Stage 3 Mechanical Engineering, UCD
EEEN30150
Modelling and Simulation
Minor Project 1
Static Equations: Problem 3 – Bridge

Daniel Jakob
18409686

20th March 2021

Module Coordinator:	Assoc. Prof. Paul Curran
Date Due:	21st March 2021

Contents

Assessment Submission Form

Student Name	Daniel Jakob
Student Number	18409686
Assessment Title	Static Equations: Problem 3 – Bridge
Module Code	EEEN30150
Module Title	Modelling and Simulation
Module Coordinator	Assoc. Prof. Paul Curran
Date Submitted	20th March 2021
Date Received	
Grade/Mark	

Signed Daniel Jakob **Date** 20th March 2021

1 Modelling

1.1 Theory

First a check was made that the truss structure is statically determinate:

$$2j = m + r \tag{1}$$

where: j = number of joints
m = number of members
r = number of reactions

From the brief we are told that there is one roller joint and one hinge joint at either end of the bridge, giving 3 reactions, and we will consider only a *planar* truss for this. There are 21 members and 12 joints, as seen in Fig. 1, and 2 times 12 does indeed equal 21 plus 3, as from Eq. 1, therefore, the truss is statically determinate.

S275 Carbon Steel has a Young's Modulus, E, of 210 GPa and a density, ρ_{steel}, of 7850 kg m^{-3} (European Committee for Standardization, 2004). Reinforced concrete has a density, ρ_{concrete}, of 2400 kg m^{-3} (Dorf, 1996). Density of asphalt, ρ_{asphalt}, is 2243 kg m^{-3} (Aqua-Calc, 2021).

The assumption that the lorry drives in the middle of the bridge and drives perfectly straight across the bridge was made. The local gravitational acceleration due to Earth will be taken as −9.81 m s^{-2}.

1.2 Direct Stiffness Method

The direct stiffness method involves modelling a system into a decomposed set of idealised elements connected via nodes. In this structure, all nodes are connected via two-node bars which can be modelled as springs of stiffness k. Many assumptions are made including: members only carry axial loads, joint connections are frictionless, that deflections caused by the forces in the members do not change the length of the members and so do not change the equivalent spring stiffness for the deflected member and that there are no bending forces or moments about the hinged and rolled joints.There are three constituent parts: the global stiffness matrix, **K**, the displacement vector, **u**, and the force vector, **f**, which are related through Eq. 2, which is just a variation on Hooke's Law $f = -kx$.

$$\mathbf{f} = \mathbf{K}\mathbf{u} \tag{2}$$

where: **f** = force vector (2× no. of nodes column matrix)
K = global stiffness matrix (2× no. of nodes square, symmetric matrix)
u = nodal displacements (2× no. of nodes column matrix)

The *equivalent spring stiffness* for each member is calculated in their local coordinate system, according to Eq. 3, this is easier than calculating it from a global coordinate system. This *local stiffness* matrix is then transformed into the coordinate system of the global structure and these global components are inserted into the global stiffness matrix, **K**.

$$k = \frac{EA}{L} \tag{3}$$

(Equivalent spring stiffness)

where: E = Young's modulus of member
A = Area of member $(0.12\,\mathrm{m})^2$
L = Length of member

1.3 Simplification to 2-D

This problem can be simplified to 2-D analysis by the observation that the forces acting on the truss only act in the x–z and x–y planes and never in the y–z plane. This allows for the *cross* members (drawn in white in Fig. 3) to be ignored — as the forces in the analysed system never exert any compression or tension in these members — allowing the bridge to be analysed in 2-D. Due to this, only one side of the bridge was analysed, as the bridge is symmetric along the x–z plane. Therefore, all weights were divided by 2, to account that the left and right sides of the bridge carry all forces equally, by the lorry driving centrally assumption.

