MATLAB Cheat Sheet
(Revised: 6/1/2023)

Script File (M-file) – this is a list of commands that are run in script order

Commands

disp(‘ Write this text on screen’)	display comment in ‘ ‘ on the screen
%	Comment statement (one line only)
	Commenting multiple lines
%{	before first line
%}	after last line
;	semi-colon – calculate but suppress print statement
…	3 periods is a continuation statement

Command Window Control

close all	close all files
clear all	clear all variables
clc	clear the command window

Variables

Case sensitive, up to 19 characters, beginning with a letter

Special Variables

ans	default variable for all results
pi	3.141592654…
eps	epsilon (small positive value)
inf	infinity
NaN	Not a Number
i and j	imaginary numbers, sqrt(-1)
abs(x)	absolute value of ()
acos(x), acosd(x)	inverse cosine of (x), result is expressed in radians or degrees
acosh(x)	inverse hyperbolic cosine ()
angle(x)	four quadrant angle of complex
asin(x)	inverse sine (x), result is expressed in radians or degrees
asinh(x)	inverse hyperbolic sine ()
atan(x), atand(x)	inverse tangent (), result is expressed in radians or degrees
atan2(x,y)	four quadrant inverse tangent
atanh(x)	inverse hyperbolic tangent ()
conj(x)	complex conjugate
cos(x), cosd(x)	cosine() of x expressed in radians or degrees
cosh(x)	hyperbolic cosine
exp(x)	exponential “e to the x”
imag(x)	complex imaginary part
log(x)	natural log
log10(x)	base 10 log
rand	random number – uniformly distributed between 0 and 1
real(x)	complex real part

rem(x,y)	remainder after division of (x/y)
round(x)	round towards nearest integer
sign(x)	return sign of (x) , +, -, 0
sin(x), sind(x)	sine of (x), expressed in radians or degrees
sinh(x)	hyperbolic sine
solve(eqn,x)	find all symbolic roots (x) of equation (eqn), example:

```
syms x
eqn = x*x*x-1==0;
S = eval(solve(eqn,x))   % use eval for numbers
answer: S = 1.0000 + 0.0000i
           -0.5000 - 0.8660i
           -0.5000 + 0.8660i
```

sqrt(x)	square root
tan(x), tand(x)	tangent of (x), expressed in radians or degrees
tanh(x)	hyperbolic tangent
vpasolve(eqn,x,[1,6])	first numerical root (x) of the equation (eqn) in the range [1,6], example:

```
syms x   % 1-DOF nonlinear equation
eqn = tan(x)-(x)==0;
S = vpasolve(eqn,x,[1,6])
answer: S =4.49340945790906417530788092728

syms x y   % 2-DOF nonlinear equations
eqn1 = x*sin(10*x) == y^3
eqn2 = y^2 == exp(-2*x/3)
[sol_x, sol_y] = vpasolve( [eqn1 eqn2], [x,y])
answer: sol_x =88.90707209659114864849280774681
sol_y = 0.0000000000000134704797106766943889737037
```

Write commands

format short (default)	6 significant figures
format long	15 digits
format short e	5 digits plus exponent
format long e	16 digits plus exponent
format rat	rational expression
format +	positive, negative, or zero

M-file functions

disp(ans)	display results without identifying variables
echo	control the command window and echo script commands
input	prompt user for input
keyboard	give control to keyboard
pause	pause until user types key
pause(n)	pause for (n) seconds
quit (or exit)	stops the execution of the program and ends MATLAB
return	stops the execution of the program and leaves MATLAB running
waitforbuttonpress	pause until user presses mouse or keyboard key

File Management

cd	show current directory

cd path	change to directory or folder given by path
dir	lists all files in current directory

