NBSIR 86-341

B2DE -- A Program for Solving Systems of Partial Differential Equations in Two Dimensions

James L. Blue

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Center for Applied Mathematics
Scientific Computing Division
Gaithersburg, MD 20899

June 1986

'ARTMENT OF COMMERCE
BUREAU OF STANDARDS

NBSIR 86-3411

B2DE -- A PROGRAM FOR SOLVING SYSTEMS OF PARTIAL DIFFERENTIAL EQUATIONS IN TWO DIMENSIONS

James L. Blue

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Center for Applied Mathematics
Scientific Computing Division
Gaithersburg, MD 20899

July 1986

U.S. DEPARTMENT OF COMMERCE, Malcolm Baldrige, ***Secretary***
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, ***Director***

B2DE - A Program for Solving Systems of Nonlinear Elliptic Partial Differential Equations in Two Dimensions

Manual and Users' Guide

James L. Blue
Scientific Computing Division
National Bureau of Standards
Gaithersburg, MD 20899

ABSTRACT

B2DE is a program for solving systems of nonlinear elliptic partial differential equations (PDEs) in two dimensions. The program is a collection of modules with an interactive driver. Many types of interactive graphics plots are included.

Users may modify the driver, and may be able to construct a "black box" program for a restricted class of PDEs.

B2DE is available from the author at the above address.

1. Introduction

B2DE is a program for solving systems of nonlinear elliptic partial differential equations (PDEs) in two dimensions. B2DE has been used for solving problems as simple as Laplace's equation [Kell86] and as complicated as the coupled equations of electrostatics and charged-particle transport in a transistor [Blue83, Wils85].

Since there is no algorithm for solving all solvable nonlinear systems of PDEs, B2DE cannot work as a "black box" program, as does, for example, a Gaussian elimination program. Instead, B2DE is a collection of modules; the user must decide what to do at each point of the solution process. An interactive driver is provided for the user. Expert users may wish to modify the driver, and may construct a "black box" program for a restricted class of PDEs. Less expert users may wish to modify some routines designed for easy user modification.

Several kinds of graphic displays are provided to aid the user in understanding the solution and the solution process.

This manual includes a description of the driver program and a description of the interface to the lower level programs. It does not include a description of the detailed working of the lower level routines themselves.

The drivers and example routines are provided in well-structured and indented Fortran 77. The lower level routines were written in a private version of Ratfor [Kern75]; the output of the Ratfor processor is indented Fortran 77, but is not pretty. B2DE is single-precision.

Beginners to B2DE should first try one of the sample problems, using the interactive driver. Its use is described in Section 3.2. To do a new problem, users must modify a main program, three subroutines, and a data file to describe the PDEs. They should read Sections 3.1 - 3.3 and Section 3.4.1.

More experienced users may wish to change some of the USRxxx routines. They should read all of Section 3.4.

Expert users may wish to modify the drivers, the DRVxxx routines. They should read Section 4.

Installation of B2DE on your computer and graphics terminals is covered in Section 5.

Section 6 contains several example problems.

2. What B2DE Does

2.1 Mathematical Model

The current version of the B2DE solves systems of N PDEs, each of the form:

$$\nabla \cdot [a_i(x, y, \mathbf{u}, \partial\mathbf{u}/\partial x, \partial\mathbf{u}/\partial y)\nabla u_i] = f_i(x, y, \mathbf{u}, \partial\mathbf{u}/\partial x, \partial\mathbf{u}/\partial y) \quad i = 1, 2, \ldots, N \tag{1}$$

in a region bounded by straight line segments and circular arc segments. On each segment of the boundary, each of the u_i must obey either a nonlinear Dirichlet boundary condition (type 1):

$$g_i(x, y, \mathbf{u}) = 0 \tag{2}$$

or a nonlinear normal derivative, or Neumann, boundary condition (type 2):

$$\frac{\partial u_i}{\partial \nu} + g_i(x, y, \mathbf{u}) = 0. \tag{3}$$

On a given boundary segment, the type of boundary condition may be different for different PDEs. This is a fairly general example of a system of elliptic PDEs arising from conservation of a flux.

The numerical software was inspired by finite-element software of Bank and Sherman [Bank79, Bank82] and retains most of their philosophy.

At the heart of the package is a module which solves a system of linearized elliptic PDEs using linear finite elements on a mesh of triangles. (Boundary "triangles" may have one curved side, a circular arc segment.) The initial triangulation is an input to the package; succeeding triangulations are calculated adaptively by the package. For a given triangulation with M vertices, the calculation proceeds as follows.

The M finite-element basis functions $\{b_m\}$ are linear on each triangle; b_m is 1 at vertex m and 0 at all other vertices. For the ith PDE, the set of vertices on type 1 boundaries is denoted by D_i (for Dirichlet). The solution is approximated by a sum of basis functions

$$u_i(x, y) = \sum_{m=1}^{M} \alpha_{im} b_m(x, y). \tag{4}$$

The coefficients are determined by a Galerkin method [Stra73]; the error in the ith PDE is made orthogonal to each of the basis functions except those in D_i:

$$\iint \left[-\nabla \cdot (a_i \nabla u_i) + f_i\right] b_m = 0, \quad m = 1, 2, \ldots, M, m \text{ not in } D_i \tag{5}$$

The remaining conditions on the α's are that the type 1 boundary conditions hold exactly:

$$g_i\left(x_m, y_m, \mathbf{u}(x_m, y_m)\right) = 0, \quad m \text{ in } D_i. \tag{6}$$

Equation (5) is integrated by parts to yield

$$\iint (a_i \nabla u_i \cdot \nabla b_m + f_i b_m) + \int_2 a_i g_i b_m = 0, \tag{7}$$

where the single integral is over the type-2 parts of the boundary only. Equations (6) and (7) comprise MN nonlinear equations in MN unknowns, the coefficients $\{\alpha_{im}\}$.

The MN nonlinear equations are solved by an iterative process, a damped Newton's method. Let $\mathbf{v}$ be the Newton correction to $\mathbf{u}$; v_i has the same form as u_i,

$$v_i(x, y) = \sum_{m=1}^{M} \beta_{im} b_m(x, y). \tag{8}$$

To calculate $\mathbf{v}$, replace u_i by $u_i + v_i$ in (6) and (7) and linearize in v_i. The linearization of (6) yields

$$g_i\left(x_m, y_m, \mathbf{u}(x_m, y_m)\right) + \sum_{j=1}^{N} \frac{\partial g_i\left(x_m, y_m, \mathbf{u}(x_m, y_m)\right)}{\partial u_j} \beta_{jm} = 0, \quad m \text{ in } D_i, \tag{9}$$

and the linearization of (7) yields

$$\iint \{a_i \nabla v_i \cdot \nabla b_m +$$

$$\nabla u_i \cdot \nabla b_m \sum_{j=1}^{N} \left[\frac{\partial a_i}{\partial u_j} v_j + \frac{\partial a_i}{\partial(\partial u_j/\partial x)} \frac{\partial v_j}{\partial x} + \frac{\partial a_i}{\partial(\partial u_j/\partial y)} \frac{\partial v_j}{\partial x}\right]$$

$$+ b_m \sum_{j=1}^{N} \left[\frac{\partial f_i}{\partial u_j} v_j + \frac{\partial f_i}{\partial(\partial u_j/\partial x)} \frac{\partial v_j}{\partial x} + \frac{\partial f_i}{\partial(\partial u_j/\partial x)} \frac{\partial v_j}{\partial y}\right]\}$$

$$+ \int_2 b_m \sum_{j=1}^{N} \left[a_i \frac{\partial g_i}{\partial u_j} v_j + g_i \frac{\partial a_i}{\partial u_j} v_j + g_i \frac{\partial a_i}{\partial(\partial u_j/\partial x)} \frac{\partial v_j}{\partial x} + g_i \frac{\partial a_i}{\partial(\partial u_j/\partial y)} \frac{\partial v_j}{\partial y}\right]$$

$$= -\iint (a_i \nabla u_i \cdot \nabla b_m + f_i b_m) - \int_2 a_i g_i b_m \equiv r_{im}. \qquad (10)$$

The right-hand side is the residual, $r_{im}(\mathbf{u})$. Substituting for $\mathbf{u}$ and $\mathbf{v}$ from (4) and (8) and doing the integrals numerically give a set of MN linear equations for the correction coefficients $\{\beta_{im}\}$. The matrix of the equations is the Jacobian matrix. For some problems, the Jacobian matrix may be singular, or nearly so. B2DE allows the user to "regularize" the Jacobian matrix by adding a constant to the "a" term in the Jacobian matrix; this is effectively like adding a multiple of the identity matrix to the Jacobian matrix. (The residual is not changed, so that the solution found is the correct solution to the nonlinear equations.)

B2DE allows the user to iterate on any combination of the PDEs, leaving the others fixed.

There is no guarantee of convergence if the initial guess is not good enough; how "good" is "good enough" is problem dependent.

These linear equations are sparse; there are typically about $7N$ nonzero elements per row, on average, and MN can be a few hundred or a few thousand. On the initial triangulation, with MN a few hundred, a satisfactory way to solve these equations is directly, with sparse Gaussian Elimination [Sher78]. On later triangulations, with MN up to a few thousand, direct solutions take too much time and space; iterative methods are usually better, especially multi-level iterative methods.

The linear equations on the coarsest level are solved by sparse Gaussian elimination. The linear equations on the higher levels are solved by multi-level iteration [Bank79, Bank82, Bran77]. The smoothing iterations provided by B2DE are Gauss-Seidel, minimum-residual Gauss-Seidel, Block Gauss-Seidel using N equations per block (for the N unknowns at a point), minimum-residual Block Gauss-Seidel, and Conjugate Gradients. The user specifies r, the number of multi-level cycles, and m, the number of smoothing iterations in each part of the cycle.

For systems of PDEs, the linear equation matrices are nonsymmetric, and may be difficult to solve; there is no guarantee of convergence. Other smoothing methods would be desirable. Of the types in B2DE, Gauss-Seidel smoothing has been most satisfactory, even for problems without theoretical reason to expect convergence.

Unless $\mathbf{u}$ is close to a solution of the nonlinear finite-element equations, $\mathbf{u} + \mathbf{v}$ may be worse than $\mathbf{u}$. A damped Newton's method is used: replace $\mathbf{u}$ by $\mathbf{u} + \lambda\, \mathbf{v}$, where λ is chosen so that $\| \mathbf{r}(\mathbf{u}+\lambda\mathbf{v}) \| < \| \mathbf{r}(\mathbf{u}) \|$ where $\| \mathbf{r} \|$ is the Euclidean norm of $\mathbf{r}$. This avoids divergence of the iterative process, but does not guarantee convergence.

In practice, the initial triangulation is usually too coarse to give the desired accuracy; local refinement is needed to represent the solution accurately in part or all of the region. After the package has converged to an approximate solution on the initial triangulation, the approximate solution may be used to estimate the error and produce a new and finer triangulation adaptively. There are various ways to estimate the error in each triangle. For the present work, a local method is used.

Consider a single triangle; for simplicity we suppose its vertices are 1, 2, and 3. The approximate solution inside the triangle is

$$u_i(x,y) = \sum_{m=1}^{3} \alpha_{im} b_m(x,y). \tag{11}$$

We tentatively divide the triangle into four similar triangles by connecting the midpoints of the three sides. On the four triangles, a potentially more accurate u_i is

$$u_i'(x,y) = \sum_{m=1}^{6} \alpha_{im}' b_m'(x,y). \tag{12}$$

(The primes refer to the finer triangulation.) We keep the old values at the original 3 vertices; $\alpha_{im}' = \alpha_{im}$, $m = 1,2,3$. Values at the 3 new vertices are found by solving the linearized finite element equations for α_{im}', $m = 4,5,6$. The error in the triangle can be estimated by the largest change at any of the points 4, 5, and 6.

This estimate is not accurate in absolute value, but the relative estimates in different triangles are sufficiently accurate for refining the triangulation.

The refined triangulation is obtained by dividing the triangles with the largest estimated errors, such as all triangles with estimated error larger than 1/4 of the largest error, or the worst 1/4 of the triangles. If the refined triangulation adds only a few more triangles, the user may choose to "compress," or merge the new triangulation with the previous one.

For the refined triangulation, the previously calculated **u** is probably a good guess at the new **u**, so that few Newton iterations should be necessary. The hardest work is usually done on the initial triangulation, the one with the fewest vertices.

2.2 Program Description

B2DE is written in a private version of Ratfor [Kern75]; the result is portable Fortran 77. A few routines are unavoidably nonportable; these are described in Section 5.

The PORT library [Fox78] machine constants are used to improve portability. All work space is managed using the PORT library stack.

B2DE may be viewed as a three-layer program package. In the top layer are the programs written by the user to describe a particular problem, and the `USRxxx` programs which are user-modifiable. In the middle layer are programs callable by users, such as `SOLxxx` and `PLTxxx`. In the bottom layer are programs which the user cannot see. The user does not need to know anything about the bottom layer programs, and only needs to know the calling interface to the middle layer programs. In particular, all the data structure of B2DE is invisible to the user.

B2DE comes with interactive driver programs, `DRVxxx`, which call middle- and top-layer programs. The user is free to revise the driver programs.

3. Using B2DE with Supplied Drivers

3.1 Specifying Your Problem

To use B2DE on your program, you need to write a main program and subroutines to evaluate the coefficients a, f, and g and their first partial derivatives. The easiest way to do this is to modify an existing program. See Appendix A and the sample programs on the distribution tape for examples. The current DRVxxx driver assumes a data file to specify the geometry of the region and a few parameters. See Appendix B and the sample problems on the distribution tape.

Working space is managed inside B2DE using the PORT stack [Fox78]. Your main program must define the size of the stack and initialize it using istkin, as in Appendix A. The size of the stack determines the largest problem that can be solved.

Default versions of the USRxxx routines exist, but can be replaced by the user. See Section 3.4. For nonlinear problems, you will want to replace USRGSS to provide an initial guess at the solution.

Note that all of B2DE, including the DRVxxx routines, uses free-format reading. If you make a mistake, such as putting a decimal point in an integer, or putting in a random character, you will be informed and can make a correction if you are running interactively. If, for example, a program asks for four integers they can all be on one line, or on separate lines. Blank lines are completely ignored. Input lines can have comments in them; a # or a * begins a comment, which continues to the end of the line.

See Section 6 for the examples.

3.2 The Interactive Driver - Finding a Solution

DRVxxx programs are drivers that make calls to subroutines in B2DE to solve the PDEs. You may modify or replace any of these routines, but you should follow the model of the default version carefully.

Your main program should be modeled on USRMAI. It will probably call DRVALL, which is the driver which handles the input phase and then calls the solution driver. The default version calls USRIN1, then SOLINP, USRIN2, USRIN3, and finally DRVSOL. DRVALL assumes that needed files have been opened and that the input/output values have been set by calls to IOSET. Note that ausr, fusr, and gusr must be declared EXTERNAL; the names can be anything you like, but ausr, fusr, and gusr will be used in this manual.

```
    subroutine DRVALL(ausr, fusr, gusr)
    external ausr, fusr, gusr
```

DRVSOL is the driver for solving the PDEs. DRVSOL first calls TRMPLT to do any needed initializing for plotting, then calls SOLREF to get started. The main part of DRVSOL is a loop in which an integer is read and one of nine actions is taken. Eight of the actions are calls to subroutines DRVPRT, DRVPLT, SOLITR, SOLREF, SOLCMP, SOLSAV, SOLDIF, and SOLCHG. All these are described separately, and may be called by the user in a modified driver. The ninth action is to quit; TRMPLT is then called to do any needed finishing off for plotting.

```
    subroutine DRVSOL(ausr, fusr, gusr)
    external ausr, fusr, gusr
```

The remainder of this section is a description of the interactive driver, using a run from Example 1. The responses from B2DE are those from print level 2.

