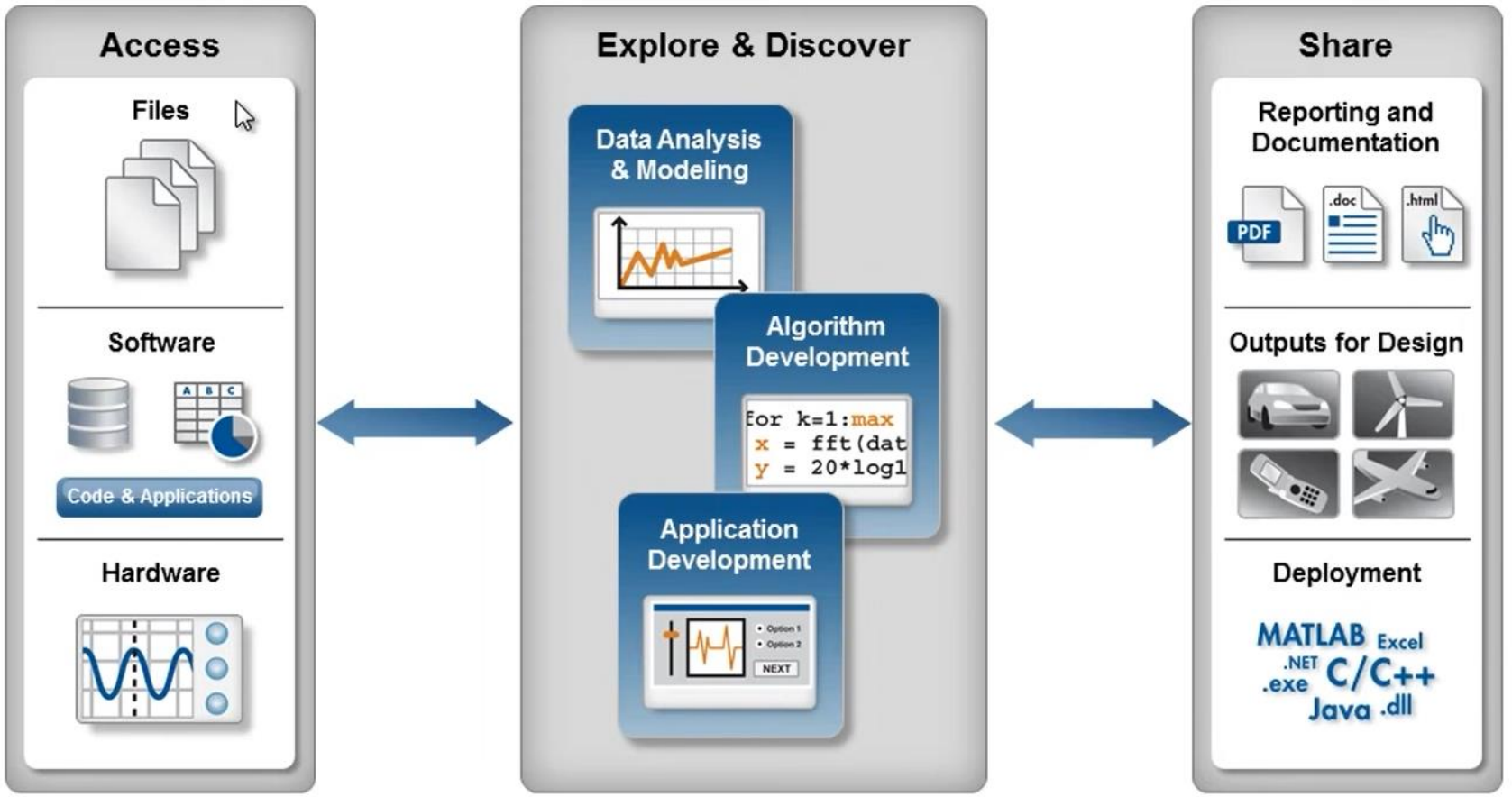
Introduction to MATLAB

Data In/Out

Advanced Graphics

Introduction



Introduction

- We often want to import data into Matlab
- This data can be from a variety of sources:
  - Spreadsheets
  - CSV files
  - Other text files
  - Movies (avi)
  - Images (bmp, gif, jpg, png, tiff, etc.)
  - Audio (wav, etc.)

File I/O functions

- You can load various data format into the workspace of MATLAB.
  - Some functions can load specified file formats.
  - *.txt / *.dat / *.bmp / *.au / *.wav / *.avi /*.gif / *.mat / *.xls /*.jpg etc

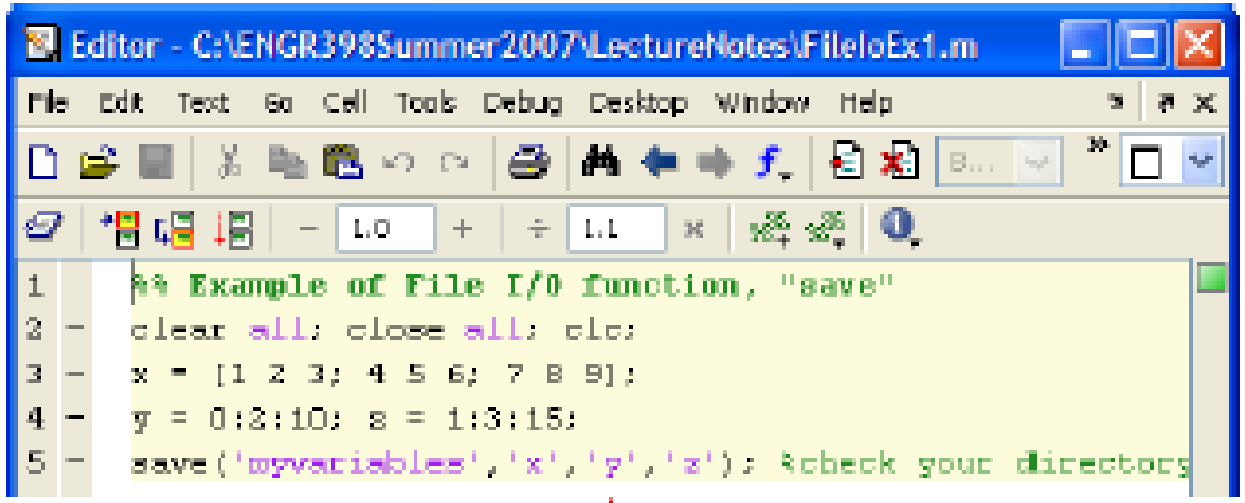
- Useful commands to read and/or write data

