Computational Physics With Python

Dr. Eric Ayars

California State University, Chico

Chapter 8

Partial Differential Equations

8.0 Laplace's Equation

Knowing the electric potential in an area of space allows us to calculate the electric field in that area, and electric field tells us what we need to know about forces on charged particles in that area.

If we know where all the charges are in an area, we can calculate the potential from

$$V(\vec{r}) = k \sum_{i=1}^{n} \frac{q_i}{\vec{r_i} - \vec{r}} \, . \tag{8.1}$$

The problem with equation 8.1 is that we rarely know the exact charge distribution in an area. We're far more likely to know the *potentials* at certain points in the area, which is quite different from knowing the *charges* in that area.

Laplace's equation describes the potential in a charge-free region. That "charge-free" region can include all the areas between the charges in any area we want, so we can use it to get beyond the problems in equation 8.1. The derivation of Laplace's equation is good material for a different course than this one, so let's just start with the equation itself:

$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} + \frac{\partial^2 V}{\partial z^2} = 0 \tag{8.2}$$

We'll start in the usual way, by breaking our space into a discrete grid of points. Then we'll develop a simplified expression of Laplace's equation, and use that to solve for the potential V.

The simplest way of calculating the derivative uses equation 2.13:

$$\frac{\partial V}{\partial x} \approx \frac{V(x_{i+1}) - V(x_i)}{\Delta x} \tag{8.3}$$

Let's simplify our notation a bit before we go further: we'll define $V_i \equiv V(x_i)$, and so on. So $V_{i+1,j,k}$ would refer to the potential at the point (x_{i+1}, y_j, z_k). Getting back to equation 8.3, this is actually the derivative at the point halfway between x_i and x_{i+1}. Let's call that point $x_{i+\frac{1}{2}}$. We could also use equation 8.3 to find $\frac{\partial V}{\partial x}$ at $x_{i-\frac{1}{2}}$:

$$\frac{\partial V}{\partial x} \approx \frac{V_i - V_{i-1}}{\Delta x} \tag{8.4}$$

Of course, for equation 8.2 we actually need the second derivative. We need this second derivative at x_i, so we use the difference in the first derivative at $x_{i+\frac{1}{2}}$ and $x_{i-\frac{1}{2}}$ in equation 2.13.

$$\frac{\partial^2 V}{\partial x^2} \approx \frac{V_{i+\frac{1}{2}} - V_{i-\frac{1}{2}}}{\Delta x^2} = \frac{V_{i+1} + V_{i-1} - 2V_i}{\Delta x^2} \tag{8.5}$$

As it turns out, equation 8.5 is better than we deserve from the method we used to calculate it. It's exactly the same as equation 2.21, which is good to third order in Δx. It's also better than we *need*, since we're never going to actually calculate that derivative!

Let's go back to equation 8.2 and put in our results from equation 8.5. In three dimensions, we get

$$V_{i,j,k} \approx \frac{1}{6}\left[V_{i+1,j,k} + V_{i-1,j,k} + V_{i,j+1,k} + V_{i,j-1,k} + V_{i,j,k+1} + V_{i,j,k-1}\right] \tag{8.6}$$

And here's the good part: equation 8.6 tells us that the solution to Laplace's equation is such that V at each point is just the *average* of the potential at the surrounding points. This has one very important implication in electrostatics: it means that there are no local minima or maxima of V in charge-free regions.[1]

Relaxation Method

The "average" nature of the solution to equation 8.2 suggests the *relaxation method* as a way of solving for the potential in charge-free regions.

[1] For further discussion of this marvelous result, see [5].

1 Populate your solution grid with some initial guess of what the solution should look like. Anything works — zero is fine — but if you have even a coarse estimate of the solution you can save quite a bit of calculation time by starting with that.

