Advances in Engineering Software 41 (2010) 748–753

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Solving PDEs in non-rectangular 3D regions using a collocation finite element method

Granville Sewell [*]

Mathematics Department, University of Texas El Paso, El Paso, TX 79968, USA

A R T I C L E I N F O

Article history:
Received 21 January 2009
Accepted 14 December 2009
Available online 8 January 2010

Keywords:
Finite elements
Collocation
Partial differential equations

A B S T R A C T

The general-purpose partial differential equation (PDE) solver PDE2D uses a Galerkin finite element method, with standard triangular elements of up to fourth degree, to solve PDEs in general 2D regions. For 3D problems, a very different approach is used, which involves a collocation finite element method, with tricubic Hermite basis functions, and an *automatic* global coordinate transformation. If the user can define the 3D region by $X = X(P1, P2, P3)$, $Y = Y(P1, P2, P3)$, $Z = Z(P1, P2, P3)$ with constant limits on $P1$, $P2$, $P3$, then the PDEs and boundary conditions can be written in their usual Cartesian coordinate form and PDE2D will automatically convert the equations to the new coordinate system and solve the problem internally in this rectangle. The result is that for a wide range of simple 3D regions, once the global coordinate system is defined, the rest of the input is as simple as if the region were a rectangle.

1. Solving 2D problems with PDE2D

The author began development of a general-purpose partial differential equation solver in 1974 and has been working continuously on this project since that date (earliest Ref.: [9]). The program was marketed by IMSL (now VNI) from 1980 to 1984 under the name TWODEPEP [10], then from 1984 to 1991 as PDE/PROTRAN [11]. In 1991 the author developed a new version, with an interactive user interface, which has been sold under the name PDE2D since that date. In a 1993 *Advances in Engineering Software* article [12], the 2D algorithm, which uses the Galerkin finite element method with standard isoparametric triangular finite elements of up to fourth degree, was described in some detail. PDE2D solves very general, nonlinear steady-state, time-dependent and eigenvalue systems of PDEs, with general boundary conditions, in general 2D regions, with curved boundaries and even curved interfaces (see Fig. 1). The user supplies an initial triangulation with just enough triangles to define the region, and a graded or adaptively refined triangulation is generated automatically.

2. Solving 3D problems

In 1994 the author began development of a collocation finite element algorithm for 3D problems, which is quite different from the Galerkin method used for 2D problems. In particular, the way in which non-rectangular 3D regions are handled will be the primary focus of this article. This approach cannot handle completely general 3D regions, only "a wide range of simple 3D regions"; however, for those regions which can be handled, the PDE2D algorithm has some important advantages, particularly with respect to ease-of-use.

PDE2D solves three-dimensional, nonlinear, steady-state systems of the form:

$$F_1(x,y,z,U1,\dots U1_x,\dots U1_y,\dots U1_z,\dots U1_{xx},\dots U1_{yy},\dots U1_{zz},\\ \dots U1_{xy},\dots U1_{xz},\dots U1_{yz}) = 0,$$

$$\vdots,$$

$$F_M(x,y,z,U1,\dots U1_x,\dots U1_y,\dots U1_z,\dots U1_{xx},\dots U1_{yy},\dots U1_{zz},\\ \dots U1_{xy},\dots U1_{xz},\dots U1_{yz}) = 0,$$

in $xa \leqslant x \leqslant xb$, $ya \leqslant y \leqslant yb$, $za \leqslant z \leqslant zb$. It also handles the related time-dependent and eigenvalue problems, see [[13], Appendix A] for more detail.

Boundary conditions have the form:

$$G_1(x,y,z,U1,\dots,UM,U1_x,\dots,UM_x,U1_y,\dots,UM_y,U1_z,\dots,UM_z)\\ = 0,$$

$$\vdots,$$

$$G_M(x,y,z,U1,\dots,UM,U1_x,\dots,UM_x,U1_y,\dots,UM_y,U1_z,\dots,UM_z)\\ = 0.$$

Periodic and "no" boundary conditions are also permitted.

For these problems, PDE2D uses a collocation finite element method, with "tricubic Hermite" basis functions [7]. That is, each

[*] Tel.: +1 915 747 6762.
E-mail address: sewell@math.utep.edu

0965-9978/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.advengsoft.2009.12.015

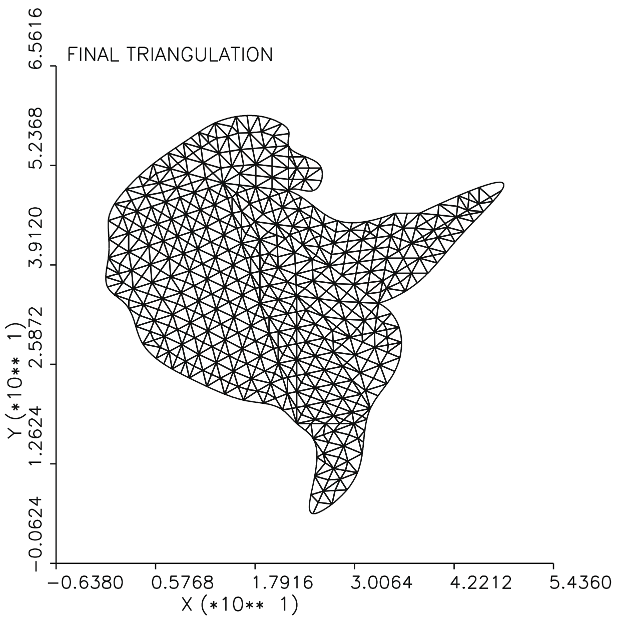


Fig. 1. Triangulation of Corpus Christi Bay.

