AERO417: Further Aerostructural Analysis

Coursework: 01

By

Group number: 09

Full name	Student ID
1. Louis Allpress	201584369
2. Michael Parker	201560622
3. Jude Braden	201576768
4. Charles Tonge	201855397

School of Engineering

University of Liverpool

1 Table of Contents

1.0 Introduction

1.1 Boeing B-17 Landing Gear

The Boeing B-17 Flying Fortress was an iconic aircraft from the 1940s. Built in reaction to the start of World War 2, it was widely recognised for its range, durability and effectiveness for its varied mission roles.

Figure 1 Boeing B-17 Landing Gear (Moradi, 2024)

An essential component of the aircraft's design was its landing gear system. This was pivotal to ensure its operational reliability and safety for take-off and landing mission phases. Due to the B-17's unique mission requirements, heavy payload and challenging landing strips its landing gear faced considerable demands, understanding the mechanics and performance of the landing gear is critical.

The B-17's landing gear is a retractable tricycle design with shock absorbing structs designed to withstand the stress and strain of heavy loads.

1.2 Task Background and Solvers

This report investigates the B-17's landing gear to understand how the stress and strain of the structure withstands loads via various analysis. By analysing its design and performance, we can better understand how the structure performs under load.

Three types of analysis were performed on the landing gear:

1. A static analysis was performed to determine x and y displacements of each node as well as the stresses in each element.
2. A modal analysis was preformed to determine all natural frequencies and mode shapes of the truss setup.

3. A dynamic analysis was preformed to determine the transient response of all moving nodes over time.

Each analysis was undertaken with the implementation of different solving methods: a numerical approach and an ABAQUS approach. The numerical approach consisted of hand calculations (static analysis only) and the use of a MATLAB code. The ABAQUS approach is in reference to the use of the FEA software, ABAQUS. The motivation behind the use of different solving methods was to investigate the effectiveness and accuracy of each approach, with respect to each other.

2.0 Problem Overview

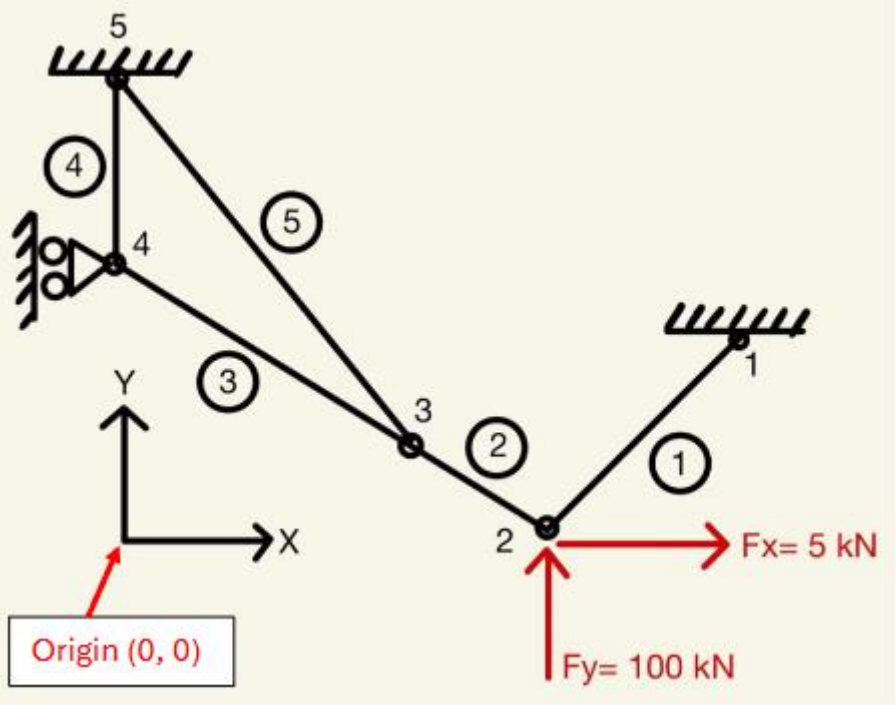


Figure 2 Idealised Landing Gear (Moradi, 2024)

As seen in figure 2, the idealised B-17 landing gear's truss structure is comprised of five nodes and fives elements. There are two external forces applied to the structrue: Fy, due to the weight of the plane, and Fx ,due to the plane moving on the ground. Each truss element is made of steel. The problem setup and parameters are listed in tables 1 and 2 respectively.

Table 1 Landing Gear Setup

Node	Coordinates from Origin (mm)	Boundary condition	Force (kN) (x, y) in direction of origin	Element Connections	Section Area (mm^2)
1	(2060,1000)	Pinned	(0,0)	(1,2)	1000
2	(1700,0)	Free	(5,100)	(2,3)	100
3	(1250,225)	Free	(0,0)	(3,4) (3,5)	100

4	(0,850)	Roller (Y-Direction free)	(0,0)	(4,5)	100
5	(0,1750)	Pinned	(0,0)	(3,5)	100

Table 2 Landing Gear Parameters

Parameters	Value
Modulus of Elasticity	209 GPa
Mass Density	7800 $\frac{Kg}{m^3}$
Poisson's Ratio	0.3

3.0 Analysis

3.1 Static Analysis

3.1.1 Numerical Approach (hand calculation):

The first step of the numerical approach was to determine the angle, $\theta^{(e)}$ between each element's local x axis, x' and the positive global x axis, x as well as the lengths of each element, $L^{(e)}$ This was done using trigonometry, the calculations for which are below.

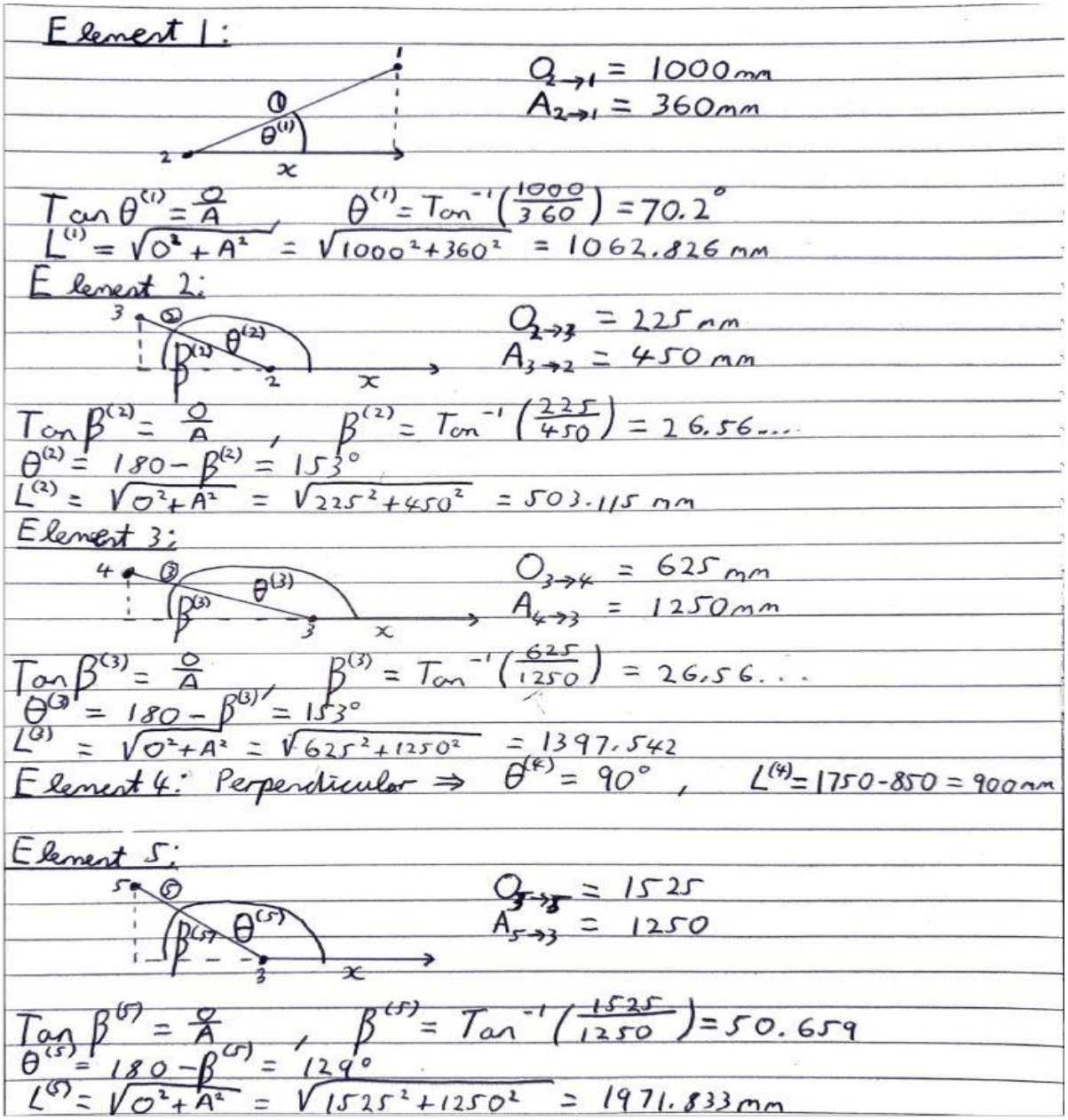


Figure 3 Hand calculations of relevant geometric features of truss setup

The values calculated in figure 1 with additional calculations needed to obtain each elements stiffness matrix is shown in table 2 below.

Table 3 Geometric data of truss system.

Element	θ (°)	C	S	C^2	S^2	CS	L (m)
1	70.2	0.3387	0.9409	0.1147	0.8853	0.31877	1.062826
2	153	-0.8910	0.4531	0.7939	0.2061	-0.4045	0.503115
3	153	-0.8910	0.4531	0.7939	0.2061	-0.4045	1.397542
4	90	0	1	0	1	0	0.9
5	129	-0.6293	0.7771	0.3960	0.6040	-0.4891	1.971833

The next step was to determine the local stiffness matrix, $[k^{(e)}]$ for each element, the equation for which is shown below.

$$[k^{(e)}] = \frac{AE}{L}\begin{bmatrix} C^2 & CS & -C^2 & -CS \\ CS & S^2 & -CS & -S^2 \\ -C^2 & -CS & C^2 & CS \\ -CS & -S^2 & CS & S^2 \end{bmatrix} \quad (1)$$

For consistency, a factor of 10^6 was taken out of each value of $\frac{AE}{L}$. A summary of the remaining values needed to calculate each element's local stiffness matrix is listed below.