Arrays and Matrices

x = cell(5, 100)	Returns an empty matrix X of size (5,100)
x = [1 2 3 4]	Creates a row vector by using spaces.
x = [1; 2; 3; 4]	Creates a column vector by using semi-colon.
x = [1 2; 3 4]	Create matrix using a combination of space and semi-colon
x(3)	Position (3) in the "x" array
x(1:5)	Address the five positions (1 through 5) in x
x(3,2,8)	Address 3, 5, 7 in x. x(first, increment, last)
x = s(2:5, 3:6)	Extract (4,4) matrix "x" from larger matrix "s" with rows 2 through 5 and columns 3 through 6
length(f)	Returns the size of vector (f)
linspace(first, last, n)	Create a row vector of size (n) where the magnitude of the terms are the linear interpolation from first to last
a = cell2mat(d)	Converts a cell array into an ordinary array
a = num2cell(d)	Converts array (d) into cell array (a)
a = size(x,1)	Returns the number of rows in matrix (x)
a = size(x,2)	Returns the number of columns in matrix (x)

Arrays and Matrix Multiplications

a+b	scalar or matrix addition
a-b	scalar or matrix subtraction
A*B	matrix multiplication
A.*B	matrix term-by-term multiplication
A./B	matrix term by term division
A'	transpose of (A)
det(A)	determinate of matrix
diag(a)	Diagonal matrix of vector (a) terms.
eye(x)	Identity matrix of size (x)
eig(A)	eigenvalues and eigenvectors [V,D] = eig(A)
inv(A)	inverse of matrix
max(A)	maximum value in (A)
min(A)	minimum value in (A)
prod(A)	product of all the elements in (A)
qz(A,B)	generalized eigenvalues
sum(A)	sum all the elements in (A)
trace(A)	sum of diagonal elements
zeros()	a matrix of all zeros. A = zero(5), a 5x5 matrix of zero
sparse()	develop a large sparse matrix

Eigenvalues / Eigenvectors – Sort then order results from smallest to largest

```
[Evect,Eigen]   = eig(A);             % Calculate eigenvalues and vectors
[EigenS,index]  = sort(diag(Eigen)); % Sort eigenvalues (small to large)
EvectS          = Evect(:,index);    % Reorder eigenvectors to match
```

Eigenvalues / Eigenvectors - Complex Conjugate Pairs– State Space

```
A = [0 , 1; -Kt/Is, -C/Is ];              % State-Space Form
[V,lam]  = eig(A);                        % Complex eigenvectors and eigenvalues
lam1r(1) = real(lam(1,1));                % Real portion of eigenvalue (#1)
lam2r(1) = real(lam(2,2));                % Real portion of eigenvalue (#2)
lam1i(1) = imag(lam(1,1));                % imaginary portion of eigenvalue (#1)
lam2i(1) = imag(lam(2,2));                % imaginary portion of eigenvalue (#2)
wn1(1)   = sqrt(lam1r(1)*lam1r(1) + lam1i(1)*lam1i(1)); % Undamped nat frequency
zeta1(1) = -lam1r(1)/wn1(1);                             % modal damping
```

Fast Fourier Transform (FFT) – (remember to clean data by subtracting RBM and apply Hanning window)

```
Fs   = 1000.0;       % Sampling Frequency (1000 Hz)
Per  = 1/Fs;         % Time period for each sample (0.001 second)
N    = 1500;         % Number of Samples
Time = 1500*Per;     % Total Time (1500*0.001 = 1.5 seconds)
T    = (0:N-1)*Per;  % Vector of time steps starting at t=0.0 to 1.499 secs
X                    % Measured signal at (N) time steps
Y    = fft(X);       % Fast fourier transform of signal (X)
PS2  = abs(Y/N);     % Two-sided power-spectrum
PS1  = PS2(1:N/2+1)  % Single-sided power spectrum based on even valued length
PS1(2:end-1) = 2*PS1(2:end-1);
f = Fs*(0:(N/2))/N;  % Frequency domain
plot(f,PS1);
title('Single-Sided Amplitude Spectrum of X');
xlabel('f (Hz)'), ylabel('PS1(X)');
```

Relational and Logical Operators

<	Less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
-=	not equal to
&	AND
|	OR
~	NPT

Control Flow

for loop (fixed times)

```
for n=1:10
  x(n) = n*n
end
```

nest for loop (fixed times)

```
for n = 1:10
  for m=1:10
    x(m,n) = n*m
  end
end
```

while loop (indefinite time)

```
eps = 1
```

```
while (1 + EPS)>1
  eps = 1 - eps / 10
end
```

break — Immediately terminates a "for" or "while" loop and exits the at the bottom (end) of that loop.

