INTRODUCTION TO MATLAB

Kadin Tseng
Research Computing Services, IS&T
Boston University

What is MATrix LABoratory ?

- It is developed by The Mathworks, Inc. (http://www.mathworks.com)

- It is an interactive, integrated, environment
  - for numerical/symbolic, scientific computations and other apps.
  - shorter program development and debugging time than traditional programming languages such as FORTRAN and C.
  - slower (compared with FORTRAN or C) because it is interpreted.
  - automatic memory management; no need to declare arrays.
  - intuitive, easy to use.
  - compact notations.

Getting Started With MATLAB

- Latest version is MATLAB 2014a
- For Windows: double click MATLAB icon
- For Linux clusters: scc1% matlab
- Either case spawns a MATLAB window with >> prompt.

```
>>   % symbol to end of line used for code documentation
>>
>> version          % running MATLAB version
ans =
8.1.0.604 (R2013a)
>> help             % lists available packages/toolboxes on system.
>> help elfun       % lists functions in elementary functions package
>> help sin         % instructions on the sine function
>> lookfor sine     % if you don't know the function name …
>> doc              % detail MATLAB and toolbox documentation
>> doc sin          % more detail than help (usually)
>> quit             % quit MATLAB;  exit  works too!
```

Rules on Variable and File Names

- Variables
  - case sensitive, *e.g.,* NAME and Name are 2 distinct names.
  - variable begins with a *letter*, *e.g.,* A2z or a2z
  - can be a mix of letters, digits, and underscores (*e.g.,* vector_A)
  - reserved characters: % = + – ~ ; : ! ' [ ] ( ) , @ # $ & ^
  - up to 63 characters (no reserved characters)
- Commands/Functions/scripts
  - performs specific tasks; same naming rules apply
- File names
  - *MATLAB* command files should be named with a suffix of ".m", *e.g., myfile.m*. An m-file typically contains a sequence of MATLAB commands that will be executed in order
  - An m-file may also contain other m-files
  - A file may also be just data (strings, numbers*)* – *ascii text or binary*

Reserved Characters % = ; ,

- Some characters are **reserved** by *MATLAB* for various purposes. Some as arithmetic or matrix operators: =, +, - , *, / , \ and others are used to perform a multitude of operations. Reserved characters cannot be used in variable or function names. They may have multiple uses.
- `>> % anything after % until the end of line is treated as comments`

  `>>`
- `>> a = 3      % define a to have the value 3`

  ```
  a =
        3
  ```
- `>> a = 3;    % ";" suppresses printing`

  `>>`
- `>> b = 4; c = 5;     % ";" enables multiple commands on same line`

  `>>`
- `>> d = 6, e = 7;      % "," delimits commands but enables printing`

  ```
  d =
        6
  ```

Reserved Characters : [ ] ( )

- >> *x = 1:2:9   % define vector x with : operator (begin:interval:end)*

```
x =
      1     3     5     7     9
```

- >> *y = 3:5   % interval default to 1; same as y = 3:1:5 = [3:5]*

```
y =
      3     4     5
```

- >> *X = [1, 2, 3; 4, 5, 6] % 2D array. The ; is vertical concatenation.*
  % [ ] for arrays. Prevents ambiguity
  % ; concatenates vertically (new row)
  % , concatenates horizontally (new columns)

```
X =
      1     2     3
      4     5     6
```

- >> *X(2,3)   % ( ) for subscripting; why ans ?*

```
ans =
      6
```

Reserved Characters … and '

```
>> x = [1 2 3 ...   % elipses ... means to be continued on the next line
     4 5 6]
x =
    1     2     3     4     5     6
>> s = 'this is a character string';     % blanks preserved within quotes
>> x = [1 2 3]'    %  '  performs transpose (e.g., turns row into column)
x =
       1
       2
       3
>> X = [1 2 3; 4 5 6]; size(X)    % figure out the size (dimensions) of X
ans =
        2     3
>> X = [1 2 3; 4 5 6]; numel(X)  % total number of entries in X
ans =
        6
```

Reserved Character ! (or system)

- >> *!dir % "!" lets you run local host command (MS Windows)*

```
Volume in drive C has no label.
  Volume Serial Number is 6860-EA46
      Directory of C:\Program Files\MATLAB704\work
      01/31/2007  10:56 AM    <DIR>          .
      01/31/2007  10:56 AM    <DIR>          ..
      06/13/2006  12:09 PM                12 foo.exe
      06/13/2006  08:57 AM                77 mkcopy.m
```