  -- “save”/”load”/”audioread”/”audiowrite”/ “imread” / “imwrite” / “fopen” / “xlsread” / “xlswrite” etc

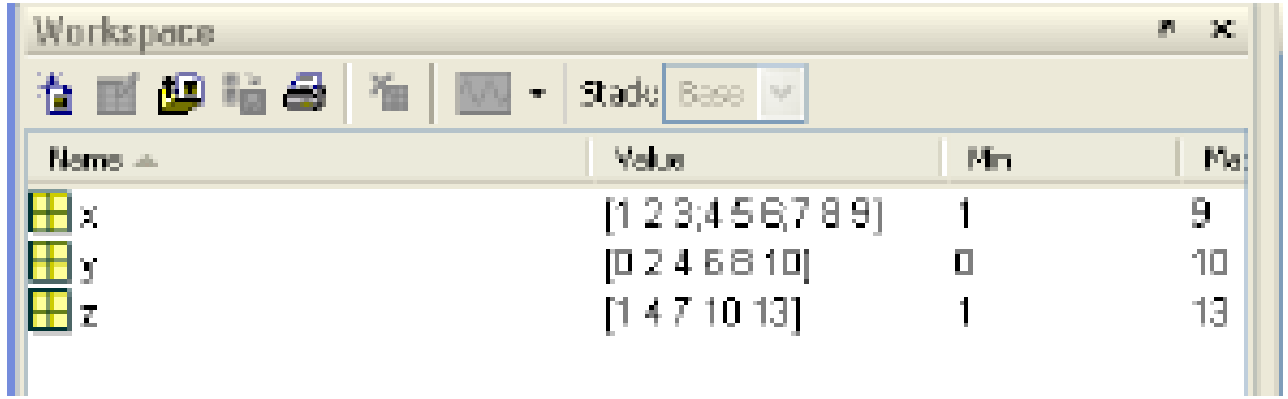
Save and Load *.mat files

- `save filename var1 var2 var3`
- `load filename`
- The save command saves data from the workspace into an `*.mat` file.
- The load command loads the variables stored in the `filename.mat` file previously saved.
- `filename` is the name of the file (a string) and `var1`, `var2`, `var3` are the variables to be saved in a MATLAB readable file.
- If the variables are not listed after the filename, save saves all the variables in the workspace.

Saving to MATLAB .mat File



After executing the .m script



Loading From a .mat File

Editor - C:\ENGR398Summer2007\LectureNotes\

File Edit Text Go Cell Tools Debug Desktop Window

```
1     %% File I/O function, 'load'
2 -   clear all; close all;
3 -   MySTR = 'ENGR320 labs';
4 -   whos
5 -   load('myvariables.mat');
6 -   whos %Check what variables are load
```

Workspace

Stack: Base

Name	Value	Min	Ma
MySTR	'ENGR320 labs'		
x	[1 2 3;4 5 6;7 8 9]	1	9
y	[0 2 4 6 8 10]	0	10
z	[1 4 7 10 13]	1	13

Command Window

```
>> FileIoEx2
  Name          Size                Bytes  Class     Attributes

  MySTR         1x12                   24  char


  Name          Size                Bytes  Class     Attributes

  MySTR         1x12                   24  char
  x             3x3                    72  double
  y             1x6                    48  double
  z             1x5                    40  double
```

After executing the "m-file"

Import Spreadsheet Data Using readtable

```
readtable('datafile for test.xlsx')
```

You can also select the range of data to import by specifying the range parameter. For example, read the first five rows and three columns of the spreadsheet.

```
A=readtable('datafile for test.xlsx','Range','A1:C5')
```

In addition to tables, you can import your spreadsheet data into the MATLAB workspace as a timetable, a numeric matrix, a cell array, or separate column vectors. Based on the data type you need, use one of these functions.

Data Type of Output	Function
Timetable	`readtimetable`
Numeric Matrix	`readmatrix`
Cell Array	`readcell`
Separate Column Vectors	`readvars`

Read Spreadsheet Data into Matrix

Import numeric data from `basic_matrix.xls` into a matrix.

```
M = readmatrix('basic_matrix.xls')
```

```
M = 5×4

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

You can also select the data to import from the spreadsheet by specifying the `Sheet` and `Range` parameters. For example, specify the `Sheet` parameter as `'Sheet1'` and the `Range` parameter as `'B1:D3'`. The `readmatrix` function reads a 3-by-3 subset of the data, starting at the element in the first row and second column of the sheet named `'Sheet1'`.

```
M = readmatrix('basic_matrix.xls','Sheet','Sheet1','Range','B1:D3')
```

```
M = 3×3

     8     3     1
     4     7     3
     6     7    10
```

Write Data to Excel Spreadsheets

Write Tabular Data to Spreadsheet File

For example, create a sample table of column-oriented data and display the first five rows.

```
load patients.mat
T = table(LastName,Age,Weight,Smoker);
T(1:5,:)
```

```
ans=5×4 table
      LastName      Age    Weight    Smoker
    ____________    ___    ______    ______

    {'Smith'   }    38      176      true
    {'Johnson' }    43      163      false
    {'Williams'}    38      131      false
    {'Jones'   }    40      133      false
    {'Brown'   }    49      119      false
```

Write table `T` to the first sheet in a new spreadsheet file named `patientdata.xlsx`.

```
filename = 'patientdata.xlsx';
writetable(T,filename,'Sheet',1,'Range','D1')
```

Write the table `T` without the variable names to a new sheet called `'MyNewSheet'`. To write the data without the variable names, specify the name-value pair `WriteVariableNames` as `false`.

```
writetable(T,filename,'Sheet','MyNewSheet','WriteVariableNames',false);
```

Write Numeric and Text Data to Spreadsheet File

To export a numeric array and a cell array to a Microsoft Excel spreadsheet file, use the `writematrix` or `writecell` functions. You can export data in individual numeric and text workspace variables to any worksheet in the file, and to any location within that worksheet. By default, the import functions write your matrix data to the first worksheet in the file, starting at cell `A1`.

For example, create a sample array of numeric data, `A`, and a sample cell array of text and numeric data, `C`.

```
A = magic(5)
C = {'Time', 'Temp'; 12 98; 13 'x'; 14 97}
```

```
A =

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9


C =

    'Time'    'Temp'
    [  12]    [  98]
    [  13]    'x'
    [  14]    [  97]
```

Write array `A` to the 5-by-5 rectangular region, `E1:I5`, on the first sheet in a new spreadsheet file named `testdata.xlsx`.

