

ESC103 Group Assignment

Group Members: Mark Estrada, Clara Fleisig, Sean O'Rourke, Riddhiman Roy,
Trinity Yu
TUT0109

Contributions

Mark Estrada: Throughout the activity, I worked with the group with a traditional CIV perspective to identify the intuition and the expected results (i.e. ratios for questions 6 and 7, question 4) before applying the linear algebra and MATLAB-oriented lens to the assignment. Due to my lack of experience in MATLAB programming, I tried to contribute to the initial matrix set-up. I also often inquired about the MATLAB coding that the rest of my group members developed in my greatest attempts to help and debug the code, specifically with our greatest issue of having plus or minus signs switched. I worked on writing the report's solutions and conclusion, as well as working with Trin on documentation and in better representing the report in a clear and concise manner.

Clara Fleisig: I worked with Riddhiman to devise how we wanted the code to work. We broke the problem down into steps on a zoom whiteboard. Using our understanding of the relationships of member forces and the equilibrium of internal forces at joints from CIV102 we wrote out what each row of matrix A and vector b should represent. We then wrote a few lines of matrix A by hand to find a pattern which helped us easily input the required values for matrix A into MATLAB. Using Sean's code as a starting point, I wrote algorithms to produce matrix A. After discovering there were a few problems with signs on the whiteboard work Riddhiman and I did, I went back and corrected matrix A. I developed the augmented form and reduced normal form, checked the answer's using Riddhiman's CIV102 assignment answers, and commented throughout the code.

Sean O'Rourke: During this assignment, I was responsible for writing the first iteration of the algorithm used to find matrix A. I used a combination of Clara and Riddhiman's initial work, along with trial and error, until the for loops produced the correct values for the matrix.

Riddhiman Roy: For this assignment, I worked with Clara in the early phases to conceptually map out the matrices and understand what each operation would mean with respect to our truss. I worked with her on a zoom whiteboard, often mapping out individual rows by hand. Through this, we worked to find patterns in our truss analysis and linked the matrices and the operations to the method of joints. I also helped to debug Clara's and Sean's code in MATLAB, trying to solve issues related to signs and missing rows. I'd try different solutions to their code in a separate copy to resolve the aforementioned issues. In addition, I worked, in MATLAB, on problems 5, 6 and 7 with my team on zoom calls. I also contributed to the creation of the final report, doing most of questions 4-7.

Trinity Yu: In this assignment, I contributed to the creation of this document and affirmation of the code in MATLAB that was created. I helped with ensuring that our initial pattern of the A matrix was correct before starting to write our code. Individually, we hand wrote what we thought the matrix would be, then came together to see our results. Since I am a beginner at MATLAB, my team members helped me understand the program and how to apply it to this assignment. My team members are much more advanced than I am using this code so they were able to quickly solve the questions. Despite this, I attempted to debug and improve the code. I feel that I have greatly increased my understanding of the program. Mark and I were able to create this document and help in writing and clarifying the explanations and formatting to ensure we met the assignment requirements.

Signatures - Names *below* signatures

A handwritten signature in black ink that reads "Mark Estrada". The letters are cursive and fluid, with a long horizontal stroke at the end.

Mark Estrada

A handwritten signature in black ink that reads "Clara Fleisig". The signature is written in a cursive style with a prominent horizontal line crossing through the middle of the name.

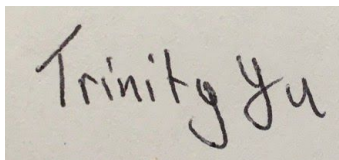
Clara Fleisig

A handwritten signature in black ink that reads "Sean O'Rourke". The signature is written in a cursive style with a large, looped 'S' and a long horizontal stroke at the end.

Sean O' Rourke

A handwritten signature in black ink that reads "Riddhiman Roy". The signature is written in a cursive style with a large, looped 'R' and a long horizontal stroke at the end.

Riddhiman Roy

A handwritten signature in black ink that reads "Trinity Yu". The signature is written in a cursive style with a large, looped 'T' and a long horizontal stroke at the end.

