

Introduction to MATLAB

Mughees Asif

MSc. Artificial Intelligence

MathWorks Campus Ambassador

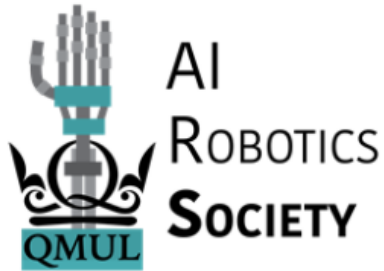


Overview

- **Introduction** 🖐️:
 - Interface
 - Documentation
 - Getting help
 - Common commands
 - MATLAB symbols
- **Syntax** ✂️:
 - Matrix manipulation
 - ``:`` operator
 - *for* loop
 - *while* loop
 - *if* statements
 - *switch* statements
 - Vectorization
- **Graphics** 📊:
 - Common commands
 - Examples - 2D plot
 - Examples - 3D plot
- **Scripts & Functions** ⚡
- **Demonstration** 🖥️
- **Final word** 🔊
- **Quiz** 📋
- **Pizza** 🍕

Artificial Intelligence and Robotics Society

Educational, Employability & Enterprising



About

Artificial Intelligence & Robotics Society is for people of all skill sets and backgrounds who want to learn or practice robotics skills & knowledge.




Please visit our website for more information, photos & videos

qmrs.co.uk

Contact Us

Artificial Intelligence and Robotics Society

Contact us with any questions you may have:

-  robosociety@qmsu.org
-  www.facebook.com/QueenMaryAIR
-  [qmairs](https://www.instagram.com/qmairs)

Join

Artificial Intelligence and Robotics Society standard Membership 2021/22

£6.99

Add to Basket

Artificial Intelligence and Robotics Society Associate Membership 2021/22

£12.00

Add to Basket

Products/ Tickets

Ai and Robotics Society T-shirt

£15.00

Overview

- **Introduction** 🖐️:

- Interface
- Documentation
- Getting help
- Common commands
- MATLAB symbols

- **Syntax** ✂️:

- Matrix manipulation
- ``:`` operator
- *for* loop
- *while* loop
- *if* statements
- *switch* statements
- Vectorization

- **Graphics** 📊:

- Common commands
- Examples - 2D plot
- Examples - 3D plot

- **Scripts & Functions** ⚡

- **Demonstration** 🖥️

- **Final word** 🔊

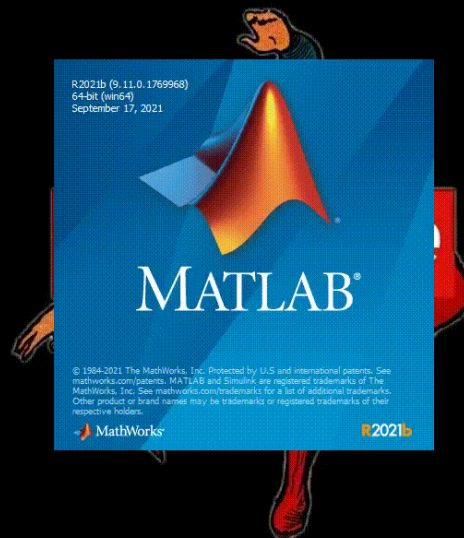
- **Quiz** 📋

- **Pizza** 🍕

Introduction

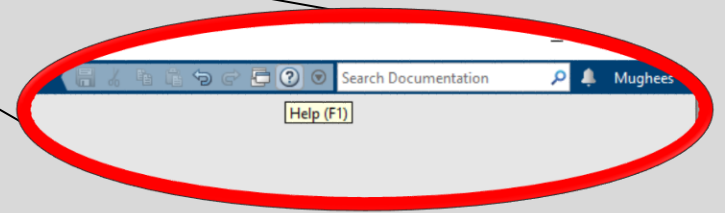
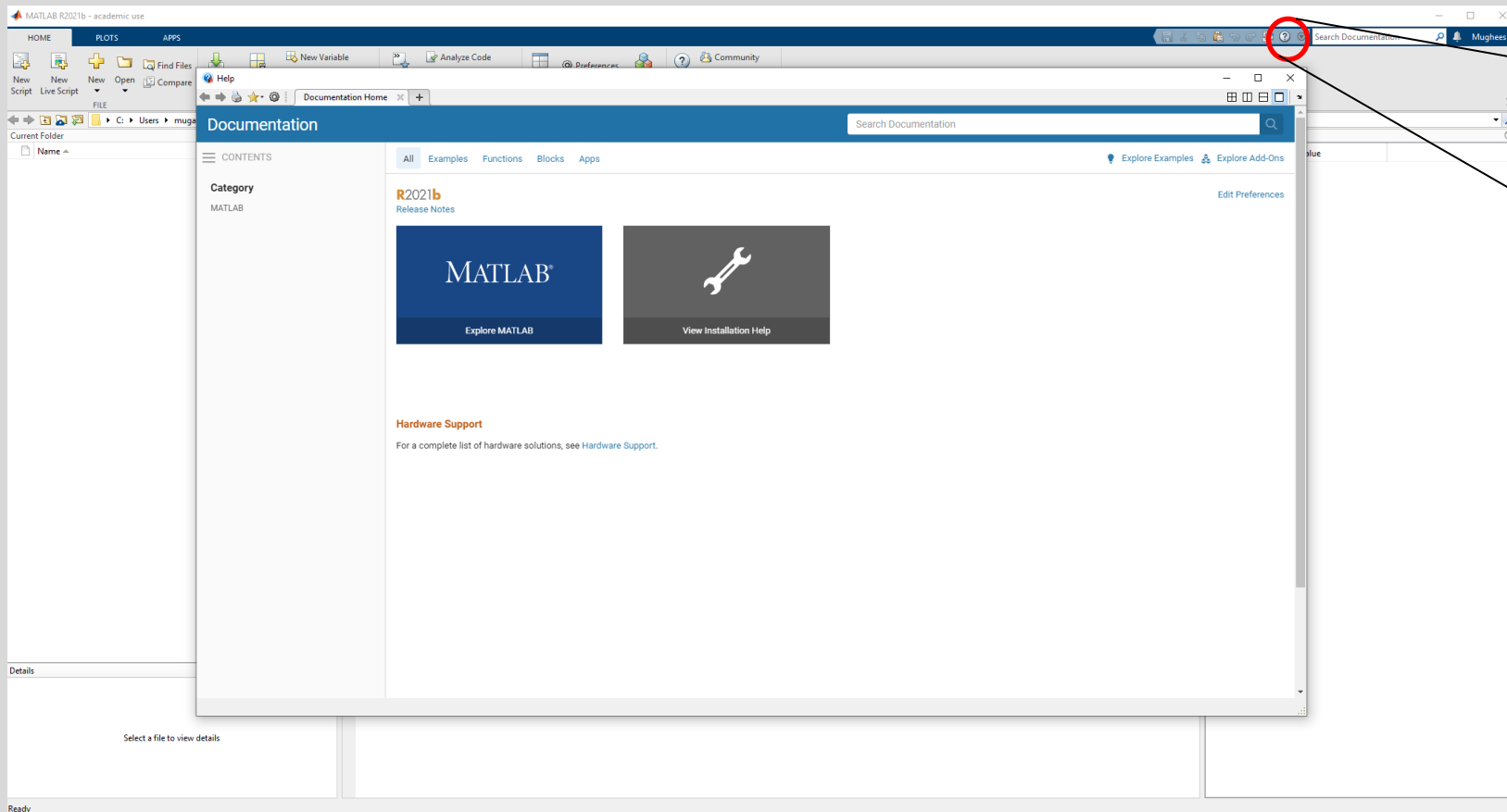
- Stands for **MAT**rix **LAB**oratory.
- Designed for the manipulation of matrices:
 - **Engineering / Physics**: Model physical systems and perform precise calculations.
 - **Robotics / Kinematics**: Rotation matrices; translations through planes to be easily calculated.
 - **Betting**: Complex betting combinations without separate formulae such as multiple complex simultaneous equations.
 - **Data Mining**: Fundamental to the handling of data.
 - **Graphics/Gaming**: From particle collision to ray tracing.
- Great visualisation capabilities.
- Multiple libraries of built-in functions.

Interface



Documentation

- Most useful.
- Most unused.



https://uk.mathworks.com/help/

Applications

[expand all](#)

▼ Math, Statistics, and Optimization

[Curve Fitting Toolbox](#)
[Deep Learning HDL Toolbox](#)
[Deep Learning Toolbox](#)
[Global Optimization Toolbox](#)
[Optimization Toolbox](#)
[Partial Differential Equation Toolbox](#)
[Statistics and Machine Learning Toolbox](#)
[Symbolic Math Toolbox](#)
[Text Analytics Toolbox](#)

> Data Science and Deep Learning

> Signal Processing and Wireless Communications

> Control Systems

> Image Processing and Computer Vision

> Parallel Computing

> Test and Measurement

> Computational Finance

> Computational Biology

> Application Deployment

> Event-Based Modeling

> Physical Modeling

▼ Robotics and Autonomous Systems

[Automated Driving Toolbox](#)
[Lidar Toolbox](#)
[Navigation Toolbox](#)
[RoadRunner](#)
[Robotics System Toolbox](#)
[ROS Toolbox](#)
[Sensor Fusion and Tracking Toolbox](#)
[UAV Toolbox](#)

