

# Relaxation Methods for Partial Differential Equations: Applications to Electrostatics\*

David G. Robertson<sup>†</sup>

*Department of Physics and Astronomy  
Otterbein University, Westerville, OH 43081*

(Dated: December 8, 2010)

## Abstract

Approaches for numerically solving elliptic partial differential equations such as that of Poisson or Laplace are discussed. The basic technique is that of “relaxation,” an iterative scheme based on a discretization of the domain of interest. Applications to problems in electrostatics in two and three dimensions are studied. Extensions including overrelaxation and the multigrid method are described.

Keywords: electrostatics, Poisson equation, Laplace equation, electric potential, electric field, relaxation, overrelaxation, multigrid technique, boundary value problem

---

\* Work supported by the National Science Foundation under grant CCLI DUE 0618252.

<sup>†</sup> drobertson@otterbein.edu

## CONTENTS

I. Module Overview	3
II. What You Will Need	4
III. Physics Background	4
IV. Discretization and the Relaxation Method	6
A. Exercises	12
V. Simulation Projects	13
VI. Elaborations	14
A. Overrelaxation	14
B. Multigrid Methods	15
C. Arbitrary Boundary Shapes	19
A. Gaussian Units in Electrodynamics	20
References	21
Glossary	22

## I. MODULE OVERVIEW

In this module we will study the numerical solution of elliptic partial differential equations using relaxation techniques. A typical example is Laplace’s equation,

$$\nabla^2 V = 0, \tag{1.1}$$

which determines the electric potential in a source-free region, given suitable boundary conditions, or the steady-state temperature distribution in matter.

We shall focus here on applications to electrodynamics. In that context the potential  $V$  is related to the potential energy of charges; it also gives the electric field, via the relation

$$\vec{E} = -\vec{\nabla}V. \tag{1.2}$$

Determining  $V$  is generally much easier than calculating  $\vec{E}$  directly, since it is a scalar quantity.

Courses in electromagnetism typically devote considerable time to developing solutions to Laplace’s equation in analytically tractable cases. These are limited, however, to situations where the boundary geometry is especially simple and maps onto a standard coordinate system, e.g., cartesian, cylindrical or spherical coordinates. In other cases – meaning virtually all cases of practical interest – analytical techniques are not useful and a numerical approach is necessary.

In this module we will discuss one approach to this problem, based on “relaxation.” This is only one of several possible approaches, and is itself a large subject with many technical aspects. Our goal will be to become familiar with the basic techniques and issues; readers interested in more details should continue by consulting a standard book on numerical analysis [1–3].

The ability to produce numerical solutions to the Laplace or Poisson equations for arbitrary geometries will allow us to explore many aspects of electrostatics, making concrete the vector calculus relations that connect the basic concepts and aiding in the development of intuition and understanding. Ideally it will sharpen your understanding of the physics as well as your programming skills.

## II. WHAT YOU WILL NEED

You should have had exposure to the Laplace and Poisson equations and the standard techniques (based on separation of variables) for solving them, the relation between electric field and potential, properties of conductors, and so on. If you are taking or have passed through a traditional junior-level course on electromagnetism you should be well equipped, and you should also have sufficient facility with the relevant mathematics.

The computing resources needed are actually rather modest for the problems we will consider. The natural framework for scientific computing is a high-level language like Fortran, C or C++, and this is an excellent project for you to sharpen their programming skills. However, the needed calculations can be done using Matlab or Mathematica, or even with a spreadsheet such as Excel, though this will limit significantly the size of the problem that can be considered.

Some facility for generating plots, particularly contour plots, will also be essential. `Gnuplot` [4] is a good option, but data can also be imported into Matlab, Mathematica or Excel for this purpose.

## III. PHYSICS BACKGROUND

The equations of Poisson and Laplace are of central importance in electrostatics (for a review, see any textbook on electrodynamics, for example [5]). For a region of space containing a charge density  $\rho(\vec{x})$ , the electrostatic potential  $V$  satisfies Poisson's equation:

$$\nabla^2 V = -4\pi\rho, \tag{3.1}$$

where we have adopted cgs (Gaussian) units. (See the Appendix for a quick review.) The differential operator

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \tag{3.2}$$

is known as the Laplace operator or laplacian. In a source-free region ( $\rho = 0$ ), eq. (3.1) reduces to Laplace's equation,

$$\nabla^2 V = 0. \tag{3.3}$$

Together with suitable boundary conditions on the region of interest, eq. (3.1) or (3.3) determines the electrostatic potential uniquely. The significance of  $V$  derives primarily from

its connection to the electric field,

$$\vec{E} = -\vec{\nabla}V, \quad (3.4)$$

where

$$\vec{\nabla} = \hat{x} \frac{\partial}{\partial x} + \hat{y} \frac{\partial}{\partial y} + \hat{z} \frac{\partial}{\partial z}. \quad (3.5)$$

In addition, the potential energy of a test charge  $q$  at a location  $\vec{x}$  is given by

$$U = qV(\vec{x}). \quad (3.6)$$

As mentioned above, the solution to Laplace's or Poisson's equation requires the specification of *boundary conditions* on the domain of interest. There are several possibilities. We can specify the value of  $V$  itself on the boundary (Dirichlet condition), or the derivative of  $V$  in the direction normal to the boundary (Neumann condition). (Mixed conditions involving both  $V$  and  $dV/dn$  are also possible, though we will not consider these here.) In some cases we can also impose *periodic* (or anti-periodic) boundary conditions, which effectively wraps the domain around on itself, or equivalently generates neighboring "images" of the domain. All of these possibilities can be useful depending on the specific physical context.

A nice feature of a concrete numerical formulation of the problem is that it shows *why* such conditions are necessary to determine the solution uniquely. We shall explore this point below.

As a simplifying feature we will mainly consider problems in two dimensions, rather than three – this makes visualization easier, in particular. In this case  $V = V(x, y)$  only. Note that this is also appropriate for three-dimensional problems in which nothing varies with  $z$  or, equivalently, systems that are effectively infinite in the  $z$ -direction. As an example, consider a sheet of charge lying in the  $x$ - $z$  plane, of some finite width  $L$  in the  $x$ -direction and infinite in the  $z$ -direction. In this case  $V$  will depend only on  $x$  and  $y$ ; by symmetry, there can be no component of  $\vec{E}$  in the  $z$ -direction, so  $\partial V/\partial z$  must vanish. Thus the problem is effectively two-dimensional, with charge density  $\rho(x, y)$  that of a finite line of charge lying parallel to the  $x$ -axis. The total charge of this line, in the two-dimensional sense, should be thought of as the charge per unit length (in  $z$ ), in the three-dimensional sense. Likewise, a capacitance calculated in the two-dimensional problem should be interpreted as a capacitance per unit length in the three-dimensional sense, etc.

For more on the relation between two- and three-dimensional problems, see exercise IV.A.3.

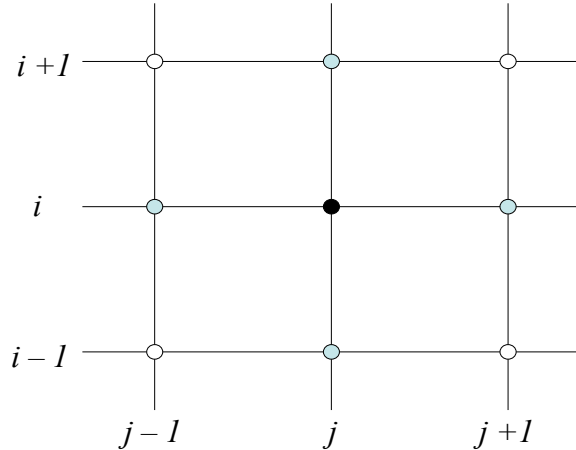


FIG. 1. Part of the uniform grid introduced to discretize Laplace’s equation. The black dot is the  $(i, j)$  grid point; the shaded dots are the nearest-neighbor points involved in the approximate second derivatives.

#### IV. DISCRETIZATION AND THE RELAXATION METHOD

We first discretize the problem by introducing a mesh of grid points that covers the domain of interest. We work in two dimensions, and for simplicity we assume a cartesian grid with the same spacing  $h$  between grid points in each direction. (Later we will discuss other coordinate systems, e.g., polar coordinates.) Fig. 1 shows a section of this grid.

We label the grid points by a pair of indices  $i, j$ , which just count grid points from some reference point, say the lower left hand corner. Thus assuming the reference point has  $(x, y) = (0, 0)$ , the  $x$  and  $y$  coordinates of the  $i, j$  grid point are

$$x_i = ih, \quad y_j = jh. \tag{4.1}$$

The basic idea is to calculate the approximate values of  $V$  at the grid points,  $V(x_i, y_j) \equiv V_{i,j}$ . The accuracy of the approximation should improve as the number of mesh points is increased, that is, as  $h$  shrinks toward zero for a fixed physical domain size.

The next step is to translate the differential operator  $\nabla^2$  into discrete form, i.e., to obtain a representation of the needed second derivatives in terms of finite differences. There is actually considerable freedom in how this is done, and we will only touch on the basics. To motivate one approach, consider first a single second derivative,

$$\frac{d^2V}{dx^2} = 0. \tag{4.2}$$

Imagine that we Taylor expand the function  $V$  about the point  $x$ :

$$V(x+h) = V(x) + h \frac{dV}{dx} + \frac{1}{2} h^2 \frac{d^2V}{dx^2} + \dots \quad (4.3)$$

where all derivatives are evaluated at the point  $x$ . (In the full problem these are all partial derivatives, of course.) Eq. (4.3) implies that

$$V(x+h) + V(x-h) = 2V(x) + h^2 \frac{d^2V}{dx^2} + \mathcal{O}(h^4). \quad (4.4)$$

If we now drop the higher order terms as unimportant for sufficiently small  $h$ , then this can be interpreted as an approximate formula for the second derivative:

$$\frac{d^2V}{dx^2} \approx \frac{V(x+h) + V(x-h) - 2V(x)}{h^2}. \quad (4.5)$$

This difference formula is analogous to, e.g.,

$$\frac{dV}{dx} \approx \frac{V(x+h) - V(x)}{h} \quad (4.6)$$

which gives a discrete approximation to the first derivative. Actually, it is more analogous to the “symmetric” first derivative:

$$\frac{dV}{dx} \approx \frac{V(x+h) - V(x-h)}{2h}, \quad (4.7)$$

showing that there is indeed freedom in how these approximations are constructed. Both eqs. (4.6) and (4.7) approach the exact derivative as  $h \rightarrow 0$  and so either is a perfectly valid approximation for finite  $h$ . The latter is often preferred, however, because it is more accurate for a given value of  $h$ . You can explore this point in the exercises.

