DOING PHYSICS WITH MATLAB

GETTING STARTED WITH MATLAB

MATLAB BASICS

Ian Cooper
School of Physics, University of Sydney
ian.cooper@sydney.edu.au

DOWNLOAD MATLAB SCRIPTS

This document gives you a brief introduction and a reference section for getting started with Matlab. The first section describes the extensive on-line help that is provided within the Matlab environment. The remaining sections describe some of the more commonly used Matlab commands, how data is stored in matrices and how to create your own Matlab functions.

HELP

There are an enormous number of Matlab commands that can be used. Using the Command Window or the Help Window, one can access most of the information about using Matlab. Access to the Help Window is though typing **helpdesk** or selecting the Help menu or typing **help** in the Command Window. A good way to get more familiar with using Matlab is to know how to use the help provided effectively.

The Help has a hierarchical structure, for example:

help (help topics)→ **help elfun** (elementary math functions) → **help atan2** (four quadrant inverse tangent).

The help entries can be searched for keywords using the **lookfor** command. For example, searching for the keyword gives numerous matches:

lookfor 'inverse' →

nvhilb	inverse Hilbert matrix.
ipermute	inverse permute array dimensions.
acos	inverse cosine.
acosh	inverse hyperbolic cosine.
acot	inverse cotangent.
acoth	inverse hyperbolic cotangent.
…	

Typing **help** in the Command Window gives a list of the available help topics.

help

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 and numerical linear algebra
matlab\datafun	Data analysis and Fourier transforms
matlab\audio	Audio support
matlab\polyfun	Interpolation and polynomials
matlab\funfun	Function functions and ODE solvers
matlab\sparfun	Sparse matrices
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\iofun	File input/output
matlab\timefun	Time and dates
matlab\datatypes	Data types and structures
matlab\verctrl	Version control
matlab\winfun	Windows Operating System Interface Files (DDE/COM)
matlab\demos	Examples and demonstrations
toolbox\local	Preferences
images\images	Image Processing Toolbox
images\imdemos	Image Processing Toolbox --- demos and sample images
signal\signal	Signal Processing Toolbox
signal\signal	Signal Processing Toolbox
signal\sigtools	Design & Analysis Tool (GUI)
signal\sptoolgui	Signal Processing Toolbox GUI
signal\sigdemos	Signal Processing Toolbox Demonstrations.

For more help on directory/topic, type "help topic".
For command syntax information, type "help syntax".

Help elfun

Elementary math functions

Trigonometric			
sin	Sine	**sinh**	Hyperbolic sine
asin	Inverse sine	**asinh**	Inverse hyperbolic sine
cos	Cosine	**cosh**	Hyperbolic cosine
acos	Inverse cosine	**acosh**	Inverse hyperbolic cosine
tan	Tangent.	**tanh**	Hyperbolic tangent
atan	- Inverse tangent	**atan2**	Four quadrant 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	**coth**	Hyperbolic cotangent
acot	Inverse cotangent	**acoth**	Inverse hyperbolic cotangent

Exponential			
exp	Exponential	**log**	Natural logarithm
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	**nextpow2**	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		

help atan2

ATAN2 Four quadrant inverse tangent.
ATAN2(Y,X) is the four quadrant arctangent of the real parts of the elements of X and Y. -pi <= ATAN2(Y,X) <= pi.

See also ATAN.

The Matlab help accessed through the Help Window contains more information than the information displayed in the Command Window. For example, searching in the Help Window for the function **atan2** gives:

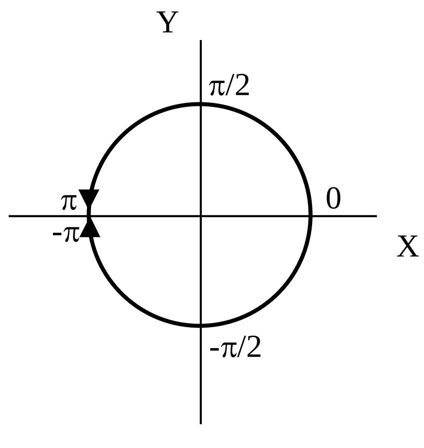
atan2 Four-quadrant inverse tangent
Syntax
P = atan2(Y,X)

Description
P = atan2(Y,X) returns an array P the same size as X and Y containing the element-by-element, four-quadrant inverse tangent (arctangent) of the real parts of Y and X. Any imaginary parts are ignored.

