Chemical Engineering 541

Computer Aided Design Methods

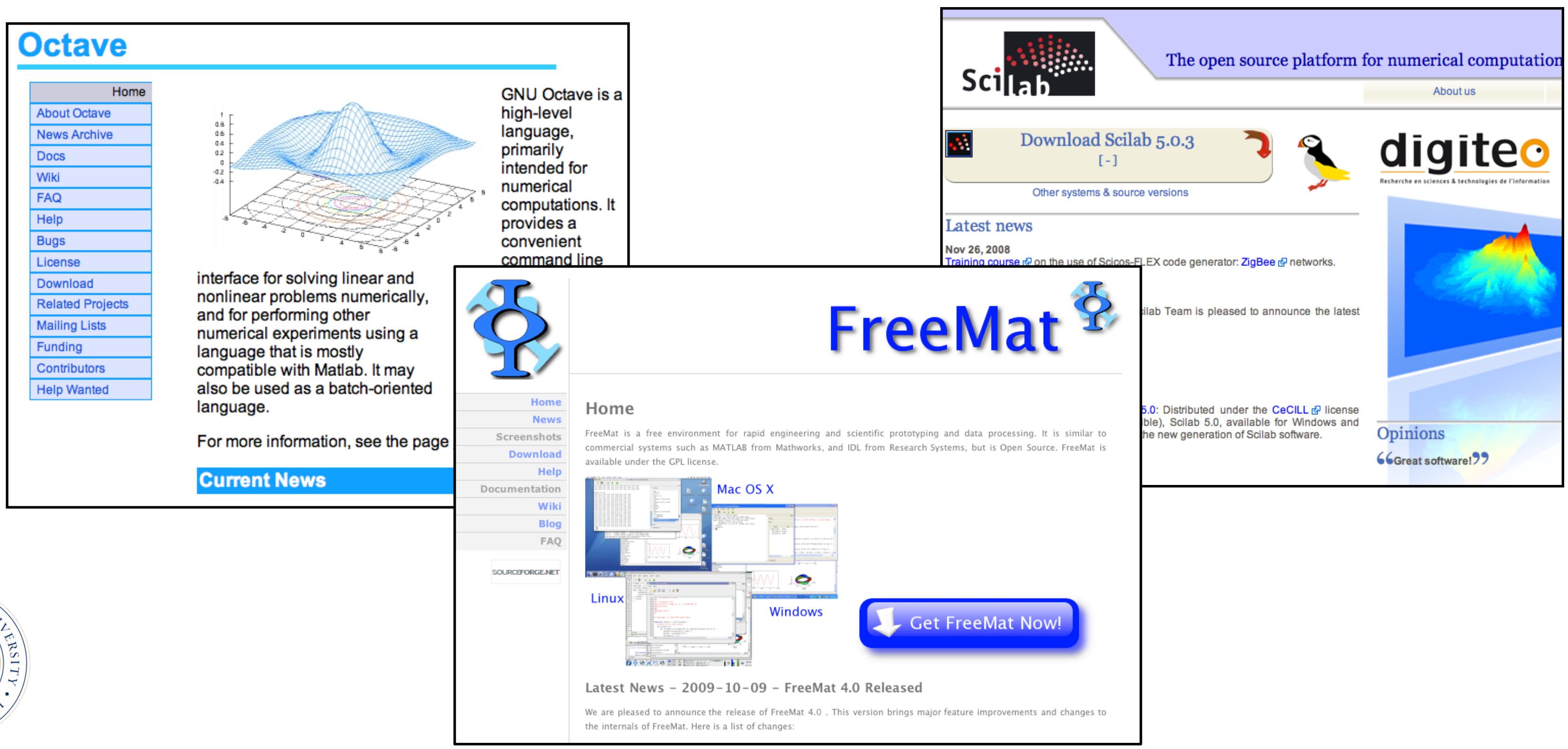
Matlab Tutorial

Overview

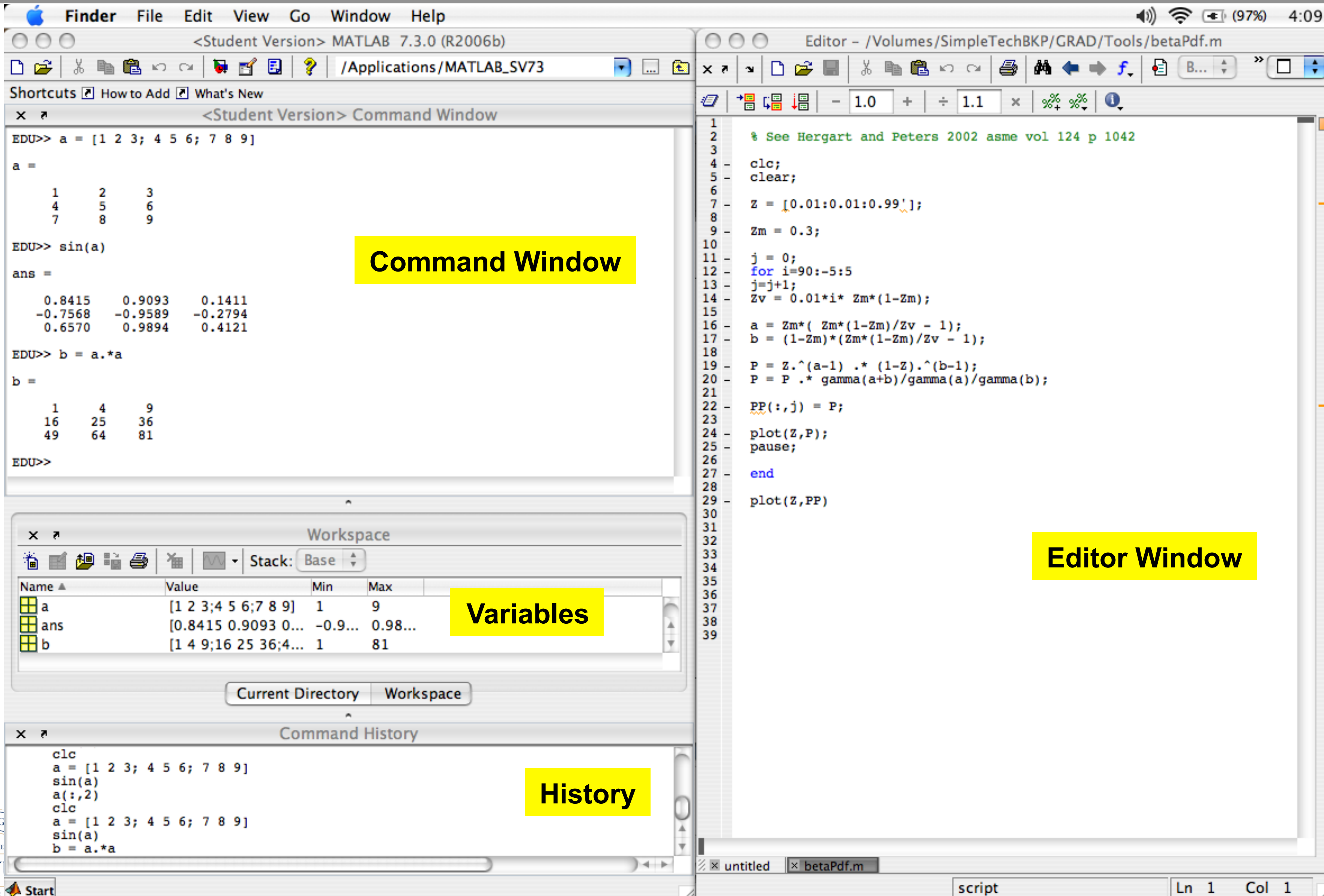
- Matlab is a programming language suited to numerical analysis and problems involving vectors and matricies.
  - Matlab = Matrix Laboratory
  - Many built in functions for solution of linear systems, interpolation, integration, solution of ODEs, etc.
  - Straightforward syntax
  - No need for external compilation/linking
- Built in 2D, 3D graphics, very flexible
- Can interface with C++, Java, Fortran
- Object oriented programming capabilities
- Graphical interface.
- Built-in debugging capability.
- Great for rapid programming/prototyping.
