Truss Construction through Linear Optimization

Max Ptasiewicz

March 26, 2022

1 Introduction

Numerical optimization can be used in many different circumstances, namely problems that are too complicated to solve by hand. Specifically, numerical optimization will be used for a truss topology problem, where we are searching for the optimal arrangement of beams to support a load.

For this problem, we started with the idea of creating an optimal truss to support a load, given a location of anchors and location and magnitude of the input forces. The optimal solution to find is the weight efficient solution, as that will use the smallest total length of beam, which will ultimately decrease the cost of the truss. In any truss example, there are several variables to consider: The beams might each have different lengths and cross sectional areas, there is some number of nodes, some number and location of anchors, and some number and location of applied forces.

In our problem, we are given an 11x20 grid of nodes to use, for a total of 220 nodes. There are three anchors on the middle three left side nodes, and an applied force on the far right. The grid is shown in figure 1.

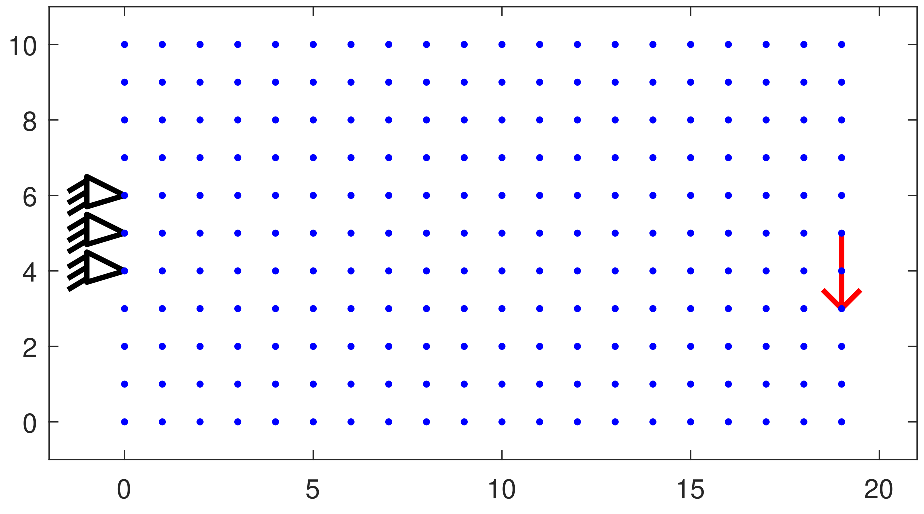

Figure 1: Given base grid on which to place a truss.

The next issue is to determine how many possible beams could exist on this grid. Our goal will be to assume that each possible beam can exist, however using the 1-norm we will try to make the force through as many as possible 0, effectively eliminating that beam. Each node could theoretically be connected to any of the other of the 220 nodes, which initially leads to an assumption of 220*219 possible beams. Upon further examination, this proves to double count many of the beams, as we would be counting a beam from node 1 to node 2 and from node 2 to node 1, which is the same beam. This leads to the logic of 219 connections from node 1, 218 from node 2, 217 from node 3, and so on. Studying smaller examples and creating a pattern from that, we come to the formula of:

$$C = \frac{n(n-1)}{2} \tag{1}$$

Where C represents the total number of connections, or beams in our case, and n is the total number of nodes. For our example of $n = 220$, we come to a total number of possible beams of 24090. This is way too large to think about solutions or even to visualize, so to start we take a much smaller problem with six nodes in a 2x3 grid, and solve that, and then use the formulation to solve the larger problem.

2 Formulation

There are several constraints that are present in our problem, as outlined earlier. Each beam has a cross sectional area, a length, and a yield strength. There are of course several other properties such as beam shape and other material properties like corrosion resistance and thermal properties. For our problem, we will make several simplifications:

- Each member is in either tension or compression only.
- The yield strength is the same for both tension and compression.
- The cross sectional area of each beam is the same, and is 1.
- All material properties such as corrosion resistance or thermal properties will be ignored.
- Strain will be ignored in the problem, so there will be no deformation before failure. As soon as the yield strength is reached, the beam will be assumed to break without displacement.

