

Isoparametric Formulation

1 INTRODUCTION

Isoparametric elements were first developed by Taig in 1958. However, related works were published only in 1966. The development of isoparametric formulation provided the finite element method its versatility, beauty, flexibility and power. Elements with curved boundaries could be generated using this concept. Such elements are useful in modelling curved and irregularly shaped boundaries accurately and in grading a mesh from coarse to fine. The isoparametric elements have been successfully employed for solving one-, two- and three-dimensional problems. These include elements used for cable structures, and plane, plate, shell and solid problems. The concept of isoparametric mapping is being used in several mesh generation algorithms. Special elements used for modelling crack-tip singularity which have wide applications in fracture mechanics, are also based on the isoparametric concept. It is needless to state that these applications are not limited to structural mechanics problems alone. Problems of heat conduction, fluid flow, magnetism, and other non-structural problems also have been solved successfully making use of this excellent idea.

In isoparametric formulation, we make use of certain natural coordinates usually denoted by ξ , η , ζ . As a result, the Jacobian of the transformation matrix needs to be invoked. The expressions for the element stiffness coefficients and the element load vector become complicated, necessitating the use of numerical integration schemes. The Gauss quadrature is most often used for this purpose.

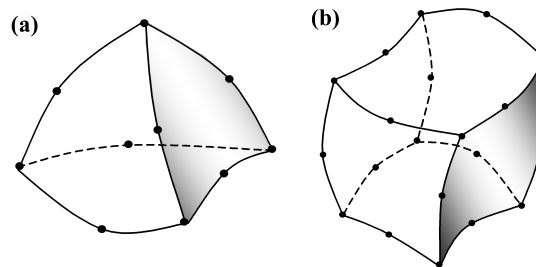


Figure 1.1 Isoparametric quadratic (a) tetrahedron and (b) hexahedron elements with curved edges.

Fig. 1.1a shows a quadratic isoparametric tetrahedron element which has 10 boundary nodes and curved edges. Fig. 1.1b depicts a quadratic hexahedral element with curved edges which has 20 nodes. This element is a member of the so-called serendipity family and is one of the most popular elements currently used for modelling three-dimensional continua. A finite element mesh for an elastic inclusion problem that makes use of two-dimensional eight-noded isoparametric quadrilateral elements is shown in Fig. 1.2.

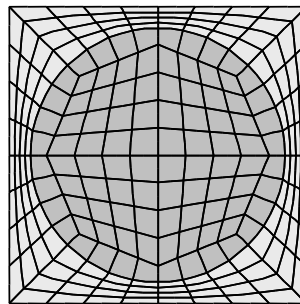


Figure 1.2 A mesh containing eight-noded plane isoparametric quadrilateral elements used for modelling an elastic inclusion problem.

In this chapter, we shall first see the overall idea of isoparametric formulation. Specific case of a four noded isoparametric quadrilateral element is then addressed. A simple computer program is presented to demonstrate the basic concepts of finite element programming. The program makes use of two simple classes, viz. the matrix and the vector classes. Thus, the object oriented programming concepts are introduced in an elementary way.

2 SUB, SUPER AND ISOPARAMETRIC FORMULATIONS

By means of isoparametric formulation, we can produce elements with curved and irregular boundaries. This is done by making use of a mapping from the physical space of the actual curved element into a natural coordinate space of a parent element with straight and regular boundaries as shown in Fig. 2.1.

Let the coordinates of the nodes of the element in the physical space be denoted by $x_i, y_i, z_i, i = 1$ to n , where n is the number of nodes the element has. These nodes on the physical element are mapped onto the corresponding nodes of the parent element as shown in Fig. 2.1. The nodal coordinates of the parent element are known in terms of the natural coordinate system (ξ, η, ζ) , and those of the physical element are known in terms of the physical coordinate system (x, y, z) . In Fig. 2.1b, we see a 20-noded regular hexahedral element. The nodal coordinates of the parent element in natural coordinate system lie between the limits -1 and $+1$. Thus, the nodes A and B of the physical element correspond to nodes a and b of the parent element, whose coordinates are $(+1, -1, +1)$ and $(+1, -1, -1)$ respectively with respect to the natural coordinate system. The physical coordinates of these nodes could be represented as (x_A, y_A, z_A) and (x_B, y_B, z_B) respectively.

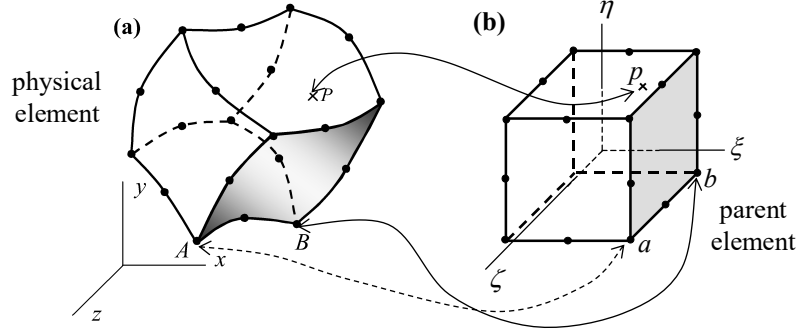


Figure 2.1 (a) A curved hexahedral element in physical space being mapped onto (b) a regular hexahedron in the natural coordinate space.

The mapping between the natural coordinates (ξ, η, ζ) and the physical coordinates (x, y, z) is related by certain functions $N'_i(\xi, \eta, \zeta)$. Thus, if we know the (ξ, η, ζ) coordinates of a point p in the natural coordinate system, the coordinates of the corresponding point P in the physical coordinate system (x, y, z) are given by

$$x = \sum_{i=1}^n N'_i x_i, \quad y = \sum_{i=1}^n N'_i y_i, \quad \text{and} \quad z = \sum_{i=1}^n N'_i z_i, \quad 2.1$$

where n is the number of nodes on the element.

Let us consider the displacement field of the element next. Let the displacement components of the i^{th} node be denoted by u_i, v_i, w_i , with $i = 1$ to n . Then the components of the displacement vector at any point (either interior or exterior) of the element are given by interpolation (as we had seen earlier in Chapter 10, Eq. 10.3.2). Thus, we have

$$u = \sum_{i=1}^n N_i u_i, \quad v = \sum_{i=1}^n N_i v_i, \quad \text{and} \quad w = \sum_{i=1}^n N_i w_i. \quad 2.2$$

It may be noted that N'_i and N_i appearing in Eqs. 2.1 and 2.2 are polynomial functions of the natural coordinates (ξ, η, ζ) . Depending on the relative degree of these two sets of polynomials, we have the following three types of possible finite element formulations:

- (i) *subparametric formulation* wherein N'_i are of degree lower than N_i ,
- (ii) *isoparametric formulation* when N'_i are same as N_i , and
- (iii) *superparametric formulation* where N'_i are of higher degree than N_i .

It can be proved that amongst the three formulations listed above, only the isoparametric formulation is mathematically correct and hence valid. For this reason, we shall consider only the isoparametric formulation in this book.