  - Excellent learning environment, ideas carry over to faster, more flexible (and complex) languages, such as C, Fortran.

FreeMat, Octave, Scilab

- Freemat, Octave, and SciLab are open source, Matlab-like variants
- Octave contains fewer features, but very similar syntax, and runs most Matlab scripts without modification.
  - Visualization is via gnuplot
- Scilab has a Matlab-like look and feel.
- Freemat has a nice interface, and good plotting capabilities.
- www.gnu.org/software/octave, www.scilab.org, http://freemat.sourceforge.net



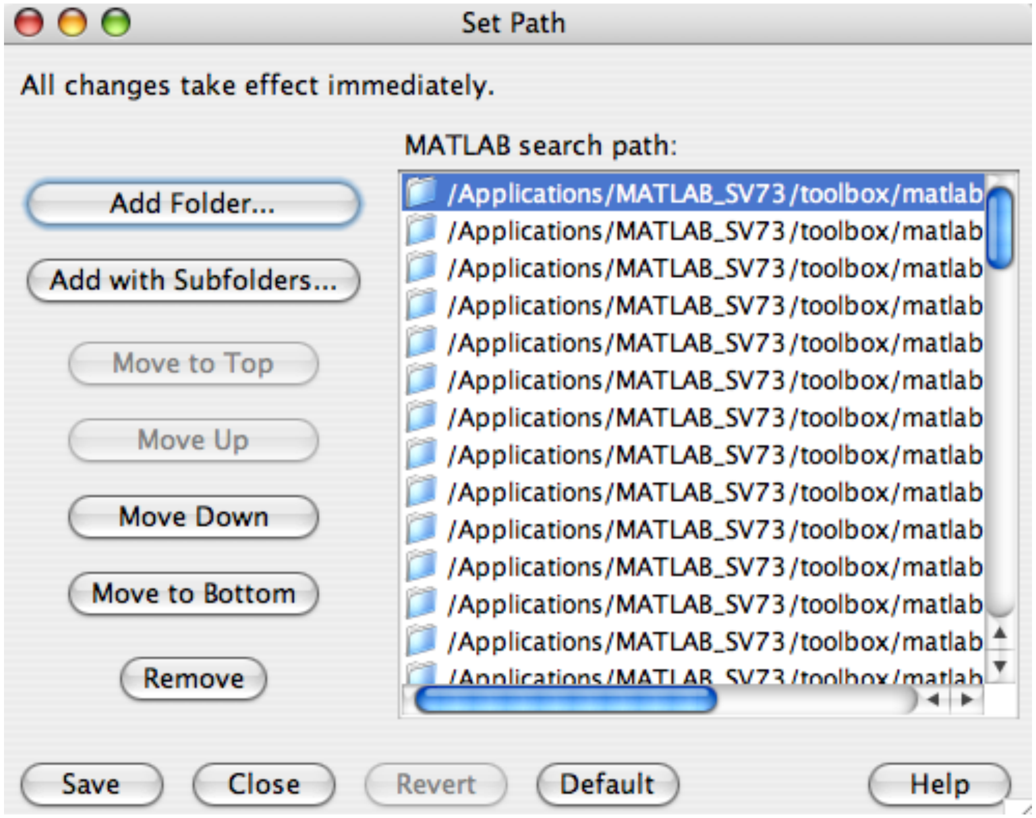
Environment



Matlab Search Path

- File >> set path
- Organize files into one or more place as you create them.
  - This goes for other environments/languages as well.
- Search path: EDU>> myvar

