Chapter 11
Plane stress

11.1 Introduction

This chapter deals with the static analysis of 2D solids, particularly in plane stress. Plane stress analysis refers to problems where the thickness is quite small when compared to other dimensions in the reference plane $x-y$. The loads and boundary conditions are applied at the reference or middle plane of the structure. Displacements are computed at the reference plane. The stresses related with z coordinates are assumed to be very small and not considered in the formulation. In this chapter we consider only isotropic, homogeneous materials, and the four-node quadrilateral (Q4).

The problem is defined in a domain Ω bounded by Γ, as illustrated in figure 11.1.

11.2 Displacements, strains and stresses

The plane stress problem considers two global displacements, u and v, defined in global directions x and y, respectively.

$$\mathbf{u}(x,y) = \begin{bmatrix} u(x,y) \\ v(x,y) \end{bmatrix} \tag{11.1}$$

Strains are obtained by derivation of displacements

$$\boldsymbol{\epsilon}(x,y) = \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial v}{\partial y} \\ \dfrac{\partial u}{\partial y} + \dfrac{\partial v}{\partial x} \end{bmatrix} \tag{11.2}$$

A.J.M. Ferreira, *MATLAB Codes for Finite Element Analysis: Solids and Structures,* Solid Mechanics and Its Applications 157,

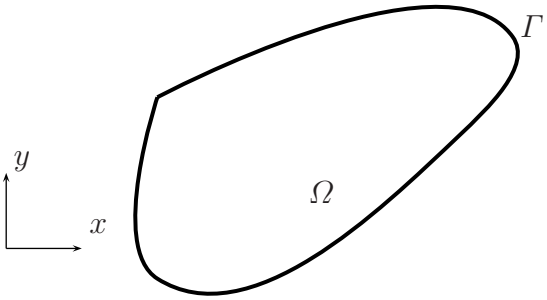

Fig. 11.1 Plane stress: illustration of domain (Ω) and boundary (Γ)

By assuming a linear elastic material, we obtain stresses as

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} = \mathbf{C}\boldsymbol{\epsilon} = \begin{bmatrix} \frac{E}{1-\nu^2} & \frac{\nu E}{1-\nu^2} & 0 \\ \frac{\nu E}{1-\nu^2} & \frac{E}{1-\nu^2} & 0 \\ 0 & 0 & G = \frac{E}{2(1+\nu)} \end{bmatrix} \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{bmatrix} \tag{11.3}$$

where E is the modulus of elasticity and ν the Poisson's coefficient.

The static equilibrium equations are defined as

$$\frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + b_x = 0 \tag{11.4}$$

$$\frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + b_y = 0 \tag{11.5}$$

where b_x, b_y are body forces.

11.3 Boundary conditions

Essential or displacement boundary conditions are applied on the boundary displacement part Γ_u, as

$$\mathbf{u} = \hat{\mathbf{u}} \tag{11.6}$$

Natural or force boundary conditions are applied on Γ_t, so that

$$\boldsymbol{\sigma}_n = \hat{\mathbf{t}} \tag{11.7}$$

where $\hat{\mathbf{t}}$ is the surface traction per unit area, and $\boldsymbol{\sigma}_n$ the normal vector to the plate.

If necessary, $\boldsymbol{\sigma}_n$ can be computed in natural coordinates by

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_x n_x + \tau_{xy} n_y \\ \tau_{xy} n_x + \sigma_y n_y \end{bmatrix} = \begin{bmatrix} n_x & 0 & n_y \\ 0 & n_y & n_x \end{bmatrix} \begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} \tag{11.8}$$

11.4 Potential energy

The potential energy can be defined as

$$\Pi = U - W \tag{11.9}$$

where U is the elastic strain deformation,

$$U = \frac{1}{2} \int_{\Omega} h \boldsymbol{\epsilon}^T \boldsymbol{\sigma} d\Omega = \frac{1}{2} \int_{\Omega} h \boldsymbol{\epsilon}^T \mathbf{C} \boldsymbol{\epsilon} d\Omega \tag{11.10}$$

