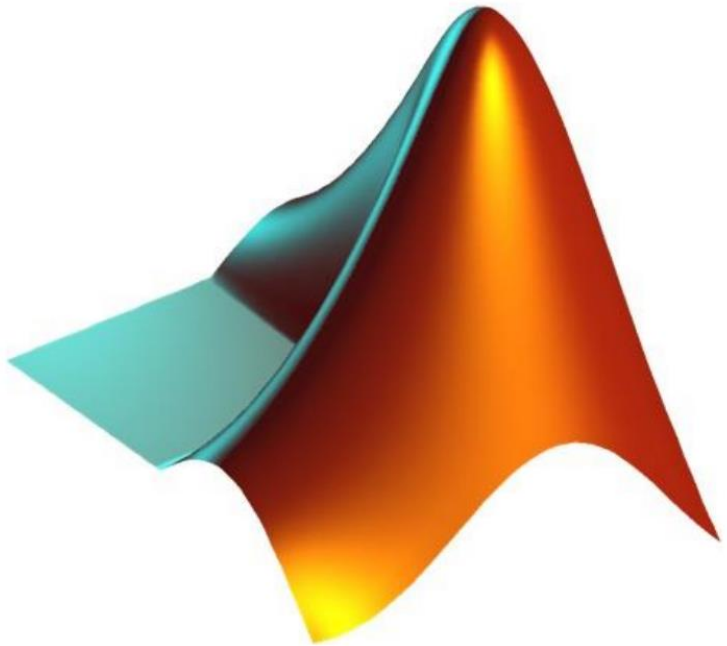
Practical Course MATLAB/SIMULINK

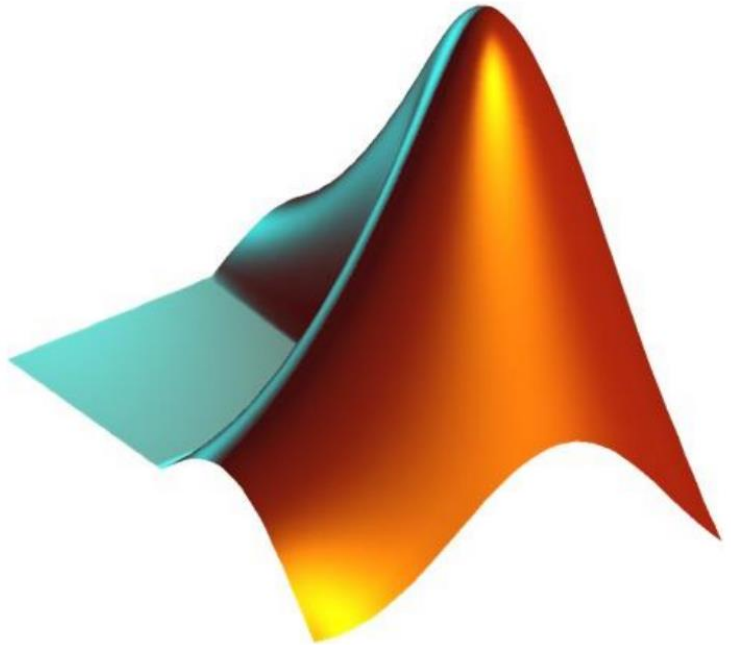
Session 3: Data Handling and Visualization



Lecture Objectives & Preparation

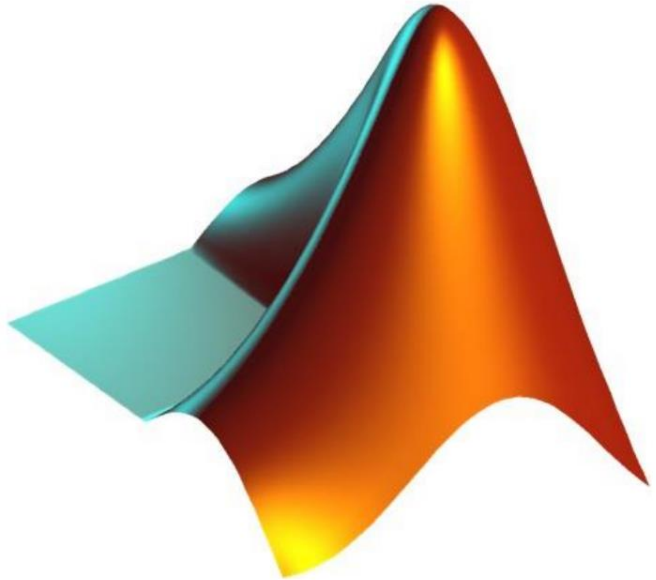
- Which MathWorks products are covered?
  - MATLAB
- What skills are learnt?
  - Data types, import and export
  - Memory management
  - Visualization, graphics tools
- How to prepare for the session?
  - MathWorks Tutorials:
    - https://matlabacademy.mathworks.com/details/matlab-for-data-processing-and-visualization/mlvi
    - https://www.mathworks.com/examples/matlab/category/graphics

Lecture Outline



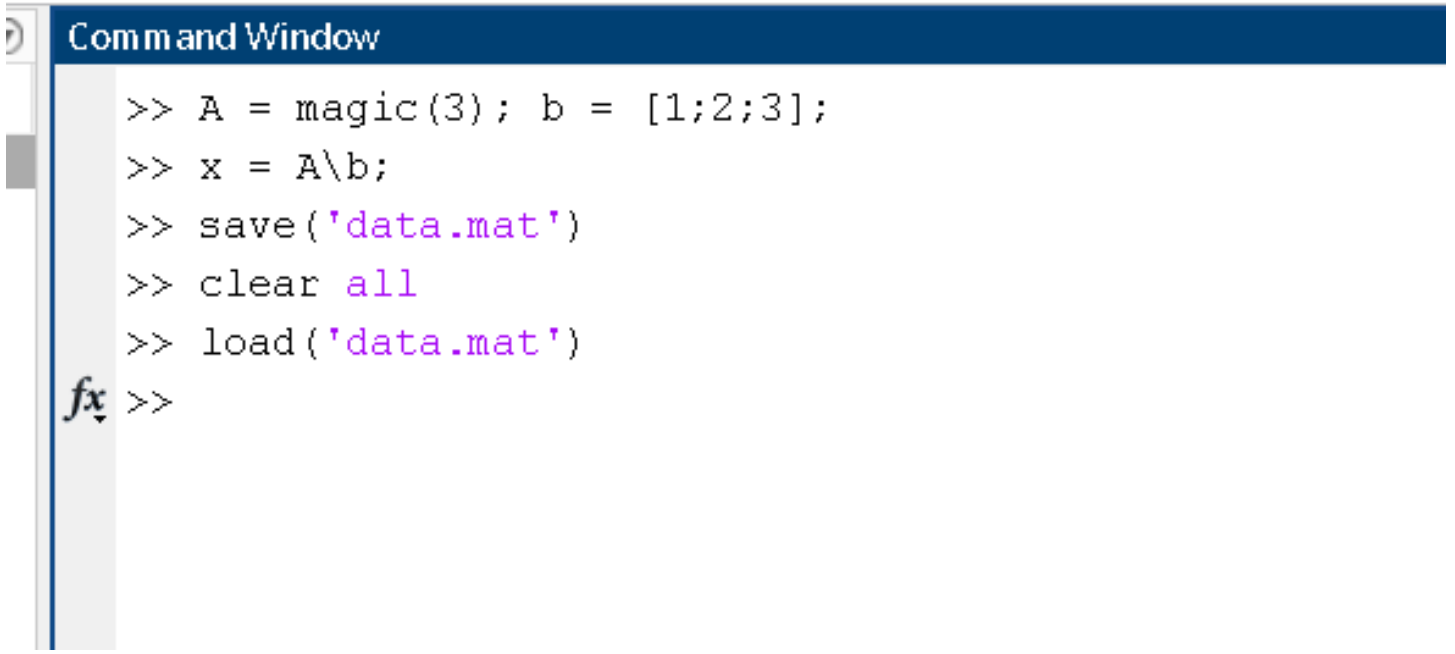
1. Import and Export Data

1.1. Import of Various Data Formats



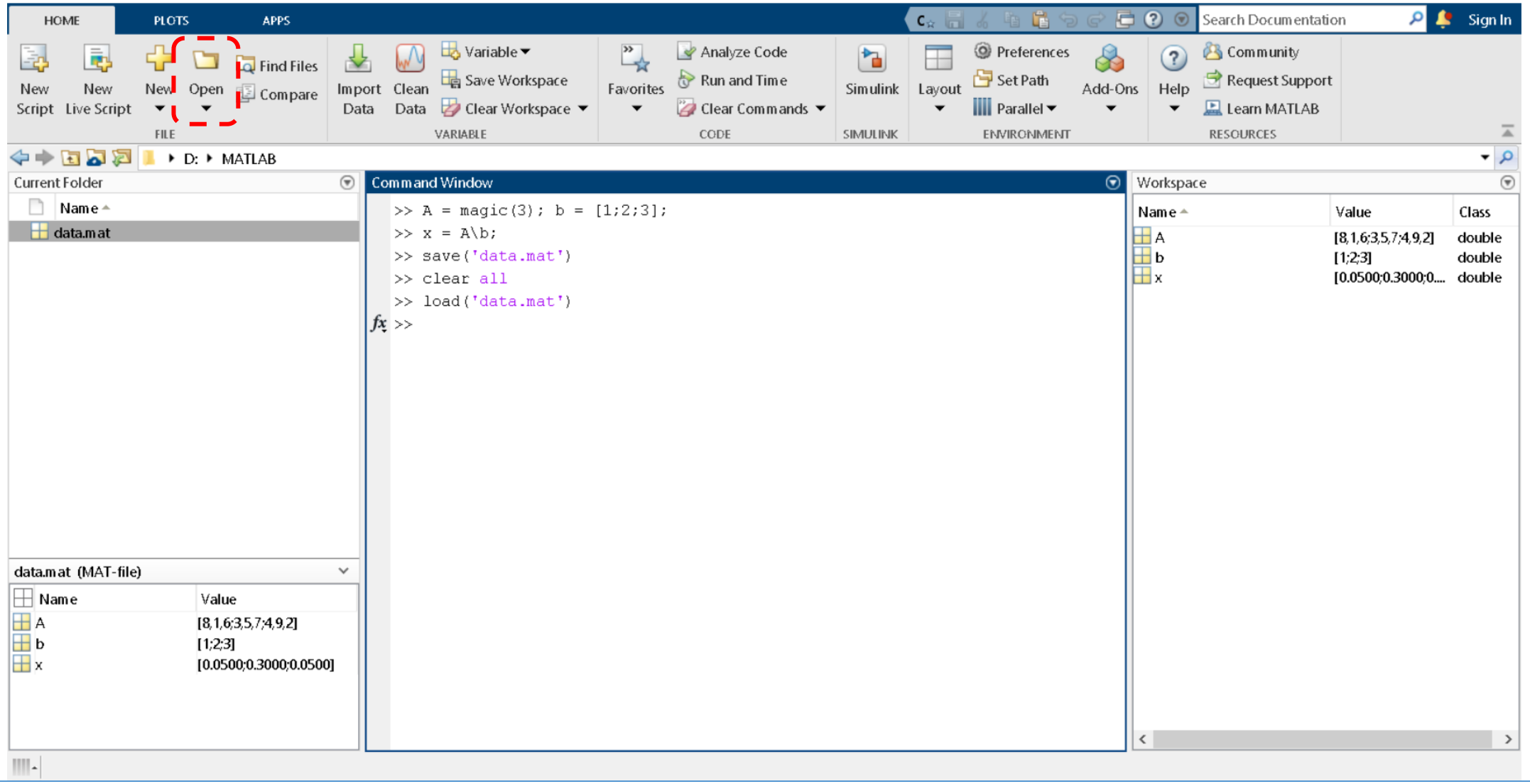
Loading MAT-Files

- **MAT-Files** are:
  - **Binary** file containing MATLAB **formatted data**
  - Used to **save variables** from the MATLAB **workspace**
- Use save and load commands.



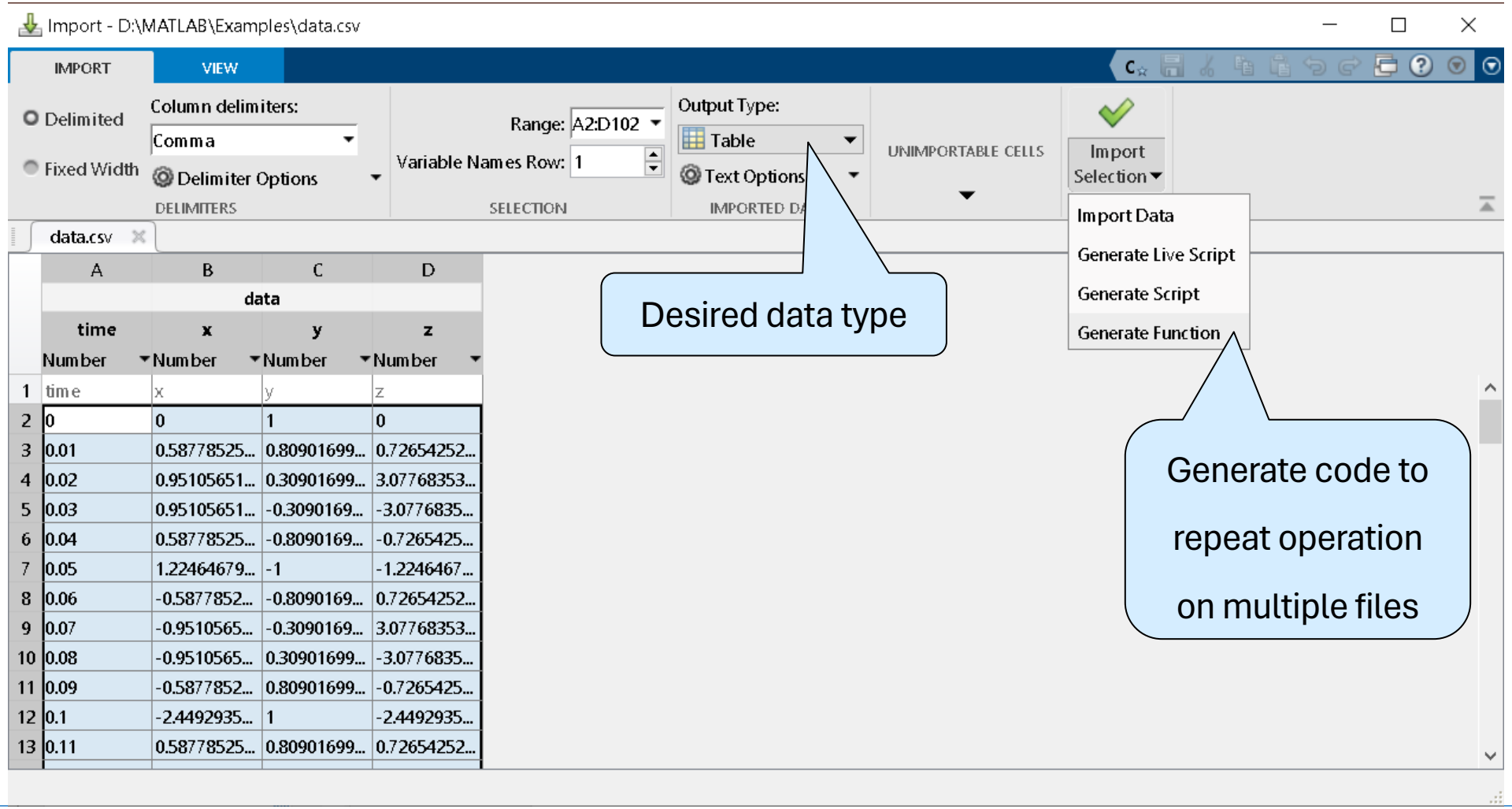
Loading MAT-Files

- Mat-files can be loaded **interactively** using the Open button.



Importing Text Files Interactively

- **Text files** (including .txt, .dat, .csv, .asc, .tab, and .dlm) can be imported using MATLAB's **Import Tool**:



	A	B	C	D
		data		
	time	x	y	z
	Number	Number	Number	Number
1	time	x	y	z
2	0	0	1	0
3	0.01	0.58778525...	0.80901699...	0.72654252...
4	0.02	0.95105651...	0.30901699...	3.07768353...
5	0.03	0.95105651...	-0.3090169...	-3.0776835...
6	0.04	0.58778525...	-0.8090169...	-0.7265425...
7	0.05	1.22464679...	-1	-1.2246467...
8	0.06	-0.5877852...	-0.8090169...	0.72654252...
9	0.07	-0.9510565...	-0.3090169...	3.07768353...
10	0.08	-0.9510565...	0.30901699...	-3.0776835...
11	0.09	-0.5877852...	0.80901699...	-0.7265425...
12	0.1	-2.4492935...	1	-2.4492935...
13	0.11	0.58778525...	0.80901699...	0.72654252...