unknown is assumed to be a linear combination of the 8 NXGRID NYGRID NZGRID basis functions:

$$H_i(x)H_j(y)H_k(z), \quad H_i(x)H_j(y)S_k(z),$$
$$H_i(x)S_j(y)H_k(z), \quad H_i(x)S_j(y)S_k(z),$$
$$S_i(x)H_j(y)H_k(z), \quad S_i(x)H_j(y)S_k(z),$$
$$S_i(x)S_j(y)H_k(z), \quad S_i(x)S_j(y)S_k(z)$$
$$(i = 1, \ldots, \text{NXGRID}, \; j = 1, \ldots, \text{NYGRID}, \; k = 1, \ldots, \text{NZGRID}),$$

where the cubic Hermite basis functions H_i and S_i are piecewise cubic polynomials with $H_i(x_j) = \delta_{ij}, \; H_i'(x_j) = 0$ and $S_i(x_j) = 0, \; S_i'(x_j) = \delta_{ij}$ ([13], p. 183), where the gridpoints $x_i, \; y_j, \; z_k$ are not necessarily uniformly distributed. This choice of basis function ensures that the first derivatives of the approximate solution are all continuous, as required by the collocation method ([14], p. 117). The approximate solution is required to satisfy the PDEs exactly at 8 collocation points $(x_i + \beta_{1,2}(x_{i+1} - x_i), \; y_j + \beta_{1,2}(y_{j+1} - y_j), \; z_k + \beta_{1,2}(z_{k+1} - z_k))$, where $\beta_1 = 0.5 - 0.5/\sqrt{3}, \beta_2 = 0.5 + 0.5/\sqrt{3}$, in each of the $(NXGRID - 1)$ $(NYGRID - 1)$ $(NZGRID - 1)$ subrectangles, and to satisfy the boundary conditions at certain boundary points. The number of boundary collocation points plus the number of interior collocation points is equal to the number of basis functions (8 *NXGRID NYGRID NZGRID*), so that the number of equations equals the number of unknowns ($N = 8$ *NXGRID NYGRID NZGRID M*).

PDE2D actually uses the collocation finite element method for 3D problems, and for 1D and 2D problems it offers both Galerkin and collocation FEM options. From the above problem description, it appears that the PDE2D collocation algorithm can only solve problems in rectangular boxes, but it can actually solve problems in any 2D or 3D region which can be described by parametric equations with constant limits on the parameters, as discussed in the next section, such as spheres, cylinders, tori, parallelopipeds, cones, ellipsoids, and many more, and for all such regions, high accuracy ($O(h^4)$) solutions are produced (h = maximum element diameter, see [14], p. 118), assuming appropriate smoothness in the solution.

Although the Galerkin method is easier to apply to general regions, the collocation method is easier to apply to more general PDEs and boundary conditions, because:

1. The user does not have to manipulate the equations into the "divergence" form required by the Galerkin method, that is, the form $(A)_x + (B)_y = F$ (for 2D), where $A, \; B, \; F$ may be functions of the solution components and their first derivatives. While the divergence form is natural for many physical applications, it is quite unnatural for others, for example, most math finance applications (eg, [15] p. 249). And it is easy to transform an equation *out* of divergence form (unless A or B are not smooth, then the Galerkin method is to be preferred) but often difficult, sometimes impossible, to transform an equation or system of equations *into* divergence form.
2. Even after a user has manipulated the equations into divergence form $(A)_x + (B)_y = F$, if the unknowns are not specified on the boundary (Dirichlet conditions) the Galerkin method requires "natural" boundary conditions of the form $An_x + Bn_y = g$, where (n_x, n_y) is the outward unit normal to the boundary. Again, these conditions are indeed "natural" for many physical applications, but not for general PDEs. For complicated systems of PDEs, it is often difficult or impossible to manipulate the boundary conditions into the "natural" form required by the Galerkin method.

For example, if Laplace's equation is solved, $A = U_x, \; B = U_y$, standard implementations of the Galerkin method allow only boundary conditions of the form $U = f(x, y)$ or $\frac{\partial U}{\partial n} = U_x n_x + U_y n_y = g(x, y, U)$. (We are assuming a 2D problem is solved.) If one wants to solve Laplace's equation with $U_x = 0$ on a boundary segment that is not an x =constant line, one is out of luck with most Galerkin-based FEM packages. (Actually, the PDE2D Galerkin method *can* solve such problems, because g is allowed to depend on U_x and U_y, but even with PDE2D the set-up of such boundary conditions is very unnatural; for the collocation method, this boundary condition can be simply entered as "$Ux = 0$", regardless of the form of the PDE.)

