Introduction to Matlab programming

Patrick Winistörfer
Study Center Gerzensee

Fabio Canova
ICREA-UPF, CREI, AMeN and CEPR

January 22, 2008

Contents

1 The Command Window **3**
- 1.1 Interacting with the Matlab Command Window . . . 3
- 1.2 Command Line Editor . . . 3

2 Matrix Construction and Manipulation **4**
- 2.1 Building a Matrix . . . 4
- 2.2 The Help Facility . . . 6
- 2.3 Exporting and Importing Data . . . 6
  - 2.3.1 Exporting Data Files . . . 6
  - 2.3.2 Importing Data Files . . . 7

3 Basic Manipulations **8**
- 3.1 Using Partitions of Matrices . . . 8
- 3.2 Matrix Operations . . . 9
  - 3.2.1 Making Vectors from Matrices and Reverse . . . 9
  - 3.2.2 Transposition of a Matrix . . . 9
  - 3.2.3 Basic Operators . . . 9
  - 3.2.4 Array Operations . . . 10
  - 3.2.5 Relational Operations . . . 10
  - 3.2.6 Logical Operations . . . 10
- 3.3 More Built-in Functions . . . 11
  - 3.3.1 Display Text or Array . . . 11
  - 3.3.2 Sorting a Matrix . . . 11
  - 3.3.3 Sizes of Each Dimension of an Array . . . 12
  - 3.3.4 Sum of Elements of a Matrix . . . 12
  - 3.3.5 Smallest (Largest) Elements of an Array . . . 12
  - 3.3.6 Inverse of a Matrix . . . 13
  - 3.3.7 Eigenvectors and Eigenvalues of a Matrix . . . 13
- 3.4 Special matrices . . . 13
  - 3.4.1 Elementary Matrices . . . 14
  - 3.4.2 Special Matrices . . . 14
  - 3.4.3 Manipulating Matrices . . . 14

4 Mathematical Functions **15**
- 4.1 Trigonometric Functions . . . 15
- 4.2 Exponential Functions . . . 16
- 4.3 Complex Functions . . . 16
- 4.4 Numeric Functions . . . 16

5 Data Analysis **17**
- 5.1 Using Column-Oriented Analysis . . . 17
- 5.2 Missing values . . . 17
- 5.3 Polynomials . . . 18
- 5.4 Functions functions . . . 20

6 Graphics **23**
6.1 Histogramm 23
6.2 Plotting Series versus their Index 23
6.3 Scatterplot (2-D) 23
6.4 Adding Titles and Labels 24
6.5 Plotting Commands 25
6.6 3-D Graphics 25

7 Scripts and Functions **29**
7.1 Script m-files 29
7.2 Function m-files 29

8 Controlling the Flow **31**
8.1 The FOR Loop 31
8.2 The WHILE Loop 32
8.3 The IF Statement 32
8.4 How to Break a FOR or WHILE Loop 33

9 Random Numbers **33**

10 Some Examples **33**
10.1 loaddata.m 33
10.2 example_var.m 34
10.3 example_gmm.m 40
10.4 draw_rn.m 42

1 The Command Window

When we invoke Matlab, the command window is created and made the active window. The command window is the interface through which we communicate with the Matlab interpreter. The interpreter displays its prompt ($\gg$) indicating that it is ready to accept instructions. We can now type in Matlab commands. In what follows, `typeset` letters denote Matlab code.

1.1 Interacting with the Matlab Command Window

After typing a command you execute it by pressing the *enter* key. Issuing the command `clear x` clears the value of variable x from the memory and `clear` clears all values that have been created during a Matlab session.
In general, if you type

```
>> A
```

Matlab does the following:

1. It verifies if `A` is a *variable* in the Matlab workspace.
2. If not, it verifies if `A` is a *built-in-function*.
3. If not, it verifies if a *M-file* named *A.m* exists in the current directory.
4. If not it verifies if *A.m* exists anywhere on the Matlab search path, by searching the path in the order in which it is specified.

Finally, if Matlab can not find *A.m* anywhere on the Matlab search path, an error message will appear in the command window.
We can display and change the current Matlab search path: Select *Set Path* from the *File* menu.
When Matlab displays numerical results it displays real numbers with *four digits* to the right of the decimal point. If you want to have more than four digits displayed issue the command `format long`. To go back to displaying four digits after the decimal point (the default setting), type `format short`.
Note also that a ';' (semicolon) at the end of a line tells Matlab not to display the results from the command it executes. Text following '%' is interpreted as a comment (plain text) and is not executed.

1.2 Command Line Editor

To simplify the process of entering commands to the Matlab interpreter, we can use the arrow keys on the keypad to edit mistyped commands and to recall previous command lines. For example, suppose we execute the command line:

```
>> log(sqt(atan2(3,4)))
```

and misspell 'sqrt'. Matlab responds with the error message:

```
??? Undefined function or variable 'sqt'
```

Instead of retyping the entire line, simply press the Up-arrow key. The misspelled command is redisplayed, and we can move the cursor over the appropriate place to insert the missing 'r' by using the Left-arrow key or the mouse. After pressing Return, the command returns the appropriate answer.

```
>> log(sqrt(atan2(3,4)))
ans =
   -0.2204
```

2 Matrix Construction and Manipulation

In Matlab each variable is a matrix. Since a matrix contains m rows and n columns, it is said to be of dimension m-by-n. An m-by-*1* or *1*-by-n matrix is called a *vector*. A *scalar*, finally, is a *1*-by-*1* matrix.
You can give a matrix whatever name you want. Numbers may be included in the names. Matlab is sensitive with respect to the upper or lower case. If a variable has previously been assigned a value, the new value *overwrites* the old one.

2.1 Building a Matrix

There are several ways to build a matrix:
One way to do it is to declare a matrix as if you wrote it by hand

```
>> A=[1 2 3
4 5 6]
```

This results in the output

```
A =
    1   2   3
    4   5   6
```

Another way is to separate rows with ';'

```
>> A=[1 2 3;4 5 6]
```

A final way is to do it element by element

```
>> A(1,1)=1;
>> A(1,2)=2;
>> A(1,3)=3;
>> A(2,1)=4;
>> A(2,2)=5;
>> A(2,3)=6;
```

There are some special matrices that are extremely useful:
The zero (or Null) matrix: `zero(m,n)` creates a m-by-n matrix of zeros. Thus,

```
>> B=zeros(3,2)
```

results in

```
B =
    0   0
    0   0
    0   0
```

The ones matrix: `zero(m,n)` creates a m-by-n matrix of zeros. Thus,

```
>> C=ones(2,3)
```

results in

```
C =
    1   1   1
    1   1   1
```

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

```
>> D=eye(3)
```

results in

```
D =
    1   0   0
    0   1   0
    0   0   1
```

You can generate a m-by-n matrix of random elements using the command **`rand(m,n)`**, for *uniformly* distributed elements, **`randn(m,n)`**, for *normally* distributed elements. That is, **`rand`** will draw numbers in $[0;1]$ while **`randn`** will draw numbers from a $N(0,1)$ distribution.
The colon ':' is an important symbol in Matlab. The statement

```
>> x=1:5
```

generates a row vector containing the numbers from 1 to 5 with unit increments. We can also use increments different from one:

```
>> y=0:pi/4:pi
```

results in

```
y =
    0    0.7854    1.5708    2.3562    3.1416
```

2.2 The Help Facility

The help facility provides online information about Matlab functions, commands and symbols.
The command

```
>> help
```

with no arguments displays a list of directories that contains Matlab related files.
Typing `help topic` displays help about that topic if it exists. The command `help elmat` provides information on '*Elementary matrices and matrix manipulation*', `help elfun` for information on '*Elementary math functions*', and `help matfun` for information on '*Matrix functions - numerical linear algebra*'.