Importing Text Files Programmatically

- Text files can also be imported **programmatically**.
- As the **first output**, a **matrix**, multidimensional **array** or a **scalar structure array** is returned depending on the characteristics of the file.

```
>> [data, delimiter, headerlines] = importdata('data.csv');
>> data
data =
          data: [201x4 double]
      textdata: {'time'  'x'  'y'  'z'}
    colheaders: {'time'  'x'  'y'  'z'}
>> delimiter
delimiter =
;
>> headerlines
headerlines =
     1
```

Commands to Import Text Files Programmatically

Import Option	Description
readtable	Import column-oriented data into a table.
csvread	Import a file or range of comma-separated numeric data to a matrix.
dlmread	Import a file or a range of numeric data separated by any single delimiter to a matrix.
TabularTextDatastore	Import one or more column-oriented text files. Each file can be very large and does not need to fit in memory.
textscan	Import a nonrectangular or arbitrarily formatted text file to a cell array.

readmatrix is now the preferred over csvread and dlmread

Commands to Import Text Files Programmatically

- readtable creates **table** from data:

```
>> tabledata = readtable('data.csv','delimiter',';');
>> whos tabledata
```

```
  Name              Size              Bytes  Class    Attributes

  tabledata       201x4               8596  table
```

- csvread imports comma separated numeric data to a **matrix**:

```
>> csvdata = csvread('data.txt',1,0); whos csvdata
```

```
  Name              Size              Bytes  Class     Attributes

  csvdata         201x4               6432  double
```

Input arguments R(1) und C(0) specify offset of first row/column

Commands to Import Text Files Programmatically

- textscan reads **formatted data** from text file or string
  - The first row contains the data **header**:

```
>> fid = fopen('data.txt');
>> Header = textscan(fid, '%s',4,'Delimiter',','); disp(Header{:}')
    'time'    'x'    'y'    'z'
```

  - From this **cursor position** data can be read using a **different format specification**:

```
>> [num, position] = textscan(fid, '%f,%f,%f,%f'); whos num
  Name      Size              Bytes  Class    Attributes
  num       1x4                6880  cell
```

Commands to Import Text Files Programmatically

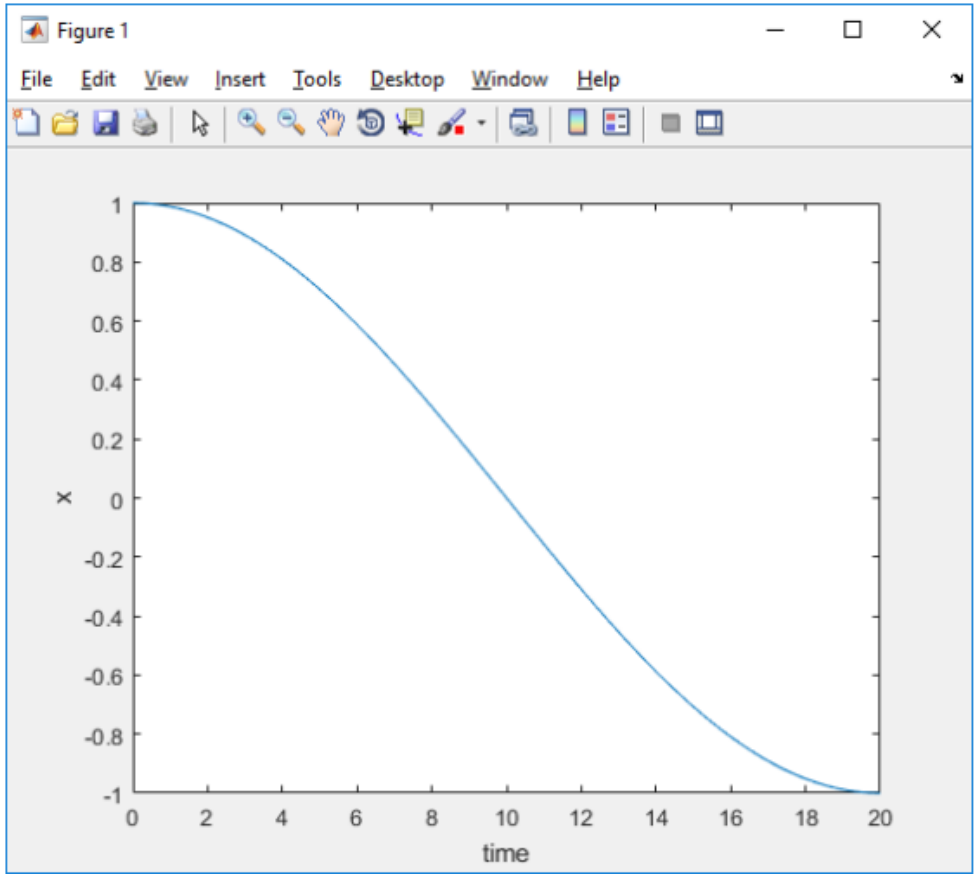
```
>> [num, position] = textscan(fid, '%f,%f,%f,%f'); whos num
  Name        Size                Bytes  Class     Attributes
  num         1x4                  6880  cell
```

position contains the file or string

position at the end of the scan

- The cell array num contains a cell for each data column:

```
>> plot(num{1},num{2})
```



Importing Spreadsheets

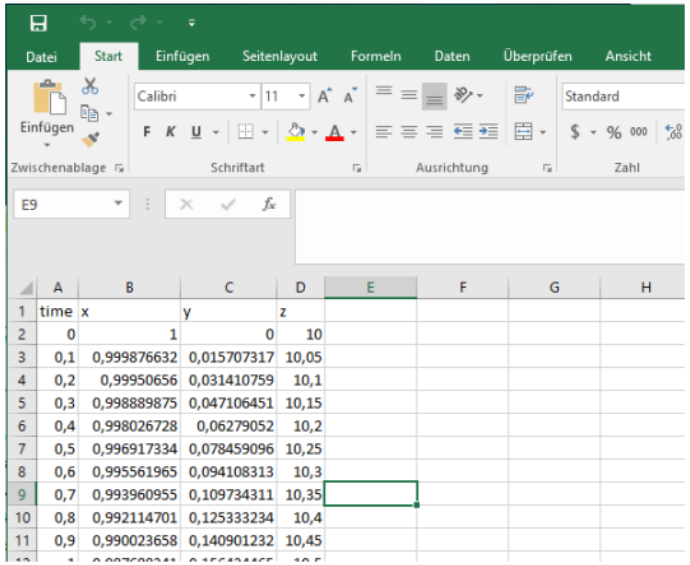
- MATLAB offers the xlsread command to read Microsoft Excel **spreadsheet** files:

```
>> [num, txt, raw] = xlsread('data.xlsx'); whos
```

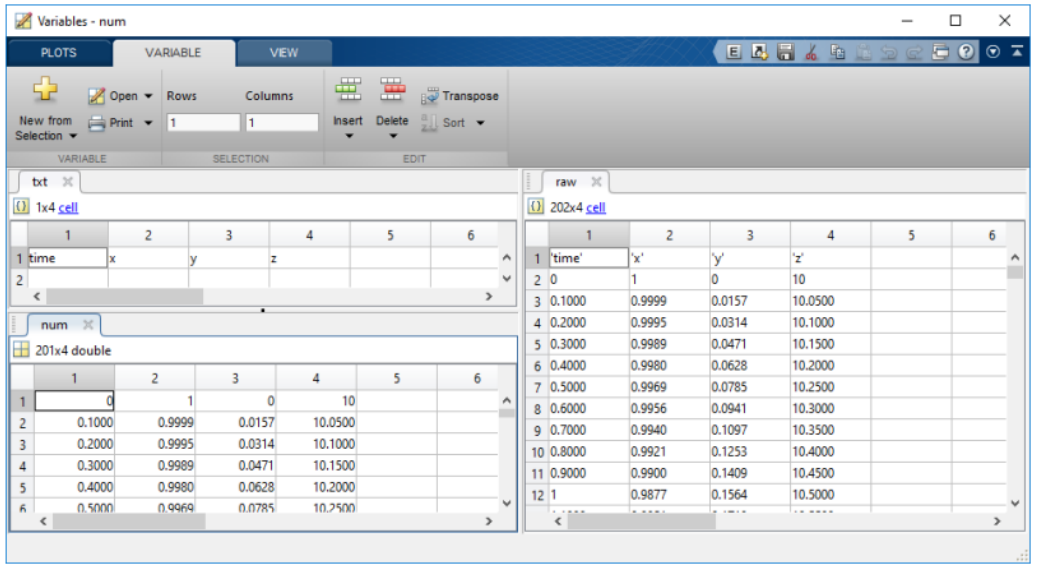
```
  Name          Size                Bytes  Class     Attributes

  num         201x4                  6432  double
  raw         202x4                 96942  cell
  txt           1x4                   462  cell
```

num	contains numeric data
raw	cell array of all data as char arrays
txt	cell array containing only text data



	A	B	C	D	E	F	G	H
1	time	x	y	z				
2	0	1	0	10				
3	0,1	0,999876632	0,015707317	10,05				
4	0,2	0,99950656	0,031410759	10,1				
5	0,3	0,998889875	0,047106451	10,15				
6	0,4	0,998026728	0,06279052	10,2				
7	0,5	0,996917334	0,078459096	10,25				
8	0,6	0,995561965	0,094108313	10,3				
9	0,7	0,993960955	0,109734311	10,35				
10	0,8	0,992114701	0,125333234	10,4				
11	0,9	0,990023658	0,140901232	10,45				



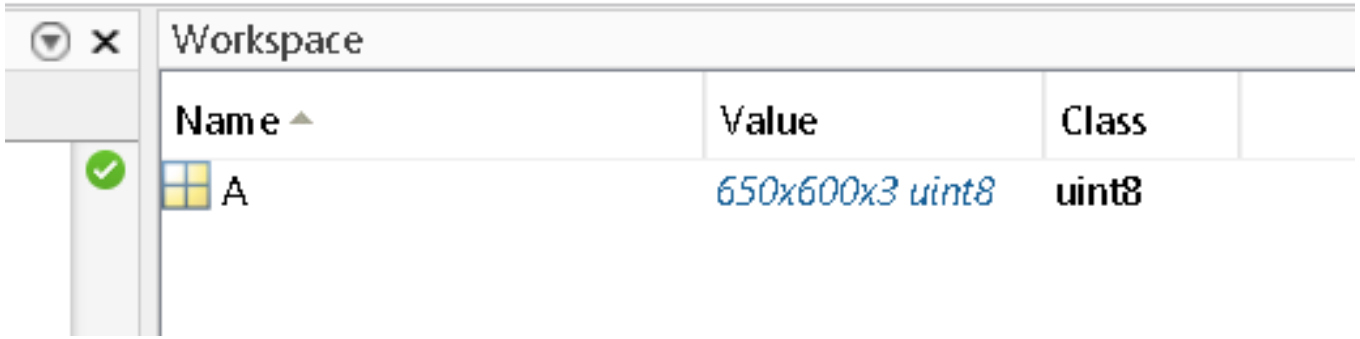
	1	2	3	4	5	6
1	time	x	y	z		
2						

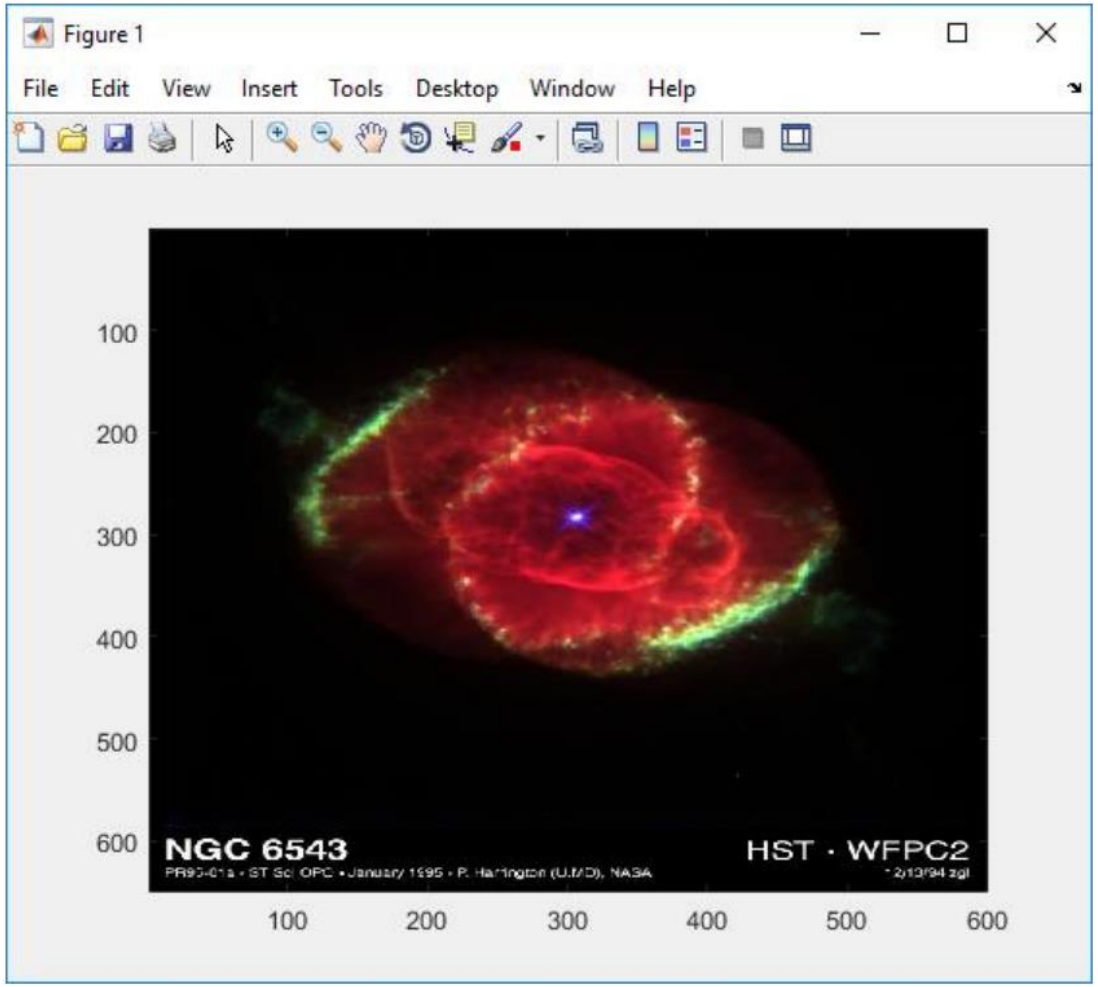
	1	2	3	4	5	6
1	'time'	'x'	'y'	'z'		
2	0	1	0	10		
3	0.1000	0.9999	0.0157	10.0500		
4	0.2000	0.9995	0.0314	10.1000		
5	0.3000	0.9989	0.0471	10.1500		
6	0.4000	0.9980	0.0628	10.2000		
7	0.5000	0.9969	0.0785	10.2500		
8	0.6000	0.9956	0.0941	10.3000		
9	0.7000	0.9940	0.1097	10.3500		
10	0.8000	0.9921	0.1253	10.4000		
11	0.9000	0.9900	0.1409	10.4500		
12	1	0.9877	0.1564	10.5000		

Importing Images

- MATLAB can import **images** files in many **standard file formats** (TIFF, GIF, JPEG, PNG...)

```
>> A = imread('ngc6543a.jpg');
>> image(A)
```



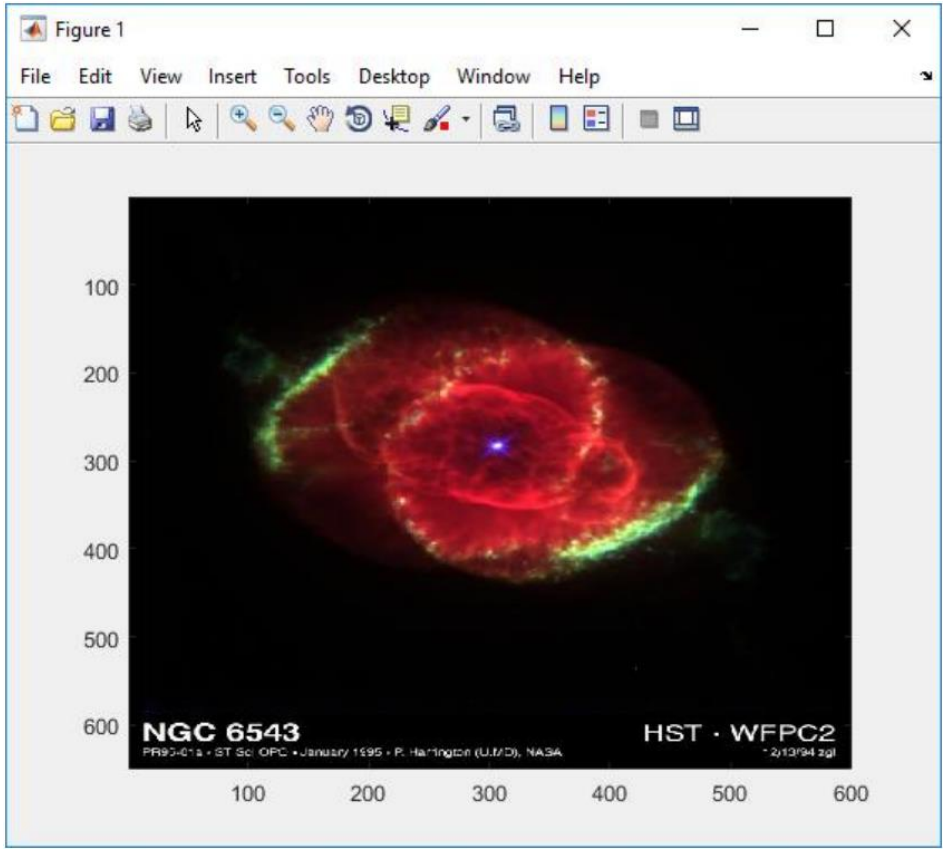


mathworks.de

Importing Images

- MATLAB can import **images** files in many **standard file formats** (TIFF, GIF, JPEG, PNG…)

```
>> ImgFileInfo = imfinfo('ngc6543a.jpg');
>> ImgFileInfo.Height
ans =
   650
>> ImgFileInfo.Format
ans =
jpg
>> ImgFileInfo.BitDepth
ans =
    24
```



mathworks.de

Importing Audio

- Import **audio files** can be imported using the audioread command:

```
>> [y,Fs] = audioread('handel.wav');
>> sound(y,Fs)
```

- **Information** about the audio file can be retrieved using the audioinfo command.

```
>> AudioFileInfo = audioinfo('handel.wav');
>> AudioFileInfo.Artist
ans =
Georg Friedrich Haendel
>> AudioFileInfo.Duration
ans =
    8.9249
```

Importing Video

- **Videos** can be imported using the **Video Reader Class.**

```
>> vidObj = VideoReader('xylophone.mp4');
```

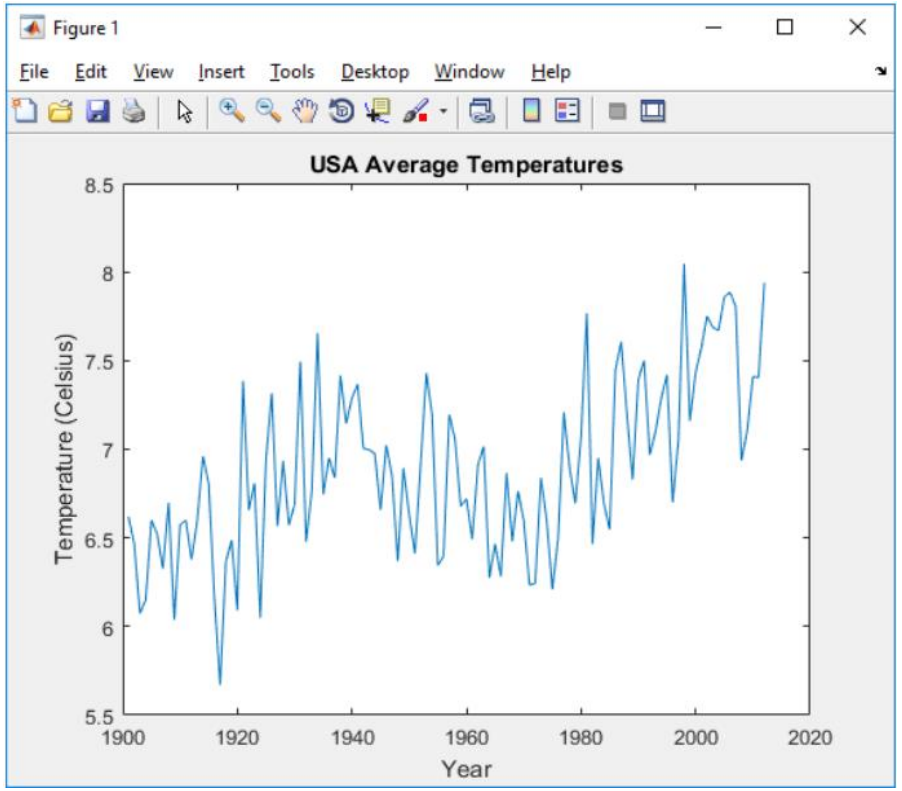
Importing Web Data

- Using the webread command, data can be read from a **web service** specified by an **url**.

```
>> api = 'http://climatedataapi.worldbank.org/climateweb/rest/v1/';
>> url = [api 'country/cru/tas/year/USA'];
>> climateData = webread(url)
```

```
climateData =
112x1 struct array with fields:
    year
    data
```

The World Bank Climate Data API returns a JSON object, which is converted to structure array.



