Chapter 2: Data types and variables

Chapter 2: Data types and variables

2.1– Introduction

Like any programming language, MATLAB allows you to define variable data. A variable is designated by an identifier which is made up of a combination of letters and numbers. Please note, the first character of the identifier must necessarily be a letter. For MATLAB any variable is considered to be an array of elements of a given type.

In this chapter, we will see in particular how to create a variable, give it a value, and use it.

2.2– Types of variables

The three main types of variables used by Matlab are: real, complex and strings. The logical type is associated with the result of certain functions [5].

Variable names must follow a few rules:

• The length of the name is arbitrary;

• The name may contain letters in lowercase or uppercase, without accent;

• The name may contain numbers;

• The name cannot contain special characters (space or punctuation);

• The name must start with a letter.

Example:

```
>> a=1.2
a =
1.2000
```

We can now include this variable in new mathematical expressions, to define a new one:

```
>> b = 5*a^2+a
b =
8.4000
```

and then use these two variables:

```
>> c = a^2 + b^3/2
```

c =
297.7920

I now have three variables a, b and c. These variables are not permanently displayed on the screen. But to see the content of a variable, nothing could be simpler, we tap its name:

>>b
b =
8.4000

or double-click on its name in the workspace.

2.2.1– Complex

The imaginary unit is designated by **i** or **j**. Complex numbers can be written in Cartesian form:

a+ib (a+i*b (or a+b*i))

Example:

>> Z=a+ b*i

Z=

1.2000 + 8.4000i

The symbol * can be omitted if the imaginary part is a numerical constant. All previous operators work in complex. For example:

>> Z = (a+b*i)^2

Z =

-69.1200 +20.1600i

Here are some commands regarding complex numbers:

imag(Z): returns the imaginary part of Z

real(Z): returns the real part of Z

abs(Z): returns the modulus of Z

angle (Z): returns the angle of Z

conj(Z): returns the conjugate of Z (Z *)

2.2.2– Character string

A string is an array of characters. Character string (char) data is represented in the form of a series of characters surrounded by single apostrophes (‘).

Example:

>> ch1='hello'

ch1 =

Good morning

2.2.3– Logic

A logical type result is returned by certain functions or in the case of certain tests. The logical type has two forms: true or false (1 or 0).

2.3- Clearing variables

The **clear** command clears part or all of the variables defined so far.

Syntax:

clear a b c . . .

If no variables are specified, all variables will be cleared.

2.4- Predefined variables

There are a number of pre-existing variables. We have already seen "**ans**" which contains the last calculation result, as well as i and j which represent.

There is also **pi**, which represents π, and a few others. Remember that "**eps**", the name we often tend to use, is a predefined variable. ***WARNING:*** These variables are not protected, so if you assign them, they do not keep their initial value. This is often the problem for **i** and **j** which we often use spontaneously as loop indices, so that we can no longer define a complex.

2.5- Predefined functions

All common functions and many of the less common ones exist. Most of them operate in complexes. Please note that to apply a function to a value, you must put the latter in parentheses.

Example:

>> sin(pi/12)

ans =

0.16589613269342

Here is a non-exhaustive list:

- trigonometric and inverse functions: sin, cos, tan, asin, acos, atan
- hyperbolic functions (we add "h"): sinh, cosh, tanh, asinh, acosh, atanh
- root, logarithms and exponentials: sqrt, log, log10, exp

- error functions: erf, erfc

- Bessel and Hankel functions: besselj, bessely, besseli, besselk, besselh.

2.6- Matrices and tables

Despite the title of this section, MATLAB does not differentiate between the two. The concept of table is important because it is the basis of the graph: typically for a curve of n points, we will define a table of n abscissa and a table of n ordinates. But we can also define rectangular arrays with two indices to define matrices in the mathematical sense of the term, and then perform operations on these matrices.

