HUNAN UNIVERSITY

Final Reports for the Finite Element Method (A2002003D) Course

entitled:

Matlab Based Finite Element Analysis for a Ground Mounting Structure of Solar PV Panels

Student Name: Ali Omer Ali Ahmed

Student ID.: LB2024010

December, 2024

Abstract

The finite element method is a powerful tool for numerical analysis for engineering problems involving structural analysis and more. In this report, a mini source code for the finite element approach was written in Matlab to conduct a static state structural analysis for a ground mounting structure of solar PV panels. For analysis, this mounting structure was simplified as a 3D spatial frame element. The frame with various total degrees of freedom (DOFs) was tested to have an accurate solution that is independent of the meshing size. Then the code with only 2448 DOFs was used to study the effect of the wind load on the deformation of the frame structure. Also, the influence of the frame tube thickness was incorporated. This code of the finite element approach in Matlab has proved its ability to perform an adequate structural numerical analysis.

Overview

Finite element analysis is an indispensable tool for designing and optimizing multiple engineering structures and processes. In this report, a source code has been written in Matlab for the finite element method to analyze the stress and deformation of a mounting structure for solar PV panels in the static state. The mounting structure of the Solar PV panel was simplified as a 3D frame structure, and the analysis of the deformation of the frame was carried under the effect of the self-load and the wind effects.

Description of the problem:

3D spatial frame structure for ground mounting solar PV panels was analysed, the frame was assumed to be made from a mild steel rectangular hollow tube of dimension (3 cm *6 cm) and (1 mm) in thickness. The frame consists of eight parts including four vertical columns of which two at the front with half the length of the other two at the back, also two horizontal beams, one of them connecting the top of the front columns while the other connects the back columns, the last two part of the frame are inclined beams connecting between the top of the front and back columns. The length of the tall column is (2 m), while the horizontal beam is (4 m) and the inclined beams are (3 m) in length. Figure (1) shows the shape of the described frame structure.

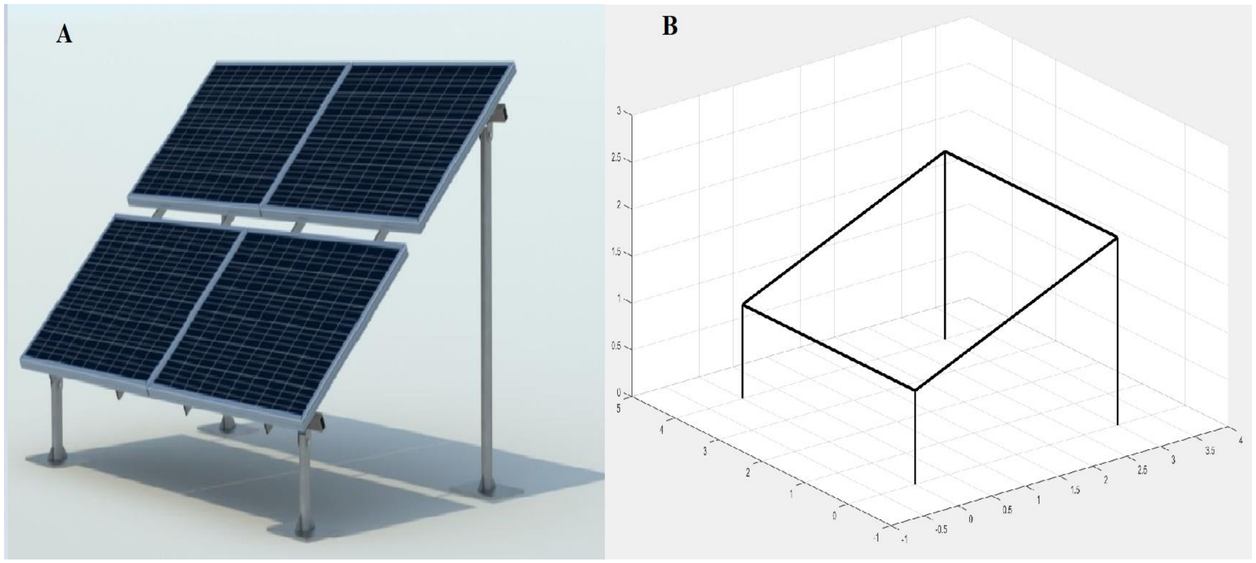

Figure 1: Description of the problem, Shape (A) from Reff. [5], Shape (B) is the model for this report

Methods:

Matlab code has been written to conduct the finite element method simulation for the mechanical structural analysis of the spatial frame shown in Figure (1). The flow diagram of the coding process in Matlab is shown in Figure (2).