```
filename = 'testdata.xlsx';
writematrix(A,filename,'Sheet',1,'Range','E1:I5')
```

Write cell array `C` to a rectangular region that starts at cell `B2` on a worksheet named `Temperatures`. You can specify range using only the first cell.

```
writecell(C,filename,'Sheet','Temperatures','Range','B2');
```

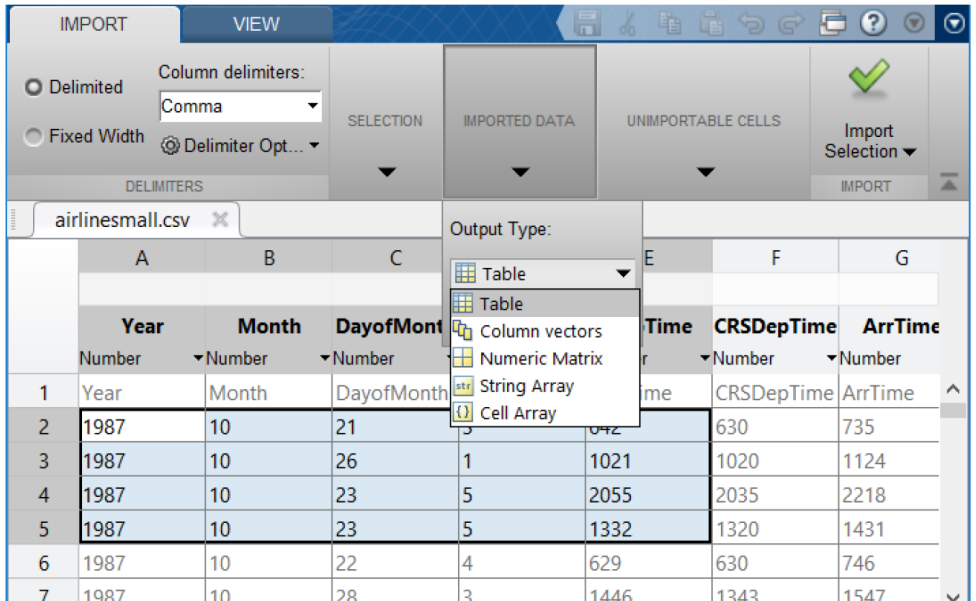
Import Text Files

Text files often contain a mix of numeric and text data as well as variable and row names, which is best represented in MATLAB as a table. You can import tabular data from text files into a table using the **Import Tool** or the `readtable` function.

Import Text Files Using the Import Tool

Import Text Files Using the Import Tool

The **Import Tool** allows you to import into a table or other data type. For example, read a subset of data from the sample file `airlinesmall.csv`. Open the file using the **Import Tool** and select options such as the range of data to import and the output type. Then, click on the **Import Selection** button  to import the data into the MATLAB workspace.



Import Text Files Using readtable

Alternatively, you can read tabular data from a text file into a table using the `readtable` function with the file name, for example:

```
T = readtable('airlinesmall.csv');
```

Display the first five rows and columns from the table.

```
T(1:5,1:5)
```

```
ans =


  5×5 table

    Year    Month    DayofMonth    DayOfWeek    DepTime
    ____    _____    __________    _________    _______

    1987      10         21           3        {'642' }
    1987      10         26           1        {'1021'}
    1987      10         23           5        {'2055'}
    1987      10         23           5        {'1332'}
    1987      10         22           4        {'629' }
```

Import Numeric Data from Text Files into Matrix

Import numeric data as MATLAB arrays from files stored as comma-separated or delimited text files.

Import Comma-Separated Data

This example shows how to import comma-separated numeric data from a text file. Create a sample file, read all the data in the file, and then read only a subset starting from a specified location.

Create a sample file named `ph.dat` that contains comma-separated data and display the contents of the file.

```
rng('default')
A = 0.9*randi(99,[3 4]);
writematrix(A,'ph.dat','Delimiter',',')
type('ph.dat')
```

```
72.9,81.9,25.2,86.4
81,56.7,49.5,14.4
11.7,9,85.5,87.3
```

Read the file using the `readmatrix` function. The function returns a `3`-by-`4` `double` array containing the data from the file.

```
M = readmatrix('ph.dat')
```

```
M = 3×4

   72.9000   81.9000   25.2000   86.4000
   81.0000   56.7000   49.5000   14.4000
   11.7000    9.0000   85.5000   87.3000
```

Import Delimited Numeric Data

This example shows how to import numeric data delimited by any single character using the `writematrix` function. Create a sample file, read the entire file, and then read a subset of the file starting at the specified location.

Create a tab-delimited file named `num.txt` that contains a 4-by-4 numeric array and display the contents of the file.

```
rng('default')
A = randi(99,[4,4]);
writematrix(A,'num.txt','Delimiter','\t')
type('num.txt')
```

```
81    63    95    95
90    10    96    49
13    28    16    80
91    55    97    15
```

Read the entire file. The `readmatrix` function determines the delimiter automatically and returns a 4-by-4 `double` array.

Write Data to Text Files

Export Table to Text File

You can export tabular data from MATLAB® workspace into a text file using the `writetable` function. Create a sample table, write the table to text file, and then write the table to text file with additional options.

Create a sample table, `T`, containing the variables `Pitch`, `Shape`, `Price` and `Stock`.

```
Pitch = [0.7;0.8;1;1.25;1.5];
Shape = {'Pan';'Round';'Button';'Pan';'Round'};
Price = [10.0;13.59;10.50;12.00;16.69];
Stock = [376;502;465;1091;562];
T = table(Pitch,Shape,Price,Stock)
```

```
T=5×4 table
    Pitch      Shape       Price    Stock
    _____    __________    _____    _____

     0.7     {'Pan'   }       10      376
     0.8     {'Round' }    13.59      502
       1     {'Button'}     10.5      465
    1.25     {'Pan'   }       12     1091
     1.5     {'Round' }    16.69      562
```

Export the table, `T`, to a text file named `tabledata.txt`. View the contents of the file. By default, `writetable` writes comma-separated data, includes table variable names as column headings.

```
writetable(T,'tabledata.txt');
type tabledata.txt
```

```
Pitch,Shape,Price,Stock
0.7,Pan,10,376
0.8,Round,13.59,502
1,Button,10.5,465
1.25,Pan,12,1091
1.5,Round,16.69,562
```

Export Numeric Array to Text File

You can export a numerical array to a text file using `writematrix`.

Create a numeric array A.