3 THE ISOPARAMETRIC FORMULATION

We have seen in the above section that in the isoparametric finite element formulation we use the same set of polynomial functions for the interpolation of the displacement field and for the mapping between the physical and natural coordinate systems. Thus, we have

$$u = \sum_{i=1}^n N_i u_i, \quad v = \sum_{i=1}^n N_i v_i, \quad \text{and} \quad w = \sum_{i=1}^n N_i w_i, \quad 3.1$$

and

$$x = \sum_{i=1}^n N_i x_i, \quad y = \sum_{i=1}^n N_i y_i, \quad \text{and} \quad z = \sum_{i=1}^n N_i z_i. \quad 3.2$$

In order to arrive at the element stiffness coefficients (and also the element load vector, which we shall consider subsequently), we need to consider the strain-displacement relations given by Eq. 2.2.2. These relations make use of derivatives of the displacement field with respect to the physical coordinates (x, y, z) . On the other hand, the interpolation functions N_i 's given by Eq. 3.1 are available in terms of the natural coordinates (ξ, η, ζ) . Consequently, we need to invoke the chain rule of partial differentiation given by

$$\frac{\partial}{\partial x} = \frac{\partial}{\partial \xi} \frac{\partial \xi}{\partial x} + \frac{\partial}{\partial \eta} \frac{\partial \eta}{\partial x} + \frac{\partial}{\partial \zeta} \frac{\partial \zeta}{\partial x}.$$

The above rule, along with two companion expressions for the partial derivatives with respect to y and z can conveniently be written together using matrix notation as

$$\begin{Bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{Bmatrix} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} & \frac{\partial \zeta}{\partial x} \\ \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} & \frac{\partial \zeta}{\partial y} \\ \frac{\partial \xi}{\partial z} & \frac{\partial \eta}{\partial z} & \frac{\partial \zeta}{\partial z} \end{bmatrix} \begin{Bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \\ \frac{\partial}{\partial \zeta} \end{Bmatrix}.$$

However, unfortunately, we do not have means to evaluate the elements of the transformation matrix (the 3×3 square matrix) given above. Hence, we try the inverse relationship

$$\begin{Bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \\ \frac{\partial}{\partial \zeta} \end{Bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} \begin{Bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{Bmatrix}. \quad 3.3$$

The transformation matrix in the above is called the *Jacobian matrix*, and is denoted by $[J]$. That is,

$$[J] = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix}. \quad 3.4a$$

The Jacobian matrix can be written concisely using the comma notation for partial derivatives as

$$[J] = \begin{bmatrix} x_{,\xi} & y_{,\xi} & z_{,\xi} \\ x_{,\eta} & y_{,\eta} & z_{,\eta} \\ x_{,\zeta} & y_{,\zeta} & z_{,\zeta} \end{bmatrix}. \quad 3.4b$$

Hence, we have

$$\begin{Bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{Bmatrix} = J^{-1} \begin{Bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \\ \frac{\partial}{\partial \zeta} \end{Bmatrix}. \quad 3.5$$

The elements of the Jacobian matrix are evaluated by making use of Eqs. 3.2 and 3.4. For example, the first element of $[J]$, viz. J_{11} is obtained as

$$J_{11} = \frac{\partial x}{\partial \xi} = \sum_{i=1}^n \frac{\partial N_i}{\partial \xi} x_i = \sum_{i=1}^n N_{i,\xi} x_i.$$

Once the interpolation polynomials N_i are known, the derivatives appearing in the above equation, viz. $\partial N_i / \partial \xi$, can be readily evaluated. It may be recalled that x_i are the nodal coordinates (in terms of the physical coordinate system) which are known. Thus, the complete Jacobian matrix is obtained from the following:

$$[J] = \begin{matrix} \begin{matrix} \boxed{[\partial N]} \\ \swarrow \\ \begin{matrix} N_{1,\xi} & N_{2,\xi} & \cdots & N_{n,\xi} \\ N_{1,\eta} & N_{2,\eta} & \cdots & N_{n,\eta} \\ N_{1,\zeta} & N_{2,\zeta} & \cdots & N_{n,\zeta} \end{matrix} \\ \downarrow \\ \begin{matrix} 3 \times 3 \\ \end{matrix} \end{matrix} \begin{matrix} \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{bmatrix} \\ \begin{matrix} n \times 3 \end{matrix} \end{matrix} \end{matrix} \quad 3.6$$

Note that the sizes of the matrices are also indicated in the above equation. As the matrix of derivatives of shape functions $[\partial N]$ and the matrix containing the element nodal coordinates are both easily available, the Jacobian matrix can be calculated with no difficulty. Once the Jacobian matrix is known, we can get the derivatives of the displacement components with respect to the physical coordinates. For example, from Eq. 3.5, we have

$$\begin{Bmatrix} u_{,x} \\ u_{,y} \\ u_{,z} \end{Bmatrix} = [J]^{-1} \begin{Bmatrix} u_{,\xi} \\ u_{,\eta} \\ u_{,\zeta} \end{Bmatrix}. \quad 3.7$$

We can write the strain-displacement relations as given by Eq. 2.2.2 alternatively as

$$\begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{Bmatrix} u_{,x} \\ u_{,y} \\ u_{,z} \\ v_{,x} \\ v_{,y} \\ v_{,z} \\ w_{,x} \\ w_{,y} \\ w_{,z} \end{Bmatrix}. \quad 3.8$$

$\boxed{[A]}$

Now, from Eq. 3.7, we get

$$\begin{Bmatrix} u_{,x} \\ u_{,y} \\ u_{,z} \\ v_{,x} \\ v_{,y} \\ v_{,z} \\ w_{,x} \\ w_{,y} \\ w_{,z} \end{Bmatrix} = \begin{matrix} \begin{matrix} \boxed{[G]} \\ \swarrow \\ \begin{bmatrix} [J]^{-1} & [0] & [0] \\ [0] & [J]^{-1} & [0] \\ [0] & [0] & [J]^{-1} \end{bmatrix} \\ \downarrow \\ \begin{matrix} 9 \times 9 \end{matrix} \end{matrix} \begin{matrix} \begin{bmatrix} u_{,\xi} \\ u_{,\eta} \\ u_{,\zeta} \\ v_{,\xi} \\ v_{,\eta} \\ v_{,\zeta} \\ w_{,\xi} \\ w_{,\eta} \\ w_{,\zeta} \end{bmatrix} \\ \begin{matrix} 9 \times 9 \end{matrix} \end{matrix} \end{matrix} \quad 3.9$$

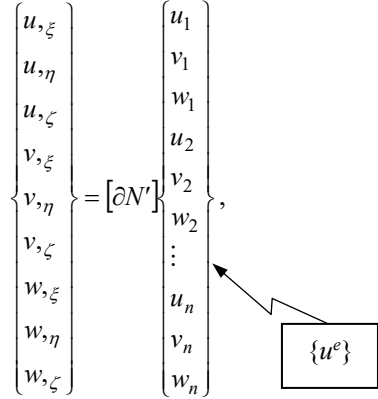
The derivatives of the displacement components with respect to the natural coordinates (ξ, η, ζ) are readily available from Eq. 3.2. Thus, for example, we have

$$u_{,\xi} = \sum_{i=1}^n N_{i,\xi} u_i.$$

The above can be written in terms of the element nodal displacement vector as

$$u_{,\xi} = [N_{1,\xi} \ 0 \ 0 \ N_{2,\xi} \ 0 \ 0 \ \dots \ N_{n,\xi} \ 0 \ 0] \{u^e\},$$

where $\{u^e\} = [u_1 \ v_1 \ w_1 \ u_2 \ v_2 \ w_2 \ \dots \ u_n \ v_n \ w_n]^T$ is the element nodal displacement vector. Similarly, we can write all the derivatives of the displacement vector with respect to the natural coordinates. These derivatives can be jointly written using matrix notation as:

$$\begin{Bmatrix} u_{,\xi} \\ u_{,\eta} \\ u_{,\zeta} \\ v_{,\xi} \\ v_{,\eta} \\ v_{,\zeta} \\ w_{,\xi} \\ w_{,\eta} \\ w_{,\zeta} \end{Bmatrix} = [\partial N'] \begin{Bmatrix} u_1 \\ v_1 \\ w_1 \\ u_2 \\ v_2 \\ w_2 \\ \vdots \\ u_n \\ v_n \\ w_n \end{Bmatrix}, \quad 3.10a$$


where $[\partial N']$ contains the derivatives of the interpolation polynomials N_i and is given by

$$[\partial N'] = \begin{bmatrix} N_{1,\xi} & 0 & 0 & N_{2,\xi} & 0 & 0 & \dots & N_{n,\xi} & 0 & 0 \\ N_{1,\eta} & 0 & 0 & N_{2,\eta} & 0 & 0 & \dots & N_{n,\eta} & 0 & 0 \\ N_{1,\zeta} & 0 & 0 & N_{2,\zeta} & 0 & 0 & \dots & N_{n,\zeta} & 0 & 0 \\ 0 & N_{1,\xi} & 0 & 0 & N_{2,\xi} & 0 & \dots & 0 & N_{n,\xi} & 0 \\ \vdots & & & & & & & & & \\ 0 & 0 & N_{1,\zeta} & 0 & 0 & N_{2,\zeta} & \dots & 0 & 0 & N_{n,\zeta} \end{bmatrix}. \quad 3.10b$$