Elements of P lie in the closed interval [-pi,pi], where pi is the MATLAB floating-point representation of π.

atan uses sign(Y) and sign(X) to determine the specific quadrant.

atan2(Y,X) contrasts with atan(Y/X), whose results are limited to the interval [$-\pi/2$, $\pi/2$] , or the right side of this diagram.



Examples
Any complex number z = x + iy is converted to polar coordinates with
r = abs(z)
theta =atan2(imag(z),real(z))

For example, z = 4 + 3i;
r = abs(z);
theta = atan2(imag(z), real(z))
r = 5 theta = 0.6435

help FILEFORMATS

Readable file formats.

Data Formats		
Format	**Command**	**Returns**
MAT - MATLAB workspace	Load	Variables in file
CSV - Comma separated numbers	csvread	Double array.
DAT - Formatted text	importdata	Double array
DLM - Delimited text	dlmread	Double array
TAB - Tab separated text	dlmread	Double array
Spreadsheet formats		
XLS - Excel worksheet	xlsread	Double & cell array
Scientific data formats		
CDF - Common Data Format	cdfread	Cell array of CDF records
FITS - Flexible Image Transport System	fitsread	Primary or extension table data
HDF - Hierarchical Data Format	hdfread	HDF or HDF-EOS data set
Movie formats		
AVI - Movie	aviread	MATLAB movie
Image formats		
TIFF - TIFF image	imread	Truecolor, grayscale or indexed image(s).
PNG - PNG image	imread	Truecolor, grayscale or indexed image
HDF - HDF image	imread	Truecolor or indexed image(s)
BMP - BMP image	imread	Truecolor or indexed image
JPEG - JPEG image	imread	Truecolor or grayscale image
GIF - GIF image	imread	Indexed image
PCX - PCX image	imread	Indexed image
XWD - XWD image	imread	Indexed image
CUR - Cursor image	imread	Indexed image
ICO - Icon image	imread	Indexed image.
RAS - Sun raster image	imread	Truecolor or indexed
PBM - PBM image	imread	Grayscale image
PGM - PGM image	imread	Grayscale image
PPM - PPM image	imread	Truecolor image
Audio formats		
AU - NeXT/Sun sound	auread	Sound data and sample rate
SND - NeXT/Sun sound	auread	Sound data and sample rate
WAV - Microsoft Wave sound	wavread	Sound data and sample rate

See also IOFUN

Matlab help is very useful but extensive and so the purpose of this Chapter is to review many of the common Matlab features and commands through illustrative examples.

GENERAL PURPOSE COMMANDS

The following table list just a few of the Matlab commands that are used for managing the Matlab environment. The Matlab command is typed into the Command Window. For more information on any of the commands lists use help, e.g., help ver.

Matlab Command	Function / Purpose
helpdesk	Opens Help Window
demo	Can view and run available Matlab demonstrations.
info	Provides contact information for getting extra assistance with Matlab.
ver	Displays the current Matlab, Simulink and toolbox version information.
dir	Lists the files in a directory. Pathnames and wildcards may be used. For example, dir *.m lists all the M-files in the current directory.
cd	cd by itself, prints out the current directory. Change current working directory. cd directory-spec: sets the current directory to the one specified cd \a03\mat\graphics cd .. moves to the directory above the current one.
path	Controls Matlab's search path. For example, the following statements add another directory to Matlab's search path Windows: path(path,'c:\ao3\mat\graphics')
pdf	Shows current working directory.
what	List MATLAB specific files in the current directory.
which	Locates functions and files which result → result not found. which sinc → C:\a03\mat\mg\scripts\sinc.m
save **load**	save test.mat → saves all workspace variables to the file test.mat in the current directory. load test → loads the variables saved in the file test.mat. save xData → saves only the variable xData.

	load Xdata → loads the variable xData into the Matlab workspace. save test.mat xData yData zData → saves the variables Xdata, yData and zData in the file test.mat in the current directory.
delete	delete test.mat → deletes the file test.mat from the current directory.
pack	Consolidate workspace memory: performs memory garbage collection. Extended Matlab sessions may cause memory to become fragmented, preventing large variables from being stored. pack saves all variables on disk, clears the memory, and then reloads the variables.
diary	Save text of MATLAB session. diary filename → causes a copy of all subsequent command window input and most of the resulting command window output to be appended to the named file. If no file is specified, the file 'diary' is used. diary off → suspends it. diary on → turns it back on. diary → , by itself, toggles the diary state.
clear	clear → clears all variables and functions from workspace. clear all → removes all variables, globals, functions and MEX links. clear Xdata yData → clears the variables xData and yData from the workspace.
home	Moves the cursor to the upper left corner of the Command Window and clears the visible portion of the window. You can use the scroll bar to see what was on the screen previously.
clc	Clears the command window and homes the cursor.
echo on / off	Toggles the printing of instructions from m-script in Command Window.