```
A = magic(5)/10
```

```
A = 5×5

    1.7000    2.4000    0.1000    0.8000    1.5000
    2.3000    0.5000    0.7000    1.4000    1.6000
    0.4000    0.6000    1.3000    2.0000    2.2000
    1.0000    1.2000    1.9000    2.1000    0.3000
    1.1000    1.8000    2.5000    0.2000    0.9000
```

Write the numeric array to `myData.dat` and specify the delimiter to be `';'`. Then, view the contents of the file.

```
writematrix(A,'myData.dat','Delimiter',';')
type myData.dat
```

```
1.7;2.4;0.1;0.8;1.5
2.3;0.5;0.7;1.4;1.6
0.4;0.6;1.3;2;2.2
1;1.2;1.9;2.1;0.3
1.1;1.8;2.5;0.2;0.9
```

In-class exercise

Load Sample_Text_data.txt into variables **a**, **b**, **c**, and **d**.

a. Do not suppress the command line output (“;”) so that your import shows up when you publish your script.
b. The import wizard is not an acceptable import method for this exercise.

Load Sample_Text_data.csv into variables **Date**, **Name**, and **Score**.

a. Do not suppress the command line output (“;”) so that your import shows up when you publish your script.
b. Note that the data is separated by commas (i.e. Comma Separated Variables or .csv).
c. The import wizard is not an acceptable import method for this exercise.

Write variables generated during the previous step (i.e. **Date**, **Name**, and **Score**) to an Excel files (you choose the filename).

Animation

- To animate a plot, simply generate a series of snapshots and then use "move" to show them
- Example, animate sin(x)*sin(2*pi*t/20)
- Get file ***anim.m***

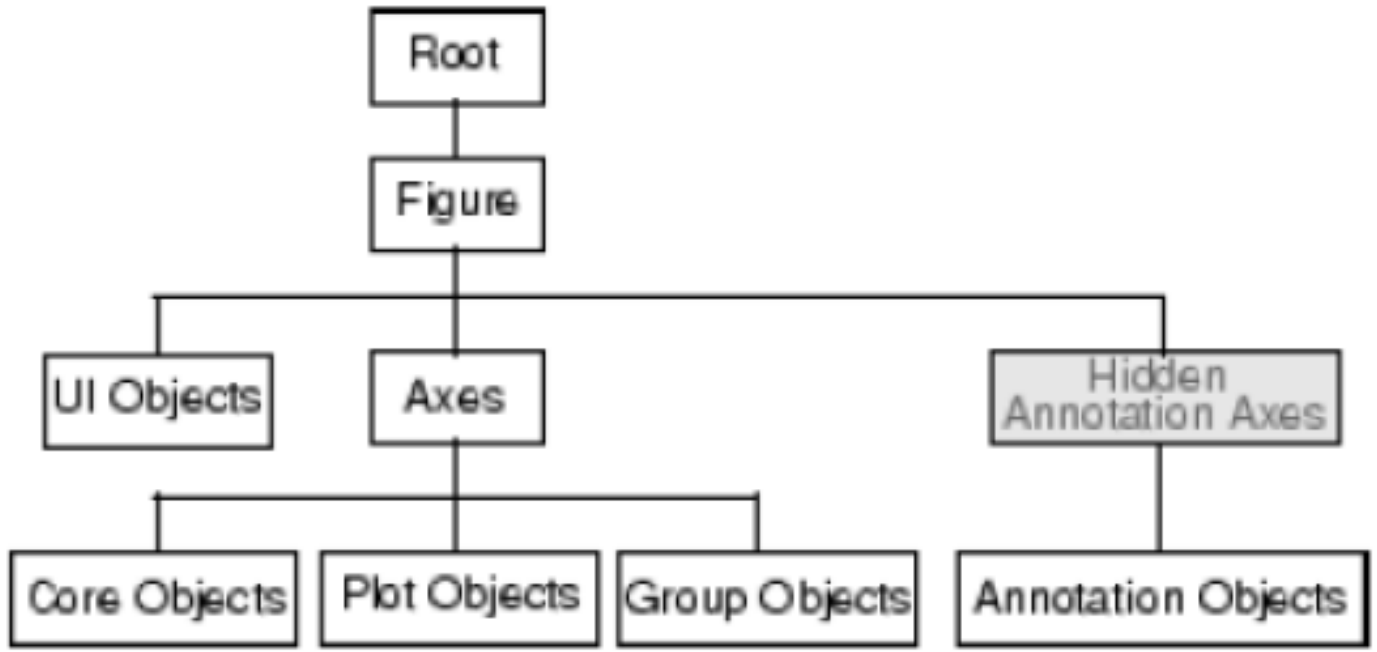
Animation Example

