Programming Problem 4 - newton.py

Directions: Download the template files I have provided on Blackboard. Then open Spyder, load the template file, and write the following program. Submit your source code via Gradescope in `.py` format; do NOT send any other files. READ THE INSTRUCTIONS on how to submit your work in the Course Documents section of Blackboard.

Be sure to read the SPECIFICATIONS carefully! And write comments!

Write a program that solves polynomial equations of up to fifth degree using Newton's method.

You should use the starter code in the template provided that accepts coefficients for c_0, c_1, c_2, c_3, c_4, and c_5, It also accepts a "guess" value, and a threshold for the approximation of a solution using Newton's method.

Your job is to write the code to use Newton's method to solve the equation

$$c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0 = 0.$$

For example, when you run the program, it may look like this:

```
Enter x^0 coefficient: -10
Enter x^1 coefficient: -3
Enter x^2 coefficient: 0
Enter x^3 coefficient: 0
Enter x^4 coefficient: 2
Enter x^5 coefficient: 1
Enter guess: 1.1
Enter threshold for approximation: 0.01
1.428744421467379
It took 4 iterations
```

This is because a solution of $x^5 + 2x^4 - 3x - 10 = 0$ is given by $x \approx 1.428744421467379$, and the given polynomial function evaluated at this value of x is within 0.01 of zero. Also, the number of iterations of Newton's method that it takes to get within this threshold is 4. You can assume that the threshold input will be a positive float, and probably close to zero.

The Newton's method algorithm which discovers this solution is described below.

Recall that Newton's method is a method to approximate solutions to an equation $f(x) = 0$ very quickly. It works by starting with a (more-or-less random) guess x_0 for a solution, and then coming up with better and better approximations x_1, x_2, x_3, etc., using the following process:

$$x_1 = x_0 - f(x_0)/f'(x_0)$$
$$x_2 = x_1 - f(x_1)/f'(x_1)$$
$$x_3 = x_2 - f(x_2)/f'(x_2)$$

and so on and so forth, with $x_{n+1} = x_n - f(x_n)/f'(x_n)$ in general. For reasons that are best explained by a textbook (or me, in person), x_1 will usually be close to a solution, and x_2 will be closer, and x_3 closer still, etc.

Time for an example: Let's try to solve $x^2 - 4x + 1 = 0$. Here, $f(x) = x^2 - 4x + 1$. We start with a guess of $x_0 = 1$.

Then $x_0 = 1$; $f(x_0) = -2$; $f'(x_0) = -2$; and so

$$x_1 = 1 - (-2)/(-2) = 0.$$

Then $x_1 = 0$; $f(x_1) = 1$; $f'(x_1) = -4$; and so

$$x_2 = 0 - (1)/(-4) = 0.25.$$

Then:

$$x_3 = 0.25 - f(0.25)/f'(0.25) = 0.267857142 \text{ (approximately)}.$$

Then:

$$x_4 = 0.267857142 - f(0.267857142)/f'(0.267857142) = 0.267949190 \text{ (approximately)}.$$

And so on. The actual value of one solution to 9 places is 0.267949192, so we were already at 8 decimal place precision after only four iterations – not bad.

A suggestion for an approach you could use (you do not have to use this approach): use a list to store the coefficients for the original polynomial, and then create the corresponding list of coefficients for the derivative. For example, if the original polynomial $2 + x + 3x^2 + 10x^3$ is represented by `[2, 1, 3, 10, 0, 0]`, then the derivative could be represented by `[1, 6, 30, 0, 0, 0]` (that would be $1 + 6x + 30x^2$). Then, you can use the list to evaluate a polynomial at specific x-values (in other words, "plug in" x-values – e.g., when you plug $x = 2$ into $1 + 6x + 30x^2$, you get 133).

Finally, when it comes to actually implementing Newton's method – don't overthink it, that's actually not hard code at all! Hint: you can use the same variable for x_0, x_1, x_2, etc., because as soon as you know the value of, say, x_{12}, you don't need to know the value of x_{11} anymore.

Note that Newton's method sometimes fails for various reasons, and even when it converges, it converges to one specific answer, depending on what x_0 you provide. But it should work well with the samples shown above ($x^5 + 2x^4 - 3x - 10 = 0$ with $x_0 = 1.1$ and $x^2 - 4x + 1 = 0$ with $x_0 = 1$) and the test cases on Gradescope.

Specifications: your program must

- not modify or delete any of code in the template file
- the program should display the value of the approximation (which will usually be close to a solution) and the number of iterations it took for Newton's method to arrive the approximation, given the input threshold.

This problem has one invisible and three visible test cases on Gradescope. Each test case is worth two points, and two points will be manually assigned for following all specifications. Additional points may be deducted for incorrect code or code not following specifications, even if test cases pass.