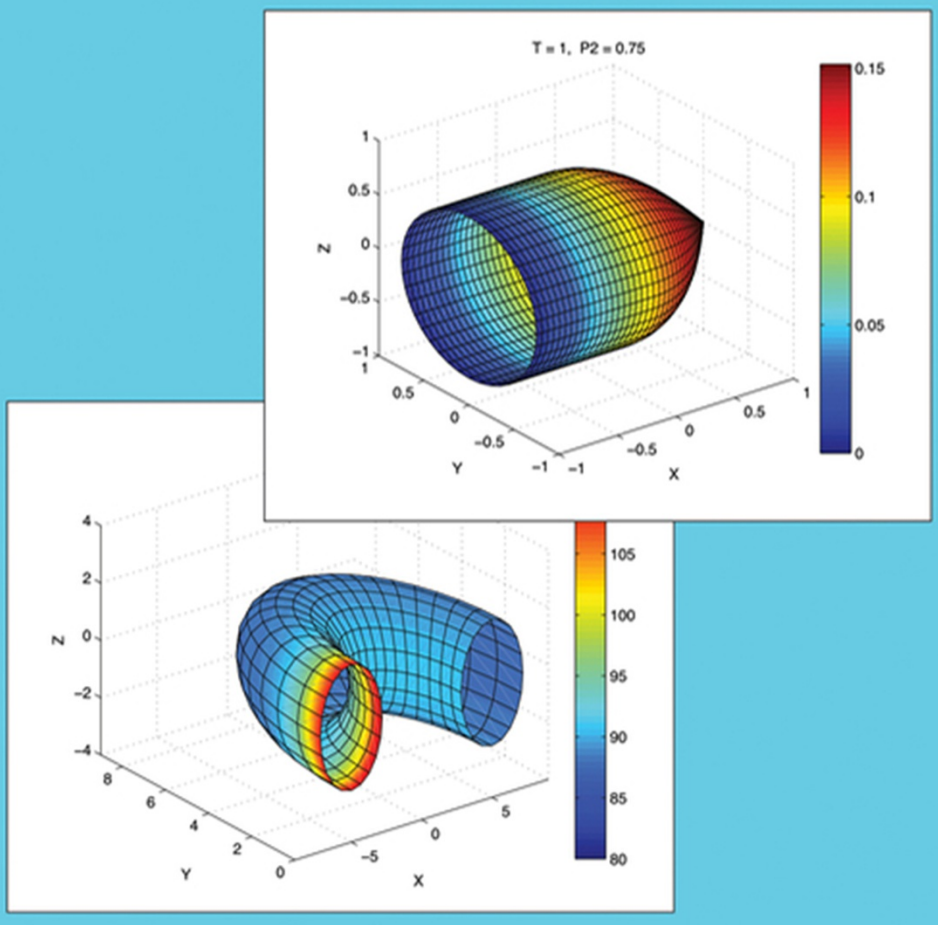
Solving Partial Differential Equation Applications with PDE2D

Granville Sewell



WILEY

Solving Partial Differential Equation Applications with PDE2D

Solving Partial Differential Equation Applications with PDE2D

Granville Sewell

Mathematics Department
University of Texas El Paso
El Paso, TX, USA

WILEY

This edition first published 2018

Registered Office
John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, USA

Editorial Office
111 River Street, Hoboken, NJ 07030, USA

For details of our global editorial offices, customer services, and more information about Wiley products visit us at www.wiley.com.

Wiley also publishes its books in a variety of electronic formats and by print-on-demand. Some content that appears in standard print versions of this book may not be available in other formats.

Library of Congress Cataloging-in-Publication Data

Names: Sewell, Granville, author.
Title: Solving partial differential equation applications with PDE2D / Granville Sewell.
Description: Hoboken, NJ : John Wiley & Sons, 2018. | Includes bibliographical references and index. |
Identifiers: LCCN 2018018589 (print) | LCCN 2018031767 (ebook) | ISBN 9781119507956 (Adobe PDF) | ISBN 9781119507963 (ePub) | ISBN 9781119507932 (hardcover)
Subjects: LCSH: Differential equations, Partial–Numerical solutions–Computer programs.
Classification: LCC QA377 (ebook) | LCC QA377 .S4643 2018 (print) | DDC 515/.353028553–dc23
LC record available at https://lccn.loc.gov/2018018589

