William Daniels AMS 326 homework one writeup

February 17, 2023

All of my source files can be found in this github repositiory:
https://github.com/William-J-Daniels/DanielsAms326.git

A detailed README on how the repositiory is used is there. The commit that represents the state of the repository upon submission of the assignment is the one at 2:30 AM on Feb. 18 (I can't give the hash before I commit).

Problem one

Description

This problem is to find the root of the given function, $\cos x - x^3$, using four different methods:

- Bisection using initial guesses of 0 and 1,
- Newton-Raphson using an initial guess of 0.3,
- Secant, using intial guesses of 0 and 1, and
- Fixed-point iteration, using an intial guess of 0.

Algorithms

Before giving my implimentations for each algorithm, I should first describe how I implimented them. I used an object oriented approach in which each root finding algoritm is implimented in a class which inherits from an abstract class RootFinder. It's implimentation is given in `RootFinding/include/RootFinder.h` and `RootFinding/src/RootFinding.cpp`.

Bisection

Given initiail bounds a and b such that $f(a)f(b)<0$
while (b – a)/2 > desired precision
    $c=(a+b)/2$
    if $f(c)=0$ **break**
    if $f(a)f(b)<0$
        b=c
    else
        a=c
The approximate root is (a+b)/2

Listing 1: Bisection psuedocode

The implimentation is in `RootFinding/include/Bisection.h` and `RootFinding/src/Bisection.cpp`.

Newton-Raphson

Given an itital guess of the root x0
while consecutive iterations adjust the estimate by an amount less than the desired precision
    $x_i=x_{i-1}-\frac{f(x_{i-1})}{f'(x_{i-1})}$

Listing 2: Newton-Raphson psudo code

The implimentation can be found in `RoodFinding/include/NewtonRaphson.h` and `RoodFinding/src/NewtonRaphson.cpp`. My implimentation also contains a method that uses a second order two point approximation of the derivative, though only the analytic derivative is used for the assignment.

Secant

Given initial guesses a and b that are to the left and right of the root respectively
while consecutive iterations adjust the estimate by an amount less than the desired precision
    $x_i=x_{i-1}-\frac{f(x_{i-1})f(x_{i-2})}{f(x_{i-1})-f(x_{i-2})}$

Listing 3: Secant Psudocode

This is the a modification of the Newton method using a first order forward approximation of the dirivative.

Fixed-point

Given an intial guess x_0
while consecutive iterations adjust the estimate by an amount less than the desired precision
$$x_i = f(x_i)$$

Listing 4: Fixed-point psudocode

Results

- The root according to bisection is 0.865474. It took 39 iterations.
- The root according to Newton-Raphson is 0.865474. It took 8 iterations.
- The root according to secant is 0.865474. It took 9 iterations.
- The fixed point of the suggested transformation is 0.60352. It took 22 iterations.

This is the expected result for all methods. As per the parameters in `RootFinding/include/RootFinder.h`, all of these solution are precice to 10^{-12}.

- My bisection implimentation uses about 27 FLOPs per iteration with the given function, so it used about 1053 FLOPs to find the root .
- My Newton-Raphson implimentation (analytic derivative) uses about 52 FLOPs per iteration with the given function and its derivative, so it used about 416 FLOPs to find the root.
- My secant implimentation uses about 75 FLOPs per iteration with the given function, so it used about 675 FLOPs to find the root.
- My fixed-point implimentation uses about 21 FLOPs per iteration with the given transformation, so it used about 462 FLOPs.

Problem 2

I was out of time to finish implimenting interpolation in C++, so I used Python for this problem.

Description

The problem is to perform a polynomial interpolation on the following data

t	1	2	3	4	5
y	122.44	123.45	123.22	118.85	119.77

Algorithm

$$\begin{bmatrix} x_0^n & x_0^{n-1} & x_0^{n-2} & \dots & x_0 & 1 \\ x_1^n & x_1^{n-1} & x_1^{n-2} & \dots & x_1 & 1 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ x_n^n & x_n^{n-1} & x_n^{n-2} & \dots & x_n & 1 \end{bmatrix} \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_0 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

Results

My program produced the polynomial $-0.1\,x^4 + -0.02\,x^3 + 5.2\,x^2 - 15\,x + 130$. This produces the prediction of 96, but it is a meaningless prediction-- interpolation is useful for learning about what happens between your data points, not beyond them.