Santa Fe Institute
Complex Systems Summer School 2005
Matlab Tutorial

Jonas Buchli

June 10, 2005

1 Preamble

Motivation and Goal This tutorial gives a hands-on introduction to MATLAB , no previous knowledge of MATLAB is required. However, a certain familiarity with common programming concepts, and for later sections some basic math is helpful. As dynamical systems constitute an important aspect in the studies of complex systems, I decided to include a section on the simulation of dynamical systems with MATLAB .
I would be glad to get your feedback on the tutorial! Contact me at
`jonas@buchli.org`

How the tutorial is organized The tutorial is split up in several sections dealing with different aspects of MATLAB . Some of the sections contain a Basic and an Advanced Section. It should be sufficient to work through the Basic sections in order to get you going with MATLAB . The advanced sections you may want to skip if this is your first encounter with MATLAB - these sections explain further commands or details that prove very useful to simplify your daily life with MATLAB . Further there are Exercises in each section, again split up in Basics (B) and Advanced (A).

The idea of the tutorial is to get you started with MATLAB by showing the first steps and an overview what exists, to learn to use the commands and more about MATLAB use of the manual pages and the Mathworks help web-site is recommended, see next section.

2 Starting Matlab

Basics MATLAB is specialized on *numerical* calculations, as opposed to packages specialized on symbolic computation e.g. Mathematica. It is an interpreted language, often one works on a command prompt, or runs simple scripts.

Start MATLAB by typing `matlab` in your shell. This will bring up MATLAB with its default configuration and a graphical user interface (see the advanced sessions for some useful startup flags). The graphical user interface (GUI) gives additional information and shortcuts. We do not focus on the GUI here as it is mostly self explanatory. Note that all the information and functions offered by the GUI can also be accessed directly by the command prompt.

If you entered an operation that takes too long time to complete you can interrupt it by hitting ctrl-c. Be aware that MATLAB is slightly sensitive on such interruptions and can crash while doing so (see Sect.4 on how to save your data).

You can get help on a command by entering `help <cmd name>` . For example:
`>> help plot` brings up the help page of the `plot` command.

Advanced Some useful MATLAB startup flags include:

`-nodesktop` Brings up MATLAB without the graphical user interface but only the shell.

`-nojvm` Brings up MATLAB without using the java virtual machine, no graphical user interface is started but only the shell. This sometimes speeds things up or makes it more stable. It is very useful when you do batch processing with MATLAB . The basic stuff works, there are a few things (mainly related with GUIs) that do not work without the Java virtual machine running.

`-nosplash` Does not show the initial splash screen. Useful in conjunction with the above flags in environments where there is no X environment available (distributed computing, etc.).

In addition to `help` the command `lookfor` is very useful. It searches the first line of each command help page for the given keyword.

Exercises

B.1 Start MATLAB , look at the help pages of commands given in the document (e.g. `help ones`), note how they are organized - what information do they provide?

B.2 Check out the MATLAB help page given in the references, bookmark it!

A.1 Use the command `lookfor` to find the command that returns the eigenvalues and eigenvectors of a matrix.

3 Basic syntax

Basics The main way of giving your input to MATLAB is the command prompt. It is designated by `>>` after which you can enter your commands and

hit enter to conclude and execute the commands. You can directly enter algebraic expressions:
```
>> 1+2

ans =

3
```

An assignment to a variable is done with the equal sign:
```
>> a = 4

a =

4
```

The variable can then be accessed and used by its name:
```
>> a
>> ans =

4
>> a+1

ans =

5
```

A semicolon can be used to suppress the output of an operation:
```
>> b = a+1;
>> b

b =

5
```

Note that MATLAB is case sensitive (variables and commands):
```
>> A

???  Undefined function or variable 'A'.
```

The most important data structure in MATLAB is the matrix. The matrix is denoted with square brackets. A semi-colon is used to introduce line breaks.

