Cédric Dumoulin (cedumoul@ulb.ac.be)
Arnaud Deraemaeker (aderaema@ulb.ac.be)

Introduction to Matlab

1 Matlab Basics

MATLAB is a 'Matrix Oriented' programming software (MATLAB stands for MATrix LABoratory). Operations and commands in MATLAB are intended to work with matrices as they would be written on a paper. It is a command interpreter software which has both a user interface capacity (the 'Command Window') and programming capacities (basically the 'scripts' and the 'functions').

This document is only a small introduction to MATLAB and only presents some of the main concepts of this powerful program that are needed in the framework of the exercises sessions. As for any other programs, it is necessary to practice **as often as possible** in order to become familiar. For new users particularly, the *help menu* is essential in that it contains many examples that can be tested. Because MATLAB is widely used in many fields (aerospace, finances, bioinformatic, statistics, signal processing,. . .) many other examples and tutorials can be easily found on the internet.

1.1 Matrices

The $[A]$ matrix is expressed with square brackets [and] and the element of the matrix are written as follows

- Each entry of a same column is separated by white spaces or commas
- Each row is separated by a semi-colon (';') or a new line.

so that the 3 by 3 matrix $[A]$ defined as

$$[A] = \begin{bmatrix} 1 & 8 & 10 \\ 9 & 4 & 6 \\ 2 & 5 & 7 \end{bmatrix}$$

can be typed in the matlab command window as follows

```
>> A = [1,8,10;9,4,6;2,5,7];
>> A = [1 8 10
     9 4 6
     2 5 7 ];
```

where the two expressions lead to the same matrix and the semi-colon at the end of the command is used to avoid the matrix to be displayed. To display the matrix one has to remove the semi-colon from the command or type

```
>> A

A =

     1     8    10
     9     4     6
     2     5     7
```

One can directly access the value (i, j) of a matrix by typing ($i = 2,\ j = 3$)

```
>> b=A(2,3)

b =

     6
```

or a row of the matrix, typing

```
>> R1=A(2,:)

R1 =

     9     4     6
```

a column of the matrix, typing

```
>> C1=A(:,2)

C1 =

     8
     4
     5
```

1.2 Vectors

Vector are simply matrices with a single row or column (as C1 and R1). One can simply write a sequence row vector typing

```
>> x=0:0.1:1;
```

or

```
>> x=linspace(0,1,11);
```

which both generate an equally spaced row vector of length 11 from 0 to 1.

1.3 Matrix Operations

In terms of matrix operations one should distinguish between the array operations and the matrix operations. The array operations work on a element by element basis. They are typically written with the same symbols as the usual operation but with an extra 'dot'.

Matrix Command

Operation	Matlab Command	Example	help
Sum of Matrix	`+`	`C=A+B`	
Difference of Matrix	`-`	`C=A-B`	
Product of Matrix	`*`	`C=A*B`	mtimes
Matrix power	`^`	`C=A^b`	mpower
Solve the system $Ax = b$	`\`	`x=A\b`	mldivide
Solve the system $xA = b$	`/`	`x=b/A`	mrdivide
Transpose	`.'`	`tA=A.'`	transpose
Get the Eigenvalues and Eigenvectors matrices Λ, Ψ of the generalized eigenvalue problem $A\Psi = B\Psi\Lambda$)	`eig()`	`[Psi,L]=eig(A,B)`	eig
Complex conjugate transpose	`'`	`tA=A'`	ctranspose
Inverse of Matrix A	`inv()`	`IA=inv(A)`	inv
Determinant of Matrix A	`det()`	`dA=det(A)`	
Rank of Matrix A	`rank()`	`rA=rank(A)`	
Get Diagonal of A	`diag()`	`b=diag(A)`	
Make Diagonal matrix A from vector b	`diag()`	`A=diag(b)`	
Make Unitary NxN Diag. matrix	`eye()`	`A=eye(N)`	
Make Unitary NxM matrix	`ones()`	`A=ones(N,M)`	
Make Zeros NxM matrix	`zeros()`	`A=zeros(N,M)`	

Array Command (Element By Element)