2 Go through your grid of (x, y, z) points, recalculating each point to be the average of the surrounding points.

3 As you go, keep track of how much things change. When the largest change is less than your solution tolerance, stop.

Of course there are some subtle —or not-so-subtle— points to keep track of when using the relaxation method. First, you need to make sure that the area of space in which your calculations take place is actually *bounded.* In other words, make sure that the boundary of the area you're relaxing is set to some fixed potential. Zero usually works well, and the easiest way of ensuring this is to initially set your guess to something that has zero on all the boundaries, then not re-calculating the boundary elements.

Second, your solutions are going to be very boring unless there are some fixed potential points inside the area. Don't recalculate the potential at those points! They're supposed to be fixed.

A plot of the potential near a "parallel-plate capacitor" on a 100×100 grid is shown in figure 8.0. Note that this is the solution for V in a 2-D region: in 3-D this is the equivalent to the potental around two infinitely long parallel plates.

Over-Relaxation

We can speed the relaxation method considerably by using "over-relaxation" [4]. At each point on the grid, calculate the new value of V as before, but instead of assigning the new value to the grid point calculate the *change* in V: $\Delta V = V_{new} - V_{old}$. Multiply this ΔV by some factor α, and add the result to the original to get the new value of V at that point. In equation form,

$$V_{new} = \alpha \Delta V + V_{old} \tag{8.7}$$

The reason this method is faster is that V moves farther each step. It may even move past the "correct" value, but then it will just move back the next step. Overall, the solution converges faster.

The factor α should be between 1 and 2. $\alpha = 1$ corresponds to the regular relaxation algorithm, so there's no benefit. Values of α greater than 2 cause the algorithm to become unstable: V will oscillate from step to step

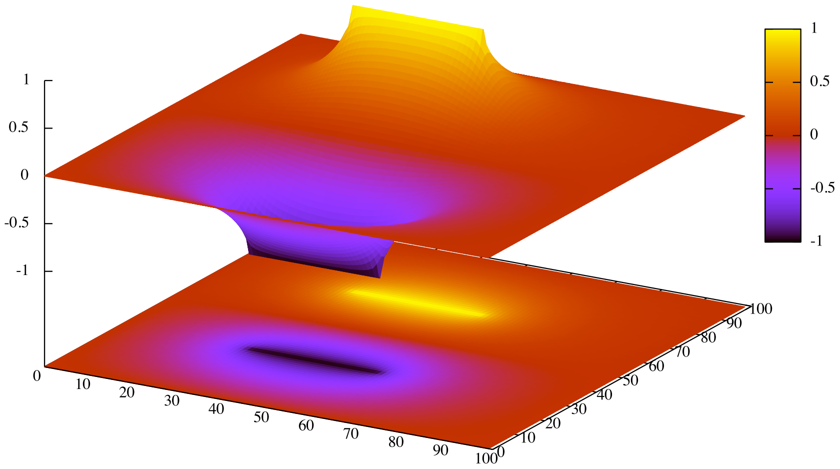

Figure 8.0: 2-D relaxation solution for V in the region surrounding two parallel plates held at opposite potentials. Both top and bottom show the same information in different ways.

without converging on the solution. The optimal value of α depends on the details of the problem, but values between 1.2 and 1.5 seem to work well most of the time.

8.1 Wave Equation

For our next partial differential equation, let's consider the wave equation,

$$\frac{\partial^2 y}{\partial t^2} = c^2 \frac{\partial^2 y}{\partial x^2} \ . \tag{8.8}$$

We can derive this equation to some degree of plausibility by starting with a string, as shown in figure 8.1. As you might expect by now, we break the string into segments of length Δx and label each segment with an index i. The mass per length of the string is μ, so the mass of each string segment

