In the name of God, the Merciful, the Compassionate

Spring 2018 CE 40_215 **Numerical Analysis Project**

First report deadline: Wednesday, April 18 2018

Phase one deadline: Wednesday, May 09 2018

Phase two deadline: Wednesday, May 30 2018

Project Overview:

In this project, you are expected to provide a MATLAB implementation of all numerical methods studied throughout this course. Technically, you need to design a user-friendly software along with a good looking GUI (containing back buttons and title for each input, also additional features in the GUI helping to make it more user-friendly like help buttons and . . . will be considered as extra marks) due to the software engineering cycle. Thus, the following steps should be taken:

1. Data Gathering
2. Design
3. Implementation
4. Testing
5. error handling
6. Maintenance

You may not be able to accomplish the last step in the desired time, but it is strongly recommended that you do it for your own experience.

The project will be done in 4-person teams. Each team selects a leader who is responsible for any communications between the team members and the TAs or the instructor. Moreover, all project and report files are uploaded by the leader on the course page. Each team should provide two reports. The first report contains the names of the members, the leader and a detailed description of the job scheduling in the team. Note that this report should be more than a simple email notification which is sent to the instructor. A face-to-face delivery will be held subsequently. The second report, which is sent at the deadline of the project, contains a concise description of each member's job and project steps (two pages suffices). Finally, the project files are delivered in either install-able or executable form.

Project Details:

The first page of the software includes a menu with six buttons, one for each of the six chapters. By clicking on each of them, a window is opened. correspondingly it is needed to get the number of mantissa desired by the user in each chapter separately, please note that the number of mantissa entered by the user should be applied in all of the steps of your program's calculations.

Chapter One:

Required Parts:

In this window, the user is asked to enter a stream, an expression comprised of a series of variables and the operators $+$, $-$, $*$, $/$,^ in addition to the parenthesis. In order to process the stream, the software basically needs to recognize the number of the variables. Note that the number of variables does not exceed 6. For simplicity, they are named a, b, c, d, e and f. The user is also requested to enter both exact value and absolute error of each variable.and also, your software should be able to calculate an appropriate error for each variable noticing its value if no error is set by the user (in chopping and symmetric cases, which can be chosen by the user clicking buttons or any other ways). Ultimately, the result of the expression, absolute and relative error of each variable and the expression itself, are calculated and shown to the user. A step-wise representation of absolute and relative errors calculation is required. Note that if the user does not enter the value of the variables, they should be considered parametric.

Bonus Part:

In the case that the base and exponent are both variables (for example a^b), calculating absolute and relative errors has extra mark.

Chapter Two:

Required Parts:

In this chapter, the user is requested to enter a non-linear equation with variable 'x' along with an initial interval except for the 'Fixed Point' method. The user then chooses among a series of numerical methods to solve the equation with. The must-have methods are Bisection, Secant, False Position, Newton Raphson and Fixed Point. in Fixed Point method, you should give your program three inputs, namely the equation, the initial point and the number of iterations. If initial point is not valid, a suitable warning message will be displayed. A section containing the curve and steps of finding the roots should also be included in the window. Furthermore, your software should implement generalized Newton-Raphson method for solving a system of 'n' equations where n is given by the user and the 'n' equations will be entered afterwards. Assuming the variables to be $a_1, a_2, \dots$, the root(s) should be found within an interval specified by the user and shown on a plot. if the equation had more than one root your program should notify the user and find them using the method chosen if desired by them.

Bonus Part:

Implementing Brent's method has extra marks in this chapter. Brent's method is an algorithm combining the bisection method, the secant method and inverse quadratic interpolation, which gives the reliability of bisection but it can be more quick.

Chapter Three:

Required Parts:

In this window, either some points or a function with variable ‘x’ are given as inputs (if the input is a function you are expected to take a number of samples of it as the points) . In case of providing the points, the number of them should also be provided. The user then selects between interpolation and curve-fitting (the output should be in form of a polynomial). If interpolation, the user is asked to specify its type afterwards. Finally, the resulted curve and the points must be shown as the output. In the case that curve-fitting is chosen, if the user enters a function and a set of points, you should compute the y values and continue. If the user does not select a specific function to be fitted to, you should fit it to the best function which has the least error (you can try all the functions and choose the one with the minimum error). The user may also want to enter a specific point; your software then should be able to represent the value of the fitted curve or the interpolated one in this point. The following methods are expected to be implemented for interpolation and curve-fitting parts, respectively:

Interpolation:

- Lagrange
- Newton Divided Differences
- Newton Forward, Backward and Central Differences

Curve-Fitting:

- $y = \alpha e^{\beta x}$
- $y = a \ln x + b$
- $y = \frac{a}{x} + b$
- $y = \frac{1}{ax+b}$
- $y = \sum a_i x^i (i \leq 3)$

