23
Electrostatics Potentials via Finite Differences (PDEs)

23.1
PDE Generalities

Physical quantities such as temperature and pressure vary continuously in both space and time. Such being our world, the function or *field* $U(x, y, z, t)$ used to describe these quantities must contain independent space and time variable. As time evolves, the changes in $U(x, y, z, t)$ at any one position affects the field at neighboring points. This means that the dynamical equations describing the dependence of U on four independent variables must be written in terms of partial derivatives, and therefore the equations must be *partial differential equations* (PDEs), in contrast to ordinary differential equations (ODEs).

The most general form for a PDE with two independent variables is

$$A\frac{\partial^2 U}{\partial x^2} + 2B\frac{\partial^2 U}{\partial x \partial y} + C\frac{\partial^2 U}{\partial y^2} + D\frac{\partial U}{\partial x} + E\frac{\partial U}{\partial y} = F \tag{23.1}$$

where A, B, C, and F are arbitrary functions of the variables x and y. In the table below we define the classes of PDEs by the value of the discriminant d in the second row [68], with the next two rows being examples:

Elliptic	*Parabolic*	*Hyperbolic*
$d = AC - B^2 > 0$	$d = AC - B^2 = 0$	$d = AC - B^2 < 0$
$\nabla^2 U(x) = -4\pi\rho(x)$	$\nabla^2 U(\mathbf{x}, t) = a\partial U/\partial t$	$\nabla^2 U(\mathbf{x}, t) = c^{-2}\partial^2 U/\partial t^2$
Poisson's	Heat	Wave

We usually think of a parabolic equation as containing a first-order derivative in one variable and second-order in the other; a hyperbolic equation as containing second-order derivatives of all the variables, with opposite signs

Computationyal Physics. Problem Solving with Computers (2nd edn).
Rubin H. Landau, Manuel José Páez, Cristian C. Bordeianu
Copyright © 2007 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim
ISBN: 978-3-527-40626-5

when placed on the same side of the equal sign; and an elliptic equation as containing second-order derivatives of all the variables, with all having the same sign when placed on the same side of the equal sign.

Tab. 23.1 The relation between boundary conditions and uniqueness for PDEs.

Boundary condition	Elliptic (Poisson equation)	Hyperbolic (Wave equation)	Parabolic (Heat equation)
Dirichlet open surface	Underspecified	Underspecified	*Unique and stable (1D)*
Dirichlet closed surface	*Unique and stable*	Overspecified	Overspecified
Neumann open surface	Underspecified	Underspecified	*Unique and stable (1D)*
Neumann closed surface	*Unique and stable*	Overspecified	Overspecified
Cauchy open surface	Unphysical	*Unique and stable*	Overspecified
Cauchy closed surface	Overspecified	Overspecified	Overspecified

After solving enough problems, one often develops some physical intuition as to whether one has sufficient *boundary conditions* for there to exist a unique solution for a given physical situation (this, of course, is in addition to requisite *initial conditions*). For instance, a string tied at both ends, or a heated bar placed in an infinite heat bath, are physical situations for which the boundary conditions are adequate. If the boundary condition is the value of the solution on a surrounding closed surface, we have a *Dirichlet boundary condition*. If the boundary condition is the value of the normal derivative on the surrounding surface, we have a *Neumann boundary condition*. If both the value of solution and its derivative are specified on a closed boundary, we have a *Cauchy boundary condition*. Although having an adequate boundary condition is necessary for a unique solution, having too many boundary conditions, for instance, both Neumann and Dirichlet, may be an overspecification for which no solution exists. (Although conclusions drawn for exact PDEs may differ from those drawn for the finite-difference equations, they are usually the same; in fact, Morse and Feshbach [69] use the finite difference form to derive the relations between boundary conditions and uniqueness for each type of equation shown in Tab. 23.1 [70].)

Solving PDEs numerically differs from solving ODEs in a number of ways. First, because we are able to write all ODEs in a standard form,

$$\frac{d\mathbf{y}(t)}{dt} = \mathbf{f}(\mathbf{y}, t) \tag{23.2}$$

with t the single independent variable, we are able to use a standard algorithm, rk4 in our case, to solve all such equations. Yet because PDEs have

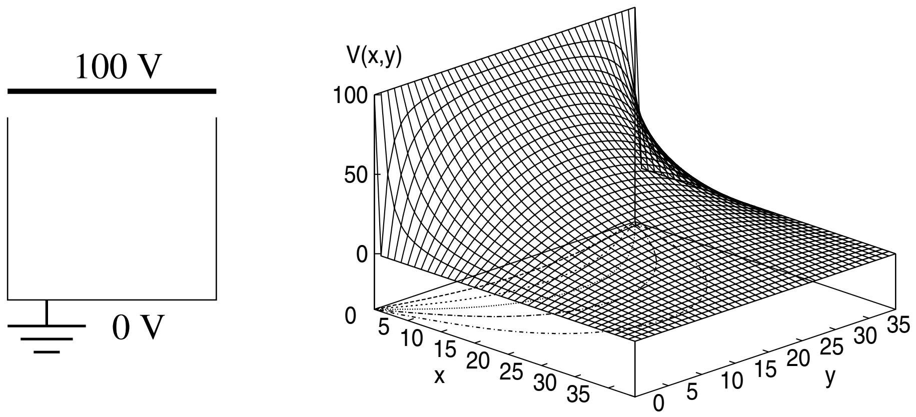

Fig. 23.1 *Left:* The region of space within a square in which we want to determine the electric potential. There is a wire at the top kept at a constant 100 V and a grounded wire at the sides and bottom. *Right*: The electric potential as a function of x and y. The projections onto the xy plane are equipotential surfaces or lines.

several independent variables, to illustrate, $\rho(x, y, z, t)$, we would have to apply (23.2) simultaneously and independently to each variable, which would be very complicated. Second, since there are more equations to solve with PDEs than ODEs, we need more information than just the two *initial conditions* $[x(0), v(0)]$. Yet because each PDE often has its own particular set of boundary conditions, we have to develop a special algorithm for each particular problem.

