Objective
In this lab you will do the following:

- Computationally model the electric field of a proton
- Computationally model the electric field of a dipole (two equal and opposite charges)

Overview
Computation / programming is one of the three basic approaches (along with experiment and theory) we have for exploring and understanding nature. A computer program can bring to life both analytic and non-analytic models, thereby affording greater insight. So, a significant component of our Lab activities this semester will be working on and with computer models.

1. Calculating and displaying the electric field of a single charged particle

![Diagram of electric field calculation](image)

**Theory**
If a particle with charge $q_p$ is at position $\vec{r}_p$, then its electric field at observation location $\vec{r}_o$ is

$$\vec{E}_p = \frac{1}{4\pi \varepsilon_0} \frac{q_p}{r^2_{o-p}} \hat{r}_{o-p} = \frac{1}{4\pi \varepsilon_0} \frac{q_p}{r^3_{o-p}} \hat{r}_{o-p},$$

where $r_{o-p} = |\vec{r}_{o-p}| = |\vec{r}_o - \vec{r}_p|$ is the distance from proton $p$ to observation location $o$, and $\hat{r}_{o-p} = \vec{r}_{o-p} / r_{o-p}$ is the unit vector pointing in that direction. The second form (with $r^3$ in the denominator) is equivalent but often handier.

If the particle has a net positive charge, the electric field vector points from the source charge toward the observation location (parallel to $\hat{r}_{o-p}$); if the particle has net negative charge, the electric field vector points from the observation location toward the source charge (anti-parallel to $\hat{r}_{o-p}$).

While the calculation is straightforward and can be done with pencil and paper, if you wish to visualize the electric field throughout a region of space (rather than at just a few points), it quickly becomes tedious, so a computational approach (writing a program) is an appealing alternative.
Programming

First, you’ll create a simple program to visualize a proton’s electric field at just one observation location. Then you’ll extend the program to visualize the field at several locations at once.

I. Organization

You should organize a program in the following way, inserting comments (##) to explain what the different sections of the program are doing:

- First, a section giving names to any constants that will be used.
- Second, a section in which visible objects like spheres or arrows are created and named. (Sometimes you will end up creating more objects later.)
- Third, a section in which important variables are named and given initial values, if any.
- Fourth, the calculations.

II. A VPython program to calculate the electric field due to a single charged particle.

II.1. Creating a new program file

Start a new VPython program by opening VIDLE from the desktop and saving the program as “proton1.py” (note: you have to type the “.py” yourself.)

The first two lines at the very beginning of your program should be:

```python
from __future__ import division
from visual import *
```

In the first line note the two underscores before and two after the word “future”. This makes Python treat (1/2) as a floating point number (0.5) instead of an integer (0).

After that, please use a comment line (starting with ##) to list your name(s).

II.2. Constants

Constants should be defined at the beginning of your program. After the lines you’d just written, add:

```python
## constants
oofpez = 9e9 # N m**2 / C**2
qproton = 1.6e-19 # in Coulombs, charge of a proton
```

(oofpez stands for One Over Four Pi Epsilon-Zero)

- To make sure you have typed everything correctly, add the statement

  ```python
  print(oofpez)
  ```

- Run your program by pressing F5 or by using the Run menu. Make sure the correct value of the constant prints out in the Python Shell window.

Note: Leave the shell window open (it will automatically close when you close the editing window); closing it can cause errors when you run the program the next time.
II.3. Objects

- Add the following two lines to mark this section of the code for creating objects and to create your charged particle. You’ll want to fill in the blanks so the particle is located at the origin and has a radius of $1 \times 10^{-11}$ meters (an actual proton is much smaller; exaggerating the size will make it more visible in the display.)

```
## objects
proton = sphere(pos = (   ,   ,   ), radius =  , color = color.red)
```

- Run the program. You should see a sphere in the center of your display window. If you don’t see anything, first check the Python Shell window for error messages (that’s what it’s there for), address them, and then try again.

II.4 Initial values of variables

- In your program add these lines:
  ```
  ## initial values
  obsloc = vector(2.1e-10, 2.1e-10, 0)
  ```

This marks this section of your code for creating and initializing variables and creates the variable “obsloc”, which is short for “observation location.” You’ll use “obsloc” in your program to represent the locations where you determine or ‘observe’ the electric field.