Combining Eqs. 3.8, 3.9 and 3.10, we get the strain-displacement matrix $[B]$ (which relates the strain field within the element to the element nodal displacement vector by $\{\varepsilon\} = [B]\{u^e\}$) as

$$[B] = [A][G][\partial N']. \quad 3.11$$

In the above, $[A]$ is a matrix containing zeros and ones as given in Eq. 3.8. The elements of the other two matrices, viz. $[G]$ and $[\partial N']$, are nonlinear functions of the natural coordinates ξ , η and ζ .

The element stiffness matrix is obtained as

$$[k^e] = \int_{V_e} B^T DB dV_e = \int_{V_e} B^T DB dx dy dz. \quad 3.12$$

It can be shown that

$$dx dy dz = |J| d\xi d\eta d\zeta, \quad 3.13$$

where $|J|$ is the determinant of the Jacobian matrix which is denoted simply by J and is termed the *Jacobian*. Hence, the element stiffness matrix is given by

$$[k^e] = \int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} B^T DB J d\xi d\eta d\zeta. \quad 3.14$$

It is impossible to evaluate the above integral analytically except in simple cases involving a regular element geometry (e.g., a regular hexahedron). Hence, numerical integration is used for this purpose. The most popular numerical method is the *Gauss quadrature*. See Appendix (given at the end of this) for a brief description of numerical integration.

4 FOUR-NODED QUADRILATERAL ELEMENT FOR PLANE PROBLEMS

In this section, we shall demonstrate the implementation of a plane bilinear isoparametric quadrilateral element. Such an element has four nodes. The parent element and an element in the physical coordinate space are depicted in Fig. 4.1.

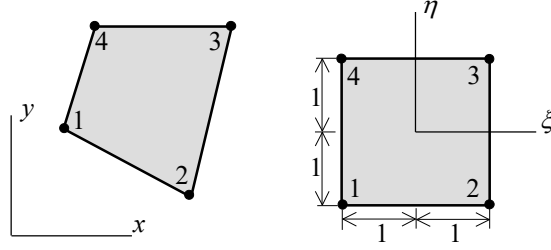


Figure 4.1 Plane bilinear isoparametric element.

The interpolation polynomials used for this element are obtained by multiplying the two linear Lagrangian interpolation polynomials, one in terms of ξ and the other η . Thus, we have

$$\begin{aligned} N_1 &= \frac{1}{4}(1-\xi)(1-\eta) & N_2 &= \frac{1}{4}(1+\xi)(1-\eta) \\ N_3 &= \frac{1}{4}(1+\xi)(1+\eta) & N_4 &= \frac{1}{4}(1-\xi)(1+\eta). \end{aligned} \quad 4.1$$

The interpolation polynomials given above can be concisely written as

$$N_i = \frac{1}{4}(1 + \xi\xi_i)(1 + \eta\eta_i), \quad i = 1 \text{ to } 4, \quad 4.2$$

where ξ_i and η_i are the values of the natural coordinates at node i .

This element is also known as the *bilinear element*, as the interpolation polynomials are the product of two linear polynomials in ξ and η . Note that, it is *not* a linear element (this is due to the presence of the $\xi\eta$ term).

The derivatives of the shape functions with respect to the natural coordinates are easily evaluated. Thus, the matrix $[\partial N]$ of Eq. 3.6 is obtained as

$$[\partial N] = \frac{1}{4} \begin{bmatrix} -(1-\eta) & (1-\eta) & (1+\eta) & -(1+\eta) \\ -(1-\xi) & -(1+\xi) & (1+\xi) & (1-\xi) \end{bmatrix}.$$

The derivatives of the shape functions can also be written concisely by taking the derivatives of Eq. 4.2. Thus, we have

$$N_{i,\xi} = \frac{1}{4}\xi_i(1 + \eta\eta_i), \quad i = 1 \text{ to } 4 \quad 4.3$$

$$N_{i,\eta} = \frac{1}{4}\eta_i(1 + \xi\xi_i), \quad i = 1 \text{ to } 4. \quad 4.4$$

We shall use the expressions given by Eqs. 4.2 to 4.4 in the computer code that we present in this chapter.

The Jacobian matrix is of size 2×2 , and is obtained as

$$[J] = \begin{bmatrix} x_{,\xi} & y_{,\xi} \\ x_{,\eta} & y_{,\eta} \end{bmatrix} = [\partial N] \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}.$$

The inverse of the Jacobian matrix can be written as

$$[J]^{-1} = \frac{1}{|J|} \begin{bmatrix} J_{11} & -J_{12} \\ -J_{21} & J_{22} \end{bmatrix},$$

where J_{ij} , with $i, j = 1, 2$, are the elements of the Jacobian matrix. The quantity $|J|$ is the determinant of Jacobian matrix (known as the *Jacobian* and is denoted by J), and is obtained as

$$J = |J| = J_{11}J_{22} - J_{12}J_{21}.$$

Before we present a simple computer-code using four-noded isoparametric quadrilateral elements for the analysis of plane problems of elasticity.

5 COMPUTER CODE WITH ISOPARAMETRIC QUADRILATERAL ELEMENTS

Computer code that uses 4-noded isoparametric quadrilateral elements for elastostatic problems is given below.

