MATLAB Bootcamps 1, 2 and 3

- 1: Getting up to speed (or back up to speed) with MATLAB
- 2: Learning to use MATLAB to solve typical problem scenarios
- 3: Detailed modeling of packed-bed and plug-flow reactors

Bootcamp 1 Outline

Prerequisite: install MATLAB on your computer

https://oit.colorado.edu/software-hardware/software-catalog

MATLAB

Description:

Mathematical matrix lab for interactive computation

Operating System:

Windows macOS Linux

Campuses:

Boulder

Eligibility:

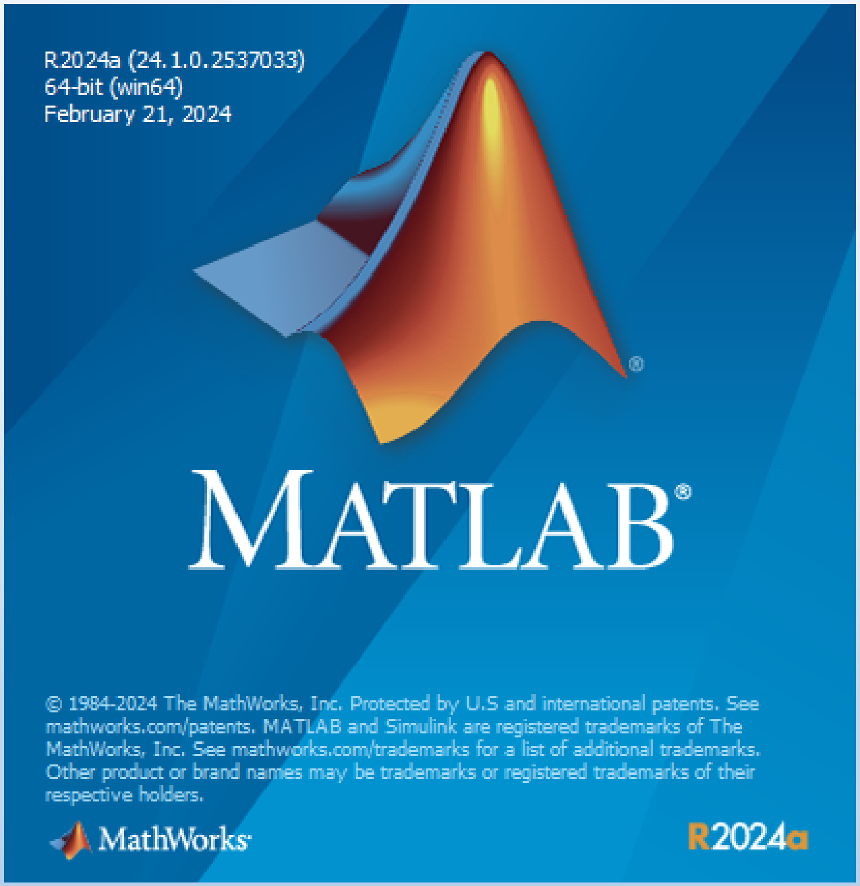
No fee to end-user (Department, Faculty/Staff, or Student)

Toolboxes typically included in the installation:

Simulink, Control System, Curve Fitting, Optimization, Signal Processing, Spreadsheet Link, Symbolic Math, Statistics and Machine Learning, System Identification

You can include all toolboxes. That just requires more memory.

MATLAB launch window

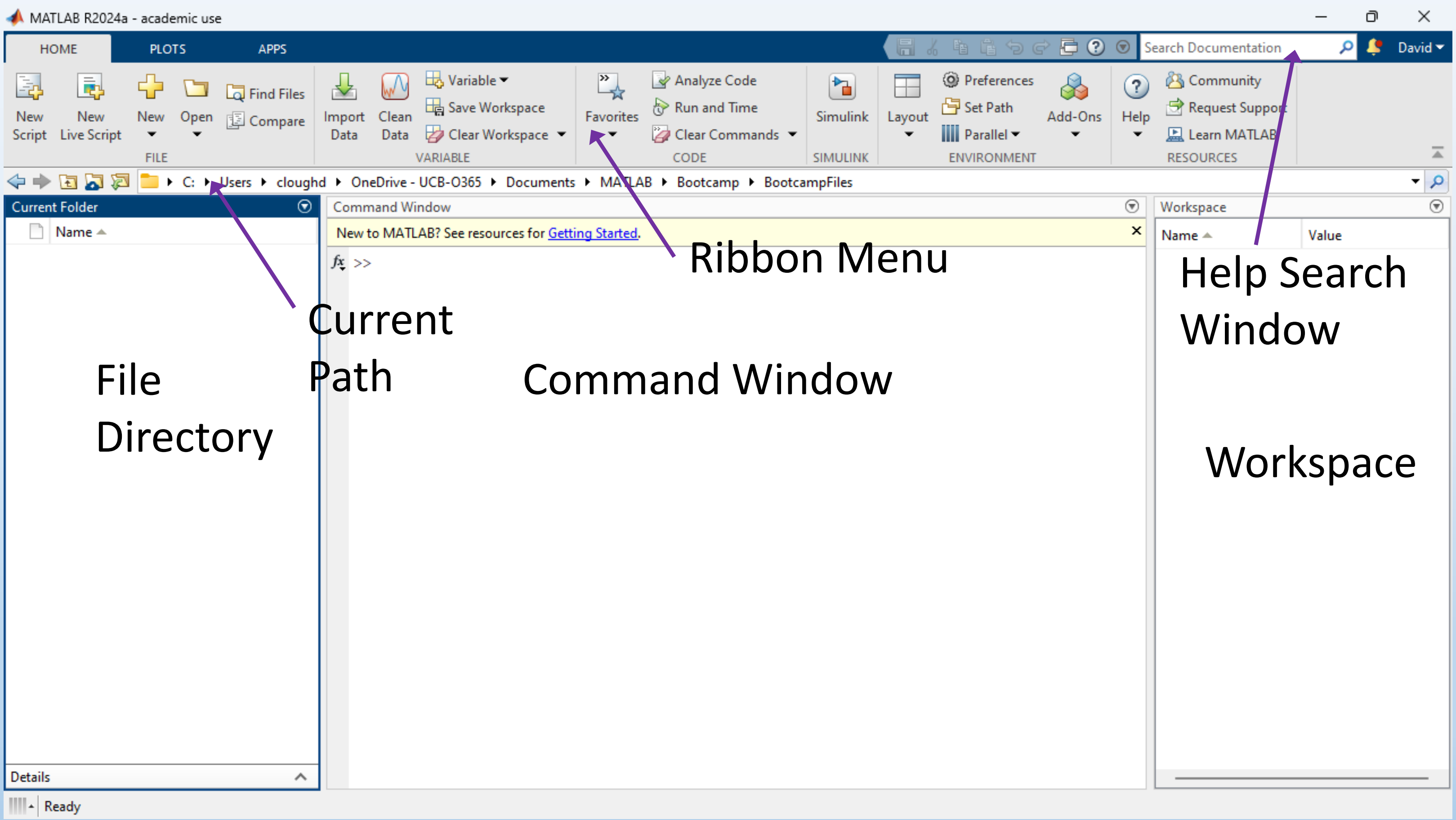


MATLAB is an abbreviation of Matrix Laboratory

MATLAB is a product of Mathworks, Inc.
It was pioneered by Cleve Moler (https://en.wikipedia.org/wiki/Cleve_Moler) in the 1970s and based originally on the Fortran programming language.

Initial MATLAB application window

(based on MATLAB Version 2024a)



Checking the MATLAB version and toolboxes installed

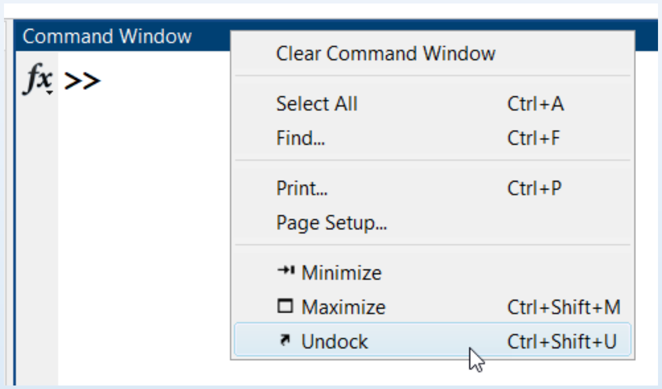
Command Window

New to MATLAB? See resources for Getting Started.

```
>> ver
-----------------------------------------------------------------------------------------------------
MATLAB Version: 24.1.0.2537033 (R2024a)
MATLAB License Number: 361635
Operating System: Microsoft Windows 11 Enterprise Version 10.0 (Build 22621)
Java Version: Java 1.8.0_202-b08 with Oracle Corporation Java HotSpot(TM) 64-Bit Serve
-----------------------------------------------------------------------------------------------------
MATLAB                                                      Version 24.1        (R2024a)
Simulink                                                    Version 24.1        (R2024a)
```

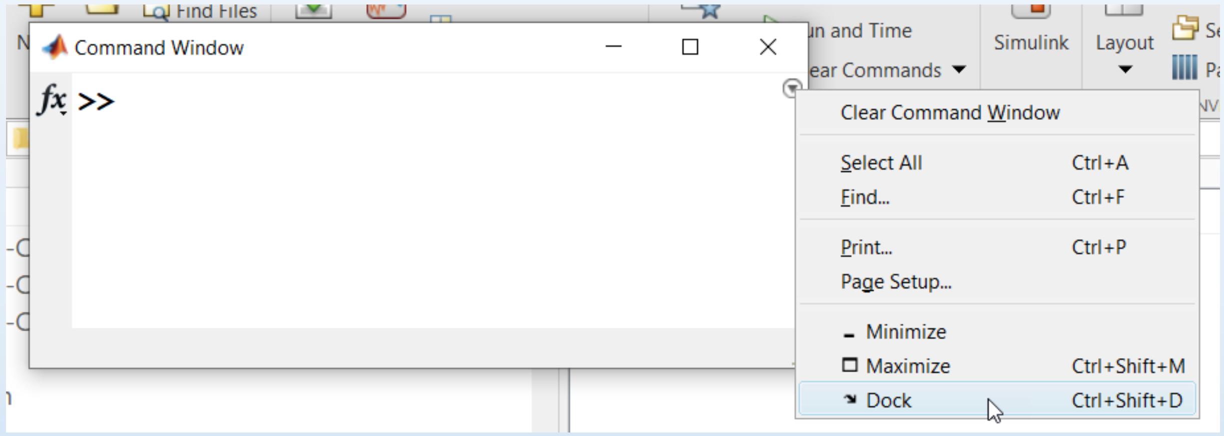
MATLAB window management

Undocking a window



Ctrl-Shift-U

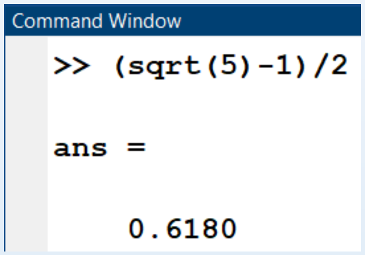
Docking a window



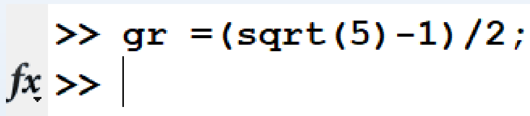
Ctrl-Shift-D

Command Window

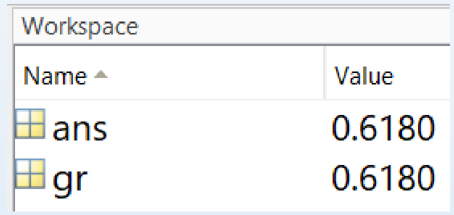
Entering commands directly in the Command Window



calculator

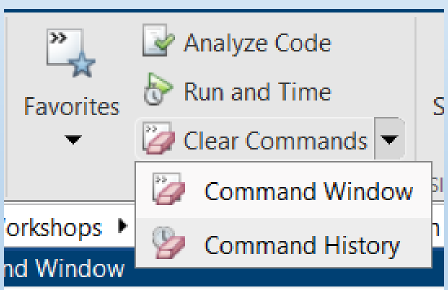


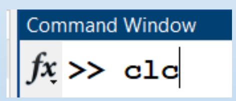
assignment with no echo, using ;



