CS 510: Fall 2017 Final

Due: December 15th, 2017

Given complex numbers z and c, consider the following discrete update map:

$$z \mapsto z^2 + c. \tag{1}$$

Recall that we used this map to generate *Julia sets* by fixing c as an initial constant, varying the value z across the complex plane, and counting how many iterations N of the map were required for the magnitude of z to exceed the radius $|z| < 2$.

We can also perform a similar task, but switch which variable is held constant. That is, we can fix an initial value z and vary the value c across the complex plane, while counting how many iterations of the map are required for the magnitude of z to exceed the radius $|z| < 2$. This variation in the algorithm produces what is known as the *Mandelbrot set*.

In this project you will generate Julia sets in C, Mandelbrot sets in C++, plot them both in Python, and automate the entire work flow in bash.

Problem 0. Framework (20pt)

Edit the `README.md` file so that it describes this assignment properly (5pt), and sign it to verify that all submitted work is your own (5pt). Place your code in directories `src` and `bin` as appropriate (see below) (5pt), with a `Makefile` in each directory that needs one (5pt). Note that no Travis testing will be required for this project for simplicity, but your code will be expected to work properly, so test your code accordingly.

Problem 1. Julia Sets in C (20pt)

Write a C program in `src/julia/julia.c` that creates a Julia set. Create a struct type, COMPLEX, in a header file `src/julia/complex.h` with implementation in `src/julia/complex.c`. (This code can be reused from previous class and homeworks.) For the COMPLEX struct type, implement functions `add_complex`, `square_complex`, and `julia`, which will be used to implement the map in Eq. (1).

In the file `src/julia/julia.c` create a main function that expects 8 command line arguments: `xmin, xmax, ymin, ymax, xpoints, ypoints, creal,` and `cimag`. The first 6 will be parsed to specify a gridded region of the complex plane. The last two will be parsed together as a COMPLEX struct, `c = creal + cimag i`. Create a function `iterate(x, y)` that takes two long doubles, creates a COMPLEX `z = x + y i`, then iterates the function `julia(z,c)` until the complex result has a magnitude larger than 2 or the number of iterations exceeds the global upper bound `#DEFINE MAXITER 256`, then returns an unsigned integer of the number of iterations required, or 0 if `MAXITER` is reached. Have the main function iterate over pairs of values `(x, y)` with `x` evenly spaced in the range `[xmin, xmax]` and `y` evenly spaced in the range `[ymin, ymax]`, such that there are a total of `xpoints` columns and `ypoints` rows. For each pair `(x, y)` call `iterate(x, y)` to find the result `out`, then print a single CSV line to STDOUT `"x, y, out\n"` with the values for that iteration. By printing the CSV output to STDOUT in this way, we will be able to use bash later on to redirect the output to any desired filename.

Create a Makefile that properly compiles `julia.c` and `complex.c`, and links `src/julia/complex.o` into the executable file `bin/julia`. Done properly, executing `"julia -2 2 -2 2 1000 1000 -1.037 0.17"` in a bash shell will run the program and print the CSV file for the indicated Julia set to STDOUT.

Problem 2. Mandelbrot Sets in C++ (20pt)

Write a C++ program in `src/mandelbrot/mandelbrot.cc` that creates a Mandelbrot set. Using the built-in type `std::complex<long double>` write a function `mandelbrot` that implements the map in Eq. (1). Create a main function that expects 8 command line arguments: `xmin, xmax, ymin, ymax, xpoints, ypoints, zreal,` and `zimag`. The first 6 will be parsed to specify a gridded region of the complex plane. The last two will be parsed together as a complex initial point `z0 = zreal + zimag i`. Create a function

`iterate(x, y)` that takes two long doubles and creates a complex `c = x + y i`, then iterates the function `julia(z0,c)` until the complex result has a magnitude larger than 2 or the number of iterations exceeds the global upper bound `#DEFINE MAXITER 256`, then returns an unsigned integer of the number of iterations required, or 0 if `MAXITER` is reached. Have the main function iterate over pairs of values `(x, y)` with `x` evenly spaced in the range `[xmin, xmax]` and `y` evenly spaced in the range `[ymin, ymax]`, such that there are a total of `xpoints` columns and `ypoints` rows. For each pair `(x, y)` call `iterate(x, y)` to find the result `out`, then print a single CSV line to STDOUT `"x, y, out\n"` with the values for that iteration. Similar to the C program, we are using STDOUT to allow the output to be redirected later with bash.

Create a Makefile that properly compiles the executable file `bin/mandelbrot`. Done properly, executing `"mandelbrot -2 2 -2 2 0 0"` in a bash shell will print the CSV file for the standard Mandelbrot set to STDOUT.

Problem 3. Plotting in Python (20pt)

Write a python script named `bin/plot_data.py`. It should parse one command line argument that gives the name of a CSV file, with each line formatted as in your program output: `x, y, out\n`. The script should load this CSV file into a pandas dataframe, generate a 2D heatmapped plot using matplotlib, and save the plot to a PNG file with the same basename as the CSV file. (Hint: you will need the line `matplotlib.use('Agg')` below the import statements and the command `savefig` for the python program to directly output plots to PNG files.) As an example, calling `bin/plot_data.py data/mandelbrot.csv` from bash should produce the file `data/mandelbrot.png` containing a nice image of your Mandelbrot set. The plot inside the PNG file should look similar to the plots created earlier for the python-generated julia sets and the midterm, but zoomed into the regions of x and y specified by the data in the CSV file, and clearly labeled.

Problem 4. Automation scripting in Bash (10pt)

Write a bash script named `bin/generate_data.sh`. It should change directory into the top repository directory, execute `make` to build your two programs, create a `data` directory, then execute `bin/julia -2 2 -2 2 800 800 -1.037 0.17 > data/julia.csv` as well as `bin/mandelbrot -2 2 -2 2 800 800 0 0 > data/mandelbrot.csv` to create default CSV datafiles in your data directory. After generating these data files it should run `bin/plot_data.py data/julia.csv` and `bin/plot_data.py data/mandelbrot.csv` to create the associated plots in PNG images. Note that the compiled binaries, data files, and plots should not be committed to git: When done properly, one should be able to clone your git repository and run `bin/generate_data.sh` in your top repository directory to have all your code built properly and the default datafiles and plots generated. (Don't forget to `chmod` files in the bash script if needed to make them executable.)

Problem 5. Scientific Report (10pt)

Now that you have working code, use it! Create and zoom into interesting regions of the complex plane in both Julia and Mandelbrot sets. Pick three Julia sets that you really like (i.e., with a particular `c` values) and produce a high-resolution plots (e.g., 1920x1920 points) showing an interesting region. Similarly, produce three high-resolution plot for an interesting regions of the standard Mandelbrot set with `z0 = 0 + 0i`. Commit your chosen high-resolution plots in the `data` directory, load the images in a well-formatted Jupyter notebook `Final.ipynb` (using Markdown cells), and describe your plots, as well as why you thought those regions were particularly interesting.