As with the Galerkin method, Newton's method is used to solve the nonlinear algebraic equations resulting from the collocation method formulation, and again there are several options available to solve the linear system each Newton iteration (or each time step, for time-dependent problems; or each shifted inverse power iteration, for eigenvalue problems). The linear systems generated by Galerkin finite element methods have symmetric nonzero structures, even when the matrices themselves are nonsymmetric. The systems generated by collocation finite element methods, on the other hand, do not even have symmetric nonzero structures, and almost all iterative and sparse direct solvers perform poorly on such systems, so solving these linear systems is challenging. PDE2D currently offers the following options:

1. The "normal" equations are formed by multiplying both sides of $A\mathbf{x} = \mathbf{b}$ by A^T, and the resulting symmetric, positive definite linear system is solved using the Harwell sparse direct solver MA27 [3], a minimal-degree algorithm. The matrix A^TA is still sparse, in fact the normal equations are essentially the equations that would result if a "least squares" finite element method were used, and the nonzero structure of these equations is exactly the same as for a Galerkin method. Although the normal equations are significantly more ill-conditioned than the original equations, this is rarely a problem in practice for MA27, provided double precision is used.
2. The original equations $A\mathbf{x} = \mathbf{b}$ are solved directly by a frontal method, which is basically an out-of-core band solver. This option is generally much slower than the others, but requires a very small amount of memory.
3. The normal equations are solved using a Jacobi conjugate gradient method, that is, a conjugate gradient method applied to $D^{-1}A^TA\mathbf{x} = D^{-1}A^T\mathbf{b}$, where D is the diagonal of A^TA. Since A^TA is always symmetric and positive-definite, convergence is

theoretically guaranteed (assuming infinite precision is used!), but in practice this iterative solver can converge slowly or not at all, because the condition number of A^TA can be large. On multi-processor systems, this method is "MPI-enhanced", that is, the matrix-vector multiplications are distributed over the available processors.
4. Both the PDE2D Galerkin (2D) and collocation (3D) algorithms have clean interfaces so that the user can easily "plug in" any linear system solver desired, to see if it performs better than the built-in options. Both the original and normal equations are available in sparse format, already distributed over the available processors, for multi-processor systems. In fact, an interface to the very efficient parallel sparse solver MUMPS ([1], http://mumps.enseeiht.fr) is provided. The plug-in interface makes PDE2D a useful tool for testing linear system software and algorithms on a wide range of symmetric and nonsymmetric problems. For an overview of modern algorithms and available software for the direct and iterative solution of linear systems, see [2,4–6,8].

3. Nonrectangular 3D regions

When PDE2D was generalized to handle three-dimensional PDE systems, the logical extension of the 2D algorithm would have been to use a Galerkin method with (possibly isoparametric) tetrahedral elements and require the user to supply an initial "tetrahedralization" of the region, thereby facilitating the solution of problems in general 3D regions. But developing a user interface for defining general regions and boundary conditions is a *much* more difficult problem in three dimensions than in two, for reasons that are obvious and well-known. The decision was made *initially*, therefore, to avoid the difficulties in handling general three-dimensional regions, and to develop software that could solve 3D PDE systems as general as those solved by the 2D algorithm, with comparable ease-of-use, but only in 3D boxes.

For 3D problems, then, a collocation finite element method was selected, with tricubic Hermite basis functions, because the fact that the Galerkin method is easier to apply to general regions was now not an issue, and the collocation method has some important advantages over the Galerkin brand with regard to ease-of-use, as discussed in the previous section.

However, after the abilities to handle periodic and "no" boundary conditions were added, it became possible to solve, with high accuracy, $O(h^4)$, problems in many simple non-rectangular domains, such as spheres, cylinders, tori, pyramids, ellipsoids, and cones, by writing the PDEs in terms of an appropriate system of variables with constant limits. However, rewriting the partial differential equations in the new coordinate system was often extremely unpleasant. For example, suppose we want to solve $\nabla^2 U = 1$, in a torus of major radius R_0 and minor radius R_1. A "toroidal" coordinate system can be used, where

$$\begin{aligned} X &= (R_0 + P3\cos(P2))\cos(P1), \\ Y &= (R_0 + P3\cos(P2))\sin(P1), \\ Z &= P3\sin(P2). \end{aligned} \tag{1}$$

Here X,Y, and Z are Cartesian coordinates, $P1$ is the major (toroidal) angle, $P2$ is the minor (polodial) angle, and $P3$ is radial distance from the torus centerline. In the new coordinate system the region is rectangular, because the limits on $P1,\ P2$, and $P3$ are constants, and PDE2D can be used to solve this problem. To convert the Laplacian, $U_{xx} + U_{yy} + U_{zz}$, to the new coordinate system, one has to use the chain rule; for example, U_{xx}, the second derivative of U with respect to X, is $\left(U_i = \frac{\partial U}{\partial Pi},\ U_{ij} = \frac{\partial^2 U}{\partial Pi \partial Pj}\right)$:

$$\begin{aligned} U_{xx} = {} & \left(U_{11}\frac{\partial P1}{\partial X} + U_{12}\frac{\partial P2}{\partial X} + U_{13}\frac{\partial P3}{\partial X}\right)\frac{\partial P1}{\partial X} \\ & + \left(U_{21}\frac{\partial P1}{\partial X} + U_{22}\frac{\partial P2}{\partial X} + U_{23}\frac{\partial P3}{\partial X}\right)\frac{\partial P2}{\partial X} \\ & + \left(U_{31}\frac{\partial P1}{\partial X} + U_{32}\frac{\partial P2}{\partial X} + U_{33}\frac{\partial P3}{\partial X}\right)\frac{\partial P3}{\partial X} \\ & + U_1\frac{\partial^2 P1}{\partial X^2} + U_2\frac{\partial^2 P2}{\partial X^2} + U_3\frac{\partial^2 P3}{\partial X^2}. \end{aligned}$$

