and Y: 0 if it is fixed, and 1 otherwise. Similarly, `F` is a 2-dimensional array such that `F[i,:]` is the external force applied on the i-th node (as a 2-length array with components X and Y). All those data has to be defined previously to the function called.

In the function `opttruss`, lines 5–10 compute the number of bars (`n`) and the number of nodes (`m`) which are obtained from the corresponding dimensions of connections and coordinates, and also the unit vectors $\mathbf{e}$ for each bar (according to (4)), and the matrix from (9)

$$B = \left(\begin{array}{c|c} \mathbf{e}\,\mathbf{e}^T & -\mathbf{e}\,\mathbf{e}^T \\ \hline -\mathbf{e}\,\mathbf{e}^T & \mathbf{e}\,\mathbf{e}^T \end{array}\right).$$

Then, three local functions are defined, all of them depending on the same variable `x`, which is the design vector of the cross section areas of all bars. The functions `fobj` and `volume` compute the objective function and the volume constraint, respectively, along with their derivatives; and `drawtruss` allows to draw the structure.

Finally, the optimization process is in lines 67–76, where we set up and solve the problem through the function **`NLP`** of the module `openopt` (cf. [6]).[2]

The interface of the function **`NLP`** (line 75) is easy to use: it needs the cost functional and its derivative (`f` and `derf`), and the constraints and their derivatives (`constr` and `dconstr`); all those values have been obtained from the functions `fobj` and `volume` which are used through a **`lambda`** function (lines 69–70). Note that the constraint has been normalized for a better performance of the algorithm (line 71).

A starting point for the iterations (`x0`) is also required; we have chosen $x_j = 10^{-4}$, $\forall j$ (line 74), and lower and upper bounds (`xmin` and `xmax`) according to the definition of the admissible set $\mathcal{U}_{\text{ad}}$. A couple of optional parameters (the name of the problem and a parameter to control how often iterations have to be printed) have also been used. The problem is solved using the MMA algorithm (line 76).

Computing displacements

In order to solve the system (2), the function `fobj` perfoms the construction of the global stiffness matrix of the structure (`K`) in lines 14–22. The assembly of the global stiffness matrix is done in line 22; this process is carried out by summing the elemental matrix $\mathbf{k}(x)$ in (10) (given by `D[i]*k0`, see (15)) to the global matrix in the appropriate places, according to the right indexes (`index`).

Once the stiffness matrix has been built, the indexes associated to fixed displacements of nodes have to be blocked, that is, taken out from the stiffness matrix and the external forces. The result are the matrix `matrix` and the right hand side `rhs`, and then the system is solved (lines 24–27).

[2] We have chosen this module because it provides the MMA algorithm through the open library NLOPT ([4]).

Finally, the displacements (`U`) of the whole structure are obtained using the solution of the system together with the zero value on the fixed nodes (lines 28–30).

Compliance and its derivative

Now, computing the compliance is straightforward (line 33), and its derivative (line 34) is easy using (17), where `s` is a vector whose components are the forces acting on each bar (according to (7)).

Drawing results

Finally, we have implemented a function `drawtruss` for drawing the optimal structure and displacements (optionally). We perform a loop over all the bars in the structure and built the original bars (`bar1`) and the displaced bars (`bar2`); for the last one we have used the displacement and a factor scale in order to highlight the figure. Also, we modify the `linewidth` parameter of **`plot`** commands to emphasize the structure according to the cross section area (`x`) of each bar. The color on the left hand side figure reflects whether the bar is working under tension (red) or compression (blue).

4 Examples

4.1 Example 1

As we have seen in the previous section, in order to find an optimal truss structure we have to define the Young modulus of each bar, loads in the nodes and fixed and free nodes, along to the structure itself.