A precisely analogous formula holds for the second partial derivative with respect to  $y$ , of course, and combining these together gives the desired discrete approximation to the laplacian:

$$\begin{aligned} \nabla^2 V \approx & \frac{V(x+h, y) + V(x-h, y) - 2V(x, y)}{h^2} \\ & + \frac{V(x, y+h) + V(x, y-h) - 2V(x, y)}{h^2}. \end{aligned} \quad (4.8)$$

Switching to the grid notation where  $V(x \pm h, y) = V_{i \pm 1, j}$  and  $V(x, y \pm h) = V_{i, j \pm 1}$ , the discretized version of Laplace’s equation becomes

$$V_{i+1, j} + V_{i-1, j} + V_{i, j+1} + V_{i, j-1} - 4V_{i, j} = 0. \quad (4.9)$$

This can be re-written in the form

$$V_{i,j} = (V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1})/4, \quad (4.10)$$

which says that a solution of Laplace's equation has the (remarkable) property that at any point it is equal to the average of the values at neighboring points. This is simple to appreciate in one dimension, where Laplace's equation reads

$$\frac{d^2V}{dx^2} = 0, \quad (4.11)$$

and has as its solution

$$V = ax + b, \quad (4.12)$$

where  $a$  and  $b$  are constants. Certainly a linear function has the property that its value at any point is equal to the average of the values at neighboring points (assuming of course that the neighbors are the same distance from the point in question).

We can also use eq. (4.10) to motivate an iterative scheme for finding solutions to Laplace's equation. We just guess a solution initially, then sweep across the grid updating the value at each point according to eq. (4.10), that is, on each iteration we set  $V_{i,j}$  at every grid point to the average value of its nearest neighbors. The result will not be exact – at least in the beginning – because the neighbors will also get updated, but by sweeping again and again we should converge to a solution. When the change from one iteration to the next is less than some specified tolerance, we declare the solution converged and quit. This procedure is known as *relaxation*.

There is a physical way of thinking about this that involves imagining that  $V$  depends on time,  $V = V(x, y, t)$  and considering the diffusion equation,

$$\frac{\partial V}{\partial t} = D\nabla^2V, \quad (4.13)$$

where  $D$  is a constant. Solutions to this equation will eventually “relax” to a steady state in which the time derivative vanishes; the steady-state solution then satisfies the Laplace equation. Let's discretize in time as well as space, by introducing

$$V(x, y, t) \rightarrow V_{i,j}^n \quad (4.14)$$

where  $n$  labels the time slice, and approximating the time derivative as

$$\frac{\partial V}{\partial t} \approx \frac{V_{i,j}^{n+1} - V_{i,j}^n}{\epsilon}, \quad (4.15)$$



where  $\epsilon$  is the interval between time slices. The discrete diffusion equation then becomes

$$\frac{V_{i,j}^{n+1} - V_{i,j}^n}{\epsilon D} = \frac{V_{i+1,j}^n + V_{i-1,j}^n + V_{i,j+1}^n + V_{i,j-1}^n - 4V_{i,j}^n}{h^2}. \quad (4.16)$$

Next let us choose for simplicity

$$\epsilon D = h^2/4. \quad (4.17)$$

Then eq. (4.16) can be written as

$$V_{i,j}^{n+1} = (V_{i+1,j}^n + V_{i-1,j}^n + V_{i,j+1}^n + V_{i,j-1}^n)/4. \quad (4.18)$$

This is just the iterative scheme we outlined above, with iterations labeled by the “time.” That is, we can think of the relaxation as time evolution towards a steady state configuration that satisfies Laplace’s equation.

What happens if charges are present? In this case the charge density  $\rho(\vec{x})$  must also be replaced by a discrete distribution  $\rho_{i,j} = \rho(x_i, y_j)$ . Poisson’s equation then takes the discrete form

$$V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1} - 4V_{i,j} = -4\pi\rho_{i,j}h^2. \quad (4.19)$$

Note that  $h^2\rho_{i,j}$  is just the total charge (density times volume) contained in a cell of area  $h^2$  located at grid point  $i, j$ . It should be clear that the same iterative procedure will work here as well. We simply update  $V$  across the grid as

$$V_{i,j}^{n+1} = (V_{i+1,j}^n + V_{i-1,j}^n + V_{i,j+1}^n + V_{i,j-1}^n)/4 + \pi\rho_{i,j}h^2. \quad (4.20)$$

This concrete formulation of the problem shows both why suitable boundary conditions are necessary and also how they should be incorporated. To see why they are necessary, imagine performing the grid sweep. To update the  $i, j$  grid point, we need values for its four neighboring points (plus the local charge density, if nonzero). Thus when we come to points neighboring the boundary, we will need to have the values on the boundary to complete the update. This is true even if the “boundary” is at infinity, though obviously this is impossible to simulate on a computer. Fig. 2 shows the basic situation.

Hence  $V$  must be determined on the domain boundary in order to obtain a solution.