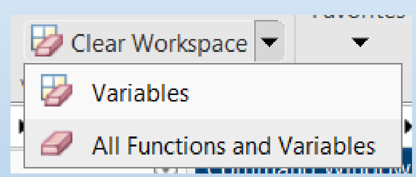
workspace

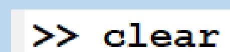
Clearing commands





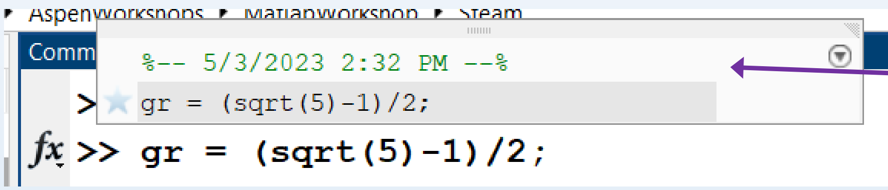
Command Window





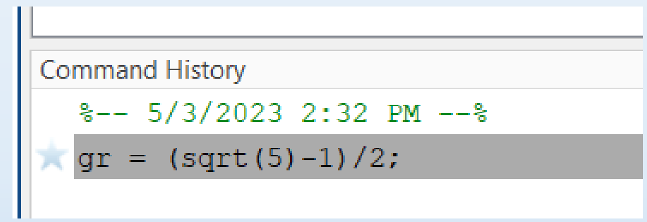
Command Window

Retrieving previous commands with the up arrow, ↑



Command History popup shows

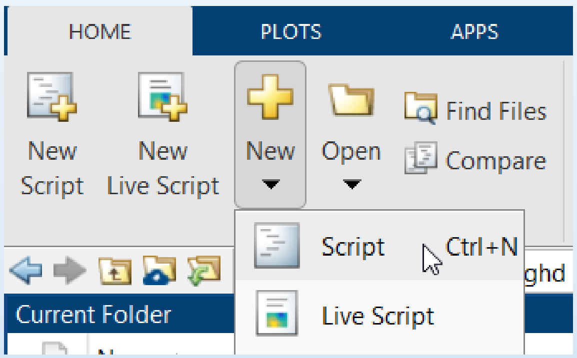
can dock, Ctrl-Shift-D

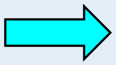


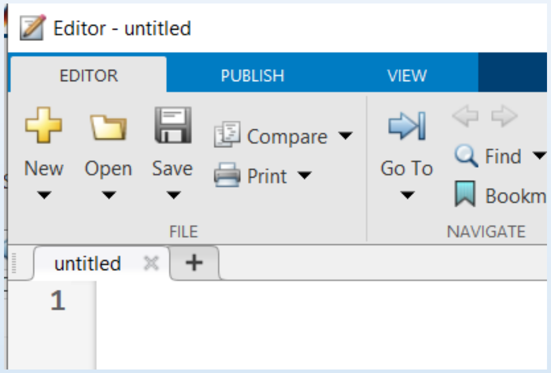
Can edit the command and press Enter from anywhere in the command.

Creating MATLAB Scripts

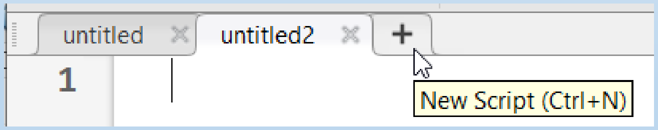
Opening an Editor window for a script





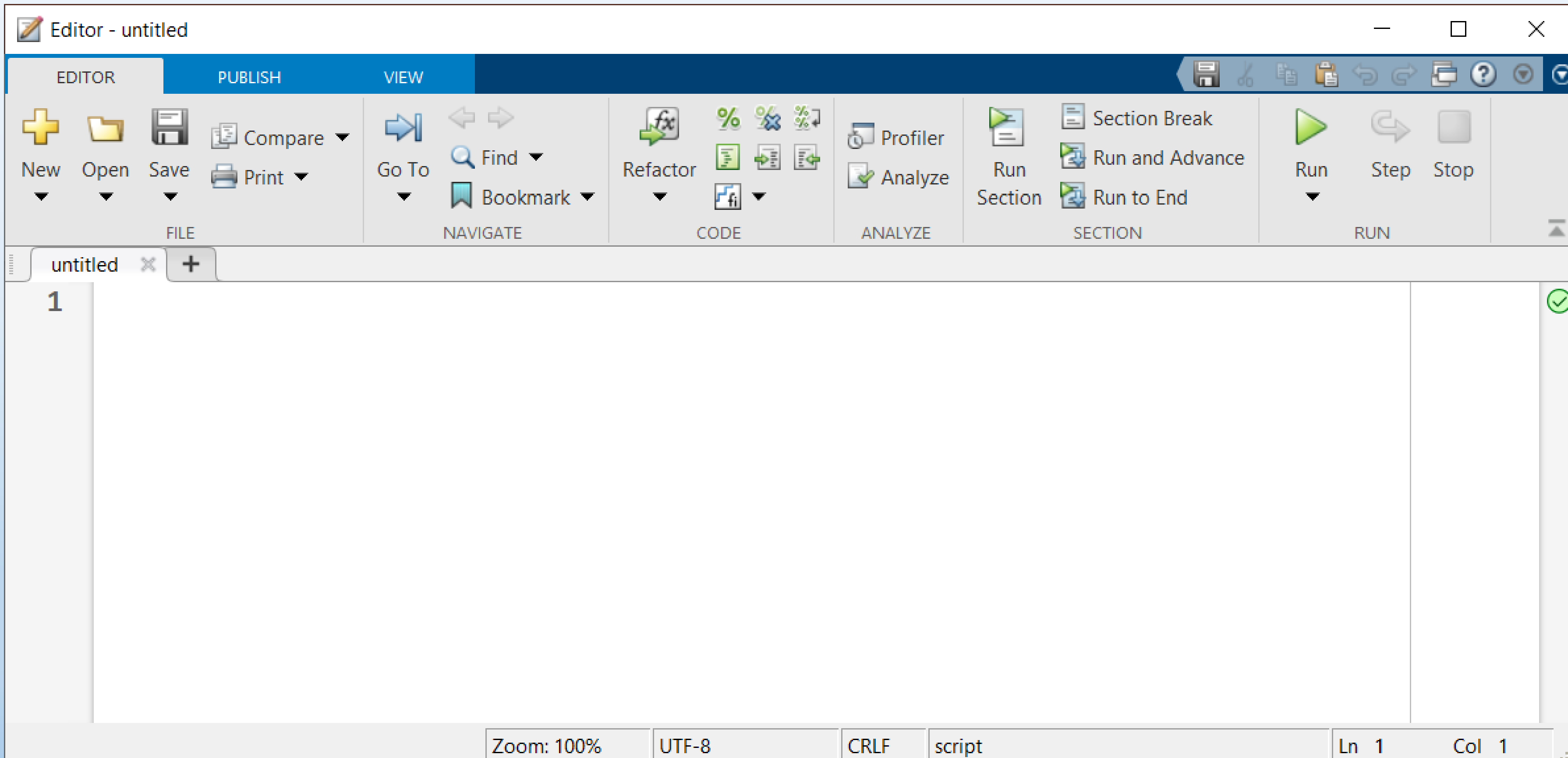


Multiple scripts open, each with its own tab

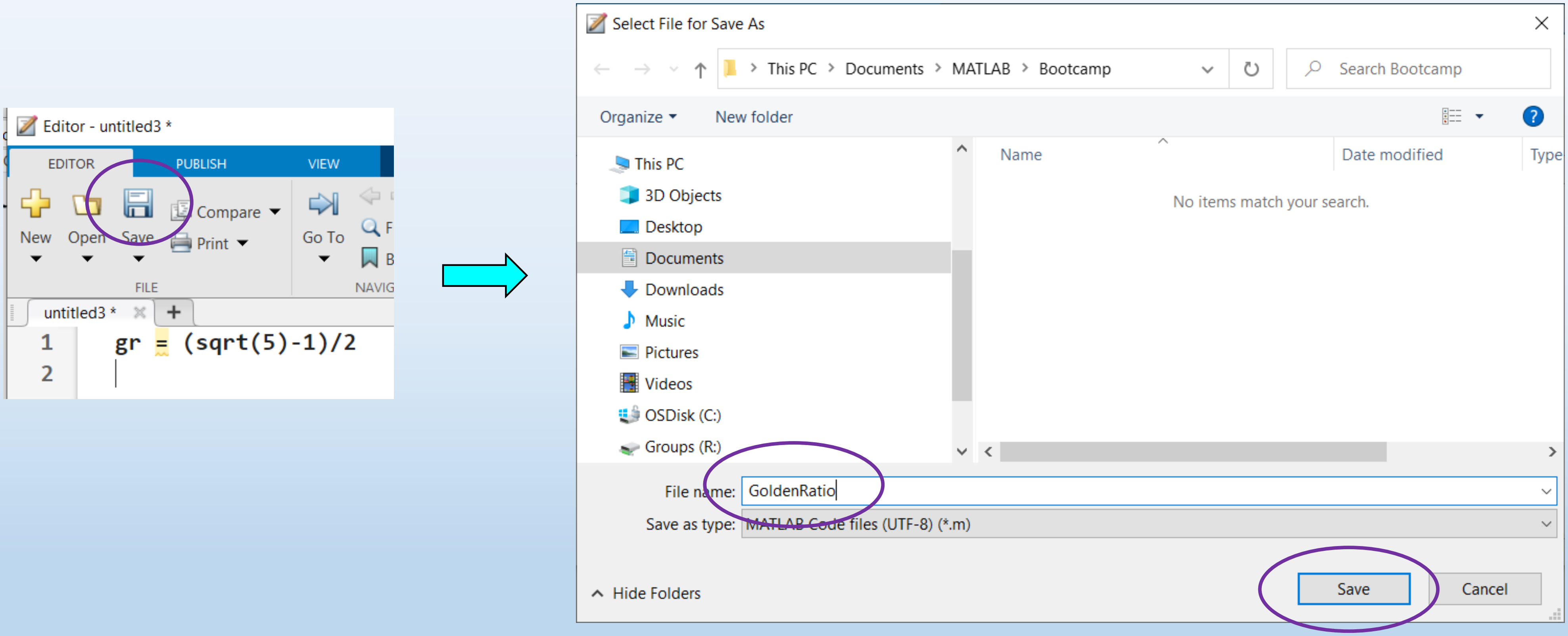


MATLAB Editor Window

Operates as a simple text editor like Windows Notepad or Wordpad



Saving a script as an m-file



Simple MATLAB Expressions

- Expressions are evaluated left to right
- Parentheses can modify the order of evaluation
- Functions in expressions are evaluated first, e.g., sqrt(•)

Arithmetic operators in order of precedence

^	exponentiation
–	unary minus, negation (not subtraction)
*, /	multiplication, division
+, –	addition, subtraction

```
>> -3^2

ans =

    -9
```

```
>> 3/2*4

ans =

     6
```

```
>> 3/2/4

ans =

    0.3750
```

Logical operators, not, and, or, etc. introduced later.

Relational operators – yield a T/F result, 1 or 0

==	equality	<	less than
>	greater than	>=	less than or equal to
>=	greater than or equal to	<>	not equal

MATLAB Arrays

One-dimensional row vector
1 x 4

```
>> [ 12 -3 5 97.6 ]

ans =

   12.0000   -3.0000    5.0000   97.6000
```

Can separate by commas. ,

One-dimensional column vector
4 x 1

```
>> [ -4.5 ; 12.6 ; 44 ; -0.1 ]

ans =

   -4.5000
   12.6000
   44.0000
   -0.1000
```

Two-dimensional matrix
3 x 2

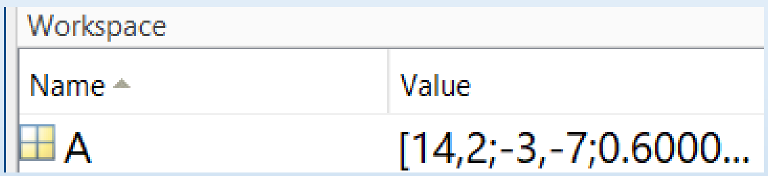
```
>> [ 14 2 ; -3 -7 ; 0.6 -3.4 ]

ans =

   14.0000    2.0000
   -3.0000   -7.0000
    0.6000   -3.4000
```

Assigning Arrays and Examining in the Workspace and Spreadsheet

```
>> A = [ 14 2 ; -3 -7 ; 0.6 -3.4 ];
```

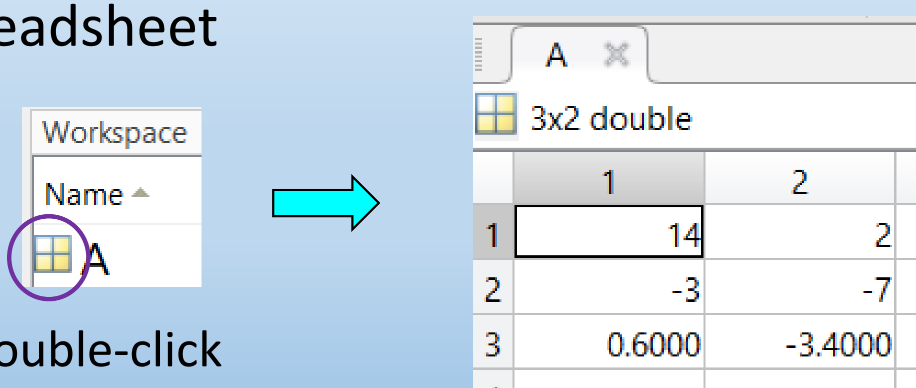
Although not required, we typically assign matrices to capital letters (or names beginning with capitals) and vectors to lower-case letters (or names beginning with lower-case letters).

