Finite Element Concepts and Implementation

Written by:

Daniel V. Swenson

X. Jack Xin

Liang-Wu Cai

Kansas State University
Mechanical and Nuclear Engineering Department
Manhattan, KS 66506
1.785.532.5610

©1996–2006, D. V. Swenson, X. J. Xin & L.-W. Cai

Preface

Approach

The goal of these notes is to provide a complete presentation of the Galerkin weighted residual approach to the finite element method. Doing this should result in a thorough understanding of the topics discussed, while not providing a completely general discussion of all methods. Many excellent books can be used to fill in gaps in the presentation. Some recommended texts include:

Hughes, T. J. R., *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis,* Prentice-Hall, Inc., 1987.

Bathe, K.-J., *Finite Element Procedures,* Prentice-Hall, Inc., 1996.

Cook, R. D., Malkus, D. S., Plesha, M. E., and Witt, R. J., *Concepts and Applications of Finite Element Analysis,* 4th Edition, John Wiley & Sons, 2002.

In these notes, our approach is similar to Hughes (1987), but presented at a more elementary level. After an introduction to the method, we implement two-dimensional elasticity using the eight-noded quadrilateral element. This introduces the topics of shape functions, numerical integration, assembly of the global equations, solution, and post-processing.

We believe that understanding comes from implementing all fundamental concepts in code. Therefore, we provide pseudo-coding of all functions, but expect the student to implement details. In our classes, we provide a banded solver for use by students.

When the student completes the course they should have a thorough understanding of a finite element program. They should also be in a position to learn additional concepts of the method from other texts.

In the appendices, we discuss topics that may be of selective interest to students.

Distribution

These notes are provided free of charge to any reader. Students at Kansas State University can little afford to pay the high costs of most textbooks. Any reader may use the notes with

two stipulations:

- Please reference these notes whenever they are used. The reference should be, *Finite Element Concepts and Implementation*, Daniel Swenson, X. J. Xin, Liang-Wu Cai, 2005, available at `http://www.engg.ksu.edu/~swenson/`.
- The notes must be freely distributed by you if you distribute them further.

Contents

Chapter 1

A Simple One-Dimensional Example of Discretization

1.1 Objective

Our objective is to introduce one view of the finite element method using discrete elements. In this view, the finite element method consists of the following steps:

- Dividing the problem into elements.
- Obtaining a stiffness matrix for each element.
- Assembling each element stiffness matrix into a global matrix.
- Solving the global matrix equations.

This approach is used for frame analysis by civil engineers. It also is the basis of the first papers on finite element analysis of a continuum (Clough, 1960). In later chapters we will develop a more mathematical approach, where the finite element method will be obtained using weighted residuals.

In this chapter, we will first illustrate the solution of a spring system by writing equations of equilibrium. We will then write the stiffness matrix for a spring element and assemble these into a global matrix. Finally, the solution using Gauss elimination simulates a computer solution.

1.2 One-Dimensional Spring System

The example problem we will use is shown in Figure 1–1. This figure shows a system of connected springs with a load. What we will call "nodes" (points at which elements are connected) are indicated using circles and the "elements" (springs) are indicated using squares. The displacement of each node is given by u_i, where i is the node number.

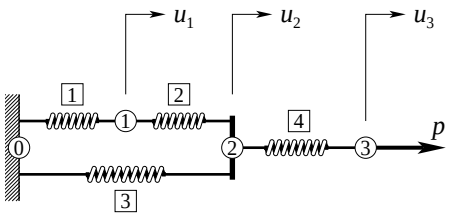


Fig. 1–1 Spring system example problem.

Question 1: *How dose the numbering of nodes and elements affect the solution?*

1.3 Equations Using Nodal Equilibrium

Recall that the force in a spring is related to the stretching:

$$f = k\delta \tag{1–1}$$

where k is the spring stiffness and δ is the spring stretching. The free body diagram at node 1 is shown in Figure 1–2. The equation of equilibrium is:

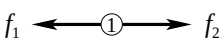


Fig. 1–2 Free body diagram for node 1.

$$\sum F_X = 0$$
$$-f_1 + f_2 = 0$$

or

$$-k_1(u_1 - u_0) + k_2(u_2 - u_1) = 0 \tag{1–2}$$

Similarly at node 2 (Figure 1–3):

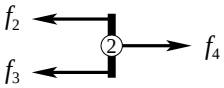


Fig. 1–3 Free body diagram for node 2.

$$\sum F_X = 0$$
$$-f_2 - f_3 + f_4 = 0$$

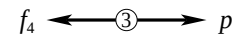


Fig. 1–4 Free body diagram for node 3.

$$-k_2(u_2 - u_1) - k_3(u_2 - u_0) + k_4(u_3 - u_2) = 0 \tag{1–3}$$

And at node 3 (Figure 1–4):

$$\sum F_X = 0$$
$$-f_4 + p = 0$$

$$-k_4(u_3 - u_2) + p = 0 \tag{1–4}$$

Now assemble eqns. (1–2), (1–3) and (1–4) into matrix form:

$$\begin{aligned} (k_1 + k_2)u_2 - (k_2)u_2 &= k_1u_0 \\ -(k_2)u_2 + (k_2 + k_3 + k_4)u_2 - (k_4)u_3 &= k_3u_0 \\ -(k_4)u_2 + (k_4)u_3 &= p \end{aligned}$$

or:

$$\begin{bmatrix} (k_1 + k_2) & -(k_2) & 0 \\ -(k_2) & (k_2 + k_3 + k_4) & -(k_4) \\ 0 & -(k_4) & (k_4) \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix} = \begin{Bmatrix} k_1u_0 \\ k_3u_0 \\ p \end{Bmatrix} \tag{1–5}$$

or:

$$[\boldsymbol{K}]\{\boldsymbol{u}\} = \{\boldsymbol{f}\} \tag{1–6}$$

In finite element terminology, $[\boldsymbol{K}]$ is the stiffness matrix, $\{\boldsymbol{u}\}$ is the displacement vector and $\{\boldsymbol{f}\}$ is the load vector. Note: Bold upper case is a matrix, bold lower case is a vector.

1.4 Equations Using Spring Element

Equation (1–5) was obtained using equilibrium at each node. We will now show how the same result can be obtained using the concept of a spring element with an associated element stiffness matrix. A typical spring is shown in Figure 1–5. It has nodes i and j, associated displacements (or degrees of freedom) u_i and u_j, and stiffness k.

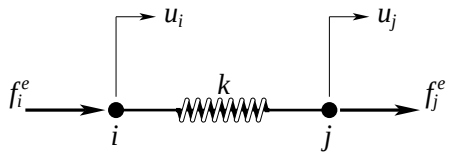


Fig. 1–5 Typical spring element.

The stiffness matrix may be thought of as the matrix that relates forces on an element to displacements of the element, *i.e.*,

$$\begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \begin{Bmatrix} u_i \\ u_j \end{Bmatrix} = \begin{Bmatrix} f_i^e \\ f_j^e \end{Bmatrix} \tag{1–7}$$

The coefficients of the stiffness matrix are the forces at the nodes that correspond to a unit displacement at each node, holding all other degrees of freedom (DOF's) fixed. For example, giving node i a unit displacement, while holding node j fixed results in the forces:

$$\begin{aligned} f_i^e &= ku \\ f_j^e &= -ku \end{aligned} \tag{1–8}$$

This gives the first column of coefficients in the element stiffness matrix:

$$\begin{bmatrix} k & ? \\ -k & ? \end{bmatrix} \begin{Bmatrix} u_i \\ u_j \end{Bmatrix} = \begin{Bmatrix} f_i^e \\ f_j^e \end{Bmatrix} \tag{1–9}$$

Similarly, giving node j a unit displacement, while holding node i fixed results in the forces:

$$\begin{aligned} f_i^e &= -ku \\ f_j^e &= ku \end{aligned} \tag{1–10}$$

This completes the element stiffness matrix.

$$\begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \begin{Bmatrix} u_i \\ u_j \end{Bmatrix} = \begin{Bmatrix} f_i^e \\ f_j^e \end{Bmatrix} \tag{1–11}$$

Note that the element matrix is symmetric.

To assemble the global stiffness matrix using the element approach, we must define a relationship between the element node numbering scheme and the global numbering scheme. We will do this for element 2 first, since this element is not associated with a boundary condition. Referring to Figure 1–1, we see that for element 2, node i corresponds to global node 1, and that node j corresponds to global node number 2. Letting $i = 1$ and

$j = 2$, we can take the associated element stiffness coefficients and place them in the global matrix.

$$\begin{bmatrix} k_2 & -k_2 & 0 \\ -k_2 & k_2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix} = \begin{Bmatrix} f_1^2 \\ f_2^2 \\ 0 \end{Bmatrix} \tag{1–12}$$

We now return to element 1. This element is special because element node i corresponds to global node 0 (Figure 1–1) which is a boundary condition node and has a known displacement. There are two consequences:

1. We do not want to include node 0 in the solution, since we already know its displacement, and
2. We need to accommodate the general condition where the displacement of global node 0 is any fixed value, not just zero.

Following eqn. (1–11), the general expression for the element 1 stiffness matrix and nodal forces can be written as:

$$\begin{bmatrix} k_1 & -k_1 \\ -k_1 & k1 \end{bmatrix} \begin{Bmatrix} u_0 \\ u_1 \end{Bmatrix} = \begin{Bmatrix} f_0^1 \\ f_1^1 \end{Bmatrix} \tag{1–13}$$

However, recognizing that u_0 is known, we can rearrange this to be:

$$\begin{bmatrix} 1 & 0 \\ 0 & k_1 \end{bmatrix} \begin{Bmatrix} u_0 \\ u_1 \end{Bmatrix} = \begin{Bmatrix} u_0 \\ f_1^1 + k_1 u_0 \end{Bmatrix} \tag{1–14}$$

Since we know u_0, we only need the row corresponding to u_1. Therefore, only the j local node contributes to the global matrix

$$\begin{bmatrix} k_1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix} = \begin{Bmatrix} f_1^1 + k_1 u_0 \\ 0 \\ 0 \end{Bmatrix} \tag{1–15}$$

Proceeding in the same manner, for element 3 the contributions to the global matrix will be:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & k_3 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ f_2^3 + k_3 u_0 \\ 0 \end{Bmatrix} \tag{1–16}$$

And finally, for element 4 the contributions to the global matrix will be:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & k_4 & -k_4 \\ 0 & -k_4 & k_4 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ f_2^4 \\ f_3^4 \end{Bmatrix} \tag{1–17}$$

To obtain the complete global stiffness matrix, we add eqns. (1–12), (1–15), (1–16) and (1–17) giving:

$$\begin{bmatrix} (k_1+k_2) & -(k_2) & 0 \\ -(k_2) & (k_2+k_3+k_4) & -(k_4) \\ 0 & -(k_4) & (k_4) \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix} = \begin{Bmatrix} f_1^1 + f_1^2 + k_1 u_0 \\ f_2^1 + f_2^3 + f_2^4 + k_3 u_0 \\ f_3^4 \end{Bmatrix} \tag{1–18}$$

Now, we can simplify the right hand side by recognizing that the nodal force terms are simply a statement of static equilibrium at each node and sum to zero. This gives:

$$\begin{bmatrix} (k_1+k_2) & -(k_2) & 0 \\ -(k_2) & (k_2+k_3+k_4) & -(k_4) \\ 0 & -(k_4) & (k_4) \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix} = \begin{Bmatrix} k_1 u_0 \\ k_3 u_0 \\ p \end{Bmatrix} \tag{1–19}$$

Clearly, this is the same as obtained using nodal equilibrium, eqn. (1–5). We have thus demonstrated how an element approach can be used to obtain the global stiffness matrix. The advantage of the element approach is that assembling the matrix can be easily automated for computer use.

Question 2: *The stiffness matrix in finite element method usually has a lot of zero elements. How to judge, from the physical structure of the system being studied, whether an element of* $[\boldsymbol{K}]$ *is zero or not?*

Question 3: *The assembly involves putting element stiffnesses together to form the global stiffness matrix. Illustrate graphically how to decide the position in the global matrix.*

1.5 Solution Using Gauss Elimination

As a simple reminder and to simulate the actual computer solution, we will solve eqn. (1–5) using Gauss elimination. Assume that $k_1 = 500\,\text{lb/in}$, $k_2 = 250\,\text{lb/in}$, $k_3 = 2000\,\text{lb/in}$, $k_4 = 1000\,\text{lb/in}$, and $p = 1000\,\text{lb}$, then the global matrix becomes:

$$\begin{bmatrix} 750 & -250 & 0 \\ -250 & 3250 & -1000 \\ 0 & -1000 & 1000 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 1000 \end{Bmatrix} \tag{1–20}$$

First, normalize row 1:

$$\begin{bmatrix} 1 & -0.3333 & 0 \\ -250 & 3250 & -1000 \\ 0 & -1000 & 1000 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 1000 \end{Bmatrix} \tag{1–21}$$

Next, zero the remaining coefficients in column 1 by multiplying row 1 by the appropriate factor and subtracting row 1 from each of the remaining rows.

$$\begin{bmatrix} 1 & -0.3333 & 0 \\ 0 & 3166.7 & -1000 \\ 0 & -1000 & 1000 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 1000 \end{Bmatrix} \tag{1–22}$$

Normalize row 2 and repeat:

$$\begin{bmatrix} 1 & -0.3333 & 0 \\ 0 & 1 & -0.3158 \\ 0 & 0 & 684.2 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 1000 \end{Bmatrix} \tag{1–23}$$

Finally backsubstitute to obtain the final answer:

$$\{\boldsymbol{u}\} = \begin{Bmatrix} 0.1538 \\ 0.4616 \\ 1.4615 \end{Bmatrix} \tag{1–24}$$

Question 4 *: In what situations, apart from singular matrix, will the Gauss elimination method become ill-behaved? How to overcome the problem?*

Question 5: *If a truss member, which has an area A, length L and Young's modulus E, is made equuivalent to a spring, what should be the value of k for the srpring?*

1.6 Check Solution

All analyses should be viewed with skepticism until confidence in the solution can be justified. Highest confidence can be obtained by verifying the results using an independent method; just repeating the same calculation carefully is *not* an independent approach. We will check our solution by calculating the forces in the springs. For spring 1, the force is:

$$\begin{aligned} f_1 &= k_1(u_1) \\ &= 250(0.1538) \\ &= 77\,\text{lb} \end{aligned} \tag{1–25}$$

The other forces are calculated in a similar manner and plotted in Figure 1–6.

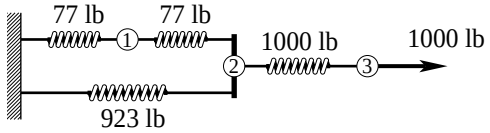


Fig. 1–6 Spring forces.

As can be seen, the results satisfy equilibrium.

1.7 Hints to Questions

Hint 1: The solutions will be the same at the same physical locations.

Hint 2: Look at the connectivity between two nodes.

Hint 3: Use the relation between local numbering and global numbering.

Hint 4: When the diagonal term is not dominating. Use row and column pivoting.

Hint 5: $k = AE/L$

Chapter 2

Solutions Using the Finite Difference Method

2.1 Objective

To remind the student of the finite difference method in order to provide a contrast to the finite element method of solution of differential equations.

We first solve a model problem (an elastic rod with body and end loads) that we will use to demonstrate the different methods. We will then derive the finite difference approximation to the differential equation and obtain an approximate solution to the problem.

2.2 Differential Equation for Elastic Rod

Let us first derive a differential equation that will provide a model problem with which we can demonstrate concepts. We will use an elastic rod (Figure 2–1) with constant area A, fixed at one end and loaded with a body load per unit length, $f^B(x)$, and an end load, P. The Young's modulus of the rod is E.

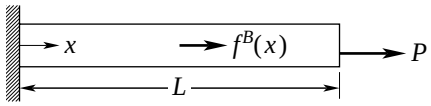


Fig. 2–1 Schematic of elastic rod.

We focus on a differential element of the rod of length Δx at location x, with the stresses and body loads shown in Figure 2–2.

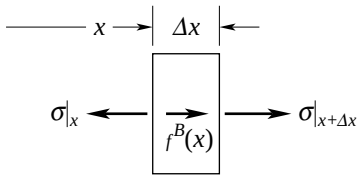


Fig. 2–2 Differential element in rod.

For static equilibrium, the summation of the forces is zero:

$$\sum F_x = 0$$

or:

$$-(A\sigma)|_x + f^B(x)\Delta x + (A\sigma)|_{x+\Delta x} = 0 \tag{2–1}$$

Rearranging and assuming constant area:

$$A\frac{(\sigma|_{x+\Delta x} - \sigma_x)}{\delta x} + f^B(x) = 0 \tag{2–2}$$

Taking the limit as $\Delta x \to 0$,

$$A\frac{d\sigma}{dx} + f^B(x) = 0 \tag{2–3}$$

For an elastic rod with an axial load and no lateral constraints, the stress is related to the strain by,

$$\sigma = E\varepsilon \tag{2–4}$$

where E is Young's modulus. The strain is related to the displacements by:

$$\varepsilon = \frac{du}{dx} \tag{2–5}$$

Substituting eqns. (2–5) and (2–4) into eqn. (2–3) gives,

$$A\frac{d}{dx}\left(E\frac{du}{dx}\right) + f^B(x) = 0 \tag{2–6}$$

Assuming Young's modulus is constant with $f^B(x) = b$,

$$AE\frac{d^2u}{dx^2} + b = 0 \qquad \text{for } 0 \le x \le L \tag{2–7}$$

Note, we have assumed constant area and modulus to obtain a simple differential equation. As we will see, variable values are easy to include in our future finite formulation.

Our particular problem is now described by the differential equation (2–7) with boundary conditions:

$$u|_{x=0} = 0 \tag{2–8}$$

an essential boundary condition and

$$AE \left.\frac{du}{dx}\right|_{x=L} = P \tag{2–9}$$

a natural boundary condition.

Solving by integrating:

$$\begin{aligned} AE\frac{d^2u}{dx^2} &= -b \\ AE\frac{du}{dx} &= -bx + C_1 \\ AEu &= -\frac{1}{2}bx^2 + C_1x + C_2 \end{aligned} \tag{2–10}$$

Use boundary conditions to solve for the constants, at $x = 0$,

$$\begin{aligned} u|_{x=0} &= 0 \\ 0 &= 0 + 0 + C_2 \\ C_2 &= 0 \end{aligned} \tag{2–11}$$

and at $x = L$,

$$\begin{aligned} AE \left.\frac{du}{dx}\right|_{x=L} &= P \\ P &= -bL + C_1 \\ C_1 &= P + bL \end{aligned}$$

Giving:

$$u = \frac{1}{AE}\left[-\frac{1}{2}bx^2 + (P + bL)x\right] \tag{2–12}$$

Evaluating at $x = L/2$:

$$u|_{x=L/2} = \frac{1}{AE}\left(b\frac{3L^2}{8} + P\frac{L}{2}\right) \tag{2–13}$$

The solution is plotted in Figure 2–3 for values of $A = E = L = P = b = 1$. Note that the displacement is linear when the only load is the end load P. The body load adds a quadratic displacement to the solution.

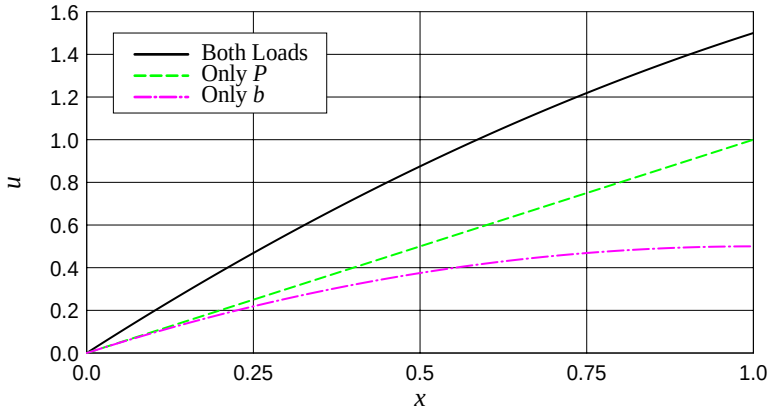


Fig. 2–3 Analytic solutions for displacement of rod.

2.3 The Finite Difference Method

We derive the finite difference approximation using a Taylor series expansion. The displacement near any point x is given by:

$$u(x+\Delta x) = u(x) + \Delta x \left.\frac{du}{dx}\right|_x + \left.\frac{\Delta x^2}{2}\frac{d^2u}{dx^2}\right|_x + HOT \tag{2–14}$$

where $x+\Delta x$ is a point near x, and HOT represents the higher order terms. Alternatively, using the central difference theorem with $x+\theta$ being a position between x and $x+\Delta x$:

$$u(x+\Delta x) = u(x) + \Delta x \left.\frac{du}{dx}\right|_x + \left.\frac{\Delta x^2}{2}\frac{d^2u}{dx^2}\right|_{x+\theta} \tag{2–15}$$

If we use a mesh as shown below in Figure 2–4, we can then do a forward difference approximation:

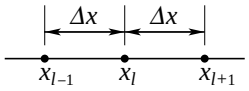


Fig. 2–4 Finite difference mesh.

$$u_{l+1} = u_l + \Delta x \left.\frac{du}{dx}\right|_l + \left.\frac{\Delta x^2}{2}\frac{d^2u}{dx^2}\right|_{l+\theta} \tag{2–16}$$

where u_l denotes u evaluated at $x = x_l$. Rearranging:

$$\left.\frac{du}{dx}\right|_l = \frac{u_{l+1}-u_l}{\Delta x} - \left.\frac{\Delta x}{2}\frac{d^2u}{dx^2}\right|_{l+\theta} \tag{2–17}$$

The last term in eqn. (2–17) is the error term. It has a coefficient of Δx, so we say the error is of order Δx:

$$\text{Error} = O(\Delta x) \tag{2–18}$$

We don't know the error exactly, but we can bound it:

$$|\,\text{Error}\,| \leq \frac{\Delta x}{2} \max_{x_l, x_{l+1}} \left| \frac{d^2u}{dx^2} \right| \tag{2–19}$$

We can also do a similar backward difference approximation.

The accuracy can be improved using a central difference approximation. Again start with a Taylor's expansion:

$$u_{l+1} = u_l + \Delta x \left.\frac{du}{dx}\right|_l + \left.\frac{\Delta x^2}{2}\frac{d^2u}{dx^2}\right|_l + \left.\frac{\Delta x^3}{6}\frac{d^3u}{dx^3}\right|_{l+\theta_1} \tag{2–20}$$

$$u_{l-1} = u_l - \Delta x \left.\frac{du}{dx}\right|_l + \left.\frac{\Delta x^2}{2}\frac{d^2u}{dx^2}\right|_l - \left.\frac{\Delta x^3}{6}\frac{d^3u}{dx^3}\right|_{l-\theta_2} \tag{2–21}$$

Subtracting eqn. (2–21) from eqn. (2–20) gives:

$$u_{l+1} - u_{l-1} = 2\Delta x \left.\frac{du}{dx}\right|_l + \frac{\Delta x^3}{6}\left(\left.\frac{d^3u}{dx^3}\right|_{l+\theta_1} + \left.\frac{d^3u}{dx^3}\right|_{l-\theta_2} \right)$$

or:

$$\left.\frac{du}{dx}\right|_l = \frac{u_{l+1} - u_{l-1}}{2\Delta x} + \frac{\Delta x^2}{12}\left(\left.\frac{d^3u}{dx^3}\right|_{l+\theta_1} + \left.\frac{d^3u}{dx^3}\right|_{l-\theta_2} \right) \tag{2–22}$$

The error term is now $O(\Delta x^2)$, so that as we make Δx smaller, the error will become smaller faster for the central difference approximation than for either the forward or backward difference approximations. We can also obtain an expression for the second derivative:

$$u_{l+1} = u_l + \Delta x \left.\frac{du}{dx}\right|_l + \left.\frac{\Delta x^2}{2}\frac{d^2u}{dx^2}\right|_l + \left.\frac{\Delta x^3}{6}\frac{d^3u}{dx^3}\right|_l + \left.\frac{\Delta x^4}{24}\frac{d^4u}{dx^4}\right|_{l+\theta_3} \tag{2–23}$$

$$u_{l-1} = u_l - \Delta x \left.\frac{du}{dx}\right|_l + \left.\frac{\Delta x^2}{2}\frac{d^2u}{dx^2}\right|_l - \left.\frac{\Delta x^3}{6}\frac{d^3u}{dx^3}\right|_l + \left.\frac{\Delta x^4}{24}\frac{d^4u}{dx^4}\right|_{l-\theta_4} \tag{2–24}$$

Adding equation eqns. (2–23) and (2–24) gives:

$$u_{l+1} + u_{l-1} = 2u_l + \Delta x^2 \left.\frac{d^2u}{dx^2}\right|_l + \frac{\Delta x^4}{24}\left(\left.\frac{d^4u}{dx^4}\right|_{l+\theta_3} + \left.\frac{d^4u}{dx^4}\right|_{l-\theta_4} \right)$$

or:

$$\left.\frac{du^2}{dx^2}\right|_l = \frac{u_{l-1} - 2u_l + u_{l+1}}{\Delta x^2} - \frac{\Delta x^2}{24}\left(\left.\frac{d^4u}{dx^4}\right|_{l+\theta_3} + \left.\frac{d^4u}{dx^4}\right|_{l-\theta_4}\right) \tag{2–25}$$

Again, the error term is $O(\Delta x^2)$. Substituting eqn. (2–25) into eqn. (2–7) and dropping the higher order terms gives the finite difference approximation for the rod:

$$AE\left(\frac{u_{l-1} - 2u_l + u_{l+1}}{\Delta x^2}\right) + f^B(x) = 0 \tag{2–26}$$

Let us now use this approximation to solve our problem. We will use five nodes, as shown in Figure 2–5.

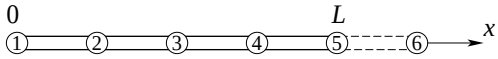


Fig. 2–5 Discretization of rod.

The finite difference statements of the boundary conditions are:

$$u_1 = 0 \tag{2–27}$$

$$AE\left(\frac{u_6 - u_4}{2\Delta x}\right) = P \tag{2–28}$$

where node 6 is a fictitious node used to impose the force boundary condition, based on eqn. (2–22).

We now develop a global matrix for the solution of the problem. Using the finite difference approximation of eqn. (2–26) for node 2 gives:

$$AE\left(\frac{u_1 - 2u_2 + u_3}{\Delta x^2}\right) + f_2^B = 0$$

but $u_1 = 0$, so,

$$\frac{AE}{\Delta x^2}(2u_2 - u_3) = f_2^B \tag{2–29}$$

For node 3:

$$AE\left(\frac{u_2 - 2u_3 + u_4}{\Delta x^2}\right) + f_3^B = 0$$

$$\frac{AE}{\Delta x^2}(-u_2 + 2u_3 - u_4) = f_3^B \tag{2–30}$$

Similarly for node 4:

$$AE\left(\frac{u_3 - 2u_4 + u_5}{\Delta x^2}\right) + f_4^B = 0$$
$$\frac{AE}{\Delta x^2}(-u_3 + 2u_4 - u_5) = f_4^B \tag{2–31}$$

At node 5 we have:

$$AE\left(\frac{u_4 - 2u_5 + u_6}{\Delta x^2}\right) + f_5^B = 0$$

Combining with the boundary condition, eqn. (2–28), gives,

$$\frac{AE}{\Delta x^2}(2u_5 - 2u_4) = f_5^B + \frac{2P}{\Delta x} \tag{2–32}$$

In matrix form, eqns. (2–29) through (2–32) become:

$$\frac{AE}{\Delta x^2}\begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -2 & 2 \end{bmatrix}\begin{Bmatrix} u_2 \\ u_3 \\ u_4 \\ u_5 \end{Bmatrix} = \begin{Bmatrix} f_2^B \\ f_3^B \\ f_4^B \\ f_5^B + \dfrac{2P}{\Delta x} \end{Bmatrix} \tag{2–33}$$

or, assuming $A = E = L = P = f^B(x) = 1$:

$$\frac{1}{(0.25)^2}\begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -2 & 2 \end{bmatrix}\begin{Bmatrix} u_2 \\ u_3 \\ u_4 \\ u_5 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 1 \\ 1 \\ 9 \end{Bmatrix} \tag{2–34}$$

Solving eqn. (2–34) gives,

$$\{\boldsymbol{u}\} = \begin{Bmatrix} 0.469 \\ 0.875 \\ 1.219 \\ 1.500 \end{Bmatrix} \tag{2–35}$$

A comparison of the finite difference and analytic solutions is given in Figure 2–6. Note that the finite difference solution is exact at the nodes. This can be proved (Hughes), and the same behavior will be seen for one-dimensional finite element solutions.

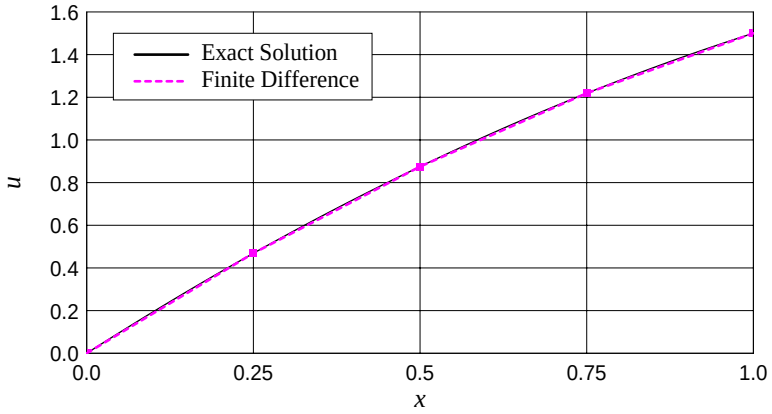


Fig. 2–6 Finite difference solution.

Chapter 3

Weighted Residuals

3.1 Objective

Our objective is to introduce the method of weighted residuals for obtaining approximate solutions to differential equations.

We will first introduce the method of weighted residuals using a single term approximation and show how different methods result in different results.

3.2 Weighted Residual Method

Weighted residual methods are another way to develop approximate solutions. Recall that in the finite difference method we developed an approximation to the differential equation at a *point*. In contrast, in weighted residual methods, we assume the form of the *global* solution and then adjust parameters to obtain the best global fit to the actual solution.

Let us be more precise in our definitions. We are given a body B with boundary S. As shown in Figure 3–1, the boundary is divided into two regions, a region S_u with essential (Dirichlet) boundary conditions and a region S_f with natural (Neumann) boundary conditions. The **essential boundary conditions** are specifications of the solution on the boundary (for example, known boundary displacements), while the **natural boundary conditions** are specifications of derivatives of the solution (for example, surface tractions). *All points on the boundary must have one or the other type of specified boundary condition if the problem is well defined mathematically.*

In this lecture, we will present the weighted residual methods in ways that require the approximating function to satisfy both the essential and natural boundary conditions. In the next lecture, we will show how the weak form of the differential equation can be used to develop methods that loosen this requirement, so that only the essential boundary

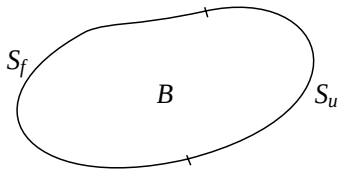


Fig. 3–1 General body with boundary.

conditions must be satisfied by our approximating function. The basic step in weighted residual methods is to assume a solution of the form:

$$u_n = \sum_{i=1}^{n} a_i \phi_i \tag{3–1}$$

Our job is to solve for the coefficients a_i that give a best approximation (by some measure) to the exact solution.

3.3 Solutions Using Weighted Residuals

Let us use weighted residuals to solve the problem of a rod subjected to body and end loads (Figure 3–1). As previously derived, the differential equation for a constant body load is:

$$AE\frac{d^2u}{dx^2} + b = 0 \qquad \text{for} \quad 0 \le x \le L \tag{3–2}$$

with boundary conditions:

$$u|_{x=0} = 0 \tag{3–3}$$

$$EA\frac{du}{dx}\bigg|_{x=L} = P \tag{3–4}$$

For the weighted residual formulation, we first choose a weighting function $w(x)$ and multiply eqn. (3–2) by the weighting function:

$$w\left(AE\frac{d^2u}{dx^2} + b\right) = 0 \tag{3–5}$$

and then integrate over the entire body:

$$\int_0^L w\left(AE\frac{d^2u}{dx^2} + b\right)dx = 0 \tag{3–6}$$

This is called the weighted residual formulation. It is called this because if we assume an approximate solution u_n (that satisfies all boundary conditions) then it will not satisfy the differential equation exactly.

$$AE\frac{d^2u_n}{dx^2} + b = R(x) \neq 0 \tag{3–7}$$

Instead, we have an error, called the residual $R(x)$, that is a function of x. Thus eqn. (3–6) takes the residual at all points, multiplies it by a weighting factor, integrates this result over the body, and then sets the integral to zero.

$$\int_0^L wRdx = 0 \tag{3–8}$$

Question 1: *Why can we say the weighted residual form of an equation is more general than the traditional concept of equation?*

3.4 Example Using One Term Approximation

To see how this allows us to obtain values for the coefficients, let us complete an example. We will pick an approximate function that satisfies all boundary conditions:

$$u_n = \frac{P}{AE}x + a_1 \sin\left(\frac{\pi x}{2L}\right) \tag{3–9}$$

Clearly, this function satisfies the essential boundary condition (eqn. (3–3)). In addition, the first term satisfies the natural boundary condition (eqn. (3–4)). The second term is zero at the essential condition and has zero slope at the natural condition, so it does not add additional terms at the boundaries. Taking the derivatives:

$$\begin{aligned}\frac{du_n}{dx} &= \frac{P}{AE} + a_1\frac{\pi}{2L}\cos\left(\frac{\pi x}{2L}\right)\\ \frac{d^2u_n}{dx^2} &= -a_1\left(\frac{\pi}{2L}\right)^2 \sin\left(\frac{\pi x}{2L}\right)\end{aligned} \tag{3–10}$$

Substitute eqn. (3–10) into eqn. (3–7) to obtain the residual,

$$-AEa_1\left(\frac{\pi}{2L}\right)^2 \sin\left(\frac{\pi x}{2L}\right) + b = R(x) \tag{3–11}$$

For simplicity, pick $A = E = P = L = b = 1$. Then eqn. (3–11) becomes:

$$R(x) = -a_1\frac{\pi^2}{4}\sin\left(\frac{\pi x}{2}\right) + 1 \tag{3–12}$$

Figure 3–2 plots the residual according to eqn. (3–12) for different values of a_1.