The energy produced by the external forces is given by

$$W = \int_{\Omega} h \mathbf{u}^T \mathbf{b} d\Omega + \int_{\Gamma_t} h \mathbf{u}^T \hat{\mathbf{t}} d\Gamma \tag{11.11}$$

11.5 Finite element discretization

Given a domain denoted by Ω^e and a boundary denoted by Γ^e, the n-noded finite element displacement vector is defined by $2n$ degrees of freedom,

$$\mathbf{u}^e = \begin{bmatrix} u_1 & v_1 & u_2 & v_2 & ... & u_n & v_n \end{bmatrix}^T \tag{11.12}$$

11.6 Interpolation of displacements

The displacement vector in each element is interpolated by the nodal displacements as

$$u = \sum_{i=1}^{n} N_i^e u_i; \quad v = \sum_{i=1}^{n} N_i^e v_i \tag{11.13}$$

where N_i^e denote the element shape functions. This can also be expressed in matrix form as

$$\mathbf{u} = \begin{bmatrix} N_1^e & 0 & 0 & N_2^e & 0 & ... & N_n^e & 0 \\ 0 & N_1^e & 0 & 0 & N_2^e & 0 & ... & N_n^e \end{bmatrix} \mathbf{u}^e = \mathbf{N}\mathbf{u}^e \tag{11.14}$$

The strain vector can be obtained by derivation of the displacements as

$$\boldsymbol{\epsilon} = \begin{bmatrix} \frac{\partial N_1^e}{\partial x} & 0 & \frac{\partial N_2^e}{\partial x} & 0 & ... & \frac{\partial N_n^e}{\partial x} & 0 \\ 0 & \frac{\partial N_1^e}{\partial y} & 0 & \frac{\partial N_2^e}{\partial y} & 0 & ... & \frac{\partial N_n^e}{\partial y} \\ \frac{\partial N_1^e}{\partial y} & \frac{\partial N_1^e}{\partial x} & \frac{\partial N_2^e}{\partial y} & \frac{\partial N_2^e}{\partial x} & ... & \frac{\partial N_n^e}{\partial y} & \frac{\partial N_n^e}{\partial x} \end{bmatrix} \mathbf{u}^e = \mathbf{B}\mathbf{u}^e \tag{11.15}$$

where $\mathbf{B}$ is the strain-displacement matrix. This matrix is needed for computation of the stiffness matrix, and the stress vector at each element.

11.7 Element energy

The total potential energy can de defined at each element by

$$\Pi^e = U^e - W^e \tag{11.16}$$

where the strain energy is defined as

$$U^e = \frac{1}{2}\int_{\Omega^e} h\boldsymbol{\epsilon}^T\boldsymbol{\sigma}d\Omega^e = \frac{1}{2}\int_{\Omega^e} h\boldsymbol{\epsilon}^T\mathbf{C}\boldsymbol{\epsilon}d\Omega^e \tag{11.17}$$

and the energy produced by the surface tractions is given by

$$W^e = \int_{\Omega^e} h\mathbf{u}^T\mathbf{b}d\Omega^e + \int_{\Gamma^e} h\mathbf{u}^T\hat{\mathbf{t}}d\Gamma^e \tag{11.18}$$

We can introduce these expressions into the total potential energy as

$$\Pi^e = \frac{1}{2}\mathbf{u}^{eT}\mathbf{K}^e\mathbf{u}^e - \mathbf{u}^{eT}\mathbf{f}^e \tag{11.19}$$

where the element stiffness matrix is obtained as

$$\mathbf{K}^e = \int_{\Omega^e} h\mathbf{B}^T\mathbf{C}\mathbf{B}d\Omega^e \tag{11.20}$$

and the vector of nodal forces is obtained as

$$\mathbf{f}^e = \int_{\Omega^e} h\mathbf{N}^T\mathbf{b}d\Omega^e + \int_{\Gamma^e} h\mathbf{N}^T\hat{\mathbf{t}}d\Gamma^e \tag{11.21}$$

