Chapter 1
Short introduction to MATLAB

1.1 Introduction

MATLAB is a commercial software and a trademark of The MathWorks, Inc., USA. It is an integrated programming system, including graphical interfaces and a large number of specialized toolboxes. MATLAB is getting increasingly popular in all fields of science and engineering.

This chapter will provide some basic notions needed for the understanding of the remainder of the book. A deeper study of MATLAB can be obtained from many MATLAB books and the very useful help of MATLAB.

1.2 Matrices

Matrices are the fundamental object of MATLAB and are particularly important in this book. Matrices can be created in MATLAB in many ways, the simplest one obtained by the commands

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

Note the semi-colon at the end of each matrix line. We can also generate matrices by pre-defined functions, such as random matrices

```
>> rand(3)
ans =
    0.8147    0.9134    0.2785
    0.9058    0.6324    0.5469
    0.1270    0.0975    0.9575
```

A.J.M. Ferreira, *MATLAB Codes for Finite Element Analysis: Solids and Structures,* Solid Mechanics and Its Applications 157,

Rectangular matrices can be obtained by specification of the number of rows and columns, as in

```
>> rand(2,3)
ans =
    0.9649    0.9706    0.4854
    0.1576    0.9572    0.8003
```

1.3 Operating with matrices

We can add, subtract, multiply, and transpose matrices. For example, we can obtain a matrix $C = A + B$, by the following commands

```
>> a=rand(4)
a =
    0.2769    0.6948    0.4387    0.1869
    0.0462    0.3171    0.3816    0.4898
    0.0971    0.9502    0.7655    0.4456
    0.8235    0.0344    0.7952    0.6463
>> b=rand(4)
b =
    0.7094    0.6551    0.9597    0.7513
    0.7547    0.1626    0.3404    0.2551
    0.2760    0.1190    0.5853    0.5060
    0.6797    0.4984    0.2238    0.6991
>> c=a+b
c =
    0.9863    1.3499    1.3985    0.9381
    0.8009    0.4797    0.7219    0.7449
    0.3732    1.0692    1.3508    0.9515
    1.5032    0.5328    1.0190    1.3454
```

The matrices can be multiplied, for example $E = A * D$, as shown in the following example

```
>> d=rand(4,1)
d =
    0.8909
    0.9593
    0.5472
    0.1386
>> e=a*d
e =
    1.1792
    0.6220
```

```
    1.4787
    1.2914
```

The transpose of a matrix is given by the apostrophe, as

```
>> a=rand(3,2)
a =
    0.1493    0.2543
    0.2575    0.8143
    0.8407    0.2435
>> a'
ans =
    0.1493    0.2575    0.8407
    0.2543    0.8143    0.2435
```

1.4 Statements

Statements are operators, functions and variables, always producing a matrix which can be used later. Some examples of statements:

```
>> a=3
a =
     3

>> b=a*3
b =
     9

>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
```

If one wants to cancel the echo of the input, a semi-colon at the end of the statement suffices.

Important to mention that MATLAB is case-sensitive, variables a and A being different objects.

We can erase variables from the workspace by using clear, or clear all. A given object can be erased, such as clear A.

1.5 Matrix functions

Some useful matrix functions are given in table 1.1

Table 1.1 Some useful functions for matrices

eye	Identity matrix
zeros	A matrix of zeros
ones	A matrix of ones
diag	Creates or extract diagonals
rand	Random matrix

Some examples of such functions are given in the following commands (here we build matrices by blocks)

```
>> [eye(3),diag(eye(3)),rand(3)]
ans =
    1.0000         0         0    1.0000    0.9293    0.2511    0.3517
         0    1.0000         0    1.0000    0.3500    0.6160    0.8308
         0         0    1.0000    1.0000    0.1966    0.4733    0.5853
```

Another example of matrices built from blocks:

```
>> A=rand(3)
A =
    0.5497    0.7572    0.5678
    0.9172    0.7537    0.0759
    0.2858    0.3804    0.0540
>> B = [A, zeros(3,2); zeros(2,3), ones(2)]
B =
    0.5497    0.7572    0.5678         0         0
    0.9172    0.7537    0.0759         0         0
    0.2858    0.3804    0.0540         0         0
         0         0         0    1.0000    1.0000
         0         0         0    1.0000    1.0000
```