Listing 3 shows the user interface for setting up the problem for the structure of Figure 4. For simplicity, we consider that all bars are made of the same material with constant Young modulus and there is only one vertical load on the middle right side. Note that we need to know the right index of the node (line 5). Also, we need to provide the index of nodes that are fixed; we first define all the nodes as free (value equal to 1) and then put 0 in the indexes corresponding to the fixed ones (line 7). Finally, we perform the optimization with 10% of the maximum volume allowed (line 8). The terminal will show the following:

```
------------------------- OpenOpt 0.5306 -------------------------
solver: mma   problem: Truss    type: NLP   goal: minimum
 iter   objFunVal   log10(maxResidual)
    0  8.194e+00            -100.00
  100  3.195e-01            -100.00
  179  3.179e-01            -100.00
istop: 1000
Solver:   Time Elapsed = 5.86   CPU Time Elapsed = 5.85
objFunValue: 0.31792522 (feasible, MaxResidual = 0)
```

and Figure 7 the result.

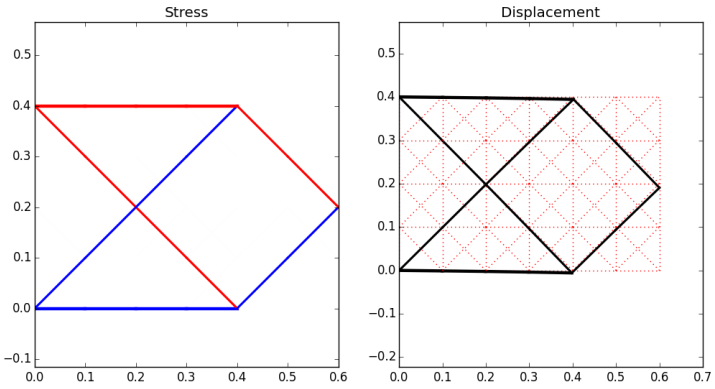
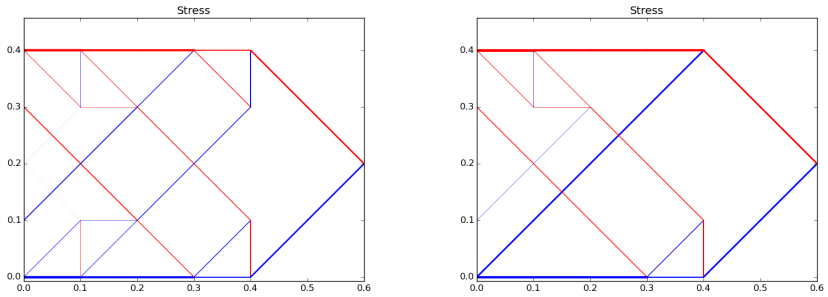


Figure 7: Result of Example 1



(a) Result without node 16

(b) Result without bar 16–22

Figure 8: Example 2

Example 2

Finally, we show a second example where we explore how would affect to the optimal structure if we delete some bars or nodes in the structure. As we have defined the truss structure through two arrays, all we have to do is to implement a couple of functions to eliminate the bars we choose. Listing 4 shows those functions.

The functions are easy to use: `remove_bar` will remove the bar between nodes `n1` and `n2` from the `connec` array, and `remove_node` will remove all the bars with origin or end in the given node. Note that the node is not removed, so that it will have to be blocked in order to have a non-singular matrix.

In the example above, we can try removing the node where diagonal bars cross each other (node 16). Listing 5 shows the code. We use the function `remove_node` to redefine the connections array and fixed the node. The result is shown in Figure 8a. We can see how the structure is forced to create parallel bars to replace the diagonal ones.

A similar situation occurs if we eliminate the bar from node 16 to 22 from the truss as we can see in Listing 6. Figure 8b shows the optimal configuration. We see how the diagonal bar working under compression remains unchanged, and the other diagonal bar is substituted by two parallel bars.

