ME502L: Finite Element Methods for Engineering Mechanics
Determination of Young's Moduli of Nanocomposite: Report

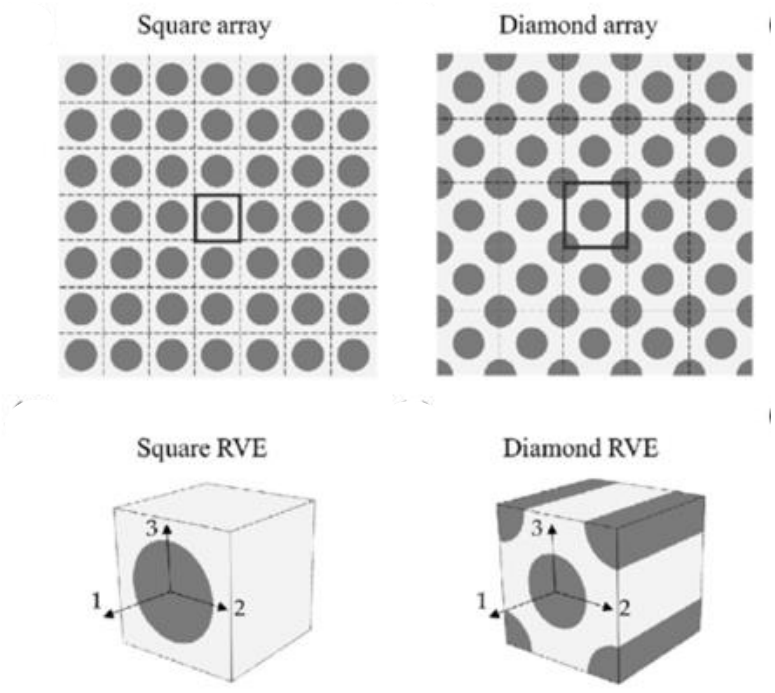
07/05/2023 Sneha M S – ME20B052

Table of Contents:

- Problem Statement
- Formulation
- Isoparametric Mapping
- Methodology
- Results

Problem Statement:

A nanocomposite is made of polystyrene matrix and graphene nanoparticles. Typically, graphene particles have a size of 150 nm and a concentration of 5% volume fraction. For the purpose of the present 2-D analysis, assume they are circular in shape with uniformly spaced rectangular array. Determine Young's moduli E_x, E_y in the x and y-directions.



The code development for square and diamond array Representative Volume Elements (RVE) uses MATLAB.

Formulation: Finite Element Method

The finite element method (FEM) is a powerful numerical technique that can solve the mechanical problem of the nanocomposite consisting of a polystyrene matrix and graphene nanoparticles. In this approach, the nanocomposite is divided into small finite elements, and each Element is characterized by its own set of governing equations based on the material properties and boundary conditions. By discretizing the nanocomposite into these finite elements, the FEM approximates the overall behaviour of the structure. The equations governing the mechanical behaviour of each Element are then solved iteratively to obtain the displacement and stress fields throughout the nanocomposite. By utilizing appropriate constitutive models for the polystyrene matrix and graphene nanoparticles, the FEM can provide valuable insights into the distribution of stresses, strain concentrations, and deformation patterns within the nanocomposite under different loading conditions. This information can be used to assess the mechanical performance, optimize the material composition, and guide the design of nanocomposites with enhanced properties.

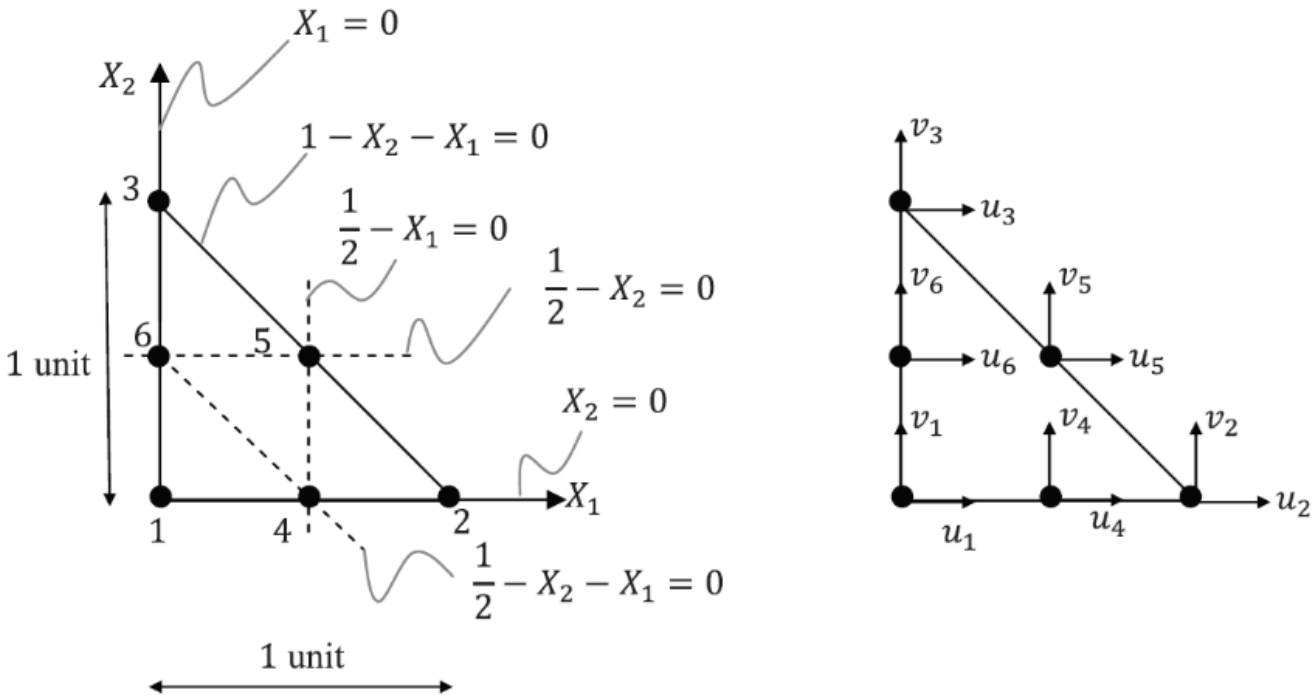
Finite Element Used: 6-noded Isoparametric Triangle

