Introduction to MATLAB
Practical 1

Daniel Carrera

2014

1 Introduction

Matlab is both an application and a programming language. It was initially designed to teach linear algebra to university students. The fundamental building blocks are vectors and matrices. Over the years, Matlab has grown to become one of the most popular tools in science and engineering. The language is succint, and very practical for solving problems in scientific computing.

I believe that the best way to learn Matlab is "*hands on*". I have tried to design this toturial to encourage experimentation. I assume no prior knowledge of Matlab, but you must already have it installed.

2 Whirlwind tour of MATLAB

2.1 Start the program

Locate the Matlab program and run it. After the program starts, you will be presented with a command shell where you enter Matlab instructions. In the rest of this tutorial, any line that starts with "`>>`" denotes the Matlab command shell. For example:

```
>> 2 + 3

   5
```

2.2 Matlab as a calculator

At the simplest level you can use Matlab as a scientific calculator. Enter the following code and see what it does.

```
>> 2 * 5
>> 2 + 3 * 5 + 1/2
>> 3^2
>> 3^3
>> 3^4
>> 4^(-1)
>> 4^(1/2)
>> sqrt(2)
>> sqrt(4)
>> sqrt(9)
>> sqrt(25)
>> pi
>> cos(pi)
>> sin(pi/2)
>> tan(pi/4)
>> atan(1)
>> atan(tan(0))
```

You can enter arithmetic operations directly into the command window. The symbol `^` denotes exponentiation (so you write 5^{17} as `5^17`). Matlab has some pre-defined constants such as `pi`, and it supports most standard math functions such as square roots and trig functions.

Enter the following code and see what it does:

```
>> exp(0)
>> exp(1)
>> exp(2)
>> e = exp(1)
>> log(e)
>> log(10)
>> log(e^173)
>> log(exp(173))
>> log10(e)
>> log10(10)
>> log10(1000)
>> log10(10^189)
```

The function `exp(n)` returns e^n where $e = 2.7183...$ is the base of the natural logarithm. The function `log()` gives the natural logarithm while `log10()` gives the logarithm base 10.

Enter the following code and see what it does:

```
>> 1i
>> 5i
>> 2 + 1i
>> (-1)^0.5
>> exp(i*pi)
>> log(i)
>> sin(i)
>> sinh(i)
```

The syntax "`3i`, `5i`, etc" lets you write imaginary numbers safely. Do not use the variable "`i`" on its own because it can be overwritten (try `i = 7`) and your program will behave unpredictably. You can use Matlab to work with complex numbers, including complex exponentiation, logarithms and trigonometry.

Enter the following code and see what it does:

```
>> help log10
```

Matlab comes with extensive help about all its functions. You are encouraged to use the help system to learn how functions work and possibly discover other functions that might be helpful in your work.

2.3 Machine precision

Try to determine whether Matlab stores numbers as single precision or double precision. You can use Wikipedia to learn about the differences between single and double precision.

2.4 Vectors and linear algebra

Enter the following code and see what it does:

```
>> u = [1 2 3]
>> v = [3 4 12]
>> u + v
>> u - v
>> 3 * u
>> 3 * u - v
>> dot(u,v)
>> dot(v,v)
>> sqrt( dot(v,v) )
>> norm(v)
```

One way to create a vector is to enter the vector components directly inside square brackets. In this example you created row vectors $\vec{u} = [1, 2, 3]$ and $\vec{v} = [3, 4, 12]$. You can do vector addition, subtraction, scalar multiplication and dot product. The function `norm` returns the norm, or the *magnitude* of a vector.

Enter the following code and see what it does:

```
>> u
>> v
>> u'
>> v'
>> u' * v
>> u * v'
```

The vectors `u` and `v` are row vectors. You need to take the transpose of one of the vectors to multiply them. An apostrophe denotes matrix transpose, and the `*` operator performs matrix multiplication.

Enter the following code and see what it does:

```
>> w = [1 ; 2 ; 3 ]
>> w * v
>> v * w
```

You can also enter a column vector directly if you separate the vector components with semicolons instead of spaces.

2.5 Conditionals

Enter the following code and see what it does:

```
>> a = 1;
>> b = 2;
>> if a == b
       x = 2;
   elseif a >= 3
       x = 7;
   else
       x = 0;
   end
```

Experiment with different values of `a` and `b` and different conditions (e.g. `==` `<=` `=>` `<` `>`). What happens if `a` and `b` are arrays? Try the following examples:

```
>> a = [1 2 3];
>> b = [1 2 3];
>> if a == b
       x = 2;
   else
       x = 0;
   end
>>
>> b = [1 2 4];
>> if a == b
       x = 2;
   else
       x = 0;
   end
>>
>> if any(a == b)
       x = 2;
   else
       x = 0;
   end
>>
>> if all(a == b)
       x = 2;
   else
       x = 0;
   end
>>
```

Experiment with different arrays (e.g. `b = [3 1 2]`). What do the functions `any` and `all` do?

2.6 Datasets and plotting

Notice how the default multiplication operator performs matrix multiplication. In astronomy we are often more interested in taking two data sets and applying operations term-by-term. Enter the following code and see what it does:

```
>> u = [1 2 3];
>> v = [3 4 12];
>> u .* v
>> u ./ v
>> v ./ u
>> u .^ 2
>> v .^ 2
```

The operators `.*`, `./` and `.^` perform *component-wise* multiplication, division and exponentiation. Notice also that the semicolon at the end of a line (";") causes Matlab to withhold output. When you write longer programs, you will usually not want to see output for every intermediate step.