1. variable?
2. built-in function?
3. script file in current directory?
4. Matlab path?
5. Error



Defining Variables, Expressions

- Expressions are saved to ans
- Variables are case sensitive: no spaces, start with a letter.
- Semicolon supresses output to screen
- Variables defined, use who, whos

```
EDU>> who

Your variables are:

a_var   ans    b_var  c_var
```

- Special Vars:

  ans, beep, pi, eps, inf, NaN, i, j, nargin, nargout, realmin, realmax, bitmax, varargin, varargout

- Reserved Words

  for end if while function return elseif case otherwise switch continue else try catch global persistent break

- Operators: + - * / \ ^
- Comments: EDU>> a=b+c; % this is a comment

```
EDU>> 1+2

ans =

     3

EDU>> a_var = 1

a_var =

     1

EDU>> b_var = 2

b_var =

     2

EDU>> c_var = a_var + b_var

c_var =

     3

EDU>> c_var = a_var + b_var;
EDU>>
```

Vectors and Matricies

- Vectors, Matricies, Arrays are synonymous
- Enter elements between [...]
  - column elements separated by "," or " "
  - rows separated by ";"
  - transpose with single quote.
  - elements can be expressions
- Access elements with mat(index)
  - indexing starts at 1
  - Column notation
  - end
  - index can be an array
  - note index increment:
    - istart : inc : iend

```
EDU>> vec = [1 2 3];          % row vector
EDU>> vec = [1 2 3]';         % column vector
EDU>> vec = [1; 2; 3];        % column vector
EDU>> mat = [1 2; 3 4; 5 6];% 3x2 matrix
EDU>> vec = [1*pi 2*pi 3*pi]

vec =

    3.1416    6.2832    9.4248
```

```
EDU>> x = [1 2 3 4 5 6 7];
EDU>> x(3);
EDU>> x(3:5)

ans =

     3     4     5

EDU>> x(5:end)

ans =

     5     6     7

EDU>> x([2 3])

ans =

     2     3

EDU>> x(end:-1:5)

ans =

     7     6     5
```

Array Construction

Array Construction Summary

```
EDU>> x = [2 2*pi sqrt(2) 2-3];
EDU>>
EDU>> x = first:last;
EDU>>
EDU>> x=first:increment:last;
EDU>>
EDU>> x=linspace(first,last,n);
EDU>>
EDU>> x=logspace(first,last,n);
```

- Scalars operate directly on array elements:
  EDU>> g = [1 2 3; 4 5 6; 7 8 9];
  EDU>> g-2; 2*g-1, etc.
- Array-Array operations are as in matrix algebra
  EDU>> h = [5 6 7; 8 9 10; 11 12 13];
  EDU>> g+h;   2*g+h; etc
- Matrix multiplication:
  EDU>> g*h;
- Matrix element operations:
  EDU>> g.*h; g.^h; sin(g); 1./g; g.^2; etc.

Standard arrays

```
EDU>> a=zeros(2,3)

a =

     0     0     0
     0     0     0

EDU>> a = ones(2,3)

a =

     1     1     1
     1     1     1

EDU>> a = rand(2,3)

a =

    0.1988    0.7468    0.9318
    0.0153    0.4451    0.4660

EDU>> a = eye(3)

a =

     1     0     0
     0     1     0
     0     0     1

EDU>> a = [1 2 3];
EDU>> b = diag(a,1)

b =

     0     1     0     0
     0     0     2     0
     0     0     0     3
     0     0     0     0
```

More Array Operations

- Automatic expansion possible
- Reshape function operates on columns.
- Automatic deletion
- repmat function to create new matricies from existing matrices.
- Other functions
  - sort, find, flipud, fliplr, rot90, max, min.
  - length, size, numel, A(:)

```
EDU>> a = [1,2,3;4,5,6];
EDU>> a(3,4)=-1

a =

     1     2     3     0
     4     5     6     0
     0     0     0    -1

EDU>> a = reshape(a,4,3)

a =

     1     5     0
     4     0     0
     0     3     0
     2     6    -1

EDU>> b = a; b(:,2)=[]

b =

     1     0
     4     0
     0     0
     2    -1

EDU>> repmat(b,1,2)

ans =

     1     0     1     0
     4     0     4     0
     0     0     0     0
     2    -1     2    -1
```