2.3 Exporting and Importing Data

First, we have to indicate the correct *current directory* in the *command toolbar*, choose "d:\".

2.3.1 Exporting Data Files

In the following exercise we create four different data files:

1. Clear the workspace by issuing the command `clear`.
2. Define the following two matrices:

   ```
   >> x=[0:0.5:50]'
   >> y=randn(6,12)
   ```

3. Create the Matlab data file *result1.mat*:

   ```
   >> save result1
   ```

   The two matrices x and y are saved in the file *result1.mat*.
4. Create another Matlab data file *result2.mat*:

   ```
   >> save result2 y
   ```

   Only the matrix y is saved in the file *result2.mat*.
5. Create the data file *result3.txt*:

   ```
   >> save result3.txt y -ascii
   ```

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

6. Create the data file *result4.dat*:

   ```
   >> save result4.dat x
   ```

   Only the matrix x is saved in the file *result4.dat*.

2.3.2 Importing Data Files

In the next exercise we import the four results we previously saved:

1. Clear the workspace by issuing the command `clear`.
2. Import the data file *result1.mat*:

   ```
   >> load result1
   ```

   The two matrices x and y are imported in the workspace.
3.
4. Clear again the workspace by issuing the command `clear`.
5. Import the data file *result2.mat*:

   ```
   >> load result2
   ```

   The matrix y is imported in the workspace.
6. Clear again the workspace by issuing the command `clear`.
7. Import the data file *result3.txt*:

   ```
   >> load result3.txt -ascii
   ```

   A matrix called *result3* is imported in the workspace. This matrix is exactly the y matrix we created previously.
8. Clear again the workspace by issuing the command `clear`.
9. Import the data file *result4.dat*:

   ```
   >> a=importdata('result4.dat')
   ```

   A matrix called a is imported in the workspace. This matrix is exactly the x matrix we created previously.

3 Basic Manipulations

3.1 Using Partitions of Matrices

You can build matrices out of several submatrices. Suppose you have submatrices A to D.

```
>> A=[1 2; 3 4];
>> B=[5 6 7; 8 9 10];
>> C=[3 4; 5 6];
>> D=[1 2 3; 4 5 6];
```

In order to build E, which is given by $E = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$, you type:

```
>> E=[A B; C D]
E =
    1   2   5   6   7
    3   4   8   9  10
    3   4   1   2   3
    5   6   4   5   6
```

One of the most basic operation is to extract some elements of a matrix (called a partition of matrices). Consider matrix E we defined previously. Let's now isolate the central matrix $F = \begin{pmatrix} 2 & 5 \\ 4 & 8 \\ 4 & 1 \end{pmatrix}$. In order to do this we type

```
>> F=E(1:3,2:3)
F =
    2   5
    4   8
    4   1
```

Suppose we just want to select columns 1 and 3, but take all the lines.

```
>> F=E(:,[1 3])
F =
    1   5
    3   8
    3   1
    5   4
```

where the colon (:) means *select all*.

3.2 Matrix Operations

3.2.1 Making Vectors from Matrices and Reverse

Suppose we want to obtain the vectorialisation of a matrix, that is we want to obtain vector B from matrix A.

```
A =
    1   2
    3   4
    5   6
>> B=A(:)
B =
    1
    2
    3
    4
    5
    6
```

Suppose we have vector B. We want to obtain matrix A from B.

```
>> A=reshape(B,3,2)
A =
    1   2
    3   4
    5   6
```

3.2.2 Transposition of a Matrix

```
A =
    1   2
    3   4
    5   6
>> A'
ans =
    1   3   5
    2   4   6
```

Note: The *ans* variable is created automatically when no output argument is specified. It can be used in subsequent operations.

3.2.3 Basic Operators

$+$	Addition	
$-$	Subtraction	
$*$	Multiplication	
$/$	Right division	$B/A = BA^{-1}$
$\backslash$	Left division	$A\backslash B = A^{-1}B$
$\hat{}$	Exponentiation	$A\hat{}(-1) = A^{-1}$

3.2.4 Array Operations

To indicate an *array* (element-by-element) operation, precede a standard operator with a period (dot). Matlab array operations include multiplication(.*), division (./) and exponentiation (.^).
Thus, the "dot product" of x and y is

```
>> x=[1 2 3];
>> y=[4 5 6];
>> x.*y
ans =
    4   10   18
```

3.2.5 Relational Operations

$<$ $\quad$ $>$	Less (greater) than
$<=$ $\quad$ $>=$	Less (greater) than or equal to
$==$	Equal to
$\sim=$	Not equal to

Matlab compares the pairs of corresponding elements; the result is a matrix of ones and zeros, with one representing 'true' and zero representing 'false'. E.g. let's consider the following matrix G. the command `G<=2` finds elements of G that are less or equal 2. The resulting matrix is T.

```
>> G=[1 2; 3 4];
>> T=G<=2;
T =
    1   1
    0   0
```

3.2.6 Logical Operations

&	AND
|	OR
$\sim$	NOT

```
>> C=A&B;
```

C is a matrix whose elements are ones where A *and* B have nonzero elements, and zeros where either has a zero element. A and B must have the same dimension unless one is a scalar. A scalar can operate with everything.

```
>> C=A|B;
```

C is a matrix whose elements are ones where A *or* B have a nonzero element, and zeros where both have a zero element.

```
>> B=~A;
```

B is a matrix whose elements are ones where A has a zero element, and zeros where A has a nonzero element.

3.3 More Built-in Functions

The type of commands used to build special matrices are called *built-in functions*. There are a large number of built-in functions in Matlab. Apart from those mentioned above, the following are particularly useful:

3.3.1 Display Text or Array

`disp(A)` displays an array, without printing the array name. If A contains a text string, the string is displayed.

```
>> disp('     Corn     Oats     Hay')
     Corn     Oats     Hay
```

3.3.2 Sorting a Matrix

The function `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.

```
>> A=[1 2; 3 5; 4 3]
A =
    1   2
    3   5
    4   3
>> sort(A)
ans =
    1   2
    3   3
    4   5
```

3.3.3 Sizes of Each Dimension of an Array

The function `size(A)` returns the sizes of each dimension of matrix A in a vector. `[m,n]=size(A)` return the size of matrix A in variables m and n (recall: in Matlab arrays are defined as m-by-n matrices). `length(A)` returns the size of the longest dimension of A.

```
>> A=[1 2; 3 5; 4 3];
>> [m,n]=size(A)
m =
    3
n =
    2
>> length(A)
ans =
    3
```

3.3.4 Sum of Elements of a Matrix

If A is a vector, `sum(A)` returns the sum of the elements. If A is a matrix, `sum(A)` treats the columns of A as vectors, returning a row vector of the sums of each column.

```
>> A=[1 2; 3 5; 4 3];
>> B=sum(A)
B =
    8   10
```

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

```
>> B=cumsum(A)
B =
    1   2
    4   7
    8  10
```

3.3.5 Smallest (Largest) Elements of an Array

If A is a matrix, `min(A)` treats the columns of A as vectors, returning a row vector containing the minimum element from each column. If A is vector, `min(A)` returns the smallest element in A. `max(A)` returns the maximum elements.

```
>> A=[1 2; 3 5; 4 3];
>> B=min(A)
ans =
    1   2
```

3.3.6 Inverse of a Matrix

The function `inv(A)` returns the inverse of the square matrix A.

```
>> A=[4 2; 1 3];
>> B=inv(A)
B =
    0.3000   -0.2000
   -0.1000    0.4000
```

A frequent misuse of `inv` arises when solving the system of linear equations $Ax = b$. One way to solve this is with `x=inv(A)*b`. A better way, from both an execution time and numerical accuracy standpoint, is to use the matrix division operator `x=A\b`.