If-else-end

```
if (expression)
  Do commands if expression is true
else
  Do commands if expression is false
end
```

switch — A simple way of implementing multiple if-else-end statements by using the "case" "otherwise" and "end" commands. When the variable name is equal to the case number (or name), then that group of commands are performed, if the variable is not equal to any option, then the "otherwise" command is performed.

```
Switch variable
  case 1
     [perform this group of commands when the
     variable=1]
  case 2
     [perform this group of commands when the
     variable=2]
  case 99
     [perform this group of commands when the
     variable=99]
  otherwise
     [perform this group of commands when the
     variable is not equal to 1, 2, or 99]
  end
```

Excel Input (Read) File

inFile = 'file1.xlsx';	
[num, text,raw] = xlsread(infile)	In general in numbers or text, or raw (either numbers or text)
x = xlsread(inFile, 1, 'A10');	numerical variable (x) is taken from 1st worksheet position (A10)
y = xlsread(inFile, 1, 'D5:D10');	the 6-element array is taken from the 1st worksheet positions (D5 through D10) in the excel file
[~,title] = xlsread(inFile, 1, 'C2')	Read text from the 1st worksheet position (C2)
[~,~,a] = xlsread(inFile, 1, 'A1:C10)	Creates matrix (a) size (3,10) and fills the cells with "raw" data

Alternate. Read entire worksheet once instead of lots of individual reads, faster but all the data is in cell format so you need to use "cell2mat" to convert to real numbers. Also any Excel formatting (font, size) is destroyed

inFile = 'file1.xlsx';

insheet = readcell(inFile, 'Sheet',1) Read entire 1st worksheet from inFile.
insheet = readcell(inFile, 'Sheet', 'name') Instead of (1st) worksheet, (name) worksheet

Excel Output (Write) File

outFile = 'file2.xlsx'	
xlswrite(outFile, {'John'}, 1, 'A2')	write text to 1st worksheet in position (A2) in the output file
xlswrite(outFile, title, 1, 'C2')	write text in variable name test to 1st worksheet position (C2)
xlswrite(outFile, y, 1, 'D5')	write variable (y) to 1st worksheet position (D5)
xlswrite(outFile, y, 1, 'D5:D10')	write 6-element array (y) to 1st worksheet (D5 through D10)

Alternate. Write entire worksheet once instead of lots of individual writes, faster but you need to convert all data to cell format using "num2cell" then use **writecell**. Alternatively you can write real numbers using **writematrix**. This approach changes all Excel formatting (font, size) back to default.

```
writecell(variableName, outFile,'Sheet','SheetName','Range','C2')
writematrix(variableName, outFile,'Sheet','SheetName','Range','C2')
```

2-D Plotting

plot(x,y) 2-dimensional plot
plot(x1,y1,'ko',x2,y2,'b+') plot two sets of data on one figure with different symbols

Colors:

```
k   black (default)
b   blue
c   cyan
g   green
m   magenta
r   red
w   white
y   yellow
```

Line Types:

```
-   solid line (default)
--  dashed line
-.  Dash-dotted line
:   Dotted line
```

Data Markers:

```
.   dot (.)
*   asterisk (*)
x   cross (x)
o   circle (o)
+   plus sign (+)
s   square
d   diamond
p   five-pointed star
```

axis([xmin xmax ymin ymax])	user defined plot bounds
axis square	resizes axes so plot is square (circles, may look like ovals)
axis equal	resizes axes so x and y are the same size (circles are correct)
grid on	put gridlines on plot

legend('name')
xlabel('xlabel')	label along x-axis
ylabel('ylabel')	label along y-axis
title('title')	plot title (DOES NOT WORK)

hold on / hold off	CRITICAL for multiple commands and data sets on a plot

loglog(x,y)	log-log plot
semilogx(x,y)	semi-log plot along x-axis
semilogy(x,y)	semi-log plot along y-axis
plotyy(x1,y1,'o',x2,y2,'+')	plot with (y1) axis on left and (y2) axis on right
errorbar(x,y,error)	plot x vs y with error bars.