Workspace



A and a are different variable names in MATLAB. Variable names are "case sensitive."

Spreadsheet



Double-click

Spreadsheet can be docked and undocked.

Values can be modified directly.

Creating vectors with the colon (:) operator

start : interval : end   if interval left out, assumed to be 1

```
>> x = 1:5

x =

     1     2     3     4     5
```

```
>> y = 5:-1:1

y =

     5     4     3     2     1
```

```
>> z = 0:0.2:1

z =

  Columns 1 through 5

         0    0.2000    0.4000    0.6000    0.8000

  Column 6

    1.0000
```

```
>> w = 0:3:7

w =

     0     3     6
```

Creating vectors with the *linspace*(•) and *logspace*(•)

```
>> a = linspace(0,10,11)

a =

  Columns 1 through 9

     0     1     2     3     4     5     6     7     8

  Columns 10 through 11

     9    10
```

linspace(start,end,number)

create 11 equally spaced numbers from 0 to 10

leave *number* out, default is 100

```
>> a = logspace(0,1,11)

a =

  Columns 1 through 5

    1.0000    1.2589    1.5849    1.9953    2.5119

  Columns 6 through 10

    3.1623    3.9811    5.0119    6.3096    7.9433

  Column 11

   10.0000
```

logspace(start exponent,end exponent,number)

create 11 numbers equally spaced by logarithm base 10 from 10^0 to 10^1

Referencing Elements of Vectors and Matrices

```
>> A = [ 14 2 ; -3 -7 ; 0.6 -3.4 ];
>> A(2,2)

ans =

    -7
```

A(i,j) element in i^{th} row and j^{th} column

```
>> A(1,:)

ans =

    14     2
```

first row, all columns

use of the colon (:) operator

```
>> A(:,2)

ans =

    2.0000
   -7.0000
   -3.4000
```

all rows, second column

```
>> b = [ 12 -3 5 97.6 ];
>> b(2:end)

ans =

   -3.0000    5.0000   97.6000
```

Vector-Matrix Operations

Addition and Subtraction are item-by-item

```
>> a = [ 1 2 3 4];
>> b = [ 6 7 8 9];
>> a+b

ans =

     7     9    11    13
```

Dimensions must be compatible

```
>> c = [ 5 7 9 ];
>> a - c
Arrays have incompatible sizes for this operation.
```

Vector-Matrix Operations

Addition and Subtraction with scalars

```
>> a + 3

ans =

     4     5     6     7
```

Scalar added to each element of a

Addition and Subtraction with compatible vectors/matrices

```
>> c = [ 5 ; 7 ; 9 ; 11 ];
>> a - c

ans =

    -4    -3    -2    -1
    -6    -5    -4    -3
    -8    -7    -6    -5
   -10    -9    -8    -7
```

c subtracted from each element of a
(this gets tricky!)

Vector-Matrix Multiplication

Inner product of two vectors

```
a =

     1     2     3     4
```

```
b =

     6
     7
     8
     9
```

```
>> a*b

ans =

    80
```

$$\begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n = \sum_{i=1}^{n} a_i b_i$$

$1 \times n \cdot n \times 1 \Rightarrow 1 \times 1$

must agree

Outer product of two vectors

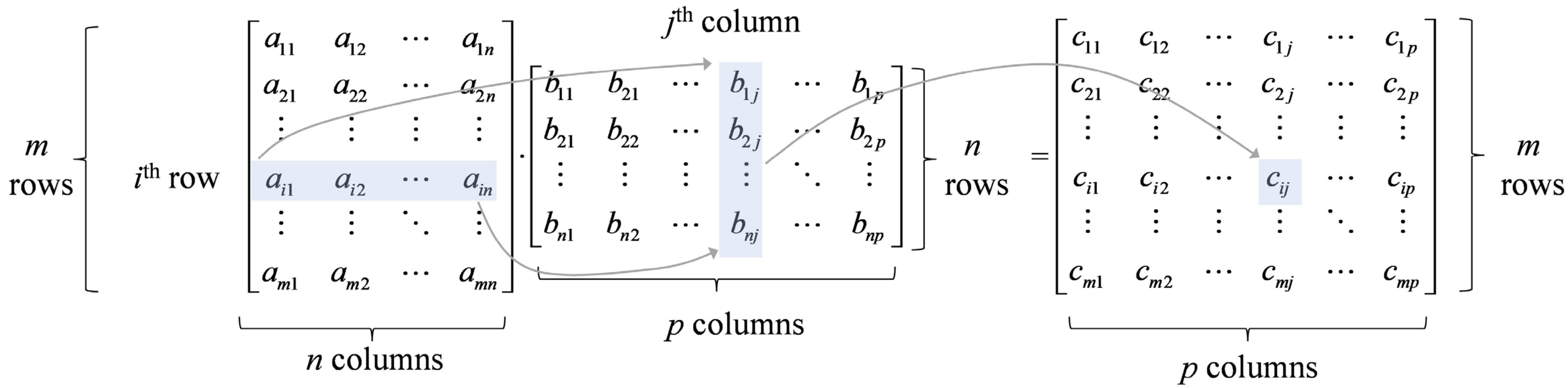
```
b =

     6
     7
     8
     9
```

```
a =

     1     2     3     4
```

```
>> b*a

ans =

     6    12    18    24
     7    14    21    28
     8    16    24    32
     9    18    27    36
```

$$ba_{ij} = b_i a_j \qquad i = 1, \ldots, n \; ; \; j = 1, \ldots, n$$

Vector-Matrix Multiplication



$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots a_{in}b_{nj}$$
$$= \sum_{k=1}^{n} a_{ik}b_{kj}$$

$$[m\times n]\cdot[n\times p]\Rightarrow[m\times p]$$

```
A =

   20.0000    2.0000
   -3.0000   -7.0000
    0.6000   -3.4000
```

```
B =

     2     3     5     1
    -4     6    -8     2
```

```
>> A*B

ans =

   32.0000   72.0000   84.0000   24.0000
   22.0000  -51.0000   41.0000  -17.0000
   14.8000  -18.6000   30.2000   -6.2000
```

Vector-Matrix Operations

Transpose (rows become columns and vice versa)

```
A =

   20.0000    2.0000
   -3.0000   -7.0000
    0.6000   -3.4000
```

```
>> A'

ans =

   20.0000   -3.0000    0.6000
    2.0000   -7.0000   -3.4000
```

Use the right apostrophe, ‘ , as the transpose operator.

```
>> [ 1 2 3 4]'

ans =

     1
     2
     3
     4
```

```
a =

     1     2     3     4
```

```
>> a*a'

ans =

    30
```

Inner product of a row vector with its transpose is sum of squares of the elements.

Vector-Matrix Operations

Special Matrices

Identity matrix

$$\begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

```
>> eye(4)

ans =

     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

Diagonal square matrix

$$\begin{bmatrix} d_1 & 0 & \cdots & \cdots & 0 \\ 0 & d_2 & 0 & \cdots & 0 \\ 0 & 0 & d_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & d_n \end{bmatrix}$$

```
>> d = 2;
>> d*eye(4)

ans =

     2     0     0     0
     0     2     0     0
     0     0     2     0
     0     0     0     2
```

Vector-Matrix Operations

Inverse of a Square Matrix – $\mathbf{A}^{-1}$

$$\mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{I}$$

```
A =

     1     5     6
     2     9    -1
     3    -7     4
```

```
>> Ainv = A^(-1)

Ainv =

   -0.1066    0.2279    0.2169
    0.0404    0.0515   -0.0478
    0.1507   -0.0809    0.0037
```

```
>> A*Ainv

ans =

    1.0000         0    0.0000
    0.0000    1.0000   -0.0000
    0.0000   -0.0000    1.0000
```

There is also a built-in function to compute the inverse, *inv*(•)

```
>> inv(A)

ans =

   -0.1066    0.2279    0.2169
    0.0404    0.0515   -0.0478
    0.1507   -0.0809    0.0037
```

There are various numerical methods that can be used to compute the inverse of a matrix. The *inv*(•) function uses a method based on "LU decomposition."

Array Operations

item-by-item calculations using dot operators

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

A =

     1     5     6
     2     9    -1
     3    -7     4
```

```
>> 1 ./ A

ans =

    1.0000    0.2000    0.1667
    0.5000    0.1111   -1.0000
    0.3333   -0.1429    0.2500
```

```
>> b = [ 2 5 -8 ]'

b =

     2
     5
    -8
```

```
>> A .* b

ans =

     2    10    12
    10    45    -5
   -24    56   -32
```

Vectorizing a Calculation

$$y = 4 + 0.2x - 0.7x^2 + \frac{1}{x}$$

```
>> x = 0.1:0.1:0.9;
>> y = 4 + 0.2*x -0.7*x.^2+1./x

y =

  Columns 1 through 5

   14.0130    9.0120    7.3303    6.4680    5.9250

  Columns 6 through 9

    5.5347    5.2256    4.9620    4.7241
```

Common Mathematical Functions

abs(•)	absolute value
log(•)	natural logarithm
log10(•)	base 10 logarithm
log2(•)	base 2 logarithm
exp(•)	exponential, *e* to the power
sign(•)	signum, 1 if positive, -1 if negative, 0 if zero

sinh(•)	hyperbolic sine
cosh(•)	hyperbolic cosine
tanh(•)	hyperbolic tangent
asinh(•)	inverse hyperbolic sine
acosh(•)	inverse hyperbolic cosine
atanh(•)	inverse hyperbolic tangent

sin(•)	sine	arguments in radians
cos(•)	cosine	
tan(•)	tangent	
atan(•)	arctangent	results in radians
asin(•)	arcsine	
acos(•)	arccosine	

rad2deg(•)	convert radians to degrees
deg2rad(•)	convert degrees to radians

atan2(x,y)	4-quadrant arctangent
abs(c)	magnitude of complex no.
angle(c)	polar angle of complex no.
real(c)	real part of complex no.
imag(c)	imaginary part of complex no.

Constants: pi value of π eps machine epsilon i , j imaginary unit, $\sqrt{-1}$

Output Display

```
>> A

A =

     1     5     6
     2     9    -1
     3    -7     4
```

display variable value(s) in the Command window

```
>> disp(A)
     1     5     6
     2     9    -1
     3    -7     4
```

use the *disp* function

```
>> disp([x' y'])
    0.1000   14.0130
    0.2000    9.0120
    0.3000    7.3303
    0.4000    6.4680
    0.5000    5.9250
    0.6000    5.5347
    0.7000    5.2256
    0.8000    4.9620
    0.9000    4.7241
```

two vectors

```
>> disp('      x          y');disp([x' y'])
      x          y
    0.1000   14.0130
    0.2000    9.0120
    0.3000    7.3303
    0.4000    6.4680
    0.5000    5.9250
    0.6000    5.5347
    0.7000    5.2256
    0.8000    4.9620
    0.9000    4.7241
```

with headings

The format Statement

to control the display of numerical quantities in the Command window

Example: format short

this is the default
4 digits to the right of the decimal point

long	15 digits after the decimal point (7 for single types)
shortE	short with scientific notation
longE	long with scientific notation
shortG	short fixed or scientific, whichever is more compact
longG	long fixed or scientific, whichever is more compact
shortEng	short with exponent a multiple of 3
longEng	long with exponent a multiple of 3
+	positive/negative signs displayed
bank	currency with two digits after the decimal point
hex	hexadecimal representation of a binary number
rational	ratio of integers

The format Statement

to control line spacing

format loose	add blank lines for readability – this is the default
format compact	suppress excess blank lines

to reset to all format default settings

format default

Output with the *fprintf* Statement

to control format on the display and write data to an external file similar to C/C++

for output to the Command Window

fprintf(formatSpec,variable list)

Example

```
>> TF = 450;
>> TC = (TF-32)/1.8;
>> formatSpec = '%5.1f degF = %5.1f degC\n';
>> fprintf(formatSpec,TF,TC)
450.0 degF = 232.2 degC
```