API and data courtesy of the World Bank: Climate Data API.

Importing XML Data

- Using the xmlread command, XML documents can be read into MATLAB. The command returns a **Document Object Model** (DOM).

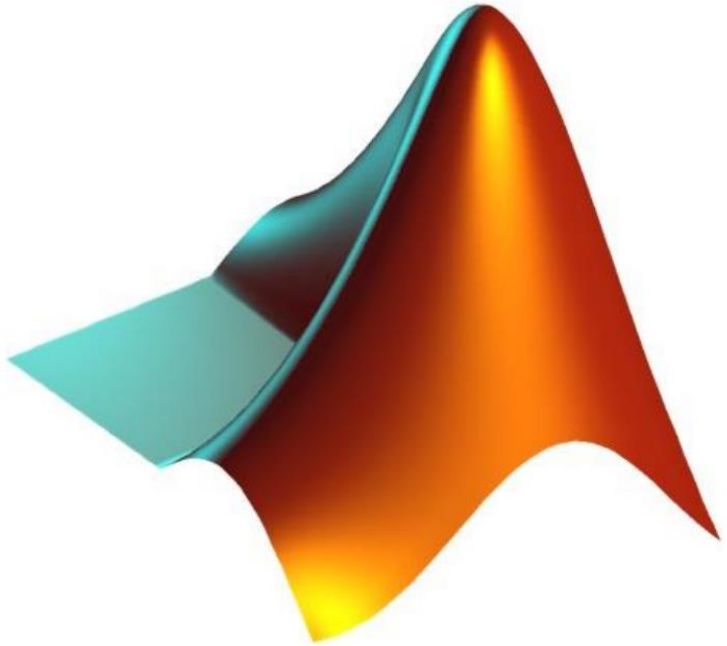
```
>> xDoc = xmlread('attendant.xml');
>> FirstNames = xDoc.getElementsByTagName('firstname');
>> FirstNames.item(0).getFirstChild.getData
```

```
ans =
Markus
```

Every item of the xml file corresponds to a node in the DOM, which can be accessed according to standards set by the World Wide Web consortium.

```
<?xml version="1.0" encoding="UTF-8"
standalone="yes"?>
<party>
    <attendant>
        <firstname>Markus</firstname>
        <lastname>Müller</lastname>
        <age>53</age>
        <attendance>true</attendance>
        <company>
            <name>Marta Müller</name>
            <name>Michael Müller</name>
            <name>Martina Müller</name>
        </company>
    </attendant>
    <attendant>
        <firstname>Peter</firstname>
```

1.2. Low Level Imports



Data Import with Low-Level I/O

- **Low-level** file I/O function
  - **Allow** most **control** over reading or writing data to a file
  - **Require** detailed **information** about the file

- Useful commands include:

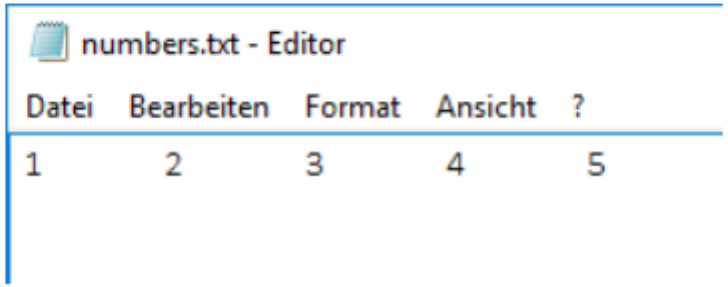
Command	Description
fscanf	Reads formatted data in a text of ASCII file (human readable)
fgetl / fgets	Reads one line at a time, where a newline character separates each line
fread	Reads stream of data at a byte of bit level

Data Import with Low-Level I/O

Command	Description
fscanf	Reads formatted data in a text of ASCII file (human readable)
fread	Reads stream of data at a byte of bit level

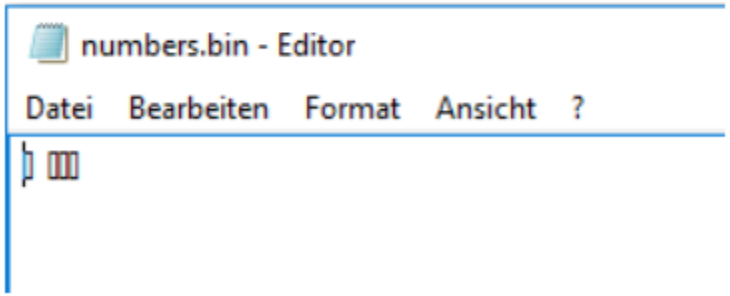
fscanf

```
>> fid = fopen('numbers.txt');
>> numbers = fscanf(fid,'%i\t')'
numbers =
     1     2     3     4     5
>> fclose(fid);
```

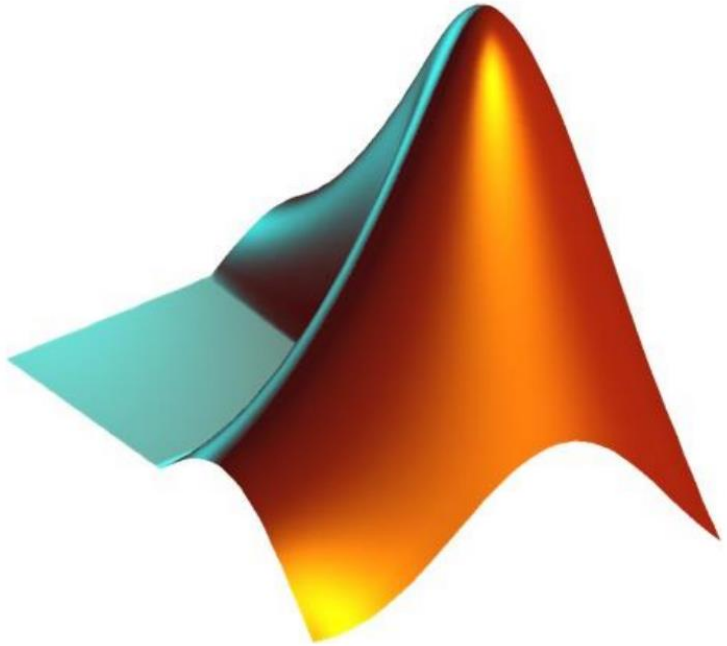


fread

```
>> fid = fopen('numbers.bin');
>> numbers = fread(fid)'
numbers =
     1     2     3     4     5
>> fclose('all');
```



1.3. Large Files and Big Data



Datastores

Several tools are provided by MATLAB for handling **large data sets** that **do not fit** into available memory or take **long to process**:

- Datastore: a repository for data that has the same structure and formatting. Useful when:
  - Each file in the collection is **too large to fit into memory**
  - Files in the store have **arbitrary names**
  - **Application example**: chunk through a big file to **find maximum delay** (manually):

```
>> ds = datastore('airlinesmall.csv');
>> ds.SelectedVariableNames = {'DepDelay'}; ds.TreatAsMissing = 'NA';
>> ds.ReadSize = 5000;
>> maxDelay = 0; reset(ds);
>> while hasdata(ds)
    T = read(ds);
    maxDelay = max(maxDelay, max(T.DepDelay));
   end
>> fprintf('Maximum Delay: %0.3f\n', maxDelay)
Maximum Delay: 1438.000
```

Mapreduce

- mapreduce is a programming technique for analyzing large data (using a datastore) in two steps:
  - The map function receives chunks of data and creates **intermediate results**
  - The reducer reads the intermediate results and produces a **single final result**
  - **Application example:** chunk through a big file to find **mean delay** (using mapreduce):

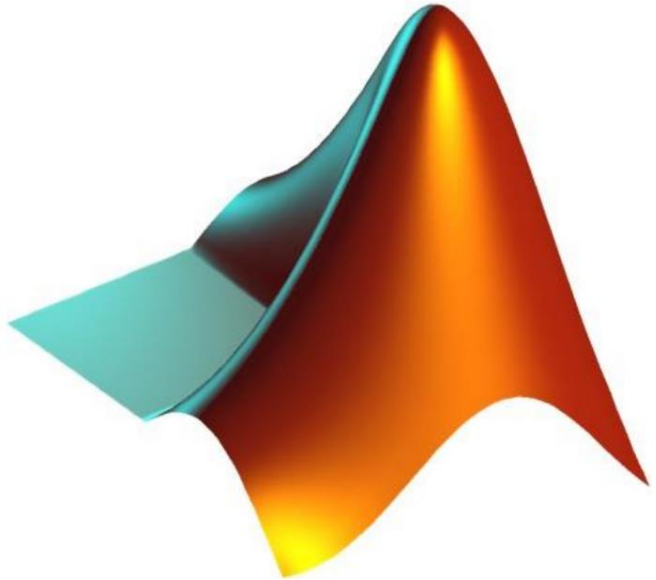
```
>> ds = datastore('airlinesmall.csv');
>> ds.ReadSize = 5000;
>> ds.TreatAsMissing = 'NA';
>> ds.SelectedVariableNames = 'ArrDelay';
>> meanDelay = mapreduce(ds,...
   @meanArrDelayMapper,...
   @meanArrDelayReducer);
>> T = readall(meanDelay);
>> fprintf('The mean arrival delay is
%0.3f\n',T.Value{:})
The mean arrival delay is 7.120
```

meanArrDelayMapper returns the count and sum of arrival delay data in each chunk

meanArrDelayReducer processes the results by summing counts and sums

2. Memory Management

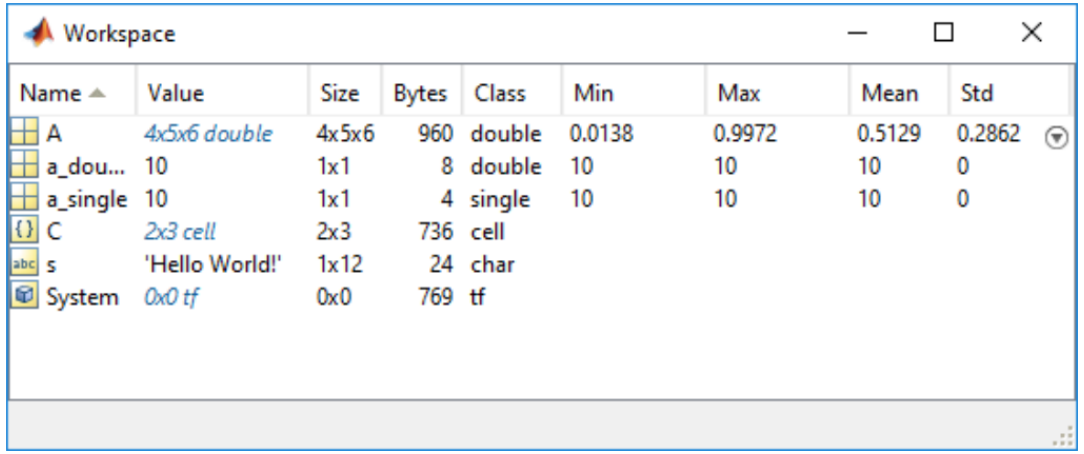
2.1. MATLAB Workspace



MATLAB Workspace

The MATLAB Workspace:

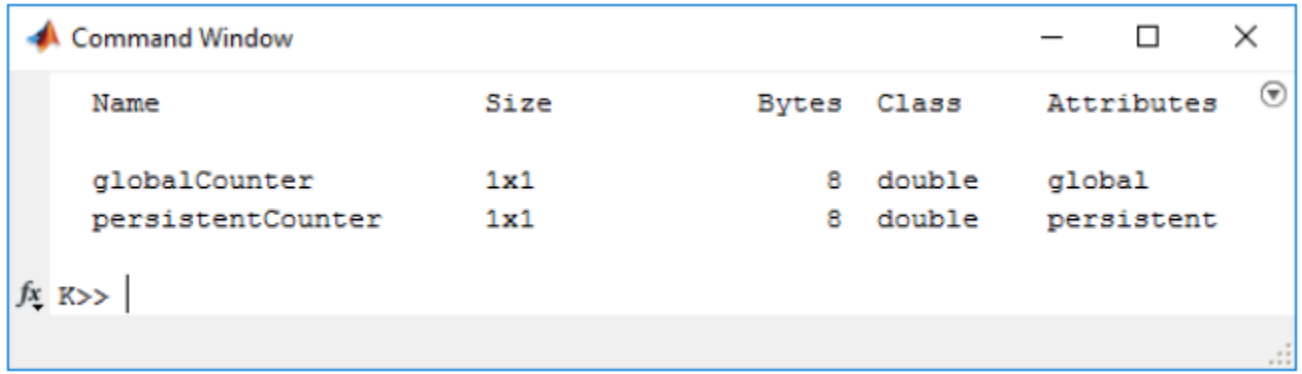
- Consists of the variables **created** and **stored** to memory during a MATLAB session.
- Variables can be **added**:
  - using **functions**
  - running MATLAB **code**
  - **loading** saved workspaces
- Variables in the workspace are **displayed** by the Workspace Browser along with relevant **information**:
  - Name
  - Value
  - Dimensions (Size)
  - Memory (Bytes)
  - Class ...



Base Workspace

The Base Workspace:

- Contains variables **created**:
  - at the **command line**
  - in scripts that are **run from the command line** or **editor**
- Variables **exist** until:
  - they are **cleared**
  - the MATLAB **session is ended** (Matlab is closed)

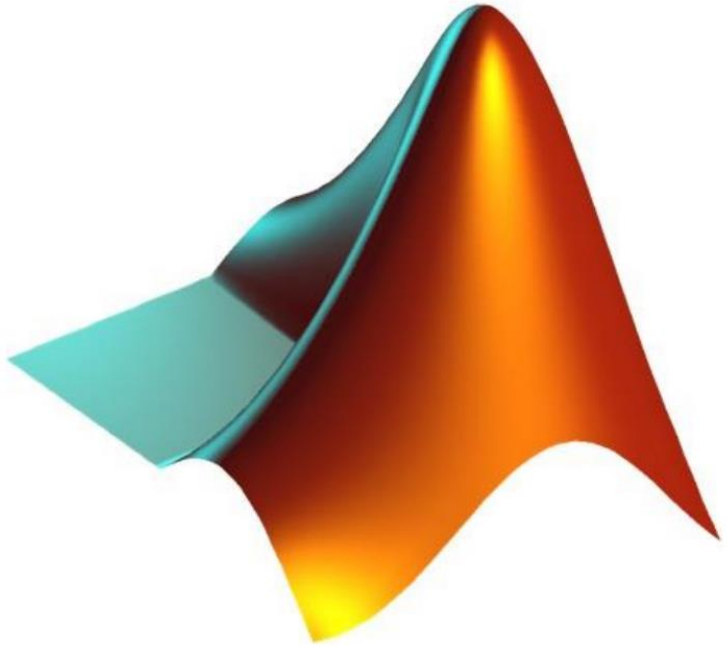


Function Workspace

The Function Workspace:

- Contains variables **created** in its function.
- Each function has its own **individual** function workspace to ensure data **integrity**.
- Variables typically exist until function is **terminated**.
- Exceptions:
  - **Persistent** variables
  - **Global** variables
- Recall:
  - **Local** functions have their **own** workspace
  - **Nested** function can **access** and **modify** variables in the function workspace of the **parent** function

2.2. Global and Persistent Variables



Global and Persistent Variables

- Persistent Variables:
  - **Local** to the function in which they are
  - Declared by **keyword** persistent
  - Declared value is **retained between function calls**
- Global Variables:
  - All functions that **declare** a variable global share a **single copy**
  - Declared by **keyword** global

Illustration of Global & Persistent Variables Using a Counter

```
function [nextInc_global, nextInc_persistent] = Count()
% declare global and persistent variables
global globalCounter
persistent persistentCounter
% check if persistent variable has been initialized
if isempty(persistentCounter)
    persistentCounter = 0;
else
    persistentCounter = persistentCounter + 1;
end
% call global counter function
incGlobalCounter;
% return results
nextInc_global      = globalCounter;
nextInc_persistent  = persistentCounter;
end

function incGlobalCounter()
% declare global variable
global globalCounter
% check if global variable has been initialized
if isempty(globalCounter)
    globalCounter = 0;
else
    globalCounter = globalCounter + 1;
end
end
```

Declaration of both the persistent and the global variable, so they are **available** in the function.

Incrementing after checking if the variable is **empty**. For a **persistent** variable, this is either because the function was **never called before**, or because the **function's workspace** was **cleared**.

Incrementing after checking if the variable is **empty**. For a **global** variable, this is either because the function was **never called before**, or because it was **cleared** from the **base workspace**.

Writing output.