m-files

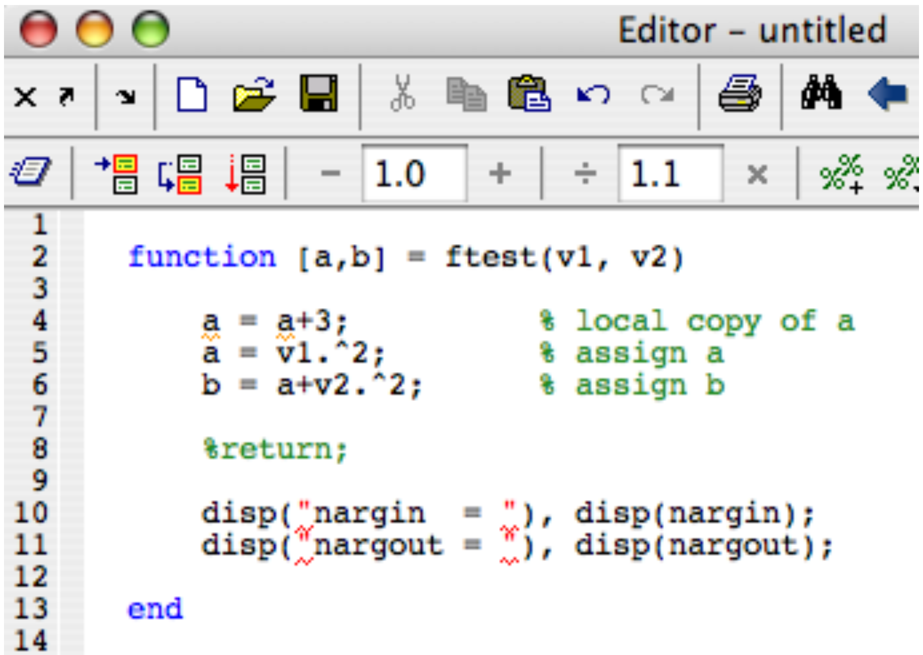
- m-files are script files containing batches of matlab commands
  - save and edit myfile.m
  - run EDU>> myfile to execute commands.
  - these files constitute the program and are the usual mode of use except for simple jobs at the command prompt.
  - files can call other files for code organization
    - think of the execution of commands as if typed directly at the command prompt.
  - useful functions: clc, clear, tic, toc, date, diary, format

Functions

- Purpose of functions.
  - Organize code
  - Reuse functionality
    - simplifies code
    - easier to maintain
- Variable Scope
  - Variables are local to the function, and can only be used in the function.
  - `global` statement allows variable access.
    - `global` var1 var2 ...
    - naming
  - `persistent`
- Function content
  - input arguments
  - return values
- Function file
  - name
  - subfunctions
  - M-file calls

Function Syntax

```
function a = functionName(arg1, arg2, ...)
function [a, b, c] = functionName(arg1, arg2, ...)
```




```
[x,y] = ftest(2,3);
[x,y] = ftest(1:4, 7:10);
```

- Name the function M-file functionName.m
- Input arguments
  - pass in when called
  - can be any type (e.g. an array)
  - can pass fewer than needed
- Return values
  - these are the outputs
  - one or many
  - again any type
- `varargin, varargout`

Function Documentation

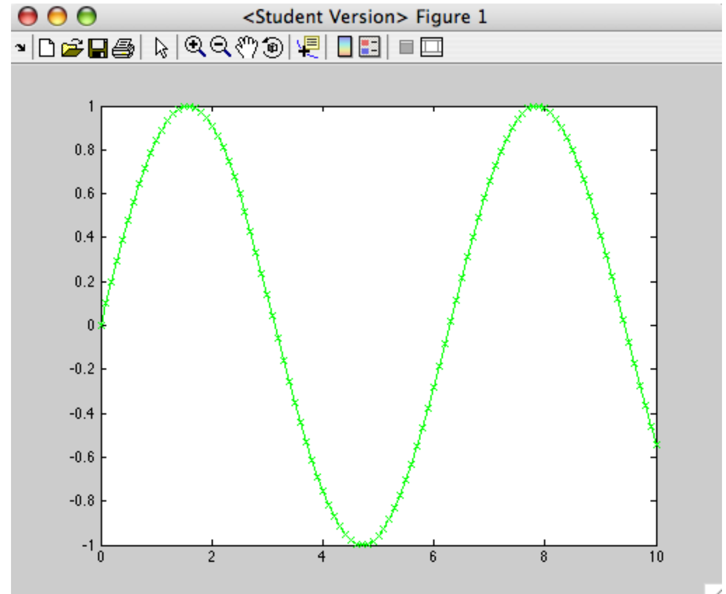
- Documenting functions is good code practice
  - Eases maintenence to you and others
- Purpose of the function
- Example of useage
- What are the inputs/ outputs
- Any issues, limitations, suggested improvements.
- Initial continuous comments are displayed with `help funcName`

```
EDU>> open linspace

EDU>> help linspace
```

```
function y = linspace(d1, d2, n)
%LINSPACE Linearly spaced vector.
%   LINSPACE(X1, X2) generates a row vector of 100 linearly
%   equally spaced points between X1 and X2.
%
%   LINSPACE(X1, X2, N) generates N points between X1 and X2.
%   For N < 2, LINSPACE returns X2.
%
%   Class support for inputs X1,X2:
%      float: double, single
%
%   See also LOGSPACE, :.

%   Copyright 1984-2004 The MathWorks, Inc.
%   $Revision: 5.12.4.1 $  $Date: 2004/07/05 17:01:20 $

if nargin == 2
    n = 100;
end

n = double(n);
y = [d1+(0:n-2)*(d2-d1)/(floor(n)-1) d2];floo
```

Visualization: 2-D Plots

- `x=1:0.1:10;`
- `plot(x)`
- `plot(x,sin(x))`
- General: `plot(x,y,'S')`
  - `S` is color, symbol, line style
  - Example: `plot(x,y,'gx--');`



