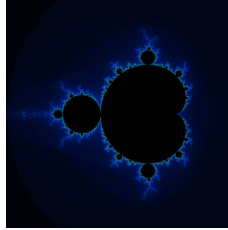


CTAC 2006 Workshop: Python in Scientific Computing

Exercise 2 - The Mandelbrot set

Ole Nielsen, Geoscience Australia and Australian National University



1 The Mandelbrot set

Fractals are non-regular geometric shapes that have the same degree of non-regularity on all scales. Mathematicians have attempted to describe fractal shapes for over one hundred years, but with the processing power and imaging abilities of modern computers, fractals have enjoyed a new popularity because they can be digitally rendered and explored in all of their fascinating beauty.

The most famous fractal is probably generated by the Mandelbrot set. The fractal structure is found at the periphery of a certain set of points in the complex plane. The set is defined by the iteration

$$\begin{aligned}z_{k+1} &= z_k^2 + c, \quad k = 0, 1, \dots, \text{kmax} - 1 \\z_0 &= 0 + 0i\end{aligned}$$

for a range of complex numbers c . The iteration continues as long as

$$k < \text{kmax} \text{ and } |z_k| \leq 2$$

The numbers c for which $|z_k| \leq 2$ for all $k < \text{kmax}$ are said to belong to the Mandelbrot set. The set is usually visualised by assigning different colours to the points based on how many iterations (k) they take. This exercise aims to write a sequential Python program for computing and visualising the Mandelbrot set.

2 Prerequisites

Download the following program files from <http://datamining.anu.edu.au/~ole/teaching/summerschool2005/prerequisites2> using either a browser or the `wget` command.

- `mandelplot.py`: A plotting routine mapping iteration counts into colours.
- `mandelplot_ext.c`: The c-extension for plotting.
- `compile.py`: Python script for cross-platform compilation of c-extensions.
- `test_calculate_point.py`: Test suite for computation of one iteration.
- `test_calculate_region.py`: Test suite for computation of a region.

When files are download, compile the plotting routine by issuing the command:

```
python compile.py mandelplot_ext.c
```

Please ignore the warnings about unresolved symbols.

3 The Mandelbrot program

We are now ready to implement the mandelbrot computation.

3.1 Computational routine

Create a file named `mandelbrot.py` and paste in the following code while making sure you use the automatic indentation facilities and color codings of your editor. The source code is available in the `html` version of this exercise at <http://datamining.anu.edu.au/~ole/work/teaching/summerschool2005/exercise2.html>.

```
"""Fundamentals for computing the Mandelbrot set
"""

def calculate_point(c, kmax):
    """Calculate one point of the Mandelbrot set by iterating the
    governing equation

        z_{k+1} = z_k^2 + c,  k = 0, 1, ... kmax-1
        z_0 = 0 + 0i

    for as long as |z_k| <= 2 and k < kmax
```

```

Inputs:
    c:    A complex number for which the iteration is computed
    kmax: The maximal number of iterations

Output:
    count: The value of k after the iteration has completed.
    """

z = complex(0,0) #Create complex number with initial value
k = 0           #Initialise iteration counter

#
#Your code goes in here
#

return k

```

3.1.1 Exercise:

Complete the function to compute the specified Mandelbrot iteration using the following hints:

- The modulus (or length) of the complex number is computed in Python using `abs(z)`. Similarly, all other operations such as `+`, `-`, `*`, `/` work as expected for complex numbers.
- There is no need to keep track of all values of z_k . Hence z should always keep the latest value by overwriting previous values.
- You can use the boolean operator `and` to combine boolean expressions.
- The code snippet can be done in three lines.

The correctness must be verified by successfully running the provided test harness:

```
python test_calculate_point.py
```

3.2 Calculate all points in region

Now we need an outer loop to calculate all points in a specified region making use of the routine we just completed. Paste the following function into the file `mandelbrot.py`

```

def calculate_region(real_min, real_max, imag_min, imag_max, kmax, M, N):
    """Calculate the mandelbrot set in a given rectangular subset of the complex plane.
    Inputs:
        real_min: Left boundary
        real_max: Right boundary
        imag_min: Lower boundary
        imag_max: Upper boundary
        kmax: Maximal iteration count
        M: Number of points along the real axis
        N: Number of points along the imaginary axis

    Output:
        A: M by N integer matrix containing iteration counts for each point
        the complex region.
    """

    from Numeric import zeros
    A = zeros((M, N)) # Create M x N matrix

    #Compute Mandelbrot iteration for each point in the rectangular subset
    #and store iteration counts in matrix A

    #
    #Your code goes in here
    #

    return A

```

3.2.1 Exercise:

Complete the function using the following hints:

- Work out coordinates for each point in the set such that there are M points on the real axis

$$x = i * (\text{real_max} - \text{real_min}) / M + \text{real_min}, \text{ for } i = 0, 1, \dots, M - 1$$

and N points on the imaginary axis

$$y = j * (\text{imag_max} - \text{imag_min}) / N + \text{imag_min}, \text{ for } j = 0, 1, \dots, N - 1$$

Notice that the lower boundaries are included but not the upper boundaries.

- Create a complex number for each point in the region. In Python a complex number is created as `c = complex(x, y)` where x is the real part and y the imaginary part.

- The iteration count for point i, j should be stored in $A[i,j]$
- The code snippet can be done in about six lines (plus comments).

The correctness must be verified by successfully running the provided test harness:

```
python test_calculate_region.py
```

3.3 Sequential mandelbrot program

Create a file called `mandel_sequential.py` with the following contents:

```
"""Sequential program computing the Mandelbrot set.
"""

from mandelbrot import calculate_region
from mandelplot import plot

kmax = 2**8 # Maximal number of iterations (=number of colors)
M = N = 200 # width = height = N (200 is a good number)

# Region in complex plane [-2,1] x [-1.5,1.5]
real_min = -2.0
real_max = 1.0
imag_min = -1.5
imag_max = 1.5

# Compute Mandelbrot set
import time
t0 = time.time()
A = calculate_region(real_min, real_max, imag_min, imag_max, kmax, M, N)
print 'Computed region in %.2f seconds' %(time.time()-t0)
```

Running this program should yield something like the following output:

```
Computed region in 8.11 seconds
```

3.4 Plotting the result

Computing the Mandelbrot set without viewing the result is not very interesting, so we need a plotting routine for viewing the contents of the matrix `A`. The provided module `mandelplot.py` contains a suitable code for plotting `A` and can be called by adding the line

```
plot(A, kmax)
```

to your main program.

You should see an image looking like this:

