

The course is designed assuming no knowledge of the program. The main reference is Winistörfer, P. and Canova, F. (2008), "Introduction to Matlab".

- 1 The Command Window
- 2 Matrix Construction and Manipulation
- 3 Data Importing and Exporting
- 4 Mathematical Functions
- 5 Random Numbers
- 6 Graphics
- 7 Scripts and Functions
- 8 Controlling the Flow (if, for, do-while)

Evaluation: Final homework

1. The Command Window

1. The Command Window

- When you open Matlab the command window is created and made the active window.
- It is the interface which with we communicate with Matlab.
- The prompt (») indicates that Matlab is ready to obtain instructions.
- From now on we will denote Matlab code as `typeset` letters.
- Command can be executed by pressing the ENTER key.
- Typing the command `clear x` deletes the value of variable `x` .
- `clear` deletes all values that have been created during a Matlab session.
- Matlab is case sensitiv.

1. The Command Window

- If you type `>> x` Matlab does the following
 - 1 It verifies if `x` is a variable in the Matlab workspace.
 - 2 If not, it verifies if `x` is a built-in-function.
 - 3 If not, it verifies if a *m-file* named `X.m` exists in the current directory.
 - 4 If not, it verifies if `X.m` is anywhere on the Matlab search path
- If Matlab cannot find `X.m` anywhere an error message will appear.
- A Matlab search path can be added or removed by selecting *Set Path* in the *file* menu.
- A semicolon (`;`) at the end of the line tells Matlab not to display the results from the executed command.
- Text following `'%'` is considered as comment and is not executed.
- We can press the Up-arrow key to recall previous command lines.

1.2 The Help Facility

- The help facility provides information about Matlab functions, commands and symbols.
- The command `»help` returns a list of directories that contain Matlab related files.
- Typing `»help topic` displays the help about this topic if it exists.
- `help elmat` provides information on elementary matrices and matrix manipulation.

2. Matrix Construction and Manipulation

2.1 Building a Matrix

- Defining a vector:

rowvector: » `V=[1 2 3 4]`

columnvector: » `V=[1; 2; 3; 4]`

- Defining a matrix:

» `B=[1 2 3 4; 5 6 7 8]`

B is a 2×4 matrix, i.e.,

$$B = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}$$

- Defining a 3D matrix:

» `A(:, :, 1)=B`

» `A(:, :, 2)=[9 10 11 12; 13 14 15 16]`

A is $2 \times 4 \times 2$ matrix.

2.1 Building a Matrix

There are some special matrices which are extremely useful:

- The zero matrix: `zeros(m,n)` creates a $m \times n$ matrix of zeros.

» `B=zeros(2,3)`

results in

$$B = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

- The ones matrix: `ones(m,n)` creates a $m \times n$ matrix of ones.

» `B=ones(2,3)`

results in

$$B = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

2.1 Building a Matrix

- The identity matrix: `eye(n)` creates a $n \times n$ matrix with ones along the diagonal and zeros everywhere else.

» `B=eye(3)`

results in

$$B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The colon notation is a shortcut for producing rowvectors.

`a:b:c` produces a vector of entries starting with the value a , incrementing by the value b until it gets to c .

» `V=1:4`

results in

$$V = (1 \quad 2 \quad 3 \quad 4)$$

» `V=0.32:0.1:0.6`

results in

$$V = (0.32 \quad 0.42 \quad 0.52)$$

2.2 Accessing a Matrix

- You can access the element at the i th row and the j th column by typing `B(i, j)`
- Accessing i th row `B(i, :)`
- Accessing j th column `B(:, j)`
- Accessing i th to $(i+c)$ th row and j th to $(j+n)$ th column `B(i:i+c, j:j+n)`
- Accessing last row and last column `B(end, end)`
- `diag(B)` extracts diagonal elements of a matrix.
- `tril(B)` creates lower triangular matrix.
- `triu(B)` creates upper triangular matrix.