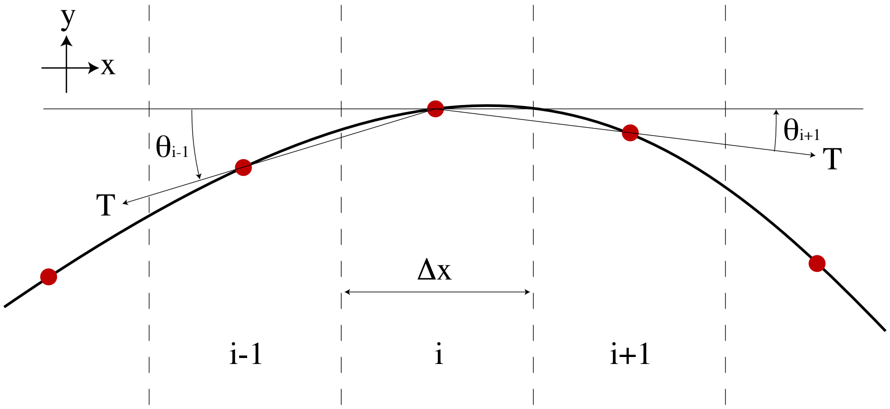

Figure 8.1: A bit of string for deriving the wave equation. The vertical scale is greatly exaggerated for clarity.

is $\mu\Delta x$. Next, bring in Newton's second law, $F = ma$, to find the vertical acceleration $\frac{d^2y}{dt^2}$ of segment i. The force on the string segment is just the sum of the vertical components of tension on each side, since the horizontal components of T must necessarily cancel out. Taking the angles θ to be positive in the directions shown, Newton's second law becomes

$$\mu\Delta x\frac{d^2y}{dt^2} = T\sin\theta_{i+1} - T\sin\theta_{i-1}\ . \tag{8.9}$$

For small angles, the distance between points is approximately Δx, so

$$\sin\theta_{i+1} \approx \frac{y_{i+1} - y_i}{\Delta x} \tag{8.10}$$

and

$$\sin\theta_{i-1} \approx \frac{y_i - y_{i-1}}{\Delta x}\ . \tag{8.11}$$

Substituting equations 8.10 and 8.11 into equation 8.9 and rearranging slightly, we obtain

$$\frac{d^2y}{dt^2} \approx \left(\frac{T}{\mu}\right)\frac{y_{i+1} + y_{i-1} - 2y_i}{\Delta x^2}\ . \tag{8.12}$$

But the right-hand side of equation 8.12 is just the second derivative of y with respect to x (See equation 8.5), so we now have the discrete-particle equivalent of equation 8.8, with $c = \sqrt{\frac{T}{\mu}}$.

For a numeric solution to the wave equation, we apply our numeric second derivative equation to each side of equation 8.8. Since we're doing second derivatives with respect to both time and space, we need to make sure the notation is clear. We'll use i to indicate steps in space, and n to indicate steps in time, so

$$y_{i,n} \equiv y(i\Delta x, n\Delta t)$$

and then equation 8.8 becomes

$$\frac{y_{i,n+1} + y_{i,n-1} - 2y_{i,n}}{\Delta t^2} = c^2 \left[\frac{y_{i+1,n} + y_{i-1,n} - 2y_{i,n}}{\Delta x^2} \right] . \tag{8.13}$$

Solve equation 8.13 for $y_{i,n+1}$ to obtain

$$y_{i,n+1} = 2\left[1 - r^2\right] y_{i,n} - y_{i,n-1} + r^2 \left[y_{i+1,n} + y_{i-1,n}\right] \tag{8.14}$$

where $r \equiv \frac{c\Delta t}{\Delta x}$.

Let's stop and think what equation 8.14 tells us. Remember that i refers to position and n refers to time; so it's saying that the position of the string for *this* location at the *next* time step ($y_{i,n+1}$) depends on where the string segment is now ($y_{i,n}$), where the segment used to be ($y_{i,n-1}$), and where the segments on each side are at present ($y_{i+1,n}, y_{i-1,n}$).

To use equation 8.14, we need two starting configurations. We need to know the y value of the string at each i for n, and the y value for each i one time-step in the past, at $n - 1$. The easiest way of doing this is to start with the string at some non-equilibrium position, and not moving. Make two arrays of y values: y_{old} and y_{now}. Since the string is not moving, their values should be the same. This produces a wave in a manner equivalent to plucking a guitar string: the string is pulled to one side, held briefly, and released.