Now it is clear why we integrate. The error is a function of x. We can weight the error (residual) in any way we want over the interval and force the integral to zero. Depending on how we weight the residual, we get different solutions.

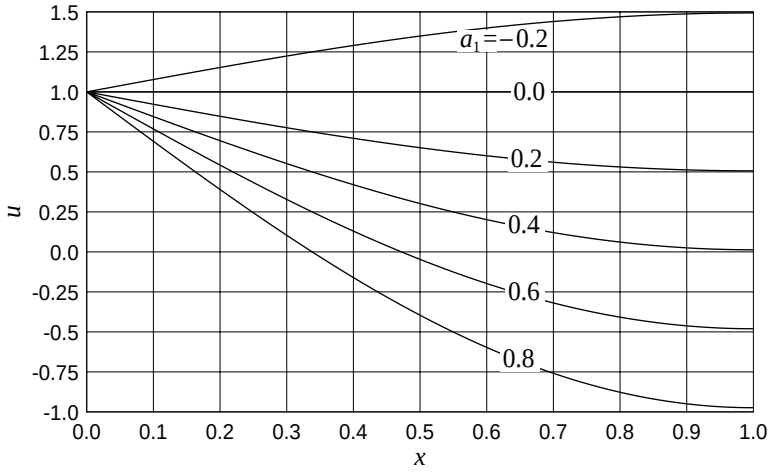


Fig. 3–2 Residual plotted over body for different values of a_1.

3.4.1 Callocation Method

In this case, we force the residual to be zero at a specific location ("nail-down" method). We accomplish this by selecting the *Dirac delta function* as the weighting function:

$$w(x) = \delta(x - x_i) = \begin{cases} \infty & \text{when} \quad x = x_i \\ 0 & \text{when} \quad x \neq x_i \end{cases}$$

The integral of the Dirac delta function is one, so for any function $f(x)$:

$$\int_{-\infty}^{\infty} f(x)\delta(x - x_i)dx = f(x_i)$$

The weighted residual according to eqn. (3–8) is:

$$\int_{o}^{1} \delta(x - x_i)Rdx = 0 \tag{3–13}$$

or, using the Dirac delta function properties,

$$R(x_i) = 0 \tag{3–14}$$

If we pick $x_i = 0.5$, that is, we want the residual to equal zero at the midpoint, then inspection of Figure 3–2 shows that a value of $a_1 \approx 0.60$ satisfies that condition. Solving exactly,

$$-a_1 \frac{\pi^2}{4} \sin \frac{\pi}{4} + 1 = 0$$

$$a_1 = 0.573 \tag{3–15}$$

3.4.2 Subdomain Method

Alternately, let us weight the residual uniformly over the interval ("glue" method). That is:

$$w(x) = 1$$

Then,

$$\int_0^1 (1) R dx = 0 \tag{3–16}$$

By inspection, we can again see that a value of $a_1 = 0.6$ will result in approximately equal areas above and below the x axis. Let us solve exactly:

$$\begin{aligned} \int_0^1 1\left(-a_1 \frac{\pi^2}{4} \sin\frac{\pi x}{2} + 1\right) dx &= 0 \\ \left(a_1 \frac{\pi}{2} \cos\frac{\pi x}{2} + x\right)\Big|_0^1 &= 0 \\ 1 - a_1 \frac{\pi}{2} &= 0 \\ a_i = \frac{2}{\pi} &= 0.637 \end{aligned} \tag{3–17}$$

3.4.3 Least Squares Method

In the least squares method, we require that the squared residual be minimized with respect to the adjusting parameter, *i.e.*,

$$\text{Minimize}\left(\int_0^1 R^2 dx\right)$$

or

$$\int_0^1 R \frac{\partial R}{\partial a_i} dx = 0 \tag{3–18}$$

This is equivalent to selecting $\partial R/\partial a_i$ as the weighting function. For our particular problem, there is only one adjusting parameter a_1, and

$$\begin{aligned} \frac{\partial R}{\partial a_1} &= -\frac{\pi^2}{4} \sin\frac{\pi x}{2} \\ \int_0^1 R \frac{\partial R}{\partial a_i} dx = \int_0^1 \left(-a_1 \frac{\pi^2}{4} \sin\frac{\pi x}{2} + 1\right)\left(-\frac{\pi^2}{4} \sin\frac{\pi x}{2}\right) dx &= 0 \\ -a_1 \frac{\pi^2}{4} \int_0^1 \sin^2\frac{\pi x}{2} dx + \int_0^1 \sin\frac{\pi x}{2} dx &= 0 \end{aligned}$$

$$-a_1\frac{\pi^2}{4}\left(\frac{1}{2}x-\frac{1}{2\pi}\sin(\pi x)\right)\Big|_0^1+\left(-\frac{1}{\pi}\cos(\pi x)\right)\Big|_0^1=0$$

$$-a_1\frac{\pi^2}{8}+\frac{2}{\pi}=0$$

$$a_1=\frac{16}{\pi^3}=0.516 \tag{3–19}$$

3.4.4 Galerkin Method

Finally, if we use the same function for the weighting function as we used for the approximating function (except that we do not include the term in the approximating function that satisfies the essential boundary conditions, which does not enter into this case) then we have:

$$w(x)=d_1\sin\frac{\pi x}{2} \tag{3–20}$$

solving,

$$\int_0^1\left(d_1\sin\frac{\pi x}{2}\right)\left(-a_1\frac{\pi^2}{4}\sin\frac{\pi x}{2}+1\right)dx=0$$

$$-a_1\frac{\pi^2}{4}\int_0^1\sin^2\frac{\pi x}{2}dx+\int_0^1\sin\frac{\pi x}{2}dx=0$$

$$-a_1\frac{\pi^2}{4}\left(\frac{1}{2}x-\frac{1}{2\pi}\sin(\pi x)\right)\Big|_0^1+\left(-\frac{1}{\pi}\cos(\pi x)\right)\Big|_0^1=0$$

$$-a_1\frac{\pi^2}{8}+\frac{2}{\pi}=0$$

which leads to:

$$a_1=\frac{16}{\pi^3}=0.516 \tag{3–21}$$

Note that for our particular example the least-squares method and the Galerkin method yield identical results. In general, however, the two methods may give different answers.

Question 2: *How do you judge whether an approximate method is legitimate or acceptable? If there are two approximate methods, with what criterion can you say one is better than the other?*

Question 3: *Theoretically, how can you obtain an exact solution (or as accurate as you like) with an approximate method?*

3.4.5 Comparison of Solutions

Figure 3–3 compares the analytical and approximate solutions. This shows that the Galerkin (and least squares) methods give the best approximation to the actual solution.

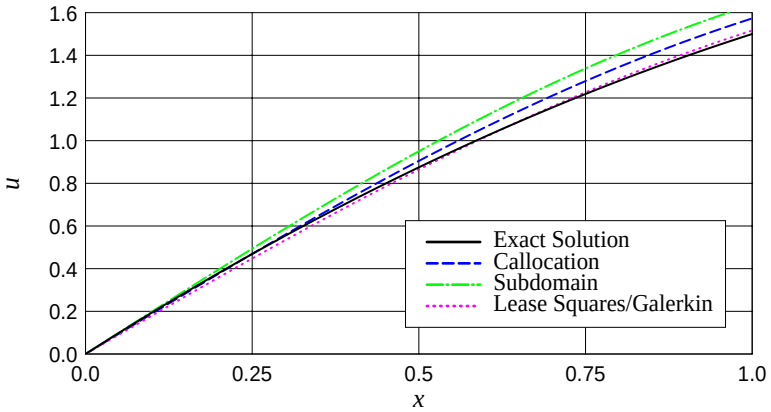


Fig. 3–3 Comparison of analytical and approximate weighted residual solutions.

3.5 Using More Terms in Approximation

It would be possible to solve this problem using more terms in the approximate solution, but practically speaking, this is not a good approach if we must satisfy natural boundary conditions. It is much better to proceed to the weighted residuals formulation based on the weak form of the differential equation. This formulation will automatically incorporate the natural boundary conditions, making it much simpler to develop the approximate solution.

3.6 Hints to Questions

Hint 1: The weighted residual approach extends the concept of solution which includes the exact solution.

Hint 2: The method must not exclude the exact solution if found; the method should produce the exact solution for constant function.

Hint 3: The bases for functional space should be complete, and number of terms must be large enough.

Chapter 4

Strong and Weak Forms of the Differential Equation

4.1 Purpose

The purpose of this chapter is to introduce the weak form of the differential equation. This is used for the basis of our final finite element formulation. It will give us a symmetric stiffness matrix (for the elasticity problem) and clearly shows the natural boundary condition (surface traction) terms.

We will use the one-dimensional elastic rod as our prototype for the derivation.

4.2 Weak Form of Differential Equation

The strong statement of the differential equation is the one we derived using the differential element (Chapter 2). It is given by:

$$AE\frac{d^2u}{dx^2} + b = 0 \qquad \text{for} \quad 0 \le x \le L \tag{4–1}$$

with the boundary conditions:

$$u|_{x=0} = 0 \tag{4–2}$$

an essential boundary condition, and

$$EA\frac{du}{dx}\bigg|_{x=L} = P \tag{4–3}$$

a natural boundary condition.

We call the differential eqn. (4–1) with boundary conditions eqns. (4–2) and (4–3) the **strong form** of the differential equation. Sometimes this is also called the **strong statement**. This is the conventional form of differential equation in almost all calculus courses.

Now let us proceed to derive the weak form. Similar to Chapter 3, we multiply eqn. (4–1) by a weighting function $w(x)$. This function is chosen *to be zero at all essential boundary conditions (but not zero at the natural boundary conditions)*. We then integrate over the body:

$$\int_0^L w\left(AE\frac{d^2u}{dx^2} + b\right)dx = 0 \tag{4–4}$$

Now separate the integral:

$$AE\int_0^L w\left(\frac{d^2u}{dx^2}\right)dx + \int_0^L wbdx = 0 \tag{4–5}$$

Recall integration by parts, where:

$$\frac{d}{dx}\left(w\frac{du}{dx}\right) = \frac{dw}{dx}\frac{du}{dx} + w\frac{d^2u}{dx^2}$$

Rearranging gives:

$$w\frac{d^2u}{dx^2} = \frac{d}{dx}\left(w\frac{du}{dx}\right) - \frac{dw}{dx}\frac{du}{dx} \tag{4–6}$$

Substitute eqn. (4–6) into eqn. (4–5) gives:

$$AE\int_0^L \frac{d}{dx}\left(w\frac{du}{dx}\right)dx - AE\int_0^L \frac{dw}{dx}\frac{du}{dx}dx + \int_0^L wbdx = 0 \tag{4–7}$$

The first term on the left hand side of eqn. (4–7) can be simplified using the fundamental theorem of calculus:

$$AE\int_0^L \frac{d}{dx}\left(w\frac{du}{dx}\right)dx = EA\left(w\frac{du}{dx}\right)\Big|_0^L = EA\left(w\frac{du}{dx}\right)\Big|^L - EA\left(w\frac{du}{dx}\right)\Big|_0 \tag{4–8}$$

However, we have specifically chosen $w(x)$ to be zero at the essential boundary conditions, that is, $u = 0$ at $x = 0$. As a result, the second term on the right hand side of eqn. (4–8) is zero, and eqn. (4–8) becomes,

$$AE\int_0^L \frac{d}{dx}\left(w\frac{du}{dx}\right)dx = w(L)\left(AE\frac{du}{dx}\right)\Big|_L = w(L)P \tag{4–9}$$

Substituting into eqn. (4–7) gives:

$$AE\int_0^L \frac{dw}{dx}\frac{du}{dx}dx = \int_0^L wbdx + w(L)P \tag{4–10}$$

Equation eqn. (4–10) is called the **weak form** of the differential equation. Sometimes it is also called the **weak statement**. The word weak here does not imply anything that is wrong or inadaquate. It means that some requirements might be loosened. *The weak statement is equivalent to the strong statement.* This is carefully discussed in Hughes, 1987.

The advantage of the weak statement is that we have *reduced the order of the differential by one* and made possible *symmetry of the resulting matrix formulation* when we approximate the true solution. We have also explicitly handled the natural boundary conditions, so that we can include them in our analysis. Note also that the weak form is still a weighted residual statement. The weak form does not show the weighted residual formulation as clearly, but it is there.

Question 1: *What are the similarities and differences between the weak form in eqn. (4–10) and the weighted residual form in previous lecture?*

Question 2: *What is the advantage of reducing the order of the differentiation of the trial function u?*

Question 3: *What is the advantage of having a symmetric stiffness matrix?*

4.3 Approximate Solution Using the Weak Formulation

The weak form will be the starting point of our approximate solution. We approximate the solution using a function in which the first known term (or terms) satisfies the essential boundary conditions. The rest of the approximate solution consists of functions that are zero at the essential boundary conditions and are multiplied by coefficients to be determined.

$$u_n = \phi_0 + \sum_{j=1}^{n} a_j \phi_j \tag{4–11}$$

The derivative of the approximate solution is given by:

$$\frac{du_n}{dx} = \frac{d\phi_0}{dx} + \sum_{j=1}^{n} a_j \frac{d\phi_j}{dx} \tag{4–12}$$

We will use the Galerkin approach, so the weighting function will be the same as the approximate function, but without the essential boundary condition terms.

$$w_n = \sum_{i=1}^{n} d_i \phi_i \tag{4–13}$$

and the derivatives of the weighing function are:

$$\frac{dw_n}{dx} = \sum_{i=1}^{n} d_i \frac{d\phi_i}{dx} \tag{4–14}$$

Substituting eqns. (4–12) through (4–14) into eqn. (4–10):

$$AE \int_0^L \left(\sum_{i=1}^{n} d_i \frac{d\phi_i}{dx} \right) \left(\frac{d\phi_0}{dx} + \sum_{j=1}^{n} a_j \frac{d\phi_j}{dx} \right) dx = \int_0^L \left(\sum_{i=1}^{n} d_i \phi_i \right) b dx + \left(\sum_{i=1}^{n} d_i \phi_i(L) \right) P \tag{4–15}$$

Rearranging:

$$\sum_{i=1}^{n} d_i \left[AE \int_0^L \left(\frac{d\phi_i}{dx} \right) \left(\frac{d\phi_0}{dx} + \sum_{j=1}^{n} a_j \frac{d\phi_j}{dx} \right) dx - \int_0^L \phi_i b dx - \phi_i(L) P \right] = 0 \tag{4–16}$$

Since this must hold true for all d_i, then:

$$AE \int_0^L \left(\frac{d\phi_i}{dx} \right) \left(\frac{d\phi_0}{dx} + \sum_{j=1}^{n} a_j \frac{d\phi_j}{dx} \right) dx - \int_0^L \phi_i b dx - \phi_i(L) P = 0 \tag{4–17}$$

or, for $i = 1, 2, \cdots, n$,

$$AE \int_0^L \left(\frac{d\phi_i}{dx} \right) \left(\frac{d\phi_0}{dx} + \sum_{j=1}^{n} a_j \frac{d\phi_j}{dx} \right) dx = \int_0^L \phi_i b dx + \phi_i(L) P \tag{4–18}$$

Since the terms satisfying the essential boundary conditions ϕ_0 are known, we will move them to the right hand side also:

$$AE \int_0^L \left(\frac{d\phi_i}{dx} \right) \left(\sum_{j=1}^{n} a_j \frac{d\phi_j}{dx} \right) dx = \int_0^L \phi_i b dx - AE \int_0^L \left(\frac{d\phi_i}{dx} \right) \left(\frac{d\phi_0}{dx} \right) dx + \phi_i(L) P \tag{4–19}$$

The term on the left contains the unknown coefficients. The first term on the right comes from body loads. The second term is the contribution of the essential boundary conditions. The third term is the contribution of the natural boundary conditions. Note that it is not necessary that all these terms appear in every problem.

Question 4: *In what aspects are eqns. (4–19) and (4–10) different?*

Equation eqn. (4–19) is really a set of n equations with n unknowns. We can expand eqn. (4–19) to see the individual terms.

$$\begin{bmatrix} AE\int_0^L \left(\frac{d\phi_1}{dx}\right)\left(\frac{d\phi_1}{dx}\right)dx & AE\int_0^L \left(\frac{d\phi_1}{dx}\right)\left(\frac{d\phi_2}{dx}\right)dx & \cdots & AE\int_0^L \left(\frac{d\phi_1}{dx}\right)\left(\frac{d\phi_n}{dx}\right)dx \\ AE\int_0^L \left(\frac{d\phi_2}{dx}\right)\left(\frac{d\phi_1}{dx}\right)dx & AE\int_0^L \left(\frac{d\phi_2}{dx}\right)\left(\frac{d\phi_2}{dx}\right)dx & \cdots & AE\int_0^L \left(\frac{d\phi_2}{dx}\right)\left(\frac{d\phi_n}{dx}\right)dx \\ \vdots & \vdots & \vdots & \vdots \\ AE\int_0^L \left(\frac{d\phi_n}{dx}\right)\left(\frac{d\phi_1}{dx}\right)dx & AE\int_0^L \left(\frac{d\phi_n}{dx}\right)\left(\frac{d\phi_2}{dx}\right)dx & \cdots & AE\int_0^L \left(\frac{d\phi_n}{dx}\right)\left(\frac{d\phi_n}{dx}\right)dx \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{Bmatrix}$$

$$= \begin{Bmatrix} \int_0^L \phi_1 b dx + \phi_1(L)P - AE\int_0^L \left(\frac{d\phi_1}{dx}\right)\left(\frac{d\phi_0}{dx}\right)dx \\ \int_0^L \phi_2 b dx + \phi_2(L)P - AE\int_0^L \left(\frac{d\phi_2}{dx}\right)\left(\frac{d\phi_0}{dx}\right)dx \\ \vdots \\ \int_0^L \phi_n b dx + \phi_n(L)P - AE\int_0^L \left(\frac{d\phi_n}{dx}\right)\left(\frac{d\phi_0}{dx}\right)dx \end{Bmatrix} \tag{4–20}$$

There is beauty and power here. We now have a general method to develop approximate solutions to differential equations. We must select appropriate shape functions, but as we will see, we can automate that using finite elements. If we write eqn. (4–20) in matrix form, this will be:

$$[\boldsymbol{K}]\{\boldsymbol{a}\} = \{\boldsymbol{f}\} \tag{4–21}$$

This is the basis of our future finite element development. As derived, we have assumed that the shape functions are global functions over the entire body. We will show that the finite element formulation arises when we use local shape functions.

Question 5: *From eqn. (4–20), list some reasons that the Galerkin method is better than other methods such as collocation and sub-domain methods.*

4.4 Example Solution Using Single Term Approximation

To illustrate obtaining a solution using the weak formulation, let us solve the same problem as done in the previous chapter. This is an elastic rod with body and end forces. We assume the approximate solution:

$$u_n = a_1\phi_1 \tag{4–22}$$

where

$$\phi_1 = \sin \frac{\pi x}{2L} \tag{4–23}$$

This function was chosen to be zero at the essential boundary condition. The derivative is:

$$\frac{d\phi_1}{dx} = \frac{\pi}{2L} \cos \frac{\pi x}{2L} \tag{4–24}$$

Substituting into equations eqn. (4–20) (assume $A = E = L = P = b = 1$) gives:

$$\left[\frac{\pi^2}{4} \int_0^1 \left(\cos \frac{\pi x}{2} \cos \frac{\pi x}{2} \right) dx \right] \{a_1\} = \left\{ \int_0^1 \sin \frac{\pi x}{2} dx + 1 \right\} \tag{4–25}$$

Evaluating the integrals:

$$\begin{aligned} \left[\frac{\pi^2}{4} \left(\frac{1}{2} x + \frac{1}{2\pi} \sin(\pi x) \right) \Big|_0^1 \right] \{a_1\} &= \left\{ -\frac{2}{\pi} \cos \frac{\pi x}{2} \Big|_0^1 + 1 \right\} \\ \frac{\pi^2}{8} a_1 &= \frac{2}{\pi} + 1 \\ a_1 &= 1.326 \end{aligned} \tag{4–26}$$

This is a rough approximation, since we only have one term in the approximate solution.

4.5 Example Solution Using Two Terms

We will now do an example solution using two terms in the approximation. We will add a linear term to our previous solution.

$$u_n = \sum_{j=1}^{2} a_j \phi_j \tag{4–27}$$

where:

$$\phi_1 = x \tag{4–28}$$

$$\phi_2 = \sin \frac{\pi x}{2L} \tag{4–29}$$

These functions both are zero at the essential boundary condition. The derivatives are:

$$\frac{d\phi_1}{dx} = 1 \tag{4–30}$$

$$\frac{d\phi_2}{dx} = \frac{\pi}{2L} \cos \frac{\pi x}{2L} \tag{4–31}$$

Equation (4–20) now becomes:

$$\begin{bmatrix} AE\int_0^L (1)(1)dx & AE\int_0^L (1)\left(\frac{\pi}{2L}\cos\frac{\pi x}{2L}\right)dx \\ AE\int_0^L \left(\frac{\pi}{2L}\cos\frac{\pi x}{2L}\right)(1)dx & AE\int_0^L \left(\frac{\pi}{2L}\cos\frac{\pi x}{2L}\right)\left(\frac{\pi}{2L}\cos\frac{\pi x}{2L}\right)dx \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix} \tag{4–32}$$

where

$$\begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix} = \begin{Bmatrix} \int_0^L xbdx + LP \\ \int_0^L \sin\frac{\pi x}{2L}dx + (1)P \end{Bmatrix} \tag{4–33}$$

Again, assume $A = E = L = P = b = 1$. Performing the integrations in eqns. (4–32) and (4–33) gives:

$$\begin{bmatrix} 1 & \frac{\pi}{2} \\ \frac{\pi}{2} & \frac{\pi^2}{8} \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \end{Bmatrix} = \begin{Bmatrix} \frac{1}{2}+1 \\ \frac{2}{\pi}+1 \end{Bmatrix} \tag{4–34}$$

Solving:

$$\begin{Bmatrix} a_1 \\ a_2 \end{Bmatrix} = \begin{Bmatrix} 0.915 \\ 0.584 \end{Bmatrix} \tag{4–35}$$

Both solutions are plotted in Figure 4–1. Clearly the two term solution is very close to the exact analytic solution.

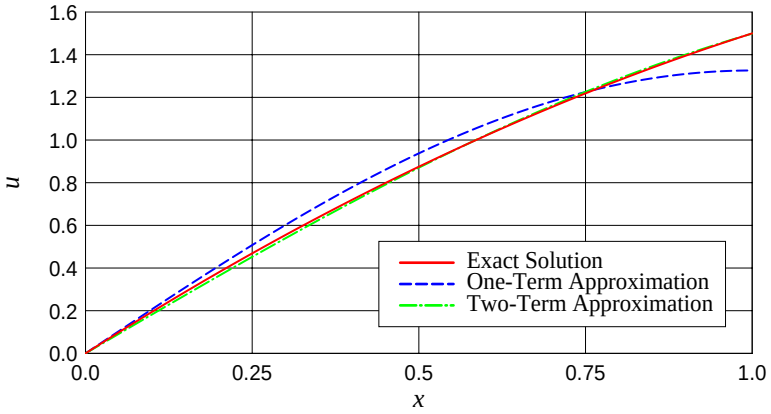


Fig. 4–1 Comparison of one and two term solutions using weak forumulation.

4.6 Hints to Questions

Hint 1:

Hint 2: Allows for the use of piecewise function; weakens the requirements of differentiability.

Hint 3: Increases computational efficiency; reduce storage requirement.

Hint 4: One is exact, while the other is approximate.

Hint 5:

Chapter 5

Hat Functions

5.1 Objective

Our objective is to use hat functions as a prelude to defining a finite element and its associated shape functions.

We will first describe hat functions. We will then show that hat functions are equivalent to defining shape functions over an element. The student will also see that the global integration is equivalent to the summation of integration over all elements.

5.2 Hat Functions

We are now approaching the final definition of a finite element. Before we get there, we will first look at special hat functions that are global functions, but only non-zero locally. We will use these functions in our previously derived Galerkin weighted residual formulation that we developed from the weak form of the differential equation. Again, we will be solving the elastic bar. We will define "nodes" on the bar as shown in Figure 5–1.

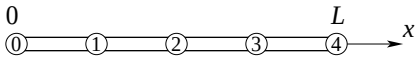
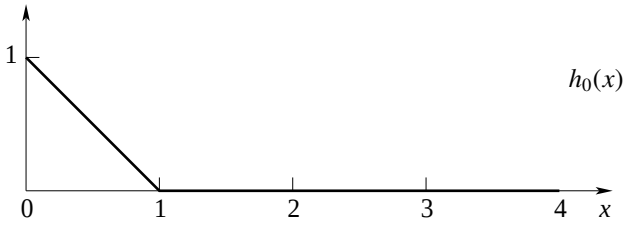


Fig. 5–1 Discretization for hat functions.

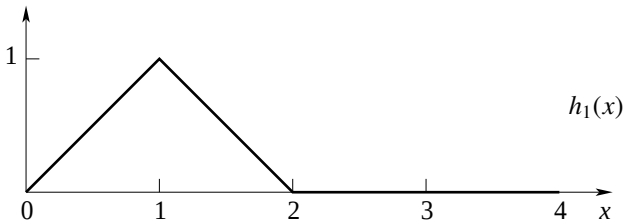
The associated hat functions are shown in Figure 5–1.

Note that once the hat function for $x = 0$, *i.e.*, h_0 for $x_{-1} < x < x_1$, is defined, the hat function centered at x_i can be simply expressed by $h_i(x) = h_0(x - x_i)$.

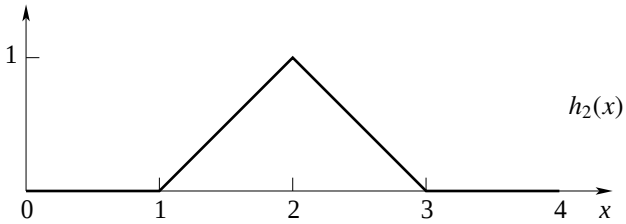
The hat functions can be used to create a multi-linear (piecewise linear) curve by adding them together and using different coefficients multiplying each hat function, as



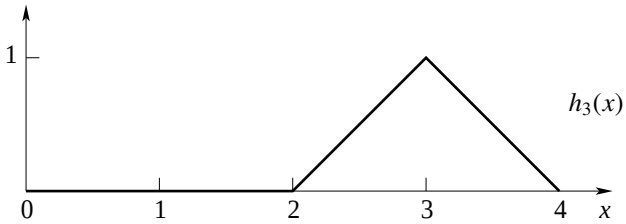
$$h_0(x) = \begin{cases} 1 - \dfrac{x - x_0}{x_1 - x_0} & x_0 \le x \le x_1 \\ 0 & \text{Otherwise} \end{cases}$$



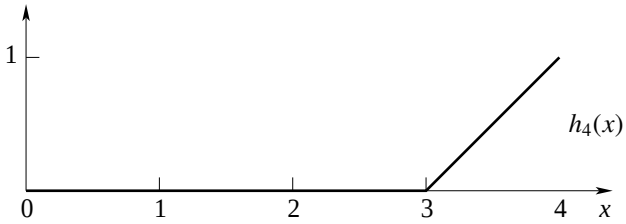
$$h_1(x) = \begin{cases} \dfrac{x - x_0}{x_1 - x_0} & x_0 \le x \le x_1 \\ 1 - \dfrac{x - x_1}{x_2 - x_1} & x_1 \le x \le x_2 \\ 0 & \text{Otherwise} \end{cases}$$



$$h_2(x) = \begin{cases} \dfrac{x - x_1}{x_2 - x_1} & x_1 \le x \le x_2 \\ 1 - \dfrac{x - x_2}{x_3 - x_2} & x_2 \le x \le x_3 \\ 0 & \text{Otherwise} \end{cases}$$



$$h_3(x) = \begin{cases} \dfrac{x - x_2}{x_3 - x_2} & x_2 \le x \le x_3 \\ 1 - \dfrac{x - x_3}{x_4 - x_3} & x_3 \le x \le x_4 \\ 0 & \text{Otherwise} \end{cases}$$



$$h_4(x) = \begin{cases} \dfrac{x - x_3}{x_4 - x_3} & x_3 \le x \le x_4 \\ 0 & \text{Otherwise} \end{cases}$$

Fig. 5–2 Hat functions.

illustrated in Figure 5–3:

$$f(x) \approx \sum_{i}^{N} f(x_i) h_i(x) \tag{5–1}$$

Notice that at the node the approximation is exactly the nodal value, since the hat function

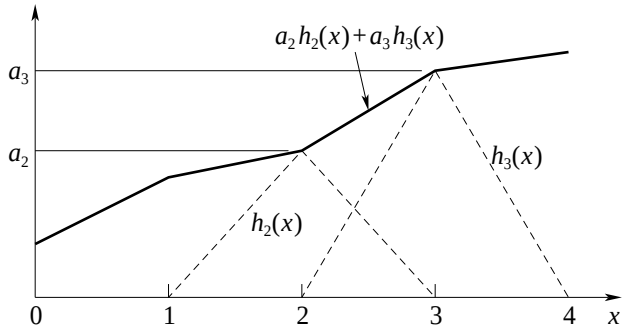


Fig. 5–3 Multi-linear curve generated using hat functions with variable coefficients.

equals 1 at each node. Between nodes, there is a linear interpolation.

Question 1: *Compare the hat function and the Dirac delta function*

$$f(x) \approx \sum_{i}^{N} f(x_i) h_i(x) = \sum_{i}^{N} f(x_i) h_0(x - x_i)$$
$$f(x) = \int_{-\infty}^{\infty} f(\xi) \delta(x - \xi) d\xi$$

5.3 Solution Using Hat Functions

We will now use the hat functions to solve a problem. We will use the Galerkin method developed in Chapter 4. The problem we will solve is the same elastic rod problem with body and end loads. The rod is divided as shown in Figure 5–1.

The approximate solution is developed using the hat functions shown in Figure 5–2. Note that there is an essential boundary condition at node 0 and a natural boundary condition at node 4. The value of the essential boundary conditions is zero. Recall that the first term (terms) in the approximate solution satisfy the essential boundary conditions, the rest are zero at the essential boundary conditions. Because the essential boundary condition is zero, we do not need any special functions to satisfy it. However, using the function:

$$\phi_0 = u_0 h_0 \tag{5–2}$$

would satisfy the essential boundary condition and could be used in a problem where the essential boundary condition is not zero but is equal to some value u_0.

Then, the hat functions associated with nodes 1, 2, 3, and 4 will be the ones we will use in our approximate solution.

$$u_n = u_1 h_1 + u_2 h_2 + u_3 h_3 + u_4 h_4 \tag{5–3}$$

For fitting the approximating function to the matrix form of the Galerkin method derived using the weak form of the differential equation in Chapter 4, we have:

$$u_n = a_1 \phi_1 + a_2 \phi_2 + a_3 \phi_3 + a_4 \phi_4 \tag{5–4}$$

Comparing eqns. (5–3) and (5–4), we recognize the correspondence between the nodal displacements u_i and the coefficients a_i and between the hat functions h_i and the functions ϕ_i. That is:

$$\phi_1 = h_1 = \begin{cases} \dfrac{x - x_0}{x_1 - x_0} & \text{for} \quad x_0 \le x \le x_1 \\ 1 - \dfrac{x - x_1}{x_2 - x_1} & \text{for} \quad x_1 \le x \le x_2 \end{cases} \tag{5–5}$$

and,

$$\frac{d\phi_1}{dx} = \frac{dh_1}{dx} = \begin{cases} \dfrac{1}{x_1 - x_0} & \text{for} \quad x_0 \le x \le x_1 \\ \dfrac{-1}{x_2 - x_1} & \text{for} \quad x_1 \le x \le x_2 \end{cases} \tag{5–6}$$

Note that if the division is even along the x-axis, we have, $x_{i+1} - x_i = L/4$, and,

$$\frac{d\phi_i}{dx} = \begin{cases} \dfrac{4}{L} & \text{for} \quad x_{i-1} \le x \le x_i \\ -\dfrac{4}{L} & \text{for} \quad x_i \le x \le x_{i+1} \end{cases} \tag{5–7}$$

We will now evaluate the first row of the in the Galerkin formulation of Chapter 4, eqn. (4–20). The first coefficient is given by (where we have assumed $A = E = 1$):

$$\int_0^L \left(\frac{d\phi_1}{dx}\right)\left(\frac{d\phi_1}{dx}\right) dx \tag{5–8}$$

For ϕ_1, which is zero except for $x_0 \le x \le x_2$, we will divide the integral into two parts as shown below:

$$\int_0^L \left(\frac{d\phi_1}{dx}\right)\left(\frac{d\phi_1}{dx}\right) dx = \int_{x_0}^{x_1} \left(\frac{d\phi_1}{dx}\right)\left(\frac{d\phi_1}{dx}\right) dx + \int_{x_1}^{x_2} \left(\frac{d\phi_1}{dx}\right)\left(\frac{d\phi_1}{dx}\right) dx \tag{5–9}$$

Substituting eqn. (5–6) into eqn. (5–9) and using the values for the positions of the nodes ($L = 1$), gives:

$$\int_0^L \left(\frac{d\phi_1}{dx}\right)\left(\frac{d\phi_1}{dx}\right)dx = \int_0^{0.25} \left(\frac{1}{0.25}\right)\left(\frac{1}{0.25}\right)dx + \int_{0.25}^{0.5} \left(\frac{-1}{0.25}\right)\left(\frac{-1}{0.25}\right)dx \tag{5–10}$$

Evaluating the integrals,

$$\int_0^L \left(\frac{d\phi_1}{dx}\right)\left(\frac{d\phi_1}{dx}\right)dx = 16(x)|_0^{0.25} + 16(x)|_{0.25}^{0.5}$$

$$\int_0^L \left(\frac{d\phi_1}{dx}\right)\left(\frac{d\phi_1}{dx}\right)dx = 8 \tag{5–11}$$

The second term in the Galerkin formulation of Chapter 4 is given by:

$$\int_0^L \left(\frac{d\phi_1}{dx}\right)\left(\frac{d\phi_2}{dx}\right)dx \tag{5–12}$$

Carefully examining Figure 5–2, we see that the only region over the body that the two approximating functions interact is over the region $x_1 \leq x \leq x_2$. Then the integral becomes:

$$\int_0^L \left(\frac{d\phi_1}{dx}\right)\left(\frac{d\phi_2}{dx}\right)dx = \int_{x_1}^{x_2} \left(\frac{-1}{x_2 - x_1}\right)\left(\frac{1}{x_2 - x_1}\right)dx \tag{5–13}$$

Evaluating:

$$\int_0^L \left(\frac{d\phi_1}{dx}\right)\left(\frac{d\phi_2}{dx}\right)dx = \int_{0.25}^{0.5} \left(\frac{-1}{0.25}\right)\left(\frac{1}{0.25}\right)dx \tag{5–14}$$

or,

$$\int_0^L \left(\frac{d\phi_1}{dx}\right)\left(\frac{d\phi_2}{dx}\right)dx = -16(x)|_{0.25}^{0.5}$$

and,

$$\int_0^L \left(\frac{d\phi_1}{dx}\right)\left(\frac{d\phi_2}{dx}\right)dx = -4 \tag{5–15}$$

The third term in the Galerkin formulation of Chapter 4 is given by:

$$\int_0^L \left(\frac{d\phi_1}{dx}\right)\left(\frac{d\phi_3}{dx}\right)dx \tag{5–16}$$

Looking again at Figure 5–2, we see that the two shape functions do not interact. Therefore, the integral is zero.

