MATLAB Basics

Based on lecture notes from Prof. Sara Wilson, University of Kansas, Course: ME 208, Introduction to Digital Computational Methods in Mechanical Engineering
What is MATLAB?

• An application software package for numerical computation and visualization
• MATrix LABoratory
• Originated as a language to easily program in matrices (much easier to do matrices in than C)
• Evolved to do many tasks. Other programs have evolved towards MATLAB such as Maple, Labview and Mathematica but MATLAB remains the most effective.
Workspaces

- Command Window – for typing in commands
- Workspace – lists variables in the memory
- Command History
- Current Directory – Where things will be saved to/ opened from
- Graphics Window – opens when called for plots and figures
- Edit Window – window for writing and saving programs, must be called
This is a Classroom License for instructional use only. Research and commercial use is prohibited.

To get started, select "MATLAB Help" from the Help menu.
Edit Window

Figure Window
File Types

• M-files – Matlab Code (**.m) (in ascii format)
• Mat files – Special Matlab format used to efficiently (small space) save data (**.mat)
• Mex files – Matlab callable C or FORTRAN programs (**.mex)
• ASCII files – another way of saving matlab data (generally I label these as **.txt)
Script files (*.m)

- script files are independent text files denoted by .m that allow the running of several lines of matlab code

- comments are started with the symbol %
- they can be called if the path is correct by typing the name of the file (minus .m)
- don’t use any reserved words for the file name
Function files (*.m)

- Function files are similar to script functions except that the only data that is passed in or out of the workspace is designated

- first line of the file:
  function [output1, output2] = nameoffunction(input1,input2);

- to call:
  [output1, output2] = nameoffunction(input1,input2);
The path

- All functions, script, and files are opened from the current path except when specified in the call
- Default path \matlab\work
- you will want to change them to your personal workplace
Command Window Versus Script File

• Command Window
  – can run command immediately
  – works like a calculator
  – comments are not useful here as this is a one time shot
  – once command is run it is in command history but is otherwise gone (no saving a program of commands)

• Script File
  – opens with new window button or by typing “edit ***.m”
  – can save commands and rerun them
  – can be run by copying content to command window or by typing (saved) file name (minus .m)
  – can use ginput and other input functions
  – should be used in homeworks
Basic Commands

- help – help topics
- help topic – help on a specific topic
- who – list variables in workspace
- whos – who + variable size
- clear – clears workspace
- clear x y – clear the variables x and y
- clf – clear figure
- pwd – show current directory
- cd – change directory
- dir or ls – list what is in directory
- mkdir – make directory
- clock – gets time
- date – gets date
- quit or exit – quits Matlab
Math Operations

+ (add), - (subtract), * (times), / (divide), ^ (power)

```plaintext
>> 2*10
ans = 20

>> a = 2*10
a = 20

>> a= 2*10;
(a=20) not displayed when line ends with ;

a is floating point number

>>a = ‘2’ a is the string (character) 2
```
Basic Math Functions

- trig functions: \( \sin(x) \), \( \cos(y) \), \( \tan(z) \), \( \arccos(x) \), \( \arcsin(x) \), \( \arctan(x) \), \( \cosh(x) \), \( \sinh(x) \), \( \csc(x) \), \( \sec(x) \), … (in radians)
- logarithm:
  - \( \exp(x) = e^x \)
  - \( \log(x) = \log_e(x) \)
  - \( \log_{10}(x) = \log_{10}(x) \)
  - \( \log_2(x) \)
  - \( \sqrt{x} \)
- complex number
  - \( i \) or \( j \) - reserved for \( \sqrt{-1} \) unless otherwise designated
- rounding
  - \( \text{fix}(x) \) – round towards zero
  - \( \text{floor} \) – round towards \( -\infty \)
  - \( \text{ceil} \) – round towards \( +\infty \)
  - \( \text{round} \) – round to nearest integer
Variables

• Most variable types are available (integer, float, strings, characters, …)
• names of variables
  – case sensitive
  – only the first 31 letters count
  – no spaces, you can use underscore
  – must start with a letter (a,b,c,…)
  – you should avoid reserved words (for, end, if, while, function, return, elseif, else, case, switch, continue, try, catch, global, persistent, break, otherwise, and …)
Operating Systems

- Windows 95/98/2000/NT/XP/ME
- Mac – only to version 5
- Unix/ Linux etc.

- originally designed for Unix, used Unix type commands (cd, ls, mkdir)
Array operations

• \( x = [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10]; \) 10 columns
• \( x = [1;2;3;4;5;6;7;8;9;10]; \) 10 rows
• \( A = [1 \ 2; 3 \ 4]; \)

\[
A = \begin{pmatrix}
1 & 2 \\
3 & 4
\end{pmatrix}
\]
Math operations on elements of an array

.*   ./   .^  Note period before operation symbol

\[ x = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}; \]

\[ x.*x = \begin{bmatrix} 1 & 4 & 9 \end{bmatrix} = \begin{bmatrix} 1*1 & 2*2 & 3*3 \end{bmatrix} \]
\[ x./x = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1/1 & 2/2 & 3/3 \end{bmatrix} \]
\[ x.^x = \begin{bmatrix} 1 & 4 & 27 \end{bmatrix} = \begin{bmatrix} 1 & 2^2 & 3^3 \end{bmatrix} \]
Math operations on arrays

\[
\begin{align*}
\text{x} &= [1 \ 2 \ 3]; \\
\text{y} &= [1;2;3]; \\
\text{A} &= [1 \ 1; \ 1 \ 1]; \\
\text{x*y} &= 14 \\
\text{A/A} &= [\text{Inf Inf}; \text{Inf Inf}] = \text{A*inv(A)} \\
\text{A^2} &= [2 \ 2; \ 2 \ 2] = \text{A*A}
\end{align*}
\]
Trig and Elementary Math Functions with Arrays

- Trig and elementary math functions (sin, cos, exp, log) operate on each term of an array or matrix individually

\[ x = \text{asin}([1 0 1]) = [1.57 0 1.57] \]
\[ x = \text{exp}([0 1 2]) = [1 2.7 7.4] \]
Creating Arrays

- `linspace(a, b, c);` creates an array (row vector) from a to b with c elements linearly or logarithmically spaced.

- `logspace(a, b, c);` does the same but spaced logarithmically.

- `ones(m, n);` creates an mxn matrix of the value one.

- `zeros(m, n);` creates an mxn matrix of the value zero.
Plotting

• figure - generates a new figure window
• plot - creates a two-dimensional plot

axis([a b c d]) - changes x-axis range from a to b, and y-axis from c to d

xlabel('Label')
ylabel('Label')
title('Title')
print

labels axes and titles, more sophisticated arguments are possible
print hardcopy to printer
Graphics Tips

• Multiple plots
  To keep plots on separate windows, assign each to a figure number and call figure(#) before any plot command. This will make sure you don’t throw two plots to the same window.

  figure(1)
  plot3(dx,dy,dz)
  figure(2)
  surf(X,Y,Z)
Graphics Tips

- subplots
  Using subplots is a lot like multiple figure windows. Call the subplot(row,column,#) before calling the plot command. Make sure the subplots have the same numbers of rows and columns.

  subplot(2,1,1)
  plot3(dx,dy,dz)
  subplot(2,1,2)
  surf(X,Y,Z)
plot commands

• In plot commands, like array operations, the row and columns have to line up

\[
\text{plot}(x,y)
\]\nwith \( x = [1 \ 2 \ 3 \ 4 \ 5] \quad y = [1 \ 2 \ 3 \ 4 \ 5 \ 6] \) won’t work

• Any math operations within the plot command should work as well as standing alone

ie. \( \text{plot}(x+dx,y) \) will only work if \( x+dx \) works
\begin{verbatim}
plot(x,y)
plot(x,y,x2,y2)
plot(x,y,'c+:')
plot(x,y,'y-',x,y,'go')
\end{verbatim}

<table>
<thead>
<tr>
<th>Color</th>
<th>Symbol</th>
<th>Shape</th>
<th>Description</th>
<th>Line Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>blue</td>
<td>.</td>
<td>point</td>
<td>solid</td>
</tr>
<tr>
<td>g</td>
<td>green</td>
<td>o</td>
<td>circle</td>
<td>dotted</td>
</tr>
<tr>
<td>r</td>
<td>red</td>
<td>x</td>
<td>x-mark</td>
<td>dashdot</td>
</tr>
<tr>
<td>c</td>
<td>cyan</td>
<td>+</td>
<td>plus</td>
<td>dashed</td>
</tr>
<tr>
<td>m</td>
<td>magenta</td>
<td>*</td>
<td>star</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>yellow</td>
<td>s</td>
<td>square</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>black</td>
<td>d</td>
<td>diamond</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>v</td>
<td>triangle (down)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>^</td>
<td>triangle (up)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;</td>
<td>triangle (left)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt;</td>
<td>triangle (right)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>p</td>
<td>pentagram</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>h</td>
<td>hexagram</td>
<td></td>
</tr>
</tbody>
</table>

After plot you can use the following commands to edit the plot:

```matlab
xlabel('Time (s)', 'fontsize', 16)
    labels the x-axis with the label in fontsize 16
ylabel('Flexion (deg)')
    labels the y-axis
title('Fred')
    gives the plot a title
text(1, -30,'help?')
    puts text at the point 1,-30 on the graph
```
• You can also edit the graph using the property editor in MATLAB version 7
• Property editors can be found under the edit menu of the plot or by double clicking a feature after first selecting Tools> Edit plot
• If you want to add more data to an existing plot you can ask MATLAB to hold the plot.

```matlab
plot(x,y)
hold on
plot(x,y2)
hold off
```

hold on – hold plot
hold off – release plot
hold – toggle between on or off
• Pasting a figure from Matlab into a word document