11.8 Quadrilateral element Q4

We consider a quadrilateral element, illustrated in figure 11.2. The element is defined by four nodes in natural coordinates (ξ, η). The coordinates are interpolated as

$$x = \sum_{i=1}^{4} N_i x_i; \quad y = \sum_{i=1}^{4} N_i y_i \tag{11.22}$$

where N_i are the Lagrange shape functions, given by

$$N_1(\xi, \eta) = l_1(\xi) l_1(\eta) = \frac{1}{4}(1-\xi)(1-\eta) \tag{11.23}$$

$$N_2(\xi, \eta) = l_2(\xi) l_1(\eta) = \frac{1}{4}(1+\xi)(1-\eta) \tag{11.24}$$

$$N_3(\xi, \eta) = l_2(\xi) l_2(\eta) = \frac{1}{4}(1+\xi)(1+\eta) \tag{11.25}$$

$$N_4(\xi, \eta) = l_1(\xi) l_2(\eta) = \frac{1}{4}(1-\xi)(1+\eta) \tag{11.26}$$

Displacements are interpolated as

$$u = \sum_{i=1}^{4} N_i u_i; \quad v = \sum_{i=1}^{4} N_i v_i \tag{11.27}$$

where u, v are the displacements at any point in the element and $u_i, v_i;\ i = 1, ..., n$ the nodal displacements.

Derivatives $\dfrac{\partial}{\partial \xi}, \dfrac{\partial}{\partial \eta}$, can be found as

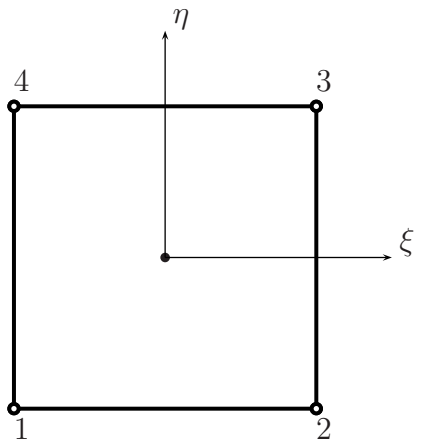

Fig. 11.2 Quadrilateral Q4 element in natural coordinates

$$\begin{bmatrix} \dfrac{\partial}{\partial \xi} \\ \\ \dfrac{\partial}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} \begin{bmatrix} \dfrac{\partial}{\partial x} \\ \\ \dfrac{\partial}{\partial y} \end{bmatrix} \tag{11.28}$$

In matricial form, we can write relations (11.28) as

$$\frac{\partial}{\partial \xi} = \mathbf{J} \frac{\partial}{\partial \mathbf{x}} \tag{11.29}$$

where $\mathbf{J}$ is the Jacobian operator, relating natural and global coordinates. The derivatives with respect to the global coordinates can be found as

$$\frac{\partial}{\partial \mathbf{x}} = \mathbf{J}^{-1} \frac{\partial}{\partial \xi} \tag{11.30}$$

Note that in very distorted elements the Jacobian inverse, $\mathbf{J}^{-1}$ may not exist.

The stiffness matrix is then obtained by

$$\mathbf{K} = \int_V \mathbf{B}^T \mathbf{C} \mathbf{B} dV \tag{11.31}$$

Note that $\mathbf{B}$ depends on the element natural coordinates ξ, η. The element volume is given by

$$dV = h \; det\mathbf{J} d\xi d\eta \tag{11.32}$$

where $det\mathbf{J}$ is the determinant of the Jacobian matrix and h the thickness of the plate. The integral in the stiffness matrix is computed numerically by Gauss quadrature in two dimensions. Taking

$$\mathbf{K} = h \int_A \mathbf{F} d\xi d\eta \tag{11.33}$$

where $\mathbf{F} = \mathbf{B}^T \mathbf{C} \mathbf{B} det\mathbf{J}$, we perform the numerical computation by

$$\mathbf{K} = \sum_{i,j,k} \mathbf{F}_{i,j} \alpha_{i,j} \tag{11.34}$$

Here $\mathbf{F}_{i,j}$ is a matrix dependent on the natural points (ξ_i, η_j). Integration points (ξ_i, η_j) and integration weights depend on the type of integration the user wishes to perform. In the four-node element we can use a 2×2 numerical integration for exact integration.

