7. Boundary Value Problems

We have defined the Chebyshev differentiation matrix D_N and put together a MATLAB program, `cheb`, to compute it. In this chapter we illustrate how such matrices can be used to solve some boundary value problems arising in ordinary and partial differential equations.

As our first example, consider the linear ODE boundary value problem

$$u_{xx} = e^{4x}, \qquad -1 < x < 1, \quad u(\pm 1) = 0. \tag{7.1}$$

This is a Poisson equation, with solution $u(x) = [e^{4x} - x\sinh(4) - \cosh(4)]/16$. We use the PDE notation u_{xx} instead of u'' because we shall soon increase the number of dimensions.

To solve the problem numerically, we can compute the second derivative via D_N^2, the square of D_N. The first thing to note is that D_N^2 can be evaluated either by squaring D_N, which costs $O(N^3)$ floating point operations, or by explicit formulas [GoLu83a, Pey86] or recurrences [WeRe00, Wel97], which cost $O(N^2)$ floating point operations. There are real advantages to the latter approaches, but in this book, for simplicity, we just square D_N.

The other half of the problem is the imposition of the boundary conditions $u(\pm 1) = 0$. For simple problems like (7.1) with homogeneous Dirichlet boundary conditions, we can proceed as follows. We take the interior Chebyshev points $x_1, \ldots, x_{N-1}$ as our computational grid, with $v = (v_1, \ldots, v_{N-1})^T$ as the corresponding vector of unknowns. Spectral differentiation is then carried out like this:

- *Let $p(x)$ be the unique polynomial of degree $\le N$ with $p(\pm 1) = 0$ and $p(x_j) = v_j$, $1 \le j \le N-1$.*

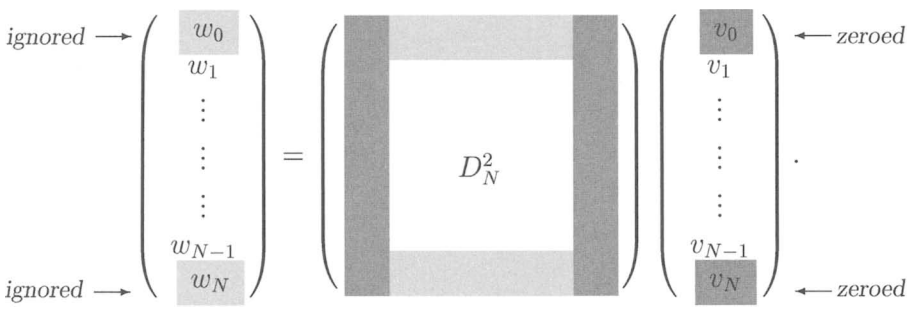
- *Set* $w_j = p''(x_j)$, $1 \le j \le N-1$.

This is not the only means of imposing boundary conditions in spectral methods. We shall consider alternatives in Chapter 13, where among other examples, Programs 32 and 33 (pp. 136 and 138) solve (7.1) again with inhomogeneous Dirichlet and homogeneous Neumann boundary conditions, respectively.

Now D_N^2 is an $(N+1)\times(N+1)$ matrix that maps a vector $(v_0,\dots,v_N)^T$ to a vector $(w_0,\dots,w_N)^T$. The procedure just described amounts to a decision that we wish to:

- *Fix* v_0 *and* v_N *at zero.*
- *Ignore* w_0 *and* w_N.

This implies that the first and last columns of D_N^2 have no effect (since multiplied by zero) and the first and last rows have no effect either (since ignored):



In other words, to solve our one-dimensional Poisson problem by a Chebyshev spectral method, we can make use of the $(N-1)\times(N-1)$ matrix $\widetilde{D}_N^2$ obtained by stripping D_N^2 of its first and last rows and columns. In MATLAB notation:

$$\widetilde{D}_N^2 = D_N^2(1\colon N-1,\ 1\colon N-1).$$

In an actual MATLAB program, since indices start at 1 instead of 0, this will become `D2 = D2(2:N,2:N)` or `D2 = D2(2:end-1,2:end-1)`.