Array Product $A_{ij} \cdot B_{ij}$	`.*`	`C=A.*B`	times
Array left divide A_{ij}/B_{ij}	`.\`	`C=A.\B`	ldivide
Array right divide A_{ij}/B_{ij}	`./`	`C=B./A`	rdivide

Table 1 – Basic Matrix and Array Operations

Exercice 1

Consider the following matrices

$$[K] = \begin{bmatrix} 32 & -16 & 0 & 0 \\ -16 & 32 & -16 & 0 \\ 0 & -16 & 20 & -4 \\ 0 & 0 & -4 & 4 \end{bmatrix}, \quad [B] = \begin{bmatrix} 1 & -2 & 5 \\ 6 & 1 & -1 \end{bmatrix}, \quad [C] = \begin{bmatrix} 10 & -5 \\ 3 & 1 \end{bmatrix},$$

$$[M] = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

1. Compute the matrix products KB, BK and KB^T
2. Compute the array products $K_{ij}B_{ij}$, $K_{ij}M_{ij}$
3. Compute $D = I - BB^T$
4. Compute the determinants of $K,\ B,\ C,\ D,\ M$
5. Compute the inverse of $K,\ B,\ C,\ D,\ M$
6. Considering the column vector b, compute the solution of the system $Kx = b$

   $$\{b\} = \begin{Bmatrix} 2 \\ 4 \\ -1 \\ 0 \end{Bmatrix}$$

7. Considering the matrix $b2$

   $$[b2] = \begin{bmatrix} 2 & 5 \\ 4 & 3 \\ -1 & -6 \\ 0 & 7 \end{bmatrix}$$

   compute the following command

   ```
   >> x2 = K\b2
   ```

8. Determine the eigenvalue matrix L and the eigenvectors matrix V such that

   $$\mathbf{KV} = \mathbf{MVL}$$

   Note that this system is fully equivalent at findind the eigenvalues λ_i and the corresponding eigenvectors V_i of the eigenvalue problem

   $$(\mathbf{K} - \lambda_i \mathbf{M})\, V_i = 0$$

   The matrices $\mathbf{L}$ and $\mathbf{V}$ are simply given by

   $$\mathbf{L} = \begin{bmatrix} \lambda_1 & 0 & \cdots \\ 0 & \lambda_1 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \qquad \mathbf{L} = \begin{bmatrix} V_1 & V_2 & \cdots \end{bmatrix}$$

Answer

```
clear all;close all;clc

K = [     32    -16      0      0
         -16     32    -16      0
           0    -16     20     -4
           0      0     -4      4 ];

B=[ 1 -2 5 4; 6 1 -1 2]
C=[10 -5 ; 3 1]
M=diag([5 4 6 2])


%%% 1)
K*B      %does not work
B*K      %ok
K*B'     %ok

%%% 2)
K.*B     %does not work
K.*M     %works (same size)


%%% 3)
D = eye(size(B*B.',1))-B*B.'

%%% 4)
DK = det(K)     %works
DB = det(B)     %does not work (Matrix must be square.)
DD = det(D)     %works
DC = det(C)     %works
DM = det(M)     %works

%%% 5)
IK = inv(K)     %works
IB = inv(B)     %does not work (Matrix must be square.)
ID = inv(D)     %works
IC = inv(C)     %works
IM = inv(M)     %works

%%% 6)
b = [2 4 -1 0].';
X=K\b

%%% 7)
b2 = [2 4 -1 0;5 3 -6 7].';
X2 = K\b2       %equivalent at solving X2 = [K\b2(:,1) K\b2(:,2)] but ...
    more efficient!!

%%% 8)
[V,L]=eig(K,M)
%Generalized eigenvalue problem
RES=K*V-M*V*L;
disp(RES)
RES=round(RES*1e10)/1e10; %round at 10^-10
disp(RES)
```

```
%Eigenvalue problem
l = diag(L);
for i1=1:size(V,2);
    res(:,i1)=(K-l(i1)*M)*V(:,i1);
end
disp(res)
res=round(res*1e10)/1e10; %round at 10^-10
disp(res)
```

2 Programming in Matlab

Scripts and Functions are text files (M-Files .m), which can be called immediately from the command line without the need of compilation. They are either called from the MATLAB command window (prompt ») or inside a script of a function by typing the name of the M-File:

- A Script file is a text file which automates a list of instructions.
- A function file is a text file which automates a list of instructions depending of the arguments and which allows to return variables. A function `my_function.m` should always start as follows

  ```
  function [out1,out2,out3] = my_function(in1,in2,in3)

  list of instructions
  ```

  One simply calls the function from another function, script or the command window by typing `[o1,o2,o3] = my_function(i1,i2,i3)`. Writing a function is therefore particularly appropriate when a same task has to be carried out multiple times with different input variables.
- MATLAB (and the related toolboxes) provides hundreds of functions (trigonometric, statistical, signal processing, . . .).

2.1 Loops

MATLAB allows iteration over a sequence of instructions using a `for` loop.

```
for i1=1:N
list of instructions
end
```

At each iteration the variable `i1` takes the value of the current iteration so that all the following syntax contains valid instructions

```
v = 1:N
for i1=v
list of instructions
end
%%%
for i1=N:-2:1
list of instructions
end
```

Loops are very useful but it has to be underlined that `for` is by far less efficient than matrix operation. Many functions (such as trigonometric functions) accept matrices as argument, wherever possible, it is always preferable to use the matrix form instead of using a loop. It is also a good practice to pre-allocate the variables before the loop even it is not mandatory.

```
t=0:0.001:10;
%Matrix form
tic
```

```
y=sin(t);
toc
%Elapsed time is 0.001490 seconds.

%loop form
tic
for i1=1:size(t,2)
    y1(i1)=sin(t(i1));
end
toc
%Elapsed time is 0.030021 seconds.

%loop form with pre-allocation
tic
y2=zeros(size(t));
for i1=1:size(t,2)
    y2(i1)=sin(t(i1));
end
toc
%Elapsed time is 0.011879 seconds.
```

2.2 Logical tests

Logical tests allow to choose between different groups of commands depending on the logical test results.

if, elseif, else

In MATLAB, logical tests can be performed with commands `if`, `else` and `elseif`. As for the loops, the logical tests are ended with `end`. Here is the general syntax

```
if expression 1
   list of instructions 1
elseif expression 2
   list of instructions 2
else
   list of instructions 3
end
```

and an example of the syntax in a for loop

```
t=0:0.001:10;
%loop form
tic
for i1=1:size(t,2)

    if t(i1)<=2
       y1(i1)=sin(2*t(i1));
    elseif t(i1) > 3 && t(i1)<= 5
       y1(i1)=sin(5*t(i1));
    else
       y1(i1)=sin(t(i1));
    end

end
toc
%Elapsed time is 0.018511 seconds.
```

Switch-Case

Another way to choose between several groups of commands is the switch-case command:

```
switch switch_expression
   case case_expression 1
      list of instructions 1
   case case_expression 2
      list of instructions 2
   otherwise
      list of instructions 3
end
```

```
result = 52;

switch(result)
   case 52
      disp('result is 52')
   case {52, 78}
      disp('result is 52 or 78')
   otherwise
      disp('result is out of range')

end
```

Logical Indexing

One can also want to find the indices of a vector or a matrix which correspond to given logical conditions.

A logical instruction returns boolean variables. For instance, the following command will return a vector of boolean variables (1 if the condition is satisfied, 0 otherwise)

```
t=0:0.001:10;
it = (t>=1)
```

and one could use this result to get the part of the vector which satisfies the condition:

```
t2=t(it)
t2=t(t>=1)
```

An alternative is to use the function `find(instruction)` which returns the indices which satisfy the condition:

```
it=find(t>=1);
t2=t(it)
```

The previous example related to the if, elseif, else structure can therefore be expressed as (by far more efficient)

```
t=0:0.001:10;
tic
y=sin(t);
y(t>3 && t<=5)=sin(2*t(t>3 && t<=5));
y(t<=2)=sin(2*t(t<=2));
```

```
toc
%Elapsed time is 0.003891 seconds.

%%% OR
tic
y=sin(t);
i1=find(t>3 & t<=5);y(i1)=sin(2*t(i1));
i1=find(t<=2);y(i1)=sin(2*t(i1));
toc
%Elapsed time is 0.003425 seconds.
```

Exercice 2

Code a function which has two matrices of the same dimension as inputs. That function has to give the sum and the difference of these two matrices. The function will also return the product term by term of the two input matrices. To do so, you should consider two different approaches:

- Using the adequate operator;
- Using a loop.