There are two main physical constraints to consider when solving this problem: The sum of forces at each node must be zero, and the maximum stress in each beam must be less than the yield strength. The second constraint is much easier to implement, as we can just use the following equation:

$$-S_y \leq \frac{u_i}{x_i} \leq S_y \tag{2}$$

Where S_y is the yield strength, u_i is each force through each beam, and x_i is the cross sectional area of each beam. In our example, that cross sectional area is the same for each beam, and the value is 1, so the equation can be simplified to:

$$-S_y \leq u_i \leq S_y \tag{3}$$

This can be turned into two inequality constraints, with variables on the left being less than or equal to constants on the right:

$$u_i \leq S_y \tag{4}$$

$$u_i \leq -S_y \tag{5}$$

Finally, we can turn this constraint into a linear programming form, using the following formulation:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} S_y \\ S_y \\ S_y \\ \vdots \\ S_y \\ -S_y \\ -S_y \\ -S_y \\ \vdots \\ -S_y \end{bmatrix}$$

For easy use later, lets call the stacked identity matrices A_{S_y} and the stacked S_y vectors b_{S_y}.

Next, we need to tackle the constraint of the sum of forces in each node being zero. First, we split the problem into x and y components, to balance using simple force analysis. In vector equation form, this comes out to the following equation:

$$\sum_{i=1}^{m} u_i \begin{bmatrix} cos(\theta_{ij}) \\ sin(\theta_{ij}) \end{bmatrix} + \begin{bmatrix} f_{x_i} \\ f_{y_i} \end{bmatrix} = 0 \tag{6}$$

This translates to every force going into any single node, along with any applied force on that node sums to 0. In order to determine how to turn this into a linear programming format, we need to figure out a pattern for each node being accounted for correctly. This started by writing out each static equilibrium equation for each node using a smaller example. The grid used for the test example is a 2x3 grid, for a total of 6 nodes and 15 beams. This resulted in the following equations:

$$\begin{gathered}
u_1 cos(\theta_1) + u_2 cos(\theta_2) + u_3 cos(\theta_3) + u_4 cos(\theta_4) + u_5 cos(\theta_5) = 0 \\
u_1 cos(\theta_1) + u_6 cos(\theta_6) + u_7 cos(\theta_7) + u_8 cos(\theta_8) + u_9 cos(\theta_9) = 0 \\
u_2 cos(\theta_2) + u_6 cos(\theta_6) + u_{10} cos(\theta_{10}) + u_{11} cos(\theta_{11}) + u_{12} cos(\theta_{12}) = 0 \\
u_3 cos(\theta_3) + u_7 cos(\theta_7) + u_{10} cos(\theta_{10}) + u_{13} cos(\theta_{13}) + u_{14} cos(\theta_{14}) = 0 \\
u_4 cos(\theta_4) + u_8 cos(\theta_8) + u_{11} cos(\theta_{11}) + u_{13} cos(\theta_{13}) + u_{15} cos(\theta_{15}) = 0 \\
u_5 cos(\theta_5) + u_9 cos(\theta_9) + u_{12} cos(\theta_{12}) + u_{14} cos(\theta_{14}) + u_{15} cos(\theta_{15}) = 0
\end{gathered}$$

The same pattern would repeat using sines, to force the y components to be 0. This initially looked like a jumbled mess to me, so I decided to rearrange it into a matrix where each row corresponded to a specific force equation, and each column represents a force in a beam. This resulted in the following matrix-vector equation, with a 1 in place of each cosine or sine value for easy viewing.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

This matrix of ones is multiplied by the cosine of each θ for each beam, where θ represents the angle between the left side of the beam and horizontal. Each column is multiplied by its corresponding θ, and then a second identical matrix is created, except with sine values.

This does not currently account for the anchors or applied forces, but that can be fixed easily. To account for the anchors, the solution is to simply delete the row of the matrix and zero vector corresponding to the equilibrium equation for an anchor. This is because the reaction force is assumed to be enough to support the truss, so there is effectively no constraint regarding the anchors. The applied forces are added to the zero vector, simply as the applied force in the correct location. In our problem, which has an applied force of four newtons downward on the 215th node in the 11x20 grid, this is simply a -4 in the 215th row of the vector with the results of the y component equations.

