Chapter 10

An Introduction to MATLAB

10.1 Introduction

MATLAB is a modern software package for technical computing. It proved to be a powerful tool for scientific and engineering numerical computation, visualization, and programming. It is able to solve efficiently complex numerical problems arising in different areas of science and engineering. The name MATLAB is derived from MATrix LABoratory.

This tutorial is designed to assist you in learning to use MATLAB. No history of using MATLAB is required. A preliminary knowledge of elementary linear algebra concepts is assumed. A background in programming principles is desirable for understanding the programming capabilities of the package. For additional information on functions, commands, and examples the reader is encouraged to use the on-line help facility and Reference/User's guide attached to the software and available from http://www.mathworks.com.

10.2 Running MATLAB

To start MATLAB on a Microsoft Windows platform run file matlab.exe (this can be done by simply double-clicking the MATLAB shortcut icon which is usually located on Windows desktop). To start MATLAB on Unix, type `matlab` at the operating system prompt.

This will open the MATLAB desktop.

10.3 Matrix Basics

10.3.1 Creating Matrices

Matrices in MATLAB can be entered in several different ways. We will start with introducing a matrix by an explicit list of its elements. In this case, the list of elements representing the matrix is surrounded by square brackets. The elements of the same row are separated by blanks or commas; finally, two rows are separated by semicolon or by starting a new line.

Example 10.1. The matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 4 & 3 \\ 2 & 2 & 4 \end{bmatrix}$$

can be equivalently introduced in the following ways.

(a) `>> A = [1 2 3 ; 4 4 3 ; 2 2 4]`

(b)
```
>> A = [1 2 3
       4 4 3
       2 2 4]
```

(c)
```
>> A = [1, 2, 3; 4 4 3
       2 2 4]
```

In all three cases MATLAB will reply with

```
A =
    1  2  3
    4  4  3
    2  2  4
```

In the above, `A` is called *variable*, the set of symbols after equal sign is called *expression*; the varible and expression, together with the equal sign form *statement*.
In general, a statement can consist of a (previously defined) variable only or of an expression only. In the last case (when a statement is an expression), the result of the expression is assigned to a default variable `ans` (a short for answer).

Example 10.2.

```
>> [4 5 6 7]

ans =
     4     5     6     7
```

In the case when a statement consists of a variable only, MATLAB returns the value of this variable (or an error message, if the variable was not previously defined). If a statement is finished with a semicolon (;), the output is suppressed, but the operation determined by this statement is completed.

Example 10.3.

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

B =
     1     2     3
     4     5     6

>> b
??? Undefined function or variable 'b'.
```

The last example also demonstrates that MATLAB is case-sensitive in the names of variables (functions, commands), i.e. variable `B` is not the same as `b`.

10.3.2 Dimensioning of Matrices

Dimensioning is done automatically as soon as you enter or change a matrix.

Example 10.4. If we enter

```
>> B = [1 1; 2 1]
```

and then later on we enter

```
>> B = [1 1 1; 2 2 3]
```

then matrix `B` is automatically changed from 2×2 to a 2×3 matrix.

For an $m \times n$ matrix B and given $1 \leq i \leq m$ and $1 \leq j \leq n$, the expression `B(i,j)` (called *subscript*) refers to the element in the i-th row and the j-th column of B. The expression `B(i,j)` can be treated as a variable which is already defined when $1 \leq i \leq m$ and $1 \leq j \leq n$, and can be defined later on for $i \geq m$ or $j \geq n$. For a number k and positive integers i and j, the statement `B(i,j) = k` will result in the following:

- if $1 \leq i \leq m$ and $1 \leq j \leq n$ then the element in the i-th row and the j-th column of B takes value k, and the rest of the matrix is unchanged;
- if $i \geq m+1$ or $j \geq n+1$ then the size of the matrix is increased to fit $B(i,j)$. All the new components (except $B(i,j)$) which appeared as a result of the size increasing are set equal to zero.

In both cases, MATLAB will output the whole new matrix B.

Example 10.5.

```
>> B=[1 2
      2 1];
>> B(1,1)=5

B =

      5           2
      2           1

>> B(3,4)=8

B =

      5           2           0           0
      2           1           0           0
      0           0           0           8
```

Given a vector c, a single subscript `c(i)` represents i-th element of the vector. If applied to an $m \times n$ - matrix B, a single subscript `B(i)` refers to the i-th element of the $m \times n$- dimensional vector formed from the columns of B.

Example 10.6.

```
>> c=[9 6 5 4];
>> c(3)

ans =

     5

>> B=[1 7 4; 3 5 9];
>> B(3)

ans =

     7
```

Given a matrix `C` we can find its size by using the following command:

```
[m, n] = size(C);
```

where the variable `m` is assigned the number of rows of `C` and the variable `n` the number of columns. To find the length of a vector `c` we can also use function `length`:

```
n = length(c)
```

Example 10.7.