Transforming PDEs into a new coordinate system can be quite difficult, however, as is apparent when any of the above partial derivative terms are displayed. For example, the last term is computed by *Mathematica*® as:

$$\begin{aligned} \frac{\partial^2 P3}{\partial X^2} = {} & -\frac{X^2(-R_0+\sqrt{X^2+Y^2})^2}{(X^2+Y^2)((-R_0+\sqrt{X^2+Y^2})^2+Z^2)^{3/2}} \\ & + \frac{X^2}{(X^2+Y^2)\sqrt{(-R_0+\sqrt{X^2+Y^2})^2+Z^2}} \\ & - \frac{X^2(-R_0+\sqrt{X^2+Y^2})}{(X^2+Y^2)^{3/2}\sqrt{(-R_0+\sqrt{X^2+Y^2})^2+Z^2}} \\ & + \frac{-R_0+\sqrt{X^2+Y^2}}{\sqrt{X^2+Y^2}\sqrt{(-R_0+\sqrt{X^2+Y^2})^2+Z^2}}. \end{aligned}$$

Fortunately, now the need for hand transformations has been removed: now the user only has to supply the global coordinate transformation equations (e.g., Eq. (1)) and can then write the PDEs in their usual Cartesian form. For example, the PDE above would be written simply as $U_{xx} + U_{yy} + U_{zz} = 1$; PDE2D will automatically compute $U_{xx} + U_{yy} + U_{zz}$ in terms of $U_{11}, U_{12}, \ldots$ using the chain rule, and solve the problem internally in the $P1,\ P2,\ P3$ coordinate system.

If a cylindrical or spherical coordinate system is used, PDE2D automatically supplies the first and second derivatives of $P1,\ P2,\ P3$ with respect to $X,\ Y,\ Z$, required to apply the chain rule; the user only has to indicate that $P1,\ P2,\ P3$ represent cylindrical or spherical coordinates. If another user-specified system is used, such as toroidal coordinates, PDE2D will use finite differences to compute the first and second derivatives of $X,\ Y,\ Z$ with respect to $P1,\ P2,\ P3$; alternatively, the user can supply these analytically. Then PDE2D computes the required derivatives of $P1,\ P2,\ P3$ with respect to $X,\ Y,\ Z$ from these. For the first derivatives, the conversion uses the fact that the Jacobian matrices for the forward and inverse transforms are inverses:

$$J \equiv \begin{bmatrix} \frac{\partial P1}{\partial X} & \frac{\partial P1}{\partial Y} & \frac{\partial P1}{\partial Z} \\ \frac{\partial P2}{\partial X} & \frac{\partial P2}{\partial Y} & \frac{\partial P2}{\partial Z} \\ \frac{\partial P3}{\partial X} & \frac{\partial P3}{\partial Y} & \frac{\partial P3}{\partial Z} \end{bmatrix} = \begin{bmatrix} \frac{\partial X}{\partial P1} & \frac{\partial X}{\partial P2} & \frac{\partial X}{\partial P3} \\ \frac{\partial Y}{\partial P1} & \frac{\partial Y}{\partial P2} & \frac{\partial Y}{\partial P3} \\ \frac{\partial Z}{\partial P1} & \frac{\partial Z}{\partial P2} & \frac{\partial Z}{\partial P3} \end{bmatrix}^{-1}$$

The second derivatives are computed using ($Pi = P1,\ P2,\ P3$):

$$Pi_H = -J^T\left[\frac{\partial Pi}{\partial X}X_H + \frac{\partial Pi}{\partial Y}Y_H + \frac{\partial Pi}{\partial Z}Z_H\right]J$$

where the subscript H denotes a Hessian matrix. (This formula can be derived directly from the chain rule.) It should be noted that it is *essential* that all first derivatives of $X,\ Y,\ Z$ with respect to $P1,\ P2,\ P3$ be continuous.

What sorts of regions can be parameterized, with constant limits on the parameters? We have listed a few simple regions, such as spheres and cylinders and tori, which have simple parameterizations, but the class of 3D regions that can be handled is much larger than this list might suggest. For example, any region of the form:

$$\begin{aligned} &A \leqslant X \leqslant B \\ &C(X) \leqslant Y \leqslant D(X) \\ &E(X,Y) \leqslant Z \leqslant F(X,Y) \end{aligned} \tag{2}$$

where C, D, E and F are arbitrary smooth functions, can be easily parameterized as follows:

$$\begin{aligned} X &= A + P1(B - A) \\ Y &= C(X) + P2(D(X) - C(X)) \\ Z &= E(X,Y) + P3(F(X,Y) - E(X,Y)) \end{aligned}$$

with limits of $(0,1)$ on $P1, P2$ and $P3$. Replace X, Y, Z in (2) by, for example, the spherical coordinates ρ, ϕ, θ and you have another set of regions in which PDE2D can solve problems.