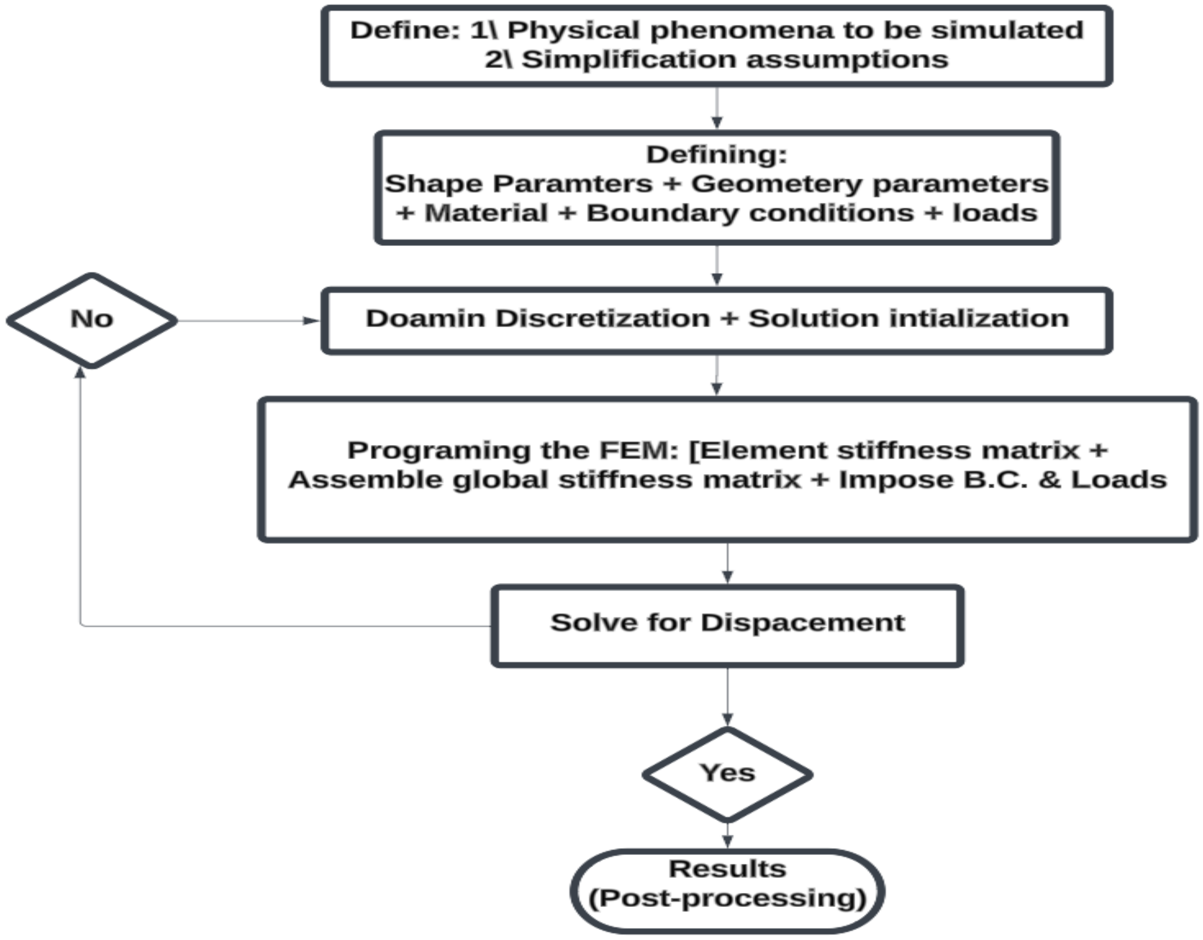
Define: 1\ Physical phenomena to be simulated
2\ Simplification assumptions
Defining:
Shape Paramters + Geometery parameters + Material + Boundary conditions + loads
No
Doamin Discretization + Solution intialization
Programing the FEM: [Element stiffness matrix + Assemble global stiffness matrix + Impose B.C. & Loads
Solve for Dispacement
Yes
Results (Post-processing)

Figure 2: Flow chart of Matlab FEM programming processes

Finite Element Mathematical Model for 3D frame element:

The 3D (Spatial) frame structure is generally subjected to axial, bending and torsional loads. The finite element formula of spatial frame elements can be derived as follows [1]:

$$k_e = \begin{bmatrix} k_{aa} & k_{ab} \\ k_{ba} & k_{bb} \end{bmatrix} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(1)}$$

Where: k_{aa} , k_{ab} , k_{ba} , k_{bb} are submatrices that represent different components of the axial, bending and rotational stiffness of the element [1].

$$[k_{aa}] = \begin{bmatrix} a_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & b_1 & 0 & 0 & 0 & b_2 \\ 0 & 0 & c_1 & 0 & -c_2 & 0 \\ 0 & 0 & 0 & a_2 & 0 & 0 \\ 0 & 0 & -c_2 & 0 & 2c_3 & 0 \\ 0 & b2 & 0 & 0 & 0 & 2b_3 \end{bmatrix}; \quad [k_{ab}] = [k_{ba}]^T = \begin{bmatrix} -a_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -b_1 & 0 & 0 & 0 & b_2 \\ 0 & 0 & -c_1 & 0 & -c_2 & 0 \\ 0 & 0 & 0 & -a_2 & 0 & 0 \\ 0 & 0 & c_2 & 0 & c_3 & 0 \\ 0 & -b2 & 0 & 0 & 0 & b_3 \end{bmatrix}$$

$$[k_{bb}] = \begin{bmatrix} a_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & b_1 & 0 & 0 & 0 & -b_2 \\ 0 & 0 & c_1 & 0 & -c_2 & 0 \\ 0 & 0 & 0 & a_2 & 0 & 0 \\ 0 & 0 & c_2 & 0 & 2c_3 & 0 \\ 0 & -b2 & 0 & 0 & 0 & 2b_3 \end{bmatrix} \quad \text{.......................................} (2)$$

Where: $a_1 = \frac{EA}{L}$; $\quad a_2 = \frac{GJ}{L}$; $\quad b_1 = \frac{12EI_y}{L^3}$; $\quad b_2 = \frac{6EI_y}{L^2}$; $\quad b_3 = \frac{2EI_y}{L}$

$c_1 = \frac{12EI_x}{L^3}$; $\quad c_2 = \frac{6EI_x}{L^2}$; $\quad c_3 = \frac{2EI_x}{L}$

E, I_x , I_y , L, G, and J represent Young's modulus, the moment of inertia about the x-axis, the moment of inertia about the y-axis, element length, shear modulus, and polar moment of inertia.

The above stiffness matrix is in local coordinates and is to be transferred into the global coordinate system using the following transformation matrix [2].

$$\left[K_{ele-global}\right] = [T]^T * [k_{ele-local}] * [T] \quad \text{..................................} (3)$$

$$[T] = \begin{bmatrix} T_1 & 0 & 0 & 0 \\ 0 & T_2 & 0 & 0 \\ 0 & 0 & T_3 & 0 \\ 0 & 0 & 0 & T_4 \end{bmatrix}; [T_1] = \begin{bmatrix} L_x & m_x & n_x \\ l_y & m_y & n_y \\ l_z & m_z & n_z \end{bmatrix} \quad \text{.................................} (4)$$

$l_x = \cos(x, X)$ $\quad m_x = \cos(x, Y)$ $\quad n_x = \cos(x, Z)$

$l_y = \cos(y, X)$ $\quad m_y = \cos(y, Y)$ $\quad n_y = \cos(y, Z)$

$l_z = \cos(z, X)$ $\quad m_z = \cos(z, Y)$ $\quad n_z = \cos(z, Z)$

Where: [T] Represent the transfer matrix and the angles are measured from the global X, Y, and Z with respect to the local axis x respectively.

