Maximum Load Determination of a Truss by Linear Programming

Riley Kenyon

03/16/2020

Abstract

In this report linear programming will be used to determine the optimal solution to a truss topology problem based on a variety of constraints and cost functions. The final solution yields the geometry that best minimizes the length of individual beams while considering the internal forces of the truss members.

1 Introduction

Optimization has applications for a variety of problems. When evaluating possible solutions to a problem, it is usually required to determine the one which best meets a given criteria. Mathematically, the criteria for which solutions are evaluated are cost functions. The cost function is a user defined requirement that produces an optimal solution by varying the problems' parameters. In this report, a structural problem is proposed to be optimized according to cost functions related to minimizing the overall internal forces of the truss members. In general, a cost function could be derived to minimize length (weight), incorporate restricted availability of certain lengths, or weigh specific node locations according to preference. In essence as long as the cost function can be mathematically defined, it can be a criteria for determining the optimal solution. In conjunction with the cost function there are parameters that describe the system; these are known as constraints. The constraints determine the limits of the system and describe what parameter bounds produce acceptable solutions. In linear programming, a method

used to solve linear problems, the combination of constraint and criteria are written in the form

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \end{array} \tag{1}$$

where the first equation represents the cost function, and the second states the problem is "subject to" the constraints. This form in particular is representative of equality constraints, however the constraints can also described by an inequality.

1.1 Problem Statement

The truss system proposed has a footprint of $m = 11$ nodes in height and $n = 20$ nodes in width. The grid of possible solutions is shown in figure 1 with a force applied at the location [5,19] with a magnitude of 4. The cross section of the truss members are assumed to have a unit area and subject to a yield strength of 8. The mounting locations for the beams to connect structurally to a foundation are located at points [4,0], [5,0], and [6,0], located on the leftmost possible nodes.

2 Stress Formulation

Using the information provided, there are $\binom{220}{2}$ combinations, yielding 24,090 possible truss elements. This is the number of non-repeated link combinations connecting two nodes on the grid shown in figure 1. One of the systems' constraints are in the form of the yield strength of the material. The stress in each element can be described by equation 2.

$$\sigma_i = \frac{u_i}{A} \tag{2}$$

The material is assumed to be uniform in compression and tension, therefore the constraint for any given stress σ_i is described by

$$|\sigma_i| \leq S_y \tag{3}$$

where S_y is equal to a stress of 8. Each element can, due to possessing unit area, have a maximum internal force equal in magnitude to the yield strength. The maximum force bounds the upper and lower limits of the

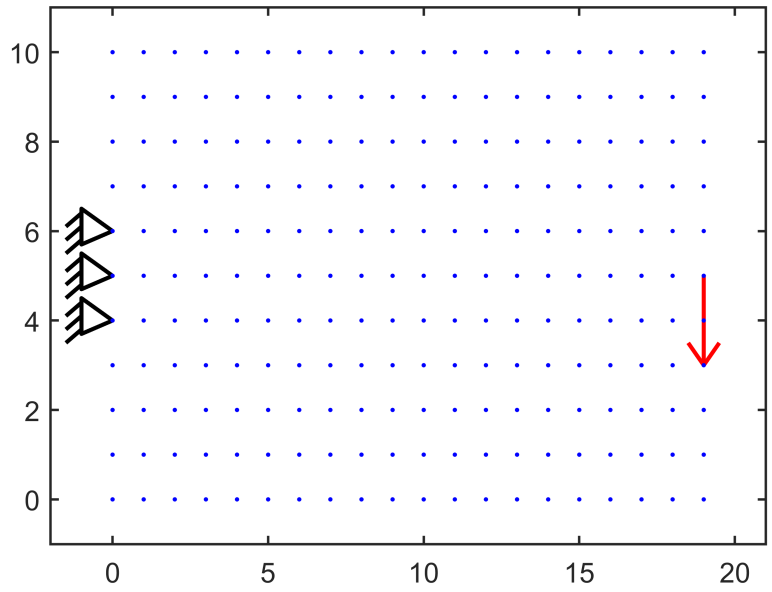

Figure 1: Example node grid consisting of 11 rows and 20 columns with anchors centered on the left edge.

internal forces. Another representation of equation 3, particularly useful in constructing matrix equivalents is shown in equation 4.

$$\begin{aligned} u_i &\leq S_y A \\ -u_i &\leq S_y A \end{aligned} \tag{4}$$

This requirement is the simplest of the constraints present in this problem, the matrix equivalent of the positive section of equation 4 is the form