Cover design by Wiley
Cover image: © Courtesy of Granville Sewell

Set in 10/12pt Warnock by SPi Global, Pondicherry, India

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Contents

Preface

This book looks at a wide range of ordinary and partial differential equation (PDE) applications. Students using this text will actually solve many interesting science and engineering applications using PDE2D, an easy-to-use, general-purpose PDE solver developed by the author over a 40-year period, which is free with the purchase of this book. They will learn to derive and solve the ordinary or partial differential equations, with boundary and initial conditions, for many time-dependent, steady-state, and eigenvalue applications, including diffusion, heat conduction and convection, image processing, math finance, fluid flow, elasticity, and quantum mechanics, in one, two, and three space dimensions. That PDE2D can be used to solve a wide variety of applications is evidenced by the list of over 250 journal articles and books at www.pde2d.com, in which it has been used to generate some or all of the numerical results.

Much of the material in the book was developed for two graduate courses, "Seminar in Applied Mathematics" at Texas A&M University and "Advanced Scientific Computing" at the University of Texas El Paso, but the book could also be used as a supplementary text for a number of science or engineering courses in which PDE applications are studied. It could also be used as a reference by individual students or researchers interested in using PDE2D to solve their specific applications.

Some documentation on the mathematical algorithms used by PDE2D can be found in Appendix B. The focus in this book, however, is on use of PDE2D, not the mathematics behind it, and students are not primarily learning about numerical methods, though they will learn some things, but rather about modeling real-world applications using differential equations.

The book starts with some simple ordinary differential equation problems in Chapter 1, which give the student a chance to become familiar with PDE2D before proceeding to more difficult problems, and ends with the solution in Chapter 9 of a very difficult nonlinear problem, which requires a moving adaptive grid because the solution has sharp, moving peaks.

The Windows/GFortran version of PDE2D is available at no cost with the book purchase: go to the "Free with Book" page at www.pde2d.com.

Linux and Mac OS X versions are available for a fee, also from www.pde2d.com. All three versions require the GNU GFortran compiler, but this can be downloaded from http://gcc.gnu.org/wiki/GFortranBinaries for free. Instructors using this book as a required text for a university class, and their students, can also obtain the Linux or Mac OS X version at no cost.

Each problem comes with some graphical or numerical output so the student can tell when his/her program is working correctly. The example programs can be downloaded from www.pde2d.com.

I

Introduction to PDE2D

I.1 The Collocation and Galerkin Finite Element Methods

PDE2D is a general-purpose partial differential equation (PDE) solver that solves very general systems of nonlinear, steady-state, time-dependent, and eigenvalue PDEs in 1D intervals, general 2D regions (see Figure I.1), and a wide range of simple 3D regions (see Figure I.2), with general boundary conditions.

PDE2D uses a collocation finite element method for 3D problems, and either a collocation or Galerkin finite element method can be used for 1D and 2D problems. This book is primarily concerned with using PDE2D to solve applications, so we will not attempt in this section to explain the mathematics behind either method. Readers who want to know more about the mathematics should read Appendix B and for more details, *The Numerical Solution of Ordinary and Partial Differential Equations* (*Third Edition*) (Sewell 2015). Here we are only concerned with helping users decide which method to use.

There are two major differences between the collocation and Galerkin methods, as far as users are concerned:

1) The Galerkin method requires that PDEs be in "divergence" form. For 2D steady-state problems, this means it must be possible to put them into the form:

$$\begin{aligned} &\frac{\partial}{\partial x}A_1(x,y,U1,U1_x,U1_y,...,UN,UN_x,UN_y) \\ +&\frac{\partial}{\partial y}B_1(x,y,U1,U1_x,U1_y,...,UN,UN_x,UN_y) = \\ &F_1(x,y,U1,U1_x,U1_y,...,UN,UN_x,UN_y) \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad . \quad = \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad . \quad = \end{aligned}$$

Solving Partial Differential Equation Applications with PDE2D, First Edition. Granville Sewell.

