Numerical Methods (NUM101) — Coursework 1

This coursework consists of three smaller parts, (a), (b) and (c), which combine for 17% of the credits for this unit. The maximum number of marks for this coursework is 17.

Deadlines and marks

part	marks	published	deadline	hand-in or upload?
(a)	5	now	24 Feb (Thu)	10:30am upload Victory Assignment `CW1(a)`
(b)	5	24 Feb	03 Mar (Thu)	10:30am upload Victory Assignment `CW1(b)`
(c)	7	24 Feb	14 Mar (Mon)	11:30pm upload Victory Assignment `CW1(c)`
(abc)			14 Mar (Mon)	hand-in of **all** printouts at CAM office

Coursework 1 is not directly related to *Numerics* but it gives you the opportuniy to practise your generic programming skills.

Instructions and rules

- Material to be handed in to CAM: printouts of all program codes for parts (a), (b) and (c) with comments.
- Material to be uploaded to Victory: Matlab scripts and functions only. See intructions for each of the parts.
- Marks will be awarded for your work only if both, the electronic upload and the printed hand-in from the CAM office, are present. The uploaded code will be tested by me and receive a score that is provisional until I mark the printout.
- Scripts and functions that generate Matlab errors receive provisionally 0 marks. The student then has to demonstrate in the lab session that the error is minor to justify partial credit.
- Credit for part (c):

  100% code performs computation correctly and efficiently, is well structured and commented;

  ≥80% code performs computation correctly and efficiently

  ≥60% code performs computation correctly but has problems[1];

  ≥40% code does not perform computations correctly but could be made to work with minor corrections;

  ≥20% the intentions behind the code are discernible with some effort.

- For questions, clarifications and further help contact:

Jan Sieber (`jan.sieber@port.ac.uk`, office LG.146).

[1] for example, the function works correctly most of the time but fails for some valid arguments

Part (a)

Preliminary programming exercise — logic, loops and if-else **5 marks**

Set-up steps:

1. Create a new folder for coursework 1 (call it, for example, `CW1`),
2. download the file `GenerateCw1.p` from Victory folder `Coursework1->part (a)` into your new folder,
3. change Matlab's working directory to your new folder,
4. execute in Matlab

```
clear;
R=GenerateCw1(XXXYYY); % replace XXXYYY by your student ID!
```

   where you replace `XXXYYY` with your six-digit student ID.
5. The command has created two files in your current folder, `cw1.m` and `R.mat`. The file `cw1.m` is a template for your script containing the questions. Insert your answers into this file and upload the completed file `cw1.m` as an attachment to the Victory assignment `cw1a`.

The other file, `R.mat` contains a table listing football results from 14 leagues from the season 2004/2005 (downloaded from `footballdata.co.uk`). The league and team names don't matter for the question but you can look them up on Victory. The columns have the following meaning (explained for row 1):

```
>> R(1,:)
     1     1     2     2     0     6     8     4
```

Column	meaning	in the example
1	league	1==Belgian league,
2	home team	1==Mechelen,
3	away team	2==Anderlecht,
4	goals of home team	2
5	goals of away team	0
6, 7, 8	day, month, year of game date	6 August 2004.

In short, this row means that in the Belgian league, on 6 August 2004, Mechelen (the home team) beat Anderlecht (the away team) 2-0.

In the file `cw1.m` you find 6 questions about the table, which you have to answer using Matlab commands. Question 1 is answered already to show you how to do this:

```
%% Question 1 (with example answer)
% How many goals have been scored all in all?
goals=R(:,4)+R(:,5); % add up goals for each game
res{1}=sum(goals);   % add up goals for all games, assign to res{1}
```

Before you submit your file to Victory make sure that your script runs without genereating errors and that at the end the variable `res` is defined. For example, after executing `cw1`, if you type `res{3}` on the command line (note the curly braces!) it should give your answer to question 3. If you don't know the answer leave the entry for `res` empty (for example, `res{5}=[];`).