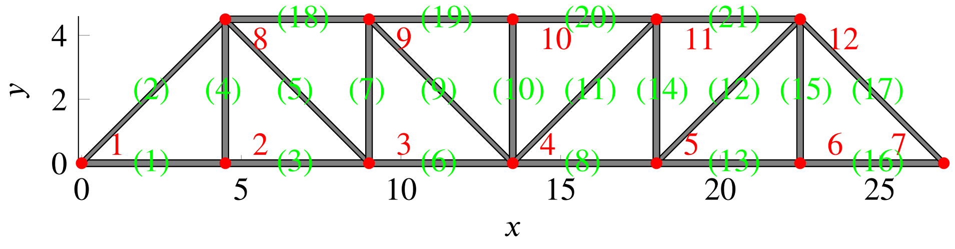


Figure 1: 2-D nodes and members

2 Simulation

2.1 Solving Systems of Linear Equations

The method of *Gaussian Elimination* with total pivoting will be used to solve systems of linear equations (of the form $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{x}$ is unknown) for this problem. This method relies on transforming a matrix $\mathbf{A}$ into an upper triangular matrix via *elementary row operations*. Once the matrix is in upper triangular form, $\mathbf{x}$ can be found by iterative back substitution. Total pivoting involves finding the largest coefficient by searching the columns and rows in $\mathbf{A}$ and moving this entry to the top right of the matrix — as opposed to just the column like in *partial* pivoting — reflecting this change in $\mathbf{b}$ and $\mathbf{x}$.

2.2 Loadings

There are three main loadings on the truss bridge: the members, the floor (reinforced concrete, asphalt and floor panels), and the lorry (front axle and rear axle). From the brief, we can assume that “all loads are applied to joints only”, and that if a “member is of significant weight, then half of its weight may be considered to be applied to the joints at either end of the member”. This by extension applies to the flooring, and the lorry’s weight. Since we analyse the truss bridge in 2-D and then translate to 3-D as explained in Subsec. 1.3, all weights associated with the flooring and lorry were divided by 2, as the left and right sides carry the weights equally. Then each node was considered in terms of how many members and how much flooring was attached to it. Take Node 1 for example from Fig. 1, it has two members attached to it, and one quarter of a flooring section, while node 4 has 5 members and a half of a flooring panel (a quarter to each side). The weights of the members were calculated by Eq. 4, and the weight of a flooring panel was calculated by Eq. 5.

$$W_{\text{short}} = \rho_{\text{steel}} A_{\text{bar}} \times 4.5 \times -9.81 \; ; \; W_{\text{long}} = \rho_{\text{steel}} A_{\text{bar}} \sqrt{4.5^2 + 4.5^2} \times -9.81 \quad (4)$$

$$W_{\text{flooring}} = \left(\rho_{\text{concrete}} (0.15 + 0.025) + \rho_{\text{asphalt}} (0.05)\right) \times 4.5^2 \times -9.81 \quad (5)$$

The loading for the lorry was done by finding a weighted average of the distance the axle is from one node to the next. If an axle is located at $x = 7.1$ for example, this is between node 2 at $x = 4.5$ and node 3 at $x = 9$. The weighted average was computed according to Eq. 6, where $\bar{x}$ is the x-coordinate of the axle in question.

$$\texttt{load}_{\left(\left\lfloor \frac{\bar{x}}{4.5} \right\rfloor + 1\right),2} = \left[1 - \left(\frac{\bar{x}}{4.5} - \left\lfloor \frac{\bar{x}}{4.5} \right\rfloor\right)\right] W_{\text{axle}} \quad (6)$$

$$\texttt{load}_{\left(\left\lfloor \frac{\bar{x}}{4.5} \right\rfloor + 2\right),2} = \left(\frac{\bar{x}}{4.5} - \left\lfloor \frac{\bar{x}}{4.5} \right\rfloor\right) W_{\text{axle}} \quad (7)$$