2.2 Accessing a Matrix

- Linear indexing: `B(n)`
Accesses the n th element if counting columnwise
- `find(B>c)`
finds linear index of values of B which are bigger than c .
- `B(find(B>c))`
access values of B which are bigger than c .
- logical expressions:
& 'and', | 'or', == 'equal', ~= 'not equal'
<= 'smaller or equal', < 'strictly less', >= 'bigger or equal',
> 'strictly bigger'

2.2 Accessing a Matrix

Consider matrix

$$B = \begin{pmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \\ 9 & 4 & 8 \end{pmatrix}$$

» `F=B(1:2,2:3)`

yields

$$F = \begin{pmatrix} 2 & 3 \\ 6 & 7 \end{pmatrix}$$

» `F=B(2:end,[1 3])`

yields

$$F = \begin{pmatrix} 5 & 7 \\ 9 & 8 \end{pmatrix}$$

2.2 Accessing a Matrix

Consider matrix

$$B = \begin{pmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \\ 9 & 4 & 8 \end{pmatrix}$$

» `a=B(4)`

yields `a = 2`

» `F=find(B>4)`

yields

$$F = \begin{pmatrix} 2 \\ 3 \\ 5 \\ 8 \\ 9 \end{pmatrix}$$

» `F=B(find(B>4))`

yields

$$F = \begin{pmatrix} 5 \\ 9 \\ 6 \\ 7 \\ 8 \end{pmatrix}$$

Exercises

- 1 Write a 50×1 vector of 5s. (Don't write it manually)
- 2 Write the following rowvector (1 2 ... 30 30 ... 2 1).
- 3 Define `» A=[-1 2 -3; 4 5 -6]` and set all positive elements to 1 and all the negative ones to 0.
- 4 Given matrix A from the previous point set all the negative elements in the first row to zero.
- 5 Given matrix A from the previous point set all the negative elements and all elements bigger or equal than 5 to zero.
- 6 Given matrix A from the previous point set all elements lying between 2 and 5 to zero.
- 7 Type `» A>0`. What's the difference to `» find(A>0)` ?

Exercises: Solutions

- 1 » `ones(50,1)*5`
- 2 » `[1:30 (-30:-1)*-1]`
» `[1:30 30:-1:1]`
- 3 » `A(A<0)=0`
» `A(A>0)=1`
- 4 » `A(1,find(A(1,:)<0))=0`
- 5 » `A(find(A<0|A>=5))=0`
- 6 » `A(find(A>=2&A<=5))=0`
- 7 It does not produce an index but a logical vector in which 1 stand for 'true' and 0 stands for 'false'.

2.3 Matrix Operations (Vectorisation)

Making Vectors from Matrices and Reverse:

Define

$$B = \begin{pmatrix} 1 & 2 \\ 5 & 6 \\ 9 & 4 \end{pmatrix}$$

The vectorisation of B can be obtained by `»A=B(:)`

$$A = \begin{pmatrix} 1 \\ 5 \\ 9 \\ 2 \\ 6 \\ 4 \end{pmatrix}$$

Suppose we have A and want to get B . This can be done by typing

```
» reshape(A,3,2)
```

2.3 Matrix Operations (Transpose)

Transpose of a Matrix:

Define

$$B = \begin{pmatrix} 1 & 2 \\ 5 & 6 \\ 9 & 4 \end{pmatrix}$$

The Transpose of B can be obtained by `»A=B'` and yields

$$A = \begin{pmatrix} 1 & 5 & 9 \\ 2 & 6 & 4 \end{pmatrix}$$

2.3 Matrix Operations (Basic Operators)

- Addition: +
- Substraction: -
- Multiplication: *
- Right Division: / ($B/A = BA^{-1}$)
- Left Division: \ ($A \setminus B = A^{-1}B$)
- Exponentiation: ^ ($A^{(2)} = A^2$)
- Array Operations (element-by-element): To indicate an array operation the standard operator is preceded by a dot, i.e., .* ./ .^

```
» x=[1 2 3]
```

```
» y=[4 5 6]
```

```
» x.*y
```

```
ans =
```

```
4 10 18
```

2.4 Built-in Functions

- `sort(A)`: sorts the elements in ascending order. If A is a matrix, `sort(A)` treats the columns of A as vectors returning sorted columns.
- `size(A)` returns the size of matrix A .
`[m,n]=size(A)` returns the size of matrix A in variables m and n .
» `A=[1 2; 3 4; 5 6]`
» `[m,n]=size(A)`
yields
`m=3`
`n=2`