With $\widetilde{D}_N^2$ in hand, the numerical solution of (7.1) becomes a matter of solving a linear system of equations:

$$\widetilde{D}_N^2 v = f.$$

Program 13 carries out this process. We should draw attention to a feature of this program that appears here for the first time in the book and will reappear in a number of our later programs. Although the algorithm calculates the vector $(v_1,\dots,v_{N-1})^T$ of approximations to u at the grid points, as always with

spectral methods, we really have more information about the numerical solution than just point values. Implicitly we are dealing with a polynomial interpolant $p(x)$, and in MATLAB this can be calculated conveniently (though not very quickly or stably) by a command of the form `polyval(polyfit(...))`. Program 13 uses this trick to evaluate $p(x)$ on a fine grid, both for plotting and for measuring the error, which proves to be on the order of 10^{-10}. Exercise 7.1 investigates the more stable method for constructing $p(x)$ known as barycentric interpolation. For practical plotting purposes with spectral methods, much simpler local interpolants are usually adequate; see, e.g., the use of `interp2( ...,'cubic')` in Program 16 (p. 70).

What if the equation is nonlinear? For example, suppose we change (7.1) to

$$u_{xx} = e^u, \qquad -1 < x < 1, \quad u(\pm 1) = 0. \tag{7.2}$$

Because of the nonlinearity, it is no longer enough simply to invert the second-order differentiation matrix $\widetilde{D}_N^2$. Instead, we can solve the problem iteratively. We choose an initial guess, such as the vector of zeros, and then iterate by repeatedly solving the system of equations

$$\widetilde{D}_N^2 v_{\text{new}} = \exp(v_{\text{old}}),$$

where $\exp(v)$ is the column vector defined componentwise by $(\exp(v))_j = e^{v_j}$. Program 14 implements this iteration with a crude stopping criterion, and convergence occurs in 29 steps.

To convince ourselves that we have obtained the correct solution, we can modify Program 14 to print results for various N. Here is such a table:

N	no. its.	u(0)
2	34	-0.35173371124920
4	29	-0.36844814823915
6	29	-0.36805450387666
8	29	-0.36805614384219
10	29	-0.36805602345302
12	29	-0.36805602451189
14	29	-0.36805602444069
16	29	-0.36805602444149
18	30	-0.36805602444143
20	29	-0.36805602444143

Evidently $u(0)$ is accurate to 12 or 13 digits, even with $N = 16$. The convergence of this iteration is analyzed in Exercise 7.3.

As a third application of the modified second-order differentiation matrix $\widetilde{D}_N^2$, consider the eigenvalue boundary value problem

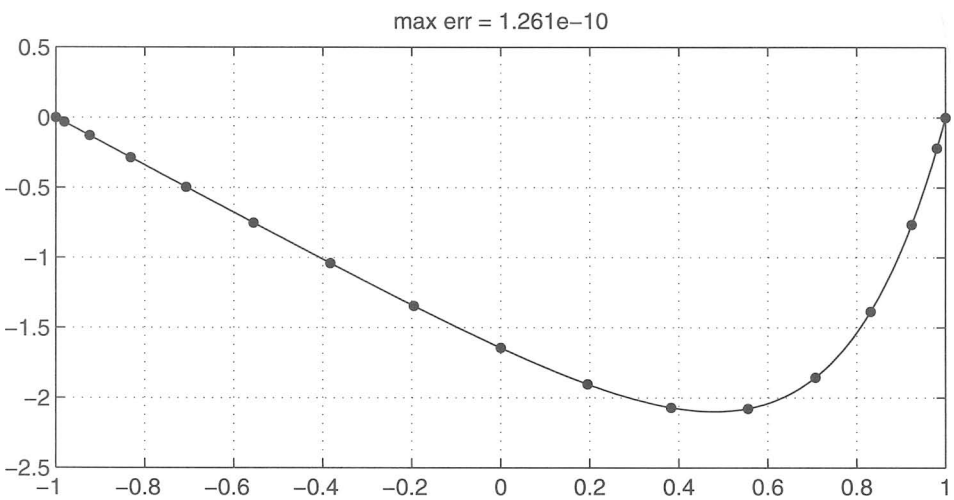
$$u_{xx} = \lambda u, \qquad -1 < x < 1, \quad u(\pm 1) = 0. \tag{7.3}$$

Program 13

```
% p13.m - solve linear BVP u_xx = exp(4x), u(-1)=u(1)=0

  N = 16;
  [D,x] = cheb(N);
  D2 = D^2;
  D2 = D2(2:N,2:N);                    % boundary conditions
  f = exp(4*x(2:N));
  u = D2\f;                            % Poisson eq. solved here
  u = [0;u;0];
  clf, subplot('position',[.1 .4 .8 .5])
  plot(x,u,'.','markersize',16)
  xx = -1:.01:1;
  uu = polyval(polyfit(x,u,N),xx);     % interpolate grid data
  line(xx,uu)
  grid on
  exact = ( exp(4*xx) - sinh(4)*xx - cosh(4) )/16;
  title(['max err = ' num2str(norm(uu-exact,inf))],'fontsize',12)
```

