Physics 120 – Problem Set # 3

(due Friday, February 1)

1. Griffiths, problem 2.35 and 2.36.

2. Griffiths, problem 3.6.

3. Griffiths, problems 2.47 and 3.11.

4. Since the equations of electrostatics are the same as the equations for an incompressible, irrotational fluid, we should be able to solve fluid problems using methods from electrostatics. In particular, to solve the equations

$$\vec{\nabla} \cdot \vec{v} = 0 \ , \qquad \vec{\nabla} \times \vec{v} = 0 \tag{1}$$

we can define a *velocity potential* ψ such that

$$\vec{v} = -\vec{\nabla}\psi \ . \tag{2}$$

   (a) In an incompressible, irrotational fluid, justify eq. (2) and show that ψ obeys Laplace's equation.

   (b) Show that a nozzle at $\vec{r}_0$ that squirts fluid out isotropically with total flux

$$\Phi = \int d^2x \, \hat{n} \cdot \vec{v} \tag{3}$$

   gives a flow with the velocity potential

$$\psi(\vec{r}) = \frac{\Phi}{4\pi} \frac{1}{|\vec{r} - \vec{r}_0|} \ . \tag{4}$$

   and find the corresponding $\vec{v}(\vec{r})$.

   (c) Consider a nozzle placed a distance d from an infinite wall. On the wall, the fluid velocity vector must be parallel to the wall. (Why?) Show that this corresponds to the Neumann boundary condition for ψ: $\partial_n \psi = 0$. Now solve the problem for ψ and $\vec{v}$ using the method of images.

   (d) Consider a nozzle placed a distance a from the center of a solid sphere of radius b ($a > b$). Using the method of images, find ψ and $\vec{v}$.

   (e) By judiciously taking the limit $a \to \infty$, find the incompressible irrotational flow of a fluid moving with velocity $\vec{v}_0$ at infinity about a stationary solid sphere of radius b. Make a sketch of $\vec{v}(\vec{r})$ and check that this flow is sensible.

5. In this problem, you will construct a java applet that solves Laplace's equation numerically in two dimensions.

   (a) When we solve a differential equation numerically, we first must reduce the continuous problem to a discrete one. Laplace's equation in two dimensions involves the function $\phi(x, y)$. We must replace x and y with discrete variables and replace the differential equation by a difference equation. One way to do this is to replace continuous space by a lattice of points

   $$x = i \cdot a, \qquad y = j \cdot a \tag{5}$$

   where a is the lattice spacing and i, j are integers. The values $\phi(x, y)$ can be replaced by the elements of a matrix ϕ_{ij} or $\phi[i][j]$. A first derivative can be approximated by

   $$\frac{\partial}{\partial x}\phi(ia, ja) \approx (\phi[i+1][j] - \phi[i][j])/a \tag{6}$$

   A second derivative can be approximated as a higher difference. Using this approximation scheme, show that Laplace's equation can be written:

   $$(\phi[i+1][j] - 2\phi[i][j] + \phi[i-1][j]) + (\phi[i][j+1] - 2\phi[i][j] + \phi[i][j-1]) = 0 \tag{7}$$

   Rearrange this equation, and show that it is equivalent to the statement that the value of $\phi[i][j]$ at any point on the lattice is the *average* of the values of ϕ at the nearest neighbor points.

   This suggests a method for solving Laplace's equation numerically. We fix boundary values of ϕ, then sweep through the lattice and set the value of ϕ at every other point to be the average of the nearest neighbors, then repeat until the process equilibrates. This is called solving Laplace's equation by the *relaxation method*.

   (b) Go to the Physics 120 Web site and download the files:

   `Laplace.html, Laplace.java, LaplaceGUI.java, PhysicsApplet.java, Laplace.jar`

   Compile the applet `Laplace.java` and run it. This applet manipulates an array `A[i][j]`, $i, j = 0 \ldots 100$ which represents a discretized electrostatic potential ϕ. You will find that the applet has the following behavior: (1) When you press the button 'Cathode', you will be able to draw in purple in the box. The corresponding points of an array `phi[i][j]` are set to 100 Volts. (2) When you press 'Ground', you will be able to draw in black in the box. The corresponding points of the array are set to 0 Volts. For definiteness, the black boundary of the box, corresponding to the array elements $i = 0$, $j = 0$, $i = 100$, $j = 100$ are also set to 0 Volts. (3) When you press the button 'Erase', you can erase purple and black points. (4) When you press the button 'Reset All', the whole drawing disappears. (5) When you press the button 'Measure Voltage' and then click in the box, the value of the

potential at that point is displayed. (6) When you press the buttons 'Compute Energy', meaningless values are displayed. (7) Finally , when you press the button 'Solve Laplace', the box turns pink.

