BIEN/CENG 2310
MODELING FOR CHEMICAL AND BIOLOGICAL ENGINEERING

HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY, FALL 2022

HOMEWORK #2 (DUE SEPT. 27, 2022)

1. In this problem, you will practice your MATLAB programming skills by computing the Fibonacci series 1, 1, 2, 3, 5, 8, 13, …, defined by the recurrent relation:

$$F_1 = F_2 = 1$$

$$F_i = F_{i-1} + F_{i-2} \quad for \;\; i = 3, 4, 5, 6, \dots, n$$

You may recall that in the MATLAB tutorial video "Functions and Scripts," an implementation using recursion (that is, a function calling itself) was shown. The code is elegant, but not very efficient. It also only outputs the Fibonacci number F_n but does not store the whole series.

(a) Write a MATLAB function that, given a positive integer n, compute the first n Fibonacci numbers $F_1, F_2, F_3, \dots, F_n$, and returns them as a column vector. Try to make it as efficient as possible and do not recompute any value that you have obtained before. Your function definition should be:

```
function F = fiboseries(n)
```

Do not use MATLAB's `fibonacci` function or other direct formulae for Fibonacci numbers (e.g., Binet's formula). Do not use recursion as in the MATLAB tutorial video.

(b) Write another MATLAB script (name it `fiboplot_<LastName>_<FirstName>.m`) to plot the ratio F_i/F_{i-1} versus i, for $2 \le i \le 20$. You may only call the function from Part (a) ***once***, and are ***not*** allowed to use `for` or `while` loops in this script.

(c) Rewrite your function in Part (a), but this time, do not use `for` or `while` loops. Instead, use matrix multiplication and exponentiation. The MATLAB function to construct diagonal matrices `diag` will be useful. Your function definition should be:

```
function F = fiboseries_noloop_<LastName>_<FirstName>(n)
```

DELIVERABLES:

Submit the required plot for Part (b), as a `.jpg` file, together with the script to produce it. Include your name in the title of the plot by the `title` command, e.g., `title('CHAN, Tai Man. HW2 Q1(b)')`. Submit your MATLAB programs for Parts (c), following exactly the definitions specified.

Note: No need to submit anything for Part (a), which will be done in class as a demo.

2. Cramer's Rule (http://mathworld.wolfram.com/CramersRule.html) is a century-old method to solve systems of linear equations, though not a very good one. A system of linear equations can be written in matrix form as:

$$\mathbf{A}\,\underline{x} = \underline{b}$$

where $\mathbf{A}$ is the coefficient matrix, $\underline{x}$ is a column vector of the unknowns, and $\underline{b}$ is a column vector of the constant terms:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \qquad \underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \underline{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Write a MATLAB function that takes in a *square* matrix $\mathbf{A}$ (of any size n), and a vector $\underline{b}$, and solves for $\underline{x}$ using Cramer's Rule, keeping in mind that Cramer's Rule only works when the system has a unique solution. Use the following function definition:

```
function x = cramer_<LastName>_<FirstName>(A, b)
```

where `<LastName>` is your last name, and `<FirstName>` is your first name. You are allowed to use the built-in function `det` to calculate the determinant of a matrix. Try to anticipate exceptions and handle them gracefully. That is, catch such exceptions and print out understandable messages rather than causing MATLAB to crash or spit out some cryptic error message.

Hint : To check your answers, you can simply run the command "`x = A \ b`" which means $\underline{x} = \mathbf{A}^{-1}\underline{b}$. However, instead of actually computing $\mathbf{A}^{-1}$, the inverse of $\mathbf{A}$, MATLAB chooses the most efficient and numerically stable method based on the properties of $\mathbf{A}$. If you are curious, you can read: https://www.mathworks.com/help/matlab/ref/mldivide.html.

DELIVERABLES:

Submit a MATLAB program "`cramer_<LastName>_<FirstName>.m`" which will solve the linear system using Cramer's method. No need to provide any write-up. Note that the TAs will try your program with various linear systems to see if it can handle exceptional cases well.

