PHYS-UA 210 Homework 1

Alex Edwards

September 18, 2019

1 Decimal To Binary

This problem required finding the binary value of the decimal number 121. A simple function was written based on the standard method of finding the binary equivalent of a number. The function works as follows:

Algorithm 1 Determine Binary Representation

while $N \geq 1$ **do**
  store value N % 2 in a list
  $N \leftarrow \lfloor \frac{N}{2} \rfloor$
end while
reverse list order
return list

2 Madelung Constant

Introduction

An ion in a crystal lattice has some electric potential V from the rest surrounding ions in the crystal. The value of V can be determined by approximating each surrounding ion as a point charge and summing over all ions. The Madelung constant M_i is associated with a particular ion in a given crystal configuration, and is used to determine the electric potential for the i^{th} ion in a crystal.

The electric potential V_{ab} of ion a acting on ion b is inversely related to the distance between a and b as:

$$V \propto \frac{1}{r_{ab}}$$

As such, assuming distance is uniform between axis (meaning the lattice is symmetrical) we can express the position of a given ion as a function of the indices of its position in the lattice:

$$r_{ijk} = c\sqrt{i^2 + j^2 + k^2}$$

where c is some constant. As such, for a given ion at i, j, k position in a lattice:

$$V \propto \frac{1}{\sqrt{i^2 + j^2 + k^2}}$$

For a uniform sodium chloride crystal of infinite size, we can define the Madelung constant to take the form:

$$M = \sum_{i,j,k=-\infty}^{\infty} \frac{\pm 1}{\sqrt{i^2 + j^2 + k^2}}$$

where $i \neq j \neq k \neq 0$ and the sign depends on whether the ion at the ijk index is positive or negatively charged. In NaCl crystals, each positive sodium ion is surrounded by 4 neighbouring negative chlorine ions (and vice versa). As such, this creates an alternating positive-negative structure, so the sign at ijk can be determined by: $-1^{|i|+|j|+|k|}$. This gives the final equation determining the Madelung constant for sodium chloride:

$$M = \sum_{i,j,k=-\infty}^{\infty} \frac{-1^{|i|+|j|+|k|}}{\sqrt{i^2 + j^2 + k^2}}$$

Methods

As the Madelung constant approaches zero for large i, j, k we can approximate M by setting the limits of the triple summation to some large value $N \approx 100$.
The first method used to approximate M depended on a triple for loop, successively adding each value. This method had a runtime of 7.87 s ± 556 ms.

The second method made use of numpy's inbuilt $fromfunction$. fromfunction performs a function on a given set, for instance transforming an input of $f(n), N)$ into a 1-d matrix where each element is the value given by $f(i), i \in [0...N]$. In this case we passed three values to obtain a 3-d array, where

$$A[i,j,k] = \frac{-1^{|i|+|j|+|k|}}{\sqrt{i^2+j^2+k^2}}$$

As the values passed run from $[0, N]$ the values first had to be shifted to encompass negative values by $i = \lfloor \frac{N}{2} \rfloor - i$. If the values had not been shifted, only the ions in the first octant (where $i, j, k \geq 0$ would have been accounted for. However, this would only allow the indices to run from 0 to $\frac{N}{2}$, so N was double to allow the full range of values. These values could then be summed over to obtain an approximation of the Madelung constant. For the value $N = 100$ this took had a runtime of 813 ms $\pm$ 25.9 ms.

Discussion

As expected, the latter method invoking python's inbuilt function vastly beat triple for loops. This method took approximately $\frac{1}{10}$ of the time, with the same results (to 11 decimal places). The first method returned a value of 1.7475645950377328, and the second method returned 1.7475645950341998. These values agree with the literature given values of 1.748. fromfunction must rely on array-based optimisations, allowing it such significantly superior performance.

3 Mandelbrot

Introduction

The Mandelbrot set is the set of complex numbers $z = a + ib$ such that z does not exceed $[-2, 2]$ when z is iteratively defined as:

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

for some complex number $c = a_c + ib_c$. The Mandelbrot set is obtained by allowing the parameters for c to vary as $a_c, b_c \in [-2, 2]$. The initial value of z is 0.

When graphed on the complex plane, the Mandelbrot set is a fractal, displaying an infinite amount of detail on the border regardless of magnification.

