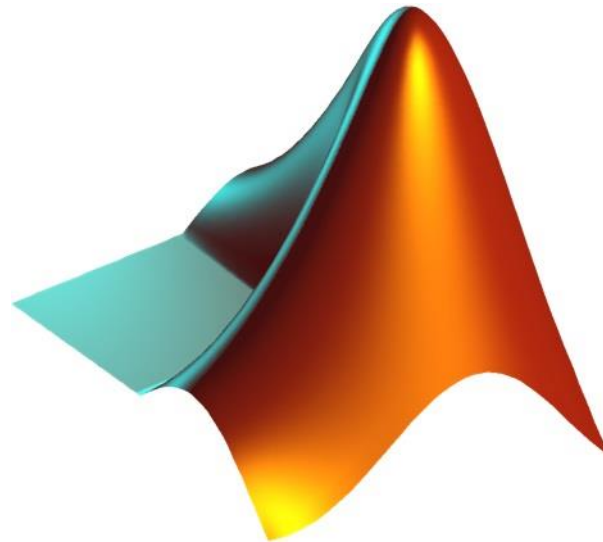




# Practical Course MATLAB/SIMULINK

## Data Handling and Visualization

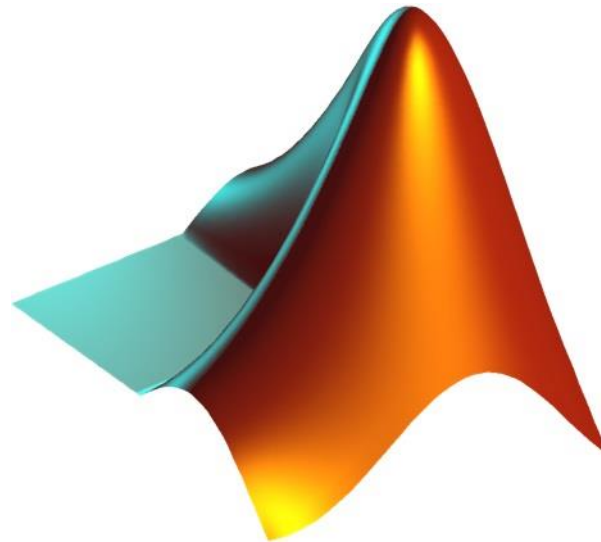


# Objectives & Preparation “Data Handling and Visualization”

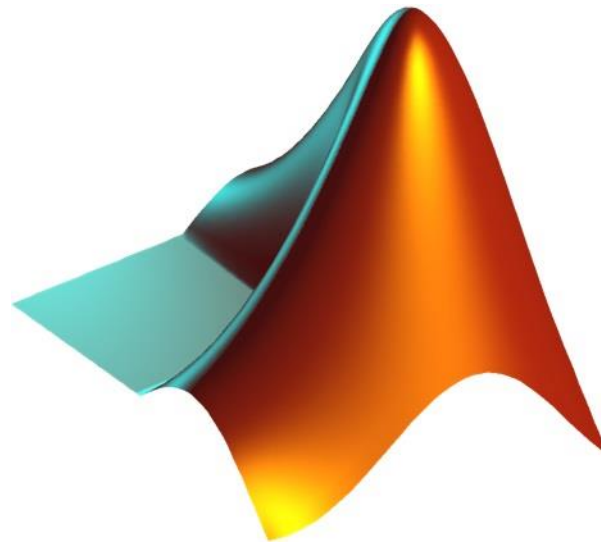
- Which MathWorks products are covered?
  - ⇒ MATLAB
  
- What skills are learnt?
  - ⇒ Data types, import and export
  - ⇒ Memory management
  - ⇒ Visualization, graphics tools
  
- How to prepare for the session?
  - ⇒ MathWorks Tutorials:
    - <https://matlabacademy.mathworks.com/R2017b/portal.html?course=mlvi>  

    - <https://www.mathworks.com/examples/matlab/category/graphics>  


# Outline

1. Import and Export Data
2. Memory Management
3. Graphics
4. Plot Tools
5. List of Useful Commands



# 1. Import and Export Data



# Import and Export Data - Loading MAT-Files

- Binary file containing MATLAB formatted data
- Used to save variables from the MATLAB workspace
- Use save and load command

Mat-files can be loaded interactively using the Open Button



The screenshot displays the MATLAB environment. The top ribbon includes tabs for HOME, PLOTS, and APPS. The HOME tab is active, showing various icons for file operations (New Script, New, Open, Compare), variable management (New Variable, Open Variable, Save Workspace, Clear Workspace), code execution (Analyze Code, Run and Time, Clear Commands), and other tools (Simulink Library, Layout, Set Path, Add-Ons, Help, Community, Request Support).

The Command Window shows the following commands:

```
>> x=A\b;  
>> save('data.mat')  
>> clear all  
>> load('data.mat')  
fx >> |
```

The Current Folder panel on the left shows the file 'data.mat' (MAT-file) selected. Below it, a table lists the variables saved in the file:

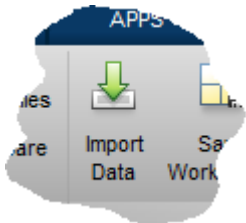
Name	Value
A	[8 1 6; 3 5 7; 4 9 ...]
b	[1; 2; 3]
x	[0.0500; 0.3000; 0.0500]

The Workspace panel on the right shows the current variables in the workspace:

Name	Value
A	[8 1 6; 3 5 7; 4 9 2]
b	[1; 2; 3]
x	[0.0500; 0.3000; 0.0500]

# Import and Export Data - Importing Text Files Interactively

- Text files (including .txt, .dat, **.csv**, .asc, .tab, and .dlm) can be imported using MATLAB's Import Tool
- Desired data type is saved to MATLAB workspace



Import - D:\Matlab\data.txt

**Desired data type**

IMPORT | VIEW

☒ Delimited    Column delimiters: Comma    Range: A1:A1    Variable Names Row: 1

☐ Fixed Width    More Options

Column vectors

- ☒ Numeric Matrix
- ☐ Cell Array
- ☐ Table

Replace    unimportable cells with    NaN

Import Selection

	A	B	C	D
	data			
	time	x	y	z
	NUMBER	NUMBER	NUMBER	NUMBER
1	time	x	y	z
2	0	1	0	10
3	0.1	0.99987663...	0.01570731...	10.05
4	0.2	0.99950656...	0.03141075...	10.1
5	0.3	0.99888987...	0.04710645...	10.15
6	0.4	0.99802672...	0.06279051...	10.2
7	0.5	0.99691733...	0.07845909...	10.25
8	0.6	0.99556196...	0.09410831...	10.3

**Generate code to repeat operation for multiple similar files**

## Import and Export Data - Importing Text Files Programmatically

- Text files (including .txt, .dat, .csv, .asc, .tab, and .dlm) 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
```

## Import and Export Data - Ways to Import Text Files

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



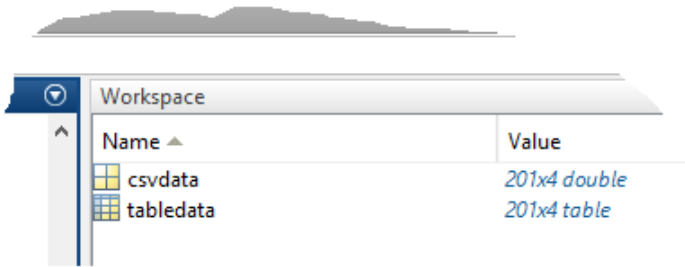
# Import and Export Data - Ways to Import Text Files - Examples

## 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

# Import and Export Data - Ways to Import Text Files - Examples

## textscan

Read formatted data from text file or string

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

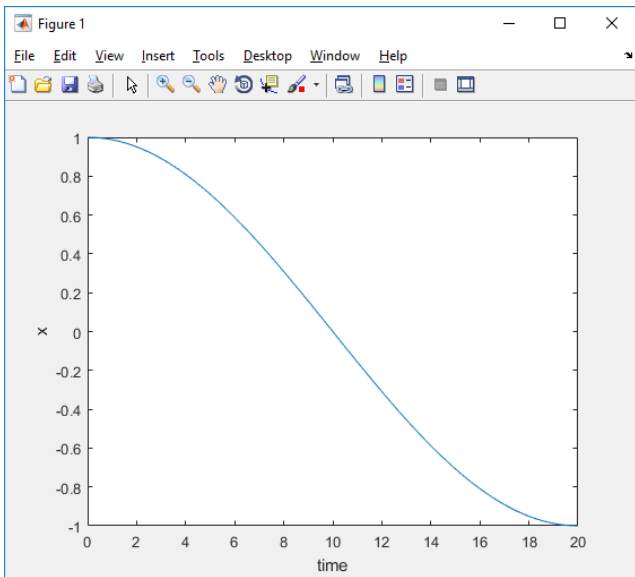
```
>> [num, position] = textscan(fid, '%f,%f,%f,%f'); whos num
```

Name	Size	Bytes	Class	Attributes
num	1x4	6880	cell	

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})
```