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ & & \ddots & & \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \mathbf{x} \leq \begin{bmatrix} S_y A \\ S_y A \\ \vdots \\ S_y A \end{bmatrix} \tag{5}$$

where x is a vector of length 24,090 containing all the u_i. The matrix preceding the vector x is an identity matrix, represented by a bold-faced $\mathbf{I}$. To accommodate the full constraints of an absolute value, the bounds can be represented as

$$\begin{bmatrix} \mathbf{I_k} \\ -\mathbf{I_k} \end{bmatrix} \mathbf{x} \leq \begin{bmatrix} \mathbf{S_y A} \\ -\mathbf{S_y A} \end{bmatrix} \tag{6}$$

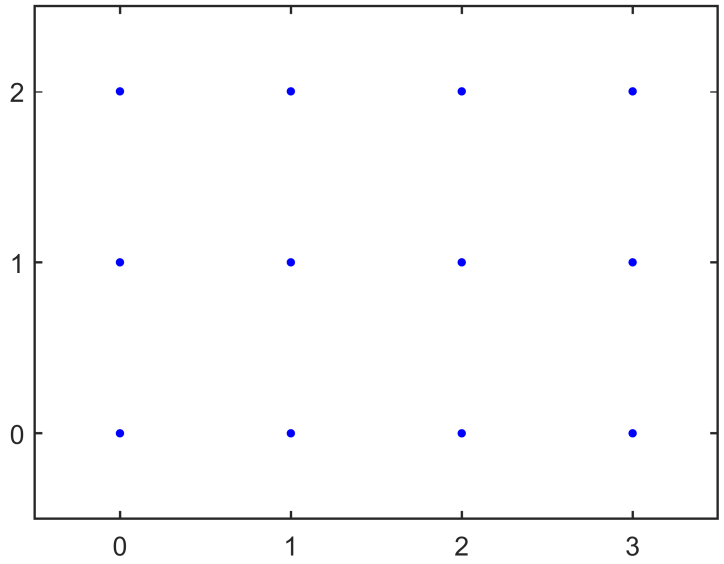

Figure 2: Example grid of spacing one with 3 rows and 4 columns for a total of 12 nodes. This example grid shows how the formulation of the force balance matrices are calculated.

where k denotes the total number of permutations for 220 nodes with a truss connecting two nodes.The constraints are vectors of length k of magnitude S_yA. This representation will be used as part of the inequality constraints in the linear program.

3 Force Balances

The next section is to determine how to describe the reaction forces in terms of the angle of their connection. In order to determine how the forces will interact with one another, a matrix of the angles will be created. Take the 3x4 grid shown in figure 2, there are a total of 12 nodes.

3.1 Angular Relation

The length and angle of each beam can be determined by what row and column the starting and ending nodes are. To accomplish this, two fundamental

matrices are created related to the 'x' and 'y' position of each node. The resulting x matrix consists of the elements

$$x_{pos} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \end{bmatrix} \tag{7}$$

starting at an index of 1, each column of the matrix has the same value. The y matrix consist of the elements

$$y_{pos} = \begin{bmatrix} 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{8}$$

where the rows of the matrix are the same value and increment from the bottom row. For this to be useful, each matrix will be replicated in both dimensions to create a matrix of size (nm x nm), in this case 12x12. The format of the matrix is shown below

$$x_{Big} = \begin{bmatrix} x_{mn}^T \\ \vdots \\ x_2^T \\ x_1^T \end{bmatrix} \tag{9}$$

Where x_i^T denotes the rows of the expanded x_{pos} matrix. The equivalent pattern applies for the y matrix but with respect to the columns of the matrix.

$$y_{Big} = \begin{bmatrix} y_1 & y_2 & \cdots & y_{mn} \end{bmatrix} \tag{10}$$

where y_i denotes the columns of the expanded y_{pos} matrix. By using this approach, the spacial difference between an individual node and the rest of the nodes can be written.