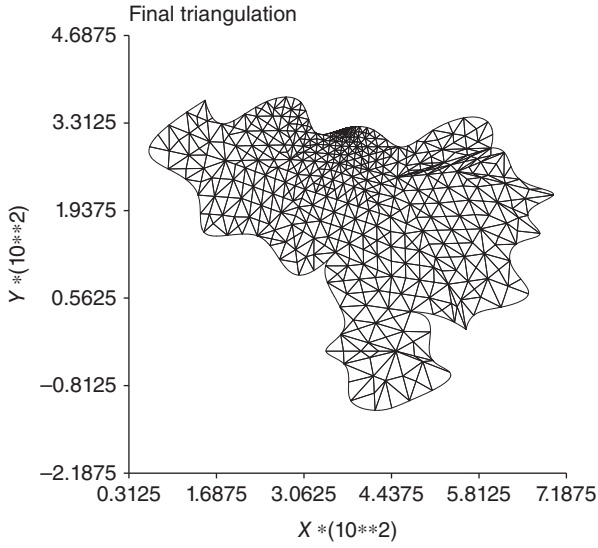


Figure I.1 Triangulation of Venezuela.

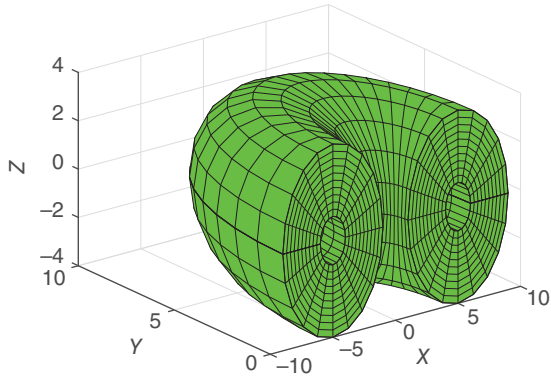


Figure I.2 3D grid, hollowed-out half torus.

$$\frac{\partial}{\partial x}A_N(x,y,U1,U1_x,U1_y,...,UN,UN_x,UN_y)$$
$$+\frac{\partial}{\partial y}B_N(x,y,U1,U1_x,U1_y,...,UN,UN_x,UN_y) =$$
$$F_N(x,y,U1,U1_x,U1_y,...,UN,UN_x,UN_y)$$

where $U1(x,y)$, ..., $UN(x,y)$ are the N unknown functions.

The boundary conditions are either "fixed",

$$\begin{aligned} U1 &= FB_1(x,y) \\ . &= . \\ . &= . \\ UN &= FB_N(x,y) \end{aligned}$$

or more general "free" conditions,

$$\begin{aligned} A_1N_x + B_1N_y &= GB_1(x,y,U1,U1_x,U1_y,...,UN,UN_x,UN_y) \\ . &= . \\ . &= . \\ A_NN_x + B_NN_y &= GB_N(x,y,U1,U1_x,U1_y,...,UN,UN_x,UN_y) \end{aligned}$$

where (N_x, N_y) is the unit outward normal vector on the boundary. The equations for time-dependent and eigenvalue problems, and 1D problems (listed in Appendix C), must be put into a similar divergence form.

New users sometimes find the Galerkin "free" boundary condition format confusing at first, especially when there are fixed and free boundary conditions on the same boundary, such as in Problem 2 of Chapter 2.

The collocation method allows a simpler, apparently more natural form, which for 2D steady-state problems means

$$\begin{aligned} F_1(x,y,U1,U1_x,U1_y,U1_{xx},U1_{yy},U1_{xy},U2,...) &= 0 \\ . &= . \\ . &= . \\ F_N(x,y,U1,U1_x,U1_y,U1_{xx},U1_{yy},U1_{xy},U2,...) &= 0 \end{aligned}$$

The boundary conditions also seem to be more natural:

$$\begin{aligned} G_1(x,y,U1,U1_x,U1_y,...,UN,UN_x,UN_y) &= 0 \\ . &= . \\ . &= . \\ G_N(x,y,U1,U1_x,U1_y,...,UN,UN_x,UN_y) &= 0 \end{aligned}$$

Periodic and "no" boundary conditions are also permitted.

The equations for time-dependent and eigenvalue problems, and 1D and 3D problems (listed in Appendix C), have similar (nondivergence) formats.

For problems such as the Black–Scholes equation (4.9), the collocation format is simpler. It requires some additional work to put the Black–Scholes equation into divergence form, and similar, but nonlinear, math finance equations are sometimes difficult or occasionally even impossible to put into divergence form, so the collocation method is preferred for problems such as these.

