COMPUTER SCIENCE 349A, SPRING 2019
ASSIGNMENT #4 - 20 MARKS

DUE FRIDAY, MARCH 8 2019 (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 ConneX course website and shoud be **SINGLE PDF FILES**. 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 asnwers 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 assignment related email questions should have a subject line of the form CSC349A Assignment X, where X is the number of the corresponding assignment.
- The total number of points for this assignment is 20.

Question #1 - 10 Marks

For this question your are going to create a function in MATLAB for approximating a root of a function using the Newton-Raphson method. I have included my Bisect.m function M-File for the Bisection Method which you may use as a template. You will then use your Newton function to solve an engineering problem from the textbook.

(a) **(2 points)** Write a MATLAB function M-file for Newton's method corresponding to the following pseudocode:

function root = Newton(x_0, ε, `imax`)
$i \leftarrow 1$
output heading
while $i \leq$ `imax`
    $root \leftarrow x_0 - f(x_0)/f'(x_0)$
    output i, root
    if $|1 - x_0/root| < \varepsilon$
        return
    end if
    $i \leftarrow i + 1$
    $x_0 \leftarrow root$
end while
output "failed to converge"

Use the following (or similar) MATLAB print statements for output.

```
fprintf ( ' iteration     approximation \n')
fprintf ( ' %6.0f %18.8f \n', i, root )
fprintf ( ' failed to converge in %g iterations\n', imax )
```

Pass the additional parameters `f` and `fp` for f and f' and use the function handle @functionname to pass the functions. (similar to the way f was a parameter in the bisection function in my code).

DELIVERABLES: A copy of your MATLAB M-file as part of your PDF.

(b) **(2 points)**

For the rest of this question, we will explore a problem from Mechanical/Aerospace engineering related to pipe friction. Determining fluid flow of liquids and gasses through pipe systems is an important application in mechanical engineering (for example in the design of cooling systems for aircraft). The resistance to flow in pipes is parametrized by a dimensionless number called the *friction factor*. For turbulent flow, the *Colebrook* equation can be used to calculate the friction factor by setting $g(f)$ to zero:

$$g(f) = \frac{1}{\sqrt{f}} + 2.0 \log_{10} \left(\frac{\epsilon}{3.7D} + \frac{2.51}{Re\sqrt{f}} \right) \tag{1}$$

where ϵ is the roughness, D is the diameter of the pipe, and Re is the *Reynolds number* which characterizes the flow.

$$Re = \frac{\rho V D}{\mu} \tag{2}$$

where ρ is the fluid density, V is its velocity, and μ is the dynamic viscocity.

In our case, we will try to calculate f for air flow through a smooth, thin tube. The parameters will be $\rho = 1.23$, $\mu = 1.79x10^{-5}$, $D = 0.005m$, $V = 40$ and $\epsilon = 0.0000015$. From the application we know that friction factors range from 0.008 to 0.08.

The *Reynolds* number can be computed as:

$$Re = \frac{\rho V D}{\mu} = \frac{1.23(40)0.005}{1.79 \times 10^{-5}} = 13743 \tag{3}$$

This value, along with the other parameters can be substituted to give:

$$g(f) = \frac{1}{\sqrt{f}} + 2.0 \log_{10}\left(\frac{0.0000015}{3.7(0.005)} + \frac{2.51}{13743\sqrt{f}}\right) \tag{4}$$

Note. The log in the above equation is the base 10 logarithm function which in MATLAB can be written as *log10*.

You will need to write MATLAB function M-files with headers something like

`function y = fQ1(x)` and `function y = fpQ1(x)`

corresponding to the function $f(x)$ that you are computing a zero of, and its derivative $f'(x)$, which you will need to determine using calculus.

DELIVERABLES: Copies of your M-files for the function and its derivative in your PDF.

(c) **(2 points)** Use the MATLAB function M-file `Newton` with $\varepsilon = 10^{-8}$, $x_0 = 0.008$, and $imax = 20$ to solve the problem in Q1(b). Remember, you cannot pass the functions f and f' directly as arguments to the Newton function. You should use function handles, thus your call should look like the following:

```
Newton(0.008,1e-08,20,@fQ1,@fpQ1)
```

DELIVARABLES: Provide the call you used and the output of the function.

(d) **(2 points)** Use the MATLAB function M-file `Newton` with $\varepsilon = 10^{-8}$, $x_0 = 0.08$, and $imax = 20$ to solve the problem in Q1(b).