FILE NAME: femQuad4.cpp

```
// Program for FE Analysis of Elastostatic Problems
// Using Isoparametric 4-noded Quadrilateral Element;
// Non-zero displacement d.o.f. can be prescribed;
// Uses dynamic memory allocation

#include <fstream.h>
#include <iostream.h>
#include <iomanip.h>
#include <math.h>
#include "Mat.h"
#include "Vec.h"

void quad4(int, double, double D[4][4]);
void shape_fun(double, double);
void quad4_stress(int, double, double D[4][4], double, double, double[], int);

double eStiff[9][9];
double eLoad[9], xl[5], yl[5];
int nGauss, nEq, semiband;
double B[4][9], Jac[3][3], detJac, stress[4], strain[4];

// Gaussian coordinates and weights
const long double place[5][5] =
{
  {0., 0., 0., 0., 0.},
  {0, 0, -0.577350269189626, -0.774596669241483,
   -0.861136311594053},
  {0, 0, 0.577350269189626, 0, -0.339981043584856},
  {0, 0, 0, 0.774596669241483, 0.339981043584856},
  {0, 0, 0, 0, 0.861136311594053}
};
const long double wgt[5][5] =
{
  {0, 0, 0, 0, 0},
  {0, 2., 1., 0.555555555555555, 0.347854845137454},
  {0, 0, 1., 0.888888888888888, 0.652145154862546},
  {0, 0, 0, 0.555555555555555, 0.652145154862546},
  {0, 0, 0, 0, 0.347854845137454}
};
void main(void)
{
  ifstream fin ("femQuad4.inp");
  ofstream fout("femQuad4.out");
  fout.setf(ios::showpoint);
  fout.setf(ios::floatfield, ios::fixed);
  fout.precision(4);
  int probType, nElems, nNodes, nRestNodes, nLoadedNodes;
  int i, j, k;

  // Input data::
  fin >> nGauss >> probType;
  fin >> nNodes >> nElems;

  dMatrix X(2,nNodes);
  for (i=1; i<=nNodes; ++i)
  {
    fin >> k;
    for (j=1; j<=2; ++j)
      fin >> X(j, k);
  }
  fout << "\n*****\n";
  fout << "\n FINITE ELEMENT ANALYSIS PROGRAM\n";
  fout << "\n*****\n\n";
}
```

```

if (probType == 0)
  fout << "Problem Type: Plane Stress\n";
else
  fout << "Problem Type: Plane Strain\n";
fout << "\nNumber of nodes    = " << nNodes;
fout << "\nNumber of elements = " << nElems;
fout << "\n\nNodal Coordinates";
fout << "\n~~~~~\n";
fout << "\nNode    X-coord.    Y-coord.\n";
fout << "=====";
for (i=1; i<=nNodes; ++i)
{
  fout << "\n" << setw(4) << i << setw(12);
  for (j=1; j<=2; ++j)
    fout << X(j, i) << setw(12);
}
iMatrix elemConn(4, nElems);
fout << "\n\nElement Connectivity";
fout << "\n~~~~~\n";
fout << "\n Elem Nod1  Nod2  Nod3  Nod4  Thick  E\n";
fout << "=====";
dVector thick(nElems), nu(nElems), E(nElems);
for (i=1; i<=nElems; ++i)
{
  fin >> k;
  for (j=1; j<=4; ++j)
    fin >> elemConn(j, k);
  fin >> thick[k] >> E[k] >> nu[k];
}
for (i=1; i<=nElems; ++i)
{
  fout << "\n" << setw(4) << i << setw(5);
  for (j=1; j<=4; ++j)
    fout << elemConn(j, i) << setw(7);
  fout << setw(10) << thick[i] << setw(15) << E[i];
}

dVector U(2*nNodes);
iMatrix destn(2, nNodes);

int dof;
fin >> nRestNodes;
fout << "\n\nNumber of nodes at which";
fout << " displacement is prescribed = " << nRestNodes;
for (i=1; i<=nRestNodes; ++i)
{
  fin >> k;
  for (j=1; j<=2; ++j)
  {
    dof = 2*(k-1) + j;
    fin >> destn(j,k) >> U[dof];
  }
}
nEq = 0;
for (j=1; j<=nNodes; ++j)
  for (i=1; i<=2; ++i)
  {
    if (destn(i, j) == 0)
    {
      nEq++;
      destn(i, j) = nEq;
      continue;
    }
    else
      destn(i, j) = 0;
  }
fout << "\n\nThe Destination Array:";
fout << "\n~~~~~\n";

```



```

fout << "\nNode      X-dof      Y-dof\n";
fout <<"=====";
for (i=1; i<=nNodes; ++i)
{
  fout << "\n" << setw(4) << i << setw(9);
  for (j=1; j<=2; ++j)
    fout << destn(j,i) << setw(9);
}
fout << "\n\nNo. of Degrees of Freedom = " << nEq;
dVector gLoad(nEq);
dVector gDisp(nEq);

double load;
fin >> nLoadedNodes;
for (i=1; i<=nLoadedNodes; ++i)
{
  fin >> k;
  for (j=1; j<=2; ++j)
  {
    dof = destn(j, k);
    fin >> load;
    if (dof != 0)
      gLoad[dof] += load;
  }
}
int node;

// Bandwidth calculation
int small, large, diff;
int n, l, kk[9];
semiband = 0;
for (i=1; i<=nElems; ++i) // scan over each element
{
  small = nEq;
  large = 1;
  for (j=1; j<=4; ++j) // scan over each node
  {
    node = elemConn(j, i);
    for (k=1; k<=2; ++k) // scan over each dof
    {
      dof = destn(k, node);
      if (dof == 0) continue;
      if (dof < small) small = dof;
      if (dof > large) large = dof;
    }
  }
  diff = large - small;// diff gives the semi-bandwidth
  if (diff > semiband) // pick up the largest of diff
    semiband = diff;
}
semiband = semiband + 1;
fout << "\nSemi-band width = " << semiband;
int dof1, tot_dof;
dMatrix gStiff(nEq, semiband);

fout << "\n\nGlobal Load Vector:";
fout << "\n~~~~~\n";
fout << "\n Node          Load-X          Load-Y";
fout <<"\n=====";
int dof2;
for (i=1; i<=nNodes; ++i)
{
  dof1 = destn(1, i);
  dof2 = destn(2, i);
  double zero = 0.0;
  fout << "\n" << setw(4) << i;

```

```

    if (dof1 != 0)
        fout << setw(15) << gLoad[dof1];
    else
        fout << setw(15) << zero;
    if (dof2 != 0)
        fout << setw(15) << gLoad[dof2];
    else
        fout << setw(15) << zero;
}

// Assembly of element matrices
double thickness, D[4][4], pres_disp[9], eLoad_disp[9];
for (n=1; n<=nElems; ++n)
{
    for (i=1; i<=4; i++)
    {
        node = elemConn(i,n);
        xl[i] = X(1,node);
        yl[i] = X(2,node);
    }
    if (probType == 0)
    {
        // plane stress
        D[1][1] = D[2][2] = E[n]/(1 - nu[n]*nu[n]);
        D[1][2] = D[2][1] = D[1][1] * nu[n];
        D[3][3] = D[1][1] * (1 - nu[n])/2.0;
    }
    else
    {
        // plane strain
        double c;
        c = E[n]/(1 + nu[n])/(1 - 2*nu[n]);
        D[1][1] = D[2][2] = c*(1 - nu[n]);
        D[1][2] = D[2][1] = c * nu[n];
        D[3][3] = c * (1 - 2*nu[n])/2.0;
    }
    D[1][3] = D[2][3] = D[3][1] = D[3][2] = 0;

    thickness = thick[n];
    quad4(n, thickness, D);
    dof = 0;
    for (i=1; i<=4; ++i)
    {
        for (j=1; j<=2; ++j)
        {
            dof ++;
            node = elemConn(i, n);
            kk[dof] = destn(j, node);
        }
    }
    tot_dof = 8;
    dof1 = 0;
    for (i=1; i<=4; ++i)
    {
        node = elemConn(i, n);
        for (j=1; j<=2; ++j)
        {
            dof1 ++;
            dof = 2*(node - 1) + j;
            pres_disp[dof1] = U[dof];
        }
    }
    for (i=1; i<=tot_dof; ++i)
    {
        eLoad_disp[i] = 0.;
        for (j=1; j<=tot_dof; ++j)
        {
            eLoad_disp[i] += eStiff[i][j]*pres_disp[j];
        }
    }
}