The Gauss quadrature in two dimensions replaces the integration by a summation over Gauss points, by

$$\int_{-1}^{1} \int_{-1}^{1} F(\xi, \eta) d\xi d\eta = \sum_{i=1}^{p} \sum_{j=1}^{q} w_i w_j F(\xi_i, \eta_j) \tag{11.35}$$

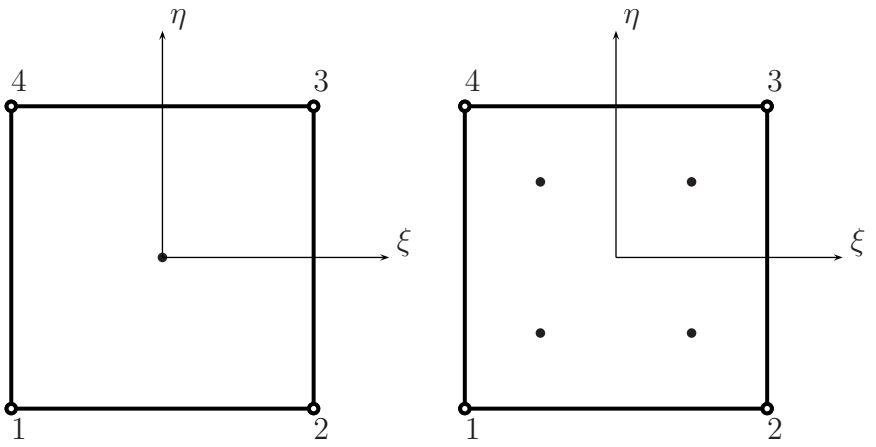

Fig. 11.3 One Gauss point integration ($\xi = 0, \eta = 0$); two Gauss point integration ($\xi, \eta = \pm\frac{1}{\sqrt{3}}$)

where p, q are the number of integrating points in the ξ, η directions, respectively, and w_i, w_j are the corresponding weights, as illustrated in figure 11.3, for some integration orders.

The stiffness matrix can be computed using 2×2 Gauss points as

$$\mathbf{K}^e = \int_{\Omega^e} h\mathbf{B}^T\mathbf{C}\mathbf{B}d\Omega^e = \int_{-1}^{1}\int_{-1}^{1} h\mathbf{B}^T\mathbf{C}\mathbf{B}det\mathbf{J}d\xi d\eta = h\sum_{i=1}^{2}\sum_{j=1}^{2} \mathbf{B}^T\mathbf{C}\mathbf{B}det\mathbf{J}w_i w_j \tag{11.36}$$

All Gauss points have unit weight, in this integration rule.

11.9 Example: plate in traction

We consider a thin plate under uniform traction forces at its extremes. The problem is illustrated in figure 11.4.

In figure 11.5 we show the finite element mesh considering 10×5 elements. In figure 11.6 the deformed shape of the problem is illustrated.

The MATLAB code for this problem is (problem17.m).

```
%.....................................................................
% MATLAB codes for Finite Element Analysis
```

