CS5810 Assessed Coursework 2

This assignment must be submitted by November 25th, 2022 at 10:00
Feedback will be provided within ten working days of the submission deadline.

Learning outcomes assessed

This coursework will test some concepts of Matlab programming for data analysis, including writing scripts, functions, plotting, manipulating strings and using cell arrays and structure variables.

Instructions

Submit your files through Moodle – follow the link marked "Click here to submit coursework 2" on the Moodle page for CS5810.
The files you submit cannot be overwritten by anyone else, and they cannot be read by any other student. You can, however, overwrite your submission as often as you like, by resubmitting, though only the last version submitted will be kept. Submission after the deadline will be accepted, but it will automatically be recorderd as being late and is subject to College Regulations on late submissions.

IMPORTANT: In this assignment, exercises 1, 2, 3, 4, 5, 6, 8, and 9 require you to write functions, while exercise 7 requires you to write scripts. For this assignment you will have to submit:

- A file containing function *waferstore* required for exercise 1.
- A file containing function *uniqueword* required for exercise 2.
- A file containing function *mysin* required for exercise 3.
- A file containing function *buildrandomstrings* required for exercise 4.
- A file containing function *calctrianglearea* required for exercise 5.
- A file containing function *mytemperatur*e required for exercise 6. This exercise also requires you to write another function which you can write in an extra file with a name of your choice.
- Two scripts *myevalue1* and *myevalue2* required for exercise 7.
- A file containing function *sumcomplex* required for exercise 8.
- A file containing function *wordscount* required for exercise 9.

If you will write any other extra function or script as part of your solution for these exercises, you should also submit these. For these extra functions or scripts, you can choose the file name.

A zip file containing a template for each of these files is provided on the course page on Moodle. You need to insert your code for each exercise where indicated and submit them. The file "RedRidingHood.txt", which you will need for exercise 9, is also provided in the same zip file.

Please do not change the above file names in your submission.

All the work you submit should be solely your own work. Coursework submissions are routinely checked for this.

EXERCISE 1 (5 marks)

A silicon wafer manufacturer stores, for every part in their inventory, a part number, how many there are in the factory, and the cost for each part. Create a function called “waferstore”, that will:

- Take as input arguments 3 vectors, one containing the part numbers, one containing the quantities and one containing the costs of the parts in the inventory.
- Build an array of structs containing, for each part, number, quantity and cost. If displayed, the array of structs could look something like this:

```
>> parts
parts =
1x3 struct array with fields:
    partno
    quantity
    costper

>> parts(1)
ans =
      partno: 123
    quantity: 4
     costper: 33

>> parts(2)
ans =
      partno: 142
    quantity: 1
     costper: 150


>> parts(3)
ans =
      partno: 106
    quantity: 20
     costper: 7.5000
```

- Provide the array of structs as an output argument to the calling script or function.
- Print the part number and the total cost (quantity of the parts multiplied by the cost of each) in a column format. For example, if the variable `parts` stores the previous values, the result would be:

```
123 132.00
142 150.00
106 150.00
```

- Your function will need to check that the 3 vectors provided as inputs are of equal size and provide the user with an appropriate error message in case they are of different sizes.

EXERCISE 2 (3 marks)

Write a function called “uniqueword” that will output a series of 5 unique words. When the function is called, a string is passed as the only input argument. The functions adds an integer to the end of the string, and returns the resulting string. The first time the function is called it will add 1 to the word, and afterwards, every time the function is called, the integer that it adds is incremented by 1, until it reaches a value of 5. Further calls to the function will create an error message alerting the user that 5 words have already been created. Here is an example of calling the function for the first 3 times:

```
>> uniqueword('hello')
ans =
hello1
>> myvar = uniqueword('John')
myvar =
John2
>> another_var = uniqueword('mouse')
another_var =
mouse3
```

EXERCISE 3 (5 marks)