3. Although we can perform many modeling tasks in Excel, a full-fledged scientific computing programming language like MATLAB can do much more and do so more efficiently. In this problem, we will learn how to write simple MATLAB programs to solve an ODE numerically.

(a) In the last module, we learned how to use Euler method to solve an ODE numerically:

$$\frac{dy}{dt} = f(t, y) \quad ; \quad y(t = 0) = y_0$$

Write a MATLAB function that implements Euler method. The definition should be:

```
[tt, yy] = feuler(fun, y0, tf, h)
```

where `fun` is a user-defined function that should take in two arguments, t and y, in that order, and return $f(t, y)$. The program will simulate the ODE from $t = 0$ to $t = t_f$ with time step h, and return the simulated data points (t_i, y_i) as two vectors `tt` and `yy`, consisting of the t_i's and the corresponding y_i's, respectively.

(b) The Euler method we learned is properly called the "forward Euler" method since it assumes that the slope between (t_i, y_i) and (t_{i+1}, y_{i+1}) can be approximated by the slope evaluated at (t_i, y_i). The formula is:

$$y_{i+1} = y_i + hf(t_i, y_i)$$

Another variant of Euler method is the "backward Euler" method, which uses the slope evaluated at the point (t_{i+1}, y_{i+1}) instead, resulting in the formula:

$$y_{i+1} = y_i + hf(t_{i+1}, y_{i+1})$$

Since we actually do not know y_{i+1} yet, we cannot evaluate $f(t_{i+1}, y_{i+1})$ right away and plug it into the formula. Instead, we need to solve this implicit formula to calculate y_{i+1}. In MATLAB, you can do this with the `fzero` function (see the MATLAB tutorial video on "Root Finding").

Write a MATLAB function that implements "backward Euler" method. The definition should be:

```
[tt, yy] = beuler_<LastName>_<FirstName>(fun, y0, tf, h)
```

(c) In Homework 1, we model the well-stirred tank problem with this ODE:

$$\frac{dC}{dt} = \frac{1}{\tau}(C_{in} - C)$$

with initial condition $C(t = 0) = 0$. Here, we combined V and F into a parameter $\tau = V/F$ called the "residence time" that governs the dynamics of the system.

Write a MATLAB function to solve this ODE by calling the `feuler` function from Part (a), and the `beuler` function from Part (b), and overlay the two numerical solutions in the same plot. For comparison, also overlay the analytical solution:

$$C(t) = C_{in}\left[1 - \exp\left(-\frac{t}{\tau}\right)\right]$$

The function definition should be:

```
[] = stirredtank_<LastName>_<FirstName>(Cin, tau, tf, h)
```

(d) Try a large step size h to better observe the error behaviors of the numerical methods. Which one, forward Euler or backward Euler, seems to be more accurate? Increase the step size further until the forward Euler solution fails to approximate the analytical solution before the concentration C stabilizes. (You should see the forward Euler solution fluctuates wildly, which is clearly unphysical.) Do you see the same behavior for the backward Euler solution? Can you rationalize your observation?

For the written answers to the questions, use the "Text box" function in MATLAB (under the "Insert" menu of the plot) to create a text box in some empty part of the plot, and include your answers there. This way, you don't need to submit a separate file with the written answers.

DELIVERABLES:

Part (a) will be done in a demo in class.

Modify the provided program `euler.m` to implement the backward Euler method for Part (b). Submit your program – remember to rename it "`beuler_<LastName>_<FirstName>.m`".

Submit your program for Part (c) as "`stirredtank_<LastName>_<FirstName>.m`" following the demo in class. Remember to order the input arguments as required. Note that the TAs will try your program with various parameter settings.

For Part (d), submit a plot as a `.jpg` file, overlaying the analytical solution and the numerical solutions by forward Euler method and backward Euler method, with axis labels and a legend to distinguish which curve is which. Choose a large step size so that the differences between the 3 curves are easy to see. Include your name in the title of the figure by the `title` command, e.g., `title('CHAN, Tai Man. HW2 Q3(d)')`. Include your written answers in a text box. Note that the TAs will try your program with various ODEs.