Table 4 Summary of remaining values need to calculate each element's local stiffness matrix

Element	A (m^2)	E (GPa)	$\frac{AE}{L}$ (x 10^6)
1	0.001	209	196.64554687
2	0.0001	209	41.541119833
3	0.0001	209	14.95482783
4	0.0001	209	23.2̇
5	0.0001	209	10.59927488

By substituting the values found in tables 2 and 3 into equation 1, the local stiffness matrix of each element is found.

$$[k^{(1)}] = 10^6\begin{bmatrix} 22.555 & 62.671 & -22.555 & -62.671 \\ 62.671 & 174.090 & -62.671 & -174.090 \\ -22.555 & -62.671 & 22.555 & 62.671 \\ -62.671 & -174.090 & 62.672 & 174.090 \end{bmatrix} \quad (2)$$

$$[k^{(2)}] = 10^6\begin{bmatrix} 32.979 & -16.803 & -32.979 & 16.803 \\ -16.803 & 8.561 & 16.803 & -8.561 \\ -32.979 & 16.803 & 32.979 & -16.803 \\ 16.803 & -8.561 & -16.803 & 8.561 \end{bmatrix} \quad (3)$$

$$[k^{(3)}] = 10^6\begin{bmatrix} 11.873 & -6.049 & -11.873 & 6.049 \\ -6.049 & 3.082 & 6.049 & -3.082 \\ -11.873 & 6.049 & 11.873 & -6.049 \\ 6.049 & -3.082 & -6.049 & 3.082 \end{bmatrix} \quad (4)$$

$$[k^{(4)}] = 10^6\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 23.\dot{2} & 0 & -23.\dot{2} \\ 0 & 0 & 0 & 0 \\ 0 & -23.\dot{2} & 0 & 23.\dot{2} \end{bmatrix} \quad (5)$$

$$[k^{(5)}] = 10^6\begin{bmatrix} 4.197 & -5.184 & -4.197 & 5.184 \\ -5.184 & 6.402 & 5.184 & -6.402 \\ -4.197 & 5.184 & 4.197 & -5.184 \\ 5.184 & -6.402 & -5.184 & 6.402 \end{bmatrix} \quad (6)$$

By combining equations 2,3,4,5 and 6, the global stiffness matrix, [K] of the system is found.

$$[K] = 10^6 \begin{bmatrix} 22.555 & 62.671 & -22.555 & -62.671 & 0 & 0 & 0 & 0 & 0 & 0 \\ 62.671 & 174.090 & -62.671 & -174.090 & 0 & 0 & 0 & 0 & 0 & 0 \\ -22.555 & -62.671 & 55.534 & 45.868 & -32.979 & 16.803 & 0 & 0 & 0 & 0 \\ -62.671 & -174.090 & 45.868 & 182.803 & 16.803 & -8.561 & 0 & 0 & 0 & 0 \\ 0 & 0 & -32.979 & 16.803 & 49.049 & -28.036 & -11.873 & 6.049 & -4.197 & 5.184 \\ 0 & 0 & 16.803 & -8.561 & -28.036 & 18.045 & 6.049 & -3.082 & 5.184 & -6.402 \\ 0 & 0 & 0 & 0 & -11.873 & 6.049 & 11.873 & -6.049 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6.049 & -3.082 & -6.049 & 26.304 & 0 & -23.\dot{2} \\ 0 & 0 & 0 & 0 & -4.179 & 5.184 & 0 & 0 & 4.197 & -5.184 \\ 0 & 0 & 0 & 0 & 5.184 & -6.402 & 0 & -23.\dot{2} & -5.184 & 29.624 \end{bmatrix} \quad (7)$$

The global stiffness matrix (seen in equation 7) relates the global nodal forces, {F} to the global nodal displacements, {d} by the following equation:

$$\{F\} = [K]\{d\} \quad (8)$$

The next step is to apply the boundary conditions described in the problem overview. These are as follows: *will write in at the end*. By substituting the boundary conditions into equation 7 and substituting into equation 8, the following equation is obtained:

$$\begin{bmatrix} 5kN \\ 100kN \\ F_{x3} \\ F_{y3} \\ F_{y4} \end{bmatrix} = 10^6 \begin{bmatrix} 55.534 & 45.868 & -32.979 & 16.803 & 0 \\ 45.868 & 182.651 & 16.803 & -8.561 & 0 \\ -32.979 & 16.803 & 49.049 & -28.036 & 6.049 \\ 16.803 & -8.561 & -28.036 & 18.045 & -3.082 \\ 0 & 0 & 6.049 & -3.082 & 26.304 \end{bmatrix} \begin{bmatrix} u_2 \\ v_2 \\ u_3 \\ v_3 \\ v_4 \end{bmatrix} \quad (9)$$

The next step was to determine the nodal displacements by linearly solving the relevant linear equations considering the boundary conditions $(u_1, v_1, u_5, v_5 = 0)$ due to the nodes being fixed in the x and y directions. Which produces a reduced stiffness matrix and Force vector:

$$\text{Reduced stiffness matrix}[k] = \begin{bmatrix} 55.534 & 45.868 & -32.974 & 16.803 & 0 \\ 45.868 & 182.651 & 16.803 & -8.561 & 0 \\ -32.974 & 16.803 & 49.049 & -28.036 & 6.049 \\ 16.803 & -8.561 & -28.036 & -7.319 & -3.082 \\ 0 & 0 & 6.049 & -3.082 & 26.304 \end{bmatrix} 10^6$$

$$\text{Reduced Force vector } \{F\} = \begin{bmatrix} 50{,}000N \\ 100{,}000N \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Then the nodal placements $\{d\}$ are calculated using the following equation:

$$\{d\} = [k]^{-1}\{F\}$$

After solving the displacement values came out as:

$$\{d\} = \begin{bmatrix} u_2 \\ v_2 \\ u_3 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} -0.00192 \\ 0.00119 \\ -0.00162 \\ -0.0023 \\ 0.004 \end{bmatrix}$$

To calculate each element's stress the following formula is used. Considering the same geometric values from table 2.

$$\sigma = \frac{E}{L}[-c \; -s \; c \; s]\{d\}$$

$$E = 209 \times 10^9 (Pa)$$
$$L = length\ (m)$$
$$c = \cos(\theta)$$
$$s = \sin(\theta)$$
$$\{d\} = Nodal\ displacments:$$

Element 1:

$$\sigma = \frac{209 \times 10^9}{1.063}[-(-0.343) - (-0.939)\ -0.343\ -0.939]\begin{Bmatrix} 0 \\ 0 \\ -0.00192 \\ 0.00119 \end{Bmatrix}$$

$$\sigma = \frac{209 \times 10^9}{1.063}[0 - 0 + 0.343\ \times -0.00192 + 0.939 \times 0.00119]$$
$$\sigma = \ 9.229 \times \ 10^7 Pa$$

Element 2:

$$\sigma = \frac{209 \times 10^9}{0.503}[-(-0.6293) - (0.447)\ -0.894\ -0.447]\begin{Bmatrix} -0.00192 \\ 0.00119 \\ -0.00162 \\ -0.00023 \end{Bmatrix}$$

$$\sigma = 2.898 \times \ 10^7\ Pa$$

Element 3:

$$\sigma = \frac{209 \times 10^9}{1.398}[-(-0.894) - (0.447)\ -0.894\ \ 0.447]\begin{Bmatrix} -0.00162 \\ -0.00023 \\ 0 \\ 0.0004 \end{Bmatrix}$$

$$\sigma = 2.856\ \times \ 10^7\ Pa$$

Element 4:

$$\sigma = \frac{209 \times 10^9}{0.9}[0 - 1\ \ 0\ 1]\begin{Bmatrix} 0 \\ 0.00040 \\ 0 \\ 0 \end{Bmatrix}$$
$$\sigma = 1.277\ \times \ 10^7\ Pa$$

Element 5:

$$\sigma = \frac{209 \times 10^9}{1.971833}[-(-0.894) - (0.7771)\ -0.6293\ \ 0.7771]\begin{Bmatrix} -0 \\ 0 \\ -0.00162 \\ -0.00023 \end{Bmatrix}$$

$$\sigma = 0\ Pa$$

3.1.2 Numerical Approach (MATLAB):

The first step to complete the analysis was the development of the setup function, this function is used for all subsequent analysis. The first step was to calculate the length of each element using the PlaneTrussElementLength Function. A new function had to be

developed, Angle Calculator, that would calculate the angle of each element against the global coordinate system. For each element the stiffness matrix was found using the PlaneTrussElementStiffness Function. Finaly all the individual element stiffness where combined into one global stiffness matrix using the PlaneTrussAssemble function.

Currently the global stiffness matrix does not account for the boundary conditions on each node. Once the boundary conditions are included the global stiffness matrix becomes the reduced stiffness matrix. The setup of the problem is now completed, and the first analysis can begin.

The final step for the static problem was to determine the displacement matrix at each node and the subsequent stresses along each element, completed using the PlaneTrussElementStress function.

3.1.3 Abaqus Approach:

To analyse static approach in Abaqus a truss model that represents the idealised landing gear model shown in Figure 2 was produced. A general model was made for all three analysis approaches. A standard model file was created acting as a base for the entire structure. A part was created within the model, acting as a deformable 2D planar wire. A sketch based on the x and y coordinates of the landing gear. Once the part was created the steel material shown in Table 2 was assigned to the part with inputs for the material being mass density, Young's modulus and Poisson's ratio. The model was split into two separate sections due to the difference in cross-sectional area. Both sections had the same steel material assigned. An Assembly of the part was created with instance type being dependent. An initial was then created so that boundary conditions could be introduced. Shown in Table 1 nodes 1 and 5 are pinned were U1=U2=U2=0 therefore preventing them from translation. Nodes 2 and 3 are free therefore are not constrained. Node 4 is a roller constrained in the x and z-axis with translation possible in y-axis. Figure 4 shows the boundary conditions within Abaqus.