> FPGA, ASIC, and SoC Development

> Real-Time Simulation and Testing

> Code Generation

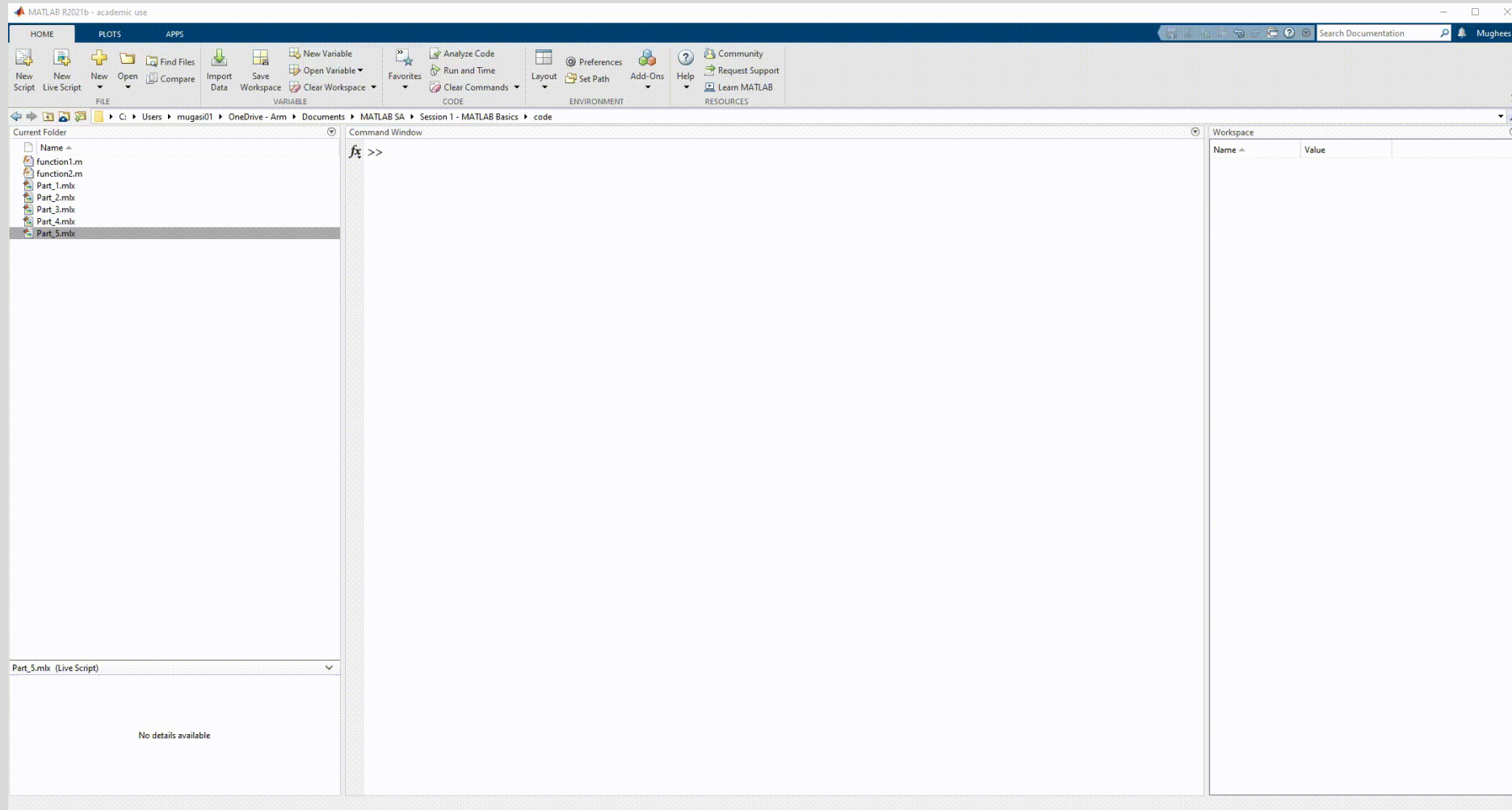
> Verification, Validation, and Test

> Database Access and Reporting

> Simulation Graphics and Reporting

> Systems Engineering

Getting help on specific functions








Common commands

- **who, whos** – current workspace vars
- **save** – save workspace vars to *.mat file
- **load** – load vars from *.mat file
- **clear all** – clear workspace vars
- **close all** – close all figures
- **clc** – clear screen
- **clf** – clear figure

MATLAB symbols

- >> prompt
- . . . continue statement on next line
- , separate statements and data
- % start comment which ends at end of line
- ;
 - (1) suppress output
 - (2) used as a row separator in a matrix

Overview

- **Introduction** :
 - Interface
 - Documentation
 - Getting help
 - Common commands
 - MATLAB symbols
- **Syntax** :
 - Matrix manipulation
 - `:` operator
 - *for* loop
 - *while* loop
 - *if* statements
 - *switch* statements
 - Vectorization
- **Graphics** :
 - Common commands
 - Examples - 2D plot
 - Examples - 3D plot
- **Scripts & Functions** 
- **Demonstration** 
- **Final word** 
- **Quiz** 
- **Pizza** 

Matrix manipulation

- Do not need to initialise type, or dimensions.

```
>> A = [3 2 1; 5 1 0; 2 1 7]
```

```
A =
```

```
    3    2    1
    5    1    0
    2    1    7
```


square brackets to define matrices

; specifies the next row in the matrix

Matrix manipulation

- Access elements of a matrix
- Syntax: *matrixname(row, column)*

indices of matrix
element(s)



```
>> A = [3 2 1; 5 1 0; 2 1 7]
```

```
A =
```

3	2	1
5	1	0
2	1	7

```
>> A(1, 2)
```

```
ans =
```

```
2
```

```
>> A(3, 3)
```

```
ans =
```

```
7
```

Matrix manipulation

```
>> A.'           % transpose
>> B * A         % matrix multiplication
>> B.* A         % element by element
                  % multiplication
>> B / A         % matrix division
>> B./ A         % element by element
                  % division
>> [B A]         % join matrices (horizontally)
>> [B; A]        % join matrices (vertically)
```

```
>> A

A =

     3     2     1
     5     1     0
     2     1     7

>> B

B =

     1     3     1
     4     9     5
     2     7     2

>> C = B * A

C =

    20     6     8
    67    22    39
    45    13    16

>> D = B.* A

D =

     3     6     1
    20     9     0
     4     7    14
```

`:` operator

- Vector creation, array subscripting, and *for* loop iteration.

```
>> a = 5;  
b = 15;  
c = a:b  
  
c =  
  
5      6      7      8      9      10      11      12      13      14      15
```

```
>> 1:10  
ans =  
  
1      2      3      4      5      6      7      8      9      10  
  
>> 1:2:10  
ans =  
  
1      3      5      7      9
```

```
>> A  
  
A =  
  
3      2      1  
5      1      0  
2      1      7  
  
>> % n-th column of matrix A  
>> A(:,2)  
  
ans =  
  
2  
1  
1  
  
>> % m-th row of matrix A  
>> A(2,:)   
  
ans =  
  
5      1      0
```


for loops

Syntax

```
for index = values  
    statements  
end
```

```
>> for i = 1:10      % Start at 1, finish  
    disp(i)         % at 10.  
end  
1  
  
2  
  
3  
  
4  
  
5  
  
6  
  
7  
  
8  
  
9  
  
10
```

for loops

```
>> for v = 1:-0.2:0      % Start at 1, decrement by -0.2,  
    disp(v)             % and stop when 0  
end  
    1  
  
    0.8000  
  
    0.6000  
  
    0.4000  
  
    0.2000  
  
    0
```

for loops

```
fx >> x = 0;  
      for i = 1:2:15  
          x = x + i;  
          sprintf('i: %d, x: %d', i, x)  
      end
```

?

for loops

```
>> x = 0;  
    for i = 1:2:15           % start at 1, increment i by 2,  
        x = x + i;           % keep adding x & i, stop when i = 15.  
        sprintf('i: %d, x: %d\n', i, x)  
    end
```

'i: 1, x: 1'

'i: 3, x: 4'

'i: 5, x: 9'

'i: 7, x: 16'

'i: 9, x: 25'

'i: 11, x: 36'

'i: 13, x: 49'

'i: 15, x: 64'

while loops

Syntax

```
while expression
    statements
end
```

```
>> k = 1;
    while k <= 10    % Keep looping until k is less than or
        disp(k)      % equal to 10.
        k = k + 1;
    end
    1
    2
    3
    4
    5
    6
    7
    8
    9
    10
```

while loops

```
>> n = 10;
    while n > 1      % Keep looping until n is more
        disp(n)      % than 1.
        n = n-1;
    end
    10

    9

    8

    7

    6

    5

    4

    3

    2
```

if statements

Syntax

```
if expression
    statements
elseif expression
    statements
else
    statements
end
```

```
>> x = 10;
    minVal = 2;
    maxVal = 6;

    if (x >= minVal) && (x <= maxVal)
        disp('Value within specified range.')
    elseif (x > maxVal)
        disp('Value exceeds maximum value.')
    else
        disp('Value is below minimum value.')
    end
Value exceeds maximum value.
```

if statements

```
>> x = 10;  
    minVal = 2;  
    maxVal = 12;  
  
    if (x >= minVal) && (x <= maxVal)  
        disp('Value within specified range.')    elseif (x > maxVal)  
        disp('Value exceeds maximum value.')    else  
        disp('Value is below minimum value.')    end  
Value within specified range.
```

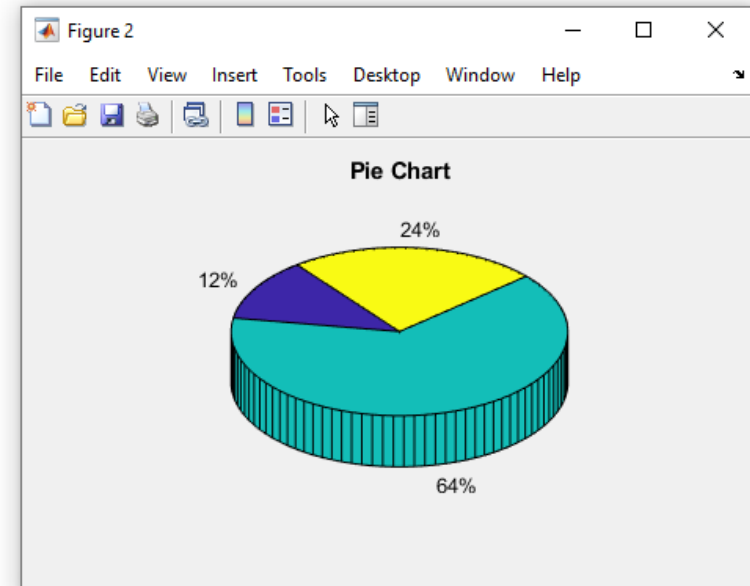

switch statement

Syntax

```
switch switch_expression
  case case_expression
    statements
  case case_expression
    statements
  ...
  otherwise
    statements
end
```

switch statement

```
>> x = [12 64 24];  
    plottype = 'pie3';  
  
    switch plottype  
        case 'bar'  
            bar(x)  
            title('Bar Graph')  
        case {'pie', 'pie3'}  
            pie3(x)  
            title('Pie Chart')  
        otherwise  
            warning('Unexpected plot type. No plot created.')  
    end
```



Vectorization

- Optimized for operations involving matrices and vectors.
- **Interpreted language**, i.e., it is not compiled before execution, loops run **slowly**.
- *Vectorized code* runs faster in MATLAB.