Output 13



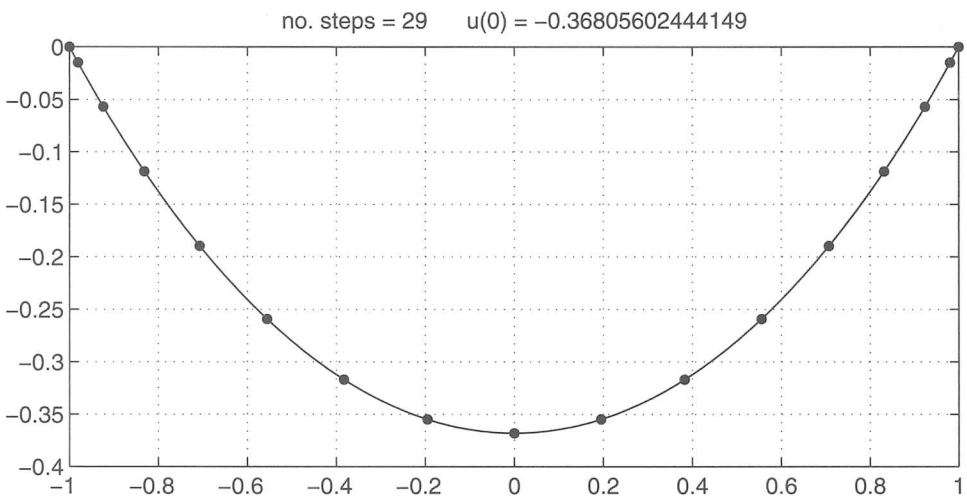
Output 13: *Solution of the linear boundary value problem* (7.1).

Program 14

```
% p14.m - solve nonlinear BVP u_xx = exp(u), u(-1)=u(1)=0
%          (compare p13.m)

  N = 16;
  [D,x] = cheb(N); D2 = D^2; D2 = D2(2:N,2:N);
  u = zeros(N-1,1);
  change = 1; it = 0;
  while change > 1e-15                    % fixed-point iteration
    unew = D2\exp(u);
    change = norm(unew-u,inf);
    u = unew; it = it+1;
  end
  u = [0;u;0];
  clf, subplot('position',[.1 .4 .8 .5])
  plot(x,u,'.','markersize',16)
  xx = -1:.01:1;
  uu = polyval(polyfit(x,u,N),xx);
  line(xx,uu), grid on
  title(sprintf('no. steps = %d      u(0) =%18.14f',it,u(N/2+1)))
```

Output 14

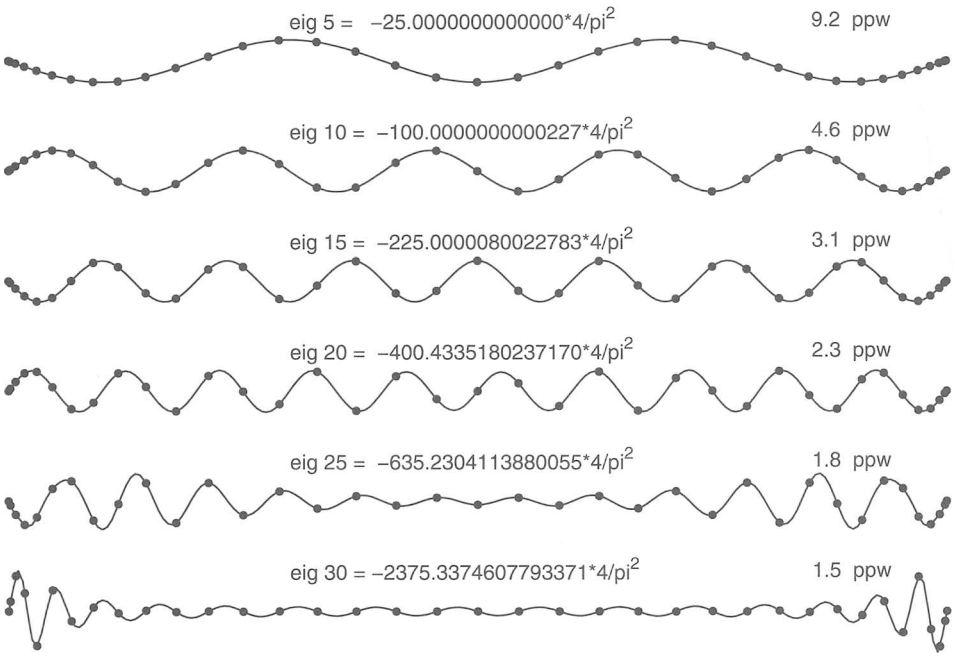


Output 14: *Solution of the nonlinear boundary value problem* (7.2).