2.4 Built-in Functions

- `sum(A)` returns the sum of elements if A is a vector. If A is a matrix, `sum(A)` returns a rowvector of the sums of each column.

```
» A=[1 2; 3 4; 5 6]
```

```
» C=sum(A)
```

yields

```
C = 9 12
```

- `cumsum(A)` returns the cumulative sum. If A is a matrix, it returns a matrix of the same size as A containing sums for each column.

```
» C=cumsum(A)
```

yields

```
C = 1 2  
    4 6  
    9 12
```

2.4 Built-in Functions

- `min(A)` (`max(A)`) returns a row vector containing the minimum (maximum) element of each column if A is a matrix.
- `mean(A)` returns a row vector containing the mean of the elements of each column.
- `median(A)` returns a row vector containing the median of the elements of each column.
- `std(A)` returns a row vector containing the standard deviation of each column.
- `inv(A)` returns the inverse of a square matrix.
» `A=[3 2; 1 4]`
» `B=inv(A)` yields

$$B = \begin{pmatrix} 0.4000 & -0.2000 \\ -0.1000 & 0.3000 \end{pmatrix}$$

2.4 Built-in Functions

- `[P,D]=eig(A)` returns a matrix of eigenvectors (P) and eigenvalues (D) of a quadratic matrix.

```
» A=[0.95 0.05; 0.25 0.7]
```

```
» [P,D]=eig(A)
```

yields

$$P = \begin{pmatrix} 0.7604 & -0.1684 \\ 0.6495 & 0.9857 \end{pmatrix}$$

$$D = \begin{pmatrix} 0.9927 & 0 \\ 0 & 0.6573 \end{pmatrix}$$

- `kron(A,B)` produces the Kronecker product of A and B

- 1 From the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

obtain the columnvector

$$a = (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9)'$$

- 2 Multiply $a = (1 \ 2 \ 3)$ and $b = (2 \ 4 \ 6)$ in such a way to obtain $c = (2 \ 4 \ 18)$.

Exercises

- 1 `reshape(A', 9, 1)`
- 2 `c=a.*b`

3. Data Importing and Exporting

3.1 Exporting Data Files

Before starting we have to indicate the correct *current directory* in the *command toolbar*. As an exercise we will create 4 different data files.

- Clear the workspace.

- Define

```
»x=[1:1:50]'
```

```
»y=[0.1:0.1:5]'
```

- Create the Matlab data file *result1.mat*

```
» save result1
```

The two vectors are saved in the file *result1.mat*.

- Create another Matlab file *result2.mat*

```
» save result2 y
```

The vector *y* is saved in the file *result2.mat*

3.1 Exporting Data Files

- Create another Matlab file *result3.txt*

- » `save result3 y -ascii`

The vector *y* is saved in the 8-digit ASCII text format in the file *result3.txt*

- Create another Matlab file *result4.dat*

- » `save result4.dat x`

The vector *x* is saved in the file *result4.txt*

3.2 Importing Data Files

Now we will import the 4 datafiles we previously saved.

- Clear the workspace.
- Import data file *result1.mat*
» `load result1`
The two vectors *y* and *x* are imported in the workspace.
- clear workspace again
- Import data file *result2.mat*
» `load result2`
The vector *y* is imported in the workspace.

3.2 Importing Data Files

- Clear the workspace again.
- Import data file *result3.txt*
» `load result3`
The vector called *result3* is imported in the workspace.
- clear workspace again
- Import data file *result4.dat*
» `a=importdata('result4.dat')`
The vector *a* is imported in the workspace.

3.3 Importing Data from Excel

- Matlab can also read data from .xls files directly (no need to transform everything into ASCII format).
- `data = xlsread(filename, sheet)`
reads the specified worksheet, where `sheet` is either a positive, double scalar value or a quoted string containing the sheet name.
- `data = xlsread(filename, 'range')`
reads data from a specific rectangular region of the default worksheet (Sheet1). Specify range using the syntax '`C1:C2`', where `C1` and `C2` are two opposing corners that define the region to be read. For example, '`D2:H4`' represents the 3-by-5 rectangular region between the two corners `D2` and `H4` on the worksheet. The range input is not case sensitive and uses Excel A1 notation.

3.3 Importing Data from Excel

- `data = xlsread(filename, sheet, 'range')`
reads data from a specific rectangular region (range) of the worksheet specified by sheet.
- Example.xls contains data on GDP (column B), Consumption (column C), Investment (column D), Exports (column E) and Imports (column F). To read these 5 times series into Matlab type
» `data=xlsread('Example.xls', 'National Account', 'B4:F205')`

4. Mathematical Functions

4.1 Rounding Numbers

Mathematical functions are applied on an element-by-element basis.