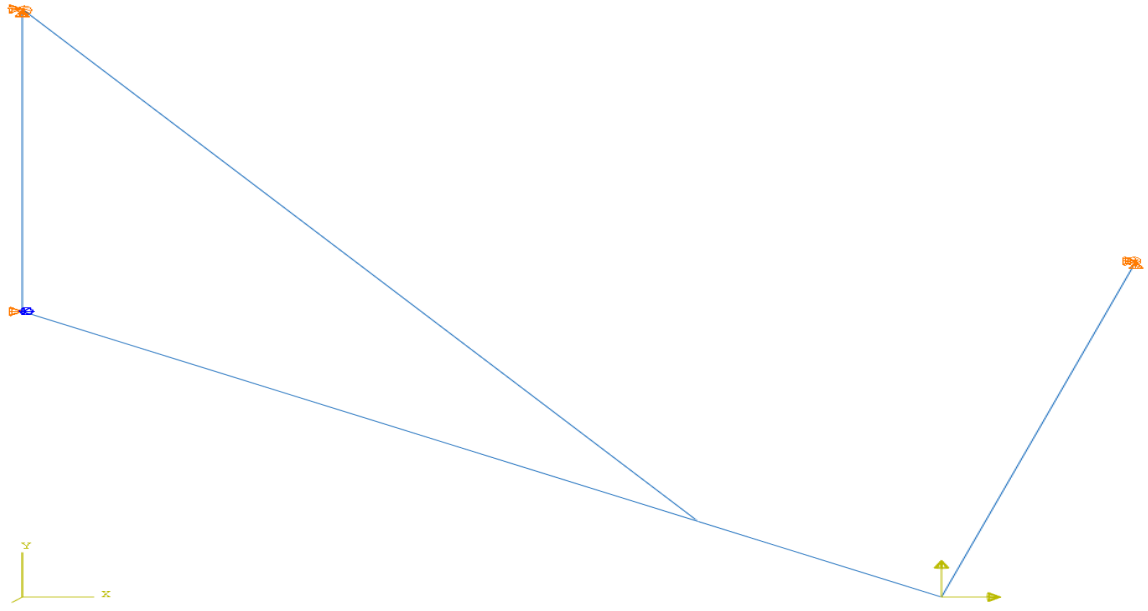


Figure 4 Abaqus boundary conditions

The steps mentioned above are the initial steps for all three analyses. After the boundary conditions were applied for the static analysis and additional step was created with the procedure type being static, general. A concentrated force was applied to node 2 as seen in Figure 2. The model was then meshed with the element type being a truss and the geometric order being linear. The edges of the mesh were meshed by number, with the number being 5 as there was 5 elements in the structure. Once the mesh was generated a static analysis was conducted by submitting a job through the analysis window. Once the job ran the data from the model could be extracted.

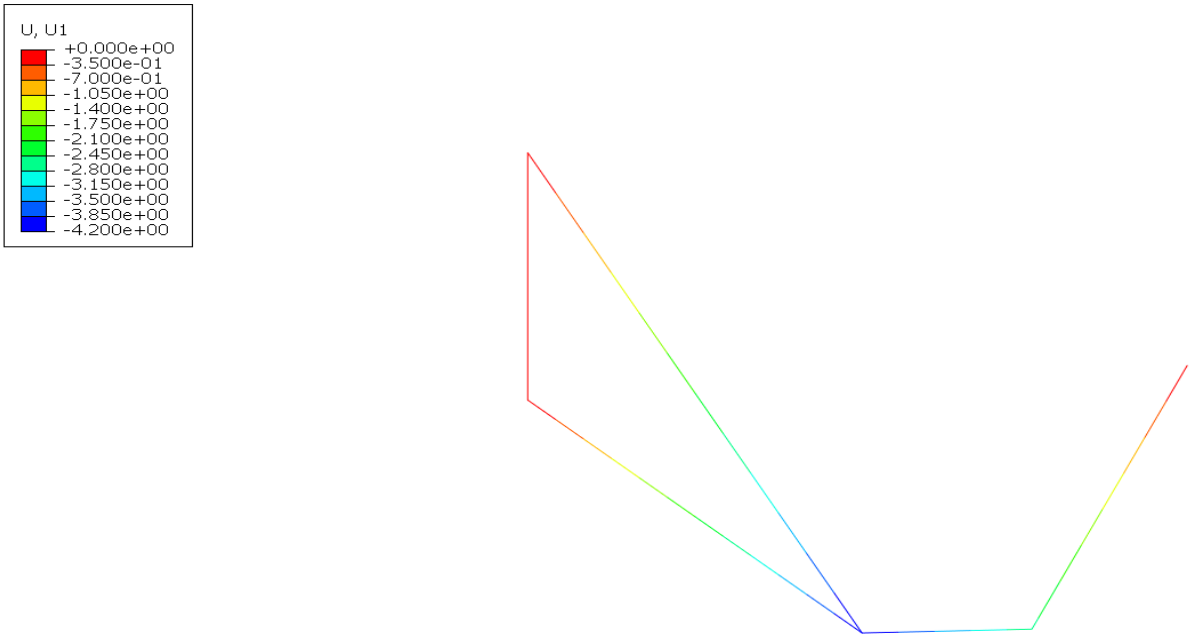


Figure 5 displacement in x-axis for static analysis

Shown in Figure 5 was the static analysis from Abaqus showing a contour plot of the displacement in the x-axis, using this model the displacement and stresses of the landing gear was produced.

3.2 Modal Analysis

3.2.1 Numerical Approach (MATLAB):

Following the setup from the previous problem, the next step was to calculate the lumped mass matrix of the system, this was completed by solving the mass matrix of each element before combining them into a global mass matrix. This matrix was then reduced, by the fixed and free degrees of freedom of the system. Using this reduced global mass matrix an eigen analysis can be completed, subsequently using the natural frequency equation $\sqrt{\frac{Eigen\ Values}{2\pi}}$ the natural frequency of each mode is found. The mode shape is the eigen vectors. And plotted against the non-deformed structure.

3.2.2 Abaqus Approach:

For this approach the steps to produce the idealised landing gear were described above in the static analysis, were the boundary conditions shown in Figure 4 are the same. Were it differs is after the boundary conditions were applied. For the modal analysis a

linear perturbation for frequency step is produced. Within the step the eigen solver was set to lanczos and the number set to 5 as there was 5 mode shapes. After that the same meshing procedure was followed from the static analysis. Once the analysis was completed data from the different modes was taken from the step/frame menu. Figure 6 shows mode 4 of the modal analysis.

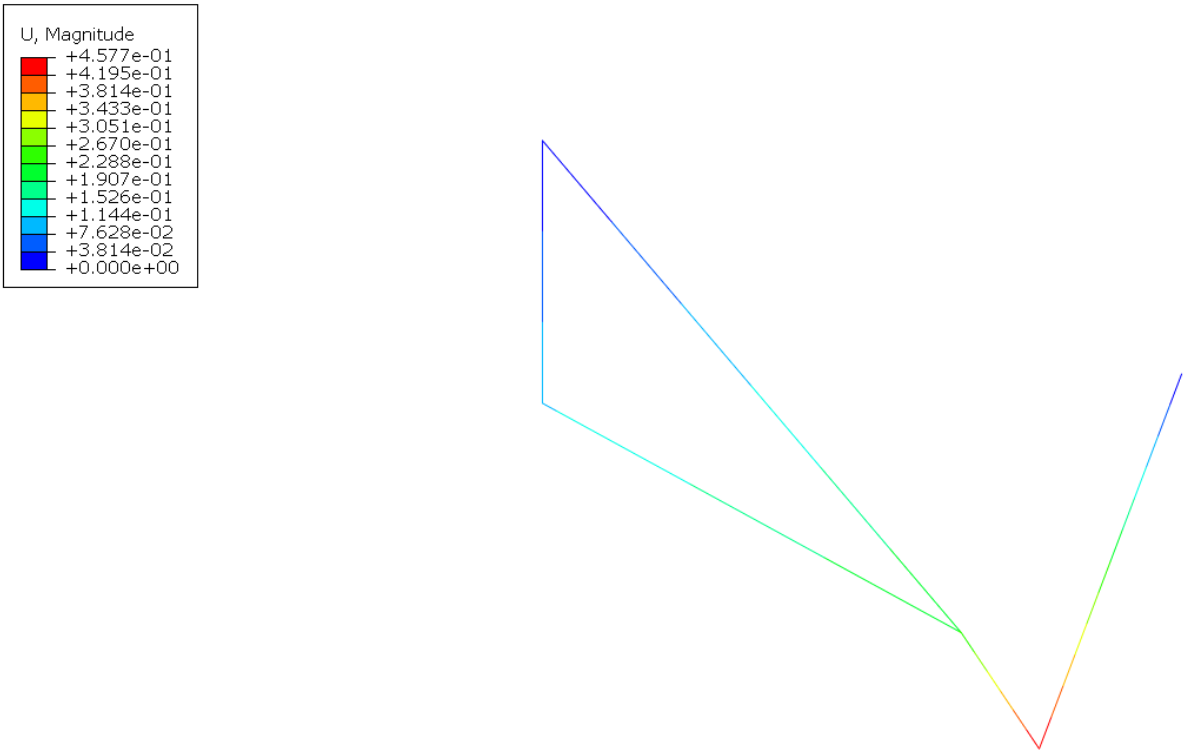


Figure 6 mode shape 4 of the modal analysis

3.3 Dynamic Analysis

3.3.1 Numerical Approach (MATLAB):

Using the above found global mass matrix and the equation to calculate the dynamic response of each node over time, the dynamic analysis of the structure could be completed. This was completed using an iterative process dividing the total time into sections, the size of these sections will impact the accuracy of results, however increasing the number of iterations exponentially increases the times vs accuracy of the simulation. For this simulation it was decided to match the number of iterations with that of the Abaqus model to match the fidelity of the test.

3.3.2 Abaqus Approach:

Like the modal analysis the same model setup was followed from the static analysis up to the boundary conditions shown in Figure 4. For the dynamic approach a general explicit dynamic step was created. Within the created step the time period was set at 1.4 seconds. Under the created step a load consisting of a concentrated force. After that the same meshing procedure was followed from the static analysis. Once the analysis was complete time history of the displacement was plotted to show the response shown in Figure 7.

