Relaxation Methods for Partial Differential Equations: Instructor Notes*

David G. Robertson†

Department of Physics and Astronomy

Otterbein University, Westerville, OH 43081

(Dated: August 20, 2010)

Abstract

Some notes on the simulations are presented, along with solutions to the exercises. Please contact the author if you have any comments or suggestions.

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

†drobertson@otterbein.edu

CONTENTS

I. GOALS AND TIME NEEDED

This module is an introduction to relaxation techniques for solving elliptical partial differential equations, notably the equations of Laplace and Poisson, with application to electrodynamics. Its principal goals are:

- to provide an opportunity to develop insight into electrostatics; and
- to familiarize students with the basic concepts involved in the numerical solution of partial differential equations.

In my experience, working through these kinds of exercises is a great way to develop intuition about how electrodynamics works. The need to translate formulas from vector calculus into concrete expressions that can be evaluated by a computer is an excellent way to learn what these formulas actually mean. The ability to produce solutions to Laplace's equation for situations that would be difficult or impossible to analyze analytically helps develop intuition about how electric phenomena operate.

The minimal physics background required includes electrostatics at the level of a typical junior-level course. Students should ideally have studied Laplace's equation and the analytical approaches to its solution, as well as, e.g., the relation between surface charge density and normal component of electric field. It is conceivable that students who have taken only calculus-based introductory physics could also benefit from this module, with appropriate preparation. Vector calculus is probably essential.

There is considerable flexibility in which parts of this module that you use. The most interesting and instructive problems are those relating to lines of charge and the capacitance

of concentric rectangles. But there is a lot of interesting computational physics just in generating solutions (with or without point charges) in a rectangular domain, calculating the electric field, etc. For those with more time, advanced topics on overrelaxation and the multigrid approach can be explored.

My estimate is that one to two weeks of class time, assuming 3-4 hours per week in class and some work outside of it, should be sufficient to explore the basic numerical issues and develop codes to solve Laplace's equation in several geometries, including determining the capacitance of nested rectangles. If the instructor wishes to pursue some of the elaborations, in particular the multigrid approach, then an additional week or so will probably be needed.

II. SOLUTIONS TO EXERCISES

1. Eq. (4.3) can be rearranged to give

$$V(x+h) - V(x) = h\frac{dV}{dx} + \mathcal{O}(h^2). \tag{2.1}$$

Hence

$$\frac{dV}{dx} = \frac{V(x+h) - V(x)}{h} + \mathcal{O}(h), \tag{2.2}$$

showing that the error made in the "asymmetric" difference formula is $\mathcal{O}(h)$. This means that to cut the error in half, we must reduce h by a factor of two.

The Taylor expansion also implies

$$V(x+h) - V(x-h) = 2h\frac{dV}{dx} + \mathcal{O}(h^3), \tag{2.3}$$

since all terms with even powers of h cancel in the difference. Hence

$$\frac{dV}{dx} = \frac{V(x+h) - V(x-h)}{2h} + \mathcal{O}(h^2), \tag{2.4}$$

and the symmetric difference formula is accurate to $\mathcal{O}(h^2)$. In this case, to cut the error in half we must reduce h by a factor of only $\sqrt{2}$.

2. The solution follows the development given in the text. Expressed in polar coordinates the laplacian takes the form

$$\nabla^2 f = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial f}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 f}{\partial \theta^2} \tag{2.5}$$

Difference expressions for the various first and second derivatives can be obtained as before by Taylor expansion:

$$f(r+\Delta r,\theta) = f(r,\theta) + \Delta r \frac{\partial f}{\partial r} + \frac{1}{2}\Delta r^2 \frac{\partial^2 f}{\partial r^2} + \cdots \tag{2.6}$$

$$f(r,\theta+\Delta\theta) = f(r,\theta) + \Delta\theta \frac{\partial f}{\partial \theta} + \frac{1}{2}\Delta\theta^2 \frac{\partial^2 f}{\partial \theta^2} + \cdots \tag{2.7}$$

The final result is

$$\nabla^2 f = \frac{1}{r}\left[\frac{f(r+\Delta r,\theta) - f(r-\Delta r,\theta)}{2\Delta r}\right] + \frac{f(r+\Delta r,\theta) + f(r-\Delta r,\theta) - 2f(r,\theta)}{\Delta r^2} + \frac{1}{r^2}\left[\frac{f(r,\theta+\Delta\theta) + f(r,\theta-\Delta\theta) - 2f(r,\theta)}{\Delta\theta^2}\right] + \cdots \tag{2.8}$$

If desired, this can be solved for $f(r,\theta)$ to give an iteration rule in terms of neighboring points.