However, there are also many 3D regions which PDE2D cannot handle. Since a rectangle in $P1, P2, P3$ has six smooth faces, any smooth transformation of this rectangle into a region in Cartesian coordinates will also have six or fewer (as few as one, in the case of a sphere) smooth boundary pieces. Thus any 3D region with more than six smooth boundary pieces (or 2D region with more than four) cannot be handled by the PDE2D algorithm. However, the last example in Section 4 shows that sometimes such regions can still be handled approximately.

Implementing the coordinate transformation did not involve any internal modifications to the PDE2D library routines, only to the function subprograms where the PDE coefficients and boundary condition coefficients are defined by the user. These functions are called by PDE2D with various values of $P1, P2, P3, U, U_1, U_2, U_3, U_{11}, \ldots$; all that is required is to insert code to compute $X, Y, Z, U_x, U_y, U_z, U_{xx}, \ldots$ for given $P1, P2, P3, U_1, U_2, U_3, U_{11}, \ldots$, using the chain rule. Then the user can simply define the PDE and boundary condition coefficients in terms of $X, Y, Z, U, U_x, U_y, U_z, U_{xx}, \ldots$, though the non-Cartesian variables and derivatives can still be used as well, if desired.

4. Examples

In the first example, an elasticity problem was solved in a region that is half a torus, of major radius $R_0 = 5$ and minor radius $R_1 = 4$, with a smaller torus, of minor radius $0.5R_1$, removed (see Fig. 2).

The partial differential equations for the elastic body are

$$\begin{aligned} &AU_{xx} + BV_{yx} + BW_{zx} + C(U_{yy} + V_{xy}) + C(U_{zz} + W_{xz}) = 0, \\ &C(U_{yx} + V_{xx}) + AV_{yy} + BU_{xy} + BW_{zy} + C(V_{zz} + W_{yz}) = 0, \\ &C(U_{zx} + W_{xx}) + C(V_{zy} + W_{yy}) + AW_{zz} + BU_{xz} + BV_{yz} = 0, \end{aligned}$$

where (U, V, W) is the displacement vector, $A = 2.963$, $B = 1.460$, and $C = 0.752$ are constants (involving the elastic modulus and Poisson ratio).

There are periodic boundary conditions on $P2$, and at $P3 = 0.5R_1$ the displacements are set to zero. On the outer surface of the torus, $P3 = R_1$, there is a unit inward boundary force; that is, the boundary force vector is $-(Nx, Ny, Nz)$, where (Nx, Ny, Nz) is the unit outward normal to the boundary, in Cartesian coordinates. This means the boundary conditions are:

$$\begin{aligned} &(AU_x + BV_y + BW_z)Nx + C(U_y + V_x)Ny + C(U_z + W_x)Nz = -Nx \\ &C(U_y + V_x)Nx + (AV_y + BU_x + BW_z)Ny + C(V_z + W_y)Nz = -Ny \\ &C(U_z + W_x)Nx + C(V_z + W_y)Ny + (AW_z + BU_x + BV_y)Nz = -Nz \end{aligned}$$

On one flat end $(P1 = 0)$, there are zero displacements, and on the other $(P1 = \pi)$, there are zero boundary forces.

Once the coordinate transformation is defined (1), the user does not need to convert the PDEs or boundary conditions into the new coordinate system—saving tremendous human effort. The PDEs and boundary conditions are input exactly as shown above (see Fig. 3), in a GUI session. Volume integrals and boundary integrals can also be written using Cartesian coordinates. The unit outward normal vector in Cartesian coordinates is available to the boundary condition functions and boundary integrals. In short, while PDE2D internally solves the problem in the new $P1, P2, P3$ coordinate system, the user can supply *everything* in Cartesian coordinate form.

Cross-sectional contour plots of scalar variables, and cross-sectional vector field plots, can be made that reflect the true geometry of the cross-section. For example, Fig. 4 shows the displacement field (U, V, W) at a cross-section midway between the two flat ends, $P1 = \frac{\pi}{2}$, which is plotted using axes $P3 * \cos(P2)$ vs. $P3 * \sin(P2)$, rather than $P2$ vs. $P3$, so that the cross-section looks like an annulus, as it should, rather than a rectangle. If the *MATLAB*® m-file generated automatically by PDE2D is run, it produces 3D cross-sectional plots, with values coded by color, which give an even better picture of the 3D solution. Fig. 5 shows plots of the vertical displacement W at two $P3$ = constant cross-sections.