3.3.7 Eigenvectors and Eigenvalues of a Matrix

The n-by-n (quadratic) matrix A can often be decomposed into the form $A = PDP^{-1}$, where D is the matrix of eigenvalues and P is the matrix of eigenvectors. Consider

```
>> A =
    0.9500   0.0500
    0.2500   0.7000
>> [P,D]=eig(A)
>> P =
    0.7604  -0.1684
    0.6495   0.9857
>> D =
    0.9927        0
         0   0.6573
```

If you are just interested in the eigenvalues:

```
>> eig(A)
ans =
    0.9927
    0.6573
```

3.4 Special matrices

A collection of functions generates special matrices that arise in linear algebra and signal processing is next.

3.4.1 Elementary Matrices

`zeros`	Matrix of zeros
`ones`	Matrix of ones
`eye`	Identity matrix
`rand`	Matrix of uniformly distributed random numbers
`randn`	Matrix of normally distributed ($N(0,1)$) random numbers
`meshgrid`	X and Y arrays for 3-D plots

3.4.2 Special Matrices

`compan`	Companion matrix
`hadamard`	Hadamard matrix
`hankel`	Hankel matrix
`kron`	Kronecker tensor product
`toeplitz`	Toeplitz matrix

3.4.3 Manipulating Matrices

Several functions rotate, flip, reshape, or extract portions of a matrix.

`diag`	Create or extract diagonals
`fliplr`	Flip matrix in the left/right direction
`flipud`	Flip matrix in the up/down direction
`reshape`	Change size
`tril`	Extract lower triangular part
`triu`	Extract upper triangular part
`size`	Size of a matrix (two element vector with row and column dimension)
`length`	Size of a vector

For example to reshape a 3-by-4 matrix A into a 2-by-6 matrix B use the command:

```
>> B = reshape(A,2,6)
```

4 Mathematical Functions

A set of elementary functions are applied on an element by element basis to arrays. For example:

```
>> A = [1 2 3; 4 5 6];
>> B = fix(pi*A)
```

(`fix(X)` rounds the elements of X to the nearest integers towards zero; `pi` stands for π) produces:

```
B =
     3     6     9
    12    15    18
```

and

```
>> C = cos(pi*B)
```

gives

```
C =
    -1     1    -1
     1    -1     1
```

The following subsections list Matlab's principal mathematical functions.

4.1 Trigonometric Functions

`sin`	Sine
`sinh`	Hyperbolic sine
`cos`	Cosine
`cosh`	Hyperbolic cosine
`tan`	Tangent
`tanh`	Hyperbolic tangent
`sec`	Secant
`sech`	Hyperbolic secant

4.2 Exponential Functions

`exp`	Exponential (e to the power)
`log`	Natural logarithm
`log10`	Common logarithm
`sqrt`	Square root

4.3 Complex Functions

`abs`	Absolute value
`angle`	Phase angle
`conj`	Complex conjugate
`imag`	Complex imaginary part
`real`	Complex real part

4.4 Numeric Functions

`fix`	Round towards zero
`floor`	Round towards minus infinity
`ceil`	Round towards plus infinity
`round`	Round towards nearest integer
`rem`	Remainder after division
`sign`	Signum function

5 Data Analysis

This section presents an introduction to data analysis using Matlab and describes some elementary statistical tools.

5.1 Using Column-Oriented Analysis

Matrices are, of course, used to hold all data, but this leaves a choice of orientation for multivariate data. By convention, the different variables in a set of data are put in columns, allowing observations to vary down through the rows. Therefore, a data set of 50 samples of 13 variables is stored in a matrix size 50-by-13.
We can generate a sample data set (let's call it B) of 13 variables with 50 observations each by using the command:

```
>> B = rand(50,13)
```

and analyze the main properties of the generated variables using the data analysis capabilities provided by the following functions:

`max`	Largest component
`min`	Smallest component
`mean`	Average or mean value
`median`	Median value
`std`	Standard deviation
`sort`	Sort in ascending order
`sum`	Sum of elements
`prod`	Product of elements
`cumsum`	Cumulative sum of elements
`cumprod`	Cumulative product of elements
`trapz`	Numerical integration using trapezoidal method

For vector arguments, it does not matter whether the vectors are oriented in a row or column direction. For array arguments, the functions described above operate in a column-oriented fashion. This means that if we apply the operator `max` on an array, the result will be a row vector containing the maximum values over each column.

5.2 Missing values

The correct handling of NAs (Not Available) is a difficult problem and often varies in different situations. Matlab, however, is uniform and rigorous in its treatment of NAs; they propagate naturally to the final result in any calculation. In Matlab NAs are treated as NaNs (Not a Number). We should remove NaNs from the data before performing statistical computations. The NaNs in a vector x are located at:

```
>> i = find(isnan(x));
```

so

```
>> x = x(find(~isnan(x));
```

returns the data in x with the NaNs removed. Two other ways of doing the same thing are:

```
>> x = x(~isnan(x));
```

or

```
>> x(isnan(x)) = [];
```

If instead of a vector, the data are in the columns of a matrix, and we are removing any rows of the matrix with NaNs, use:

```
>> X(any(isnan(X)'),:) = [];
```

5.3 Polynomials

Matlab represents polynomials as row vectors containing the coefficients ordered by descending powers. For example, the coefficients of the characteristic equation of the matrix:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

can be computed with the command:

```
>> p = poly(A)
```

which returns

```
p =
     1    -6   -72   -27
```

This is the Matlab representation of the polynomial:

$$x^3 - 6x^2 - 72x - 27$$

The roots of this equation can be obtained by:

```
>> r = roots(p);
```

These roots are, of course, the same as the eigenvalues of matrix A. We can reassemble them back into the original polynomial with `poly`:

```
>> p2 = poly(r);
```

Now consider the polynomials $a(x) = x^2 + 2x + 3$ and $b(x) = 4x^2 + 5x + 6$. The product of the polynomials is the convolution (`conv`) of the coefficients:

```
>> a = [1 2 3];
>> b = [4 5 6];
>> c = conv(a,b);
```

We can also use deconvolution (`deconv`) to divide $a(x)$ back out:

```
>> [q,r] = deconv(c,a)
```

gives

```
q =
      4      5      6

r =
      0      0      0      0      0
```

Other polynomial functions:

`roots`	Find polynomial roots
`poly`	Construct polynomial with specified roots
`polyval`	Evaluate polynomial
`polyvalm`	Evaluate polynomial with matrix argument
`residue`	Partial-fraction expansion (residues)
`polyfit`	Fit polynomial to data
`polyder`	Differentiate polynomial
`conv`	Multiply polynomials
`deconv`	Divide polynomials

5.4 Functions functions

There is a class of functions in Matlab that doesn't work with numerical matrices, but with mathematical functions. These function functions include:

(a) Numerical integration.

(b) Nonlinear equations and optimization.

(c) Differential equation solution.