Isoparametric finite element is a widely used technique in finite element analysis that allows for a more accurate and efficient discretization of the domain. In isoparametric finite elements, both the shape of the element and the interpolation functions used to approximate the unknowns (e.g., displacements, stresses) are defined in terms of the same set of parameters. By employing isoparametric elements, the geometry of the actual domain can be accurately represented using a relatively small number of elements. This leads to significant computational efficiency, as fewer elements are required to discretize the domain compared to other techniques. Furthermore, isoparametric elements provide higher accuracy in capturing complex geometries and curved boundaries. The shape functions can be designed to closely match the actual shape of the domain, leading to more precise approximations of the unknowns and more accurate solutions.

The Quadratic Triangular Element

This is the element onto which each subdomain is mapped to iso parametrically. The displacement function on the element can be assumed to have the following form:

$$u_{2D} = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} a_0 + a_1 X_1 + a_2 X_2 + a_3 X_1 X_2 + a_4 X_1^2 + a_5 X_2^2 \\ b_0 + b_1 X_1 + b_2 X_2 + b_3 X_1 X_2 + b_4 X_1^2 + b_5 X_2^2 \end{pmatrix}$$



where a_0, a_1, a_2, a_3, a_4, a_5, b_0, b_1, b_2, b_3, b_4, and b_5 are twelve generalized degrees of freedom of the element. The approximate displacement function in terms of the nodal degrees of freedom has the form:

$$u_{2D} = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} N_1 u_1 + N_2 u_2 + N_3 u_3 + N_4 u_4 + N_5 u_5 + N_6 u_6 \\ N_1 v_1 + N_2 v_2 + N_3 v_3 + N_4 v_4 + N_5 v_5 + N_6 v_6 \end{pmatrix}$$

The shape functions are as follows:

$$N_1 = (1 - X_1 - X_2)(1 - 2X_1 - 2X_2)$$

$$N_2 = X_1(2X_1 - 1)$$

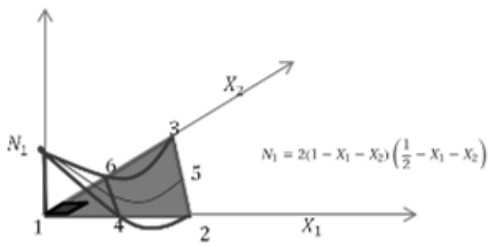
$$N_3 = X_2(2X_2 - 1)$$

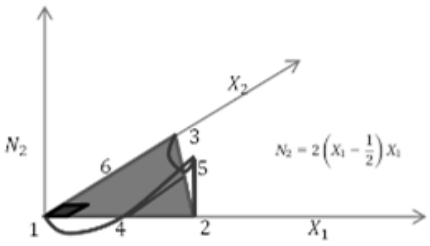
$$N_4 = 4X_1(1 - X_1 - X_2)$$

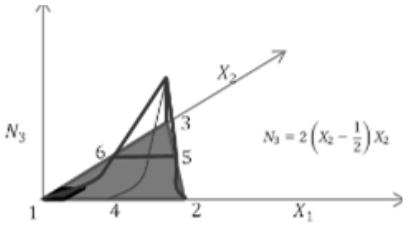
$$N_5 = 4X_1X_2$$

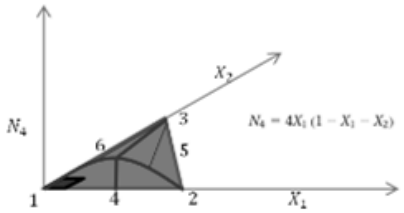
$$N_6 = 4X_2(1 - X_1 - X_2)$$

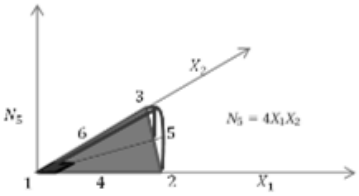
The Shape Function Distribution is as follows:

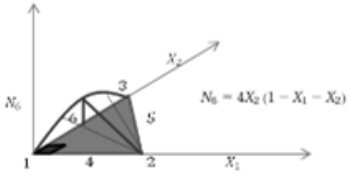












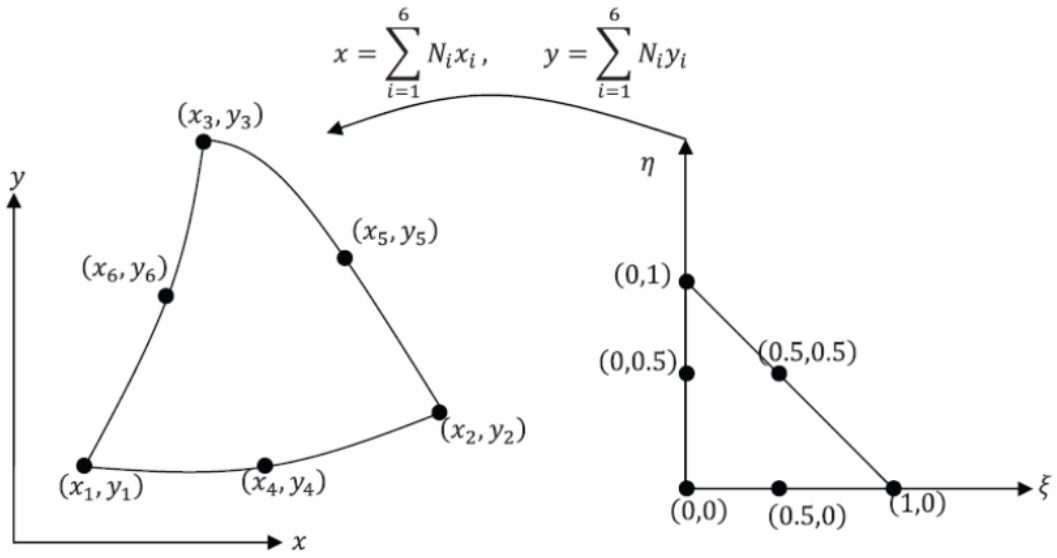
Isoparametric Mapping:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 & 0 & N_5 & 0 & N_6 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 & 0 & N_5 & 0 & N_6 \end{bmatrix} \begin{vmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \\ u_5 \\ v_5 \\ u_6 \\ v_6 \end{vmatrix} \quad \Rightarrow \begin{pmatrix} u \\ v \end{pmatrix} = \lfloor N \rfloor \{u^e\}$$

$$\begin{pmatrix} \varepsilon_x \\ \varepsilon y \\ r_{xy} \end{pmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \quad \Rightarrow \{\varepsilon\} = [\mathcal{L}][N]\{u^e\} \qquad J = \begin{bmatrix} \frac{\partial x}{\partial \mathrm{X}_1} & \frac{\partial y}{\partial \mathrm{X}_1} \\ \frac{\partial x}{\partial X_2} & \frac{\partial y}{\partial X_2} \end{bmatrix}$$

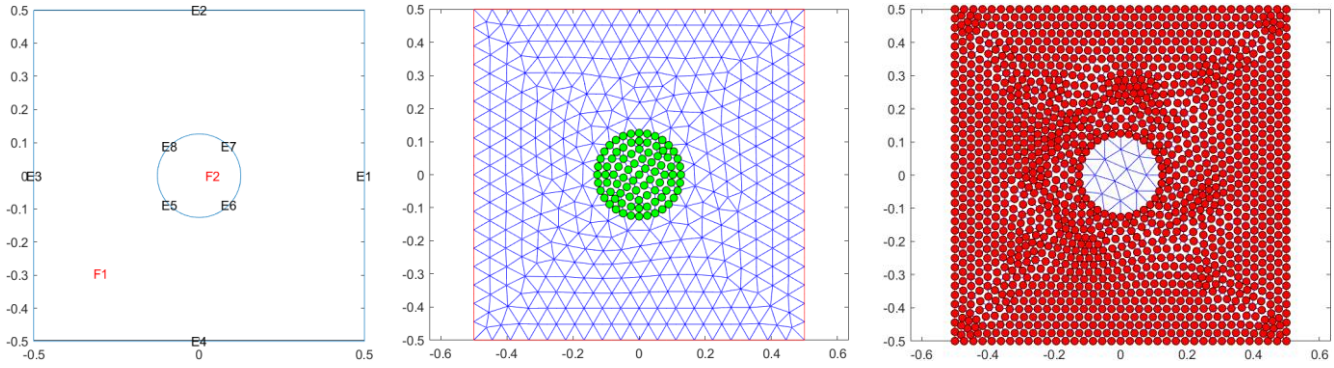
$$[J]^{-1} = M \qquad \{\varepsilon\} = \begin{bmatrix} M_{11} & M_{12} & 0 & 0 \\ 0 & 0 & M_{21} & M_{22} \\ M_{21} & M_{22} & M_{11} & M_{12} \end{bmatrix} \begin{pmatrix} \frac{\partial u}{\partial x_1} \\ \frac{\partial u}{\partial x_2} \\ \frac{\partial v}{\partial x_1} \\ \frac{\partial v}{\partial x_2} \end{pmatrix} \qquad \Rightarrow \{\varepsilon\} = [G][P]\{u^e\}$$

$$K = \int_{\Omega^{(e)}} [B]^T[D][B]\,dV = \int_{-1}^{11}\int [P]^T[G]^T[D][G][P]|J|tdx_1dx_2$$