Miscellaneous m-script commands

<table>
<tr>
<td>

pause wait for user response (press any key to continue)

pause(10) → halts execution of m-script for 10 seconds

pause(0.1) → halts execution of m-script for 0.1 seconds

pause off → subsequent pause ignored

pause on → subsequent pause commands should pause

</td>
<td>

keyboard
stops execution of the m-script and gives control to keyboard.

K appears before the prompt.

Variables may be examined or changed – all commands are valid.

Keyboard mode terminated by hitting Enter

input
prompt user for input

`num = input('How many particles?  ')`

</td>
</tr>
<tr>
<td colspan="2">

menu
choice = menu(header, item1, item2, ...)

Generate a menu of choices for user input.

```
K = menu('Choose a color','Red','Blue','Green')   →

----- Choose a color -----

  1) Red
  2) Blue
  3) Green

  Select a menu number:
```

The number entered by the user in response to the prompt is returned as K (i.e. K = 2 implies that the user selected Blue).

</td>
</tr>
</table>

Operators and special functions

<table>
<tr><td>+</td><td>plus</td><td>-</td><td>minus</td><td>^</td><td>matrix power</td></tr>
<tr><td>.^</td><td>array power</td><td>\</td><td>backslash or left division</td><td>/</td><td>slash or right division</td></tr>
<tr><td>./</td><td>array division</td><td>:</td><td>colon (subscripting, array manipulation)</td><td>( )</td><td>parentheses (contains arguments)</td></tr>
<tr><td>.</td><td>decimal point</td><td>..</td><td>parent directory</td><td>...</td><td>continuation</td></tr>
<tr><td>,</td><td>comma (argument / statement separator)</td><td>;</td><td>semicolon (suppress statement output)</td><td>*</td><td>matrix multiplication</td></tr>
<tr><td>.*</td><td>array multiplication</td><td>%</td><td>comment</td><td>'</td><td>transpose</td></tr>
<tr><td>.'</td><td>nonconjugated transpose</td><td>=</td><td>assignment</td><td>==</td><td>equality</td></tr>
<tr><td><</td><td>less than</td><td>></td><td>greater than</td><td><=</td><td>less than or equal to</td></tr>
<tr><td>>=</td><td>greater than or equal to</td><td>&</td><td>logical AND</td><td>|</td><td>logical OR</td></tr>
<tr><td>~</td><td>logical NOT</td><td>xor</td><td>logical EXCLUSIVE OR</td><td></td><td></td></tr>
</table>

Rational and logical operations

=   <   <=   >   >=   ==   ~=

```
x = 1; y = 20;
if x == 2, a = 0, end;
if y >= 15, a = 1, end
      → a = 1

x == 4  →  0     x == 1  →  1
```

&

logical **and**

```
x = 1; y = 20;
if x == 1 & y <= 2, a = 0, end;
if x > 0 & y < 100, a = 1, end
      → a = 1
```

|

logical **or**

```
x = 1; y = 20; a = 99;   b = 1;
if x < 1 | y >= 2, a = 0, end;
if x > 10 | y < 1, a = 1, end
      → a = 0

if (x == 2 | y ~= 19) & (a == 99),
   b =  0;
        →  b = 0
```

~

logical **not**

```
x = 1; y = 20; a = 0;

~x  → 0   ~y →  0   ~a  →  1

x ~= 4  →  1
```

ARRAYS AND MATRICIES

Matlab works with arrays or matrices and the elements may be strings, real or complex numbers and functions can have real or complex arguments. Matlab functions and arithmetic operations can be performed directly on matrices. A matrix of a single element can be through of a single constant or variable (A = 3). A matrix can be a row vector or a column vector or a multi-dimensional array. Unlike most programming languages, commands can act simultaneously on all elements of an array. For example the set of numbers 1, 4, 9, 16, 25, 36, 49 can be entered into a row vector by a statement in the Command, for example:

```
xR = [1 4 9 16 25 36 49]
```

The command **sqrt(xR)** will act on each element of the array xR by taking the square root of each number

```
sqrt(xR)  →  1  2  3  4  5  6  7
```

The name of a matrix must be start with any letter, followed by any combination of letters (upper or lower case) and numbers, for example,

A, a, ScreenWidth, xR, xC, Slit_separation   (A and a refer to different matrices)

The tables below show how data can be entered into a matrix; how to perform some of the operations and use functions that can act on matrices; and lists some of the special Matlab matrices. The appearance and number of significant figures of a matrix displayed in the Command Window can be changed using the **format** command.