The interactive driver is menu-driven; user responses are one or more numbers. Except at the top level, entering 0 as the answer to a question gets you to the next-higher menu.

On entering, B2DE prints the level (this will be 1 unless an old solution is being retrieved), the time taken to set up the mesh (grid), the number of vertices in the level 1 mesh (nv), and the time taken to do initial set-up (asmbnl).

Then the main menu is printed.

```
    refsol entered at level 1 7 vertices
    grid .033 sec.  16 vertices
    asmbnl .033 sec.
    1=print, 2=plot, 3=iter, 4=refine, 5=comp, 6=save, 7=diff, 8=end, 9=change
```

The possible responses will be described separately.

3.2.1 Printing

The DRVPRT driver program is called. Its main printing menu DRVPRT is

```
    1=summary, 2=timing, 3=storage, 4=vertices, 5=pde variables, 6=settings
```

Printing is to the terminal unless the 6=settings menu has been used to direct printing and plotting to files. Each of the responses will be treated in order.

3.2.1.1 Summary

PRTSUM is called. For each level, the number of vertices and the number of triangles are printed. For each PDE and each level for which refinement has been attempted, a measure of the solution size, unorm, an estimate of the size of the error, enorm, and the estimated number of correct significant digits in the solution are printed. The error and the estimated number of digits are only estimates; they are not guaranteed; they are almost certainly not accurate.

```
        n lvl  vertices triangles      unorm      enorm  digits
        1   1        52       114  7.04E-01   1.04E-02    1.83
        1   2       103       258
```

3.2.1.2. Timing

PRTTIM is called. The number of seconds taken in each of the major parts of B2DE is printed, together with the percentage and the total. assembly is the time taken to calculate Jacobian matrices. factor is the time taken to factor the level 1 matrices. mg is the time taken to solve the linear equations to get the Newton step. rhs check is the time taken to evaluate the residual to see if a Newton step is acceptable.

```
              timing distribution
                  seconds  percent
    plot            0.000     0.00
    assembly         2.200    41.51
    factor           0.210     3.96
    mg               0.180     3.40
    rhs check        1.780    33.58
    refine           0.930    17.55
    total            5.300   100.00
```

3.2.1.3. Storage

PRTSTO is called. The maximum number of triangles and vertices are printed, together with the number used so far. The maximum number depends on the size of the stack used.

```
             103  vertices used.
            2094  vertices allowed.
             263 triangles used.
            5583 triangles allowed.
```

3.2.1.4. Vertices

PRTVER is called. The coordinates of each vertex are printed, together with the "parents" of the vertex; the parents are the two vertices of the triangle edge which was bisected to produce the vertex. (The original, user-defined vertices are their own parents.)

```
                 parents  x              y
            1    1     1  -5.00000E-01    0.00000E+00
            2    2     2   0.00000E+00    5.00000E-01
            3    3     3   5.00000E-01    5.00000E-01
            4    4     4   0.00000E+00    0.00000E+00
            5    5     5   5.00000E-01    0.00000E+00
            6    6     6   0.00000E+00   -5.00000E-01
            7    7     7  -5.00000E-01   -5.00000E-01
            8    2     4   0.00000E+00    2.50000E-01
            9    1     4  -2.50000E-01    0.00000E+00
           10    1     2  -3.53553E-01    3.53553E-01
           11    3     4   2.50000E-01    2.50000E-01
          ...
```

3.2.1.5. PDE variables

PDE values are printed. If a transformation is in effect, the transformed valueds are printed when PDE 1 is asked for. PRTPDE is called. Its main menu is :

```
    1=on grid, 2=at vertices, 3=on grid as scaled integers
```

Values can be obtained on a regular rectangular grid or at the vertices. Grid values can be printed as floating-point numbers or as scaled integers (useful for quickly getting an idea of the solution). The three types will be discussed in order.

3.2.1.5.1. On Grid

A PDE is evaluated on a rectangular grid aligned with the x- and y-axes. If the magnification is 1, the grid is on a rectangle defined by the minimum and maximum x and y vertex values. If the magnifications is greater than 1, the grid covers only a portion of this rectangle.

The next menu is:

```
    enter pde no., 1=value only, 2=gradient only, 3=both
```

Enter the PDE number, from 1 through npde, then 1, 2, or 3.

```
    nx, ny
```

You are then asked for the number of grid points along x and along y. If the region is not rectangular, or if its edges are not aligned with the axes, some grid points will fall outside the region. These points have a large, machine-dependent number printed to denote out-of-bounds; in these examples, the number is 1.70141E+38.

A typical printing is:

```
                x              y              u
         1  -5.00000E-01  -5.00000E-01   0.00000E+00
         2  -2.50000E-01  -5.00000E-01   0.00000E+00
         3   0.00000E+00  -5.00000E-01   0.00000E+00
         4   2.50000E-01  -5.00000E-01   1.70141E+38
         5   5.00000E-01  -5.00000E-01   1.70141E+38
         6  -5.00000E-01  -2.50000E-01   4.03474E-01
        ...
```

3.2.1.5.2. At Vertices

You are asked which PDE to evaluate. Since the gradient is not continuous at the vertices, it is not printed.

```
    enter pde no.
```

A typical printing is:

```
                x              y              u
         1  -5.00000E-01   0.00000E+00   1.00000E+00
         2   0.00000E+00   5.00000E-01   1.00000E+00
         3   5.00000E-01   5.00000E-01   6.68726E-01
         4   0.00000E+00   0.00000E+00   7.04038E-01
         5   5.00000E-01   0.00000E+00   0.00000E+00
         6   0.00000E+00  -5.00000E-01   0.00000E+00
        ...
```

3.2.1.5.3. On Grid as Scaled Integers

A PDE is evaluated on a rectangular grid aligned with the x- and y-axes. If the magnification is 1, the grid is on a rectangle defined by the minimum and maximum x and y vertex values. If the magnifications is greater than 1, the grid covers only a portion of this rectangle.

The next menu is:

```
    enter pde no., 1=value only, 2=gradient only, 3= both
```

Enter the PDE number, from 1 through npde, then 1, 2, or 3.

```
    nx, ny
```

You are then asked for the number of grid points along x and along y. If the region is not rectangular, or if its edges are not aligned with the axes, some grid points will fall outside the region. These points have 9999 printed to denote out-of-bounds. Other points are scaled to lie from -100 to 100. A typical printing is:

```
x limits   -1.66667E-01   1.66667E-01
y limits   -5.00000E-01  -1.66667E-01
pde   1
scaled values from -6.085E-08 to  5.492E-01  scale  5.492E-01
   90   90   91   93   95   97  100
   78   78   80   82   85   89   92
   64   65   67   70   75   80   85
   49   50   54   58   65   71   77
   34   35   39   44   54   62   70
   17   19   21   28   42   55   65
    0    0    0    0 9999 9999 9999
```

3.2.1.6. Settings

The next menu is

```
1=color, 2=erase, 3=magnify, 4=window, 5=file print/plot, 6=transform
```

The same menu is available from the plotting menu. All six menu choices affect plotting, but only choices 3, 5, and 6 affect printing. This menu is discussed in detail in Section 3.3.6, under plotting.

3.2.2 Plotting

Plotting is covered in great detail in Section 3.3.

3.2.3 Iterating

Using the current mesh, do one or more Newton iterations on the nonlinear finite element equations which represent the PDEs. In each iteration, a Jacobian matrix and a right-hand-side are calculated; the solution of these equations is the Newton correction. If the mesh level is 1, the linear equations are solved by sparse Gaussian elimination. If the mesh level is greater than 1, the linear equations are solved by a multi-level iteration, with the level 1 equations solved by sparse Gaussian elimination.

If the PDEs are linear, one iteration is sufficient if the linear equations are solved sufficiently accurately. The prompt message is

```
enter niter, ieps, relerr, abserr, jsmax, param
```

niter is an integer, the maximum number of Newton iterations to be done. Fewer will be done if the accuracy requirements are met.

ieps is an integer which governs the accuracy with which the linear equations are solved. Let $\| \mathbf{r} \|$ be the Euclidean norm of the initial right-hand-side; this is also the residual of the nonlinear equations and the initial residual of the linear equations. The multi-level iterations are stopped when the new residual is $2^{-ieps} \| \mathbf{r} \|$ or less, or when maxr multi-level iterations have been done. maxr is set by subroutine USRIN3.

The value of ieps is immaterial if the level is 1, since level 1 equations are solved by sparse Gaussian elimination.

If ieps is too small, the linear equations will not be solved accurately enough, and more iterations may be needed. If ieps is too large, more work may be used in solving the linear equations than is justified by the accuracy of the approximate solution. relerr and abserr are real numbers which govern the accuracy with which the nonlinear equations are solved. Newton iterations are stopped when the residual is small enough, when $\| \mathbf{r} \| <=$ (relerr $\| \mathbf{u} \|$ +abserr), where $\| \mathbf{u} \|$ is the Euclidean norm of the current approximate solution.

jsmax is an integer which governs the damped Newton solution. If $\mathbf{v}$ is the Newton correction, the new approximate solution is $\mathbf{u} + t\mathbf{v}$. First $t = t_0$ is tried, where t_0 is the value returned by USRSTP, then $t_0/2$, then $t_0/4$, and so on, until the new residual is small enough; that is, until $\| \mathbf{r}(\mathbf{u}+t\mathbf{v}) \| < \| \mathbf{r}(\mathbf{u}) \|$. A total of jsmax tries are made.

param is a real parameter for the convenience of the user; it is put into a common block and can be accessed by the user. It may be useful in, for example, USRREG. The common block is

```
COMMON /USRCOM/ PARAM
```

If the nonlinear iterations have not converged by the niter-th iteration, DRVSEE is called to give the user an opportunity to look at the Newton correction before the correction has been tested. The menu is

```
1=print, 2=plot, 3=continue
```

Menu choices 1 and 2 are exactly the same as choices 1 and 2 of the main menu, except that the user prints or plots the current Newton correction rather than the current approximate solution. The current magnification, window, and color are used. Use menu choice 3 to go on to test the Newton correction.

B2DE has provision for user-supplied adjustments. For example, the PDEs may depend on the solution to some nonlinear equation formally separate from the PDEs. After $\mathbf{u}$ has been replaced by $\mathbf{u} + t\mathbf{v}$, but before the residual is calculated, USRAD1 is called. After a satisfactory value of t has been found, and after the residual is calculated, USRAD2 is called.

3.2.4 Refining

Using the current approximate solution, estimate the error in the solution and adaptively refine the mesh. B2DE first displays the current level number, the number of vertices, and a crude guess at the maximum error in the solution for each PDE. A typical example is:

```
refsol re-entered at level   2    26 vertices
asmbnl     .050 sec.
  1 estimated maximum error   8.01E-03
```

B2DE then gives you an opportunity to plot the estimated error for each PDE. When you are finished, if npde is 2 or more, B2DE gives you an opportunity to plot the total estimated error for all the PDEs combined.

B2DE then shows the estimated Euclidean error for each PDE, shows how many triangles will be refined because the local estimated error is high enough, and then refines. The estimated error and the number of correct digits are almost certainly wrong, but are indicative of the progress of the solution from level to level.

If any of the weights returned by USRWTS is negative, B2DE asks for the weights to use and asks if you want to refine.

```
level  1 pde  1  unorm, enorm   7.05E-01   2.86E-02 est. digits  1.39
min error / max error  1.44 percent
     4 of    30 triangles ( 13.33) above 50.00 percent
     9 of    30 triangles ( 30.00) above 25.00 percent
     8 of    30 triangles ( 26.67) above 37.50 percent
```

B2DE tells you the number of vertices at the new level.

```
refine     .700 sec.   level   2      52 vertices
```

3.2.5 Compressing

The vertices of the highest-level mesh are merged with those of the next-to-highest-level mesh, and the level number is reduced by 1.

This option is useful in cases where a refinement generates only a few new vertices, as may happen in problems with a singularity in the solution.

3.2.6 Saving the solution

The current approximate solution is saved on a file. The file must have been opened by the user before calling B2DE. Let n = ioget(8).

If $n > 0$, the information is written as unformatted (binary) information on logical file unit n. If $n < 0$, the information is saved as formatted information on logical file unit -n. The prompt message is

```
enter the number of significant figures to save
```

After the user enters the number of significant figures to save, B2DE constructs an appropriate format and writes the file.

Formatted files take more space to write, but can be used to transport data from one computer to another.

3.2.7 Looking at the Difference

Look at the change in the current solution since the last time the mesh was refined, or since the start of the current session with B2DE, whichever is later. DRVSEE is called.

The menu is

```
1=print, 2=plot, 3=continue
```

Menu choices 1 and 2 are exactly the same as choices 1 and 2 of the main menu, except that the user can print or plot the current difference rather than the current approximate solution. Use menu choice 3 to go back to the main menu.

3.2.8 Quitting

Quit.

3.2.9 Changing Parameters

Change user-defined parameters, internal B2DE parameters, or change which PDE variables are being iterated on.

First `USRCHG` is called; in `USRCHG`, the user may change any user-defined parameters as desired.

If npde is 2 or more, the user may keep the values of one or more of the PDE variables fixed, and calculate Newton corrections for the remaining PDE variables. If npde is 2 or more, B2DE next prints

```
    PDE variables - 0 for fixed, 1 for varying
```

and prints the current status of each of the npde PDE variables.

```
    enter new values
```

The user must enter the new values as integers: 1 means the PDE is to vary, and any other integer means the PDE is held fixed. At least one 1 must be entered.

The next prompt is

```
    enter 1 to change parameters, 0 to return
```

If 1 is entered, the next prompt is

```
    parameters:  maxm, maxr, ittype, ithrsh, iprsw
```

and the current values are printed out. Enter the new values, all integers. Refer to Section 3.4.9, `USRIN3`, for the meanings of the parameters. `maxm`, `maxr`, and `ittype` all govern multi-level iteration. `ithrsh` governs refinement. `iprsw` governs the amount of printing as the solution progresses.

3.3 The Interactive Driver - Graphics

Seeing pictures is necessary if the user is (1) to understand what is happening during the iterative process leading to a solution and (2) to understand the properties of the solution.

B2DE gives the user ample opportunity to make plots. There are standard plots available of several kinds. The variables plotted are either a PDE variable or the magnitude of its gradient. Since the author cannot guess everything the user might want to plot, B2DE allows the user to define problem-dependent plot variables (Section 3.4.12). Plots may be made to the user's graphics terminal, or the vectors may be written to a file for later processing.

Most of the graphics program code is independent of the computer used and of the graphics output device. The dependent parts are treated in the installation part of the manual, Section 5.

Five kinds of plots are available: triangles, contours, surfaces, profiles, and flow lines. Each plot type is done by a subroutine which may be called directly from a user's program. Each plot type also has an interactive driver which may be called directly from a user's program. There is an overall interactive plot driver.

The interactive driver is DRVPLT. The user may wish to modify or replace this program. DRVPLT asks interactively for user input, then calls any or all of DRVTRI, DRVCON, DRVSUR, DRVPRO, DRVFLW, and DRVSET. The user may wish to modify or replace any of these programs.

