Introduction to Matlab

Elementary arithmetic

Elementary arithmetic operations in order of precedence from lowest to highest: To change the order

+ , −	addition, subtraction,
∗ , /	multiplication, division,
ˆ	power

Table 1: Elementary Matlab operations

of precedence in an expression use parentheses ().
Example

```
>> 2+3

ans =

     5

>> 3.17 - 2.77

ans =

     0.4000

>> 3/2^3

ans =

     0.3750

>> (3/2)^3

ans =

     3.3750
```

To suppress output use semicolon ';' at the end of formula/command
Example

```
>> 37^21;
>>
```

Functions

Non-user defined functions in Matlab start with lower-case letter and are of the form

$$\text{functionname}(\textit{variable1}, \textit{variable2}, \ldots)$$

Some well known functions are given in following table:

sqrt(x)	=	square root of x
exp(x)	=	e^n
log(x)	=	natural logarithm of x
sin(x)	=	sine of x in radians
cos(x)	=	cosine of x in radians
tan(x)	=	tangent of x in radians

Table 2: Some elementary Matlab functions

Here x can be number, expression possibly containing another function, matrix (see further below) etc. You can find more information about any internal function and its usage by placing caret on name of a function and pressing F1 key.
Example

```
>> sqrt(2)

ans =

    1.4142

>> sin(3.1415/2)

ans =

    1.0000

>> sqrt(sin(3.1415/4))

ans =

    0.8409
```

Variables

Variable names must begin with a letter, which can be followed by any number of letters, digits and underscores. Operator '=' is used to assign a value to a variable. To display defined variable (i.e. variable with assigned value) write name of the variable. You can use defined variables in computations with functions and arithmetic operators.
Example

```
>> sroot2 = sqrt(2)

sroot2 =

    1.4142

>> sroot2

sroot2 =

    1.4142

>> expression = sroot2 + 3 * sroot2
```

```
expression =

                 5.6569

>> sin(expression)

ans =

                 -0.5862
```

Matrices

Matlab is essentially working with matrices. Even numbers are considered as 1 by 1 matrices. To define a matrix use brackets '[ ]'. You define a matrix line by line; '[' starts definition, elements in lines are separated by comma ',' or space ' ', lines are separated by semicolon ';' or new line (Enter). ']' finishes definition of matrix.

Example

```
>> A = [1  2  3;  4  5  6]

ans =

                 1     2     3
                 4     5     6

>> B = [9 ,8 ,7
 6 ,5 ,4]

ans =

                 9     8     7
                 6     5     4
```

There are more elementary operations on matrices which are given in next table with obvious constrains on dimensions of matrices:

$+$, $-$	standard matrix addition, subtraction,
$*$	standard matrix multiplication,
$.*$	element-wise multiplication,
$'$	transposition with complex conjugation
$.'$	transposition without complex conjugation
ˆ	power of matrix
.ˆ	element-wise power
./	element-wise left division; A./B has elements A(i,j)/B(i,j)
$\backslash$	finds solution to matrix equation AX=B (Gauss or least squares)

Table 3: Matrix operations

Exercise

1. Define a few matrices and try out the aforementioned operations.
2. Use matrix as input in functions from Table 2 and explain obtained result. In fact, many functions work in similar way for matrices.

There are other ways to define matrices with certain pattern of its elements. Function 'zeros(n)' creates n by n zero matrix, 'zeros(m,n)' creates zero matrix with m lines and n columns. Function 'ones(n)' defines square matrix with all its elements equal to 1 and it has nonsquare version as well. 'rand(m,n)' generates (pseudo)random m by n matrix.
Operator ':' is used in the form 'a:b' or 'a:'step':b and generates one row matrix of numbers in sequence from a to b with difference 1 or 'step'.
Example

```
>> 2:8

ans =

            2    3    4    5    6    7    8

>> 1.2:0.5:3

ans =
            1.2    1.7    2.2    2.7
```

Exercise

1. Define one-column matrix with squares of integers from 1 to 10 as its elements.

Matrix elements access

For defined matrix A, the command 'A(n)', for positive integer 'n', returns n-th element of A; here A is considered as one column matrix with all columns of A stacked one under another.
To get an element in the i-th row and j-th column, one uses A(i,j).
Other matrices can be used for indexing as well, e.g. A(1:3:end,:) returns matrix which consists of first, fourth, seventh ... row of matrix A.
Exercise

1. Define 10 by 10 random matrix A and generate matrix B consisting of all elements of A with even coordinates.

Logicals

We can compare two matrices of the same dimension, using standard relations >,<,>=,<=,~=,==. Comparison of matrices returns a matrix with elements 0 and 1, depending on true or false state on corresponding coordinates.
Example

```
>> A=[1  2  3;2  3  4];
>> B=[2  2  2;2  2  2];
>> A > B

ans =

            0    0    1
            0    1    1

>> A ~= B

ans =
     1   0   1
     0   1   1
```