Illustration of Global & Persistent Variables Using a Counter

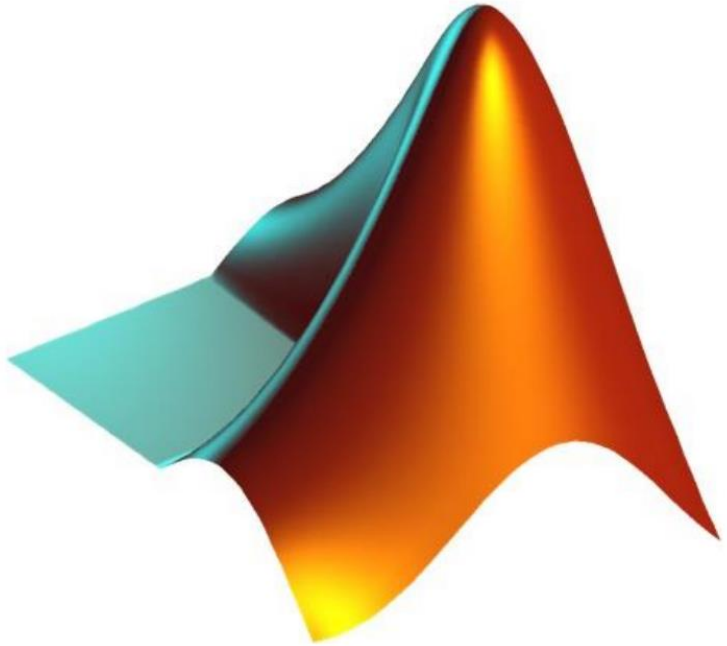
```
function [nextInc_global, nextInc_persistent] = Count()
% declare global and persistent variables
global globalCounter
persistent persistentCounter
% check if persistent variable has been initialized
if isempty(persistentCounter)
    persistentCounter = 0;
else
    persistentCounter = persistentCounter + 1;
end
% call global counter function
incGlobalCounter;
% return results
nextInc_global      = globalCounter;
nextInc_persistent  = persistentCounter;
end

function incGlobalCounter()
% declare global variable
global globalCounter
% check if global variable has been initialized
if isempty(globalCounter)
    globalCounter = 0;
else
    globalCounter = globalCounter + 1;
end
end
```

```
>> [g,p] = Count;
>> [g,p] = Count;
>> clear Count
>> [g,p] = Count;
>> global globalCounter
>> globalCounter = 6;
>> [g,p] = Count;[g,p]
```

globalCounter	persistentCounter
0	0
1	1
1	[]
2	0
2	0
6	0
7	1

2.3. Memory Allocation



Allocating Memory

- For **most cases**, MATLAB's internal operations **automatically** allocate memory in an **efficient way**.
- **Example:** assignment of numeric array,
  - MATLAB allocates **two memory blocks**:
    1. **Contiguous** virtual block containing **array data**
    2. **Separate small block** called **header** containing information about the array data such as Class, Dimensions, …

Allocating Memory

- If a **new element** is added to the array, MATLAB **expands** the existing array in memory, keeping storage contiguous. This usually requires finding a **new block of memory**, hence the need for **pre-allocation**.

```
tic;
x = 0;
for k = 2:1000000
    x(k) = x(k-1) + 2;
end
toc
```

```
Elapsed time is 0.070668 seconds.
```

Memory is allocated only once, hence speed increase.

```
tic;
x = zeros(1,1000000);
for k = 2:1000000
    x(k) = x(k-1) + 2;
end
toc
```

```
Elapsed time is 0.017849 seconds.
```

Copying Memory

MATLAB **only** copies memory when it is **needed** (lazy copy implementation).

- When a variable is **copied** to **another variable**, MATLAB makes a copy of the **array reference**, not of the array itself:

```
>> clearvars
>> A = magic(20000);
>> B=A;
```

Memory used (Gigabyte)
1.0575
4.2516
4.2516

- When **reducing** B's size by half, MATLAB must **allocate new memory** to store the changed data:

```
>> B(10001:20000,:) = [];
```

5.8516

Copying Memory

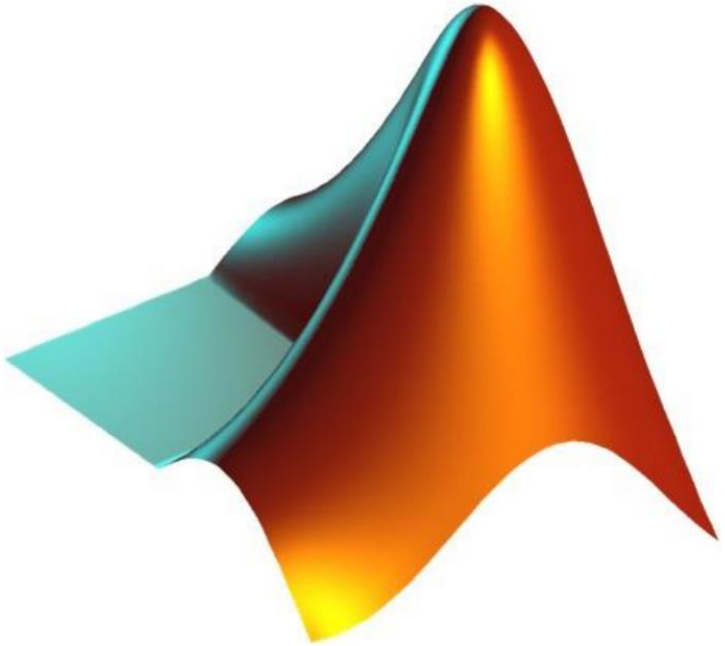
- Structure members are treated as a **separate arrays** in MATLAB:

```
>> S.A = A; S.B = B;
>> S.A(1) = 0;
>> S.B(1) = 0;
```

Memory used (Gigabyte)
5.8590
9.0635
10.655

- Similarly, function arguments are **passed as a reference** unless they are **changed within the function**.

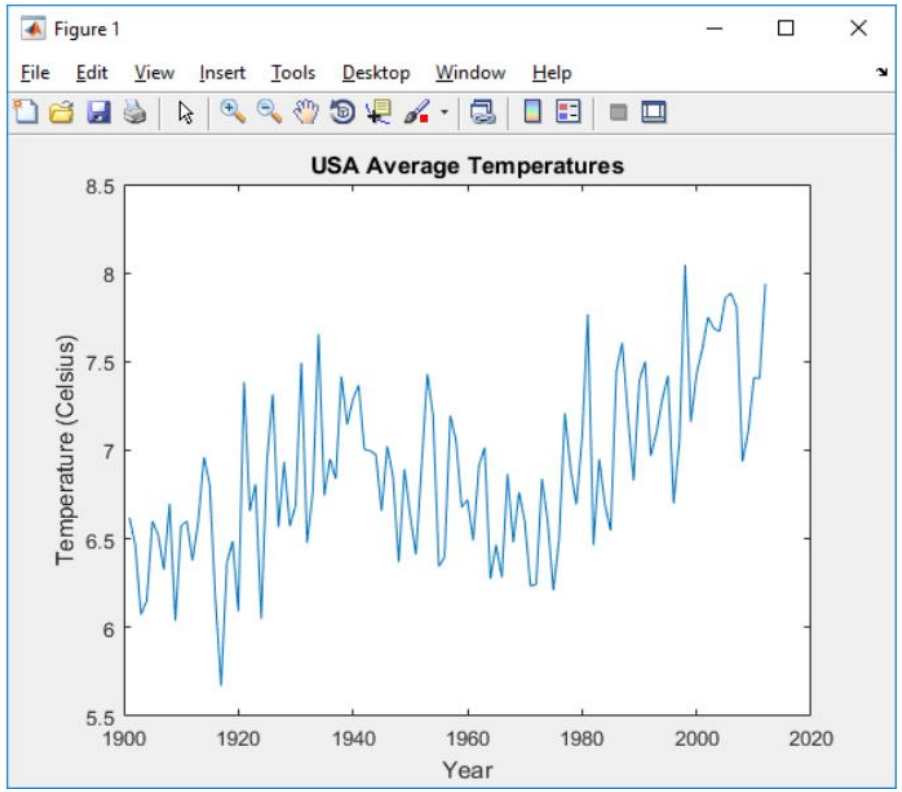
3. Graphics



Plotting Basics

- MATLAB offers a **large variety of visualizations of data**. The **simplest** way to visualize data is by using the plot command:

```
plot(climateData.year, climateData.data);
title('USA Average Temperatures');
xlabel('Year');
ylabel('Temperature (Celsius)');
```



- The plot command automatically creates a figure object. The figure command can be used to create a new figure:

```
h = figure('Name', 'Sinusoids'); whos h
```

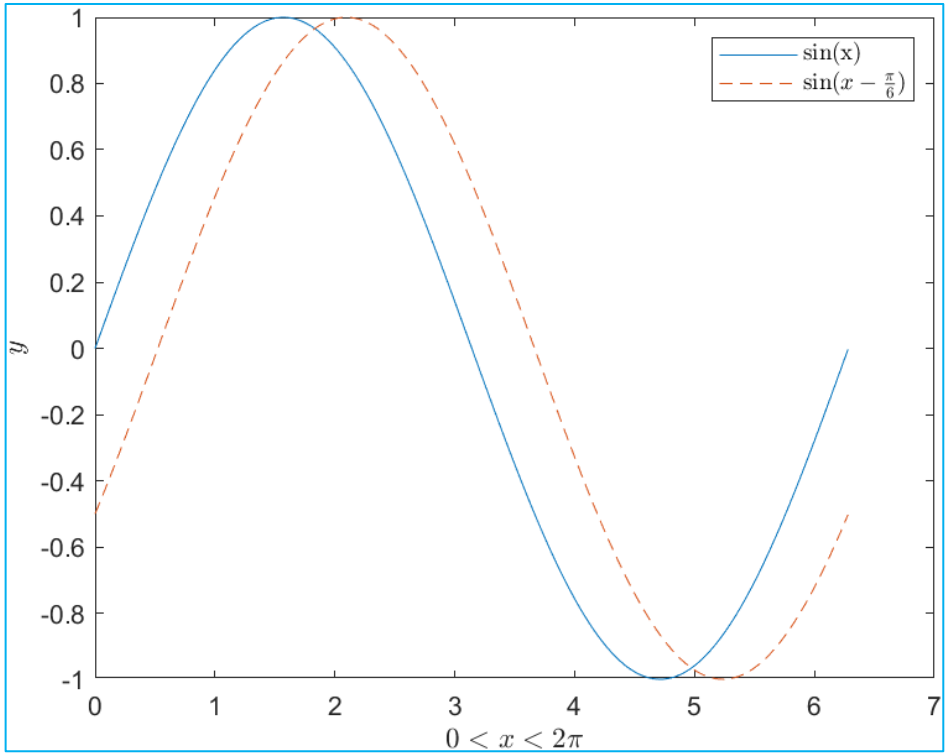
Name	Size	Bytes	Class
h	1x1	112	matlab.ui.Figure

Plotting Basics

- **Multiple graphs** can be plotted into the **same figure** by passing multiple x,y pairs to the plot function.

- **Several annotations** can be added to graphics such as title, axis labels and legends.

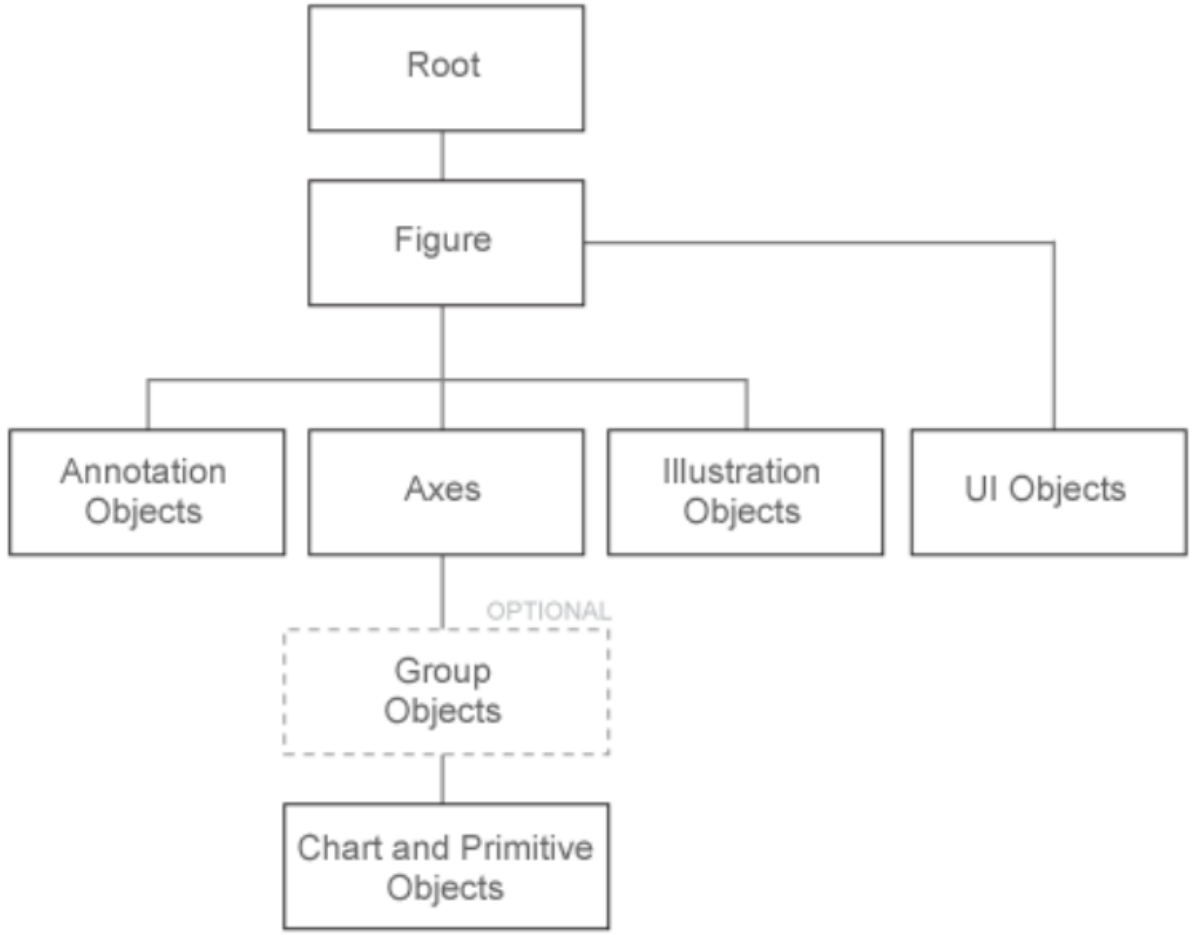
```
phi = 0:0.01:2*pi;
p = plot(phi, sin(phi), phi, sin(phi-pi/6),'--'); whos p
legend('sin(x)','$\sin(x - \frac{\pi}{6})$', ...
        'interpreter','latex');
xlabel('$0 < x < 2\pi$','interpreter','latex');
ylabel('$y$','interpreter','latex');
```



Name	Size	Bytes	Class
p	2x1	120	matlab.graphics.chart.primitive.Line

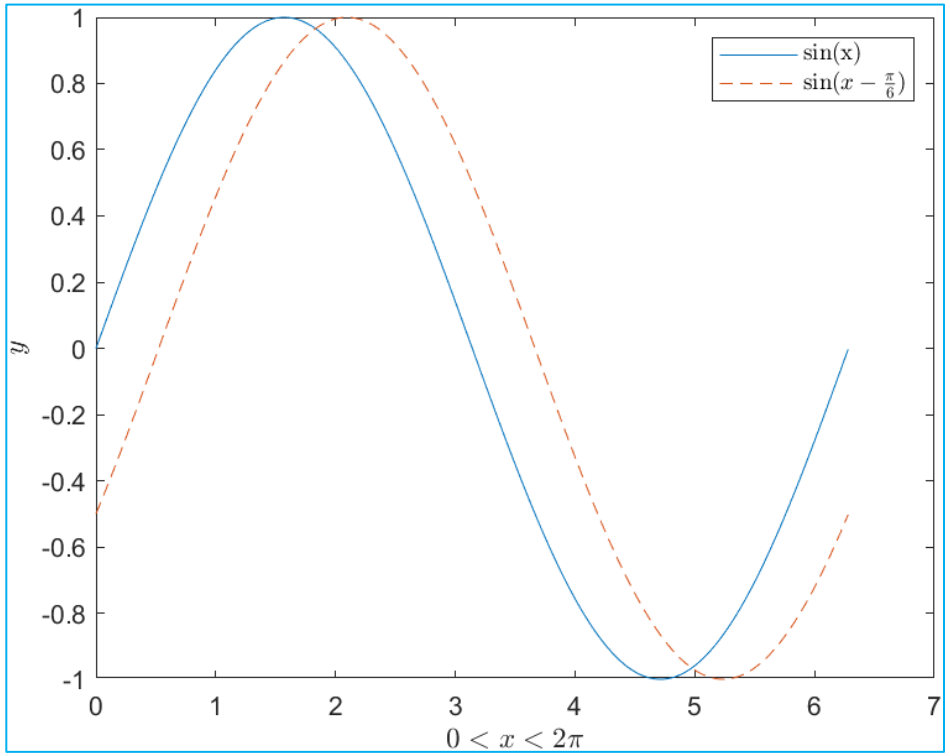
Graphics Objects

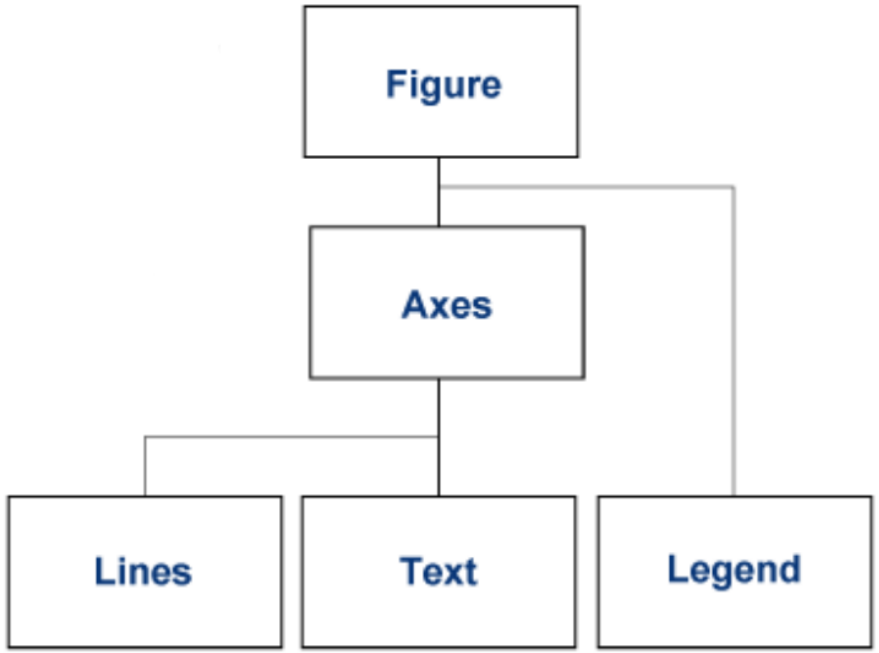
- Graphics objects are **visual components** to display data.
- MATLAB **automatically creates all objects** necessary and sets **appropriate values** to all properties
- Recall: if no figures exist, a new one is **automatically** created when using the plot command).