Example

- This code computes the *multiplication table* of 2:

```
>> x = 2;  
>> for i = 1:10  
        disp(x * i)  
    end
```

- This is a vectorized version of the same code:

```
>> timestable = (1:10)'*(2)
```

Example

- This code computes the *sine* of 1,001 values ranging from 0 to 10:

```
>> i = 0;  
    for t = 0:.01:10  
        i = i + 1;  
        y(i) = sin(t);  
    end
```

- This is a vectorized version of the same code:

```
>> t = 0:.01:10;  
    y = sin(t);
```

Overview

- **Introduction** 🖐️:

- Interface
- Documentation
- Getting help
- Common commands
- MATLAB symbols

- **Syntax** ✂️:

- Matrix manipulation
- ``:`` operator
- *for* loop
- *while* loop
- *if* statements
- *switch* statements
- Vectorization

- **Graphics** 📐:

- Common commands
- Examples - 2D plot
- Examples - 3D plot

- **Scripts & Functions** ⚡

- **Demonstration** 🖥️

- **Final word** 🔊

- **Quiz** 📋

- **Pizza** 🍕

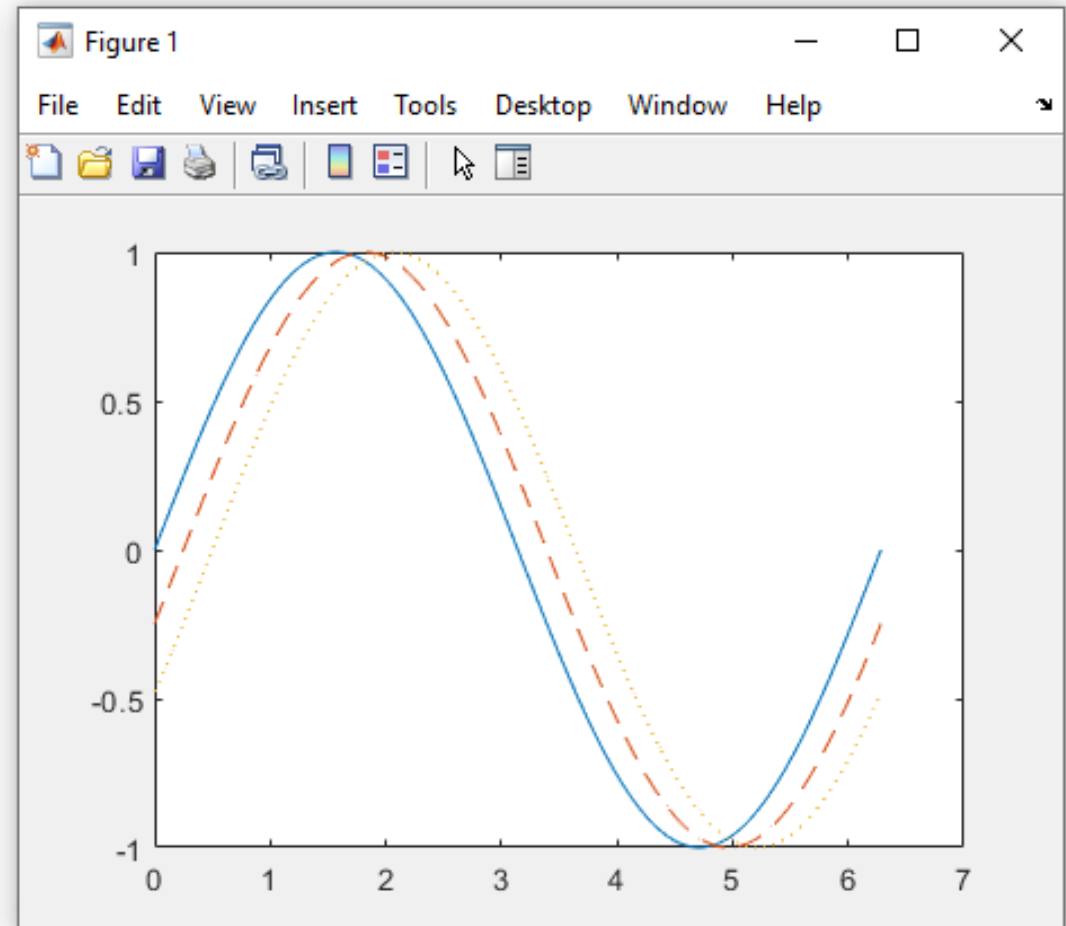
Common commands

- **plot(x, y);** % plots y vs. x.
- **plot(x, y, 'k-');** % plots a black line of y vs. x.
- **hold on;** % put several plots in the same
% figure window.
- **figure;** % open new figure window.

Common commands

```
>> x = 0:pi/100:2*pi;  
y1 = sin(x);  
y2 = sin(x-0.25);  
y3 = sin(x-0.5);  
  
figure  
plot(x,y1,x,y2,'--',x,y3,':')
```

fx >>

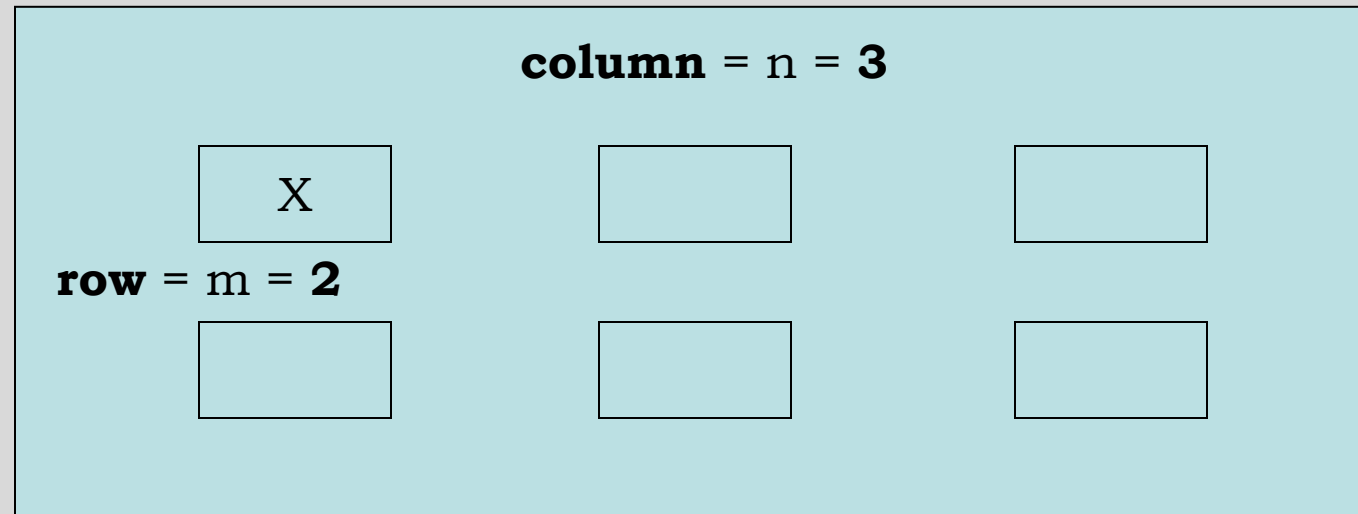


Common commands

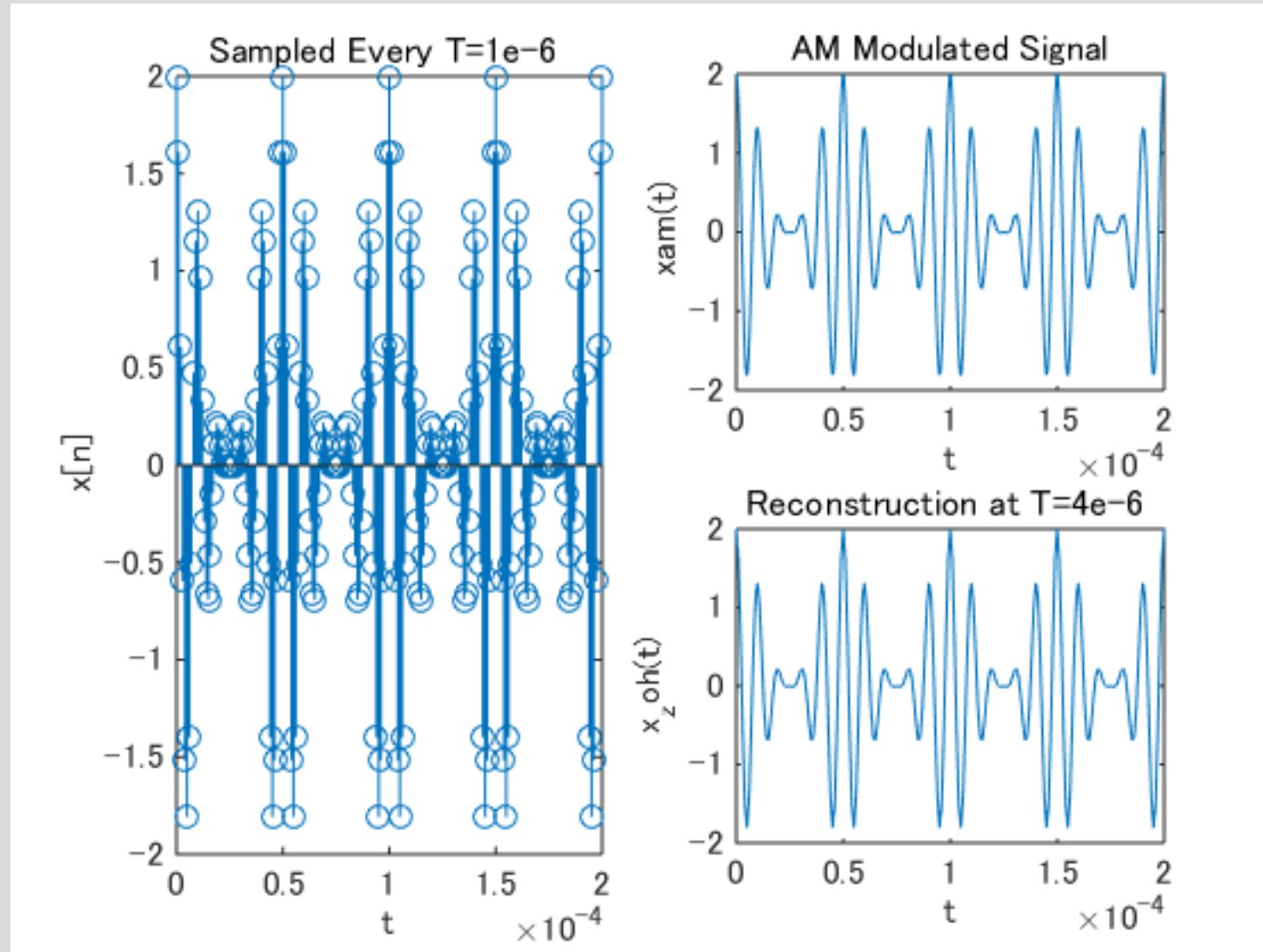
- **plot3**(x , y , z)
 - **plot** 2D function
- **mesh**(x , y , z)
 - **surface** plot
- **contour**(z)
 - **contour** plot of z
- **axis**($[x_{min} \ x_{max} \ y_{min} \ y_{max}]$)
 - change **axis** limits
- **title**('My title')
 - add **title** to figure
- **xlabel**('x label'), **ylabel**('y label')
 - **label** axes
- **legend**
 - add **key** to figure