```

```

    }
  }
  for (i=1; i<=tot_dof; ++i)
  {
    if (kk[i] <= 0)
      continue;
    k = kk[i];
    gLoad[k] += eLoad[i] - eLoad_disp[i];
    for (j=1; j<=tot_dof; ++j)
    {
      if (kk[j] < k)
        continue;
      l = kk[j] - k + 1;
      gStiff(k, l) += eStiff[i][j];
    }
  }
}

fout <<"\n";

gDisp = gStiff^gLoad;   invoking the Gauss elimination by the operator “^”

fout << "\n\nGlobal Dis:
fout << "\n~~~~~";
fout << "\n\n Node        Disp-X        Disp-Y";
fout << "\n=====";
for (j=1; j<=nNodes; ++j)
{
  for (i=1; i<=2; ++i)
  {
    dof = 2*(j-1) + i;
    if (destn(i, j) != 0)
      U[dof] = gDisp[destn(i, j)];
  }
}

for (j=1; j<=nNodes; ++j)
{
  fout << "\n" << setw(4) << j << setw(15);
  for (i=1; i<=2; ++i)
  {
    dof = 2*(j-1) + i;
    fout << U[dof] << setw(15);
  }
}

// Compute stresses at element Gaussian points
fout << "\n\nAverage Stresses and Strains";
fout << " at Element Centroid";
fout << "\n~~~~~";
fout << "\n\n Elem    Sig_x    Sig_y    Sig_xy ";
fout << "\n=====";
double pxi, pet;
for (n=1; n<=nElems; ++n)
{
  thickness = thick[n];
  if (probType == 0)
  {
    // plane stress
    D[1][1] = D[2][2] = E[n]/(1 - nu[n]*nu[n]);
    D[1][2] = D[2][1] = D[1][1] * nu[n];
    D[3][3] = D[1][1] * (1 - nu[n])/2.0;
  }
  else
  {
    // plane strain
    double c;
    c = E[n]/(1 + nu[n])/(1 - 2*nu[n]);

```

```

    D[1][1] = D[2][2] = c*(1 - nu[n]);
    D[1][2] = D[2][1] = c * nu[n];
    D[3][3] = c * (1 - 2*nu[n])/2.0;
}
D[1][3] = D[2][3] = D[3][1] = D[3][2] = 0;
for (int ii=1; ii<=4; ++ii)
{
    node = elemConn(ii, n);
    xl[ii] = X(1, node);
    yl[ii] = X(2, node);
}
pxi = pet = 0; //at place[1][1]-i.e., element-centroid

// Get element displacement vector
int dof, dof1, node;
double u[9];
dof1 = 0;
for (ii=1; ii<=4; ++ii)
{
    node = elemConn(ii, n);
    for (j=1; j<=2; ++j)
    {
        dof1 ++;
        dof = 2*(node - 1) + j;
        u[dof1] = U[dof];
    }
}
quad4_stress(n, thickness, D, pxi, pet, u, 8);
fout << "\n" << setw(4) << n ;
for (i=1; i<=3; i++)
    fout << setw(15) << stress[i];
}
}

void quad4(int elem, double thick, double D[4][4])
{
    int j, k, l, n, na, nb, nrow, ncol;
    double dv, dum, pxi, pet;
    double BtD[9][4];

// Clear load vector and upper triangle of stiffness matrix
for (k=1; k<=8; ++k)
{
    eLoad[k] = 0.;
    for (l=k; l<=8; ++l)
        eStiff [k][l] = 0.;
}
// Start Gauss quadrature loop. Use nGauss by nGauss rule
for (na=1; na<=nGauss; ++na)
{
    pxi = place[na][nGauss];
    for (nb=1; nb<=nGauss; ++nb)
    {
        pet = place[nb][nGauss];
        shape_fun(pxi,pet);
        dv = wgt[na][nGauss]*wgt[nb][nGauss] * thick * detJac;
// Store [B] trans. times [D] in 8 by 3 work array [BtD]
        for (j=1; j<=4; ++j)
        {
            l = 2 * j;
            k = l - 1;
            // Multiplication that gives a nonzero product only
            // considered
            for (n=1; n<=3; ++n)
            {
                BtD[k][n] = B[1][k] * D[1][n] + B[3][k] * D[3][n];
                BtD[l][n] = B[2][l] * D[2][n] + B[3][l] * D[3][n];
            }
        }
    }
}

```

```

    }
    // Loop on rows of [k]
    for (nrow=1; nrow<=8; ++nrow)
    {
        for (ncol=nrow; ncol<=8; ncol++)
        {
            dum = 0.;
            // Loop for product [B]t[D][B].
            // Zeros in [B] not skipped
            for (j=1; j<=3; ++j)
                dum += BtD[nrow][j] * B[j][ncol];
            eStiff[nrow][ncol] += dum * dv;
        }
    }
}

// Fill in lower triangle of element stiffness matrix
// by symmetry
for (k=1; k<=7; ++k)
    for (l=k; l<=8; ++l)
        eStiff[l][k] = eStiff[k][l];
}

void shape_fun(double pxi, double pet)
{
    double N[5], Nxi[5], Net[5], dum;
    int i, l, j, k;
    int xii[5] = {0, -1, 1, 1, -1};
    int eti[5] = {0, -1, -1, 1, 1};
    // Shape functions and their derivatives
    for (i=1; i<=4; ++i)
    {
        double pX = 0.25 * (1 + pxi*xii[i]);
        double pT = 0.25 * (1 + pet*eti[i]);
        N[i] = 4 * pX * pT;
        Nxi[i] = xii[i] * pT;
        Net[i] = eti[i] * pX;
    }
    // Clear array Jac and B
    for (i=1; i<=2; ++i)
        for (j=1; j<=2; ++j)
            Jac[i][j] = 0.;
    for (i=1; i<=3; ++i)
        for (j=1; j<=8; ++j)
            B[i][j] = 0.;

    // Find Jacobian and its determinant.
    for (i=1; i<=4; ++i)
    {
        Jac[1][1] += Nxi[i]*xl[i];
        Jac[1][2] += Nxi[i]*yl[i];
        Jac[2][1] += Net[i]*xl[i];
        Jac[2][2] += Net[i]*yl[i];
    }
    detJac = Jac[1][1]*Jac[2][2] - Jac[1][2]*Jac[2][1];
    // Replace Jac by its inverse.
    dum = Jac[1][1]/detJac;
    Jac[1][1] = Jac[2][2]/detJac;
    Jac[1][2] = - Jac[1][2]/detJac;
    Jac[2][1] = - Jac[2][1]/detJac;
    Jac[2][2] = dum;
    // Form [B] matrix (zero entries already set)
    for (j=1; j<=4; ++j)
    {
        l = 2 * j;
        k = l - 1;
        B[1][k] = Jac[1][1]*Nxi[j] + Jac[1][2]*Net[j];
    }
}

```

```

    B[2][1] = Jac[2][1]*Nxi[j] + Jac[2][2]*Net[j];
    B[3][k] = B[2][1];
    B[3][1] = B[1][k];
}
}

void quad4_stress(int elem, double thick, double D[4][4], double pxi, double pet, double
u[], int ndof)
{
// to get stresses and strains
    int i, j, k;
    double DB[4][9];
// Get B matrix
    shape_fun(pxi, pet);
// [D]*[B]
    for (i=1; i<=3; ++i)
    {
        for (j=1; j<=8; ++j)
        {
            DB[i][j] = 0;
            for (k=1; k<=3; ++k)
                DB[i][j] += D[i][k] * B[k][j];
        }
    }
// Compute stress as DB * u and strains as B * u
    for (i=1; i<=3; ++i)
    {
        stress[i] = 0;
        strain[i] = 0;
        for (j=1; j<=8; ++j)
        {
            stress[i] += DB[i][j] * u[j];
            strain[i] += B[i][j] * u[j];
        }
    }
}
}

```

The reader, it is hoped, will find no difficulty in understanding the main program and functions given above. Comment lines have been inserted abundantly to improve the readability of the program. It may be noted that this program is also an elementary one, and there is ample scope in improving it. The reader is urged to go ahead and improve the program, depending on his/her programming skill. It is possible to include a number of classes such as an element class (a base class from which other element subclasses can be derived), a node class, material property class, shape function class, Gauss point class and a finite element mesh class (which keeps track of all the other classes). There are a good number of references available*, including some text books on the topic of object-oriented finite element programming†.

SAMPLE INPUT/OUTPUT:

The following is the printout of a sample input and the consequent output files. They correspond to the example shown in Fig. 6.1.