Enter the following code and see what it does:

```
>> 3:11
>> 3:2:11
>> 2.5:7.5
>> 2.5:2:7.5
>> 2:1.5:7
```

Matlab offers a number of short-cuts for commonly used vectors and matrices. The syntax `a:b` creates a row vector from `a` to `b` in steps of 1. The syntax `a:q:b` creates a row vector from `a` to `b` in steps of `q`.

Enter the following code and see what it does:

```
>> x = 1:10;
>> y = x .^ 2;
>> plot(x, y)
>> plot(x, cos(x))
```

By this point you have learnt enough Matlab to produce many useful plots. Use `x = a:q:b` to create a series of X values, and use vector or array operations to produce the corresponding Y values. The `plot` command is very flexible and can produce a lot of nice plots.

Enter the following code and see what it does. Use the `Up` arrow key to pull up previous lines so that you can save typing:

```
>> x = 0:0.1:2*pi;
>> y = sin(x);

>> plot(x,y,'r')
>> plot(x,y,'g')
>> plot(x,y,'b')
>> plot(x,y,'m')
>> plot(x,y,'y')
>> plot(x,y,'k')
>> plot(x,y,'c')

>> plot(x,y,'r-')
>> plot(x,y,'ro')
>> plot(x,y,'r+')
>> plot(x,y,'r*')
>> plot(x,y,'rx')
>> plot(x,y,'r^')

>> plot(x,y,'ro-')
>> plot(x,y,'r+-')
>> plot(x,y,'go-')

>> y2 = cos(x);
>> plot(x,y,'r', x,y2,'b')
>> legend('Sine','Cosine')
>> title('Sines and Cosines')
>> xlabel('This is the X axis')
>> ylabel('This is the Y axis')
```

The `plot` function accepts an optional formatting string to define the colour, plot style and data label. You can include any number of data sets in the same plot. Run `help plot` to learn more about Matlab's plotting features. Enter the following program in the Matlab editor:

Enter the following code and see what it does:

```
>> plot( x, sin(x), 'r' )
>> hold on
>> plot( x, cos(x), 'b' )
>> hold off
```

The `hold on` command causes Matlab to draw any new plotting commands on top of the existing window. It is one way to draw several data sets on the same plot. The command `hold off` returns Matlab to standard behaviour.

2.7 Making Matlab programs

Click on the "New Script" button to open the built-in Matlab editor. You can use the editor to write Matlab programs. Matlab programs contain a sequence of Matlab commands, allowing you to store complex operations fore later use. When you save the program, the file name must have the extension "`.m`". Enter this in the editor:

```
% % % % % % % % % % % % % % % %
%
% wave.m -- An simple animation of a travelling wave.
%
% % % % % % % % % % % % % % % %

for j = 1:4
    j
end
```

Save this file as "`wave.m`" in the current working directory. If the Matlab working directory is not the same directory that contains the program, Matlab will not be able to find it. After the file is save, Enter the following in the command window and see what it does;

```
>> wave
```

This program should print the integers from 1 to 4. This program only contains a short *for loop*. A *for loop* allows you to repeat commands a fixed number of times as the index variable (in this case `j`) steps through the values of an array.

Replace the for loop with the one below and run the program again to see what it does:

```
for j = 1.5:0.5:3.5
    j
    j * j
end
```

The index variable in a for loop can step through the values of any array. The program also shows Matlab comments. A Matlab comment goes from the `%` sign to the end of the line. Always comment your programs generously clearly.

Update the "`wave.m`" program with the following one. Run the program and see what it does:

```
% % % % % % % % % % % % % % % %
%
% wave.m -- An simple animation of a travelling wave.
%
% % % % % % % % % % % % % % % %

x = 0:0.1:4*pi;  % X values from 0 to 4*pi.
v = -0.1;        % Wave velocity.

for t = 1:1000
    plot( x, sin(x + v*t), 'r' )
    drawnow
    pause(0.2)
end
```

The `drawnow` command causes Matlab to update all figure windows. Without it, the plot may not be shown until the for loop is complete. The commands `sleep` and `usleep` cause Matlab to suspend the program for a given number of seconds (for `sleep`) or microseconds (for `usleep`). Without a delay, the for loop may run too fast.

2.8 Exercises

It is important that you do the exercises, as they will help sharpen your skills with loops, plotting and the Matlab help. The Bonus problem is different - it is meant to help you play and explore Matlab but is not directly relevant to this course.

Exercise 1: The Fibonacci sequence is a series of integers, starting with "1,1", where the next value is the sum of the previous two values. The first few terms of the Fibonacci sequence are: "1,1,2,3,5,8,13,..". Write a program that prints the first 50 values of the Fibonacci sequence.

Exercise 2: Use the help function to figure out how to add axis labels and a title to a plot. Then make a plot of your favourite function with labels on both axes and a title.

Bonus problem: Type in the following code and see what it does:

```
[x,y] = meshgrid(-8:0.5:8);
r = sqrt(x.^2 + y.^2);
z = sin(r) ./ r;
mesh(x,y,z)
surf(x,y,z)
```

Use your mouse to grab the image and rotate it, so you can see it from different angles. Use the help function to learn what `meshgrid`, `mesh` and `surf` do. Then try to make a plot with an interesting shape (a bowl, a volcano, a sombrero, etc).