**Exercise #5 – Functions**

Tutor in charge of this HW: Lior Faeyrman. Questions will be answered through the Moodle forum or at the reception hour (27.6 14:00-15:00 at FGS room B).

Read **carefully** the following instructions, and don’t hesitate to reach us (preferably through the forum) if something is not clear.

**Moodle instructions:**

1. Before submission, you need to create a team through the “Team Work” window. You can team with different students for every HW, but even if you don’t change your team – you still need to do this step for every assignment.
2. Submit the .m file directly to the Moodle. Only one member of the team need to submit – but **all** of you should make sure that you see your submitted file in your Moodle!

**General Matlab instructions:**

1. **Do not submit a script that does not run properly**. To check that your script runs without errors, make sure your workspace is cleared, and then run the script (press F5). If you are reading data, avoid full paths (“C:\My\_computer\HW\data.mat”), as they will return an error when you run the code on a different machine (i.e. the computer of the TA who checks your HW ☺).
2. Name your script "hw5\_< ID number 1>\_< ID number 2>.m".
3. Start your script with a header – a few lines of comments which describe your script. The header should contain **all** of this information: your names, IDs, HW number and the Matlab version (e.g. (R2021a) you wrote your code on.
4. Divide your code into **sections** according to the questions, or any other reasonable division (use the '%%' sign). At the beginning of each question, write its number and a short explanation of your answer – e.g. what the code does, etc.
5. If the question involves printing a result to the console, you should **also** print the question number: You can use “*disp('\*\*\* Question ## \*\*\*')*”.
6. **Document** your work with comments in the script (use the '%' sign). Over-documentation is better than under-documentation: we want to see that you understand how your code is working at each step.
7. Give your variables **meaningful names**.
8. For every question that has a defined output (a number, a figure, etc. – not an “open question”), **read carefully** the HW instructions and make sure that your output is **exactly** what we asked, and what you would expect it to be. This is very important also for you in your “debugging” procedure. For example, if you expect a scalar output but your code returns a matrix (and vice versa), you clearly have a mistake somewhere… ☺
9. **Do not** use "magic numbers" – **every** number that appears in your script should be assigned to a variable, preferably at the beginning of the code section.
10. Do not calculate anything implicitly, but make your code calculate everything explicitly. For example, if you have a vector with the values [2, 5, 3, 1] and you need to find its largest value, you **cannot** write “*disp(5)*”.
11. **Avoid** “code duplication”! Code duplication is copying & pasting lines of code with minimal changes. Instead, use matrix and vector operations, and loops.

**Specific instructions for this HW:**

1. This HW focuses on creating functions. Functions are pieces of code which work independently, and solve small tasks. Every function should do **one** thing.
2. Every function should include **proper documentation**. The first line is a short description of the function. Then, beginning with the next line, a longer documentation follows. This documentation must include explanations about the inputs and outputs of the function, and their correct types (scalar/vector, numeric, char, etc.).

Example function header:

*function [output1,output2] = function\_name (input1,input2)*

*% function\_name does XXX*

*% function\_name gets input1 and input 2, and returns output1 and output2.*

*% ….. (more explanation if needed)……..*

*% Input1 (TYPE): aaaaaaa*

*% Input2 (TYPE): bbbbbb*

*% Output1 (TYPE): ccccc*

*% Output2 (TYPE): dddd*

1. Every function must include **input checks** for all its arguments. Input checks should include type validation (as in point 2), and also question-specific conditions (positive numbers, integers, etc.).
2. You should submit **only one file**, which will be a function by itself (primary function). Therefore, the first line in your code should be:

*function hw5\_< ID number 1>\_< ID number 2>()*

When you are asked to create additional functions, they should be inside this primary function (nested functions).

**Question 1**

Create a function that imitates the function *subplot* in matlab, call it *mySubplot* (without using subplot of course). The function should open an axis in the correct size and position according to the inputs. The inputs should be the number of rows and columns of axes in the figure, and the number of the current axis (just as in subplot).