23.2 Electrostatic Potentials

Your **problem** is to find the electric potential for all points *inside* the charge-free square shown in Fig. 23.1. The bottom and sides of the region are made up of wires that are "grounded" (kept at 0 V). The top wire is connected to a battery that keeps it at a constant 100 V.

23.2.1 Laplace's Elliptic PDE (Theory)

We consider the entire square in Fig. 23.1 as our boundary with voltages prescribed upon it. If we imagine infinitesimal insulators placed at the top corners of the box, then we have a closed boundary within which we will solve our problem. Since the values of the potential are given on all sides, we have Neumann conditions on the boundary, and, according to Tab. 23.1, a unique and stable solution.

It is known from classical electrodynamics, that the electric potential $U(\mathbf{x})$ arising from static charges satisfies Poisson's PDE [70]:

$$\nabla^2 U(\mathbf{x}) = -4\pi\rho(\mathbf{x}) \tag{23.3}$$

where $\rho(\mathbf{x})$ is the charge density. In charge-free regions of space, that is, regions where $\rho(\mathbf{x}) = 0$, the potential satisfies *Laplace's equation*:

$$\nabla^2 U(\mathbf{x}) = 0. \tag{23.4}$$

Both these equations are elliptic PDEs of a form that occurs in various applications. We solve them in 2D rectangular coordinates:

$$\frac{\partial^2 U(x,y)}{\partial x^2} + \frac{\partial^2 U(x,y)}{\partial y^2} = \begin{cases} 0 & \text{Laplace's equation} \\ -4\pi\rho(\mathbf{x}) & \text{Poisson's equation} \end{cases} \tag{23.5}$$

In both cases, we see that the potential depends simultaneously on x and y. For Laplace's equation, the charges, which are source of the field, enter indirectly by specifying the potential values in some region of space; for Poisson's equation they enter directly.

23.3 Fourier Series Solution of PDE

For the simple geometry of Fig. 23.1, it is possible to find an analytic solution in the form of an infinite series. However, we will see that this is often not a good approach if numerical values are needed because the series may converge painfully slowly and may contain spurious oscillations. We start with Laplace's equation in Cartesian form,

$$\frac{\partial^2 U(x,y)}{\partial x^2} + \frac{\partial^2 U(x,y)}{\partial y^2} = 0 \tag{23.6}$$

with the boundary conditions given along a square of side L. If we assume that the potential is the product of independent functions of x and y, and substitute this product into (23.6), we obtain

$$U(x,y) = X(x)Y(y) \qquad \Rightarrow \qquad \frac{d^2X(x)/dx^2}{X(x)} + \frac{d^2Y(y)/dy^2}{Y(y)} = 0 \tag{23.7}$$

Because $X(x)$ is a function of only x and $Y(y)$ of only y, the derivatives in (23.7) are *ordinary* as opposed to *partial* derivatives. Since $X(x)$ and $Y(y)$ are assumed to be independent, the only way (23.7) can be valid for *all* values of

x and y is for each term in (23.7) to be equal to a constant:

$$\frac{d^2Y(y)/dy^2}{Y(y)} = -\frac{d^2X(x)/dx^2}{X(x)} = k^2 \tag{23.8}$$

$$\Rightarrow \quad \frac{d^2X(x)}{dx^2} + k^2X(x) = 0 \qquad \frac{d^2Y(y)}{dy^2} - k^2Y(y) = 0 \tag{23.9}$$

We shall see that this choice of sign for the constant matches the boundary conditions and gives us periodic behavior in x. The other choice of sign would give periodic behavior in y, and that would not work.

It is worth pointing out here that even though an analytic solution in the form of the product of separate functions of x and y exists to Laplace's equation, this does not mean that the solution to a realistic problem will have this form. Indeed, we shall see that a realistic solution requires an infinite sum of such products, which is no longer separable into functions of x and y, and is, accordingly, not a good approach if numerical values of the potential are needed.

