# Chapter 1

# INTRODUCTION

## 1.1 Finite Element Methods

The goal of this text is to introduce finite element methods from a rather broad perspective. We will consider the basic theory of finite element methods as utilized as an engineering tool. Likewise, example engineering applications will be presented to illustrate practical concepts of heat transfer, stress analysis, and other fields. Today the subject of error analysis for adaptivity of finite element methods has reached the point that it is both economical and reliable and should be considered in an engineering analysis. Finally, we will consider in some detail the typical computational procedures required to apply modern finite element analysis, and the associated error analysis. In this chapter we will begin with an overview of the finite element method. We close it with consideration of modern programming approaches and a discussion of how the software provided differs from the author's previous implementations of finite element computational procedures.

In modern engineering analysis it is rare to find a project that does not require some type of finite element analysis (FEA). The practical advantages of FEA in stress analysis and structural dynamics have made it the accepted tool for the last two decades. It is also heavily employed in thermal analysis, especially in connection with thermal stress analysis.

Clearly, the greatest advantage of FEA is its ability to handle truly arbitrary geometry. Probably its next most important features are the ability to deal with general boundary conditions and to include nonhomogeneous and anisotropic materials. These features alone mean that we can treat systems of arbitrary shape that are made up of numerous different material regions. Each material could have constant properties or the properties could vary with spatial location. To these very desirable features we can add a large amount of freedom in prescribing the loading conditions and in the postprocessing of items such as the stresses and strains. For elliptical boundary value problems the FEA procedures offer significant computational and storage efficiencies that further enhance its use. That class of problems include stress analysis, heat conduction, electrical fields, magnetic fields, ideal fluid flow, etc. FEA also gives us an important solution technique for other problem classes such as the nonlinear Navier-Stokes equations for fluid dynamics, and for plasticity in nonlinear solids.

Here we will show what FEA has to offer and illustrate some of its theoretical formulations and practical applications. A design engineer should study finite element methods in more detail than we can consider here. It is still an active area of research. The current trends are toward the use of error estimators and automatic adaptive FEA procedures that give the maximum accuracy for the minimum computational cost. This is also closely tied to shape modification and optimization procedures.

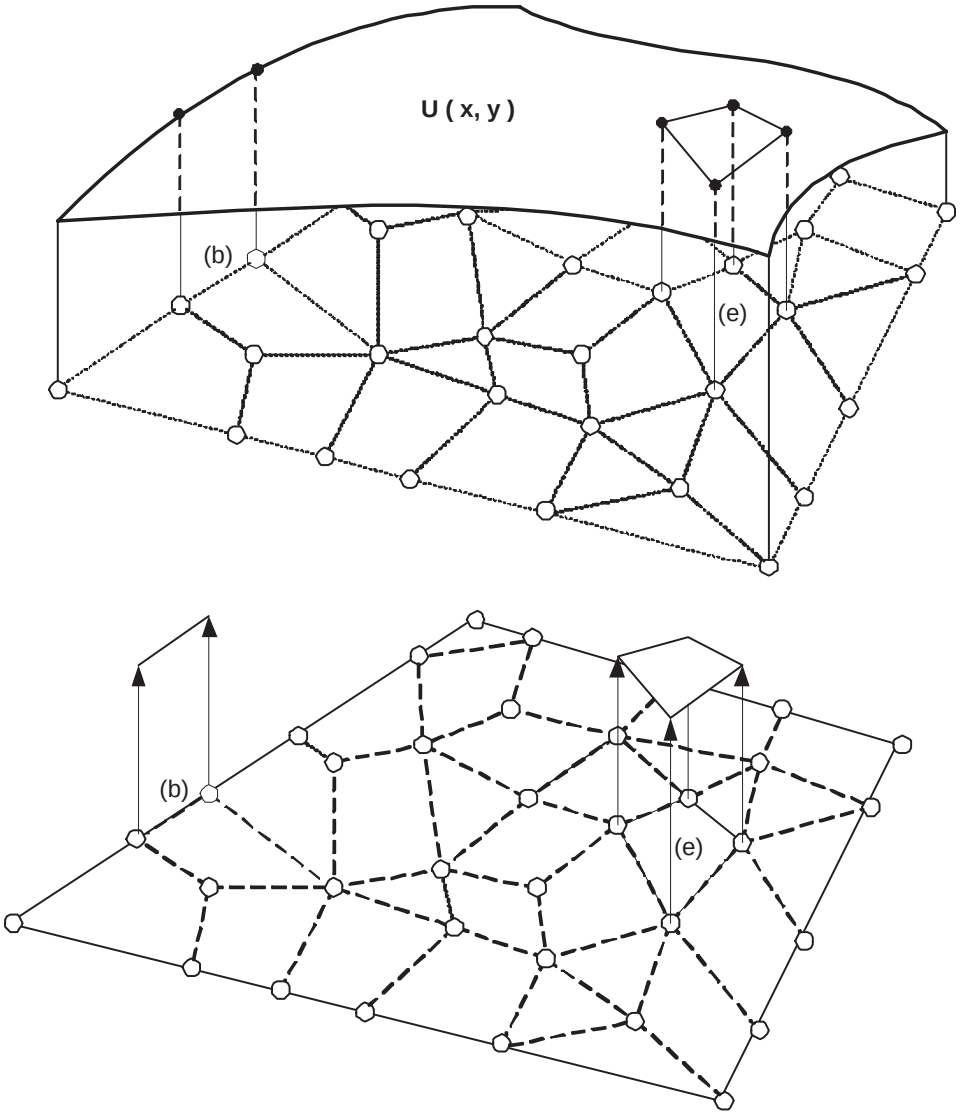



Figure 1.2.1 Piecewise approximation of a scalar function

## 1.2 Capabilities of FEA

There are many commercial and public-domain finite element systems that are available today. To summarize the typical capabilities, several of the most widely used software systems have been compared to identify what they have in common. Often we find about 90% of the options are available in all the systems. Some offer very specialized capabilities such as aeroelastic flutter or hydroelastic lubrication. The mainstream capabilities to be listed here are found to be included in the majority of the commercial systems. The newer adaptive systems may have fewer options installed but they are rapidly adding features common to those given above. Most of these systems are available on engineering workstations and personal computers as well as mainframes and supercomputers. The extent of the usefulness of a FEA system is directly related to the extent of its element library. The typical elements found within a single system usually include membrane, solid, and axisymmetric elements that offer linear, quadratic, and cubic approximations with a fixed number of unknowns per node. The new hierarchical elements have relatively few basic shapes but they do offer a potentially large number of unknowns per node (up to 81 for a solid). Thus, the actual effective element library size is extremely large.

In the finite element method, the boundary and interior of the region are subdivided by lines (or surfaces) into a finite number of discrete sized subregions or finite elements. A number of nodal points are established with the mesh. The size of an element is usually associated with a reference length denoted by $h$. It, for example, may be the diameter of the smallest sphere that can enclose the element. These nodal points can lie anywhere along, or inside, the subdividing mesh, but they are usually located at intersecting mesh lines (or surfaces). The elements may have straight boundaries and thus, some geometric approximations will be introduced in the geometric idealization if the actual region of interest has curvilinear boundaries. These concepts are graphically represented in Fig. 1.2.1.

The nodal points and elements are assigned identifying integer numbers beginning with unity and ranging to some maximum value. The assignment of the nodal numbers and element numbers will have a significant effect on the solution time and storage requirements. The analyst assigns a number of generalized degrees of freedom to each and every node. These are the unknown nodal parameters that have been chosen by the analyst to govern the formulation of the problem of interest. Common nodal parameters are displacement components, temperatures, and velocity components. The nodal parameters do not have to have a physical meaning, although they usually do. For example, the hierarchical elements typically use the derivatives up to order six as the midside nodal parameters. This idealization procedure defines the total number of degrees of freedom associated with a typical node, a typical element, and the total system. Data must be supplied to define the spatial coordinates of each nodal point. It is common to associate some code to each node to indicate which, if any, of the parameters at the node have boundary constraints specified. In the new adaptive systems the number of nodes, elements, and parameters per node usually all change with each new iteration.

Another important concept is that of *element connectivity*, (or topology) i.e., the list of global node numbers that are attached to an element. The element connectivity data defines the topology of the (initial) mesh, which is used, in turn, to assemble the system

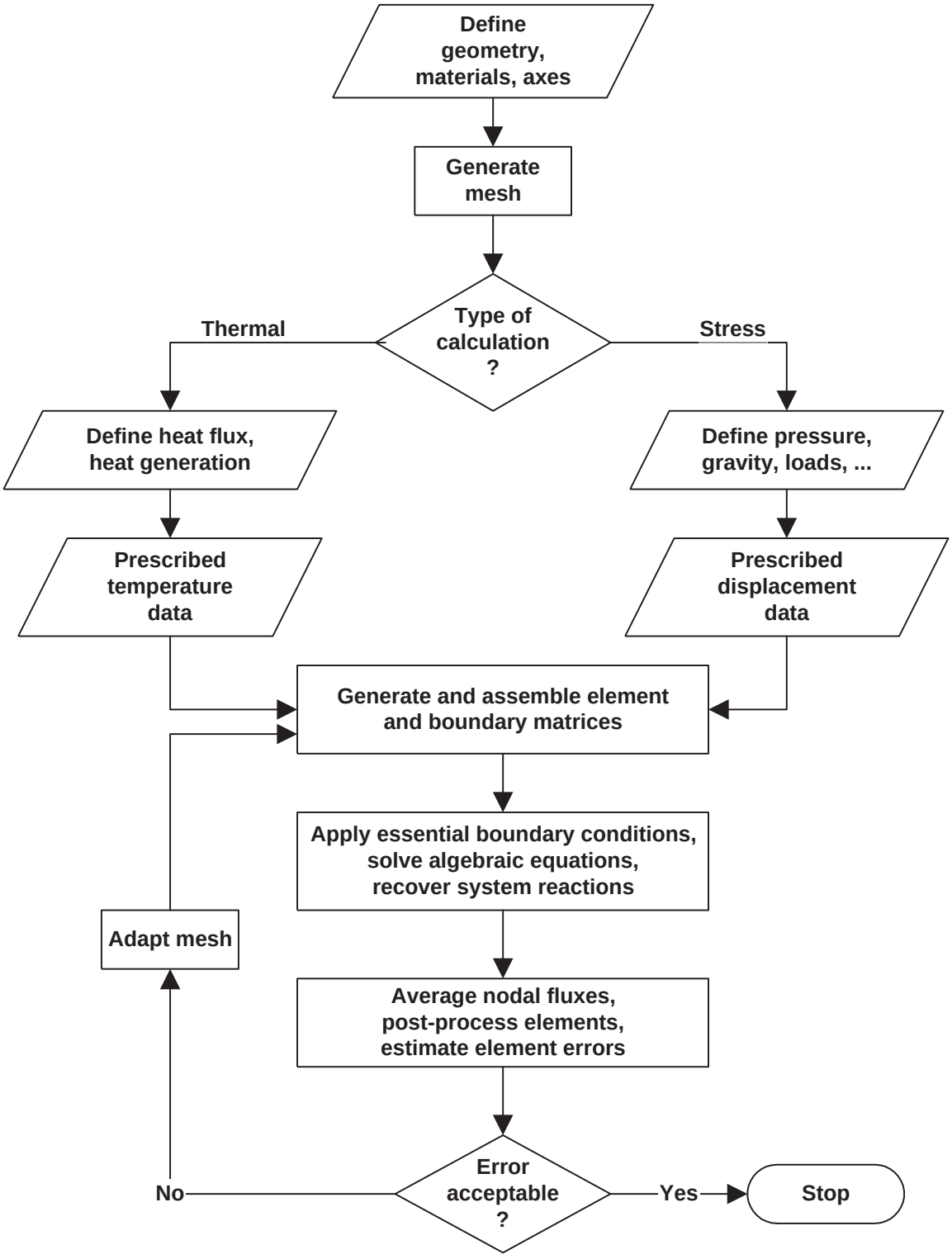



Figure 1.2.2 Typical stages in a finite element analysis

algebraic equations. Thus, for each element it is necessary to input, in some consistent order, the node numbers that are associated with that particular element. The list of node numbers connected to a particular element is usually referred to as the element incident list for that element. We usually associate a material code, or properties, with each element.

Finite element analysis can require very large amounts of input data. Thus, most FEA systems offer the user significant data generation or supplementation capabilities. The common data generation and validation options include the generation and/or replication of coordinate systems, node locations, element connectivity, loading sets, restraint conditions, etc. The verification of such extensive amounts of input and generated data is greatly enhanced by the use of computer graphics.

In the adaptive methods we must also compute the error indicators, error estimators, and various energy norms. All these quantities are usually output at 1 to 27 points in each of thousands of elements. The most commonly needed information in an engineering analysis is the state of temperatures, or displacements and stresses. Thus, almost every system offers linear static stress analysis capabilities, and linear thermal analysis capabilities for conduction and convection that are often needed to provide temperature distributions for thermal stress analysis. Usually the same mesh geometry is used for the temperature analysis and the thermal stress analysis. Of course, some studies require information on the natural frequencies of vibration or the response to dynamic forces or the effect of frequency driven excitations. Thus, dynamic analysis options are usually available. The efficient utilization of materials often requires us to employ nonlinear material properties and/or nonlinear equations. Such resources require a more experienced and sophisticated user. The usual nonlinear stress analysis features in large commercial FEA systems include buckling, creep, large deflections, and plasticity. Those advanced methods will not be considered here.

There are certain features of finite element systems which are so important from a practical point of view that, essentially, we cannot get along without them. Basically we

**Table 1.1 Typical unknown variables in finite element analysis**

| Application | Primary | Associated | Secondary |
|---|---|---|---|
| Stress analysis | Displacement,<br>Rotation | Force,<br>Moment | Stress,<br>Failure criterion<br>Error estimates |
| Heat transfer | Temperature | Flux | Interior flux<br>Error estimates |
| Potential flow | Potential function | Normal velocity | Interior velocity<br>Error estimates |
| Navier-Stokes | Velocity | Pressure | Error estimates |

**Table 1.2 Typical given variables and corresponding reactions**

| Application | Given | Reaction |
|---|---|---|
| Stress analysis | Displacement | Force |
| | Rotation | Moment |
| | Force | Displacement |
| | Couple | Rotation |
| Heat transfer | Temperature | Heat flux |
| | Heat flux | Temperature |
| Potential flow | Potential | Normal velocity |
| | Normal velocity | Potential |
| Navier-Stokes | Velocity | Force |

have the ability to handle completely arbitrary geometries, which is essential to practical engineering analysis. Almost all the structural analysis, whether static, dynamic, linear or nonlinear, is done by finite element techniques on large problems. The other abilities provide a lot of flexibility in specifying loading and restraints (support capabilities). Typically, we will have several different materials at different arbitrary locations within an object and we automatically have the capability to handle these nonhomogeneous materials. Just as importantly, the boundary conditions that attach one material to another are usually automatic, and we don't have to do anything to describe them unless it is possible for gaps to open between materials. Most important, or practical, engineering components are made up of more than one material, and we need an easy way to handle that. What takes place less often is the fact that we have *anisotropic materials* (one whose properties vary with direction, instead of being the same in all directions). There is a great wealth of materials that have this behavior, although at the undergraduate level, anisotropic materials are rarely mentioned. Many materials, such as reinforced concrete, plywood, any filament-wound material, and composite materials, are essentially anisotropic. Likewise, for heat-transfer problems, we will have thermal conductivities that are directionally dependent and, therefore, we would have to enter two or three thermal conductivities that indicate how this material is directionally dependent. These advantages mean that for practical use finite element analysis is very important to us. The biggest disadvantage of the finite element method is that it has so much power that large amounts of data and computation will be required.

All real objects are three-dimensional but several common special cases have been defined that allow two-dimensional studies to provide useful insight. The most common examples in solid mechanics are the states of *plane stress* (covered in undergraduate mechanics of materials) and *plane strain*, the *axisymmetric solid* model, the *thin-plate* model, and the *thin-shell* model. The latter is defined in terms of two parametric surface coordinates even though the shell exists in three dimensions. The *thin beam* can be thought of as a degenerate case of the thin-plate model. Even though today's solid modelers can generate three-dimensional meshes relative easily one should learn to

approach such problems carefully. A well planned series of two-dimensional approximations can provide important insight into planning a good three-dimensional model. They also provide good "ballpark" checks on the three-dimensional answers. Of course, the use of basic handbook calculations in estimating the answer before approaching a FEA system is also highly recommended.

The typical unknown variables in a finite element analysis are listed in Table 1.1 and a list of related action-reaction variables are cited in Table 1.2. Figure 1.2.2 outlines as a flow chart the major steps needed for either a thermal analysis or stress analysis. Note that these segments are very similar. One of the benefits of developing a finite element approach is that most of the changes related to a new field of application occur at the element level definitions and usually represent less than 5 percent of the total programming effort.

## 1.3 Outline of Finite Element Procedures

From the mathematical point of view the finite element method is an integral formulation. Modern finite element integral formulations are usually obtained by either of two different procedures: *weighted residual* or *variational formulations*. The following sections briefly outline the common procedures for establishing finite element models. It is fortunate that all these techniques use the same bookkeeping operations to generate the final assembly of algebraic equations that must be solved for the unknowns.

