Imperial College London

DE2 Electronics 2

Tutorial 1

Learning Matlab

Peter Cheung
Dyson School of Design Engineering

URL: www.ee.ic.ac.uk/pcheung/teaching/DE2_EE/
E-mail: p.cheung@imperial.ac.uk

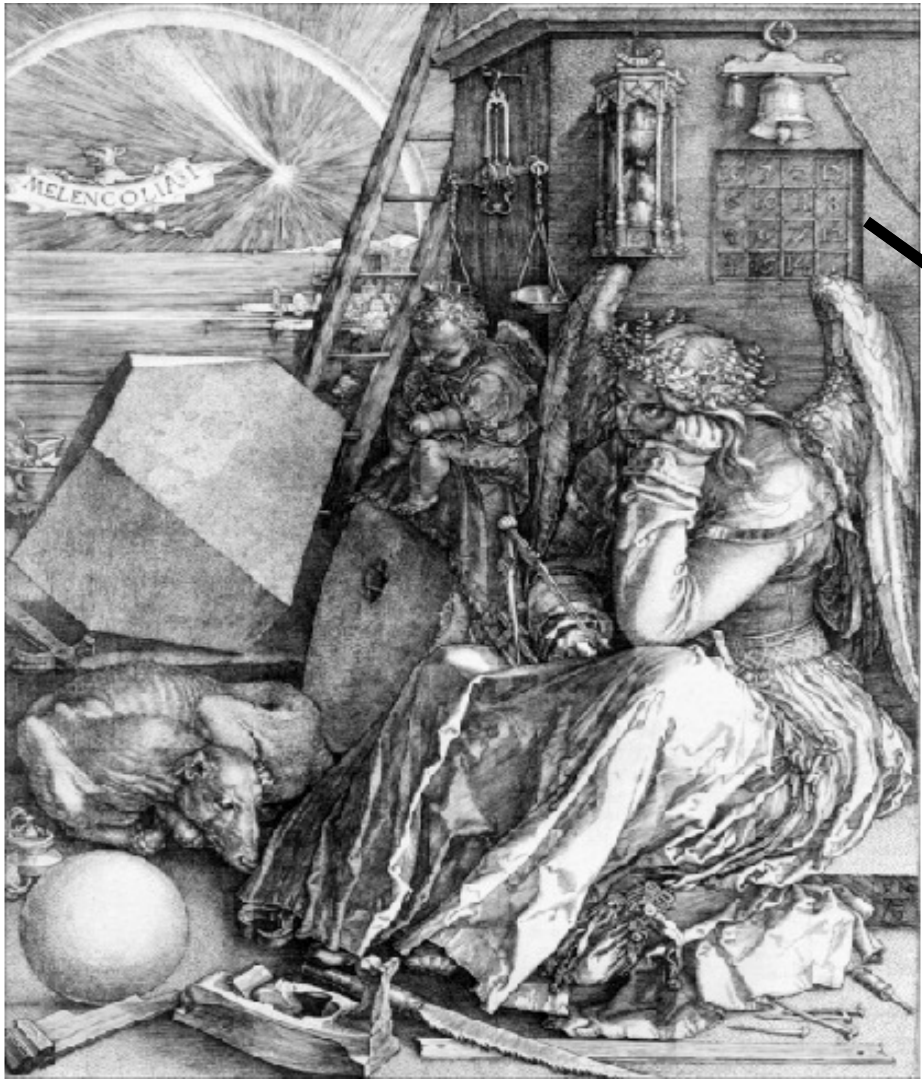
Introduction to MATLAB

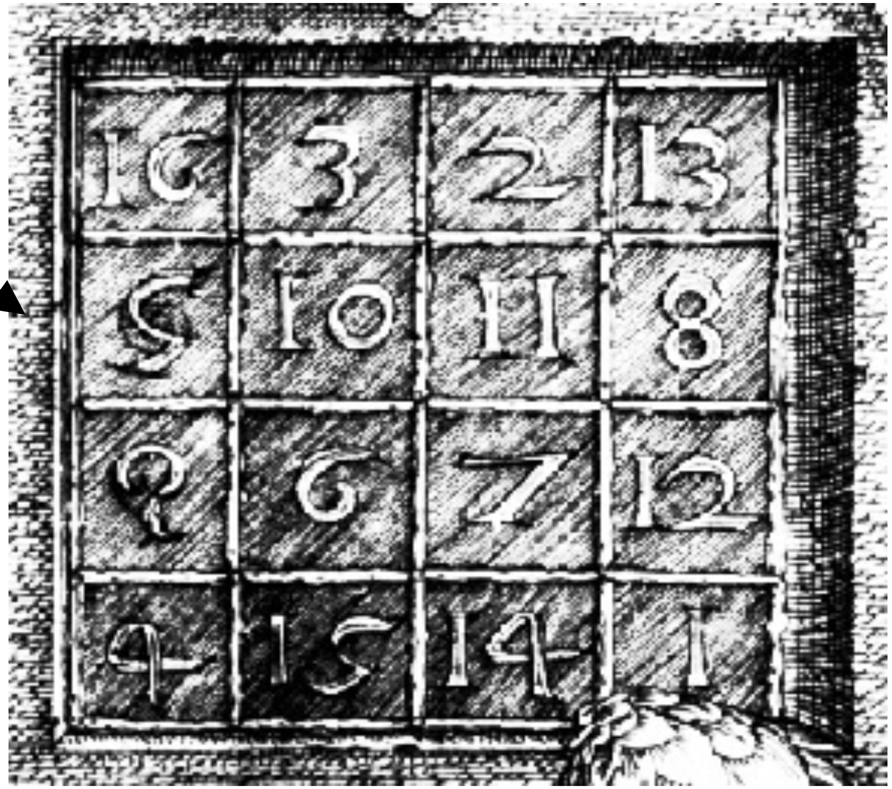
- **MATLAB** is a high-performance language for ***technical computing***. It integrates computation, visualization, and programming in an easy-to-use environment. Typical uses include:
  - Math and computation
  - Algorithm development
  - Modeling, simulation, and prototyping
  - Data analysis, exploration, and visualization
  - Scientific and engineering graphics
- MATLAB is an ***interactive*** system whose basic data element is an ***array*** that does not require dimensioning. This allows you to solve many technical computing problems, especially those with ***matrix*** and ***vector*** formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or Fortran.

Five Parts of Matlab

- **The MATLAB language**
  - High-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features
- **The MATLAB working environment**
  - Facilities for managing the variables and importing and exporting data
  - Tools for developing, managing, debugging, and profiling M-files
- **Handle Graphics**
  - Two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics
  - Graphical User Interface functions
- **The MATLAB mathematical function library**
- **The MATLAB Application Program Interface (API)**
  - Allows you to write C and Fortran programs that interact with MATLAB

Entering Matrices (1) - Magic Square





- Engraving by Albrecht Dürer, German artist and mathematician in 1514.

Entering Matrices (2) - Method 1:Direct entry

- **4 ways of entering matrices in MATLAB**:
  - Enter an explicit list of elements
  - Load matrices from external data files
  - Generate matrices using built-in functions
  - Create matrices with your own functions in M-files
- **Rules of entering matrices**:
  - Separate the elements of a row with ***`blanks`*** or commas
  - Use a ***`semicolon`*** “`;`“ to indicate the end of each row
  - Surround the entire list of elements with ***`square brackets`***, `[ ]`
- To enter Dürer's matrix, simply type:

```
» A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
```

- MATLAB displays the matrix you just entered,

```
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
```

No need to define or declare size of A

Entering Matrices (3) - as lists

- Why is this a magic square? Try this in Matlab :-

Result in row vector variable `ans`

```
» sum(A)

ans =

     34     34     34     34

» A'
ans =
     16      5      9      4
      3     10      6     15
      2     11      7     14
     13      8     12      1
» sum(A')'
ans =       34
            34
            34
            34
```

Compute the sum of each column in A

Transpose matrix A

Compute the sum of each row in A

Result in cloumn vector variable `ans`

Entering Matrices (4) - subscripts

- `A(i,j)` refers to element in row `i` and column `j` of `A` :-

row col

```
» A(4,2)
ans = 15
» A(1,4) + A(2,4) + A(3,4) + A(4,4)
ans = 34
» X = A;
» X(4,5) = 17
X =
    16     3     2    13     0
     5    10    11     8     0
     9     6     7    12     0
     4    15    14     1    17
```