2.3 Code

Listing 1: `makelink.m` function m-file

```matlab
function [] = makelink(x1, y1, z1, x2, y2, z2, t, c)
%makelink(x1, y1, z1, x2, y2, z2, thickness)
%   MAKELINK plots a square based bar in 3D space
%   takes 8 inputs:
%   x1, y2, z1 are the 3D coordinates of the start of the link
%   x2, y2, z2 are the 3D coordinates of the end of the link
%   t is the thickness of the bar
%   c is the colour (RGB array) of the bar

% Author: Daniel Jakob (18409686), 06/03/2021, Version 1.2

t = t/2; % bar thickness halved, centering bar
if z1 == z2 % flat members
    if y1 == y2 % going in x-direction
        vertices =  [x1 y1+t z1-t;
16                      x1 y1+t z1+t;
17                      x1 y1-t z1+t;
18                      x1 y1-t z1-t;
19                      x2 y2+t z2-t;
20                      x2 y2+t z2+t;
21                      x2 y2-t z2+t;
22                      x2 y2-t z2-t];
23
24      else % going in y-direction (must rotate by 90 deg)
25          vertices =  [x1+t y1 z1-t;
26                      x1+t y1 z1+t;
27                      x1-t y1 z1+t;
28                      x1-t y1 z1-t;
29                      x2+t y2 z2-t;
30                      x2+t y2 z2+t;
31                      x2-t y2 z2+t;
32                      x2-t y2 z2-t];
33      end
34  else % non-flat members
35      vertices=   [x1+t y1-t z1;
36                  x1+t y1+t z1;
37                  x1-t y1+t z1;
38                  x1-t y1-t z1;
39                  x2+t y2-t z2;
40                  x2+t y2+t z2;
41                  x2-t y2+t z2;
42                  x2-t y2-t z2];
43
44  end
45
46  faces = [1 2 3 4;
47           1 2 6 5;
48           2 3 7 6;
49           3 4 8 7;
50           4 1 5 8;
51           5 6 7 8];
52  patch('Vertices', vertices, 'Faces', faces, 'FaceColor', c);
53  end
```

List. 1 is the function m-file that draws the members of the bridge. It takes 8 inputs, the three spacial coordinates of the beginning of a member, the three coordinates of the end of a member, thickness, `t` in meters and an RGB triplet colour array, `c`. First the thickness is halved, in Ln. 12. Then, a check is made for if the bar is flat or a vertical type bar in Ln. 13, directly after which, a check is made for if the bar is going in the *x*-direction or *y*-direction. The vertices are setup accordingly, for the *x*-direction this is from Ln. 15 to Ln. 22 and is illustrated in Fig. 2(b). The vertices are subtly different for the three variations of bars, but the differences essentially rotate the bar appropriately to create a visually pleasing bar. Then, the `faces` matrix sets up how the vertices connect to one another, Ln. 46 to Ln. 51. Finally, Ln. 52 draws the bar in the figure using `vertices`, `faces` and `c` utilising the `MATLAB patch` command (MathWorks, 2020).

Listing 2: `gauss_elim.m` function m-file

```
1  function v = gauss_elim(A,b)
2  %function v = gauss_elim(A,b):
3  %   GAUSS_ELIM solves linear equations expressed in matrix form Av = b
4  %   using Gaussian Elimination with total pivoting.
5  %   Input A = square matrix of coefficients
```

```
6   %   Input b = vector of constants
7   %   Output v = solution vector.
8
9   % Original Author: Paul Curran. Adapted by Daniel Jakob.
10  % Version 1.1: created 18/02/21.
11
12  % A needs to be nxn matrix. b needs to be nx1 vector
13  [rowsA,colsA] = size(A);
14  [rowsb,colsb] = size(b);
15  if ~(rowsA==colsA) || ~(colsb==1) || ~(rowsb==rowsA) || ~ismatrix(A)
16      error('A must be a square matrix and b must be a column vector')
17  end
18  if ~isreal(A) || ~isreal(b)
19      error('Matrix A and b must be made of real numbers only')
20  end
21  if ~ismatrix(A) || ~ismatrix(b)
22      error('Inputs A and b must be a matrix and array respectively')
23  end
24
25  Ra = size(A);
26  n = Ra(1);
27  Id = eye(n);
28  Q_tot = Id;
29  A1 = A; % running values of A
30  b1 = b; % running values of b
31
32  for m = 1:n-1
33      % Total Pivoting
34      row_index = [m:n];
35      [Yc,Ir] = max(abs(A1(row_index, row_index)));
36      [~,Ic1] = max(Yc);
37      row_num = Ir(Ic1);
38      col_num = Ic1;
39      p = [1:n]; p(m) = m-1+row_num; p(m-1+row_num) = m;
40      q = [1:n]; q(m) = m-1+col_num; q(m-1+col_num) = m;
41      P = Id(p,:); Q = Id(:,q);
42      A1 = P*A1*Q; b1 = P*b1; Q_tot = Q*Q_tot;
43
44      % Elimination
45      I_vec = zeros(1,n); I_vec(1,m) = 1;
46      if A1(m, m) == 0 % check that division of zero doesn't take place
47          error('Cannot divide by 0')
48      end
49      L1 = Id - ([zeros(m,1);A1(m+1:n,m)]*I_vec)/A1(m,m);
50      A1 = L1*A1; b1 = L1*b1;
51  end
52
53  % Back Substitution
54  v = zeros(n,1);
55  for m = 0:n-1
56      if A1(n-m, n-m) == 0 % check that division of 0 doesn't take place
57          error('Cannot divide by 0')
58      end
59      v(n-m,1) = (b1(n-m,1) - (A1(n-m,:)*v))/A1(n-m,n-m);
60  end
61
62  v = Q_tot'*v; % output of solution vector
63
64  end
```