Functions
Matlab represents mathematical functions by function m-files. For example, the function:

$$f(x) = \frac{1}{(x-0.3)^2 + 0.01} + \frac{1}{(x-0.9)^2 + 0.04} - 6$$

is made available to Matlab by creating a m-file called `humps.m`:

```
function y = humps(x)
y = 1 ./ ((x-.3).^2 + .01) + 1 ./((x-.9).^2 + .04) - 6;
```

A graph of the function can be obtained by typing:

```
>> x = -1:0.01:2;
>> plot(x,humps(x))
```

(a) **Numerical Integration**

The area beneath a function, $f(x)$ can be determined by numerically integrating $f(x)$, a process referred to as quadrature. To integrate the function defined by `humps.m` from 0 to 1 type:

```
>> q = quad('humps',0,1)
```

this command returns

```
q =
   29.8583
```

The two Matlab functions for quadrature are

`quad`	Adaptive Simpson's rule
`quad8`	Adaptive Newton Cotes 8 panel rule

Notice that the first argument to quad is a quoted string containing the name of a function. This is why quad is called a function function - it is a function that operates on other functions.

(b) **Nonlinear Equations and Optimization Functions**

The function functions for nonlinear equations and optimization include: (**check, if your matlab version is 6.5 or above the names have changed!**)

`fmin`	Minimum of a function of one variable
`fmins`	Minimum of a multivariate function (unconstrained nonlinear optimization
`fzero`	Zero of a function of one variable

Using again the function defined in `humps.m`, the location of the minimum in the region from 0.5 to 1 is computed with `fmin`:

```
>> xm = fmin('humps',0.5,1)
```

It returns:

```
xm =
    0.6370
```

Its value at the minimum is:

```
>> y = humps(xm)
y =
   11.2528
```

Looking at the graph, it is apparent that humps has two zeros. The location of the zero near $x = 0$ is:

```
>> xz1 = fzero('humps',0)
xz1 =
   -0.1316
```

The Optimization Toolbox contains several additional functions:

`attgoal`	Multi-objective goal attainment
`constr`	Constrained minimization
`fminu`	Unconstrained minimization
`fsolve`	Nonlinear equation solution
`leastsq`	Nonlinear least squares
`minimax`	Minimax solution
`seminf`	Semi-infinite minimization

6 Graphics

Matlab provides a variety of functions for displaying data as planar plot or 3-D graphs and for annotating these graphs. This section describes some of these functions and provides examples of some typical applications.
The first important instruction is the `clf` instruction that clears the graphic screen.

6.1 Histogramm

The instruction `hist(x)` draws a 10-bin histogram for the data in vector x. The command `hist(x,c)`, where c is a vector, draws a histogram using the bins specified in c. Here is an example:

```
>> c=-2.9:0.2:2.9;
>> x=randn(5000,1);
>> hist(x,c);
```

6.2 Plotting Series versus their Index

2-D graphs essentially rely on the instruction `plot(x)`. It plots the data in vector x versus its index. The command `plot([x y])` plots the two column vectors of the matrix $[x \ \ y]$ versus their index (within the same graph). Consider the following example:

```
>> x=rand(100,1);
>> y=rand(100,1);
>> clf;
>> plot([x y]);
```

6.3 Scatterplot (2-D)

The general form of the `plot` instruction is

```
plot(x,y,S)
```

where x and y are vectors or matrices and S is a one, two or three charter string specifying color, marker symbol, or line style. The command `plot(x,y,S)` plots y against x, if x and y are vectors. If x and y are matrices, the columns of y are plotted versus the columns of x in the same graph. If only x or y is a matrix, the vector is plotted versus the rows or columns of the matrix. Note that `plot(x,[a b],S)` and `plot(x,a,S,x,b,S)`, where a and b are vectors, are alternative ways to create exactly the same scatterplot.

The S string is optional and is made of the following (and more) characters:

Symbol	Colour	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

Thus,

```
>> plot(x,y,'b*');
```

plots a blue asterisk at each point of the data set.
Here is an example:

```
>> x=pi*(-1:.01:1);
>> y=sin(x);
>> clf;
>> plot(x,y,'r-');
```

6.4 Adding Titles and Labels

You can add titles and labels to your plot, using the following instructions:

```
>> title('Graph');
>> xlabel('x-axis');
>> ylabel('y-axis');
```

Consider a list of further graphical commands:

```
>> help graph2d
```

6.5 Plotting Commands

Elementary X-Y graphs commands

`plot`	Linear plot
`loglog`	Log-log scale plot
`semilogx`	Semi-log scale plot (log scale for the x-axis)
`semilogy`	Semi-log scale plot

Specialized X-Y graphs

`polar`	Polar coordinate plot
`bar`	Bar graph
`stem`	Discrete sequence or "stem" plot
`stairs`	Stairstep plot
`hist`	Histogram plot
`fplot`	Plot function
`comet`	Comet-like trajectory
`title`	Graph title
`xlabel`	X-axis label
`ylabel`	Y-axis label
`text`	Text annotation
`gtext`	Mouse placement of text
`grid`	Grid lines

6.6 3-D Graphics

Matlab provides a variety of functions to display 3-D data. Some plot lines in three dimensions, while others draw surfaces and wire frames using pseudocolors to represent a fourth dimension. The following list summarizes these functions.

`plot3`	Plot lines and points in 3-D space
`fill3`	Draw filled 3-D polygons in 3-D space
`contour`	Contour plot
`countour3`	3-D Contour plot
`mesh`	3-D mesh surface
`meshc`	Combination mesh/contour plot
`meshz`	3-D Mesh with zero plane
`surf`	3-D shaded surface
`surfc`	Combination of surf/contour plot
`surfl`	3-D shaded surface with lighting

In addition to the graph annotation functions described in the section on 2-D plotting, Matlab supports the following functions for annotating the graphs discussed in this section:

`zlabel`	Creates a label for the z-axis
`clabel`	Adds contour labels to a contour plot

Matlab allows us to specify the point from which we view the plot. In general, views are defined by a 4-by-4 transformation matrix that in Matlab uses to transform the 3-D plot to the 2-D screen. However, two functions allow us to specify the view point in a simplified manner:

`view`	3-D graph viewpoint specification
`viewmtx`	View transformation matrices

Line Plots

The three dimensional analog of the plot function is `plot3`. If x, y and z are three vectors of the same length,

```
>> plot3(x,y,z)
```

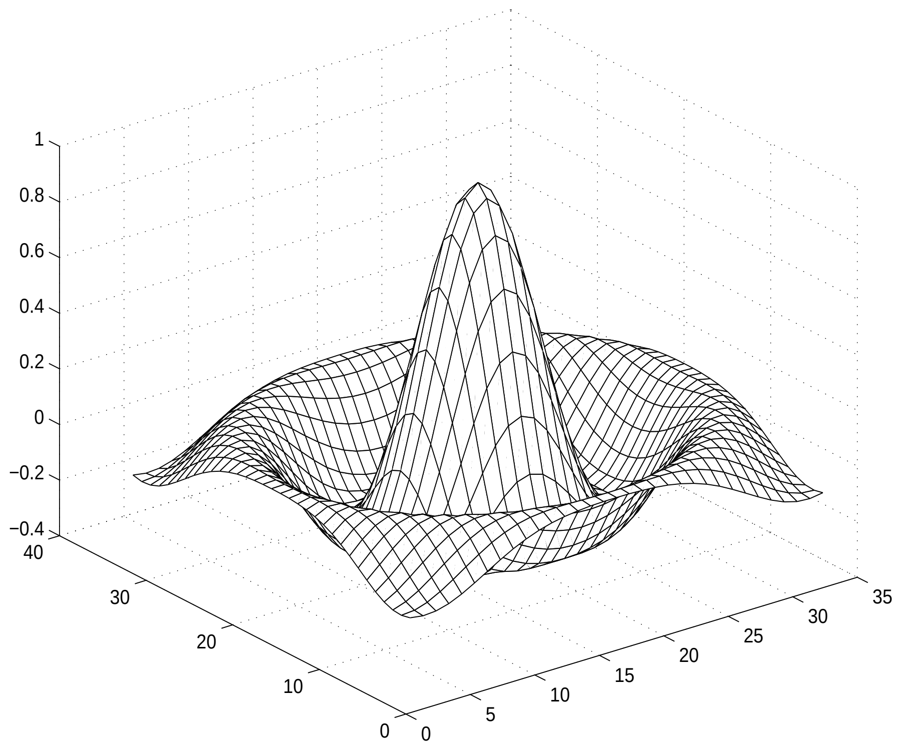
generates a line in 3-D space through the points whose coordinates are the elements of x, y and z.