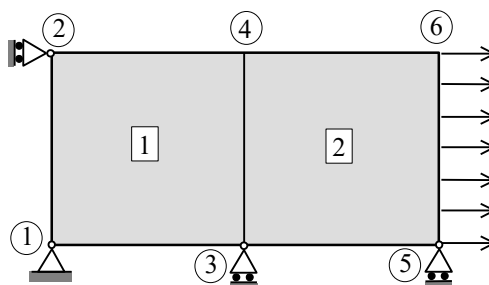


Figure 5.1 The discretisation and element and node numbering of rectangular plate problem.

* The reader may just give OOP and FEM as key words and search to find a large amount of data available on the net. There are plenty of computer codes in C++ too, available for free download!

† See, for example, Mackie, R.I., *Object Oriented Methods and Finite Element Analysis*, Saxe-Coburg Publ., Edinburgh, 2000.

THE INPUT FILE "FEMQUAD4.INP"

```

2      0
6      2

1      0      0
3      100     0
5      200     0
2      0      100
4      100     100
6      200     100

1      1      3      4      2      10      10000      0.25
2      3      5      6      4      10      10000      0.25

2
1      1      0      1      0
2      1      0      0      0

2
5      300     0
6      300     0

```

THE OUTPUT FILE "FEMQUAD4.OUT"

```

*****
FINITE ELEMENT ANALYSIS PROGRAM
*****

```

Problem Type: Plane Stress

Number of nodes = 6
Number of elements = 2

Nodal Coordinates

~~~~~

| Node | X-coord. | Y-coord. |
|------|----------|----------|
| 1    | 0.0000   | 0.0000   |
| 2    | 0.0000   | 100.0000 |
| 3    | 100.0000 | 0.0000   |
| 4    | 100.0000 | 100.0000 |
| 5    | 200.0000 | 0.0000   |
| 6    | 200.0000 | 100.0000 |

Element Connectivity

~~~~~

Elem	Nod1	Nod2	Nod3	Nod4	Thickness	E
1	1	3	4	2	10.0000	10000.0000
2	3	5	6	4	10.0000	10000.0000

Number of nodes at which displacement is prescribed = 2

The Destination Array:

~~~~~

| Node | X-dof | Y-dof |
|------|-------|-------|
| 1    | 0     | 0     |
| 2    | 0     | 1     |
| 3    | 2     | 3     |
| 4    | 4     | 5     |
| 5    | 6     | 7     |
| 6    | 8     | 9     |

No. of Degrees of Freedom = 9

Semi-band width = 8

Global Load Vector:

```

~~~~~
Node Load-X Load-Y
=====
1 0.0000 0.0000
2 0.0000 0.0000
3 0.0000 0.0000
4 0.0000 0.0000
5 300.0000 0.0000
6 300.0000 0.0000

```

Global Displacement Vector

```

~~~~~
Node      Disp-X      Disp-Y
=====
1         0.0000      0.0000
2         0.0000     -0.0015
3         0.0060      0.0000
4         0.0060     -0.0015
5         0.0120      0.0000
6         0.0120     -0.0015
    
```

Average Stresses and Strains at Element Centroid

```

~~~~~
Elem Sig_x Sig_y Sig_xy
=====
1 0.6000 -0.0000 -0.0000
2 0.6000 0.0000 0.0000

```

The data used in the above example is identical to that in Chapter 10 (see Fig. 10.4.1). The results are exact as in the case of triangular elements. The reader is urged to modify the program for plotting given in Section 10.5.1 for the case of four-noded quadrilateral elements. Minor modification is needed to include the fourth node of the element.

As a second example, let us consider the same cantilever beam given in Section 10.5.2. The geometry of the beam, the support conditions and the loading are shown in Fig. 6.2. A simple program to generate the mesh is made. The results from the finite element analysis program are presented in Table 6.1.

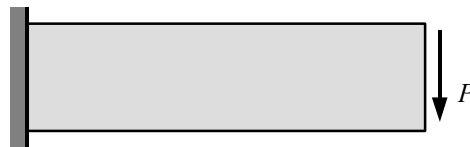


Figure 5.2 Geometry, support and loading of the cantilever beam

The data used for this example problem (in consistent units) are:

- Span of the beam = 1000
- Depth of the beam = 100
- Thickness = 1
- Modulus of elasticity,  $E$  =  $2 \times 10^5$
- Poisson's ratio,  $\nu$  = 0.2
- Load,  $P$  = 100

The strength of materials solution for the beam are obtained as follows:

- (a) The deflection at the free end of the beam,  $\Delta^{\max} = P^3/3EI = 2.0$
- (b) Maximum bending stress,  $\sigma_x^{\max} = M/Z = 60$

Table 5.1 Finite element results of the cantilever beam problem for different discretisations (compare with the strength of materials solutions of  $\Delta^{\max} = 2.0$  and  $\sigma_x^{\max} = 60$ ).

| Discretisation* | nNodes | nElems | $\Delta^{\max}$ | $\sigma_x^{\max}$ |
|-----------------|--------|--------|-----------------|-------------------|
| 10×4            | 55     | 40     | 1.4175          | 30.17             |
| 20×6            | 147    | 120    | 1.8208          | 44.44             |
| 40×10           | 451    | 400    | 1.9614          | 53.36             |

(\* the number of divisions along the span and depth directions)



It can be seen from the above table that the deflection converges faster than the convergence with triangular elements. The stresses shown in the table correspond to the centre of the elements located at the top and bottom ends at the fixed support of the cantilever beam, and hence are not at the extreme fibres. This is the reason why the stresses given in the above table seem to be less accurate than the ones in Table 10.5.2 in comparison with the strength of material solution. It can further be observed, for example, corresponding to the last set of discretisation (i.e., with 400 elements), the stress calculated with the strength of materials approach at the centre location of the element at the top (or bottom) edge close to the support works out as  $60 \times (45/50) \times (987.5/1000) = 53.325$ , which is quite close to the value shown in the table.

#### OPTIMAL STRESS POINTS

We had seen in the last chapter that once the nodal displacements of an element are known, it is possible to obtain the displacements at any point on the element using interpolation. Then the strain at any point on the element is obtained by making use of the strain-displacement relations, and the stresses by using the constitutive relations. As the strain-displacement relations involve (partial) differentiation, and as numerical differentiation always introduces additional errors, the strains, and hence the stresses too, are evaluated with less accuracy.

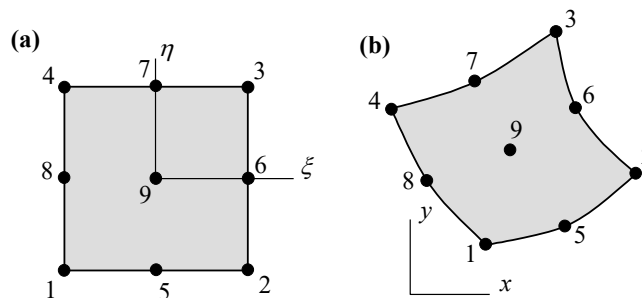
As a general observation, it can be stated that the stresses computed at the nodes are least accurate. On the other hand, in most of the finite element analyses, it is convenient and important to obtain the nodal values of the stresses. The stresses are evaluated with maximum accuracy at a few points on the element called optimal points (known as the *Barlow points*) which are located in the interior of the element. The stress values at the finite element nodes are obtained by extrapolating from the values at the optimal sampling points after averaging the values obtained at each node from the neighbouring elements. For isoparametric elements, these sampling points are located at the Gauss points of one order less. There exists a standard Gauss rule for each type of element. For example, for a plane four-noded quadrilateral element a  $2 \times 2$  Gauss rule is sufficient, and for an 8- or 9-noded quadratic element a  $3 \times 3$  rule is needed. For these elements, the Barlow points are located at the Gauss points corresponding to  $1 \times 1$  rule for the four-noded element, and  $2 \times 2$  rule for the 8- or 9-noded element. This is the reason why the stresses in the above example are computed at the centre (which coincides with the  $1 \times 1$  Gauss point location) of the element.