1. Have Word open and a plot created
2. Go to the edit window and select “Copy Figure” (not copy) (Selecting Copy options first allow one to select alternate background colors, etc.)
3. Go to Word. In the Edit menu select Paste or Paste Special.
4. In Word, select the figure and click the right mouse button to get Format Picture. This will allow you to change the way it appears in your word document.
Example

x = linspace(0,10,100);
y = 5.*exp(-x).*sin(10.*x);

plot(x,y)
axis([0 10 -4 4])
xlabel('Time (s)' , 'fontsize',18)
ylabel('Displacement (m)' , 'fontsize',18)
title('Mass-spring Model' , 'fontsize',18)
Indexing

- Say you have the matrix:

\[
A = \begin{pmatrix}
1 & 2 & 3 & 4 & 5 & 6 \\
11 & 12 & 13 & 14 & 15 & 16 \\
21 & 22 & 23 & 24 & 25 & 26
\end{pmatrix}
\]

- If you want to get a particular value you can index it by row and column (in that order)

\[
A(2,3) = 13 \\
A(1,5) = 5
\]
Indexing

- If you type: \( A(2,3) \) you will get:
  \[
  \text{ans} = 13
  \]
- If you type \( A(2,3) = 1; \) the matrix will be altered to:
  \[
  A = \begin{pmatrix}
  1 & 2 & 3 & 4 & 5 & 6 \\
  11 & 12 & 1 & 14 & 15 & 16 \\
  21 & 22 & 23 & 24 & 25 & 26
  \end{pmatrix}
  \]
- If you have an array you only need to denote one-dimension but you can denote both if you want:
  \[
  a = [1 \ 2 \ 3 \ 4 \ 5 \ 6];
  \]
  \[
  a(6) \longrightarrow 6
  \]
  \[
  a(1,6) \longrightarrow 6
  \]
Indexing

• What if you want to get more than one value out of a matrix?
• \( A(a:b,c:d) \) will give the values of \( A \) with indices from \( a \) to \( b \) rows and \( c \) to \( d \) columns.
• For example, typing \( A(1:2,3:4) \) with our original matrix will give:

\[
A = \begin{bmatrix}
3 & 4 \\
13 & 14
\end{bmatrix}
\]
• If you want to get all rows or columns you can denote that by \( a : \) For example \( A(1:2,:) \) would give you:

\[
A = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 & 6 \\
11 & 12 & 13 & 14 & 15 & 16
\end{bmatrix}
\]
Basic formats

A(1,1) - denotes the values in row 1 column 1 of A
A(:, :) - denotes the values in all rows, all columns of A
A(:,1:2) - denotes the values in all rows, columns 1 and 2
A(1:2,:) - denotes the values in rows 1 and 2, all columns

[1:10] – denotes an array from 1 to 10 in increments of 1
[1:2:10] – denotes an array from 1 to 10 in increments of 2
[1 2 3] – denotes an array with 3 columns
[1;2;3] – denotes an array with 3 rows
[1 2; 3 4] – denotes a matrix with 2 rows and two columns

A([anything]) – denotes the values in A with indices equal to the values in the array in [anything ]
Creating Vectors

• In addition to linspace, logspace, ones and zeros you can create a vector using the format:

\[ v = \text{initialvalue}:\text{increment}:\text{finalvalue}; \]
Adding a row or column

- Adding
  - a row or column can be added with an equations such as:
    \[ A = [A ; u] \] (a row added) or \[ A = [A v] \] (a column)
  - where \( u \) and \( v \) are the rows or columns to be added
  - the dimensions of the added rows or columns should match the dimensions of \( A \)
Deleting a row or column

- A row or column can be deleted by assigning it to the null matrix `[]`
- For example, typing `A(2,:) = []`; gives

\[
A = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 & 6 \\
21 & 22 & 23 & 24 & 25 & 26
\end{bmatrix}
\]
Dimensioning

• You can find out the size of a matrix or array by typing:
  size(A)   - gives you the number of rows and columns
  length(x) - gives you the largest dimension
Examples

• If we have the commands:
  A = ones(10,10);
  B = zeros(10,10);
  y = [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15];
  A([1 3 6 9],:) = [B(1:3,:); y(1:10)];

What do we have?
Reshaping Matrices

• Transpose - swaps rows and columns:
  \[ A = B'; \]
  \[ v = u'; \]
  (arrays go from row arrays to column arrays)

• Reshape(A,p,q) reshapes A from an mxn matrix to a pxq matrix as long as m*n=p*q
  (takes each column and strings them together and then fills in the columns of the new matrix). This method is not used often.
Automatic Dimensioning/Initializing

- MATLAB will automatically assign space for a matrix and dimension the matrix. If you append more data it has to add space to increase the dimensions. Adding space in fragments can cause things to run slower.

- You can initialize a large matrix to make it run faster by using the zeros command.

- You can also assign a matrix to the null matrix [] if you want to do dynamic looping (we will talk about this later).
Strings

• In both Matlab and C++, a string is really just an array of characters
• Therefore you can manipulate any string the same way you might manipulate a array/matrix:

```plaintext
a = 'my cat sebastian'
b = a(8:16)
```

```plaintext
b = ‘sebastian’
```
Examples

• Which commands below will produce errors?

\[
y = [1 2 3 4 5 6 7 8 9 10];
\]
\[
x = y';
\]
\[
x(3)
\]
\[
y(3)
\]
\[
x(3,1)
\]
\[
x(1,3)
\]
\[
y(3,1)
\]
\[
y(1,3)
\]
Examples

• What will the string b say?

\[
a = 'I am the very model of a modern major general';
b(1:16) = [a(15:19) 's are ' a(33:37)];
\]
Gauss Elimination

- Matrices are often used to solve a system of linear equations of the form:

\[ a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 \ldots + a_{1n}x_n = b_1 \]
\[ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 \ldots + a_{2n}x_n = b_2 \]
\[ \ldots \]
\[ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + a_{m4}x_4 \ldots + a_{mn}x_n = b_m \]
Gauss Elimination

• These can be solved in a series of operations in which one multiples one equation by a number and then subtracts it from another equation to eliminate a variable.

• Example:

\[
\begin{align*}
    x_1 + 2x_2 + x_3 &= 4 \\
    x_1 + x_2 + 3x_3 &= 0 \quad \rightarrow \quad -x_2 + 2x_3 &= -4 \\
    2x_1 - x_2 - x_3 &= 1
\end{align*}
\]

• Then you can solve for \(x_3\), \(x_2\) and \(x_1\)
Matrix Representation

• Alternately we can represent the equations in matrix form:

\[ Ax = b \]

where

\[
A = \begin{pmatrix}
1 & 2 & 1 \\
1 & 1 & 3 \\
2 & -1 & -1 \\
\end{pmatrix}
\]

\[
x = \begin{pmatrix}
x_1 \\
x_2 \\
x_3 \\
\end{pmatrix}
\]

\[
b = \begin{pmatrix}
4 \\
0 \\
1 \\
\end{pmatrix}
\]
Matrix Representation

• We can do the same operation on the A matrix and b vector (array) as we did with the equations.

• We can also have a new mathematics to represent working with these matrices in ways that keep the equations intact. This is called linear algebra.
Linear Algebra Rules

• The matrix product of two matrices $A \times B$ follows the rule:

$$c_{ij} = \sum a_{ik} b_{kj}$$

• $A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$ $B = \begin{bmatrix} b_{11} & b_{12} & b_{33} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$

• $A \times B = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} \\ a_{21}b_{11} + a_{22}b_{21} + a_{33}b_{31} \\ a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} \end{bmatrix}$
Linear Algebra

• The inverse of a matrix \( (A^{-1}) \) is a matrix such that the cross product with \( A \) is an identity matrix:

\[
AA^{-1} = A^{-1}A = \begin{pmatrix}
1 & 0 & 0 & \ldots \\
0 & 1 & 0 & \ldots \\
0 & 0 & 1 & \ldots \\
\vdots & \vdots & \vdots & \ddots
\end{pmatrix}
\]
So that...

- If you start with $Ax = b$

$$A^{-1}Ax = A^{-1}b$$
$$x = A^{-1}b$$

in MATLAB: $x = \text{inv}(A)\times b$
Matrix Math

C = A*B  
C is the matrix product of A and B

x = inv(A)*b   or x = A\b   (not b\A as you might think)
the solution of Ax=b

(A/B only works if A and B are the same dimensions and is the same as
A*inv(B))

A^2 = A*A

A+B   adds each element (row/column) in A to the matching
      element in B

A-B   subtracts each element (row/column) in A to the matching
      element in B
Array Operations

- Dot Product
- Cross Product (used in Physics, Dynamics)
- Vector Math
Dot Product

- Dot products are like matrix-products of a row and column vector.
  \[ x = [\ x_1 \ x_2 \ x_3 \] \quad y = [\ y_1 \ y_2 \ y_3 \]
  \[ x*y = \text{dot product of } x \text{ and } y \]
- Dot products are a single number (scalar/floating point)
- The square root of the dot product is the length of a vector.
- \[ C = \text{DOT}(a,b) \]
  \[ \text{dot product} = |x||y|\cos(\theta) \]
Cross Product

• A cross product is a function of matrix products that is often used in dynamics

\[
\mathbf{u} \times \mathbf{v} = \begin{vmatrix}
  u_y & u_z \\
  v_y & v_z \\
  u_x & u_z \\
  v_x & v_y
\end{vmatrix}
\]

• \( c = \text{CROSS}(\mathbf{u}, \mathbf{v}) \)
• “right-hand” rule
Vector Addition

- $u + v$ adds two vectors together if they are both row or both column vectors.
- $u - v$ subtracts two vectors.
- i.e. relative velocity: if a toy car is moving within a larger car at a speed $v_l$ and the large car is moving $v_b$ then the total speed is $v_t = v_l + v_b$. 

