Computational Physics Weekly Assessment

Week 8: chaos and fractals

Weekly assessment task & hints

When finding the roots of a complex polynomial using the Newton-Raphson method there is often a chaotic relationship between the seed point and the root found by NR within certain regions of the complex plane.

Task: Consider using NR to find the roots of the complex polynomial '`z`^{4}`-1=0`', where z is a point on the complex plane, i.e. `z=x+iy`. Demonstrate the chaotic nature of NR by visualising the root found for a range of seed points in a suitable region of the complex plane. Do this by generating one or more subplots covering regions of the complex plane where each point in the plot has a colour or grey-scale value that corresponds to the root found when seeding NR with that point. The identified root is not the only chaotic quantity associated with the seed point - the number of iterations required to identify a root to some pre-defined accuracy is also chaotic. This iteration count may take on many more values than the root, and therefore makes for a smoother, more graduated plot. Accompany each subplot of identified root with a corresponding subplot showing the chaotic nature of this 'convergence time'

Hint: **Use the analytic derivative**

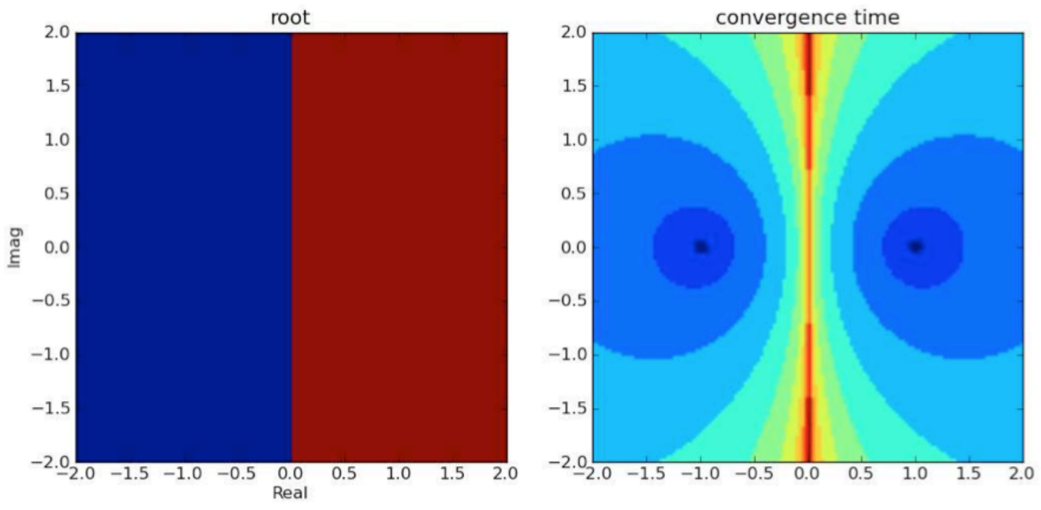
Hint: As some of the roots are imaginary numbers and some are real, you can't directly plot them as colours. You may wish to consider either enumerating the roots or using their argument for visualisation. Enumerating the roots refers to assigning each root a different number (e.g. 1+0i becomes 0, 0+1i becomes 1, another root becomes 2 etc.) and plotting these assigned numbers instead of the roots. To find the argument of a complex number *z* use numpy.angle(*z*).

Hint: Develop and test your NR function prior to the visualisation. Assure yourself that it is working correctly with some simple test cases, perhaps with a real valued input to the polynomial function.

Hint: The variable – z – this week is **not** a numpy array of (x, y), it is a single, complex number made by taking 'x' as the real part and 'y' as the imaginary part.

Hint: This is similar to the Mandelbrot 'escape time' example from the lecture This week's assessment is quite small, requiring perhaps 40-50 lines of code.

Example: Consider finding the roots of the equation 'z^2-1=0' which has the roots z=1+0j and z=-1+0j. Using various regularly spaced points in the complex plane we invoke the NR method and encode the identified root as a number (1+0j becomes '0' and -1+0j becomes '1'). We store these numbers in a 2d numpy array whose elements correspond to positions on the complex plane. This array is visualised with pyplot.imshow. We also keep track of how many iterations are required to evaluate the location of the root to some precision (you may look at either the change in magnitude of the position between iterations or the proximity of f(z) to 0) and make a second plot of the required iteration count. These plots are shown below. The left plot is remarkably dull for our second order polynomial and the right hand plot demonstrates the quadratic convergence time of the NR method. Perhaps your graphs for the fourth order polynomial will be more interesting.



Outputs:

1. When run, your script should produce a single matplotlib figure that illustrates the chaotic nature of both the identified root (left) and convergence time (right) of the NR method for the equation 'z^4-1=0'. You may find it useful to separately plot more than one 'zoom level'

2. A single variable, ANSWER1, in which you give a paragraph explaining how your diagrams demonstrate the chaotic behavior of the NR method and why the images in the diagrams are said to be fractal in nature.

3. The instructions given for figure layout etc. are guidelines, and if you wish to produce your figure differently that is fine – as long as, combined with your answer, it demonstrates the chaotic and fractal aspects of the NR method. You may for example consider a single image where colour corresponds to the identified root and intensity corresponds to the convergence time.

4. Important: Only call pyplot.figure() and pyplot.show() once.

General comments:

- Place two variables at the start of the code, USER=”your name” and USER_ID = “your CIS login”
- Your module must run in order to be awarded any marks
- Your solution should contain no more than about 95 lines of code, comments and whitespace. Excessively long submissions may be penalized.