ELEMENT-BY-ELEMENT SOLUTION PROCEDURES FOR NONLINEAR TRANSIENT HEAT CONDUCTION ANALYSIS

Thesis by

James Michael Winget

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

1984

(Submitted August 22, 1983)

Acknowledgements

I wish to thank my advisor, Professor Thomas J. R. Hughes, for his guidance, assistance, and encouragement throughout this work.

Two others who also deserve mention are my colleague Itzhak Levit for his active participation in the early stages of this research and K. C. Park for his many helpful suggestions during my time at Lockheed.

This research was made possible by financial support from the International Business Machines Corporation, NASA Langley Research Center (Grant No. NAG-1-259), and the California Institute of Technology. Computing resources for early software development were donated by the Lockheed Corporation. Most important however, was the strong support, financial and otherwise, of the Sutherlands and the Sproulls, without which this work might not have been completed.

Special thanks go to my colleague Christine Miller for proofreading the manuscript and urging me on to the light at the end of the tunnel.

Finally, I wish to express my deepest gratitude to my family, who provided wholehearted encouragement and support every step of the way.

Abstract

Despite continuing advancements in computer technology, there are many problems of engineering interest that exceed the combined capabilities of today's numerical algorithms and computational hardware. The resources required by traditional finite element algorithms tend to grow geometrically as the "problem size" is increased. Thus, for the forseeable future, there will be problems of interest which cannot be adequately modeled using currently available algorithms. For this reason, we have undertaken the development of algorithms whose resource needs grow only linearly with problem size. In addition, these new algorithms will fully exploit the "parallel-processing" capability available in the new generation of multi-processor computers.

The approach taken in the element-by-element solution procedures is to approximate the global implicit operator by a product of lower order operators. This type of "product" approximation originated with ADI techniques and was further refined into the "method of fractional steps." The current effort involves the use of a more natural operator split for finite element analysis based on "element operators." This choice of operator splitting based on element operators has several advantages. First, it fits easily within the architecture of current FE programs. Second, it allows the development of "parallel" algorithms. Finally, the computational expense varies only linearly with the number of elements.

The particular problems considered arise from nonlinear transient heat conduction. The nonlinearity enters through both material temperature dependence and radiation boundary conditions. The latter condition typically introduces a "stiff" component in the

resultant matrix ODE's which precludes the use of explicit solution techniques. Implicit solution techniques can be prohibitively expensive. Instead, the matrix equations are solved by combining a modified Newton–Raphson iteration scheme with an element-by-element preconditioned conjugate gradient subiteration procedure. The resultant procedure has proven to be both accurate and reliable in the solution of medium-size problems in this class.

Contents

Plates

Chapter 1

Introduction

The finite element method is a useful technique for obtaining approximate solutions to many problems of engineering interest. This is accomplished through a multi-level procedure that combines spatial discretization with an appropriate weighted residual formulation to yield a set of nonlinear ordinary differential equations. These equations are then solved using a combination of transient and nonlinear equation solution algorithms, resulting in a linear system of equations that must be solved at each iteration of every time step.

The particular problems considered arise from nonlinear transient heat conduction. Two types of nonlinearities are considered. The first is a material temperature dependence which is frequently needed to accurately model problem behavior over the range of temperatures of engineering interest. The second nonlinearity is introduced by radiation boundary conditions. The resultant matrix ordinary differential equations are typically "stiff." This "stiffness" precludes the use of "explicit" solution techniques due to severe restrictions on time step size. Thus, "implicit" methods which entail the storage and solution of sparse global matrix equations are generally called for.

The size of the resulting matrix equations is governed by the degree of "spatial complexity" one wishes to resolve. In general, accurately obtaining the "fine-scale" solution to a "large" problem results in a large equation system. The accurate solution

of many problems of current interest result in systems with more than a quarter of a million equations. The primary costs associated with the solution of these large problems lie in the storage and solution of the global matrix equations. As an example, the cost incurred in the solution of a three-dimensional, nonlinear, transient heat conduction problem using a regular $63 \times 63 \times 63$ mesh (250,047 equations) is 1×10^9 words of storage and 2×10^{12} operations per iteration per step. Translating these costs into more easily visualized terms, it would require approximately $15\frac{1}{2}$ large disk drives to hold the matrix and an estimated $5\frac{1}{2}$ hours of CPU time on a Cray-1 to factor the matrix and obtain the solution for one iteration (this ignores an estimated $2\frac{1}{2}$ days of IO time needed to read and write the matrix from disk). Clearly, traditional solution techniques do not offer the performance needed for solving large problems, even when implemented on today's "super computers" [Juba 82], [Noor 82], [Robinson 82].

In an effort to reduce the costs associated with the solution of large transient heat conduction problems, an element-by-element (EL×EL) algorithm was presented by Hughes, Levit, and Winget in [Hughes 83a]. In that work, the EL×EL concept was used to develop a non-iterative, second-order time-accurate, unconditionally stable, transient algorithm for both linear and nonlinear problems. In the EL×EL method, the global matrix is never formed but, instead, approximated by a product of element operators, each of which is processed individually. An important result is that the storage and solution costs tend to grow only linearly with problem size as compared to the geometric growth of a globally implicit method.

This work resulted from a synthesis of the finite element method with the approximate factorization techniques developed for the solution of finite difference equations. These approximate factorization techniques developed from the spatial-operator splitting ADI technique presented by Douglas and Rachford in [Douglas 56,62] and the more general "fractional step" approach taken by Marchuk and Yanenko in [Marchuk 74,75] and [Yanenko 71].

Unlike the aforementioned finite difference techniques, the EL×EL method retains

the generality of the finite element method and, as such, imposes no geometric or topological restrictions on the problem domain or its spatial discretization. In addition, the EL×EL method fits naturally within the architecture of current finite element programs. Perhaps the single most significant feature of the EL×EL method is the ease with which it may be efficiently implemented on a multi-processor computer. This last feature can not be over emphasized, since it is clear that the trend in "super computers" is towards parallel-processing machines.[1]

Although our initial numerical testing of the EL×EL time integration algorithm showed promise, later analysis indicated that, under certain circumstances, the accuracy of the governing implicit method was not attained. Loss of accuracy was traced to spatial truncation error terms similar to those encountered in some of the classical split-operator finite difference methods such as the DuFort-Frankel method [Ames 77].

To overcome these accuracy deficiencies, we were led to reformulate the EL×EL procedure as an iterative linear equation solution procedure [Hughes 83c]. This allowed the use of standard time-discretization and solution techniques. Thus, the issues of accuracy and stability are resolved by the choice of time-integration technique.

There are three main ingredients in an EL×EL iterative linear equation solution algorithm. These are: an iterative driver algorithm, a matrix which approximates the global implicit matrix and is amenable to EL×EL approximate factorization, and, of course, the EL×EL approximate factorization scheme itself. Note that the architectural simplicity and cost reduction of the original EL×EL algorithm are retained in the iterative solution algorithm.

An outline of the remainder of the thesis follows :

Chapter 2 formulates the finite element equations arising from the solution of nonlinear transient heat conduction problems. The finite element matrix equations

[1] A good discussion of the need for parallelism in hardware and software for the solution of large problems is given in a recent CAL TECH research proposal by Fox and Seitz [Fox 83].

are then temporally discretized and a nonlinear iterative solution algorithm proposed. Finally, the resultant linear equation system is discussed.

In Chapter 3, we explore the techniques available for solving linear equation systems. We consider both direct and iterative techniques for solving the linear system. We also discuss the costs incurred by both of the aforementioned techniques.

Chapter 4 introduces the concept of approximate factorizations. First, the simplest approximate factorization, based on two-component splitting, is defined. Generalization of this leads to the definition of multi-component splittings. These, in turn, are used to define a variety of element-by-element approximate factorizations based on both "sum" and "product" element-level factorizations. Finally, we consider the optimal choice of the matrix to be approximated and the size of the error in the approximation.

In Chapter 5, we discuss a variety of implementational aspects. We first describe subiteration algorithms; these are combinations of EL×EL approximate factorizations and iterative algorithms that we have found successful in the solution of the linear equation system. Next, the convergence measures used to terminate an iterative scheme are considered. Based on an analysis of the effect of error in the subiteration loop on the convergence of the iteration loop, optimal convergence criteria for the subiteration loop are developed. Next, we consider the use of step-size selection strategies to control the error in the transient solution. The use of substructuring to reduce computational costs is considered. Some of the additional benefits of the EL×EL solution algorithms are then briefly outlined. Finally, the modifications required to implement EL×EL algorithms on parallel-processor machines are considered.

Chapter 6 contains a discussion of numerical results obtained using EL×EL solution algorithms. A variety of both linear and nonlinear problems are solved and the EL×EL algorithms are shown to be especially well suited for solving nonlinear problems.

Finally, in Chapter 7, we present conclusions.

Chapter 2

Derivation of Finite Element Equations for Nonlinear Transient Heat Conduction

This chapter formulates in detail the finite element equations arising from the solution of problems in nonlinear transient heat conduction. First, the strong form of the initial boundary-value problem is defined. This leads to the weighted residual or weak form of the problem. Next, the Galerkin approximation is introduced which, along with an assumed spatial discretization, leads to the finite element matrix form of the problem. The finite element matrix equations are then temporally discretized. The solution of the resulting nonlinear matrix equations motivates the development of iterative algorithms. Finally, the linear problem generated by the iterative algorithms is discussed.

§2.1 Strong Form of the Initial Boundary-Value Problem

To define the strong form, we must first define the problem domain, material properties, and boundary and initial conditions.

2.1.1 Problem Domain

The problem is posed for a body occupying a spatial domain Ω, a finite region of $\Re^{N_{SD}}$, where $\Re$ is the set of real numbers and N_{SD} is the number of space dimensions.

A general point in $\overline{\Omega}$ will be denoted as $\mathbf{x} = \{x_i\}$, $i = 1, 2, \ldots, N_{SD}$. The boundary of Ω, denoted Γ, is assumed to be piecewise smooth. At every point on Γ there is a unique outward normal unit vector $\mathbf{n} = \{n_i\}$, $i = 1, 2, \ldots, N_{SD}$. In addition, Γ can be subdivided into two disjoint subsets, Γ_g and Γ_h. Thus Γ admits the following decomposition :

$$\Gamma = \overline{\Gamma_g \bigcup \Gamma_h} \tag{2.1.1}$$

and

$$\emptyset = \Gamma_g \bigcap \Gamma_h \,. \tag{2.1.2}$$

In (2.1.1) , the superposed bar represents set closure and in (2.1.2) , $\emptyset$ denotes the empty set. The time interval under consideration is 0 to $\mathcal{T}$, where $\mathcal{T}$ is a given real positive number.

2.1.2 Material Properties

In order to model a wide variety of nonlinear heat conduction problems, very few restrictions will be placed on the material composing the body. The material is inhomogeneous. Therefore the mass density ρ, specific heat C_p, conductivity $\mathbf{K}$, and internal heat generation Q may vary throughout the body. Nonlinear temperature dependence as well as time dependence is allowed in C_p, $\mathbf{K}$, and Q. Finally, $\mathbf{K}$ is allowed to be anisotropic although symmetry is assumed [Carslaw 59]. Thus, the domains and ranges of the functions defining the material properties are given as follows :

$$\rho : \Omega \mapsto \Re^+ \qquad \text{(mass density)}, \tag{2.1.3}$$

$$C_p : \Re^+ \times \Omega \times]0, \mathcal{T}[\mapsto \Re^+ \qquad \text{(specific heat)}, \tag{2.1.4}$$

$$\mathbf{K} : \Re^+ \times \Omega \times]0, \mathcal{T}[\mapsto \mathrm{S\overset{+}{y}m} \qquad \text{(thermal conductivity)}, \tag{2.1.5}$$

$$Q : \Re^+ \times \Omega \times]0, \mathcal{T}[\mapsto \Re \qquad \text{(internal heat generation)}. \tag{2.1.6}$$

In the preceding relations, $\Re^+$ denotes positive real numbers and $\mathrm{S\overset{+}{y}m}$ is the set of all symmetric positive-definite two-tensors.

2.1.3 Boundary and Initial Values

The boundary conditions defined on Γ fall into two distinct subsets. The first subset, Γ_g of Γ, contains those points at which a "g-type" or "essential" boundary condition is applied. This type of boundary condition prescribes a given temperature g which may depend on both position and time, thus

$$g : \Gamma_g \times]0, \mathcal{T}[\mapsto \Re^+ \quad \text{(prescribed temperature)} . \tag{2.1.7}$$

The other subset of Γ, Γ_h, contains those points at which an "h-type" or "natural" boundary condition is applied. This type of boundary condition applies a heat flux loading h which may depend on temperature as well as position and time, thus

$$h : \Re^+ \times \Gamma_h \times]0, \mathcal{T}[\mapsto \Re^{N_{SD}} \quad \text{(prescribed heat flux)} . \tag{2.1.8}$$

Several of the more common forms of the function h are

1. *Adiabatic* or *no-flux* boundary condition :

$$h = \mathbf{o} . \tag{2.1.9}$$

2. Prescribed heat flux boundary condition :

$$h = h_f(\mathbf{x}, t) , \tag{2.1.10}$$

where h_f is a given function.

3. *Convective* boundary condition :

$$h = h_c(\mathbf{x}, t)(T(\mathbf{x}, t) - T_c(\mathbf{x}, t)) , \tag{2.1.11}$$

where h_c is a given function defining the convection coefficient, and T_c is the given external temperature field.

4. *Radiation* boundary condition :

$$h = h_r(\mathbf{x}, t)(T^4(\mathbf{x}, t) - T_r^4(\mathbf{x}, t)) , \tag{2.1.12}$$

where the radiation coefficient, h_r, depends on the Stefan-Boltzmann constant as well as the emissivity and geometric view factors for the surface, and T_r is the given external temperature field [Siegel 81]. This condition may also be written :

$$h = \kappa(T, \mathbf{x}, t)(T(\mathbf{x}, t) - T_r(\mathbf{x}, t)) , \tag{2.1.13}$$

where κ is defined as

$$\kappa(T, \mathbf{x}, t) = h_r(\mathbf{x}, t)(T^2(\mathbf{x}, t) + T_r^2(\mathbf{x}, t))(T(\mathbf{x}, t) + T_r(\mathbf{x}, t)) . \tag{2.1.14}$$

Note that two or more of the above forms may be combined to represent a complex physical flux loading.

Finally, the initial temperature distribution in the body is

$$T_0 : \Omega \mapsto \Re^+ \quad (\text{ initial temperature }) . \tag{2.1.15}$$

For physical reasons, T_0 need not be continuous.

2.1.4 The Strong Form

The strong form of the I.B.V.P. , based on a generalization of the *Fourier Law* of heat conduction [Carslaw 59], [Holman 72], [Isachenko 75], and [Ozisik 80], is stated as

follows :

$$
(\mathbf{S})\begin{cases}
\text{Given : } \rho,\ C_p,\ \mathbf{K},\ Q,\ g,\ h,\ T_0 \text{ as in (2.1.3)—(2.1.15)} , \\
\text{Find :} \\
\qquad T : \bar{\Omega} \times [0, \mathcal{T}] \mapsto \Re^+ , \\
\text{Such that :} \\
\qquad \rho C_p \dfrac{\partial T}{\partial t} = \nabla \cdot (\mathbf{K} \nabla T) + Q \quad \text{on} \quad \Omega \times]0, \mathcal{T}[, & (2.1.16) \\
\qquad T = g \quad \text{on} \quad \Gamma_g \times]0, \mathcal{T}[, & (2.1.17) \\
\qquad \mathbf{n} \cdot (\mathbf{K} \nabla T) = h \quad \text{on} \quad \Gamma_h \times]0, \mathcal{T}[, & (2.1.18) \\
\qquad T(\mathbf{x}, 0) = T_0(\mathbf{x}) \quad \forall \quad \mathbf{x} \in \Omega . & (2.1.19)
\end{cases}
$$

§2.2 Weighted Residual Form of the Initial Boundary-Value Problem

The "weighted residual" or "weak" form of (S) is generated by a suitable choice of solution and variational spaces and the application of the divergence theorem [Strang 73], [Mitchell 77].

2.2.1 Solution and Variational Spaces

In order to develop a weak formulation for the I.B.V.P. , a trial solution and a variational space are defined. Let u, v, and w denote temperature-like fields. We define the trial solution space $\mathcal{S}$ as follows :

$$\mathcal{S} = \{ u \mid u = g \quad \text{on} \quad \Gamma_g \} , \tag{2.2.1}$$

and the variational space $\mathcal{V}$ as

$$\mathcal{V} = \{ w \mid w = 0 \quad \text{on} \quad \Gamma_g \} . \tag{2.2.2}$$

Note that $\mathcal{S}$ is time-dependent due to its use of the g-type boundary condition while $\mathcal{V}$ is not time-dependent.

2.2.2 The Weak Form

The weak form of the problem (**W**) is obtained by multiplying (2.1.16) and (2.1.19) by $w \in \mathcal{V}$, integrating over Ω, applying the divergence theorem, and making use of the boundary conditions (2.1.17) and (2.1.18) to simplify the result. This yields the following weak form for the I.B.V.P. :

$$
(\mathbf{W})\begin{cases}
\text{Given : } \rho,\ C_p,\ \mathbf{K},\ Q,\ g,\ h,\ T_0 \text{ as in (2.1.3)—(2.1.15)} , \\
\text{Find :} \\
\qquad u : [0, \mathcal{T}] \mapsto \mathcal{S} , \\
\text{Such that for every } w \in \mathcal{V} : \\
\qquad \mathcal{M}(u; \dot{u}, w) + \mathcal{N}(u; u, w) = \mathcal{F}(u; Q, w) + \mathcal{H}(u; h, w) \quad \text{on} \quad]0, \mathcal{T}[, \qquad (2.2.3) \\
\qquad (u(\mathbf{x}, 0) - T_0, w) = 0 \quad \text{on} \quad \Omega . \qquad (2.2.4)
\end{cases}
$$

The operators $\mathcal{M}$, $\mathcal{N}$, $\mathcal{F}$, $\mathcal{H}$, and $(\cdot, \cdot)$ are defined as :

$$\mathcal{M}(v; \dot{u}, w) = \int_{\Omega} \rho(\mathbf{x}) C_p(v(\mathbf{x}, t), \mathbf{x}, t) \dot{u}(\mathbf{x}, t) w(\mathbf{x})\, d\Omega , \qquad (2.2.5)$$

$$\mathcal{N}(v; u, w) = \int_{\Omega} \nabla u(\mathbf{x}, t) \cdot \mathbf{K}(v(\mathbf{x}, t), \mathbf{x}, t) \nabla w(\mathbf{x})\, d\Omega , \qquad (2.2.6)$$

$$\mathcal{F}(v; Q, w) = \int_{\Omega} Q(v(\mathbf{x}, t)) w(\mathbf{x})\, d\Omega , \qquad (2.2.7)$$

$$\mathcal{H}(v; h, w) = \int_{\Gamma_h} h(v(\mathbf{x}, t), \mathbf{x}, t) w(\mathbf{x})\, d\Gamma , \qquad (2.2.8)$$

$$(u, w) = \int_{\Omega} u(\mathbf{x}, t) w(\mathbf{x})\, d\Omega . \qquad (2.2.9)$$

Remarks:

1. Given suitable smoothness conditions, **(S)** $\Leftrightarrow$ **(W)**.

2. $M(v;\cdot,\cdot)$, $N(v;\cdot,\cdot)$, and $(\cdot,\cdot)$ are symmetric bilinear forms.

§2.3 Galerkin Approximation of the Initial Boundary-Value Problem

The Galerkin form is derived from the weak form by approximating the variational and solution spaces with finite subspaces.

2.3.1 Galerkin Spaces of Approximation

The Galerkin approximation uses a finite number of linearly independent functions to span a subspace $\mathcal{V}^h$ of the variational space $\mathcal{V}$. We may represent $\mathcal{V}^h$ as

$$\mathcal{V}^h = \{ w^h \mid w^h = \sum_{A=1}^{n} N_A(\mathbf{x}) d_A; \quad w^h = 0 \quad \text{on} \quad \Gamma_g \} \subset \mathcal{V} , \tag{2.3.1}$$

where N_A, $A = 1, 2, \ldots, n$, are n linearly independent functions in $\mathcal{V}$ and d_A are constants. Similarly, the approximation to the trial solution space $\mathcal{S}^h$ is defined as

$$\mathcal{S}^h = \{ u^h \mid u^h = v^h + g^h, \quad v^h \in \mathcal{V}^h, \quad g^h \in \mathcal{S} \} \subset \mathcal{S} . \tag{2.3.2}$$

2.3.2 The Galerkin Form

The Galerkin approximation of the I.B.V.P. may be stated as

(G)
Given : ρ, C_p, $\mathbf{K}$, Q, g, h, T_0 as in (2.1.3)—(2.1.15) ,

Find :

$$u^h = v^h + g^h : [0, \mathcal{T}] \mapsto \mathcal{S}^h ,$$

Such that for every $w^h \in \mathcal{V}^h$:

$$\mathcal{M}(u^h; \dot{v}^h, w^h) + \mathcal{N}(u^h; v^h, w^h) = \mathcal{F}(u^h; Q, w^h) + \mathcal{H}(u^h; h, w^h) - \mathcal{M}(u^h; \dot{g}^h, w^h) - \mathcal{N}(u^h; g^h, w^h) \quad \text{on} \quad]0, \mathcal{T}[, \tag{2.3.3}$$

$$(v^h(\mathbf{x}, 0), w^h) = (T_0 - g^h(\mathbf{x}, 0), w^h) \quad \text{on} \quad \Omega . \tag{2.3.4}$$

§2.4 Finite Element Matrix Approximation of the Initial Boundary-Value Problem

The finite element matrix equations are derived from the Galerkin form by defining the approximation of the variational and solution spaces based on a given spatial discretization [Zienkiewicz 77], [Bathe 82], [Akin 82].

2.4.1 Spatial Discretization

To obtain the finite element matrix form of the I.B.V.P. , the domain Ω must be discretized into disjoint element subdomains Ω^e, $e = 1, 2, \ldots, N_{EL}$, where N_{EL} is the number of elements. Thus

$$\bigcup_{e=1}^{N_{EL}} \overline{\Omega}^e = \overline{\Omega} , \tag{2.4.1}$$

and

$$\bigcap_{e=1}^{N_{EL}} \Omega^e = \emptyset . \tag{2.4.2}$$

Γ^e is the boundary of an element subdomain Ω^e.

Each element subdomain is defined by an ordered set of nodal points. These element nodal points belong to the set η of nodal points contained in the domain Ω. A subset of η, η_g, contains those nodal points on the prescribed temperature boundary Γ_g. The number of nodes in η is N_{NP}, while the number of nodes in $\eta - \eta_g$ is N_{EQ}.

2.4.2 Approximation Spaces

It is desirable to define a basis for the Galerkin spaces $\mathcal{S}^h$ and $\mathcal{V}^h$ which affords easy computability. For this reason, we choose interpolation functions based at nodal points which are only *"locally"* non-zero (in the elements connected to the given node). The interpolation functions have the following properties :

$$N_A(\mathbf{x}) = \begin{cases} 1, & \mathbf{x} = \mathbf{x}_A \\ > 0, & \mathbf{x} \text{ is } \textit{"local"} \\ 0, & \mathbf{x} = \mathbf{x}_B,\ B \neq A \\ 0, & \mathbf{x} \text{ is } \textit{"non-local"} \end{cases} \qquad A, B \in \eta \tag{2.4.3}$$

where $\mathbf{x}_A$ and $\mathbf{x}_B$ are the coordinates of nodes A and B respectively.

This choice of interpolation functions allows us to represent functions in the Galerkin spaces as :

$$v^h(\mathbf{x}, t) = \sum_{A \in \eta - \eta_g} N_A(\mathbf{x}) d_A(t) : [0, T] \mapsto \mathcal{V}^h , \tag{2.4.4}$$

and

$$g^h(\mathbf{x}, t) \approx \sum_{A \in \eta_g} N_A(\mathbf{x}) g_A(t) : [0, T] \mapsto \mathcal{S}^h . \tag{2.4.5}$$

Thus, a function in $\mathcal{V}^h$ or $\mathcal{S}^h$ may be represented as a time-varying vector $\mathbf{d}$ of N_{EQ} components that are the coefficients associated with the *shape functions* N_A, $A \in \eta - \eta_g$.

Note that the time-dependent functions g_A are chosen to be a "good" approximation to the exact g-type boundary conditions in an appropriate sense.

2.4.3 The Finite Element Matrix Form

The finite element matrix form of the I.B.V.P. is

$$
(\mathrm{M})\begin{cases}
\text{Given : } \rho,\ C_p,\ \mathbf{K},\ Q,\ g,\ h,\ T_0 \text{ as in (2.1.3)—(2.1.15)} , \\
\text{Find :} \\
\qquad \mathbf{d} : [\,0, \mathcal{T}\,] \mapsto \Re^{N_{EQ}} , \\
\text{Such that :} \\
\qquad \mathbf{M}(\mathbf{d}, t)\dot{\mathbf{d}} + \mathbf{N}(\mathbf{d}, t) = \mathbf{F}(\mathbf{d}, t) \quad \text{on} \quad]0, \mathcal{T}[\ , \qquad (2.4.6) \\
\qquad \mathbf{d}(0) = \mathbf{d}^0 \ . \qquad (2.4.7)
\end{cases}
$$

Here $\mathbf{d}(t)$ is the vector of nodal temperatures at time t, and $\mathbf{d}^0$ is a "good" approximation to the exact initial temperature T_0. The capacity matrix $\mathbf{M}$ is

$$
\mathbf{M}(\mathbf{d}, t) = \mathop{\mathsf{A}}_{e=1}^{N_{EL}} \mathbf{m}^e(\mathbf{d}^e, t) , \qquad (2.4.8)
$$

where the element capacity matrix is

$$
\mathbf{m}^e(\mathbf{d}^e, t) = [m^e_{ab}(\mathbf{d}^e, t)] , \qquad (2.4.9)
$$

with components

$$
m^e_{ab}(\mathbf{d}^e, t) = \int_{\Omega^e} N_a(\mathbf{x})\rho(\mathbf{x})C_p(u^h, \mathbf{x}, t)N_b(\mathbf{x})\, d\Omega \ . \qquad (2.4.10)
$$

The finite element assembly operator $\mathbf{A}$ in (2.4.8) is a combination of summation and a Boolean map which expands the element contribution to global size. The mapping converts local degrees of freedom to their equivalent global degrees of freedom.

The conductive flux vector $\mathbf{N}$ is

$$\mathbf{N}(\mathbf{d},t) = \mathop{\mathbf{A}}_{e=1}^{N_{EL}} \mathbf{n}^e(\mathbf{d}^e,t)\,, \tag{2.4.11}$$

where the element conductive flux vector is

$$\mathbf{n}^e(\mathbf{d}^e,t) = \{n_a^e(\mathbf{d}^e,t)\}\,, \tag{2.4.12}$$

with components

$$n_a^e(\mathbf{d}^e,t) = \int_{\Omega^e} \mathbf{B}_a^T(\mathbf{x})\mathbf{K}(u^h,\mathbf{x},t)\mathbf{B}_b(\mathbf{x})d_b^e\,d\Omega\,. \tag{2.4.13}$$

Finally, the internal heat flux and h-type boundary conditions vector $\mathbf{F}$ is

$$\mathbf{F}(\mathbf{d},t) = \mathop{\mathbf{A}}_{e=1}^{N_{EL}} \mathbf{f}^e(\mathbf{d}^e,t)\,, \tag{2.4.14}$$

where the element contribution is given by

$$\mathbf{f}^e(\mathbf{d},t) = \{f_a^e(\mathbf{d}^e,t)\}\,, \tag{2.4.15}$$

with components

$$f_a^e(\mathbf{d}^e,t) = \int_{\Omega^e} N_a(\mathbf{x})Q(u^h,\mathbf{x},t)\,d\Omega + \int_{\Gamma^e \bigcap \Gamma_h} N_a(\mathbf{x})h(u^h,\mathbf{x},t)\,d\Gamma\,. \tag{2.4.16}$$

In the preceding definitions, the subscripts $a,b = 1,2,\ldots,N_{EN}$, where N_{EN} is the number of element nodes. Repeated subscripts denote summation, and the e superscript

denotes the element number. Within an element e the temperature field is interpolated by

$$u^h = \sum_{a=1}^{N_{EL}} N_a(\mathbf{x}) d_a^e(t) \,, \tag{2.4.17}$$

where the element nodal temperatures d_a^e are defined as

$$d_a^e = \begin{cases} d_A(t), & \text{if } A \text{ is } \textit{"free"} \\ g_A^h(t), & \text{if } A \text{ is } \textit{"fixed"} \,, \end{cases} \tag{2.4.18}$$

with A being the global node number corresponding to the a^{th} node of element e. The vector $\mathbf{B}_a$ contains the spatial derivatives of the shape functions and is defined as

$$\mathbf{B}_a(\mathbf{x}) = \frac{\partial N_a(\mathbf{x})}{\partial \mathbf{x}} = \left\{ N_{a,i}(\mathbf{x}) \right\} \quad i = 1, 2, \ldots, N_{SD} \,. \tag{2.4.19}$$

§2.5 Temporal Algorithm

The nonlinear transient heat conduction problem has been reduced to a set of semi-discrete nonlinear ordinary differential equations. Since no general analytic solution is possible, the next step is to discretize the dependent variable, time, and choose an algorithm for the generation of the approximate solution of the differential equations [Hughes 77].

2.5.1 Temporal Discretization

For the moment we will consider only a simple temporal discretization using constant time steps $\Delta t = T/N_{STEPS}$, where N_{STEPS} is the total number of time steps. Thus, we will compute an approximate solution to the nonlinear matrix O.D.E.'s at a

number of discrete time steps. The discrete temperature $\mathbf{d}_n$ is an approximation of the exact solution at the n^{th} time step, thus

$$\mathbf{d}_n \approx \mathbf{d}(t_n) . \tag{2.5.1}$$

Similarly, the discrete *"velocity"* $\mathbf{v}_n$ (partial derivative of temperature with respect to time) is an approximation to $\dot{\mathbf{d}}$ at the n^{th} time step, thus

$$\mathbf{v}_n \approx \dot{\mathbf{d}}(t_n) . \tag{2.5.2}$$

The discrete solution times are given by

$$t_n = n\Delta t . \tag{2.5.3}$$

2.5.2 Generalized Trapezoidal Rule

The time integration method chosen is the generalized trapezoidal rule. Applying it to (**M**) leads to the following time integration scheme :

(**T**)

Given : **M**, **N** and **F** as in (2.4.8)—(2.4.16) ,

Find :

$$\mathbf{d}_n \quad n \in \{0, 1, \ldots, N_{STEPS}\} ,$$

Such that :

$$\mathbf{M}(\mathbf{d}_{n+1}, t_{n+1})\mathbf{v}_{n+1} + \mathbf{N}(\mathbf{d}_{n+1}, t_{n+1}) = \mathbf{F}(\mathbf{d}_{n+1}, t_{n+1}) , \tag{2.5.4}$$

$$\mathbf{d}_0 = \mathbf{d}^0 , \tag{2.5.5}$$

$$\mathbf{d}_{n+1} = \mathbf{d}_n + \Delta t\{(1-\alpha)\mathbf{v}_n + \alpha\mathbf{v}_{n+1}\} , \tag{2.5.6}$$

$$\alpha \in [0, 1] . \tag{2.5.7}$$

Remarks:

1. The parameter α controls the accuracy and stability of the time integration method.
2. If $\alpha = 0$ the method is termed *explicit*. When $\mathbf{M}$ is diagonal the equations uncouple and no matrix solution is required. However, there is a stringent stability condition which governs the size of Δt.
3. If $\alpha \neq 0$ the method is termed *implicit*, and a matrix solution is required.
4. If $\alpha \in [1/2, 1]$ the method is unconditionally stable and Δt may be chosen on the basis of accuracy considerations alone.
5. Solution of the nonlinear equation (2.5.4) requires the use of an iterative solution technique.

§2.6 Nonlinear Iterative Algorithms

In the preceding section we reduced the solution of a set of nonlinear ordinary differential equations to the solution of a single nonlinear matrix equation for each time step. We now consider iterative techniques for solving (2.5.4) .

2.6.1 Preliminaries

The iterative scheme proposed is a variant of Newton-Raphson iteration which makes use of the continuity of the temporal discretization; it is called the *predictor-corrector* method. The first phase of the algorithm uses the previously-computed results for the temperature and velocity at step n to *predict* what the temperature will be at step $n + 1$. The second phase then does successive *corrections* until convergence is achieved. The corrections require the use of the *linearized* operator to compute solution increments. A complete linearization of the nonlinear operators would result in a non-symmetric linear equation system which is undesirable from both a physical and a

computational point of view. Instead, we employ a ***symmetric linearization*** which uses values from the previous iterate in selected locations to produce a symmetric equation system. Since the problem is cast into a residual formulation, the converged solution of the symmetric linearization is identical to that obtained from the consistent operator.

2.6.2 Newton-Raphson Iteration

Eliminating $\mathbf{v}_{n+1}$ between (2.5.4) and (2.5.6) produces the nonlinear matrix equation

$$\mathbf{f}(\mathbf{d}_{n+1}) = \mathbf{o} \,, \tag{2.6.1}$$

where

$$\begin{aligned} \mathbf{f}(\mathbf{d}_{n+1}) = \mathbf{M}(\mathbf{d}_{n+1}, t_{n+1}) \frac{1}{\alpha} \left\{ \frac{1}{\Delta t}(\mathbf{d}_{n+1} - \mathbf{d}_n) - (1-\alpha)\mathbf{v}_n \right\} \\ + \mathbf{N}(\mathbf{d}_{n+1}, t_{n+1}) \\ + \mathbf{F}(\mathbf{d}_{n+1}, t_{n+1}) \,. \end{aligned} \tag{2.6.2}$$

Using a Taylor series expansion about the exact solution $\mathbf{d}_{n+1}$, we may approximate the function $\mathbf{f}$ at a point $\mathbf{d}_{n+1}^{(i)}$ by

$$\mathbf{f}(\mathbf{d}_{n+1}) = \mathbf{f}(\mathbf{d}_{n+1}^{(i)}) + \left. \frac{\partial \mathbf{f}(\mathbf{d})}{\partial \mathbf{d}} \right|_{\mathbf{d}=\mathbf{d}_{n+1}^{(i)}} (\mathbf{d}_{n+1} - \mathbf{d}_{n+1}^{(i)}) + \cdots . \tag{2.6.3}$$

Defining the solution increment as

$$\Delta \mathbf{d}_{n+1}^{(i)} = \mathbf{d}_{n+1} - \mathbf{d}_{n+1}^{(i)} \,, \tag{2.6.4}$$

and the tangent operator as

$$D\mathbf{f}(\mathbf{d}_{n+1}^{(i)}) = \left. \frac{\partial \mathbf{f}(\mathbf{d})}{\partial \mathbf{d}} \right|_{\mathbf{d}=\mathbf{d}_{n+1}^{(i)}} , \tag{2.6.5}$$

and ignoring the higher-order terms, we may write

$$D\mathbf{f}(\mathbf{d}_{n+1}^{(i)})\Delta\mathbf{d}_{n+1}^{(i)} = -\mathbf{f}(\mathbf{d}_{n+1}^{(i)}) . \tag{2.6.6}$$

The solution of (2.6.6) for $\Delta\mathbf{d}_{n+1}^{(i)}$ allows us to compute a better approximation to the exact solution ,

$$\mathbf{d}_{n+1}^{(i+1)} = \mathbf{d}_{n+1}^{(i)} + \Delta\mathbf{d}_{n+1}^{(i)} . \tag{2.6.7}$$

Examining the terms in the tangent operator in more detail,

$$D\mathbf{f}(\mathbf{d}) = D\{\mathbf{M}(\mathbf{d},t)\mathbf{v}\} + D\mathbf{N}(\mathbf{d},t) + D\mathbf{F}(\mathbf{d},t) . \tag{2.6.8}$$

The capacitive contribution is

$$D\{\mathbf{M}(\mathbf{d},t)\mathbf{v}\} = D\mathbf{M}(\mathbf{d},t)\mathbf{v} + \mathbf{M}(\mathbf{d},t)D\mathbf{v} , \tag{2.6.9}$$

where

$$D\mathbf{M}(\mathbf{d},t)\mathbf{v} = \mathop{\mathbf{A}}_{e=1}^{N_{EL}} D\mathbf{m}^e(\mathbf{d}^e,t)\mathbf{v}^e , \tag{2.6.10}$$

with element contributions

$$D\mathbf{m}^e(\mathbf{d}^e,t)\mathbf{v}^e = \left[\frac{\partial m_{ab}^e(\mathbf{d}^e,t)}{\partial d_c^e} v_b^e\right] , \tag{2.6.11}$$

and components

$$\frac{\partial m_{ab}^e(\mathbf{d}^e,t)}{\partial d_c^e} v_b^e = \int_{\Omega^e} N_a(\mathbf{x})\rho(\mathbf{x}) \frac{\partial C_p(u^h,\mathbf{x},t)}{\partial u^h} N_c(\mathbf{x}) N_b(\mathbf{x}) v_b^e \, d\Omega . \tag{2.6.12}$$

The consideration of $\mathbf{v}$ as a function of $\mathbf{d}$ gives

$$D\mathbf{v} = \frac{\mathbf{1}}{\alpha\Delta t} . \tag{2.6.13}$$

The contribution to the tangent operator from the convective flux vector **N** is

$$D\mathbf{N}(\mathbf{d},t) = \mathop{\mathbf{A}}_{e=1}^{N_{EL}} D\mathbf{n}^e(\mathbf{d}^e,t)\,, \tag{2.6.14}$$

with element contributions

$$D\mathbf{n}^e(\mathbf{d}^e,t) = \left[\frac{\partial n_a^e(\mathbf{d}^e,t)}{\partial d_b^e}\right], \tag{2.6.15}$$

and components

$$\frac{\partial n_a^e(\mathbf{d}^e,t)}{\partial d_b^e} = \int_{\Omega^e} \mathbf{B}_a^T(\mathbf{x})\mathbf{K}(u^h,\mathbf{x},t)\mathbf{B}_b(\mathbf{x})\,d\Omega + \int_{\Omega^e} \mathbf{B}_a^T(\mathbf{x})\frac{\partial \mathbf{K}(u^h,\mathbf{x},t)}{\partial u^h}N_b(\mathbf{x})u^h\,d\Omega\,. \tag{2.6.16}$$

Note that the second term in (2.6.16) is nonsymmetric.

Finally, the contribution to the tangent operator by **F**, the internal heat flux and h-type boundary conditions vector, is

$$D\mathbf{F}(\mathbf{d},t) = \mathop{\mathbf{A}}_{e=1}^{N_{EL}} D\mathbf{f}^e(\mathbf{d}^e,t)\,, \tag{2.6.17}$$

with element contributions

$$D\mathbf{f}^e(\mathbf{d}^e,t) = \left[\frac{\partial f_a^e(\mathbf{d}^e,t)}{\partial d_b^e}\right], \tag{2.6.18}$$

and components

$$\frac{\partial f_a^e(\mathbf{d}^e,t)}{\partial d_b^e} = \int_{\Omega^e} N_a(\mathbf{x})\frac{\partial Q(u^h,\mathbf{x},t)}{\partial u^h}N_b(\mathbf{x})\,d\Omega + \int_{\Gamma^e \bigcap \Gamma_h} N_a(\mathbf{x})\frac{\partial h(u^h,\mathbf{x},t)}{\partial u^h}N_b(\mathbf{x})\,d\Gamma\,. \tag{2.6.19}$$

2.6.3 Symmetric Linearization

A symmetric linearization approximating the exact tangent operator is now defined. We will consider approximations to each of the three terms of (2.6.8) .

First, we approximate the capacity contribution by

$$D\{\mathbf{M}(\mathbf{d},t)\mathbf{v}\} \approx \frac{1}{\alpha\Delta t}\mathbf{M}(\mathbf{d},t)\,. \tag{2.6.20}$$

This approximation avoids the additional computational expense engendered by the full $D\mathbf{M}$ calculation of (2.6.12) .

The second term, the conductive contribution, is approximated by

$$D\mathbf{N}(\mathbf{d},t) \approx \mathop{\mathbf{A}}_{e=1}^{N_{EL}} \mathbf{k}_k^e(\mathbf{d}^e,t)\,, \tag{2.6.21}$$

with element contributions

$$\mathbf{k}_k^e(\mathbf{d}^e,t) = \left[k_{(k)ab}^e(\mathbf{d}^e,t)\right], \tag{2.6.22}$$

and components

$$k_{(k)ab}^e(\mathbf{d}^e,t) = \int_{\Omega^e} \mathbf{B}_a^T(\mathbf{x})\mathbf{K}(u^h,\mathbf{x},t)\mathbf{B}_b(\mathbf{x})\,d\Omega\,. \tag{2.6.23}$$

We only keep the symmetric term of equation (2.6.16) .

Finally, to approximate the third term we make two additional assumptions. The first is that we can ignore the dependence of the internal heat generation Q on temperature. Thus, in computing the symmetric tangent operator we assume

$$\frac{\partial Q(u^h,\mathbf{x},t)}{\partial u^h} = 0\,. \tag{2.6.24}$$

The second assumption is that the contributions to the symmetric tangent operator from the h-type boundary conditions come only from one of the four types (2.1.9)—(2.1.12). The contribution of the convective and radiation boundary conditions is

$$D\mathbf{F}(\mathbf{d},t) \approx \mathop{\mathbf{A}}_{e=1}^{N_{EL}} \left(\mathbf{k}_c^e(\mathbf{d}^e,t) + \mathbf{k}_r^e(\mathbf{d}^e,t)\right), \tag{2.6.25}$$

where the convective element *stiffness*[1] is

$$\mathbf{k}_c^e(\mathbf{d}^e,t) = \left[k_{(c)ab}^e(\mathbf{d}^e,t)\right], \tag{2.6.26}$$

with components

$$k_{(c)ab}^e(\mathbf{d}^e,t) = \int_{\Gamma^e \bigcap \Gamma_h} N_a(\mathbf{x}) h_c(\mathbf{x},t) N_b(\mathbf{x})\, d\Gamma\,; \tag{2.6.27}$$

and the radiation element stiffness is

$$\mathbf{k}_r^e(\mathbf{d}^e,t) = \left[k_{(r)ab}^e(\mathbf{d}^e,t)\right], \tag{2.6.28}$$

with components

$$k_{(r)ab}^e(\mathbf{d}^e,t) = \int_{\Gamma^e \bigcap \Gamma_h} N_a(\mathbf{x}) \kappa(u^h,\mathbf{x},t) N_b(\mathbf{x})\, d\Gamma\,. \tag{2.6.29}$$

We can now define the tangent stiffness as

$$\mathbf{K}_T(\mathbf{d},t) = \mathop{\mathbf{A}}_{e=1}^{N_{EL}} \mathbf{k}_T^e(\mathbf{d}^e,t)\,, \tag{2.6.30}$$

[1] We will use the terms stiffness, mass, and force, instead of conductivity, capacity, and flux, respectively, when we wish to infer finite element connotations.

where

$$\mathbf{k}_T^e(\mathbf{d}^e, t) = \mathbf{k}_k^e(\mathbf{d}^e, t) + \mathbf{k}_c^e(\mathbf{d}^e, t) + \mathbf{k}_r^e(\mathbf{d}^e, t) \,. \tag{2.6.31}$$

This leads to the following definition of $\mathbf{S}$, the symmetric linearization of the tangent operator,

$$\mathbf{S}(\mathbf{d}, t) = \frac{1}{\alpha \Delta t}\mathbf{M}(\mathbf{d}, t) + \mathbf{K}_T(\mathbf{d}, t) \,, \tag{2.6.32}$$

where

$$D\mathbf{f}(\mathbf{d}) \approx \mathbf{S}(\mathbf{d}, t) \,. \tag{2.6.33}$$

2.6.4 Predictor-Corrector Algorithm

The predictor-corrector iterative algorithm to compute the solution for step $n+1$ may be defined as :

Given : $\mathbf{d}_n$, $\mathbf{v}_n$,

Compute the predicted values :

$$i = 0 \,, \tag{2.6.34}$$

$$\mathbf{d}_{n+1}^{(i)} = \tilde{\mathbf{d}}_{n+1} = \mathbf{d}_n + \Delta t(1-\alpha)\mathbf{v}_n \,, \tag{2.6.35}$$

$$\mathbf{v}_{n+1}^{(i)} = \tilde{\mathbf{v}}_{n+1} = \mathbf{o} \,, \tag{2.6.36}$$

(I) then iterate over :

$$\mathbf{M}^{*(i)}_{n+1}\Delta\mathbf{v}_{n+1}^{(i)} = \Delta\mathbf{F}_{n+1}^{(i)} \,, \tag{2.6.37}$$

$$\mathbf{v}_{n+1}^{(i+1)} = \mathbf{v}_{n+1}^{(i)} + \Delta\mathbf{v}_{n+1}^{(i)} \,, \tag{2.6.38}$$

$$\mathbf{d}_{n+1}^{(i+1)} = \tilde{\mathbf{d}}_{n+1} + \alpha\Delta t\mathbf{v}_{n+1}^{(i+1)} \,, \tag{2.6.39}$$

$$i = i + 1 \,, \tag{2.6.40}$$

until *convergence* is obtained.