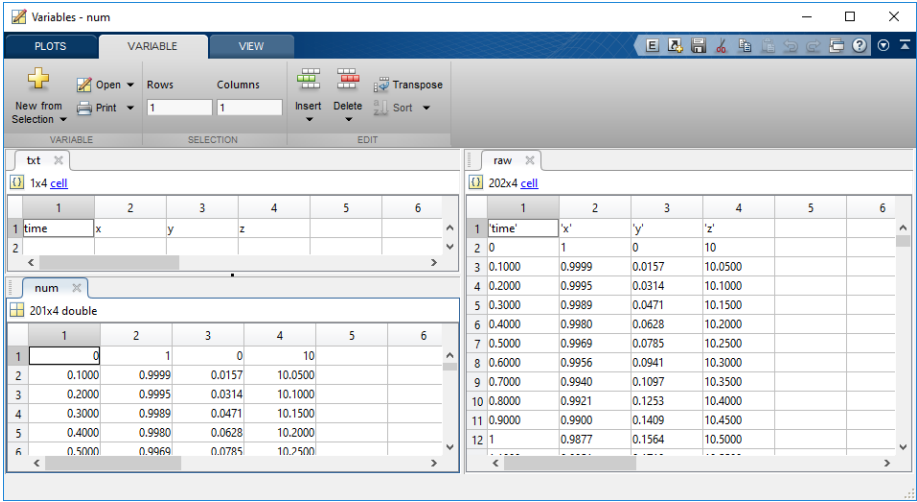
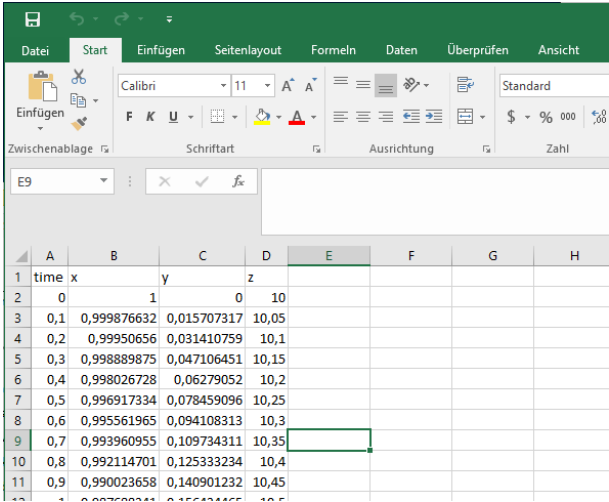


# Import and Export Data - Importing Spreadsheets

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	num contains numeric data
raw	202x4	96942	cell	raw Cell array of all data as char arrays
txt	1x4	462	cell	txt Cell array containing only text data



## Import and Export Data - Importing Images

- Import image files in many standard file formats including TIFF, GIF, JPEG, PNG...

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

- Information about the image file can be retrieved using the `imfinfo` command

```
>> ImgFileInfo = imfinfo('ngc6543a.jpg');  
>> ImgFileInfo.Height
```

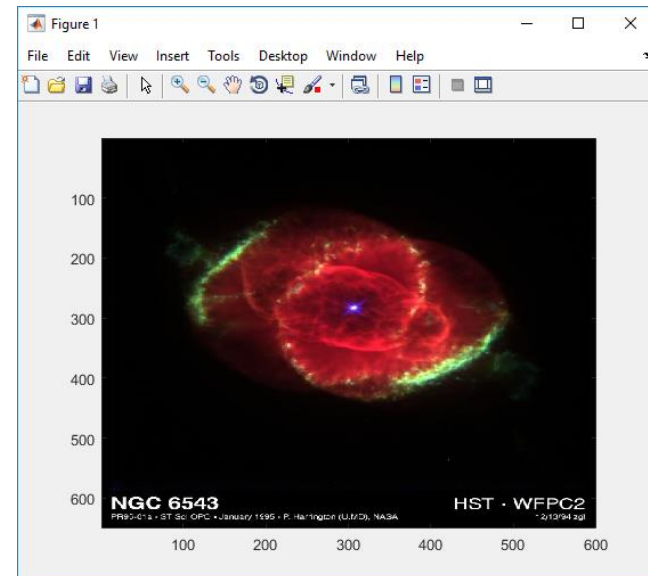
```
ans =  
  
    650
```

```
>> ImgFileInfo.Format
```

```
ans =  
  
    jpg
```

```
>> ImgFileInfo.BitDepth
```

```
ans =  
  
    24
```



[www.mathworks.de](http://www.mathworks.de)



## Import and Export Data - Importing Audio and Video Files

### 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
```

### Videos

- Videos can be imported using the Video Reader Class

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

## Import and Export Data - Read Web Data

### webread

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)
```

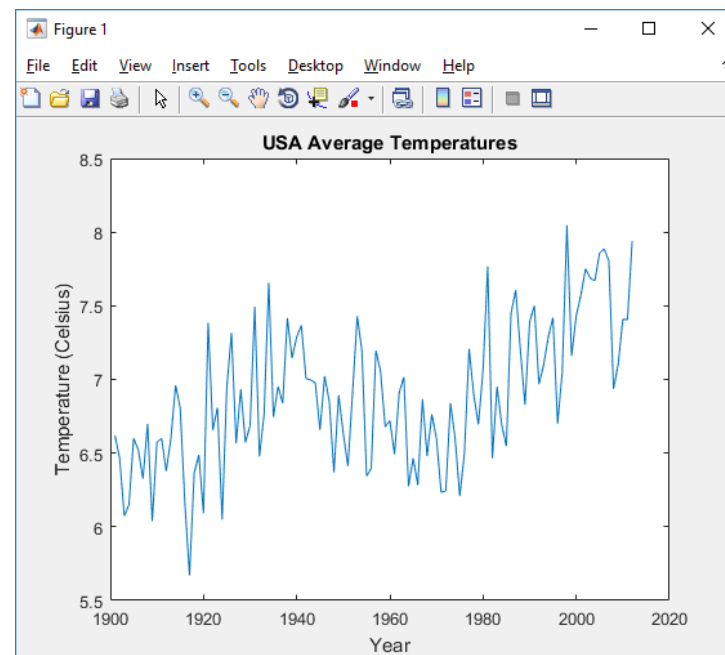
climateData =

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.

## Import and Export Data - Read XML Data

### xmlread

XML documents can be read into MATLAB using the `xmlread` command, which 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>
```

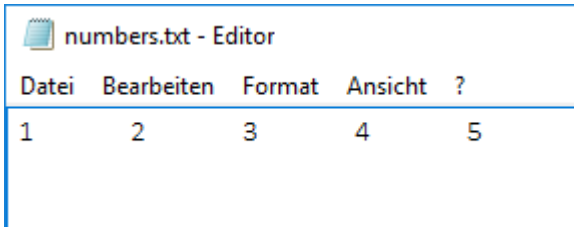
# Import and Export Data - Data Import with Low-Level I/O

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

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 or bit level

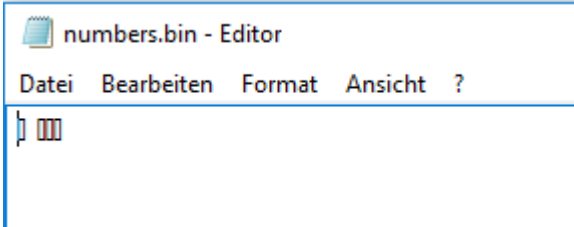
## fscanf

```
>> 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');
```





## Import and Export Data - Large Files and Big Data

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

### ▪ **Parallel Pools**

If several independent operations have to be performed a parallel pool can be created containing several MATLAB workers.

Example: Using five workers to perform three large eigenvalue computations

```
>> c = nan(1000,5);  
>> tic;  
>> parfor iter = 1 : 5  
    c(:,iter) = eig(rand(1000));  
end  
>> toc
```

Elapsed time is 2.124193 seconds.

```
>> c = nan(1000,5);  
>> tic;  
>> for iter = 1 : 5  
    c(:,iter) = eig(rand(1000));  
end  
>> toc
```

Elapsed time is 3.198071 seconds.

- Iterations are executed individually on their own thread
- Running dependent iterations in parallel can lead to errors

## Import and Export Data - Large Files and Big Data

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

- **datastore**

A datastore object is a repository for data that has the same structure and formatting. A datastore is useful when

- each file in the collection is too large to fit into memory
- files in the store have arbitrary names

- 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
```

Only 5000 values are read into memory every iteration

## Import and Export Data - Large Files and Big Data

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

- **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 final result

- Example: Chunk through a big file to find mean delay (using mapreduce)