- There are several ways of rounding and chopping real numbers to give integers.
- Define `» x=pi*(-1:3)`
`x=(-3.1416 0 3.1416 6.2832 9.4248)`
- `round(x)` rounds the elements of `x` to the nearest integers.
- `fix(x)` rounds the elements of `x` to the nearest integers towards zero.
- `floor(x)` rounds the elements of `x` to the nearest integers towards minus infinity.
- `ceil(x)` rounds the elements of `x` to the nearest integers towards infinity.
- `sign(x)` is the Signum function. For each element of `x`, `sign(x)` returns 1 if the element is greater than zero, 0 if it equals zero and -1 if it is less than zero.

4.2 Trigonometric Functions

- $\sin(x)$ gives the Sine.
- $\sinh(x)$ gives the Hyperbolic Sine.
- $\cos(x)$ gives the Cosine.
- $\cosh(x)$ gives the Hyperbolic Cosine.
- $\tan(x)$ gives the Tangent.
- $\tanh(x)$ gives the Hyperbolic Cosine.
- $\sec(x)$ gives the Secant.
- $\operatorname{sech}(x)$ gives the Hyperbolic Secant.

4.3 Exponential Functions

- `exp(x)` is the exponential, i.e., e^x .
- `log(x)` is the natural logarithm.
- `log10(x)` is the common logarithm.
- `sqrt(x)` is the square root.

4.4 Complex functions

- `abs(x)` is the absolute value.
- `angle(x)` is the phase angle.
- `conj(x)` is the complex conjugate.
- `imag(x)` is the complex imaginary part.
- `real(x)` is the complex complex real part.

5. Random Numbers

5. Random Numbers

Matlab has a number of functions which allow to draw random numbers.

- `rand(m,n)` produces $m \times n$ matrix containing pseudorandom values drawn from the standard uniform distribution on the open interval(0,1).
- To generate values from the uniform distribution on the interval $[a, b]$ use
$$r = a + (b-a) .* \text{rand}(m,n)$$
- `randn(m,n)` returns an $m \times n$ matrix containing pseudorandom values drawn from the standard normal distribution.
- To generate values from a normal distribution with mean μ and standard deviation σ .
$$r = \mu + \sigma .* \text{randn}(m,n)$$

5. Random Numbers

- `betarnd(A,B,m,n)` returns a $m \times n$ matrix of random numbers chosen from the beta distribution with parameters A and B.
- `chi2rnd(V,m,n)` returns a $m \times n$ matrix of random numbers chosen from the chi-square distribution with V degrees of freedom.
- `exprnd(MU,m,n)` returns an $m \times n$ matrix containing random numbers chosen from the exponential distribution with location parameter MU.
- `gamrnd(A,B,m,n)` returns an $m \times n$ matrix of random numbers chosen from the gamma distribution with shape parameter A and scale parameter B.
- `lognrnd(MU,SIGMA,m,n)` returns an $m \times n$ matrix of random numbers generated from the lognormal distribution with parameters MU and SIGMA.
- `trnd(V,m,n)` returns an $m \times n$ matrix of random numbers chosen from Student's t distribution with V

- 1 Compute $\frac{\sqrt{\log e^4/2}}{2\pi}$.
- 2 Write a command that will simulate the n throws of a pair of dice.
- 3 Given a 20×2 matrix of iid $N(1,2)$ random vectors, build the corresponding demeaned values.
- 4 Build two $N \times 1$ vectors k and y where each entry of y is $y_t = 3 + 0.5X_{1t} + 0.9X_{2t} + \epsilon_t$ where X_{it} is $Normal(i, 2)$ and ϵ_t is $Normal(0, 1)$ and $k_t = 3 + 0.5X_{1t} + 0.9X_{2t}$ Using the standard OLS formula estimate the coefficients,

$$\text{i.e., } \hat{\beta} = (X'X)^{-1}X'y \text{ where } X = \begin{pmatrix} 1 & X_{11} & X_{21} \\ \vdots & \vdots & \vdots \\ 1 & X_{1N} & \end{pmatrix} \text{ for}$$

$N = 50, 1000, 5000$. What do you observe?