2.6.1- Definition of a table

We use the square brackets [and] to define the start and end of the matrix. So to define a variable M containing the matrix $\begin{bmatrix} 1 & 2 & 3 \\ 11 & 12 & 13 \\ 21 & 22 & 23 \end{bmatrix}$, we will write:

```
>> M = [1,2,3;11,12,13;21,22,23]
M =
1 2 3
11 12 13
21 22 23
```

We use the symbol '**,**' which serves as a column separator and '**;**' line separator. We can also define vectors, row or column, using this syntax (a space is interpreted as a comma).

Example:

```
>> U = [1 2 3]
U =
1 2 3
```

defines a line vector, whereas (an "enter" command is interpreted as a semicolon):

```
>> V = [11
12
13]
V =
11
12
13
```

Defines a column vector. The transition from one command line to the next is carried out by hitting the <Enter> key.

We could also have defined the latter by:

```
>> V=[11;12;13]
```

2.6.2- Accessing an array element

Simply enter the name of the table followed in parentheses by the index(s) whose value you want to read or write.

Example:

If I want the value of M, line 3, column 2:

```
>> M(3,2)
ans =
22
```

To modify only one element of an array, we use the same principle. For example, I want M32 to equal 32 instead of 22:

```
>> M(3,2)=32
M =
1 2 3
11 12 13
21 32 23
```

You will notice that MATLAB suddenly redisplays the entire matrix, taking the modification into account.

We may wonder what happens if we affect the component of a matrix that does not yet exist.

Example:

```
>> P(2,3) = 3
P =
0 0 0
0 0 3
```

Here's the answer: MATLAB automatically constructs a table large enough to reach the specified indices, and puts zeros everywhere except at the term in question. You will notice that unlike traditional languages, there is no need to size the tables in advance: they are built gradually.

2.6.3- Extracting sub-arrays

It is often useful to extract blocks from an existing table. To do this, we use the '**:**' character. The start value and the end value must be specified for each index. The general syntax is therefore as follows (for an array with two indices):

array(start: end, start: end)

So to extract the block $\begin{bmatrix} 2 & 3 \\ 12 & 13 \end{bmatrix}$, we will tap:

```
>> M(1:2,2:3)
ans =
2 3
12 13
```

We use the "**:**" character alone to take all possible indices.

Example:

```
>> M(1:2,:)
ans =
 1 2 3
11 12 13
```

This is very useful for extracting rows or columns from a matrix. For example to obtain the second line of M:

```
>> M(2,:)
ans =
11 12 13
```

2.6.4- Construction of tables by blocks

You know this principle in mathematics. For example, from the previously defined matrices and vectors, we want to define the matrix $\begin{bmatrix} M & V \\ U & 0 \end{bmatrix}$ which is a 4x4 matrix. To do this in MATLAB, we act as if the blocks were scalars, and we simply write:

```
>> N=[M V
U 0]
N =
 1 2 3 11
 11 12 13 12
 21 32 23 13
 1 2 3 0
```

or by using the characters '**,**' and '**;**'

```
>> N=[M, V; U, 0]
```

This syntax is widely used to extend vectors or matrices, for example, if I want to add a column to M, constituted by V:

```
>> M=[M V]
M =
1 2 3 11
1 12 13 12
21 32 23 13
```

If I want to add a row to it, consisting of U:

```
>> M = [M;U]
M =
1 2 3
11 12 13
21 32 23
1 2 3
```

2.6.5- Table operations

2.6.5.1- Addition and subtraction

The two operators are the same as for scalars. As long as the two tables concerned have the same size, the resulting table is obtained by adding or subtracting the terms of each table.

2.6.5.2- Multiplication, division and power term by term

These operators are noted "**.***", "**./**" and "**.^**" (be careful not to forget the period). They are designed to carry out forward operations on two tables of the same size. These symbols are important when you want to draw curves.

2.6.5.3- Multiplication, division and power in the matrix sense

Since we want to manipulate matrices, it seems interesting to have matrix multiplication.

This is simply noted * and should not be confused with term-to-term multiplication. It goes without saying that if we write ***A*B*** the number of columns of ***A*** must be equal to the number of rows of ***B*** for the multiplication to work.

