An Introduction to MATLAB

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*.

Matrix Basics

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; two rows are separated by semicolon or by starting a new line.

Example1

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

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

Example2

```
>> 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.

MATRIX BASICS

Matrix Subscripts

- For an *m x n* matrix B and given 1 <= i <= m and 1 <= j <= n, the expression B(i ,j) (called subscript) refers to the element in the i^{th} row and the j^{th} column of B.
- Given a vector c, a single subscript c(i) represents i^{th} element of the vector.

Example

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

>> B=[2 7 4; 3 5 9];
>> B(1,2)
ans = 7
>>B(2,3)
ans =  9
```

Matrix Size

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

```
>> c = [2 0 0 1; 8 7 3 2];
>> [m, n] = size(c)
        m  = 2
        n  = 4
>> n = length(c)
        n = 4
```

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; : : : ; a + kh]

Example1

```
>> 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.

Example2

```
>> 1:8
        ans = 1     2     3     4     5     6     7     8
```

Colons can be used to construct not only vectors, but matrices as well.

Matrix Operations

The basic matrix operations are the following:

Operator	Meaning
+	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 denotes the transpose of the matrix or a vector.

Example1

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.

Example2

Consider the following system of equations:

$$\mathbf{x_1 + 4x_2 + 3x_3 = 12}$$
$$\mathbf{-x_1 - 2x_2 = -12}$$
$$\mathbf{2x_1 + 2x_2 + 3x_3 = 8}$$

To solve the previous system of equations using Matlab, we must put it in the following form:

$$\begin{pmatrix} 1 & 4 & 3 \\ -1 & -2 & 0 \\ 2 & 2 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 12 \\ -12 \\ 8 \end{pmatrix}$$

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
```

Remember that $A\backslash b = A^{-1} * b$

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

Example

```
>> 3/2
        ans = 1.5000
>> 3\2
        ans = 0.6667
```

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

Example

```
>> 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).

Special Matrix - Building Functions

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

Special Matrix - Building Functions

Function	**Result**
A = eye(m,n)	*m* x *n* Identity matrix
A = zeros(m,n)	*m* x *n* Zero matrix
A = ones(m,n)	*m* x *n* Matrix with ones
A = rand(m,n)	*m* x *n* Uniform random elements matrix
A = randn(m,n)	*m* x *n* Normal random elements matrix

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

Example1

```
>> A = rand(3,2)
A =
    0.2190    0.6793
    0.0470    0.9347
    0.6789    0.3835
```

creates a 3x2 matrix of uniformly distributed on (0; 1) random elements

Example2

```
>> B=eye(3)+ones(3)
B =
     2     1     1
     1     2     1
     1     1     2
```

Example3

```
>> C=zeros(2,3)
C =
     0     0     0
     0     0     0
```

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.

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 edit" 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.

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

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

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.

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

Function in Matlab	**Meaning**	**Function in Matlab**	**Meaning**
sin(x)	sin(x)	**asin(x)**	arcsin(x)
cos(x)	cos(x)	**acos(x)**	arccos(x)
tan(x)	tan(x)	**atan(x)**	arctan(x)
sinh(x)	sinh(x)	**asinh(x)**	$\sinh^{-1}(x)$
cosh(x)	cosh(x)	**acosh(x)**	$\cosh^{-1}(x)$
tanh(x)	tanh(x)	**atanh(x)**	$\tanh^{-1}(x)$
exp(x)	e^x	**ceil(x)**	$\lceil x \rceil$
log(x)	ln(x)	**floor(x)**	$\lfloor x \rfloor$
log10(x)	$\log_{10} x$	**round(x)**	rounding (nearest)
abs(x)	|x|	**fix(x)**	round towards zero
sqrt(x)	$\sqrt{x}$	**rem(x, y)**	remainder after division

Example

By definition, $\sinh(x) = \dfrac{e^x - e^{-x}}{2}$

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

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(x)	Average or mean value
median(x)	Median value
std(x)	Standard deviation
all(x)	True (1) if all elements of a vector are nonzero
any(x)	True (1) if any element of a vector is nonzero

Example

By definition,

$$mean(x) = \frac{\sum_{i=1}^{n} x_i}{n} \qquad std(x) = \sqrt{\frac{\sum_{i=1}^{n} x_i^2 - n * mean(x)^2}{n-1}}$$

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

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
```

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:

Function	Meaning
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

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 manipulating polynomials easily. Some of these functions are listed in the table below.