Format x = 51.12345678987654321 y = 5.1123456×10^{23}	
format or **format short** x → 51.1235 y → 5.1123e+023	**format long** x → 51.12345678987654 y → 5.112345600000000e+023
format short e x → 5.1123e+001 y → 5.1123e+023	**format long e** x → 5.112345678987654e+001 y → 5.112345600000000e+023
disp (display array) disp(x) → 51.1235 tm = ' time t (s)' disp(tm) → time t (s)	

There is a very extensive set of Matlab mathematical functions. Some of the functions which are most commonly used are given in the table below. It is a good idea to practice using these functions in the Command Window.

Miscellaneous functions

abs(x) abs(-51) → 51	**sqrt(x)** sqrt(51) → 7.1414 sqrt(-51) → 0 + 7.1414i
round(x) round to nearest integer round(51) → 51 round(-51) → -51 round(51.145) → 51 round(-51.145) → -51 round(51.845) → 52 round(-51.845) → -52	**fix(x)** round towards zero fix(51) → 51 fix(-51) → -51 fix(51.145) → 51 fix(-51.145) → -51 fix(51.845) → 51 fix(-51.845) → -51
floor(x) round toward - ∞ floor(51) → 51 floor(-51) → -51 floor(51.145) → 51 floor(-51.145) → -52 floor(51.845) → 51 floor(-51.845) → -52	**ceil(x)** round toward + ∞ ceil(51) → 51 ceil(-51) → -51 ceil(51.145) → 52 ceil(-51.145) → -51 ceil(51.845) → 52 ceil(-51.845) → -51
sign(x) sign sign(51.145) → 1 sign(-51.145) → -1 sign(0) → 0	**mod(x,y)** modulus mod(30,5) → 0 mod(-30,5) → 0 mod(31,5) → 1 mod(-31,5) → 4 mod(34,5) → 4 mod(-34,5) → 1
rem(x,y) remainder rem(30,5) → 0 rem(-30,5) → 0 rem(31,5) → 1 rem(-31,5) → -1 rem(34,5) → 4 rem(-34,5) → -4	**exp(x)** exponential base e exp(1) → 2.7183 exp(0) → 1 exp(-5.145) → 0.0058
log(x) log base e log(exp(1)) → 1 log(10) → 2.3026	**log10(x)** log base 10 log10(exp(1)) → 0.4343 log10(51.145) → 1.7088
factorial(x) factorial x! factorial(4) → 24	**rand** random number 0 to 1 rand → 0.9318 rand(2,3) → 0.4660 0.8462 0.2026 0.4186 0.5252 0.6721 a = 1; b = 10; floor(a + b*rand) → random integer from 1 to 10 ? rand('state', sum(100*clock)) reset random number generator, so different results are obtained

date s = date; s → 04-Oct-2003	**clock** clock = [year month day hour minute seconds] fix(clock) → 2003 10 4 16 23 35
etime difference in seconds between two dates T0 = [2000 10 15 0 0 0] T1 = [2004 10 10 12 0 0] etime(T1,T0) → 125841600	**tic** starts a seconds counter **toc** stops the seconds counter
cputime elapsed CPU time in seconds a = cputime → a = 1.811010000000000e+003	
eval evaluates a string expression s = '123' a = eval(s) → a = 123	**feval** executes functions specified by strings
global defines global variables	**...** *(press Enter)* Long lines converted into short lines y = amp1*sin(2*pi*x/lambda1) ... + amp2*sin(2*pi*x/lambda2)

Complex numbers $z = x + i\,y$ $i = \sqrt{-1}$ or $j = \sqrt{-1}$

angle(z) z = 4.5600 + 1.2300i angle(z) → 0.2635	**real(z)** z = 4.5600 + 1.2300i real(z) → 4.5600
imag(z) z = 4.5600 + 1.2300i imag(z) → 1.2300	**conj(z)** z = 4.5600 + 1.2300i conj(z) → 4.5600 – 1.2300i
abs(z) z = 4.5600 + 1.2300i abs(z) → 4.7230	