Trinity Yu

Table of Contents

Question 1	6
Question 2	7
Question 3	11
Question 4	12
Question 5	14
Question 6	15
Question 7	16
Conclusion	18

Question 1

Expressing joint and member force variables in the form $Ax = b$:

```
% x is the vector we are solving for and contains the forces of each member
% in the order [AB AC BC BD CD ... LM], transposed
x = zeros(23, 1)';

% b contains the sum of the internal x and y forces at each joint
% in order [Ax Ay Bx By Cx Cy ... Mx My]
b = zeros(26, 1);

% Odd rows represent the sum of x force equations
% Even rows represent the sum of y force equations
A = zeros(26, 23);

% Angle between each member
theta = pi()/3;
```

Question 2

Putting values into vector B

Input:

```
% add applied joint loads from tributary analysis
for i = 6:4:length(x)
    b(i, 1) = -350;
end
% We define upwards as positive y, right as positive x
%add reaction forces into reaction forces
%b(2, 1) is the sum of internal forces for Ay
b(2, 1) = 875;

%the last item in b is the sum of internal forces, My
b(length(b), 1) = 875;
```

Putting values into matrix A

Input:

```
%%Putting equations for sum of y forces into even rows
%for diagonal force members (sin(theta) values)
%Using pattern found on whiteboard
%if statements ensures values not put in coordinates outside of
%23 * 26 matrix size bounds

%determines if the value should be negative or positive
sign_of_row = 1;

for i = 2:2:length(b)
    %switches between adding negative value and positive values to rows
    sign_of_row = sign_of_row * -1;
    if i <= length(x) + 1
        A(i, i - 1) = sin(theta) * sign_of_row;
    end
    if i >= 4
        A(i, i-3) = sin(theta) * sign_of_row;
    end
end

%%Putting in the equation for sum of x-forces into odd rows
%for members (1 and cos(theta) magnitude values)
%Using pattern found on whiteboard
%if statements ensures values not put in coordinates outside of
```

```

%23 * 26 matrix size bounds
for i = 1:2:length(x) + 2
    if i + 1 <= length(x)
        A(i, i + 1) = -1;
    end
    A(i, i) = -cos(theta);
    if i + 1 <= length(x)
        A(i, i + 1) = cos(theta);
    end
    if i <= length(x)
        A(i, i) = 1;
    end
    if i - 2 >= 1 && i - 2 <= length(x)
        A(i, i - 2) = 1;
    end
    if i - 3 >= 1
        A(i, i - 3) = 1;
    end
end

%Make augmented matrix by adding b as an extra column on A
Augmented = zeros(length(b), length(x) + 1);
Augmented(:, [1:length(x)]) = A;
Augmented(:, length(x) + 1) = b;

%Apply built in gaussian elimination function to obtain reduced row
form
result_1 = rref(Augmented) %Solution to Q3

```

Output:

26x24 double																									
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	
1	-0.5000	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	-0.8660	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	875	
3	0.5000	0	-0.5000	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0.8660	0	0.8660	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
5	0	1	0.5000	0	-0.5000	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
6	0	0	-0.8660	0	-0.8660	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-350	
7	0	0	0	1	0.5000	0	-0.5000	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
8	0	0	0	0	0.8660	0	0.8660	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
9	0	0	0	0	0	1	0.5000	0	-0.5000	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
10	0	0	0	0	0	0	-0.8660	0	-0.8660	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-350	
11	0	0	0	0	0	0	0	1	0.5000	0	-0.5000	-1	0	0	0	0	0	0	0	0	0	0	0	0	
12	0	0	0	0	0	0	0	0	0.8660	0	0.8660	0	0	0	0	0	0	0	0	0	0	0	0	0	
13	0	0	0	0	0	0	0	0	0	1	0.5000	0	-0.5000	-1	0	0	0	0	0	0	0	0	0	0	
14	0	0	0	0	0	0	0	0	0	0	-0.8660	0	-0.8660	0	0	0	0	0	0	0	0	0	0	-350	
15	0	0	0	0	0	0	0	0	0	0	0	1	0.5000	0	-0.5000	-1	0	0	0	0	0	0	0	0	
16	0	0	0	0	0	0	0	0	0	0	0	0	0.8660	0	0.8660	0	0	0	0	0	0	0	0	0	
17	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.5000	0	-0.5000	-1	0	0	0	0	0	0	
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-0.8660	0	-0.8660	0	0	0	0	0	0	-350	
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.5000	0	-0.5000	-1	0	0	0	0	
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.8660	0	0.8660	0	0	0	0	0	
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.5000	0	-0.5000	-1	0	0	
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-0.8660	0	-0.8660	0	0	-350	
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.5000	0	-0.5000	0	
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.8660	0	0.8660	0	
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.5000	0	
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-0.8660	875	