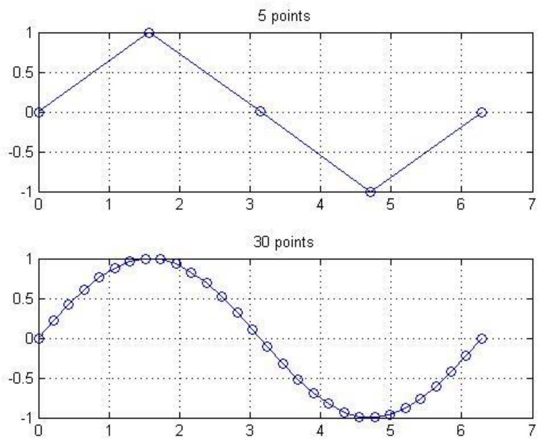
Write a function called "mysin" that will plot the sin function with a different number of points between 0 and 2 Π.

- The function will receive two arguments, which are the number of points to use in two different plots of the sin function.
- The function will present the user with simple menus where he/she can choose a colour among 'red', 'blue', or 'green' and a style for the points among 'circle' or 'star'.
- The function will then create two plots of the sin function with the two different sets of connected points. These two plots need to be within the same figure and need to show in their title the number of points used.

For example, the following call to the function:

```
>> mysin(5,30)
```

in which the user chooses colour 'blue' and style 'circle' will result in a figure similar to the following one, in which the first plot has 5 points altogether in the range from 0 to 2 Π, inclusive, and the second has 30.



EXERCISE 4 (4 marks)

Write a function called “buildrandomstrings” that will receive as input an integer *n*.

- If *n* is positive: it will create and return a cell array with strings of random characters of increasing lengths, from 1 to *n*. Each string will be constituted by the previous random string plus an extra random character.
- If *n* is negative: it will create and return a cell array with strings of random characters of decreasing lengths, from |n| to *1*. Each string will be constituted by the previous random string minus the last character.

Here is an example of running the function for 2 different inputs:

```
>> buildrandomstrings(4)

ans =

    'x'    'xg'    'xgk'    'xgke'


>> buildrandomstrings(-3)

ans =

    'jdu'    'jd'    'j'
```

EXERCISE 5 (6 marks)

The distance between any two points (x_1,y_1) and (x_2,y_2) is given by:

$$\text{distance} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

The area of a triangle is:

$$\text{area} = \sqrt{s*(s-a)*(s-b)*(s-c)}$$

where a, b, and c are the lengths of the sides of the triangle, and s is equal to half the sum of the lengths of the three sides of the triangle.

Write a function called “calctrianglearea” that will receive as input argument a *nx2* matrix of real numbers, representing *n* points in 2D space.

The function will then calculate the areas of the triangles whose vertices are the sets of 3 consecutive points in the matrix, starting with the first point and using each point only once (that is, it will first calculate the area of the triangle whose vertices are points 1, 2, 3; then it will calculate the area of the triangle whose vertices are points 4, 5, 6; and so on).

The function will provide as output argument a vector containing the area of each triangle.

The function will write a message to the user explaining how many triangle areas have been calculated; the message will also contain the coordinates of any points which were not used (in case *n* is not a multiple of 3).

[Hint: it might be useful to create a separate function that calculates the length of the side formed by any two points (the distance between them)].

EXERCISE 6 (3 marks)

Write a function called “mytemperature” for temperature conversions between Fahrenheit and Celsius.

- mytemperature will receive as input arguments two numbers, representing the minimum and maximum temperatures in degrees Fahrenheit.
- mytemperature will provide as output argument a matrix M with 2 columns: the first column will contain the temperatures in degrees Fahrenheit, from the minimum to the maximum, in steps of 1 degree; the second column will contain the corresponding temperatures in degrees Celsius. The conversion is:
  C = (F – 32) * 5/9.
- mytemperature will also call another function (you can chose any name for it), which will receive as input argument the matrix M and will print in the command window two columns: the first column containing the temperatures in degrees Fahrenheit ordered from the minimum to the maximum, and the second column containing the corresponding temperatures in degrees Celsius.
- Note that the input arguments can be provided to mytemperature in any order; that is mytemperature(0, 10) and mytemperature(10,0) will generate the same output.

