academic year 2019

Introduction to Finance with MATLAB

University Paris-Dauphine
Magistère Banque, Finance & Assurance

Gauthier Vermandel[1] *gauthier.vermandel@dauphine.fr*

Lecture 1:
An Introduction to MATLAB Programming

Content of the Lecture

Lecture's goals

- Utilize the basic mathematical operations;
- Get know the general purpose commands of Matlab;
- Manipulate matrix calculus;

[1]Department of Economics, Paris-Dauphine University. Codes available on my website: : http://vermandel.fr/bfa1.

1. MATLAB in a nutshell

MATLAB (matrix laboratory) is a multi-paradigm numerical computing environment and fourth-generation programming language. A proprietary programming language developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, C#, Java, Fortran and Python.

MATLAB (the name stands for: *Matrix Laboratory*) is a high performance programming language and a computing environment that uses vectors and matrices as one of its basic data types (MATLAB®is a registered trademark of the MathWorks, Inc.). It is a powerful tool for mathematical and technical calculations and it can also be used for creating various types of plots.[2]

1.1 Basic calculus

It is the main window in which the user communicates with the software. In the command window, the user can view the prompt symbol ">>" which indicates that MATLAB is ready to accept various commands by the user. Via this window, the user can employ the basic arithmetic operators like: "+" (addition), "-" (subtraction), "*" (multiplication), "/" (division), "^" (powers) and the "( )" (brackets), as well as many other build in elementary and other functions and commands that will be referred to later.

As a first example, enter the following command that performs a basic calculation:

```
>> 5+6/12+9/3-2
ans =
6.5000
```

The MATLAB displays the results into a variable named as ans. This is a default variable name that is used by the command window to display the most recent results instructed by the user that has not defined a specific name. Continue by entering the following command that creates a four element vector:

```
>> [1 2 3 4]
ans =
1     2     3     4
```

The square brackets "[ ]" indicate the definition of the vector. Also, the space between the vector numbers separates the vector's elements. The comma "," is another way of separating the elements. Additionally, note that the default variable ans has lost its previous value in order to store the results of the most recent operation.

As another example, enter the following command that creates a 3-by-3 magic square saved in the matrix variable M:

```
>> M=magic(3)
M =
```

[2]This section is buildt on the work of Panayiotis Andreou entitled "Introductory Course to Matlab with Financial Case Studies".

```
8     1     6
3     5     7
4     9     2
```

The magic square is created using the element function magic() that is already built in Matlab.

Type also the following:

```
>> x=[(2^2+1)^2-15/4*6.1, 1.23e-2]
x =
2.1250    0.0123
```

Observe that the very first command that you have entered in the command window has vanished with the additional of the last one (of course it depends on the size of the window, in here the command window is minimized so earlier commands vanish too quickly). You can use the scroll bar on the right to navigate the command window and go up to see the earlier commands (or view them via the command history window). MATLAB calculates the quantities as follows:

- First come the quantities in brackets;
- Power calculation follow: (*e.g.* 5 + 2^2 = 5 +4 = 9);
- "*" and "/" follow by working from left to right: (*e.g.* 2*8/4 = 16/4 = 4);
- "+" and "-" follow last, from left to right: (*e.g.* 6-7+2= -1+2 = 1).

Note that "e" notation is used for very large or very small numbers. By definition: 1e1=1X101 and 1e-1=1X10-1.

Trigonometric		*Exponential*	
sin	sine	exp	exponential
sinh	hyperbolic sine	log	natural logarithm
asin	inverse sine	log10	common (base10) logarithm
asinh	inverse hyperbolic sine	log2	base 2 power and scale floating point
cos	cosine	pow2	base 2 power and scale floating point
cosh	hyperbolic cosine	realpow	power that will error out on complex
acos	inverse cosine	reallog	natural logarithm of real number
acosh	inverse hyperbolic cosine	realsqrt	square root of number greater than $x^{0.5}$
tan	tangent	sqrt	square root
tanh	hyperbolic tangent	nextpow2	next higher power of 2
atan	inverse tangent	*Complex*	
atan2	four quadrant inverse	abs	absolute value
atanh	inverse hyperbolic tangent	angle	phase angle
sec	secant	complex	construct complex data from real & imaginary
sech	hyperbolic secant	conj	complex conjugate
asec	inverse secant	imag	complex imaginary part
asech	inverse hyperbolic secant	real	complex real part
csc	cosecant	unwrap	unwrap phase angle
csch	hyperbolic cosecant	isreal	true for real array
acsc	inverse cosecant	cplxpair	sort numbers into complex conjugate pairs
acsch	inverse hyperbolic cosecant	*Rounding and Remainder*	
cot	cotangent	fix	round towards zero
coth	hyperbolic cotangent	floor	round towards minus infinity
acot	inverse cotangent	ceil	round towards plus infinity
acoth	inverse hyperbolic cotangent	round	round toward nearest integer
		mod	modulus
		rem	remained after division
		sign	signum

Table 1: MATLAB's elementary build-in functions

Matlab can handle three different kinds of numbers: integers, real numbers and complex numbers (with imaginary parts). Moreover, it can handle non-number expressions like: NaN (Not-a-Number) produced from mathematically undefined operations like: $0/0$, $\infty^* \infty$ and inf produced by operations like $1/0$. Matlab as a calculator includes a variety of build-in mathematical functions like: trigonometric, exponential, complex, rounding and remainder, etc. Table 1 below, depicts these elementary mathematical build-in functions.

Note the build-in functions exhibited in Table 1 have their own calling syntax. For example, if you type sin in the command window, MATLAB returns the following error massage:

```
>> sin
Error using sin
Not enough input arguments.
```

This happened because the sin function requires an input expression like a number enclosed in brackets. If you enter:

```
>> sin(5)
ans =
-0.9589
```

In the rest of this handout, only the name of the build in functions will be given.

Beside some exceptions where the correct calling syntax of a function is fully tabulated, the user is responsible in knowing the necessary input arguments to the function. Additional the use of the command window will be apparent as you read this manuscript since the learning of a computer programming language is pure a "learning by doing" process.

Exercise 1

1. Create a file exercice1.m and edit with MATLAB to store your exercise.
2. Perform these simple calculus in MATLAB:

$$8+9$$
$$8\times 9$$
$$\frac{8}{5}$$
$$8^9$$

3. Create two variables $u = 8$ and $v = 9$ and reproduce the previous calculus using u and v.
4. Clear the workspace using clear; function.
5. Use buildin MATLAB function to solve:

$$\sin\left(\frac{\pi}{2}\right)$$
$$\sqrt{16}$$
$$\log(10e^{-5})$$

6. Let's consider $u = 1 + \sqrt{-4}$. Separate the imaginary and the real part of u.
7. Show the documentation for function sin(·) using doc sin in the console.

2. Vector manipulations

A matrix or an array is the basic element on which Matlab can operate. A 1-by-1 matrix forms a scalar or a single number, whereas a matrix with only one row or column forms a row or column vector respectively. This section exhibits the mathematical manipulation of vectors (arrays) and of two dimensional matrices. Most of build-in function for matrix calculation are reported in Table 4.

2.1 Row of vectors

A vector is a list of numbers separated by either space or commas. Each different number/entry located in the vector is termed as either element or component. The number of the vector elements/components determines the length of the vector. In