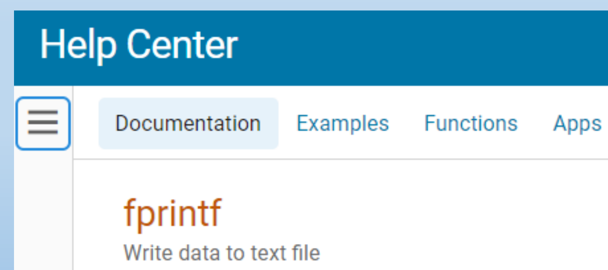
floating point

%5.1f

field width

decimal places displayed

Details are available in Help



Output with the *fprintf* Statement

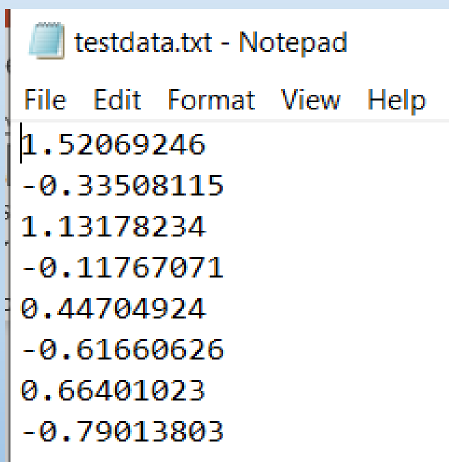
for output to a file

fprintf(fileID,formatSpec,variable list)

Example

```
>> rn = randn(5000,1);
>> formatSpec = '%10.8f\n';
>> fileID = fopen('testdata.txt','w');
>> fprintf(fileID,formatSpec,rn);
>> fclose(fileID);
```

randn generates 5000 random numbers from a standard normal distribution


testdata.txt - Notepad

File Edit Format View Help

```
1.52069246
-0.33508115
1.13178234
-0.11767071
0.44704924
-0.61660626
0.66401023
-0.79013803
```

Input from the User

numerical

variable = input(prompt string)

string

variable = input(prompt string,’s’)

InputExamples.m

```
clear
x = input('enter a value for x: ');
ans = input('enter Y or N: ','s');
```

```
>> InputExamples
enter a value for x: 23
enter Y or N: Y
```

Workspace

Name	Value	Size
ans	'Y'	1x1
x	23	1x1

Text File Input with the *fscanf* Statement

y = *fscanf*(fileID,formatSpec)

Example

```
>> formatSpec = '%f';
>> fileID = fopen('testdata.txt','r');
>> rn1 = fscanf(fileID,formatSpec);
```

%f for general floating-point numerical format

'r' for read from file

rn1 *5000x1 double*

5000x1 double

	1
1	1.5207
2	-0.3351
3	1.1318
4	-0.1177
5	0.4470
6	-0.6166
7	0.6640

More details available in Help

Saving and Loading Workspace

```
>> save('testfile')
```

Workspace

Name	Value	Size

```
>> load('testfile')
```

Workspace

Name	Value	Size
CO2	*296x1 double*	296x1
Fdata	*296x3 double*	296x3
fileID	4	1x1
formatSpec	'%f'	1x2
fuel	*296x1 double*	296x1
sizeFurnace	[3,Inf]	1x2
ts	*296x1 double*	296x1
x	*1x100 double*	1x100

Can reload the workspace during a subsequent work session to pick up where you left off.

Program Structure - Selection (If structures)

One-line One-way If

if condition; statement ; end

```
x = input('enter a value for x:');
Sgnx2 = x^2;
if x < 0 ; Sgnx2 = -Sgnx2 ; end
disp(Sgnx2)
```

```
>> OneLineIfExample
enter a value for x:-2
    -4

>> OneLineIfExample
enter a value for x:2
     4
```

OneLineIfExample.m

Program Structure - Selection (If structures)

Multple-line One-way If

```
if condition
   statement(s)
end
```

```
a = input('enter a value for a:');
b = input('enter a value for b:');
if a < b ;
    temp = a;
    a = b;
    b =temp;
end
disp([a b])
```

```
>> OneWayIfExample
enter a value for a:2
enter a value for b:4
     4     2
```

OneWayIfExample.m

Program Structure - Selection (If structures)

Two-way If (if else)

```
if condition
   statement(s)
else
   statement(s)
end
```

```
x = input('enter a value for x:');
if x > 0;
    lx = log(x);
else
    lx = log(-x);
end
disp(lx)
```

```
>> TwoWayIf_Example
enter a value for x:-0.5
   -0.6931

>> TwoWayIf_Example
enter a value for x:12
    2.4849
```

TwoWayIf_Example.m

Program Structure - Selection (If structures)

Multi-alternative If (if elseif)

```
if condition1
   statement(s)
elseif condition2
   statement(s)
elseif condition3
   statement(s)
•
•
else
   statement(s)
end
```

```
x = input('enter a value for x:');
if x > 1;
    xlim = 1;
elseif x < -1;
    xlim = -1;
else
    xlim = x;
end
disp([x xlim])
```

```
>> ClampX_Example
enter a value for x:-1.5
   -1.5000   -1.0000

>> ClampX_Example
enter a value for x:2.5
    2.5000    1.0000

>> ClampX_Example
enter a value for x:-0.3
   -0.3000   -0.3000
```

ClampX_Example.m

Program Structure - Selection

the switch structure (also called the select-case structure)

```
switch switch_expression
   case case_expression1
      statement(s)
   case case_expression2
      statement(s)
•
•
   otherwise
      statement(s)
end
```

SwitchExample.m

```
TF = input('\nenter temperature in degF: ');
Un = input('\nenter C, K, or R: ','s');
switch Un
case 'C'
    TC = (TF-32)/1.8;
    fprintf('\ntemp in degC = %6.2f\n',TC)
case 'K'
    TK = (TF-32)/1.8 + 273.15;
    fprintf('\ntemp in K = %6.2f\n',TK)
case 'R'
    TR = TF + 459.67;
    fprintf('\ntemp in degR = %6.2f\n',TR)
end
```

```
>> TempConvert

enter temperature in degF: -100

enter C, K, or R: R

temp in degR = 360.00
```

Program Structure - Repetition

the for loop structure

count-controlled repetition

```
for index = start:increment:end
   statement(s)
end
```

increment = 1 if left out

the break option

```
for index = start:increment:end
   statement(s)
   if condition ; break ; end
   statement(s)
end
```

premature exit from a for loop

the continue option

```
for index = start:increment:end
   statement(s)
   if condition ; continue ; end
   statement(s)
end
```

premature cycling in a for loop

Program Structure - Repetition

the for loop structure – computing the median absolute deviation (MAD)

```
x = [10.1,11.5,9.6,9.6,10.4,9.4,10.2,9.9,9.1,9.8];
xmed = median(x);
n = length(x);
for i = 1:n
    xdev(i) = abs(x(i) - xmed);
end
xMAD = median(xdev)/0.6745;
disp(xMAD)
```

```
>> MAD
    0.4448
```

MAD1.m

<u>Note</u>: when possible look for the opportunity to use vector/matrix operations in lieu of for loops

```
x = [10.1,11.5,9.6,9.6,10.4,9.4,10.2,9.9,9.1,9.8];
xmed = median(x);
xdev = abs(x-xmed);  % vector operation
xMAD = median(xdev)/0.6745;
disp(xMAD)
```

MAD2.m

Program Structure - Repetition

using the break command

```
x = [ 1 1 2 3 5 8 13 21];
for i = 1:8
    if x(i) > 5
        break
    end
end
disp([i x(i)])
```

```
>> ArrayFind
     6     8
```

breakexample.m

using the continue command

```
x = randn(5,1);
n = length(x);
for i = 1:n
    if x(i) > 0; continue; end
    x(i) = 0;
end
disp(x)
```

```
>> zero_out_negatives
    0.3999
         0
         0
         0
    1.1454
```

continueexample.m

Program Structure - Repetition

the for loop structure – nested for loops

```
for i = 1:5
    for j = 1:5
        A(i,j) = 1/(i+j-1);
    end
end
disp(A)
```

nestedloopsexample.m

```
>> HilbertMatrix
    1.0000    0.5000    0.3333    0.2500    0.2000
    0.5000    0.3333    0.2500    0.2000    0.1667
    0.3333    0.2500    0.2000    0.1667    0.1429
    0.2500    0.2000    0.1667    0.1429    0.1250
    0.2000    0.1667    0.1429    0.1250    0.1111
```

Program Structure - Repetition

the while loop structure

```
while condition
   statement(s)
end
```

cycles while condition remains true

break and continue can be used

the general loop structure

```
while 1
   pre-test statement(s)
   if condition ; break ; end
   post-test statement(s)
end
```

1 is equivalent to True, so loop never exits on while statement

this is called a mid-test loop

if no pre-test statements ⇨ called a pre-test loop

if no post-test statements ⇨ called a post-test loop

Program Structure - Repetition

the while loop structure

find a root of

$$f(x) = \frac{1}{(x-q)^2 + 0.01} + \frac{1}{(x-r)^2 + 0.04} - s = 0$$

$$f'(x) = \frac{2(q-x)}{\left((x-q)^2 + 0.01\right)^2} + \frac{2(r-x)}{\left((x-r)^2 + 0.04\right)}$$

```
x = 0.5;
xerr = x;
q = 0.3;
r = 0.9;
s = 12;
while abs(xerr) > 1.e-7
    fx = 1./((x-q)^2+0.01) + 1./((x-r)^2+0.04)-s;
    fpx = 2*(q-x)/((x-q)^2+0.01)^2+2*(r-x)/((x-r)^2+0.04)^2;
    xnew = x - fx/fpx; % Newton-Raphson method
    xerr = xnew-x;
    x = xnew;
end
disp(x)
```

WhileLoop1.m

```
>> WhileLoop1
   1.1335
```

Program Structure - Repetition

the general loop structure

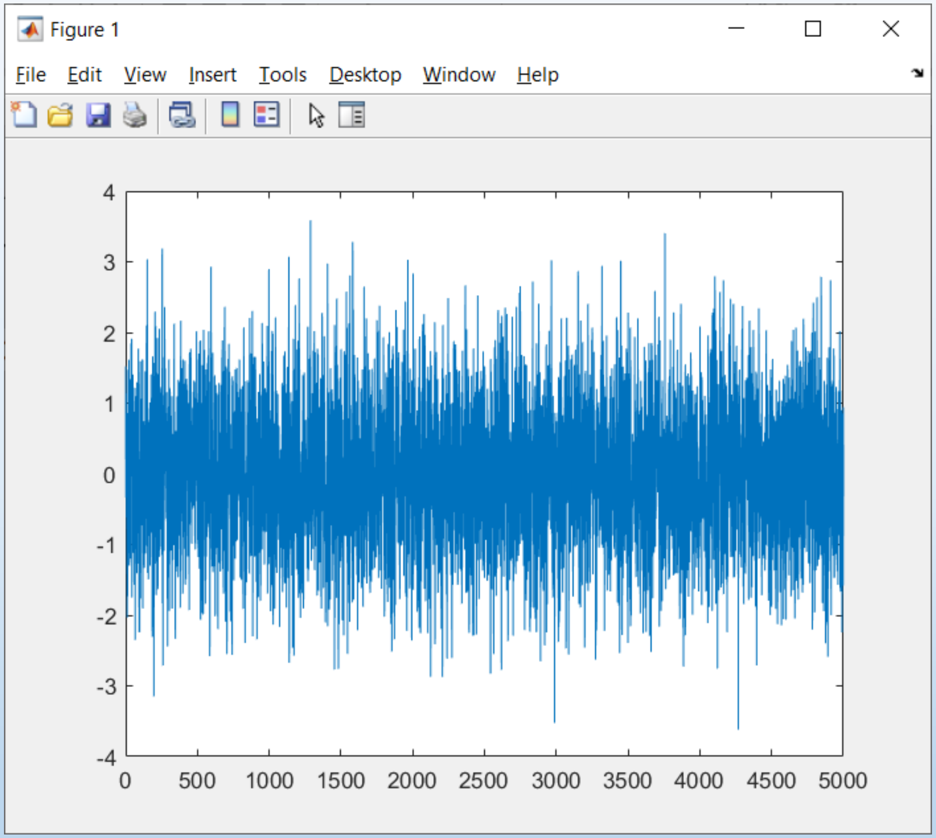
```
x = 0.5;
xerr = x;
q = 0.3;
r = 0.9;
s = 12;
while 1
    fx = 1./((x-q)^2+0.01) + 1./((x-r)^2+0.04)-s;
    fpx = 2*(q-x)/((x-q)^2+0.01)^2+2*(r-x)/((x-r)^2+0.04)^2;
    xnew = x - fx/fpx; % Newton-Raphson method
    xerr = xnew-x;
    if abs(xerr) < 1.e-7 ; break; end
    x = xnew;
end
disp(x)
```