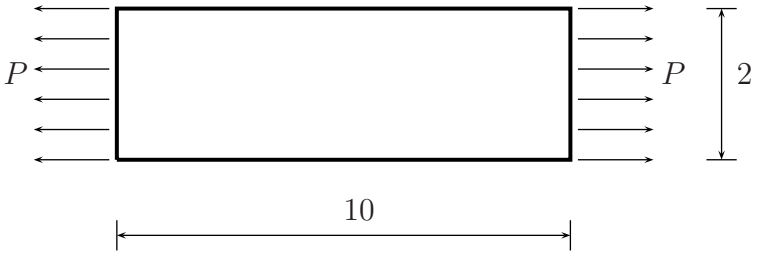

Fig. 11.4 Thin plate in traction, problem17.m

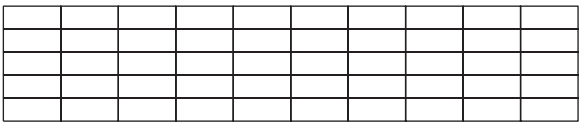

Fig. 11.5 Finite element mesh for a thin plate in tension

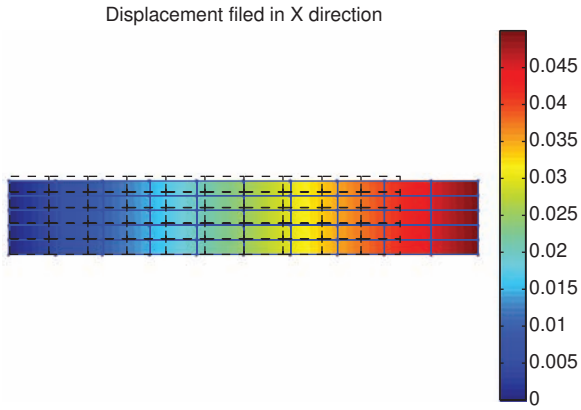

Fig. 11.6 Plate in traction: displacements in XX

```
% problem17.m
% 2D problem: thin plate in tension
% antonio ferreira 2008

% clear memory
clear all;colordef white;clf
```

```
% materials
E  = 10e7;     poisson = 0.30;

% matriz C
C=E/(1-poisson^2)*[1 poisson 0;poisson 1 0;0 0 (1-poisson)/2];

% load
P = 1e6;

%Mesh generation
Lx=5;
Ly=1;
numberElementsX=20;
numberElementsY=10;
numberElements=numberElementsX*numberElementsY;
[nodeCoordinates, elementNodes] = ...
    rectangularMesh(Lx,Ly,numberElementsX,numberElementsY);
xx=nodeCoordinates(:,1);
yy=nodeCoordinates(:,2);
drawingMesh(nodeCoordinates,elementNodes,'Q4','k-');
numberNodes=size(xx,1);

% GDof: global number of degrees of freedom
GDof=2*numberNodes;

% calculation of the system stiffness matrix
stiffness=formStiffness2D(GDof,numberElements,...
    elementNodes,numberNodes,nodeCoordinates,C,1,1);

% boundary conditions
fixedNodeX=find(nodeCoordinates(:,1)==0);  % fixed in XX
fixedNodeY=find(nodeCoordinates(:,2)==0);  % fixed in YY
prescribedDof=[fixedNodeX; fixedNodeY+numberNodes];

% force vector (distributed load applied at xx=Lx)
force=zeros(GDof,1);
rightBord=find(nodeCoordinates(:,1)==Lx);
force(rightBord)=P*Ly/numberElementsY;
force(rightBord(1))=P*Ly/numberElementsY/2;
force(rightBord(end))=P*Ly/numberElementsY/2;

% solution
displacements=solution(GDof,prescribedDof,stiffness,force);
```

```
% displacements
disp('Displacements')
jj=1:GDof; format
f=[jj; displacements'];
fprintf('node U\n')
fprintf('%3d %12.8f\n',f)
UX=displacements(1:numberNodes);
UY=displacements(numberNodes+1:GDof);
scaleFactor=10;

% deformed shape
figure
drawingField(nodeCoordinates+scaleFactor*[UX UY],...
    elementNodes,'Q4',UX);%U XX
hold on
drawingMesh(nodeCoordinates+scaleFactor*[UX UY],...
    elementNodes,'Q4','k-');
drawingMesh(nodeCoordinates,elementNodes,'Q4','k--');
colorbar
title('U XX  (on deformed shape)')
axis off

% stresses at nodes
stresses2D(GDof,numberElements,elementNodes,numberNodes,...
    nodeCoordinates,displacements,UX,UY,C,scaleFactor)
```

11.10 Example: beam in bending

We show in this example (code problem18.m) how MATLAB can help in the computation of a beam in bending (figure 11.7). Note some of the differences to problem17.m:

- The boundary conditions are different, in this case both u and v are fixed at $x = 0$.
- The applied force is in the y direction, so care must be taken to ensure that degrees of freedom are properly assigned.

