MAS212 Assignment #1 (2019): The Newton-Raphson method and fractals

Introduction: The *Newton-Raphson (NR) method* is an iterative method for finding roots of differentiable functions. (Recall that $\bar{x}$ is a root of a function f iff $f(\bar{x}) = 0$.)

Starting with an initial value x_0, the NR method generates a sequence $x_0, x_1, x_2, x_3, \ldots$ by repeated application of an iterative step:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}, \tag{1}$$

where $f' \equiv \frac{df}{dx}$. The limit of the sequence, if it exists, is a root of $f(x)$.

In practice, we say the method *converges* if $|x_{i+1} - x_i| < \varepsilon$ for some $i < i_{\max}$, where ε is a suitably small positive constant greater than machine precision and $i_{\max}$ is the maximum number of iterations. (For example, $\varepsilon = 10^{-12}$ and $i_{\max} = 20$.)

In this assignment we will use the Newton-Raphson method to find the roots of non-linear functions such as $Axe^{-x} - 1$ and $z^3 - 1$. Your completed assignment will consist of **one** `.py` script file. Your files should be submitted at `http://somas-uploads.shef.ac.uk/mas212`. The submission deadline is found on the course website.

Please refer to Appendix A for the preferred format for the file, and to Appendix B for the sample output of your script.

Part 1: Newton-Raphson method

Let

$$f_1(x) = Axe^{-x} - 1,$$

where A is a real parameter ($A \in \mathbb{R}$).

(a) Write the functions below in your `.py` script file. Add a docstring to the beginning of each function to explain its purpose.

```
def f1(x, A=3):
    return A*x*np.exp(-x) - 1

def f1prime(x, A=3):
    return A*(1-x)*np.exp(-x)
```

(b) Write a function `NRmethod` to implement the Newton-Raphson method.

Please refer to the docstring in Appendix A for instructions.

Starting with the initial value $x_0 = 0.0$ and $A = 3$, call `NRmethod` function to find a root. Print the root $0.619\ldots$ to 10 decimal places. How many steps were required to find the root to this accuracy?

Find a second root of $f_1(x)$ to the same accuracy, by calling your function with a different starting value x_0.

(c) The NR method is *not* guaranteed to converge to a root. Find a starting value x_0 for which the NR method fails. Explain in your own words what has gone wrong, that is, add a line to your code giving an explanation:
`print("The method fails for x0 = ...  because ...")`).

(d) The number of real roots of $f_1(x)$ depends on the value of A. For $A > e$ there are two roots; for $A = e$ there is one root; and for $A < e$ there are no roots. (Here $e = \exp(1)$).

Print the sequence of estimates for $A = e$ starting with $x_0 = 0$. How close is the final value x_{20} to the real root $\bar{x}$?

The NR method appears to be converging "more slowly" in this case. Explain why the convergence is slow in this case, in your own words:
`print("The NR method converges more slowly for A=e because ...")`

Part 2: Cubic roots of unity and fractals

The equation $z^3 = 1$ has three solutions in the complex plane:

$$\bar{z}_1 = 1, \quad \bar{z}_2 = e^{2i\pi/3} = \frac{1}{2}\left(-1 + \sqrt{3}\,i\right), \quad \bar{z}_3 = e^{-2i\pi/3} = \frac{1}{2}\left(-1 - \sqrt{3}\,i\right).$$

Suppose we were to apply the Newton-Raphson method to look for these solutions, by finding the roots of the complex function $f_2(z) = z^3 - 1$. Here we address two questions: Given a starting guess z_0, does the NR method succeed, and if so, which root does it converge upon? [1]

(a) The NR method fails for the starting value $z_0 = 0$. Therefore, it will also fail for starting values z_0 which lead to $z_1 = 0$. By considering the iterative formula (1), find three other starting values for which the NR method fails (print your answers to screen).

(*for discussion only*: How many initial values are there for which the NR method fails? How are they distributed in the complex plane?).

(b) Which root does the NR method converge upon in the following cases?

(i) $z_0 = 0.02$

(ii) $z_0 = 0.02i$

(iii) $z_0 = -0.01 \times (1 + i)$

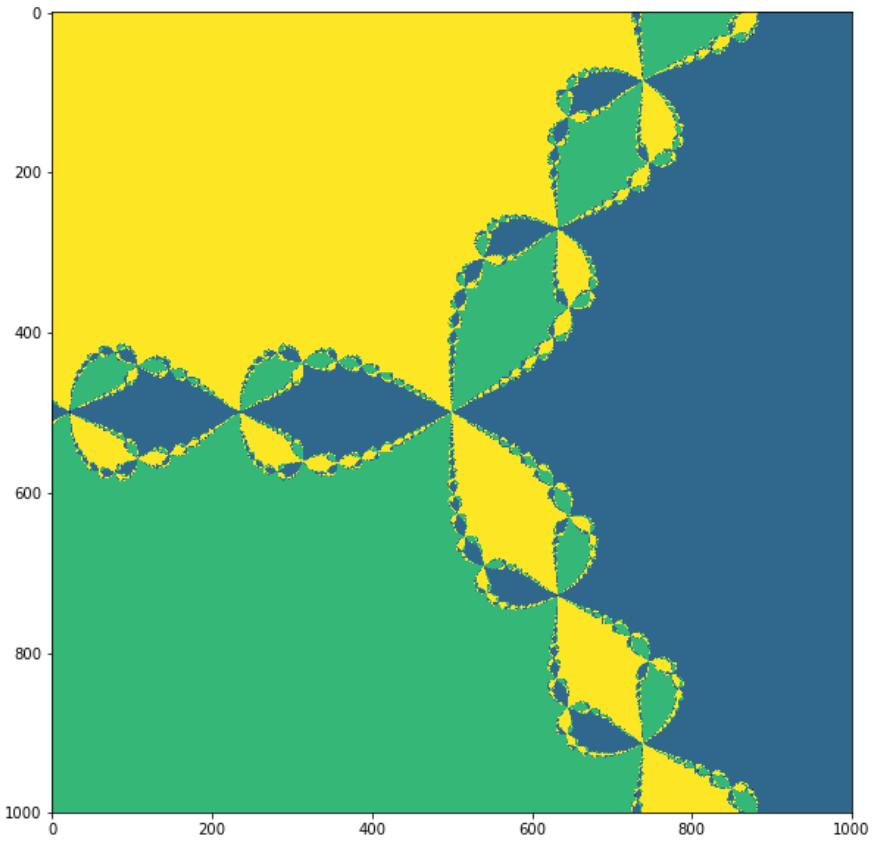
(iv) $z_0 = -2^{-1/3} + 0.001 \times (1 - i)$

