[60 marks]

Name:

Introduction to University Mathematics 2017

MATLAB Worksheet III

Complete the following tasks and hand in before the end of the lab session.

1. All about arrays. Let's use MATLAB to play with vectors and matrices (all considered *arrays* in MATLAB). In your command window, create the following row vector v and matrix M (see Sheet 2 if you've forgotten how).

$$v = \begin{pmatrix} 1 & 0 & 3 & -1 \end{pmatrix} \qquad M = \begin{pmatrix} 1 & 0 & 3 & -1 \\ 0 & 2 & 10 & 5 \\ -4 & 1 & 1 & 5 \end{pmatrix}$$

Experiment with the commands below, *e.g.* by typing `size(v)` and `size(M)`, and describe precisely what each command does to vectors and matrices in general. The first row has been done for you. Feel free to experiment with your own arrays.

Command	effect on a vector	effect on a matrix
`size`	gives the dimension of the vector in the form [#rows, #columns].	gives the dimension of the matrix in the form [#rows, #columns].
`length`	gives the length of the vector	gives max(# of rows, # of columns)
`numel`	same as 'length'	gives total # of elements
`max`	gives the maximum element in the vector	gives an array of the maximum element in each column
`sum`	gives the sum of all elements	gives an array of the sum in each column
`prod`	gives the product of all elements	gives an array of the product in each column

[10]

i) Devise a command which picks out the *overall maximum* of M (*i.e.* 10 in this case).

Ans: max(max(M)) or max(M(:))

[2]

ii) Explain precisely what these commands do

`v(end)`	shows the last element of $\bar{v}$.
`P = rot90(M)`	Replace M by a 90°-rotated version of itself
`Q = repmat(M,2)`	produces the matrix 
`[r,c] = find(M==10)`	r = row # c = column # } in which '10' appears in M.

[4]

iii) Write down a line of command which will quickly generate a matrix A which consists of 20 rows of the array $\begin{pmatrix} 1 & 3 & 5 & 7 & 9 \end{pmatrix}$.

A = repmat ([1:2:9], 20, 1)

[2]

2. Magic squares. Generate a 3×3 matrix called `mat`, with the command

```
mat=magic(3)
```

The matrix represents 3×3 *magic square*, in which the sum of elements in any row, column or diagonal is the same. Let's call this sum the *magic total*. For example, the magic total for `magic(3)` is 15.

Use the commands in the previous questions to answer the following questions.

i) What commands can be used to verify that the rows and columns of `mat` all add up to the magic total?

sum(mat)
 sum(mat') } all entries are the same (15)

[2]

ii) The *trace* of a matrix is defined as the sum of the entries along its main diagonal (*i.e.* top left to bottom right).

Write down two lines of commands which can be used to verify that the 2 diagonals of `mat` add up to the magic total?

trace(mat)
 trace(rot90 (mat))

[2]

iii) Calculate the magic total for a 51×51 magic square.

Command: trace(magic(51)) Numerical answer: 66351 [2]

iv) In which row/column does the entry 999 appear in the 100×100 magic square? Give the command used (avoid any low-tech method).

```
[r, c] = find(magic(100) == 999)
```

Ans: row 91, column 2 [2]

3. Element-wise operations. We will often need to perform operations on entire arrays of numbers, element by element.

For example, let's start with the matrix `mat=[4:6 ; 3:-1:1]` (see last week's sheet if this command doesn't make sense).

a) To multiply every element of `mat` by 2, we type `2*mat`. Easy enough. Similar for division. Describe what the following commands do. [1]

`mat+10` add 10 to every entry of mat

`sqrt(mat)` square root of every entry of mat [2]

b) Using ideas from last week, or otherwise, write down two ways in which you can quickly create a 15×20 matrix consisting entirely of numbers 9.

```
zeros(15, 20) + 9
9 * ones(15, 20)
```

[4]

c) Now let's suppose we want to square every element in the matrix, so that the desired result is

$$\begin{pmatrix} 16 & 25 & 36 \\ 9 & 4 & 1 \end{pmatrix}$$

The command `mat^2` will produce an error because, as you may know, a 2-by-3 matrix cannot be multiplied to another 2-by-3 matrix.