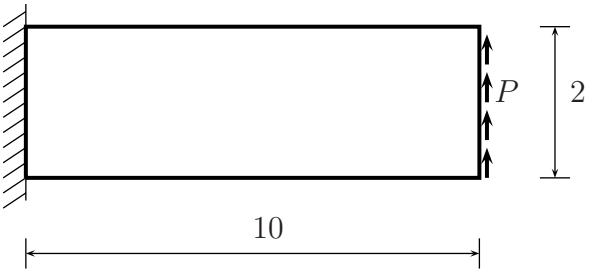

Fig. 11.7 Thin plate in bending, problem18.m

```
%.....................................................................

% MATLAB codes for Finite Element Analysis
% problem18.m
% 2D problem: beam in bending
% antonio ferreira 2008

% clear memory
clear all;colordef white;clf

% materials
E  = 10e7;     poisson = 0.30;

% matriz C
C=E/(1-poisson^2)*[1 poisson 0;poisson 1 0;0 0 (1-poisson)/2];

% load
P = 1e6;

%Mesh generation
Lx=5;
Ly=1;
numberElementsX=20;
numberElementsY=10;
numberElements=numberElementsX*numberElementsY;
[nodeCoordinates, elementNodes] = ...
    rectangularMesh(Lx,Ly,numberElementsX,numberElementsY);
xx=nodeCoordinates(:,1);
yy=nodeCoordinates(:,2);
```

```
drawingMesh(nodeCoordinates,elementNodes,'Q4','k-');
numberNodes=size(xx,1);

% GDof: global number of degrees of freedom
GDof=2*numberNodes;

% computation of the system stiffness matrix
stiffness=formStiffness2D(GDof,numberElements,...
    elementNodes,numberNodes,nodeCoordinates,C,1,1);

% boundary conditions
fixedNodeX=find(nodeCoordinates(:,1)==0);  % fixed in XX
fixedNodeY=find(nodeCoordinates(:,1)==0);  % fixed in YY
prescribedDof=[fixedNodeX; fixedNodeY+numberNodes];

% force vector (distributed load applied at xx=Lx)
force=zeros(GDof,1);
rightBord=find(nodeCoordinates(:,1)==Lx);
force(rightBord+numberNodes)=P*Ly/numberElementsY;
force(rightBord(1)+numberNodes)=P*Ly/numberElementsY/2;
force(rightBord(end)+numberNodes)=P*Ly/numberElementsY/2;
% solution
displacements=solution(GDof,prescribedDof,stiffness,force);

% displacements and deformed shape
disp('Displacements')
jj=1:GDof; format
f=[jj; displacements'];
fprintf('node U\n')
fprintf('%3d %12.8f\n',f)
UX=displacements(1:numberNodes);
UY=displacements(numberNodes+1:GDof);
scaleFactor=0.1;

figure
drawingField(nodeCoordinates+scaleFactor*[UX UY],...
    elementNodes,'Q4',UX);%U XX
hold on
drawingMesh(nodeCoordinates+scaleFactor*[UX UY],...
    elementNodes,'Q4','k-');
drawingMesh(nodeCoordinates,elementNodes,'Q4','k--');
colorbar
title('U XX  (on deformed shape)')
```

```
axis off

% stresses at nodes
stresses2D(GDof,numberElements,...
    elementNodes,numberNodes,nodeCoordinates,...
    displacements,UX,UY,C,scaleFactor);
```

In figure 11.8 we show the evolution of displacements u on top of the deformed shape of the beam. If the user wishes to plot another displacement, just change the number of the displacement component upon calling drawingField.m.

In figure 11.9 we show the evolution of σ_x stress in the beam. If the user wishes to plot another stress, just change the number of the stress component upon calling stresses2D.m.