The solutions for $X(x)$ are periodic and those for $Y(y)$ are exponential:

$$X(x) = A\sin kx + B\cos kx \qquad Y(y) = Ce^{ky} + De^{-ky} \tag{23.10}$$

The $x = 0$ boundary condition $U(x = 0, y) = 0$ can be met only if $B = 0$. The $x = L$ boundary condition $U(x = L, y) = 0$ can be met only for

$$kL = n\pi \qquad n = 1, 2, \ldots \tag{23.11}$$

Such being the case, for each value of n there is the solution

$$X_n(x) = A_n \sin\left(\frac{n\pi}{L}x\right) \tag{23.12}$$

For each value of k_n that satisfies the x boundary conditions, $Y(y)$ must satisfy the y boundary condition $U(x, 0) = 0$, which requires $D = -C$:

$$Y_n(y) = C(e^{k_n y} - e^{-k_n y}) \equiv 2C \sinh\left(\frac{n\pi}{L}y\right) \tag{23.13}$$

Because we are solving linear equations, the principle of linear superposition holds, which means that the most general solution is the sum of the products:

$$U(x, y) = \sum_{n=1}^{\infty} E_n \sin\left(\frac{n\pi}{L}x\right) \sinh\left(\frac{n\pi}{L}y\right) \tag{23.14}$$

The E_n values are arbitrary constants and are fixed by requiring the solution to satisfy the remaining boundary condition at $y = L$, $U(x, y = L) = 100$ V:

$$\sum_{n=1}^{\infty} E_n \sin\frac{n\pi}{L}x \sinh n\pi = 100\text{ V} \tag{23.15}$$

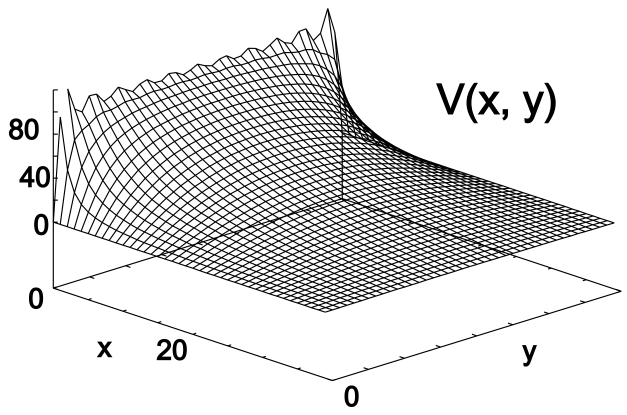

Fig. 23.2 The analytic (Fourier series) solution of Laplace's equation showing Gibb's-overshoot oscillations near $y = 0$. The solution shown here uses 21 terms, yet the oscillations remains even if a large number of terms is summed.

We determine the constants E_n by projection: multiply both sides of the equation by $\sin m\pi/Lx$, with m an integer, and integrate from 0 to L:

$$\sum_n^{\infty} E_n \sinh n\pi \int_0^L dx \sin\frac{n\pi}{L}x \sin\frac{m\pi}{L}x = \int_0^L dx\, 100 \sin\frac{m\pi}{L}x \tag{23.16}$$

The integral on the LHS is nonzero only for $n = m$, which yields

$$E_n = \begin{cases} 0 & \text{for } n \text{ even} \\ \frac{4(100)}{n\pi \sinh n\pi} & \text{for } n \text{ odd} \end{cases} \tag{23.17}$$

Finally, we obtain the potential at any point (x, y) as

$$U(x,y) = \sum_{n=1,3,5,\ldots}^{\infty} \frac{400}{n\pi} \sin\left(\frac{n\pi x}{L}\right) \frac{\sinh(n\pi y/L)}{\sinh(n\pi)} \tag{23.18}$$

23.3.1
Shortcomings of Polynomial Expansions

It is interesting to observe that the solution via the numerical algorithm (23.25) starts with the values of the potential on the boundaries and then propagates them through all space via repeated iterations. In contrast, the "exact" solution must sum an infinite number of terms to be exact, which of course is never possible in a practical calculation, and so in practice it is too a numerical approximation—but not a good one! First, the oscillatory nature of the terms being summed leads to cancellation s, and so the sum of many terms suffers from the roundoff error. Second, the sinh functions in (23.18) overflows for large n, which can be avoided somewhat by expressing the quotient of the two sinh functions in terms of exponentials, and then taking a large n limit:

$$\frac{\sinh(n\pi y/L)}{\sinh(n\pi)} = \frac{e^{n\pi(y/L-1)} - e^{-n\pi(y/L+1)}}{1-e^{-2n\pi}} \quad \rightarrow \quad e^{n\pi(y/L-1)} \tag{23.19}$$

A third problem with the "analytic" solution is that a Fourier series converges only in the *mean square* (Fig. 23.2). This means that it converges to the *average* of the left- and right-hand limits in the regions where the solution is discontinuous [71], such as in the corners of the box. Explicitly, what you see in Fig. 23.2 is a phenomenon known as the **Gibb's overshoot** that occurs when a Fourier series with a finite number of terms is used to represent a discontinuous function. Rather than fall off abruptly, the series develops large oscillations that tend to overshoot the function at the corner. To obtain a smooth solution, we had to sum 40,000 terms.

23.4 Solution: Finite Difference Method

To solve our 2D PDE, we divide space up into a lattice (Fig. 23.3) and solve for U at each site on the lattice. Since we will express derivatives in terms of the finite differences in the values of U at the lattice sites, this is called a *finite difference* method. A numerically more efficient, but also more complicated approach, is the *finite-element* method (Unit II), which solves the PDE for small geometric elements, and then matches the elements.