To create the first row of the x difference matrix, the value of the first x position will be subtracted from the rest of the row. This process continues until all nodes are accounted for. Mathematically, this is seen in equation

11.

$$\Delta x = \begin{bmatrix} x_{mn}^T - (n-1) \\ \vdots \\ x_8^T - 3 \\ \vdots \\ x_5^T - 0 \\ x_4^T - 3 \\ x_3^T - 2 \\ x_2^T - 1 \\ x_1^T - 0 \end{bmatrix} \tag{11}$$

The difference in y position can be determined through a similar process where each column vector is subtracted from the y-value for each of the 12 nodes. Mathematically, this can be described by

$$\Delta y = \begin{bmatrix} y_1 & y_2 - 1 & \dots & y_4 & y_5 - 1 & \dots & y_8 - 1 & \dots & y_{mn} - (m-1) \end{bmatrix} \tag{12}$$

Using the inverse tangent, the angle for each node can be determined by taking evaluating element-wise the function for the quantity of $\frac{\Delta y}{\Delta x}$. The theta matrix can then be used to describe the relation of any given node to any of the other nodes (i.e. lookup table for a start node as row and end node as column). Using the theta matrix, it is now feasible to construct a system of equations that describe the force balance on each truss member.

3.2 Equilibrium Conditions

For a truss in equilibrium, it is necessary that all the forces on the joint in the vertical and horizontal directions sum to zero. Although this also applies for the mounting nodes, it is assumed the mounting locations are able to exert any required restoring force to maintain equilibrium and are not explicitly bounded in terms of constraints. Decomposing a force into the subsequent horizontal and vertical components is graphically represented in figure 3. For an example truss, the force on the member can be described as

$$\begin{aligned} \sum_i^{mn} u_i \sin\theta_{ij} = 0 \\ \sum_i^{mn} u_i \cos\theta_{ij} = 0 \end{aligned} \tag{13}$$

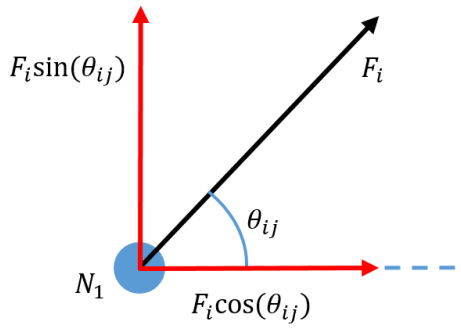

Figure 3: Example force balance to show how the force of a truss can be broken down into horizontal and vertical components.

where θ_{ij} represents the angle between node i and node j. Notice that the sine and cosine components correspond to the forces in the y direction and x direction respectively. It is assumed that the truss members act as freely movable joints, therefore moments are not considered in this analysis. Although simple in formulation, the method to create the matrix is less so.

Take the 3x4 sample grid mentioned in the previous section, the concept of permutation and combination can be described using the node connections. There are $(mn!)$ permutations of truss elements, however many of these correspond to existing links effectively creating redundant equations. Combinations represent the number of unique truss elements such that no single member is defined twice. This simplifies the number of unknowns to solve for and removes redundancy. Starting at node 1, the connections to the remaining nodes are all unique. However, upon initializing node 2, the connection from the node 2 to node 1 has already been defined. This means that the force of the link can be representing using u_1 and simply exchange the value of θ used in the formulation. The same follows for node 3, where the connection to node 1 can be represented as u_2 and the connection to node 2 can be represented as u_{12} (the first unique link of node two). The first equation of the system (describing node 1) looks like

$$u_1 \cos\theta_{1,2} + u_2 \cos\theta_{1,3} + \ldots + u_{11} \cos\theta_{1,12} = 0 \tag{14}$$

followed by the second equation describing node 2 is as follows

$$u_1 \cos\theta_{2,1} + u_{12} \cos\theta_{2,3} + \ldots + u_{21} \cos\theta_{2,12} = 0 \tag{15}$$

Notice that the direction of theta has flipped for the first element, however it is still referencing the same force as in node 1, u_1. The rest of the equation describes new forces. Manipulating this form of equations into a matrix results in the following.

$$\begin{bmatrix} \theta_{1,2} & \theta_{1,3} & \dots & \theta_{1,12} & 0 & \dots & 0 & 0 \\ \theta_{2,1} & 0 & \dots & 0 & \theta_{2,3} & \dots & \theta_{2,12} & 0 \\ & & & & \vdots & & & \\ 0 & 0 & \dots & \theta_{mn,1} & 0 & \dots & 0 & \theta_{mn,mn-1} \end{bmatrix} \tag{16}$$

Looking at this pattern closer, the major block can be written as constituents of other common blocks. The block below the top row is the identity matrix multiplied with the first column of the theta matrix. The process continues with smaller identities until the last column which is just a scalar one.