Most of the DRVxxx programs call the corresponding PLTxxx programs: PLTTRI, PLTCON, PLTSUR, PLTPRO, and PLTFLW. The calling sequences for the PLTxxx programs are described in Section 4.3; PLTxxx programs may be called directly by the user, but are not meant to be changed by the user. Some PLTxxx programs require work space, which is obtained from the stack; if insufficient stack space is available, the plot is skipped.

DRVSET calls any of six sub-driver programs: DRVCOL, DRVERA, DRVMAG, DRVWDW, DRVFPL, and DRVSHF. The DRVxxx programs call the corresponding SETxxx programs: SETERA, SETCOL, SETMAG, SETWDW, SETFPL, and SETSHF. The calling sequences for the SETxxx programs are described in Section 4.1; SETxxx programs may be called directly by the user, but are not meant to be changed by the user.

Plots can be drawn to the full screen or plotting surface, or can be drawn in a subwindow. Contour plots, flow line plots, and triangle plots can be magnified.

All plots are of vectors only, with no plot labeling. If the hardware used supports it, erasing between plots is optional. If the hardware used supports it, the line type or color of the plots can be changed.

The following description of plotting assumes the overall interactive driver, DRVPLT. All the DRVxxx programs are menu-driven and use a free-format input package. As in the rest of B2DE's interactive drivers, responses of 0 return the user to the next higher-up menu. Erroneous responses are dealt with reasonably; error messages should be self-explanatory. The top-level menu of DRVPLT is:

```
    1=triangles, 2=contours, 3=surface, 4=profiles, 5=flow lines, 6=settings
```

The response should be a non-negative integer. A response of 0 means to go to the next higher menu. Other responses outside the indicated range are ignored. The responses will be treated in order.

3.3.1 Triangle plots

The DRVTRI driver is called. The next menu is :

```
    enter level
```

Any existing level of the triangulation can be plotted. The current magnification and window are used. The plot is in the current color.

After the plot is done, the program may wait for the user's signal. The signal, if any, is that required by the terminal-dependent program, PLTUTL. After the signal, the time taken is displayed, and the

```
    enter level
```

menu comes up again. To get back to the previous menu, enter

```
    0 0
```

3.3.2 Contour Plots

The DRVCON driver is called. The next menu is :

```
pde no., 1=function/2=abs(grad), 1=fill/2=no fill
```

Enter the PDE number, from 1 through N, the number of PDEs in the problem. Enter 1 for contours of the PDE value, 2 for the absolute value of its gradient. Enter 1 to fill in the contours with different colors in the current palette, or 2 to just draw the contours in the current line color. The current magnification and window are used. If fill is chosen, the next menu is :

```
start with color number
```

Enter the first color of the color palette, which may be from 0 on up. The space between the bottom two contours will be filled with this color; succeeding spaces will be filled with successive colors. The next menu is :

```
number of contours
```

The number of contours is limited to 101 by PARAMETER (MAXCON = 101) in program DRVCON. The magnification is treated as for triangle plots.

If nc, the number of contours, is positive, the program finds the maximum and minimum values of the plot variable and spaces the contours equally between, but not including, the extreme values. Both the extreme values and the contour values are displayed.

If nc is 0, the extreme values are displayed, but no plot is done.

If nc is negative, the user is prompted for extreme values for contours. There are -nc contours, equally spaced between and including the extreme values.

If no fill is chosen, the next prompt is :

```
number of contours
```

and the replies are as above.

After the plot is done, the time taken is displayed, and the

```
pde no., 1=function/2=abs(grad), 1=fill/2=no fill
```

menu comes up again. To get back to the previous menu, enter

```
0 0 0
```

3.3.3 Surface Plots

The DRVSUR driver is called. The current color and window are used. The magnification is 1. The next menu is :

```
pde no., 1=function/2=abs(grad)
```

Enter the PDE number, from 1 through N. Enter 1 for a plot of the PDE value, 2 for a plot of the absolute value of its gradient. The next menu is :

```
level, 1=surface triangles, 2=surface contours, 3=surface grids
```

Enter the triangulation level desired for the plot, and the type of surface plot desired. The absolute value of the level is used. In all three types, the function is plotted as a parallel projection plot (no perspective); the height of the surface above or below the x-y plane represents the function value. The surface can be viewed from any angle. Hidden lines are omitted unless the level is negative; then hidden lines are plotted. Plotting without removing hidden lines requires much less computation, and can be much faster.

The three types will be discussed in order.

3.3.3.1 Surface triangles

The surface is defined by the triangles of the specified level. The next menu is :

```
    enter viewing direction -- x, y, z
```

The plot is displayed as if seen by a viewer looking towards the surface along a line extending from the origin through the point (x, y, z). After the plot is done, the time taken is displayed, and the

```
    pde no., 1=function/2=abs(grad)
```

menu comes up again. To get back to the previous menu, enter

```
    0 0
```

3.3.3.2 Surface contours

The surface is defined by the contour lines drawn on triangles of the specified level. Triangle boundaries are not shown. The first menu is again :

```
    pde no., 1=function/2=abs(grad)
```

and the responses are as before. The next menu is :

```
    number of contours
```

Responses to this menu are like those for the planar contour plot, except no magnification is allowed. The number of contours is limited to 202 by PARAMETER (MAX = 202) in program DRVSUR.

The next menu is :

```
    enter viewing direction -- x, y, z
```

Responses are as before. After the plot is done, the time taken is displayed, and the

```
    pde no., 1=function/2=abs(grad)
```

menu comes up again. To get back to the previous menu, enter

```
    0 0
```

3.3.3.3 Surface grids

The surface is defined by grid lines parallel to the x and y axes, drawn on triangles of the specified level. Triangle boundaries are not shown. The first menu is again

```
    pde no., 1=function/2=abs(grad)
```

and the responses are as before. The next menu is

```
    number of x lines, number of y lines
```

The x lines are equally spaced along the x-axis, from the minimum x-value to the maximum x-value; the y lines are similarly spaced along the y-axis. The total number of grid lines, x plus y, is limited to 202 by `PARAMETER (MAX = 202)` in program `DRVSUR`. The next menu is:

```
    enter viewing direction -- x, y, z
```

Responses are as before. After the plot is done, the time taken is displayed, and the

```
    pde no., 1=function/2=abs(grad)
```

menu comes up again. To get back to the previous menu, enter

```
    0 0
```

3.3.4 Profile Plots

The `DRVPRO` driver is called. The current color and window are used. The next menu is :

```
    enter pde no., number of profiles
```

Enter the PDE number, from 1 through N, and the number of profiles desired. A profile is a slice through a surface plot, viewed from the side. The slices are parallel and equally spaced. The next menu is :

```
    enter indices of profile plane(s) x, y
```

The slices are parallel to a line running from the origin to the point (`x, y`). Viewing is from the direction as defined for surface plots, (y, -x, 0), perpendicular to the slices. After the plot is done, the time taken is displayed, and the

```
    enter pde no., number of profiles
```

menu comes up again. To get back to the previous menu, enter

```
    0 0
```

3.3.5 Flow Line Plots

The `DRVFLW` driver is called. Flow line plots attempt to describe the flux, or vector field, $a_i \nabla u_i$. The user specifies a line segment, or set of connected segments; the positive and negative fluxes across the line are calculated; points along the line are spaced evenly in the flux; and flow lines are drawn both ways from these points. Flow lines are terminated at boundaries of the region or when the flow lines would re-enter a triangle. The current magnification and window are used. The plot is in the current color.

Since the solutions are composed of linear finite elements on triangles, ∇u_i is constant within each triangle. Gradients tend not to go smoothly to zero when they should, but overshoot instead; this makes the flow lines stop.

The next menu is :

```
    enter pde no., number of points, number of flow lines
```

The PDE number can be from 1 to N. The number of points is the number in the set of connected line segments, two or more. The next menu is :

```
    enter # x-values followed by # y-values
```

where # is the number of flow lines just entered. The total flow in a positive and in a negative directions across the broken line is printed, and then the flow lines are drawn. After the plot is done, the time taken is displayed, and the

```
enter pde no., number of points, number of flow lines
```

menu comes up again. To get back to the previous menu, enter

```
0 0 0
```

3.3.6 Settings

The DRVSET driver is called. Several values can be set; these apply to future plots until the settings are changed. The menu is

```
1=color, 2=erase, 3=magnify, 4=window, 5=file print/plot, 6=transform
```

Setting the Color

The DRVCOL driver is called. Some terminals can support vectors drawn in different colors or line types. The menu is :

```
1=change color number, 2=change color palette
```

If 1 is entered, the next menu is:

```
enter new color number
```

SETCOL is called, which calls TRMCOL, and then the settings menu appears.

If 2 is entered in response to

```
1=change color number, 2=change color palette
```

the system-dependent subroutine TRMPAL is called. Then the settings menu appears.

Setting the Erase Flag and Erasing the Screen

The DRVERA driver is called. Some terminals support optional erasing between plots. If the erase flag is ON, the screen is erased between plots; if the erase flag is OFF, the screen is not erased between plots. You might want to plot, for instance, surface contours on the same plot as a surface grid, or planar contours of more than one PDE. The menu is:

```
1=set erase flag off, 2=set erase flag on, 3=erase screen
```

If option 3 is chosen, the screen is erased immediately, but the erase flag is not changed. There are no lower menus. After the desired option is chosen, SETERA is called and the settings menu appears.

Setting the Magnification

The DRVMAG driver is called. B2DE can make magnified triangle, flow line, and contour plots. The parameters necesary are the magnification and the point which will be the center of the magnified plot. The magnification also affects printing of PDE variables on a grid, either as real values or as scaled integers. The initial menu is:

```
enter magnification
```

If the magnification entered is 1 or less, the magnification is set to 1 and you are returned to the main plot driver menu. If the magnification is greater than 1, the next menu is:

```
    enter x-center and y-center (0 to 100)
```

An unmagnified plot fits in a rectangle aligned with the x- and y-axes, defined by the minimum and maximum x and y vertex values. For magnifications greater than 1, the center of the magnified plot is defined with respect to this rectangle; the center is defined by the percentage distance from the minimum to the maximum coordinates of the rectangle. For plotting, the center may be adjusted to fill as much of the screen as possible. For example, if the center is given as 100 100, the upper right corner of the rectangle will be plottd to the upper right corner of the screen rather than to the center of the screen.

SETMAG is called and the settings menu appears.

Setting the Window

The DRVWDW driver is called. B2DE can draw its plots in the entire plotting surface as defined by TRMSCL, or can use a subwindow. The prompt is

```
    enter x-left, x-right, y-bottom, y-top (from 0 to 100)
```

The numbers entered are interpreted as percentages of the full window. For example, 0 50 50 100 means the upper-left quarter of the full window. SETWDW is called and the settings menu appears.

Printing and Plotting to Files

The DRVFPL driver is called. Printing and plotting can be done to the screen of the terminal or to a file. One flag governs both printing and plotting. The prompt is

```
    1=plot/print to screen, 2=plot/print to file
```

SETFPL is called and the settings menu appears.

Users may write their own programs to read the plot file and produce plots on other devices. The format of the file is as follows. The first line is produced by the terminal-dependent program PLTUTL, and may be changed by the user. The default version writes a line containing the minimum and maximum integer values produced by the plot,

```
    xmin ymin xmax ymax
```

in (4I7) format. These values come from the terminal-dependent program TRMSCL, and may be changed by the user; the default values are 500, 9500, 500, and 7250.

The plot vectors are next in the file. Each is on a separate line,

```
    x1 y1 x2 y2
```

in (4I7) format. The end of the plot is marked by a plot vector

```
    -1 -1 -1 -1
```

which is produced by TRMPLT, and which may be changed by the user.

Transforming a Solution

The DRVSHF driver is called. The user may specify transformations of PDEs and print or plot the transformed values. The prompt is

```
    enter index of transformation
```

SETSHF is called, which calls USRSHF, and the settings menu appears.

3.4 Easy Modifications - USRxxx Programs

Default routines USRxxx are supplied; these routines are designed to be modifiable by the user. They provide easy ways to alter the performance of B2DE without going inside B2DE. Users should read the default versions carefully before modifying the programs.

3.4.1 USRMAI is the Fortran "main" program. USRMAI declares storage, sets up for input and output (I/O), and calls a driver routine. The *a*, *f*, and *g* programs should be declared EXTERNAL in this program. The stack must be declared and initialized in the main program. I/O values must be set by calling IOSET for all nine values. These values may be retrieved by calling IOGET; B2DE routines needing to do I/O get file numbers in this way.

io(1) is the file number to be used for reading from the terminal.

io(2) is the file number to be used for writing to the terminal.

io(3) is the terminal type; it is mainly used by graphics programs. If io(3) is non-positive, the session is assumed to be batch, not interactive. All plots will be made to files, and FREEIN, the free-format input routine, will abort if an input line is bad.

io(4) is the file number to be used for reading input data; it is only used in USRxxx programs.

io(5) is the file number to be used for writing plot data, if any plotting is done to files rather than to the terminal.

io(6) is the file number to be used for writing print data, if any printing is done to files rather than to the terminal.

io(7) is the file number to be used for reading the old solution data, if an old solution is to be read. If io(7) is positive, the solution data is assumed to have been written as unformatted data. If io(7) is negative, the solution data is assumed to have been written as formatted data.

io(8) is the file number to be used for writing the new solution data, if a solution is to be saved. If io(8) is positive, the solution data will be written as unformatted data. If io(8) is negative, the solution data will be written as formatted data.

io(9) is the file number to be used by B2DE for writing scratch data. Data is written to and read from io(9) as unformatted data.

Data files corresponding to file numbers io(4), io(5), io(6), io(7), and io(8) should be opened if there is any chance they will be used. File io(9) is always used and should be opened.

DRVALL or another driver should be called.

Any files opened should be closed.

3.4.2 USRAD1 is called to make any needed adjustments after a tentative Newton step has been taken, but before the new residual is calculated. The default version does nothing.

```
    subroutine USRAD1
```

3.4.3 USRAD2 is called to make any needed adjustments after a Newton step has been accepted, after the new residual is calculated. The default version does nothing.

```
subroutine USRAD2
```

3.4.4 USRCHG is called when the 9=change selection from the main menu is taken. It should return .TRUE. if the changes made affect the residuals of the PDEs so they must be recalculated; it should return .FALSE. if the residuals need not be recalculated. The default version returns .FALSE.

```
logical function USRCHG(junk)
```

junk is a dummy parameter.

3.4.5 USRGET is called when an old solution is being read in, to read anything that was saved by USRSAV. The default version does nothing. USRGET should read values in the same order and with the same formats as USRSAV writes. USRGET is always called with isin equal to io(7).

```
subroutine USRGET(isin)
integer isin
```

isin is an input, the file number to read from.

3.4.6 USRGSS is called to get the initial guess at the solution. The default version guesses 0; this is adequate for linear problems, but is probably not adequate for most nonlinear problems. USRGSS is called for each point in the initial mesh.

```
subroutine USRGSS(npde, x, y, i, u)
integer npde, i
real x, y, u(npde)
```

npde is the number of PDEs.

x and y are the coordinates of the point.

i is the number of the user-supplied triangle in which the point lies.

u is an output, the guess at the solution at the point.

3.4.7 USRIN1 is the first input routine. It returns six values. The default version calls IOGET(4) to get a file number, and calls FREEIN to read the values from the file. USRIN1 is called by DRVALL; if DRVALL is replaced by the user, USRIN1 may not be needed.

```
subroutine USRIN1(oldsol, nt, nv, nc, nb, npde)
logical oldsol
integer nt, nv, nc, nb, npde
```

oldsol should be .TRUE. if an old solution is to be read in, and .FALSE. if a new solution is to be started from the beginning. If oldsol is .TRUE., the remaining five values do not need to be returned; the old values are used by B2DE.

nt is the number of triangles in the user-supplied coarsest mesh.

nv is the number of vertices in the user-supplied coarsest mesh.

nc is the number of curved boundary edges in the user-supplied coarsest mesh.

nb is the number of boundary edges in the user-supplied coarsest mesh.

npde is the number of PDEs.