However, many applications, such as the elasticity equations (5.2) and the minimal surface equation (8.1), are already in divergence form, and it may require additional work to put them into a form suitable for the collocation method (see Problem 1 of Chapter 8). In fact, even the apparently unnatural Galerkin free boundary condition format is actually very natural for many applications, including elasticity problems with boundary forces given (5.5), which are much simpler to handle with the Galerkin finite element method. Furthermore, some problems, such as diffusion in a composite material (3.5), where the diffusion coefficient is discontinuous at an interface between materials, can be easily handled using the Galerkin method (see also Problem 6 of Chapter 1), while the collocation method cannot handle such composite problems at all.

2) The other major difference between the PDE2D Galerkin and collocation finite element methods, as far as users are concerned, is in the regions they can handle and how they are defined by the user. Since PDE2D only offers a collocation option for 3D problems, and for 1D problems the "regions" are just intervals (which both methods handle easily!), this difference is really only important for 2D problems.

For 3D problems, the PDE2D collocation method can only handle regions that can be parameterized (smoothly) as $X = X(p1, p2, p3), Y = Y(p1, p2, p3), Z = Z(p1, p2, p3)$, with constant limits on the parameters $p1, p2, p3$ (similarly for 2D problems). This means it can handle "a wide range of simple regions," but certainly not complicated regions such as Venezuela. The torus section of Figure I.2, used in Problem 4 of Chapter 3, was parameterized using the option ITRANS=−3 with equations

$$\begin{aligned} X &= (5 + p3 * cos(p2)) * cos(p1) \\ Y &= (5 + p3 * cos(p2)) * sin(p1) \\ Z &= p3 * sin(p2) \end{aligned} \tag{I.1}$$

where $p1$ (the toroidal angle) varies from 0 to π (only half of a torus is generated), $p2$ (the poloidal angle) varies from 0 to 2π, and $p3$ (the radial distance from centerline) varies from 1 to 4. Although regions are defined in terms of parameters $p1, p2, p3$, the PDEs, boundary conditions, and everything else are still defined using rectangular coordinates. For example, Laplace's equation can be written $Uxx + Uyy + Uzz = 0$ no matter how the region is parameterized.

If the collocation method is used to solve the minimal surface Problem 1 of Chapter 8, the following parameterization could be used, where $-1 \le p1 \le 1, -1 \le p2 \le 1$):

$$\begin{aligned} X &= p1 \\ Y &= p2 * (1 + p1^2) \end{aligned} \tag{I.2}$$

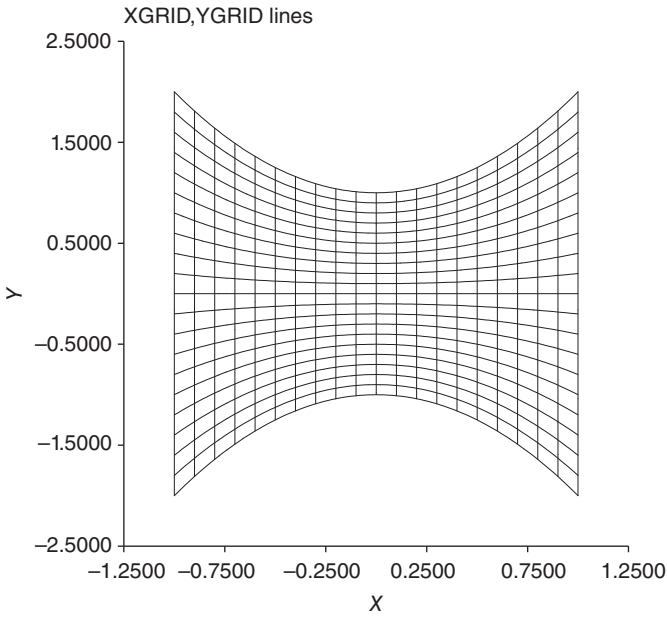


Figure I.3 Collocation grid for Problem 1, Chapter 8.

Figure I.3 shows the collocation grid generated for this problem, with $NP1GRID = NP2GRID = 21$.

The PDE2D Galerkin method, on the other hand, can solve problems in completely general 2D regions, with arbitrary curved boundaries (see Figure I.1). However, the user must (if the INTRI=3 option is chosen) supply an initial triangulation of the region, which normally consists of the minimal number of triangles needed to define the region. Inputting the initial triangulation is sometimes painstaking, but once the initial triangulation is defined, it can be automatically refined and graded by PDE2D. An initial triangulation of the minimal surface region of Figure I.3 is shown in Figure I.4.

