Minimum Weight Determination of a Truss by Linear Programming

Gabe Chapel

Abstract

In this report we describe how the weight of a truss can be optimized using linear programming (LP), given an applied load. Two methods for approximating the optimization are presented along with their solutions.

1 Introduction

Linear programming is a tool used for solving complex optimization problems that do not have a closed form solution and are, otherwise, too difficult to solve. We apply this by formulating a truss topology problem as an LP so it can be solved using MATLAB's linprog() function.

An 11x20 grid is used for the truss problem, as shown in Figure 1, where each point represents a node at which truss members may connect. This provides 220 nodes to be used, represented as N, and a possible 24,090 members, represented as m.

From Figure 1, we see that a load is initially applied at the end of the grid, suggesting that a truss needs to be designed to support it. In a case like this, a company may desire to minimize the weight of the structure in an effort to reduce the cost as much as possible (e.g. material costs). Since a minimum weight problem can incorporate many variables, though, it can be more easily approximated by minimizing the number of truss members required.

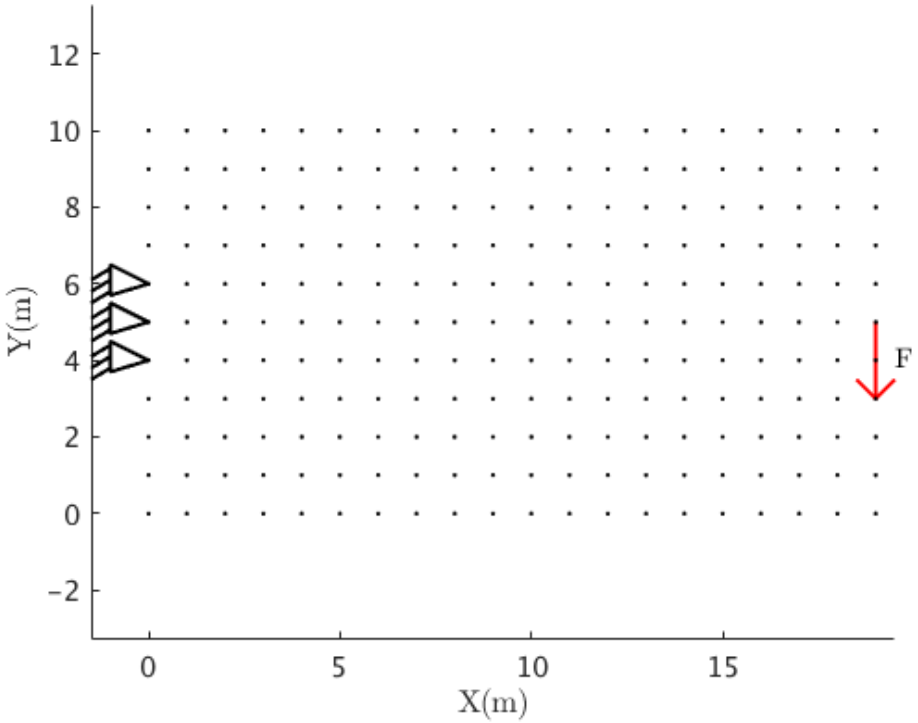

Figure 1: 11x20 Truss Grid

For the computations, it is important to establish a convention with which to distinguish each individual node and each individual member. The numbering conventions used for both the nodes and the members are displayed in Figure 2. Notice that Node 1 is in the top left corner and, from there, the nodes are counted horizontally, row by row. Similarly, the links are first counted at each individual node, with the numbering then progressing horizontally, row by row. For example, Links 1-5 in Figure 2 are all connected to Node 1, and then Links 6-9 are connected to Node 2.

To solve this problem, a system of static equilibrium equations as well as yield strength constraints can be combined to formulate an LP [1]. The following sections describe this formulation and present a numerical solution for unweighted and weighted cases.