Graphics Objects

- Observe the hierarchy for the previous figure:





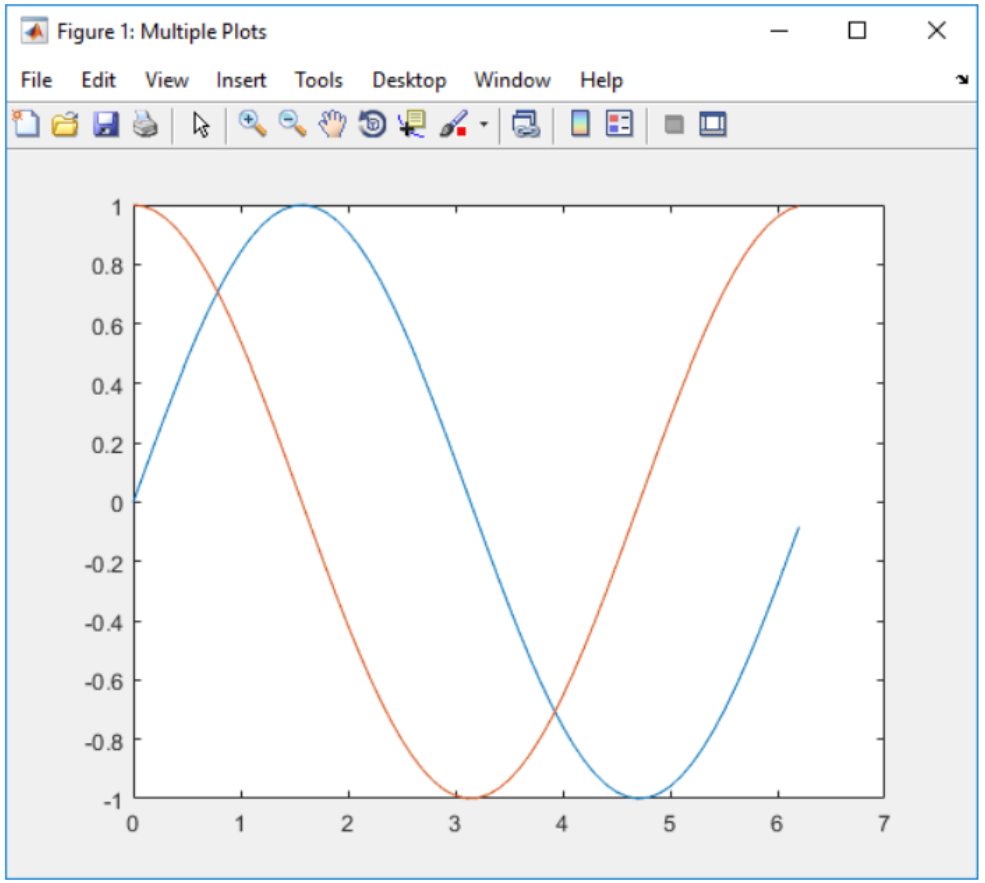
Multiple Plots

- hold command is used:
  - when calling the plot function **twice for the same axes**, the first plot is **replaced**
  - to **retain** plots

```
phi = 0:0.1:2*pi;
h = figure('Name','Multiple Plots');

ax = axes('Parent',h);
% retain existing plots
hold(ax,'on');

p = gobjects(2);
p(1) = plot(ax, phi, sin(phi));
p(2) = plot(ax, phi, cos(phi));
```



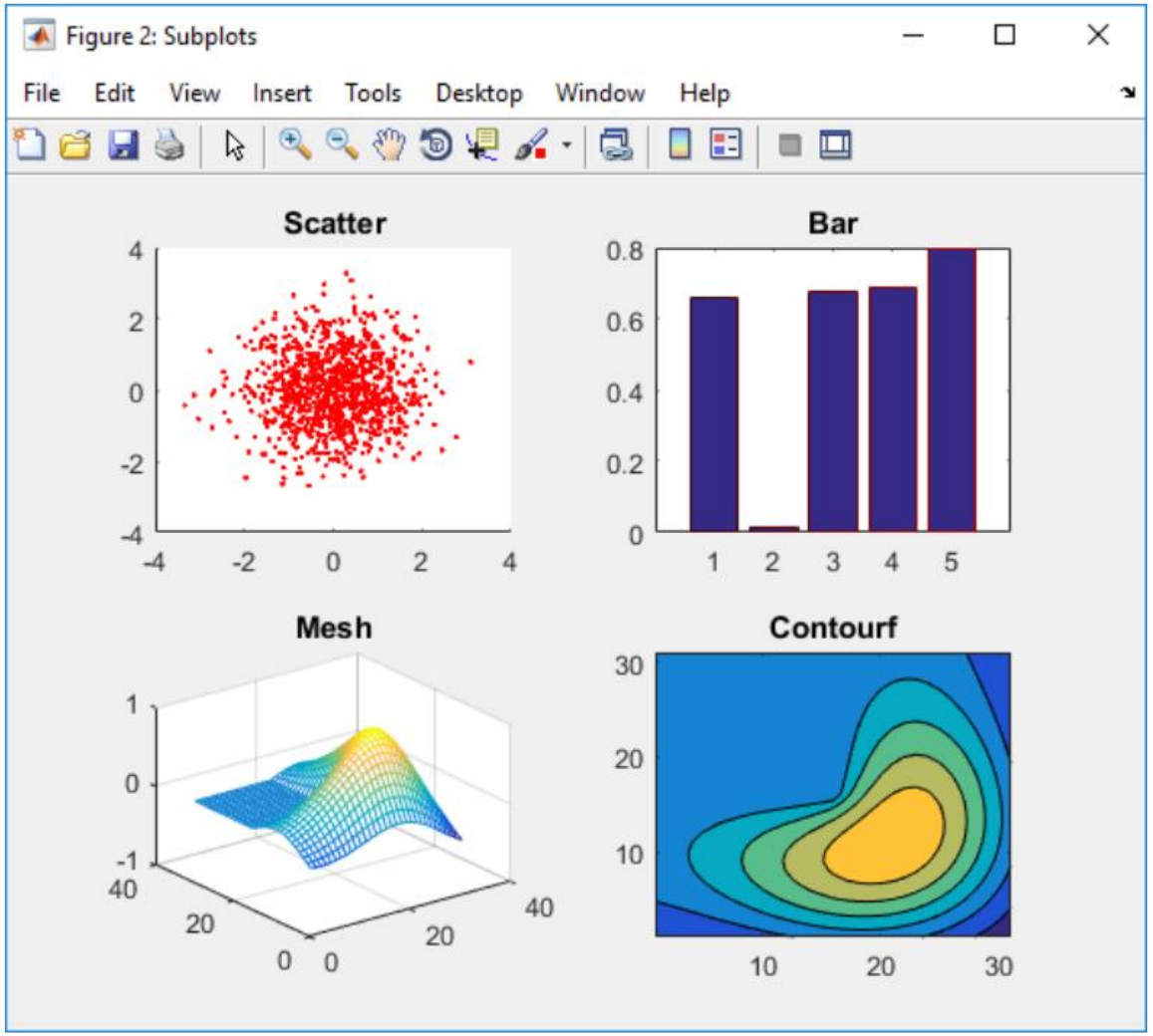
Multiple Plots

- subplot:
  - The subplot command can be used to create **multiple plots within one figure object**.
  - subplot(m,n,p) **divides the figure into an m-by-n grid** and creates an axes at position p (subplots are numbered by row).
  - On more recent versions, MATLAB **advises** using tiledlayout.

Multiple Plots

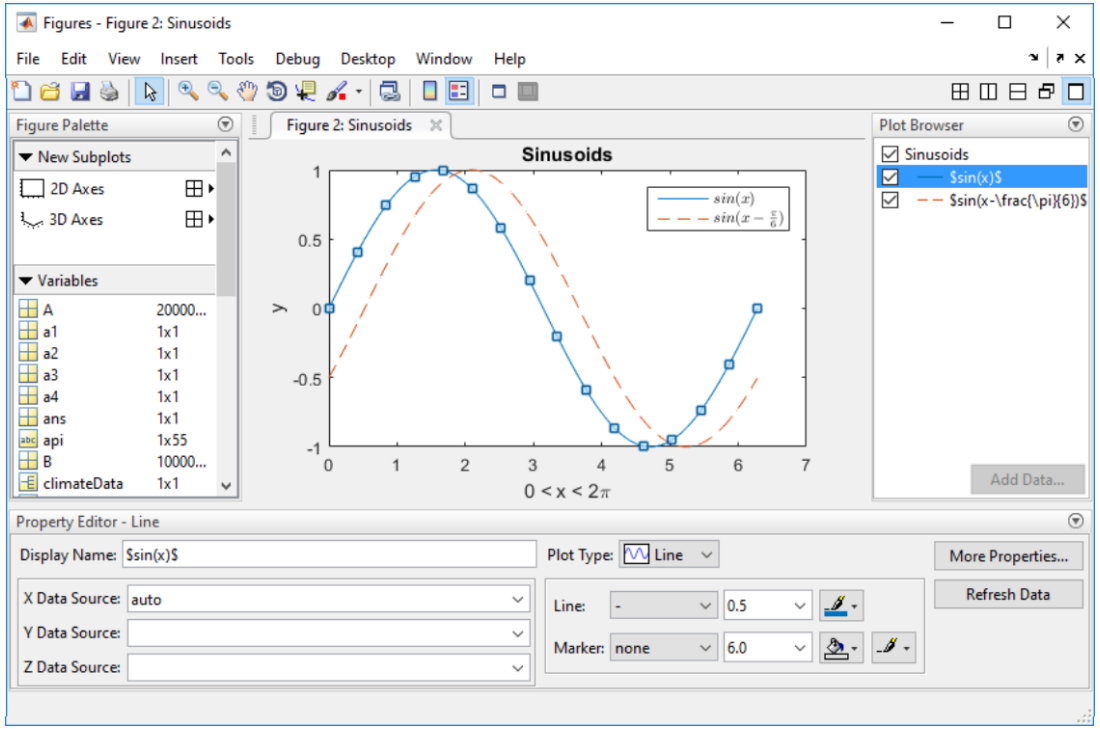
- subplot:

```
h = figure('Name','Subplots');
p = gobjects(2);
for piter = 1:numel(p)
    p(piter) = subplot(2,2,piter);
end
% scatter plot
scatter(p(1),randn(1000,1), randn(1000,1),'r.');
title(p(1),'Scatter');
% bar plot
bar(p(2), 1:5,rand(1,5)); title(p(2),'Bar');
% mesh plot
mesh(p(3),membrane); title(p(3),'Mesh');
% filled contour plot
contourf(p(4), membrane); title(p(4),'Contourf');
```



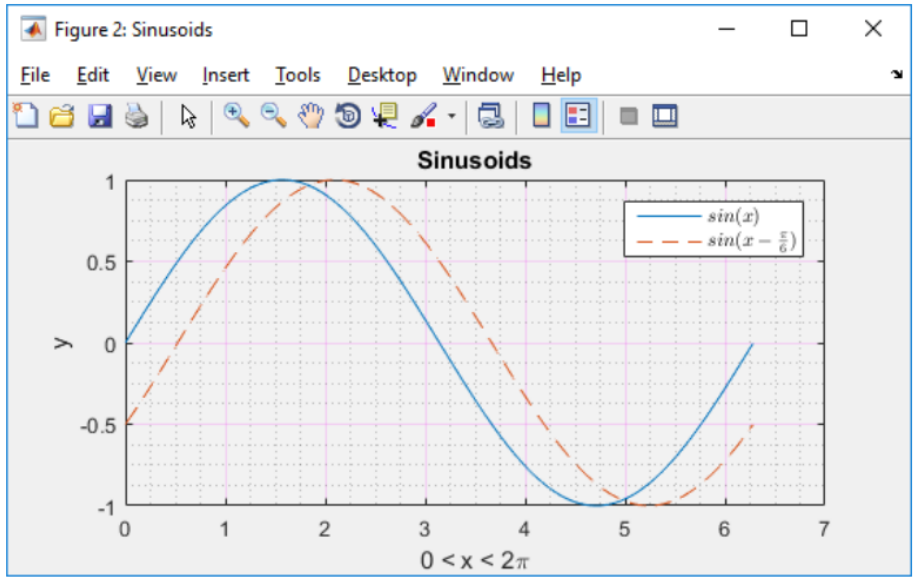
Graphics Object Properties

- **Behavior** and **appearance** of a graphics object can be **controlled by setting its properties.**
  - **Interactively:** using plot tools
  - **Programmatically:** access and modify properties using the dot notation




```
ax = gca;
ax.Color = 0.95*[1,1,1]
grid(ax,'on'); grid(ax,'minor');
ax.GridColor = 'magenta';
```

Current axes



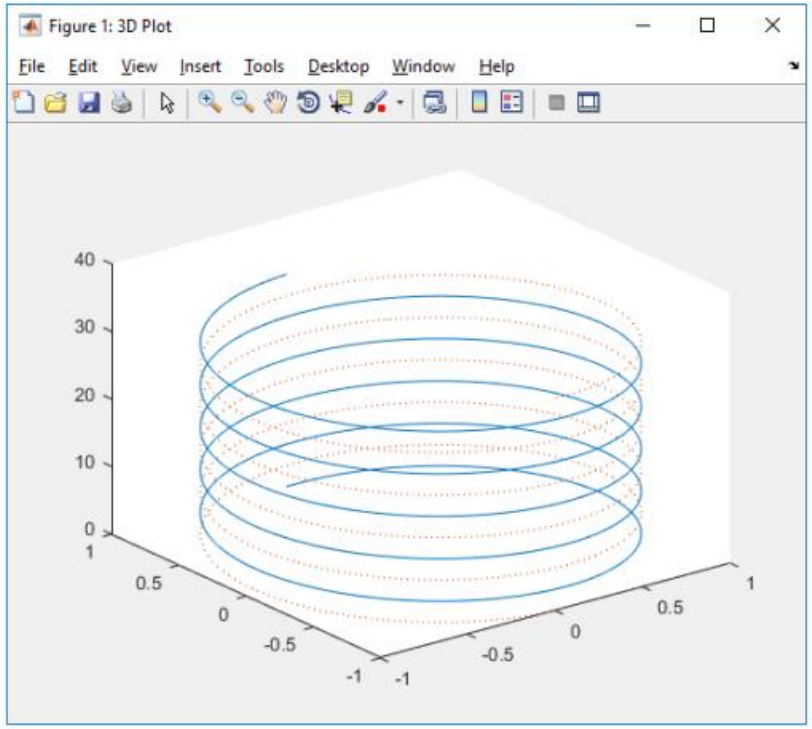
Plot Types

- Besides the 2D line plot, various functions **exist** to plot data in a suitable way.
- Plots can be selected **interactively** from the PLOTS **ribbon** or by **typing** the respective command.

Plot Type Examples

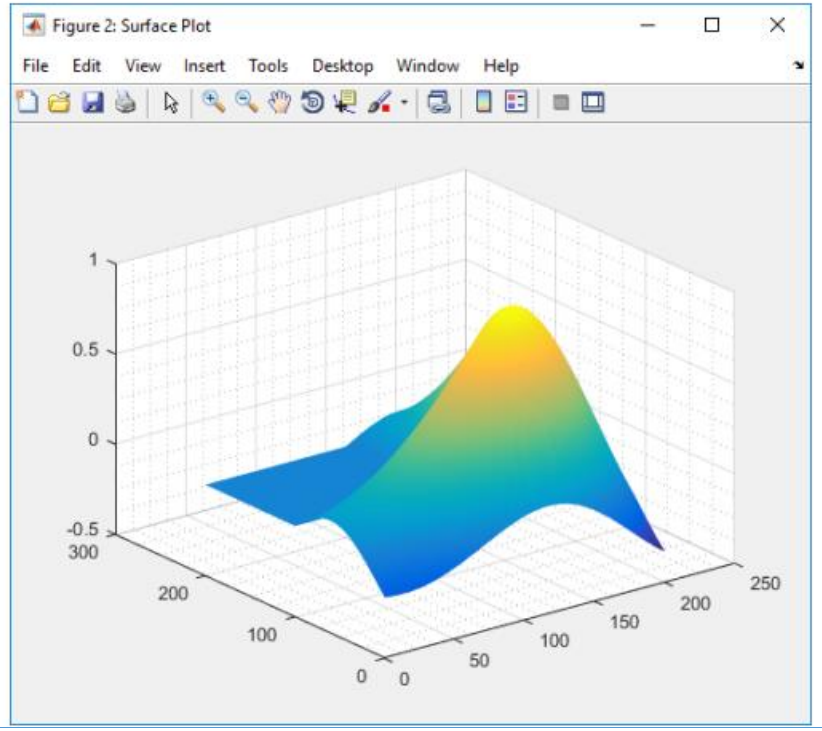
- plot3: creates a **line plot in three dimensions** similarly to the plot function.

```
>> t = 0:pi/50:10*pi;
>> st = sin(t);
>> ct = cos(t);
>> h = figure
>> p = plot3(st, ct, t,-st,-ct, t,':')
```



- surf: creates a 3-D shaded **surface plot** from data consisting of (x,y,z) triplets