Figure 1: Augmented matrix $A|b$, where A is from columns 1 - 23, and column 24 is the column vector b .

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1.0104e+03
2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	505.1815
3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0104e+03
4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1.0104e+03
5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-606.2178
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.3135e+03
7	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	606.2178
8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1.6166e+03
9	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-202.0726
10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1.7176e+03
11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	202.0726
12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	-1.8187e+03
13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	202.0726
14	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1.7176e+03
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-202.0726
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-1.6166e+03
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	606.2178
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1.3135e+03
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	-606.2178
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	-1.0104e+03
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1.0104e+03
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	505.1815
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	-1.0104e+03
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 2: Reduced Normal Form Matrix of size 26x23 (the last extra column is due to the answer column vector, called result_1).

The dimensions of Matrix A in reduced normal form is 26 x 24, but there are 3 zero rows. The rank can be recognized from the number of linearly independent row vectors in the matrix, which in Matrix A are non-zero rows. Therefore, $r = \text{rank } A = 23$.

Question 3

Output:

23x1 double	
	1
1	-1.0104e+03
2	505.1815
3	1.0104e+03
4	-1.0104e+03
5	-606.2178
6	1.3135e+03
7	606.2178
8	-1.6166e+03
9	-202.0726
10	1.7176e+03
11	202.0726
12	-1.8187e+03
13	202.0726
14	1.7176e+03
15	-202.0726
16	-1.6166e+03
17	606.2178
18	1.3135e+03
19	-606.2178
20	-1.0104e+03
21	1.0104e+03
22	505.1815
23	-1.0104e+03

Figure 3: result_1, the solution to Question 3

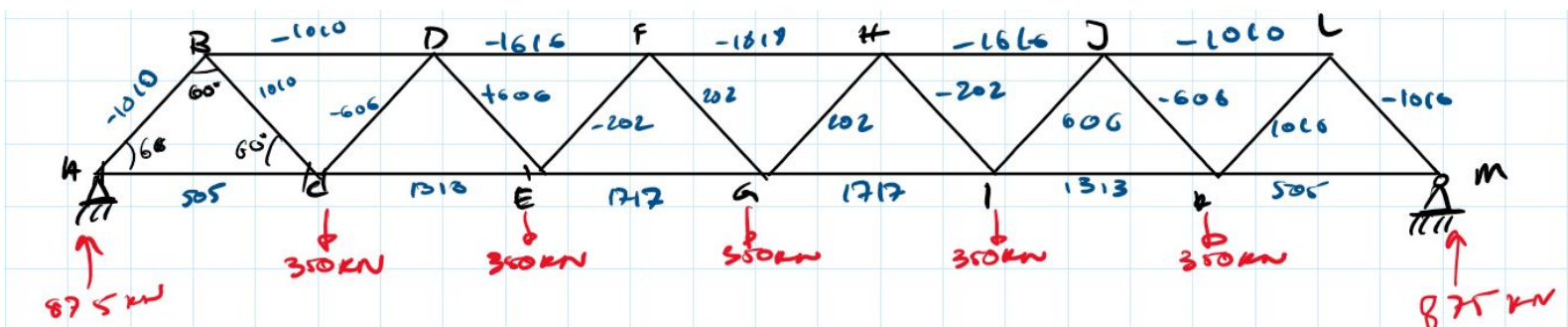


Figure 4: Our solutions, by method of joints and TA Allan, from Assignment 5. All forces are in kN, specified or otherwise.