3. In integral form Gauss's law reads

$$\oint \vec{\mathrm{E}} \cdot d\vec{A} = 4\pi Q_{encl}$$

(Gaussian units). In two dimensions the surface integral is taken over a line enclosing the charge Q_{encl}. For a point charge take as a gaussian surface a circle of radius r centered on the charge. Then the usual symmetry arguments apply and the surface integral reduces to

$$\oint \vec{\mathrm{E}} \cdot d\vec{A} = |\vec{\mathrm{E}}| \cdot 2\pi r$$

(assuming $q > 0$ so the electric field is directed radially outward). Setting this equal to $4\pi q$ gives the desired result

$$|\vec{\mathrm{E}}| = \frac{2q}{r}. \tag{2.9}$$

Note that this is identical to the field of an infinite line of charge in three dimensions; in this case one takes a gaussian cylinder (radius r and length l) and obtains

$$\oint \vec{\mathrm{E}} \cdot d\vec{A} = |\vec{\mathrm{E}}| \cdot 2\pi r l = 4\pi\lambda l, \tag{2.10}$$

where λ is the (positive) charge per unit length. Hence

$$|\vec{\mathrm{E}}| = \frac{2\lambda}{r}. \tag{2.11}$$

The potential is now given by

$$V(r) - V(r_0) = -\int_{r_0}^{r} \vec{\mathrm{E}} \cdot d\vec{l} = 2q \ln(r_0/r), \tag{2.12}$$

where we set $V(r_0) = 0$. Note that in this case we cannot take the reference point $r_0 \to \infty$, since the logarithm becomes ill-defined. Of course, in any question involving potential energy we will encounter a *difference* of such logarithms, in which r_0 will properly cancel.

4. Apply Gauss's law in integral form to the surface of the conductor. By symmetry the electric field is normal to the conductor surface; hence the surface integral reduces to

$$\oint \vec{\mathrm{E}} \cdot d\vec{A} = 2|\vec{\mathrm{E}}_n| l, \tag{2.13}$$

where l is the length of the gaussian curve. Gauss's law then implies

$$|\vec{\mathrm{E}}_n| l = 4\pi\lambda l, \tag{2.14}$$

where λ is the charge per unit length on the conductor surface. Hence

$$|\vec{\mathrm{E}}_n| = 4\pi\lambda. \tag{2.15}$$

III. NOTES ON THE SIMULATION PROJECTS

In this section I collect some notes and suggestions for the simulation projects. I have deliberately left the student manual fairly general, placing most of the hints here. That way you can choose what and when to reveal to the students.

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

   This project will get the students through many of the basic issues and problems of the module.

   If working in a programming language like C, C++ or Fortran, the program design is relatively straightforward. There should be a two-dimensional array to hold the potential values on the grid, including the boundary surfaces where the potential (or its derivative) is specified. The sweep through the grid only passes over "interior" points, of course. A good way to test for convergence is to keep a record of the largest

fractional change in value in one iteration step; when this drops below a specified threshold the code stops and prints the result.

Sample programs (written in C) are provided. Programs `jacobi.c` and `gauss.c` implement the basic schemes with fixed boundary conditions, while `gauss-pbc.c` uses periodic boundary conditions. The program `overrelax.c` implements over-relaxation.

It is possible to perform these calculations in Excel, although the size of the grid and the number of iterations will be rather limited. It may be instructive, however, since the student can easily see what is happening in each iteration. Thus even if they will eventually implement the calculations in a standard programming language, it may be worth some time to examine the approach in Excel.

Note that some of the algorithms, e.g., Gauss-Seidel and multigrid, will be quite difficult to implement in Excel. The Jacobi method is fairly straightforward, however, and it can easily be extended to incorporate overrelaxation. If you are limited to using Excel then this would be a good exercise showing at least one technique for speeding things up.