$$\begin{bmatrix} 0_{1,nm-1} & 0_{2,nm-2} & \dots & 0_{nm-1,1} \\ \theta_1 I_{nm-1} & \theta_2 I_{nm-2} & \dots & \theta_{nm-1,nm-1} \end{bmatrix} \tag{17}$$

In the array, θ_i corresponds to all the angles connecting to node i, or row i of the theta matrix. There also exists a block diagonal matrix that spans the matrix consisting of each nodes' unique truss members. These quantities decrease in size until the last element which is no longer unique and results in a zero.

$$\begin{bmatrix} \theta_1 & 0 & 0 & \dots & 0 \\ 0 & \theta_2 & 0 & \dots & 0 \\ & & \ddots & & \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix} \tag{18}$$

Now that the expanded matrix is defined, the entries of the upper section will consist of the sine of the elements of matrix A stacked on the cosine of the elements of matrix A, where A is defined as the sum of matrices in equation 17 and 18. Scaled up to $m = 11$ and $n = 20$, this surmounts to 220 base equations for the force balance relations. To allow for attachment to a supporting structure, the three nodes mentioned in the introduction will be removed, these correspond to rows 81, 101, and 121 from both the cosine and sine matrices that will be omitted. The result is a 434 x 24090 matrix that describes the equilibrium of any possible truss given the discrete node locations.

4 Structural Optimization

Most importantly, the success criteria is defined through a cost function. The three cases that will be evaluated are for minimum internal forces, minimum force weighed by considering length of truss element, and minimum length using only the upper half of the plane.

4.1 Augmenting Matrices

Now that the matrix for the load balances has been determined, it is important to consider what how the cost function will be evaluated. For all of the optimizations to be made the magnitude of the loads will be minimized, regardless of compression or tension. To accommodate this in the optimization, both the equality and inequality matrix will be augmented to incorporate slack variables. These slack variables will be used in the cost function rather than the values of the forces which are signed. For instance, take the equation

$$\begin{aligned} u_i &\leq |u_i| \\ -u_i &\leq |u_i| \end{aligned} \tag{19}$$

where u_i is the force within truss element i. If u_i is negative, the top equation evaluates less than the absolute value and the second equation is equal to the absolute value. Looking at the intersection of the two equations, if $|u_i|$ is replaced by a a variable t, the value of t will evaluate to the magnitude of the element u_i. This will be used in the inequality matrix to augment the structure and allow for the slack variables in the optimized solution. Ultimately the optimization will evaluate the cost function utilizing the magnitudes of each truss element, represented by the slack variable vector t. The new equality matrix can be described as

$$\hat{A}_{eq} = \begin{bmatrix} A & \mathbf{0} \end{bmatrix} \tag{20}$$

to not incorporate the slack variables when multiplying $\hat{A}_{eq}$ and the vector of x. The matrix A is the matrix that describes the equilibrium position of all the nodes both in the horizontal and vertical direction. The zero here is a zero matrix of size 434 x 24090. The constraints of this matrix are of the

form

$$\hat{b}_{eq} = \begin{bmatrix} \mathbf{0} \\ -F_y \\ \mathbf{0} \end{bmatrix} \tag{21}$$

where the force F_y is applied at node location [5,19] or in relation to flattened node position $(19+1)(5+1) = 120$. In addition, adjustments will be made to the inequality matrix to reflect addition of slack variables in the form of equation 19. If t is moved to the other side of the inequality, the inequality matrix can be represented as

$$\hat{A}_{ineq} = \begin{bmatrix} \mathbf{I_n} & -\mathbf{I_n} \\ -\mathbf{I_n} & -\mathbf{I_n} \\ \mathbf{0_n} & \mathbf{I_n} \end{bmatrix} \tag{22}$$

where n is equal to the total number of combinations. Notice the intuition from equation 6 is embedded into the last row of this matrix, where the absolute value of the force of a truss element is less than the yield strength of the material. The new matrices have corresponding constraints to obtain the final representation of the system.

$$\begin{bmatrix} \mathbf{I_n} & -\mathbf{I_n} \\ -\mathbf{I_n} & -\mathbf{I_n} \\ \mathbf{0_n} & \mathbf{I_n} \end{bmatrix} x \leq \begin{bmatrix} \mathbf{0_n} \\ \mathbf{0_n} \\ \mathbf{S_y A} \end{bmatrix} \tag{23}$$