Exercises: Solutions

- 1 » `(sqrt(log(exp(4)))/2)/(2*pi)`
- 2 » `floor(1+6*rand(n,2))`
- 3 » `rv=1+2*randn(10,2)`
» `Demeaned=rv-ones(10,1)*mean(rv)`

Exercises: Solutions

Since for y data was simulated with an error term, we expect the betas to be around 3, 0.5 and 0.9, respectively. The estimates will get closer as N increases. Since for k we simulated the data without an error term, the betas will be exactly 3, 0.5 and 0.9, respectively, independently of N .

```
» N=50 ;  
» X1=1+2*randn(N,1) ;  
» X2=2+2*randn(N,1) ;  
» Y=3+0.5*X1+0.9*X2+randn(N,1) ;  
» K=3+0.5*X1+0.9*X2 ;  
» X=[ones(N,1) X1 X2] ;  
» Betay=inv(X'*X)*X'*Y ;  
» Betak=inv(X'*X)*X'*K ;
```

Define $N=1000$, $N=5000$, and redo the commands.

6. Graphics

6.1 Histogram

One important command when dealing with graphics is `clf` which clears the graphic screen. Graphs can be edited manually by opening the plot tools.

- `hist(x)` draws a 10-bin histogram of the vector x . If Y is a matrix, `hist` works down the columns.
- `hist(x,m)` draws a m -bin histogram of the vector x .
- `hist(x,c)` draws a histogram of the vector x using the bins specified in c .
- ```
» x=randn(5000,1);
» hist(x)
» hist(x,100)
» hist(x,-3.9:0.2:3.9)
```

## 6.2 Plotting Series versus their Index

- `plot(x)` plots the data in `x` against versus its index
- `plot([x y])` plots the data of `x` and `y` versus their index in the same graph. Therefore, if the input is a matrix `plot` plots the data of each column versus their index.
- ```
» x=randn(50,1);  
» plot(x)  
» plot([x randn(50,1)])  
» plot([x randn(50,1) randn(50,1)])
```

6.3 2D-Scatterplot

- `plot(x,y,S)` is the general form of a scatterplot
- `x` and `y` are vectors and `S` is string specifying colour, marker symbol or line style.

Symbol	Color	Symbol	Linestyle
y	yellow	.	point
m	magenta	o	circle
c	cyan	x	x-mark
r	red	+	plus
g	green	*	asterisk
b	blue	-	solid
w	white	:	dotted
k	black	-.	dashdot
		--	dashed

- » `load scatterplot`
 - » `plot(data(:,5),data(:,3),'b.')`
 - » `plot(data(:,4),data(:,3),'k*')`
 - » `plot(data(:,4),data(:,5),'cx')`

6.3 2-D Scatterplot

- `plot(x,y,S,x,z,S)` plots two scatterplots in one.
 - » `plot(data(:,5),data(:,3),`
`'c*',data(:,5),data(:,4),'b+')`
- Adding titles and labels »
 - `plot(data(:,5),data(:,3),'b.')`
 - » `title('Scatterplot of Credit Card`
`Expenditure vs. Age')`
 - » `xlabel('Credit Card Expenditure')`
 - » `ylabel('Age')`

6.4 More 2-D Plotting Commands

- `loglog` produces a log-log scale plot.
- `semilogx` produces a semi-log scale plot in which the x-axis has a log scale.
- `semilogy` produces a semi-log scale plot in which the y-axis has a log scale.
- `polar` produces a polar coordinate plot.
» `t = 0:.01:2*pi;`
`polar(t, sin(2*t).*cos(2*t), '-r')`
- `bar(y)` draws the columns of the $M \times N$ matrix y as M groups of N vertical bars.
» `bar(rand(10,5), 'stacked'), colormap(cool)`
» `bar(rand(5), 'grouped')`
- `fplot(f, lim)` plots the function f between the x-axis limits specified by `lim = [xmin xmax]`.
» `fplot(@(x) [sin(x) cos(x)], 2*pi*[-1 1])`

6.5 3-D Line Plots

- `plot3` is a three-dimensional analogue of `plot` .
- `plot3(x,y,z,S)` , where x , y and z are three vectors of the same length, plots a line in 3-space through the points whose coordinates are the elements of x , y and z .
- `plot3(X,Y,Z)` , where X , Y and Z are three matrices of the same size, plots several lines obtained from the columns of X , Y and Z .
- Example:
 - » `t = 0:pi/50:10*pi;`
 - » `plot3(sin(t),cos(t),t)`

6.6 Surface Plots

- The first step in displaying a function of variables $z = f(x, y)$ is to generate a grid on the domain of this function.
- Matlab then evaluates and graphs this functions using this grid.
- `[X,Y]=meshgrid(x,y)` transforms the domain specified by vectors x and y into matrices X and Y (grid).
- `surf(X,Y,Z)` plots the colored parametric surface defined by X , Y and Z .
- `mesh(X,Y,Z)` plots the colored parametric mesh defined by X , Y and Z .