However, if the region is rectangular or simple enough to be defined using parametric equations with constant limits on the parameters p, q (i.e. if it can be handled by the collocation method![1]), PDE2D can generate the initial triangulation automatically. Figure I.5 shows the initial triangulation generated by PDE2D if the option INTRI=2 is chosen and the parameterization (I.2) is used (with $p1, p2$ replaced by p, q). Notice that each grid square is divided into 4 triangles. For some regions, this will save the user a large amount of work.

1 Except that the parametric equations that define the region for the collocation method must be continuously differentiable, while they only need to be continuous for the Galerkin method.

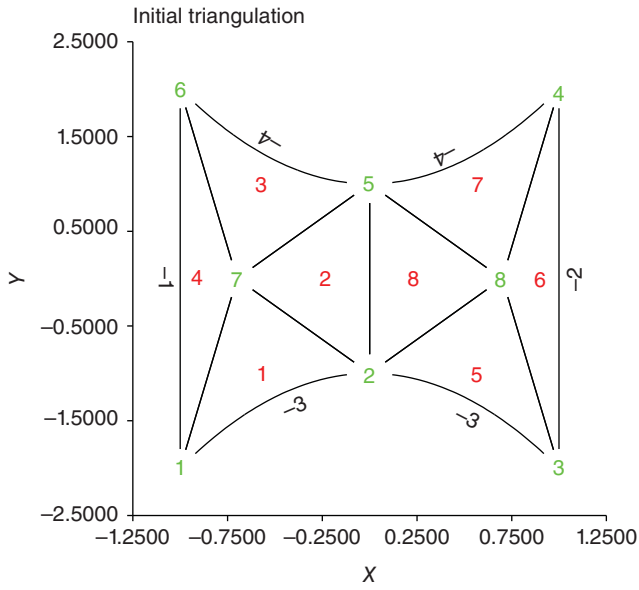


Figure I.4 Manually supplied initial triangulation (Galerkin method).

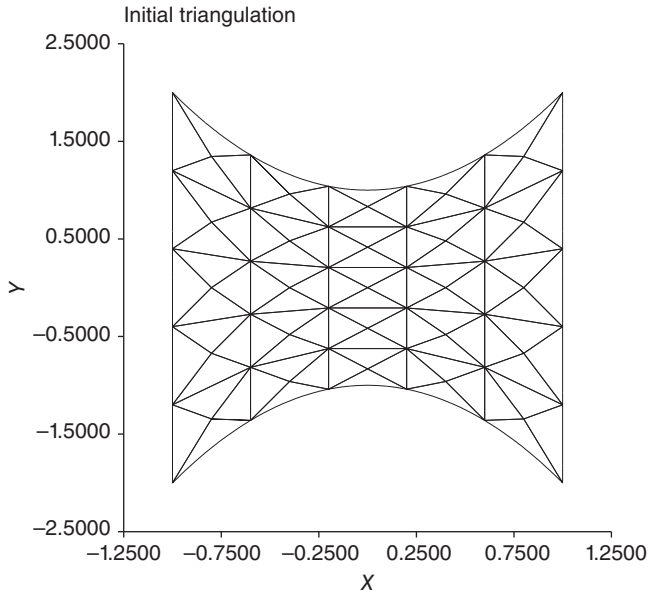


Figure I.5 Automatically generated initial triangulation (Galerkin method).

I.2 The PDE2D User Interfaces

There are two user interfaces, a graphical user interface (GUI) (Figure I.6, invoked by "pde2d_gui *name*"), which can be used to access the collocation finite element methods, and an interactive driver (Figure I.7, "pde2d *name*"), which can be used to access both the collocation and Galerkin finite element algorithms. The GUI is extremely easy to use, chooses more defaults, and