4.2 Minimize Forces

The solution that minimizes the total amount of force, cumulative from each member can best be described by

$$\min |u_1| + |u_2| + ... + |u_k| \tag{24}$$

where the subscript k represents the total number of combinations. The sum of absolute values in the minimization is also known as the 1-norm $||u||_1$. The vector of truss forces is the vector produced through optimizing the system of equations, determined through matlab's command *linprog*(). Formally, the problem the optimizer will solve is as such

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & \hat{A}_{eq} x = \hat{b}_{eq} \\ & \hat{A}_{ineq} x \leq \hat{b}_{ineq} \end{array} \tag{25}$$

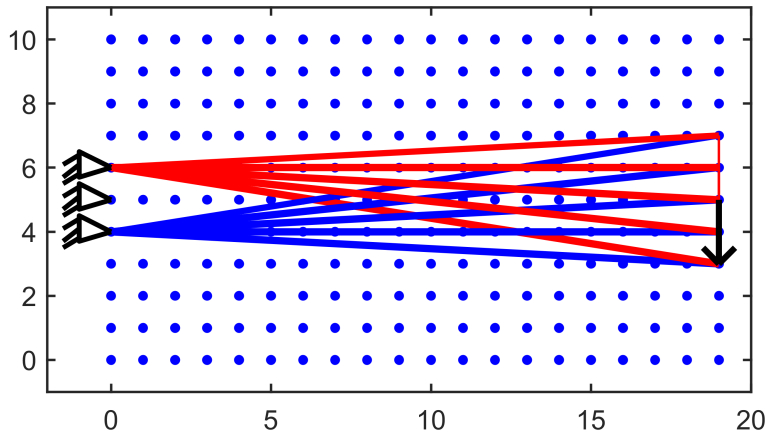

Figure 4: The solution to the optimization problem of minimizing the sum of force magnitudes to minimize the overall internal forces. The total number of links in this solution is 14.

where c is a column vector of ones, length 24090. $\hat{A}_{eq}$ and $\hat{b}_{eq}$ are defined based on the force balances outlined in the previous sections. These constraints were equalities, the truss system is required to be in equilibrium. The secondary constraints are that the values of x are bounded above and below by the threshold $S_y A$, these are represented in $\hat{A}_{ineq}$ and $\hat{b}_{ineq}$ which contain the matrices from equation 6. The resulting solution contains truss elements, and the structure of the elements is shown in figure 4

4.3 Minimize Length

The minimum length solution taking into consideration the force exerted on each truss member is similar to the previous solution for minimizing weight. The 1-norm will be used to evaluate the solution, however the weights will not be unitary as in the minimum force solution. The weights will correspond to the length of each element. This is obtained by taking the element-wise operation of $l_{ij} = \sqrt{\Delta x_{ij}^2 + \Delta y_{ij}^2}$. Flattening the matrix of lengths allows the corresponding cost function to be defined as

$$\min |l_1 u_1| + |l_2 u_2| + \ldots + |l_k u_k| \tag{26}$$

where c is a column vector of the lengths describing the force between nodes i and j for all possible truss members k. The optimal solution to this case

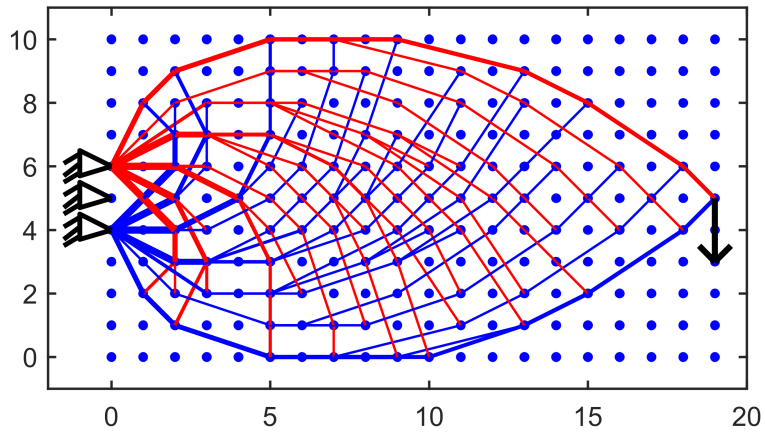

Figure 5: The solution here optimizes length while considering the internal forces of the nodes. The geometry is interesting and contains larger forces in elements closer to the mounting locations. The total number of elements in this configuration is 178 truss members.

is the most interesting, where the geometry mimics a teardrop and contains more heavily stressed elements closer to the mounting locations. The plot is shown in figure 5.

