Copyright (c) 2018, by the author.

This work was supported by the National Science Foundation under grants DMS-1318480, DMS-1345103, and MRI-0821816.

This software is made available for research and instructional use only. You may copy and use this software without charge for these non-commercial purposes, provided that the copyright notice and associated text is reproduced on all copies. For all other uses (including distribution of modified versions), please contact the author. This software is provided “as is”, without any expressed or implied warranty. In particular, the author does not make any representation or warranty of any kind concerning the fitness of this software for any particular purpose.
To Barbara
Contents

Preface ix

1 Introduction 1
  1.1 Problem Specification. ......................................... 1
    1.1.1 Approximation Spaces. ................................... 2
    1.1.2 Elliptic Boundary Value Problem. ......................... 3
    1.1.3 Obstacle Problem. ....................................... 3
    1.1.4 Continuation Problem. ................................... 4
    1.1.5 Parameter Identification Problem. ....................... 4
    1.1.6 Optimal Control Problem. ................................ 5
  1.2 Main Subroutines ............................................. 6
  1.3 Installation. .................................................. 7

2 Data Structures 9
  2.1 Overview. ..................................................... 9
  2.2 Edge Definitions ..............................................
    2.2.1 Curved Edges – Circular Arcs ............................. 10
    2.2.2 Curved Edges – Parametric ................................ 11
  2.3 The Triangulation. ............................................ 13
  2.4 The Skeleton. ............................................... 14
  2.5 Finite Element Data Structures. ............................... 19
  2.6 Parallel Processing Data Structure. .......................... 21
  2.7 Parameter Arrays. ............................................ 22
  2.8 Coefficient Functions. ........................................ 28
  2.9 Sparse Matrix Storage. ....................................... 33

3 Mesh Generation 37
  3.1 Overview. ..................................................... 37
  3.2 Creating a Triangulation from a Skeleton. .................... 38
  3.3 A Posteriori Error Estimates. ................................ 41
  3.4 Adaptive Mesh Refinement and Unrefinement. .................. 43
    3.4.1 Procedure Refine ......................................... 44
    3.4.2 Procedure Unrefine ....................................... 45
    3.4.3 h Refinement ............................................ 46
3.4.4  \( h \) Unrefinement  
3.4.5  \( p \) Refinement  
3.4.6  \( p \) Unrefinement  
3.5  Adaptive Mesh Smoothing.  
3.6  Uniform Refinement.  
3.6.1  \( h \) Uniform Refinement  
3.6.2  \( p \) Uniform Refinement  
3.7  An Example  
3.8  Parallel Adaptive Methods.  
3.8.1  Mesh Partitioning.  
3.8.2  Reconciling the Mesh.  

4  Equation Solution  
4.1  Overview.  
4.2  Elliptic Boundary Value Problems.  
4.3  Linear Solvers.  
4.4  Domain Decomposition Solver  
4.5  Obstacle Problems.  
4.6  Continuation Problems.  
4.7  Parameter Identification Problems.  
4.8  Optimal Control Problems.  

5  Graphics  
5.1  Overview.  
5.2  Subroutine \texttt{TRIPLT}.  
5.2.1  Surface Plots.  
5.2.2  Vector Plots.  
5.2.3  Parameters \texttt{RMAG}, \texttt{CENX}, and \texttt{CENY}.  
5.2.4  Parameters \texttt{ISCALE}, \texttt{LINES}, \texttt{NUMBRS}, and \texttt{MPIRGN}.  
5.2.5  Parameters \texttt{ICRSN} and \texttt{ITRGT}.  
5.2.6  Some Algorithmic Details.  
5.3  Subroutine \texttt{INPLT}.  
5.3.1  Triangle Plots.  
5.3.2  Skeleton Plots.  
5.4  Subroutine \texttt{GPHPLT}.  
5.4.1  Iteration Information.  
5.4.2  Timing Statistics.  
5.4.3  Continuation Path.  
5.4.4  Parallel Statistics  
5.4.5  Error Estimates.  
5.4.6  Displaying Data Arrays.  

6  Test Driver  
6.1  Overview.  
6.2  Terminal Mode.  
6.3  Web Browser Mode.  

## Contents

6.4 Batch Mode ............................................. 109  
6.5 Parallel Processing ................................. 109  
6.6 Command Line Parameters ....................... 110  
6.7 Array Dimensions and Initialization .......... 111  
6.8 Reading and Writing Files ....................... 112  
6.9 Journal Files ....................................... 112  
6.10 Subroutine USRCMD ............................... 112  
6.11 Subroutine GDATA ............................... 114  
6.12 Machine Dependent Routines ................... 115  
  6.12.1 Arithmetic Specification ................... 115  
  6.12.2 Timing Routine .............................. 116  
  6.12.3 Command Line Interface ................. 117  
  6.12.4 Graphics Interface ......................... 117  
  6.12.5 MPI Interface ............................. 120  

7 Test Problems ........................................ 121  
7.1 Overview ........................................... 121  
7.2 Test Problem CIRCLE ......................... 121  
7.3 Test Problem SQUARE ......................... 122  
7.4 Test Problem DOMAINS ......................... 124  
7.5 Test Problem NACA ............................... 124  
7.6 Test Problem JCN .............................. 126  
7.7 Test Problem OB .................................. 127  
7.8 Test Problem MNSURF ......................... 128  
7.9 Test Problem BURGER ......................... 128  
7.10 Test Problem BATTERY ....................... 129  
7.11 Test Problem CONTROL ....................... 129  
7.12 Test Problem IDENT ......................... 130  
7.13 Test Problem BOX ............................... 131  
7.14 Test Problem MESSAGE ....................... 131  
7.15 Test Problem USMAP ......................... 132  

Bibliography ............................................ 135  

Index .................................................. 141
Preface

Many people have made contributions to the development of this version of PLTMG; I am indebted to them all for their help. The original grid refinement algorithms used in PLTMG were derived in 1976 as joint work with Todd Dupont of the University of Chicago. The approximate Newton strategies incorporated in the present version of PLTMG represent joint work with Donald J. Rose. The gradient recovery and a posteriori error estimation procedures are joint work with Jinchao Xu of Pennsylvania State University and Bin Zheng of Pacific Northwest National Laboratory. The algorithms used in the pseudo-arclength continuation procedures are joint work with Tony Chan of the Hong Kong University of Science and Technology and Hans Mittelmann of Arizona State University. The interior point algorithms used in the optimization problems treated in this version are joint work with Philip Gill of University of California at San Diego. The adaptive mesh smoothing algorithms are joint work with R. Kent Smith. The $hp$ refinement algorithms and associated data structures are joint work with Hieu Nguyen of the Universitat Politècnica De Catalunya and Chris Deotte of the University of California at San Diego. The load balance algorithms for parallel computations are also joint work with Chris Deotte. The web browser interface was developed and implemented by Chris Deotte. The parallel adaptive paradigm is joint work with Michael Holst of the University of California at San Diego. The parallel domain decomposition solver is joint work with Shaoying Lu of the University of California at San Diego and Panayot Vassilevski of Lawrence Livermore National Laboratory. The dual function used for parallel adaptive meshing is joint work with Jeffrey Ovall of Portland State University. Many people made contributions to the test problems, reported bugs and suggested improvements that have been incorporated in the current version.

This version of PLTMG was supported by the National Science Foundation through grants DMS-1318480 and DMS-1345013 (University of California at San Diego). The UCSD Scicomp Beowulf cluster was built using funds provided by the National Science Foundation through MRI-0821816.

University of California at San Diego
September, 2018

Randolph E. Bank
Chapter 1

Introduction

1.1 Problem Specification.

Consider the elliptic boundary value problem

\[-\nabla \cdot a(x, y, u, \nabla u, \lambda) + f(x, y, u, \nabla u, \lambda) = 0 \quad \text{in } \Omega, \quad (1.1)\]

with boundary conditions

\[u = g_2(x, y, \lambda) \quad \text{on } \partial \Omega_2,\]
\[a \cdot n = g_1(x, y, u, \lambda) \quad \text{on } \partial \Omega_1, \quad (1.2)\]
\[u, a \cdot n \quad \text{continuous} \quad \text{on } \partial \Omega_0.\]

Here \(\Omega\) is a bounded region in \(\mathbb{R}^2\), \(n\) is the unit normal, \(a\) is the vector \((a_1, a_2)^t\), \(a_1, a_2, f, g_1,\) and \(g_2\) are scalar functions. \(\partial \Omega_0\) is a portion of \(\partial \Omega\) where periodic boundary conditions are applied. In some problems solved by PLTMG, the parameter \(\lambda\) is not used, while in others \(\lambda \in \mathbb{R}^k, k \geq 1,\) is a vector of scalar parameters or \(\lambda \in H^1(\Omega),\) where \(H^1(\Omega)\) denotes the usual Sobolev space. Let

\[H^1_p = \{ \phi \in H^1(\Omega) \mid \phi \text{ is continuous on } \partial \Omega_0 \},\]
\[H^1_g = \{ \phi \in H^1_p \mid \phi = g_2 \text{ on } \partial \Omega_2 \},\]
\[H^1_e = \{ \phi \in H^1_p \mid \phi = 0 \text{ on } \partial \Omega_2 \}.\]

Then the weak form of (1.1)-(1.2) is: find \(u \in H^1_g\) such that

\[a(u, v) = 0 \quad \text{for all } v \in H^1_e, \quad (1.3)\]

where

\[a(u, v) = \int_{\Omega} a(u, \nabla u, \lambda) \cdot \nabla v + f(u, \nabla u, \lambda) v \, dx \, dy - \int_{\partial \Omega_1} g_1(u, \lambda) v \, ds. \quad (1.4)\]
In some problems solved by PLTMG, a functional $\rho(u, \lambda)$ plays an important role. Functionals we consider are of the form

$$\rho(u, \lambda) = \int_{\Omega} p_1(x, y, u, \nabla u, \lambda) \, dx \, dy + \int_{\Gamma} p_2(x, y, u, \nabla u, \lambda) \, ds,$$

(1.5)

where $p_1$ and $p_2$ are scalar functions. Here $\Gamma = \partial \Omega \cup \Gamma_0$, where $\Gamma_0$ consists of certain internal curves specified by the user.

This version of the PLTMG package addresses five major problem classes. These are briefly described below.

### 1.1.1 Approximation Spaces.

PLTMG is based on a family of conforming $C^0$ finite element spaces. Let $\mathcal{T}$ denote a triangulation of $\Omega$ and let $\mathcal{M}$ be the space of $C^0$ piecewise polynomials associated with $\mathcal{T}$. In this version of PLTMG, the degree of the polynomial can vary element by element. The maximum degree allowed at present is $p = 9$, a condition imposed by the availability of suitable quadrature formulas. PLTMG represents such a piecewise polynomial using the standard Lagrange nodal basis; a function can then be specified by giving its values at the principal lattice points of the element, as illustrated in Figure 1.1 for the cases $1 \leq p \leq 3$.

![Figure 1.1. Nodal degrees of freedom for the continuous piecewise linear element, $p = 1$ (left), the continuous piecewise quadratic element, $p = 2$ (middle), and the continuous piecewise cubic element, $p = 3$ (right).](image)

When two elements of different degrees share a common edge, the element of lower degree becomes a transition element. If such a element is of degree $p$, sharing an edge with an element of degree $q > p$, the element contains all polynomials of degree $p$ plus some additional polynomials of degree $q$, which allow the overall finite element space to remain conforming. In particular, along the shared edge, the degrees of freedom correspond to those of the higher degree element. Some examples are given in Figure 1.2. Finally, PLTMG allows the use of isoparametric versions.

---

1PLTMG uses quadrature formulas given in Zhang, Cui, and Liu [67].
Figure 1.2. Nodal degrees of freedom for the a quadratic transition element with a cubic edge (left), and a cubic transition element with one edge of degree four and one edge of degree five (right).

of this family of Lagrange elements to address problems with curved boundaries or interfaces.

1.1.2 Elliptic Boundary Value Problem.

For this problem, PLTMG solves a discrete analog of (1.3). The parameter $\lambda$ does not play a role in this problem. Let $I : H^1(\Omega) \rightarrow M$ denote continuous piecewise polynomial interpolation operator that interpolates at the degrees of freedom of $T$. Then

$$M_p = \{ \phi \in M \mid \phi \text{ is continuous on } \partial \Omega_0 \},$$
$$M_g = \{ \phi \in M_p \mid \phi = I(g_2) \text{ on } \partial \Omega_2 \},$$
$$M_e = \{ \phi \in M_p \mid \phi = 0 \text{ on } \partial \Omega_2 \}.$$

The discrete equations solved by PLTMG are formulated as follows: find $u_h \in M_d$ such that

$$a(u_h, v) = 0 \quad \text{for all } v \in M_e. \quad (1.6)$$

1.1.3 Obstacle Problem.

The second class of problems addressed by PLTMG are the subset of variational inequalities known as obstacle problems. Let

$$K = \{ \phi \in H^1_g \mid u \leq \phi \leq \pi \}.$$

The obstacle problem is formulated as

$$\min_{u \in K} \rho(u) \quad (1.7)$$

where $\rho$ is a functional of the form (1.5). The parameter $\lambda$ is not used in this problem. Implicit in our formulation of this problem is an assumption that the
Frechet derivative of \( \rho \) corresponds to an elliptic boundary problem of the form (1.3). We also assume that the bound constraints are consistent with the boundary conditions.

The discrete form of this problem is as follows. Let

\[
K_h = \{ \phi \in \mathcal{M}_g \mid I(u) \leq \phi \leq I(\bar{u}) \}.
\]

We then seek \( u_h \in K_h \) that satisfies

\[
\min_{u_h \in K_h} \rho(u_h)
\]  

(1.8)

1.1.4 Continuation Problem.

Continuation problems addressed by PLTMG are all of the form (1.3), where the parameter \( \lambda \in \mathbb{R} \). Continuation problems also require a functional \( \rho \) as in (1.5). Solutions of (1.3)–(1.5) in general define a family of curves on the \((\lambda, \rho)\) plane. Typical curves are shown in Figure 1.3.

![Figure 1.3. Continuation curves \( \rho = \rho(\lambda) \).](image)

The singular point labeled “A” in the figure on the left is a limit (turning) point, and those labeled “B” in the figure on the right are bifurcation points (this figure corresponds to the special case of a linear eigenvalue problem). The purpose of the continuation process is to compute solutions \((u, \lambda)\) corresponding to points on these curves.

PLTMG provides a suite of options for solving continuation problems. Among them are options for following a solution curve to a target value in \( \lambda \) or \( \rho \), locating limit and bifurcation points, and switching branches at bifurcation points. Because some problems might have more than one parameter of interest, PLTMG also has options for switching parameters and functionals (changing the definitions of \( \lambda \) and \( \rho \)) during the calculation, as a means of exploring higher dimensional spaces.

1.1.5 Parameter Identification Problem.

In this problem, a partial differential equation of the form (1.3) appears as a constraint in an optimization problem. Here we seek \( \lambda \in \mathbb{R}^k, 1 \leq k \leq 10, \) and \( u \in \mathcal{H}_g \).
1.1. Problem Specification.

that satisfy

\[ \min \rho(u, \lambda) \]  \tag{1.9} 

subject to the constraint (1.3) and the simple bounds

\[ \lambda_j \leq \lambda_j \leq \bar{\lambda}_j, \]  \tag{1.10} 

for \( 1 \leq j \leq k \). In addition to appearing within the coefficients of the partial differential equation and the boundary conditions, parameters \( \lambda_j \) can be used to describe the shape of the boundary \( \partial \Omega \) or some internal interface. This allows the solution of problems where certain geometric properties of \( \Omega \) are to be optimized.

We define the Lagrangian

\[ L(u, v, \lambda) = \rho(u, \lambda) + a(u, v), \]  \tag{1.11} 

where \( v \in H_e \) is a Lagrange multiplier. We can solve the optimization problem by seeking stationary points of \( L(u, v, \lambda) \) constrained by the simple bounds (1.10).

In the discretized problem, we seek \( u_h \in \mathcal{M}_g \), a discrete Lagrange multiplier \( v_h \in \mathcal{M}_e \), and \( \lambda_h \in \mathcal{R}^k \) that correspond to a stationary point of \( L(u_h, v_h, \lambda_h) \), constrained by the simple bounds

\[ \lambda_j \leq \lambda_{h,j} \leq \bar{\lambda}_j, \]  \tag{1.12} 

for \( 1 \leq j \leq k \).

1.1.6 Optimal Control Problem.

This problem is very similar to the parameter identification problem, except now \( \lambda \in H^1(\Omega) \) (or perhaps some weaker space where pointwise values of (1.14) below are defined). Thus we seek \( u \in H_g \) and \( \lambda \in H^1(\Omega) \) that satisfy

\[ \min \rho(u, \lambda) \]  \tag{1.13} 

subject to the constraint (1.3) and the simple bounds

\[ \Lambda(x, y) \leq \lambda \leq \overline{\Lambda}(x, y) \]  \tag{1.14} 

for \( (x, y) \in \Omega \). As before, we define the Lagrangian

\[ L(u, v, \lambda) = \rho(u, \lambda) + a(u, v), \]  \tag{1.15} 

where \( v \in H_e \) is a Lagrange multiplier. We seek stationary points of \( L(u, v, \lambda) \) constrained by the simple bounds (1.14).

In the discretized problem, we seek \( u_h \in \mathcal{M}_g \), a discrete Lagrange multiplier \( v_h \in \mathcal{M}_e \), and \( \lambda_h \in \mathcal{M} \) that correspond to a stationary point of \( L(u_h, v_h, \lambda_h) \), constrained by the simple bounds

\[ I(\Lambda) \leq \lambda_h \leq I(\overline{\Lambda}). \]  \tag{1.16} 

Inequalities (1.16) are imposed only at the nodes of each element in the mesh.
1.2 Main Subroutines

The software package consists of five primary subroutines. These main routines and their functions are summarized in Table 1.2. The package uses two basic data structures to specify the domain Ω: the triangulation and the skeleton. Loosely speaking, a triangulation specifies the domain Ω as the union of triangles. A skeleton specifies the domain as the union of one or more subdomains and requires only a description of the boundary of each subdomain. The user can specify the domain as either a triangulation or a skeleton. Specifying a triangulation generally requires less data only for simple domains that can be triangulated with very few triangles. If the domain has a complicated geometry or has internal interfaces that the user would like the triangulation to respect, then it is usually easier to specify the domain as a skeleton. Both data structures are documented in Chapter 2.

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Main Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRIGEN</td>
<td>Mesh generation and modification</td>
</tr>
<tr>
<td>PLTMG</td>
<td>Solve partial differential equation</td>
</tr>
<tr>
<td>TRIPLT</td>
<td>Display solution or related function</td>
</tr>
<tr>
<td>INPLT</td>
<td>Display input data</td>
</tr>
<tr>
<td>GPHPLT</td>
<td>Display performance statistics</td>
</tr>
</tbody>
</table>

Table 1.1. The main subroutines in the package.

Subroutine TRIGEN is mainly concerned with transforming the data structures defining the domain. TRIGEN also provides a posteriori error estimates for the solution in the $H^1(\Omega)$ and $L_2(\Omega)$ norms. TRIGEN provides options for creating a triangulation from a skeleton, and adaptively modifying the triangulation data structure. Options for $h$, $p$ and $hp$ adaptive refinement and coarsening, as well as mesh moving ($r$ adaptivity) are provided. TRIGEN also provides options for various tasks related to parallel processing, namely partitioning the mesh, broadcasting a given mesh to all processors, and reconciling a fine mesh distributed among several processors. TRIGEN is documented in Chapter 3.

Subroutine PLTMG uses finite element discretizations based on family of nodal $C^0$ piecewise polynomial spaces described above, and includes algorithms to address each of the five problem classes. In the case of parallel processing, PLTMG includes a domain decomposition solver for each problem class. PLTMG is described in detail in Chapter 4.

Subroutine TRIPLT provides graphical displays of the solution and other grid functions. Three-dimensional color surface/contour plots with shading and an arbitrary viewing perspective are available. Subroutine INPLT provides a graphical display of the mesh data (triangulation or skeleton) defining Ω. Subroutine GPHPLT provides a variety of graphical displays of convergence histories, statistical data, and other interesting output from PLTMG. These routines are described in detail in Chapter 5.

An elementary interactive test driver, ATEST, is described in Chapter 6. ATE-
EST provides options for calling each of the main routines, as well as other useful functions such as writing and reading data files, resetting parameters, and executing problem specific subroutines provided by the user. Several short machine dependent routines are required for timing, graphics, and specifying the precision of the floating point number system. These are also described in Chapter 6. In Chapter 7, the example problem data sets included with the source code are briefly described.

PLTMG was originally conceived as a prototype program to study the theoretical and practical aspects of the multigrid iterative method, adaptive grid refinement and error estimation procedures, and their interaction. As such, PLTMG was designed to (formally) handle a wide class of elliptic operators and reasonably general domains. The boundary of the problem class has expanded as problems were encountered that required its enlargement to be solved. The problem class addressed by this version of PLTMG should not be interpreted as the limit of the class of problems that could be successfully solved by the techniques embodied by this package. Conversely, one should not assume that every problem (formally) within this class can be solved using the existing code.

As with other versions of the package, time efficiency is a secondary consideration to robustness, versatility, and ease of maintenance. While PLTMG is probably not the fastest code that could be used for any particular problem, we believe that it will deliver reasonable execution times in most environments.

1.3 Installation.

This version of PLTMG can be compiled in single, double, or quadruple precision, depending on settings in the machine dependent module MTHDEF. MTHDEF and other machine dependent routines are documented in detail in Section 6.12. The majority of the code is machine independent and written to the specifications of Fortran 90. The web browser interface of Chris Deotte [47] is written in ANSI C. The source code is contained in several files as indicated in Table 1.3.

<table>
<thead>
<tr>
<th>File</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>mg0.f</td>
<td>sets arithmetic precision</td>
</tr>
<tr>
<td>pltmg.f</td>
<td>most source code</td>
</tr>
<tr>
<td>mgmpi.f (mgmpi_stubs.f)</td>
<td>MPI interface</td>
</tr>
<tr>
<td>webgui.c</td>
<td>web browser interface</td>
</tr>
<tr>
<td>atest.f</td>
<td>main driver program</td>
</tr>
<tr>
<td>battery.f, box.f, burger.f, circle.f, control.f, domains.f, ident.f, jcn.f, message.f, mnsurf.f, naca.f, ob.f, square.f, usmap.f</td>
<td>test problem data sets</td>
</tr>
</tbody>
</table>

Table 1.2. Files in the basic distribution.

The parallel processing options in PLTMG are based on MPI, and the MPI library must also be loaded in order to resolve all external names. If MPI is not
available or not desired, one can substitute the supplied *stub* interface routines. The stub routines are a set of MPI interface routines with all calls to MPI library functions and subroutines deleted. By using the stub routines in place of the regular interface, one can create an executable with no unresolved external references without loading the MPI library. In this case, however, all the parallel options of *PLTMG* are disabled.

To create an executable, one needs Fortran 90 and C compilers and an MPI installation. One should create a test problem data file (myfile.f in the following) or use one of the supplied example test problem data sets. One should compile mg0.f, atest.f, pltmg.f mgmpi.f, myfile.f and webgui.c, and link the resulting object files with the MPI library. One can set most of the usual compiler options as desired. However, some compilers may require the -pthread option to successfully compile and link webgui.c. If MPI is not available or not needed, one may substitute mgmpi_stubs.f for mgmpi.f as described above.
Chapter 2

Data Structures

2.1 Overview.

In this chapter, we define the data structures used in the PLTMG package. There are two basic data structures that define the domain $\Omega$: the skeleton and the triangulation. Basic to both data structures is the concept of an edge. The various subregions that define a skeleton are described by a sequence of edges that traverse its boundary in a counter clockwise fashion. In the case of a triangulation, edges on the boundary $\partial \Omega$ need to be explicitly defined in order to assign boundary conditions. Additional internal edges can be defined if they have some attribute of interest; e.g., they are curved. Other internal edges are defined implicitly by the definitions of the triangles that comprise the triangulation. In the case of parallel processing, PLTMG explicitly defines edge data structures for all edges lying on the internal interface system generated by the partitioning of $\Omega$ among the processors. The edge related data structures are defined in Section 2.2. The triangulation and skeleton are defined in Sections 2.3 and 2.4, respectively.

The next few sections define several internal data structures used by PLTMG. The user is never asked to provide data for these structures; they are all computed internally by various routines in the package. However, their contents may still be of interest to the user. Data structures that track degrees of freedom associated with individual elements, as well as the solution and other finite element functions, are described in Section 2.5. The IPATH data structure describes relationships between the subdomains associated with different processors in a parallel adaptive calculation. It is described in Section 2.6.

The arrays IP, RP, and SP contain many scalar parameters, switches, control variables, flags, and pointers, some that must be specified by the user and others that are internally computed but may be of interest to the user. These are described in Section 2.7. Finally, the coefficient functions defining the differential operator and functional $\rho$ in (1.1)–(1.3), and the optional function QXY used by TRIGEN and TRIPLT, are described in Section 2.8.
2.2 Edge Definitions

In this section, we define geometry data structures common to both the triangulation and the skeleton. In both cases, the domain is described by a list of vertices \( v_i, 1 \leq i \leq NVF \), and edges \( b_i, 1 \leq i \leq NBF \). In the case of a triangulation, the \( v_i \) enumerate all vertices of all triangles that comprise the triangulation. In the case of a skeleton, the \( v_i \) enumerate the vertices of all regions that comprise the skeleton. In both cases, the \((x,y)\) coordinates of the vertices are given in the arrays \( VX \) and \( VY \). In particular,

\[
v_I = (x_I, y_I) = (VX(I), VY(I)), \quad 1 \leq I \leq NVF.
\]

Edges are defined in terms of the integer array \( IBNDRY \) of size \( 7 \times NBF \) and the real array \( SF \) of size \( 2 \times NBF \). The latter is used only for curved edges. Curved edges can be most easily be defined by circular arcs (as in early versions of PLTMG) or parametrically through the function \( SXY \) provided by the user. The definitions of \( IBNDRY \) is given in Table 2.1.

Column 1 of the \( IBNDRY \) array contains information about edge \( b_I \). The first two entries are pointers to the \( VX \) and \( VY \) arrays and denote the two vertices that form the endpoints of the edge. The third entry is used to indicate if the edge is curved, and is described more fully below.

Entry \( IBNDRY(4,I) \) describes the type of boundary conditions to be applied, or if the edge is internal to \( \Omega \). A fourth type of edge is a linked edge. Linked edges occur only in pairs. If \( b_I \) and \( b_J \) are a pair of linked edges, then \( IBNDRY(4,I) = -J \) and \( IBNDRY(4,J) = -I \). Linked edges \( b_I \) and \( b_J \) must be geometrically congruent. That is, \( b_I \) must be mapped to \( b_J \) using a translation and orthogonal rotation. Continuity of the solution \( u_h \) and weak continuity of \( a \cdot n \) is imposed on linked edge pairs. Thus if \( b_I \) and \( b_J \) are boundary edges, this is equivalent to imposing periodic boundary conditions. In the course of parallel processing, PLTMG creates edges of types 3–5. Entries \( IBNDRY(5,I) \) and \( IBNDRY(6,I) \) are used internally by PLTMG for parallel processing.

Entry \( IBNDRY(7,I) \) contains an integer label for the edge; this user defined label can be used to uniquely identify a particular edge, or to associate some property with the edge.

2.2.1 Curved Edges – Circular Arcs

If a triangle has a curved edge, it can be specified as a circular arc or given a parametric definition. In the case of a circular arc, one should set \( IBNDRY(3,I) = 1 \). The arc passes through the edge endpoints specified in \( IBNDRY(1,I) \) and \( IBNDRY(2,I) \) and its center \((x_c, y_c)\) is specified in the array \( SF \) as

\[
(x_c, y_c) = (SF(1,I), SF(2,I)).
\]

Because there are generally two such arcs for every pair of endpoints, the shorter arc is taken to be the correct edge; therefore, one must specify arcs that subtend (strictly) less than \( \pi \) of arc; \( \pi/4 \) is a reasonable upper bound.
### 2.2. Edge Definitions

<table>
<thead>
<tr>
<th>array entry</th>
<th>definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBNDRY(1,I)</td>
<td>first endpoint number</td>
</tr>
<tr>
<td>IBNDRY(2,I)</td>
<td>second endpoint number</td>
</tr>
<tr>
<td>IBNDRY(3,I)</td>
<td>curved edge switch</td>
</tr>
<tr>
<td>IBNDRY(4,I)</td>
<td>edge type</td>
</tr>
<tr>
<td>IBNDRY(5,I)</td>
<td>reserved for parallel processing</td>
</tr>
<tr>
<td>IBNDRY(6,I)</td>
<td>reserved for parallel processing</td>
</tr>
<tr>
<td>IBNDRY(7,I)</td>
<td>edge label</td>
</tr>
</tbody>
</table>

**IBNDRY definition.**

<table>
<thead>
<tr>
<th>IBNDRY(3,I)</th>
<th>curved edge type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Straight edge</td>
</tr>
<tr>
<td>1</td>
<td>Curved edge – circular arc</td>
</tr>
<tr>
<td>−K</td>
<td>Curved edge – parametric</td>
</tr>
</tbody>
</table>

**Curved edge types.**

<table>
<thead>
<tr>
<th>IBNDRY(4,I)</th>
<th>edge type</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Dirichlet boundary</td>
</tr>
<tr>
<td>1</td>
<td>natural boundary</td>
</tr>
<tr>
<td>0</td>
<td>internal</td>
</tr>
<tr>
<td>−K</td>
<td>linked with edge K</td>
</tr>
<tr>
<td>3, 4, 5</td>
<td>reserved for parallel processing</td>
</tr>
</tbody>
</table>

**Edge type definitions.**

|                      | 2.2.2 Curved Edges – Parametric                  |

To simplify data entry, we provide the routine `CENTRE` for computing the center of a circle given three points on its boundary. `CENTRE` is called using the statement

\[
\text{Call CENTRE}( X1, Y1, X2, Y2, X3, Y3, XC, YC )
\]

Here \((X1,Y1)\) and \((X2,Y2)\) are the endpoints of an arc of the circle, and \((X3,Y3)\) is a third point on the arc (e.g., the midpoint). `CENTRE` returns the center of the circle in \((XC,YC)\).

#### 2.2.2 Curved Edges – Parametric

A second way to specify a curved edge is through a parametric representation. Since there may be several parametric curves, they are indexed by the user. In particular,
if \( \text{IBNDRY}(3,I) = -K \), then the parametric function \((q_K, r_K)\) is used to define the edge, where

\[
\begin{pmatrix}
x(s) \\
y(s)
\end{pmatrix} = \begin{pmatrix}
q_K(s) \\
r_K(s)
\end{pmatrix}, \quad s_1 \leq s \leq s_2.
\]

The point \( s = s_1 \) corresponds to the first endpoint and \( s = s_2 \) corresponds to the second. In this case, the values in column \( I \) of the array \( SF \) are given by

\[
(s_1, s_2) = (SF(1,I), SF(2,I)).
\]

The parameterization itself is defined by the user in routine \( \text{SXY} \). Subroutines \( \text{SXY} \), has calling sequence

\[
\text{Call} \ SXY( \text{RL}, S, \text{ITAG}, \text{VALUES} )
\]

Here \( RL = \lambda \) is an input array of size \( \text{NRL} \) giving the current value of the parameters \( \lambda \). \( 1 \leq \text{NRL} \leq 10 \) for parameter identification problems, while \( \text{NRL} = 1 \) for the other classes of problems. The parameter \( s_1 \leq S \leq s_2 \) is input specifying the point where \((q_K(S), r_K(S))\) is required. \( \text{ITAG} = K \), where \( K \) is the input index of the functional, originally provided by the user as \( \text{BNDRY}(3,I) = -K \). Also required are the derivative values \((\partial q_K/\partial s, \partial r_K/\partial s)\) and \((\partial q_K/\partial \lambda_J, \partial r_K/\partial \lambda_J)\) for \( 1 \leq J \leq \text{NRL} \).

The output is provided in the array \( \text{VALUES} \), a two dimensional array with 2 rows and \( \text{NRL} + 2 \) columns. To simplify this process, \( \text{PLTMG} \) supplies a labeled common block

\[
\text{common} /\text{VAL4}/ J0, JS, JL
\]

containing a predefined list of integer pointers mapping function and derivative values to particular entries in the \( \text{VALUES} \) array. The details of this mapping are given in Table 2.2.

<table>
<thead>
<tr>
<th>pointer</th>
<th>index</th>
<th>\text{VALUES}(1,\cdot)</th>
<th>\text{VALUES}(2,\cdot)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( J0 = 1 )</td>
<td>( J0 )</td>
<td>( q_K )</td>
<td>( r_K )</td>
</tr>
<tr>
<td>( JS = 2 )</td>
<td>( JS )</td>
<td>( q_{K,s} )</td>
<td>( r_{K,s} )</td>
</tr>
<tr>
<td>( JL = 3 )</td>
<td>( JL + J - 1 )</td>
<td>( q_{K,\lambda_J} )</td>
<td>( r_{K,\lambda_J} )</td>
</tr>
<tr>
<td></td>
<td>( 1 \leq J \leq \text{NRL} )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2. VALUES array for subroutine \( \text{SXY} \).

It is important to emphasize that the parameterization is assumed to roughly correspond to arc length along the curved edge. For example, when the edge is bisected, the “midpoint” \((x_m, y_m)\) is computed from

\[
\begin{pmatrix}
x_m \\
y_m
\end{pmatrix} = \begin{pmatrix}
q_K((s_1 + s_2)/2) \\
r_K((s_1 + s_2)/2)
\end{pmatrix}.
\]
Nodes for isoparametric basis functions are computed using a similar formula. The quality of such calculations is thus dependent on these user defined parameterizations.

### 2.3 The Triangulation.

In this section, we define the triangulation data structure. Let $\mathcal{T}$ denote the triangulation consisting of triangles $t_i$, $1 \leq i \leq NTF$, vertices $v_i$, $1 \leq i \leq NVF$, and edges $b_i$, $1 \leq i \leq NBF$. Triangles may have curved edges, as described in Section 2.2. Curved edges may be on the boundary or in the interior of the region $\Omega$. The $ITNODE$ data structure is a $5 \times NTF$ integer array that defines triangles that comprise $\mathcal{T}$. The details of this data structure are given in Table 2.3.

<table>
<thead>
<tr>
<th>array entry</th>
<th>definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ITNODE(1,I)$</td>
<td>first vertex number</td>
</tr>
<tr>
<td>$ITNODE(2,I)$</td>
<td>second vertex number</td>
</tr>
<tr>
<td>$ITNODE(3,I)$</td>
<td>third vertex number</td>
</tr>
<tr>
<td>$ITNODE(4,I)$</td>
<td>reserved for parallel processing</td>
</tr>
<tr>
<td>$ITNODE(5,I)$</td>
<td>element label</td>
</tr>
</tbody>
</table>

Table 2.3. $ITNODE$ definition for a triangulation.

A given triangle $t_I \in \mathcal{T}$ is specified by giving an accounting of its three vertices and by specifying an integer label or tag. The $J$th column of the $ITNODE$ array contains information about $t_I$. The first three entries of $ITNODE$ contain the three vertex numbers of triangle $t_I$. $ITNODE(J,I) = K$, for $1 \leq J \leq 3$, means $(VX(K), VY(K))$ is the $J$th vertex of $t_I$. The ordering of the vertices of a given triangle is arbitrary and independent of the other triangles.\(^2\) Entry $ITNODE(4,I)$ is used internally by PLTMG in parallel processing, denoting the processor that “owns” $t_I$; one can simply initialize $ITNODE(4,I) = 0$. Entry $ITNODE(5,I)$ contains any user provided label for $t_I$. Such labels are provided strictly for the convenience of the user and can be used to identify differing regions or material properties associated with the element.

For example, consider the circle of radius one with a crack along the positive $x$-axis. This domain can be triangulated using $NTF = 8$ triangles, $NVF = 10$ vertices, and $NBF = 10$ boundary edges, 8 of which are curved, as illustrated in Figure 2.1. Vertices $v_2$ and $v_{10}$ have the same $(x, y)$ coordinates, but $v_2$ is “above” the crack and $v_{10}$ is “below.” Similarly, edge $b_1$ is the top of the crack, while edge $b_{10}$ is the bottom. The ordering of vertices, triangles, and edges is arbitrary. In this example, we will define the curved edges using the parameterization

$$
\begin{pmatrix}
  x \\
  y
\end{pmatrix} =
\begin{pmatrix}
  q_1(s) \\
  r_1(s)
\end{pmatrix} =
\begin{pmatrix}
  \cos(s) \\
  \sin(s)
\end{pmatrix}.
$$

\(^2\) PLTMG reorders vertices as necessary to ensure a counterclockwise orientation for elements.
The data for our example is shown in Table 2.4. In this example, we have chosen to label the triangles in \( ITNODE(5,I) \) by the quadrant in the Euclidean plane in which they lie. In our example, we impose Dirichlet boundary conditions on the outer boundary of the circle, and also along the top of the crack, and Neumann boundary conditions on the bottom of the crack. The outer boundary of the circle is labeled 0, the top of the crack 2, and the bottom of the crack 1.

Several routines in the package check triangulation data structures for common errors in the data. If found, such errors are reported by setting the parameter \( IFLAG \) as described in Table 2.9.

2.4 The Skeleton.

The skeleton data structure is often the easiest data structure for the user to specify by hand, especially if the domain has complicated geometry, symmetry, or internal interfaces. In the skeleton data structure, the domain \( \Omega \) is viewed as the union of \( NTF \) simply connected subregions \( \Omega_i, \ 1 \leq i \leq NTF \). The regions need not be convex, and the case \( NTF = 1 \) is not excluded. A shared boundary between two subregions (an internal interface) will be respected by the triangulation process in \( TRIGEN \); that is, the interface will be represented as one or more triangle edges in the triangulation.

The boundary of each \( \Omega_i \) should be a simple closed curve that does not intersect itself. Thus, for example, if \( \Omega \) has a hole, adding a single cut between the outer boundary and the hole will not be adequate. At least two subregions will be
2.4. The Skeleton.

The VX and VY arrays. NVF = 10.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX(I)</td>
<td>0</td>
<td>1</td>
<td>1/√2</td>
<td>0</td>
<td>-1/√2</td>
<td>-1</td>
<td>-1/√2</td>
<td>0</td>
<td>1/√2</td>
<td>1</td>
</tr>
<tr>
<td>VY(I)</td>
<td>0</td>
<td>0</td>
<td>1/√2</td>
<td>1</td>
<td>1/√2</td>
<td>0</td>
<td>-1/√2</td>
<td>-1</td>
<td>-1/√2</td>
<td>0</td>
</tr>
</tbody>
</table>

The IBNDRY and SF arrays. NBF = 10.

<table>
<thead>
<tr>
<th>I</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBNDRY(1,I)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>IBNDRY(2,I)</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>IBNDRY(3,I)</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>IBNDRY(4,I)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>IBNDRY(5,I)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>IBNDRY(6,I)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>IBNDRY(7,I)</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SF(1,I)</td>
<td>0</td>
<td>π/4</td>
<td>π/2</td>
<td>3π/4</td>
<td>π</td>
<td>5π/4</td>
<td>3π/2</td>
<td>7π/4</td>
<td>2π</td>
<td></td>
</tr>
<tr>
<td>SF(2,I)</td>
<td>0</td>
<td>π/4</td>
<td>π/2</td>
<td>3π/4</td>
<td>π</td>
<td>5π/4</td>
<td>3π/2</td>
<td>7π/4</td>
<td>2π</td>
<td></td>
</tr>
</tbody>
</table>

The ITNODE array. NTF = 8.