The *effective mass*[2], $\mathbf{M}^*$, is defined as

$$\mathbf{M}^{*(i)}_{n+1} = \mathbf{M}(\mathbf{d}^{(i)}_{n+1}, t_{n+1}) + \alpha \Delta t \mathbf{K}_T(\mathbf{d}^{(i)}_{n+1}, t_{n+1}) \,, \tag{2.6.41}$$

and the *out-of-balance force*[3] or *residual force* is

$$\Delta \mathbf{F}^{(i)}_{n+1} = \mathbf{F}(\mathbf{d}^{(i)}_{n+1}, t_{n+1}) - \mathbf{M}(\mathbf{d}^{(i)}_{n+1}, t_{n+1})\mathbf{v}^{(i)}_{n+1} - \mathbf{N}(\mathbf{d}^{(i)}_{n+1}, t_{n+1}) \,. \tag{2.6.42}$$

In the above equations, $\mathbf{M}$, $\mathbf{N}$, and $\mathbf{F}$ are the nonlinear matrix and vector functions defined in (2.4.8)—(2.4.16) , and $\mathbf{K}_T$ is the tangent stiffness matrix resulting from the symmetric linearization.

Remarks:

1. For large problems, the cost per iteration in the above scheme is almost solely governed by the cost of formation and factorization of the linearized operator $\mathbf{M}^*$.
2. If the nonlinearities are "small", it may be computationally expedient to form and factor $\mathbf{M}^*$ only once, at the beginning of the time integration.
3. If the nonlinearities vary slowly, a *modified Newton-Raphson* scheme in which $\mathbf{M}^*$ is formed and factored every few iterations (or steps) may be appropriate.
4. The definition of the predictor-corrector scheme (**I**) uses a *velocity formulation.* It may be equivalently defined in terms of temperature increments.

§2.7 Result : The Linear Problem ($\mathbf{Ax} = \mathbf{b}$)

In the preceding sections we have derived equations representing both the spatial and temporal discretization of the original nonlinear partial differential equation for

[2]See footnote on page 23.

[3]See footnote on page 23.

heat conduction. Then, an iterative algorithm was presented for the solution of the resulting nonlinear matrix equations. The algorithm requires the repeated solution of a linear equation system which has the "generic" representation

$$\boxed{\mathbf{Ax} = \mathbf{b}} \tag{2.7.1}$$

It is important to note that the matrix $\mathbf{A}$ is both symmetric and positive-definite. These properties contribute greatly to the ease with which a solution to (2.7.1) may be obtained. In the next chapter we discuss ways of obtaining the solution to (2.7.1) by using both direct and iterative techniques.

Chapter 3

Algorithms for Solving the Linear Problem

This chapter explores the techniques available for the solution of the linear equation system which resulted from the application of the finite element method to nonlinear transient heat conduction problems. The first class of solution techniques considered are *direct techniques*. These techniques typically use variations of *Gaussian elimination* to compute an exact solution. The second class of solution techniques to be studied are *iterative techniques*. These techniques produce a sequence of approximate solutions which, under the right conditions, converge to the exact solution. To fully appreciate the potential cost savings of iterative techniques, it is necessary to understand the expense incurred by direct solution techniques.

§3.1 Direct Techniques

The most widely used forms of direct solution techniques are variations of Gaussian elimination [Fellipa 75], [Jennings 77], [Kamel 78], [George 81], [Winget 82]. They all possess the desirable attribute of generating the exact solution within the bounds of machine accuracy. This deterministic quality, or robustness, is one of the chief reasons for their popularity.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & 0 & a_{14} & 0 & 0 & 0 & 0 & 0 \\ & a_{22} & a_{23} & a_{24} & 0 & a_{37} & 0 & 0 & 0 \\ & & a_{33} & a_{24} & 0 & a_{36} & 0 & 0 & 0 \\ & & & a_{44} & a_{45} & a_{46} & 0 & 0 & 0 \\ & & & & a_{55} & a_{56} & 0 & a_{58} & 0 \\ & & \text{Symmetric} & & & a_{66} & a_{67} & a_{68} & 0 \\ & & & & & & a_{77} & a_{78} & a_{79} \\ & & & & & & & a_{88} & a_{89} \\ & & & & & & & & a_{99} \end{bmatrix}$$

Figure 3.1.1 Example of skyline storage for a 9×9 matrix.

3.1.1 Form of the Matrix Equations

The matrix generated by the finite element procedure of the last chapter is typically sparse, because the equations are only locally coupled.[1] Careful ordering of the nodes, or optimal reordering of the equations, transforms the matrix to a ***variable bandwidth*** form. To store the matrix efficiently, a ***profile*** or ***skyline storage*** scheme is commonly used, see Figure 3.1.1.

The important parameters representing the matrix are N_{EQ}, the number of equations; $\bar{b}$, the *mean half-bandwidth*; and $\hat{b}$, the *computational half-bandwidth*. The bandwidth parameters are important for cost computations, and are defined as

$$\bar{b} = \frac{\sum_{i=1}^{N_{EQ}} b_i}{N_{EQ}}, \tag{3.1.1}$$

[1]There are only non-zero terms connecting equations with nodes in the same element.

$$\hat{b} = \left[\frac{\sum_{i=1}^{N_{EQ}} b_i^2}{N_{EQ}} \right]^{\frac{1}{2}}, \tag{3.1.2}$$

where b_i is the height of the i^{th} column.

3.1.2 Gaussian Elimination

We will consider forms of Gaussian elimination known as *triangular decomposition* techniques. The underlying idea is to factor the matrix **A** into a product of simpler terms. There are two important constraints the decomposition must satisfy. First, the simpler terms must be easier to "invert" than the original matrix. Second, the method must be *compact*, that is the decomposition should need no more storage then the original matrix. Thus, we do not permit *fill-in* outside of the original profile.

We define the *product decomposition* as

$$\mathbf{A} = \mathbf{L}_\pi(\mathbf{A})\mathbf{D}_\pi(\mathbf{A})\mathbf{U}_\pi(\mathbf{A}), \tag{3.1.3}$$

where the subscript π denotes *product*. The matrix $\mathbf{L}_\pi$ is a lower-triangular matrix with unit diagonal and has the form

$$\mathbf{L}_\pi = \begin{bmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & 1 \end{bmatrix}. \tag{3.1.4}$$

$\mathbf{D}_\pi$ is a diagonal matrix,

$$\mathbf{D}_\pi = \begin{bmatrix} d_{11} & 0 & \dots & 0 \\ 0 & d_{22} & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & d_{nn} \end{bmatrix}, \tag{3.1.5}$$

and $\mathbf{U}_\pi$ is a upper-triangular matrix with unit diagonal,

$$\mathbf{U}_\pi = \begin{bmatrix} 1 & u_{12} & \dots & u_{1n} \\ 0 & 1 & \dots & u_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}. \tag{3.1.6}$$

Equation (3.1.3) represents the *Crout factorization.*

If $\mathbf{A}$ is symmetric, then $\mathbf{L}_\pi(\mathbf{A}) = \mathbf{U}_\pi^T(\mathbf{A})$. If the entries of $\mathbf{D}_\pi(\mathbf{A})$ are nonnegative, then

$$\mathbf{A} = \tilde{\mathbf{L}}_\pi(\mathbf{A})\tilde{\mathbf{U}}_\pi(\mathbf{A}), \tag{3.1.7}$$

where

$$\tilde{\mathbf{L}}_\pi(\mathbf{A}) = \mathbf{L}_\pi(\mathbf{A})\mathbf{D}_\pi^{1/2}(\mathbf{A}), \tag{3.1.8}$$

and

$$\tilde{\mathbf{U}}_\pi(\mathbf{A}) = \mathbf{D}_\pi^{1/2}(\mathbf{A})\mathbf{U}_\pi(\mathbf{A}). \tag{3.1.9}$$

If the matrix $\mathbf{A}$ is symmetric positive-definite, then it also admits the *Cholesky decomposition*:

$$\mathbf{A} = \tilde{\mathbf{L}}_\pi(\mathbf{A})\tilde{\mathbf{L}}_\pi^T(\mathbf{A}), \tag{3.1.10}$$

where $\tilde{\mathbf{L}}_\pi$ is a lower-triangular matrix defined by (3.1.8) which has the form

$$\tilde{\mathbf{L}}_\pi = \begin{bmatrix} \tilde{l}_{11} & 0 & \dots & 0 \\ \tilde{l}_{21} & \tilde{l}_{22} & \dots & 0 \\ \vdots & & \ddots & \vdots \\ \tilde{l}_{n1} & \tilde{l}_{n2} & \dots & \tilde{l}_{nn} \end{bmatrix}. \tag{3.1.11}$$

A solution, $\mathbf{x}$, of the factorized matrix equations is then obtained by the following three steps :

$$\mathbf{L}_\pi(\mathbf{A})\mathbf{z} = \mathbf{b} \quad (\text{ forward reduction }), \tag{3.1.12}$$

$$\mathbf{D}_\pi(\mathbf{A})\mathbf{y} = \mathbf{z} \quad (\text{ scaling }), \tag{3.1.13}$$

$$\mathbf{U}_\pi(\mathbf{A})\mathbf{x} = \mathbf{y} \quad (\text{ backsubstitution }). \tag{3.1.14}$$

For either of the two decompositions it is a two-step process to obtain a solution. First, it is necessary to *factor* the matrix $\mathbf{A}$ into its product terms. Then, we may *solve* for $\mathbf{x}$ by a series of reductions of the right-hand-side vector, $\mathbf{b}$.

3.1.3 Cost

There are several important considerations when evaluating the *cost* of a solution technique. First, one must consider the amount of *storage* required by the technique. We will define the storage unit *word* as the amount of storage required by one *floating-point* number. The amount of storage required by an algorithm is composed of the *primary storage*, to hold the terms in the matrix, and the *overhead storage*, to hold the indexing or addressing information.

A second and perhaps more often-considered cost, is the *computational cost* or cost of *CPU* . We will restrict ourselves to CPU cost estimates for arithmetic operations. An *operation* is defined as one floating-point multiplication plus one addition.

The storage cost associated with the factors of the matrix $\mathbf{A}$ is

$$S_{\pi-factor}(\mathbf{A}) = (\bar{b} + 1)N_{EQ} \text{ words} . \tag{3.1.15}$$

The $1 \times N_{EQ}$ in (3.1.15) is associated with the overhead cost of indexing the terms in $\mathbf{A}$. The CPU cost to factor $\mathbf{A}$ into its Crout decomposition is

$$C_{\pi-factor}(\mathbf{A}) = (\frac{1}{2}\hat{b}^2 N_{EQ} + \cdots) \text{ ops} . \tag{3.1.16}$$

Similarly, the CPU cost to solve the factored equations is

$$C_{\pi-solve}(\mathbf{A}) = (2\bar{b}N_{EQ} + \cdots) \text{ ops} . \tag{3.1.17}$$

We consider only the highest order terms in $C_{\pi-factor}(\mathbf{A})$ and $C_{\pi-solve}(\mathbf{A})$, since they dominate the cost for large problems.

For an example of the cost of storage and CPU, consider the three-dimensional model problem depicted in Figure 3.1.2, which is assumed meshed with linear eight-node brick elements. The total number of equations is $N_{EQ} = pqr$. For ease in the following discussion we will assume $p \geq q \geq r$. Thus, using a standard, near optimal, node ordering scheme, the bandwidth parameters are

$$\hat{b} \approx \bar{b} \approx qr . \tag{3.1.18}$$

The total storage required to hold the factored matrix is

$$S_{\pi-factor}(\mathbf{A}) = pq^2r^2 + pqr \text{ words} , \tag{3.1.19}$$

and the factorization and solution costs are

$$C_{\pi-factor}(\mathbf{A}) = \frac{1}{2}pq^3r^3 \text{ ops} , \tag{3.1.20}$$

$$C_{\pi-solve}(\mathbf{A}) = 2pq^2r^2 \text{ ops} . \tag{3.1.21}$$

If we consider equal refinement in all three directions, then

$$p = q = r , \tag{3.1.22}$$

$$N_{EQ} = p^3 , \tag{3.1.23}$$

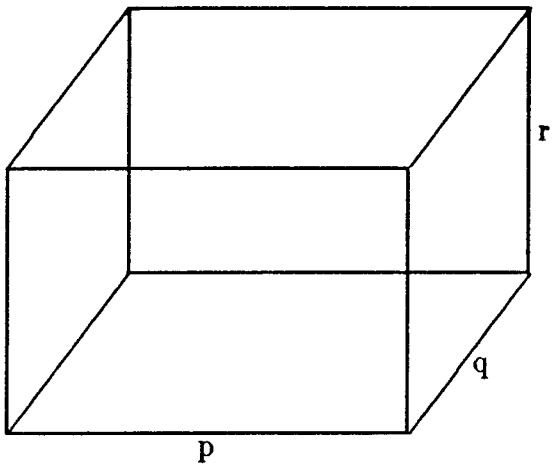


Figure 3.1.2 Three-dimensional model problem with $p \times q \times r$ regular mesh.

$$S_{\pi-factor}(\mathbf{A}) = p^5 + p^3 \text{ words}, \tag{3.1.24}$$

$$C_{\pi-factor}(\mathbf{A}) = \frac{1}{2}p^7 \text{ ops}, \tag{3.1.25}$$

and

$$C_{\pi-solve}(\mathbf{A}) = 2p^5 \text{ ops}. \tag{3.1.26}$$

It is important to distinguish between the factorization and solution costs for small to medium problems since a modified Newton-Raphson nonlinear solution algorithm may use the same factorization for many iterations. However, as the problem becomes large, the cost of one factorization exceeds the cost of all the solutions combined.

§3.2 Iterative Techniques

A second class of methods for the solution to the linear problem are *iterative techniques*[2] [Householder 64], [Bakhvalov 77], [Mitchell 80], [Gill 81], [Hageman 81], [Vemuri 81]. Iterative techniques have not been widely used in finite element analysis

[2]We will use the term iterative although in practice the techniques will be applied at a sub-iteration or inner-iteration level.

due to their indeterminate nature. Unless a great deal is known about the system to be solved, there is no way to predict *a priori* the number of iterations required to produce a solution of the desired accuracy.

3.2.1 Basic Definitions

We first consider *linear stationary methods of the first degree.* These methods have the classical form :

Given : $\mathbf{x}^{(0)}$,

Iterate over :

$$\mathbf{x}^{(k+1)} = \mathbf{G}\mathbf{x}^{(k)} + \mathbf{k}\,, \tag{3.2.1}$$

$$k = k + 1\,, \tag{3.2.2}$$

until convergence is achieved.

Where

$$\mathbf{G} = \mathbf{I} - \mathbf{B}^{-1}\mathbf{A}\,, \tag{3.2.3}$$

$$\mathbf{k} = \mathbf{B}^{-1}\mathbf{b}\,. \tag{3.2.4}$$

They may also be expressed in an equivalent *residual form* :

Given : $\mathbf{x}^{(0)}$,

Iterate over :

$$\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}\,, \tag{3.2.5}$$

$$\mathbf{B}\Delta\mathbf{x}^{(k)} = \mathbf{r}^{(k)}\,, \tag{3.2.6}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)}\,, \tag{3.2.7}$$

$$k = k + 1\,, \tag{3.2.8}$$

until convergence is achieved.

The method is of first degree since $\mathbf{x}^{(k+1)}$ depends only on $\mathbf{x}^{(k)}$. It is stationary since $\mathbf{G}$ and $\mathbf{k}$ are independent of k and linear since $\mathbf{G}$ and $\mathbf{k}$ are independent of $\mathbf{x}^{(k)}$. In classical iterative methodology, $\mathbf{B}$ is referred to as the *splitting matrix*. Note that if $\mathbf{B} = \mathbf{A}$, the method converges to the exact solution in one iteration. We will consider only methods that are *symmetrizable*. A method is symmetrizable if

$$\exists \ \mathbf{P}, \ \det(\mathbf{P}) \neq 0 \quad \text{such that}$$
$$\mathbf{P}(\mathbf{I} - \mathbf{G})\mathbf{P}^{-1} = \mathbf{P}(\mathbf{B}^{-1}\mathbf{A})\mathbf{P}^{-1} \in \mathrm{S\overset{+}{y}m} \,. \tag{3.2.9}$$

The symmetrizability is guaranteed if $\mathbf{A}$ and $\mathbf{B}$ are both symmetric and positive-definite. This is easily shown by choosing $\mathbf{P} = \mathbf{A}^{1/2}$ or $\mathbf{P} = \mathbf{B}^{1/2}$. The convergence rate of an iterative scheme usually depends on the *spectral condition number*, κ, which is defined as

$$\kappa(\mathbf{A}) = \|\mathbf{A}\|_2 \|\mathbf{A}^{-1}\|_2 \,. \tag{3.2.10}$$

For symmetric positive-definite matrices, κ may be expressed in terms of the eigenvalues of the matrix as

$$\kappa(\mathbf{A}) = \frac{\mathrm{M}(\mathbf{A})}{\mathrm{m}(\mathbf{A})} \,, \tag{3.2.11}$$

where $\mathrm{m}(\mathbf{A})$ is the smallest eigenvalue of $\mathbf{A}$, and $\mathrm{M}(\mathbf{A})$ is the largest.

3.2.2 Standard Iterative Techniques

As with direct techniques, we find it convenient to define some standard matrix decompositions. For iterative techniques, these decompositions are based on the *sum decomposition*, which is defined as

$$\mathbf{A} = \mathbf{L}_\sigma(\mathbf{A}) + \mathbf{D}_\sigma(\mathbf{A}) + \mathbf{U}_\sigma(\mathbf{A}) \,, \tag{3.2.12}$$

where the subscript σ denotes *sum.*

The matrix $\mathbf{L}_\sigma$ is a lower-triangular matrix with zero diagonal and thus has the form

$$\mathbf{L}_\sigma = \begin{bmatrix} 0 & 0 & \dots & 0 \\ l_{21} & 0 & \dots & 0 \\ \cdot & & \cdot & \cdot \\ \cdot & & & \cdot \\ l_{n1} & l_{n2} & \dots & 0 \end{bmatrix}. \tag{3.2.13}$$

$\mathbf{D}_\sigma$ is a diagonal matrix,

$$\mathbf{D}_\sigma = \begin{bmatrix} d_{11} & 0 & \dots & 0 \\ 0 & d_{22} & \dots & 0 \\ \cdot & & \cdot & \cdot \\ \cdot & & & \cdot \\ 0 & 0 & \dots & d_{nn} \end{bmatrix}, \tag{3.2.14}$$

and $\mathbf{U}_\sigma$ is a upper-triangular matrix with zero diagonal,

$$\mathbf{U}_\sigma = \begin{bmatrix} 0 & u_{12} & \dots & u_{1n} \\ 0 & 0 & \dots & u_{2n} \\ \cdot & & \cdot & \cdot \\ \cdot & & & \cdot \\ 0 & 0 & \dots & 0 \end{bmatrix}. \tag{3.2.15}$$

In analogy with the product decomposition, we may write

$$\mathbf{A} = \tilde{\mathbf{L}}_\sigma(\mathbf{A}) + \tilde{\mathbf{U}}_\sigma(\mathbf{A}), \tag{3.2.16}$$

where

$$\tilde{\mathbf{L}}_\sigma(\mathbf{A}) = \mathbf{L}_\sigma(\mathbf{A}) + \frac{1}{2}\mathbf{D}_\sigma(\mathbf{A}), \tag{3.2.17}$$

$$\tilde{\mathbf{U}}_\sigma(\mathbf{A}) = \mathbf{U}_\sigma(\mathbf{A}) + \frac{1}{2}\mathbf{D}_\sigma(\mathbf{A}) . \tag{3.2.18}$$

Further, if $\mathbf{A}$ has non-zero diagonals, then $\mathbf{D}_\sigma^{-1}(\mathbf{A})$ exists, and we may define the scaled matrices

$$\hat{\mathbf{L}}_\sigma(\mathbf{A}) = \mathbf{D}_\sigma^{-1}(\mathbf{A})\mathbf{L}_\sigma(\mathbf{A}) \tag{3.2.19}$$

and

$$\hat{\mathbf{U}}_\sigma(\mathbf{A}) = \mathbf{D}_\sigma^{-1}(\mathbf{A})\mathbf{U}_\sigma(\mathbf{A}) . \tag{3.2.20}$$

If $\mathbf{A}$ is symmetric, $\mathbf{L}_\sigma(\mathbf{A}) = \mathbf{U}_\sigma^T(\mathbf{A})$, $\tilde{\mathbf{L}}_\sigma(\mathbf{A}) = \tilde{\mathbf{U}}_\sigma^T(\mathbf{A})$ and $\hat{\mathbf{L}}_\sigma(\mathbf{A}) = \hat{\mathbf{U}}_\sigma^T(\mathbf{A})$.

The decomposition (3.2.16)—(3.2.18) has figured in transient analysis algorithms developed by Trujillo [Trujillo 77a,77b,77c] and subsequently discussed by Park [Park 82].

3.2.2.1 Richardson's Method

The simplest iterative scheme is a variation of the method of *Richardson* which is defined by :

$$\textbf{(Richardson)}\begin{cases} \mathbf{G} = \mathbf{I} - \mathbf{A} , & (3.2.21) \\ \mathbf{k} = \mathbf{b} , & (3.2.22) \\ \mathbf{B} = \mathbf{I} . & (3.2.23) \end{cases}$$

This scheme is only convergent if $\mathbf{A}$ satisfies the very stringent spectral condition $\mathrm{M}(\mathbf{A}) < 2$.

3.2.2.2 Jacobi Iteration

The next simplest iterative scheme is *Jacobi iteration*, or the method of simultaneous corrections, which is defined by :

$$\text{(Jacobi)}\begin{cases} \mathbf{G} = \mathbf{I} - \mathbf{D}_\sigma^{-1}(\mathbf{A})\mathbf{A}\,, & (3.2.24) \\ \mathbf{k} = \mathbf{D}_\sigma^{-1}(\mathbf{A})\mathbf{b}\,, & (3.2.25) \\ \mathbf{B} = \mathbf{D}_\sigma(\mathbf{A})\,. & (3.2.26) \end{cases}$$

3.2.2.3 Gauss-Seidel Iteration

The *Gauss-Seidel iteration* technique, or method of successive corrections, is defined by :

$$\text{(Gauss-Seidel)}\begin{cases} \mathbf{G} = -(\mathbf{I} - \hat{\mathbf{L}}_\sigma(\mathbf{A}))^{-1}\hat{\mathbf{U}}_\sigma(\mathbf{A})\,, & (3.2.27) \\ \mathbf{k} = (\mathbf{I} + \hat{\mathbf{L}}_\sigma(\mathbf{A}))^{-1}\mathbf{D}_\sigma^{-1}(\mathbf{A})\mathbf{b}\,, & (3.2.28) \\ \mathbf{B} = \mathbf{L}_\sigma(\mathbf{A}) + \mathbf{D}_\sigma(\mathbf{A})\,. & (3.2.29) \end{cases}$$

3.2.2.4 Successive Overrelaxation

The *successive overrelaxation method* (SOR) is defined by :

$$\text{(SOR)}\begin{cases} \mathbf{G} = (\mathbf{I} - \omega\hat{\mathbf{L}}_\sigma(\mathbf{A}))^{-1}((1-\omega)\mathbf{I} - \omega\hat{\mathbf{U}}_\sigma(\mathbf{A}))\,, & (3.2.30) \\ \mathbf{k} = \omega(\mathbf{I} + \omega\hat{\mathbf{L}}_\sigma(\mathbf{A}))^{-1}\mathbf{D}_\sigma^{-1}(\mathbf{A})\mathbf{b}\,, & (3.2.31) \\ \mathbf{B} = \mathbf{L}_\sigma(\mathbf{A}) + \frac{1}{\omega}\mathbf{D}_\sigma(\mathbf{A})\,. & (3.2.32) \end{cases}$$

The relaxation parameter, ω, is a real number chosen to optimize convergence. If $\omega \in]0,2[$ and $\mathbf{A}$ is symmetric positive-definite then the method always converges. The optimal convergence rate is usually given by $1 < \omega < 2$. Note that for $\omega = 1$, the method reduces to Gauss-Seidel iteration.

3.2.2.5 Symmetric Successive Overrelaxation

The successive overrelaxation method is not symmetrizable. However, making a second sweep and reversing the roles of $\mathbf{L}_\sigma$ and $\mathbf{U}_\sigma$ results in the *symmetric successive overrelaxation method* (SSOR) which is symmetrizable.

$$
(\text{SSOR})\begin{cases}
\mathbf{G} = (\mathbf{I} - \omega\hat{\mathbf{U}}_\sigma(\mathbf{A}))^{-1}((1-\omega)\mathbf{I} - \omega\hat{\mathbf{L}}_\sigma(\mathbf{A})) \\
\qquad \times (\mathbf{I} - \omega\hat{\mathbf{L}}_\sigma(\mathbf{A}))^{-1}((1-\omega)\mathbf{I} - \omega\hat{\mathbf{U}}_\sigma(\mathbf{A}))\,, & (3.2.33) \\
\mathbf{k} = \omega(2-\omega)(\mathbf{I} + \omega\hat{\mathbf{U}}_\sigma(\mathbf{A}))^{-1}(\mathbf{I} + \omega\hat{\mathbf{L}}_\sigma(\mathbf{A}))^{-1}\mathbf{D}_\sigma^{-1}(\mathbf{A})\mathbf{b}\,, & (3.2.34) \\
\mathbf{B} = \dfrac{\omega}{2-\omega}(\mathbf{L}_\sigma(\mathbf{A}) + \dfrac{1}{\omega}\mathbf{D}_\sigma(\mathbf{A})) \\
\qquad \times \mathbf{D}_\sigma(\mathbf{A})^{-1}(\dfrac{1}{\omega}\mathbf{D}_\sigma(\mathbf{A})) + \mathbf{U}_\sigma(\mathbf{A}))\,. & (3.2.35)
\end{cases}
$$

As in SOR, the relaxation parameter ω is a real number chosen to optimize convergence.

3.2.2.6 Approximate Factorization

In general, the splitting matrix $\mathbf{B}$ is any product of terms, each of which is easily invertible. A measure of the "quality" of the approximation is the condition number of $\mathbf{B}^{-1}\mathbf{A}$. The closer $\kappa\left(\mathbf{B}^{-1}\mathbf{A}\right)$ is to 1 the faster the convergence. The $\mathbf{B}$'s used in the standard iterative schemes have used sum decompositions to avoid the high cost of factorization. In the next chapter we explore approximate factorizations suitable for finite element computation based on both product and sum decompositions.

3.2.3 Parabolic Regularization

The *parabolic regularization* method changes solving the linear equation system into finding the steady-state solution of an associated pseudo-time-dependent parabolic problem [Hughes 83a]. The discrete solutions of the pseudo-time-dependent problem play the role of iterates. In particular, we look for the steady state solution of

$$\mathbf{W}\dot{\mathbf{x}} + \mathbf{A}\mathbf{x} = \mathbf{b}\,, \tag{3.2.36}$$

where $\mathbf{W}$ is a given, positive-definite, diagonal, pseudo-mass matrix and the superposed dot denotes differentiation with respect to τ, pseudo-time. If $\mathbf{A}$ is positive-definite, then given any initial condition $\mathbf{x}_0$, the solution converges to a steady state value, $\mathbf{A}^{-1}\mathbf{b}$. To obtain this solution, we apply a backward difference time integration scheme to (3.2.36), resulting in the following iteration algorithm :

$$(\mathrm{PR})\begin{cases} \mathbf{G} = (\mathbf{W} + \Delta\tau\mathbf{A})^{-1}\mathbf{W}\,, & (3.2.37)\\ \mathbf{k} = \Delta\tau(\mathbf{W} + \Delta\tau\mathbf{A})^{-1}\mathbf{b}\,, & (3.2.38)\\ \mathbf{B} = \dfrac{1}{\Delta\tau}(\mathbf{W} + \Delta\tau\mathbf{A})\,. & (3.2.39) \end{cases}$$

Here $\Delta\tau$ is a positive real number which governs the convergence rate of the iterative algorithm. In practice, we have found the following choices for $\mathbf{W}$ and $\Delta\tau$ to be acceptable when coupled with an approximate factorization :

$$\mathbf{W} = \mathbf{D}_\sigma(\mathbf{A})\,, \tag{3.2.40}$$

and

$$\Delta\tau = 1\,. \tag{3.2.41}$$

3.2.4 Variational Techniques

We now consider two variational techniques based on *gradient methods*. These techniques cannot be expressed in classical form since they are neither linear nor stationary. Both methods convert the problem of solving the linear equation system into the equivalent problem of minimizing a function of N_{EQ} variables.

3.2.4.1 Steepest Descent Technique

The method of *steepest descent* (SD) is motivated as follows. Consider the quadratic form

$$F(\mathbf{x}^{(k)}) = \frac{1}{2}(\mathbf{x}^{(k)}, \mathbf{A}\mathbf{x}^{(k)}) - (\mathbf{b}, \mathbf{x}^{(k)}) . \tag{3.2.42}$$

Solving $\mathbf{A}\mathbf{x} = \mathbf{b}$ is equivalent to minimizing $F(\mathbf{x}^{(k)})$ with respect to $\mathbf{x}^{(k)}$. The gradient of $F(\mathbf{x}^{(k)})$ is

$$\begin{aligned} \nabla F(\mathbf{x}^{(k)}) &= \mathbf{A}\mathbf{x}^{(k)} - \mathbf{b} \\ &= -\mathbf{r}^{(k)} . \end{aligned} \tag{3.2.43}$$

Thus, if the mimumum of F occurs at $\mathbf{x}^{(k)}$,

$$\nabla F(\mathbf{x}^{(k)}) = 0 = \mathbf{A}\mathbf{x}^{(k)} - \mathbf{b} , \tag{3.2.44}$$

and

$$\nabla(\nabla F(\mathbf{x}^{(k)})) = \mathbf{A} > 0 . \tag{3.2.45}$$

If, however, F is not mimimal at $\mathbf{x}^{(k)}$, then the gradient of $F(\mathbf{x}^{(k)})$ gives the direction which maximizes the change in F near the point $\mathbf{x}^{(k)}$. Constraining ourselves to move only along the path of steepest descent defined by the one-dimensional subspace $\mathbf{x}^{(k)} +$

$s^{(k)}\mathbf{r}^{(k)}$, $s^{(k)} \in \Re$, we minimize $F(\mathbf{x}^{(k)} + s^{(k)}\mathbf{r}^{(k)})$ with respect to $s^{(k)}$. The variable $s^{(k)}$ is the *search parameter*. Successive minimizations lead to the iterative algorithm :

$$
\text{(SD)}\left\{
\begin{array}{l}
\text{Given : } \mathbf{x}^{(0)}, \\
\text{Iterate over :} \\
\qquad \mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}, \qquad (3.2.46) \\
\qquad s^{(k)} = \dfrac{(\mathbf{r}^{(k)}, \mathbf{r}^{(k)})}{(\mathbf{r}^{(k)}, \mathbf{A}\mathbf{r}^{(k)})}, \qquad (3.2.47) \\
\qquad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + s^{(k)}\mathbf{r}^{(k)}, \qquad (3.2.48) \\
\qquad k = k + 1, \qquad (3.2.49) \\
\text{until convergence is achieved.}
\end{array}
\right.
$$

3.2.4.2 Preconditioned Steepest Descent Technique

The SD method can be generalized to the *preconditioned steepest descent* (PSD) method. The PSD technique uses one of the standard iterative algorithms to determine a descent direction $\Delta\mathbf{x}^{(k)}$ and then minimizes $F(\mathbf{x}^{(k)} + s^{(k)}\Delta\mathbf{x}^{(k)})$ with respect to $s^{(k)}$ as before. This yields the following algorithm

$$
\text{(PSD)}\left\{
\begin{array}{l}
\text{Given : } \mathbf{x}^{(0)}, \\
\text{Iterate over :} \\
\qquad \mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}, \qquad (3.2.50) \\
\qquad \mathbf{B}\Delta\mathbf{x}^{(k)} = \mathbf{r}^{(k)}, \qquad (3.2.51) \\
\qquad s^{(k)} = \dfrac{(\Delta\mathbf{x}^{(k)}, \mathbf{r}^{(k)})}{(\Delta\mathbf{x}^{(k)}, \mathbf{A}\Delta\mathbf{x}^{(k)})}, \qquad (3.2.52) \\
\qquad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + s^{(k)}\mathbf{r}^{(k)}, \qquad (3.2.53) \\
\qquad k = k + 1, \qquad (3.2.54) \\
\text{until convergence is achieved.}
\end{array}
\right.
$$

Clearly, the SD algorithm is equivalent to using the identity matrix as the preconditioner for PSD. Note that it is possible to extend the technique to search for a minimum in several directions simultaneously (see for example [Levit 82]). However, we have found the following methods to be more robust.

3.2.4.3 Conjugate Gradient Technique

The *conjugate gradient* (CG) method is a variation of the method of steepest descent in which the directions of descent, $\mathbf{p}^{(k)}$, are constrained to be mutually **A**-conjugate [Hestenes 52]. This constraint on the direction vectors yields two desirable properties; first, the convergence is monotonic, and second, the solution is obtained in at most N_{EQ} iterations, assuming the computations are performed with infinite precision. The basic conjugate gradient algorithm is :

(CG)

Given : $\mathbf{x}^{(0)}$,

Initialize :

$$k = 0 , \tag{3.2.55}$$

$$\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)} , \tag{3.2.56}$$

$$\mathbf{p}^{(0)} = \mathbf{r}^{(0)} , \tag{3.2.57}$$

Iterate over :

$$\alpha^{(k)} = \frac{(\mathbf{p}^{(k)}, \mathbf{r}^{(k)})}{(\mathbf{p}^{(k)}, \mathbf{A}\mathbf{p}^{(k)})} , \tag{3.2.58}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{p}^{(k)} , \tag{3.2.59}$$

$$\mathbf{r}^{(k+1)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k+1)} , \tag{3.2.60}$$

$$= \mathbf{r}^{(k)} - \alpha^{(k)}\mathbf{A}\mathbf{p}^{(k)} , \tag{3.2.61}$$

$$\beta^{(k+1)} = \frac{(\mathbf{r}^{(k+1)}, \mathbf{A}\mathbf{p}^{(k)})}{(\mathbf{p}^{(k)}, \mathbf{A}\mathbf{p}^{(k)})} , \tag{3.2.62}$$

$$\mathbf{p}^{(k+1)} = \mathbf{r}^{(k+1)} + \beta^{(k+1)}\mathbf{p}^{(k)} , \tag{3.2.63}$$

$$k = k + 1 , \tag{3.2.64}$$

until convergence is achieved.

In practice, this algorithm may fail to converge in N_{EQ} iterations due to an accumulation of round-off error.

3.2.4.4 Preconditioned Conjugate Gradient Technique

To reduce the accumulation of error and achieve faster convergence, we couple the conjugate gradient algorithm with one of the standard iterative methods. The first step is to *precondition* the original equation system by the splitting matrix **B** of the iterative method chosen. Thus, we solve the equivalent system

$$\mathbf{B}^{-1}\mathbf{A}\mathbf{x} = \mathbf{B}^{-1}\mathbf{b}\,. \tag{3.2.65}$$

Since we are considering only symmetrizable iterative methods, we may symmetrize the preconditioned system by the transformation

$$\mathbf{P}(\mathbf{B}^{-1}\mathbf{A})\mathbf{P}^{-1}(\mathbf{P}\mathbf{x}) = \mathbf{P}\mathbf{B}^{-1}\mathbf{b}\,, \tag{3.2.66}$$

which yields the new system

$$\tilde{\mathbf{A}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}\,, \tag{3.2.67}$$

where

$$\tilde{\mathbf{A}} = \mathbf{P}(\mathbf{B}^{-1}\mathbf{A})\mathbf{P}^{-1}\,, \tag{3.2.68}$$

$$\tilde{\mathbf{x}} = \mathbf{P}\mathbf{x}\,, \tag{3.2.69}$$

and

$$\tilde{\mathbf{b}} = \mathbf{P}\mathbf{B}^{-1}\mathbf{b}\,. \tag{3.2.70}$$

Now, applying the conjugate gradient procedure to (3.2.67), we arrive at the *preconditioned conjugate gradient* (PCG) algorithm :

(PCG)
Given : $\mathbf{x}^{(0)}$,

Initialize :

$$k = 0\,, \tag{3.2.71}$$

$$\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}\,, \tag{3.2.72}$$

$$\mathbf{d}^{(0)} = \mathbf{B}^{-1}\mathbf{r}^{(0)}\,, \tag{3.2.73}$$

$$\mathbf{p}^{(0)} = \mathbf{d}^{(0)}\,, \tag{3.2.74}$$

Iterate over :

$$\alpha^{(k)} = \frac{(\mathbf{r}^{(k)}, \mathbf{B}^{-1}\mathbf{r}^{(k)})}{(\mathbf{p}^{(k)}, \mathbf{A}\mathbf{p}^{(k)})}\,, \tag{3.2.75}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{p}^{(k)}\,, \tag{3.2.76}$$

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha^{(k)}\mathbf{A}\mathbf{p}^{(k)}\,, \tag{3.2.77}$$

$$\mathbf{d}^{(k+1)} = \mathbf{B}^{-1}\mathbf{r}^{(k+1)}\,, \tag{3.2.78}$$

$$\beta^{(k+1)} = \frac{(\mathbf{r}^{(k+1)}, \mathbf{B}^{-1}\mathbf{r}^{(k+1)})}{(\mathbf{r}^{(k)}, \mathbf{B}^{-1}\mathbf{r}^{(k)})}\,, \tag{3.2.79}$$

$$\mathbf{p}^{(k+1)} = \mathbf{d}^{(k+1)} + \beta^{(k+1)}\mathbf{p}^{(k)}\,, \tag{3.2.80}$$

$$k = k + 1\,, \tag{3.2.81}$$

until convergence is achieved.

Note that the CG algorithm is equivalent to one which uses the identity matrix as the preconditioner for PCG.

3.2.5 Cost

As with the direct techniques, we consider the costs associated with both storage and computation. Since most of the algorithms are implemented in the residual form, we only consider the costs for that form. Since the costs vary widely between the algorithms, only the general form of the costs is given.

The storage cost of an iterative method in residual form has three components. The first is the number of *vectors* of length N_{EQ} needed by the solution algorithm. The second is the storage of the sparse symmetric positive-definite matrix **A** and the third is the storage required to hold the splitting matrix **B**, typically stored in factored form.

The standard, linear-stationary methods may all be implemented with only three vectors of storage, while the preconditioned conjugate gradient algorithm requires five. The storage required by a vector **x** is denoted $S(\mathbf{x})$ and is defined as

$$S(\mathbf{x}) = N_{EQ} \text{ words} . \tag{3.2.82}$$

There are many ways of storing the sparse matrix **A**. In choosing one, it is important to consider the trade-offs among storage, speed and ease of implementation within the framework of forming the matrix-vector product **Ax**. If **A** is very sparse, as typical in large finite element problems, it may be advantageous to use a *row-column* storage scheme. In a row-column scheme, the non-zero terms of **A** are stored along with an indexing vector that defines each term's location in **A**. The amount of storage required is S_{RC} words, which includes both the primary storage of the non-zero terms and the overhead storage for addressing. It is

$$S_{RC}(\mathbf{A}) = 2\,\aleph(\mathbf{A}) \text{ words} , \tag{3.2.83}$$

where $\aleph(\mathbf{A})$ is the total number of non-zero terms in **A**.

Another possibility that fits well with the finite element nature of the problem is the storage of **A** as a collection of unassembled element contributions. The matrix-vector product **Ax** can then be easily computed at the element level and assembled into a global vector. This scheme entails more storage than the row-column scheme, but has the distinct advantage of compatibility with traditional finite element data structures and architecture. The storage required is

$$S_{EL}(\mathbf{A}) = N_{EL} \times \frac{1}{2}(N_{ELEQ} + 1)N_{ELEQ} \text{ words} , \tag{3.2.84}$$

where N_{ELEQ} is the number of element equations.

The storage required by $\mathbf{B}$ depends on the iterative algorithm. It can range from zero for the simplest algorithm, Richardson iteration, to $S_{\sigma-factor}(\mathbf{A})$ for the more complex, like Gauss-Seidel. The storage required by a sum decomposition, $S_{\sigma-factor}$, depends on the storage form chosen, and may only be bounded by

$$S_{RC}(\mathbf{A}) \leq S_{\sigma-factor}(\mathbf{A}) \leq S_{EL}(\mathbf{A}) . \tag{3.2.85}$$

The computational expense also differs between iterative algorithms. It is computed on a "per iteration" basis, since the number of iterations is unknown in advance. As with storage, the costs emanate from three different types of terms.