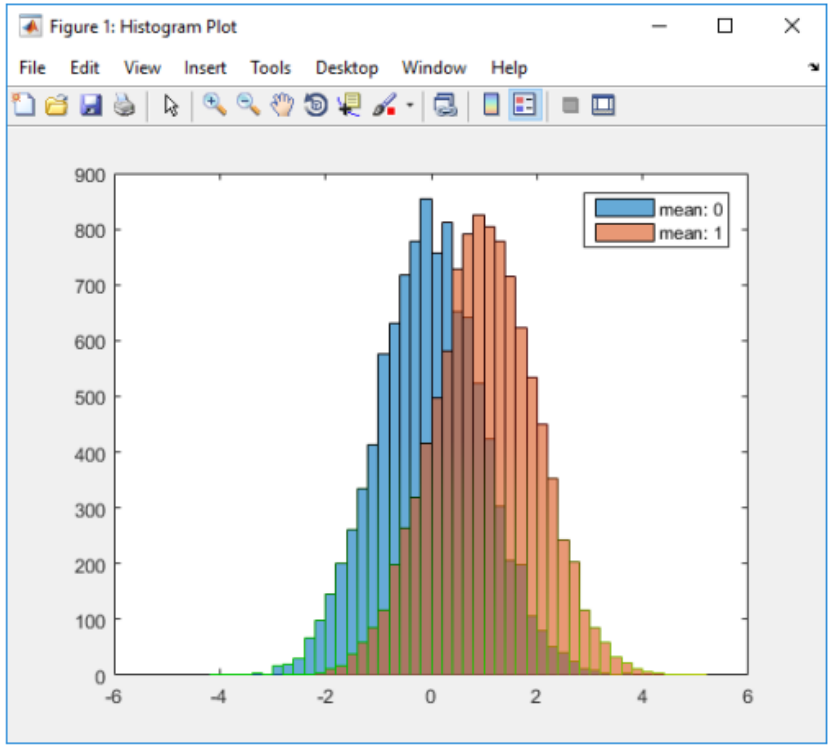
```
>> h = figure;
>> h.Name = 'Surface Plot';
>> p = surf(membrane(1,100));
>> grid('minor')
>> p.EdgeColor = 'none';
```



Plot Type Examples

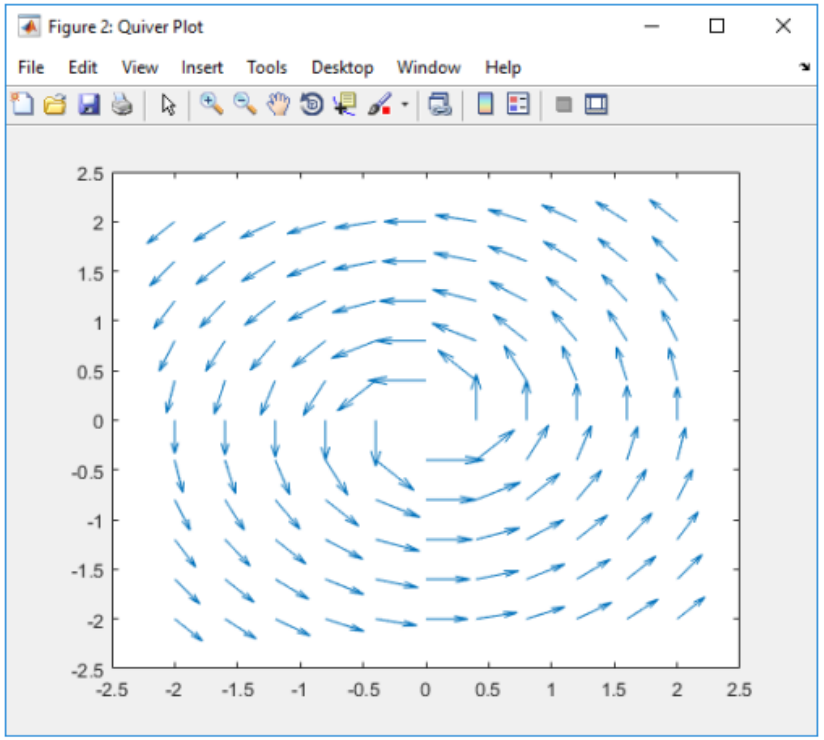
- histogram: type of bar plot for numeric data that group the data into bins.

```
>> x1 = randn(10000,1); x2 = randn(10000,1) + 1;
>> h = figure('Name','Histogram Plot');
>> p = gobjects(2,1);
>> p(1) = histogram(x1); hold on;
>> p(2) = histogram(x2); legend('mean: 0','mean: 1');
```



- quiver: (aka velocity plot) displays a vector field as arrows with components u and v.

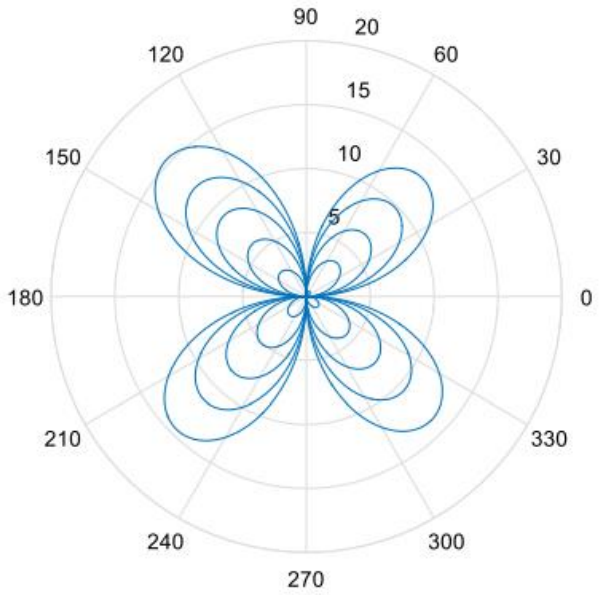
```
>> [x,y] = meshgrid(-2:.4:2,-2:.4:2);
>> V = 1./(x.^2 + y.^2).^.1; phi = atan2(y,x);
>> V(isinf(V)) = 0;
>> u = -V .* sin(phi); v =  V .* cos(phi);
>> figure('Name','Quiver Plot'); quiver(x,y,u,v);
```



Plot Type Examples

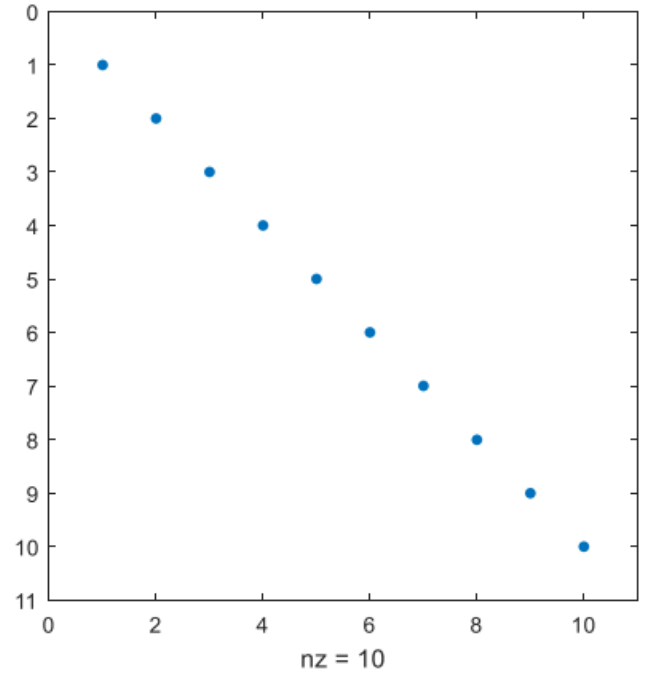
- polar: takes data in polar coordinates and plots them in the Cartesian plane.

```
>> h = figure('Name','Polar Plot');
>> phi = 0:0.01:10*pi;
>> r   = phi.*sin(phi).*cos(phi);
>> p = polar(phi, r);
```



- spy: visualizes the sparsity pattern of a sparse matrix.

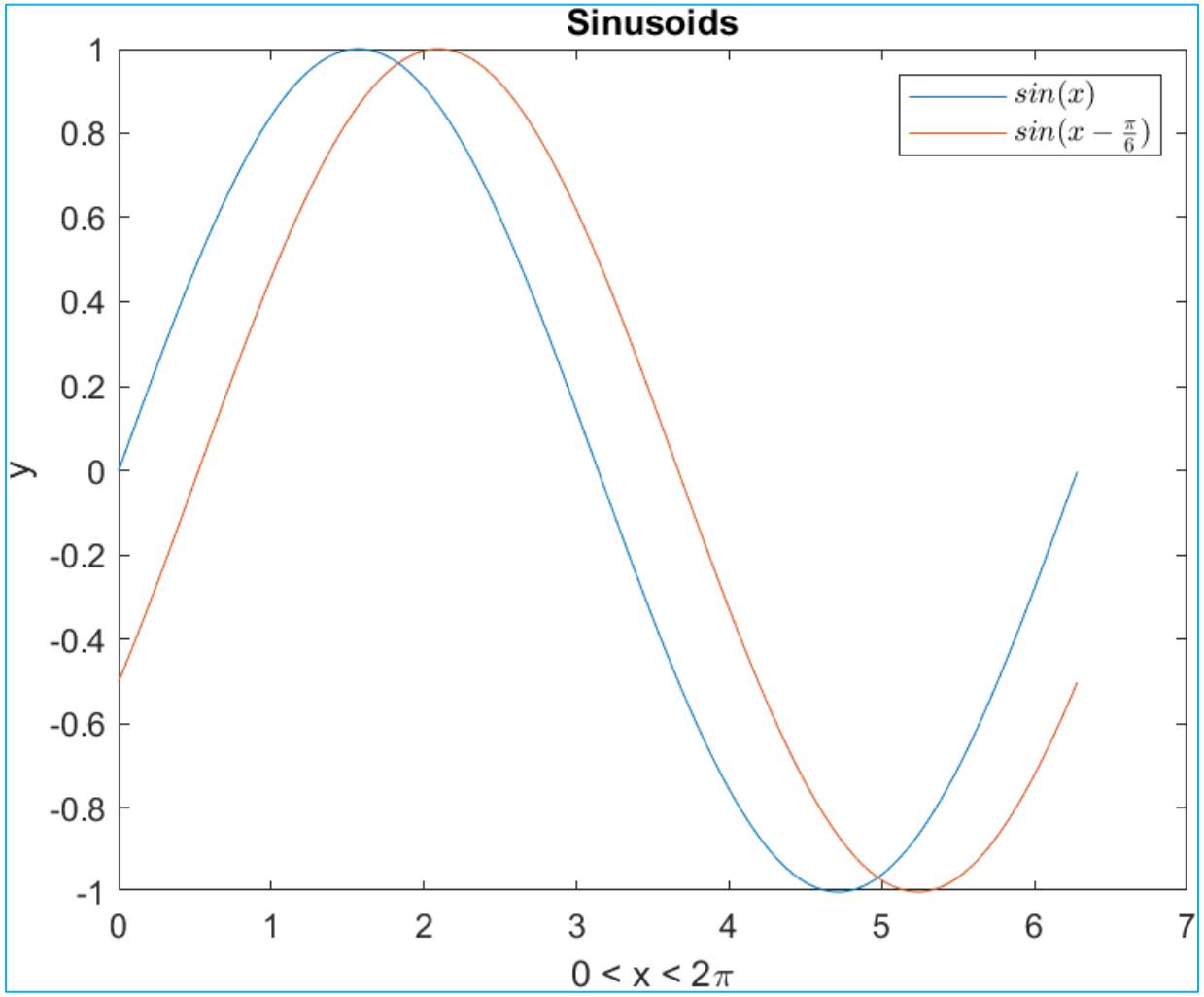
```
>> A = eye(10);
>> p = spy(A,'.');
```



Title, Labels & Legends

- To create **meaningful graphics**, title, labels and legends can be added to a plot.
- Text can be **interpreted** using TeX or LaTeX.

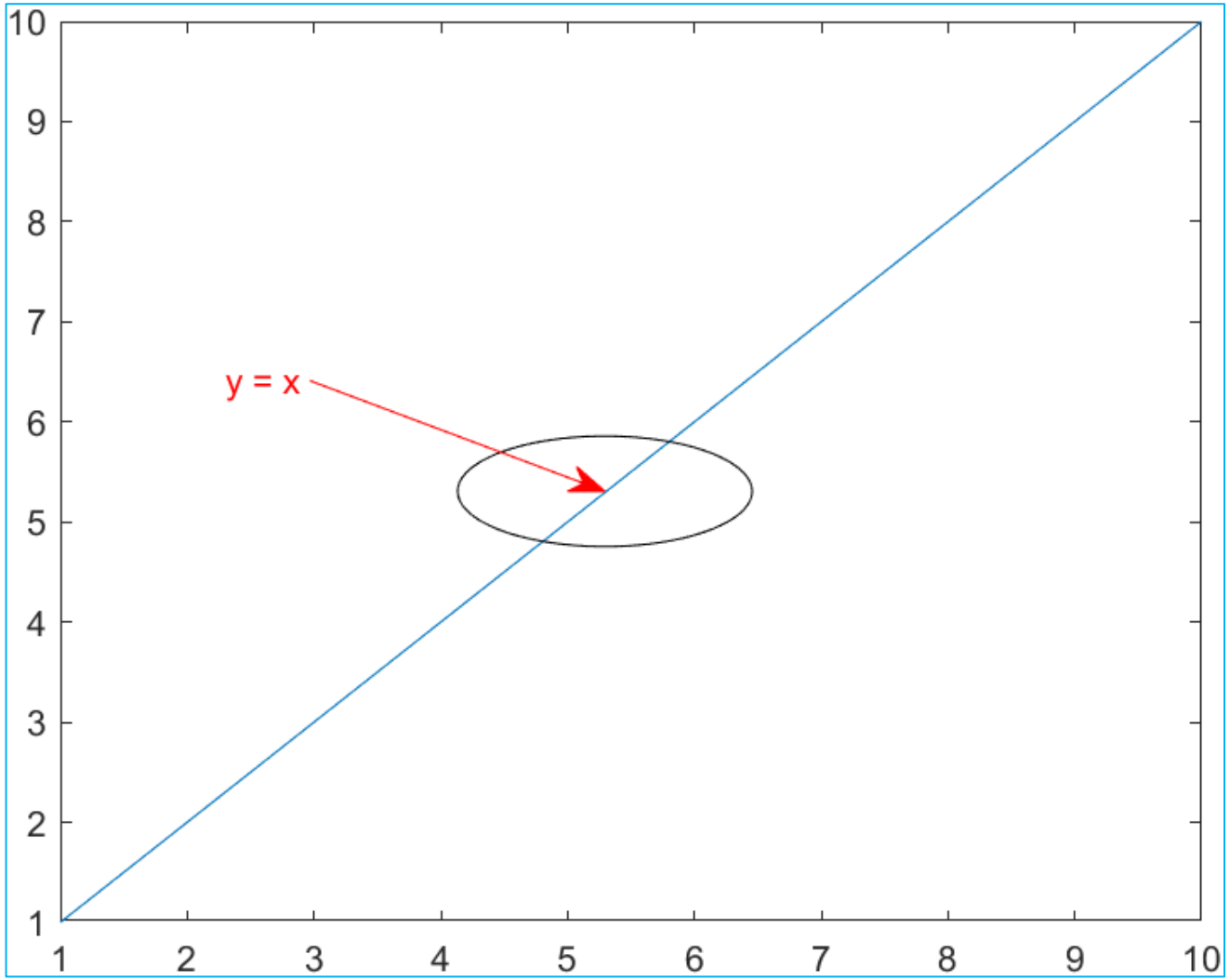
```
phi = 0:0.01:2*pi; plot(phi, sin(phi), phi, ...
            sin(phi-pi/6))
xlabel('0 < x < 2\pi'); ylabel('y'); title('Sinusoids');
l = legend('$sin(x)$','$sin(x-\frac{\pi}{6})$');
l.Interpreter = 'latex';
```



Annotations

- Annotations such as **text** and **shapes** can be added to **emphasize important details.**
- Use the annotation command.

```
h = figure('Name','Plot with Annotation');
plot(1:10)
x = [0.3 0.5];
y = [0.6 0.5];
a(1) = annotation('ellipse',[0.4 .45 .2 .1]);
a(2) = annotation('textarrow',x,y,'String', ...
            'y = x ');
a(2).Color = 'red';
```

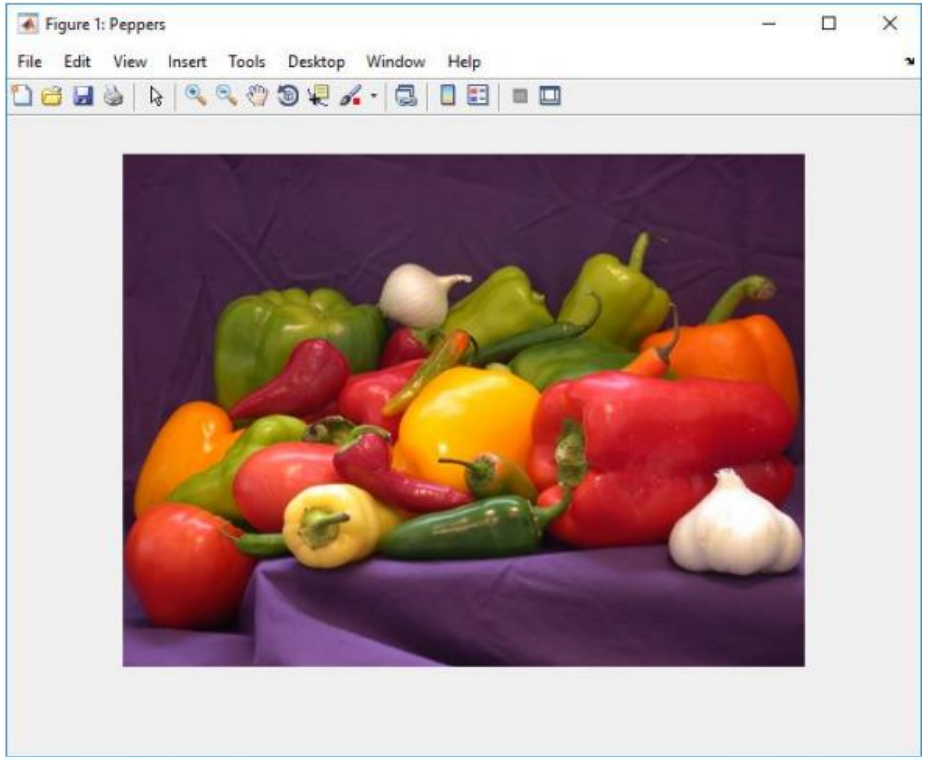


Images

- Images can be plotted using the imshow command.

