Chapter 1

Introduction to MATLAB

MATLAB is a computer software commonly used in both education and industry to solve a wide range of problems.

This chapter provides a brief introduction to MATLAB, and the tools and functions that help you to work with MATLAB variables and files. The programs were tested with version 7.4 (R2007a) of Matlab under a Kubuntu Linux shell, but most could be used as is with previous releases or other systems.

The MATLAB Environment

☆ To start MATLAB double-click on the MATLAB shortcut icon or type matlab & at the prompt (Unix/Linux). The MATLAB desktop opens. On the right side of the desktop you find the Command Window, where commands are entered at the prompt >>.

On the left side you will generally find the Launch Pad and Workspace windows, and the Command History and Current Directory windows. For all practical purposes we have in mind I recommend closing the Launch Pad, Workspace, and Command History windows, if opened. It is convenient to keep the Current Directory window opened to check for files you create and use in the Command Window.

Note that windows within the MATLAB desktop can be resized by dragging the separator bar(s). A typical MATLAB desktop is shown in Fig. 1.1.

★ To exit MATLAB do one of the following:

• Click on the close box ☒ at the top right of the MATLAB Desktop.

• Select File > Exit from the desktop File menu.

• Type quit or exit at the Command Window prompt >>.

Basics And Help

Commands are entered in the Command Window.
★ Basic operations are +, −, ∗, and /. The sequence

```matlab
>> a=2; b=3; a+b, a*b,
ans =
5
```

```matlab
ans =
6
```
defines variables \(a\) and \(b\) and assigns values 2 and 3, respectively, then computes the sum \(a + b\) and product \(ab\). Each command ends with \, (output is visible) or \; (output is suppressed). The last command on a line does not require a \,.

\* Standard functions can be invoked using their usual mathematical notations. For example

\[
\begin{align*}
\text{>> } & \theta = \frac{\pi}{5}; \\
\text{>> } & \cos(\theta)^2 + \sin(\theta)^2 \\
\text{ans } & = 1
\end{align*}
\]

verifies the trigonometric identity \(\sin^2 \theta + \cos^2 \theta = 1\) for \(\theta = \frac{\pi}{5}\). A list of elementary math functions can be obtained by typing \texttt{help elfun} in the Command window:

\[
\begin{align*}
\text{>> } & \text{help elfun} \\
\text{Elementary math functions.} \\
\text{Trigonometric.} \\
\text{sin } & \text{- Sine.} \\
\text{sind } & \text{- Sine of argument in degrees.} \\
\text{sinh } & \text{- Hyperbolic sine.} \\
\text{asin } & \text{- Inverse sine.} \\
\text{asind } & \text{- Inverse sine, result in degrees.} \\
\text{asinh } & \text{- Inverse hyperbolic sine.} \\
\text{cos } & \text{- Cosine.}
\end{align*}
\]
cosd - Cosine of argument in degrees.
cosh - Hyperbolic cosine.
acos - Inverse cosine.
acosh - Inverse hyperbolic cosine.
tan - Tangent.
tanh - Hyperbolic tangent.
atan - Inverse tangent.
atanh - Inverse hyperbolic tangent.
sec - Secant.
sech - Hyperbolic secant.
asec - Inverse secant.
asech - Inverse hyperbolic secant.
csc - Cosecant.
csch - Hyperbolic cosecant.
acsc - Inverse cosecant.
acsch - Inverse hyperbolic cosecant.
cot - Cotangent.
cotd - Cotangent of argument in degrees.
coth - Hyperbolic cotangent.
acot - Inverse cotangent.
acotd - Inverse cotangent, result in degrees.
acoth - Inverse hyperbolic cotangent.
hypot - Square root of sum of squares.

Exponential.

exp - Exponential.
expm1 - Compute exp(x)-1 accurately.
log - Natural logarithm.
log1p - Compute log(1+x) accurately.
log10 - Common (base 10) logarithm.
log2 - Base 2 logarithm and dissect floating point number.
pow2 - Base 2 power and scale floating point number.
realpow - Power that will error out on complex result.
reallog - Natural logarithm of real number.
realsqrt - Square root of number greater than or equal to zero.
sqrt - Square root.
nthroot - Real n-th root of real numbers.
nexppow2 - Next higher power of 2.