To derive the finite-difference algorithm for the numeric solution of (23.5), we follow the same path taken in Section 6.1 to derive the forward-difference algorithm for differentiation. We start by adding the two Taylor expansions of the potential to the right and left of (x,y):

$$U(x+\Delta x, y) = U(x,y) + \frac{\partial U}{\partial x}\Delta x + \frac{1}{2}\frac{\partial^2 U}{\partial x^2}(\Delta x)^2 + \cdots \tag{23.20}$$

$$U(x-\Delta x, y) = U(x,y) - \frac{\partial U}{\partial x}\Delta x + \frac{1}{2}\frac{\partial^2 U}{\partial x^2}(\Delta x)^2 - \cdots \tag{23.21}$$

All odd terms cancel when we add these equations together, and we obtain a central-difference approximation for the second partial derivative good to order $(\Delta x)^4$:

$$\Rightarrow \quad \frac{\partial^2 U(x,y)}{\partial x^2} \simeq \frac{U(x+\Delta x,y) + U(x-\Delta x,y) - 2U(x,y)}{(\Delta x)^2} + \mathcal{O}(\Delta x^4)$$

Likewise, we add the two Taylor expansions of the potential above and below (x,y) to obtain

$$\frac{\partial^2 U(x,y)}{\partial y^2} \simeq \frac{U(x,y+\Delta y) + U(x,y-\Delta y) - 2U(x,y)}{(\Delta y)^2} + \mathcal{O}(\Delta y^4) \tag{23.22}$$

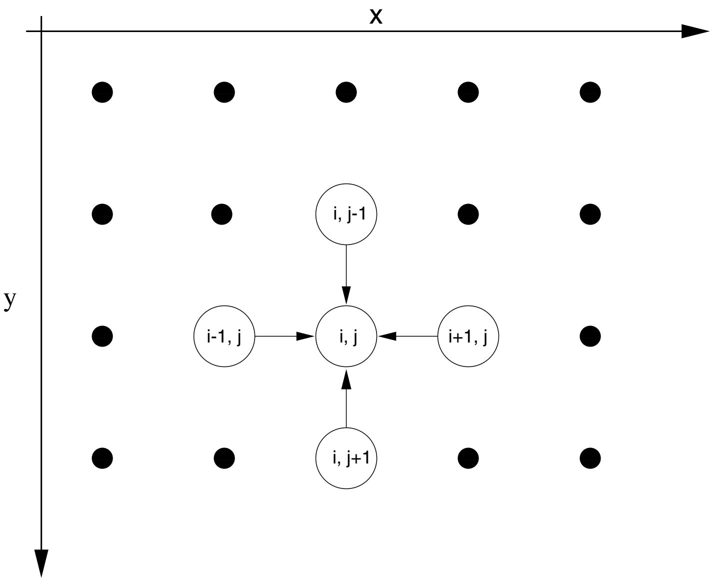

Fig. 23.3 The algorithm for Laplace's equation in which the potential at the point $(x, y) = (i, j)\Delta$ equals the average of the potential values at the four nearest-neighbor points. The nodes with white centers correspond to fixed values of the potential along the boundaries.

Substituting both these approximations into Poisson's PDE (23.5) leads to a finite-difference form of the equation:

$$\begin{aligned}&\frac{U(x+\Delta x, y) + U(x-\Delta x, y) - 2U(x, y)}{(\Delta x)^2}\\ &+ \frac{U(x, y+\Delta y) + U(x, y-\Delta y) - 2U(x, y)}{(\Delta y)^2} = -4\pi\rho \end{aligned} \tag{23.23}$$

If the x and y grids are of equal spacings, this takes the simple form

$$U(x+\Delta x, y) + U(x-\Delta x, y) + U(x, y+\Delta y) + U(x, y-\Delta y) - 4U(x, y) = -4\pi\rho \tag{23.24}$$

The reader will notice that this equation is a relation among the solutions at five points in space. When $U(x, y)$ is evaluated for the N_x x values on the lattice, and for the N_y y values, there results a set of $N_x \times N_y$ simultaneous linear algebraic equations for `U[i][j]`. One approach is to solve these linear equations explicitly as a (big) matrix problem, using the computer to do the matrix algebra. This is attractive as it is a direct solution, but it requires a great deal of memory and book keeping.

The approach we follow here is a simple one. We solve (23.24) for $U(x,y)$:

$$U(x,y) \simeq \tfrac{1}{4}\left[U(x+\Delta,y)+U(x-\Delta,y)+U(x,y+\Delta)+U(x,y-\Delta)\right] + \pi\rho(x,y)\Delta^2 \qquad (23.25)$$

where we would just leave off the $\rho(x)$ term for Laplace's equation. In terms of discrete locations on our lattice, the x and y variables are:

$$x = x_0 + i\Delta, \qquad y = y_0 + j\Delta \qquad i,j = 0,\dots,N_{\text{max-1}} \qquad (23.26)$$

$$\Delta x = \Delta y = \Delta = L/(N_{\text{max}}-1) \qquad (23.27)$$

where we have placed our lattice in a square of side L. The potential is represented by the array $U[N_{\text{max}}][N_{\text{max}}]$, and the finite-difference algorithm (23.25) becomes

$$U_{i,j} = \tfrac{1}{4}\left[U_{i+1,j}+U_{i-1,j}+U_{i,j+1}+U_{i,j-1}\right] + \pi\rho(i\Delta,j\Delta)\Delta^2 \qquad (23.28)$$

This equation says that when we have a proper solution it will be the average of the potential at the four nearest neighbors (Fig. 23.3), plus a contribution from the local charge density. As an algorithm, (23.28) does not provide a direct solution to Poisson's equation, but rather must be repeated many times to converge upon the solution. We start with an initial guess for the potential, improve it by sweeping through all space taking the average over nearest neighbors at each node, and keep repeating the process until the solution no longer changes to some level of precision, or until failure is evident. When converged, the initial guess is said to have *relaxed* into the solution.