3-D Mohr's Circles (2-D Plots)

```
axis equal;
grid;
xlabel('sigma (Ksi)'), ylabel('tau (Ksi)');
viscircles(Center1, Radius1);
viscircles(Center2, Radius2);
viscircles(Center3, Radius3);
```

3-D Plotting

plot3(x,y,z)	3-dimensional line plot
plot3(x1,y1,'o',x2,y2,'+')	plot two sets of data on one figure with different symbols
axis([xmin xmax ymin ymax])	user defined plot bounds
grid	put gridlines on plot
legend('name')	
xlabel('xlabel')	label along x-axis
ylabel('ylabel')	label along y-axis
zlabel('ylabel')	label along z-axis
title('title')	plot title
meshgrid(x,y)	define (x,y) grid with (x = xmin, xspacing, xmax) and (y = ymin, yspacing, ymax)
mesh(x,y,z)	generates 3-D surface plot
contour(x,y,z)	generates 3-D surface plot
meshC(x,y,z)	generates 3-D surface plot with a contour plot underneath
meshZ(x,y,z)	generates 3-D surface plot with vertical (z) lines to x-y-plane

Functions

A user-defined *function* (subroutines) is used to group repetitive calculations. The main (M) file calls a function that can have multiple input variables and multiple output variables. All of your many functions are usually written as separate (m) files from the main code, but they must all be in the same directory. They can also be included at the end of the main code by simply transforming the main (m) code into a function (i.e., add a function name above the 1st line and an "end" after the last line) before appending your functions. The benefit of including everything into one (m) file, is that the resulting p-code is only one file. In general, (MyFunction) has the general form:

[OutputVariables] = MyFunction(InputVariables)

Example:

```
function [F] = Quadratic(x,A,B,C)
% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
% .     This function calculates (F), which is the right side of a
% .     quadratic equation, where (A,B,C) are constants and (x) is
% .     a variable.  If (x) is a quadratic equation solution, then
% .     (F = 0).
% .                          A*x^2 + B*x + C = F
% .     Input:
% .     A, B, C                 coefficients
% .     x                       variable
% .
% .     Output:
% .     F                       calculated output
% .
% .     MATLAB program call:
% .     [F] = Quadratic(x,A,B,C)
% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

    F = A*x*x + B*x + C

End
```

Solve a (1-DOF) Nonlinear Equation

A nonlinear (1-DOF) equation can be solved using (vpasolve), where the equation (F) to be solved must be written in the general form (F(x) == 0), where the returned value of (x) finds (F(x) = 0) nearest the starting point (Xo). Since (vpasolve) can only solve a (1-DOF) nonlinear equation, the anonymous function %(x) is used to define that (x) is the variable and everything are constants.

Example: Find a root of: sin(x) + x*x = 3, nearest to (x = 1).

```
Xo =  1.0                                   % Starting Point (Xo)
syms X                                      % define symbolic variable x
s = vpasolve( sin(x) + x*x -3 == 0, x, Xo)  % calculated root (s) of equation
```

Example: Find a root of F(x) = 0.7x^2 + 5x – 3, nearest to (x = 1).

```
%  Define Polynomial Constants (A,B,C) for a function called “Quadratic”
A  =  0.7
B  =  5.0
C  = -3.0
Xo =  1.0                                  % Starting Point (Xo)
X  = vpasolve(@(x) Quadratic(x,A,B,C), Xo)      % Find (X) closest to (Xo)
```

Solve a Nonlinear Equation (or Set of Equations)

S = vpasolve(function == 0, x, xo)   nonlinear solver, solve systems of equations. Define a function As “n” equations and this will return a vector (X) of “n” solutions

P-Code (Protected Code)

The (m) code can be converted to a protected (P) version so that the text and commands can not be shared. The m-code can be a program or a function. Function makes more sense. To create a p-code, in the command line (not the editor), type:

>> pcode myprogram.m

The program is returned in the same directory with the name "myprogram.p" If the original (m) file was a program, then it can be run in the command line by typing:

>> run myprogram.p

If the original program was a function, then the P-file is placed in the directory and used as any regular protected MATLAB function.