The first type of term is vector-scalar or vector-vector operations. This includes multipling a vector by a scalar, vector inner-products, addition of two vectors, etc. The cost of the these terms is given by

$$C(\mathbf{x} \cdot \mathbf{x}) = N_{EQ} \text{ ops} . \tag{3.2.86}$$

The second type of term is the matrix-vector product of the form $\mathbf{Ax}$. The cost of this operation is determined by the storage form chosen for $\mathbf{A}$. If $\mathbf{A}$ is stored in row-column form, the cost is

$$C_{RC}(\mathbf{Ax}) = \aleph(\mathbf{A}) \text{ ops} , \tag{3.2.87}$$

while the element storage cost is

$$C_{EL}(\mathbf{Ax}) = N_{EL} \times N_{ELEQ}^2 \text{ ops} . \tag{3.2.88}$$

Finally, there are the costs associated with terms of the form $\mathbf{B}^{-1}\mathbf{r}$. In general, there are two components to this, the possible initial factorization of $\mathbf{B}$ and the solution

of $\mathbf{B}^{-1}\mathbf{r}$. For the standard iterative algorithms using sum decompositions, these costs are in the range

$$C_{\sigma-factor}(\mathbf{B}) = 0 \rightarrow N_{EQ} \text{ ops} \tag{3.2.89}$$

and

$$C_{\sigma-solve}(\mathbf{B}) = 0 \rightarrow C_{EL}(\mathbf{Ax}) \text{ ops} . \tag{3.2.90}$$

The exact cost is algorithm-dependent. Note the reversal in high-cost roles between the factor and solve costs.

For the three-dimensional model problem in Figure 3.1.2, the costs are

$$S(\mathbf{x}) = pqr \text{ words} , \tag{3.2.91}$$

$$S_{RC}(\mathbf{A}) \approx 16pqr \text{ words} , \tag{3.2.92}$$

$$S_{EL}(\mathbf{A}) = 36(p-1)(q-1)(r-1) \text{ words} , \tag{3.2.93}$$

$$C(\mathbf{x} \cdot \mathbf{x}) = pqr \text{ ops} , \tag{3.2.94}$$

$$C_{RC}(\mathbf{Ax}) \approx 8pqr \text{ ops} , \tag{3.2.95}$$

and

$$C_{EL}(\mathbf{Ax}) = 64pqr \text{ ops} . \tag{3.2.96}$$

To compute the costs for two-dimensional problems, the leading constants must be changed to reflect the reduced element connectivity.

As before, if we consider equal refinement in all directions,

$$p = q = r , \tag{3.2.97}$$

$$S(\mathbf{x}) = p^3 \text{ words}, \tag{3.2.98}$$

$$S_{RC}(\mathbf{A}) \approx 16p^3 \text{ words}, \tag{3.2.99}$$

$$S_{EL}(\mathbf{A}) = 36(p-1)^3 \text{ words}, \tag{3.2.100}$$

$$C(\mathbf{x} \cdot \mathbf{x}) = p^3 \text{ ops}, \tag{3.2.101}$$

$$C_{RC}(\mathbf{Ax}) \approx 8p^3 \text{ ops}, \tag{3.2.102}$$

$$C_{EL}(\mathbf{Ax}) = 64p^3 \text{ ops}. \tag{3.2.103}$$

In the next chapter, we introduce approximate factorizations found to be cost-effective for the solution of finite element problems.

Chapter 4

Approximate Factorizations

This chapter describes the techniques developed for approximating the matrix **A**. These approximations may either be used as the splitting matrix in the definition of an iterative algorithm or as a preconditioner for any of the standard iterative algorithms in the last chapter. The convergence rate of the iterative algorithm depends heavily upon the approximating matrix **B**. Note that if $\mathbf{B} = \mathbf{A}$, all of the iterative algorithms immediately converge to the exact solution **x**. First we consider the form of the matrix to be approximately factored. Next the simplest approximate factorization based on two-component splitting is defined. Generalization of two-component splitting leads to the definition of multi-component splits. Next a variety of element-by-element approximate factorizations are defined. Finally, the choice of parameters governing the accuracy of the approximate factorizations is considered.

§4.1 Form of the Approximate Factors

For ease in representation and to encompass a wide range of algorithms we employ the following two-stage approximate factorization of **A**. The first stage is the reduction of the matrix **A** to a form that may be easily factored. Instead of directly approximately factorizing the matrix **A**, we will approximately factor a matrix $\tilde{\mathbf{A}}$ which is "close" to

$\mathbf{A}$ and has a known form. We consider

$$\tilde{\mathbf{A}} \approx \mathbf{A}\,, \tag{4.1.1}$$

where $\tilde{\mathbf{A}}$ has the form

$$\tilde{\mathbf{A}} = \mathbf{W}^{1/2}(\mathbf{I} + \epsilon\overline{\mathbf{A}})\mathbf{W}^{1/2}\,. \tag{4.1.2}$$

$\mathbf{W}$ is a diagonal, symmetric, positive-definite matrix which should be thought of as a scaling or normalization matrix. It reduces the terms of $\mathbf{A}$ to $O(1)$. The scalar ϵ is a positive real number that should be thought of as a "small" parameter. The matrix $\overline{\mathbf{A}}$ is a prescaled approximation to $\mathbf{A}$ which has the same sparsity structure as $\mathbf{A}$. Specific choices of $\mathbf{W}$, ϵ and $\overline{\mathbf{A}}$ are considered later in Section 4.

The second and final stage of the approximate factorization is the definition of the splitting matrix $\mathbf{B}$ as an approximation to the matrix $\tilde{\mathbf{A}}$. Thus, we define

$$\mathbf{B} = \mathbf{W}^{1/2}\mathbf{C}\mathbf{W}^{1/2}\,, \tag{4.1.3}$$

where

$$\mathbf{C} \approx \mathbf{I} + \epsilon\overline{\mathbf{A}}\,. \tag{4.1.4}$$

The matrix $\mathbf{C}$ should be easily factored, and it must be possible to store $\mathbf{C}^{-1}$ in a compact form. Further, $\mathbf{C}^{-1}$ must be well-behaved. We have found it advantageous for $\mathbf{C}$ to be symmetric positive-definite. However, we do not impose this restriction in the derivation of the following approximations. We consider definitions of $\mathbf{C}$ based on *sum-to-product* approximations. A sum-to-product approximation approximates the sum of a number of terms by the product of scaled terms augmented by the identity. Various choices for the matrix $\mathbf{C}$ are explored in the following sections.

§4.2 Two-Component Splitting

The simplest of the sum-to-product type approximations is *two-component splitting.* This form of approximation is similar to the *alternating direction* method [Douglas 56,62]. It decomposes the operator into the sum of two simpler operators. Let $\overline{\mathbf{A}}$ be decomposed as follows:

$$\overline{\mathbf{A}} = \overline{\mathbf{A}}_1 + \overline{\mathbf{A}}_2 \,. \tag{4.2.1}$$

Then a possible definition of $\mathbf{C}$ is

$$\mathbf{C} = (\mathbf{I} + \epsilon\overline{\mathbf{A}}_1)(\mathbf{I} + \epsilon\overline{\mathbf{A}}_2) \,. \tag{4.2.2}$$

Expanding terms yields

$$\begin{aligned} \mathbf{C} &= \mathbf{I} + \epsilon\overline{\mathbf{A}} + \epsilon^2\overline{\mathbf{A}}_1\overline{\mathbf{A}}_2 \\ &= \mathbf{I} + \epsilon\overline{\mathbf{A}} + O\left(\epsilon^2\right) . \end{aligned} \tag{4.2.3}$$

The last line suggests the nature of the approximation. Computational simplicity is gained if $\overline{\mathbf{A}}_1$ and $\overline{\mathbf{A}}_2$ are very sparse and are easier to factor than $\overline{\mathbf{A}}$. Note that if $\overline{\mathbf{A}}_1$ and $\overline{\mathbf{A}}_2$ do not commute, $\mathbf{C}$ will not in general be symmetric even if $\overline{\mathbf{A}}$ is symmetric. In addition, the ordering of terms in the product approximation influences the error in the approximation, the ϵ^2-term. An alternative definition of the two-component split is thus

$$\mathbf{C} = (\mathbf{I} + \epsilon\overline{\mathbf{A}}_2)(\mathbf{I} + \epsilon\overline{\mathbf{A}}_1) \,. \tag{4.2.4}$$

Expanding terms yields

$$\begin{aligned} \mathbf{C} &= \mathbf{I} + \epsilon\overline{\mathbf{A}} + \epsilon^2\overline{\mathbf{A}}_2\overline{\mathbf{A}}_1 \\ &= \mathbf{I} + \epsilon\overline{\mathbf{A}} + O\left(\epsilon^2\right) . \end{aligned} \tag{4.2.5}$$

Note the difference in the ϵ^2 error terms of the two orderings. We explore ways of symmetrizing the product approximation in the next section.

As an example of a two-component split, consider

$$\overline{\mathbf{A}}_1 = \tilde{\mathbf{L}}_\sigma(\overline{\mathbf{A}}) \tag{4.2.6}$$

and

$$\overline{\mathbf{A}}_2 = \tilde{\mathbf{U}}_\sigma(\overline{\mathbf{A}}) \,. \tag{4.2.7}$$

B has the simple form

$$\mathbf{B} = \mathbf{W}^{1/2}\big(\mathbf{I} + \epsilon\tilde{\mathbf{L}}_\sigma(\overline{\mathbf{A}})\big)\big(\mathbf{I} + \epsilon\tilde{\mathbf{U}}_\sigma(\overline{\mathbf{A}})\big)\mathbf{W}^{1/2} \,. \tag{4.2.8}$$

B is already factored and the factors require no more storage than the factors of **A**. Only diagonal scaling, forward reductions, and back substitutions with sparse triangular arrays are needed to solve equations with **B** as coefficient matrix. This eliminates the cost of factorization and obviates the storage penalties due to fill-in. Equation (4.2.8) represents a symmetrized Gauss-Seidel type approximate factorization.

§4.3 Multi-Component Splitting

Having defined the two-component splitting method, we now consider the generalization to *multi-component splitting*. The operator is approximated by a product formed of its N components. The "quality" of the approximation depends on the form and order of the terms in the product.

4.3.1 One-Pass Multi-Component Splitting

We first analyze the simplest of the multi-component splittings, the *one-pass* multi-

component splitting. Consider a multi-component sum decomposition of $\overline{\mathbf{A}}$:

$$\overline{\mathbf{A}} = \sum_{i=1}^{N} \overline{\mathbf{A}}_i \,. \tag{4.3.1}$$

Let the matrix $\mathbf{C}$ be defined by

$$\mathbf{C} = \prod_{i=1}^{N} (\mathbf{I} + \epsilon \overline{\mathbf{A}}_i) \,. \tag{4.3.2}$$

Expanding terms yields

$$\begin{aligned} \mathbf{C} &= (\mathbf{I} + \epsilon \overline{\mathbf{A}}_1)(\mathbf{I} + \epsilon \overline{\mathbf{A}}_2) \ldots (\mathbf{I} + \epsilon \overline{\mathbf{A}}_N) \\ &= \mathbf{I} + \epsilon \overline{\mathbf{A}} + \epsilon^2 \sum_{i=1}^{N-1} \left(\overline{\mathbf{A}}_i \sum_{j=i+1}^{N} \overline{\mathbf{A}}_j \right) + O\left(\epsilon^3\right) \\ &= \mathbf{I} + \epsilon \overline{\mathbf{A}} + O\left(\epsilon^2\right) . \end{aligned} \tag{4.3.3}$$

Clearly, this is a straight-forward generalization of the two-component splitting. The ordering of the terms or, equivalently, the definition of the $\overline{\mathbf{A}}_i$'s, affects the exact form of the error.

4.3.2 Two-Pass Multi-Component Splitting

The generalization of the preceding case has qualitative advantages under certain circumstances [Marchuk 75]. In particular, we attempt to symmetrize $\mathbf{C}$ by using both forward and backward products of terms. Let $\mathbf{C}$ be defined by

$$\mathbf{C} = \prod_{i=1}^{N} (\mathbf{I} + \frac{\epsilon}{2} \overline{\mathbf{A}}_i) \prod_{i=N}^{1} (\mathbf{I} + \frac{\epsilon}{2} \overline{\mathbf{A}}_i) \,. \tag{4.3.4}$$

Expanding terms yields

$$\begin{aligned}\mathbf{C} &= (\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_1)(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_2)\dots(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_N)\\ &\quad\times(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_N)(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_{N-1})\dots(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_1)\\ &= \mathbf{I}+\epsilon\overline{\mathbf{A}}+\epsilon^2\left(\frac{1}{2}\overline{\mathbf{A}}^2-\frac{1}{4}\sum_{i=1}^{N}\overline{\mathbf{A}}_i^2\right)+O\left(\epsilon^3\right)\\ &= \mathbf{I}+\epsilon\overline{\mathbf{A}}+O\left(\epsilon^2\right).\end{aligned} \tag{4.3.5}$$

If each $\overline{\mathbf{A}}_i$ is symmetric and positive semi-definite, then $\mathbf{C}$ is symmetric and positive-definite. This is easily shown by

$$\begin{aligned}\mathbf{x}^T\mathbf{C}\mathbf{x} &= \mathbf{x}^T(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_1)(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_2)\dots(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_N)\\ &\quad\times(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_N)(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_{N-1})\dots(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_1)\mathbf{x}\\ &= \mathbf{x}_1^T(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_2)(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_3)\dots(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_N)\\ &\quad\times(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_N)(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_{N-1})\dots(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_2)\mathbf{x}_1\\ &= \mathbf{x}_{N-1}^T(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_N)(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_N)\mathbf{x}_{N-1}\\ &= \mathbf{x}_N^T\mathbf{x}_N\\ &> 0 \quad \forall \quad \mathbf{x}\neq\mathbf{o}.\end{aligned} \tag{4.3.6}$$

Definitions of two-pass forms based on more general orderings exist, but are not to be considered here.

4.3.3 Multi-Pass Multi-Component Splitting

Analogous to the generalization from two-component to multi-component, we now generalize from two-pass to multi-pass. Let the matrix $\mathbf{C}$ be defined by

$$\mathbf{C} = \prod_{j=1}^{N_{PASS}}\left[\prod_{i=1}^{N}(\mathbf{I}+\frac{\epsilon}{N_{PASS}}\overline{\mathbf{A}}_{k_j(i)})\right]. \tag{4.3.7}$$

Expanding terms yields

$$\mathbf{C} = \mathbf{I} + \epsilon\overline{\mathbf{A}} + O\left(\epsilon^2\right), \tag{4.3.8}$$

where N_{PASS} is the number of passes and $k_j(i)$ defines the order of the components for pass j. Typically, one uses a symmetrized form; thus N_{PASS} is even. Symmetry is maintained by choosing $k_j(i)$ such that

$$k_j(i) = k_{N_{PASS}-j+1}(N - i + 1)\,. \tag{4.3.9}$$

For the symmetric case, composed of pairs of simple forward and backward sweeps,

$$\mathbf{C} = \prod_{j=1}^{N_{PASS}/2}\left[\prod_{i=1}^{N}(\mathbf{I} + \frac{\epsilon}{N_{PASS}}\overline{\mathbf{A}}_i)\prod_{i=N}^{1}(\mathbf{I} + \frac{\epsilon}{N_{PASS}}\overline{\mathbf{A}}_i)\right]. \tag{4.3.10}$$

Expanding terms yields

$$\begin{aligned}\mathbf{C} &= \mathbf{I} + \epsilon\overline{\mathbf{A}} + \epsilon^2\left(\frac{1}{2}\overline{\mathbf{A}}^2 - \frac{1}{2N_{PASS}}\sum_{i=1}^{N}\overline{\mathbf{A}}_i^2\right) + O\left(\epsilon^3\right)\\ &= \mathbf{I} + \epsilon\overline{\mathbf{A}} + O\left(\epsilon^2\right).\end{aligned} \tag{4.3.11}$$

§4.4 Element-by-Element Splits

The *element-by-element* (EL×EL) approximate factorization is simply a multi-component splitting. The components of the matrix are the prescaled finite element contributions to the global matrix.

We assume

$$\overline{\mathbf{A}} = \sum_{e=1}^{N_{EL}}\overline{\mathbf{A}}_e\,, \tag{4.4.1}$$

where $\overline{\mathbf{A}}_e$ is the e^{th} element contribution to $\overline{\mathbf{A}}$. To simplify the notation, we use globalized element arrays $\overline{\mathbf{A}}_e$ instead of a combination of the local element array $\mathbf{a}^e$ and a Boolean mapping matrix. These globalized element arrays are very sparse[1]. In practice, however, all operations are performed by localizing all necessary information for the element under consideration, performing the operation on the local (element) level, and then globalizing the result.

The term *element* is used in the generic sense of a *subdomain model*, where an element could be an individual finite element or a subassembly of elements. Thus, we allow limited assembly. Various equivalent terminologies have been used to define this concept, such as *substructures* and *superelements*. Subdomain finite element models inherit the symmetry and definiteness properties of the global array.

4.4.1 One Pass EL×EL Splitting

Substituting the definition of $\overline{\mathbf{A}}_e$, given in (4.4.1) , into the general one-pass multi-component splitting defines $\mathbf{C}$ as

$$\mathbf{C} = \prod_{e=1}^{N_{EL}} (\mathbf{I} + \epsilon \overline{\mathbf{A}}_e) \,. \tag{4.4.2}$$

In practice, the result of $\mathbf{C}^{-1}$ acting on a vector is computed by peeling off the "inverses" of successive terms in the product. The effects of these element inverses are actually computed using one of the standard direct solution techniques which are based on product factorizations. The Crout-factored form corresponding to (4.4.2) is

$$\mathbf{C} = \prod_{e=1}^{N_{EL}} \mathbf{L}_\pi(\mathbf{I} + \epsilon \overline{\mathbf{A}}_e)\mathbf{D}_\pi(\mathbf{I} + \epsilon \overline{\mathbf{A}}_e)\mathbf{U}_\pi(\mathbf{I} + \epsilon \overline{\mathbf{A}}_e) \,. \tag{4.4.3}$$

[1]For a four-node two-dimensional heat conduction element, $\overline{\mathbf{A}}_e$ has at most 6 non-zero off-diagonal terms independent of the number of global equations.

If $\mathbf{D}_\pi(\mathbf{I}+\epsilon\overline{\mathbf{A}}_e)$ is positive, it is also possible to express (4.4.2) in terms of Cholesky factors as

$$\mathbf{C} = \prod_{e=1}^{N_{EL}} \tilde{\mathbf{L}}_\pi(\mathbf{I}+\epsilon\overline{\mathbf{A}}_e)\tilde{\mathbf{U}}_\pi(\mathbf{I}+\epsilon\overline{\mathbf{A}}_e)\,. \tag{4.4.4}$$

Note that both (4.4.3) and (4.4.4) are equivalent to (4.4.2) .

To avoid the cost of computing the exact product factors, one may instead approximate (4.4.2) using a sum factorization for the terms in the product. Thus, we define

$$\mathbf{C} = \prod_{e=1}^{N_{EL}} \big(\mathbf{I}+\epsilon\mathbf{L}_\sigma(\overline{\mathbf{A}}_e)\big)\big(\mathbf{I}+\epsilon\mathbf{D}_\sigma(\overline{\mathbf{A}}_e)\big)\big(\mathbf{I}+\epsilon\mathbf{U}_\sigma(\overline{\mathbf{A}}_e)\big)\,, \tag{4.4.5}$$

or

$$\mathbf{C} = \prod_{e=1}^{N_{EL}} \big(\mathbf{I}+\epsilon\tilde{\mathbf{L}}_\sigma(\overline{\mathbf{A}}_e)\big)\big(\mathbf{I}+\epsilon\tilde{\mathbf{U}}_\sigma(\overline{\mathbf{A}}_e)\big)\,. \tag{4.4.6}$$

Note both (4.4.5) and (4.4.6) are approximations to (4.4.2) .

4.4.2 Two-Pass EL×EL Splits

As with multi-component splits, we symmetrize the approximation by using two passes, a forward sweep, and a backward sweep. Thus, $\mathbf{C}$ is defined as

$$\mathbf{C} = \prod_{e=1}^{N_{EL}} (\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_e) \prod_{e=N_{EL}}^{1} (\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_e)\,. \tag{4.4.7}$$

The Crout-factored form corresponding to (4.4.7) is

$$\begin{aligned}\mathbf{C} = &\prod_{e=1}^{N_{EL}} \mathbf{L}_\pi(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_e)\mathbf{D}_\pi(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_e)\mathbf{U}_\pi(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_e)\\ &\times \prod_{e=N_{EL}}^{1} \mathbf{L}_\pi(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_e)\mathbf{D}_\pi(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_e)\mathbf{U}_\pi(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_e)\,,\end{aligned} \tag{4.4.8}$$

and the Cholesky-factored form is

$$\begin{aligned}\mathbf{C} = &\prod_{e=1}^{N_{EL}} \tilde{\mathbf{L}}_\pi(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_e)\tilde{\mathbf{U}}_\pi(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_e) \\ &\times \prod_{e=N_{EL}}^{1} \tilde{\mathbf{L}}_\pi(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_e)\tilde{\mathbf{U}}_\pi(\mathbf{I}+\tfrac{\epsilon}{2}\overline{\mathbf{A}}_e)\,.\end{aligned} \tag{4.4.9}$$

Note that both (4.4.8) and (4.4.9) are equivalent to (4.4.7) .

As with the one-pass form of the EL×EL split, it may be advantageous to approximate the factors of the element arrays with a sum factorization. Thus, we may define $\mathbf{C}$ by

$$\begin{aligned}\mathbf{C} = &\prod_{e=1}^{N_{EL}} \Big(\mathbf{I}+\tfrac{\epsilon}{2}\mathbf{L}_\sigma(\overline{\mathbf{A}}_e)\Big)\Big(\mathbf{I}+\tfrac{\epsilon}{2}\mathbf{D}_\sigma(\overline{\mathbf{A}}_e)\Big)\Big(\mathbf{I}+\tfrac{\epsilon}{2}\mathbf{U}_\sigma(\overline{\mathbf{A}}_e)\Big) \\ &\times \prod_{e=N_{EL}}^{1} \Big(\mathbf{I}+\tfrac{\epsilon}{2}\mathbf{L}_\sigma(\overline{\mathbf{A}}_e)\Big)\Big(\mathbf{I}+\tfrac{\epsilon}{2}\mathbf{D}_\sigma(\overline{\mathbf{A}}_e)\Big)\Big(\mathbf{I}+\tfrac{\epsilon}{2}\mathbf{U}_\sigma(\overline{\mathbf{A}}_e)\Big)\,,\end{aligned} \tag{4.4.10}$$

or alternatively

$$\begin{aligned}\mathbf{C} = &\prod_{e=1}^{N_{EL}} \Big(\mathbf{I}+\tfrac{\epsilon}{2}\tilde{\mathbf{L}}_\sigma(\overline{\mathbf{A}}_e)\Big)\Big(\mathbf{I}+\tfrac{\epsilon}{2}\tilde{\mathbf{U}}_\sigma(\overline{\mathbf{A}}_e)\Big) \\ &\times \prod_{e=N_{EL}}^{1} \Big(\mathbf{I}+\tfrac{\epsilon}{2}\tilde{\mathbf{L}}_\sigma(\overline{\mathbf{A}}_e)\Big)\Big(\mathbf{I}+\tfrac{\epsilon}{2}\tilde{\mathbf{U}}_\sigma(\overline{\mathbf{A}}_e)\Big)\,.\end{aligned} \tag{4.4.11}$$

Note once again that the sum factorizations (4.4.10) and (4.4.11) are only approximations to (4.4.7) .

4.4.3 Reordered EL×EL Splits

The ordering of the factors in the EL×EL splits influences how well $\mathbf{C}$ approximates $\mathbf{I}+\epsilon\overline{\mathbf{A}}$. The global product decomposition based on Crout factorization,

$$\mathbf{I}+\epsilon\overline{\mathbf{A}} = \mathbf{L}_\pi(\mathbf{I}+\epsilon\overline{\mathbf{A}})\mathbf{D}_\pi(\mathbf{I}+\epsilon\overline{\mathbf{A}})\mathbf{U}_\pi(\mathbf{I}+\epsilon\overline{\mathbf{A}})\,, \tag{4.4.12}$$

suggests that it might be worthwhile to reorder the factors in the EL×EL splits such that all lower triangular factors precede diagonals which, in turn, precede upper triangular factors. This results in the following *reordered* schemes. In reordered one-pass Crout factorization, $\mathbf{C}$ is defined by

$$\mathbf{C} = \left[\prod_{e=1}^{N_{EL}} \mathbf{L}_\pi(\mathbf{I} + \epsilon\overline{\mathbf{A}}_e)\right]\left[\prod_{e=1}^{N_{EL}} \mathbf{D}_\pi(\mathbf{I} + \epsilon\overline{\mathbf{A}}_e)\right]\left[\prod_{e=N_{EL}}^{1} \mathbf{U}_\pi(\mathbf{I} + \epsilon\overline{\mathbf{A}}_e)\right]. \tag{4.4.13}$$

When $\mathbf{D}_\pi(\mathbf{I} + \epsilon\overline{\mathbf{A}}_e)$ is positive, we may also define $\mathbf{C}$ corresponding to the reordered one-pass Cholesky factors as

$$\mathbf{C} = \left[\prod_{e=1}^{N_{EL}} \tilde{\mathbf{L}}_\pi(\mathbf{I} + \epsilon\overline{\mathbf{A}}_e)\right]\left[\prod_{e=N_{EL}}^{1} \tilde{\mathbf{U}}_\pi(\mathbf{I} + \epsilon\overline{\mathbf{A}}_e)\right]. \tag{4.4.14}$$

Note (4.4.14) and (4.4.13) are not generally identical.

Similarly, we may define the reordered one-pass sum factorizations as

$$\mathbf{C} = \left[\prod_{e=1}^{N_{EL}} \left(\mathbf{I} + \epsilon\mathbf{L}_\sigma(\overline{\mathbf{A}}_e)\right)\right]\left[\prod_{e=1}^{N_{EL}} \left(\mathbf{I} + \epsilon\mathbf{D}_\sigma(\overline{\mathbf{A}}_e)\right)\right]\left[\prod_{e=N_{EL}}^{1} \left(\mathbf{I} + \epsilon\mathbf{U}_\sigma(\overline{\mathbf{A}}_e)\right)\right], \tag{4.4.15}$$

or

$$\mathbf{C} = \left[\prod_{e=1}^{N_{EL}} \left(\mathbf{I} + \epsilon\tilde{\mathbf{L}}_\sigma(\overline{\mathbf{A}}_e)\right)\right]\left[\prod_{e=N_{EL}}^{1} \left(\mathbf{I} + \epsilon\tilde{\mathbf{U}}_\sigma(\overline{\mathbf{A}}_e)\right)\right]. \tag{4.4.16}$$

Note that in the case of symmetric $\overline{\mathbf{A}}$, symmetry is preserved by (4.4.13)—(4.4.16); thus there is little motivation for similarly reordering the two-pass versions.

4.4.4 Cost

In discussing the cost of EL×EL approximate factorizations, we consider only those costs directly associated with the storage, factorization, and solution of the matrix $\mathbf{B}$.

The other costs depend on the particular iterative solution technique coupled with the approximate factorization and are described in Chapter 3.

The storage cost for the approximate factorization matrix **B**, given in the form defined by (4.1.1)—(4.1.4) , is composed of two parts. The first is the storage for the diagonal scaling matrix **W** or its square root,

$$S(\mathbf{W}) = S(\mathbf{W}^{1/2}) = N_{EQ} \text{ words} . \tag{4.4.17}$$

The second is the storage of the matrix **C** or its element factors,

$$\begin{aligned} S_{EL}(\mathbf{C}) =& S_{EL-\pi-factor}(\mathbf{C}) = S_{EL-\sigma-factor}(\mathbf{C}) \\ =& N_{EL}N_{ELEQ}^2 \text{ words} . \end{aligned} \tag{4.4.18}$$

If $\overline{\mathbf{A}}$ is symmetric,

$$\begin{aligned} S_{EL}(\mathbf{C}) =& S_{EL-\pi-factor}(\mathbf{C}) = S_{EL-\sigma-factor}(\mathbf{C}) \\ =& N_{EL}\frac{N_{ELEQ}(N_{ELEQ}+1)}{2} \text{ words} . \end{aligned} \tag{4.4.19}$$

Note that **C** need not be stored, but may be computed on an element-by-element basis if storage for the matrix is unavailable.

The cost of computing $\mathbf{B}^{-1}\mathbf{x}$ also has two parts. The first is the cost associated with the factorization of **B**. The factorization of **B** entails the calculation of the square root of **W** and the element factors of **C**; these are

$$C_{\sqrt{}}(\mathbf{W}) = O\left(N_{EQ}\right) \text{ ops} , \tag{4.4.20}$$

$$C_{EL-\sigma-factor}(\mathbf{C}) = 0 \rightarrow N_{EL}N_{ELEQ} \text{ ops} , \tag{4.4.21}$$

they can be processed in parallel. The eight groups, however, need to be processed sequentially. For analogous two-dimensional domains, four-element groups need to be employed. We discuss this further in the next chapter.

It has been our computational experience that if **A** is symmetric and positive-definite, qualitatively faithful approximate factorizations, which preserve these properties, perform much better then those which do not. Consequently, in the numerical examples presented herein we employed qualitatively faithful approximate factorizations.

§4.5 Choice of Parameters

The splitting matrix **B** is defined through the two-stage approximation of the matrix **A** given in the first section. In the previous sections, we have considered definitions for **C**, the second stage in the approximation of **A**. We now consider the first stage of approximation, converting **A** into $\overline{\mathbf{A}}$, and the associated definitions of **W**, ϵ, and $\overline{\mathbf{A}}$.

4.5.1 Parabolic Regularization Parameters

The choice of parameters is motivated by the derivation of the parabolic regularization algorithm in the previous chapter. The parameters are defined by

$$\mathbf{W} = \mathbf{D}_\sigma(\mathbf{A}) \tag{4.5.1}$$

and

$$\overline{\mathbf{A}} = \frac{1}{\epsilon}\mathbf{W}^{-1/2}\mathbf{A}\mathbf{W}^{-1/2}\,, \tag{4.5.2}$$

where the pseudo-time step of the parabolic regularization algorithm, $\Delta\tau$, is assumed equal to 1. Thus the matrix which we approximately factor, $\tilde{\mathbf{A}}$, is

$$\tilde{\mathbf{A}} = \mathbf{D}_\sigma(\mathbf{A}) + \mathbf{A}\,. \tag{4.5.3}$$

The additional term $\mathbf{D}_\sigma(\mathbf{A})$ in (4.5.3) is spurious and, as such, reduces the convergence rate of the iterative scheme employed.

4.5.2 Optimum Parameters

To improve the convergence rate, we suggest alternate definitions for the parameters governing the first stage of the approximate factorization. As before, we define the scaling matrix as

$$\mathbf{W} = \mathbf{D}_\sigma(\mathbf{A}) . \tag{4.5.4}$$

However, to remove the spurious term in $\tilde{\mathbf{A}}$ we define $\overline{\mathbf{A}}$ as

$$\overline{\mathbf{A}} = \frac{1}{\epsilon}\mathbf{W}^{-1/2}(\mathbf{A} - \mathbf{D}_\sigma(\mathbf{A}))\mathbf{W}^{-1/2} , \tag{4.5.5}$$

which leads to the optimum value for $\tilde{\mathbf{A}}$,

$$\tilde{\mathbf{A}} = \mathbf{A} . \tag{4.5.6}$$

Matrices of the form $\tilde{\mathbf{A}} = \mathbf{D}_\sigma(\mathbf{A}) + \Delta\tau\mathbf{A}$ were introduced in [Hughes 83a]. Nour-Omid and Parlett [Nour-Omid 82] analytically investigated the effectiveness of matrices of this type on a simple one-dimensional model problem and concluded that the optimal value of $\Delta\tau$ was ∞. This limit is achieved by the definitions (4.5.4) and (4.5.5) .

4.5.3 Size of ϵ

The size of the parameter ϵ is as yet unspecified. The approximate factorizations under consideration have all had error terms of order ϵ^2 and higher. The "quality" of the approximation is governed by the size of these error terms. Clearly, if the terms in $\overline{\mathbf{A}}$ are $O(1)$, the approximate operator $\mathbf{B}$ approaches the exact operator $\mathbf{A}$ as $\epsilon \to 0$. We now

determine the value of ϵ which scales the terms in $\overline{\mathbf{A}}$ to be $O(1)$. This scaling is implicit, unlike that of $\mathbf{W}^{1/2}$, and is never actually performed during numerical computation. In the following analysis we assume that $\mathbf{W}$ and $\overline{\mathbf{A}}$ are specified by (4.5.4) and (4.5.5), respectively. To simplify the following calculations, we consider the case of a body composed of an isotropic, homogeneous, linear material that has been uniformly meshed. Thus, away from the nodes on the boundary, we can assume that all the diagonal (d) components of the matrix $\mathbf{A}$ are approximately equal, so that

$$A_{ii} \approx A_d \quad \forall \quad i \in \{1, 2, \ldots, N_{EQ}\} \text{ (no sum)} . \tag{4.5.7}$$

Similarly, the magnitudes of the off-diagonal (o) terms are assumed approximately equal,

$$|A_{ij}| \approx A_o \quad \forall \quad i \neq j \;\; i, j \in \{1, 2, \ldots, N_{EQ}\} . \tag{4.5.8}$$

Substitution into (4.5.5) gives :

$$|\overline{A}_{ij}| = \begin{cases} 0, & i = j \\ \dfrac{1}{\epsilon}\dfrac{A_o}{A_d}, & i \neq j \end{cases} \qquad \forall \quad i, j \in \{1, 2, \ldots, N_{EQ}\} . \tag{4.5.9}$$

For the terms in $\overline{\mathbf{A}}$ to be $O(1)$, we define ϵ as

$$\epsilon = \frac{A_o}{A_d} . \tag{4.5.10}$$

From the positive-definitness of the global operator, we know $A_d > A_o$, which implies

$$\epsilon \leq 1 . \tag{4.5.11}$$

Considering the general form of $\mathbf{A}$ in more detail, we recall from Chapter 2

$$\mathbf{A} = \mathbf{M} + \alpha \Delta t \mathbf{K}_T , \tag{4.5.12}$$

where $\mathbf{M}$ is the positive-definite symmetric mass matrix, α is a parameter governing the stability and accuracy of time integration, Δt is the time step, and $\mathbf{K}_T$ is the positive-semi-definite symmetric tangent stiffness. Reducing the terms in $\mathbf{A}$ into their diagonal and off-diagonal components, we find

$$A_d = M_d + \alpha \Delta t K_d \tag{4.5.13}$$

and

$$A_o = M_o + \alpha \Delta t K_o \,. \tag{4.5.14}$$

Thus, ϵ is given by the ratio

$$\epsilon = \frac{M_o + \alpha \Delta t K_o}{M_d + \alpha \Delta t K_d} \,. \tag{4.5.15}$$

In certain cases, the results of [Emery 79,82], [Gresho 79] show that a more accurate solution of the transient equations may be achieved through the use of a *lumped mass* matrix. This technique lumps all the mass associated with a given equation on the diagonal, thus producing a diagonal mass matrix. With a lumped mass matrix, or $M_o = 0$, we find

$$\epsilon = \frac{\alpha \Delta t K_o}{M_d + \alpha \Delta t K_d} \,. \tag{4.5.16}$$

Looking at the limiting values of ϵ as a function of time-step size, we find

$$\Delta t \to 0 \;\Rightarrow\; \epsilon \to 0 \tag{4.5.17}$$

and

$$\Delta t \to \infty \;\Rightarrow\; \epsilon \to \frac{K_o}{K_d} \leq 1 \,. \tag{4.5.18}$$

To further compare the diagonal terms in $\mathbf{K}_T$ to the off-diagonal terms, it is necessary to specify the dimensionality and order of interpolation for the problem.

We first consider the mass and stiffness arising from the two-dimensional, linear-interpolation, four-node, quadrilateral heat conduction element. If we restrict ourselves to a rectangular mesh aligned with the coordinate axes, the consistent element mass matrix $\mathbf{m}^e$ is

$$\mathbf{m}^e = \rho C_p \frac{lw}{4} \begin{bmatrix} \frac{4}{9} & \frac{2}{9} & \frac{1}{9} & \frac{2}{9} \\ \frac{2}{9} & \frac{4}{9} & \frac{2}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{2}{9} & \frac{4}{9} & \frac{2}{9} \\ \frac{2}{9} & \frac{1}{9} & \frac{2}{9} & \frac{4}{9} \end{bmatrix}. \tag{4.5.19}$$

The corresponding lumped version is

$$\mathbf{m}^e = \rho C_p \frac{lw}{4} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.5.20}$$

The element stiffness $\mathbf{k}^e$ is

$$\mathbf{k}^e = k \begin{bmatrix} +\frac{l}{3w} + \frac{w}{3l} & +\frac{l}{6w} - \frac{w}{3l} & -\frac{l}{6w} - \frac{w}{6l} & -\frac{l}{3w} + \frac{w}{6l} \\ +\frac{l}{6w} - \frac{w}{3l} & +\frac{l}{3w} + \frac{w}{3l} & -\frac{l}{3w} + \frac{w}{6l} & -\frac{l}{6w} - \frac{w}{6l} \\ -\frac{l}{6w} - \frac{w}{6l} & -\frac{l}{3w} + \frac{w}{6l} & +\frac{l}{3w} + \frac{w}{3l} & +\frac{l}{6w} - \frac{w}{3l} \\ -\frac{l}{3w} + \frac{w}{6l} & -\frac{l}{6w} - \frac{w}{6l} & +\frac{l}{6w} - \frac{w}{3l} & +\frac{l}{3w} + \frac{w}{3l} \end{bmatrix}. \tag{4.5.21}$$

In (4.5.19)—(4.5.21) , l is the length and w the width of the element. Exact quadrature was used for the element-level integration.

The size of the diagonal and off-diagonal terms in the assembled global matrices $\mathbf{M}$ and $\mathbf{K}_T$ depends on the number and size of the elements contributing to the term. In computing an upper bound for ϵ, a "safe" estimate of the size of the diagonal terms

in the global matrix **A** is the minimum diagonal term, thus

$$A_d = \min_{a=1}^{N_{ELEQ}} (n_{aa} a^e_{aa}) . \tag{4.5.22}$$

For the off-diagonal terms, the safe estimate is the magnitude of the maximum off-diagonal term, or

$$A_o = \max_{\substack{a,b=1 \\ a \neq b}}^{N_{ELEQ}} (n_{ab} |a^e_{ab}|) , \tag{4.5.23}$$

where n_{ab} is the number of elements contributing to the term A_{AB}[2] in the global matrix. For the two-dimensional, four-node, quadrilateral element, n_{ab} is

$$[n_{ab}] = \begin{bmatrix} 4 & 2 & 1 & 2 \\ 2 & 4 & 2 & 1 \\ 1 & 2 & 4 & 2 \\ 2 & 1 & 2 & 4 \end{bmatrix} . \tag{4.5.24}$$

We may compute the safe estimates for the terms in the consistent mass matrix as

$$M_d = \rho C_p l w \frac{4}{9} , \tag{4.5.25}$$

and

$$\begin{aligned} M_o &= \max\{ \rho C_p l w \frac{1}{9}, \ \rho C_p l w \frac{1}{36} \} \\ &= \rho C_p l w \frac{1}{9} , \end{aligned} \tag{4.5.26}$$

[2] A and B are the global equation numbers corresponding to the local numbers a and b respectively.

while the safe estimates for the terms of the lumped mass matrix are

$$M_d = \rho C_p l w \,, \tag{4.5.27}$$

and

$$M_o = 0 \,. \tag{4.5.28}$$

Similarly, we compute the safe estimate for the diagonal of the global stiffness as

$$K_d = k\frac{4}{3}(\frac{l}{w} + \frac{w}{l}) \,, \tag{4.5.29}$$

and for the off-diagonal as

$$K_o = \max\{\, 2k|\frac{l}{6w} - \frac{w}{3l}|,\ k(\frac{l}{6w} + \frac{w}{6l}),\ 2k|\frac{-l}{3w} + \frac{w}{6l}|\,\} \,. \tag{4.5.30}$$

If we further assume that the element aspect ratio satisfies

$$\frac{1}{\sqrt{2}} w \le l \le \sqrt{2} w \,, \tag{4.5.31}$$

then we may let

$$K_o = k(\frac{l}{6w} + \frac{w}{6l}) \,. \tag{4.5.32}$$

Having computed the diagonal and off-diagonal terms in **M** and **K**, we now compute ϵ for consistent mass as

$$\epsilon = \frac{\rho C_p l w \frac{1}{9} + \alpha \Delta t k \frac{1}{6}\left(\frac{l}{w} + \frac{w}{l}\right)}{\rho C_p l w \frac{4}{9} + \alpha \Delta t k \frac{4}{3}\left(\frac{l}{w} + \frac{w}{l}\right)} \,, \tag{4.5.33}$$

which has the limiting behavior

$$\Delta t \to 0 \;\Rightarrow\; \epsilon \to \frac{1}{4} \tag{4.5.34}$$

and

$$\Delta t \to \infty \;\Rightarrow\; \epsilon \to \frac{1}{8}. \tag{4.5.35}$$

Similarly, the ϵ for lumped mass is

$$\epsilon = \frac{\alpha \Delta t k \frac{1}{6}\left(\frac{l}{w} + \frac{w}{l}\right)}{\rho C_p l w + \alpha \Delta t k \frac{4}{3}\left(\frac{l}{w} + \frac{w}{l}\right)}, \tag{4.5.36}$$

which has the limiting behavior

$$\Delta t \to 0 \;\Rightarrow\; \epsilon \to 0 \tag{4.5.37}$$

and

$$\Delta t \to \infty \;\Rightarrow\; \epsilon \to \frac{1}{8}. \tag{4.5.38}$$

A similar analysis for the one-dimensional, linear-interpolation, two-node "rod" element of length l with unit area and lumped mass yields

$$\epsilon = \frac{\alpha \Delta t k}{\rho C_p l + 2\alpha \Delta t k}, \tag{4.5.39}$$

which has the limiting behavior

$$\Delta t \to 0 \;\Rightarrow\; \epsilon \to 0 \tag{4.5.40}$$

and

$$\Delta t \to \infty \;\Rightarrow\; \epsilon \to \frac{1}{2}. \tag{4.5.41}$$

For a three-dimensional, linear-interpolation, eight-node "brick" element with edge length l and lumped mass we find

$$\epsilon = \frac{\frac{1}{3}\alpha\Delta t k}{\rho C_p l^3 + \frac{16}{3}\alpha\Delta t k}\,, \tag{4.5.42}$$

which has the limiting behavior

$$\Delta t \to 0 \;\Rightarrow\; \epsilon \to 0 \tag{4.5.43}$$

and

$$\Delta t \to \infty \;\Rightarrow\; \epsilon \to \frac{1}{16}\,. \tag{4.5.44}$$

The size of ϵ measures the error in the approximate factorization. From the above analysis, we see that in all cases $\epsilon \leq 1/2$. For lumped mass, the actual value of ϵ may be even less if small time steps are used.

Chapter 5

Implementational Aspects

This chapter explores a variety of implementational topics. It first describes subiteration algorithms; these are combinations of EL×EL approximate factorizations and iterative algorithms that we have found successful in the solution of the linear equation system arising from the nonlinear iterative solution scheme. Next, the convergence measures used to terminate an iterative scheme are considered. Based on an analysis of the effect of error in the subiteration loop on the convergence of the iteration loop, optimal convergence criteria for the subiteration loop are developed. This leads to an investigation of the control of error in the time integration algorithm. Next, the use of substructuring to reduce computational costs is considered. Some of the additional benefits of the EL×EL solution algorithms are then briefly outlined. Finally, the modifications required to implement EL×EL algorithms on parallel processor machines are considered.