A reasonable question with this simple approach is "Does it always converge, and if so, does it converge fast enough to be useful?" In some sense the answer to the first question is not an issue; if the method does not converge then we will know it, else, we have ended up with a solution and the path we followed to get there does not matter! The answer to the question of speed is that relaxation methods may converge slowly (although still faster than a Fourier series), yet we will show you two clever tricks to accelerate the convergence.

At this point it is important to remember that our algorithm arose from expressing the Laplacian ∇^2 in rectangular coordinates. While this does not restrict us from solving problems with circular symmetry, there may be geometries where it is better to develop an algorithm based on expressing the Laplacian in the cylindrical or spherical coordinates in order to have grids that fit the geometry better.

23.4.1
Relaxation and Over-Relaxation

There are a number of variations in how to iterate the algorithm (23.25), and so continually convert the initial boundary conditions into a solution. Its most basic form is the ***Jacobi method***, and is one in which the potential values are not updated until an entire sweep of applying (23.25) at each point is completed. This maintains the symmetry of the initial guess and boundary conditions.

A rather obvious improvement of the Jacobi method employs the updated guesses for the potential in (23.25) as soon as they are available. As a case in point, if the sweep starts in the upper left-hand corner of Fig. 23.3, then the left-most `(i-1,j)` and top-most `(i,j-1)` values of the potential used will be from the present generation of guesses, while the other two values of the potential will be from the previous generation:

$$U_{i,j}^{(\text{new})} = \tfrac{1}{4}\left[U_{i+1,j}^{(\text{old})} + U_{i-1,j}^{(\text{new})} + U_{i,j+1}^{(\text{old})} + U_{i,j-1}^{(\text{new})} \right] \qquad \text{(GS)} \qquad (23.29)$$

This technique, known as the ***Gauss–Seidel method***, usually leads to accelerated convergence, which in turn leads to less roundoff error. It also uses less memory as there is no need to store two generations of guesses. However, it does distort the symmetry of the boundary conditions or the initial guess, which one hopes is insignificant when convergence is reached.

A less obvious improvement to the relaxation technique, known as ***successive over-relaxation*** (SOR), starts by writing the algorithm (23.25) in a form that determines the new values of the potential $U^{(\text{new})}$ as the old values $U^{(\text{old})}$ plus a correction or residual r:

$$U_{i,j}^{(\text{new})} = U_{i,j}^{(\text{old})} + r_{i,j} \qquad (23.30)$$

While the Gauss–Seidel technique may still be used to incorporate the updated values in $U^{(\text{old})}$ to determine r, we just rewrite the algorithm in the general form:

$$\begin{aligned} r_{i,j} &\equiv U_{i,j}^{(\text{new})} - U_{i,j}^{(\text{old})} \\ &= \tfrac{1}{4}\left[U_{i+1,j}^{(\text{old})} + U_{i-1,j}^{(\text{new})} + U_{i,j+1}^{(\text{old})} + U_{i,j-1}^{(\text{new})} \right] - U_{i,j}^{(\text{old})} \end{aligned} \qquad (23.31)$$

The successive over-relaxation technique [9,72] proposes that if convergence is obtained by adding r to U, then even more rapid convergence might be obtained by adding more of r:

$$U_{i,j}^{(\text{new})} = U_{i,j}^{(\text{old})} + \omega\, r_{i,j} \qquad \text{(SOR)} \qquad (23.32)$$

where ω is a parameter that amplifies, or reduces, the effect of the residual. The nonaccelerated relaxation algorithm (23.29) is obtained with $\omega = 1$, accelerated convergence (over relaxation) is obtained with $\omega \geq 1$, and under relaxation occurs for $\omega < 1$. Values of $\omega \leq 2$ often work well, yet $\omega > 2$ may lead to numerical instabilities. Although a detailed analysis of the algorithm is needed to predict the optimal value for ω, we suggest that you explore different values for ω to see which one works best for your particular problem.

Listing 23.1: `LaplaceLine.java` is the framework for the solution of Laplace's equation via relaxation. The various parameters need to be adjusted for a good and realistic solution.

```
/* LaplaceLine.java: Laplace eqn via finite difference mthd
                      wire in grounded box, Output for 3D gnuplot */

import java.io.*;

public class LaplaceLine  {
  static int  Nmax = 100;                                  // Size of box

  public static void main(String[] argv)
                        throws IOException, FileNotFoundException {

    double V[][] = new double[Nmax][Nmax];
    int i, j, iter;

    PrintWriter w =   new PrintWriter              // Save data in file
                     (new FileOutputStream("LaplaceLine.dat"), true);
                                                        // Initialize
    for (i=0;  i<Nmax;   i++)
       { for (j=0;  j<Nmax;   j++) V[i][j] = 0.; }
    for ( i=0;  i < Nmax;   i++ ) V[i][0] = 100. ;   // V(wire) = 100 V
                                                          // Iterations
    for ( iter=0;  iter < 1000;    iter++ ) {
                                                 // x, then y directions
      for ( i=1;  i < (Nmax-1);  i++ )   {
        for ( j=1;  j < (Nmax-1);  j++ )
                                                       // THE ALGORITM
          V[i][j] = (V[i+1][j]+V[i-1][j]+V[i][j+1]+V[i][j-1])/4.;
      }
    }
    for ( i=0;  i < Nmax ;  i=i + 2)  {       // Data in gnuplot format
      for ( j=0;  j < Nmax;   j=j + 2) w.println("" + V[i][j] + "");
      w.println("");                          // Blank line separates rows
    }
    System.out.println("data stored in LaplaceLine.dat");
  }                                                          // End main
}                                                           // End class
```