Trigonometric functions (angles must be radians)

sin(x) x= 30 % x in degrees sin(x*(pi/180)) → 0.5000	**asin(x)** x = 0.7071 asin(x) → 0.7854 % angle in radians (180/pi)*asin(x) → 44.995 % angle in degrees
cos(x) x= 30 % x in degrees cos(x*(pi/180)) → 0.8660	**acos(x)** x = 0.7071 acos(x) → 0.7854 % angle in radians (180/pi)*acos(x) → 45.0005 % angle in degrees
tan(x) x= 30 % x in degrees tan(x*(pi/180)) → 0.5774	**atan(x)** x = 0.7071 atan(x) → 0.0706 % angle in radians (180/pi)*atan(x) → 4.0447 % angle in degrees
atan2(y,x) y = 1; x = 1 tan2(y,x)/pi → 0.2500 y = 1; x = -1 tan2(y,x)/pi → 0.7500 % answers in rad / π	**atan2(y,x)** y = -1; x = 1 tan2(y,x)/pi → -0.2500 y = 1; x = -1 tan2(y,x)/pi → 0.7500 % answers in rad / π

Setting up matrices

Simple variables

```
u = 6; v = -12;
```

Complex variable

```
z = 12 – 3i
```

Row vector

```
xR = [1 2 3 4]   →  1 2 3 4
yR = [9,  6,  3]  →  9 6 3

Xmin = 0;   Xmax = 3;  dX = 0.5;
```

X = Xmin : dX : Xmax

```
→  0   0.5   1.0  1 .5   2.0   2.5   3.0

length(XR)  →   4
length(yR)  →   3
length(X)   →   7
```

Removing an element

```
X(6) = []  →
          0   0.5   1.0  1 .5   2.0   3.0
```

Column Vector

```
xC = [1 ; 2; 3; 4]  →
                    1
                    2
                    3
                    4

length(xC)  →  4
```

Removing element

```
xC(3) = []  →
           1
           2
           4
```

Adding a column

```
xC(:,2) = [6; 7; 8; 9]  →
           1    6
           2    7
           3    8
           4    9
```

Matrix <u>3</u> rows × <u>4</u> columns

```
M = [1 2 3 4; 5 6 7 8; -1 -5 -8 -7]  →
      1     2     3     4
      5     6     7     8
     -1    -5    –8    -7

M(3,2)  →  -5    M(1,4)  →  4

M(:, 2)  →   2               column vector
             6
            -5

M(1 , :)  →  1  2  3  4          row vector

M(2, [1 3])  →   5   7

M(3, 2:3)  →  -5   -8

Removing a column
M(:, 2) = []   →  1     3     4
                  5     7     8
                 -1    –8    -7
```

Matricies

size size of an array size(u) → 1, 1 size(xR) → 1, 4 size(xC) → 4, 1 size(M) → 3, 4	**who** lists the current variables **whos** lists all the variables in the current workspace, together with information about their size, bytes, class, etc.
linspace(Xmin; Xmax, N) linear spaced row vector with N elements from Xmin to Xmax values with a spacing between elements of dX = (X(N)-X(1))/(N-1) Xmin = 0; Xmax = 10; N = 10; X = linspace(Xmin,Xmax,N) → 0, 1.111, 2.222, 3.333, 4.444, 5.556, 6.667, 7.778, 8.889, 1.000 Xmin = 0; Xmax = 10; N = 11; X = linspace(Xmin,Xmax,N) → 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10	**logspace(a, b, N)** generates a row vector of N logarithmically equally spaced points between decades 10^a and 10^b a = 0; b = 3; N = 6; X = logspace(a,b,N) → 1.0e+003 * 0.0010 0.0040 0.0158 0.0631 0.2512 1.0000
eye(N) $N \times N$ **identity** matrix eye(4) → $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	**zeros(M,N)** M×N zero matrix zeros(3,5) → $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$
ones(M,N) M×N unit matrix ones(2,6) → $\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$	**sort(X)** X = [9 5 7 3 1 2 8] sort(X) → 1 2 3 5 7 8 9

sum(X)

```
X = [9 5 7 3 1 2 8]

sum(X) → 35

M = [1 2 3 4; 5 6 7 8; -1 -5 -8 -7] →
     1    2    3    4
     5    6    7    8
    -1   -5   -8   -7

sum(M) or sum(M,1)  → 5  3  2  5
```
sums columns

```
sum(M(:,1)) → 5
```
sums 1st column

```
sum(M,2) → 10
           26
          -21
```
sums rows

```
sum(sum(M)) → 15
```
sums all elements

max(X) min(X)

```
X = [9 5 7 3 1 2 8]

max(X) → 9
min(X) → 1

M = [1 2 3 4; 5 6 7 8; -1 -5 -8 -7] →
     1    2    3    4
     5    6    7    8
    -1   -5   -8   -7

min(M)  → -1  -5  -8  -7
```
min columns

```
max(M) → 5  6  7  8
```
max columns

```
min(min(M))  → -8
max(max(M))  →  8
```
max or min of all elements

```
max(M')  → 4  8  -1
```
max rows

