A very brief Matlab introduction

Siniša Krajnović January 24, 2006

This is a very brief introduction to Matlab and its purpose is only to introduce students of the CFD course into Matlab. After reading this short note and truing the presented examples one should be able to write short programs that are required in the CFD course. Matlab itself provides a comprehensive help that can be reached by typing **help** in the command window or by using the Matlab **Help** window.

Vectors and Matrices

Every variable in Matlab is a matrix. Here we show how to assign numbers, vectors and matrices to variables. The Matlab prompt is >> and the commands below are typed in after the prompt and concluded with a carriage return. The response of Matlab appears on the next line.

Ex:

```
>> a=10

a =

   10

>> v=[1;7;9]

v =

    1
    7
    9

>> B=[1,2,3;4,5,6;7,8,9]

B =

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

As we can see from these examples, the rows of a matrix are separated by semicolons, while the entries on any given row are separated by commas (Note that spaces can be used instead of commas). Each of these three variables **a, v** and **B** is regarded as a matrix. The scalar **a** is a **1 x 1** matrix, the vector **v** is a **3 x 1** matrix and **B** is a **3 x 3** matrix.

Indices of variables must be positive integers. Thus, **x(0), x(-3), x(1.2) or B(0,1)** are not allowed. The size of a matrix can be found using the function **size(B),** which returns the pair of numbers **m, n** when **B** is an **m x n** matrix.

Ex:

>> size(B)

ans =

3 3

>> size(v)

ans =

3 1

The length of the vector **v** is found by using the function **length(v).**

Ex:

>> length(v)

ans =

3

If we want to prevent Matlab from displaying the entered matrix immediately afterward, we need to end the line with a semicolon

Ex:

>> v=[1;7;9];
>>

Although Matlab says nothing when you ended the line with a semicolon, it has remembered your definition of **v.**

The square root function **sqrt** is a built-in feature

>> sqrt(27)

ans =

5.1962

The variable **ans** contains the result of the most recent computation which can then be used as an ordinary variable in subsequent computations:

>> ans*10

ans =

51.9615

In this example the result of **sqrt(27)** from previous computation that was stored in **ans** was multiplied with **10** and the result of this multiplication is stored in **ans** overwriting the old value of **ans.**

Another built-in variable is **pi:**

>> pi

ans =

3.1416

The predefined functions in Matlab (besides the square root) include:

abs	absolute value
angle	phase angle of a complex number
real,imag	real part, imaginary part of complex number
conj	complex conjugation
exp	exponential function
round	round to the nearest integer
fix	round towards zero
sign	signum function
rem	remainder
sin, cos, tan	trigonometric functions
asin, acos, atan	inverse trigonometric functions

Ex:

>> sin(1)^3*tan(1)

ans =

0.9279

>> exp(1)

ans =

2.7183

>> log(ans)

ans =

1

Matlabs' online help includes a list of built-in special functions and routines, as well as a list of other commands on which help is available. To obtain the list type **help.** For help on a particular command, type **help** followed by the command. You can also get help by using the Matlab **Help** window, which is accessed by going to the **Help** menu at the top of the screen.

Ex:

>> help sin
SIN Sine of argument in radians.
SIN(X) is the sine of the elements of X.

See also ASIN, SIND.

You can also type **help help** to find out how to find help. For demonstrations of Matlab features type **demo** in the command window.

Operations on matrices

For matrices **A** and **B,** Matlab can compute the sum, difference and the product of these two matrices. To do this, type **A+B, A-B,** and **A*B,** respectively. **Note! The order is important in matrix multiplication:**

>> A=[1,2;3,4]

A =

1 2
3 4

>> B=[1,0;1,0]

B =

```
   1     0
   1     0

>> A*B

ans =

   3     0
   7     0

>> B*A

ans =

   1     2
   1     2
```

Typing **A^2**, for a square **(n x n)** matrix **A** , yields the matrix product **A*A.**

Applying to a matrix **A** any of the built-in functions returns a matrix of the same dimensions containing the values of the functions as if it had been applied componentwise:

```
>> A=[pi,0;pi/2,3*pi/2]

A =

   3.1416         0
   1.5708    4.7124

>> cos(A)

ans =

  -1.0000    1.0000
   0.0000   -0.0000
```

When we want Matlab to multiply two matrices **A** and **B** that have the same dimension in a comprehensive fashion (rather than the matrix multiplication), we use the operator **.*** rather than ***.** Similarly, componentwise division of two matrices of the same dimension can be accomplished by writing **A./B** which creates a matrix whose entries are **A(i,j)/B(i,j).**

Ex:

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

A =

   1   2
   3   4
>> B=[2,4;6,8]

B =

   2   4
   6   8

>> A*B

ans =

  14  20
  30  44

>> A.*B

ans =

   2   8
  18  32
```

There are several commands in Matlab which helps you to easy input certain standard types of matrices. The built-in function **zeros(m,n**) returns an **m x n** matrix of zeros. **ones (m,n)** returns an **m x n** matrix of ones.

When the matrix contains only one row, we have a row vector instead of a matrix, and a Matlab provides a special ways to input certain types of row vectors. In Matlab, the notation **start:step:stop** denotes a row vector whose first entry is **start** and whose consecutive entries differ by **step**, and whose last entry does not excide **stop**.

Ex:

```
>> a=2:2:8

a =

   2   4   6   8
```

The parameters **start, step** and **stop** do not have to be integers:

Ex:

```
>> v=-5.1:0.2:-4

v =

  -5.1000   -4.9000   -4.7000   -4.5000   -4.3000   -4.1000
```

If the parameter **step** is omitted, Matlab assumes it equal to one

Ex:

```
>> x=3:6

x =

    3     4     5     6
```

Graphics

The ability to do plotting in Matlab is very useful. Here we shall describe how to plot 2D graphs, 3D surfaces, 2D contours of 3D surfaces and vector arrows.

To plot a number of ordered pairs (x,y) connected by straight lines, we built row vectors **x** and **y** containing the **x** and **y** values and ask Matlab to make a plot.

Ex:

```
>> x=1:5;
>> y=0:0.2:.8;
>> plot(x,y)
```

Vectors x and y contain two different coordinates of the same set of points in the plane and thus must have the same length.

Matlab functions are vectorized and constructing the needed vectors to graph a build –in function is done by first constructing a vector of the desired x values, and than the corresponding y values come from applying the buil-in function to the x vector:

Ex:

```
>> x=0:0.1:pi;
>> y=sin(x);
>> plot(x,y)
```

Some useful commands in making plots are

xlabel('x axis label'), ylabel('y axis label'), title('title) labels the horizontal and the vertical axes and the title, respectively, in the current plot.

axis([a b c d])	changes the viewing window of he current graph
grid	adds a rectangular grid to a current plot
hold on	freezes the current plot so that the subsequent plots will be displayed on the same axes
hold off	releases the current plot
subplot	puts multiple plots in one graphics window
legend	creates a small box inside the current plotting window that identifies multiple plots in the same window