Methods

The total set of c used was generated by using a range

$$R = -2, -2 + \delta, ..., 2 - \delta, 2$$

where δ was set to be $\frac{1}{250}$ making $N = 1000$ data points. 100,000 c values were generated by combining two sets of R as

$$c = \begin{bmatrix} R_0 + iR_0 & R_0 + iR_2 & R_0 + iR_3 & \dots & R_0 + iR_N \\ R_1 + iR_0 & R_1 + iR_2 & R_1 + iR_3 & \dots & R_1 + iR_N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ R_N + iR_0 & R_N + iR_2 & R_N + iR_3 & \dots & R_N + iR_N \end{bmatrix}$$

This was done using the meshgrid function from numpy. The Mandelbrot definition was then encoded as a function and vectorised, to allow the function to work on R without using a for loop. The function deemed a value c to be in the Mandelbrot set if it still had not exceeded the range after 100 iterations. The log of the number of iterations needed before breaking (or reaching the end of the cycle) was save for each point and then graphed onto a heat map to obtain the Mandelbrot set visualisation. The log was used to allow finer detail to be seen.

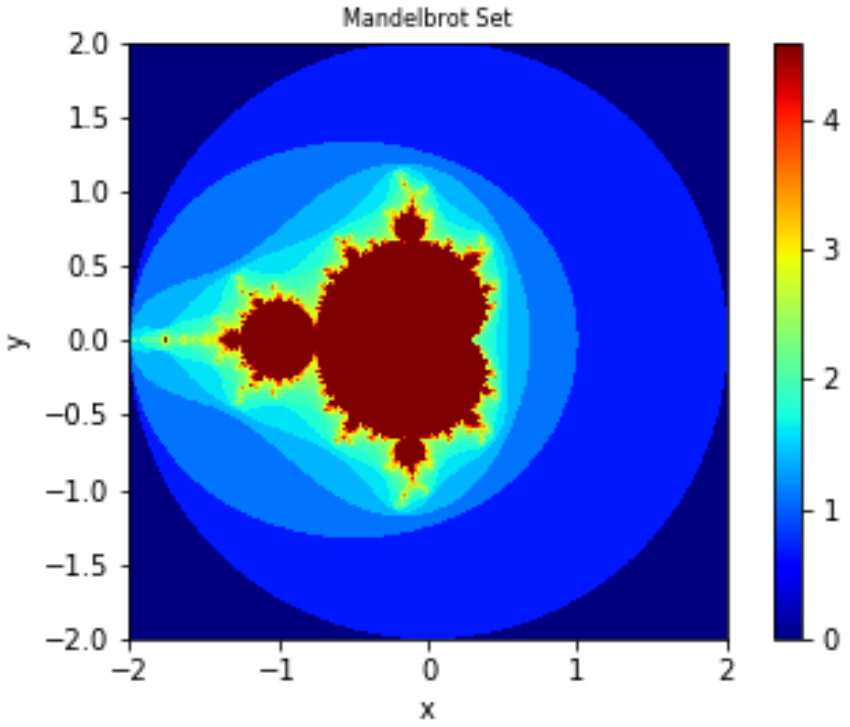
Discussion

The vectorised function sped up runtime by an order of magnitude, again displaying how good numpy's inbuilt functions are. Originally the values were saved and iterated through with for loops, but vectorization of the function sped up run time significantly.

The log function was useful as much of the fine detail occur in the lower levels of iterations so

if the set was graphed linearly those values would have been diluted.

I also found it interesting to change the exponential values for the Mandelbrot function. I added a few interesting ones to the appendix!



4 Quadratic Equations

Introduction The classic solutions to quadratic equations can always be found as

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

However, this causes issue when computers evaluate this expression when:

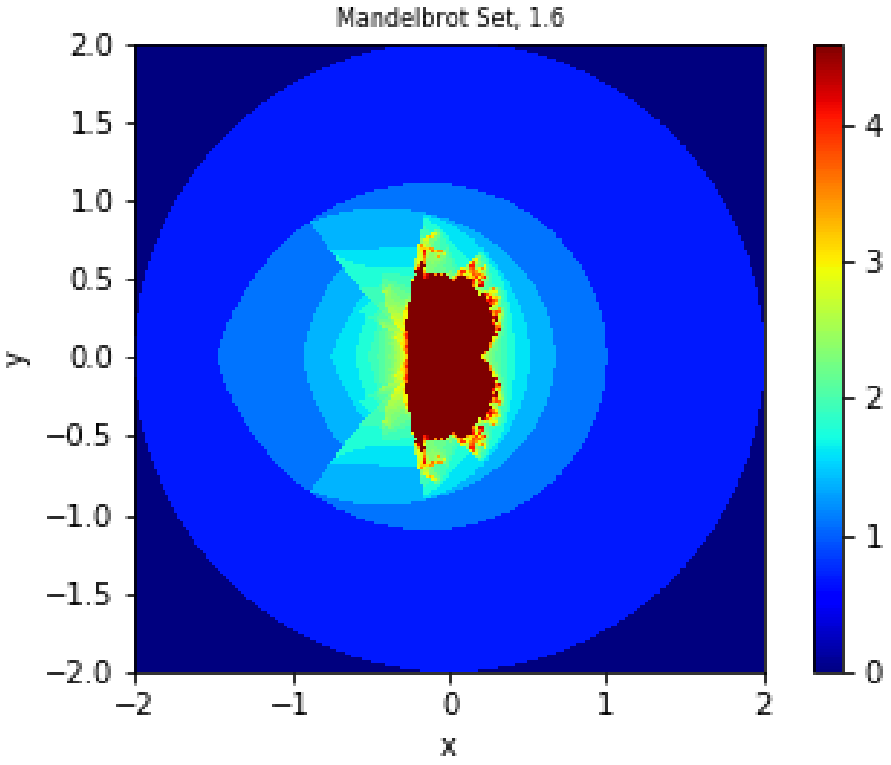
$$b \approx \sqrt{b^2 - 4ac}$$

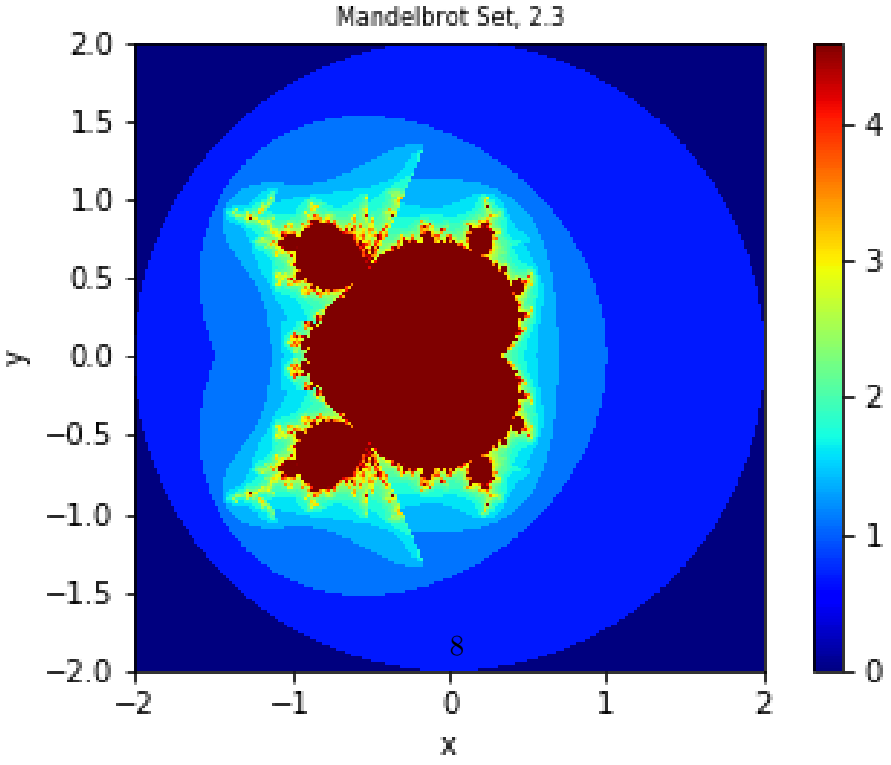
which occurs when $ac << b$. This is because one of the roots depends on the difference of very large, approximately similar values and cannot be precise enough. As such, we can use the Citardauq formula which eliminates this problem.