Common commands

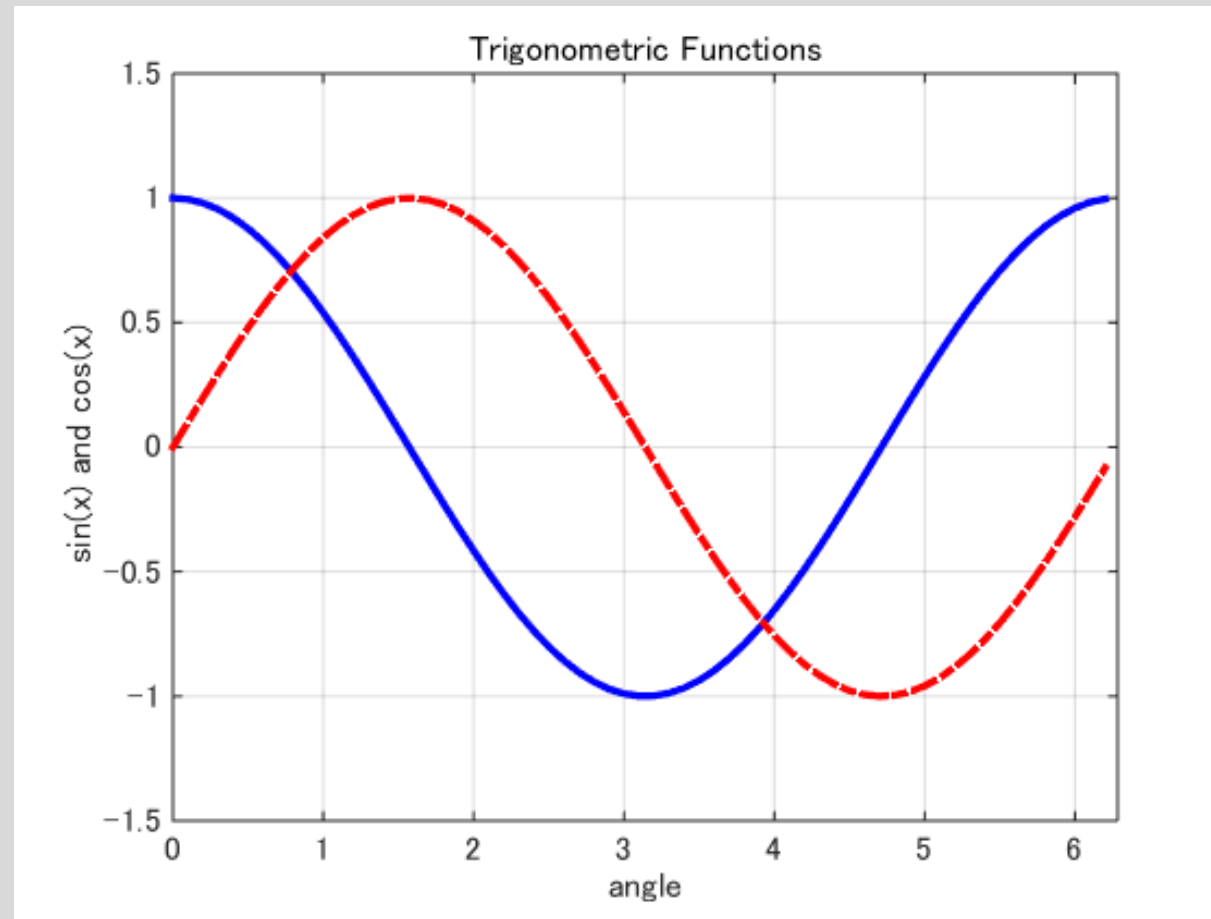
- **subplot(m, n, 1)** % Makes an **m x n** array
% for plots. Will place plot in 1st
% position.



