MTH 3300
Spring 2018
E. Fink
October 14, 2018

Homework 6

Directions: Download the template files I have provided on Blackboard. Then open Spyder, load these template files, and write the following programs. Submit your source code to me via Blackboard, 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.

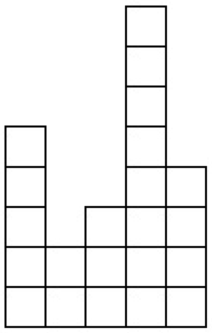
Be sure to read the SPECIFICATIONS carefully for all problems! And write comments!

1) boxchart.py

My file `boxchart.py`, and you will see a list

```
VALUES = [5, 2, 3, 8, 4]
```

Write a program that uses the `turtle` module to make this thing, where the number of boxes in each column is the corresponding entry of `VALUES`:



Your program must be written in such a way that if I replaced

```
VALUES = [5, 2, 3, 8, 4]
```

with

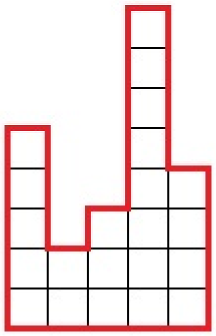
```
VALUES = [6, 2, 3, 9]
```

(or any other list of positive integers), it will still work properly. (In particular, you must use the variable name `VALUES` for your list.)

Specifications: your program must

- use the `turtle` module to draw a boxchart like the one shown, where the number of boxes in the ith column is equal to the ith entry in the list called `VALUES`. Note that each column should have its bottom at the same height; each square in a column should be the same size, placed directly on top of the one below it; and each column should be placed immediately to the right of the column that precedes it, with no space in between adjacent columns. The turtle may end wherever you like, as long as the boxchart has been drawn.
- write your program in such a way that if I were to replace the given line defining the list `VALUES` with any other list of positive integers, the program would still draw a boxchart, with the appropriate number of boxes in each column.

Challenge: *make the program outline the exterior of the boxchart in red, like this:*



To be clear, adding color is easy – but outlining ALL the exterior and NOTHING BUT the exterior is challenging!

---

2) quintic.py

Write a program that solves fifth-degree polynomial equations using Newton's method.

Your program should accept coefficients for c_0, c_1, c_2, c_3, c_4 and c_5. It should then accept a "guess" value, and then should 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 x_0: 1.1
1.4287159979621487
```

This is because a solution of $x^5 + 2x^4 - 3x - 10 = 0$ is given by $x \approx 1.4287159979621487$. The Newton's method algorithm which discovers this solution is described below.

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.

So, your code should "get the function" by having the user input the coefficients; it should also have the user input x_0. The program should run 10 passes through Newton's method – that is, it should calculate and print out x_{10}, which usually is very close to an actual solution.

Here are some suggestions. First, I think it would be a good idea to store the coefficients that the user inputs as a list, with index `i` holding the x^i coefficient.

It might also help to have code which 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]` (that would be $2 + x + 3x^2 + 10x^3$), then the derivative would be represented by `[1,6,30]` (that would be $1 + 6x + 30x^2$).

Furthermore, I suggest making a function called `poly` which takes two arguments, a list of coefficients and a value of x, and outputs the corresponding polynomial evaluate at that value of x. (For example, `poly([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 both with the coefficients for $f(x)$ *and* its derivative.

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 $x_0 = 0.1$ and $x^2 - 4x + 1 = 0$ with $x_0 = 1$.

Specifications: your program must

- ask the user to input 6 coefficients, representing the constant, x, x^2, x^3, x^4, and x^5 coefficients of a polynomial, $f(x)$.
- ask the user also to input a "guess" value, x_0.
- estimate a solution to the equation $f(x) = 0$ using 10 iterations of Newton's method starting with the value x_0; the program should display the value x_{10}, which will usually be close to a solution.

Challenge: *have your program also ask the user to enter an error tolerance. Then, instead of having your loop run 10 times, have it run until $|f(x_i)|$ is less than the given error tolerance ($|f(x_i)|$ is supposed to be 0, but it won't usually be exactly 0 – if it is close to 0, however, then this is indicative of an accurate solution).*