Slow way of finding sum of column 4

Make another copy of A in X
';' suppress output

Add one element in column 5, auto increase size of matrix

Entering Matrices (5) - colon : Operator

- ':' used to specify range of numbers

```
» 1:10
ans = 1   2   3   4   5   6   7   8   9   10
» 100:-7:50
ans = 100   93   86   79   72   65   58   51
» 0:pi/4:pi
ans = 0   0.7854   1.5708   2.3562   3.1416

» A(1:k,j);
» sum(A(1:4,4))
ans = 34
» sum(A(:,end))
ans = 34
```

start

end

incr

'0' to 'pi' with incr. of 'pi/4'

First k elements of the j[th] column in A

Short-cut for "all rows"

last col

Expressions & built-in functions

```
» rho = (1+sqrt(5))/2
rho =  1.6180

» a = abs(3+4i)
a =  5

» z = sqrt(besselk(4/3,rho-i))
z = 0.3730+ 0.3214i

» huge = exp(log(realmax))
huge =  1.7977e+308

» toobig = pi*huge
toobig = Inf
```

Elementary functions

Complex number

Special functions

Built-in constants (function)

- `pi` 3.14159265
- `I or j` Imaginary unit, -1
- `eps` FP relative precision, 2^{-52}
- `realmin` Smallest FP number, 2^{-1022}
- `realmax` Largest FP number, $(2-)2^{1023}$
- `Inf` Infinity
- `NaN` Not-a-number

Entering Matrices (6) - Method 2: Generation

```
» Z = zeros(2,4)
Z =   0       0       0       0
      0       0       0       0

» F = 5*ones(3,3)
F =   5       5       5
      5       5       5
      5       5       5

» N = fix(10*rand(1,10))
N = 4       9       4       4

» R = randn(4,4)
R =   1.0668   0.2944   0.6918  -1.4410
      0.0593  -1.3362   0.8580   0.5711
     -0.0956   0.7143   1.2540  -0.3999
     -0.8323   1.6236  -1.5937   0.6900
```

Useful Generation Functions

- `zeros` — All zeros
- `ones` — All ones
- `rand` — Uniformly distributed random elements between (0.0, 1.0)
- `randn` — Normally distributed random elements, mean = 0.0, var = 1.0

Entering Matrices (7) - Method 3 & 4: Load & M-File

magik.dat

```
16.0  3.0  2.0 13.0
 5.0 10.0 11.0  8.0
 9.0  6.0  7.0 12.0
 4.0 15.0 14.0  1.0
```

```
» load magik.dat
```

Read data from file into variable `magik`

Three dots (...) means continuation to next line

```
» magik
```

.m files can be run by just typing its name in Matlab

magik.m

```
A = [ ...
16.0  3.0  2.0 13.0
 5.0 10.0 11.0  8.0
 9.0  6.0  7.0 12.0
 4.0 15.0 14.0  1.0];
```

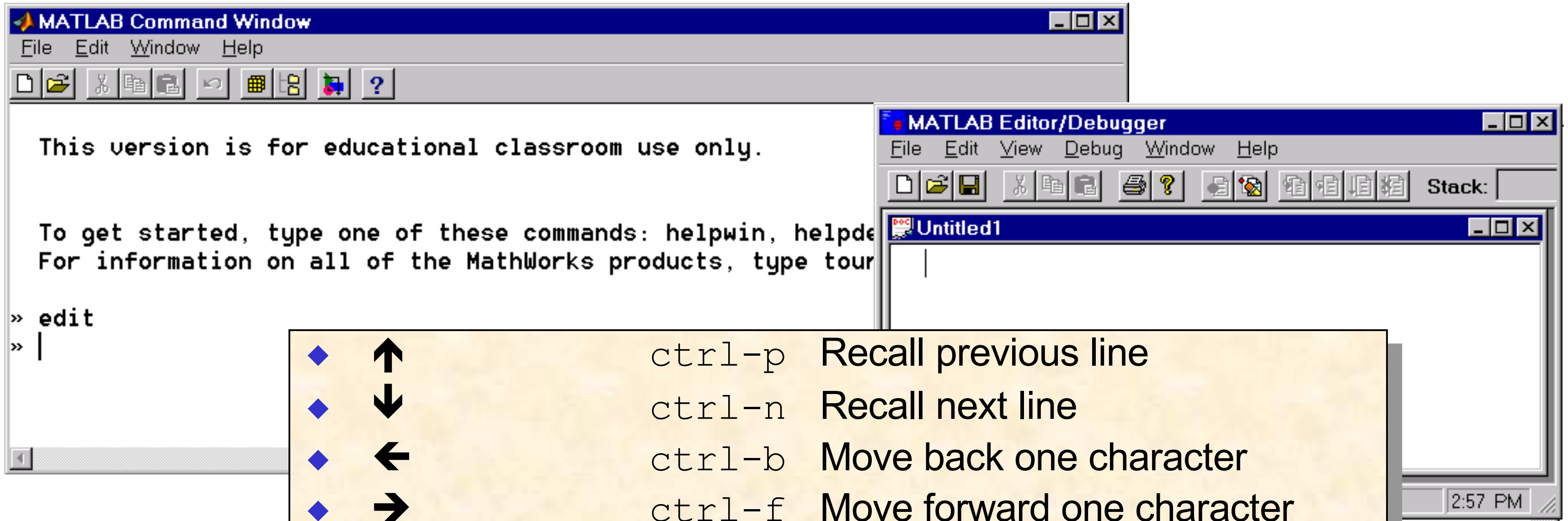
Entering Matrices (8) - Concatenate & delete

```
» B = [A  A+32; A+48  A+16]
B =
    16     3     2    13    48    35    34    45
     5    10    11     8    37    42    43    40
     9     6     7    12    41    38    39    44
     4    15    14     1    36    47    46    33
    64    51    50    61    32    19    18    29
    53    58    59    56    21    26    27    24
    57    54    55    60    25    22    23    28
    52    63    62    49    20    31    30    17
```

2nd column deleted

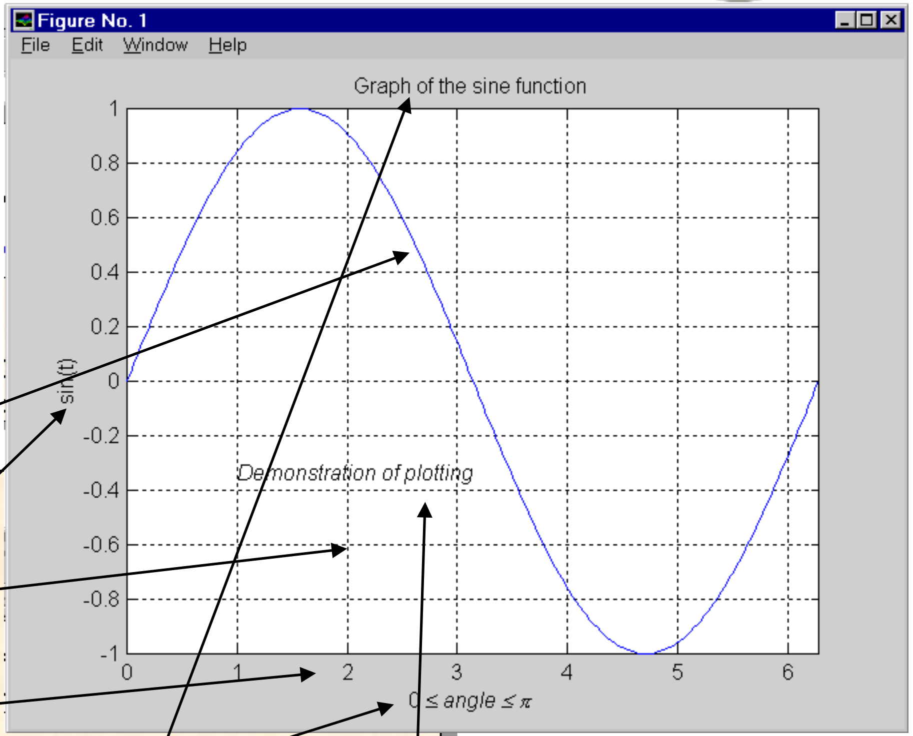
```
» X = A;
» X(:,2) = []
X =
    16     2    13
     5    11     8
     9     7    12
     4    14     1
```

Command Window



↑	ctrl-p	Recall previous line
↓	ctrl-n	Recall next line
←	ctrl-b	Move back one character
→	ctrl-f	Move forward one character
ctrl - →	ctrl-r	Move right one word
ctrl - ←	ctrl-l	Move left one word
home	ctrl-a	Move to beginning of line
end	ctrl-e	Move to end of line
esc	ctrl-u	Clear line
del	ctrl-d	Delete character at cursor
backspace	ctrl-h	Delete character before cursor
	ctrl-k	Delete to end of line

