

MS Macroeconomics – Tech Slides 1
Intro to Stochastic Processes, Time Series, and Matlab

Camilo Granados
University of Texas at Dallas

Stochastic Processes

A stochastic process refers to a family of random variables whose behaviour depends totally or partially on a stochastic or random component.

A variable can take on values **deterministically**, such as $x = 3$, or $x = 2 + 3t$ with $t = 1, 2, \dots, 10$

Or **randomly** such as x : *number displayed after rolling a dice*.

We know such values can be 1, 2, 3, 4, 5, or 6. But we can't know with certainty which will be the actual value if we roll it once.

This latter type are denoted as **random variables**, and their already observed values are **realizations**.

In macroeconomics, we will form an expectation on variables whose actual future value is unknown. But we can form an idea on its expected value if we study its features as an **stochastic process**.

As with the dice, sometimes we know the probability of each value the random variable can take
In that case, the **expected value** of a random variable (let X be such variable) is:

$$E(X) = \sum_i x_i Pr(x_i)$$

Where $Pr(x_i)$ is the probability that X takes on the value x_i .

In the case of the dice, where each value may appear with probability $\frac{1}{6}$ the expected value is $\frac{1}{6}(1) + \frac{1}{6}(2) + \frac{1}{6}(3) + \frac{1}{6}(4) + \frac{1}{6}(5) + \frac{1}{6}(6) = 3.5$.

For the case in which X is continuous, the expected value will be $E[X] = \int_{-\infty}^{\infty} xf(x)dx$ where $f(x)$ is the PDF or probability density function.

Properties of the Expected Value

1. The expected value of a constant is the constant itself: Let a be a constant, then $E[a] = a$
2. Expected value of a constant times a random variable: $E[aX] = aE[X]$
3. The expected value is a linear operator: $E[aX + b] = aE[X] + E[b]$
4. Mixing these properties we get the expectation of a weighted sum of random variables:
 $E[aX + bY] = aE[X] + bE[Y]$
5. The expected value cannot pass through non-linear functions: let g be a non-linear function, then $E[g(X)] \neq g(E[X])$

But we know this result for concave/convex functions (**Jensen's Inequality**, used a lot in stats and econ)

$$E[g(X)] = \begin{cases} > g[E(X)] & \text{if } g \text{ is convex} \\ = g[E(X)] & \text{if } g \text{ is linear} \\ < g[E(X)] & \text{if } g \text{ is concave} \end{cases}$$

Conditional Expectation

We can also think of the **joint probability for pairs of events** or for several random variables. That is, we will consider a joint probability function $Pr(x, y) = Pr(X = x, Y = y)$.

In that case we say that X and Y are jointly distributed.

If X and Y are jointly distributed we can get info about one variable if we know about the other one:

- ▶ Conditional Probability: $Pr(y|x) = Pr(y|X = x) = \frac{Pr(x,y)}{Pr(x)}$
- ▶ Conditional Expectations: $E[Y|X = x] \equiv E[Y|X] = \sum_y y \cdot Pr(y|X = x)$

Moreover, evaluating this expectation over the possible values of X would yield $E[Y]$:

$$E_X[E[Y|X = x]] = E[Y]$$

In macroeconomics, this concept is useful to denote the expectation conditional on an information set:

$$E_{t-1}[Y_t] = E[Y_t|I_{t-1}]$$

Where I_{t-1} is the information set available at time $t - 1$.

A useful property of these expectations is the **Law of Iterated Expectations**:

1. $E[E[Y|X]|X] = E[Y|X]$, then $E_{t-j}[E_{t-j}[Y_t]] = E_{t-j}[Y_t]$
2. $E[E[Y|I_1]|I_2] = E[Y|I_2]$, where I_1 and I_2 are two information sets such that $I_2 \subseteq I_1$ (I_1 has at least all the information contained in I_2). Then,

$$E_{t-j-k}[E_{t-j}[Y_t]] = E_{t-j-k}[Y_t]$$

This property implies that when we conditioned on several information sets, **the smaller information set dominates (i.e. the less complete)**.

Intuitively (*read this part slowly*), the expectation about a future variable we were able to form conditional on the info today, conditional (again) on the information we had a day before ...

...is just the expectation of the variable of interest conditional on the info available yesterday.