```
>> c = [2 0 0 1];
>> [m, n] = size(c)

m =
     1

n =
     4

>> n = length(c)

n =

     4
```

10.3.3 The Colon Operator

One of the most important MATLAB's operators is the colon, :. It enables users to create and manipulate matrices efficiently. Using colon notation with some numbers a, b and h, the expression

```
a:h:b
```

represents a vector

$$[a, a+h, a+2h, \ldots, a+kh].$$

In the above k is the largest integer, for which $a+kh$ is in the interval $[\min\{a,b\},\ \max\{a,b\}]$.

Example 10.8.

```
>> v = 1:2:10

v =

     1     3     5     7     9

>> u=10:-3:-3

u =

    10     7     4     1    -2
```

When $h = 1$, the expression `a:h:b` is equivalent to a shorter one `a:b`.

Example 10.9.

```
>> 1:8

ans =

     1     2     3     4     5     6     7     8
```

Colons can be used to constract not only vectors, but matrices as well (see the example below). A subscript expression containing colons refers to a submatrix of a matrix. A colon by itself in the subscript denotes the entire row or entire column.

Example 10.10. First, we constract a matrix, consisting of three rows each of which is built using colon notation:

```
>> A = [1:5; 5:-1:1; 2:6]

A =

     1     2     3     4     5
     5     4     3     2     1
     2     3     4     5     6
```

Then we use colons in the subscript to extract a submatrix of A corresponding to its first two rows and first three columns:

```
>> A(1:2,1:3)

ans =

     1     2     3
     5     4     3
```

The submatrix consisting of the first and the third rows of A can be found as follows:

```
>> A([1,3],:)

ans =

     1     2     3     4     5
     2     3     4     5     6
```

Similarly, we can modify a submatrix of A:

```
>> A(1:2,[1,3])=[1 0; 0 1]

A =

     1     2     0     4     5
     0     4     1     2     1
     2     3     4     5     6
```

Here a 2×2 submatrix of A built from elements of its fist two rows and its first and third columns, is replaced by the 2×2 identity matrix.

10.3.4 Matrix Operations

The basic matrix operations are the following:

```
+  addition
-  subtraction
*  multiplication
^  power
'  transpose
\ left division
/  right division
```

Noted that the dimensions of the matrices used should be chosen in a way that all these operations are defined, otherwise an error message will occur. To add two matrices `A` and `B` we type

```
E = A + B
```

and the matrix `E` is the result of their addition. Respectively, for multiplication

```
E = A*B
```

To raise a square matrix `A` to a power `p`

```
E = A^p
```

Note that `E = A^(-1)` is nothing else than assigning the inverse of `A` to `E`.

The `'` operator defines the transpose of the matrix or a vector.

Example 10.11. The following statements are equivalent

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

>> v = [1 2 3 4]'
```

(try it).

The matrix division operators are convenient for solving systems of linear equations, where `A\b` is nothing else than $A^{-1}b$, provided that A is nonsingular.

Example 10.12. Solve a system $Ax = b$, where A and b are defined below, using left division:

```
>> A = [1 4 3; -1 -2 0; 2 2 3]

A =
     1     4     3
    -1    -2     0
     2     2     3

>> b = [12 ; -12 ; 8]

b =
    12
   -12
     8

>> x=A\b

x =
    4.0000
    4.0000
   -2.6667
```

Having defined the left division, the right division is then introduced as `b/A = (A'\b')'`. When applied to scalars, right and left division differ only in the direction in which the division is made.

Example 10.13.

```
>> 3/2

ans =
     1.5000

>> 3\2

ans =
     0.6667
```

If the operators `\ / * ^` are preceded by a period, then they perform the same acion but entry-wise (similarly to addition or subtraction of matrices).

Example 10.14.

```
>> A=[2 4
      3 1];
>> B=[3 4
      2 3];
>> A.*B

ans =

     6    16
     6     3
```

(Compare the answer to the result of matrix multiplication of `A` and `B`, `A*B`).

10.3.5 Special Matrix - Building Functions

MATLAB includes some functions that generate known special matrices. Some of the most frequently used are

Special Matrix - Building Functions		
`A = eye(m,n)`	$\rightarrow$	$m \times n$ Identity matrix
`A = zeros(m,n)`	$\rightarrow$	$m \times n$ Zero matrix
`A = ones(m,n)`	$\rightarrow$	$m \times n$ Matrix with ones
`A = rand(m,n)`	$\rightarrow$	$m \times n$ Uniform random elements matrix
`A = randn(m,n)`	$\rightarrow$	$m \times n$ Normal random elements matrix

To create a square matrix, the second argument can be omitted, for example, to generate an identity matrix $A_{n\times n}$ we simply type `A = eye(n)` .

Example 10.15.

```
>> A = rand(3,2)

A =
    0.2190    0.6793
    0.0470    0.9347
    0.6789    0.3835
```

creates a 3×2 matrix of uniformally distributed on $(0, 1)$ random elements

```
>> B=eye(3)+ones(3)

B =
     2     1     1
     1     2     1
```

```
     1     1     2

>> C=zeros(2,3)

C =
     0     0     0
     0     0     0
```

10.4 Managing the Workspace and the Command Window

10.4.1 Saving a session; hardcopy

Whenever you start a MATLAB session and start working, you are actually working in a workspace where all the variables and matrices that you define are kept in memory, and can be recalled any time. To clear some of the variables, the command `clear` is used, which if entered without arguments clears the values of all variables in the workspace. Entering

```
>> clear x
```

would simply clear the value of variable `x`.

