MAS212 Assignment #1 (2016): The Newton-Raphson method and fractals

Dr. Sam Dolan (s.dolan@sheffield.ac.uk)

Introduction: A root of a function $f(x)$ is a value $\bar{x}$ such that $f(\bar{x}) = 0$. The **Newton-Raphson (NR) method** is an iterative method for finding roots of differentiable functions. Starting with an initial estimate x_0, we generate a sequence $x_0, x_1, x_2, x_3, \ldots$ by repeated application of the 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 $i_{\max} = 25$.)

In this assignment we will use the Newton-Raphson method to find the roots of non-linear functions such as $xe^{-x} - c$ 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: the Newton-Raphson method

(a) Let $f_1(x) = xe^{-x} - 1/10$. Enter the function below in your `.py` script file. Add comments to the function (using `#`) to explain how it works. Add a docstring to the beginning of the function.

```python
def nr_step_f1(x):
    f1 = x*np.exp(-x) - 0.1
    f1x = (1-x)*np.exp(-x)
    return x - f1 / f1x
```

(b) Write a function `findroot` to implement the Newton-Raphson method. Please refer to the docstring in Appendix A for instructions.

Starting with the estimate $x_0 = 0.2$, use your function to find and display the root $0.11183\ldots$ 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) Let $f_2(x) = x + \sin x - \ln x - 2$. Write code to find *four* roots of $f_2(x)$ in the domain $x \in (0, 5]$. Display your results formatted to 10 decimal places.

(d) (*Not assessed, for discussion.*) In part (b) the NR method converged rapidly. To see why, suppose our guess x is already close to the root $\bar{x}$ with $\Delta x = x - \bar{x}$ the error. Now expand $f(x)$ in a Taylor series, $f(x) = 0 + f'(\bar{x})\Delta x + \frac{1}{2}f''(\bar{x})(\Delta x)^2 + \ldots$. If Δx is sufficiently small (i.e. if we are close enough to the root) then we are justified in neglecting the higher-order terms in the series. Now find the derivative $f'(x)$, and insert into the iterative step (1). Assuming that $f'(\bar{x}) \neq 0$, one gets $x_{i+1} = \bar{x} + \frac{f''(\bar{x})}{2f'(\bar{x})}(\Delta x)^2 + \ldots$. Observe that the new error is proportional to the old error *squared*, so it is much smaller. This means that we can get approximately twice as many digits of accuracy after each step, provided that we are sufficiently close to the root for the above argument to hold.

Now consider the limitations of the argument above. For example:

- What can happen if the initial guess is not close to the root?
  Try $x_0 = -20$ in your code for part (b).
- What happens if $f'(\bar{x}) = 0$?
  Consider the root $\bar{x} = 0$ of $f(x) = x\sin(x)$ as an example.
- Does the NR method always converge?
  Consider seeking the root of $f(x) = xe^{-x}$ starting with $x_0 > 1$.

Part 2: Cubic roots of unity and fractals

The equation $z^3 = 1$ has three solutions in the complex plane: $\bar{z}_1 = 1$, $\bar{z}_2 = e^{2i\pi/3} = \frac{1}{2}\left(-1+\sqrt{3}i\right)$ and $\bar{z}_3 = e^{-2i\pi/3} = \frac{1}{2}\left(-1-\sqrt{3}i\right)$. Suppose we were to apply the Newton-Raphson method to look for these solutions, by finding the roots of the complex function $f_3(z) = z^3 - 1$. We may ask the question: given a starting guess z_0, does the NR method succeed, and if so, which root does it converge upon? (In fact, this question was first considered by Sir Arthur Cayley in 1879).

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

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

   (i) $z_0 = 0.02$

   (ii) $z_0 = 0.02i$

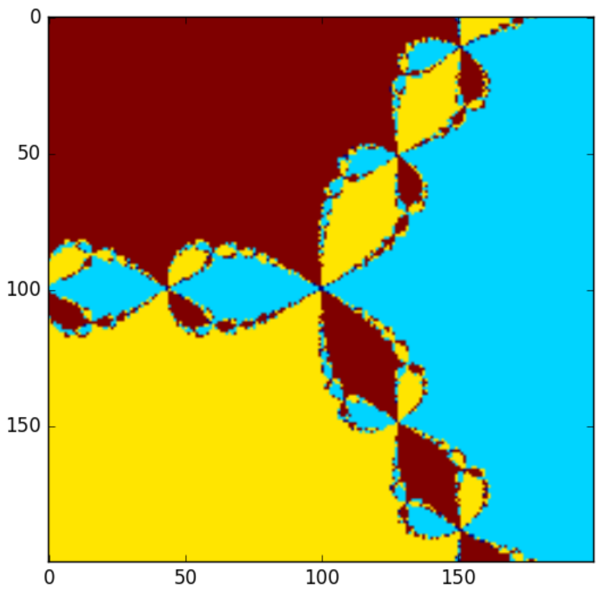
   (iii) $z_0 = -0.01(1+i)$

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

   You may need to use more than 25 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, 200×200, for the domain $\text{Re}(z_0) \in [-a, a]$, $\text{Im}(z_0) \in [-a, a]$ with $a = 1.4$. Your code should save the image as `xxxxxxxxx.png`, where `xxxxxxxxx` is your registration number.

   Suggestion: Use a 200×200 `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.) 25 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)`. An example image is shown below:



Guidance:

This assignment will count for 25%–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. Note that you **may** discuss the assignment in general terms on the online discussion board. However, you **may not** post any code there. 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. Include a docstring at the top of the script file with 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.

```python
"""
Sam Dolan. Registration number: xxxxxxxxx
"""
#####
# (1) Import modules
import numpy as np
import matplotlib.pyplot as plt

#####
# (2) Define all the functions we will use here

def nr_step_f1(x):
    """ Add a docstring here """
    f1 = x*np.exp(-x)  # Add short comments here.
    f1x = (1-x)*np.exp(-x)
    return x - f1/f1x

def findroot(x0, step_func, max_steps=25, tol=1e-12):
    """
    Seeks a root of f1(x) by applying the Newton-Raphson step
    repeatedly to generate a sequence of estimates. Here "x0" is the
    initial guess, and "step_func" is a function (such as nr_step_f1)
    that carries out a single iterative step.
    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 comments and a docstring to the function.")
print("-----")

print("Part 1(b)")
x0 = 0.2
seq = findroot(x0, nr_step_f1)
print("With a starting value of x0=\%.2f the NR method converged" \
      + " after \%i iterations to:" \% (x0, len(seq)-1))
print("   %.10f" \% seq[-1])
# etc.
```

Appendix B: Example output of your script

```
Part 1(a)
See code: I have added comments and a docstring to the function.
-----
Part 1(b)
With a starting value of x0=0.20 the NR method converged after 5 iterations to:
   0.1118325592
With a starting value of x0=2.00 the NR method converged after 6 iterations to:
   3.5771520640
-----
Part 1(c)
The function f2(x) has four roots in the domain (0,5]:
   0.2027145610
   1.2976059913
   3.1317516943
   4.4665495910
-----
Part 1(d)
  This part is for discussion, and is not assessed.
-----
Part 2(a)
...
```