Instead, to perform an *element-wise* operation, simply place *a dot* in front of the operation. In this case, the command

```
mat.^2
```

produces the desired result (try it).

i) What command can be used to produce a matrix consisting of the reciprocals of the elements of `mat`?

```
mat.^(-1)
```

or `1./mat` [1]

ii) In fact, we can use the `.*` operation to multiply (or `./` to divide) two matrices element-wise. For instance, you can check that

```
mat.*mat
```

gives exactly the same result as `mat.^2`. Now fill in the blanks below.

$$\texttt{mat.*}\begin{pmatrix} 0 & -1 & -2 \\ 1 & 0.5 & 10 \end{pmatrix} = \begin{pmatrix} 0 & -5 & -12 \\ 3 & 1 & 10 \end{pmatrix} \qquad [1]$$

$$\texttt{mat.*}\begin{pmatrix} 1.75 & 1.6 & 1.5 \\ 3 & 4 & 7 \end{pmatrix} = \begin{pmatrix} 7 & 8 & 9 \\ 9 & 8 & 7 \end{pmatrix} \qquad [2]$$

Warning: Always use `/` and `./` for division. There is also the backslash `\`, but you will only need it in when solving linear systems. Don't confuse the two slashes.

4. Series calculation. Here is an example of how you can use the element-wise operations in MATLAB to compute series. Let's supposed we want to evaluate the series

$$1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \ldots + \frac{1}{100^2} \qquad (\star)$$

You could do the following. Write down the command needed in each step.

- Create an array `A=[1 2 3 ... 100]`
  (warning: don't type this line in literally).
  As the output is going to create a huge mess on your screen, *suppress the output* with a *a semicolon* ; at the end of the command. Even though nothing seems to happen, you can check the *Workspace* window that `A` has been created.

  Command: `A=[1:100];` [1]

- Create an array `B` containing the elements of `A` squared; `B=[1 4 9 ... 10000]` (suppress output).

  Command: `B=A.^2;` [1]

- Create an array `C` containing the reciprocals; `C=[1 1/4 1/9 ... 1/10000]` (suppress output).

  Command: `C=1./B;` [1]

- Sum all elements in `C`. Obviously don't suppress the output this time.

  Command and answer: `sum(C) = 1.6350` [2]

Write down a *single* line that combines all the previous commands, giving you the final answer for the series $(\star)$ right away without using predefined variables.

Command: `sum(1./[1:100].^2)` [2]

Use the array techniques you learnt above (and playing around with arrays), answer the following questions.

(a) Write down a single line of command that will evaluate the following series.

$$1+\frac{1}{3}+\frac{1}{5}+\frac{1}{7}+\ldots+\frac{1}{101}$$

Command: `sum(1./[1:2:101])` [2]

Numerical answer = 2.9477 [1]

(b) Write down the series that is being evaluated in this line of command. Use the summation sign $\sum$.

`sum(exp([1:20])./factorial([1:20]))` $= \sum_{n=1}^{20} \frac{e^n}{n!}$ [1]

(c) Let's try to evaluate the series

$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\ldots-\frac{1}{1000}.$$

Try following these steps (consult last week's sheet if necessary).

- First create an array `A=[1 1 1 1 ...]` (1000 entries).
- Replace the even entries of `A` by -1.
- Divide `A` by the array `[1 2 3 ...1000]` (element-wise).
- Sum the result.

Write down the commands used below (suppress output where appropriate):

```
A = ones(1,1000);
A(2:2:end) = -1;
sum(A./[1:1000])
```

[3]

Numerical answer = 0.6926 [1]

(d) Write down another set of commands to calculate the series in part (c), but this time, instead of using the steps above, use this idea:

series = (positive terms) + (negative terms)

```
pos = 1./[1:2:999];
neg = 1./[2:2:1000];
sum(pos - neg)
```

[3]

5. Error and accuracy. Consider the series ($\star$) in Question 4 (page 4) again. In second year, you will be able to show (using *Fourier series*) that when we include sufficiently many terms, this series will approach a surprising number. In symbols, we have:

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6},$$

where equality holds only if *infinitely* many terms are summed up (similar to the "*sum to infinity*" for geometric series).

Of course, computers cannot add infinitely many terms together. However, using a sufficiently large number of terms (say, 100 or more), the result should be pretty close to $\pi^2/6$.

Suppose we want to know how accurate is the 100-term series in estimating the above infinite sum. One way to quantify the error is to use the formula

$$\text{Percentage error} = \left(\frac{\text{Estimate} - \text{Actual}}{\text{Actual}}\right) \times 100\%$$

where "Estimate" is the 100-term series, and "Actual" is the theoretical value of the infinite series.

The percentage error could be negative – it just means that your estimate is less than the actual answer (an "underestimate").

What is the percentage error when we use 100 terms to estimate the infinite series? Write down all the commands used.

```
est = sum( 1./[1:100].^2) ;
act = pi^2/6 ;
err = 100 * (est - act)/act
```

[3]

Ans: Percentage error = -0.6049 % (to 4 significant figures) [1]

Does the percentage error become smaller or larger *in magnitude* when more terms are included?

Ans: smaller [1]

NEXT WEEK: writing your first proper MATLAB code (M-file).