Meshgrid

Matlab defines a mesh surface by the z coordinates of points above a rectangular grid in the x-y plane. It forms a plot by joining adjacent points with straight lines. The first step in displaying a function of two variables, $(z = f(x, y))$, is to generate X and Y matrices consisting of repeated rows and columns, respectively, over the domain of the function. Then use these matrices to evaluate and graph the function. The meshgrid function transforms the domain specified by two vectors , x and y, into row matrices X and Y. We then use these matrices to evaluate functions of two variables. The rows of X are copies of the vector x and the columns of Y are copies of the vector y. Example:

```
>> x = -8:0.5:8;
>> y = x;
>> [X,Y] = meshgrid(x,y);
>> R = sqrt(X.^2 + Y.^2) + eps;
>> Z = sin(R)./R;
>> mesh(Z)
```

Note: Adding `eps` (floating point relative accuracy) prevents the divide by zero which produces NaNs in the data.



Contour Plots

Matlab supports both 2-D and 3-D functions for generating contour plots. The `contour` and `contour3` functions generate plots composed of lines of constant data values obtained from a matrix input argument. We can optionally specify the number of contour lines, axis scaling, and the data value at which to draw contour lines. For example, the following statement creates a contour plot having 20 contour lines and using the peaks m-file to generate the input data.

```
>> contour(peaks,20)
```

Contour cycles through the line colors listed in the paragraph about the plot function.
To create a 3-D contour plot of the same data, use `contour3`:

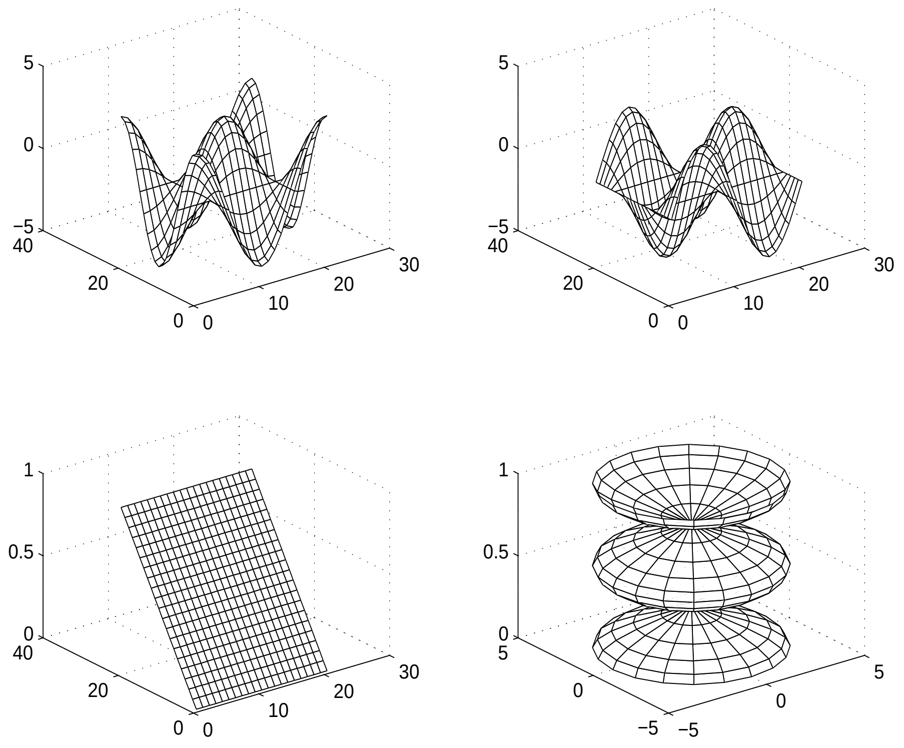
```
>> contour3(peaks,20)
```

Subplots

We can display multiple plots in the same window or print them on the same piece of paper with the `subplot` function. `subplot(m,n,p)` breaks the figure window into an *m*-by-*n* matrix of small subplots and select the *p*-th subplot for the current plot. The plots are numbered along first the top row of the figure window, then the second row, etc. For example:

```
>> t = 0:pi/10:2*pi;
>> [X,Y,Z] = cylinder(4*cos(t));
>> subplot(2,2,1)
>> mesh(X)
>> subplot(2,2,2)
>> mesh(Y)
>> subplot(2,2,3)
>> mesh(Z)
>> subplot(2,2,4)
>> mesh(X,Y,Z)}
```

gives as a result:



7 Scripts and Functions

Matlab is usually used in a command-driven mode, when we enter single-line commands, Matlab processes them immediately and displays the results. Matlab can also execute sequences of commands that are stored in files. Such files are called *m-files* because they have a file type of `.m` as last part of the filename. Two types of m-files can be used: *Script* and *Function*. Script m-files automate long sequences of commands. Function m-files provide extensibility to Matlab. They allow us to add new functions to the existing functions. Much of the power of Matlab derives from this ability to create new functions that solve user-specific problems.

7.1 Script m-files

When a script is invoked, Matlab simply executes the commands found in the file. The statements in a script file can operate on existing data in the workspace, or they can create new data on which to operate. Although scripts do not return output arguments, any variables that they create remain in the workspace, to be used in subsequent computations.
Scripts are useful for performing analyses, solving problems, or designing long sequences of commands that become cumbersome to do interactively.
To create a script, choose **New** from the **File** menu and select **m-file** (alternatively you just type `edit` in the Matlab prompt and press *Enter*). This procedure brings up a text editor window, the *Matlab editor/debugger*. Let's type the following sequence of statements in the editor window:

```
A=[1 2];
B=[3 4];
C=A+B
```

This file can be saved as the m-file *example.m* on drive xx by choosing **Save** from the **File** menu. Matlab executes the commands in *example.m* when you simply type `example` in the Matlab command window (provided there is a file *example.m* in your working directory or path[1]).

```
>> example
C =
    4   6
```

7.2 Function m-files

An m-file that contains the word "function" at the beginning of the first line is a function file. A function m-file differs from a script file in that arguments may be passed, and variables defined and manipulated inside the file are local to the function and do not operate globally on the workspace. Functions files

[1]Do modify the default adjustments over **File/Set Path** or choose the *working directory* directly in the command toolbar.

are useful for extending Matlab, that is, creating new Matlab functions using the Matlab language itself.
We add new functions to Matlab's vocabulary by expressing them in term of existing commands and functions. The general syntax for defining a function is

```
function[output1,..]=<name of function>(input1,..);
% include here the text of your online help statements;
```

The first line of a function m-file defines the m-file as a function and specifies its name. The name of the m-file and the one of the function should be the same. It also defines its *input* and *output variables*.
Next, there is a sequence of comment lines with the text displayed in response to the `help` command:

```
>> help <name of function>
```

Finally, the remainder of the m.file contains Matlab commands that create the output variables.
Let's consider an example:

```
function y=mean(x)
% Average or mean value
% For vectors, mean(x) returns the mean value.
% For matrices, mean(x) is a row vector containing the mean of each column.
[m,n]=size(x);
if m==1
m=n;
end
y=sum(x)/m;
```

The existence of this file defines a new function called "mean". The new function is used like any other Matlab function. Let's test the new function:

```
>> b=[1:99; 101:199]';
>> mean(b)
ans =
    50   150
```

In the list below there are some details about the `mean.m` file:

- The first line declares the function name, the input arguments, and the output arguments. Without this line, the file is a script file, instead of a function file.
- The % symbol indicates that the rest of the line is a comment and should be ignored.
- The first lines document the m-file and display when we type:

  ```
  >> help mean
  ```

- The variables m, n and y are local to `mean` and do not exist in the workspace after the command `mean` has terminated its task. Or, if previously existing, they remain unchanged.