Higher order expectations

We can also consider expectations of higher orders:

$$E(X^j) = \sum_i x_i^j Pr(x_i), \quad \text{for } j = 2, \dots$$

We will use $j = 2$ to obtain the variance of our random variables.

The **variance** of X is defined as:

$$\text{Var}(X) = \sigma_X^2 = E[(X - E(X))^2] = E[X^2] - (E[X])^2$$

The last line implies, that if we already now the expected value ($E[X]$), then we just need to get the expected value of the squared variable $E[X^2]$ to determine the variance.

A more general concept is the **covariance**, it measures the linear comovement between two variables:

$$\text{Cov}(X, Y) = E[(X - E[X])(Y - E[Y])] = E[XY] - E[X]E[Y]$$

Notice the variance is just the covariance of a variable with itself.

We will use this last formula repeatedly, particularly whenever we want to substitute for the expectation of the product of random variables ($E[XY] = \text{Cov}(X, Y) + E[X]E[Y]$).

Properties

For the variance:

1. $\text{Var}(X) \geq 0$
2. The variance of a constant is zero.
3. Let a, b be constants, then: $\text{Var}(aX + b) = a^2\text{Var}(X)$
4. $\text{Var}(X) = E[X^2] - (E[X])^2$

For the covariance:

1. $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y) + 2\text{Cov}(X, Y)$
2. $\text{Cov}(X, Y) = E[XY] - E[X]E[Y]$

Independent and Uncorrelated variables

- ▶ If two variables X and Y are independent, then they are uncorrelated ($\text{Cov}(X, Y) = 0$) and the expectation of the product is just the product of the individual expectations $E[XY] = E[X]E[Y]$
- ▶ If two variables are not correlated, we cannot guarantee that they are independent.

Distributions

Some variables take values on (countable and uncountable) an infinite support, so it may be unfeasible to list the probability for each possible realization value.

However, we may know how their probability is defined. That information may be enough to characterize its moments of interest (e.g. expected value and variance). Some examples are:

- ▶ Uniform Distribution: $X \sim U[a, b]$, and its density function is:

$$f(x) = \begin{cases} \frac{1}{b-a}, & \text{if } a \leq x \leq b \\ 0, & \text{otherwise} \end{cases}$$

- ▶ Exponential Distribution: $X \sim \exp(\lambda)$

$$f(x) = \begin{cases} \lambda e^{-\lambda x}, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

- ▶ Normal Distribution: $X \sim N(\mu, \sigma^2)$

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

From the latter, what we'll use the most is that the expected value of X is μ and its variance σ^2 .

Bivariate Normal

We may also know the **joint** distribution of a combination of random variables. A commonly used is the bivariate (or multivariate) Normal distribution:

$$\begin{pmatrix} X \\ Y \end{pmatrix} \sim N \left(\begin{pmatrix} \mu_X \\ \mu_Y \end{pmatrix}, \begin{pmatrix} \sigma_X^2 & \sigma_{X,Y} \\ \sigma_{X,Y} & \sigma_Y^2 \end{pmatrix} \right)$$

Where $\sigma_{X,Y} = \text{Cov}(X, Y)$, $\sigma_X^2 = \text{Var}(X)$, $\sigma_Y^2 = \text{Var}(Y)$.

A useful property we will use in our analysis of rational expectations is:

$$E[X|Y] = \mu_X + \frac{\sigma_{X,Y}}{\sigma_Y^2}(y - \mu_Y)$$

You can also express this formula in terms of the correlation $\rho_{X,Y}$: $E[X|Y = y] = \mu_X - \rho_{X,Y} \frac{\sigma_X}{\sigma_Y}(y - \mu_Y)$

Log-normal distribution

A random variable X is said to have a log-normal distribution if its log is normally distributed. Let $\log(X) = Z \sim N(\mu, \sigma^2)$. Then, we can find the expected value of X as:

$$E[X] = E[\exp(Z)] = \exp\left(\mu + \frac{1}{2}\sigma^2\right)$$

Autoregressive Processes

Time series whose value is given by a deterministic component, which is a function of its previous realizations plus an stochastic component.

Widely used in macroeconomics.