DELIVARABLES: Provide the call you used and the output of the function.

(e) **(2 points)** In questions 1(c) and 1(d) we use for initial guesses the lower end of the range ($x_0 = 0.008$) as well as the upper end of the range ($x_0 = 0.08$), respectively. Write a short sentence describing what happens in these two cases and report on the number of iterations in each case. Why is this happening?

Question #2 - 6 marks.
Let R denote any positive number. Newton's method is often used to determine an iterative formula for calculating frequently-used functions of R on computers and calculators; for example, to compute $\sqrt{R}$, $\sqrt[3]{R}$, or $\sqrt[m]{R}$ for any given value of m, or $1/R$ (using only addition and multiplication), and so on. The basic idea is to find an equation $f(x) = 0$ that has the value you want to compute (for example, $\sqrt{R}$) as one of its roots, and then apply Newton's method to compute a root of this equation. If you run Newton's method and it converges, then it will converge to the desired answer. For example, $x = \pm\sqrt{R}$ are the only zeros of the function $f(x) = x^2 - R$. If Newton's method is applied using this particular $f(x)$, the following iterative formula is obtained:

$$(*) \qquad x_{i+1} \;=\; x_i - \frac{f(x_i)}{f'(x_i)} \;=\; x_i - \frac{x_i^2 - R}{2x_i} \;=\; \left(x_i + \frac{R}{x_i}\right)/2.$$

If this sequence of values $\{x_i\}$ converges to a value $\hat{x}$, then $\hat{x}$ must be a zero of $f(x)$; that is, $\hat{x} = \pm\sqrt{R}$. Thus, the iterative formula (*) can be used to compute square roots using only addition and division.

(a) **(2 points)** Let $m \geq 2$ denote any positive integer. Apply Newton's method to

$$f(x) = x^m - R$$

in order to determine an iterative formula (similar to (*)) for computing $\sqrt[m]{R}$. Simplify your iterative formula so that it is in a form that looks exactly like the rightmost part of (*) when $m = 2$.

(b) **(2 points)** Write a MATLAB function M-file to compute $\sqrt[m]{R}$ based on the iterative formula in (a). This M-file can be obtained by modifying the `Newton` function you wrote in question 1 as follows:

change the function header to

```
function root = mth_root(m, R, x0, eps, imax)
```

and replace the line

$$root \leftarrow x_0 - f(x_0)/f'(x_0)$$

by an expression for computing *root* by the formula in (a) above. Save this function as `mth_root.m`. Note that `mth_root.m` does not need to call any functions `f` or `fp`. Your iterative formula from (a) contains the particular functions `f` and `fp` that are required to compute $\sqrt[m]{R}$. If you modify `Newton` as described here, then each successive computed approximation to $\sqrt[m]{R}$ will be output to your screen. Print a copy of your MATLAB function `mth_root.m` and hand it in.

(c) **(2 points)** Use `mth_root.m` with $m = 3$, $R = 2\pi$, $x_0 = 1$, $eps = 10^{-12}$ and $imax = 20$ to compute an approximation to $\sqrt[3]{2\pi}$. Print and hand in the MATLAB output.

Question #3 - 4 Marks.

(a) **(2 points)** Given input consisting of

- a positive integer n,
- a vector a with $n+1$ entries $a_1, a_2, \ldots, a_{n+1}$,
- a vector y with n entries $y_1, y_2, \ldots, y_n$, and
- a scalar x,

write a MATLAB function with header

```
function p = PolyEval ( n, a, y, x )
```

to evaluate the polynomial

$$\begin{aligned} p(x) &= a_1 + a_2(x+y_1) + a_3(x+y_1)(x+y_2) + a_4(x+y_1)(x+y_2)(x+y_3) + \cdots \\ &\quad \cdots + a_{n+1}(x+y_1)(x+y_2)(x+y_3)\cdots(x+y_n) \\ &= \sum_{j=1}^{n+1} a_j \prod_{k=1}^{j-1}(x+y_k) \end{aligned}$$

using Horner's algorithm. Include a copy of your function M-file in your solution pdf.
Note. Your function should use exactly $3n$ flops (floating-point operations).

(b) **(2 points)** Use your function M-file from (a) to evaluate $p(1.234)$ when $n = 4$ and

$$a = [-1,\ 0,\ 2.33,\ -1.2,\ 2.2]$$
$$y = [-1,\ 1,\ -2,\ -2]$$

Include the call to the function you used to get the result.