Another way of generating the required two starting configurations is to create two slightly different arrays of y values. A simple example would be to create y_{old} with an equilibrium configuration (zero throughout) and then create y_{new} to be also zero but with one element just slightly positive. This produces a wave in a manner similar to striking a piano string with a felt hammer.

8.2 Schrödinger's Equation

Quantum theory is one area that seems to require computers for further progress. The time-independent Schrödinger's equation,

$$-\frac{\hbar^2}{2m}\nabla^2\psi + V(\mathbf{r})\psi = -i\hbar\frac{\partial}{\partial t}\psi \tag{8.15}$$

describes the wavefunction ψ for a quantum particle. This wavefunction can then be used to calculate the probability of finding the particle in a given location as well as the position, velocity, momentum, and so on.[2] We can solve Schrödinger's equation analytically for simple cases: particles in boxes, harmonic oscillators, and hydrogen atoms are examples. But beyond that, we're pretty much stuck.

We're not going to get unstuck in part of one chapter of a book on computational physics, either! This is a complicated topic. But we can get a short survey of one useful technique if we limit ourselves to the time-independent Schrödinger's equation, and to bound particles in one dimension.

$$-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}\psi + V(x)\psi = E\psi \tag{8.16}$$

This looks straightforward enough. $V(r)$ is the potential well in which the particle is bound, $\hbar$ and m are known, and E is just some constant. There are conditions on ψ, though: ψ must be continuous, and if V is not infinite then the derivative of ψ must also be continuous. There is also a normalization condition: the probability of finding the particle *somewhere* is exactly one, so

$$\int_{-\inf}^{\inf} \psi * \psi \; dx = 1 \tag{8.17}$$

The fascinating thing about solutions to this equation — the *quantum* thing, if you will — is that these conditions result in valid solutions only for certain values of E. The usual way of showing this is to work through an actual solution[12] but this is a computational class, so let's just take a brute-force approach and see what happens.

Imagine a square potential well, symmetric about the origin, with width $2L$. This well contains a single particle. Since the particle is bound by the potential, we know that the particle energy $E < V$ for $-L < x < L$. By symmetry, we can see that there are two possible classes of solution to Schrödinger's equation for this configuration: symmetric and anti-symmetric

[2] Take a Quantum class for more details, if you haven't already.

solutions. From the normalization condition, we can see that ψ must go to zero for large x. Two possible solutions that meet these criteria are shown in figure 8.2.