List. 2 is the function m-file which utilises Gaussian Elimination method outlined in Subsec. 2.1. This function is adapted from Curran, 2020. It takes two inputs, matrix **A** and column vector **b**. Error checking occurs from Ln. 13 to Ln. 23;

checking that input `A` is a matrix, square and is comprised of real numbers only, and that `b` has an equal number of rows as `A` has columns, and that is a column vector. From Ln. 25 to Ln. 30, are the various initialisations, finding sizes, creating running values, etc. Then the `for` loop begins from index $m = 1$ up to size of the column of `A` minus 1, $n - 1$. Then from Ln. 34 to Ln. 42, the total pivoting takes place. The `max(abs(A1...))` on Ln. 35 along with the next line, locates the entry with the largest coefficient in `A`, and then Ln. 39 and Ln. 40 swap the row and column of all entries to move the largest coefficient to the `(1, 1)` index. Then in Ln. 42, **A**, which is now going by its running value `A1` and **b**, now `b1` get pre- and post-multiplied by `P` and `Q` respectively, which are the permutation matrices which appropriately swap the columns and rows of the matrix to reflect the changes made in Ln. 39 and Ln. 40. The elimination phase takes place from Ln. 45 to Ln. 50 where the running matrices of `A1` and `b1` are transformed into upper triangular matrices iteratively over the course of the loop. Finally, the back substitution phase takes place on Ln. 59, where the **v** matrix is initialised as a zero array and is populated according to Eq. 8. A check for division of zero is made in Ln. 46 and Ln. 56 during the elimination and back substitution phases of the algorithm. This *should* not ever happen as this only occurs in *Naïve* Gaussian Elimination, but a check is made nonetheless, just in case.

$$\mathbf{v}_{n-m,1} = \mathbf{b}_{n-m,1} - \frac{\left(\mathbf{A}_{n-m} \times \mathbf{v}\right)}{\mathbf{A}_{n-m,n-m}} \tag{8}$$

where: $\mathbf{v}$ = initialised as a zeros matrix

Listing 3: `direct_stiffness.m` function m-file

```matlab
function [F, U] = direct_stiffness(Coord, Con, Re, Load, EA)
%function [F, U] = direct_stiffness(Coord, Con, Re, Load, E, A)
%   Analyse a Truss using the direct stiffness method
%   Inputs: Coord = N x 2  array of node coordinates
%           Con   = N x 2  array of connector or member mapping
%           Re    = N x 2  array of node freedom  1 = fixed 0 = free
%           Load  = N x 2  array of load force vectors
%           EA    = Pre-calculated E*A to reduce computations, where:
%                   A = Member cross section area [m^2]
%                   E = Member Elasticity (Young's Modulus)
%
%   Outputs:F   = M x 1 array of force along members
%           U   = N x 2 array of Node displacement vectors

% Original code by  Hossein Rahami 13/04/07, updated by
% Frank McHugh 06/09/12. Adapted by Daniel Jakob, 09/03/21, version 1.1

w=size(Re);      % 2 x number of nodes
K=zeros(2*w(2)); % stiffness matrix is 2*(no. of nodes) square matrix
U=1-Re;      % U is displacement  matrix
             % column index by node
             % x, y, z by rows
             % initialise U to 1 for non fixed nodes 0 for fixed
f=[1 3 4 5 6 7 8 9 10 11 12 15 16 17 18 19 20 21 22 23 24]';
             % the free nodes of U
Tj = zeros(size(Con)); % preallocating Tj
for i=1:size(Con,2) % Loop through Connectors (members)
```

