CAPSTONE PROJECT INSTRUCTIONS

Rubric

Due Dec 3, 2021

The components of the final project are as follows:

- The following items are due on **December 3** by **11:59 pm**
  - 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, 12pt font, double spaced. This report should contain your responses to question prompts and figures where necessary.

Question	Part	Score	Total Points
1 (30pts total)	a		9pts
	b		6pts
	c		15pts
2 (30pts total)	a		13pts
	b		4pts
	c		13pts
3 (30pts total)	a		14pts
	b		12pts
	c		4pts
4 (10pts total)	–		10pts

Total Score /100

CAPSTONE PROJECT INSTRUCTIONS

Rubric

1: Root Finding Methods 30pts

(a) **Secant Method 9pts**

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:

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

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 a *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. **Use one of your differentiation codes to approximate the derivative.**

```
Secant(double x0, double x1, double tol, int Maxit, double(*f)(double x))
```

Points	
9	Code compiles and is well commented.
7	Code compiles, but is not well commented.
5	Code compiles, but does not call a differentiation method. Code does not compile, but errors are small.
0	Code does not compile, errors are significant.

(b) **Secant Method Test 6pts**

Use your `Secant` 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. The acceleration of gravity is 9.81 m/s^2. Use initial guesses of 40 kg and 50 kg and tol = 0.0001.

Report:

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

Do the same experiment using your `Bisect` and `NewtonRaphson` codes. For `Bisect`, use an initial guess of 40 kg and 250 kg. For `NewtonRaphson`, use an initial guess of 50 kg. **Compare** the results of the three methods. Recall, the function for the bungee jumper is

$$f(m) = \sqrt{\frac{gm}{c_d}} \tanh\left(t\sqrt{\frac{gc_d}{m}}\right) - v(t),$$

CAPSTONE PROJECT INSTRUCTIONS

Rubric

with the derivative

$$f'(m) = \frac{1}{2}\sqrt{\frac{g}{mc_d}}\tanh\left(t\sqrt{\frac{gc_d}{m}}\right) - \frac{gt}{2m}\left(1 - \tanh^2\left(t\sqrt{\frac{gc_d}{m}}\right)\right).$$

Method		Mass	Iterations	Correct	Incorrect due to small logic error	Not submitted Incorrect due to major logic error
Secant	Backward Difference Forward Difference Centered Difference	142.7329 142.7329 142.7376	6 6 8	2pts	1pts	0pts
Newton-Raphson		142.7376	5	2pts	1pts	0pts
Bisection		142.7313	14	2pts	1pts	0pts

(c) **Case Study - Newton-Raphson and the Bisection Method 15pts**

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 $= 0.0001$.

The derivative of $f(x)$ is

$$f'(x) = 10x^9.$$

Report:

- the root for each method 1pt per method, 2pts total
- the number of iterations it took each method to find the root 1pt per method, 2pts total

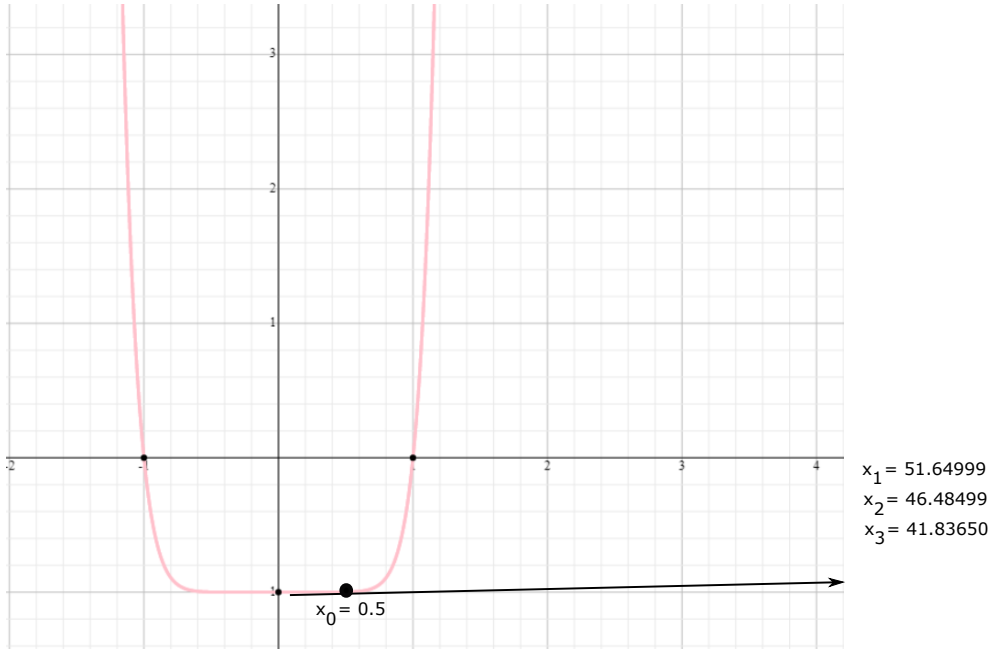
Compare your results. Sketch the first few iterations of the Newton-Raphson method (5pts) to explain your results (6pts).