## 6 ISOPARAMETRIC LAGRANGIAN ELEMENTS FOR PLANE PROBLEMS

Higher order isoparametric Lagrangian elements are available for plane problems. However, Lagrangian quadratic and higher order elements have one or more internal nodes. These nodes are not connected to either neighbouring elements or boundary points. This is the main reason why higher order Lagrangian elements are not as popular as their counterparts—the so-called *serendipity* elements (to be discussed in the next section). Nevertheless, Lagrangian elements have some specific advantages over the serendipity elements: (i) it is easy to obtain the shape functions for any higher order element, and (ii) they are numerically more stable under high distortion of the element.

#### QUADRATIC ELEMENT

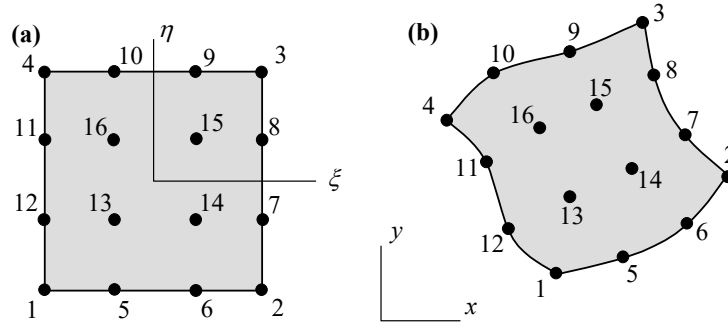
Fig. 6.1 shows the Lagrangian quadratic element which has nine nodes. Truly speaking, this is a bi-quadratic element (i.e., quadratic in two orthogonal directions). The most common numbering of the nodes for this element proceeds with the corner nodes being numbered first, followed by the mid-side nodes and the interior node being numbered last—as node 9—as shown in the figure.



**Figure 6.1** Isoparametric Lagrangian quadratic element with nine nodes: (a) the parent element and (b) the physical element.

#### CUBIC ELEMENT

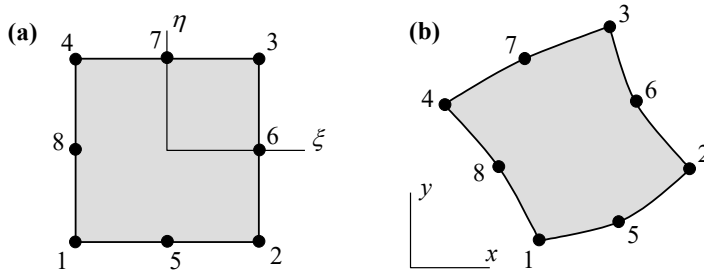
The Lagrangian cubic element has 16 nodes and is shown in Fig. 7.2 in both the physical coordinate system and in the natural coordinate system. There are four internal nodes. See Appendix A for a more elaborate discussion on Lagrangian interpolation polynomials.



**Figure 6.2** Isoparametric Lagrangian cubic element with 16 nodes: (a) the parent element and (b) the physical element.

**7 SERENDIPITY ELEMENTS**

Serendipity elements are a class of finite elements which have very few or no interior nodes. The quadratic element of this family has eight (boundary) nodes as shown in Fig. 8.1. The shape functions of the serendipity elements are derived by an interesting procedure described below.



**Figure 7.1** Isoparametric quadrilateral element of the serendipity family with eight nodes: (a) the parent element and (b) the physical element.

The shape functions of the midside nodes are found first as the product of a quadratic interpolation polynomial along the direction of the edge (i.e.,  $\xi$  or  $\eta$ ) and a linear interpolation in the perpendicular direction. Thus, the interpolation polynomial of node 5 is worked out as

$$N_5 = \frac{(\xi - \xi_1)(\xi - \xi_2)}{(\xi_5 - \xi_1)(\xi_5 - \xi_2)} \frac{(\eta - \eta_7)}{(\eta_5 - \eta_7)} = \frac{(\xi + 1)(\xi - 1)}{(0 + 1)(0 - 1)} \frac{(\eta - 1)}{(-1 - 1)} = \frac{1}{2}(1 - \xi^2)(1 - \eta).$$

Similarly, we obtain the shape functions of the remaining midside nodes as

$$N_6 = \frac{1}{2}(1 + \xi)(1 - \eta^2),$$

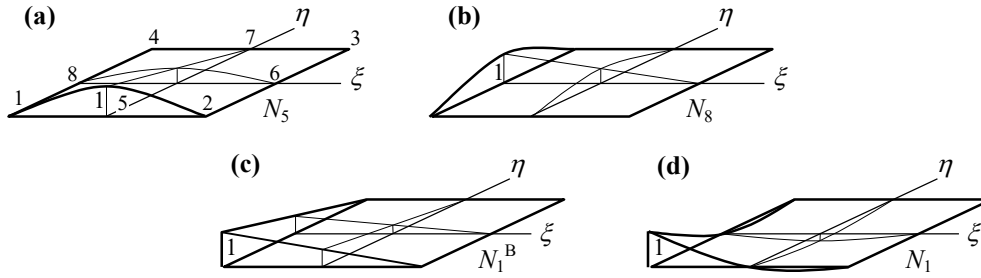
$$N_7 = \frac{1}{2}(1 - \xi^2)(1 + \eta)$$

and

$$N_8 = \frac{1}{2}(1 - \xi)(1 - \eta^2).$$

The shape of  $N_5$  and  $N_8$  are shown in Fig. 8.2a and b. Now, let us see how the shape function for the corner node, say node 1, is obtained. Consider the bi-linear shape function for node 1 (i.e., considering it as a 4-noded element). The shape of this bilinear interpolation function (denoted by  $N_1^B$ ) is shown in Fig. 8.2c. The interpolation function for node 1 is obtained by nailing down the bilinear shape at nodes 5 and 8 (so that they have zero values at these nodes) by subtracting one half each of  $N_5$  and  $N_8$ . That is

$$N_1 = N_1^B - \frac{1}{2}N_5 - \frac{1}{2}N_8.$$



**Figure 7.2** Serendipity shape functions for the midside nodes: **(a)** node 5 and **(b)** node 8; **(c)** the bilinear shape function for node 1 denoted by  $N_1^B$ ; and **(d)** serendipity shape function for the corner node 1 obtained as  $N_1^B - \frac{1}{2} N_5 - \frac{1}{2} N_8$ .

The shape functions for the remaining three corner nodes are obtained likewise. Thus, we have

$$N_2 = N_2^B - \frac{1}{2} N_5 - \frac{1}{2} N_6,$$

$$N_3 = N_3^B - \frac{1}{2} N_6 - \frac{1}{2} N_7$$

and

$$N_4 = N_4^B - \frac{1}{2} N_7 - \frac{1}{2} N_8.$$

The above shape functions put together can be written as

$$N_i = \frac{1}{4}(1 + \xi\xi_i)(1 + \eta\eta_i) - \frac{1}{4}(1 - \xi^2)(1 + \eta\eta_i) - \frac{1}{4}(1 + \xi\xi_i)(1 - \eta^2),$$

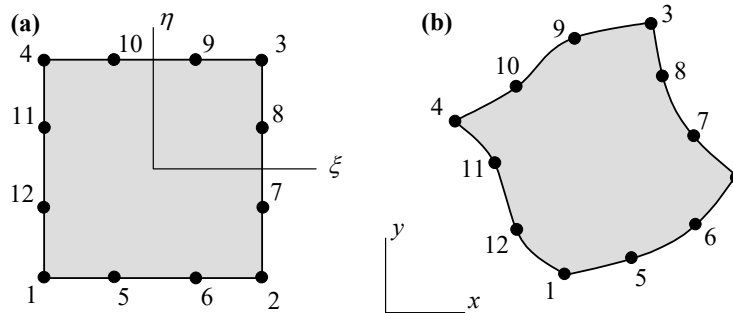
for  $i = 1$  to 4,

$$N_i = \frac{1}{2}(1 - \xi^2)(1 + \eta\eta_i),$$

for  $i = 5$  and 7, and

$$N_i = \frac{1}{2}(1 + \xi\xi_i)(1 - \eta^2),$$

for  $i = 6$  and 8. In these expressions,  $\xi_i$  and  $\eta_i$  are the values of the natural coordinates at node  $i$ .