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

A =
1 2
3 4
```

Elements of an matrix can be accessed by indexes `A(row,col)` (note that the index starts with 1, i.e. the first element has index 1):

```
>> A(2,1)

ans =

3


>> A(1,2)=5

A =

1 5
3 4
```

If a single index is given the elements are accessed consecutively down the columns from "top left" to "bottom right":

```
>> A(2)

ans =

3
```

A way of specifying long vectors (1xN dimensional matrices) in a convenient way is the column notation: `[begin:step:end]`, e.g.
`>> v = [1:0.1:2];` assigns the numbers 1,1.1,...,2 to `v`.

Besides the common operators (`+`,`-`,`*`,`/`) the "power" operator is worth noting: `2^ 5` gives 32. In addition to the normal operators, MATLAB knows the dotted version some of these operators: `.* .^ ./`. These operators operate element-wise, instead of doing the "mathematical" matrix operation. For example:
`>> rand(3,4)^ 2` gives an error while
`>> rand(3,4).^ 2` works fine.

Be careful when specifying matrices as you can easily specify *huge* matrices, especially simple to do with stepping:
`[X,Y] = meshgrid([0:0.01:100],[0:0.01:100])`
would produce two matrices using 800MB of memory and certainly makes the computer paging for a while.

Advanced MATLAB supports N-dimensional matrices, or "tensors":

```
>> r(:,:,1) = [1 2; 3 4];
>> r(:,:,2) = [5 6; 7 8];
```

The column notation can also be used to access several items at once:

```
>> v = [1:10];
>> v(3:2:7)

ans =

3 5 7
```

`>> r2 = R(2,:);` copies the second row of `R` to `r2` .

The keyword `end` can be used as a substitute for the highest index:

```
>> v(end-3:end)=1
```

thus

```
>> v(end)==v(length(v))

ans =

1
```

is always true.

We can use smaller matrices to build up larger ones by matrix concatenation:

```
>> A = rand(3,4);
>> B = rand(2,4);
```

`>> C = [A;B];` the command `repmat` comes handy when a matrix can be constructed by repeatedly using a smaller matrix:

```
>> repmat([0 1; 1 0],3,2)
```

MATLAB has some commands to easily create special matrices:

zeros - Gives a matrix of given dimensions filled with zeros.

ones - Gives a matrix of given dimensions filled with ones.

diag - Lets you create diagonal matrices.

rand – Returns uniformly distributed random matrix $\in [0, 1]$ (of given dimensions. See also `randn` and reference on the Statistics Toolbox for more distributions in the reference section.

You can get a list of defined variables by typing `who` or `whos` where the later one shows the memory usage with it. The size (in number of elements, not memory) of a variable can be obtained by `length` and `size` .

The memory can be cleared by typing `clear` , single variables can be deleted by `clear varname` .

Another data type of MATLAB are cell arrays, they are denoted with curly braces (}). The advantage is that they can contain elements of different length

and types:
`>> c = {'a',1}` The elements of a cell array can be accessed by `()` or . The difference is that the first notation gives you back another cell array (consisting of the specified elements, while the second notation gives back "normal" arrays:

```
>> c(1)


ans =

'a'

>> iscell(c(1))


ans =

1

>> c{1}


ans =

a

>> iscell(c{1})


ans =

0
```

Furthermore, Matlab supports structs, which are useful to group together some information:

```
>>  s.key = 'first';
>>  s.value = 1;
>> s

s =

key:  'first'
value:  1
```

Matlab keeps a history of the commands you typed, you can fetch them by hitting the "up"-arrow on your keyboard. You can also start a command you

would like to find back to narrow down the choices in the history, i.e.
`>> pl` and then hitting the up-arrow would only find lines starting with `pl`, such as `plot(t,s)`.

Exercises

B.1 Make yourself comfortable with the way how to assign matrices, try out the column notation.

B.2 Try out the different ways to access elements of a matrix: single index, col/row indexing, using vectors as indices.

B.3 The same as in B.2 but this time make assignments. How can you assign to several elements in the same time?

B.4 Generate 2 $N \times N$ matrices `A` and `B`. Do `A*B` and `A.*B` . Explain the results.

B.5 Execute this command `rand(3,2)^ -1` . What do you get and why?

A.1 Construct in a single assignment the following vector: 0,0.1,. . . ,10,5,4.5,. . . ,1

4 Input/Output

The simplest way to save data into files is the command `save` . Without arguments it saves all current variables to the file `matlab.mat`.
`>> save data s t` saves the variables s t into file `data.mat`. The default format of the file is a binary format (MAT file). The advantage of this format is that it supports storing multiple variables (of possibly different) type into one file. However, often it would be convenient to have a simple ASCII format to save a matrix to:
`>> save -ascii data.txt A`

This is a straight forward format where the matrix is written row by row separated by white-spaces. It has the drawback, that it supports only 2D matrices and the variable names are not stored with it. A file can be read by the `load` command.

Advanced MATLAB also supports C-style file output and input. The following writes a random number with 8 digits after the comma into file `random.dat`.

