Physics 120 – Problem Set # 8

(due Wednesday, March 13)

1. Griffiths, problem 5.46. You can use the analysis of the $\vec{B}$ field of one current loop that we did in class.

2. Griffiths, problem 6.7, 6.8.

3. Griffiths, problems 6.16.

4. In this problem, you will build an applet that computes magnetic fields for systems of long wires in the presence of magnetic material.

   (a) A good strategy for computing magnetic fields with arbitrary currents in the presence of arbitrary lumps of magnetic material is to solve for $\vec{A}$. With luck, the solution will look like that for the potential of a system of dielectrics that we did in Problem Set 6. A convenient choice of gauge helps. Show that, for wires and magnetic materials extended (infinitely long) in the $\hat{z}$ and for currents flowing parallel to $\hat{z}$, the magnetic field is perpendicular to $\hat{z}$. Show that the most general divergenceless $\vec{B}$ field with this property can be generated from a vector potential of the form

   $$\vec{A}(x, y, z) = (0, 0, A(x, y)) \tag{1}$$

   (b) Discretize the relation between $\vec{B}$ and $\vec{A}$. For example: $B^x[i][j] = (A[i][j] - A[i][j-1])/a$, where a is the lattice spacing. If μ (`mu[i][j]`) is the dimensionless permeability of a linear magnetic material, write a discretized form of the equation for H^x, H^y in terms of A.

   (c) Consider first the case where $\mu = 1$ everywhere. Let $\mathcal{I}(x, y)$ be the current density flowing in the $\hat{z}$ direction (in A/m^2). Show that $A(x, y)$ solves the Poisson equation in two dimensions with a source proportional to $\mathcal{I}(x, y)$. Discretize this equation. It is convenient to integrate over each cell of the lattice

   $$\texttt{Icell} = \int d^2x\, \mathcal{I}(x, y) \tag{2}$$

   So `Icell` has the units of Amps. Solve for `A[i][j]` and define an iteration procedure for computing `A[i][j]`.

   (d) Generalize this equation to the case in which the domain contains regions of nonzero μ.

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

   `Magnet.html, Magnet.java, MagnetGUI.java, PhysicsApplet.java, MagnetGUI.jar`

(Once again, the file `PhysicsApplet.java` is an updated version different from the previous one.) Compile the applet `Magnet.java` and run it. This applet manipulates an array `A[i][j]`, $i, j = 0 \ldots 100$ which represents the vector potential above discretized on a 2-dimensional grid. It also defines an array `State[i][j]`, $i, j = 0 \ldots 100$ which tells whether a given site is wire, magnetic matterial ('iron'), or free space. The value of the μ is fixed by a scrollbar at the bottom of the applet. Another scrollbar give the scale for the representation of the $\vec{B}$ and $\vec{H}$ fields. Both $\vec{B}$ and $\vec{H}$ are computed in gauss.

You will find that the applet has the following behavior: (1) When you press the button 'Up Wire', you will be able to draw in green in the box. The corresponding squares are assigned a current up out of the page with `Icell` = 10 A. (2) When you press 'Down Wire', you will be able to draw in red in the box. The corresponding squares are assigned a current down into the page with `Icell` = -10 A. (3) When you press the button 'Current', the total up and down current, in Amps, is displayed. (4) When you press the button 'Iron', you can draw in yellow in the box. The corresponding points of are assigned to be magnetic material. (7) When you press the button 'Measure' and then click in the box, the value of the $\vec{B}$ field at that point is displayed. (8) Finally , when you press the button 'Solve', very little happens.

Look at the source code, given below, and figure out how the `solve()` method works. The code is very similar to the `Dielectric` applet. The current and permeability at each site are found by testing for different state conditions. The computation is done in units of gauss, Amps, and meters, assuming a total domain size of 10 cm $\times$ 10 cm or $a = 1.0 \times 10^{-3}$ m for a 100 $\times$ 100 grid.

(f) Consider first $\mu = 1$. Implement the method of part (c) for solving the electrostatic equation by appropriately modifying the `solve()` method.

(g) Try out your applet in some simple situations. Draw parallel lines of up and down current, and see if the magnetic field has the correct form.

(h) Now design a magnet. Using at most 2000 A of current in each direction, construct a magnet that gives a 2 cm $\times$ 2 cm region with a field of about 100 gauss. Adjust the wires to make this field as uniform as possible. What uniformity can you achieve?

(i) Implement the method of part (d) to include magnetic material.

(j) Try out this capability in some simple examples. Draw a filled circle of iron, and put a wire at the center. Draw a circle of iron with vacuum in its center, and put a wire, first inside, then outside. Do you find the correct B and H fields for these situations?

(k) Draw a rectangle of iron, and put wires around it to make a 2-dimensional 'solenoid'. Do you find the correct B and H fields for this situation?

(l) Using at most 1000 A of current in each direction, (and unlimited amounts of iron) design a magnet that gives a field of 200 gauss in a 2 cm $\times$ 2 cm region region of free space. To what accuracy can you make the field constant in this region?

Hand in your java code for (f) and (h), your answers to (a), (b), (c), (d), (h), (k), and a few pictures of field configurations.

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

public class Magnet extends MagnetGUI{

  double mu0 =  (1.0e-3) * 4.0 * Math.PI;    /*  units are gauss m/A  */
  double criterion = 1.0e-6;

  public Magnet(){
    a = 1.0e-3;   /* grid spacing, in m  */
    Bscale = 5.0;   /*  scale, in gauss, for the display of B */
    Icell = 10.0;   /* current in A flowing out of /into one cell */
  }

  void solve(){
    double maxdiff = 1.0;
    int iteration = 0;
    double mu1, mu2, mu3, mu4, cellcurrent;
    do {
      maxdiff = 0.0;
      for (int n = 1; n <= 20; n++){
        for (int i = 1; i < Nx; i++){
          for (int j =1 ; j < Ny; j++){
    /*  query to find the value of mu and current in each cell */
    mu1 =  (State[i-1][j-1] == IronState) ?  mu : 1.0;
    mu2 =  (State[i][j-1] == IronState) ?  mu : 1.0;
    mu3 =  (State[i-1][j] == IronState) ?  mu : 1.0;
    mu4 =  (State[i][j] == IronState) ?  mu : 1.0;
            cellcurrent = 0.0;
            if (State[i][j] == UpWireState)  cellcurrent = Icell;
            if (State[i][j] == DownWireState) cellcurrent = -Icell;
            double oldA = A[i][j];
            double newA = 0.0;
        /* 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);
  }
}
```