§5.1 Subiteration Algorithms

In Chapter 3, we described a number of iterative algorithms for the solution of linear equation systems and in Chapter 4, we developed EL×EL approximate factorizations. We now consider combinations of these algorithms for solving the linear equation system arising in the nonlinear iterative solution algorithm. We call the composite

schemes ***subiteration*** or ***inner-iteration*** algorithms, since they are iterative algorithms used "below" or "inside" the nonlinear iteration level [Hageman 81] [Wachpress 66]. We assume that the **A** matrix is symmetric positive-definite, although most of what is presented in this and in the following sections may be generalized to the nonsymmetric case.

5.1.1 Definitions

A subiteration scheme is defined by the triplet $\langle I, \tilde{\mathbf{A}}, E \rangle$, where I is an iterative algorithm, $\tilde{\mathbf{A}}$ is the matrix approximated by the EL×EL factorization, and E is an EL×EL approximate factorization algorithm. We limit our discussion to the following iterative algorithms :

1. Preconditioned Richardson's method (PRF),
2. Preconditioned steepest descent (PSD),
3. Preconditioned conjugate gradient (PCG).

The choice of $\tilde{\mathbf{A}}$ may be either :

1. Parabolic Regularization ($\tilde{\mathbf{A}}_{PR}$),

$$\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{D}_{\sigma}(\mathbf{A}) \, ; \tag{5.1.1}$$

2. Optimal ($\tilde{\mathbf{A}}_{OPT}$),

$$\tilde{\mathbf{A}} = \mathbf{A} \, . \tag{5.1.2}$$

Finally, the EL×EL approximate factorization E is itself defined in terms of the triplet $\langle n, O, F \rangle$, where n is the number of passes; O is the ordering, natural ($\mathcal{N}$), or reordered ($\mathcal{R}$); and F is the symmetric factorization. F may be

1. Crout (π) :

$$\mathbf{C} = \prod_{e=1}^{N_{EL}} \mathbf{L}_\pi(\mathbf{I} + \epsilon\overline{\mathbf{A}}_e)\mathbf{D}_\pi(\mathbf{I} + \epsilon\overline{\mathbf{A}}_e)\mathbf{L}_\pi^T(\mathbf{I} + \epsilon\overline{\mathbf{A}}_e) \,, \tag{5.1.3}$$

2. Cholesky $(\tilde{\pi})$:

$$\mathbf{C} = \prod_{e=1}^{N_{EL}} \tilde{\mathbf{L}}_\pi(\mathbf{I} + \epsilon\overline{\mathbf{A}}_e)\tilde{\mathbf{L}}_\pi^T(\mathbf{I} + \epsilon\overline{\mathbf{A}}_e) \,, \tag{5.1.4}$$

3. Symmetric Gauss-Seidel $(\tilde{\sigma})$:

$$\mathbf{C} = \prod_{e=1}^{N_{EL}} \left(\mathbf{I} + \epsilon\tilde{\mathbf{L}}_\sigma(\overline{\mathbf{A}}_e)\right)\left(\mathbf{I} + \epsilon\tilde{\mathbf{L}}_\sigma^T(\overline{\mathbf{A}}_e)\right) . \tag{5.1.5}$$

A subiteration algorithm composed of a preconditioned steepest descent iterative algorithm applied to a reordered one-pass Cholesky factorization based on the optimal $\tilde{\mathbf{A}}$ is thus specified by $\langle PSD, \tilde{\mathbf{A}}_{OPT}, \langle 1, \mathcal{R}, \tilde{\pi}\rangle\rangle$.

5.1.2 Cost

Having defined the components of a subiteration algorithm, we are now prepared to compute the total storage and CPU costs associated with the algorithm. The total cost is the sum of the costs of the specified iterative scheme and the EL×EL approximate factorization.

The storage costs for the iterative algorithms considered with element-level storage for the **A** matrix are given in Table 5.1.1. Although these costs could be reduced by using a sparse matrix storage technique, the savings to be gained are outweighed by

Algorithm	Storage Cost (words)
PRF	$\frac{1}{2}N_{EL}N_{ELEQ}(N_{ELEQ}+1)+2N_{EQ}$
PSD	$\frac{1}{2}N_{EL}N_{ELEQ}(N_{ELEQ}+1)+4N_{EQ}$
PCG	$\frac{1}{2}N_{EL}N_{ELEQ}(N_{ELEQ}+1)+7N_{EQ}$

Table 5.1.1 Storage costs for iterative algorithms.

Algorithm	Storage Cost (words)
$\langle PRF,*,*\rangle$	$N_{EL}N_{ELEQ}(N_{ELEQ}+1)+4N_{EQ}$
$\langle PSD,*,*\rangle$	$N_{EL}N_{ELEQ}(N_{ELEQ}+1)+6N_{EQ}$
$\langle PCG,*,*\rangle$	$N_{EL}N_{ELEQ}(N_{ELEQ}+1)+9N_{EQ}$

*(independent of this argument)

Table 5.1.2 Total storage costs for subiteration algorithms.

the simplicity of the element-level implementation. The storage cost for all the EL×EL algorithms under consideration is

$$S_{EL-factor}(\mathbf{B}) = \frac{1}{2}N_{EL}N_{ELEQ}(N_{ELEQ}+1)+N_{EQ} \text{ words}. \qquad (5.1.6)$$

The total storage cost for the subiterative algorithms is independent of the exact type of EL×EL factorization used. The totals are given in Table 5.1.2.

As indicated in the discussion of the costs of iterative algorithms in Chapter 3, the CPU cost is composed of two parts, an initial or factorization cost and a cost associated with each iteration. The CPU costs for the iterative algorithms considered are given in Table 5.1.3. The CPU costs for the EL×EL approximate factorizations under consideration are given in Table 5.1.4. Combining the CPU costs of the iterative and EL×EL algorithms results in the total CPU costs of the algorithm. These are shown in Table 5.1.5.

To get a feeling for the costs of a subiteration algorithm, reconsider the three-

Algorithm	Initial Cost (ops)	Cost per Iteration (ops)
PRF	0	$N_{EL}N_{ELEQ}^2$
PSD	0	$N_{EL}N_{ELEQ}^2 + 3N_{EQ}$
PCG	0	$N_{EL}N_{ELEQ}^2 + 5N_{EQ}$

Table 5.1.3 CPU costs for iterative algorithms.

Algorithm	Initial Cost (ops)	Cost per Iteration (ops)
$\langle 1, \mathcal{R}, \tilde{\sigma} \rangle$	$N_{EL}N_{ELEQ}$	$N_{EL}N_{ELEQ}^2 + 2N_{EQ}$
$\langle 1, \mathcal{R}, \tilde{\pi} \rangle$	$\frac{1}{6}N_{EL}N_{ELEQ}(N_{ELEQ}-1)(N_{ELEQ}+4)$	$N_{EL}N_{ELEQ}^2 + 2N_{EQ}$
$\langle 1, \mathcal{R}, \pi \rangle$	$\frac{1}{6}N_{EL}N_{ELEQ}(N_{ELEQ}-1)(N_{ELEQ}+4)$	$N_{EL}N_{ELEQ}^2 + 2N_{EQ}$
$\langle 2, \mathcal{N}, \pi \rangle$	$\frac{1}{6}N_{EL}N_{ELEQ}(N_{ELEQ}-1)(N_{ELEQ}+4)$	$2N_{EL}N_{ELEQ}^2 + 2N_{EQ}$

Table 5.1.4 CPU costs for EL×EL algorithms.

Algorithm	Initial Cost (ops)	Cost per Iteration (ops)
$\langle PRF, *, \langle 2, \mathcal{N}, \pi \rangle \rangle$	$\frac{1}{6}N_{EL}N_{ELEQ}(N_{ELEQ}-1)(N_{ELEQ}+4)$	$3N_{EL}N_{ELEQ}^2 + 2N_{EQ}$
$\langle PSD, *, \langle 1, \mathcal{R}, \tilde{\sigma} \rangle \rangle$	$N_{EL}N_{ELEQ}$	$2N_{EL}N_{ELEQ}^2 + 2N_{EQ}$
$\langle PSD, *, \langle 1, \mathcal{R}, \tilde{\pi} \rangle \rangle$	$\frac{1}{6}N_{EL}N_{ELEQ}(N_{ELEQ}-1)(N_{ELEQ}+4)$	$2N_{EL}N_{ELEQ}^2 + 5N_{EQ}$
$\langle PSD, *, \langle 1, \mathcal{R}, \pi \rangle \rangle$	$\frac{1}{6}N_{EL}N_{ELEQ}(N_{ELEQ}-1)(N_{ELEQ}+4)$	$2N_{EL}N_{ELEQ}^2 + 5N_{EQ}$
$\langle PSD, *, \langle 2, \mathcal{N}, \pi \rangle \rangle$	$\frac{1}{6}N_{EL}N_{ELEQ}(N_{ELEQ}-1)(N_{ELEQ}+4)$	$3N_{EL}N_{ELEQ}^2 + 5N_{EQ}$
$\langle PCG, *, \langle 1, \mathcal{R}, \tilde{\sigma} \rangle \rangle$	$N_{EL}N_{ELEQ}$	$2N_{EL}N_{ELEQ}^2 + 7N_{EQ}$
$\langle PCG, *, \langle 1, \mathcal{R}, \tilde{\pi} \rangle \rangle$	$\frac{1}{6}N_{EL}N_{ELEQ}(N_{ELEQ}-1)(N_{ELEQ}+4)$	$2N_{EL}N_{ELEQ}^2 + 7N_{EQ}$
$\langle PCG, *, \langle 1, \mathcal{R}, \pi \rangle \rangle$	$\frac{1}{6}N_{EL}N_{ELEQ}(N_{ELEQ}-1)(N_{ELEQ}+4)$	$2N_{EL}N_{ELEQ}^2 + 7N_{EQ}$
$\langle PCG, *, \langle 2, \mathcal{N}, \pi \rangle \rangle$	$\frac{1}{6}N_{EL}N_{ELEQ}(N_{ELEQ}-1)(N_{ELEQ}+4)$	$3N_{EL}N_{ELEQ}^2 + 7N_{EQ}$

*(independent of this argument)

Table 5.1.5 Total CPU costs for subiteration algorithms.

dimensional model problem with $p \times q \times r$ regular mesh given in Chapter 3. Further, assume that $p = q = r$, and that the elements are eight-node bricks. Thus, the number

Algorithm	Storage Cost (words)
$\langle PRF, *, * \rangle$	$72(p-1)^3 + 3p^3$
$\langle PSD, *, * \rangle$	$72(p-1)^3 + 5p^3$
$\langle PCG, *, * \rangle$	$72(p-1)^3 + 8p^3$
"direct"	$p^5 + p^3$

*(independent of this argument)

Table 5.1.6

Total storage costs for subiteration algorithms applied to the three-dimensional $p \times p \times p$ model problem.

of elements is

$$N_{EL} = (p-1)^3 , \tag{5.1.7}$$

the number of global equations is

$$N_{EQ} = p^3 , \tag{5.1.8}$$

and the number of element equations is

$$N_{ELEQ} = 8 . \tag{5.1.9}$$

The total storage costs for the subiterative algorithms applied to the model problem are given in Table 5.1.6. The CPU costs for the subiteration algorithms applied to the model problem are given in Table 5.1.7. Note, in particular, that the cost per iteration of the subiterative algorithm is actually higher than the initial cost of factorization. This suggests that for some nonlinear problems it may be advantageous to use a full Newton-Raphson iteration scheme. One would assume that faster iteration-level convergence would be obtained even though there would be the additional expense of more element stiffness formations.

Algorithm	Initial Cost (ops)	Cost per Iteration (ops)
$\langle PRF, *, \langle 2, \mathcal{N}, \pi \rangle\rangle$	$112(p-1)^3$	$192(p-1)^3 + 2p^3$
$\langle PSD, *, \langle 1, \mathcal{R}, \tilde{\sigma} \rangle\rangle$	$8(p-1)^3$	$128(p-1)^3 + 2p^3$
$\langle PSD, *, \langle 1, \mathcal{R}, \tilde{\pi} \rangle\rangle$	$112(p-1)^3$	$128(p-1)^3 + 5p^3$
$\langle PSD, *, \langle 1, \mathcal{R}, \pi \rangle\rangle$	$112(p-1)^3$	$128(p-1)^3 + 5p^3$
$\langle PSD, *, \langle 2, \mathcal{N}, \pi \rangle\rangle$	$112(p-1)^3$	$192(p-1)^3 + 5p^3$
$\langle PCG, *, \langle 1, \mathcal{R}, \tilde{\sigma} \rangle\rangle$	$8(p-1)^3$	$128(p-1)^3 + 7p^3$
$\langle PCG, *, \langle 1, \mathcal{R}, \tilde{\pi} \rangle\rangle$	$112(p-1)^3$	$128(p-1)^3 + 7p^3$
$\langle PCG, *, \langle 1, \mathcal{R}, \pi \rangle\rangle$	$112(p-1)^3$	$128(p-1)^3 + 7p^3$
$\langle PCG, *, \langle 2, \mathcal{N}, \pi \rangle\rangle$	$112(p-1)^3$	$192(p-1)^3 + 7p^3$
"direct"	$\frac{1}{2}p^7$	$2p^{5(\dagger)}$

*(independent of this argument)

(†)(only one iteration required)

Table 5.1.7

Total CPU costs for subiteration algorithms applied to the three-dimensional $p \times p \times p$ model problem.

Finally, comparing the costs for a given subiteration algorithm to those of direct solution techniques, we can compute a ***break-even point*** beyond which the subiteration algorithm is more efficient. The break-even points for the model problem, solved using the $\langle PCG, \tilde{\mathbf{A}}_{OPT}, \langle 1, \mathcal{R}, \pi \rangle\rangle$ subiteration algorithm, are listed in Table 5.1.8. Note that although the break-even point for "solution" may be larger than that for "factorization" for large problems, $p \gg 1$, the direct factorization cost will dominate.

We can make a more meaningful comparison if we assume an average number of nonlinear iterations per factorization, α, and compare the combined costs of both factorization and solution for direct and subiterative techniques. For full Newton-Raphson iteration, $\alpha = 1$, while for modified Newton-Raphson, a typical value is $\alpha = 10$. Finally, for a linear time-dependent problem, $\alpha = N_{STEPS}$, where N_{STEPS} is the number of time steps and therefore the total number of solutions.

Cost Component	Break-Even Point
Storage	$p \geq 7$
Factor CPU	$p \geq 3$
Solve CPU	$p \geq 8\sqrt{i}^{(\dagger)}$

$^{(\dagger)}$(i is the number of iterations)

Table 5.1.8

Break-even points for the $\langle PCG, \tilde{\mathbf{A}}_{OPT}, \langle 1, \mathcal{R}, \pi \rangle \rangle$ subiteration algorithm applied to the three-dimensional $p \times p \times p$ model problem.

In comparing the overall costs between direct and subiterative techniques, it is useful to define a ***cost ratio*** $\overline{C}$ as

$$\overline{C} = \frac{\text{subiteration cost}}{\text{direct solution cost}}, \tag{5.1.10}$$

where the costs may be associated with storage or CPU. We restrict our discussion to the $\langle PCG, \tilde{\mathbf{A}}_{OPT}, \langle 1, \mathcal{R}, \pi \rangle \rangle$ subiteration algorithm applied to the three-dimensional $p \times p \times p$ model problem. The storage cost ratio is

$$\overline{C}_S = \frac{72(p-1)^3 + 8p^3}{p^5 + p^3} = O\left(p^{-2}\right). \tag{5.1.11}$$

An example of the storage savings for the EL×EL solution algorithm applied to the model problem is given in Table 5.1.9.

The CPU cost ratio is

$$\overline{C}_C = \frac{\frac{1}{\alpha}\left[112(p-1)^3\right] + i\left[128(p-1)^3 + 7p^3\right]}{\frac{1}{\alpha}\left[\frac{1}{2}p^7\right] + [2p^5]}, \tag{5.1.12}$$

where i is the number of iterations required for the subiteration scheme to converge. Unfortunately, i is a problem-dependent parameter, and no *a priori* estimate is available.

p	S(Direct) (words)	S(EL×EL) (words)	$\overline{C}_S$
4	1.09×10^3	2.46×10^3	2.30
10	1.01×10^5	6.05×10^4	0.60
25	9.78×10^6	1.12×10^6	0.11
50	3.13×10^8	9.47×10^6	3.03×10^{-2}
100	1.00×10^{10}	7.79×10^7	7.79×10^{-3}
250	9.77×10^{11}	1.24×10^9	1.27×10^{-3}
1000	1.00×10^{15}	7.98×10^{10}	7.98×10^{-5}

Table 5.1.9

Example of storage cost ratio for the $\langle PCG, \tilde{\mathbf{A}}_{OPT}, \langle 1, \mathcal{R}, \pi \rangle \rangle$ subiteration algorithm applied to the three-dimensional $p \times p \times p$ model problem.

A worst-case estimate based on theoretical considerations yields $i \leq N_{EQ} = p^3$, however, experience indicates $i \ll N_{EQ}$ is typical. We may assume $i = O(p)$ for many cases based on the worst-case information propagation time between elements, which is $\sqrt{p^2 + p^2 + p^2} = \sqrt{3}p$ for the model problem. Further research is necessary to confirm this estimate. The parameter α is also problem-dependent, and may be weakly related to problem size. Thus, to continue with the CPU cost ratio calculation, it is necessary to consider the dependence of α and i on p. The results of substituting $\alpha = O\left(p^k\right)$ for $k = 0, 1, 2, 3$ and $i = O\left(p^j\right)$ for $j = 0, 1, 2, 3$ into (5.1.12) are given in Table 5.1.10. Any entry of size $O\left(p^{-1}\right)$ or smaller indicates that the EL×EL subiteration algorithm has the potential for CPU cost savings as the problem size p increases. In particular, the upper left-hand quadrant of the table indicates that there are substantial savings to be gained in the solution of strongly nonlinear problems using a full or modified Newton-Raphson method coupled with a quickly convergent subiteration scheme. Examples of CPU cost ratios for the model problem are given in Table 5.1.11.

In addition, even when the CPU cost ratio is $O(1)$, the storage cost ratio remains

	$i = O(1)$	$i = O(p)$	$i = O(p^2)$	$i = O(p^3)$
$\alpha = O(1)$	$O(p^{-4})$	$O(p^{-3})$	$O(p^{-2})$	$O(p^{-1})$
$\alpha = O(p^1)$	$O(p^{-3})$	$O(p^{-2})$	$O(p^{-1})$	$O(1)$
$\alpha = O(p^2)$	$O(p^{-2})$	$O(p^{-1})$	$O(1)$	$O(p)$
$\alpha = O(p^3)$	$O(p^{-2})$	$O(p^{-1})$	$O(1)$	$O(p)$

Table 5.1.10

CPU cost ratio for the $\langle PCG, \tilde{\mathbf{A}}_{OPT}, \langle 1, \mathcal{R}, \pi \rangle \rangle$ subiteration algorithm applied to the three-dimensional $p \times p \times p$ model problem.

p	C(Direct) (ops)	C(EL×EL)$^{(\dagger)}$ (ops)	$\overline{C}_C{}^{(\dagger)}$	$\overline{C}_C{}^{(\ddagger)}$ $(i = p)$
4	1.02×10^4	$3.90 \times 10^3(i+1)$	$0.38(i+1)$	1.80
10	5.20×10^6	$1.00 \times 10^5(i+1)$	$1.93 \times 10^{-2}(i+1)$	0.21
25	3.07×10^9	$1.88 \times 10^6(i+1)$	$6.12 \times 10^{-4}(i+1)$	1.58×10^{-2}
50	3.91×10^{11}	$1.59 \times 10^7(i+1)$	$4.07 \times 10^{-5}(i+1)$	2.07×10^{-3}
100	5.00×10^{13}	$1.31 \times 10^8(i+1)$	$2.62 \times 10^{-6}(i+1)$	2.64×10^{-4}
250	3.05×10^{16}	$2.09 \times 10^9(i+1)$	$6.83 \times 10^{-8}(i+1)$	1.71×10^{-5}
1000	5.00×10^{20}	$1.35 \times 10^{11}(i+1)$	$2.69 \times 10^{-10}(i+1)$	2.69×10^{-7}

$^{(\dagger)}$(upper bound on)

$^{(\ddagger)}$(computed exactly)

Table 5.1.11

Example of CPU cost ratio for $\langle PCG, \tilde{\mathbf{A}}_{OPT}, \langle 1, \mathcal{R}, \pi \rangle \rangle$ subiteration algorithm applied to three-dimensional $p \times p \times p$ model problem.

$O(p^{-2})$. As the problem size is increased, we reach a point where we can no longer afford to store the factorized global matrix and the EL×EL solution algorithm is attractive. We consider further the notion of convergence for subiteration schemes, and also the value for i, in the next two sections.

§5.2 Convergence Measures

This section develops and assesses measures of convergence for both iteration and subiteration schemes [Dahlquist 74], [Bathe 80]. We use the convergence measures to control the termination of the iterative/subiterative scheme employed. We would like an accurate measure of the "closeness" of the current iterative solution to the unknown exact solution. Consider the sequence

$$\mathbf{x}_i \quad i = 1, 2, \ldots , \tag{5.2.1}$$

where

$$\mathbf{x}_i = \mathbf{x}_{i-1} + \Delta\mathbf{x}_i , \tag{5.2.2}$$

and

$$\mathbf{x}_0 = \mathbf{0} . \tag{5.2.3}$$

We assume that the sequence converges to the exact solution $\mathbf{x}$. However, we can only compute a finite number of terms in the sequence and would like to assess the accuracy of the term $\mathbf{x}_i$. For this pupose, we introduce two notions of *accuracy*, or *error measures.* They are :

i. Absolute error

$$\|\mathbf{x} - \mathbf{x}_i\| \leq \epsilon_{\text{ABS}} , \tag{5.2.4}$$

ii. Relative error

$$\frac{\|\mathbf{x} - \mathbf{x}_i\|}{\|\mathbf{x}\|} \leq \epsilon_{\text{REL}} . \tag{5.2.5}$$

We consider only the second error measure, as it leads to the ordinary accuracy measure of "N digits accurate."

To compute a bound on the error of the term $\mathbf{x}_i$, we first consider the convergence rate of the sequence. We use the traditional measure of the *rate of convergence*, and define

$$q_i = \frac{\|\Delta \mathbf{x}_i\|}{\|\Delta \mathbf{x}_{i-1}\|} \quad i = 2, 3, \ldots . \tag{5.2.6}$$

In the following analysis, we assume that the sequence converges monotonically and that the convergence rate satisfies

$$1 \geq q_2 \geq q_3 \geq \cdots \geq q_i \geq 0 . \tag{5.2.7}$$

Although (5.2.7) may be violated in practice, such a violation is easily detected, and as such poses no threat to the use of the following calculations in the assessment of solution accuracy.

Assuming that the sequence $\mathbf{x}_i$ converges to the exact solution $\mathbf{x}$, we write

$$\mathbf{x} = \mathbf{x}_i + \Delta \mathbf{x}_{i+1} + \Delta \mathbf{x}_{i+2} + \cdots . \tag{5.2.8}$$

Rearranging terms,

$$\mathbf{x} - \mathbf{x}_i = \Delta \mathbf{x}_{i+1} + \Delta \mathbf{x}_{i+2} + \cdots , \tag{5.2.9}$$

and applying the triangle inequality leads to the following upper bound on the absolute error :

$$\|\mathbf{x} - \mathbf{x}_i\| \leq \|\Delta \mathbf{x}_{i+1}\| + \|\Delta \mathbf{x}_{i+2}\| + \cdots . \tag{5.2.10}$$

The definition of q_i and (5.2.7) allows computation of an upper bound on $\|\Delta \mathbf{x}_{i+1}\|$ as

$$\begin{aligned} \|\Delta \mathbf{x}_{i+1}\| &= \frac{\|\Delta \mathbf{x}_{i+1}\|}{\|\Delta \mathbf{x}_i\|} \|\Delta \mathbf{x}_i\| \\ &= q_{i+1} \|\Delta \mathbf{x}_i\| \\ &\leq q_i \|\Delta \mathbf{x}_i\| , \end{aligned} \tag{5.2.11}$$

Similarly, an upper bound on $\|\mathbf{x}_{i+2}\|$ is

$$\begin{aligned} \|\Delta\mathbf{x}_{i+2}\| &\leq q_{i+1}\|\Delta\mathbf{x}_{i+1}\| \\ &\leq q_{i+1}q_i\|\Delta\mathbf{x}_i\| \\ &\leq q_i^2\|\Delta\mathbf{x}_i\| , \end{aligned} \tag{5.2.12}$$

or more generally

$$\|\Delta\mathbf{x}_{i+k}\| \leq q_i^k\|\Delta\mathbf{x}_i\| . \tag{5.2.13}$$

Substituting (5.2.13) into (5.2.10) allows computation of an upper bound on the absolute error based on known quantities at iteration i,

$$\begin{aligned} \|\mathbf{x}-\mathbf{x}_i\| &\leq (q_i + q_i^2 + q_i^3 + \cdots)\|\Delta\mathbf{x}_i\| \\ &\leq \frac{q_i}{1-q_i}\|\Delta\mathbf{x}_i\| . \end{aligned} \tag{5.2.14}$$

If we assume that $\|\mathbf{x}_i\|$ is a good approximation to $\|\mathbf{x}\|$, a reasonable relative error measure e_i is

$$\frac{\|\mathbf{x}-\mathbf{x}_i\|}{\|\mathbf{x}\|} \approx \frac{\|\mathbf{x}-\mathbf{x}_i\|}{\|\mathbf{x}_i\|} \leq \frac{q_i}{1-q_i}\frac{\|\Delta\mathbf{x}_i\|}{\|\mathbf{x}_i\|} = e_i \leq \epsilon \approx \epsilon_{\text{REL}} , \tag{5.2.15}$$

where the third term in (5.2.15) is easily computed during the calculation of the sequence of solution vectors.

A stronger and potentially more accurate error measure can be computed given a lower bound on $\|\mathbf{x}\|$. If the sequence is "quickly" convergent, an appropriate lower bound can be found. Recall

$$\mathbf{x} = \mathbf{x}_i + \Delta\mathbf{x}_{i+1} + \Delta\mathbf{x}_{i+2} + \cdots . \tag{5.2.16}$$

Rearranging terms,

$$\mathbf{x} - \Delta\mathbf{x}_{i+1} - \Delta\mathbf{x}_{i+2} - \cdots = \mathbf{x}_i . \tag{5.2.17}$$

Applying the triangle inequality,

$$\|\mathbf{x}\| + \|\Delta\mathbf{x}_{i+1}\| + \|\Delta\mathbf{x}_{i+2}\| \cdots \geq \|\mathbf{x}_i\| \,, \tag{5.2.18}$$

and again rearranging terms,

$$\|\mathbf{x}\| \geq \|\mathbf{x}_i\| - \|\Delta\mathbf{x}_{i+1}\| - \|\Delta\mathbf{x}_{i+2}\| \cdots \,, \tag{5.2.19}$$

yields the lower bound

$$\|\mathbf{x}\| \geq \|\mathbf{x}_i\| - \frac{q_i}{1-q_i}\|\Delta\mathbf{x}_i\| \,. \tag{5.2.20}$$

Using (5.2.20) in the denominator of (5.2.5) yields the following more accurate error measure $\bar{e}_i$:

$$\begin{aligned} \frac{\|\mathbf{x}-\mathbf{x}_i\|}{\|\mathbf{x}\|} &\leq \frac{\|\mathbf{x}-\mathbf{x}_i\|}{\|\mathbf{x}_i\| - \frac{q_i}{1-q_i}\|\Delta\mathbf{x}_i\|} \\ &\leq \frac{\frac{q_i}{1-q_i}\|\Delta\mathbf{x}_i\|}{\|\mathbf{x}_i\| - \frac{q_i}{1-q_i}\|\Delta\mathbf{x}_i\|} \\ &\leq \frac{1}{\frac{1-q_i}{q_i}\frac{\|\mathbf{x}_i\|}{\|\Delta\mathbf{x}_i\|} - 1} = \bar{e}_i \leq \epsilon \approx \epsilon_{\text{REL}} \,. \end{aligned} \tag{5.2.21}$$

Note that in practice the lower bound for $\|\mathbf{x}\|$ given in (5.2.20) is useful only if the right-hand-side of (5.2.20) is positive. This is easily verified during actual computation.

The convergence measures e_i and $\bar{e}_i$ may be applied at the iteration level to the velocity vector $\mathbf{v}$, the out-of-balance force vector $\Delta\mathbf{F}$, and the incremental rate of work $\mathbf{v}^T\Delta\mathbf{F}$. In practice, separate error tolerances are specified for each of these terms, and the solution is considered to have converged when all error bounds have been met. Similarly, at the subiteration level they may be applied to the solution vector $\mathbf{x}$, the residual vector $\mathbf{r}$, and subiteration "work" $\mathbf{x}^T\mathbf{r}$. We further explore the use of these error measures in the determination of optimal subiteration convergence criteria in the next section.

§5.3 Optimal Subiteration Accuracy

In the previous section, we developed measures of convergence that resulted in computationally useful criteria for the termination of iterative schemes. In this section, we develop a scheme to optimize the choice of parameters governing the subiteration convergence criteria based on knowledge of the iteration convergence rate. We attempt to compute a useful upper bound for the acceptable error at the subiteration level such that the subiteration error will not adversely affect the iteration level convergence rate and error. This allows us to minimize the amount of "work" performed in obtaining the subiteration solution without increasing the amount of work required at the iteration level.

Since we are dealing with a well-behaved, positive-definite, symmetric matrix, the convergence rates and error measures of $\mathbf{v}$, $\Delta\mathbf{F}$, and $\mathbf{v}^T\Delta\mathbf{F}$ in the iteration loop are essentially equivalent. Using the notion of equivalent norms, we have the relationships among the convergence rates

$$q_i(\Delta\mathbf{F}) \approx c_1 q_i(\mathbf{v}) \tag{5.3.1}$$

and

$$q_i(\mathbf{v}^T\Delta\mathbf{F}) \approx c_2 q_i(\mathbf{v}) q_i(\Delta\mathbf{F}) , \tag{5.3.2}$$

and among the error measures

$$e_i(\Delta\mathbf{F}) \approx c_3 e_i(\mathbf{v}) \tag{5.3.3}$$

$$e_i(\mathbf{v}^T\Delta\mathbf{F}) \approx c_4 e_i(\mathbf{v}) e_i(\Delta\mathbf{F}) . \tag{5.3.4}$$

Here $q_i(x)$ is the convergence rate of x at iteration i, $e_i(x)$ is the error measure of x at iteration i, and the c_j's are constants. These relationships have been verified by our computational experience. The importance of this equivalence is that it allows an *a*

priori estimation of the convergence rate for $\mathbf{v}$ and $\mathbf{v}^T\Delta\mathbf{F}$ based on the convergence rate of $\Delta\mathbf{F}$, $q_i(\Delta\mathbf{F})$, which is computable prior to the solution of the linear equation system. Thus, we know ahead of time that if we solve the linear equation system exactly, we will pick up approximately $-\log_{10}(q_i(\Delta\mathbf{F}))$ significant digits in iteration i.

As an example, consider a problem in which the nonlinear iterative convergence rate is $q_i(\Delta\mathbf{F}) = 0.1$, i.e. 1 digit per iteration. It makes little sense to solve the linear equation system to 10-digit accuracy. In fact, if the equation system is well-behaved, there seems little point in solving the equation system to more than 1-digit accuracy, for any additional digits will be lost in the error at the nonlinear iteration level.

This notion may be formalized by analyzing the effect of error at the subiteration level on convergence rate and error at the iteration level. Assume that we have exactly solved the iterative linear equation system through iteration $i-1$. Thus, we have calculated $\mathbf{v}_{i-1}$, $\Delta\mathbf{v}_{i-1}$, etc. At this point, we can exactly solve once more for $\Delta\mathbf{v}_i$ and $\mathbf{v}_i$, which have a convergence rate q_i of

$$q_i = q_i(\mathbf{v}) = \frac{\|\Delta\mathbf{v}_i\|}{\|\Delta\mathbf{v}_{i-1}\|} \tag{5.3.5}$$

and error e_i of

$$e_i = e_i(\mathbf{v}) = \frac{q_i}{1-q_i}\frac{\|\Delta\mathbf{v}_i\|}{\|\mathbf{v}_i\|} . \tag{5.3.6}$$

Now let us consider the effect on the iteration-level convergence rate and error if we instead use an approximate solution $\Delta\tilde{\mathbf{v}}_i$ to the linear equation system which results in our perturbed or approximate i^{th} iterate $\tilde{\mathbf{v}}_i$. We assume that our approximate solution $\Delta\tilde{\mathbf{v}}_i$ satisfies the subiteration error criteria

$$\frac{\|\Delta\mathbf{v}_i - \Delta\tilde{\mathbf{v}}_i\|}{\|\Delta\tilde{\mathbf{v}}_i\|} \leq e_{S(i)} , \tag{5.3.7}$$

where $e_{S(i)}$ is the parameter governing the convergence of the subiteration scheme during iteration i. Then the convergence rate for the approximate solution $\tilde{\mathbf{v}}_i$ is

$$\tilde{q}_i = \frac{\|\Delta\tilde{\mathbf{v}}_i\|}{\|\Delta\mathbf{v}_{i-1}\|} , \tag{5.3.8}$$

and the associated error is

$$\tilde{e}_i = \frac{\tilde{q}_i}{1-\tilde{q}_i}\frac{\|\Delta\tilde{\mathbf{v}}_i\|}{\|\tilde{\mathbf{v}}_i\|} . \tag{5.3.9}$$

We would like to have upper bounds for $\tilde{q}_i$ and $\tilde{e}_i$ in terms of their exact counterparts q_i and e_i and the subiteration error criterion $e_{S(i)}$.

The first step in obtaining these relationships is the determination of a useful upper bound on $\|\Delta\tilde{\mathbf{v}}_i\|$. Rearranging terms in (5.3.7) yields

$$e_{S(i)}\|\Delta\tilde{\mathbf{v}}_i\| \geq \|\Delta\mathbf{v}_i - \Delta\tilde{\mathbf{v}}_i\| . \tag{5.3.10}$$

Squaring both sides and using the Schwartz inequality yields

$$\begin{aligned} e^2_{S(i)}\|\Delta\tilde{\mathbf{v}}_i\|^2 &\geq \|\Delta\mathbf{v}_i - \Delta\tilde{\mathbf{v}}_i\|^2 \\ &\geq \|\Delta\mathbf{v}_i\|^2 + \|\Delta\tilde{\mathbf{v}}_i\|^2 - 2\Delta\mathbf{v}_i\cdot\Delta\tilde{\mathbf{v}}_i \\ &\geq \|\Delta\mathbf{v}_i\|^2 + \|\Delta\tilde{\mathbf{v}}_i\|^2 - 2\|\Delta\mathbf{v}_i\|\|\Delta\tilde{\mathbf{v}}_i\| \\ &\geq (\|\Delta\mathbf{v}_i\| - \|\Delta\tilde{\mathbf{v}}_i\|)^2 ; \end{aligned} \tag{5.3.11}$$

thus

$$e_{S(i)}\|\Delta\tilde{\mathbf{v}}_i\| \geq |(\|\Delta\mathbf{v}_i\| - \|\Delta\tilde{\mathbf{v}}_i\|)| . \tag{5.3.12}$$

Further evaluation requires the consideration of two cases, $\|\Delta\mathbf{v}_i\| \geq \|\Delta\tilde{\mathbf{v}}_i\|$ and $\|\Delta\mathbf{v}_i\| < \|\Delta\tilde{\mathbf{v}}_i\|$. In the latter case, (5.3.12) becomes

$$e_{S(i)}\|\Delta\tilde{\mathbf{v}}_i\| \geq \|\Delta\tilde{\mathbf{v}}_i\| - \|\Delta\mathbf{v}_i\| . \tag{5.3.13}$$

Rearranging terms in (5.3.13) yields

$$\|\Delta\mathbf{v}_i\| \geq (1 - e_{S(i)})\|\Delta\tilde{\mathbf{v}}_i\| , \tag{5.3.14}$$

which gives the upper bound on $\|\Delta\tilde{\mathbf{v}}_i\|$

$$\frac{1}{1-e_{S(i)}}\|\Delta\mathbf{v}_i\| \geq \|\Delta\tilde{\mathbf{v}}_i\| \,. \tag{5.3.15}$$

Note that (5.3.15) is also trivially satisfied in the former case if $e_{S(i)} < 1$, a reasonable assumption. Thus (5.3.15) provides a useful upper bound on $\|\Delta\tilde{\mathbf{v}}_i\|$. Substituting (5.3.15) into (5.3.8) yields the upper bound on $\tilde{q}_i$,

$$\tilde{q}_i \leq \frac{1}{1-e_{S(i)}} q_i \,. \tag{5.3.16}$$

Thus we have determined an upper bound on the convergence rate of the perturbed iterative solution as a function of the accuracy with which the linear equation system was solved.

To compute an upper bound on the error in the approximate solution $\tilde{e}_i$, we must first compute a lower bound on $\|\tilde{\mathbf{v}}_i\|$ in terms of $\|\mathbf{v}_i\|$. This is accomplished by first taking the definition of $\tilde{\mathbf{v}}_i$, adding and subtracting the exact solution increment $\Delta\mathbf{v}_i$, and applying the triangle inequality to obtain

$$\begin{aligned}
\|\tilde{\mathbf{v}}_i\| &= \|\mathbf{v}_{i-1} + \Delta\tilde{\mathbf{v}}_i\| \\
&= \|\mathbf{v}_{i-1} + \Delta\mathbf{v}_i + \Delta\tilde{\mathbf{v}}_i - \Delta\mathbf{v}_i\| \\
&= \|\mathbf{v}_i + (\Delta\tilde{\mathbf{v}}_i - \Delta\mathbf{v}_i)\| \\
&\geq \|\mathbf{v}_i\| - \|\Delta\mathbf{v}_i - \Delta\tilde{\mathbf{v}}_i\| \,.
\end{aligned} \tag{5.3.17}$$

Factoring out $\|\mathbf{v}_i\|$ and using the upper bound for $\|\Delta\mathbf{v}_i - \Delta\tilde{\mathbf{v}}_i\|$ given by the definition of $e_{S(i)}$ in (5.3.7) , we obtain the bound

$$\|\tilde{\mathbf{v}}_i\| \geq (1 - e_{S(i)}\frac{\|\Delta\tilde{\mathbf{v}}_i\|}{\|\mathbf{v}_i\|})\|\mathbf{v}_i\| \,. \tag{5.3.18}$$

Having computed upper bounds for $\|\Delta\tilde{\mathbf{v}}_i\|$ and $\tilde{q}_i$ and a lower bound for $\|\tilde{\mathbf{v}}_i\|$, we can now compute an upper bound on $\tilde{e}_i$ as

$$\tilde{e}_i \leq \frac{\frac{1}{1-e_{S(i)}}q_i}{1-\frac{1}{1-e_{S(i)}}q_i}\frac{\frac{1}{1-e_{S(i)}}\|\Delta\mathbf{v}_i\|}{(1-e_{S(i)}\frac{\|\Delta\tilde{\mathbf{v}}_i\|}{\|\mathbf{v}_i\|})\|\mathbf{v}_i\|} . \tag{5.3.19}$$

Rearranging terms and using the definition of e_i, we obtain

$$\tilde{e}_i \leq \left(\frac{1}{1-e_{S(i)}}\right)^2 \frac{1-q_i}{1-\frac{1}{1-e_{S(i)}}q_i}\frac{1}{1-e_{S(i)}\frac{\|\Delta\tilde{\mathbf{v}}_i\|}{\|\mathbf{v}_i\|}}e_i . \tag{5.3.20}$$

If we assume $e_{S(i)}$ is a small parameter in (5.3.16) and (5.3.20) , then a bound on the convergence rate of the perturbed iterative solution $\tilde{q}_i$ is

$$\tilde{q}_i \leq \left[1+e_{S(i)}+O\left(e_{S(i)}^2\right)\right]q_i , \tag{5.3.21}$$

and a bound on the error $\tilde{e}_i$ is

$$\tilde{e}_i \leq \left[1+(2+\frac{q_i}{1-q_i}+\frac{\|\Delta\tilde{\mathbf{v}}_i\|}{\|\mathbf{v}_i\|})e_{S(i)}+O\left(e_{S(i)}^2\right)\right]e_i . \tag{5.3.22}$$

For a quickly convergent sequence, $q_i \ll 1$, the previous analysis indicates that the same order of accuracy in the approximate or perturbed solution can be maintained by choosing $e_{S(i)} = O(q_i)$. This choice leads to the following upper bounds for the convergence rate and error in the perturbed solution ;

$$\tilde{q}_i \leq q_i + O\left(q_i^2\right), \tag{5.3.23}$$

and

$$\tilde{e}_i \leq [1+O(q_i)]e_i . \tag{5.3.24}$$

Thus if $q_i \ll 1$ and we choose $e_{S(i)} = O(q_i)$, we maintain the same order of convergence rate and error as that of the exact solution. Note that small errors in the iterative solution caused by approximating the solution of the linear equation system do not accumulate but are corrected for in the nonlinear solution algorithm by the use of the true residual.

The preceding analysis leads to the following three rules governing the choice of subiteration error criteria :

1. If the nonlinear iteration-level convergence rate is slow, say $q_i \approx \frac{1}{2}$, then compute the subiteration-level solution with enough accuracy, say $e_{S(i)} =$ 0.1 (1 digit accuracy), to eliminate any contribution to divergence at the nonlinear iteration level.

2. If the nonlinear iteration-level convergence rate is fast, say $q_i \leq 0.1$, then compute the subiteration-level solution with enough accuracy, say $e_{S(i)} = O(q_i) \approx \frac{1}{2} q_i$, to maintain the same convergence rate and error at the nonlinear iteration level.

3. At no time should the subiteration error criteria be substantially "tighter" than the equivalent nonlinear iterative level error criteria.

In practice, we have found it advantageous to choose convergence criteria for the subiteration terms $\mathbf{x}$, $\mathbf{r}$, and $\mathbf{x}^T\mathbf{r}$ as follows :

$$e_{S(i)}(\mathbf{x}) = \max\{\, \min\{\, \alpha_1, \beta_1 q_i(\Delta\mathbf{F})\,\}, \gamma_1 e(\mathbf{v})\,\}\,, \tag{5.3.25}$$

$$e_{S(i)}(\mathbf{r}) = \max\{\, \min\{\, \alpha_2, \beta_2 q_i(\Delta\mathbf{F})\,\}, \gamma_2 e(\Delta\mathbf{F})\,\}\,, \tag{5.3.26}$$

and

$$e_{S(i)}(\mathbf{x}^T\mathbf{r}) = \max\{\, \min\{\, (\alpha_3, \beta_3 q_i(\Delta\mathbf{F})\,\})^2, \gamma_3 e(\mathbf{v}^T\Delta\mathbf{F})\,\}\,, \tag{5.3.27}$$

where the α_j's are used to implement rule one, the $\beta_j q_i$'s to implement rule two, and the γ_j's to implement rule three. Typical values are $\alpha_j = 0.1$, $\beta_j = 0.5$ and $\gamma_j = 0.9$.