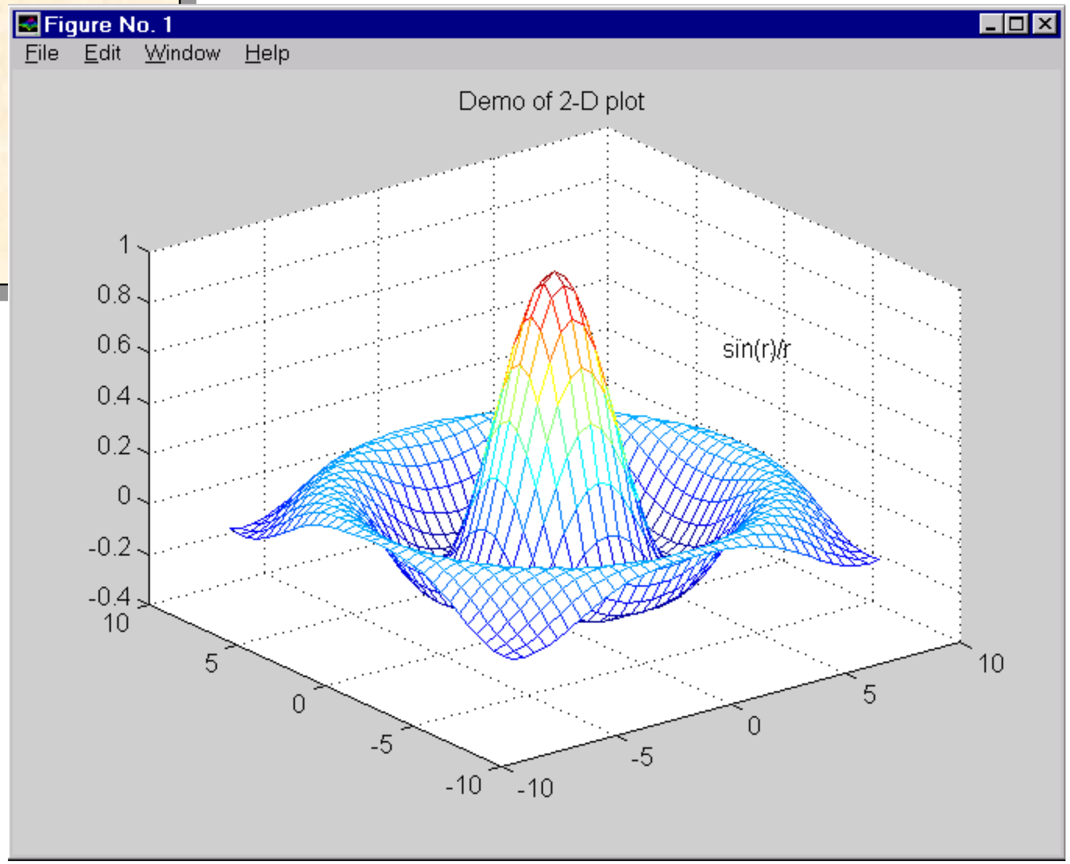
MATLAB Graphics(1) - Creating a Plot




```
» t = 0:pi/100:2*pi;
» y = sin(t);
» plot(t,y)
» grid
» axis([0 2*pi -1 1])
» xlabel('0 \leq \itangle \leq \pi')
» ylabel('sin(t)')
» title('Graph of the sine function')
» text(1,-1/3,'\it{Demonstration of plotting}')
```

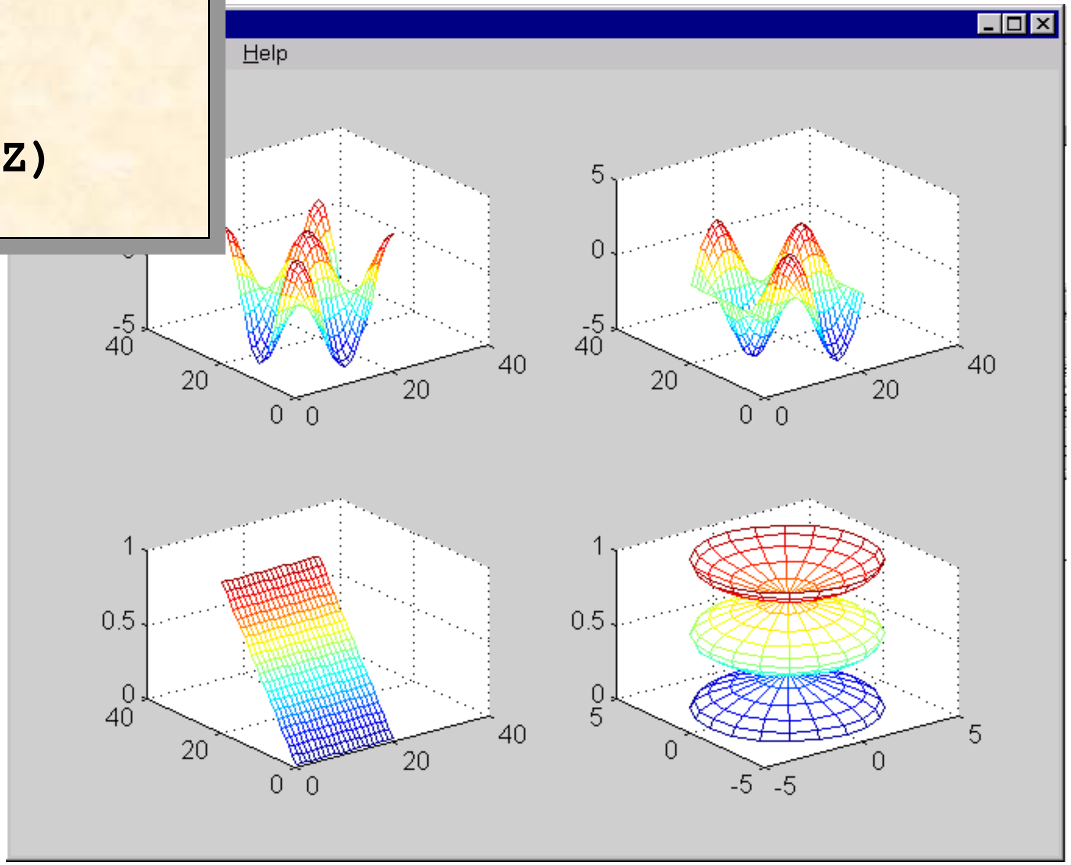
MATLAB Graphics(2) - Mesh & surface plots

```
» [X,Y] = meshgrid(-8:.5:8);
» R = sqrt(X.^2 + Y.^2) + eps;
» Z = sin(R)./R;
» mesh(X,Y,Z)
» text(15,10,'sin(r)/r')
» title('Demo of 2-D plot');
```



MATLAB Graphics(3) - Subplots

```
» t = 0:pi/10:2*pi;
» [X,Y,Z] = cylinder(4*cos(t));
» subplot(2,2,1); mesh(X)
» subplot(2,2,2); mesh(Y)
» subplot(2,2,3); mesh(Z)
» subplot(2,2,4); mesh(X,Y,Z)
```