Function	Meaning
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

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
$p(x) = c(x) * d(x) = x^4 – 6x^3 + 10x^2 – 2x + 3$, corresponding to the vector p:

```
>>  c = [1     -4      3]
>> 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 $p'(x) = 4x^3 - 18x^2 + 20x - 2$:

```
>> p = 1   -6          10      -2      -3
>> polyder(p)
ans =4          -18     20      -2
```

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

```
>> polyval(c,1)
ans = 0
```

The polyfit function

POLYFIT Fit polynomial to data. POLYFIT(X, Y, N) finds the coefficients of a polynomial P(X) of degree N that fits the data, P(X (I)) ~= Y (I), in a least-squares sense.

Example:

```
x = [1    2     3     4     5];
y = [5    7     9    11   13];
polyfit(x,y,1) % finds the coefficients of a polynomial P(X) of
                % degree 1 that fits the data, P(X(I))~=Y(I), in a
                %  least-squares sense.
```

The answer of the third statement will be as follows:

```
2.0000    3.0000
```

This means that the relation between *x* and *y* can be approximated as follows:

```
y = 2*x + 3
```

The polyval function

Y = POLYVAL(P, X), when P is a vector of length N+1 whose elements are the coefficients of a polynomial, is the value of the polynomial evaluated at X.

Y = P(1)*X^N + P(2)*X^(N-1) + ... + P(N)*X + P(N+1)

Example1

```
p = [1 2 3];
y = polyval(p,2);
```

This means that $y = 3*2^0 + 2*2^1 + 1*2^2 = 3+4+4 = 11$

Example2

```
p = [5 1 4 -1];
y = polyval(p,3);
```

This means that $y = -1*3^0 + 4*3^1 + 1*3^2 + 5*3^3 = 155$

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 disk file 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

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

```
Q = [1 2; 3 4];
[a, b] = size(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
a = 2
b = 2
```

User-Defined functions

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} = {function name}({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 function_name.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

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.

Exercise

Write a function called average() that takes a matrix as a parameter and returns the average of its elements.

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.

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

```
>> 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.

Loops and if statements

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

The for statement

The general form of a for statement is:

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

Example

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
```

Exercise

The Fibonacci sequence is a sequence of numbers as follows:-

1, 1, 2, 3, 5, 8, 13, 21, …

The first two elements are defined to be 1. Each of other elements is the sum of its two predecessors .Write a program that accepts an integer value *n* from the user then calculates the first *n* elements of the Fibonacci sequence and stores them in a one-dimensional array. The program must then print the result

The output of the program must be as shown in the following samples:-

Enter the number of terms required in the Fibonacci sequence: 10
1 1 2 3 5 8 13 21 34 55
89 144

Enter the number of terms required in the Fibonacci sequence: 15
1 1 2 3 5 8 13 21 34 55
89 144 233 377 610 987 1597

Exercise

The value of e^x can be calculated by the expression of the Taylor's series:

$$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^3}{3!} + ... + \frac{x^n}{n!}$$

Write a Matlab program that reads x, n and then calculates the value of e^x, use a function called fact() that calculates the factorial of a given number.

The while statement

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.

The if statement

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

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.

Integrations

Matlab provides four functions for computing an integral:

quad, quadl, dblquad and triplequad.

The quad function

QUAD numerically evaluates integral, adaptive Simpson quadrate.

Q = QUAD (FUN, A, B) tries to approximate the integral of function FUN from A to B to within an error of 1.e-6 using recursive adaptive Simpson quadrate. The function Y = FUN(X) should accept a vector argument X and return a vector result Y, the integrand evaluated at each element of X.

Q = QUAD (FUN, A, B, TOL) uses an absolute error tolerance of TOL instead of the default, which is 1.e-6. Larger values of TOL result in fewer function evaluations and faster computation, but less accurate results. The QUAD function in MATLAB 5.3 used a less reliable algorithm and a default tolerance of 1.e-3.

Note
Use array operators .*, ./ and .^ in the definition of FUN so that it can be evaluated with a vector argument.

Example

To calculate the following integral:

$$\int_{x=1}^{x=2} x^2 dx$$

We can write the following program:

First write the following .m file to declare the function:

```
function y = f(x)
y = x.^2;
```

To calculate the integration, we write the following statement:

```
Q = quad(@f,1,2);
```

Note the use of the symbol @ to indicate that f is a function.

Exercise1

Write a program to calculate the following integral:

$$\int_{x=-2}^{x=4} (x^2 + 2x + 1)dx$$

Exercise

Write a program to calculate the following integral:

$$\int_{x=a}^{x=b} (\sqrt{x})dx$$

The program must read the values of a and b from a file called 'data.txt' and the program must print the value of the integral to a file called 'results.txt'

The quadl function

The same as quad but QUADL but it uses adaptive Lobatto quadrate.

The dblquad function

DBLQUAD numerically evaluate double integral.

DBLQUAD (FUN, XMIN, XMAX, YMIN, YMAX) evaluates the double integral of FUN (X, Y) over the rectangle XMIN <= X <= XMAX, and YMIN <= Y <= YMAX.

FUN (X, Y) should accept a vector X and a scalar Y and return a vector of values of the integrand.

DBLQUAD (FUN, XMIN, XMAX, YMIN, YMAX, TOL) uses a tolerance TOL instead of the default, which is 1.e-6.

Example

To evaluate the integral:

$$\int_{y=0}^{y=\pi}\int_{x=\pi}^{x=2\pi}(y\sin(x)+x\cos(y))dxdy$$

First write the following .m file to declare the function:

```
function z =g(x,y)
z = y*sin(x)+x*cos(y);
```

Then write the following statement to evaluate the integral:

```
q=dblquad(@g,pi,2*pi,0,pi);
```

Exercise

Write a program to calculate the following integral:

$$\int_{y=y1}^{y=y2}\int_{x=x1}^{x=x2}(\sqrt{x^2+y^2})dxdy$$

The program must read the values of x1, x2, y1 and y2 from a file called 'data.txt' and the program must print the value of the integral to a file called 'results.txt'

The triplequad function

TRIPLEQUAD Numerically evaluate triple integral.
TRIPLEQUAD (FUN, XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX) evaluates the triple integral of FUN(X, Y, Z) over the three dimensional rectangular region XMIN <= X <= XMAX, YMIN <= Y <= YMAX, and ZMIN <= Z <= ZMAX.
FUN(X, Y, Z) should accept a vector X and scalar Y and Z and return a vector of values of the integrand.

TRIPLEQUAD(FUN,XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX,TOL) uses a tolerance TOL instead of the default, which is 1.e-6.

Example:

To evaluate the integral:

$$\int_{z=-1}^{z=1}\int_{y=0}^{y=1}\int_{x=0}^{x=\pi}(y\sin(x)+z\cos(y))dxdydz$$

First write the following .m file to declare the function:

```
function v =h(x,y,z)
v = y*sin(x)+z*cos(y);
```

Then write the following statement to evaluate the integral:

```
q=triplequad(@h,0,pi,0,1,-1,1)
```

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.

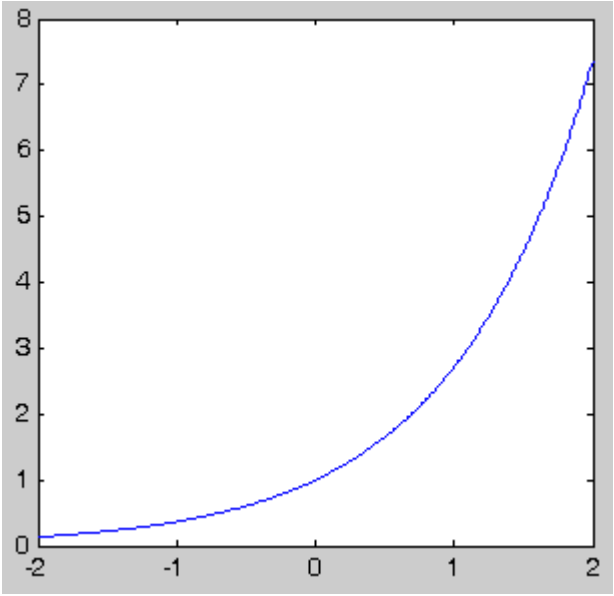
Planar plots

Planar plots are created using the command plot.

Example

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 [fi2; 2] with meshsize 0.01. The vector y contains the values of the function exp(x) in the points given by x.

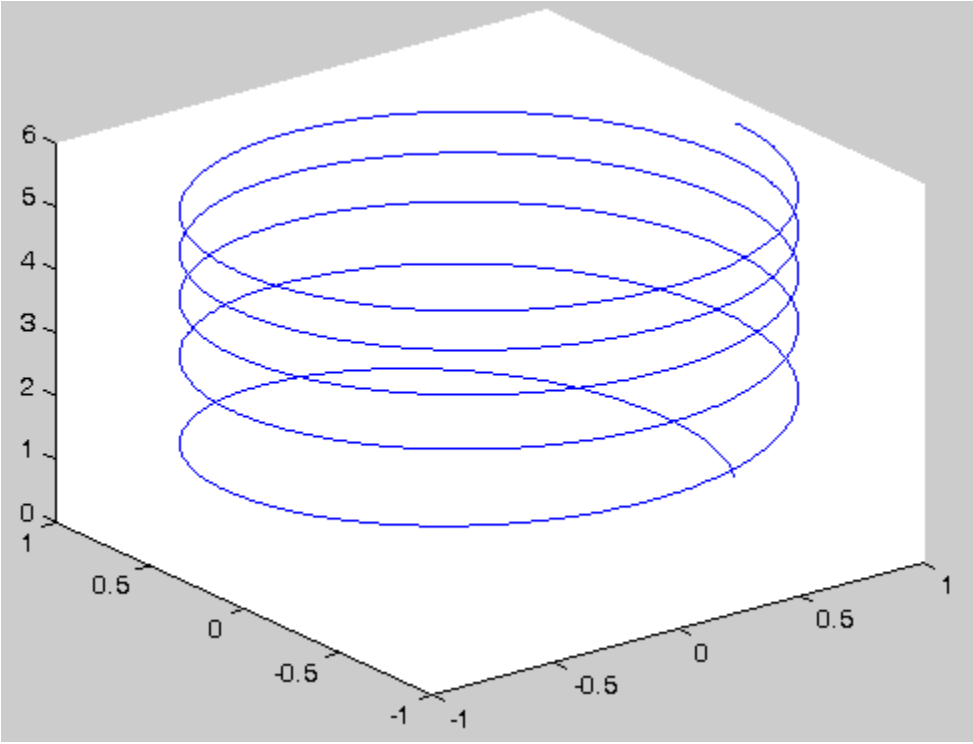
Three-dimensional (3-D) plots

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

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



Mesh and surface plots

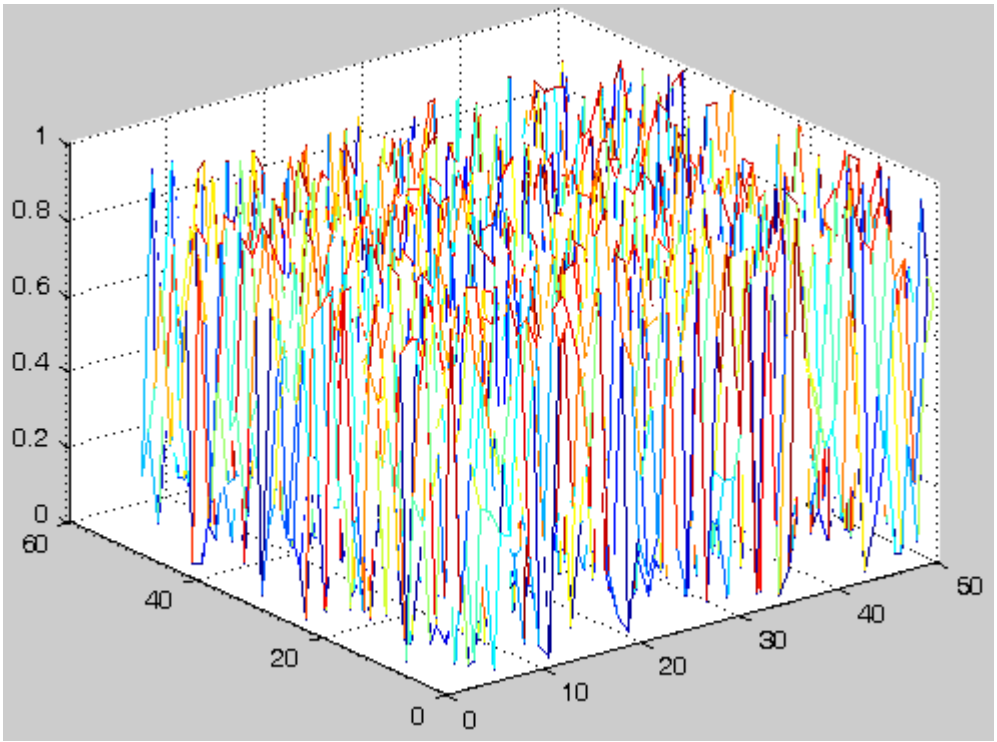