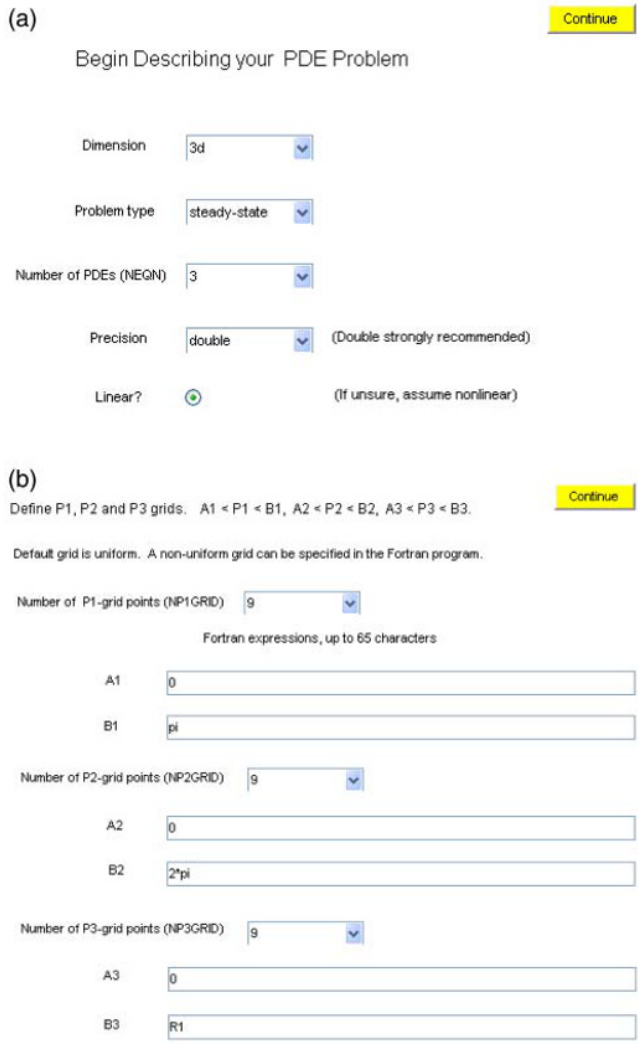


Figure I.6 (a and b) Two pages from GUI session.

```
     Now enter FORTRAN expressions to define the PDE coefficients, which
     may be functions of

                          X,Y,U,Ux,Uy,V,Vx,Vy

     They may also be functions of the initial triangle number KTRI
     and, in some cases, of the parameter T.

     Recall that the PDEs have the form

                    d/dX*A1 + d/dY*B1 = F1
                    d/dX*A2 + d/dY*B2 = F2

    Do you want to write a FORTRAN block to define some parameters to be
     used in the definition of these coefficients?
  |---- Enter yes or no
->yes
   Remember to begin FORTRAN statements in column 7
  |-----7-----Input FORTRAN now (type blank line to terminate)-----------|
->      Rmu = 0.1
->      Rho = 22.0
->      P = -alpha*(Ux+Vy)
->      f1 = y
->      f2 = -x

   F1 =         (Press [RETURN] to default to 0)
  |----Enter constant or FORTRAN expression----------------------|
-> Rho*(U*Ux+V*Uy) - f1
   A1 =         (Press [RETURN] to default to 0)
  |----Enter constant or FORTRAN expression----------------------|
-> 2*Rmu*Ux - P
   B1 =         (Press [RETURN] to default to 0)
  |----Enter constant or FORTRAN expression----------------------|
-> Rmu*(Uy+Vx)
   F2 =         (Press [RETURN] to default to 0)
  |----Enter constant or FORTRAN expression----------------------|
-> Rho*(U*Vx+V*Vy) - f2
   A2 =         (Press [RETURN] to default to 0)
  |----Enter constant or FORTRAN expression----------------------|
-> Rmu*(Uy+Vx)
   B2 =         (Press [RETURN] to default to 0)
  |----Enter constant or FORTRAN expression----------------------|
-> 2*Rmu*Vy - P
```

Figure I.7 Portion of PDE2D interactive session.

offers fewer options than the interactive driver interface, but cannot handle completely general 2D regions (because it only supports the collocation methods), and has some other limitations, for example, it cannot handle more than eight simultaneous PDEs. The GUI basically makes relatively easy problems extremely easy and is very useful in the classroom, but users who need to solve more difficult problems should create their programs using the interactive driver. There is a video at www.pde2d.com that is a good tutorial on how to use the GUI and interactive driver to create PDE2D programs.