The generation of finite element models by the utilization of weighted residual techniques is increasingly important in the solution of differential equations for non-structural applications. The weighted residual method starts with the governing differential equation to be satisfied in a domain $\Omega$ :

$$L(\phi) \;=\; Q\,, \tag{1.1}$$

where $L$ denotes a differential operator acting on the primary unknown, $\phi$, and $Q$ is a source term. Generally we assume an approximate solution, say $\phi^*$, for the spatial distribution of the unknown, say

$$\phi(x) \approx \phi^* = \sum_i^n h_i(x)\,\Phi_i^*\,, \tag{1.2}$$

where the $h_i(x)$ are spatial distributions associated with the coefficient $\Phi_i^*$. That assumption leads to a corresponding assumption for the spatial gradient of the assumed behavior. Next we substitute these assumptions on spatial distributions into the differential equation. Since the assumption is approximate, this operation defines a residual error term, $R$, in the differential equation

$$L(\phi^*) \;-\; Q = R \neq 0\,. \tag{1.3}$$

Although we cannot force the residual term to vanish, it is possible to force a weighted integral, over the solution domain, of the residual to vanish. That is, the integral of the product of the residual term and some weighting function is set equal to zero, so that

$$I_i = \int_\Omega R\, w_i\, d\Omega = 0 \tag{1.4}$$

leads to the same number of equations as there are unknown $\Phi_i^*$ values. Most of the time

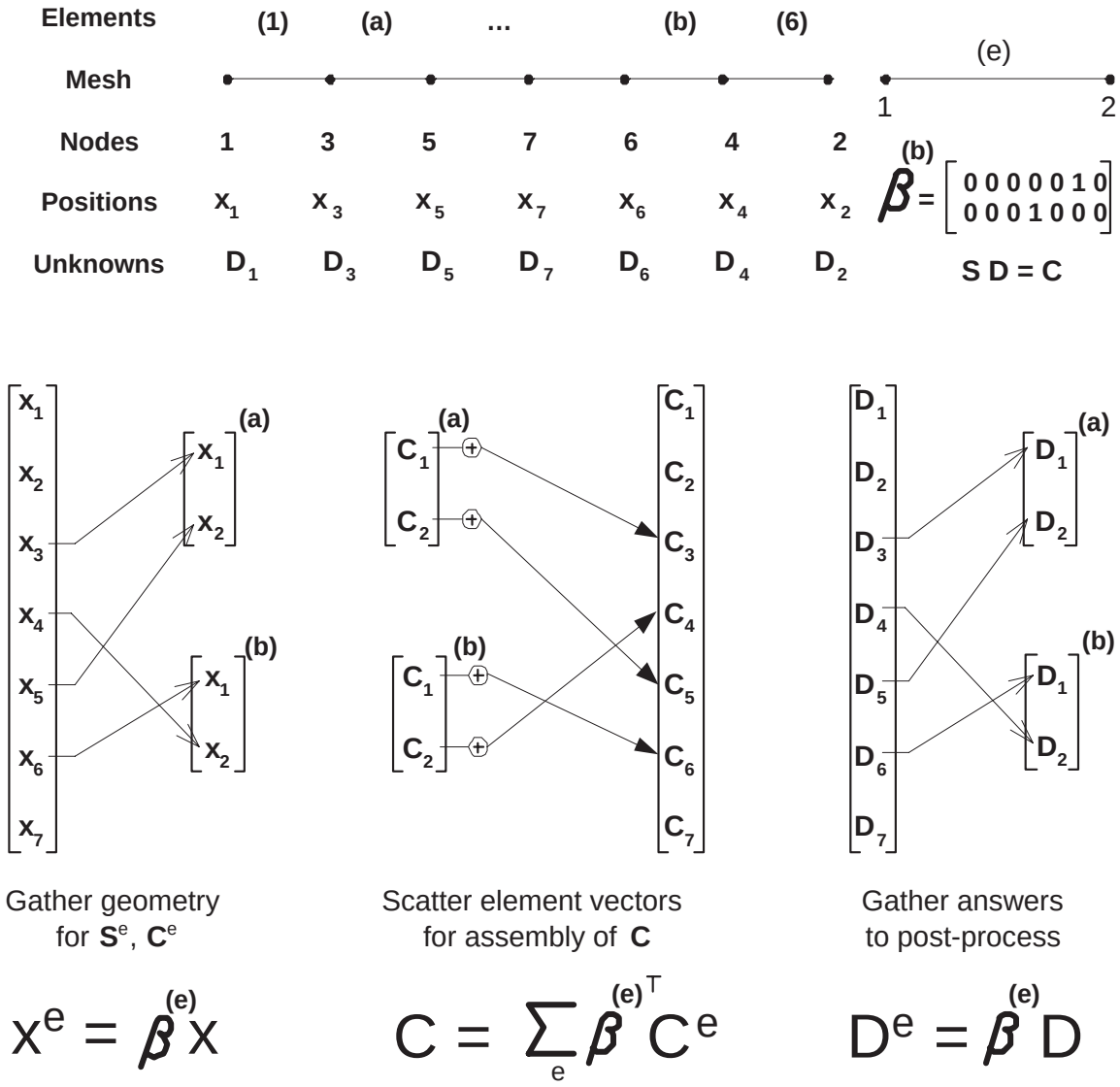



Figure 1.3.1 Gather and scatter concepts for finite elements

we will find it very useful to employ integration by parts on this governing integral.

Substituting an assumed spatial behavior for the approximate solution, $\phi^*$, and the weighting function, $w$, results in a set of algebraic equations that can be solved for the unknown nodal coefficients in the approximate solution. This is because the unknown coefficients can be pulled out of the spatial integrals involved in the assembly process.

The choice of weighting function defines the type of weighted residual technique being utilized. The Galerkin criterion selects

$$w_i = h_i(x)\,, \tag{1.5}$$

to make the residual error "orthogonal" to the approximate solution. Use of integration by parts with the Galerkin procedure (i.e., the Divergence Theorem) reduces the continuity requirements of the approximating functions. If a Euler variational procedure

exists, the Galerkin criterion will lead to the same element matrices.

A spatial interpolation, or blending function is assumed for the purpose of relating the quantity of interest within the element in terms of the values of the nodal parameters at the nodes connected to that particular element. For both weighted residual and variational formulations, the following restrictions are accepted for establishing convergence of the finite element model as the mesh refinement increases:

1. The element interpolation functions must be capable of modeling any constant values of the dependent variable or its derivatives, to the order present in the defining integral statement, in the limit as the element size decreases.
2. The element interpolation functions should be chosen so that at element interfaces the dependent variable and its derivatives, of one order less than those occurring in the defining integral statement, are continuous.

Through the assumption of the spatial interpolations, the variables of interest and their derivatives are uniquely specified throughout the solution domain by the nodal parameters associated with the nodal points of the system. The parameters at a particular node directly influence only the elements connected to that particular node. The domain will be split into a mesh. That will require that we establish some bookkeeping processes to keep up with data going to, or coming from a node or element. Those processes are commonly called gather and scatter, respectively. Figure 1.3.1 shows some of these processes for a simple mesh with one scalar unknown per node, $n_g = 1$, in a one-dimensional physical space. There the system node numbers are shown numbered in an arbitrary fashion. To establish the local element space domain we must usually gather the coordinates of each of its nodes. For example, for element $b(= 5)$ it gathers the data for system node numbers 6 and 4, respectively, so that the element length, $L^{(5)} = x_6 - x_4$, can be computed. Usually we also have to gather some data on the coefficients in the differential equation (material properties usually). If the coefficients vary over space they may be supplied as data at the nodes that must also be gathered to form the element matrices.

After the element behavior has been described by spatial assumptions, then the derivatives of the space functions are used to approximate the spatial derivatives required in the integral form. The remaining fundamental problem is to establish the element matrices, $\mathbf{S}^e$ and $\mathbf{C}^e$. This involves substituting the approximation space functions and their derivatives into the governing integral form and moving the unknown coefficients, $\mathbf{D}^e$, outside the integrals. Historically, the resulting matrices have been called the element stiffness matrix and load vector, respectively.

Once the element equations have been established the contribution of each element is added, using its topology (or connectivity), to form the system equations. The system of algebraic equations resulting from FEA (of a linear system) will be of the form $\mathbf{S}\,\mathbf{D} = \mathbf{C}$. The vector $\mathbf{D}$ contains the unknown nodal parameters, and the matrices $\mathbf{S}$ and $\mathbf{C}$ are obtained by assembling the known element matrices, $\mathbf{S}^e$ and $\mathbf{C}^e$, respectively. Figure 1.3.1 shows how the local coefficients of the element source vector, $\mathbf{C}^e$, are scattered and added into the resultant system source, $\mathbf{C}$. That illustration shows a conversion of local row numbers to the corresponding system row numbers (by using the element connectivity data). An identical conversion is used to convert the local and system

column numbers needed in assemblying each $\mathbf{S}^e$ into **S**. In the majority of problems $\mathbf{S}^e$, and thus, **S**, will be symmetric. Also, the system square matrix, **S**, is usually banded about the diagonal or at least *sparse*. If **S** is unsymmetric its upper and lower triangles have the same sparsity.

After the system equations have been assembled, it is necessary to apply the *essential boundary constraints* before solving for the unknown nodal parameters. The most common types of essential boundary conditions (EBC) are (1) defining explicit values of the parameter at a node and (2) defining constraint equations that are linear combinations of the unknown nodal quantities. The latter constraints are often referred to in the literature as *multi-point constraints* (MPC). An essential boundary condition should not be confused with a forcing condition of the type that involves a flux or traction on the boundary of one or more elements. These element boundary source, or forcing, terms contribute additional terms to the governing integral form and thus to the element square and/or column matrices for the elements on which the sources were applied. Thus, although these (*Neumann-type*, and *Robin* or *mixed-type*) conditions do enter into the system equations, their presence may not be obvious at the system level. Wherever essential boundary conditions do not act on part of the boundary, then at such locations, source terms from a lower order differential equation automatically apply. If one does not supply data for the source terms, then they default to zero. Such portions of the boundary are said to be subject to natural boundary conditions (NBC). The natural boundary condition varies with the integral form, and typical examples will appear later.