These final matrices will be called A_{cos} and A_{sin}, and the resultant force vectors will be called b_{cos} and b_{sin} for later use. In the later formulation, these matrices and vectors will be scaled up from $n = 6$ to $n = 220$ for the large truss. The code to create each of them is as follows, where the variables x_{dots} and y_{dots} are the x and y locations of each node.

```
A_trig = zeros(numel(x_dots),total_beams);
column_num = 1;

for m = 1:(numel(x_dots) - 1)
    A_trig(m,column_num:(numel(x_dots) - m + column_num - 1)) =
    -ones(size(column_num:(numel(x_dots) - m + column_num - 1)));
    A_trig((m + 1):end,column_num:(numel(x_dots) - m + column_num - 1)) =
    eye(numel(x_dots) - m);
    column_num = column_num + numel(x_dots) - m;
end

A_cos = A_trig;
A_sin = A_trig;

for n = 1:total_beams
    A_cos(:,n) = A_cos(:,n)*cos(beam_angles(n));
    A_sin(:,n) = A_sin(:,n)*sin(beam_angles(n));
```

```
end

b_cos = zeros(numel(x_dots),1);
b_sin = zeros(numel(x_dots),1);
b_sin(215) = -4;

for p = 1:numel(x_dots)
    if p == 5
        b_cos(p:(p + 2)) = [];
        b_sin(p:(p + 2)) = [];
        A_cos(p:(p + 2),:) = [];
        A_sin(p:(p + 2),:) = [];
    end
end
```

This is a problem with this constraint, and that's because a linear program formulation needs to be made of inequality constraints, and this is an equality constraint. In MATLAB, linprog allows for both inequality and equality constraints, so technically this could still work with the solver being used, however I chose to reformulate this into an inequality constraint. This is done by doubling the sets of equations and saying they are both less than and greater than a number, so the only option is for them to be equal to that number. This is done using the following formulation:

$$\begin{bmatrix} A_{cos} \\ A_{sin} \\ -A_{cos} \\ -A_{sin} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} b_{cos} \\ b_{sin} \\ -b_{cos} \\ -b_{sin} \end{bmatrix}$$

This will transform the equality constraint into an inequality constraint, which will allow us to shove everything into the same matrix vector equation, and into one linear programming line of code.

So far, I've talked about the lengths of each beam and the angles of each beam, and these are both values that need to be calculated for each possible beam. This is done by going through each possible connection by starting the inside for loop at the location of the iterator variable from the first for loop, so the inside loop decreases in size. Every time the loop goes to the next run, it creates points 1 and 2, to find the distance between them and to compute the inverse tan of the y distance divided by the x distance. This results in the length of each possible beam, as well as the angle away from horizontal, between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$. The cosine or sine can then be taken of these angles for inputting into the matrix. The code to create the vectors containing these values follows:

```
total_beams = (numel(x_dots)*(numel(x_dots) - 1))/2;

beam_lengths = zeros(total_beams,1);
beam_angles = zeros(total_beams,1);

counter_beams = 1;
```

```
for k = 1:(numel(x_dots) - 1)
    for l = (k + 1):numel(x_dots)
        p1 = [x_dots(k),y_dots(k)];
        p2 = [x_dots(l),y_dots(l)];
        tan_arg = (p2(2) - p1(2))/(p2(1) - p1(1));
        beam_angles(counter_beams) = atan(tan_arg);
        beam_lengths(counter_beams) = norm(p1 - p2);
        counter_beams = counter_beams + 1;
    end
end
```

There is one final constraint, to transform this problem into a 1-norm formulation, where we minimize the sum of the absolute values of each force, as opposed to the sum of the squares of each force. This is because the 1-norm causes the largest amount of zeros to appear in the solution set, and the more zeros, the fewer beams. This is done by adding a slack variable, called t, that will represent the maximum of u_i and $-u_i$, which is an inequality definition of the 1-norm. This results in the following set of equations:

$$\begin{array}{cc} u_1 \le t_1 & -u_1 \le t_1 \\ u_2 \le t_2 & -u_2 \le t_2 \\ \vdots & \vdots \\ u_n \le t_n & -u_n \le t_n \end{array}$$