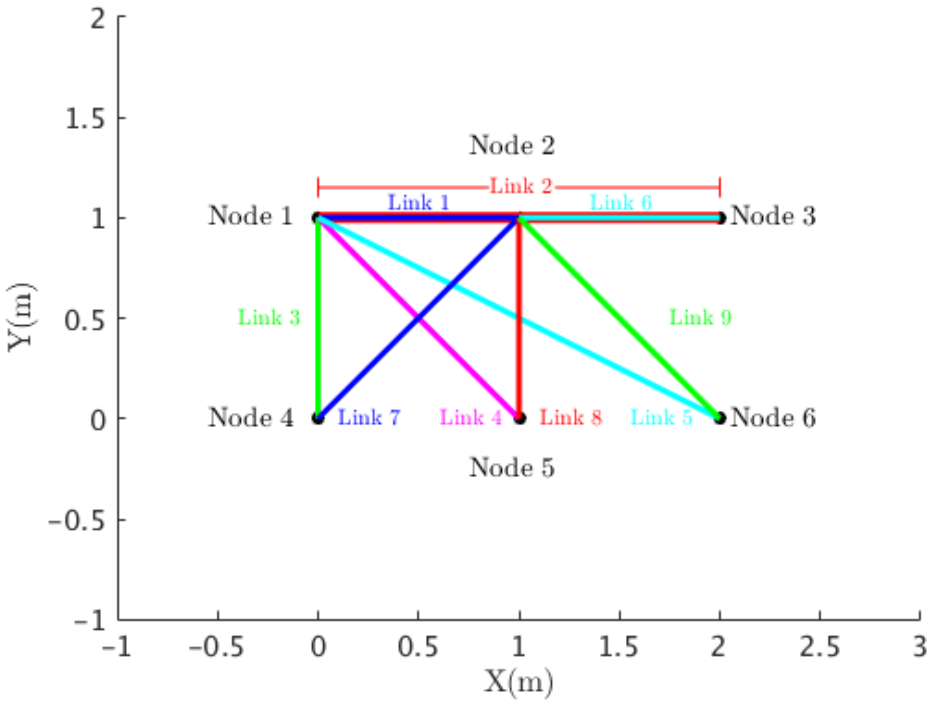

Figure 2: Truss Member and Node Numbering Convention

2 LP formulation

To begin formulating the LP, it is necessary to first understand the cost function and the constraints bounding the problem. As stated previously, the goal is to minimize the weight of the truss, which can be approximated by minimizing the number of members m required to support the applied load. Since each member used in the truss will have an internal force, u_i, we want to make as many equal to zero as possible, essentially removing them from the problem. This is a trait often provided by the l_1-norm, thus, for this problem we aim to minimize $||u_j||_1 = |u_1| + |u_2| + |u_3| + ... + |u_m|$. For this, we introduce a set of auxiliary variables, t_j, that we try to minimize

where

$$\begin{aligned} u_j &\leq t_j \\ -u_j &\leq t_j \end{aligned} \tag{1}$$

This can be rewritten as

$$\begin{bmatrix} \boldsymbol{I} & -\boldsymbol{I} \\ -\boldsymbol{I} & -\boldsymbol{I} \end{bmatrix} \hat{x} \leq \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{2}$$

where

$$\hat{x} = \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{t} \end{bmatrix} \tag{3}$$

In order to reduce the computational cost of this inequality, since the matrix will be $48,180 \times 48,180$, we use MATLAB's speye() function when constructing it. This creates a sparse identity matrix, which requires less memory to store.

We begin the LP by stating that we want to minimize $\hat{c}^T \hat{x}$, where

$$\hat{c} = \begin{bmatrix} \mathbf{0} & \mathbf{1} \end{bmatrix} \tag{4}$$

and x is given by Equation 3.

With the cost function defined, the constraints can now be set using the yield strength, S_y, of the material and equations of static equilibrium.

2.1 Yield Strength

The yield strength of the members determine when they will fail, so the stress at each member, σ_j (for j=$1 \cdots m$ bars), must not exceed S_y:

$$|\sigma_j| \leq S_y \tag{5}$$

where stress is the force experienced by the member, u_j, over the cross-sectional area A, which is assumed to be constant:

$$\sigma_j = \frac{u_j}{A} \tag{6}$$

Equations 5 and 6 can be rewritten as the following inequality:

$$-A * S_y \leq u_j \leq A * S_y \tag{7}$$

which can then be input into MATLAB's linprog() function as the variables' lower and upper bounds.

2.2 Static Equilibrium

The second set of constraints derives from static equilibrium, where the sums of the forces, in both the x and y directions, are equal to zero. Since this includes all the forces at each node, the first step is to list the Cartesian coordinates of every node, forming an $N \times 2$ array in numerical order, as described previously by the node-numbering convention. This list can then be used to determine the lengths, l_i, of every possible member by finding the differences between each point. This creates an $m \times 2$ array in the order described by the link-numbering convention. Since the internal forces may act in various directions, it is also important to account for the angles of each member. For computational purposes, instead of using trigonometric functions, it is useful to represent the directions of the members in unit vector notation:

$$\hat{m}_j = \frac{l_j}{||l_j||} \tag{8}$$

where $\hat{m}_j$ is the set of x and y directions in which the member is pointing. Recall that the convention does not repeat members and a node cannot connect to itself, so $N-1$ directions are recorded as acting on Node 1 but $N-2$ directions are recorded as acting on Node 2, $N-3$ directions are recorded as acting on Node 3, and so on.

With the angles defined, the equilibrium equations can now be established:

$$\sum_{j=1}^{m} u_j \hat{m}_i + f_i = \mathbf{0} \tag{9}$$

where f_i represents the external forces acting on the truss in both the x and y directions, of which there is only one in this case: F. This summation can be transformed into the form $\hat{A}\hat{x} = \hat{b}$, where $\hat{x}$ now includes the external force F and $\hat{b}$ is a length $2N$ column vector of zeros, except for the 120^{th} row, corresponding to the node at which F is applied. Force F only occupies one row because it is only applied in the y direction and its x-component is zero. These are shown below.

$$x = \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{t} \\ F \end{bmatrix} \tag{10}$$

$$\hat{b} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ -f_{120} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{11}$$