If you exit MATLAB then all of the previously defined variables and matrices will be lost. To avoid this, you can save your work before quitting the session using the command

```
>> save filename.mat
```

This will save the session to a binary file named `filename.mat`, which can be later retrieved with the command

```
>> load filename.mat
```

If you omit the filename then the default name given by MATLAB is `matlab.mat`. However, even if you save the variables of your workspace in a file, all the output that was generated during a session has to be re-generated. This is where the `diary` command is used. More specifically, if you enter

```
diary myfile.out
```

then MATLAB starts saving all the output that is generated in the workspace to the file `myfile.out`; the command `diary off` stops saving the output.

10.4.2 Command line editing and recall

When editing the command line in MATLAB, the left/right arrows are used for the cursor positioning, whereas pressing the Backspace or Delete key deletes the character to the left of the cursor. Enter `help cedit` to check other command line editing settings.

To recall one of the previous command lines, we can use the up/down arrows. When recalled, a command line can be modified and executed in the revised form. This feature is especially convenient when we are dealing with long statements.

10.4.3 Output format

All computations in MATLAB are done in double precision. The command `format` can be used to change the output display format for the most convenient at the moment.

`format`	Default. Same as `short`.
`format short`	Scaled fixed point format with 5 digits.
`format long`	Scaled fixed point format with 15 digits.
`format short e`	Floating point format with 5 digits.
`format long e`	Floating point format with 15 digits.
`format short g`	Best of fixed or floating point format with 5 digits.
`format long g`	Best of fixed or floating point format with 15 digits.
`format hex`	Hexadecimal format.
`format +`	The symbols +, - and blank are printed for positive, negative and zero elements. Imaginary parts are ignored.
`format bank`	Fixed format for dollars and cents.
`format rat`	Approximation by ratio of small integers.
Spacing:	
`format compact`	Suppress extra line-feeds.
`format loose`	Puts the extra line-feeds back in.

When a format is chosen, it remains effective until changed.

Example 10.16. Numbers π and e are well-known irrational constants. In MATLAB, there is a built-in constant `pi` approximating π. An approximation of e can be found using the function `exp(1)` (see Section 10.5).

```
>> a = pi

a =
    3.1416

>> b = exp(1)

b =
    2.7183

>> format long
>> a

a =
   3.14159265358979

>> b

b =
   2.71828182845905
```

10.4.4 Entering long command lines

If a statement is too long to fit on one line, three or more periods, ..., to continue the statement on the next line.

Example 10.17.

```
>> q = 1000 + sqrt(10) - 35/2 - exp(5) + log10(120) -...
cos(8*pi/13) + 3^2;
```

10.5 Functions

There is a great number and variety of functions that are available in MATLAB. All MATLAB functions can be subdivided into two types, built-in functions and user-defined functions. In this section a brief overview of the most important built-in functions is provided, while the development of user-defined functions is described in Section 10.7.

10.5.1 Scalar Functions

MATLAB has built-in functions for all the known elementary functions, plus some specialized mathematical functions. Below is a list of some of the most common scalar functions.

Some of MATLAB Scalar Functions					
`sin(x)`	$\rightarrow$	$\sin(x)$	`asin(x)`	$\rightarrow$	$\arcsin(x)$
`cos(x)`	$\rightarrow$	$\cos(x)$	`acos(x)`	$\rightarrow$	$\arccos(x)$
`tan(x)`	$\rightarrow$	$\tan(x)$	`atan(x)`	$\rightarrow$	$\arctan(x)$
`sinh(x)`	$\rightarrow$	$\sinh(x)$	`asinh(x)`	$\rightarrow$	$\sinh^{-1}(x)$
`cosh(x)`	$\rightarrow$	$\cosh(x)$	`acosh(x)`	$\rightarrow$	$\cosh^{-1}(x)$
`tanh(x)`	$\rightarrow$	$\tanh(x)$	`atanh(x)`	$\rightarrow$	$\tanh^{-1}(x)$
`exp(x)`	$\rightarrow$	e^{x}	`ceil(x)`	$\rightarrow$	$\lceil x \rceil$
`log(x)`	$\rightarrow$	$\ln(x)$	`floor(x)`	$\rightarrow$	$\lfloor x \rfloor$
`log10(x)`	$\rightarrow$	$\log_{10} x$	`round`	$\rightarrow$	rounding (nearest)
`abs(x)`	$\rightarrow$	$\lvert x \rvert$	`fix(x)`	$\rightarrow$	round towards zero
`sqrt(x)`	$\rightarrow$	$\sqrt{x}$	`rem(x,y)`	$\rightarrow$	remainder after division

Example 10.18. By definition, $\sinh(x) = \frac{e^x - e^{-x}}{2}$. So, for $x = 1$ we have

```
>> sinh(1)

ans =
     1.1752
```

```
>> (exp(1)-exp(-1))/2

ans =
     1.1752
```