num2str

converts a number to a string

```
pi  = 3.14159265358979
h  =  6.626076×10^-34
S = [1.126 2.123 ; 3.123 4.123]
  →   1.123   2.123
      3.123   4.123

num2str(pi)  →  '3.1416'
num2str(pi, 0)  → '3'
num2str(pi,8)  → '3.1415927'
num2str(h)  →  '6.6261e-034'
num2str(h, 0)  → '7e-034'
num2str(h,8)  → '6.63e-034'
num2str(S,2)  →
                 '1.1    2.1'
                 '3.1    4.1'
```

str2num

converts a character array representation of a matrix of numbers to a numeric matrix

```
str2num('123')  →  123

str2num('abc123')  →  []
```

disp

displays the array, without printing the array name, same as leaving the semi-colon off an expression except that empty arrays don't display.

```
disp(pi)   →  3.1416
disp('Speed')  →  Speed
```

```
max_speed = 25.45
disp('The maximum speed is ',num2str(max_speed),'  m/s ')  →

  The maximum speed is 25.45 m/s
```

Format is a format control string containing conversion specifications or any optional text

%P.Qe for exponential
%P.Qf for fixed point
%P.Qg select shorter of %P.Qe or %P.Qf

P integer specifying field width
Q integer specifying number of decimal places

\n produces a new line

fprintf

Write formatted data to file.

```
x = 0:.1:1; y = [x; exp(x)];
fid = fopen('exp.txt','w');
fprintf(fid,'%6.2f  %12.8f\n',y);
fclose(fid);
```

sprintf

Write formatted data to string.

```
sprintf('%0.5g',(1+sqrt(5))/2)  →  1.618
sprintf('%0.5g',1/eps)  →  4.5036e+15
sprintf('%15.5f',1/eps)  →   503599627370496.00000
sprintf('%d',round(pi))  →  3
sprintf('%s','Speed')  →  Speed
sprintf('The array is %dx%d.',2,3)  →  The array is 2x3.
sprintf('\n')  →  line termination character
```

csvrad read a file of comma-separated values

cswrite write a file of comma-separated values

fclose close file
fopen open file
fread read binary data from file
fwrite write binary data to file
fprintf write formatted data to file
fscanf read formatted data from file

Matrix operations

Matrices that have identical dimensions can be **added** or **subtracted**.

A = [1 2 3; 4 5 6] → $\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix}$ B = [9 8 7; 6 5 4] → $\begin{matrix} 9 & 8 & 7 \\ 6 & 5 & 4 \end{matrix}$

A + B → $\begin{matrix} 10 & 10 & 10 \\ 10 & 10 & 10 \end{matrix}$ A – B → $\begin{matrix} -8 & -6 & -4 \\ -2 & 0 & 2 \end{matrix}$

C = [A ; 10 11 12] → $\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 10 & 11 & 12 \end{matrix}$

D = A + C → ??? Error using ==> + Matrix dimensions must agree.

Matrices can be **multiplied** together. For example, C = A B where the matrix A has elements a_{ik} (i row and k column), B has elements b_{kj} and C has elements c_{ij}

$$c_{ij} = \sum_k a_{ik}\, b_{kj}$$

A = [1 2 3 ; 4 5 6] → $\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix}$ B = [1 4 ; 2 6 ; 3 8] → $\begin{matrix} 1 & 4 \\ 2 & 6 \\ 3 & 8 \end{matrix}$

A*B → $\begin{matrix} 14 & 40 \\ 32 & 94 \end{matrix}$ B*A → $\begin{matrix} 17 & 22 & 27 \\ 26 & 34 & 42 \\ 35 & 46 & 57 \end{matrix}$

Element by element multiplication can be done using the **dot** • operator, for example,

C = [2 8; 4 12; 6 16]

B .* C = C .*B → $\begin{matrix} 2 & 32 \\ 8 & 72 \\ 18 & 128 \end{matrix}$

For element by element multiplication, the two matrices must have matching dimensions. For example, error messages are returned for A .* B or B .*A

A .*B → ??? Error using ==> .* Matrix dimensions must agree.

B.*A → ??? Error using ==> .* Matrix dimensions must agree.

The **transpose** of a matrix is gives by the command transpose or **'**. For example,

transpose(A) →
```
1   4
2   5
3   6
```

B' →
```
1   2   3
4   6   8
```

xR = [2 4 6 8] → 2 4 6 8   yR = [-1 1 1 −1] → -1 1 1 -1

yR' →
```
−1
 1
 1
−1
```

xR * yR → ??? Error using ==> * Inner matrix dimensions must agree.

xR * yR' → 0

xR .* yR → -2 4 6 -8

xR .* yR' → ??? Error using ==> .* Matrix dimensions must agree.