Bonus Part:

Implementation of any other type of curve-fitting (excluding the types studied in the course) has extra marks. As an example, trigonometric functions or you can consider the possibility of fitting a multiple rule function to your points. For instance, one solution for the fitted curve might be the following:

$$\begin{cases} 2x+1 & x < 0 \\ 2lnx+3 & x \geq 0 \end{cases}$$

Chapter Four:

Required Parts:

The user may also want to exploit integration or differentiation features of your software. Hence, the data entry here includes a desired interval and a value for ‘h’ as well. A step-by-step representation of all integration calculations should be also displayed for the user. The following integration methods are expected to be implemented:

- Trapezoidal
- Simpson $\frac{1}{3}$ and $\frac{3}{8}$
- Rom-berg (For any n)
- Gauss Legendre (For any n)

For differentiation part, the user enters the desired function, the desired point, the degree of the derivative, the order of error and the value of 'h'. Your software is expected to calculate the derivative using the central methods. Moreover, Richardson Extrapolation is proposed to the user. For Richardson Extrapolation, use the error and the 'h's that user provides for your computations . ($g(h_i)$ should be represented in all methods implemented.)

Bonus Part:
You may also add a new method on your own, named 'Customized Simpson' functioning as bellow: Having divided the interval to two sub-intervals, it integrates on the first and second sub-intervals using Simpson $\frac{1}{3}$ and Simpson $\frac{3}{8}$, respectively. Note that the original interval should be divided in a way that leads to the minimum amount of error. In fact, you should take advantage of Simpson $\frac{1}{3}$ as much as possible. The advantage of this method is that it works for any number of points.

Chapter Five:

For implementing numerical methods for solving ordinary differential equations, the user enters three inputs: the equation, the initial conditions and the value for 'h'. A step wise representation is needed here, likewise. You can get the step as an input from the user as well. The following methods should be offered to the user:

- Taylor (The value of n should be provided as input.)
- Euler
- Modified Euler
- Runge-Kutta(Mid-Point + 3rd Order + 4Th Order)
- Adams-Moulton
- 2Nd order D.E By Euler method and 4Th order Runge-Kutta

Chapter Six:

Required Parts:
Your software also has a feature to solve a system of 'n' equations. The data entry includes a value for 'n' and all 'n' equations which are given afterwards in the form of either a matrix or a set of equations. You are required to implement these functions by yourself. In the Jacobi and Gauss Seidel methods, the initial guess and the number of steps are given as input. In addition, you should modify the matrix to be Strictly Diagonal. Assuming the variables to be $a_1, a_2, \ldots$, the user is then able to choose among the following methods:

- Cramer
- Gausse Elimination

- LU Doolittle
- LU Cholesky
- LU Crout
- Jacobi
- Gauss Seidel

The step-by-step representation is required here as before.
Note that for the methods that require special conditions, your software should be able to check on the inputs given by the user. Furthermore, a warning message should be displayed if the inputs are invalid.
Last but not least, your software is expected to get a matrix with MATLAB syntax to calculate its eigenvalues and the corresponding eigen-vectors. For implementing Power method, the number of steps and an initial guess are given by the user as well.

Bonus Part:
For the first part, converting the invalid input to a valid one, if it is possible, has extra marks. (Notice that using MATLAB's built-in functions instead of implementing computational methods is not allowed in both required and bonus parts)

Other Bonus Parts:

1- Implementing any extra method has extra marks. For this reason, you are encouraged to implement other chapters, e.g. "Solving Partial Differential Equations".
2- Adding special features, such as a control for sketching speed, a step-by-step plotting of the curves and printing output in LaTeX involves extra marks.
3- Replacing MATLAB symbols with genuine mathematical ones in getting input from the user will have extra mark, e.g. instead of MATLAB notations 2ˆ3 and sqrt(2), mathematical notations 2^3 and $\sqrt{2}$ are given as input. This feature of your software is designed for the ease of non-programmer users.
4- Any team whose software is more flexible and user-friendly will achieve extra marks.

Final Notes:

1- All your outputs should be symmetrically rounded off to 'n' floating digits where 'n' is given by the user at the first page of your software. Using VPA for the numbers with more than 16 digits is recommended.

2- Bear in mind that the user of your software is considered as a non-programmer expert, thus there should exist a 'hide' ability throughout your program to let the user decide what feature is displayed to him/her. This ability involves all the plots and step-by-step representations.

3- You are considered to have prepared main page of your software and chapters one, two and three up until the first phase. This phase contains 40% of your score. Furthermore, 10% of your score is devoted to the delivery and modification after the first phase, which is graded in the second phase. That implies you should have fixed the bugs of the first phase up until the deadline of the second phase. The remaining chapters will be done in the second phase. A face-to-face delivery will subsequently be held for the second phase.

Good Luck