```
x=0:pi/100:2*pi;
y=sin(x);
plot(x,y)
axis tight
% Record the movie
for j = 1:20
    plot(x,sin(2*pi*j/20)*y)
    F(j) = getframe;
end
% Play the movie two times
movie(F,2)
```

Advanced Graphics in MATLAB

- "*Graphics Objects*" are the basic drawing elements used by MATLAB to display data.
- "*Graphics Handles*" are kind of "ID" to identify each graphic object such as figure, individual plot lines, surface, legend, text, etc.
- Using the handles, we can manipulate characteristics of each graphics object. This give us tremendous control of figure parameters.
- The graphic objects are arranged in "parent-children" relationship.

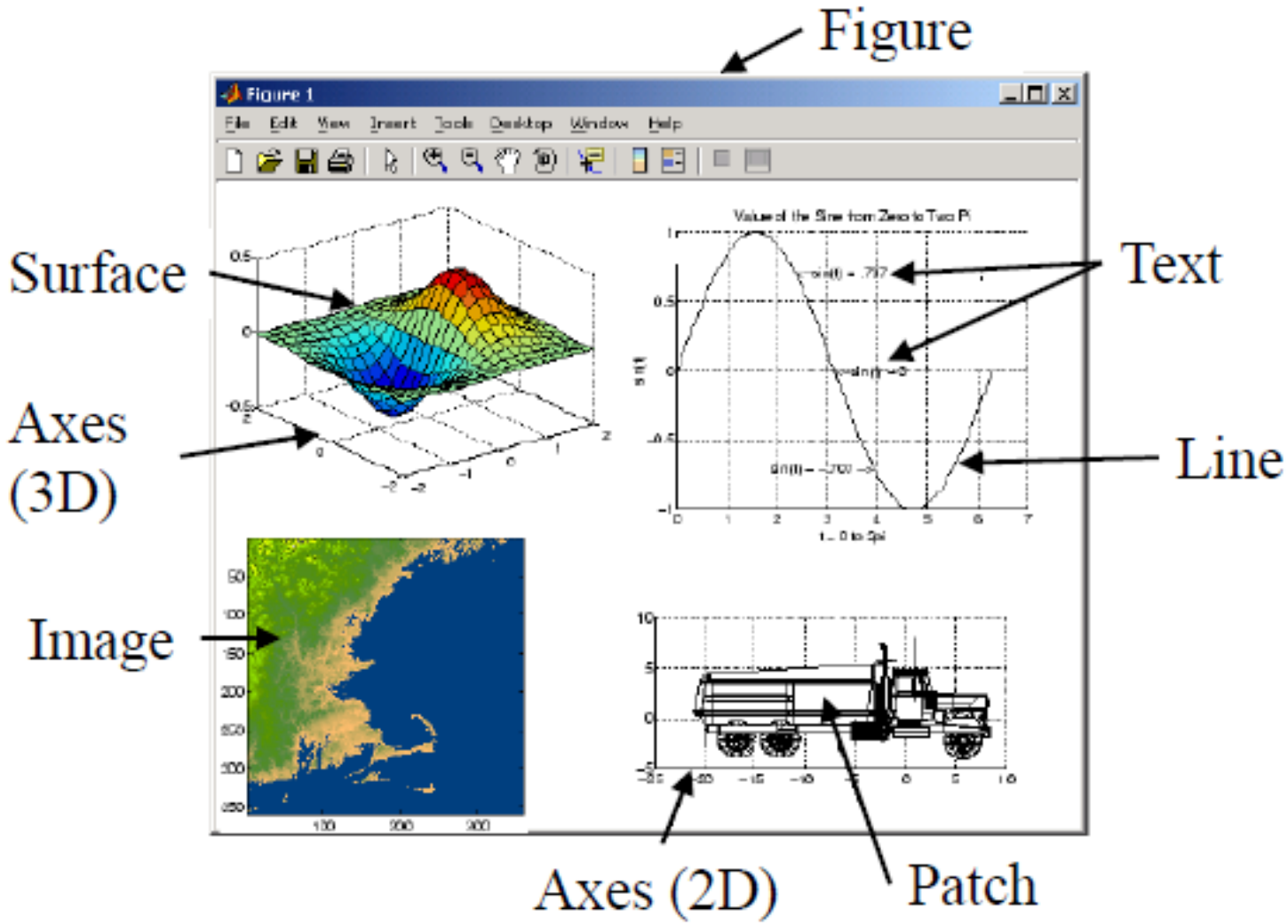
Hierarchy of Graphics Objects



Note: 1. Axes object is a child of Figure object and so on..

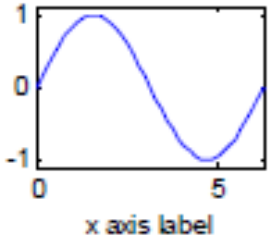
2. Line, Surface, Text objects are children of an Axes object

Graphics Objects



Graphics Handles

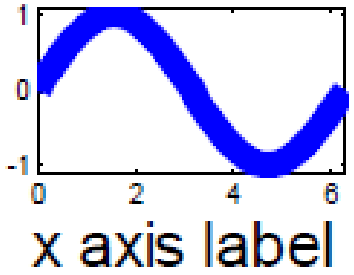
- All graphic objects have a handle that can be used to modify them
- For example:




```
>> x = 0:pi/20:2*pi;
>> hsin = plot(x,sin(x)) %Plots and returns a handle
>> hx = xlabel('x axis label') %Returns xlabel handle
```

- Using its handle, object properties can be modified

```
>> set(hsin, 'LineWidth', 10); % Increase line width
>> set(hx,'FontSize', 24) % Change Font Size of xlabel
```



Finding Available Graphics Properties

- To get a complete list of an object's properties, use the **`get( )`** function along with the object's handle.
- For example **`get(hsin)`** returns over 30 properties for the line object that has the handle **`hsin`** as shown.
- Notice that one of them is **`Color: [0 0 1]`**
- **`set( )`** can then be used to modify the line's color

```
>> set (hsin, 'color', [1   0   0  ]) % red
>> set (hsin, 'color', [0   1   0  ]) % green
>> set (hsin, 'color', [0   0   1  ]) % blue
>> set (hsin, 'color', [0.5 0.5 0.5]) % grey
```

etc.

- Note: the three elements in the color array define the ratio of red, green, and blue (a.k.a. RGB)