Both the GUI and interactive driver interfaces produce a Fortran90 program (*name*.f), which is compiled and linked to the PDE2D library, and run ("runpde2d *name*"). Figure I.8 shows part of the Fortran program generated by the interactive session of Figure I.7. Typically, users work through a GUI or interactive driver session once and then make minor corrections or changes to the model directly in this Fortran program. The questions seen in either session are repeated in the Fortran comments, which makes it easy to modify the program to change options you selected or input you provided during the GUI or interactive session and to change many other options that were selected for you by default (especially if you used the GUI); only if major changes to the model are required is it necessary to work through a new session. The fact that a Fortran program is generated gives PDE2D, despite the user-friendliness of the human interface, all the flexibility of Fortran: for example, you can write Fortran functions to define any PDE and boundary condition coefficients (see Example 3.4, for example) or write your own Fortran postprocessing code. However, users do *not* need to be too familiar with Fortran, usually all they need to know is how to form Fortran expressions, such as $(X + 3 * Y) * Ux$, which are similar to MATLAB expressions, and many of the same intrinsic functions (sqrt,abs,sin,...) are available. The program is generated automatically, and normally users only need to modify the Fortran expressions they provided during the GUI or interactive driver session and do not need to touch the more complicated portions of the program.

Here is a short summary of a few of the more important additional features of PDE2D. More detail on each of these is contained in the remaining chapters, and especially in Appendix B.

1) PDE2D not only produces its own graphical output in a PostScript file ("*name*.ps") but also will automatically produce a MATLAB program ("pde2d.m"), which can be run to produce certain MATLAB plots, and can be easily modified to generate any other graphics MATLAB can create or otherwise postprocess the PDE2D results.
2) For 2D problems, using the Galerkin option, a user-supplied initial triangulation can be refined adaptively or graded according to user-supplied specifications (see Figure I.1).

```
                          else
C#############################################################################
C     Now enter FORTRAN expressions to define the PDE coefficients, which     #
C     may be functions of                                                     #
C                                                                             #
C                             X,Y,U,Ux,Uy,V,Vx,Vy                             #
C                                                                             #
C     They may also be functions of the initial triangle number KTRI          #
C     and, in some cases, of the parameter T.                                 #
C                                                                             #
C     Recall that the PDEs have the form                                      #
C                                                                             #
C                     d/dX*A1 + d/dY*B1 = F1                                  #
C                     d/dX*A2 + d/dY*B2 = F2                                  #
C                                                                             #
C#############################################################################
      Rmu = 0.1
      Rho = 22.0
      P = -alpha*(Ux+Vy)
      f1 = y
      f2 = -x
               if (j8z.eq.0) then
      yd8z = 0.0
C                                                     F1 DEFINED
      if (i8z.eq.    1) yd8z =
     & Rho*(U*Ux+V*Uy) - f1
C                                                     A1 DEFINED
      if (i8z.eq.    2) yd8z =
     & 2*Rmu*Ux - P
C                                                     B1 DEFINED
      if (i8z.eq.    3) yd8z =
     & Rmu*(Uy+Vx)
C                                                     F2 DEFINED
      if (i8z.eq.    4) yd8z =
     & Rho*(U*Vx+V*Vy) - f2
C                                                     A2 DEFINED
      if (i8z.eq.    5) yd8z =
     & Rmu*(Uy+Vx)
C                                                     B2 DEFINED
      if (i8z.eq.    6) yd8z =
     & 2*Rmu*Vy - P
               else
               endif
                         endif
```

Figure I.8 Portion of Fortran program created by interactive session.

3) For 2D problems, using the Galerkin option, curved boundaries can be defined by parametric equations, or a cubic spline can be drawn through user-supplied boundary points.
4) The Galerkin finite element method options use up to 4th degree elements, thus up to $O(h^5)$ accuracy (h=element diameter). The collocation options use cubic elements and thus generally provide $O(h^4)$ accuracy.

5) Newton's method is used to solve the nonlinear algebraic equations, for nonlinear problems.
6) The shifted inverse power method is used to find a single eigenvalue, with associated eigenfunction, of an eigenvalue PDE. If a direct solver is used, the LU decomposition that is calculated the first iteration is used to make the other iterations run much faster. If all eigenvalues are desired (without eigenfunctions), a shifted QR iteration is used.
7) Adaptive time step control is available for time-dependent problems.
8) For 2D and 3D problems, there are several options to solve the large linear systems, including iterative methods, sparse direct solvers, and frontal methods. On multiprocessor systems with MPI message passing software, PDE2D also offers parallel linear system solver options (see Section I.4).
9) The GUI and interactive driver do a lot of error checking, and there are many runtime error checks also.
10) Accurate integrals of user-supplied functions of the solution and its derivatives can be calculated, in a 1D, 2D, or 3D region. Accurate boundary integrals can also be calculated over the boundary of the region.
11) The solution can be dumped and restarted exactly if the same finite element grid is used on restart or dumped on a regular grid and interpolated to restart using a different finite element grid. There is also a function that reads and interpolates the solution saved by another PDE2D program.

