COMPUTER SCIENCE 349A, FALL 2021
ASSIGNMENT #6 - 20 MARKS

DUE TUESDAY, DECEMBER 7, 2021 (11:30 p.m. PST)

This is a really large class and the logistics of grading assignments are challenging. Me and the markers require your help in making this process go smoothly. Please ensure that your assignments conform to the following requirements - any violation will result in getting a zero for the particular assignment.

- All assignments should be submitted electronically through the Brightspace course website and should be a **SINGLE PDF FILE** with all the material plus files for any functions (.m or .py). No other formats will be accepted.. Handwritten answers are ok but they will need to be scanned and merged into a single pdf file together with any code examples and associated plots.
- The assignment number, student name and student number should be clearly visible on the top of every page of your assignment submission.
- **PLEASE DO NOT COPY THE ASSIGNMENT DESCRIPTION IN YOUR SUBMISSION**
- The answers to the questions should be in the same order as in the assignment specification.
- Some of the questions of the assignments are recycled from previous years but typically with small changes in either the description or the numbers. Any submission that contains numbers from previous years in any questions will be immediately graded with zero.
- Any general assignment related questions can be posted in the Discusscion Forum in Brightspace for the assignment.
- Any specific assignment related questions (details about your solution) should be e-mailed (rlittle@uvic.ca) to me and have a subject line of the form CSC349A Assignment X, where X is the number of the corresponding assignment.
- You may use Python instead of MATLAB anywhere the instructions say MATLAB. In that case replace M-FILE or .m file with .py file. Again, I recommend the Spyder IDE for this.

Question #1 - 8 Marks

(a) **(4 points)** Fill in the blanks in the following MATLAB function m-file `trap` so that it implements the composite trapezoidal rule, using a sequence of $m = 1, 2, 4, 8, 16, \ldots$ trapezoids, to approximate $\int_a^b f(x)dx$. Note, you may do this in Python using this template and making the appropriate changes.

This m-file uses the MATLAB built-in function `trapz` (in Python you can use `numpy.trapz`, but note that the arguments should be reveresed in this case, i.e. `numpy.trapz(y, x)`). If

$$x = (x_1, x_2, \ldots, x_{m+1}) \text{ and } y = (f(x_1), f(x_2), \ldots, f(x_{m+1})),$$

then the execution of

```
z = trapz(x, y)
```

approximates

$$\int_{x_1}^{x_{m+1}} f(x)dx$$

using the composite trapezoidal rule (with m trapezoids). (Note that the entries of x are labeled starting at x_1 because MATLAB does not allow an index of 0.)

The function m-file `trap` has 4 input parameters:

- a and b (the lower and upper limits of the integral);
- `maxiter` (the maximum number of iterations that are allowed; the above program will implement the composite trapezoidal rule with a sequence of $1, 2, 4, 8, 16, \ldots$ trapezoids, up to a maximum of $2^{\texttt{maxiter}}$ trapezoids);
- `tol` (a tolerance for testing the relative error of the computed approximation). If the difference between 2 successive approximations (with respect to relative error) is $<$ `tol`, then the algorithm terminates.

You also must define and store a MATLAB function m-file (or Python py-file)

```
y = f(x)
```

which must be written to accept a vector `x` as the argument.

```
function trap(a, b, maxiter, tol)
m = 1;
x = linspace(a, b, m+1);  // numpy.linspace(a, b, m+1) in Python
y = f(x);
approx = trapz(x, y);  // numpy.trapz(y, x) in Python
disp('     m      integral approximation');
fprintf(' %5.0f %16.10f \n ', m, approx);
for i = 1 : maxiter
```

```
    m = _______________________________  ;

    oldapprox = ___________________________________  ;

    x = linspace ( _______ , ________ , __________ ) ;
    y = f(x);
    approx = trapz(x, y);
    fprintf(' %5.0f %16.10f \n ', m, approx);

    if abs( ______________________________ ) < tol
        return
    end
end
fprintf('Did not converge in %g iterations', maxiter)
```

DELIVERABLES: A copy of your function `trap`. Feel free to add an extra parameter `f` at the end of parameter list to make your function calls simpler.

(b) **(4 points)**

Use the above function to approximate the following two integrals:

$$\int_{0.1}^{3} x\cos(1/x)dx \text{ using } \texttt{maxiter} = 20 \text{ and } \texttt{tol} = 10^{-5},$$

$$\int_{-1}^{1} e^{3x}\sin(\sqrt{x+1}+1)dx \text{ using } \texttt{maxiter} = 20 \text{ and } \texttt{tol} = 10^{-7}.$$

DELIVERABLES: The function calls and output for each plus all the .m or .py files.

Question #2 - 6 Marks. Use a combination of your function *trap* in Question 1 and the following open Newton-Cotes quadrature formula for $n = 2$

$$\int_{x_{-1}}^{x_3} f(x)dx \approx \frac{4h}{3}\left[2f(x_0) - f(x_1) + 2f(x_2)\right]$$

in order to approximate

$$I = \int_0^1 \sqrt{\ln(1/x)}dx$$

Do this as follows: use *trap* (with $maxiter = 20$ and $tol = 1e-6$) to approximate I on the interval $[0.02, 1]$ and use the above open quadrature formula to approximate I on $[0, 0.02]$. Add these two results together to get an approximation to I. (All of this computation can be done in MATLAB or Python.)

Note: choose the value of h and the three quadrature points correctly for the open quadrature formula. Specify these values in your submitted solution.

Note: the exact value of I is 0.886227. Your computed approximation should have about 3 correct significant digits.

DELIVERABLES: The function calls and output for each. If you use function files to evaluate each of the functions I also want the files for those.

Question #3 - 6 Marks.

(a) **(4 points)** Derive the order $O(h^2)$ numerical differentiation formula

$$f'''(x_0) \approx \frac{1}{2h^3}\left[f(x_0+2h) - 2f(x_0+h) + 2f(x_0-h) - f(x_0-2h)\right]$$

using Taylor polynomial approximations of order 5 (with their remainder terms simply written as $O(h^6)$) for each of $f(x_0+2h)$, $f(x_0+h)$, $f(x_0-h)$, and $f(x_0-2h)$, expanded about x_0.

To do this, start with the expression $f(x_0+2h)-2f(x_0+h)+2f(x_0-h)-f(x_0-2h)$ on the left-hand side and susbstitute the Taylor approximations above into this expression on the right-hand side and then solve for $f'''(x_0)$. What is the error associated with this approximation? (Note: Your error should include a big-Oh term).

DELIVERABLES: All your work in constructing the formula and it's error.

(b) **(2 points)** Use the following data and the formula given in (a) to approximate $f'''(0.5)$ and calculate the truncation error given that the exact value of $f'''(0.5)$ is 1.312900.

x_i	$f(x_i)$
0.4	0.580944
0.45	0.682162
0.55	0.905949
0.6	1.028846

DELIVERABLES: All your work in calculating the approximation and truncation error.