MATLAB Graphics(3) - Subplots

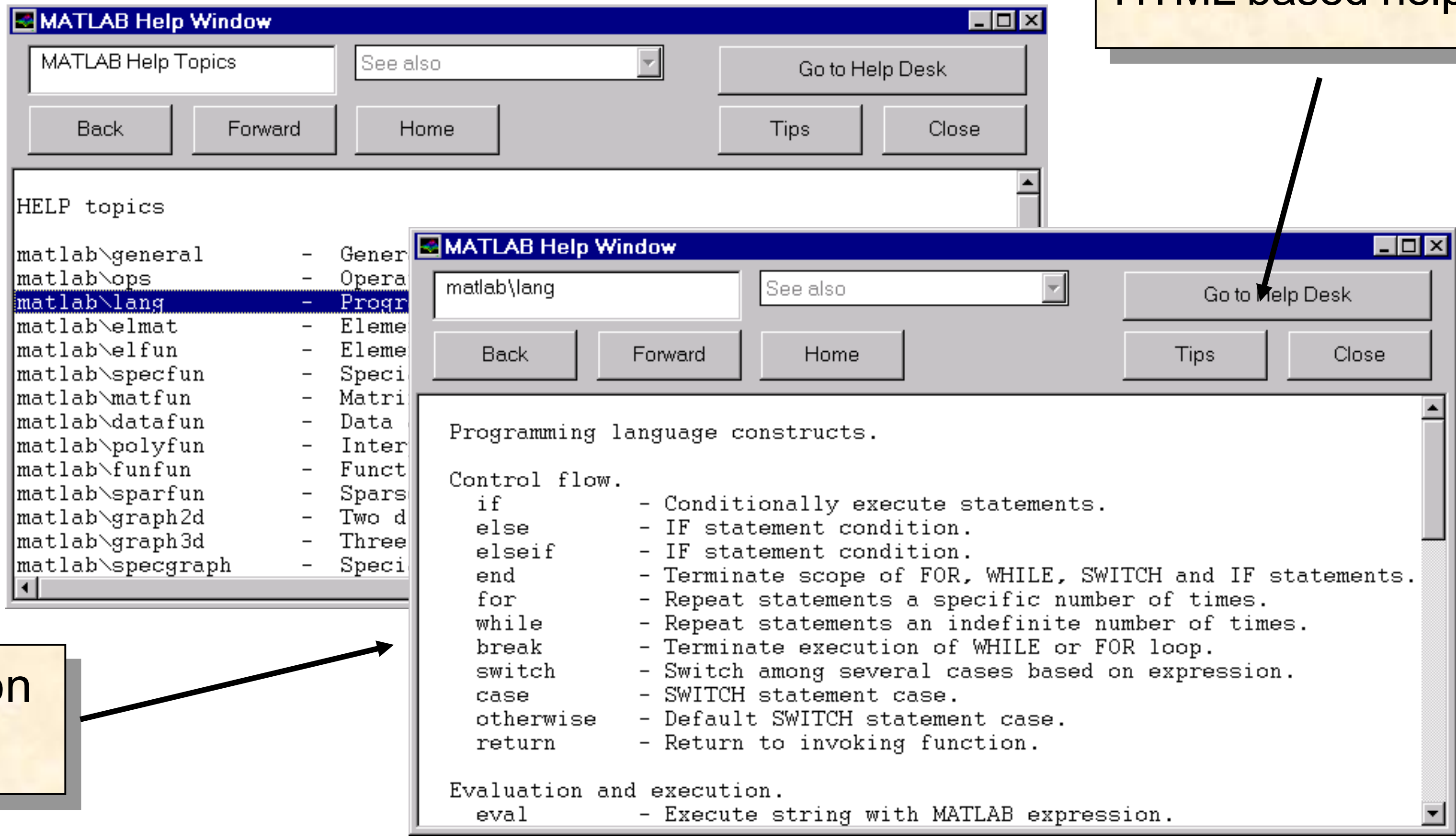
- Matlab official method: generate encapsulated postscript files -
  - » `print -depsc2 mesh.eps`

- My method:-
  - Use <***PrintScreen***> key (top right corner) to capture the plot on screen
  - Use MS Photo Editor or similar bit-map editing program to cut out the the plot that I want
  - Paste it into MS Word or MS PowerPoint or save it as .BMP/.GIF file
  - Resize as necessary
  - Fit as many as required on page
  - Type written description (or report) if needed
  - Print document to any printer (not necessarily postscript printer)

MATLAB Help and Online Tutorial

» **helpwin**

Click here for HTML based help



Double click on matlab\lang

Web-based MATLAB Help & Documentation

MATLAB Environment (1)

- **Managing Commands and Functions**
  - **addpath** Add directories to MATLAB's search path
  - **help** Online help for MATLAB functions and M-files
  - **path** Control MATLAB's directory search path
- **Managing Variables and the Workspace**
  - **clear** Remove items from memory
  - **length** Length of vector
  - **load** Retrieve variables from disk
  - **save** Save workspace variables on disk
  - **size** Array dimensions
  - **who, whos** List directory of variables in memory

MATLAB Environment (2)

- **Working with Files and the Operating Environment**
  - **cd** Change working directory
  - **delete** Delete files and graphics objects
  - **diary** Save session in a disk file
  - **dir** Directory listing
  - **edit** Edit an M-file
  - **!** Execute operating system command

Control flow in Matlab

- **MATLAB has five flow control constructs:**
  - if statements
  - switch statements
  - for loops
  - while loops
  - break statements
- **if statement**

```
if A > B
    'greater'
elseif A < B
    'less'
elseif A == B
    'equal'
else
    error('Unexpected situation')
end
```

>, < and == work with *scalars*, but NOT matrices

Matrix Comparison - Beware!

» `A=magic(4)`

A

$$A = \begin{matrix} 16 & 2 & 3 & 13 \\ 5 & 11 & 10 & 8 \\ 9 & 7 & 6 & 12 \\ 4 & 14 & 15 & 1 \end{matrix}$$

» `B = A'`

B

$$B = \begin{matrix} 16 & 5 & 9 & 4 \\ 2 & 11 & 7 & 14 \\ 3 & 10 & 6 & 15 \\ 13 & 8 & 12 & 1 \end{matrix}$$

`C=A>B`

$$\begin{matrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{matrix}$$

1 = true

`C=(A==B)`

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

0 = true

`C=A<B`

$$\begin{matrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{matrix}$$

Built-in Logic functions for matrices

- Several functions are helpful for reducing the results of matrix comparisons to scalar conditions for use with `if, including`
  - `isequal(A,B)` returns ‘1’ if A and B are identical, else return ‘0’
  - `isempty(A)` returns ‘1’ if A is a null matrix, else return ‘0’
  - `all(A)` returns ‘1’ if ***all*** elements A is non-zero
  - `any(A)` returns ‘1’ if ***any*** element A is non-zero

```
if isequal(A,B)
    'equal'
else
    'not equal'
end
```

Control Flow - Switch & Case

- Assume `method` exists as a string variable:

```
switch lower(method)
    case {'linear','bilinear'}
        disp('Method is linear')
    case 'cubic'
        disp('Method is cubic')
    case 'nearest'
        disp('Method is nearest')
    otherwise
        disp('Unknown method.')
end
```

Use otherwise to catch all other cases

Control Flow - For Loop

This makes it faster and use less memory

```
n = 4;
a = zeros(n,n)   % Preallocate matrix
for i = 1:n
    for j = 1:n
        H(i,j) = 1/(i+j);
    end
end
```

“Life is too short to spend writing for-loops”

- Create a table of logarithms:

```
x = 0;
for k = 1:1001
   y(k) = log10(x);
   x = x + .01;
end
```

- A vectorized version of the same code is

```
x = 0:.01:10;
y = log10(x);
```

Matrix versus Array Operations

A

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

Inner product matrix multiply

A * A

341	285	261	269
261	301	309	285
285	309	301	261
269	261	285	341

Element-by-element array multiply

A .* A

256	9	4	169
25	100	121	64
81	36	49	144
16	225	196	1

Matrix Operators

+	Addition or unary plus. A+B adds A and B. A and B must have the same size, unless one is a scalar. A scalar can be added to a matrix of any size.
-	Subtraction or unary minus. A-B subtracts B from A. A and B must have the same size, unless one is a scalar. A scalar can be subtracted from a matrix of any size.
*	Matrix multiplication. C = A*B is the linear algebraic product of the matrices A and B. For nonscalar A and B, the number of columns of A must equal the number of rows of B. A scalar can multiply a matrix of any size.
/	Slash or matrix right division. B/A is roughly the same as B*inv(A). More precisely, B/A = (A'\B')'. See \\.
\\	Backslash or matrix left division. If A is an n-by-n matrix and B is a column vector with n components, or a matrix with several such columns, then X = A\B is the solution to the equation $AX = B$.
^	Matrix power. X^p is X to the power p, if p is a scalar. If p is an integer, the power is computed by repeated multiplication.
'	Matrix transpose. A' is the linear algebraic transpose of A. For complex matrices, this is the complex conjugate transpose.