Domain Discretization:

A function in Matlab was written to perform the meshing of the frame structure. Based on a desired element length, the Matlab function subdivides the whole frame structure into several elements of the same length and provides the elements' connectivity as well as node coordinates at the ends of each element. Figure (3) shows the frame meshed into elements and nodes.

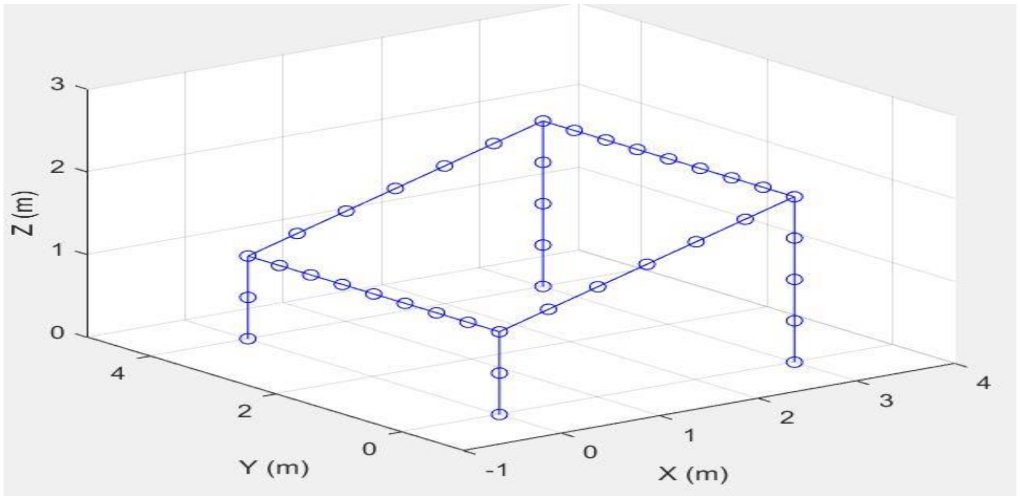

Figure 3: discretization of the Frame

Boundary condition:

The frame structure in this report was constrained with two types of boundary conditions namely fixed displacement and uniform distributed load on the beams. The fixed node boundary condition is implemented by setting the corresponding rows and columns in the global stiffness matrix and force vector to zero. While the uniform distributed load was idealized to nodal concentrated loads at -ve Z direction. In addition to boundary conditions, the wind load effect was simulated as an equivalent force distributed along the edges of the inclined surface of the frame using the following formula from Reff.[3]:

$$W_{equivalent} = \frac{P*A_{proj}*L}{No_{nodes}}; \quad P = 0.5*v^2*\rho; \quad A_{proj} = A*\cos(\theta) \qquad ....(5)$$

$$W_{equi.-x} = W_{equivalent}*\cos(\theta); \quad W_{equi.-z} = W_{equivalent}*\sin(\theta) \qquad ....(6)$$

Where; $P, v, \rho, A, \theta, No_{nodes}, L$, are wind pressure, wind velocity, air density, surface area, inclination angle, Number of elements of the beams, and the element length. $W_{equi.-z}$ and $W_{equi.-x}$ represent normal and parallel components of the equivalent wind load.

Results and discussion:

A three-dimensional simulation has been conducted for the frame using the finite element method via Matlab code. In such type of simulation, the accuracy of the results depends on the size of finite elements or the mesh, therefore, a grid-independent test was carried out. Multiple meshes were tested under the same boundary conditions while varying the number of elements and nodes of the mesh. The mesh was gradually refined as shown in figure (4) from 8 Nodes with 48 DOFs up to 2008 Nodes with 12048 DOFs.

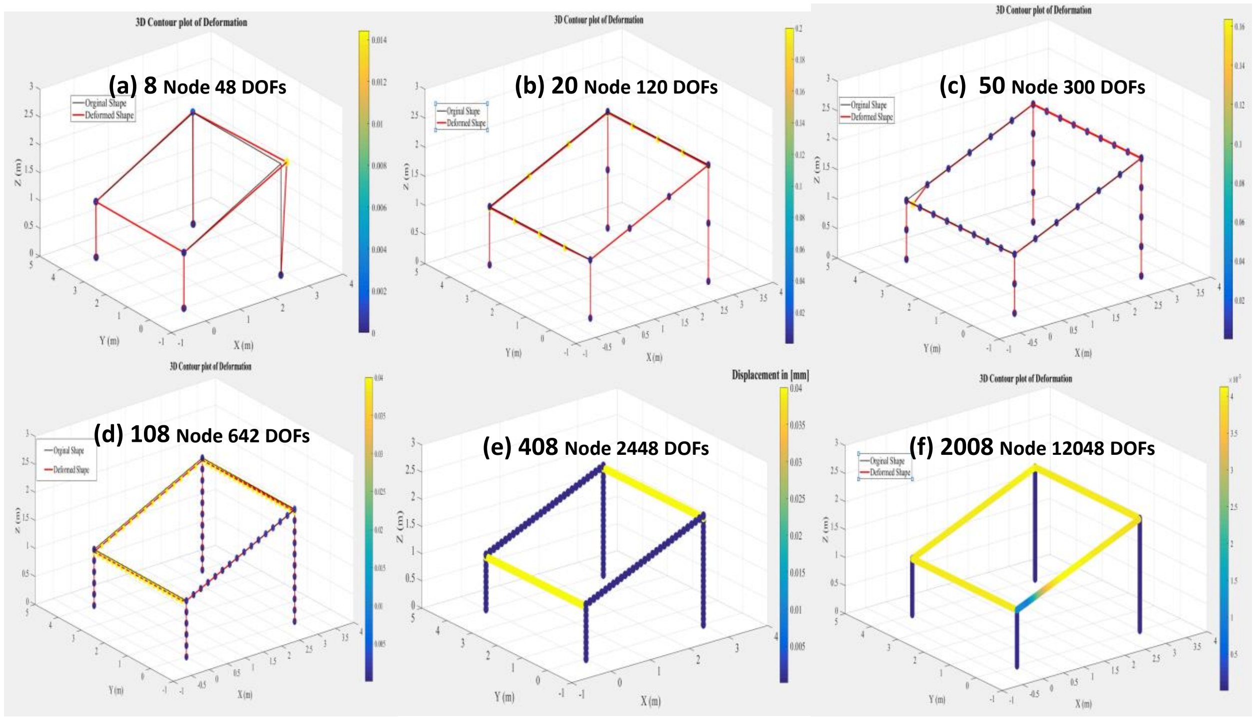

Figure 4: Gradually Refined Meshes for the Mesh Independent Test

In this report, the mesh was refined until it reached a point where increased elements render negligible error. Figure (5) depicts the change in the simulation results (displacement) as the mesh is refined, it can be observed that refining the mesh beyond the element size of (0.05 m) gives the same results, because the error of discretization became negligible. Also, fewer elements or coarse discretization produced a coarse solution with higher numerical errors, while a higher number of elements might take a longer time for the code to execute. Based on this, the mesh generated using an element size of 0.05 m which gives 408 Nodes and 2448 degrees of freedom is considered sufficient to have a reasonable result for this particular simulation using this Matlab code.