3.4.8 USRIN2 is the second input routine. It is called to obtain the geometry of the user-supplied coarsest mesh. It is called only if oldsol is .FALSE.. The default version calls IOGET(4) to get a file number, and calls FREEIN to read the values from the file.

```
    subroutine USRIN2(nt, nv, nc, nb, npde, lref, vxin, vyin,
   * xmid, ymid, itnode, ibdry)
    integer nt, nv, nc, nb, npde
    real vxin(nv), vyin(nv), xmid(nc), ymid(nc)
    integer ibdry(nb, npde+4), itnode(3, nt), lref(nv)
```

nt, nv, nc, nb, and npde are all inputs; they are as described under USRIN1. The remainder of the arguments are outputs.

lref(j) is the initial refinement level for vertex j.

vxin(j) is the x-coordinate of vertex j.

vyin(j) is the y-coordinate for vertex j.

xmid(j) is the x-coordinate for the midpoint of curved edge j, if nc > 0.

ymid(j) is the y-coordinate for the midpoint of curved edge j, if nc > 0.

itnode(1,k), itnode(2,k), and itnode(3,k) are the three vertices for triangle k.

ibdry(m, n) contains data for boundary edges m.

ibdry(m, 1) must be m.

ibdry(m, 2) is the number of the triangle containing the boundary edge, in the range 1 to nt.

ibdry(m, 3) is the edge number of the boundary edge within the triangle. It must be 1, 2, or 3; the first edge is opposite the first-named vertex in the itnode array for the triangle.

ibdry(m, 4) is 0 if the edge is straight, and is the curved edge number, in the range 1 to nc, if the edge is curved.

ibdry(m, n+4) is the boundary condition type for the n-th PDE on this boundary edge; it should be 1 for Dirichlet-type boundary conditions and 2 for Neumann-type boundary conditions.

3.4.9 USRIN3 is the third input routine. It is called to obtain parameters for the solution process. It is called only if oldsol is .FALSE. The default version calls IOGET(4) to get a file number, and calls FREEIN to read the values from the file. All the arguments are outputs.

```
    subroutine USRIN3(maxm, maxr, ittype, lvlref, lvlusr, numlvl,
   * ithrsh, iprsw, iwork)
```

maxm is the "m" parameter for multi-level iterations [Bank79].

maxr is the "r" parameter for multi-level iterations [Bank79].

ittype is the type of smoothing iteration.

0: point Gauss-Seidel.

1: conjugate gradient.

2: block Gauss-Seidel.

lvlref is the maximum pre-refinement level assigned to any vertex.

lvlusr is the maximum pre-refinement level used in USRTST.

numlvl is the number of mesh levels allowed.

ithrsh is the threshold percentage for mesh adapting.

ithrsh $= 0$: do uniform refinement.

ithrsh > 0 : refine triangles whose estimated error is at least this percent of the largest estimated error.

ithrsh < 0 : refine triangles whose estimated error puts them in the worst abs(ithrsh) percent of all triangles.

iprsw determines how much printing is done as the solution is being found.

0 : as little as possible.

1 : more.

2 : reasonable for interactive use.

3 : far too much.

iwork is the number of words of stack space to use for solving. B2DE will use the smallest of iwork, the value initialized by USRMAI, and the actual available space.

3.4.10 USRREG should return the regularization parameter for the n-th PDE. In calculating the Jacobian matrix, B2DE uses a_n + USRREG(n) to allow for regularizing nearly-singular Jacobian matrices. The default version returns zero.

```
real function USRREG(n)
integer n
common / usrcom / param
real param
```

3.4.11 USRSAV is called when a solution is being saved, for the user to save necessary problem parameters not known to B2DE. The default version does nothing. USRGET should read values in the same order and with the same formats as USRSAV writes. USRSAV is always called with isout equal to io(8).

```
subroutine USRSAV(isout)
integer isout
```

isout is the file number to write to.

3.4.12 USRSHF is called to "shift," or transform PDE variables for plotting. All the arguments are inputs except unew, which is an output. The default version does nothing.

```
subroutine USRSHF(ixtype, npde, nv, u, unew, vx, vy)
integer ixtype, npde, nv
real u(npde,nv), unew(nv), vx(nv), vy(nv)
```

ixtype is the index of the transform type.

npde is the number of PDEs.

nv is the number of vertices in the current finest mesh.

u is the solution on the finest mesh.

vx and vy are the coordinates of the vertices.

unew is the transformed solution. It may be any desired function of the PDE variables, the coordinates, or anything else.

3.4.13 USRSTP provides the user an opportunity to avoid problems such as overflow or the square root of a negative number, by restricting the maximum fraction of a Newton step that can be used. B2DE will try to evaluate residuals at (uold + t ustep), where t is the value returned by USRSTP. USRSTP should return a number greater than 0 and less than or equal to 1. All the arguments are inputs. The default version returns 1.

```
real function USRSTP(npde, nv, vx, vy, uold, ustep)
integer npde, nv
real vx(nv), vy(nv), uold(npde, nv), ustep(npde, nv)
```

npde is the number of PDEs.

nv is the number of vertices in the current finest mesh.

vx and vy are the coordinates of the vertices.

uold is the current solution.

ustep is the tentative Newton step.

3.4.14 USRTST allows pre-refinement of the initial mesh in addition to that provided by the array lref in USRIN2. After the pre-refinement scheduled by lref, USRTST is called for each triangle, and should return .TRUE. if the triangle is to be refined at least one more time. All triangles are tested until no more triangles are refined. All the arguments are inputs. The default version always returns .FALSE.

```
logical function USRTST(x, y, lvltri, level1)
real x(3), y(3)
integer lvltri, level1
```

x and y are the coordinates of vertices of the triangle.

lvltri is the level of refinement of this triangle so far.

level1 is the maximum level of pre-refinement allowed, as specified by USRIN3.

3.4.15 USRWTS is called when the mesh is being refined; it should return the relative weight for the n-th PDE. If any of the weights is negative and the run is interactive, the driver asks what the weights should be, ignoring the values returned by USRWTS. The default version always returns 1.

```
real function USRWTS(n)
integer n
```

n is the PDE number.

4. Writing Your Own Driver for B2DE

The routines callable by the user are described in several groups: DRVxxx, SETxxx, SOLxxx, PLTxxx, PRTxxx, TRMxxx, USRxxx, and miscellaneously-named routines.

DRVxxx routines are driver programs, and can be modified or replaced by the user. DRVALL and DRVSOL are described in Section 3.2, and DRVPLT is described in Section 3.3.

DRVSEE can be used to inspect the solution

```
subroutine DRVSEE(ausr)
external ausr
```

DRVSEE provides the menu discussed in Section 3.3.3:

```
1=print, 2=plot, 3=continue
```

SOLxxx routines do the solution of the PDEs. They are described in Section 4.2.

PLTxxx routines do plotting. They are described in Section 4.3.

PRTxxx routines do printing. They are described in Section 3.2.1.

SETxxx routines set parameters used for plotting. They are described in Section 4.1.

TRMxxx routines are terminal-dependent programs having to do with plotting. They are described in Section 5.2.

USRxxx routines provide the user with opportunities to do things without requiring detailed knowledge of the interior of B2DE. They are described in Section 3.4.

4.1 SETxxx Routines

4.1.1 SETCOL sets the color or line type for plotting. It saves the input value, then calls the terminal-dependent routine TRMCOL; TRMCOL is described in Section 5.2.1.

```
subroutine SETCOL(newcol)
integer newcol
```

newcol is the new color or line type.

4.1.2 SETERA either erases the screen or sets the flag that determines whether the screen is erased before a new plot is drawn. On some terminals, it may not be possible or desirable to plot without erasing the screen.

```
subroutine setera(iflag)
integer iflag
```

If iflag is 1, the screen will not be erased before each plot.

If iflag is 2, the screen will be erased before each plot.

If iflag is 3, the screen is erased immediately.

Other values for iflag do nothing.

4.1.3 SETMAG sets the magnification and the x- and y-centers for the magnified plots. Magnification applies to triangle plots, contour plots, and flow line plots. It also applies to printing of PDE values on a grid.

```
subroutine setmag(xymag, xctr, yctr)
real xymag, xctr, yctr
```

xymag is the magnification. If xymag is less than 1, B2DE uses a magnification of 1.

xctr and yctr are the coordinates of the center of the image to be magnified, in percentages of the full window size. For example, if xctr and yctr are each 50, the center of the image will be magnified.

4.1.4 SETWDW sets the window in which subsequent plots will be drawn.

```
subroutine setwdw(xmin, xmax, ymin, ymax)
real xmin, xmax, ymin, ymax
```

xmin and xmax are the x-coordinates of the window, in percentages of the maximum window size as given by TRMSCL. Similarly, ymin and ymax are the y-coordinates of the window. For example, if xmin and ymin are each 50, and xmax and ymax are each 100, the plot window will be the upper right quarter of the plotting area.

4.1. SETFPL sets a flag for plotting and printing.

```
subroutine SETFPL(flag)
logical flag
```

If flag is .TRUE., subsequent printing and plotting will be done to a file.

If flag is .FALSE., subsequent printing and plotting will be done to the terminal.

4.1. SETSHF is used to "shift", or transform, the first PDE variable for inspecting a user-defined function of the PDE variables, and to shift it back.

```
logical function SETSHF(iflag)
integer iflag
```

If iflag is not 0, SETSHF calls USRSHF to get new values to put in place of the values of the first PDE, exchanges the values, and returns for plotting or printing. iflag is passed to USRSHF for use as an index.

If iflag is 0, SETSHF restores the values saved for the first PDE.

4.2 SOLxxx Routines

SOLxxx routines are called by driver programs to solve the PDEs. They are described individually below.

4.2.1 SOLCHG is used to change parameters during a solution. There are three types of parameters. First, USRCHG is called for changing user-supplied parameters. Second, if there are two or more PDEs in the problem, one or more PDE variables can be held fixed during the nonlinear iterations. Third, B2DE parameters maxm, maxr, ittype, ithrsh, and iprsw can be changed.

```
    subroutine SOLCHG
```

4.2.2 SOLCMP is used to compress, or merge, the top two levels of mesh into a single mesh. Lower level meshes are not affected. SOLCMP is useful if a refinement did not add many new vertices to the mesh. There are no inputs or outputs.

```
    subroutine SOLCMP
```

4.2.3 SOLDIF is used to inspect the "difference" in a solution; this is the difference between the starting solution values (initial guess, read-in values from old solution, or interpolated solution from next coarser mesh) on the current finest mesh and the current solution values. Norms of the changes are printed, and DRVSEE is called for possible printing or plotting. SOLDIF is useful for inspecting the changes in the solution. The only input is the *a* function; there are no outputs.

```
    subroutine SOLDIF(ausr)
    external ausr
```

4.2.4 SOLINP handles most of the details of input for a problem. All the arguments are outputs. Their meanings were discussed in Section 3.4.7, under USRIN1.

```
    subroutine SOLINP(oldsol, nt, nv, nc, nb, npde)
    logical oldsol
    integer nt, nv, nc, nb
```

4.2.5 SOLITR handles the nonlinear iterations; it is the workhorse routine. The inputs are the names of the three subroutines describing the PDEs. SOLITR asks for values of iteration parameters and then proceeds.

```
    subroutine SOLITR(ausr, fusr, gusr)
    external ausr, fusr, gusr
```

ausr, fusr, and gusr are the names of subroutines that describe the PDEs and the boundary conditions.

4.2.6 SOLREF is used either to start a new solution or to refine the mesh for an existing solution.

```
    subroutine SOLREF(ausr, fusr, gusr, newlvl)
    external ausr, fusr, gusr
```

ausr, fusr, and gusr are the names of subroutines that describe the PDEs and the boundary conditions.

newlvl should be 0 to get started on the same level, or 1 to refine to the next higher level.

4.2.7 SOLSAV saves the solution values and the data structure by writing them to a file in a format suitable for later use by B2DE. SOLSAV calls USRSAV in case the user wants to write anything for later retrieval by USRGET.

```
    subroutine SOLSAV
```

4.3 PLTxx Routines

PLTxxx routines are callable by the user, but should not be changed. In the default driver, only the corresponding DRVxxx routines call the PLTxxx routines. The five routines are PLTCON, PLTFLW, PLTPRO, PLTSUR, and PLTTRI. They are described separtately.

4.3.1 PLTCON does a contour plot of a PDE variable. The contour values are printed to the standard output, but do not appear on the plot. The plot uses the current color (or color palette for filled contours), window, and magnification.

```
subroutine PLTCON(nth, ntype, fill, istcol, nlev, ulev)
integer nth, ntype, istcol, nlev
real ulev(*)
logical fill
```

nth: Plot contours of the n-th PDE.

ntype: If ntype is 1, plot contours of $u_n(x,y)$; otherwise plot contours of $|\nabla u_n(x,y)|$.

fill: If fill is .TRUE., the spaces between contours are filled in; otherwise only the contour lines are drawn.

istcol is the starting color for filling spaces between contours. istcol is ignored if fill is .FALSE.

nlev: If nlev is positive, plot nlev contours, using automatic scaling, spacing the contours evenly between, but not including, the maximum and minimum values of the plot variable; ulev is ignored by PLTCON. If nlev is negative, plot -nlev contours, using the contour values in the ulev array. If nlev is zero, just find and print the minimum and maximum values of the plot variable.

4.3.2 PLTFLW draws flow lines for a pde variable. The amount of positive and negative flow is printed to the standard output. The plot uses the current colors, window, and magnification.

```
subroutine PLTFLW(axy, nth, npf, nflow, xyval, rp)
real xyval(*)
external axy
```

axy is the name of the a subroutine.

nth is as described under PLTCON.

npf is the number of points defining the crossing line.

xyval is an array of the points, first the npf x-values, then the npf y-values.

nflow is the number of flow lines to be plotted.

4.3.3 PLTPRO draws profile plots, the outline of slices through a projection plot. The plot uses the current color and window, but uses magnification of 1.

```
subroutine PLTPRO(nth, nprof, dir)
integer nth, nprof
real dir(2)
```

nth is as described under PLTCON.

nprof is the number of profiles.

dir are the indices of the slicing plane, as discussed in Section 3.3.4.

4.3.4 PLTSUR draws a surface projection plot of a PDE variable. The plot uses the current color and window, but uses magnification of 1.

```
 subroutine PLTSUR(nsurf, level, nth, ntype, viewpt, ngrid,
* nlev, ulev)
 integer nsurf, level, nth, ntype, ngrid(2), nlev
 real viewpt(3), ulev(*)
```

nsurf controls what is plotted on the surface. If nsurf is 1, triangles of the specified level are plotted. If nsurf is 2, contours are plotted. If nsurf is 3, x-y grids are plotted.

level is the level of solution to be plotted.

nth and ntype are as described under PLTCON.

viewpt contains the (x, y, z) components of the view point. The surface projection will appear as if it were at the origin, seen from this point.

nlev and ulev are used if only nsurf is 2. Their meanings are as given under PLTCON.

4.3.5 PLTTRI draws a plot of one triangulation. The plot uses the current color, window, and magnification.

```
subroutine PLTTRI(level)
integer level
```

level is the level of the mesh to be plotted.