8 Controlling the Flow

The structure of the sequences of instructions used so far was rather straightforward; commands were executed one after the other, running from top to bottom. Like any computer programming language Matlab offers, however, features that allow you to *control the flow of command execution.*

8.1 The FOR Loop

The most common use of a FOR loop arises when a set of statements is to be repeated a fixed, predetermined number of times n. The general form of a FOR loop is:

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

The variable `expression` is thereby a row vector of the length n, that is processed element-by-element (in many cases it is the row vector `(1:n)`). `Statements` stands for a sequence of statements to the program.
Two simple examples:

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

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

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

The semicolon after the inner statement suppresses repeated printing, while `disp(A)` displays the final result.

8.2 The WHILE Loop

Matlab has its own version of the WHILE loop, which allows a statement, or group of statements, to be repeated an indefinite number of times, under the control of a logical condition. The general form of a WHILE loop is the following:

```
while condition;
    statements;
end;
```

While the condition is satisfied, the statements will be executed.
Consider the following example:

```
EPS=1;
while (1+EPS)>1;
    EPS=EPS/2;
end;
EPS=EPS*2
```

This example shows one way of computing the special Matlab value eps, which is the smallest number that can be added to 1 such that the result is greater than 1 using finite precision (see: `help eps`, we use uppercase `EPS` so that the Matlab value `eps` is not overwritten). In this example, `EPS` starts at 1. As long as `(1+EPS)>1` is true (nonzero), the commands inside the WHILE loop are evaluated. Since `EPS` is continually divided in two, `EPS` eventually gets so small that adding `EPS` to 1 is no longer greater than 1 (Recall that this happens because a computer uses a fixed number of digits to represent numbers. Matlab uses 16 digits, so you would expect `EPS` to be near 10^{-16}.) At this point, `(1+EPS)>1` is false (zero) and the WHILE loop terminates. Finally, `EPS` is multiplied by 2 because the last division by 2 made it too small by the factor of two.

8.3 The IF Statement

A third possibility of controlling the flow in Matlab is called the IF statement. The IF statement executes a set of instructions if a condition is satisfied. The general form is:

```
if condition 1;
    % commands if condition 1 is true
    statements 1;
elseif condition 2;
    % commands if condition 2 is true
    statements 2;
else;
    % commands if condition 1 and 2 are both false
    statements 3;
end;
```

The last set of commands is executed if both conditions 1 and 2 are false (i.e. not true). When you have just two conditions you skip the `elseif` condition and immediately go to `else`. Consider the following example:
Assume a demand function of the form

$$D(P) = \begin{cases} 0 & , \text{ if } P \leq 2 \\ 1 - 0.5P & , \text{ if } 2 < P \leq 3 \\ 2P^{-2} & , \text{ otherwise} \end{cases}$$

The code will be:

```
P=input('Enter a price : ');  % displays the message
                              % on the screen and waits
                              % for an answer
if P<=2;
    D=0;                      % statement 1
elseif (P>2)&(P<=3);
    D=1-0.5*P;                % statement 2
else;                         % otherwise
    D=2*P^(-2);               % statement 3
end;
disp(D);
```

8.4 How to Break a FOR or WHILE Loop

The command `break` terminates the execution of a FOR or WHILE loop.

9 Random Numbers

Matlab has a number of functions which allow to draw random numbers. Here are a few useful ones:

`rand(m,n)`	creates an $m \times n$ matrix of U(0,1)
`randn(m,n)`	creates an $m \times n$ matrix of N(0,1)
`betarnd(A,B,m,n)`	creates an $m \times n$ matrix of Beta(A,B)
`chi2rnd(nu,m,n)`	creates an $m \times n$ matrix of $\chi^2(\nu)$
`exprnd(mu,m,n)`	creates an $m \times n$ matrix of $\exp(\mu)$
`gamrnd(A,B,m,n)`	creates an $m \times n$ matrix of $\Gamma(A,B)$
`logrnd(mu,sig,m,n)`	creates an $m \times n$ matrix of $\log N(\mu,\sigma)$
`trnd(nu,m,n)`	creates an $m \times n$ matrix of $t(\nu)$

10 Some Examples

10.1 loaddata.m

```
% load_data.m, f.canova 6-6-2004
```

```
%
% example on how to load data, run a linear regression calling a ols
% routine, plotting the time series and computing some statistics of the
% data

% this program call CAL.m, OLS,m, MPRINT.m


% a) read in the data from an ascii file

load ypr.dat;              % loading data
[nobs,nvar] = size(ypr);   % # Observations in time series
y = ypr(:,1); p = ypr(:,2); r = ypr(:,3); dates = cal(1948,1,4);
%detrend y, p remove linear trend
tr = (1:nobs)'; cr = ones(nobs,1); yt = ols(y,[cr tr]);
y1 = y-yt.beta(2)*tr;       % output gap
pt = ols(p,[cr tr]);
p1 = p-pt.beta(2)*tr;       % detrended inflation gap

% collect detrended data in X
X = [y1 p1 r];

 % b)  plot the time series
header = 'Fig. 1: Time series'; F1 = figure(1); clf
set(F1,'numbertitle','off') set(F1,'name',header) tit =
strvcat('y','p','r'); tt = (1:nobs)'; for i=1:3
    subplot(2,2,i)
    plot(tt,X(:,i))
    title(tit(i,:))
end

% c) statistics on the time series
stats_r  = [mean(r) std(r) min(r) max(r)]; stats_y1  = [mean(y1)
std(y1) min(y1) max(y1)]; stats_p1  = [mean(p1) std(p1) min(p1)
max(p1)]; inv.cnames = strvcat('mean','std','min','max');
inv.rnames = strvcat('series','r','y1','p1'); disp('*** Summary
Statistics of time series ***')
disp('---------------------------------------- ')
mprint([stats_r;stats_y1;stats_p1],inv)
```