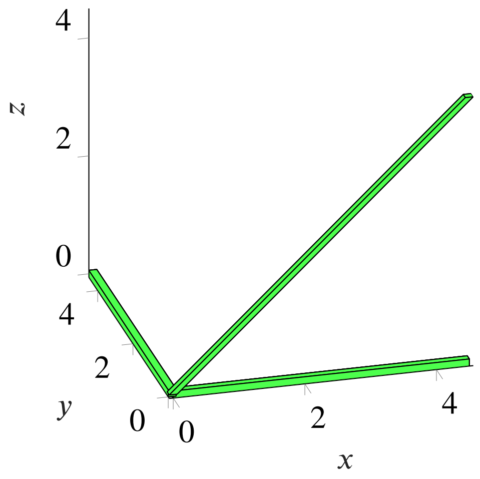
```
28    H=Con(:,i);
29    C=Coord(:,H(2))-Coord(:,H(1));  % C is vector for connector i
30    Le=norm(C);                      % Le length of connector i
31    T=C/Le;                          % T is unit vector for connector i
32    s=T*T';       %   Member Stiffness matrix is of form
33                  %   k * |  s  -s |
34                  %       |  -s  s | in global truss coordinates
35    G=EA/Le;     % G aka k stiffness constant of member = E*A/L
36    Tj(:,i)=G*T;            % Stiffness vector of this member
37    e=[2*H(1)-1:2*H(1),2*H(2)-1:2*H(2)];
38                 % indexes into Global Stiffness matrix K for this member
39    K(e,e)=K(e,e)+G*[s -s;-s s];
40                 % add this members stiffness to stiffness matrix
41 end
42 U(f)=gauss_elim(K(f,f), Load(f)); % solve for displacements of free
43                          %  nodes, i.e., solve F = K*U  for U where K
44                          %  is stiffness matrix using gauss_elim.m
45 F=sum(Tj.*(U(:,Con(2,:))-U(:,Con(1,:))));
46                  % project displacement of each node pair on to member
47                  % between f = Tj dot (U2j - U1j).
48                  % Then sum over all contributing node pairs.
```

List. 3 is the implementation of the Direct Stiffness method outlined in Subsec. 1.2. This function was adapted from McHugh, 2021. This function takes in 5 inputs, the coordinates matrix `Coord`, the connection matrix `Con`, the reactions for each node matrix `Re`, the load at each node matrix `Load` and the pre-computed `EA` value which is the product of E and A, which happens to be constant in this problem. The function begins by defining some constants and initialising some matrices to be used later. Then a `for` loop loops through all the members of the structure. At Ln. 28, the `H` variable is declared which finds the nodes of the `i`-th connection. Ln. 29 to Ln. 31 find the vector for the `i`-th member akin to $(x_2 - x_1, y_2 - y_1)$, finds the magnitude/length of this vector, and then finds the normalised/unit vector of this and stores in it `T`. Directly after, the product of `T` and its transpose are calculated to give the member stiffness matrix `s`. Then in Ln. 35, the equivalent stiffness constant `G` is found according to Eq. 3. Next the stiffness vector `Tj` is found for the `i`-th member and is the product of `G` and `T`. Then a variable `e` is declared which is used to index into the global stiffness matrix `K`. In Ln. 39, the `K` matrix, previously initialised as a zeros matrix is iteratively populated with `G*[s -s;-s s]` (which converts the local member stiffnesses to global system), in position (`e`, `e`). Once the `for` loop is complete, the displacement vector `U` is calculated by solving Eq. 2; the loads at each node are known and the global stiffness matrix has just been constructed. The `gauss_elim.m` function is utilised here to solve this system of linear equations. Finally the forces in each member is calculated in Ln. 45 by find the sum of the piecewise multiplication of `Tj` with $(\mathbf{u}_{2_j} - \mathbf{u}_{1_j})$ which is the element force.