Array Operators

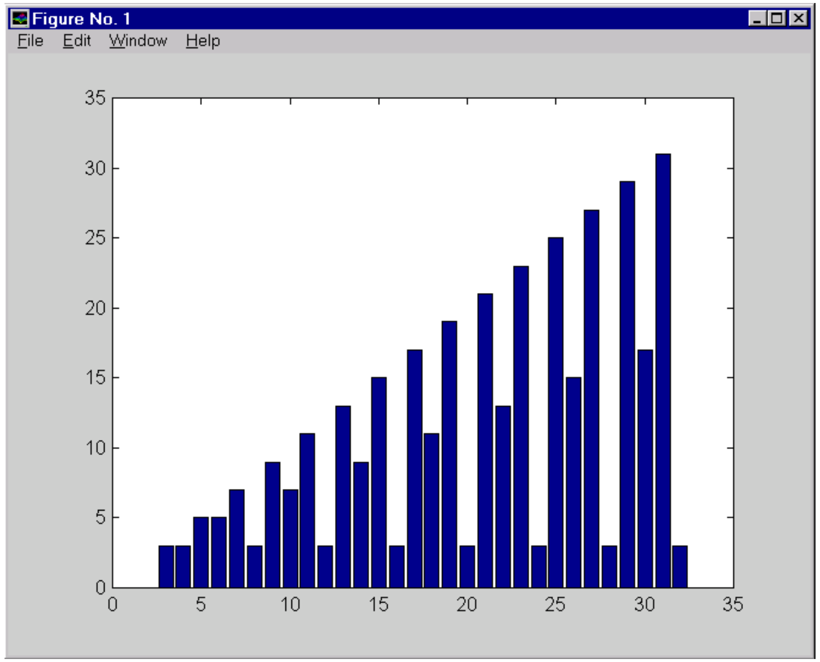
+	Element-by-element addition or unary plus.
-	Element-by-element subtraction or unary minus.
.*	Array multiplication. `A.*B` is the element-by-element product of the arrays `A` and `B`. `A` and `B` must have the same size, unless one of them is a scalar.
./	Array right division. `A./B` is the matrix with elements `A(i,j)/B(i,j)`. `A` and `B` must have the same size, unless one of them is a scalar.
.\	Array left division. `A.\B` is the matrix with elements `B(i,j)/A(i,j)`. `A` and `B` must have the same size, unless one of them is a scalar.
.^	Array power. `A.^B` is the matrix with elements `A(i,j)` to the `B(i,j)` power. `A` and `B` must have the same size, unless one of them is a scalar.
.'	Array transpose. `A.'` is the array transpose of `A`. For complex matrices, this does not involve conjugation.

M-files: Scripts and Functions

- **There are two kinds of M-files**:
  - Scripts, which do not accept input arguments or return output arguments. They operate on data in the workspace.
  - Functions, which can accept input arguments and return output arguments. Internal variables are local to the function.

Script magic_rank.m

```
% Investigate the rank of magic squares
r = zeros(1,32);
for n = 3:32
   r(n) = rank(magic(n));
end
r
bar(r)
```



Functions

Return variable

Define function name and arguments

function myfunct.m

```
function r = myfunct (x)
% Calculate the function:
%       r = x^3 - 2*x - 5
% x can be a vector

r = x.^3 - x.*2 -5;
```

% on column 1 is a comment

This is how plot on p.2-7 was obtained

```
» X = 0:0.05:3;
» y = myfunct (x);
» plot(x,y)
```

Scopes of variables

- All variables used inside a function are local to that function
- Parameters are passed in and out of the function explicitly as defined by the first line of the function
- You can use the keyword **global** to make a variable visible everywhere
- As a good programming practice, only use global variables when it is absolutely required

MATLAB Programming Style Guide (1)

- This Style Guideline is originally prepared by **Mike Cook**
  - The first line of code in **script m-files** should be indicate the name of the file.
  - The first line of **function** m-files has a **mandatory** structure. The first line of a function is a declaration line. It has the word function in it to identifies the file as a function, rather than a generic m-file. For example, for a function named *abs_error.m*, the the first line would be:

```
function [X,Y] = abs_error(A,B)
```

  - A block of comments should be placed at the top of the regular m-files, and just ***after*** the function definition in function m-files. This is the header comment block. The formats are different for m-files and functions.

Style Guide (2)

- Variables should have meaningful names. This will make your code easier to read, and will reduce the number of comments you will need. However here are some pitfalls about choosing variable names:
  - Meaningful variable names are good, but when the variable name gets to 15 characters or more, it tends to obscure rather than improve code.
  - The maximum length of a variable name is 19 characters and all variables *must start with a character (not number)*.
  - Be careful of naming a variable that will conflict with matlab's built-in functions, or reserved names: if, while, end, pi, sin, cos, etc.
  - Avoid names that differ only in case, look similar, or differ only slightly from each other.
- Make good use of white space, both horizontally and vertically, it will improve the readability of your program greatly.

Style Guide (3)

- Comments describing tricky parts of the code, assumptions, or design decisions should be placed above the part of the code you are attempting to document.
- Do not add comment statements to explain things that are obvious.
- Try to avoid big blocks of comments except in the detailed description of the m-file in the header block.
- **Indenting**. Lines of code and comments inside branching (if block) or repeating (for and while loop) logic structures will be indented 3 spaces. NOTE: don't use tabs, use spaces. For example:

```
for i=1:n
   disp('in loop')
   if data(i) < x
       disp('less than x')
   else
       disp('greater than or equal to x')
   end
   count = count + 1;
end
```

Style Guide (4)

- Be careful what numbers you "hardwire" into your program. You may want to assign a constant number to a variable. If you need to change the value of the constant before you re-run the program, you can change the number in one place, rather than searching throughout your program.

Bad!

```
%  This program "hardwires" the constant 100
%  in three places in the code.

for i = 1:100
   data = r(i);
end
temp = data/100;
meanTemp = sum(temp)/100;
```

Good

```
% This program assigns the constant value to
% the variable, n.

n = 100;                   % number of data points.

for i = 1:n
   data = r(i);
end
temp = data/n;
meanTemp = sum(temp)/n;
```

Style Guide (5)

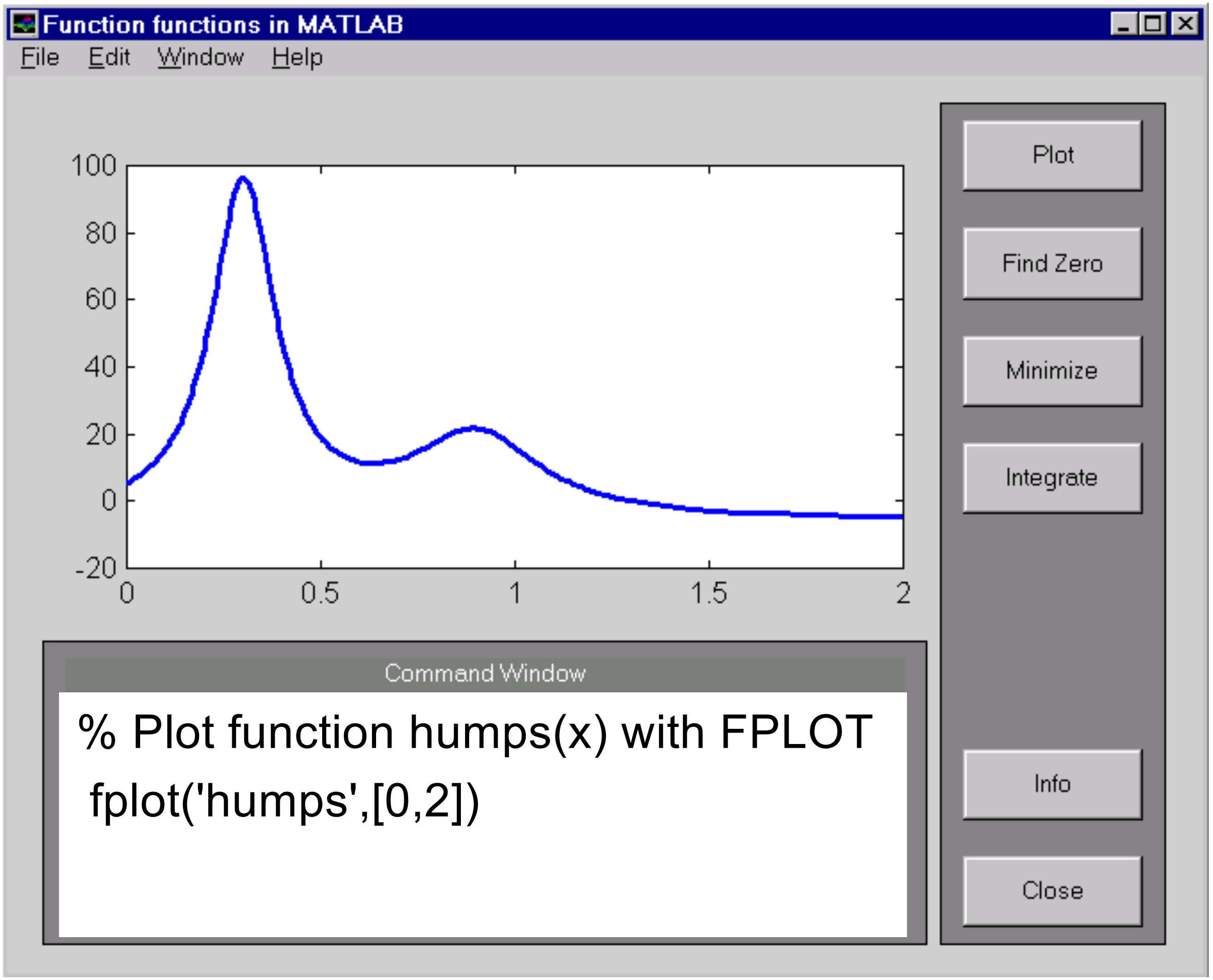
- No more than **one** executable statement per line in your regular or function m-files.
- No line of code should exceed 80 characters. (There may be a few times when this is not possible, but they are rare).
- The comment lines of the function m-file are the printed to the screen when *help* is requested on that function.

```
function bias = bias_error(X,Y)
% Purpose:  Calculate the bias between input arrays X and Y
% Input:    X, Y, must be the same length
% Output:   bias = bias of X and Y
%
% filename: bias_error.m
% Mary Jordan,  3/10/96
%
bias = sum(X-Y)/length(X);
```