Write code to address this question, and re-use the function `NRmethod` from part 1. You may need to use more than 20 iterations.

(c) *Challenging task (***20**% *of mark).* In the corridor outside G18 Hicks is a poster showing the basins of attraction of the three cubic roots of unity in the complex plane. Write a Python code to recreate this image at lower resolution, 300×300, for the domain $\mathrm{Re}(z_0) \in [-a, a]$, $\mathrm{Im}(z_0) \in [-a, a]$ with $a = 1.5$. Your code should save the image as `xxxxxxxxx.png`, where `xxxxxxxxx` is your registration number.

Suggestion: Use a 300×300 `numpy` array, to contain only the integer values $0, 1, 2, 3$. Each entry in the array corresponds to one starting value z_0 in the complex plane. Set an entry to 0 if the NR method did not converge after (e.g.) 20 iterations, 1 if it converged to the real root $\bar{z}_1$, and 2 or 3 if it converged to complex roots $\bar{z}_2$ or $\bar{z}_3$. Then plot this array using `plt.imshow(arr)`. A example image at higher resolution is shown below:

[1] In fact, this question was first considered by Sir Arthur Cayley in 1879.



Guidance:

This assignment will count for $\sim$ 30% of your module mark. This assignment is intended to help you learn as well as to test your skills. If stuck, please don't struggle in silence: talk to your classmates and tutors.

For this assignment, **fair means** include: discussing the assignment with classmates & tutors; making use of Appendix B to check the output of your code; using a code example from this brief; etc. **Unfair means** include (but are not limited to): sharing or distributing files; copying-and-pasting from work that is not your own; passing off other's work as your own; posting your work online, etc. If in doubt, please contact the course tutor.

Note that all submissions will be checked for excessive similarities. Where there is circumstantial evidence of unfair means, I reserve the right to award zero marks for this assignment.

Appendix A: File format. First, put your name and registration number. Next, import any modules you will use. Next, define all the functions used. Finally, call the functions to produce human-readable output for each of the 'tasks'. Please make the output easy to read for the marker running the script (for example, indicate which task is being completed). See the example below.

```
# Sam Dolan. Registration number: xxxxxxxx

# (1) Import modules
import numpy as np
import matplotlib.pyplot as plt

# (2) Define all the functions we will use here
def f1(x, A=3):
    """You should write your docstring here."""
    return A*x*np.exp(-x) - 1

def f1prime(x, A=3):
    """You should write your docstring here."""
    return A*(1-x)*np.exp(-x)

def NRmethod(f, fprime, x0, max_steps=20, tol=1e-12, A=3):
    """
    Seeks a root of a function by applying the Newton-Raphson step
    repeatedly to generate a sequence of estimates.
    The first two arguments are functions that return f(x) and f'(x).
    'x0' is the initial guess.
    The method should stop once either of two conditions is met:
     (i) the absolute difference between successive estimates is
    less than 'tol' (indicating that the sequence has
    converged sufficiently), or,
     (ii) the number of steps exceeds 'max_steps'.
    The function should return a list of the sequence of estimates,
       [x0, x1, x2, ...]
    """
    ...

# (3) Produce nicely-formatted output for each task on the sheet.
print("Part 1(a)")
print("See code: I have added a docstring to each function.")
print("-----")

print("Part 1(b)")
x0 = 0.0
xseq = NRmethod(f1, f1prime, x0)
print("With a starting value of x0=%.2f the NR method converged" \
      + " after %i iterations to:" % (x0, len(xseq)-1))
print("   %.10f" % xseq[-1])
# etc.
```

Appendix B: Example output of your script

```
Part 1(a)
See code: I have added a docstring to each function.
-----
Part 1(b)
With a starting value of x0 = 0.0 and A = 3.0 the NR method converged after 7 iterations to:
   0.6190612867
With a starting value of x0 = 2.0 and A = 3.0 the NR method converged after 5 iterations to:
   .....
-----
Part 1(c)
The method fails for x0 = ... because ...
-----
Part 1(d)
The sequence for A = e is :
 ...

After 20 iterations, the value x20 is 1.82e-06 from the root at x=1

The NR method convergences more slowly for A=e because ...
-----
Part 2(a)
...
```