NEWTON'S METHOD : 25.0 POINTS

Newton's method (also known as the Newton-Raphson method) is a successive approximation method for finding the roots of a function. Recall that the roots of a function $f(x)$ are the values of x such that $f(x) = 0$. You can read more about Newton's method here (http://en.wikipedia.org/wiki/Newton%27s_method).

In this problem, we define an ***iteration*** as a specific computation to update our guess for the root. Thus, if your procedure gets lucky and finds a close enough answer using x_0, that would be iteration 0.

Here is how Newton's method works:

1. On the first iteration, we guess some x_0.
2. We check to see if it's a root or close enough to a root by calculating $f(x_0)$. If $f(x_0)$ is within some small value epsilon of 0, we say that's good enough and call x_0 a root.
3. If $f(x_0)$ is not good enough, we need to come up with a better guess, x_1. The value of x_1 is calculated by the equation:

   $$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

4. We check to see if x_1 is close enough to a root. If it is not, we make an even better guess x_2 and check that. And so on and so on.

   For every x_n that is not close enough to a root, we replace it with $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ and check if that is close enough to a root. We repeat until we finally find a value close to a root.

For simplicity, we will only be using polynomial functions in this problem set.

Implement the `computeRoot` function. This function applies Newton's method of successive approximation as described above to find a root of the polynomial function. It takes in a list of numbers `poly`, an initial guess `x_0` that serves as the "first iteration" of the method, and an error bound `epsilon`. It returns a list. The first element of this list is the root of the polynomial represented by `poly`, and the second element is the number of iterations it took to get to that root.

The function starts at `x_0`. It then applies Newton's method. It ends when it finds a root x such that the absolute value of $f(x)$ is less than `epsilon`, i.e. when $f(x)$ is close enough to zero. It returns the root it finds as a float. The root found does not have to equal the example provided to the exact decimal point.

Example Usage:

```
# - 13.39 + 17.5x^2 + 3x^3 + x^4
>>> poly = [-13.39, 0.0, 17.5, 3.0, 1.0]
>>> x_0 = 0.1
>>> epsilon = .0001
>>> print computeRoot(poly, x_0, epsilon)
```

```
>>> print computeRoot(poly, x_0, epsilon)
[0.80679075379635201, 8]
```

You should implement this function on your own machine, in the file `ps3_newton.py`. Test your code well in Idle, and when you are convinced it is correct, cut and paste your definition into this tutor window.

Here are some example Test Cases to test your code with - feel free to make up your own additional test cases. Be sure to test these on your own machine - and that you get the same output! - before running your code on this webpage!

Note: You may use the functions `evaulatePoly` and `computeDeriv` directly, you do not have to copy your definitions into this box. We have provided definitions of these two functions that your code may call and use.

```
1  def computeRoot(poly, x_0, epsilon):
2      '''
3      Uses Newton's method to find and return a root of a polynomial function.
4      Returns a list containing the root and the number of iterations required
5      to get to the root.
6
7      poly: list of numbers, length > 1.
8          Represents a polynomial function containing at least one real root.
9          The derivative of this polynomial function at x_0 is not 0.
10     x_0: float
11     epsilon: float > 0
12     returns: list [float, int]
13     '''
14     # FILL IN YOUR CODE HERE...
15
```

Unsubmitted

Check **Save** *You have used 0 of 30 submissions*

Show Discussion **New Post**