Question 4

We removed the equations corresponding to G_x , G_y and M_y as the product of the column vector of forces and each row of the coefficient matrix is the method of joints for the x or y direction for a particular joint. Therefore, we removed the equations corresponding to G_x and G_y since that joint, in human calculations, is never evaluated. Furthermore, since we're proceeding from the left side, evaluating L_y gives us L_M , leaving no need to calculate M_y .

Input:

```
%Problem 4: remove equations corresponding to
Gx, Gy, My (G is never evaluated, have My
from L)
A(13, :) = [];
A(13, :) = []; %with deletion, the size
changes, so the position also changes
A(24, :) = [];
b(13) = [];
b(13) = [];
b(24) = [];
Result_2 = A\b;
```

Output:

23x23 double

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	-0.5000	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.8660	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	-0.5000	0	0.5000	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0.8660	0	0.8660	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0.5000	0	-0.5000	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0.8660	0	0.8660	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	-1	-0.5000	0	0.5000	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0.8660	0	0.8660	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	1	0.5000	0	-0.5000	-1	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0.8660	0	0.8660	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	-1	-0.5000	0	0.5000	1	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0.8660	0	0.8660	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	-1	-0.5000	0	0.5000	1	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0.8660	0	0.8660	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.5000	0	-0.5000	-1	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.8660	0	0.8660	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	-0.5000	0	0.5000	1	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.8660	0	0.8660	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.5000	0	-0.5000	-1	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.8660	0	0.8660	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.5000	0	-0.5000
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.8660	0	0.8660
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.5000

Figure 5: The modified A matrix after removing the redundant rows.

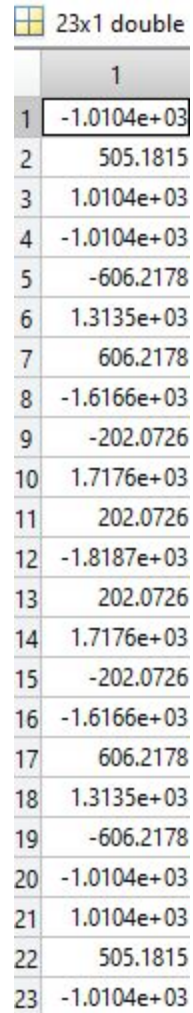
	23x1 double
	1
1	-1.0104e+03
2	505.1815
3	1.0104e+03
4	-1.0104e+03
5	-606.2178
6	1.3135e+03
7	606.2178
8	-1.6166e+03
9	-202.0726
10	1.7176e+03
11	202.0726
12	-1.8187e+03
13	202.0726
14	1.7176e+03
15	-202.0726
16	-1.6166e+03
17	606.2178
18	1.3135e+03
19	-606.2178
20	-1.0104e+03
21	1.0104e+03
22	505.1815
23	-1.0104e+03

Figure 6: Result_2, solution to Question 4.

Question 5

Input:

```
%Problem 5  
A_inv = inv(A);  
Result_3 = A_inv*b
```



	1
1	-1.0104e+03
2	505.1815
3	1.0104e+03
4	-1.0104e+03
5	-606.2178
6	1.3135e+03
7	606.2178
8	-1.6166e+03
9	-202.0726
10	1.7176e+03
11	202.0726
12	-1.8187e+03
13	202.0726
14	1.7176e+03
15	-202.0726
16	-1.6166e+03
17	606.2178
18	1.3135e+03
19	-606.2178
20	-1.0104e+03
21	1.0104e+03
22	505.1815
23	-1.0104e+03

Figure 7: Result_3, solution to Question 5

Question 6

For this, we'd multiply every force value by a factor of $\frac{90}{70}$. Therefore, only b_r in $[A_r | b_r]$ is expected to change, being increased by a factor of $\frac{90}{70}$. To verify, we can reevaluate the matrix, this time changing the initial b vector's values as following:

$$P_{joint} = 90 * 5 = 450kN \Rightarrow P_A = P_B = \frac{(450 * 5)}{2} = 1125kN$$