6.6 Surface Plots

- For example, to evaluate the function $x \exp^{-x^2-y^2}$ over the range $-2 < x < 2$, $-2 < y < 2$,
 - » `[X,Y] = meshgrid(-2:.2:2, -2:.2:2);`
 - » `Z = X .* exp(-X.^ 2 - Y.^ 2);`
 - » `surf(X,Y,Z)`
 - » `mesh(X,Y,Z)`
- `contour3(X,Y,Z)` produces a 3-D contours plot.
- `zlabel` creates a label for the z-axis.

6.7 Subplots

`subplot(m,n,p)` breaks the Figure window into an m -by- n matrix of subplots and selects the p -th subplot for the current plot.

```
» subplot(2,2,1), contour3(peaks)
```

```
» subplot(2,2,2), mesh(peaks)
```

```
» subplot(2,2,3), surf(peaks)
```

```
» subplot(2,2,4), surf1(peaks)
```

where `peaks` is a function of two variables, obtained by translating and scaling Gaussian distributions.

7. Scripts and Functions

7. Scripts and Functions

- In the command-driven mode Matlab (when we enter single-line commands) Matlab processes the commands immediately.
- Matlab can also execute a sequence of commands that are stored in .m files.
- There are two types of .m files: Scripts and Functions.
- Scripts .m files are used to execute a long list of commands.
- Functions allow us to create new functions in addition to the built-in-functions of Matlab that solve user-specific problems.

7.1 Scripts

- When a script is invoked Matlab simply execute the commands found in that .m file.
- Statements in a script file can operate on the data in the workspace or create new data to the workspace.
- To create a script, choose **File - New - m-file**.
- A text editor window is opened in which you can type the sequence of commands.
- To save a script, choose **File - Save**.
- You can execute the script by typing the filename in the command window or press the "Run" button (white square with green triangle).
- Make sure that Matlab can find your script file in the current directory or in the path.

7.1 Scripts

Example: Type the following sequence in the editor window.

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

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

```
S=A+B;
```

```
P=A*B;
```

```
DP=A.*B;
```

```
K=kron(A,B);
```

7.2 Functions

- A .m file that contains the word "function" in the first line of the file is a function file.
- You can pass arguments into the function.
- Variables defined and manipulated inside the file are local to the function and cannot be accessed globally in the workspace.
- The general syntax for defining a function is

```
function [ output1, output2, ...] = name(  
input1, input2, ...)  
% include here the text for the help  
statement
```
- The name of the function and the .m file (the name under which you save it) should be the same.
- The remainder of the functions is a sequence of commands producing the output variables.

7.2 Functions

Example: Define a function which calculates the mean of matrices.

```
function average = mymean(X)
% Calculate mean value of a matrix X
% average is a row vector containing the mean
of each column.
m=size(X,1);
average=sum(X)/m;
```

This new function can be used like any other Matlab function.

```
Type in the command window. » B=[10:10:100;
5:5:50]';
»mymean(B)
ans=
55.0000 27.5000
```

7.2 Exercises

- 1 Plot $y = \sin 3\pi x$ using `fplot` and `plot` for $0 \leq x \leq 1$.
- 2 Write a function that calculate the Standard Deviation of a matrix.
- 3 Write a function that will simulate N throws of 2 dice and returns the mean of the throws.
- 4 Build a $N \times 1$ vector y where each entry of y is $y_t = a + bX_{1t} + cX_{2t} + \epsilon_t$ where X_{it} is $Normal(i, 2)$ and ϵ_t is $Normal(0, 1)$. Write a function which calculates OLS estimates for β of the model y having as inputs N , a , b and c .