There are several ways we could do this. The obvious one is to just specify the boundary values directly; this is the Dirichlet condition. Another is to specify the normal derivative at the boundary (Neumann). Assume for simplicity that the specific condition is

$$\frac{dV}{dn} = 0. \quad (4.21)$$

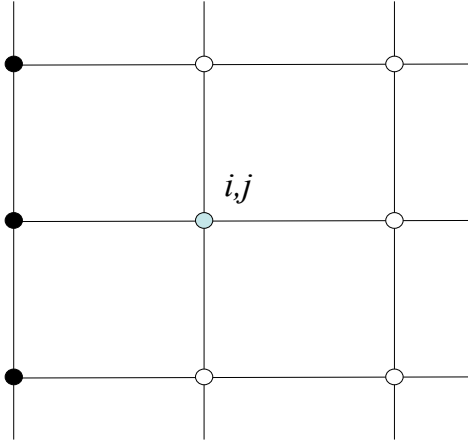


FIG. 2. Grid section near a boundary (black points). To be updated, the  $(i, j)$  grid point requires the values of its nearest neighbors.

To see how this works, discretize eq. (4.21) to give schematically

$$V_{bdy} - V_{in} = 0, \quad (4.22)$$

where  $bdy$  is a point on the boundary and  $in$  is a neighboring interior point. (The points should be separated in a direction perpendicular to that of the boundary surface, of course, to give the normal derivative.) Again we have sufficient information to perform the update: now we just use eq. (4.22) to determine the boundary value  $V_{bdy}$  after each sweep. In this case the boundary value changes from one iteration to the next. What is important is not that it is constant, but rather than it is *determined*.

Yet another possibility is to impose *periodic* boundary conditions, in which edge values “wrap around” and provide the neighbors for the opposite edge. More concretely, if there are  $N$  grid points in each direction we define

$$V_{1,j} = V_{N,j}, \quad V_{i,1} = V_{i,N}. \quad (4.23)$$

This would be appropriate for example in a crystal lattice, where the nuclear charge density is periodic in space.

To summarize, the relaxation scheme proceeds by the following general steps:

- Discretize the domain of interest, generating a difference approximation to the laplacian operator appropriate for the domain shape and coordinates. For cartesian coordinates a suitable choice is given in eq. (4.8). In the exercises you can derive a version appropriate for polar coordinates, or cylindrical geometries in three dimensions.

- Specify boundary conditions (values or normal derivatives as appropriate).
- Assign interior points an arbitrary starting value. The final solution should not depend on what is chosen here, but the solution may converge more rapidly if a “good” guess is made, one that is reasonably close to the full solution. One simple choice is to take interior points equal to the the average of the boundary values. Readers can experiment with other choices and the effect both on the rate of convergence and the final solution itself.
- Sweep through the grid, updating the values for  $V_{i,j}$  according to eq. (4.10) or whichever discrete laplacian you are using.
- Repeat the previous step until a specified level of accuracy is reached.

The last point requires some thought – how shall we decide when the relaxation has converged? Clearly, what we need is some measure of how much the solution changes in an iteration; if this change is sufficiently small we declare the solution to have converged. A convenient measure of this is the maximum relative change over the grid. That is, as we sweep we check each point and record the value of the largest relative change (change as a fraction of starting value). When this drops below some specified value we stop relaxing.

All well and good, but there is one problem: the method is generally rather slow. The basic problem is that information about updates travels very slowly through the mesh, since each point is affected only by its nearest neighbors. Hence information is communicated at only one grid step per iteration. If the grid is a modest  $100 \times 100$ , say, then it takes 100 iterations before any information from one boundary has propagated to the opposite one.

There exist methods for speeding things up, however, and we shall explore some of these below. As a first step in this direction, let us think some more about how we will perform the sweep. One way to proceed would be to compute all the new values for  $V_{i,j}$  using values from the previous iteration, and then replace the old values with the new ones. That is, we update everything before replacing any of the values. This is known as the Jacobi method.

An alternative would be to use the updated values immediately in computing the changes to their neighbors. To be concrete, assume we are sweeping through the grid by stepping along rows. In this approach we compute the new value of each point and then use it immediately in updating the next point in the row. In this way most points will be updated

using half values from the previous iteration and half from the current one. This is known as Gauss-Seidel relaxation, and it should be faster than Jacobi since some partial information is propagating throughout the grid in every sweep. A detailed analysis (beyond the scope of this module) shows that Gauss-Seidel is roughly twice as fast as Jacobi.

Once the potential is known, a variety of physically interesting quantities may be computed. The main one is the electric field, defined by

$$\vec{E} = -\vec{\nabla}V = -\left(\hat{e}_x \frac{\partial}{\partial x} + \hat{e}_y \frac{\partial}{\partial y}\right)V. \quad (4.24)$$

Derivatives are approximated as differences using eq. (4.6) or (4.7). This could be taken as the basis for a calculation of the trajectory of a charged particle through the region of interest, for example.

In addition, we can imagine that the boundary surfaces are conductors (since they are equipotentials); then the normal component of the electric field at the surface gives the induced surface charge density. In Gaussian units this relation takes the form

$$\lambda = \frac{\hat{n} \cdot \vec{E}}{4\pi}, \quad (4.25)$$

where  $\hat{n}$  is a unit normal vector pointing into the region of interest (i.e., out of the conductor). This in turn can lead to a calculation of the total charge on a boundary surface, the capacitance of a pair of conductors, and so on.

