CAPSTONE PROJECT INSTRUCTIONS

Due Dec 6, 2020

The components of the final project are as follows:

- The following items are due on **December 6** by **11:59 pm** and make up 90% of the project grade.
  - You must complete sections 1-3 outlined below. Any code written must be well commented.
  - You must submit your full SciProgLib library containing all codes completed during lecture and any additional programs written for this project. See details in section 4.
  - A written report between 6 and 10 pages, 12pt font, double spaced. This report should contain your programs (single spaced), responses to question prompts, and figures when necessary.
- A short presentation (5-10 minutes, with an extra 3-5 minutes for questions) that details your code and your results. Presentations will take place over the last 6 lecture periods. (10%)
  - Nov 18
  - Nov 20
  - Nov 23
  - Nov 30
  - Dec 2
  - Dec 4

  If you do not have results by this time, present your code and share where you are at in the project. Treat the presentation like a progress report, where we can give you feedback for your code and written report.

CAPSTONE PROJECT INSTRUCTIONS

1: Root Finding Methods

(a) **Secant Method**

A potential problem in implementing the Newton-Raphson method is the evaluation of the derivative. Although this is not inconvenient for polynomials and many other functions, there are certain functions whose derivatives may be difficult or inconvenient to evaluate. For these cases, the derivative can be approximated by a backward finite divided difference:

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

If we take the approximation for the derivative

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

and substitute it into our Newton-Raphson formula

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

we end up with

$$x_{i+1} = x_i - \frac{f(x_i)\,(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}$$

This is the formula for the *secant method*. Notice that the approach requires two initial estimates of x. However, because $f(x)$ is not required to change signs between the estimates, it is not classified as a bracketing method.

Write a function that implements the secant method. Use the function name and parameters specified below.

```
SecantMethod(double x0, double x1, double tol, int maxit, double(*f)(double x))
```

(b) **Secant Method Test**

Use your `SecantMethod` code to determine the mass of the bungee jumper with a drag coefficient of 0.25 kg/m to have a velocity of 36 m/s after 4 s of free fall. Note: The acceleration of gravity is 9.81 m/s^2. Use initial guesses of 50 kg and 40kg and tol =0.0001.

Report

- the mass of the bungee jumper
- iterations it needed to find the root of the function

Compare your results with the Newton-Raphson method.

CAPSTONE PROJECT INSTRUCTIONS

(c) **Case Study - Newton-Raphson and the Bisection Method**
Determine the positive root of $f(x) = x^{10} - 1$ using the

- Newton-Raphson method and an initial guess of $x = 0.5$ and tol $= 0.0001$.
- Bisection method with an initial guess of $a = 0.5$, and $b = 1.1$, and tol $= 0.0001$.

Report

- the root for each method
- the number of iterations it took each method to find the root.

Compare your results. Sketch the first few iterations of the Newton-Raphson method to explain your results.

CAPSTONE PROJECT INSTRUCTIONS

2: Optimization

(a) **Golden-Section Search**
Your current `GoldenSectionSearch` code only finds the minimum values of a function. **Explain** why and **modify** your function so that it takes in a boolean as an input, i.e.
`GoldenSectionSearch(double xu, double xl, double tol, double (*f)(double x), bool min)`

- if `bool min` is `true`, your function should return the minimum of the function.
- if `bool min` is `false`, your function should return the maximum of the function.

(b) **Golden-Section Search Test**
Use your function `GoldenSectionSearch` to find the

- minimum using $x_u = 2\pi$ and $x_l = 0$
- maximum using $x_u = 2\pi$ and $x_l = 0$

of the function $f(x) = \sin(x)$.

(c) **The optima is 0**
Currently, the `ParabolicInterp` method will not return the optima for the following function

$$f(x) = -x^5 + 2x^2 + 1$$

with the initial guesses $x_1 = -0.7$, $x_2 = 0.5$, and $x_3 = 1$.

- **Explain** why.
- Modify the method to address all instances of where the issue could occur so that it returns where the optima is located, $x \approx 0$.
- Modify the `GoldenSectionSearch` method to address the same type of issue.

CAPSTONE PROJECT INSTRUCTIONS

3: Integration

(a) **Gauss Quadrature**

In addition to the methods we covered in class, we can also use the Gauss-Legendre formula (Gauss quadrature for short) to help us evaluate an integral. The objective of Gauss quadrature is to determine the unknown coefficients $c_0, ..., c_{n-1}$ and x-values $x_0, ..., x_{n-1}$ in the equation below

$$I \cong c_0 f(x_0) + .. + c_{n-1} f(x_{n-1}),$$

where n is the number of Gauss-Legendre points used to approximate the integral. These values are derived analytically on the interval $[-1, 1]$, and have been summarized in the table below.