The remainder of the integrals can be evaluated by the student.

We will also evaluate the right hand side of the Galerkin equation developed in Chapter 4. The first term to be evaluated is the one that includes the body force:

$$\int_0^L \phi_1 dx \tag{5–17}$$

where we have assumed $b = 1$. As before, we divide the integral into two parts:

$$\int_0^L \phi_1 dx = \int_{x_0}^{x_1} \left(\frac{x - x_0}{x_1 - x_0} \right) dx + \int_{x_1}^{x_2} \left(1 - \frac{x - x_1}{x_2 - x_1} \right) dx \tag{5–18}$$

Evaluating:

$$\int_0^L \phi_1 dx = \int_0^{0.25} \left(\frac{x - 0}{0.25} \right) dx + \int_{0.25}^{0.5} \left(1 - \frac{x - 0.25}{0.25} \right) dx$$

Integrating,

$$\int_0^L \phi_1 dx = 4 \frac{x^2}{x} \Big|_0^{0.25} + (2x - 2x^2) \Big|_{0.25}^{0.5}$$

or,

$$\int_0^L \phi_1 dx = \frac{1}{8} + \frac{1}{8} = \frac{1}{4} \tag{5–19}$$

The second term on the right hand side is:

$$\phi_1(L)P \tag{5–20}$$

Clearly, $\phi_1(L) = 0$, so the natural boundary condition term is zero. For this problem, the essential boundary conditions are zero, therefore the last term for f_1 does not contribute anything.

Similar integrations are performed for the other terms. These should be done by the student. The only row in which the natural boundary condition term contributes is for 4, where $\phi_4(L)P = 1$. When all integrations are completed, the coefficients obtained using the hat functions will be those given below:

$$\begin{bmatrix} 8 & -4 & 0 & 0 \\ -4 & 8 & -4 & 0 \\ 0 & -4 & 8 & -4 \\ 0 & 0 & -4 & 4 \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{Bmatrix} = \begin{Bmatrix} \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{8} + 1 \end{Bmatrix} \tag{5–21}$$

Solving gives:

$$\{\boldsymbol{a}\} = \begin{Bmatrix} 0.46875 \\ 0.875 \\ 1.21875 \\ 1.5 \end{Bmatrix} \tag{5–22}$$

The comparison with the analytic solution is given in Figure 5–4. As for the finite difference solution, the solution is exact at the nodes. This is only true in the one-dimensional case. The two curves in Figure 5–4 are almost indistinguishable.

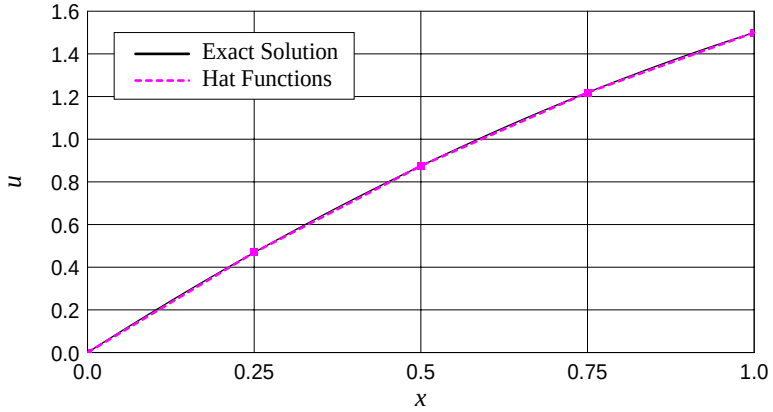


Fig. 5–4 Comparison of analytic and hat function solutions.

We have now obtained a multi-linear approximation to the solution.

Question 2: *Compare the hat function approach with the multiple term approach in Chapter 4. What are the respective advantages and disadvantages?*

5.4 Hints to Questions

Hint 1: Think of computational efficiency and smoothness of the function.

Chapter 6

A First Finite Element and Shape Functions

6.1 Objective

Our objective is to develop our first finite element approximation and to demonstrate a general method of obtaining element shape functions in natural coordinates. This is an extension of the hat functions developed in Chapter 5.

We first discuss the desirable characteristics of a numerical technique for analysis of general problems. We then derive the shape functions for an element and show the equivalence between global integration using hat functions and the summation of integrals over each element. We also introduce the concept of natural coordinates and demonstrate how to use Lagrange polynomials to derive the shape functions.

6.2 Desired Characteristics of an Effective Computational Method

As discussed in Reddy (*An Introduction to the Finite Element Method,* 2nd Edition, 1993), "an effective computational method should have the following characteristics:

- It should have a sound mathematical as well as physical basis (*i.e.*, yield convergent solutions and be applicable to practical problems).
- It should not have limitations with regard to the geometry, the physical composition of the domain, or the nature of the loading.
- The formulative procedure should be independent of the shape of the domain and the specific form of the boundary conditions.
- The method should be flexible enough to allow different degrees of approximation without reformulating the entire problem.

- It should involve a systematic procedure that can be automated for use on digital computers."

These criteria are met by the finite element method.

6.3 A Linear Finite Element

We will now begin our derivation of a simple finite element. Our first element will be a linear element with two nodes, Figure 6–1.

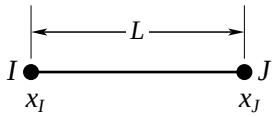


Fig. 6–1 Simple linear element.

We will assume that the displacement in the element is linear:

$$u = c_1 + c_2 x \tag{6–1}$$

Then, we can evaluate the displacements at the nodes using the coordinates of the nodes:

$$u_I = c_1 + c_2 x_I \tag{6–2}$$

$$u_J = c_1 + c_2 x_J \tag{6–3}$$

We can assemble in matrix form and solve:

$$\begin{bmatrix} 1 & x_I \\ 1 & x_J \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \end{Bmatrix} = \begin{Bmatrix} u_I \\ u_J \end{Bmatrix} \tag{6–4}$$

Solving:

$$\begin{Bmatrix} c_1 \\ c_2 \end{Bmatrix} = \frac{1}{x_J - x_I} \begin{bmatrix} x_J & -x_I \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} u_I \\ u_J \end{Bmatrix} \tag{6–5}$$

Substituting the solution back into eqn. (6–1), the displacement is now given by:

$$u = \frac{x_J u_I - x_I u_J}{x_J - x_I} + \frac{u_J - u_I}{x_J - x_I} x \tag{6–6}$$

Rearranging:

$$u = \left(\frac{x_J - x}{x_J - x_I} \right) u_I + \left(\frac{x - x_I}{x_J - x_I} \right) u_J \tag{6–7}$$

This is a simple, but important result. Given the nodal displacements (u_I and u_J) and a position on the element (x), we can interpolate to find the displacement at that point. The interpolation functions N_i are typically called "shape functions":

$$u = N_I u_I + N_J u_J \tag{6–8}$$

or

$$u = \sum_i N_i(x) u_i \tag{6–9}$$

These are plotted in Figure 6–2. Both are linear and both range from a value of 1 at the node they are associated with to 0 at the other node. They also sum to 1 at any x.

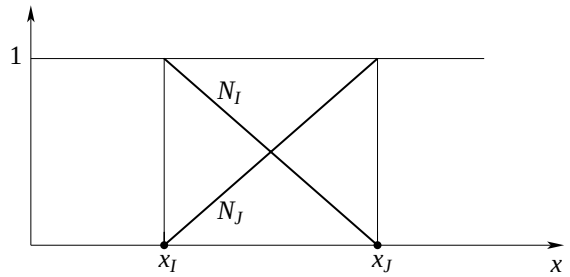


Fig. 6–2 Linear shape functions.

We can now see the connection between the shape functions and the hat functions we used in our previous lecture. They are in fact the same over the interior of the element, Figure 6–3.

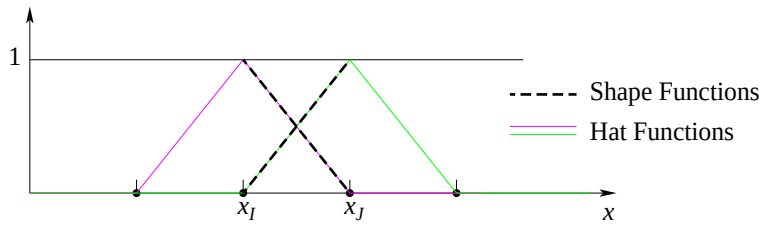
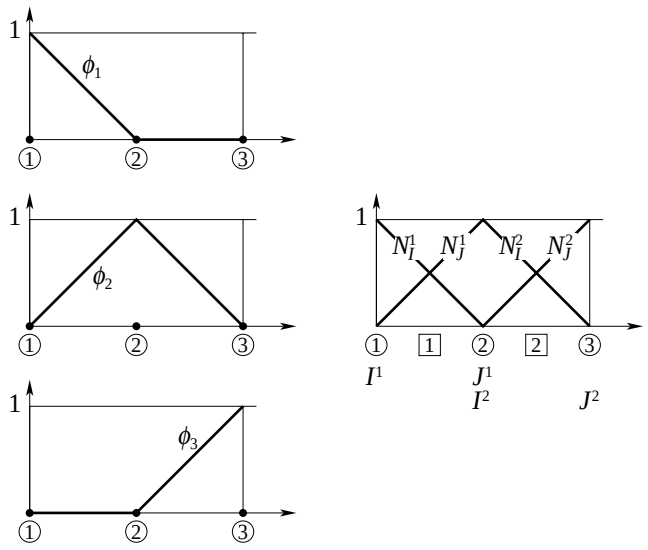


Fig. 6–3 Comparison of hat and shape functions.

Since the hat functions are only local, the only non-zero terms over the interval I to J occur with the two element shape functions. Therefore, integrating over each element and adding, give the same result as integrating over the entire length.

6.4 Equivalence Between Global Integration of Hat Functions and Summing the Integration of Element Shape Functions

With such a long title, this section must be significant. It is, in the sense that it shows that using hat functions and integrating globally is equivalent to integrating over each element, then summing the integrals. We will first show the global hat function and element shape function integrations schematically, Figure 6–4.



(a) Global hat functions. (b) Element shape functions.

Fig. 6–4 Global and element shape functions.

We have already derived the Galerkin approximation in Chapter 4:

$$\begin{bmatrix} AE\int_0^L \left(\frac{d\phi_1}{dx}\right)\left(\frac{d\phi_1}{dx}\right)dx\, AE\int_0^L \left(\frac{d\phi_1}{dx}\right)\left(\frac{d\phi_2}{dx}\right)dx\, AE\int_0^L \left(\frac{d\phi_1}{dx}\right)\left(\frac{d\phi_3}{dx}\right)dx \\ AE\int_0^L \left(\frac{d\phi_2}{dx}\right)\left(\frac{d\phi_1}{dx}\right)dx\, AE\int_0^L \left(\frac{d\phi_2}{dx}\right)\left(\frac{d\phi_2}{dx}\right)dx\, AE\int_0^L \left(\frac{d\phi_2}{dx}\right)\left(\frac{d\phi_3}{dx}\right)dx \\ AE\int_0^L \left(\frac{d\phi_3}{dx}\right)\left(\frac{d\phi_1}{dx}\right)dx\, AE\int_0^L \left(\frac{d\phi_3}{dx}\right)\left(\frac{d\phi_2}{dx}\right)dx\, AE\int_0^L \left(\frac{d\phi_3}{dx}\right)\left(\frac{d\phi_3}{dx}\right)dx \end{bmatrix} \cdot \begin{Bmatrix} a_1 \\ a_2 \\ a_n \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ f_3 \end{Bmatrix} \tag{6–10}$$

Our previous work has made us familiar with this approach and has demonstrated how well it works.

We now show that the same result is obtained by integrating element shape functions and then summing the integrals. For the element, the form of the integrals is the same as in eqn. (6–10). For element 1 ($I = 1$, $J = 2$) the integral is given by:

$$\begin{bmatrix} AE\int_0^{L^1}\left(\frac{dN_I^1}{dx}\right)\left(\frac{dN_I^1}{dx}\right)dx & AE\int_0^{L^1}\left(\frac{dN_I^1}{dx}\right)\left(\frac{dN_J^1}{dx}\right)dx \\ AE\int_0^{L^1}\left(\frac{dN_J^1}{dx}\right)\left(\frac{dN_I^1}{dx}\right)dx & AE\int_0^{L^1}\left(\frac{dN_J^1}{dx}\right)\left(\frac{dN_J^1}{dx}\right)dx \end{bmatrix} \begin{Bmatrix} u_I^1 \\ u_J^1 \end{Bmatrix} = \begin{Bmatrix} f_I^1 \\ f_J^1 \end{Bmatrix} \tag{6–11}$$

For element 2 ($I = 2$, $J = 3$) the integral is given by:

$$\begin{bmatrix} AE\int_0^{L^2}\left(\frac{dN_I^2}{dx}\right)\left(\frac{dN_I^2}{dx}\right)dx & AE\int_0^{L^2}\left(\frac{dN_I^2}{dx}\right)\left(\frac{dN_J^2}{dx}\right)dx \\ AE\int_0^{L^2}\left(\frac{dN_J^2}{dx}\right)\left(\frac{dN_I^2}{dx}\right)dx & AE\int_0^{L^2}\left(\frac{dN_J^2}{dx}\right)\left(\frac{dN_J^2}{dx}\right)dx \end{bmatrix} \begin{Bmatrix} u_I^2 \\ u_J^2 \end{Bmatrix} = \begin{Bmatrix} f_I^2 \\ f_J^2 \end{Bmatrix} \tag{6–12}$$

Here x is understood as measured locally from the left node of each element.

We now add the two element integrals together to assemble a global stiffness matrix. As this is done we must map from the element nodes (I, J) to the global nodes $(1, 2, 3)$. For instance, for element 1, $I = 1$ and $J = 2$. Thus the I, I integral term for element 1 will go in the 1,1 global coefficient, and the I, J integral term will go in the 1,2 global coefficient. Similarly, for element 2 where $I = 2$ and $J = 3$, the I, I term will go to the 2,2 global coefficient and the I, J term will go to the 2,3 coefficient. When all terms are summed, we get the matrix in (6-13). The reader should verify this.

$$\begin{bmatrix} AE\int_0^{L^1}\left(\frac{dN_I^1}{dx}\right)\left(\frac{dN_I^1}{dx}\right)dx & AE\int_0^{L^1}\left(\frac{dN_I^1}{dx}\right)\left(\frac{dN_J^1}{dx}\right)dx & 0 \\ AE\int_0^{L^1}\left(\frac{dN_J^1}{dx}\right)\left(\frac{dN_I^1}{dx}\right)dx & AE\int_0^{L^1}\left(\frac{dN_J^1}{dx}\right)\left(\frac{dN_J^1}{dx}\right)dx + AE\int_0^{L^2}\left(\frac{dN_I^2}{dx}\right)\left(\frac{dN_I^2}{dx}\right)dx & AE\int_0^{L^2}\left(\frac{dN_I^2}{dx}\right)\left(\frac{dN_J^2}{dx}\right)dx \\ 0 & AE\int_0^{L^2}\left(\frac{dN_J^2}{dx}\right)\left(\frac{dN_I^2}{dx}\right)dx & AE\int_0^{L^2}\left(\frac{dN_J^2}{dx}\right)\left(\frac{dN_J^2}{dx}\right)dx \end{bmatrix} \tag{6–13}$$

Again, x is understood as measured from the left node of each element.

Now, using Figure 6–4, the reader can see that eqns. (6–10) and (6–13) are equivalent. Because the hat and element shape functions are local, we can integrate over each element

and sum the element contributions. Thus, integration using global hat functions is equivalent to the summing integration performed over each element using the element shape functions. In the following lectures, we will show how to perform numerical integration for the integrals in the stiffness matrix and, ultimately, how to solve the system equations of FEM.

Question 1: *What are the advantages of using element shape functions over using the global hat functions?*

6.5 Hints to Questions

Hint 1: Unified expression; convenience in developing higher order interpolation.

Chapter 7

Generalized Matrix Form of the Finite Element Equations

7.1 Purpose

Our purpose is to first write the finite element equations using matrix notation. This represents a more formal transition from global approximations to the use of finite elements. We will then show how the element stiffness matrix and load vector are obtained. We will again use the one-dimensional elastic rod as our prototype for the derivation. We will then calculate the element stiffness matrix and load vector for a simple linear element.

7.2 Matrix Derivation of Equations — First Approach

This derivation gives us the final finite element form of our differential equation. A key concept is the use of element shape functions as approximating functions in the solution. In this section we provide a derivation that is not as explicit in defining the details of the use of the element shape functions. A more detailed alternate derivation is given in the next section.

We start from the weak statement of the differential equation (Chapter 4):

$$AE \int_0^L \frac{dw}{dx}\frac{du}{dx} = \int_0^L wbdx + w(L)P \tag{7–1}$$

with the boundary conditions:

$$u|_{x=0} = 0 \qquad \text{(an essential boundary condition)} \tag{7–2}$$

$$AE\frac{du}{dx}\bigg|_{x=L} = P \qquad \text{(a natural boundary condition)} \tag{7–3}$$

As before, we approximate the solution using a function in which the first known term (or terms) combines to satisfy the essential boundary conditions. The rest of the approximate solution consists of functions that are zero at the essential boundary conditions and are multiplied by coefficients whose values are to be determined.

We will represent the shape functions using N rather than ϕ, to show that these shape functions are only locally non-zero. Then:

$$u_n = N_0 u_0 + N_1 u_1 + N_2 u_2 + \cdots + N_n u_n \tag{7–4}$$

or:

$$u_n = N_0 u_0 + \lfloor \boldsymbol{N} \rfloor \{\boldsymbol{u}\} \tag{7–5}$$

We also need the derivative of the displacements:

$$\frac{du_n}{dx} = \frac{dN_0}{dx} u_0 + \left\lfloor \frac{d\boldsymbol{N}}{dx} \right\rfloor \{\boldsymbol{u}\}$$

or:

$$\frac{du_n}{dx} = B_0 u_0 + \lfloor \boldsymbol{B} \rfloor \{\boldsymbol{u}\} \tag{7–6}$$

where the $\lfloor \boldsymbol{B} \rfloor$ matrix contains the derivatives of the $\lfloor \boldsymbol{N} \rfloor$ matrix.

We will use the Galerkin method, so we use the same functions for the weighting function as we used to approximate the solution. As before, we do not include the term that satisfies the essential boundary condition. Then, the weighting function is given by:

$$w_n = \lfloor \boldsymbol{N} \rfloor \{\boldsymbol{d}\} \tag{7–7}$$

Note that since w_n is a scalar, we may also write:

$$w_n = \{\boldsymbol{d}\}^T \lfloor \boldsymbol{N} \rfloor^T \tag{7–8}$$

The derivative of the weighting function is given by:

$$\frac{dw_n}{dx} = \{\boldsymbol{d}\}^T \lfloor \boldsymbol{B} \rfloor^T \tag{7–9}$$

Substituting eqns. (7–6) and (7–9) into eqn. (7–1):

$$AE \int_0^L \{\boldsymbol{d}\}^T \lfloor \boldsymbol{B} \rfloor^T \left(B_0 u_0 + \lfloor \boldsymbol{B} \rfloor \{\boldsymbol{u}\}\right) dx = \int_0^L \{\boldsymbol{d}\}^T \lfloor \boldsymbol{N} \rfloor^T b dx + \{\boldsymbol{d}\}^T \lfloor \boldsymbol{N}(L) \rfloor^T P \tag{7–10}$$

Rearranging:

$$\{\boldsymbol{d}\}^T \left[AE \int_0^L \lfloor \boldsymbol{B} \rfloor^T \left(B_0 u_0 + \lfloor \boldsymbol{B} \rfloor \{\boldsymbol{u}\}\right) dx - \int_0^L \lfloor \boldsymbol{N} \rfloor^T b dx - \lfloor \boldsymbol{N}(L) \rfloor^T P \right] = 0 \tag{7–11}$$

Since this must hold true for all $\{\boldsymbol{d}\}$, then:

$$AE \int_0^L \lfloor \boldsymbol{B} \rfloor^T \left(B_0 u_0 + \lfloor \boldsymbol{B} \rfloor \{\boldsymbol{u}\} \right) dx - \int_0^L \lfloor \boldsymbol{N} \rfloor^T b dx - \lfloor \boldsymbol{N}(L) \rfloor^T P = 0 \tag{7–12}$$

or

$$AE \int_0^L \lfloor \boldsymbol{B} \rfloor^T \lfloor \boldsymbol{B} \rfloor \{\boldsymbol{u}\} dx = \int_0^L \lfloor \boldsymbol{N} \rfloor^T b dx + \lfloor \boldsymbol{N}(L) \rfloor^T P - AE \int_0^L \lfloor \boldsymbol{B} \rfloor^T B_0 u_0 dx \tag{7–13}$$

With respect to the integration, $\{\boldsymbol{u}\}$ is constant, so:

$$AE \int_0^L \lfloor \boldsymbol{B} \rfloor^T \lfloor \boldsymbol{B} \rfloor dx \{\boldsymbol{u}\} = \int_0^L \lfloor \boldsymbol{N} \rfloor^T b dx + \lfloor \boldsymbol{N}(L) \rfloor^T P - AE \int_0^L \lfloor \boldsymbol{B} \rfloor^T B_0 u_0 dx \tag{7–14}$$

The reader should note that we can still view these shape functions as global, so this gives us a global set of equations. If we write it in matrix form, this will be:

$$[\boldsymbol{K}]\{\boldsymbol{u}\} = \{\boldsymbol{f}\} \tag{7–15}$$

Here $[\boldsymbol{K}]$ is called the stiffness matrix, $\{\boldsymbol{u}\}$ is the unknown displacement vector, and $\{\boldsymbol{f}\}$ is the load vector.

In Chapter 6 we showed that, using shape functions that are only locally non-zero, we could perform the integration over individual elements and then sum the contribution of each element to obtain the same global matrix as given using global integration. If we use this approach, eqn. (7–14) becomes:

$$\begin{aligned} \sum_{e=1}^{\text{numel}} AE \int_0^{L_e} \lfloor \boldsymbol{B} \rfloor^T \lfloor \boldsymbol{B} \rfloor dx \{\boldsymbol{u}\} = & \sum_{e=1}^{\text{numel}} \int_0^L \lfloor \boldsymbol{N} \rfloor^T b dx + \sum_{e=1}^{\text{numel}_f} \lfloor \boldsymbol{N}(L) \rfloor^T P \\ & - \sum_{e=1}^{\text{numel}_u} AE \int_0^L \lfloor \boldsymbol{B} \rfloor^T B_0 u_0 dx \end{aligned} \tag{7–16}$$

Note that the boundary terms are only included for the elements with boundary conditions. In addition, there is an implied mapping from the local element numbering system to the global degrees of freedom.

7.3 Matrix Derivation of Equations — Second Approach

We start from the weak statement of the differential equation (Chapter 4):

$$AE \int_0^L \frac{dw}{dx} \frac{du}{dx} = \int_0^L wb dx + w(L)P \tag{7–17}$$

with the boundary conditions:

$$u|_{x=0} = 0 \qquad \text{(an essential boundary condition)} \tag{7–18}$$

$$AE\frac{du}{dx}\bigg|_{x=L} = P \qquad \text{(a natural boundary condition)} \tag{7–19}$$

As before, we approximate the solution using a function in which the first known term (or terms) combines to satisfy the essential boundary conditions. The rest of the approximate solution consists of functions that are zero at the essential boundary conditions and are multiplied by coefficients whose values are to be determined. In addition, we have shown that the undetermined coefficients equal the displacements at the nodal points. That is, the approximate solution is, for a total of m nodes:

$$u^{(m)} = u_0\phi_0 + u_1\phi_1 + u_2\phi_2 + \cdots + u_m\phi_m \tag{7–20}$$

Recall that in Chapter 6 we introduced the element *shape functions*. For the linear shape functions, we had shown that they are in fact split expressions of hat functions. For quadratic and other more general shape functions that we will discuss in Chapter 8, the analogy remains that shape functions can be considered as split expressions for some localized functions. Here we consider ϕ_i as hat functions, and each hat function is split into two halves, each occupying only the region of one element. This way, the approximate solution in eqn. (7–20) can be written as:

$$\begin{aligned} u^{(m)} &= u_0\phi_0 + u_1N_1^1 + u_2(N_2^1 + N_2^2) + u_3(N_3^2 + N_3^3) + \cdots + u_m\phi_m \\ &= u_0\phi_0 + \left[u_1N_1^1 + u_2N_2^1\right] + \left[u_2N_2^2 + u_2N_3^2\right] + \cdots \end{aligned} \tag{7–21}$$

where N_i^e denotes the shape function for element e at node i. In the last equation, we have reorganized terms by elements. Similar splitting and regrouping can be done for other shape functions, and the corresponding localized global functions ϕ_i.

Within an element, the displacement is approximated by shape functions of order n as

$$u(x) = \sum_i^N N_i u_i = \lfloor \boldsymbol{N}^e \rfloor \{\boldsymbol{u}^e\} \tag{7–22}$$

where we use $\lfloor\ \rfloor$ to denote a row matrix, and $\{\ \}$ to denote a column matrix. The superscript e denotes an element. Equation (7–20) can be re-written as

$$u^{(m)} = u_0\phi_0 + \left\lfloor \boldsymbol{N}^1 \right\rfloor \left\{\boldsymbol{u}^1\right\} + \left\lfloor \boldsymbol{N}^2 \right\rfloor \left\{\boldsymbol{u}^2\right\} + \cdots = u_0\phi_0 + \sum_{e=1}^{N_{\text{elm}}} \lfloor \boldsymbol{N}^e \rfloor \{\boldsymbol{u}^e\} \tag{7–23}$$

where N_{elm} denotes the number of elements in the model. (Note: Equation (7–23) is more general than eqn. (7–20). Equation (7–20) should be viewed as a particular example when linear shape functions are used. Equations (7–22) and (7–23) allow the use of general shape functions.)

We also need the derivative of the displacements:

$$\frac{du^{(m)}}{dx} = \frac{d\phi_0}{dx}u_0 + \sum_{e=1}^{N_{\text{elm}}} \left\lfloor \frac{d\boldsymbol{N}^e}{dx} \right\rfloor \{\boldsymbol{u}^e\} \tag{7–24}$$

or

$$\frac{du^{(m)}}{dx} = B_0 u_0 + \sum_{e=1}^{N_{\text{elm}}} \lfloor \boldsymbol{B}^e \rfloor \{\boldsymbol{u}^e\} \tag{7–25}$$

where the $\lfloor \boldsymbol{B}^e \rfloor$ matrix contains the derivatives of the $\lfloor \boldsymbol{N}^e \rfloor$ matrix.

We will use the Galerkin method, so we use the same functions for the weighting function as we used to approximate the solution. As before, we do not include the term that satisfies the essential boundary condition. Then, the weighting function is given by:

$$w^{(m)} = \sum_{e=1}^{N_{\text{elm}}} \lfloor \boldsymbol{N}^e \rfloor \{\boldsymbol{d}^e\} \tag{7–26}$$

Note that since $w^{(m)}$ is a scalar, we may also write:

$$w^{(m)} = \sum_{e=1}^{N_{\text{elm}}} \{\boldsymbol{d}^e\}^T \lfloor \boldsymbol{N}^e \rfloor^T \tag{7–27}$$

The derivative of the weighting function is given by:

$$\frac{dw^{(m)}}{dx} = \sum_{e=1}^{N_{\text{elm}}} \{\boldsymbol{d}^e\}^T \lfloor \boldsymbol{B}^e \rfloor^T \tag{7–28}$$

Substituting eqns. (7–25) and (7–28) into eqn. (7–17):

$$\begin{aligned} AE \int_0^L \left(\sum_{e=1}^{N_{\text{elm}}} \{\boldsymbol{d}^e\}^T \lfloor \boldsymbol{B}^e \rfloor^T \right) \left(B_0 u_0 + \sum_{e=1}^{N_{\text{elm}}} \lfloor \boldsymbol{B}^e \rfloor \{\boldsymbol{u}^e\} \right) dx \\ = \int_0^L \sum_{e=1}^{N_{\text{elm}}} \{\boldsymbol{d}^e\}^T \lfloor \boldsymbol{N}^e \rfloor^T b dx + \sum_{e=1}^{N_{\text{elm}}} \{\boldsymbol{d}^e\}^T \lfloor \boldsymbol{N}^e(L) \rfloor^T P \end{aligned} \tag{7–29}$$

or

$$\begin{aligned} AE \int_0^L \left(\sum_{e=1}^{N_{\text{elm}}} \{\boldsymbol{d}^e\}^T \lfloor \boldsymbol{B}^e \rfloor^T \right) \left(\sum_{e=1}^{N_{\text{elm}}} \lfloor \boldsymbol{B}^e \rfloor \{\boldsymbol{u}^e\} \right) dx = \int_0^L \sum_{e=1}^{N_{\text{elm}}} \{\boldsymbol{d}^e\}^T \lfloor \boldsymbol{N}^e \rfloor^T b dx \\ + \sum_{e=1}^{N_{\text{elm}}} \{\boldsymbol{d}^e\}^T \lfloor \boldsymbol{N}^e(L) \rfloor^T P - AE \int_0^L \left(\sum_{e=1}^{N_{\text{elm}}} \{\boldsymbol{d}^e\}^T \lfloor \boldsymbol{B}^e \rfloor^T \right) B_0 u_0 dx \end{aligned} \tag{7–30}$$

Recall that in Chapter 6, by using shape functions that are only non-zero within each element, we have shown that any product of a shape function of one element with any shape function for a different element is zero. The two summations on the left-hand side of eqn. (7–30) effectively become just one summation:

$$AE\int_0^L\left(\sum_{e=1}^{N_{\text{elm}}}\{\boldsymbol{d}^e\}^T\lfloor\boldsymbol{B}^e\rfloor^T\lfloor\boldsymbol{B}^e\rfloor\{\boldsymbol{u}^e\}\right)dx=\int_0^L\sum_{e=1}^{N_{\text{elm}}}\{\boldsymbol{d}^e\}^T\lfloor\boldsymbol{N}^e\rfloor^T\,b dx$$
$$+\sum_{e=1}^{N_{\text{elm}}}\{\boldsymbol{d}^e\}^T\lfloor\boldsymbol{N}^e(L)\rfloor^T\,P-AE\int_0^L\left(\sum_{e=1}^{N_{\text{elm}}}\{\boldsymbol{d}^e\}^T\lfloor\boldsymbol{B}^e\rfloor^T\right)B_0u_0dx \quad (7\text{–}31)$$

Also recall that in Chapter 6, by using shape functions that are only non-zero within each element, we showed that integration over the entire problem domain is equivalent to a summation of integrations over individual elements. The effectively changes the integrations in eqn. (7–31) to become integrating over individual elements, and the summations be performed after the integrations:

$$\sum_{e=1}^{N_{\text{elm}}}\left(AE\int_0^{L^e}\{\boldsymbol{d}^e\}^T\lfloor\boldsymbol{B}^e\rfloor^T\lfloor\boldsymbol{B}^e\rfloor dx\{\boldsymbol{u}^e\}\right)=\sum_{e=1}^{N_{\text{elm}}}\left(\int_0^{L^e}\{\boldsymbol{d}^e\}^T\lfloor\boldsymbol{N}^e\rfloor^T\,b dx\right)$$
$$+\sum_{e=1}^{N_f}\left(\{\boldsymbol{d}^e\}^T\lfloor\boldsymbol{N}^e(L)\rfloor^T\,P\right)-\sum_{e=1}^{N_u}\left(AE\int_0^{L^e}\{\boldsymbol{d}^e\}^T\lfloor\boldsymbol{B}^e\rfloor^T\right)B_0u_0dx \quad (7\text{–}32)$$

where in writing eqn. (7–32), we took notice that the boundary conditions (on the right hand side) only involve the elements that actually contains the boundaries; and different type of boundary conditions involve different number of elements.