![Diagram of vector addition](image.png)
Example

- A muscle creates a 5 N force in the z direction, 0.05 m in the x direction from the center of the joint, what is the moment?

\[
M = F \times d \quad \text{(from statics or physics)}
\]

\[
F = [0;0;5];
\]
\[
d = [0.05; 0; 0];
\]
\[
M = \text{cross}(F,d) \quad \rightarrow [0; .25; 0]
\]
Polynomials

- Polynomials are often represented as arrays. For example:

\[5x^4 + 4x^3 + 2x^2 + 3x + 1\]

might be represented as:

\[\begin{bmatrix} 5 & 4 & 2 & 3 & 1 \end{bmatrix}\]
Polynomial Functions

\[ y = \text{polyval}(p, x) \] - gives the value of the polynomial with coefficients \( p \) for value \( x \)

\[ p = \text{polyfit}(x, y, n) \] - creates the coefficients of a polynomial that best fits the data \( x, y \) with \( x^n, x^{n-1}, \ldots, x^0 \) as the terms

\[ r = \text{roots}(p) \] – finds the roots of a polynomial

\[ y = p(1) x^n + p(2) x^{n-1} + \ldots + p(n) x + p(n+1) \]
Basic statistics terms

• mean(x) – mean of x
• std(x) – standard deviation
• min(x) – minimum
• max(x) – maximum
• median(x) – median
• rand(N) – an NxN matrix of random numbers between 0 and 1
Doing a linear regression in MATLAB

- \( p = \text{polyfit}(x,y,1); \)
- \( y\text{fit} = \text{polyval}(p,x); \)

\[ y = p(1)x + p(2) \]
Eval

- Say you wanted to take some data you had and create a command that could then be run. Eval does this:
- `eval(s)` runs the string `s`
- example:
  ```
  name = 'fred';
  eval(['load ' name ' .txt'])
  ```
Loading Data

load fred.txt  or load –ascii fred.txt  - loads ASCII text file (strict format)
load fred.mat or load fred
load  or  load matlab.mat

uiimport  - imports data from text or spreadsheet

always from the current directory
Loading data

load file.txt

When loading text files the data should be in a matrix with spaces or tabs separating the data. The file shouldn’t have any column headers or extra blank lines.
Saving data

save → matlab.mat

save fred.mat or save fred

save fred A B or save fred.mat A B (saves only A and B)

save fred.txt A –ascii

always to the current directory unless …
Loading or Saving outside the current directory

save ..\folder/fred.mat  save u:myfolder/fred.mat

To any file name you can add path information in a Unix format:

.. directory one level down
. current directory
/ next level up
Figures

• In your program you can save and print figures with the commands `saveas` and `print`.
• `saveas` will save a figure in a number of formats including tif, bmp and mfig (Matlab):

    `saveas(gcf,'filename', 'tif')`

    `gcf` is the function “get current figure”, we will talk more about this later.
C based I/O functions

- fopen
- fclose
- fread
- fwrite
- fscanf
- fprintf
- sprintf
- fgets
- fgetl
- frewind
- fseek
- ftell
- ferror
C based I/O

fid = fopen('filename'); % opens a files fid is
               % the file ID

count = 1; % sets up dummy value for count
icnt = 0; % starts off a counter

while count>0 % read until file is finished
    [Btemp,count] = fscanf(fid,'%17f %17f %17f/n')
    icnt = icnt+1;
    B(icnt,1:length(Btemp)) = Btemp'; % assign read values to B
end
st = fclose(fid); % close the file
File ID

• FID
  – a file ID that keeps track of the file you are working with
  – only one file is assigned to each file id
  – the file id is an integer

  – a file is opened with `fopen` and closed with `fclose`
Reading options

fgets, fgetl: gets the next line as a string (with or without \n character)

fscanf: reads the data based on a format

sscanf: like fscanf for a string instead of a file

fread: read binary files (different from ASCII)

Format for the scanf commands:

%d: an integer
%f: a floating point
%c: a single character
%s: a string of characters
%3s: a string of three characters
%lf: double precision floating point
Writing commands

fprintf reads the data based on a format
sprintf like fscanf for a string instead of a file
fwrite read binary files (different from ASCII)

some formats for the printf commands:

- \%5d: a five character integer
- \%15.3f: a 15 character floating point with 3 values after the decimal
- \%c: a single character
- \%s: a string of characters
- \%lf: double precision floating point
- \%15.3e: a 15 character scientific notation with 3 values after the decimal
Other C based I/O commands

- `frewind(fid)`
  - rewinds file back to beginning (otherwise once you have read something you are on the next spot/line)

- `ftell(fid)`
  - tells you where you are in a file (in characters)
Graphical input

\[ [x, y] = \text{ginput}(n) \]

gets the x and y position off of a graph for n points from the user’s input

returns the x and y position marked by the crosshairs
Scripts

- m files that contain Matlab commands
- The main program (that calls function) is generally a script (easier for running a debugger)
- All data used is in the general workspace
Functions

- Like a script except the variables in a function file are local to that function (do not go to workspace)
- Variables are passed in and out of the function in the call

function definition → function [xout, yout] = funcname(xin,yin)
% add description of function here
% anything in these lines can be accessed
% by typing help funcname
% Be sure to include name and date

output list → function name → input list

comments used by help

function code

xout = xin*….
yout = yin*xin….
function [startpoint,endpoint] = grab(flexion);

% Datagrab gets the initial and final time
% values for flexion levels
% Author:  Sara E. Wilson
% Date:  9/1/02
%Variables:  i - a counter
%            x - point number of ginput collection
%            y - dummy variable
%            startpoint - beginning point of level
%            endpoint - ending point of level

% It might have been helpful if I had put in what needs to be passed in these lines

>> help grab

Datagrab gets the initial and final time values for flexion levels
Author:  Sara E. Wilson
Date:  9/1/02
Variables:  i - a counter
            x - point number of ginput collection
            y - dummy variable
            startpoint - beginning point of level
            endpoint - ending point of level

% Plot the flexion variable against point number
plot(flexion)
ylabel('Flexion')
xlabel('Point index')
title('Pick a beginning and end for each level')

% For each level of flexion, display instructions, get input
% data, and ascertain the start and end points
for i = 1:4
    disp('Get the beginning point of level')
    [x,y] = ginput(1);
    startpoint(i) = round(x);
    disp('Get the end point of level')
    [x,y] = ginput(1);
    endpoint(i) = round(x);
end
Calling a function

\[
[x_{\text{output}}, y_{\text{output}}] = \text{funcname}(x_{\text{in}}, y_{\text{in}});
\]
\[
[\text{first}, \text{second}] = \text{funcname}(\text{big}, \text{small});
\]
\[
[\text{first}, \text{second}] = \text{funcname}(0, 1.1);
\]
\[
[\text{first}, \text{second}] = \text{funcname}(1); \quad \text{- (you can do only the first couple of values if the function is programmed to handle it)}
\]
\[
\text{funcname}(x,y); \quad \text{- no output assigned (first output displayed w/o ; if no output arguments:}
\]
\[
[] = \text{funcname}(x,y);
\]
Inline functions

• If a function can be expressed in a single line you might write it as an inline function in your script:

\[
Fx = \text{inline}('f(x)', 'x');
\]

ie. \[
Fx = \text{inline}('x.^2 + \sin(x)', 'x');
\]
\[
x = \text{linspace}(0, \pi, 5);
\]
\[
y = Fx(x);
\]
Structure of a Program

Main Script

Function 1

Function 2

Script 1

Matlab functions (fzero, ode45)

Function A

Function B

Function C

Keeping things in small modules makes them easier to read and understand
The **global** command assigns a variable as a global variable (global zeta). The variable will appear anywhere the global command is evoked:

- script -> workspace
- function -> local workspace
Loops

- **for:**
  
  ```plaintext
  for var = start:increment:stop
      actions
  end
  for var = st:inc:stop, action, end
  for var = x, action, end  (where x is a vector)
  ```

- **while:**
  
  ```plaintext
  while value < othervalue
      actions
  end
  ```
Cases

- if:

  if value > othervalue
      actions
  end

  if value ~= othervalue
      actions
  elseif value > othervalue
      actions
  else
      actions
  end
Cases

• switch:

    switch variable
    case a
    actions
    case b
    actions
    case c
    actions
    end
If and while logicals

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

&  and (both true)
|  or (one true)
~  not
xor exclusive or (one true, one false)
Beware the infinite loop