The initial sparseness (the relative percentage of zero entries) of the square matrix, **S**, is an important consideration since we only want to store non-zero terms. If we employ a direct solver then many initially zero terms will become non-zero during the solution process and the assigned storage must allow for that. The "fill-in" depends on the numbering of the nodes. If the FEA system being employed does not have an automatic renumbering system to increase sparseness, then the user must learn how to number nodes (or elements) efficiently. After the system algebraic equations have been solved for the unknown nodal parameters, it is usually necessary to output the parameters, **D**. For every essential boundary condition on **D**, there is a corresponding unknown *reaction* term in **C** that can be computed after **D** is known. These usually have physical meanings and should be output to help check the results.

In rare cases the problem would be considered completed at this point, but in most cases it is necessary to use the calculated values of the nodal parameters to calculate other quantities of interest. For example, in stress analysis we use the calculated nodal displacements to solve for the strains and stresses. All adaptive programs must do a very large amount of postprocessing to be sure that the solution, **D**, has been obtained to the level of accuracy specified by the analyst. Figure 1.3.1 also shows that the gather operation is needed again for extracting the local results, $\mathbf{D}^e$, from the total results, **D**, so they can be employed in special element post-processing and/or error estimates.

Usually the postprocessing calculations involve determining the spatial derivatives of the solution throughout the mesh. Those gradients are continuous within each element domain, but are discontinuous across the inter-element boundaries. The true solution usually has continuous derivatives so it is desirable to somehow average the individual element gradient estimates to create continuous gradient estimate values that can be

reported at the nodes. Fortunately, this addition gradient averaging process also provides new information that allows the estimate of the problem error norm to be calculated. That gradient averaging process will be presented in Chapter 2.

In the next chapter we will review the historical approach of the method of weighted residuals and its extension to finite element analysis. The earliest formulations for finite element models were based on variational techniques. This is especially true in the areas of structural mechanics and stress analysis. Modern analysis in these areas has come to rely on FEA almost exclusively. Variational models find the nodal parameters that yield a minimum (or stationary) value of an integral known as a functional. In most cases it is possible to assign a physical meaning to the integral. For example, in solid mechanics the integral represents the *total potential energy*, whereas in a fluid mechanics problem it may correspond to the rate of entropy production. Most physical problems with variational formulations result in quadratic forms that yield algebraic equations for the system which are symmetric and positive definite. The solution that yields a minimum value of the integral functional and satisfies the essential boundary conditions is equivalent to the solution of an associated differential equation. This is known as the Euler theorem.

Compared to the method of weighted residuals, where we start with the differential equation, it may seem strange to start a variational formulation with an integral form and then check to see if it corresponds to the differential equation we want. However, from Euler's work more than two centuries ago we know the variational forms of most even order differential equations that appear in science, engineering, and applied mathematics. This is especially true for elliptical equations. Euler's Theorem of Variational Calculus states that the solution, $u$, that satisfies the essential boundary conditions and renders stationary the functional

$$I = \int_{\Omega} f\left( x, y, z, \phi, \frac{\partial \phi}{\partial x}, \frac{\partial \phi}{\partial y}, \frac{\partial \phi}{\partial z} \right) d\Omega + \int_{\Gamma} \left( q\phi + a\phi^2/2 \right) d\Gamma \tag{1.6}$$

also satisfies the partial differential equation

$$\frac{\partial f}{\partial \phi} - \frac{\partial}{\partial x} \frac{\partial f}{\partial (\partial \phi / \partial x)} - \frac{\partial}{\partial y} \frac{\partial f}{\partial (\partial \phi / \partial y)} - \frac{\partial}{\partial z} \frac{\partial f}{\partial (\partial \phi / \partial z)} = 0 \tag{1.7}$$

in $\Omega$, and satisfies the natural boundary condition that

$$n_x \frac{\partial f}{\partial (\partial \phi / \partial x)} + n_y \frac{\partial f}{\partial (\partial \phi / \partial y)} + n_z \frac{\partial f}{\partial (\partial \phi / \partial z)} + q + a\phi = 0 \tag{1.8}$$

on $\Gamma$ that is not subject to an essential boundary. Here $n_x$, $n_y$, $n_z$ are the components of the normal vector on the boundary, $\Gamma$. Note that this theorem also defines the natural boundary condition, as well as the corresponding differential equation. In Chapter 7 we will examine some common Euler variational forms and their extensions to finite element analysis.

## 1.4 Assembly into the System Equations

### 1.4.1 Introduction

An important but often misunderstood topic is the procedure for *assemblying* the system equations from the element equations and any boundary contributions. Here assemblying is defined as the operation of adding the coefficients of the element equations into the proper locations in the system equations. There are various methods for accomplishing this but most are numerically inefficient. The numerically efficient *direct assembly* technique will be described here in some detail. We begin by reviewing the simple but important relationship between a set of local (nodal point, or element) degree of freedom numbers and the corresponding system degree of freedom numbers.

The assembly process, introduced in part in Fig. 1.3.1, is graphically illustrated in Fig. 1.4.1 for a mesh consisting of six nodes ($n_m = 6$), three elements ($n_e = 3$). It has a four-node quadrilateral and two three-node triangles, with one generalized parameter per node ($n_g = 1$). The top of the figure shows the nodal connectivity of the three elements and a cross-hatching to define the source of the various coefficients that are occurring in the matrices assembled in the lower part of the figure. The assembly of the system **S** and **C** matrices is graphically coded to denote the sources of the contributing terms but not their values. A hatched area indicates a term that was added in from an element that has the same hash code. For example, the load vector term C(6), coming from the only parameter at node 6, is seen to be the sum of contributions from elements 1 and 2, which are hatched with horizontal (-) and vertical (|) lines, respectively. The connectivity table implies the same thing since node 6 is only connected to those two elements. By way of comparison, the term C(1) has a contribution only from element 2. The connectivity table shows only that element is connected to that corner node.

Note that we have to set **S** = **0** to begin the summation. Referring to Fig. 1.4.1 we see that 10 of the coefficients in **S** remain initially zero. So that example is initially about 27 % sparse. (This will changed if a direct solution process is used.) In practical problems the assembled matrix may initially be 90 % sparse, or more. Special equation solving techniques take advantage of this feature to save memory space and to reduce operation counts.

### 1.4.2 Computing the Equation Index

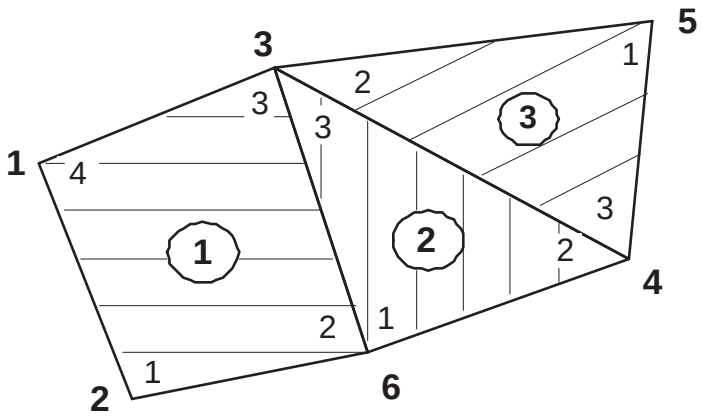
There are a number of ways to assign the equation numbers of the algebraic system that results from a finite element analysis procedure. Here we will select one that has a simple equation that is valid for most applications. Consider a typical nodal point in the system and assume that there are $n_g$ parameters associated with each node. Thus, at a typical node there will be $n_g$ local degree of freedom numbers ($1 \le J \le n_g$) and a corresponding set of system degree of freedom numbers. If $I$ denotes the system node number of the point, then the $n_g$ corresponding system degrees of freedom, $\Phi_k$ have their equation number, $k$ assigned as

$$k(I,J) = n_g * (I-1) + J \qquad 1 \le I \le n_m, \qquad 1 \le J \le n_g, \tag{1.9}$$