Color

b	blue
g	green
r	red
c	cyan
m	magenta
y	yellow
k	black
w	white

Symbol

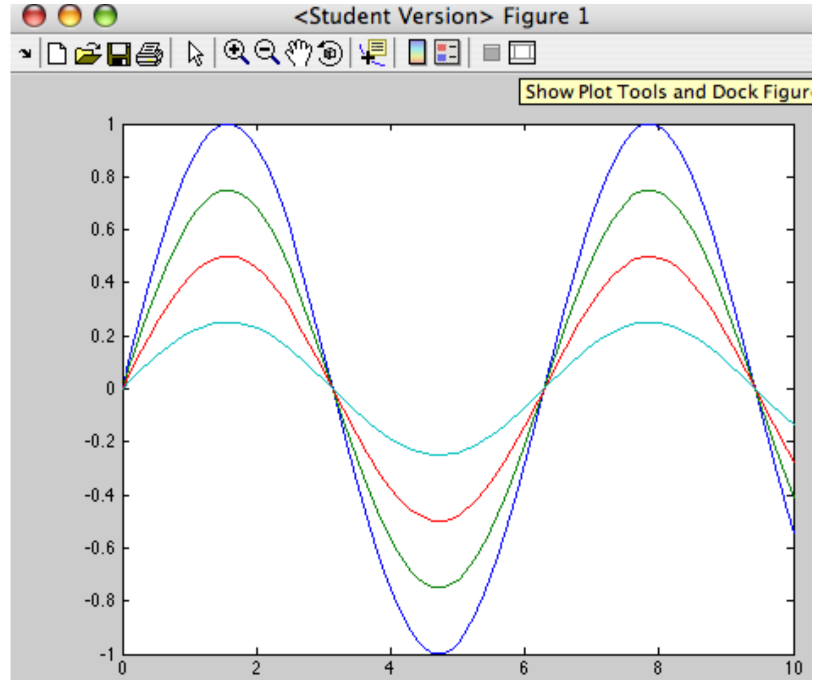
.	point
o	circle
x	x-mark
+	plus
*	star
s	square
d	diamond
v	triangle (down)
^	triangle (up)
<	triangle (left)
>	triangle (right)
p	pentagram
h	hexagram

Line Style

-	solid
:	dotted
-.	dashdot
--	dashed
(none)	no line

Multiple Plots

- Three methods for multiple plots
  1. hold on, hold off
  2. plot x and columns of y
  3. successive triplets of plot arguments.



1

```
EDU>> clf
EDU>> hold on;
EDU>> plot(x,sin(x))
EDU>> plot(x,0.75*sin(x),'k')
EDU>> plot(x,0.5*sin(x),'g')
EDU>> plot(x,0.25*sin(x),'r')
EDU>>
```

2

```
EDU>> clf;
EDU>> hold off;
EDU>> plot(x,[sin(x) 0.75*sin(x) 0.5*sin(x) 0.25*sin(x)])
EDU>>
```

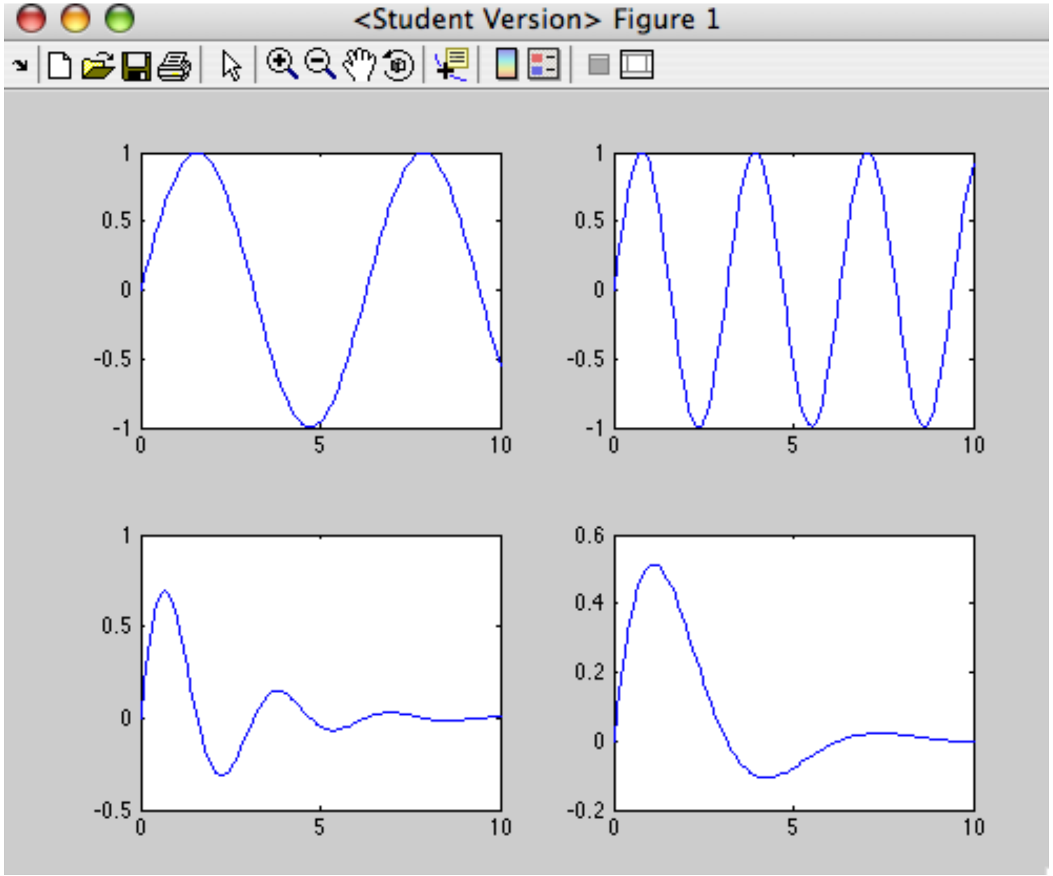
3

```
EDU>> clf;
EDU>> plot(x, sin(x), 'k', x, 0.*5*sin(x), 'g')
EDU>>
```