The division makes sense with respect to the inverses of matrices. Thus ***A/B*** represents ***A*** multiplied (in the sense of matrices) by the inverse matrix of ***B***. There also exists a division on the left which is noted **\.** Thus ***A\B*** means the inverse of ***A*** multiplied by ***B***. This symbol can also

be used to solve linear systems: if v is a vector, $A\backslash v$ mathematically represents $A^{-1}v$, that is to say the solution of the linear system $Ax = v$.

The n^{th} power of a matrix represents this matrix multiplied n times in the sense of matrices by itself.

To clearly show the difference between the "**.***" and "***" operators, a small example involving the identity matrix multiplied by the matrix $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$.

Here is the multiplication in the sense of matrices:

```
>> [1 0; 0 1] * [1 2; 3 4]

ans =
1 2
3 4
```

and now the multiplication term by term:

```
>> [1 0; 0 1].* [1 2; 3 4]

ans =
1 0
0 4
```

Example:

A. Matrix and extraction

Write a script that defines the following matrix M and displays the last row:

```
 1 2 3 4 5
 9 10 11 12 13
-1 -2 -3 -4 -5
```

B. Random element matrix

Determine the syntax of the rand function, create 2 matrices ***A*** and ***B*** of size 5X6, with random values between -10 and 10. Display ***C= A+B***

C. Operations on a matrix

A=[1.4;6.8]; Is it possible to apply the sin function to the matrix a? What is the result? Write a script that displays the successive results of sin(a).^n with n varying from 1 to 10 and using a for...end loop.

2.6.5.4- Transposition

The transpose operator is the ' (apostrophe) character and is often used to transform row vectors into column vectors and vice versa.

2.6.6- Table lengths

The size function applied to a matrix returns an array of two integers: the first is the number of rows, the second the number of columns. The command also works on vectors and returns 1 for the number of rows (resp. columns) of a row (resp. column) vector.

For vectors, the length command is more convenient and returns the number of components of the vector, whether row or column.

2.6.7- Fast table generation

2.6.7.1- Classic matrices

We can define matrices of a given size containing only **0s** with the zeros function, or containing only **1s** with the ones function. You must specify the number of rows and the number of columns. Here are two examples:

```
>> ones(2,3)
ans =
1 1 1
1 1 1
>> zeros(1,3)
ans =
0 0 0
```

The identity is obtained with **eye**. We only specify the dimension of the matrix (which is square...)

```
>> eye(3)
ans =
1 0 0
0 1 0
0 0 1
```

There is also a **diag** function for creating diagonal matrices (see the built-in help).

2.6.7.2- Lists of values

This notion is crucial for the construction of curves. This involves generating in a vector a list of values equidistant between two extreme values. The general syntax is:

variable = start value: step: end value

This syntax always creates a row vector. For example to create a vector x of values equidistant from 0.1 between 0 and 1:

```
>> x = 0:0.1:1
x =
Columns 1 through 7
0 0.1000 0.2000 0.3000 0.4000 0.5000 0.6000
Columns 8 through 11
0.7000 0.8000 0.9000 1.0000
```

It is recommended to put a semicolon at the end of this type of instruction to avoid tedious display of the result.

Another example to create 101 equally distributed values on the interval [0; 2π]:

```
>> x = 0: 2*pi/100: 2*pi;
```

You can also use the linspace function:

```
>> x=linspace(0,2*pi,101);
```

Values can be distributed logarithmically with the logspace function.

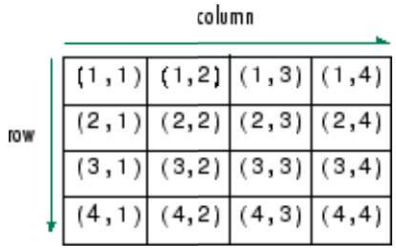
y = linspace(x_1,x_2,n) generates n points. The spacing between the points is (x_2-x_1)/(n-1).

Example: Create a vector of 7 evenly spaced points in the interval [-5,5].