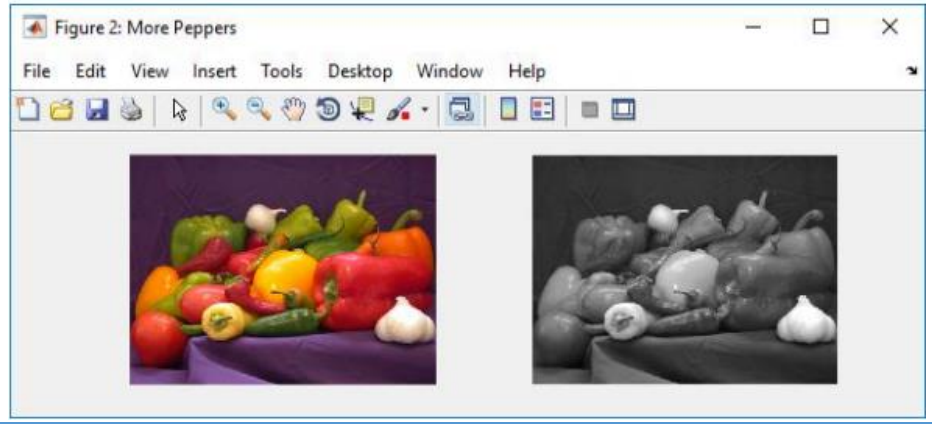
```
>> RGB = imread('peppers.png');

>> h = figure('Name','Peppers');
>> image = imshow(RGB);
```

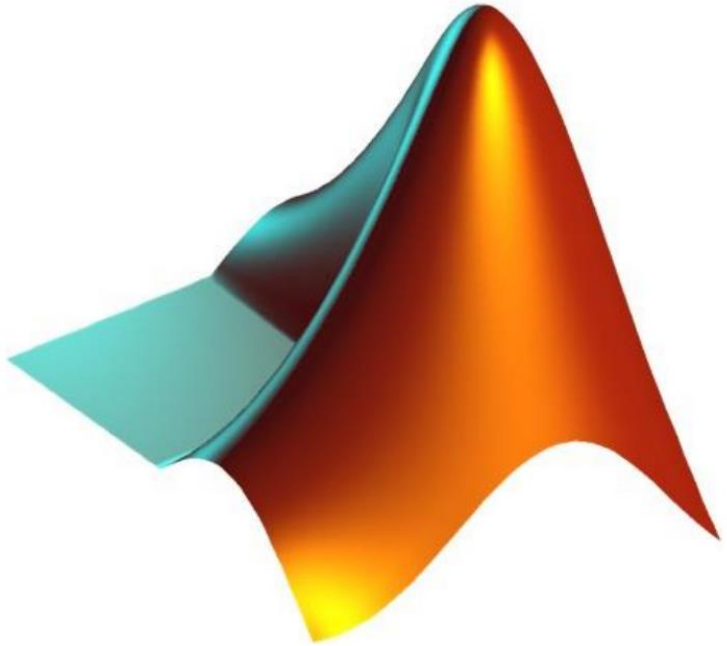


- The subimage command can be used to plot multiple images into a single figure object

```
>> h = figure('Name','More Peppers');
>> p = gobjects(2,1);
>> for piter = 1:numel(p)
       p(piter) = subplot(1,2,piter);
   end
>> axes(p(1)); subimage(RGB); axis off
>> axes(p(2)); subimage(rgb2gray(RGB)); axis off
```

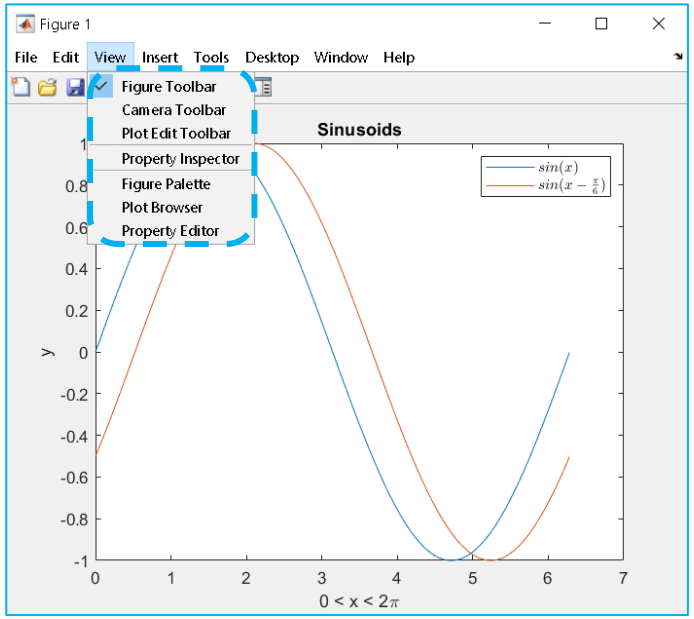


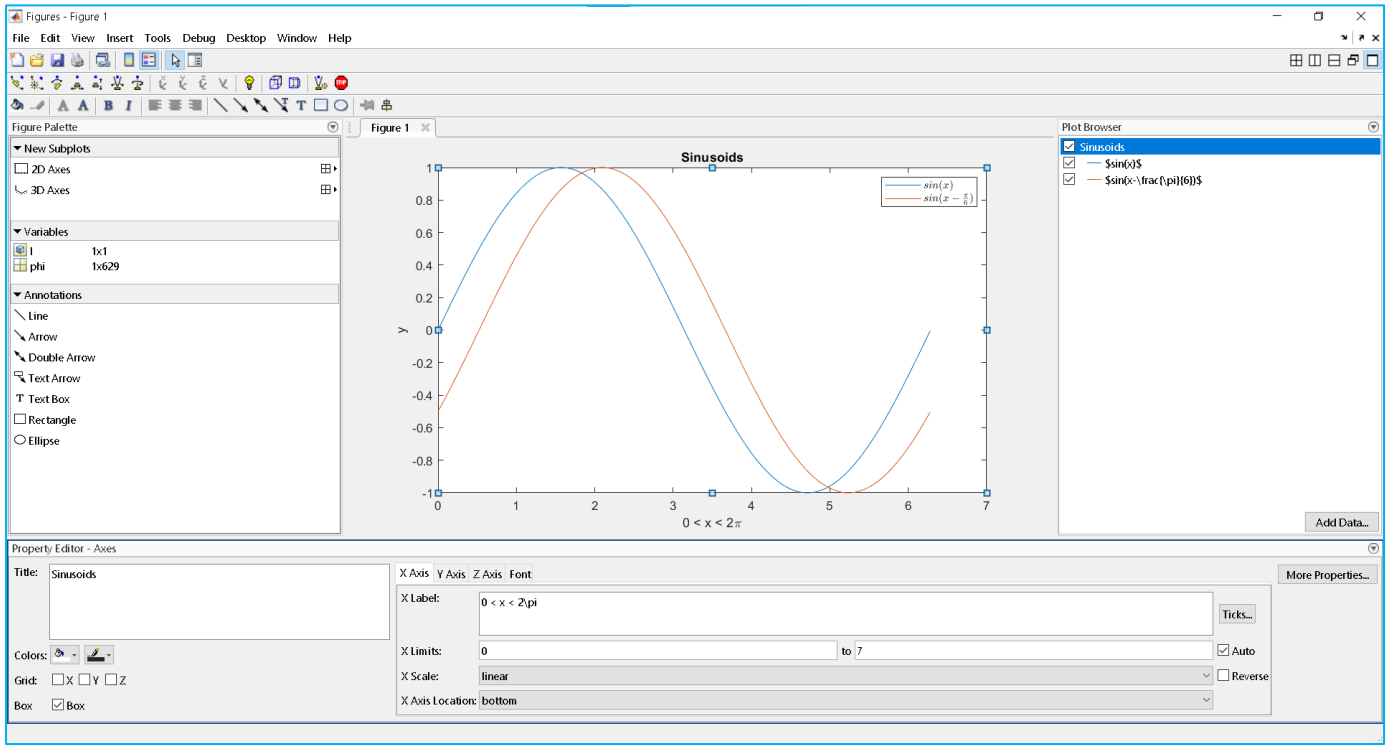
4. Plot Tools



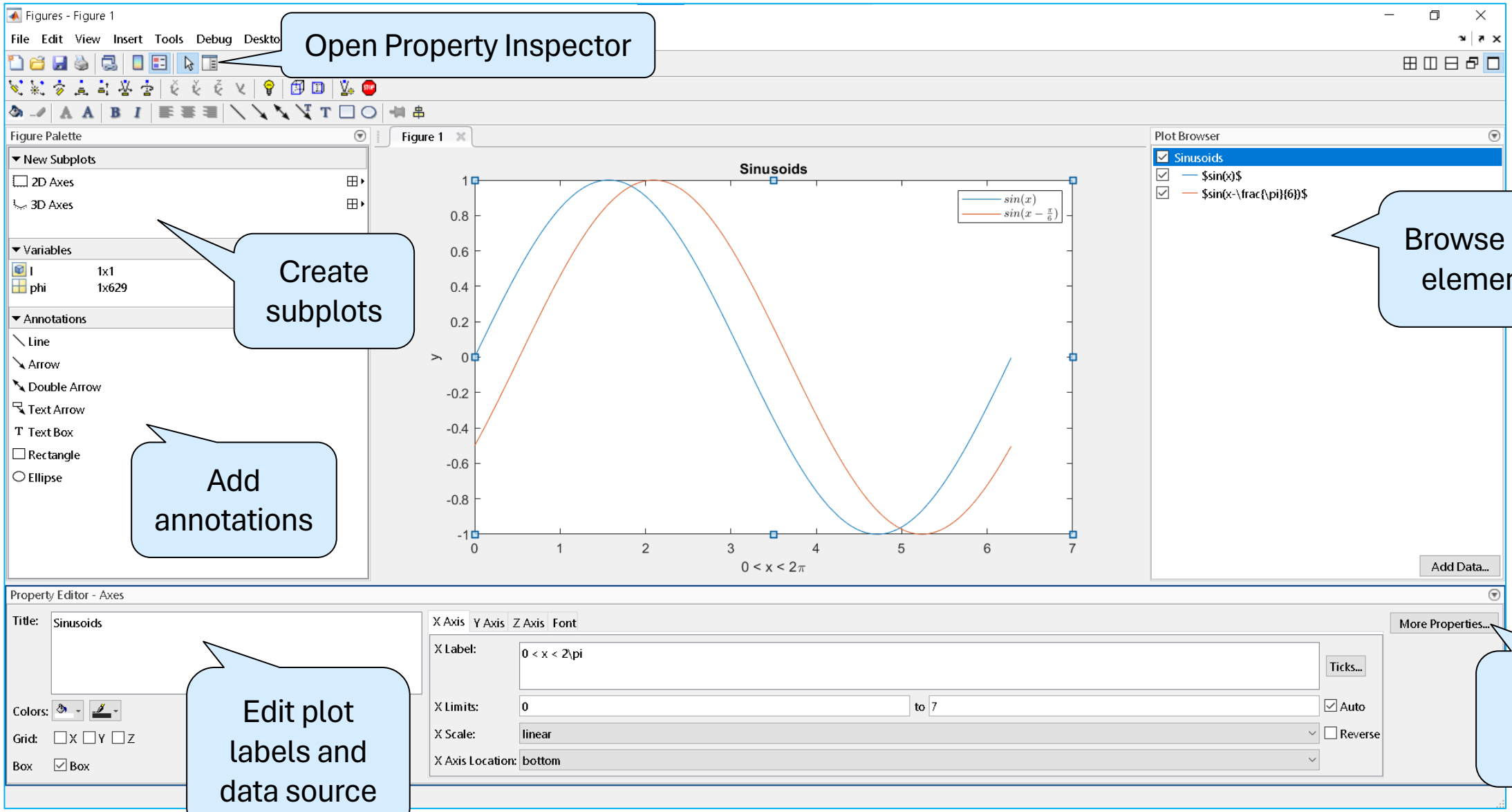
Plot Tools

- Use Plot Tools (now Figures) to **interactively modify a plot:**



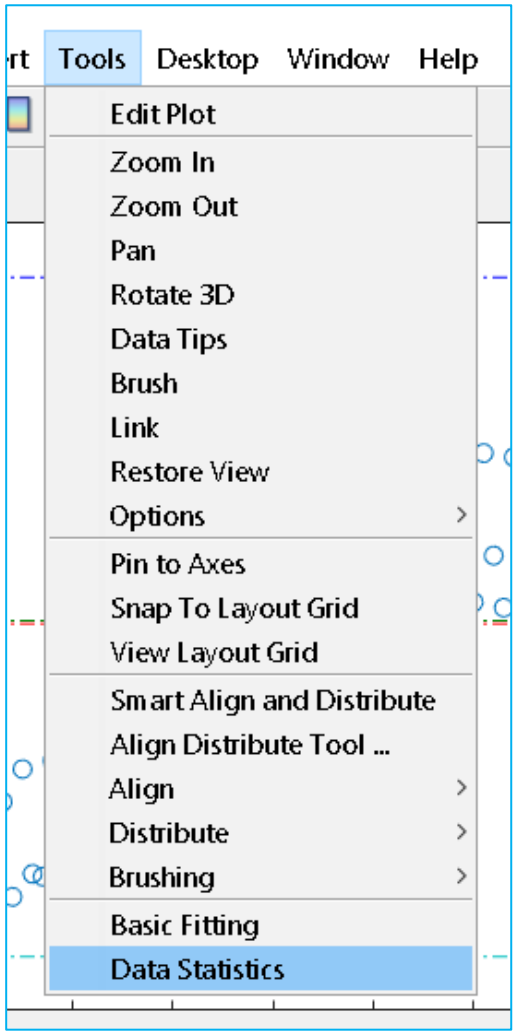


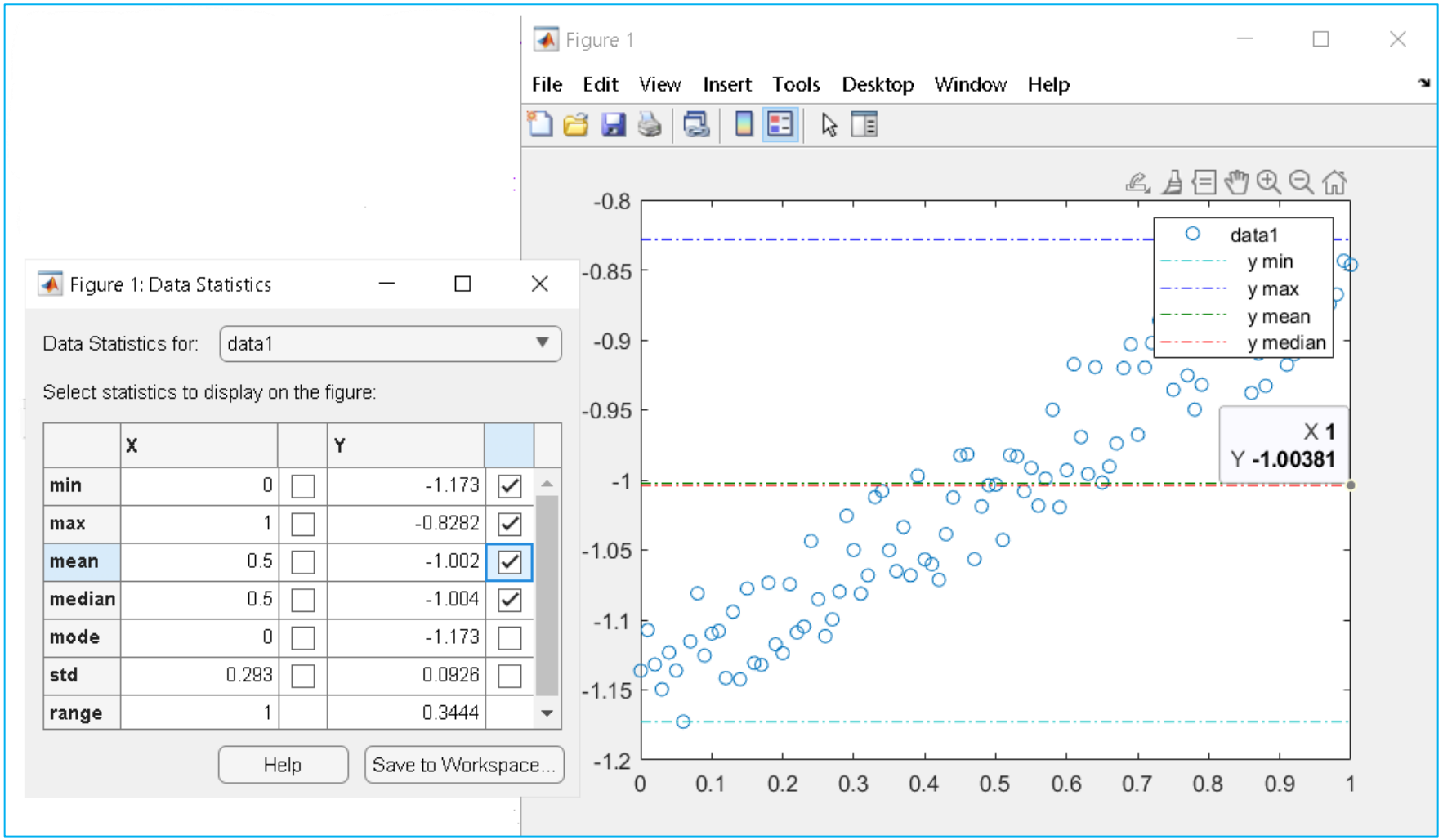
Plot Tools


Figures - Figure 1
File Edit View Insert Tools Debug Deskto
Open Property Inspector
Figure Palette
New Subplots
2D Axes
3D Axes
Variables
I 1x1
phi 1x629
Annotations
Line
Arrow
Double Arrow
Text Arrow
Text Box
Rectangle
Ellipse
Create subplots
Add annotations
Figure 1
Sinusoids
$sin(x)$
$sin(x-\frac{\pi}{6})$
0 < x < 2π
Plot Browser
Sinusoids
$sin(x)$
$sin(x-\frac{\pi}{6})$
Add Data...
Browse plot elements
Property Editor - Axes
Title: Sinusoids
Colors:
Grid: X Y Z
Box Box
X Axis Y Axis Z Axis Font
X Label: 0 < x < 2\pi
Ticks...
X Limits: 0 to 7
Auto
X Scale: linear
Reverse
X Axis Location: bottom
More Properties...
Edit plot labels and data source
Edit graphical properties

Plot Statistics

- In the Tools drop-down menu, select Data Statistics to see the data for the current plot.