Style Guide (6) - Another good example

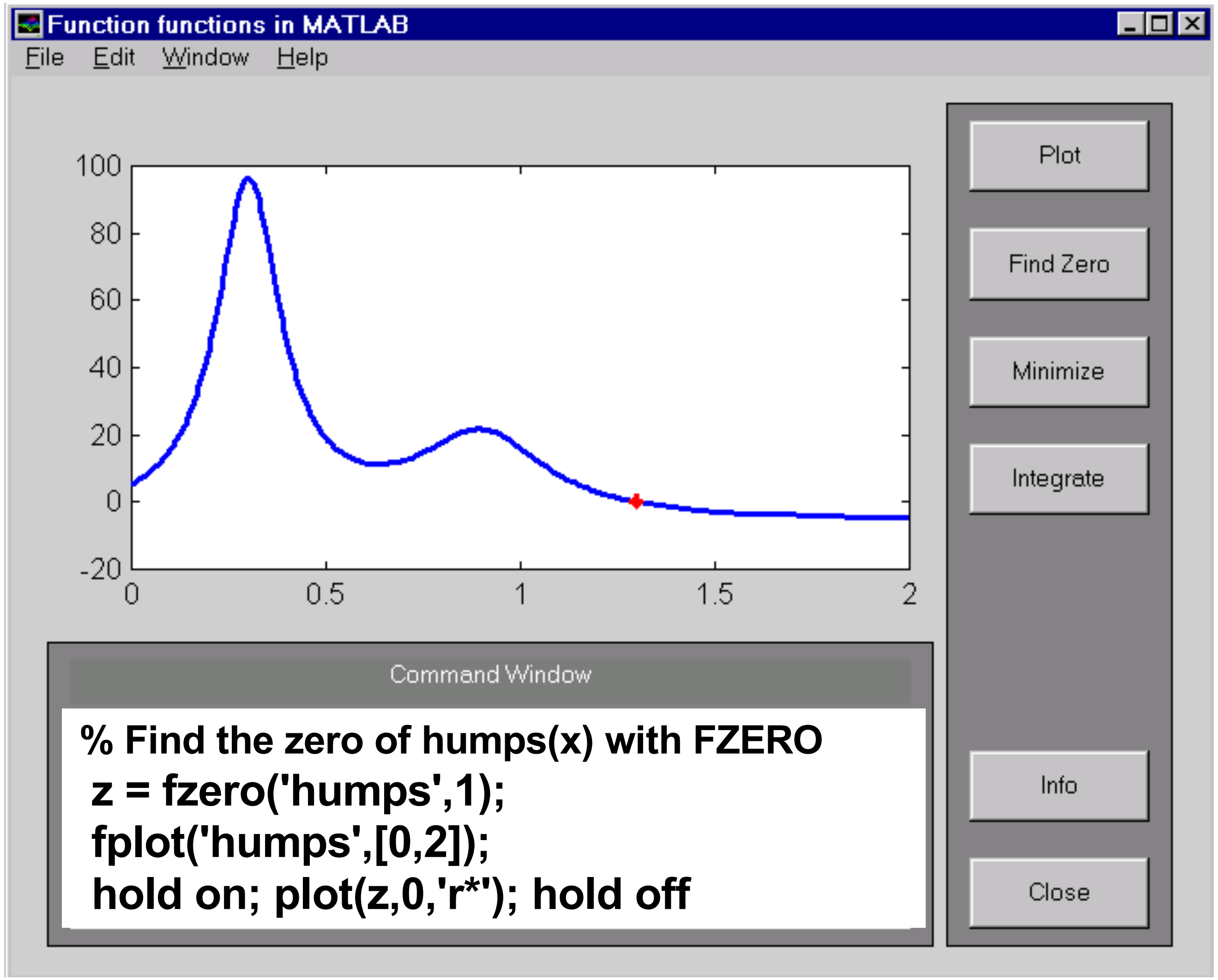
```
function [out1,out2] = humps(x)
%
%   Y = HUMPS(X) is a function with strong maxima near x = .3
%   and x = .9.
%
%   [X,Y] = HUMPS(X) also returns X.  With no input arguments,
%   HUMPS uses X = 0:.05:1.
%
%   Copyright (c) 1984-97 by The MathWorks, Inc.
%   $Revision: 5.3 $  $Date: 1997/04/08 05:34:37 $

if nargin==0, x = 0:.05:1; end

y = 1 ./ ((x-.3).^2 + .01) + 1 ./ ((x-.9).^2 + .04) - 6;

if nargout==2,
  out1 = x; out2 = y;
else
  out1 = y;
end
```

Function of functions - fplot



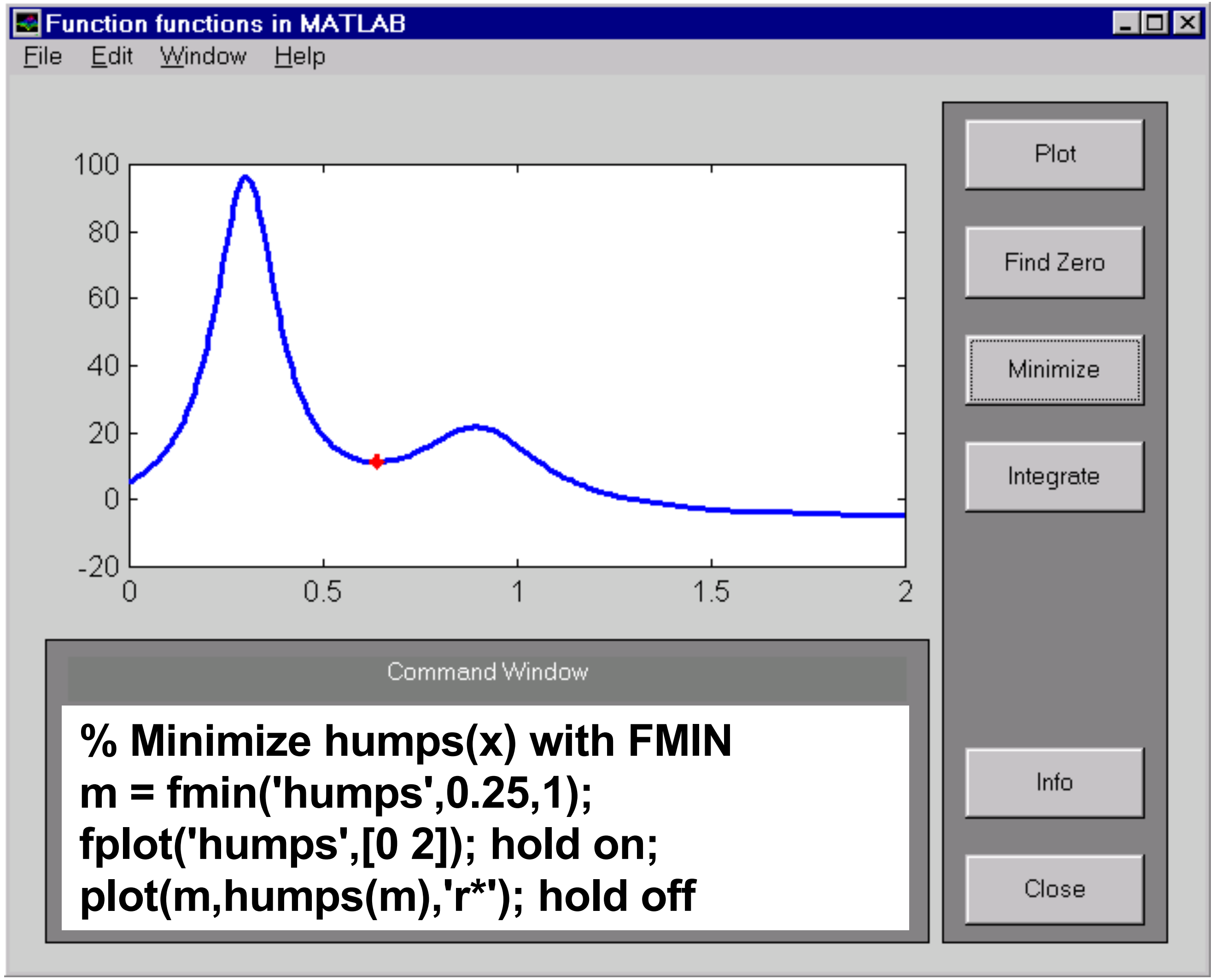
FPLOT(FUN,LIMS) plots the function specified by the string FUN between the x-axis limits specified by LIMS = [XMIN XMAX]