10.2 example_var.m

```
% example_var.m; f. canova, version 1.2, june 28-6-2005
% This example shows how to run a VAR (there are better programs around
% but this does every step clearly).
% It estimates parameters, computes unconditional forecasts and impulse responses
% using  a choleski decomposition

%%%%%%%%%%%%%%%%%%%%% practice exercises %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% exercise 1: change the ordering of the varables in the var
% exercise 2: reduce the number of lags to 4 and eliminate the linear trend
% exercise 3: run the system with output, interest rates and inflation
% exercise 4: run impulse responses assuming a non-zero impact of variable
%             2 on variable 1 (need to modify the command that generates s
%             choose e.g. the upper triangular value to be -0.5 o 0.5)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear session clear

% read in the data (ascii format TxN matrix, T=# of observations N =# of  variables)
load var.dat
% data from 1973:1 1993:12;  order of data
% 1: y
% 2: p
% 3: short rate
% 4: m1
% 5: long rate


% pick the variables of the var system
indepvar=1  % column of variable  1
depvar=4    % column of variable  4
pociatok=1; %

dimension=(size(var)) [var(:,4) var(:,1)];
rawdat=[var(pociatok:dimension(1,1),indepvar)
var(pociatok:dimension(1,1),depvar)];
%display('Rawdat:')
%rawdat(:,:)
dimension=(size(rawdat));     %# of observations
totnobs=dimension(1,1); nvars=dimension(1,2);

%dat = log(rawdat); % converts raw data to logs
dat = rawdat;

% INITIALIZATION FOR VAR ESTIMATION AND FORECASTING
nfore = 12;                 % # OF FORECASTING STEPS TO BE CALCULATED @
nsteps = 36;                % # of steps to compute for impulse response functions @
nlags = 6;                  % # OF LAGS INCLUDED IN VAR @
ndeterm = 2;                % # OF DETERMINISTIC VARIABLES IN VAR @
depnobs = totnobs - nlags;  % # OF DEPENDENT OBSERVATIONS @
ncoeffs = nvars*nlags+ndeterm; % # OF COEFFICIENTS IN EACH VAR EQUATION @
```

```
nparams = nvars*ncoeffs;    % TOTAL NUMBER OF ESTIMATED PARAMETERS @
df = depnobs - ncoeffs;     % DEGREES OF FREEDOM PER EQUATION @


%  SET UP DETERMINISTIC VARIABLES
const = ones(depnobs,1); dummy=1:1:depnobs; trend=dummy'; determ =
[const trend];

% SET UP DEPENDENT VARIABLES
y = zeros(depnobs,nvars); y(:,:) = dat((nlags+1):totnobs,:);

% SET UP INDEPENDENT VARIABLES
x = zeros(depnobs,nvars*nlags); i = 1; while i <= nvars;
   j = 1;
   while j <= nlags;
      x(:,j+((i-1)*nlags)) = dat(nlags+1-j:totnobs-j,i);
      j = j+1;
   end;
   i = i+1;
end;

x = [x determ]; sizex=size(x); sizey=size(y);

% ESTIMATE VAR
xxx = inv(x'*x);      % (x'x)^(-1) matrix
sizexxx=size(xxx);
b = xxx*(x'*y);       % ols estimator (this is a system wide estimator)
sizexy=size(x'*y); sizeb=size(b);
res = y - x*b;        % residuals
sizeres=size(res);
vmat = (1/depnobs)*(res'*res);  %vcov of residuals
stds = sqrt(diag(vmat));        % Std devs of residuals
%disp('Std devs'); stds

s = chol(vmat)';                       % Cholesky decomp of vmat
seb = sqrt( diag(xxx)*(diag(vmat)') ); % STANDARD ERRORS estimated coefficients
%disp('STANDARD ERRORS OF B '); seb
vcorr = zeros(nvars,nvars);            % CORRELATION MATRIX OF residuals

i = 1; while i <= nvars;
   j = 1;
   while j <= nvars;
        vcorr(i,j) = vmat(i,j)/(sqrt(vmat(i,i)*vmat(j,j)));
      j = j+1;
   end;
   i = i+1;
```

```
end;

display('Estimates, std errors and t statistics for variable');
% order is estimates for the two equations, st. err of the two equations,
% t- stat for the two equations
[b(:,:) seb(:,:) b(:,:)./seb(:,:)]

%disp('COVARIANCE MATRIX OF RESIDUALS:'); vmat
disp('CORRELATION MATRIX OF RESIDUALS:'); vcorr


% CONSTUCTION OF A COMPANION MATRIX (FOR COMPUTING
% FORECASTS) AND COMPANION MATRIX LESS DETERMINISTIC COMPONENTS (FOR
% COMPUTING DOMINANT ROOTS, IMPULSE RESPONSES


dimcmp = ncoeffs;         % dimension of companion matrix
iden = eye(dimcmp);
dimcmpld = nvars*nlags; % "ld" trailer: "less deterministic components"
idenld = eye(dimcmpld);

% CONSTRUCT FINAL OBSERVATIONS, TO BE USED BY THE COMPANION MATRIX TO COMPUTE FORECASTS

yT = zeros(ncoeffs,1); i = 1; while i <= nvars;
   j = 1;
   while j <= nlags;
        yT(((i-1)*nlags+j),1) = y(depnobs-j+1,i);
    j = j+1;
    end;
    i = i+1;
end;

yT(ncoeffs-1,1) = 1;            % ADJ SPECIFIES CONSTANT IN yT
yT(ncoeffs,1) = depnobs+1;      % ADJ SPECIFIES TREND IN yT

% CONSTRUCT COMPANION MATRIX FROM B (NCOEFFS X NVARS).
% THIS IS COMPLETELY GENERAL WITH REGARDS TO THE DIMENSIONALITY OF THE VAR
dimb=size(b);
bld = b(1:(dimb(1,1)-ndeterm),:);   % REMOVES DETERMINISTIC COMPONENTS @
ald = zeros(dimcmpld,dimcmpld);     % COMPANION MATRIX LESS DETERM COMPS @
a = zeros(dimcmp,dimcmp);           % COMPANION MATRIX @

dimbld=size(bld'); dimidenld=size(idenld); j = 1;

while j <= nvars;
    ald(((j-1)*nlags+1):(j*nlags),:) =[ bld(:,j)';...
    idenld(((j-1)*nlags+1):(j*nlags-1),:)];
    a(((j-1)*nlags+1):(j*nlags),:) =    [b(:,j)';...
    iden(((j-1)*nlags+1):(j*nlags-1),:)];
   j = j+1;
end; dimald=size(ald);

a(dimcmp-1,:) = iden(dimcmp-1,:);       % ADJ SPECIFIES ROW FOR CONSTANT
a(dimcmp,:) = iden(dimcmp,:);           % ADJ THIS AND NEXT LINE SPECIFY...
a(dimcmp,dimcmp-1) = 1;                 % ADJ ... ROW FOR TREND

% COMPUTE ROOTS OF THE VAR FROM COMPANION MATRIX
EI=eig(ald); size(EI); ROOTS=abs(EI); ROOTS = sortrows(ROOTS,1);
MAXRT = ROOTS(dimcmpld,:);

display('Roots of the VAR'); ROOTS

% COMPUTE NFORE-STEP-AHEAD FORECASTS
fore = zeros(nfore,nvars);          % WILL CONTAIN FORECASTS
an = a;                             % AN WILL BECOME A^N BELOW
forecst = an*yT;

j = 1; while j <= nvars;
    fore(1,j) = forecst(((j-1)*nlags+1),1);
   j = j+1;
end;

k = 2; while k <= nfore;
    an = an*a;
    forecst = an*yT;
   j = 1;
   while j <= nvars;
      fore(k,j) = forecst(((j-1)*nlags+1),1);
      j = j+1;
   end;
   k = k+1;
end; fore

 % Here, we graph the last few observations of each series, and the forecastS.

clear dummy dummy=1:1:2*nfore; xrange =dummy'; clear dummy;

dimxrange=size(xrange); dimy=size(y); depnobs-(depnobs-nfore+1);
dimfore=size(fore);

yfore=[y(depnobs-nfore+1:depnobs,:);fore(:,:) ];
dimyfore=size(yfore);
```

```
message=strcat('Last few observations of each series, and the '...
'corresponding forecast -indep. variable # ',num2str(indepvar,3))
figure; plot(xrange,yfore(:,1),'-b',xrange,yfore(:,2),'-r')
title(message) grid on ; xlabel('Month');
ylabel('Observation/Forecast')


% ************ Generate impulse response functions. ************************
% We generate nvars*nvars responses: one for each variable in response to
% each shock. Each response is of length nsteps. The following code is
% inefficient (it calculates A^nsteps nvars times), but transparent. The
% responses are a function of the ordering selected for
% the Cholesky decomposition of vmat.

%steps = seqa(0,1,nsteps+1);
pom= 0:1:nsteps; steps=pom';clear pom;

respzeros=zeros(nsteps+1,nvars+1,nvars); i = 1;
while i <= nvars;                 % i indexes the shocks
   esp = zeros(nsteps+1,nvars);   % WILL CONTAIN IMPULSE RESPONSE FCNS
   shock = zeros(nvars,1);
   shock(i) = 1/s(i,i);           % Normalizes shocks to (orthogonalized)
   shock = s*shock;               % std dev units
   resp(1,:) = shock';
   bigshock = zeros(dimcmpld,1);  % expands shock to be compatible with
   j = 1;                         % companion matrix
   while j <= nvars;
      bigshock((j-1)*nlags+1) = shock(j);
      j = j+1;
   end
   an = ald;
   k = 2;
   while k <= nsteps+1;            % k indexes the step size
      bigresp = an*bigshock;
      j = 1;
      while j <= nvars;
         resp(k,j) = bigresp((j-1)*nlags+1,1);
         j = j+1;
      end;
      an = an*ald;
      k = k+1;
   end;
   dimsteps=size(steps);
   dimresp=size(resp);
   aaa=nsteps+1;
   display('impulse responses to shock to variable');
   i
   [steps resp(:,:)]
   respzeros(:,:,i)=[resp(:,:) zeros(nsteps+1,1)];
   i = i+1;
end;

%respzeros=[resp zeros(nsteps+1,1)]

for i=1:nvars;
   figure;
   message=strcat('Impulse responses - shock to variable # ',...
   num2str(i,3),' system with variables ', num2str(indepvar,3),...
   'and ', num2str(depvar,3));
   plot(steps,respzeros(:,1,i),'b.-',steps,respzeros(:,2,i),...
   '-r',steps,respzeros(:,3,i),'k--');
   title(message) ;
   grid on ;
   if i==1    axis([0 20 -5.0 1.5]); end;
   xlabel('Month');
   ylabel('Response');
   legend('Variable # 1 ','Variable #4','zero value');
end
```