<table>
<thead>
<tr>
<th>I</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITNODE(1,I)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ITNODE(2,I)</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>ITNODE(3,I)</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>ITNODE(4,I)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ITNODE(5,I)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 2.4. Data structures for a triangulation.

required in this case.

Having decomposed the domain into NTF subregions, we decompose the boundaries of the subregions into NBF edges b_i, 1 ≤ i ≤ NBF. Each edge has two endpoints ν_j, 1 ≤ j ≤ 2, and if it is a curved edge, it will have a circle center or some parameterization. Globally, the vertices are labeled v_k, 1 ≤ k ≤ NVF, and curved edges are indicated as described in Section 2.2. The intersection of any two edges should be at most one common endpoint.

As an example, we consider the square region with a hole illustrated in Figure 2.2. In this example, we decompose the region into 2 subregions (NTF = 2), using 10 vertices (NVF = 10), and 12 edges (NBF = 12) as shown.

Global numbering of the subregions, edges, and vertices is arbitrary. These arrays for our example domain are shown in Table 2.6. Edges are specified in IBNDRY as in the case of the triangulation. Descendants of Dirichlet, natural, and
array entry | definition
--- | ---
ITNODE(1,I) | first vertex number
ITNODE(2,I) | first edge number
ITNODE(3,I) | congruent region number
ITNODE(4,I) | reserved for parallel processing
ITNODE(5,I) | region label

Table 2.5. ITNODE definition for a skeleton.

Figure 2.2. Example domain decomposed into two subregions (left); vertex numbers (middle); edge numbers (right).

linked edges are included in the output IBNDRY array when Ω is triangulated using TRIGEN. Descendants of internal edges are retained only if they separate regions with different labels or of the edge is curved. Descendant edges inherit the label of the original edge. In our example, we will assign Dirichlet boundary conditions to the left and right sides and the bottom of the domain, and natural boundary conditions elsewhere. The four curved edges are defined by circular arcs of a circle with its center at the origin. The IBNDRY and SF arrays then have the form given in Table 2.6.

A subregion Ωᵢ, 1 ≤ i ≤ NTF, is defined by an ordered sequence of edges (at least three) that form its boundary. The sequence is ordered such that the boundary of Ωᵢ is traversed in a counterclockwise direction (thus providing notions of “inside” and “outside”). Each edge in the sequence shares exactly one endpoint with the edge that precedes it and the edge that follows it in the sequence; the first and last edges in the sequence also share one endpoint. A particular edge can appear only once in the sequence.

The array ITNODE is used to define the subregions. Column I of ITNODE corresponds to the region Ωᵢ. Entry ITNODE(1,I) is a global vertex number for one of the vertices on the boundary of Ωᵢ. Unless ITNODE(3,I) ≠ 0 (see below) the choice of vertex is arbitrary. The second entry, ITNODE(2,I), is the global edge number of the first edge in a counterclockwise traversal of Ωᵢ, beginning at vertex vₖ, where ITNODE(1,I) = k.
The Skeleton.

Table 2.6. Skeleton data structures.

<table>
<thead>
<tr>
<th>$I$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$VX(I)$</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$VY(I)$</td>
<td>2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

The $VX$ and $VY$ arrays. NVF = 10.

<table>
<thead>
<tr>
<th>$I$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$IBNDRY(1,I)$</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>$IBNDRY(2,I)$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>$IBNDRY(3,I)$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$IBNDRY(4,I)$</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$IBNDRY(5,I)$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$IBNDRY(6,I)$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$IBNDRY(7,I)$</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>$SF(1,I)$</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>$SF(2,I)$</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>–</td>
<td></td>
</tr>
</tbody>
</table>

The $IBNDRY$ and $SF$ arrays. NBF = 12.

<table>
<thead>
<tr>
<th>$I$</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ITNODE(1,I)$</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>$ITNODE(2,I)$</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>$ITNODE(3,I)$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$ITNODE(4,I)$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$ITNODE(5,I)$</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$I$</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ITNODE(1,I)$</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>$ITNODE(2,I)$</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>$ITNODE(3,I)$</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>$ITNODE(4,I)$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$ITNODE(5,I)$</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

The $ITNODE$ array for mapping by rotation (left) and by reflection (right). NTF = 2.

Entry $ITNODE(3,I)$ is used to specify certain symmetries the user may wish to impose on the triangulation. Two subregions are congruent if one can be mapped onto the other using an affine transformation consisting of a translation, an orthogonal rotation, and perhaps a simple reflection. If this mapping also induces one-to-one correspondences between the edges and vertices used to define the regions, then the user can specify that the two regions be triangulated in a similar fashion.

$ITNODE(3,I) = 0$ specifies that $\Omega_I$ can be triangulated independently of other regions. $ITNODE(3,I) = J$, $0 < J < I$, specifies that $\Omega_I$ can be mapped onto $\Omega_J$ using just a translation and rotation. $ITNODE(3,I) = -J$, $0 < J < I$, specifies that $\Omega_I$ can be mapped onto $\Omega_J$ using a translation, rotation, and a reflection. If $ITNODE(3,I) = \pm J$, then $ITNODE(1,I)$ must correspond to the vertex
on $\partial \Omega_I$ which is mapped to the vertex corresponding to $ITNODE(1,J)$ on $\partial \Omega_J$. If $ITNODE(3,I) \neq 0$, TRIGEN will map the triangulation generated for $\Omega_J$ onto $\Omega_I$, ensuring the desired symmetry properties of the overall triangulation. Note that this is not a symmetric relation; $ITNODE(3,I) = J$ does not mean $ITNODE(3,J) = I$. In particular, if $|ITNODE(3,I)| \geq I$, TRIGEN will return in an error condition.

In our example, $\Omega_2$ can be mapped onto $\Omega_1$ by either rotation or reflection. We can ensure the triangulation for $\Omega_2$ will be similar to that for $\Omega_1$, either under rotation or reflection. The resulting triangulations may be different in the two cases. $ITNODE$ arrays for the two situations are illustrated in Table 2.6. Entry $ITNODE(4,I)$ is used by PLTMG in parallel processing. Entry $ITNODE(5,I)$ is a user defined label for the region; all the triangles created in $\Omega_I$ inherit this label.

We provide the utility subroutine SKLUTL to aid in the creation of the skeleton data structures. Subroutine SKLUTL is called using the statement

\begin{verbatim}
Call SKLUTL( ISW, VX, VY, SF, ITNODE, IBNDRY, IP, RP, IFLAG, SXY )
\end{verbatim}

This routine takes as input a skeleton data structure defined VX, VY, SF, IBNDRY, (except when ISW = 0) ITNODE, and the routine SXY, called if curved edges are defined by a parameterization. The integers NTF, NBF, and (except when ISW = 0) NTF should be specified in the IP array, and $\lambda = RL$ should be specified in RP if SXY is to be called. The integer ISW specifies the task, as indicated in Table 2.7.

<table>
<thead>
<tr>
<th>ISW</th>
<th>task</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>create ITNODE array</td>
</tr>
<tr>
<td>1</td>
<td>refine long circular arcs</td>
</tr>
<tr>
<td>2</td>
<td>determine congruent regions</td>
</tr>
</tbody>
</table>

Table 2.7. The values of ISW.

If ISW = 0, SKLUTL computes all entries of the ITNODE array, given the remaining arrays in the skeleton data structure (VX, VY, SF, and IBNDRY), and the parameters NVF, and NBF in the IP array. The value of NTF is returned in the IP array. The regions are labeled with $ITNODE(5,I) = I$ for $1 \leq I \leq NTF$, although these labels can subsequently be reset by the user. Also $ITNODE(3,I) = 0$ for $1 \leq I \leq NTF$. If ISW = 1, SKLUTL accepts as input a complete skeleton description, and divides curved edges defined as circular arcs as necessary to ensure that all such edges subtend less than $\pi/4$ of arc. New edges and vertices are added as necessary, and the relevant skeleton parameters updated. New values of NBF and NVF are returned in the IP array. If ISW = 2, SKLUTL accepts as input a complete skeleton description, and finds congruent regions. The values of $ITNODE(3,I)$ (and

\footnote{We could ensure greater symmetry in the triangulation by decomposing $\Omega$ into 4 or 8 congruent regions instead of 2 and then setting $ITNODE(3,I)$ appropriately.}
possibly \( \text{ITNODE}(1,I) \) and \( \text{ITNODE}(2,I) \) are reset as necessary. If two regions are congruent but the congruence is not unique, as in our example, an arbitrary choice is made from among the possibilities. Errors are returned in the integer \( \text{IFLAG} \) as described in Table 2.9.

Several other routines in the package check skeleton data structures for common errors in the data. If found, such errors are reported by setting the parameter \( \text{IFLAG} \) as described in Table 2.9.

### 2.5 Finite Element Data Structures.

Several data structures in \textit{PLTMG} define and maintain the finite element functions associated with a particular problem. In particular, the \( 8 \times NTF \) integer array \( \text{ITDOF} \) contains information about polynomial spaces on each element, the real array \( \text{GF} \) contains the numerical values of the solution and other finite element functions, and the real array \( E \) contains information about the a posteriori error estimates in each element. These data structures are not defined or initialized by the user, but it may be of interest for a user to understand their contents.

\textit{PLTMG} uses local notation to define certain quantities related to a given element in the mesh. See Figure 2.3. For example, each element locally has vertices labeled \( \nu_k, 1 \leq k \leq 3 \). From this viewpoint, the \( \text{ITNODE} \) array contains a mapping for these locally defined vertices to globally defined vertices, with \( \nu_K \) corresponding to \( \text{ITNODE}(K, \cdot) \) for \( 1 \leq K \leq 3 \). Edges are locally labeled as in Figure 2.3, with edge \( \epsilon_k \) opposite vertex \( \nu_k, 1 \leq k \leq 3 \).

![Figure 2.3](image)

**Figure 2.3.** Local notation for element \( t_I \).

In the \( \text{ITDOF} \) array, column \( I \) contains information related to element \( t_I \). The first three entries give the global indices for the degrees of freedom associated with the three vertices of the the element. If the element has an edge with degree \( p \geq 2 \) there are \( p - 1 \) degrees of freedom (bump functions) associated with that edge. In \textit{PLTMG} these degrees of freedom are given consecutive global indices, that either increase or decrease with a counter clockwise traversal of that edge. Entries 4–6 in column \( I \) provide the starting degree of freedom for each edge, with a sign that indicates whether they increase or decrease. If the element has degree
array entry | definition
---|---
$ITDOF(1,I)$ | degree of freedom for vertex $\nu_1$
$ITDOF(2,I)$ | degree of freedom for vertex $\nu_2$
$ITDOF(3,I)$ | degree of freedom for vertex $\nu_3$
$ITDOF(4,I)$ | $\pm$ first degree of freedom for edge $\epsilon_1$
$ITDOF(5,I)$ | $\pm$ first degree of freedom for edge $\epsilon_2$
$ITDOF(6,I)$ | $\pm$ first degree of freedom for edge $\epsilon_3$
$ITDOF(7,I)$ | first interior degree of freedom for element $t_I$
$ITDOF(8,I)$ | polynomial degrees for element $t_I$

Table 2.8. $ITDOF$ definition.

$p \geq 3$ there are $(p - 1)(p - 2)/2$ degrees of freedom (bubble functions) associated with the element interior. These are also given consecutive global indices. The lowest numbered corresponds to the interior node closest to vertex $\nu_1$; subsequently they are numbered “row-by-row;” within each row, degrees of freedom are numbered “left-to-right” with the lowest number for that row corresponding to the node closest to edge $\epsilon_3$. The global index for the lowest numbered interior degree of freedom is given in $ITDOF(7,I)$.

Entry $ITDOF(8,I)$ contains information about the degree of the polynomials associated with element $t_I$. Let the element itself have degree $p_0$. Each edge $\epsilon_k$ can have a different degree $p_k \geq p_0$, $1 \leq k \leq 3$. In PLTMG, elements and edges can have degree at most nine. These degrees are encoded in $ITDOF(8,I)$ as

$$ITDOF(8,I) = p_0 + 16p_1 + 16^2p_2 + 16^3p_3.$$}

The total number of degrees of freedom associated with the finite element space is $NDF$. Values for finite element functions are stored in the array $GF$, with $\text{MAXD} \geq NDF$ rows and $1 \leq I \leq 13$ columns, depending on the problem, with each column associated with a specific finite element function. The definitions for each problem are given in Table 2.9.

The first column of $GF$ always contains the finite element solution $u$. If the problem is solved via parallel computation, then the last column of $GF$ contains the function $\omega$, which is a locally computed dual function that indicates the influence of the regions outside of the given processor’s assigned domain on that domain. For simple pde equations, the user can also specify a functional computed from the user supplied function $\rho$. For continuation problems, the tangent vector $\dot{u}$, as well as $u_0$ and $\dot{u}_0$ from the previous step, are stored. The vectors $\psi_r$ and $\psi_\ell$ are the right and left singular vectors, respectively, associated with the smallest singular value of the Jacobian (stiffness) matrix; these are used in the determination of limit and bifurcation points, among other things. Parameter identification and optimal control problems require a Lagrange multiplier function $v$. The optimal

---

4This constraint is due to limitations of available quadrature rules, given by Zhang, Cui, and Liu in [67], and not to any intrinsic constraint on the spaces themselves.
control problem also requires the distributed control function $\lambda$, and the parameter identification problem requires NRL vectors $u_{\lambda k}$ for $1 \leq k \leq NRL$.

The array $E$ contains information about a posteriori error estimates. It has $\text{MAXT} \geq \text{NTF}$ rows and two columns. The first column contains the local error indicator $\eta_I$ for element $t_I$, and the second column contains a normalization constant used in the decision process in $hp$ refinement. We note that information in the $E$ array is typically modified by various adaptive algorithms in $\text{TRIGEN}$, and the output from $\text{TRIGEN}$ in the $E$ array should be considered unreliable except in the case when $\text{TRIGEN}$ is called to only compute error estimates.

### 2.6 Parallel Processing Data Structure.

When $\text{PLTMG}$ solves a problem using parallel processing, it partitions the domain $\Omega$ into $\text{NPROC}$ subdomains $\Omega_I$ if $\text{NPROC}$ processors are used. This creates an internal interface system $\Gamma$; $\text{PLTMG}$ creates internal edges as needed such that every edge in the internal interface system is represented in the $\text{IBNDRY}$ array. At the conclusion of the adaptive process, the global conforming finite element space needs to be created, and corresponding edges and degrees of freedom on different processors need to be linked in order to carry out the domain decomposition solve that computes the final finite element solution on the global mesh.

The integer array $\text{IPATH}$ is a data structure jointly computed by all of the processors; each processor provides a block of data within the $\text{IPATH}$ array that describes its part of the global interface system. In particular, each processor begins its adaptive enrichment starting from the same interface system, consisting of so-called root edges. Root edges may be bisected ($h$-refined) or have their degree increased ($p$-refined) in potentially arbitrary combinations. The data in the $\text{IPATH}$ array provides a binary tree for each root edge, as well as pointers that indicate the global edge numbers and degrees of freedom for that edge in that processors own data structures. Data for different types of nodes in the tree is given in Table 2.10.

The $\text{IPATH}$ array has six rows. For all edges in the tree, the first entry
Table 2.10. *IPATH* definition – tree section.

<table>
<thead>
<tr>
<th>array entry</th>
<th>root</th>
<th>root/leaf</th>
<th>internal</th>
<th>leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>IPATH</em>(1,I)</td>
<td>neighbor</td>
<td>neighbor</td>
<td>neighbor</td>
<td>neighbor</td>
</tr>
<tr>
<td><em>IPATH</em>(2,I)</td>
<td>child</td>
<td>-edge</td>
<td>child</td>
<td>-edge</td>
</tr>
<tr>
<td><em>IPATH</em>(3,I)</td>
<td>–</td>
<td>vertex 1</td>
<td>–</td>
<td>vertex 1</td>
</tr>
<tr>
<td><em>IPATH</em>(4,I)</td>
<td>–</td>
<td>vertex 2</td>
<td>–</td>
<td>vertex 2</td>
</tr>
<tr>
<td><em>IPATH</em>(5,I)</td>
<td>–</td>
<td>± edge</td>
<td>–</td>
<td>± edge</td>
</tr>
<tr>
<td><em>IPATH</em>(6,I)</td>
<td>–</td>
<td>degree</td>
<td>–</td>
<td>degree</td>
</tr>
</tbody>
</table>

(neighbor) is a pointer to the column in *IPATH* that contains the same edge, but for the neighbor processor; this is the key that identifies the same physical edge on different processors. The second entry identifies the (first) child for non-leaf edges in the tree; these are just pointers to other columns in the *IPATH* array. The two children of a bisected edge appear in consecutive columns in *IPATH*. Leaf entries have a pointer (-edge) that points to the column corresponding to that edge in the given processors *IBNDRY* array; the negative sign is to distinguish it from a child pointer.

In the domain decomposition solve, interface information corresponding to all nodes lying on the global interface system must be exchanged via *MPI*. This is done using a transient interface data structure, with blocks of data provided by each processor. Entries 3–6 in the *IPATH* array for leaf edges provide pointers that indicate the location in the transient data structure for data corresponding to the two endpoints, and if the edge degree \( p \geq 2 \), the edge data. The fifth location is stored as \( ± \)edge, with the sign indicating an increase or decrease in index with a counter clockwise traversal.

The first \( NPROC + 2 \) columns of the *IPATH* data structure contain pointers, one column for each processor, one column for the global mesh and one column for the coarse part of the interface of the given processor. These pointers indicate the blocks of the *IPATH* array and the shared array for the domain decomposition solver that are used by the given processor. Note that after the basic *IPATH* array is computed jointly by all of the processors, each processor appends a tree section for the coarse part of its interface to the end of the *IPATH* array. This information is different on every processor and is used by the domain decomposition solver.

### 2.7 Parameter Arrays.

*IP*, *RP*, and *SP* are integer, real, and *CHARACTER*\(^{*80}\) arrays, respectively, of length 100 containing various user specified parameters, and internally generated parameters, switches, flags, and pointers. A list of the currently used locations, their names, and brief definitions appears in Tables 2.12–2.14. Parameters marked “u” should be supplied by the user.

The parameter *IFIRST* is an initialization switch specifying the degree of the
2.7. Parameter Arrays.

<table>
<thead>
<tr>
<th>$IFIRST$</th>
<th>option</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no initialization</td>
</tr>
<tr>
<td>1</td>
<td>initialize for piecewise linear elements</td>
</tr>
<tr>
<td>2</td>
<td>initialize for piecewise quadratic elements</td>
</tr>
<tr>
<td>3</td>
<td>initialize for piecewise cubic elements</td>
</tr>
<tr>
<td>4</td>
<td>initialize for piecewise quartic elements</td>
</tr>
<tr>
<td>5</td>
<td>initialize for piecewise quintic elements</td>
</tr>
<tr>
<td>6</td>
<td>initialize for piecewise polynomials of degree 6</td>
</tr>
<tr>
<td>7</td>
<td>initialize for piecewise polynomials of degree 7</td>
</tr>
<tr>
<td>8</td>
<td>initialize for piecewise polynomials of degree 8</td>
</tr>
<tr>
<td>9</td>
<td>initialize for piecewise polynomials of degree 9</td>
</tr>
</tbody>
</table>

Table 2.11. The values of $IFIRST$.

finite element space to be used, as indicated in Table 2.11. If $IFIRST = 0$, no initialization takes place. If $IFIRST = p$, $1 \leq p \leq 9$, triangulation data structures are checked, and various arrays are initialized for piecewise polynomial elements of degree $p$. Array entry $IP(25)$ is the error flag $IFLAG$. A summary of the possible values for $IFLAG$ is given in Table 2.15.

<table>
<thead>
<tr>
<th>$I$</th>
<th>$IP(I)$</th>
<th>u definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$NTF$</td>
<td>u number of triangles / regions</td>
</tr>
<tr>
<td>2</td>
<td>$NVF$</td>
<td>u number of vertices</td>
</tr>
<tr>
<td>3</td>
<td>$NBF$</td>
<td>u number of edges</td>
</tr>
<tr>
<td>4</td>
<td>$NDF$</td>
<td>u number of degrees of freedom</td>
</tr>
<tr>
<td>5</td>
<td>$IFIRST$</td>
<td>u initialization switch</td>
</tr>
<tr>
<td>6</td>
<td>$IPROB$</td>
<td>u problem type</td>
</tr>
<tr>
<td>7</td>
<td>$ITASK$</td>
<td>u problem task</td>
</tr>
<tr>
<td>8</td>
<td>$ISPD$</td>
<td>u symmetric / nonsymmetric switch</td>
</tr>
<tr>
<td>9</td>
<td>$METHOD$</td>
<td>u preconditioner options</td>
</tr>
<tr>
<td>10</td>
<td>$MXCG$</td>
<td>u maximum conjugate gradient iterations</td>
</tr>
<tr>
<td>11</td>
<td>$MXNWTT$</td>
<td>u maximum damped Newton iterations</td>
</tr>
<tr>
<td>12</td>
<td>$ISING$</td>
<td>u switch for singular Neumann problem</td>
</tr>
<tr>
<td>13</td>
<td>$NRL$</td>
<td>u number of parameters $\lambda$</td>
</tr>
<tr>
<td>17</td>
<td>$IRTYPE$</td>
<td>u refinement / coarsening options</td>
</tr>
<tr>
<td>18</td>
<td>$MXORD$</td>
<td>u maximum polynomial degree</td>
</tr>
<tr>
<td>19</td>
<td>$IERRSW$</td>
<td>u error recovery switch</td>
</tr>
<tr>
<td>20</td>
<td>$IADAPT$</td>
<td>u mesh generation option switch</td>
</tr>
<tr>
<td>21</td>
<td>$IREFN$</td>
<td>u uniform refinement control</td>
</tr>
<tr>
<td>22</td>
<td>$NDTRGT$</td>
<td>u target value for number of vertices</td>
</tr>
<tr>
<td>23</td>
<td>$NOCHNG$</td>
<td>u $PLTMG$-$TRIGEN$ communication flag</td>
</tr>
</tbody>
</table>

Table 2.12: $IP$ array definitions. (Continued next page.)
<table>
<thead>
<tr>
<th>I</th>
<th>IP(I)</th>
<th>u</th>
<th>definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>MFLAG</td>
<td>u</td>
<td>parallel error flag</td>
</tr>
<tr>
<td>25</td>
<td>IFLAG</td>
<td>u</td>
<td>error flag</td>
</tr>
<tr>
<td>27</td>
<td>NEWNTF</td>
<td></td>
<td>number of elements owned by processor</td>
</tr>
<tr>
<td>28</td>
<td>NEWNVF</td>
<td></td>
<td>number of vertices owned by processor</td>
</tr>
<tr>
<td>29</td>
<td>NEWNBF</td>
<td></td>
<td>number of edges owned by processor</td>
</tr>
<tr>
<td>30</td>
<td>NEWNDF</td>
<td></td>
<td>number of degrees of freedom owned by processor</td>
</tr>
<tr>
<td>31</td>
<td>NVV</td>
<td></td>
<td>number of interface vertices</td>
</tr>
<tr>
<td>32</td>
<td>NBB</td>
<td></td>
<td>number of interface edges</td>
</tr>
<tr>
<td>33</td>
<td>NDD</td>
<td></td>
<td>number of interface degrees of freedom</td>
</tr>
<tr>
<td>34</td>
<td>NVI</td>
<td></td>
<td>number of coarse interface vertices</td>
</tr>
<tr>
<td>35</td>
<td>NBI</td>
<td></td>
<td>number of coarse interface edges</td>
</tr>
<tr>
<td>36</td>
<td>NDI</td>
<td></td>
<td>number of coarse degrees of freedom</td>
</tr>
<tr>
<td>37</td>
<td>NTG</td>
<td></td>
<td>global number of elements</td>
</tr>
<tr>
<td>38</td>
<td>NVG</td>
<td></td>
<td>global number of vertices</td>
</tr>
<tr>
<td>39</td>
<td>NBG</td>
<td></td>
<td>global number of edges</td>
</tr>
<tr>
<td>40</td>
<td>NDG</td>
<td></td>
<td>global number of degrees of freedom</td>
</tr>
<tr>
<td>41</td>
<td>IUSRSW</td>
<td>u</td>
<td>USRCMD switch</td>
</tr>
<tr>
<td>42</td>
<td>MODE</td>
<td>u</td>
<td>ATTEST mode switch</td>
</tr>
<tr>
<td>43</td>
<td>WEBPRT</td>
<td>u</td>
<td>communications socket for WEBGUI</td>
</tr>
<tr>
<td>47</td>
<td>MPRGN</td>
<td>u</td>
<td>region for file read/write and graphics</td>
</tr>
<tr>
<td>48</td>
<td>MPISW</td>
<td>u</td>
<td>MPI switch</td>
</tr>
<tr>
<td>49</td>
<td>NPROC</td>
<td></td>
<td>number of processes</td>
</tr>
<tr>
<td>50</td>
<td>IRGN</td>
<td></td>
<td>individual process number</td>
</tr>
<tr>
<td>51</td>
<td>MXCOLR</td>
<td>u</td>
<td>maximum number of colors</td>
</tr>
<tr>
<td>52</td>
<td>IFUN</td>
<td>u</td>
<td>alternate function switch for TRIPLT</td>
</tr>
<tr>
<td>53</td>
<td>INPLSW</td>
<td>u</td>
<td>alternate graph switch for INPLT</td>
</tr>
<tr>
<td>54</td>
<td>IGRSW</td>
<td>u</td>
<td>alternate graph switch for GPHPLT</td>
</tr>
<tr>
<td>55</td>
<td>NCON</td>
<td></td>
<td>number of contours</td>
</tr>
<tr>
<td>56</td>
<td>ICONT</td>
<td>u</td>
<td>continuity switch</td>
</tr>
<tr>
<td>57</td>
<td>ISCALE</td>
<td>u</td>
<td>scale option switch</td>
</tr>
<tr>
<td>58</td>
<td>LINES</td>
<td>u</td>
<td>line drawing option switch</td>
</tr>
<tr>
<td>59</td>
<td>NUMBRS</td>
<td>u</td>
<td>numbering option switch</td>
</tr>
<tr>
<td>60</td>
<td>NX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>NY</td>
<td>u</td>
<td>(NX,NY,NZ)</td>
</tr>
<tr>
<td>62</td>
<td>NZ</td>
<td>u</td>
<td>is the viewing perspective for TRIPLT</td>
</tr>
<tr>
<td>63</td>
<td>MX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>MY</td>
<td>u</td>
<td>(MX,MY,MZ)</td>
</tr>
<tr>
<td>65</td>
<td>MZ</td>
<td>u</td>
<td>is the viewing perspective for GPHPLT</td>
</tr>
<tr>
<td>66</td>
<td>ICRSN</td>
<td>u</td>
<td>graphics coarsening switch</td>
</tr>
<tr>
<td>67</td>
<td>ITRTGT</td>
<td>u</td>
<td>target size of graphics mesh</td>
</tr>
<tr>
<td>68</td>
<td>IDEVCE</td>
<td>u</td>
<td>graphics device switch</td>
</tr>
</tbody>
</table>

Table 2.12: IP array definitions. (Continued next page.)
2.7. Parameter Arrays.

<table>
<thead>
<tr>
<th>(I)</th>
<th>(IP(I))</th>
<th>u</th>
<th>definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>69</td>
<td>FPANE</td>
<td>u</td>
<td>WEBGUI canvas for (TRIPLT)</td>
</tr>
<tr>
<td>70</td>
<td>GPANE</td>
<td>u</td>
<td>WEBGUI canvas for (GPHPLT)</td>
</tr>
<tr>
<td>71</td>
<td>JPANE</td>
<td>u</td>
<td>WEBGUI canvas for (INPLT)</td>
</tr>
<tr>
<td>73</td>
<td>MXLABL</td>
<td></td>
<td>maximum label for MPI interface edges</td>
</tr>
<tr>
<td>74</td>
<td>NVDD</td>
<td></td>
<td>total number of interface vertices</td>
</tr>
<tr>
<td>75</td>
<td>LIPATH</td>
<td></td>
<td>length of (IPATH) array</td>
</tr>
<tr>
<td>76</td>
<td>NEF</td>
<td></td>
<td>number of error functions</td>
</tr>
<tr>
<td>77</td>
<td>NGF</td>
<td></td>
<td>number of grid functions</td>
</tr>
<tr>
<td>78</td>
<td>NDL</td>
<td></td>
<td>order of error recovery systems</td>
</tr>
<tr>
<td>79</td>
<td>IEVALS</td>
<td></td>
<td>number of function evaluations on last call</td>
</tr>
<tr>
<td>80</td>
<td>ITNUM</td>
<td></td>
<td>number of Newton iterations on last call</td>
</tr>
<tr>
<td>82</td>
<td>MAXPTH</td>
<td>u</td>
<td>number of columns in the array (IPATH)</td>
</tr>
<tr>
<td>83</td>
<td>MAXT</td>
<td>u</td>
<td>number of columns in the array (ITNODE)</td>
</tr>
<tr>
<td>84</td>
<td>MAXV</td>
<td>u</td>
<td>length of the arrays (VX) and (VY)</td>
</tr>
<tr>
<td>85</td>
<td>MAXD</td>
<td>u</td>
<td>length of grid function array (GF)</td>
</tr>
<tr>
<td>86</td>
<td>MAXB</td>
<td>u</td>
<td>number of columns in the array (IBNDRY)</td>
</tr>
<tr>
<td>90</td>
<td>NDF</td>
<td></td>
<td>order of the linear system</td>
</tr>
<tr>
<td>91</td>
<td>NB</td>
<td></td>
<td>number of blocks in the linear system</td>
</tr>
<tr>
<td>92</td>
<td>LENJA</td>
<td></td>
<td>length of (JA) array</td>
</tr>
<tr>
<td>93</td>
<td>LENAD</td>
<td></td>
<td>length of diagonal part (A) array</td>
</tr>
<tr>
<td>94</td>
<td>LENAOD</td>
<td></td>
<td>length of upper / lower triangular (A) array</td>
</tr>
<tr>
<td>95</td>
<td>LENJU</td>
<td></td>
<td>maximum length of (JU) array</td>
</tr>
<tr>
<td>96</td>
<td>LENUOD</td>
<td></td>
<td>maximum length of upper / lower triangular (U) array</td>
</tr>
<tr>
<td>97</td>
<td>LENJU0</td>
<td></td>
<td>length of (JU) array</td>
</tr>
<tr>
<td>98</td>
<td>LENU0</td>
<td></td>
<td>length of (U) array</td>
</tr>
<tr>
<td>99</td>
<td>LENJA0</td>
<td></td>
<td>length of (JA) for HB decomposition</td>
</tr>
<tr>
<td>100</td>
<td>LENJUC</td>
<td></td>
<td>length of (JU) for HB decomposition</td>
</tr>
</tbody>
</table>

Table 2.12: \(IP\) array definitions.

<table>
<thead>
<tr>
<th>(I)</th>
<th>(RP(I))</th>
<th>u</th>
<th>definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RLTRGT</td>
<td>u</td>
<td>target value for (\lambda)</td>
</tr>
<tr>
<td>2</td>
<td>RTRGT</td>
<td>u</td>
<td>target value for (\rho(u, \lambda))</td>
</tr>
<tr>
<td>3</td>
<td>RMTRGT</td>
<td>u</td>
<td>target value for (\mu)</td>
</tr>
<tr>
<td>4</td>
<td>DTOL</td>
<td>u</td>
<td>drop tolerance for incomplete factorization</td>
</tr>
<tr>
<td>6</td>
<td>SMIN</td>
<td>u</td>
<td>lower limit for contour colors</td>
</tr>
<tr>
<td>7</td>
<td>SMAX</td>
<td>u</td>
<td>upper limit for contour colors</td>
</tr>
<tr>
<td>8</td>
<td>RMAG</td>
<td>u</td>
<td>window magnification factor</td>
</tr>
<tr>
<td>9</td>
<td>CENX</td>
<td>u</td>
<td>((CENX,CENY)) are the window center coordinates</td>
</tr>
</tbody>
</table>

Table 2.13: \(RP\) array definitions. (Continued next page.)
<table>
<thead>
<tr>
<th>I</th>
<th>RP(I)</th>
<th>u definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>CENY u</td>
<td>u definition</td>
</tr>
<tr>
<td>12</td>
<td>HMAX u</td>
<td>approximate largest element size</td>
</tr>
<tr>
<td>13</td>
<td>GRADE u</td>
<td>largest growth factor for adjacent elements</td>
</tr>
<tr>
<td>14</td>
<td>HMIN u</td>
<td>approximate smallest edge length</td>
</tr>
<tr>
<td>16</td>
<td>XMIN</td>
<td>Ω ⊂ (XMIN, XMAX) × (YMIN, YMAX)</td>
</tr>
<tr>
<td>17</td>
<td>XMAX</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>YMIN</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>YMAX</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>RL</td>
<td>current value of $\lambda_h$</td>
</tr>
<tr>
<td>22</td>
<td>R</td>
<td>current value of $\rho(u_h, \lambda_h) = \rho_h$</td>
</tr>
<tr>
<td>23</td>
<td>RLDOT</td>
<td>current value of $\dot{\lambda}_h$</td>
</tr>
<tr>
<td>24</td>
<td>RDOT</td>
<td>current value of $\dot{\rho}_h$</td>
</tr>
<tr>
<td>25</td>
<td>SVAL</td>
<td>current value of smallest singular value</td>
</tr>
<tr>
<td>26</td>
<td>RLSTRT</td>
<td>starting value for $\lambda_h$</td>
</tr>
<tr>
<td>27</td>
<td>RSTRT</td>
<td>starting value for $\rho(u_h, \lambda_h)$</td>
</tr>
<tr>
<td>31</td>
<td>RL0</td>
<td>previous value of $\lambda_h$</td>
</tr>
<tr>
<td>32</td>
<td>R0</td>
<td>previous value of $\rho(u_h, \lambda_h) = \rho_h$</td>
</tr>
<tr>
<td>33</td>
<td>RL0DOT</td>
<td>previous value of $\dot{\lambda}_h$</td>
</tr>
<tr>
<td>34</td>
<td>R0DOT</td>
<td>previous value of $\dot{\rho}_h$</td>
</tr>
<tr>
<td>35</td>
<td>SVAL0</td>
<td>previous value of smallest singular value</td>
</tr>
<tr>
<td>37</td>
<td>ENORM1</td>
<td>estimate for $|u - u_h|_{\mathcal{H}^1(\Omega)}$</td>
</tr>
<tr>
<td>38</td>
<td>UNORM1</td>
<td>the norm $|u_h|_{\mathcal{H}^1(\Omega)}$</td>
</tr>
<tr>
<td>39</td>
<td>ENORM2</td>
<td>estimate for $|u - u_h|_{\mathcal{L}_2(\Omega)}$</td>
</tr>
<tr>
<td>40</td>
<td>UNORM2</td>
<td>the norm $|u_h|_{\mathcal{L}_2(\Omega)}$</td>
</tr>
<tr>
<td>41</td>
<td>RELERP</td>
<td>relative size of solution error $|e_h|<em>{\mathcal{H}^1(\Omega_I)}/|u_h|</em>{\mathcal{H}^1(\Omega_I)}$</td>
</tr>
<tr>
<td>42</td>
<td>EAVE2</td>
<td>arithmetic average of $|e_h|_{\mathcal{H}^1(\Omega_I)}$</td>
</tr>
<tr>
<td>52</td>
<td>STEP</td>
<td>damping step $s$ for Newton’s method</td>
</tr>
<tr>
<td>53</td>
<td>RELER0</td>
<td>relative size of solution error $|e_h|<em>{\mathcal{H}^1(\Omega_I)}/|u_h|</em>{\mathcal{H}^1(\Omega)}$</td>
</tr>
<tr>
<td>54</td>
<td>RELERR</td>
<td>relative size of Newton update $|\delta U|/|U|$</td>
</tr>
<tr>
<td>55</td>
<td>ANORM</td>
<td>maximum diagonal entry in Jacobian matrix</td>
</tr>
<tr>
<td>56</td>
<td>RELRES</td>
<td>the relative residual $|G_k|/|G_0|$</td>
</tr>
<tr>
<td>57</td>
<td>BRATIO</td>
<td>the relative residual $|\hat{G}<em>k|/|\hat{G}</em>{k-1}|$</td>
</tr>
<tr>
<td>58</td>
<td>DNEW</td>
<td>the discrete inner product $-\langle G_u \delta U, G \rangle$</td>
</tr>
<tr>
<td>59</td>
<td>BNORM0</td>
<td>scaling factor $|G_0|$</td>
</tr>
<tr>
<td>60</td>
<td>BMNRM0</td>
<td>scaling factor for $\rho$</td>
</tr>
<tr>
<td>63</td>
<td>RMU</td>
<td>current value of interior point parameter $\mu$</td>
</tr>
<tr>
<td>64</td>
<td>REG4</td>
<td>internal regularization parameter for IPROB = 4</td>
</tr>
<tr>
<td>65</td>
<td>REG5</td>
<td>internal regularization parameter for IPROB = 5</td>
</tr>
<tr>
<td>67</td>
<td>SCLEQN</td>
<td>current value of scalar equation $N - \sigma$</td>
</tr>
<tr>
<td>68</td>
<td>SCALE</td>
<td>scaling factor for scalar equation</td>
</tr>
</tbody>
</table>

Table 2.13: RP array definitions. (Continued next page.)
### 2.7. Parameter Arrays.

<table>
<thead>
<tr>
<th>I</th>
<th>RP(I)</th>
<th>u</th>
<th>definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>69</td>
<td>THETAL</td>
<td>u</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>THETAR</td>
<td>u</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>SIGMA</td>
<td>u</td>
<td>the step $\sigma$ for scalar equation</td>
</tr>
<tr>
<td>72</td>
<td>DELTA</td>
<td>u</td>
<td>Newton update for $\lambda_h$</td>
</tr>
<tr>
<td>73</td>
<td>DRDRL</td>
<td>u</td>
<td>the value of $\partial \rho / \partial \lambda$</td>
</tr>
<tr>
<td>74</td>
<td>SEQDOT</td>
<td>u</td>
<td>the value of $\dot{N}$</td>
</tr>
<tr>
<td>76</td>
<td>QUAL</td>
<td>u</td>
<td>target element quality</td>
</tr>
<tr>
<td>77</td>
<td>ANGMIN</td>
<td>u</td>
<td>target minimum angle</td>
</tr>
<tr>
<td>78</td>
<td>DIAM</td>
<td>u</td>
<td>approximate diameter of $\Omega$</td>
</tr>
<tr>
<td>79</td>
<td>BEST</td>
<td>u</td>
<td>value of TRIGEN quality function</td>
</tr>
<tr>
<td>80</td>
<td>AREA</td>
<td>u</td>
<td>area of $\Omega$</td>
</tr>
<tr>
<td>82</td>
<td>N0</td>
<td>u</td>
<td>degrees of freedom for region $\Omega_I$</td>
</tr>
<tr>
<td>83</td>
<td>E0</td>
<td>u</td>
<td>error for region $\Omega_J$</td>
</tr>
<tr>
<td>84</td>
<td>NF</td>
<td>u</td>
<td>global degrees of freedom</td>
</tr>
<tr>
<td>85</td>
<td>EF</td>
<td>u</td>
<td>global error</td>
</tr>
<tr>
<td>91</td>
<td>RL1</td>
<td>u</td>
<td>value of $\lambda_1$</td>
</tr>
<tr>
<td>92</td>
<td>RL2</td>
<td>u</td>
<td>value of $\lambda_2$</td>
</tr>
<tr>
<td>93</td>
<td>RL3</td>
<td>u</td>
<td>value of $\lambda_3$</td>
</tr>
<tr>
<td>94</td>
<td>RL4</td>
<td>u</td>
<td>value of $\lambda_4$</td>
</tr>
<tr>
<td>95</td>
<td>RL5</td>
<td>u</td>
<td>value of $\lambda_5$</td>
</tr>
<tr>
<td>96</td>
<td>RL6</td>
<td>u</td>
<td>value of $\lambda_6$</td>
</tr>
<tr>
<td>97</td>
<td>RL7</td>
<td>u</td>
<td>value of $\lambda_7$</td>
</tr>
<tr>
<td>98</td>
<td>RL8</td>
<td>u</td>
<td>value of $\lambda_8$</td>
</tr>
<tr>
<td>99</td>
<td>RL9</td>
<td>u</td>
<td>value of $\lambda_9$</td>
</tr>
<tr>
<td>100</td>
<td>RL10</td>
<td>u</td>
<td>value of $\lambda_{10}$</td>
</tr>
</tbody>
</table>

Table 2.13: RP array definitions.