Hint Useful Matlab constructs and functions (look up their help or Graham's Matlab tutorial if you don't know them): `sum`, `max`,`length`, `find`, `unique`, `accumarray`, `for` loops, `if`–`else`–`end` branching.

Part (b)

Preliminary programming exercise — making your code universal **5 marks**

Answer exactly the same questions in the file `cw1.m` that you have answered already in part (a). However, make sure that your code gives the correct answer also for the table `R.mat` for the season 2005/2006 in Victory folder `Coursework1-> part (b)` and every other season. Your script will be tested using tables, which, for example,

- may be from a different season,
- may contain different leagues (also fewer or more),
- may contain fewer or different games, or
- may have a different ordering of their rows.

The column format will stay the same.

For example in question 1

```
%% Question 1 (with example answer)
% How many goals have been scored all in all?
res{1}=13868;
```

would have been correct for part (a) but is very likely to be wrong in general. So, go through your answers, and check in which places you have inserted "magic" numbers (such as the 13868 above) Replace these magic numbers with *robust* code, that is, code that works for all tables (in the example `res{1}=sum(R(:,4)+R(:,5));`). For example, you should not make assumptions about

- the number of leagues,
- the number of teams in a league (for example, the Portuguese league changed in 2006),
- the ordering of the games by league and date,
- number of years covered (in tests I might mix tables from several years), etc.

You may assume that the leagues and the teams in each league will be numbered consecutively, starting from 1.

Upload: the improved file `cw1.m` as an attachment to the Victory assignment `CW1(b)`.

Part (c)

A first function: multiply large numbers **7 marks**

Matlab can only handle integer numbers up to 2^{53} ($\sim 10^{16}$) accurately. But we can use row vectors such as `x=[1,9,8]` or `y=[2,0,4,9]` that consist only of single digit numbers from `0` to `9` to represent arbitrarily large non-negative integers (in this case `x=198`, `y=2049`). Write a function that takes two row vectors of single digits and multiplies them, treating the row vectors as integers. Your function should behave like this on the command-line:

```
>> MyMult([1,9,8],[2 0 4 9])
ans =
     4     0     5     7     0     2
>> MyMult([1:9,9:-1:0],[7*ones(1,12)])
ans =
  Columns 1 through 13
     9     6     0     2     1     9     4     7     7     6     8     0     7
  Columns 14 through 26
     9     5     6     1     0     5     2     2     3     1     8     2     4
  Columns 27 through 30
     4     1     7     0
```

Step-by-step instructions:

1. Create a new `m` file and save it as `MyMult.m`. This file will contain the function `MyMult`.
2. The first line of the function file `MyMult.m` has to look like this:

   ```
   function res=MyMult(x,y)
   ```

   Add your code and test your function on some examples (see Hints).
3. Upload the file `MyMult.m` and, if necessary, other files that contain functions which you call in `MyMult` (see Hints below!). Assignment is `CW1(c)`.

Hints and further instructions

- This is **individual** coursework. If you copy code from other students you will be turned in for plagiarising.
- You do not have to worry about negative numbers. You can assume that all input vectors consist only of single digits and that the first digit is greater than zero, unless the input vector is equal to `0`. This means that you do not need to check the input arguments for sanity.
- The function should return a vector that contains only non-negative single digit numbers. So `[4,0,57,0,2]` instead of `[4,0,5,7,0,2]` is not valid.
- Remove leading zeros from the result: `[0,4,0,5,7,0,2]` is not valid.
- **MyAdd** In Victory in folder `MyAdd` you will find functions `MyAdd` (named, for example, `MyAdd023.m`) written by last year's students. They *add* row vectors of digits. You may download one of these function files, modify it if you need to, and call it inside your function. Beware that, if you use one of these functions, you have to upload it to Victory, too. It is your responsibility to test if your adopted `MyAdd` works. Errors caused by `MyAdd` count as your errors. These files are not guaranteed to work (many of them don't), and they are provided only for your convenience or inspiration.
- If your main function file is not called `MyMult.m` then it will fail the automatic test, awarding you a provisional mark of 0.