Methodology

- Representative Volume Element

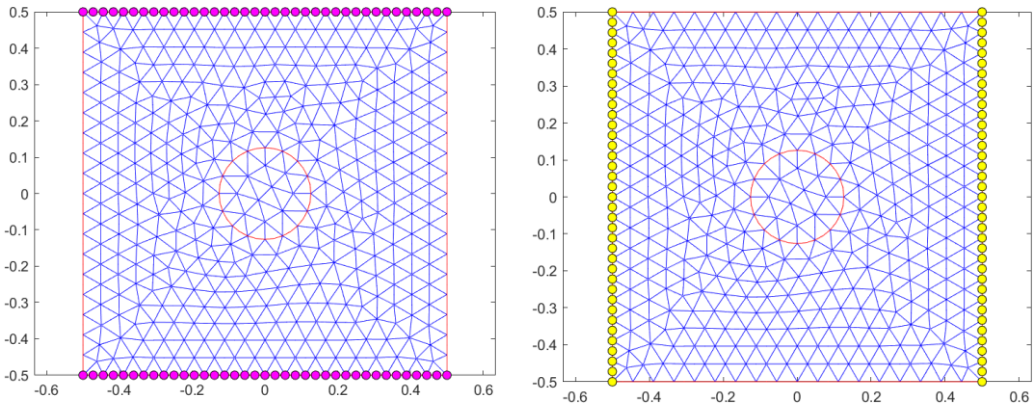


Rectangular array RVE; Nodes corresponding to Fibre and Matrix are highlighted

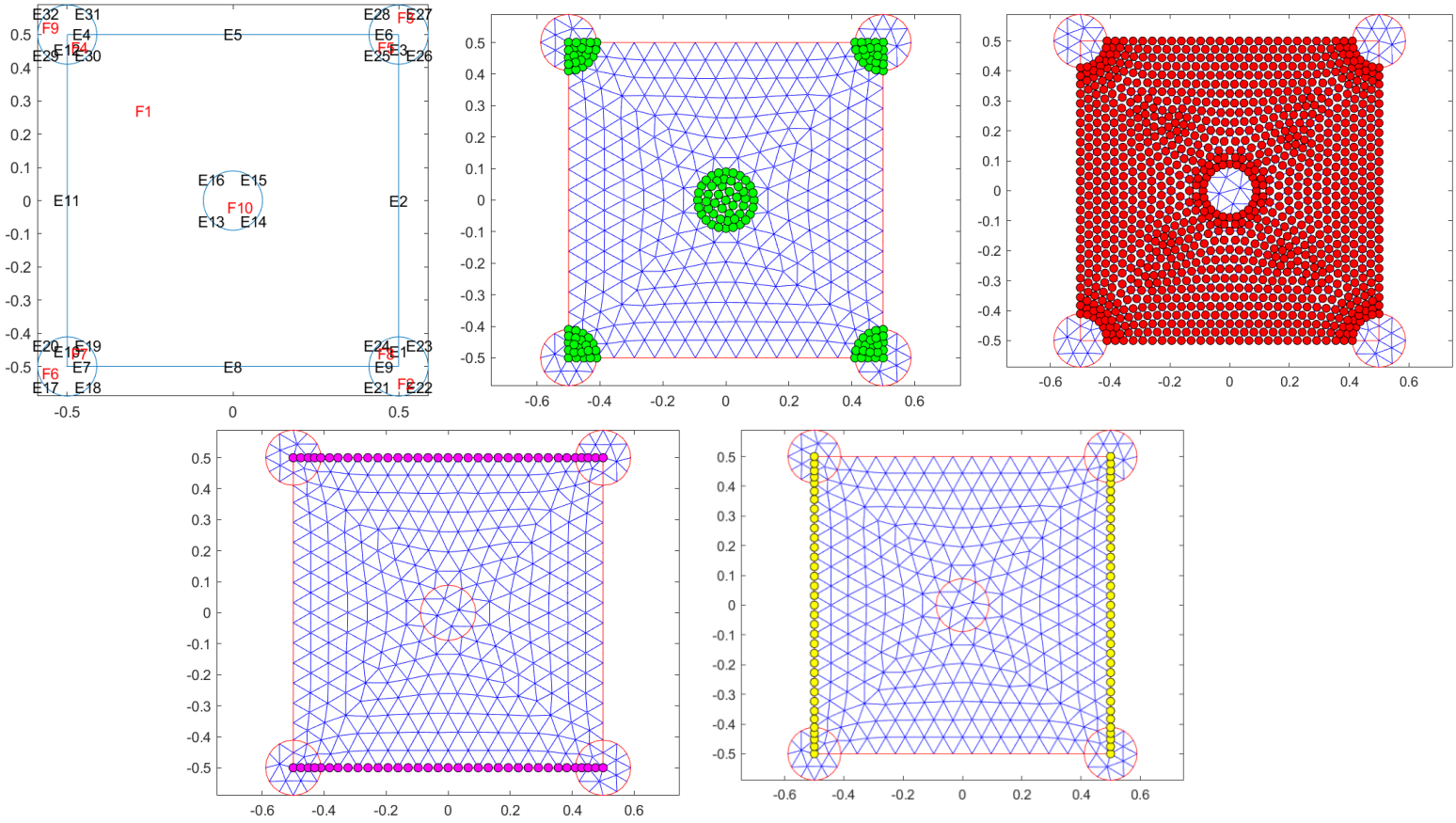
- Extract the nodes corresponding to the Boundaries.