### A. Exercises

1. Using the Taylor expansion (4.3), show that the formula (4.6) is accurate to order  $h$ ; that is, that the first neglected term is  $\mathcal{O}(h)$ . Show further that the symmetric derivative (4.7) is accurate to  $\mathcal{O}(h^2)$ .
2. Obtain a discrete form for the laplacian in plane polar coordinates. Hint: start from the continuum formula and Taylor expand in  $r$  and  $\theta$ .
3. Apply Gauss's law (in integral form) in two dimensions to determine the dependence of  $|\vec{E}|$  and  $V$  on the distance  $r$  from a point charge. Verify that these are the same as for an infinite *line* of charge in three dimensions. How are the charges that appear in the two- and three-dimensional problems related?
4. Verify eq. (4.25).

## V. SIMULATION PROJECTS

In these projects we will explore a variety of problems in two-dimensional electrostatics.

### 1. **Jacobi *vs.* Gauss-Seidel**

Write a program to compute  $V$  in a rectangular domain with Dirichlet boundary conditions. Allow the potentials on all four sides to be specified arbitrarily. Relax an initial guess until the maximum relative change in one iteration, over the entire grid, is below a specified threshold.

Study the relative speed of the Jacobi and Gauss-Seidel methods. For Gauss-Seidel, does it matter in what order the points are updated, i.e., whether we sweep by rows or columns (or some other method)?

Also, does the choice of initial guess have a significant impact on overall speed? Do different initial guesses converge to the same final result? Any difference can be used to estimate the overall accuracy of the solution. Is this comparable to the tolerance you specified in the relaxation?

A tool for visualization will be essential, not just to examine the results but also as an aid in debugging. Specifically a facility for generating contour plots (equipotentials) will be helpful. The free utility `gnuplot` works very well, but data can also be imported into Excel or Matlab for this purpose.

### 2. **Electric Field**

Write a program to calculate the electric field at every mesh point from the potentials you obtain. Plot the result using the vector data option in `gnuplot` or Matlab.

### 3. **Adding Charges**

Modify your code to allow point charges to be placed at arbitrary mesh points. Compute the potential due to a single charge and an electric dipole and verify that they have the expected behavior. What should you choose for the potential on the outer boundary in these calculations?

#### 4. Parallel Sheets of Charge

- (a) Use your program to calculate the potential for parallel lines of charge. (The result is also appropriate for three-dimensional sheets of charge, infinite in extent in the direction normal to the  $x$ - $y$  plane.) How should the boundaries be handled, i.e., what sort of condition(s) should we impose?
- (b) From the configuration with equal and oppositely charged “plates,” compute the capacitance (capacitance per unit length in three dimensions) of the system. How does this compare to the analytical result that it approximates?

#### 5. Concentric Rectangles

- (a) Modify your program to solve for  $V$  between two rectangular boundaries held at constant potential.
- (b) Calculate the surface charge density on the boundary surfaces using eq. (4.25). From this, determine the total charge on each surface. How do they compare?
- (c) From the calculated charge and the known potential difference, determine the capacitance of this system.
- (d) Say we squash the rectangle so that it becomes smaller in one direction (keeping the other dimension fixed). How do you expect the capacitance to change? Check your intuition by direct calculation.

### VI. ELABORATIONS

As discussed above, relaxation tends to be rather slow. In this section we introduce some of the techniques used to improve the performance of the method. Most practical applications of relaxation are based on one or the other (or both) of these approaches. If interested, you can apply either or both to any of the above projects.

#### A. Overrelaxation

The basic idea of *overrelaxation* is to make a bigger change at each iteration than is called for in the basic approach, thus speeding the approach to the solution. Specifically, eq. (4.10)

is replaced by

$$V_{i,j}^{n+1} = \omega V_{\text{AVG}}^n + (1 - \omega)V_{i,j}^n, \quad (6.1)$$

where  $V_{\text{AVG}}$  is a shorthand for the average nearest-neighbor potential on the right hand side of eq. (4.10). The parameter  $\omega$  is known as the “overrelaxation parameter” and is in the range  $0 < \omega < 2$ . (For further details including a study of the allowed range for  $\omega$ , see, e.g., ref. [1].) If  $\omega < 1$  we speak of “underrelaxation,” if  $\omega > 1$  we have overrelaxation. The net effect when  $\omega > 1$  is to make a larger change that would be obtained otherwise. Note that when the solution has converged, so that  $V_{i,j}^{n+1} = V_{i,j}^n$ , then eq. (4.10) is again satisfied. (The terms not involving  $\omega$  in eq. (6.1) cancel, and then the factor  $\omega$  drops out.) Thus overrelaxation leads to the correct solution.

It is unfortunately not possible to make general statements about the “best” value for  $\omega$ , that is, the value that results in the fastest convergence. The answer depends on the specific problem and in most cases cannot be estimated *a priori*. Typically one performs a few iterations for different values of  $\omega$  and identifies the one that produces the biggest change; the remaining (many) iterations are then taken with that value.