1.6 Conditionals, if and switch

Often a function needs to branch based on runtime conditions. MATLAB offers structures for this similar to those in most languages. Here is an example illustrating most of the features of if.

```
x=-1
if x==0
    disp('Bad input!')
elseif max(x) > 0
    y = x+1;
else
    y = x^2;
end
```

If there are many options, it may better to use switch instead. For instance:

```
switch units
    case 'length'
        disp('meters')
    case 'volume'
        disp('cubic meters')
    case 'time'
        disp('hours')
    otherwise
        disp('not interested')
end
```

1.7 Loops: for and while

Many programs require iteration, or repetitive execution of a block of statements. Again, MATLAB is similar to other languages here. This code for calculating the first 10 Fibonacci numbers illustrates the most common type of for/end loop:

```
>> f=[1 2]
f =
     1     2
>> for i=3:10;f(i)=f(i-1)+f(i-2);end;
>> f
f =
     1     2     3     5     8    13    21    34    55    89
```

It is sometimes necessary to repeat statements based on a condition rather than a fixed number of times. This is done with while.

```
>> x=10;while x > 1; x = x/2,end
x =
     5
x =
    2.5000
x =
    1.2500
x =
    0.6250
```

Other examples of for/end loops:

```
>> x = []; for i = 1:4, x=[x,i^2], end
```

```
x =
     1
x =
     1     4
x =
     1     4     9
x =
     1     4     9    16
```

and in inverse form

```
>> x = []; for i = 4:-1:1, x=[x,i^2], end
x =
    16
x =
    16     9
x =
    16     9     4
x =
    16     9     4     1
```

Note the initial values of `x = []` and the possibility of decreasing cycles.

1.8 Relations

Relations in MATLAB are shown in table 1.2.

Note the difference between '=' and logical equal '=='. The logical operators are given in table 1.3. The result if either 0 or 1, as in

```
>> 3<5,3>5,3==5
```

Table 1.2 Some relation operators

<	Less than
>	Greater than
<=	Less or equal than
>=	Greater or equal than
==	Equal to
~=	Not equal

Table 1.3 Logical operators

&	and
|	or
~	not

```
ans =
     1
ans =
     0
ans =
     0
```

The same is obtained for matrices, as in

```
>> a = rand(5), b = triu(a), a == b
a =
    0.1419    0.6557    0.7577    0.7060    0.8235
    0.4218    0.0357    0.7431    0.0318    0.6948
    0.9157    0.8491    0.3922    0.2769    0.3171
    0.7922    0.9340    0.6555    0.0462    0.9502
    0.9595    0.6787    0.1712    0.0971    0.0344
b =
    0.1419    0.6557    0.7577    0.7060    0.8235
         0    0.0357    0.7431    0.0318    0.6948
         0         0    0.3922    0.2769    0.3171
         0         0         0    0.0462    0.9502
         0         0         0         0    0.0344
ans =
     1     1     1     1     1
     0     1     1     1     1
     0     0     1     1     1
     0     0     0     1     1
     0     0     0     0     1
```

1.9 Scalar functions

Some MATLAB functions are applied to scalars only. Some of those functions are listed in table 1.4. Note that such functions can be applied to all elements of a vector or matrix, as in

```
>> a=rand(3,4)
a =
    0.4387    0.7952    0.4456    0.7547
```

Table 1.4 Scalar functions

sin	asin	exp	abs	round
cos	acos	log	sqrt	floor
tan	atan	rem	sign	ceil

```
    0.3816    0.1869    0.6463    0.2760
    0.7655    0.4898    0.7094    0.6797
>> b=sin(a)
b =
    0.4248    0.7140    0.4310    0.6851
    0.3724    0.1858    0.6022    0.2725
    0.6929    0.4704    0.6514    0.6286
>> c=sqrt(b)
c =
    0.6518    0.8450    0.6565    0.8277
    0.6102    0.4310    0.7760    0.5220
    0.8324    0.6859    0.8071    0.7928
```

1.10 Vector functions

Some MATLAB functions operate on vectors only, such as those illustrated in table 1.5.