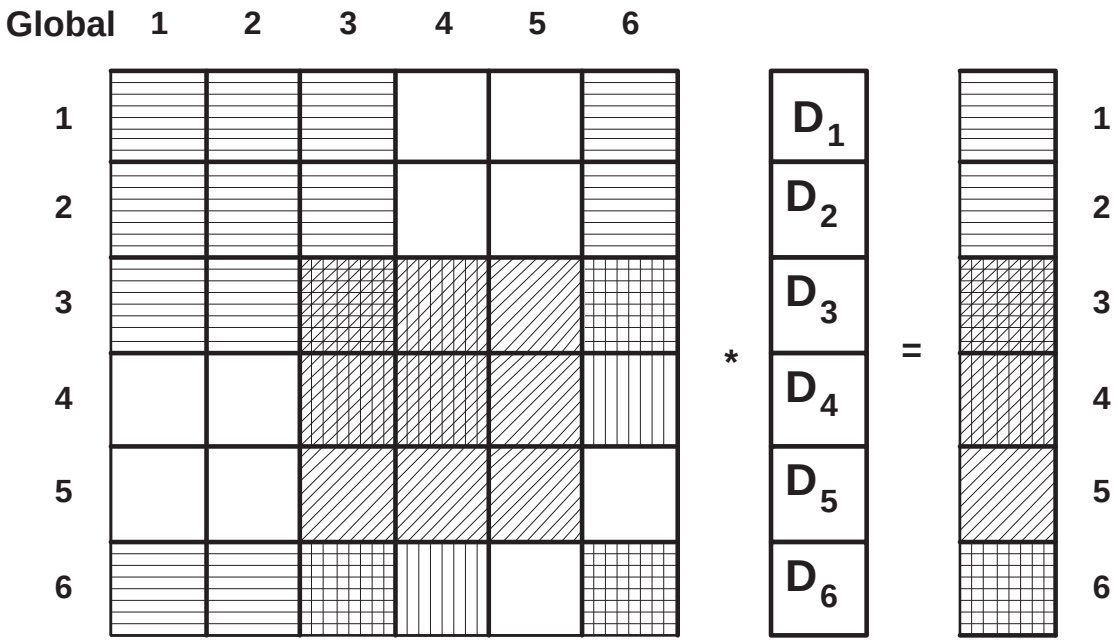
where $n_m$ is the maximum node number in the system. That is, they start at 1 and range to $n_g$ at the first system node then at the second node they range from ($n_g$ + 1) to (2 $n_g$)

| Element | Topology |
|---|---|
| 1 | 2, 6, 3, 1 |
| 2 | 6, 4, 3, 0 |
| 3 | 5, 3, 4, 0 |
| local | 1, 2, 3, 4 |




Global Assembly: S * D = C




$$\beta^{(2)} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Figure 1.4.1 Graphical illustration of matrix assembly

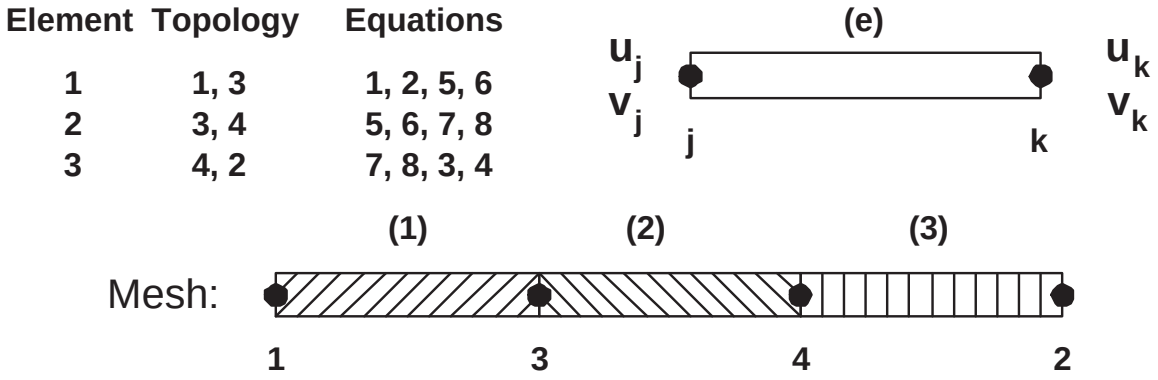
 

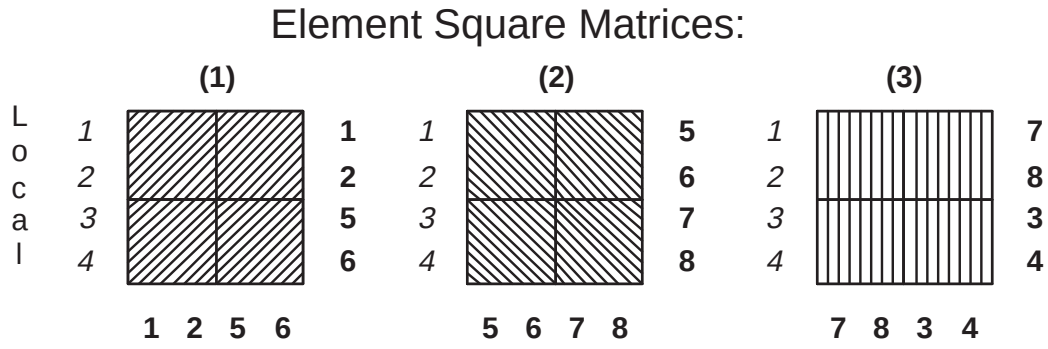
```
FUNCTION GET_INDEX_AT_PT (I_PT) RESULT (INDEX)                ! 1
! * * * * * * * * * * * * * * * * * * * * * * * * * * * *    ! 2
!  DETERMINE DEGREES OF FREEDOM NUMBERS AT A NODE             ! 3
! * * * * * * * * * * * * * * * * * * * * * * * * * * * *    ! 4
Use System_Constants ! for N_G_DOF                            ! 5
 IMPLICIT NONE                                                ! 6
 INTEGER, INTENT(IN)  :: I_PT                                 ! 7
 INTEGER              :: INDEX (N_G_DOF)                      ! 8
 INTEGER :: J ! implied loop                                  ! 9
                                                              !10
! N_G_DOF = NUMBER OF PARAMETERS (DOF) PER NODE               !11
! I_PT    = SYSTEM NODE NUMBER                                !12
! INDEX   = SYSTEM DOF NOS OF NODAL DOF                       !13
!           INDEX (J) = N_G_DOF*(I_PT - 1) + J                !14
                                                              !15
  INDEX = (/ (N_G_DOF*(I_PT - 1) + J, J = 1, N_G_DOF) /)      !16
END FUNCTION GET_INDEX_AT_PT                                  !17

FUNCTION GET_ELEM_INDEX (LT_N, ELEM_NODES) RESULT(INDEX)      ! 1
! * * * * * * * * * * * * * * * * * * * * * * * * * * * *    ! 2
!   DETERMINE DEGREES OF FREEDOM NUMBERS OF ELEMENT           ! 3
! * * * * * * * * * * * * * * * * * * * * * * * * * * * *    ! 4
Use System_Constants    ! for N_G_DOF                         ! 5
 IMPLICIT NONE                                                ! 6
 INTEGER, INTENT(IN) :: LT_N, ELEM_NODES (LT_N)               ! 7
 INTEGER             :: INDEX (LT_N * N_G_DOF)   ! OUT        ! 8
 INTEGER :: EQ_ELEM, EQ_SYS, IG, K, SYS_K        ! LOOPS      ! 9
                                                              !10
! ELEM_NODES = NODAL INCIDENCES OF THE ELEMENT                !11
! EQ_ELEM    = LOCAL  EQUATION NUMBER                         !12
! EQ_SYS     = SYSTEM EQUATION NUMBER                         !13
! INDEX      = SYSTEM DOF NUMBERS OF ELEMENT DOF NUMBERS      !14
! INDEX (N_G_DOF*(K-1)+IG) = N_G_DOF*(ELEM_NODES(K)-1) + IG   !15
! LT_N       = NUMBER OF NODES PER ELEMENT                    !16
! N_G_DOF    = NUMBER OF GENERAL PARAMETERS (DOF) PER NODE    !17
                                                              !18
  DO K = 1, LT_N               ! LOOP OVER NODES OF ELEMENT   !19
    SYS_K = ELEM_NODES (K)     ! SYSTEM NODE NUMBER           !20
    DO IG = 1, N_G_DOF         ! LOOP OVER GENERALIZED DOF    !21
      EQ_ELEM = IG + N_G_DOF * (K     - 1)      ! LOCAL  EQ   !22
      EQ_SYS  = IG + N_G_DOF * (SYS_K - 1)      ! SYSTEM EQ   !23
      IF ( SYS_K > 0 ) THEN                     ! VALID NODE  !24
        INDEX (EQ_ELEM) = EQ_SYS                              !25
      ELSE                                      ! ALLOW MISSING NODE !26
        INDEX (EQ_ELEM) = 0                                   !27
      END IF ! MISSING NODE                                   !28
    END DO ! OVER DOF                                         !29
  END DO ! OVER LOCAL NODES                                   !30
END FUNCTION GET_ELEM_INDEX                                   !31
```