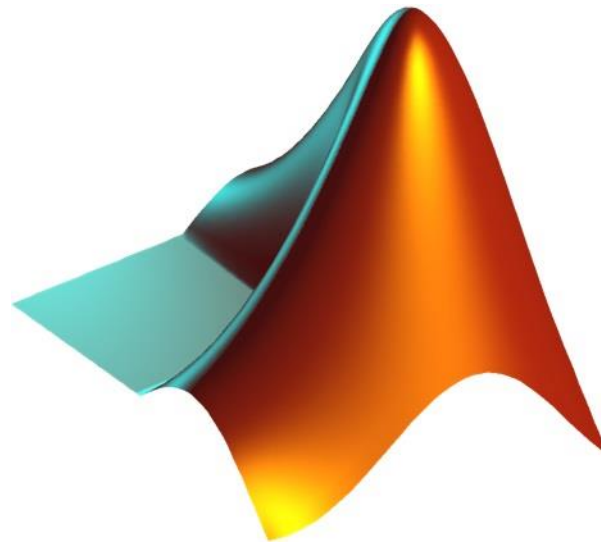
```
>> ds = datastore('airlinesmall.csv');  
>> ds.ReadSize = 5000;  
>> ds.TreatAsMissing = 'NA';  
>> ds.SelectedVariableNames = 'ArrDelay';  
>> meanDelay = mapreduce(ds,...  
    @meanArrDelayMapper,...  
    @meanArrDelayReducer);  
>> T = readall(meanDelay);  
>> fprintf('The mean arrival delay 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

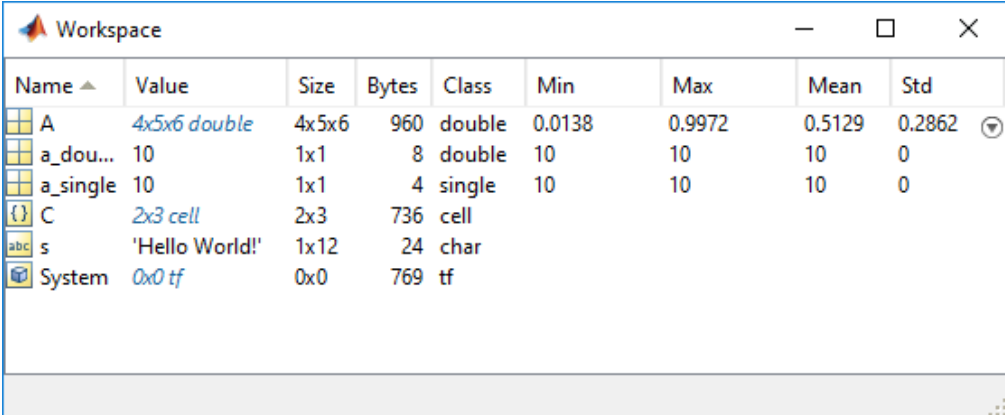
## 2. Memory Management



# Memory Management - Workspaces

## MATLAB Workspace

- Consists of the variable 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
  - Statistical data (Min, Max, Mean...)
  - ...

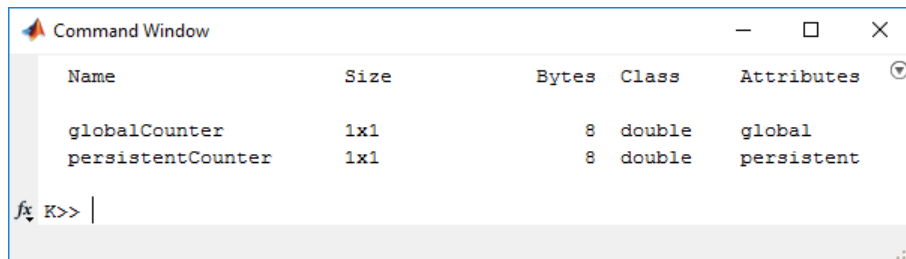


Name	Value	Size	Bytes	Class	Min	Max	Mean	Std
A	4x5x6 double	4x5x6	960	double	0.0138	0.9972	0.5129	0.2862
a_dou...	10	1x1	8	double	10	10	10	0
a_single	10	1x1	4	single	10	10	10	0
C	2x3 cell	2x3	736	cell				
s	'Hello World!'	1x12	24	char				
System	0x0 tf	0x0	769	tf				

# Memory Management - Workspaces

## 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)



A screenshot of the MATLAB Command Window. The title bar says 'Command Window'. Below the title bar is a table with the following columns: Name, Size, Bytes, Class, and Attributes. The table contains two rows of data: 'globalCounter' with size '1x1', 8 bytes, class 'double', and attribute 'global'; and 'persistentCounter' with size '1x1', 8 bytes, class 'double', and attribute 'persistent'. At the bottom of the window, the prompt 'fx K>> |' is visible.

Name	Size	Bytes	Class	Attributes
globalCounter	1x1	8	double	global
persistentCounter	1x1	8	double	persistent

## 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

# Memory Management – Global and Persistent

## Persistent Variables

- local to the function in which they are declared
- Value is retained between function calls

## Global Variables

- All functions that declare a variable global share a single copy

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

ans =  
7 1

global Counter	persistent Counter
0	0
1	1
1	[]
2	0
2	0
6	0
7	1

Both persistent and global variables are initialized empty

```
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
```

## Memory Management – 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:
  - Contiguous virtual block containing array data
  - Separate small block called *header* containing information about the array data such as
    - Class
    - Dimensions
    - ...
- If a new element is added to the array, MATLAB expands the existing array in memory keeping storage contiguous
  - usually requires finding a new block of memory
  - Preallocation:

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

Elapsed time is 0.070668 seconds.

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

Elapsed time is 0.017849 seconds.

Storage has to be allocated only once



# Memory Management – 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;
```

There is no need to copy data for an exact copy of variables

Memory used (Gigabyte)
1.0575
4.2516
4.2516

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

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

5.8516
--------

- structure members are treated as a separate arrays in MATLAB

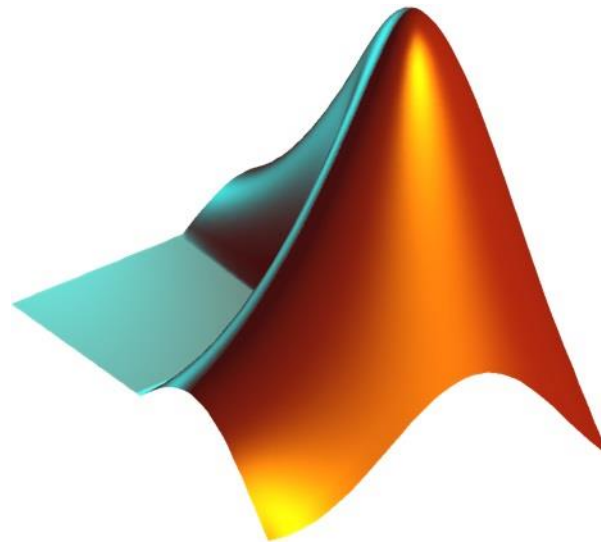
```
>> S.A = A; S.B = B;
>> S.A(1) = 0;
>> S.B(1) = 0;
```