• One of the most common mistakes with loops is to make a loop that never ends. (while condition never satisfied or function that calls itself)

  x = 10
  while x<20
    y = x+y
  end

• Matlab supports recursion (a function that calls itself) but it is not a great thing to do (confusing and difficult)
Nested loops

\[
\begin{align*}
a &= 1 \\
&\text{for } ci = 1:10 \\
&\quad \text{for } cj = 1:5 \\
&\quad \quad v(ci,cj) = a; \\
&\quad \quad a = a+1; \\
&\quad \text{end} \\
&\text{end} \\
v &= \begin{bmatrix}
1 & 2 & 3 & 4 & 5 \\
6 & 7 & 8 & 9 & 10 \\
11 & 12 & 13 & 14 & 15 \\
16 & 17 & 18 & 19 & 20 \\
21 & 22 & 23 & 24 & 25 \\
26 & 27 & 28 & 29 & 30 \\
31 & 32 & 33 & 34 & 35 \\
36 & 37 & 38 & 39 & 40 \\
41 & 42 & 43 & 44 & 45 \\
46 & 47 & 48 & 49 & 50
\end{bmatrix}
\end{align*}
\]

Executes inner for loop for each time through outer loop
Break, Error and Return

• **Break** will terminate a loop from inside
  – Stylistically crude but occasionally useful
    for count = 1:length(u)
      if u(i) < 0
        break    % breaks out of for loop
      end
    a = a + u(i);
  end

• **Error** will display a message and abort the function or script
  function c = crossprod(a,b)
    if b == NaN
      error(‘This function can’t run anymore’)
    end
  end

• **Return** will exit a function
When to use loops

• While - when doing a calculation until a certain limit is reached
• For – when wishing to repeat a task a given number of times
Nesting Loops

• Each time an outside loop is run, inside loops are run for all values

```
for cnt = 1:1:10
    cnt = 1
    while end
        if end
    end
end
```

```
for ...
    cnt = 2
    while end
        if end
    end
end
```
Nesting if statements

if condition1
  if condition2
    ‘both conditions met’
  else
    ‘condition1 met’
  end
else
  ‘condition1 not met’
end
Solving Problems with a computer

- Finding when a function goes to zero (root finding)
- Interpolating data
- Fitting data to an equation
- Solving differential equations
- Integrating or Differentiating Data
- Finding a Max or Min

<table>
<thead>
<tr>
<th>MATLAB functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>fzero, roots</td>
</tr>
<tr>
<td>interp1</td>
</tr>
<tr>
<td>polyfit</td>
</tr>
<tr>
<td>ode23, ode45, ode113...</td>
</tr>
<tr>
<td>trapz, cumtrapz, quad</td>
</tr>
<tr>
<td>fminbnd, fmincon</td>
</tr>
</tbody>
</table>
Numerical Methods

• Each of these routines requires an iterative (repeating) process (for or while loops)
• Several methods exist for solving these problems, each with its own advantages and disadvantages.
• You can learn more about some of the methods in books such as:
  Numerical Recipes (in C or in FORTRAN)
  Numerical Methods for Engineers
Numerical Integration

- Trapezoid Method
  \[ I_1 = \frac{(b-a)(f(a)+f(b))}{2} \]
  \[ I_2 = \frac{(c-b)(f(b)+f(c))}{2} \]
  ...
  \[ I = I_1 + I_2 + \ldots \]

- In MATLAB, `trapz` and `cumtrapz`
function [area] = trapezoid(a,b,n);

% Gets the area of a function from a to b
% Author: Sara E. Wilson
% Date: 9/17/02
% Variables: a = start point
% b = end point
% n = number of steps
% ci = counter
% start = beginning of each increment
% en = end of each increment
% equation = a function with the equation to be integrated

% Calculate the area for each increment using the trapezoid method
incr = (b-a)/(n-1);
area = 0;
for ci = a:incr:(b-incr)
    start = ci;
    en = ci + incr;
    area = area + (en-start)*(equation(start)+equation(en))/2;
end
Root Finding

• Matlab offers:
  \texttt{fzero(‘function’,guess)}
  \texttt{roots(p) (for polynomials)}
• \texttt{fzero} uses a combination method of \textit{secant}, \textit{bisection}
  and inverse quadratic interpolation
Root Finding - Bisection

1. Choose $x_l$ and $x_u$ such that $f(x_l)f(x_u) < 0$
2. $x_r = (x_l+x_u)/2$
3. If $f(x_l)f(x_r)<0$  - root is in lower interval
   $x_u = x_r$
   If $f(x_l)f(x_r)>0$  - root is in upper interval
   $x_l = x_r$
   If $f(x_l)f(x_r) = 0$  - the root is equal to $x_r$
4. Repeat steps 2 and 3 until $f(x_r)$ is close to zero

- Tolerance- you will never get exactly to zero so you a choose a tolerance such that it is essentially zero (ie. $\text{abs}(f(x_r))<0.5e-9$ )
Root Finding - False Position

1. Choose xl and xu such that \( f(x_l)f(x_u) < 0 \)
2. \( x_r = x_u - \frac{f(x_u)(x_l-x_u)}{(f(x_l)-f(x_u))} \)
3. If \( f(x_l)f(x_r)<0 \) - root is in lower interval
   \( x_u = x_r \)
   If \( f(x_l)f(x_r)>0 \) - root is in upper interval
   \( x_l = x_r \)
   If \( f(x_l)f(x_r) = 0 \) - the root is equal to \( x_r \)
4. Repeat steps 2 and 3 until \( f(x_r) \) is close to zero
Root Finding - Newton-Raphson

1. \( x_i \) - first guess
2. get \( f(x_i) \) and \( f'(x_i) \) (df/dx)
3. \( x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \)
4. repeat until \( f(x_i) \) is close to zero
Root Finding - Secant

1. $x_i$ - first guess
2. get $f(x_i)$ and $f'(x_i) \approx (f(x_{i-1}) - f(x))/(x_{i-1} - x_i)$
3. $x_{i+1} = x_i - f(x_i)/f'(x_i)$
4. repeat until $f(x_i)$ is close to zero
Root Finding - Issues

- Slow convergence
- Multiple roots

Bisection and Secant Confused

Slope = zero
Newton-Raphson and Secant fail to converge
Finding a maximum or minimum

• Newton’s method
  \[ x_{i+1} = x_i - \frac{f'(x)}{f''(x)} \]
  repeat until \( x_i \) and \( x_{i+1} \) are approximately the same (to a tolerance)

• Golden section search
  given \( x_{lo} \) and \( x_{u} \),
  \[ d = \frac{\sqrt{5}-1}{2}(x_u-x_{lo}) \]
  \[ x_1 = x_{lo} + d \]
  \[ x_2 = x_u - d \]
  if \( f(x_1) < f(x_2) \) then \( x_{lo} = x_2 \)  \( (\text{min would be the other way}) \)
  else \( x_u = x_1 \)

• \texttt{fminbnd} uses the golden section search
Golden Search

\[ x_2 = xu - R \]
\[ x_1 = xlo + R \]
\[ R = 0.618 \text{ (optimal)} \]

if \( x_1 > x_2 \) cut off top
else cut off bottom
Our problem

• Find the minimum of $f(x) = x^a + bx^c$
• Using the Golden Search Algorithm in Matlab
How would we write this routine?

Program Structure

Main Script

function

Minimum Finder

function

equation
Our function

function [fofx] = myfunction(x, a, b, c)
fofx = x^a + b*x^c;

called:
output = myfunction(x, a1,b1,c1);
Min Finder: Golden Search

function [xmin] = minfinder(xlo, xu,a,b,c)

while abs(xu-xlo)>0.1e-5

    d = (sqrt(5)-1)/2*(xu-xlo)
    xone = xlo+d;
    xtwo = xu-d;

    if myfunction(xone,a,b,c)<myfunction(xtwo,a,b,c)
        xlo = xtwo;
    else
        xu = xone;
    end

end

xmin = xu;

Golden section search
1. Given xlo and xu, d = (sqrt(5)-1)/2*(xu-xl)
2. x1 = xlo+d x2 = xu-d
3. if f(x1)< f(x2) then xlo = x2
   (min would be the other way)
   else xu = x1
a = 2;
b = 5;
c = 1;
xlo = -20;
xu = 20;

xvalues = [-10:.1:10];

for count = 1:length(xvalues)
    yvalues(count) = myfunction(xvalues(count),a,b,c);
end

plot(xvalues, yvalues)

[xmin] = minfinder(xlo,xu,a,b,c);
Debugging

- Plotting the points during the subroutine can help you to see if it is working correctly in minfinder function:

```matlab
y1 = myfunction(xone,a,b,c);
y2 = myfunction(xtwo,a,b,c);
ylo = myfunction(xlo,a,b,c);
yu = myfunction(xu,a,b,c);
hold on
plot(xone,y1,'*')
plot(xtwo,y2,'s')
plot(xlo,ylo,'d')
plot(xu,yu,'o')```

![Graph](image-url)
Debugging

• If you don’t put the ; on the end of a line, whatever is calculated (the left side of the equation) is displayed in the command window. This can be helpful in debugging.
Debugging

- You can also use breaks to follow the progress of the program and work in the function workspace.
Debugging options

• Work better when started in scripts
• You can set break points and then step through the program with the options under debug
Error Messages

??? Error: File: D:\homeworks\inclass3.m Line: 10 Column: 6
Expected a variable, function, or constant, found "incomplete string".

Look in Line 10 of your code for the error described:

```matlab
% In Class 3: This program draws a smiley face.
% Author:  Sara E. Wilson
% Date:  9/10/02

% Variables: none
% Functions: [] = circle(radius, x center point, y center point, color co
% [] = oval(xradius, yradius, x center point, y center point, color co