WhileLoop2.m

Creating Plots

Simple 2-D Plots

```
rn = randn(5000,1);
plot(rn);
```



randomnumberplot.m

Creating Plots

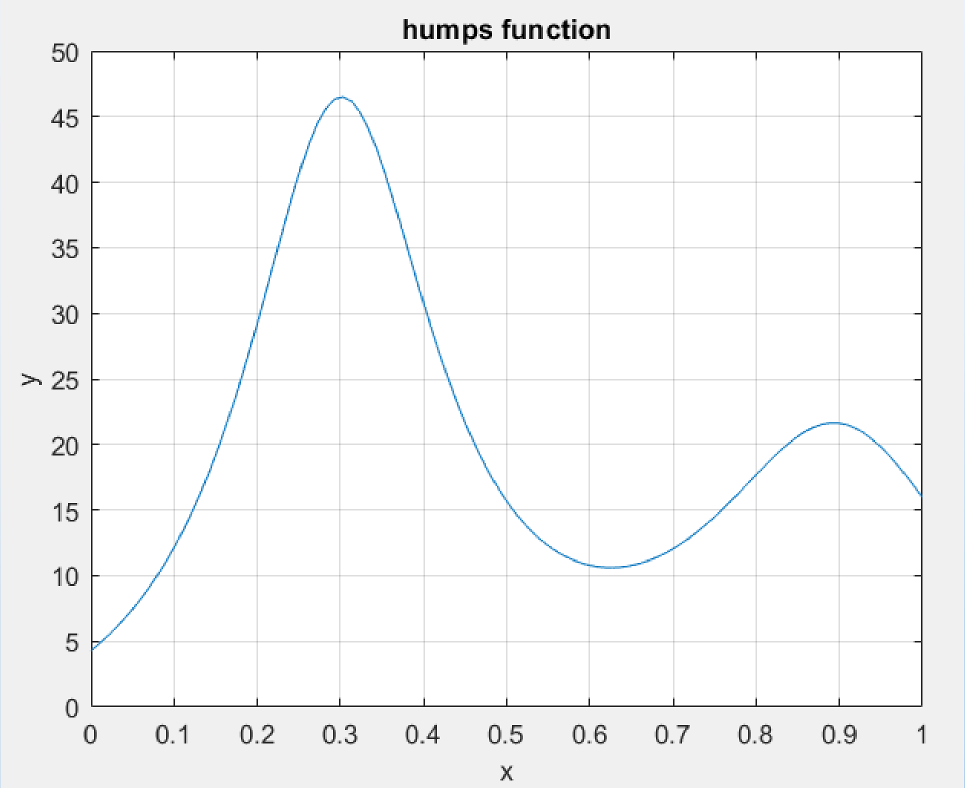
Simple 2-D Plots

Plotting a function using vectorization
Adding a grid, axis labels, and a title

100 evenly spaced points from 0 to 1

```
>> x = linspace(0,1);
>> y = 1./((x-0.3).^2+0.02)+1./((x-0.9).^2+0.04)-6;
>> plot(x,y)
>> grid;
>> xlabel('x')
>> ylabel('y')
>> title('humps function')
```

humpsplot.m



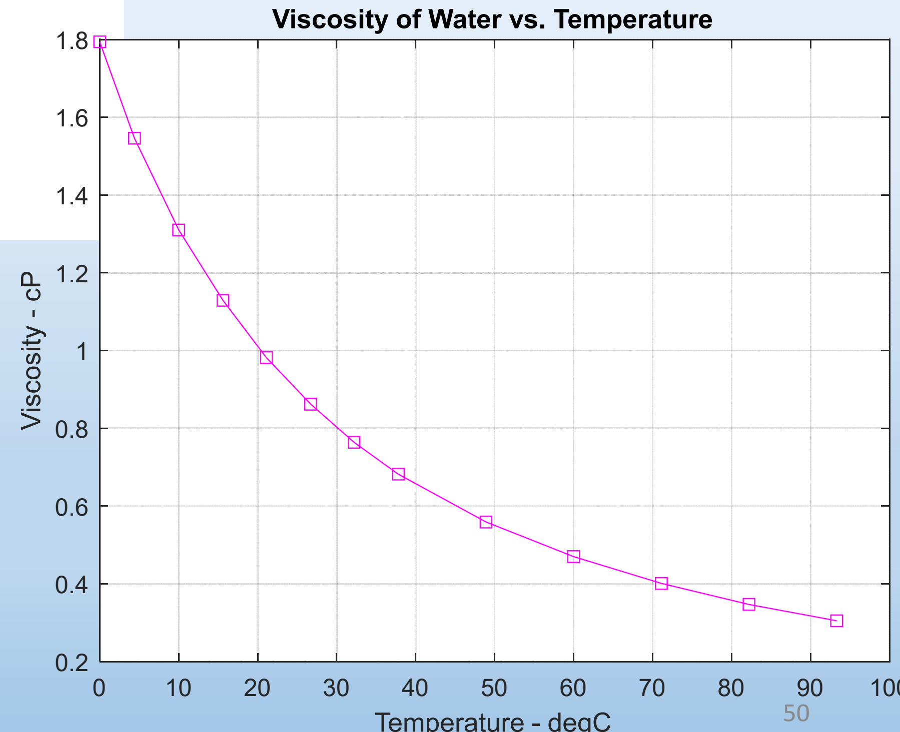
Creating Plots

Simple 2-D Plots

Plotting data with markers and lines
Controlling color and marker shape

```
>> TempC=[0,4.4,10,15.6,21.1,26.7,32.2,37.8,48.9, ...
          60,71.1,82.2,93.3];
>> Visc=[1.794,1.546,1.31,1.129,0.982,0.862,0.764, ...
         0.682,0.559,0.47,0.401,0.347,0.305];
>> plot(TempC,Visc,'sm-')
>> grid;
>> xlabel('Temperature - degC')
>> ylabel('Viscosity - cP')
>> title('Viscosity of Water vs. Temperature')
```

use of an ellipsis, **...** to continue a line



Marker code: s for square
Color code: m for magenta
Linestyle code: - for solid line

See tables of codes in Help plot.

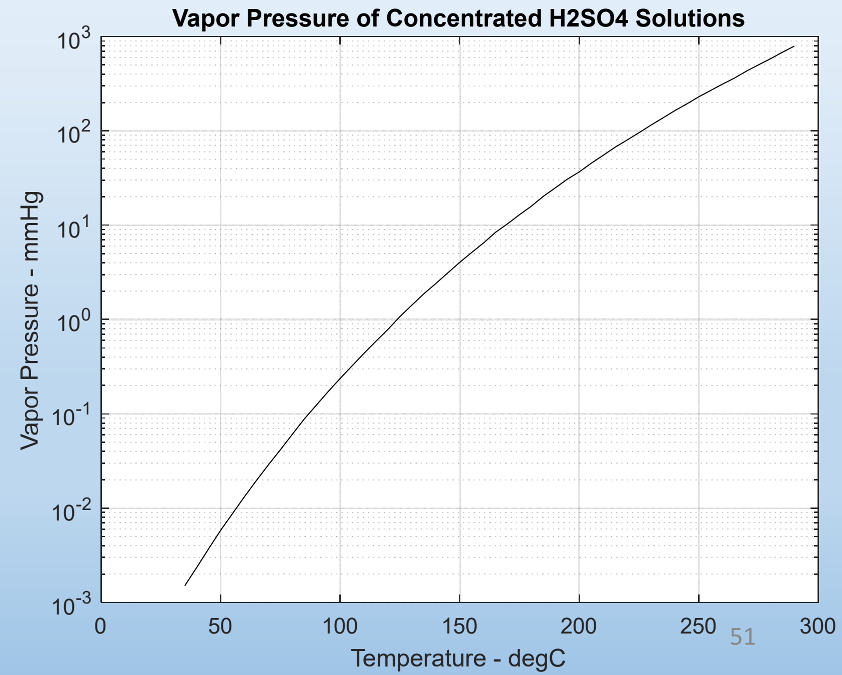
datawithmarkersplot.m

Creating Plots

2-D Plots with logarithmic scale(s)

```
>> sizeVP = [2 Inf];
>> formatSpec = '%f';
>> fileID = fopen('H2SO4VaporPressure.txt','r');
>> TPV = fscanf(fileID,formatSpec,sizeVP);
>> TPV = TPV';
```

```
>> T = TPV(:,1);
>> PV = TPV(:,2);
>> semilogy(T,PV,'k')
>> grid
>> xlabel('Temperature - degC')
>> ylabel('Vapor Pressure - mmHg')
>> title('Vapor Pressure of Concentrated H2SO4 Solutions')
```



Alternate forms:

semilogx	log scale on x axis
loglog	log scale for both axes

logplotexample.m

Creating Plots

SaltSolutionsPlotExample.m

2-D Plots with multiple curves and a legend

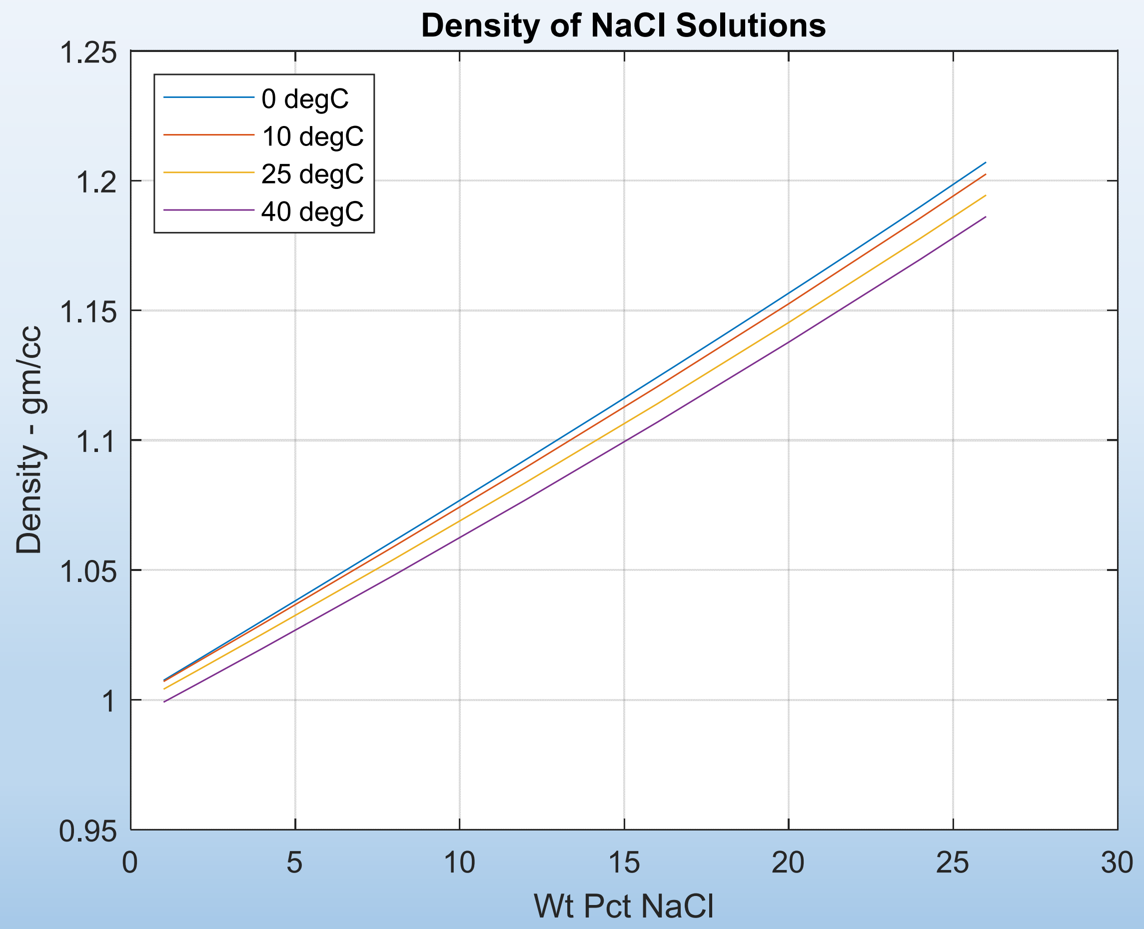
Density of NaCl Aqueous Solutions					
		Temperature			
		0 °C	10 °C	25 °C	40 °C
Wt % NaCl	1	1.00747	1.00707	1.00409	0.99908
	2	1.01509	1.01442	1.01112	1.00593
	4	1.03038	1.02920	1.02530	1.01977
	8	1.06121	1.05907	1.05412	1.04798
	12	1.09244	1.08946	1.08365	1.07699
	16	1.12419	1.12056	1.11401	1.10688
	20	1.15663	1.15254	1.14533	1.13774
	24	1.18999	1.18557	1.17776	1.16971
	26	1.20709	1.20254	1.19443	1.18614

```
>> Dens = [1.00747,1.00707,1.00409,0.99908; ...
           1.01509,1.01442,1.01112,1.00593; ...
           1.03038,1.02920,1.02530,1.01977; ...
           1.06121,1.05907,1.05412,1.04798; ...
           1.09244,1.08946,1.08365,1.07699; ...
           1.12419,1.12056,1.11401,1.10688; ...
           1.15663,1.15254,1.14533,1.13774; ...
           1.18999,1.18557,1.17776,1.16971; ...
           1.20709,1.20254,1.19443,1.18614];
```

```
>> WtPct = [1,2,4,8,12,16,20,24,26]';
>> plot(WtPct,Dens)
>> grid;
>> xlabel('Wt Pct NaCl')
>> ylabel('Density - gm/cc')
>> title('Density of NaCl Solutions')
>> legend('0 degC','10 degC','25 degC','40 degC', ...
          'location','northwest')
```