## B. Multigrid Methods

A very powerful approach that can result in dramatic speedup is known the *multigrid* method. As its name suggests, it involves attacking the problem on a variety of grids, from coarser to finer. This is actually a large and technical subject, and we can give only the merest outline of the approach here. The reader who wishes to go further should consult a textbook on numerical analysis [1–3].

Recall that the basic problem is that changes propagate only slowly throughout the lattice. The idea of the multigrid approach is to tackle the problem on a variety of length scales. A coarse lattice can give rapid (albeit crude) propagation across the domain, while finer lattices give more accuracy and detail on smaller scales.

To outline the basic idea, consider a pair of grids covering the same domain. (In practice one typically uses more than two lattices, but the extension is basically obvious.) We’ll call one the “coarse” grid and the other the “fine” grid. For simplicity, we take the coarse grid to have twice the grid spacing as the fine one, and also align things so that every point on the coarse grid is also a point on the fine grid. This means that every *other* point on the

fine grid is also a coarse grid point.

Now we want to use both grids to represent the potential in the domain of interest. We thus need a way of translating a potential configuration from one grid to another. Passing from the fine to the coarse grid involves a “coarse graining” of the solution and is known as *restriction*. Passing from the coarse to the fine grid is known as *prolongation*. How shall we define the prolongation and restriction operations?

As in the case of defining discrete derivatives, there is considerable flexibility in how this is done. Any suitable scheme should give correct results in the end, although some may be more efficient than others. We shall sketch here one popular approach.

First, consider prolongation, the translation from coarse to fine. Each coarse grid point is also a point on the fine grid, so these values stay unchanged. Next, consider the points on the fine grid that lie midway between coarse grid points, i.e., have coarse grid points as nearest neighbors. For these we may take the average of the values at these neighbor points. Finally, there are points with four coarse grid points as next-to-nearest neighbors (along the diagonals). For these let us take the average of the four next-nearest neighbors; this is the same as the average of the four nearest neighbors, since these are (as above) already averages of pairs of the same four coarse grid points.

The result is to produce a potential configuration on the fine lattice that interpolates linearly between coarse grid points.

Regarding restriction, consider a “unit cell” of the grid, centered on a fine grid point (see fig. 3). The restriction operation will replace this cell with a single value at the center point, which is a point on the coarse grid. One option is to simply take the central value as defining the coarse grid value. This is known as *straight injection*, and while conceptually simple it can lead to problems in practical applications. A more useful scheme involves averaging the values over the unit cell.

Now, the four nearest neighbor points each belong to two unit cells, so let’s agree to split their values between them. The four next-to-nearest neighbors – on the corners of the unit cell – each belong to four unit cells, so let’s split these four ways. A possible average value is then, schematically,

$$V^{\text{NEW}} = a \left( V_{\text{center}} + \frac{1}{2} \sum V_{NN} + \frac{1}{4} \sum V_{NNN} \right) \quad (6.2)$$

where  $V_{NN}$  and  $V_{NNN}$  are nearest neighbor and next-to-nearest neighbor points, respectively.



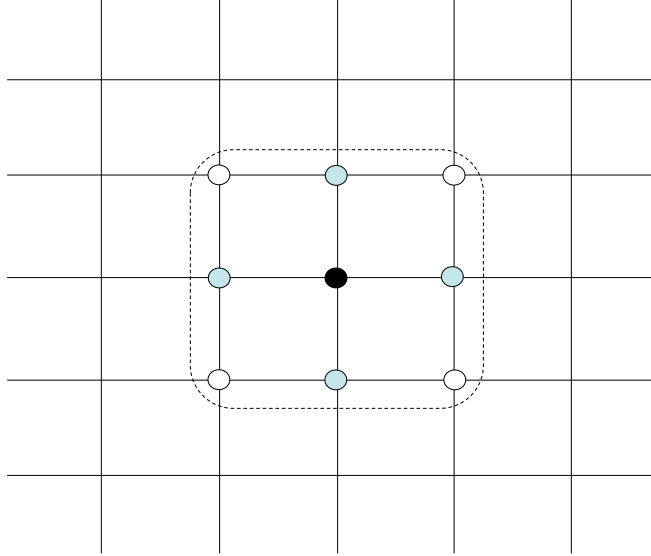


FIG. 3. Unit cell centered on the  $(i, j)$  point (black dot). Nearest-neighbor points (gray) are each shared with one other unit cell; next-to-nearest neighbors (white) are each shared with three other unit cells.

The factor  $a$  is introduced to make the overall normalization sensible. In particular, if all grid points have the *same* value, then clearly that is the value that should be produced upon restriction. That is, if  $V_{center} = V_{NN} = V_{NNN}$ , then we should find  $V^{NEW} = V_{center} = \dots$  as well. This is achieved for  $a = 1/4$ .

It may be convenient to express the final operation as a matrix showing the coefficient of each term in the weighted average:

$$\begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix} \quad (6.3)$$

Now, how shall we make use of these tools in solving our partial differential equation? There are very many possibilities and we can only sketch a few of them here.

In most cases we would define a hierarchy of grids, the finest of which represents the accuracy we would like to achieve in the final solution. It is most convenient if the same restriction and prolongation operations can be re-used at each stage; this means that each grid should half the spacing of the next coarsest one. This also indicates that the number of interior (non-boundary) points in each direction should be of the form  $2^n - 1$  where  $n$  is an integer known as the *grid level*.