4.4 Miscellaneous routines

4.4.1 IOGET is used to get the value of one of the input/output parameters. See the USRMAI section for a list of the meanings.

```
integer function ioget(ionum)
integer ionum
```

ionum is the index of the parameter whose value is desired.

4.4.2 IOSET is used to set an input/output parameter. See the USRMAI section for a list of the meanings.

```
subroutine ioset(ionum, ioval)
integer ionum, ioval
```

ionum is the index of the parameter to be set.

ioval is the value to set.

4.4.3 FREEIN is a subroutine to do free-format input. The calling program asks for some real numbers and/or integers to be read from an input file. FREEIN reads a line at a time, parses it, evaluates the numbers, and does error checking. If an error occurs and the session is interactive, FREEIN prompts the user for a new line. Items are separated by spaces; lines are read until all the requested items are obtained. If there are more items on a line than the call to FREEIN requires, the excess is ignored. A * or # character marks the beginning of a comment; the character and the rest of the line are ignored.

```
subroutine FREEIN(ioread, ioerr, nitems, itype, ibuff, rbuff)
integer ioread, ioerr, nitems, itype(nitems), ibuff(nitems)
real rbuff(nitems)
```

ioread is the file number from which to read the data.

ioerr is the file number on which to write error messages.

nitems is the number of items to be read.

itype(j) is the type of the j-th item, 1 for an integer and 2 for a real number.

ibuff and rbuff are arrays into which the data is put. If itype(j) is 1, the j-th item is interpreted as an integer and put into ibuff(j); if itype(j) is 2, the j-th item is interpreted as a real number and put into rbuff(j).

4.4.4 FLTGET is provided for convenience; it returns one real value typed at the terminal, using FREEIN.

```
real function fltget(junk)
```

junk is a dummy parameter.

4.4.5 INTGET is provided for convenience; it returns one integer value typed at the terminal, using FREEIN.

```
integer function intget(junk)
```

junk is a dummy parameter.

4.4.6 EVAL evaluates PDE variables for passing to other programs.

```
subroutine EVAL(kind, nth, npde, npts, x, y, u, dudx, dudy)
real x(npts), y(npts), u(npts), dudx(npts), dudy(npts)
```

If kind is 1, only the PDE variable is evaluated and returned in u; dudx and dudy are not referenced. If kind is 2, only the gradient of the PDE variable is evaluated and returned in dudx and dudy; u is not referenced. If kind is 3, both the PDE variable and its gradient are evaluated.

The nth PDE variable is evaluated.

npde is the number of PDE variables.

npts is the number of points at which to evaluted the nth PDE.

x and y are the points at which to evaluate the nth PDE.

If kind is 1 or 3, the PDE values are returned in u.

If kind is 2 or 3, the PDE gradient values are returned in dudx and dudy.

5. Installing B2DE on Your Computer

Most of B2DE is standard Fortran 77; it should not need changes by the user and should not be changed. All known machine dependencies have been isolated for easy changing.

5.1 The PORT Stack

Working space is managed inside B2DE using the PORT stack [Fox78]. The size of the stack determines how large a problem can be solved. The stack takes advantage of a feature which is in most Fortrans, but which is not in the Fortran 77 standard: the size of a labelled common block is determined by the size in the main program, or by the size in the first program loaded. To change the size of the stack, only the main program needs to be edited and re-compiled. The following block of code occurs many times within B2DE, defining the stack common, cstak, to be a nominal size.

```
    parameter (irsize=1000)
    parameter (idsize= 500)
    common/cstak/dstak
    double precision dstak(idsize)
    real r(irsize)
    integer i(irsize)
    logical lstak(irsize)
    equivalence (dstak(1),r(1)), (dstak(1),i(1)), (dstak(1),lstak(1))
    save /cstak/
```

Your main program will define the true size of the stack.

5.2 Machine-Dependent Programs

To be as portable as possible, B2DE uses the machine constants from the PORT library [Fox78]. The machine constant functions I1MACH and R1MACH must be changed as appropriate for your computer. The distributed version is correct for the DEC VAX 11/700 series and the VAX/VMS operating system. Constants for many other computers are in the routines, but commented out. See [Fox78] for instructions on changing machine constants for computers not listed.

B2DE uses two timing routines. Each should calculate an elapsed amount of time used in milliseconds. ISTIME is called at the beginning of an activity to be timed, and IFTIME is called at the conclusion. The difference is used as the elapsed time.

```
    integer function iftime ( junk )
```

junk is a dummy parameter.

```
    integer function istime ( junk )
```

junk is a dummy parameter.

5.3 Terminal-Dependent Programs

TRMxxx routines are terminal-dependent. The default versions work for DEC VT125 terminals and Hewlett Packard 264x and 262x terminals. New versions must be written for other terminals. Some TRMxxx routines are not called if plotting to a file is being done, and therefore are not necessary if only batch operation is planned, although dummy versions must be provided to satisfy most operating systems.

5.2.1 TRMCOL should set the color or the line type for the plots. If the terminal does not support color or alternate line types, TRMCOL need not do anything. TRMCOL is called by SETCOL, which is called by DRVCOL; DRVCOL only allows non-negative values for the color.

```
subroutine TRMCOL(newcol)
integer newcol
```

newcol is the new color or line type.

5.2.2 TRMDRW should draw a line from the last point reached by TRMDRW or TRMMOV to the point (ix, iy). The point (ix, iy) is guaranteed to be within the limits set by TRMSCL. TRMDRW is not called if plotting to a file is being done. The line should be drawn in the current color or line type as last set by TRMCOL.

```
subroutine TRMDRW(ix, iy)
integer ix, iy
```

ix and iy are the screen coordinates of the new point.

5.2.3 TRMMOV should move the current point to the point (ix, iy). No line should be drawn. The point (ix, iy) is guaranteed to be within the limits set by TRMSCL. TRMMOV is not called if plotting to a file is being done.

```
subroutine TRMMOV(ix, iy)
integer ix, iy
```

ix and iy are the screen coordinates of the new point.

5.2.4 TRMPAL changes the color palette available to the user. It has no arguments.

```
subroutine TRMPAL
```

5.2.5 TRMPLT is a plotting utility. Its function depends upon its argument, as described below.

```
subroutine TRMPLT(iflag)
integer iflag
```

iflag = 1: TRMPLT should do whatever is necessary to initialize all plotting for the session.

iflag = 2: TRMPLT should do whatever is necessary to begin a plot, and should erase the screen.

iflag = 3: TRMPLT should do whatever is necessary to begin a plot, but should not erase the screen. This is not possible on some terminals.

iflag = 4: TRMPLT should do whatever is necessary to finish a plot. If the session is interactive, and if plots are done on the same screen as text, TRMPLT should wait for a signal from the user before returning.

iflag = 5: TRMPLT should do whatever is necessary to finish all plotting for the session.

5.2.6 TRMSCL should give the minimum and maximum screen coordinates for terminal type iterm. If iterm is zero or an unknown type, the values for plotting to a file should be returned.

```
integer function TRMSCL(iterm, iflag)
integer iterm, iflag
```

iflag = 1: TRMSCL should return the minimum x value.

`iflag` = 2: TRMSCL should return the maximum x value.

`iflag` = 3: TRMSCL should return the minimum y value.

`iflag` = 4: TRMSCL should return the maximum y value.

6. Examples

6.1 Laplace's Equation

The equation is Laplace's equation, the simplest elliptic PDE. The region is a square with two corners rounded off, as in Fig. 1.

$$\nabla \cdot \nabla u = 0$$

The boundary conditions are $u = 1$ on edges 1 and 2, $u = 0$ on edge 5, and $\partial u / \partial \nu = 0$ on sides 3, 4, and 6. The solution is singular at vertices 1 and 6.

Appendix A is a program to solve this example; Appendix B is an input data file suitable for use with the default driver; and Appendix C is a sample run.

A typical initial triangulation supplied by a user is shown in Figure 2. Sides 1 and 4 are defined as circular arcs, but the curvature does not show up until the sides are refined. Figure 3a is the actual coarsest triangulation used; the input data file specifies initial refinement around the singularities. Figures 3b and 3c are refined meshes generated by B2DE in the sample run. Note that the adaptive refinement concentrated new triangles near the singularities. Figure 4 shows other plots from the sample run: 4a is a contour plot; 4b is a flow line plot; and 4c is a profile plot.

6.2 A Nonlinear Example

This is a concocted set of three nonlinear elliptic PDEs. The solutions are smooth, and the adaptive mesh generates uniform triangulations. The region is the rectangle $-1 \le x \le 1$, $0 \le y \le 1$.

$$\nabla \cdot \left[\left(1 + \left(\frac{\partial u_2}{\partial x}\right)^2 + \left(\frac{\partial u_2}{\partial y}\right)^2 + \left(\frac{\partial u_3}{\partial x}\right)^2 + \left(\frac{\partial u_3}{\partial y}\right)^2 \right) \nabla u_1 \right] = 2u_1$$

$$\nabla \cdot (\nabla u_2) = 0$$

$$\nabla \cdot (u_1 u_2 \nabla u_3) = 0$$

The boundary conditions are all Dirichlet:

$$u_1 u_2 u_3 = 1$$

$$u_2 = e^x$$

$$u_3 = e^y$$

The solution is

$$u_1 = e^{-x-y}$$

$$u_2 = e^{x}$$

$$u_3 = e^{y}$$

Appendix D is a program to solve this example; Appendix E is an input data file suitable for use with the default driver; and Appendix F is a sample run.

A typical initial triangulation supplied by a user is shown in Figure 5a. Figure 5b is the actual coarsest triangulation used; the input data file specifies initial refinement around the left corner. (There is no singularity in the problem; this initial refinement is unnecessary.) Figures 5c and 5d are refined meshes generated by B2DE in the sample run. Note that the adaptive refinement made the mesh uniform; the incorrect initial mesh has been overcome.

Figure 6 shows other plots from the sample run: 6a is a surface plot showing triangles; 6b is a surface plot showing contours, with hidden lines not removed; and 6c is a surface plot showing grid lines.

References

[Bank79] R. E. Bank and A. H. Sherman, "PLTMG Users' Guide," CNA 152, Center for Numerical Analysis, The University of Texas at Austin, September, 1979.

[Bank82] R. E. Bank, "PLTMG Users' Guide, June, 1981 version," Technical Report, Department of Mathematics, University of California at San Diego, August, 1982.

[Blue83] J. L. Blue and C. L. Wilson, "Two-Dimensional Analysis of Semiconductor Devices Using General-Purpose Interactive PDE Software," *IEEE Trans. Electron Devices*, vol. ED-30, pp. 1056-1070, 1983.

[Bran77] A. Brandt, "Multi-Level Adaptive Solutions to Boundary-Value Problems," *Math. Comput.*, vol. 31, pp. 333-390, 1977.

[Fox78] P. A. Fox, A. D. Hall, and N. L. Schryer, "The PORT Mathematical Subroutine Library," *ACM Trans. Mathematical Software*, vol. 4, pp. 104-126, 1978.

[Kell86] E. F. Kelley, R. E. Hebner, W. E. Anderson, J. A. Lechner, and J. L. Blue, "The Effect of an Oil-Paper Interface Parallel to an Electric Field on the Breakdown Voltage at Elevated Temperatures." Submitted to *IEEE Trans. Power Apparatus and Systems.*

[Kern75] B. W. Kernighan, "RATFOR – A Preprocessor for a Rational Fortran," *Software – Practice and Experience*, vol. 5, pp. 395-406, 1975.

[Sher78] A. H. Sherman, "Algorithms for Sparse Gaussian Elimination with Partial Pivoting," *ACM Trans. Mathematical Software*, vol. 4, pp. 330- 338, 1978.

[Stra73] G. Strang and G. J. Fix, *An Analysis of the Finite Element Method* (Prentice-Hall, Englewood Cliffs, New Jersey, 1973).

[Wils85] C. L. Wilson, P. Roitman, and J. L. Blue, "High Accuracy Physical Modeling of Submicron MOSFETs," *IEEE Trans. Electron Devices*, vol. ED-32, pp. 1246-1258, 1985.

Figure Captions

1. Region for Laplace's equation example. The boundary conditions are $u = 1$ on edges 1 and 2, $u = 0$ on edge 5, and $\partial u_i / \partial \nu = 0$ on sides 3, 4, and 6.
2. User-supplied triangulation of region. The curvature of sides 1 and 4 will show up in refinements of the region.
3a. Initial triangulation of region. Vertices 1 and 6 were specified as refinement level 3; other vertices were level 1.
3b. Triangulation after first adaptive refinement.
3c. Triangulation after second adaptive refinement.
4a. Contours of the approximate solution on the mesh of Fig. 3c. The contours are uniformly spaced.
4b. Flow line plot of the approximate solution on the mesh of Fig. 3c. Equal flux flows between each pair of flow lines.
4c. Profile plot of the approximate solution on the mesh of Fig. 3c, with slicing direction 1 0.
5a. User-supplied triangulation of region for example 2.
5b. Initial triangulation of region. Vertex 4 was specified as refinement level 3; other vertices were level 1.
5c. Triangulation after first adaptive refinement.
5d. Triangulation after second adaptive refinement.
6a. Surface plot of error in PDE 2, on mesh of Fig. 5d. Triangles are shown, and hidden lines are removed.
6b. Surface plot of error in PDE 2, on mesh of Fig. 5d. Contours are shown, and hidden lines are not removed.
6c. Surface plot of error in PDE 2, on mesh of Fig. 5d. Grid lines are shown, 21 along x and 11 along y, and hidden lines are removed.