- >> *!ls –l % "!" lets you run local host command (Unix/Linux)*

```
total 0
-rw-r--r--  1 kadin scv 0 Jan 19 15:53 file1.m
-rw-r--r--  1 kadin scv 0 Jan 19 15:53 file2.m
-rw-r--r--  1 kadin scv 0 Jan 19 15:53 file3.m
```

>> *system('ls –l') % more general form; also unix('ls –l')*

Array operations

```
>> a = 1:3;        % a is a row vector
>> b = 4:6;        % b is a row vector
>> c = a + b       % a & b agree in shape, size; c inherit same shape & size
     5     7     9
>> A = [a;b]   % combines vectors into array;   Does A=a;b  work ?
A =
     1     2     3
     4     5     6
>> B = A'    % B is transpose of A
B =
     1     4
     2     5
     3     6
```

Other ways to create B ? (hint: with *a* & *b* directly)

Matrix Operations

```
>> C = A*B     % * is overloaded  as matrix multiply operator
C =
    14    32
    32    77
>> D = A.*A         %  a  .*  turns matrix multiply to elemental  multiply
D =
     1     4     9
    16    25    36
>> E = A./A         % elemental divide
E =
     1     1     1
     1     1     1
>> who        % list existing variables in workspace
Your variables are:
A    B    C    D    E    a    b    d
```

Data Precisions

```
>> whos      % detail listing of workspace variables
 Name       Size                  Bytes     Class     Attributes
      A        2x3                    48  double
      B        3x2                    48  double
      C        2x2                    32  double
      D        2x3                    48  double
      E        2x3                    48  double
      a        1x3                    24  double
      b        1x3                    24  double
      c        1x3                    24  double

>> A = single(A);    % recast A to single data type to save memory
>> whos
Name       Size                  Bytes  Class
     A        2x3                    24  single
>> clear  % delete all workspace variables
```

For Loops

```
for j=1:5          % use for-loops to execute iterations / repetitions
    for i=1:3
        a(i, j) = i + j ;
    end
end
```

Utilities to initialize or define arrays: *ones, rand, eye, . . .*

Trigonometric and hyperbolic functions : *sin, cos, sqrt, exp, . . .*

These utilities can be used on scalar or vector inputs

```
>> a = sqrt(5); v = [1 2 3]; A = sqrt(v);
```

if Conditional

Scalar operation . . .

```
for j=1:3                  % column index
   for i=1:3               % row index
      a(i,j) = rand;       % a(i,j) = random number
      b(i,j) = 0;          % b(i,j) = 0
      if a(i,j) > 0.5      % unless . . .
         b(i,j) = 2;
      end
  end
end
```

Equivalent vector operations . . .

```
A = rand(3);        % A is a 3x3 random number double array
B = zeros(3);       % Initialize B as a 3x3 array of zeroes
B(A > 0.5) = 2;     % for all A(i,j) > 0.5, set B(i,j) to 2
```

Cell Arrays

A cell array is a special array of arrays. Each element of the cell array may point to a scalar, an array, or another cell array.

```
>> C = cell(2, 3);   % create 2x3 empty cell array
>> M = magic(2);
>> a = 1:3; b = [4;5;6]; s = 'This is a string.';
>> C{1,1} = M; C{1,2} = a; C{2,1} = b; C{2,2} = s; C{1,3} = {1};
C =
    [2x2 double]     [1x3 double]          {1x1 cell}
    [2x1 double]     'This is a string.'    []
>> C{1,1}     % prints contents of a specific cell element
ans =
     1     3
     4     2
>> C(1,:)     % prints first row of cell array C; not its content
```

Related utilities: *iscell, cell2mat*

Structures

Ideal layout for grouping arrays that are related.

```
>> name(1).last = ‘Smith’;  name(2).last  = ‘Hess’;
>> name(1).first = ‘Mary’;   name(2).first = ‘Robert’;
>> name(1).sex = ‘female’; name(2).sex = ‘male’;
>> name(1).age = 45;         name(2).age = 50;
>> name(2)
ans =
     last: 'Hess'
    first: 'Robert'
      sex: 'male'
      age: 50
```

Alternative style:

```
>> name = struct(‘last’,{Smith’,’Hess’}, ‘first’,{Mary’,’Robert’},…
                         (‘sex’,{female’,’male’}, ‘age’,{45,50});
```