Consider for example vector `X=1:10`. The sum, mean and maximum values are evaluated as

```
>> x=1:10
x =
     1     2     3     4     5     6     7     8     9    10
>> sum(x)
ans =
    55
>> mean(x)
ans =
    5.5000
>> max(x)
ans =
    10
```

Table 1.5 Vector functions

max	sum	median	any
min	prod	mean	all

1.11 Matrix functions

Some important matrix functions are listed in table 1.6.

In some cases such functions may use more than one output argument, as in

```
>> A=rand(3)
A =
    0.8147    0.9134    0.2785
    0.9058    0.6324    0.5469
    0.1270    0.0975    0.9575
>> y=eig(A)
y =
   -0.1879
    1.7527
    0.8399
```

where we wish to obtain the eigenvalues only, or in

```
>> [V,D]=eig(A)
V =
    0.6752   -0.7134   -0.5420
   -0.7375   -0.6727   -0.2587
   -0.0120   -0.1964    0.7996
D =
   -0.1879         0         0
         0    1.7527         0
         0         0    0.8399
```

where we obtain the eigenvectors and the eigenvalues of matrix A.

Table 1.6 Matrix functions

eig	Eigenvalues and eigenvectors
chol	Choleski factorization
inv	Inverse
lu	LU decomposition
qr	QR factorization
schur	Schur decomposition
poly	Characteristic polynomial
det	Determinant
size	Size of a matrix
norm	1-norm, 2-norm, F-norm, ∞-norm
cond	Conditioning number of 2-norm
rank	Rank of a matrix

1.12 Submatrix

In MATLAB it is possible to manipulate matrices in order to make code more compact or more efficient. For example, using the colon we can generate vectors, as in

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

or using increments

```
>> x=1.2:0.5:3.7
x =
    1.2000    1.7000    2.2000    2.7000    3.2000    3.7000
```

This sort of vectorization programming is quite efficient, no for/end cycles are used. This efficiency can be seen in the generation of a table of sines,

```
>> x=0:pi/2:2*pi
x =
         0    1.5708    3.1416    4.7124    6.2832
>> b=sin(x)
b =
         0    1.0000    0.0000   -1.0000   -0.0000
>> [x' b']
ans =
         0         0
    1.5708    1.0000
    3.1416    0.0000
    4.7124   -1.0000
    6.2832   -0.0000
```

The colon can also be used to access one or more elements from a matrix, where each dimension is given a single index or vector of indices. A block is then extracted from the matrix, as illustrated next.

```
>> a=rand(3,4)
a =
    0.6551    0.4984    0.5853    0.2551
    0.1626    0.9597    0.2238    0.5060
    0.1190    0.3404    0.7513    0.6991
>> a(2,3)
ans =
    0.2238
```

```
>> a(1:2,2:3)
ans =
    0.4984    0.5853
    0.9597    0.2238
>> a(1,end)
ans =
    0.2551
>> a(1,:)
ans =
    0.6551    0.4984    0.5853    0.2551
>> a(:,3)
ans =
    0.5853
    0.2238
    0.7513
```

It is interesting to note that arrays are stored linearly in memory, from the first dimension, second, and so on. So we can in fact access vectors by a single index, as show below.

```
>> a=[1 2 3;4 5 6; 9 8 7]
a =
     1     2     3
     4     5     6
     9     8     7
>> a(3)
ans =
     9
>> a(7)
ans =
     3
>> a([1 2 3 4])
ans =
     1     4     9     2
>> a(:)
ans =
     1
     4
     9
     2
     5
     8
     3
     6
     7
```

Subscript referencing can also be used in both sides.

```
>> a
a =
     1     2     3
     4     5     6
     9     8     7
>> b
b =
     1     2     3
     4     5     6
>> b(1,:)=a(1,:)
b =
     1     2     3
     4     5     6
>> b(1,:)=a(2,:)
b =
     4     5     6
     4     5     6
>> b(:,2)=[]
b =
     4     6
     4     6
>> a(3,:)=0
a =
     1     2     3
     4     5     6
     0     0     0
>> b(3,1)=20
b =
     4     6
     4     6
    20     0
```

As you noted in the last example, we can insert one element in matrix B, and MATLAB automatically resizes the matrix.