zeros	Create array of all zeros	isscalar	Determine whether input is scalar
ones	Create array of all ones	isvector	Determine whether input is vector
rand	Uniformly distributed random numbers	ismatrix	Determine whether input is matrix
true	Logical 1 (true)	isrow	Determine whether input is row vector
false	Logical 0 (false)	iscolumn	Determine whether input is column vector
eye	Identity matrix	isempty	Determine whether array is empty
diag	Create diagonal matrix	sort	Sort array elements
blkdiag	Construct block diagonal matrix	sortrows	Sort rows of array, table, or timetable
cat	Concatenate arrays	issorted	Determine whether array is sorted
horzcat	Concatenate arrays horizontally	flip	Flip order of elements
vertcat	Concatenate arrays vertically	fliplr	Flip array left to right
repelem	Repeat copies of array elements	flipud	Flip array up to down
repmat	Repeat copies of array	rot90	Rotate array 90 degrees
linspace	Generate linearly spaced vector	transpose	Transpose vector or matrix
logspace	Generate logarithmically spaced vector	ctranspose	Complex conjugate transpose
freqspace	Frequency spacing for frequency response	permute	Rearrange dimensions of N-D array
meshgrid	2-D and 3-D grids	ipermute	Inverse permute dimensions of N-D array
ndgrid	Rectangular grid in N-D space	circshift	Shift array circularly
length	Length of largest array dimension	shiftdim	Shift dimensions
size	Array size	reshape	Reshape array
ndims	Number of array dimensions	squeeze	Remove singleton dimensions
numel	Number of array elements	colon	Create vectors, array subscripting, and for-loop

Table 2: MATLAB's Matrices and Arrays build-in functions

Matlab, square brackets "[ ]" are used to both define a vector and a matrix. For instance, the following command returns a row vector with 5 elements:

```
>> y=[ 5 exp(2) sign(-5) sqrt(9) pi]
y =
5.0000    7.3891   -1.0000    3.0000    3.1416
```

Note that the definition of the row vector, the user it free to use any built-in function as long as this is used properly. In the above definition, exp(.) is the exponential, sign(.) returns the sign, sqrt(.) is the square root and pi represents π. General speaking and except some special cases, when a function is applied to a 1-by-1 scalar, the result is a scalar, when applied to a row or column vector is a row or column vector and when applied to a matrix the output is again a matrix. This happens because MATLAB applied the build-in functions element-wise. The length of the above vector is obtained via the following command:

```
>> length(y)
ans =
5
```

Note that length(.) is a build-in function that given a vector, it returns its length. Since the result of this function is not stored in a user-defined variable, MATLAB saves the result in the ans variable. If it is need to save the vector's length to a variable named Ylength the command would be:

```
>> Ylength=length(y)
Ylength =
5
```

With MATLAB, vectors can be easily multiplied by a scalar and added or subtracted with other vectors of similar length. Moreover, a scalar can be added or subtracted to or from a vector and smaller vectors can be used to construct larger ones. All these operations are performed element-wise. Note that each vector represents a variable.

An element-by-element multiplication of vector a by 5:

```
>> a=[-1 2 -3]; b=[2 1 2];
c=5*a
c =
-5    10   -15
```

An element-by-element addition of vector b with 5:

```
>> c1=5+b
c1 =
7     6     7
```

An element-by-element addition of two equal length vectors:

```
>> d=a+b
d =
1     3    -1
```

A larger vector is created after certain manipulations:

```
>> d=a+b
d =
1     3    -1
```

To refer to specific elements of the vector, the vector's name is followed by the element's rank in brackets. For instance we can change the values of the ee vector with the following command:

```
>> e(2)=-99; e(4)=-99; e(6)=-99;
e
e =
0   -99     0   -99     0   -99
```

2.2 The colon notation

The colon notation ":" can be used to pick out selected rows, columns and elements of vectors, matrices, and arrays. It is a shortcut that is used to create row vectors. Moreover, the colon notation is used to view or extract parts of vectors (afterwards, with matrices, the colon can be used to view a certain part of a matrix). For instance, the following commands produce three different row vectors:

```
>> v1=1:8, v2=-4:2:2, v3=[0.1:0.2:0.6]
v1 =
1     2     3     4     5     6     7     8
v2 =
-4    -2     0     2
```

```
v3 =
0.1000    0.3000    0.5000
```

The careful viewer should have notice that the vector creation via the use of the colon notation has the form: starting_value:step:finishing_value. The starting_value consists the first value/element of the vector, the finishing_value is the last value/element of the vector and all other elements differ by a value equal to step. Also, when the interval between finishing_value and starting_value is not divisible by the step, the last value of the vector is not the finishing_value (*i.e.* in v3 the last element is 0.5 and not 0.6). The colon notation is also used to view or extract parts of a vector.

