MATLAB Exercise Set 2

FEB22012(X)

September 9, 2018

Making the assignment

You should make this assignment **individually**. You are allowed and encouraged to discuss ideas necessary to solve the assignment with your peers, but you should write the code by yourself. We will employ fraud checking software specifically created for programming classes such as this.

In case we suspect fraud, we can invite you to explain your code. Due to university policy cases of (suspected) fraud must be reported to the exam committee. The exam committee will then decide on possible disciplinary actions.

The deadline for this assignment is **Wednesday, 19th of September 2018, 22:00, CEST**.

Grade and Scores

The grade of your assignment will depend on two important criteria:

1. Correctness: does your code work according to the specification in the assignment?
2. Code Quality: is your code well written and understandable?

For this assignment you can earn **10 points** in total and counts for 10% towards the final grade of the course. Specific parts of the assignment count as follows:

Part of Assignment	Points
Plotting	1
Growing a Maze	1
Selling Cheese	2
Optimal Buy and Sell Portfolio	2
Picking Dollars	2
Code Quality	2
Total	10

When we ask you to implement a function using a **specific** algorithm, you will get no points if you use a different algorithm. Your are also not allowed to use functions from MATLAB Toolboxes as these are not available in Autolab.

Handing in your assignment

- You need to hand in your assignments via **Autolab**. If you hand in an exercise more than 35 times, you wil l get a **strike**. After three strikes, you can only obtain 75% of the correctness score for subsequent exercises you hand in more than 35 times.
- When **Autolab** tells you that your assignment is not working correctly, i.e. you get less than 100 points, you will get 0 points for that assignment towards your grade. Your code should produce the correct answer in **all** situations, not just in some.
- The scores generated by **Autolab** only give an indication of your possible grade. Final grades will only be given after your code has been checked manually.
- Deliberate attempts to trick Autolab into accepting answers that are clearly not a solution to the assignments will be considered *fraud*.

Code Quality

Part of your grade will also depend on the quality of your code.

- Your code should be well structured with sensible methods and parameters.
- Give clear, understandable names to methods and variables. Something like `cost` should be preferred to `c`.
- Use consistent layout, formatting and indent style for your code.
- Write comments where appropriate. In MATLAB, functions should be documented. Directly after the function signature you should write the name of the function in upper case, with a short description of the function. The second line of the comment should contain the function signature. Please look at the lecture slides for an example.
- Try to keep code as clean and elegant as possible.
- Avoid magic numbers, define constants.

For an in-depth style guide on writing clean MATLAB code, Mathworks provides a document with many tips and suggestion you can adopt at this page: `https://nl.mathworks.com/matlabcentral/fileexchange/46056-matlab-style-guidelines-2-0`. Note that you need a Mathworks account and login to obtain this document.

General Remarks

- For many operations, such as summing up the elements in a matrix or vector, you can either write a loop or use a standard function in MATLAB that does it for you. Clever use of such functions can save writing a lot of code.
- The last two exercises this week are quite challenging. Please be aware that these are also the final exercises of the MATLAB module of the course and resemble some of the data processing problems you may run into when you need to program for a thesis project.
- The Windows version of Octave can be very slow and unresponsive the first time you call a plotting function. If you wait some time (up to a minute), the plot will become visible and subsequent plotting calls will hapenn very fast.

Plotting

In this exercise you will practice plotting in MATLAB. The goal is to create a function which generates several plots depending on the value of the input variable `flag`. The function should have the following signature:

```
function plotting(flag)
```

In this function, the `flag` variable can be an integer from 1 to 7. For each number, a different plot has to be created with the `plot` command. Unless otherwise specified, no additional styling of the plot should be done. Note that the plot can be marked as incorrect by Autolab if the plot style differs from the expected plot style. The expected plots for each value of `flag` are:

1. A plot of the function $f(x) = x^2 - 3x + 8$, with $x \in [-5, 5]$. Use 11 evenly spaced points on the interval for x.
   Hint: You may want to use the `linspace` function. See `help linspace` for help.

2. A plot of the polar function $r = \sin(\theta) + \sin^3(5\theta/2)$ for $\theta \in [0, 4\pi]$ in 500 evenly spaced points. First compute the r for each θ, and then convert the pairs of polar coordinates (r, θ) to regular coordinates (x, y) using the equations $x = r\cos(\theta)$ and $y = r\sin(\theta)$.