Where n is the number of nodes in the truss. The problem with this setup is that t is a variable as well as u, so we must rearrange the equations slightly to move each variable to the left side:

$$\begin{array}{cc} u_1 - t_1 \le 0 & -u_1 - t_1 \le 0 \\ u_2 - t_2 \le 0 & -u_2 - t_2 \le 0 \\ \vdots & \vdots \\ u_n - t_n \le 0 & -u_n - t_n \le 0 \end{array}$$

Finally, we can formulate this into a matrix and vector equation as follows:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & -1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 & -1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & 0 & 0 & \cdots & 1 \\ -1 & 0 & 0 & \cdots & 0 & -1 & 0 & 0 & \cdots & 0 \\ 0 & -1 & 0 & \cdots & 0 & 0 & -1 & 0 & \cdots & 0 \\ 0 & 0 & -1 & \cdots & 0 & 0 & 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -1 & 0 & 0 & 0 & \cdots & -1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \\ t_1 \\ t_2 \\ t_3 \\ \vdots \\ t_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Since this is just a set of identity matrices and zero vectors, there won't be any specific names for these matrices.

Now the formulation comes to stacking each of the matrix equations we've created on top of each other, to create the final matrix and vector to give to linprog. The resulting matrix vector equation is as follows:

$$\begin{bmatrix} A_{S_y} & 0 \\ A_{cos} & 0 \\ A_{sin} & 0 \\ -A_{cos} & 0 \\ -A_{sin} & 0 \\ I & -I \\ -I & -I \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \\ t_1 \\ t_2 \\ t_3 \\ \vdots \\ t_n \end{bmatrix} = \begin{bmatrix} b_{S_y} \\ b_{cos} \\ b_{sin} \\ -b_{cos} \\ -b_{sin} \\ 0 \\ 0 \end{bmatrix}$$

This matrix will be called A_1 and the resultant vector will be called b_1, for the 1-norm formulation of this problem. There are several matrices of zeros added to the right of the initial constraint matrices, as there was no instance of the t variable in any of those equations, so the t was zeroed out. The last thing to do is to create two c vectors for a problem where we're first simply optimizing for weight, and secondly for optimizing for weight while penalizing the length of the beam. The cost function for the equation without penalizing length is just zeros followed by ones, as we are only minimizing the t values, so we want to ignore the u values in the final cost function, as long as they satisfy the constraints. In the second cost function, instead of ones, we use the length of each beam. This means that the longer the beam is, the more of an impact it will have on the cost function, so it is advantageous to use smaller beams. This helps with the real world analogy, as in reality longer beams will be harder to procure, and there will be more of an error due to strain with longer beams.

3 Analysis

After all of the formulation was done, which consisted of several concatenations, the next step is to plug everything into linprog and plot the results. Initially, I plotted the resulting truss that did not penalize longer length beams. This appeared as a long and skinny truss that most likely wouldn't work in reality, if the beams were large and made of steel.