Complex.

abs - Absolute value.
angle - Phase angle.
complex - Construct complex data from real and imaginary parts.
conj - Complex conjugate.
imag - Complex imaginary part.
real - Complex real part.
unwrap - Unwrap phase angle.
isreal - True for real array.
cplxpair - Sort numbers into complex conjugate pairs.

Rounding and remainder.
fix - Round towards zero.
floor - Round towards minus infinity.
ceil - Round towards plus infinity.
round - Round towards nearest integer.
mod - Modulus (signed remainder after division).
rem - Remainder after division.
sign - Signum.

To obtain a description of the use of a particular function type help followed by the name of the function. For example:

```matlab
>> help cosh
COSH    Hyperbolic cosine.
COSH(X) is the hyperbolic cosine of the elements of X.

See also acosh.
```

Overloaded functions or methods (ones with the same name in other directories)
help sym/cosh.m

Reference page in Help browser
doc cosh

To get a list of other groups of MATLAB programs already available enter help:

```matlab
>> help
HELP topics:

matlab/general - General purpose commands.
matlab/ops - Operators and special characters.
matlab/lang - Programming language constructs.
matlab/elmat - Elementary matrices and matrix manipulation.
matlab/elfun - Elementary math functions.
matlab/specfun - Specialized math functions.
matlab/matfun - Matrix functions - numerical linear algebra.
matlab/datafun - Data analysis and Fourier transforms.
matlab/polyfun - Interpolation and polynomials.
matlab/funfun - Function functions and ODE solvers.
matlab/sparfun - Sparse matrices.
matlab/scribe - Annotation and Plot Editing.
matlab/graph2d - Two dimensional graphs.
matlab/graph3d - Three dimensional graphs.
matlab/specgraph - Specialized graphs.
matlab/graphics - Handle Graphics.
matlab/uitools - Graphical User Interface Tools.
matlab/strfun - Character strings.
matlab/imagesci - Image and scientific data input/output.
matlab/iofun - File input and output.
matlab/audiovideo - Audio and Video support.
```
matlab/timefun - Time and dates.
matlab/datatypes - Data types and structures.
matlab/verctrl - Version control.
matlab/codetools - Commands for creating and debugging code.
matlab/help - Help commands.
matlab/demos - Examples and demonstrations.
matlab/timeseries - Time series data visualization and exploration.
matlab/hds - (No table of contents file)
matlab/guide - Graphical User Interface Tools.
matlab/plottools - Graphical User Interface Tools.
toolbox/local - General preferences and configuration information.
shared/controllib - Control Library
simulink/simulink - Simulink
simulink/blocks - Simulink block library.
...

The actual list of topics depends on your system and the toolboxes installed on your machine.

Another way to obtain help is through the desktop Help menu, Help > MATLAB Help, or by connecting to the Mathworks web site at www.mathworks.com.

MATLAB is case-sensitive. For example

```matlab
>> theta=1e-3, Theta=2e-5; ratio=theta/Theta

theta =
   0.0010

Theta =
     2.0000e-005

ratio =
     50
```

The quantities Inf (∞) and NaN (Not a Number) also appear frequently. Compare

```matlab
>> c=1/0
Warning: Divide by zero.

c =
Inf
```

with

```matlab
>> d=0/0
Warning: Divide by zero.

d =
NaN
```
Plotting with MATLAB

★★ To plot a function you have to create two arrays (vectors): one containing the abscissae, the other the corresponding function values. Both arrays should have the same length. For example, consider plotting the function

\[ y = f(x) = \frac{x^2 - \sin(\pi x) + e^x}{x - 1} \]

for \(0 \leq x \leq 2\). First choose a sample of \(x\) values in this interval:

\[
\text{>> } \text{x}=[0,.1,.2,.3,.4,.5,.6,.7,.8,.9,1, \ldots \\
1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2] \\
\]