23.4.2
Lattice PDE Implementation

In Listing 23.1 we present the code `LaplaceLine.java` that solves the square-wire problem (Fig. 23.1). Here we have kept the code simple by setting the length of the box $L = N_{\max}\Delta = 100$, taking $\Delta = 1$:

$$\begin{array}{rclllrcll} U(i, N_{\max}) & = & 99 & \text{(top)} & & U(1, j) & = & 0 & \text{(left)} \\ U(N_{\max}, j) & = & 0 & \text{(right)} & & U(i, 1) & = & 0 & \text{(bottom)} \end{array} \qquad (23.33)$$

We also run the algorithm (23.28) for a fixed number, 1000 iterations. A better code would vary Δ and the dimensions, and would quit iterating once the solution converges. Study, compile, and execute the basic code.

23.5 Assessment via Surface Plot

After executing `LaplaceLine.java`, you should have a file with data in the format appropriate for a surface plot like Fig. 23.1. Seeing that it is important to visualize your output to ensure the reasonableness of the solution, you should learn how to make such a plot before exploring more interesting problems. The 3D surface plots we show in this chapter were made with *gnuplot*. Here we repeat the commands used for Gnuplot with the output file of `LaplaceLine.java`:

```
> gnuplot                                  Start gnuplot system from a shell
gnuplot> set hidden3d                      Hide surface whose view is blocked
gnuplot> set unhidden3d                    Show surface though hidden from view
gnuplot> splot 'Laplace.dat' with lines    Surface plot of Laplace.dat with lines
gnuplot> set view 65,45                    Set x and y rotation viewing angles
gnuplot> replot                            See effect of your change
gnuplot> set contour                       Project contours onto the x–y plane
gnuplot> set cntrparam levels 10           10 contour levels
gnuplot> set terminal PostScript           Output in PostScript format for printing
gnuplot> set output "Laplace.ps"           Output to file Laplace.ps
gnuplot> splot 'Laplace.dat' w l           Plot again, output to file
gnuplot> set terminal x11                  To see output on screen again
gnuplot> set title 'Potential V(x,y) vs x,y'   Title graph
gnuplot> set xlabel 'x Position'           Label x axis
gnuplot> set ylabel 'y Position'           Label y axis
gnuplot> set zlabel 'V(x,y)'; replot       Label z axis and replot
gnuplot> help                              Tell me more
gnuplot> set nosurface                     Do not draw surface, leave contours
gnuplot> set view 0, 0, 1                  Look down directly onto base
gnuplot> replot                            Draw plot again; may want to write to file
gnuplot> quit                              Get out of gnuplot
```

Here we have explicitly stated the viewing angle for the surface plot. Because gnuplot 4 permits you to rotate surface plots interactively, we recommend that you do just that to find the best viewing angle. Changes made to a plot are seen when you redraw the plot using the `replot` command. For this sample session, the default output for your graph is your terminal screen. To print a paper copy of your graph we recommend first saving it to a file as a *PostScript*

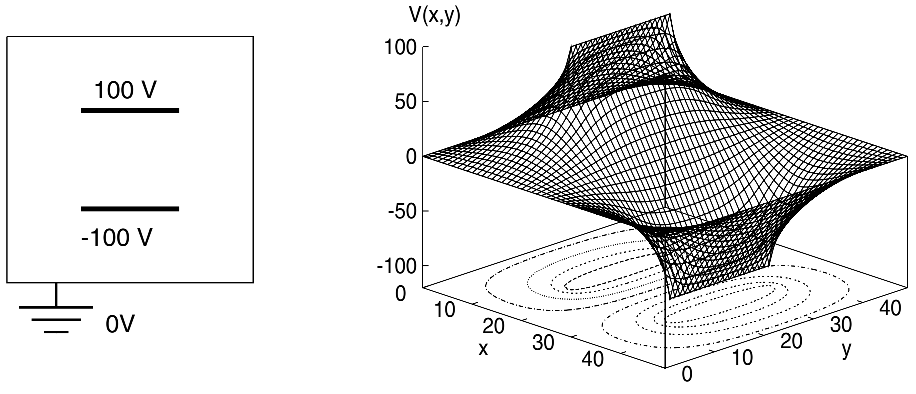

Fig. 23.4 *Left*: A simple model of a parallel-plate capacitor (or of a vacuum-tube diode). A realistic model would have the plates closer together, in order to condense the field, and the enclosing, grounded box so large that it has no effect on the field near the capacitor. *Right*: A numerical solution for the electric potential for this geometry. The projection on the xy plane gives the equipotential lines.

document (suffix `.ps`), and then printing out that file to a PostScript printer. You create the PostScript file by changing the terminal type to `Postscript`, setting the name of the file, and then issuing the subcommand `splot` again. This plots the result out to a file. If you want to see plots on your screen again, you need to set the terminal type back to `x11` again (for Unix's *X Windows System*), and then plot it again.