% Draw a yellow face
circle(1,0,0,'y')

disp('Did the yellow face')
```
Common Errors

??? In an assignment \( A(I) = B \), the number of elements in \( B \) and \( I \) must be the same.

Error in ==> D:\homeworks\inclass3.m
On line 11 ==> x(1) = [1 2;3 4]

This one usually mean that the dimensions on the left of the equal sign are different than the dimensions on the right

??? Error: File: D:\homeworks\inclass3.m Line: 11 Column: 15
Expected a variable, function, or constant, found "end of line".

This one usually means there is a missing )
Common Errors

??? Error using ==> plot
Vectors must be the same lengths.

Error in ==> D:\homeworks\inclass3.m
On line 13  ==> plot(x,y)

This one usually mean that the dimensions of x don’t match the dimensions of y
Fixing minfinderwrong.m

function [xmin] = minfinderwrong(xlo, xu,a,b,c)

while abs(xu-xlo)>0.1e-5
    d = (sqrt(5)-1)/2*(xu-xlo);
    xone = xlo+d;
    xtwo = xu-d;
    y1 = myfunction(xone,a,b,c);
    y2 = myfunction(xtwo,a,b,c);
    ylo = myfunction(xlo,a,b,c);
    yu = myfunction(xu,a,b,c);
    if y1>y2
        xlo = xtwo;
    else
        xu = xone;
    end
    diff = xu-xlo
end
xmin = xu;
Advanced Topics and Concepts
Writing optimal code

• In the beginning of the semester we talked a little about algorithms and why one might be better.

• How might we improve upon some of the algorithms we wrote in terms of: time it takes to run the amount of memory space used
Time

• avoid loops (particularly while loops)
• minimize the operations within loops in the lab:
  \[ \text{[xs,ys,zs]} = \text{sphere(20)} \text{ did not need to be in the loop:} \]
  \[
  \text{for } i = 1: \text{length(dx)}
  \text{ surf(xs+dx(i)),...}
  \]
  \end
• For operations such as root finding where initial guesses and searching are required, make sure equations are solvable and guesses are good.
• Avoid excessive graphics calls (it takes a lot of CPU time to draw a surf plot) use handles to redraw only what is necessary
• Avoid spewing to the screen (; on every line)
Memory

• Most of what we have done has not used much memory. However, in the future you may find yourself faced with memory space issues when dealing with large datasets.

• Avoid unnecessary global variables

• Use functions to contain variables that don’t need to be passed around

• Don’t have multiple variables with the same data:

  ```matlab
  load indata.txt
  data = indata(:,1);
  data2 = indata(:,2);
  processeddata = myfunction(data);
  processeddata2 = myfunction2(data2);
  ```