The reminder after division of 3 by 2 is equal to 1:

```
>> rem(3,2)

ans =
     1
```

The function `round` rounds a number towards the closest integer. It is not the same as the function `fix`, which outputs the closest integer towards zero:

```
>> round(2.9)
ans =
     3
```

```
>> fix(2.9)
ans =
     2
```

Functions `floor` and `ceil` round a number to the closest non-larger and non-smaller integer, respectively:

```
>> floor(3.5)
ans =
     3
```

```
>> ceil(3.5)
ans =
     4
```

The following example shows using exponential and logarithmic functions:

```
>> exp(1)
ans =
    2.7183
```

```
>> log10(1000)
ans =
     3.0000
```

In all of the presented functions, if the argument `x` is an $m \times n$ matrix, then the function will be applied to each of its elements and the dimension of the resulting answer will be also a $m \times n$ matrix.

Example 10.19. In this example we take the square roots of absolute values of all elements of a matrix A, and then round up the resulting elements, assigning the final result to a matrix D. First, we do this step by step:

```
>> A = [2 3 -5; 4 2 2; -1 -4 7]

A =
     2     3    -5
     4     2     2
    -1    -4     7

>> B = abs(A)

B =
     2     3     5
     4     2     2
     1     4     7

>> C = sqrt(B)

C =
    1.4142    1.7321    2.2361
```

```
    2.0000    1.4142    1.4142
    1.0000    2.0000    2.6458

>> D = ceil(C)

D =
     2     2     3
     2     2     2
     1     2     3
```

The same can be done in one line, using a superposition of all of the functions applied:

```
>> D = ceil(sqrt(abs(A)))

D =
     2     2     3
     2     2     2
     1     2     3
```

10.5.2 Vector Functions

MATLAB's vector functions operate both on vector-rows and vector-columns. When applied to a matrix, a vector function acts column-wise. For example, if `fun` is a vector function, and `fun(x)` is a number, then if applied to a matrix A, `fun(A)` produces a row vector, containing the results of application of this function to each column of A.

Some of the MATLAB Vector Functions		
`max(x)`	→	Largest component
`min(x)`	→	Smallest component
`sort(x)`	→	Sort in ascending order
`sum(x)`	→	Sum of elements
`prod(x)`	→	Product of elements
`mean`	→	Average or mean value
`median`	→	Median value
`std`	→	Standard deviation
`all`	→	True (1) if all elements of a vector are nonzero
`any`	→	True (1) if any element of a vector is nonzero

Example 10.20. By definition,

$$\texttt{mean(x)} = \sum_{i=1}^{n} x_i/n;$$

$$\texttt{std(x)} = \sqrt{\frac{\left(\sum_{i=1}^{n} x_i^2 - n \cdot \texttt{mean(x)}^2\right)}{n-1}}.$$

Let's generate a random vector x, and check these definitions using MATLAB.

```
>> n=5;
>> x=rand(1,n)

x =
    0.3843    0.9427    0.2898    0.4357    0.3234

>> mean(x)

ans =
     0.4752

>> sum(x)/n

ans =
     0.4752

>> std(x)

ans =
     0.2673

>> sqrt((sum(x.^2)-n*mean(x)^2)/(n-1))

ans =
     0.2673
```

Example 10.21. This example shows the (column-wise) action of the function `max` applied to a randomly generated matrix A. When applied twice, `max(max(A))` outputs the maximum element in the entire matrix.

```
>> A=rand(3)

A =
    0.8637    0.0562    0.6730
    0.8921    0.1458    0.3465
    0.0167    0.7216    0.1722

>> max(A)

ans =
    0.8921    0.7216    0.6730

>> max(max(A))

ans =
    0.8921
```

10.5.3 Matrix Functions

The matrix functions included in MATLAB cover the majority of matrix operations from elementary Gaussian elimination to sparse matrix operations used in topics such as graph theory. Below is a list of various elementary matrix functions.

Elementary Matrix Functions		
`norm(A)`	→	The norm of `A`
`rank(A)`	→	The dimension of the row space of `A`
`det(A)`	→	The determinant of `A`
`trace(A)`	→	The sum of the diagonal elements of `A`
`diag(A)`	→	Diagonal of `A`
`tril(A)`	→	Lower triangular part of `A`
`triu(A)`	→	Upper triangular part of `A`
`null(A)`	→	The nullspace of `A`
`rref(A)`	→	Reduced Row Echelon Form of `A`
`[l,u]=lu(A)`	→	LU factorization triangular matrices
`inv(A)`	→	The inverse of `A`
`[v,d]=eig(A)`	→	v: eigenvectors, d: eigenvalues of `A`
`poly(A)`	→	$p(\lambda) = det(\mathtt{A} - \lambda I)$: characteristic polynomial of `A`

If `A` is an $n \times 1$ vector then, by definition,

$$\mathtt{norm(A,p)} = \left(\sum_{i=1}^{n} |\mathtt{A}_i|^{\mathtt{p}} \right)^{\frac{1}{\mathtt{p}}}.$$

