The Electric Field of a Uniformly Charged Rod
Part I: One Observation Location

1. Objectives

In your textbook you can find the derivation of an equation for the magnitude of the electric field of a uniformly charged thin rod, at observation locations in a plane perpendicular to the rod and bisecting the rod (the "midplane"). However, what if we need to know the electric field at a location not in the midplane–for example, a location near the end of the rod? In this case, as in many other real-world cases, we can do the calculation numerically, by approximating the rod as a linear array of point charges, and adding up their contributions to the field at the observation location.

In this activity you will:
• Construct a program that does a numerical calculation of the electric field of a uniformly charged rod, in a form general enough that only a one-line change is required to specify a different observation location, or to change the number of point charges in your model.
• Make sure your program gives correct results, by comparing your program's calculation of the electric field in the midplane of the rod to the results of applying the equation derived in the textbook.
• Calculate and display the electric field at locations not in the midplane, where the analytical (symbolic) equation derived in the textbook does not apply.
• Explore the effect of increasing or decreasing the number of point charges used in the model.
• Explore how far from the midplane an observation location must be before the field differs markedly in magnitude or direction from the field in the midplane.

You should finish Part I in 60 minutes or less. Check with your instructor as to whether you are supposed to do Part II or Part III, too.

Before starting, agree on the responsibilities of the Manager, the Recorder, and the Skeptic for this activity. Record your answers to the QUESTIONS asked here; you will need them later.

2. Explain and predict program visualization

A link to a minimal working program has been provided on the VPython Files and Resources page of the Course Materials area of the course Canvas site. **Click on the linked file (named Efield_rod_mwp.py) and save it. Remember to give it the extension** .py. **Open VIDLE, then open the saved program. DO NOT RUN THE PROGRAM YET. Read through the program together. Make sure everyone in the group understands the function of *each* program statement. Reading and explaining program code is an important part of learning to create and modify computational models.**

After reading *every* line of the program, answer the following questions:

• SYSTEM: What is the physical system being modeled? Approximately, what should the pattern of electric field look like? **On the left side of a whiteboard, draw a sketch showing the source charges and how you think the net electric field should look at the specfied observation location.**

• PROGRAM: How many loops are in the program? What is each loop supposed to do?

• PROGRAM: Initially, how many point charges are used in the model of the rod?

• PROGRAM: Initially, how much electric charge does each of the point charges have?

• PROGRAM: What is the significance of the variable dxq?

• PROGRAM: Why does the loop start at -L/2 + dxq/2, rather than at -L/2?

• PROGRAM: Will the program *as it is now written* accurately calculate the electric field of the real system?

DO NOT RUN THE PROGRAM YET. Study the program again. **On the right side of the whiteboard, draw a sketch of how the electric field displayed by the program will look, based on your interpretation of the code.**

Run your program *after* everyone agrees on both predictions. Rotate the display to help understand where objects are located. Discuss how closely your prediction of what the program would do matches what you see in the visual output.

Checkpoint: Discuss your answers to the questions above with your instructor.

2.1. Modify the program

Complete or correct the program, to compute and display the electric field at the single observation location $\langle 0, L/4, 0\rangle$. It may be helpful to look at your program that calculated the electric field of a single point charge, or a dipole.

2.1.1. Notes on the minimal working program

In the calculation loop, the location of each of the point source charges is calculated as a vector sourcelocation = vector(xq,0,0) so that you can use the source location in vector calculations of the electric field dE contributed by that source charge. (Note that since sourcelocation is a vector and not a graphical object, it does not have a position. Just say sourcelocation; don't use .pos.)

In the constants section are the values of the physical constants you will need. You will eventually need a scale factor for representing electric field vectors with arrows, and this has temporarily been given the value of 1.0 (you will need to change this later). You may wish to review the video "VPython Instructional Videos: 5. Scalefactors" at http://www.youtube.com/VPythonVideos to see how and why to scale arrows in your program to represent physical vector quantities.

3. Numerical experiments

1. Look at your display. Does it make sense? Vary the x component of the observation location several times and redisplay the electric field each time. Do the magnitudes and directions of the orange arrows make sense? Repeat a few times for changes in the y component of obsloc.

2. Rotate the "camera" with the mouse to get a feeling for the nature of the pattern of electric field.

3. Change the sign of the charge to negative and run. Does the display make sense? Reset the charge to positive.

4. To quantitatively check your program, set the *x* coordinate of the observation location to zero and record the numerically computed electric field (this will take a print statement). Using your calculator, calculate the electric field at this location using the equation for field in the midplane derived in the textbook:

$$E_{rod} = \frac{1}{4\pi\varepsilon_0}\frac{Q}{r\sqrt{r^2 + (L/2)^2}}$$

• Does the value calculated by your program have the same order of magnitude as the value you obtain using the analytical equation? If not, check both the program and your calculation.

Checkpoint: Ask your instructor to look over your work, including your answers to the questions above.

4. Use your program to answer WebAssign questions, and turn it in.

Once you are certain that your program is correct, complete the WebAssign assignment for this lab exercise (named “Electric field of a charged rod”), which will require that you make a few changes to the parameters of the rod and use your program to answer a few questions, before handing it in.

5. Optional: Playing around (extra credit)

Here are some suggestions of things you might like to try after turning in your program.

• If there are red-cyan stereo glasses available (red lens goes on left eye), you can display true stereo by adding the following statement near the start of your program (for more about this, see the section “Controlling Windows” in the Visual reference manual available from the Help menu):

```
scene.stereo = ‘redcyan’
```

• In the Visual reference manual available on the Help menu, in the section on “Mouse Interactions”, there is documentation on how to make your program interact with the mouse. Some of the techniques described there are used in the following:

To wait for a mouse click and get its location as the observation location, do this in a perpetual loop (while True:) that calculates the net electric field:

```
while True:
    # replace obslocation calculation with this:
    rate(30)  # check for a mouse click 30 times per second
    obslocation = scene.mouse.getclick().pos # wait for mouse click
    .... # calculate the electric field at obslocation
```

If instead you want to drag the mouse and continuously update the electric field, remove the arrow and print statements from inside a perpetual loop that calculates the net electric field, and create just one arrow before starting the loop, then continuously update that arrow’s attributes:

(continued)

```
Earrow = arrow(axis=(0,0,0), color=color.orange)
scene.autoscale = False # prevent large mouse moves from autoscaling the scene
while True:
    rate(30)  # check for a mouse move 30 times per second
    obslocation = scene.mouse.pos
    # calculate Enet
    ...
    Earrow.pos = obslocation
    Earrow.axis = scalefactor*Enet
```