7.2 Exercises (Solutions)

```
1 » fplot(@(x) sin(3*pi*x), [0 1])  
  » N=100; h=1/N; x=0:h:1  
  » y=sin(3*pi*x);  
  » plot(x,y)
```

```
2 function a=mystd(data)  
  % calculates the standard deviation of a  
  matrix.  
  m=size(data,1);  
  a=sqrt(sum((data-ones(m,1)*mean(data)).  
  2)/(m-1));
```

```
3 function m = throws(n)  
  % Simulates n throws of a pair of dice and  
  returns the mean.  
  sim=floor(1+6*rand(n,2));  
  m=mean(sim);
```

4.

```
function Beta = OLSSimulation(N,a,b,c)
% returns OLS estimates from simulated data
regression.
X1=1+2*randn(N,1);
X2=2+2*randn(N,1);
Y=a+b*X1+c*X2+randn(N,1);
X=[ones(N,1) X1 X2];
Beta=inv(X'*X)*X'*Y;
```

8. Controlling the Flow

8.1 The FOR Loop

- Most often FOR loops are used when a set of statements is to be repeated a fixed, predetermined number of times n .

- The FOR loop takes the form

```
for variable = expression
statements;
end
```

where *expression* is a row vector of the length n , that is processed element-by-element (usually we use the row vector $1:1:n$). *statements* stands for a sequence of statements to the program.

- Example:

```
for i=1:1:10
x(i)=i;
end
```

The loop assigns values 1 to 10 to the first 10 elements of x .

8.1 The FOR Loop

- Example (nested FOR loop):

```
for i=1:1:10
    for j=1:1:10
        A(i,j)=1/(i+j-1);
    end
end
```

8.2 The WHILE Loop

- The WHILE loop allows a set of statements to be repeated an indefinite number of times, under the control of some or one logical conditions.

- The general form of a WHILE loop is

```
while condition
    statements;
end
```

- While the conditions are satisfied the statement will be executed.

- Example:

```
a=1;
while a<=10
    a=a+1;
end
a
```

8.3 The IF Statement

- The IF statement execute a set of statements if a condition is satisfied.

- Its general form is

```
if condition1
    %commands if condition1 is true
    statement1;
elseif condition2
    % commands if condition2 is true
    statement2;
else
    % commands if condition1 and condition2
    are false
    statment3;
end
```

8.3 The IF Statement

```
Example: x=-1+2*rand(100,1);  
for i=1:1:100  
    if x(i)>0  
        d(i)=1;  
    elseif x(i)<0  
        d(i)=-1;  
    else  
        d(i)=0;  
    end  
end  
end
```

Exercises

- 1 Using a FOR loop write a script that collects 10 random numbers from the uniform(0,1) in a column vector. Can you do the same thing using "vectorization"?
- 2 Replicate the first twenty numbers of the Fibonacci series: 1 2 3 5 8 13 21 34 55 89 144. Hint: for $i \geq 3$,
$$x_i = x_{i-1} + x_{i-2}.$$
- 3 Compute $a = \sum_{t=1}^T \frac{c}{(1+r)^t}$, where $T=20$, $C=10$, $r=0.05$. Also do it by vectorization.
- 4 Write a function which adds two matrices (1. Check if matrices are conformable, 2. If yes, add the corresponding elements).
- 5 What is the greatest value of n that can be used in the sum $1^2 + 2^2 + \dots + n^2$ to get a value of less than 100?
- 6 Generate a series of returns (i.e. `randn(100,1)`) using FOR and IF implement the following trading signal. Buy if the Return is bigger than the average of the past 5 years. (Build a series "Buy" that takes on value 0 if we don't buy

Exercises

1. for i=1:1:10

 R(1,:)=rand;

end

R=rand(10,1);

2.

S=[1 2];

for i=3:1:20

 S(i)=S(i-1)+S(i-2);

end

3.

for t=1:1:20

 a(t)=10/(1+0.05)^ t;

end

sum(a);

C=10*ones(1,20);

R=(1+0.05).^ [1:20];

sum(C./R);

4.

```
function c=add(a,b)
% c=add(a,b). This is the function which adds
% the matrices a and b. It duplicates
% the MATLAB function a+b.
[m,n]=size(a);
[k,l]=size(b);
if m~=k | n~=l,
    r='ERROR using add: matrices are not the
    same size';
end
c=zeros(m,n);
for i=1:m
    for j=1:n
        c(i,j)=a(i,j)+b(i,j);
    end
end
```

5.

```
S=1; n=1;  
while S+(n+1)2 < 100  
    n=n+1;  
    S=S+n2;  
end  
»[n,S]
```

6.

```
R=randn(100,1);
for t=1:size(R,1)
    if t>5
        if R(t)>mean(R(t-5:t))
            Buy(t)=1;
        else
            Buy(t)=0;
        end
    end
end
```