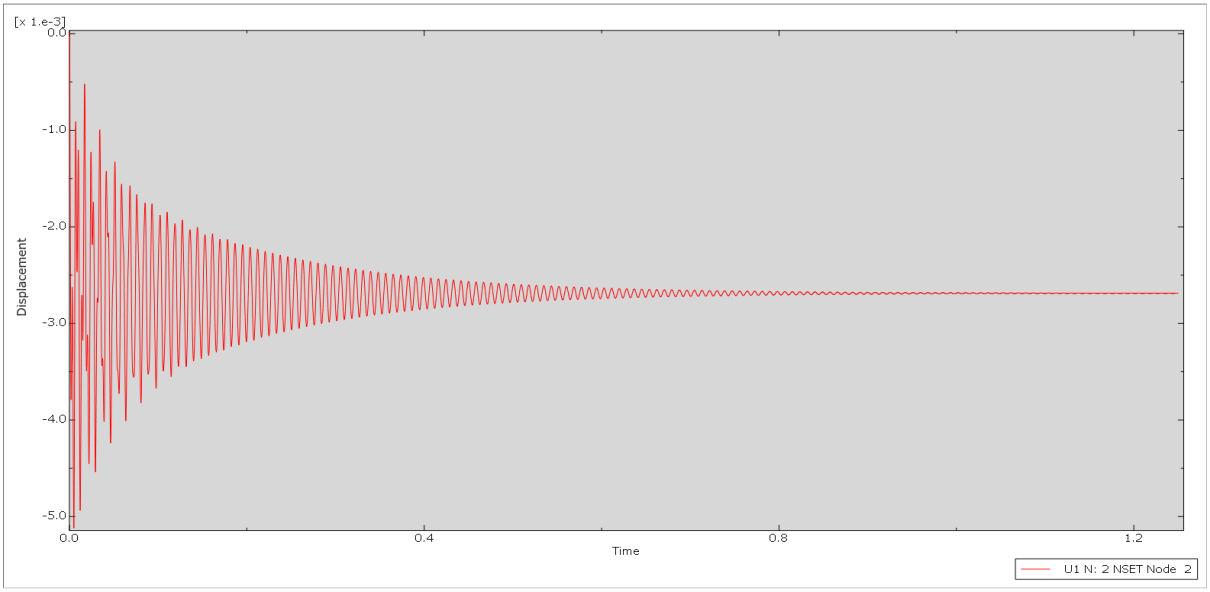


Figure 7 x-axis displacement for node 2

4. Results and Discussion

4.1 Static Analysis Results

The static analysis results shown in Table 5 show that there are discrepancies between the hand calculations and the MATLAB, whilst the MATLAB and the Abaqus results are identical. The discrepancies could have been due to rounding errors as large matrices calculations are more accurately computed using programs such as MATLAB as the risk of loosing number accuracy is reduced. Secondly if values are rounded all the way through the calculation these rounding errors can accumulate, something computer-based solutions do not struggle with. For the displacement results the largest error percentage for the hand calculations was node 3 in the x direction with a difference of 88.66%.

Table 5 Displacement Static Analysis Results

Displacement (m)	Node 1	Node 2	Node 3	Node 4	Node 5
Hand Calculations					
U1(x)	0	-1.92×10^3	-1.62×10^3	0	0
U2(y)	0	1.19×10^3	-2.30×10^3	0.40×10^4	0
MATLAB					
U1(x)	0	-2.5591×10^3	-4.2000×10^3	0	0
U2(y)	0	1.4203×10^3	-3.4426×10^3	0.5656×10^3	0
Abaqus					
U1(x)	0	-2.5591×10^3	-4.2000×10^3	0	0
U2(y)	0	1.4203×10^3	-3.4426×10^3	0.5656×10^3	0

For the static stress results shown in Table 6 there was slight differences between the hand calculations and the other two approaches with the biggest difference being element 4 with a difference of 2.49%

Table 6 Stress Static Analysis Results

Stress(S11) (Pa)	Element 1	Element 2	Element 3	Element 4	Element 5
Hand Calculations	9.229×10^7	2.898×10^7	2.856×10^7	1.277×10^7	0
MATLAB	9.232×10^7	2.937×10^7	2.9372×10^7	1.314×10^7	0
Abaqus	9.232×10^7	2.937×10^7	2.9372×10^7	1.314×10^7	0

4.2 Modal Analysis Results

Table 7 Natural frequencies modal analysis

Natural frequencies (Hz)	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5
MATLAB	123.76	287.12	843.79	1064.6	1145.8
Abaqus	123.76	287.12	843.79	1064.6	1145.8

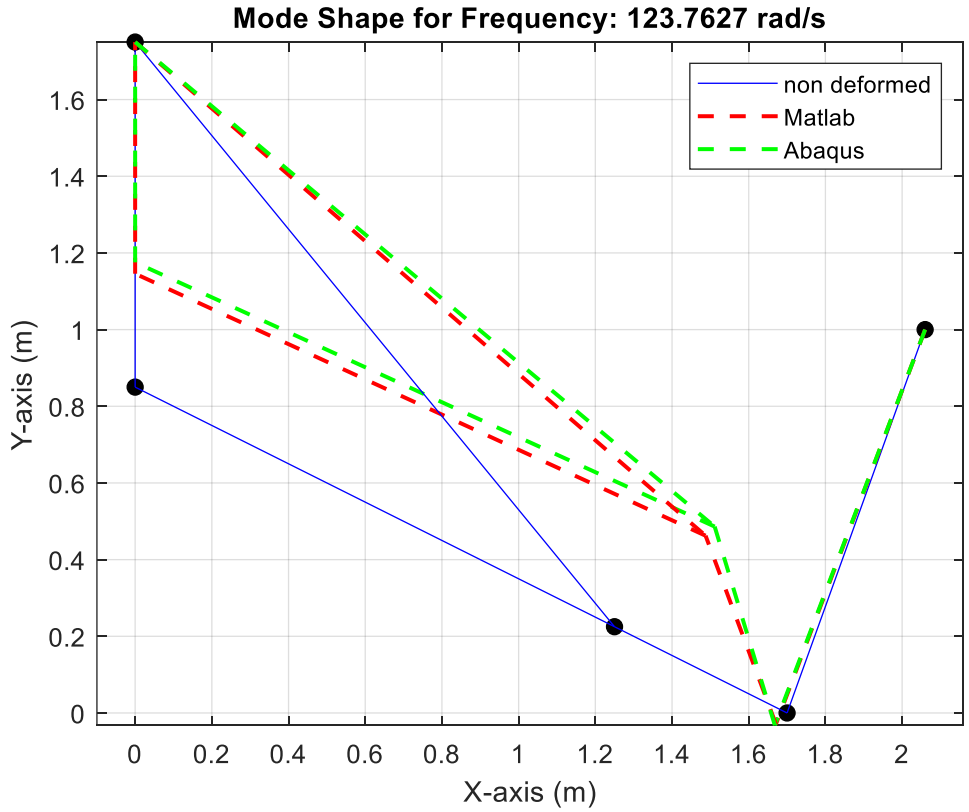


Figure 8 Mode shape for mode 1, 123.76 rad/s of the Landing gear

The modal analysis showed excellent corelation between the MATLAB results and Abaqus, with exact results, when looking at the natural frequencies. This shows the simplicity of the equations when calculating the natural frequencies. However the

mode shapes where not exact with an average percentage difference of $\sim 10\%$ across the dynamic modes. This is likely to be a result of the different solving approach taken by Abaqus when dealing with dynamic loads (Research Gate, 2018). The remainder of the mode shapes can be found in 7.2.4 Dynamic Response Plots

4.3 Dynamic Analysis Results

Table 8 Displacement Dynamic Analysis Results

Displacement (m)	Maximum Displacement (x, y)			Final Displacement (x,y)		
	Nodes			Nodes		
	2	3	4	2	3	4
MATLAB	0.0047, 0.0022	0.0081, 0.0093	0, 0.0011	-2.5591×10^3, 1.4203×10^3	-4.2000×10^3, -3.4426×10^3	0, 0.5656×10^3
Abaqus	0.0051, 0.0024	0.0086, 0.0098	0, 0.0012	-2.7×10^3, 1.5×10^3,	-4.5×10^3, -3.8×10^3,	0, 0.59×10^3

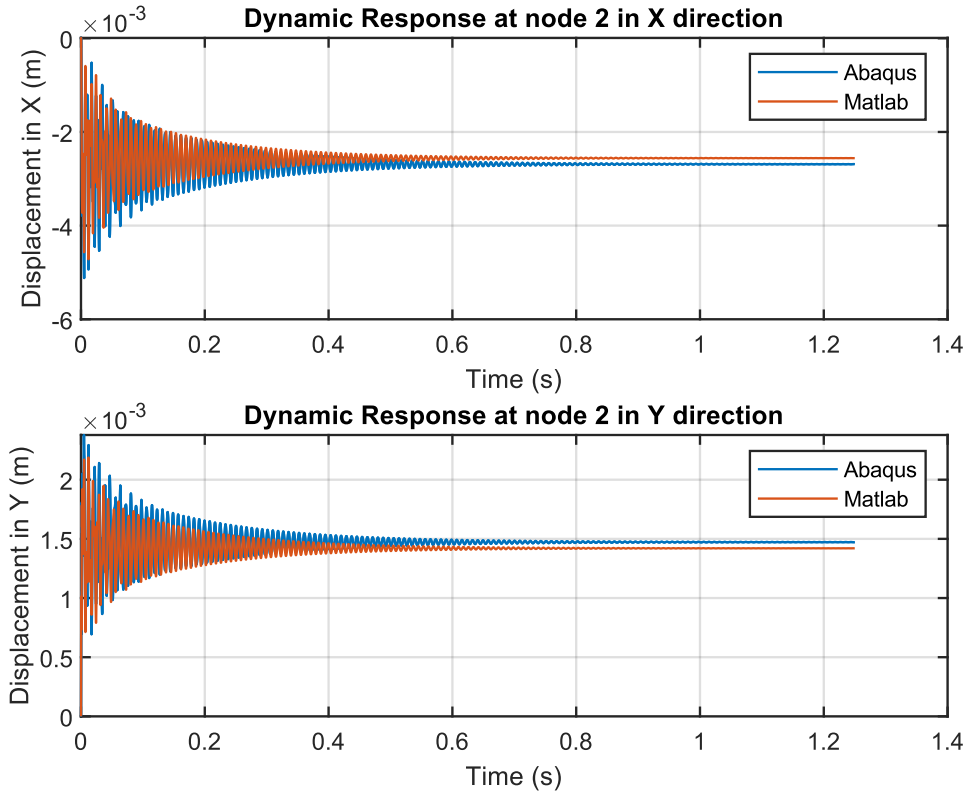


Figure 9 Dynamic Response at node 2

When looking at a dynamic response the key importance is the maximum displacement, does the node displace to any extreme that would lead to failure or a plastic deformation of one of the elements. This data suggests that none of the node's displacement too far with the maximum being 9.8mm. the Abaqus data overestimated all values for this analysis, secondly the final resting value of the dynamic response was

not that found within the static response. This increase in final displacement requires further investigation to understand. A feasible option is the system of equations used for the dynamic response, with the response varying with time the equations may be less accurate than those of the static response (Research Gate, 2018). The MATLAB final displacement is coincident with that of the static analysis. There is an average percentage difference of $\sim 8\%$ between the MATLAB and Abaqus. The remaining figures can be found in Appendix 7.3.2 Transient Response Plots