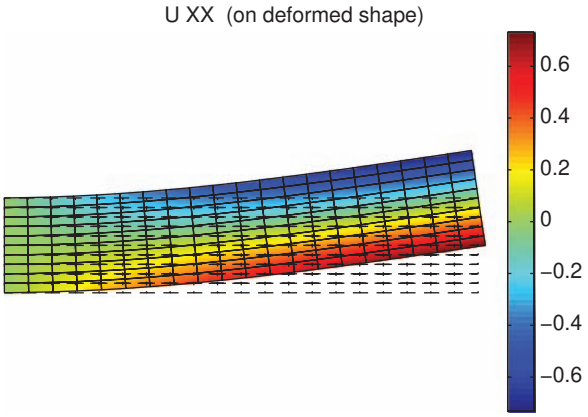

Fig. 11.8 Beam in bending: plot of displacements u

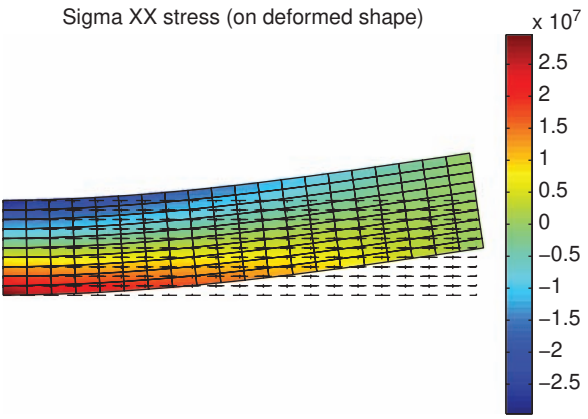

Fig. 11.9 Beam in bending: plot of σ_x stresses

Functions formStiffness2D.m and stresses2D.m are listed below.

```
%.....................................................................

function [stiffness,mass]=formStiffness2D(GDof,numberElements,...
    elementNodes,numberNodes,nodeCoordinates,C,rho,thickness)

% compute stiffness matrix (and mass matrix)
% for plane stress Q4 elements

stiffness=zeros(GDof);
mass=zeros(GDof);

% 2 by 2 quadrature
[gaussWeights,gaussLocations]=gaussQuadrature('complete');

for e=1:numberElements
  indice=elementNodes(e,:);
  elementDof=[ indice indice+numberNodes ];
  ndof=length(indice);

  % cycle for Gauss point
  for q=1:size(gaussWeights,1)
    GaussPoint=gaussLocations(q,:);
    xi=GaussPoint(1);
    eta=GaussPoint(2);

% shape functions and derivatives
    [shapeFunction,naturalDerivatives]=shapeFunctionQ4(xi,eta)

% Jacobian matrix, inverse of Jacobian,
% derivatives w.r.t. x,y
    [Jacob,invJacobian,XYderivatives]=...
        Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

%  B matrix
    B=zeros(3,2*ndof);
    B(1,1:ndof)       = XYderivatives(:,1)';
    B(2,ndof+1:2*ndof)  = XYderivatives(:,2)';
    B(3,1:ndof)       = XYderivatives(:,2)';
    B(3,ndof+1:2*ndof)  = XYderivatives(:,1)';

% stiffness matrix
```

```
    stiffness(elementDof,elementDof)=...
        stiffness(elementDof,elementDof)+...
        B'*C*thickness*B*gaussWeights(q)*det(Jacob);
% mass matrix
    mass(indice,indice)=mass(indice,indice)+...
        shapeFunction*shapeFunction'*...
        rho*thickness*gaussWeights(q)*det(Jacob);
    mass(indice+numberNodes,indice+numberNodes)=...
        mass(indice+numberNodes,indice+numberNodes)+...
        shapeFunction*shapeFunction'*...
        rho*thickness*gaussWeights(q)*det(Jacob);

  end
end
```

```
%.............................................................

function stresses2D(GDof,numberElements,...
    elementNodes,numberNodes,nodeCoordinates,...
    displacements,UX,UY,C,scaleFactor)

% 2 by 2 quadrature
[gaussWeights,gaussLocations]=gaussQuadrature('complete');

% stresses at nodes
stress=zeros(numberElements,size(elementNodes,2),3);
stressPoints=[-1 -1;1 -1;1 1;-1 1];

for e=1:numberElements
  indice=elementNodes(e,:);
  elementDof=[ indice indice+numberNodes ];
  nn=length(indice);
  for q=1:size(gaussWeights,1)
    pt=gaussLocations(q,:);
    wt=gaussWeights(q);
    xi=pt(1);
    eta=pt(2);
% shape functions and derivatives
    [shapeFunction,naturalDerivatives]=shapeFunctionQ4(xi,eta)

% Jacobian matrix, inverse of Jacobian,
```

```
% derivatives w.r.t. x,y
    [Jacob,invJacobian,XYderivatives]=...
        Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

%  B matrix
    B=zeros(3,2*nn);
    B(1,1:nn)       = XYderivatives(:,1)';
    B(2,nn+1:2*nn)  = XYderivatives(:,2)';
    B(3,1:nn)       = XYderivatives(:,2)';
    B(3,nn+1:2*nn)  = XYderivatives(:,1)';

% element deformation
    strain=B*displacements(elementDof);
    stress(e,q,:)=C*strain;
  end
end

% drawing stress fields
% on top of the deformed shape
figure
drawingField(nodeCoordinates+scaleFactor*[UX UY],...
    elementNodes,'Q4',stress(:,:,1));%sigma XX
hold on
drawingMesh(nodeCoordinates+scaleFactor*[UX UY],...
    elementNodes,'Q4','k-');
drawingMesh(nodeCoordinates,elementNodes,'Q4','k--');
colorbar
title('Sigma XX stress (on deformed shape)')
axis off
```