The primary unknowns for the problem are the nodal displacements. We can arrange the unknowns into a global matrix. By doing so, the summations in eqn. (7–32) also involve (imply) the assemblage process, and the results can be written as

$$\{\boldsymbol{d}\}^T\left[\sum_{e=1}^{N_{\text{elm}}}\left(\int_0^{L^e}AE\lfloor\boldsymbol{B}^e\rfloor^T\lfloor\boldsymbol{B}^e\rfloor dx\right)\{\boldsymbol{u}\}-\sum_{e=1}^{N_{\text{elm}}}\int_0^{L^e}\lfloor\boldsymbol{N}^e\rfloor^T\,b dx\right.$$
$$\left.-\sum_{e=1}^{N_f}\lfloor\boldsymbol{N}^e(L)\rfloor^T\,P+\sum_{e=1}^{N_u}AE\int_0^{L^e}\lfloor\boldsymbol{B}^e\rfloor^T\,B_0u_0dx\right]=0 \quad (7\text{–}33)$$

Since this must hold true for all $\{\boldsymbol{d}\}$, then:

$$\left(\sum_{e=1}^{N_{\text{elm}}}AE\int_0^{L^e}\lfloor\boldsymbol{B}^e\rfloor^T\lfloor\boldsymbol{B}^e\rfloor dx\right)\{\boldsymbol{u}\}=\sum_{e=1}^{N_{\text{elm}}}\int_0^{L^e}\lfloor\boldsymbol{N}^e\rfloor^T\,b dx$$
$$+\sum_{e=1}^{N_f}\lfloor\boldsymbol{N}^e(L)\rfloor^T\,P-\sum_{e=1}^{N_u}AE\int_0^{L^e}\lfloor\boldsymbol{B}^e\rfloor^T\,B_0u_0dx \quad (7\text{–}34)$$

If we write it in matrix form, this will be:

$$[\boldsymbol{K}]\{\boldsymbol{u}\} = \{\boldsymbol{f}\} \tag{7–35}$$

Here $[\boldsymbol{K}]$ is called the stiffness matrix, $\{\boldsymbol{u}\}$ is the unknown displacement vector, and $\{\boldsymbol{f}\}$ is the load vector.

7.4 The Element Stiffness Matrix

Let us now look at an individual element stiffness matrix:

$$[\boldsymbol{K}^e] = AE \int_0^{L^e} \lfloor \boldsymbol{B} \rfloor^T \lfloor \boldsymbol{B} \rfloor \, dx \tag{7–36}$$

In Chapter 6 we derived the 1-D linear element shape functions:

$$\lfloor \boldsymbol{N} \rfloor = \begin{bmatrix} \dfrac{x_J - x}{x_J - x_I} & \dfrac{x - x_I}{x_J - x_I} \end{bmatrix} \tag{7–37}$$

Taking the derivatives:

$$\lfloor \boldsymbol{B} \rfloor = \begin{bmatrix} \dfrac{-1}{L} & \dfrac{1}{L} \end{bmatrix} \tag{7–38}$$

where $L = x_J - x_I$, the length of the element.

Now, substitute eqn. (7–38) into eqn. (7–36), to obtain:

$$[\boldsymbol{K}^e] = AE \int_0^{L^e} \begin{Bmatrix} \dfrac{-1}{L} \\ \dfrac{1}{L} \end{Bmatrix} \begin{bmatrix} \dfrac{-1}{L} & \dfrac{1}{L} \end{bmatrix} dx \tag{7–39}$$

multiplying

$$[\boldsymbol{K}^e] = AE \int_0^{L^e} \begin{bmatrix} \dfrac{1}{L^2} & \dfrac{-1}{L^2} \\ \dfrac{-1}{L^2} & \dfrac{1}{L^2} \end{bmatrix} dx \tag{7–40}$$

simplifying

$$[\boldsymbol{K}^e] = \frac{AE}{L^2} \int_0^{L^e} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} dx \tag{7–41}$$

and finally, integrating:

$$[\boldsymbol{K}^e] = \frac{AE}{L^2} \begin{bmatrix} L & -L \\ -L & L \end{bmatrix} \tag{7–42}$$

or:

$$[\boldsymbol{K}^e] = \frac{AE}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{7–43}$$

This is our old friend. Go back to Chapter 1, where we developed the stiffness matrix for a spring using intuition. That matrix was exactly the same as eqn. (7–43). It all makes sense.

We now look at the right hand side of eqn. (7–14).

$$\{\boldsymbol{f}^e\} = \int_0^{L^e} \lfloor \boldsymbol{N} \rfloor^T b dx + \lfloor \boldsymbol{N}(L) \rfloor^T P + AE \int_0^{L^e} \lfloor \boldsymbol{B} \rfloor^T B_0 u_0 dx \tag{7–44}$$

We will look at the first term on the right hand side:

$$\{\boldsymbol{f}^e\}_{\text{Body Loads}} = \int_0^{L^e} \lfloor \boldsymbol{N} \rfloor^T b dx \tag{7–45}$$

Substituting for the shape functions gives:

$$\{\boldsymbol{f}^e\}_{\text{Body Loads}} = \int_0^{L^e} \begin{Bmatrix} \dfrac{x_J - x}{x_J - x_I} \\ \dfrac{x - x_I}{x_J - x_I} \end{Bmatrix} b dx \tag{7–46}$$

or,

$$\{\boldsymbol{f}^e\}_{\text{Body Loads}} = \frac{b}{L} \int_0^{L^e} \begin{Bmatrix} x_J - x \\ x - x_I \end{Bmatrix} dx \tag{7–47}$$

After integrating, we obtain:

$$\{\boldsymbol{f}^e\}_{\text{Body Loads}} = bL \begin{Bmatrix} \frac{1}{2} \\ \frac{1}{2} \end{Bmatrix} \tag{7–48}$$

That is, for a constant body force, the equivalent nodal loads each equal to one-half the total load. If the body load was not constant, we could use the shape functions to interpolate the body load and we would obtain different equivalent nodal loads.

The second term of eqn. (7–14) is:

$$\{\boldsymbol{f}^e\}_{\text{Natural BC}} = \lfloor \boldsymbol{N}(L) \rfloor^T P \tag{7–49}$$

This term is only active on the boundaries with natural boundary conditions. In addition, since we are using shape functions that evaluate to 1 at the nodes, eqn. (7–49) turns out to be a statement that the loads at the nodes are directly included in the load vector in the appropriate global locations.

The last term of eqn. (7–14) is:

$$\{\boldsymbol{f}^e\}_{\text{Essential BC}} = AE \int_0^{L^e} \lfloor \boldsymbol{B} \rfloor^T B_0 u_0 dx \tag{7–50}$$

This term is just like the element stiffness matrix. In fact, using element shape functions allows us to evaluate the entire element stiffness matrix and then move the terms associated with known displacements to the right hand side. In this case, since the essential boundary condition is zero, this will not contribute to the load vector.

We now have completed a discussion of how we can assemble the global stiffness matrix by summing the individual element matrices.

Chapter 8

More About Shape Functions

8.1 Purpose

Our purpose is to demonstrate a general method of obtaining element shape functions in natural coordinates.

We first derive shape functions directly, then present the general equation for Lagrange polynomials.

8.2 Shape Functions in Natural Coordinates

8.2.1 Linear Shape Functions

In Chapter 6, we derived shape functions for a linear element in the actual element coordinates:

$$\lfloor \boldsymbol{N} \rfloor = \begin{bmatrix} \dfrac{x_J - x}{x_J - x_I} & \dfrac{x - x_I}{x_J - x_I} \end{bmatrix} \tag{8–1}$$

We will now introduce the idea of *natural coordinates*. The natural coordinates will run from -1 to 1. That is, in the natural coordinate system, $\zeta_I = -1$ and $\zeta_J = 1$.

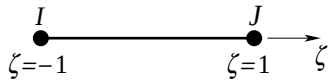


Fig. 8–1 Linear element in natural coordinate system.

The natural coordinate system can be understood as a local coordinate system. It is a convenient way to perform numerical integration. We will use a mapping from the natural

coordinates to the real coordinates for the element. In one-dimensional system, there is a very simple relation between the actual coordinates and the natural coordinates:

In the natural coordinate system, eqn. (8–1) becomes:

$$\lfloor \boldsymbol{N} \rfloor = \left\lfloor \begin{array}{cc} \frac{1-\zeta}{2} & \frac{\zeta+1}{2} \end{array} \right\rfloor \tag{8–2}$$

As illustrated in Figure 8–2, these shape functions have a value of 1 at the associated node and 0 at the other node. They interpolate linearly.

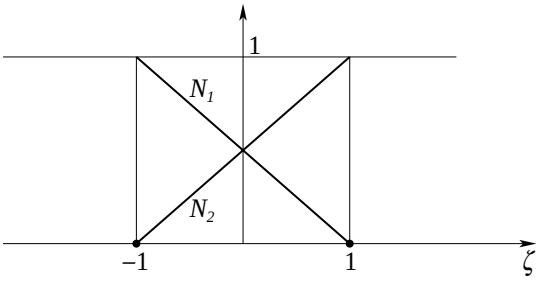


Fig. 8–2 Linear shape functions.

Question 1: *What are the advantages of using the natural coordinates?*

8.2.2 Quadratic Shape Functions

If we use quadratic interpolation over an element, we have three coefficients to solve for, so we use three nodes (Figure 8–3).

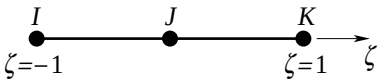


Fig. 8–3 Quadratic element.

As for the linear element (Chapter 6) we assume a quadratic interpolation function:

$$u = c_1 + c_2\zeta + c_3\zeta^2 \tag{8–3}$$

Since the displacements at the three nodes are known, we can write eqn. (8–3) at each of the nodes:

$$\begin{aligned} u_I &= c_1 + c_2(-1) + c_3(-1)^2 \\ u_J &= c_1 + c_2(0) + c_3(0) \\ u_K &= c_1 + c_2(1) + c_3(1) \end{aligned}$$

or in matrix form,

$$\begin{bmatrix} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \end{Bmatrix} = \begin{Bmatrix} u_I \\ u_J \\ u_K \end{Bmatrix} \tag{8–4}$$

Solving:

$$\begin{Bmatrix} c_1 \\ c_2 \\ c_3 \end{Bmatrix} = \begin{Bmatrix} u_J \\ \frac{1}{2}(u_I - 2u_J + u_K) \\ \frac{1}{2}(u_K - u_I) \end{Bmatrix} \tag{8–5}$$

Substituting into eqn. (8–3) gives:

$$u = u_J + \frac{1}{2}(u_I - 2u_J + u_K)\,\zeta + \frac{1}{2}(u_K - u_I)\,\zeta^2 \tag{8–6}$$

Finally, collecting terms gives:

$$u = \frac{\zeta}{2}(\zeta - 1)u_I + \left(1 - \zeta^2\right)u_J + \frac{\zeta}{2}(\zeta + 1)\,u_K \tag{8–7}$$

or,

$$u = N_I u_I + N_J u_J + N_K u_K \tag{8–8}$$

We have now derived the quadratic shape functions. They are plotted in Figure 8–4.

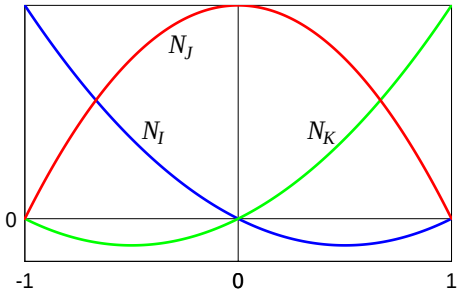


Fig. 8–4 Quadratic shape functions.

8.3 Lagrange Polynomials

Interpolation functions obtained using the dependent unknown (not its derivatives) are Lagrange interpolation functions. These are the functions we use for element shape functions. They have the properties (Reddy) that they have a value of 1 at their corresponding node and 0 at all other nodes and that they sum to 1 everywhere:

$$N_i(\zeta_j) = \delta_{ij} = \begin{cases} 1 & \text{if} \quad i = j \\ 0 & \text{if} \quad i \neq j \end{cases} \tag{8–9}$$

$$\sum_i N_i(\zeta) = 1 \tag{8–10}$$

We have already observed these same properties in the linear and quadratic shape functions we have derived directly. The shape function is equal to 1 at its associated node and 0 at the other nodes. Thus the coefficient multiplying the shape function is the actual value at a node. Also, the sum of all the shape functions at any position is 1. As a result, if the displacements at the nodes are the same, the values between the nodes will also be constant. These conditions are necessary for convergence. Physically, this means that if we assign constant values to each of the nodes (for example a uniform rigid body displacement) then everywhere between the nodes will also have the same displacement.

In general, a series of Lagrange polynomials of order are derived for nodes. We first define a function that is zero at all nodes except the i-th node:

$$N_i = c_i\,(\zeta - \zeta_1)\,(\zeta - \zeta_2) \cdot \cdots \cdot (\zeta - \zeta_{i-1})\,(\zeta - \zeta_{i+1}) \cdot \cdots \cdot (\zeta - \zeta_n) \tag{8–11}$$

where ζ_k denotes the value of ζ at that node k. In addition, we force this function to equal 1 at node i:

$$1 = c_i\,(\zeta_i - \zeta_1)\,(\zeta_i - \zeta_2) \cdot \cdots \cdot (\zeta_i - \zeta_{i-1})\,(\zeta_i - \zeta_{i+1}) \cdot \cdots \cdot (\zeta_i - \zeta_n) \tag{8–12}$$

We then solve eqn. (8–12) for c_i and substitute into eqn. (8–11) to get:

$$N_i = \frac{(\zeta - \zeta_1)\,(\zeta - \zeta_2) \cdot \cdots \cdot (\zeta - \zeta_{i-1})\,(\zeta - \zeta_{i+1}) \cdot \cdots \cdot (\zeta - \zeta_n)}{(\zeta_i - \zeta_1)\,(\zeta_i - \zeta_2) \cdot \cdots \cdot (\zeta_i - \zeta_{i-1})\,(\zeta_i - \zeta_{i+1}) \cdot \cdots \cdot (\zeta_i - \zeta_n)} \tag{8–13}$$

which is the general form of Lagrange polynomial that can be used as shape functions.

As an illustration of the use of eqn. (8–13), let us obtain the shape functions for $n = 3$. Then, for node 1:

$$N_1(\zeta) = \frac{(\zeta - \zeta_2)(\zeta - \zeta_3)}{(\zeta_1 - \zeta_2)(\zeta_1 - \zeta_3)} \tag{8–14}$$

$$N_1(\zeta) = \frac{(\zeta - 0)(\zeta - 1)}{(-1 - 0)(-1 - 1)} = \frac{\zeta}{2}(\zeta - 1) \tag{8–15}$$

Similarly at node 2:

$$N_2(\zeta) = \frac{(\zeta - \zeta_1)(\zeta - \zeta_3)}{(\zeta_2 - \zeta_3)(\zeta_2 - \zeta_3)} \tag{8–16}$$

$$N_2(\zeta) = \frac{(\zeta + 1)(\zeta - 1)}{(0 - (-1))(0 - 1)} = \left(1 - \zeta^2\right) \tag{8–17}$$

and

$$N_2(\zeta) = \frac{\zeta}{2}(\zeta + 1) \tag{8–18}$$

As expected, these shape functions are the same as we derived directly.

The method of deriving the shape function from Lagrange polynomials can be extended to 2 or 3 dimensions. We will work on that in future chapters.

8.4 Element Formulations in Natural Coordinates

In Chapter 6 we derived the element stiffness matrix using shape functions in the true coordinate system.

$$[\boldsymbol{K}^e] = AE \int_0^{L^e} \lfloor \boldsymbol{B}^e \rfloor^T \lfloor \boldsymbol{B}^e \rfloor \, dx \tag{8–19}$$

We shall demonstrate how to use natural coordinates in calculating the element stiffness matrix. The shape functions in the natural coordinate system are:

$$\lfloor \boldsymbol{N} \rfloor = \begin{bmatrix} \frac{1}{2}(1 - \zeta) & \frac{1}{2}(1 + \zeta) \end{bmatrix} \tag{8–20}$$

we use the shape functions to interpolate both the geometry and the solution. This is called an *isoparametric element*. It is permissible to use different shape functions for geometry and the solution, but isoparametric elements are the most common approach.

Then:

$$x = \lfloor \boldsymbol{N} \rfloor \{\boldsymbol{x}\}$$

where $\{\boldsymbol{x}\}$ is the nodal coordinates, and

$$x = \frac{1}{2}(1 - \zeta)x_I + \frac{1}{2}(1 + \zeta)x_J \tag{8–21}$$

and

$$\frac{dx}{d\zeta} = \frac{1}{2}(-1)x_I + \frac{1}{2}(1)x_J = \frac{1}{2}(x_J - x_I) = \frac{L^e}{2} \tag{8–22}$$

Since

$$\lfloor \boldsymbol{B} \rfloor = \left\lfloor \frac{d\boldsymbol{N}}{dx} \right\rfloor$$

We need to obtain the derivatives of the shape functions with respect to the actual coordinate x. Since the shape functions are functions of the natural coordinate, we must use the chain rule and then substitute (8-22). We will see that in 2D or 3D, the relationships between the global and natural coordinates will not be a simple constant, but will be a matrix. The transformation between coordinate systems is called the Jacobian.

$$\left\lfloor \frac{d\boldsymbol{N}}{dx} \right\rfloor \frac{dx}{d\zeta} = \left\lfloor \frac{d\boldsymbol{N}}{d\zeta} \right\rfloor \tag{8–23}$$

and

$$\left\lfloor \frac{d\boldsymbol{N}}{dx} \right\rfloor = \left(\frac{dx}{d\zeta}\right)^{-1} \frac{d\lfloor \boldsymbol{N} \rfloor}{d\zeta} = \frac{2}{L^e} \left\lfloor \frac{d\boldsymbol{N}}{d\zeta} \right\rfloor$$
$$\lfloor \boldsymbol{B} \rfloor = \left\lfloor \frac{d\boldsymbol{N}}{dx} \right\rfloor = \frac{2}{L^e} \left\lfloor \frac{-1}{2} \quad \frac{1}{2} \right\rfloor = \left\lfloor \frac{-1}{L^e} \quad \frac{1}{L^e} \right\rfloor \tag{8–24}$$

Then, changing the bounds of our integration to the natural coordinates bounds and integrating with respect to the natural coordinates,

$$\begin{aligned} [\boldsymbol{K}^e] &= \frac{AE}{L^{e2}} \int_0^{L^e} \begin{Bmatrix} -1 \\ 1 \end{Bmatrix} \begin{bmatrix} -1 & 1 \end{bmatrix} dx \\ &= \frac{AE}{L^{e2}} \int_{-1}^{1} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \frac{dx}{d\zeta} d\zeta = \frac{AE}{2L^e} \int_{-1}^{1} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} d\zeta \end{aligned} \tag{8–25}$$

and the integration gives

$$[\boldsymbol{K}^e] = \frac{AE}{2L^e} \int_{-1}^{1} \begin{bmatrix} 2 & -2 \\ -2 & 2 \end{bmatrix} = \frac{AE}{L^e} \int_{-1}^{1} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{8–26}$$

which is the same as our previous result. The same results for the load vectors can also be obtained using the shape functions in terms of natural coordinate.

8.5 Hints to Questions

Hint 1: Unified expression; convenience in developing higher order interpolation.

Chapter 9

Numerical Integration in 1D

9.1 Purpose

Our purpose is to show how we can integrate the element stiffness matrices numerically. In this chapter we present only 1D integration, but as we will see in the future, the same approach can be used for 2d and 3D.

In general, we will not integrate element stiffness matrices directly because, in most cases, there is no simple closed form integral. Since numerical integration formulae are given in natural coordinates, we need to map our real coordinates to the natural coordinates. We will then demonstrate integration in the natural coordinate system. We derive the mapping first, then implement numerical integration.

9.2 Change of Variables

From calculus, the change of variables formula is:

$$\int_a^b f(x)dx = \int_{\zeta_1}^{\zeta_2} f(g(\zeta))\frac{dg(\zeta)}{d\zeta}d\zeta \tag{9–1}$$

where:

$$x = g(\zeta) \qquad \text{and} \qquad g(\zeta_1) = a, \quad g(\zeta_2) = b$$

Here we use our 1-D example as the example to illustrate how the integration is done. We want to integrate the element stiffness matrix (which was computed analytically using both the real and natural coordinates at the end of Chapter 7). We want to integrate the

element stiffness matrix. The relation between the stiffness matrix in real coordinates and the natural coordinates is given by:

$$[\boldsymbol{K}^e] = AE \int_0^{L^e} \lfloor \boldsymbol{B} \rfloor^T \lfloor \boldsymbol{B}(\zeta) \rfloor \, dx = AE \int_{-1}^{1} \lfloor \boldsymbol{B}(\zeta) \rfloor^T \lfloor \boldsymbol{B}(\zeta) \rfloor \frac{dx}{d\zeta} d\zeta \tag{9–2}$$

We have already derived the shape functions and their derivatives in terms of the natural coordinate, but we still need to define the relationship between x and ζ. We do this using the same shape functions we used for interpolating the displacement. For a linear (two-noded) one-dimensional element:

$$x = N_I x_I + N_J x_J \tag{9–3}$$

where the shape functions are

$$N_I = \frac{1}{2}(1 - \zeta)$$
$$N_J = \frac{1}{2}(1 + \zeta)$$

Then:

$$\frac{dx}{d\zeta} = \frac{dN_I}{d\zeta} x_I + \frac{dN_J}{d\zeta} x_J \tag{9–4}$$

Substituting the shape functions gives

$$\frac{dx}{d\zeta} = \frac{d}{d\zeta}\left[\frac{1}{2}(1 - \zeta)\right] x_I + \frac{d}{d\zeta}\left[\frac{1}{2}(1 + \zeta)\right] x_J$$

Simplifying:

$$\frac{dx}{d\zeta} = \left[\frac{1}{2}(-1)\right] x_I + \left[\frac{1}{2}(1)\right] x_J = \frac{L^e}{2} \tag{9–5}$$

To integrate the element stiffness matrix of eqn. (9–2), we first need to obtain an expression for the derivative (with respect to x) of the shape functions expressed in natural coordinates. For example, we will illustrate this using the first shape function:

$$\frac{dN_I(\zeta)}{d\zeta} = \frac{dN_I(\zeta)}{dx} \frac{dx}{d\zeta} \tag{9–6}$$

We can evaluate the left hand side and the second term on the right hand side of eqn. (9–6):

$$\left[\frac{1}{2}(-1)\right] = \frac{dN_I(\zeta)}{dx} \frac{L^e}{2}$$

or:

$$\frac{dN_I(\zeta)}{dx} = \left(\frac{L^e}{2}\right)^{-1}\left(\frac{-1}{2}\right) = \frac{-1}{L^e} \tag{9–7}$$

We will look at the first term in the matrix:

$$AE\int_{-1}^{1} B_I(\zeta)B_I(\zeta)\frac{dx}{d\zeta}d\zeta = AE\int_{-1}^{1}\left(\frac{-1}{L^e}\right)\left(\frac{-1}{L^e}\right)\frac{L^e}{2} = \frac{AE}{L^e} \tag{9–8}$$

This is the same value we calculated in Chapter 8. Although a simple example, the same concept applies to more complicated integrals.

Question 1: *What are the possible advantages of changing variables in calculating an integral?*

9.3 Gauss-Legendré Integration

Numerical integration involves calculating a function at selected locations, multiplying the value by a weight, and then summing to obtain the integral.

$$\int_{-1}^{1} f(\zeta)d\zeta \approx \sum_{i=1}^{N_{GP}} w_i f(\zeta_i) \tag{9–9}$$

Let us illustrate. Figure 9–1 shows a linear function over the interval −1 to 1.

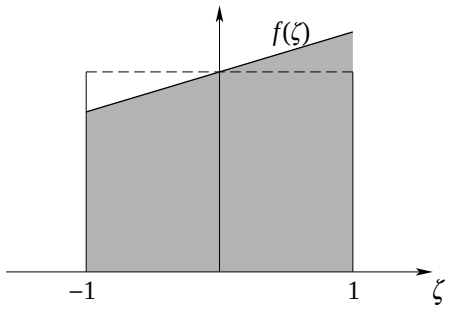


Fig. 9–1 Integration of a linear function.

Clearly, the linear function can be evaluated exactly using one point:

$$\int_{-1}^{1} f(\zeta)d\zeta = \sum_{i=1}^{1} w_i f(\zeta_i) = 2f(0) \tag{9–10}$$

That is, if we evaluate the function at $\zeta = 0$ and multiply that value by 2, we will obtain the exact integral.

If the function is of higher order than linear, the one-point numerical integration is only approximate. We can increase the accuracy by selecting more integration points. Many numerical integration schemes use evenly spaced evaluation locations such as the Simpson's formula and the Newton-Cotes formula. The *Gauss-Legendré integration* (also called as *Gaussian quadrature*), however, evaluates the integral at cleverly picked locations and weights leading to high accuracy at low cost. This is sketched in Figure 9–2.

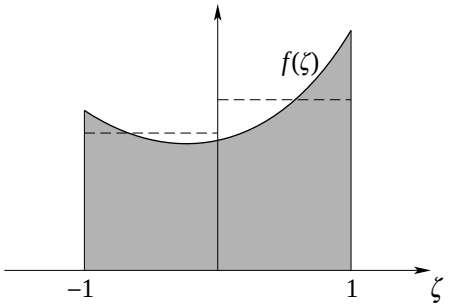


Fig. 9–2 Sketch showing integration of higher order curve.

We will derive the equations for Guass-Legendré integration for a cubic polynomial. To do this, we let both the weights and the locations at which we will evaluate the function be unknown. We also want to know how many points we need in order to obtain an exact integration. Let us assume that our integrand, a cubic polynomial, can be expressed as

$$f(\zeta) = a + b\zeta + c\zeta^2 + d\zeta^3 \tag{9–11}$$

where a, b, c and d are arbitrary constants which in general are non-zero. Assume we would need n Gauss points to achieve an exact integration. The exact integration would be

$$\int_{-1}^{1} f(\zeta)d\zeta = a\zeta\big|_{-1}^{1} + \frac{1}{2}b\,\zeta^2\Big|_{-1}^{1} + \frac{1}{3}b\,\zeta^3\Big|_{-1}^{1} + \frac{1}{4}b\,\zeta^4\Big|_{-1}^{1} = 1a + 0 + \frac{2}{3}c + 0 \tag{9–12}$$

A numerical integration using n Gauss points, called n-point Gaussian integral, would give

$$\int_{-1}^{1} f(\zeta)d\zeta = \sum_{i=1}^{n} w_i f(\zeta_i) = \sum_{i=1}^{n} w_i \left(a + b\zeta + c\zeta^2 + d\zeta^3\right) \tag{9–13}$$

We require that eqn. (9–13) be exact for any polynomial up to degree 3. That is, for any coefficients a, b, c, and d, eqns. (9–12) and (9–13) should give the same result. This leads to following requirements for the n pairs of weights (w_i) and the locations (ζ_i):

$$2 = \sum_{i=1}^{n} w_i$$

$$0 = \sum_{i=1}^{n} w_i \zeta_i$$
$$\frac{2}{3} = \sum_{i=1}^{n} w_i \zeta_i^2$$
$$0 = \sum_{i=1}^{n} w_i \zeta_i^3$$

This is a set of 4 equations for $2n$ unknowns. Hence, we should be able to solve for $n = 2$. That is, 2 Gaussian points will be capable of providing exact numerical integration for any polynomial up to degree 3. The above equations can be explicitly written as

$$2 = w_1 + w_2 \tag{9–14}$$
$$0 = w_1\zeta_1 + w_2\zeta_2 \tag{9–15}$$
$$\frac{2}{3} = w_1\zeta_1^2 + w_2\zeta_2^2 \tag{9–16}$$
$$0 = w_1\zeta_1^3 + w_2\zeta_2^3 \tag{9–17}$$

We can now solve for the four unknowns. First we use eqns. (9–15) and (9–17). Arranging in matrix form:

$$\begin{bmatrix} \zeta_1 & \zeta_2 \\ \zeta_1^3 & \zeta_2^3 \end{bmatrix} \begin{Bmatrix} w_1 \\ w_2 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix} \tag{9–18}$$

For this to be true, for non-zero weights, the determinant must equal zero:

$$\zeta_1\zeta_2^3 - \zeta_1^3\zeta_2 = \zeta_1\zeta_2\left(\zeta_2^2 - \zeta_1^2\right) = 0 \tag{9–19}$$

For $\zeta_1 \neq 0$ and $\zeta_2 \neq 0$:

$$\zeta_2^2 - \zeta_1^2 = 0 \tag{9–20}$$

or:

$$\zeta_2 = \pm\zeta_1 \tag{9–21}$$

For two distinct points:

$$\zeta_2 = -\zeta_1 \tag{9–22}$$

Using this information in eqn. (9–15), gives:

$$w_1 = w_2 \tag{9–23}$$

Then, using eqn. (9–14) gives:

$$w_1 = w_2 = 1 \tag{9–24}$$

and using eqn. (9–16) gives:

$$\zeta_1 = -\zeta_2 = \frac{1}{\sqrt{3}} \tag{9–25}$$

Finally, we obtain:

$$\int_{-1}^{1} f(\zeta)d\zeta = 1f\left(\frac{1}{\sqrt{3}}\right) + 1f\left(-\frac{1}{\sqrt{3}}\right) \tag{9–26}$$

That is, we can obtain the exact integral for a polynomial of up to order using two Gauss points with:

$$w_1 = w_2 = 1 \tag{9–27}$$

and

$$\zeta_1 = \frac{1}{\sqrt{3}}, \quad \zeta_2 = -\frac{1}{\sqrt{3}} \tag{9–28}$$

For polynomials of order greater than three, the numerical integral with 2 Gauss points is only approximate.

Question 2: *Why is Gaussian integration efficient?*

9.4 Examples

To illustrate, let us integrate the following function:

$$f(\zeta) = 100 + 50\zeta + 75\zeta^2 \tag{9–29}$$

The exact result is:

$$\int_{-1}^{1} f(\zeta)d\zeta = 250 \tag{9–30}$$

Using our derived Gauss method:

$$\int_{-1}^{1} f(\zeta)d\zeta = (1)f\left(-\frac{1}{\sqrt{3}}\right) + (1)f\left(\frac{1}{\sqrt{3}}\right) \tag{9–31}$$

Evaluating gives:

$$\int_{-1}^{1} f(\zeta) f\zeta = (1)\left[100 + 50\left(-\frac{1}{\sqrt{3}}\right) + 75\left(-\frac{1}{\sqrt{3}}\right)^2\right] + (1)\left[100 + 50\left(\frac{1}{\sqrt{3}}\right) + 75\left(\frac{1}{\sqrt{3}}\right)^2\right] \tag{9–32}$$

or

$$\int_{-1}^{1} f(\zeta) d\zeta = 250 \tag{9–33}$$

This is the exact integral, as expected, since the order of the function being integrated was 2.

As stated, if the function is of order $N > 3$, the numerical integral will be approximate. For example, if the function is:

$$f(\zeta) = \cos(\zeta) \tag{9–34}$$

the integration will not be exact. We know this because the series expansion of $\cos(x)$ is:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \cdots \tag{9–35}$$

where the order of the polynomial is infinite, obviously > 3.

The exact integral from -1 to 1 is:

$$\int_{-1}^{1} \cos(\zeta) d\zeta = 1.682941 \tag{9–36}$$

While the approximate integral using two Gauss points is:

$$\int_{-1}^{1} \cos(\zeta) d\zeta = 1.675823 \tag{9–37}$$

Thus, the numerical integral is approximate. Using more Gauss points would lead to a more accurate numerical integral.

Question 3: *For one-dimensional, 3-noded quadratic elements, to obtain accurate element stiffness matrix, how many integration points are needed?*

9.5 Higher Order Gauss-Legendre Integration

Table for higher order integration using more Gauss points are given in many books. Table 9–1 gives values for up to three Gauss points. A polynomial of order $N \leq 2n - 1$, where is the number of Gauss points will be integrated exactly.

Table 9–1: Higher Order Gauss-Legendré Integration

n	ζ_i	w_i
1	0.0	2.0
2	0.577350269189626 (or $1/\sqrt{3}$)	1.0
3	0.774596669241483 (or $\sqrt{0.6}$)	0.555555555555555 (or $5/9$)
	0.0	0.888888888888889 (or $8/9$)

9.6 Application of Numerical Integration to Evaluation of Element Matrices

We will now demonstrate how numerical integration can be used to calculate an element stiffness matrix. We previously changed variables for the element stiffness matrix in eqn. (9–2):

$$[\boldsymbol{K}^e] = AE \int_0^{L^e} \lfloor \boldsymbol{B} \rfloor^T \lfloor \boldsymbol{B}(\zeta) \rfloor \, dx = AE \int_{-1}^{1} \lfloor \boldsymbol{B}(\zeta) \rfloor^T \lfloor \boldsymbol{B}(\zeta) \rfloor \frac{dx}{d\zeta} d\zeta \tag{9–38}$$

We have also shown how we can implement a numerical integration scheme where we evaluate the function at a given number of points and multiply the values by a weight and then sum the results:

$$\int_{-1}^{1} f(\zeta) d\zeta \approx \sum_{i=1}^{N_{GP}} w_i f(\zeta_i) \tag{9–39}$$

Applying eqn. (9–39) into eqn. (9–38) gives:

$$[\boldsymbol{K}^e] \approx \sum_{i=1}^{N_{GP}} w_i \left[AE \lfloor \boldsymbol{B}(\zeta_i) \rfloor^T \lfloor \boldsymbol{B}(\zeta_i) \rfloor \frac{dx(\zeta_i)}{d\zeta} \right] \tag{9–40}$$

This maybe looks a bit complicated, but all it means is that we will evaluate the term in square brackets at however many Gauss points we are using, multiply by the weights, and then summing the result.