If the matrix that is to be transposed has complex elements, then the ' operator gives the complex conjugate transpose. To give the transpose without conjugation, use the .' operation

C = [1 4+8i ; 2-i 5 ; 3+6i 6-3i] →
```
1.0000                4.0000 + 8.0000i
2.0000 - 1.0000i      5.0000
3.0000 + 6.0000i      6.0000 - 3.0000i
```

C' →
```
1.0000                2.0000 + 1.0000i      3.0000 - 6.0000i
4.0000 - 8.0000i      5.0000                6.0000 + 3.0000i
```

C.' →
```
1.0000                2.0000 - 1.0000i      3.0000 + 6.0000i
4.0000 + 8.0000i      5.0000                6.0000 - 3.0000i
```

Matrix division

A / B →
```
2.3333   -3.3333
3.3333   -4.3333
```

A \ B →
```
-6.0000   -5.5000   -5.0000
 0         0         0
 5.0000    4.5000    4.0000
```

A ./ B →
```
0.1111    0.2500    0.4286
0.6667    1.0000    1.5000
```

Manipulating matrices

```
          1   2   3
A   =     4   5   6
          7   8   9
```

rot90 rotate matrix by 90 degrees `rot90(A)  →  3 6 9 / 2 5 8 / 1 4 7` 3 6 9 2 5 8 1 4 7	**diag** create or extract diagonals diag(A) → 1 5 9
trace sum of diagonal elements trace(A) → 15	**det** determinant det(A) → 0
tril extract lower triangular part tril(A) → 1 0 0 4 5 0 7 8 9	**triu** extract upper triangular part triu(A) → 1 2 3 0 5 6 0 0 9
fliplr flip matrix in the left – right direction fliplr(A) → 3 2 1 6 5 4 9 8 7	**flipup** flip matrix in the up – down direction flipup(A) → 7 8 9 4 5 6 1 2 3
flipdim flip matrix along specified dimension flipdim(A,1) → 7 8 9 4 5 6 1 2 3 flipdim(A,2) → 3 2 1 6 5 4 9 8 7	**norm** matrix or vector norm norm(x) gives Euclidean length x = [0 1 2 3] norm(x) → sqrt(0+1+4+9) = 3.7417

Find

Find indices of nonzero elements.

`I = FIND(X)` → the indices of the vector `X` that are non-zero.

`I = FIND(A>100)` → the indices of `A` where `A` is greater than 100.

`[I,J] = FIND(X)` → row and column indices of the nonzero entries in matrix `X`.

`[I,J,V] = FIND(X)` → vector containing the nonzero entries in `X`.

Note that `find(X)` and `find(X~=0)` will produce the same `I` and `J`, but the latter will produce a V with all 1's.

MATLAB AS A PROGRAMMING LANGUAGE

Control of the execution of a program

Matlab is a program language where the code is stored in text files as m-script or as functions. An important set of commands are used to control the flow of the program by testing when some condition is satisfied using **if-else-end** or **switch-case** commands and by using **for** and **while** loops to repeat a set of statements.

Examples: if-end, if-else-end, if-elseif-end commands

```
N = input('   Enter a number   ');
text = 'The number is not an integer'
if mod(N,2) == 0, text = 'The number is even integer'; end
if mod(N,2) == 1, text = 'The number is odd integer'; end
disp(text)

N = input('   Enter a number   ');
if mod(N,2) == 0
 text = 'The number is even integer';
else
text = 'The number is not an even integer';
end
disp(text)

N = input('   Enter a number   ');
if mod(N,2) == 0
  text = 'The number is even integer';
elseif mod(N,2)== 1
 text = 'The number is an odd integer';
else
 text = 'The number is not an integer';
end
disp(text)
```

Loops

To maximize speed of execution, matrices should be pre-allocated before a For or While Loop.