10.3 example_gmm.m

```
% Program name: example_gmm.m;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Example GMM Estimation                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% The program uses the firm's euler  equation to estimate the ajustment cost
% parameter (gamma) and the capital share (alpha). See Adda-Cooper section 8.4.3
% for a discussion of  this model.
% NOTE: the program uses fminserach which does not produce standard error
% of the estimates

%it call model1.m, varcov.m, and the data book2.m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% exercise 1: fix alpha to 0.5 and find optimal gamma
% exercise 2: jointly estimate alpha, gamma, delta
% exercise 3: use more instruments (instruments are defined in the
%              file model1)
% exercise 4: check if results are different with a two-step estimator.
% exercise 5: allow for serial correlation in the error (need to change
%               the definition of W in varcov)
% exercise 6: cut sample size by half (use the last 190 data points)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;                          % Clears all variables
clc                                 % Clears the command window

% Start the timer
tic

% Load the data set
% invpc = investment rate (i)
% shc = productivity shock (A)
% prok = profit rate (AK^alpha-1)
load book2; data = [invpc; shc; prok];

% Parameters
beta = .95;   % these are calibrated
delta = .069; guess = [1 1];
p = 1;                       % Normalize price to one

T = length(data);
maxiter = 10;           % Convergence is fast therefore only need two iterations
                        % this is a parameter to play with
iter = 1;
W = eye(2);             % Step One of the GMM algorithm

while iter <= maxiter;
    [results J]= fminsearch(@model1,guess,[],data,beta,delta,p,T,W);
    gamma = results(1);
    alpha = results(2);

    % Calculate the Optimal Weight Matrix (Step Two)
    W = varcov(gamma,alpha,data,beta,delta,p,T);
    iter = iter +1;
end

disp('*****************') disp('     Results')
disp('*****************') disp(['Gamma = ' num2str(gamma)])
disp(['Alpha = ' num2str(alpha)]) disp(['J statistic = '
num2str(T*J)]) disp('*****************')

% End the timer
toc
```

10.4 draw_rn.m

```
%%% example of how to draw random numbers

%%%% Random Number Generators in matlab
%%%% randn.m     Normal distribution
%%%% rand.m      Uniform distribution
%%%% gamrnd.m    Gamma distribution
%%%% wishrnd.m   Whishart distribution
%%%% iwishrnd.m  Inverted Whishart distribution
%%%% mvnrnd.m    Multivariate Normal
%%%% betarnd.m   Beta Distribution
%%%% type 'help xxx.m' for information on what these functiosn do.

% note: without the seed the draws will be different at each run.


% exercise 1: draw measurement errors from a uniform distribution with
%             the same variance
% exercise 2: draw technology shocks from a t- distribution with 4 dof.
% exercise 3: draw consumption measurement error from a beta(5,2)

% parameters
ndraws=2000; % number of draws
k0=100;      % initial condition on capital
A=2.83;      % constant in the production function
etta=0.36;   % share of capital in production function
betta=0.99;  % discount factor
ssig1=0.1;   % standard error of technology shock
ssig2=0.06;  % measurement errors
ssig3=0.02; ssig4=0.2; ssig5=0.01;

for t=1:ndraws
    uu1(t)=1+randn(1)*ssig1;
    uu2(t)=randn(1)*ssig2;
    uu3(t)=randn(1)*ssig3;
    uu4(t)=randn(1)*ssig4;
    uu5(t)=randn(1,1)*ssig5;
    kk(t)=A*etta*betta*k0*uu1(t)+uu2(t);
    yy(t)=A*k0^etta*uu1(t)+uu3(t);
    cc(t)=yy(t)*(1-etta*betta)+uu4(t);
    rr(t)=etta*(yy(t)/k0)+uu5(t);
    k0=kk(t);
end

% this is the same loop in matrix format
```

```
%
%uu1=ones(ndraws,1)+rand(ndraws,1).*ssig1;
%uu2=rand(ndraws,1)*ssig2;
%uu3=rand(ndraws,1)*ssig3;
%uu4=rand(ndraws,1)*ssig4;
%uu5=rand(ndraws,1)*ssig5;
%kk=A*etta*betta*uu1+uu2;
%yy=A*k0*etta*uu1+uu3;
%cc=yy*(1-etta*betta)+uu4;
%k0=kk(t);

% collecting statistics of simulated data
cc1=cc(400:end); yy1=yy(400:end); disp(['                Mean
Std       Skewness      Kurtosis']) disp(['Consumption    '
num2str([mean(cc1) sqrt(cov(cc1)) skewness(cc1) kurtosis(cc1)])])
disp(['Output         ' num2str([mean(yy1) sqrt(cov(yy1))
skewness(yy1) kurtosis(yy1)])])

% scalign variables
cc2=(cc1-mean(cc1))/sqrt(cov(cc1));
yy2=(yy1-mean(yy1))/sqrt(cov(yy1));

% histogram and density estimation
[f1]=ksdensity(cc2,sort(cc2)); [f2]=ksdensity(yy2,sort(yy2));
figure(1), subplot(2,1,1),hist(cc2,100),axis
tight,title('Consumption'); subplot(2,1,2),hist(yy2,100),axis
tight,title('Output'); figure(2),
subplot(2,1,1),plot(sort(cc2),f1,'r'),axis
tight,title('Consumption')
subplot(2,1,2),plot(sort(yy2),f2,'r'),axis tight,title('Output');
```