4.4 Minimize Length Constrained

In addition to the general cases for minimizing length and forces, an additional configuration is considered where truss elements can only be utilized for the upper half of the grid. The inspiration for this technique is inspired by the typical bridge geometry where truss elements typically are mounted above the bridge to allow for passage beneath it. In this scenario, to incorporate the reduction of node elements, the first half of the $\hat{A}_{eq}$ matrix will be removed for both sine and cosine components to represent only above ground nodes. The cost function will be the same as minimizing length, where the column vector c describes the lengths corresponding to the nodes used in this analysis. The outcome is seen in figure 6 where the geometry represents a curved oval structure as seen with a typical truss bridge, however it is optimized and cantilevered, so the internal structure deviates from reality but is similar. .

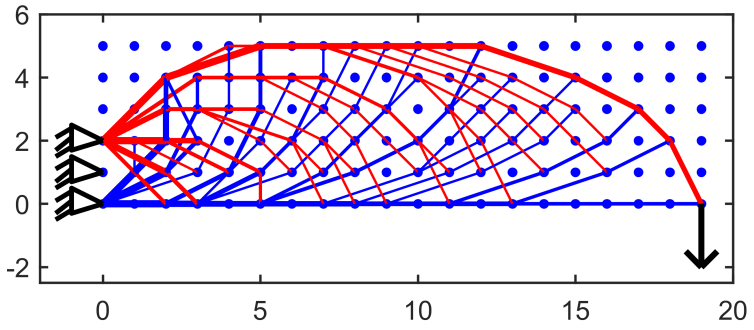

Figure 6: The optimal solution for nodes above a horizontal plane with mounting location located at indexes [1, 21, 41] and a force $F_y = 4$ applied at node 20. The total number of truss elements used for this configuration is 146 truss members.

5 Conclusion

In this structural analysis of various truss configurations, the node grid $m = 11$, $n = 20$ was used to represent the points where truss elements could connect. The setup was done considering the material limitations of the truss members, assumed uniform in tension and compression. This constraint was incorporated in the form of an inequality as shown in equation 6. The equality constraints of the linear program were determined through static analysis of the beams, the system was required to be in equilibrium. Each force contribution from node i to j was determined through the angle between the two nodes θ_{ij} and decomposing the force into horizontal and vertical components using trigonometry. The stacked matrix in total represented a system of 24090 possible elements, of which 14 were used to minimize overall force contributions and 178 to minimize length while considering the magnitude of the internal forces. As an additional configuration, the truss elements were constrained to the upper half plane of the node grid. The result was 146 links and a geometry resembling that of a typical truss bridge. Each solution was determined using matlab's linear programming optimizer *linprog* to constrain each system and solve using a user-defined cost function.

A Code

A.1 Main Function

```
%% Truss Project - Optimal Design
%--------------------------------------------------------------------------
% MEID: 272-513
% Riley Kenyon
% 03/13/2020
%--------------------------------------------------------------------------
clear all; close all; clc;

% Define Structure
truss.m = 11;
truss.n = 20;
truss.s_y = 8;
truss.F_y = 4;
middle = (truss.m+1)/2 - 1;
truss.point = (middle+1)*truss.n-2;
truss.anchors = [(middle-1)*truss.n+1,middle*truss.n+1,(middle+1)*truss.n+1];

% Run truss problem according to weight optimization
solveTruss('weight',truss);

% Run truss problem according to length optimization
solveTruss('length',truss);

% Run truss problem according to custom optimization
% solveTruss('custom',truss);
truss.m = 6;
truss.n = 20;
truss.s_y = 8;
truss.F_y = 4;
truss.point = truss.n-1;
truss.anchors = [1 21 41];
solveTruss('custom',truss);

% Example grid
```

```
truss.m = 3;
truss.n = 4;
truss.s_y = 8;
truss.F_y = 4;
truss.point = truss.n-1;
truss.anchors = [1];

plotTruss([],truss);
gcf;
xlim([-0.5, 3.5])
ylim([-0.5, 2.5])

% Get Example Theta
calculateTruss('run',truss)
```