The simplest one is the $AR(1)$ model, or autoregressive process of order one:

$$y_t = \alpha_0 + \alpha_1 y_{t-1} + \epsilon_t \quad (1)$$

$$\epsilon_t \stackrel{i.i.d.}{\sim} N(0, \sigma_\epsilon^2) \quad (2)$$

This process will be stationary if $|\alpha_1| < 1$

Its expected value and variance is given as follow:

$$E[y_t] = E[\alpha_0 + \alpha_1 y_{t-1} + \epsilon_t]$$

\vdots

$$\mu_y = \frac{\alpha_0}{1 - \alpha_1}$$

(filling the steps in the dots is part of your HW1)

For the variance we just need to find $E[y_t^2]$ and then use: $Var(y_t) = E[y_t^2] - (E[y_t])^2$.

The variance will be:

$$Var(y_t) = \frac{\sigma_\epsilon^2}{1 - \alpha_1^2}$$

(again, for the HW1 you should provide the steps leading to these -or similar- results.)

Summary

We have provided you a **very** basic toolkit on statistics (stochastic processes and time series).

However, this is just the tip of the iceberg. Some of these topics have their own courses and the cases and applications go well beyond what we showed.

Yet, this may be just enough for you to understand most of the applications in this course.

Here are some references in case you want to know more:

- ▶ *Econometric Analysis*, by William H. Greene.
- ▶ *Introduction to Econometrics*, by Bruce Hansen.
(<https://www.ssc.wisc.edu/~bhansen/probability/>)
- ▶ *Time Series for Macroeconomics and Finance*, by John Cochrane, unpublished lecture notes
(https://www.johnhcochrane.com/s/time_series_book.pdf).

Introduction to Matlab

MATLAB stands for MATrix LABoratory.

It is a software and a programming language designed for numeric computing. It is based mainly in calculations around matrices manipulations. However, there's many more things you could use it for (plots, algorithms, etc.).

Widely used in engineering. In economics is also very popular, particularly in macroeconomics

For those familiar with other languages, the code syntax is **very** similar to R.

MATLAB is *case sensitive*: `mean([1 2 3])` will work, whereas `Mean([1 2 3])` won't

Why should we use MATLAB and not R, Python, Mathematica, etc? ... just *because*. These and many other options would work too.

But...you wanna work with the same tool used by the people in your field, so that you can benefit from their work (public goods). For Macro that implies knowing your way around MATLAB.

Where to get it

Great news! for UTD students MATLAB is free:

<https://atlas.utdallas.edu/TDClient/30/Portal/Requests/ServiceDet?ID=211>

What Is It?

MATLAB combines a desktop environment tuned for iterative analysis and design processes with a programming language that expresses matrix and array mathematics directly. It includes the Live Editor for creating scripts that combine code, output, and formatted text in an executable notebook. UT Dallas has a site license for MATLAB.

Who Is Eligible to Use It?

Students, Faculty, Staff

Click in 'Installing MATLAB on Personally Owned Computers'

Follow the instructions. Implies creating a Mathworks account. Make sure to use your UTD email.

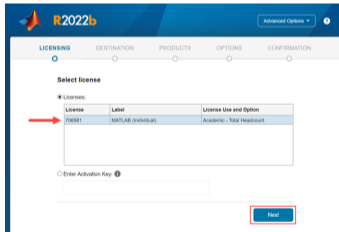
Make sure to use your UTD email:



MathWorks
Email
No account? Create one!
By signing in, you agree to our privacy policy.

Next

If it asks, use the Student License Option (Not the activation key one)



LICENSING DESTINATION PRODUCTS OPTIONS CONFIRMATION

Select license

* Licenses:

License	Label	License Use and Option
70681	MATLAB (Individual)	Academic - Total Headcount

Enter Activation Key

Next

Everything else is standard except: when picking products, select also the Econometrics Toolbox and the Optimization Toolbox.

Select products (recommended products are preselected)

<input type="checkbox"/>	Select All
<input type="checkbox"/>	Deep Learning Toolbox
<input type="checkbox"/>	DSP System Toolbox
<input checked="" type="checkbox"/>	Econometrics Toolbox
<input type="checkbox"/>	Embedded Coder
<input type="checkbox"/>	Filter Design HDL Coder
<input type="checkbox"/>	Financial Instruments Toolbox
<input type="checkbox"/>	Financial Toolbox
<input type="checkbox"/>	Fixed-Point Designer
<input type="checkbox"/>	Fuzzy Logic Toolbox
<input type="checkbox"/>	Global Optimization Toolbox
<input type="checkbox"/>	HDL Coder

Next

Then you just confirm:

R2022b

Advanced Options

LICENSING DESTINATION PRODUCTS OPTIONS CONFIRMATION

Confirm selections

LICENSING
100% of products

DESTINATION
C:\Program Files\R2022b

PRODUCTS
1 of 110 products
4.16 GB required

Begin Install

The screenshot displays the MATLAB R2019b environment. The top menu bar includes options like HOME, PLOTS, APPS, SHORTCUTS, EDITOR, PUBLISH, and VIEW. Below the menu is a toolbar with icons for file operations (New, Open, Save, Compare, Print), navigation (Find Files, Go To, Find), editing (Insert, Comment, Indent, Breakpoints), and execution (Run, Run and Advance, Run Section, Run and Time).

The main window is divided into three panes:

- Current Folder:** Shows the directory structure, including files like `data_ps1.xlsx`, `PS1.tex`, `PS1.pdf`, `PS1.out`, `ps1q2_code.m`, `hpfilter.m`, `PS1.log`, `PS1.synctex.gz`, and `PS1.aux`.
- Editor:** Contains the MATLAB script `ps1q2_code.m`. The code includes comments and commands for loading data, filtering, and plotting. Key lines include:


```

      data = xlsread('data_mtl.xlsx', 2);
      gdp = data(:,2); % 2 refers to the column of the variable
      con = data(:,3); % consumption (or other variable), 3 refers to column
      gdp_hp = hpfilter(log(gdp), 1000); % Trend
      con_hp = hpfilter(log(con), 1000);
      gdp_cyc = log(gdp) - gdp_hp; % Cyclical component (detrending)
      con_cyc = log(con) - con_hp;
      x = 1990.25:0.25:2008.75; % date variable for plots
      figure % figure opens a window for a new graph useful to keep all plots if you run the whole
      subplot(2,1,1);
      plot(x, log(gdp)); hold; plot(x, gdp_hp, 'red'); axis tight;
      ylabel('log of GDP');
      title('Real Australian GDP and Trend');
      subplot(2,1,2);
      plot(x, gdp_cyc); axis tight;
      title('Cycle of GDP (Using HP filter)');
      figure
      subplot(2,1,1);
      plot(x, log(con)); hold; plot(x, con_hp, 'red'); axis tight;
      ylabel('log of Consumption');
      title('Real Australian Total Consumption and Trend');
      subplot(2,1,2);
      plot(x, con_cyc); axis tight;
      title('Cycle of Total Consumption (Using HP filter)');
      % Statistics for filtered variables
      % Standard deviations
      std(gdp_hp)
      std(con_hp)
      std(gdp_cyc)
      std(con_cyc)
      
```
- Command Window:** Shows the output of the script, including the results of `autocorr` for `gdp_cyc` and `con_cyc`. The output for `con_cyc` is:


```

      con_cyc = autocorr(con_cyc)
      con_cyc =
      1.0000
      0.5982
      0.2759
      0.1274
      -0.0019
      -0.1868
      -0.1661
      -0.2328
      -0.2764
      -0.2624
      -0.2529
      -0.2051
      -0.1399
      -0.0708
      -0.0212
      0.0184
      0.0261
      -0.0031
      -0.0033
      0.0015
      0.0114
      
```
- Workspace:** Lists variables created in the workspace:

Name	Value
ans	0.0140
con	243x1 d
con_cyc	243x1 d
con_hp	243x1 d
cor_con	21x1 do
cor_gdp	21x1 do
con1	1.0000
con2	0.5432
data	243x3 d

Your code goes in an m-file shown in the middle. After running it, the output will show up in the *command window*.

To the left, you will see what variables you have created in the *workspace*. Above it, you'll see the content of your *working directory*

Usage

You can use MATLAB as a calculator (type directly on the command window and press Enter):

```
>> 1+2+3+4+5
ans =
15
>> 1*2*(3+4)/5
ans =
2.8000
```

But if you want to keep your code (e.g. for future use) you should use an **m-file**.

From that file you can run the whole code by typing the name of the file in the command window.

Or you can select a part of the code and run it with Ctrl-click -> "evaluate selection", or Shift+fn+F7 (in Mac).

Usage (cont.)

Syntax: `name = command;`

Here we create an object given by the output of the command.

Also the semicolon prevents the output from showing in the *command window*.

Notice: no need to specify the type of an object (it does it automatically), you can overwrite objects/variables just by creating them again.