3. A plot of the function $B(t) = 10^3 \cdot 1.05^t$ with $t \in [0, 10]$, using 31 evenly spaced points for t. Label the horizontal axis *time (days)*, the vertical axis *bacteria*, and give the plot the title *Bacteria growth over time*.

4. A plot of the functions $f(x) = x^2 + \frac{1}{2}$, $g(x) = \frac{\sin(4x)}{8} + \frac{1}{4}$ and $h(x) = -\exp(-x^2/3)$ on the interval $x \in [-5, 5]$, using 101 points for x. Make sure that the vertical axis is restricted to the interval $[-1, 1]$ and the horizontal axis to $[-5, 5]$. Style the lines as follows: $f(x)$ is a red solid line, $g(x)$ is a green dashed line, and $h(x)$ is a black dash-dotted line. *Hint: Note that `'.-'` and `'-.'` are different line styles in* MATLAB.

5. A three dimensional surface plot of the function

$$f(x, y) = \frac{4x^2 - x^2y^2 - 1}{(x^2 + y^2 + 1)^2},$$

   with $x, y \in [-10, 10]$. Use 101 evenly spaced points for both x and y.

6. A histogram of all the primes below 10^6, using 100 equally sized bins.
   Hint: you are allowed to use the `primes` function.

7. For a constant r and a stating point $y_1 = \frac{1}{2}$, we can iterate the function $y_t = r \cdot y_{t-1} \cdot (1 - y_{t-1})$ to cmpute y_2, y_3, y_4, and so on. This function is called the logistic map, and it shows very different behavior for different values of r. For example, the next two values for $r = 1.5$ and $y_1 = \frac{1}{2}$, we get $y_2 = 0.375$ and $y_3 = 0.3515625$, and for $r = 1$ the next two values are $y_2 = 0.25$ and $y_3 = 0.1875$. Create four subplots of the first 15 steps of this process where the x-axis contains the steps 1 to 15 and the y-axis contains the y corresponding to that step. In the top-left subplot use $r = 0.7$, in the top-right subplot use $r = 2.3$, in the bottom left subplot use $r = 3.4$ and in the bottom right subplot use $r = 3.7$. If you want to learn more about the logistic map, you can watch the following Numberphile video:
   https://www.youtube.com/watch?v=ETrYE4MdoLQ.

This exercise must be handed in through Autolab. More information about handing in your exercise through Autolab can be found on Blackboard.

Growing a Maze

A *cellular automaton* consists of a grid of cells where each cell is either *on* or *off*. In a single time step a new generation is computed based on the previous generation and a fixed rule. The most famous example of a cellular automaton is *Conway's Game of Life*. In this exercise we will implement a variation which results in maze-like structures.

Create a function with the following signature

```
function [B] = grow_maze(A, steps)
```

The input `A` is a matrix with the initial state of each cell, where a value of `0` indicates a cell is off and a `1` indicates a cell is on. The input `steps` indicates how many new generations need to be computed.

A new generation can be computed with the following rules:

Survival If a cell is on and it has at least one and at most four neighbors which are on, it will be on in the new generation.

Birth If a cell is off and it has precisely three neighbors which are on, it will be on in the new generation.

Death All other cells will be off in the new generation.

The neighbors of a cell are all cells which are directly adjacent to it, including the diagonal cells. Most cells have eight neighbors, while those at the sides of the grid have five neighbors and the cells in the corners only three neighbors.

Hint: If you need far more than 30 lines of code for this problem, you may need to reconsider your approach. Also, make sure to take special care at the boundaries of the matrix. One way to deal with this is to make use of the min and max functions in MATLAB.

For testing purposes, you can create a random 0-1 matrix using the following code:

```
A = randi([0, 1], 10, 10)
```

When you've finished writing your program, you can animate the process in MATLAB with the following code (note that animation works slightly different in Octave):

```
B = [];
while ~isequal(A,B)
    B = A;
    A = grow_maze(A,1);
    imshow(A,'InitialMagnification','fit');
    drawnow;
    pause(0.25);
end
```

This exercise must be handed in through Autolab. More information about handing in your exercise through Autolab can be found on Blackboard.