Program 15

```
% p15.m - solve eigenvalue BVP u_xx = lambda*u, u(-1)=u(1)=0
  N = 36; [D,x] = cheb(N); D2 = D^2; D2 = D2(2:N,2:N);
  [V,Lam] = eig(D2); lam = diag(Lam);
  [foo,ii] = sort(-lam);          % sort eigenvalues and -vectors
  lam = lam(ii); V = V(:,ii); clf
  for j = 5:5:30                  % plot 6 eigenvectors
    u = [0;V(:,j);0]; subplot(7,1,j/5)
    plot(x,u,'.','markersize',12), grid on
    xx = -1:.01:1; uu = polyval(polyfit(x,u,N),xx);
    line(xx,uu), axis off
    text(-.4,.5,sprintf('eig %d =%20.13f*4/pi^2',j,lam(j)*4/pi^2))
    text(.7,.5,sprintf('%4.1f  ppw', 4*N/(pi*j)))
  end
```

Output 15



Output 15: *Eigenvalues and eigenmodes of* $\widetilde{D}_N^2$ *and the number of grid points per wavelength (*ppw*) at the center of the grid.*

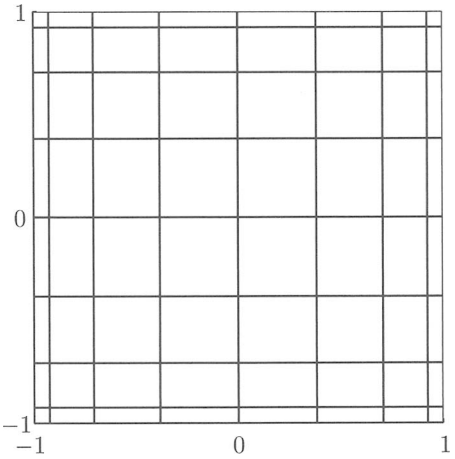


Fig. 7.1. *A tensor product grid.*

The eigenvalues of this problem are $\lambda = -\pi^2 n^2/4$, $n = 1, 2, \ldots$, with corresponding eigenfunctions $\sin(n\pi(x+1)/2)$. Program 15 calculates the eigenvalues and eigenvectors of $\widetilde{D}_N^2$ for $N = 36$ by MATLAB's built-in matrix eigenvalue routine `eig`. The numbers and plots of Output 15 reveal a great deal about the accuracy of spectral methods. Eigenvalues 5, 10, and 15 are obtained to many digits of accuracy, and eigenvalue 20 is still pretty good. Eigenvalue 25 is accurate to only one digit, however, and eigenvalue 30 is wrong by a factor of 3. The crucial quantity that explains this behavior is the number of points per wavelength ("ppw") in the central, coarsest part of the grid near $x = 0$. With at least two points per wavelength, the grid is fine enough everywhere to resolve the wave. With less than two points per wavelength, the wave cannot be resolved, and eigenvectors are obtained that are meaningless as approximations to the original problem.

We now consider how to extend these methods to boundary value problems in several space dimensions. To be specific, here is a two-dimensional Poisson problem:

$$u_{xx} + u_{yy} = 10\sin(8x(y-1)), \quad -1 < x, y < 1, \quad u = 0 \text{ on the boundary.} \tag{7.4}$$

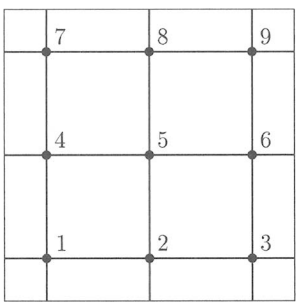
(The right-hand side has been chosen to make an interesting picture.) For such a problem we naturally set up a grid based on Chebyshev points independently in each direction, called a *tensor product grid* (Figure 7.1). Note that whereas in one dimension, a Chebyshev grid is $2/\pi$ times as dense in the middle as an equally spaced grid, in d dimensions this figure becomes $(2/\pi)^d$. Thus the great majority of grid points lie near the boundary. Sometimes this is wasteful, and techniques have been devised to reduce the waste [For96,

KaSh99, KoTa93]. At other times, when boundary layers or other fine details appear near boundaries, the extra resolution there may be useful.