Now one approach starts with the fine grid. (In a multi-level approach this would be the finest grid used.) After a round of “smoothing,” or iteration towards the solution using the Jacobi or Gauss-Seidel schemes, the solution is restricted to a coarser grid and solved (iterated) there. We then prolong back to the fine grid and continue iterating. The result is that the coarse grid iteration determines the long-distance behavior of the solution while the fine grid accounts for structure on shorter scales.

Another approach would be to start on a coarse grid, relax quickly to a (crude) solution there, then prolong to the finer grid and re-solve the problem. The idea is that the coarse solution provides a good starting point for the fine solution, or at any rate a better one than would be obtained by guessing. The coarse solution effectively transmits a rough approximation of the solution over the entire domain; the second step fills in the details. We proceed to finer and finer grids until the desired level of accuracy is reached.

This is sometimes known as the “Full Multigrid Algorithm,” although practical schemes often involve cycles of restriction and prolongation. For example, one could first cycle one level down in the grid hierarchy and back up again, then two down and back up, etc., until the final level is reached. Which specific approach is most efficient depends on the problem under consideration, and cannot in general be predicted – it must be discovered “empirically.”

## **Project Extension**

Implementing the multigrid transformations is fairly straightforward for most of the earlier projects. (The nested rectangles problem is a little tricky, due to the need to correctly map the inner boundary when prolonging or restricting.) A nice way to proceed is to design your code so that at each stage the user can choose whether to go up or down in the grid hierarchy. You can also allow the user to specify at each stage whether Gauss-Seidel iteration is carried out for a specified number of steps or until a specified accuracy is reached.

To check the performance improvement, some measure of computing time is needed. A simple way to quantify this is to just count arithmetic operations. The total computational work will be proportional to the number of iterations times the number of grid points. A running total of this quantity as you manually traverse the grid hierarchy will provide an accurate measure of speed.

Alternatively, you can define an automated multigrid approach, i.e., one that does not

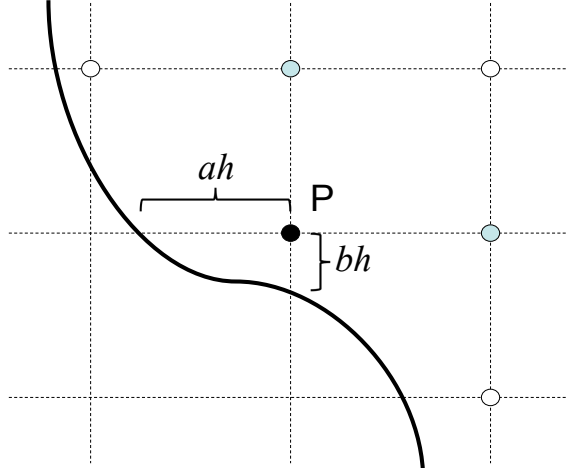


FIG. 4. Geometry for treating boundaries of arbitrary shapes.

require user intervention, and time it using the Unix `time` command or an internal timing routine. A simple automated approach that works well for these problems is to start on a very coarse grid (one interior grid point, say) and then proceed straight down to the finest grid desired. At each stage you can iterate using Gauss-Seidel until some specified tolerance is reached. Compare the overall performance of this code to both straight Gauss-Seidel and overrelaxation.

You can also experiment with algorithms that move down and back up in the grid hierarchy.

### C. Arbitrary Boundary Shapes

We have so far discussed rectangular and circular boundaries in two dimensions. These are most straightforward when writing basic programs, but of course they are also mostly amenable to analytical treatment. The application of numerical methods really comes into its own in cases with oddly shaped boundaries which would be impossible to treat analytically. In this section I sketch how to deal with such cases.

Consider a rectangular grid with an arbitrarily shaped boundary, as shown in fig. 4. The grid spacing is  $h$  as usual. Now, it is clear that for points away from the boundary the usual updating procedure applies. The difficulty comes when considering points near the boundary such as point P in the figure, for which one or more of their neighboring grid points lie on the other side of the boundary surface. How should we update such points?

The basic approach parallels the one we developed initially. In the example shown, assume that the distance from point P to the boundary along the  $x$  and  $y$  axes is  $ah$  and  $bh$ , respectively. Of course, these distances (and hence the numbers  $a$  and  $b$ ) can be computed from the known location of the boundary. Now Taylor expand around the point P:

$$V(x+h) = V(x) + h\frac{\partial V}{\partial x} + \frac{1}{2}h^2\frac{\partial^2 V}{\partial^2 x} + \dots \quad (6.4)$$

$$V(x-ah) = V(x) - ah\frac{\partial V}{\partial x} + \frac{1}{2}a^2h^2\frac{\partial^2 V}{\partial^2 x} + \dots \quad (6.5)$$

(I have suppressed the  $y$  dependence for notational simplicity.) Multiplying the first equation by  $a$  and adding the result to the second we obtain

$$aV(x+h) + V(x-ah) = (1+a)V(x) + \frac{1}{2}a(1+a)h^2\frac{\partial^2 V}{\partial^2 x} + \dots \quad (6.6)$$

This can be solved to give a difference approximation to the second derivative, just as before; the result is

$$\frac{\partial^2 V}{\partial x^2} \approx \frac{2}{h^2} \left[ \frac{V(x+h)}{1+a} + \frac{V(x-ah)}{a(1+a)} - \frac{V(x)}{a} \right]. \quad (6.7)$$

A precisely analogous result is obtained for the  $y$  derivatives, with  $a \rightarrow b$ .