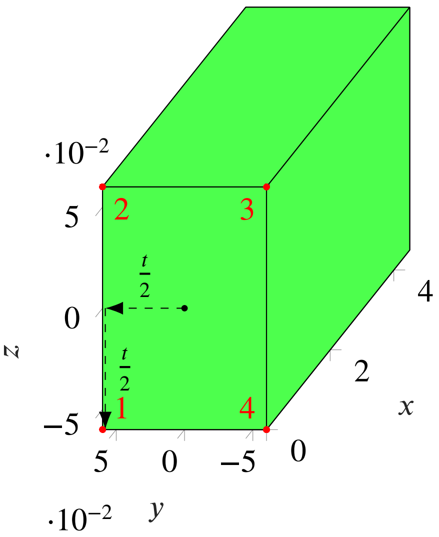
3 Visualisation

Although the truss analysis was carried out in 2-D as outlined in Subsec. 1.3, visualisation had to be done in 3-D. This was achieved by mirroring the right side of the bridge (the one which was analysed) to the left side. To do this, the nodes, and members had to be consistent between 2-D and 3-D (otherwise a complicated table of relations from 2-D members to 3-D members would be needed). Therefore, two matrices for each coordinates of nodes and connections exist for both 2-D and 3-D, and the 3-D connection matrix contains the analysed members, the members which are mirrored and the cross-bracing members (coloured in white in Fig. 3).

One type of bar was used in List. 1 to create the members of the truss bridge, a square-based cuboid, reflecting the brief. The bar had to be rotated by 90° for the 3 different directions that the bar could go in, along the x-direction, the y-direction and the diagonal members going in the z-direction, as seen in Fig. 2(a).



(a) Three variants of the members



(b) bar going in x-direction vertices

Figure 2: Visualisations

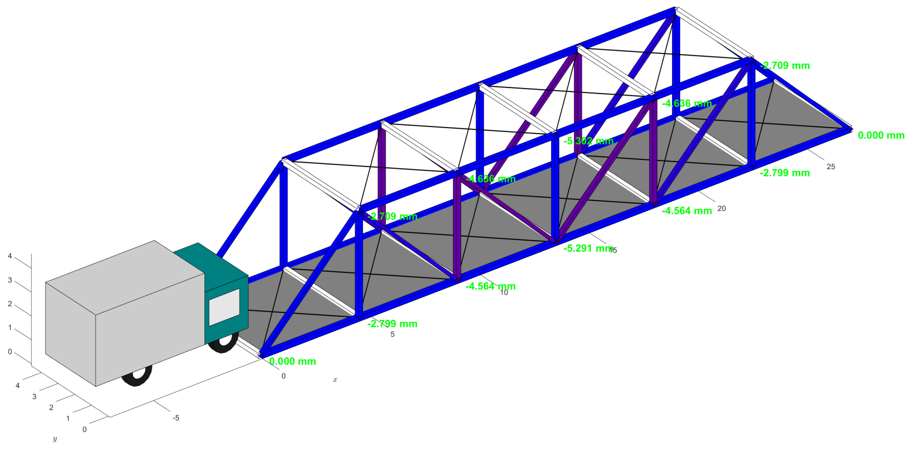
3.1 Colouring of Bars

The `patch` command takes a colour RGB triplet array as one of its inputs; the brief describes that the members should be coloured in such a way that a gradual transition from blue to red should exist for small to large forces respectively. By running the simulation once to find what the maximum force ever experienced in a member in the system was, and setting that to F_{big} ($\approx 367\,800\,\text{N}$). The colour red is `[1 0 0]` and blue is `[0 0 1]`. The colour of the truss is found by Eq. 9. When F_{truss} is small, ς is small, which only barely changes the colour slightly redder; whereas when the force in the bar is large, then ς is close to 1, and the red term goes close to one and the blue term becomes almost 0.