Generating the matrix $\hat{A}$ is slightly more complicated, since it consists of the directions, $\hat{m}_{ijx}$ and $\hat{m}_{ijy}$, that the truss members are aligned with at each node, where subscript i indicates the node number and subscript j provides the member number:

$$\hat{A} = \begin{bmatrix} \hat{\boldsymbol{m}}_{1,1x} & \hat{\boldsymbol{m}}_{1,2x} & \hat{\boldsymbol{m}}_{1,3x} & \dots & \hat{\boldsymbol{m}}_{1mx} \\ \hat{\boldsymbol{m}}_{1,1y} & \hat{\boldsymbol{m}}_{1,2y} & \hat{\boldsymbol{m}}_{1,3y} & \dots & \hat{\boldsymbol{m}}_{1my} \\ \hat{\boldsymbol{m}}_{2,1x} & \hat{\boldsymbol{m}}_{2,2x} & \hat{\boldsymbol{m}}_{2,3x} & \dots & \hat{\boldsymbol{m}}_{2,mx} \\ \hat{\boldsymbol{m}}_{2,1y} & \hat{\boldsymbol{m}}_{2,2y} & \hat{\boldsymbol{m}}_{2,3y} & \dots & \hat{\boldsymbol{m}}_{2,my} \\ \vdots & \vdots & & \ddots & \\ \hat{\boldsymbol{m}}_{N,1x} & \hat{\boldsymbol{m}}_{N,2x} & \hat{\boldsymbol{m}}_{N,2x} & \dots & \hat{\boldsymbol{m}}_{N,mx} \\ \hat{\boldsymbol{m}}_{N,1y} & \hat{\boldsymbol{m}}_{N,2y} & \hat{\boldsymbol{m}}_{N,2y} & \dots & \hat{\boldsymbol{m}}_{N,my} \end{bmatrix} \tag{12}$$

Notice that every two rows in $\hat{A}$ will represent the members attached to an individual node in the x and y directions, so some of the entries will be zero.

For the remaining discussion, in the interest of simplicity, we will assume the patterns in x are the same as those in y and will refer to matrix $\hat{A}$ and array $\hat{\boldsymbol{m}}$ as only containing one direction. To populate this through MATLAB using array $\hat{\boldsymbol{m}}$, it is necessary to recognize patterns in the lists we have developed so far. First, we notice that in $\hat{\boldsymbol{m}}$, the first $N-1$ rows correspond to Node 1, the following $N-2$ rows correspond to Node 2, the next $N-3$ rows correspond to Node 3, and so on. Since each row in $\hat{A}$ represents a node, populating this portion is trivial:

$$\hat{A} = \begin{bmatrix} \hat{\boldsymbol{m}}(1) & \hat{\boldsymbol{m}}(2) & \dots & \hat{\boldsymbol{m}}(N-1) & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & \hat{\boldsymbol{m}}(N) & \hat{\boldsymbol{m}}(N+1) & \dots & \hat{\boldsymbol{m}}(2N-2) & 0 & \dots & 0 \\ \vdots & \vdots & \dots & 0 & 0 & 0 & \dots & 0 & \ddots & & \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & \dots & \hat{\boldsymbol{m}}(m) \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{bmatrix} \tag{13}$$

The second characteristic to notice is that, due to Newton's third law of motion, each row has an entry the same as another in magnitude but opposite in sign. This means that the diagonal starting at $(2,1)$ is equivalent to the negative of the first $N-1$ rows in $\hat{\boldsymbol{m}}$. Similarly, the diagonal starting at $(3,N)$ is the negative of the next $N-2$ rows in $\hat{\boldsymbol{m}}$, the diagonal starting at $(3,2N-1)$ is the following $N-3$ rows, and so on. When included in $\hat{A}$, as shown below, this pattern gives each row $N-1$ nonzero entries.

$$\hat{A} = \begin{bmatrix} \hat{\boldsymbol{m}}(1) & \hat{\boldsymbol{m}}(2) & \dots & \hat{\boldsymbol{m}}(N-1) & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ -\hat{\boldsymbol{m}}(1) & 0 & \dots & 0 & \hat{\boldsymbol{m}}(N) & \hat{\boldsymbol{m}}(N+1) & \dots & \hat{\boldsymbol{m}}(2N-2) & 0 & \dots & 0 \\ \vdots & \ddots & \dots & 0 & 0 & 0 & \ddots & 0 & \ddots & & \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & \dots & \hat{\boldsymbol{m}}(m) \\ 0 & 0 & \dots & -\hat{\boldsymbol{m}}(N-1) & 0 & 0 & \dots & -\hat{\boldsymbol{m}}(N+1) & 0 & \dots & -\hat{\boldsymbol{m}}(m) \end{bmatrix} \tag{14}$$

Since we added an auxiliary variable to x in Equation 10 to form the inequality when minimizing $||u_j||_1$, we must also remember to take this into consideration for the equality here. Thus, we append a $2N \times m$ matrix of zeros to $\hat{A}$ so that the t_j variables do not get added into the equilibrium equations:

$$\hat{A} = [\hat{A} \quad \mathbf{0}] \tag{15}$$

We also need to remember that x and $\hat{b}$ include the force F, as shown in Equations 10 and 11, and this needs to be incorporated into both $\hat{A}$ and the inequality from Equation 2. To do this, we append a column of zeros to the matrices in both the equality and the inequality, as shown below.

$$\hat{A} = [\hat{A} \quad \mathbf{0}] \tag{16}$$

$$\begin{bmatrix} \boldsymbol{I} & -\boldsymbol{I} & \mathbf{0} \\ -\boldsymbol{I} & -\boldsymbol{I} & \mathbf{0} \end{bmatrix} x \leq \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{17}$$

To include F in the equilibrium equations in $\hat{A}$, we place a 1 in this appended column of zeros in the row corresponding to the node at which F is applied, which is the 120^{th} node. This is similar to the formation of $\hat{b}$ from Equation 11 previously. Since F is not a part of the inequality, though, we keep that appended column of zeros unchanged.