The easiest way to solve a problem on a tensor product spectral grid is to use tensor products in linear algebra, also known as *Kronecker products.* The Kronecker product of two matrices A and B is denoted by $A \otimes B$ and is computed in MATLAB by the command `kron(A,B)`. If A and B are of dimensions $p \times q$ and $r \times s$, respectively, then $A \otimes B$ is the matrix of dimension $pr \times qs$ with $p \times q$ block form, where the i,j block is $a_{ij}B$. For example,

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \otimes \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \left(\begin{array}{cc|cc} a & b & 2a & 2b \\ c & d & 2c & 2d \\ \hline 3a & 3b & 4a & 4b \\ 3c & 3d & 4c & 4d \end{array}\right).$$

To explain how Kronecker products can be used for spectral methods, let us consider the case $N = 4$. Suppose we number the internal nodes in the obvious, "lexicographic" ordering:



Also suppose that we have data $(v_1, v_2, \ldots, v_9)^T$ at these grid points. We wish to approximate the Laplacian by differentiating spectrally in the x and y directions independently. Now the 3×3 differentiation matrix with $N = 4$ in one dimension is given by `D = cheb(4); D2 = D^2; D2 = D2(2:4,2:4)`:

$$\widetilde{D}_4^2 = \begin{pmatrix} -14 & 6 & -2 \\ 4 & -6 & 4 \\ -2 & 6 & -14 \end{pmatrix}.$$

If I denotes the 3×3 identity, then the second derivative with respect to x

will accordingly be computed by the matrix `kron(I,D2)`:

$$I \otimes \widetilde{D}_N^2 = \left(\begin{array}{ccc|ccc|ccc} -14 & 6 & -2 & & & & & & \\ 4 & -6 & 4 & & & & & & \\ -2 & 6 & -14 & & & & & & \\ \hline & & & -14 & 6 & -2 & & & \\ & & & 4 & -6 & 4 & & & \\ & & & -2 & 6 & -14 & & & \\ \hline & & & & & & -14 & 6 & -2 \\ & & & & & & 4 & -6 & 4 \\ & & & & & & -2 & 6 & -14 \end{array}\right).$$

The second derivative with respect to y will be computed by `kron(D2,I)`:

$$\widetilde{D}_N^2 \otimes I = \left(\begin{array}{ccc|ccc|ccc} -14 & & & 6 & & & -2 & & \\ & -14 & & & 6 & & & -2 & \\ & & -14 & & & 6 & & & -2 \\ \hline 4 & & & -6 & & & 4 & & \\ & 4 & & & -6 & & & 4 & \\ & & 4 & & & -6 & & & 4 \\ \hline -2 & & & 6 & & & -14 & & \\ & -2 & & & 6 & & & -14 & \\ & & -2 & & & 6 & & & -14 \end{array}\right).$$

Our discrete Laplacian is now the *Kronecker sum* [HoJo91]

$$L_N = I \otimes \widetilde{D}_N^2 + \widetilde{D}_N^2 \otimes I. \tag{7.5}$$

This matrix, though not dense, is not as sparse as one typically gets with finite differences or finite elements. Fortunately, thanks to spectral accuracy, we may hope to obtain satisfactory results with dimensions in the hundreds rather than the thousands or tens of thousands.

Program 16 solves the Poisson problem (7.4) numerically with $N = 24$. The program produces two plots, which we label Output 16a and Output 16b. The first shows the locations of the 23,805 nonzero entries in the 529×529 matrix L_{24}. The second plots the solution and prints the value $u(x, y)$ for $x = y = 2^{-1/2}$, which is convenient because this is one of the grid points whenever N is divisible by 4. The program also notes the time taken to perform the solution of the linear system of equations: on my Sparc Ultra 5 workstation in MATLAB version 6.0, 1.2 seconds.

A variation of the Poisson equation is the *Helmholtz equation*,

$$u_{xx} + u_{yy} + k^2 u = f(x, y), \quad -1 < x, y < 1, \quad u = 0 \text{ on the boundary}, \tag{7.6}$$

Program 16

```
% p16.m - Poisson eq. on [-1,1]x[-1,1] with u=0 on boundary

% Set up grids and tensor product Laplacian and solve for u:
  N = 24; [D,x] = cheb(N); y = x;
  [xx,yy] = meshgrid(x(2:N),y(2:N));
  xx = xx(:); yy = yy(:);       % stretch 2D grids to 1D vectors
  f = 10*sin(8*xx.*(yy-1));
  D2 = D^2; D2 = D2(2:N,2:N); I = eye(N-1);
  L = kron(I,D2) + kron(D2,I);                        % Laplacian
  figure(1), clf, spy(L), drawnow
  tic, u = L\f; toc          % solve problem and watch the clock

% Reshape long 1D results onto 2D grid:
  uu = zeros(N+1,N+1); uu(2:N,2:N) = reshape(u,N-1,N-1);
  [xx,yy] = meshgrid(x,y);
  value = uu(N/4+1,N/4+1);

% Interpolate to finer grid and plot:
  [xxx,yyy] = meshgrid(-1:.04:1,-1:.04:1);
  uuu = interp2(xx,yy,uu,xxx,yyy,'cubic');
  figure(2), clf, mesh(xxx,yyy,uuu), colormap([0 0 0])
  xlabel x, ylabel y, zlabel u
  text(.4,-.3,-.3,sprintf('u(2^{-1/2},2^{-1/2}) = %14.11f',value))
```

where k is a real parameter. This equation arises in the analysis of wave propagation governed by the equation