When `A` is an $n \times m$ matrix, then `norm(A)` is the largest singular value of `A` while `norm(A,'fro')` is the Frobenius norm of the matrix

$$\|\mathtt{A}\|_F = \left(\sum_{i=1}^{n} \sum_{j=1}^{m} \mathtt{A}_{ij}^2 \right)^{\frac{1}{2}}$$

The function `rref(A)` provides the reduced row echelon form of matrix `A`, for example

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

A =
      3           4           5
      3          -2           1

>> rref(A)

ans =
      1           0         7/9
      0           1         2/3
```

An interesting version of this function is the `rrefmovie(A)` which pauses at each elementary row operation used to reduced the matrix `A` to row echelon form.

An $n \times n$ system of equations can be written as $Ax = b$. Expressing A as the product of two triangular matrices $A = LU$ and letting

$$y = Ux,$$

we have

$$Ly = b.$$

Therefore, we can first solve the last system for y using forward substitution, and knowing y we can solve the system $Ux = y$ for x using back substitution. The matrices L, U are provided by the function `[L,U]=lu(A)`.

Example 10.22. In this example we find the solution to the linear system

$$\begin{array}{rcrcrcr} 3x_1 & + & 4x_2 & + & 1x_3 & = & -2 \\ 2x_1 & - & 1x_2 & + & 5x_3 & = & 5 \\ 5x_1 & + & 6x_2 & + & 2x_3 & = & 4 \end{array}$$

and verify the solution. We use rational approximation output format (`format rat`).

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

>> b=[-2; 5; 4];

>> A\b

ans =
     138/5
     -89/5
     -68/5

>> [L,U]=lu(A)

L =
     3/5         -2/17          1
     2/5           1            0
      1            0            0


U =
      5            6            2
      0         -17/5         21/5
      0            0          5/17

>> y=L\b
```

```
y =
       4
    17/5
     -4

>> x=U\y

x =
    138/5
    -89/5
    -68/5
```

The eigenvalues of a matrix and its eigenvectors can be easily found using the function `[v,d]=eig(A)`.

Example 10.23.

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

>> [v,d]=eig(A)

v =
    1898/2653    -605/1154     594/2549
   -430/633     -929/1224    -709/900
    145/887     -751/1945    3904/6847


d =
    130/1987        0            0
       0          771/83         0
       0            0         1007/612
```

where the columns of `v` are the eigenvectors and the diagonal elements of `d` are the corresponding eigenvalues.

The characteristic polynomial of a matrix `A`, $p(\lambda) = det(\mathtt{A} - \lambda I)$, can be found using the function `poly(A)`, which returns a row vector with the coefficients of $p(\lambda)$. Recall, that the roots of the characteristic polynomial are the eigenvalues of `A`.

Example 10.24.

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

A =
      1            1
      2            2

>> [v,d]=eig(A)
```

```
v =
    -985/1393   -1292/2889
     985/1393   -2584/2889

d =
      0            0
      0            3

>> poly (A)

ans =
      1           -3            0
```

where it is easily seen that the roots of the characteristic polynomial

$$p(\lambda) = \lambda^2 - 3\lambda$$

are the eigenvalues 0 and 3.

10.5.4 Polynomial Functions

Recall that in MATLAB a polynomial is represented by the vector of its coefficients. For example, the polynomial $x^2 + 2x - 3$ is expressed by the vector `[1 2 -3]`. MATLAB contains a set of built-in functions which allow to manipulate polynomials easily. Some of these functions are listed in the table below.

Polynomial Functions		
conv	→	Convolution and polynomial multiplication
deconv	→	Deconvolution and polynomial division
poly	→	Polynomial with specified roots
polyder	→	Polynomial derivative
polyfit	→	Polynomial curve fitting
polyint	→	Analytic polynomial integration
polyval	→	Polynomial evaluation
polyvalm	→	Matrix polynomial evaluation
roots	→	Polynomial roots

Mentioned in the previous subsection function `poly(v)`, when applied to a vector `v`, produces the vector of the coefficients of the polynomial whose roots are the elements of `v`. Below we give examples of using this and other polynomial functions.

Example 10.25. The elements of the vector `r=[1 3]` are the roots of the polynomial $c(x) = (x-1)(x-3) = x^2 - 4x + 3$ (represented by vector `c`):

```
>> r=[1 3];
>> c=poly(r)

c =
   1    -4     3
```

Given polynomials $c(x) = x^2 - 4x + 3$ and $d(x) = x^2 - 2x - 1$ (corresponding to vector `d`), their product is the polynomial $p(x) = c(x) \cdot d(x) = x^4 - 6x^3 + 10x^2 - 2x + 3$, corresponding to the vector `p`:

```
>> d=[1 -2 -1];
>> p = conv(c,d)

p =
     1    -6    10    -2    -3
```

Then $h(x) = p(x)/d(x) = c(x)$:

```
>> h=deconv(p,d)

h =
     1    -4     3
```

The derivative of $p(x)$ is $p'(x) = 4x^3 - 18x^2 + 20x - 2$:

```
>> polyder(p)

ans =
     4   -18    20    -2
```

The value of $c(x)$ at $x = 1$ is $c(1) = 0$:

```
>> polyval(c,1)

ans =
     0
```

10.6 Programming in MATLAB

Probably one of the most appealing features of MATLAB is its programming capabilities. All the classical programming techniques can be used, which when combined with the mathematical capabilities of the package results in a very effective tool for implementing and testing algorithms.