There are two exceptions to the preceding analysis. The first occurs during the first iteration of a new time step. For $i = 1$, there is no available convergence rate for $\Delta\mathbf{F}$; thus the bounds as given cannot be computed. There are two easily implemented remedies for this first exception. One is the assignment of a fixed error criterion for the subiteration level on the first iteration, for example let $q_1(\Delta\mathbf{F}) = 0.1$. Alternatively, if the time step and nonlinearities are slowly varying, then the convergence rate of the second iteration from the previous time step is probably a good estimate and we can let $q_1 = q_{2(n-1)}$.

The second exception occurs when solving linear problems. For a linear problem, we have *a priori* knowledge that the iterative solution will converge in one iteration to the exact solution, if the linear equation system is solved exactly. Thus the subiteration error criterion $e_{S(1)}$ should be chosen to satisfy the time-stepping-level error criterion.

The preceding criteria minimize the computational expense of solving of the linear equation system arising in the nonlinear iterative solution loop. Note that, in principle, these same estimates could be applied to direct solution algorithms. However, at present, no hardware is available to perform efficiently the variable-precision arithmetic required for computation of the direct solution at other then full machine accuracy.

§5.4 Control of Time Step Errors

In the preceding sections we have developed termination criteria for the nonlinear iteration and linear subiteration loops. In this section we consider the measure and control of errors at the time step level. The error in the discrete time solution vectors, $\mathbf{d}_n$ and $\mathbf{v}_n$, is a function of the time step, Δt, used to obtain them. Clearly we would like to minimize this error, but at the same time keep Δt as large as possible to avoid excessive "work" in obtaining the solution for a given time interval. To accomplish this we need

an error measure for the discrete time solution vectors and a relationship between these errors and the size of Δt. Clearly the nonlinear iteration and linear subiteration error tolerances should be compatable with the accuracy required at the time step level.

Ideally, we would like an *a priori* estimate of the solution error as a function of Δt This would allow the computation of a sufficiently small Δt to assure the satisfaction of the solution error criteria [Taylor 75], [Park 79], [Underwood 79]. Unfortunately *a priori* estimates are usually only valid for linear problems.

The approach that we use, and one that is valid for nonlinear problems [Narasimhan 77], [Hibbitt 79], is an *a posteriori* estimation of the solution error. In this approach a step size is assumed and a prospective solution computed. Then using the newly computed solution an *a posteriori* estimate is made of the error in the step. If the error is unacceptable the new solution is "thrown out" and we "backup" to the previous step and try again with a reduced step size. If the solution error is acceptable, we accept the new solution and proceed with the time integration. Note that a "very small" solution error indicates that the step size should be increased to reduce the amount of "work" required to integrate the time interval under consideration.

Thus the algorithm for controlling the time step error has two parts. First, the *a posteriori* error estimate for the newly computed solution. Second, an algorithm that uses this error measure to accept or reject the solution and modify the time step accordingly. We now consider the error measures, a step selection strategy, and the effect of temporal discontinuities.

5.4.1 Time Step Error Measure

Many error measures for the discrete solution are possible. Two important attributes of the error measure chosen are that it accurately estimate the likelyhood of "true" error in the solution and that it be cheaply computable. The solution errors typically have two components: the first is due to the temporal discretization, and the

second is due to changing nonlinearities over the step. Both of these should be accounted for in the error measure chosen.

Perhaps the simplest error measure is based on the notion of percentage change in value over a time step. Considering some solution component x, we define the error (percentage change) in x, $e_{\%}(x)$ as

$$e_{\%}(x) = \frac{|\text{difference in } x \text{ over the step}|}{|\text{average of } x \text{ over the step}|} . \tag{5.4.1}$$

Applying this error measure to the velocity vector

$$\begin{aligned} e_{\%}(\mathbf{v}) &= \max_{i=1}^{N_{EQ}} e_{\%}(v_i) \\ &= \max_{i=1}^{N_{EQ}} \frac{|v_{n+1(i)} - v_{n(i)}|}{\frac{1}{2}(|v_{n+1(i)}| + |v_{n(i)}|)} . \end{aligned} \tag{5.4.2}$$

As an example of the use of this error measure we could limit the changes in velocity to 1% by specifying that $e_{\%}(\mathbf{v}) \leq 0.01$ must be satisfied. Similar error measures may be defined for $\mathbf{d}$, $\mathbf{F}$, $\mathbf{v}^T\mathbf{F}$, etc. Although these measures are physically appealing they do not accurately estimate the influence of changing nonlinearities on solution error.

To overcome this weakness, and to obtain a more accurate estimate of solution error, we introduce solution error measures based on $\frac{1}{2}$-*step residuals*. Given an accepted solution $\mathbf{d}_n$, $\mathbf{v}_n$ at time t_n and the "proposed" new solution $\mathbf{d}_{n+1}$, $\mathbf{v}_{n+1}$ at time t_{n+1} we compute the residual at step $n + \frac{1}{2}$ and use it to estimate the error at step $n + 1$.

The $\frac{1}{2}$-step solution values are based on a "consistent" interpolation of the solutions for the n and $n + 1$ steps. By consistent we mean that the velocity is defined over the step using a linear interpolation as

$$\mathbf{v}_{n+\tau} = (1 - \tau)\mathbf{v}_n + \tau\mathbf{v}_{n+1} \quad 0 \leq \tau \leq 1 , \tag{5.4.3}$$

and the temperature is defined as the trapezoidal integration of the velocity, yielding

$$\mathbf{d}_{n+\tau} = \mathbf{d}_n + \tau\Delta t(1 - \alpha)\mathbf{v}_n + \tau\Delta t\alpha\mathbf{v}_{n+\tau} \quad 0 \leq \tau \leq 1 . \tag{5.4.4}$$

Given $\mathbf{d}_{n+\tau}$ and $\mathbf{v}_{n+\tau}$ it is then possible to define the out-of-balance force at time $t_{n+\tau}$ as

$$\Delta\mathbf{F}_{n+\tau} = \mathbf{F}(\mathbf{d}_{n+\tau}, t_{n+\tau}) - \mathbf{M}(\mathbf{d}_{n+\tau}, t_{n+\tau})\mathbf{v}_{n+\tau} - \mathbf{N}(\mathbf{d}_{n+\tau}, t_{n+\tau}) \quad 0 \leq \tau \leq 1\,. \quad (5.4.5)$$

Note that $\Delta\mathbf{F}_n = \Delta\mathbf{F}_{n+1} = \mathbf{o}$ by definition. If the point-wise out-of-balance force is small throughout the step (i.e. $0 \leq \tau \leq 1$) then it is reasonable to assume that the consistently interpolated solution $\mathbf{d}_{n+\tau}$, $\mathbf{v}_{n+\tau}$ is accurate and thus that the final step values $\mathbf{d}_{n+1}$, $\mathbf{v}_{n+1}$ are themselves accurate. The effect of the out-of-balance force on the solution may be estimated by considering the point-wise error in the velocity vector given by

$$\Delta\tilde{\mathbf{v}}_{n+\tau} = \mathbf{M}^{-1}(\mathbf{d}_{n+\tau}, \mathbf{v}_{n+\tau})\Delta\mathbf{F}_{n+\tau} \quad 0 \leq \tau \leq 1\,. \quad (5.4.6)$$

We now define the relative point-wise error measure for velocity over the step as

$$e(\Delta\tilde{\mathbf{v}}_{n+\tau}) = \frac{\|\Delta\tilde{\mathbf{v}}_{n+\tau}\|}{\|\mathbf{v}_{n+1}\|} \quad 0 \leq \tau \leq 1\,. \quad (5.4.7)$$

or more conservatively as

$$e(\Delta\tilde{\mathbf{v}}_{n+\tau}) = \max_{i=1}^{N_{EQ}} \frac{|\Delta\tilde{v}_{n+\tau(i)}|}{\frac{1}{2}(|v_{n+1(i)}| + |v_{n(i)}|)} \quad 0 \leq \tau \leq 1\,. \quad (5.4.8)$$

Note that (5.4.8) avoids the possible masking of local component error that is possible using (5.4.7). Similar error measures may be defined for $\Delta\mathbf{F}_{n+\tau}$ and $\mathbf{v}_{n+\tau}^T\Delta\mathbf{F}_{n+\tau}$. Although we would like to know the maxima of these errors over the step, we find it computationally expedient to use the values computed at the $\frac{1}{2}$-step (i.e. $\tau = \frac{1}{2}$).

5.4.2 Time Step Selection Strategy

The first step in defining the time step selection strategy is the normalization of the step error measures with respect to the user specified error tolerance. We define the

normalized step errors as

$$\bar{e}_v = \frac{e(\Delta\tilde{\mathbf{v}}_{n+\frac{1}{2}})}{\epsilon_v}\,, \tag{5.4.9}$$

$$\bar{e}_f = \frac{e(\Delta\mathbf{F}_{n+\frac{1}{2}})}{\epsilon_f}\,, \tag{5.4.10}$$

$$\bar{e}_{vf} = \frac{e(\Delta\tilde{\mathbf{v}}^T_{n+\frac{1}{2}}\Delta\mathbf{F}_{n+\frac{1}{2}})}{\epsilon_{vf}}\,, \tag{5.4.11}$$

where ϵ_v, ϵ_f, and ϵ_{vf} are user specified error tolerances. We define the maximum normalized error, $\bar{e}_{max}$, as

$$\bar{e}_{max} = \max\{\,\bar{e}_v, \bar{e}_f, \bar{e}_{vf}\,\}\,. \tag{5.4.12}$$

Thus as long as $\bar{e}_{max} \leq 1$ the solution satisfies the user specified error tolerance.

Our selection of Δt as a function of $\bar{e}_{max}$ is based on the following two rules: (i) at no time should a time step be accepted if $\bar{e}_{max} > 1$, and (ii) the step size Δt should be increased until $\bar{e}_{max} = O(1)$. These rules are embodied in the following algorithm.

1. If $\bar{e}_{max} > 1$ then the solution is unacceptable. Replace Δt by $\mu\Delta t$, $\mu < 1$, reset the step growth rate ν to ν_0, and restart the time integration at the previous step.

2. If $\bar{e}_{good} < \bar{e}_{max} \leq 1$ then the solution is acceptable. Proceed with the time integration using the current Δt.

3. If $\bar{e}_{max} \leq \bar{e}_{good}$ for N successive steps then the solution is "overly" accurate. Replace Δt by $\nu\Delta t$, $\nu > 1$, and proceed with the time integration.

4. If M successive step 3's have occurred then the step size growth rate ν is not large enough. Replace ν by $\eta\nu$ and Δt by $\eta\Delta t$, $\eta > 1$, and proceed with the time integration.

5. The computed step size should always be within user specified bounds, $\Delta t_{min} \leq \Delta t \leq \Delta t_{max}$. If Δt is reduced below Δt_{min} the user should be informed, and the integration should proceed with $\Delta t = \Delta t_{min}$.

Note that the algorithm will generate an accurate solution for any value of μ less than one if the lower bound Δt_{min} is not encountered. The purpose of step 4 is to provide a variable step size growth rate which allows Δt to be increased at a fast enough rate to raise $\bar{e}_{max}$ to $O(1)$ for even the fastest decaying exponential. The cost effectiveness of the algorithm depends on the subtle interplay between increasing and decreasing step sizes. In general it is advisable to decrease the step size quickly to reduce the number of "restarts" needed to find a small enough size. The step size should only be increased slowly, however, to avoid accidently exceeding the desired error level. In practice we have found the values $\mu = 0.5$, $\bar{e}_{good} = 0.25$, $N = 2$, $\nu = 1.25$, $M = 3$, and $\eta = 1.1$ to perform well.

5.4.3 Effect of Temporal Discontinuities

We now briefly consider the effects of temporal discontinuity on the error measures, and thus the time step selection strategy. The definitions of the preceding error measures presupposed the temporal continuity of the solution vectors. The temporal continuity of the solution vectors is in turn dependent on the continuity of the parameters defining the problem. These parameters may be classified as material constants, prescribed temperature boundary conditions, heat flux boundary conditions, and external heat sources. In fact, a temporal discontinuity in any of the aforementioned parameters will in general induce a discontinuity in the out-of-balance force which will in turn cause the velocities to be temporally discontinuous. Note that these discontinuities are physically

"legitimate" and thus are due serious consideration in the implementation of automatic step selection strategies.

Neither of the error measures previously introduced can properly account for these discontinuities. In fact, both of the error measures defined generate a "large" fixed error for a time step encompassing a discontinuity, even when the step size is reduced to zero. This suggests using measures based on solution temperatures, which are always continuous, in place of the previous measures based on velocity and force. However, we would then lose the measure of error due to changing nonlinearities. Thus measures based strictly on temperature are not satisfactory.

Instead we consider two alternative solutions. The first is to allow the step selection strategy to unsuccessfully try smaller and smaller time steps before finally selecting the user specified lower limit Δt_{min}. At this point a step is taken using Δt_{min} and the resultant "error" is ignored. Note that this error is not a "true" error in the solution, but an artifact of the continuous nature of the error measures coupled with the solution discontinuity being "spread" over a finite step Δt_{min}. The time integration then proceeds normally. If the location of the discontinuity in time is not known beforehand, this strategy is the best that we can do.

We now consider a second solution which avoids the unnecessary work entailed in reducing Δt by trial and error at a known discontinuity. If the user knows of a significant temporal discontinuity at time t_D, then the following three part scheme should be used.

1. Integrate to just before the discontinuity, $t_n = t_D^-$

2. Solve for the exact solution at the discontinuity as follows :

Let

$$t_{n+1} = t_D \,, \tag{5.4.13}$$

$$\mathbf{d}_{n+1} = \mathbf{d}_n \,. \tag{5.4.14}$$

Solve for $\mathbf{v}_{n+1}$ using

$$\mathbf{M}(\mathbf{d}_{n+1}, t_{n+1})\mathbf{v}_{n+1} = \mathbf{F}(\mathbf{d}_{n+1}, t_{n+1}) - \mathbf{N}(\mathbf{d}_{n+1}, t_{n+1}) \,. \tag{5.4.15}$$

3. Continue with normal time integration.

Note that step two is equivalent to integrating a step as $\Delta t \to 0$.

§5.5 Minimal-Cost Substructuring

In this section, we consider ways of combining "basic" elements into substructures to minimize the storage and CPU costs.

The goals of substructuring are :

1. Minimize :
   - Storage cost
   - Factorization CPU cost
   - Solution CPU cost
   - Total CPU cost
   - Error in EL×EL approximate factorization
2. Maximize : "implicitness"

5.5.1 Definitions

To achieve the aforementioned goals, we define ***substructures*** as subassembles of the basic elements in the problem. The matrix contribution for substructure E, $\tilde{\mathbf{A}}_E$, is

defined

$$\tilde{\overline{\mathbf{A}}}_E = \underset{e \in \mathcal{E}_E}{\mathbf{A}} \, \mathbf{a}^e, \quad E = 1, 2, \ldots, N_{SS} \,, \tag{5.5.1}$$

where $\mathbf{A}$ is the "local" assembly operator for the equations in the substructure, $\mathcal{E}_E$ is the set of element numbers in substructure E, $\mathbf{a}^e$ is the basic element matrix for element e, and N_{SS} is the number of substructures. We will assume that an element may only belong to one substructure, thus

$$\bigcup_{E=1}^{N_{SS}} \mathcal{E}_E = \mathcal{E} = \{1, 2, \ldots, N_{EL}\} \tag{5.5.2}$$

and

$$\mathcal{E}_{E_1} \bigcap \mathcal{E}_{E_2} = \emptyset, \quad E_1 \neq E_2, \quad E_1, E_2 \in \{1, 2, \ldots, N_{SS}\} \,. \tag{5.5.3}$$

There are two limiting cases: The first is the basic EL×EL algorithm with one element per substructure; thus $\mathcal{E}_E = \{E\}$ and $N_{SS} = N_{EL}$. The second is the fully-implicit, or global, algorithm with $\mathcal{E}_1 = \mathcal{E}$ and $N_{SS} = 1$.

5.5.2 Cost

We now consider the costs associated with substructuring. In the following analysis, we ignore the "constant" costs and consider only the "element definition"-dependent costs. These are associated with the matrix $\mathbf{C}$ in the EL×EL approximate factorization. The storage costs for an EL×EL approximate factorization using substructures for elements is

$$S(\mathbf{C}) = \sum_{E=1}^{N_{SS}} S(\tilde{\overline{\mathbf{A}}}_E) \,, \tag{5.5.4}$$

where $S(\tilde{\bar{\mathbf{A}}}_E)$ is the storage cost for substructure $\tilde{\bar{\mathbf{A}}}_E$. This substructure storage cost is

$$S(\tilde{\bar{\mathbf{A}}}_E) = \bar{b}_E N_{EQ(E)} \text{ words} , \tag{5.5.5}$$

where $\bar{b}_E$ is the mean half-bandwidth for substructure E, and $N_{EQ(E)}$ is the number of equations in substructure E. Recall from Chapter 3 that the mean half-bandwidth is

$$\bar{b}_E = \frac{1}{N_{EQ(E)}} \sum_{i=1}^{N_{EQ(E)}} b_{E(i)} , \tag{5.5.6}$$

where $b_{E(i)}$ is the half-bandwidth for row i in substructure E. The total storage cost is then

$$S(\mathbf{C}) = \sum_{E=1}^{N_{SS}} \bar{b}_E N_{EQ(E)} \text{ words} . \tag{5.5.7}$$

Note that this is also the cost of storing the matrix $\mathbf{A}$ at the substructure level.

The factorization cost is

$$C_{factor}(\mathbf{C}) = \sum_{E=1}^{N_{SS}} C_{factor}(\tilde{\bar{\mathbf{A}}}_E) . \tag{5.5.8}$$

The factorization cost for substructure E, assuming product factors, is

$$C_{\pi-factor}(\tilde{\bar{\mathbf{A}}}_E) \approx \frac{1}{2}\hat{b}_E^2 N_{EQ(E)} \text{ ops} . \tag{5.5.9}$$

With the computational mean half-bandwidth $\hat{b}_E$ defined as

$$\hat{b}_E = \left[\frac{1}{N_{EQ(E)}} \sum_{i=1}^{N_{EQ(E)}} b_{E(i)}^2 \right]^{\frac{1}{2}} , \tag{5.5.10}$$

the total factorization cost is

$$C_{\pi-factor}(\mathbf{C}) \approx \frac{1}{2} \sum_{E=1}^{N_{SS}} \hat{b}_E^2 N_{EQ(E)} \text{ ops} . \tag{5.5.11}$$

Similarly, the solution costs (cost per iteration) are

$$C_{solve}(\tilde{\mathbf{A}}_E) = 2\hat{b}_E N_{EQ(E)} \text{ ops} \tag{5.5.12}$$

and

$$C_{solve}(\mathbf{C}) = 2 \sum_{E=1}^{N_{SS}} \hat{b}_E N_{EQ(E)} \text{ ops} . \tag{5.5.13}$$

Note that the solution costs are proportional to the storage costs and that the cost of forming the matrix vector product $\mathbf{Ax}$ is equal to the solution cost if the matrix $\mathbf{A}$ is stored at the substructure level.

The error in the EL×EL approximate factorization is minimized by reducing the interaction of the matrices contributing to terms ϵ^2 and higher in the product expansion. The absolute minimum error occurs when $N_{SS} = 1$ (i.e. fully implicit); this suggests minimizing N_{SS} will reduce the error.

Maximizing the "implicitness" again suggests minimizing the number of substructures, since subassemblies of adjacent elements are more implicit than individual elements. The approximate operator's implicitness is one of the factors governing the rate of information propagation through the mesh. The more implicit the operator, the faster the transfer of information through the mesh. Thus fewer iterations may be required to obtain convergence.

5.5.3 Model Problem

We now consider the decomposition into substructures of a three-dimensional model problem with $p \times q \times r$ regular mesh. Without loss of generality we assume $p \geq q \geq$

$r \geq 2$. We restrict decompositions to those producing "congruent" substructures. Two substructures are ***congruent*** if they have identical element topology or nodal interconnectivity. Congruent substructures have identical costs for storage, factorization, and solution. The total storage cost is computed as

$$\begin{aligned} S(\mathbf{C}) &= N_{SS} S(\tilde{\mathbf{A}}_E) \\ &= N_{EL} \frac{S(\tilde{\mathbf{A}}_E)}{N_{EL(E)}} \\ &= N_{EL} S_e(\tilde{\mathbf{A}}_E) , \end{aligned} \tag{5.5.14}$$

where $N_{EL(E)}$ is the number of elements per substructure and $S_e(\tilde{\mathbf{A}}_E)$ is the average storage cost per element for an array with the topology of $\tilde{\mathbf{A}}_E$. Since the number of elements N_{EL} is constant for a given problem, minimization of $S_e(\tilde{\mathbf{A}}_E)$ with respect to the topology of $\tilde{\mathbf{A}}_E$ gives the minimum total storage cost $S(\mathbf{C})$.

We further restrict our attention to congruent substructures composed of one or more disjoint[1] $\alpha \times \beta \times \gamma$ regular meshes. Without loss of generality we assume

$$\alpha \geq \beta \geq \gamma \geq 2 . \tag{5.5.15}$$

The half-bandwidth for standard nodal ordering $b_i(\alpha, \beta, \gamma)$, $1 \leq i \leq \alpha\beta\gamma$, is

$$\begin{aligned} b_i(\alpha, \beta, \gamma) =& b_{i(j,k,l)}(\alpha, \beta, \gamma) \\ =& 1 \\ &+ \begin{cases} 1, & l < \gamma \\ 0, & l = \gamma \end{cases} \\ &+ \begin{cases} \gamma, & k < \beta \\ 0, & k = \beta \end{cases} \\ &+ \begin{cases} \beta\gamma, & j < \alpha \\ 0, & j = \alpha , \end{cases} \end{aligned} \tag{5.5.16}$$

[1]Two meshes are disjoint if they contain no common degrees-of-freedom.

where the indexing function $i(j,k,l)$ is

$$i(j,k,l) = l + \gamma(k-1) + \beta\gamma(j-1) \quad \begin{cases} 1 \le j \le \alpha \\ 1 \le k \le \beta \\ 1 \le l \le \gamma\,. \end{cases} \tag{5.5.17}$$

The previous equalities allow computation of the mean half-bandwidth $\bar{b}(\alpha,\beta,\gamma)$ as

$$\begin{aligned} \bar{b} &= \frac{1}{\alpha\beta\gamma} \sum_{i=1}^{\alpha\beta\gamma} b_i \\ &= \frac{1}{\alpha\beta\gamma} \sum_{j=1}^{\alpha} \sum_{k=1}^{\beta} \sum_{l=1}^{\gamma} b_{i(j,k,l)}\,. \end{aligned} \tag{5.5.18}$$

Substituting (5.5.16) into (5.5.18) and simplifying, we find the mean half-bandwidth for an $\alpha \times \beta \times \gamma$ mesh as

$$\begin{aligned} \bar{b}(\alpha,\beta,\gamma) = \frac{1}{\alpha\beta\gamma}\Big[& (\alpha-1)\{(\beta-1)(\gamma-1)(\beta\gamma+\gamma+2) \\ & + (\beta-1)(\beta\gamma+\gamma+1) + (\gamma-1)(\beta\gamma+2) + (\beta\gamma+1)\} \\ & + \{(\beta-1)(\gamma-1)(\gamma+2) \\ & + (\beta-1)(\gamma+1) + (\gamma-1)(2) + (1)\}\Big], \end{aligned} \tag{5.5.19}$$

and, similarly, the computational mean half-bandwidth as

$$\begin{aligned} \hat{b}(\alpha,\beta,\gamma) = \Bigg(\frac{1}{\alpha\beta\gamma}\Big[& (\alpha-1)\Big\{(\beta-1)(\gamma-1)(\beta\gamma+\gamma+2)^2 \\ & + (\beta-1)(\beta\gamma+\gamma+1)^2 + (\gamma-1)(\beta\gamma+2)^2 + (\beta\gamma+1)^2\Big\} \\ & + \Big\{(\beta-1)(\gamma-1)(\gamma+2)^2 \\ & + (\beta-1)(\gamma+1)^2 + (\gamma-1)(2)^2 + (1)^2\Big\}\Big]\Bigg)^{\frac{1}{2}}. \end{aligned} \tag{5.5.20}$$

The average storage for an element in an $\alpha \times \beta \times \gamma$ mesh is

$$S_e(\alpha,\beta,\gamma) = \frac{\bar{b}(\alpha,\beta,\gamma)\alpha\beta\gamma}{(\alpha-1)(\beta-1)(\gamma-1)} \text{ words} . \tag{5.5.21}$$

Expanding terms,

$$\begin{aligned} S_e(\alpha,\beta,\gamma) = & \left\{(\beta\gamma+\gamma+2)+\frac{1}{\gamma-1}(\beta\gamma+\gamma+1)\right. \\ & \left.+\frac{1}{\beta-1}(\beta\gamma+2)+\frac{1}{(\beta-1)(\gamma-1)}(\beta\gamma+1)\right\} \\ & +\frac{1}{\alpha-1}\left\{(\gamma+2)+\frac{1}{\gamma-1}(\gamma+1)\right. \\ & \left.+\frac{2}{\beta-1}+\frac{1}{(\beta-1)(\gamma-1)}\right\} \text{ words} . \end{aligned} \tag{5.5.22}$$

To minimize $S_e(\alpha,\beta,\gamma)$ for $\alpha \geq \beta \geq \gamma \geq 2$, α,β,γ integers, we first compute $S_e(\alpha,\beta,\gamma)$ for $O(\alpha) \geq O(\beta) \geq O(\gamma) \geq O(1)$, which yields

$$\begin{aligned} S_e(\alpha,\beta,\gamma) = & \{O(\beta\gamma)+O(\beta)+O(\gamma)+O(1)\} \\ & +O\left(\alpha^{-1}\right)\{O(\gamma)+O(1)\} . \end{aligned} \tag{5.5.23}$$

This result suggests that the absolute minimum occurs for small β and γ but large α. The values for $S_e(\alpha,\beta,\gamma)$ for $6 \geq \beta \geq \gamma \geq 2$ are listed in Table 5.5.1. We see that the choice $\alpha \times 2 \times 2$ minimizes $S_e(\alpha,\beta,\gamma)$.

Since the substructures are composed of disjoint $\alpha \times \beta \times \gamma$ meshes, we find $S_e(\tilde{\mathbf{A}}_E) = S_e(\alpha,\beta,\gamma)$. Thus, the minimum total storage is realized by subdividing the mesh into quasi-one-dimensional substructures composed of "chains" of brick elements. The longer the chain, the less the cost per element. The minimum storage for the $p \times q \times r$ mesh is thus

$$S(\mathbf{C}) = N_{EL} S_e(p,2,2) . \tag{5.5.24}$$

$\alpha \times \beta \times \gamma$	$S_e(\alpha, \beta, \gamma)$ (words)	$S_e(\alpha, \beta, \gamma)/S_e(2, 2, 2)$
$2 \times 2 \times 2$	$26.00 + 10.00$	1
$\alpha \times 2 \times 2$	$26.00 + 10.00/(\alpha - 1)$	$0.7222 + 0.2778/(\alpha - 1)$
$\alpha \times 3 \times 2$	$26.50 + 8.500/(\alpha - 1)$	$0.7361 + 0.2361/(\alpha - 1)$
$\alpha \times 3 \times 3$	$28.50 + 8.250/(\alpha - 1)$	$0.7917 + 0.2292/(\alpha - 1)$
$\alpha \times 4 \times 2$	$29.33 + 8.000/(\alpha - 1)$	$0.8148 + 0.2222/(\alpha - 1)$
$\alpha \times 4 \times 3$	$31.83 + 7.833/(\alpha - 1)$	$0.8843 + 0.2176/(\alpha - 1)$
$\alpha \times 4 \times 4$	$36.89 + 8.444/(\alpha - 1)$	$1.025 + 0.2346/(\alpha - 1)$
$\alpha \times 5 \times 2$	$32.75 + 7.750/(\alpha - 1)$	$0.9097 + 0.2153/(\alpha - 1)$
$\alpha \times 5 \times 3$	$35.75 + 7.625/(\alpha - 1)$	$0.9931 + 0.2118/(\alpha - 1)$
$\alpha \times 5 \times 4$	$41.58 + 8.250/(\alpha - 1)$	$1.155 + 0.2292/(\alpha - 1)$
$\alpha \times 5 \times 5$	$48.13 + 9.063/(\alpha - 1)$	$1.337 + 0.2517/(\alpha - 1)$
$\alpha \times 6 \times 2$	$36.40 + 7.600/(\alpha - 1)$	$1.011 + 0.2111/(\alpha - 1)$
$\alpha \times 6 \times 3$	$39.90 + 7.500/(\alpha - 1)$	$1.108 + 0.2083/(\alpha - 1)$
$\alpha \times 6 \times 4$	$46.53 + 8.133/(\alpha - 1)$	$1.293 + 0.2259/(\alpha - 1)$
$\alpha \times 6 \times 5$	$53.95 + 8.950/(\alpha - 1)$	$1.499 + 0.2486/(\alpha - 1)$
$\alpha \times 6 \times 6$	$61.68 + 9.840/(\alpha - 1)$	$1.713 + 0.2733/(\alpha - 1)$

Table 5.5.1

Average storage cost per element, S_e, for an $\alpha \times \beta \times \gamma$ mesh.

The ratio of storage for this $p \times 2 \times 2$ substructuring to the storage for the basic elements is

$$\frac{S_e(p, 2, 2)}{S_e(2, 2, 2)} = \frac{13}{18} + \frac{5}{18p}, \tag{5.5.25}$$

which tends to 13/18 as p tends to infinity. It is thus possible to save up to 28% on storage for the elements with $p \times 2 \times 2$ substructuring.

Similar cost calculations may be performed for the factor and solve costs. The average product-factor cost per element $C_{\pi f - e}(\alpha, \beta, \gamma)$ is

$$C_{\pi f-e}(\alpha,\beta,\gamma) = \frac{1}{2}\frac{\hat{b}^2(\alpha,\beta,\gamma)\alpha\beta\gamma}{(\alpha-1)(\beta-1)(\gamma-1)} \text{ ops}, \tag{5.5.26}$$

or

$$\begin{aligned} C_{\pi f-e}(\alpha,\beta,\gamma) = &\frac{1}{2}\Big\{(\beta\gamma+\gamma+2)^2 + \frac{1}{\gamma-1}(\beta\gamma+\gamma+1)^2 \\ &+\frac{1}{\beta-1}(\beta\gamma+2)^2 + \frac{1}{(\beta-1)(\gamma-1)}(\beta\gamma+1)^2\Big\} \\ &+\frac{1}{2}\frac{1}{\alpha-1}\Big\{(\gamma+2)^2 + \frac{1}{\gamma-1}(\gamma+1)^2 \\ &+\frac{4}{\beta-1} + \frac{1}{(\beta-1)(\gamma-1)}\Big\} \text{ ops}. \end{aligned} \tag{5.5.27}$$

Thus $C_{\pi f-e}(\alpha,\beta,\gamma)$ is also minimal for large α and small β and γ. The values for $C_{\pi f-e}(\alpha,\beta,\gamma)$ for $6 \geq \beta \geq \gamma \geq 2$ are listed in Table 5.5.2. We see that the choice $\alpha \times 2 \times 2$ also minimizes the number of operations per element required to factor an $\alpha \times \beta \times \gamma$ mesh. The minimum factorization cost for the $p \times q \times r$ mesh is thus

$$C_{\pi-factor}(\mathbf{C}) = N_{EL}C_{\pi f-e}(p,2,2) \text{ ops}. \tag{5.5.28}$$

This represents a maximum cost reduction of 15% relative to the basic element factorization. Recall that the CPU cost for solution is proportional to the storage cost and therefore is also minimized by $p \times 2 \times 2$ substructuring. This is likewise true for the matrix vector product $\mathbf{Ax}$.

This analysis shows that minimal cost substructuring reduces the problem from $(p-1)(q-1)(r-1)$ basic elements to $(q-1)(r-1)$ substructures with mesh topology $p \times 2 \times 2$. Using the notion of disjoint element groups, discussed in the section on parallel algorithms, we may further combine disjoint $p \times 2 \times 2$ meshes to reduce the total number of substructures to four.

$\alpha \times \beta \times \gamma$	$C_{\pi f-e}(\alpha,\beta,\gamma)$ (ops)	$C_{\pi f-e}(\alpha,\beta,\gamma)/C_{\pi f-e}(2,2,2)$
$2 \times 2 \times 2$	$87.00 + 15.00$	1
$\alpha \times 2 \times 2$	$87.00 + 15.00/(\alpha - 1)$	$0.8529 + 0.1471/(\alpha - 1)$
$\alpha \times 3 \times 2$	$118.8 + 13.75/(\alpha - 1)$	$1.164 + 0.1348/(\alpha - 1)$
$\alpha \times 3 \times 3$	$183.0 + 17.63/(\alpha - 1)$	$1.794 + 0.1728/(\alpha - 1)$
$\alpha \times 4 \times 2$	$162.7 + 13.33/(\alpha - 1)$	$1.595 + 0.1307/(\alpha - 1)$
$\alpha \times 4 \times 3$	$255.3 + 17.25/(\alpha - 1)$	$2.502 + 0.1691/(\alpha - 1)$
$\alpha \times 4 \times 4$	$385.6 + 22.89/(\alpha - 1)$	$3.780 + 0.2244/(\alpha - 1)$
$\alpha \times 5 \times 2$	$215.6 + 13.13/(\alpha - 1)$	$2.114 + 0.1287/(\alpha - 1)$
$\alpha \times 5 \times 3$	$342.4 + 17.06/(\alpha - 1)$	$3.357 + 0.1673/(\alpha - 1)$
$\alpha \times 5 \times 4$	$521.0 + 22.71/(\alpha - 1)$	$5.108 + 0.2226/(\alpha - 1)$
$\alpha \times 5 \times 5$	$744.4 + 29.53/(\alpha - 1)$	$7.298 + 0.2895/(\alpha - 1)$
$\alpha \times 6 \times 2$	$277.0 + 13.00/(\alpha - 1)$	$2.716 + 0.1275/(\alpha - 1)$
$\alpha \times 6 \times 3$	$443.5 + 16.95/(\alpha - 1)$	$4.349 + 0.1662/(\alpha - 1)$
$\alpha \times 6 \times 4$	$678.6 + 22.60/(\alpha - 1)$	$6.653 + 0.2216/(\alpha - 1)$
$\alpha \times 6 \times 5$	$972.9 + 29.42/(\alpha - 1)$	$9.538 + 0.2885/(\alpha - 1)$
$\alpha \times 6 \times 6$	$1325. + 37.32/(\alpha - 1)$	$12.99 + 0.3659/(\alpha - 1)$

Table 5.5.2

Average product-factor cost per element, $C_{\pi f-e}$, for an $\alpha \times \beta \times \gamma$ mesh.

5.5.4 General Topologies

The preceding analysis assumed a regular mesh. Note, however, that the topology and not the geometry is important in the decomposition into substructures. There is a large class of problems with regular topology but irregular geometry. For large problems, the finite element mesh is usually generated automatically and thus has "zonal" topological regularity. Although the generalization to non-regular topology is non-trivial, the analysis suggests that "chains" of elements in the topologically longest direction produce minimal cost substructures. Irregular regions of the mesh left as individual elements will

incur no cost penalty with respect to the basic EL×EL algorithms.

These schemes are easily simplified to the two-dimensional case. The result is "strips" of quadrilateral elements in the p or q directions. If the mesh is regular it can be decomposed into two substructures composed of alternate strips. Thus, substructuring results in a two-component splitting of the operator.

5.5.5 Approximate Factorizations

We now consider approximate factorizations based on substructures. Clearly, all of the EL×EL approximate factorizations are applicable. For example, a reordered, one-pass, Crout EL×EL approximate factorization based on substructures in the p-direction is

$$\mathbf{C} = \left[\prod_{E=1}^{N_{SS}^{(p)}} \mathbf{L}_\pi(\mathbf{I} + \epsilon\tilde{\bar{\mathbf{A}}}_E^{(p)})\right]\left[\prod_{E=1}^{N_{SS}^{(p)}} \mathbf{D}_\pi(\mathbf{I} + \epsilon\tilde{\bar{\mathbf{A}}}_E^{(p)})\right]\left[\prod_{E=N_{SS}^{(p)}}^{1} \mathbf{L}_\pi^T(\mathbf{I} + \epsilon\tilde{\bar{\mathbf{A}}}_E^{(p)})\right], \qquad (5.5.29)$$

where $N_{SS}^{(p)}$ is the number of substructures resulting from the partial assembly in the p-direction and $\tilde{\bar{\mathbf{A}}}_E^{(p)}$ are the associated operators. An undesirable feature of (5.5.29) is bias in the p-direction, since the implicitness of the $\tilde{\bar{\mathbf{A}}}_E^{(p)}$ operators is greatest in the p-direction. In fact, information may be transferred across the mesh in the p-direction in just one solution, while requiring up to q or r solutions for the q and r directions, respectively.

To remove this bias, we propose using three sets of orthogonal "overlapping" substructures as in the following symmetrized three-pass alternating direction EL×EL approximate factorization :

$$\mathbf{C}=\left[\prod_{E=1}^{N_{SS}^{(p)}}\mathbf{L}_\pi(\mathbf{I}+\frac{\epsilon}{3}\tilde{\overline{\mathbf{A}}}_E^{(p)})\right]\left[\prod_{E=1}^{N_{SS}^{(q)}}\mathbf{L}_\pi(\mathbf{I}+\frac{\epsilon}{3}\tilde{\overline{\mathbf{A}}}_E^{(q)})\right]\left[\prod_{E=1}^{N_{SS}^{(r)}}\mathbf{L}_\pi(\mathbf{I}+\frac{\epsilon}{3}\tilde{\overline{\mathbf{A}}}_E^{(r)})\right]$$
$$\times\left[\prod_{E=1}^{N_{SS}^{(p)}}\mathbf{D}_\pi(\mathbf{I}+\frac{\epsilon}{3}\tilde{\overline{\mathbf{A}}}_E^{(p)})\right]\left[\prod_{E=1}^{N_{SS}^{(q)}}\mathbf{D}_\pi(\mathbf{I}+\frac{\epsilon}{3}\tilde{\overline{\mathbf{A}}}_E^{(q)})\right]\left[\prod_{E=1}^{N_{SS}^{(r)}}\mathbf{D}_\pi(\mathbf{I}+\frac{\epsilon}{3}\tilde{\overline{\mathbf{A}}}_E^{(r)})\right]$$
$$\times\left[\prod_{E=N_{SS}^{(r)}}^{1}\mathbf{L}_\pi^T(\mathbf{I}+\frac{\epsilon}{3}\tilde{\overline{\mathbf{A}}}_E^{(q)})\right]\left[\prod_{E=N_{SS}^{(q)}}^{1}\mathbf{L}_\pi^T(\mathbf{I}+\frac{\epsilon}{3}\tilde{\overline{\mathbf{A}}}_E^{(r)})\right]\left[\prod_{E=N_{SS}^{(p)}}^{1}\mathbf{L}_\pi^T(\mathbf{I}+\frac{\epsilon}{3}\tilde{\overline{\mathbf{A}}}_E^{(p)})\right], \quad (5.5.30)$$

where the $\tilde{\overline{\mathbf{A}}}_E^{(q)}$ are the operators ($2 \times q \times 2$ for example) resulting from substructuring to improve the q-direction implicitness, and the $\tilde{\overline{\mathbf{A}}}_E^{(r)}$ are the operators ($2 \times 2 \times r$ for example) resulting from substructuring to improve the r-direction implicitness. Thus each of the basic elements is used three times, once in each direction.

When coupled with an iterative scheme, the total cost of this three-pass subiteration algorithm is approximately $1\frac{1}{2}$ times the cost of a typical one-pass EL×EL approximate factorization, or approximately twice the cost of a one-pass substructure approximate factorization. However, it has the important advantage of propagating information throughout the entire mesh in one iteration. This suggests that the number of iterations required for convergence of the subiteration scheme may be reduced to $O(1)$.

An additional advantage of substructuring is the reduced amount of scatter/gather[2] required during the iterative loop. For the basic one-pass EL×EL approximate factorization applied to a three-dimensional problem, each nodal value is accessed eight times during an element level solution. The one-pass substructure approximate factorization reduces this number to four.

[2]Scatter/gather is a term coined for the movement of data from global vectors to temporary local vectors for processing, and back again.

§5.6 Effect of EL×EL Algorithms on Choice of A

In most of the current finite element programs, the algorithms and parameters governing the generation of the global operator **A** are specified *a priori* and remain fixed throughout the program execution. This is due largely to the constraints imposed by direct solution techniques. In a direct solution technique, large amounts of contiguous memory must be allocated for the factorized form of **A**. Modifying the overall structure[3] or size of **A** may be difficult. In addition, the factorization cost, incurred by modification of even a small part of **A**, is enormous for large problems.

The subiteration solution algorithms based on EL×EL approximate factorizations alleviate many of these constraints. They may be implemented with a simple but flexible data structure that easily allows structural modification. In addition, the ability to refactor independently any part of **A** is inherent in the algorithm.

This flexibility has many implications for the algorithms used to generate **A**. We now consider several ideas for modifying the form of **A** to reduce the cost of solving problems. The potential savings in nonlinear transient analysis procedures incorporating these ideas is clearly significant.

5.6.1 Selective Formation-Factorization

The most common method employed for the solution of nonlinear equations is a modified Newton-Raphson iteration procedure. This scheme entails the formation and factorization of the matrix **A** whenever the inaccuracy of the previously computed **A** begins to degrade convergence. The cost associated with the formation and factorization can be quite substantial. It dominates the problem-solution cost for large problems, as was shown previously.

---

[3]Structure denotes the topology or interconnectivity of the global equation system.

The EL×EL approximate factorization can reduce this cost in one of two ways. First, if the inaccuracy in **A** due to nonlinearities is "global", the entire operator may be reformed and factored at a much lower cost than that of a direct technique. Second, if the change in **A** is "local", for example due to a nonlinear radiation boundary condition or local material nonlinearity, then only those elements whose stiffnesses have changed significantly need be reformed and factored.

This, along with the cost analysis of the EL×EL approximate factorization sub-iteration algorithms performed previously, suggests that the nonlinear solution algorithm should use full Newton-Raphson iteration coupled with element-level selective reformation-factorization based on change in stiffness. Such a scheme would yield faster iteration-level convergence at a lower overall cost.

5.6.2 Implicit-Explicit Partitioning

The *implicit-explicit* finite element concept [Hughes 78a,78b,79,81,82a] has a very simple and clean implementation within EL×EL approximate factorizations. The standard implementation partitions the elements into two groups, implicit elements which contribute both mass and stiffness to the global operator **A**, and explicit elements which only contribute a diagonal mass matrix to **A**. In the EL×EL approximate factorization, the diagonal terms in **A** are accounted for in the scaling matrix **W**. **W** will contain the entire contribution to **A** from the explicit elements, and their corresponding $\overline{\mathbf{A}}_e$'s will be identically zero. This means that the explicit elements may be omitted from the formula for **C**, and thus ignored in the unwinding of the element-array products during the solution of $\mathbf{B}^{-1}\mathbf{r}$.

In nonlinear problems, this opens the way to time-adaptive implicit-explicit element partitions that would be impractical using direct solution techniques. In calculating the element contribution to the out-of-balance force, $\Delta\mathbf{F}$, a check is made to determine if the critical time step for the element is exceeded by the global time step. If it is not

exceeded, the element may be treated explicitly, and a flag is set to indicate that this element's contribution to **C** may be ignored. Thus, we can reduce the cost per iteration at the subiteration level by reducing the number of elements that need to be used in the "solution."