Lastly, we need to define the force F, since it is a constant. For this, we append a row of zeros to the end of $\hat{A}$ with a 1 in its final entry, such as $[0 \quad 0 \dots 0 \quad 1]$. This will, ultimately, establish $F = f_{120}$ where $-f_{120} = 4N$, as shown in the next section.

With this, we now have a fully defined LP:

$$\begin{aligned} &\text{minimize} && \hat{c}^T \hat{x} \\ &\text{subject to} && \hat{A}\hat{x} = \hat{b} \\ & && -A * S_y \leq u_j \leq A * S_y \\ & && \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{I} & \boldsymbol{0} \\ -\boldsymbol{I} & -\boldsymbol{I} & \boldsymbol{0} \end{bmatrix} x \leq \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix} \end{aligned} \tag{18}$$

where

$$\hat{c} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{1} \\ 0 \end{bmatrix} \tag{19}$$

and

$$\hat{x} = \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{t} \\ F \end{bmatrix} \tag{20}$$

It is possible to turn the equality constraints $\hat{A}\hat{x} = \hat{b}$ into two inequality constraints and transform the yield strength constraints into general form $\hat{A}\hat{x} \leq \hat{b}$ and then merge both with the other inequality constraints, but since the linprog() function in Matlab allows equality constraints and upper and lower bounds as an input, they will remain as they are. Through linprog(), we will use the dual simplex algorithm to solve the problem.

3 Numerical Results

Now that the LP formulation is finished, we will incorporate some initial values, as found in Table 1. With these constants, we can solve two variations of the weight problem: the first, disregarding the lengths of the truss members and the second, attempting to reduce the lengths of the members.

3.1 Unweighted Solution

The unweighted problem, in which we ignore the lengths of the members, is solved almost using exactly the LP described previously. One change made to the LP, though, is the removal of the anchor points. We assume that the anchors in the problem will never fail and can support any load they

Table 1: Constants For Truss Problems

Constant	Symbol	Value
Yield Strength	S_y	8 N/m^2
Area	A	1 m^2
Force	F	4 N

need to. That said, we recognize that there is no need to include equilibrium equations at the anchor points, in this case at Nodes 81, 101, and 121, so we remove them from both $\hat{A}$ and $\hat{b}$ in the equality constraints $\hat{A}\hat{x} = \hat{b}$.

After this, MATLAB's linprog() function can be used to solve the LP, which uses a dual simplex algorithm. Figure 3 illustrates the truss formed by minimizing the number of members, where the blue lines indicate a member in tension and the red lines indicate compression.

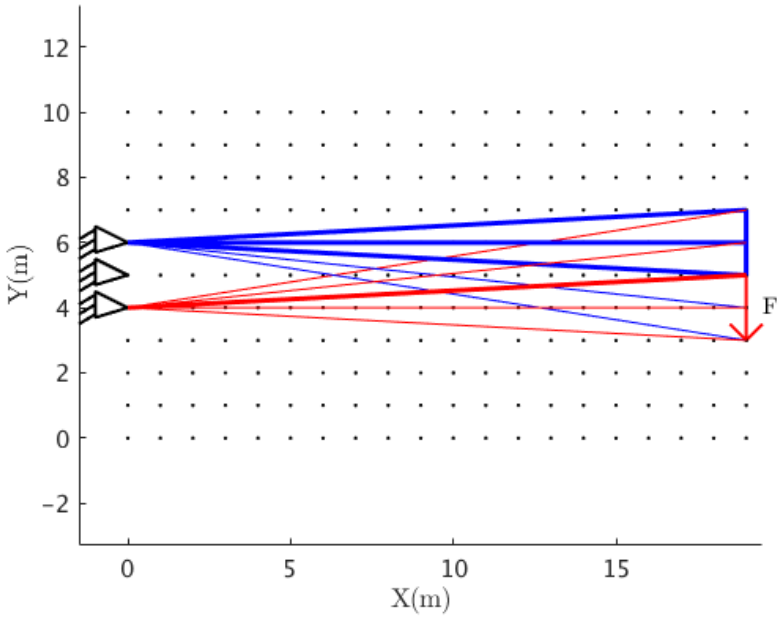

Figure 3: Unweighted Truss (blue=tension, red=compression)

This shows a successful reduction of the number of members from the possible 24,090 down to only 14. The forces experienced by these members are given below to show that they remain within the yield strength constraints. All other forces can be assumed to be zero and will not be listed.

$$u = \begin{bmatrix} 7.9130 \\ 0.8318 \\ -8.0000 \\ 7.9560 \\ 8.0000 \\ 8.0000 \\ 6.2736 \\ 0.8375 \\ -8.0000 \\ -8.0000 \\ -0.8375 \\ -0.6523 \\ -7.9560 \\ -6.2054 \end{bmatrix} \tag{21}$$

It is interesting to notice that the forces used are either relatively close to zero or to the yield strength. This makes sense when there are so few members to distribute the load. It is also interesting that the internal forces are not symmetric, and most of the force is being applied in tension.