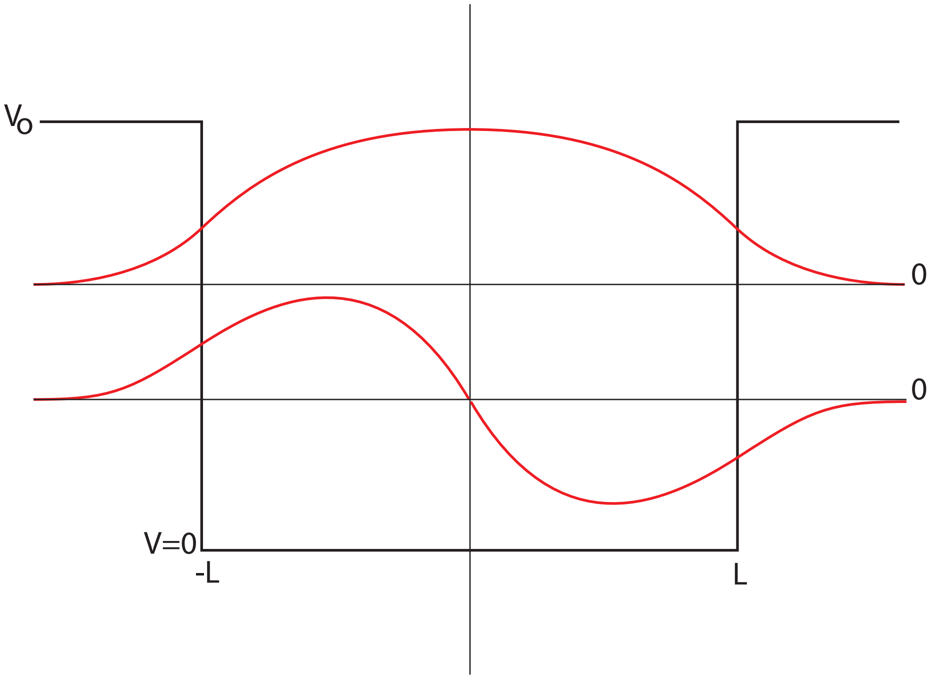

Figure 8.2: Two possible solutions to Schrödinger's equation, overlaid on a square potential well. The upper solution is symmetric, the lower is anti-symmetric. The two are on different axes: don't interpret the relative vertical positions of the graphs as meaning anything!

What we want to do is find the value(s) of E that give valid solutions to ψ. We'll do this by starting with a guess of the value of E. We'll take this guess and use our ODE-solving techniques to find ψ, starting from the center and working out to positive x values. We will know the solution is a good one if $\psi \to 0$ at large values of x, and we will adjust E until we find a good solution.

For initial conditions in the symmetric case, we will start with $\psi = 1$ and $\frac{d\psi}{dx} = 0$. This value of ψ is almost certainly incorrect, but we can scale it to its correct value later by using equation 8.17. For the anti-symmetric case, the appropriate initial conditions are $\psi = 0$ and $\frac{d\psi}{dx} = 1$. Again, that value of $\frac{d\psi}{dx}$ is almost certainly incorrect, but we can normalize ψ later to correct it.

Equation 8.16 is a second-order ODE. We can simplify it further without loss of generality by redefining our system of units so that $\hbar = 1$ and $m = 1$:

$$\ddot{\psi} = 2 * (V(x) - E)\psi \tag{8.18}$$

For the finite square well in figure 8.2, it's easy to define the potential $V(x)$:

```
def V(x):
    """
    Potential in which the particle exists.
    L = 1
    """
    if x<1.0:
        return 0            # square well
    else:
        return Vo
```

We can then define the ODE for solution with odeint():

```
def SE(y,x):
    """
    Returns derivs for the 1-D TISE, for use in odeint.
    Requires global value E to be set elsewhere.
    Note that we are using x as time here... Python doesn't care!
    """
    g0 = y[1]
    g1 = 2.0*(V(x)-E)*y[0]
    return array([g0,g1])
```

And we can plot the resulting wavefunction ψ. If our value of E is too large or too small, ψ diverges quickly, as shown in figure 8.3. This method is called the "shooting method", for reasons which are fairly obvious.

We can automate the process of finding the correct value of E, of course. In section 2.0 we saw several ways of finding a zero of a function: this is no different except that the "function" for which we want a zero is just something that returns the last value of ψ in our odeint solution. Sample code for this, using the brentq() root-finding function, is shown below.

```
#!/usr/bin/env python

"""
SEsolve.py
```