Thus, implementing this into the original code:

```
% add applied joint loads from tributary analysis
for i = 6:4:length(x)
    b(i, 1) = -450;
end
% We define upwards as positive y, right as positive x
%add reaction forces into reaction forces
%b(2, 1) is the sum of internal forces for Ay
b(2, 1) = 1125;

%the last item in b is the sum of internal forces, My
b(length(b), 1) = 1125;
```

This leaves the following solution:

23x1 double		23x1 double	
	1		1
1	-1.2990e+03	1	-1.2990e+03
2	649.5191	2	649.5191
3	1.2990e+03	3	1.2990e+03
4	-1.2990e+03	4	-1.2990e+03
5	-779.4229	5	-779.4229
6	1.6887e+03	6	1.6887e+03
7	779.4229	7	779.4229
8	-2.0785e+03	8	-2.0785e+03
9	-259.8076	9	-259.8076
10	2.2084e+03	10	2.2084e+03
11	259.8076	11	259.8076
12	-2.3383e+03	12	-2.3383e+03
13	259.8076	13	259.8076
14	2.2084e+03	14	2.2084e+03
15	-259.8076	15	-259.8076
16	-2.0785e+03	16	-2.0785e+03
17	779.4229	17	779.4229
18	1.6887e+03	18	1.6887e+03
19	-779.4229	19	-779.4229
20	-1.2990e+03	20	-1.2990e+03
21	1.2990e+03	21	1.2990e+03
22	649.5191	22	649.5191
23	-1.2990e+03	23	-1.2990e+03

Figure 8: The new matrix, on the left with new loading as described in the equation above, compared to the old matrix, multiplied by a factor of $\frac{90}{70}$. They are identical.

Question 7

To make sure that the maximum compressive force does not exceed 2000kN, we use the following ratio:

$$\frac{1818kN}{2000kN} = \frac{70kN}{x \text{ kN}}$$

Where x is the maximum allowed joint load to ensure that the maximum compressive force in members does not exceed 2000kN. Solving this, we have $x = 77.00770077\text{kN}$ (the extra significant figures are left in for precision in further calculations). To check this, we can modify code similarly to problem 6. Thus, implementing this into the original code:

```
% add applied joint loads from tributary analysis
for i = 6:4:length(x)
    b(i, 1) = -385;
end
% We define upwards as positive y, right as positive x
%add reaction forces into reaction forces
%b(2, 1) is the sum of internal forces for Ay
b(2, 1) = 962.5;

%the last item in b is the sum of internal forces, My
b(length(b), 1) = 962.5;
```


res3	
23x1 double	
	1
1	-1.1115e+03
2	555.7552
3	1.1115e+03
4	-1.1115e+03
5	-666.9063
6	1.4450e+03
7	666.9063
8	-1.7784e+03
9	-222.3021
10	1.8896e+03
11	222.3021
12	-2.0007e+03
13	222.3021
14	1.8896e+03
15	-222.3021
16	-1.7784e+03
17	666.9063
18	1.4450e+03
19	-666.9063
20	-1.1115e+03
21	1.1115e+03
22	555.7552
23	-1.1115e+03

Figure 9: The modified solutions, with the new loading, showing a new maximum load of ~2000.7kN. It is not precisely 2000kN due to imprecision in the ratio in MATLAB.

Conclusion

This assignment has helped demonstrate the *priceless* utility of MATLAB in the realm of engineering and the way in which linear algebra lends itself towards real-world engineering applications. Solving a series of repetitive equations using matrices in MATLAB rather than solving each individual equation by hand was much more efficient. We learned the importance of knowing what the code “should” do at each step before writing in the live script as it made debugging easier. We could easily see exactly which blocks of code in which mistakes were made and thus fix those mistakes more easily. For example, while we were debugging the loops, we could easily see where the produced pattern differed from the human calculated pattern. This transparency is often missing in other languages like python, which often results in disastrously (and hilariously) wrong code and outputs. Recognizing truss problems under the perspective of matrix vector relationships has allowed us to expand our understanding of MATLAB linguistics and reinforces our understanding of linear algebra.

This lab truly demonstrated the sheer power of MATLAB and more importantly, of linear algebra. It made concrete the real world application of concepts that seem arcane at times. We come out of this project with a better understanding of linear algebra and hope to employ it more significantly and frequently in future projects.