Executable File

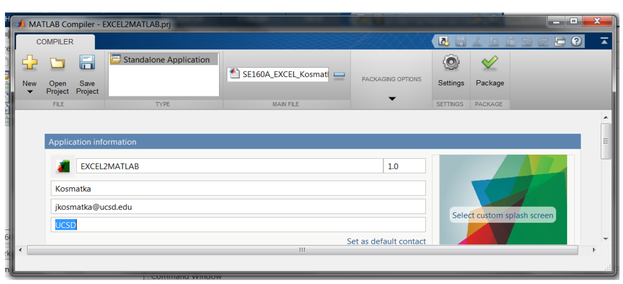
The (m) code can be transformed into an executable (*.exe) file that can be run on any computer as long as the latest version of the MATLAB Runtime library is installed on that computer. The MATLAB Runtime is a standalone set of shared numerical libraries.

STEP 1: Transform (m)-file to executable (*.exe) file

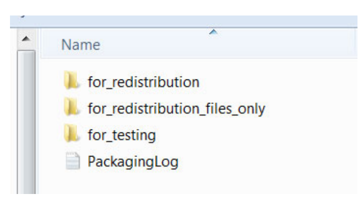
In the command line (not the editor), type:

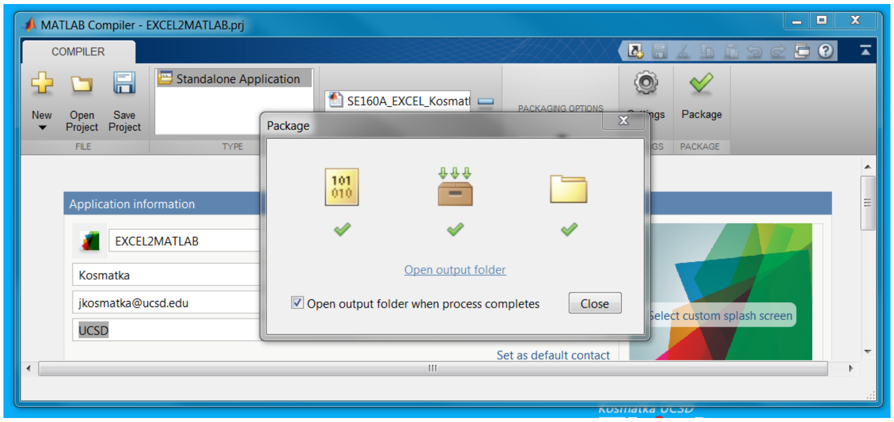
>> deploytool

At the pop-up windows, click on "Application Compiler" then define the project in the next window, select "**Standalone Application**" and type "**Application Name** (*.exe file name)," "Author", etc. Then add main file from (m)-file list. In "**Packaging Options**" select *Runtime included in Package*. Note: if your program requires (m) or (p) functions, then these needed to be attached now. Click "**Package**" to run. Takes time to run (3 green check marks).



Click on "Open output folder." The executable file is on the folder entitled; "for_redistribution_files_only" Drag executable file and any other needed files (EXCEL input and output files) into a new folder. Run executable file. If the program doesn't run then go to step (2). If you wish to distribute the program to other computers that do not have MATLAB installed, then go to step (2).





STEP 2: Install latest MATLAB Runtime library if program doesn't run on computer.

2.1) Go to the MATLAB website and download the latest version (free) of the MATLAB Runtime library for the computers operating system (32-bit, 64-bit, Linux, Mac).

http://www.mathworks.com/products/compiler/mcr/index.html

2.2) Install downloaded MCRInstaller.exe. NOTE: You will need administrator rights to run MCRInstaller.

2.3) Run executable file.