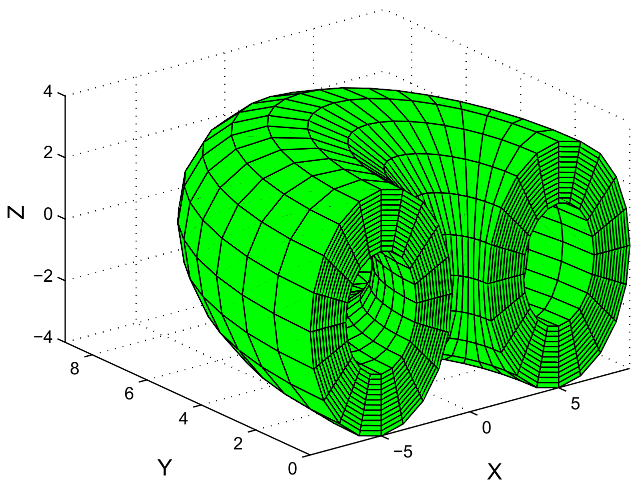
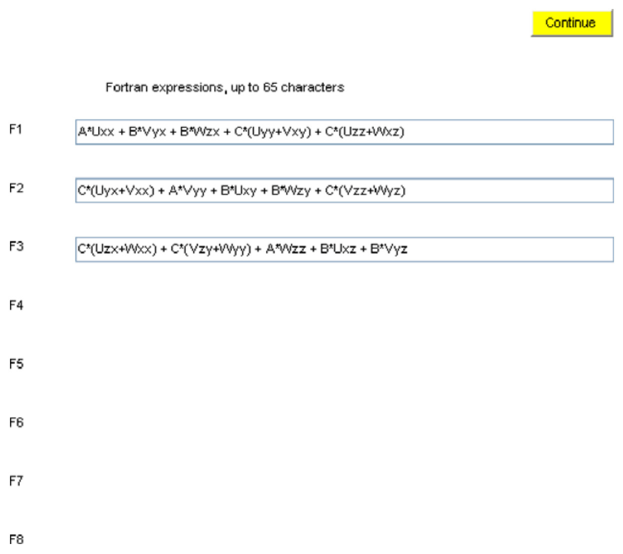


Fig. 2. Part of a torus.



Fortran expressions, up to 65 characters

F1	A*Uxx + B*Vyx + B*Wzx + C*(Uyy+Vxy) + C*(Uzz+Wxz)
F2	C*(Uyx+Vxx) + A*Vyy + B*Uxy + B*Wzy + C*(Vzz+Wyz)
F3	C*(Uzx+Wxx) + C*(Vzy+Wyy) + A*Wzz + B*Uxz + B*Vyz

Fig. 3. PDEs defined in GUI session.

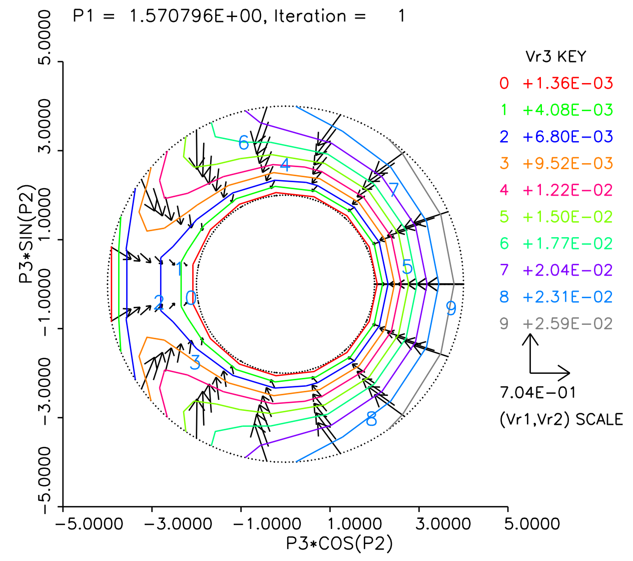


Fig. 4. Displacement field at cross-section midway between ends.