  ```matlab
  load indata.txt
  indata(:,1) = myfunction(indata(:,1));
  indata(:,2) = myfunction2(indata(:,1));
  ```
Clarity

• At some point in your career you will write code that someone else will have to try and understand. This will go much better if the code is clear and well commented.

• Avoid too many nested layers

• Be consistent and clear in your variable names (time rather than t)

• Make your code modular (small functions that accomplish simple tasks)

• Comment profusely and clearly
Object-oriented programming

• An object with many properties is considered as a grouped element
• Abstraction – the ability to focus only on the relevant properties while ignoring the rest
• Encapsulation – the separation of groups of properties so that one group of properties of an object can be used while ignoring the rest.
• Modularity – groups of properties
Example: A car

- Car
  - Appearance
    - Color
    - Size
  - Combustion
    - Engine Type/Size
  - Motion
    - 0-60mph
    - 2/4-wheeled drive
Structures

• A structure is a data construct with several fields.
  – fields can have different types of data
  – structures can have multiple levels (nested structures)

  – The syntax for structures is:
    Structurename1.field1a.field1b
  – These fields can be arrays:
    Structurename1.field1(2).field1b
Creating a structure

The struct command:

\[
s = \text{struct}('field1',{},'field2',{},\ldots);
\]

\[
s = \text{struct}('field1',\text{values1},'field2',\text{values2},\ldots);
\]

\[
\text{exp1} = \text{struct}('subject1',{\text{struct}('motion', \{\text{data}\}, 'name', \{'fred'\})},'subject2',{\text{struct}('motion', \{\text{data}\}, 'name', \{'sally'\})});
\]

Or by assigning values:

\[
\text{exp1}.subject1.motion = \text{data};
\]

\[
\text{exp1}.subject1.name = \text{'fred'};
\]

\[
\text{exp1}.subject2.motion = \text{data2};
\]

\[
\text{exp1}.subject2.name = \text{'sally'};
\]

\[
\text{exp1} =
\]

\[
\begin{array}{l}
\text{subject1: [1x1 struct]} \\
\text{subject2: [1x1 struct]} \\
\text{exp1.subject1 =} \\
\text{motion: [48x10 double]} \\
\text{name: 'fred'}
\end{array}
\]
Manipulating a Structure

- \texttt{Fall\_sem(3).score(2)} gets you the second value in the field score for the third level of the structure
- \texttt{a = Fall\_sem(3).score;} assigns array \texttt{a} to the 3 scores in the third structure level
Some useful structure commands

- **cell2struct**: Cell array to structure array conversion
- **class**: Return object's class name (e.g., struct)
- **deal**: Deal inputs to outputs
  
  *(eg. [S.field] = deal(X) makes all entries in the S.field X)*
- **fieldnames**: Field names of structure
- **getfield**: Get field of structure array
- **isa**: Detect object of given class (e.g., struct)
- **isfield**: Determine if item is structure array field
- **isstruct**: Determine if item is structure array
- **rmfield**: Remove structure fields
- **setfield**: Set field of structure array
- **struct**: Create structure array
- **struct2cell**: Structure to cell array conversion
Cell Arrays

• Cell arrays are similar to structures but with a different syntax and in some ways more versatile.

• Cell arrays are like matrices with each cell containing its own data:

<table>
<thead>
<tr>
<th>5</th>
<th>array x</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘fred’</td>
<td>struct time</td>
</tr>
</tbody>
</table>
Creating cell arrays

The cell command:
\[ c = \text{cell}(2,2); \]
\[ c\{1,1\} = 5; \]
\[ c\{1,2\} = x; \]
\[ c\{2,1\} = 'fred'; \]
\[ c\{2,2\} = \text{time}; \]

Or by assigning values:
\[ c = \{5 \ x; \ 'fred' \ \text{time}\}; \]
\[ c = \begin{bmatrix}
5 & [1\times5 \text{ double}] \\
'fred' & [1\times1 \text{ struct}]
\end{bmatrix} \]
Manipulating Cell Arrays

- \( \text{c\{1,2\}(3)} \) gets you third value of the array \( x \)
- \( \text{a = c\{2,2\}\{1,1\};} \) assigns the values in the first cell of time to \( a \).
- \( \text{a = c\{2,2\}.min;} \) assigns the values in the field min of time to \( a \).
Some useful Cell array commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell</td>
<td>Construct cell array</td>
</tr>
<tr>
<td>cellfun</td>
<td>Apply function to each element in cell array</td>
</tr>
<tr>
<td>cellstr</td>
<td>Create cell array of strings from character array</td>
</tr>
<tr>
<td>cell2struct</td>
<td>Cell array to structure array conversion</td>
</tr>
<tr>
<td>celldisp</td>
<td>Display cell array contents</td>
</tr>
<tr>
<td>cellplot</td>
<td>Graphically display structure of cell arrays</td>
</tr>
<tr>
<td>class</td>
<td>Return object's class name (e.g., cell)</td>
</tr>
<tr>
<td>deal</td>
<td>Deal inputs to outputs</td>
</tr>
<tr>
<td>isa</td>
<td>Detect object of given class (e.g., cell)</td>
</tr>
<tr>
<td>iscell</td>
<td>Determine if item is cell array</td>
</tr>
<tr>
<td>iscellstr</td>
<td>Determine if item is cell array of strings</td>
</tr>
<tr>
<td>isequal</td>
<td>Determine if arrays are numerically equal</td>
</tr>
<tr>
<td>num2cell</td>
<td>Convert numeric array into cell array</td>
</tr>
<tr>
<td>struct2cell</td>
<td>Structure to cell array conversion</td>
</tr>
</tbody>
</table>
What to remember about structures

- Structures have levels each of which can have all types of variables
- Structures allow one to organize data into groups
- An element of a structure is called by the following type of command:
  ```
  structname.fieldname
  structname.fieldname.fieldname2
  ```
- This can be used just like any other variable:
  ```
  structname.fieldname = 10;
  y = structname.fieldname(2,4);
  ```
What to remember about cell arrays

- Cell arrays have cells each of which can have all types of variables
- Cell arrays allow one to organize data into groups
- An element of a structure is called by the following type of command:
  \[
  \text{cellarray}\{1,3\}
  \]
  \[
  \text{cellarray}\{1,3\}\{2,3\} \quad \text{- nested cell arrays}
  \]
- This can be used just like any other variable:
  \[
  \text{cellarray}\{3,4\} = 10;
  \]
  \[
  y = \text{cellarray}\{1,2\};
  \]
  \[
  \text{cellarray}\{1,3\}(2,3) = 5; \quad \text{- if cell element is a matrix}
  \]
Structures

- How might one use structures or cell arrays?

sacrdata

\[
\begin{align*}
\text{upright}: & \quad [1x1 \text{ struct}] \\
\text{seat}: & \quad [1x1 \text{ struct}] \\
\text{preosc}: & \quad [1x1 \text{ struct}] \\
\text{fastosc}: & \quad [1x1 \text{ struct}] \\
\text{slowosc}: & \quad [1x1 \text{ struct}] \\
\text{rom}: & \quad [1x1 \text{ struct}] \\
\text{lift}: & \quad [1x1 \text{ struct}] \\
\end{align*}
\]

\[
\begin{align*}
\text{flex}: & \quad [1x1 \text{ struct}] \\
\text{lat}: & \quad [1x1 \text{ struct}] \\
\text{twist}: & \quad [1x1 \text{ struct}] \\
\end{align*}
\]

\[
\begin{align*}
\text{a11}: & \quad [1x1448 \text{ double}] \\
\text{a12}: & \quad [1x1448 \text{ double}] \\
\text{a13}: & \quad [1x1448 \text{ double}] \\
\text{a21}: & \quad [1x1448 \text{ double}] \\
\text{a22}: & \quad [1x1448 \text{ double}] \\
\text{a23}: & \quad [1x1448 \text{ double}] \\
\text{a31}: & \quad [1x1448 \text{ double}] \\
\text{a32}: & \quad [1x1448 \text{ double}] \\
\text{a33}: & \quad [1x1448 \text{ double}] \\
\end{align*}
\]

\[
eval(['a{1,1} = sacrdata.' 'trialtype ' '.a11;'])
\]

\[
a{1,1} = sacrdata.rom.flex.a11;
\]
What does this do?

freddata = 
[1 3 5 7 9 11 13 15 17 19 21 23 25 27 29];
cell{3,1} = 'fred';
cell{2,1} = freddata([1:2:8]);
cell{1,1} = 24;

newdata = cell{2,1}(1:4)

disp(['The data for ' cell{3,1} ' is ' num2str(newdata)])
Graphics

- So far we have used:
  `plot(x,y,’style options’)`
  where the style options are such things as line type, color and symbols.

- We can label this with:
  `xlabel(‘ x axis label’)`
  `ylabel(‘ y axis label’)`
  `title(‘Title’)`
  `text(2,6,’text at the coordinates 2,6’)`
  `legend(‘data set 1’, ‘dataset 2’,…)`
Graphics

- We can control the axes with:
  
  \[ \text{axis}([\text{xmin} \ \text{xmax} \ \text{ymin} \ \text{ymax}]) \]
  
  \[ \text{axis}('\text{square}') \] - shape is square
  
  \[ \text{axis}('\text{equal}') \] - equal scale
  
  \[ \text{axis}('\text{off}') \] - remove tick marks and numbers

- We can plot multiple lines of data with:
  
  \[ \text{plot}(x_1,y_1,'\text{style1'},x_2,y_2,'\text{style2'})…) \]

- We can use the “hold on” and “hold off” commands to plot one plot over the other
Graphics

- Other 2-D plot options are on page 167 of your book. These include:
  
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorbar</td>
<td>Plot graph with error bars</td>
</tr>
<tr>
<td>loglog</td>
<td>Plot using log-log scales</td>
</tr>
<tr>
<td>polar</td>
<td>Polar coordinate plot</td>
</tr>
<tr>
<td>plotyy</td>
<td>Plot graphs with Y tick labels on the left and right</td>
</tr>
<tr>
<td>semilogx</td>
<td>Semi-log scale plot (x-axis)</td>
</tr>
<tr>
<td>semilogy</td>
<td>Semi-log scale plot (y-axis)</td>
</tr>
<tr>
<td>fill</td>
<td>Draws filled polygons</td>
</tr>
<tr>
<td>contour</td>
<td>make a contour plot</td>
</tr>
<tr>
<td>fplot</td>
<td>plots a function</td>
</tr>
<tr>
<td>subplot</td>
<td>creates multiple plots on one figure</td>
</tr>
<tr>
<td>Function</td>
<td>Example Script</td>
</tr>
<tr>
<td>----------</td>
<td>----------------</td>
</tr>
</tbody>
</table>
| fplot    | $f(t) = t \sin t, \ 0 \leq t \leq 10\pi$<br>$fplot('x.*sin(x)',[0 10*pi])$
|          | Note that the function to be plotted must be written as a function of $x$. |
| semilogx | $x = e^{-t}, \ y = t, \ 0 \leq t \leq 2\pi$<br>$t=linspace(0,2*pi,200);$<br>$x = \exp(-t); \ y = t;$<br>$semilogx(x,y), \ grid$ |
| semiogy  | $x = t, \ y = e^t, \ 0 \leq t \leq 2\pi$<br>$t=linspace(0,2*pi,200);$<br>$semiogy(t,\exp(t))$
<p>|          | $\ grid$ |
| loglog   | $x = e^t, \ y = 100 + e^{2t}, \ 0 \leq t \leq 2\pi$&lt;br&gt;$t=linspace(0,2<em>pi,200);$&lt;br&gt;$x = \exp(t);$&lt;br&gt;$y = 100+\exp(2</em>t);$&lt;br&gt;$loglog(x,y), \ grid$ |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Example Script</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>polar</td>
<td>$r^2 = 2\sin 5t, \ 0 \leq t \leq 2\pi$&lt;br&gt;t=linspace(0,2<em>pi,200);&lt;br&gt;r=sqrt(abs(2</em>sin(5*t))));&lt;br&gt;polar(t,r)</td>
<td><img src="image" alt="Polar Plot" /></td>
</tr>
<tr>
<td>fill</td>
<td>$r^2 = 2\sin 5t, \ 0 \leq t \leq 2\pi$&lt;br&gt;$x = r \cos t, y = r \sin t$&lt;br&gt;t=linspace(0,2<em>pi,200);&lt;br&gt;r=sqrt(abs(2</em>sin(5*t))));&lt;br&gt;x=r.*cos(t);&lt;br&gt;y=r.*sin(t);&lt;br&gt;fill(x,y,'k');&lt;br&gt;axis('square')</td>
<td><img src="image" alt="Fill Plot" /></td>
</tr>
<tr>
<td>bar</td>
<td>$r^2 = 2\sin 5t, \ 0 \leq t \leq 2\pi$&lt;br&gt;$y = r \sin t$&lt;br&gt;t=linspace(0,2<em>pi,200);&lt;br&gt;r=sqrt(abs(2</em>sin(5*t))));&lt;br&gt;y=r.*sin(t);&lt;br&gt;bar(t,y)&lt;br&gt;axis([0 pi 0 inf])</td>
<td><img src="image" alt="Bar Plot" /></td>
</tr>
<tr>
<td>errorbar</td>
<td>$f_{\text{approx}} = x - \frac{x^3}{3!}, \ 0 \leq x \leq 2$&lt;br&gt;$\text{error} = f_{\text{approx}} - \sin x$&lt;br&gt;x=0:.1:2;&lt;br&gt;aprx2=x-x.'*3/6;&lt;br&gt;er=aprx2-sin(x);&lt;br&gt;errorbar(x,aprx2,er)</td>
<td><img src="image" alt="Errorbar Plot" /></td>
</tr>
<tr>
<td>Function</td>
<td>Example Script</td>
<td>Output</td>
</tr>
<tr>
<td>----------</td>
<td>----------------</td>
<td>--------</td>
</tr>
<tr>
<td>hist</td>
<td>World population by continents&lt;br&gt;cont=char('Asia','Europe','Africa',&lt;br&gt;'N. America','S. America');&lt;br&gt;pop = [3332;696;694;437;307];&lt;br&gt;barch(pop)&lt;br&gt;for i=1:5,&lt;br&gt;  gtext(cont{i,:});&lt;br&gt;end&lt;br&gt;xlabel('Population in millions')&lt;br&gt;Title('World Population (1992)',&lt;br&gt;'fontsize',18)</td>
<td><img src="image1.png" alt="" /></td>
</tr>
<tr>
<td>plotyy</td>
<td>$y_1 = e^{-x} \sin x$, $0 \leq t \leq 10$&lt;br&gt;$y_2 = e^x$&lt;br&gt;x=1:.1:10;&lt;br&gt;y1 = exp(-x).*sin(x);&lt;br&gt;y2 = exp(x);&lt;br&gt;Ax = plotyy(x,y1,x,y2);&lt;br&gt;hy1 = get(Ax(1),'ylabel');&lt;br&gt;hy2 = get(Ax(2),'ylabel');&lt;br&gt;set(hy1,'string','$e^{-x} \sin x$');&lt;br&gt;set(hy2,'string','$e^x$');</td>
<td><img src="image2.png" alt="" /></td>
</tr>
<tr>
<td>area</td>