```
fid = fopen('random.dat','w');
fprintf(fid,'%12.8f \n',rand);
```

A convenient way to start an external program from MATLAB is the command `system` :
`>> [s,w] = system('ls');` `s` contains the return value of the program (normally zero if everything went fine) and `w` contains a copy of the standard output

of the command, (in the example above the directory listing). On the interactive prompt you can use an exclamation mark as a shell escape:
```
>> !ls
```

Exercises

B.1 Save and read back variables of your choice to a 1) mat file 2) ASCII file.

A.1 Use fprintf to write a random numbers with 4 digits after the comma into an ASCII file.

A.2 Write a C program (or in any other language) that reads a value from a file and writes its value to another file. Use the commands in this section to write the initial value from MATLAB and read it back once the C program has finished.

5 Scripts and Functions: `m`-files

Instead of typing and retyping every command on the prompt MATLAB supports scripts. Scripts are simple text files, that are executed line by line as if the commands would be typed on the prompt[1]. As a convention the files have the suffix `*.m`. The scripts - being simple ASCII files - can edited with any simple editor. MATLAB has also a built-in editor: `edit myscript` brings it up, editing `myscript.m`. To execute the script type the filename without the suffix on the prompt:
`>> myscript` looks for the file `myscript.m` and executes it.

In almost the same way new functions can be defined. The difference between scripts and functions are mainly the following: Functions can take parameters or arguments. Furthermore, the scope of variable is restricted in functions, i.e. any variables used in function are only known to the function itself and are never global. The variables being global in scripts can lead to difficult to detect errors, where a variable with the same name is used twice or still present from previous scripts. So be careful when you are using variables in scripts, use explicit names and initialize them correctly.

The convention for the functions is the following: the first line contains the definition of the return values and variable names: `ret = foo(arg1,arg2)` after that usually a block of comments is followed (comments are started with the percent sign: `%`. This block of comments is displayed as help page with the command
`>> help foo` . Note that you do not need to give an explicit return statement, since the return value is defined by name. You can however use the keyword `return` to return from your function (for example in an `if` statement). The function should be stored in a `m`-file with the name of the function (`foo.m` in our example). See Sec. 8 for an example of a function.

[1]Note that there are actually some small differences between scripts and the command prompt.

In order to organize the flow of your scripts or functions the following control structures can be used:

- The for-loop:

  ```
  for it=1:10;
    disp(it);
  end;
  ```

- A while loop:

  ```
  it = 10;
  while(it >1);
    disp(it);
    it=it-1;
  end;
  ```

- If-else statements:

  ```
  if(a>0)
    disp('a positive');
  else
    disp('a negative');
  end;
  ```

- And finally the switch:

  ```
  switch str
    case 'apple'
      disp('you got an apple')
    case 'pear'
      disp('this time a pear')
    case 'strawberry'
      disp('strawberries now')
    otherwise
      disp('unknown exotic fruit')
  end;
  ```

A hint for debugging your script: use `disp()` to print out debugging information or try removing some semicolons in order to see more output and understand what happens. You can also use the command `keyboard` to stop your script at a given point and return to the interactive console. Enter `return` to continue with the execution of the script. Note that you can speed up the execution of MATLAB by preallocating arrays instead of appending to them step by step. Execute the following two pieces of code to see the difference:

```
% slow
clear;
for it=1:10e5;
 a(it) = sqrt(it);
end;

% faster
clear;
a = zeros(1,10e5);
for it=1:10e5;
 a(it) = sqrt(it);
end;
```

Exercises

B.1 Use a for loop to write ten times a message on the console. Do the same with the while loop.

6 Some advanced operations and math

Matlab supports complex numbers in a straight forward manner:

```
>> i

ans =

0 + 1.0000i
```

related to complex number are the commands `abs(z)` and `angle(z)` to obtain the modulus and argument respectively.

Furthermore, Matlab has some nice algorithms built in

fft - Implementation of the (Fast) Fourier Transformation

hilbert - The Hilbert Transform, useful to find the phase of certain types of signals.

filter - Discrete filter

polyval - Find the value of a polynomial at given points

polyfit - Fit a function with a polynomial of given order

regress - Multiple linear regression using least squares

nlinfit - Nonlinear least-squares regression

conv - Convolution

corr - Correlation

max - Find the maximum

min - Find the minimum

mean - Mean value

var - Variance

std - Standard deviation

sign - You guess it, don't you?

find - Find all elements satisfying a condition.

transpose The transpose of a matrix: $\mathbf{A}^T$. Can also be written as `A.'` .

input Read input from the user.

pause Pause the script for a given time or until key pressed.

textread A more versatile file input function.

Some simple statistics with Matlab The MATLAB Statistics Toolbox provides a lot of random number generators satisfying the most common used distributions, see the the Statistics Toolbox on the MATLAB web-page (in the references). A few commands that are useful

hist - Computes a histogram

boxplot - Nifty statistics graphs.

qqplot - Compare your data samples against distributions.

7 Graphics

Basics MATLAB has a vast number of utilities to produce graphs and figures. Probably the most basic one is the `plot` command. The following commands plot a sine curve:

```
>> t = [0:0.1:10];
>> s = sin(t);
>> plot(t,s);
```

The vectors can be followed by a specification of the line type. Several curves can be produced at the same time.

```
>> plot(t,s,'r:',t,cos(t),'k>');
```

Plots the sine curve in red and dotted, and the cosine along with black diamonds. See `help plot` for more options on the line styles and the advanced section for some more formatting options.

`xlabel` , `ylabel` respectively let you label the axis. Latex notation can be used:

```
>> xlabel('\Sigma_\alpha')
```

`text` allows to place a text label at a given coordinate. `line` places straight lines, with specified start and end coordinates into the graph.

In order to plot 2D data, the commands `mesh` and `surf` are useful. The difference is the fill style of the surface that is created, either only a mesh grid or a filled surface. A variant that is sometimes useful is `contour` or `contourf` which plots curves of the same value for the given matrix.

Matlab deletes by default the old figure as soon as a new plot command is issued. Thus if it is needed that the graphic stays in order to draw over it we have to disable this behavior by typing `hold on` . This command can be reverted by typing `hold off` .

Advanced A graphic is normally composed of a figure (window), one or several axes in the figure window, and in there one or several graphic elements like lines,patches, etc.

As a matter of fact all graphic functions return so called handles. A handle is a way to accessing and modifying the graphic object at a later stage. This is commonly done with the `get` and `set` commands:
`>> h=plot(t,s);`
`>> get(h)` returns a list with properties of handle `h`. For example:

```
>> get(h,'color')

ans =

0 0 1
```

returns the color of the line. In the same way
`>> set(h)` can be used to see a list of modifiable properties of a handle. Then,
`>> set(h,'linewidth',3)` draws the lines with width 3. `set(h,'color','r')` sets the color to red.

A list of additional useful graphical commands:

figure - Called without argument it brings up a new figure. Called as `figure(n)` - it makes figure n the active one.

gcf - Returns the active figure.

gca - Returns the active axis.

clf - Clears the current figure.

grid on/off - Switches the drawing of grid on or off.

axes - Creates new axes.

subplot - Used to create evenly spaced sub-figures.

axis - Get or set the current axis parameters.

loglog - Plots the given vectors in a log-log plot

semilog - Plots the given vectors in a semi-log plot-

logspace - Creates a vector evenly spaces in logspace.

meshgrid - Gives back two matrices (X,Y) that can be used as index for two dimensional functions.

plot3d - Lets you plot 3D data.

bar - Plots bar diagrams.

bar3 - Plots 3D bars.

quiver - Plots arrow plots, useful to create vector field plots.

The command `print` lets you print out the graphics directly from a script. Most common used graphics and image formats are supported:
`>> print(1,'-deps2','-r300', 'fig1.eps');` prints figure 1, using resolution of 300dpi and format Encapsulated Postscript v2 into file fig1.eps. Ready to include into Latex documents.

With a combination of the described graphic commands it is possible to create publication quality graphs directly out of Matlab . A few things that normally have to be done in order to arrive there: Adjust line width and styles (careful with color), increase font size, place labels at correct position, add lines/arrows/text and other things that help to understand the graph, reposition the axes and finally print it in high resolution by using `'-r<res>'` option in the `print` command and/or a vector oriented format (such as EPS).

8 Dynamical Systems in Matlab

Matlab also includes a very convenient solver that can be used to integrate numerically (nonlinear) dynamical systems. It is probably best we walk trough a small example.

The goal is to integrate the Fitzhugh-Nagumo Equations, a nonlinear oscillator based on the simplification of neuron dynamics. The equations of the FHN Oscillator are the following:

$$\dot{x} = c\left(y + x - \frac{x^3}{3} + f\right) \quad (1)$$

$$\dot{y} = -\frac{(x - a + by)}{c} \quad (2)$$

The ODE systems needs now to be implemented as a Matlab function, which we store in `fhn.m`:

```
function dy = fhn(t,y)
%
% The FITZHUGH-NAGUMO Oscillator
%
a = 0.1;
b = 0.5;
```

```
c = 10;
f = 1;

dy = [ c * (y(2) + y(1) - y(1)^3/3 + f); ...
        -(y(1) - a + b * y(2))/c ];
```

The function takes to arguments, the time t and the current system state y. It returns a column vector `dy` which corresponds to $[\dot{x}, \dot{y}]^T$.

This equation can now be integrated by using for example the Runge-Kutta algorithm, with the steps output as chosen by the adaptive step integrator:
`>> ode45(@fhn,[0 100],[0.1 0.2])` If rather the results should be stored for further processing it can be called like this:
`>> [t,h]=ode45(ode45(@fhn,[0:0.1:100],[0.1 0.2])`

In addition, here we used a time vector with fixed time step at which we wish the obtain the output. Note that the integrator internally still uses adaptive steps (this is of importance when integrating e.g. Langevin equations).

A note concerning the efficiency of the integration in MATLAB : Since the ODE function is implemented as "interpreted" function which has to be called for each time step, the integration in MATLAB has a very large overhead and is slow. If you need to run longer simulations or larger systems it is better to implement the integration in C/C++ or the like (see http://birg.epfl.ch/page56667.html). The difference in speed is very significant (several orders of magnitude!).

Exercises

B.1 The Lorenz Equations are the following

$$\begin{aligned} \dot{x} &= \sigma(y - x) && (3) \\ \dot{y} &= -xz + rx - y && (4) \\ \dot{z} &= xy - bz && (5) \end{aligned}$$

Use the MATLAB ODE-solver to integrate the system and plot the famous Lorenz attractor (Parameters: $\sigma = 10$,$r = 28$,$b = \frac{8}{3}$).

A.1 The equation for the van der Pol Oscillator is

$$\ddot{x} = \mu(p^2 - x^2)\dot{x} - \omega^2 x \qquad (6)$$

Use the MATLAB ODE-solver to integrate the system and plot the limit cycle of the van der Pol Oscillator (vary parameter p). Note that you will have to transform the equation into a system of first order ODE first.

9 Toolboxes

What makes MATLAB so versatile is its extensibility with so called Toolboxes. Toolboxes for almost every aspect of engineering and scientific life exist, some commercial others freely available. As I can not give a complete list let me highlight some of the most important (under the aspect of Complex Systems research):

Statistics - `http://www.mathworks.com/products/statistics/`

Signal processing - `http://www.mathworks.com/products/statistics/`

Neural networks - `http://www.mathworks.com/products/neuralnet/`

Wavelet Toolbox - `http://www.mathworks.com/products/wavelet/`

Cellular automata - `http://www.teuscher-research.ch/rbntoolbox/`

Time-frequency Toolbox - `http://iut-saint-nazaire.univ-nantes.fr/˜auger/tftb.html`

10 Alternatives to Matlab

There exist several packages that are similar to MATLAB and are freely available.

Octave - `http://www.octave.org/`

Scilab - `http://scilabsoft.inria.fr/`

matplotlib (plotting only) - `http://matplotlib.sourceforge.net/`

References

The Mathworks Website: http://www.mathworks.com

The help desk: http://www.mathworks.com/access/helpdesk/help/helpdesk.html

Acknowledgments

Some inspiration for this tutorial stems from a MATLAB introductory course by the Automatic Control Laboratory at the ETH Zürich.

I would like to thank Ludovic Righetti for rereading the tutorial and giving constructive comments. Thanks to Ronan Fleming for the time-frequency toolbox, Brett Calcott for the matplotlib, and Alexander Hellervik for the `keyboard` command.