A sample Excel file implementing the Jacobi method with overrelaxation on a 20×20 grid is included. Basically it lays out a grid of starting values, with the boundary values set by the user and the interior points set equal to the average of the boundary values. One then sets up a second grid below the first, with the boundary cells referencing the initial boundary points and the interior cells set to calculate the average of nearest neighbors in the initial grid. One can then copy and paste this grid structure below the previous one, with the copies referencing the grids just above. Be sure to paste the grids in the same relative position so that the references are propagated correctly. As the iteration proceeds the solution will "fill in" from the edges and then settle down to a steady state. Achieving high accuracy in the solution will be difficult since going past about 20 or 30 iterations may become tedious, but the general trend should be discernible.

The best way I have found to evaluate the change in the solution from one iteration to the next us using the Excel function SUMXMY2, which can compute the sum of $(V_{i,j}^{n+1} - V_{i,j}^{n})^2$ over the lattice; when divided by the number of lattice points this gives a measure of the average change in the iteration.

Speeds can be measured on a Unix system using the `time` command, or an internal timing routine. Alternatively, one can simply count the number of calculations done in the relaxation; this is proportional to the number of iterations times the number of (interior) grid points. This will be the only option if Excel is used for the calculations. Students should find that Gauss-Seidel is indeed faster than Jacobi by roughly a factor of two, i.e., takes about a factor of two fewer total calculations.

The initial guess can also have a noticeable impact on the rate of convergence. (Of course, if one guessed the exact solution the relaxation would converge in a single iteration!) For example, in the sample code the the effect of setting interior points to the average of the boundary values, compared with setting them to zero, also amount to about a factor of two in speed. The variation in final values obtained from different starting points should be comparable to the overall tolerance specified.

Contour plots will be the best way to examine the results of the calculation. If you are using `gnuplot`, you can try this set of commands:

```
set contour both; set cntrparam levels 30
unset clabel
unset surface
set pm3d map
set size ratio -1
splot "outfile" with line palette notitle
```

Here `outfile` is a text file containing potential data. This can simply be the potential values printed one per line as you sweep through the grid by rows. Data for successive rows should be separated by a blank line in the file. Alternatively, you can print data in the form x, y, $V(x, y)$.

In Excel, use the chart tool and look under "Surface." It is helpful to first make a true surface plot, i.e., a 3d plot where the height of the surface is given by the potential value. This allows you to set the number of increments in the z direction. This is how you change the number of contour levels, i.e., the spacing between data values for which different contours are drawn. Once the z increment is set, switch to a contour plot to see the result.

2. **Electric Field**

   This is a rather straightforward exercise in computing the gradient of the potential. The derivatives should be computed using one of the formulas given in the text. Students should verify that the field is perpendicular to the boundary surfaces.

   Examples of these calculations are given in several of the sample codes, notably `sheets-pbc.c` (see the commented statements near the end).

3. **Adding Charges**

   Specification of charge strengths and locations can be handled in may ways; for example, the sample code allows users to specify a strength and x and y values for any number of point charges as command line arguments. The main difference here is that we include the $4\pi\rho_{i,j}$ term in computing the updates to V.

   Students can now verify that the potential of a single point charge falls off logarithmically, i.e., if $r \to 2r$ then the potential decreases by a factor $\ln 2$ (for a positive charge). Dipoles and other charge configurations can be studied. The potential contours will look qualitatively similar to those in three dimensions, although the detailed behavior is different.

   The square boundary will introduce some error unless the correct logarithmically-varying boundary values are computed. In practical applications one does not have the luxury of doing this since the potential is not known beforehand, so it is better to think about the effect of the boundary and how it can be minimized. The obvious idea is to move it farther away. At least for points close to the central charge, a distant boundary should have minimal effect. Students can study how this effect diminishes with distance, especially since they can compute directly the exact result (for a single point charge) for comparison.

   If the discrete laplacian in polar coordinates has been developed, then these can be employed with a circular boundary. In this case it is clear from symmetry that the potential depends only on r, hence the potential has a fixed value on the boundary. Since we are free to add a constant to the potential this can be set to any value desired, even zero. In this case the circular symmetry will be realized exactly and the expected fall-off of $V \propto 1/\ln r$ will be clear. (Indeed in this case nothing depends on the polar

angle θ so the problem is really one-dimensional. The grid can be made one unit in size in the angular direction, and the relations

$$V(r, \theta \pm \Delta\theta) = V(r, \theta)$$

assumed in the updates.)