Figure 1.4.2 Computing equation numbers for homogeneous nodal dof



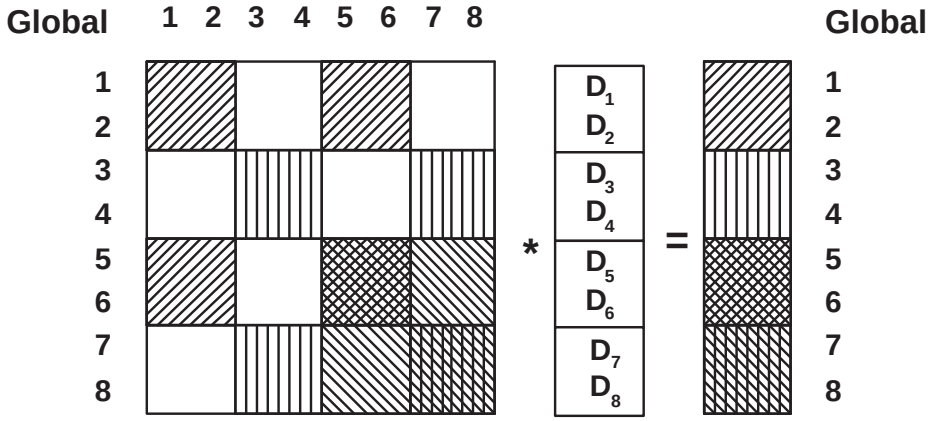




Figure 1.4.3 Assemblying two unknowns per node

```
SUBROUTINE STORE_COLUMN (N_D_FRE, N_EL_FRE, INDEX, C, CC)      ! 1
! * * * * * * * * * * * * * * * * * * * * * * * * * * * * *    ! 2
! STORE ELEMENT COLUMN MATRIX IN SYSTEM COLUMN MATRIX           ! 3
! * * * * * * * * * * * * * * * * * * * * * * * * * * * * *    ! 4
Use Precision_Module ! Defines DP for double precision         ! 5
 IMPLICIT NONE                                                 ! 6
 INTEGER,  INTENT(IN)    :: N_D_FRE, N_EL_FRE                  ! 7
 INTEGER,  INTENT(IN)    :: INDEX (N_EL_FRE)                   ! 8
 REAL(DP), INTENT(IN)    :: C     (N_EL_FRE)                   ! 9
 REAL(DP), INTENT(INOUT) :: CC    (N_D_FRE)                    !10
 INTEGER :: I, J                                               !11
                                                               !12
! N_D_FRE  = NO DEGREES OF FREEDOM IN THE SYSTEM               !13
! N_EL_FRE = NUMBER OF DEGREES OF FREEDOM PER ELEMENT          !14
! INDEX    = SYSTEM DOF NOS OF THE ELEMENT DOF                 !15
! C        = ELEMENT COLUMN MATRIX                             !16
! CC       = SYSTEM COLUMN MATRIX                              !17
                                                               !18
  DO I = 1, N_EL_FRE                      ! ELEMENT ROW        !19
    J = INDEX (I)                         ! SYSTEM ROW NUMBER  !20
    IF ( J > 0 ) CC (J) = CC (J) + C (I) ! SKIP INACTIVE ROW   !21
  END DO ! OVER ROWS                                           !22
END SUBROUTINE STORE_COLUMN                                    !23

SUBROUTINE STORE_FULL_SQUARE (N_D_FRE, N_EL_FRE, S, SS, INDEX) ! 1
! * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  ! 2
!     STORE ELEMENT SQ MATRIX IN FULL SYSTEM SQ MATRIX         ! 3
! * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  ! 4
Use Precision_Module ! Defines DP for double precision         ! 5
 IMPLICIT NONE                                                 ! 6
 INTEGER,  INTENT(IN)    :: N_D_FRE, N_EL_FRE                  ! 7
 INTEGER,  INTENT(IN)    :: INDEX (N_EL_FRE)                   ! 8
 REAL(DP), INTENT(IN)    :: S     (N_EL_FRE, N_EL_FRE)         ! 9
 REAL(DP), INTENT(INOUT) :: SS    (N_D_FRE,  N_D_FRE)          !10
 INTEGER   :: I, II, J, JJ                                     !11
                                                               !12
! N_D_FRE  = TOTAL NO OF SYSTEM DEGREES OF FREEDOM             !13
! N_EL_FRE = NO DEGREES OF FREEDOM PER ELEMENT                 !14
! INDEX    = SYSTEM DOF NOS OF ELEMENT PARAMETERS              !15
! S        = FULL ELEMENT SQUARE MATRIX                        !16
! SS       = FULL SYSTEM SQUARE MATRIX                         !17
                                                               !18
  DO I = 1, N_EL_FRE                      ! ELEMENT ROW        !19
    II = INDEX (I)                        ! SYSTEM ROW NUMBER  !20
    IF ( II > 0 ) THEN                    ! SKIP INACTIVE ROW  !21
      DO J = 1, N_EL_FRE                  ! ELEMENT COLUMN     !22
        JJ = INDEX (J)                    ! SYSTEM COLUMN      !23
        IF ( JJ > 0 ) SS (II, JJ) = SS (II, JJ) + S (I, J)    !24
      END DO ! OVER COLUMNS                                    !25
    END IF                                                     !26
  END DO ! OVER ROWS                                           !27
END SUBROUTINE STORE_FULL_SQUARE                               !28
```

Figure 1.4.4 Assembly of element arrays into system arrays

and so on through the mesh. These elementary calculations are carried out by subroutine *GET_INDEX_AT_PT*. The program assigns $n_g$ storage locations for the vector, say *INDEX*, containing the system degree of freedom numbers associated with the specified nodal point — see Tables 1.4.1 and 1.4.2 for the related details.

A similar expression defines the local equation numbers in an element or on a boundary segment. The difference is that then $I$ corresponds to a local node number and has an upper limit of $n_n$ or $n_b$, respectively. In the latter two cases the local equation number is the subscript for the *INDEX* array and the corresponding system equation number is the integer value stored in *INDEX* at that position. In other words, Eq. 1.9 is used to find local or system equation numbers and $J$ always refers to the specific dof of interest and $I$ corresponds to the type of node number ($I_S$ for a system node, $I_E$ for a local element node, or $I_B$ for a local boundary segment node). For a typical element type subroutine GET_ELEM_INDEX, Fig. 1.4.2, fills the above element *INDEX* vector for any standard or boundary element. In that subroutine storage locations are likewise established for the $n_n$ element incidences (extracted by subroutine GET_ELEM_NODES) and the corresponding $n_i = n_n \times n_g$ system degree of freedom numbers associated with the element, in vector *INDEX*.

Figure 1.4.3 illustrates the use of Eq. 1.9 for calculating the system equation numbers for $n_g = 2$ and $n_m = 4$. The **D** vector in the bottom portion shows that at each node we count its dof before moving to the next node. In the middle section the cross-hatched element matrices show the 4 local equation numbers to the left of the square matrix, and the corresponding system equation numbers are shown to the right of the square matrix, in bold font. Noting that there are $n_g = 2$ dof per node explains why the top left topology list (element connectivity with $n_n = 2$) is expanded to the system equation number list with 4 columns.

Once the system degree of freedom numbers for the element have been stored in a vector, say *INDEX*, then the subscripts of a coefficient in the element equation can be directly converted to the subscripts of the corresponding system coefficient to which it is to be added. This correspondence between local and system subscripts is illustrated in Fig. 1.4.5. The expressions for these assembly, or scatter, operations are generally of the form

$$C_I = C_I + C_i^e , \qquad S_{I,J} = S_{I,J} + S_{i,j}^e \tag{1.10}$$

where $i$ and $j$ are the local subscripts of a coefficient in the element square matrix, $\mathbf{S}^e$, and $I, J$ are the corresponding subscripts of the system equation coefficient, in **S**, to which the element contribution is to be added. The direct conversions are given by $I = INDEX(i)$, $J = INDEX(j)$, where the *INDEX* array for element, $e$, is generated from Eq. 1.1 by subroutine GET_ELEM_INDEX.

Figure 1.4.2 shows how that index could be computed for a node or element for the common case where the number of generalized degrees of freedom per node is everywhere constant. For a single unknown per node ($n_g = 1$), as shown in Fig. 1.4.1, then the nodal degree of freedom loop (at lines 16 and 21 in Fig. 1.4.2) simply equates the equation number to the global node number. An example where there are two unknowns per node is illustrated in Fig. 1.4.3. That figure shows a line element mesh with two nodes per element and two dof per node (such as a standard beam element). In that case it is similar to the assembly of Fig. 1.4.1, but instead of a single coefficient we are adding a set of smaller square sub-matrices into **S**. Figure 1.4.4 shows how the assembly can be implemented for column matrices (subroutine STORE_COLUMN) and full (non-sparse) square matrices (STORE_FULL_SQUARE) if one has an integer index that

**Table 1.4.1 Degree of freedom numbers at system node $I_s$**

| Local | System† | |
|---|---|---|
| | 1 | $INDEX\,(1)$ |
| | 2 | $INDEX\,(2)$ |
| | • | • |
| | • | • |
| | • | • |
| | J | $INDEX\,(J)$ |
| | • | • |
| | • | • |
| | • | • |
| | $n_g$ | $INDEX\,(n_g)$ |

† $INDEX\,(J) = n_g * (\,I_s - 1\,) + J$

**Table 1.4.2 Relating local and system equation numbers**

| Local node $I_L$ | Parameter number $J$ | System node $I_S = node\,(\,I_L\,)$ | Element degree of freedom numbers: Local $n_g * (\,I_L - 1\,) + J$ | Element degree of freedom numbers: System $n_g * (\,I_S - 1\,) + J$ |
|---|---|---|---|---|
| 1 | 1 | $node(1)$ | 1 | $n_g * [node(1)-1] + 1$ |
| 1 | 2 | $node(1)$ | 2 | |
| • | • | • | | |
| • | • | • | • | • |
| 1 | $n_g$ | $node(1)$ | • | • |
| 2 | 1 | $node(2)$ | • | • |
| • | • | • | | |
| • | • | • | | |
| • | • | • | | |
| $K$ | $J_g$ | $node(K)$ | $n_g * (K-1) + J_g$ | $n_g * [node(K)-1] + J_g$ |
| • | • | • | | |
| • | • | • | • | • |
| $n_n$ | 1 | $node(n_n)$ | • | • |
| • | • | • | • | • |
| • | • | • | | |
| $n_n$ | $n_g$ | $node(n_n)$ | $n_n * n_g$ | $n_g * [node(n_n)-1] + n_g$ |

relates the local element degrees of freedom to the system dof.