Order for mathematical expressions: From left to right + PEMDAS (Parentheses are evaluated first, then Exponents, Multiplications/Divisions, then Additions/Subtractions).

Whenever in doubt, enclose what you want evaluated first in parentheses.

Example:

```
>> 1+2+3*4
ans =
15
>> (1+2+3)*4
ans =
24
```

Most important command: `help function-name`, e.g., `help mean` (doc function-name also works)

Functions

We can use *built-in* functions to make specific calculations.

Examples: `mean`, `max`, `std`

But we can also create our own functions:

- ▶ Standard way: create a separate m-file named as the function where we set our calculation code in function format (i.e. define inputs and declare a procedure to be executed in order to generate an output). Example: `hpfilter.m`
- ▶ Less usual: with very small functions we can create *anonymous* functions in the same m-file we're working:

```
>> f1 = @(x) x^2 + 2*x + exp(x);  
f1(3)
```

```
f2 = @(x,a) x^a + a*x + exp(x);  
f2(3,2)  
ans =  
35.0855
```

Setting your directory and accessing/saving data

You can get the current directory with `pwd` (print working directory):

```
>> pwd
ans =
'/Users/jocagraca-mbp15/Dropbox/Teaching/UTD/MacroeconomicsCore_MS/HW/PS1'
```

if you want to change it use `cd(path)` (change directory):

```
cd('/Users/jocagraca-mbp15/Dropbox/Teaching/UTD/MacroeconomicsCore_MS')
```

Saving results:

Let's say we have our result variables (A, B, C) and want to save them, just use `save` command:

```
save exampleFile A B C
```

Here we specified we want to save A, B, and C. If we want to save everything we have in the current workspace we just use: `save exampleFile`.

The resulting file (saved in your working directory) is: `exampleFile.mat`

Loading data:

Matlab Files: Exactly the same, but use `load` instead (e.g. `load exampleFile`)

Loading Excel data

If your data starts at the cell A1 in the first sheet of your spreadsheet just use:

```
data = xlsread('data_ps1.xlsx') ;
```

More complicated applications:

```
% Read a specific range of data in the second sheet:
```

```
subsetA = xlsread('myExample.xls', 2, 'B2:C3')
```

```
% Read from a named worksheet:
```

```
B = xlsread('myExample.xls', 'MySheet')
```

Just as with `save/load`, you can save your results as excel files with `xlswrite`

Matrices and Vectors

By default, MATLAB is used to work with matrices/vectors. Then, you can create them directly without particular commands:

Create a row vector: `>> A = [1 2 3]`

Create a column vector (with semicolon you move down to next row): `>> A = [1; 2; 3]`

Create a matrix (element by element):

```
>> C = [1 2 3; 4 5 6; 7 8 9]
```

```
C =
```

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

We can extract particular elements from the matrix with `(rows, columns)`:

```
>> C(2,3) %Displays the element in the 2nd row and 3rd column of C.
```

```
ans =
```

```
6
```


Matrices and Vectors (cont.)

If you want to extract a whole row, or a whole column, then you indicate you want all the elements in the dimension with colon (:):

```
>> C(:,2)
```

```
ans =
```

```
2
```

```
5
```

```
8
```

```
>> C(1,:)
```

```
ans =
```

```
1    2    3
```

Similarly, you can extract sequences of rows or columns:

```
>> C(1:2,2:3) %Displays the first two elements of the 2nd and third column
```

```
ans =
```

```
2    3
```

```
5    6
```

Matrices and Vectors (cont.)

Other useful commands:

Extract the diagonal elements of a matrix: `>> diag(C)`

Delete elements (including whole rows or columns) by setting them equal to `[]`:

```
>> C(1,:) = []
```

```
C =
```

```
4     5     6
```

```
7     8     9
```

Generate a predetermined matrices:

```
>> D=zeros(3,3) %displays a 3x3 matrix of zeros
```

```
>> E=eye(4,4) %displays a 4x4 identity matrix
```

```
>> F=ones(3,1) %displays a 3x1 matrix of ones
```

Operations with matrices

Let's create another matrix: `>> A = magic(3) %3x3 matrix w/ rows, cols, diag summing same`

Addition: `>> A+C`

Subtraction: `>> A-C`

Multiplication: `>> A*C`

Inverse of a matrix:

```
>> Ainv = inv(A);
```

```
Ainv =
```

```
0.1472    -0.1444    0.0639
-0.0611     0.0222    0.1056
-0.0194     0.1889   -0.1028
```