Recall, for a linear element:

$$\lfloor \boldsymbol{B}(\zeta) \rfloor = \begin{bmatrix} \frac{dN_I(\zeta)}{dx} & \frac{dN_J(\zeta)}{dx} \end{bmatrix} \tag{9–41}$$

and we have already shown in eqn. (9–7) that:

$$\frac{dN_I(\zeta)}{dx} = \frac{-1}{L^e} \tag{9–42}$$

Similarly:

$$\frac{dN_J(\zeta)}{dx} = \frac{1}{L^e} \tag{9–43}$$

Also, in eqn. (9–5) we derived that (for a linear element):

$$\frac{dx}{d\zeta} = \frac{L^e}{2} \tag{9–44}$$

Substituting eqns. (9–41) through (9–44) into eqn. (9–40), we obtain:

$$[\boldsymbol{K}^e] \approx \sum_{i=1}^{N_{GP}} w_i \left[AE \left\{ \begin{array}{c} \frac{-1}{L^e} \\ \frac{1}{L^e} \end{array} \right\} \left\lfloor \begin{array}{cc} \frac{-1}{L^e} & \frac{1}{L^e} \end{array} \right\rfloor \left(\frac{L^e}{2} \right) \right] \tag{9–45}$$

or:

$$[\boldsymbol{K}^e] \approx \sum_{i=1}^{N_{GP}} w_i \left(\frac{AE}{L^e} \left[\begin{array}{cc} \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{array} \right] \right) \tag{9–46}$$

Obviously, this is a simple integral. The function we are integrating is a constant, so we only need to use one Gauss point to integrate it exactly. In that case, $n = 1$, $w_i = 2$, and $\zeta_i = 0$ (not used for constant function). Implementing the Gauss-Legendré integration, gives:

$$[\boldsymbol{K}^e] \approx 2 \left(\frac{AE}{L^e} \left[\begin{array}{cc} \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{array} \right] \right) \tag{9–47}$$

and we finally obtain our old friend:

$$[\boldsymbol{K}^e] \approx \frac{AE}{L^e} \left[\begin{array}{cc} 1 & -1 \\ -1 & 1 \end{array} \right] \tag{9–48}$$

In this case, the integration is exact, but in general, the numerical integration will be approximate.

9.7 Hints to Questions

Hint 1: Because both the locations and weights are optimized.

Hint 2: 2 integration points.

Hint 3: The change of the integration area from one coordinate system to another.

Chapter 10

Brief Review of Elasticity

10.1 Purpose

Our purpose is to briefly review the basic formulas and equations in the theory of linear elasticity. This will prepare us for the finite element treatment of elasticity problems.

10.2 Concept of Stress

In the following derivations, we assume the stresses on an infinitesimal block are as shown in Figure 10–1:

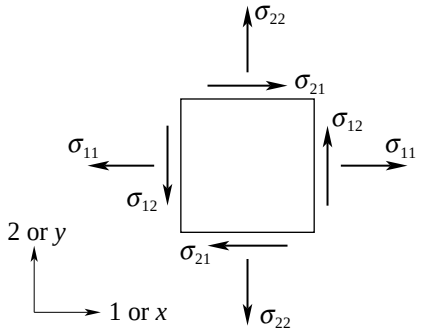


Fig. 10–1 Stresses on infinitesimal block.

We will equivalently either use indices or x and y to indicate directions. Note that moment equilibrium of the block implies symmetry of the stress tensor. We can write the components of the stress tensor either as a matrix or as a vector:

$$[\boldsymbol{\sigma}] = \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{12} & \sigma_{22} \end{bmatrix} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} \end{bmatrix} \tag{10–1}$$

or:

$$\{\boldsymbol{\sigma}\} = \left\{ \begin{array}{c} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{array} \right\} = \left\{ \begin{array}{c} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{array} \right\} \tag{10–2}$$

Note that the (2,1) element of the matrix in eqn. (10–1) is written as σ_{12} rather than σ_{21}. This is because the stress tensor is symmetric and $\sigma_{12} = \sigma_{21}$ as will be indicated later. Very often, the stress tensor is denoted as σ_{ij}, where i and j may assume the value of 1 or 2 in two dimensional problems, or 1, 2, or 3 in three dimensional problems. Note that *the first index, i, denotes the direction of the outward normal of the plane on which the stress is defined, while the second index, j, refers to the direction the traction is pointing.*

10.3 Relation Between Surface Traction and Stress

We first establish a relationship between surface tractions and stress. To do this, we look at a wedge of unit thickness:

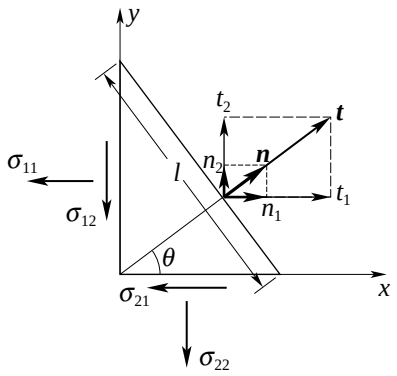


Fig. 10–2 Infinitesimal wedge.

Equilibrium in the X direction gives:

$$t_1 l - \sigma_{11} l \cos\theta - \sigma_{12} l \sin\theta = 0$$

and in the Y direction:

$$t_2 l - \sigma_{22} l \sin\theta - \sigma_{12} l \cos\theta = 0$$

Or,

$$t_1 = \sigma_{11} \cos\theta + \sigma_{12} \sin\theta \tag{10–3}$$

$$t_2 = \sigma_{22} \sin\theta + \sigma_{12} \cos\theta \tag{10–4}$$

Looking at the normal:

$$n_1 = \cos\theta \qquad n_2 = \sin\theta \tag{10–5}$$

Then eqns. (10–3) and (10–4) become:

$$t_1 = \sigma_{11} n_1 + \sigma_{12} n_2 \tag{10–6}$$

$$t_2 = \sigma_{22} n_2 + \sigma_{12} n_1 \tag{10–7}$$

or:

$$t_i = \sigma_{ij} n_j \tag{10–8}$$

for short. Repeated index indicates summation from 1 to 2 for 2D, or from 1 to 3 for 3D.

10.4 Equilibrium

We next look at the stresses on a differential element:

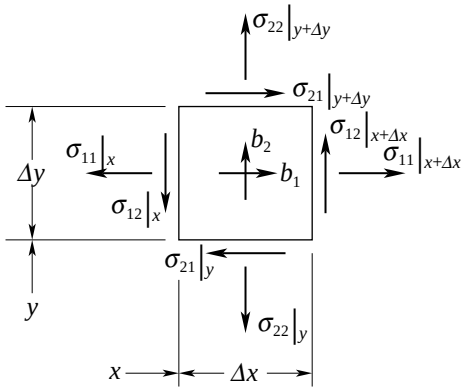


Fig. 10–3 Differential element.

Here, b_i is a body force per unit volume.
Equilibrium in the X direction gives:

$$\sum F_x = 0$$

$$-\sigma_{11}|_x \Delta y + \sigma_{11}|_{x+\Delta x} \Delta y - \sigma_{12}|_y \Delta x + \sigma_{12}|_{y+\Delta y} \Delta x + b_1 \Delta x \Delta y = 0 \tag{10–9}$$

Rearranging and divide by $\Delta x \Delta y$:

$$\frac{\sigma_{11}|_{x+\Delta x} - \sigma_{11}|_x}{\Delta x} + \frac{\sigma_{11}|_{y+\Delta y} - \sigma_{11}|_y}{\Delta y} + b_1 = 0 \tag{10–10}$$

Taking the limit as $\Delta x \to 0$, $\Delta y \to 0$:

$$\frac{\partial \sigma_{11}}{\partial x_1} + \frac{\partial \sigma_{12}}{\partial x_2} + b_1 = 0 \tag{10–11}$$

A similar statement of equilibrium in the Y direction gives:

$$\frac{\partial \sigma_{22}}{\partial x_2} + \frac{\partial \sigma_{12}}{\partial x_1} + b_2 = 0 \tag{10–12}$$

Equations eqns. (10–11) and (10–12) can be combined as:

$$\sigma_{ij,j} + b_i = 0, \qquad i = 1, 2 \tag{10–13}$$

Equilibrium of the moment on the material element leads to $\sigma_{12} = \sigma_{21}$. The stress tensor is therefore symmetric.

10.5 Strain-Displacement Relations

As for all these relations we are deriving, there are different approaches to the same end. In this case, we will use a simple derivation assuming small displacements (deformed and original lengths are approximately the same) and small rotations ($\sin\theta = \theta$ and $\cos\theta = 1$).

The displacements are defined as functions of position:

$$u_1 = u_1(x, y) \qquad u_2 = u_2(x, y) \tag{10–14}$$

Notation can be equivalently:

$$\{\boldsymbol{u}\} = \left\{ \begin{array}{c} u_1 \\ u_2 \end{array} \right\} = \left\{ \begin{array}{c} u \\ v \end{array} \right\} = \left\{ \begin{array}{c} u_x \\ u_y \end{array} \right\} \tag{10–15}$$

shows an element that has first undergone a rigid translation, then a stretching deformation, a uniform rotation, and a shear.

A normal strain is the stretching, that is, change in length divided by length:

$$\varepsilon_1 = \lim_{\Delta x \to 0} \frac{u_1|_{x+\Delta x, y} - u_1|_{x,y}}{\Delta x} = \frac{\partial u_1}{\partial x} \tag{10–16}$$

Similarly,

$$\varepsilon_2 = \lim_{\Delta y \to 0} \frac{u_2|_{x, y+\Delta y} - u_2|_{x,y}}{\Delta y} = \frac{\partial u_2}{\partial y} \tag{10–17}$$

Note that a rigid body displacement does not induce a strain.

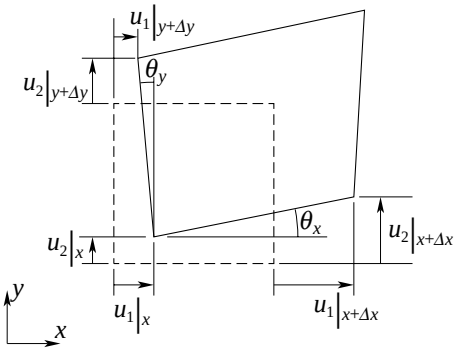


Fig. 10–4 Schematic of deformed element.

The shear strain is measured by the change in angle of the deformed element. In this case:

$$\theta_x = \lim_{\Delta x \to 0} \frac{u_2|_{x+\Delta x,y} - u_2|_{x,y}}{\Delta x} = \frac{\partial u_2}{\partial x} \tag{10–18}$$

and

$$\theta_y = \lim_{\Delta y \to 0} \frac{u_1|_{x,y+\Delta y} - u_1|_{x,y}}{\Delta y} = \frac{\partial u_1}{\partial y} \tag{10–19}$$

The engineering shear strain is measured using the total change in angle:

$$\gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \tag{10–20}$$

The tensor shear strain is one-half the engineering shear strain:

$$\varepsilon_{xy} = \frac{1}{2}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \tag{10–21}$$

or

$$\varepsilon_{xy} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{10–22}$$

or, for all strains:

$$\varepsilon_{ij} = \frac{1}{2}\left(u_{i,j} + u_{j,i}\right) \tag{10–23}$$

From the definition of the strain, it is trivial to show that $\varepsilon_{ij} = \varepsilon_{ji}$. The strain tensor is therefore also symmetric.

Question 1: $\varepsilon_{ij} = {}^{(u_{i,j}+u_{j,i})}/_{2}$ *represents pure deformation (extension and shear). What does the displacement (deformation) gradient represent?*

10.6 Stress-Strain Relations (3D)

In order to understand properly the stress-strain relations in 2D, it is necessary to start from the 3D relations. The generalized Hooke's law links the stresses to the strains or the strains to the stresses as follows:

Stiffness matrix

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \sigma_{xy} \\ \sigma_{xz} \\ \sigma_{yz} \end{Bmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix} \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ 2\varepsilon_{xy} \\ 2\varepsilon_{xz} \\ 2\varepsilon_{yz} \end{Bmatrix} \tag{10–24}$$

Compliance matrix

$$\begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ 2\varepsilon_{xy} \\ 2\varepsilon_{xz} \\ 2\varepsilon_{yz} \end{Bmatrix} = \frac{1}{E} \begin{bmatrix} 1 & -\nu & -\nu & 0 & 0 & 0 \\ -\nu & 1 & -\nu & 0 & 0 & 0 \\ -\nu & -\nu & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2(1+\nu) & 0 & 0 \\ 0 & 0 & 0 & 0 & 2(1+\nu) & 0 \\ 0 & 0 & 0 & 0 & 0 & 2(1+\nu) \end{bmatrix} \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \sigma_{xy} \\ \sigma_{xz} \\ \sigma_{yz} \end{Bmatrix} \tag{10–25}$$

If we define the following Lamé constants:

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)} \tag{10–26}$$

$$G = \frac{E}{2(1+\nu)} \tag{10–27}$$

we can also write the stress-strain relation as:

$$\sigma_{ij} = 2G\varepsilon_{ij} + \lambda\varepsilon_{kk}\delta_{ij} \tag{10–28}$$

Even though we wish to deal with 2-D elastic problems only, the above expressions show clearly that, if the Poisson's ratio of the material is non-zero and the stress-strain state is not trivially zero, it is impossible for both the normal stress and normal strain (σ_{33} and ε_{33}) to be zero simultaneously. It is therefore necessary to distinguish two types of plane problems in elasticity: plane strain and plane stress.

In plane strain, the thickness dimension is much larger then the in-plane dimensions ("thick plate"), and all out-of-plane strain components (ε_{13}, ε_{23}, and ε_{33}) are zero. For plane strain, although the out-of-plane shear stresses (σ_{13} and σ_{23}) are zero, the normal stress in the thickness direction (σ_{33}) is usually non-zero.

In plane stress, the thickness dimension is much smaller then the in-plane dimensions ("thin plate"), and all out-of-plane stress components (σ_{13}, σ_{23}, and σ_{33}) are zero. For plane stress, although the out-of-plane shear strains (ε_{13} and ε_{23}) are zero, the normal strain (ε_{33}) in the thickness direction is usually non-zero.

In this course, we will focus on the plane strain problems.

10.7 Stress-Strain Relations (Plane Strain)

For plane strain, the Hooke's law reduces (taking the submatrix from the 3D stiffness matrix) to the following:

$$\left\{ \begin{array}{c} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{array} \right\} = \frac{E}{(1+\nu)(1-2\nu)} \left[\begin{array}{ccc} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \dfrac{1-2\nu}{2} \end{array} \right] \left\{ \begin{array}{c} \varepsilon_x \\ \varepsilon_y \\ 2\varepsilon_{xy} \end{array} \right\} \tag{10–29}$$

We must invert eqn. (10–29), instead of taking the submatrix of the 3D compliance matrix, to obtain the expression for the strains in terms of the stresses:

$$\left\{ \begin{array}{c} \varepsilon_x \\ \varepsilon_y \\ 2\varepsilon_{xy} \end{array} \right\} = \frac{1+\nu}{E} \left[\begin{array}{ccc} 1-\nu & -\nu & 0 \\ -\nu & 1-\nu & 0 \\ 0 & 0 & 2 \end{array} \right] \left\{ \begin{array}{c} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{array} \right\} \tag{10–30}$$

Question 2: *Why is the plane strain compliance matrix not a submatrix of the 3D case?*

10.8 Stress-Strain Relations (Plane Stress)

For completeness, the stress-strain relations for plane stress are also given in the following:

Compliance (taking the submatrix of the 3D compliance matrix):

$$\left\{ \begin{array}{c} \varepsilon_x \\ \varepsilon_y \\ 2\varepsilon_{xy} \end{array} \right\} = \frac{1}{E} \left[\begin{array}{ccc} 1 & -\nu & 0 \\ -\nu & 1 & 0 \\ 0 & 0 & 2(1+\nu) \end{array} \right] \left\{ \begin{array}{c} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{array} \right\} \tag{10–31}$$

Stiffness (the inverse of the above):

$$\left\{ \begin{array}{c} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{array} \right\} = \frac{E}{1-\nu^2} \left[\begin{array}{ccc} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \dfrac{1-\nu}{2} \end{array} \right] \left\{ \begin{array}{c} \varepsilon_x \\ \varepsilon_y \\ 2\varepsilon_{xy} \end{array} \right\} \tag{10–32}$$

Question 3: *Why is the plane stress stiffness matrix not a submatrix of the 3D case?*

10.9 Hints to Questions

Hint 1: Deformation as well as rigid body rotation.

Hint 2: Because the normal stress in thickness is non-zero.

Hint 3: Because the normal strain in thickness is non-zero.

Chapter 11

Finite Elements in Elasticity

11.1 Purpose

Our purpose is to develop a finite element formulation for two-dimensional elasticity. We first develop the weak form of the elasticity equation. We then develop the matrix formulation for the finite element method using element shape functions.

11.2 Elastic Finite Element Formulation

Recall the definition of our problem:

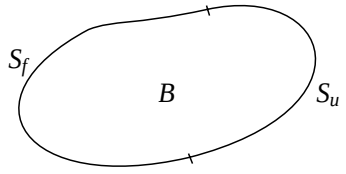


Fig. 11–1 Problem definition.

We start with equilibrium (the strong form of the differential equation, eqn. (10–13)):

$$\sigma_{ij,j} + b_i = 0 \tag{11–1}$$

Again, we multiply by a weighting function that is zero at the essential boundary conditions. We will call this weighting function $\delta \boldsymbol{u}$ to emphasize the equivalence between the weak statement and the variational approach:

$$\left(\sigma_{ij,j} + b_i\right) \delta u_i = 0 \tag{11–2}$$

We then integrate over the volume:

$$\int_V \left(\sigma_{ij,j} + b_i\right) \delta u_i dV = 0 \tag{11–3}$$

or

$$\int_V \sigma_{ij,j} \delta u_i dV + \int_V b_i \delta u_i dV = 0 \tag{11–4}$$

We integrate the left hand term by parts:

$$\int_V \sigma_{ij,j} \delta u_i dV = \int_V \left(\sigma_{ij} \delta u_i\right)_{,j} dV - \int_V \sigma_{ij} \delta u_{i,j} dV \tag{11–5}$$

But we can simplify the term on the right:

$$\frac{\sigma_{ij} \delta u_{i,j}}{2} = \frac{\sigma_{ij} \delta u_{i,j}}{2} \tag{11–6}$$

By symmetry of stress tensor,

$$\frac{\sigma_{ij} \delta u_{i,j}}{2} = \frac{\sigma_{ji} \delta u_{i,j}}{2} \tag{11–7}$$

Switching indices for the term on the right-hand side:

$$\frac{\sigma_{ij} \delta u_{i,j}}{2} = \frac{\sigma_{ij} \delta u_{j,i}}{2} \tag{11–8}$$

Summing eqns. (11–6) and (11–8):

$$\sigma_{ij} \delta u_{i,j} = \frac{1}{2} \sigma_{ij} \left(\delta u_{i,j} + \delta u_{j,i}\right) \tag{11–9}$$

or, using the strain-displacement definition eqn. (10–23):

$$\sigma_{ij} \delta u_{i,j} = \frac{1}{2} \sigma_{ij} \delta \varepsilon_{ij} \tag{11–10}$$

Substituting eqn. (11–10) into eqn. (11–5) gives:

$$\int_V \sigma_{ij,j} \delta u_i dV = \int_V \left(\sigma_{ij} \delta u_i\right)_{,j} dV - \int_V \sigma_{ij} \delta \varepsilon_{ij} dV \tag{11–11}$$

We now use the divergence theorem on the second term of eqn. (11–11) to convert the volume integral to a surface integral:

$$\int_V \sigma_{ij,j} \delta u_i dV = \int_S \sigma_{ij} \delta u_i n_j dS - \int_V \sigma_{ij} \delta \varepsilon_{ij} dV \tag{11–12}$$

We use the relationship between surface traction and stress eqn. (10–8) to obtain:

$$\int_V \sigma_{ij,j}\delta u_i dV = \int_S t_i \delta u_i dS - \int_V \sigma_{ij}\delta\varepsilon_{ij} dV \tag{11–13}$$

Finally, we substitute eqn. (11–13) into eqn. (11–4) to obtain:

$$\int_S t_i \delta u_i dS - \int_V \sigma_{ij}\delta\varepsilon_{ij} dV + \int_V b_i \delta u_i dV = 0 \tag{11–14}$$

or

$$\int_V \sigma_{ij}\delta\varepsilon_{ij} dV = \int_V b_i \delta u_i dV + \int_S t_i \delta u_i dS \tag{11–15}$$

This is the weak form of the differential equation. Along with the essential boundary conditions, it describes our problem. Note that the surface integration is only performed over the surface that has natural boundary conditions. It is also called the "principal of virtual work." The left hand side corresponds to internal work, while the right hand side corresponds to external and body force work. Also note that the strains and displacements are self-consistent, as are the stresses and loads, but that the stress terms do not need to be consistent with the strain terms.

This forms the basis of our finite element approximation.

Question 1: *What is the physical meaning of the left hand side of eqn. (11–15)? What is the physical meaning of the right hand side of eqn. (11–15)? What is the physical meaning of eqn. (11–15)?*

11.3 Finite Element Statement

For convenience, we now introduce matrix notation where:

$$\{\boldsymbol{\sigma}\} = \begin{Bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{12} \\ \sigma_{23} \\ \sigma_{31} \end{Bmatrix} \qquad \{\delta\boldsymbol{\varepsilon}\} = \begin{Bmatrix} \delta\varepsilon_{11} \\ \delta\varepsilon_{22} \\ \delta\varepsilon_{33} \\ \delta\gamma_{12} \\ \delta\gamma_{23} \\ \delta\gamma_{31} \end{Bmatrix} \tag{11–16}$$

$$\{\delta\boldsymbol{u}\} = \begin{Bmatrix} \delta u_1 \\ \delta u_2 \\ \delta u_3 \end{Bmatrix} \qquad \{\boldsymbol{t}\} = \begin{Bmatrix} t_1 \\ t_2 \\ t_3 \end{Bmatrix} \tag{11–17}$$

The reader should note that the use of engineering shear strain (twice the tensorial shear strain) is deliberate. This is because the full stress and strain tensors have nine components and

$$\sigma_{ij}\delta\varepsilon_{ij} = \lfloor\, \sigma_{11}\ \sigma_{22}\ \sigma_{33}\ \sigma_{12}\ \sigma_{23}\ \sigma_{31}\ \sigma_{21}\ \sigma_{32}\ \sigma_{13}\, \rfloor \begin{Bmatrix} \delta\varepsilon_{11} \\ \delta\varepsilon_{22} \\ \delta\varepsilon_{33} \\ \delta\varepsilon_{12} \\ \delta\varepsilon_{23} \\ \delta\varepsilon_{31} \\ \delta\varepsilon_{21} \\ \delta\varepsilon_{32} \\ \delta\varepsilon_{13} \end{Bmatrix}$$

$$= \lfloor\, \sigma_{11}\ \sigma_{22}\ \sigma_{33}\ \sigma_{12}\ \sigma_{23}\ \sigma_{31}\, \rfloor \begin{Bmatrix} \delta\varepsilon_{11} \\ \delta\varepsilon_{22} \\ \delta\varepsilon_{33} \\ 2\delta\varepsilon_{12} \\ 2\delta\varepsilon_{23} \\ 2\delta\varepsilon_{31} \end{Bmatrix} \tag{11–18}$$

Substituting into eqn. (11–15):

$$\int_V \{\delta\boldsymbol{\varepsilon}\}^T\{\boldsymbol{\sigma}\}dV = \int_V \{\delta\boldsymbol{u}\}^T\{\boldsymbol{b}\}dV + \int_S \{\delta\boldsymbol{u}\}^T\{\boldsymbol{t}\} \tag{11–19}$$

Question 2: *Why is it necessary to use the engineering shear strain, $\delta\gamma_{12} = 2\delta\varepsilon_{12}$, etc., in expression eqn. (11–16)?*

Question 3: *What is the phyical meaning of eqn. (11–19)?*

We now use shape functions to interpolate the solution. We will use functions that have a unit value at nodes, so we indicate the nodal values by $\{\delta\boldsymbol{u}_a\}$:

$$\{\delta\boldsymbol{u}\} = [\boldsymbol{N}]\,\{\delta\boldsymbol{u}_a\} \tag{11–20}$$

Note that in 3D, each node has three degrees of freedom (u, v, and w), and the expression

in the above becomes

$$\begin{Bmatrix} \delta u \\ \delta v \\ \delta w \end{Bmatrix} = \begin{bmatrix} N_1 & 0 & 0 & N_1 & 0 & 0 & \cdots \\ 0 & N_1 & 0 & 0 & N_1 & 0 & \cdots \\ 0 & 0 & N_1 & 0 & 0 & N_1 & \cdots \end{bmatrix} \begin{Bmatrix} \delta u_1 \\ \delta v_1 \\ \delta w_1 \\ \delta u_2 \\ \delta v_2 \\ \delta w_2 \\ \delta u_3 \\ \delta v_3 \\ \delta w_3 \\ \vdots \end{Bmatrix} \tag{11–21}$$

where the subscript denotes node number. Note that the shape functions are functions of x, y, and z, but the nodal displacements $\{\delta \boldsymbol{u}_a\}$ are not.

We can also write:

$$\{\delta \boldsymbol{u}\}^T = \{\delta \boldsymbol{u}_a\}^T [\boldsymbol{N}]^T \tag{11–22}$$

And similarly:

$$\begin{aligned} \{\delta \boldsymbol{\varepsilon}\} &= [\boldsymbol{B}]\{\delta \boldsymbol{u}_a\} \\ \{\delta \boldsymbol{\varepsilon}\}^T &= \{\delta \boldsymbol{u}_a\}^T [\boldsymbol{B}]^T \end{aligned} \tag{11–23}$$

Note here the formulation is in 3D, and the shape function should also be understood as function of x, y, and z, *i.e.*, 3D shape function. Later, we will specialize the formulas to 2D problems for simplicity in programming. We will elaborate on 2D shape functions in later lectures.

Substituting eqns. (11–22) and (11–23) into eqn. (11–19) gives:

$$\int_V \{\delta \boldsymbol{u}_a\}^T [\boldsymbol{B}]^T \{\boldsymbol{\sigma}\} dV - \int_V \{\delta \boldsymbol{u}_a\}^T [\boldsymbol{N}]^T \{\boldsymbol{b}\} dV - \int_S \{\delta \boldsymbol{u}_a\}^T [\boldsymbol{N}]^T \{\boldsymbol{t}\} dS = 0 \tag{11–24}$$

Since the nodal displacements associated with the weighting function are constant with respect to the integration:

$$\{\delta \boldsymbol{u}_a\}^T \left(\int_V [\boldsymbol{B}]^T \{\boldsymbol{\sigma}\} dV - \int_V [\boldsymbol{N}]^T \{\boldsymbol{b}\} dV - \int_S [\boldsymbol{N}]^T \{\boldsymbol{t}\} dS \right) = 0 \tag{11–25}$$

And since this must hold true for all values of the weighting function constants:

$$\int_V [\boldsymbol{B}]^T \{\boldsymbol{\sigma}\} dV = \int_V [\boldsymbol{N}]^T \{\boldsymbol{b}\} dV + \int_S [\boldsymbol{N}]^T \{\boldsymbol{t}\} dS \tag{11–26}$$

By recognizing that we can integrate over each element and sum the integrals, we obtain:

$$\sum_{e=1}^{N_{\text{elm}}} \int_{V_e} [\boldsymbol{B}]^T \{\boldsymbol{\sigma}\} dV = \sum_{e=1}^{N_{\text{elm}}} \int_{V_e} [\boldsymbol{N}]^T \{\boldsymbol{b}\} dV + \sum_{e=1}^{N_f} \int_{S_{\text{et}}} [\boldsymbol{N}]^T \{\boldsymbol{t}\} dS \tag{11–27}$$

Although not our final endpoint, (11-27) is an important finite element statement. It is used in "explicit" dynamic finite element codes. Such codes are used for problems that involve large displacements and large strains, such as automobile crash simulation or weapons simulations. The primary development of explicit codes occurred at the National Laboratories by people including Sam Key (HONDO, Sandia) and Hallquist (DYNA2D/3D, Livermore). These codes have now been commercialized (Livermore Associates, Hallquist) and Abaqus Explicit (Taylor and Flannegan).

Our path is not quite finished. We will now assume an elastic problem with small strains. The displacements are interpolated using shape functions:

$$\{\boldsymbol{u}\} = [\boldsymbol{N}_0]\{\boldsymbol{u}_0\} + [\boldsymbol{N}]\{\boldsymbol{u}_a\} \tag{11–28}$$

Note that these are now the real nodal displacements, not a weighting function (variation). As before, we have included shape functions, , that satisfy the essential boundary conditions. Taking the appropriate derivatives of the shape functions we can write:

$$\{\boldsymbol{\varepsilon}\} = [\boldsymbol{B}_0]\{\boldsymbol{u}_0\} + [\boldsymbol{B}]\{\boldsymbol{u}_a\} \tag{11–29}$$

and:

$$\{\boldsymbol{\sigma}\} = [\boldsymbol{D}]\{\boldsymbol{\varepsilon}\} \tag{11–30}$$

where the $[\boldsymbol{D}]$ matrix defines the elastic relation between strain and stress. Using eqns. (11–29) and (11–30):

$$\{\boldsymbol{\sigma}\} = [\boldsymbol{D}]\left([\boldsymbol{B}_0]\{\boldsymbol{u}_0\} + [\boldsymbol{B}]\{\boldsymbol{u}_a\}\right) \tag{11–31}$$

Substituting eqn. (11–31) into eqn. (11–27) gives:

$$\sum_{e=1}^{N_{\text{elm}}} \int_{V_e} [\boldsymbol{B}]^T [\boldsymbol{D}] \left([\boldsymbol{B}_0]\{\boldsymbol{u}_0\} + [\boldsymbol{B}]\{\boldsymbol{u}_a\}\right) dV = \sum_{e=1}^{N_{\text{elm}}} \int_{V_e} [\boldsymbol{N}]^T \{\boldsymbol{b}\} dV + \sum_{e=1}^{N_f} \int_{S_{\text{et}}} [\boldsymbol{N}]^T \{\boldsymbol{t}\} dS \tag{11–32}$$

and rearranging:

$$\begin{aligned} \sum_{e=1}^{N_{\text{elm}}} \int_{V_e} [\boldsymbol{B}]^T [\boldsymbol{D}][\boldsymbol{B}] dV \{\boldsymbol{u}_a\} = & \sum_{e=1}^{N_{\text{elm}}} \int_{V_e} [\boldsymbol{N}]^T \{\boldsymbol{b}\} dV + \sum_{e=1}^{N_f} \int_{S_{\text{et}}} [\boldsymbol{N}]^T \{\boldsymbol{t}\} dS \\ & - \sum_{e=1}^{N_{\text{elm}}} \int_{V_e} [\boldsymbol{B}]^T [\boldsymbol{D}][\boldsymbol{B}_0] dV \{\boldsymbol{u}_0\} \end{aligned} \tag{11–33}$$

This is our final finite element statement. In the next lecture, we define the shape functions for our elements.

Question 4: *Is the stiffness matrix* [*bmD*] *the same as eqn. (10–24) in Chapter 10?*

11.4 Hints to Questions

Hint 1: Internal strain energy; work of external forces; energy conservation.

Hint 2: In expression $\sigma_{ij}\delta\varepsilon_{ij}$, shear terms will appear twice. Using $\delta\gamma_{12} = 2\delta\varepsilon_{ij}$ in expression (11-3) takes care of it.

Hint 3: Internal virtual energy = external virtual work.

Hint 4: No, because $\delta\gamma_{12} = 2\delta\varepsilon_{ij}$ is used in (8-3). $[\boldsymbol{D}]$ should be

$$[\boldsymbol{D}] = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix}$$

Chapter 12

Element Shape Functions

12.1 Purpose

Our purpose is to examine the details of finite element formulation for 2-dimensional elasticity analyses using an eight-noded quadrilateral element.

Our goal here is not to describe all element shape functions. Instead, we will focus on the eight-noded element. This element is commonly used for elasticity analysis. We will see that it performs well. The reader will be referred to other texts for a more complete discussion of shape functions.

12.2 Finite Element Families

We have already derived one-dimensional elements with linear and quadratic shape functions (Figure 12–1: One-dimensional elements).

Fig. 12–1 One-dimensional elements.

Similar elements can be derived in two dimensions. These include triangular elements (Figure 12–2) and quadrilateral elements (Figure 12–3).

Finally, the same families of elements can be developed in three-dimensions (Figure 12–4).

Most commercial finite element codes will include all the above elements (and others). There is no simple answer to what element is "best." Large deformation nonlinear analyses typically use four-noded (2-D) or eight-noded (3-D) elements. This technology

Fig. 12–2 Triangular two-dimensional elements.

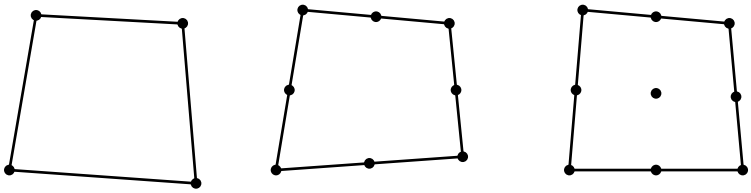


Fig. 12–3 Two-dimensional quadralateral elements.

was originally developed at the national laboratories to solve the large deformation problems associated with weapons. Eight-noded (2-D) and twenty-noded (3-D) elements are often used for elasticity. One advantage of the quadratic elements is that fewer elements are needed for a given accuracy than using the simpler elements. Another advantage is the capability to move the side node to the quarter-points to represent the singularity around a crack tip.

The bottom line is that the analyst has the responsibility to ensure that the mesh being used to represent a problem is appropriate. This must be done knowing the type of element being used and intuition into the particular problem being solved. In the future (and in some current codes) automatic mesh refinement will help ensure a desired accuracy of solution.

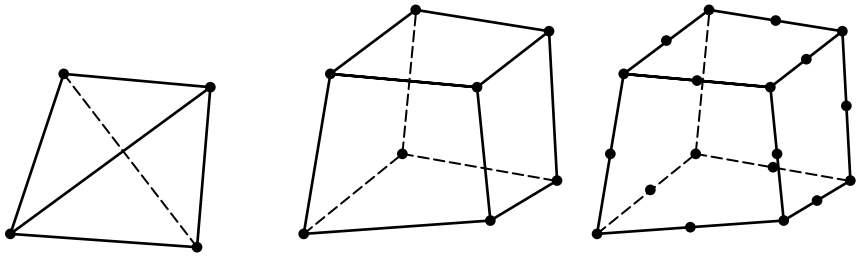


Fig. 12–4 Some three-dimensional elements.