Extracting the 2th, 3rd and 4th elements of v1:

```
>> v4=v1(2:4)
v4 =
2     3     4
```

Extracting the 2th, 5th and 8th elements of v1:

```
>> v1(2:3:8)
ans =
2     5     8
```

Exercise 2

1. Create a file exercice2.m and edit with MATLAB to store your exercise (and so on for each exercise).
2. Create the vector $z = \begin{bmatrix} 1 & 5 & 7 & 9 \end{bmatrix}$.
3. Determine its length using the function length(·).
4. Create a vector w of size 4 composed of random values using function rand(·).
5. Check whether w is a row vector using function isrow(·).
6. Show the 3^{rd} value of w.
7. Multiply the 3^{rd} value of w by 2.

2.3 The column vectors

Column vectors are created via the use of semi-colon ";" instead of commas and spaces. Operations with column vectors are similar as with row vectors. The following examples depict the manipulation of column vectors.

Creating a four-element column vector:

```
>> cv=[1;4;7;9]
cv =
1
4
```

```
7
9
```

Taking the vector's length:

```
>> length(cv)
ans =
4
```

Creating CV that is a four-element column vector. Its first two elements are the two first elements of cv increased by 2 whereas the last two elements are the 2nd and 3rd elements of cv multiplied by 5:

```
>> CV=[cv(1:2)+2; cv(2:3)*5]
CV =
3
6
20
35
```

3. Matrices calculus

3.1 Matrices scalar product

Matlab can perform, scalar products (or inner products) and dot products, as well as dot division and power operations. The only restriction is that the length of the vectors must be the same. The priorities concerning vector manipulations is the same as in the case that we use MATLAB as a calculator (power operations first followed by "*" and "/" followed by "+" and "-").

The scalar product or otherwise termed as inner product, concerns the multiplication of two equal length vectors. The symbol "*" is used to carry out this operation. Given a row vector W and a column vector U of length N:

$$W = \left[\ w_1,\quad w_2,\quad ...,\quad w_N\ \right], \qquad U = \begin{bmatrix} u_1 \\ u_2 \\ ... \\ u_N \end{bmatrix}$$

The inner product is defined as the linear combination:

$$W \times U = \sum_n^N w_n u_n$$

Producing the inner product of various operations. For example prod1 is computed as: prod1=(1*2)+(0*4)+(2*(-2))+((-1)*0.5)=-2.5.

```
>> w=[1 0 2 -1]; u=[2; 4; -2; 0.5];
prod1=w*u, prod2=(2+w)*(u/2), prod3=w*w', prod4=w*u*w*u
prod1 =
-2.5000
```

```
prod2 =
3.2500
prod3 =
6
prod4 =
6.2500
```

Exercise 3

1. Create the vector $z = \begin{bmatrix} 2 & 2 & 2 & 2 \end{bmatrix}$ using function ones(·).
2. Determine its length using the function length(·) and store the output in variable l.
3. Sum the vector in variable s1.
4. Perform a sum with a vector product.
   (a) To do so, declare a vector full of ones with same size $x = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$ using function ones(·) and variable l.
   (b) Compute the sum via $z \times x'$.
5. Create the following matrices: $U = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$ through three ways:
   (a) by hand.
   (b) using function linspace(·).
   (c) using MATLAB's synthax a:b:c where a is the starting value, c is the ending value and b is the step size.

3.2 Matrices dot product

The dot product involves the multiplication of two similar vectors (to be either column or row vectors with same length) element-by-element. With the dot product, a new vector is created. The dot product between the row vectors W and U^T (where the superscript T represents the transpose symbol) is another row vector with the following form:

$$W \times U^T = \begin{bmatrix} w_1u_1, & w_2u_2, & ..., & w_Nu_N \end{bmatrix}$$

The dot product in Matlab is performed via the ".*" symbol. By the use of dot product we can get the inner product. This is explained in the following examples.