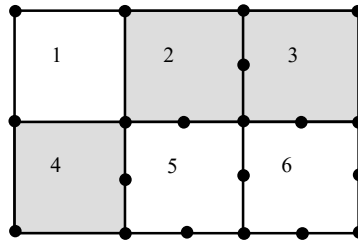
**Figure 7.3** Cubic element of the serendipity family with 12 nodes: **(a)** the parent element and **(b)** the physical element.

The shape functions for higher order (cubic and higher) serendipity elements are derived in the same way. The cubic element of the serendipity family has 12 boundary nodes (and no interior node) and is shown in Fig. 8.3.

## 8 TRANSITION ELEMENTS

While using isoparametric elements of different degree interpolations, it becomes necessary to use transition elements. For example, consider the finite element mesh shown in Fig. 9.1 where the mesh is changing from bilinear element at top left corner (element 1) to a biquadratic elements at bottom right corner (elements 5 and 6). Transition elements with nodes varying from 5 to 7 become necessary to connect the bilinear and quadratic elements as shown in the figure. The interpolation polynomials for such elements are developed following the procedure for the serendipity element explained in the last section.

For example, for the element 4 in Fig. 9.1, there are five nodes. The shape function for the only midside node is first derived as the product of one-dimensional linear polynomial in  $\xi$  and a quadratic polynomial in  $\eta$ . The interpolation polynomials at corners on either side of this node are obtained by subtracting half of this mid-node interpolation function from the bilinear functions of the corner nodes. The remaining two corners will have the usual bilinear shape functions.



**Figure 8.1** Element 1 is a bilinear element; elements 5 and 6 are eight-noded quadratic serendipity elements; and elements 2, 3 and 4 are transition elements with six, six and five nodes respectively.

## Numerical Integration

### A STANDARD GAUSS QUADRATURE

Amongst the various numerical integration schemes, the Gauss quadrature is the most popular one. It makes use of less number of functional evaluations to achieve maximum accuracy. The advent of isoparametric finite element formulation gave an added popularity to Gauss quadrature. It is still the most preferred quadrature rule for both finite and boundary element applications.

#### ONE-DIMENSIONAL INTEGRATION

A one-dimensional integral having arbitrary limits  $a$  and  $b$  is first transformed into one with limits  $-1$  and  $+1$ , by making use of the substitution  $x = \frac{1}{2}(b-a)u + \frac{1}{2}(b+a)$ . Thus, the integral

$$I = \int_a^b f(x)dx$$

gets transformed into

$$I = \int_{-1}^1 F(u)du,$$

where  $F(u)$  includes the Jacobian of the transformation,  $J = dx/du = \frac{1}{2}(b-a)$ , as well.

Gauss quadrature formula is given by

$$I = \int_{-1}^1 F(u)du = \sum_{i=1}^n W_i F(u_i), \quad \text{A.1}$$

where  $n$  is the number of Gaussian points (which equals the number of functional evaluations). The above formula gives the integral as the weighted sum of the function evaluated at  $n$  number of sample points  $u_i$ , which are called the *abscissae*. The quantities  $W_i$  are known as the *weights*. Table A.1 provides the values of  $u_i$  and  $W_i$  for several values of  $n$ . As many digits as the computer allows should be used in programming Gauss quadrature to avoid round off errors. The derivation of the formula given in Eq. A.1 with detailed description and extensive tables containing the values of  $u_i$  and  $W_i$  for different values of  $n$  are available elsewhere\*.

With  $n$  number of Gaussian points a polynomial of degree  $2n-1$  is integrated *exactly*. If  $F(u)$  is not a polynomial, Gauss quadrature is not exact. However, accuracy improves with more number of Gaussian points. The abscissae values are known to be the roots of Gauss-Legendre polynomial and they are located symmetrically with respect to the centre of the interval.

\* See, for example, Stroud, A.H, and Secrest, D., *Gaussian Quadrature Formulas*, Prentice-Hall, Englewood Cliffs, NJ, 1966, and Kopal, Z, *Numerical Analysis*, John Wiley & Sons Inc., New York, 1961.

**Table A.1** Abscissae and weights for standard Gauss quadrature.

| Number of Gauss points | Abscissa, $u_i$                                                                                                          | Weight, $W_i$                                                                            |
|------------------------|--------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| 1                      | 0.0                                                                                                                      | 2.0                                                                                      |
| 2                      | $\pm 0.57735\ 02691\ 89626$                                                                                              | 1.0                                                                                      |
| 3                      | $\pm 0.77459\ 66692\ 41483$<br>0.                                                                                        | 0.55555 55555 55556<br>0.88888 88888 88889                                               |
| 4                      | $\pm 0.86113\ 63115\ 94053$<br>$\pm 0.33998\ 10435\ 84856$                                                               | 0.34785 48451 37454<br>0.65214 51548 62546                                               |
| 6                      | $\pm 0.93246\ 95142\ 03152$<br>$\pm 0.66120\ 93864\ 66265$<br>$\pm 0.23861\ 91860\ 83197$                                | 0.17132 44923 79170<br>0.36076 15730 48139<br>0.46791 39345 72691                        |
| 8                      | $\pm 0.96028\ 98564\ 97536$<br>$\pm 0.79666\ 64774\ 13627$<br>$\pm 0.52553\ 24099\ 16329$<br>$\pm 0.18343\ 46424\ 95650$ | 0.10122 85362 90376<br>0.22238 10344 53374<br>0.31370 66458 77887<br>0.36268 37833 78362 |

## TWO- AND THREE-DIMENSIONAL INTEGRATIONS

By successive application of one-dimensional Gauss rule, we get the rules for two- and three-dimensional integrations as

$$I_2 = \int_{-1}^1 \int_{-1}^1 F(u, v) du dv = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} W_i W_j F(u_i, v_j) \quad \text{A.2}$$

and

$$I_3 = \int_{-1}^1 \int_{-1}^1 \int_{-1}^1 F(u, v, w) du dv dw = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \sum_{k=1}^{n_3} W_i W_j W_k F(u_i, v_j, w_k). \quad \text{A.3}$$

In Eqs. A.2 and A.3, the quantities  $n_1$ ,  $n_2$  and  $n_3$  are the number of Gaussian integration points along  $u$ ,  $v$  and  $w$  directions respectively. These numbers could be, in general, different.

The three Gauss formulae as given by Eqs. A.1 to A.3 can be easily implemented on a computer. The abscissae and weights are usually stored in two-dimensional arrays. A nested loop is used to evaluate the integrals.

**SUMMARY**

Let us briefly examine the important points that we have seen in this chapter.

- The advantages of the isoparametric formulation were first discussed.
- Sub, super and isoparametric formulations were defined.
- The isoparametric formulation was considered at length. We saw that the transformation is effected by the Jacobian matrix.
- Numerical integration becomes necessary to obtain the resulting stiffness matrix and the load vector.
- A simple computer code for the analysis of plane problems of elasticity using four-noded isoparametric bilinear element was presented. The code is elementary with ample scope for further refinements and sophistications.
- Higher order Lagrangian and serendipity elements were discussed. The use of transition elements was indicated.

We shall see some of the advanced topics related to the finite element analysis in the next chapter.