Although this is a good minimization of the number of members, it may not be a manufacturable design, since most of them are over 19 meters long. To remedy this, we can weight the problem with the lengths of the links so that longer links are penalized.

3.2 Weighted Solution

By weighting the problem with the lengths of the members, we now want to minimize $l_j||u_j||_1 = l_1|u_1| + l_2|u_2| + l_3|u_3| + ... + l_m|u_m|$. This is only a minor altercation to the unweighted problem, where we just need to change the cost function $\hat{c}$. We merely replace the vector of ones in the unweighted

$\hat{c}$ with the list of lengths found earlier:

$$\hat{c}_{weight} = \begin{bmatrix} \mathbf{0} \\ \boldsymbol{l} \\ 0 \end{bmatrix} \tag{22}$$

Everything else remains the same as in the unweighted case and we, once again, use linprog() to solve the LP.

Unfortunately, the resulting $\hat{x}_{weight}$ contains several very small forces, so we need to decide what error is acceptable. This should, ultimately, be decided based on the factor of safety requirement, so we will just present several solutions generated using a range of acceptable errors. For a problem of this magnitude, where the maximum forces are 8N, it seems reasonable to assume that internal forces in the realm of $10^{-7}N$ and below are negligible, so we will set this as the error's lower bound. From there, we will increase the error until the design seems feasible. Figures 4 through 6 show how the design varies when setting any forces less than 10^{-7}, 10^{-6}, and 10^{-5} to zero.