Comparison returns so called logical matrix, which can be used for indexing
Example

```
>> A=[1  2  3;2  3  4];
>> B=[2  2  2;2  2  2];

>> In = A ~= B

In =

     1   0   1
     0   1   1

>> A(In) = -1

ans =
     -1     2    -1
      2    -1    -1
```

If you create matrix A which contains only 0 and 1 as elements, you cannot use such a matrix for indexing, because the matrix is not of logical type, however you can change its type using command 'logical(A)'

```
>> A=[1  2  3;2  3  4];
>> B=[1  0  1;0  1  0];

>> A(B)=-1

??? subscript indices must either be real positive
        integers or logicals

>> A(logical(B))=-1

ans =
     -1     2    -1
      2    -1     4
```

Problems

1. Solve system of linear equation

$$\begin{aligned} 2x - 3y + 5z - w &= 17 \\ -4x + y - 11z + 3w &= -19 \\ -17x - 5y \qquad + 3w &= 2 \\ 5x - 3y + 12z + 12w &= 9 \end{aligned}$$

   If A is matrix of the system, B is right side and X is solution of the system, compute B–A∗X.

2. Solve overdetermined system

$$\begin{aligned} 2x - 3y + 5z &= 17 \\ -4x + y - 11z &= -19 \\ -17x - 5y \qquad &= 2 \\ 5x - 3y + 12z &= 9 \end{aligned}$$

Another useful function is 'size(A)', which returns a matrix containing dimensions of matrix A, i.e. number of rows and number of columns respectively. There can be defined multidimensional matrices with more than two dimensions, typical example is matrix representation of RGB image.

Images

The command 'imread('image_file.bmp/jpg/...')' returns RGB matrix representation of an image. This matrix has three dimensions (use 'size' command), first two are dimensions of the image and second is 3 – one for each colour. Elements of this matrix represent intensity of given colour. There are two possible representation of this intensities, either by real number between and including 0 and 1, or by integers from 0 to 255. But only one of these is used for a particular image matrix. Standard two dimensional matrix represents greyscale image.

To show matrix A (two dimensional greyscale or 3 dimensional RGB) as image use command 'imshow(A)'.

Example

```
>> A = 0:1/255:1;
>> B = repmat(A,100,1);
>> imshow(B)
```

here the function 'repmat' creates matrix with 100 rows and 1 column but each element of this matrix is matrix A. That means at the end we get matrix with 100 rows and 256 columns, because A has 1 row and 256 columns.

```
>> A = 0:255;
>> B = repmat(A,100,1);
>> imshow(uint8(B))
```

Function 'uint8' changes type of matrix B which was user defined, i.e. has type double and in that case (double) function 'imshow' requires real values 0 to 1 for intensities. For (unsigned) integer values (uint8) the intensities are represented by integers from 0 to 255.

Exercise

1. Load a colour image and extract three colour channels into three different matrices. Show this channels in their proper colours (i.e. red in red colour, etc.).

2. Load a colour image and construct grayscale representation of the image. Hint: You have to combine red, green and blue channels of image matrix into one number. There is obvious combination (arithmetic mean) but this does not represent intensities very well (we don't perceive intensities of red, green and blue evenly). Experiment with different weights of colour channels.

There is function 'rgb2gray(RGBmatrix)' which returns grayscale matrix representation from RGB matrix representation of an image.

Example

```
>> RGBimg = imread('image.bmp');
>> imshow(RGBimg);
>> GsImg = rgb2gray(RGBimg);
>> imshow(GsImg)
```

Problems

1. Load image and downsample it by the factor of 2, e.g. take submatrix containing element on places with even coordinates. Repeat this process for obtained matrix several times (4 or 5), show resulting images and observe changes.

To save a matrix use command 'imwrite(n,'*name.ext*'), where n is the image matrix and *ext* is type of image file, e.g. bmp, jpg, png.

Functions

We write function into file with extension m, name of the file is the same as name of the function.

The file starts with function header definition in the form

```
function ReturnValue = FunctionName(InputValue1, InputValue2)
```

or

```
function [ReturnValue1,ReturnValue2,...] =
                         FunctionName(InputValue1, InputValue2)
```

if you want to return more then one value (matrix). Body of the function then contains a sequence of commands. To return a value, assign it to the name given to it in the header. E.g. following function will be defined in file 'AritMean2.m'.

```
function Result = AritMean2(A, B)
        Result = (A + B)/2;
```

Problems

1. Write and test a function, which gets a grey-scale image matrix as input and returns an intensity histogram of this image, i.e. a vector which has number of pixels of intensity i on $i+1$-th position (positions are counted from 1, intensities from 0). Function 'sum' may be useful. You can plot the histogram in the form of a bar graph using command 'bar'. Or you can learn about functions 'reshape' and 'hist' as well and use them.

2. Write and test a function, which gets RGB image matrix and returns three histograms, one for each colour channel. You can use previous function inside this one.