A.2 Solving LP

```
function[] = calculateTruss(option,truss)
% Function to calculate the truss geometry and setup the matrices required
% to solve using linear programming.

% Parse Input
if ~isempty(truss)
    m = truss.m;
    n = truss.n;
    s_y = truss.s_y;
    F_y = truss.F_y;
    tot = nchoosek(n*m,2);
else
    disp('Incorrect Truss Input')
    return
end

% Set filenames and cost functions
switch option
    case 'weight'
        filename = 'weight_data.mat';
```

```
        C = [zeros(tot,1);ones(tot,1)]; % Cost function for pure force
    case 'length'
        filename = 'length_data.mat';
        coord = cell(n*m,1);
        end_coor = cell(n*m,1);
        for i = 1:n*m
            coord{i} = (i-1)*ones(1,n*m-i);
            end_coor{i} = (i:n*m-1);
        end
        coord = horzcat(coord{:});
        end_coor = horzcat(end_coor{:});
        x_coor = mod(coord,n);
        y_coor = floor(mod(coord/n,m));
        x_end = mod(end_coor,n);
        y_end = floor(mod(end_coor/n,m));
        l = sqrt((x_coor' - x_end').^2 + (y_coor' - y_end').^2);
        C = [zeros(tot,1);l];           % Create cost function for length
    case 'custom'
        filename = 'custom_data.mat';
        coord = cell(n*m,1);
        end_coor = cell(n*m,1);
        for i = 1:n*m
            coord{i} = (i-1)*ones(1,n*m-i);
            end_coor{i} = (i:n*m-1);
        end
        coord = horzcat(coord{:});
        end_coor = horzcat(end_coor{:});
        x_coor = mod(coord,n);
        y_coor = floor(mod(coord/n,m));
        x_end = mod(end_coor,n);
        y_end = floor(mod(end_coor/n,m));
        l = sqrt((x_coor' - x_end').^2 + (y_coor' - y_end').^2);
        C = [zeros(tot,1);l];           % Create cost function for length
case 'run'
        filename = 'example.mat';
        C = [zeros(tot,1);ones(tot,1)]; % Cost function for pure force

    otherwise
```

```
        disp('Incorrect Option Input')
        return
end

% Create matrix of x and y nodes
x_data = repmat([1:n],[m*n,m]);
y_data = repelem([1:m],m*n,n);

% Create indexes up to number of columns, in total mxn nodes
i = [1:m*n]';
i = mod(i-1,n) + 1;
j = repelem([1:m]',n,1);

% Determine differences between nodes for each row i
x_diff = x_data - repmat(i,[1,m*n]);
y_diff = y_data - repmat(j,[1,m*n]);

% Theta determined from x and y matrices
theta = atan2d(y_diff,x_diff);

% Determine sine and cosine of theta
theta_sin = sind(theta);                % Sin in degrees
theta_cos = cosd(theta);

% Create large matrix of values
for i = 1:n*m
   A_temp_sin{i} = theta_sin(i,[i+1:end]);        % Main diagonal of big matrix
   B_temp_sin{i} = [zeros(i,n*m-i);theta_sin([i+1:end],i).*eye(n*m-i)]; % Lower se
   A_temp_cos{i} = theta_cos(i,[i+1:end]);        % Main diagonal of big matrix
   B_temp_cos{i} = [zeros(i,n*m-i);theta_cos([i+1:end],i).*eye(n*m-i)]; % Lower se
end

% Top is sine contribution (F_y balance)
A_sin= blkdiag(A_temp_sin{:});
B_sin = horzcat(B_temp_sin{:});
A_top = A_sin+B_sin;
A_top([truss.anchors],:) = [];   % Remove support nodes
```

```
% Bottom is cosine contribution (F_x balance)
A_cos = blkdiag(A_temp_cos{:});
B_cos = horzcat(B_temp_cos{:});
A_bot = A_cos+B_cos;
A_bot([truss.anchors],:) = [];   % Remove support nodes

A_eq = [sparse([A_top;A_bot]),sparse(zeros(2*(n*m-length(truss.anchors)),tot))];
A_ineq = [sparse(eye(tot)),sparse(-eye(tot));sparse(-eye(tot)),sparse(-eye(tot));s

% Constraint matrix - sum of forces = 0, otherwise at point of applied
% force = -(f_y) for sine, and -(f_x) for cosine
b_eq = [zeros(2*(m*n-length(truss.anchors)),1)];
b_eq(truss.point) = F_y;
b_ineq = [zeros(2*tot,1);s_y*ones(tot,1)];

% Save output
save(filename,'C','A_eq','A_ineq','b_eq','b_ineq','theta','truss')

end

function [] = solveTruss(option,truss)
% Function for running linprog according to the option
switch option
    case 'weight'
        if ~isfile('weight_data.mat')
            calculateTruss(option,truss);
        end
        if ~isfile('weight_sol.mat')
            load weight_data.mat
            sol = linprog(C,A_ineq,b_ineq,A_eq,b_eq);
            save('weight_sol.mat','sol','truss')
        end
        load('weight_sol.mat')
        plotTruss(sol,truss);

    case 'length'
        if ~isfile('length_data.mat')
            calculateTruss(option,truss);
```

```
        end
        if ~isfile('length_sol.mat')
            load length_data.mat
            sol = linprog(C,A_ineq,b_ineq,A_eq,b_eq);
            save('length_sol.mat','sol','truss')
        end
        load('length_sol.mat')
        plotTruss(sol,truss);

    case 'custom'
        if ~isfile('custom_data.mat')
            calculateTruss(option,truss);
        end
        if ~isfile('custom_sol.mat')
            load custom_data.mat
            sol = linprog(C,A_ineq,b_ineq,A_eq,b_eq);
            save('custom_sol.mat','sol','truss')
        end
        load('custom_sol.mat');
        plotTruss(sol,truss);

    otherwise
        disp('Incorrect Input Option')
        return
end
end
```