5.0 Conclusion

In conclusion, the comparison between MATLAB and Abaqus Finite Element approaches for analysing static, dynamic, and transient responses of a landing gear highlights the strengths and limitations of each technique.

The MATLAB approach allows flexibility and computational efficiency for simple models such as the landing gear. It allows for greater control over parameters, making it excellent for understanding the underlying mechanics, and the speed in which changes can be assessed is unparalleled . However, its applicability is limited when dealing with more complex geometries, and intricate boundary conditions.

Abaqus provides a comprehensive environment for detailed finite element analysis. Its advanced capabilities facilitate accurate modelling of complex structures under static, dynamic, and transient conditions. Abaqus is particularly advantageous for capturing non-linear behaviours and detailed stress distributions along individual elements, of which a MATLAB based approach struggles. However, the trade-offs include longer computation times and less flexibility for experimentation compared to MATLAB. There is a significant cost associated with finite element analysis that can be reduced through the use of MATLAB based approaches.
(Research Gate, 2018)

6.0 References

Moradi, D. A., 2024. *AERO417 coursework 1*. [Online]
[Accessed 1 November 2024].
Research Gate, 2018. *FEA: abaqus vs matlab, why results do not match?*. [Online]
Available at:
https://www.researchgate.net/post/FEA_abaqus_vs_matlab_why_results_do_not_match
[Accessed 09 November 2024].

7.0 Appendix

The raw data and a working MATLAB project are available they are omitted due to the raw size however are available upon request.

7.1 Static Analysis

7.1.1 Main

```
clc
clear

%Run the setup function
[E,A1,A2t5,L1,L2,L3,L4,L5,Theta1,Theta2,Theta3,Theta4,Theta5,K,k] = Setup();


% External force vector (applied forces)
f1 = [5000; 100000; 0;0;0]; % Force of 50 kN applied at Node 2 in the negative Y
direction

% Solve for displacements at free nodes (Node 2)
d = k \ f1;

% Assemble the complete displacement vector (including fixed nodes)
D = [ 0;0; d(1);d(2);d(3);d(4);0;d(5);0;0]; % Nodes 1, 4 and 5 are fixed, hence
their displacements are 0

% Compute reaction forces at all nodes
F = K * D;

% Extract displacements for each element to calculate stress
d1 = [D(3); D(4); D(1); D(2)]; % Displacement vector for element 1
d2 = [D(3); D(4); D(5); D(6)]; % Displacement vector for element 2
d3 = [D(5); D(6); D(7); D(8)]; % Displacement vector for element 3
d4 = [D(7); D(8); D(9); D(10)]; % Displacement vector for element 4
d5 = [D(5); D(6); D(9); D(10)]; % Displacement vector for element 5

% Calculate stress in each element
sigma1 = PlaneTrussElementStress(E, L1, Theta1, d1); % Stress in element 1
sigma2 = PlaneTrussElementStress(E, L2, Theta2, d2); % Stress in element 2
sigma3 = PlaneTrussElementStress(E, L3, Theta3, d3); % Stress in element 3
sigma4 = PlaneTrussElementStress(E, L4, Theta4, d4); % Stress in element 4
sigma5 = PlaneTrussElementStress(E, L5, Theta5, d5); % Stress in element 5

% Display results
fprintf('---------- Nodal Displacements ----------\n');
fprintf('Node  Displacement X (m)  Displacement Y (m)\n');
for i = 1:5
    fprintf('%4d  %16.3e  %16.3e\n', i, D(2*i-1), D(2*i));
end

fprintf('\n---------- Reaction Forces ----------\n');
fprintf('Node  Reaction Force X (N)  Reaction Force Y (N)\n');
for i = 1:5
    fprintf('%4d  %16.3e  %16.3e\n', i, F(2*i-1), F(2*i));
end
```

```
fprintf('\n---------- Element Stresses ----------\n');
fprintf('Element  Stress (Pa)\n');
fprintf('%7d  %16.3e\n', 1, sigma1);
fprintf('%7d  %16.3e\n', 2, sigma2);
fprintf('%7d  %16.3e\n', 3, sigma3);
fprintf('%7d  %16.3e\n', 4, sigma4);
fprintf('%7d  %16.3e\n', 5, sigma5);

% End of script
```

7.1.2 Setup Function

```
function [E,A1,A2t5,L1,L2,L3,L4,L5,Theta1,Theta2,Theta3,Theta4,Theta5,K,k] =
Setup()

    % Material properties and cross-sectional area
    E = 209e9;    % Young's Modulus (Pa)
    A1 = 1e-3;    % Cross-sectional area (m^2)
    A2t5 = 1e-4;  % Cross-sectional area (m^2)

    %Node Coordinates

    N1 = [2060,1000];
    N2 = [1700,0];
    N3 = [1250,225];
    N4 = [0,850];
    N5 = [0,1750];


    % Lengths of the truss elements
    L1 = (PlaneTrussElementLength(N1(1),N1(2),N2(1),N2(2))) / 1e3;  % Length of
elements 1 (m)
    L2 = (PlaneTrussElementLength(N2(1), N2(2), N3(1), N3(2))) / 1e3; % Length of
element 2 (m)
    L3 = (PlaneTrussElementLength(N3(1), N3(2), N4(1), N4(2))) / 1e3; % Length of
element 3 (m)
    L4 = (PlaneTrussElementLength(N4(1), N4(2), N5(1), N5(2))) / 1e3; % Length of
element 4 (m)
    L5 = (PlaneTrussElementLength(N5(1), N5(2), N3(1), N3(2))) / 1e3; % Length of
element 5 (m)

    % Angles of truss elements

    Theta1 = AngleCalculator(2060,1000,1700,0);  % Angle of elements 1 (degrees)
    Theta2 = AngleCalculator(1700, 0, 1250, 225); % Angle of element 2 (degrees)
    Theta3 = AngleCalculator(1250, 225, 0, 850); % Angle of element 3 (degrees)
    Theta4 = AngleCalculator(0, 850, 0, 1750); % Angle of element 4 (degrees)
    Theta5 = AngleCalculator(0, 1750, 1250, 225); % Angle of element 5 (degrees)

    % Element stiffness matrices for the three elements
    k1 = PlaneTrussElementStiffness(E, A1, L1, Theta1);   % Stiffness of element 1
    k2 = PlaneTrussElementStiffness(E, A2t5, L2, Theta2); % Stiffness of element 2
    k3 = PlaneTrussElementStiffness(E, A2t5, L3, Theta3); % Stiffness of element 3
    k4 = PlaneTrussElementStiffness(E, A2t5, L4, Theta4); % Stiffness of element 4
    k5 = PlaneTrussElementStiffness(E, A2t5, L5, Theta5); % Stiffness of element 5

    % Initialise global stiffness matrix (8x8 since we have 4 nodes with 2 DOF
each)
```

```
    K = zeros(10, 10);

    % Assembly of global stiffness matrix from element stiffness matrices
    K = PlaneTrussAssemble(K, k1, 1, 2); % Assemble element 1 between nodes 1 and
2
    K = PlaneTrussAssemble(K, k2, 2, 3); % Assemble element 2 between nodes 2 and
3
    K = PlaneTrussAssemble(K, k3, 3, 4); % Assemble element 3 between nodes 3 and
4
    K = PlaneTrussAssemble(K, k4, 4, 5); % Assemble element 4 between nodes 4 and
5
    K = PlaneTrussAssemble(K, k5, 3, 5); % Assemble element 5 between nodes 3 and
5

    % Extract the reduced stiffness matrix for free degrees of freedom (DOF)
    k = K([3,4,5,6,8] , [3,4,5,6,8]); % Node 2 is free (not fixed)
end
```

7.1.3 Plane Truss Element Length

```
function y = PlaneTrussElementLength(x1,y1,x2,y2)
    %PlaneTrussElementLength   This function returns the length of the
    %                          plane truss element whose first node has
    %                          coordinates (x1,y1) and second node has
    %                          coordinates (x2,y2).
    y = sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1));
end
```

7.1.4 Angle Calculator

```
function output = AngleCalculator(x1,y1,x2,y2)

    %Calculate x and y length from coordinates
    xlength = (x1-x2);
    ylength = (y1-y2);

    %determine the angle using arctan
    output = atand(ylength/xlength);

    %if the value is below 0 sub 180 to make positive
    if output < 0
        output = 180 + output;
    end

end
```

7.1.5 Plane Truss Element Stiffness

```
function y = PlaneTrussElementStiffness(E,A,L, theta)
%PlaneTrussElementStiffness   This function returns the element
%                             stiffness matrix for a plane truss
%                             element with modulus of elasticity E,
%                             cross-sectional area A, length L, and
%                             angle theta (in degrees).
%                             The size of the element stiffness
%                             matrix is 4 x 4.
x = theta*pi/180;
C = cos(x);
S = sin(x);
```

```
y = E*A/L*[C*C C*S -C*C -C*S ; C*S S*S -C*S -S*S ;
   -C*C -C*S C*C C*S ; -C*S -S*S C*S S*S];
```

