MAS2008 ASSIGNMENT: FINDING ROOTS

1. Introduction

This assignment will ask you to implement, test and compare two methods for finding roots of a function $f(x)$: the Newton-Raphson method and the secant method. To submit your work, you should upload a single `.ipynb` file to Blackboard.

Section 2 below describes the Newton-Raphson method, and gives some details of how you should implement it. Section 3 gives some indications about the secant method, but with fewer details.

Section 4 specifies some test cases. You should run your code on each of these cases and comment on what happens. Further details about the required commentary will be specified in Section 4.

As a reminder, you are free to use web search and/or AI assistants to help you write code. However, you should include a record of such assistance in the file that you submit. Also, you should remember that these methods may provide code that is simply incorrect or that is a correct solution to a different problem than the one that you intended. You are responsible for understanding and testing your code and ensuring that it complies with the specifications in this document.

Marks will be awarded as follows:

- 30% for code. For full credit code should be correct, efficient and clearly structured, with docstrings for all nontrivial functions and comments explaining the main steps of the methods.
- 70% for explanation of the test cases. For full credit you will need to write a significant amount of mathematical working to explain why the algorithms behave in the way that they do in particular cases, and you will need to illustrate your explanations with clear diagrams produced by matplotlib.

On the course web page you will find a sample assignment (of fairly similar style to this one, but with significantly different content). You should take that as a guide to the kind of thing you should be doing.

2. The Newton-Raphson method

The Newton-Raphson method is as follows. Given a function $f(x)$ with derivative $f'(x)$ and an initial point x_0, we define x_n recursively for $n > 0$ by $x_{n+1} = x_n - f(x_n)/f'(x_n)$. Typically (but not always) the sequence (x_n) will converge to a value x_∞ such that $f(x_\infty) = 0$ (so x_∞ is a root of the function $f(x)$). When implementing this on a computer, we typically choose a tolerance $\epsilon > 0$, and we stop when the absolute value of $f(x_n)$ is less than ϵ, returning x_n as an approximate root of $f(x)$.

Write a Python function `newton(a, b, N, epsilon, f, df)` which implements the above method. The arguments should be as follows:

- You can assume that `a` and `b` will be real numbers with $a < b$. Your code should look for roots of $f(x)$ in the interval $[a, b]$. Your initial value for the Newton-Raphson process should be the middle of that interval. If x_n is ever outside that interval, then your code should raise a `RuntimeError` with an appropriate error message, which should in particular include the value of n.
- You can assume that `N` will be a positive integer, and that `epsilon` will be a positive real number. Your code should calculate x_n until either the absolute value of $f(x_n)$ is less than `epsilon`, or $n = N$. In the first case your code should return x_n; in the second case your code should raise a `RuntimeError` with an appropriate error message.
- The arguments `f` and `df` should be Python functions that calculate calculate $f(x)$ and $f'(x)$ respectively. For example, we should be able to find a root of $p(x) = x^2 - 1$ with $x \in [0, 3]$ as follows:

```
def p(x):
    return x * x - 1
def dp(x):
    return 2 * x
newton(0, 3, 100, 1e-8, p, dp)
```

- If no errors occur, your code should return the approximate root that you have found.
- Your code should have a docstring explaining the parameters and return value.
- As your code will be checked in a partially automated way, you should name everything exactly as specified above: your function should be called `newton`, and the parameters should be called `a`, `b`, `N`, `epsilon`, `f` and `df`.

3. The secant method

You should write a Python function `secant(a, b, N, epsilon, f)` which implements the secant method to search for a root of $f(x)$ in the interval $[a, b]$. I will not give details of the secant method here; part of your job is to find one of the many internet sources that will explain it to you. The parameters should be the same as for the Newton-Raphson method, except that the parameter `df` is not required. You should start the secant method with $x_0 = a$ and $x_1 = b$, and you should raise a `RuntimeError` with an appropriate error message if x_n lies outside $[a, b]$ at any stage.