\[
x = \\
\text{Columns 1 through 4} \\
0 \quad 0.1000 \quad 0.2000 \quad 0.3000 \\
\text{Columns 5 through 8} \\
0.4000 \quad 0.5000 \quad 0.6000 \quad 0.7000 \\
\text{Columns 9 through 12} \\
0.8000 \quad 0.9000 \quad 1.0000 \quad 1.1000 \\
\text{Columns 13 through 16} \\
1.2000 \quad 1.3000 \quad 1.4000 \quad 1.5000 \\
\text{Columns 17 through 20} \\
1.6000 \quad 1.7000 \quad 1.8000 \quad 1.9000 \\
\text{Column 21} \\
2.00000 \\
\]

or simply

\[
\text{>> } \text{x}=0:.1:2 \\
x = \\
\text{Columns 1 through 4} \\
0 \quad 0.1000 \quad 0.2000 \quad 0.3000 \\
\text{Columns 5 through 8} \\
0.4000 \quad 0.5000 \quad 0.6000 \quad 0.7000 \\
\text{Columns 9 through 12} \\
\]
Try also

```matlab
>> x=linspace(0,2,21)
```

```
x =
Columns 1 through 4
     0     0.1000    0.2000    0.3000
Columns 5 through 8
     0.4000    0.5000    0.6000    0.7000
Columns 9 through 12
     0.8000    0.9000    1.0000    1.1000
Columns 13 through 16
     1.2000    1.3000    1.4000    1.5000
Columns 17 through 20
     1.6000    1.7000    1.8000    1.9000
Column 21
     2.0000
```

★ Note that an ellipsis \ldots was used to continue a command too long to fit in a single line.
★ The output for \texttt{x} can be suppressed (by adding ; at the end of the command) or condensed by entering \texttt{format compact}:

```matlab
>> format compact
>> x
```

```
Columns 1 through 4
     0     0.1000    0.2000    0.3000
Columns 5 through 8
     0.4000    0.5000    0.6000    0.7000
```
Introduction to MATLAB

From now on we shall use such format for all output.

To evaluate the function $f$ simultaneously at all the values contained in $x$, type

```matlab
>> y=(x.^2-sin(pi.*x)+exp(x))./(x-1)
Warning: Divide by zero.
y =
Columns 1 through 4
  -1.0000  -0.8957  -0.8420  -0.9012
Columns 5 through 8
  -1.1679  -1.7974  -3.0777  -5.6491
Columns 9 through 12
  -11.3888 -29.6059    Inf   45.2318
Columns 13 through 16
  26.7395  20.5610  17.4156  15.4634
Columns 17 through 20
  14.1068  13.1042  12.3468  11.7832
Column 21
   11.3891
```

Note that the function becomes infinite at $x = 1$ (vertical asymptote). The array $y$ inherits the dimension of $x$, namely 1 (row) by 21 (columns). Note also the use of parentheses.

**IMPORTANT REMARK**

In the above example *, / and ~ are preceded by a dot . in order for the expression to be evaluated for each component (entry) of $x$. This is necessary to prevent MATLAB from interpreting these symbols as standard linear algebra symbols operating on arrays. Because the standard + and - operations on arrays already work componentwise, a dot is not necessary for + and -.