Related utilities: *isstruct, fieldnames, getfield, isfield*

File Types

There are many types of files in MATLAB.
Only script-, function-, and mat-files are covered here:

1. script m-files (.m) -- group of commands; reside in base workspace
2. function m-files (.m) -- memory access controlled; parameters passed as input, output arguments; reside in own workspace
3. mat files (.mat) -- binary (or text) files handled with *save* and *load*
4. mex files (.mex) -- runs C/FORTRAN codes from m-file
5. eng files (.eng) -- runs m-file from C/FORTRAN code
6. C codes (.c) – C codes generated by MATLAB compiler
7. P codes (.p) – converted m-files to hide source for security

Script m–file

If you have a group of commands that are expected to be executed repeatedly, it is convenient to save them in a file . . .

```
>> edit mytrig.m   % enter commands in editor window
a=sin(x);   % compute sine x (radians)
b=cos(x);   % compute cosine x (radians)
disp( ['a = '    num2str(a) ] )   % prints a; here, [ . . . ] constitutes a string array
disp( ['b = '    num2str(b) ] )   % prints b
```

Select File/Save to save it as mytrig.m

A script m-file shares same memory space from which it was invoked. Define x, then use it in mytrig.m (mytrig can "see" x):

```
>> x=30*pi/180; %  converts 30 degrees to radians
>> mytrig    % x is accessible to mytrig.m; share same workspace
   a = 0.5000
   b = 0.8660
```

Script works as if sequentially inserting the commands in mytrig.m at the >>

Function m–files

- Declared with the key word *function*, with optional output parameters on the left and optional input on the right of =. All other parameters within function reside in function's own workspace; deleted upon exiting the function.

    Use MATLAB editor to create file: >> edit average.m

```
function avg=average(x)
% function avg=average(x)
% Computes the average of x
% x      (input)   matrix for which an average is sought
% avg  (output) the average of x
nx = numel(x);         % number of elements in x; in own workspace
avg = sum(x)/nx;     % avg is the average value on exit
end
```

- Keep file name the same as function name to avoid confusions
- May be called from a script or another function
- >> a = average(1:3)   % a = (1 + 2 + 3) / 3

```
a =
     2
```

```
>> help average     % prints contiguous lines with % at top of average
```

Script or Function m–file ?

Scripts

- Pros:
    - convenient; script's variables are in same workspace as caller's
- Cons:
    - slow; script commands loaded and interpreted each time used
    - risks of variable name conflict inside & outside of script

Functions

- Pros:
    - Scope of function's variables is confined to within function. No worry for name conflict with those outside of function.
    - What comes in and goes out are tightly controlled which helps when debugging becomes necessary.
    - Compiled the first time it is used; runs faster subsequent times.
    - Easily be deployed in another project.
    - Auto cleaning of temporary variables.
- Cons:
    - I/O are highly regulated, if the function requires many pre-defined variables, it is cumbersome to pass in and out of the function – a script m-file is more convenient.