23.6 Three Alternate Capacitor Problems

We give you (or your instructor) a choice now. You can carry out the assessment using our wire-plus-grounded box problem, or you can replace that problem with the more interesting one of a realistic capacitor. We now describe the capacitor problem, and then get on to the assessment and exploration.

The standard solution for a capacitor's field is for the region between two infinite plates, where the field is uniform and completely confined. The field in a finite capacitor will vary near the edges ("edge effects"), and extend beyond the edges of the capacitor ("fringe fields"). We model the realistic capacitor in a grounded box (Fig. 23.4) as two plates (wires) of finite length. Write your program such that it is convenient to vary the grid spacing Δ and the geometry of the box and plate, without having to get into the details of the program. We pose three versions of this problem, each displaying some different physics. In each case the boundary conditions $V = 0$ along the box must be satisfied for all steps during the solution process.

1. For the simplest version, assume that the plates are very thin sheets of conductors, with the top plate maintained at 100 V and the bottom at -100 V. Because the plates are conductors, they must be equipotential surfaces, and a battery can maintain them at constant voltages. Write or modify the given program to solve Laplace's equation such that the plates have fixed voltages.

2. For the next version of this problem, assume that the plates are made of a thin dielectric material with uniform charge densities ρ on the top and $-\rho$ on the bottom. Solve Poisson's equation (23.3) in the region including the plates, and Laplace's equation elsewhere. Experiment until you find a numerical value for ρ that gives a similar potential to that shown in Fig. 23.4 for plates with fixed voltages.

3. For the final version of this problem we would like to investigate how the charges on the conducting plates of the capacitor distributes themselves. To do that, assume that the plates have a finite thickness (Fig. 23.6). Since the plates are conductors, we can still assume that they are equipotential surfaces at 100 and -100 V, only now we want them to have a thickness of at least 2Δ (so we can see the difference between the potential near the top and the bottom of the plates). Such being the case, we solve Laplace's equation (23.4) much as before to determine $U(x, y)$. Once we have $U(x, y)$, substitute it into Poisson's equation (23.3) and determine how the charge density distributes itself along the top and bottom surfaces of the plates. *Hint:* Since the electric field is no longer uniform, we know that the charge distribution also will no longer be uniform. In addition, since the electric field now extends beyond the ends of the capacitor, and since field lines begin and end on charge, we suspect that some charge may end up on the edges and outer surfaces of the plates (Fig. 23.5).

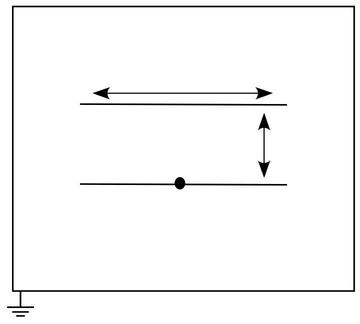

Fig. 23.5 A suggested scheme for labeling the geometry of a parallel-plate capacitor within a box.

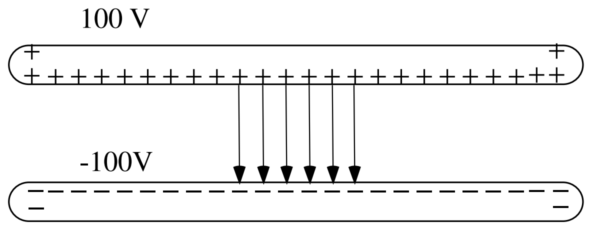

Fig. 23.6 An indication of how the electric charge may rearrange itself on a capacitor with conducting plates.

23.7 Implementation and Assessment

1. Write a program or modify the one on the CD to find the electric potential for a capacitor within a grounded box. (In Listing 23.2 we list our version of successive over relaxation.) Use the labeling scheme in Fig. 23.5 and make your program general enough so that you can increase the size of the box, or the geometry of the capacitor, without having to rewrite the program.

2. Our sample program undertakes 1000 iterations and quits. You probably want to stick with this assumption until you have your program working. As least during debugging, examine how the potential changes in some key locations as you iterate toward a solution.

3. Repeat the process for different step sizes Δ and judge if the process is stable and convergent for all sizes.

4. Once your program produces a reasonable graph, modify the program so it stops iterating after convergence is reached (or after a very large number of iterations, in case there is no convergence). Rather than try to discern small changes in highly compressed surface plots, use a numerical measure of precision. While it might be best to test for convergence throughout the entire box, this could require comparing millions of values, most of which are so close to zero that their precision does not matter anyway. Instead, look at the surface plot of the potential in Fig. 23.4, and observe how the line along the diagonal samples the whole range of the potential:

   $$\texttt{trace} = \sum_i |\texttt{V[i][i]}| \tag{23.34}$$

   Print out the value of `trace` for each iteration and note how much precision is obtained. You may well need hundreds or thousands of iterations to obtain stable and accurate answers; this is part of the price paid for simplicity of algorithm.

5. Once you know the value of `trace` corresponding to the best precision you can obtain, incorporate it in an automated test of convergence. To illustrate, define `tol = 0.0001` as the desired relative precision and modify `LaplaceLine.java` so that it stops iterating when the relative change from one iteration to the next is less than `tol`:

$$\left|\frac{\texttt{trace - traceOld}}{\texttt{traceOld}}\right| < \texttt{tol} \tag{23.35}$$

   The `break` command or a `while` loop is useful for this type of iteration.