1. Make sure to write errors to the user if there are invalid inputs (NANs for example).
2. Use the built-in function *axes*. You can specify the size and position of the axis by the ‘Position’ argument: *axes(‘Position’,[x\_position, y\_position, width, height])*. Note that the numbers in the position vector are in proportions of the total size of the figure (the boundary of the windows is [0 0 1 1]).
3. You should leave margins from the left, right, top and bottom. These margins don’t have to be in the same size, but choose something that looks nice. You could use *get(gcf,'DefaultAxesPosition')* to get Matlab’s default values.
4. You should have space between the axes (two subplots should have space between them). There are multiple ways to choose a proper spacing: it could be proportional to the number of subplots, or to their size, or even be a fixed number. No matter what method you choose, the output should look good (spacing not too small or too large) both for the 1X1 case and up to 10X10 subplots.
5. Hint: remember that for n columns of subplots, we need n-1 spacings. Also, the width of a single subplot should be the total width that remains after removing both margins and spacings. For example, for 1X3: *axis\_width = (1 – 2\*margins – 2\*spacing) / 3*
6. Your function should create a single axes, at the correct position given by the third input, just like in subplot. However, if you prefer, your index can work based on columns and not rows (just like with regular matrix indexing).

**Question 2**

1. Create a [function](https://www.mathworks.com/help/matlab/matlab_prog/nested-functions.html) named *myFactorial*(n) which calculates the factorial of the number n and prints its value. The factorial is defined as the product

You should program this function using recursion only - without any loops, matrices or other Matlab functions. Make sure that you have a proper “stopping criterion”, and proper input checks. **Hint**: to program a recursion in Matlab, simply call your function inside itself.

1. Call the function with n=10, and print the result.

**Question 3**

1. Create a [function](https://www.mathworks.com/help/matlab/matlab_prog/nested-functions.html) named *calcTailorExp*(x,N) which gets as input a number x and the upper limit N (where n=0,1,2,…,N), and calculates the following series:

You should program this function using recursion without any loops, matrices or other Matlab functions. You should use the factorial function that you wrote in the previous question.

1. Assume x=2, what is the minimal value of N that gives you a good approximation (with an error of less than 0.005) for an exponent (This is a matlab course, not a math course.. we want to see the code, not the final result). Write a script that checks which order N the error is less than 0.005 and writes the error and the order N.

**\* NOTE-** we know not all of you know what Tailor expansion is. All you need to know is that in our case it is a way to approximate with a sum of numbers. If you add more and more numbers from this series you get closer and closer to the real number (the error will decrease).

**Question 4**

Use the function you built in question 1 to plot the results you got in question 3.

You should have N subplots, where N is the number you got in question 3.2. In each subplot you should plot two things:

* 1. The real number as a horizontal line
  2. The approximated numbers based on *calcTailorExp(n)*, where n are all numbers between 1 until the current number of the subplot. Each approximation will be a different point in the graph. Its x-value should be the number n (1 for the first point, 2 for the second and so on..), and its y-value should be the number you calculated for this n.

Add a title to each subplot ‘approximating e with n=4’ in the fourth subplot for example.

The Y axis of all subplots should be the **same** and start with zero until ceil()

For example, if N is 4 (based on the result you got in q3.2) you will have 4 subplots, where:

* subplot 1: the real number (horizontal line) and *calcTailorExp(2,1)* (in total 1 points in the graph: where the horizontal axes x=1, and vertical axes y= *calcTailorExp(2,1)*).
* subplot 2: the real number (horizontal line), *calcTailorExp(2,1)* and *calcTailorExp(2,2)* (in total 2 points in the graph: where the first point is the same as in the previous graph and the second is: x=2, y= *calcTailorExp(2,2)*).
* subplot 3: the real number (horizontal line), *calcTailorExp(2,1)*, *calcTailorExp(2,2)* and *calcTailorExp(2,3)* (in total 3 points in the graph: where the first two points are the same as in the previous graph and the third is: x=3, y= *calcTailorExp(2,3)*).
* subplot 4: the real number (horizontal line), *calcTailorExp(2,1)*, *calcTailorExp(2,2)*, *calcTailorExp(2,3)* and *calcTailorExp(2,4)* (in total 4 points in the graph: where the first three points are the same as in the previous graph and the fourth is: x=4, y= *calcTailorExp(2,4)*).