Some Frequently Used Functions

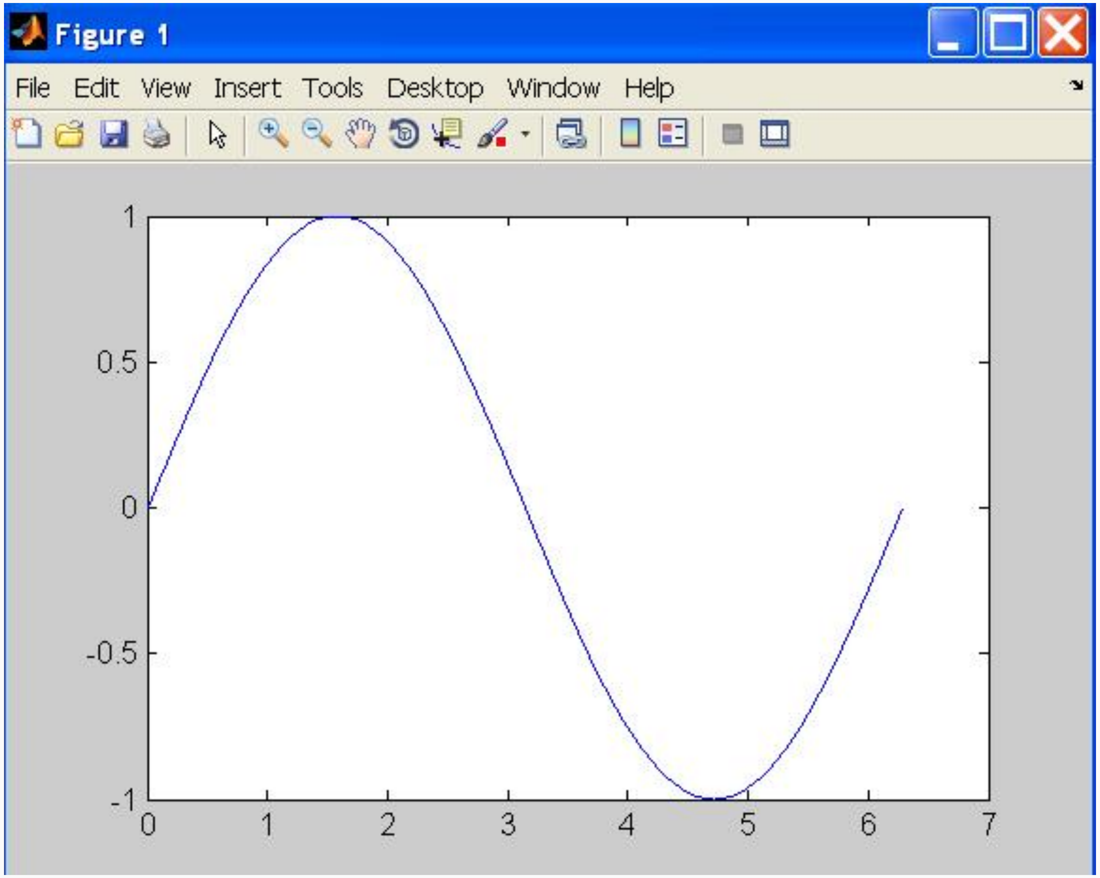
>> *magic(n)* % creates a special *n x n* matrix; handy for testing
>> zeros(n,m) % creates *n x m* matrix of *zeroes* (0)
>> ones(n,m) % creates *n x m* matrix of *ones* (1)
>> rand(n,m) % creates *n x m* matrix of random numbers
>> repmat(a,n,m) % replicates *a* by *n* rows and *m* columns
>> diag(M) % extracts the diagonals of a matrix M
>> help elmat % list all elementary matrix operations (or *elfun*)
>> abs(x); % absolute value of *x*
>> exp(x); % *e* to the x-th power
>> fix(x); % rounds x to integer towards 0
>> log10(x); % common logarithm of x to the base 10
>> rem(x,y); % remainder of x/y
>> mod(x, y); % modulus after division – unsigned rem
>> sqrt(x); % square root of x
>> sin(x); % sine of x; x in radians
>> acoth(x) % inversion hyperbolic cotangent of x

MATLAB Graphics

- Line plot
- Bar graph
- Surface plot
- Contour plot
- MATLAB tutorial on 2D, 3D visualization tools as well as other graphics packages available in our tutorial series.