Producing the dot product of various operations. For example dprod1 is computed as: dprod1=[1*2, 0*4, 2*(-2), (-1)*0.5].

```
>> dprod1=w.*u', dprod2=u'.*u'.* w
dprod1 =
2.0000         0   -4.0000   -0.5000
dprod2 =
4.0000         0    8.0000   -0.2500
```

Exercise 4

1. Build the following matrix 2x2:

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

2. Build exactly the same matrix, but using the reshape function. First create a vector $\begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$ and use the reshape function (tips: you have to transpose the matrix).

3. Using a matrix 2x2 full of all two, and use a dot product to multiply by 2 the matrix a.

3.3 Matrices dot power

The dot power of vectors works in a similar way as with other dot products. Usually there is the need to square (or to rise to some other power) the elements of a vector. This can be done with the dot power operation as explained in the following example set.

```
>> x=-5:5;
d_power=sqrt(x.^2)
d_power =
5     4     3     2     1     0     1     2     3     4     5
```

3.4 Matrix operators

MATLAB comparison operators are:

==	equal	>	strictly higher than
~=	different	<=	less or equal to
<	strictly less than	>=	greater than or equal to

Table 3: MATLAB's Matrices operators

The comparison of two scalars returns 1 if the relationship is true and 0 if false. It is also possible to make comparison between matrices of same size or between a matrix and a scalar. The following examples illustrates the flow controls overs matrices:

```
>> A = [1 2;3 4]; B = 2*ones(2);
>> A == B
ans =
0     1
0     0
>> A > 2
ans =
0     0
1     1
```

To assess how two matrices are identical, it is possible to use the IsEqual:

```
>> isequal(A,B)
ans =
0
```

The IsEqual(·) function is one of the many boolean functions offered by MATLAB, characterized by their names starting with is statement. The table below summarizes some of them:

ischar(x)	True if x is a character	isnan(x)	True if the elements of x are not a number
isempty(x)	True if x is empty	islogical(x)	True if element is logical
isequal(x,y)	True if the elements of x and y are identical	isnumeric(x)	True if the elements of x are numeric
isfinite(x)	True if the elements of x are finite	isreal(x)	True if the elements of x are real
isinf(x)	True if the elements of x are infinite		

Table 4: MATLAB's functions checking some propreties of arrays/matrices

The find(·) command can be used to return the indices corresponding to a non-zero element of the vector. for example:

```
>> x = [-3 1 0 -inf 0];
f = find(x)
f =
1     2     4
```

Find the results can then be used to extract the elements of the vector:

```
>> x(f)
ans =
-3     1  -Inf
```

You can also use the find command to get the finite elements of the vector x above:

```
>> x(find(isfinite(x)))
ans =
-3     1     0     0
```

If one wishes to replace the negative elements of x with a zero, we do this:

```
>> x(find(x < 0)) = 0
x =
0     1     0     0     0
```

It is possible to assign a value for multiples elements.

Exercise 5

Let us assume that we have to solve the following linear system:

$$2x - 3y = 5$$
$$x + 5y = 1$$

MATLAB is very efficient at making matrix calculations such as matrix inver-

sions. We can rewrite the problem as:

$$B \times \begin{bmatrix} x \\ y \end{bmatrix} = A$$

The solution of the system is given by the matrix division $B \backslash A$ which is equivalent to $B^{-1}A$.

1. Find the numerical solution of the problem.
2. Check whether your results are good.

Let us now work on a production function. The production process of goods Y involves labor demand L, physical capital K and technology A. Capital renting cost is R and real wage is W. The technology is given by:

$$Y = AK^{\alpha}H^{1-\alpha}$$

The optimal demand for each inputs are:

$$R = \alpha A \left(\frac{H}{K}\right)^{1-\alpha}$$
$$W = (1-\alpha) A \left(\frac{K}{H}\right)^{\alpha}$$

1. Assuming that $\alpha = 0.4$, $Y = 10$, $R = 1.04$ and $W = 5$, determine the amount of inputs A, K, L satisfying the production plans of the firm. (tips: this non-linear problem can be linearized which allows to get an analytical solution for that problem).
2. The central bank decides to increase the interest rate by one percent $R = 1.05$. Discuss the implications for the demand for physical capital.