<table>
<thead>
<tr>
<th>I</th>
<th>SP(I)</th>
<th>u</th>
<th>definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ITITLE</td>
<td>u</td>
<td>title for INPLT</td>
</tr>
<tr>
<td>2</td>
<td>FTITLE</td>
<td>u</td>
<td>title for TRIPLT</td>
</tr>
<tr>
<td>3</td>
<td>GTITLE</td>
<td>u</td>
<td>title for GPHPLT</td>
</tr>
<tr>
<td>4</td>
<td>LOGO</td>
<td>u</td>
<td>title for web browser tab</td>
</tr>
<tr>
<td>5</td>
<td>GRFILE</td>
<td>u</td>
<td>file for hard copy graphics output</td>
</tr>
<tr>
<td>6</td>
<td>RWFILE</td>
<td>u</td>
<td>save file for read/write commands</td>
</tr>
<tr>
<td>7</td>
<td>JRFILE</td>
<td>u</td>
<td>read file for journal command</td>
</tr>
<tr>
<td>8</td>
<td>JWFILE</td>
<td>u</td>
<td>write file for journal command</td>
</tr>
<tr>
<td>9</td>
<td>BFILE</td>
<td>u</td>
<td>output file</td>
</tr>
<tr>
<td>10</td>
<td>JTFILE</td>
<td>u</td>
<td>temporary file for journal command</td>
</tr>
</tbody>
</table>

Table 2.14: SP array definitions. (Continued next page.)
Table 2.14: SP array definitions.

<table>
<thead>
<tr>
<th>I</th>
<th>SP(i)</th>
<th>u</th>
<th>definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>IOMSG</td>
<td></td>
<td>error message string</td>
</tr>
<tr>
<td>12</td>
<td>CMD</td>
<td></td>
<td>current command string</td>
</tr>
</tbody>
</table>

PLTMG has seven labeled common blocks:

- Common /pltmg1/ic(3,363),jc(12)
- Common /pltmg2/c(2,78),wt(78),np1(13)
- Common /pltmg3/c(3,746),wt(746),np2(22)
- Common /pltmg4/fc(2541)
- Common /pltmg5/cb(65,65),cd(12,65),cs(12,45),iptr(12),jptr(12)
- Common /pltmg6/path(101,6)
- Common /pltmg7/time(3,50),hist(22,30)

Common block PLTMG1 contains basic definitions of the family of finite elements. Blocks PLTMG2 and PLTMG3 contain definitions of quadrature rules for one dimensional integrals on intervals (Gauss Quadrature), and two dimensional integrals on triangles, from Zhang, Cui, and Liu, [67]. Block PLTMG4 contains information used in the two level HB solver described in Section 4.3. Block PLTMG5 contains information used in the evaluation of basis functions on transition elements. Block PLTMG6 collects data on various aspects of continuation problems, IPROB = 3 (See Section 4.6). Block PLTMG7 collects statistical data on various aspects of the calculation.

### 2.8 Coefficient Functions.

Several routines in the package require knowledge of the partial differential equation (1.1), the boundary conditions (1.2), the functional $\rho$ in (1.3), and, on occasion, an alternate function of the solution. This information is provided by the user through subroutines A1XY, A2XY, FXY, GNXY, GDXY, P1XY, P2XY, and QXY.

Subroutines A1XY, A2XY, FXY, and P1XY have identical argument lists.

*Call A1XY( X, Y, U, UX, UY, RL, ITAG, VALUES ),*  
*Call A2XY( X, Y, U, UX, UY, RL, ITAG, VALUES ),*  
*Call P1XY( X, Y, U, UX, UY, RL, ITAG, VALUES ),*  
*Call FXY( X, Y, U, UX, UY, RL, ITAG, VALUES ).*

In these subroutines, all of the arguments except VALUES are provided as
### 2.8. Coefficient Functions.

<table>
<thead>
<tr>
<th>IFLAG</th>
<th>general return codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>normal return</td>
</tr>
<tr>
<td>25</td>
<td>wrong input data structure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IFLAG</th>
<th>PLTMG and TRIGEN errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>zero pivot in sparse factorization</td>
</tr>
<tr>
<td>2</td>
<td>Newton method line search failed</td>
</tr>
<tr>
<td>5</td>
<td>solution zero; cannot compute error estimate</td>
</tr>
<tr>
<td>6</td>
<td>illegal problem type</td>
</tr>
<tr>
<td>7</td>
<td>continuation procedure failed</td>
</tr>
<tr>
<td>10</td>
<td>multigraph iteration failed to converge</td>
</tr>
<tr>
<td>11</td>
<td>Newton (Newton/DD) iteration failed to converge</td>
</tr>
<tr>
<td>24</td>
<td>Error on one or more MPI processes</td>
</tr>
<tr>
<td>48</td>
<td>MPI was off for a command needing MPI</td>
</tr>
<tr>
<td>49</td>
<td>\textsc{NPROC} &gt; \textsc{NTF} in load balance</td>
</tr>
<tr>
<td>71</td>
<td>no interface unknowns in DD solver</td>
</tr>
<tr>
<td>72</td>
<td>\textsc{IPATH} array not created</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IFLAG</th>
<th>storage errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>82</td>
<td>storage exhausted in array \textsc{IPATH}</td>
</tr>
<tr>
<td>83</td>
<td>storage exhausted in arrays \textsc{ITNODE} and \textsc{ITDOF}</td>
</tr>
<tr>
<td>84</td>
<td>storage exhausted in arrays \textsc{VX} and \textsc{VY}</td>
</tr>
<tr>
<td>85</td>
<td>storage exhausted in array \textsc{GF}</td>
</tr>
<tr>
<td>86</td>
<td>storage exhausted in arrays \textsc{IBNDRY} and \textsc{SF}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IFLAG</th>
<th>data errors for triangulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>−31</td>
<td>illegal \textsc{ITNODE}(K,*), (K = 1, 2, 3)</td>
</tr>
<tr>
<td>−32</td>
<td>overlapping triangles in \textsc{ITNODE}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IFLAG</th>
<th>data errors for triangulation and skeleton</th>
</tr>
</thead>
<tbody>
<tr>
<td>−40</td>
<td>illegal value for \textsc{NVF}, \textsc{NTF}, or \textsc{NBF}</td>
</tr>
<tr>
<td>−41</td>
<td>illegal \textsc{IBNDRY}(K,*), (K = 1, 2)</td>
</tr>
<tr>
<td>−42</td>
<td>illegal \textsc{IBNDRY}(3,*)</td>
</tr>
<tr>
<td>−43</td>
<td>illegal \textsc{IBNDRY}(4,*)</td>
</tr>
<tr>
<td>−44</td>
<td>incorrect circle center coordinates</td>
</tr>
<tr>
<td>−45</td>
<td>arc greater than (\pi/2) in length</td>
</tr>
<tr>
<td>−46</td>
<td>error in linked edges</td>
</tr>
<tr>
<td>−47</td>
<td>boundary vertex without two boundary edges</td>
</tr>
<tr>
<td>−48</td>
<td>\textsc{ITNODE} and \textsc{IBNDRY} are not consistent</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IFLAG</th>
<th>data errors for skeleton</th>
</tr>
</thead>
<tbody>
<tr>
<td>−51</td>
<td>illegal \textsc{ITNODE}(1,*)</td>
</tr>
<tr>
<td>−52</td>
<td>illegal \textsc{ITNODE}(2,*)</td>
</tr>
<tr>
<td>−53</td>
<td>skeleton tracing error</td>
</tr>
<tr>
<td>−54</td>
<td>region specified in clockwise order</td>
</tr>
<tr>
<td>−55</td>
<td>illegal \textsc{ITNODE}(3,*)</td>
</tr>
</tbody>
</table>

Table 2.15. Error flag values.
input. In particular $(X,Y) \in \Omega$ is the evaluation (quadrature) point, and 

$$U = u_h(X,Y),$$

$$UX = \frac{\partial u_h}{\partial x}(X,Y),$$

$$UY = \frac{\partial u_h}{\partial y}(X,Y),$$

$$RL = \lambda_h,$$

For the parameter identification problem, $RL$ is an array for size $NRL$ with the value of the vector $\lambda_h$, and for the distributed control problem, $RL = \lambda_h(X,Y)$. The parameter $ITAG=ITNODE(5,I)$ is the user specified label associated with element $\tau_I \in \mathcal{T}$ containing $(X,Y)$. From this input data, the user provides values of the given function and its derivatives in the array $VALUES$. This array is of size $4+NRL$. All entries are initially set to zero by the calling routine; thus the user need supply only nonzero values.

To simplify this process, PLTMG supplies a labeled common block

```plaintext
common /VAL0/ K0, KU, KX, KY, KL
```

containing a predefined list of integer pointers mapping function and derivative values to particular entries in the $VALUES$ array. The details of this mapping are given in Table 2.16 for the case of $f$; the identical mapping is used for $a_1$, $a_2$ and $p_1$.

<table>
<thead>
<tr>
<th>pointer</th>
<th>index</th>
<th>VALUES(·)</th>
</tr>
</thead>
<tbody>
<tr>
<td>K0 = 1</td>
<td>K0</td>
<td>$f$</td>
</tr>
<tr>
<td>KU = 2</td>
<td>KU</td>
<td>$fu$</td>
</tr>
<tr>
<td>KX = 3</td>
<td>KX</td>
<td>$fu_x$</td>
</tr>
<tr>
<td>KY = 4</td>
<td>KY</td>
<td>$fu_y$</td>
</tr>
<tr>
<td>KL = 5</td>
<td>KL+J−1</td>
<td>$f\lambda_j$</td>
</tr>
<tr>
<td></td>
<td>$1 \leq J \leq NRL$</td>
<td></td>
</tr>
</tbody>
</table>

**Table 2.16. VALUES array for subroutine FXY.**

For example, if 

$$f = \lambda \frac{\partial u}{\partial x} + u^2,$$

then the following code fragment would be included in Subroutine FXY.

```plaintext
VALUES(K0)= RL * UX + U**2
VALUES(KX)= RL
VALUES(KU)= 2 * U
VALUES(KL)= UX
```
The subroutine corresponding to $p_2$ is $P2XY$ and is called using

\[ \text{Call P2XY( } X, Y, DX, DY, U, UX, UY, RL, ITAG, JTAG, VALUES ).} \]

The arguments are a superset of those of the previous subroutines, and all arguments with the same name serve the same purpose. This routine is called only with points $(X,Y)$ lying on some edge $e_J \in \Gamma$. The additional arguments $(DX, DY)$ are the unit normal direction for the edge, and $JTAG=IBNDRY(7,J)$ is the user specified label for the given edge. The mapping given in Table 2.16 is used here as well.

The subroutine corresponding to $g_1$ is $GNXY$ and is called using

\[ \text{Call GNXY( } X, Y, U, RL, ITAG, VALUES ).} \]

This routine is called only for points $(X,Y) \in \partial \Omega_1$, and as in the previous cases, all arguments except the array $VALUES$ are input. In this case $ITAG=IBNDRY(7,I)$ is the user supplied label for the edge, and $VALUES$ is an array of size $2 + NRL$. Here the labeled common block

\[ \text{common } /\text{VAL1} / K0, KU, KL \]

assists in mapping function and derivative values to particular entries in the $VALUES$ array. The details of the mapping are given in Table 2.17.

\[
\begin{array}{|c|c|c|}
\hline
\text{pointer} & \text{index} & VALUES(\cdot) \\
\hline
K0 = 1 & K0 & g \\
KU = 2 & KU & g_u \\
KL = 3 & KL + J - 1 & g_{\lambda,J} \\
1 \leq J \leq NRL & & \\
\hline
\end{array}
\]

\textbf{Table 2.17. VALUES array for subroutine GNXY.}

The subroutine corresponding to $g_2$ is $GDXY$ and is called using

\[ \text{Call GDXY( } X, Y, RL, ITAG, VALUES ).} \]

This routine also supplies the upper and lower bounds for the inequality constraints on $u_h$ for the obstacle problem, bounds on $\lambda_h$ in the case that $\lambda = \lambda(x,y)$, and the initial guess $u_0$, for the solution $u_h$. For parameter identification problems, the Lagrange multiplier can be initialized using $v_0$, and for optimal control problems the Lagrange multiplier can be initialized with $v_0$ and $\lambda(x,y)$ can be initialized with $\lambda_0$. When called to supply a Dirichlet boundary condition, $(X,Y) \in \partial \Omega_2$ and $ITAG=IBNDRY(7,I)$ is an edge label. When called in regard to inequality constraints and the initial guess, $(X,Y) \in \Omega$ and $ITAG=ITNODE(5,I)$ is the element
label supplied by the user. Similar to the other routines, VALUES is an output array of size $3 + 4 \cdot \text{NRL}$. It’s entries can be conveniently accessed through pointers provided in the labeled common block

```
common /VAL2/ K0, KL, KLB, KUB, KIC, KIM, KIL
```

The details are provided in Table 2.18.

<table>
<thead>
<tr>
<th>pointer</th>
<th>index</th>
<th>VALUES(·)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K0 = 1$</td>
<td>$K0$</td>
<td>$g$</td>
</tr>
<tr>
<td>$KL = 2$</td>
<td>$KL + J - 1$</td>
<td>$g_{\lambda_j}$</td>
</tr>
<tr>
<td>$KLB = 2 + NRL$</td>
<td>$KLB + J - 1$</td>
<td>$\bar{u}_j, \lambda_j$</td>
</tr>
<tr>
<td>$KUB = 2 + 2 NRL$</td>
<td>$KUB + J - 1$</td>
<td>$u_0$</td>
</tr>
<tr>
<td>$KIC = 2 + 3 NRL$</td>
<td>$KIC$</td>
<td>$v_0$</td>
</tr>
<tr>
<td>$KIM = 3 + 3 NRL$</td>
<td>$KIM$</td>
<td>$\lambda_{0,j}$</td>
</tr>
<tr>
<td>$KIL = 4 + 3 NRL$</td>
<td>$KIL + J - 1$</td>
<td>$1 \leq J \leq \text{NRL}$</td>
</tr>
</tbody>
</table>

Table 2.18. VALUES array for subroutine GDXY.

Subroutine QXY is

```
Call QXY( X, Y, U, UX, UY, RL, ITAG, VALUES)
```

This routine provides the alternate function to display in TRIPLT and the alternate function for adaptive algorithms in TRIGEN. The arguments are defined as in the other coefficient functions. The output array VALUES has dimension 4; It’s entries can be conveniently accessed through pointers provided in the labeled common block

```
common /VAL3/ KF, KF1, KF2, KAD
```

whose entries are documents in Table 2.19.

<table>
<thead>
<tr>
<th>pointer</th>
<th>index</th>
<th>VALUES(·)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K0 = 1$</td>
<td>$K0$</td>
<td>alternate scalar function for TRIPLT</td>
</tr>
<tr>
<td>$KF1 = 2$</td>
<td>$KF1$</td>
<td>first component of vector function for TRIPLT</td>
</tr>
<tr>
<td>$KF2 = 3$</td>
<td>$KF2$</td>
<td>second component of vector function for TRIPLT</td>
</tr>
<tr>
<td>$KAD = 4$</td>
<td>$KAD$</td>
<td>alternate function for adaptive algorithms in TRIGEN</td>
</tr>
</tbody>
</table>

Table 2.19. VALUES array for subroutine QXY

In the case of a singular Neumann problem (e.g., $a_1 \equiv u_x$, $a_2 \equiv u_y$, $f \equiv 0$, and $\partial \Omega_1 = 0$ in (1.1)), the solution $u$ not unique but is determined only up to an
2.9. Sparse Matrix Storage.

arbitrary constant. Setting the switch ISING = 1 causes both right hand sides and solutions in all linear systems to be orthogonalized with respect to constants, in effect computing least squares solutions in the orthogonal complement subspace. In other situations, one should set ISING = 0.

2.9 Sparse Matrix Storage.

Although sparse matrices are presently generated internally within PLTMG, it may still be of interest to understand the data structures involved. This version of PLTMG uses two variants of a basic sparse matrix data structure – a point version, where matrix elements are simple scalar values, and a block version where matrix elements are allowed to be blocks of arbitrary size. The block version is of interest, since the degrees of freedom associated with a single edge or element interior form a so-called clique within the graph of the matrix. These correspond to dense blocks within the sparse matrix if all members of a clique are ordered consecutively, as is the case here. Taking advantage of these dense blocks can reduce the integer overhead and indirect addressing associated with processing those cliques.

We begin discussion with the point version of the data structure. Here matrices are stored in the sparse matrix format described in [5] using an integer array JA and a real array A. As an example, consider the $4 \times 4$ matrix given by

\[
A = \begin{pmatrix}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{pmatrix}.
\]  

(2.1)

This matrix is stored in JA and A as illustrated in Table 2.20. All nonzeros are stored in the array A. First the diagonal entries are stored, followed by the upper triangular entries, stored row by row. If the matrix is nonsymmetric, this is followed by the lower triangular entries, stored column by column. Symmetric and nonsymmetric storage is governed by the parameter ISPD as indicated in Table 2.20.

<table>
<thead>
<tr>
<th>ISPD</th>
<th>storage/iteration options</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>nonsymmetric/biconjugate gradient</td>
</tr>
<tr>
<td>1</td>
<td>symmetric/conjugate gradient</td>
</tr>
</tbody>
</table>

Table 2.20. The values of ISPD.

The first $NDF + 1$ entries of JA are pointers. In particular, entries $JA(I)$ to $JA(I+1) - 1$ of the JA array contain column indices for nonzeros in row $I$ of the strict upper triangle. As illustrated in Table 2.21, the column indices stand in correspondence to the nonzeros of the upper triangle stored in the array $A$. If nonsymmetric storage is used, entries of the transposed lower triangle are stored in the same order as the upper triangle.
Table 2.21. Sparse matrix data structures. JA has 9 entries. A has 9 entries if ISPD = 1 or 13 entries if ISPD = 0.

<table>
<thead>
<tr>
<th>I</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>JA(I)</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A(I)</td>
<td>$a_{11}$</td>
<td>$a_{22}$</td>
<td>$a_{33}$</td>
<td>$a_{44}$</td>
<td>$a_{12}$</td>
<td>$a_{14}$</td>
<td>$a_{23}$</td>
<td>$a_{24}$</td>
<td>$a_{21}$</td>
<td>$a_{41}$</td>
<td>$a_{32}$</td>
<td>$a_{42}$</td>
<td></td>
</tr>
</tbody>
</table>

Now suppose the elements $a_{ii}$ in (2.1) are $k_i \times k_i$ square matrices. Then the off-diagonal blocks $a_{i,j}$ are $k_i \times k_j$ rectangular blocks. Suppose that there are $NB$ blocks, where

$$\text{NDF} = \sum_{i=1}^{NB} k_i$$

The JA array for the block case is identical to the point case, except that now entries refer to block rows and columns rather than individual elements. This could be much smaller than the point version of the JA array. For example, a mesh with NVF vertices and all elements of degree $p$ will have approximately $\text{NDF} \approx p^2 \text{NVF}$ degrees of freedom and a point JA array with $O(p^4 \text{NVF})$ entries. On the other hand, for this case $NB \approx 6 \times \text{NVF}$, and the corresponding block JA array will have about $39 \times \text{NVF}$ entries.

Additionally we need an array $IBS$ of size $NB$ to indicate the sizes of the diagonal blocks

$$IBS(I) = k_I \quad 1 \leq I \leq NB.$$ 

The A array in this case is more complicated. Following the pattern of the scalar case, we store the diagonal blocks first, followed by the upper triangular blocks, stored (block) row-wise. If ISPD = 0, the upper triangle is followed by the lower triangular block, stored (block) column-wise. The individual diagonal blocks are stored in the same pattern; the diagonal stored first, followed by the upper triangle, stored row-wise, and if ISPD = 0, this is followed by the lower triangle stored column-wise. The upper triangular blocks are stored row-wise, and the lower triangular blocks, if present, are stored column-wise.

To access this data, we need an additional integer array $JAP$ of pointers, where $JAP(I)$ indicates the location in the $A$ array where the block corresponding to $JA(I)$ begins. This array is the same size as $JA$ (plus one for convenience).

For the case ISPD = 1, $JAP(1) = 1$ and

$$JAP(I+1) = JAP(I) + \{IBS(I) \times (IBS(I) + 1)\}/2,$$

while for ISPD = 0

$$JAP(I+1) = JAP(I) + IBS(I)^2,$$

for $1 \leq I \leq NB$. Note that the value of $JAP(NB+1)$ is defined. For the upper triangle, we have $JAP(NB+2) = JAP(NB+1)$. For $I = 1, 2, \ldots NB$ and $JA(I) \leq$
2.9. Sparse Matrix Storage.

\( K \leq JA(I+1) - 1 \), we have

\[ JAP(K+1) = JAP(K) + IBS(I) \times IBS(JA(K)). \]

The array \( JAP \) can be computed once and saved, but we prefer to compute it as needed from the \( IBS \) and \( JA \) arrays. Some of our problem classes involve several sparse matrices, some symmetric and some nonsymmetric. For this case, one instance of the \( IBS \) and \( JA \) arrays can be used for all sparse matrices, independent of their symmetry, and routines that need \( JAP \) (e.g., a routine to compute a matrix-vector multiply) can compute it based on the symmetry status of the particular matrix involved.

Data structures \( JU \) and \( U \) are analogous to \( JA \) and \( A \), respectively, and contain the (incomplete) \( A \approx LDU \) factorization, where \( D \) is (block) diagonal, \( U \) is unit (block) upper triangular, and \( L \) is unit (block) lower triangular, with \( L^t = U \) if \( A^t = A \).
Chapter 3
Mesh Generation

3.1 Overview.

Subroutine TRIGEN creates or adaptively modifies the data structures defining the region Ω. There are options to generate a triangulation from a skeleton, adaptively refine or unrefine a triangulation, uniformly refine a triangulation, and adaptively smooth the vertices of a triangulation. TRIGEN also has several options for partitioning and mesh management in parallel computation environments. The parameter IADAPT specifies various options for TRIGEN, summarized in Table 3.1.

TRIGEN is called using the statement

\[
\text{Call TRIGEN( VX, VY, SF, ITNODE, IBNDRY, ITDOF, IPATH,}
\]

\[
E, \text{ IP, RP, SP, IU, RU, SU, GF, QXY, SXY )}
\]

Except for the case IADAPT = 5, on input the arrays VX, VY, SF, ITNODE, and IBNDRY should define a triangulation. For IADAPT = 5, the input should be a skeleton. The arrays IU, RU, and SU are broadcast and received in MPI communication steps, but are not directly used in TRIGEN. When TRIGEN is used to adaptively modify an existing triangulation the procedures generally rely on local a posteriori error estimates for the finite element approximation, although some options are provided for adaptation based on other functions.

If IADAPT = −K for 1 ≤ k ≤ 3, the refinement and/or unrefinement or adaptive mesh smoothing processes are carried out using interpolation errors for the function QXY in place of the a posteriori error estimates. In particular, for a given element t of degree p, let \( q_{p+1} \) denote the interpolating polynomial for QXY of degree \( p + 1 \), characterized by nodes at the usual Lagrange lattice points of t. In this situation, we can use the (constant) derivatives of order \( p + 1 \) of \( q_{p+1} \) in place of the corresponding recovered derivatives for \( u_h \). Once this substitution is made, the adaptive algorithms proceed in the usual fashion.

We do not anticipate that this option will be used much; it was originally implemented to allow subroutine TRIGEN to be debugged independently of subrou-
Table 3.1. Some options use a posteriori error estimates for the computed solution $u_h$ or interpolation errors for the alternative function $QXY$. Other options require MPI for parallel communication.

<table>
<thead>
<tr>
<th>IADAPT</th>
<th>mesh generation option</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>error estimates only</td>
</tr>
<tr>
<td>1</td>
<td>refine or unrefine mesh using $u_h$</td>
</tr>
<tr>
<td>-1</td>
<td>refine or unrefine mesh using $QXY$</td>
</tr>
<tr>
<td>2</td>
<td>unrefine and refine mesh using $u_h$</td>
</tr>
<tr>
<td>-2</td>
<td>unrefine and refine mesh using $QXY$</td>
</tr>
<tr>
<td>3</td>
<td>smooth mesh points using $u_h$</td>
</tr>
<tr>
<td>-3</td>
<td>smooth mesh points using $QXY$</td>
</tr>
<tr>
<td>4</td>
<td>uniform mesh refinement</td>
</tr>
<tr>
<td>-4</td>
<td>uniform degree refinement</td>
</tr>
<tr>
<td>5</td>
<td>skeleton $\rightarrow$ triangulation</td>
</tr>
<tr>
<td>6</td>
<td>load balance (MPI)</td>
</tr>
<tr>
<td>7</td>
<td>reconcile mesh (MPI)</td>
</tr>
</tbody>
</table>

tine PLTMG. On the other hand, there may be special cases where some functional other than the a posterior error estimate for $\|\nabla (u - u_h)\|_{L^2(t)}$ should be optimized. Note that if TRIGEN is called before a solution $u_h$ is computed by PLTMG, values of the arguments $U$, $UX$, $UY$, and $RL$ provided by TRIGEN to QXY are arbitrary and should be ignored.

### 3.2 Creating a Triangulation from a Skeleton.

When $IADAPT = 5$, on input the arrays $VX$, $VY$, $SF$, $ITNODE$, and $IBNDRY$ should define a skeleton as described in Section 2.4. TRIGEN triangulates the subregions defining the skeleton in the order that they are given in $ITNODE$, taking into account shared internal boundaries and the symmetry requirements.

Let $t$ be a triangle with area $a$ and side lengths $h_1$, $h_2$, and $h_3$. The quality of $t$, $q(t)$, is measured using the formula

$$q(t) = 4\sqrt{3}a/(h_1^2 + h_2^2 + h_3^2).$$

The function $q(t)$ is normalized to equal one for an equilateral triangle and to approach zero for triangles with small angles. In attempting to compute a high quality triangulation, TRIGEN uses

$$q(t) \geq .6$$

as a test for acceptability of a triangle (sufficiently small interior angles on the boundaries of the subregions $\Omega_i$ could cause (3.2) to be violated).
The triangulation process for those regions for which $ITNODE(3, I) \neq 0$ is simple and is carried out by generating the appropriate affine mapping. The triangulation process for subregions with $ITNODE(3, I) = 0$ is somewhat complicated but embodies three straightforward heuristics.

Given a subregion viewed as a polygon (possibly with curved edges, and interior angles of size $\pi$ or greater), $TRIGEN$ first tries to reduce the order of the polygon by one by “chopping” off a triangle using a vertex with small interior angle. Inequality (3.2) and several less obvious conditions must be satisfied for a successful chop. When the chopping strategy is no longer successful, $TRIGEN$ checks to see if the remaining polygon is convex with six or fewer sides. If it is, $TRIGEN$ tries to triangulate the entire remaining subregion by adding the centroid as a vertex and connecting it to each boundary vertex. All the resulting triangles must satisfy (3.2) and some other conditions for this strategy to be successful.

If the second strategy fails or is inapplicable, $TRIGEN$ tries to break the polygon into two smaller polygons by connecting two nonadjacent vertices by a straight line. $TRIGEN$ excludes many potential cuts as geometrically infeasible or otherwise undesirable. From the remaining possibilities $TRIGEN$ picks the cut that maximizes the minimum of the four interior angles the cut creates. $TRIGEN$ then applies the three strategies to the two newly created polygons in recursive fashion. After the region has been successfully triangulated, $TRIGEN$ tries to improve the triangulation by (locally) rearranging edges and adjusting vertex locations such that the criterion (3.2) is optimized.

The user can control the triangulation process to some extent through the parameters $HMAX$ and $GRADE$. Element size is controlled by $HMAX$. Normally, one should choose $0 < HMAX \leq 1$. $TRIGEN$ then attempts to create triangles with edges shorter than $HMAX \cdot \text{diam}(\Omega)$. If $HMAX \leq 0$ or $HMAX > 1$, $TRIGEN$ will reset $HMAX = 1$. Setting $HMAX$ only places an upper bound on triangle sizes; the sizes of the triangles actually generated depend strongly on the geometry of the $\Omega_i$ and may not achieve the bound.

$GRADE$ is (approximately) the largest ratio of sizes of elements sharing a common edge ($1/GRADE$ is the smallest ratio). $GRADE$ should be set on the interval $1.5 \leq GRADE \leq 2.5$; values outside this interval are set to the appropriate end point. Generally speaking, smaller values of $GRADE$ result in smoother transitions from regions of large elements to those of small elements, and a higher overall quality measured by (3.1). On the other hand, larger values of $GRADE$ tend to produce meshes with fewer elements, more rapid transitions in element size, and lower overall quality. One may have to experiment to achieve the proper balance between these conflicting objectives.

For example, consider the domain pictured in Figure 3.1, top left. The remaining pictures in Figure 3.1 show triangulations generated by $TRIGEN$ for various values of $HMAX$ and $GRADE$, illustrating their effect on the resulting triangulation.

The pictures are made by $INPLT$ (see Section 5.3), which draws the mesh with elements colored according to the quality measure $q(t)$ in (3.1). In the pictures, an element is “good” if $q(t) \geq \sqrt{3}/2$, “fair” if $0.6 \leq q(t) < \sqrt{3}/2$, and “poor” if $q(t) < 0.6$. This is an interesting region to triangulate because the two narrow subregions at
A skeleton with $NTF = 6$, $NVF = 30$, $NBF = 35$ (left). The triangulation for $HMAX = 0$, $GRADE = 1.5$ has $NTF = 509$, $NVF = 292$ (right).

The triangulation for $HMAX = 0$, $GRADE = 2.0$ has $NTF = 329$, $NVF = 199$ (left). The triangulation for $HMAX = 0$, $GRADE = 2.5$ has $NTF = 262$, $NVF = 163$ (right).

The triangulation for $HMAX = .03$, $GRADE = 1.5$ has $NTF = 1269$, $NVF = 695$ (left). The triangulation for $HMAX = .06$, $GRADE = 1.5$ has $NTF = 634$, $NVF = 361$ (right).

The triangulation for $HMAX = .03$, $GRADE = 2.5$ has $NTF = 849$, $NVF = 480$ (left). The triangulation for $HMAX = .06$, $GRADE = 2.5$ has $NTF = 377$, $NVF = 227$ (right).

Figure 3.1.
the top require small elements. **TRIGEN** tries to use larger elements in the larger subregions, but is constrained by the choices of **HMAX** and **GRADE**. Decreasing **HMAX** or **GRADE** tends to improve the overall quality of the triangulation, at the expense of introducing more elements.

3.3 A Posteriori Error Estimates.

Of central importance to the adaptive procedures is the computation of a posteriori local error estimates [2, 1, 64, 66]. In the case of piecewise polynomials of degree \( p \), our a posteriori error estimate is based on a superconvergent approximation of the derivatives of \( u \) of order \( p \) [36, 37, 38]. In particular, given the finite element function \( u_h \), we compute the piecewise linear vector functions \( S_h Q_h \frac{\partial^k u}{\partial x^k \partial y} u_h \), for \( 0 \leq k \leq p \). where \( Q_h \) is the \( L_2 \) projection from the space of discontinuous piecewise constant functions into the space of continuous piecewise linear polynomials, and \( S_h \) is a smoothing operator based on the discrete Laplace operator; in **PLTMG**, we take \( m \leq 1 \). See [36, 37, 38] for details. For meshes with variable \( p \), we recover derivatives patchwise, processing all elements of the same degree in the same patch.

The switch **IERRSW** allows the user to control the continuity of the recovered derivatives. The options are specified in 3.2. In some problems, one expects the gradient or higher derivatives of the solution to be discontinuous, typically due to discontinuities in the coefficient functions. If **IERRSW** = 1, a patchwise continuous recovery is made. The user defines the patch boundaries by specifying different values of **ITNODE(5,*)** for different patches. The parameter **NDL** is the combined order of the (block diagonal) linear systems that are used in the recovery.

<table>
<thead>
<tr>
<th><strong>IERRSW</strong></th>
<th>error recovery option</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>globally continuous recovery</td>
</tr>
<tr>
<td>1</td>
<td>patchwise continuous recovery</td>
</tr>
</tbody>
</table>

**Table 3.2.** Patches are defined using element labels **ITNODE(5,*)**.

Using these recovered derivatives, we compute a local error estimate \( \varepsilon_t \) for \( t \in \mathcal{T} \). Suppose the finite element space consists of continuous piecewise polynomials of degree \( p \), and denote by \( u_p \) the usual Lagrange interpolant. In [40], it is shown that in many cases interpolation error is both an upper and lower bound on the finite element error,

\[
C_1 \| \nabla (u - u_p) \|_{L_2(\Omega)} \leq \| \nabla (u - u_h) \|_{L_2(\Omega)} \leq C_2 \| \nabla (u - u_p) \|_{L_2(\Omega)},
\]

implying that error indicators based on interpolation error are both reliable and efficient. In [27], constants involved in such estimates are numerically computed for a variety of finite element spaces, including the Lagrange family of triangular elements used in **PLTMG**.

Our estimate is motivated by noting that under certain circumstances, \( \| \nabla (u_{p+1} - u_p) \|_{L_2(\Omega)} \) is an asymptotically exact estimate of \( \| \nabla (u - u_h) \|_{L_2(\Omega)} \). This is known...
for the cases $p = 1$ and $p = 2$ [37, 38]. Since the usual interpolation points for $u_p$ and generally not a subset of those for $u_{p+1}$, on each individual element $t$, we replace $u_{p+1}$ by $	ilde{u}_{p+1} = u_p + e_{p+1}$, where $e_{p+1}$ is a locally defined polynomial of degree $p + 1$ that is zero at the interpolation points for the polynomial of degree $p$ and has the same (constant) derivatives of order $p + 1$ as $u_{p+1}$ (see Figure 1.1). Such polynomials form a (local) vector space of dimension $p + 2$. For example, $e_2$ is a locally defined quadratic polynomial with value zero at all vertices of the mesh. On a given element $t$, $e_2$ can be expressed as a linear combination of three quadratic “bump functions” $q_k$ associated with the edge midpoints of $t$,

$$e_2 = \sum_{k=1}^{3} \ell_k^2 t_k^k M_t t_k q_k(x,y)$$

(3.3)

where $\ell_k$ is the length of edge $k$, $t_k$ is the unit tangent, and

$$M_t = -\frac{1}{2} \begin{pmatrix} \partial_{xx} u_2 & \partial_{xy} u_2 \\ \partial_{xy} u_2 & \partial_{yy} u_2 \end{pmatrix}.$$  

is the Hessian matrix. All terms on the right hand side of (3.3) are known except for the second derivatives appearing in the Hessian matrix $M_t$. In our local error indicator, we simply approximate the second derivatives in the Hessian matrix $M_t$ using derivatives of $S^m Q_h \nabla u_h$. In particular, let

$$\tilde{M}_t = -\frac{1}{2} \begin{pmatrix} \partial_x S^m Q_h \partial_x u_h & \partial_x S^m Q_h \partial_y u_h \\ \partial_y S^m Q_h \partial_x u_h & \partial_y S^m Q_h \partial_y u_h \end{pmatrix},$$

$$\bar{M}_t = \frac{\alpha_t}{2} (\tilde{M}_t + \tilde{M}_t^T),$$

$$\epsilon_t = \sum_{k=1}^{3} \ell_k^2 t_k^k \tilde{M}_t t_k q_k(x,y).$$

(3.4)

The normalization constant $\alpha_t$ is chosen such that the local error indicator $\eta_t$ satisfies

$$\eta_t \equiv \| \nabla \epsilon_t \|_{L^2(t)} = \| (I - S^m Q_h) \nabla u_h \|_{L^2(t)}.$$  

Normally we expect that $\alpha_t \approx 1$, which is likely to be the case in regions where the Hessian matrix for the true solution is well defined. Near singularities, $u$ is not smooth and we anticipate difficulties in estimating the Hessian. For elements near such singularities, $\alpha_t$ provides a heuristic for partly compensating for poor approximation. For the cases $e_{p+1}$, $p > 1$, more complicated formulas of similar nature are used. In particular, $\epsilon_t$ is expressed in terms of parameters describing the geometry of $t$, and the derivatives of order $p + 1$ in $t$, that are obtained from $\partial_x S^m Q_h \partial_x^{p-k} \partial_y^{p-k} u_h$, and $\partial_y S^m Q_h \partial_x^{p-k} \partial_y^{p-k} u_h$, for $0 \leq k \leq p$, in a fashion analogous to the case $p = 1$ described above. Global a posteriori estimates $\| \epsilon_t \|_{L^2(\Omega)}$ and $\| \nabla \epsilon_t \|_{L^2(\Omega)}$ are stored as the parameters $\text{ENORM2}$ and $\text{ENORM1}$, respectively.

In the case of parameter identification problems, the error in the Lagrange multiplier $\tilde{\epsilon}_t$ is computed by the a similar procedure to that described above. The
3.4 Adaptive Mesh Refinement and Unrefinement.

Local error indicator is given by

$$\eta_t = \left\{ |\nabla \epsilon_t|_{L^2(t)}^2 + |\nabla \tilde{\epsilon}_t|_{L^2(t)}^2 \right\}^{1/2}.$$ 

In the case of optimal control problems, errors in both the Lagrange multiplier $\tilde{\epsilon}_t$ and the control $\hat{\epsilon}_t$ are computed, and the local error indicator is given by

$$\eta_t = \left\{ |\nabla \epsilon_t|_{L^2(t)}^2 + |\nabla \tilde{\epsilon}_t|_{L^2(t)}^2 + |\nabla \hat{\epsilon}_t|_{L^2(t)}^2 \right\}^{1/2}.$$ 

In both the cases, the definitions of $ENORM1$ and $ENORM2$ are similarly modified.

Local error estimates are stored in the array $E$. This array has $MAXT$ rows and two columns. Row $I$ of $E$ corresponds to element $t_I$, with entry $E(I,1) = \eta_{t_I}^2$, and $E(I,2) = \alpha_{t_I}$. The contents of the $E$ array can be graphically displayed using TRIPLT (see Section 5.2). The $E$ array is typically updated in TRIGEN as part of adaptive algorithms that make use of the error estimates. Thus if one is interested in viewing uncorrupted versions of these quantities, plot them after calling TRIGEN with $IADAPT = 0$.

3.4 Adaptive Mesh Refinement and Unrefinement.

Our adaptive algorithms are based on work described in Nguyen [57], Bank and Nguyen [22, 24, 23, 25], Bank and Deotte [12], and Bank and Yserentant [41]. When $IADAPT = 1$, the current mesh is adaptively refined or unrefined. When the target value $NDTRGT > NDF$, the mesh is refined, while if $NDTRGT < NDF$, the mesh is unrefined. In either case, the goal is to achieve the best possible mesh using (approximately) $NDTRGT$ degrees of freedom. The switch $IRTYPE$ specifies the type of adaptivity to be used – $h$, $p$, or $hp$ as indicated in Table 3.3.