5.6.3 Adaptive Mesh Refinement

Another benefit worth mentioning is the ease of implementation of *adaptive mesh refinement* schemes within the framework of EL×EL approximate factorizations. Adaptive mesh refinement techniques employ a measure of solution error due to spatial discretization to selectively refine portions of the finite element mesh [Carey 81]. This refinement entails the addition of global degrees-of-freedom and a restructuring of the element connectivity. This, in turn, causes changes in the structure and size of the matrix **A**. Even a small local mesh refinement means complete reformation and factorization of **A** when using a direct solution technique.

The EL×EL algorithm, on the other hand, allows easy and natural implementation of adaptive mesh refinement. The elements in the region to be refined are simply removed from the list of elements for the problem and the new elements appended. Thus, only local modifications are required. The only formation/factorization performed is for the new elements. In addition, the natural "tree" type data structure proposed for adaptive mesh refinement may now be readily and efficiently utilized without the burden of global factorization costs.

§5.7 Algorithms for Parallel Computation

As mentioned in Chapter 4, the computations involved in an EL×EL approximate factorization are parallelizable. In this section, we consider modifications to the basic EL×EL algorithm which allow efficient use of a parallel-processing system. In contrast to EL×EL algorithms, direct solution algorithms based on product factorizations are

not easily implemented on a parallel-processing system due to their inherent serial nature. Although new methods of product factorizations have been developed to allow parallelization [Evans 82], they are full matrix techniques and thus unsuited to large finite element problems.

5.7.1 Machine Architecture

Before describing the implementation of EL×EL algorithms on a parallel-processing system, we must define the characteristics of the machine architecture [Hockney 81]. The architecture of parallel-processing systems varies greatly, and is a major factor in the development of efficient, parallel, numerical algorithms.

The single most important characteristic is N_P, the number of processors. The number of processors determines the upper bound on the ***speed-up factor***, S_{N_P}, the number of times faster an algorithm runs on a multi-processor machine than on a single processor. The speed-up factor depends on the machine architecture, the problem, and the algorithm. It is defined as

$$S_{N_P}(N) = \frac{T_1(N)}{T_{N_P}(N)} \leq N_P \,, \tag{5.7.1}$$

where N is a measure of the problem size, $T_1(N)$ is the time required to solve the problem using 1 processor with the "best" serial algorithm, and $T_{N_P}(N)$ is the time required to solve the problem using N_P processors and a parallelized algorithm. The measure of problem size N is typically the number of global equations N_{EQ} or the number of elements N_{EL}. The normalized version of the speed-up factor is the ***efficiency ratio***,

$$E_{N_P}(N) = \frac{S_{N_P}(N)}{N_P} \leq 1 \,. \tag{5.7.2}$$

The next important characteristic of parallel-processing systems is the degree of autonomy of the individual processors. In a *SIMD* (Single Instruction Multiple Data)

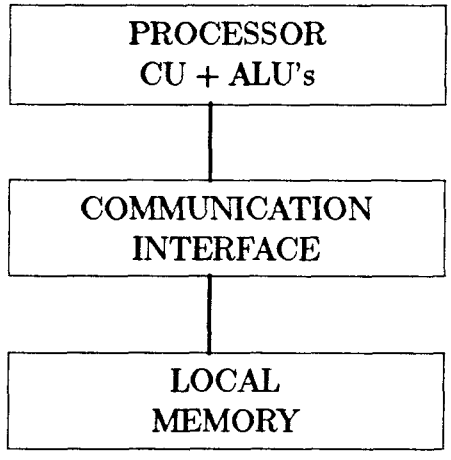


Figure 5.7.1 Example of typical processor (PR).

machine, all processors execute the same instruction in lock step. Thus, all processors run the identical program using different pieces of data. Examples of SIMD machines are the Cray-1, STAR, Illiac IV, Cyber 203/205, etc. A slight generalization of SIMD architecture allows processors to conditionally ignore the next several instructions in the stream, thus allowing a limited branching ability. A second, more general architecture exists in the *MIMD* (Multiple Instruction Multiple Data) machine. In these machines, the processors are autonomous. Each processor executes its own individual program with its own data. Examples of MIMD machines are the SMS 201 and C.mmp. In the implementation of EL×EL algorithms, we require that each processor have its own local memory. Thus, we will not consider the ***vector*** or ***pipeline*** computers such as the STAR, Cray-1, etc., but only the truly parallel computers like the Illiac IV, SMS 201, etc. A typical processor (PR) consists of several components: a control unit (CU), one or more arithmetic logic units (ALU's), a communication interface, and local memory. An example is given in Figure 5.7.1.

A final important characteristic is the ***communication topology***. This is the topology of the bus network joining the processors to each other and to main memory.

The simplest communication topology is one bus. Only one processor may use the

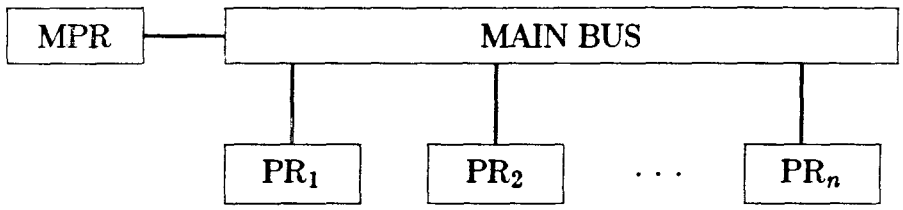


Figure 5.7.2 Example of zero-dimensional communication topology.

bus at a time. We refer to this topology as ***zero-dimensional*** topology, as the processors may only communicate through one point, the main bus. It has the advantage of having no local complexity and so it is easy to build. However, the limited communication facility may lead to communication bottlenecks. An example of zero-dimensional topology is shown in Figure 5.7.2.

At the other extreme is a topology where each processor is connected to every other processor and main memory through private busses. This scheme entails global complexity, requires $O\left(N_P^2\right)$ private busses, and has proven impractical for large N_P.

A practical topology, and one that offers significant computational advantage, is an *array topology*. In this architecture, a processor is only allowed to communicate directly with other processors in a restricted neighborhood. There is a constant local complexity independent of the number of processors. We further classify the array topology by its dimensionality. An example of a two-dimensional array topology is shown in Figure 5.7.3.

5.7.2 Implementation of EL×EL Algorithms

We now explore the implementation of the EL×EL approximate factorization solution algorithm on a variety of machine architectures.

5.7.2.1 Element Groups

We first define the notion of a ***disjoint element group***. A disjoint element group $\mathcal{E}_i$

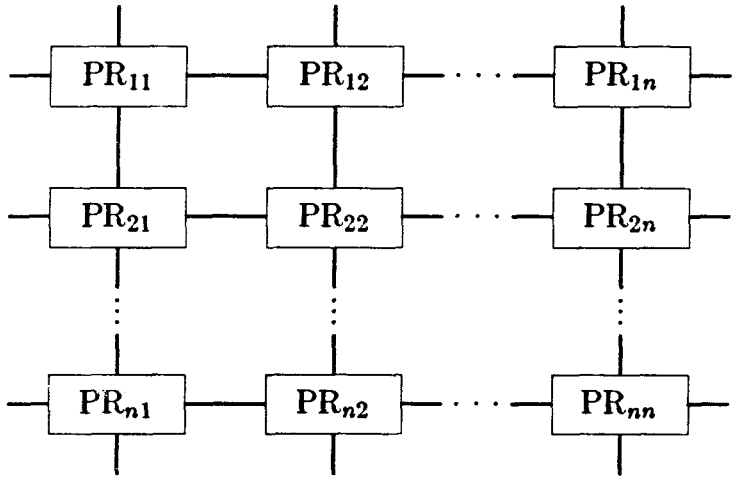


Figure 5.7.3
Example of two-dimensional array communication topology with toroidal closure.

is a subset of the set of all elements $\mathcal{E}$ with the property that no two elements in $\mathcal{E}_i$ may share a common global equation. We henceforth use the term *element group* to unambiguously mean a disjoint element group. The set of all elements is decomposed into a number of element groups such that

$$\bigcup_{i=1}^{N_{EG}} \mathcal{E}_i = \mathcal{E} = \{1, 2, \ldots, N_{EL}\}\,, \tag{5.7.3}$$

$$\mathcal{E}_i \bigcap \mathcal{E}_j = \emptyset \quad \forall \quad i \neq j \;\; i, j \in \{1, 2, \ldots, N_{EG}\}\,, \tag{5.7.4}$$

and

$$\mathcal{D}_{e_1} \bigcap \mathcal{D}_{e_2} = \emptyset \quad \forall \quad e_1 \neq e_2 \;\; e_1, e_2 \in \mathcal{E}_i \quad \forall \quad i \in \{1, 2, \ldots, N_{EG}\}\,, \tag{5.7.5}$$

where N_{EG} is the number of element groups and $\mathcal{D}_e$ is the set of all global equation numbers associated with element e.

For Each	Global Level	Element Level
iter*	form **A**	form $\mathbf{a}^e$
iter*	factor **B**	factor $(\mathbf{I}^e + \epsilon\mathbf{a}^e)$
iter	form **F**	form $\mathbf{f}^e$
subiter	form **Ax**	form $\mathbf{a}^e\mathbf{x}^e$
subiter	solve $\mathbf{B}^{-1}\mathbf{r}$	solve $(\mathbf{I}^e + \epsilon\mathbf{a}^e)^{-1}\mathbf{r}^e$

*(less often for modified Newton)

Table 5.7.1 Equivalent element level calculations.

5.7.2.2 Computational Considerations

Given a subdivision of the elements into element groups, there is a class of calculations that is easily parallelizable. This class is the element-level calculations for the elements within an element group. These calculations may be carried out in parallel after the necessary global values have been transferred to the processor's local memory. *Synchronization* is required only between element groups when the results of the element calculation needs to be transferred to main memory. The need for synchronization below the element group level is eliminated. Examples of calculations that are parallelizable at the element level for both iteration and subiteration loops are given in Table 5.7.1.

The simplest division satisfying (5.7.3)—(5.7.5) assigns one element per group, or $\mathcal{E}_i = \{i\}$ with $N_{EG} = N_{EL}$. This choice reduces the parallel algorithm to serial calculation.

If we assume that $N_{EL} \gg N_P$, valid for the forseeable future, then an integer multiple of N_P elements is needed in each group to achieve maximum efficiency. If the number of elements in a group i is $\mathcal{N}(\mathcal{E}_i)$, the time to process group i, ignoring communication time, is

$$T_{N_P}(k\mathcal{N}(\mathcal{E}_i)) = \left\lceil \frac{\mathcal{N}(\mathcal{E}_i)}{N_P} \right\rceil T_1(k) \,. \tag{5.7.6}$$

Here k is the size of the element calculation and $\lceil x \rceil$ is the smallest integer greater than or equal to x. The total time is

$$T_{N_P}(kN_{EL}) = \sum_{i=1}^{N_{EG}} \left\lceil \frac{\mathcal{N}(\mathcal{E}_i)}{N_P} \right\rceil T_1(k) , \tag{5.7.7}$$

which yields a speed-up factor of

$$S_{N_P}(kN_{EL}) = \frac{N_{EL}}{N_C} . \tag{5.7.8}$$

N_C is the total number of cycles of element level calculation,

$$N_C = \sum_{i=1}^{N_{EG}} \left\lceil \frac{\mathcal{N}(\mathcal{E}_i)}{N_P} \right\rceil . \tag{5.7.9}$$

Finally, the efficiency ratio is

$$E_{N_P}(kN_{EL}) = \frac{N_{EL}}{N_P N_C} . \tag{5.7.10}$$

Clearly, the speed-up factor and efficiency ratio are maximized by choosing $\mathcal{E}_i$ to minimize N_C. N_C is minimal when

$$\left\lceil \frac{\mathcal{N}(\mathcal{E}_i)}{N_P} \right\rceil = \frac{\mathcal{N}(\mathcal{E}_i)}{N_P} \quad \forall \quad i \in \{1, 2, \ldots, N_{EG}\} , \tag{5.7.11}$$

or $N_C = N_{EL}/N_P$, $S_{N_P}(kN_{EL}) = N_P$, and $E_{N_P}(kN_{EL}) = 1$.

One approach to minimizing N_C is to minimize the number of element groups N_{EG}, thus reducing the number of possible unfilled cycles. By minimizing N_{EG}, we are assured of being within N_{EG} cycles of the absolute minimum. This gives the worst-case estimate

for N_C

$$N_C = \frac{N_{EL}}{N_P} + N_{EG} \,. \tag{5.7.12}$$

With this worst-case estimate for N_C, the total calculation time is

$$T_{N_P}(kN_{EL}) = \left(\frac{N_{EL}}{N_P} + N_{EG}\right) T_1(k) \,, \tag{5.7.13}$$

the speed-up factor is

$$S_{N_P}(kN_{EL}) = \frac{N_P}{1 + \frac{N_{EG}N_P}{N_{EL}}} \,, \tag{5.7.14}$$

and the efficiency ratio is

$$S_{N_P}(kN_{EL}) = \frac{1}{1 + \frac{N_{EG}N_P}{N_{EL}}} \,. \tag{5.7.15}$$

All are very close to their optimal values if $N_{EL} \gg N_P$ and $N_{EG} = O(1)$.

For a general mesh, an absolute lower bound on N_{EG} may be computed after examining the restrictions that (5.7.5) places on group partitioning. This equation requires that "no group shall contain two elements contributing to the same global equation." An absolute lower bound on N_{EG} is thus

$$N_{EG} \geq \max_{eq=1}^{N_{EQ}} C_{eq} \,, \tag{5.7.16}$$

where C_{eq} is the number of elements contributing to equation eq.

For the regular three-dimensional "brick-like" domain shown in Figure 5.7.4, the lower bound $N_{EG} \geq 8$ may be satisfied identically, and the elements may be divided among the eight groups as indicated. For the analogous, regular, two-dimensional, "rectangular-like" domain, four element groups need be employed, while for one-dimensional "bar-like" domains, only two groups are required.

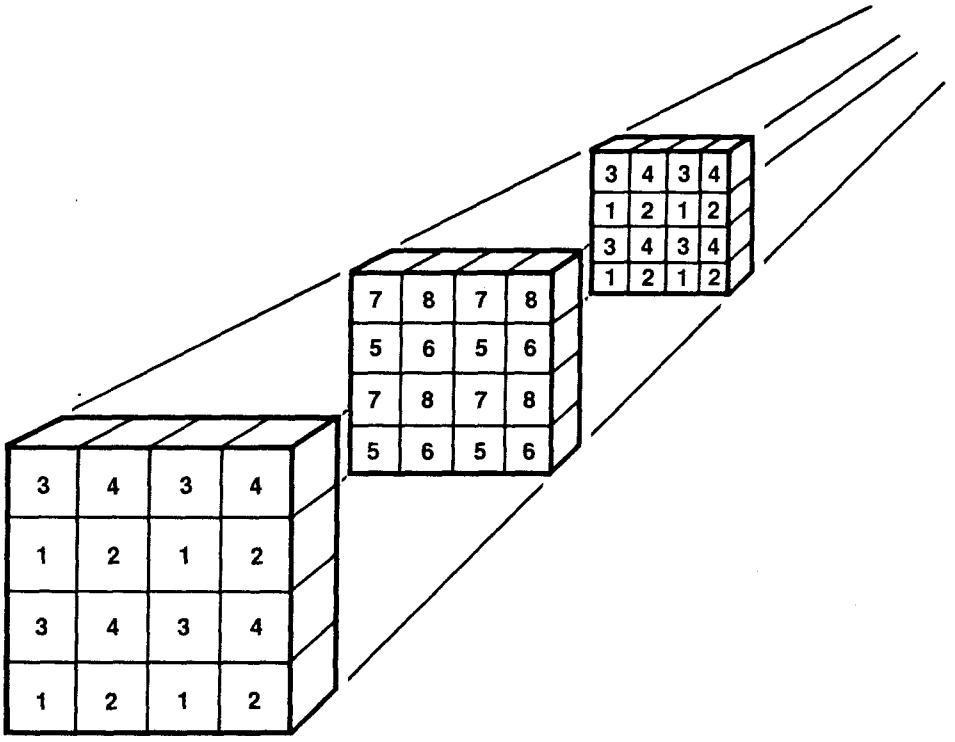


Figure 5.7.4
Decomposition of three-dimensional domain into eight groups of brick elements for parallel processing.

For a general mesh, a simple algorithm generating an *a priori* estimate of the minimum N_{EG} is as yet unknown. However, we have determined a recursive exhaustive-search algorithm which takes time $O(N_{EL})$ in the best case and $O\left(N_{EL}^2\right)$ in the worst case to sort the elements into groups. Fortunately, the decomposition into element groups need only be done in the pre-processing stage of problem solution, and is similar in cost to the equation-reordering schemes commonly used to reduce the cost of direct solution techniques. It should also be mentioned that all of the ideas governing the choice of **A** mentioned in the last section, selective formation-factorization, etc., admit easy and natural parallel-processor implementation.

For Each	Global Level	Localize	Element Level	Globalize
iter*	form $\mathbf{A}$	$\mathbf{d}, \mathbf{v} \rightarrow \mathbf{d}^e, \mathbf{v}^e$	form $\mathbf{a}^e$	$\mathbf{A}\,\mathbf{a}^e \rightarrow \mathbf{A}$
iter*	factor $\mathbf{B}$		factor $(\mathbf{I}^e + \epsilon\bar{\mathbf{a}}^e)$	
iter	form $\mathbf{F}$	$\mathbf{d}, \mathbf{v} \rightarrow \mathbf{d}^e, \mathbf{v}^e$	form $\mathbf{f}^e$	$\mathbf{A}\,\mathbf{f}^e \rightarrow \mathbf{F}$
subiter	form $\mathbf{Ax}$	$\mathbf{x} \rightarrow \mathbf{x}^e$	form $\mathbf{a}^e\mathbf{x}^e$	$\mathbf{A}\,\mathbf{a}^e\mathbf{x}^e \rightarrow \mathbf{r}$
subiter	solve $\mathbf{B}^{-1}\mathbf{r}$	$\mathbf{r} \rightarrow \mathbf{r}^e$	solve $(\mathbf{I}^e + \epsilon\bar{\mathbf{a}}^e)^{-1}\mathbf{r}^e$	$\mathbf{r}^e \rightarrow \mathbf{x}$

*(less often for modified Newton)

Table 5.7.2 Communication costs associated with element level calculation.

5.7.2.3 Communication Considerations

In the preceding analysis of the efficiency of the parallelized EL×EL algorithms, we were concerned only with the computational cost and associated speed-up of calculations at the element level. We did not concern ourselves with the communication costs associated with localization of data before element calculation and globalization of data after element calculation. We now consider the communication costs, those costs associated with the overhead incurred by interference in accessing shared resources.

It is instructive to add communication considerations to Table 5.7.1. The result is shown in Table 5.7.2. There are clearly potential communication bottlenecks generated by the localization/globalization of nodal data values at both the iteration and subiteration levels. We now consider the causes and possible solutions to these communication bottlenecks for two parallel-processor communication topologies.

The first topology considered is the zero-dimensional or main bus topology. For this topology, the communication bottlenecks can be quite severe. The simplest approach to the localization of data is broadcasting the necessary components of the global vector on the main bus and allowing each processor to pick up the values it requires. Similarly, the globalization may be performed by granting each processor ownership of the bus for the transference of its updated global values. The difficulty with this approach is

that each localization/globalization requires $O(N_P)$ bus accesses during which many processors may be idle. A more sophisticated approach assigns elements to processors based on the locality of data. Given enough local memory, a processor may retain selected global values for use in succeeding element calculations. This approach may reduce the communication bottlenecks incurred in the simpler approach by a factor of N_{EG}. However, to get a real improvement, we must tailor the communication topology to that required by the algorithm.

The second topology, and by far the more interesting, is that of an n-dimensional array. If we assume that n is equal to or greater than the dimensionality of the problem, we may map the elements of the problem onto the processors of the system in such a way that spatially adjacent elements are mapped to processors sharing local communication. The information transfer previously performed by a globalization/localization sequence may now be accomplished using only local interprocessor communication. This reduces the communication time to $O(1)$ access cycles and eliminates a major communication bottleneck.

5.7.2.4 Effect of Parallel Orderings on EL×EL Errors

One of the surprising benefits of the parallel ordering schemes is the additional insight into the error terms in the EL×EL approximate factorizations. For the general one-pass EL×EL approximate factorization, the exact form of the error terms elude explicit description. This is due both to the number of elements and the interaction of the $\overline{\mathbf{A}}_e$ matrices being as yet unknown. However, by reordering the elements in accordance with a specified parallel scheme, we may transform the EL×EL factorization into an equivalent multi-component factorization with N_{EG} components. This transformation is accomplished by defining

$$\tilde{\overline{\mathbf{A}}}_i = \sum_{e \in \mathcal{E}_i} \overline{\mathbf{A}}_e, \quad i \in \{1, 2, \ldots, N_{EG}\} \tag{5.7.17}$$

and noting that

$$\overline{\mathbf{A}}_{e_1}\overline{\mathbf{A}}_{e_2} = \mathbf{0}, \quad e_1, e_2 \in \mathcal{E}_i, \quad i \in \{1, 2, \ldots, N_{EG}\}\,. \tag{5.7.18}$$

The $\tilde{\overline{\mathbf{A}}}_i$ may be thought of as a form of superelement which contains the assembled contributions for those elements in $\mathcal{E}_i$. Making use of (5.7.18) in the definition of one-pass EL×EL approximate factorizations, we find

$$\begin{aligned}\prod_{e=1}^{N_{EL}}(\mathbf{I}+\epsilon\overline{\mathbf{A}}_{p(e)}) &= \prod_{i=1}^{N_{EG}}\left[\prod_{e\in\mathcal{E}_i}(\mathbf{I}+\epsilon\overline{\mathbf{A}}_e)\right] \\ &= \prod_{i=1}^{N_{EG}}(\mathbf{I}+\epsilon\sum_{e\in\mathcal{E}_i}\overline{\mathbf{A}}_e) \\ &= \prod_{i=1}^{N_{EG}}(\mathbf{I}+\epsilon\tilde{\overline{\mathbf{A}}}_i)\,,\end{aligned} \tag{5.7.19}$$

where $p(e)$ is the mapping into parallel order. Thus, for a regular two-dimensional mesh with $N_{EG} = 4$, the problem is reduced to a multi-component split using four superelements. We find

$$\begin{aligned}\prod_{e=1}^{N_{EL}}(\mathbf{I}+\epsilon\overline{\mathbf{A}}_{p(e)}) =&(\mathbf{I}+\epsilon\tilde{\overline{\mathbf{A}}}_1)(\mathbf{I}+\epsilon\tilde{\overline{\mathbf{A}}}_2)(\mathbf{I}+\epsilon\tilde{\overline{\mathbf{A}}}_3)(\mathbf{I}+\epsilon\tilde{\overline{\mathbf{A}}}_4) \\ =&\mathbf{I}+\epsilon\overline{\mathbf{A}} \\ &+\epsilon^2\left[\tilde{\overline{\mathbf{A}}}_1(\tilde{\overline{\mathbf{A}}}_2+\tilde{\overline{\mathbf{A}}}_3+\tilde{\overline{\mathbf{A}}}_4)+\tilde{\overline{\mathbf{A}}}_2(\tilde{\overline{\mathbf{A}}}_3+\tilde{\overline{\mathbf{A}}}_4)+\tilde{\overline{\mathbf{A}}}_3\tilde{\overline{\mathbf{A}}}_4\right] \\ &+\epsilon^3\left[\tilde{\overline{\mathbf{A}}}_1\tilde{\overline{\mathbf{A}}}_2\tilde{\overline{\mathbf{A}}}_3+\tilde{\overline{\mathbf{A}}}_1\tilde{\overline{\mathbf{A}}}_2\tilde{\overline{\mathbf{A}}}_4+\tilde{\overline{\mathbf{A}}}_1\tilde{\overline{\mathbf{A}}}_3\tilde{\overline{\mathbf{A}}}_4+\tilde{\overline{\mathbf{A}}}_2\tilde{\overline{\mathbf{A}}}_3\tilde{\overline{\mathbf{A}}}_4\right] \\ &+\epsilon^4\left[\tilde{\overline{\mathbf{A}}}_1\tilde{\overline{\mathbf{A}}}_2\tilde{\overline{\mathbf{A}}}_3\tilde{\overline{\mathbf{A}}}_4\right].\end{aligned} \tag{5.7.20}$$

By assuming a parallel ordering, we can therefore compute an explicit representation of the error in the EL×EL approximate factorization which is independent of the number of elements and contains terms only up through $\epsilon^{N_{EG}}$.

Although we have not, as yet, used this explicit form of the error to improve the approximation, we hope through the use of successive corrections [Yanenko 71] to achieve a more accurate expansion. In addition, this analysis suggests the use of hexagonal elements to tile regular two-dimensional regions. Regular hexagonal tiling results in

$$N_{EG} \geq \max_{eq=1}^{N_{EQ}} C_{eq} = 3 \; ; \tag{5.7.21}$$

thus, only three element groups are required. This would eliminate $\tilde{\overline{A}}_4$ and the ϵ^4 error term completely.

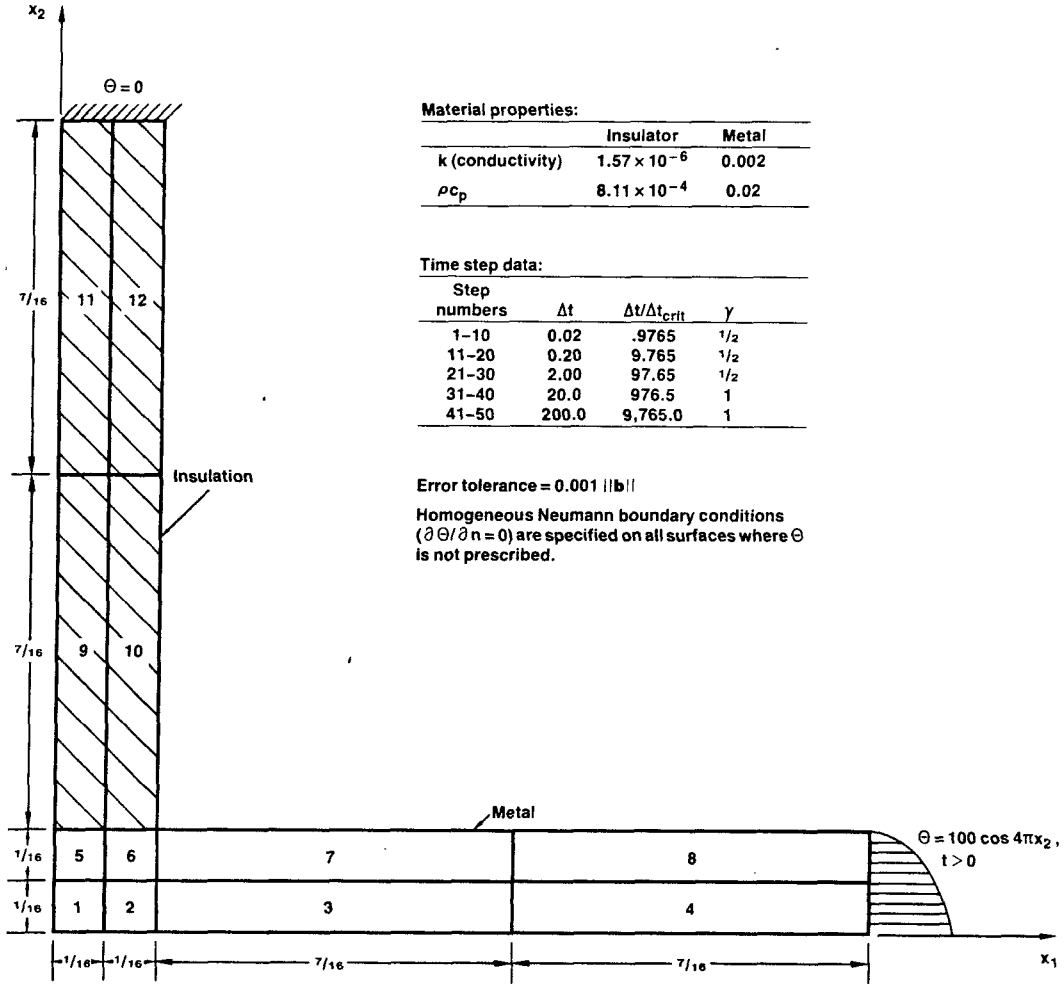
Chapter 6

Numerical Results

In this chapter, we discuss the numerical results obtained using the EL×EL subiteration technique in the solution of nonlinear transient heat conduction analysis. Unless otherwise noted, all results compare favorably with solutions obtained using a fully implicit direct solution technique.

§6.1 NASA Insulated Structure Test Problem

The first problem we consider is the NASA insulated test problem. The problem description is illustrated in Figure 6.1.1. The problem is posed on an L-shaped region; the horizontal bar is metal while the vertical bar is insulation. Although the materials are linear, obtaining a complete transient solution is non-trivial. The difficulty lies in the "stiffness" of the problem. The diffusivity ratio between the two materials is approximately 50. Thus the solution components have two important time scales. First is the fast conduction of heat through the metal. Then, at a rate 50 times slower, the insulation responds to what it sees as a changed boundary condition. Using an explicit integration algorithm on this problem, one would be forced to choose a time step less than or equal to the critical time step for the smallest metallic element. To obtain the complete transient solution, from initial condition to steady state, would then require several hundred time steps. Instead, using the EL×EL approximation to the implicit



Material properties:

	Insulator	Metal
k (conductivity)	1.57×10^{-6}	0.002
ρc_p	8.11×10^{-4}	0.02

Time step data:

Step numbers	Δt	$\Delta t/\Delta t_{crit}$	γ
1–10	0.02	.9765	1/2
11–20	0.20	9.765	1/2
21–30	2.00	97.65	1/2
31–40	20.0	976.5	1
41–50	200.0	9,765.0	1

Error tolerance = 0.001 ||**b**||

Homogeneous Neumann boundary conditions ($\partial\Theta/\partial n = 0$) are specified on all surfaces where Θ is not prescribed.

Figure 6.1.1

NASA insulated test problem description.

operator, we start the integration at the critical time step, but increase it by a factor of 10 after every 10 steps, requiring only 50 steps to reach steady state.

A number of comparisons of the various techniques proposed were made for this problem. In Table 6.1.1 subiteration algorithms using $\tilde{\mathbf{A}}_{OPT}$, $\tilde{\mathbf{A}} = \mathbf{A}$, are seen to converge faster than those using $\tilde{\mathbf{A}}_{PR}$, $\tilde{\mathbf{A}} = \mathbf{D}_\sigma(\mathbf{A}) + \mathbf{A}$. In addition, the steady-state residual potential energy, measured by $\log_{10}(-P_f)$, attains a smaller value when using $\tilde{\mathbf{A}}_{OPT}$. This and other calculations have indicated that $\tilde{\mathbf{A}}_{OPT}$ is the superior choice. It

		Avg Subiter. per Step	
Algorithm	$\log_{10}(-P_f)^{(\dagger)}$	steps 1-20	steps 21-50
$\langle PSD, \tilde{\mathbf{A}}_{PR}, \langle 2, \mathcal{N}, \pi \rangle\rangle$	−13.0	6.00	$10^{(\ddagger)}$
$\langle PSD, \tilde{\mathbf{A}}_{OPT}, \langle 2, \mathcal{N}, \pi \rangle\rangle$	−15.4	5.03	$10^{(\ddagger)}$

$^{(\dagger)}$(The more negative $\log_{10}(-P_f)$ the closer the solution is to steady state)
$^{(\ddagger)}$(Failed to converge in 10 iterations)

Table 6.1.1

Comparison of $\tilde{\mathbf{A}}_{PR}$ and $\tilde{\mathbf{A}}_{OPT}$ using $\langle PSD, *, \langle 2, \mathcal{N}, \pi \rangle\rangle$ subiteration algorithm

		Avg Subiter. per Step	
Algorithm	$\log_{10}(-P_f)^{(\dagger)}$	steps 1-20	steps 21-50
$\langle PSD, \tilde{\mathbf{A}}_{OPT}, \langle 1, \mathcal{R}, \tilde{\sigma} \rangle\rangle$	−13.4	5.41	$10^{(\ddagger)}$
$\langle PSD, \tilde{\mathbf{A}}_{OPT}, \langle 2, \mathcal{N}, \pi \rangle\rangle$	−15.4	5.03	$10^{(\ddagger)}$
$\langle PSD, \tilde{\mathbf{A}}_{OPT}, \langle 1, \mathcal{R}, \tilde{\pi} \rangle\rangle$	−15.8	3.95	$10^{(\ddagger)}$
$\langle PSD, \tilde{\mathbf{A}}_{OPT}, \langle 1, \mathcal{R}, \pi \rangle\rangle$	−15.1	3.95	$10^{(\ddagger)}$
$\langle PCG, \tilde{\mathbf{A}}_{OPT}, \langle 1, \mathcal{R}, \tilde{\sigma} \rangle\rangle$	−25.3	3.95	9.0
$\langle PCG, \tilde{\mathbf{A}}_{OPT}, \langle 2, \mathcal{N}, \pi \rangle\rangle$	−25.3	3.50	8.3
$\langle PCG, \tilde{\mathbf{A}}_{OPT}, \langle 1, \mathcal{R}, \tilde{\pi} \rangle\rangle$	−25.3	3.45	7.9
$\langle PCG, \tilde{\mathbf{A}}_{OPT}, \langle 1, \mathcal{R}, \pi \rangle\rangle$	−25.3	2.95	8.0

$^{(\dagger)}$(The more negative $\log_{10}(-P_f)$ the closer the solution is to steady state)
$^{(\ddagger)}$(Failed to converge in 10 iterations)

Table 6.1.2

Comparison of *PSD* and *PCG* iterative algorithms using $\tilde{\mathbf{A}}_{OPT}$ combined with various EL×EL approximate factorizations.

is used in the comparisons shown in Table 6.1.2.

The first observation which may be made here is that the *PCG* algorithm is more effective than the *PSD* algorithm. Thus, our current preference in symmetric positive-

definite cases is the *PCG* method. The EL×EL factorizations ranked from best to worst are : $\langle 1, \mathcal{R}, \pi \rangle$, $\langle 1, \mathcal{R}, \tilde{\pi} \rangle$, $\langle 2, \mathcal{N}, \pi \rangle$, $\langle 1, \mathcal{R}, \tilde{\sigma} \rangle$. Nevertheless, it must be kept in mind that the overall computational efficiency may alter this ordering. For example, although the symmetrized Gauss-Seidel was slowest to converge, it does not require element factorization, which is an advantage.

A final point is that convergence is typically slower during the larger time step sequences (i.e. steps 21-50) than the smaller step sequences (i.e. 1-20). There appear to be two reasons for this. Firstly, for the larger time steps, the solution closely approximates the steady state. Thus, the initial residual is fairly small, resulting in a more stringent convergence criteria. (In fact, even the "non-converged" solutions possessed adequate accuracy from a practical standpoint.) Secondly, the conditioning of the element factors deteriorates for larger steps as indicated by the analysis of the error in approximate factorizations performed in Chapter 4.

§6.2 Parallel/Sequential Test Problem

The problem description is given in Figure 6.2.1. The purpose of this problem is to compare convergence characteristics for "natural" element orderings, which necessitate sequential processing, with orderings that lend themselves to parallel computations. The comparisons are all performed with the $\langle PCG, \tilde{\mathbf{A}}_{OPT}, \langle 1, \mathcal{R}, \pi \rangle \rangle$ subiteration algorithm.

Over the thirty time steps, the sequential ordering averaged 2.53 iterations per step to attain convergence, whereas the parallel ordering averaged 3.47 iterations. Despite the fact that the parallel ordering is slower, which might be anticipated, the fact that it is reasonably fast is extremely encouraging. For the 256 element mesh shown, a 64-processor computer could attain speeds 64 times faster than a single processor. This more than compensates for the somewhat slower convergence of the parallel ordering. The gains in larger problems are potentially even more spectacular.

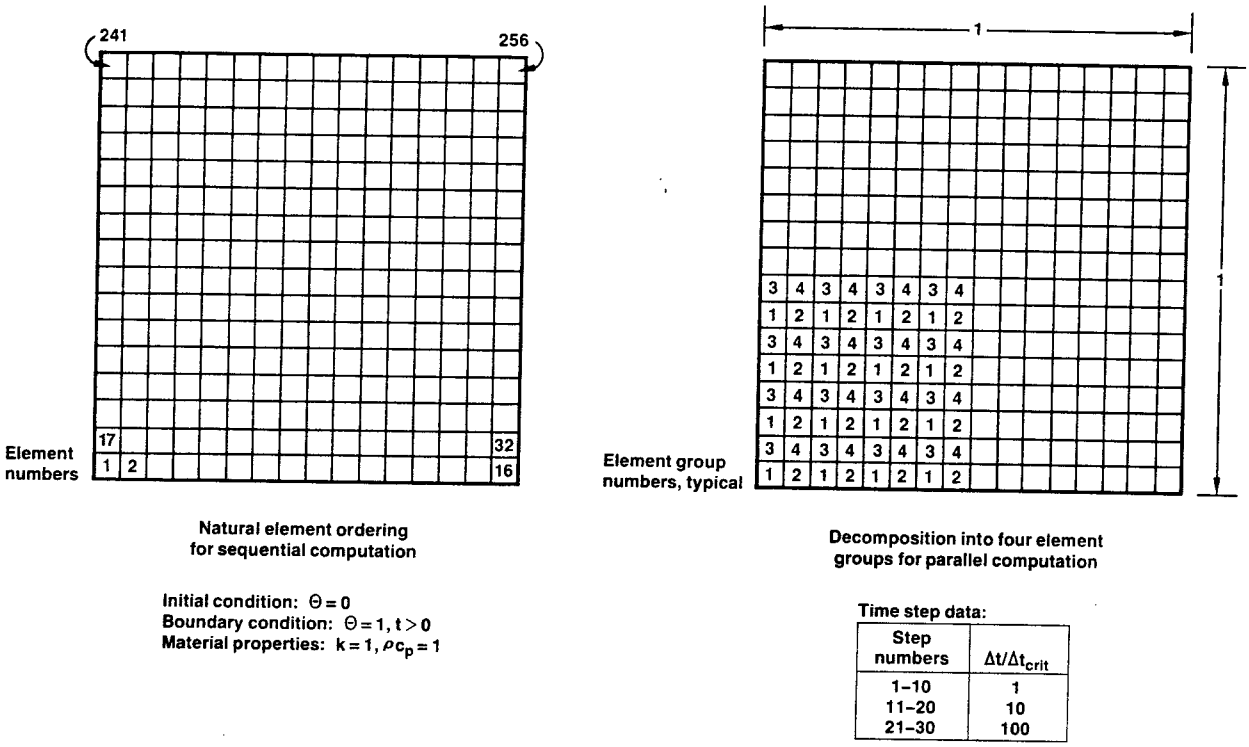


Figure 6.2.1

Problem description for parallel/sequential test problem.

§6.3 Arch Problem

The next problem we consider is the arch problem. This problem has been previously investigated by several researchers [Bruch 74], [Bell 76], [Trujillo 77b,82]. Although we solved both linear and nonlinear versions of this problem, only the results for the nonlinear version are presented.

A distinctive feature of this problem are the two re-entrant corners. Near sharp corners, there may be singularities in the solution which cause the spatial derivatives of the solution to become unbounded.

The material properties are: constant density and specific heat,

$$\rho = 1.0 \frac{\text{kg}}{\text{m}^3} \tag{6.3.1}$$

and

$$C_p = 1.0 \frac{\mathrm{W \cdot s}}{\mathrm{kg \cdot {}^\circ K}} , \tag{6.3.2}$$

and a nonlinear isotropic thermal conductivity,

$$k = \left(1 + \frac{T}{1000^\circ \mathrm{K}}\right) \frac{\mathrm{W}}{\mathrm{m \cdot {}^\circ K}} . \tag{6.3.3}$$

The boundary conditions are: prescribed temperature on the leftmost edge,

$$T = 1000^\circ \mathrm{K} \quad \text{on} \quad x = 0\,\mathrm{m}, \ t \geq 0 , \tag{6.3.4}$$

prescribed temperature on the rightmost edge,

$$T = 0^\circ \mathrm{K} \quad \text{on} \quad x = 1\,\mathrm{m}, \ t \geq 0 , \tag{6.3.5}$$

and insulation on all other boundaries,

$$\mathbf{n} \cdot (\mathbf{K} \nabla T) = 0 \frac{\mathrm{W}}{\mathrm{m}} \ t \geq 0 . \tag{6.3.6}$$

The initial temperature distribution was taken to be

$$T(x, y, t^*) = 10^3 \mathrm{erfc}\left(\frac{x}{2\sqrt{\kappa t^*}}\right)^\circ \mathrm{K} , \tag{6.3.7}$$

which is the short-time linear solution at time t^* for a plane semi-infinite medium. In our analysis, we assumed $\kappa = 1\,\mathrm{m}^2/\mathrm{s}$ and $t^* = 0.0005\,\mathrm{s}$ in the calculation of our initial conditions.

The discretized spatial domain is shown in Figures 6.3.1 and 6.3.2. The time domain of interest is $0 \geq t \geq 0.1\,\mathrm{s}$.

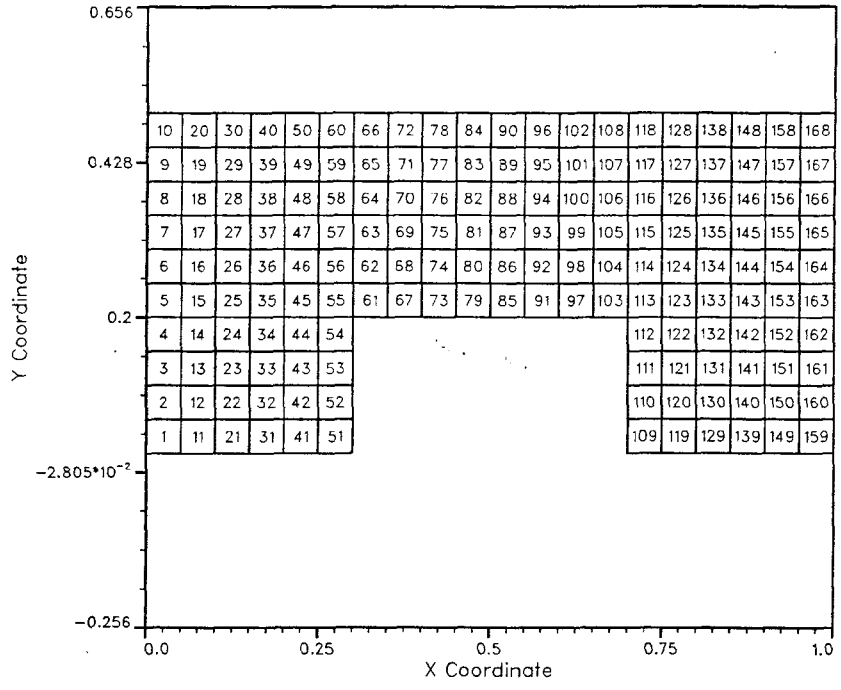


Figure 6.3.1

Arch Problem: Finite element mesh (element numbers).