  The adjacent sides of RVE must remain straight and parallel, and hence edges 2 and 4 must be given a displacement boundary condition such that the average stress $\bar{\sigma}$ along those edges is 0.

Edge 2	Y - displacement
Edge 4	Fixed
Edge 1	-dX displacement corresponding to zero average stress along the edge
Edge 2	dX displacement corresponding to zero average stress along the edge



Nodes corresponding to boundaries as described above.



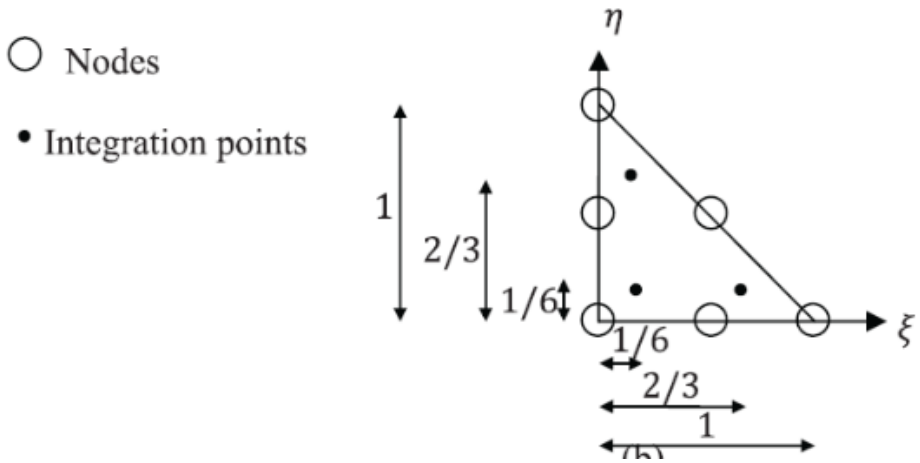
A Diamond array RVE with fibres and Matrix highlighted and boundary nodes

- Finding the displacement boundary condition dx that will result in average stress of zero such that average stress $\bar{\sigma}$ along E3 =0;
  For dx=0.01, the corresponding dy was found to make $\bar{\sigma}$ along E3 zero

dy for dx=0.01 for rectangular RVE	
dy	**Average stress along edge E3**
0.1	2.91e+08
0.2	1.44e+08
0.3	-0.02e+08
0.28	0.26e+08
0.29	0.11e+08
0.2975	0.0078e+08
0.2985	-0.0070e+08
0.298	0.0004e+08
0.2981	-0.0011e+08
0.29801	0.0002e+08
0.29802	0.0001e+08
0.298025	0

- Finding the Elemental Siffness Matrix and then assemblingthe global Stiffness Matrix. Guassian Quadrature Integration was employed for finding the elements of the stiffness matrix.
- Define the nodal displacements at the boundaries. Reduce the dof of Global stiffness Matrix to only the unknown nodal displacements.
- Find all the unknown nodal displacements
- Calculate the stresses and strains at each element. Again Guassian Quadrature Integration was employed.
- The Average stess and strains are computed.

$$\bar{\sigma} = \frac{1}{v}\sum_{e=1}^{n} \int [D][B]\{u^e\}|J|\, dv \qquad \vec{\varepsilon} = \frac{1}{V}\sum_{e=1}^{n} \int [B]\{u^e\}|J|\, dV$$



- The Young's Modulus is the ratio of average stress to average strain.

$$E = \frac{\bar{\sigma}}{\bar{\varepsilon}}$$

Results

The homogeneous young's moduli in both X and Y directions were determined. The values of E_x and E_y reflect the material's stiffness or rigidity in the respective directions. Higher values indicate greater resistance to deformation, while lower values indicate greater flexibility or ease of deformation. Understanding the Young's moduli in both the X and Y directions allows for accurate modeling and prediction of the material's behavior under different loading conditions, such as tension, compression, or bending, in different orientations.

Material	Ex	Ey	vxy	Gxy
Graphene	1.13e+12	1.13e+12	0.186	358e+9
Polystyrene	3.4e+9	3.4e+9	0.34	0.75e+9

From the code, E was found to be 5.3728e+10

From Method of mixtures E=Ef*vf+Em*Vm= 5.9730e+10

Error = 0.10048

Code:

Retangular array:

```
model = createpde;
vf=0.05;
r=sqrt(double(vf)/pi);
R1 = [3,4,-0.5,0.5,0.5,-0.5,-0.5,-0.5,0.5,0.5]';
C1 = [1,0,0,r]';
C1 = [C1;zeros(length(R1) - length(C1),1)];
gd = [R1,C1];
sf = 'R1+C1';
ns = char('R1','C1');
ns = ns';
gd = decsg(gd,sf,ns);
geometryFromEdges(model,gd);
figure
pdegplot(model,"FaceLabels","on","EdgeLabels","on");
mesh1=generateMesh(model,"GeometricOrder","quadratic");
[p,e,t] = meshToPet(mesh1);
E1=findNodes(mesh1,"region","Edge",2);
E2=findNodes(mesh1,"region","Edge",4);
E3=findNodes(mesh1,"region","Edge",1);
E4=findNodes(mesh1,"region","Edge",3);
Nf = findNodes(mesh1,"region","Face",2);
Nm =findNodes(mesh1,"region","Face",1);
figure
pdemesh(model)
hold on
plot(mesh1.Nodes(1,Nf),mesh1.Nodes(2,Nf),"ok","MarkerFaceColor","g");
figure
pdemesh(model)
hold on
plot(mesh1.Nodes(1,Nm),mesh1.Nodes(2,Nm),"ok","MarkerFaceColor","r");
figure
pdemesh(model)
hold on
plot(mesh1.Nodes(1,E1),mesh1.Nodes(2,E1),"ok","MarkerFaceColor","m")
plot(mesh1.Nodes(1,E2),mesh1.Nodes(2,E2),"ok","MarkerFaceColor","m")
figure
pdemesh(model)
hold on
plot(mesh1.Nodes(1,E4),mesh1.Nodes(2,E4),"ok","MarkerFaceColor","y")
plot(mesh1.Nodes(1,E3),mesh1.Nodes(2,E3),"ok","MarkerFaceColor","y")
```

Diamond Array:

```
model = createpde;
vf=0.05;
r=sqrt(double(vf)/(2*pi));
R1 = [3,4,-0.5,0.5,0.5,-0.5,-0.5,-0.5,0.5,0.5]';
C0 = [1,0,0,r]';
C1 = [1,-0.5,-0.5,r]';
C2 = [1,0.5,-0.5,r]';
C3 = [1,0.5,0.5,r]';
C4 = [1,-0.5,0.5,r]';
C0 = [C0;zeros(length(R1) - length(C0),1)];
C1 = [C1;zeros(length(R1) - length(C1),1)];
C2 = [C2;zeros(length(R1) - length(C2),1)];
C3 = [C3;zeros(length(R1) - length(C3),1)];
C4 = [C4;zeros(length(R1) - length(C4),1)];
```

```
gd = [R1,C0,C1,C2,C3,C4];
sf = 'R1+C0+C1+C2+C3+C4';
ns = char('R1','C0','C1','C2','C3','C4');
ns = ns';
gd = decsg(gd,sf,ns);
geometryFromEdges(model,gd);
pdegplot(model,"FaceLabels","on","EdgeLabels","on");
mesh1=generateMesh(model,"GeometricOrder","quadratic");
[p,e,t] = meshToPet(mesh1);
E1=findNodes(mesh1,"region","Edge",[4,5,6]);
E2=findNodes(mesh1,"region","Edge",[8,7,9]);
E3=findNodes(mesh1,"region","Edge",[1,2,3]);
E4=findNodes(mesh1,"region","Edge",[10,11,12]);
Nf = findNodes(mesh1,"region","Face",[4,5,7,8,10]);
Nm=findNodes(mesh1,"region","Face",1);
figure
pdemesh(model)
hold on
plot(mesh1.Nodes(1,Nf),mesh1.Nodes(2,Nf),"ok","MarkerFaceColor","g");
figure
pdemesh(model)
hold on
plot(mesh1.Nodes(1,Nm),mesh1.Nodes(2,Nm),"ok","MarkerFaceColor","r");
figure
pdemesh(model)
hold on
plot(mesh1.Nodes(1,E1),mesh1.Nodes(2,E1),"ok","MarkerFaceColor","m")
plot(mesh1.Nodes(1,E2),mesh1.Nodes(2,E2),"ok","MarkerFaceColor","m")
figure
pdemesh(model)
hold on
plot(mesh1.Nodes(1,E3),mesh1.Nodes(2,E3),"ok","MarkerFaceColor","y")
plot(mesh1.Nodes(1,E4),mesh1.Nodes(2,E4),"ok","MarkerFaceColor","y")
```