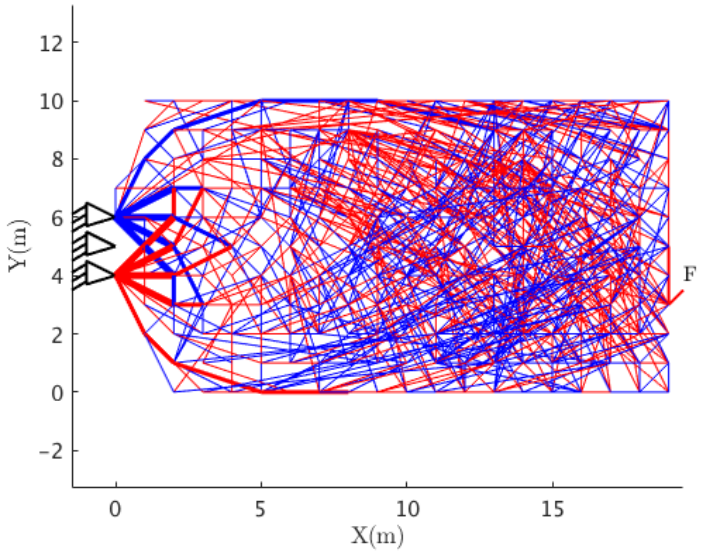

Figure 4: Weighted Truss After Replacing $u < 10^{-7}$N With 0

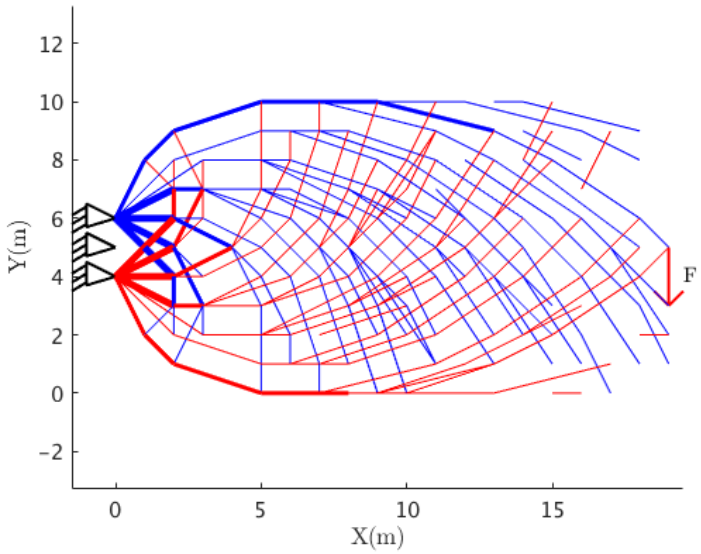

Figure 5: Weighted Truss After Replacing $u < 10^{-6}$N With 0

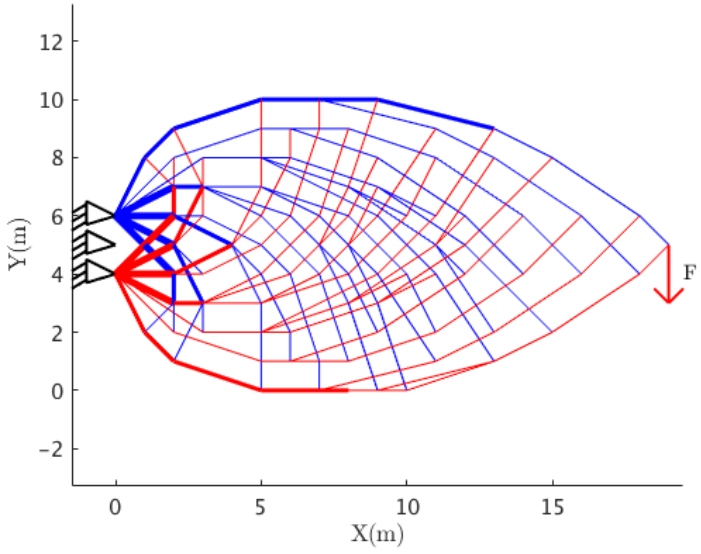

Figure 6: Weighted Truss After Replacing $u < 10^{-5}$N With 0

Again, the blue represents tension, red represents compression and the thicker lines indicate a larger force. Since Figures 4 and 5 show infeasible designs, we accept an error of $10^{-5}N$. This requires 187 members in a symmetric teardrop shape, which is still much less than the possible 24,090. Also, although there are more links than in the unweighted case, the maximum length of the members is only 4 meters, which is much shorter than the unweighted 19 meter links and, thus, can be more easily manufactured. Notice that the internal forces are more symmetric in this case than those of the unweighted, but

4 Conclusion

We presented the LP formulation for designing a truss that supports a given load while minimizing the number of members in the truss. Though it is not exact, minimizing the number of members is a decent approximation of minimizing the truss' weight. We found that without weighting the problem, the number of members required is reduced substantially, but they result in being extremely long. We observed that a simple manipulation, such as weighting with the member lengths, can significantly change the outcome, increasing the number of links but reducing their lengths. Additional variables in the future may be included to more completely attempt a weight minimization, such as link shape or material, but this displayed the application of linear programming on a real problem.

A Code

A.1 Setup

```
truss = [11,20]; % size of truss
N = truss(1)*truss(2); % number of nodes
m = nchoosek(N,2); % number of beams
area = 1; % surface area
Sy = 8; % yield strength
F = 4; % External force

% Positions using convention [1 2 3 4
%                             5 6 7 8 ....
```

```
Floc = 120; % position of the force
anchLoc = [81,101,121]; % position of anchors

% Truss grid
figure
hold on
plot([19 19],[5 3],'r','Linewidth',2)
plot([19 18.5],[3 3.5],'r','Linewidth',2)
plot([19 19.5],[3 3.5],'r','Linewidth',2)
plot([0 -1 -1 0],[6 5.7 6.5 6],'k','Linewidth',2)
plot([0 -1 -1 0],[5 4.7 5.5 5],'k','Linewidth',2)
plot([0 -1 -1 0],[4 3.7 4.5 4],'k','Linewidth',2)
plot([-1.5 -1],[3.5 3.8],'k','Linewidth',2)
plot([-1.5 -1],[3.8 4.1],'k','Linewidth',2)
plot([-1.5 -1],[4.1 4.4],'k','Linewidth',2)
plot([-1.5 -1],[4.5 4.8],'k','Linewidth',2)
plot([-1.5 -1],[4.8 5.1],'k','Linewidth',2)
plot([-1.5 -1],[5.1 5.4],'k','Linewidth',2)
plot([-1.5 -1],[5.5 5.8],'k','Linewidth',2)
plot([-1.5 -1],[5.8 6.1],'k','Linewidth',2)
plot([-1.5 -1],[6.1 6.4],'k','Linewidth',2)
for i = 1:truss(2)
for j = 1:truss(1)
plot(i-1,j-1, 'k.')
end
end
text(19.5, 4, 'F', 'interpreter', 'Latex', 'FontSize', 15, 'Color', 'k')
xlabel ('X(m)','interpreter','Latex')
ylabel ('Y(m)','interpreter','Latex')
set(gca,'FontSize',15)
set(gca,'FontName','cmr12')
axis equal

% Numbering convention example
figure
hold on
for i = 1:3
for j = 1:2
```

```
plot(i-1, j-1, 'k.', 'MarkerSize', 25)
end
end
plot([0 2], [1 1], 'r', 'Linewidth', 6)
plot([0 1], [1 1], 'b', 'Linewidth', 3)
plot([0 0], [1 0], 'g', 'Linewidth', 3)
plot([0 1], [1 0], 'm', 'Linewidth', 3)
plot([0 2], [1 0], 'c', 'Linewidth', 3)
plot([1 2], [1 1], 'c', 'Linewidth', 3)
plot([1 0], [1 0], 'b', 'Linewidth', 3)
plot([1 1], [1 0], 'r', 'Linewidth', 3)
plot([1 2], [1 0], 'g', 'Linewidth', 3)

text(-.55, 1, 'Node 1', 'interpreter', 'Latex', 'FontSize', 15, 'Color', 'k')
text(.75, 1.35, 'Node 2', 'interpreter', 'Latex', 'FontSize', 15, 'Color', 'k')
text(2.05, 1, 'Node 3', 'interpreter', 'Latex', 'FontSize', 15, 'Color', 'k')
text(-.55, 0, 'Node 4', 'interpreter', 'Latex', 'FontSize', 15, 'Color', 'k')
text(.75, -.25, 'Node 5', 'interpreter', 'Latex', 'FontSize', 15, 'Color', 'k')
text(2.05, 0, 'Node 6', 'interpreter', 'Latex', 'FontSize', 15, 'Color', 'k')

text(.35, 1.07, 'Link 1', 'interpreter', 'Latex', 'FontSize', 12, 'Color', 'b')
text(.85, 1.15, 'Link 2', 'interpreter', 'Latex', 'FontSize', 12, 'Color', 'r')
plot([0 .85], [1.15 1.15], 'r', 'Linewidth', 1)
plot([1.18 2], [1.15 1.15], 'r', 'Linewidth', 1)
plot([0 0], [1.1 1.2], 'r', 'Linewidth', 1)
plot([2 2], [1.1 1.2], 'r', 'Linewidth', 1)
text(-.4, .5, 'Link 3', 'interpreter', 'Latex', 'FontSize', 12, 'Color', 'g')
text(.6, 0, 'Link 4', 'interpreter', 'Latex', 'FontSize', 12, 'Color', 'm')
text(1.55, 0, 'Link 5', 'interpreter', 'Latex', 'FontSize', 12, 'Color', 'c')
text(1.35, 1.07, 'Link 6', 'interpreter', 'Latex', 'FontSize', 12, 'Color', 'c')
text(0.1, 0, 'Link 7', 'interpreter', 'Latex', 'FontSize', 12, 'Color', 'b')
text(1.1, 0, 'Link 8', 'interpreter', 'Latex', 'FontSize', 12, 'Color', 'r')
text(1.75, .5, 'Link 9', 'interpreter', 'Latex', 'FontSize', 12, 'Color', 'g')

xlabel ('X(m)','interpreter','Latex')
ylabel ('Y(m)','interpreter','Latex')
set(gca,'FontSize',15)
set(gca,'FontName','cmr12')
```

```
axis equal
xlim([-1 3])
ylim([-1 2])
```