EXERCISE 7 (4 marks)

The inverse of the mathematical constant *e* can be approximated as follows:

$$\frac{1}{e} \approx \left(1 - \frac{1}{n}\right)^n$$

Write a script “myevalue1” that will loop through values of *n* until the difference between the approximation of *e* and the actual value of *e* (that is, the built-in value of *e*) is less than 0.0001. The script should then print out the built-in value of *e* and the approximation to 4 decimal places, and also print the value of *n* required for such accuracy.

The value of the constant *e* also can be approximated by:

$$e = \sum_{n=0}^{\infty} \frac{1}{n!}$$

Where n! indicates the factorial of *n*. Write a script "myevalue2" that will provide an approximation of *e* using this formula such that the difference between the approximation and the actual value of *e* is less than 0.0001. The script should then print out the built-in value of *e* and the approximation to 4 decimal places, and also print the value of *n* required for such accuracy.

EXERCISE 8 (4 marks)

A complex number can be expressed in the form *x+yi,* where x and y are real numbers and *i* is the imaginary unit which satisfies $i^2 = -1$. Given a complex number *x+yi*, *x* is called the real part and *y* the imaginary part. The sum of two complex numbers is a complex number whose real part is constituted by the sum of the real parts and whose imaginary part is constituted by the sum of the imaginary parts, that is: *(a+bi) + (c+di) = (a+c) + (b+d)i* . In this exercise you will write a function “sumcomplex” that will calculate sums of complex numbers.

sumcomplex will receive as input arguments 3 vectors. The first two vectors, R and I, should be of equal length, *n*, with R containing real parts and I containing the imaginary parts of *n* complex numbers. sumcomplex will build an array of structs which will store one complex number per struct, each separated into real and imaginary part. Each struct should look something like this:

```
>> imaginaryNumber
imaginaryNumber=

1x2 struct array with fields:
    real
    img
```

The third vector, S, will be a vector of indices, indicating which of the *n* complex numbers should be added. For example S = [1, 3, 4] indicates that the first, third and fourth complex numbers should be added.

sumcomplex will perform the addition and provide two output arguments. The first output argument will be the array of structures containing all the complex numbers. The second output argument will be the result of the sum as specified in vector S.

sumcomplex will also check that the length of vectors R and I is the same and provide to the user a proper error message if they are of different length.

sumcomplex will also check that S does not contain indices larger than *n* and provide to the user a proper error message if necessary.

EXERCISE 9 (6 Marks)

Write a function "wordscount" that will read the text file "RedRidingHood.txt" provided in the zip file on the Moodle page. Wordscount will receive as input arguments an integer *n* and a word *w* and return two output arguments:

- The first output argument will be a cell array containing the *n* most frequent words in the text, together with the number of times they appeared.
- The second output argument will be the number of times the word *w* appears in the text.

wordscount will need to be case-insensitive, that is the words "Red" and "red" should be counted as the same word. If several words appear the same number of times, then any of them can be provided in the output. You can assume that the word *w* is always present in the text (no need for error checking).

[Hint: the Matlab function textscan is useful for reading text.]

Marking Criteria

This coursework is assessed and mandatory and is worth 40% of your total final grade for this course.
In order to obtain full marks for each question, you must answer it correctly but also completely, based on the contents taught in this course.
It will be important to provide input and output arguments to the functions as requested in the exercises. Unless explicitly specified in the exercise, avoid using the "input" function or printing outputs on the command window.
Marks will be given for writing elegant, compact, vectorised code, avoiding the use of "loops" (*for* or *while* loops) where possible, and including comments.