Look at the source code, given below, and figure out how the `solve()` method works. The program goes through all of the elements of the array `A[i][j]` (with `Nx = Ny = 100` and tries to set each element equal to 33.0 (a value chosen at random). When you color a square with the mouse, the GUI not only sets the value of `A[i][j]` but also sets the value of the corresponding elements of an array `State[i][j]` to a nonzero value. The `if` statement instructs the program to check for this and, if a square has been preset, to 'continue' directly to the next case in the loop. The statements at the bottom of the loop check the difference between the old and new values of `A[i][j]`. In a more realistic situation, you would monitor the maximum difference as a criterion for stopping the update of the array. The method `Legend.write` writes a character string just below the drawing.

Finally, notice that the whole operation is embedded in an outer loop that is executed 5 times. The operations outside this loop are `refreshPicture`, which redraws the picture with the new values of `A[i][j]`, and an `if` statement that checks whether the 'Stop' button has been pushed. You will find that it takes much longer to redraw the picture than to carry out one sweep through the array for the solution of the Laplace equation. You might wish to increase the number of times this outer loop is executed to make the program run faster.

(c) Rewrite the method `solve` so that it actually solves the Laplace equation using the relaxation method. Sweep through the array and set each element equal to the average of its neighbors. Respect the protected sites, which provide the Dirichlet boundary conditions. Find an appropriate criterion for stopping the process so that typical elements `A[i][j]` are computed with 1% accuracy.

(d) Try out your solver. Draw some configurations of cathode and ground and see if the resulting potential distributions are reasonable. Here is nice way to check that you have come close enough to equilibrium: draw a cathode which is a large closed circle. We know that the solution of Laplace's equation is a constant inside the circle, which should be equal to 100 Volts in this case. Probe the voltage in the center of the circle. How many iterations does it take before you find 100 Volts with 1% accuracy?

(e) Compute the distribution of the potential ϕ or `A` for some interesting shapes of cathode and ground. One shape that I would recommend to you is a cathode in the center of the screen surrounded by a broken circle of grounded segments (a 'Faraday cage'). How small can these segments be and still bring the potential outside the circle down to a few Volts?

(f) Construct a discrete approximation to the energy of the field configuration, and fill in the function `Energy()` in the applet to compute this energy from the array

`A[i][j])`. The value that you return from this function will then appear when you press the button 'Energy'. To be concrete, assume that the square is a 10 cm $\times$ 10 cm cross section of a system extended in the third direction, so that the discrete spacing of points is $a = 0.1$ cm. Return the energy/meter in J/m.

(g) Compute the energy/m stored in a small parallel-plate capacitor: two conducting plates parallel to $\hat{y}$, very long in the $\hat{z}$ direction, 3 cm long in $\hat{y}$, and separated by 2 cm in $\hat{x}$, one at 100 V, one at 0 V.

Hand in your java code, your answers to (a), (d), and (g), and a few pictures of potentials.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;

public class Laplace extends LaplaceGUI{

  double criterion = 1.0e-4;

  void solve(){
    double maxdiff = 1.0;
    int iteration = 0;
    do {
      for (int n = 1; n <= 5; n++){
        maxdiff = 0.0;
        for (int i = 1; i < Nx; i++){
          for (int j =1 ; j < Ny; j++){
            if (State[i][j] != NormalState) continue;
            double oldA = A[i][j];
            double newA = 33.0;
               //  put something more sensible here
            A[i][j] = newA;
            double delta = Math.abs(newA-oldA);
            if (delta > maxdiff) maxdiff = delta;
          }
        }
        iteration++;
    }
    refreshPicture();
    Legend.write("max. diff : "+maxdiff+ "      "+iteration);
    if (timetostop) break;
    } while ( maxdiff > criterion);
  }

  double Energy(){
     return 0.0;
  }
}
```

Figure 1: The source file `Laplace.java`.