<table>
<thead>
<tr>
<th>$IRTYPE$</th>
<th>adaptivity option</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$hp$ refinement / unrefinement</td>
</tr>
<tr>
<td>1</td>
<td>$h$ refinement / unrefinement</td>
</tr>
<tr>
<td>-1</td>
<td>$p$ refinement / unrefinement</td>
</tr>
</tbody>
</table>

Table 3.3. Adaptivity options using $IRTYPE$.

When $IADAPT = 2$, both refinement and unrefinement are employed. First, the mesh is unrefined to obtained a mesh with approximately $NDTRGT < NDF$ degrees of freedom. The mesh is then refined to obtain a mesh with approximately $NDF$ degrees of freedom. The output triangulation thus has approximately the same number of degrees of freedom as the input triangulation, but the topology of the mesh and the distribution of degrees of freedom can be quite different.
3.4.1 Procedure Refine

Our $hp$ refinement procedure is summarized in Figure 3.2. We initialize a heap data structure where all elements are placed in the heap according to the size of $\eta_t$, with the element with largest error indicator at the root.

**Procedure Refine**

- **R1** Create a heap with respect to $\eta_t$ with the largest error estimate $\eta_{t_{\text{max}}}$ at the root;
- **R2** If $NDF \approx NDTRGT_0$, then go to **R6**.
  If $\eta_{t_{\text{max}}}^2 \leq \eta_{\text{ave}}^2 / 3$, then go to **R6**.
- **R3** Execute case specific tests for $h$, $p$, and $hp$ refinement of element $t_{\text{max}}$.
- **R4** Refine element $t_{\text{max}}$, and possibly others as required.
- **R5** Update error indicators for affected elements.
  Add new elements as needed. Remake the heap.
  Go to **R2**.
- **R6** Smooth the mesh based on geometry ((3.1)–(3.2)).
  Clean up data structures as needed.

**Figure 3.2.**

When $IADAPT = \pm 1$ and $NDTRGT > NDF$, the target number of degrees of freedom for the new mesh, denoted by $NDTRGT_0$, is given by

$$NDTRGT_0 = \min(NDTRGT, NDF \times 4).$$  \hspace{1cm} (3.5)

The use of (3.5) tries to force a geometric increase in the number of degrees of freedom in each refinement step. For the case $IADAPT = \pm 2$, $NDTRGT_0 = NDF_0$, where $NDF_0$ was the value of $NDF$ when TRIGEN was entered.

While we normally expect the refinement loop to exit when the target number of degrees of freedom is approximately achieved in line **R2** of Procedure Refine, we can also exit if the largest error in the current mesh is sufficiently small. In particular,

$$\eta_{\text{ave}}^2 = \frac{1}{N} \sum_{t \in \Omega_I} \eta_t^2 \equiv E\text{AVE2},$$  \hspace{1cm} (3.6)

where $\Omega_I$ is the fine subregion associated with processor $I$ in the case of parallel computation, and $\Omega_I \equiv \Omega$ otherwise; $N$ is the number of triangles in $\Omega_I$.

For $hp$-refinement, the critical test is to decide between between $h$-refinement and $p$-refinement for element $t_{\text{max}}$. The main test is to use $h$-refinement if

$$E(t_{\text{max}}, 2) \equiv \alpha_{t_{\text{max}}} \geq 2.$$  \hspace{1cm} (3.7)

If the scaling factor $\alpha_t \approx 1$, then the recovered derivatives and the error estimate are consistent, and we assume that the solution is locally smooth, which in turn justifies $p$-refinement. Large values of $\alpha_t$ empirically correspond to locally non-smooth behavior of the solution, and this in turn suggests $h$-refinement. See [25, 12]
3.4. Adaptive Mesh Refinement and Unrefinement.

for more detailed explanations and numerical experiments, and Bank, Parsania and Sauter [28] for some convergence analysis.

While (3.7) is the main test for \(hp\)-refinement, we also make some case specific tests on line \(R3\) of Procedure Refine. In the case of \(h\)-refinement, we test for potential round-off error problems if \(h\)-refined elements become too small in size. In the case of \(p\)-refinement, we check to be sure the \(p\)-refined element will have degree less than \(MXORD\). We require that \(MXORD \leq 9\) due to limits on the order of accuracy for the suite of numerical quadrature rules implemented in PLTMG. These quadrature formulas were provided by Zhang, Cui, and Liu in [67]. Failure of these case specific tests could reverse the decision suggested by (3.7). Finally, if element \(t_{max}\) has degree \(k\), and

\[
RELERP \leq \frac{1}{5 \times 3^{k-1}}
\]  

(3.8)

then \(h\)-refinement is selected. The thresholds (3.8) are based on the empirical observation that it is efficient to require sufficiently many elements in the mesh (as measured by (3.8)) before allowing increasingly higher degree elements.

At the conclusion of the main refinement loop, in \(R6\) we smooth the mesh, locally “flipping” edges and adjusting the location of vertices to locally optimize the geometric quality measure \(q(t)\) given in (3.1).\(^5\)

3.4.2 Procedure Unrefine

Our unrefinement procedure is complementary to the refinement procedure, as summarized Figure 3.3. In many details, it implements the opposite rules of Procedure Refine. For example, we initialize a heap data structure where all elements are placed in the heap according to the size of \(\eta_t\), but now with the element \(t_{min}\) with the smallest error error indicator at the root.

If \(IADAPT = \pm 1\) or \(IADAPT = \pm 2\), then \(NDTRGT_0\) is given by

\[
NDTRGT_0 = \max(NDTRGT, NDF/4).
\]

This generally provides a geometric decrease in the number of degrees of freedom. The parameter \(\eta_{ave}\) in \(U2\) is computed as in (3.7).

For \(hp\)-unrefinement, the main test is based on (3.7). If (3.7) and

\[
RELERP \geq \frac{1}{5}
\]

are both satisfied, then use \(h\)-unrefinement; otherwise use \(p\)-unrefinement. Our bias here is to try to preserve the largest number of elements in the mesh. The result of this test is possibly changed on the basis of the \(h\) and \(p\) specific tests mentioned below. The case specific test for \(h\)-unrefinement determines the best vertex of

\(^5\)In adjusting the mesh, we take into account constraints imposed by boundary geometry, boundary conditions, and internal interfaces defined by the the user or by PLTMG in the context of parallel computation.
Procedure Unrefine

U1  Create a heap with respect to \( \eta \) with the smallest error estimate \( \eta_{min} \) at the root;

U2  If \( NDF \approx NDTRG_{0} \), then go to U6.
    If \( \eta_{min}^{2} \geq \eta_{ave}^{2}/2 \), then go to U6.

U3  Execute case specific tests for \( h \) or \( p \)
    unrefinement of element \( t_{min} \).

U4  Unrefine element \( t_{min} \), and possibly others as required.

U5  Update error indicators for affected elements.
    Remove elements as needed. Remake the heap.
    Go to U2.

U6  Smooth the mesh based on geometry ((3.1)–(3.2)).
    Clean up data structures.

Figure 3.3.

element \( t_{min} \) to be removed from the mesh.\(^6\) For \( p \)-unrefinement, an element must have degree at least two.

Similar to Procedure Refine, at the conclusion of Procedure Unrefine, the final mesh is smoothed, and some edges possibly flipped in U6 to locally optimize the geometric quality as measured by (3.1).

3.4.3  \( h \) Refinement

Our basic \( h \)-refinement algorithm uses a relaxed version of the longest edge bisection procedure of Rivara \([52, 61]\) but does not generate a refined element tree. The element \( t_{max} \) to be refined is bisected along its longest edge. If \( t_{max} \) has a neighbor element across its longest edge, and the shared edge is longer than 0.9 times its longest edge, then it is refined. If not, the neighbor is refined along its longest edge, and the procedure described above is recursively applied to its longest-edge neighbor. An example is shown in Figure 3.4. The classic (unrelaxed) longest edge bisection process is known to have finite termination, typically in a very small number of steps. The relaxation factor 0.9 attempts to make the process terminate even sooner; it is small enough that the test is satisfied by most shape regular elements, so that only one edge is bisected in most steps of R4 in Procedure Refine (Figure 3.2).\(^7\)

When our relaxed longest edge bisection process finally results in a triangulation, elements are bisected, new elements created, (in reverse order to always maintain a triangulation) and the triangulation data structures updated. New elements inherit the (constant) derivative values from their parents, allowing error estimates to be computed for the refined elements, and the heap to be updated.

\(^6\)Certain vertices lying on the boundary or on internal interfaces are not eligible to be removed from the mesh. In exceptional cases an element might be deemed ineligible for \( h \)-unrefinement.

\(^7\)Since shape regularity is improved in step R6 of Procedure Refine, we relax shape regularity requirements during the refinement process itself to improve efficiency.
3.4. Adaptive Mesh Refinement and Unrefinement

Figure 3.4. Element $t$ is refined by the longest edge bisection method. The original mesh is on the left. The first step of bisection (middle) does not yield a compatible triangulation. However, the second step (right) does yield a triangulation.

3.4.4 $h$ Unrefinement

In the case of $h$-unrefinement, the basic step consists of deleting vertices from the mesh; this is accomplished by merging two vertices of element $t_{min}$ that share a common edge. This is illustrated in Figure 3.5.

Figure 3.5. Element $t$ is unrefined by the merging vertices $v$ and $v'$, and collapsing the edge connecting them. The location of the merged vertex could be $v$, $v'$, or $(v + v')/2$, depending on the shape quality of other elements having $v$ or $v'$ as one of their vertices.

While this merging process tends to degrade the geometric quality of surrounding elements, it is typically restored in step U6 of Procedure Unrefine, when element edges are flipped and vertices moved to locally optimize the quality measure (3.1).
3.4.5 \( p \) Refinement

The \( p \)-refinement algorithm is relatively straightforward. Let element \( t_{\text{max}} \) have degree \( p \) with edges of degree \( p_i \geq p, 1 \leq i \leq 3 \). The refined element has degree \( p + 1 \) with edges of degree \( \max\{p_i, p + 1\}, 1 \leq i \leq 3 \). As a technical point, since we require degrees of freedom associated with element interiors and edges have consecutive global indices,\(^8\) storage arrays (e.g., \( GF \)) have degrees of freedom for \( p \)-refined edges and interiors appended to their tails, leaving gaps where earlier edge and interiors degrees of freedom were stored. There is a global cleanup step at the end of the refinement process to compress the data structure and remove these gaps, similar to that performed at the end of unrefinement algorithms.

3.4.6 \( p \) Unrefinement

\( p \)-unrefinement is similar in structure to \( p \)-refinement. An element of degree \( p \) has its interior decreased to degree \( p - 1 \). The edge degrees of freedom \( p_i, 1 \leq i \leq 3 \), are reduced to \( p_i - 1 \) or remain at degree \( p_i \), depending on the degree of the neighbor element (if present) that shares the given edge. In the case of degree reduction, the reduced degrees of freedom can occupy space previously used by the higher degree edges or interior degrees of freedom, leaving small gaps in these data structures that are removed at the end of the unrefinement process.

3.5 Adaptive Mesh Smoothing.

When \( IADAPT = \pm 3 \), subroutine \( \text{TRIGEN} \) does no refinement or unrefinement of the mesh but rather adjusts the \((x, y)\) coordinates of the mesh points (\( VX \) and \( VY \)) in an attempt to optimize the mesh.

The procedure consists of a Gauss–Seidel-like iteration on the vertices in the mesh, where each vertex is locally optimized with all other vertices held fixed \([31]\). Four sweeps are performed in each call. Let \( \Omega_v \) denote the patch of elements that share a given vertex \( v \); an example is shown in Figure 3.6. Typically, vertex \( v \) is allowed to move within the region \( \Omega_v \). However, not all vertices in the mesh are allowed to move. Some boundary and interface vertices must remain fixed to preserve the definition of the region. These vertices are called corners. Corners include actual geometric corners of the region, vertices where boundary conditions change type or label, vertices where interfaces intersect the boundary, and vertices where two or more interfaces intersect. An interface here is taken as any sequence of triangle edges that separate triangles with different user defined labels. Vertices on the boundary or on interfaces that are not designated corners are allowed to move only along the boundary or interface. The remaining vertices, called interior vertices, are allowed to move freely within \( \Omega_v \). As in our refinement algorithms, some local mesh smoothing based on (3.1) is used to locally optimize the shape regularity of the mesh.

\(^8\)Having consecutive indices for interior and edge degrees of freedom makes the amount of information stored in array \( \text{ITDOF} \) for a given element independent of its degree.
3.6 Uniform Refinement.

For each vertex \( v = (x, y) \) in the mesh, we solve the minimization problem

\[
\min_{x, y} \| \nabla \epsilon_t \|_{L^2(\Omega_v)}^2
\]

of order two by a damped Newton’s method. As noted above, we assume the derivatives of order \( p+1 \) are constant in each element \( t \) having \( v \) as a vertex, leading to an overall piecewise constant approximation of these derivatives on \( \Omega_v \). All other dependencies on \( v = (x, y) \) are taken into account by Newton’s method. Boundary and interface vertices have an additional constraint equation, so an appropriately constrained version of problem (3.9) is solved for those vertices. Besides its usual task of ensuring sufficient decrease, the damping strategy for Newton’s method is also used to ensure that the point \( (x, y) \) remains well within \( \Omega_v \), so that all triangles are always well defined. It is interesting to note that the function \( \| \nabla \epsilon_t \|_{L^2(\Omega_v)} \) contains a natural barrier function that becomes infinite as \( (x, y) \) approaches \( \partial \Omega_v \).

In the case \( IADAPT = -3 \), the adaptive smoothing procedure uses the interpolation errors for the function \( QXY \) in place of the a posteriori error estimates, in a fashion analogous to the cases of refinement and unrefinement with \( IADAPT < 0 \).

### 3.6 Uniform Refinement.

\textit{TRIGEN} allows options for uniform refinement in \( h \) and \( p \). In the standard sequential setting, the uniform refinement is standard and straightforward. In the case of parallel computation, each processor approximately uniformly refines its own subregion. If the parallel computation uses \( NPROC \) processors, each processor adds approximately \( 1/NPROC \) of the expected increase in degrees of freedom resulting from uniform refinement in the standard sequential case. Because different subregions can have widely varying numbers of elements, this can result in different levels of uniform refinement in different regions. See Section 3.8 for discussion of the mesh partitioning process.
3.6.1  *h* Uniform Refinement

When \( IADAPT = 4 \), subroutine \( TRIGEN \) performs uniform \( h \)-refinement of the existing triangulation. The refinement is controlled by the parameter \( IREFN > 1 \). Each element in the triangulation is uniformly divided into \( IREFN^2 \) similar triangles. Some examples are shown in Figure 3.7. If \( IADAPT = 4 \) and \( MPISW = 1 \), the situation is different. In this case, \( TRIGEN \) tries to increase the global dimension of the space by roughly a factor of \( IREFN^2 \). The target value for each processor is given by

\[
NDTRGT_0 = \min \left( NDTRGT, NDF \times \left\{ 1 + \frac{IREFN^2 - 1}{NPROC} \right\} \right).
\]

Each processor refines uniformly, restricting this refinement mainly to region \( IRGN \). Rather than the uniform refinement described above, the relaxed longest edge procedure is used. Regions with relatively few elements may require several levels of refinement to achieve the target number of degrees of freedom; those with many elements may need only one partial level of refinement to achieve this target. In the case of partial levels of refinement, the elements chosen to be refined are essentially random.

\[ \text{Figure 3.7. Uniform refinement for the cases } IREFN = 2, 3, 4. \]

3.6.2  \( p \) Uniform Refinement

When \( IADAPT = -4 \), subroutine \( TRIGEN \) performs uniform \( p \)-refinement of the existing triangulation. The refinement is controlled by the parameter \( IREFN > 0 \). In this case, every element interior of degree \( p \) is increased to degree \( \min(p + IREFN, MXORD) \), and every edge of degree \( p_i \) is increased to degree \( \min(p_i + IREFN, MXORD) \). If \( IADAPT = -4 \) and \( MPISW = 1 \), \( TRIGEN \) does uniform \( p \) refinement, but restricts this refinement mainly to region \( IRGN \). In this case, the target number of degrees of freedom is given by

\[
NDTRGT_0 = \min \left( NDTRGT, NDF \times \left\{ 1 + \frac{[p_{ave} + IREFN]/p_{ave}]^2 - 1}{NPROC} \right\} \right),
\]
3.7. An Example

where \( p_{\text{ave}} \) estimates the average degree of elements in the mesh and is given by

\[
p_{\text{ave}} = \sqrt{\frac{\text{NDF}}{\text{NVF}}}.
\]

Similar to the case of \( IADAPT = 4 \), degrees of some elements may be increased by more than \( IREFN \) if few elements are present in \( IRGN \). If there are many elements in \( IRGN \) some elements may have their degrees increased by less than \( IREFN \), or even left unchanged. The elements chosen for partial levels of \( p \) refinement are essentially random, as in the case of \( h \) uniform refinement.

3.7 An Example

Some examples of our adaptive procedures are shown in Figures 3.8–3.9. In these examples, we employ the alternate function \( QXY = r^{1/4} \sin(\theta/4) \) defined on the circular domain with a crack shown in Figure 2.1. The initial mesh with \( NVF = 10 \) is shown in Figure 3.8, top. The mesh was then refined with \( IADAPT = -1 \) to produce a sequence of \( hp \) adapted meshes. In Figure 3.8, we see the resulting mesh with \( NTF = 9824 \) elements and \( NDF = 43319 \) degrees of freedom.

In the next experiment, we begin with the same mesh with \( NTF = 8 \) elements and uniformly refine it using \( IADAPT = 4 \) and \( IREFN = 8 \), creating a uniform mesh of \( NTF = 512 \) elements. We the uniformly refine in \( p \) (\( IADAPT = -4 \) and \( IREFN = 2 \)) to create a uniform mesh of piecewise cubic polynomials with a total of \( NDF = 2425 \) degrees of freedom.

We employ adaptive mesh smoothing (\( IADAPT = -3 \)); in Figure 3.9 we see the results of one step and eight steps of mesh smoothing. It is visually apparent that the mesh smoothing tries to concentrate degrees of freedom near the singularity at the origin. However, mesh smoothing is limited by the constraint that the mesh topology remain invariant, and this ultimately limits its effectiveness in this setting.

Next, we call \( TRIGEN \) int \( IADAPT = -2 \) (unrefine and refine) and \( NVTRGT = 1800 \). The results are shown in Figure 3.9, where the mesh is colored by both polynomial degree and by element size. We see that in the unrefinement phase, most of the coarsening consisted mostly of \( p \)-unrefinement, reducing the polynomial degree from three to two in those elements most distant from the origin, when the error should be smallest. The refinement step was mostly \( h \)-refinement, where elements near the origin were refined. This seems like a reasonable outcome given that the unrefinement and refinement steps are conducted sequentially and independently. However, one could argue that a more informed strategy in this situation is to coarsen in \( p \) the elements near the origin, which have larger errors, and then \( h \)-refine those elements using lower degree polynomials. Such a strategy requires more specialized coarsening and refinement algorithms than currently exist in \( PLTMG \) and is a topic for future research.

3.8 Parallel Adaptive Methods.

In this section we summarize the general strategy for adaptive mesh generation that is implemented in \( PLTMG \). A number of static and dynamic load balancing
The initial triangulation with $NTF = 8$, $NVF = 10$, and $NBF = 10$.

The refined triangulation with $NTF = 9824$, $NDF = 43319$. Elements are colored by polynomial degree (left) and by element size (right).

The refined triangulation with $NTF = 9824$, $NDF = 43319$. The mesh is magnified near the origin with $RMAG = 10^{16}$, revealing the presence of low degree polynomials (left) and very small elements (right).

Figure 3.8.

approaches for unstructured meshes have been proposed in the literature [65, 62, 45, 48, 42, 50, 43]; most of the dynamic strategies involve repeated application of a particular static strategy. One of the difficulties in all of these approaches is the amount of communication that must be performed both to assess the current load imbalance severity, and to redistribute the work among the processors once the imbalance is detected and an improved distribution is calculated.

The approach used by PLTMG is based upon the Bank-Holst paradigm [15, 16, 4, 26, 58, 53, 54], that addresses the load balancing problem in a new way, requiring far less communication. Another important point is that our approach

A uniformly refined mesh with \( NTF = 512 \), \( NVF = 297 \), and \( NBF = 80 \).

The uniformly refined mesh after one step of mesh smoothing (IADAPT = \(-3\)), with piecewise cubic elements, \( NDF = 2425 \) (left). The same mesh after eight steps of mesh smoothing (right).

The mesh resulting from one step of unrefinement and refinement (IADAPT = \(-2\)) with NDTRGT = 1800. The resulting mesh is colored by polynomial degree (left) and element size (right) and has \( NTF = 647 \) elements with \( NDF = 2422 \) degrees of freedom.

Figure 3.9.

allows PLTMG to run in a parallel environment without a large investment in additional coding. This approach has three main components:

Step 1: A small (nonlinear) problem is solved on an initial coarse mesh, and a posteriori error estimates are computed for the coarse grid solution. The triangulation is partitioned such that each subdomain has approximately equal error (although they can significantly differ in size, numbers of elements and degrees of freedom).
Step 2: Each processor is provided the complete coarse mesh and solution, and instructed to solve the entire (nonlinear) problem, with the stipulation that its adaptive refinement should be limited largely to its own partition. Load balancing is achieved by instructing each processor to create a refined mesh with the same number of degrees of freedom.

Step 3: A final mesh is computed using the union of the refined partitions provided by each processor. This mesh is reconciled such that the (virtual) mesh made up of the union of the refined subregions would be conforming. A final solution is computed, using a domain decomposition method. An initial guess is provided by the local solutions.

The above approach has several interesting features. First, the load balancing problem (Step 1) is reduced to the numerical solution of a small problem on a single processor, without requiring any modifications to PLTMG. Second, the adaptive mesh generation calculation (Step 2) takes place independently on each processor, and can also be performed with no communication.

The only parts of the calculation requiring communication are

1. the initial fan-out of the mesh distribution to the processors, once the decomposition is determined by the error estimator.

2. the mesh regularization, requiring communication to produce a global conforming mesh.

3. the final solution phase. Note that a good initial guess for Step 3 is provided in Step 2 by taking the solution from each subregion restricted to its partition.

The options $6 \leq IADAPT \leq 7$ provide basic parallel mesh management tools that support this paradigm. The domain decomposition solver is implemented as an option in subroutine PLTMG. These options require the use of MPI library routines for communication.

The basic strategy employed in Step 2 is an adaptive feedback loop

solve $\rightarrow$ estimate $\rightarrow$ refine.

The solve phase of this loop is realized by subroutine PLTMG while the estimate and refine phases are realized by subroutine TRIGEN. A potential problem arises from the possibility that with $hp$ refinement, it is highly likely that different processors will require different numbers of feedback loops in order to achieve a given target number of degrees of freedom. See [12, 41] for some discussion of this topic. Even in the simple case of $h$ refinement, it might be difficult to estimate the number of feedback loops that might be needed to achieve the target number of degrees of freedom. The parameter NOCHNG addresses this issue by providing a communication link between subroutines PLTMG and TRIGEN as summarized in Table 3.4.

A normal call to TRIGEN will return with NOCHNG = 0. However, if TRIGEN determines that the target number of degrees of freedom has already been achieved, it sets NOCHNG = 1 and returns with no further computation.

<table>
<thead>
<tr>
<th>NOCHNG</th>
<th>refinement option</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>normal call to TRIGEN</td>
</tr>
<tr>
<td>1</td>
<td>target degrees of freedom achieved</td>
</tr>
</tbody>
</table>

Table 3.4. The parameter NOCHNG.

If PLTMG is called with NOCHNG = 0, it will proceed normally. If PLTMG is called with NOCHNG = 1, it will return without further computation, since the subspace was unchanged by the previous call to TRIGEN. This allows the user to provide an overestimate the number of feedback loops required without incurring the penalty of additional computational costs. In this scenario, all processors can achieve NDTRGT degrees of freedom, although each may require a different number of feedback loops to do so.

3.8.1 Mesh Partitioning.

When IADAPT = 6, TRIGEN computes a posteriori error estimates and partitions the mesh as in the Bank-Holst paradigm. If PLTMG is running on NPROC processors, then the mesh is partitioned into NPROC subregions, such that each subregion has approximately equal error. Deotte [46] and Bank and Deotte [13] examine several partitioning strategies, including the one used in PLTMG, and in particular study their effect on the convergence rate of Domain Decomposition solvers using such partitions. The underlying algorithm employed is a variant of the recursive spectral bisection algorithm [44, 60, 63]. While this particular mesh partitioning algorithm is among the more expensive of the choices that we could make, it is typically used only once on a relatively small problem. Although this calculation is important in the parallel processing environment, it is done on a single processor and does not use the MPI library. At the conclusion of the load balancing step, TRIGEN creates new internal edges in IBNDRY at the interface between different subregions. Then the processor corresponding to IRGN = 1 broadcasts its mesh, solution, and supporting data to all processors using an MPI broadcast command.

The partitioning process begins with the creation of patches of elements with small errors called macro-elements. Macro-element patches contain a minimum of one and a maximum of 100 elements and must form a geometrically connected set. Let

\[ E = \frac{1}{NPROC} \sum_t \| \nabla \epsilon_t \|_{L^2(t)}^2. \]

For a patch \( P \), let

\[ E_P = \sum_{t \in P} \| \nabla \epsilon_t \|_{L^2(t)}^2. \]

If the patch \( P \) contains more than one element, we require \( E_P \leq 10^{-2} \times E \).

Suppose the mesh is composed of \( N \) macro-elements. We define the \( N \times N \)
symmetric, positive semi-definite \( M \)-matrix \( A \) by

\[
A_{ij} = \begin{cases} 
-\ell & i \neq j \text{ and patches } i \text{ and } j \text{ share } \ell \text{ common edges} \\
0 & i \neq j \text{ and patches } i \text{ and } j \text{ share no common edge} \\
s_i & i = j, s_i = -\sum_{k \neq i} A_{ik}
\end{cases}
\]

Macro element patches are created to reduce the order of the matrix \( A \), and thus reduce the cost of solving the eigenvalue problems described below. The matrix \( A \) corresponds to the discrete Laplacian for the dual graph of the macro element mesh, in which the macro elements are considered nodes, and the off-diagonal entries correspond to edges defined by the adjacency relation, weighted by the number of overlapping edges in the original triangulation.

We consider the eigenvalue problem

\[
A\psi = \lambda\psi
\]  

(3.10)

Our approach is standard; by construction, the smallest eigenvalue for (3.10) is \( \lambda_1 = 0 \) and \( \psi_1 = (1, 1, \ldots, 1)^t \). Our interest is in the second eigenvector \( \psi_2 \), known as the Fiedler vector.

We use a standard binary tree with \( 2^{NPROC} - 1 \) nodes (\( NPROC \) leaves and \( NPROC - 1 \) internal nodes). The root is labeled \( i = 1 \) and node \( i \) has children \( 2i \) and \( 2i + 1 \), \( 1 \leq i \leq NPROC - 1 \). Associated with each node is a weight \( \omega_i \), denoting the number of leaves contained in its subtree. In particular, \( \omega_1 = 1 \), \( i = 2 \) \( NPROC - 1, \ldots, NPROC \) and \( \omega_i = \omega_{2i} + \omega_{2i+1} \) for \( i = NPROC - 1, \ldots, 1 \).

The entire mesh is assigned to root, and it is partitioned among its two children as follows. We first approximately solve the eigenvalue problem (3.10) for the whole mesh, and then create a permutation of the macro-elements \( \{q_i\} \) such that

\[ q_i < q_j \implies \psi_{2,i} \leq \psi_{2,j}. \]

We then find the index \( k \) which provides the best partition of the form

\[
\frac{1}{\omega_2} \sum_{q_i \leq k} E_{P_{q_i}} \approx \frac{1}{\omega_3} \sum_{q_i > k} E_{P_{q_i}}.
\]

The corresponding submeshes are assigned to the children nodes.

We apply this procedure recursively, at each level dividing each group of element patches into two smaller groups by solving an eigenvalue problem of the type (3.10) restricted to that group of patches. The final result is \( NPROC \) subregions with approximately equal error \( E \).

We now briefly describe some details of our procedure for computing the second eigenvector of (3.10). Our procedure is essentially just a classical Rayleigh quotient iteration \([59]\), modified both to bias convergence to \( \lambda_2 \), and to account for the fact that the linear systems arising in the inverse iteration substep are solved (approximately) by an iterative process. To simplify notation and avoid multiple subscripts, we let \( \phi_k \approx \psi_2 \), where \( k \) now denotes the iteration index.

We suppose that we have a current iterate \( \phi_k \) which satisfies \( \phi_k^t \phi_k = 1 \) and \( \psi_1^t \phi_k = 0 \). Using \( \phi_k \), we compute the approximate eigenvalue \( \tilde{\lambda}_{2,k} \approx \lambda_2 \) from the

Rayleigh quotient $\tilde{\lambda}_{2,k} = \phi_k^t A \phi_k$, and approximately solve the linear system

$$A \tilde{\delta}_k = r_k \equiv \tilde{\lambda}_{2,k} \phi_k - A \phi_k.$$ 

Note that by construction $\psi_1^t r_k = \phi_k^t r_k = 0$. This linear system is approximately solved using a simple Symmetric Gauss-Seidel iteration.

From $\tilde{\delta}_k$, we form the vector $\delta_k$ satisfying $\delta_k^t \delta_k = 1$ and $\psi_1^t \delta_k = \phi_k^t \delta_k = 0$. Finally, we solve the $3 \times 3$ eigenvalue problem for $\hat{A}$, where

$$\hat{A} = \begin{pmatrix} \phi_k^t \\ \delta_k^t \\ \xi_k^t \end{pmatrix} A \begin{pmatrix} \phi_k & \delta_k & \xi_k \end{pmatrix}$$

where $\xi_k$ is defined below. If $v = (\alpha, \beta, \gamma)^t$ is an eigenvector corresponding to the smallest nonzero eigenvalue, we form $\tilde{\phi}_{k+1} = \alpha \phi_k + \beta \delta_k + \gamma \xi_k$ and $\tilde{\xi}_{k+1} = \beta \delta_k + \gamma \xi_k$ with $\xi_1 = 0$. Then $\phi_{k+1}$ and $\xi_{k+1}$ are formed from $\tilde{\phi}_{k+1}$ and $\tilde{\xi}_{k+1}$, respectively, by normalization and orthogonalization to $\psi_1$. Solving the $3 \times 3$ eigenvalue problem rather than a $2 \times 2$ problem was motivated by the work of Knyazev [49].

3.8.2 Reconciling the Mesh.

The option $\text{IADAPT} = 7$ reconciles the mesh. This is the most complex of the MPI options in TRIGEN, and is typically called once, at the conclusion of the second step of the Bank-Holst paradigm. It must be called before the domain decomposition solution in subroutine PLTMG, as PLTMG makes use of the parallel interface data structure IPATH generated by this call.

In creating the IPATH data structure, each processor first organizes its triangulation and solution data structures. Generally, edges, vertices, and degrees of freedom on the interface between region IRGN and the rest of the domain appear first in their respective arrays ($\text{IBNDRY}$, VX, VY, GF, etc). This data is organized to correspond to counter clockwise traversal of the interface. Next in all arrays comes data corresponding to the interior of subregion IRGN; generally, this should be the majority of the data. Finally, at the end of each array appears data corresponding to regions other than IRGN. Each processor then assembles its contributions to the preliminary IPATH array based on the reordered data, and this information is then exchanged among processors using MPI (see Section 2.6).

After the boundary exchange, each processor tries to match its boundary interface edges to those provided by neighboring regions, in order to establish the structure of the global mesh. Typically this mesh is not conforming. When non-matching edges are found, the region that is less refined does additional refinement until its boundary edges form a one-to-one match with those of its neighbors. An example is shown in Figure 3.10.

After the mesh is made $h$-conforming it is made $p$-conforming. When a $p$-nonconforming edge is found, the region containing the edge of lower degree $p$-refines its edge appropriately in order to resolve the nonconformity.

Each processor then reorders its data structures and communicates its contribution to the IPATH array a second time. This time the edge matching process
concludes with no nonconforming edges found. By matching boundary edges at the interface, one also effectively matches degrees of freedom on the interface; it is this information that is needed for the domain decomposition solver.
Chapter 4

Equation Solution

4.1 Overview.

Subroutine \texttt{PLTMG} solves the problems described in Section 1.1. The solution process for each class of problems has certain unique aspects, but all make use of Newton’s method. Subroutine \texttt{PLTMG} is entered using the statement

\begin{verbatim}
Call PLTMG( VX, VY, SF, ITNODE, IBNDRY, ITDOF, IPATH,
E, IP, RP, SP, GF, A1XY, A2XY, FXY, GNXY, GDXY,
P1XY, P2XY, SXY )
\end{verbatim}

On input, the arrays \texttt{VX}, \texttt{VY}, \texttt{SF}, \texttt{ITNODE}, \texttt{ITDOF}, and \texttt{IBNDRY} define a triangulation. Arrays \texttt{IPATH} \texttt{E}, and \texttt{GF} are discussed in Sections 2.5 and 2.6. Fortran subroutines \texttt{A1XY}, \texttt{A2XY}, \texttt{FXY}, \texttt{GNXY}, \texttt{GDXY}, \texttt{P1XY}, and \texttt{P2XY} are documented in Section 2.8. Subroutine \texttt{SXY} is documented in Section 2.2. Parameters in the \texttt{IP}, \texttt{RP}, and \texttt{SP} arrays read and written by \texttt{PLTMG} are summarized in Tables 2.12–2.14.

The parameter \texttt{IPROB} indicates the problem class; the various options are shown in Table 4.1. The case \texttt{IPROB} > 0 indicates a standard sequential solve, either on a single processor, or on multiple processors as part of the second phase of the Bank-Holst paradigm. The case \texttt{IPROB} < 0 indicates the global parallel domain decomposition solve as part of the Bank-Holst paradigm. Because this is a global solve it involves some MPI communication at each iteration step. When \texttt{IPROB} < 0, the parallel domain decomposition solve is preceded by a local solve on each processor, in order to insure the quality of the initial guess for the global problem.

The cases \texttt{IPROB} = ±3 and \texttt{IPROB} = ±4 have various suboptions unique to their particular problem class. The available options are specified through the parameter \texttt{ITASK}. These are summarized in Table 4.2.
### Table 4.1. The parameter IPROB.

<table>
<thead>
<tr>
<th>IPROB</th>
<th>problem option</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>elliptic boundary value problem</td>
</tr>
<tr>
<td>2</td>
<td>obstacle problem</td>
</tr>
<tr>
<td>3</td>
<td>continuation problem</td>
</tr>
<tr>
<td>4</td>
<td>parameter identification problem</td>
</tr>
<tr>
<td>5</td>
<td>optimal control problem</td>
</tr>
<tr>
<td>-1</td>
<td>DD solve for elliptic boundary value problem</td>
</tr>
<tr>
<td>-2</td>
<td>DD solve for obstacle problem</td>
</tr>
<tr>
<td>-3</td>
<td>DD solve for continuation problem</td>
</tr>
<tr>
<td>-4</td>
<td>DD solve for parameter identification problem</td>
</tr>
<tr>
<td>-5</td>
<td>DD solve for optimal control problem</td>
</tr>
</tbody>
</table>

### Table 4.2. The parameter ITASK.

<table>
<thead>
<tr>
<th>ITASK</th>
<th>IPROB</th>
<th>option</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>default</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>use functional</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>default</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>continue to the nearest target point</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>continue to the nearest target or singular point</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>switch branches at a bifurcation point</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>switch $\lambda$ and/or $\rho$; initialize with $\lambda$ fixed</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>switch $\lambda$ and/or $\rho$; initialize with $\rho$ fixed</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>solve with $\sigma = 0$, $\theta = 0$ ($\lambda$ fixed)</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>solve with $\sigma = 0$, $\theta = 2$ ($\rho$ fixed)</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>solve with $\sigma = 0$, $\theta = 1$</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>default</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>$\lambda$ affects domain shape</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>default</td>
</tr>
</tbody>
</table>

### 4.2 Elliptic Boundary Value Problems.

When $IPROB = 1$, PLTMG solves the discrete system (1.6). If the underlying boundary values problem is not self-adjoint some upwinding terms inspired by the Scharfetter–Gummel discretization scheme [7, 11] are added to the discretization; in this case (1.6) should be replaced by: find $u_h \in \mathcal{M}_d$ such that

\[
a_h(u_h, v) = 0 \quad \text{for all } v \in \mathcal{M}_e, \tag{4.1}
\]
where $a_h(u_h, v)$ reflects the additional stabilization terms. We note that the upwinding terms are derived for the case of piecewise linear finite elements ($p = 1$). While a similar upwinding scheme is also formally applied to higher degree elements, its stability and convergence properties are not yet analyzed. In any event, (4.1) corresponds to the system of nonlinear equations

$$G(U) = 0,$$

(4.2)

where the unknown vector $U$ corresponds to the values of the finite element solution $u_h$ at the vertices of the triangulation. The Jacobian matrix

$$A(U) = \frac{\partial G(U)}{\partial U}$$

is a sparse stiffness matrix corresponding to a linear elliptic boundary value problem (linearized about $U$). Even in the event the original problem is linear, PLTMG solves all problems with IPROB = 1 as nonlinear, and formally applies Newton’s method to (4.2). In Figure 4.1, we summarize our approximate Newton procedure with line search.

**Procedure Newton**

1. **N1** Begin with initial guess $U_0$, and a sufficient decrease parameter $\tau$. Set $k \leftarrow 0$, $s_0 \leftarrow 1$, and compute $G_0$ and $\|G_0\|$.
2. **N2** solve (approximately) $A_k \delta U_k = -G(U_k)$.
3. **N3** compute $U_{k+1} = U_k + s_k \delta U_k$, $G_{k+1}$, $\|G_{k+1}\|$, and $\xi_{k+1} = \|G_{k+1}\|/\|G_k\|$.
4. **N4** if $1 - \xi_{k+1} \leq \tau s_k$, then decrease $s_k$ and go to **N3**; else set $s_{k+1} \leftarrow s_k / (s_k + (1 - s_k)\xi_{k+1}/100)$ and $k \leftarrow k + 1$.
5. **N5** if converged, then exit; else go to **N2**.

**Figure 4.1.**

The scalar $s_k$ is the damping parameter. When the sufficient decrease criterion is not satisfied on line **N4** and $s_k$ must be reduced, the next value is found through application of one step of a guarded secant/bisection algorithm to the one-dimensional minimization problem

$$\min_{s_k} \|G(U_k + s_k \delta U_k)\|.$$ 

If sufficient decrease is achieved, the current $s_k$ is used to predict $s_{k+1}$; this formula is designed to force rapid increase of $s_{k+1} \rightarrow 1$ when $\xi_{k+1}$ becomes small as superlinear convergence occurs, and at the same time provide a reasonable first guess in the early stages of the Newton iteration, when damping is most important. A maximum of MXNWTT damped Newton iterations are allowed. PLTMG reports the actual number of Newton iterations used on the most recent call in the parameter ITNUM,
and the number of evaluations of \( G \) as \( IEVALS; IEVALS \geq ITNUM \), since more than one function evaluation may be used in each line search.