Points	Weighting Factors	Function Arguments	Truncation Error
1	$c_0 = 2$	$x_0 = 0.0$	$\cong f^{(2)}(\xi)$
2	$c_0 = 1$ $c_1 = 1$	$x_0 = -1/\sqrt{3}$ $x_1 = 1/\sqrt{3}$	$\cong f^{(4)}(\xi)$
3	$c_0 = 5/9$ $c_1 = 8/9$ $c_2 = 5/9$	$x_0 = -\sqrt{3/5}$ $x_1 = 0.0$ $x_2 = \sqrt{3/5}$	$\cong f^{(6)}(\xi)$
4	$c_0 = (18 - \sqrt{30})/36$ $c_1 = (18 + \sqrt{30})/36$ $c_2 = (18 + \sqrt{30})/36$ $c_3 = (18 - \sqrt{30})/36$	$x_0 = -\sqrt{525 + 70\sqrt{30}}/35$ $x_1 = -\sqrt{525 - 70\sqrt{30}}/35$ $x_2 = \sqrt{525 - 70\sqrt{30}}/35$ $x_3 = \sqrt{525 + 70\sqrt{30}}/35$	$\cong f^{(8)}(\xi)$
5	$c_0 = (322 - 13\sqrt{70})/900$ $c_1 = (322 + 13\sqrt{70})/900$ $c_2 = 128/225$ $c_3 = (322 + 13\sqrt{70})/900$ $c_4 = (322 - 13\sqrt{70})/900$	$x_0 = -\sqrt{245 + 14\sqrt{70}}/21$ $x_1 = -\sqrt{245 - 14\sqrt{70}}/21$ $x_2 = 0.0$ $x_3 = \sqrt{245 - 14\sqrt{70}}/21$ $x_4 = \sqrt{245 + 14\sqrt{70}}/21$	$\cong f^{(10)}(\xi)$
6	$c_0 = 0.171324492379170$ $c_1 = 0.360761573048139$ $c_2 = 0.467913934572691$ $c_3 = 0.467913934572691$ $c_4 = 0.360761573048131$ $c_5 = 0.171324492379170$	$x_0 = -0.932469514203152$ $x_1 = -0.661209386466265$ $x_2 = -0.238619186083197$ $x_3 = 0.238619186083197$ $x_4 = 0.661209386466265$ $x_5 = 0.932469514203152$	$\cong f^{(12)}(\xi)$

Write a function `GaussLegendre` that takes as an input

- the number of Gauss-Legendre points to be used, n
- a pointer to the function f

```
GaussLegendre(int n, double (*f)(double(x)))
```

Your function should **Return** the integral approximate, I.

CAPSTONE PROJECT INSTRUCTIONS

(b) Test your code using the function

$$f(y) = 0.2 + 25y - 200y^2 + 675y^3 - 900y^4 + 40y^5$$

between the limits $y = 0$ to 0.8. The exact value of the integral is 1.640533.

NOTE: Before we can approximate this function, we have to perform a change in variable so that the limits are from -1 to 1. To do this, we substitute $a = 0$ and $b = 0.8$ into the equation

$$y = \frac{(b+a) + (b-a)x}{2} = 0.4 - 0.4x$$

and

$$dy = \frac{b-a}{2}dx = 0.4dx$$

Our equation becomes

$$f(x) = 0.4[0.2 + 25(0.4 - 0.4x) - 200(0.4 - 0.4x)^2 + 675(0.4 - 0.4x)^3 - 900(0.4 - 0.4x)^4 + 400(0.4 - 0.4x)^5]$$

Using your code, fill in the table below.

n	I	Percent relative error
1		
2		
3		
4		
5		
6		

(c) How might you modify your code to compute the integral to a desired tolerance?

CAPSTONE PROJECT INSTRUCTIONS

4: Your Library

Your final library containing ALL codes listed below. Your function declarations should be identical to the ones listed below. This includes the **function name**, the **number of parameters**, and the **order of parameters.**

- The Bisection method,
  `Bisect(double a, double b, double tol, double (*f)(double x))`
- Newton-Raphson method,
  `NewtonRaphson(double a, double tol, int maxit, double (*f)(double x), double (*df)(double x))`
- Golden-Section Search Method,
  `GoldenSectionSearch(double xu, double xl, double tol, double (*f)(double x), bool min)`
- Parabolic Interpolation,
  `ParabolicInterp(double x1, double x2, double x3, double tol, double (*f)(double x))`
- The Recursive Trapezoid Rule for functions,
  `CompositeTrapRule(double a, double b, double n, double avgdf2, double tol, double (*f)(double x))`
- Recursive Simpson's 1/3 Rule for functions,
  `CompositeSimps13(double a, double b, double n, double avgdf4, double tol, double (*f)(double x))`
- Simpson's Rules for data,
  `DataSimpsonsRule(double x[], double fx[], double n)`
- Trapezoid Rule for data,
  `DataTrapezoidalRule(vector<double> x, vector<double> fx, int n)`
- Trapezoid Rule for function, non recursive,
  `TrapezoidRule(double a, double b, int n, double (*f)(double x))`
- 4th order Romberg Integration,
  `RombergIntegration(double a, double b, double tol, double (*f)(double x))`
- Adaptive Quadrature using Boole's Rule,
  `AdaptiveQuadrature(double a, double b, double tol, double (*f)(double x))`
- The Secant method
  `SecantMethod(double x0, double x1, double tol, int maxit, double(*f)(double x))`
- Gauss Quadrature
  `GaussLegendre(int n, double (*f)(double(x)))`
- Forward finite difference
  `forwardDifference(double x, double h, double (*f)(double x)`

CAPSTONE PROJECT INSTRUCTIONS

- Backward finite difference
  `backwardDifference(double x, double h, double (*f)(double x))`
- Centered finite difference
  `centeredDifference(double x, double h, double (*f)(double x))`