3-D mesh plots are created using the command mesh. When an mfin matrix Z is used as a single argument (mesh (Z)), then the mesh surface uses the values of Z as z-coordinates of points defined over a geometrically rectangular grid {1, 2, . . ,m} x {1, 2, . . ., n} in the x-y plane. Similarly, 3-D colored 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 entry-wise over the matrices X and Y, producing the matrix Z of the function values, to which mesh(Z) or surf(Z) can be applied.

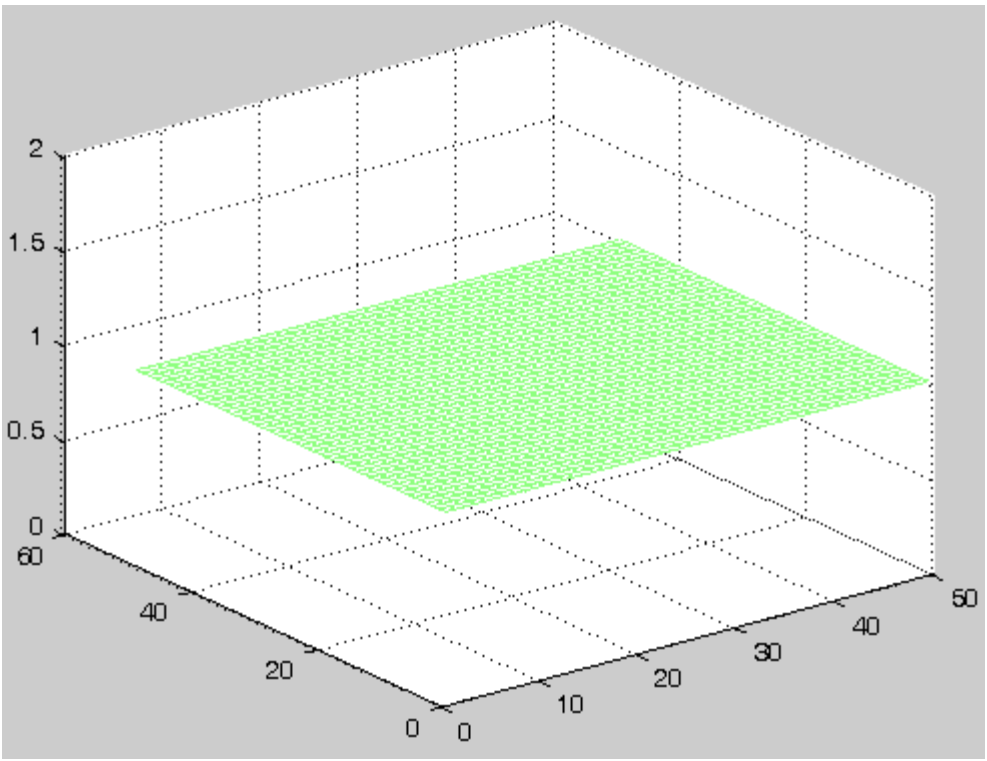
Example

Below we draw the mesh surfaces of a random matrix and the matrix consisting of all ones, both of dimension 50 x 50 :

>> mesh(rand(50))



>> mesh(ones(50))


2
1.5
1
0.5
0
60
40
20
0
0
10
20
30
40
50