A.2 Static Equilibrium

```
% Make a list of points
k = 1;
for i = 1:truss(1)
for j = 1:truss(2)
pointList(k,:) = [j-1,-(i-1)]; % switch i and j to make list [x,y]
k = k+1;
end
end

% Make a list of angles for Aeq and a list of lengths for plotting
k = 1;
mhat = [];
plotLen = [];
for i = 1:truss(1)
for j = 1:truss(2)
beamLen = pointList(k+1:end,:) - pointList(k,:);
mhat = [mhat; beamLen./vecnorm(beamLen')'];
k = k+1;
plotLen = [plotLen; beamLen];
end
end

% Set up equality constraint that the sum of the forces equals zero
Aeq = zeros(2*N, m);
ind2 = 0; % index 1
ind1 = 1; % index 2
inc2 = 1; % increment by 2
inc1 = 1; % increment by 1
for i = 1:N-1
ind2 = ind2+(N-1)-(i-1);
Aeq(inc2:inc2+1,ind1:ind2) = mhat(ind1:ind2,:)'; % Current node to other nodes
```

```
for j = 2*i+1:2:2*N           % Other nodes to current node
Aeq(j,inc1) = -mhat(inc1,1);
Aeq(j+1,inc1) = -mhat(inc1,2);
inc1 = inc1+1;
end
ind1 = ind1+(N-1)-(i-1);
inc2 = inc2+2;
end


beq = zeros(N*2,1);
```

A.3 $|u| < t$

```
%% Minimize ||u||_1 ==> |u|<t
Aineq = [speye(m) -speye(m);
-speye(m) -speye(m)];
bineq = [zeros(m,1); zeros(m,1)];
```

A.4 Develop LP

```
% Account for t in eq system
Aeq = [Aeq zeros(2*N,m)];

% External Forces
Aeq = [Aeq zeros(N*2,1)]; % Add external force to static equations
Aeq(2*Floc,end) = -1;
Aeq = [Aeq; zeros(1,m*2+1)]; % Specify value of external force
Aeq(end,end) = 1;
beq = [beq; F];

Aineq = [Aineq zeros(m*2,1)]; % account for force variable in inequality

% Remove anchors from static equation
Aeq([anchLoc(1)*2-1:anchLoc(1)*2, anchLoc(2)*2-1:anchLoc(2)*2,...
anchLoc(3)*2-1:anchLoc(3)*2],:) = [];
```

```
beq([anchLoc(1)*2-1:anchLoc(1)*2, anchLoc(2)*2-1:anchLoc(2)*2,...
anchLoc(3)*2-1:anchLoc(3)*2]) = [];


% Cost function
c = [zeros(1,m) ones(1,m) 0]';
```