10.6.1 Relational Operators

The relational operators used by MATLAB are

```
==       equals
~=       not equals
<        less than
>        greater than
<=       less or equal to
>=       greater or equal to
```

and the value of a relation can be either true (1) or false (0).

Example 10.26.

```
>> v = [2 3 4 5];
>> x = [2 3 6 7];
>> relation = v == x

relation =

     1     1     0     0

>> relation = v < x

relation =

     0     0     1     1
```

Here the components of vector `relation` show if the specified relation is true for the corresponding elements of vectors `v` and `x`.

10.6.2 Loops and `if` statements

`For, while` loops, and `if` statements in MATLAB operate similarly to those in other programming languages.

for

The general form of a `for` statement is:

```
for {variable = expression}
   {statements}
end
```

The columns of the matrix represented by the expression are stored one at a time in the variable, and the statements are executed. A frequently used form of a for loop is

```
for i=1:n
   {statements}
end
```

although any other vector, or even matrix can be used instead of `1:n`. A for loop in the above form will repeat the set of statements contained inside exactly `n` times, each time increasing the value of `i` by 1, and then terminate. The following are examples of using `for` loop in MATLAB.

Example 10.27. This loop sums up all integers between 1 and 10:

```
>> n = 10;
>> s = 0;
>> for i = 1:n
     s = s + i;
  end
>> s

s =
    55
```

Below we show how `for` loops are used to sum up all the elements of a matrix.

```
>> A=[1:10; 2:11];
>> [m, n] = size(A);
>> sumA = 0 ;
>> for i = 1 : m
     for j = 1 : n
     sumA = sumA + A(i,j);
     end
   end
>> sumA

sumA =
      120
```

while

The syntax of `while` loop is

```
while {relation}
   {statements}
end
```

The statements contained in this loop are repeated as long as the relation in the first line remains true. Note that some of the variables participating in this relation should be modified inside the loop, otherwise it may never terminate.

if

The general form of the `if` statement is

```
if {relation}
   {statements}
elseif {relation}
   {statements}
else
   {statements}
end
```

The statements will be performed only if the relation is true.

Example 10.28. To sum the elements above, below and on the diagonal of a matrix `A` in three different sums we can use the following sequence of MATLAB statements:

```
>> [m, n] = size(A);
>> Usum = 0;
>> Lsum = 0;
>> Dsum = 0;
>> for i = 1 : m
     for j = 1 : n
       if i < j
  Usum = Usum + A(i, j) ;
```

```
      elseif j < i
  Lsum = Lsum + A(i, j) ;
      else
         Dsum = Dsum + A(i, j) ;
      end
    end
  end
```

where `Usum` is the sum of the elements of the upper triangular part, `Lsum` - of the lower triangular part, and `Dsum` is the sum of the diagonal elements of `A`, respectively.

10.6.3 Timing

One of the most important performance measures used to estimate the efficiency of a developed algorithm, is time required for an algorithm to perform a certain job. In MATLAB, the function `etime` in conjunction with the function `clock` is used to record the elapsed time:

```
tstart = clock;
     {statements}
tend = clock;
totaltime = etime(tend,tstart);
```

and `totaltime` will be the time needed for the program to execute the `statements`.

10.7 M-files

An M-file is a MATLAB-executable file that consists of a sequence of statements and commands. An M-file can be created in any text editor, but it should be saved in a diskfile with the extension `.m`. There are two types of M-files, *scripts* and *functions*. A script is nothing else but a file containing series of statements.

Example 10.29. Suppose that we created the following script file called `script1.m`.

```
Q=[1 2; 3 4];
determinantQ = det(Q)
traceQ = trace(Q)
```

Then with entering

```
>> script1
```

(which is the file's name without the extension `.m`) in the MATLAB command window, the script is automatically loaded and its statements and commands are executed:

```
>> script1

determinantQ =
              -2

traceQ =
        5
```

The following file, `script2.m`, works interactively, i.e. it requires a user's input to continue computations.

```
Q=input('input a square matrix: ');
disp(['the determinant of ' mat2str(Q) ' is ' num2str(det(Q))]);
disp(['its trace is equal to ' num2str(trace(Q))]);
```

Here the functions `input`, `mat2str` and `num2str` are used to read user's input, and to convert the matrix and a number to a string, respectively. Function `disp` is used to display a vector of elements, each of which is a string in this case. When executed, the script asks the user to input a square matrix:

```
>> script2
input a square matrix:
```

and when a square matrix is entered, it outputs the determinant and the trace of this matrix in the following way:

```
input a square matrix: [1 2; 3 4]
the determinant of [1 2;3 4] is -2
its trace is equal to 5
```

A function file is created in a similar way as scripts, the only difference is that a function also has input arguments. The first line of a function file is usually in the form

```
function {output arguments} = {functionname}({input arguments})
```

It declares the function name, input and output arguments. A function with the first line as above should be saved in a file with the name `functionname.m` (corresponding to the function name in the starting line). To execute a function, we type the function name followed by input arguments in the parenthesis, in exactly the same way as we did it for built-in functions.

