CS100 Homework 8
Introduction to Programming, Spring 2021

2021-06-07

Submission Deadline: 2021-06-22 23:59:59

1 Implementation of `poly.hpp`

You are to implement a class named `Poly` that simply represents a uni-variate, scalar valued polynomial. A polynomial can be represented by a vector of coefficients, in ascending order of powers. For example: $x^3 + 2x^2 + 2$ can be represented by $\{2, 0, 2, 1\}$, since the coefficients of x^0, x^1, x^2, x^3 are $2, 0, 2, 1$, respectively.
The interface of the `class Poly` is given to you, where you need to implement basic functions and operators. We have provided comments for them, and they are also pretty self explaining.
There are some further functions in the Poly class that deserve attention.

1. The function `eval` evaluates the value of this polynomial as a function at `param`.
2. The function `der` constructs the first-order derivative of the polynomial.
3. The most important one is

   ```
   1     std::pair<T,T> operator()( T param ) const
   ```

   which returns the value of the polynomial at `param` and also the first order derivative at `param` (just an evaluation of the derivative polynomial). This function is going to be useful for Problem 2.

Note: there is an intricate link to functional programming: a polynomial can be interpreted as a simple function that takes in a single parameter and returns a scalar. More interestingly, the sum, product, etc. or even composition of two scalar polynomials remains a scalar polynomial, so many of the operations on these polynomials can be understood as functional forms. However, not all functional forms can be easily realized, as the division of two polynomials does not return a polynomial.

Submission: please submit to autolab a .tar file containing poly.hpp.

2 Implementation of `functional_forms.hpp`

Here we consider polynomials as functional forms, and use lambda functions that either one or two uni-variate functions and construct a new uni-variate function from it. Six basic operations need to be defined:

- sum
- subtraction
- multiplication
- division
- function composition
- power function

These functions should be easily understood: if $f(x) = x^2 + 1$, and $g(x) = x - 3$, then $(f + g)(x) = x^2 + x - 2$, and $(f(g))(x) = x^2 - 6x + 10$. *(verify it!)*
There is one speciality about the involved functions. For all of them, we assume that they return an `std::pair<double,double>`, where `first` represents the actual value of that the function at at certain point, and `second` is the first order derivative.
Example: if $f(x) = x^2 + 1$, and $g(x) = x - 3$, then calling `derSum(f, g)(0.5)` will return an `std::pair` of $(-1.25, 2)$. *(verify it!)*

Notes:

1. For all above functional forms the derivative is clearly defined:

$$\text{if } g(x) = f(x) + h(x), \text{ then } \frac{dg}{dx}(x) = \frac{df}{dx}(x) + \frac{dh}{dx}(x)$$

$$\text{if } g(x) = f(x) - h(x), \text{ then } \frac{dg}{dx}(x) = \frac{df}{dx}(x) - \frac{dh}{dx}(x)$$

$$\text{if } g(x) = f(x) \cdot h(x), \text{ then } \frac{dg}{dx}(x) = \frac{df}{dx}(x)h(x) + f(x)\frac{dh}{dx}(x)$$

$$\text{if } g(x) = \frac{f(x)}{h(x)}, \text{ then } \frac{dg}{dx}(x) = \frac{\frac{df}{dx}(x)h(x) - f(x)\frac{dh}{dx}(x)}{h(x) \cdot h(x)}$$

$$\text{if } g(x) = h(f(x)), \text{ then } \frac{dg}{dx}(x) = \frac{dh}{dx}(f(x))\frac{df}{dx}(x)$$

$$\text{if } g(x) = (f(x))^p, \text{ then } \frac{dg}{dx}(x) = p(f(x))^{(p-1)}\frac{df}{dx}(x)$$

2. The polynomials implement the `operator()` and it returns `std::pair<double,double>` (if double is the template parameter). They are used as an input to the above functional forms.

The advantage of using the above functional forms is that we have automatic differentiation! There is an example in the file `test_lambdas.cpp` which uses the below complicated function:

$$g(x) = \frac{(p_2(p_1(x)))^3 - p_3(x)}{p_4(x) + p_5(x) \cdot p_6(x)}$$

It wouldn't be possible to combine polynomials into new polynomials as the expression contains a division. Furthermore, deriving the deriviative by hand would be tedious, and derivate calculation with finite differences is inaccurate. However, calculating either the value or the derivative at any given point would become simple by using above lambda functions.
Hint: (This is not part of your implementation) If you want to verify the correctness of your differentiation, you can easily do so by using finite difference approximation:

$$\frac{dg}{dx} = \lim_{\Delta x \to 0} \frac{g(x + \Delta x) - g(x)}{\Delta x}$$

Submission: please submit to autolab a .tar file containing "poly.hpp" and "functional_forms.hpp".

3 Implementation of `newton.hpp`

Here you need to implement Newton's method (for simple uni-variate functions). It's a method for finding nearest root from a starting point by iteratively using linear approximation. You can follow the link below, or search for "Newton's method" on any search engine.
https://en.wikipedia.org/wiki/Newton%27s_method
Its basic idea is, we starting at

$$x_0 = \text{startingPoint}$$

, and repeatedly finding the next x by

$$x_{k+1} = x_k - (f(x_k)/f'(x_k))$$

until $f(x_n)$ is close enough to 0.
We can easily apply it to the functions from problem 2, as they return the value and the derivative. The first parameter of this function (f) is exactly a lambda function generated from problem 2. You can refer to the usage in the provided example `test_newton.cpp` for details.

Submission: please submit to autolab a .tar file containing "poly.hpp", "functional_forms.hpp", and "newton.hpp".

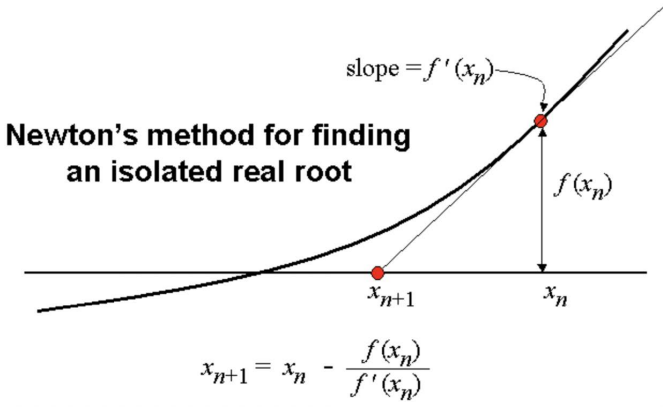

Figure 1: Newton's method