In this step, only the field A is copied in memory since B is not changed

5.8590
9.0635
10.655

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

# 3. Graphics



# 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)
```

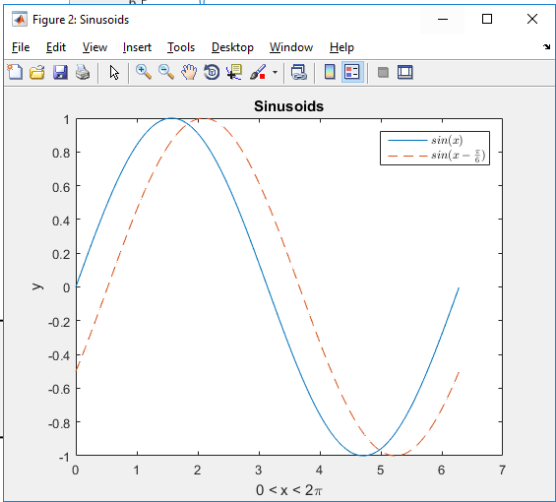
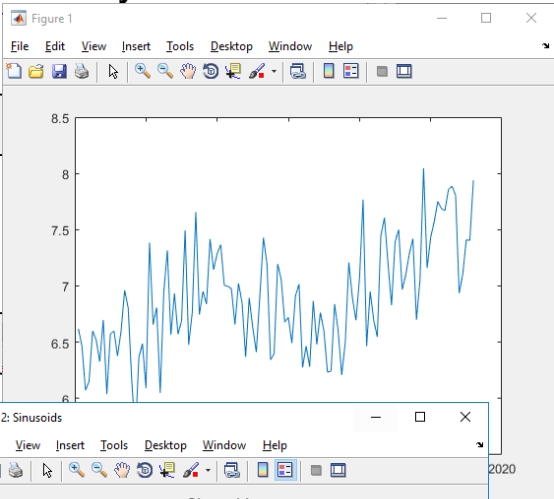
- The plot command automatically creates a figure object. The figure command can be used to create a new figure.

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

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

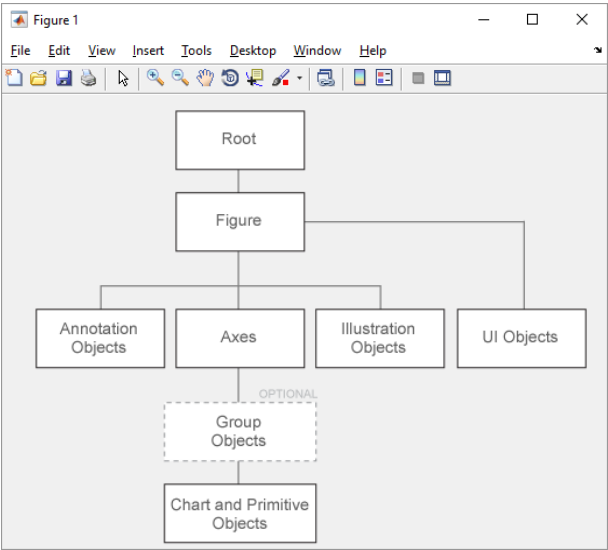
```
>> phi = 0:0.01:2*pi;  
>> p = plot(phi, sin(phi), phi, sin(phi-pi/6)); whos p
```

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

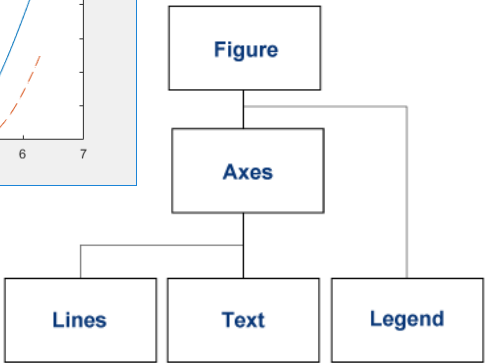
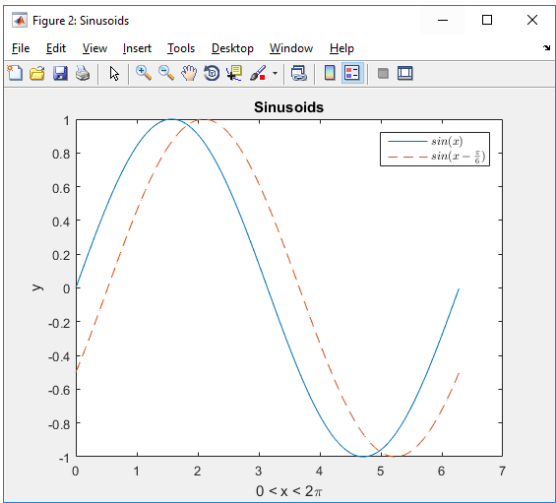


# Graphics – Graphics Objects

- 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)



Example:  
Observe the hierarchy for the figure created on the last slide:

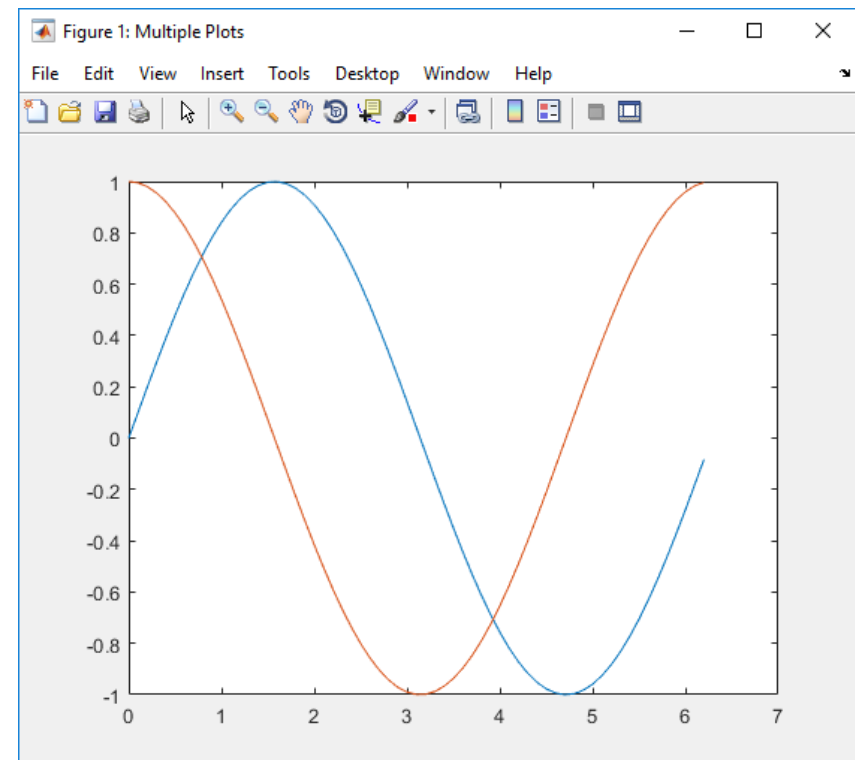


## Graphics – Multiple Plots

### hold

- when calling the plot function twice for the same axes, the first plot is replaced
- using the hold command, plots can be retained

```
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));
```

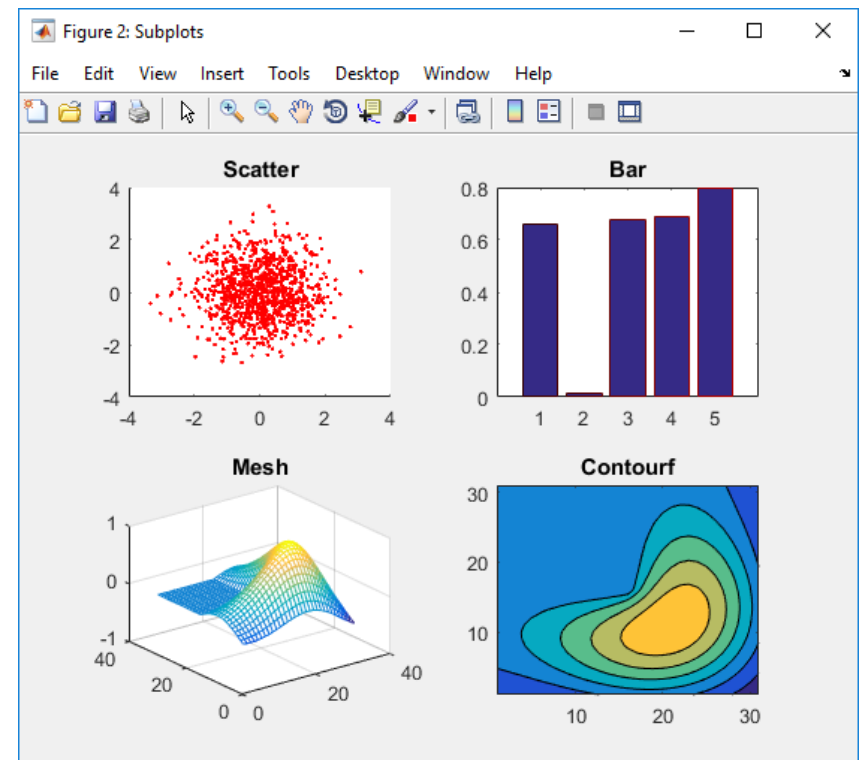


# Graphics – Multiple Plots

## 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)

```
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');
```



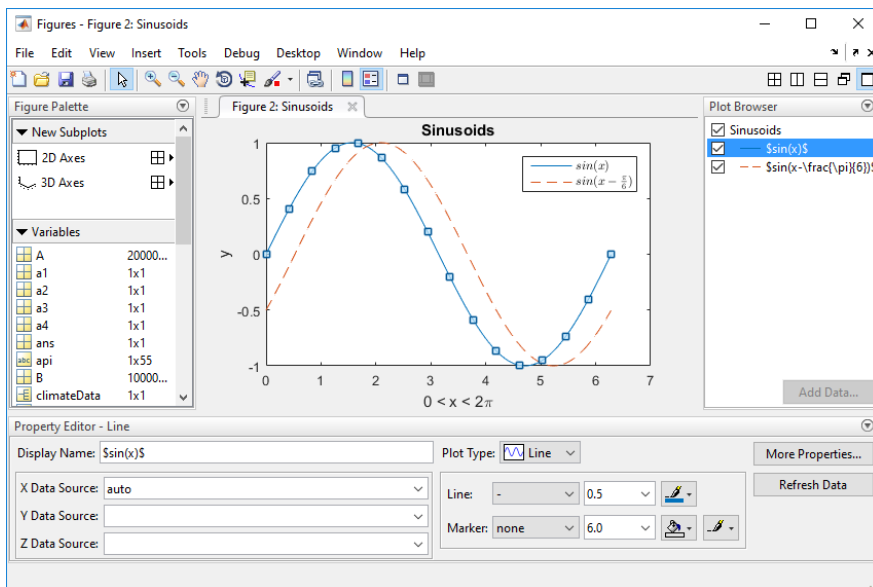
# Graphics – Graphics Object Properties

Behavior and appearance of a graphics object can be controlled by setting its properties.

## Interactively

Using plot tools a graph can be customized

- change graphics objects' properties
- Add new plots and annotations
- ...



## Programmatically

current axes (see also: gcf – current figure)

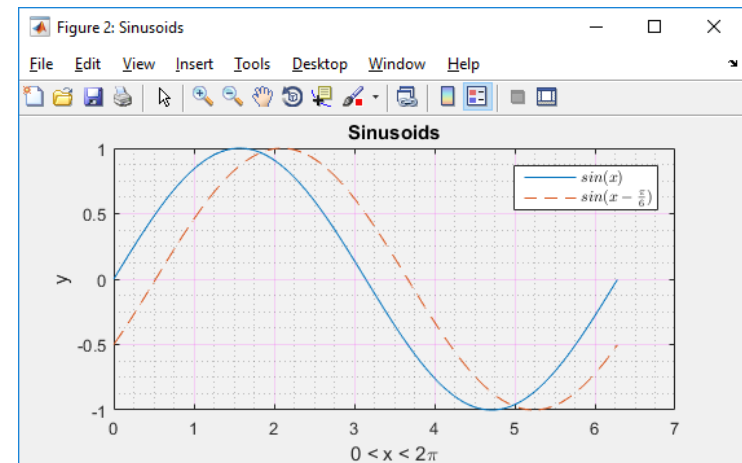
```
>> ax = gca;
```

- Access and modify properties using the dot notation (starting in R2014b)