The boundary effect should fall off faster for a dipole, since there is a screening effect and the long-range field falls off as $1/r^2$ (potential as $1/r$).

If there is time and interest, students can experiment with *periodic* boundary conditions as an alternative to direct specification on the boundary. This is physically equivalent to having an infinite array of neighboring "cells" with the same charge configuration. Those distant images do contribute to the potential but they do so in a way that is smoother than just specifying a fixed value on a rectangular boundary – they allow the potential to at least vary over the boundary surfaces in something like the correct way.

Most of the sample programs allow point charges to be sprinkled across the domain if desired.

4. **Parallel Sheets of Charge**

This problem is a paradigm for capacitance calculations.

The easiest way to proceed is to use two edges of the domain as the "plates," setting them to some specified potential (not the same!). Use periodic boundary conditions in the direction along the plates; this has the effect of making the problem translationally invariant in that direction, equivalent to infinite plates. This again becomes in effect a one-dimensional problem.

To determine the capacitance of the plates you will need to compute the charge on each plate. This can be determined from the normal component of the electric field at the surface, as discussed in the text. This gives a charge density that can be used to calculate the capacitance per unit area. The standard result should of course be recovered (in Gaussian units!): $C/A = 1/4\pi d$. The sample program `sheets.c` implements this calculation.

Perhaps more instructive is to introduce actual sheets (lines) of equal and opposite charge, not on the boundaries, and solve Poisson's equation both between and outside the plates. In this case one faces some interesting issues associated with boundary conditions.

The simplest approach is to stretch the lines of charge across the domain and impose periodic boundary conditions in the direction along the sheets; this is then equivalent to infinite lines of charge. In this case we expect a constant electric field (linear potential) between the plates, and zero field (constant potential) outside. What boundary conditions should we impose in the direction perpendicular to the plates? It is clear that a constant value should be chosen, but the potential is not the same above and below the plates.

To see how to proceed, note that given one solution to Laplace's (or Poisson's) equation we can obtain another by adding a constant to V, that is, by adding the same constant to all the $V_{i,j}$.

This just reflects that the physical object is the electric field, which is the gradient of V; hence adding a constant to V does not change $\vec{\mathrm{E}}$. In the discrete approximation it is easy to see that if $V_{i,j}$ is a solution, meaning that upon iteration we recover the same values for $V_{i,j}$, then $V_{i,j} + c$ will also be a solution, i.e., will be reproduced upon iteration.

Now we can use this freedom to assume that the potential is defined so that V above the plates is equal and opposite to V below the plates. If this were not the case, we could just add to V whatever constant was needed to make it so. We can enforce this in the simulation by imposing *anti-periodic* boundary conditions in the direction perpendicular to the sheets. That is we require

$$V_{bottom} = -V_{top}.$$

This leads straightforwardly to the correct results. See `sheets-pbc.c` for an example calculation.

If one wishes to study a single sheet, then some additional thought is required. In this case the potential is not constant anywhere (the electric field is constant, so the potential is increasing or decreasing linearly with distance on either side of the sheet).

Now if we place the sheet in the middle of the domain, then by symmetry we will have

$$V_{bottom} = V_{top}.$$

To simulate a single slab, then, place it at the center and impose periodic boundary conditions in the perpendicular direction.

Finite slabs are also well worth studying. If they have equal and opposite charges and are centered in the domain, then by symmetry we again expect $V_{top} = -V_{bottom}$ (though these will no longer be constant along the edges of the domain). So we again use anti-periodic boundary conditions in this direction. In the direction along the slabs, one can try both imposing a fixed value (the same on both sides if the plates are centered) and periodic boundary conditions. Since the true potential is not constant on these surfaces, the constant value will introduce some distortion into the solution. This can be reduced, however, by moving the boundaries farther away from the plates.

Alternatively, periodic boundary conditions allow the potential to vary along the boundary surfaces, and so permit a closer realization of the true solution. They effectively place an "image" of the plates in a neighboring domain (on each side), which contribute to the potential in the region of interest. Again, however, if the boundary surfaces are moved farther away the effect will diminish. (For sufficiently large distances the potential should fall off as $1/d$, since the plate configuration has zero monopole moment.) Students should find the periodic conditions work best here. The program `sheets-finite.c` allows both options to be explored.