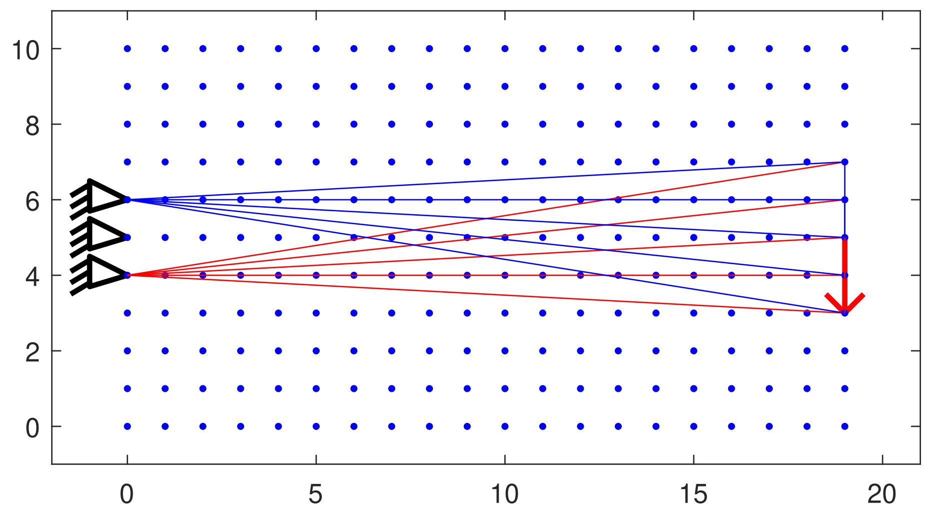

Figure 2: This is the result of a linear programming algorithm done that does not penalize lengths. Red represents beams in compression, and blue represents beams in tension.

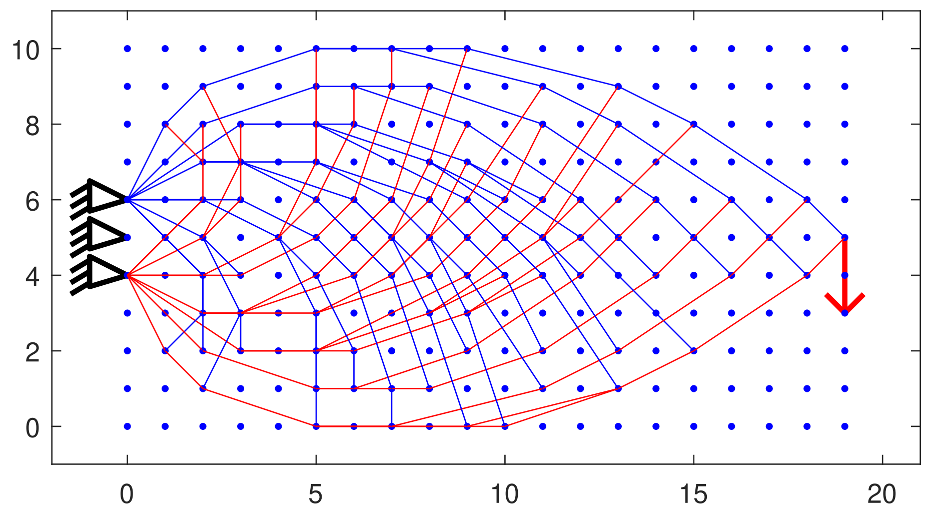

Figure 3: This is the result of a linear programming algorithm done that does penalize lengths. Red represents beams in compression, and blue represents beams in tension.

There were only minimal issues with the final results of this algorithm. There were none with the algorithm that did not penalize lengths, but there were several values in the results of the algorithm that does penalize length. These issues were several values extremely close to zero that should be zero in a real life example. The value of the stress going through these beams was negligible, as compared to their theoretical weight in real life. For this reason, these extra beams were eliminated, and the resulting image is almost identical to the solution on the class website.

4 Conclusion

Once the formulation of the project was done on paper, the hard part was over, and the only thing left to do is the final calculation. The algorithm and plotting took around 12 minutes to run, for both variations of the algorithm together. This is likely a function of my computer's lack of processing power, as it is fairly slow when it comes to difficult computation. The number of beams used in each solution was particularly interesting; it was only 15 for the solution without penalizing lengths, and only increased to 184 in my solution for penalized lengths. This is staggering compared to the total number of possible beams, which was 24090, however the optimal solution used an extremely minimal amount of them. This is a product of the 1-norm making as many zeros as possible in the solution vector. In the future, there could be further constraints added to the grid to optimize for different real life situations, such as variables that could relate the force through each beam to corrosion resistance or thermal properties. Additionally, the current constraints could be changed to create any situation desired.