```
    DisplayName: ''
     Annotation: [1x1 hg.Annotation]
          Color: [0 0 1]
      LineStyle: '-'
      LineWidth: 10
         Marker: 'none'
     MarkerSize: 6
 MarkerEdgeColor: 'auto'
 MarkerFaceColor: 'none'
          XData: [1x41 double]
          YData: [1x41 double]
          ZData: [1x0 double]
    BeingDeleted: 'off'
  ButtonDownFcn: []
       Children: [0x1 double]
       Clipping: 'on'
       CreateFcn: []
       DeleteFcn: []
     BusyAction: 'queue'
 HandleVisibility: 'on'
        HitTest: 'on'
  Interruptible: 'on'
       Selected: 'off'
 SelectionHighlight: 'on'
            Tag: ''
           Type: 'line'
  UIContextMenu: []
       UserData: []
        Visible: 'on'
         Parent: 173.0503
    XDataMode: 'manual'
  XDataSource: ''
  YDataSource: ''
  ZDataSource: ''
```

Useful Functions to Get Handles

- If a handle is not known, here are a few functions that can be used to get it
  - **`gcf`** gets the handle of the current figure
  - **`gca`** gets the handle of the current axes
  - **`gco`** gets the handle of the current graphics object
  - **`gcbo`** gets the handle of object whose callback is executing
- For example, **`ha = gca`**, gets the handle for the current axis and assigns it to the variable **`ha`**.

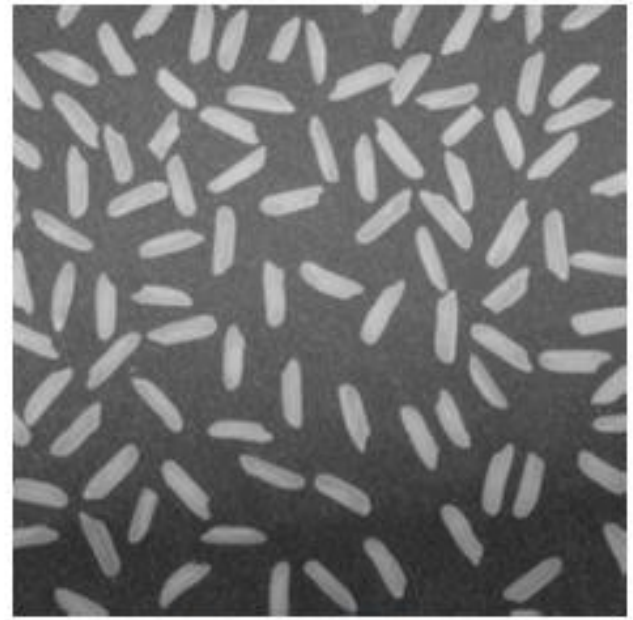
MATLAB and Images

- MATLAB has many tools for importing, viewing, and manipulating images
- Possible applications
  - Custom image processing scripts to resize, adjust contrast, crop, etc.
  - Extract still image from a video

MATLAB Image Processing Applications

Image Creation/Manipulation

Collages, etc.



Machine Vision

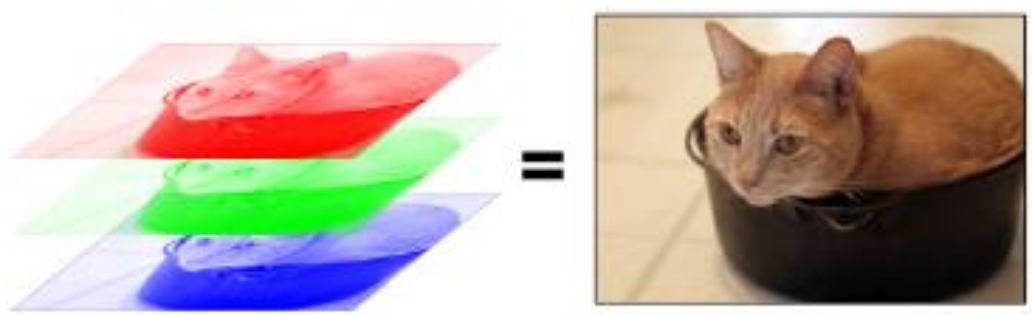
1. Counting
2. Measurements (size, area, color)
3. Defect Identification

Image Basics

- In the MATLAB workspace, most images are represented as two-dimensional arrays (matrices), in which each element of the matrix corresponds to a single pixel in the displayed image.
- Thus, MATLAB can manipulate images at the pixel level

RGB Images

- RGB images are 3D arrays composed of a stack of three 2D arrays (layers)
- The arrays (layers) are a maps of red, green, and blue intensities that, when combined, describe the final image



Supported Image Formats

- JPEG (Joint Photographic Experts Group)
- GIF (Graphics Interchange Files)
- PNG (Portable Network Graphics)
- TIFF (Tagged Image File Format)
- BMP (Microsoft® Windows® Bitmap)
- PCX (Paintbrush)
- Others

Basic Image Functions

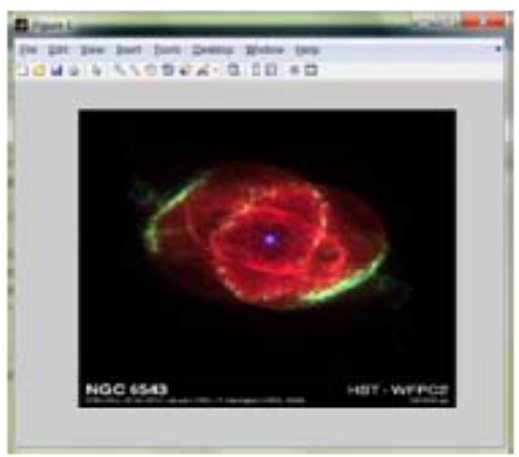
Function	Description
`image( )`	Display image object
`imshow( )`	Display image object or image file
`imread( )`	Read image from graphics file
`imwrite( )`	Write image to graphics file
`imfinfo( )`	Information about graphics file
`imagesc( )`	Scale data and display image object
`frame2im( )`	Return image data associated with movie frame

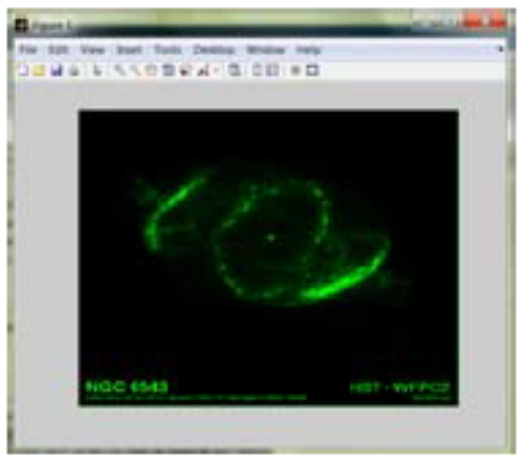
Image Example