Note that a `vector` such as this is not a displayable graphical object, so it doesn’t have a “pos” attribute (rather, it can be used to define the position of an object such as an arrow or a sphere.) We refer to this vector simply by its name `obsloc`, whereas the position attribute of the displayable sphere object is a vector which is referred to as `proton.pos`.

II.5 Calculations

- In your program add (and complete) the following lines to calculate the electric field at the observation location.
  ```
  ## calculations
  rp = #relative position vector from proton to observation location
  rmpag = mag(rp) #magnitude of relative position vector
  Ep = #electric field evaluated at observation location
  ```

Syntax Note, Exponentials: $x^n$ would be written `x**n`

To make sure you’ve calculated this properly, print the value of the electric field vector and check it in WebAssign.

- To visualize the relative position vector, $\vec{r}_{o-p}$ in the figure at the beginning of this handout, create an arrow whose tail is positioned at the particle’s position and tip extends from there to the observation location. Once you’ve added (and completed) the following line of code, run your program to make sure it does what you intend.
To visualize the electric field vector, create an arrow whose tail is positioned at the observation location and whose direction and length (axis) are that of the electric field vector. Once you’ve added (and completed) the following line of code, run your program to make sure it does what you intend.

```python
Earrow = arrow(pos = , axis = , color = color.orange)
```

You probably noticed that `Earrow` completely dwarfs `rarrow` and the proton’s sphere. So that the electric field can be conveniently visualized in the same space as the particle itself, you’ll want to scale down its length. So, you’ll define a constant “`scalefactor`”; but first, what’s a good value for it? The exact value you choose isn’t important, but you generally want the Electric field vector to be on the same scale as the relative position vector. So a reasonable value would be `r`mag/mag(E).

To determine an appropriate value for `scalefactor`, temporarily add the following line of code.

```python
print(rmag/mag(E))
```

Up in the constants section of your code, define `scalefactor` and give it the value that got printed (or any nice, round value that’s in its ball park.) Once you’ve done that, feel free to comment out or erase the print statement that you’d added.

In the line of code that defines `Earrow`, change axis to equal `scalefactor`*E. Run the program to make sure things look good (if not, tweak `scalefactor`’s value until they do.)

Save and upload your code.

Save a new copy of your code as proton2.py; you’ll modify this copy to show the electric field at multiple locations.

### II.6 Add more observation locations at constant distance

To see the pattern of electric field around a charged particle, you will extend your program to calculate the electric field at many locations, all the same distance from the source charge. From Phys 231, you’re familiar with using a loop to evaluate properties (positions, momenta, and forces) at different times; now you’ll do something similar to evaluate a property (electric field) at different positions. Since you want to evaluate that field at different locations with the same distance, you’ll loop through different angles, `theta`, to your observation locations.

- In the **constants** section of your code, add the following line (analogous to defining your `deltat`):

  ```python
deltatheta = pi/10
```

- In the **initial values** section of your code, add the following line (analogous to starting at `t=0`):
theta = 0

- At the beginning of your calculations section, add the following two lines to rephrase the observations locations in terms of polar coordinates, magnitude and angle.

\[
\text{obslocmag} = \text{mag}(\text{obsloc}) \\
\text{obsloc} = \text{obslocmag} * \text{vector}(\cos(\theta), \sin(\theta), 0)
\]

To loop through angles:
- Just between these two new lines add (analogous to while \( t < \text{max\_time} \))

```python
while theta < 2*pi:
```

- At the end of the calculations section add (analogous to \( t = t + \text{deltat} \))

```python
theta = theta + deltatheta
```

- So all lines after the while are included in the loop, indent all of them (that is, all lines after, but not including, the while line.) The easiest way is to highlight all of them, then under the “Format” menu, choose “indent region.”
- Finally, comment out the rarrow line and save and run your program. Compare the picture your program generates with that of another group (remember, you may have chosen slightly different scale factors.)

When confident that your program is working correctly, save and upload it.

Resave the program as proton3.py; you’ll modify this copy to show the electric field at multiple distances.

II.7 Add more observation locations at greater distances

To see that its strength drops off with distance, you’ll need to evaluate the electric field at greater distances from the proton. Just as you nested the calculation of electric field within a loop over angle, now you nest \( \theta \) within a loop over distance. You could use another while loop, but since we’ll only look at a few distances, a new structure will be handier, a for loop.