### 1.4.3 Example Equation Numbers

Consider a two-dimensional problem ($n_s = 2$) involving 400 nodal points ($n_m = 400$) and 35 elements ($n_e = 35$). Assume two parameters per node ($n_g = 2$) and let these parameters represent the horizontal and vertical components of some vector. In a stress analysis problem, the vector could represent the displacement vector of the node, whereas in a fluid flow problem it could represent the velocity vector at the nodal point. Assume the elements to be triangular with three corner nodes ($n_n = 3$). The local numbers of these nodes will be defined in some consistent manner, e.g., by numbering counter-clockwise from one corner. This mesh is illustrated in Fig. 1.4.5.

By utilizing the above control parameters, it is easy to determine the total number of degrees of freedom in the system, $n_d$, and associated with a typical element, $n_i$ are: $n_d = n_m * n_g = 400 * 2 = 800$, and $n_i = n_n * n_g = 3 * 2 = 6$, respectively. In addition to the total number of degrees of freedom in the system, it is important to be able to identify the system degree of freedom number that is associated with any parameter in the system. Table 1.4.2 or subroutine GET_DOF_INDEX, provides this information. This relation has many practical uses. For example, when one specifies that the first parameter ($J = 1$) at system node 50 ($I_S = 50$) has some given value what one is indirectly saying is that system degree of freedom number $DOF = 2 * (50 - 1) + 1 = 99$ has a given value. In a similar manner, we often need to identify the system degree of freedom numbers that

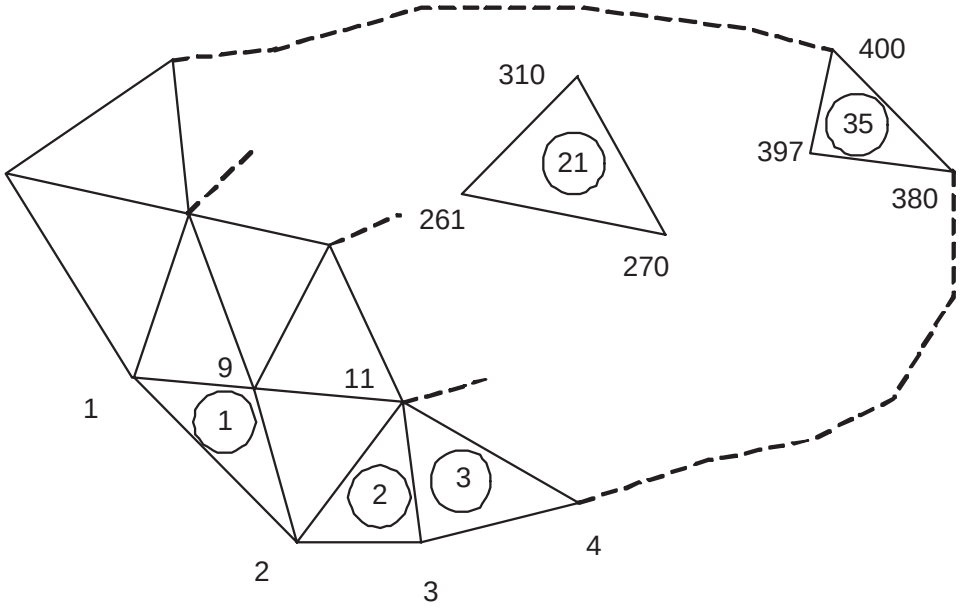



Figure 1.4.5 Mesh for assembly equation number calculations

**Table 1.4.3 Example mesh connectivity**

| Element | Element indices | | |
|---|---|---|---|
| 1 | 1 | 2 | 9 |
| 2 | 2 | 3 | 11 |
| 3 | 3 | 4 | 11 |
| ⋮ | ⋮ | ⋮ | ⋮ |
| 21 | 261 | 270 | 310 |
| ⋮ | ⋮ | ⋮ | ⋮ |
| 35 | 397 | 398 | 400 |
| | Array NODES | | |

**Table 1.4.4 Example element and system equation numbers**

| Node local $i_L$ | Number system $I_S$ | Parameter number $J$ | Degree of Freedom System $DOF_S$ | Degree of Freedom Local $DOF_L$ |
|---|---|---|---|---|
| 1 | 261 | 1 | 521 | 1 |
| 1 | 261 | 2 | 522 | 2 |
| 2 | 270 | 1 | 539 | 3 |
| 2 | 270 | 2 | 540 | 4 |
| 3 | 310 | 1 | 619 | 5 |
| 3 | 310 | 2 | 620 | 6 |
| | Array *ELEM_NODES* | | Array *INDEX* | |

correspond to the $n_i$ local degrees of freedom of the element. In order to utilize Eq. 1.8 to do this, one must be able to identify the $n_n$ node numbers associated with the element of interest. This is relatively easy to accomplish since those data are part of the input data (element incidences). For example from Table 1.4.3, for element number 21 we find the three element incidences (row 21 of the system connectivity array) to be

$$\begin{array}{lcccl} \text{System} & 261 & 270 & 310 & \leftarrow \text{Array ELEM\_NODES} \\ \text{Local} & 1 & 2 & 3 & (1 \times n_n) \end{array}$$

Therefore, by applying Eq. 1.1, we find the degree of freedom numbers in Table 1.4.4.

The element array *INDEX* has many programming uses. Its most important application is to aid in the assembly (scatter) of the element equations to form the governing system equations. We see from Fig. 1.4.2 that the element equations are expressed in terms of local degree of freedom numbers. In order to add these element coefficients into the system equations one must identify the relation between the local degree of freedom numbers and the corresponding system degree of freedom numbers. Array *INDEX* provides this information for a specific element. In practice, the assembly procedure is as follows. First the system matrices **S** and **C** are set equal to zero. Then a loop over all the elements if performed. For each element, the element matrices are generated in terms of the local degrees of freedom. The coefficients of the element matrices are added to the corresponding coefficients in the system matrices. Before the addition is carried out, the element array *INDEX* is used to convert the local subscripts of the coefficient to the system subscripts of the term in the system equations to which the coefficient is to be added. That is, we scatter

$$S^e_{i,j} \overset{+}{\rightarrow} S_{I,J}\ , \qquad C^e_i \overset{+}{\rightarrow} C_I \tag{1.11}$$

where $I_S = INDEX\,(\,i_L\,)$ and $J_S = INDEX\,(\,j_L\,)$ are the corresponding row and column numbers in the system equations, $i_L$, $j_L$ are the subscripts of the coefficients in terms of the local degrees of freedom, and $\overset{+}{\rightarrow}$ reads "is added to". Considering all of the the terms in the element matrices for element 21 in the previous example, one finds six typical scatters from the $\mathbf{S}^e$ and $\mathbf{C}^e$ arrays are

$$\begin{aligned}
S^e_{1,1} &\overset{+}{\rightarrow} S_{521,521} \qquad & C^e_1 &\overset{+}{\rightarrow} C_{521}\\
S^e_{2,3} &\overset{+}{\rightarrow} S_{522,539} \qquad & C^e_2 &\overset{+}{\rightarrow} C_{522}\\
S^e_{3,4} &\overset{+}{\rightarrow} S_{539,540} \qquad & C^e_3 &\overset{+}{\rightarrow} C_{539}\\
S^e_{4,5} &\overset{+}{\rightarrow} S_{540,620} \qquad & C^e_4 &\overset{+}{\rightarrow} C_{540}\\
S^e_{5,6} &\overset{+}{\rightarrow} S_{619,620} \qquad & C^e_5 &\overset{+}{\rightarrow} C_{619}\\
S^e_{1,6} &\overset{+}{\rightarrow} S_{521,620} \qquad & C^e_6 &\overset{+}{\rightarrow} C_{620}\ .
\end{aligned}$$

## 1.5 Error Concepts

Part of the emphasis of this book will be on error analysis in finite element studies. Thus this will be a good point to mention some of the items that will be of interest to us later. We will always employ integral forms. Denote the highest derivative occurring in the integral by the integer $m$. Assume all elements have the same shape and use the same interpolation polynomial. Let the characteristic element length size be the real value $h$, and assume that we are using a complete polynomial of integer degree $p$. Later we will be interested in the asymptotic convergence rate, in some norm, as the element size