```
>> ax.Color = 0.95*[1,1,1]
```






















































```
>> grid(ax, 'on'); grid(ax, 'minor');
```

```
>> ax.GridColor = 'magenta';
```



# Graphics – 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

Line Plots	Pie Charts, Bar Plots, and Histograms	Discrete Data Plots	Polar Plots	Contour Plots	Vector Fields	Surface and Mesh Plots		Polygons	Animation
<code>plot</code> 	<code>area</code> 	<code>stairs</code> 	<code>polar</code> 	<code>contour</code> 	<code>quiver</code> 	<code>surf</code> 	<code>mesh</code> 	<code>fill</code> 	<code>animatedline</code> 
<code>plot3</code> 	<code>pie</code> 	<code>stem</code> 	<code>rose</code> 	<code>contourf</code> 	<code>quiver3</code> 	<code>surfc</code> 	<code>meshc</code> 	<code>fill3</code> 	<code>comet</code> 
<code>loglog</code> 	<code>pie3</code> 	<code>stem3</code> 	<code>compass</code> 	<code>contour3</code> 	<code>feather</code> 	<code>surf1</code> 	<code>meshz</code> 	<code>patch</code> 	<code>comet3</code> 
<code>semilogx</code> 	<code>bar</code> 	<code>scatter</code> 	<code>ezpolar</code> 	<code>contourslice</code> 	<code>streamslice</code> 	<code>ezsurf</code> 	<code>waterfall</code> 		
<code>semilogy</code> 	<code>barh</code> 	<code>scatter3</code> 		<code>ezcontour</code> 	<code>streamline</code> 	<code>ezsurfc</code> 	<code>ezmesh</code> 		
<code>errorbar</code> 	<code>bar3</code> 	<code>spy</code> 		<code>ezcontourf</code> 	<code>streamribbon</code> 	<code>ribbon</code> 	<code>ezmeshc</code> 		
<code>ezplot</code> 	<code>bar3h</code> 	<code>plotmatrix</code> 			<code>streamtube</code> 	<code>pcolor</code> 			
<code>ezplot3</code> 	<code>histogram</code> 				<code>coneplot</code> 				
	<code>pareto</code> 								

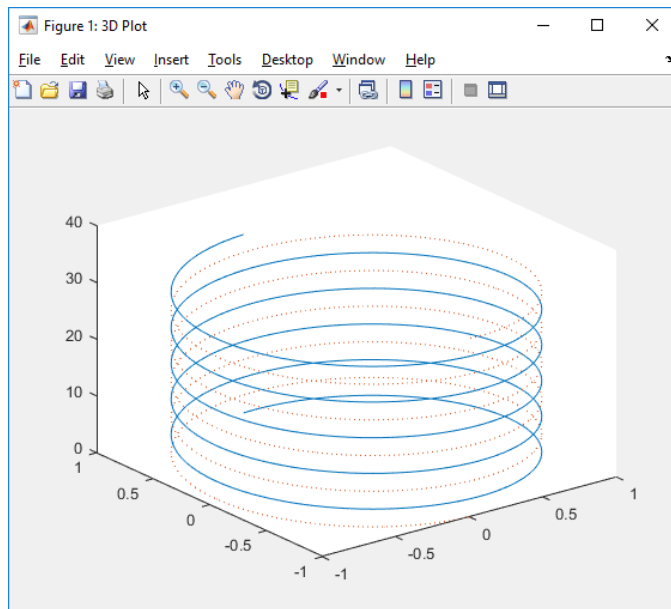


# Graphics – Plot Type Examples

## plot3

Creates a line plot in three dimensions similarly to the plot function.

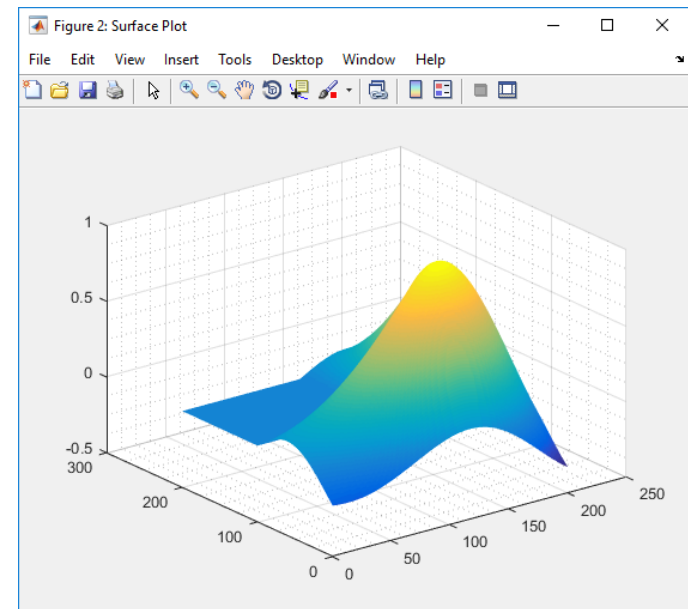
```
>> t = 0:pi/50:10*pi;  
>> st = sin(t);  
>> ct = cos(t);  
>> h = figure  
>> p = plot3(st, ct, t, '-st, -ct, t, ':'')
```



## 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';
```

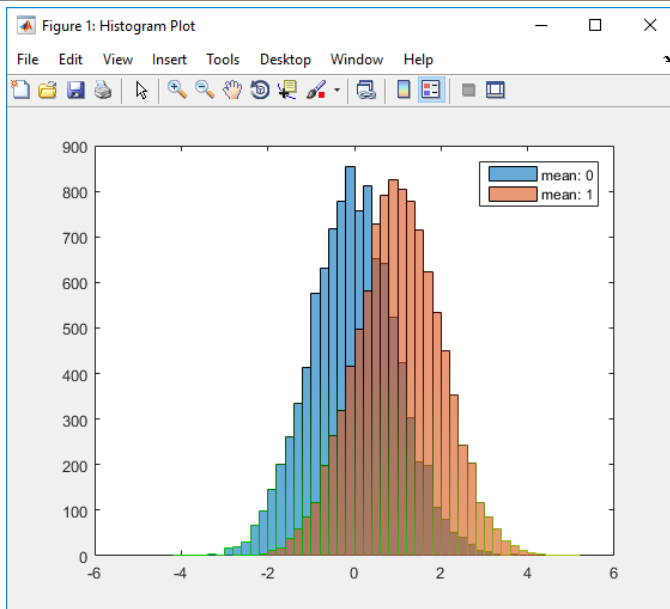


# Graphics – Plot Type Examples

## histogram

Histograms are a type of bar plot for numeric data that group the data into bins.

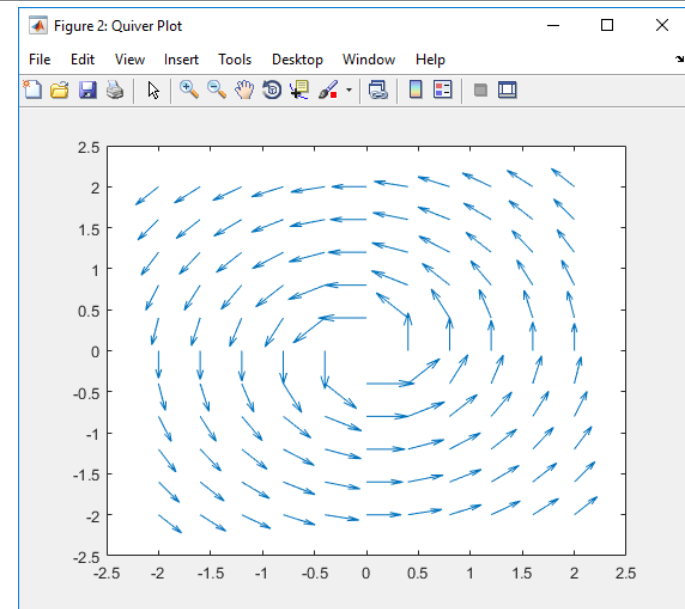
```
>> x1 = randn(10000,1); x2 = randn(10000,1) + 1;
>> h = figure('Name','Histogram Plot');
>> p = gobjects(2,1);
>> p(1) = histogram(x1); hold on;
>> p(2) = histogram(x2); legend('mean: 0','mean: 1');
```