```
A = imread('ngc6543a.jpg'); % Create image object
"A" from file
image(A); % Display image in figure
axis off; % Turn off axes
greens = A;
greens(:,:,[1 3]) = 0; % Set red (1) and blue (3)
intensities to zeros, leaving green (2) alone
image(greens) % Display green layer
imwrite(greens,'galaxy.jpg')
axis equal
```

All pixel columns

All pixel rows





MATLAB and Animation/Videos

- MATLAB has many tools for importing, viewing, and manipulating videos
- Possible applications
  - Create custom animations for presentations
  - Import images and stitch them together to create a movie or animated gif
  - Extract a single image from a video
  - Anything else you can dream up…

Animation and Movies Methods

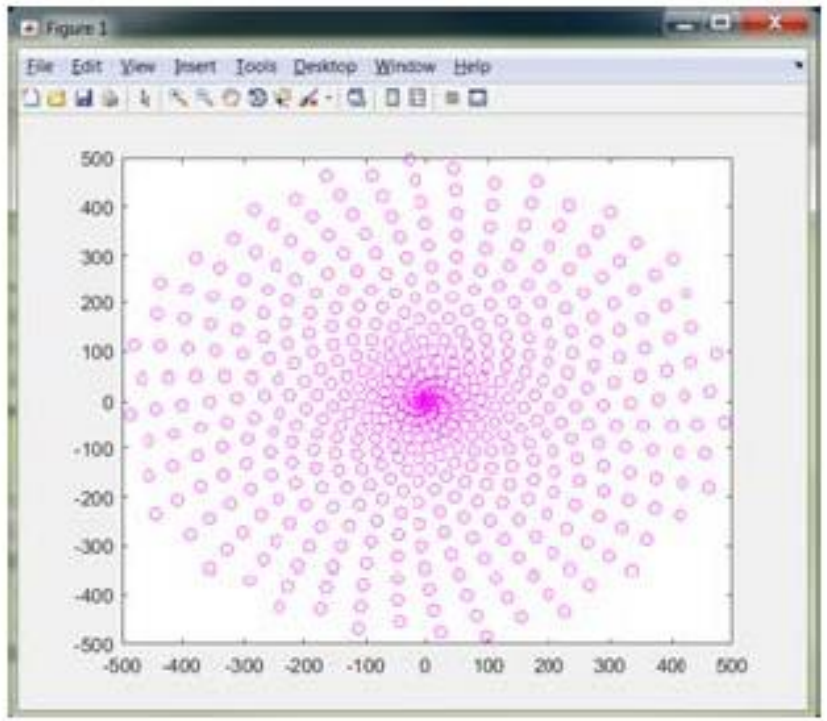
- Create a script using any of the 2D or 3D plot functions and the **`pause( )`** command
  - Can only "view/play" animation while inside MATLAB
- Create a movie object from a collection of images/plots
  - Can view/play in MATLAB AND export video file

Basic Animation Functions

Function	Description
`movie( )`	Play recorded movie frames
`getframe( )`	Capture movie frame from figure
`im2frame( )`	Convert image to movie frame
`VideoWriter( )`	Write videos to a file (avi, mpg, etc)
`VideoReader( )`	Import video file into MATLAB
`frame2im( )`	Return image data associated with movie frame
`pause(n)`	Pause for n seconds before continuing

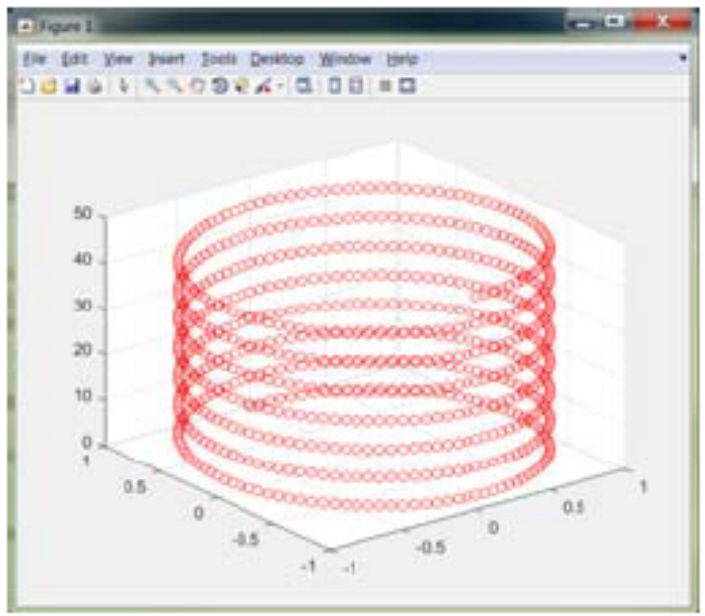
Animation Example with `pause( )`

```
clear all; close all; clc;
for t = 0:1:500
    x = t.*cos(t);
    y = t.*sin(t);
    plot(x,y,'mo')

    % keep previous plot point
    hold on

    % freeze axis size for
    % smooth animation
    axis([-500 500 -500 500]);

    % set time between frames
    % (limited by computer
    % capabilities)
    pause(0.005)
end
```



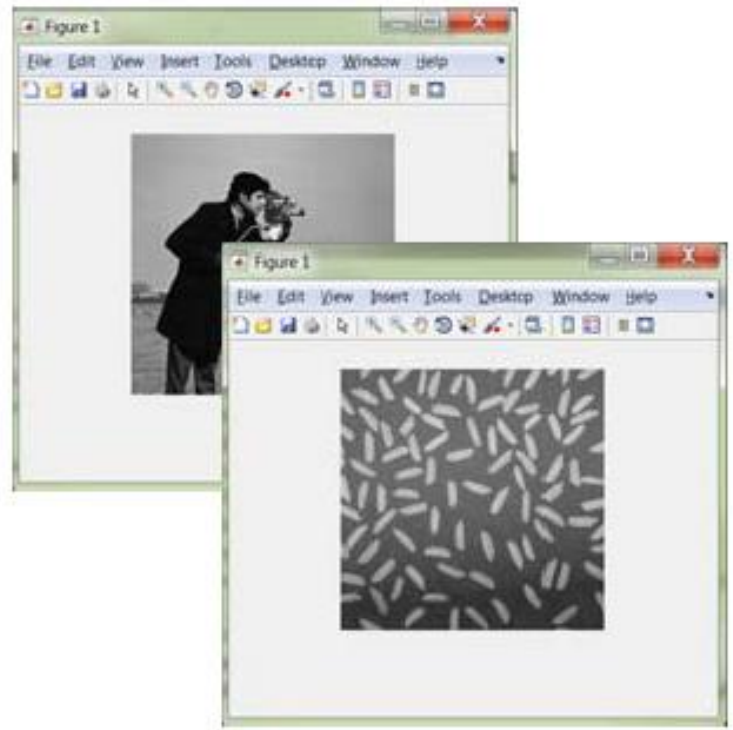
Animation Example with `pause( )`