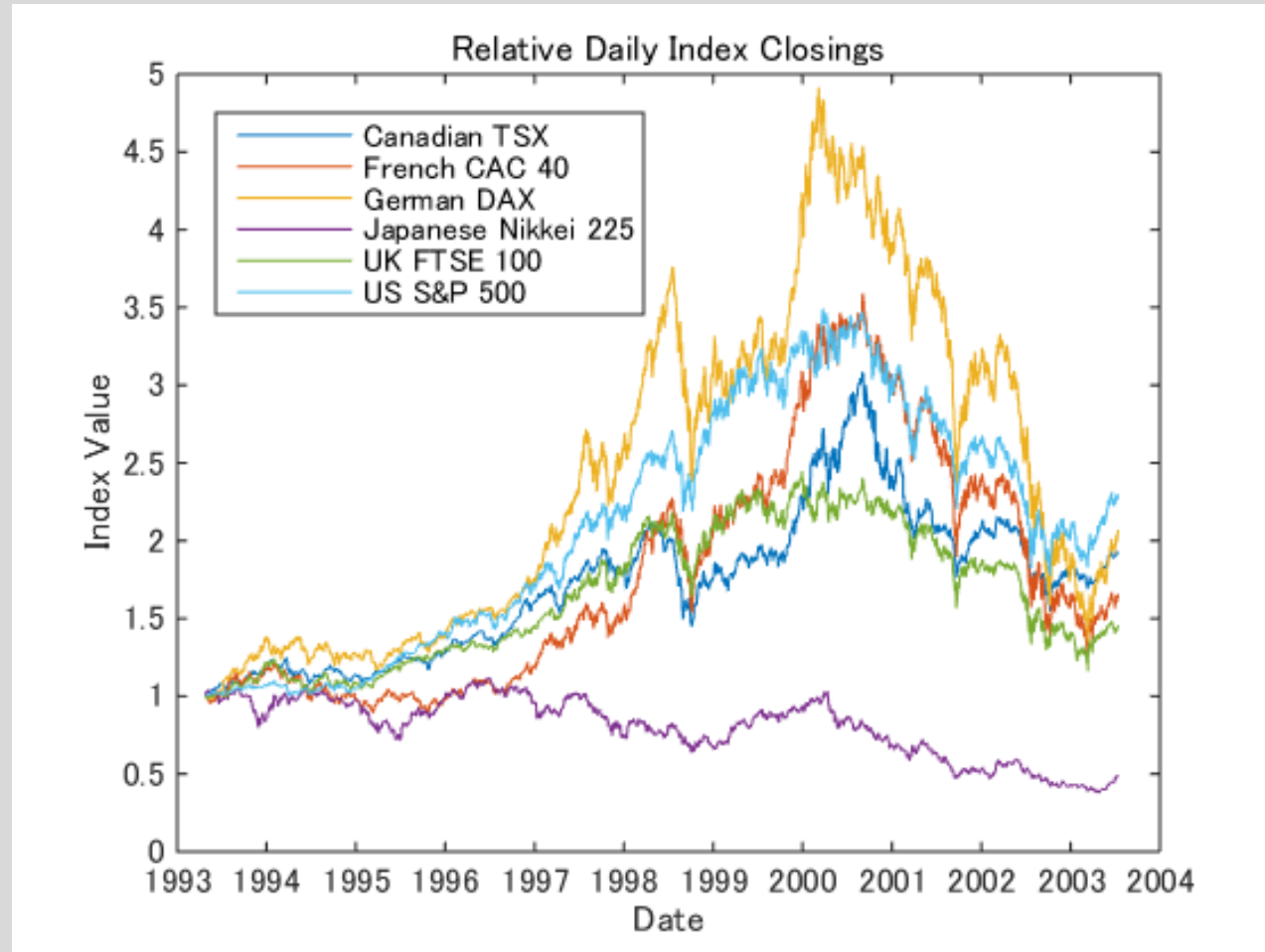
Common commands



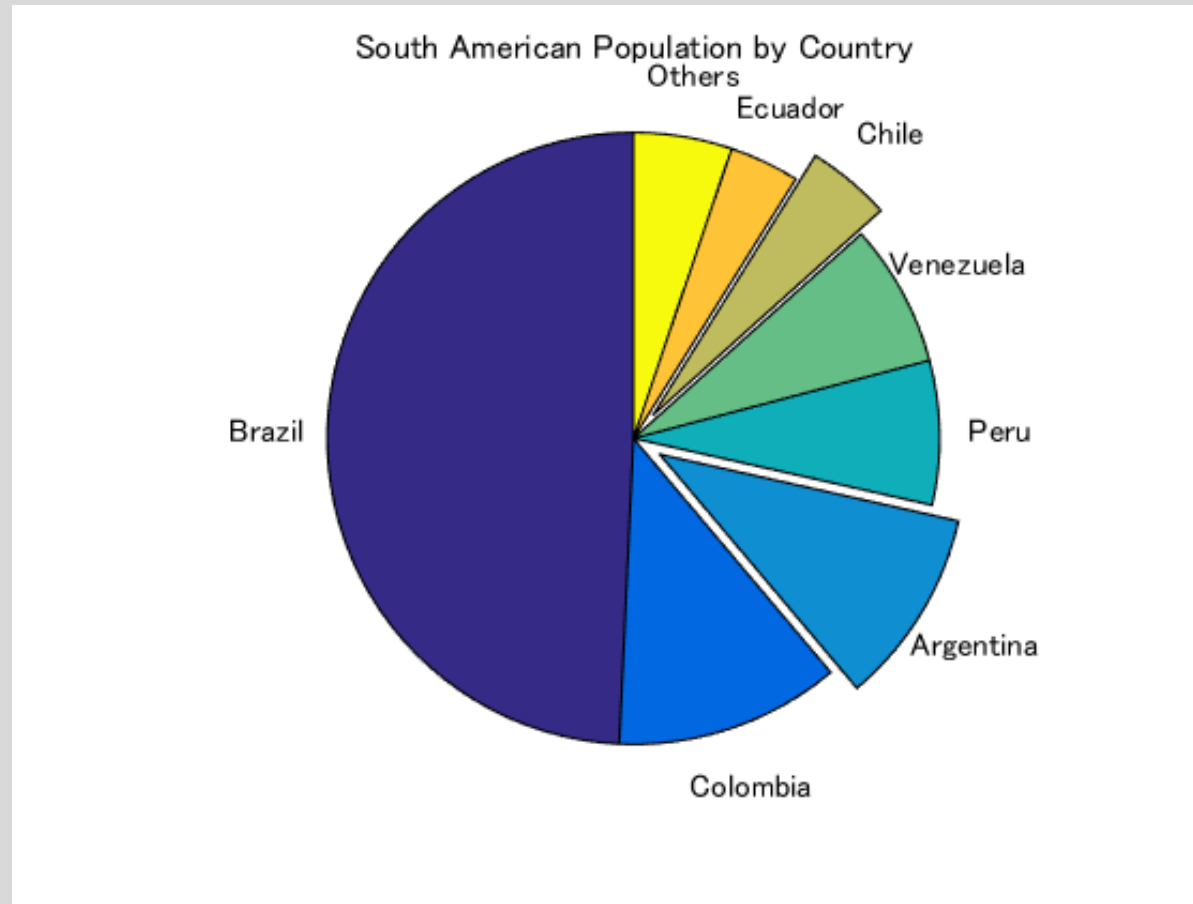
Examples of Plots – $2D$



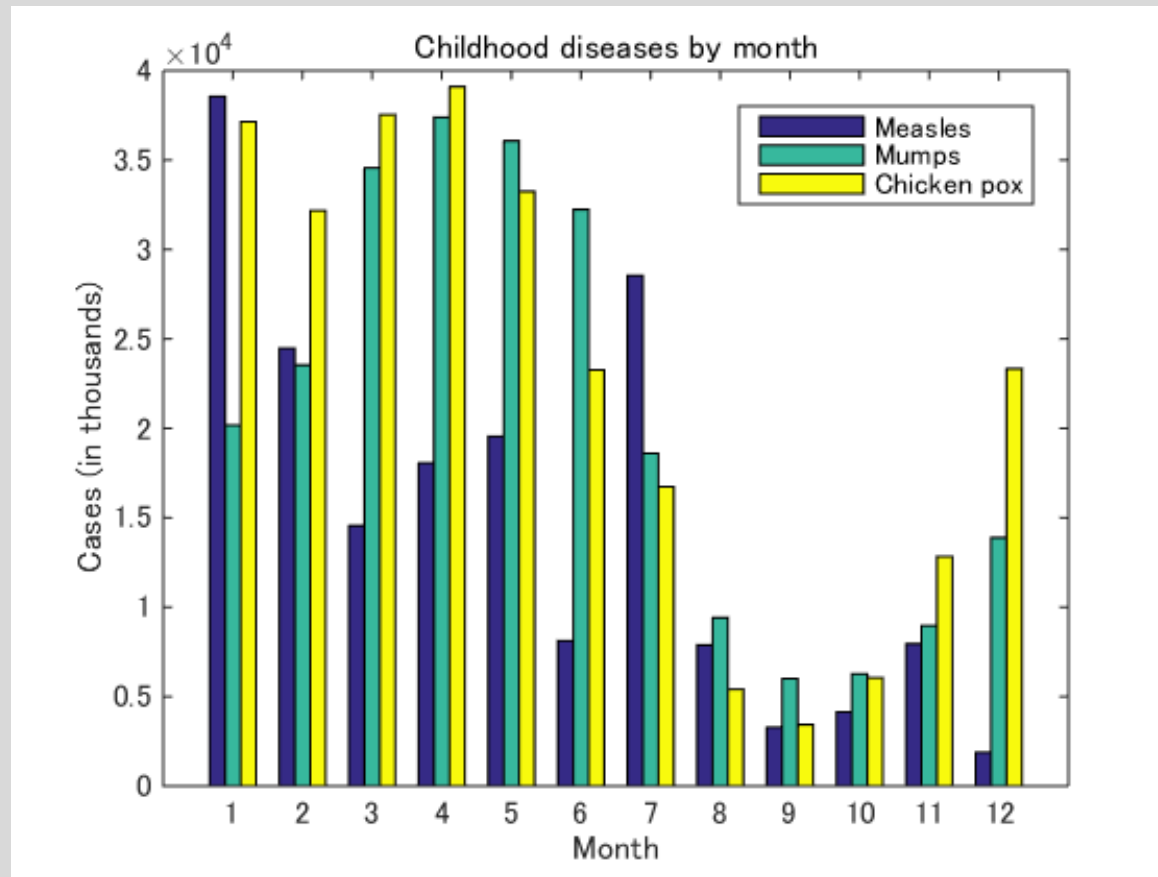
Examples of Plots – $2D$



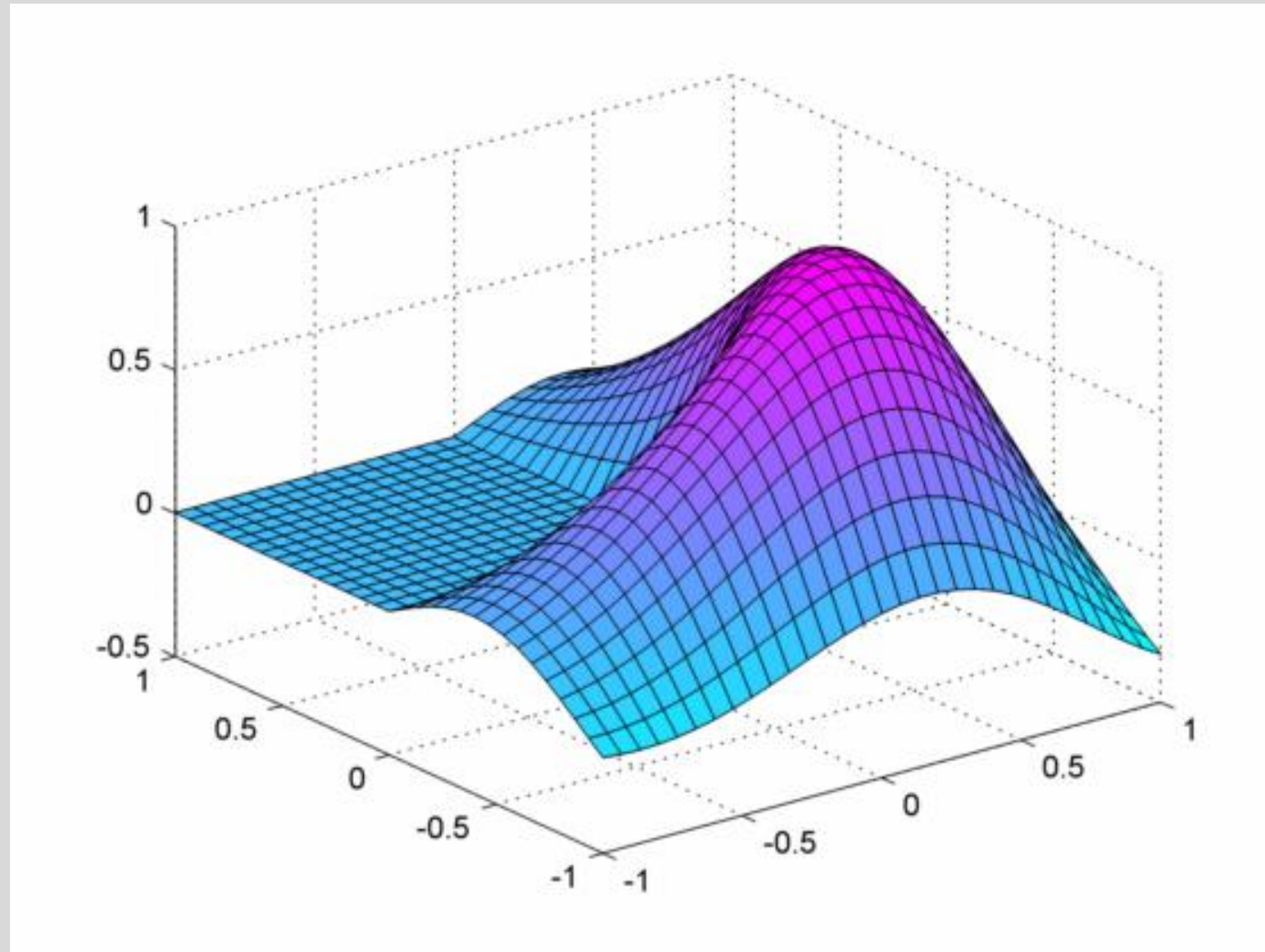
Examples of Plots – $2D$



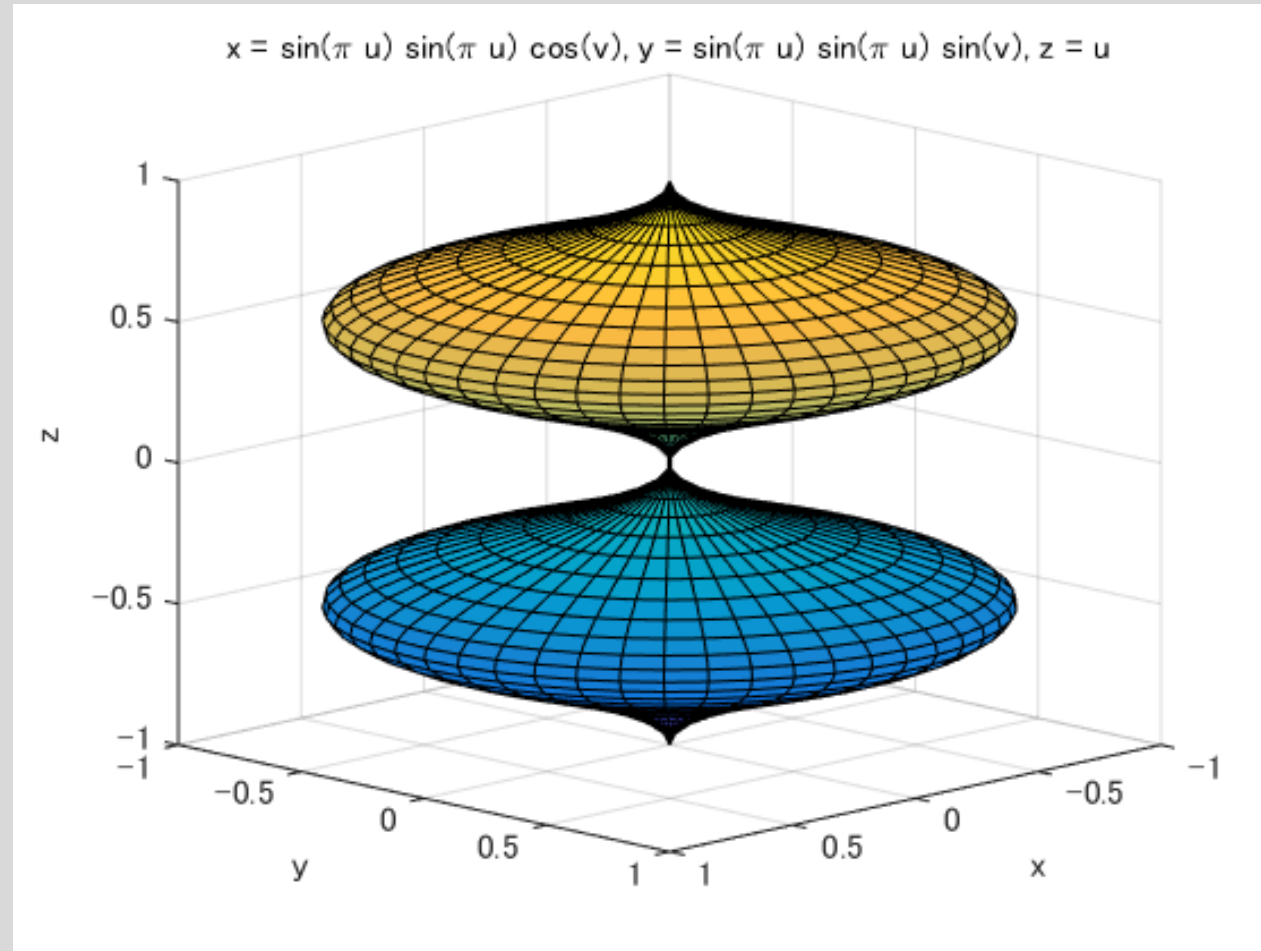
Examples of Plots – 2D



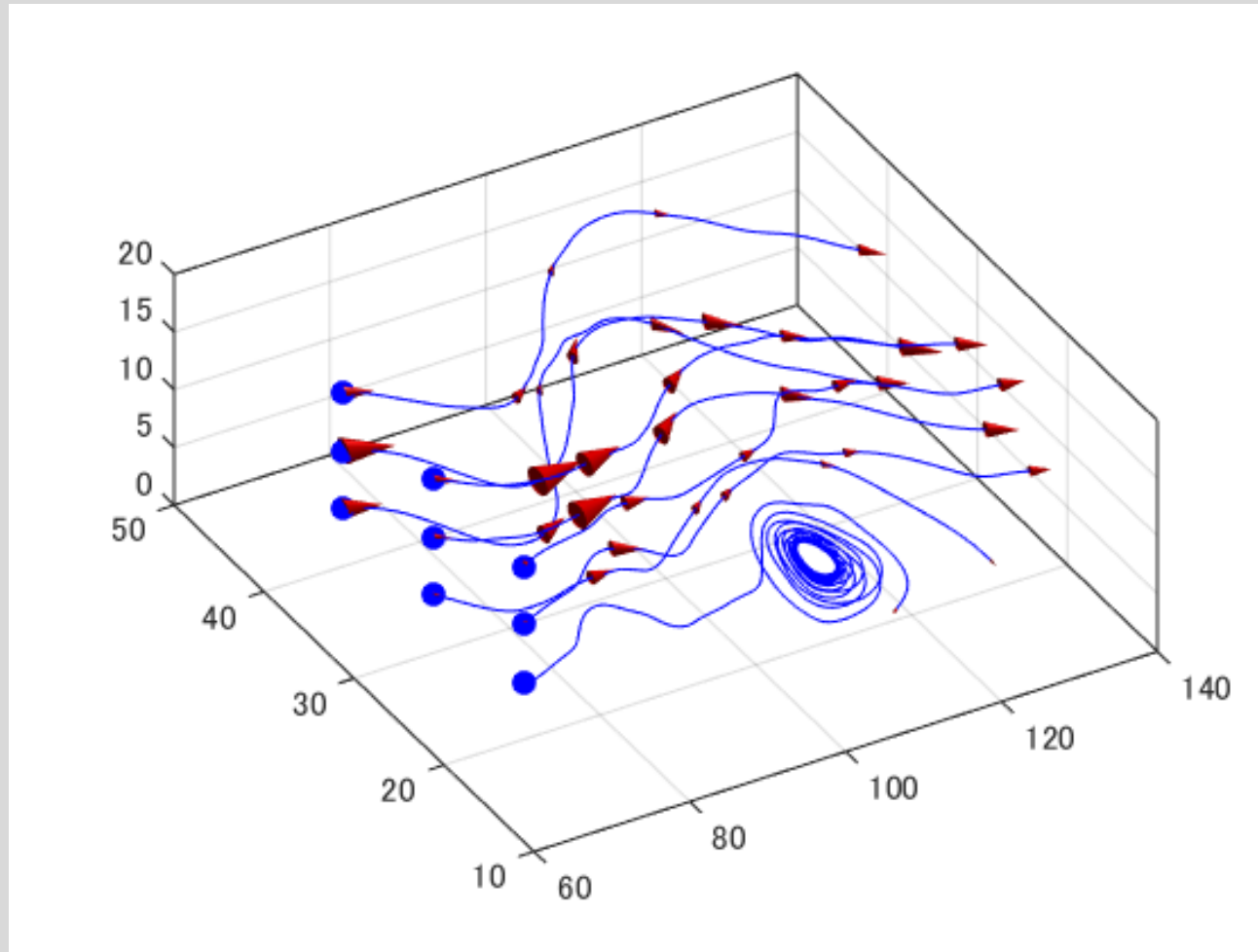
Examples of Plots – 3D



Examples of Plots – 3D



Examples of Plots – 3D



Overview

- **Introduction** 🖐️:

- Interface
- Documentation
- Getting help
- Common commands
- MATLAB symbols

- **Syntax** ✂️:

- Matrix manipulation
- `:` operator
- *for* loop