The command

```matlab
>> plot(x,y)
```

creates a Figure window and shows the function, see Fig. 1.2. The figure can be edited and manipulated using the Figure window menus and buttons. Alternately, properties of the figure can also be defined directly at the command line:

```matlab
>> x=0:.01:2;
>> y=(x.^2-sin(pi.*x)+exp(x))./(x-1);
Warning: Divide by zero.
>> plot(x,y,'r-','LineWidth',2);
>> axis([0,2,-10,20]); grid on;
>> title('f(x)=(x^2-sin(pi * x)+e^x)/(x-1)');
>> xlabel('x'); ylabel('y');
Figure 1.2: A Figure window

Figure 1.3: The function $y = f(x) = \frac{x^2 - \sin(\pi x) + e^x}{x-1}$. 

The number of \( x \)-values has been increased for a smoother curve (what is the new size of \( x \)?). The curve now appears wider and in red. The range of \( x \) and \( y \) values has been reset (always a good idea in the presence of vertical asymptotes). A title and labels have been added. The resulting new plot is shown in Fig. 1.3. For more options type \texttt{help plot} in the Command Window.

**Scripts and Functions**

★ Files containing MATLAB commands are called \texttt{m}-files and have a \texttt{.m} extension. They are two types:

1. A \texttt{script} is simply a collection of MATLAB commands gathered in a single file. The value of the data created in a script is still available in the Command Window after execution.
2. A \texttt{function} is similar to a script, but can accept and return arguments. Unless otherwise specified any variable inside a function is local to the function and not available in the Workspace. A function invariably starts with the command

\[
\text{function output = function name(input)}
\]

and should contain one or several commands defining the output.

Use a function when a group of commands needs to be evaluated multiple times.

★ To create a new script or function select the MATLAB desktop File menu \texttt{File > New > M-file}. In the MATLAB text editor window enter the commands as you would in the Command window. To save the file use the menu \texttt{File > Save or File > Save As...}, or the shortcut Save button ．

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{myplot.m}
\caption{The script \texttt{myplot.m} in the MATLAB Editor window.}
\end{figure}

★ Examples of script/function:

1. \texttt{script}

\begin{verbatim}
myplot.m
\end{verbatim}

\begin{verbatim}
x=0:.01:2; \hspace{1cm} \text{\% x-values}
y=(x.^2-sin(pi.*x)+exp(x))./(x-1); \hspace{1cm} \text{\% y-values}
plot(x,y,'r-','LineWidth',2); \hspace{1cm} \text{\% plot in red}
axis([0,2,-10,20]); grid on; \hspace{1cm} \text{\% set range and add grid}
title('f(x)=(x^2-sin(pi x)+e^x)/(x-1)'); \hspace{1cm} \text{\% add title}
xlabel('x'); ylabel('y'); \hspace{1cm} \text{\% add labels}
\end{verbatim}
### Introduction to MATLAB

2. **script+function** (two separate files)

**myplot2.m** (driver script)

```matlab
x=0:.01:2; % x-values
y=myfunction(x); % evaluate myfunction at x
plot(x,y,'r-','LineWidth',2); % plot in red
axis([0,2,-10,20]); grid on; % set range and add grid
title('f(x)=(x^2-sin(\pi x)+e^x)/(x-1)'); % add title
xlabel('x'); ylabel('y'); % add labels
```

**myfunction.m** (function)

```matlab
function y=myfunction(x) % defines function
y=(x.^2-sin(pi.*x)+exp(x))./(x-1); % y-values
```

3. **script+function** (one single file)

**myplot1.m** (driver script converted to function + function)

```matlab
function myplot1
x=0:.01:2; % x-values
y=feval(myfunction,x); % evaluate myfunction at x
plot(x,y,'r-','LineWidth',2); % plot in red
axis([0,2,-10,20]); grid on; % set range and add grid
title('f(x)=(x^2-sin(\pi x)+e^x)/(x-1)'); % add title
xlabel('x'); ylabel('y'); % add labels
%-----------------------------------------
function y=myfunction(x) % defines function
y=(x.^2-sin(pi.*x)+exp(x))./(x-1); % y-values
```

In case 2 **myfunction.m** can be used in any other `.m`-file (just as other predefined MATLAB functions). In case 3 **myfunction.m** can be used by any other function in the same `.m`-file (**myplot1.m**) only. Use 3 when dealing with a single project and 2 when a function is used by several projects.

★ It is convenient to add descriptive comments into the script file. Anything appearing after `%` on any given line is understood as a comment (in green in the MATLAB text editor).

★ To execute a script simply enter its name (without the .m extension) in the Command Window, e.g.,

```matlab
>> myplot;
```

in case 1,

```matlab
>> myplot2;
```

in case 2 and

```matlab
>> myplot1;
```

in case 3 above. The function **myfunction** can also be used independently if implemented in a separate file **myfunction.m**:

```matlab
>> x=2; y=myfunction(x)
y =
  11.3891
```
A script can be called from another script or function (in which case it is local to that function).
If any modification is made, the script or function can be re-executed by simply retyping the script or function name in the Command Window (or use the up-arrow on the keyboard to browse through past commands).

**IMPORTANT REMARK**
By default MATLAB saves files in the Current Directory (folder). To change directory use the Current Directory box on top of the MATLAB desktop.

★ A function file can contain a lot more than a simple evaluation of a function \( f(x) \) or \( f(t, y) \). But in simple cases \( f(x) \) or \( f(t, y) \) can simply be defined using the **inline** syntax. Compare

\[
\begin{align*}
&> \quad \ldots \\
&> \quad \text{slope} = \text{feval}(@f, 2, 1) \quad \% \text{note use of @} \quad \text{EXPLAIN!!!!}
\end{align*}
\]

\[
\text{slope} = 3
\]

where \( f.m \) is the file containing

\[
\begin{align*}
&\text{function dydt} = f(t, y) \\
&\text{dydt} = t^2 - y;
\end{align*}
\]

to

\[
\begin{align*}
&> \quad \ldots \\
&> \quad f = \text{inline}('t^2-y', 't', 'y') \\
&\quad \text{Inline function:} \\
&\quad f(t,y) = t^2-y \\
&> \quad \text{slope} = \text{feval}(f, 2, 1) \quad \% \text{note no @} \\
&\quad \text{slope} = 3
\end{align*}
\]

However, an inline function is only available where it is used and not to other functions. It is not recommended when the function implemented is too complicated or involves too many statements.

★ The three statements

\[
\begin{align*}
&\text{y=feval}(@\text{myfunction}, x); \\
&\text{y=myfunction(x)};
\end{align*}
\]

and

\[
\begin{align*}
&\text{y=feval('myfunction',x)}; \\
&\text{are equivalent.}
\end{align*}
\]

★ The statements

\[
\begin{align*}
&\text{myfunction} = \text{ @(x) (x}^2-\sin(pi.x)+exp(x))./(x-1); \\
&\text{and}
\end{align*}
\]

\[
\begin{align*}
&\text{myfunction} = \text{ inline}('x^2-\sin(pi.x)+exp(x))./(x-1)', 'x'); \\
&\text{are equivalent.}
\end{align*}
\]
Matrices and Linear Algebra

We have used one-dimensional $1 \times 21$ arrays $x$ and $y$ in previous examples. MATLAB can handle higher dimensional arrays. Two-dimensional arrays (matrices) are commonly used in many situations.

★ Matrices can be constructed in MATLAB in different ways. For example the $3 \times 3$ matrix $A = \begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix}$ can be entered as

\[
\begin{align*}
\text{>> } & A=[8,1,6;3,5,7;4,9,2] \\
& A = \\
& 8 \ 1 \ 6 \\
& 3 \ 5 \ 7 \\
& 4 \ 9 \ 2 
\end{align*}
\]

or

\[
\begin{align*}
\text{>> } & A=[8,1,6;3,5,7;4,9,2] \\
& A = \\
& 8 \ 1 \ 6 \\
& 3 \ 5 \ 7 \\
& 4 \ 9 \ 2 
\end{align*}
\]

or defined as the concatenation of 3 rows

\[
\begin{align*}
\text{>> } & \text{row1}=[8,1,6]; \text{row2}=[3,5,7]; \text{row3}=[4,9,2]; \text{A=[row1,row2,row3]} \\
& A = \\
& 8 \ 1 \ 6 \\
& 3 \ 5 \ 7 \\
& 4 \ 9 \ 2 
\end{align*}
\]

or 3 columns

\[
\begin{align*}
\text{>> } & \text{col1}=[8;3;4]; \text{col2}=[1;5;9]; \text{col3}=[6;7;2]; \text{A=[col1,col2,col3]} \\
& A = \\
& 8 \ 1 \ 6 \\
& 3 \ 5 \ 7 \\
& 4 \ 9 \ 2 
\end{align*}
\]

Note the use of , and ;. Concatenated rows/columns must have the same length. Larger matrices can be created from smaller ones in the same way:

\[
\begin{align*}
\text{>> } & C=[A,A] \quad \% \text{ Same as } C=[A \ A] \\
& C = \\
& 8 \ 1 \ 6 \ 8 \ 1 \ 6 \\
& 3 \ 5 \ 7 \ 3 \ 5 \ 7 \\
& 4 \ 9 \ 2 \ 4 \ 9 \ 2 
\end{align*}
\]

The matrix $C$ has dimension $3 \times 6$ ("3 by 6"). On the other hand smaller matrices (submatrices) can be extracted from any given matrix:

\[
\begin{align*}
\text{>> } & A(2,3) \quad \% \text{ coefficient of } A \text{ in 2nd row, 3rd column} \\
& \text{ans} = \\
& 7 \\
\text{>> } & A(1,:) \quad \% \text{ 1st row of } A \\
& \text{ans} = 
\end{align*}
\]
\begin{verbatim}
8 1 6
>> A(:,3) % 3rd column of A
ans =
  6
  7
  2
>> A([1,3],[2,3]) % keep coefficients in rows 1 & 3 and columns 2 & 3
ans =
  1 6
  9 2
\end{verbatim}

★ Some matrices are already predefined in MATLAB:

\begin{verbatim}
>> I=eye(3) % the Identity matrix
I =
  1 0 0
  0 1 0
  0 0 1
>> magic(3)
ans =
  8 1 6
  3 5 7
  4 9 2
(what is magic about this matrix?)
\end{verbatim}