7.1.6 Plane Truss Assemble

```
function y = PlaneTrussAssemble(K,k,i,j)
    %PlaneTrussAssemble   This function assembles the element stiffness
    %                     matrix k of the plane truss element with nodes
    %                     i and j into the global stiffness matrix K.
    %                     This function returns the global stiffness
    %                     matrix K after the element stiffness matrix
    %                     k is assembled.
    K(2*i-1,2*i-1) = K(2*i-1,2*i-1) + k(1,1);
    K(2*i-1,2*i) = K(2*i-1,2*i) + k(1,2);
    K(2*i-1,2*j-1) = K(2*i-1,2*j-1) + k(1,3);
    K(2*i-1,2*j) = K(2*i-1,2*j) + k(1,4);
    K(2*i,2*i-1) = K(2*i,2*i-1) + k(2,1);
    K(2*i,2*i) = K(2*i,2*i) + k(2,2);
    K(2*i,2*j-1) = K(2*i,2*j-1) + k(2,3);
    K(2*i,2*j) = K(2*i,2*j) + k(2,4);
    K(2*j-1,2*i-1) = K(2*j-1,2*i-1) + k(3,1);
    K(2*j-1,2*i) = K(2*j-1,2*i) + k(3,2);
    K(2*j-1,2*j-1) = K(2*j-1,2*j-1) + k(3,3);
    K(2*j-1,2*j) = K(2*j-1,2*j) + k(3,4);
    K(2*j,2*i-1) = K(2*j,2*i-1) + k(4,1);
    K(2*j,2*i) = K(2*j,2*i) + k(4,2);
    K(2*j,2*j-1) = K(2*j,2*j-1) + k(4,3);
    K(2*j,2*j) = K(2*j,2*j) + k(4,4);
    y = K;
end
```

7.1.7 Plane Truss Element stress

```
function y = PlaneTrussElementStress(E,L,theta,d)
    %PlaneTrussElementStress   This function returns the element stress
    %                          given the modulus of elasticity E, the
    %                          the length L, the angle theta (in
    %                          degrees), and the element nodal
    %                          displacement vector d.
    x = theta * pi/180;
    C = cos(x);
    S = sin(x);

    y = -E/L*[-C -S C S]* d;
end
```

7.2 Modal Analysis

7.2.1 Main

```
Abaqus_Mode = readtable('Abaqus Mode Shape.xlsx');

%run the setup function
[E,A1,A2t5,L1,L2,L3,L4,L5,Theta1,Theta2,Theta3,Theta4,Theta5,K,k] = Setup();
rho = 7800;
```

```
%Node Geometry setup
N1 = [2060,1000];
N2 = [1700,0];
N3 = [1250,225];
N4 = [0,850];
N5 = [0,1750];

nodes = [N1; N2 ;N3; N4; N5]; % Node coordinates
elements = [1 2; 2 3; 3 4;4 5;5 3]; % Element connectivity
numNodes = size(nodes, 1);
numElements = size(elements, 1);



% Element Mass matrices for the 5 elements
m1 = PlaneTrussElementMass(rho,A1,L1);
m2 = PlaneTrussElementMass(rho,A2t5,L2);
m3 = PlaneTrussElementMass(rho,A2t5,L3);
m4 = PlaneTrussElementMass(rho,A2t5,L4);
m5 = PlaneTrussElementMass(rho,A2t5,L5);

% Initialise global Mass matrix (10x10 since we have 5 nodes with 2 DOF each)
M = zeros(10, 10);

% Assembly of global stiffness matrix from element stiffness matrices
M = PlaneTrussMassAssemble(M, m1, 1, 2); % Assemble element 1 between nodes 1 and
2
M = PlaneTrussMassAssemble(M, m2, 2, 3); % Assemble element 2 between nodes 2 and
3
M = PlaneTrussMassAssemble(M, m3, 3, 4); % Assemble element 3 between nodes 3 and
4
M = PlaneTrussMassAssemble(M, m4, 4, 5); % Assemble element 4 between nodes 4 and
5
M = PlaneTrussMassAssemble(M, m5, 3, 5); % Assemble element 5 between nodes 3 and
5

% Apply boundary conditions (constrain nodes 2, 3, and 4)
fixedDofs = [1 2 7 9 10]; % Constrained DOFs
freeDofs = setdiff(1:2*numNodes, fixedDofs);

% Solve the eigenvalue problem for natural frequencies
[phi, omega2] = eig(K(freeDofs, freeDofs), M(freeDofs, freeDofs));
natural_frequencies = sqrt(diag(omega2)) / (2*pi) ;


% Plot mode shapes
for i = 1:height(natural_frequencies)
    figure;
    mode_shape = phi(:, i);
    mode_shape_Abaqus = table2array(Abaqus_Mode(:,i));
    mode_shape([1,5]) = 0;
    %collect the mode shapes from abaqu and calculated
    plot((nodes(:, 1))/1e3, (nodes(:, 2))/1e3, 'ko', 'MarkerFaceColor', 'k'); hold
on;
    for j = 1:numElements
        n1 = elements(j, 1);
        n2 = elements(j, 2);
        scaleFactor = 0.5; % Adjust this factor for better visualization
```

```
        scaleFactor2 = 0.5;
        scaledShape = mode_shape * scaleFactor;
        scaledShapeAbaqus = mode_shape_Abaqus * scaleFactor2;
        %create the scaled shape of the modes
        plot([(nodes(n1, 1))/1e3 (nodes(n2, 1))/1e3], [(nodes(n1, 2))/1e3
(nodes(n2, 2))/1e3], 'b-');
        % Calculate the deformed positions MATLAB
        x1 = (nodes(n1, 1))/1e3 + scaledShape(n1);
        y1 = (nodes(n1, 2))/1e3 + scaledShape(n1);
        x2 = (nodes(n2, 1))/1e3 + scaledShape(n2);
        y2 = (nodes(n2, 2))/1e3 + scaledShape(n2);
          % Calculate the deformed positions Abaqus
        xA1 = (nodes(n1, 1))/1e3 + (scaledShapeAbaqus(n1) );
        yA1 = (nodes(n1, 2))/1e3 + (scaledShapeAbaqus(n1) );
        xA2 = (nodes(n2, 1))/1e3 + (scaledShapeAbaqus(n2) );
        yA2 = (nodes(n2, 2))/1e3 + (scaledShapeAbaqus(n2) );


        %for the pinned nodes do not allow movement
        if n1 == 4
             x1 = (nodes(n1, 1))/1e3;
             xA1 = (nodes(n1, 1))/1e3;
        elseif n2 == 4
            x2 = (nodes(n2, 1))/1e3;
            xA2 = (nodes(n2, 1))/1e3;
        end



        plot([x1, x2], [y1, y2], 'r--', 'LineWidth', 1.5); % Deformed shape Matlab
        plot([xA1, xA2], [yA1, yA2], 'g--', 'LineWidth', 1.5); % Deformed shape
Abaqus
        legend('','non deformed','Matlab','Abaqus')

    end
    %titles for graphs and labels
    title(['Mode Shape for Frequency: ' num2str(natural_frequencies(i)) '
rad/s']);
    xlabel('X-axis (m)');
    ylabel('Y-axis (m)');
    axis equal;
    grid on;
end

%display results
fprintf('\n---------- Natural Frequency ----------\n');
fprintf('Natural Frequency (Hz)\n');
fprintf('%7d  %16.3e\n', 1, natural_frequencies(1));
fprintf('%7d  %16.3e\n', 2, natural_frequencies(2));
fprintf('%7d  %16.3e\n', 3, natural_frequencies(3));
fprintf('%7d  %16.3e\n', 4, natural_frequencies(4));
fprintf('%7d  %16.3e\n', 5, natural_frequencies(5));
```

7.2.2 Plane Truss Element Mass

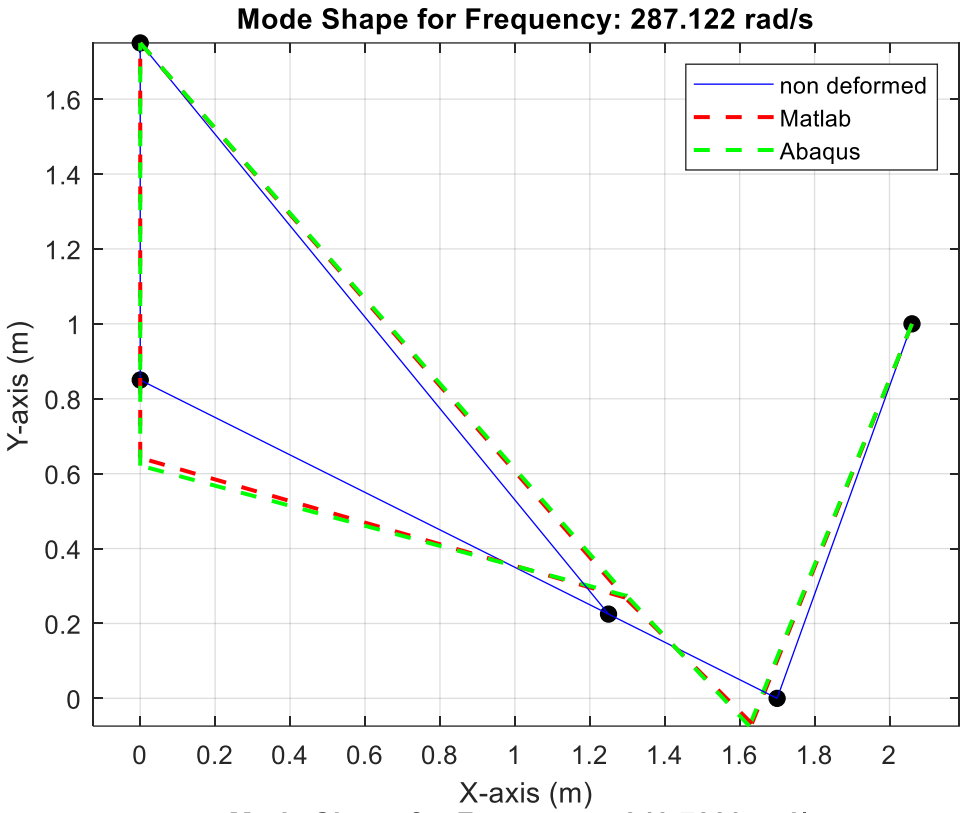
```
function y = PlaneTrussElementMass(rho,A,L)
    %PlaneTrussElementStiffness   This function returns the element
    %                             lumped mass matrix for a plane truss
```