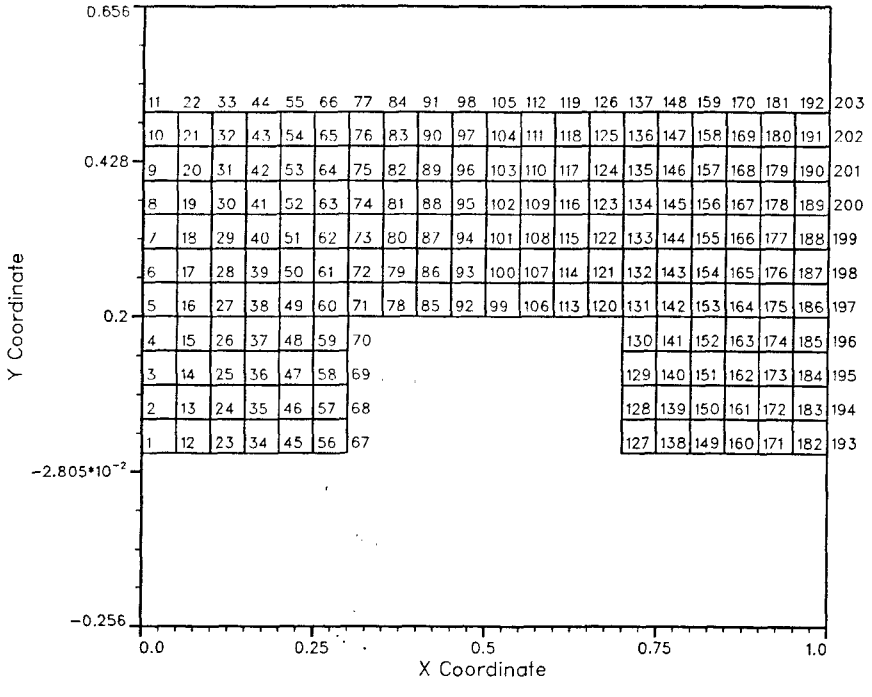


Figure 6.3.2

Arch Problem: Finite element mesh (node numbers).

This problem was first used to test and verify the error measures developed for the nonlinear iteration and linear subiteration levels. These tests confirmed the strong relationship among the convergence rates and errors in $\mathbf{v}$, $\Delta\mathbf{F}$, and $\mathbf{v}^T\Delta\mathbf{F}$. This allowed the development and implementation of the optimal subiteration error criteria discussed in Chapter 5.

The final analysis of the problem consisted of a series of five computer runs made to assess the effectiveness of the optimal subiteration error control. All runs used an iteration error tolerance of 10^{-7} and the $\langle PCG, \tilde{\mathbf{A}}_{OPT}, \langle 1, \mathcal{R}, \pi \rangle \rangle$ subiteration algorithm. The five runs are defined as follows :

A. Subiteration error tolerance equal to iteration convergence rate, $\beta_j = 1$.

B. Fixed at one subiteration per iteration regardless of error, $\beta_j = \infty$.

C. Subiteration error tolerance equal to one one-hundredth of the iteration convergence rate, $\beta_j = 0.01$.

D. Subiteration error tolerance equal to the iteration convergence rate, $\beta_j = 1$, and a bail-out option implemented for quickly convergent subiteration solutions.

E. Same as Run D but with a parallel ordering.

The time-step sizes used are listed in Table 6.3.1 and a summary of the costs of the five runs is given in Table 6.3.2.

Comparison of Runs A, B, and C indicates that the choice of $\beta_j = 1$ is justified. Run B, which used a larger subiteration error tolerance than Run A, required fewer total subiterations but 94% more iterations to compensate for the reduced subiteration accuracy. On the other hand, Run C, with a smaller subiteration error tolerance than Run A, required 50% more subiterations without reducing the number of iterations or improving the solution accuracy.

Steps	Δt
$1 \rightarrow 10$	0.0005
$11 \rightarrow 15$	0.001
$16 \rightarrow 20$	0.002
$21 \rightarrow 25$	0.004
$26 \rightarrow 31$	0.01

Table 6.3.1

Step sizes used in the solution of the Arch problem.

Run	Subiter. error tol.	Total Iter.	Total Subiter.	Avg Subiter per Iter.
A	$\beta_j = 1$	142	354	2.49
B	$\beta_j = \infty^{(\dagger)}$	276	276	1.00
C	$\beta_j = 0.01$	142	528	3.72
D	$\beta_j = 1^{(\ddagger)}$	142	328	2.31
E	$\beta_j = 1^{(\ddagger)}$	142	331	2.33

$^{(\dagger)}$(Fixed one subiter per iter)
$^{(\ddagger)}$(With bail-out procedure)

Table 6.3.2

Summary of costs for solution of the Arch problem.

In runs D and E, a "bail-out" option was added to the subiteration loop which allowed the subiteration loop to exit early based on reduction of the norm of the residual. This further reduced the total number of subiterations required to 328 without adversely affecting the iteration level accuracy as shown by Run D.

In Run E, the elements were subdivided into disjoint-element groups as shown in Figure 6.3.3 and ordered to simulate solution on a parallel processor as shown in Figure 6.3.4. This resulted in less than a one percent increase in the total number of

subiterations required over the natural ordering used in Run D. Note that a speedup-factor of up to 42 is possible for this problem on a suitable parallel processing machine.

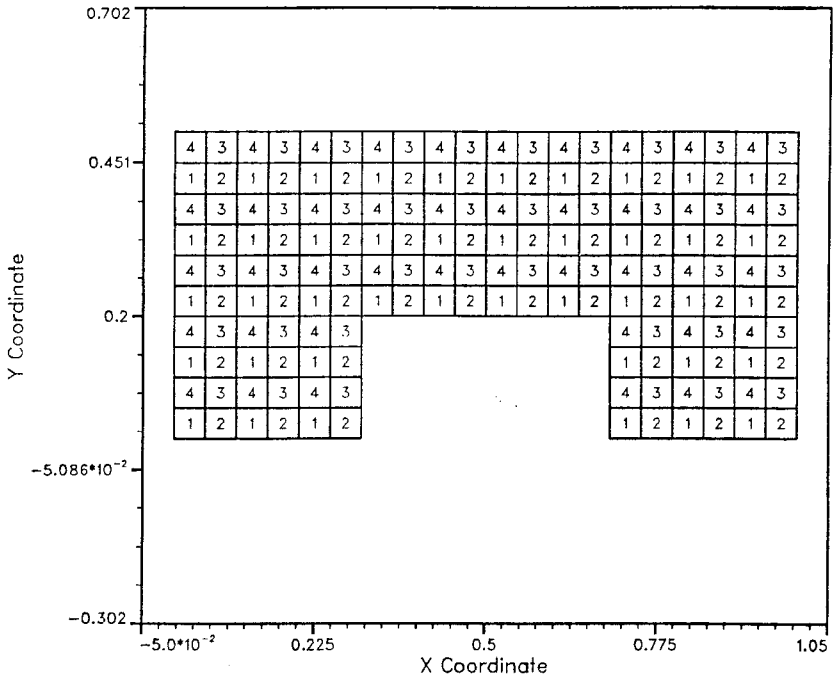


Figure 6.3.3

Arch Problem Run E: Disjoint element groups.

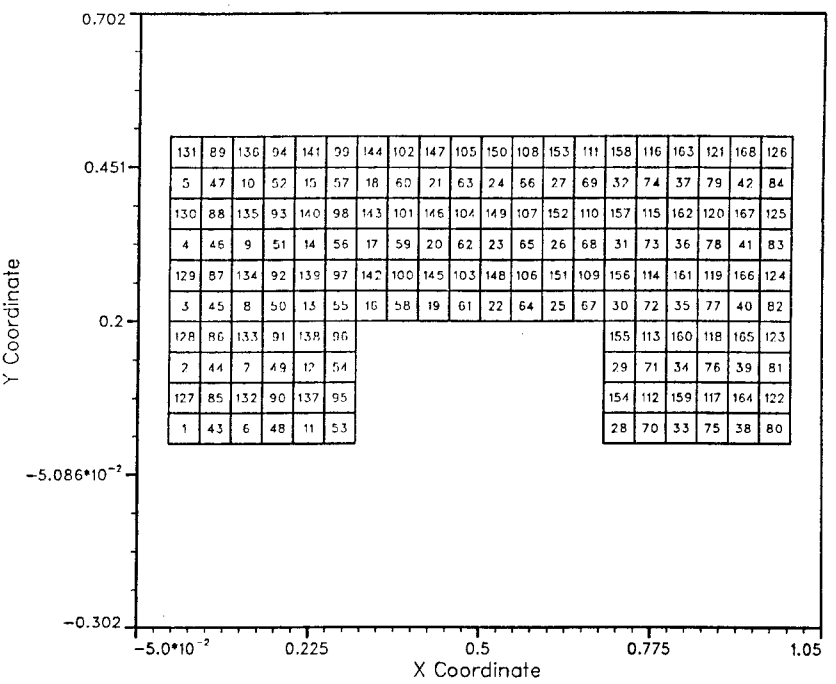


Figure 6.3.4

Arch Problem Run E: Element order corresponding to disjoint element group numbers.

The solutions at the time-step level for all runs were identical (to seven digits). The temperature profiles as a function of time for selected nodes are shown in Figure 6.3.5. The temperature and velocity contours at $t = 0.1\,\mathrm{s}$ are shown in Figures 6.3.6 and 6.3.7. Finally, the unnormalized step error measures based on $\frac{1}{2}$-step residuals are shown in Figure 6.3.8. Note that between two and three significant digits are maintained except at the steps where the step size is doubled. This motivated the development and implementation of the automatic step-selection strategy used in the solution of the next problem.

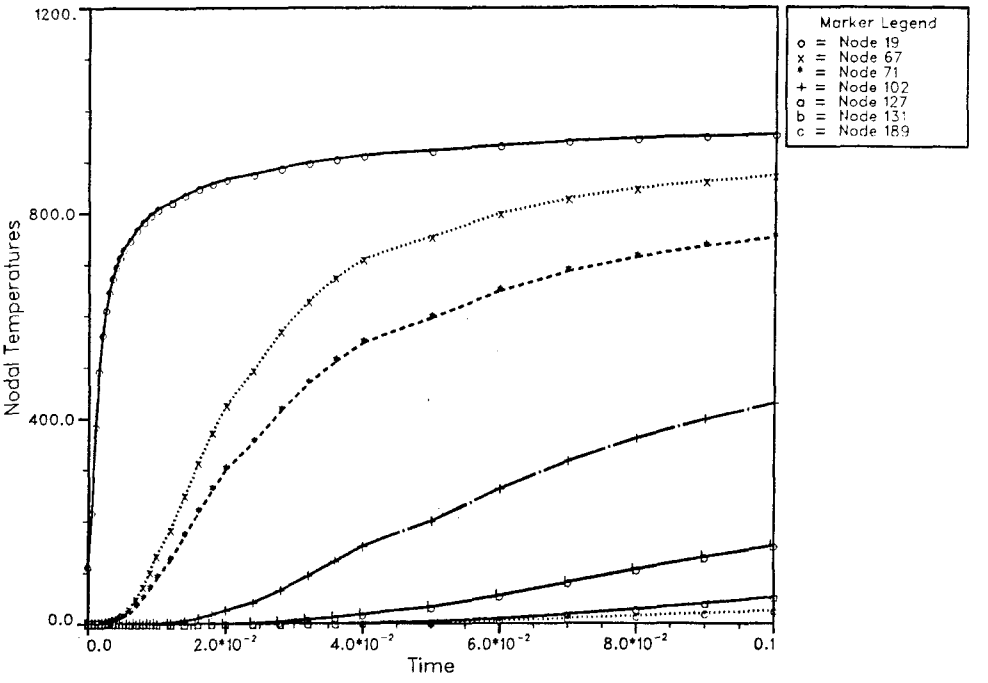


Figure 6.3.5

Arch Problem Run E: Temperature profiles for selected nodes as a function of time.

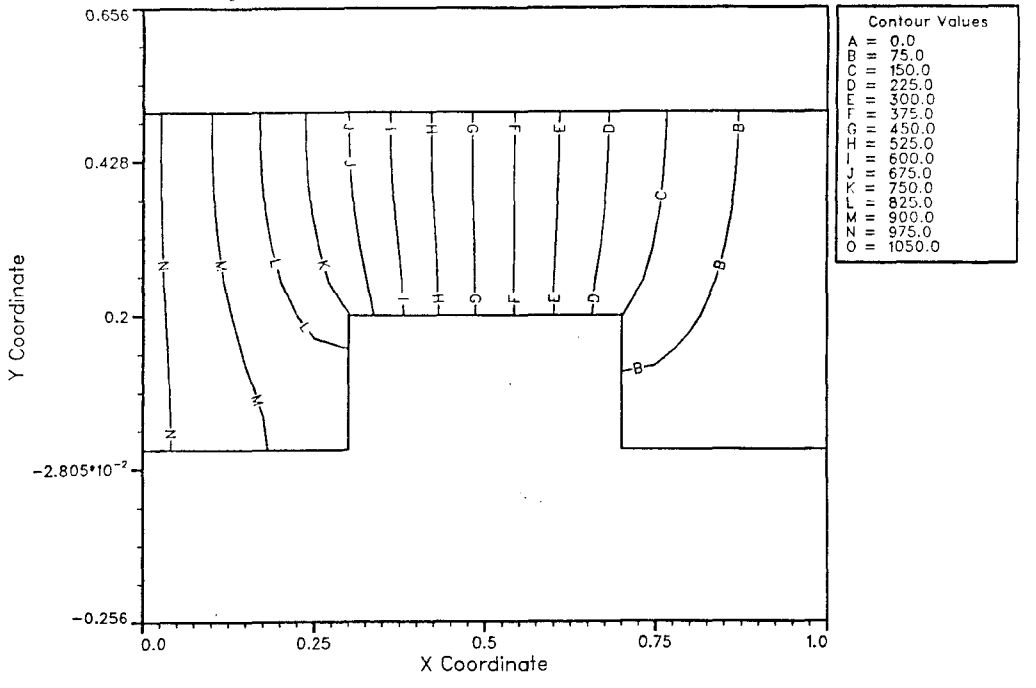


Figure 6.3.6

Arch Problem Run E: Temperature contours at $t = 0.1$ s.

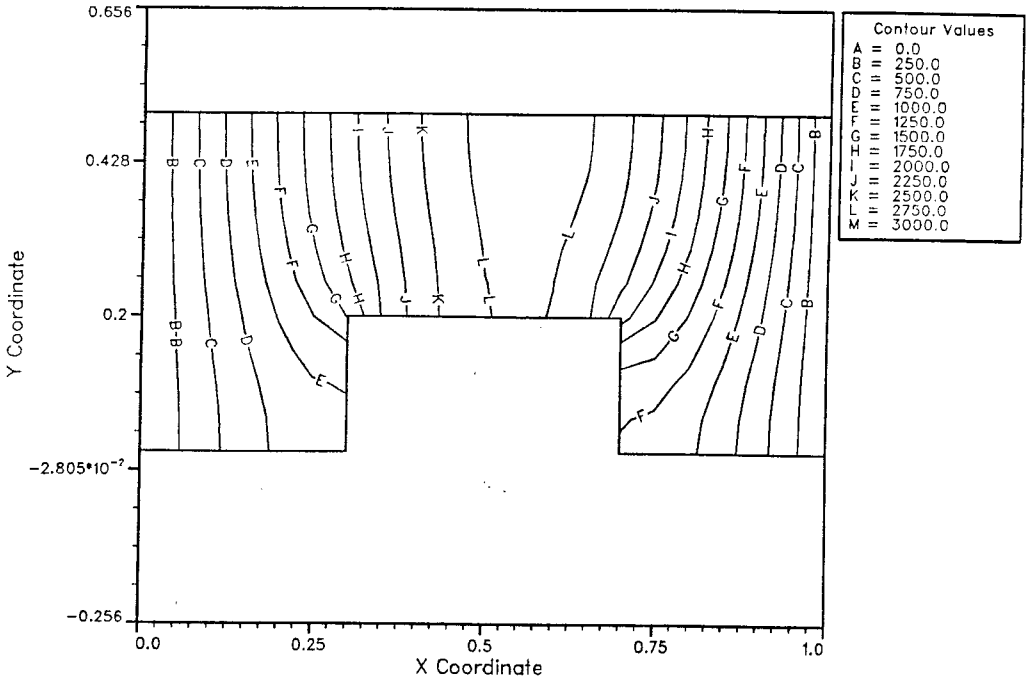


Figure 6.3.7

Arch Problem Run E: Velocity contours at $t = 0.1$ s.

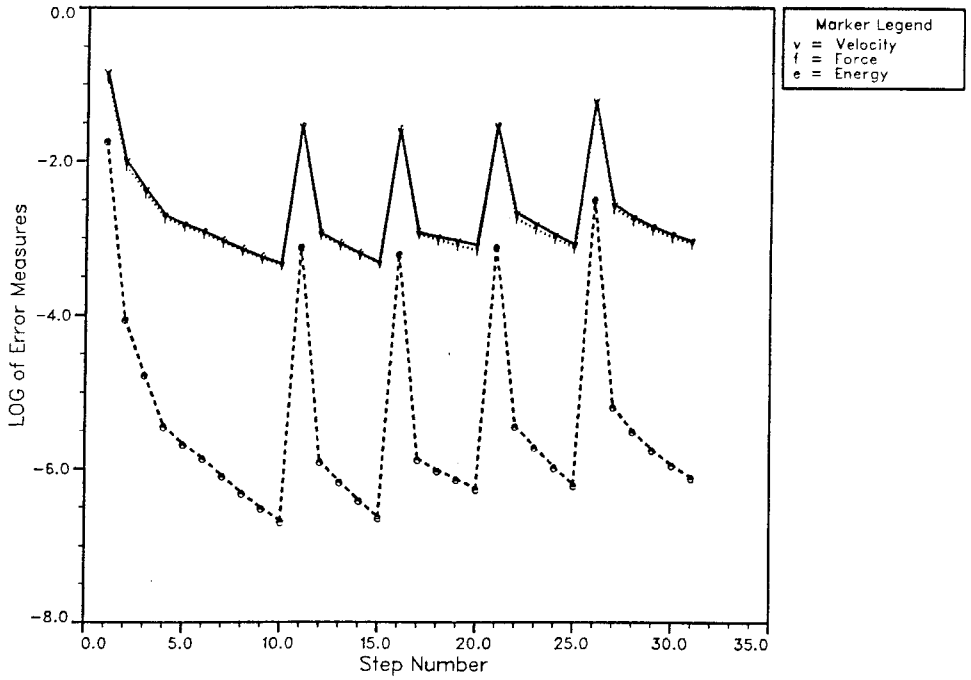


Figure 6.3.8

Arch Problem Run E: Unnormalized step errors based on $\frac{1}{2}$-step residual.

§6.4 Thermal Radiation Problem

The final problem we consider is a thermal radiation problem. Radiation boundary conditions play a very important role in the thermal analysis of space structures [Adelman 82], [Gong 82], [Haftka 82], [Ko 82], [Ried 82]. As a model for problems in this class, we consider a square plate with a nonlinear radiation boundary condition and a significantly nonlinear thermal conductivity. To further increase the problem "difficulty", the radiation boundary condition is discontinuous with respect to time.

The problem spatial domain is a square region defined by :

$$0 \leq x, y \leq 1\,\mathrm{m}\,. \tag{6.4.1}$$

The time range of interest is $t \in [0, 2\,\mathrm{s}]$.

The material properties are: constant density and specific heat,

$$\rho = 1.0 \frac{\mathrm{kg}}{\mathrm{m}^3} \tag{6.4.2}$$

and

$$C_p = 50.0 \frac{\mathrm{W} \cdot \mathrm{s}}{\mathrm{kg} \cdot {}^\circ\mathrm{K}}\,, \tag{6.4.3}$$

strongly nonlinear isotropic thermal conductivity,

$$k = 2\left(1 + 5\frac{T}{1000{}^\circ\mathrm{K}}\right)\frac{\mathrm{W}}{\mathrm{m} \cdot {}^\circ\mathrm{K}}\,, \tag{6.4.4}$$

and a constant heat generation per unit volume,

$$Q = 2500 \frac{\mathrm{W}}{\mathrm{m}^3}\,. \tag{6.4.5}$$

The problem has several different types of boundary conditions. There is a prescribed temperature on the lower edge,

$$T = 0{}^\circ\mathrm{K} \quad \text{on} \quad y = 0\,\mathrm{m},\ t \geq 0\,, \tag{6.4.6}$$

insulation on the upper edge,

$$\mathbf{n} \cdot (\mathbf{K}\nabla T) = 0 \frac{\mathrm{W}}{\mathrm{m}} \quad \text{on} \quad y = 1\,\mathrm{m}, \ t \geq 0\,, \tag{6.4.7}$$

a prescribed constant heat flux on the right edge,

$$\mathbf{n} \cdot (\mathbf{K}\nabla T) = 1000 \frac{\mathrm{W}}{\mathrm{m}} \quad \text{on} \quad x = 1\,\mathrm{m}, \ t \geq 0\,, \tag{6.4.8}$$

and a radiation boundary condition on the left edge

$$\mathbf{n} \cdot (\mathbf{K}\nabla T) = h_r(T^4 - T_r^4) \quad \text{on} \quad x = 0\,\mathrm{m}, \ t \geq 0\,. \tag{6.4.9}$$

The radiation coefficient is

$$h_r = 1.0 \times 10^{-9} \frac{\mathrm{W}}{\mathrm{m}(^\circ\mathrm{K})^4}\,, \tag{6.4.10}$$

and the radiation temperature T_r is a step function in time defined as

$$T_r = \begin{cases} 0^\circ\mathrm{K} & t < 1 \\ 1500^\circ\mathrm{K} & t \geq 1\,. \end{cases} \tag{6.4.11}$$

The initial condition is $T = 0^\circ\mathrm{K}$.

The discretized spatial domain is shown in Figures 6.4.1 and 6.4.1.

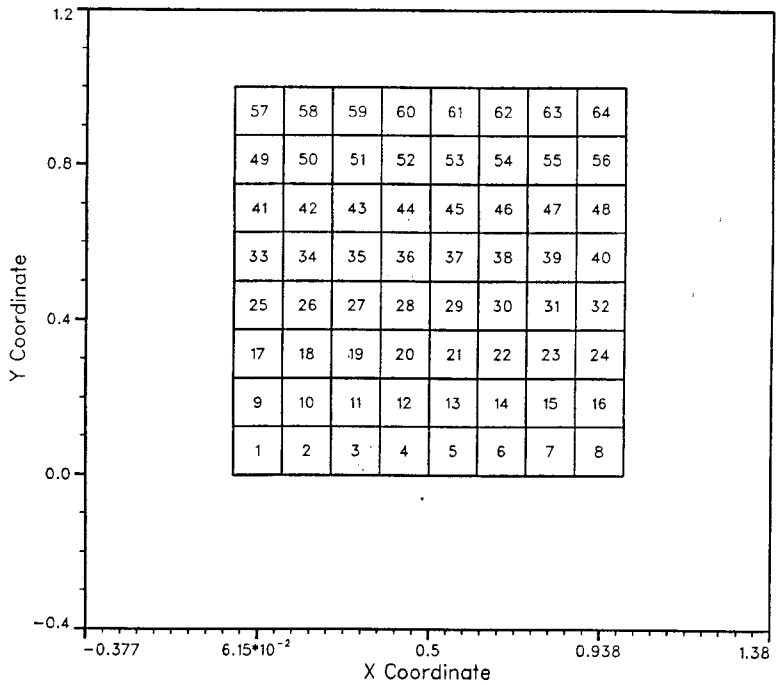


Figure 6.4.1

Radiation Problem: Finite element mesh (element numbers).

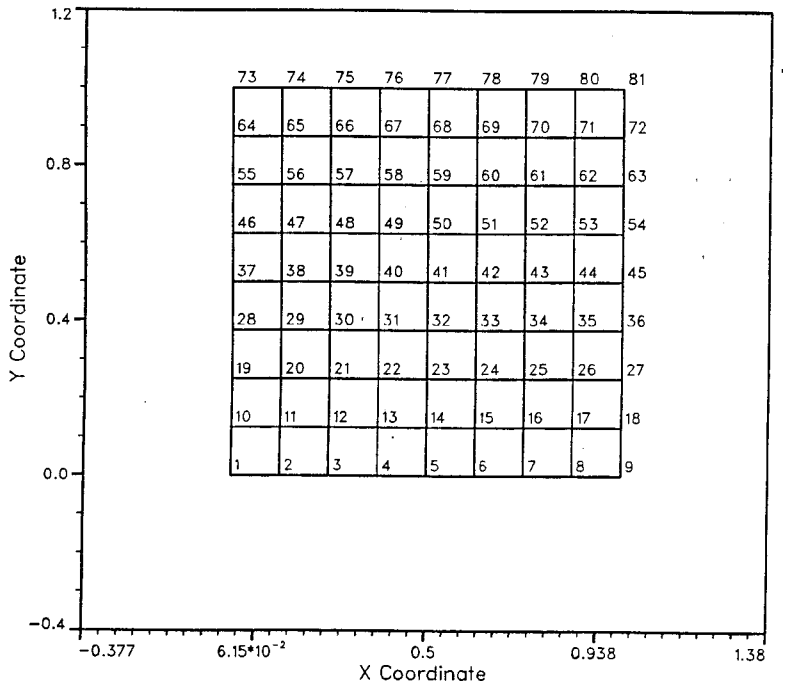


Figure 6.4.2

Radiation Problem: Finite element mesh (node numbers).

We now discuss a series of five computer runs used to evaluate the effectiveness of the EL×EL approximate factorizations on problems with radiation boundary conditions. These runs also demonstrate the effectiveness of the time-stepping selection strategy for use in controlling step errors. All runs used both a step and iteration error tolerance of 10^{-2} and the $\langle PCG, \tilde{\mathbf{A}}_{OPT}, \langle 1, \mathcal{R}, \pi \rangle \rangle$ subiteration algorithm. The subiteration error tolerance was computed using $\beta_j = 1$ coupled with the bail-out procedure. The five runs are defined as follows:

A. Fully automatic selection of Δt for the time interval $0 \geq t \geq 2\,\mathrm{s}$.

B. Two-interval time integration. First, the solution was obtained for the time interval $0 \geq t \geq 1\,\mathrm{s}$. Then, the velocity was recomputed at $t = 1^+$ using the increased radiation temperature. Finally, the integration was continued normally until $t = 2\,\mathrm{s}$.

C. Same as Run B but the second interval was extended to $t = 50\,\mathrm{s}$ (steady state).

D. A high-frequency prescribed nodal heat source was added to the center node (node 41):

$$F_{41}^{ext}(t) = 1000 \sin^2\left(\frac{\pi t}{0.5\,\mathrm{s}}\right) \mathrm{W}\,. \tag{6.4.12}$$

E. Same as Run B but with the radiation coefficient h_r increased by a factor of 100 and the second time interval extended to $t = 10\,\mathrm{s}$ (steady state).

A summary of the costs of the five runs is given in Table 6.4.1.

The time-step size as a function of step number for Run A is shown in Figure 6.4.3. Note the sudden decrease in step size at step 24 ($t = 1\,\mathrm{s}$) as the auto-time-stepping scheme attempts to reduce the normalized step errors shown in Figure 6.4.4. The continuous nature of the temperature is shown in Figure 6.4.5, while the discontinuity in velocity for a node on the radiation boundary is shown in Figure 6.4.6.

Run	Max. time (s)	Total Steps	Avg Iter per Step	Avg Subiter per Iter.
A	2	53	2.00	1.15
B	2	39	2.00	1.21
C	50	141	2.00	2.84
D	2	101	2.00	1.00
E	10	106	2.00	1.78

Table 6.4.1

Radiation Problem: Summary of solution costs for runs A–E.

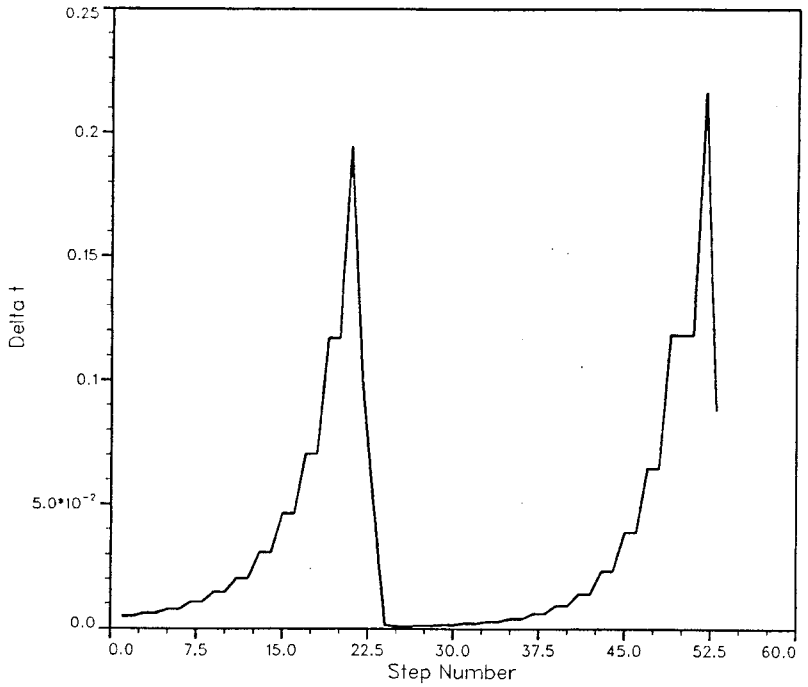


Figure 6.4.3

Radiation Problem Run A: Time-step size as a function of step number.

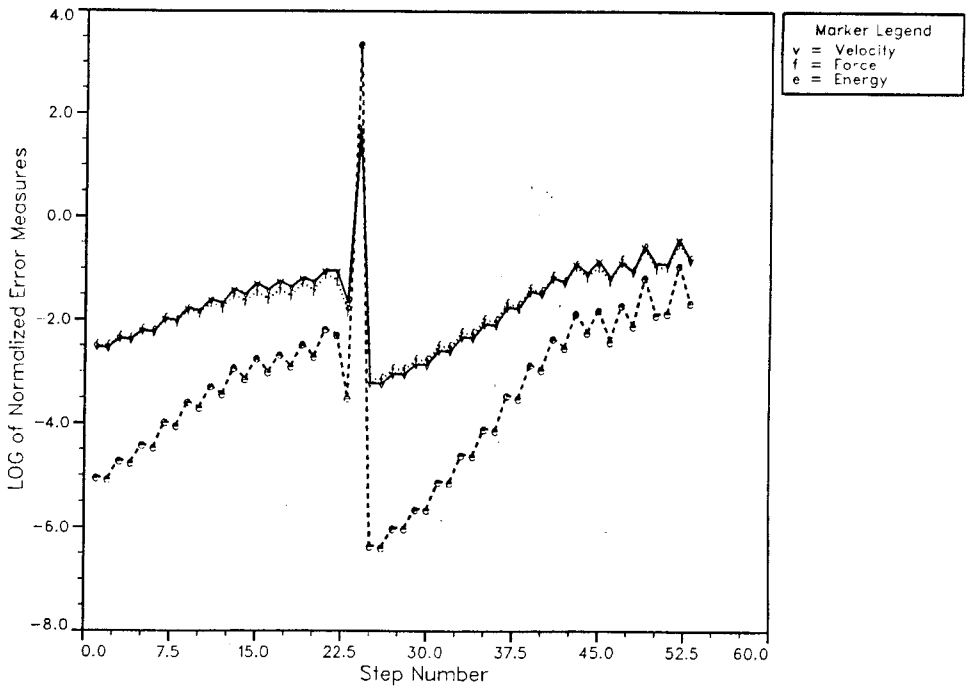


Figure 6.4.4

Radiation Problem Run A: Normalized step errors as a function of step number.

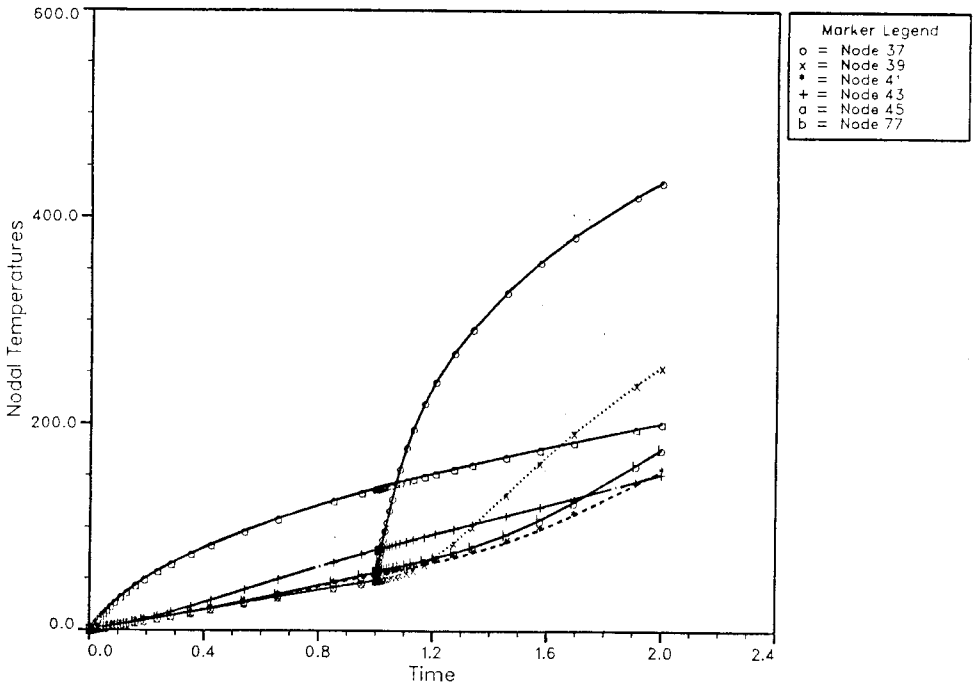


Figure 6.4.5

Radiation Problem Run A: Temperature as a function of time for selected nodes.

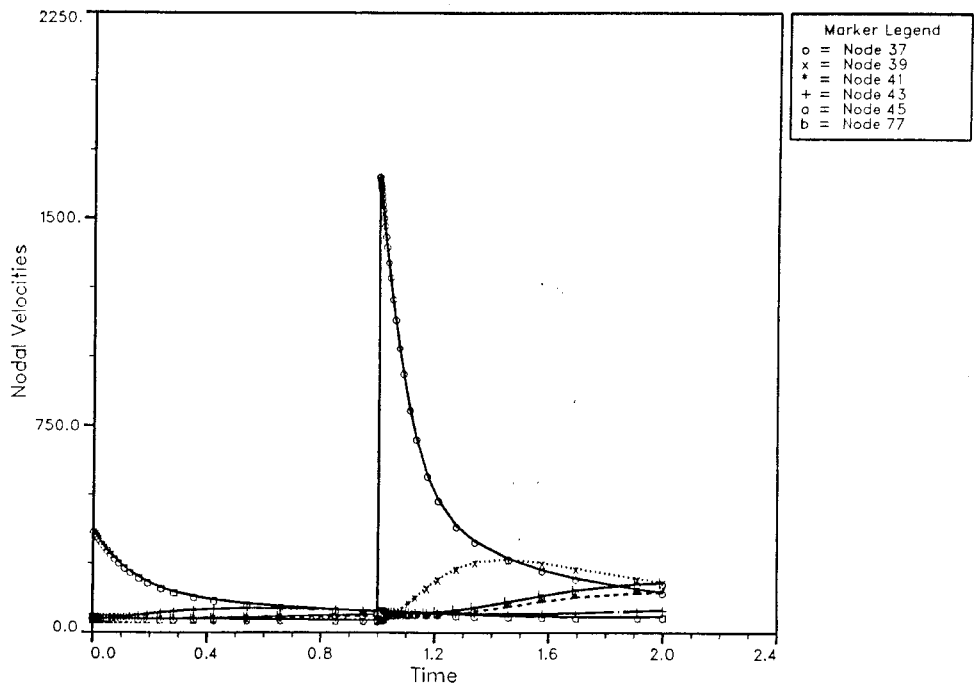


Figure 6.4.6

Radiation Problem Run A: Velocity as a function of time for selected nodes.

In Run B, the number of steps needed to obtain the solution was reduced to 39. Fewer steps were required since the time step did not need to be reduced nearly as much with the discontinuity in velocity accounted for exactly. The temperature and velocity contours at $t = 1\,\mathrm{s}$ are shown in Figures 6.4.7 and 6.4.8, and at $t = 2\,\mathrm{s}$ in Figures 6.4.9 and 6.4.10.

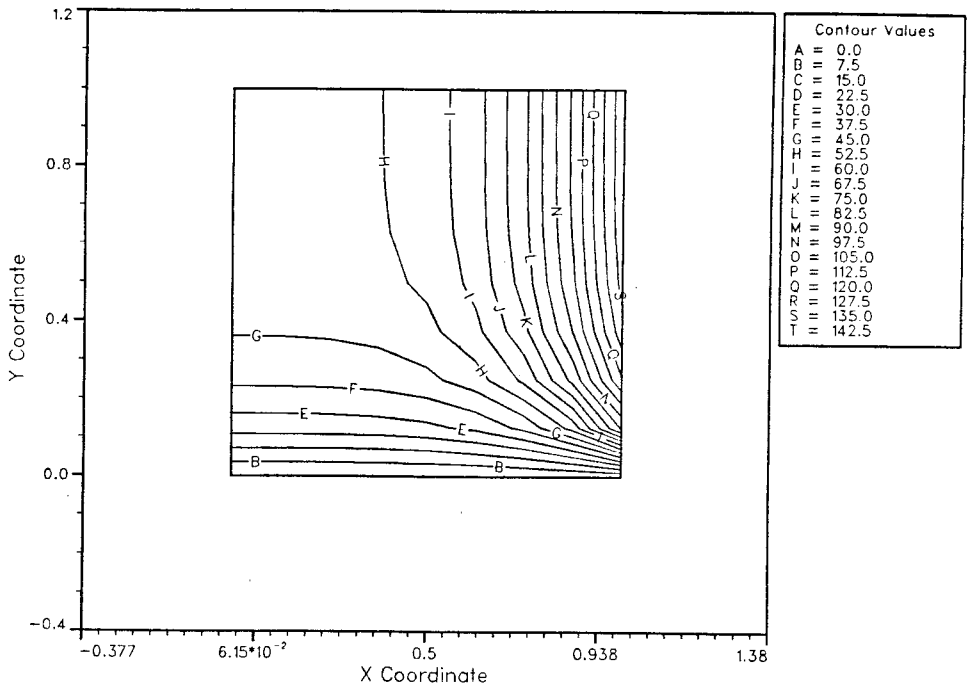


Figure 6.4.7

Radiation Problem Run B: Temperature contours at $t = 1$ s.

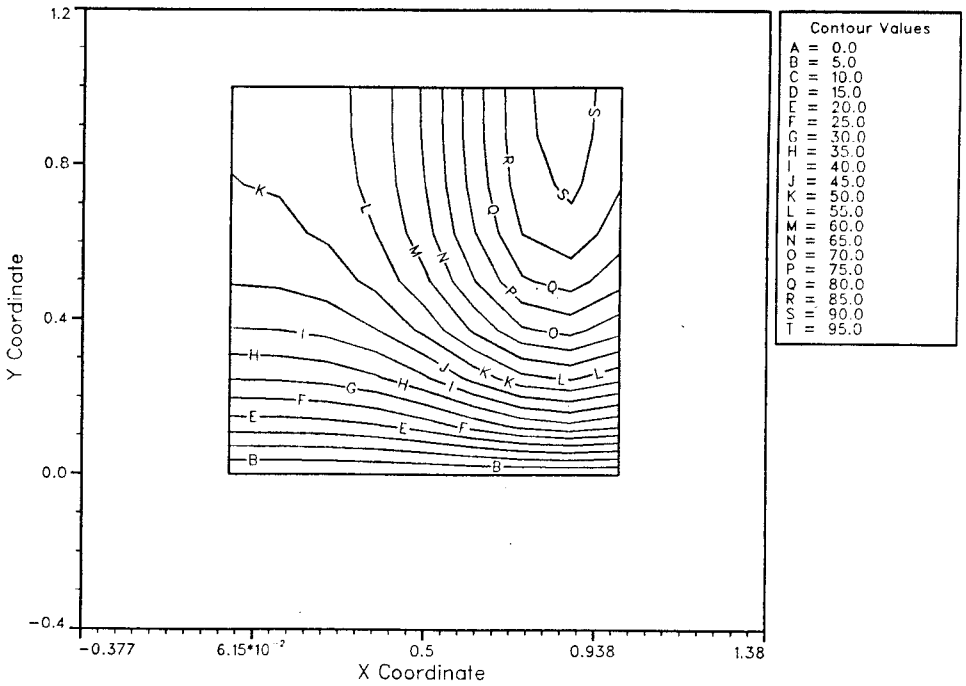


Figure 6.4.8

Radiation Problem Run B: Velocity contours at $t = 1$ s.

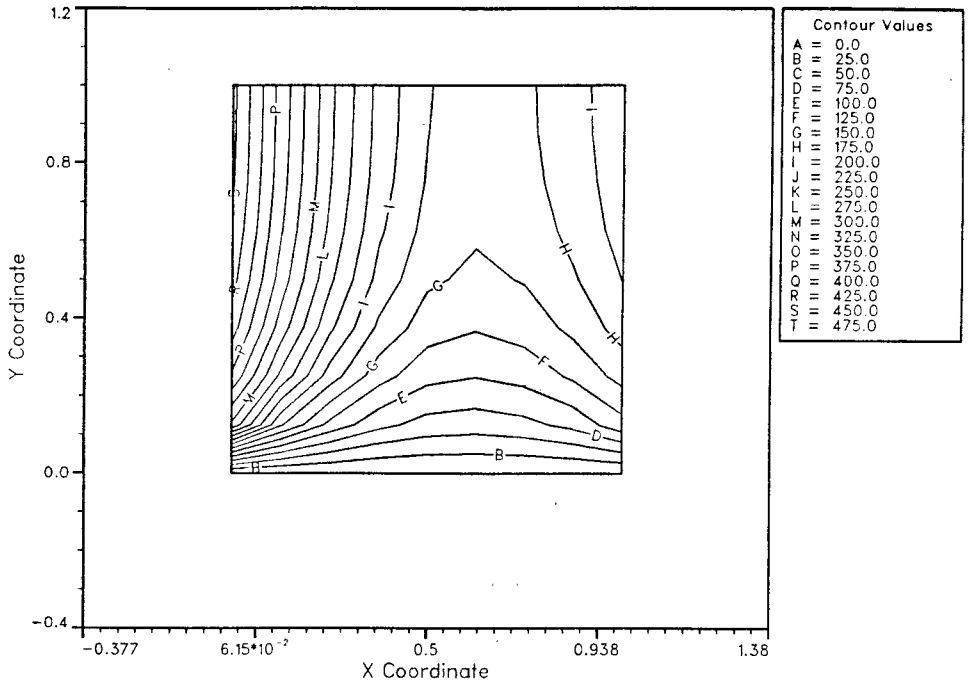


Figure 6.4.9

Radiation Problem Run B: Temperature contours at $t = 2\,\mathrm{s}$.