Subplot

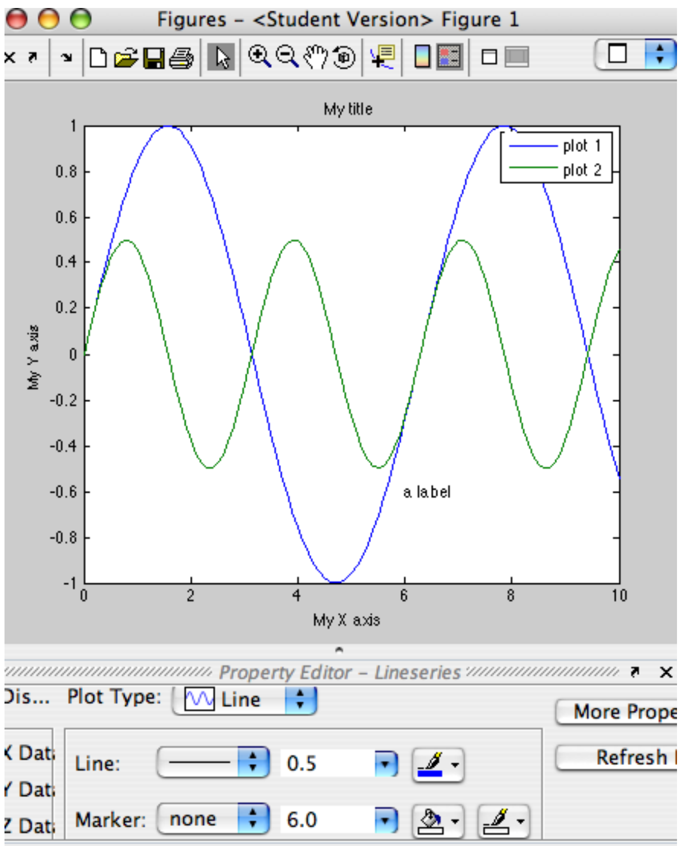
- Subplot allows multiple plots in a matrix format
- `subplot(nx,ny,pos)` activates an `nx` by `ny` matrix of plots with plot `pos` selected

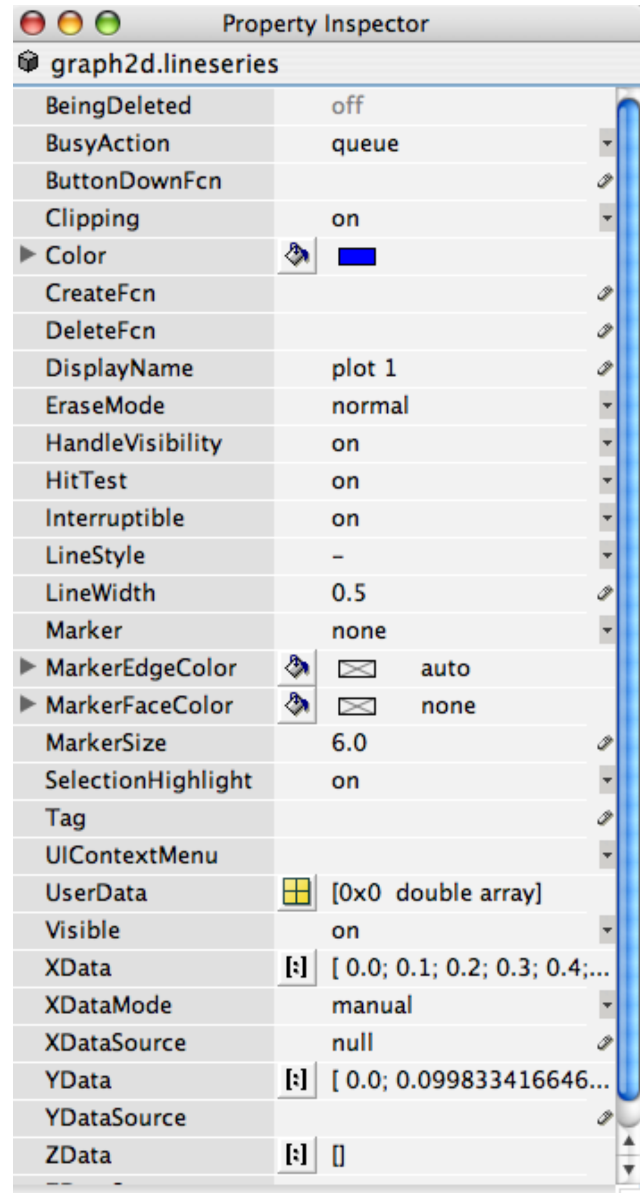
```
EDU>> clf
EDU>> subplot(2,2,1);
EDU>> plot(x,sin(x))
EDU>> subplot(2,2,2);
EDU>> plot(x,sin(2*x))
EDU>> subplot(2,2,3);
EDU>> plot(x,exp(-0.5*x).*sin(2*x))
EDU>> subplot(2,2,4);
EDU>> plot(x,exp(-0.5*x).*sin(x))
EDU>>
```



Labeling, Formatting

```
EDU>> plot(x,sin(x), x, 0.5*sin(2*x))
EDU>> xlabel('My X axis');
EDU>> ylabel('My Y axis');
EDU>> title('My title');
EDU>> legend('plot 1', 'plot 2')
EDU>> text(6,-0.6,'a label')
EDU>> |
```