- *while* loop
- *if* statements
- *switch* statements
- Vectorization

- **Graphics** 📐:

- Common commands
- Examples - 2D plot
- Examples - 3D plot

- **Scripts & Functions** ⚡

- **Demonstration** 🖥️

- **Final word** 🔊

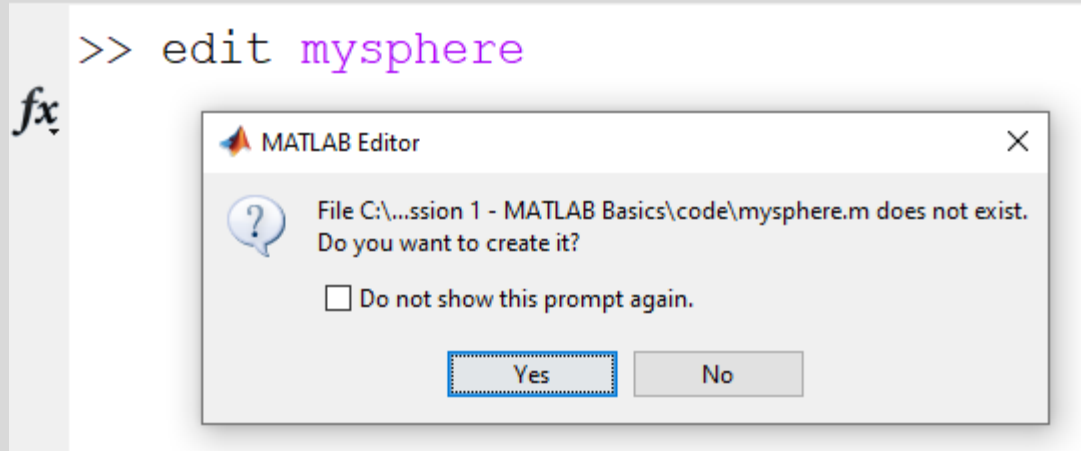
- **Quiz** 📋

- **Pizza** 🍕

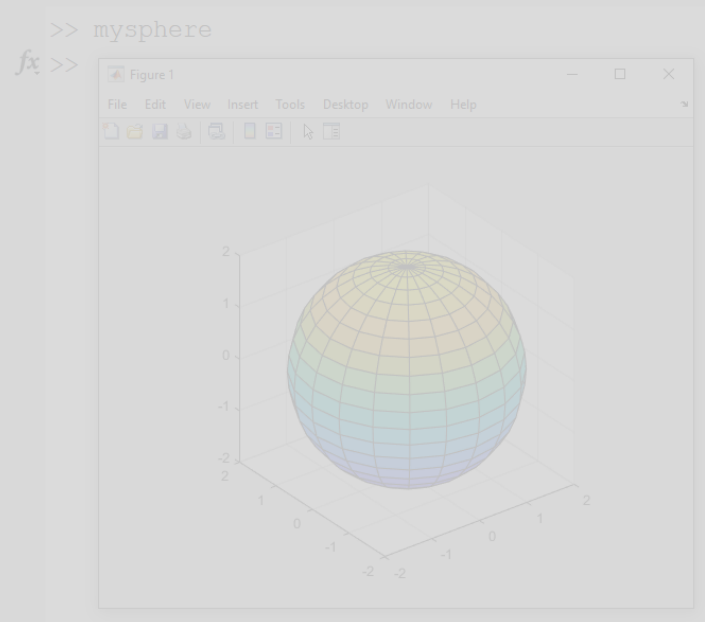
Scripts and Functions

- Two main types of `.m` files:
 - **Scripts:** do not accept input arguments or return output arguments. They operate on data in the workspace. *FIXED*
 - **Functions:** can accept input arguments and return output arguments. Internal variables are local to the function. *VARIABLE*