5 Codes

Listing 1: Function meshtruss: creation of a uniform truss structure

```
import numpy as np
def meshtruss(p1,p2,nx,ny):
    nodes = []
    bars = []
    xx = np.linspace(p1[0],p2[0],nx+1)
    yy = np.linspace(p1[1],p2[1],ny+1)
    for y in yy:
        for x in xx:
            nodes.append([x,y])
    for j in range(ny):
        for i in range(nx):
            n1 = i + j*(nx+1)
            n2 = n1 + 1
            n3 = n1 + nx + 1
            n4 = n3 + 1
            bars.extend([[n1,n2],[n1,n3],[n1,n4],[n2,n3]])
        bars.append([n2,n4])
    index = ny*(nx+1) + 1
    for j in range(nx):
        bars.append([index+j-1,index+j])
    return np.array(nodes), np.array(bars)
```

Listing 2: Function opttruss: optimization of a truss structure

```
from openopt import NLP
from matplotlib.pyplot import figure,show

def opttruss(coord,connec,E,F,freenode,V0,plotdisp=False):
    n = connec.shape[0]
    m = coord.shape[0]
    vectors = coord[connec[:,1],:] - coord[connec[:,0],:]
    l = np.sqrt((vectors**2).sum(axis=1))
    e = vectors.T/l
    B = (e[np.newaxis] * e[:,np.newaxis]).T


    def fobj(x):
        D = E * x/l
        kx = e * D
        K = np.zeros((2*m, 2*m))
        for i in range(n):
            aux = 2*connec[i,:]
            index = np.r_[aux[0]:aux[0]+2, aux[1]:aux[1]+2]
            k0 = np.concatenate((np.concatenate((B[i],-B[i]),axis=1),
                \
                                 np.concatenate((-B[i],B[i]),axis=1))
                                    ,axis=0)
```

```
            K[np.ix_(index,index)] = K[np.ix_(index,index)] + D[i] *
                k0

        block = freenode.flatten().nonzero()[0]
        matrix = K[np.ix_(block,block)]
        rhs = F.flatten()[block]
        solution = np.linalg.solve(matrix,rhs)
        u = freenode.astype('float').flatten()
        u[block] = solution
        U = u.reshape(m,2)
        axial = ((U[connec[:,1],:] - U[connec[:,0],:]) * kx.T).sum(
            axis=1)
        stress = axial / x
        cost = (U * F).sum()
        dcost = -stress**2 / E * l
        return cost, dcost, U, stress

    def volume(x):
        return (x * l).sum(), l

    def drawtruss(x,factor=3, wdt=5e2):
        U, stress = fobj(x)[2:]
        newcoor = coord + factor*U
        if plotdisp:
            fig = figure(figsize=(12,6))
            ax = fig.add_subplot(121)
            bx = fig.add_subplot(122)
        else:
            fig = figure()
            ax = fig.add_subplot(111)
        for i in range(n):
            bar1 = np.concatenate( (coord[connec[i,0],:][np.newaxis],
                coord[connec[i,1],:][np.newaxis]),axis=0 )
            bar2 = np.concatenate( (newcoor[connec[i,0],:][np.newaxis
                ], newcoor[connec[i,1],:][np.newaxis]),axis=0 )
            if stress[i] > 0:
                clr = 'r'
            else:
                clr = 'b'
            ax.plot(bar1[:,0],bar1[:,1], color = clr, linewidth = wdt
                * x[i])
            ax.axis('equal')
            ax.set_title('Stress')
            if plotdisp:
                bx.plot(bar1[:,0],bar1[:,1], 'r:')
                bx.plot(bar2[:,0],bar2[:,1], color = 'k', linewidth=
                    wdt * x[i])
                bx.axis('equal')
                bx.set_title('Displacement')
```

```
        show()

    xmin = 1e-6 * np.ones(n)
    xmax = 1e-2 * np.ones(n)
    f = lambda x: fobj(x)[0]
    derf = lambda x: fobj(x)[1]
    totalvolume = volume(xmax)[0]
    constr = lambda x: 1./totalvolume * volume(x)[0] - V0
    dconstr= lambda x: 1./totalvolume * volume(x)[1]
    x0 = 1e-4*np.ones(n)
    problem = NLP(f,x0,df=derf,c=constr,dc=dconstr, lb=xmin, ub=xmax,
        name='Truss', iprint=100)
    result = problem.solve("mma")

    drawtruss(result.xf)
```