Other Plotting Commands

- grid on; grid off;
- axis auto (manual tight, fill, on, off, square, etc.)
- axis([xmin, xmax, ymin, ymax]); or axis(array);
- xlim([xmin, xmax]), ylim ([ymin, ymax]);
- figure;
- figure(n)
- close
- close(n)
- semilogx; semilogy; loglog
- surf(X,Y,Z), mesh(X,Y,Z)
  - shading flat (or interp ...)

```
EDU>> x = 1:3; y = 0.1:0.1:0.5;
EDU>> [X,Y] = meshgrid(x,y);
EDU>> size(X)

ans =

     5     3
```

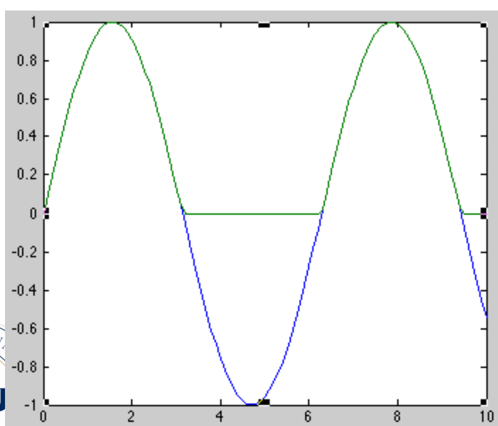
- Latex capable text formatting:
- \alpha, \beta, \gamma, \delta, etc.
- \it italic
- ^ superscript
- _ subscript
- texlabel('lambda = 3*alpha')
- title('{\itAe}^{\alpha \itt}sin\beta{\itt} \alpha<<\beta')

$$Ae^{-\alpha t}\sin\beta t\ \alpha<<\beta$$

Conditionals

- Relational Operators:
  - <, <=, >, >=, ==, ~=
  - (a+b) == (c+d)
  - B - (A>2)
- Logical Operators:
  - and: &, or: |, not: ~
  - (a>2) & (a<6)

```
x = 0:0.1:10;
y = sin(x);
z = (y>=0).*y;
plot(x,y, x,z)
```



- Conditionals:

```
if expression
    (command)
end
```

```
if expression
    (command)
else
    (command)
end
```

```
if expression
    (command)
elseif expression
    (command)
else
    (command)
end
```

- Switch-Case

```
switch expression
    case test_1
        (commands)
    case {test_2, test_2}
        (commands)
    otherwise
        (commands)
end
```

```
x = 2.7;
units = 'm';
switch units
   case {'inch', 'in'}
      y = x*2.54;
   case {'feet', 'ft'}
      y = x*2.54/12;
   case {'meter', 'm'}
      y = x/100;
   otherwise
      disp(['Unknown Units: ' units])
      y = nan;
end
```

Loops

- loops offer explicit control over element assignment and other operations
- Preallocate arrays before loops.
- Loops can be nested
- `break` statement
- Avoid for loops whenever there is an equivalent array approach.
  - *Vectorized* solutions are often orders of magnitude faster!
  - less typing, easier to read, more intuitive
- While loops execute till some expression holds

```
for x = array
   (commands)
end
```

```
for i = 1:10
   x(i)=sin(i)
end
```

```
for i = 1:10
   for j= 1:3
      A(i,j) = i^2 + j^2;
   end
end
```

```
i = 1:10;
j = 1:3;
[ii,jj] = meshgrid(i,j);
A = ii.^2 + jj.^2;
```

```
tend = 10;
t    = 0;
dt   = 1.1;
while t < tend
    (commands)
    t = t + dt;
end
```

Basic File I/O

- `save -ASCII filename x y`
  - saves variables x, y to the file filename
    - if omitted, all variables saved
  - `-ASCII` writes a text file
    - if omitted, a binary file results (smaller)
      - file called filename.mat
- `load filename x y`
  - load the saved varialbes
  - if `x y` is omitted, all variables are loaded
- `dlmread, dlmwrite, textread`, others
- `fopen, fclose, fread, fwrite, fscanf, fprintf, sprintf, sscanf`, others
  - `myfile = ‘filelist’`
  - `f1 = fopen(myfile);`
  - `file = fscanf(f1, ‘%s’, 1)`

File I/O Example

```
clc; clear;

myfile = 'CO2List';

f1 = fopen(myfile);

i = 1;
while(1);
    file = fscanf(f1, '%s', 1);
    if(feof(f1)) break; end
    flist{i,1} = file;
    file = strrep(file, '_', ' ');
    times(i,1) = sscanf(file, '%*s %*s %f');
    i = i+1;
end
fclose(f1);
```

```
[nfiles, d1] = size(flist);

for ifi=1:nfiles
    f1 = fopen(flist{ifi,1});
    ln = fgetl(f1);
    i=1;
    while(~feof(f1))
        ln = fgetl(f1);
        A(i,:) = [sscanf(ln,'%f')]';
        i = i+1;
    end
    fclose(f1);
    if(ifi==1)
        mixf = A(:,1);
    end
    data(:,ifi) = A(:,6);
    clear A;
end

[X,Y] = meshgrid(mixf, times);
surf(X,Y,data');
```