Method	Root	Iterations	Correct	Incorrect due to small logic error	Not submitted Incorrect due to major logic error
Newton-Raphson	1.0000239	13	2pts	1pts	0pts
Bisection	1.0000244	42	2pts	1pts	0pts

The bisection method converges quicker than Newton-Raphson even though it is a bracketing method. This is because of our initial guess for the method. At this point, the slope of the curve is very close to 0 which yields a much larger guess for x_1, since x_1 is the point where the tangent curve passes through the x-axis. We see the subsequent values of x decrease, meaning that it is converging to our solution. Because of this, it takes several iterations for the Newton-Raphson method to converge to the root. (6pts full response, 3pts partial response)

CAPSTONE PROJECT INSTRUCTIONS

Rubric



5pts figure contains x_0, initial tangent curve, and itentifies at least 2 more points.
3pts figure contains x_0 and initial tangent curve but does not identify any future points.
1pt figure only contains x_0.
0pts Incorrect response.

CAPSTONE PROJECT INSTRUCTIONS

Rubric

2: Optimization 30pts

(a) **Golden-Section Search 13pts**

Your current `GoldenSectionSearch` code only finds the minimum values of a function. **Explain** why (3pts) and **modify** your function so that it takes in a boolean as an input (10pts), i.e. `GoldenSectionSearch(double xl, double xu, double tol, double (*f)(double x), bool min)`

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

The Golden-Section search compares $f(x_i)$ and $f(x_{i+1})$ and looks for the smaller value. This allows the method to specifically converge to the minimum value. If we want the method to look for a maximum, we would need to compare these two values and look for the larger value. (3pts correct response, 2pts partially correct response, 0pts incorrect/no resonse.)

Points	
10	Code compiles and is well commented.
8	Code compiles, but is not well commented.
5	Code does not compile, but errors are small.
0	Code does not compile, errors are significant. Code compiles but does not address the question.

(b) **Golden-Section Search Test 4pts**

Use your function `GoldenSectionSearch` to find the

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

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

	Optima value	Correct	Incorrect due to small logic error	Not submitted Incorrect due to major logic error
Maximum	4.7119887	2pts	1pts	0pts
Minimum	1.5708417	2pts	1pts	0pts

(c) **The optima is 0 13pts**

Currently, the `ParabolicInterpolation` 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. (2pts)
- **Modify** the method to address all instances of where the issue could occur so that it returns where the optima where $x \approx 0$. (7pts)
- **Modify** the `GoldenSectionSearch` method to address the same type of issue. (4pts)

CAPSTONE PROJECT INSTRUCTIONS

Rubric

This method breaks down when $x \approx 0$ because the denominator becomes 0. To avoid this, we can divide by the smallest machine value. (2pts complete response, 1pt incomplete response, 0pts incorrect/no response)

	Optima value	Correct	Incorrect/Not submitted
Parabolic interpolation	0.00000000088569751	1pt	0pts
Golden Section search	0.00000000745005601	1pt	0pts

Points Parabolic	Points GS Search	
6	3	Code compiles and is well commented.
4	2	Code compiles, but is not well commented.
2	1	Code does not compile, but errors are small.
0	0	Code does not compile, errors are significant. Code compiles but does not address the question.

3: Integration 30pts

(a) **Gauss Quadrature (14pts)**

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.

CAPSTONE PROJECT INSTRUCTIONS

Rubric

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.

Points	
14	Code compiles and is well commented.
12	Code compiles, but is not well commented.
7	Code does not compile, but errors are small.
0	Code does not compile, errors are significant. Code compiles but does not address the question.

CAPSTONE PROJECT INSTRUCTIONS

Rubric

(b) (12pts) Test your code using the function

$$f(y) = 0.2 + 25y - 200y^2 + 675y^3 - 900y^4 + 400y^5$$

between the limits $y = 0$ to 0.8. The exact value of the integral is 1.640533333333341 .

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

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

Using your code, fill in the table below.

n	I	% relative error	Correct	Incorrect/Not submitted
1	1.96480000000000032	19.765929778933145 40%	1pt/ea, 2pts tot	0pts
2	1.82257777777777719	11.096662332032391 16%	1pt/ea, 2pts tot	0pts
3	1.64053333333332896	0.000000000000730 88%	1pt/ea, 2pts tot	0pts
4	1.64053333333333073	0.000000000000622 61%	1pt/ea, 2pts tot	0pts
5	1.64053333333332985	0.000000000000676 75%	1pt/ea, 2pts tot	0pts
6	1.64053333333333007	0.000000000000663 21%	1pt/ea, 2pts tot	0pts

(c) (4pts) How might you modify your code to compute the integral to a desired tolerance?

There are several ways to approach this question:
As long as the student demonstrates good logic (4pts).
If student's logic is slightly flawed (2pts).
If student's logic is significantly flawed (1pt).
Student did not attempt to answer the question (0pts).

CAPSTONE PROJECT INSTRUCTIONS

Rubric

4: Your Library 10pts

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.**

Deduct 1pt per function that fails the unit test up to 10 pts.

- 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 xl, double xu, double tol, double (*f)(double x), bool min)`
- Parabolic Interpolation,
  `ParabolicInterpolation(double x1, double x2, double x3, double tol, double (*f)(double x))`
- The Recursive Trapezoid Rule for functions,
  `CompositeTrapezoid(double a, double b, int n, double avgdf2, double tol, double (*f)(double x))`
- Recursive Simpson's 1/3 Rule for functions,
  `CompositeSimpsons13(double a, double b, int n, double avgdf4, double tol, double (*f)(double x))`
- Simpson's Rules for data,
  `DataSimpsons(double x[], double fx[], int n)`
- Trapezoid Rule for data,
  `DataTrapezoid(vector<double> x, vector<double> fx, int n)`
- The Secant method
  `Secant(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

Rubric

- Backward finite difference
  `BackwardDifference(double x, double h, double (*f)(double x))`
- Centered finite difference
  `CenteredDifference(double x, double h, double (*f)(double x))`