5 Code

```
clc
clear
close all

tic

S_y = 8;

figure(1)
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
xlim([-2 21])
ylim([-1 11])
box on

x_dots = zeros(1,11*20);
y_dots = zeros(1,11*20);

for i = 0:19
    for j = 0:10
```

```
        x_dots(i*11 + j + 1) = i;
        y_dots(i*11 + j + 1) = j;
    end
end

scatter(x_dots,y_dots,7,'filled','b')

% lengths
total_beams = (numel(x_dots)*(numel(x_dots) - 1))/2;

beam_lengths = zeros(total_beams,1);
beam_angles = zeros(total_beams,1);

counter_beams = 1;

for k = 1:(numel(x_dots) - 1)
    for l = (k + 1):numel(x_dots)
        p1 = [x_dots(k),y_dots(k)];
        p2 = [x_dots(l),y_dots(l)];
        tan_arg = (p2(2) - p1(2))/(p2(1) - p1(1));
        beam_angles(counter_beams) = atan(tan_arg);
        beam_lengths(counter_beams) = norm(p1 - p2);
        counter_beams = counter_beams + 1;
    end
end

A_trig = zeros(numel(x_dots),total_beams);
column_num = 1;

for m = 1:(numel(x_dots) - 1)
    A_trig(m,column_num:(numel(x_dots) - m + column_num - 1)) =
    -ones(size(column_num:(numel(x_dots) - m + column_num - 1)));
    A_trig((m + 1):end,column_num:(numel(x_dots) - m + column_num - 1)) =
    eye(numel(x_dots) - m);
    column_num = column_num + numel(x_dots) - m;
end

A_cos = A_trig;
A_sin = A_trig;

for n = 1:total_beams
    A_cos(:,n) = A_cos(:,n)*cos(beam_angles(n));
    A_sin(:,n) = A_sin(:,n)*sin(beam_angles(n));
end

b_cos = zeros(numel(x_dots),1);
```

```
b_sin = zeros(numel(x_dots),1);
b_sin(215) = -4;

for p = 1:numel(x_dots)
    if p == 5
        b_cos(p:(p + 2)) = [];
        b_sin(p:(p + 2)) = [];
        A_cos(p:(p + 2),:) = [];
        A_sin(p:(p + 2),:) = [];
    end
end

A_both = sparse([A_cos; A_sin]);

A_add = speye(total_beams);

b_8 = [b_cos; b_sin];
b_15 = S_y*ones(total_beams,1);

b_1 = [b_8; -b_8; b_15; b_15; zeros(2*total_beams,1)];

A_1 = [A_both, zeros(numel(b_8),total_beams);
       -A_both, zeros(numel(b_8),total_beams);
       A_add, zeros(size(A_add));
       -A_add, zeros(size(A_add));
       A_add, -A_add;
       -A_add, -A_add];

c_nolengths = [zeros(total_beams,1); ones(total_beams,1)];

c_lengths = [zeros(total_beams,1); beam_lengths];

u_test = linprog(c_nolengths,A_1,b_1);

u_use = u_test(1:total_beams);

u_test_lengths = linprog(c_lengths,A_1,b_1);

u_use_lengths = u_test_lengths(1:total_beams);

counter_plot = 1;
num_beams_nolengths = 1;

for q = 1:(numel(x_dots) - 1)
    for r = (q + 1):numel(x_dots)
        xs = [x_dots(q),x_dots(r)];
        ys = [y_dots(q),y_dots(r)];
        if abs(u_use(counter_plot)) > 1e-4
            if u_use(counter_plot) < 0
                plot(xs,ys,'r')
            elseif u_use(counter_plot) > 0
                plot(xs,ys,'b')
            end
            num_beams_nolengths = num_beams_nolengths + 1;
        end
        counter_plot = counter_plot + 1;
    end
end

hold off

figure(2)
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
xlim([-2 21])
ylim([-1 11])
box on

scatter(x_dots,y_dots,7,'filled','b')

counter_plot_lengths = 1;
num_beams_lengths = 1;

for u = 1:(numel(x_dots) - 1)
    for v = (u + 1):numel(x_dots)
        xs = [x_dots(u),x_dots(v)];
        ys = [y_dots(u),y_dots(v)];
        if abs(u_use_lengths(counter_plot_lengths)) > 1e-4
            if u_use_lengths(counter_plot_lengths) < 0
                plot(xs,ys,'r')
            elseif u_use_lengths(counter_plot_lengths) > 0
                plot(xs,ys,'b')
            end
            num_beams_lengths = num_beams_lengths + 1;
        end
        counter_plot_lengths = counter_plot_lengths + 1;
    end
end

toc
```