As a simple example, we solve the Poisson equation

\[
\begin{align*}
-\Delta u &= 1 & \text{in } \Omega, \\
u &= 0 & \text{on } \partial \Omega,
\end{align*}
\]

The domain \( \Omega \) was provided as a skeleton and is shown in Figure 4.4. This problem was solved using \( hp \) adaptive refinement using eight processors. The skeleton was triangulated, and then a mesh with \( NVF = 1879 \) and \( NDF = 5358 \), was adaptively created on one processor. The processor then did a load balance step (\( IADAPT = 6 \) in \( TRIGEN \)) and broadcast this mesh to all processors. The load balance is shown in Figure 4.4. Each processor then independently continued the refinement process on its subregion, using a maximum of five \( hp \) adaptive refinement steps. The global refined mesh was made conforming (\( IADAPT = 7 \) in \( TRIGEN \)) and the domain decomposition solver invoked in PLTMG (\( IPROB = -1 \)). The resulting global mesh had \( NDF = 221657 \) degrees of freedom; the global solution is shown in Figure 4.4. The global mesh colored by element size and polynomial degree is shown in Figure 4.4, along with a timing summary for the entire calculation, and a convergence history for the Domain Decomposition solver (see Section 4.4).

### 4.3 Linear Solvers.

All sets of linear equations involving the matrices \( A(U) \) and \( A(U)^t \) have the appearance of finite element discretizations of linear elliptic boundary value problems. These systems can be solved using a variety of preconditioners, coupled with the composite step conjugate gradient method (\( ISPD = 1 \)) or composite step biconjugate gradient method (\( ISPD = 0 \)). The composite step algorithms \([10, 9]\) are similar to the standard biconjugate gradient and conjugate gradient methods, respectively, except that they occasionally proceed from the iterate for step \( k \) to the iterate for step \( k + 2 \). Such composite steps are taken to improve the stability of the recurrence relations and smooth the behavior of the residual norm. Note in particular that the composite step conjugate gradient method can be applied to symmetric but indefinite problems. The maximum number of iterations to be used per solution is specified by the parameter \( MXCG \). Note that as many as \( MXCG \) iterations are used each time a system of linear equations is solved.

The selection of preconditioner is governed by the parameter \( METHOD \) as summarized in Table 4.3. Combinations of three different matrices are used in composing these preconditioners. We remark that in all cases the matrices are ordered using a (block) minimum degree algorithm. In particular, dense blocks corresponding to element interior degrees of freedom are always ordered first, as these blocks could be eliminated through Gaussian Elimination (static condensation) without causing any fill-in.

\( ILU \) is an incomplete \( LDU \) factorization based on the multigraph algorithm \([33, 32]\). The parameter \( DTOL \) is the drop tolerance for this approximate factorization. Generally, smaller values of \( DTOL \) result in more accurate \( ILU \) factorizations,
4.3. Linear Solvers.

The skeleton with NVF = 135 (left) and the resulting triangulation with NVF = 1381 (right).

The mesh with NDF = 5358 showing the load balance (left), and the solution on the global fine mesh NDG = 221657 (right).

The global refined mesh colored by element size (left) and by polynomial degree (right).

Some timing statistics (left) and convergence history for the DD solver (right).

Figure 4.2.
Table 4.3. *The parameter METHOD.*

<table>
<thead>
<tr>
<th>METHOD</th>
<th>Preconditioner</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Block SGS smoother, HB coarse space</td>
</tr>
<tr>
<td>1</td>
<td>ILU</td>
</tr>
</tbody>
</table>

but higher costs in space and time per iteration. The extreme case $DTOL = 0$ results in a sparse direct factorization.

SGS is a block symmetric Gauss-Seidel preconditioner, based on the diagonal, upper and lower triangular blocks of the matrix $A$. The diagonal blocks of $A$ are factored using dense $LDU$ factorizations. HB is a coarse grid correction based on hierarchical basis. Let $\mathcal{M}$ denote the $hp$ finite element space used in assembling the matrix $A$. Our coarse space $\mathcal{M}_c \subset \mathcal{M}$ is composed as the union of the following sets of basis functions.

1. A standard continuous piecewise linear nodal basis function is associated with each vertex in the mesh.

2. For every edge in the triangulation with $p \geq 2$, we associate a continuous piecewise quadratic nodal basis function (quadratic bump function) corresponding to the edge midpoint.

3. For every element in the triangulation with $p \geq 3$, we associate a continuous piecewise cubic nodal basis function (cubic bubble function) corresponding to the element barycenter.

The maximum dimension of $\mathcal{M}_c$ is approximately $6 \times NVF$, if every quadratic bump and cubic bubble function is present. The coarse space matrix $A_c$ is computed from $A$ by

$$A_c = S^t AS$$

for an appropriately defined rectangular change of basis matrix $S$. Then we compute an approximate $LDU$ factorization of the matrix $A_c$. The parameter $DTOL$ is the drop tolerance for this approximate factorization. This approximate factorization serves as the coarse grid correction in a two-level preconditioner. One cycle of the two level iteration consists of one pre-smoothing step (SGS), followed by the coarse grid correction step, and one post-smoothing step (SGS). See [6] for additional discussion.

As a general remark, the $SGS/\text{HB}$ two-level solver typically requires less setup time but more solution time compared to $ILU$. Overall, $ILU$ becomes increasingly effective when there are many high degree elements and the matrix $A$ becomes increasingly dense. $ILU$ also is very effective on convection dominated highly nonsymmetric problems. The $SGS/\text{HB}$ two-level solver is most effective when there are many low degree elements in the mesh. Finally, note that a direct method is available using $ILU$ with $DTOL = 0$. 
4.4 Domain Decomposition Solver

Here we describe the domain decomposition algorithm implemented in PLTMG for Step 3 of the Bank-Holst paradigm (see Section 3.8). This algorithm is described in detail in [19, 3, 51, 35]. It is motivated by and similar to the domain decomposition algorithms described in [18, 17]. In the case $IPROB = -1$, this solver is used in place of the simple linear solver in line $N2$ of Procedure Newton given in Figure 4.1.

For simplicity in our discussion here, we restrict attention to the case of just two subdomains. In our scheme, each subregion contributes equations corresponding all fine degrees of freedom, including its interface. Thus in general there will be multiple unknowns and equations in the global system corresponding to the interface degrees of freedom. This is handled by equality constraints that impose continuity at all degrees of freedom on the interface. The result is a mortar-element like formulation, using Dirac $\delta$ functions for the mortar element space. In any event, with a proper ordering of unknowns, the global system of equations has the block $5 \times 5$ form

$$\begin{pmatrix}
A_{11} & A_{1\gamma} & I \\
A_{\gamma1} & A_{\gamma\gamma} & A_{\nu2} \\
A_{\nu\nu} & A_{\nu2} & -I \\
I & -I & \\
\end{pmatrix}
\begin{pmatrix}
\delta U_1 \\
\delta U_\gamma \\
\delta U_\nu \\
\delta U_2 \\
\Lambda \\
\end{pmatrix}
= 
\begin{pmatrix}
R_1 \\
R_\gamma \\
R_\nu \\
R_2 \\
U_\nu - U_\gamma \\
\end{pmatrix}.
$$

(4.3)

Here $A_{11}$ and $A_{22}$ correspond to the fine degrees of freedom on processors 1 and 2, respectively, that are not on the interface, while $A_{\gamma\gamma}$ and $A_{\nu\nu}$ correspond to interface points. The fifth block equation imposes continuity, and its corresponding Lagrange multiplier is $\Lambda$. The identity matrix appears because the global fine mesh is conforming. The introduction of the Lagrange multiplier and the saddle point formulation (4.3) are only for expository purposes; indeed, $\Lambda$ is never computed or updated.

On processor 1, we develop a similar but “local” saddle point formulation. That is, the fine mesh subregion on processor 1 is “mortared” to the remaining coarse mesh on processor 1. This leads to a linear system of the form

$$\begin{pmatrix}
A_{11} & A_{1\gamma} & I \\
A_{\gamma1} & A_{\gamma\gamma} & A_{\nu2} \\
\bar{A}_{\nu\nu} & \bar{A}_{\nu2} & -I \\
I & -I & \\
\end{pmatrix}
\begin{pmatrix}
\delta U_1 \\
\delta U_\gamma \\
\delta \bar{U}_\nu \\
\delta \bar{U}_2 \\
\Lambda \\
\end{pmatrix}
= 
\begin{pmatrix}
R_1 \\
R_\gamma \\
R_\nu \\
0 \\
U_\nu - U_\gamma \\
\end{pmatrix},
$$

(4.4)

where quantities with a bar (e.g., $\bar{A}_{22}$) refer to the coarse mesh. A system similar to (4.4) can be derived for processor 2. With respect to the right hand side of (4.4), the interior residual $R_1$ and the interface residual $R_\gamma$ are locally computed on processor 1. We obtain the boundary residual $R_\nu$, and boundary solution $U_\nu$ from processor 2; processor 2 in turn must be sent $R_\gamma$ and $U_\gamma$. The residual for the coarse grid interior points is set to zero. This avoids the need to obtain $R_2$ via communication, and to implement a procedure to restrict $R_2$ to the coarse mesh on
processor 1. Given our initial guess, we expect \( R_1 \approx 0 \) and \( R_2 \approx 0 \) at all iteration steps. \( R_\gamma \) and \( R_\nu \) are not generally small, but \( R_\gamma + R_\nu \to 0 \) at convergence.

As with the global formulation (4.3), equation (4.4) is introduced mainly for exposition. The goal of the calculation on processor 1 is to compute the updates \( \delta U_1 \) and \( \delta U_\gamma \), which contribute to the global conforming solution. To this end, we formally reorder (4.4) as

\[
\begin{pmatrix}
-I & -I \\
-\bar{A}_{\nu\nu} & I & \bar{A}_{\nu2} \\
I & A_{\gamma1} & A_{\gamma\gamma} \\
\bar{A}_{2\nu} & \bar{A}_{2\nu} & \bar{A}_{22}
\end{pmatrix}
\begin{pmatrix}
\Lambda \\
\delta \bar{U}_\nu \\
\delta U_1 \\
\delta U_\gamma \\
\delta \bar{U}_2
\end{pmatrix}
= \begin{pmatrix}
U_\nu - U_\gamma \\
R_\nu \\
R_1 \\
R_\gamma \\
0
\end{pmatrix}. \tag{4.5}
\]

Block elimination of the Lagrange multiplier \( \Lambda \) and \( \delta \bar{U}_\nu \) in (4.5) leads to the block 3 \times 3 Schur complement system

\[
\begin{pmatrix}
A_{11} & A_{1\gamma} \\
A_{\gamma1} & \gamma + \bar{A}_{\nu\nu} \\
\bar{A}_{2\nu} & \bar{A}_{22}
\end{pmatrix}
\begin{pmatrix}
\delta U_1 \\
\delta U_\gamma \\
\delta \bar{U}_2
\end{pmatrix}
= \begin{pmatrix}
R_1 \\
R_\gamma + \bar{A}_{\nu\nu}(U_\nu - U_\gamma) \\
\bar{A}_{2\nu}(U_\nu - U_\gamma)
\end{pmatrix}. \tag{4.6}
\]

The system matrix in (4.6) corresponds to the final adaptive refinement step on processor 1, with possible modifications due to global fine mesh regularization. It is exactly the matrix used in the preliminary local solve to generate the initial guess for the global domain decomposition iteration. In the solution of (4.6), the components \( \delta U_1 \) and \( \delta U_\gamma \) contribute to the global solution update, while \( \delta \bar{U}_2 \) is discarded. We remark that the global iteration matrix corresponding to this formulation is not symmetric, even if all local system matrices are symmetric.

The domain decomposition algorithm is incorporated as the solver for the approximate Newton iteration described in Figure 4.1. In particular, only one domain decomposition iteration (a so-called inner iteration) is used in each approximate Newton step. Thus, loosely speaking, each solve of (4.6) alternates with a line search step in which the global solution is updated. The Newton line search procedure requires global communication to form some norms and inner products, as well as the boundary exchange described above.

### 4.5 Obstacle Problems.

When \( \text{IPROB} = 2 \), PLTMG solves the obstacle problem (1.8). The inequality constraints are treated via an interior point procedure [14]. In particular, we consider the Lagrange function

\[
L(u_h) = \rho(u_h) - \mu \sum_{i=1}^{NDF} \left\{ \log(u_h(p_i) - u(p_i)) + \log(\pi(p_i) - u_h(p_i)) \right\} \tag{4.7}
\]

where \( \mu > 0 \) is a small barrier parameter; the user specifies the target value in \( \text{RMTRGT} \). Nodes corresponding to degrees of freedom are denoted by \( p_i = (x_i, y_i) \),
and $d_i$ is the diagonal entry of the mass matrix corresponding to $p_i$. The weights $d_i = O(h_i^2)$ scale the barrier terms in a fashion similar to $\rho(u_h)$, and make $\mu$ independent of the mesh.

The overall solution strategy is to compute stationary points of the Lagrange function (4.7) for a decreasing sequence of $RMTRGT = \mu > 0$ values, following a smooth trajectory moving towards the boundary of the feasible region. This shares some features with the more general path following procedures used in the case $IPROB = 3$.

The assembly and solution procedures are quite similar to the case $IPROB = 1$. In particular, the right hand side is modified by terms of the form

$$-\mu d_i \left\{ (u_h(p_i) - u(p_i))^{-1} + (u_h(p_i) - \bar{u}(p_i))^{-1} \right\},$$

and the diagonal of the stiffness matrix (Hessian matrix of the functional $\rho(u_h)$) is modified by terms of the form

$$\mu d_i \left\{ (u_h(p_i) - u(p_i))^{-2} + (u_h(p_i) - \bar{u}(p_i))^{-2} \right\}.$$

With these modifications, the approximate Newton strategy described in Section 4.2 is used here.

When $IPROB = -2$, the domain decomposition algorithm outlined in Section 4.4 is used, with appropriate modifications to the stiffness matrix and right hand sides. As in the case $IPROB = -1$, only one domain decomposition solve (inner iteration) is used in each approximate Newton iteration.

As an example, we use PLTMG to solve the variational inequality

$$\min_{u \in K} \int_{\Omega} \{ |\nabla u|^2 - 2f(x, y)u \} dx \, dy$$

where the domain $\Omega = (0, 1) \times (0, 1)$, and

$$K = \left\{ u \in H^1_0(\Omega) : |u| \leq \frac{1}{4} - \frac{1}{10} \sin(\pi x) \sin(\pi y) \right\},$$

$$f(x, y) = -\Delta(\sin(3\pi x) \sin(3\pi y)).$$

In the absence of the obstacle, this is a simple elliptic equation with exact solution $u = \sin(3\pi x) \sin(3\pi y)$.

In this example, we compare $h$-refinement using piecewise linear elements with $hp$-refinement. In both cases, we begin with linear elements on a uniform $9 \times 9$ mesh, as illustrated in Figure 4.5. In both cases, we went through eight adaptive loops, followed by solving the problem with $\mu = 1$. Then we made three additional solutions with $\mu = 10^{-k}$ for $1 \leq k \leq 3$.

In the case of piecewise linear elements, the final mesh had $NTF = 33105$ elements and $NDF = 17068$ degrees of freedom. The solution, the mesh colored by element size, and the a posterior error estimate for the final solution are shown in Figure 4.5. In the case of $hp$-refinement, the final mesh had $NTF = 7105$ elements and $NDF = 12258$ degrees of freedom. The solution, the mesh colored by element size and element degree, and the error estimate are shown in Figure 4.5.
We note that in the $hp$-adaptive case, the mesh is coarser and more uniform. In the $h$-refinement case, there was more refinement near the boundaries where the inequality constraint was satisfied as equality, and much less refinement in the interior of those regions. The $hp$ mesh used mostly quadratic and cubic elements, and was much more uniform. In the regions where the inequality constraint was satisfied as equality, one sees the effect of imposing the inequalities just at the nodes; one observes small oscillations in these regions due to the higher degree polynomials.

4.6 Continuation Problems.

In the case of continuation problems ($IPROB = 3$), the parameter $ITASK$ specifies the the continuation option. Available options are summarized in Table 4.2. For convenience in notation, we will systematically drop the subscript $h$ from all variables in this section (e.g., $\lambda_h$ will be denoted $\lambda$).

When the continuation process is used, we use a normalization equation of the form

$$N(u, \lambda) = \sigma.$$  

The scalar $\sigma = SIGMA$ is the steplength. PLTMG uses then the normalization equation described in [8, 56],

$$N(u, \lambda) = \theta \dot{\rho}_0 (\rho - \rho_0) + (2 - \theta) \dot{\lambda}_0 (\lambda - \lambda_0).$$  

(4.8)

Here $\theta = THETA$ is a parameter selected by PLTMG; by choosing $\theta$ and $\sigma$ properly, it is possible to achieve target values in either $\rho$ or $\lambda$. The vector $(u^t_0, \lambda_0)$ is the current solution point and $(\dot{u}^t_0, \dot{\lambda}_0)$ the current unit tangent vector. The scalar $\dot{\rho}$ is defined formally using the chain rule for differentiation:

$$\dot{\rho} = \rho_u \ddot{u} + \rho_\lambda \ddot{\lambda}.$$  

The values $0 \leq ITASK \leq 4$ embody the basic continuation path following options available in PLTMG. The values $5 \leq ITASK \leq 7$ are designed for updating the solution at a fixed point when the mesh has been changed by a call to TRIGEN.

An initial solution is provided by the user through subroutine GDXY. Thereafter, the continuation proceeds from the last successfully computed point. A brief outline of the basic continuation process ($ITASK = 0$ or $ITASK = 1$) is given in Figure 4.4.

PLTMG always returns with $(RLTRGT, RTRGT) = (RL, R) \equiv (\lambda, \rho)$. To continue with $ITASK = 0$ or $ITASK = 1$, the user specifies a target value for either $RTRGT$ or $RLTRGT$. If $RLTRGT \neq RL$, then PLTMG seeks a solution with $\lambda = RLTRGT$. Similarly, if $RTRGT \neq R$, then PLTMG seeks a solution with $\rho = RTRGT$.

A step $\sigma$ and a predicted solution are computed on line $C2$. The predictor is a standard Euler type commonly used in continuation procedures. The step size calculation is influenced not only by the user request but also by imposed requirements that the predicted solution be sufficiently accurate. The procedures used in this portion of the calculation are described in detail in [21]. The solution
4.6. Continuation Problems.

The initial uniform mesh (left), and the final mesh for the case of $h$-refinement, colored by element size (right).

The piecewise linear solution (left) and corresponding error estimate (right).

The final adaptive mesh for $hp$-refinement, colored by element size (left) and polynomial degree (right).

The $hp$-adaptive solution (left) and corresponding error estimate (right).

Figure 4.3.
Procedure Continue

C1 Begin with initial solution \((u^0, \lambda_0)\) and tangent vector \((\dot{u}^0, \dot{\lambda}_0)\).

C2 compute the step \(\sigma\) for the normalization equation; predict \((u^t, \lambda) \leftarrow (u^0, \lambda_0) + \alpha(\dot{u}^0, \dot{\lambda}_0)\).

C3 correct \((u^t, \lambda) \leftarrow \text{NWT}(u^t, \lambda)\); compute \(\psi_\ell, \psi_r\), and \(\nu\); compute tentative \(\dot{u}\) and \(\dot{\lambda}\).

C4 if a singular point was detected and ITASK = 1, then go to C7.

C5 set \((u^0, \lambda_0) \leftarrow (u^t, \lambda)\) and \((\dot{u}^0, \dot{\lambda}_0) \leftarrow (\dot{u}^t, \dot{\lambda})\).

C6 if \((u^0, \lambda_0)\) is a target point, then exit; else go to C2.

C7 compute the singular point using secant/bisection algorithm on \(\nu(\sigma) = 0\); exit.

Figure 4.4.

is corrected on line C3. The correction process symbolized by the operator NWT involves the solution of a set of nonlinear equations, and is discussed in greater detail below.

PLTMG locates singular points by computing the smallest singular value \(\nu\) of the Jacobian matrix. A modified inverse iteration procedure computes the left and right singular vectors \(\psi_\ell\) and \(\psi_r\) corresponding to \(\nu\) as part of each correction step C3. If the matrix is symmetric (ISPD = 1), then \(\psi_\ell \equiv \psi_r\). In a somewhat nonstandard fashion for singular values, we normalize the singular vectors to have unit length and satisfy

\[
\int_\Omega \psi_\ell \psi_r \, dx > 0.
\]

Requiring the sign of the inner product of \(\psi_\ell\) and \(\psi_r\) to be positive forces the singular value \(\nu\) to change sign at a singular point (normally one requires \(\nu \geq 0\) and then the inner product changes sign at singular points). Unfortunately, while \(\nu\) changes sign in a continuous fashion at singular points, it can also change sign discontinuously at regular points. For example, in the self-adjoint linear eigenvalue problem, along the trivial branch \(\nu\) will continuously pass through zero at each eigenvalue and will discontinuously change sign at some point between each consecutive pair of eigenvalues where the smallest singular value of the Jacobian changes from the preceding to the following eigenvalue.

If PLTMG detects a change in sign in \(\nu\) along the solution curve between the starting point and target point, and if ITASK = 1, the computation of the target point is abandoned in favor of computation of the possible singular point. A secant/bisection algorithm for the equation \(\nu(\sigma) = 0\) is used. More details of these procedures can be found in Bank and Chan [8] and the references therein. At the conclusion of this iteration, some tests are made to determine if the point is a bifurcation point, a limit point, or a regular point.
The algorithms in PLTMG are designed to handle only simple limit and bifurcation points, although on occasion we have observed them to work on some higher degree singular points as well. When a target or singular point has been successfully computed, PLTMG returns with \((RLTRGT, RTRGT)\) set to the current values of \((\lambda, \rho)\).

If PLTMG is called with \(ITASK = 2\) at a bifurcation point, parameters relevant for the continuation procedure are initialized for the bifurcating branch, but the solution itself remains unchanged. In the next call to PLTMG with \(ITASK = 0\) or \(ITASK = 1\), the continuation procedure will follow the bifurcating branch.

If PLTMG is called with \(ITASK = 3\) or \(ITASK = 4\), parameters relevant for the continuation procedure are reinitialized using the new parameter or functional; the solution itself remains unchanged. The two cases differ in that either \(\lambda\) or \(\rho\) can be held fixed during the reinitialization; for either case it is possible to specify either a new continuation parameter \(\lambda\), a new functional \(\rho\), or both.

The successful use of the continuation procedure requires guidance from the user. For example, it is possible to specify target values that cannot be reached. Also, since singular points are detected by changes in sign of \(\nu\), one can fool the singular-point detection algorithm by specifying target values sufficiently far away that two sign changes are passed on one step.

We now consider the cases \(5 \leq ITASK \leq 7\). We begin by noting that the discretization process can introduce spurious solution curves or cause significant distortions in the solution curves of the continuous problem (1.1); one must therefore be cautious in interpreting the numerical results [55]. As the mesh is refined or the mesh points are smoothed, the solution curves generally will move; the assumption of PLTMG is that, as a function of the discretization, the solution curves converge in some uniform fashion to those of the continuous problem, and that the mesh is sufficiently fine to capture the qualitative features of the continuous problem's solution curves in the regions of interest [8, 20]. Typically, for each point on the current grid, there are three natural points on a nearby new grid solution curve that can be associated with it: the point with the same \(\lambda\) value (\(ITASK = 5\)), the point with the same \(\rho\) value (\(ITASK = 6\)), and the point of intersection with the perpendicular hyperplane passing through the current solution point (\(ITASK = 7\)). Some typical examples are illustrated in Figure 4.5.

In some situations, all three points may not exist, or they may not be distinct. This is illustrated in Figure 4.5, right, where \(ITASK = 6\) and \(ITASK = 7\) correspond to the same fine grid point, while no (nearby) solution exists for \(ITASK = 5\).

We now consider the linear algebraic aspects of the problem. As with other problem types, the nonlinear systems for \(IPROB = 3\) are solved by the approximate Newton iteration \([30, 29]\) described in Figure 4.1. The nonlinear system to be solved has the form

\[
\begin{align*}
G(u, \lambda) &= 0, \\
N(u, \lambda) &= \sigma.
\end{align*}
\]

Here the operator \(G\) represents the finite element equations of order \(NDF\), and \(N\) the normalizing equation used in the continuation process; \(\sigma\) is the steplength. At
each step of the Newton process, the linear system to be solved has the form
\[
\begin{pmatrix}
G_u & G_\lambda \\
N_u & N_\lambda
\end{pmatrix}
\begin{pmatrix}
\delta u \\
\delta \lambda
\end{pmatrix}
= - \begin{pmatrix}
G(u, \lambda) \\
N(u, \lambda) - \sigma
\end{pmatrix},
\]
where $\delta u$ is a vector of length $NDF$ and $\delta \lambda$ is a scalar. The solution is constructed by solving
\[
G_u v = -G,
\]
\[
G_u w = -G_\lambda - G_\lambda \bar{u}_\lambda,
\]
\[
\bar{u}_\lambda \leftarrow \bar{u}_\lambda + w,
\]
\[
\delta \lambda = -\frac{N_u v + N - \sigma}{N_u \bar{u}_\lambda + N_\lambda},
\]
\[
\delta u = v + \delta \lambda \bar{u}_\lambda.
\]

The vector $\bar{u}_\lambda$, initially set to zero, is updated at every step. Thus the right-hand side $G_\lambda + G_\lambda \bar{u}_\lambda$ has the appearance of a residual, and $w$ may be viewed as an incremental update. At convergence, $\bar{u}_\lambda \lambda = \bar{u}$, so $\bar{u}$ is known at every Newton step. The linear systems involving $G_u$ are solved by the one of the linear solvers described in Section 4.3.

The block elimination process is embedded in the overall damped Newton process [21, 29] given in Figure 4.1. Here $U_k = (u^t, \lambda)$ is the $k$th Newton iterate, $\delta U_k = (\delta u^t, \delta \lambda)$, and $G_k = (G^t, N - \sigma)$. The norm $\|G_k\|$ is given by
\[
\|G_k\|^2 = \|G\|^2 + c|N - \sigma|^2,
\]
where $c$ is a scaling parameter ($SCALE$ in the $RP$ array) chosen to balance the two terms appropriately.

The case $IPROB = -3$ corresponds to a parallel solve of the block linear system (4.9), embedded in the overall Newton iteration. It is defined only for the cases $ITASK = 5, 6, 7$; at present there is no parallel implementation of the basic path following options. Thus we assume that the continuation is done on a coarse mesh on a single processor, and parallel computation is used only in the context of computing a highly refined solution at a particular point.
For continuation problems, PLTMG provides a limited amount of written output summarizing the state of the computation. All formats are designed for output devices having a minimum of 80 characters per line. All output is directed to the subroutine \textit{FILUTL}, which is responsible for creating the files \textit{BFILE} and \textit{JWFILE}.

For each call to PLTMG a banner is printed. Each continuation step results in a single line of output containing seven numbers. A typical example of such output is illustrated below:

\begin{verbatim}
pltmg: lambda rho lambda dot rho dot eigenvalue
0 3 0.99004E+01 0.39814E+01 -0.80768E-02 0.39890E+01 -0.94673E-04
\end{verbatim}

The first column contains the current value of IFLAG (in this example, IFLAG = 0). The second contains the value of ITNUM, the actual number of approximate Newton iterations used. The next four columns contain the current values of the parameter $\lambda$, the functional $\rho$, and their derivatives with respect to arclength along the current solution manifold $\dot{\lambda}$ and $\dot{\rho}$. The column labeled “eigenvalue” gives an approximation to the smallest singular value $\nu$ of the Jacobian matrix $G_u$.

As an example, we consider the nonlinear eigenvalue problem

\[-\Delta u = \lambda \sin u \quad \text{in } \Omega \equiv (0,1) \times (0,1),
\]

\[u = 0 \quad \text{on } \partial \Omega,
\]

with the functional given by

\[\rho(u, \lambda) = \int_{\Omega} u^2 \, dx \, dy.
\]

This problem has bifurcation points at the eigenvalues of the linear eigenvalue problem, $-\Delta u = \lambda u$, which are given by $\lambda_{k\ell} = (k^2 + \ell^2)\pi^2$, $k = 1, 2, \ldots$, $\ell = 1, 2, \ldots$. We chose as our coarse mesh a $17 \times 17$ uniform mesh, and will employ piecewise linear elements.

Our goal is to compute the first four eigenvalues and eigenfunctions. The first and third eigenvalues have multiplicity one. The second and fourth eigenvalues have multiplicity two. While the algorithms in PLTMG are not designed to handle multiplicities greater than one, the code performed in a satisfactory fashion and computed all four eigenvalues without difficulty. As a cautionary remark, one should not assume that the situation in this respect will always be so favorable.

We initialize at $\lambda = 0$ and continue to $\lambda = 10$ with \textit{ITASK} = 0 and then to $\lambda = 22$ with \textit{ITASK} = 1. At $\lambda = 22$, the sign of $\nu$ (eigenvalue) has changed, so PLTMG computes the singular point, in this case the first eigenvalue. These basic continuation steps were done on a uniform $17 \times 17$ mesh, using piecewise linear finite elements.

\begin{verbatim}
pltmg: lambda rho lambda dot rho dot eigenvalue
0 1 0.00000E+00 0.00000E+00 0.10000E+01 0.00000E+00 0.78964E-01
0 1 0.10000E+02 0.00000E+00 0.10000E+01 0.00000E+00 0.38375E-01
0 1 0.22000E+02 0.00000E+00 0.10000E+01 0.00000E+00 -0.81464E-02
\end{verbatim}
Note that the secant/bisection algorithm converged in three steps. After determining that the singular point was a bifurcation point, PLTMG makes an additional calculation to ensure that the tangent vector $\dot{u}_h$ corresponds to the current branch (in this case, the trivial branch).

We save the solution in a file in order to continue from this point to the second eigenvalue in a convenient manner (see Section 6.8), and switch branches ($ITASK = 2$). We then routinely continue on the bifurcating branch in several steps ($\rho = .01, \lambda = 25, 50, 100, 150, 300, 500$). At $\lambda = 500$, we $hp$-refine the mesh with a three calls to TRIGEN with $IADAPT = 1$, alternating with calls to PLTMG with $ITASK = 7$. The eigenfunction and mesh are shown in Figure 4.6.

We restore the solution at the bifurcation point and continue along the trivial branch to the second eigenvalue. We save the solution, switch branches and explore the bifurcating branch in a fashion similar to the first eigenvalue. A similar procedure is repeated for the third and fourth eigenvalues. The eigenfunctions computed on $hp$-refined meshes are shown in Figure 4.6. In Figure 4.8, we show the complete history of the calculation in terms of the continuation path.

### 4.7 Parameter Identification Problems.

When $IPROB = 4$, PLTMG solves the parameter identification problem (1.9)-(1.12). Up to ten scalar parameters are allowed; $1 \leq NRL \leq 10$ denotes the number of parameters. If one or more of the parameters influences the shape of $\Omega$ through $SXY$, then one should set $ITASK = 8$. This signals PLTMG to invoke certain additional procedures within the basic Newton iteration that modify the shape of $\Omega$.

The simple bounds on $\lambda_h$ are treated in a fashion analogous to the case $IPROB = 2$. In particular, we consider the Lagrangian

$$L(u_h, v_h, \lambda_h) = \rho(u_h, \lambda_h) + a(u_h, v_h) - \mu \left\{ \sum_{k=1}^{NRL} \log(\lambda_{h,k} - \lambda_k) - \log(\lambda_k - \lambda_{h,k}) \right\}$$

where $\mu > 0$ is the barrier parameter and $v_h$ is the Lagrange multiplier (a member of the finite element subspace). Our procedure computes a stationary point of the Lagrangian (4.10) using an approximate Newton method.

The linear algebra problem at each Newton iteration is of the form

$$
\begin{pmatrix}
H & A^T & C_u \\
A & 0 & C_v \\
C_u^T & C_v^T & D
\end{pmatrix}
\begin{pmatrix}
\delta u \\
\delta v \\
\delta \lambda
\end{pmatrix}
=
\begin{pmatrix}
b_u \\
b_v \\
b_\lambda
\end{pmatrix}
.$$
4.7. Parameter Identification Problems.

The first eigenfunction; $\text{NTF} = 1490$ and $\text{NDF} = 3506$.

The second eigenfunction; $\text{NTF} = 2150$ and $\text{NDF} = 4709$.

The third eigenfunction; $\text{NTF} = 2506$ and $\text{NDF} = 4999$.

The fourth eigenfunction; $\text{NTF} = 4684$ and $\text{NDF} = 4998$.

Figure 4.6.
Here the matrix $A$ is the Jacobian matrix corresponding the the bilinear form $a(u_h, v_h)$. In particular, linear systems involving $A$ (or $A^t$) are solved using the linear solver specified by METHOD. The matrix $H$ is symmetric and has the same sparsity pattern as $A$; other characteristics strongly depend on the particular problem. $C_u$ and $C_v$ are $NDF \times NRL$ rectangular matrices, generally composed of $NRL$ dense column vectors, and $D$ is a $NRL \times NRL$ symmetric matrix. The vectors $\delta u$ and $\delta v$ are the (Newton) updates for $u_h$ and the Lagrange multiplier $v_h$, respectively, and $\delta \lambda$ is the update for $\lambda_h$. $b_u$, $b_v$ and $b_\lambda$ correspond to the appropriate Newton residuals.

To describe the solution process for 4.11, we begin with the block factorization

$$
\begin{pmatrix}
H & A^t & C_u \\
A & 0 & C_v \\
C_u^t & C_v^t & D
\end{pmatrix}
= 
\begin{pmatrix}
I & 0 & 0 \\
0 & A & 0 \\
0 & 0 & I
\end{pmatrix}
\begin{pmatrix}
H & I & 0 \\
I & 0 & 0 \\
C_u^t & C_v^t & D
\end{pmatrix}
\begin{pmatrix}
I & 0 & \tilde{C}_v \\
0 & I & \tilde{C}_u \\
0 & 0 & I
\end{pmatrix}
\begin{pmatrix}
I & 0 & 0 \\
0 & A^t & 0 \\
0 & 0 & I
\end{pmatrix}
$$

where

$$
AC_v = C_v, \\
\tilde{C}_u = C_u - HC_v, \\
\tilde{D} = D - C_u^t \tilde{C}_v - \tilde{C}_v^t \tilde{C}_u.
$$

Computing $\tilde{C}_v$ requires solving $NRL$ linear systems with $A$. The two block diagonal matrices each require solution of one linear system with $A$ or $A^t$. Thus a total of $2 + NRL$ elliptic pde systems need to be solved in each Newton step. The block
lower triangular system requires solving one dense linear system with the symmetric \( NRL \times NRL \) matrix \( \bar{D} \).

The overall solution procedure is summarized below. Since the linear systems involving \( A \) are generally solved only approximately by iteration, we introduce the matrix \( \bar{u}_\lambda \approx \bar{C}_v \), that is generally the approximate \( \bar{C}_v \) saved from the previous Newton step. This allows the computation of the current \( C_v \) to be done as an update with a residual-like right hand side. \( \bar{u}_\lambda \) is initially set to zero, and updated with the solution of every linear system.

First we solve

\[
\begin{align*}
A\bar{b}_v &= b_v, \\
Aw &= C_v - A\bar{u}_\lambda, \\
\bar{u}_\lambda &\leftarrow \bar{u}_\lambda + w, \\
\bar{C}_v &= \bar{u}_\lambda.
\end{align*}
\]

All linear systems involving \( A \) are (approximately) solved using the linear solver specified by \textsc{METHOD}. Then we form

\[
\begin{align*}
\bar{b}_u &= b_u - H\bar{b}_v, \\
\bar{C}_u &= C_u - HC_v,
\end{align*}
\]

which requires sparse matrix multiplications with \( H \). Next we compute \( \delta \lambda \) using the Schur complement

\[
\begin{align*}
\bar{D} &= D - C_v^t\bar{C}_v - \bar{C}_v^t\bar{C}_u, \\
\bar{D}\delta \lambda &= b_\lambda - C_v^t\bar{b}_v - \bar{C}_v^t\bar{b}_u.
\end{align*}
\]

Finally, we form \( \delta u \) and \( \delta v \) from

\[
\begin{align*}
\delta u &= \bar{b}_v - \bar{C}_v\delta \lambda, \\
A^t\delta v &= \bar{b}_u - \bar{C}_u\delta \lambda.
\end{align*}
\]

The latter requires the use of the linear solver for \( A^t \). The basic Newton iteration is again that given in Figure 4.1 with the interpretation \( \mathcal{U}^t = (u^t_h, v^t_h, \lambda^t_h) \) and \( \mathcal{G}^t = (b^t_u, b^t_v, \bar{b}^t_\lambda) \).

When \textsc{IPROB} = -4, a parallel Newton algorithm is implemented, similar in structure to the case \textsc{IPROB} = -1. A domain decomposition solver analogous to that described in Section 4.4 is incorporated into the block elimination algorithm defined above.

As an example, we consider the problem

\[
\min \int_\Omega \nabla u^2 + \delta \sum_{i=1}^3 \lambda_i^2 \, dx,
\]

subject to the boundary value problem and inequality constraints

\[
\begin{align*}
-\Delta u &= 1 & \text{in } \Omega \\
u &= 0 & \text{on } \partial \Omega \\
\underline{\lambda}_i &\leq \lambda_i \leq \overline{\lambda}_i, & \text{for } 1 \leq i \leq 3.
\end{align*}
\]
The domain \( \Omega \) is shown in Figure 4.8. The interior box is free to move within the domain, with its position governed by three parameters. \((\lambda_1, \lambda_2)\) are the \((x, y)\) coordinates of the center of the box and \(\lambda_3\) is its angle of rotation. This is the special case \(\text{IPROB} = 4\) and \(\text{ITASK} = 8\) allowing parameters to control the geometry of the domain.

The domain was provided as a skeleton and the initial mesh generated by \(\text{TRIGEN}\). Both are shown in Figure 4.8. The optimization problem was then solved on this mesh using piecewise linear elements. This first optimization step involved substantial movement of the boundary, but the number of degrees of freedom was relatively small. Next, using alternating refinement and optimization steps, we created a final mesh with \(\text{NDF} = 22183\) degrees of freedom. In each optimization step, the interior point parameter \(\text{RMTRGT}\) was reduced by a factor of 2 starting from its initial value of 1. While the boundary was allowed to move on all these additional optimization steps, it moved very little. In Figure 4.8, we also show the final solution, Lagrange multiplier, and the a posteriori error estimate computed on the final mesh.

### 4.8 Optimal Control Problems.

When \(\text{IPROB} = 5\), \(\text{PLTMG}\) solves the control problem (1.13)-(1.16). This problem is similar to the case \(\text{IPROB} = 4\) except that now \(\lambda_h\) is a finite element function rather than a scalar. Here we consider the Lagrangian

\[
L(u_h, v_h, \lambda_h) = \rho(u_h, \lambda_h) + a(u_h, v_h)
- \mu \sum_{i=1}^{\text{NDF}} d_i \left\{ \log(\lambda_h(p_i) - \lambda(p_i)) + \log(\lambda(p_i) - \lambda_h(p_i)) \right\}
\]

(4.12)

where \(\mu > 0\) is the barrier parameter, \(d_i\) is the diagonal of the mass matrix corresponding to node \(p_i\), and \(v_h\) is the Lagrange multiplier. As usual, our algorithm seeks a stationary point of the Lagrangian (4.12) using an approximate Newton method.