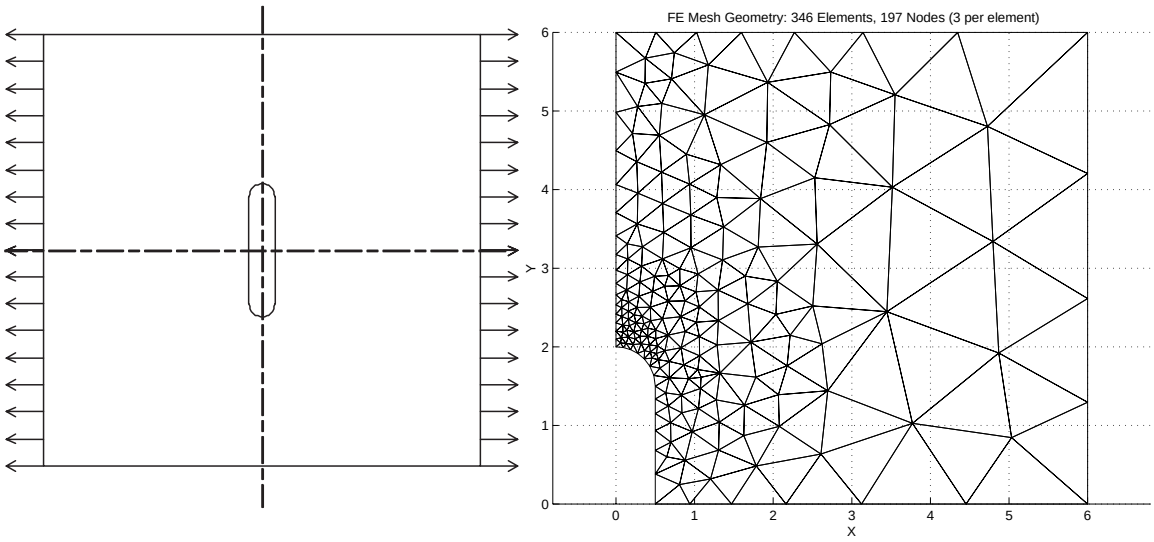



Figure 1.5.1 Relating element size to expected gradients

approaches zero, $h \to 0$. Here we will just mention the point wise error that provides the insight into creating a good manual mesh or a reasonable starting point for an adaptive mesh. For a problem with a smooth solution the local finite element error is proportional to the product of the $m-th$ derivative at the point and the element size, $h$, raised to the $p-th$ power. That is,

$$e(x) \propto h^p \, \partial^m u(\mathbf{x}) / \partial \mathbf{x}^m \, . \tag{1.12}$$

This provides some engineering judgement in establishing an initial mesh. Where you expect the gradients (the $m-th$ derivative) to be high then make the elements very small. Conversely, where the gradients will be zero or small we can have large elements to reduce the cost. These concepts are illustrated in Fig. 1.5.1 where the stresses around a hole in a large flat plate are shown. There we see linear three noded triangles (so $p = 1$) in a quarter symmetry mesh. Later we will show that the integral form contains the first derivatives (gradient, so $m = 1$). Undergraduate studies refer to this as a stress concentration problem and show that the gradients rapidly increase by a factor of about 3 over a very small region near the top of the hole. Thus we need small elements there. At the far boundaries the tractions are constant so the gradients displacement are nearly zero there and the elements can be big. Later we will automate estimating local error calculations and the associated element size changes needed for an accurate and cost effective solution.

## 1.6 Exercises

1. Assume (unrealistically) that all the entries in an element square matrix and column vector are equal to the element number. Carry out the assembly of the system in Fig. 1.4.1 to obtain the final numerical values for each coefficient in the **S** and **C** matrices.

Hint: manually loop over each element and carry out the line by line steps given in *GET_ELEM_INDEX*, and then those in *STORE_COLUMN*, and finally those in *STORE_FULL_SQUARE* before going to the next element.

2. Assume (unrealistically) that all the entries in an element square matrix and column vector are equal to the element number. Carry out the assembly of the system in Fig. 1.4.3 to obtain the final numerical values for each coefficient in the **S** and **C** matrices.

3. In Fig. 1.4.3 assume that the global nodes are numbered consecutively from 1 to 4 (from left to right). Write the element index vector for each of the three elements.

4. List the topology (connectivity data) for the six elements in Fig. 1.3.1.

5. Why does the $\boldsymbol{\beta}$ Boolean array in Fig. 1.3.1 have two rows and seven columns?

6. What is the Boolean array, $\boldsymbol{\beta}$, for element $a(= 2)$ in Fig. 1.3.1?

7. What is the percent of sparsity of the **S** matrix in Fig. 1.4.3?

8. What is the Boolean array, $\boldsymbol{\beta}$, for element 3 in Fig. 1.4.1?

9. What is the size of the Boolean array, $\boldsymbol{\beta}$, for any element in Fig. 1.4.3? Explain why.

10. What is the Boolean array, $\boldsymbol{\beta}$, for element 1 in Fig. 1.4.3?

11. Referring to Fig. 1.4.1, multiply the given $6 \times 1$ **D** array by the $3 \times 6$ Boolean array, $\boldsymbol{\beta}$, for the second element to gather its corresponding local $\mathbf{D}^e$ local dof.

12. In a FEA stress analysis where a translational displacement is prescribed the reaction necessary for equilibrium is _____: a) heat flux, b) force vector, c) pressure, d) temperature, e) moment (couple) vector.

13. In a FEA stress analysis where a rotational displacement is prescribed the reaction necessary for equilibrium is ____: a) heat flux, b) force vector, c) pressure, d) temperature, e) moment (couple) vector.

14. In a FEA thermal analysis where a temperature is prescribed the reaction necessary for equilibrium is _____: a) heat flux, b) force vector, c) pressure, d) temperature, e) moment (couple) vector.

15. A material that is the same at all points is ___: a) homogeneous, b) non-homogeneous, c) isotropic, d) anisotropic, e) orthotropic.

16. A material that is the same in all directions at a point is ___: a) homogeneous, b) non-homogeneous, c) isotropic, d) anisotropic, e) orthotropic.

17. A material that is the same at all points is ___: a) homogeneous, b) non-homogeneous, c) isotropic, d) anisotropic, e) orthotropic.

18. A material that is the same in all directions at a point is ___: a) homogeneous, b) non-homogeneous, c) isotropic d) anisotropic, e) orthotropic.

19. Define a scalar, vector, and tensor quantity to have zero, one, and two subscripts, respectively. Identify which of the above describe the following items: _____ mass, _____ time, _____ position, _____ centroid, _____ volume, _____ surface area, _____ displacement, _____ temperature, _____ heat flux, _____ heat source, _____ stress, _____ moment of inertia, _____ force, _____ moment, _____ velocity.

20. In a finite element solution at a node, the _____: a) primary variable is most accurate, b) primary variable is least accurate, c) secondary variable is most accurate, d) secondary variable is least accurate.

21. Interior to an element in an FEA solution, the _____: a) primary variable is most accurate, b) primary variable is least accurate, c) secondary variable is most accurate, d) secondary variable is least accurate.

## 1.7 Bibliography

[1] Adams, V. and Askenazi, A., *Building Better Products with Finite Element Analysis*, Santa Fe: Onword Press (1999).

[2] Akin, J.E., *Finite Elements for Analysis and Design*, London: Academic Press (1994).

[3] Akin, J.E. and Singh, M., "Object-Oriented Fortran 90 P-Adaptive Finite Element Method," pp. 141−149 in *Developments in Engineering Computational Technology*, ed. B.H.V. Topping, Edinburgh: Civil_Comp Press (2000).

[4] Bathe, K.J., *Finite Element Procedures*, Englewood Cliffs: Prentice-Hall (1996).

[5] Becker, E.B., Carey, G.F., and Oden, J.T., *Finite Elements − An Introduction*, Englewood Cliffs: Prentice-Hall (1981).

[6] Cook, R.D., Malkus, D.S., and Plesha, N.E., *Concepts and Applications of Finite Element Analysis*, New York: John Wiley (1989).

[7] Cook, R.D., Malkus, D.S., Plesha, N.E., and Witt, R.J., *Concepts and Applications of Finite Element Analysis*, New York: John Wiley (2002).

[8] Desai, C.S. and , T. Kundu, *Introduction to the Finite Element Method*, Boca Raton: CRC Press (2001).

[9] Gupta, K.K and Meek, J.L., *Finite Element Multidisciplinary Analysis*, Reston: AIAA (2000).

[10] Huebner, K.H., Thornton, E.A., and Byrom, T.G., *Finite Element Method for Engineers*, New York: John Wiley (1994).

[11] Hughes, T.J.R., *The Finite Element Method*, Englewood Cliffs: Prentice-Hall (1987).

[12] Norrie, D.H. and DeVries, G., *Finite Element Bibliography*, New York: Plenum Press (1976).

[13] Pironneau, O., *Finite Element Methods for Fluids*, New York: John Wiley (1991).
[14] Rao, S.S., *The Finite Element Method in Engineering*, Boston : Butterworth Heinemann (1999).
[15] Segerlind, L.J., *Applied Finite Element Analysis*, New York: John Wiley (1987).
[16] Silvester, P.P. and Ferrari, R.L., *Finite Elements for Electrical Engineers*, Cambridge: Cambridge University Press (1996).
[17] Smith, I.M. and Griffiths, D.V., *Programming the Finite Element Method*, 3rd Edition, Chichester: John Wiley (1998).
[18] Szabo, B. and Babuska, I., *Finite Element Analysis*, New York: John Wiley (1991).
[19] Whiteman, J.R., *A Bibliography of Finite Elements*, London: Academic Press (1975).
[20] Zienkiewicz, O.C. and Taylor, R.L., *The Finite Element Method*, 4th Edition, New York: McGraw-Hill (1991).