1.13 Logical indexing

Logical indexing arise from logical relations, resulting in a logical array, with elements 0 or 1.

```
>> a
a =
     1     2     3
     4     5     6
     0     0     0
```

```
>> a>2
ans =
     0     0     1
     1     1     1
     0     0     0
```

Then we can use such array as a mask to modify the original matrix, as shown next.

```
>> a(ans)=20
a =
     1     2    20
    20    20    20
     0     0     0
```

This will be very useful in finite element calculations, particularly when imposing boundary conditions.

1.14 M-files, scripts and functions

A M-file is a plain text file with MATLAB commands, saved with extension .m. The M-files can be scripts of functions. By using the editor of MATLAB we can insert comments or statements and then save or compile the m-file. Note that the percent sign % represents a comment. No statement after this sign will be executed. Comments are quite useful for documenting the file.

M-files are useful when the number of statements is large, or when you want to execute it at a later stage, or frequently, or even to run it in background.

A simple example of a **script** is given below.

```
% program 1
% programmer: Antonio ferreira
% date: 2008.05.30
% purpose : show how M-files are built

% data: a - matrix of numbers; b: matrix with sines of a


a=rand(3,4);
b=sin(a);
```

Functions act like subroutines in fortran where a particular set of tasks is performed. A typical function is given below, where in the first line we should name the function and give the input parameters (m,n,p) in parenthesis and the output parameters (a,b,c) in square parenthesis.

```
function  [a,b,c] = antonio(m,n,p)
```

```
a = hilb(m);
b= magic(n);
c= eye(m,p);
```

We then call this function as

```
>> [a,b,c]=antonio(2,3,4)
```

producing

```
>> [a,b,c]=antonio(2,3,4)
a =
    1.0000    0.5000
    0.5000    0.3333
b =
     8     1     6
     3     5     7
     4     9     2
c =
     1     0     0     0
     0     1     0     0
```

It is possible to use only some output parameters.

```
>> [a,b]=antonio(2,3,4)
a =
    1.0000    0.5000
    0.5000    0.3333
b =
     8     1     6
     3     5     7
     4     9     2
```

1.15 Graphics

MATLAB allows you to produce graphics in a simple way, either 2D or 3D plots.

1.15.1 2D plots

Using the command `plot` we can produce simple 2D plots in a `figure`, using two vectors with x and y coordinates. A simple example

```
x = -4:.01:4;  y = sin(x);  plot(x,y)
```

producing the plot of figure 1.1.

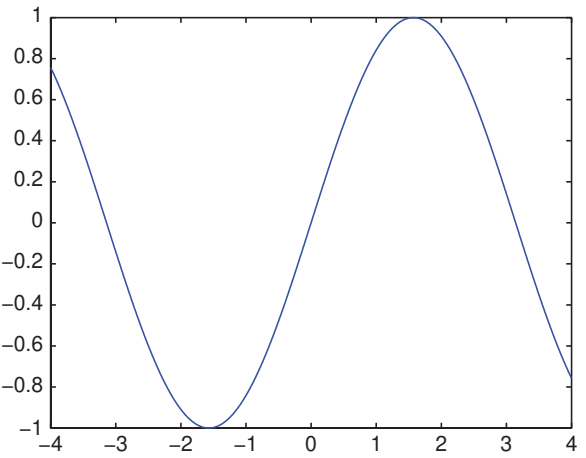


Fig. 1.1 2D plot of a sinus

Table 1.7 Some graphics commands

Title	Title
xlabel	x-axis legend
ylabel	y-axis legend
Axis([x_{min},x_{max},y_{min},y_{max}])	Sets limits to axis
Axis auto	Automatic limits
Axis square	Same scale for both axis
Axis equal	Same scale for both axis
Axis off	Removes scale
Axis on	Scales again

We can insert a title, legends, modify axes etc., as shown in table 1.7.

By using `hold on` we can produce several plots in the same figure. We can also modify colors of curves or points, as in

```
>> x=0:.01:2*pi; y1=sin(x); y2=sin(2*x); y3=sin(4*x);
>> plot(x,y1,'--',x,y2,':',x,y3,'+')
```

producing the plot of figure 1.2.