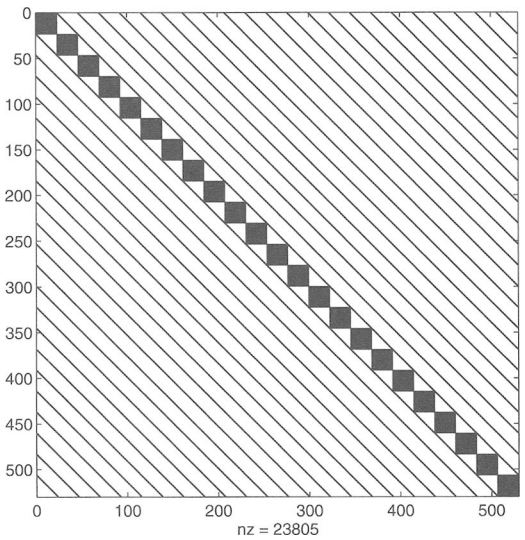
$$-U_{tt} + U_{xx} + U_{yy} = e^{ikt} f(x,y), \quad -1 < x, y < 1, \quad U = 0 \text{ on the boundary} \tag{7.7}$$

after separation of variables to get $U(x,y,t) = e^{ikt} u(x,y)$. Program 17 is a minor modification of Program 16 to solve such a problem for the particular choices

$$k = 9, \qquad f(x,y) = \exp(-10\,[(y-1)^2 + (x-\tfrac{1}{2})^2]\,). \tag{7.8}$$

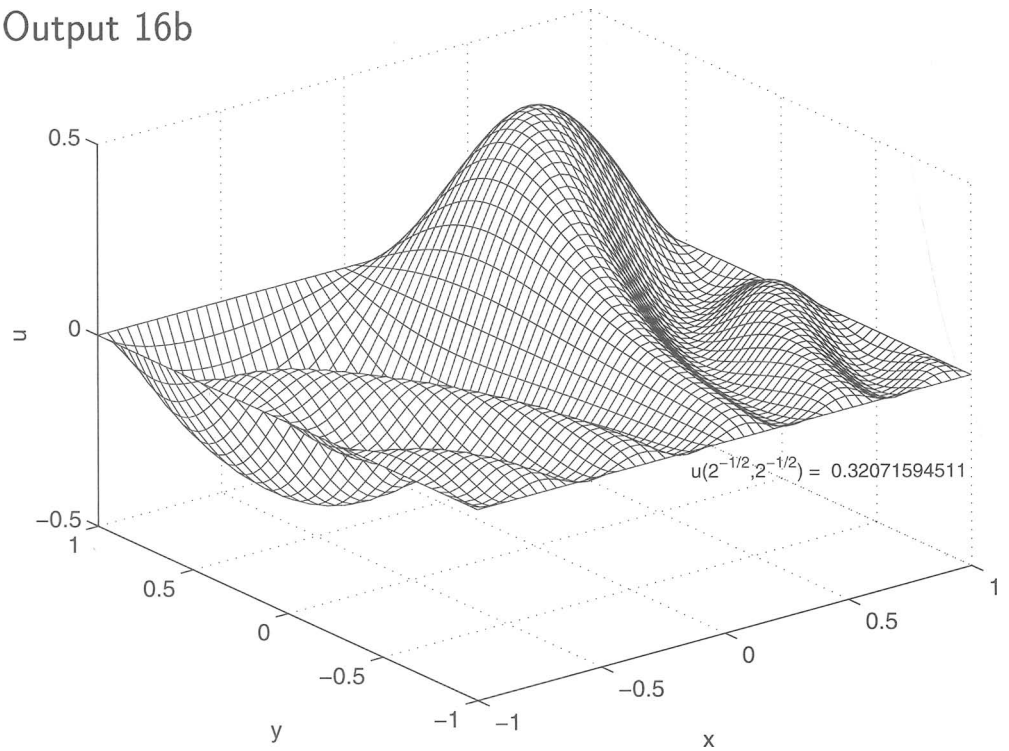
The solution appears as a mesh plot in Output 17a and as a contour plot in Output 17b. It is clear that the response generated by this forcing function $f(x,y)$ for this value $k = 9$ has approximately the form of a wave with three half-wavelengths in the x direction and five half-wavelengths in the y direction. This is easily explained. Such a wave is an eigenfunction of the homogeneous

Output 16a



Output 16a: *Sparsity plot of the* 529×529 *discrete Laplacian* (7.5).