4. Test cases

Run your code for each of the following sets of parameters.

a	b	N	ϵ	$f(x)$	$f'(x)$
0	3	100	$1e-8$	$x^2 - 1$	$2x$
0	3	100	$1e-8$	$e^{-x} - x$	$-e^{-x} - 1$
0	3	100	$1e-8$	$x^2 - 2x + 1.1$	$2x - 2$
0	6	100	$1e-8$	$x^3 - 6x^2 + 12x - 6$	$3x^2 - 12x + 12$
2	18	100	$1e-8$	$\sin(x)$	$\cos(x)$
2	18	100	$1e-8$	$\cos(x)$	$\cos(x)$
0	8	100	$1e-8$	$x^2 - 5x + 13$	$2x - 5$
0	4	100	$1e-8$	$\|x - 3\| - 2$	$2H(x - 3) - 1.$

In the last line, the notation $H(t)$ refers to the Heaviside function, given by

$$H(t) = \begin{cases} 0 & \text{if } t < 0 \\ \frac{1}{2} & \text{if } t = 0 \\ 1 & \text{if } t > 0. \end{cases}$$

You could code this as a Python function yourself, or search for information about an existing implementation.

For each line in the above table, you should run both the Newton-Raphson method and the secant method. Some lines involve various kinds of errors or problems. In cases where your method successfully finds a root in the specified interval, only brief comments are required, but you should compare the performance of the two methods. In cases where there are problems, you should explain in detail what is happening, which may require mathematical analysis and/or discussion of programming technicalities. For each test case you should include a markdown cell in your `.ipynb` notebook, using LaTeX for mathematical expressions and backquotes for Python code. Your explanations should be illustrated using diagrams generated using `matplotlib`. At the most basic level, you could print out the values x_n, or you could define a function like this:

```
def show_newton_step(f, df, x0):
    y0 = f(x0)
    x1 = x0 - y0/df(x0)
    y1 = f(x1)
    xs = np.linspace(min(x0, x1), max(x0, x1), 100)
    ys = f(xs)
    plt.plot(xs, ys, 'r-')
    plt.plot([x0, x1], [y0, y1], 'bo')
```

This defines $x_1 = x_0 - f(x_0)/f'(x_0)$ and displays the points $(x_0, f(x_0))$ and $(x_1, f(x_1))$ together with the graph of f for x between x_0 and x_1. However, much better diagrams are possible; a search for pictures of the Newton-Raphson method will find examples. You should think carefully about what would be the best way to illustrate your explanations. Credit will also be given for well-structured code, which means that you should write general functions like `show_newton_step()` that will work for any f rather then using *ad hoc* code for each test case.

5. Guidance

- You may discuss the assignment with other students, but you must not copy code or text from them. You must write your own notebook in your own words based on your own understanding. You must also mention any collaboration in the acknowledgements section of your notebook, including the names of people with whom you worked.
- You may search the internet for information, but you must mention all sources that you have used in your acknowledgements, with specific URLs.
- You can use code that you find on the internet or that is given to you by an AI assistant, but you must acknowledge it. You must also ensure that the code you submit complies precisely with the notation and terminology used in this briefing document, and that the function names, arguments and return values are exactly as specified. You will probably need to modify code obtained from elsewhere to achieve this.
- All acknowledgements must appear in a separate markdown cell at the top of your notebook, with heading "Acknowledgements".
- All nontrivial functions should have docstrings.
- For all code that implements a nonobvious algorithm, you should add detailed comments in the code to prove that you understand how the algorithm works.
- When developing your notebook, you will probably move backwards and forwards, inserting things and executing code in different places. However, before submission you should tidy up your code. Remove anything that is not needed and check that the rest can be executed in order from the top to the bottom without errors and that this generates all the required plots and prints all the required messages.
- Upload your notebook using Blackboard.
- Do not include your own name or registration number in your notebook. (Blackboard will ensure that your work is tagged with your name at the point when that becomes necessary.)