Homework 1

Marek Mateusz Narozniak

September 18, 2017

Abstract

First homework consists of computing Mandelbrot set and fitting line to a dataset using Least-Squares method. Completing this work requires ability of handling numerical computations and basic programming skills in Python.

1 Introduction

Plotting an image of Mandelbrot fractal is good problem to learn how to handle complex numbers in Python programming language. It also leaves a lot of space for improvements to train more skills, such as handling arbitrary precision floating point numbers by zooming in the fractal or learn parallel programming by splitting fractal generation into multiple cores. The final result is visually pleasing which makes this project more fun to work on.

The interesting part about Least-Squares methods is its usability - even if phenomenon is linear in nature, measurement doesn't need to be accurate enough to meet the computer precision of floating point numbers. Fitting line to set of points is common practice and for many researchers, useful almost on daily basis.

Rendering Mandelbrot fractal appears to be simulation-like problem, while Least-Squares method is an optimization method. Understanding and fluency in both of those domains is crucial to begin work in the field of Computational Physics, which makes those problems perfect for start.

2 Mandelbrot

2.1 Implementation

2.1.1 Saving data and multithreading

Generating Mandelbrot set takes more time than plotting it due to difference of computational complexity between those tasks. Finding good way to plot higher resolution data takes time and having to regenerate it at each attempt would be time consuming, to deal with that issue I implemented a way to save resulting points set to a file using Pickle module and restore it in same way.

During implementation I noticed problem of generating Mandelbrot set is not concurrent, in simple words, each final point depends only on single initial point. This makes it easy to parallelize the processing take advantage of multiple processors to speed-up the generation. To split problem into multiple threads I split initial points into multiple regions and run Mandelbrot set generation asynchronously on each of those regions. After processing is done I combine results of each of those processes.

2.1.2 Plotting

There is a significant difference between the method I used to plot Mandelbrot set and method suggested by the author of the assignment. I decided not to use density plot with number of iterations influencing the color, instead I used two dimensional histogram with Mandelbrot datased binned into given number of bins. Number of bins influences the final resolution of the picture - each region is one bin and its color depends on number of points included inside of it.

Primary reason why I chosen this method is I find it more aesthetically pleasing to have slightly "pixelated" image, secondary reason implementing it this way allowed me to perform much faster as I was already familiar with that method.

2.2 Results

Due to plotting strategy I chosen picture is not as clear as it can be found in other sources, however it is clear enough to recognize famous Mandelbrot fractal shape.

Mode	Processing time
Single thread	165.816s
Two threads	113.978s

Table 1: Real processing time of Mandelbrot set generation depending on number of threads used.

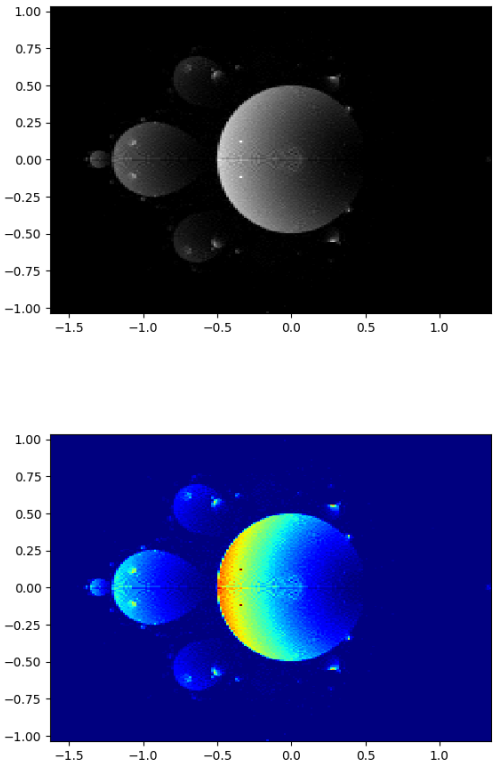

Figure 1: Mandelbrot set generated using my program, in grayscale and with Jet color map.

Multithreading provides noticeable speed-up, for generating Mandelbrot set with grid of size 1000×1000 and 1000 iteration maximum (plot visible on figure 1) two threads performed 37% faster. This computation was performed on using two core processor Intel(R) Core(TM) i5-5257U CPU @ 2.70GHz.

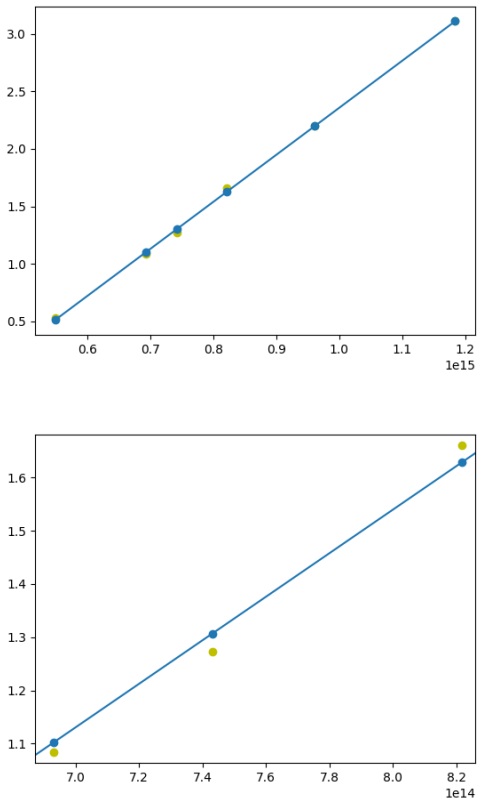

Figure 2:

3 Least-Squares

3.1 Implementation

I followed the instructions of implementation exactly as explained in the assignment. I also made it possible to run this program on any dataset that follows the same data format.

3.2 Results

Millikan dataset contains only 6 points and those points are very close to line that fits them. This makes it hard to see they do not lay on that line. I zoomed in a part of this figure to make it visible more clearly. Numerical results are included in the file *millikan_res.txt* and they follow the same format as input file used *millikan.txt*.

4 Usage instructions

I implemented programs for this assignment with command line arguments and I included example commands in *README.md* file in root directory of GitHub repository. Repository contains all the results of this work and *README.md* file contain all information required to reproduce my results.