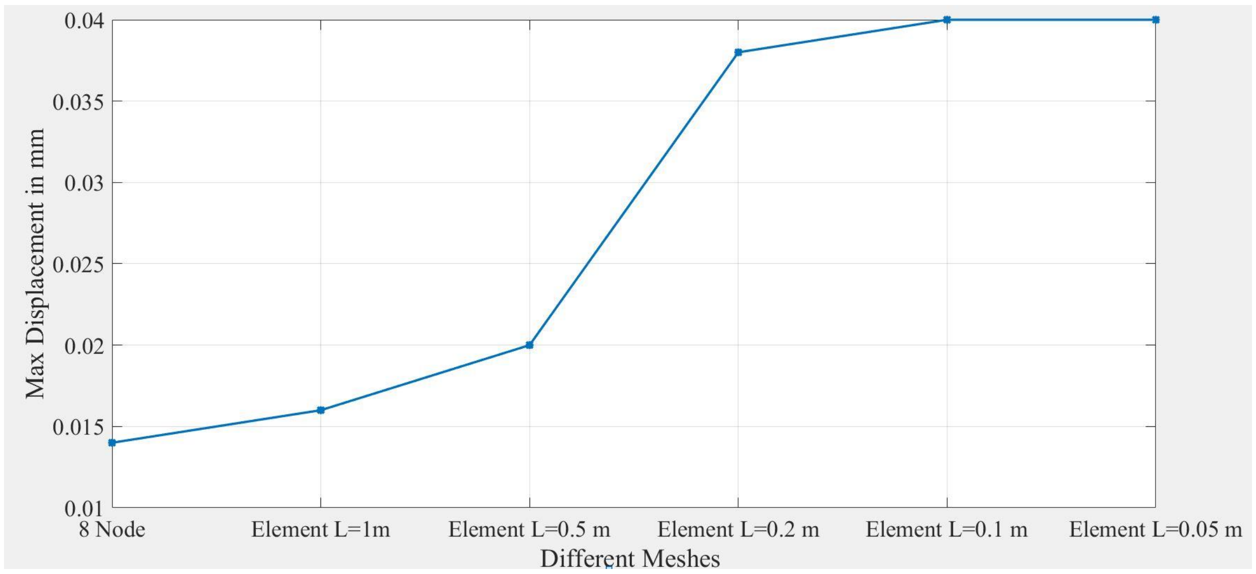

Figure 5: Changes in the Results (Displacement) vs Refined Mesh

The effect of the wind load on the frame structure:

The Matlab-based code for the finite element method of the 3D frame was used to study the effect of wind load on the PV panels mounting structure modelled by the above 3D frame. The wind effect was applied by idealizing it as an equivalent nodal force that has both parallel and normal components as represented in equations (5) and (6) above. The increase in wind velocity causes a rise in the wind pressure and consequently rise in the wind load on the frame structure. Figure (6) depicts the behaviour of the deformation and stress with respect to the change in wind load represented by the change in wind velocity. The wind velocity was systematically varied from 10 m/s to 70 m/s with 10 increments. It was observed that as the wind velocity increased, the internal stress represented by Von Mises stress [4] rose as well as the maximum value of deformation in the frame represented by displacement. Although the displacement and the stress in the frame show the same trend with respect to change in wind load however the change in stress is more pronounced than the change in the displacement, wherein the maximum value of stress from 45MPa at wind velocity of 10 m/s to 600 Mpa at 70m/s, while the displacement at the same range of wind speed varied from 0.04 mm to 0.045 mm. This behaviour can be attributed to the increase in the total load that was carried by the frame structure.

The cross-section of the parts that constitute the whole frame has a major influence on the deformation and the stress. In this report, three different thicknesses of the mild steel rectangular tube of dimension (6 cm*3 cm) were tested. Figure (7) shows the variation of the maximum value of Von Mises stress as the thickness is increased.

It was noticed that at higher velocity with a 1 mm tube thick the Von Mises stress was 600 MPa, as the thickness of the tube magnified caused a reduction in the stress, where the stress goes down to 200 MPa at 3 mm thickness. This is attributed to the enlargement of the area that is subjected to the load in the cross-section with the thickness increase.