## quiver

Quiver (or 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);
```

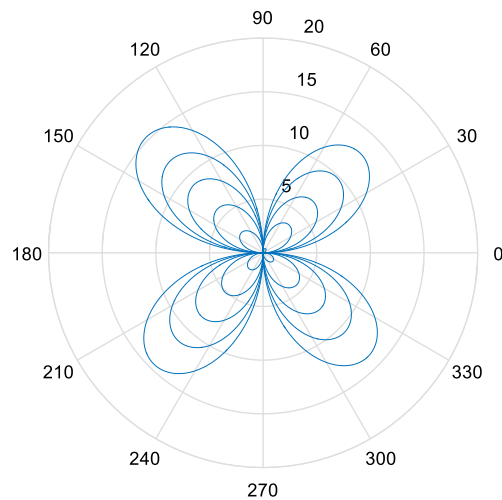


## Graphics – Plot Type Examples

### polar

Accepts data in polar coordinates and plots them in the Cartesian plane.

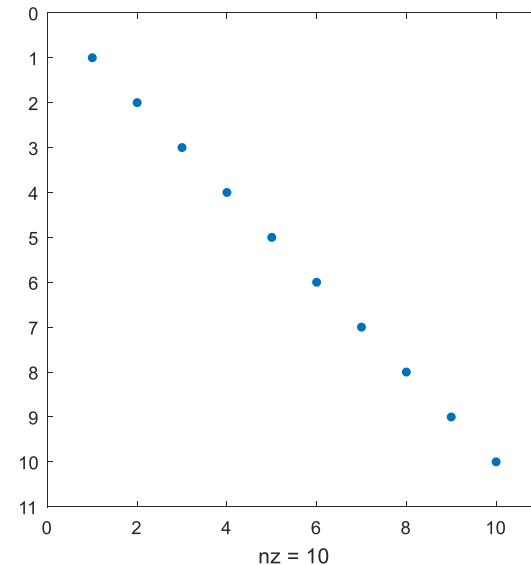
```
>> h = figure('Name','Polar Plot');  
>> phi = 0:0.01:10*pi;  
>> r = phi.*sin(phi).*cos(phi);  
>> p = polar(phi, r);
```



### spy

Visualizes the sparsity pattern of a sparse matrix.

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

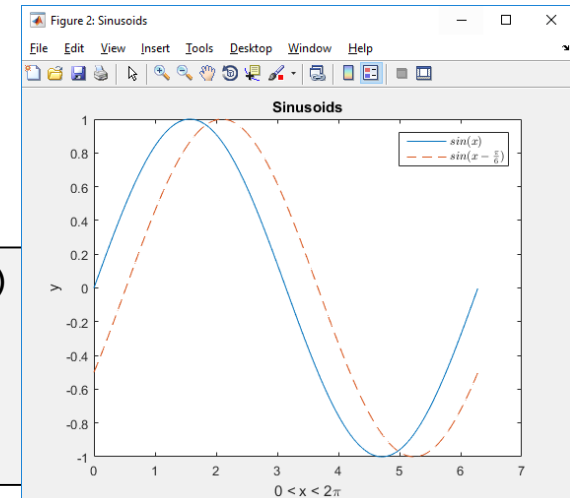


# Graphics – Formatting and Annotations

## Title, Labels and Legends

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

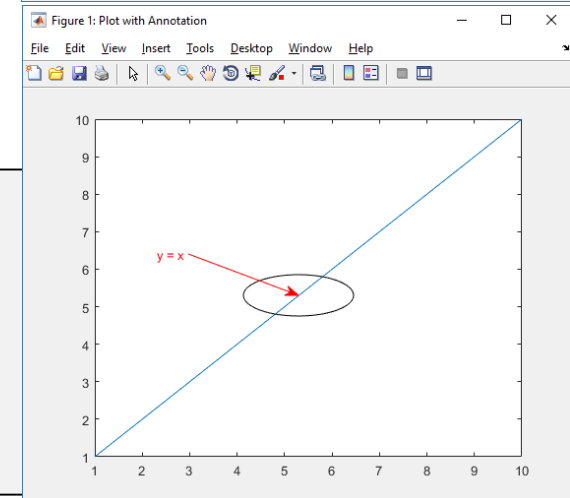
```
>> phi = 0:0.01:2*pi; plot(phi, sin(phi), phi, sin(phi-pi/6))
>> xlabel('0 < x < 2\pi'); ylabel('y'); title('Sinusoids');
>> l = legend('$sin(x)$','$sin(x-\frac{\pi}{6})$');
>> l.Interpreter = 'latex';
```



## Annotations

Annotations such as text and shapes can be added to emphasize important details.

```
>> 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';
```



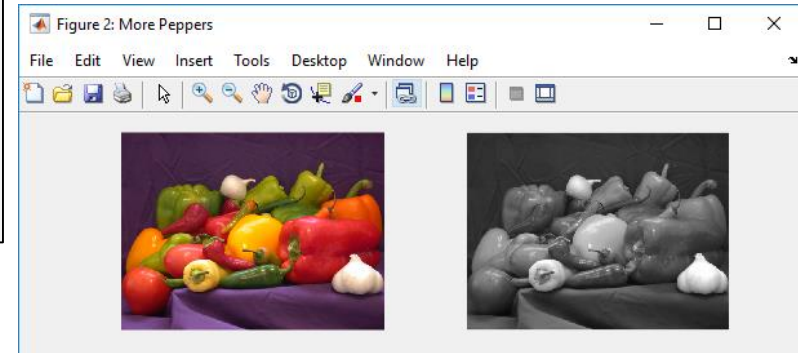
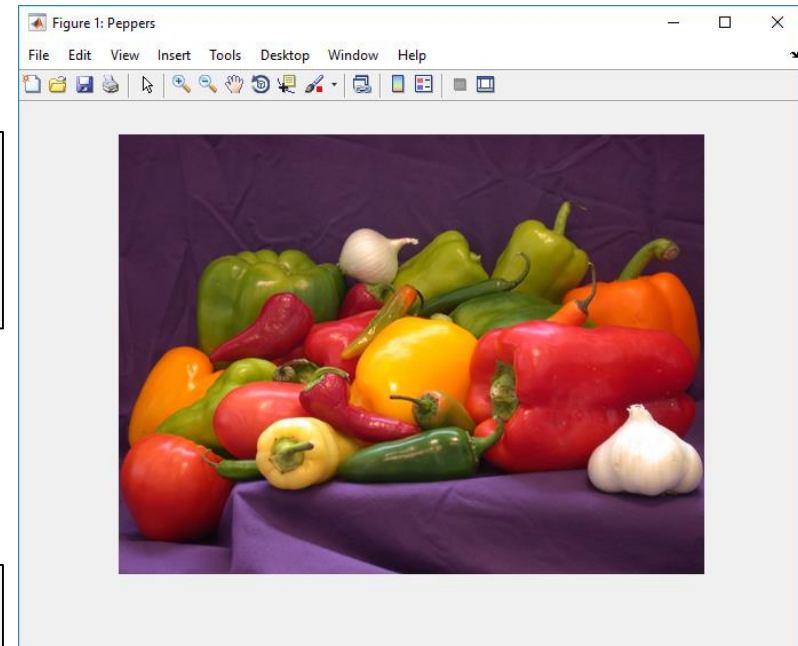
## Graphics - Images

- Images can be plotted using the `imshow` command

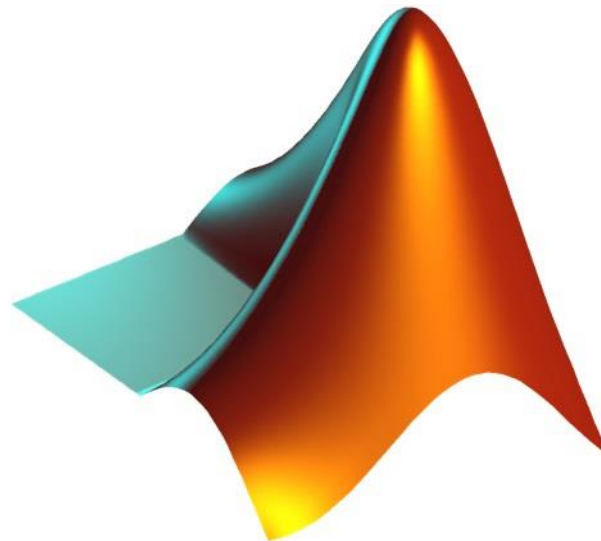
```
>> RGB = imread('peppers.png');  
  