Appendix A. Program to Solve Laplace's Equation Example

```
c
c main program to drive nonlinear multi-variable code to solve
c Laplace's equation. adapted from umain.for
c
c the user-supplied routines that determine the PDEs
c
      external alap, flap, glap
c
c i/o value array
c
      integer io(9)
c
c the stack must be declared and initialized in the main program.
c it is 'save'ed for safety's sake. the equivalence is required on
c some computers to get the word alignment correct.
c
      parameter (irsize = 100000)
      parameter (idsize =  50000)
      common/cstak/dstak
      double precision dstak(idsize)
      real rstak(irsize)
      equivalence (dstak(1), rstak(1))
      save /cstak/
c
c initialize the stack to tell it the true total size
c
      call istkin(irsize, 3)
c
c specify i/o values
c
c  io(1) = file number for reading from the terminal
c  io(2) = file number for writing to the terminal
c  io(3) = terminal type, used by graphics programs
c
      io(1) = 5
      io(2) = 6
      io(3) = 125
c
c  io(4) = file number for reading input data;
c          only used by usrin1, usrin2, and usrin3
c  io(5) = file number for writing plot data, if not to terminal
c  io(6) = file number for writing print data, if not to terminal
c
      io(4) = 20
      io(5) = 21
      io(6) = 22
c
```

```
c  io(7) = file number for reading old solution data
c  io(8) = file number for writing new solution data
c  io(9) = file number for writing scratch data
c    For reading and writing solution data, the data is written as
c    unformatted, or binary, if the io value is positive, and as
c    formatted if the io value is negative.
c
      io(7) = -23
      io(8) = -24
      io(9) = 25
c
c install the i/o values
c
      do 10 j = 1, 9
          call ioset(j, io(j))
   10 continue
c
c open data files.
c
      open(unit = io(4), file = 'input1.dat', status = 'old')
      open(unit = io(5), file = 'plot.dat'  , status = 'unknown')
      open(unit = io(6), file = 'print.dat'  , status = 'unknown')
c
      if ( io(7) .gt. 0) then
          open(unit = io(7), file = 'insol.dat', status = 'unknown',
     *        form='unformatted')
      else
          open(unit = -io(7), file = 'insol.dat', status = 'unknown',
     *        form='formatted')
      endif
c
      if ( io(8) .gt. 0) then
          open(unit = io(8), file = 'outsol.dat', status = 'unknown',
     *        form='unformatted')
      else
          open(unit = -io(8), file = 'outsol.dat', status = 'unknown',
     *        form='formatted')
      endif
c
      open(unit = io(9), file = 'scrtch.dat', status = 'unknown',
     *     form='unformatted')
c
c take it away . . .
c
      call drvall(alap, flap, glap)
c
c close the files, just to be safe
c
```

```
      close(io(4))
      close(io(5))
      close(io(6))
      close(iabs(io(7)))
      close(iabs(io(8)))
      close(io(9))
c
      stop
      end
c
c
      subroutine alap(npde, ndim, dopde, x, y, i, u, dudx, dudy,
     *                a, dadu, dadux, daduy)
          real x, y, u(*), dudx(*), dudy(*),
     *         a(*), dadu(ndim,*), dadux(ndim,*), daduy(ndim,*)
          logical dopde(npde)
c
c inputs:
c     npde      xthe number of pdes in the problem
c     ndim      the first dimension of arrays dadu, dadux, and daduy
c     dopde     dopde(n) is .TRUE. if the nth is active.
c     x, y      the coordinates of the point at which to evaluate
c     i         the macro triangle number that the point is in
c     u         the current pde values at the point
c     dudx,dudy the gradient of the pdes at the point
c
c outputs:
c     a         the values of the a coefficient
c     dadu      the first derivatives of a with respect to u
c     dadux     the first derivatives of a with respect to du/dx
c     daduy     the first derivatives of a with respect to du/dy
c
      a(1) = 1.
      dadu(1, 1) = 0.e0
      dadux(1, 1) = 0.e0
      daduy(1, 1) = 0.e0
c
      return
      end
c
c
      subroutine flap(npde, ndim, dopde, x, y, i, u, dudx, dudy,
     *               f, dfdu, dfdux, dfduy)
          real x, y, u(*), dudx(*), dudy(*),
     *         f(*), dfdu(ndim,*), dfdux(ndim,*), dfduy(ndim,*)
      logical dopde(npde)
c
c
```

```
c inputs:
c     npde      xthe number of pdes in the problem
c     ndim      the first dimension of arrays dadu, dadux, and daduy
c     dopde     dopde(n) is .TRUE. if the nth is active.
c     x, y      the coordinates of the point at which to evaluate
c     i         the macro triangle number that the point is in
c     u         the current pde values at the point
c     dudx,dudy the gradient of the pdes at the point
c
c outputs:
c     f         the values of the f coefficient
c     dfdu      the first derivatives of f with respect to u
c     dfdux     the first derivatives of f with respect to du/dx
c     dfduy     the first derivatives of f with respect to du/dy
c
      f(1) = 0.e0
      dfdu (1, 1) = 0.e0
      dfdux(1, 1) = 0.e0
      dfduy(1, 1) = 0.e0
c
      return
      end
c
      subroutine glap(npde, ndim, dopde, x, y, i, j, u, g, dgdu)
          real x, y, u(*), g(*), dgdu(ndim,*)
          logical dopde(npde)
c
c inputs:
c     npde      xthe number of pdes in the problem
c     ndim      the first dimension of arrays dadu, dadux, and daduy
c     dopde     dopde(n) is .TRUE. if the nth is active.
c     x, y      the coordinates of the point at which to evaluate
c     i         the macro triangle number that the point is in
c     j         the side of the macro triangle that the point is on
c     u         the current pde values at the point
c
c outputs:
c     g .       the values of the g coefficient
c     dgdu      the first derivatives of g with respect to u
c
c
c square with 2 round corners
c
      if (i.eq.1 .or. i.eq.2) then
         g(1) = u(1) - 1.0
         dgdu(1, 1) = 1.0
      else if (i .eq. 5) then
         g(1) = u(1)
         dgdu(1, 1) = 1.0
      else
         g(1) = 0.0
         dgdu(1, 1) = 0.0
      endif
c
      return
      end
```

Appendix B. File “input1.dat” for Laplace’s Equation Example

```
# input.dat for Laplace example
# square with two rounded corners
0           # old solution flag
6 7 2 6 1   # nt nv nc nb npde
# vert lvl x y
1 3 -0.5  0.0
2 1  0.0  0.5
3 1  0.5  0.5
4 1  0.0  0.0
5 1  0.5  0.0
6 3  0.0 -0.5
7 1 -0.5 -0.5
# curved edges
1  .35355339 -.35355339
2 -.35355339  .35355339
# triangles
1 1 2 4
2 2 3 4
3 3 4 5
4 4 5 6
5 4 6 7
6 1 4 7
1 1 3 2 1    # bc tri edge curv bc
2 2 3 0 1
3 3 2 0 2
4 4 1 1 2
5 5 1 0 1
6 6 2 0 2
2 8 0   # maxm maxr ittype
4 1 7   # lvlref lvlusr maxl
-30     # ithresh
2       # iprsw
80000   # work space
```

Appendix C. Transcript of interactive session for Laplace's Equation Example
Output from the computer is in typewriter font. Numbers typed by the user are in boldface and have a • at the beginning of the line.

```
refsol entered at level   1     7 vertices
grid      0.040 sec.       26 vertices
asmbnl    0.030 sec.
1=print, 2=plot, 3=iter, 4=refine, 5=comp, 6=save, 7=diff, 8=end, 9=change
• 3
enter niter, ieps, relerr, abserr, jsmax, param
• 1 0 0 0 1 0
Iter   1
 1 residual L2 norm   4.397E-01
factor    0.040 sec.      26 eqs.
asmbnl    0.250 sec.
 mg       0.000 sec.  GE
 1 newton norm,solution norm,relerr   1.000E+00   0.000E+00   1.000E+00
0=continue, 1=print, 2=plot
• 0
  0  t  0.000000  L2(rhs)  4.397E-01    0.110 sec.
  1  t  1.000000  L2(rhs)  1.454E-07    0.110 sec.
1=print, 2=plot, 3=iter, 4=refine, 5=comp, 6=save, 7=diff, 8=end, 9=change
• 4
refsol re-entered at level   2    26 vertices
asmbnl    0.030 sec.
  1 estimated maximum error   8.01E-03
estimated individual PDE errors
0=continue, 1=print, 2=plot
• 0
level  1 pde  1  unorm, enorm   7.05E-01   2.86E-02 est. digits  1.39
min error / max error  1.44 percent
     4 of    30 triangles ( 13.33) above 50.00 percent
     9 of    30 triangles ( 30.00) above 25.00 percent
     8 of    30 triangles ( 26.67) above 37.50 percent
refine    0.300 sec.   level   2      52 vertices
1=print, 2=plot, 3=iter, 4=refine, 5=comp, 6=save, 7=diff, 8=end, 9=change
• 5
compressed to level  1
1=print, 2=plot, 3=iter, 4=refine, 5=comp, 6=save, 7=diff, 8=end, 9=change
• 3
enter niter, ieps, relerr, abserr, jsmax, param
• 1 0 0 0 1 0
Iter   1
 1 residual L2 norm   3.650E-02
factor    0.080 sec.      52 eqs.
asmbnl    0.480 sec.
 mg       0.000 sec.  GE
 1 newton norm,solution norm,relerr   7.564E-02   1.000E+00   7.564E-02
0=continue, 1=print, 2=plot
```

```
• 0
  0  t  0.000000  L2(rhs)  3.650E-02    0.260 sec.
  1  t  1.000000  L2(rhs)  7.992E-08    0.240 sec.
1=print, 2=plot, 3=iter, 4=refine, 5=comp, 6=save, 7=diff, 8=end, 9=change
• 4
refsol re-entered at level   2    52 vertices
asmbnl    0.050 sec.
  1 estimated maximum error   2.64E-03
estimated individual PDE errors
0=continue, 1=print, 2=plot
• 0
level  1 pde  1  unorm, enorm   7.04E-01   1.04E-02 est. digits  1.83
min error / max error  5.91 percent
     9 of    72 triangles ( 12.50) above 50.00 percent
    26 of    72 triangles ( 36.11) above 25.00 percent
    14 of    72 triangles ( 19.44) above 37.50 percent
    19 of    72 triangles ( 26.39) above 31.25 percent
    23 of    72 triangles ( 31.94) above 28.13 percent
    20 of    72 triangles ( 27.78) above 29.69 percent
    21 of    72 triangles ( 29.17) above 28.91 percent
    22 of    72 triangles ( 30.56) above 28.52 percent
refine    0.720 sec.   level   2     103 vertices
1=print, 2=plot, 3=iter, 4=refine, 5=comp, 6=save, 7=diff, 8=end, 9=change
• 3
enter niter, ieps, relerr, abserr, jsmax, param
• 1 10 0 0 1 0
Iter   1
 1 residual L2 norm   1.761E-02
factor    0.080 sec.      52 eqs.
asmbnl    1.240 sec.
mg  2  7.619E-01  3.472E-02  4.557E-02
mg  2  3.472E-02  4.404E-03  1.268E-01
mg  2  4.404E-03  5.974E-04  1.357E-01
 mg       0.160 sec.  GS     reduction 7.84E-04
 1 newton norm,solution norm,relerr   5.398E-02   1.000E+00   5.398E-02
0=continue, 1=print, 2=plot
• 0
  0  t  0.000000  L2(rhs)  1.761E-02    0.510 sec.
  1  t  1.000000  L2(rhs)  1.659E-05    0.500 sec.
1=print, 2=plot, 3=iter, 4=refine, 5=comp, 6=save, 7=diff, 8=end, 9=change
• 1
1=summary, 2=timing, 3=storage, 4=vertices, 5=pde variables, 6=settings
• 1
  n lvl  vertices triangles      unorm      enorm  digits
  1  1        52       114   7.04E-01  1.04E-02    1.83
  1  2       103       258
1=summary, 2=timing, 3=storage, 4=vertices, 5=pde variables, 6=settings
• 2
```

```
         timing distribution
                 seconds  percent
plot              0.000     0.00
assembly          2.080    39.77
factor            0.200     3.82
mg                0.160     3.06
rhs check         1.730    33.08
refine            1.060    20.27
total             5.230   100.00
1=summary, 2=timing, 3=storage, 4=vertices, 5=pde variables, 6=settings
•  3
       103  vertices used.
      2094  vertices allowed.
       263 triangles used.
      5583 triangles allowed.
1=summary, 2=timing, 3=storage, 4=vertices, 5=pde variables, 6=settings
•  6
1=color, 2=erase, 3=magnify, 4=window, 5=file print/plot, 6=transform
•  3
enter magnification
•  3
enter x-center and y-center (0 to 100)
•  50 0
1=color, 2=erase, 3=magnify, 4=window, 5=file print/plot, 6=transform
•  0
1=summary, 2=timing, 3=storage, 4=vertices, 5=pde variables, 6=settings
•  5
1=on grid, 2=at vertices, 3=on grid as scaled integers
•  3
enter pde no., 1=value only, 2=gradient only, 3= both
•  1 1
nx, ny
•  7 7
x limits   -1.66667E-01   1.66667E-01
y limits   -5.00000E-01  -1.66667E-01
pde   1
scaled values from -6.085E-08 to  5.492E-01  scale  5.492E-01
   90   90   91   93   95   97  100
   78   78   80   82   85   89   92
   64   65   67   70   75   80   85
   49   50   54   58   65   71   77
   34   35   39   44   54   62   70
   17   19   21   28   42   55   65
    0    0    0    0 9999 9999 9999
1=on grid, 2=at vertices, 3=on grid as scaled integers
•  0
1=summary, 2=timing, 3=storage, 4=vertices, 5=pde variables, 6=settings
•  0
```

```
1=print, 2=plot, 3=iter, 4=refine, 5=comp, 6=save, 7=diff, 8=end, 9=change
• 2
1=triangles, 2=contours, 3=surface, 4=profiles, 5=flow lines, 6=settings
• 6
1=color, 2=erase, 3=magnify, 4=window, 5=file print/plot, 6=transform
• 3
enter magnification
• 1
1=color, 2=erase, 3=magnify, 4=window, 5=file print/plot, 6=transform
• 0
1=triangles, 2=contours, 3=surface, 4=profiles, 5=flow lines, 6=settings
• 1
enter level
• 1
plot     0.360 sec.
enter level
• 2
plot     0.520 sec.
enter level
• 0
1=triangles, 2=contours, 3=surface, 4=profiles, 5=flow lines, 6=settings
• 2
pde no., 1=function/2=abs(grad), 1=fill/2=no fill
1 1 2
number of contours
• 11
min and max values  0.000000E+00  1.000000E+00
contour values
   8.333E-02   1.667E-01   2.500E-01   3.333E-01   4.167E-01
   5.000E-01   5.833E-01   6.667E-01   7.500E-01   8.333E-01
   9.167E-01
plot     0.480 sec.
pde no., 1=function/2=abs(grad), 1=fill/2=no fill
• 0 0 0
1=triangles, 2=contours, 3=surface, 4=profiles, 5=flow lines, 6=settings
• 3
pde no., 1=function/2=abs(grad)
• 0 0
1=triangles, 2=contours, 3=surface, 4=profiles, 5=flow lines, 6=settings
• 5
pde no., number of points, number of flow lines
• 1 2 11
enter     2 x-values followed by     2 y-values
• -.5 .5 -.5 .5
total +, - flow  9.743972E-01  0.000000E+00
plot     0.760 sec.
pde no., number of points, number of flow lines
• 0 0 0
```

```
1=triangles, 2=contours, 3=surface, 4=profiles, 5=flow lines, 6=settings
• 4
pde no., number of profiles
• 1 21
indices of profile planes
• 1 0
plot     0.910 sec.
pde no., number of profiles
• 0 0
1=triangles, 2=contours, 3=surface, 4=profiles, 5=flow lines, 6=settings
• 0
1=print, 2=plot, 3=iter, 4=refine, 5=comp, 6=save, 7=diff, 8=end, 9=change
• 8
```