You may wish to apply these results in the exploration of more complicated geometries, e.g., elliptical or other shapes.

## Appendix A: Gaussian Units in Electrodynamics

As you will probably know all too well, units are a constant source of headaches in electrodynamics. In introductory treatments it is common to adopt the SI system, with charge measured in coulombs. In this case Coulomb's law takes the familiar form

$$\vec{E} = \frac{Q}{4\pi\epsilon_0 r^2} \hat{r}, \quad (A1)$$

where  $\hat{r}$  is the unit vector pointing from charge  $Q$  to the point where the field is evaluated,  $r$  is the distance from  $Q$  to that point, and  $\epsilon_0 = 8.85 \times 10^{-12} \text{ C}^2/\text{Nm}^2$  is the permittivity of free space. This form is not especially convenient for numerical work, however, because of the constants involved have magnitudes much different than one. Thus we choose Gaussian units, which is the CGS system with the factor  $4\pi\epsilon_0$  absorbed into the definition of the

charge, giving a new unit of charge, the esu or electrostatic unit. Coulomb's law then appears as (in three dimensions)

$$\vec{E} = \frac{q}{r^2} \hat{r}. \quad (\text{A2})$$

Gauss's law now takes the form

$$\vec{\nabla} \cdot \vec{E} = 4\pi\rho, \quad (\text{A3})$$

valid in any number of dimensions and with  $\rho$  measured in esu/cm<sup>3</sup>. The electric potential  $V$  is introduced in the usual way, via

$$V_b = V_a - \int_a^b \vec{E} \cdot d\vec{l} \quad (\text{A4})$$

where  $a$  is any reference point and the integral is taken along any curve connecting  $a$  and  $b$ .

This is equivalent to

$$\vec{E} = -\vec{\nabla}V, \quad (\text{A5})$$

so that  $V$  satisfies

$$\nabla^2 V = -4\pi\rho. \quad (\text{A6})$$

The potential  $V$  is now measured in *statvolts* rather than volts.

- 
- [1] S. E. Koonin, *Computational Physics* (Benjamin/Cummings, 1985)
  - [2] T. Pang, *An Introduction to Computational Physics* (Cambridge University Press, 2006)
  - [3] L. D. Fosdick, E. R. Jessup, C. J. C. Schauble, and G. Domik, *Introduction to High-Performance Scientific Computing* (MIT Press, 1996)
  - [4] <http://www.gnuplot.info> (Free Software Foundation, 2010)
  - [5] D. J. Griffiths, *Introduction to Electrodynamics* (Benjamin/Cummings, 1999)

## GLOSSARY

antiperiodic BC	Requirement that the function on one edge of the domain be the negative of the function on the opposite edge of the domain.
BC	Boundary condition.
boundary condition	Conditions that specify the solution on the domain boundary; needed to obtain a unique solution to an elliptic PDE.
difference approximation	Technique of approximating derivatives of functions by finite differences.
Dirichlet BC	Specification of the function itself on the domain boundary.
electrostatic unit	Unit of electric charge in the Gaussian system.
elliptic PDE	A PDE of the form (in two dimensions) $Af_{xx} + Bf_{xy} + Cf_{yy} = 0$ , where the subscripts indicate partial derivatives with respect to the independent variables $x$ and $y$ , and with $B^2 - AC < 0$ . Examples include the Laplace, Poisson and Helmholtz equations.
esu	Electrostatic unit.
Gauss-Seidel method	Relaxation technique in which updated grid values are used as soon as they are produced.
Jacobi method	Relaxation technique in which new values are calculated for all grid points before any are updated. The slowest relaxation algorithm.
Laplace equation	Second-order PDE that determines the electric potential in a source-free region of space.
laplacian	The differential operator $\nabla^2 = \partial/\partial x^2 + \partial/\partial y^2$ (in two dimensions).

multigrid method	General strategy of solving a PDE on grids of several different resolution. Can give significant speedup over classical methods, including overrelaxation.
Neumann BC	Specification of the normal derivative of the function on the domain boundary
overrelaxation	Variant on relaxation in which a larger change is made at each iteration, thus speeding convergence.
partial differential equation	An equation for a function of more than one independent variable, involving partial derivatives of that function with respect to the independent variables.
PDE	Partial differential equation.
periodic BC	Requirement that the function be equal on opposite edges of the domain.
Poisson equation	Second-order PDE that determines the electric potential in a region with charges.
prolongation	Multigrid transformation for passing from a coarser to a finer grid.
relaxation	Iterative scheme for solving elliptic PDEs.
restriction	Multigrid transformation for passing from a finer to a coarser grid.
separation of variables	Technique for separating a partial differential equation into ordinary differential equations. The basic assumption is a product form for the solutions.
statvolt	Unit of electric potential in the Gaussian system.
straight injection	Restriction operation in which the central value of the unit cell is taken as the value on the coarse lattice.