Testing if it worked:

```
>> A*Ainv
```

```
ans =
```

```
1.0000         0   -0.0000
-0.0000    1.0000         0
0.0000         0    1.0000
```

Other useful commands: `det()`, `trace()`, `eig()` (determinant, trace and eigenvalues)

Other applications with matrices

Concatenation: if the dimensions are conformable we can concatenate matrices/vectors/

```
>> D = [A C]
D =
 8     1     6     1     2     3
 3     5     7     4     5     6
 4     9     2     7     8     9
>> E = [A; C]
E =
 8     1     6
 3     5     7
 4     9     2
 1     2     3
 4     5     6
 7     8     9
```

Element-wise operations: sometimes it's useful to perform calculations with (or on) each element of the matrix separately

```
>>A^2 %Performs the usual matrix multiplication A*A
>>A.^2 %Squares each element of the matrix A
```

Loops

"Control structure" that affects the flow of the executed program.

Saves you time. Use it when you are doing stuff -repeatedly-. Don't do it for one-time applications.

Types: `for`, `while`, `if-else-elseif`

For: executes a set of command a given number of times

```
>> x = zeros(10,1);
for i=1:10
x(i,1) = i;
>> x' % ' transposes a matrix
ans =
1     2     3     4     5     6     7     8     9    10
```

While: executes a command until a conditions is no longer met

```
>> y = 20;
while y >= 18 %the command below will be executed as long as y >= 18
y = y - 1;
end
>> y
y =
17
```

Loops (cont.)

If, else:

```
>> a = 30; b = 11;
>> if a < 20
c = 3+2;
else
c = 4;
end
>> c
c =
4
```

With Else-if:

```
>> if a > 30
c = 12;
elseif b == 11
c = 13;
else
c = 0;
end
>> c
c =
13
```

When writing conditionals, you can also consider the intersection of conditions (Cond 1 AND Cond 2) or the union (Cond 1 OR Cond 2):

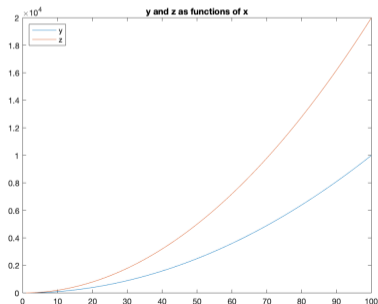
```
>> if 3 > 4 & 3 < 4 % & or && refers to AND
c = 6;
elseif 3 > 4 | 3 < 4 % | or || refers to OR
c = 7;
else
c = 0;
end
>> c
c =
7
```

Plots

To make a plot of a variable x and y use: `>> plot(x,y)`

To add another variable z to that same plot use `hold` and then `>> plot(x,z)`

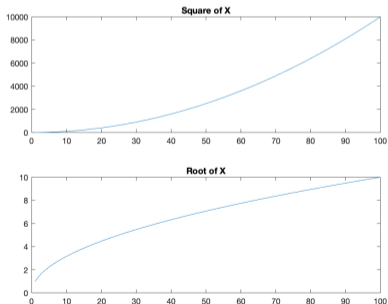
```
x = 1:100; %x will be the numbers from 1 to 100
y = x.^2; z = 2*x.^2;
plot(x,y); hold; plot(x,z)
title('y and z as functions of x')
legend('y','z','Location','NorthWest');
```



Plots(cont)

You can also make a matrix of plots in a window with `>> subplot(rows,columns,number of plot):`

```
subplot(2,1,1);  
plot(x,y);  
title('Square of X')  
subplot(2,1,2);  
plot(x,sqrt(x))  
title('Root of X')
```



Deleting, quitting

If you want to delete an element (called `A`): `clear A`

If you want to delete everything in your workspace: `clear all`

To close all the plots' windows: `close all`

To clear your command window: `clc`

Finally, you can close MATLAB by typing this in the command window: `quit`

Conclusion

This is only a tiny glimpse of what you can do in MATLAB.

The best way to learn is by trying it yourself (rather than reading manuals or these slides).

Nobody memorizes the whole code and syntax. You get the gist, then look up the manual or google what you need (e.g. StackOverFlow).

In the course website you can find an application with a code for the Question 2 of the problem set.