Appendix D. Program to Solve Nonlinear System Example

```
      external aex, fex, gex
      integer io(9)
      parameter (irsize = 400000)
      parameter (idsize = 200000)
      common/cstak/dstak
      double precision dstak(idsize)
      real rstak(irsize)
      equivalence (dstak(1), rstak(1))
      save /cstak/
      call istkin(irsize, 3)
      io(1) = 5
      io(2) = 6
      io(3) = 125
      io(4) = 20
      io(5) = 21
      io(6) = 22
      io(7) = 23
      io(8) = 24
      io(9) = 25
      do 10 j = 1, 9
          call ioset(j, io(j))
   10 continue
      open(unit = io(4), file = 'input2.dat', status = 'old')
      open(unit = io(5), file = 'plot.dat' , status = 'unknown')
      open(unit = io(6), file = 'print.dat' , status = 'unknown')
      if ( io(7) .gt. 0) then
          open(unit = io(7), file = 'insol.dat', status = 'unknown',
     *        form='unformatted')
      else
          open(unit = -io(7), file = 'insol.dat', status = 'unknown',
     *        form='formatted')
      endif
      if ( io(8) .gt. 0) then
          open(unit = io(8), file = 'outsol.dat', status = 'unknown',
     *        form='unformatted')
      else
          open(unit = -io(8), file = 'outsol.dat', status = 'unknown',
     *        form='formatted')
      endif
      open(unit = io(9), file = 'scrtch.dat', status = 'unknown',
     *     form='unformatted')
      call drvall(aex, fex, gex)
      close(io(4))
      close(io(5))
      close(io(6))
      close(iabs(io(7)))
      close(iabs(io(8)))
   close(io(9))
   stop
   end
   subroutine aex(npde, ndim, dopde, x, y, i, u, dudx, dudy,
  *                 a, dadu, dadux, daduy)
      real x, y, u(*), dudx(*), dudy(*),
  *          a(*), dadu(ndim,*), dadux(ndim,*), daduy(ndim,*)
      logical dopde(npde)
   do 10 j = 1, npde
      do 10 k = 1, npde
         dadu (j, k) = 0.0
         dadux(j, k) = 0.0
         daduy(j, k) = 0.0
10 continue
   a(1) = 1.0 + dudx(2)**2 + dudy(2)**2 + dudx(3)**2 + dudy(3)**2
   dadux(1, 2) = 2.0*dudx(2)
   daduy(1, 2) = 2.0*dudy(2)
   dadux(1, 3) = 2.0*dudx(3)
   daduy(1, 3) = 2.0*dudy(3)
   a(2) = 1.0
   a(3) = u(1)*u(2)
   dadu(3, 1) = u(2)
   dadu(3, 2) = u(1)
   return
   end
   subroutine fex(npde, ndim, dopde, x, y, i, u, dudx, dudy,
  *                 f, dfdu, dfdux, dfduy)
      real x, y, u(*), dudx(*), dudy(*),
  *          f(*), dfdu(ndim,*), dfdux(ndim,*), dfduy(ndim,*)
   logical dopde(npde)
   do 10 j = 1, npde
      do 10 k = 1, npde
         dfdu (j, k) = 0.0
         dfdux(j, k) = 0.0
         dfduy(j, k) = 0.0
10 continue
   f(1) = 2.0*u(1)
   dfdu(1, 1) = 2.0
   f(2) = u(2)**2 * exp(-x)
   dfdu(2, 2) = 2.0 * u(2) * exp(-x)
   f(3) = 0.0
   return
   end
   subroutine gex(npde, ndim, dopde, x, y, i, j, u, g, dgdu)
      real x, y, u(*), g(*), dgdu(ndim,*)
      logical dopde(npde)
   g(1) = u(1)*u(2)*u(3) - 1.0
   dgdu(1, 1) = u(2)*u(3)
```

```
      dgdu(1, 2) = u(1)*u(3)
      dgdu(1, 3) = u(1)*u(2)
      g(2) = u(2) - exp(x)
      dgdu(2, 2) = 1.0
      g(3) = u(3) - exp(y)
      dgdu(3, 3) = 1.0
      return
      end
      subroutine usrgss(npde, x, y, i, u)
          real u(*)
      do 10 n = 1, npde
        u(n) = 1.0
   10 continue
      return
      end
      subroutine usrshf(ixtype, npde, nv, u, unew, vx, vy)
          real u(npde,nv), unew(nv), vx(nv), vy(nv)
      if (ixtype .le. npde) then
          nth = ixtype
          do 10 i = 1, nv
              unew(i) = u(nth, i) - utrue(nth, vx(i), vy(i))
   10     continue
      else
          do 20 i = 1, nv
              unew(i) = u(1,i) * u(2,i) * u(3,i)
   20     continue
      endif
      return
      end
      real function utrue(nth, x, y)
          real x, y
      if (nth .eq. 1) utrue = exp(-x-y)
      if (nth .eq. 2) utrue = exp(x)
      if (nth .eq. 3) utrue = exp(y)
      return
      end
```

Appendix E. File "input2.dat" for Nonlinear System Example

```
# two squares
# dirichlet b.c. on all sides
0    # old solution flag
# nt nv nc nb npde
   4  6  0  6   3
# vertices
# i lvl x y
 1 1 -1.0  1.0
 2 1  0.0  1.0
 3 1  1.0  1.0
 4 3 -1.0  0.0
 5 1  0.0  0.0
 6 1  1.0  0.0
# curved edge centers
# (none)
# triangles
# i v1 v2 v3
1 1 2 4
2 2 3 5
3 2 4 5
4 3 5 6
# boundary conditions:
# i triangle edge curve b.c.
 1   1 2 0 1 1 1
 2   1 3 0 1 1 1
 3   2 3 0 1 1 1
 4   4 2 0 1 1 1
 5   4 1 0 1 1 1
 6   3 1 0 1 1 1
2 8 0   # maxm maxr ittype
4 0 7   # lvlref lvlusr lvlmax
20      # threshold percentage
1       # print level
350000  # work space
```

Appendix F. Transcript of interactive session for Nonlinear System Example

Output from the computer is in typewriter font. Numbers typed by the user are in boldface and have a • at the beginning of the line.

```
refsol entered at level   1     6 vertices
grid           16 vertices
1=print, 2=plot, 3=iter, 4=refine, 5=comp, 6=save, 7=diff, 8=end, 9=change
• 3
enter niter, ieps, relerr, abserr, jsmax, param
• 10 0 0 .1 5 0
Iter   1
 1 newton norm,solution norm,relerr   3.437E+00   1.000E+00   3.437E+00
 2 newton norm,solution norm,relerr   1.718E+00   1.000E+00   1.718E+00
 3 newton norm,solution norm,relerr   1.718E+00   1.000E+00   1.718E+00
using newton step fraction   2.500E-01
Iter   2
 1 newton norm,solution norm,relerr   6.817E-01   1.158E+00   5.887E-01
 2 newton norm,solution norm,relerr   1.289E+00   1.430E+00   9.015E-01
 3 newton norm,solution norm,relerr   1.289E+00   1.430E+00   9.015E-01
Iter   3
 1 newton norm,solution norm,relerr   8.785E-01   1.840E+00   4.775E-01
 2 newton norm,solution norm,relerr   8.669E-03   2.718E+00   3.189E-03
 3 newton norm,solution norm,relerr   5.293E-02   2.718E+00   1.947E-02
Iter   4
 1 newton norm,solution norm,relerr   5.897E-03   2.718E+00   2.169E-03
 2 newton norm,solution norm,relerr   4.080E-06   2.718E+00   1.501E-06
 3 newton norm,solution norm,relerr   1.173E-02   2.718E+00   4.315E-03
1=print, 2=plot, 3=iter, 4=refine, 5=comp, 6=save, 7=diff, 8=end, 9=change
• 4
refsol re-entered at level   2    16 vertices
  1 estimated maximum error   7.12E-02
  2 estimated maximum error   6.13E-02
  3 estimated maximum error   6.38E-02
estimated individual PDE errors
0=continue, 1=print, 2=plot
• 0
estimated total error (as PDE #1)
0=continue, 1=print, 2=plot
• 0
refine to level   2      27 vertices
1=print, 2=plot, 3=iter, 4=refine, 5=comp, 6=save, 7=diff, 8=end, 9=change
• 5
compressed to level  1
1=print, 2=plot, 3=iter, 4=refine, 5=comp, 6=save, 7=diff, 8=end, 9=change
• 3
enter niter, ieps, relerr, abserr, jsmax, param
• 10 0 0 .01 5 0
Iter   1
 1 newton norm,solution norm,relerr   2.014E-01   2.718E+00   7.411E-02
 2 newton norm,solution norm,relerr   2.140E-01   2.718E+00   7.874E-02
 3 newton norm,solution norm,relerr   2.104E-01   2.718E+00   7.741E-02
Iter   2
 1 newton norm,solution norm,relerr   1.007E-02   2.718E+00   3.706E-03
 2 newton norm,solution norm,relerr   9.477E-04   2.718E+00   3.486E-04
 3 newton norm,solution norm,relerr   3.569E-02   2.718E+00   1.313E-02
Iter   3
 1 newton norm,solution norm,relerr   7.878E-05   2.718E+00   2.898E-05
 2 newton norm,solution norm,relerr   4.151E-08   2.718E+00   1.527E-08
 3 newton norm,solution norm,relerr   1.327E-04   2.718E+00   4.883E-05
1=print, 2=plot, 3=iter, 4=refine, 5=comp, 6=save, 7=diff, 8=end, 9=change
• 4
refsol re-entered at level   2    27 vertices
  1 estimated maximum error   1.70E-02
  2 estimated maximum error   6.18E-02
  3 estimated maximum error   7.45E-02
estimated individual PDE errors
0=continue, 1=print, 2=plot
• 0
estimated total error (as PDE #1)
0=continue, 1=print, 2=plot
• 0
refine to level   2     45 vertices
1=print, 2=plot, 3=iter, 4=refine, 5=comp, 6=save, 7=diff, 8=end, 9=change
• 3
enter niter, ieps, relerr, abserr, jsmax, param
• 10 10 0 .001 5 0
Iter   1
Chkmtx   135 equations. Worst equation    10   9.894E-02
This set of equations may be unsuitable for Gauss-Seidel iteration.
 1 newton norm,solution norm,relerr   7.858E-02   2.718E+00   2.891E-02
 2 newton norm,solution norm,relerr   6.731E-02   2.718E+00   2.476E-02
 3 newton norm,solution norm,relerr   6.650E-02   2.718E+00   2.446E-02
Iter   2
Chkmtx   135 equations. Worst equation    10   9.894E-02
This set of equations may be unsuitable for Gauss-Seidel iteration.
 1 newton norm,solution norm,relerr   1.728E-03   2.718E+00   6.357E-04
 2 newton norm,solution norm,relerr   9.014E-05   2.718E+00   3.316E-05
 3 newton norm,solution norm,relerr   4.439E-03   2.718E+00   1.633E-03
Iter   3
Chkmtx   135 equations. Worst equation    10   9.894E-02
This set of equations may be unsuitable for Gauss-Seidel iteration.
 1 newton norm,solution norm,relerr   6.482E-06   2.718E+00   2.385E-06
 2 newton norm,solution norm,relerr   1.491E-07   2.718E+00   5.485E-08
 3 newton norm,solution norm,relerr   1.452E-05   2.718E+00   5.343E-06
1=print, 2=plot, 3=iter, 4=refine, 5=comp, 6=save, 7=diff, 8=end, 9=change
• 1
1=summary, 2=timing, 3=storage, 4=vertices, 5=pde variables, 6=settings
```

```
• 1
  n lvl  vertices triangles      unorm      enorm  digits
  1   1        27        50   9.40E-01   3.25E-02    1.46
  2   1        27        50   1.40E+00   6.81E-02    1.31
  3   1        27        50   1.84E+00   7.25E-02    1.40
  1   2        45        90
  2   2        45        90
  3   2        45        90
1=summary, 2=timing, 3=storage, 4=vertices, 5=pde variables, 6=settings
• 2
        timing distribution
            seconds  percent
plot          0.000     0.00
assembly      7.980    50.44
factor        2.480    15.68
mg            1.210     7.65
rhs check     3.080    19.47
refine        1.070     6.76
total        15.820   100.00
1=summary, 2=timing, 3=storage, 4=vertices, 5=pde variables, 6=settings
• 3
        45  vertices used.
      1710  vertices allowed.
        97 triangles used.
      4559 triangles allowed.
1=summary, 2=timing, 3=storage, 4=vertices, 5=pde variables, 6=settings
• 0
1=print, 2=plot, 3=iter, 4=refine, 5=comp, 6=save, 7=diff, 8=end, 9=change
• 2
1=triangles, 2=contours, 3=surface, 4=profiles, 5=flow lines, 6=settings
• 1
enter level
• 1
plot     0.150 sec.
enter level
• 2
plot     0.220 sec.
enter level
• 0
1=triangles, 2=contours, 3=surface, 4=profiles, 5=flow lines, 6=settings
• 6
1=color, 2=erase, 3=magnify, 4=window, 5=file print/plot, 6=transform
• 6
enter index of transformation
• 2
transformation number        2 as PDE number 1
1=color, 2=erase, 3=magnify, 4=window, 5=file print/plot, 6=transform
• 0
```

```
1=triangles, 2=contours, 3=surface, 4=profiles, 5=flow lines, 6=settings
• 3
pde no., 1=function/2=abs(grad)
• 1 1
level, 1=surface triangles, 2=surface contours, 3=surface grids
• 2 1
viewing direction -- nx, ny, nz
• 1 -2 1
min and max values -1.334548E-03  0.000000E+00
plot     0.940 sec.
pde no., 1=function/2=abs(grad)
• 1 1
level, 1=surface triangles, 2=surface contours, 3=surface grids
• -2 2
number of contours
• 9
viewing direction -- nx, ny, nz
• 1 -2 1
min and max values -1.334548E-03  0.000000E+00
contour values
  -1.201E-03  -1.068E-03  -9.342E-04  -8.007E-04  -6.673E-04
  -5.338E-04  -4.004E-04  -2.669E-04  -1.335E-04
plot     0.460 sec.
pde no., 1=function/2=abs(grad)
• 1 1
level, 1=surface triangles, 2=surface contours, 3=surface grids
• 2 3
number of x lines, number of y lines
• 21 11
viewing direction -- nx, ny, nz
• 1 -2 1
min and max values -1.334548E-03  0.000000E+00
plot     1.610 sec.
pde no., 1=function/2=abs(grad)
• 0 0
1=triangles, 2=contours, 3=surface, 4=profiles, 5=flow lines, 6=settings
• 0
1=print, 2=plot, 3=iter, 4=refine, 5=comp, 6=save, 7=diff, 8=end, 9=change
• 8
```