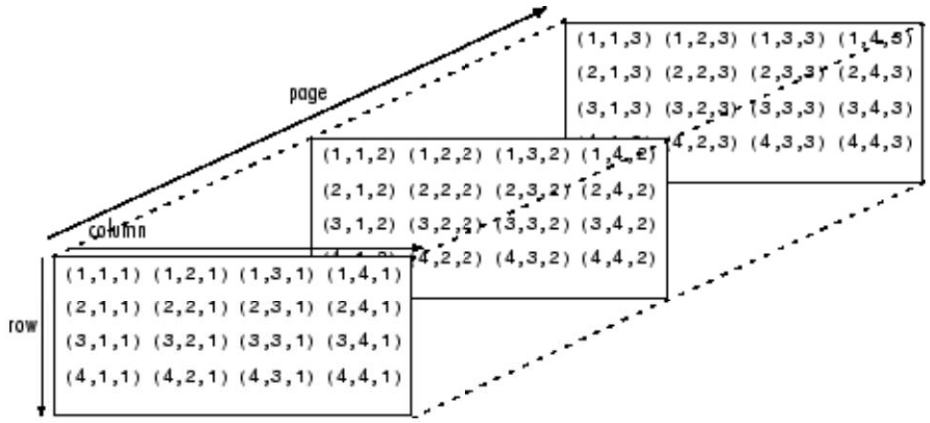
```
y1 = linspace(-5,5,7)
y1 = 1×7
  -5.0000   -3.3333   -1.6667         0    1.6667    3.3333    5.0000
```

2.6.8- Multidimensional Arrays

A multidimensional array in MATLAB is an array with more than two dimensions. In a matrix, the two dimensions are represented by rows and columns.



Each element is defined by two subscripts, the row index and the column index. Multidimensional arrays are an extension of 2-D matrices and use additional subscripts for indexing. A 3-D array, for example, uses three subscripts. The first two are just like a matrix, but the third dimension represents *pages* or *sheets* of elements.



2.6.8.1- Creating Multidimensional Arrays

You can create a multidimensional array by creating a 2-D matrix first, and then extending it. For example, first define a 3-by-3 matrix as the first page in a 3-D array [6].

A = [1 2 3; 4 5 6; 7 8 9]
A = *3×3*

1 2 3
4 5 6
7 8 9

Now add a second page. To do this, assign another 3-by-3 matrix to the index value 2 in the third dimension. The syntax `A(:,:,2)` uses a colon in the first and second dimensions to include all rows and all columns from the right-hand side of the assignment.

A(:,:,2) = [10 11 12; 13 14 15; 16 17 18]
A =
A(:,:,1) =

1 2 3
4 5 6
7 8 9

A(:,:,2) =

10 11 12
13 14 15
16 17 18

The **cat** function can be a useful tool for building multidimensional arrays. For example, create a new 3-D array **B** by concatenating **A** with a third page. The first argument indicates which dimension to concatenate along.

```
B = cat(3,A,[3 2 1; 0 9 8; 5 3 7])
B =
B(:,:,1) =

    1    2    3
    4    5    6
    7    8    9
B(:,:,2) =

   10   11   12
   13   14   15
   16   17   18


B(:,:,3) =

    3    2    1
    0    9    8
    5    3    7
```

Another way to quickly expand a multidimensional array is by assigning a single element to an entire page. For example, add a fourth page to **B** that contains all zeros.

```
B(:,:,4) = 0
B =
B(:,:,1) =

    1    2    3
    4    5    6
    7    8    9


B(:,:,2) =

   10   11   12
   13   14   15
   16   17   18


B(:,:,3) =
```

```
3   2   1
0   9   8
5   3   7
```

B(:,:,4) =

```
0   0   0
0   0   0
0   0   0
```

Examples:

A. Solving a system of equations

- Consider the system of 5 equations with 5 unknowns:

x+2y+3z+4u+5v=1
x+2y+3z+4u+v=2
x+2y+3z+u+v=3
-x-2y+3z+4u+v=4
x+y+z+u+v=5

- Write the script to resolve this system

B. Fast matrix generation

Let A=**ones**(2,3). Is it possible to calculate A-1, A+A, 2*A, A*A, A.*A? Analyze the results.