The linear algebra problem at each Newton step is of the form

\[
\begin{pmatrix}
H & A^t & S_u \\
A & 0 & S_v \\
S_u^t & S_v^t & G
\end{pmatrix}
\begin{pmatrix}
\delta u \\
\delta v \\
\delta \lambda
\end{pmatrix}
= 
\begin{pmatrix}
b_u \\
b_v \\
b_\lambda
\end{pmatrix}
\]

(4.13)

Here \(H\) and \(A\) are defined as before. In typical problems \(G\) is a symmetric, positive definite matrix, corresponding the regularization terms in (4.12). The matrix \(G\) also has a nonnegative diagonal term arising from the inequality constraints for \(\lambda_h\). As before, linear systems involving \(A\) and \(A^t\) are easily solved using the linear solver specified by \(\text{METHOD}\). Additionally, since \(G\) formally has the same sparsity as the stiffness matrix \(A\), linear systems involving \(G\) are solved using a similar preconditioning strategy. The matrices \(S_u\) and \(S_v\) have the same symmetric sparsity structure as \(G\) and \(A\), but are generally not symmetric.

The skeleton (left) and initial mesh with \( \text{NTF} = 132 \) and \( \text{NDF} = 82 \) (right).

The mesh after the first optimization step (left) and the final mesh with \( \text{NTF} = 4215 \) and \( \text{NDF} = 22183 \) colored by polynomial degree (right).

The final mesh colored by element size (left) and the solution (right).

The Lagrange multiplier (left), and the error estimate for the final mesh (right).

Figure 4.8.
Our solver is based on block Gaussian elimination, similar to the case $\text{IPROB} = 4$. However, in the case of (4.13), it is too expensive to compute an exact Schur complement for the $(3, 3)$ block; instead we approximate the Schur complement by $G$ itself. Thus, our solution algorithm is just a preconditioner. In particular, it is one step of a block symmetric Gauss-Seidel iteration. This is realized as follows:

\[
\begin{align*}
A\tilde{c}_u &= b_v, \\
A^t\tilde{c}_v &= b_u - H\tilde{c}_u, \\
G\delta\lambda &= b_\lambda - S''_u\tilde{c}_u - S''_v\tilde{c}_v, \\
A\delta u &= b_v - S_v\delta\lambda, \\
A^t\delta v &= b_u - H\delta u - S_u\delta\lambda.
\end{align*}
\]

Linear systems involving $A, A^t,$ and $G$ are solved using the appropriate linear solver. If $G$ were replaced by the Schur complement and all linear systems solved exactly, this would yield the exact solution. This approximate solver is used as the preconditioner for the composite step conjugate gradient iteration.

When $\text{IPROB} = -5$, a parallel Newton algorithm is implemented, similar in structure to the case $\text{IPROB} = -1$. A domain decomposition solver analogous to that described in Section 4.4 is incorporated into the block preconditioner defined above.

As an example, we solve the optimal control problem

\[
\min \int_{\Omega} (u - u_0)^2 + \beta (\nabla u - \nabla u_0)^2 + \gamma \lambda^2 \, dx
\]

subject to the constraint equation

\[
\begin{align*}
-\Delta u &= \lambda \quad \text{in } \Omega \equiv (0, 1) \times (0, 1), \\
u &= 0 \quad \text{on } \partial\Omega,
\end{align*}
\]

and the inequalities

\[1 \leq \lambda \leq 10.\]

The target function $u_0$ was

\[u_0 = \sin(3\pi x) \sin(3\pi y),\]

$\beta = 10^{-2}$, and the regularization parameter $\gamma = 10^{-4}$.

This problem was solved in parallel using 16 processors, starting from an initial uniform $17 \times 17$ mesh and piecewise linear elements. This mesh was adaptively refined to $\text{NTF} = 13700$ elements and $\text{NDF} = 6935$, and partitioned, as illustrated in Figure 4.9. Each processor went through five iterations of adaptive refinement, with $\text{MXORD} = 1$, so that only piecewise linear polynomials were used. This produced a global mesh with $\text{NDG} = 450796$ and block $3 \times 3$ linear systems of order 1352388. The interior point parameter $\mu = \mu_0 = 10^{-2}$ on the $17 \times 17$ mesh, and thereafter was reduced by a factor of 2 in each refinement step, for a final size of $\mu \approx 1.4 \times 10^{-4}$. 
The final global mesh, colored by element size, is shown in Figure 4.9. In Figure 4.9, the solution (state variable), the Lagrange multiplier, the control $\lambda$, and the error estimate are also shown. Note that PLTMG chooses only one approximation space that is used for all three functions. Also all three functions contribute to the error estimate used in the adaptive procedure. The fact that the control $\lambda$ is essentially a piecewise constant, while the other two functions are relatively smooth is likely responsible for the observed refinement pattern.
The initial $17 \times 17$ mesh (left) and the load balance with $NTF = 13700$ (right).

The global refined mesh with $NDG = 450796$ colored by size (left) and the convergence history of the Domain Decomposition solver (right).

The solution $u$ (left) and the Lagrange multiplier $v$ (right).

The control $\lambda$ (left) and the error estimate on the final mesh (right).

Figure 4.9.
Chapter 5

Graphics

5.1 Overview.

The graphics package associated with PLTMG is composed of subroutines TRIPLT, INPLT, and GPHPLT. These routines are written in self-contained, portable Fortran, addressing the graphics output device through subroutines PLINE, PFILL, PFRAME, and PLTUTL. The specifications for these routines are given in Section 6.12.

Typical graphics output consists of three windows or frames. There is a large square window on the left, and two smaller square windows on the right. The main image typically appears in the large frame, and other useful information (for example, a legend matching colors to function values) appears in the smaller frames. The graphics interface provides z-buffer information, for use in three dimensional imaging systems such as OpenGL. All the graphics routines are written such that the image appearing in the main window can be animated using such graphics systems when appropriate.

Subroutine TRIPLT graphs the solution and various associated functions (e.g., $\dot{u}$, $\psi_r$, $\epsilon_t$). TRIPLT also has options for plotting vector functions (e.g., $\nabla u_h$). Subroutine INPLT can display either a triangulation or a skeleton, with elements or regions colored according to various attributes such as the quality of the elements in a triangulation. Subroutine GPHPLT displays various graphs and charts containing timings, convergence histories, and other items of interest.

The parameter MXCOLR is a device dependent constant, stating the maximum number of colors available for use by the graphics package. We assume that $2 \leq MXCOLR$. While it is possible to make some interesting plots and contour maps with TRIPLT using only monochrome devices ($MXCOLR = 2$), TRIPLT makes extensive use of available color facilities in producing (shaded) three-dimensional surface plots and vector plots. GPHPLT and INPLT also use color, but in less critical ways.

Subroutines TRIPLT, INPLT and GPHPLT offer some capabilities for parallel processing. In the parallel processing environment, only the master process (corre-
sponding to \textit{IRGN = 1}) makes calls to the graphics interface routines (\textit{PLTUTL, PFRAME, PLINE,} and \textit{PFILL}). However, in the case of \textit{TRIPTL} or \textit{INPLT}, one may wish to plot the solution, error, or some other function in situations where the data is distributed among the processors. If MPI is turned on (\textit{MPISW = 1}), and \textit{MPIRGN = 0}, then \textit{TRIPTL} and \textit{INPLT} collect data from all other processors, and draw a composite picture consisting of the union of the refined regions for each processor. If the problem is sufficiently large that it is impossible or inefficient to collect all the data on a single processor, each processor can coarsen its data before sending it to the master process. This coarsening process is controlled by the parameters \textit{ICRSN} and \textit{ITRGT}. If MPI is turned off (\textit{MPISW = -1}), then \textit{TRIPTL} and \textit{INPLT} draw the function on processor one (refined in region one and coarse elsewhere). If one wishes to see the complete image as it exists on some other processor, say processor \textit{I}, set \textit{MPISW = 1}, \textit{MPIRGN = I}, and call \textit{TRIPTL} or \textit{INPLT}. For some options, \textit{GPHPLT} collects data from all processors when MPI is turned on, for example in presenting timing and load balancing data.

For most of the examples of graphics output, we solved Laplace’s equation in a circle of radius one with a crack along the positive \textit{x} axis. This domain was used to illustrate the triangulation data structure in Section 2.3. Nonhomogeneous Dirichlet boundary conditions were imposed on the circular boundary such that the true solution is \( u = r^{1/4} \sin(\theta/4) \), the leading term in the singularity due to the crack tip. Some example output in Section 5.4 came from other problems, in cases where it could not be provided by our simple example.

\section{5.2 Subroutine \textit{TRIPTL}.}

\textit{TRIPTL} is called using the statement

\begin{verbatim}
Call TRIPTL( VX, VY, SF, ITNODE, IBNDRY, ITDOF, E,
    IP, RP, SP, GF, QXY, SXY )
\end{verbatim}

The arrays \textit{VX}, \textit{VY}, \textit{SF}, \textit{ITNODE}, \textit{IBNDRY}, \textit{ITDOF}, \textit{E}, \textit{IP}, \textit{RP}, \textit{SP}, \textit{GF}, \textit{QXY}, \textit{SXY} should define a triangulation. \textit{GF} and \textit{E} contain functions for potential display. \textit{TRIPTL} uses several variables from the \textit{IP}, \textit{RP}, and \textit{SP} arrays, as shown in Tables 2.6–2.8. The string variable \textit{FTITLE} is the character string displayed as a label above the graph. Additionally, \textit{TRIPTL} can use the Fortran subroutine \textit{QXY}. Subroutine \textit{QXY} is documented in Section 2.8. The error flag \textit{IFLAG} is set as in Table 2.9.

The parameter \textit{IFUN} specifies the function to be plotted. The available options are summarized in Table 5.1. Some of these functions are not defined for all problem types. Although there are many possibilities for \textit{IFUN}, they may be classified as \textit{surface plots} and \textit{vector plots}.

For surface plots, all functions are continuous with the (possible) exceptions of the error and scaling factors, which are piecewise constant on triangles, and \textit{QXY}, which can be multivalued along element boundaries due to discontinuities in \( \nabla u_h \).

If desired, a discontinuous function can be mapped to a continuous function using a local averaging technique. This is invoked by setting the switch \textit{ICONT = 1}.  


5.2. Subroutine *TRIPLT.*

![Figure 5.1](image1)

**Figure 5.1.** The solution IFUN = 0 and the error IFUN = 5.

![Figure 5.2](image2)

**Figure 5.2.** The case IFUN = 0, (NX, NY, NZ) = (1,−1,−1), and IFUN = 5, (NX, NY, NZ) = (1,1,1).

![Figure 5.3](image3)

**Figure 5.3.** The case IFUN = 0, (NX, NY, NZ) = (1,−1,−1), RMAG = 2, CENX = .5, CENY = .3, and the case IFUN = 5, (NX, NY, NZ) = (0,0,1), RMAG = 2, CENX = .5, CENY = .3.

![Figure 5.4](image4)

**Figure 5.4.** The case LINES = 1 and the case LINES = 3. The corresponding picture for LINES = 0 is in Figure 5.1.
Figure 5.5. The case IFUN = 2, ICONT = 1. \((NX, NY, NZ) = (0, 0, 1)\) and \((NX, NY, NZ) = (1, 1, 1)\).

Figure 5.6. The case IFUN = 1, ICONT = 1. \((NX, NY, NZ) = (1, 1, 1)\). In the picture on the right RMAG = 2, CENX = .5, and CENY = .3.

Figure 5.7. Triangles colored by label \((INPLSW = 0)\) and by quality \((INPLSW = 2)\).

Figure 5.8. Triangles colored by size \((INPLSW = 6)\) and by polynomial degree \((INPLSW = 5)\).
5.2. Subroutine TRIPLT.

<table>
<thead>
<tr>
<th>IFUN</th>
<th>displayed function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>the solution $u_h$</td>
</tr>
<tr>
<td>1</td>
<td>the scalar function $</td>
</tr>
<tr>
<td>2</td>
<td>the vector function $\nabla u_h$</td>
</tr>
<tr>
<td>3</td>
<td>the alternate function $QXY$</td>
</tr>
<tr>
<td>4</td>
<td>the alternate vector function $QXY$</td>
</tr>
<tr>
<td>5</td>
<td>the error estimate $</td>
</tr>
<tr>
<td>6</td>
<td>the tangent function $\dot{u}$</td>
</tr>
<tr>
<td>7</td>
<td>the right singular vector $\psi_r$</td>
</tr>
<tr>
<td>8</td>
<td>the left singular vector $\psi_l$</td>
</tr>
<tr>
<td>9</td>
<td>the Lagrange multiplier $u_m$</td>
</tr>
<tr>
<td>10</td>
<td>the control variable $\lambda_h$</td>
</tr>
<tr>
<td>11</td>
<td>the dual function $\omega$</td>
</tr>
<tr>
<td>12</td>
<td>the scaling factor $\alpha_t$</td>
</tr>
</tbody>
</table>

Table 5.1. The values of IFUN.

5.2.1 Surface Plots.

In the case of surface plots, $NCON$ specifies the number of contours (colors) to be used. If $NCON > MXCOLR - 2$, some colors are used for more than one contour. The parameters $SMIN$ and $SMAX$ can be used to specify the limits of the color scale. If $SMIN < SMAX$, then these values are used as limits, with parts of the function lying outside $(SMIN, SMAX)$ colored white. Otherwise, the largest and smallest values of the displayed function are used as limits.

Each picture consists of three frames; a large plot on the left and a two-part legend on the right. The upper right contains a scale relating colors to function values; three scales are available using the switch $ISCALE$ as described in Section 5.2.4. For the case $IFUN = 5$, $ICONT = 0$, a histogram showing the distribution of errors $||\epsilon_t||_{H^1(t)}$ is also provided in this legend. Five line-drawing options using $LINES$ and eleven labeling options using $NUMBRS$ are also available. $RMAG$, $CENX$, and $CENY$ provide a zoom-in capability as described in Section 5.2.3.

The main image can be animated using three dimensional imaging systems such as OpenGL. The viewing perspective can be statically set as well, using the triple $d = (NX, NY, NZ)$. The three-dimensional surface is projected into the plane orthogonal to $d$, and the function is drawn as it would appear to an observer viewing the surface from a line of sight parallel to $d$. The vectors $(NX, NY, NZ)$ and $-(NX, NY, NZ)$ cause the same projection to be computed; however, different pictures are generally produced for the two cases. In the former case one observes the projection on the “front” of the plane, and in the latter case one observes the projection on the “back” of the plane. If $MXCOLR$ is sufficiently large, the surface will be shaded relative to a light source directly behind the viewer, imparting some additional three-dimensional character to the picture.

The lower right-hand legend provides guidance in understanding three-dimensional
surface plots. In this case the legend contains a “flat” version of the main picture, allowing another avenue for orienting oneself with respect to the viewing perspective. Some examples of surface plots are given in Figures 5.1–5.4.

5.2.2 Vector Plots.

Color plays an important role in the vector plots. Different colors correspond to different directions in the vector field. This is illustrated in the color wheel portion of the upper right-hand legend. The number of directions is specified by the parameter \( NCON \). Different intensities of the same color correspond to the magnitude of the vector; darker shades correspond to smaller magnitudes, and lighter shades correspond to higher magnitudes. The correspondence between color intensity and vector magnitude is illustrated for an example color in the upper right-hand legend. The parameters \( SMIN \) and \( SMAX \) are used to specify the limits of the color intensity scale for the magnitude of the vector. As with surface plots, if \( SMIN < SMAX \), then these values are used as limits; otherwise the largest and smallest magnitudes of the vector function are used.

Three scales for the vector magnitude are available using the option switch \( ISCALE \). Five line-drawing options using \( LINES \) and eleven labeling options using \( NUMBRS \) are also available, and \( RMAG, CENX, \) and \( CENY \) provide zoom-in capabilities. The triple \( (NX,NY,NZ) \) specifies a direction as in the case of surface plots. In this case the surface plotted is the linear interpolant of the magnitude of the vector function.\(^9\) In this case the elements remain colored as in a two dimensional vector plot. Some examples of vector plots are given in Figures 5.5 and 5.6.

5.2.3 Parameters \( RMAG, CENX, \) and \( CENY \).

The parameters \( RMAG, CENX, \) and \( CENY \) provide a zoom-in option. \( RMAG \) is the magnification factor relative to the picture coordinates. For example, if \( RMAG = 1 \) the whole picture will be drawn; if \( RMAG = 2 \), the picture is scaled by a factor of 2 in both directions and thus no longer fits on the output device. One must now choose a window and view only a portion of the picture. The fractions \( 0 \leq CENX \leq 1 \) and \( 0 \leq CENY \leq 1 \) are used for this purpose. In particular \( (CENX, CENY) \) specifies the point that will appear at the center of the magnified window. If \( RMAG = 1 \), the values of \( CENX \) and \( CENY \) are ignored. Some examples are shown in Figure 5.3 and Figure 5.6 (right).

As an aid to understanding, the lower right legend contains a copy of the complete picture (corresponding to \( RMAG = 1 \)). Whenever \( RMAG > 1 \), a small box is drawn in this legend depicting the portion of the picture appearing in the main graph. The box is supplemented by a crosshair locator, since the box becomes too small to be visible for large magnification factors.

\(^9\) For the actual magnitude, the surface of each triangular element is not necessarily a plane, making the hidden surface problem more difficult.
5.2.4 Parameters ISCALE, LINES, NUMBRS, and MPIRGN.

The parameter ISCALE provides three scaling options, summarized in Table 5.2. For linear scaling, drawn contours are equally spaced with respect to the largest and smallest values of the given function \( z(x, y) \). If ISCALE = 1, then the contours are equally spaced with respect to the largest and smallest values of \( \log z \). If ISCALE = 2, then the contours are equally spaced with respect to largest and smallest values of the function \( \sinh^{-1} z \). The logarithmic scaling clearly requires \( z \) to be positive. The \( \sinh^{-1} \) scaling is always defined, having a (signed) logarithmic behavior for large \( |z| \) and a linear behavior for small \( |z| \). If ISCALE = 1 and \( z \leq 0 \) at some node, then TRIPLT defaults to the \( \sinh^{-1} \) scaling. In Figure 5.1, the solution \( u_h \) was drawn using the linear scale (ISCALE = 0), while the error estimate was drawn using the logarithmic scale (ISCALE = 1).

Five line drawing options are available, specified through the parameter LINES, as summarized in Table 5.2. If LINES = 0, TRIPLT will draw edges of all triangles in the mesh. If LINES = 1, only boundary edges and edges separating triangles from different regions are drawn. The case LINES = 2 is similar to the case LINES = 1, except that here boundary edges and edges separating triangles from different processors are drawn. When LINES = 3 for surface plots, TRIPLT draws boundary triangle edges and contour lines separating contours of different colors. This option produces a traditional contour map on monochrome devices and thus is useful when MXCOLR = 2. Some examples for LINES = 1 and LINES = 3 are shown in Figure 5.4. The option LINES=3 is not implemented for vector plots. The option LINES=-1 displays the underling (typically refined) graphics triangulation that was actually used by TRIPLT in making the image; this is mainly of interest for debugging.

Eleven labeling options are available in TRIPLT; these are specified through the parameter NUMBRS, as summarized in Table 5.2. When NUMBRS \( \neq 0 \), three-dimensional plotting is disabled; the result will be a “flat” (but labeled) surface. Some examples are shown in Figures 2.1 and 2.2.

In making images using MPI, the usual situation is that each processor contributes its part of the image, corresponding to its refined subdomain. However, in certain situations (such as debugging), one may wish to see the complete image as it exists on an individual processor. The parameter MPIRGN allows this. When MPIRGN = 0, its default value, then all processors contribute to a given image in the usual way. When MPIRGN = \( I \) for \( 1 \leq I \leq \text{NPROC} \), then the complete image from processor \( I \) is drawn.

5.2.5 Parameters ICRSN and ITRGT.

When NDF becomes very large, the amount of data used to make an image may become too large for animated display systems like OpenGL or for Postscript files of reasonable size.\(^{10}\) In this situation, one may wish to compress the data and make a lower resolution image. The parameter ICRSN indicates whether or not to coarsen

\(^{10}\)Raster graphics images like those produced by XPM files are largely independent of the size of the underlying data set.
<table>
<thead>
<tr>
<th>ISCALE</th>
<th>scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>linear</td>
</tr>
<tr>
<td>1</td>
<td>logarithmic</td>
</tr>
<tr>
<td>2</td>
<td>sinh$^{-1}$</td>
</tr>
<tr>
<td>LINES</td>
<td>line drawing option</td>
</tr>
<tr>
<td>0</td>
<td>all triangle edges</td>
</tr>
<tr>
<td>1</td>
<td>boundary/interface edges</td>
</tr>
<tr>
<td>2</td>
<td>load balance boundary edges</td>
</tr>
<tr>
<td>3</td>
<td>contours</td>
</tr>
<tr>
<td>-1</td>
<td>underlying graphics triangulation</td>
</tr>
<tr>
<td>NUMBRS</td>
<td>labeling option</td>
</tr>
<tr>
<td>0</td>
<td>no labels</td>
</tr>
<tr>
<td>1</td>
<td>triangles/subregions</td>
</tr>
<tr>
<td>2</td>
<td>vertices</td>
</tr>
<tr>
<td>3</td>
<td>edges</td>
</tr>
<tr>
<td>4</td>
<td>curved edges</td>
</tr>
<tr>
<td>5</td>
<td>edge type</td>
</tr>
<tr>
<td>6</td>
<td>edge labels</td>
</tr>
<tr>
<td>7</td>
<td>processor</td>
</tr>
<tr>
<td>8</td>
<td>vertex type</td>
</tr>
<tr>
<td>9</td>
<td>degrees of freedom</td>
</tr>
<tr>
<td>10</td>
<td>element degree</td>
</tr>
<tr>
<td>MPIRGN</td>
<td>image option</td>
</tr>
<tr>
<td>0</td>
<td>all processors contribute</td>
</tr>
<tr>
<td>$I &gt; 0$</td>
<td>draw image from processor $I$</td>
</tr>
<tr>
<td>ICRSN</td>
<td>coarsening option</td>
</tr>
<tr>
<td>0</td>
<td>no coarsening</td>
</tr>
<tr>
<td>1</td>
<td>coarsen global subspace</td>
</tr>
<tr>
<td>ICONT</td>
<td>smoothing option</td>
</tr>
<tr>
<td>0</td>
<td>no smoothing</td>
</tr>
<tr>
<td>1</td>
<td>smooth piecewise constant function</td>
</tr>
</tbody>
</table>

Table 5.2. The values of ISCALE, LINES, NUMBRS, MPIRGN, ICRSN, and ICONT.

the global subspace, as indicated in Table 5.2. If ICRSN = 1, then the parameter ITRGT specifies the target number of degrees of freedom for the coarsened subspace. The coarsening option is very much like the mesh coarsening option in TRIGEN; many of the same subroutines are used, and the overall coarsening strategy is the same. However, the coarsening criteria is different. When MPI is on (MPISW = 1),
each processor independently coarsens the subspace for its subregion to a target of $\frac{\text{ITRGT}}{\text{NPROC}}$ degrees of freedom. Thus, when the subspaces are later combined, the global subspace appearing in the image will have at most $\text{ITRGT}$ degrees of freedom.

When the mesh is coarsened, all numbering options are disabled; $\text{NUMBRS} = 0$ is always used. The setting $\text{LINES} = 0$ is reset to $\text{LINES} = 1$, and $\text{ICONT} = 1$ is always used.

### 5.2.6 Some Algorithmic Details.

The main algorithms of interest in TRIPLT are those for hidden line and surface removal. In the general case of a surface plot, one must make comparisons between various triangles to determine whether a given triangle blocks another with respect to the viewer. Since the triangular mesh is generally unstructured, our goal is to organize the data to minimize the number of comparisons between triangles.

Generally, for surface plots in which $(\text{NX}, \text{NY}, \text{NZ}) \neq (0,0,1)$, a partial order is constructed in which elements farthest from the viewer are ordered first, and those closest to the viewer are ordered last. The elements are then drawn and colored in order, with the elements closer to the viewer (possibly) overwriting some elements that are farther away. The notion of distance from the viewer is defined with respect to the $x$ and $y$ coordinates only, so that the same ordering is computed independent of the function being graphed. A typical element is compared only to elements with which it shares a common edge; it is ordered before any edge neighbors closer to the viewer and after any neighbors farther away. Since any element has at most three neighbors, this greatly limits the number of comparisons necessary and completely solves the ordering problem for a convex domain with no holes.

Unfortunately, many domains are not convex and have holes, so that elements with boundary edges must be treated as special cases. Thus we make a list of triangles with boundary edges, sort them with respect to the direction (in the $(x,y)$ plane) perpendicular to the $(\text{NX}, \text{NY})$ components of the viewing direction. Boundary edges are also sorted by whether they face “backward” or “forward” with respect to $(\text{NX}, \text{NY})$. With these preliminary calculations done, all pairs of relevant triangles that might conflict are tested and appropriate ordering constraints imposed. For a mesh with $\text{NTF}$ triangles, the number of boundary triangles is $O(\sqrt{\text{NTF}})$, so that in the worst case (every boundary element compared with every other boundary element), this will still be only $O(\text{NTF})$ work. Since only $O(\text{NTF})$ work is required for the interior elements, the overall work is still $O(\text{NTF})$.

### 5.3 Subroutine INPLT.

Subroutine INPLT is a graphics routine for displaying the input data defining a triangulation or a skeleton. INPLT is called using the statement

```
Call INPLT( VX, VY, SF, ITNODE, IBNDRY, ITDOF, IP, RP, SP, SXY )
```
The arrays $V_X$, $V_Y$, $IBNDRY$, $ITNODE$, and $SF$ define either a triangulation or a skeleton ($INPLT$ uses the value of $ITNODE(3,1)$, which is zero for a skeleton and positive for a triangulation, to distinguish these cases). The string variable $ITITLE$ is displayed as a banner above the graph. Variables in the $IP$, $RP$, and $SP$ arrays used by $INPLT$ are shown in Tables 2.6–2.8. $INPLT$ was used to make Figures 3.1–3.8, and 3.9, among others in this manual.

<table>
<thead>
<tr>
<th>$INPLSW$</th>
<th>triangulation</th>
<th>skeleton</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>user label</td>
<td>user label</td>
</tr>
<tr>
<td>1</td>
<td>load balance</td>
<td>subregion</td>
</tr>
<tr>
<td>2</td>
<td>element quality</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>largest angle</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>smallest angle</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>mesh grading</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>polynomial degree</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>element diameter</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.3. The values of $INPLSW$.

### 5.3.1 Triangle Plots.

For triangle plots, the elements in the triangulation are colored to depict some feature of the mesh. The available options are controlled by the switch $INPLSW$ as summarized in Table 5.3.

If $INPLSW = 0$, the elements in the mesh are colored according to the user supplied labels in $ITNODE(5,I)$; all elements with the same label will have the same color. If $INPLSW = 1$, the elements in the mesh are colored according to the load balance ($ITNODE(4,I)$).

For $2 \leq INPLSW \leq 5$, $INPLT$ colors the elements of the triangulation according to their quality, measured by $q(t)$ in (3.1), their largest angle, their smallest angle, and the local mesh grading, respectively. For each of these measures, five numbers are printed in the upper right legend. The row labeled “average” refers to the average of that quantity over all elements in the mesh; “worst” reports the smallest value of $q(t)$, largest angle, smallest angle, or the steepest local grade over all of the elements. The rows labeled “good,” “fair,” and “poor” report the percentage of elements in each category and depict the corresponding colors.

For $q(t)$, good means $q(t) \geq \sqrt{3}/2$, fair means $.6 \leq q(t) < \sqrt{3}/2$, and poor means $q(t) < .6$. For large angles, good means $A(t) \leq \pi/2$, fair means $\pi/2 < A(t) \leq 2\pi/3$, and poor means $A(t) > 2\pi/3$ ($A(t)$ is the largest angle). For small angles, good means $\arccos(4/5) \leq a(t)$, fair means $\arccos(13/14) \leq a(t) < \arccos(4/5)$ and poor means $a(t) < \arccos(13/14)$ ($a(t)$ is the smallest angle). Triangles that are good in terms of $q(t)$ are (necessarily) also good in terms of large and small angles. Those that are fair in terms of $q(t)$ must be good or fair in terms of large and small angles.
5.3. Subroutine \textit{INPLT}.

angles (but not conversely). The local mesh grading \( g_v \) at any vertex \( v \) in the mesh is the ratio of the largest to smallest lengths among all the element edges having \( v \) as an endpoint. The mesh grade for an element \( g(t) \) is the maximum of \( g_v \) among its three vertices. For mesh grading, good means \( g(t) < 2 \), fair means \( 2 \leq g(t) \leq 3 \) and poor means \( g(t) > 3 \). In Bank and Yserentant [39], control of mesh grading is seen to be a crucial point in creating nonuniform meshes where the \( \mathcal{L}_2(\Omega) \) projection of a function \( u \in \mathcal{H}^1(\Omega) \) is stable in the \( \mathcal{H}^1(\Omega) \) norm.

When \( \text{INPLSW} = 6 \), \text{INPLT} produces an image in which each element is colored according to its polynomial degree. A histogram showing the distribution of element degrees appears in the legend. When \( \text{INPLSW} = 7 \), \text{INPLT} produces an image in which each element is colored according to its size. A histogram showing the distribution of element sizes appears in the legend. Although any scaling option available through \text{ISCALE} can be used, generally the logarithmic scaling (\text{ISCALE} = 1) produces the most useful image. Some example images made using \text{INPLT} are shown in Figures 5.7 and 5.8.

The meanings and use of \text{RMAG}, \text{CENX}, \text{CENY}, and \text{MXCOLR} are identical to \text{TRIPLT}. Labeling options using \text{NUMBRS} are summarized in Table 5.2. \text{INPLT} was used with various \text{NUMBRS} options to produce Figure 2.1 although the legends on the right-hand sides of the pictures were deleted. For the main graph, three line-drawing options are available using \text{LINES}, as summarized in Table 5.2.

The meaning and use of parameter \text{MPIRGN} is the same in subroutine \text{INPLT} as in \text{TRIPLT}. Subroutine \text{INPLT} also allows mesh coarsening, but the criterion is different. In \text{INPLT}, each element is a single color and the images are two dimensional, and the coarsening criterion reflects these differences. As with \text{TRIPLT}, \text{NUMBRS} = 0 is always specified for a coarsened mesh and \text{LINES} = 0 is reset to \text{LINES} = 1.

5.3.2 Skeleton Plots.

As with triangle plots, the subregions of the skeleton are colored according to the option specified by \text{INPLSW} as summarized in Table 5.3. If \text{INPLSW} = 0, the subregions are colored according to the user supplied labels in \text{ITNODE(5,I)}, similar to the case of a triangulation. If \text{INPLSW} = 1, each subregion is given a different color.

Subroutine \text{INPLT} draws a skeleton by first creating a crude triangulation based on the skeleton, and then drawing the triangulation. Here shape regularity and overall quality of the triangulation is not an issue; rather, keeping the number of elements small and computing the triangulation quickly are important. The option \text{LINES} = -1 displays the underlying triangulation used in the skeleton plot. It was included mainly for debugging purposes.

The parameters \text{RMAG}, \text{CENX}, \text{CENY}, and \text{MXCOLR} are the same as for triangle plots. Labeling options using \text{NUMBRS} are summarized in Table 5.2. There are no coarsening or parallel computation options available for skeleton plots. \text{INPLT} was used with various \text{NUMBRS} options to produce Figure 2.2.
5.4 Subroutine GPHPLT.

Subroutine GPHPLT displays an assortment of data related to the performance of various algorithms and subroutines in PLTMG and TRIGEN using a graphical format.

GPHPLT is called using the statement

\[
\text{Call GPHPLT( IP, RP, SP )}
\]

GPHPLT makes use of the arrays TIME and HIST, that reside in a common block initialized by PLTMG and TRIGEN when IFIRST \( \neq 0 \). The string variable GTITLE is displayed as a banner above the graph. Other variables in the IP, RP, and SP arrays used by GPHPLT are shown in Tables 2.6–2.8.

<table>
<thead>
<tr>
<th>( IGRSW )</th>
<th>displayed graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Newton iteration convergence history</td>
</tr>
<tr>
<td>1</td>
<td>CSCG/CSBCG iteration convergence history</td>
</tr>
<tr>
<td>-1</td>
<td>matrix statistics</td>
</tr>
<tr>
<td>2</td>
<td>individual subroutine timing statistics</td>
</tr>
<tr>
<td>-2</td>
<td>time pie chart</td>
</tr>
<tr>
<td>3</td>
<td>the continuation path</td>
</tr>
<tr>
<td>-3</td>
<td>load balance</td>
</tr>
<tr>
<td>4</td>
<td>error estimates for ( H^1 ) norm</td>
</tr>
<tr>
<td>-4</td>
<td>error estimates for ( L_2 ) norm</td>
</tr>
<tr>
<td>5</td>
<td>the IP array</td>
</tr>
<tr>
<td>-5</td>
<td>the SP array</td>
</tr>
<tr>
<td>6</td>
<td>the RP array</td>
</tr>
</tbody>
</table>

Table 5.4. The values of IGRSW.

\( IGRSW \) is an integer switch for selecting the displayed graph; the available possibilities are summarized in Table 5.4.

5.4.1 Iteration Information.

For the cases \( IGRSW = -1, 0, 1 \), information about various iterations and preconditioners is displayed. In all three cases, the same three graphs are drawn. The large main window contains the information indicated in Table 5.4 for the corresponding value of \( IGRSW \). The other two graphs appear in the two smaller frames on the right. Examples are shown in Figures 5.9–5.10.

In the case \( IGRSW = 0 \), in the main window GPHPLT graphs the functions

\[
\mathcal{R}_k = \log_{10} \left\{ \frac{\| \mathcal{G}_k \|}{\| \mathcal{G}_0 \|} \right\} \quad \text{and} \quad \mathcal{E}_k = \log_{10} \left\{ \frac{\| \delta \mathcal{S}_k \|}{\| \mathcal{U}_k \|} \right\}.
\]

\( \mathcal{G}_k \) is the residual for the Newton iteration, while \( \delta \mathcal{S}_k \) is the incremental change in
5.4. Subroutine GPHPLT.

Figure 5.9. The cases $\text{IGRSW} = 0$ and $\text{IGRSW} = 1$.

Figure 5.10. The cases $\text{IGRSW} = -1$ and $\text{IGRSW} = 4$ with $(MX, MY, MZ) = (1, -1, 1)$.

Figure 5.11. The cases $\text{IGRSW} = 2$ and $\text{IGRSW} = -2$.

Figure 5.12. The cases $\text{IGRSW} = 3$ and $\text{IGRSW} = -3$. 
the solution $S_k$. The precise meaning of $G_k$ and $S_k$ varies according to the system of nonlinear equations solved for each problem class addressed by PLTMG. Both convergence histories are plotted in a bar graph of $R_k$ and $E_k$ versus iteration index $k$. When this corresponds to a regular (serial) solution ($IPROB > 0$), the relative residuals are red bars, while the solution increments are blue. At most, information about the last twenty Newton iterations is displayed. When $IPROB < 0$, the Newton iteration employs the parallel domain decomposition solver in place of the simple preconditioned linear solver. In this case, the residuals are magenta bars, and the solution increments are cyan.

Nominally, the rate of convergence for Newton’s method should asymptotically be quadratic; however, the convergence becomes linear when systems of linear equations involving the Jacobian matrix are only approximately solved.

In the case $IGRSW = 1$, in the main window $GPHPLT$ graphs the function

$$S(k) = \log_{10} \left\{ \frac{\|r_k\|}{\|r_0\|} \right\}. $$

Here $r_k$ is the residual of a set of linear equations to be solved and $k$ is the iteration number. The displayed histories are for linear systems solved in the most recent Newton iteration. Up to four such systems are solved in each Newton step, depending on the value of $IPROB$. In all cases, only information about the last twenty iterations is saved and displayed.
Either the composite step conjugate gradient method or composite step bi-conjugate gradient method is used [10, 9], preconditioned by the solver specified by \textit{METHOD}. Each individual step is marked with a small icon; a color pair (green, red), (blue, yellow), (cyan, magenta), (white, black) is assigned to each history. In each case, for simple steps the icon is colored with the first color (e.g., green), while for composite steps the icon is colored with the second color (e.g., red).

In the case \textit{IGRSW} = −1, in the main window statistics related to the the sparse matrix and its factorizations is displayed. Potential information about five matrices is given:

1. The stiffness matrix, colored yellow.
2. The \textit{ILU} factorization, colored green.
3. The \textit{HB} factorization, colored cyan.
4. The block diagonal factorization, colored blue.
5. The complete \textit{LU} factorization, colored magenta.

The first four correspond to matrices and factorizations that have been computed; \textit{ILU} and \textit{HB} and block diagonal factorization data appear only if they have been selected by the parameter \textit{METHOD}. The complete \textit{LU} factorization data is included for reference, and indicates data relevant to sparse Gaussian Elimination with the minimum degree ordering.

At the top, two columns of figures appear. The first is the average number of nonzeros per row in the matrix. The second is the same, but given as a percentage relative to sparse Gaussian Elimination. Below is a bar graph reflecting the actual storage used. This is different from the numerical data, in that if the matrices are symmetric, only the diagonal and upper triangle are stored. The storage bar for Gaussian Elimination (\textit{LU}) is hypothetical only. In each colored bar, a vertical black line might appear; this indicates the size of the \textit{JA} or \textit{JU} arrays relative to the corresponding \textit{A} or \textit{U} arrays.

\subsection{5.4.2 Timing Statistics.}

If \textit{IGRSW} = 2, \textit{GPHPLT} prints a summary of timing statistics for \textit{PLTMG} and \textit{TRIGEN}. An example is given in Figure 5.11. Statistics are given both for the total accumulated time since initialization (\textit{IFIRST} = 1) and for the time spent during the last call to \textit{PLTMG} or \textit{TRIGEN}. The timings are itemized with respect to subroutines that carry out major computational tasks in the package. These subroutines are listed in Table 5.5. Depending on the problem, some of these routines may not be called.

A bar graph is drawn illustrating the percentage of time spent in each routine. Each bar in the graph is partitioned into a part corresponding to the last call to \textit{PLTMG} (red) and a part corresponding to all preceding calls (blue). The timing pie graph described below appears in the upper right frame.
If $IGRSW = -2$, $GPHPLT$ displays a pie graph summarizing the same information. Each routine in Table 5.5 is assigned to one of six categories: linear system assembly (red), multigraph solver (green), mesh generation (magenta), a posteriori error estimation (cyan), parallel processing routines (yellow), and other $PLTMG$ routines (blue). A pie graph showing the fraction of total time spent in each of the six categories is drawn in the main frame. Details of individual contributions from the subroutines listed in Table 5.5 are summarized in the upper right frame. Sample output is shown in Figure 5.11.

When $MPISW = 1$, the times displayed for $IGRSW = \pm 2$ are time averaged across all processors. In this case, in the lower right frame, a graph displaying the deviation from the average time for each processor is drawn.

### 5.4.3 Continuation Path.

When $IGRSW = 3$, $GPHPLT$ displays the continuation path generated by the continuation procedure $IPROB = 3$. Target points are marked by small boxes, generally using different colors for different values of $ITASK$. A legend appears in the upper right frame summarizing the possibilities. Up to one hundred target points generated by calls to $PLTMG$ are saved and displayed. Successive points are interpolated using parabolic arcs matching the values of $(\lambda, \rho)$ and the tangent vectors $(\dot{\lambda}, \dot{\rho})$. In the lower right frame appears a convergence history for the most recent singular vector computation. Sample output is shown in Figure 5.12.

### 5.4.4 Parallel Statistics

When $IGRSW = -3$, $GPHPLT$ plots the functions

$$T_k = \log_2 \left\{ \frac{NPROC \cdot NTF(\Omega_k)}{\sum_k NTF(\Omega_k)} \right\} \quad \text{and} \quad E_k = \log_2 \left\{ \frac{NPROC \| \epsilon_t \|_{H^1(\Omega_k)}}{\sum_k \| \epsilon_t \|_{H^1(\Omega_k)}} \right\}$$

where $1 \leq k \leq NPROC$. Both curves appear in the large frame. When $MPISW = 1$ the information from all processors is obtained by an exchange of data using the MPI library. This is the most useful situation. When $MPISW = -1$, the same graph is made using local data on the given processor; this case is typically not interesting. In the upper right frame is a similar graph for the distribution of error and elements following the initial load balancing step ($IADAPT = 7$). In the lower right frame appear convergence histories for eigenvalue computations in the load balancing phase. Convergence histories are shown for the four most recent problems. Sample output is shown in Figure 5.12.