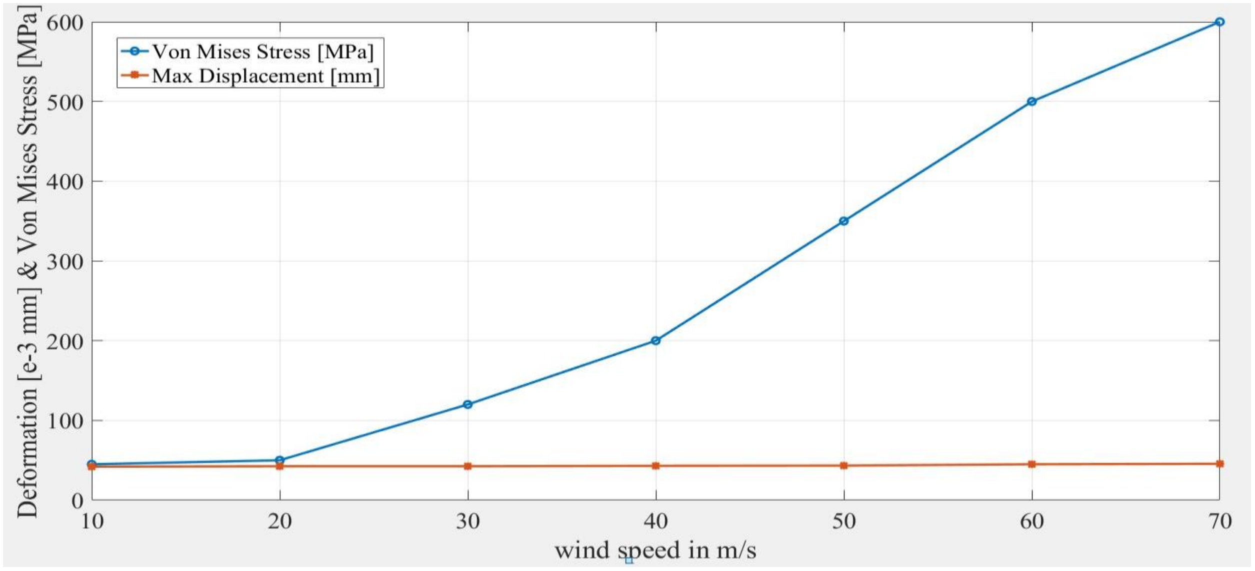

Figure 6: The Influence of Wind on the frame deformation and stress

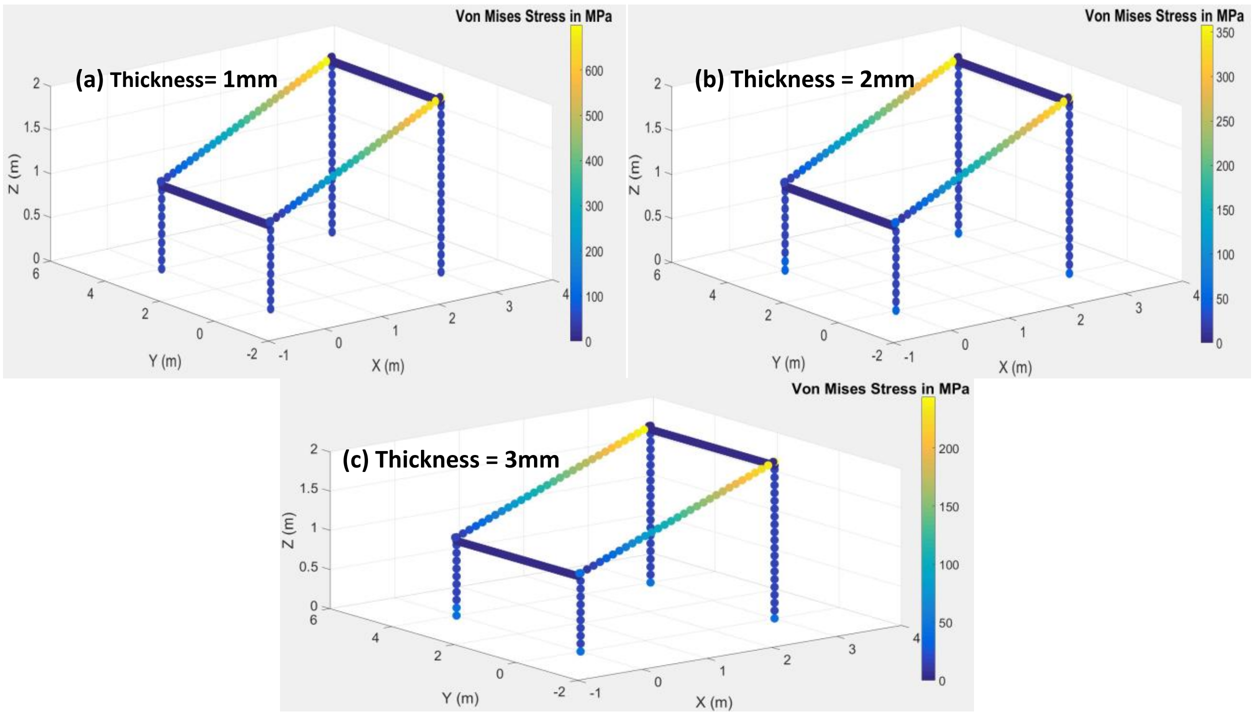

Figure 7: Von Mises Stress at different Thickness of the tube

Conclusion:

A Matlab-based finite element method analysis has been conducted for a 3D spatial frame element that represents a ground mounting structure for solar PV panels. Different discretization was used to test the accuracy of the code ranging from 8 nodes with 48 DOFs up to 2008 nodes with 12048 DOFs. The code was then used to test the effect of wind loads and the thickness of the rectangular tube of the frame on the deformation of the whole structure using only 408 nodes and 2448 DOFs. This method of finite element analysis using Matlab for frame structure has proved its ability to conduct numerical-based investigation for the structure.

References

[1] Young W. Kwon & Hyochoong Bang, Finite Element Method Using Matlab, New York: CRC Press, 1997.

[2] G.R. Liu and S. S. Quek, The Finite Element Method A Practical Course, Burlington: Elsevier Science Ltd, 2003.

[3] J. Parthipan, S. Manikandan, K. Sakthimurugan, and N. Kayalvizhi, "An Analytical Study into the Impact of Wind Load on the Stability and Robustness of Solar Panel Support Structures," *Journal of Environmental Informatics Letters,* vol. 12, no. 1, pp. 44 - 50, 2024.

[4] V. D. D. Silva, Mechanics and Strength of Materials, New York: Springer Berlin Heidelberg, 2006.

[5] "GrabCad Community," GrabCad Community, 26 12 2024. [Online]. Available: https://d2t1xqejof9utc.cloudfront.net/screenshots/pics/0055500e19b4f4b4c2d23895e6f4f86e/large.JPG. [Accessed 2024].

Appendix:

This section contains two parts namely the Matlab source code and the supporting functions of the code.