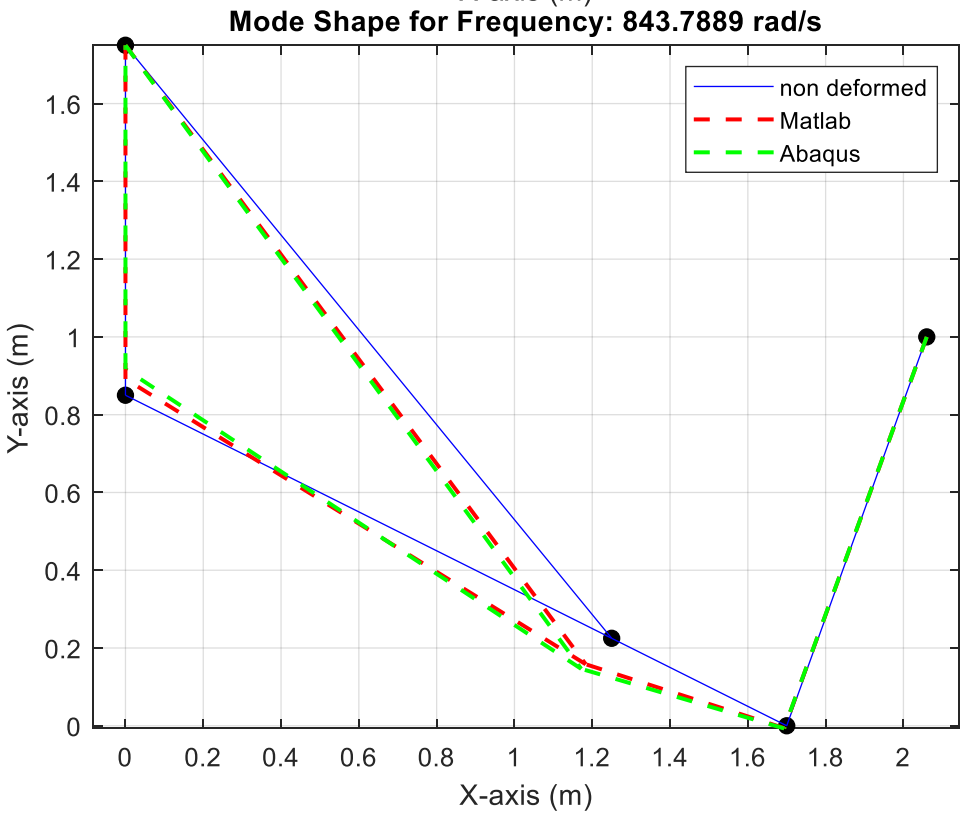
```
    %                               element with a Density rho,
    %                               cross-sectional area A, length L, and
    %                               The size of the element mass
    %                               matrix is 4 x 4.


    y = (rho * A * L / 2) * [1, 0,0,0;
                             0, 1,0,0;
                             0,0,1, 0;
                             0, 0,0,1];


end
```

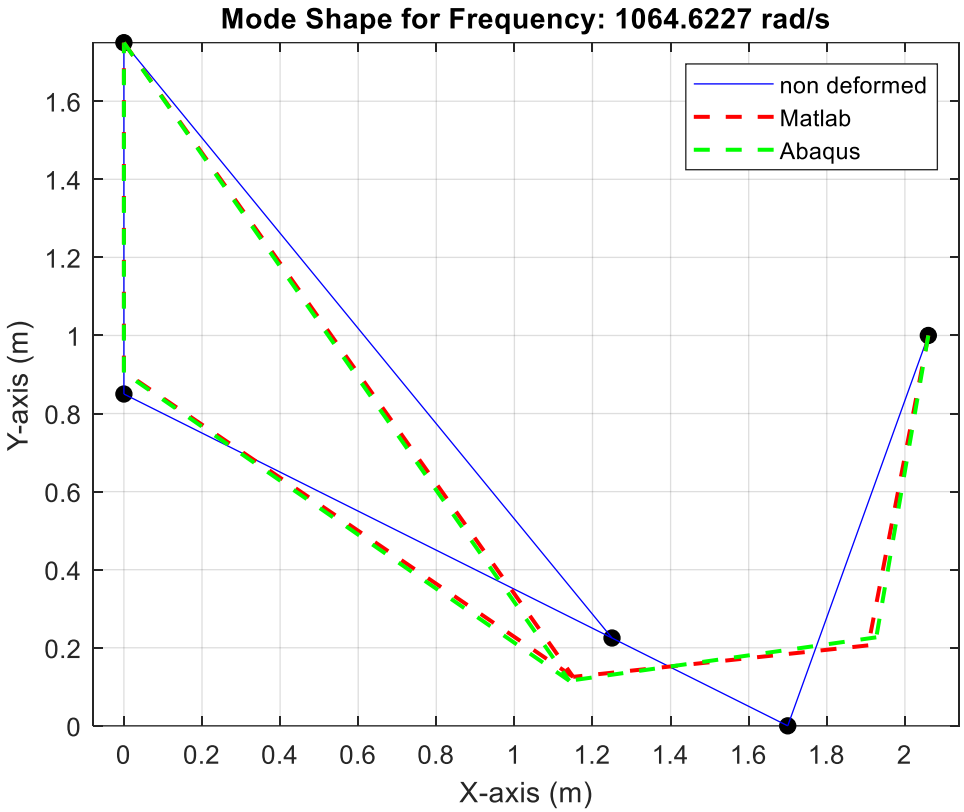
7.2.3 Plane Truss Mass Assemble

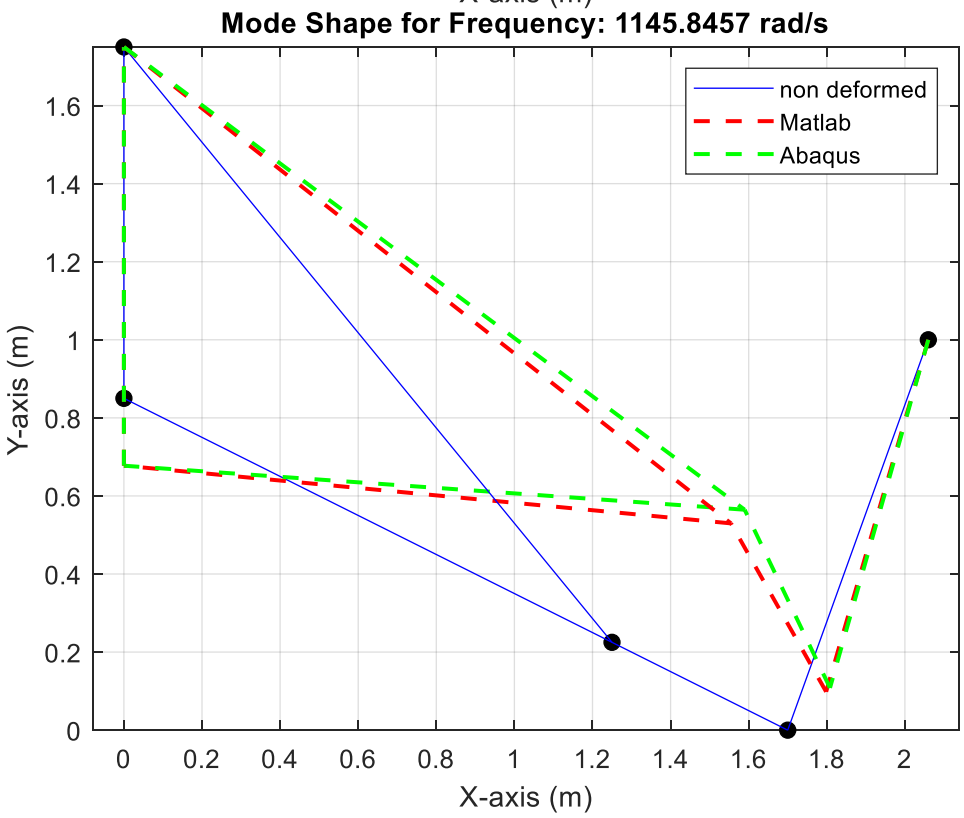
```
function y = PlaneTrussMassAssemble(M,m,i,j)
    %PlaneTrussAssemble   This function assembles the element stiffness
    %                     matrix m of the plane truss element with nodes
    %                     i and j into the global stiffness matrix M.
    %                     This function returns the global stiffness
    %                     matrix M after the element stiffness matrix
    %                     m is assembled.
    M(2*i-1,2*i-1) = M(2*i-1,2*i-1) + m(1,1);
    M(2*i-1,2*i) = M(2*i-1,2*i) + m(1,2);
    M(2*i-1,2*j-1) = M(2*i-1,2*j-1) + m(1,3);
    M(2*i-1,2*j) = M(2*i-1,2*j) + m(1,4);
    M(2*i,2*i-1) = M(2*i,2*i-1) + m(2,1);
    M(2*i,2*i) = M(2*i,2*i) + m(2,2);
    M(2*i,2*j-1) = M(2*i,2*j-1) + m(2,3);
    M(2*i,2*j) = M(2*i,2*j) + m(2,4);
    M(2*j-1,2*i-1) = M(2*j-1,2*i-1) + m(3,1);
    M(2*j-1,2*i) = M(2*j-1,2*i) + m(3,2);
    M(2*j-1,2*j-1) = M(2*j-1,2*j-1) + m(3,3);
    M(2*j-1,2*j) = M(2*j-1,2*j) + m(3,4);
    M(2*j,2*i-1) = M(2*j,2*i-1) + m(4,1);
    M(2*j,2*i) = M(2*j,2*i) + m(4,2);
    M(2*j,2*j-1) = M(2*j,2*j-1) + m(4,3);
    M(2*j,2*j) = M(2*j,2*j) + m(4,4);
    y = M;
end
```