Example 10.30. Suppose we want to write a function, which takes two vectors and multiplies them together. We create the following M-file:

```
function  H = mul(v,x)
% v, x : vectors of the same dimension
% the function mul(v,x) multiplies them together
H = v'*x ;
```

Then we save this file as `mul.m`. To run it, in the MATLAB command window we first define two vectors `v` and `x` of the same dimension, and then enter `w = mul(v,x)`. Then the variable `w` will be assigned the result of the multiplication of vectors `v, x`:

```
>> v=[1; 1; 2];
>> x=[2; -1; 0];
>> w = mul(v,x)

w =
   1
```

The % symbol is used for comments; the part of the line after the % sign is ignored by MATLAB. The first few lines of comments in the M-file are used in the on-line help facility. For example, if we enter `help mul` in the command widnow, MATLAB will reply with

```
v, x : vectors of the same dimension
the function mul(v,x) multiplies them together
```

It is recommended to include comments in all user-defined functions.

In the next example we demonstrate some more features of built-in functions. Given a scalar function $f(x)$, and two vectors x and y of the same dimension, we want to write a function which would output the values of $f(x') \cdot f(y)$ and $f(x' \cdot y)$ (recall, that a scalar function acts entrywise on vectors). First of all, our function should have two output arguments. We will use the function `nargin` ("number of input arguments") to make sure that the user inputs exactly three arguments. We will also use the function `feval`, which executes the function specified by a string, for example, if f='sin' then `feval(f,3.1415)` is the same as `sin(3.1415)`.

Example 10.31. The following function is saved in the disk file called `mulf.m`.

```
function [prodf, fprod] = mulf(func,x,y)
% f is a scalar function
% x, y : vectors of same dimension
% prodf = f(x')*f(y)
% fprod = f(x'*y)

if nargin ~= 3 | size(x)~=size(y)
   error('Please check your input')
end

prodf = feval(func,x')*feval(func,y);
fprod = feval(func,x'*y);
```

Now, suppose that we want to apply this function for $f(x) = x^2$, $x = (1, 2, 3)'$, and $y = (3, 2, 1)'$. First, we need to create the following user-defined function for $f(x)$ and save it as `f.m`.

```
function f=f(x)
% Given x, f(x) returns the square of x.
f=x.^2;
```

Then we input the values for the vectors x and y:

```
>> x = [1; 2; 3]; y = [3; 2; 1];
```

Finally, to see both output arguments we type

```
>> [pf, fp] = mulf('f',x,y)
```

in the MATLAB command window, giving

```
pf =
    34

fp =
    100
```

10.8 Graphics

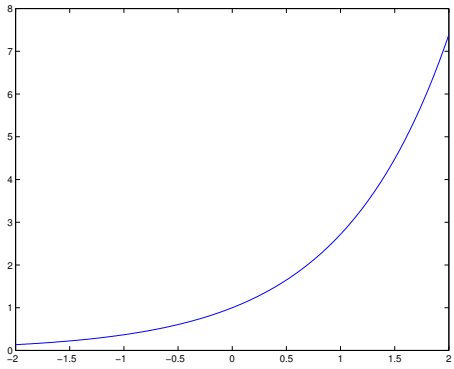
MATLAB has excellent visualization capabilities: it is able to produce planar plots and curves, three-dimensional plots, curves, and mesh surfaces, etc. In this section we will introduce some of the basic MATLAB graphics features.

10.8.1 Planar plots

Planar plots are created using the command `plot`.

Example 10.32. The following opens a graphics window and draws the graph of the exponential function over the interval -2 to 2:

```
>> x=-2:0.01:2;
>> y=exp(x);
>> plot(x,y)
```



The vector `x` represents the interval over which the plot is built; in this example we use a partition of $[-2, 2]$ with meshsize 0.01. The vector `y` contains the values of the function $\exp(x)$ in the points given by `x`.

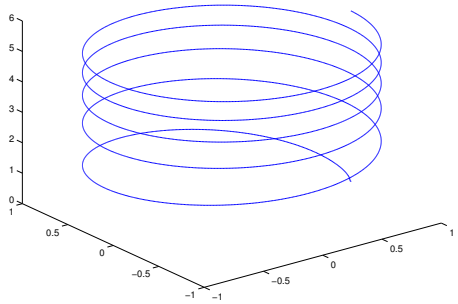
10.8.2 Three-dimensional (3-D) plots

Line plots

3-D line plots are built using the command `plot3`, which operates similarly to `plot` in two dimensions. Namely, given three vectors `x, y` and `z` of the same length, `plot3(x,y,z)` builds a plot of the piecewise linear curve connecting the points with coordinates defined by the corresponding components of `x, y` and `z`. In the example below we define `x, y` and `z` parametrically, using vector `t`.