★ Matrices can be manipulated very easily in MATLAB (unlike MAPLE). Here are sample commands to exercise with:

\begin{verbatim}
>> A=magic(3);
>> B=A' % transpose of A, i.e, rows of B are columns of A
B =
  8 3 4
  1 5 9
  6 7 2
>> A+B % sum of A and B
ans =
 16  4 10
  4 10 16
 10 16  4
>> A*B % standard linear algebra matrix multiplication
ans =
 101  71  53
  71  83  71
  53  71 101
>> A.*B % coefficientwise multiplication
ans =
  64   3  24
   3  25  63
  24  63   4
\end{verbatim}

Try \texttt{A*A}, \texttt{A^2}, \texttt{A.^2}.

★ Two MATLAB commands are especially relevant when studying the solution of linear systems of differentials equations:

1. \texttt{x=A\backslash b} determines the solution \(x = A^{-1}b\) of the linear system \(Ax = b\). The array \(b\) must have as many rows as the matrix \(A\).
2. \([S,D]=\text{eig}(A)\) determines the eigenvectors of \(A\) (columns of \(S\)) and associated eigenvalues (diagonal coefficients of \(D\)) (note that the \text{eig} function has one input and two output arguments).

As an example consider the matrix \(A=\text{magic}(3)\) again and \(x=\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}\):

\[
\begin{align*}
\text{>> } A&=\text{magic}(3) \\
A &= \\
&= \begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix} \\
\text{>> } x&=[1,2,3]' \quad \% \text{same as } x=[1;2;3] \\
x &= \\
&= \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \\
\text{>> } b&=A*x \\
b &= \\
&= \begin{bmatrix} 28 \\ 34 \\ 28 \end{bmatrix} \\
\text{>> } A\backslash b & \quad \% \text{this is } x! \\
an &= \\
&= \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \\
\text{>> } \text{inv}(A)*b & \quad \% \text{less efficient and accurate} \\
an &= \\
&= \begin{bmatrix} 1.0000 \\ 2.0000 \\ 3.0000 \end{bmatrix} \\
\text{>> } [S,D]=\text{eig}(A) \\
S &= \\
&= \begin{bmatrix} -0.5774 & -0.8131 & -0.3416 \\ -0.5774 & 0.4714 & -0.4714 \\ -0.5774 & 0.3416 & 0.8131 \end{bmatrix} \\
D &= \\
&= \begin{bmatrix} 15.0000 & 0 & 0 \\ 0 & 4.8990 & 0 \\ 0 & 0 & -4.8990 \end{bmatrix} \\
\text{>> } A*S(:,1)-D(1,1)*S(:,1) & \quad \% \text{test 1st eigenvector-eigenvalue pair} \\
an &= \\
&= \begin{bmatrix} 1.0e-014 * \\ -0.1776 \\ 0.3553 \\ -0.3553 \end{bmatrix}
\end{align*}
\]

Note the multiplicative factor \(10^{-14}\) in the last computation. MATLAB performs all operations using standard IEEE double precision.

### MATLAB Programming and Debugging

Several constructs are used in MATLAB:
1. repetitive loops (fixed number of times)

\[
\text{for } <\text{expression}>
\begin{align*}
&<\text{list of commands}> \\
&\text{end}
\end{align*}
\]

2. repetitive loops (indefinite number of times)

\[
\text{while } <\text{expression}>
\begin{align*}
&<\text{list of commands}> \\
&\text{end}
\end{align*}
\]

3. conditional branching

\[
\begin{align*}
&\text{if } <\text{expression}> \\
&<\text{list of commands}> \\
&\text{elseif } <\text{expression}> \\
&<\text{list of commands}> \\
&\text{else} \\
&<\text{list of commands}> \\
&\text{end}
\end{align*}
\]

or

\[
\begin{align*}
&\text{switch } <\text{expression}> \\
&\text{case } <\text{expression}> \\
&<\text{list of commands}> \\
&\text{case } <\text{expression}> \\
&<\text{list of commands}> \\
&\text{otherwise} \\
&<\text{list of commands}> \\
&\text{end}
\end{align*}
\]

The following examples illustrate the use of each construct:

1. **for** loop: determine the sum of the squares of integers from 1 to 10

\[
\begin{align*}
S &= 0; \ % \text{ initialize running sum} \\
\text{for } k &= 1:10 \\
&\text{S} = S+k^2; \\
\text{end}
\end{align*}
\]

What is \(S\)? Verify with MATLAB.

2. **while** loop: determine the sum of the inverses of squares of integers from 1 until the inverse of the integer square is less than \(10^{-5}\)

\[
\begin{align*}
S &= 0; \ % \text{ initialize running sum} \\
k &= 1; \ % \text{ initialize current integer} \\
\text{incr} &= 1; \ % \text{ initialize test value} \\
\text{while } &\text{ incr}>1e-10 \\
&\text{S} = S+\text{incr}; \\
&k = k+1; \\
&\text{incr} = 1/k^2; \\
\text{end}
\end{align*}
\]
What is the value of $S$ returned by this script? Compare to $\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}$.

Can $k$ and incr be both initialized by 0?

3. if statement: evaluate $y = \frac{1}{x-2}$ for a given (but unknown) scalar $x$