I.3 Accuracy

The collocation method's use of cubic elements, and the Galerkin method's use of up to 4th degree isoparametric elements, allows PDE2D to solve PDEs accurately even in curved regions. To document this, we solved:

$U_{xx} + U_{yy} = 2U - (16x + 36y + 26)e^{x+y}$ in the region of Figure I.9
with $U = 0$ on the outer (elliptical) boundary
and $U = (36 - 4x^2 - 9y^2)e^{x+y}$ on the rest of the boundary.

which has exact solution $U^* = (36 - 4x^2 - 9y^2)e^{x+y}$.

Table I.1 shows the L_1 relative error (the integral of $|U - U^*|$ divided by the integral of $|U^*|$) using the Galerkin method with fourth degree elements; the linear systems were solved by a sparse direct method, based on Harwell's MA37 code (Duff and Reid 1984). Notice that the relative error produced using 64 000 triangles, in only 45 seconds, is about $1.3 * 10^{-11}$. Of course, it will not be possible to achieve this much precision on more difficult problems.

When 64 000 *linear* (IDEG=1) elements were used, the relative error was $3.12 * 10^{-4}$, twenty million times larger than with 64 000 quartic (IDEG=4) elements. Of course there are fewer unknowns with linear elements, so a fairer comparison would be with the 4 000 quartic triangle test, which required the solution of a problem with exactly the same number of unknowns but produced

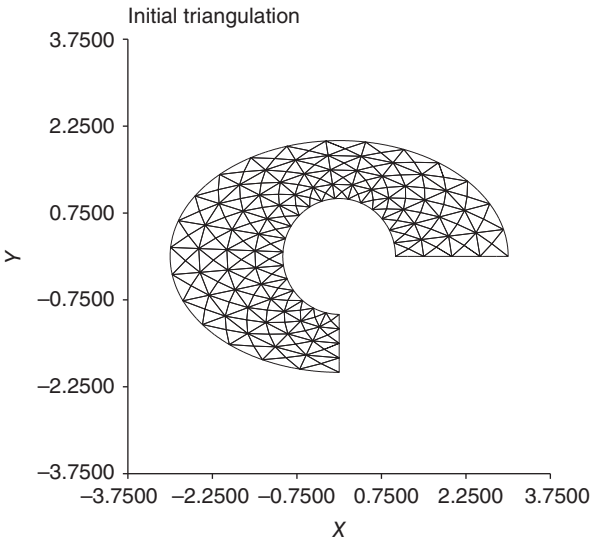


Figure I.9 Region used for accuracy tests.

Table I.1 Galerkin method (IDEG=4, ISOLVE=4).

Triangles	Unknowns	$\| U - U^* \|_1 / \| U^* \|_1$	CPU time (sec)	Memory (Mwords)
4 000	31 721	$8.07 * 10^{-9}$	0.8	2.8
16 000	127 441	$2.62 * 10^{-10}$	6.2	12.6
64 000	510 881	$1.31 * 10^{-11}$	45.7	57.0

Table I.2 Collocation method (ISOLVE=1).

Rectangles	Unknowns	$\| U - U^* \|_1 / \| U^* \|_1$	CPU time (s)	Memory (Mwords)
1 000	4 284	$1.06 * 10^{-6}$	1.4	0.8
4 000	16 564	$6.64 * 10^{-8}$	6.1	3.7
16 000	65 124	$4.06 * 10^{-9}$	28.5	18.3

an error 40 000 times smaller. When the region has a curved boundary, just using higher order elements does not improve on the asymptotic order ($O(h^2)$) of linear elements, unless isoparametric elements (see Section B.3) are used.

Table I.2 shows the relative errors when the collocation method was used. The PDE2D collocation method also produces high order ($O(h^4)$) accuracy in curved regions, because of the global transformation of coordinates used.