Elemental Stiffness Matrix

```
% Element stiffness matrix calculation function
function Ke = calculateElementStiffness(coords, E, t)
    % Gauss quadrature points and weights
    gp = [1/6, 1/6, 2/3; 1/6, 2/3, 1/6];
    w = [1/6 1/6 1/6];

    % D Matrix - Constitutive Relations
    D=D_matrix(E);

    % Element stiffness matrix
    Ke = zeros(12);
    for i = 1:size(gp, 2)
        X1 = gp(1, i);
        X2 = gp(2, i);

        % Shape functions and derivatives
        N1 = (1-X1-X2)*(1-2*X1-2*X2);
        N2 = X1*(2*X1-1);
        N3 = X2*(2*X2-1);
        N4 = 4*X1*(1-X1-X2);
        N5 = 4*X1*X2;
        N6 = 4*X2*(1-X1-X2);

        dN1_dX1 = 4*X1+4*X2-3;
        dN1_dX2 = 4*X1+4*X2-3;
```

```
        dN2_dX1 = 4*X1-1;
        dN2_dX2 = 0;
        dN3_dX1 = 0;
        dN3_dX2 = 4*X2-1;
        dN4_dX1 = 4-8*X1-4*X2;
        dN4_dX2 = -4*X1;
        dN5_dX1 = 4*X2;
        dN5_dX2 = 4*X1;
        dN6_dX1 = -4*X2;
        dN6_dX2 = 4-4*X1-8*X2;

        % Jacobian matrix
        J = [dN1_dX1, dN2_dX1, dN3_dX1, dN4_dX1, dN5_dX1, dN6_dX1; dN1_dX2, dN2_dX2, dN3_dX2,
dN4_dX2, dN5_dX2, dN6_dX2] * coords;

        % Determinant of the Jacobian Matrix
        detJ = abs(det(J));

        % Inverse of the Jacobian Matrix
        invJ = inv(J);

        % B matrix (strain-displacement Matrix)
        P=[dN1_dX1, 0, dN2_dX1, 0, dN3_dX1, 0, dN4_dX1, 0, dN5_dX1, 0, dN6_dX1, 0; dN1_dX2, 0,
dN2_dX2, 0, dN3_dX2, 0, dN4_dX2, 0, dN5_dX2, 0, dN6_dX2, 0; 0, dN1_dX1, 0, dN2_dX1, 0, dN3_dX1, 0,
dN4_dX1, 0, dN5_dX1, 0, dN6_dX1; 0 , dN1_dX2, 0, dN2_dX2, 0, dN3_dX2, 0, dN4_dX2, 0, dN5_dX2, 0,
dN6_dX2];
        G=[invJ(1,1), invJ(1,2), 0 , 0 ; 0 , 0 , invJ(2,1), invJ(2,2); invJ(2,1), invJ(2,2),
invJ(1,1), invJ(1,2)];
        B = G*P;
        % Element stiffness matrix contribution
        Ke = Ke + B' * D * B * detJ * t * w(i);
    end
end
```

D Matrix Computation:

```
function D = D_matrix(E)
    D=zeros(3);
    v21=E(3,1)*E(2,1)/E(1,1);
    D(1,1)=E(1,1)/(1-(E(3,1)*v21));
    D(2,2)=E(2,1)/(1-(E(3,1)*v21));
    D(3,3)=E(4,1)/(1-(E(3,1)*v21));
    D(1,2)=E(2,1)*E(3,1)/(1-(E(3,1)*v21));
    D(2,1)=D(1,2);
end
```

Global Stiffness Matrix Assembly

```
Ef=[1.13e+12;1.13e+12;0.186;358e+9];
Em=[3.4e+9;3.4e+9;0.34;0.75e+9];
z=1;
n=size(p,2);
N=size(t,2);
K=zeros(n*2);
for i=1:N
    conn=t(:,i);
    coords=zeros([6,2]);
    for j=1:6
        coords(j,:)=p(:,conn(j,1))';
    end
    check=check_f_or_m(Nf,Nm,conn);
```

```
    if(check=='f')
        Ke=calculateElementStiffness(coords,Ef,z);
    elseif(check=='m')
        Ke=calculateElementStiffness(coords,Em,z);
    else
        continue;
    end
    for l=1:6
        for k=1:6
            nodei=conn(l);
            nodej=conn(k);
            K([2*nodei-1,2*nodei],[2*nodej-1,2*nodej])=Ke([2*l-1,2*l],[2*k-1,2*k]);
        end
    end
end
```

Solve u for dx and dy

```
dx=0.298025;
K_mod=K;
e=[E1,E2,E3,E4];
a=ones([1,size(E1,2)]);
v=[a,2*a,3*a,4*a];
[e,idX]=sort(e);
v=v(idX);
le=size(e,2);
count=0;
for i=9:le
    if(v(i)==1 || v(i)==2)
        true_node=2*e(i);
    else
        true_node=2*e(i)-1;
    end
    modified_node=true_node-count;
    K_mod(modified_node,:)=[];
    count=count+1;
end
K_mod=K_mod((9:end),:);
f_mod=zeros(2*n,1);
l1=size(E1,2);
for i=1:l1
    node=E1(i);
    f_mod(2*node)=0.01;
end
l2=size(E2,2);
for i=1:l2
    node=E2(i);
    f_mod(2*node)=0;
end
l3=size(E3,2);
for i=1:l3
    node=E3(i);
    f_mod(2*node-1)=-dx;
end
l4=size(E4,2);
for i=1:l4
    node=E4(i);
    f_mod(2*node-1)=dx;
end
f_u=zeros([size(K_mod,1),1]);
for i=1:l3
```

```
    true_node=2*E3(i)-1;
    f_u=f_u+(f_mod(true_node,1)*K_mod(:,true_node));
end
for i=1:l1
    true_node=2*E1(i);
    f_u=f_u+(f_mod(true_node,1)*K_mod(:,true_node));
end
f_u=-1*f_u;
count=0;
le=size(e,2);
for i=9:le
    if(v(i)==1 || v(i)==2)
        true_node=2*e(i);
    else
        true_node=2*e(i)-1;
    end
    modified_node=true_node-count;
    K_mod(:,modified_node)=[];
    count=count+1;
end
K_mod=K_mod(:,(9:end));
u_sol=inv(K_mod)*f_u;
for i=9:le
    if(v(i)==1 || v(i)==2)
        ind=2*e(i);
        p1=u_sol(1:ind-1,1);
        p2=u_sol(ind:end,1);
        if(v(i)==1)
            u_sol=[p1;0.01;p2];
        elseif(v(i)==2)
            u_sol=[p1;0;p2];
        else
        continue;
        end
    else
        ind=2*e(i)-1;
        p1=u_sol(1:ind-1,1);
        p2=u_sol(ind:end,1);
        if(v(i)==3)
            u_sol=[p1;-dx;p2];
        elseif(v(i)==4)
            u_sol=[p1;dx;p2];
        else
            continue;
        end
    end
end
u_sol=[-dx;0;-dx;0.01;dx;0.01;dx;0;u_sol];
```