Some functions are used for all 2D problems (even for plates). Function shapeFunctionQ4.m computes the shape functions and derivatives of the shape functions with respect to natural ξ, η coordinates. Function Jacobian.m computes the Jacobian matrix, and its inverse. The computation of Gauss point locations and weights is made in function gaussQuadrature.m. The listing of these functions is given as follows

```
% ..........................................................
    function [shape,naturalDerivatives]=shapeFunctionQ4(xi,eta)

    % shape function and derivatives for Q4 elements
```

```
    % shape : Shape functions
    % naturalDerivatives: derivatives w.r.t. xi and eta
    % xi, eta: natural coordinates (-1 ... +1)

    shape=1/4*[ (1-xi)*(1-eta);(1+xi)*(1-eta);
            (1+xi)*(1+eta);(1-xi)*(1+eta)];
    naturalDerivatives=...
           1/4*[-(1-eta), -(1-xi);1-eta,    -(1+xi);
         1+eta,      1+xi;-(1+eta),   1-xi];

    end % end function shapeFunctionQ4
```

```
% ..........................................................

    function [JacobianMatrix,invJacobian,XYDerivatives]=...
        Jacobian(nodeCoordinates,naturalDerivatives)

    % JacobianMatrix    : Jacobian matrix
    % invJacobian : inverse of Jacobian Matrix
    % XYDerivatives  : derivatives w.r.t. x and y
    % naturalDerivatives  : derivatives w.r.t. xi and eta
    % nodeCoordinates  : nodal coordinates at element level

    JacobianMatrix=nodeCoordinates'*naturalDerivatives;
    invJacobian=inv(JacobianMatrix);
    XYDerivatives=naturalDerivatives*invJacobian;

    end % end function Jacobian
```

```
% ..........................................................

    function [weights,locations]=gaussQuadrature(option)
    % Gauss quadrature for Q4 elements
    % option 'complete' (2x2)
    % option 'reduced'  (1x1)
    % locations: Gauss point locations
    % weights: Gauss point weights
```

```
switch option
    case 'complete'

    locations=...
      [ -0.577350269189626 -0.577350269189626;
         0.577350269189626 -0.577350269189626;
         0.577350269189626  0.577350269189626;
        -0.577350269189626  0.577350269189626];
    weights=[ 1;1;1;1];

    case 'reduced'

    locations=[0 0];
    weights=[4];
end

end  % end function gaussQuadrature
```

Note that we analyse free vibrations with 2D elements, the function formStiffness2D.m already computes the mass matrix. The analysis is left to the reader.