# Mandelbrot Set with Python

Pratixan Sarmah, Jagiellonian University

September 2022

## 1 Introduction

The Mandelbrot set is a set of complex numbers following a set of recursion relation that has a unique and distinctive pattern when plotted on the complex plane. It is a very popular pattern which also depicts the idea of fractals i.e patterns that repeat itself to infinity. The mathematical equation governing this set of complex numbers is -

$$z_{i+1} = z_i^2 + c \tag{1}$$

where z and c are complex numbers and $z_0 = 0$.
For each c in the complex number, the complex number z either diverges to infinity or converges. Thus the set of all complex numbers c which converge make up the unique pattern on the complex plane and is called the Mandelbrot Set.

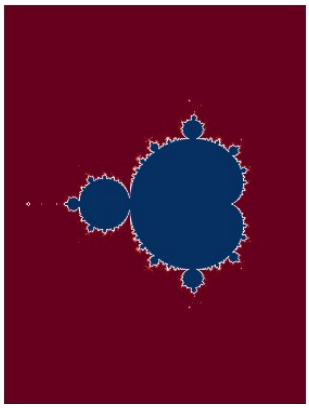


Figure 1: An example of the Mandelbrot set

## 2 Generating the Mandelbrot set with Python

In this section I list out the bits of codes which make up the python program to generate the Mandelbrot set and plot it. At first the necessary libraries are imported.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
import sys
import argparse
```

## 2.1 The Recursion Relation

The recursion relation can be coded as a function in Python as -

```
def z(i,C):
    if i==0:
        return 0
    else:
        return z(i-1,C)**2 + C
```

Here the function z(i,C) takes in two arguments i and C where i is the index of the complex numbers z and C is the complex number we want to check for convergence or divergence.

## 2.2 Generating a Grid

The next step is to generate a grid of complex number points. These points will be used as inputs to the function z(i,C) as C and hence generate the set to be plotted. The function **complex_grid** is defined to do this as shown -

```
def complex_grid(xmin, xmax, ymin, ymax, grid_points):
    Re = np.linspace(xmin, xmax, int((xmax-xmin)*grid_points))
    Im = np.linspace(ymin, ymax, int((ymax-ymin)*grid_points))
    return Re[np.newaxis,:]+Im[:,np.newaxis]*1j
```

The function takes as input the boundary points in the x-axis and the y-axis and the number of points to be generated within the area. It then returns a complex plane with all the points which can be then used to check convergence.

## 2.3 Checking convergence

The convergence of the complex number can be checked using a function as shown.

```
def converge(c,num_iter):
    z = 0 #initial point
    j = 0
    while j<=num_iter:
        z = z**2 + c
        j+= 1
    return abs(z) <= 2
```

Here the function takes as input two arguments, c and num_iter. c is the complex number we want to check for convergence and num_iter is the number of iterations we want the function to run. It then returns those values of z which have a modulus of less than or equal to 2.

### 2.4 Plotting

For plotting the matplotlib.pyplot library is used with a color map of binary i.e black and white. Black for the points that converge and white for that don't. The plots are then saved as a JPG file.

```
plt.imshow(converge(c,args.N), cmap = "binary")
plt.axis("off")
plt.title("Mandelbrot Set")
plt.savefig("Mandelbrot.jpg")
```

## 3 Running the Program

The program takes in two arguments from the user i.e the number of iterations the code should run and the number of grid points it should generate. These are passed to the program as flags in the command line as shown below. The more number of iterations gives more distinct fractals and the more number of grid points give better resolution. This is shown in Fig. 3 and Fig .
To run -

```
$python mandelbrot.py -N 50 -d 512
```

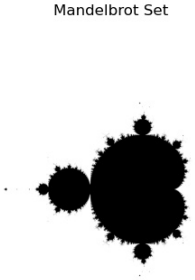
Here, The flag for the number of iterations is -N and the flag for the grid points is -d.

# 4 Results


(a) N = 5

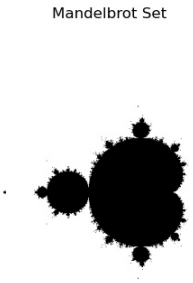

(b) N = 10


(c) N = 20

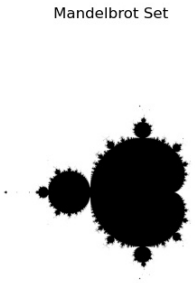



(d) N = 50

Figure 2: Plots of Mandelbrot set with fixed number of grid points at 1024 and different number of iterations (N).




(a) Grid Points = 128




(b) Grid points = 1024

Figure 3: Plots of Mandelbrot set with fixed number of 50 iterations and different number of grid points.