ECSE 343: Numerical Methods in Engineering
Assignment 1

Due Date: Feb 12, 2021 at 11:30pm

Submit the assignment via the myCourses submission box. Submit a ZIP file containing the assignment report in PDF format and the MATLAB files.
Show all your work, the methods used should be clearly described, and include all your codes in the appendix of your report.
All your handwritten answers must be scanned, and please submit a single PDF document. You must include all figures/plots, and references in the report.

Question 1

a) The machine epsilon ϵ_m can also be thought of as the difference between 1 and the next higher number that can be stored using floating point representation on a computer. The matlab function `eps()` provides this value. Note that the distance between two floating point numbers is not constant. The following function computes the value of the epsilon

```
function eplison = ep(n)
    eplison = double(1);
    r = eplison/2;
    while (n+eplison) ~= n
        eplison = eplison/2;
        r = eplison/2;
    end
    eplison= eplison*2;
end
```

Use the above function to compute the value of ϵ for different values of n. Use n = 1,2,3,4,5, 7, 8, 15, 200, 1022, and 1023, 1.3e6. Explain the results you obtain.

b) The function provided in part a gives an estimate of the absolute roundoff error when storing a Real number as a floating-point number n (the round-off error is $\epsilon/2$). Write a similar function `rel_ep(n)` that computes the relative error. Call the function for the same values of n used in part a. Explain the difference in the results.

c) Using your knowledge of the floating-point representation find and evaluate an expression that computes epsilon at 1 for a single precision (32bit) floating point number (compare with the matlab function `eps('single')`.

Question 2

The derivative of a function, f(x), can be computed using the *finite difference* method, which can be derived using the first-order terms of the Taylor series expansion of the function, about x_0:

$$f(x_0+h) = f(x_0) + hf'(x_0) + \frac{1}{2!}f''(x_0) + \frac{1}{3!}(x_0)f'''(x_0)h^3 + \cdots \qquad 2.1$$

Rearranging terms to solve for $f'(x_0)$ and ignoring the higher-order terms, we obtain the first order approximation of derivative

$$f'(x_0) \approx \frac{f(x_0+h) - f(x_0)}{h} \qquad 2.2$$

a) Implement the function using the structure given below to compute the derivative of sin(x). Use this function to compute the derivative of the sin(x) at $x_0 = \frac{\pi}{4}$.

```
function der = dydx(x0)
% this fucntion computes the derivative of sinx(x)
% YOUR CODE GOES HERE.
end
```

b) In this part, we will implement the code to compute the derivative of sin(x) at $x_0 = \frac{\pi}{4}$ using the divided differences formula shown in equation (2.2). What is the expression for truncation error?
Use the provided script (*dydx_approx.mlx)*, implement the divided difference formula and the truncation error.
Include plot of the absolute error and the truncation error vs h in your report. Explain the why the error is smooth of the error for large values of h while it is erratic for small values of h.

c) In part (b), at which value of h did you obtain the minimum error. Justify your answer and show all the computations.

d) An another way to compute the derivative is to use the *Central Difference method*. Writing the Taylor series expansion of $f(x_0 - h)$ around x_0, we get

$$f(x_0 - h) = f(x_0) + f'(x_0)\,(-h) + \frac{1}{2!}f''(x_0)h^2 + \frac{1}{3!}f''(x_0)(-h^3) + \cdots \qquad 2.3$$

Subtracting (2.3) from (2.1), and ignoring the higher-order terms, we get

$$f'(x_0) \approx \frac{f(x_0+h) - f(x_0-h)}{2h} \qquad 2.4$$

Implement the code to compute the derivative of sin(x) at $x_0 = \frac{\pi}{4}$ using the divided differences formula shown in equation (2.4) Name your MATLAB script as *dydx_central_approx.mlx.* What is the expression for truncation error? Show the plot of the absolute error and the truncation error vs h.

e) Which difference formula gives a smaller absolute error? Explain.

Question 3

a) Write a MATLAB function name *LU_decomposition.m* to compute the LU factorization using Gaussian Elimination. Use the stencil of function provided below to implement your code.

```
function [L, U] = LU_decompositon(A)
% L  is lower triagular matrix
% U is upper triangular matrix
% YOUR CODE GOES HERE
end
```

b) Write a MATLAB function named *LU_rowpivot.m* to compute the Gaussian Elimination based LU factorization **using the partial pivoting** (row pivoting). The function should take matrix as the input and should output L, U and P matrices as described below.

```
function [L, U, P] = LU_rowpivot(A)
% L  is lower triagular matrix
% U is upper triangular matrix
% P is the permutation matrix
% P*A = L*U
% YOUR CODE GOES HERE
end
```

c) Write the MATLAB functions named *forward_sub.m* and *backward_sub.m* to compute the forward and backward substitutions, respectively. The use the structure of the function shown below

```
function y = forward_sub(L,b)
% y is the vector
% L is the lower triangular matrix
%    YOUR CODE GOES HERE
end
function X = backward_sub(U,y)
% X is the vector
% U is the upper triangular matrix
%    WRITE YOUR CODE HERE
end
```

d) Use the LU decomposition implemented in part (a) along with forward and backward substitution functions written in part (c) to solve the following system of equations. List the values of vector X obtained.

$$\underbrace{\begin{bmatrix} 1e-16 & 2 & 5 & 5 \\ 0.2 & 1.6 & 7.4 & 5 \\ 0.5 & 4 & 8.5 & 5 \\ 0.5002 & 8 & 11 & 97 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{X} = \underbrace{\begin{bmatrix} 400 \\ 5 \\ 18 \\ 95 \end{bmatrix}}_{b}$$

e) Use the LU decomposition with partial pivoting implemented in part (b) along with forward and backward substitution functions written in part (c) to solve the above system of equations. List the values of vector X obtained.

f) Comment on solutions obtained in part (d) and (e), which is more accurate?