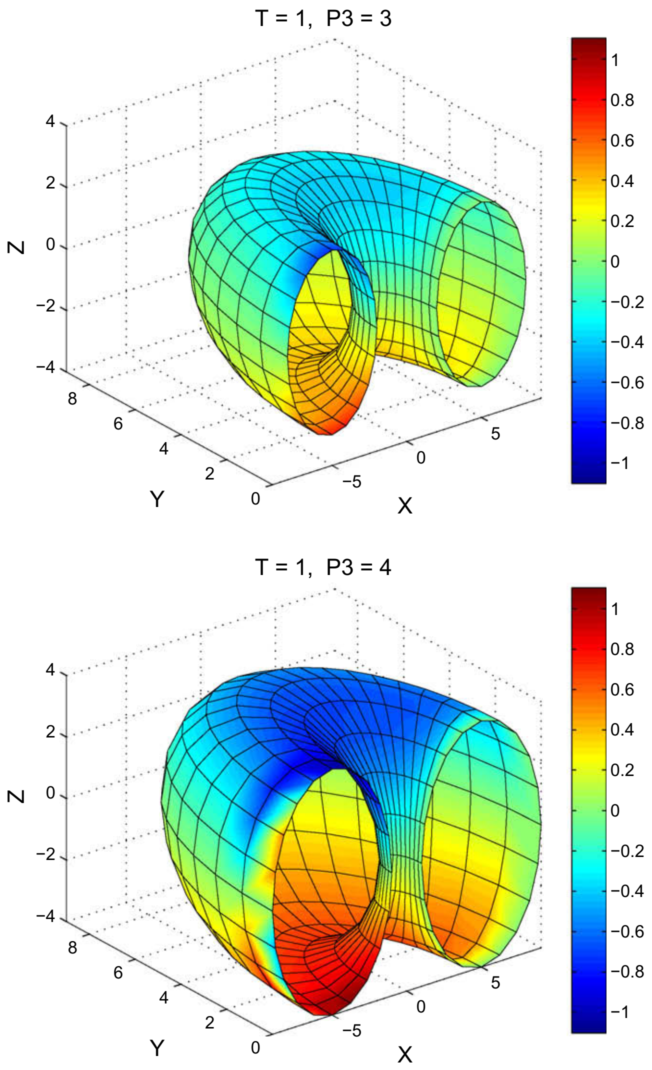


Fig. 5. Vertical displacement (W) at two cross-sections.

When $NP1GRID = NP2GRID = 12$, $NP3GRID = 8$, the computed integral of $Ux + Vy + Wz$ (total volume change) is -229.821. The computed value of the boundary integral of $U\ Nx + V\ Ny + W\ Nz$ is -229.785; by the divergence theorem this should equal the volume change, and in fact they differ by only 0.02%.

As a second example, the first eigenvalue of the Laplacian was computed in a cylinder with a hemispherical cap, with $U = 0$ on the surface. Here the transformation equations are

$$X = R(P3)P2\cos(P1),$$
$$Y = R(P3)P2\sin(P1),$$
$$Z = P3,$$

where

$$R(P3) = 1 \quad \text{when } P3 \leqslant 0$$
$$R(P3) = \sqrt{1 - P3^2} \quad \text{when } P3 > 0.$$

There are periodic boundary conditions at $P1 = 0, 2\pi$ ($\theta = 0, 2\pi$), no boundary conditions at $P3 = 1 (Z = 1$, a single point at the top) and $P2 = 0$ (centerline of the cylinder), and $U = 0$ is imposed at $P2 = 1$ (lateral boundary) and at $P3 = -1 (Z = -1$, the bottom boundary).

Fig. 6 shows the first eigenfunction at the cross-section $P1 = 0$. With $NP1GRID = 5$, $NP2GRID = NP3GRID = 25$, a first eigenvalue of -8.9294195 was computed, which is correct to six significant figures (exact value = -8.9294183). Note that high accuracy is achieved despite the discontinuity in some of the second derivatives, for example, $\frac{\partial^2 X}{\partial P3^2}$. Continuity of the *first* derivatives is essential, however; for example, if the hemispherical cap is replaced by a cone, the solution will not converge.

In the third example, we solve the problem $U_{xx} + U_{yy} + U_{zz} = 3U$ in three-quarters of a rectangle, with $U = e^{x+y+z}$ on the boundary, as shown (approximately) in Fig. 7. We could parameterize this region as follows:

$$x = P1$$
$$y = \frac{P2\cos(P3)}{max(|\cos(P3)|, |\sin(P3)|)}$$
$$z = \frac{P2\sin(P3)}{max(|\cos(P3)|, |\sin(P3)|)}$$

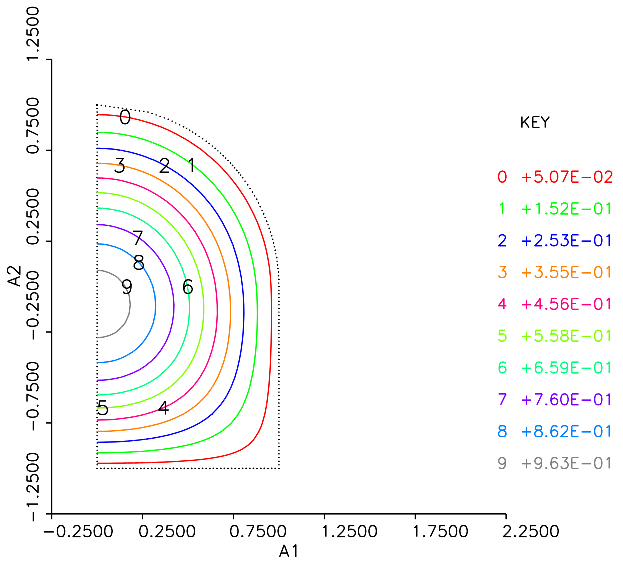


Fig. 6. First eigenfunction at $P1 = 0$ cross-section.