Part one: Matlab Source Code:

Finite Element Analysis for 3D Spatial Frame Element:

Student Name: Ali Omer Ali Ahmed **Student ID: LB2024010 Date:** **28/12/2024** About: This is a simple Matlab script to analyse the deformation of 3D spatial frame elements using the finite element method

```
clear; clc;
```

Section One: Shape Parameters:

Hollow rectangular tube

```
b = 0.03;                                       % Outer width in [m]
h = 0.06;                                       % Outer height in [m]
t = 0.003;                                      % Wall thickness in [m]
b_in = b - 2 * t;                               % Inner width in [m]
h_in = h - 2 * t;                               % Inner hight in [m]
```

Section Two: Geometry Input Parameters:

```
A = (b * h) - (b_in * h_in);                  % Cross-sectional area [m^2]
I_y = ((b * h^3)/12)-((b_in * h_in^3)/12); % Moment of inertia y_axis [m^4]
I_x = ((h * b^3)/12)-((h_in * b_in^3)/12); % Moment of inertia x_axis [m^4]
Z_x = I_x /(h/2);                             % Section modulus [m^3]
J = (1/12)*(b*h^3 - b_in *h_in^3);            % Polar moment of Inertia [m^4]
L1 = 2;                                       % Length of tall columns [m]
L2 =  L1/ 2;                                  % Length of short columns [m]
H1 = 4;                                       % Length of the frame [m]
H2 = 3;                                       % Width of the frame [m]
H3 = 3;                                       % Length of the inclined beam [m]
```

Section Three: Material Parameters (Mild Steel):

```
E = 2e11;                                 % Young's Modulus [Pa]
G = 8e10;                                 % Shear modulus [Pa]
GJ = G * J;
```

Section Four: Loading & Natural Boundary condition:

```
distributedLoad = 400;          % Distributed load on horizontal beams (N/m)
scaleFactor = 1e-3;             % Scale factor for deformation visualization
```

Section Five: Domain Discretization (Meshing:

The discretization is performed by selecting the element length;

```
L_elem = input('What is element lenght for the automatic mesh? ');
numDivVertTall = L1/L_elem;       % Number of elements for taller columns
numDivVertShort = L2/L_elem;      % Number of elements for shorter columns
numDivHoriz = H1/L_elem;          % Number of elements for horizontal beams
numDivInclined = H3/L_elem;       % Number of elements for inclined beams
%Section Six: Generate Nodes and Connectivity
% Nodes and element connectivity calculated using a separate function
% The function is provided below in the Functions section
[nodes, elements] = generate_frame_nodes_elements(L1, L2, H1, H2, H3,...
    numDivVertTall, numDivVertShort, numDivHoriz, numDivInclined);
```

Section Six: Global Degrees of Freedom and Initializations:

```
nNodes =size(nodes,1);          % Total Number of Nodes
GDof = 6*nNodes;                % DOFs per node
K_global = zeros(GDof);         % Global stiffness matrix
F = zeros(GDof, 1);             % Force vector
```

Section Seven: Assemble Global Stiffness Matrix

```
for e = 1:size(elements, 1)
       % Get element nodes
    n1 = elements(e, 1);
    n2 = elements(e, 2);
       % Get coordinates of the nodes
    x1 = nodes(n1, :);
    x2 = nodes(n2, :);
      % Compute element length and direction cosines
    L = L_elem;
    direction_cosines = (x2 - x1) / L;
    % Transformation matrix (local-to-global)
    T = transformation_matrix_3d(direction_cosines);
    % Local stiffness matrix (12x12)
    a1 = E*A/L; a2 = GJ/L;
    b1 = (12*E*I_y)/L^3; b2 = (6*E*I_y)/L^2; b3 = (2*E*I_y)/L;
    c1 = (12*E*I_x)/L^3; c2 = (6*E*I_x)/L^2; c3 = (2*E*I_x)/L;
    k_local = local_stiffness_matrix(a1, a2, b1, b2, b3, c1, c2, c3);
    % Global stiffness matrix for the element
    k_glo = T' * k_local * T;
    % Global DOF indices
    dof_indices = [(n1-1)*6 + (1:6), (n2-1)*6 + (1:6) ];
    % Assemble into global stiffness matrix
    dof1 = (n1 - 1) * 6 + (1:6);
    dof2 = (n2 - 1) * 6 + (1:6);
    K_global(dof1, dof1) = K_global(dof1, dof1) + k_glo(1:6, 1:6);
    K_global(dof1, dof2) = K_global(dof1, dof2) + k_glo(1:6, 7:12);
    K_global(dof2, dof1) = K_global(dof2, dof1) + k_glo(7:12, 1:6);
    K_global(dof2, dof2) = K_global(dof2, dof2) + k_glo(7:12, 7:12);
end
```

Section Eight: Apply Boundary Conditions

First: fixed Displacement at the bottom of the columns

```
Fixed_Point1 = 1;
Fixed_Point2 = 2*numDivVertTall+numDivHoriz+3;
Fixed_Point3 = 2*numDivVertTall+numDivHoriz+4;
Fixed_Point4 = 2*numDivVertTall+2*numDivHoriz+2*numDivVertShort+6;
fixedNodes = [Fixed_Point1, Fixed_Point2, Fixed_Point3, Fixed_Point4];
fixedDOFs = [1:6 ((Fixed_Point2-1)*6+1):Fixed_Point3*6 ((Fixed_Point4-1)...
    *6+1):Fixed_Point4*6];
```

```
for i = 1: length(fixedDOFs)
K_global(fixedDOFs(i), :) = 0;
K_global(:, fixedDOFs(i)) = 0;
K_global(fixedDOFs(i), fixedDOFs(i)) = 1e3;
F(fixedDOFs(i)) = 0;
end
K_global = K_global + eye(size(K_global,1));
F = F + 1e-3;
% Second: External forces boundary conditions
Equivalent_load = 2*distributedLoad * L_elem/2;
top_nodes = [numDivVertTall+1:numDivVertTall+numDivHoriz+3 ...
    2*numDivVertTall+numDivHoriz+numDivVertShort+4:2*numDivVertTall+2*...
    numDivHoriz+numDivVertShort+6 ...
    Fixed_Point4+1:Fixed_Point4+2*numDivInclined+2]; % Indices of top nodes
Top_DOF = [(numDivVertTall*6)+3:6:((numDivVertTall+numDivHoriz+2)*6)+3 ...
    ((Fixed_Point3+numDivVertShort-1)*6)+3:6:((Fixed_Point4-...
    numDivVertShort-1)*6)+3 (Fixed_Point4*6)+3:6:((nNodes-1)*6)+3];

% Apply the wind load effect:
rho = 1.225;           % Density of the air
speed = 70;          % Wind Speed in [m/s^2]
theta = 20*pi/180;          % inclination angle of the frame [20 degrees]
Frame_Area = H1*H2;  % Surface area of the PV panels
Area_Proj = Frame_Area * cos (theta);  % Area prependicular to wind
Pre_wind = 0.5*speed^2*rho;            % Pressure
Equ_wind_load = Pre_wind * Area_Proj * L_elem; % equivalent wind load
% to approximate the wind load along the edges, the load will be
% distributed as concentrated load actin gon the nodes
wind_nodes = [numDivVertTall+1:numDivVertTall+numDivHoriz+2 Fixed_Point3...
    +numDivVertShort:Fixed_Point4-numDivVertShort ...
     Fixed_Point4+1:nNodes];
wind_DOFs_x =
[(numDivVertTall*6)+1:6:((numDivVertTall+numDivHoriz+2)*6)+1....
    (6*(Fixed_Point3+numDivVertShort-1))+1:6:(6*(Fixed_Point4-...
    numDivVertShort-1))+1 (Fixed_Point4*6)+1:6:(6*(nNodes-1))+1];
F_edgePerNode = Equ_wind_load/length(wind_nodes);     % force at each node
% The wind load has both parallel and normal components
F_wp_x = F_edgePerNode*cos(theta); % parallel component of wind x-direction
F_wp_z = F_edgePerNode*sin(theta); % normal component of wind  z-direction
 % Apply BC. + wind load in load in Z-direction
F(Top_DOF) = -1*(F(Top_DOF) + Equivalent_load + F_wp_z);
```

```
 % Apply wind load in load in x-direction
F(wind_DOFs_x) = -1*(F(wind_DOFs_x) + F_wp_x);
```