A.3 Plotting

```
function[] =  plotTruss(sol,truss)
% Function for plotting the optimal solution
n = truss.n;
m = truss.m;

% Plot Grid
figure(), hold on
plot(repmat([0:n-1],m,1),repmat([0:m-1]',1,n),'b.','MarkerSize',12)
if isempty(sol)
```

```
   return
end

offset = floor(truss.anchors/truss.n);
mount = [0 -0.3 0.5 0];
arrow = floor(truss.point/truss.n);
u = sol(1:length(sol)/2);

% Plot Solution
for i = 1:n*m
    coord{i} = (i-1)*ones(1,n*m-i);
    end_coor{i} = [i:n*m-1];
end

coord = horzcat(coord{:});
end_coor = horzcat(end_coor{:});
x_coor = mod(coord,n);
y_coor = floor(mod(coord/n,m));
x_end = mod(end_coor,n);
y_end = floor(mod(end_coor/n,m));

% Need to do thresholding to get image to look right
for i = 1:length(u)
    if u(i) < -0.01
        if abs(u(i)) > 1
            w = 1 + abs(u(i))/4;
        else
            w = 1;
        end
        plot([x_coor(i);x_end(i)],[y_coor(i);y_end(i)],'b','LineWidth',w);
    elseif u(i) > 0.01
        if abs(u(i)) > 1
            w = 1 + abs(u(i))/4;
        else
            w = 1;
        end
        plot([x_coor(i);x_end(i)],[y_coor(i);y_end(i)],'r','LineWidth',w);
    end
```

```
end

% Plot Fixtures and auxillary information
plot([19 19],[0 -2]+arrow,'k','Linewidth',2.5)
plot([19 18.5],[-2 -1.5]+arrow,'k','Linewidth',2.5)
plot([19 19.5],[-2 -1.5]+arrow,'k','Linewidth',2.5)
plot([0 -1 -1 0],mount+offset(3),'k','Linewidth',2)
plot([0 -1 -1 0],mount+offset(2),'k','Linewidth',2)
plot([0 -1 -1 0],mount+offset(1),'k','Linewidth',2)
plot([-1.5 -1],[-0.5 -0.2]+offset(1),'k','Linewidth',2)
plot([-1.5 -1],[-0.3 0.1]+offset(1),'k','Linewidth',2)
plot([-1.5 -1],[0.1 0.4]+offset(1),'k','Linewidth',2)
plot([-1.5 -1],[-0.5 -0.2]+offset(2),'k','Linewidth',2)
plot([-1.5 -1],[-0.3 0.1]+offset(2),'k','Linewidth',2)
plot([-1.5 -1],[0.1 0.4]+offset(2),'k','Linewidth',2)
plot([-1.5 -1],[-0.5 -0.2]+offset(3),'k','Linewidth',2)
plot([-1.5 -1],[-0.3 0.1]+offset(3),'k','Linewidth',2)
plot([-1.5 -1],[0.1 0.4]+offset(3),'k','Linewidth',2)
axis equal
box on
if offset(1) == 0
    ylim([-2.5,m])
else
    ylim([-1,m])
end
xlim([-2,n])
end
```