### 5.4.5 Error Estimates.

In the case $IGRSW = 4$, $GPHPLT$ graphs the function

$$\mathcal{F}_1(NDF, TIME) = \log_{10} \left\{ \frac{\| \epsilon_t \|_{H^1(\Omega)}}{\| u_h \|_{H^1(\Omega)}} \right\}$$
and in the case $IGRSW = -4$, $GPHPLT$ graphs the function

$$F_0(NDF, TIME) = \log_{10} \left\{ \frac{\| \epsilon_t \|_{L^2(\Omega)}}{\| u_h \|_{L^2(\Omega)}} \right\}.$$  

Here $\epsilon_t$ is the computed approximation of the error $u - u_h$. While it is hoped that these approximations accurately reflect the true state of affairs, the estimates are based on a posteriori calculations involving only the computed solution. Some judgment of the validity of such computations may be required. An example is shown in Figure 5.10.

Error estimates are plotted as a function of both $NDF$ and $TIME$. In particular, $F_j$ is graphed versus $\log_{10} NDF$ and $\log_{10} TIME$ in a three-dimensional graph. All data points (up to the 20 most recent) for which error estimates are available are marked with rectangular cylinders of different colors. A legend appears in the upper right frame summarizing the possibilities. In the case $IGRSW = 4$, the plot of $F_0$ appears in the lower right frame; if $IGRSW = -4$, the plot of $F_1$ in the lower right frame.

The triple $d = (MX, MY, MZ)$ specifies the viewing perspective for these graphs in a fashion similar to $(NX, NY, NZ)$ for surface plots. The choice $(1, -1, 1)$ is a reasonable default. The choice $(0, -1, 0)$ yields a traditional two-dimensional graph of $\log_{10} F_j$ versus $\log_{10} NDF$. The choice $(1, 0, 0)$ yields a two-dimensional graph of $\log_{10} F_j$ versus $\log_{10} TIME$. The main image can be animated using three dimensional imaging systems such as OpenGL.

### 5.4.6 Displaying Data Arrays.

The options $|IGRSW| \geq 5$, $GPHPLT$ displays the $IP$, $RP$, or $SP$ arrays. Unlike other graphics options, here the entire graphics window is treated as a single frame. In the case of the $IP$ and $RP$ arrays, all 100 entries, their names, and their current values are displayed. Entries that can be interactively reset in the $ATEST$ driver are colored red, unused entries appear in black, and all other entries are colored blue. This situation is similar for the $SP$ array, except only the first 50 entries are displayed (the remainder are all presently unused). Examples are shown in Figures 5.13–5.14.
<table>
<thead>
<tr>
<th>subroutine</th>
<th>main function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TGEN</td>
<td>create triangulation from skeleton</td>
</tr>
<tr>
<td>REFIN</td>
<td>adaptively refine the triangulation</td>
</tr>
<tr>
<td>UNREFN</td>
<td>adaptively unrefine the triangulation</td>
</tr>
<tr>
<td>HUNFRM</td>
<td>uniformly $h$-refine the triangulation</td>
</tr>
<tr>
<td>PUNFRM</td>
<td>uniformly $p$-refine the triangulation</td>
</tr>
<tr>
<td>MVEMSH</td>
<td>adaptively smooth the mesh points</td>
</tr>
<tr>
<td>ERREST</td>
<td>compute error estimates for $u_h$</td>
</tr>
<tr>
<td>CBUMP</td>
<td>recover derivatives for error estimation</td>
</tr>
<tr>
<td>CDLFN</td>
<td>compute dual function</td>
</tr>
<tr>
<td>CNORMS</td>
<td>compute global a posteriori error estimate</td>
</tr>
<tr>
<td>EXPTH</td>
<td>MPI exchange $IPATH$ data</td>
</tr>
<tr>
<td>BCASN</td>
<td>broadcast mesh to all processors</td>
</tr>
<tr>
<td>LDBAL</td>
<td>compute a load balance</td>
</tr>
<tr>
<td>LDEV</td>
<td>solve eigenvalue subproblem in load balance</td>
</tr>
<tr>
<td>CUTR</td>
<td>reorganize data structures for reconciling mesh</td>
</tr>
<tr>
<td>PASTE</td>
<td>reconcile mesh along interface of $IRGN$</td>
</tr>
<tr>
<td>PASTE1</td>
<td>reconcile mesh along interface not part of $IRGN$</td>
</tr>
<tr>
<td>TRIGEN</td>
<td>all other time spent in $TRIGEN$</td>
</tr>
<tr>
<td>SETGRB</td>
<td>compute block $JA$ array</td>
</tr>
<tr>
<td>SETGR2</td>
<td>compute DD interface $JA$ array</td>
</tr>
<tr>
<td>JA2JA</td>
<td>min degree / reordering</td>
</tr>
<tr>
<td>SFBILU</td>
<td>block $ILU$ factorization</td>
</tr>
<tr>
<td>SFHB</td>
<td>$HB$ matrix factorization</td>
</tr>
<tr>
<td>MG</td>
<td>solve equations using CSCG/CSBCG iteration</td>
</tr>
<tr>
<td>BLK3</td>
<td>solve equations for $IPROB = \pm 3$</td>
</tr>
<tr>
<td>BLK4</td>
<td>solve equations for $IPROB = \pm 4$</td>
</tr>
<tr>
<td>BLK5</td>
<td>solve equations for $IPROB = \pm 5$</td>
</tr>
<tr>
<td>LINSYS</td>
<td>assemble linear system</td>
</tr>
<tr>
<td>RGNSYS</td>
<td>assemble linear system for DD</td>
</tr>
<tr>
<td>CEV</td>
<td>compute the singular value $\mu$ and vectors $\psi_r$ and $\psi_\ell$</td>
</tr>
<tr>
<td>SWBRCH</td>
<td>switch branches at a bifurcation point</td>
</tr>
<tr>
<td>PREDCT</td>
<td>compute the steplength $\sigma$ for continuation</td>
</tr>
<tr>
<td>TPICK</td>
<td>line search for Newton iteration</td>
</tr>
<tr>
<td>TPICKD</td>
<td>line search for Newton/DD iteration</td>
</tr>
<tr>
<td>PLTMG</td>
<td>all other time spent in $PLTMG$</td>
</tr>
</tbody>
</table>

**Table 5.5. Subroutines timed by GPHPLT.**
test driver

6.1 Overview.

Program ATEST is the test driver used in the development and testing of the PLTMG package. ATEST is a flexible program in that it accepts simple command strings directing it to call subroutines or perform other tasks. It is not limited to a fixed sequence of tasks on a particular run; any routine can be called as often as desired, with certain parameters reset for each call at the discretion of the user.

The program ATEST can operate in four modes, governed by the switch MODE. If MODE = −1, ATEST runs as an interactive program, accepting commands from the user via a terminal window. If MODE = 0, ATEST runs interactively, accepting commands from the user via a web browser interface. The communication link between ATEST and the web browser is a socket, so that the client web browser can be running on a different device from the server running ATEST. If MODE = 1, ATEST runs as a batch program, reading commands from a journal file and sending all output to appropriate output files. Finally, if MODE = −2, ATEST runs as a slave mode under MPI; this mode cannot be directly set by the user, but is set by ATEST if it determines that it is a slave node in a parallel computation. In this situation, the user specifies MODE only for the master node, which can be any of the three other options. The value of MODE is set in a parameter statement at the beginning of ATEST, and also can be set dynamically in the command line launch of an ATEST application as described in Section 6.6.

A common command syntax is used for all modes. This is described first for the case MODE = −1 in Section 6.2. The extensions used in the web browser interface are described in Section 6.3.

Several files are written by ATEST. The file BFILE contains a complete record of all commands and printed output produced during the session. The file JWFILE contains a record of all commands read and processed during the session, formatted as a journal file. See Section 6.9 for a discussion of journal files. ATEST also creates a temporary file JTFILE used in connection with the journal command. While most commands invoke one of the major routines in the package, there are
a few utility routines (e.g. for reading and writing files) which are documented in Sections 6.8–6.10.

6.2 Terminal Mode.

In terminal mode, commands are entered from a terminal window in character strings of 80 characters, counting blanks. The syntax of a command can take several forms, but the root command is always a single letter. The commands that are currently recognized by ATTEST are summarized in Table 6.1.

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>call PLTMG</td>
</tr>
<tr>
<td>t</td>
<td>call TRIGEN</td>
</tr>
<tr>
<td>f</td>
<td>call TRIPLT</td>
</tr>
<tr>
<td>g</td>
<td>call GPHPLT</td>
</tr>
<tr>
<td>i</td>
<td>call INPLT</td>
</tr>
<tr>
<td>r</td>
<td>read data set from a file</td>
</tr>
<tr>
<td>w</td>
<td>write data set to a file</td>
</tr>
<tr>
<td>u</td>
<td>call USRCMD</td>
</tr>
<tr>
<td>j</td>
<td>execute journal file</td>
</tr>
<tr>
<td>p</td>
<td>MPI toggle</td>
</tr>
<tr>
<td>q</td>
<td>quit</td>
</tr>
</tbody>
</table>

Table 6.1. Available commands for ATTEST.

The terminal window prompt is the string command:. At this prompt, one can enter a command string (e.g., s), reset parameters as described below, or enter a blank line to see a list of the available commands. In this latter case the terminal window will appear as follows.

```
command:
pltmg s trigen t triplt f gphplt g inplt i read r
write w usrcmd u journal j mpi p quit q
```

A syntax error in a given command string causes the entire string to be ignored. ATTEST will display the string command error and present the command prompt for a new input string.

The most simple commands are just single lower case letters as shown in Table 6.1. However, associated with most commands are various parameters which can be reset before calling the given routine. To see a listing of the parameters associated with a given command and their current values, without executing the command itself, enter the command in upper case at the command prompt. For example, the command F will display the parameters which can be interactively reset in connection with TRIPLT.
6.2. Terminal Mode.

There are fifteen integer parameters, five real parameters, one string parameter, and one file parameter affecting subroutine \textit{TRIPLT} that can be interactively reset by the user. To the right of each parameter is a one- or two-letter alias (to avoid typing long names), followed by the current value.

To reset some parameters associated with a command \(c\) (e.g., \(s, f, g\), etc.), without invoking the command itself, one can type a string of the form

\begin{verbatim}
command: C name1=value1, name2=value2, ... , namek=valuek
\end{verbatim}

Note that the root command appears in upper case. The \textit{namek} refer to variable names or their aliases, and \textit{valuek} refer to integer, real, or string values. Several parameters can be reset, with different entries separated by commas. Values for integer parameters should be integers, while values for real parameters can be specified using integer, fixed point, or exponential notation. There are three types of string parameters: short, long and file. Short strings are typically single words and can not contain any blank characters. Files are typically file names, and they also can not contain any blank characters. All other strings are long, and can contain any printable ASCII characters other than double quotes. Values of long string parameters should appear within double quotes. Short and file string parameters are not enclosed with double quotes. Blank spaces are ignored everywhere but within the value field of a long string parameter. A syntax error in the input line (e.g., a misspelled variable name) causes the entire command to be ignored and no variables to be reset. \textit{ATEST} will respond \textit{command error} and then ask for the next command. For example, here we reset \textit{ISCALE} = 1, \textit{NCON} = 20, \textit{CENX} = .3, \textit{RMAG} = 10, and \textit{FTITLE} = \textit{A new title for Circle}. Subroutine \textit{TRIPLT} is not called, but the parameters are updated and redisplayed as

\begin{verbatim}
command:F
ifun f 0 iscale s 1 lines l 0 numbrs n 0
idevce d 0 fpane p 0 ncon c 20 mxcolr mc 256
icont ic 0 nx nx 0 ny ny 0 nz nz 1
mpirgn mr 0 rmag m 10.0 cenx cx 0.3 ceny cy 0.5
smin sn 0.0 smax sx 0.0 icrsn cr 0 itrgt it 10000
ftitle t "A new title for Circle"
grfile gf circle_figxxx.ext
command:
\end{verbatim}

One can reset some parameters for a given command \(c\), and then invoke the command itself, using a string of the form

\begin{verbatim}
command:F
ifun f 0 iscale s 1 lines l 0 numbrs n 0
idevce d 0 fpane p 0 ncon c 20 mxcolr mc 256
icont ic 0 nx nx 0 ny ny 0 nz nz 1
mpirgn mr 0 rmag m 10.0 cenx cx 0.3 ceny cy 0.5
smin sn 0.0 smax sx 0.0 icrsn cr 0 itrgt it 10000
ftitle t "A new title for Circle"
grfile gf circle_figxxx.ext
command:
\end{verbatim}
command: c name1=value1, name2=value2, ... , namek=valuek

Note that the only difference is that the root command now appear in lower case rather than upper case. Thus

command: f s=1, ncon=20, cenx=.3, rmag=1.e1, t="A new title for Circle"

resets the indicated parameters as in the previous example. However, instead of displaying the updated values, subroutine TRIPLT is called.

Finally, the graphics and MPI commands (f, i, g, and p) have a short form allowing one crucial parameter (IFUN, INPLSW, IGRSW, and MPISW, respectively) to be reset without typing even the alias. For example,

command: f5

is the short form for

command: f ifun=5

The short and long forms of these commands cannot be mixed. Thus

command: f5, ncon=10

is not valid.

6.3 Web Browser Mode.

When MODE = 0, the driver ATEST connects to a web browser interface via socket WEBPRT. While the WEBGUI browser interface was written by Chris Deotte to control PLTMG, it was designed as a graphical user interface for any C or Fortran package. Complete documentation is provided in [47]; here we will mainly discuss its use in PLTMG. When an ATEST application is launched with MODE = 0, it displays a message of the form

webgui: Listening on port 15000 for web browser...

In this example WEBPRT = 15000. To establish a connection with a web browser running on the same computer as the ATEST application, type

http://localhost:15000

in the web browser’s address bar. If the web browser is running on a different client device, replace localhost with the IP address or the hostname of the computer running the ATEST application. Once the connection is established, the WEBGUI interface should appear in the the web browser. The value of WEBPRT is set in a parameter statement at the beginning of ATEST and also can be set dynamically in the command line launch of an ATEST application as described in Section 6.6.
6.3. Web Browser Mode.

The functional capabilities of the web browser interface are the same as those for terminal window mode, but the possibilities for data entry are more varied. An example of the web browser interface interface appears in Figure 6.1. The web browser display has four main elements. There are three graphics panes, and in the upper left is the main control panel. The graphics interface will be described in more detail in Section 6.12.4. Here we discuss the main control panel.

![Figure 6.1. The web browser interface.](image)

The control panel contains two elements. The upper portion of the display contains command buttons. The bottom portion of the display is the history window. The command buttons stand in one to one correspondence with the basic ATTEST command set shown in Table 6.1. To the right of each main command button is a smaller button labeled with a plus sign. Clicking on a command button is equivalent to the typed lower case version of the given command. For example, clicking on the button for the TRIPLT command causes subroutine TRIPLT to be called as in the command \( f \). Clicking on the plus button to the right of any command is equivalent to the typed upper-case version of that command. In this
case a drop down menu appears containing a display of all the parameters that can be modified by that command. For example, clicking on the plus button for the TRIPLT command causes the parameters for the TRIPLT command to be displayed as in the command $F$. This is shown in figure 6.2.

![Figure 6.2. Drop down menu for resetting parameters for TRIPLT.](image)

The parameters associated with a given command are displayed in the reset window in a format similar to terminal mode. However, each parameter value is displayed in one line text-editing window, and can be reset by typing in the new value. Some parameter names (e.g., IFUN in Figure 6.2) appear in a button. If the parameter is a switch with several options, clicking on the name button causes the drop down menu to expand and display a suite of radio buttons indicating the various options associated with the parameter. Clicking on the desired option causes the parameter to be reset to the corresponding value. The radio buttons associated with the parameter IFUN appear in Figure 6.3.

On the other hand, if the parameter is a file name, clicking on the name button causes the drop down menu to display a file selection widget. The file-
6.3. Web Browser Mode.

The history window displays the contents of the output file, BFILE, as it is created. When executing a journal file in web browser mode, if a graphics command is executed, depending on the graphics device selected, ATEST can pause after the picture is drawn, and create a small popup click to continue button. In this case, ATEST waits until the user dismisses the click to continue popup before continuing to execute the journal file. This allows time for the user to view the picture before processing the next command in the journal file.

The WEBGUI interface has several additional features that are useful in certain situations. These appear in Table 6.2. To use these features one presses the OPTION key (ALT key on Windows machines) while also pressing the indicated key. When OPTION + C are pressed, the command buttons disappear and are replaced with a command line where commands may be entered as in terminal mode. Facsimiles of the displays created in terminal mode appear in the history window.
Figure 6.4. File sector for the parameter RWFILE.

Pressing OPTION + C again displays the command buttons.

OPTION + W changes the display from its default 2 × 2 layout to a single column 4 × 1 layout, with the control panel at the top, followed by the three graphics panes. Pressing OPTION + W a second time returns to the 2 × 2 display.

OPTION + F toggles the firewall on and off. When the firewall is on, ATEST displays the message

webgui: Only accepting ip address = x.x.x.x

where x.x.x.x is the IP address of the client device running the web browser. The WEBGUI interface displays the message

FIREWALL ON: Only your ip address (x.x.x.x) can access webgui.

at the bottom left of the main display. When the firewall is turned off, ATEST displays the message

webgui: Accepting all ip addresses.
### 6.4 Batch Mode.

When $\textit{MODE} = 1$, the \textit{ATEST} driver runs as a batch program. All commands are read from the journal file specified in $\textit{JRFILE}$. Graphics output should be directed to files (PostScript and XPM) rather than to interactive displays.

### 6.5 Parallel Processing

When run as a parallel program using $\textit{NPROC}$ processors, \textit{ATEST} uses a master-slave model. One process, the master process, runs in terminal, web browser, or batch mode, and the remaining $\textit{NPROC} - 1$ slave processes all run with $\textit{MODE} = -2$. Slave nodes receive command strings from the master node via MPI communicatin.
cation. At any given time, the parallel computation is in one of two possible states that specify how slave nodes should process commands. Somewhat arbitrarily, the two states are denoted “off” and “on”. When MPI is on, all processors execute all commands from the user, whether entered interactively or through a journal file. When MPI is off, only the master process executes most commands. Slave nodes remain active and still receive and evaluate the command strings they receive. Some commands (namely $p$ and $q$) continue to be executed and some parameter updates continue on all processors in the off state. However, in the off state, slave nodes are mainly waiting for MPI to be turned on again.

The $p$ command is used to switch between the on and off states of MPI. When $MPISW = 1$, MPI is on, and when $MPISW = -1$, MPI is off. The $p$ command is unusual in that it can behave as a toggle; executing $p$ with no argument switches the MPI state. The $p$ command can also be employed in the usual way to explicitly set the MPI state using the parameter $MPISW$ (e.g., $p 1$ turns on MPI, while $p -1$ turns off MPI). The MPI command button in web browser mode is a bit unusual; when MPI is on, the MPI command button changes color. When MPI is off the MPI command button returns to its usual color.

A common and effective way to use MPI is to create a journal file that contains a script for the entire computation (including $p$ commands) The $j$ command issued in the MPI on state directs all processors to run the journal file. The master process will then execute the entire script, while the slave nodes execute the parts of the journal file that correspond to the on state.

An issue with respect to file names arises in the context of parallel processing. Some files, for example a journal file, are intended to be read by all processors. In other situations, for example writing data files, each processor is intended to process its own version of the file. Then name conflicts can potentially become catastrophic if all nodes read and write files on the same file system. To resolve this conflict in a simple way that allows the user to easily specify on a case-by-case basis if the file is a single file or a file with distinct copies on each node, $ATEST$ scans all file names, looking for the characteristic string $MPIXXX$. If found, this string is replaced by $MPI001$, $MPI002$, etc, where the integer part denotes the processor. Thus, for example if one sets

$$JRFILE = MYFILE.JNL$$

all nodes process the same file with the name $MYFILE.JNL$. If one sets

$$RWFILE = MYFILE_MPIXXX.RW$$

node one would process the file $MYFILE_MPI001.RW$, node two would process the file $MYFILE_MPI002.RW$, and so on.

### 6.6 Command Line Parameters

The parameters $MODE$ and $WEBPRT$ are given default values in $parameter$ statements at the beginning of $ATEST$. These two parameters can optionally be set
6.7 Array Dimensions and Initialization.

ATEST has seven labeled common blocks:

common /atest1/ip(100),rp(100),sp(100)
common /atest2/iu(100),ru(100),su(100)
common /atest3/mode,jnlsw,jnlr,jnlw,ibatch
common /atest4/jcmd,cmdtyp,list
common /atest5/idevce,ipane
common /atest6/nproc,myid,mpisw,mpirgn,mpiint,mpiflt
common /atest7/mlen,uname(300)

The IP, RP, and SP arrays are described in Section 2.7. The arrays IU, RU, and SU are not directly used by ATEST or any of the other routines. They are provided to the user for storing integer, real, and string parameters associated with a particular problem. The advantages in using these arrays are that they are saved and read in the w and r commands; the common block ATEST2 can be included in subroutines A1XY, A2XY, etc., where the parameters may be needed; and they can form part of the interface for resetting problem parameters using USRCMD. ATEST3 contains internal control parameters used by ATEST; each has a corresponding location in the IP array. ATEST4 contains string and integer variables that are used for internal communication among the user interface routines. The block ATEST5 contains integers specifying the current graphics output device, while ATEST6 contains parameters relevant to MPI. The block ATEST7 contains information needed to create the USRCMD parameter display in terminal and web browser modes.

The input data arrays ITNODE(5,MAXT), ITDOF(8,MAXT), E(MAXT,2), IBNDRY(7,MAXB), SF(2,MAXB), VX(MAXV), VY(MAXV), GF(MAXD,13), and IPATH(6,MAXPTH) are declared at the beginning of ATEST. The sizes of the
arrays, MAXT, MAXV, MAXD, MAXB, and MAXPTH, are specified at the beginning of ATEST using parameter statements; changing sizes to suit a particular computing environment or problem is thus a simple matter.

To use ATEST, the user must provide Fortran subroutines A1XY, A2XY, FXY, GNXY, GDXY, P1XY, P2XY, SXY, and QXY. Subroutines USRCMD and USRFIL should be provided, if only as a dummy routines. The user must also supply subroutine GDATA, in which the input arrays VX, VY, SF, ITNODE, and IBNDRY are specified, along with some parameters in IP, RP, SP, and possibly IU, RU, and SU. Other entries of the IP, RP, and SP arrays not required to be provided by the user through GDATA are given default values at the beginning of ATEST, but can be reset by the user as desired.

6.8 Reading and Writing Files.

The \texttt{w} and \texttt{r} commands are used to save and restore data sets. The arrays IP, RP, SP, IU, RU, SU, VX, VY, SF, IBNDRY, ITNODE, ITDOF, IPATH, E, and common blocks PLTMG6 and PLTMG7 are written to (\texttt{w} command) or read from (\texttt{r} command) the file RWFILE. Data files are formatted as binary files. The \texttt{w} and \texttt{r} commands can be used with both triangulation and skeleton data structures.

One can use the \texttt{w} and \texttt{r} commands to save and restore the solution at various points along a continuation path. One can also save solutions in the current run for post processing (graphics, etc.), which can then occur in a later run.

The parameter MPIRGN is also useful in this setting. When MPISW = 1 and MPIRGN=0, then a \texttt{r} or \texttt{w} command will cause all processors to read or write the specified file. However, if one would like to read or write a file on just one processor (e.g., for debugging), one can set MPISW = 1 and MPIRGN=I, for \(1 \leq I \leq NPROC\), and the file will be read or written only by processor \(I\).

6.9 Journal Files.

The \texttt{j} command causes ATEST to read its command strings from the file JRFILE, rather than accepting them interactively from the user. It is the only option available in batch mode. A journal file is an ASCII file containing a sequence of command strings as described in Section 6.2. The symbol \# appearing as the first character in a line causes that line to be interpreted as a comment. When the end of the file is reached ATEST returns to terminal or web browser mode and again accepts commands interactively. If a \texttt{q} command is encountered in a journal file, ATEST will exit.

6.10 Subroutine USRCMD.

The \texttt{u} command is used to call the user supplied routine USRCMD.

\begin{verbatim}
Call USRCMD( VX, VY, SF, ITNODE, IBNDRY, IP, RP, SP,
\end{verbatim}
This routine is written by the user to perform any tasks not covered by other commands. In our experience, the most frequent use of USRCMD has been to reset parameters unique to a particular problem.

USRCMD is affected by the variable IUSRSW. If IUSRSW = 0, the return from USRCMD causes ATEST to present the command prompt. If IUSRSW ≠ 0, the return from USRCMD results in a branch to the user supplied routine GDATA before presenting the command prompt. This switch is useful if modified parameters affect the geometry of the region, boundary conditions, etc., requiring modifications of the input arrays.

Since the most frequent use of USRCMD is to modify problem dependent parameters, we now describe how to build an interface within USRCMD allowing one to reset parameters in a fashion similar to the other commands. This is done via subroutine USRSET, that is called as follows:

\[
\text{Call USRSET( IU, RU, SU )}
\]

IU, RU, and SU are integer, real, and CHARACTER*80 arrays, respectively, of size 100 containing the parameters to be reset. It is often convenient to use the IU, RU, and SU arrays provided by ATEST in common block ATEST2 for this purpose.

In terminal mode, the command \textit{u} creates a display listing the user parameters and their current values, similar to the upper case form of other commands. Commands of the form

\text{command:u name1=value1, name2=value2, \ldots, namek=valuek}

reset the indicated parameters and then display the updated values. In web browser mode, pressing the + button associated with the the USRCMD command button creates a drop down menu as in the other commands.

To provide the data necessary for USRSET to properly update parameters and create the appropriate displays, the user must provide subroutine USRFIL, that is called as follows:

\[
\text{Call USRFIL( FILE, LEN )}
\]

FILE is a CHARACTER*80 array of length LEN, containing a list of commands that define the variables to be reset, and characterize the reset display. The commands in FILE have a syntax similar to the basic scripting language we have defined for ATEST itself. However, in this case there are just two basic commands: \textit{n} (name variable) and \textit{s} (string for a radio button). These are summarized in Table 6.3.

Note that integer variables are stored in the IU array, real variables in the RU array, and short, long and file strings are all stored as entries in the SU array. In order to correctly define the reset window, all four variables associated with the \textit{n} command should be defined in each \textit{n} command. Similarly, the three variables
Parameters associated with \texttt{n} command

<table>
<thead>
<tr>
<th>Name</th>
<th>Alias</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>vname</td>
<td>n</td>
<td>short</td>
<td>maximum of 6 characters</td>
</tr>
<tr>
<td>alias</td>
<td>a</td>
<td>short</td>
<td>maximum of 2 characters</td>
</tr>
<tr>
<td>vtype</td>
<td>t</td>
<td>short</td>
<td>i (int), r (real), s (short), l (long), f (file)</td>
</tr>
<tr>
<td>index</td>
<td>i</td>
<td>int</td>
<td>pointer to IU, RU, SU</td>
</tr>
</tbody>
</table>

Parameters associated with \texttt{s} command

<table>
<thead>
<tr>
<th>Name</th>
<th>Alias</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>vname</td>
<td>n</td>
<td>short</td>
<td>variable name</td>
</tr>
<tr>
<td>value</td>
<td>v</td>
<td>-</td>
<td>depends on vname</td>
</tr>
<tr>
<td>label</td>
<td>l</td>
<td>long</td>
<td>label associated with value in radio buttons</td>
</tr>
</tbody>
</table>

Table 6.3. Command syntax for \texttt{USRFIL}.

associated with the \texttt{s} command should all be defined in each \texttt{s} command. Otherwise, the syntax for each command follows the usual rules of the scripting language. Below is an example code fragment that could define a simple \texttt{FILE} array.

\begin{verbatim}
FILE(1) = 'N I=1, N=NTRI, A=NT, T=I'
FILE(2) = 'N I=2, N=IBC, A=BC, T=I'
FILE(3) = 'S N=IBC, V=1, L="NEUMANN BC"'
FILE(4) = 'S N=IBC, V=2, L="DIRICHLET BC"'
LEN = 4
\end{verbatim}

The first two lines are \texttt{n} commands that define two integer variables. The first line defines a variable with name \texttt{NTRI}, alias \texttt{NT}, that is stored as \texttt{IU(1)}. The second defines a variable \texttt{IBC}, alias \texttt{BC}, that is stored as \texttt{IU(2)}. The variable \texttt{IBC} can take on two values, 1 and 2, that are associated with Neumann and Dirichlet boundary conditions, respectively. The third and fourth lines above are \texttt{s} commands that define the structure of a radio box associated with the \texttt{IBC} name in the web browser interface. Note that since the \texttt{LABEL} is a long string, its value must be enclosed in double quotes.

6.11 Subroutine GDATA.

The user provides subroutine \texttt{GDATA}, which defines the region through an initial triangulation or a skeleton. A call to \texttt{GDATA} is among the first executable statements in \texttt{ATEST}.

\begin{verbatim}
Call GDATA( VX, VY, SF, ITNODE, IBNDRY, IP, RP, SP,
         IU, RU, SU, SXY )
\end{verbatim}

Through this call the user is minimally expected to supply values for NTF, NVF, and NBF in the IP array, as well as the relevant values for the input arrays VX, VY, SF, ITNODE, and IBNDRY. Entries in RP, SP, IU RU, and SU, as well as parameters in IP other than those mentioned above, may be optionally specified in GDATA.


During the initial installation of the package, the user must provide several machine dependent routines associated with timing and graphics. Default versions of these routines are provided with the package, which should work without modification in many environments, and in any event can serve as a model for a new implementation.

Fortran module MTHDEF, is used throughout the package to specify the precision of the arithmetic to be used. The timing routine TIMER is used by PLTMG and TRIGEN. The graphics routines TRIPLT, GPHPLT, and INPLT address the graphics output device through the routines PLTUTL, PFRAME, PLINE, and PFILL. These routines are documented in detail below.

6.12.1 Arithmetic Specification.

PLTMG now uses module MTHDEF to specify the precision of arithmetic to be used. In particular, PLTMG is no longer supplied in single and double precision versions, since either version as well as a quadruple precision version, can be created just by defining some parameters in MTHDEF.

Below appears the default version of MTHDEF.

```fortran
module mthdef
  integer(kind=4), parameter :: isngl=4
  integer(kind=4), parameter :: idble=8
  integer(kind=4), parameter :: rsngl=4
  integer(kind=4), parameter :: rdble=8
  integer(kind=4), parameter :: rquad=16
  integer(kind=4), parameter :: iknd=isngl
  integer(kind=4), parameter :: rknd=rdble
end module
```

The parameters RSNGL, RDBLE and RQUAD define single, double, and quadruple precision arithmetic, respectively. ISNGL and IDBLE defines standard and long integers. These definitions should work with no change on most systems. The parameter RKND can be set to RSNGL for a single precision version of the code, to RDBLE for a double precision version, or to RQUAD for a quadruple precision version. Note that nodes and weights for quadrature rules used by PLTMG, as well as some other constants used by the package, are specified with significant digits at the level of double precision. These constants are used in the case of quadruple precision arithmetic, but have only double precision accuracy. IKND can be set it ISNGL for standard or to IDBLE for long integer arithmetic.
6.12.2 Timing Routine.

Subroutine TIMER has the calling sequence

\[ \text{Subroutine \textit{TIMER}( ISW )} \]

Here \( ISW \) is an integer. The array \( TIME \) stored in common block \( \text{PLTMG7} \) records the time spent in major subroutines called by \( \text{PLTMG} \) and \( \text{TRIGEN} \). \( \text{TIMER} \) should call an appropriate system routine to determine the current time each time it is entered, and then take various actions depending on the value of \( ISW \). The cases \( ISW = -2 \) and \( ISW = -1 \) request initialization of the \( TIME \) array, while \( 1 \leq ISW \leq 50 \) request an individual entry in the \( TIME \) array be updated. The current time is saved as it is needed for the next call to \( \text{TIMER} \). Subroutine \( \text{TIMER} \) is machine independent except for the call to the system clock. An example of \( \text{TIMER} \), calling the function \( \text{CPU\_TIME} \), is given below. It is quite likely that this routine will function properly on most modern systems with no change.

```
subroutine timer(isw)

use mthdef
implicit real(kind=rknd) (a-h,o-z)
implicit integer(kind=iknd) (i-n)
integer(kind=iknd), save :: len=50
real(kind=rknd), save :: tx=0.0e0_rknd
common /pltmg7/time(3,50),hist(22,30)

call the clock and return the time in seconds
(time differences are used to compute the elapsed time)
ty=tx
call cpu_time(tx)

update time array (1.0e-10_rknd is below resolution of timer)
if(isw>0) then
  dt=max(tx-ty,1.0e-10_rknd)
time(1,isw)=time(1,isw)+dt
time(2,isw)=time(2,isw)+dt
else if(isw=-1) then
  do i=1,len
    time(i,1)=0.0e0_rknd
  enddo
else if(isw=-2) then
  do i=1,len
    time(i,1)=0.0e0_rknd
time(2,i)=0.0e0_rknd
time(3,i)=0.0e0_rknd
  enddo
endif
return
end
```

6.12.3 Command Line Interface

ATEST calls the Fortran subroutine

\[ \text{Subroutine get\_command( LIST, LENGTH, STATUS )} \]

to obtain a copy of the command line that launched an ATEST application. If this is not available in the Fortran compiler, one can comment this call, that appears only in subroutine ACMD. This disables the dynamic setting of the parameters MODE and WEBPRT but does not affect other aspects of the application.


The four device dependent routines in the graphics package are

\[ \text{Subroutine PLTUTL( NCOLOR, RED, GREEN, BLUE )} \]
\[ \text{Subroutine PFRAME( IFRAME )} \]
\[ \text{Subroutine PLINE( X, Y, Z, N, ICOLOR )} \]
\[ \text{Subroutine PFILL( X, Y, Z, N, ICOLOR )} \]

Subroutine PLTUTL takes various actions depending on the value of the integer NCOLOR. NCOLOR > 0 specifies initialization; NCOLOR denotes the number of colors to be used and satisfies \( 2 \leq \text{NCOLOR} \leq \text{MXCOLR} \). RED, GREEN, and BLUE are vectors of length NCOLOR. The entries RED\((i)\), GREEN\((i)\), and BLUE\((i)\), \(1 \leq i \leq \text{NCOLOR} \), are floating point numbers on the interval \([0, 1]\), corresponding to rgb values for the \(i\)th color. Color number 1 is always white \((\text{RED}(1) = \text{GREEN}(1) = \text{BLUE}(1) = 1.0)\), and color number 2 is always black \((\text{RED}(2) = \text{GREEN}(2) = \text{BLUE}(2) = 0.0)\). The rgb values of the remaining entries depend on the picture to be drawn and the value of MXCOLR. PLTUTL should create a color map with the required colors, as these will be referenced in future calls to PLINE and PFILL. If PLTUTL is called with NCOLOR < 0, the drawing is complete and any necessary post processing should be carried out (e.g., close the plot file).

The drawing space used by the graphics routines is always assumed to be either the unit square \((0, 1) \times (0, 1)\) or the rectangle \((0, 1.5) \times (0, 1)\). For devices that have a so-called Z-buffer, the drawing space is either the unit cube \((0, 1) \times (0, 1) \times (0, 1)\) or the brick \((0, 1.5) \times (0, 1) \times (0, 1)\). The graphics display itself is always viewed as rectangular with aspect ratio 3/2, which is either a single rectangular frame or three square frames. These frames are numbered 1 to 4 as illustrated in Figure 6.5. The graphics routines write their output to various lists. A list consists of a frame, and the attributes rotating/non-rotating. This attribute may not have realizations for all graphics devices. The five available lists are summarized in Table 6.4.

When graphics is initiated for a certain list, say list \(k\), subroutine PFRAME\((k)\) is called to indicate that subsequent calls of PLINE and PFILL contain data to be written to list \(k\). PFRAME\((−k)\) indicates that the output to the given list should be terminated. By convention, graphics routines are allowed only one open list at
a time. Therefore, when \texttt{PFRAME} is invoked with a positive argument, the given list should be opened and the mapping from the unit cube or brick to the actual device coordinates for the given list should be computed. If the rotation attribute is available, it should be set as specified in Table 6.4. When \texttt{PFRAME} is invoked with a negative argument, the given list should be closed.

![Figure 6.5. Frame definitions.](image)

<table>
<thead>
<tr>
<th>list</th>
<th>frame</th>
<th>rotating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>no</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 6.4. \textit{list specifications for pframe.}

Subroutine \texttt{PLINE} has arguments \(X, Y, Z, N,\) and \texttt{ICOLOR}. \(X, Y,\) and \(Z\) are vectors of length \(N \geq 2.\) The points \((X(i),Y(i),Z(i))\) lie in the unit cube or the brick \((0,1.5) \times (0,1) \times (0,1)\). The \(Z\) coordinate is useful only for devices that have a \(Z\)-buffer, and can be ignored in other cases. \texttt{ICOLOR} is an integer between 1 and \texttt{NCOLOR}, where \texttt{NCOLOR} was the argument that initialized \texttt{PLTUTL}, indicating the color to be used. \texttt{PLINE} should draw the given polyline \((X(i),Y(i),Z(i))\) to \((X(i+1),Y(i+1),Z(i+1))\), \(1 \leq i \leq N - 1,\) with the specified color in the proper frame.

Subroutine \texttt{PFILL} has arguments \(X, Y, Z, N,\) and \texttt{ICOLOR}. \(X, Y,\) and \(Z\) are vectors of length \(N \geq 3.\) The points \((X(i),Y(i),Z(i))\) lie in the unit cube or the brick \((0,1.5) \times (0,1) \times (0,1),\) and define a \(N\)-sided (planar) polygonal region with sides \((X(i),Y(i),Z(i))\) to \((X(i+1),Y(i+1),Z(i+1))\) for \(1 \leq i \leq N - 1,\) and \((X(N),Y(N),Z(N))\) to \((X(1),Y(1),Z(1))\). \texttt{ICOLOR} is an integer between 1 and \texttt{NCOLOR}, where \texttt{NCOLOR} was the argument that initialized \texttt{PLTUTL}, indicating the color to be used. \texttt{PFILL} should color the specified polygon with the specified color in the proper frame.

The default installation of the \texttt{PLTMG} package includes several output graphics devices. These are described in Table 6.5. PostScript and XPM are both ASCII files. The parameter \texttt{GRFILE} specifies the file name. To allow one file name vari-

<table>
<thead>
<tr>
<th>IDEVCE</th>
<th>output device</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>web browser</td>
</tr>
<tr>
<td>1</td>
<td>XPM file</td>
</tr>
<tr>
<td>2</td>
<td>PostScript file</td>
</tr>
</tbody>
</table>

Table 6.5. Default graphics devices.

able to stand for multiple files, the file name GRFILE is scanned for the string FIGXXX. If it is found, for the first actual file name, it is replaced by FIG001, for the second by FIG002, and so on, allowing each actual file to receive a unique name. The file name is also scanned for the string .EXT and if found, it is replaced by .XPM or .PS as appropriate. If the string .EXT is not found, the appropriate extension is appended to the existing name. Finally, the file name is scanned for the string MPIXXX, and if found, the XXX is replaced by the appropriate processor number.

