Iraj Eshghi

(Dated: 12 September 2017)

I. MANDELBROT

I apologise in advance for unimaginative function names.

A. Introduction

The aim of this problem is to draw a graphical representation of the Mandelbrot set. Let us first define this set:

We start by defining a class of functions $f_c(z) = z^2 + c$, and apply it iteratively for every point c in the complex plane, starting with $z = 0$.

In more formal language, if we define the set of sequences $\{c_i\} = f^{(i)}(c_0) \forall c_0 \in \mathbb{C}$, where $f^{(i)}$ indicates successive iteration of f, then the Mandelbrot set is the set of accumulation points of these sequences, which means the set of limits of all the convergent sequences in $\{c_i\}$.

One theorem we will use to make this calculation easier is that every point in the set is contained within the disk of radius 2 centered at the origin. Even stronger, we know every sequence that converges to the Mandelbrot set stays in the disk $\forall i$. Thus, if our iteration ever brings us outside this circle, we can abort it.

The goal of this assignment is to draw the Mandelbrot set.

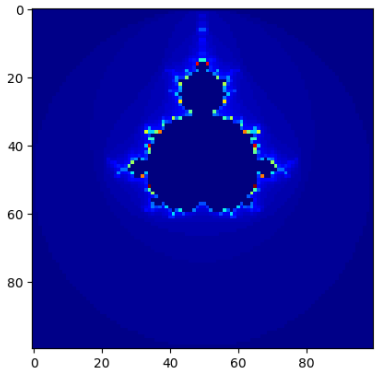
B. Code

The code is split up into two functions: the first one is called "checkmand" and takes one complex variable, and calculates whether a given point is within the Mandelbrot set or not, by successively applying the function f defined above, until either it hits 100 iterations or its modulus becomes greater than 2. If the sequence diverges, it returns the number of iterations that took, this way we can have a measure of which points "escape" the set fastest. Otherwise, it returns 0.

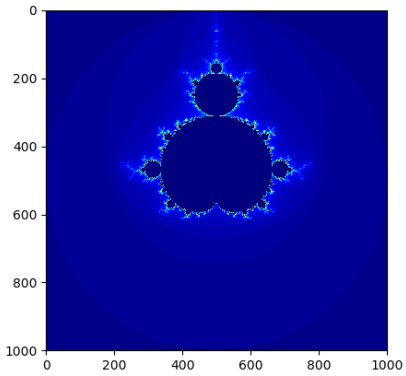
The second function (called "mandel", takes an integer) traverses the whole grid we want to consider (we chose the square bounded by $|\mathrm{Re}\{z\}| < 2$ and $|\mathrm{Im}\{z\}| < 2$), indexes it with N gridpoints on each axis, N being input by the user, and then proceeds to call *checkmand* for each of those points. Then, it simply calls the matplotlib function *imshow* on that grid, with the *jet* parameter, thus giving us a beautiful image of the set.

C. Results

Here is the output of the function for $N = 100$:



And here is the output for $N = 1000$:



D. Discussion

As we can see, an increase in N give us an obvious increase in resolution, so the image seems to converge to the shape the set is known to have. But, since the Mandelbrot set is a fractal, we cannot write down an analytical equation for it or its boundary, and so we cannot do simple convergence analysis on the picture. Thus we must limit our appreciation of the accuracy to a qualitative statement.

The main bottleneck in terms of computational time in this program is in the way that the grid function is written. There should be a pre-written parallel way to call the *checkmand* function for every point simultaneously, but this code has a nested loop which goes through the grid point by point, which is grossly inefficient. Thus, our computation time should grow as something like $\mathcal{O}(N^2)$.

II. LINEAR REGRESSION AND MILLIKAN

A. Introduction

The goal of linear regression is broadly to find general trends in data. More specifically, a linear fit attempts to find *linear* trends in the data.

Since a line is uniquely defined by two parameters, the slope and y-intercept, this problem can be reduced to an optimisation problem to minimise the total "error" of the line given the data.

The measure of "error" that we use is the χ^2 function that is defined in the book, which comes down to measuring the L_2 norm of the differences between the fit and the data points $\{x_i, y_i\}$. The book does the derivation for us, but we can write the result here. If we have:

$$\chi^2 = \sum_{i=1}^{N} (mx_i + c - y_i)^2$$

Then finding a minimum of this measure gives us values for m and c, the slope and y-intercept of our fit.

$$m = \frac{E_{xy} - E_x E_y}{E_{xx} - E_x^2}, c = \frac{E_{xx}E_y - E_x E_{xy}}{E_{xx} - E_x^2}$$

Where the values E_x, E_y, E_{xx}, E_{xy} are defined as follows:

$E_x = \frac{1}{N}\sum_{i=1}^{N} x_i$

$E_y = \frac{1}{N}\sum_{i=1}^{N} y_i$

$E_{xx} = \frac{1}{N}\sum_{i=1}^{N} x_i^2$

$E_{xy} = \frac{1}{N}\sum_{i=1}^{N} x_i y_i$

The goal of this problem is to calculate these values given a dataset, which happens to come from a photoelectric effect experiment. Then, using our linear fit, we can calculate a value of the Planck constant. This will be discussed in the *Experiment* section.

B. Code

The code contains three functions. The most elementary one is called *readfile*, and simply calls the numpy function *loadtxt* with the parameter "millikan.txt", and then slices the result and spits it out as one x and one y array.

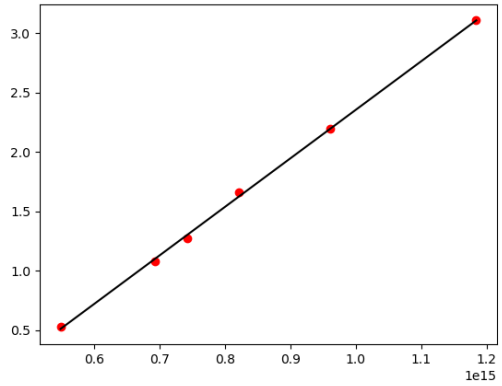
The second function is called *calce*, and calculates all four of the E quantities that are defined above, given an x and a y array.

The third and final function is called *program*, and does the following: it calls *readfile*, plots the data, then calls *calce*, and uses the relevant quantities to calculate the slope and y-intercept of the fit, and then plots that on top of the data.

Finally, *program* calculates h given what we are told in the problem, and calculates the percentage error with respect to the known value, which is hardcoded. This will be discussed in the *Experiment* section.

C. Results

Here is the fit and the datapoints, as spit out by *program()*:



The values we get for the slope and y-intercept are:

$m = 4.08822735852e-15, c = -1.73123580398.$

D. Experiment

In the problem, we are told that this data comes from Millikan's photoelectric effect, where the y values are voltage and the x axis is frequency. Quantum theory of the photoelectric effect predicts that the data should look like:

$$V = \frac{h}{e}\nu - \phi$$

Where ν is the frequency , ϕ is the work function of the material, e is the electron charge, and h is planck's constant. The accepted value of h is $6.62607004 \times 10^{-34} m^2 kg s^{-1}$.

We calculated h from our fit by taking the slope we found, and multiplying it by the accepted value of the electron charge, which is $1.6 \times 10^{-19} C$. Then, we get a value which is within $\approx 1\%$ of the true result.