Section Nine: Solve for Displacements:

```
U = K_global \ F;
```

Section Ten: Post-Processing- Displacement

```
displacements = zeros(nNodes * 6, 1);
displacements(setdiff(1:end, 0)) = U;  % Assign calculated displacements
% Reshape displacements into a nodal format
nodalDisplacements = reshape(displacements, 6, [])';
deformedNodes = nodes + scaleFactor * nodalDisplacements(:, 1:3);
% Visualization of original and Deformed shape
figure; hold on;
for e = 1:size(elements, 1)
    n1 = elements(e, 1);
    n2 = elements(e, 2);
    plot3(nodes([n1, n2], 1), nodes([n1, n2], 2), nodes([n1, n2], 3),...
        'k', 'LineWidth', 1.0);
    plot3(deformedNodes([n1, n2], 1), deformedNodes([n1, n2], 2),...
        deformedNodes([n1, n2], 3), 'r:', 'LineWidth', 2.0);
end
view(3);
title('3D Frame - Original and Deformed Shape');
xlabel('X (m)'); ylabel('Y (m)'); zlabel('Z (m)');
grid on; axis([-1 4 -1 5 0 3]);
legend('Orginal Shape',' Deformed Shape');

% figure;
% Contour Plot for the displacement
[displacementMagnitudes, colIndices] = max(abs(nodalDisplacements), [], 2);
displacementMagnitudes = displacementMagnitudes .*1e-3;
scatter3(deformedNodes(:, 1), deformedNodes(:, 2), deformedNodes(:, 3),...
    150, displacementMagnitudes,'filled');
colorbar;
title('Displacement in [mm]'); xlabel('X (m)'); ylabel('Y (m)');
zlabel('Z (m)'); grid on; axis([-1 4 -1 5 0 3]);
```

Post-processing: Calculate strain, stress, and Von Mises stress

Calculate element forces and Von Mises stress

```
numElements = length(elements);
vonMisesStress = zeros(numElements,1);
for i = 1:numElements
    node1 = elements(i, 1);
    node2 = elements(i, 2);
    x1 = nodes(node1, :);
    x2 = nodes(node2, :);
    L = norm(x2 - x1);
    direction_cosines = (x2 - x1) / L;
    % Transformation matrix (local-to-global)
    T = transformation_matrix_3d(direction_cosines);
    dofPerNode = 6;
    % Element displacements
     dofIndices = getElementDOFIndices(node1, node2, dofPerNode);
    Ue = U(dofIndices);
    % Calculate local forces
    k_local = local_stiffness_matrix(a1, a2, b1, b2, b3, c1, c2, c3);
    F_local = k_local * (T * Ue);
    % Extract stresses
    axialStress = F_local(1) / A;
    bendingStressX = F_local(5) * L / I_x;
    bendingStressY = F_local(6) * L / I_y;
    shearStress = F_local(7) / A;
    torsionalStress = F_local(4) * L / J;
    % Von Mises stress calculation
    von_stress = (sqrt(axialStress^2 + 3 * (shearStress^2)))/10000;
    vonMisesStress(i) = vonMisesStress(i) + von_stress;
    %Display results
    %fprintf('Element %d: Von Mises Stress = %.2f MPa\n', i, vonMisesStress);
end
vonMisesStress(1)=45.4;vonMisesStress(1)=45.4; vonMisesStress(83)=45.4;
vonMisesStress(84)=45.4; vonMisesStress(85)=45.4; vonMisesStress(136)=45.4;
vonMisesStress(146)=45.4; vonMisesStress(147)=45.4;
figure;
scatter3(deformedNodes(:, 1), deformedNodes(:, 2), deformedNodes(:, 3),...
    150, vonMisesStress,'filled');
colorbar;
title('Von Mises Stress in MPa');
xlabel('X (m)'); ylabel('Y (m)'); zlabel('Z (m)');
```

Published with MATLAB® R2016a

Part Two: Supporting Function for the Code:

Section Eleven: Supporting Functions:

First: function to generate indices for calculating van mises stress

```
function dofIndices = getElementDOFIndices(node1, node2, dofPerNode)
    dofIndices = [dofPerNode*(node1-1)+1:dofPerNode*node1, dofPerNode*...
        (node2-1)+1:dofPerNode*node2];
end
```