>> h = figure('Name','Peppers');  
>> image = imshow(RGB);
```

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

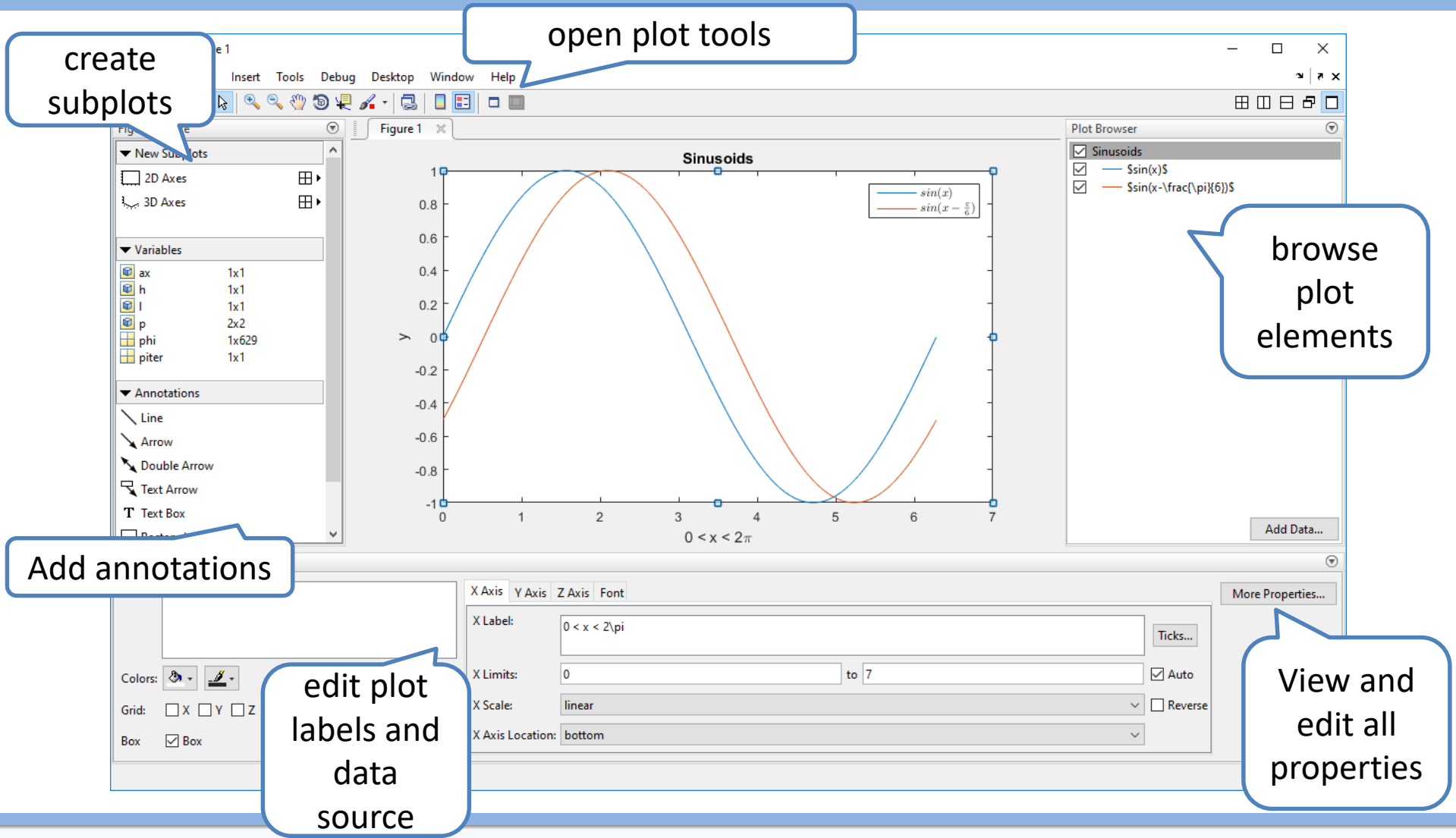
```
>> h = figure('Name','More Peppers');  
>> p = gobjects(2,1);  
>> for piter = 1:numel(p)  
    p(piter) = subplot(1,2,piter);  
end  
>> axes(p(1)); subimage(RGB); axis off  
>> axes(p(2)); subimage(rgb2gray(RGB)); axis off
```



## 4. Plot Tools



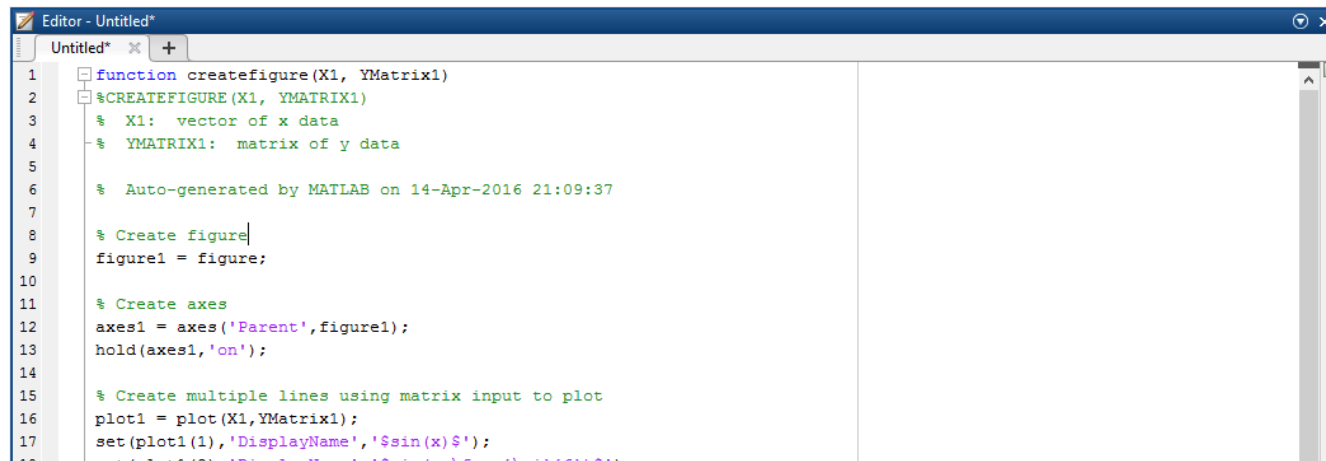
# Plot Tools



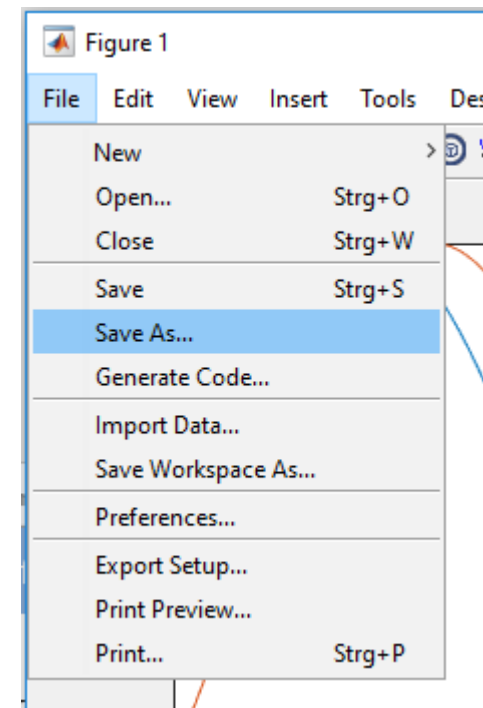
## Plot Tools – Save Figure / Generate Code

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