7.2.4 Dynamic Response Plots









7.3 Transient Response

7.3.1 Main

```
clear, close all
a = readtable('noderesults.txt');
```

```
time_Abaqus = table2array(a(:,1));
u1node2 = table2array(a(:,2));
u1node3 = table2array(a(:,3));
u2node2 = table2array(a(:,5));
u2node3 = table2array(a(:,6));
u2node4 = table2array(a(:,7));


%Run the setup function
[E,A1,A2t5,L1,L2,L3,L4,L5,Theta1,Theta2,Theta3,Theta4,Theta5,K,k] = Setup();
rho = 7800;

N1 = [2060,1000];
N2 = [1700,0];
N3 = [1250,225];
N4 = [0,850];
N5 = [0,1750];

% Define the nodes and elements
nodes = [N1; N2 ;N3; N4; N5]; % Node coordinates
elements = [1 2; 2 3; 3 4;4 5;5 3]; % Element connectivity

% Number of nodes and elements
numNodes = size(nodes, 1);
numElements = size(elements, 1);


% Element Mass matrices for the 5 elements
m1 = PlaneTrussElementMass(rho,A1,L1);
m2 = PlaneTrussElementMass(rho,A2t5,L2);
m3 = PlaneTrussElementMass(rho,A2t5,L3);
m4 = PlaneTrussElementMass(rho,A2t5,L4);
m5 = PlaneTrussElementMass(rho,A2t5,L5);

% Initialise global Mass matrix (10x10 since we have 5 nodes with 2 DOF each)
M = zeros(10, 10);

% Assembly of global stiffness matrix from element stiffness matrices
M = PlaneTrussMassAssemble(M, m1, 1, 2); % Assemble element 1 between nodes 1 and
2
M = PlaneTrussMassAssemble(M, m2, 2, 3); % Assemble element 2 between nodes 2 and
3
M = PlaneTrussMassAssemble(M, m3, 3, 4); % Assemble element 3 between nodes 3 and
4
M = PlaneTrussMassAssemble(M, m4, 4, 5); % Assemble element 4 between nodes 4 and
5
M = PlaneTrussMassAssemble(M, m5, 3, 5); % Assemble element 5 between nodes 3 and
5

% Apply boundary conditions (constrain nodes 2, 3, and 4)
fixedDofs = [1 2 7 9 10]; % Constrained DOFs
freeDofs = setdiff(1:2*numNodes, fixedDofs);

% Solve the eigenvalue problem for natural frequencies
K = K(freeDofs, freeDofs);
M = M(freeDofs, freeDofs);
F = [5000; 100000; 0;0;0];
```

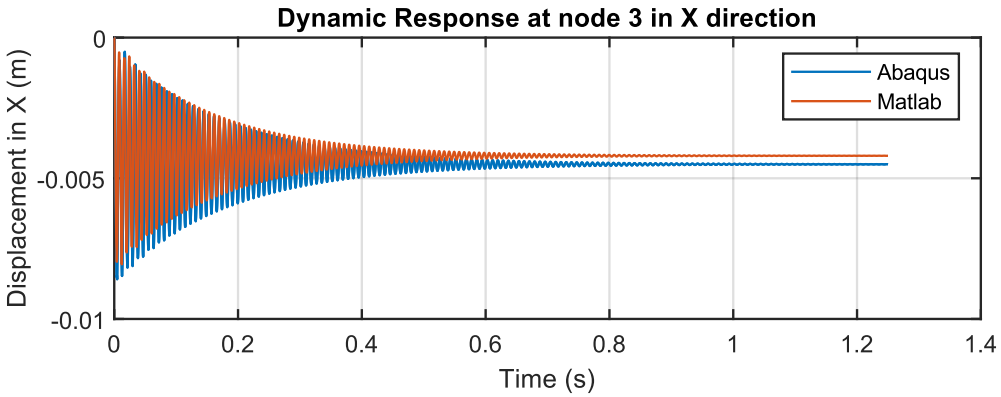
```
dt = 1.25 / (13946 - 1);
t_total = 1.25;
t = 0:dt:t_total;
n_steps = length(t);

u = zeros(5,n_steps);
v = zeros(5,1);
a = inv(M) * (F - K * u(:,1));
u(:,2) = (dt^2/2) * a;


alpha = 0.005;
beta = 0.00002;

C_reduced = alpha * M + beta * K;

for i = 2:n_steps-1
    u(:,i+1) = inv(M) * (F - K * u(:,i) - C_reduced * (u(:,i) - u(:,i-1)) / dt) *
dt^2 + 2*u(:,i) - u(:,i-1);
end


figure(1)
subplot(2,1,1)
plot(time_Abaqus,u1node2)
grid on
hold on
grid on
plot(t,u(1,:));
xlabel('Time (s)')
ylabel('Displacement in X (m)')
title('Dynamic Response at node 2 in X direction')
legend('Abaqus','Matlab')
subplot(2,1,2)
plot(time_Abaqus,u2node2)
hold on
grid on
plot(t,u(2,:));
legend('Abaqus','Matlab')
xlabel('Time (s)')
ylabel('Displacement in Y (m)')
title('Dynamic Response at node 2 in Y direction')

figure(2)
subplot(2,1,1)
plot(time_Abaqus,u1node3)
grid on
hold on
grid on
plot(t,u(3,:));
legend('Abaqus','Matlab')
xlabel('Time (s)')
ylabel('Displacement in X (m)')
title('Dynamic Response at node 3 in X direction')


subplot(2,1,2)
plot(time_Abaqus,u2node3)
grid on
```

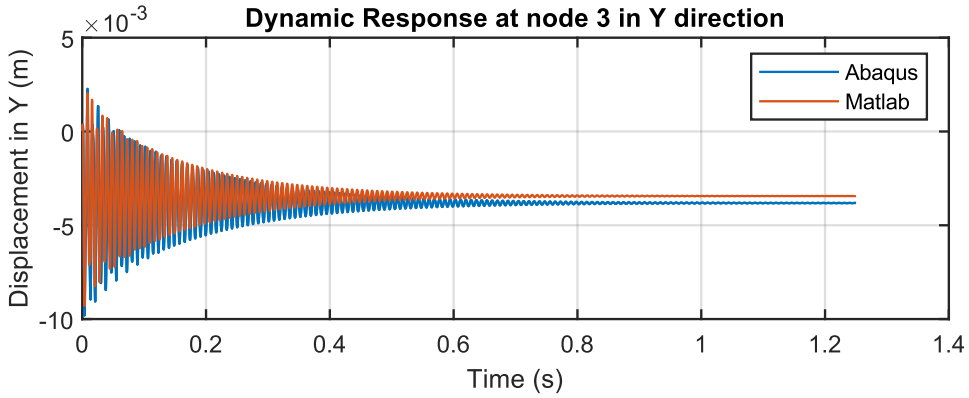
K

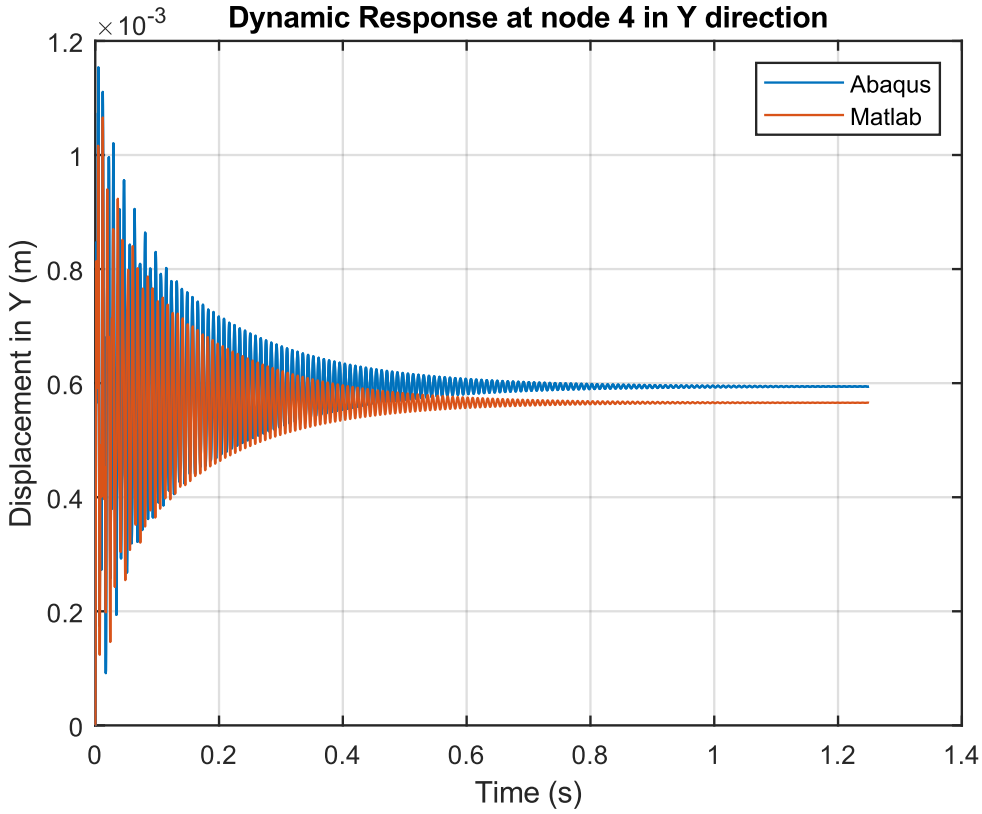
```
hold on
plot(t,u(4,:));
legend('Abaqus','Matlab')
xlabel('Time (s)')
ylabel('Displacement in Y (m)')
title('Dynamic Response at node 3 in Y direction')

figure(3)
plot(time_Abaqus,u2node4)
hold on
grid on
plot(t,u(5,:));
xlabel('Time (s)')
ylabel('Displacement in Y (m)')
title('Dynamic Response at node 4 in Y direction')
legend('Abaqus','Matlab')
```

L

7.3.2 Transient Response Plots







M