MTH 3300
Fall 2022

Programming Problem 7 - newton.py (OPTIONAL)

Directions: Download the template file I have provided on Blackboard. Then open Spyder, load this 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 submitting your work in the Course Documents section of Blackboard.

This assignment is *optional* and can be used to replace your lowest programming problem score.

Important note If any of your submissions for this problem are flagged by the plagiarism software or found to use any unauthorized resources, you will not receive any credit for the submission. Additionally, the final exam bonus will NOT be applied to compute your overall grade. (That is, your final exam score will NOT be used to replace half of your lowest midterm score, provided your final exam score is higher.) This assignment is the intellectual property of Baruch College's MTH 3300 course. It is illegal to share it, in part or whole, with anyone outside our current course, including posting on the internet. As stated in our course policy, you may be ineligible for any favorable changes to the grading criteria if you are involved in academic dishonesty.

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

Using Newton's method, write a program that solves polynomial equations up to the seventh degree.

You should use the starter code in the template provided that accepts coefficients for c_0, c_1, c_2, c_3, c_4, c_5, c_6, and c_7. It also takes a "guess" value and the number of iterations for Newton's method.

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

$$c_7x^7 + c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0 = 0.$$

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

```
x^0 coefficient: -10
x^1 coefficient: -3
x^2 coefficient: 0
x^3 coefficient: 0
x^4 coefficient: 2
x^5 coefficient: 1
x^6 coefficient: 0
x^7 coefficient: 0
guess x_0: 1.1
number of iterations: 10
an approximate solution is 1.42871599796
```

This is because a solution of $x^5 + 2x^4 - 3x - 10 = 0$ is given by $x \approx 1.42871599796$ (to eleven decimal places) using Newton's method algorithm, which discovers this solution, as described below.

Newton's method is a way to quickly approximate solutions to an equation $f(x) = 0$. It works by starting with a 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 best explained by a textbook, x_1 will usually be close to a solution, 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 eight decimal place precision after only four iterations – not bad.

You will implement the code for the function `poly_deriv` given in the template file. This function takes a list of coefficients for the original polynomial and then creates the corresponding list of coefficients for the derivative. For example, if the original polynomial is represented by `[2, 1, 3, 10]`, then the derivative would be represented by `[1, 6, 30]` (that would be $1 + 6x + 30x^2$).

You will also implement the code for the function `poly_eval`, which evaluates polynomials 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). The function takes two arguments, a list of coefficients and a value of x, and returns the corresponding polynomial evaluated at that value of x. (For example, `poly_eval([2,4,1], 20)` should evaluate to 482, since when you plug in $x = 20$ into $2 + 4x + x^2$ you get 482.) You can use this function with the coefficients for $f(x)$ *and* its derivative.

Finally, to implement Newton's method – don't overthink it; it is not complex 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 see the value of x_{11} anymore. You will write the code for Newton's method in the `main()` function in the template file.

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 above samples ($x^5 + 2x^4 - 3x - 10 = 0$ 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 the code in the template file
- implement the function `poly_deriv` as specified above and in the template file
- implement the function `poly_eval` as specified above and in the template file
- use the functions `poly_deriv` and `poly_eval` and estimate a solution to the equation $f(x) = 0$ using the input for the number of iterations of Newton's method, starting with the value x_0.
- the program should display the value x_n to eleven decimal places (where n is the number of `iterations` for Newton's method that was obtained from input), which will usually be close to a solution.