```
1 function createfigure(X1, YMatrix1)
2 %CREATEFIGURE(X1, YMATRIX1)
3 % X1: vector of x data
4 % YMATRIX1: matrix of y data
5
6 % Auto-generated by MATLAB on 14-Apr-2016 21:09:37
7
8 % Create figure
9 figure1 = figure;
10
11 % Create axes
12 axes1 = axes('Parent',figure1);
13 hold(axes1,'on');
14
15 % Create multiple lines using matrix input to plot
16 plot1 = plot(X1,YMatrix1);
17 set(plot1(1),'DisplayName','$sin(x)$');
18
```





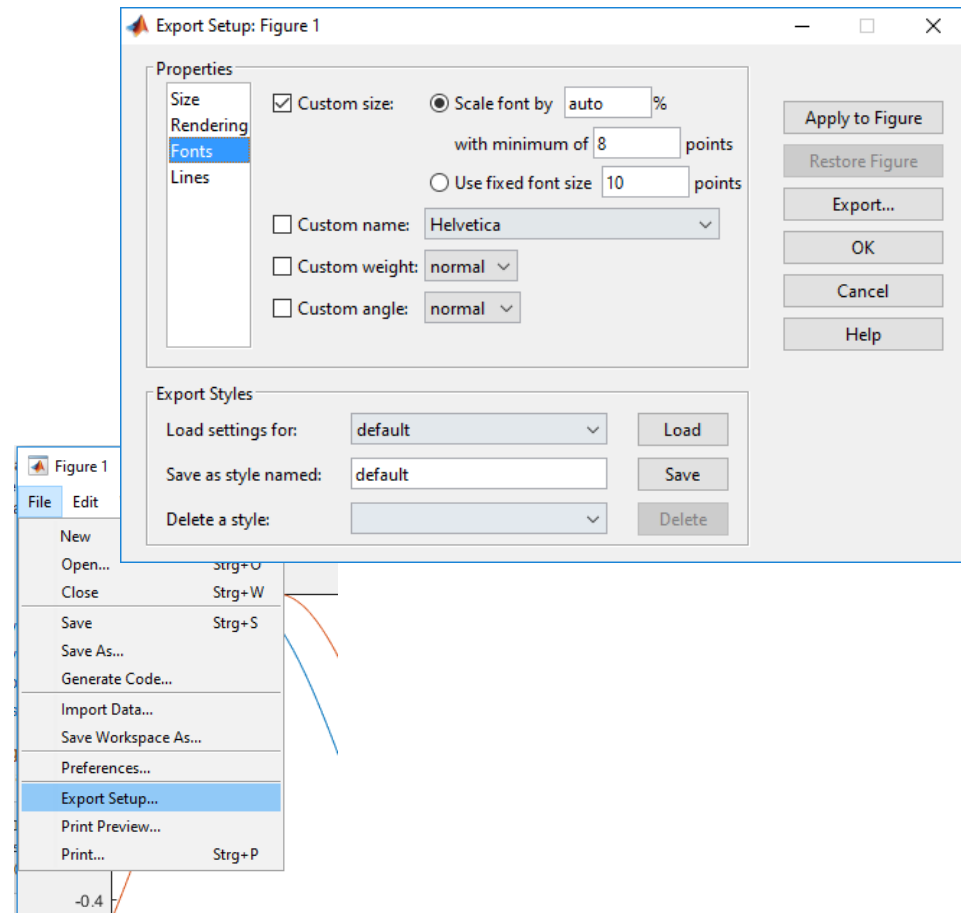
## Plot Tools – Export Figure to Presentation / Document

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

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

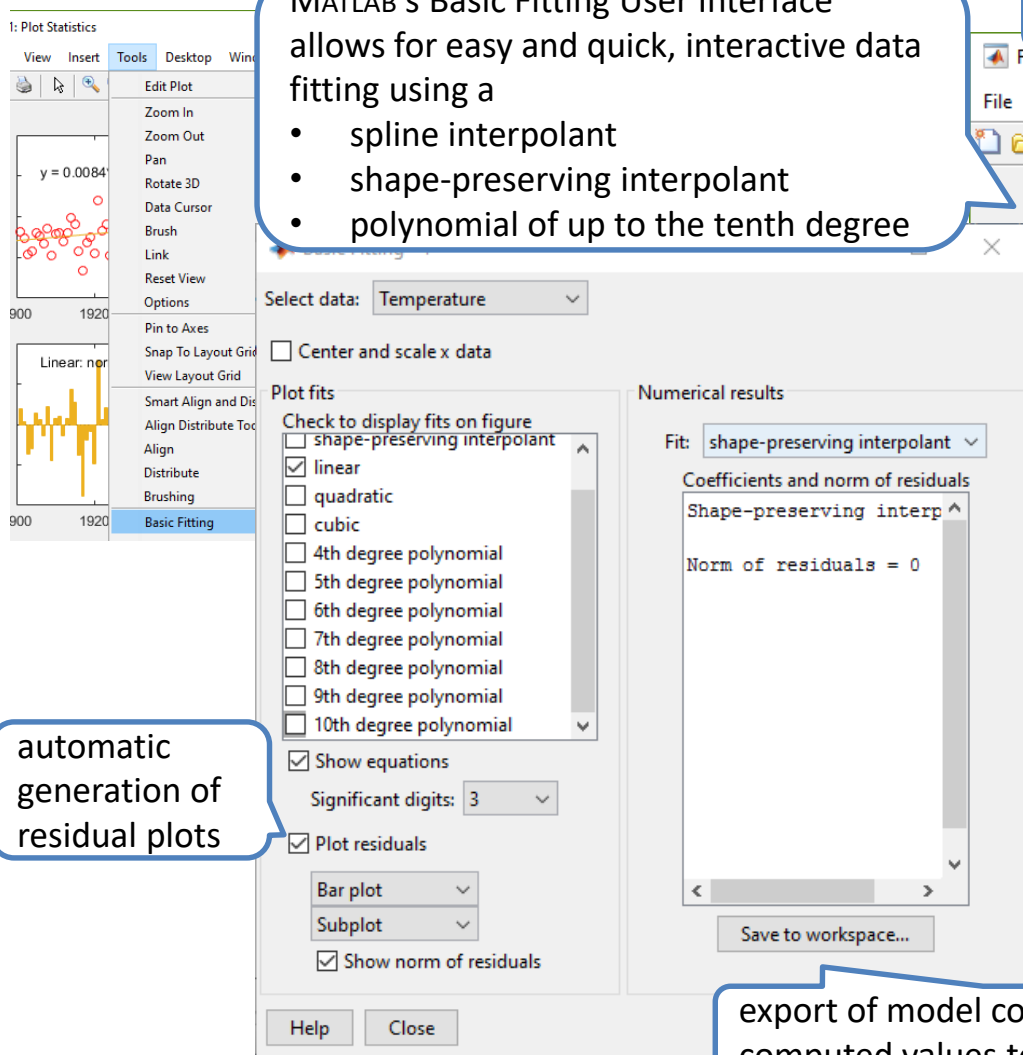


# Basic Fitting

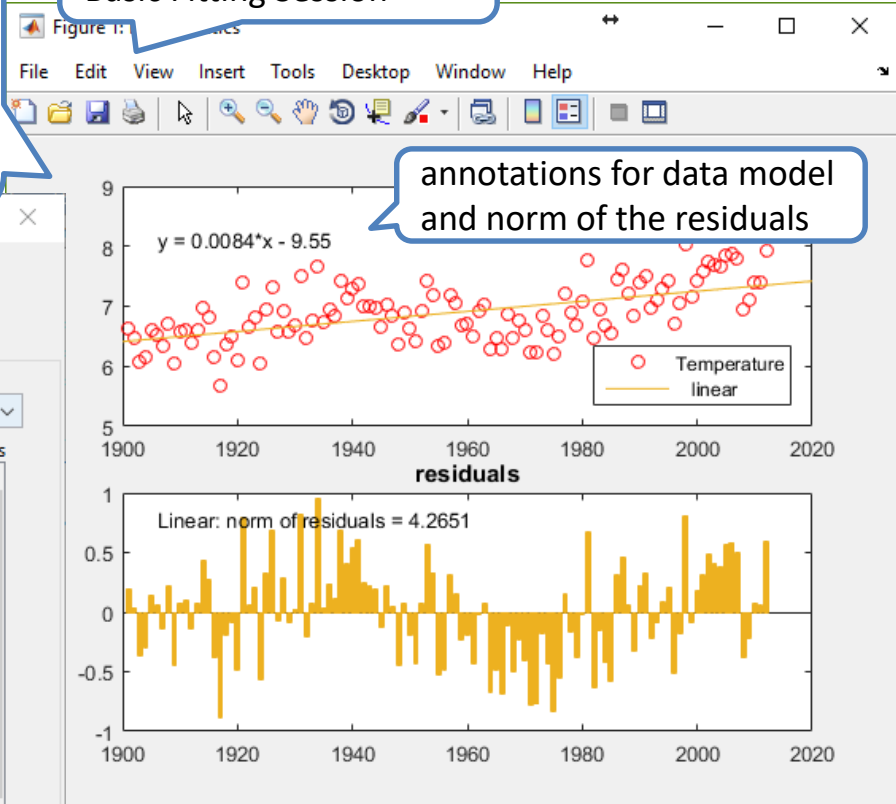
MATLAB's Basic Fitting User Interface allows for easy and quick, interactive data fitting using a

- spline interpolant
- shape-preserving interpolant
- polynomial of up to the tenth degree

code generation includes Basic Fitting Session



automatic generation of residual plots



annotations for data model and norm of the residuals

Basic Fitting is available for 2-D data (advanced fitting using MATLAB's Curve Fitting Toolbox)

export of model coefficients and computed values to MATLAB Workspace

# Data Statistics

I: Plot Statistics

View Insert Tools Desktop Window Help

- Edit Plot
- Zoom In
- Zoom Out
- Pan
- Rotate 3D
- Data Cursor
- Brush
- Link
- Reset View
- Options
- Pin to Axes
- Snap To Layout Grid
- View Layout Grid
- Smart Align and Distribute
- Align Distribute Tool ...
- Align
- Distribute
- Brushing
- Basic Fitting
- Data Statistics**

The MATLAB Data Statistics GUI allows for calculating and displaying descriptive statistics for plots such as

- min, max
- mean, median, mode
- std
- range (cannot be displayed)

Data Statistics - 1

Statistics for: Temperature

Check to plot statistics on figure:

	X		Y	
min	1901	<input checked="" type="checkbox"/>	5.671	<input type="checkbox"/>
max	2012	<input checked="" type="checkbox"/>	8.046	<input type="checkbox"/>
mean	1957	<input type="checkbox"/>	6.886	<input checked="" type="checkbox"/>
median	1957	<input type="checkbox"/>	6.845	<input type="checkbox"/>
mode	1901	<input type="checkbox"/>	5.671	<input type="checkbox"/>
std	32.48	<input type="checkbox"/>	0.4881	<input checked="" type="checkbox"/>
range	111		2.375	

Save to workspace... Help Close

export of descriptive statistics to MATLAB Workspace

Figure 1: Plot Statistics

Edit View Insert Tools Desktop Window Help

Temperature

1920 1940 1960 1980 2000 2020

8.5 8 7.5 7 6.5 6

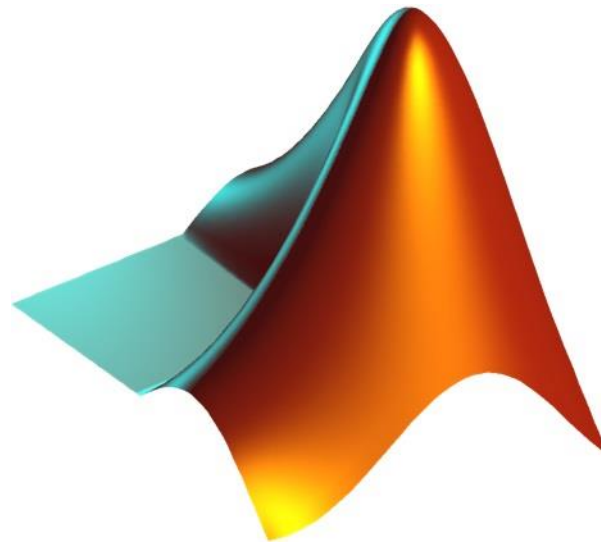
Legend:

- Temperature
- Begin of data
- x max
- Mean Temperature
- y std

Statistics can be displayed within the plot

Legend can be edited by double clicking the name of the statistics

## 5. List of Useful Commands



# List of Commands

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

# List of Commands

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