A.5 Unweighted Case

```
% Solve the LP with yield strength upper and lower bounds (dual simplex)
x = linprog(c, Aineq, bineq, Aeq, beq,-area*Sy*ones(m,1), area*Sy*ones(m,1));
u = x(1:m); % pull out forces
beamCount = find(u); % find nonzero forces

% Plot
figure
hold on
plot([19 19],[5 3],'r','Linewidth',2)
plot([19 18.5],[3 3.5],'r','Linewidth',2)
plot([19 19.5],[3 3.5],'r','Linewidth',2)
plot([0 -1 -1 0],[6 5.7 6.5 6],'k','Linewidth',2)
plot([0 -1 -1 0],[5 4.7 5.5 5],'k','Linewidth',2)
plot([0 -1 -1 0],[4 3.7 4.5 4],'k','Linewidth',2)
plot([-1.5 -1],[3.5 3.8],'k','Linewidth',2)
plot([-1.5 -1],[3.8 4.1],'k','Linewidth',2)
plot([-1.5 -1],[4.1 4.4],'k','Linewidth',2)
plot([-1.5 -1],[4.5 4.8],'k','Linewidth',2)
plot([-1.5 -1],[4.8 5.1],'k','Linewidth',2)
plot([-1.5 -1],[5.1 5.4],'k','Linewidth',2)
plot([-1.5 -1],[5.5 5.8],'k','Linewidth',2)
plot([-1.5 -1],[5.8 6.1],'k','Linewidth',2)
plot([-1.5 -1],[6.1 6.4],'k','Linewidth',2)
for i = 1:truss(2)
for j = 1:truss(1)
plot(i-1,j-1, 'k.')
end
end
```

```
text(19.5, 4, 'F', 'interpreter', 'Latex', 'FontSize', 15, 'Color', 'k')
xlabel ('X(m)','interpreter','Latex')
ylabel ('Y(m)','interpreter','Latex')
set(gca,'FontSize',15)
set(gca,'FontName','cmr12')
axis equal

k = 1;
ind = 1;
for i = 1:N-1
if ind > length(beamCount)
break
end
k = k+(N-1)-(i-1);
while beamCount(ind) < k
% Change linewidth depending on magnitude of force
if abs(u_weight(beamCount_weight(ind))) < 3
linewidth = 1;
elseif abs(u_weight(beamCount_weight(ind))) < 6
linewidth = 3;
else
linewidth = 5;
end
if u(beamCount(ind)) > 0 % Tension
plot([pointList(i,1),pointList(i,1)+plotLen(beamCount(ind),1)],...
[pointList(i,2)+10,pointList(i,2)+plotLen(beamCount(ind),2)+10], 'b');
else % Compression
plot([pointList(i,1),pointList(i,1)+plotLen(beamCount(ind),1)],...
[pointList(i,2)+10,pointList(i,2)+plotLen(beamCount(ind),2)+10], 'r');
end
ind = ind+1;
if ind > length(beamCount)
break
end
end
end
```

A.6 Weighted Case

```
%% Length Weighting
c_weight = [zeros(1,m) vecnorm(plotLen') 0]'; % weight the problem with lengths
x_weight = linprog(c_weight, Aineq, bineq, Aeq, beq,-area*Sy*ones(m,1),...
area*Sy*ones(m,1));

u_weight = x_weight(1:m);
% Set any forces less than the error to zero
for error = [0.0000001, 0.000001, 0.00001]
beamCount_weight = find(u_weight);
k = 1;
for i = beamCount_weight'
if abs(u_weight(i)) < error
beamCount_weight(k) = [];
else
k = k+1;
end
end

% Plot
figure
hold on
plot([19 19],[5 3],'r','Linewidth',2)
plot([19 18.5],[3 3.5],'r','Linewidth',2)
plot([19 19.5],[3 3.5],'r','Linewidth',2)
plot([0 -1 -1 0],[6 5.7 6.5 6],'k','Linewidth',2)
plot([0 -1 -1 0],[5 4.7 5.5 5],'k','Linewidth',2)
plot([0 -1 -1 0],[4 3.7 4.5 4],'k','Linewidth',2)
plot([-1.5 -1],[3.5 3.8],'k','Linewidth',2)
plot([-1.5 -1],[3.8 4.1],'k','Linewidth',2)
plot([-1.5 -1],[4.1 4.4],'k','Linewidth',2)
plot([-1.5 -1],[4.5 4.8],'k','Linewidth',2)
plot([-1.5 -1],[4.8 5.1],'k','Linewidth',2)
plot([-1.5 -1],[5.1 5.4],'k','Linewidth',2)
plot([-1.5 -1],[5.5 5.8],'k','Linewidth',2)
plot([-1.5 -1],[5.8 6.1],'k','Linewidth',2)
plot([-1.5 -1],[6.1 6.4],'k','Linewidth',2)
```

```
axis equal
text(19.5, 4, 'F', 'interpreter', 'Latex', 'FontSize', 15, 'Color', 'k')
xlabel ('X(m)','interpreter','Latex')
ylabel ('Y(m)','interpreter','Latex')
set(gca,'FontSize',15)
set(gca,'FontName','cmr12')
k = 1;
ind = 1;
for i = 1:N-1
if ind > length(beamCount_weight)
break
end
k = k+(N-1)-(i-1);
while beamCount_weight(ind) < k
% Change linewidth depending on magnitude of force
if abs(u_weight(beamCount_weight(ind))) < 3
linewidth = 1;
elseif abs(u_weight(beamCount_weight(ind))) < 6
linewidth = 3;
else
linewidth = 5;
end

% Change color depending on direction of force
if u_weight(beamCount_weight(ind)) > 0 % Tension
plot([pointList(i,1),pointList(i,1)+...
plotLen(beamCount_weight(ind),1)], [pointList(i,2)+...
10,pointList(i,2)+plotLen(beamCount_weight(ind),2)+10],...
'b', 'LineWidth', linewidth);
else % Compression
plot([pointList(i,1),pointList(i,1)+...
plotLen(beamCount_weight(ind),1)],[pointList(i,2)+...
10,pointList(i,2)+plotLen(beamCount_weight(ind),2)+10],...
'r', 'LineWidth', linewidth);
end
ind = ind+1;
if ind > length(beamCount_weight)
break
```

```
end
end
end
end
maxLen = max(plotLen(beamCount_weight)); % Maximum member length in weighted case
```

References

[1] Shalom Ruben. Optimal design lecture, 2018.