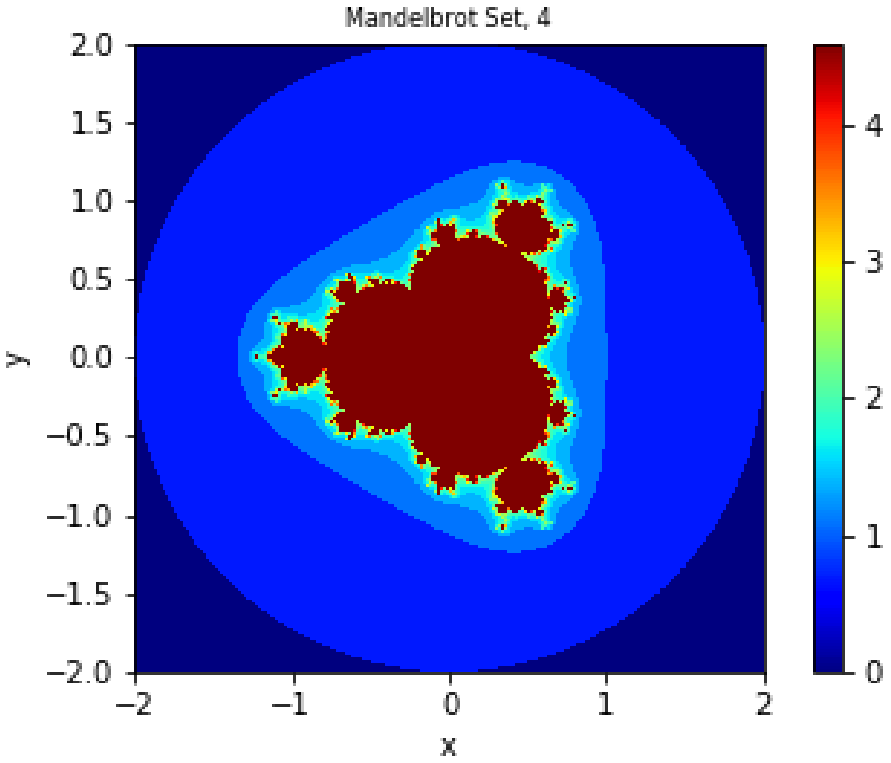
$$x = \frac{-2c}{-b \mp \sqrt{b^2 - 4ac}}$$

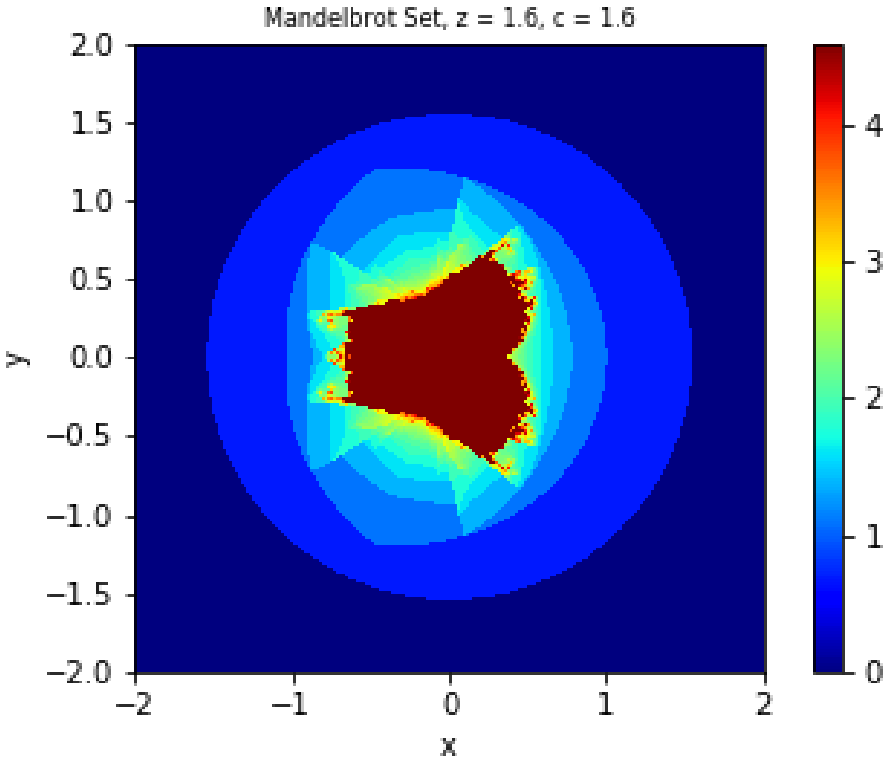
Method and Discussion The formulae were coded and their results compared those of the same expression from Wolfram Alpha. The results indicated that both solutions had one incorrect root, so by using one equation for one root and the other for the second, the correct solution could be found.

5 Appendix


Mandelbrot Set, 1.6
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
y
−2
−1
0
1
2
x
4
3
2
1
0


Mandelbrot Set, 2.3
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
y
−2
−1
0
1
2
x
4
3
2
1
0


Mandelbrot Set, 4
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
y
−2
−1
0
1
2
x
4
3
2
1
0


Mandelbrot Set, z = 1.6, c = 1.6
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
y
−2
−1
0
1
2
x
4
3
2
1
0