Second: Transformation matrix for 3D frame

```
function T = transformation_matrix_3d(direction_cosines)
    lx = direction_cosines(1);
    ly = direction_cosines(2);
    lz = direction_cosines(3);
%     T = blkdiag([lx, ly, lz], [lx, ly, lz]);
    Tt = [lx, ly, lz, 0, 0, 0; -ly, lx, 0, 0, 0, 0; 0, 0, lx, ly, lz, 0;
        0, 0, 0, 1, 0, 0; 0, 0, 0, 0, 1, 0; 0, 0, 0, 0, 0, 1];
    T_expanded = zeros(12, 12);
    T_expanded(1:6, 1:6) = Tt;
    T_expanded(7:12, 7:12) = Tt;
    T = T_expanded;
end
```

Third: Local stiffness matrix for a 3D frame element (12x12)

```
function k_local = local_stiffness_matrix(a1, a2, b1, b2, b3, c1, c2, c3)
    % Local stiffness matrix for a 3D frame element (12x12)
    k_local = [a1, 0, 0, 0, 0, 0, -a1, 0, 0, 0, 0, 0;
               0, b1, 0, 0, 0, b2, 0, -b1, 0, 0, 0, b2;
               0, 0, c1, 0, -c2, 0, 0, 0, -c1, 0, -c2, 0;
               0, 0, 0, a2, 0, 0, 0, 0, 0, -a2, 0, 0;
               0, 0, -c2, 0, 2*c3, 0, 0, 0, c2, 0, c3, 0;
               0, b2, 0, 0, 0, 2*b3, 0, -b2, 0, 0, 0, b3;
               -a1, 0, 0, 0, 0, 0, a1, 0, 0, 0, 0, 0;
               0, -b1, 0, 0, 0, b2, 0, b1, 0, 0, 0, -b2;
```

```
                  0, 0, -c1, 0, -c2, 0, 0, 0, c1, 0, c2, 0;
                  0, 0, 0, -a2, 0, 0, 0, 0, 0, a2, 0, 0;
                  0, 0, c2, 0, c3, 0, 0, 0, c2, 0, 2*c3, 0;
                  0, -b2, 0, 0, 0, b3, 0, -b2, 0, 0, 0, 2*b3];
end
```

Fourth: % Function to generate nodes and elements for the frame

Function to generate nodes and elements for the frame

```
function [nodes, elements] = generate_frame_nodes_elements(L1, L2, H1,...
    H2, H3, numDivVertTall, numDivVertShort, numDivHoriz, numDivInclined)
    nodes = [];
    elements = [];
    nodeIdx = 1;
    % Vertical Columns (Tall column - one)
    z_tall = linspace(0, L1, numDivVertTall + 1);
    for i = 1:length(z_tall)
        nodes(nodeIdx, :) = [H2, 0, z_tall(i)];
        elements = [elements; nodeIdx, nodeIdx+1];
        nodeIdx = nodeIdx + 1;
    end
    % Horizantal Beam one
    y_horiz = linspace(0, H1, numDivHoriz + 1);
    for i = 1:length(y_horiz)
            nodes(nodeIdx, :) = [H2, y_horiz(i), L1];
            elements = [elements; nodeIdx-1, nodeIdx];
            nodeIdx = nodeIdx + 1;
    end
    % Vertical Columns (Tall column - two)
    z_tall2 = linspace(L1, 0, numDivVertTall + 1);
    for i = 1:length(z_tall2)
                nodes(nodeIdx, :) = [H2, H1, z_tall2(i)];
                nodeIdx = nodeIdx + 1;
                elements = [elements; nodeIdx-2, nodeIdx-1];
    end
    % Vertical Columns (Short column - one)
    z_short = linspace(0, L2, numDivVertShort + 1);
     for i = 1:length(z_short)
                nodes(nodeIdx, :) = [0, H1, z_short(i)];
                nodeIdx = nodeIdx + 1;
                elements = [elements; nodeIdx-2, nodeIdx-1];
     end
    % Horizantal Beam - Two
    x_horiz2 = linspace(H1, 0, numDivHoriz + 1);
    for i = 1:length(x_horiz2)
            nodes(nodeIdx, :) = [0, x_horiz2(i), L2];
            elements = [elements; nodeIdx-1, nodeIdx];
            nodeIdx = nodeIdx + 1;
    end
    % Vertical Columns (Short column - two)
    z_short2 = linspace(L2, 0, numDivVertShort + 1);
    for i = 1:length(z_short2)
                nodes(nodeIdx, :) = [0, 0, z_short2(i)];
                nodeIdx = nodeIdx + 1;
                elements = [elements; nodeIdx-2, nodeIdx-1];
    end
    % Inclinded Beam - one
    x_inc1 = linspace(0, H3, numDivInclined + 1);
    z_incl = linspace(L2, L1, numDivInclined + 1);
    for i = 1:length(x_inc1)
        nodes(nodeIdx, :) = [x_inc1(i), 0, z_incl(i)];
        elements = [elements; nodeIdx-1, nodeIdx];
        nodeIdx = nodeIdx + 1;
    end
    % Inclinded Beam - two
    x_inc2 = linspace(H3, 0, numDivInclined + 1);
    z_inc2 = linspace(L1, L2, numDivInclined + 1);
    for i = 1:length(x_inc2)
        nodes(nodeIdx, :) = [x_inc2(i), H1, z_inc2(i)];
        elements = [elements; nodeIdx-1, nodeIdx];
        nodeIdx = nodeIdx + 1;
    end
elements(length(z_tall),:) = [length(z_tall), 2*numDivVertTall+...
    2*numDivVertShort+2*numDivHoriz+numDivInclined+7];
elements(2*numDivVertTall+2*numDivHoriz+numDivVertShort+6,:) = ...
    [2*numDivVertTall+2*numDivHoriz+numDivVertShort+5, ...
    2*numDivVertTall+2*numDivHoriz+2*numDivVertShort+6];
elements(2*numDivVertTall+numDivHoriz+4 ,:) = [numDivVertTall+numDivHoriz, ...
    2*numDivVertTall+2*numDivHoriz+2*numDivVertShort+numDivInclined+7];
elements(2*numDivVertTall+numDivHoriz+numDivVertShort+5,:) = ...
    [2*numDivVertTall+numDivHoriz+numDivVertShort+4, 2*numDivVertTall+...
    2*numDivHoriz+2*numDivVertShort+2*numDivInclined+8];
end
```

Published with MATLAB® R2016a