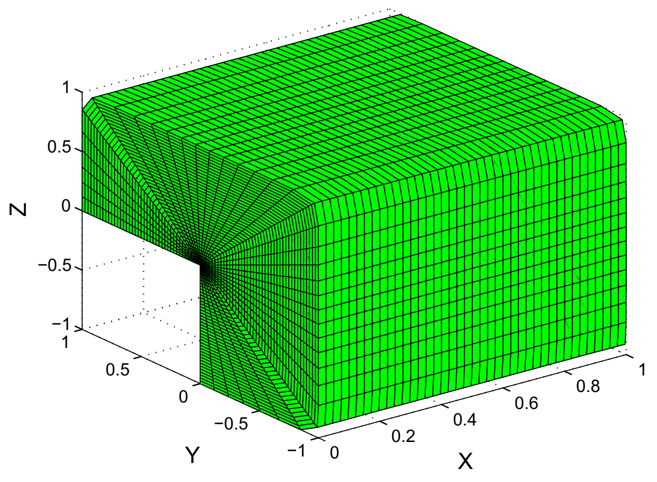


Fig. 7. Approximation of three-quarters of rectangle.

where $0 < P1 < 1,\ 0 < P2 < 1,\ 0 < P3 < \frac{3\pi}{2}$. Unfortunately, this parameterization is not smooth, because of the use of the "max" function: some first derivatives are discontinuous at $P3$ (polar coordinate θ) equal to $\frac{\pi}{4},\ \frac{3\pi}{4}$ and $\frac{5\pi}{4}$. Although this is a simple region, the boundary consists of 8 smooth parts, so it is not possible to parameterize it *smoothly* with constant limits on the parameters. However, when α is large, we can approximate the maximum function as follows (note that the left hand side is the L_∞ norm of the vector $(\cos(P3), \sin(P3))$, while the right hand side is the L_α norm):

$$max(|\cos(P3)|, |\sin(P3)|) \approx (|\cos(P3)|^\alpha + |\sin(P3)|^\alpha)^{\frac{1}{\alpha}}$$

When this is done, the resulting parameterization is now smooth, and for $\alpha = 20$ the resulting region approximates the desired region with reasonable accuracy, as seen in Fig. 7. With $NP1GRID = 6,\ NP2GRID = 10,\ NP3GRID = 25$, the volume of the parameterized region was 2.990 (should be 3), and the average relative error in the computed solution (true solution is $U = e^{x+y+z}$) was 0.6%. For regions like this one, which can only be approximately handled, the accuracy obtained is naturally not as good as would be expected with a standard Galerkin finite element method.

5. Conclusions

PDE2D still cannot solve problems in complicated 3D regions, with many boundary parts. For *simple* 3D regions, however, the coordinate transformation described here offers significant advantages over that used by other FEM software designed to handle more general 3D regions, particularly with regard to ease of use. Once the user has supplied the transformation equations, the rest of the problem description is as simple as for a rectangular region. Furthermore, this approach normally produces $O(h^4)$ accuracy even in regions with curved boundaries; without a global transformation, comparable accuracy can only be obtained if third-order *isoparametric* elements are used, and the use of high-order isoparametric elements involves much more development effort than the global transformation approach. Note that it is not claimed that the collocation method described in this paper produces accuracy *superior* to the traditional Galerkin finite element method, only that comparable accuracy can be obtained with far less effort by both the software developer and the end user, for a wide range of interesting, simple 3D regions.

The use of $P1,\ P2,\ P3 = constant$ curves as gridlines is a further advantage. This is often natural and desirable, for example, if the region is a sphere and the solution is known to have a singularity at the origin, we may use spherical coordinates and put more gridlines in the radial direction, and distribute them nonuniformly in this direction.

References

[1] Amestoy P, Duff I, L'Excellent J. Multifrontal parallel distributed symmetric and unsymmetric solvers. Comput Methods Appl Mech Eng 2000;184:501–20.
[2] Duff I. The impact of high performance computing in the solution of linear systems: trends and problems. J Comp Appl Math 2000;123:515–30.
[3] Duff I, Reid J. The multifrontal solution of indefinite sparse symmetric linear equations. ACM Trans Math Softw 1983;9:302–25.
[4] Gordon D, Gordon R. CGMN revisited: robust and efficient solution of stiff linear systems derived from elliptic partial differential equations. ACM Trans Math Softw 2008;35:18:1–18:27.
[5] Gravvanis G. OpenMP based parallel normalized direct methods for sparse finite element linear systems. J Supercomput 2009;47(1):44–52.
[6] Gravvanis G, Lipitakis E. An explicit sparse unsymmetric finite element solver. Commun Numer Methods Eng 1996;12:21–9.
[7] Rice J, Boisvert R. Elliptic problem solving with ELLPACK. Springer-Verlag; 1985.
[8] Saad Y, van der Vorst H. Iterative solution of linear systems in the 20th century. J Comp Appl Math 2000;123:1–33.
[9] Sewell G. An adaptive computer program for the solution of $\nabla \cdot (p(x,y)\nabla u) = f(x,y,u)$ on a polygonal region. In: The mathematics of finite elements and applications II. Academic Press; 1976. p. 543–53.
[10] Sewell G. TWODEPEP, a small general purpose finite element program. Angewandte Informatik 1983;4:249–53.
[11] Sewell G. Analysis of a finite element method: PDE/PROTRAN. Springer-Verlag; 1985.
[12] Sewell G. PDE2D: easy-to-use software for general two-dimensional partial differential equations. Adv Eng Softw 1993;17:105–12.
[13] Sewell G. The numerical solution of ordinary and partial differential equations. 2nd ed. John Wiley & Sons; 2005.
[14] Strang G, Fix G. An analysis of the finite element method. Prentice Hall; 1973.
[15] Topper J. Financial engineering with finite elements. John Wiley & Sons; 2005.