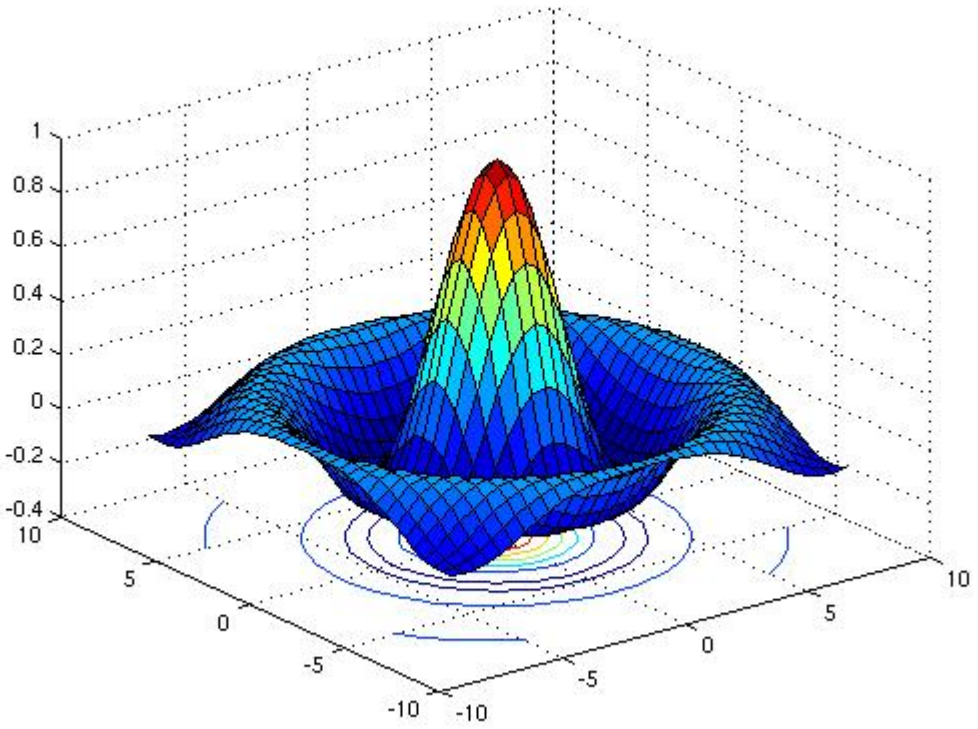
Three-dimensional plots

Making a 3D plot in Matlab is often done in two steps. First we define a mesh in an area where we want to make a 3D plot. This area is defined by two vectors **x** and **y**, of the length **n** and **m**, which represent **x**- and **y**-values in the mesh. Now we make a mesh with a command **[X,Y]=meshgrid(x,y).** The matrix **X** contains the vector **x** copied in **m** rows, and the matrix **Y** contains the vector **y** copied in **n** columns**.** The second step is to compute the value of a function using **X** and **Y** matrices.
To plot a surfaces of a function in Matlab we can use following commands

mesh(Z), surf(X,Y,Z), surfc(X,Y,Z)

Ex:

```
>> [X, Y]=meshgrid(x,y);
>> R=sqrt(X.^2+Y.^2)+eps;Z=sin(R)./R;
>> surfc(X,Y,Z)
```



If we want to plot only contours of function **Z** above we can use command **contour(x,y,Z,n)** where **n** is the number of contour lines**-2** we want to display**.**

Vector arrows can be plotted with a command **quiver(X,Y,dZdx,dZdy,n)** where **n** controls the size of the arrows.

Ex:

[dZdx,dZdy]=gradient(Z);
quiver(X,Y,dZdx,dZdy,3)

Assigning the values in the matrices

Ex: If we want to put the value of the third column in a matrix **A** equal to 5 and we know that matrix A has size n x m, than we can write **A(:,3)=5*ones(n,1)**

Ex: If we want to make a matrix **B** from several existing vectors **a, x** and **d** where all three vectors are of the same length we can write **B=[a x d] .**

A simple FOR loop

Ex:

```
for i=2:N_i-1
        for j=2:N_j-1

        Expression such as k_E(i,j)=…
        end
end
```

A simple WHILE loop

Ex:

```
while (res>max_e)
……
end
```

Printing out the figure in some JPG file

For printing out figures we can use command **print –djpg filename**

Measuring the time of the calculation

If we want to measure the time a calculation requires we can use commands **tic** and **toc**

Ex: If we want to find out the time that our while loop requires we can write

```
tic
while (res>max_e)
……
end
toc
```

Plotting several plots

If we want to plot several plots in separate windows we can use command **figure(n)** where **n** is the number of the figure.

Ex: figure(1)

Cleaning the memory of the Matlab

If we start writing a new program and want to be sure that our variables don't have some old values we can write command **clear all** at the beginning of the program.