Listing 3: User input for Example 1

```
coord, connec = meshtruss((0,0), (0.6,0.4), 6, 4)
E0=1e+7
E = E0*np.ones(connec.shape[0])
loads = np.zeros_like(coord)
loads[20,1] = -100.
free = np.ones_like(coord).astype('int')
free[::7,:]=0
opttruss(coord,connec,E,loads,free,0.1,True)
```

Listing 4: Additional functions for removing bars

```
def remove_bar (connec ,n1 ,n2):
    bars = connec.tolist()
    for bar in bars[:]:
        if (bar[0] == n1 and bar[1] == n2) or (bar[0] == n2 and bar
           [1] == n1):
            bars.remove(bar)
            return np.array(bars)
    else:
        print "There is no such bar"
        return connec

def remove_node(connec, n1):
    bars = connec.tolist()
    for bar in bars[:]:
        if bar[0] == n1 or bar[1] == n1:
            bars.remove(bar)
    return np.array(bars)
```

Listing 5: User input for Example 2, removing a node

```
coord, connec = meshtruss((0,0),(0.6,0.4),6,4)
connec = remove_node(connec,16)
E0=1e+7
E = E0*np.ones(connec.shape[0])
loads = np.zeros_like(coord)
loads[20,1] = -100.
free = np.ones_like(coord).astype('int')
free[16,:]=0
free[::7,:]=0
opttruss(coord,connec,E,loads,free,0.1)
```

Listing 6: User input for Example 2, removing a bar

```
coord, connec = meshtruss((0,0),(0.6,0.4),6,4)
connec = remove_bar(connec,16,22)
E0=1e+7
E = E0*np.ones(connec.shape[0])
loads = np.zeros_like(coord)
loads[20,1] = -100.
free = np.ones_like(coord).astype('int')
free[::7,:]=0
opttruss(coord,connec,E,loads,free,0.1)
```

Acknowledgements

This paper has been written after the authors taught two courses on *Structural Optimization* and *Scientific Computing with Python* as part of the VI Joint School UVEG-USALP-UCLM at the Facultad de Ciencias of the Universidad Autónoma de San Luis Potosí (Mexico) held in June 2014. We kindly thank the invitation for teaching those courses and the kind hospitality of the Departamento de Matemáticas of UASLP, and particularly we specially thank to Prof. José Antonio Vallejo.

We also acknowledge support for our research activities from Ministerio de Ciencia e Innovación (MICINN) in Spain through the grant MTM2010-19739 and we also thank the suggestions of the anonymous referee which contributed to improve the manuscript.

References

[1] M.P. Bendsøe and O. Sigmund. *Topology Optimization: theory, methods and applications*. Springer-Verlag, 2003.

[2] P.W. Christensen and A. Klarbring. *An Introduction to Structural Optimization*. Solid Mechanics and Its Applications. Springer Netherlands, 2010.

[3] R.T. Haftka and Z. Gürdal. *Elements of Structural Optimization*. Contributions to Phenomenology. Springer Netherlands, 1992.

[4] S.G. Johnson. The NLOPT nonlinear-optimization package, 2008.

[5] Jaan Kiusalaas. *Numerical methods in engineering with Python 3*. Cambridge University Press, Cambridge, 2013.

[6] Dmitrey Kroshko. OpenOpt: free scientific-engineering software for mathematical modeling and optimization, 2007.

[7] A.G.M. Michell. The limits of economy of material in frame-structures. *Philosophical Magazine Series 6*, 8(47):589–597, 1904.

[8] W.R. Spillers and K.M. MacBain. *Structural Optimization*. Springer, 2009.

[9] Krister Svanberg. The method of moving asymptotesa new method for structural optimization. *International Journal for Numerical Methods in Engineering*, 24(2):359–373, 1987.