Output 16b



Output 16b: *Solution of the Poisson equation* (7.4). *The result has been interpolated to a finer rectangular grid for plotting. The computed value* $u(2^{-1/2}, 2^{-1/2})$ *is accurate to nine digits.*

Program 17

```
% p17.m - Helmholtz eq. u_xx + u_yy + (k^2)u = f
%         on [-1,1]x[-1,1]    (compare p16.m)

% Set up spectral grid and tensor product Helmholtz operator:
  N = 24; [D,x] = cheb(N); y = x;
  [xx,yy] = meshgrid(x(2:N),y(2:N));
  xx = xx(:); yy = yy(:);
  f = exp(-10*((yy-1).^2+(xx-.5).^2));
  D2 = D^2; D2 = D2(2:N,2:N); I = eye(N-1);
  k = 9;
  L = kron(I,D2) + kron(D2,I) + k^2*eye((N-1)^2);

% Solve for u, reshape to 2D grid, and plot:
  u = L\f;
  uu = zeros(N+1,N+1); uu(2:N,2:N) = reshape(u,N-1,N-1);
  [xx,yy] = meshgrid(x,y);
  [xxx,yyy] = meshgrid(-1:.0333:1,-1:.0333:1);
  uuu = interp2(xx,yy,uu,xxx,yyy,'cubic');
  figure(1), clf, mesh(xxx,yyy,uuu), colormap([0 0 0])
  xlabel x, ylabel y, zlabel u
  text(.2,1,.022,sprintf('u(0,0) = %13.11f',uu(N/2+1,N/2+1)))
  figure(2), clf, contour(xxx,yyy,uuu)
  colormap([0 0 0]), axis square
```

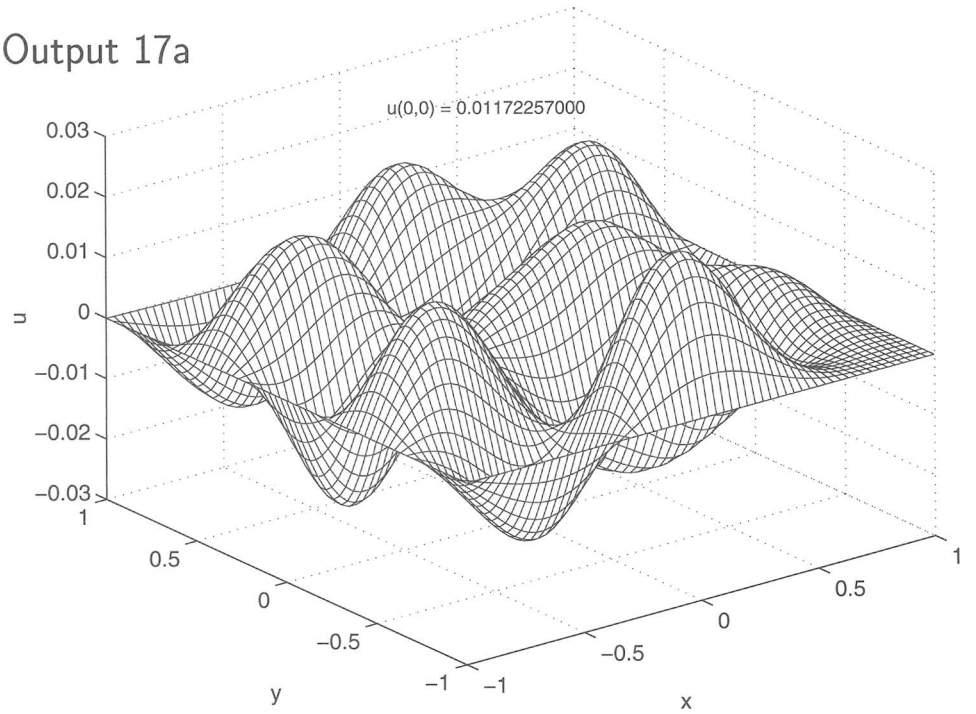
Helmholtz problem (i.e., $f(x,y) = 0$) with eigenvalue

$$k = \tfrac{1}{2}\sqrt{3^2+5^2} \approx 9.1592.$$

Our choice $k = 9$ gives near-resonance with this (3,5) mode.