<table>
<thead>
<tr>
<th>IPANE</th>
<th>WEBGUI canvas</th>
<th>graphics type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>upper right</td>
<td>WebGL 3D</td>
</tr>
<tr>
<td>1</td>
<td>lower left</td>
<td>WebGL 3D</td>
</tr>
<tr>
<td>2</td>
<td>lower right</td>
<td>WebGL 3D</td>
</tr>
<tr>
<td>3</td>
<td>upper right</td>
<td>Pixmap 2D</td>
</tr>
<tr>
<td>4</td>
<td>lower left</td>
<td>Pixmap 2D</td>
</tr>
<tr>
<td>5</td>
<td>lower right</td>
<td>Pixmap 2D</td>
</tr>
</tbody>
</table>

Table 6.6. IPANE definitions.

The web browser interface provides both 2D and 3D graphics. The canvas and the type of graphics are specified through the parameter IPANE, as described in Table 6.6. The parameters FPANE, JPANE, and GPANE are associated with graphics subroutines TRIPLT, INPLT, and GPHPLT, respectively, and specify the value of IPANE to be used with that routine.

WebGL graphics written to LIST = 5 can be manipulated with the mouse. Mouse button one (left) can be used to rotate the image. Button two (middle) is used to pan the image, and button three (right) can be used to zoom. On Apple systems with a one button mouse, buttons two and three are simulated by pressing the OPTION and COMMAND keys, respectively, simultaneously with the mouse. In WebGL windows, the OPTION + I toggle referenced in Table 6.2 can display the numerical values for the zoom, rotation matrix and translation.

Additionally, associated with each WebGL canvas are seven buttons. These are described in Table 6.7.

The RESET button returns the image to its initial state, removing any pan, zoom and rotation operations that the user may have applied. OPTION + R
<table>
<thead>
<tr>
<th>button</th>
<th>effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET</td>
<td>return a zoomed/panned/rotated image to its initial state</td>
</tr>
<tr>
<td>FREE</td>
<td>clear the image and free associated memory</td>
</tr>
<tr>
<td>SPIN</td>
<td>spin image</td>
</tr>
<tr>
<td>POP</td>
<td>place canvas in its own web browser tab</td>
</tr>
<tr>
<td>PUSH</td>
<td>return canvas to its original location in the web browser tab</td>
</tr>
<tr>
<td>SAVE</td>
<td>save image to a file</td>
</tr>
<tr>
<td>LOAD</td>
<td>read image from a file</td>
</tr>
</tbody>
</table>

Table 6.7. WebGL graphics window buttons.

This toggles between automatically invoking the RESET button each time a new image is displayed, and retaining the pan, zoom and rotate position of the previous image.

When the POP button is pushed, the graphics window is removed from the array of graphics windows in the main web browser display, and appears in a separate tab. This is useful, as typically the size of the image increases. In the new tab, the POP button is replaced by one labeled PUSH. When the PUSH button is pressed, the canvas is returned to its previous location in the main display. The SAVE button saves a binary file containing the image, which can later be restored using the LOAD button. If the OPTION key is pressed simultaneously with the SAVE button as indicated in Table 6.2, then the image is saved as an ASCII file.

If $3 \leq IPANE \leq 5$ then only the FREE button appears. These values of IPANE are useful to preview pixmaps, that can then later be saved as XPM files.

6.12.5 MPI Interface

The communication used in parallel processing is provided by calls to the MPI library. This library is not provided as part of the PLTMG package. The calls to the MPI library are all made from Fortran, and we have concentrated all calls into just a few subroutines. Thus the vast majority of the code comprising the main PLTMG routines is self-contained. If the MPI library is not available, one can use substitute stub routines supplied with the default installation in place of the regular interface. The stub routines are similar to the those in the regular interface, except that all calls to routines and functions in the MPI library have been deleted. Using the stub routines, an executable can be created without the need to load the MPI library to resolve external references. In this case, all the parallel computing options provided by PLTMG, TRIGEN, and the graphics routines are disabled.
Chapter 7
Test Problems

7.1 Overview.
In this chapter, we briefly document the test problem data sets included with the PLTMG source code. These problems encompass a variety of applications and exercise most features of the package. Each data set minimally consists of functions \(A1XY\), \(A2XY\), \(FXY\), \(GNXY\), \(GDXY\), \(P1XY\), \(P2XY\), and \(QXY\) and subroutines \(USRCMD\) and \(GDATA\). Problem specific routines are also included.

7.2 Test Problem CIRCLE.
In this problem, we solve the equation

\[-\nabla \cdot (a \nabla u) = 0,\]

where \(\Omega\) is the unit circle with a crack along the positive \(x\) axis. Homogeneous Dirichlet boundary conditions are imposed on the top of the crack, and homogeneous Neumann boundary conditions are imposed below the crack. The coefficient \(a \equiv a_k\) is piecewise constant in the eight sectors

\[\Omega_k = \{(r, \theta)| 0 \leq r \leq 1, (k-1)\pi/4 \leq \theta \leq k\pi/4\}.\]

The domain \(\Omega\) is defined by a triangulation consisting of eight similar triangles, shown in Figure 7.1, that correspond to the eight sectors of constant \(a\). On the boundary of the circle, nonhomogeneous boundary conditions are imposed such that the true solution in sector \(\Omega_k\) is given by

\[u = r^\alpha (\beta_k \sin \alpha \theta + \gamma_k \cos \alpha \theta).\] (7.1)

The exponent \(\alpha\) is chosen to correspond to the leading singularity arising from the geometry, change of boundary conditions, and coefficient jumps at the origin. The coefficients \(\beta_k\) and \(\gamma_k\) are chosen to insure continuity of the solution \(u\) and the normal component of the flux \(a \nabla u \cdot n\) across the interfaces, and to satisfy the
boundary conditions along the crack. For example, in the case $a_k = 1$ for all $k$, $\alpha = 1/4$ and

$$u = r^\alpha \sin \alpha \theta.$$ 

The \textit{USRCMD} for this test problem has ten parameters that can be set. \textit{IBC} determines the boundary conditions. If $IBC = 2$, the boundary conditions on the outer boundary of the circle are nonhomogeneous Dirichlet chosen such that (7.1) is the exact solution; if $IBC = 1$, nonhomogeneous Neumann boundary conditions are imposed on the circular part of the boundary in a similar fashion. One can also alter the geometry of the domain using the parameter \textit{NTRI}, where $1 \leq \text{NTRI} \leq 8$. If \text{NTRI} = 8 the entire circle is used as the domain; if \text{NTRI} < 8, only the first \text{NTRI} sectors are used. Some examples are shown in Figure 7.1. Eight parameters,

![Figure 7.1](image)

\textbf{Figure 7.1.} \textit{On the far right is the square domain for problems SQUARE, OB and CONTROL. The remaining domains are for test problem CIRCLE with NTRI = 8, NTRI = 7 and NTRI = 3.}

$A_1, A_2, \ldots, A_8$ define the coefficients $a_k$. Given the $a_k$ and \textit{NTRI}, the values of $\alpha, \beta_k$ and $\gamma_k$ are computed in \textit{GDATA} by solving appropriate nonlinear equations. Since the exact solution is known, we can compute the exact error. For this test problem, the function $QXY$ is defined to be the exact error for graphics options and the true solution (7.1) otherwise.

### 7.3 Test Problem SQUARE.

In this test problem, a complicated equation is solved on a simple domain. The domain is always the unit square shown in Figure 7.1; boundary conditions on each side of the square can be independently specified as Dirichlet or natural, or pairs of opposite sides can be specified as periodic. The region is specified as a triangulation.
The coefficient functions are defined by

\[ a_1 = A1X \frac{\partial u}{\partial x} + A1Y \frac{\partial u}{\partial y} + A1U u, \]

\[ a_2 = A2X \frac{\partial u}{\partial x} + A2Y \frac{\partial u}{\partial y} + A2U u, \]

\[ f = -BUX \frac{\partial u}{\partial x} - BUY \frac{\partial u}{\partial y} - CU0 - CU1 u - CU2 u^2 - CAHN (u - u^3) \]

\[ - CIR \left( \frac{\partial u}{\partial x} (y - .5) - \frac{\partial u}{\partial y} (x - .5) \right) - CEXP e^u - CSIN \sin u, -F0(y - x) \]

\[ g_1 = -DU0 - DU1 u, \]

\[ g_2 = -EU0, \]

and the functional \( \rho \) is defined by

\[ p_1 = u^2, \]

\[ p_2 = 0. \]

All of these nineteen parameters can be set using USRCMD, and any can be used as the continuation parameter \( \lambda \) by specifying the parameter \( \text{ICONT} \) in USRCMD as in Table 7.1. With this variety of nonlinearities, one can exercise most continuation features of PLTMG. If \( \text{ICONT} = 0 \), then none of the parameters is regarded as \( \lambda \), and one should set \( \text{IPROB} = 1 \) to signify that the problem does not involve continuation.

One can also set the integer parameters \( \text{LEFT}, \text{RIGHT}, \text{TOP}, \) and \( \text{BOTTOM} \) in USRCMD. These refer to the four sides of the square in an obvious fashion and can be individually set to 2 for Dirichlet boundary conditions or to 1 for natural boundary conditions for the given side of the square. A pair of opposite edges can be set to 0 (e.g., \( \text{TOP} = \text{BOTTOM} = 0 \)), and \( \text{IBNDRY} \) will then be set for periodic boundary conditions.

<table>
<thead>
<tr>
<th>( \text{ICONT} )</th>
<th>( \lambda )</th>
<th>( \text{ICONT} )</th>
<th>( \lambda )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>none</td>
<td>10</td>
<td>CU1</td>
</tr>
<tr>
<td>1</td>
<td>A1X</td>
<td>11</td>
<td>CU2</td>
</tr>
<tr>
<td>2</td>
<td>A1Y</td>
<td>12</td>
<td>CAHN</td>
</tr>
<tr>
<td>3</td>
<td>A1U</td>
<td>13</td>
<td>CEXP</td>
</tr>
<tr>
<td>4</td>
<td>A2X</td>
<td>14</td>
<td>CIR</td>
</tr>
<tr>
<td>5</td>
<td>A2Y</td>
<td>15</td>
<td>CSIN</td>
</tr>
<tr>
<td>6</td>
<td>A2U</td>
<td>16</td>
<td>DU0</td>
</tr>
<tr>
<td>7</td>
<td>BUX</td>
<td>17</td>
<td>DU1</td>
</tr>
<tr>
<td>8</td>
<td>BUY</td>
<td>18</td>
<td>EU0</td>
</tr>
<tr>
<td>9</td>
<td>CU0</td>
<td>19</td>
<td>F0</td>
</tr>
</tbody>
</table>

Table 7.1. Possible settings for \( \text{ICONT} \).
7.4 Test Problem DOMAINS.

In this test problem, a simple equation is solved on a variety of complicated domains. This test problem was designed mainly to exercise TRIGEN.

The problem to be solved is the linear partial differential equation

\[
\begin{align*}
a_1 &= A1X \frac{\partial u}{\partial x} + A1Y \frac{\partial u}{\partial y} + A1U u, \\
a_2 &= A2X \frac{\partial u}{\partial x} + A2Y \frac{\partial u}{\partial y} + A2U u, \\
f &= -BUX \frac{\partial u}{\partial x} - BUY \frac{\partial u}{\partial y} - CU1 u - CU0
\end{align*}
\]

with a combination of homogeneous Dirichlet, homogeneous Neumann, and periodic boundary conditions. The parameters \(A1X, A1Y, A1U, A2X, A2Y, A2U, BUX, BUY, CU0,\) and \(CU1\) can all be set in USRCMD. The parameter \(DOMAIN\), satisfying \(1 \leq DOMAIN \leq 21\), specifies the domain to be used. The various possibilities are shown in Figure 7.2. All domains are defined by skeletons, so TRIGEN must be called to generate a triangulation.

7.5 Test Problem NACA.

Test problem NACA solves the equation in one of several domains. The equation is of the form

\[
-\nabla \cdot (\theta \rho(\nabla u) + (1 - \theta))\nabla u = 0,
\]

where

\[
\rho(\nabla u) = (1 - u_x^2 - u_y^2)^{\frac{1}{\gamma - 1}}
\]

and \(\gamma = 1.4\). When \(\theta = 1\), this is the equation of potential flow, and \(\theta = 0\) corresponds to the Laplace equation. The local Mach number is computed in QXY and is given by

\[
q = \sqrt{\frac{2c}{\gamma - 1}}, \\
c = \frac{1}{1 - u_x^2 - u_y^2} - 1.
\]

There are four domain options, chosen using the parameter \(DOMAIN\) in USRCMD. These domains are shown in Figure 7.3. All regions are defined as skeletons, so TRIGEN must be used to generate a triangulation.

Neumann boundary conditions are imposed everywhere so each domain has \(ISING = 1\). There are several parameters in USRCMD that affect these problems. The parameter \(MINF\), specifying the Mach number at infinity \(M_\infty\), sets the boundary conditions on the outer boundary. The parameter \(ANGLE\) specifies the angle of attack (in degrees). The parameter \(SIZE\) sets the radius of the outer boundary. The parameter \(THETA\) sets the value of \(\theta\) defined above. When the local Mach number is less than one the flow is subsonic; PLTGM will work well in regions where the
Figure 7.2. The domains for $\text{DOMAIN} = i$, $1 \leq i \leq 23$. 
flow is entirely subsonic. As the $M_\infty$ is increased, the solution will begin to develop regions of supersonic flow near the airfoils; PLTMG will continue to work as these regions are forming, but eventually will fail, as the underlying discretization used by PLTMG is not really appropriate for hyperbolic problems.

### 7.6 Test Problem JCN.

Test problem JCN solves the convection diffusion equation

$$-\nabla \cdot (\nabla u + \beta u) = 0,$$

where $\beta$ is piecewise constant. The region is shown in Figure 7.4. The domain is specified by skeleton, so TRIGEN must be used to generate a triangulation.

This problem is an idealized model of the current continuity equation from the semiconductor device model that we have used to study the stability of discretizations used in device simulation. The problem has seven regions; $\beta = 0$ in regions one and seven. In the other five regions it has a magnitude of approximately $10^4$ and is directed radially in each of the five subregions. The solution develops steep gradients at the junction between region seven and the five adjoining subregions.

Constant nonhomogeneous Dirichlet boundary conditions are specified along the bottom of the domain and on the left-hand portion of the top of the domain. Homogeneous Neumann boundary conditions are imposed elsewhere. The parameters TOP and BOTTOM in USRCMD can be used to reset the Dirichlet boundary conditions.
conditions on the top and bottom of the domain. The parameter $DU$ can be used to adjust the size of $\beta$ in regions 2–5; in particular, the magnitude of $\beta$ in these five regions is proportional to $DU$.

Our original purpose in constructing this example was to test the sensitivity of various upwinding techniques [7] to poor element geometries. Since the goal of TRI-GEN is to produce elements with good geometries, the USRCMD for this problem includes a procedure for systematically degrading the quality of the triangulation by introducing new elements with obtuse angles. If $OBTUSE = 1$ in USRCMD, then each triangle in the current mesh is divided into three new triangles by connecting its barycenter to its vertices. An example is shown in Figure 7.4. Repeated application of this procedure will produce triangulations with interior angles arbitrarily close to $\pi$.

### 7.7 Test Problem OB.

Test problem $OB$ solves the a simple obstacle problem, with coefficient functions defined by

$$p_1 = AX \left( \frac{\partial u}{\partial x} \right)^2 + AY \left( \frac{\partial u}{\partial y} \right)^2 + CU u^2 - 2su,$$

$$s = (AX(IX\pi)^2 + AY(IY\pi)^2 - CU) \sin(IX\pi) \sin(IY\pi),$$

$$u = BDLW + CFLW \sin(IXL\pi) \sin(IYL\pi),$$

$$\bar{u} = BDUP + CFUP \sin(IXU\pi) \sin(IYU\pi),$$

$$g_1 = 0.$$

The domain $\Omega$ is the unit square with homogeneous Dirichlet boundary conditions. The input data structure is a triangulation consisting of eight right triangles, shown in Figure 7.1. The parameters $AX$, $AY$, $CU$, $BDLW$, $BDUP$, $CFLW$, $CFUP$ and the integers $IX$, $IY$, $IXL$, $IYL$, $IXU$, $IYU$ can all be set by the user in USRCMD. The exact solution to this problem in the absence of the obstacle is $u = \sin(IX\pi) \sin(IY\pi)$. This problem is mainly designed to test the cases $IPROB = \pm 2$ in PLTMG.
7.8 Test Problem MNSURF.

Test problem MNSURF solves the a simple minimal surface problem with an obstacle. The coefficient functions are given by

\[ p_1 = \sqrt{1 + \left( \frac{\partial u}{\partial x} \right)^2 + \left( \frac{\partial u}{\partial y} \right)^2} \]

\[ u = \begin{cases} 
1 & \text{in } \Omega_1 \\
-1 & \text{in } \Omega_2 \\
\ell & \text{in } \Omega_I
\end{cases} \]

\[ \bar{u} = 1.5 \]

\[ g_1 = 0, \]

\[ g_2 = 0. \]

The domain \( \Omega \) is the unit square with a mixture of homogeneous Dirichlet and Neumann boundary conditions. The domain is given as a skeleton, and is shown in Figure 7.5. The region \( \Omega_1 \) is the inner square with side 1/2, and \( \Omega_2 \) is the outer region. The region \( \Omega_I \) is the small band separating \( \Omega_1 \) and \( \Omega_2 \), consisting of four narrow trapezoids. In each of the four trapezoids, \( u \) is a linear polynomial in \( x \) or \( y \) that interpolates between \(-1\) and \(1\), insuring continuity of \( u \). The parameter \( \text{THETA} \), which can be set in \( \text{USRCMD} \), controls the width of the band. The upper bound \( \bar{u} \) is chosen such that it does not affect the solution. As with test problem \( \text{OB} \), this problem is mainly designed to test the cases \( \text{IPROB} = \pm 2 \) in \( \text{PLTMG} \).

7.9 Test Problem BURGER.

Test problem BURGER solves the nonlinear convection dominated flow

\[-\epsilon \Delta u + u_y + uu_x = 0.\]

This is really a time dependent Burger’s equation in one space dimension. In this setting, the \( y \) space variable plays the role of time, and we have added a small diffusion term. This problem is solved as a two dimensional steady state problem. Some
analysis of this approach to solving time dependent partial differential equations is given in [34].

The small parameter $\varepsilon > 0$ and can be set in USRCMD. The domain $\Omega$ is the quarter circle shown in Figure 7.5, and is specified as a skeleton. Natural (homogeneous Neumann) boundary conditions are applied along the circular arc, while Dirichlet boundary conditions are specified on the left side ($x = 0$) and the bottom ($y = 0$) as

$$g_2 = \begin{cases} 1 & 0 \leq x \leq 1/4 \\ 3/2 - 2x & 1/4 \leq x \leq 3/4 \\ 0 & 3/4 \leq x \leq 2 \end{cases}.$$  

This combination of boundary conditions gives rise to a solution similar to the so-called “$\lambda$ shock” of Burger’s equation.

### 7.10 Test Problem BATTERY.

In this test problem we solve the linear elliptic problem

$$-a_1 u_{xx} - a_2 u_{yy} - f = 0$$

where the piecewise constant values of the coefficients are given in Table 7.2. The domain $\Omega$ is shown in Figure 7.5 and is specified as a skeleton. The five subregions are given labels in ITNODE(5,*), allowing us to conveniently define the coefficient functions. The boundary conditions are natural boundary conditions of the form

$$g_1 = c - \alpha u.$$  

Here $c$ and $\alpha$ are piecewise constant functions defined using IBNDRY(6,*), as indicated in Table 7.2. The data for this problem was supplied by Leszek Demkowicz.

### 7.11 Test Problem CONTROL.

This problem tests the cases IPROB = $\pm 5$. The differential equation (constraint) is

$$-\Delta u = \lambda(C0 + C1 u + C2 u^2 + C3 u^3) + F0 + F1 u + F2 u^2 + F3 u^3$$
in $\Omega$, with Dirichlet boundary conditions

$$u = DBC$$

on $\partial \Omega$. The objective function $\rho$ is given by

$$\rho(u, \lambda) = \int_{\Omega} (u - u_0)^2 + \beta|\nabla (u - u_0)|^2 + \gamma \lambda^2 \, dx.$$ 

$\Omega$ is the unit square, defined as a triangulation similar to test problem SQUARE; see Figure 7.1. The function $u_0$ and the bounds on $\lambda$ are given by

$$u_0 = \sin(IX \pi x) \sin(IY \pi y),$$

$$BDLW \leq \lambda \leq BDUP.$$ 

The constants $BETA = \beta$, $GAMMA = \gamma$, $BDLW$, $BDUP$, $DBC$, $C0$, $C1$, $C2$, $C3$, $F0$, $F1$, $F2$, and $F3$, and the integers $IX$ and $IY$ can all be reset in USRCMD.

### 7.12 Test Problem IDENT.

This problem tests the cases $IPROB = \pm 4$. The differential equation is

$$-(1 + A^2)\Delta u + C2 u^2 + C1 u - C0 = 0.$$ 

The domain $\Omega$ is specified as a skeleton, and is shown in Figure 7.5. The boundary conditions are a combination of homogeneous Neumann and Dirichlet, except for the vertical edge on the right where the (possibly) nonhomogeneous Dirichlet boundary condition

$$u = D$$

is imposed. The five parameters $A$, $C0$, $C1$, $C2$, and $D$ can be set in USRCMD, and any combination of them can be used as scalar parameters $\lambda_i$ in the optimization problem. This is done setting the switches $IRL$ (also set in USRCMD) indicated in Table 7.3. For example, setting $IRL1 = 1$ makes $A$ an optimization parameter, while setting $IRL1 = 0$ keeps $A$ at its current fixed value.

<table>
<thead>
<tr>
<th>switch</th>
<th>$\lambda_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRL1</td>
<td>$A$</td>
</tr>
<tr>
<td>IRL2</td>
<td>$C0$</td>
</tr>
<tr>
<td>IRL3</td>
<td>$C1$</td>
</tr>
<tr>
<td>IRL4</td>
<td>$C2$</td>
</tr>
<tr>
<td>IRL5</td>
<td>$D$</td>
</tr>
</tbody>
</table>

**Table 7.3. IRL switches.**

The objective function $\rho$ is given by

$$\rho(u, \lambda) = \int_{\Omega} e^{-20(x^2+y^2)}(u - 1)^2 \, dx,$$

which tries to make the solution $u = 1$ near the origin, located at the center of $\Omega$. 
7.13 Test Problem BOX.

Test problem BOX tests the moving boundary optimization option in PLTMG \((IPROB = 4 \text{ and } ITASK = 8)\). The domain is a \(1 \times 1\) square with a square hole, illustrated in Figure 7.6. The hole is free to move around within the square, with its position governed by three parameters: \((x_c, y_c)\) denoting the coordinates of the center of the square, and \(\theta\) denoting its angle of rotation. Any combination of these three parameters may be chosen as optimization parameters, as summarized in Table 7.4.

\[
\begin{array}{c|c}
\text{switch} & \lambda_j \\
\hline
IRL1 & x_c \\
IRL2 & y_c \\
IRL3 & \theta \\
\end{array}
\]

\textbf{Table 7.4. IRL switches.}

The objective function is given by

\[
\rho(u, \lambda) = \min \int_\Omega \nabla u^2 + \delta \sum_{i=1}^{NRL} \lambda_i^2 \, dx.
\]

The boundary value problem and inequality constraints are given by

\[
\begin{align*}
-\Delta u &= 1 & \text{in } \Omega \\
u &= 0 & \text{on } \partial \Omega \\
\lambda_i &\leq \lambda_i \leq \bar{\lambda}_i, & \text{for } 1 \leq i \leq NRL.
\end{align*}
\]

7.14 Test Problem MESSAGE.

In this test problem, a simple equation is solved on a domain consisting of a message with up to ten lines. This test problem was designed mainly for fun, and to make software demonstrations more interesting.

The problem to be solved is the linear partial differential equation

\[
\begin{align*}
a_1 &= A1X \frac{\partial u}{\partial x} + A1Y \frac{\partial u}{\partial y} + A1U u, \\
a_2 &= A2X \frac{\partial u}{\partial x} + A2Y \frac{\partial u}{\partial y} + A2U u, \\
f &= -BUX \frac{\partial u}{\partial x} - BUY \frac{\partial u}{\partial y} - CU1 u - CU0
\end{align*}
\]

with homogeneous Dirichlet boundary conditions. The parameters \(A1X, A1Y, A1U, A2X, A2Y, A2U, BUX, BUY, CU0\), and \(CU1\) can all be set in \textit{USRCMD}. String
parameters $LINE0$, $LINE1$, ..., $LINE9$ can be set in USRCMD to a user specified message. Upper case and lower case letters, numbers, and several symbols found on a standard keyboard are available. Two possible domains are shown in Figure 7.6. All domains are defined by skeletons, so TRIGEN must be called to generate a triangulation.

![Figure 7.6. The domain for test problem BOX, and two sample domains for test problem MESSAGE.](image)

### 7.15 Test Problem USMAP.

In this test problem, a simple equation is solved on one of 51 domains; 50 are outlines of individual states in the United States, and the last is an outline of the continental U. S. As with test problem MESSAGE, this test problem was designed mainly for fun.

The problem to be solved is the linear partial differential equation

$$
a_1 = A1X \frac{\partial u}{\partial x} + A1Y \frac{\partial u}{\partial y} + A1U u,$$

$$a_2 = A2X \frac{\partial u}{\partial x} + A2Y \frac{\partial u}{\partial y} + A2U u,$$

$$f = -BUX \frac{\partial u}{\partial x} - BUY \frac{\partial u}{\partial y} - CU1 u - CU0$$

with homogeneous Dirichlet boundary conditions. The parameters $A1X$, $A1Y$, $A1U$, $A2X$, $A2Y$, $A2U$, $BUX$, $BUY$, $CU0$, and $CU1$ can all be set in USRCMD.

All domains are specified as skeletons, derived from PostScript and PDF files from the National Digital Map Library at the University of Virginia. The parameter $ISTATE$, $1 \leq ISTATE \leq 51$, specifies the domain. The parameter $ICTY$ takes on values 0 and 1; if $ICTY = 1$, county lines (state lines in the case of the U. S. map) are included as part of the skeleton. If $ICTY = 0$, the skeleton consists of just the outline of the state or country. Several domains (e. g. Michigan, Hawaii) are not connected. Many have small islands that can be excluded from the skeleton by setting the parameter $ISLE = 0$. If $ISLE = 1$, all small islands are included as part of the skeleton. Several domains are shown in Figure 7.7. Since all domains are defined by skeletons, TRIGEN must be called to generate a triangulation.

11The definition of small is problem dependent and depends on the judgment of the author.
Figure 7.7. Sample domains for test problem USMAP. ICNTY = 1 and ISLE = 0 for all domains.
Bibliography


Index

A1XY
   calling sequence, 28
A2XY
   calling sequence, 28
ANGMN, see Table 2.13
ANORM, see Table 2.13
AREA, see Table 2.13
ATEST
   array dimensions, 111
   commands, 102
   common blocks, 111
   initialization defaults, 111
   reading data files, 112
   resetting parameters
      long form, 103
      short form, 104
   writing data files, 112
ATEST1
   common block, 111
ATEST2
   common block, 111
ATEST3
   common block, 111
ATEST4
   common block, 111
ATEST5
   common block, 111
ATEST6
   common block, 111
ATEST7
   common block, 111
BEST, see Table 2.13
BFILE, see Table 2.14
   PLTMG output, 73
   definition, 101
BLUE
   definition, 117
BMNRMO, see Table 2.13
BNORM
   definition, 72
BNORM0, see Table 2.13
BRATIO, see Table 2.13
   calling sequence
      A1XY, 28
      A2XY, 28
      CENTRE, 11
      FXY, 28
      GDATA, 114
      GDXY, 31
      GNXY, 31
      GPHPLT, 94
      INPLT, 91
      P1XY, 28
      P2XY, 31
      PFILL, 117
      PFRAME, 117
      PLINE, 117
      PLTMG, 59
      PLTUTL, 117
      QXY, 32
      SKLUTL, 18
      SXY, 12
      TIMER, 116
      TRIGEN, 37
      TRIPLT, 84
      USRCMD, 112
      USRFIL, 113
      USRSET, 113
CENTRE
   calling sequence, 11
CENX, see Table 2.13
   definition, 88, 93
CENY, see Table 2.13
definition, 88, 93

CMD, see Table 2.14
coefficient functions, 28
common block
ATEST1, 111
ATEST2, 111
ATEST3, 111
ATEST4, 111
ATEST5, 111
ATEST6, 111
ATEST7, 111
PLTMG1, 28
PLTMG2, 28
PLTMG3, 28
PLTMG4, 28
PLTMG5, 28
PLTMG6, 28
PLTMG7, 28
VAL0, 30
VAL1, 31
VAL2, 32
VAL3, 32
VAL4, 12
curved edges
circular arcs, 10
parametric, 11

DELTA, see Table 2.13
definition, 72
DIAM, see Table 2.13
DNEW, see Table 2.13
DRDRL, see Table 2.13
definition, 68
DTOL, see Table 2.13
definition, 62, 64

E
definition, 19, 21, 43
E0, see Table 2.13
EAVE2, see Table 2.13
definition, 44
EF, see Table 2.13
eigenvalue problem, 73
element quality, 38
ENORM1, see Table 2.13
definition, 42
ENORM2, see Table 2.13
definition, 42

f command, see Table 6.1
IDevice, see Table 2.12
FPANE, 119
FTITLE, see Table 2.14
definition, 84
FXY
calling sequence, 28

GDATA
calling sequence, 114
FPANE, see Table 2.12
GPANE, see Table 2.12
JPANE, see Table 2.12
GDXY
calling sequence, 31
GET_COMMAND
calling sequence, 117
GF, see Table 2.9
definition, 19
GNYX
calling sequence, 31
GPANE, 119
GPPLT
calling sequence, 94
continuation path, 98
error estimates, 98
multigraph convergence histories, 96
Newton convergence history, 94
timing statistics, 97
GRADE, see Table 2.13
definition, 39
GREEN
definition, 117
GRFILE
definition, 118
GTITLE, see Table 2.14
definition, 94
HMAX, see Table 2.13
definition, 39
Index

HMIN, see Table 2.13

i command, see Table 6.1
IADAPT, see Tables 2.12 and 3.1
definition, 37
IBNDRY, see also Table 2.1
definition, 15
ICONT, see Table 2.12, see Table 5.2
definition, 84
ICRSN, see Tables 2.12 and 5.2, see Table 5.2
definition, 90
in parallel graphics, 84
IDBLE
definition, 115
IERRSW, see Tables 2.12 and 3.2
definition, 41
IEVALS, see Table 2.12
definition, 62
IFIRST, see Tables 2.12 and 2.5
definition, 22
IFLAG, see Tables 2.12 and 2.15
definition, 23
IFUN, see Tables 2.12 and 5.1
definition, 84
IGRSW, see Tables 2.12 and 5.5
definition, 94
IKND
definition, 115
INPLSW, see Tables 2.12 and 5.3, see Table 5.3
definition, 92, 93
INPLT
calling sequence, 91
skeleton plots, 93
triangle plots, 92
IOMSG, see Table 2.14
IORD, see Table 2.14
IP
definition, 22
IPANE, see Table 6.6
definition, 119
IPATH, see Table 2.10
definition, 21
IPROB, see Tables 2.12 and 4.1
definition, 59
IREFN, see Table 2.12
definition, 50
IRGN, see Table 2.12
IRTYPE, see Tables 2.12 and 3.3
definition, 43
ISCALE, see Tables 2.12 and 5.2, see Table 5.2
definition, 89
ISING, see Table 2.12
definition, 33
ISNGL
definition, 115
ISPD, see Tables 2.12 and 2.14
ITASK, see Tables 2.12 and 4.2
parameter identification problem, 74
definition, 59, 68
ITDOF, see also Table 2.8
definition, 19
ITITLE, see Table 2.14
definition, 92
ITNODE, see also Table 2.3, see also Table 2.5
definition for skeleton, 16
definition for triangulation, 13
ITNUM, see Table 2.12
definition, 62
ITRGT, see Table 2.12
definition, 90
in parallel graphics, 84
IU
definition, 111
IUSRSW, see Table 2.12
definition, 113
j command, see Table 6.1
definition, 112
journal file
definition, 112
JPANE, 119
JRFILE, see Table 2.14
definition, 112
JTFILE, see Table 2.14
definition, 101
JWFILE, see Table 2.14
PLTMG output, 73
definition, 101

$k$ command, see Table 6.1

LENAD, see Table 2.12
LENAOD, see Table 2.12
LENJA, see Table 2.12
LENJA0, see Table 2.12
LENJU, see Table 2.12
LENJU0, see Table 2.12
LENU0, see Table 2.12
LENUOD, see Table 2.12
LINES, see Tables 2.12 and 5.2, see Table 5.2
definition, 89
LIPATH, see Table 2.12
LOGO, see Table 2.14

MAXB, see Table 2.12
definition, 111
MAXD, see Table 2.12
definition, 111
MAXPTH, see Table 2.12
definition, 111
MAXT, see Table 2.12
definition, 111
MAXV, see Table 2.12
definition, 111
METHOD, see Tables 2.12 and 4.3
definition, 62
MFLAG, see Table 2.12
MODE, see Tables 2.12 and 6.1
command line, 110
definition, 101

MPI
creating IPATH, 57
domain decomposition, 65
file names, 110
interface, 120
load balancing, 55
parallel graphics, 84, 98
MPIRGN, see Table 2.12, see Table 5.2
files, 112
in graphics, 89, 93
MPISW, see Table 2.12
definition, 110

MTHDEF
definition, 115
MX, see Table 2.12
definition, 99
MXCG, see Table 2.12
definition, 62
MXCOLR, see Table 2.12
definition, 83, 93
MXLABL, see Table 2.12
MXNWTT, see Table 2.12
definition, 61
MXORD, see Tables 2.12
MY, see Table 2.12
definition, 99
MZ, see Table 2.12
definition, 99

N0, see Table 2.13
NB, see Table 2.12
definition, 34
NBB, see Table 2.12
NBF, see Table 2.12
definition, 10, 13, 15
NBG, see Table 2.12
NBI, see Table 2.12
NCOLOR
definition, 117
NCON, see Table 2.12
definition, 87, 88
NDD, see Table 2.12
NDF, see Table 2.12
definition, 20
NDG, see Table 2.12
NDI, see Table 2.12
NDL, see Table 2.12
definition, 41
NDTRGT, see Table 2.12
definition, 43
NEF, see Table 2.12
NEWNBF, see Table 2.12
NEWNDF, see Table 2.12
NEWNTF, see Table 2.12
NEWNVF, see Table 2.12
NF, see Table 2.13
NGF, see Table 2.12
Index

NOCHNG, see Table 2.12  
definition, 54
NPROC, see Table 2.12  
definition, 55, 109
NRL, see Table 2.12  
definition, 74
parametric edges, 12
NTF, see Table 2.12  
definition, 13, 14
NTG, see Table 2.12
NUMBRs, see Tables 2.12 and 5.2,  
see Table 5.2  
definition, 89, 93
NVDD, see Table 2.12
NVF, see Table 2.12  
definition, 10, 13, 15
NVG, see Table 2.12
NVI, see Table 2.12
NVV, see Table 2.12
NX, see Table 2.12  
definition, 87, 88
NY, see Table 2.12  
definition, 87, 88
NZ, see Table 2.12  
definition, 87, 88

p command, see Table 6.1
w command  
definition, 110
P1XY  
calling sequence, 28
P2XY  
calling sequence, 31
PFILL  
calling sequence, 117
PFRAME  
calling sequence, 117
PLINE  
calling sequence, 117
PLTMG  
branch switching, 71
calling sequence, 59
common blocks, 28
discretization, 3
initialization defaults, 28
normalization equations, 68
PLTMG1  
common block, 28
PLTMG2  
common block, 28
PLTMG3  
common block, 28
PLTMG4  
common block, 28
PLTMG5  
common block, 28
PLTMG6  
common block, 28
PLTUTL  
calling sequence, 117
q command, see Table 6.1
QUAL, see Table 2.13
QXY  
calling sequence, 32
R, see Table 2.13
r command, see Table 6.1  
definition, 112
R0, see Table 2.13
R0DOT, see Table 2.13
RDBLE  
definition, 115
RDOT, see Table 2.13
RED  
definition, 117
REG4, see Table 2.13
REG5, see Table 2.13
RELER0, see Table 2.13
RELERP, see Table 2.13
RELLERR, see Table 2.13
RELRES, see Table 2.13
RKND  
definition, 115
RL, see Table 2.13
RL0, see Table 2.13
RL0DOT, see Table 2.13
RL1, see Table 2.13
RL10, see Table 2.13
RL2, see Table 2.13
RL3, see Table 2.13
RL4, see Table 2.13
RL5, see Table 2.13
RL6, see Table 2.13
RL7, see Table 2.13
RL8, see Table 2.13
RL9, see Table 2.13
RLDOT, see Table 2.13
RLSTRT, see Table 2.13
RLTGT, see Table 2.13
definition, 68
RMAG, see Table 2.13
definition, 88, 93
RMTRGT, see Table 2.13
definition, 67
RMU, see Table 2.13
RP, see Table 2.13
definition, 22
RQUAD
definition, 115
RSNGL
definition, 115
RSTRT, see Table 2.13
RTRGT, see Table 2.13
definition, 68
RU
definition, 111
GRFILE, see Table 2.14
RWFILE, see Table 2.14
definition, 112
s command, see Table 6.1
SCALE, see Table 2.13
definition, 72
SCLEQN, see Table 2.13
definition, 68
SEQDOT, see Table 2.13
SF
circular arcs, 10
parametric, 12
SIGMA, see Table 2.13
definition, 68
skeleton
definition, 14
SKLUTL
calling sequence, 18
SMAX, see Table 2.13
definition, 87, 88
SMIN, see Table 2.13
definition, 87, 88
SP
definition, 22
STEP, see Table 2.13
definition, 61
SU
definition, 111
SVAL, see Table 2.13
SVAL0, see Table 2.13
SXY
calling sequence, 12
symmetry
in TRIGEN, 17
t command, see Table 6.1
test problem
SQUARE, 122
BATTERY, 129
BOX, 131
BURGER, 128
CIRCLE, 121
CONTROL, 129
DOMAINS, 124
IDENT, 130
JCN, 126
MESSAGE, 131
MNSURF, 128
NACA, 124
OB, 127
USMAP, 132
THETA
definition, 68
THETAL, see Table 2.13
THETAR, see Table 2.13
TIMER
calling sequence, 116
triangulation
definition, 13
TRIGEN
calling sequence, 37
element quality, 38
error estimates, 41
mesh smoothing, 48
refinement, 43, 50
triangulation algorithms, 39
unrefinement, 43
TRIPLT
   calling sequence, 84
   hidden lines, 91
   surface plots, 84
   vector plots, 88

u command, see Table 6.1
definition, 112
UNORM1, see Table 2.13
UNORM2, see Table 2.13
USRCMD
calling sequence, 112
USRFIL
calling sequence, 113
USRSET
calling sequence, 113

VAL0
   common block, 30
VAL1
   common block, 31
VAL2
   common block, 32
VAL3
   common block, 32
VAL4
   common block, 12
VX
definition, 10
VY
definition, 10

w command, see Table 6.1
definition, 112
WEBPRT, see Table 2.12
   command line, 110
definition, 104

XMAX, see Table 2.13
XMIN, see Table 2.13
YMAX, see Table 2.13
YMIN, see Table 2.13