In all these cases, the potential difference between the sheets can be read off from the solution to Poisson's equation and the capacitance per unit area determined in the usual way. Since this system is equivalent to the textbook infinite parallel plate capacitor in three dimensions, the standard result should be recovered (in Gaussian units!): $C/A = 1/4\pi d$. For finite plates this will be reproduced only approximately, of course, since the finite capacitor will not have a uniform charge distribution on the plates.

A final point about charge sheets. In the textbook treatment the field depends on the charge per unit area, for infinite sheets. This area is not the area covered by our domain, however! Our domain has one dimension along the sheet – together with the

direction "out of the page" this is the plane of the sheet. Hence the charge density in our simulation, times the grid spacing h, gives the charge per unit length along the sheet, or per unit area if one is considering the three-dimensional problem.

5. **Concentric Rectangles**

   The general approach here is the same as before. The outer rectangle should be the boundary of the domain. The inner and outer rectangles are held at specified potentials, so the potential difference between them is known. Then use relaxation to determine V in the region between them. From this the normal electric field can be computed, giving the charge density on each surface. This will not be constant, so the total charge will have to be obtained by adding the contributions from each small segment of the surface:

   $$A = \int \sigma dA.$$

   As a check, the surfaces should have equal and opposite charges. (A tricky point is to make sure the normal electric field has the correct sign.) Finally, the capacitance of the system is determined as $C = Q/\Delta V$. I find a result of about 0.0052 for a square capacitor with the inner boundary one third the size of the outer. The sample code `rectcap.c` implements this calculation.

IV. OVERVIEW OF SAMPLE PROGRAMS

A set of sample programs, written in C, is distributed with the module. Typically they take some command line arguments, with other parameters hard-wired via `#define` statements. Comments in the programs explain the arguments.

1. `jacobi.c`

   Implements the basic Jacobi algorithm for a rectangular domain, with fixed boundary conditions. Charges may be added via command line arguments.

2. `gauss.c`

   Implements the Gauss-Seidel method for a rectangular domain, with fixed boundary conditions. Charges may be added via command line arguments.

3. `gauss-pbc.c`

   Implements Gauss-Seidel in a rectangular domain with periodic boundary conditions. Charges may be added via command line arguments.

4. `overrelax.c`

   Implements Gauss-Seidel with overrelaxation, as described in section IV.A of the student manual. The overrelaxation parameter ω is specified on the command line, as are the strengths and locations of any charges.

5. `sheets.c`

   Gauss-Seidel relaxation for parallel "sheets" of charge, modeled as two sides of the domain boundary (the horizontal sides). These two sides are held at constant potential, with periodic boundary conditions imposed in the other direction. Calculation of the induced charge and capacitance are also performed.

6. `sheets-pbc.c`

   Gauss-Seidel relaxation for two sheets of charge specified as actual lines of charge in the domain. In this version the sheets traverse the entire domain in the x direction, with periodic boundary conditions. The y location of the sheets, and the charge per unit area on each, are specified via command line arguments. Anti-periodic BCs are used in the y direction, as discussed above, although the user can also experiment with fixed boundary conditions.

7. `sheets-finite.c`

   Gauss-Seidel for finite sheets, modeled as lines of charge (parallel to the x axis) that do not stretch across the domain. Periodic BCs are used in x and anti-periodic BCs in y; the sheets should be centered in the domain to produce the correct symmetries.

8. `rectcap.c`

   Solution of Laplace/Poisson in a rectangular domain, with calculation of the capacitance. Point charges may be added between the rectangles via command line arguments.

9. `multigrid.c`

   Implements the multigrid transformations described in sect. IV.B. At each stage the user controls the location in the grid hierarchy as well as the relaxation criteria (i.e., whether to relax for a specified number of iterations or to a specified tolerance).

10. `multigrid-auto.c`

    Implements an automated multigrid approach similar to the "Full Multigrid Algorithm" described in the text. The calculation starts at the coarsest level and proceeds to finer and finer grids, halving the grid spacing and relaxing to the specified tolerance at each stage. When relaxation is complete on the finest grid, the simulation stops.