Creating Plots

2-D Plots with multiple curves and a legend

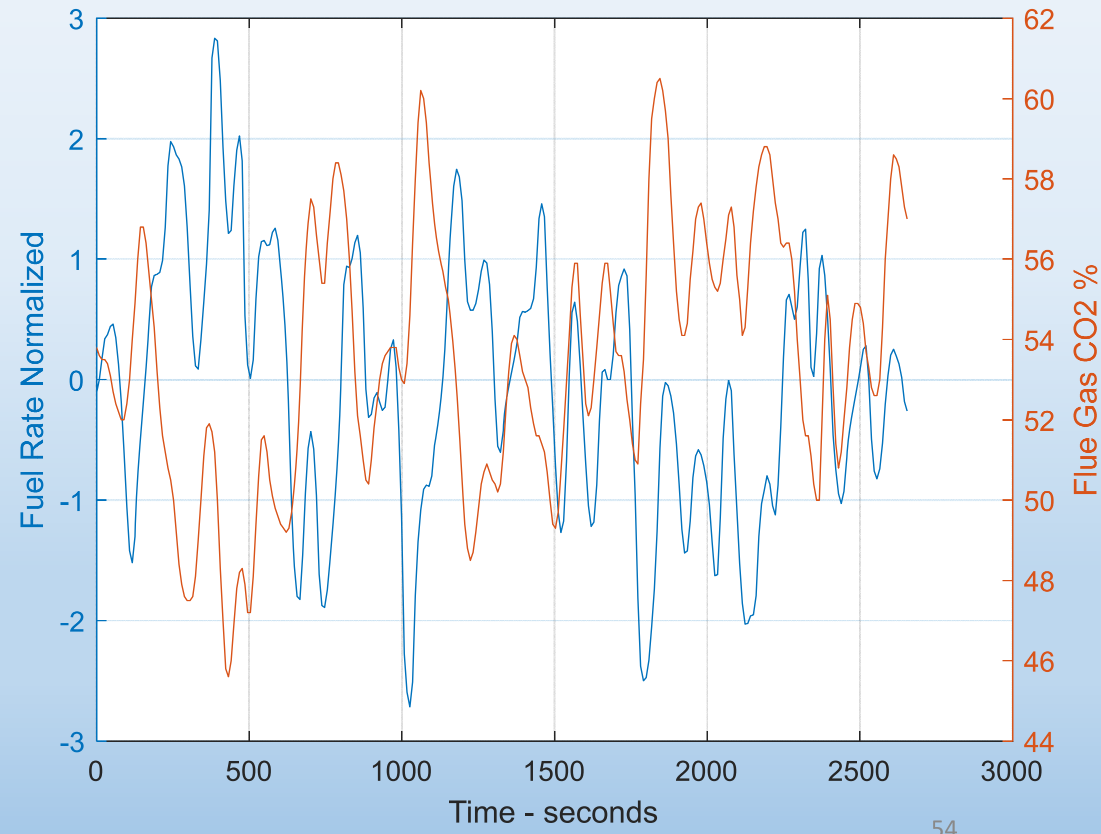


Creating Plots

2-D Plots with both left and right vertical axes

```
>> sizeFurnace = [3 Inf];
>> formatSpec = '%f';
>> fileID = fopen('FurnaceData.txt','r');
>> Fdata = fscanf(fileID,formatSpec,sizeFurnace);
>> Fdata = Fdata';
>> ts = Fdata(:,1);
>> fuel = Fdata(:,2);
>> CO2 = Fdata(:,3);
>> yyaxis left
>> plot(ts,fuel)
>> yyaxis right
>> plot(ts,CO2)
>> grid
>> xlabel('Time - seconds')
>> ylabel('Flue Gas CO2 %')
>> yyaxis left
>> ylabel('Fuel Rate Normalized')
```

yyaxis command



twinaxesplotexample.m

Creating Plots

3-D Contour and Surface Plots

Creating a meshgrid

```
x =

   -2.0000
   -1.5000
   -1.0000
   -0.5000
         0
    0.5000
    1.0000
    1.5000
    2.0000

>> y = x
```

```
>> [X,Y] = meshgrid(x,y);
```

```
X =

  Columns 1 through 5

   -2.0000   -1.5000   -1.0000   -0.5000         0
   -2.0000   -1.5000   -1.0000   -0.5000         0
   -2.0000   -1.5000   -1.0000   -0.5000         0
   -2.0000   -1.5000   -1.0000   -0.5000         0
   -2.0000   -1.5000   -1.0000   -0.5000         0
   -2.0000   -1.5000   -1.0000   -0.5000         0
   -2.0000   -1.5000   -1.0000   -0.5000         0
   -2.0000   -1.5000   -1.0000   -0.5000         0
   -2.0000   -1.5000   -1.0000   -0.5000         0

  Columns 6 through 9

    0.5000    1.0000    1.5000    2.0000
    0.5000    1.0000    1.5000    2.0000
    0.5000    1.0000    1.5000    2.0000
    0.5000    1.0000    1.5000    2.0000
    0.5000    1.0000    1.5000    2.0000
    0.5000    1.0000    1.5000    2.0000
    0.5000    1.0000    1.5000    2.0000
    0.5000    1.0000    1.5000    2.0000
    0.5000    1.0000    1.5000    2.0000
```

```
Y =

  Columns 1 through 5

   -2.0000   -2.0000   -2.0000   -2.0000   -2.0000
   -1.5000   -1.5000   -1.5000   -1.5000   -1.5000
   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000
   -0.5000   -0.5000   -0.5000   -0.5000   -0.5000
         0         0         0         0         0
    0.5000    0.5000    0.5000    0.5000    0.5000
    1.0000    1.0000    1.0000    1.0000    1.0000
    1.5000    1.5000    1.5000    1.5000    1.5000
    2.0000    2.0000    2.0000    2.0000    2.0000

  Columns 6 through 9

   -2.0000   -2.0000   -2.0000   -2.0000
   -1.5000   -1.5000   -1.5000   -1.5000
   -1.0000   -1.0000   -1.0000   -1.0000
   -0.5000   -0.5000   -0.5000   -0.5000
         0         0         0         0
    0.5000    0.5000    0.5000    0.5000
    1.0000    1.0000    1.0000    1.0000
    1.5000    1.5000    1.5000    1.5000
    2.0000    2.0000    2.0000    2.0000
```

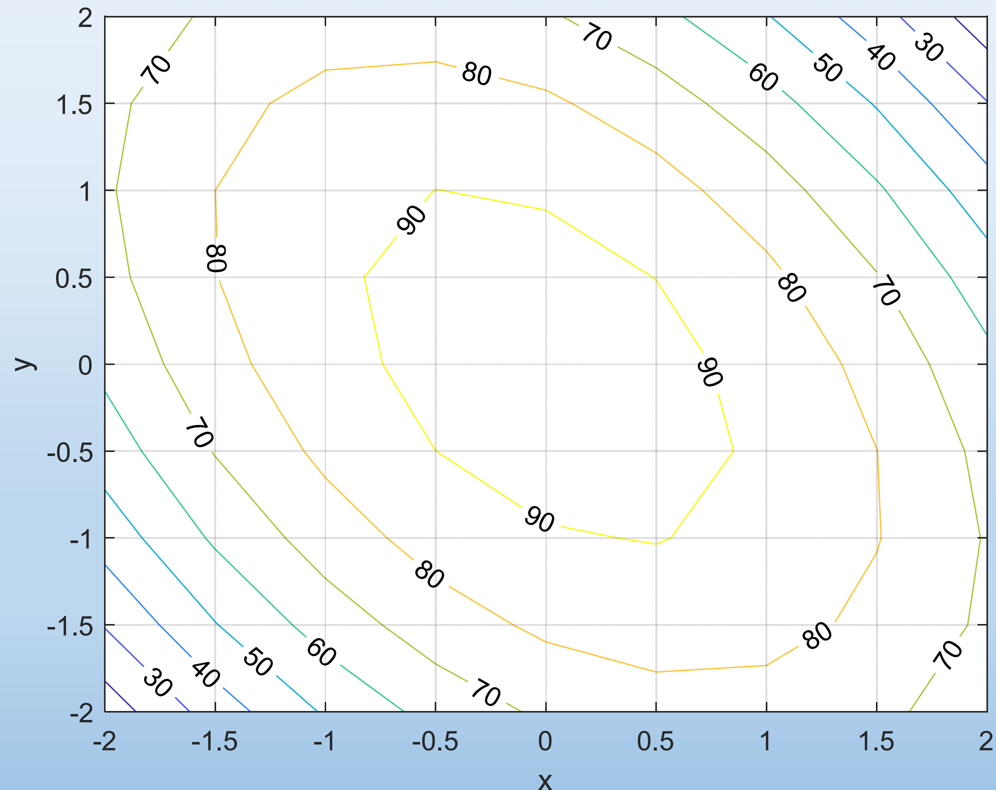
surfaceplotexample.m

Creating Plots

3-D Contour and Surface Plots

```
>> Z = 95.+0.05*X-0.145*Y-8.13*X.^2-5.87*Y.^2 ...
      -6.25*X.*Y;
>> contour(X,Y,Z,'showtext','on')
>> xlabel('x')
>> ylabel('y')
>> zlabel('z')
>> grid
```

contourplotexample1.m

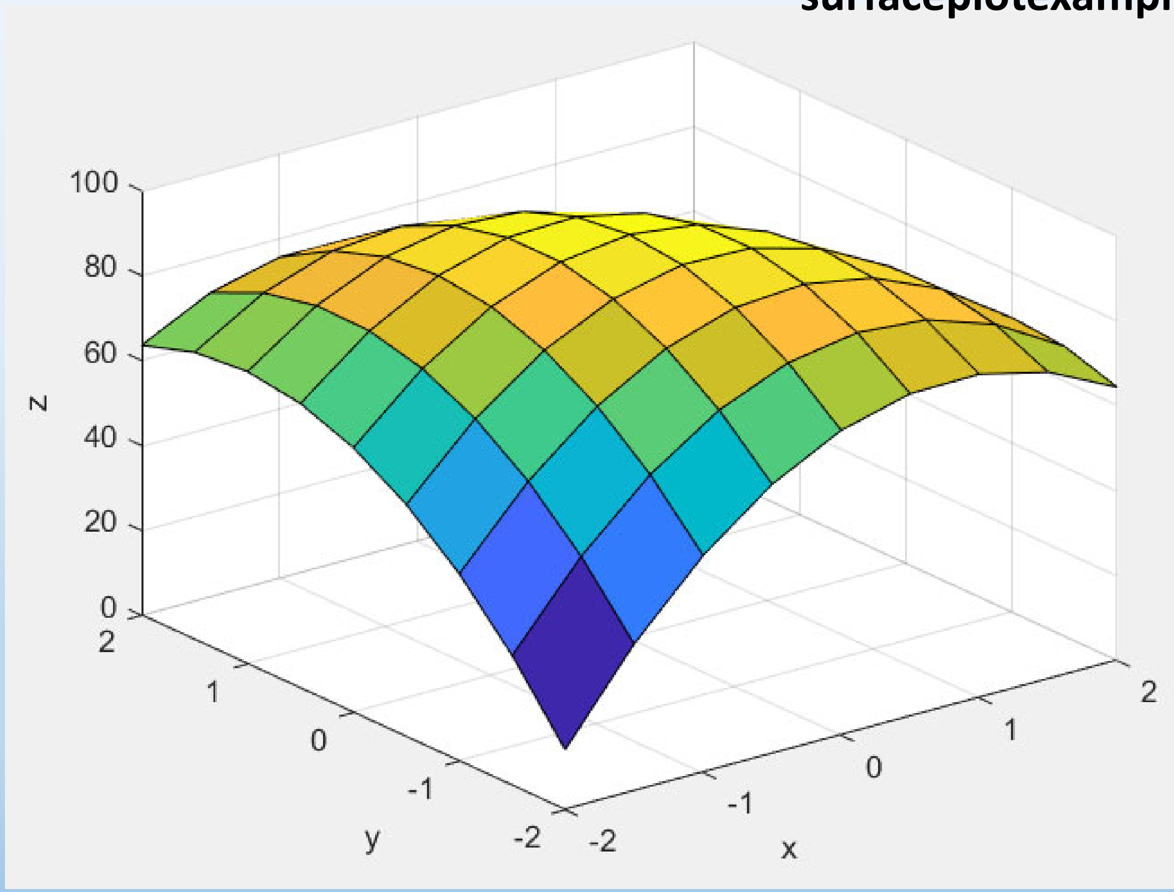


Creating Plots

3-D Contour and Surface Plots

```
>> surf(X,Y,Z)
>> grid('on')
>> xlabel('x')
>> ylabel('y')
>> zlabel('z')
```