Find Zero



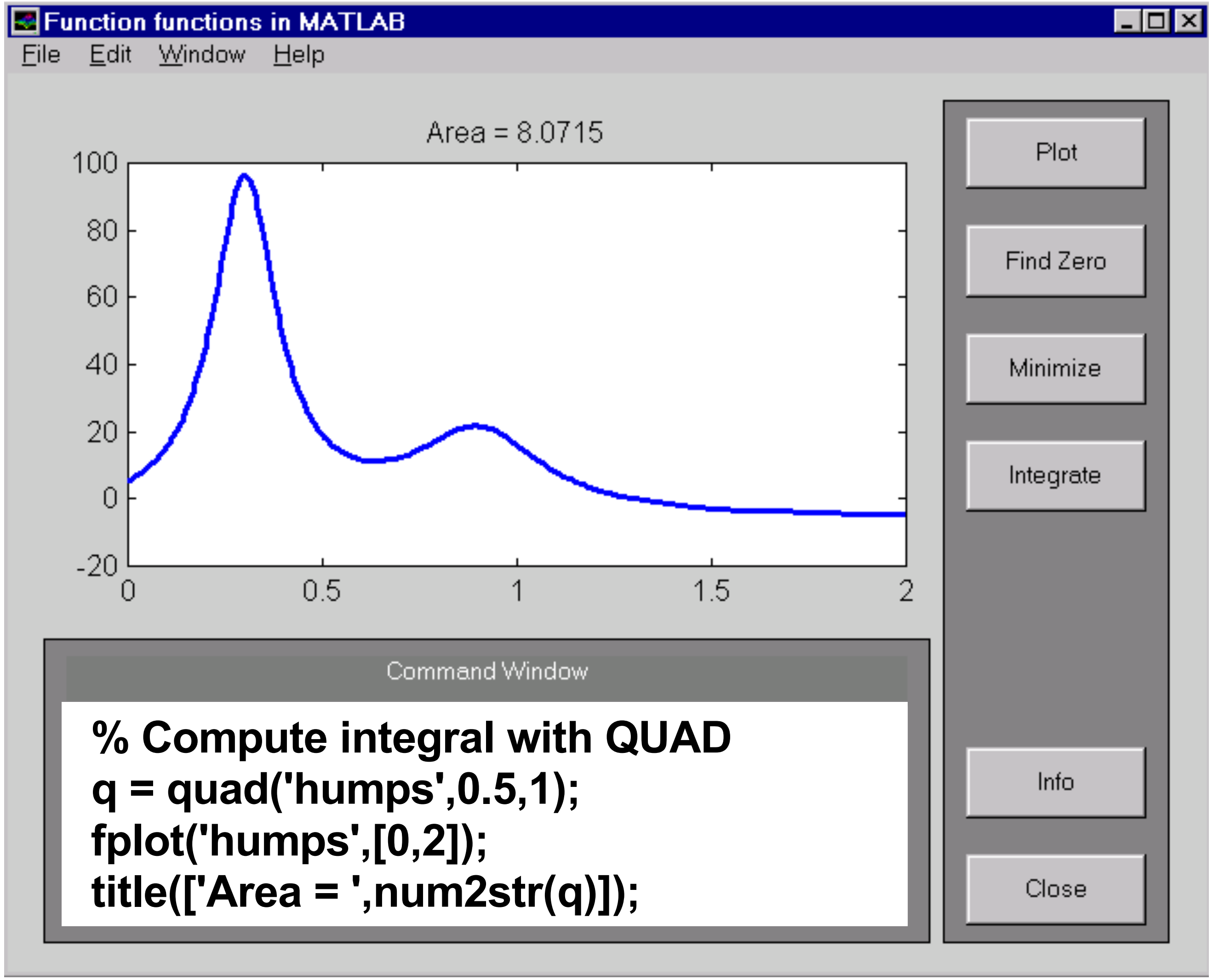
FZERO(F,X) tries to find a zero of F. FZERO looks for an interval containing a sign change for F and containing X.

Find minimum



X = FMIN('F',x1,x2) attempts to return a value of x which is a local minimizer of F(x) in the interval x1 < x < x2.

Integration of Curve

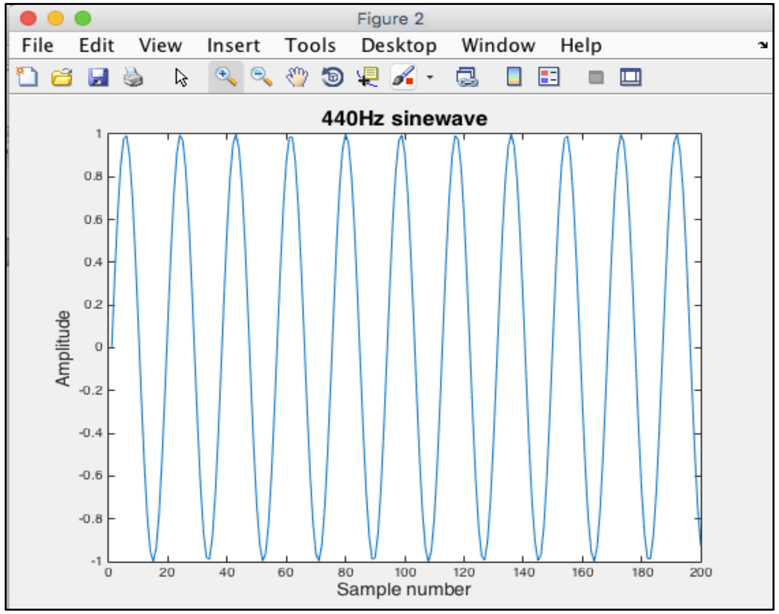


Q = QUAD('F',A,B) approximates the integral of F(X) from A to B to within a relative error of 1e-3 using an adaptive recursive Simpson's rule.

Lab 1 - Ex 1: The sine_gen function

```
function [sig] = sine_gen(amp, f, fs, T)% Function to generate a sinewave
%
%   ....   with a sampling frequency fs for a duration T
%
%   usage:     signal = sine_gen(1.0, 440, 8192, 1)
%
% author: Peter YK Cheung, 17 Jan 2017

    dt = 1/fs;
    t = 0:dt:T;
    sig = amp*sin(2*pi*f*t);
```

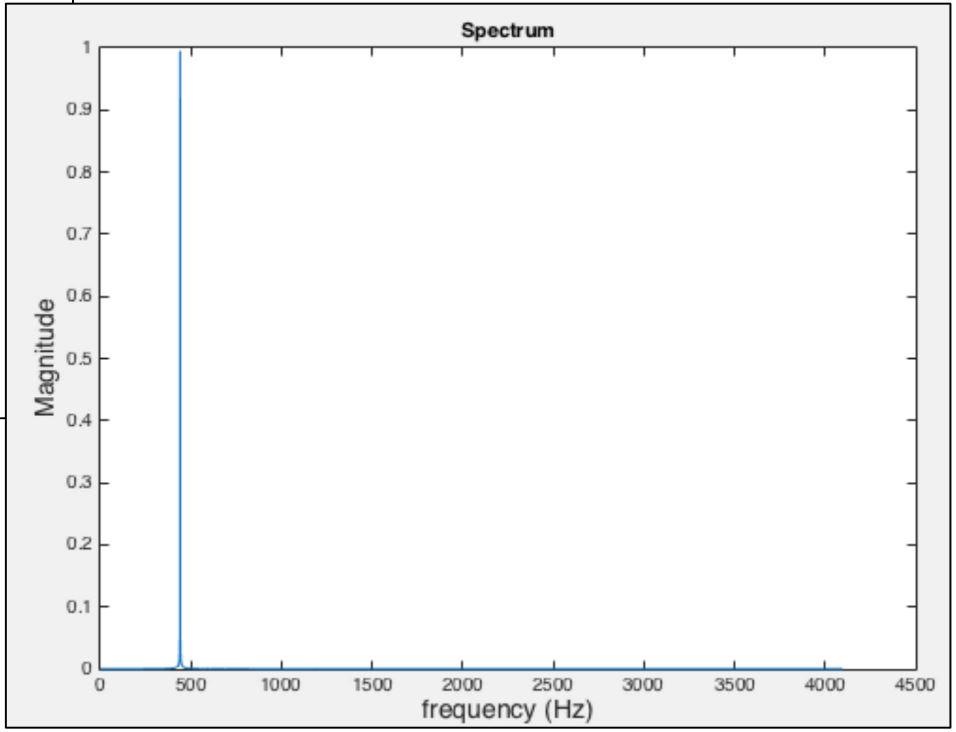
```
>> s1 = sine_gen(1.0, 440, 8192, 1);
>> plot(s1(1:200));
>> xlabel('\fontsize{14}Sample number');
>> ylabel('\fontsize{14}Amplitude');
>> title('\fontsize{16}440Hz sinewave');
```