```matlab
function y=f(x)
if x==2
    disp('y is undefined at x = 2')
else
    y=1/(x-2);
end
```

or with switch statement:

```matlab
function y=f(x)
switch x
    case 2
        disp('y is undefined at x = 2')
    otherwise
        y=1/(x-2);
end
```

Try $f(1), f(2)$. Modify the example to allow for arrays as input.

★ Whenever possible all these construct should be avoided and available MATLAB functions used to improve efficiency. In particular lengthy do loops introduce a substantial overhead. Compare

```matlab
>> tic; S=0; for k=1:1000000; S=S+1/k^2; end; toc; S
elapsed_time =
1.8830
S =
1.6449
```

and

```matlab
>> tic; S=sum(1./(1:1000000).^2); toc; S
elapsed_time =
0.0800
S =
1.6449
```

★ Programming with MATLAB is fairly easy. Still a script or function may not execute properly due to some programming error. In this case MATLAB returns an error indicating where it stopped and why. Most of the time this is sufficient to find out the error and correct it, especially when you get used to it. But keep in mind that even non-fatal mistakes may eventually force a program to later crash. Debugging tools are available in the MATLAB editor to force the execution to stop at specific places within a script or function(s) and access the current state of available variables.

Write a script or function with the MATLAB editor and position the cursor on a selected line. Then try the buttons and in the editor window before executing the file in the Command Window. Observe what happens. Check the value of variables. To continue and eventually exit the debugger press return.

Exercises

Now that you have been through the essential elements of MATLAB relevant in this text, a good exercise is to go through the commands and change values, functions, and problems to familiarize yourself with the MATLAB syntax and commands introduced in this chapter.