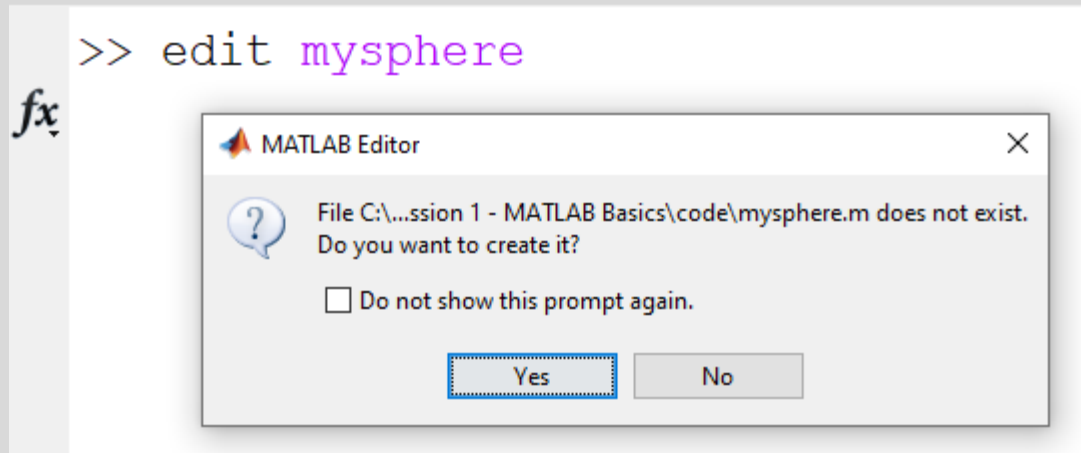
Scripts - workflow



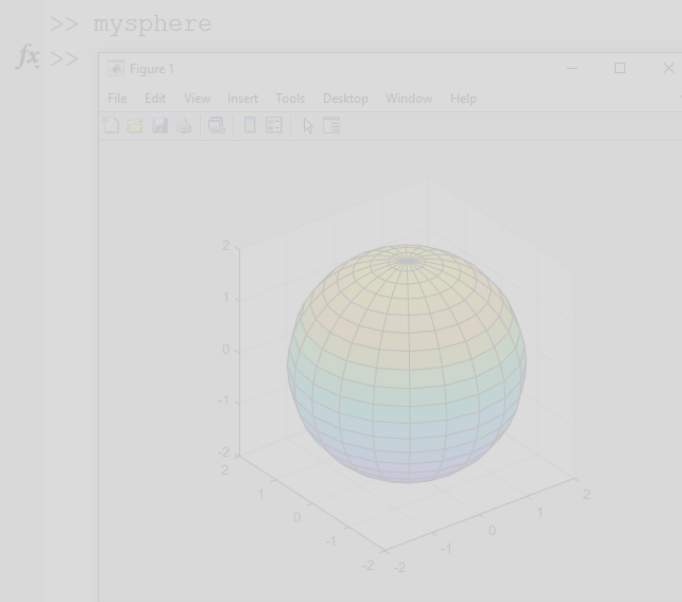
```
Editor - C:\Users\mugasi01\OneDrive - Arm\Documents\MATLAB SA\Session 1 - MATLAB Basics\code\mysphere.m
mysphere.m  x  +
1  % Create and plot a sphere with radius r.
2  [x,y,z] = sphere;           % Create a unit sphere.
3  r = 2;
4  surf(x*r,y*r,z*r)          % Adjust each dimension and plot.
5  axis equal                  % Use the same scale for each axis.
6
7  % Find the surface area and volume.
8  A = 4*pi*r^2;
9  V = (4/3)*pi*r^3;
```



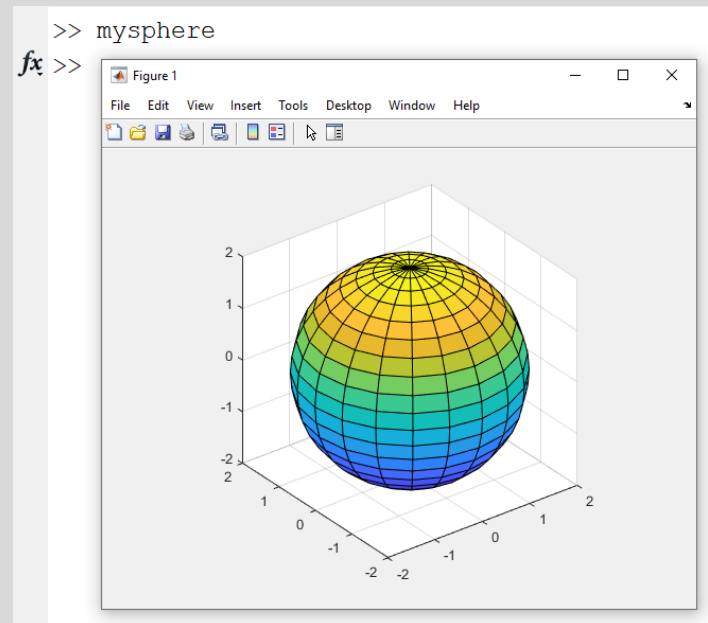
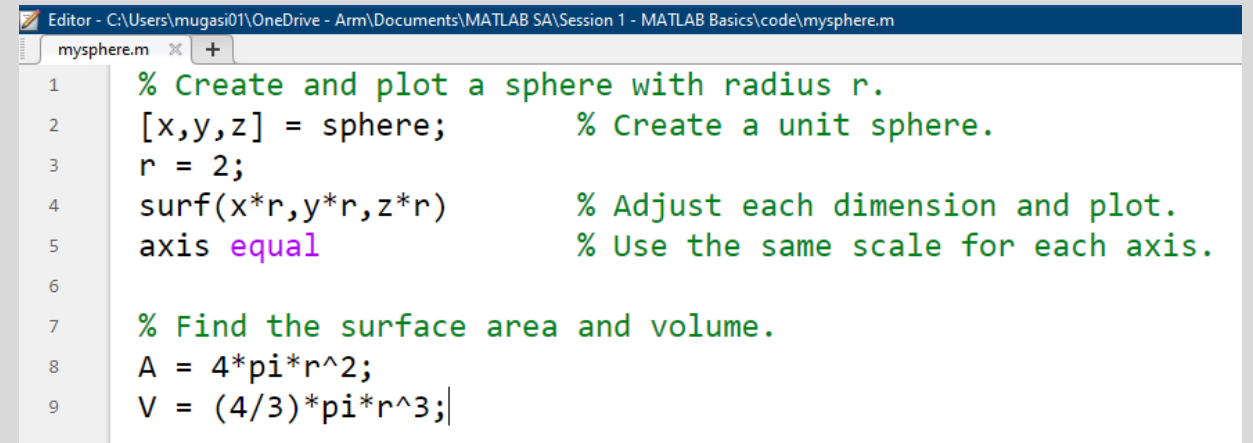
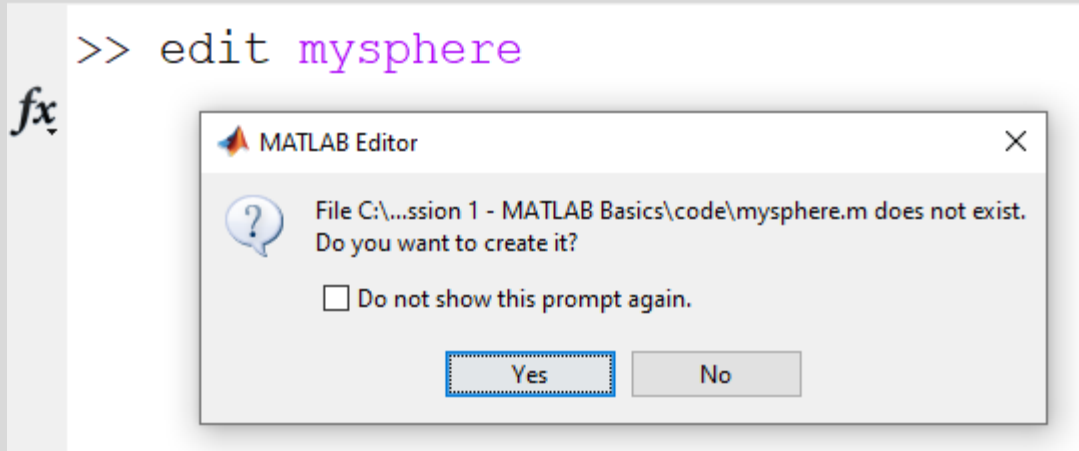
Scripts - workflow



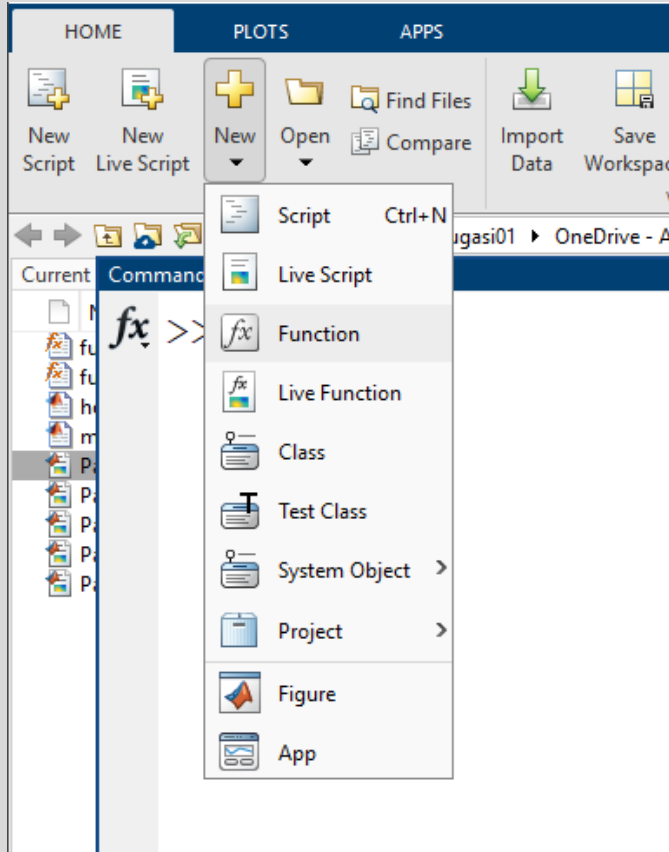
```
Editor - C:\Users\mugasi01\OneDrive - Arm\Documents\MATLAB SA\Session 1 - MATLAB Basics\code\mysphere.m
mysphere.m x +
1 % Create and plot a sphere with radius r.
2 [x,y,z] = sphere; % Create a unit sphere.
3 r = 2;
4 surf(x*r,y*r,z*r) % Adjust each dimension and plot.
5 axis equal % Use the same scale for each axis.
6
7 % Find the surface area and volume.
8 A = 4*pi*r^2;
9 V = (4/3)*pi*r^3;
```