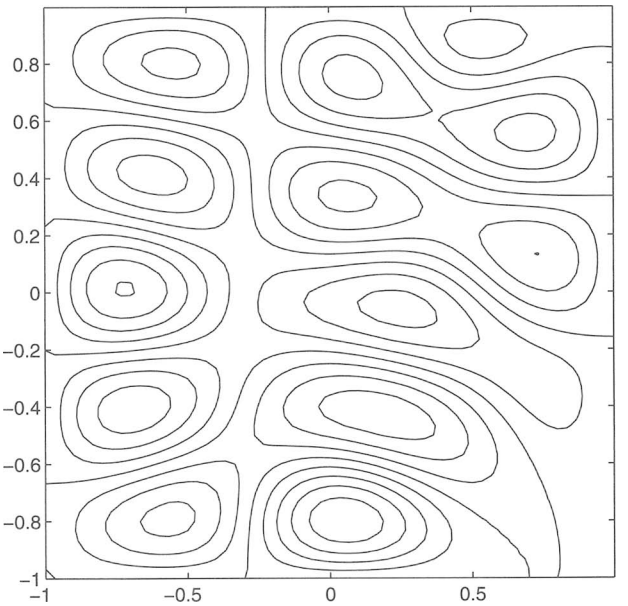
Summary of This Chapter. Homogeneous Dirichlet boundary conditions for spectral collocation methods can be implemented by simply deleting the first and/or last rows and columns of a spectral differentiation matrix. Problems in two space dimensions can be formulated in terms of Kronecker products, and for moderate-sized grids, they can be solved that way on the computer. Nonlinear problems can be solved by iteration.

Output 17a



Output 17a: *Solution of the Helmholtz problem* (7.6), (7.8). *The computed value* $u(0,0)$ *is accurate to nine digits.*

Output 17b



Output 17b: *Same result represented as contour plot.*

Exercises

7.1. Modify Program 13 so that instead of `polyval` and `polyfit`, it uses the more stable formula of *barycentric interpolation* [Hen82]:

$$p(x) = \sum_{j=0}^{N} \frac{a_j^{-1} u_j}{x - x_j} \Bigg/ \sum_{j=0}^{N} \frac{a_j^{-1}}{x - x_j}, \tag{7.9}$$

where $\{a_j\}$ are defined by (6.7). Experiment with various interpolation problems (such as that of Exercise 5.1) and find evidence of the enhanced stability of this method.

7.2. Solve the boundary value problem $u_{xx} + 4u_x + e^x u = \sin(8x)$ numerically on $[-1, 1]$ with boundary conditions $u(\pm 1) = 0$. To 10 digits of accuracy, what is $u(0)$?

7.3. In the iteration of Program 14, each step is observed to reduce the error norm by a factor of about 0.2943. This explains why 30 steps are enough to reduce the error to 10^{-14}. Add one or two lines to the code to compute the eigenvalues of an appropriate matrix to show where the number 0.2943 comes from.

7.4. Devise an alternative to Program 14 based on Newton iteration rather than fixed-point iteration, and make it work. Do you observe quadratic convergence?

7.5. A curious feature of Program 15 is that, although the problem is self-adjoint, the matrix that approximates it is not symmetric. This is typical of spectral collocation (but not Galerkin) methods. Many things can be said about how much it does or does not matter [CaGo96, McRo00], but let us consider just one: the cost in linear algebra. Perform experiments in MATLAB to estimate how much slower nonsymmetric real eigenvalue/eigenvector calculations are than symmetric ones for dense $N \times N$ matrices for values of N such as 100, 200, 300. Look up the algorithms in a book such as [Dem97] or [GoVa96] to see how your experiments match theoretical predictions.

7.6. Show how, by adding just two characters to Program 16, one can make the program solve the linear system of equations by sparse rather than dense methods of numerical linear algebra. This particular sparsity structure is not readily exploited, however. Provide evidence on this matter by comparing timings for the dense and sparse variants of the code with $N = 24$ and 32.

7.7. As explained in the text, the solution of Output 17 has the form it does because of near-resonance with the $(5, 3)$ eigenvalue $k \approx 9.1592$. Run the same program to produce contour plots for each of the integers $k = 1, 2, 3, \ldots, 20$. In each case, judge from the figure what mode (i, j), if any, seems to be principally excited, and produce a table showing how closely k matches the associated eigenvalue $(\pi/2)\sqrt{i^2 + j^2}$.