6. Equation (23.32) expresses the successive over-relaxation technique in which convergence if accelerated by using a judicious choice of ω. Determine by trial and error the approximate best value of ω. Often $\omega \leq 2$ speeds up the convergence by a factor of approximately two.

7. Now that you know the code is accurate, modify it to simulate a more realistic capacitor in which the plate separation is approximately 1/10th of the plate length. You should find the field more condensed and more uniform between the plates.

8. If you are working with the wire-in-the-box problem, compare your numerical solution to the analytic one (23.18). Do not be surprised if you need to sum thousands of terms before the "analytic" solution converges!

Listing 23.2: `LaplaceSOR.java`solves Laplace's equation with successive over relaxation.

```
// LaplaceSOR.java: Solves Laplace eqn using SOR for convergence

import java.io.*;


public class LaplaceSOR  {
  static int  max =  40;                          // Number of grid points

  public static void main(String[] argv) throws IOException,
                                              FileNotFoundException  {

   double x, tol, aux, omega, r, pi  =  3.1415926535;
   double p[][] = new double[max][max];
   int i, j, iter;

   PrintWriter w =new PrintWriter(
                         new FileOutputStream("laplR.dat"), true);
   omega = 1.8;                                          // SOR parameter
   long timeStart=System.nanoTime();
   System.out.println(""+timeStart+"");
                                                                 // Init
   for (i=0; i<max; i++) for ( j=0; j<max; j++ ) p[i][j]  =  0.;
```

```
    for (i = 0; i<max; i++) p[i][0]  =  +100.0;
    tol  =  1.0;                                             // Tolerance
    iter  =  0;
                                                             //  Iterations
    while ( (tol > 0.000001) && (iter <=  140) ) {
      tol  =  0.0;
                                                   // x, then y directions
      for (i = 1; i<(max-1); i++)  {
        for (j = 1; j<(max-1); j++)   {
                                                          // SOR ALGORITHM
          r  =  omega * ( p[i][j+1] + p[i][j-1] + p[i+1][j] +
                          p[i-1][j] - 4.0 * p[i][j] ) / 4.0;
          p[i][j] +=  r;
          if ( Math.abs(r) > tol ) tol  =  Math.abs(r);
        }
      iter++;
       }
     }
     long timeFinal=System.nanoTime();
     System.out.println(""+timeFinal+"");
     System.out.println("Nanoseconds="+(timeFinal-timeStart));
     for (i = 0; i<max ; i++)  {
       for (j = 0; j<max; j++) w.println(""+p[i][j]+"");
      w.println("");                             // Empty line for gnuplot
     }
     System.out.println("data stored in laplR.dat");
}  }
```

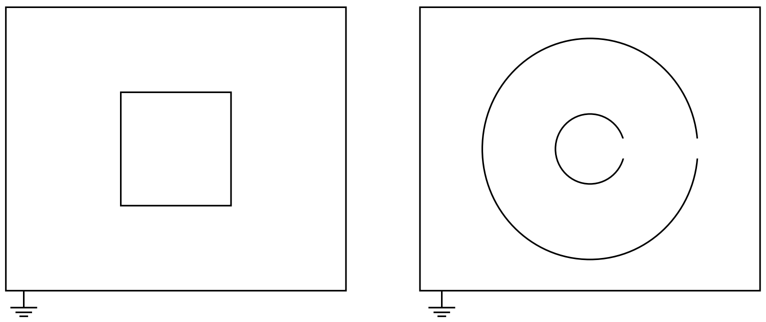

Fig. 23.7 *Left:* The geometry of a capacitor formed by placing two long, square cylinders within each other. *Right:* The geometry of a capacitor formed by placing two long, circular cylinders within each other. The cylinders are cracked on the side so that wires can enter the region.

23.8
Other Geometries and Boundary Conditions (Exploration)

The numerical solution to our PDE can be applied to arbitrary boundary conditions. Two boundary conditions to explore are triangular and sinusoidal:

$$U(x) = \begin{cases} 200\frac{x}{w} & \text{for } x \le w/2 \\ 100(1 - \frac{x}{w}) & \text{for } x \ge w/2 \end{cases} \qquad U(x) = 100 \sin\left(\frac{2\pi x}{w}\right)$$

Square Conductors: You have designed a piece of equipment consisting of a small metal box at 100 V within a larger, grounded one (Fig. 23.7). You find that sparking occurs between the boxes, which means that the electric field is too large. You need to determine where the field is greatest so that you can change the geometry and eliminate the sparking.

Modify the program to satisfy these boundary conditions and to determine the field between the boxes. Gauss's law tells us that the electric field vanishes within the inner box because it contains no charge. Plot the potential and equipotential surfaces, and sketch in the electric field lines. Deduce where the electric field is most intense and try redesigning the equipment to reduce the field.

Cracked Cylindrical Capacitor: You have designed the cylindrical capacitor containing a long outer cylinder surrounding a thin inner cylinder (Fig. 23.7 right). The cylinders have a small crack in them in order to connect them to the battery that maintains the inner cylinder at -100 V and outer cylinder at 100 V. Determine how this small crack affects the field configuration. In order for a unique solution to exist for this problem, place both cylinders within a large grounded box. Note, since our algorithm is based on the expansion of the Laplacian in rectangular coordinates, you cannot just convert it into a radial and angle grid.

Thinking Outside the Box$\odot$: Find the electric potential for all points *outside* the charge-free square shown in Fig. 23.1. Is your solution unique?