12.3 The Serendipity Element (Q8 or 8-Noded Quadratic Element)

The 2D shape functions $N_i(\xi, \eta)$ are defined by the following relations:

$$u(\xi, \eta) = \sum_i N_i(\xi, \eta) u_i$$
$$v(\xi, \eta) = \sum_i N_i(\xi, \eta) v_i$$

where u_i and v_i are the x and y displacements of node i. Similar to the one-dimensional case, the 2D shape functions satisfy the following conditions:

$$N_i(\xi_j, \eta_j) = \delta_{ij}$$
$$\sum_i N_i(\xi, \eta) = 1$$

We will focus on one particular element, the eight-noded quadrilateral (Q8), for two-dimensional problems. This element is a square in natural coordinates (Figure 12–5).

Although the original derivation of the shape functions for this element were obtained informally, hence the name "serendipity element", a formula for the shape function at node i can be generally expressed by

$$N_i = C \prod (\text{2 lines passing } \textit{other} \text{ nodes but } i)$$

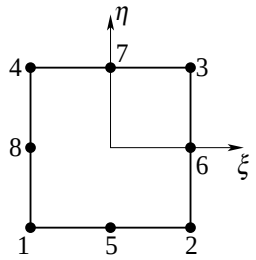


Fig. 12–5 Serendipity element.

Every node, except node i, must be passed at least once by one or more of the three lines. The constant C is determined by using the condition $N_i(\xi_i, \eta_i) = 1$. For example, for node 1, draw three lines, as shown in Figure 12–6:

Line 1: passing nodes 5 and 8	Equation:	$\xi + \eta + 1 = 0$
Line 2: passing nodes 2, 6 and 3	Equation:	$\xi - 1 = 0$
Line 3: passing nodes 4, 7 and 3	Equation:	$\eta - 1 = 0$

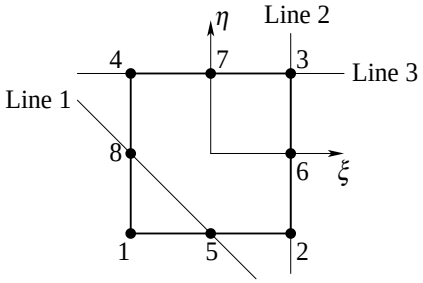


Fig. 12–6 Serendipity element.

Then,

$$N_1 = C(\xi + \eta + 1)(\xi - 1)(\eta - 1)$$

From $N_1(-1,-1) = 1$, we find that $C = -1/4$, N_1 is therefore determined to be

$$N_1 = -\frac{1}{4}(\xi + \eta + 1)(\xi - 1)(\eta - 1)$$

Question 1: *Why do we require the equation of 3 lines in the above formula?*

Following the procedures described above, the shape function for all nodes can be obtained. The shape functions for this element are:

$$N_1 = -\frac{1}{4}(1 - \xi)(1 - \eta)(\xi + \eta + 1) \tag{12–1}$$

$$N_2 = \frac{1}{4}(1 + \xi)(1 - \eta)(\xi - \eta - 1) \tag{12–2}$$

$$N_3 = \frac{1}{4}(1 + \xi)(1 + \eta)(\xi + \eta - 1) \tag{12–3}$$

$$N_4 = -\frac{1}{4}(1 - \xi)(1 + \eta)(\xi - \eta + 1) \tag{12–4}$$

$$N_5 = \frac{1}{2}(1 - \xi^2)(1 - \eta) \tag{12–5}$$

$$N_6 = \frac{1}{2}(1 - \eta^2)(1 + \xi) \tag{12–6}$$

$$N_7 = \frac{1}{2}(1 - \xi^2)(1 + \eta) \tag{12–7}$$

$$N_8 = \frac{1}{2}(1 - \eta^2)(1 - \xi) \tag{12–8}$$

The shape functions are plotted in Figure 12–7 and Figure 12–8. Note that the values are 1.0 at the associated node and 0.0 at all other nodes. The user should verify that along an

edge ($\xi = 1.0$, *etc.*) the shape functions simplify to the previously derived one-dimensional quadratic functions.

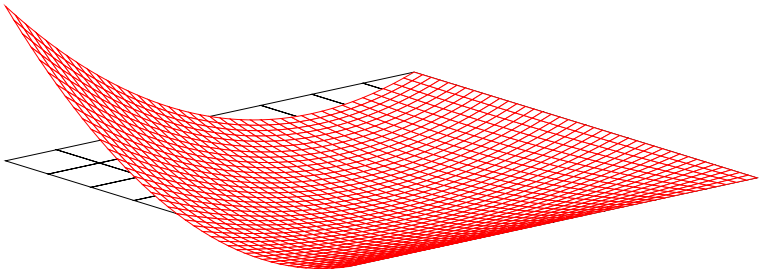


Fig. 12–7 Corner node shape function.

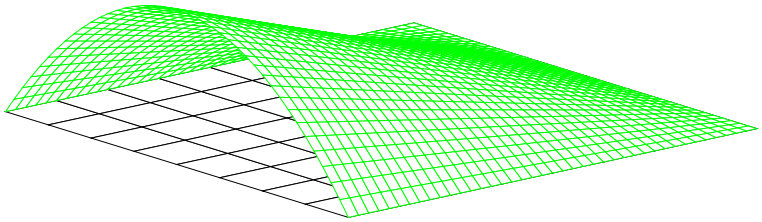


Fig. 12–8 Side node shape function.

12.4 Details of the Element Matrices

Since we have now selected the particular element we will be using, we can now give details of the $[\boldsymbol{N}]$ and $[\boldsymbol{B}]$ matrices defined in the previous chapter. The $[\boldsymbol{N}]$ matrix interpolates the nodal displacements to the interior of the element:

$$\{\boldsymbol{u}\} = [\boldsymbol{N}]\{\boldsymbol{u}_a\} \tag{12–9}$$

For two-dimensional elasticity, each node has displacements in the X and Y directions.

The interpolation then is:

$$\left\{ \begin{array}{c} u \\ v \end{array} \right\} = \left[\begin{array}{cccccccccc} N_1 & 0 & N_2 & 0 & N_3 & 0 & \cdots & N_8 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 & \cdots & 0 & N_8 \end{array} \right] \left\{ \begin{array}{c} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ \vdots \\ u_8 \\ v_8 \end{array} \right\} \tag{12–10}$$

Note that the X displacement at any point in the element, $u(x, y)$, depends on only the X-components of nodal displacements, while $v(x, y)$ depends only on the Y-components.

The $[\boldsymbol{B}]$ matrix relates strains to displacements. Since we use the shape functions to interpolate displacements:

$$u = N_1 u_1 + N_2 u_2 + N_3 u_3 + N_4 u_4 + N_5 u_5 + N_6 u_6 + N_7 u_7 + N_8 u_8 \tag{12–11}$$

and then,

$$\varepsilon_x = \frac{\partial u}{\partial x} = \frac{\partial N_1}{\partial x} u_1 + \frac{\partial N_2}{\partial x} u_2 + \frac{\partial N_3}{\partial x} u_3 + \cdots + \frac{\partial N_8}{\partial x} u_8 \tag{12–12}$$

Similarly,

$$\varepsilon_y = \frac{\partial v}{\partial y} = \frac{\partial N_1}{\partial y} v_1 + \frac{\partial N_2}{\partial y} v_2 + \frac{\partial N_3}{\partial y} v_3 + \cdots + \frac{\partial N_8}{\partial y} v_8 \tag{12–13}$$

and,

$$\gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} = \frac{\partial N_1}{\partial y} u_1 + \frac{\partial N_1}{\partial x} v_1 + \frac{\partial N_2}{\partial y} u_2 + \frac{\partial N_2}{\partial x} v_2 + \cdots + \frac{\partial N_8}{\partial y} u_8 + \frac{\partial N_8}{\partial x} v_8 \tag{12–14}$$

Combined, these define the $[\boldsymbol{B}]$ matrix:

$$\left\{ \begin{array}{c} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{array} \right\} = \left[\begin{array}{cccccccc} \dfrac{\partial N_1}{\partial x} & 0 & \dfrac{\partial N_2}{\partial x} & 0 & \cdots & \dfrac{\partial N_8}{\partial x} & 0 \\ 0 & \dfrac{\partial N_1}{\partial y} & 0 & \dfrac{\partial N_2}{\partial y} & \cdots & 0 & \dfrac{\partial N_8}{\partial y} \\ \dfrac{\partial N_1}{\partial y} & \dfrac{\partial N_1}{\partial x} & \dfrac{\partial N_2}{\partial y} & \dfrac{\partial N_2}{\partial x} & \cdots & \dfrac{\partial N_8}{\partial y} & \dfrac{\partial N_8}{\partial x} \end{array} \right] \left\{ \begin{array}{c} u_1 \\ v_1 \\ u_2 \\ v_2 \\ \vdots \\ u_8 \\ v_8 \end{array} \right\} \tag{12–15}$$

or:

$$\{\boldsymbol{\varepsilon}\} = [\boldsymbol{B}]\{\boldsymbol{u}_a\} \tag{12–16}$$

So, if we evaluate the $[\boldsymbol{B}]$ matrix at any point in the element, we can obtain the strains by multiplying it by the nodal displacements.

In eqn. (12–15), we need the derivatives of the shape functions with respect to the global coordinates, but the shape functions are given as functions of the natural coordinates. To obtain the derivatives with respect to global coordinates, we use the chain rule:

$$\frac{\partial N_1}{\partial \xi} = \frac{\partial N_1}{\partial x}\frac{\partial x}{\partial \xi} + \frac{\partial N_1}{\partial y}\frac{\partial y}{\partial \xi} \tag{12–17}$$

$$\frac{\partial N_1}{\partial \eta} = \frac{\partial N_1}{\partial x}\frac{\partial x}{\partial \eta} + \frac{\partial N_1}{\partial y}\frac{\partial y}{\partial \eta} \tag{12–18}$$

or:

$$\begin{Bmatrix} \dfrac{\partial N_1}{\partial \xi} \\ \dfrac{\partial N_1}{\partial \eta} \end{Bmatrix} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} \begin{Bmatrix} \dfrac{\partial N_1}{\partial x} \\ \dfrac{\partial N_1}{\partial y} \end{Bmatrix} \tag{12–19}$$

This defines the *Jacobian* matrix. By using eqn. (12–19), it is implied that the relation between the real and natural coordinates are known. We can choose to use the shape functions that interpolate the displacements to also interpolate the geometry (positions), that is,

$$x = N_1 x_1 + N_2 x_2 + N_3 x_3 + \cdots + N_8 x_8 \tag{12–20}$$

from which

$$\frac{\partial x}{\partial \xi} = \frac{\partial N_1}{\partial \xi} x_1 + \frac{\partial N_2}{\partial \xi} x_2 + \frac{\partial N_3}{\partial \xi} x_3 + \cdots + \frac{\partial N_8}{\partial \xi} x_8 \tag{12–21}$$

with similar expressions for $\partial y/\partial \xi$, $\partial x/\partial \eta$, and $\partial y/\partial \eta$. Elements that use the same shape functions to interpolate both the displacements and positions are called *isoparametric elements*. We can then invert the Jacobian matrix to obtain the derivatives with respect to global coordinates:

$$\begin{Bmatrix} \dfrac{\partial N_1}{\partial x} \\ \dfrac{\partial N_1}{\partial y} \end{Bmatrix} = [\boldsymbol{J}]^{-1} \begin{Bmatrix} \dfrac{\partial N_1}{\partial \xi} \\ \dfrac{\partial N_1}{\partial \eta} \end{Bmatrix} \tag{12–22}$$

Question 2: *Is it necessary to use isoparametric formulation?*

Chapter 13

2-D Numerical Integration

13.1 Purpose

Our purpose is to describe how the element matrices can be integrated numerically. Only a brief discussion of numerical integration will be given. The reader can refer to finite element texts for further details.

13.2 Numerical Integration

In Chapter 11 we derived our finite element statement:

$$\sum_{e=1}^{N_{\text{elm}}} \int_{V_e} [\boldsymbol{B}]^T [\boldsymbol{D}][\boldsymbol{B}] dV \{\boldsymbol{u}_a\} = \sum_{e=1}^{N_{\text{elm}}} \int_{V_e} [\boldsymbol{N}]^T \{\boldsymbol{b}\} dV + \sum_{e=1}^{N_f} \int_{S_{\text{et}}} [\boldsymbol{N}]^T \{\boldsymbol{t}\} dS - \sum_{e=1}^{N_{\text{elm}}} \int_{V_e} [\boldsymbol{B}]^T [\boldsymbol{D}][\boldsymbol{B}_0] dV \{\boldsymbol{u}_0\} \tag{13–1}$$

and in Chapter 12 we presented the details of the $[\boldsymbol{B}]$, $[\boldsymbol{D}]$, and $[\boldsymbol{N}]$ matrices. The integration for each element will be performed numerically.

As for one-dimensional numerical integration, we first transform from real coordinates to natural coordinates (note that in two dimensions, integration over the volume corresponds to performing an area integration multiplied by the thickness.) In two dimensions, we correspondingly need two natural coordinates . The change of variables for integration is:

$$\int_{y_1}^{y_2} \int_{x_1}^{x_2} f(x, y) = \int_{\eta_1}^{\eta_2} \int_{\xi_1}^{\xi_2} f\left(g_1(\xi, \eta), g_2(\xi, \eta)\right) |\boldsymbol{J}| d\xi d\eta \tag{13–2}$$

where:

$$x = g_1(\xi, \eta)$$
$$y = g_2(\xi, \eta)$$
$$|\boldsymbol{J}| = \begin{vmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{vmatrix} = \begin{vmatrix} \frac{\partial g_1}{\partial \xi} & \frac{\partial g_2}{\partial \xi} \\ \frac{\partial g_1}{\partial \eta} & \frac{\partial g_2}{\partial \eta} \end{vmatrix} = \text{Determinant of Jacobian matrix} \tag{13–3}$$

Question 1: *What is the physical meaning of the determinant of Jacobian matrix in eqn. (13–3)?*

In our case, the element stiffness matrix is then:

$$[\boldsymbol{K}^e] = \int_{V_e} [\boldsymbol{B}]^T [\boldsymbol{D}][\boldsymbol{B}] dV = t \int_{-1}^{1} \int_{-1}^{1} [\boldsymbol{B}]^T [\boldsymbol{D}][\boldsymbol{B}] |\boldsymbol{J}| d\xi d\eta \tag{13–4}$$

or

$$[\boldsymbol{K}^e] = t \int_{-1}^{1} \int_{-1}^{1} \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_1}{\partial y} \\ 0 & \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} \\ \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_2}{\partial y} \\ 0 & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} \\ \cdots & \cdots & \cdots \\ \frac{\partial N_8}{\partial x} & 0 & \frac{\partial N_8}{\partial y} \\ 0 & \frac{\partial N_8}{\partial y} & \frac{\partial N_8}{\partial x} \end{bmatrix} [\boldsymbol{D}] \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \cdots & \frac{\partial N_8}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & \cdots & 0 & \frac{\partial N_8}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \cdots & \frac{\partial N_8}{\partial y} & \frac{\partial N_8}{\partial x} \end{bmatrix} |\boldsymbol{J}| d\xi d\eta \tag{13–5}$$

where t is the thickness of the body. For plane strain this is usually unit thickness.

The material matrix, $[\boldsymbol{D}]$, relates the stresses to the strains:

$$\{\boldsymbol{\sigma}\} = [\boldsymbol{D}]\{\boldsymbol{\varepsilon}\} \tag{13–6}$$

For elasticity, the $[\boldsymbol{D}]$ matrix is given by Hooke's law. For plain strain:

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{Bmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix} \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} \tag{13–7}$$

and,

$$\sigma_z = \nu(\sigma_x + \sigma_y) \tag{13–8}$$

Similarly, for plane stress:

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{Bmatrix} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \dfrac{1-\nu}{2} \end{bmatrix} \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} \tag{13–9}$$

and,

$$\sigma_z = 0 \tag{13–10}$$

Question 2: *What is the size of the element stiffness matrix for Q8 element?*

13.3 Implementation

Similar to the one-dimensional case, in 2-D we integrate numerically by evaluating the function at selected points in each dimension, multiplying the value by a weight, and summing over all the integration points. In general this leads:

$$t \int_{-1}^{1} \int_{-1}^{1} [\boldsymbol{B}]^T [\boldsymbol{D}][\boldsymbol{B}]|\boldsymbol{J}| d\xi d\eta = t \sum_{i=1}^{N_{GP}} \sum_{j=1}^{N_{GP}} w_i w_j [\boldsymbol{B}(\xi_i, \eta_j)]^T [\boldsymbol{D}][\boldsymbol{B}(\xi_i, \eta_j)]|\boldsymbol{J}(\xi_i, \eta_j)| \tag{13–11}$$

The Gauss points are defined in two dimensions using a product of the one-dimensional locations previously derived. The exact values of the integration locations and weights were given in Chapter 9.

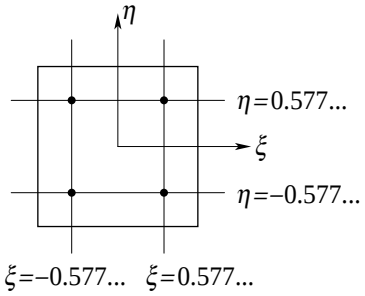


Fig. 13–1 Locations of 2×2 Gauss points.

As before, 2×2 Gauss point integration is precise for cubic polynomials (of each natural coordinate) and 3×3 integration is precise for 5th order polynomials. In our work,

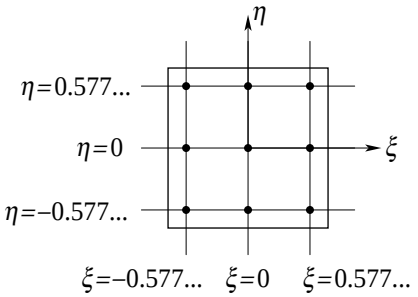


Fig. 13–2 Locations of 3×3 Gauss points.

we will use 3×3 integration. This will be exact integration for parallelepiped elements with the side nodes at the mid-points (the student should verify this).

Numerical integration is typically implemented using nested loops for each direction of integration. For instance, eqn. (13–11) would be implemented as nested loops corresponding to the ξ and η natural coordinates. Inside the inner most loop, all matrices are evaluated, multiplied, and the sum is accumulated to obtain the integral.

13.4 Body Forces

For the body force term at the element level, the numerical integration formula is:

$$\begin{aligned}\int_{V_e} [\boldsymbol{N}]^T \{\boldsymbol{b}\} dV &= t \int_{-1}^{1}\int_{-1}^{1} [\boldsymbol{N}]^T \{\boldsymbol{b}\} |\boldsymbol{J}| d\xi d\eta \\ &= t \sum_{i=1}^{N_{GP}} \sum_{j=1}^{N_{GP}} \Big(w_i w_j [\boldsymbol{N}(\xi_i,\eta_j)]^T \{\boldsymbol{b}(\xi_i,\eta_j)\} |\boldsymbol{J}(\xi_i,\eta_j)| \Big) \end{aligned} \qquad (13\text{–}12)$$

Note that the body force in 2D is a vector of 2 components, $\{\boldsymbol{b}\} = \lfloor b_x \; b_y \rfloor]^T$, while $[\boldsymbol{N}]^T$ is 16×2, therefore the above expression results in a vector of size 16.

13.5 Point Forces and Surface Tractions

When point forces are present, the mesh is usually generated in such a way that there will be a node at each loading point. The force vector then is simply the components of the

point force, e.g. in 2D,

$$\int_{S_{et}} [N]^T \{t\} dS \rightarrow \left\{ \begin{array}{c} f_{1x} \\ f_{1y} \\ f_{2x} \\ f_{2y} \\ \vdots \\ f_{8x} \\ f_{8y} \end{array} \right\} \tag{13–13}$$

For surface tractions in 2D we must integrate along the surface, which is a 1D integration. Along any edge of the Q8 element ($\xi = \pm 1$ or $\eta = \pm 1$) the only non-zero shape functions will be those associated with the edge and, on the edge, the shape functions will be identical to the quadratic shape functions derived for a 1D element (see Chapter 8). For example, for the element in Figure 13–3, along the edge $\xi = 1$, the shape functions reduce to:

$$N_2 = \frac{\xi}{2}(\xi - 1) \tag{13–14}$$

$$N_3 = \frac{\xi}{2}(\xi + 1) \tag{13–15}$$

$$N_6 = (1 - \xi^2) \tag{13–16}$$

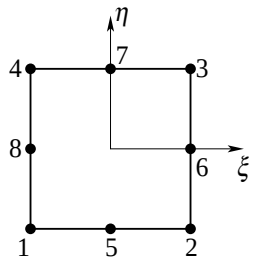


Fig. 13–3 Serendipity element.

For the surface tractions in 2D, the integration becomes one-dimensional. Depending on the element orientation, the formula takes slightly different forms. To derive a generalized formula, let's consider a curved element edge L_{123} passing 3 nodes (Q8 element) as show in the figure below:

Assuming that the element is traversed in a counter-clockwise direction (*i.e.*, the material is on the left side of the boundary), then the tangent direction of an arc line element is

$$\boldsymbol{e}_s = \frac{dx}{ds}\boldsymbol{e}_1 + \frac{dy}{ds}\boldsymbol{e}_2 \tag{13–17}$$

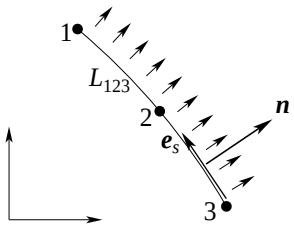


Fig. 13–4 Curved element edge.

and the outward normal to the boundary is

$$\boldsymbol{n} = \boldsymbol{e}_s \times \boldsymbol{e}_3 = \frac{dy}{ds}\boldsymbol{e}_1 - \frac{dx}{ds}\boldsymbol{e}_2 \tag{13–18}$$

Assuming the surface traction t is expressed locally in terms of normal and tangent components t_n and t_s, *i.e.*,

$$\boldsymbol{t} = t_n\boldsymbol{n} + t_s\boldsymbol{e}_s \tag{13–19}$$

The equivalent nodal forces corresponding to the three nodes on L123 are then

$$\begin{Bmatrix} f_{1x} \\ f_{1y} \\ f_{2x} \\ f_{2y} \\ f_{3x} \\ f_{3y} \end{Bmatrix} = \int_{L_{123}} [\boldsymbol{N}]^T \{\boldsymbol{t}\} dS = t \int_{L_{123}} [\boldsymbol{N}]^T \left(t_n\{\boldsymbol{n}\} + t_s\{\boldsymbol{n}\}\right) dS$$

$$= t \int_{L_{123}} \begin{bmatrix} N_1 & 0 \\ 0 & N_1 \\ N_2 & 0 \\ 0 & N_2 \\ N_3 & 0 \\ 0 & N_3 \end{bmatrix} \left(t_n \begin{Bmatrix} \dfrac{dy}{ds} \\ -\dfrac{dx}{ds} \end{Bmatrix} + t_s \begin{Bmatrix} \dfrac{dx}{ds} \\ \dfrac{dy}{ds} \end{Bmatrix} \right) dS \tag{13–20}$$

As before, using the chain rule:

$$\frac{dx}{ds}\frac{ds}{d\xi} = \frac{dx}{d\xi} \qquad \text{and} \qquad \frac{dy}{ds}\frac{ds}{d\xi} = \frac{dy}{d\xi} \tag{13–21}$$

gives

$$\frac{dx}{ds} = \frac{1}{J}\frac{dx}{d\xi} \qquad \text{and} \qquad \frac{dy}{ds} = \frac{1}{J}\frac{dy}{d\xi} \tag{13–22}$$

where

$$ds = Jd\xi \tag{13–23}$$

Integrating using the natural coordinates:

$$\begin{Bmatrix} f_{1x} \\ f_{1y} \\ f_{2x} \\ f_{2y} \\ f_{3x} \\ f_{3y} \end{Bmatrix} = t \int_{-1}^{1} \begin{bmatrix} N_1 & 0 \\ 0 & N_1 \\ N_2 & 0 \\ 0 & N_2 \\ N_3 & 0 \\ 0 & N_3 \end{bmatrix} \left(t_n \begin{Bmatrix} \frac{1}{J}\frac{dy}{d\xi} \\ -\frac{1}{J}\frac{dx}{d\xi} \end{Bmatrix} + t_s \begin{Bmatrix} \frac{1}{J}\frac{dx}{d\xi} \\ \frac{1}{J}\frac{dy}{d\xi} \end{Bmatrix} \right) Jd\xi \tag{13–24}$$

and simplifying:

$$\begin{Bmatrix} f_{1x} \\ f_{1y} \\ f_{2x} \\ f_{2y} \\ f_{3x} \\ f_{3y} \end{Bmatrix} = t \int_{-1}^{1} \begin{bmatrix} N_1 & 0 \\ 0 & N_1 \\ N_2 & 0 \\ 0 & N_2 \\ N_3 & 0 \\ 0 & N_3 \end{bmatrix} \left(t_n \begin{Bmatrix} \frac{dy}{d\xi} \\ -\frac{dx}{d\xi} \end{Bmatrix} + t_s \begin{Bmatrix} \frac{dx}{d\xi} \\ \frac{dy}{d\xi} \end{Bmatrix} \right) d\xi \tag{13–25}$$

here t is the element thickness and N_i are the 1D shape functions for the quadratic element

$$N_1(\xi) = \frac{\xi}{2}(\xi - 1) \tag{13–26}$$

$$N_2(\xi) = (1 - \xi^2) \tag{13–27}$$

$$N_3(\xi) = \frac{\xi}{2}(\xi + 1) \tag{13–28}$$

Using the isoparametric formulation

$$x = \sum_{i=1}^{3} N_i(\xi) x_i \tag{13–29}$$

$$y = \sum_{i=1}^{3} N_i(\xi) y_i \tag{13–30}$$

we have

$$\frac{dx}{d\xi} = \sum_{i=1}^{3} \frac{dN_i(\xi)}{d\xi} x_i \tag{13–31}$$

$$\frac{y}{d\xi} = \sum_{i=1}^{3} \frac{dN_i(\xi)}{d\xi} y_i \tag{13–32}$$

The above integration can be readily evaluated using 3-point Gauss integration in 1D.

Question 3: *Why is the integration formula of surface tractions different from the others?*

13.6 Hints to Questions

Hint 1: Because Q8 is quadratic.

Hint 2: No.

Hint 3: Because its dimensionality is reduced by 1.

Chapter 14

Elastic Stress Computations in Finite Element Analysis

14.1 Background

The displacement-based finite element formulation generates a continuous displacement field over the discretized problem domain. However, since the derivatives of the shape functions are not continuous across boundaries, the stresses (and strains) are only continuous within each element and experience jumps across boundaries of elements. The magnitude of the jump across an element boundary can be used as a measure of the accuracy of the solution. As a general rule, a jump of less than 10stress indicates a mesh that is sufficiently refined to provide an accurate solution.

For display purposes, we may need to provide a continuous stress fields. We need to keep in mind that such stress smoothing is a mere ”post-processing” of the computation results. It makes the data look ”nicer”, but does not make it any more accurate.

Let us make some observations first:

- For elastic problems, it can be proved that the stresses computed at the Gauss points are ”optimum” and more accurate than those calculated at other locations.
- For a Q8 element, if the element is rectangular (in the physical coordinate system), the coordinate transformation between the physical and natural coordinates is linear. In such case, stresses vary linearly with coordinates.

Therefore, the following process of producing continuous stress fields has been proposed:

1. First, calculate the stresses at all the Gauss points.
2. Second, smooth the stresses within the element using an appropriate function to obtain a linear “best-fit” to the stresses.

3. Extrapolate to calculate the stresses at the nodes, called nodal stresses.
4. Finally, the nodal stresses for a given node computed by all elements that share the node are averaged. This gives the final nodal stress.
5. (Afterwards, if the stresses for any point are desired, it can be computed using shape functions to interpolate, just as what we do with displacements. In our finite element program, this is not needed. We use the post-processor to display the stresses.)

14.2 Stresses at Gauss Points

Based on the finite element formulation, the stress at any point in an element can be computed according to

$$\{\boldsymbol{\sigma}(\xi,\eta)\} = [\boldsymbol{D}][\boldsymbol{B}(\xi,\eta)]\{\boldsymbol{u}_a\} \tag{14–1}$$

where $\{\boldsymbol{u}_a\}$ is the nodal displacement vector.

For our smoothing function within the element we will use the same function as the shape functions for a four-noded quadrilateral element.

$$f(\xi,\eta) = A + B\xi + c\eta + D\xi\eta \tag{14–2}$$

14.3 "Best Fit": Least Square Errors Method

Consider a field, given as a set of data at discrete points, $d(\xi_i,\eta_i)$, to be fitted by the function $f(\xi,\eta)$ given in eqn. (14–2). The sum of square errors over all data points is

$$\begin{aligned} E &= \sum_{i=1}^{M}\sum_{j=1}^{N}\Big[f(\xi_i,\eta_j) - d(\xi_i,\eta_j)\Big]^2 \\ &= \sum_{i=1}^{M}\sum_{j=1}^{N}\Big[A + B\xi_i + C\eta_j + D\xi_i\eta_j - d(\xi_i,\eta_j)\Big]^2 \end{aligned} \tag{14–3}$$

To minimize the square errors E, it is required that

$$\frac{\partial E}{\partial A} = 0, \quad \frac{\partial E}{\partial B} = 0, \quad \frac{\partial E}{\partial C} = 0, \quad \text{and} \quad \frac{\partial E}{\partial D} = 0 \tag{14–4}$$

which give

$$\sum_{i=1}^{M}\sum_{j=1}^{N}\Big[A + B\xi_i + C\eta_j + D\xi_i\eta_j - d(\xi_i,\eta_j)\Big] =$$

$$A\sum_{i,j=1}^{M,N}1+B\sum_{i,j=1}^{M,N}\xi_i+C\sum_{i,j=1}^{M,N}\eta_j+D\sum_{i,j=1}^{M,N}\xi_i\eta_j-D\sum_{i,j=1}^{M,N}d(\xi_i,\eta_j)=0 \tag{14–5}$$

$$\sum_{i=1}^{M}\sum_{j=1}^{N}\xi_i\left[A+B\xi_i+C\eta_j+D\xi_i\eta_j-d(\xi_i,\eta_j)\right]=$$

$$A\sum_{i,j=1}^{M,N}\xi_i+B\sum_{i,j=1}^{M,N}\xi_i^2+C\sum_{i,j=1}^{M,N}\xi_i\eta_j+D\sum_{i,j=1}^{M,N}\xi_i^2\eta_j-D\sum_{i,j=1}^{M,N}\xi_i d(\xi_i,\eta_j)=0 \tag{14–6}$$

$$\sum_{i=1}^{M}\sum_{j=1}^{N}\eta_j\left[A+B\xi_i+C\eta_j+D\xi_i\eta_j-d(\xi_i,\eta_j)\right]=$$

$$A\sum_{i,j=1}^{M,N}\eta_j+B\sum_{i,j=1}^{M,N}\xi_i\eta_j+C\sum_{i,j=1}^{M,N}\eta_j^2+D\sum_{i,j=1}^{M,N}\xi_i\eta_j^2-D\sum_{i,j=1}^{M,N}\eta_j d(\xi_i,\eta_j)=0 \tag{14–7}$$

$$\sum_{i=1}^{M}\sum_{j=1}^{N}\eta_j\xi_i\left[A+B\xi_i+C\eta_j+D\xi_i\eta_j-d(\xi_i,\eta_j)\right]=$$

$$A\sum_{i,j=1}^{M,N}\xi_i\eta_j+B\sum_{i,j=1}^{M,N}\xi_i^2\eta_j+C\sum_{i,j=1}^{M,N}\xi_i\eta_j^2+D\sum_{i,j=1}^{M,N}\xi_i^2\eta_j^2-D\sum_{i,j=1}^{M,N}\xi_i\eta_j d(\xi_i,\eta_j)=0 \tag{14–8}$$

They can be re-written as the following set of linear equation system about A, B, C and D

$$\begin{bmatrix}\sum_{i,j=1}^{M,N}1 & \sum_{i,j=1}^{M,N}\xi_i & \sum_{i,j=1}^{M,N}\eta_j & \sum_{i,j=1}^{M,N}\xi_i\eta_j\\ \sum_{i,j=1}^{M,N}\xi_i & \sum_{i,j=1}^{M,N}\xi_i^2 & \sum_{i,j=1}^{M,N}\xi_i\eta_j & \sum_{i,j=1}^{M,N}\xi_i^2\eta_j\\ \sum_{i,j=1}^{M,N}\eta_j & \sum_{i,j=1}^{M,N}\xi_i\eta_j & \sum_{i,j=1}^{M,N}\eta_j^2 & \sum_{i,j=1}^{M,N}\xi_i\eta_j^2\\ \sum_{i,j=1}^{M,N}\xi_i\eta_j & \sum_{i,j=1}^{M,N}\xi_i^2\eta_j & \sum_{i,j=1}^{M,N}\xi_i\eta_j^2 & \sum_{i,j=1}^{M,N}\xi_i^2\eta_j^2\end{bmatrix}\begin{Bmatrix}A\\B\\C\\D\end{Bmatrix}=\begin{Bmatrix}\sum_{i,j=1}^{M,N}d(\xi_i,\eta_j)\\ \sum_{i,j=1}^{M,N}\xi_i d(\xi_i,\eta_j)\\ \sum_{i,j=1}^{M,N}\eta_j d(\xi_i,\eta_j)\\ \sum_{i,j=1}^{M,N}\xi_i\eta_j d(\xi_i,\eta_j)\end{Bmatrix} \tag{14–9}$$

Note that the system matrix (the square matrix on the left) depends only on the natural coordinates of data points, which are the Gauss points for our case. This means the system matrix may only need to be constructed once and be used for all elements. For 3×3 integration, $M=N=3$. Also, $\xi_1=-\sqrt{0.6}$, $\xi_2=0$, $\xi_3=\sqrt{0.6}$ and $\eta_1=-\sqrt{0.6}$, $\eta_2=0$,

$\eta_3 = \sqrt{0.6}$. Using these definitions, the terms evaluate to:

$$\begin{bmatrix} 9 & 0 & 0 & 0 \\ 0 & 3.6 & 0 & 0 \\ 0 & 0 & 3.6 & 0 \\ 0 & 0 & 0 & 1.4 \end{bmatrix} \begin{Bmatrix} A \\ B \\ C \\ D \end{Bmatrix} = \begin{Bmatrix} \sum_{i,j=1}^{M,N} d(\xi_i, \eta_j) \\ \sum_{i,j=1}^{M,N} \xi_i d(\xi_i, \eta_j) \\ \sum_{i,j=1}^{M,N} \eta_j d(\xi_i, \eta_j) \\ \sum_{i,j=1}^{M,N} \xi_i \eta_j d(\xi_i, \eta_j) \end{Bmatrix} \tag{14–10}$$

Since only the diagonal terms are non-zero, all we need to calculate A, B, C, and D is evaluate the right hand side and then divide by the diagonal coefficients. For example, if we want to smooth the σ_x data in an element, to calculate A we sum the σ_x for all Gauss points and divide by 9. Similar calculations are required for the other terms.