for … end   break

Using the **for … end** loop commands, statements can be repeated a number of times. Long loops are more memory efficient when the colon expression appears in the **for** command since the index vector is never created. The **break** statement can be used to terminate the loop prematurely. If the initial value is xMin, the increment is dx (can be positive or negative) and xMax is the final value of the loop variable

```
for c = xMin : dx : xMax
 x = 20;
y = 10;
for cx = 1 : x;
for cy = x: -2 : y;
  psi(cx, cy) = cx^2 + cy^2;
  sin(2*pi*cx/25)*sin(2*pi*cy/55);
end
end
end
```

while … end

The while statement is used to repeat a statement a number of times until a conditions is not satisfied. For example, to calculate a function a given number of steps

```
maxSteps – input('Enter the number of steps for calculations ');
Steps = 1;

while Steps <= maxSteps
x(Steps+1) =
```

switch … case … end

The selection of a block of code to be executed can be done with the **switch - cases** statements. For example to evaluate different functions

```
a = 2; b = 0.5;
x = 0 : 2 : 10;

flag = input('Select type of equation: 1, 2, …, ')

switch flag
case 1
        y = a .* x + b;
case 2
        y = a. * x;
case 3
        y = a .* exp(-b.*x)
otherwise
        y = [];
 end
```

Running this code with flag = 2 $\rightarrow$ y = 0   4   8   12   16   20

FUNCTIONS

Functions in Matlab are a very powerful tool for evaluating a sequence of commands and / or evaluating mathematical functions. The function is a text file similar to an m-script and has a .m extension. Input variables can be passed to the function and output variables are returned, any intermediate variable values within the function are not passed on to the Matlab workspace or to other functions. The function can be executed from the Command Window or from an m-script. To illustrate the how to create and use Matlab functions, a number of examples will be considered.

Example: Distance between two points

If the coordinates of two points P(x_P, y_P, z_P) and Q(x_Q, y_Q, z_Q) are known than the distance, *d* between the points is

$$d = \sqrt{\left(x_P - x_Q\right)^2 + \left(y_P - y_Q\right)^2 + \left(z_P - z_Q\right)^2}$$

The input variables passed to the function are the six coordinates of the two points P and Q. The output variable returned from the function is the distance *d*. The text for the function `distance.m` is

```
function d = distance(xP,yP,zP,xQ,yQ,zQ)
% Function to calculate the distance between two points P and Q
d = sqrt((xP-xQ)^2 + (yP-yQ)^2 + (zP-zQ)^2);
```

The following statement when entered into the Command Window

```
d = distance(0,0,0,1,1,1)
```

gives

```
d = 1.7321
```

Example: Converting between Cartesian and polar coordinates

A point P in a plane can be specified in Cartesian (x_P, y_P) or in polar coodinates (ρ_P, θ_P). The relationships between the two coordinate systems are

$$\rho_P = \sqrt{x_P{}^2 + y_P{}^2} \qquad \theta_P = \text{atan}\left(\theta_P\right)$$

$$x_P = \rho_P \cos\theta_P \qquad y_P = \rho_P \sin\theta_P$$

The two functions to convert Cartesian to polar and polar to Cartesian coordinates are

---

```
function [rho, theta] = CartesianToPolar(x,y)

% Function to convert Cartesian coordinates (x, y)
%   to polar coordinates (rho, theta)
% The Matlab function atan2 returns an angle in radians
%    If y >=0 then 0 <= theta <= pi
%    if y < 0 then -pi < theta < 0

rho = sqrt(x^2 + y^2);

theta = atan2(y,x);
```

```
function [x, y] = PolarToCartesian(rho,theta)

% Function to convert Polar coordinates (rho, theta)
%   to Cartesian coordinates (x, y)
% The angle theta must be in radians

x = rho * cos(theta);

y = rho * sin(theta);
```

The functions can be used in the Command Window or executed from an m-script, for example,

```
[xP, yP] = PolarToCartesian(1, pi/4)
```

gives

`xP = 0.7071` and `yP = 0.7071`

[rho, theta] = CartesianToPolar(3, 4)

gives

`rho = 5` and `theta = 0.9273`

Example: Evaluating an expression

Functions are very useful for evaluating a mathematical expression from the Command Window or from an m-script. We will consider evaluating the sinc function that is widely used in physics and engineering with the function sinc.m. The sinc function can be expressed as a function of the single variable θ where θ is an angle in radians by

$$sinc(\theta) = \frac{\sin(\theta)}{\theta}$$

The sinc function approaches 1 as θ approaches 1, but this causes a problem in Matlab when you try to divide by zero. This can problem can be overcome by using the Matlab function eps which is the smallest difference between two numbers.

```
function result = sinc(theta)

% Function to evaluate the sinc function

result = sin(theta + eps) ./ (theta + eps);
```

For example, sinc(0) gives then answer 1. For the array input for θ

```
theta = 0 : 0.25 : 1
```

theta	0.0000	0.2500	0.5000	0.7500	1.0000
sinc(theta)	1.0000	0.9896	0.9589	0.9089	0.8415