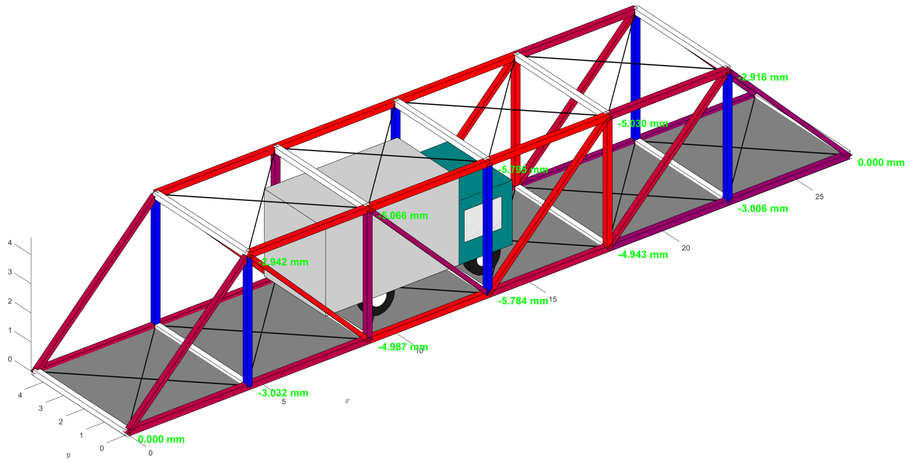
$$[\varsigma \;\; 0 \;\; 1-\varsigma] \text{ where } \varsigma = \frac{|F_{\text{truss}}|}{F_{\text{big}}} \tag{9}$$

However, due to the weight of the bridge being far greater than the lorry, the colours of the members hardly change as the lorry goes across the truss bridge with Eq. 9. Therefore, to greatly exaggerate the colour changes, the minimum force and maximum force experienced for each individual member was logged over the course of a dry run of the simulation. Then, the colour of a bar was based off of Eq. 9, taking into account its own max and min forces from the previous run. This gave Eq. 10. The +1 in the denominator mitigates fickle behaviour in some unruly members whose max and min forces are almost equal which resulted in flickering.

$$[\varsigma \;\; 0 \;\; 1-\varsigma] \text{ where } \varsigma = \frac{|F_{\mathtt{i}}| - |F_{\mathtt{i},\,\text{min}}|}{|F_{\mathtt{i},\,\text{max}}| - |F_{\mathtt{i},\,\text{min}}| + 1} \tag{10}$$



(a) Truck before crossing bridge



(b) Truck halfway across bridge

Figure 3: Still frames from animation

The `light` command was not utilised in the visualisation of the animation as the lighting affects the colour of the members in such a way as to obfuscate colour changes and the colour differences between members due to uneven lighting. The `VideoWriter` command was used to create the animation of the bridge and the lorry crossing it, along with `getframe`.

3.2 Deflection

The deflection data from `direct_stiffness.m` is passed to the overarching simulation script by the variable `U` in the form of a matrix, each entry associated with a node. The choice was made to display the data directly on the animation. The deflection data was multiplied by 1000 to convert from metres to millimetres. The `text` command was used to plot the `U(i)` deflection for the `i`-th node. Since the left and right sides of the bridge are equal in forces and thus deflections, the deflection data only had to be plotted once on one of the sides.

References

European Committee for Standardization (Apr. 2004). *Eurocode 3: Design of steel structures - Part 1-1: General rules and rules for buildings.* URL: `https://www.phd.eng.br/wp-content/uploads/2015/12/en.1993.1.1.2005.pdf`.

Dorf, Richard C. (1996). *The Engineering Handbook.* Boca Raton, Florida.

Aqua-Calc (2021). *Density of Concrete, Asphalt (material).* URL: `https://www.aqua-calc.com/page/density-table/substance/concrete-coma-and-blank-asphalt#:~:text=Concrete%5C%2C%5C%20Asphalt%5C%20weighs%5C%202.243%5C%20gram,inch%5C%20%5C%5Boz%5C%2Finch%5C%C2%5C%B3%5C%5D%5C%20.`.

MathWorks. (2020). *MATLAB* (Version 9.9; R2020b) [Computer software]. Natick, Massachusetts, USA: MathWorks. URL: `https://www.mathworks.com/products/matlab.html`.

Curran, Paul (2020). *Sim_21_10 — Systems of Linear Equations.* [Lecture].

McHugh, Frank (2021). *Truss Analysis.* URL: `https://www.mathworks.com/matlabcentral/fileexchange/38044-truss-analysis`.