surfaceplotexample.m

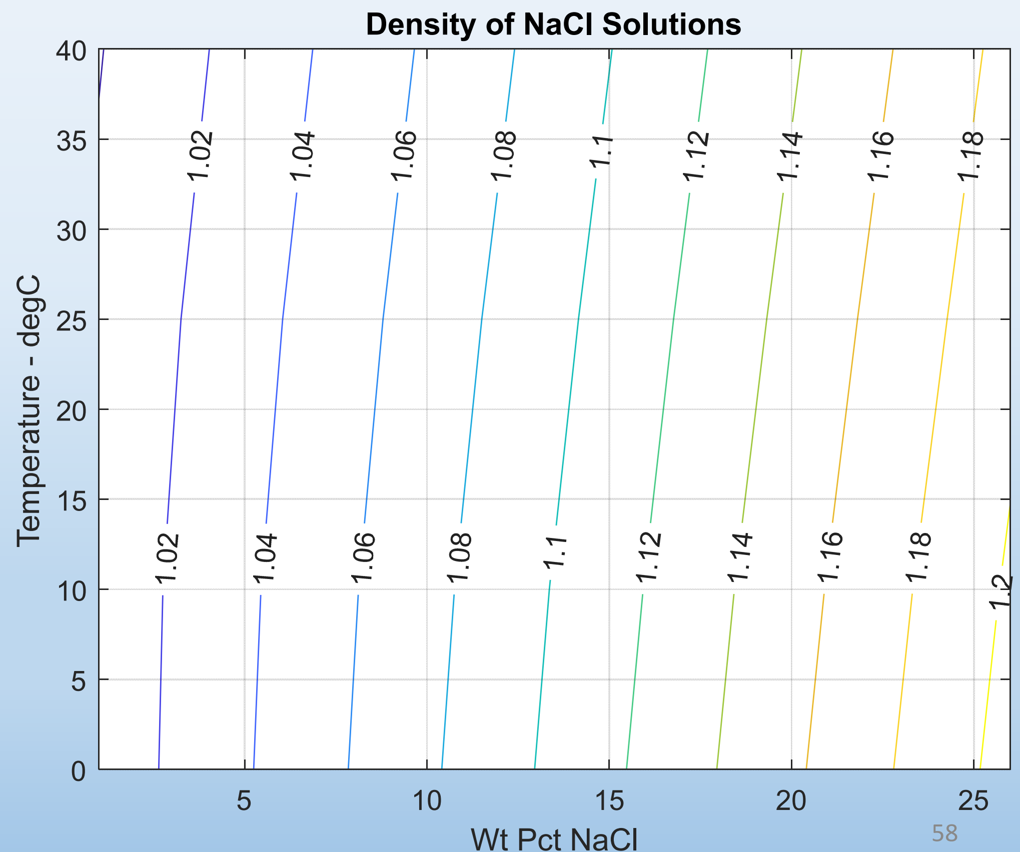


Creating Plots

3-D Contour and Surface Plots Based on Data

```
Temp = [0,10,25,40];
[WP,TP] = meshgrid(WtPct,Temp);
WP = WP' ; TP = TP';
contour(WP,TP,Dens,'ShowText','on')
grid
xlabel('Wt Pct NaCl')
ylabel('Temperature - degC')
title('Density of NaCl Solutions')
```

SaltSolutionsContourPlot.m

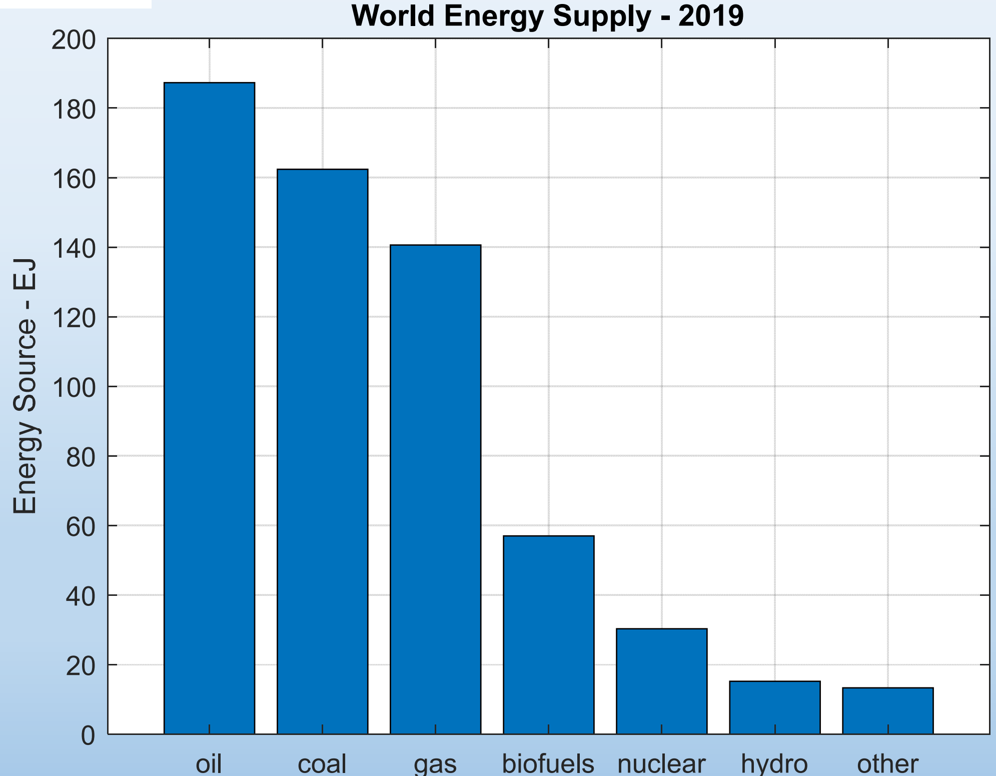


Creating Plots

Bar charts

Note use of braces, { }, for array of strings, a *cell array*.

```
>> Source = {'oil','coal','gas','biofuels', ...
'nuclear','hydro','other'};
>> EJ = [187.3,162.4,140.6,57,30.3,15.2,13.3];
>> X = categorical(Source);
>> X = reordercats(X,Source);
>> bar(X,EJ)
>> grid
>> ylabel('Energy Source - EJ')
>> title('World Energy Supply - 2019')
```



barchartexample.m

Creating Plots

Pie charts

```
>> Country = {'China','US','Germany','India', ...
'Spain','UK'};
>> WindGW = [221,96.4,59.3,35,23,21.7];
>> pie(WindGW,Country)
>> title('Wind Power Capacity - 2021')
```

piechartexample.m

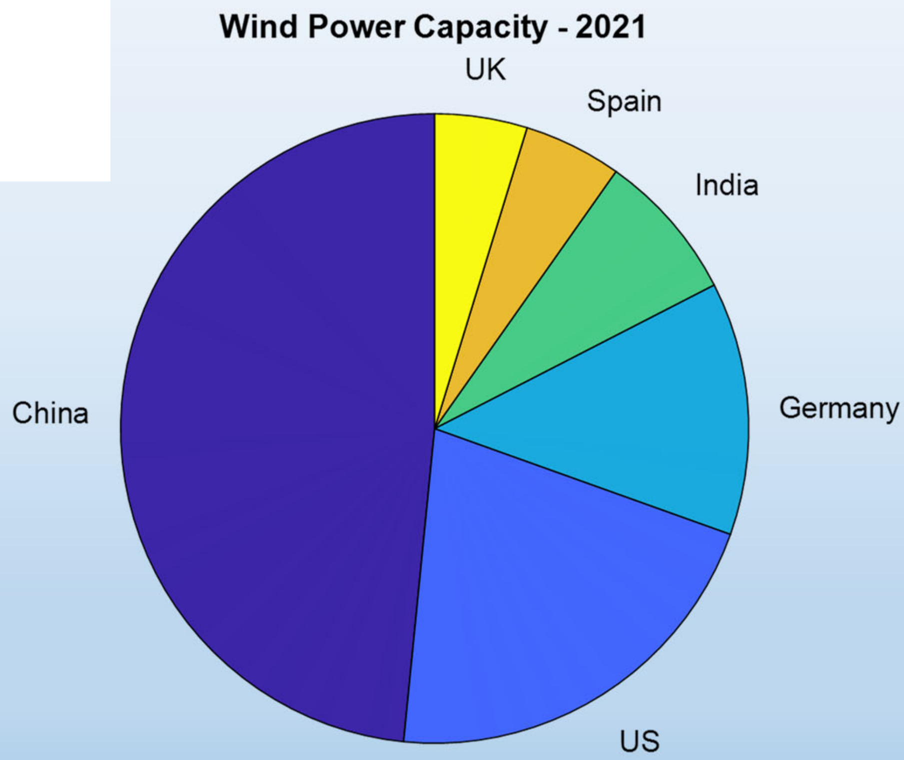


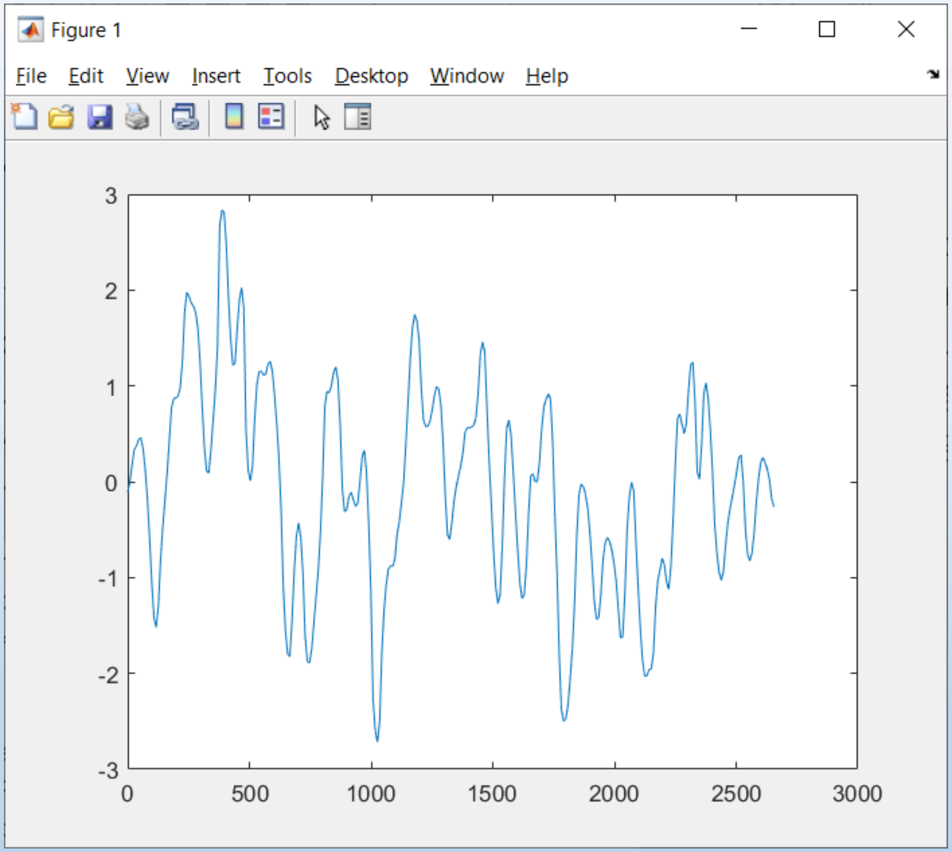
Figure Windows

A plot command creates a figure window where the plot is displayed. If another plot command is issued, that plot replaces the former plot in that figure window. To créate a new figure window for the next plot, use the figure command. Each figure window is numbered and can be made the current figure window with the *figure*(n) command where n is the window number. A figure window can be removed with the *clf*(n) command. All figures can be cleared with the *cla* command.

```
sizeFurnace = [3 Inf];
formatSpec = '%f';
fileID = fopen('FurnaceData.txt','r');
Fdata = fscanf(fileID,formatSpec,sizeFurnace);
Fdata = Fdata';
ts = Fdata(:,1);
fuel = Fdata(:,2);
CO2 = Fdata(:,3);
plot(ts,fuel)
figure
plot(ts,CO2)
```