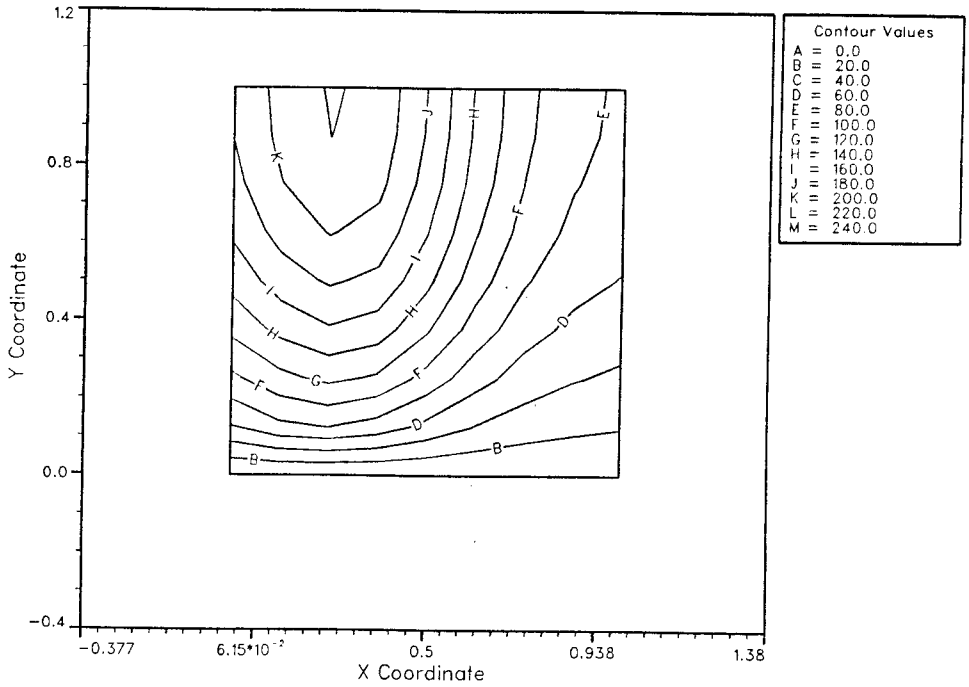


Figure 6.4.10

Radiation Problem Run B: Velocity contours at $t = 2\,\mathrm{s}$.

Run C was used to study the behavior of the subiteration and auto-time-stepping schemes when integrating to steady state. The time step was automatically increased to a maximum value of 0.53 s. This value is approximately 15 times the critical time step for the problem.

In Run D, the addition of a high-frequency heat source forced the auto-time-stepping scheme to repeatedly vary the step size, as shown in Figure 6.4.11, due to the periodic increase in the normalized step errors shown in Figure 6.4.12. The temperatures and velocities as a function of time are shown in Figures 6.4.13 and 6.4.14.

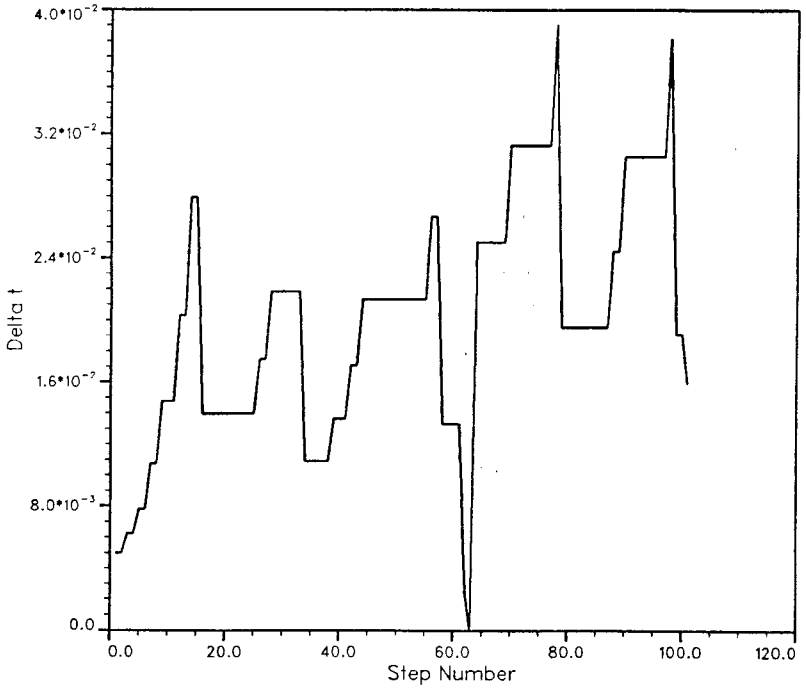


Figure 6.4.11

Radiation Problem Run D: Time-step size as a function of step number.

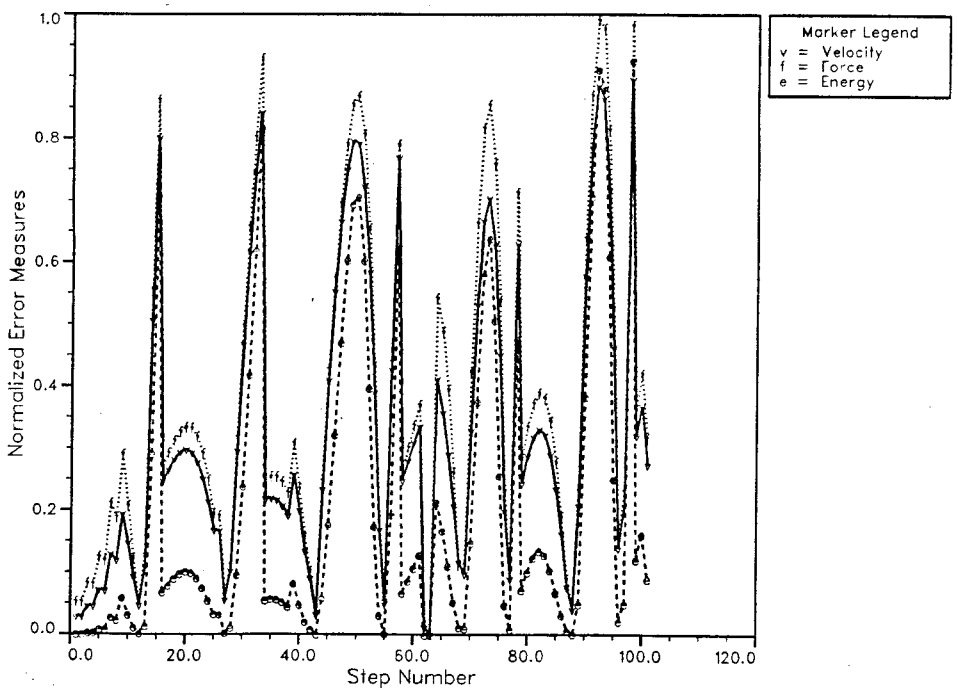


Figure 6.4.12

Radiation Problem Run D: Normalized step errors as a function of step number.

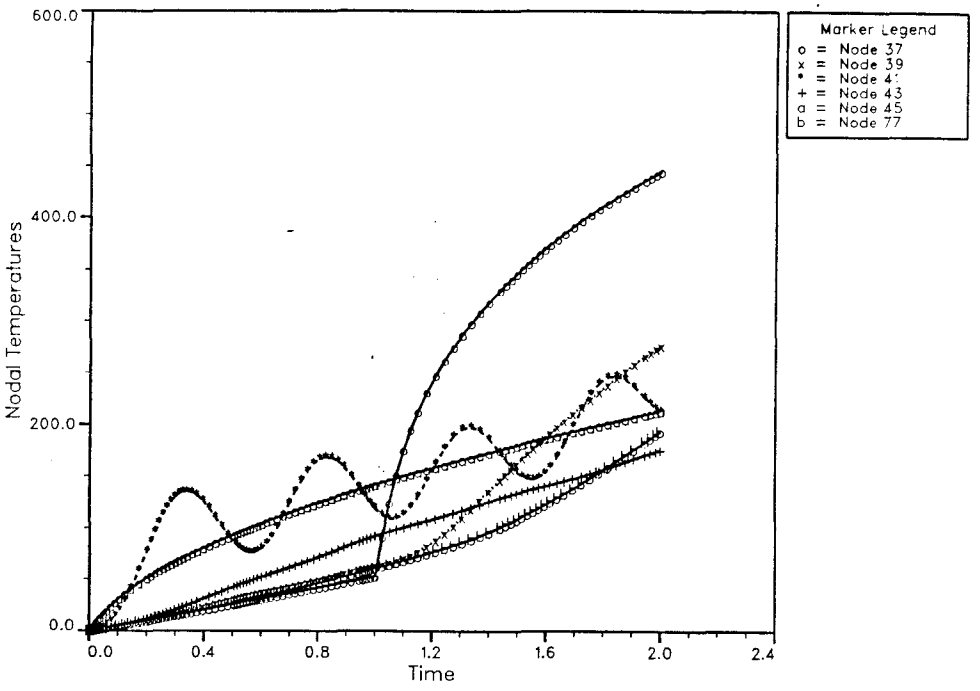


Figure 6.4.13

Radiation Problem Run D: Temperature as a function of time for selected nodes.

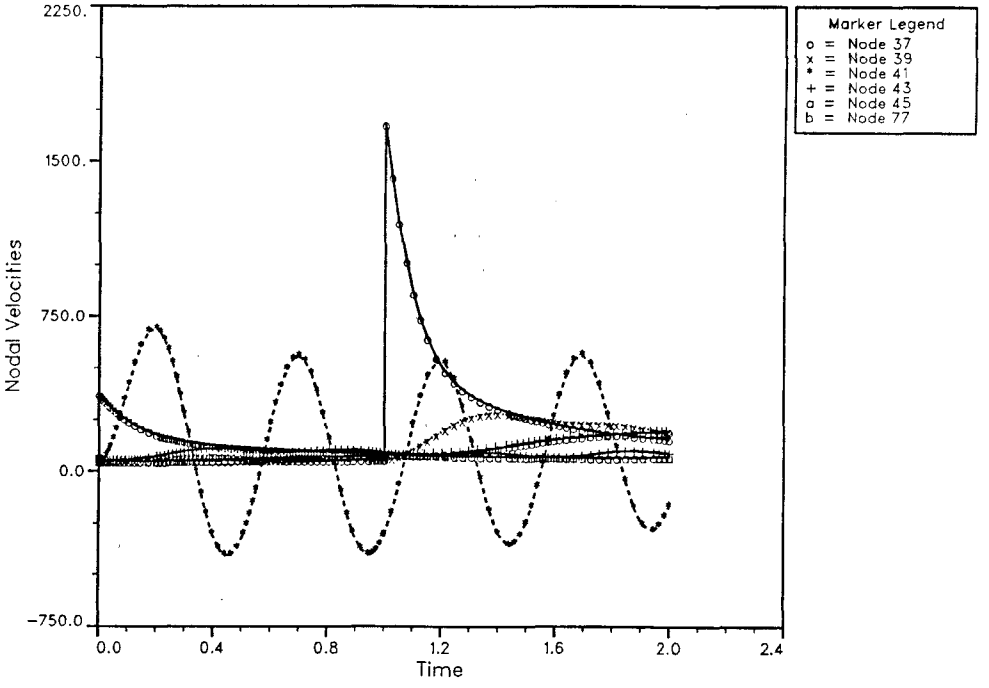


Figure 6.4.14

Radiation Problem Run D: Velocity as a function of time for selected nodes.

Finally, in Run E, the increased radiation coefficient caused significantly slower growth of the step size. The effect of the radiation discontinuity is most dramatic in steps 23 through 40, where the step size remains small due to large normalized step errors as shown in Figures 6.4.15 and 6.4.16. The temperature and velocity as functions of time are shown in Figures 6.4.17 and 6.4.18. Note the magnitude of the velocity discontinuity. For clarification, plots of temperature and velocity vs step number are shown in Figures 6.4.19 and 6.4.20. In obtaining the steady-state solution, the step size was automatically increased to a maximum value of 0.3 s which is 71 times the critical time step.

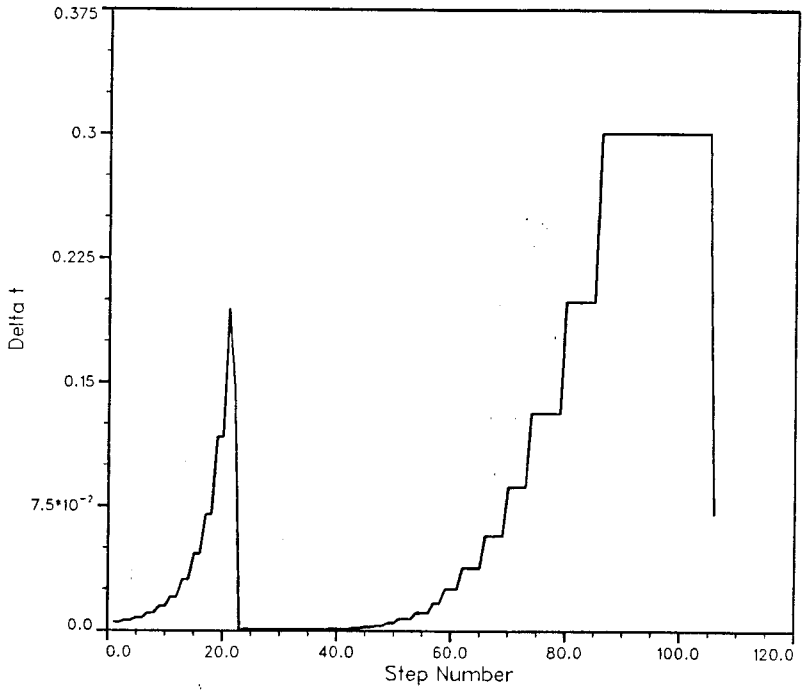


Figure 6.4.15

Radiation Problem Run E: Time-step size as a function of step number.

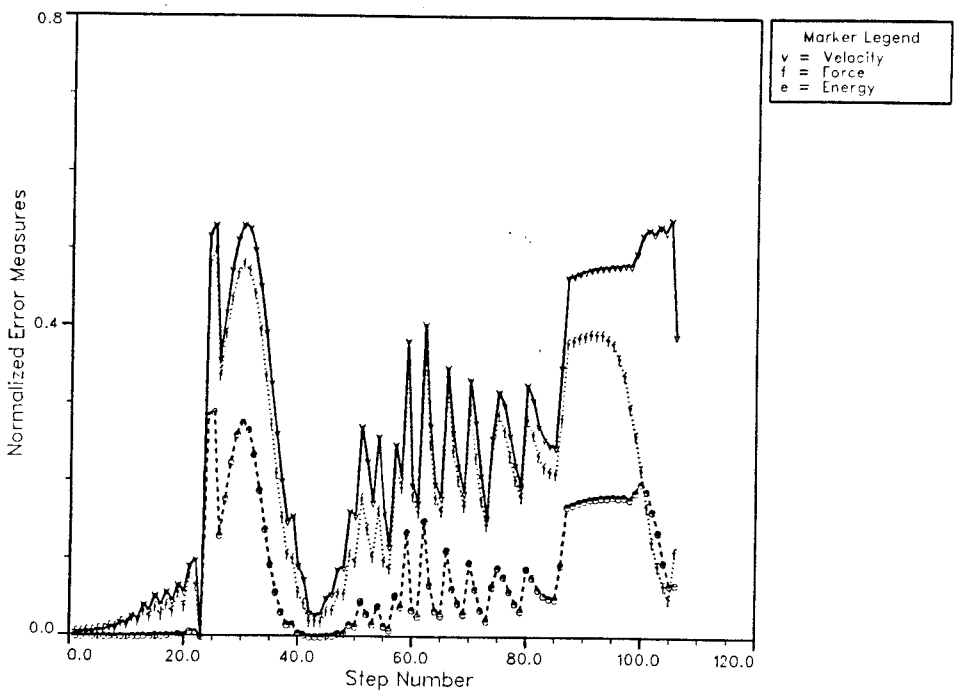


Figure 6.4.16

Radiation Problem Run E: Normalized step errors as a function of step number.

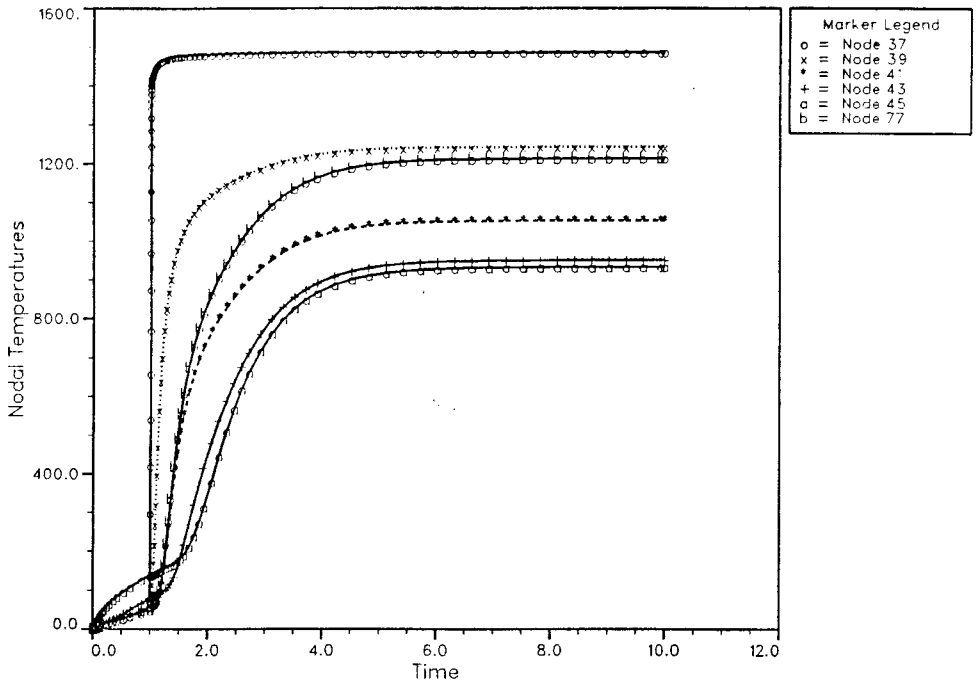


Figure 6.4.17

Radiation Problem Run E: Temperature as a function of time for selected nodes.

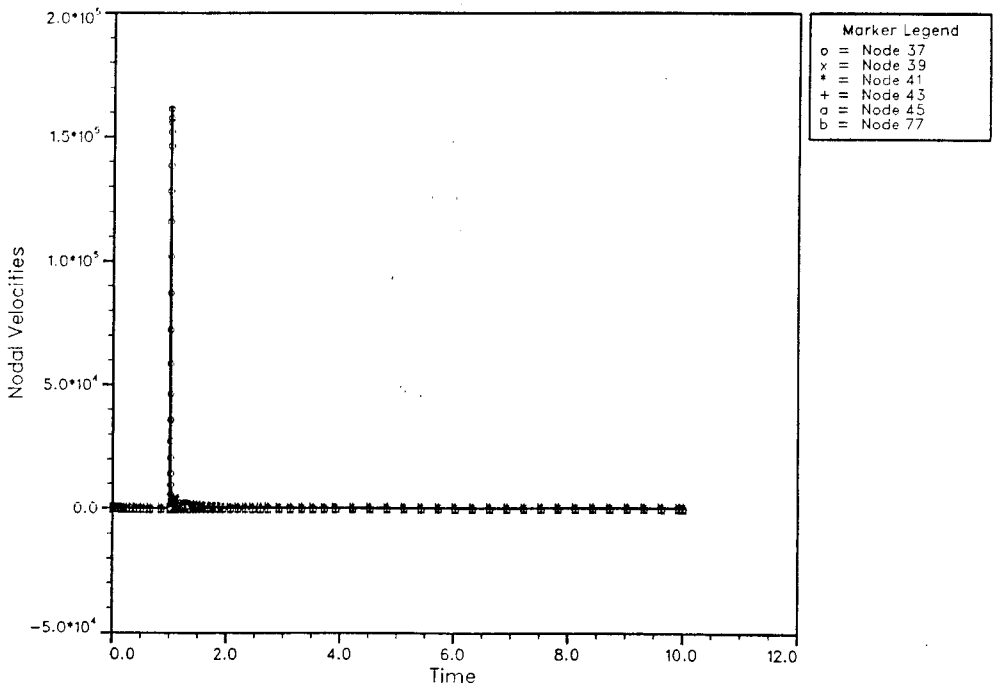


Figure 6.4.18

Radiation Problem Run E: Velocity as a function of time for selected nodes.

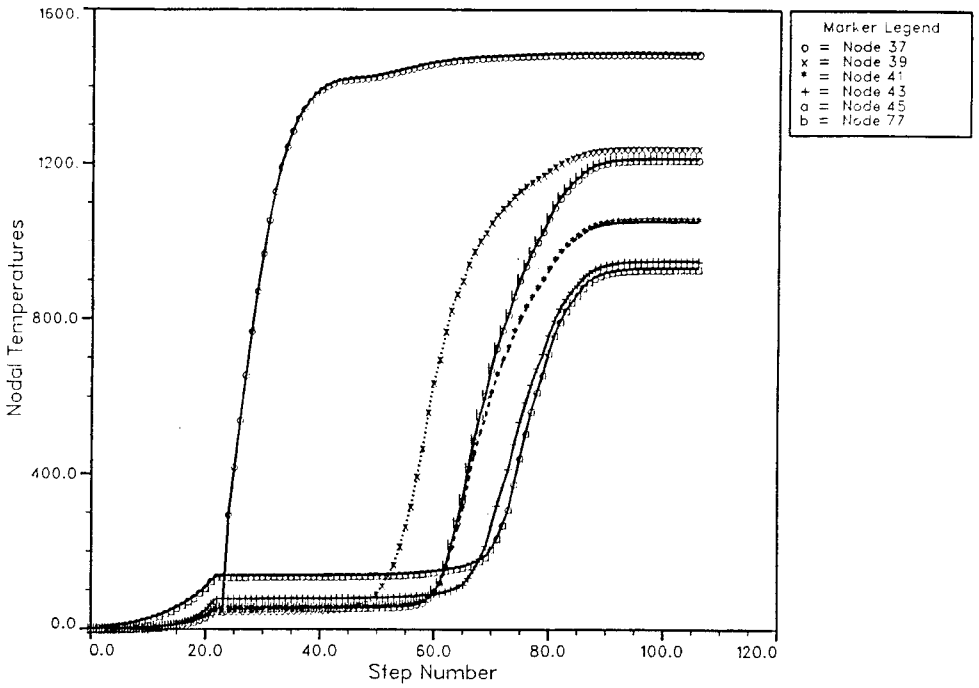


Figure 6.4.19

Radiation Problem Run E: Temperature as a function of step number for selected nodes.

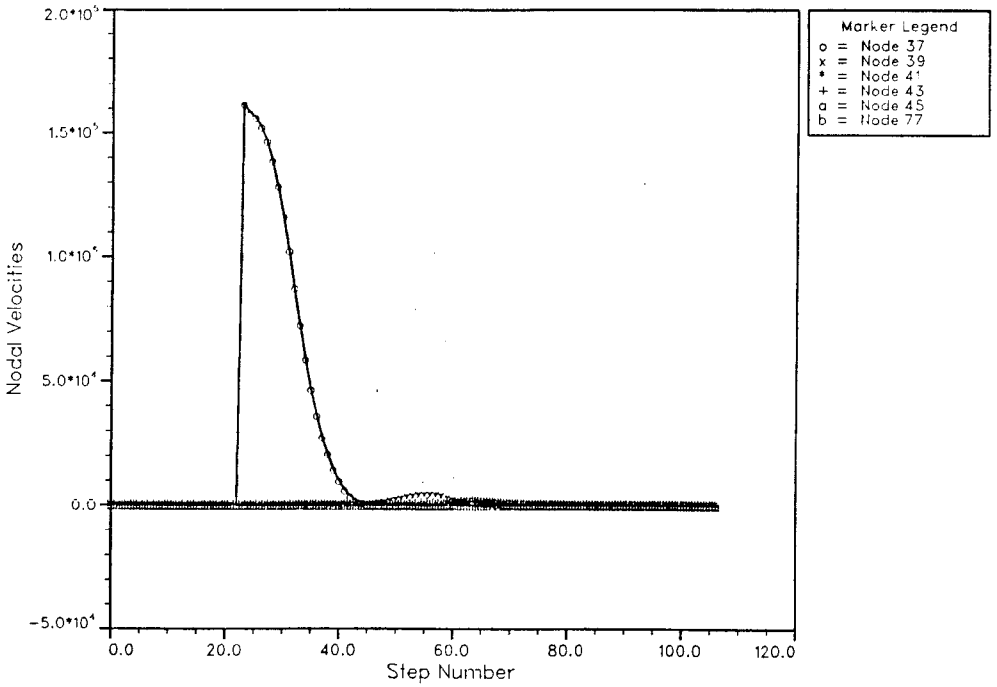


Figure 6.4.20

Radiation Problem Run E: Velocity as a function of step number for selected nodes.

Chapter 7

Conclusions

This work has focused on the development of low-cost techniques for solving the linear equation systems arising from the application of the finite element method to problems in nonlinear transient heat conduction.

These problems are typically stiff due to both the wide range of material properties used in engineering applications and the use of radiation boundary conditions. Thus, an unconditionally stable implicit time integration scheme is required. The direct solution techniques normally used to solve the resultant matrix equations were shown to be impractical as the size of the problem increases. To alleviate these cost constraints, element-by-element subiteration algorithms have been presented. These techniques have been designed to retain the accuracy and stability of the global implicit operator.

The subiteration algorithms we have developed are based on element-by-element preconditioning of standard iteration algorithms. These techniques offer significant cost reductions over traditional direct solution techniques. In particular, we have shown that the solution costs for both storage and CPU grow only linearly with problem size for the EL×EL algorithms presented. This contrasts with the geometric growth entailed by direct solution techniques. In addition, the EL×EL algorithms fit naturally within the architecture of current finite element programs.

We have tested a variety of subiteration algorithms. The iteration techniques used in the tests included Richardson's method, parabolic regularization, the steepest descent technique, and the conjugate gradient technique. These iterative methods were coupled with EL×EL preconditioning using one and two-pass, natural and reordered, EL×EL approximate factorizations based on Crout, Cholesky, and sum decompositions. Also evaluated were two choices for the matrix to be approximated. Our results indicate that the one-pass, reordered, Crout EL×EL factorization, applied to the optimal definition of $\tilde{\mathbf{A}}$ and coupled with a preconditioned conjugate gradient iteration algorithm, is particularly effective for solving symmetric positive-definite matrix systems.

Convergence studies have verified the analysis of error in the approximate factorization performed in Chapter 4. The error analysis indicates that the error in the approximation approaches a constant value as the time step is increased. This is verified by convergence studies, in which the number of subiterations required for convergence approaches a small constant as the time step is increased.

We have demonstrated that the EL×EL solution technique is very effective in solving problems with both material nonlinearities and radiation boundary conditions. This effectiveness is due in part to the coupling developed between the nonlinear iteration level and the linear subiteration level. The usefulness of the algorithm for computing optimal subiteration error criteria is confirmed by the reduced amount of "work" performed at the subiteration-level.

The algorithms are further enhanced by the development of a time-step size selection strategy. This strategy automatically maintains the desired solution accuracy by adjusting the time-step size to account for nonlinearities and temporal discretization errors. Its effectiveness was clearly demonstrated by its ability to accurately integrate over a discontinuous radiation boundary condition.

Several other features of EL×EL algorithms have been discussed, including the ease with which selective formation/factorization may be performed, the concept of

time-adaptive implicit-explicit algorithms, and the considerable potential of EL×EL adaptive mesh refinement. Also discussed was the notion of minimal cost substructuring, which led to the idea of an alternating direction EL×EL approximate factorization.

Finally, we have developed an EL×EL algorithm suitable for use on multi-processor computers. The simulations of parallel computations performed on the test problems indicate that there is very little penalty incurred by the use of parallel orderings. This result is very exciting, particularly since it seems clear that the future of large calculations on super computers lies in parallelizable algorithms.

References

[Adelman 82]
Howard M. Adelman, Raphael T. Haftka, and **James C. Robinson**, "Some Aspects of Algorithm Performance and Modeling in Transient Thermal Analysis of Structures," *Computational Aspects of Heat Transfer in Structures*, NASA Conference Publication 2216, (1982).

[Akin 82]
J. E. Akin, *Application and Implementation of Finite Element Methods*, Academic Press, New York, (1982).

[Ames 77]
W. F. Ames, *Numerical Methods for Partial Differential Equations*, second edition, Academic Press, New York, (1977).

[Bakhvalov 77]
N. S. Bakhvalov, *Numerical Methods*, Mir Publishers, Moscow, (1977).

[Bathe 80]
Klaus-Jürgen Bathe and **Arthur P. Cimento**, "Some Practical Procedures for the Solution of Nonlinear Finite Element Equations," *Computer Methods in Applied Mechanics and Engineering*, Vol. 22, pp 59–85, (1980).

[Bathe 82]
Klaus-Jürgen Bathe, *Finite Element Procedures in Engineering Analysis*, Prentice–Hall Inc., New Jersey, (1982).

[Bell 76]
Graham E. Bell and **John Crank**, "A Simple Finite-Difference Modification for Improving Accuracy Near a Corner in Heat flow Problems," *International Journal for Numerical Methods in Engineering*, Vol. 10, pp 827–832 (1976).

[Belytschko 83]
T. Belytschko and **W. K. Liu**, "On Reduced Matrix Inversion for Operator Splitting Methods," preprint, (1983).

[Bruch 74]
John C. Bruch and **George Zyvoloski**, "Transient Two-Dimensional Heat Conduction Problems Solved by the Finite Element Method," *International Journal for Numerical Methods in Engineering*, Vol. 8, pp 481–494 (1974).

[Carey 81]
Graham F. Carey and **David L. Humphrey**, "Mesh Refinement and Iterative Solution Methods for Finite Element Computations," *International Journal for Numerical Methods in Engineering*, Vol. 17, pp 1717–1734 (1981).

[Carslaw 59]
H.S. Carslaw and **J.C. Jaeger**, *Conduction of Heat in Solids*, Oxford University Press, Oxford, (1959).

[Dahlquist 74]
Germund Dahlquist and **Åke Björck**, *Numerical Methods*, Prentice–Hall Inc., New Jersey, (1974).

[Douglas 56]
J. Douglas and **H. H. Rachford**, "On the Numerical Solution of Heat Conduction Problems in Two and Three Space Variables," *Trans. Amer. Math. Soc.*, Vol. 82, pp 421–439, (1956).

[Douglas 62]
J. Douglas, "Alternating Direction Methods for Three Space Variables," *Numer. Math.*, Vol. 4, pp 41–63, (1962).

[Emery 79]
A. F. Emery, K. Sugihara, and **A. T. Jones**, "A Comparison of Some of the Thermal Characteristics of Finite Element and Finite Difference Calculations of Transient Problems," *Numerical Heat Transfer*, Vol. 2, pp 97–113, (1979).

[Emery 82]
A. F. Emery and **H. R. Mortazvi**, "A Comparison of Finite Difference and Finite Element Methods for Heat Transfer Calculations," *Computational Aspects of Heat Transfer in Structures*, NASA Conference Publication 2216, (1982).

[Evans 82]
David J. Evans, *Parallel Processing Systems*, Cambridge University Press, Cambridge, (1982).

[Fellipa 75]
Carlos A. Felippa, "Solution of Linear Equations with Skyline–Stored Symmetric Matrix," *Computers & Structures*, Vol. 5, pp 13–29, (1975).

[Fox 83]
Geoffrey C. Fox and **Charles L. Seitz**, "Concurrent Processing and the Decomposition of Problems," Research Proposal Submitted to the Department of Energy, CALT-68-1004.

[George 81]
Alan George and **Joseph W. Liu**, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice–Hall Inc., New Jersey, (1981).

[Gill 81]
Philip E. Gill, Walter Murray and **Margaret H. Wright**, *Practical Optimization*, Academic Press, New York, (1981).

[Gong 82]
Leslie Gong, Robert D. Quinn, and **William L. Ko**, "Reentry Heating Analysis of Space Shuttle with Comparision of Flight Data," *Computational Aspects of Heat Transfer in Structures*, NASA Conference Publication 2216, (1982).

[Gresho 79]
P. M. Gresho and **R. L. Lee**, "Don't Suppress the Wiggles — They're Telling You Something!," *Finite Element Methods for Convection Dominated Flows*, T. J. R. Hughes (ed.), Amd Vol. 34, ASME, New York, (1979).

[Haftka 82]
Raphael T. Haftka, and **M. Hassan Kadivar**, "Algorithmic Aspects of Transient Heat Transfer Problems in Structures," *Computational Aspects of Heat Transfer in Structures*, NASA Conference Publication 2216, (1982).

[Hageman 81]
Louis A. Hageman and **David M. Young**, *Applied Iterative Methods*, Academic Press, New York, (1981).

[Hestenes 52]
M. R. Hestenes and **E. Steifel**, "Methods of Conjugate Gradients for Solving Linear Systems," *Journal of Research of National Bureau of Standards*, Vol. 49, No. 6, pp 409–436, December 1952.

[Hibbitt 79]
H. D. Hibbitt and **B. I. Karlsson**, "Analysis of Pipe Whip," ASME Paper No. 79-PVP-122, presented at the Pressure Vessels and Piping Conference, San Francisco, California, June 25-29, 1979.

[Hockney 81]
R. W. Hockney and **C. R. Jesshope**, *Parallel Computers*, Adam Hilger Ltd, Bristol, (1981).

[Holman 72]
J. P. Holman, *Heat Transfer*, McGraw–Hill Book Company, New York, (1972).

[Householder 64]
Alston S. Householder, *The Theory of Matrices in Numerical Analysis*, Dover Publications Inc., New York, (1964).

[Hughes 77]
Thomas J.R. Hughes, "Stability of One-Step Methods in Transient Nonlinear Heat Conduction," Presented at the International Conference on Structural Mechanics in Reactor Technology, San Francisco, California, August 1977.

[Hughes 78a]
Thomas J.R. Hughes and **W. K. Lui**, "Implicit-Explicit Finite Element Techniques in Transient Analysis: Stability Theory," *Journal of Applied Mechanics*, Vol. 45, pp 371–374, (1978).

[Hughes 78b]
Thomas J.R. Hughes and **W. K. Lui**, "Implicit-Explicit Finite Element Techniques in Transient Analysis: Implementation and Numerical Examples," *Journal of Applied Mechanics*, Vol. 45, pp 375–378, (1978).

[Hughes 79]
Thomas J.R. Hughes, K. S. Pister and **R. L. Taylor**, "Implicit-Explicit Finite Elements and Nonlinear Transient Analysis," *Computer Methods in Applied Mechanics and Engineering*, Vols. 17/18, pp 159–182, (1979).

[Hughes 81]
Thomas J.R. Hughes and **R. A. Stephenson**, "Convergence of Implicit-Explicit Finite Element Algorithms in Nonlinear Transient Analysis," *International Journal of Engineering Science*, Vol. 19, pp 295–302, (1981).

[Hughes 82a]
Thomas J.R. Hughes, "Implicit-Explicit Finite Element Techniques for Symmetric and Non-symmetric Systems," *Recent Advances in Non-linear Computational Mechanics*, Pineridge Press, Swansea, U.K., (1982).

[Hughes 82b]
Thomas J.R. Hughes, James Winget, and **Itzhak Levit**, "Element-by-Element Solution Procedures for Nonlinear Structural Analysis," *Proc. of the NASA Lewis Workshop*, April 19-20, 1982.

[Hughes 83a]
Thomas J.R. Hughes, Itzhak Levit, and **James Winget**, "Implicit, Unconditionally Stable, Element-by-Element Algorithms for Heat Conduction Analysis," *Journal of the Engineering Mechanics Division*, ASCE, Vol. 109, No. 2, pp 576–585 (1983).

[Hughes 83b]
Thomas J.R. Hughes, Itzhak Levit, and **James Winget**, "An Element-by-Element Solution Algorithm for Problems of Structural and Solid Mechanics," *Computer Methods in Applied Mechanics and Engineering*, Vol. 36, No. 2, pp 241–254 (1983).

[Hughes 83c]
Thomas J.R. Hughes, James Winget, Itzhak Levit, and **Tayfun Tezduyar**, "New Alternating Direction Procedures in Finite Element Analysis Based Upon EBE Approximate Factorizations," pp 75–109 in *Proc. of Symposium on Recent Developments in Computer Methods for Nonlinear Solid and Structural Mechanics* eds. S. Atluri and N. Perrone, ASME meeting, University of Houston, Houston, Texas, June 20-22, 1983.

[Isachenko 75]
V.P. Isachenko, V.A. Osipova and **A.S. Sukomel**, *Heat Transfer*, Mir Publishers, Moscow, (1975).

[Jennings 77]
Alan Jennings, *Matrix Computation for Engineers and Scientists*, John Wiley & Sons, New York, (1977).

[Juba 82]
Susan M. Juba and **Peter E. Fogerson**, "Development of a CRAY 1 Version of the SINDA Program," *Computational Aspects of Heat Transfer in Structures*, NASA Conference Publication 2216, (1982).

[Kamel 78]
H. A. Kamel and **M. W. McCabe**, "Direct Numerical Solution of Large Sets of Simultaneous Equations," *Computers & Structures*, Vol. 9, pp 113–123, (1978).

[Ko 82]
William L. Ko, Robert D. Quinn, Leslie Gong, Lawrence S. Schuster, and **David Gonzales**, "Reentry Heat Transfer Analysis of the Space Shuttle Orbiter," *Computational Aspects of Heat Transfer in Structures*, NASA Conference Publication 2216, (1982).

[Levit 82]
Itzhak Levit, "A General Solution Strategy for Large Scale Static and Dynamic Nonlinear Finite Element Problems Employing the Element-by-Element Factorization Concept," PhD Thesis, California Institute of Technology, Pasadena, California, (1982).

[Liu 82]
Wing Kam Liu, "Development of Mixed Time Partition Procedures for Thermal Analysis of Structures," *Computational Aspects of Heat Transfer in Structures*, NASA Conference Publication 2216, (1982).

[Marchuk 74]
G. I. Marchuk, *Numerical Methods in Weather Predicition*, Academic Press, New York and London, (1974).

[Marchuk 75]
G. I. Marchuk, *Methods of Numerical Mathematics*, Springer–Verlag, New York, (1975).

[Mitchell 77]
A. R. Mitchell and **R. Wait**, *The Finite Element Method in Partial Differential Equations*, John Wiley & Sons, New York, (1977).

[Mitchell 80]
A. R. Mitchell and **D. F. Griffiths**, *The Finite Difference Method in Partial Differential Equations*, John Wiley & Sons, New York, (1980).

[Narasimhan 77]
T. N. Narasihan, S. P. Neuman, and **A. L. Edwards**, "Mixed Explicit-Implicit Iterative Finite Element Scheme for Diffusion-Type Problems: II. Solution Strategy and Examples," *International Journal for Numerical Methods in Engineering*, Vol. 11, pp 325–344 (1977).

[Noor 82]
Ahmed K. Noor, "Survey of Computer Programs for Heat Transfer Analysis," *Computational Aspects of Heat Transfer in Structures*, NASA Conference Publication 2216, (1982).

[Nour-Omid 82]
B. Nour-Omid and **B. N. Parlett**, "Element Preconditioning," Report PAM-103, Center for Pure and Applied Mathematics, University of California, Berkeley, October 1982.

[Ozisik 80]
M. Necati Özişik, *Heat Conduction*, John Wiley & Sons, New York, (1980).

[Park 79]
K. C. Park and **P. G. Underwood**, "A Variable-Step Central Difference Method for Structural Analysis–Part I: Theoretical Aspects," ASME Paper, presented at the Pressure Vessels and Piping Conference, San Francisco, California, June 25-29, 1979.

[Park 82]
K. C. Park, "Semi-Implicit Transient Analysis Procedure for Structural Dynamics Analysis," *International Journal for Numerical Methods in Engineering*, Vol. 18, pp 604–622, (1982).

[Ried 82]
Robert C. Ried, Winston D. Goodrich, Chien P. Li, Carl D. Scott, Stephen M. Derry, and **Robert J. Maraia**, "Space Shuttle Orbiter Entry Heating and TPS Response: STS–1 Predictions and Flight Data," *Computational Aspects of Heat Transfer in Structures*, NASA Conference Publication 2216, (1982).

[Robinson 82]
J. C. Robinson, K. M. Riley, and **R. T. Haftka**, "Evaluation of the SPAR Thermal Analyzer on the CYBER–203 Computer," *Computational Aspects of Heat Transfer in Structures*, NASA Conference Publication 2216, (1982).

[Siegel 81]
Robert Siegel and **John R. Howell**, *Thermal Radiation Heat Transfer*, McGraw–Hill, New York, (1981).

[Strang 73]
Gilbert Strang and **George J. Fix**, *An Analysis of the Finite Element Method*, Prentice–Hall Inc., New Jersey, (1973).

[Taylor 75]
Robert L. Taylor, "*HEAT*, A Finite Element Computer Program for Heat-Conduction Analysis," Report 75-1, Prepared for: Civil Engineering Laboratory, Naval Construction Battalion Center, Port Hueneme, California, May 1975.

[Trujillo 77a]
D. M. Trujillo, "An Unconditionally Stable, Explicit Algorithm for Finite Element Heat Conduction Analysis," *Journal of Nuclear Engineering and Design*, Vol. 41, pp 175–180, (1977).

[Trujillo 77b]
D. M. Trujillo and **H. R. Busby**, "Finite Element Nonlinear Heat Transfer Analysis using a Stable Explicit Method," *Journal of Nuclear Engineering and Design*, Vol. 44, pp 227–233, (1977).

[Trujillo 77c]
D. M. Trujillo, "An Unconditionally Stable, Explicit Algorithm for Structural Dynamics," *International Jouranl for Numerical Methods in Engineering*, Vol. 11, pp 1579–1592, (1977).

[Trujillo 82]
D. M. Trujillo and **H. R. Busby**, "Investigation of Highly Accurate Integration Formulae for Transient Heat Conduction Analysis Using the Conjugate Gradient Method," *International Journal for Numerical Methods in Engineering*, Vol. 18, pp 99–109, (1982).

[Underwood 79]
P. G. Underwood and **K. C. Park**, "A Variable-Step Central Difference Method for Structural Analysis–Part II: Implementation and Performance Evaluation," ASME Paper, presented at the Pressure Vessels and Piping Conference, San Francisco, California, June 25-29, 1979.

[Vemuri 81]
V. Vemuri and **Walter J. Karplus**, *Digital Computer Treatment of Partial Differential Equations*, Prentice–Hall Inc., New Jersey, (1981).

[Wachpress 66]
E. L. Wachpress, *Iterative Solution of Elliptic Systems and Applications to the Neutron Diffusion Equations of Reactor Physics*, Prentice–Hall Inc., New Jersey, (1966).

[Winget 82]
James M. Winget and **Thomas J. R. Hughes**, "A Profile Solver for Specially Structured Symmetric-Unsymmetric Equation Systems," *Advances in Engineering Software*, Vol. 4 No. 2, pp 64–67, (1982).

[Yanenko 71]
N. N. Yanenko, *The Method of Fractional Steps*, Springer–Verlag, New York–Heidelberg–Berlin, (1971).

[Zienkiewicz 77]
O. C. Zienkiewicz, *The Finite Element Method*, McGraw–Hill Book Company, London, (1977).

Index