Lab 1 - Ex 2: The plot_spec function

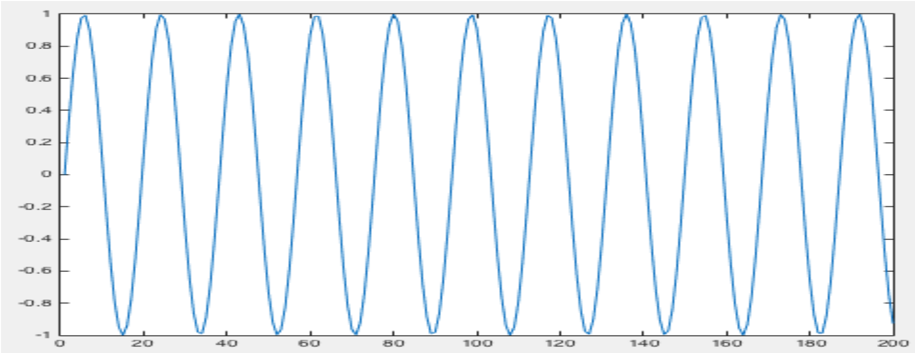
```
function plot_spec(sig, fs)

% Function to plot frequency spectrum of sig
%   usage:
%           plot_spectrum(sig, 8192)
%
% author: Peter YK Cheung, 17 Jan 2017
    magnitude = abs(fft(sig));
    N = length(sig);
    df = fs/N;
    f = 0:df:fs/2;
    Y = magnitude(1:length(f));
    plot(f, 2*Y/N)
    xlabel('\fontsize{14}frequency (Hz)')
    ylabel('\fontsize{14}Magnitude');
```

```
>> s1 = sine_gen(1.0, 440, 8192, 1);
>> plot_spec(s1,8192);
>> title('Spectrum')
```

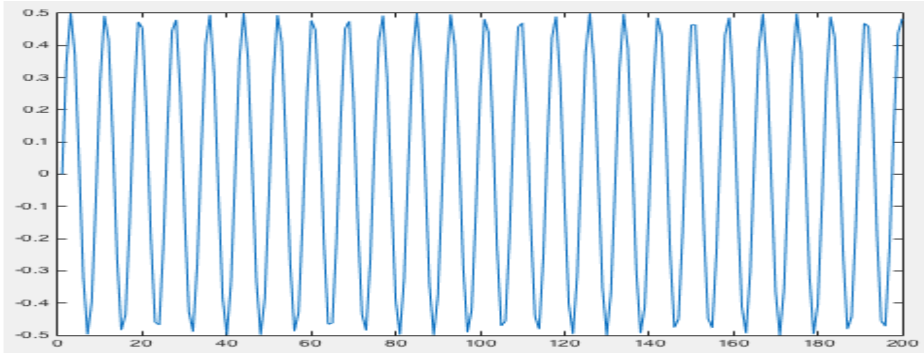


Lab 1 - Ex 3: Two tones (fs = 8192, T = 1)

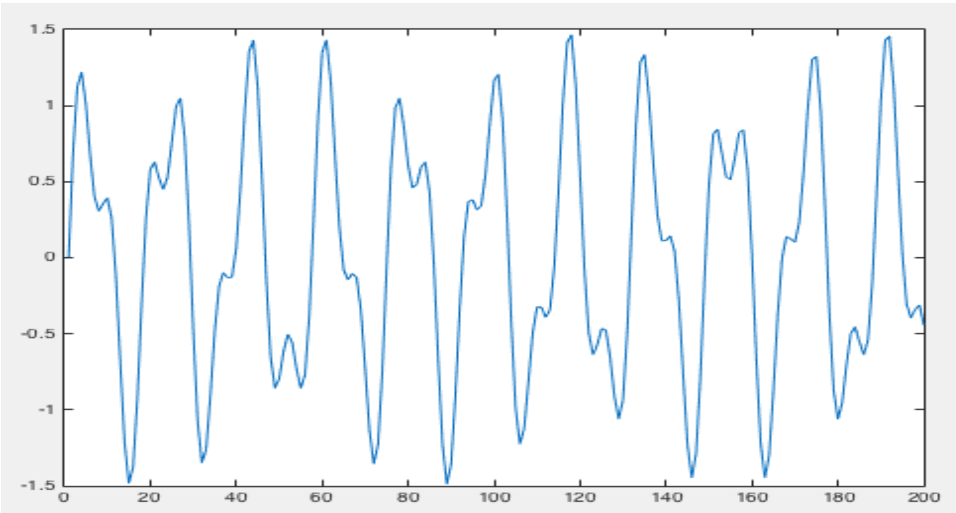
- s1 = 440Hz sine at 1V
- s2 = 1kHz sine at

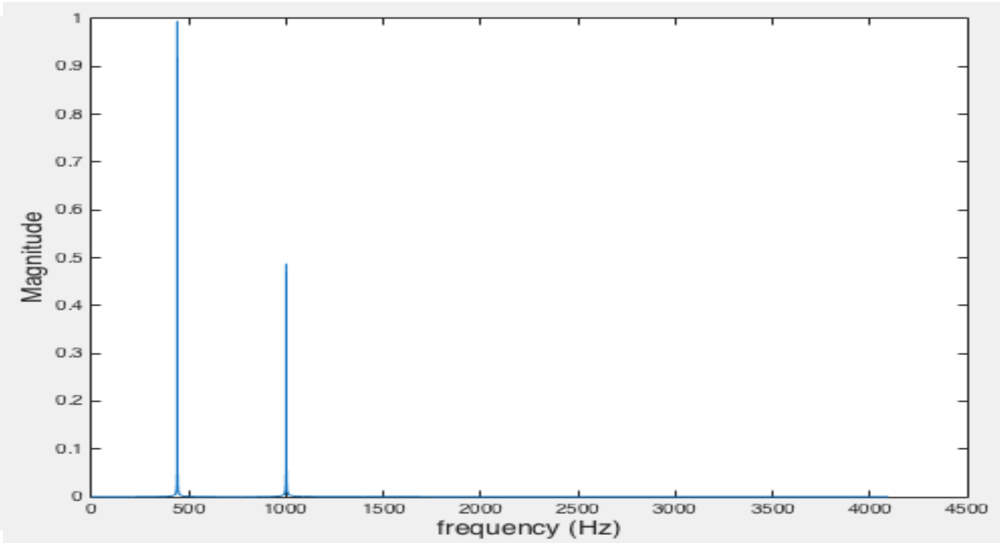


+



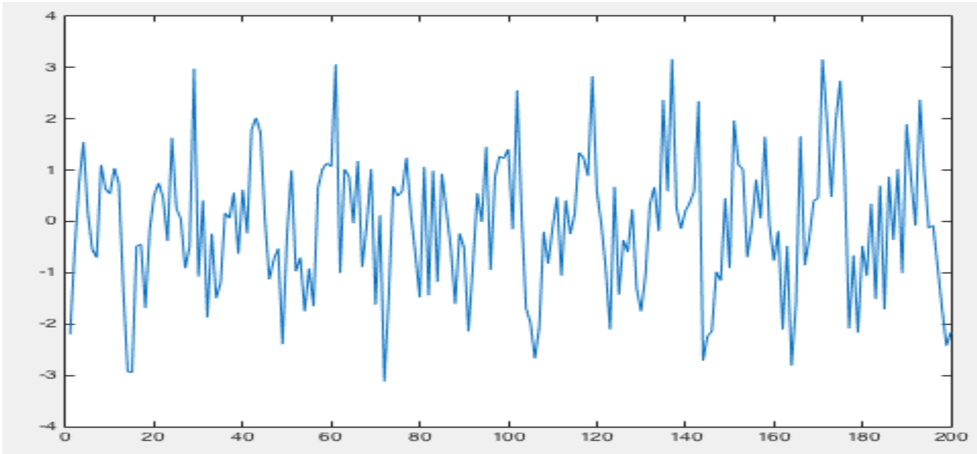
- sig = s1 + s2
- plot_spec (sig, 8192)





Lab 1 - Ex 4: Two tones + noisy

- Noisy = sig + randn(size(sig));



  - plot_spec (noisy, 8192)