1.15.2 3D plots

As for 2D plots, we can produce 3D plots with `plot3` using x, y, and z vectors. For example

```
t=.01:.01:20*pi; x=cos(t); y=sin(t); z=t.^3; plot3(x,y,z)
```

produces the plot illustrated in figure 1.3.

The next statements produce the graphic illustrated in figure 1.4.

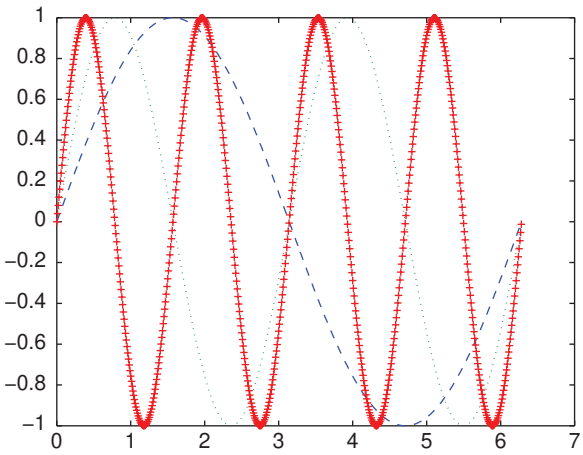


Fig. 1.2 Colors and markers

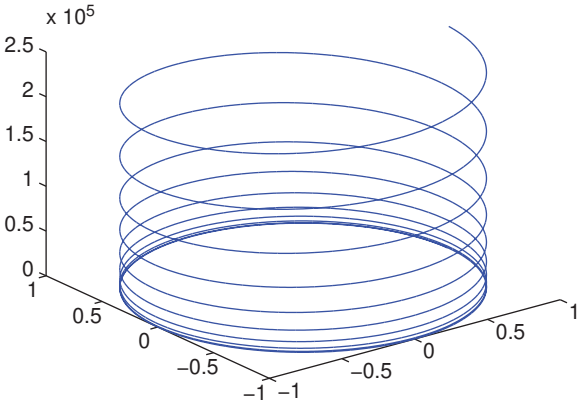


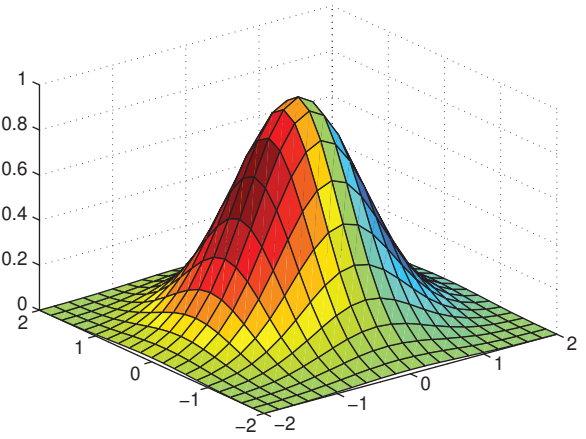
Fig. 1.3 3D plot

```
>> [xx,yy]=meshgrid(x,x);
>> z=exp(-xx.^2-yy.^2);
>> surf(xx,yy,z,gradient(z))
```

1.16 Linear algebra

In our finite element calculations we typically need to solve systems of equations, or obtain the eigenvalues of a matrix. MATLAB has a large number of functions for linear algebra. Only the most relevant for finite element analysis are here presented.

Fig. 1.4 Another 3D plot



Consider a linear system $AX = B$, where

```
>> a=rand(3)
a =
    0.8909    0.1386    0.8407
    0.9593    0.1493    0.2543
    0.5472    0.2575    0.8143
>> b=rand(3,1)
b =
    0.2435
    0.9293
    0.3500
```

The solution vector X can be easily evaluated by using the backslash command,

```
>> x=a\b
x =
    0.7837
    2.9335
   -1.0246
```

Consider two matrices (for example a stiffness matrix and a mass matrix), for which we wish to calculate the generalized eigenproblem.

```
>> a=rand(4)
a =
    0.1966    0.3517    0.9172    0.3804
    0.2511    0.8308    0.2858    0.5678
    0.6160    0.5853    0.7572    0.0759
    0.4733    0.5497    0.7537    0.0540
>> b=rand(4)
```