<td>$y = \frac{\sin x}{x}$, $-3\pi \leq x \leq 3\pi$&lt;br&gt;x=linspace(-3<em>pi,3</em>pi,100);&lt;br&gt;y=sin(x)./x;&lt;br&gt;area(x,y)&lt;br&gt;xlabel('x'),ylabel('sin(x)/x')&lt;br&gt;hold on&lt;br&gt;x1=x(46:56); y1=y(46:56);&lt;br&gt;area(x1,y1,'facecolor','y')</td>
<td><img src="image3.png" alt="" /></td>
</tr>
<tr>
<td>pie</td>
<td>World population by continents&lt;br&gt;cont=char('Asia','Europe','Africa',&lt;br&gt;'N. America','S. America');&lt;br&gt;pop = [3332;696;694;437;307];&lt;br&gt;pie(pop)&lt;br&gt;for i=1:5,&lt;br&gt;  gtext(cont{i,:});&lt;br&gt;end&lt;br&gt;Title('World Population',&lt;br&gt;'fontsize',18)</td>
<td><img src="image4.png" alt="" /></td>
</tr>
<tr>
<td>Function</td>
<td>Example Script</td>
<td>Output</td>
</tr>
<tr>
<td>----------</td>
<td>----------------</td>
<td>--------</td>
</tr>
<tr>
<td>hist</td>
<td><code>Histogram of 50 randomly distributed numbers between 0 and 1.\n\ny=randn(50,1);\n\nhist(y)</code></td>
<td><img src="image" alt="Histogram output" /></td>
</tr>
<tr>
<td>stem</td>
<td><code>f = e^{-t/5} \sin t, 0 \leq t \leq 2\pi\n\nt=linspace(0,2*pi,200);\nf=exp(-.2*t).*sin(t);\n\nstem(t,f)</code></td>
<td><img src="image" alt="Stem output" /></td>
</tr>
<tr>
<td>stairs</td>
<td><code>r^2 = 2 \sin 5t, 0 \leq t \leq 2\pi\n\ny = r \sin t\n\nt=linspace(0,2*pi,200);\nr=sqrt(abs(2*sin(5*t)));\ny=r.*sin(t);\n\nstairs(t,y);\n\naxis([0 pi 0 inf]);</code></td>
<td><img src="image" alt="Stairs output" /></td>
</tr>
<tr>
<td>compass</td>
<td><code>z = \cos \theta + i \sin \theta, -\pi \leq \theta \leq \pi\n\n\nth=-pi:pi/5:pi;\nzx=cos(th);\nzy=sin(th);\n\nz=zx+i*zy;\n\ncompass(z)</code></td>
<td><img src="image" alt="Compass output" /></td>
</tr>
<tr>
<td>Function</td>
<td>Example Script</td>
<td>Output</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>comet</td>
<td>$y = t \sin t, \ 0 \leq t \leq 10\pi$</td>
<td><img src="image1" alt="Output" /></td>
</tr>
<tr>
<td></td>
<td><code>q=linspace(0,10*pi,200);</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>y = q.*sin(q);</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>comet(q,y)</code></td>
<td><img src="image2" alt="Output" /></td>
</tr>
<tr>
<td></td>
<td>(Its better to see it on screen)</td>
<td></td>
</tr>
<tr>
<td>contour</td>
<td>$z = -\frac{1}{2}x^2 + xy + y^2$</td>
<td><img src="image3" alt="Output" /></td>
</tr>
<tr>
<td></td>
<td>$</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td><code>r=-5:.2:5;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>[X,Y]=meshgrid(r,r);</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>Z=-.5*X.^2 + X.*Y + Y.^2;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>cs=contour(X,Y,Z);</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>clabel(cs)</code></td>
<td><img src="image4" alt="Output" /></td>
</tr>
<tr>
<td>quiver</td>
<td>$z = x^2 + y^2 - 5\sin(xy)$</td>
<td><img src="image5" alt="Output" /></td>
</tr>
<tr>
<td></td>
<td>$</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td><code>r=-2:.2:2;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>[X,Y]=meshgrid(r,r);</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>Z=X.^2 -5*sin(X.*Y) + Y.^2;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>[dx,dy]=gradient(Z,.2,.2);</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>quiver(X,Y,dx,dy,2);</code></td>
<td></td>
</tr>
<tr>
<td>pcolor</td>
<td>$z = x^2 + y^2 - 5\sin(xy)$</td>
<td><img src="image6" alt="Output" /></td>
</tr>
<tr>
<td></td>
<td>$</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td><code>r=-2:.2:2;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>[X,Y]=meshgrid(r,r);</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>Z=X.^2 -5*sin(X.*Y) + Y.^2;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>pcolor(Z), axis('off')</code></td>
<td><img src="image7" alt="Output" /></td>
</tr>
<tr>
<td></td>
<td><code>shading interp</code></td>
<td></td>
</tr>
</tbody>
</table>
Plotting in three dimensions

- Matlab also has a number of options for plotting in three dimensions. These include:
  - `plot3`: plot curves in 3-D space
  - `bar3`: 3-D bar graph
  - `pie3`: 3-D pie chart
  - `comet3`: animated 3-D line plot
  - `fill3`: 3-D polygon
  - `contour3`: 3-D contour plot
  - `mesh`: 3-D mesh surface
  - `surf`: 3-D surface plots
  - `sphere`: generates a sphere
  - `ellipsoid`: generates an ellipsoid
  - `cylinder`: generates a cylinder
Graphing in 3-D

• There are many ways to view a 3-D graph
  view(azimuth, elevation) changes the viewing angle of a 3-D graph
  view(2) show the x-z projection
  view(3) same as view(-37.5,30)
  rotate3d allow the mouse to control rotation
<table>
<thead>
<tr>
<th>Function</th>
<th>Example Script</th>
<th>Output</th>
</tr>
</thead>
</table>
| plot3 | Plot of a parametric space curve:  
\[ x(t) = t, \quad y(t) = t^2, \quad z(t) = t^3. \]  
\[ 0 \leq t \leq 1. \]  
\[ t= \text{linspace}(0,1,100); \]  
\[ x=t; \quad y=t.^2; \quad z=t.^3; \]  
\[ \text{plot3}(x,y,z), \text{grid} \] | ![3D plot of a parametric curve](image) |
| fill3 | Plot of 4 filled polygons with 3 vertices each.  
\[ X=[0 \quad 0 \quad 0 ; 1 \quad 1 \quad -1 1 ; 1 \quad -1 \quad -1 -1 ]; \]  
\[ Y=[0 \quad 0 \quad 0 ; 4 \quad 4 \quad 4 4 ]; \]  
\[ Z=[0 \quad 0 \quad 0 ; 1 \quad 1 \quad -1 -1 ]; \]  
\[ \text{fill3}(X,Y,Z,\text{rand}(3,4)); \]  
\[ \text{view}(120,30) \] | ![3D filled polygons](image) |
| contour3 | Plot of 3-D contour lines of  
\[ z = \frac{5}{1 + x^2 + y^2}, \]  
\[ |x| \leq 3, |y| \leq 3. \]  
\[ r = \text{linspace}(-3,3,50); \]  
\[ [x,y]=\text{meshgrid}(r,r); \]  
\[ z=-5./(1+x.^2+y.^2); \]  
\[ \text{contour3}(z) \] | ![3D contour lines](image) |
<table>
<thead>
<tr>
<th>Function</th>
<th>Example Script</th>
<th>Output</th>
</tr>
</thead>
</table>
| surf     | \[ z = \cos x \cos y e^{\sqrt{x^2+y^2}} \]
          | \( |x| \leq 5, \ |y| \leq 5 \) | ![surf](image) |
|          | \( u = -5:.2:5; \)   | ![surf](image) |
|          | \([X,Y] = \text{meshgrid}(u, u); \) |   |
|          | \( Z = \cos(X) \cdot \cos(Y) \cdot \) |   |
|          | \( \exp(-\sqrt{(X.^2+Y.^2)/4}); \) |   |
|          | \text{surf}(X,Y,Z) \) |   |
| surfc    | \[ z = \cos x \cos y e^{x^2+y^2/4} \]
          | \( |x| \leq 5, \ |y| \leq 5 \) | ![surfc](image) |
|          | \( u = -5:.2:5; \) | ![surfc](image) |
|          | \([X,Y] = \text{meshgrid}(u, u); \) |   |
|          | \( Z = \cos(X) \cdot \cos(Y) \cdot \) |   |
|          | \( \exp(-\sqrt{(X.^2+Y.^2)/4}); \) |   |
|          | \text{surfc}(Z) \) |   |
|          | \text{view}(-37.5,20) |   |
|          | \text{axis('off')} |   |
| surfl    | \[ z = \cos x \cos y e^{x^2+y^2/4} \]
          | \( |x| \leq 5, \ |y| \leq 5 \) | ![surfl](image) |
|          | \( u = -5:.2:5; \) | ![surfl](image) |
|          | \([X,Y] = \text{meshgrid}(u, u); \) |   |
|          | \( Z = \cos(X) \cdot \cos(Y) \cdot \) |   |
|          | \( \exp(-\sqrt{(X.^2+Y.^2)/4}); \) |   |
|          | \text{surfl}(Z) \) |   |
|          | \text{shading interp} |   |
|          | \text{colormap hot} |   |

**Note:** Plotting a surface with \text{surf}(X,Y,Z) shows proper values on the \( x \) and \( y \) axes while plotting the surface with \text{surf}(Z) shows the row and column indices of matrix \( Z \) on the \( x \) and \( y \) axes. Same is true for other 3D plotting commands such as \text{mesh}, \text{contour3}, etc. Compare the values on the \( x \) and \( y \) axes in the first and the last figure in the table above.
<table>
<thead>
<tr>
<th>Function</th>
<th>Example Script</th>
<th>Output</th>
</tr>
</thead>
</table>
| mesh     | \[
\begin{align*}
  z &= -\frac{5}{1 + x^2 + y^2} \\
  &\quad \text{where } |x| \leq 3, \ |y| \leq 3
\end{align*}
\]  
\[
\begin{align*}
  x &= \text{linspace}(-3,3,50); \\
  y &= x; \\
  [x,y] &= \text{meshgrid}(x,y); \\
  z &= 5 ./ (1 + x.^2 + y.^2); \\
  \text{mesh}(z)
\end{align*}
\] | ![Mesh](image1.png) |
| meshz    | \[
\begin{align*}
  z &= -\frac{5}{1 + x^2 + y^2} \\
  &\quad \text{where } |x| \leq 3, \ |y| \leq 3
\end{align*}
\]  
\[
\begin{align*}
  x &= \text{linspace}(-3,3,50); \\
  y &= x; \\
  [x,y] &= \text{meshgrid}(x,y); \\
  z &= 5 ./ (1 + x.^2 + y.^2); \\
  \text{meshz}(z) \\
  \text{view}(-37.5, 50)
\end{align*}
\] | ![Meshz](image2.png) |
| waterfall | \[
\begin{align*}
  z &= -\frac{5}{1 + x^2 + y^2} \\
  &\quad \text{where } |x| \leq 3, \ |y| \leq 3
\end{align*}
\]  
\[
\begin{align*}
  x &= \text{linspace}(-3,3,50); \\
  y &= x; \\
  [x,y] &= \text{meshgrid}(x,y); \\
  z &= 5 ./ (1 + x.^2 + y.^2); \\
  \text{waterfall}(z) \\
  \text{hidden off}
\end{align*}
\] | ![Waterfall](image3.png) |
<table>
<thead>
<tr>
<th>Function</th>
<th>Example Script</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pie3</code></td>
<td>World population by continents</td>
<td><img src="image" alt="Pie Chart" /></td>
</tr>
</tbody>
</table>
|          | % popdata: As,EU,Af,NA,SA  
pop = [3332;696;694;437;307];  
pie3(pop)  
Title('World Population') |        |
| `stem3`  | Discrete data plot with stems | ![Stem Plot](image) |
|          | $x = t, \quad y = t \sin(t),$  
$z = e^{t/10} - 1$  
for $0 \leq t \leq 6\pi.$  
t = linspace(0,6*pi,200);  
x = t;  
y = t.*sin(t);  
z = exp(t/10)-1;  
stem3(x,y,z,'filled')  
xlabel('x'),  
ylabel('x sin(x)'),  
zlabel('e^{t/10-1}') |        |
| `ribbon` | 2D curves as ribbons in 3D | ![Ribbon Plot](image) |
|          | $y_1 = \sin(t), \quad y_2 = e^{1.5t} \sin(t)$  
$y_3 = e^{-8t} \sin(t)$  
for $0 \leq t \leq 5\pi.$  
t = linspace(0,5*pi,100);  
y1 = sin(t);  
y2 = exp(-1.5*t).*sin(t);  
y3 = exp(-8*t).*sin(t);  
y = [y1; y2; y3];  
ribbon(t,y',1) |        |
<table>
<thead>
<tr>
<th>Function</th>
<th>Example Script</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>sphere</td>
<td><code>sphere(20)</code></td>
<td><img src="image" alt="Sphere" /></td>
</tr>
<tr>
<td></td>
<td>or <code>[x,y,z]=sphere(20); surf(x,y,z)</code></td>
<td></td>
</tr>
<tr>
<td>ellipsoid</td>
<td>An ellipsoid of radii $rx = 1$, $ry = 2$ and $rz = 0.5$, centered at the origin.</td>
<td><img src="image" alt="Ellipsoid" /></td>
</tr>
<tr>
<td></td>
<td><code>cx=0; cy=0; cz=0; rx=1; ry=2; rz=0.5; ellipsoid(cx,cy,cz,rx,ry,rz)</code></td>
<td></td>
</tr>
<tr>
<td>cylinder</td>
<td>A cylinder generated by $r = \sin(3\pi z) + 2$ for $0 \leq z \leq 1$, $0 \leq \theta \leq 2\pi$.</td>
<td><img src="image" alt="Cylinder" /></td>
</tr>
<tr>
<td></td>
<td><code>z=[0:.02:1]'; r=sin(3*pi*z)+2; cylinder(r)</code></td>
<td></td>
</tr>
<tr>
<td>slice</td>
<td>Slices of the volumetric function $f(x,y,z) = x^2 + y^2 - z^2$ for $</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td><code>v = [-3:.2:3]; [x,y,z]=meshgrid(v,v,v); f=(x.^2+y.^2-z.^2); xv=[-2 2.5]; yv=2.5; zv=0; slice(x,y,z,f,xv,yv,zv); view([-30 30])</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The value of the function is indicated by the color intensity.</td>
<td></td>
</tr>
</tbody>
</table>
x0 = 100;
y0 = 10;
z0 = 200;
ax = 0;
ay = 0;
az = -9.8;
vx = -1.5;
vy = -10;
vz = 0;
dt = .2;
[x,y,z] = sphere(10);
surf(10*x+x0,10*y+y0,10*z+z0)

vx = vx+ax*dt;
vx = vy+ay*dt;
vx = vz+az*dt;
x0 = x0+vx*dt;
y0 = y0 + vy*dt;
z0 = z0 + vz*dt;

pause(dt)

if (z0)<0
    vz = abs(vz);
end

if (y0)<-100
    vy = abs(vy);
elseif (y0)>100
    vy = -abs(vy);
end

end

for i = 1:dt:100
    [x,y,z] = sphere(5);
surf(10*x+x0,10*y+y0,10*z+z0)
Dealing with Graphics

• Many of the elements of graphics are operated in an object-oriented manner using handles
• Anytime a plot-related command is called handles are created associated with the plot

\[
\begin{align*}
  h1 &= \text{plot}(x, y, 'r-') \quad \text{h1 is the handle of the line} \\
  hx1 &= \text{xlabel}(\text{'}X\text{ axis}') \quad \text{hx1 is the handle of the x-label} \\
  \text{figure}(h) &= h = \text{is the handle of the figure} \\
  h &= \text{gcf} \quad h = \text{is the handle of the current figure} \\
  ha &= \text{gca} \quad ha = \text{is the handle of the current axes} \\
  ho &= \text{gco} \quad ho = \text{is the handle of the current object}
\end{align*}
\]
get

- `get(handle)` returns all properties of the handled object/figure
- `get(handle, 'PropertyName')` returns a single property