```
%% Animation using plot3( ) and pause( )
clear all; close all; clc;
for t = 0:2*pi/100:15*pi
    x1 = sin(t);
    x2 = cos(t);
    plot3(x1,x2,t,'ro')
    hold on;  %turn this off and observe
    %keep axis size the same for each frame
    axis([-1 1 -1 1 0 50])
    axis off %optional
    grid on;
    %set time in seconds between frames
    %(limited by computer capabilities)
    pause(0.005) %modify to change animation speed
end
```



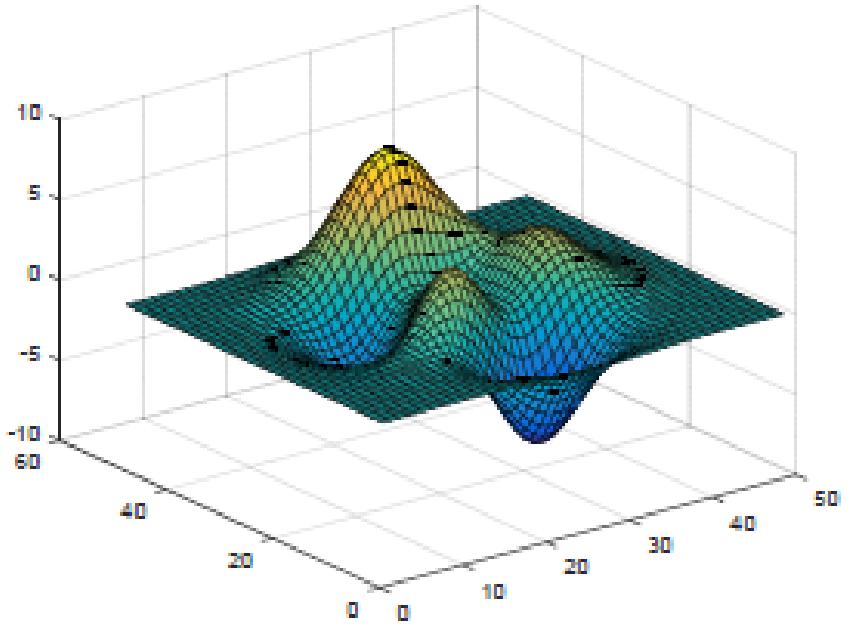
Animation Example with a Movie Object

```
clear all;close all;clc;
%display an image in current figure
imshow('cameraman.tif');
%convert the current figure to movie frame
M(1) = getframe; %add frame to movie object M
imshow('rice.png'); %disp a different image in current fig
M(2) = getframe; %add another frame to M
movie(M,5,2); %play movie M at 5 times at 2 frames per second (fps)
```



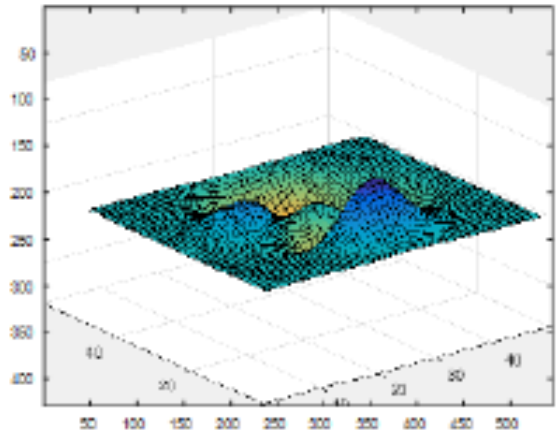
Animation Example with a Movie Object

```
clear all;close all;clc;
Z = peaks; %load data
surf(Z); %visualize data

% Initialize frame number
frameNum = 1;

for j = 1:20
  % Plot data in current figure
  surf(.01+sin(2*pi*j/20)*Z,Z);
  axis([0 50 0 60 -8 8])

  % Use current fig to create
  % a frame in movie object F
  F(frameNum) = getframe;

  % Increment frame number
  frameNum = frameNum + 1;
end
```



Extract Animation Frame and Play Movie Inside MATLAB

```
%% plot individual frame
close all;
%display frame 18 of movie object F
image(F(18).cdata)

%% play movie inside MATLAB
close all;
n = 5;    % number of times movie is repeated
fps = 6; % frames per second
         % (limited by cptr capabilities)
movie(F,n,fps)  %Play movie inside MATLAB
done = 1 % alert user when complete
```



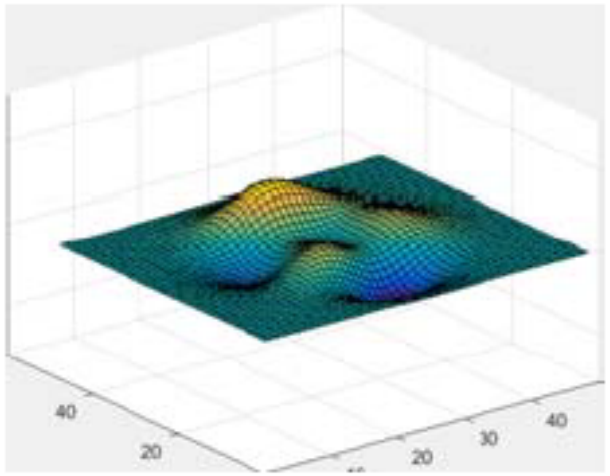
Note: Movie object **F** needs to have been created beforehand

Writing Movie Object to .avi File

```
% Create VideoWriter object
writerObj = VideoWriter('Animation_Example.avi');

open(writerObj) % Open VideoWriter Object

% Loop through all frames of movie object F
for k = 1:length(F)
    % Write single frame from movie object F
    % to VideoWriter Object
    writeVideo(writerObj,F(k))
end
close(writerObj) %Close VideoWriter object
avidone = 1 %alert user when process is done
```

Writing Movie Object to an Animated .gif file

```
filename = 'Animation_Example.gif';

% Loop through all frames of movie object F
for k = 1:length(F)
    % Extract image data of one frame of F
    im = frame2im(F(k));
    % Convert RGB image to indexed image
    [imind,cm] = rgb2ind(im,256);
    if k == 1; % Do this for the first frame
        % Write image to file
        imwrite(imind,cm,filename,'gif', 'Loopcount',inf);
    else % Do this for all the rest of the frames
        % Write image to file
        imwrite(imind,cm,filename,'gif','WriteMode','append');
    end
end

gifdone = 1 %alert user when gif has been written
```



Create a script to do the following:

1. Animation using **`plot( )`** and **`pause( )`** functions
    a. Create an x vector from 0 to 360 degrees with a stepsize of 10

    b. Create a y vector that is the sin of x. (Hint: use sind() to calculate sine in degrees)

    c. Initialize theta to be 0.

    d. Create a while-loop to do the following for 2 dance cycles:
        i. Create a y_plot vector that equals y times sind(theta)
        ii. Plot the x and y_plot data
        iii. Add your name to the plot
        iv. Make sure the axis doesn't resize every time
        v. Have MATLAB with for 0.1 seconds before continuing
        vi. Increment theta by 10

2. Animation using movie objects
    a. Create a movie object of the dancing sine wave for 2 dance cycles (you can create a new loop or insert commands into the loop you made above). Be sure to include your name somewhere on the plot.

    b. Create an avi using all the frame from your movie object.

    c. Create an animated gif of **ONLY THE FIRST 10 FRAMES** of your movie. If you do all the frames, your gif will be REALLY SLOW.