Homework #5

Physics 129 Spring 2022

Problems due **Saturday, April 30, at 11:55 P.M.**

Please read the homework guidelines handout on the course web page.

Before attempting this assignment, ensure your RPi is connected to the Internet, then run the `update_physrpi` script.

Better answers and code will get better grades.

Reading

→ Complete by **Monday, May 2**

- Read chapter 20 in Shotts.
- Read sections 8.1 and 8.2, and Appendix F in K&N.

Problems

1. **Wind Speed.** Examine the `wind.dat` file in `$HOME/physrpi/coursefiles/` . The lines in this file contain 3 numbers each, the local time of day in hours, the average wind speed in knots, and the uncertainty in the average wind speed.

   Write a program to read the `wind.dat` file and plot average wind speed as a function of local time of day. Display the data points separately with error bars, and do not connect them with any lines or curves. Set the axis limits appropriately, label the axes, and include a title. Save a copy of your plot as encapsulated PostScript (.eps) and turn in this file. You can view .eps files with the `gv` program.

   Hint: Study the `simple_plot.py` example function in `$HOME/physrpi/python/` on your RPi.

2. **Doing your backups?** Turn in the output of `ls -alF` on the top level of your flash drive and also the top level directory of your most recent backup on the flash drive. The backup must have been created no earlier than one week before the due date for this problem set.

3. **diff.** At some point while you are modifying example code to solve one of the homework problems in this set, compare your new program to the original using `diff`. Turn in the output.

   Hint: `diff` is discussed in chapter 20 of Shotts.

4. **3-4-5 Right Triangle.** Examine the code in `$HOME/physrpi/ps/rt345.eps` on your RPi, and use the `gv` program to view the result.

   Using what you need from `$HOME/physrpi/python/img.py`, write a program that draws the same figure using raster graphics (by setting and displaying pixels). Make your image 512 pixels wide. In the text file for this problem, explain how you chose which pixels to set.

   Hint: Look at the bottom of the Handouts section of the course web page for links to the *PostScript Language Tutorial and Cookbook* and the *PostScript Language Reference Manual*.

5. **Mandelbrot Set.** A complex number c is in the *Mandelbrot Set* if upon repeated iteration the expression $z_{n+1} = z_n^2 + c$, with $z_0 = 0$, does not diverge. In other words, $|z_n|$ remains bounded no matter the value of n. In typical illustrations, the complex plane is shown with the Mandelbrot Set in black and the surrounding points c colored according to the number n of iterations required for $|\, z_n^2 + c \,|$ to exceed some chosen value.

   Write a program that plots an interesting region of the complex plane in this fashion. Note: a region containing the whole set is hardly the most interesting you can choose; try zooming in near the edges. Use a grid of 512×384 pixels, and a limit of 250 iterations per point. In other words, if $|\, z_n^2 + c \,|$ is still below some small limit of your choosing (think about what a good choice might be) after 250 iterations, assume it is in the set.

   Save your plot as a PostScript file (not .eps) and use the `translate` and `scale` operators to center the image on the page at a size that leaves 56-point horizontal margins. A good place to put these commands is immediately following the line

   ```
   %%Page: 1 1
   ```

   Convert the PostScript file to PDF using `ps2pdf` and turn in the PDF file with the code and text file for the problem.

   Hints: You will probably find it useful to use a smaller grid until you have debugged and optimized your code. It will also help to include some print statements that notify you of the program's progress.

   See `cmapimg.py` in `$HOME/physrpi/python/` on your RPi for an example of plotting points with a color map rather than explicit RGB values.

6. **PostScript Flower.** Examine the code in `$HOME/physrpi/ps/petal.ps` on your RPi, and use the `gv` program to view the result.

   Write a program in Python that takes as input from the user a number of petals, then writes to disk a PostScript program (a `.ps` file) that draws a flower with the specified number of non-overlapping, symmetrically placed petals. Your program may limit the number of petals to some reasonable range.

   Extra credit will be given for a stem and leaves.

Hints: Remember you can use triple quotes to create multiple-line strings. Assemble your output as one large string, then use the appropriate file method to write it to disk. You can concatenate strings with +. I suggest you *not* try to write elegant PostScript code. Do all the work (for example, loops and calculations) in Python, and make the PostScript output as simple as possible.