	X		Y	
min	0	☐	-1.173	☑
max	1	☐	-0.8282	☑
mean	0.5	☐	-1.002	☑
median	0.5	☐	-1.004	☑
mode	0	☐	-1.173	☐
std	0.293	☐	0.0926	☐
range	1		0.3444	

Save Figure & Generate Code

There are **two options** to save a figure that can be **reopened** in MATLAB later:

- **Save** figure to a .fig-file
  - Use savefig command to **save figure** programmatically
  - Select File > Save to save figure **interactively**
- Generate **code** to recreate figure
  - Select File > Generate Code… to generate a **function that creates the figure**
  - The function takes the **data as input parameter**

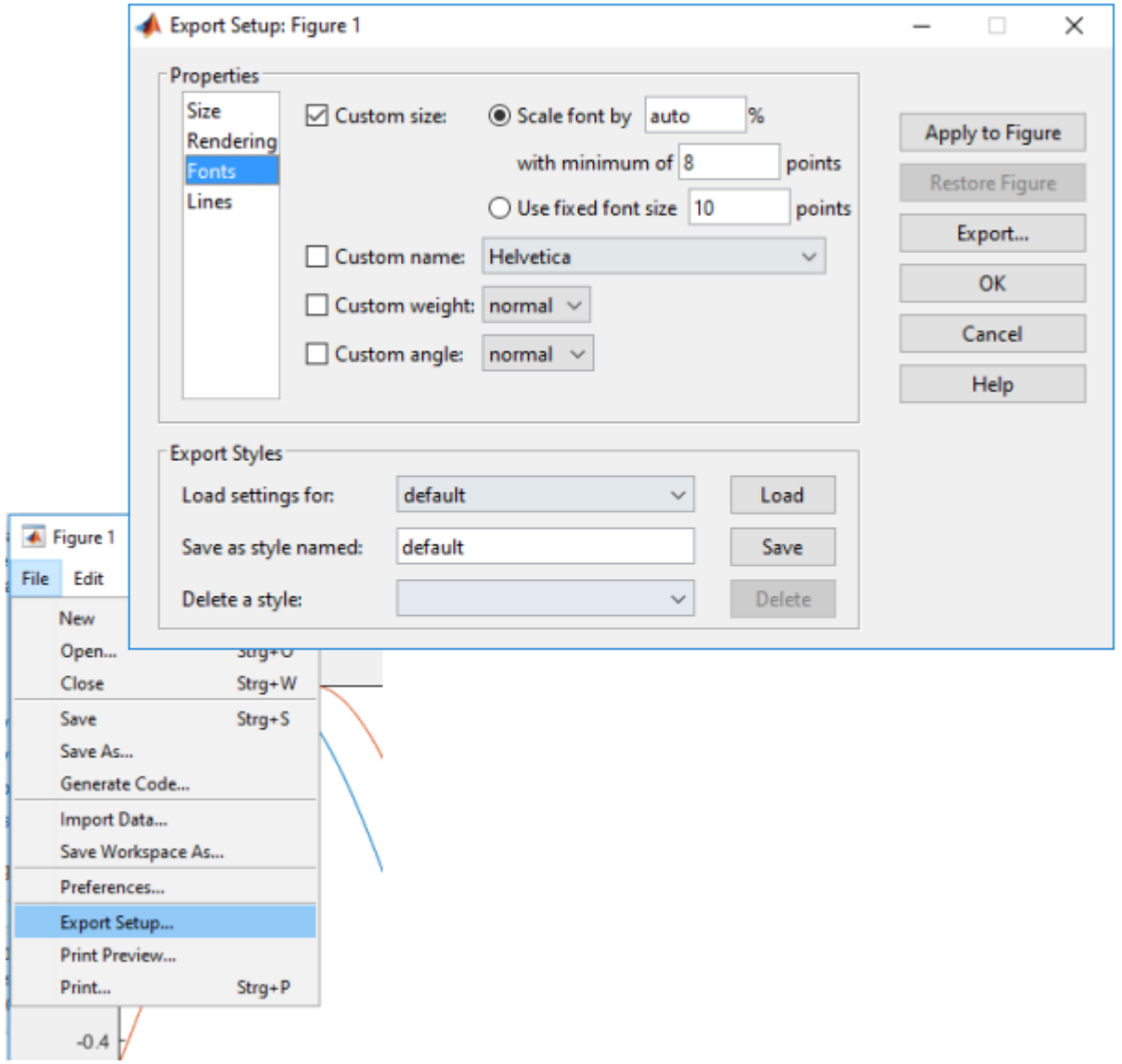
Export Figure to Presentation / Document

Figures to be used outside MATLAB can be saved to file in two types of formats:

- Bitmap image (PNG, JPEG,…)
  - Pixel-based representation
  - Widely used in web applications
  - Badly scalable
- Vector graphics (PDF, EPS, SVG,…)
  - Store commands to redraw figure
  - Well scalable
  - May result in large file

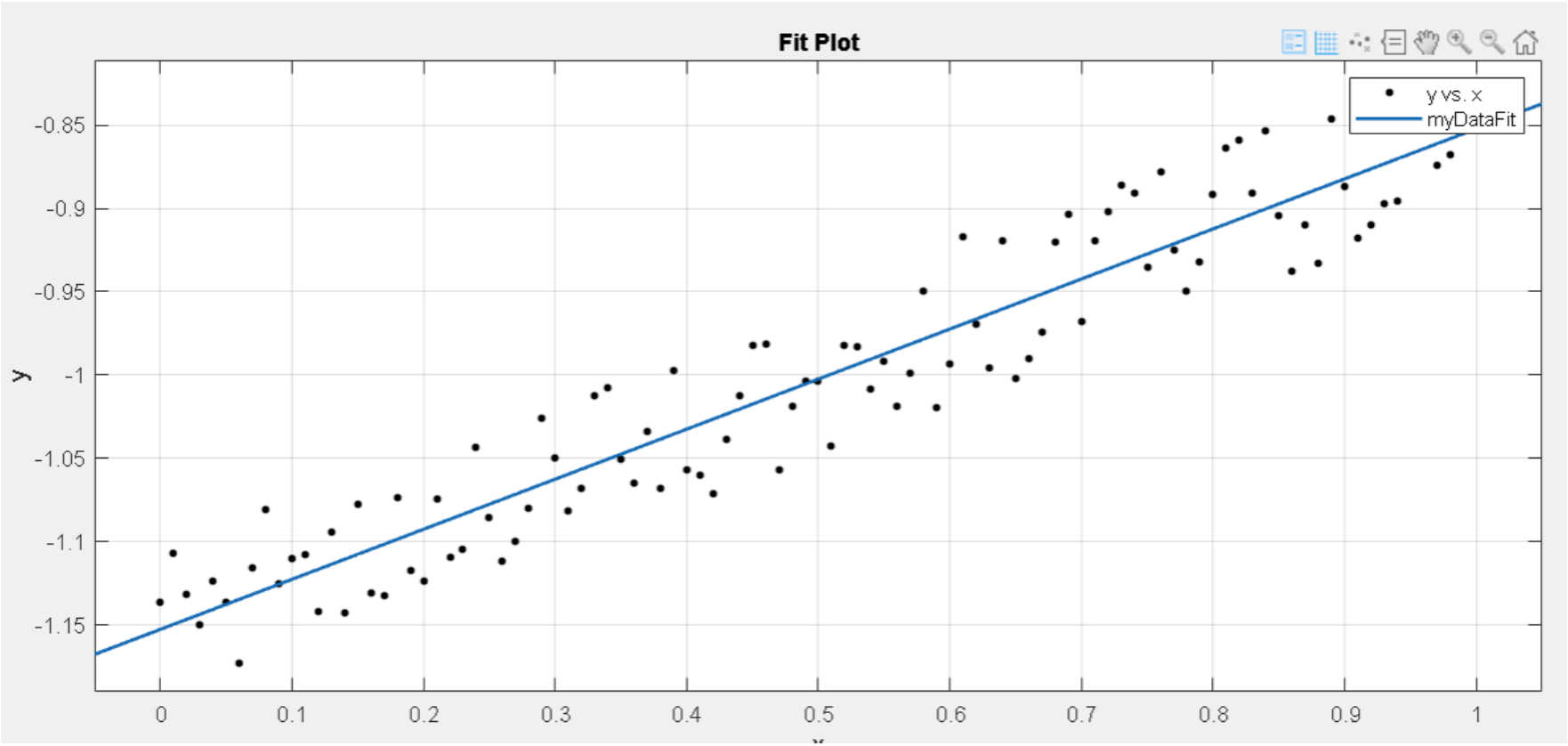
Export Figure to Presentation / Document

- Figures can be customized interactively using the Export Setup
  - Select File > Export Setup…
  - Manually customize figure or load preset export Styles
  - Apply customization to figure, export to file or restore figure

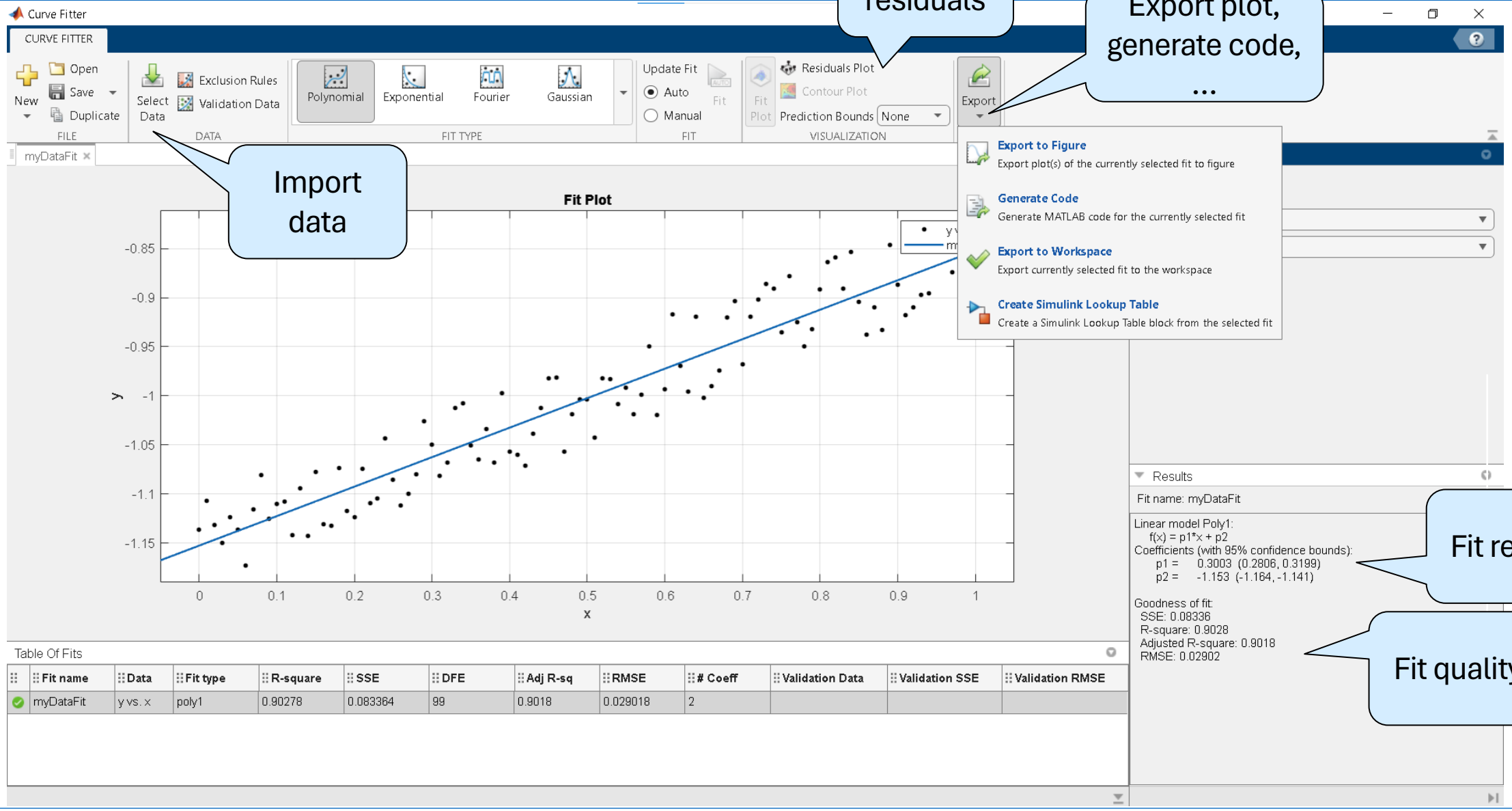


Basic Fitting

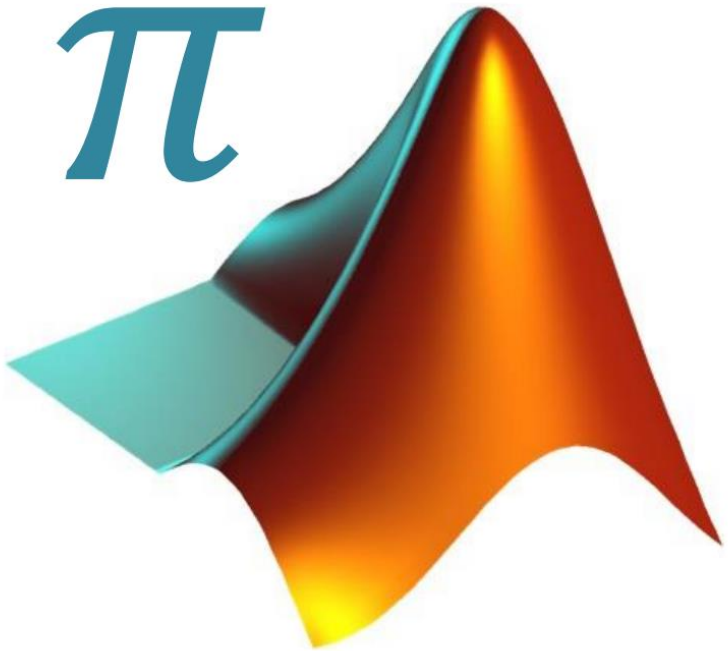
- **Curve fitting** can be done **interactively** using the curveFitter command.
- The app can also be found in the APP tab of the **banner**.
- Programmatically, use the fit command.



Basic Fitting



7. List of Useful Commands



Code Generation: C-Code

Command	Explanation	Slide #
`save`	Save workspace variables to file	4
`load`	Load variables from file into workspace	4
`importdata`	Load data from file	5
`readtable`	Create table from file	7
`csvread`	Read comma-separated value (CSV) file	7
`dlmread`	Read ASCII-delimited file of numeric data into matrix	7
`TabularTextDatastore`	Datastore for tabular text files	7
`textscan`	Read formatted data from text file or string	7
`fopen`	Open file, or obtain information about open files	9
`xlsread`	Read Microsoft Excel spreadsheet file	10
`imread`	Read image from graphics file	11
`imfinfo`	Information about graphics file	11
`audioread`	Read audio file	12
`audioinfo`	Information about audio file	12

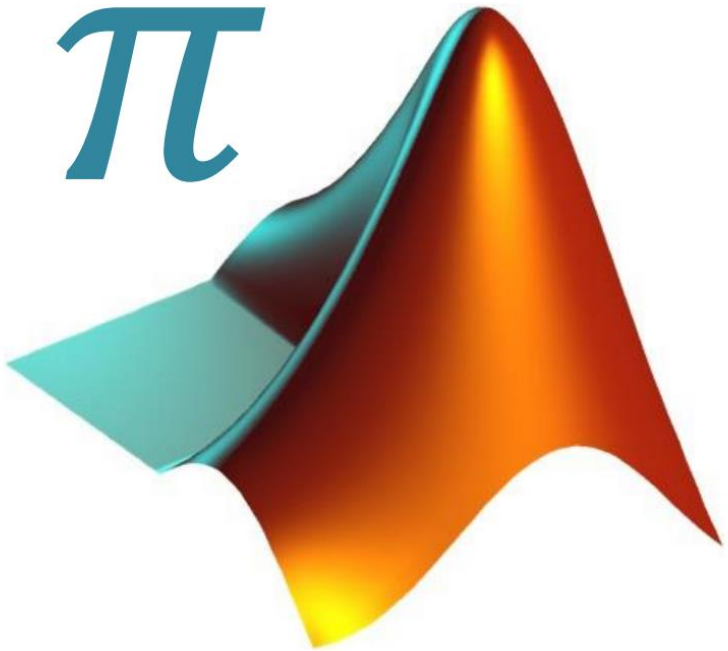
Command	Explanation	Slide #
`VideoReader`	File formats that VideoReader supports	12
`webread`	Read content from RESTful web service	13
`xmlread`	Read XML document and return Document Object Model node	14
`fscanf`	Read data from text file	15
`fgetl/fgets`	Read line from file, removing/keeping newline characters	15
`fread`	Read data from binary file	15
`parfor`	Parallel for loop	16
`datastore`	Create datastore to access collection of data	17
`hasdata`	Determine if data is available to read	17
`mapreduce`	Programming technique for analyzing data sets that do not fit in memory	18
`readall`	Read all data in datastore	18
`global`	Declare variables as global	22
`persistent`	Define persistent variable	22

Code Generation: C-Code

Command	Explanation	Slide #
`plot`	2-D line plot	26
`figure`	Create figure window	26
`hold`	Retain current plot when adding new plots	28
`subplot`	Create axes in tiled positions	29
`gca`	Current axes handle	30
`grid`	Display or hide axes grid lines	30
`plot3`	3-D line plot	32
`surf`	3-D shaded surface plot	32
`histogram`	Histogram plot	33
`quiver`	Quiver or velocity plot	33
`polar`	Polar coordinate plot	34
`spy`	Visualize sparsity pattern	34
`xlabel/ylabel`	Label x-axis/y-axis	35
`title`	Add title to current axes	35
`legend`	Add legend to graph	35

Command	Explanation	Slide #
`annotation`	Create annotations	35
`imshow`	Display image	36
`subimage`	Display multiple images in single figure	36
`gobjects`	Initialize array for graphics objects	36
`rgb2gray`	Convert RGB image or colormap to grayscale	36
`axis`	Set axis limits and appearance	36
`min/max`	Smallest/largest elements in array	42
`mean`	Average or mean value of array	42
`median`	Median value of array	42
`mode`	Most frequent values in array	42
`std`	Standard deviation	42
`range`	Range of values	42
`annotation`	Create annotations	35

8. Self-assessment



Self-assessment

- What is a CSV and what its initials stand for?
- How do you import an image in the workspace and how do you make it appear in its own window?
- What do csvread and readtable do? What does the current documentation say about them?
- What two workspaces are there in MATLAB?
- How can you clear a function’s workspace?
- What are persistent and global variables?
- Write your own counter function with a persistent and a global variable, as shown in this presentation. Pay attention to the variable syntax and try to reproduce the experiment.
- Why pre-allocate memory?
- What is lazy copy implementation?
- Draw the graphics objects hierarchy for a simple figure.
- What does gca do return?