- Replace the obslocmag line with the following

```python
for obslocmag in [mag(obsloc), 2*mag(obsloc), 3*mag(obsloc)]:
```

- Indent all lines below it (highlight them and from the “Format” menu select “indent region”)
- At the bottom, indented only as far as the while line, add

```python
theta = 0
```

Here’s what these changes to your code should do. First, it sets obslocmag equal to
mag(obsloc) and then cycles through evaluating the field at different angles. At the end of that, it resets the angle back to 0, and then changes obslocmag to 2*mag(obsloc) and once again cycles through evaluating the field at each angle, but out at the new distance. Once it’s done, it resets the angle to zero and repeats the calculations for an observation location a distance 3*mag(obsloc) out.

Save and run your program. Upload a copy.

III. A VPython program to calculate the electric field due to a dipole.

A dipole is a pair of equal and opposite charges, like an electron and a proton, a short distance apart. You’ll write a program to display the total electric field (due to both electron and proton) at several locations in space. This can be done by simply, but carefully, modifying your proton3.py program.

The dipole you’ll model consists of:
• a proton of charge +e located at < 4x10^{-11}, 0, 0 > m, represented by a red sphere, and
• an electron of charge –e located at < -4x10^{-11}, 0, 0 > m, represented by a blue sphere.
• The observation locations can be the same as in your proton3.py program.

1. Diagram and planning
Just like when you’re doing an analytical (pencil and paper) problem, you’ll find that a good diagram prepares you to tackle a computational problem. So, on a whiteboard, make a careful 2D diagram of the system, in the x-y plane, much like that appearing on the first page of this handout. It should include position vectors for the observation location, electron, and proton (\vec{r}_o, \vec{r}_e, and \vec{r}_p), relative position vectors from the charged particles to the observation location (\vec{r}_{oe} and \vec{r}_{op}), the proton’s contribution to the electric field at the observation location (\vec{E}_p), the electron’s contribution to the electric field at the observation location (\vec{E}_e), and the net electric field at the observation location (\vec{E}_o = \vec{E}_p + \vec{E}_e).

When you’ve got a complete and well labeled diagram, compare it with that of a neighboring group

2. Write/Modify the program
You just wrote a program that displays the electric field due to a single point charge. While you could write a new one from scratch that does the same for a dipole, I’d recommend opening your proton3.py and resaving it as dipole.py, and then modifying it as follows.

• In the constants section, define the charge of the electron, qelectron.
In the objects section, move the proton from the origin to its new location (see above), and define an electron placed at the appropriate position and colored blue.

In the calculations section, define the relative position from electron to observation location, re, calculate its magnitude, magre, and use these and qelectron to find its contribution to the electric field, Ee. Finally, calculate the total electric field, E = Ee + Ep, and change Earrow to depend on this total field rather than just Ep.

It’s that simple.

Save it, run it, compare it with that of a neighboring group, and upload it when you’re satisfied.

Playing around
You’re done with the lab, but the fun doesn’t have to stop here. You can take home / email yourself / download from WebAssign a copy and continue playing with it. Here are some suggestions of things you might like to try after turning in your program.

• Add a circle of observation locations in the xz plane.

• Make a “quadrupole” out of two opposed dipoles, and display the electric field. The dipoles can be aligned on one axis (+ - +) or they can be placed one above the other, with opposite orientations:

   \[ \begin{array}{ccc}
   & + & - \\
   - & - & + \\
   \end{array} \]

   You’ll need to increase scalefactor, because the two dipoles nearly cancel each other’s fields.

• In the Visual reference manual available at http://vpython.org/contents/docs/index.html, in the section on “Mouse Interactions”, find out how to determine the current position of the mouse. Display the electric field at the mouse location, either where you click the mouse, or continuously as you drag the mouse. It’s interesting to display the electric fields contributed by each charge, as well as their sum, the net electric field.

• Modify a copy of your program so that you place a proton at rest at a location on the +y axis, then release it, and animate its motion by calculating the electric force, then applying the momentum principle \( \vec{p_f} = \vec{p_i} + \vec{F_{net}} \Delta t \) to update the momentum, then update the position, in a loop over time. Note that in order to calculate the force on the proton you always have to calculate the electric field of the dipole at the current location of the proton.

Try a deltat of around 1e-17 s, to start with. The trajectory is quite surprising. Think about whether this trajectory is consistent with the Energy Principle.