figurewindowexample1.m

Figure Windows



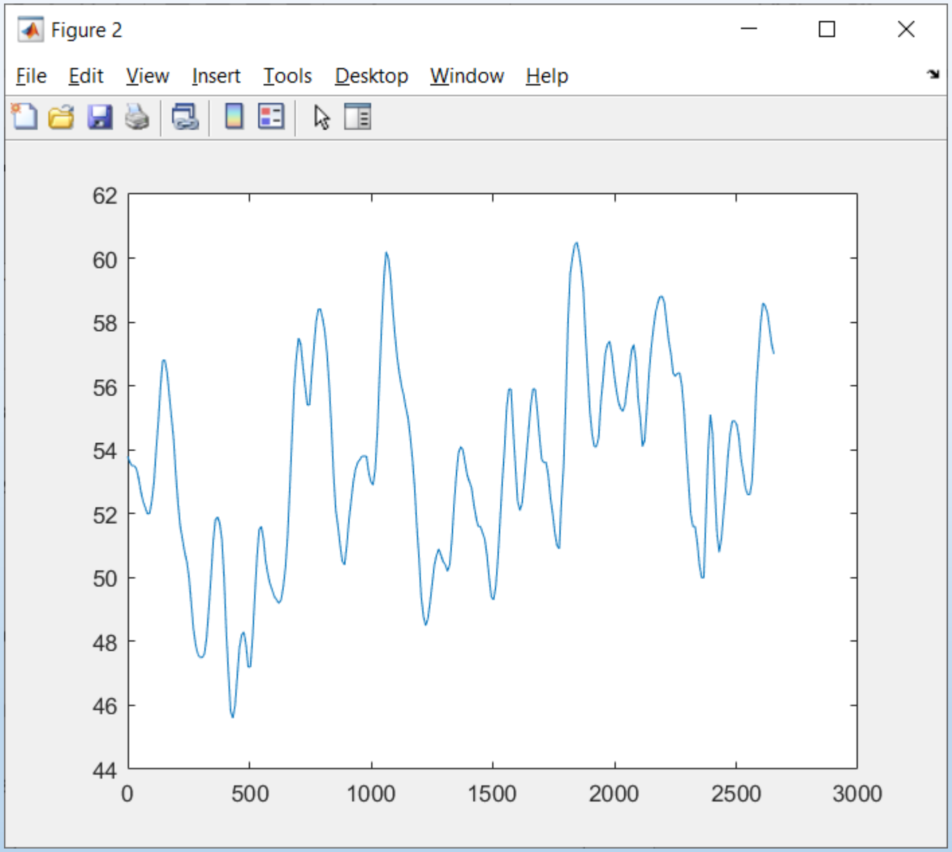
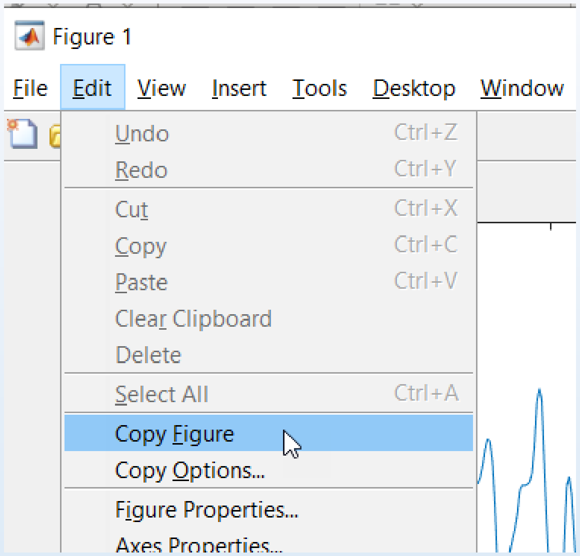
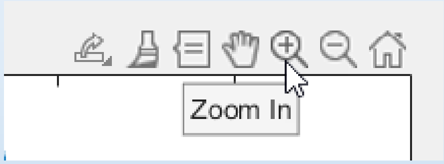


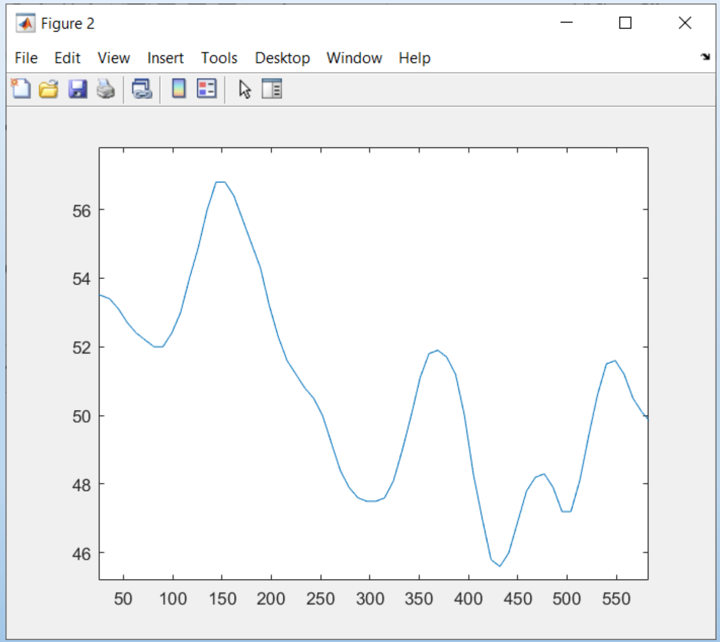
Figure Windows

Copy a plot to a Word document or PowerPoint slide or . . .



Zoom in on a portion of the plot





Draw box of zoom area

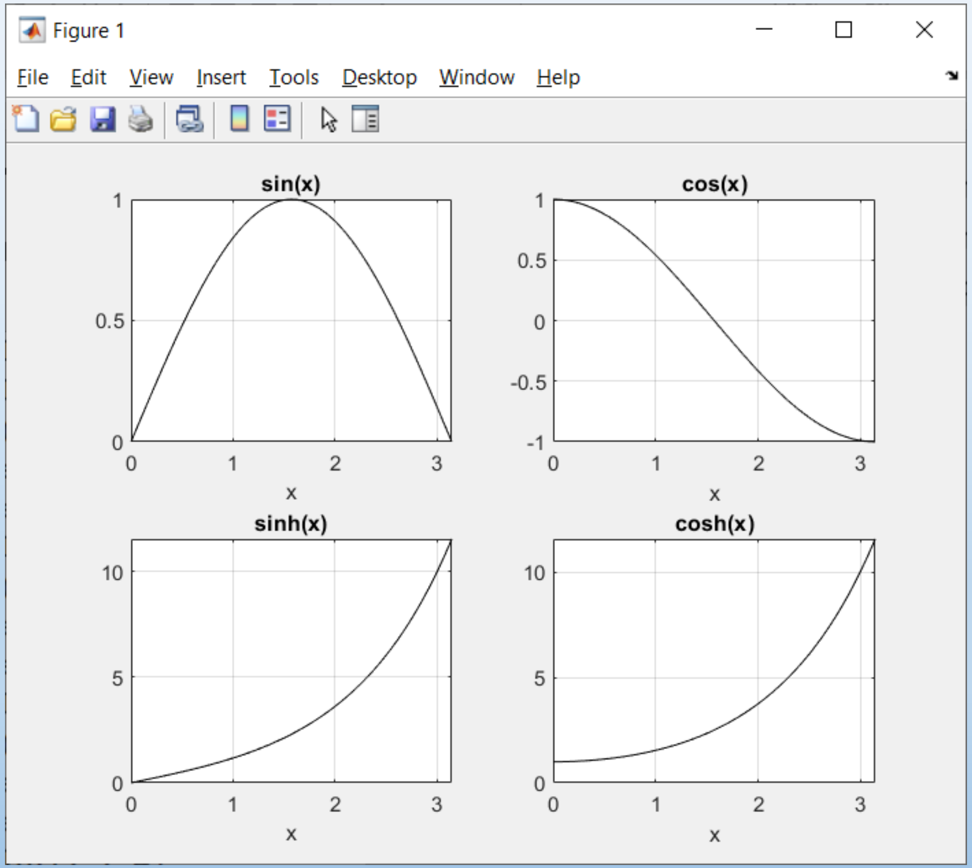
Figure Windows

Multiple subplots in a figure window

subplot(m,n,p)

m: no. of rows
n: no. of columns
p: subplot selected
(counting across then down)

```
x = linspace(0,pi);
subplot(2,2,1)
plot(x,sin(x),'k');grid;
xlabel('x')
title('sin(x)')
subplot(2,2,2)
plot(x,cos(x),'k');grid;
xlabel('x')
title('cos(x)')
subplot(2,2,3)
plot(x,sinh(x),'k');grid;
xlabel('x')
title('sinh(x)')
subplot(2,2,4)
plot(x,cosh(x),'k');grid;
xlabel('x')
title('cosh(x)')
```



figurewindowexample2.m

MATLAB Function m-Files

cart_to_polar.m

General syntax

function [outarg1, outarg2, ...] = function_name(inarg1, inarg2, ...)

-
-
-

outarg1 = ...;

outarg2 = ...;

create with the Editor in a script file saved as function_name.m

Simple example

untitled *

```
function [r,theta]=cart_to_polar(x,y)
% convert Cartesian coordinates to polar
r = sqrt(x^2+y^2);
theta = atan2(x,y);
```

Save

File name: cart_to_polar.m

Save as type: MATLAB Code files (UTF-8) (*.m)

name of function is same as name of file – suggested automatically

MATLAB Function m-Files

Example

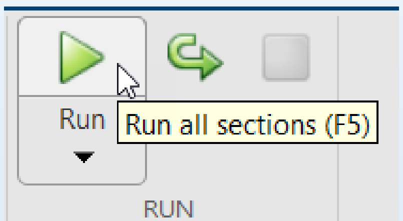
```
>> [rad th] = cart_to_polar(-3,-4)

rad =

     5


th =

   -2.4981

>> thdeg = rad2deg(th)

thdeg =

 -143.1301
```

Note: MATLAB has a built-in function *cart2pol*.

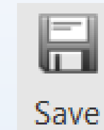
Always check via Help to see whether there is already a function available before creating one!

MATLAB m-Scripts in the Editor

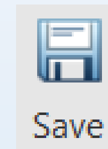
Enter code in Editor window
Save file
Run the script, button or F5



When script is saved, Save icon grays out.

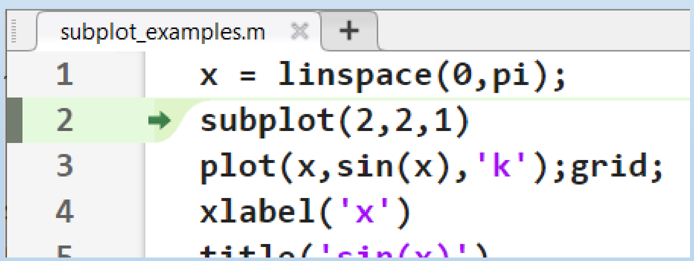


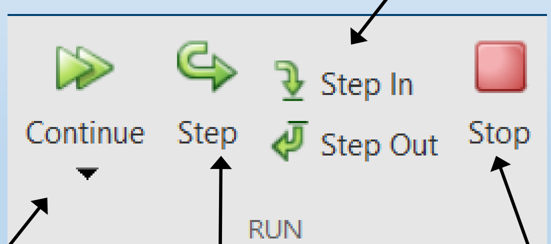
For any changes in the script, Save icon re-colors alerting the need to Save.

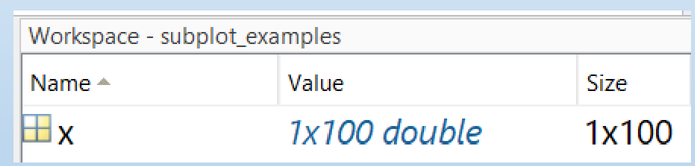


If errors, single-step the code (F10), observing variable values in the Workspace.

Whether to single-step into a function







Continuous run

Single-step

Stop execution

MATLAB m-Scripts in the Editor

PengRobinsonMethaneExample.m

Adding comments to scripts

- descriptive comments on their own line
- comments appended to code lines
- clarifying units

PengRobinsonCH4.m

```
% compute gas pressure using
% the Peng-Robinson equation
% of state for methane
clear  % clear Workspace
clc  % clear Command window
global R  % make R availble to functions
R = 8.314;  % kJ/kmol/K
% methane molecular weight
MW = 16.04; % kg/kmol
% critical properties for methane
Tc = 191.15;  % K
Pc = 4.641e6;  % Pa
% acentric factor for methane
w = 0.0115;
% conditions
T = 700;  % K
Vm = 0.1;  % m3/kmol
Tr = T/Tc;
% Peng-Robinson parameters
a = 0.45724*R^2*Tc^2/Pc;
b = 0.07780*R*Tc/Pc;
alpha = (1+(0.37464+1.54226*w-0.26992*w^2)*(1-sqrt(Tr)))^2;
% compute pressure in Pa
P = R*T/(Vm-b)-a*alpha/(Vm^2+2*b*Vm-b^2);
% convert pressure to kPa
PM = P/1000;  % kPa
% display result
fprintf('Pressure = %4.1f kPa\n',PM)
```

```
Pressure = 58.2 kPa
```

Reference

Applied Numerical Methods
with MATLAB for Engineers and Scientists
Steven C. Chapra
5th Edition, McGraw-Hill, 2022.

"Prof. Clough, may I be excused? My brain is full."

MATLAB – What's Next?

Bootcamp 2

- ✓ 1: Getting up to speed (or back up to speed) with MATLAB
- 2: Learning to use MATLAB to solve typical problem scenarios
- 3: Detailed modeling of packed-bed and plug-flow reactors