C. Fast vector generation

In a command line, create the column vector having the following values (without writing them explicitly):
[10-60;10-59; ...; 1059; 1060]
hint: logspace

D. manipulation of matrices

-Create the following matrix using the **reshape** command:

```
 1 11 21 31 41 51 61 71 81 91
 2 12 22 32 42 52 62 72 82 92
 3 13 23 33 43 53 63 73 83 93
 4 14 24 34 44 54 64 74 84 94
 5 15 25 35 45 55 65 75 85 95
 6 16 26 36 46 56 66 76 86 96
 7 17 27 37 47 57 67 77 87 97
```

8 18 28 38 48 58 68 78 88 98
9 19 29 39 49 59 69 79 89 99
10 20 30 40 50 60 70 80 90 100

-Use **triu** to create:
1 11 21 31 41 51 61 71 81 91
0 12 22 32 42 52 62 72 82 92
0 0 23 33 43 53 63 73 83 93
0 0 0 34 44 54 64 74 84 94
0 0 0 0 45 55 65 75 85 95
0 0 0 0 0 56 66 76 86 96
0 0 0 0 0 0 67 77 87 97
0 0 0 0 0 0 0 78 88 98
0 0 0 0 0 0 0 0 89 99
0 0 0 0 0 0 0 0 0 100

2.7- Polynomials

A polynomial is a function of the form:

$f(x) = a_n x_n + a_n - 1 x_{n-1} + \ldots + a_2 x_2 + a_1 x + a_0$.

The degree of a polynomial is the highest power of x in its expression. Constant (non-zero) polynomials, linear polynomials, quadratics, cubics and quartics are polynomials of degree 0, 1, 2 , 3 and 4 respectively. The function $f(x) = 0$ is also a polynomial, but we say that its degree is 'undefined'.

2.7.1- Roots of polynomial functions

You may recall that when $(x - a)(x - b) = 0$, we know that ***a*** and ***b*** are roots of the function: $f(x) = (x - a)(x - b)$. Now we can use the converse of this, and say that if a and b are roots, then the polynomial function with these roots must be $f(x) = (x - a)(x - b)$, or a multiple of this. For example, if a quadratic has roots $\boldsymbol{x} = 3$ and $x = -2$, then the function must be $f(x) = (x - 3)(x + 2)$, or a constant multiple of this. This can be extended to polynomials of any degree. For example, if the roots of a polynomial are $x = 1$, $x = 2$, $x = 3$, $x = 4$, then the function must be $f(x) = (x - 1)(x - 2)(x - 3)(x - 4)$, or a constant multiple of this [7].

Syntax:
r = roots(p)

Example : Solve the equation $3x^2 - 2x - 4 = 0$.

*Create a vector to represent the polynomial, then find the roots.

```
p = [3 -2 -4];
r = roots(p)
r = 2×1

    1.5352
   -0.8685
```

*Create vectors `u` and `v` containing the coefficients of the polynomials x^2+1 and $2x+7$.

```
u = [1 0 1];
v = [2 7];
```

Use convolution to multiply the polynomials.

```
w = conv(u,v)
w = 1×4

     2     7     2     7
```

`w` contains the polynomial coefficients for $2x^3+7x^2+2x+7$.

Exercises

1. What is a polynomial function?

2. Which of the following functions are polynomial functions?

(a) $f(x) = 4x^2 + 2$ *(b)* $f(x) = 3x^3 - 2x + \sqrt{x}$ *(c)* $f(x) = 12 - 4x^5 + 3x^2$

(d) $f(x) = \sin(x) + 1$ *(e)* $f(x) = 3x^{12} - 2/x$ *(f)* $f(x) = 3x^{11} - 2x^{12}$

3. Write down one example of each of the following types of polynomial function:

(a) cubic (b) linear (c) quartic (d) quadratic