```matlab
h1 = plot(x,y,'g-*');
get(h1,'linestyle') returns ‘-’
```

```matlab
get(gcf) returns
```

Alphamap = [ (1 by 64) double array]  
BackingStore = on  
CloseRequestFcn = closereq  
Color = [0.8 0.8 0.8]  
Colormap = [ (64 by 3) double array]  
CurrentAxes = [100.001]  
CurrentCharacter =  
CurrentObject = []  
CurrentPoint = [0 0]  
Dithermap = [ (64 by 3) double array]  
DithermapMode = manual  
DoubleBuffer = off  
...
set

• set(handle,’PropertyName’,’PropertyValue’) sets a single property

```matlab
h1 = plot(x,y,’g-*’);
set(h1,’linestyle’, ‘--’) changes the line to dashed
set(h1,’linewidth’,3) changes the line thickness
```
children

- In object-oriented style, some handles have “children”. These children are handles to sublevel properties. These can be accessed as follows:

```matlab
h2 = get(gca,’children’);
set(h2(1),’linewidth’,3)
```
advantages of handles

• In the previous lecture I showed a bouncing ball animation that ran slow because I redraw the figure every step.
• This could be done faster by using handles. For example:
  \[ h1 = \text{plot}(x1,y1) \]
  \[ \text{set}(h1,'xdata',\text{newx1},'ydata',\text{newy1}); \]
• If you want to control what gets erased, one of the handles you can use is erase:
  \[ \text{plot}(x,y,'\text{erase'},'\text{xor}') \]
  \[ \text{erase} \]
  \[ \text{plot}(x,y,'\text{erase'},'\text{none}') \]
  \[ \text{don’t erase} \]
Making 3-D surf and mesh plots

- Mesh and Surf plots use an x-y grid of data to plot. This grid can be created using meshgrid:

  \[
  \begin{align*}
  &rx = 0:1:4; \\
  &ry = -4:2:4; \\
  &[X,Y] = \text{meshgrid}(rx,ry); \\
  &Z = X.*Y.*(X.^2-Y.^2); \\
  &\text{meshc}(X,Y,Z)
  \end{align*}
  \]
Subplot

- If you want to have multiple plots on a figure you can use subplot before it plot-type command.

```
subplot(rows, columns, plotnumber)
```

```
subplot(2,1,1)  
plot(x,y)  
subplot(2,1,2)  
surf(X,Y,Z)
```
Movies

• Using the commands moviein, getframe and movie you can create your own movie in matlab.

\[ \text{Frames} = \text{moviein}(\text{nframes}) \]  %initializes the movie
for count = 1:nframes
  ...
  plot(x,y)  %any plot command
  Frames(:,count) = getframe;
end
movie(\text{Frames,n,fps})  %runs movie n times
  %at fps frames per sec

• movie2avi creates an avi file out of your movie
What does this code do?

```matlab
xv = 2*rand(1,100)-1;
yv = 2*rand(1,100)-1;
zv = 3./(1+xv.^2 + yv.^2);
xi = linspace(-1,1,30);
yi = xi';

[XI,YI,ZI] = griddata(xv,yv,zv,xi,yi,'v4');
surf(XI,YI,ZI)
```
What does this code do?

t = linspace(0,pi,50);
x = t.*sin(t);
h1 = plot(t,x);
set(h1,'linestyle','--');
set(h1,'linewidth',3);
yvec = get(h1,'ydata');
yvec(25:35) = ones(size(yvec(25:35)));
set(h1,'ydata',yvec);
How would you…?

Plot a circle that moves along a line using handles such as:
clf
clear
theta = linspace(0,2*pi,100);
x = cos(theta);
y = sin(theta);
htrail = plot(x(1),y(1),'marker','.','color','r','erase','none')
hold on
hbead = plot(x(1),y(1),'marker','o','markersize',8,'erase','xor')
hold off
axis([-1 1 -1 1])
for k = 2:length(theta)
    set(hbead,'xdata',x(k),'ydata',y(k))
    set(htrail,'xdata',x(k),'ydata',y(k))
pause(.5)
end