PS-2

Leen Alrawas

September 15, 2023

Abstract

Problem set 2 includes problems on numerics and how numbers are represented in the memory, iterations, efficiency comparisons, plots, recursive relations, round-off errors, and unit tests.

1 Answers and Computational Methods

1. In NumPy float32, numbers are broken for 32-bits: bit-31 for the sign (0 for positive and 1 for negative), bits 30-23 for the 8-bit integer exponent e of 2 in 2^{e-127}, and bits 22-0 for the mantissa.
   Now taking 100.98763 as an example:
   The highest power of two that gives a number less than 100.98763 is $6 = e - 127$, giving $e = 133$. In binary representation that is `10000101`.
   Dividing our number by 2^6 gives 1.57793171875. The mantissa in the binary form is `10010011111100110101010`.
   Hence the IEEE form is

   ```
   0 10000101 10010011111100110101010
   ```

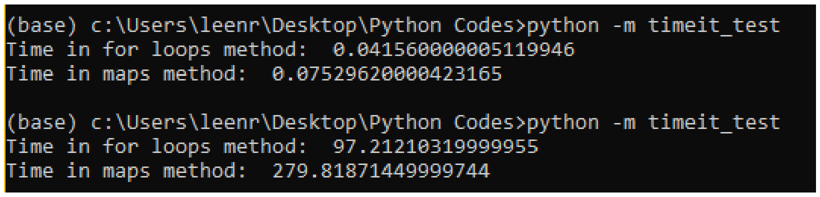
   We can convert this back to a float manually and compare it with the original.
   The zero means a positive sign, the next 8 bits give $e = 133$ so we have a factor of 2^6.
   With the decimal fractions, we get the number 100.98762512207031 which gives an error in the order of 10^{-4}. The calculations to obtain the decimal fractions from the IEEE form were done using Python float64. The order of the error agrees with the one calculated using Python float32.

2. Two methods were used to implement the series: one uses for loops and another uses maps as explained in the code script. The efficiency test was done in a separate script. Using $L = 200$, the Madelung constant was found to be

   $$M = -1.74$$

   Surprisingly, the results indicate that the for loop was considerably faster compared to the chosen Map function (Fig-1).

3. The function was implemented as a recursive relation. It gives True if the number belongs to the Mandelbrot set and False otherwise. A plot for 50,000 random points was generated (Fig-2).



```
(base) c:\Users\leenr\Desktop\Python Codes>python -m timeit_test
Time in for loops method:  0.041560000005119946
Time in maps method:  0.07529620000423165

(base) c:\Users\leenr\Desktop\Python Codes>python -m timeit_test
Time in for loops method:  97.21210319999955
Time in maps method:  279.81871449999744
```

Figure 1: Time comparison between two implementations for finding the NaCl Madelung constant: one uses "for looping" over the integers {i,j,k}, and another creates all possible combinations of them in an array then maps it to the potential function.

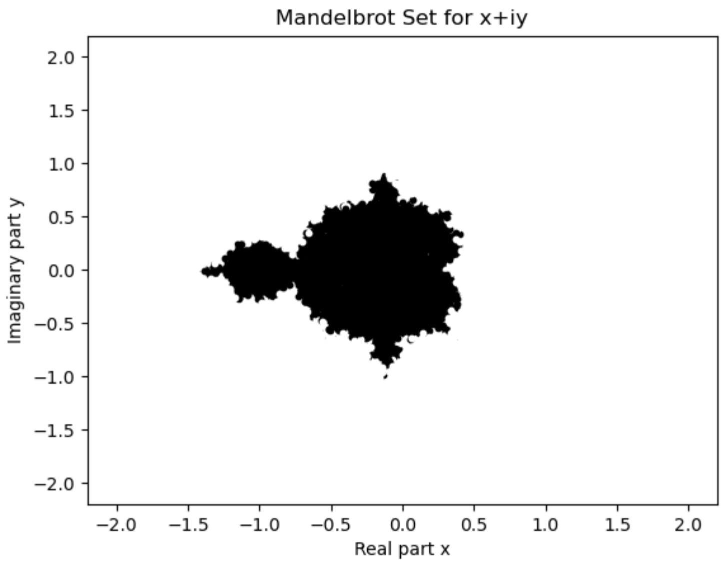


Figure 2: 50,000 random points of the Mandelbrot Set in the range $[-2, 2] \times [-2, 2]$.

4. Since we can only express numbers on the computer to a certain accuracy, problems can occur if we try to subtract two numbers of approximately the same order or if we try to add two numbers with different orders of magnitude. This is exactly what happens when we perform the calculation $-b + \sqrt{(b^2 - 4ac)}$, we are trying to subtract two very close numbers (having b=1000, a=c=0.001). Instead, we can reduce the error by writing the first solution $s_1 = -b - \sqrt{(b^2 - 4ac)}$ and expressing the other one as a function of the first $s_2 = \frac{c}{a} s_1$. A similar approach can be followed for other cases of the orders of b and $\sqrt{(b^2 - 4ac)}$ (implemented in the function `quadratic`). The unit test was applied and showed no error messages.

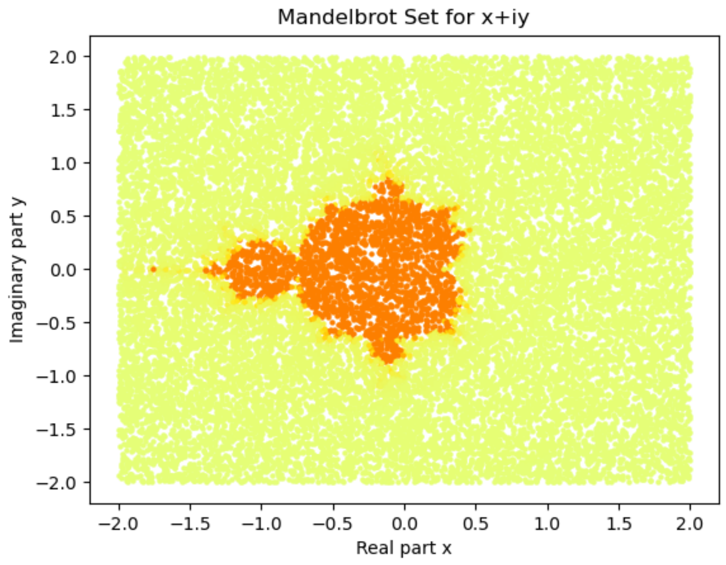


Figure 3: 20,000 random points of the Mandelbrot Set in the range $[-2, 2] \times [-2, 2]$.