Selling Cheese

The cheeses of *Trendy Cheese Inc.* are extremely popular at the moment. Every Saturday of the week, they can sell their cheeses at a local market. Unfortunately the van they use to transport the cheeses to the market cannot take the full stock: the weight of all cheeses would overload the van. So every week they need to make a selection. Since they can safely assume that their cheeses will sell out, they want to select the cheeses which will maximize profit. If a cheese would not fit fully into the van, it can be cut to an arbitrary weight.

You need to implement a function with the following signature:

```
function [profit, cheeses] = select_cheese(weights, profits, capacity)
```

Here the input `weights` is a vector of length n that contains for every cheese the amount of weight currently in stock. The input `profit` is a vector of length n containing for every cheese the profit that the company will receive when the full stock of that cheese is sold. The input `capacity` is the capacity of the van in units of weight. The output `profit` should contain the maximized profit the company can make with the cheeses transported by the van. The output `cheeses` should contain a vector of length n which contains for every cheese the amount of weight that should be put in the van.

Example: The call `select_cheese([4 3 3 2], [6 4 7 1], 6)` should yield a profit of 11.5 and a selection of cheese equal to `[3 0 3 0]`. This implies that the van will carry 3 weight units of cheese type 1 and 3 weight units of cheese type 3.

Hint: to find the correct answer, you need to fill up the capacity of the van in descending order of profit per unit of weight. The function `sortrows` can be very useful to do this.

Hint: After sorting the matrix it can be difficult to know which ratio belonged to which cheese. It can be useful to include indices of the cheeses in the matrix before sorting. Approaches that try to look up the original cheeses based on just their weight, profit or ratio will be extremely difficult to implement correctly and are thus discouraged.

Remark: You need to implement the greedy algorithm for this assignment. It is not allowed to use a Linear Programming based approach. The Optimization Toolbox is not available within Autolab.

This exercise must be handed in through Autolab. More information about handing in your exercise through Autolab can be found on Blackboard.

Buy & Sell Portfolio

Last week your Wall Street boss asked you to figure out the optimal buy and sell times on historic stock price data, and the maximum profit that could have been obtained. This week, he's back with another problem: find out the optimal buy and sell times and the maximum profit on a whole portfolio of stock prices! There's just one problem though, the data he has prepared for you will need some formatting because not all stock prices start and end at the same date.

You are given three filenames which correspond to existing CSV files. Each of these CSV files contains data in the following format:

```
16679,602.36
16678,627.54
16675,632.82
16674,639.4
16673,610.35
16672,614.91
16671,573.0
16668,639.78
16667,655.46
```

In this data the first column is a day ID and the second column is the price of the stock on that day. Note that in this data the day column is a translation of an actual date to a day ID[1]. On some days (such as weekends and special days) the market is closed, and there can be missing lines in the data.

Your job is to write a function which does the following:

- Reads the data from each CSV file into a 2-column matrix
- Finds the largest common sequence of days among all files (i.e. the range of days for which there are stock prices in all files)
- Calculates the maximum profit that could be obtained from buying all three stocks simultaneously at one point in time, and selling all three stocks simultaneously at another point in time. Just like last week, you must buy and sell once.

Your MATLAB function must have the following signature:

```
function [buyday, sellday, profit] = buysell_portfolio(file1, file2, file3)
```

Here, the buyday and sellday variables must correspond to the ID of the day on which the stocks must be bought or sold to obtain the maximum profit. For this exercise you can use the function `optimal_buysell` which you created last week. Make sure to include an `optimal_buysell` function in your `buysell_portfolio` program.

Hint: Make sure you download the handout available on Autolab (on the left, under Options), and save the three .csv files in your working directory, so you can test your function locally. You should then be able to test your program with:

```
buysell_portfolio('AAPL.csv', 'FB.csv', 'GOOG.csv')
```

[1]For instance, 16679 corresponds to Sep. 1, 2015. If you're curious, you can convert dates using this website.

Hint: To compute the common range for the series you can assume that the same weekend days are left out in all files. Therefore, you only need to find a common start time and end time to define a range.

Remark: We recommend to plot the data in the CSV files to get a feeling of what it looks like.

This exercise must be handed in through Autolab. More information about handing in your exercise through Autolab can be found on Blackboard.