Scripts - workflow



Functions - workflow



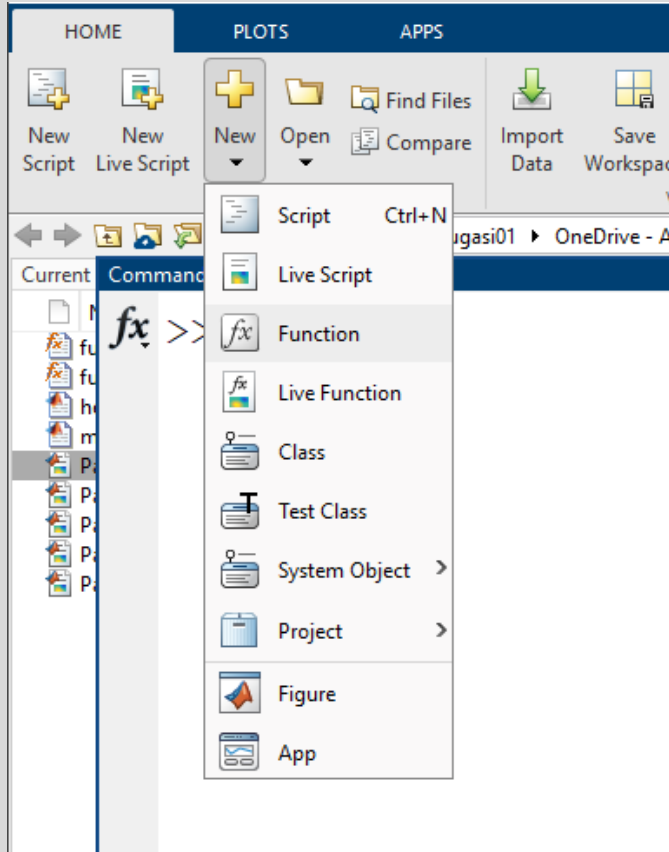
```
Editor - C:\Users\mugasi01\OneDrive - Arm\Documents\MATLAB SA\Session 1 - MATLAB Basics\code\average.m
average.m
1 % Define a function in a file named average.m that
2 % accepts an input vector,
3 % calculates the average of the values,
4 % and returns a single result.
5
6 function ave = average(x)
7     ave = sum(x(:))/numel(x);
8 end
```

```
>> z = 1:20;
    ave = average(z)
```

```
ave =
```

```
10.5000
```


Functions - workflow



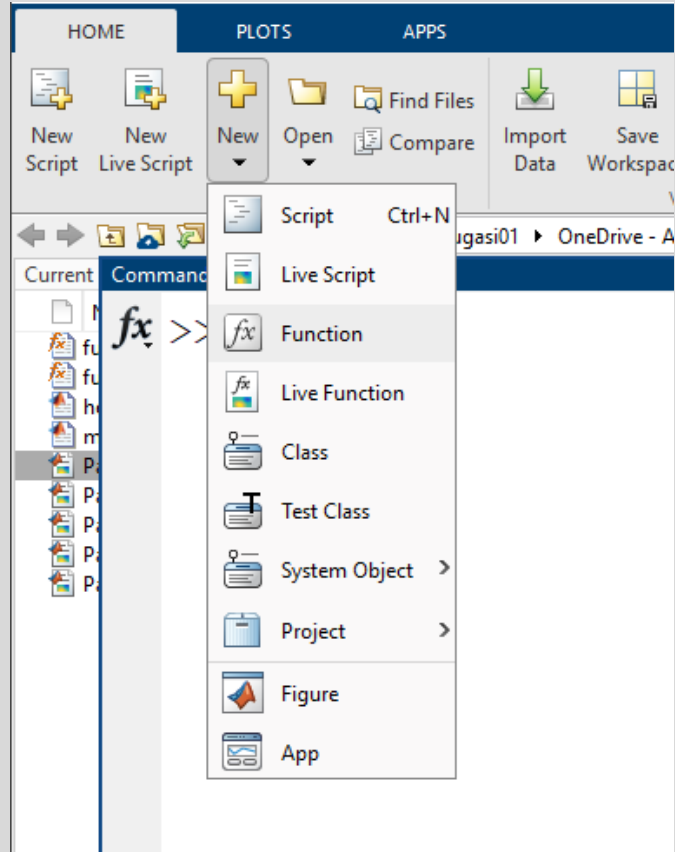
```
Editor - C:\Users\mugasi01\OneDrive - Arm\Documents\MATLAB SA\Session 1 - MATLAB Basics\code\average.m
average.m x +
1 % Define a function in a file named average.m that
2 % accepts an input vector,
3 % calculates the average of the values,
4 % and returns a single result.
5
6 function ave = average(x)
7     ave = sum(x(:))/numel(x);
8 end
```

```
>> z = 1:20;
    ave = average(z)

ave =

    10.5000
```

Functions - workflow



```
Editor - C:\Users\mugasi01\OneDrive - Arm\Documents\MATLAB SA\Session 1 - MATLAB Basics\code\average.m
average.m x +
1 % Define a function in a file named average.m that
2 % accepts an input vector,
3 % calculates the average of the values,
4 % and returns a single result.
5
6 function ave = average(x)
7     ave = sum(x(:))/numel(x);
8 end
```

```
>> z = 1:20;
    ave = average(z)

ave =

    10.5000
```




Overview

- **Introduction** 🖐️:
 - Interface
 - Documentation
 - Getting help
 - Common commands
 - MATLAB symbols
- **Syntax** ✂️:
 - Matrix manipulation
 - `:` operator
 - *for* loop
 - *while* loop
 - *if* statements
 - *switch* statements
 - Vectorization
- **Graphics** 📐:
 - Common commands
 - Examples - 2D plot
 - Examples - 3D plot
- **Scripts & Functions** ⚡
- **Demonstration** 🖥️
- **Final word** 🔊
- **Quiz** 📋
- **Pizza** 🍕

```
function [y,dist] = euclidean(x,cb) %#eul
% Initialize minimum distance as first element of cb
idx=1;
dist=norm(x-cb(:,1));
% Find the vector in cb with minimum distance to x
for i=2:size(cb,2)
    d=norm(x-cb(:,i));
    if d < dist
        dist=d;
        idx=i;
    end
    plot_distances(x,cb,i);
end
% Output the minimum distance vector
y=cb(:,idx);
end
```

Let's look
at some
code now!

<https://matlabacademy.mathworks.com/#getting-started>

 [Products](#) [Solutions](#) [Academia](#) [Support](#) [Community](#) [Events](#) [Get MATLAB](#)  


MATLAB and Simulink Training


[Training Overview](#) [Find a Course](#) [Get Certified](#) [Training at Your Facility](#) [More](#) [» My Courses](#) [Contact Training](#)


MATLAB Onramp


Learn the essentials of MATLAB® through this free, two-hour introductory tutorial on commonly used features and workflows.


[Details and launch](#)




Access to MATLAB through your web browser


Engaging video tutorials


Hands-on exercises with automated assessments and feedback



Lessons available in English, Chinese, Spanish, Japanese, and Korean

Browse self-paced online courses

Getting Started (12)

- MATLAB (4)
- Simulink (5)
- AI, Machine Learning, and Deep Learning (5)
- Math and Optimization (6)
- Image and Signal Processing (3)
- Explore over 50 virtual and in-person **classroom courses**


Getting Started



MATLAB Onramp

15 modules | 2 hours | Languages


Get started quickly with the basics of MATLAB.



Simulink Onramp

14 modules | 2 hours | Languages


Get started quickly with the basics of Simulink.



Circuit Simulation Onramp

7 modules | 2 hours | Languages


Learn the basics of simulating electrical circuits in Simscape.



Machine Learning Onramp

6 modules | 2 hours | Languages


Learn the basics of practical machine learning methods for classification problems.



Deep Learning Onramp

5 modules | 2 hours | Languages

Get started quickly using deep learning methods to perform image recognition.



Reinforcement Learning Onramp

5 modules | 3 hours | Languages

Master the basics of creating intelligent controllers that learn from experience.





Image Processing Onramp

6 modules | 2 hours | Languages

Learn the basics of practical image processing techniques in MATLAB.



Signal Processing Onramp

7 modules | 1 hour | Languages

An interactive introduction to signal processing methods for spectral analysis.

Thank you!

MATLAB® & SIMULINK®

Join the FB group to stay up to date with future events:

<https://www.facebook.com/groups/196042678284982>

The code and presentation can be downloaded from:

<https://github.com/mughees-asif/matlab-qmul>

MATLAB Workshops for QMUL



Queen Mary
University of London



Contents

Help

- [Onramps: self-paced tutorials](#)
- [Getting Started with MATLAB](#)
- [Naming Conventions](#)

Access the code

- Clone the repository to your local machine: `git clone https://github.com/mughees-asif/matlab-qmul.git`
- After accessing the sessions folders:
 - use the **PDF versions** for learning
 - use the **MATLAB versions** to change input parameters and work with example code

Sessions

Session #	Topics covered	Presentation	Code	Cheatsheet
1	MATLAB Introduction	Link	MATLAB --- PDF	Link

Contributing

Pull Request Process

- **Github Flow:** So all `code` changes happen through pull requests:
 - [Fork](#) the repo and create your branch from `master`.
 - If you've added code that should be tested, add tests.
 - If you've changed APIs, update the documentation.
 - Ensure the test suite passes.
 - Make sure your code lints.
 - Issue that [pull request!](#)
 - Will be reviewed and merged.