In order to compare the results obtained with the two approaches, the function will eventually return a Boolean parameter which is true if the two approaches give the same result and false if not.

Answer

In the command window or a script:

```
[S,D,P,chk]=Ex2(K,M)
```

In the function `Ex2.m` file (saved in the current working directory):

```
function [sumAB,diffAB,prodAB1,checkRes]=Ex2(A,B)

sumAB=A+B;            %sum of matrices
diffAB=A-B;           %difference of matrices
prodAB1=A.*B;         %Array (term by term multiplication

%term by term multiplication using for loop
[a1,a2]=size(A);         %size of the matrix
prodAB2=zeros(a1,a2);    %initiate matrix
for m=1:a1
    for n=1:a2
        prodAB2(m,n)=A(m,n)*B(m,n);   %compute term by term
    end
end
checkRes=0;                 %check of equal
if prodAB1==prodAB2
    checkRes=1;
end
```

3 Plot

The main function to plot results is the `plot` function.

- Points can be shown with different markers : o,*,+,x,. . .
- Different colors are available: r(ed), b(lue), blac(k), w(hite), y(ellow), m(agenta), g(reen). But one can define them as RGB colors. `gr=[125 125 125]/250;` of by calling the included colormap functions: jet, hsv, summer, ... For instance, one can obtain N gray colors by calling the built-in color map `gray` as follows: `COL=gray(N)`.

The following script calls that function to plot a sine.

```
t=0:0.1:10;
y=sin(t);
figure;     %Open a new figure
plot(t,y);  %Plot with defaults parameters
hold on;    %Keep the plots on the figure
y2=sin(2*t);
plot(t,y2,'color','m');     %Plot in magenta
y3=sin(t/2);
plot(t,y3,'color','black','linestyle','--','marker','x');  %Plot in ...
    black with 'x' marker

figure;     %Open a new figure
gray=[125 125 125]/250;
plot(t,y3*2,'color',gray,'linewidth',3);  %Plot in gray without marker, ...
    thicker line
```

One can add titles to the axis with the command, labels and legend as follows

```
figure;     %Open a new figure
COL=jet(10);
plot(t,y3*2,'color',COL(1,:),'linewidth',3);    %Plot in gray without ...
    marker, thicker line
hold on; box off; grid on;
plot(t,y2,'color',COL(5,:),'linewidth',3);  %Plot in gray without ...
    marker, thicker line
title('A sine function')
xlabel('time t (s)')
ylabel('sine(t)')
```

Exercice 3

Write a function which displays (plot) a polynomial function of degree n, whose coefficients are inputs of the function. Another input to be considered is the vector x for which the function should be computed and displayed. The function should also return the indices of the interval s of x between which the function is close to zero. The function will therefore have an additional input ε which defines the accuracy for you calculation (the interval s will correspond to a value of the polynome between $0 \pm \varepsilon$).

Answer

In the command window or a script:

```
A=[0 -1 2 -1];
x=linspace(-10,10,1000);

[y1,I1]=Ex3(A,x,10);
figure;hold on;
plot(x,y1);
plot(x(I1),y1(I1),'r','linewidth',2,'linestyle','none','marker','x');grid ...
    on

x=linspace(-4,4,1000);
A=[-80 -26 154 28 -26 -2 1]/100
[y2,I2]=Ex3(A,x,0.2);
figure;hold on;
plot(x,y2);
plot(x(I2),y2(I2),'r','linewidth',2,'linestyle','none','marker','x');grid ...
    on
```

In the function `Ex2.m` file (saved in the current working directory):

```
function [y,I]=Ex3(A,x,eps1)

%A are the coefficients of the polynome
%A(1) > n=0
%A(2) > n=1
%A(k) > n=k-1;

y=zeros(size(x));        %Initiate result vector
for k=1:length(A)
    y=y+A(k)*x.^(k-1);
end

I=find(abs(y)<=eps1);     %find the abs values that are lower than eps
```