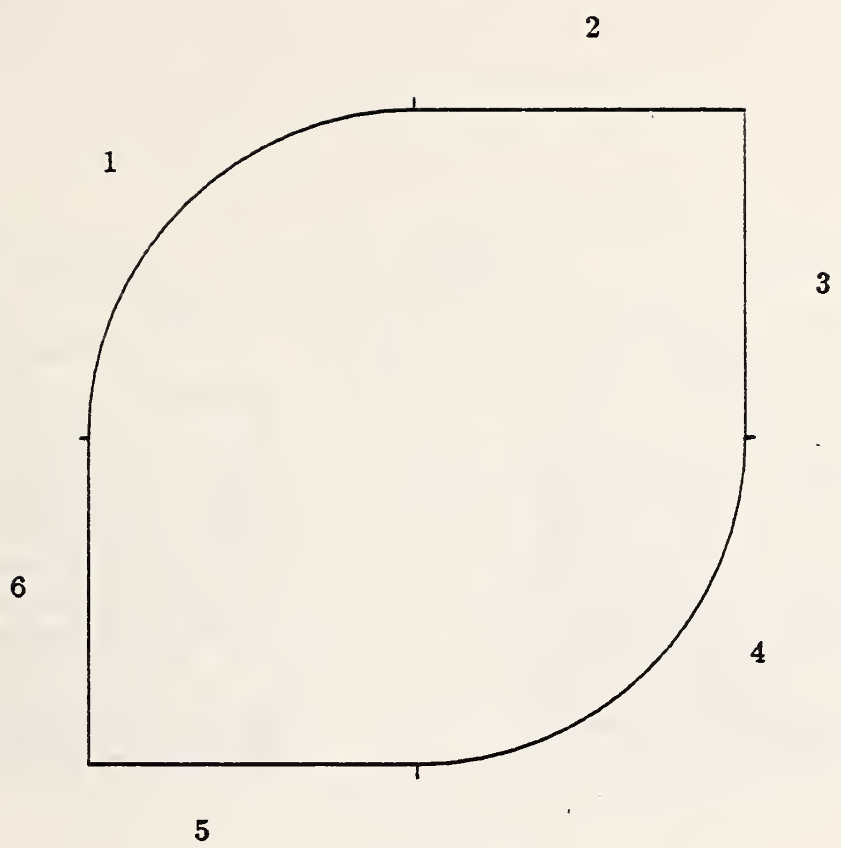


Figure 1

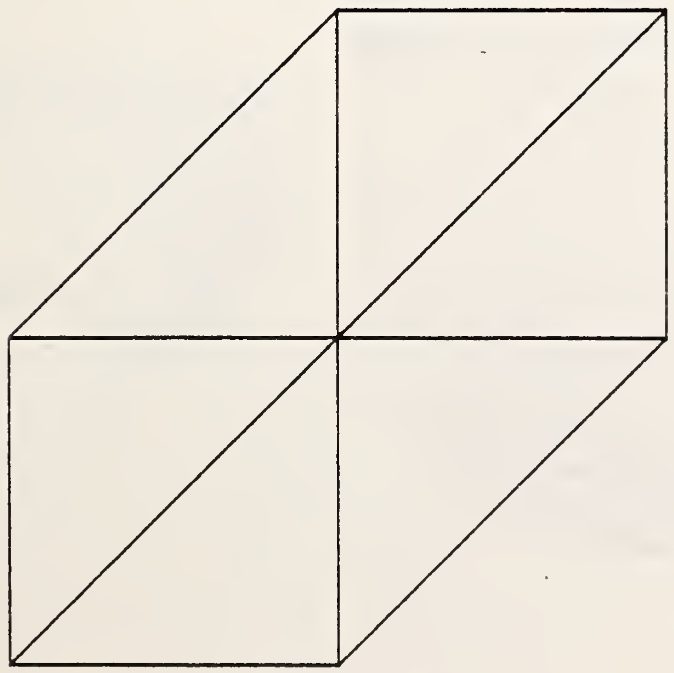
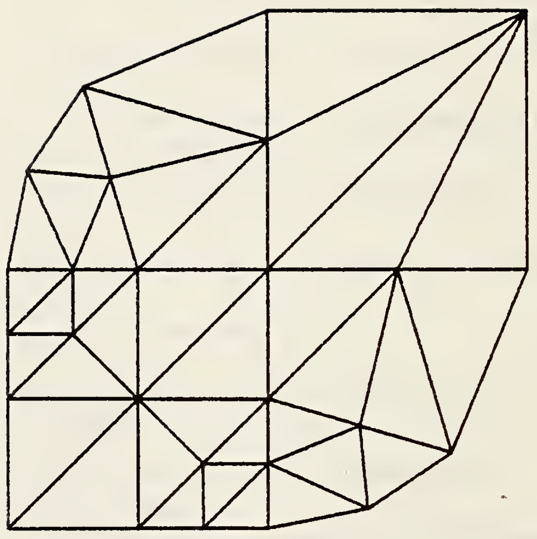
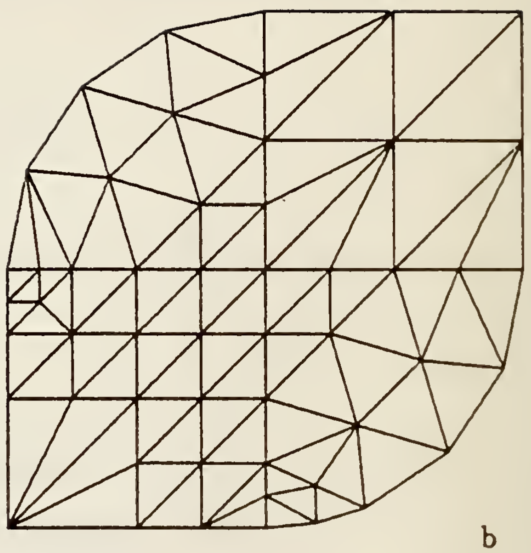


Figure 2


a


b

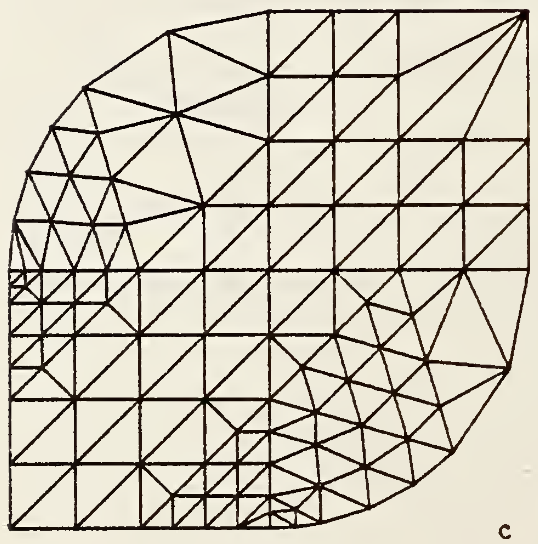
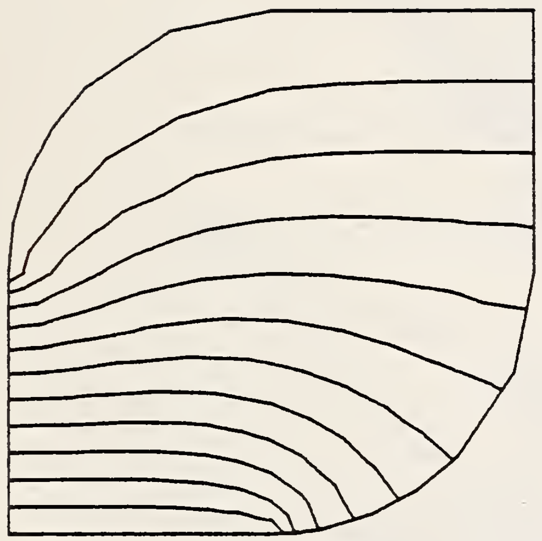
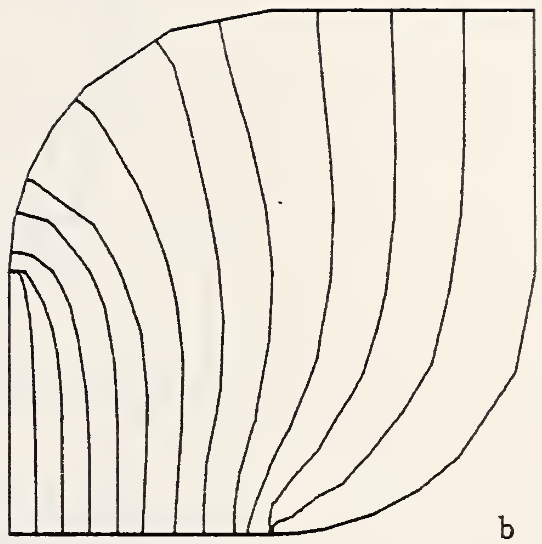

c

Figure 3


a


b

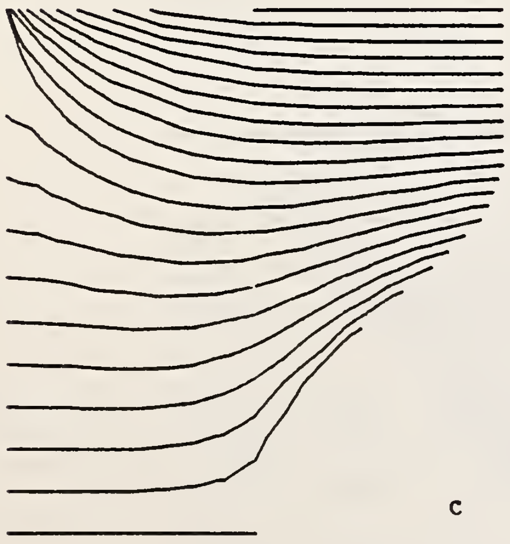

c

Figure 4

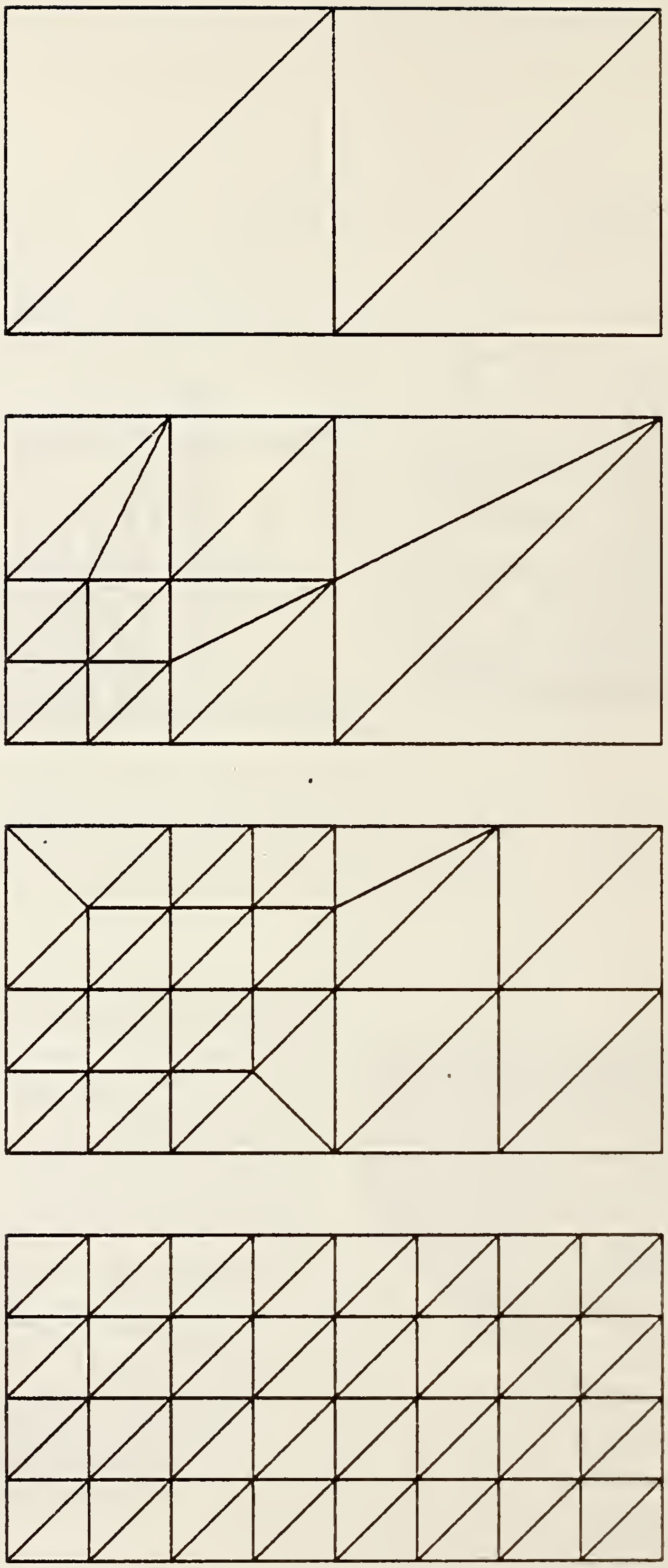


Figure 5

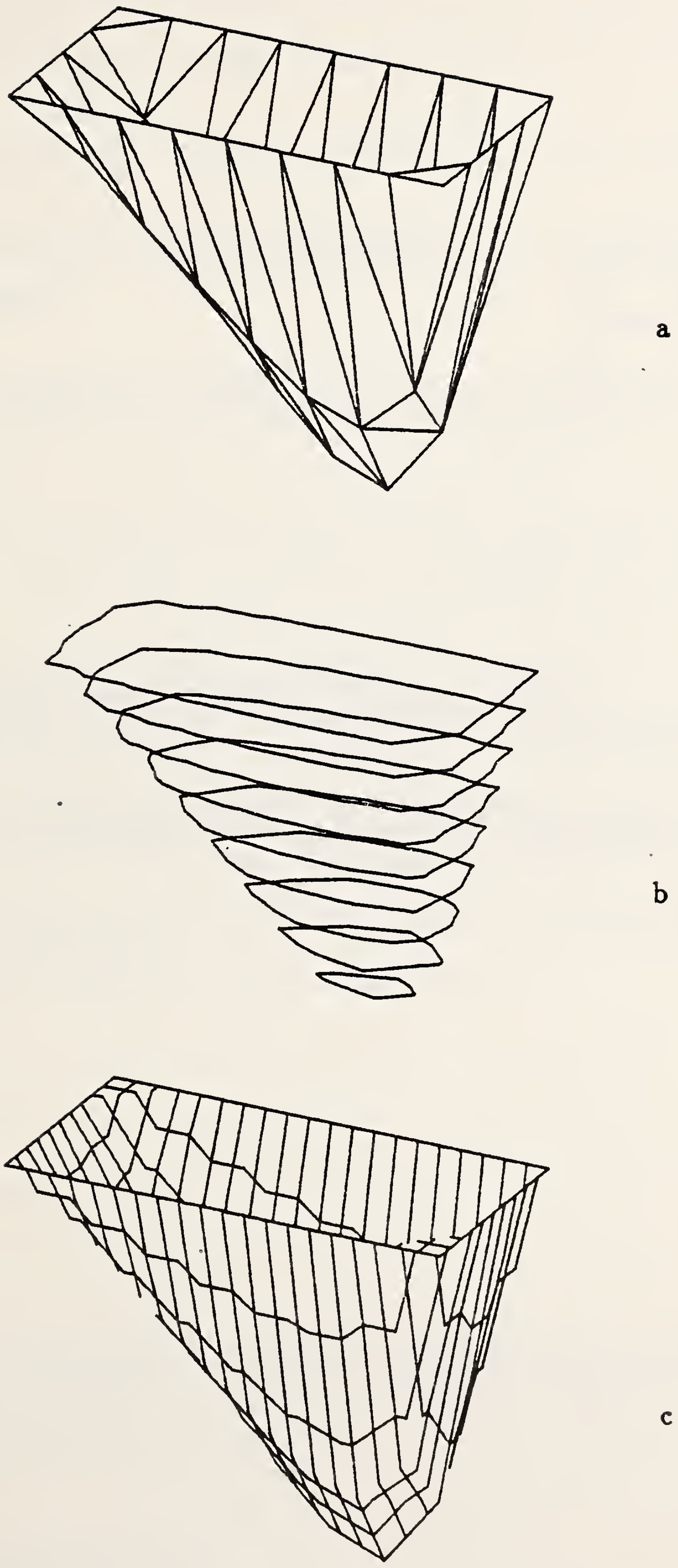


Figure 6

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. **BIBLIOGRAPHIC DATA SHEET** *(See instructions)*	1. PUBLICATION OR REPORT NO. NBSIR 86-3411	2. Performing Organ. Report No.	3. Publication Date JULY 1986

4. TITLE AND SUBTITLE

B2DE - A Program for Solving Systems of Partial Differential Equations in Two Dimensions

5. AUTHOR(S)

James L. Blue

6. PERFORMING ORGANIZATION *(If joint or other than NBS, see instructions)*

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS *(Street, City, State, ZIP)*

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)*

B2DE is a program for solving systems of nonlinear elliptic partial differential equations (PDEs) in two dimensions. The program is a collection of modules with an interactive driver. Many types of interactive graphics plots are included.
Users may modify the driver, and may be able to construct a "black box" program for a restricted class of PDEs.
B2DE is available from the author at the above address.

12. KEY WORDS *(Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)*

Adaptive mesh; elliptic; graphics; interactive; nonlinear equations; partial differential equations

13. AVAILABILITY

[✓ X] Unlimited
☐ For Official Distribution. Do Not Release to NTIS
☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.
[✓ X] Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

62

15. Price

$9.95