14.4 Calculating Nodal Stress

Once the coefficients in eqn. (14–2) are calculated, the nodal stresses can be obtained using the natural coordinates for each node.

As mentioned earlier, stresses are discontinuous across elements. There is no guarantee that nodal stresses for one node calculated for one element, will be the same as calculated from another element that the same node. Therefore, in finite element code, the nodal stress calculated at this stage is stored within the element object itself, not in the node objects.

14.5 Averaging Nodal Stresses

After the nodal stresses for every element is computed. We now can start averaging the nodal stresses. For instance, a corner node is usually shared by 4 adjacent elements. Each of these 4 elements has its own value for the nodal stress of this node. The average over all the 4 elements involved will be computed and then stored in the node object. A side node is usually shared by 2 elements, and the averaging will be performed over these two elements.

Averaging computation is very simple, if you have all values handy. The challenge is: for each node, how do we know what elements share it? A “clean” approach to this problem is that at the same time we are summing the nodal values we count the number of times this summation is being performed. The number of appearance simply means the

number of elements share this particular node. As a last step, we then divide the summation of the stresses at each node by the number of contributing elements to obtain the average.

Chapter 15

Heat Transfer Using Finite Elements

15.1 Purpose

Our purpose is to derive the finite element equations for heat transfer.

15.2 Conservation of Energy

For a differential element with unit thickness:

Figure 15 1: Differential element

Where q_x and q_y are the heat fluxes in the X and Y directions and b is a heat generation per unit volume.

For steady state problems, the heat flow in equals the heat flow out:

$$q_x|_x \, \Delta y + q_y\big|_y \, \Delta x + b\Delta x\Delta y - q_x|_{x+\Delta x} - q_y\big|_{y+\Delta y} = 0 \tag{15–1}$$

Rearranging and divide by $\Delta x\Delta y$:

$$-\frac{q_x|_{x+\Delta x} - q_x|_x}{\Delta x} - \frac{q_y\big|_{y+\Delta y} - q_y\big|_y}{\Delta y} + b = 0 \tag{15–2}$$

Taking the limit as $\Delta x \to 0$, $\Delta y \to 0$:

$$\frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y} - b = 0 \tag{15–3}$$

or:

$$q_{i,i} = b = 0 \qquad\qquad i = 1, 2 \tag{15–4}$$

All parts of the boundary have either temperature or heat flux specified, that is, $T = T_t$ on S_t or $k\,{}^{dT}/_{dn} = h$ on S_h, where n is the outward surface normal.

15.3 Fourier's Law of Heat Conduction

The general statement of Fourier's law of heat conduction is:

$$\left\{ \begin{array}{c} q_x \\ q_y \end{array} \right\} = - \left[\begin{array}{cc} k_{xx} & k_{xy} \\ k_{yx} & k_{yy} \end{array} \right] \left\{ \begin{array}{c} \dfrac{\partial T}{\partial x} \\ \dfrac{\partial T}{\partial y} \end{array} \right\} \tag{15–5}$$

or

$$q_i = -k_{ij} T_{,j} \tag{15–6}$$

where k_{ij} are the thermal conductivities. For an isotropic material, this reduces to:

$$\left\{ \begin{array}{c} q_x \\ q_y \end{array} \right\} = - \left[\begin{array}{cc} k & 0 \\ 0 & k \end{array} \right] \left\{ \begin{array}{c} \dfrac{\partial T}{\partial x} \\ \dfrac{\partial T}{\partial y} \end{array} \right\} \tag{15–7}$$

and eqn. (15–3) becomes:

$$k \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) + b = 0 \tag{15–8}$$

15.4 Weak Statement of Heat Conduction

The weak statement of the problem is developed on the following pages. For interest, the long form and the indicial notation form will be developed in parallel.

As for elasticity, the weighting function is zero at the essential boundary conditions.

15.5 Finite Element Approximation

We will now give the finite element representation of:

$$\int_V \left\lfloor \begin{array}{cc} \dfrac{\partial w}{\partial x} & \frac{pw}{\partial y} \end{array} \right\rfloor \left[\begin{array}{cc} k_{xx} & k_{xy} \\ k_{yx} & k_{yy} \end{array} \right] \left\{ \begin{array}{c} \dfrac{\partial T}{\partial x} \\ \dfrac{\partial T}{\partial y} \end{array} \right\} dV = \int_V wb dV + \int_{S_t} wh dS \tag{15–9}$$

As before, we approximate the solution using shape functions:

$$T_n = \lfloor \boldsymbol{N}_0 \rfloor \{ \boldsymbol{T}_0 \} + \lfloor \boldsymbol{N} \rfloor \{ \boldsymbol{T}_a \} \tag{15–10}$$

where the first term satisfies the essential boundary conditions and the second term represents the unknown temperatures.

Similarly for the weighting function, where the weighting functions are zero at the essential boundary conditions:

$$w = \lfloor \boldsymbol{N} \rfloor \{\boldsymbol{d}_a\} \tag{15–11}$$

The body force and surface heat flux are approximated using the same shape functions (only over the regions where applicable):

$$b = \lfloor \boldsymbol{N} \rfloor \{\boldsymbol{b}_a\} \tag{15–12}$$

$$h = \lfloor \boldsymbol{N} \rfloor \{\boldsymbol{h}_a\} \tag{15–13}$$

The derivatives are then given by:

$$\left\{ \begin{array}{c} \dfrac{\partial T}{\partial x} \\ \dfrac{\partial T}{\partial y} \end{array} \right\} = \left[\begin{array}{ccccc} \dfrac{\partial N_1}{\partial x} & \dfrac{\partial N_2}{\partial x} & \dfrac{\partial N_2}{\partial x} & \cdots & \dfrac{\partial N_n}{\partial x} \\ \dfrac{\partial N_1}{\partial y} & \dfrac{\partial N_2}{\partial y} & \dfrac{\partial N_2}{\partial y} & \cdots & \dfrac{\partial N_n}{\partial y} \end{array} \right] \left\{ \begin{array}{c} T_1 \\ T_2 \\ T_3 \\ \vdots \\ T_n \end{array} \right\} + \left[\begin{array}{c} \dfrac{\partial N_{01}}{\partial x} \\ \dfrac{\partial N_{01}}{\partial y} \end{array} \right] T_1 \{T_{01}\} \tag{15–14}$$

$$\left\{ \begin{array}{c} \dfrac{\partial T}{\partial x} \\ \dfrac{\partial T}{\partial y} \end{array} \right\} = [\boldsymbol{B}]\{\boldsymbol{T}_a\} + [\boldsymbol{B}_0]\{\boldsymbol{T}_{0a}\} \tag{15–15}$$

and

$$\left[\begin{array}{cc} \dfrac{\partial w}{\partial x} & \dfrac{\partial w}{\partial y} \end{array} \right] = \left\{ \begin{array}{c} \dfrac{\partial w}{\partial x} \\ \dfrac{\partial w}{\partial y} \end{array} \right\} = \{\boldsymbol{d}_a\}^T [\boldsymbol{B}]^T \tag{15–16}$$

Substituting into eqn. (15–9) and following the same procedure as in Chapter 11:

$$\int_V [\boldsymbol{B}]^T [\boldsymbol{K}][\boldsymbol{B}] dV \{\boldsymbol{T}_a\} = \int_V [\boldsymbol{N}]^T [\boldsymbol{N}] dV \{\boldsymbol{b}_a\} + \int_{S_h} [\boldsymbol{N}]^T [\boldsymbol{N}] dS \{\boldsymbol{h}_a\} - \int_V [\boldsymbol{B}]^T [\boldsymbol{K}][\boldsymbol{B}] dV \{\boldsymbol{T}_{0a}\} \tag{15–17}$$

By recognizing that we can integrate over each element and sum the integrals, we obtain:

$$\begin{aligned} \sum_{e=1}^{N_{\text{elm}}} \int_{V_e} [\boldsymbol{B}]^T [\boldsymbol{K}][\boldsymbol{B}] dV \{\boldsymbol{T}_a\} = & \sum_{e=1}^{N_{\text{elm}}} \int_{V_e} [\boldsymbol{N}]^T [\boldsymbol{N}] dV \{\boldsymbol{b}_a\} + \sum_{e=1}^{N_{\text{elm}}} \int_{S_h} [\boldsymbol{N}]^T [\boldsymbol{N}] dS \{\boldsymbol{h}_a\} \\ & - \sum_{e=1}^{N_{\text{elm}}} \int_{V_e} [\boldsymbol{B}]^T [\boldsymbol{K}][\boldsymbol{B}] dV \{\boldsymbol{T}_{0a}\} \end{aligned} \tag{15–18}$$

or

$$[\boldsymbol{K}]\{\boldsymbol{T}\} = \{\boldsymbol{Q}\} \tag{15–19}$$

where $[\boldsymbol{K}]$ is the conductivity matrix, $\{\boldsymbol{T}\}$ is the vector of unknown temperatures, and $\{\boldsymbol{Q}\}$ is the load vector consisting of contributions from the body heat sources, boundary fluxes, and essential boundary conditions.

The calculation of derivatives using the chain rule and the numerical integration follow the same procedures as used for elasticity.

15.6 Point Sources and Surface Fluxes

When point sources are present, the mesh is usually generated in such a way that there will be a node at each source. The load vector then is simply the value of the of the heat source (sink) at that node

$$\int_{S_h} [\boldsymbol{N}]^T [\boldsymbol{N}] dS \{\boldsymbol{h}_a\} \rightarrow \left\{ \begin{array}{c} h_1 \\ h_2 \\ \vdots \\ h_n \end{array} \right\} \tag{15–20}$$

As for surface tractions, we integrate the surface fluxes over the edge of an element.

Consider a curved element edge L_{123} passing 3 nodes (Q8 element) as show in the figure below:

Figure 15 2: Curved element edge

where the heat flux leaving the surface is $h = -\boldsymbol{q} \cdot \boldsymbol{n} = -q_x n_x - q_y n_y$.

Assuming that the element is traversed in a counter-clockwise direction (i.e. the material is on the left side of the boundary), then the tangent direction of an arc line element is

$$\boldsymbol{e}_s = \frac{dx}{ds}\boldsymbol{e}_1 + \frac{dy}{ds}\boldsymbol{e}_2 \tag{15–21}$$

and the outward normal to the boundary is

$$\boldsymbol{n} = \boldsymbol{e}_s \times \boldsymbol{e}_3 = \frac{dy}{ds}\boldsymbol{e}_1 - \frac{dx}{ds}\boldsymbol{e}_2 \tag{15–22}$$

The equivalent nodal fluxes corresponding to the three nodes on L_{123} are then

$$\left\{ \begin{array}{c} q_1 \\ q_2 \\ q_3 \end{array} \right\} = \int_{L_{123}} [\boldsymbol{N}]^T [\boldsymbol{N}] dS \{\boldsymbol{h}_a\} = t \int_{L_{123}} [\boldsymbol{N}]^T [\boldsymbol{N}] ds \{\boldsymbol{h}_a\}$$

$$= t \int_{L_{123}} \begin{Bmatrix} N_1 \\ N_2 \\ N_3 \end{Bmatrix} \lfloor N_1 \; N_2 \; N_3 \rfloor ds \begin{Bmatrix} h_1 \\ h_2 \\ h_3 \end{Bmatrix} \tag{15–23}$$

and changing the integration to be over the natural coordinates

$$\begin{Bmatrix} q_1 \\ q_2 \\ q_3 \end{Bmatrix} = t \int_{-1}^{1} \begin{Bmatrix} N_1 \\ N_2 \\ N_3 \end{Bmatrix} \lfloor N_1 \quad N_2 \quad N_3 \rfloor J d\xi \begin{Bmatrix} h_1 \\ h_2 \\ h_3 \end{Bmatrix} \tag{15–24}$$

here N_i are the 1D shape functions for the quadratic element.

$$N_1(\xi) = \frac{1}{2}\xi(\xi - 1) \tag{15–25}$$

$$N_2(\xi) = (1 - \xi^2) \tag{15–26}$$

$$N_3(\xi) = \frac{1}{2}\xi(\xi + 1) \tag{15–27}$$

and

$$ds = J d\xi \tag{15–28}$$

Expanding for ds:

$$ds = \sqrt{dx^2 + dy^2} = \sqrt{\left(\frac{dx}{d\xi}d\xi\right)^2 + \left(\frac{dy}{d\xi}d\xi\right)^2} = \sqrt{\left(\frac{dx}{d\xi}\right)^2 + \left(\frac{dy}{d\xi}\right)^2}\, d\xi \tag{15–29}$$

and

$$J = \sqrt{\left(\frac{dx}{d\xi}\right)^2 + \left(\frac{dy}{d\xi}\right)^2} \tag{15–30}$$

Using the isoparametric formulation

$$x = \sum_{i=1}^{3} N_i(\xi) x_i \tag{15–31}$$

$$y = \sum_{i=1}^{3} N_i(\xi) y_i \tag{15–32}$$

and we have

$$\frac{dx}{d\xi} = \sum_{i=1}^{3} \frac{dN_i(\xi)}{d\xi} x_i \tag{15–33}$$

$$\frac{y}{d\xi} = \sum_{i=1}^{3} \frac{dN_i(\xi)}{d\xi} y_i \tag{15–34}$$

The above integration can be readily evaluated using 3-point Gauss integration in 1D. For the case of a straight edge with the middle node at the center and a constant surface flux, the equivalent nodal fluxes are given by:

$$\left\{ \begin{array}{c} q_1 \\ q_2 \\ q_3 \end{array} \right\} = tL_{123}h \left\{ \begin{array}{c} \frac{1}{6} \\ \frac{4}{6} \\ \frac{1}{6} \end{array} \right\} \tag{15–35}$$

If the surface flux is the result of a film coefficient, then:

$$\{\boldsymbol{h}_a\} = h_{\text{film}}(\{\boldsymbol{T}\} - \{\boldsymbol{T}\}_\infty) \tag{15–36}$$

We can rewrite eqn. (15–23) as:

$$\{\boldsymbol{q}\} = [\boldsymbol{H}]\{\boldsymbol{h}_a\} \tag{15–37}$$

where

$$[\boldsymbol{H}] = t \int_{L_{123}} \left\{ \begin{array}{c} N_1 \\ N_2 \\ N_3 \end{array} \right\} \lfloor \begin{array}{ccc} N_1 & N_2 & N_3 \end{array} \rfloor ds \tag{15–38}$$

Then, for the case of a film coefficient, eqn. (15–37) becomes:

$$\begin{aligned} \{\boldsymbol{q}\} &= [\boldsymbol{H}]h_{\text{film}}\left(\{\boldsymbol{T}\} - \{\boldsymbol{T}_\infty\}\right) \\ \{\boldsymbol{q}\} &= h_{\text{film}}[\boldsymbol{H}]\{\boldsymbol{T}\} - h_{\text{film}}[\boldsymbol{H}]\{\boldsymbol{T}_\infty\} \end{aligned} \tag{15–39}$$

The first term on the right hand side provides coefficients that are subtracted from the global conduction matrix, since they multiply unknown nodal temperatures. The last term on the right hand side add to the load vector, since all coefficients are known.

Appendix A

Introduction to C++

A.1 Purpose

In this appendix, the student is introduced to the fundamentals of the C++ language as will be used to implement our finite element program. The approach taken is to give the complete code for a simple example and then to discuss the features of the code in detail.

I found the book *On to C++* by Patrick Henry Winston (Addison-Wesley Publishing Company) to be refreshingly brief and a quick introduction to the basic concepts of what can be accomplished with C++. However, this book also uses several bad practices, such as exposing object data publicly, so it should not be mimicked without thought.

A.2 Development Environment

Microsoft Visual Studio.Net is used as the development environment. To start a new project, select `File/New/Project...`. In the dialog below the project type is `Win32 Console`, a command window program that does not use graphics. I have placed the new project under my Classes directory and given the "Name" as "Geometry."

Figure A 1: Creating a new project

Selecting `OK` will create the new project. Note that this also creates a `Solution` which can encompass several projects. In our cases, there is essentially no difference. The new project will create the window as shown below. The main window displays the currently open file; the tree view on the right displays the files in the project, organized into the source files (`*.cpp`) and the header files (`*.h`). The lower part of the display includes locations for error information and debugging.

Figure A 2: The new project

Be sure to save all your work in at least two locations. When you come back to the project to continue your work, you can select `File/Open Solution...` and reopen the solution.

After your project is created, turn off the precompiled header option by selecting `Project/Properties...` and under `C/C++` select the `Precompiled Headers` option and for the `Create/Use Precompiled Header` select the `Not Using Precompiled Headers` option in the pulldown menu. Select `OK` to close the dialog. Precompiled headers can speed compilation of complex projects, but just complicate things and are not needed for our work.

Figure A 3: Turn off precompiled header option

A.3 Adding a New File to the Project

New files (`*.cpp` and `*.h`) can be added to the project by selecting `Project/Add New Item...`. To create a source file (`*.cpp`), make sure that the icon is selected. Then give a name (in this case `triangle`) and select `Open`. This will create a new blank file. The same is done to create the matching header (`*.h`) file.

Figure A 4: Creating a new source file

A.4 Example Program

Our example program will do the following tasks:

- Open an input file
- From the input file, read data for a fixed number of circles and triangles
- Echo the input data to an output file
- Calculate the total area
- Print the total area

Our discussion of the program will begin at the bottom and work up.

The basic concept in C++ is an object. An object typically represents some meaningful collection of data and operations that can be performed on that data. In this example, we will use circle and triangle objects.

The data for the circle will be the center and radius. The data for the triangle object will be the three coordinates of the corner points. This data will be private, that is, the outside world will not be able to access the circle or triangle data. If we wanted the external world to access the data, it would be done using a function that returns the data, for example, to get the radius of the circle, we could have a `getRadius()` function that returns the radius. Establishing this barrier between the data and the outside world is an example of

Data Abstraction. This means that the actual data stored in the object may be different that as exposed to the world. For example, we might actually store the diameter for the circle, but when the `getRadius()` function is called, we divide the diameter by 2 and return that value. The advantage of *Data Abstraction* is that we are now free to change and optimize the actual implementation of an object without affecting any of the outside code that uses the object.

In addition to data, an object also has functions that define an interface to the outside world. In this example program, the circle and triangle will be able to calculate their area and print themselves using the `getArea()` and `print()` functions. So, given any circle or triangle we can ask for the area. If we wanted additional capability, we would add new functions to do this.

Objects can be organized in hierarchies, in which subclasses inherit the properties of a superclass. *Duplication of code is one of the most reliable ways of introducing errors into your software*. You fix the error in one location but forget to fix it in the second location. Using hierarchies allows us to organize our objects so that shared data and functionality is located in the superclass, avoiding duplication of code. In our example, we will store the center of each shape in the superclass.

Figure A 5: Hierarchical organization of shapes

Organizing our software in objects allows us to organize complex software into logically coherent modules. By preventing access to the interior data and defining the interface to the outside, we now have the ability to test the object and ensure that it operates correctly. This greatly increases the reliability of complex software.

A.4.1 Input File

The example input file is given below. The data consists of two circles and one triangle. It is assumed that the input file is correct. The circle data consists of the *XY* center and radius. The triangle data consists of three *XY* pairs, assumed to be oriented CCW.

```
CIRCLE     5.   2.   4.
CIRCLE     1.   1.   2.
TRIANGLE   0.   0.   1.   0.  0.5   2.
```

A.4.2 Output File

This is the output file. Basically it just echoes the input data and shows the total area.

```
Circle: Center X= 5 , Y= 2 , Radius= 4
Circle: Center X= 1 , Y= 1 , Radius= 2
Triangle:   X1= 0 , Y1= 0 , X2= 1 , Y2= 0 , X3= 0.5 , Y3= 2

Total area = 63.83
```

A.4.3 Formatting

The following formatting rules are followed:

- Object (class) names begin with a capitol letter
- Function and data names begin with lower case
- Names of class data begin with "`d_`"
- Names use capitals for multi-word names
- Appropriate indention is always used

A.5 Class Objects

Here we list the entire program. As you can see, the program has been divided into discrete modules. The modules are:

- `Circle` and `Triangle` classes. These both derive from the Shape superclass. The Circle and Triangle classes hold their unique data, while the `Shape` superclass stores the center coordinate data. When an instance of a `Circle` or `Triangle` is created, the constructor initializes the data in the Shape class.
- The `Shape` class is the highest level class for the geometry objects. It includes the specification of two virtual classes to calculate the area of a shape and to print shape data. All virtual functions must be defined in the subclasses. As a result, we are guaranteed that the `Circle` and `Triangle` classes have a way to calculate their areas, or the compiler would complain that a virtual function was missing.
- The `Data` class holds all the model data. It has functions to read the data from the input file and to loop through all shapes to calculate the area and print the data.
- The `Geometry` class has the `main()` function. Basically, `main()` gets things going and controls the highest level of execution. Class files come in pairs of the header file (`*.h`) that defines the functions and data in a class, and the source file (`*.cpp`) that actually implements the functions defined in the header. When we are writing a class that uses another class, we include the header file for that other class to obtain information about that class.

A.6 Listing of Example Problem Files

Circle class header file (`circle.h`):

```
// Class definition for Circle.

#ifndef circle_h    // Prevents redundant inclusions of the circle.h file
```

```
#define circle_h    // On the first include, sets the flag

#include "shape.h"  // We need Shape information below

class Circle : public Shape // Indicates that Shape is a superclass
{
  public:   // All the following data is public to the outside world

    // Constructors and Destructors are called when an object is created and destroyed
    // They are used to perform initial checks or saving of data when the object
    // is created and to delete memory allocation if needed on destruction.
    Circle(double x, double y, double radius);  // This is the constructor.  It must
                                                // be given three arguments
    ~Circle();                                  // This is the destructor

    // Functions
    virtual double getArea() const;     // This function returns the area.  The const
                                        // indicates that it does not change any data.
    virtual void printData(ostream &out) const;   // Prints the circle data to output.

  private:  // All the following can not be accessed by the outside world.

    // Data
    double d_radius;    // This is private class data.
};

#endif  // End of the redundant include test
```

Circle class source file (`circle.cpp`):

```
// Circle object

#include "circle.h"

// Constructor.  We use it to save the data and set the center in the shape.
Circle::Circle(double x, double y, double radius)
{
    // Save the object data
    d_radius = radius;

    // Set the center shape data
    setCenter(x, y);
}

// Destructor.   Does nothing.
Circle::~Circle()
{
}

// Calculate area and return.
double Circle::getArea() const
{
    return 3.1415*d_radius*d_radius;
}

// Print the circle data to a file
void Circle::printData(ostream &out) const
{
    out << " Circle: ";
```

```
    //out.width(8);
    out << "Center X= " << getCenterX() << " , Y= " << getCenterY();
    out << " , Radius= " << d_radius;
    out << endl;
}
```

Triangle class header file (`triangle.h`)

```
// Class definition for Triangle.

#ifndef triangle_h
#define triangle_h

#include "shape.h"

class Triangle : public Shape
{
  public:
    //Constructors and Destructors
    Triangle(double x1, double y1,
             double x2, double y2,
             double x3, double y3);
    ~Triangle();

    // Functions
    virtual double getArea() const;
    virtual void printData(ostream &out) const;

  private:
    // Data
    double d_x1, d_y1;  // Coordinates of the three corners
    double d_x2, d_y2;
    double d_x3, d_y3;
};

#endif
```

Triangle class source file (`triangle.cpp`)

```
// Triangle object

#include "triangle.h"

// Constructor
Triangle::Triangle(double x1, double y1,
                   double x2, double y2,
                   double x3, double y3)
{
    // Save the object data
    d_x1 = x1;
    d_y1 = y1;
    d_x2 = x2;
    d_y2 = y2;
    d_x3 = x3;
    d_y3 = y3;

    // Save the shape center data
```

```
    double xcen = (x1+x2+x3)/3.;
    double ycen = (y1+y2+y3)/3.;
    setCenter(xcen, ycen);
}
// Destructor
Triangle::~Triangle()
{
}

double Triangle::getArea() const
{
    // Equation for area of triangle
    return 0.5*((d_x1*d_y2 + d_x2*d_y3 + d_x3*d_y1) -
                (d_y1*d_x2 + d_y2*d_x3 + d_y3*d_x1));
}

void Triangle::printData(ostream &out) const
{
    out << " Triangle: ";
    //out.width(8);
    out << "  X1= " << d_x1 << " , Y1= " << d_y1;
    out << " , X2= " << d_x2 << " , Y2= " << d_y2;
    out << " , X3= " << d_x3 << " , Y3= " << d_y3;
    out << endl;
}
```

Shape header file (`shape.h`)

```
// Class definition for Shape.

#ifndef shape_h
#define shape_h

#include <fstream>
using namespace std;

class Shape
{
  public:
    //Constructors and Destructors
    Shape();
    ~Shape();

    // Functions
    double getCenterX() const;  // Provides a way to get the center X coordinate
                                // even though the data is private and can not be
                                // accessed directly.
    double getCenterY() const;  // Provides a way to get the center Y coordinate
    virtual double getArea() const = 0; // The virtual keyword indicates that the
                                        // getArea() functions will be defined at a
                                        // lower class.  The = 0 makes it a pure
                                        // virtual function, so no instances of Shape
                                        // may be created,  only classes derived
// from Shape.
    virtual void printData(ostream &out) const = 0;

  protected:
    // Protected data can be accessed by derived classes.  This allows Circle and
    // Triangle to set the center for a shape.  Only a derived class can set this data.
```

```
    void setCenter(double x, double y);

  private:
    // Data
    double d_cenX, d_cenY;  // Coordinates of center of shape
};

#endif
```

Shape source file (shape.cpp)

```
// Shape object

#include "shape.h"

// Constructor
Shape::Shape()
{
}

// Destructor
Shape::~Shape()
{
}

void Shape::setCenter(double x, double y)
{
    d_cenX = x;
    d_cenY = y;
}

double Shape::getCenterX() const
{
    return d_cenX;
}

double Shape::getCenterY() const
{
    return d_cenY;
}
```

Data header file (data.h)

```
// Class definition for Data

#ifndef data_h
#define data_h

#include "shape.h"

#define MAXSHAPES 10    // Now we just change this one location from 10 to another
                        // number and our memory allocation changes everywhere.

class Data
{
  public:
    // Constructors and Destructors
```

```
    Data();
    ~Data();

    // Functions
    void readData(istream &inp);
    void writeData(ostream &out);
    double getTotalArea();

  private:
    // Data
    int     d_numShapes;            // Number of shapes, this depends on how many we read
    Shape   *d_shapes[MAXSHAPES];   // Array of pointers to Shape objects.  Since both
                                    // Circle and Triangle are Shape objects (Shape is
                                    // their superclass)we can store pointers to either
                                    // a Circle or Triangle in this array.  Here we
                                    // limited the array size to a fixed number.  We
                                    // will use dynamic memory allocation in our finite
                                    // element program.
};

#endif
```

Data source file (`data.cpp`)

```
// Data object

#include "data.h"
#include "Circle.h"
#include "Triangle.h"

// Constructor
Data::Data()
{
}

// Destructor
Data::~Data()
{
}

// Read the shape data
void Data::readData(istream &inp)
{
    // Local variables
    char    buffer[10];

    // Now read each shape from the data file.
    // Depending on whether it is a Circle or Triangle, we create the
    // appropriate object.  The new operator creates and object and
    // returns a pointer to that object.

    d_numShapes = 0;
    while((inp >> buffer) && (d_numShapes < MAXSHAPES)) // Keep reading until either
                                                        // we hit the end of file
                                                        // or until we have no more
                                                        // storage.  The first item
                                                        // we read is the name into buffer.
    {
        if(strcmp(buffer, "CIRCLE") == 0)               // If a Circle, read and
```

```
                                                    // create a Circle.
            double x, y, r;
            inp >> x >> y >> r;                     // Read circle data from file.
            d_shapes[d_numShapes] = new Circle(x, y, r);// Create a circle and store
                                                    // its pointer.
            d_numShapes++;                          // Increment the shape counter.
        }
        else if(strcmp(buffer, "TRIANGLE") == 0)    // Same for Triangle
        {
            double x1, y1, x2, y2, x3, y3;
            inp >> x1 >> y1;
            inp >> x2 >> y2;
            inp >> x3 >> y3;
            d_shapes[d_numShapes] = new Triangle(x1, y1, x2, y2, x3, y3);
            d_numShapes++;
        }
    }
}

// Write the shape data.  We have an array of pointers to Shape objects.
// All we do is loop through each object and tell it to write its own data.
// Becuase the Shape class had a pure virtual method called writeData(), we
// are guaranteed that the derived class will implement this function (or the
// compiler will complain).  Since the object knows whether it is a Circle or
// Triangle, it calls the appropriate function.  Handling all Shapes in the
// same manner uses the property of polymorphism.
void Data::writeData(ostream &out)
{
    for(int i=0; i<d_numShapes; i++)
    {
        d_shapes[i]->printData(out);
    }
}

// This function calculates the total area.
double Data::getTotalArea()
{
    double area = 0.;

    for(int i=0; i<d_numShapes; i++)
    {
        area += d_shapes[i]->getArea();
    }
    return area;
}
```

Geometry class source file (`geometry.cpp`)

```
// Geometry.cpp : Defines the entry point for the console application.

#include <fstream>
#include "data.h"
using namespace std;

int main()
{
    Data    data;
    ifstream input_file;
    ofstream output_file;
```

```
    // Create the input file stream
    input_file.open("shapes.dat", ios_base::in);
    output_file.open("shapes.out");

    // Read and write data
    data.readData(input_file);
    data.writeData(output_file);

    // Write total area
    output_file << endl << " Total area = " << data.getTotalArea();

    // Close files
    input_file.close();
    output_file.close();

    return 0;
}
```

Appendix B

Computer Implementation

B.1 Purpose

T he goal of this lecture is to lay the framework for an object-oriented finite element program. We will implement the simplest program that still illustrates the general approach.

B.2 Design of the Finite Element Program

The general flow of a linear finite element solution is shown in Figure B–1.

Figure B–1 shows the solution procedure as a linear process for which a procedural design is appropriate. While such a design can certainly be implemented, an object-oriented approach offers the potential for future expansion and easier maintenance.

In object-oriented design, we define objects, each with their own data and functions. These objects interact with each other through public interfaces. Ideally, such an approach allows each object to be independent of other objects and enhances reliability and maintainability.

With respect to the finite element program, some objects are intuitive, such as `femNode` and `femElement` objects. Also, examining Figure B–1, it also seems natural to have an object that stores all the problem data (`femData`) and a separate object that accesses the data to perform the solution (`femSolve`). As our discussion evolves, other classes will be identified. Also, note the format used for our class names; the first three letters identify a grouping of classes, followed by the class name with the first letter capitalized. In C++, the grouping would typically be a subproject in a larger project and the files would be found in a subdirectory with the same name (*i.e.*, `fem` for finite element classes, `ult` for utilities classes that are used to aid the computation). For a variable or function, the first letter is lower case, followed by upper case letters to help readability (as appropriate). For

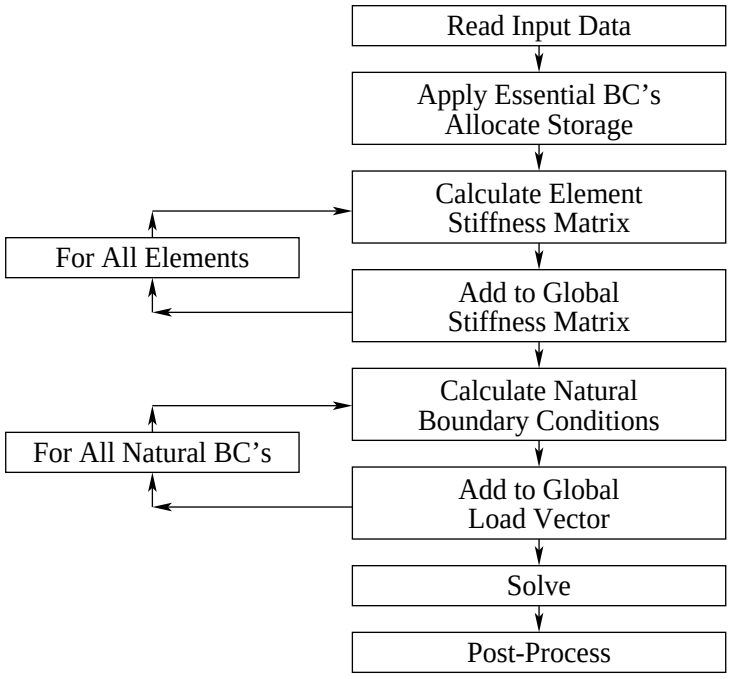


Fig. B–1 Flow of finite element solution

example, `getElementK()` is a function and `d_material` is a data member variable in a class (the data of a class begins with `d_`).