Exercise 6

Let us first work on a truncating a probability distribution:

1. Generate 10,000 random numbers drawn from a normal distribution (0,1). Draw the histogram of the normal distribution using function hist(·).
2. Suppose that we want to truncate the distribution last decile (*i.e.* removing 10% highest values). Determine the critical threshold. Replace the last decile of the distribution by NaN and plot the truncated distribution.

Let us now find the maximum of a function characterized by an unique global maximum. There a function $f(x)$ defined by:

$$f(x) = 50x - x^2$$

1. Define a vector of x between -50 and 50 (step size of one), and compute the value of $y = f(x)$.
2. Compute the global extremum of $f(x)$ (called ymax).
3. Find the value of x maximizing $f(x)$ (called xmax).

Exercise 7

Generate a matrix 5×5 full of gaussian random values with zero mean and a normalized standard deviation.

1. Determine the number of elements in the matrix above 0.
2. Express this number as a share of the total elements in the matrix. Store your result in variable myshare.
3. Contrast the share you obtained with the theoretical one you should obtain? Increase the number of draws in your matrix to observe the statistical convergence.
4. Each time the file is run, the following message should popup in the console window: "*The number of shares above zero is X*" where X denotes myshare, i.e. the share of elements above zero in the matrix you have previously computed (this number should be different each iteration of the file). To do so, follow these instructions carefully:
   (a) As a first step to illustrate concatenation, store in variable str1 "Hello " and in variable str2 "world!". To concanate two strings, use brackets [str1 str2].
   (b) Now concanate "*The number of shares above zero is* " with the number stored in myshare.
   (c) Employ function disp(.) to show a message in the console.
   (d) myshare is interpreted as a number by variable instead of a string, which creates an issue. Use function num2str(·) to convert my myshare into a string to correctly plot the share.
5. Find the same output using.fprint(·) function.

Assignment 1

Start a new file to do your assignment, copy paste at the beginning of the file the following command:

```
%% EXERCICE 1
% loading macro data
load Data_NelsonPlosser
% select data from 1910-1970
ourData = Data(51:end,:);
% time period vector 1910-1970
Time        = dates(51:end);
%% Extract data
% Stock price
SP = ourData(:,14);
% Bond Yield
BY = ourData(:,13);
% real per capita GNP
GNPPC = ourData(:,3);
```

This command loads macroeconomic data for the US economy between 1910 to 1970. It extracts US stock prices, the bond yield rate and the real per capita GNP.

1. Compute the stock price return using the variable SP with $r_t = \log(SP_t/SP_{t-1})$. Same for the real GDP growth rate g_t. *Tips:* look for function diff(.).

2. Assuming that we want to know how much asset price return r_t is affected by the GDP growth g_t. To do so, the simple regression equation is given by: $Y_i = b_0 + b_1 X_{i1} + e_i$ with $e_i \backsim \mathcal{N}(0, \sigma_e^2)$. The equation above can be expressed more compactly by a set of matrices: $Y = Xb + e$.

   (a) Create a matrix of all ones with functions ones() with the same size as g_t named X0.

   (b) Create the matrix $X = [\mathrm{X0}, g_t]$ composed of two vectors: one of all ones, and another with g_t.

   (c) Regression estimates of $b = [b_0, b_1]$ are typically found via least squares. The minimization problem of least squares boils down to:

   $$b = \left(X^T X\right)^{-1} X^T Y$$

   Compute b and interpret the result.

   (d) Confirm your result using an internal function of MATLAB performing linear regressions (use 2 methods).

   (e) Compute the residual vector e_t.

3. Now include the treasury bond yield (variable BY in MATLAB) in the regression model and estimate b. Are your results consistent with empirical evidence? *Tips:* the first value of BY must be removed to have the same number of observations as for g_t.