Example 10.33.

```
>> t=0:0.01:10*pi;
>> x=cos(t);
>> y=sin(t);
>> z=sqrt(t);
>> plot3(x,y,z)
```



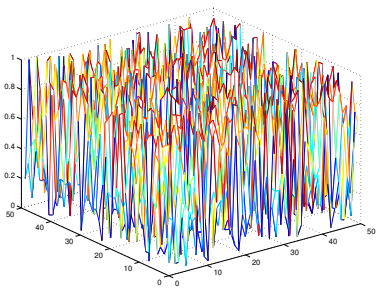
Mesh and surface plots

3-D mesh plots are created using the command `mesh`. When an $m \times n$ matrix Z is used as a single argument (mesh(Z)), then the mesh surface uses the values of `Z` as z-coordinates of points defined over a geometrically rectangular grid $\{1, 2, \ldots, m\} \times \{1, 2, \ldots, n\}$ in the $x - y$ plane. Similarly, 3-D colored surfaces are created using the command `surf`.

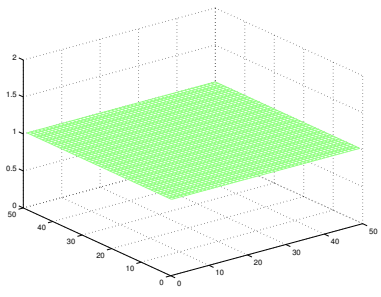
To draw a 3-D graph of a function of two variables $f(x, y)$ over a rectangle, we first use the function `[X,Y] = meshgrid(x,y)` to transforms the domain specified by vectors `x` and `y` into matrices `X` and `Y` which are used for the evaluation of $f(x, y)$. In these matrices, the rows of `X` are copies of the vector `x` and the columns of `Y` are copies of the vector `y`. The function $f(x, y)$ is then evaluated entrywise over the matrices `X` and `Y`, producing the matrix `Z` of the function values, to which `mesh(Z)` or `surf(Z)` can be applied.

Example 10.34. Below we draw the mesh surfaces of a random matrix and the matrix consisting of all ones, both of dimension 50×50 :

```
>> mesh(rand(50))
```

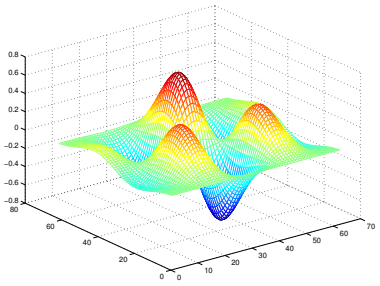


```
>> mesh(ones(50))
```

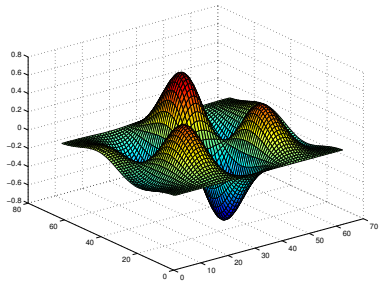


A graph of $f(x, y) = \cos(2x)\sin(y)e^{\frac{-x^2-y^2}{5}}$ over the rectangle $[-\pi, \pi] \times [-\pi, \pi]$ can be drawn as follows.

```
>> t=-pi:0.1:pi;
>> [X,Y]=meshgrid(t,t);
>> Z=cos(2*X).*sin(Y).*...
exp((-X.^2-Y.^2)/5);
>> mesh(Z)
```



```
>> t=-pi:0.1:pi;
>> [X,Y]=meshgrid(t,t);
>> Z=cos(2*X).*sin(Y).*...
exp((-X.^2-Y.^2)/5);
>> surf(Z)
```



10.9 On-line Help in MATLAB

Selected topics

Matlab Help facility is a very convenient tool for getting familiar with MATLAB built-in functions and commands. By entering "help" you will obtain the list of MATLAB Help topics. The following is a list of some of these topics (toolboxes are optional and are not necessary installed).

```
general       -  General purpose commands.
ops           -  Operators and special characters.
lang          -  Programming language constructs.
elmat         -  Elementary matrices and matrix manipulation.
elfun         -  Elementary math functions.
specfun       -  Specialized math functions.
matfun        -  Matrix functions - numerical linear algebra.
datafun       -  Data analysis and Fourier transforms.
polyfun       -  Interpolation and polynomials.
funfun        -  Function functions and ODE solvers.
sparfun       -  Sparse matrices.
graph2d       -  Two dimensional graphs.
graph3d       -  Three dimensional graphs.
specgraph     -  Specialized graphs.
graphics      -  Handle Graphics.
uitools       -  Graphical user interface tools.
strfun        -  Character strings.
iofun         -  File input/output.
timefun       -  Time and dates.
datatypes     -  Data types and structures.
demos         -  Examples and demonstrations.
optim         -  Optimization Toolbox.
signal        -  Signal Processing Toolbox.
stats         -  Statistics Toolbox.
symbolic      -  Symbolic Math Toolbox.
tour          -  MATLAB Tour
```

For more help on a topic, you can type "`help topic`" in the Matlab command window. This will give a list of commands and functions for the given topic. Then again, you can enter "help function_name" to get a detailed description of how to use the function of interest.