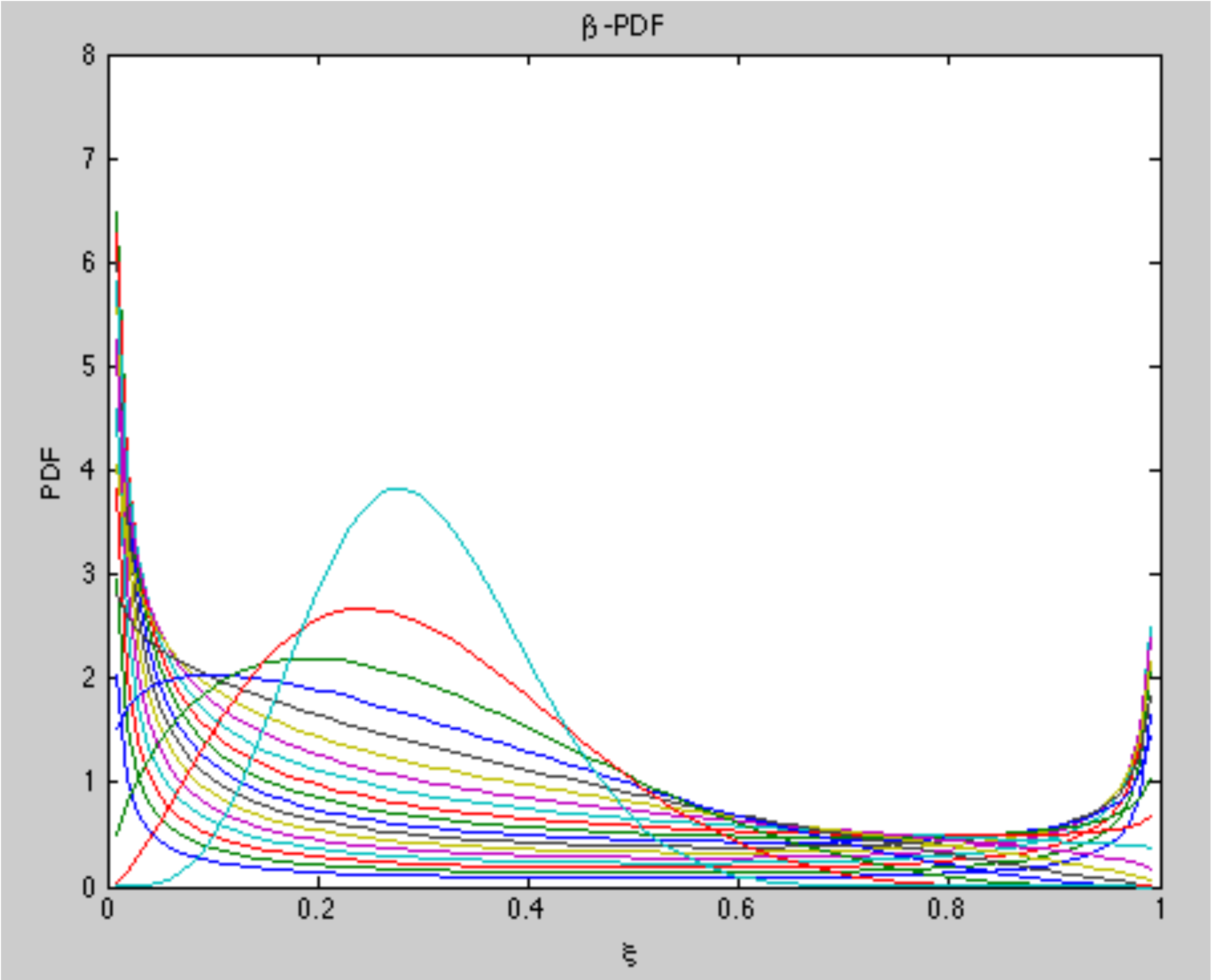
β PDF Example

- The beta-PDF represents the extent of mixing between two pure streams in turbulent flows.
- These streams are often fuel and oxidizer.
- For segregated streams, two delta functions result.
- For perfect mixing, one delta function exists.
- In between, a range of states exists

$$\bar{\phi}(\xi) = \int_{\xi} \phi(\xi) P(\xi) d\xi$$

```
% Script computes the beta-pdf for a range of variances
% for a given value of the mean
% See Hergart and Peters 2002 asme vol 124 p 1042

clc;                          % clear the screen
clear;                        % clear existing variables

Z = [0.01:0.01:0.99'];        % set the abscissa

Zm = input('Enter Zm: ');     % prompt user for mean mixf

j = 0;                        % initialize stepper
for i=90:-5:5
    j=j+1;
    Zv = 0.01*i* Zm*(1-Zm);   % set the variance

    a = Zm*( Zm*(1-Zm)/Zv - 1);                  % a parameter
    b = (1-Zm)*(Zm*(1-Zm)/Zv - 1);               % b paramter

    P = Z.^(a-1) .* (1-Z).^(b-1);                % intermediate PDF
    P = P .* gamma(a+b)/gamma(a)/gamma(b);       % PDF

    PP(:,j) = P;                                 % save the PDF to PP

    plot(Z,P);                                   % intermediate plot
    xlabel('\xi');
    ylabel('PDF');
    axis([0 1 0 8]);
    pause;

end

plot(Z,PP);                                      % plot the whole thing
title('\beta -PDF');
xlabel('\xi');
ylabel('PDF');
axis([0 1 0 8]);
```

β PDF Example



Summary

- Matlab provides a wealth of functionality for small to intermediate size projects
- Open source variants available
- Advanced visualization capabilities.
- Highly extensible
- Relatively simple syntax. (a higher level language).
- Extensible, object oriented.
- Many toolboxes available for more advanced, problem specific work
- Search the web for more tutorials, books, examples