The `femElement` class will be the base class of all elements. From the element class we will derive elements of different types, for example eight-noded quadrilateral elements (Q8) and six-noded triangular elements (T6). Please remember that this is our simple implementation, a more general implementation could separate the element functions (interpolation, integration) from the physics of the problem. In this simple implementation, we are assuming all elements solve only one type of differential equation. The elements will be organized as shown in Figure B–2.

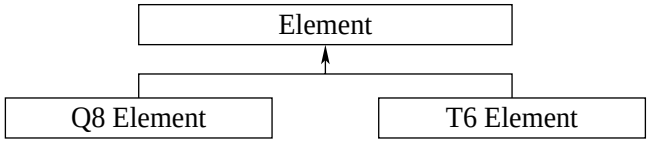


Fig. B–2 Element classes.

The reason we define a base class and derive element types from it, is that this allows

us to create different element types when we read in the data, but to still access each element in a common manner (polymorphism). Let us be specific. In the `femData` class we will have an array of element pointers. The length of the array is the number of elements (`d_numElems`). We dynamically allocate only the required memory for our particular problem.

```
femElement **elems;
elems = new femElement*[d_numElems];
```

Then, as we are reading our data, we can create new elements that are either Q8 or T6, for example:

```
elems[5] = new femElementQ8();
elems[6] = new femElementT6();
etc.
```

Each element in our array is now a specific element type. The power of this approach is that we can now access the virtual functions defined in the `femElement` interface without concern for what kind of element it is. For example, when assembling the global stiffness matrix, we can obtain each element stiffness as follows:

```
for(int i=0; i<d_numElems; i++)
{
    elems[i]->getElementK(ke);
}
```

where `ke` is a matrix where we can store the element stiffness matrix (it is passed by reference so data can be modified and accessed). The point is that we no longer care what type of element we are dealing with, we just tell each element to do its appropriate task. This same pattern is used for all the data for which we do not know the number of items we will have until we read the problem data. This includes nodes, elements, materials, essential boundary conditions, point boundary conditions, and natural boundary conditions.

In a more advanced implementation, these items would be accessed using iterators. In our implementation `femData` will have a function that returns the number of items and then provides access to a particular item. For example, to loop through each node and get the node coordinates, we use the following example code.

```
femData dat;
femNode* node;
double x, y;
for(int i=0; i<dat.getNumNodes(); i++)
{
    node = dat.getNode(i);
    x = node->getX();
    y = node->getY();
}
```

Finally, an absolute design rule is that **NO** data is public.

B.3 Class Definitions

B.3.1 femElement Class

Description

The `femElement` class is the base class for all elements. The `femElement` class defines pure virtual functions that define the external interface. It also stores information common to all element types.

The basic information contained in the `femElement` class is an id, a pointer to a material, the number of nodes in the element, and an array of pointers to the nodes. When the element is constructed, all the data is passed. It is possible to access the node id's, which is needed when post-processing. All other functions are pure virtual and defined in the specific element type.

Definition

```
class femElement

{

  protected:
    int d_id;
    femMaterial* d_material;
    int d_numNodes;
    femNode** d_nodes;

    //never to be used constructors
    femElement();
    femElement(const femElement &elem);

  public:

    // Constructors
    femElement(int id, femMaterial* mat, int numNodes, femNode* nodes[]);

    //Destructor
    virtual ~femElement();

    // Functions
    int getID() const;
    int getNumNodes() const;
    void getNodes(femNode* nodes[]) const;
    void printData(ostream &out) const;
```

```
    virtual int getNumDofs() const = 0;
    virtual int getBandwidth() const = 0;
    virtual void getElementK(utlMatrix &k, femDof* dofs[],
        ostream &out) = 0;
    virtual void calcData() = 0;
    virtual void getNodalData(utlStress2D stresses[]) = 0;
    virtual void printResults(ostream &out) const = 0;
};
```

B.3.2 femElementQ8 Class

Description

We will only illustrate a single element derived from the base class, the Q8 element. However, such a derivation allows the easy addition of other element types, such at the T6 element.

The `femElementQ8` class implements the eight-noded quadrilateral element. It is derived from the `femElement` base class.

Definition

```
class femElementQ8: public femElement
{
  private:
    utlStress2D d_stress[3][3];

    // Never to be used constructors.
    femElementQ8();
    femElementQ8(const femElementQ8 &elem);

  public:

    // Constructors
    femElementQ8(int id, femMaterial* mat,
        int numNodes, femNode* enodes[]);

    // Destructor
    virtual ~femElementQ8();

    // Functions
    virtual int getNumDofs() const;
    virtual int getBandwidth() const;
    virtual void getElementK(utlMatrix &k, femDof* dofs[],
        ostream &out);
    void calcData();
    void getNodalData(utlStress2D stresses[]);
    virtual void printResults(ostream &out) const;
```

```
  private:
    void getBMatrix(double r, double s, utlMatrix &b, double &detj);
};
```

B.3.3 femNode Class

Description

The `femNode` class stores geometry, equation numbers for each degree of freedom (dof), the solution, and boundary condition data. In this implementation, we illustrate two dof's at the node, `UX` and `UY`. The data for these dofs is stored in arrays. The appropriate value is accessed using an enumerated type as follows:

```
enum DOF {UX=0, UY};
```

For example, we can then return the UX dof value by:

```
return d_dof[UX];
```

Definition

```
class femNode
{
  private:
    int d_id;
    double d_x, d_y;
    femDof* d_dof[2];

    // Never used constructors
    femNode();
    femNode(const femNode& node);

  public:

    // Constructors
    femNode(int id);

    // Destructor
    virtual ~femNode();

    // Functions
    int getID() const;
    void setCoords(double x, double y);
    double getX() const;
    double getY() const;
    femDof* getDof(DOFType dof) const;
    void printData(ostream &out) const;
    void printResults(ostream &out) const;
};
```

B.3.4 femDof Class

Description

The `femDof` class implements a degree of freedom. Each dof has an equation number with associated value. Dof's with either no boundary condition or a natural boundary condition are active. If the dof has an essential boundary condition, the dof is not active. During matrix assembly, if the boundary condition is essential, the appropriate element stiffness coefficient is multiplied by the boundary condition value and subtracted from the global load vector. If it is a natural boundary condition the value is added to the load vector. After the solution, the results are stored in the value.

Definition

```
enum DOFType {UX=0, UY};

class femDof
{
  private:
    double d_value;
    int d_eqn;
    bool d_active;
    //never used copy constructor
    femDof (femDof &dof);

  public:
    // Constructors
    femDof();
    //Destructor
    virtual ~femDof();
    // Functions
    void setNotActive();
    bool isActive() const;
    int setEqn(int currEqn);
    int getEqn() const;
    void setValue(double value);
    double getValue() const;
    void print(ostream &out) const;
};
```

B.3.5 femData Class

Description

The `femData` class stores all problem data. It provides a means to access all elements, nodes, material, and boundary condition data. Memory for all data is allocated dynamically

after the size of the required storage is known. Simple arrays are used for the nodes and materials. These are allocated by:

```
d_nodes = new femNode[d_numNodes];
```

Because we want to use polymorphism for the elements, we allocate an array of element pointers by:

```
d_elems = new femElement*[d_numElems];
```

We then save a particular element using:

```
m_elems[i] = new femElementQ8();
```

The class also returns pointers to requested nodes and elements. In a more advanced implementation, this would be accomplished using iterators.

Definition

```
class femData
{
  private:
    femNode** d_nodes;
    femMaterial** d_matprops;
    femElement** d_elems;
    femEssentialBC** d_essentialBCs;
    femPointBC** d_pointBCs;
    femNaturalBC** d_naturalBCs;
    char* d_title;
    int d_numNodes;
    int d_numMats;
    int d_numElems;
    int d_probType;
    int d_numEssentialBCs;
    int d_numPointBCs;
    int d_numNaturalBCs;
    int d_neq;
    int d_bandw;

  public:
    // Constructors
    femData();
    //Destructor
    virtual ~femData();
    // Functions
    void readData(istream &inp);
    void writeData(ostream &out);
    int getNumNodes();
```

```
    int getNumElems();
    int getNumEssentialBCs();
    int getNumPointBCs();
    int getNumNaturalBCs();
    femNode *getNode(int node);
    femElement *getElem(int elem);
    femEssentialBC *getEssentialBC(int ebc);
    femPointBC *getPointBC(int pbc);
    femNaturalBC *getNaturalBC(int nbc);
    void writeResults(ostream &out);
    void writePlotFile(ostream &plt);
};
```

B.3.6 femEssentialBC Class

Description

The `femEssentialBC` class stores essential boundary condition data.

Definition

```
class femEssentialBC
{
  private:
    femNode* d_node;
    DOFType d_dof;
    double d_value;

    //never used constructors
    femEssentialBC();
    femEssentialBC (femEssentialBC &ebc);

  public:
    // Constructors
    femEssentialBC(femNode* node, DOFType dof, double value);
    //Destructor
    virtual ~femEssentialBC();
    // Functions
    femNode* getNode() const;
    DOFType getDof() const;
    double getValue() const;
    void printData(ostream &out) const;
};
```

B.3.7 femNaturalBC Class

Description

The femNaturalBC class stores natural boundary condition data. Data is given for two corner nodes of an element. To use this, a search is performed to identify the element. Then the value is assumed to be positive acting away from the element.

Definition

```
class femNaturalBC
{
  private:
    femNode* d_node1;
    femNode* d_node2;
    femNode* d_node3;
    double d_value;

    //never used constructors
    femNaturalBC();
    femNaturalBC (femNaturalBC &ebc);

  public:
    // Constructors
    femNaturalBC(femNode* node1, femNode* node2, femNode* node3, double value);
    //Destructor
    virtual ~femNaturalBC();
    // Functions
    void printData(ostream &out) const;
    virtual void getNaturalBCF(utlVector &f, femDof* dofs[],
        ostream &out);
};
```

B.3.8 femPointBC Class

Description

The femPointBC class stores point boundary condition data (a special case of natural boundary condition data).

Definition

```
enum PointBCType {FX=0, FY};

class femPointBC
{
  private:
    femNode* d_node;
```

```
    PointBCType d_dof;
    double d_value;

    //never used constructors
    femPointBC();
    femPointBC (femPointBC &ebc);

  public:
    // Constructors
    femPointBC(femNode* node, PointBCType dof, double value);
    //Destructor
    virtual ~femPointBC();
    // Functions
    femNode* getNode() const;
    PointBCType getDof() const;
    double getValue() const;
    void printData(ostream &out) const;
};
```

B.3.9 femSolve Class

The `femSolve` class manages the solution. The idea is to keep the solution independent of the problem data and storage.

Description

This class manages the solution, including assembly of the global stiffness matrix and load vector.

Definition

```
class femSolve
{
  private:
    utlMatrix *m_k;
    utlVector *m_f;
    utlVector *m_sol;
    int m_neq;
    int m_bandw;

    int setEquationNumbers(femData &dat);
    int getBandwidth(femData &dat);
    void assembleRHS(femData &dat, utlVector *f, ostream &out);
    void assembleK(femData &dat, utlMatrix *k, utlVector *f,
        ostream &out);
    void saveSolution(femData &dat, utlVector *sol);
    void calcData(femData &dat);
```

```
  public:
    // Constructors
    femSolve();
    //Destructor
    virtual ~femSolve();
    void solve(femData &dat, ostream &out);

};
```

B.3.10 femMaterial Class

Description

The `femMaterial` class is the base material class. It provides the interface to all materials.

Definition

```
class femMaterial
{
  protected:
    int d_id;
    //Never used constructors.
    femMaterial();
    femMaterial(const femMaterial &mat);

  public:
    //Constructors
    femMaterial(int id);

    //Destructor
    virtual ~femMaterial();

    //Functions
    int getID() const;
    virtual void getDMatrixStress(utlMatrix &d) = 0;
//  virtual void getDMatrixHeat(utlMatrix &d) = 0;
    virtual void printData(ostream &out) const = 0;
};
```

B.3.11 femMaterialElastic Class

Description

The `femMaterialElastic` class implements an elastic material.

Definition

```
class femMaterialElastic: public femMaterial
{
  protected:
    double d_e;     //young's modulus
    double d_nu;    //poisson's ration
    double d_k;     //conductivity
    double d_thick; //thickness

    // never used copy constructor
    femMaterialElastic(const femMaterialElastic &mat);

  public:
    // Constructors
    femMaterialElastic();
    femMaterialElastic(int id, double e, double nu,
        double thick);

    //Destructor
    virtual ~femMaterialElastic();

    // Functions
    virtual void printData(ostream &out) const;
    virtual void getDMatrixStress(utlMatrix &d);
//  virtual void getDMatrixHeat(utlMatrix &d);
};
```

B.3.12 femGauss3Pt Class

Description

The `femGauss3Pt` class provides three gauss point data.

Definition

```
class femGauss3Pt
{
  private:
    int d_numgp;
    double d_point[3];
    double d_weight[3];

  public:
    //Constructor
    femGauss3Pt();

    //Destructor
    virtual ~femGauss3Pt();
```

```
    // Functions
    int getNumPts() const;
    double getGaussPt(int gp) const;
    double getWeight(int gp) const;
};
```

B.3.13 utlMatrix Class

Description

The `utlMatrix` class is the matrix class. In this implementation, in order to simplify the programming task, we are not to use the sparse and banded nature of the global stiffness matrix, nor will we consider the advantages of the symmetry of the stiffness matrices. Hence, we implement a very basic class of matrix that handles most of the square matrices that are used in the program.

Definition

```
class utlMatrix
{
  protected:
    double **m_coeff;       // Coefficients of matrix
    int m_nr, m_nc;        // Number of columns and rows
    // Never to be used copy constructor
    utlMatrix(const utlMatrix &matrix);
    // Never to be used Constructor
    utlMatrix();
  public:
    //Destructor
    virtual ~utlMatrix();
    // Functions
    void zero();
    int getNumRows() const;
    int getNumCols() const;
    void print(ostream &out);
    virtual void setCoeff(int i, int j, double value) = 0;
    virtual void addCoeff(int i, int j, double value) = 0;
    virtual double getCoeff(int i, int j) = 0;
    virtual void mult(utlMatrix &b, utlMatrix &c) = 0;
    virtual void mult(utlVector &b, utlVector &c) = 0;
    virtual void trans(utlMatrix &a) = 0;
    virtual void gauss(utlVector *b, utlVector *x) = 0;
};
```

B.3.14 utlVector Class

Description

The `utlVector` class stores a vector.

Definition

```
class utlVector
{
  protected:
    double *m_coeff;        // Coefficients of matrix
    int m_nr;               // Number of rows
    //never to be used copy constructor
    utlVector(const utlVector &vector);
  public:
    // Constructors
    utlVector();
    utlVector(int nr);

    //Destructor
    virtual ~utlVector();
    // Functions
    void zero();
    int getNumRows() const;
    void setCoeff(int i, double value);
    void addCoeff(int i, double value);
    double getCoeff(int i);
    void print(ostream &out);
};
```

B.3.15 utlStress2D Class

Description

The `utlStress2D` class stores stress data.

Definition

```
class utlStress2D
{
  protected:
    // stress data
    double m_sigxx, m_sigyy, m_sigzz, m_sigxy;
    // never used constructors

    utlStress2D(utlStress2D&);
  public:
    // constructors
```

```
    utlStress2D();
    utlStress2D(double xx, double yy,
                double zz, double xy);
    // destructor
    virtual ~utlStress2D();
    //functions
    void setSigXX(double xx);
    void setSigYY(double yy);
    void setSigZZ(double zz);
    void setSigXY(double xy);
    void addSigXX(double xx);
    void addSigYY(double yy);
    void addSigZZ(double zz);
    void addSigXY(double xy);
    double getSigXX() const;
    double getSigYY() const;
    double getSigZZ() const;
    double getSigXY() const;
    void zero();
};
```

B.4 Psuedo-Codes of Key Functions

B.4.1 femElementQ8::getElementK

```
  obtain material matrix D;
  initialize matrix K;
  for every Gauss point (two loops, over zeta and eta) {
  calculate Jacobian and B matrix;
  calculate integrands for each element of the K matrix;
  multiply integrands with Gauss weight;
  add the resulting matrix to the K matrix;
}
```

B.4.2 void femSolve::assembleK

```
  for each element {
    create a temporary DOF array of 2*d_numNode
    calculate element stiffness matrix Ke
    for each DOF in the returned array of DOFs {
      if (DOF is Active): add the row and column of Ke to corresponding positions in K
                          (hint: need to check whether each node is active or not)
      else: corresponding row of f -= Ke corresponding column * Natural BC value
}
```

B.4.3 femNaturalBC::getNodalNaturalBC

```
set tn = d_value;
for each gauss point {
calculate shape function N[] (Note: This is 1-D 3-node quadratic shape function)
calculate derivative of shape function dN/d [];
calculate  and
for each of 3 nodes i {
$fx[i] += N[i]* tn *  * weight * thickness$
$fy[i] -= N[i]* tn *  * weight * thickness$
}
}

repack fx and fy into f (6-row vector)
```

B.4.4 Algorithm for constructing elements and nodes

```
  read in problem info (first line of data);
  allocate memory for double pointers;
  construct node objects (without assigning coordinates)
  for each element {
  read in data
  construct element object
  }
  for each node {
    read in data
    assign coordinates to corresponding nodes
}
```

B.4.5 femSolve::solve

```
    - find the total number of equations
    - assign equation number to each active DOF sequentially
    - assemble the global stiffness matrix and load vector
    - solve the equation using the Gaussian elimination
    - store the solution data (displacements) into node/DOF objects
- calculate stresses and perform other data for post processing.
```

Appendix C

Input and Output Data Formats

C.1 Purpose

The goal of this lecture is to lay the framework for an object-oriented finite element program. We will implement the simplest program that still illustrates the general approach.

C.2 Tutorial on Using CASCA to Create Input Data Files

C.2.1 Benchmark Problem

The benchmark problem is a plate with a hole in the center, as shown figure 1(a) below. The plate is under remote uniaxial tension P. The width (2W) and height (2H) of the plate are both 10 meters, and the radius of the hole is 1 meter. Traction P is 50 MN per unit length per unit depth. The Young's modulus of the material is 210 GPa, and the Poisson's ratio is 0.3. Treat the problem as plane strain.

Because of symmetry, only a quarter of the plate needs to be modeled, see figure 1(b).

C.2.2 About CASCA Program

CASCA is an interactive program for the creation of two-dimensional continuum finite element meshes. Its capabilities include three- and six-noded triangles and four and eight-noded quadrilaterals.

This manual is intended to be a simple description of the basic features of CASCA. It includes a brief tutorial, hints on addressing common problems, and menu descriptions.

Within the CASCA program, all user commands are made by clicking the mouse on one of the options displayed in the menu window (Figure 1). A message window is always present to prompt the user on the next step in the requested procedure. Entry into CASCA

and I/O operations invoked during the running of CASCA are made from the program control window. This is the window from which the program was started. An auxiliary window is used for attribute definition.

Figure 1: The CASCA window system

The CASCA program uses two types of cursors. The normal cursor has the shape of an arrow. When you see this cursor it means that the program is waiting for you to select a menu option or some other graphical input. The second cursor is a stylized wristwatch. When you see this cursor it means that either the program is processing data (e.g., generating a mesh), or it is waiting for input in the program control window.

The coordinate system used within the program is always fixed so that the x coordinate is horizontal, increasing to the right and the y coordinate is vertical, increasing going up.

CASCA was originally developed by Paul Wawrzynek and Louis Martha at Cornell University as a test bed for a hierarchical data base that was later implemented in FRANC3D. However, it has proven to be a useful tool, so it has continued to live beyond its initial application.

The fact that CASCA was originally written in 1987 is reflected in the menu structure. The menus were designed for display using low level XWindows commands. A rewrite of these low level commands has allowed us to port the code to Windows 95/NT in a simple fashion. The advantage is that the Unix and Windows 95/NT versions are identical; the disadvantage is that the program does not use dialog boxes or pull-down menus.

C.2.3 Step-by-Step Tutorial on Benchmark Problem

Double click the CASCA executable (”casca.exe”). You will be asked whether you want a big or small window, type s or b. At this point a new CASCA window will be created. Set Scale: Setting an Appropriate Data Space Initially you will have only three options: setting the data space (Set Scale), reading a restart file (Read), and adjusting your view (RESET, MAGNIFY, ZOOM, PAN, and SNAP). Because we are starting a new problem from scratch, we must select Set Scale.

In the Set Scale page you can change the world coordinates in the operations (mesh) window. By default, the window is 12 units wide by 12 units high with a grid spacing of one unit and the center of the grid at X=0.0, Y=0.0. This is satisfactory for our problem, so just select RETURN. Geometry: Creating the Problem Outline

Grid lines are available to simplify geometry construction. Select Grid from the menu. The X in the middle of the screen marks the center of the grid (presently X=0.0, Y=0.0). When the grid is displayed, a selection near a grid point will snap-to the grid. In addition to grid snap-to, snap-to is always active to the ends of lines that the user has defined.

From the main menu, you should see a Geometry option. Geometry is constructed by connecting edges into closed faces. Presently, only lines, circles, and arcs are available.

These allow you to specify the outline of your problem, which is the geometry used when generating a mesh. Select Geometry.

You are now presented with a number of options that you can use to specify the outline of your object. For the plate with a hole problem, we will begin with the hole. To take advantage of symmetry, we will mesh only a quarter of the plate. First select Get Circle. By default, the ARC option is active. An arc is specified in the clockwise direction by three points: beginning, ending, and the center. Click the (1, 0), (0, 1) and (0, 0) points in turn, and an arc is created. (you can also use KEYPAD option to do it.) Select DONE to indicate that you are finished defining the arc. If you select QUIT, the circle will be ignored.

To complete the plate outline we will use the Lines Connected option. Select this option. To start the connected lines, click on the top point of the circle arc (1, 0). Then move up to (5, 0), click, move to (5, 5), click, (5, 0), click, and, finally, to (1, 0) and click. To leave this mode of adding line segments, select DONE. This completes the border definition. RETURN to the main page. Subregions: Dividing the Geometry From the main menu, you should now notice more options are available. The next one we will use is Subregions. This allows you to break your object up into a number of simpler regions that are more convenient for meshing. In the plate problem, we will divide the plate and hole into 2 separate regions for meshing. Select Subregions and you will see a number of options that are similar to those available on the geometry page.

Select the Get Line option, click on (5, 5), and then click on the midpoint (as accurate as you can). Select DONE (not QUIT) to accept this line.

This is all the division that is necessary, you should now RETURN to the main menu. Note that when we added the new lines, we actually split the existing lines defining the geometry. Subdivide: Specifying Nodal Points on the Edges From the main menu, select Subdivide. In the subdivision page, one specifies nodal densities along the boundaries for all the regions in the structure. The arrow on each edge indicates its orientation, and is used for a varying density along the edges.

We will start with the two arcs that now define the hole. We will define 5 subdivisions on each quarter circle arc. To do this, select Num Segments and enter 5. Now select the Subdivide option and click on both arcs defining the circle, and on the top and left edges of the plate. You should see triangles to indicate the nodal densities. Select QUIT to leave the selection mode.

Now select Num Segments and enter 6. Define this nodal density for the two three radial segments on the right, diagonal, and bottom segments (remember that you must select Subdivide to enter the selection mode and Quit to return). Select Num Segments and enter 6. We also want a finer mesh near the hole, so select Ratio and enter 1 and 3 to define a 1:3 ratio. Now select Subdivide and click on the three radial segments. If the refinement is in the wrong direction for a segment, select Revert Ratio and click on that line segment.

As illustrated, the Ratio option can be used to specify a mesh with a density that varies along a line. For instance, selecting Ratio and entering 1 and 3 means the mesh size will vary a factor of three in the direction of the arrow defining the line segment. The Revert Ratio option can be used to change the arrow direction. Mesh: Mesh Generation for Plate Return to the main page. The next step is to generate meshes for the four regions. Select the Mesh option to move to the mesh page. The first two options on this page allow you to select element types. The defaults are Q8 quadrilateral elements, and T6 triangular elements. Select Q8. All two regions of the patched plate can be meshed with the bilinear four sided meshing algorithm (Bilinear 4 Side). This algorithm requires a topologically rectangular region with equal numbers of nodes on opposing sides.

The mesh is shown in the figure below.

(Note that it is not necessary to have regions that can be meshed using the Bilinear 4side option. For example, the Transition mesh option allows you to mesh arbitrary regions with arbitrary subdivision on the sides. This option can generate interior nodes or use only the boundary nodes.)

Meshing of the plate is now complete, you should RETURN to the main page. Create a CASCA restart file by means of the Write option. After you select Write, you must bring the program control window to the front and type in a file name (no extensions). Give a name such as ”hole”, and a hole.csc file (binary) will be written. This is used for reentering CASCA later. A *.inp file (in ASCII format) is created as data input file for your FEM program by selecting the Write Mesh option. Again specify the name ”hole”, and a hole.inp file will be created.

To obtain a snap shot of the picture displayed on screen, use SNAP command, switch to the DOS command window and type in ”y”, a postscript file called `gra#.ps` will be created in the working directory. The # is the picture number being created in this CASCA session. Use Ghostview or other software to view the picture.

Select END and CONFIRM EXIT to leave CASCA. The input file format and the input data file just created are attached for your information. This input file will be used as the input for your FEM program.

C.3 Input File Format

The `*.inp` files contain the mesh data used for input to finite element programs. These are human readable ASCII files that describe a mesh using nodes and elements in a format similar to those used by most other finite element programs. The `*.inp` file is written by CASCA using the `Write Mesh` option. The file format is given below.

```
**************************************************************************
************************  INPUT FILE FORMAT  ************************
```

```
**************************************************************************

==========================================================================

                  Card Set 1:  Title card
                  Number of cards in set: 1

   Problem_title(Char*40) - Title of problem, 40 chars

==========================================================================

                  Card Set 2:  Control card
                  Number of cards in set: 1

   Num_Nodes (I*4)
   Num_Elem (I*4)
   Num_Mat (I*4)    - Number of materials
   Prob_Type (I*4)  - Analysis type
                               0 Axisymmetric
                               1 Plane Stress
                               2 Plane Strain
                               3 Linear Bending

==========================================================================

                  Card Set 3:  Material Properties
                  Number of cards in set: Num_Mat (See Card Set 2)

   FORMAT(I5, 14F10.2)

   Mat_Type(I*4)   - The material type
                       1 Linear elastic isotropic
                       2 Linear elastic orthotropic

   If Mat_Type = 1
       Young's modulus (R*8)
       Poisson's Ratio (R*8)
       Thickness (R*8)
       Fracture Toughness KIc (R*8)
       Density (R*8)

   If Mat_Type = 2
       Young's modulus in the 1 direction (R*8)
       Young's modulus in the 2 direction (R*8)
       Young's modulus in the 3 direction (R*8)
       Modulus of rigidity in the 12 direction (R*8)
       Poisson's ratio in the 12 direction (R*8)
       Poisson's ratio in the 13 direction (R*8)
       Poisson's ratio in the 23 direction (R*8)
       Rotation angle beta (R*8)
       Thickness (R*8)
```

```
        Fracture Toughness KIc in the 1 direction (R*8)
        Fracture Toughness KIc in the 1 direction (R*8)
        Density (R*8)

=====================================================================

                    Card Set 4:  Connectivity
                    Number of cards in set: Num_Elem
    FORMAT(10I5)

    Elem_Num(I*4)         - Element number
    Material(I*4)         - Material number for element
    Elem_Nodes(1)(I*4)    - First node number
          .
          .
    Elem_Nodes(8)(I*4)    - Eighth node number

    Note:  Node numbers should be specified in a counter clockwise
    direction, starting at any corner node.  If Elem_Nodes has eight
    non-zero elements a Q8 is assumed, if 6 non-zero elements a T6 is
    assumed.

=====================================================================

                    Card Set 5:  Nodal Coordinates
                    Number of cards in set: Num_Nodes

    Node_Number(I*4)    - Node number
    X_Coord(R*4)        - X coordinate of node
    Y_Coord(R*4)        - Y coordinate of node

=====================================================================
```

C.4 Plot File Format

The `*.con` files contain the result data used for input to `conPlot.exe` program for contour plots. It contains ASCII data in a prescribed format. The file format is given below (note: `I` = integer format, `R` = real format, `S` = strings. One space should be inserted in between data on the same line.).

```
=====================================================================
Line 1:
%1%

Line 2:
I - Total number of components for contour plots
I - Total number of nodes
I - Total number of elements
```

```
Line 3:
S - Label for component 1

Line 4:
S - Label for component 2

Line 5:
S - Label for component 3

Line 6:
S - Label for component 4

Line 7:
S - Label for component 5

Line 8:
R - X coordinate of the first node
R - Y coordinate of the first node
R - UX displacement of the first node
R - UY coordinate of the first node

...... [repeat for all nodes]

Line 9:
I - 8 [for Q8 element]
I - Node number of lower left corner node for element 1 (local node 1)
I - Node number of lower right corner node for element 1 (local node 2)
I - Node number of upper right corner node for element 1 (local node 3)
I - Node number of upper left corner node for element 1 (local node 4)
I - Node number of bottom mid-edge node for element 1 (local node 5)
I - Node number of right mid-edge node for element 1 (local node 6)
I - Node number of top mid-edge node for element 1 (local node 7)
I - Node number of left mid-edge node for element 1 (local node 8)

Line 10:
R - Component 1 (sigma_xx) at local node 1 (of element 1)
R - Component 1 (sigma_yy) at local node 1 (of element 1)
R - Component 1 (sigma_zz) at local node 1 (of element 1)
R - Component 1 (sigma_xy) at local node 1 (of element 1)
R - Component 1 (sigma_1) at local node 1 (of element 1)

Line 11:
R - Component 1 (sigma_xx) at local node 2 (of element 1)
R - Component 1 (sigma_yy) at local node 2 (of element 1)
R - Component 1 (sigma_zz) at local node 2 (of element 1)
R - Component 1 (sigma_xy) at local node 2 (of element 1)
R - Component 1 (sigma_1) at local node 2 (of element 1)

Line 12:
R - Component 1 (sigma_xx) at local node 3 (of element 1)
```

```
R - Component 1 (sigma_yy) at local node 3 (of element 1)
R - Component 1 (sigma_zz) at local node 3 (of element 1)
R - Component 1 (sigma_xy) at local node 3 (of element 1)
R - Component 1 (sigma_1) at local node 3 (of element 1)

Line 13:
R - Component 1 (sigma_xx) at local node 4 (of element 1)
R - Component 1 (sigma_yy) at local node 4 (of element 1)
R - Component 1 (sigma_zz) at local node 4 (of element 1)
R - Component 1 (sigma_xy) at local node 4 (of element 1)
R - Component 1 (sigma_1) at local node 4 (of element 1)

Line 14:
R - Component 1 (sigma_xx) at local node 5 (of element 1)
R - Component 1 (sigma_yy) at local node 5 (of element 1)
R - Component 1 (sigma_zz) at local node 5 (of element 1)
R - Component 1 (sigma_xy) at local node 5 (of element 1)
R - Component 1 (sigma_1) at local node 5 (of element 1)

Line 15:
R - Component 1 (sigma_xx) at local node 6 (of element 1)
R - Component 1 (sigma_yy) at local node 6 (of element 1)
R - Component 1 (sigma_zz) at local node 6 (of element 1)
R - Component 1 (sigma_xy) at local node 6 (of element 1)
R - Component 1 (sigma_1) at local node 6 (of element 1)

Line 16:
R - Component 1 (sigma_xx) at local node 7 (of element 1)
R - Component 1 (sigma_yy) at local node 7 (of element 1)
R - Component 1 (sigma_zz) at local node 7 (of element 1)
R - Component 1 (sigma_xy) at local node 7 (of element 1)
R - Component 1 (sigma_1) at local node 7 (of element 1)

Line 17:
R - Component 1 (sigma_xx) at local node 8 (of element 1)
R - Component 1 (sigma_yy) at local node 8 (of element 1)
R - Component 1 (sigma_zz) at local node 8 (of element 1)
R - Component 1 (sigma_xy) at local node 8 (of element 1)
R - Component 1 (sigma_1) at local node 8 (of element 1)

...... [repeat for all elements]


======================================================================
```

The following is an example of how the *.con file looks like:

```
========
[data] [explanations]
%1%
5 8 1  [number of components, number of nodes, number of elements]
```

```
SIGXX  [label for comp1]
SIGYY  [label for comp2]
SIGZZ  [label for comp3]
SIGXY  [label for comp4]
SIG1 [label for comp5]
0.0000e+000   5.0000e-001   0.0000e+000  -4.3103e-005  [X Y UX UY for 1st node]
......(repeat for all nodes)
8 8 7 6 5 4 3 2 1
[8 node1 node2 node3 node4 node5 node6 node7 node8 for 1st element]
1.0000e+001  -8.7251e-016   0.0000e+000   3.7821e-015   1.0000e+001
[comp1 comp2 comp3 comp4 comp5 for node1]
1.0000e+001   1.6119e-015   0.0000e+000   2.2768e-015   1.0000e+001
[comp1 comp2 comp3 comp4 comp5 for node2]
......  [comp1 comp2 comp3 comp4 comp5 for node3]
......  [comp1 comp2 comp3 comp4 comp5 for node4]
......
......(repeat for all elements)
================
```

For our project, number of components=5, and the five components are: comp1 = , comp2 = , comp3 = , comp4 = , comp5 = (first principal stress). X, Y are the x and y coordinates of a node, and UX and UY are displacements. In the list: "8 node1 node2 node3 node4 node5 node6 node7 node8", node1 means local node 1 of that element. The nodal stresses (comp1 comp2 ...) should be the stresses at the node averaged over all neighbor elements. Since the shape functions we used count the nodes in an element from the lower left corner counter-clockwisely and consecutively (0, 1, 2, 3, 4, 5, 6, 7), while the conPlot counts the corner nodes first, and then mid-edge nodes (0, 2, 4, 6, 1, 3, 5, 7), you should map the node indexing by the following:

```
map[0] = 0;
map[1] = 2;
map[2] = 4;
map[3] = 6;
map[4] = 1;
map[5] = 3;
map[6] = 5;
map[7] = 7;
```