Check node type

```
function check=check_f_or_m(Nf,Nm,conn)
    l=size(conn,1)-1;
    nf=size(Nf,2);
    nm=size(Nm,2);
    fs=0;
    ms=0;
    for i=1:l
        for j=1:nf
            if(conn(i)==Nf(j))
                fs=fs+1;
```

```
            end
        end
        for k=1:nm
            if(conn(i)==Nm(k))
                ms=ms+1;
            end
        end
    end
    if(fs==6)
        check='f';
    elseif(ms==6)
        check='m';
    else
        check='e';
    end
end
```

Check Element on Edge

```
function c_e=check_element(conn,E1,E2)
    c_e=0;
    for j=1:6
        check=check_edge(conn(j),E1,E2);
        c_e=c_e+check;
    end
end
```

Calculate E:

```
avg_strain=0;
avg_stress=0;
stress_edge3=0;
for m=1:N
    conn=t(:,m);
    coords=zeros([6,2]);
    for j=1:6
        coords(j,:)=p(:,conn(j,1))';
    end
    check=check_f_or_m(Nf,Nm,conn);
    if(check=='f')
        D=D_matrix(Ef);

    elseif(check=='m')
        D=D_matrix(Em);
    else
        continue;
    end
    ue=[];
    for j=1:6
        node=conn(j);
        ind=2*node-1;
        ue=[ue;u_sol(ind);u_sol(ind+1)];
    end
    gp = [1/6, 1/6, 2/3; 1/6, 2/3, 1/6];
    w = [1/6 1/6 1/6];
    strain_e=zeros([3,1]);
    for i = 1:size(gp, 2)
        X1 = gp(1, i);
        X2 = gp(2, i);

        % Shape functions and derivatives
```

```
        N1 = (1-X1-X2)*(1-2*X1-2*X2);
        N2 = X1*(2*X1-1);
        N3 = X2*(2*X2-1);
        N4 = 4*X1*(1-X1-X2);
        N5 = 4*X1*X2;
        N6 = 4*X2*(1-X1-X2);

        dN1_dX1 = 4*X1+4*X2-3;
        dN1_dX2 = 4*X1+4*X2-3;
        dN2_dX1 = 4*X1-1;
        dN2_dX2 = 0;
        dN3_dX1 = 0;
        dN3_dX2 = 4*X2-1;
        dN4_dX1 = 4-8*X1-4*X2;
        dN4_dX2 = -4*X1;
        dN5_dX1 = 4*X2;
        dN5_dX2 = 4*X1;
        dN6_dX1 = -4*X2;
        dN6_dX2 = 4-4*X1-8*X2;

        % Jacobian matrix
        J = [dN1_dX1, dN2_dX1, dN3_dX1, dN4_dX1, dN5_dX1, dN6_dX1; dN1_dX2, dN2_dX2, dN3_dX2,
dN4_dX2, dN5_dX2, dN6_dX2] * coords;

        % Determinant of the Jacobian matrix
        detJ = abs(det(J));

        % Inverse of the Jacobian matrix
        invJ = inv(J);

        % B matrix (strain-displacement matrix)
        P=[dN1_dX1, 0, dN2_dX1, 0, dN3_dX1, 0, dN4_dX1, 0, dN5_dX1, 0, dN6_dX1, 0; dN1_dX2, 0,
dN2_dX2, 0, dN3_dX2, 0, dN4_dX2, 0, dN5_dX2, 0, dN6_dX2, 0; 0, dN1_dX1, 0, dN2_dX1, 0, dN3_dX1, 0,
dN4_dX1, 0, dN5_dX1, 0, dN6_dX1; 0 , dN1_dX2, 0, dN2_dX2, 0, dN3_dX2, 0, dN4_dX2, 0, dN5_dX2, 0,
dN6_dX2];
        G=[invJ(1,1), invJ(1,2), 0 , 0 ; 0 , 0 , invJ(2,1), invJ(2,2); invJ(2,1), invJ(2,2),
invJ(1,1), invJ(1,2)];
        B = G*P;
        strain_e=strain_e+(B*ue*detJ * z * w(i));
    end
    Ae=abs(polyarea(coords(:,1),coords(:,2)));
    avg_strain=avg_strain+(strain_e*Ae*z);
    stress_e=D*strain_e;
    avg_stress=avg_stress+(stress_e*Ae*z);
    if(check_element(conn,E3,E4)~=0)
        stress_edge3=stress_edge3+stress_e;
    end
end
disp("Young's Modulus")
avg_stress=abs(avg_stress);
avg_strain=abs(avg_strain);
disp(avg_stress(1)/avg_strain(1));
```