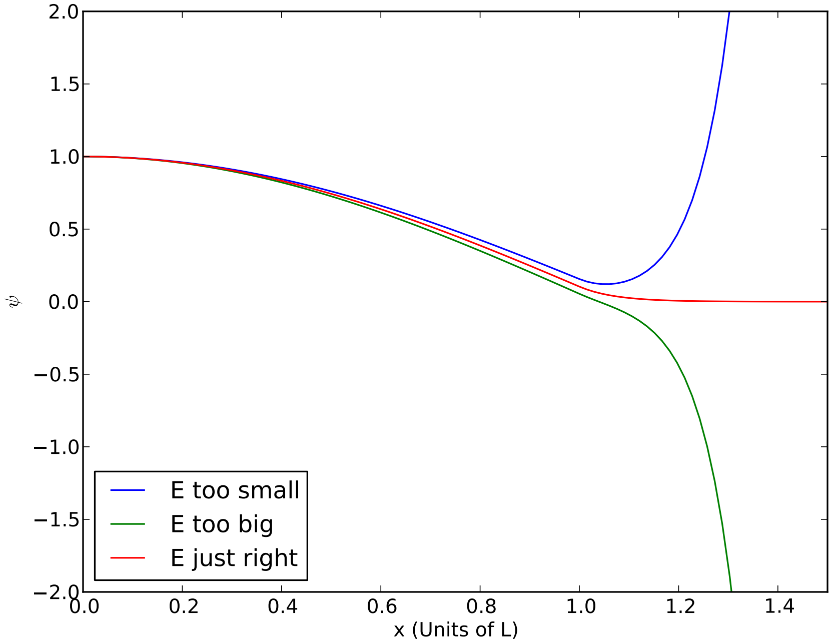

Figure 8.3: ψ diverges quickly if E is slightly off!

```
Uses shooting method to find even-parity solutions to the finite square well
Takes two guesses of energy from the command line, finds a bracketed energy
eigenvalue E, and plots the corresponding wavefunction.

hbar, m, and L are all taken as 1.

The parameter b is the point in units of L at which to check if the
wavefunction diverges. b must be greater than 1, of course.

Solutions are not normalized.
”””

from pylab import *
from scipy.integrate import odeint
from scipy.optimize import brentq
import sys
```

```
b = 2.0
Vo = 20.0          # Potential outside square well
steps = 100
E = 0.0            # global variable, changed by Final_Value()

def V(x):
    """
    Potential in which the particle exists.
    L = 1
    """
    if x<1.0:
        return 0            # square well
    else:
        return Vo


def SE(y,x):
    """
    Returns derivs for the 1-D TISE, for use in odeint.
    Requires global value E to be set elsewhere.
    Note that we are using x as time here... Python doesn't care!
    """
    g0 = y[1]
    g1 = -2.0*(E-V(x))*y[0]
    return array([g0,g1])

def Final_Value(energy):
    """
    Calculates psi for this value of E, and
    returns the value of psi at point b to check divergence.
    """
    global y
    global E
    E = energy
    y = odeint(SE,yo,x)
    return y[-1, 0]             # return final value (psi at b)

y = zeros([steps,2])
yo = array([1.0, 0.0])          # initial psi and psi-dot.
x = linspace(0, b, steps)
E1 = float(sys.argv[1])
E2 = float(sys.argv[2])

answer = brentq(Final_Value, E1, E2) # use brentq to solve for E-> psi=0 at b.
```

```
print 'Eigenvalue found at E = %.8f' % answer

plot(x, y[:,0] )
xlabel(”Position (Units of L)”)
show()
```

It is a simple matter to change this code to find the anti-symmetric solutions, and one can find different solutions by giving the program different search limits E_1 and E_2. One can also easily change the shape of the potential well from square to just about anything else. With some creativity, it is also possible to adapt this method to one-sided wells and other asymmetric cases.

8.3 Problems

8-0 Write a program that uses the relaxation method to calculate and plot the potential in a 2-D region bounded by zero potential. There are three files in /export/250/shared/ that describe a grid of fixed potentials: dipole.txt, plates.txt, and cage.txt. These files contain "0" in the non-fixed-potential regions, and ± some value on grid points at which the potential is held fixed. Make sure your program gives sensible results for dipole.txt and plates.txt. (The solution to plates.txt is shown in figure 8.0.) Save an image file of a plot of your solution to cage.txt, and turn it in with your program.

8-1 Find all bound energy levels for a particle in a triangular well. The potential should be $V = 0$ everywhere except in the well, in which it drops linearly from zero at $x = -L$ to $V = -50$ at $x = 0$ and then climbs linearly to $V = 0$ at $x = L$. Use the simplified unit system from this chapter: $\hbar = 0$, etc.