Line Plot

```
>> t = 0:pi/100:2*pi;
>> y = sin(t);
>> plot(t,y)
```

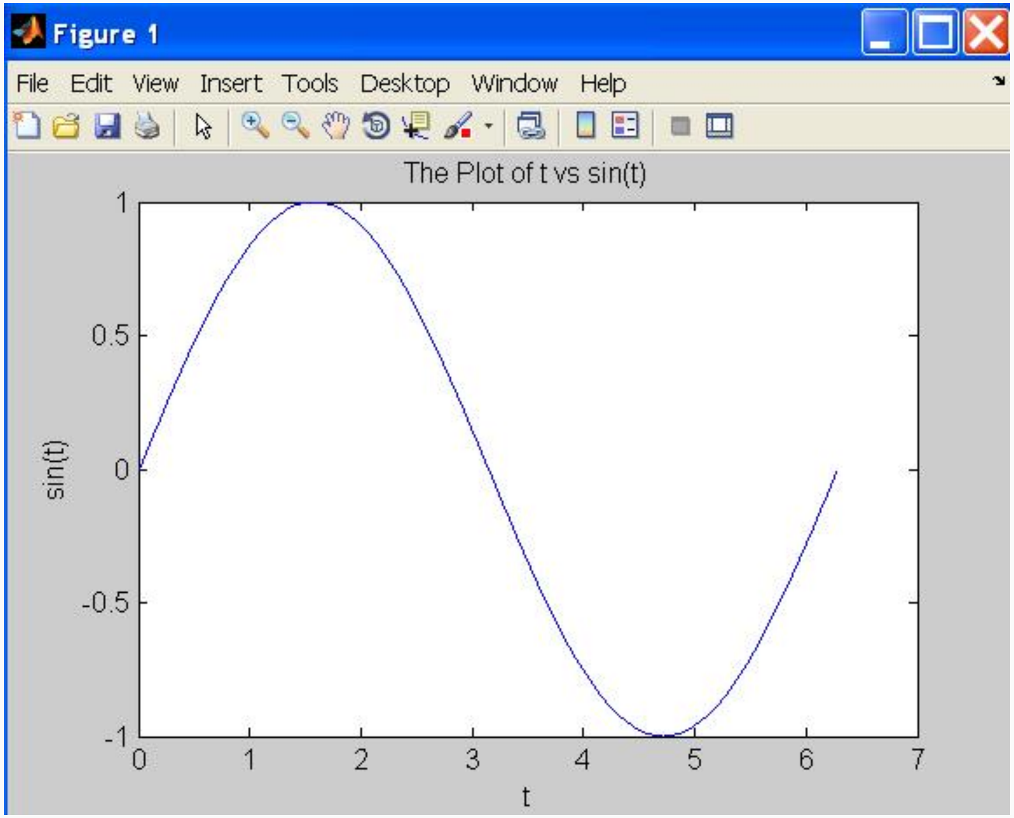


Line Plot

>> *xlabel('t');*

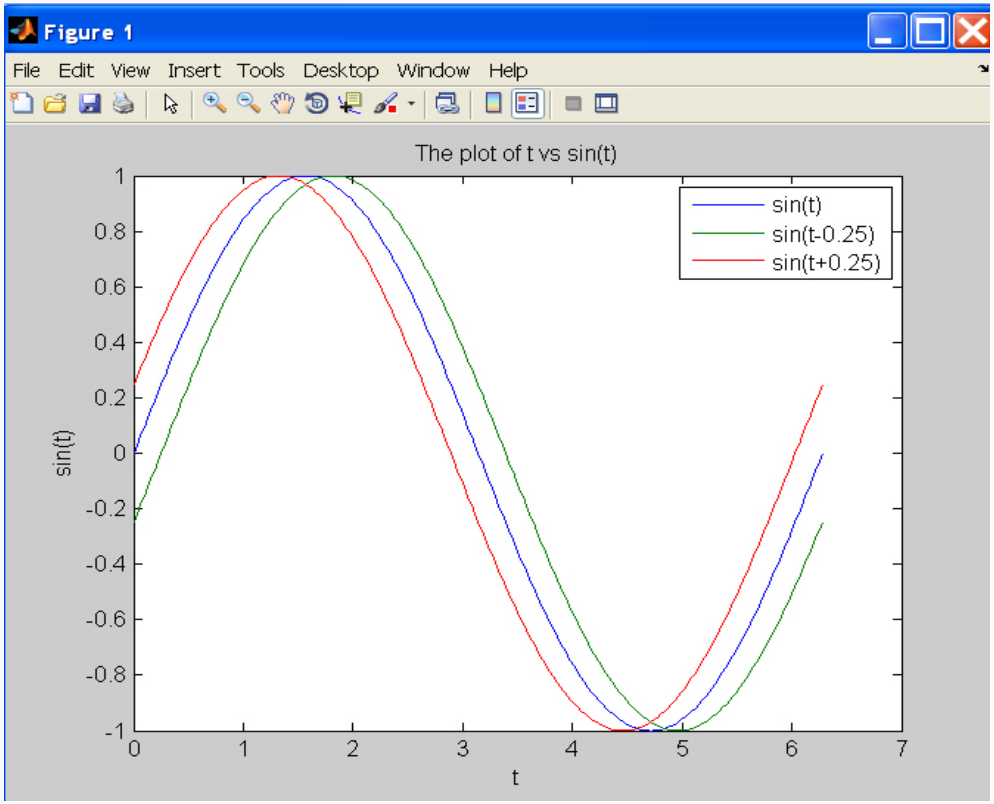
>> *ylabel('sin(t)');*

>> *title('The plot of t vs sin(t)');*



Line Plot

```
>> y2 = sin(t-0.25);
>> y3 = sin(t+0.25);
>> plot(t,y,t,y2,t,y3)   % make 2D line plot of 3 curves
>> legend('sin(t)','sin(t-0.25)','sin(t+0.25',1)
```



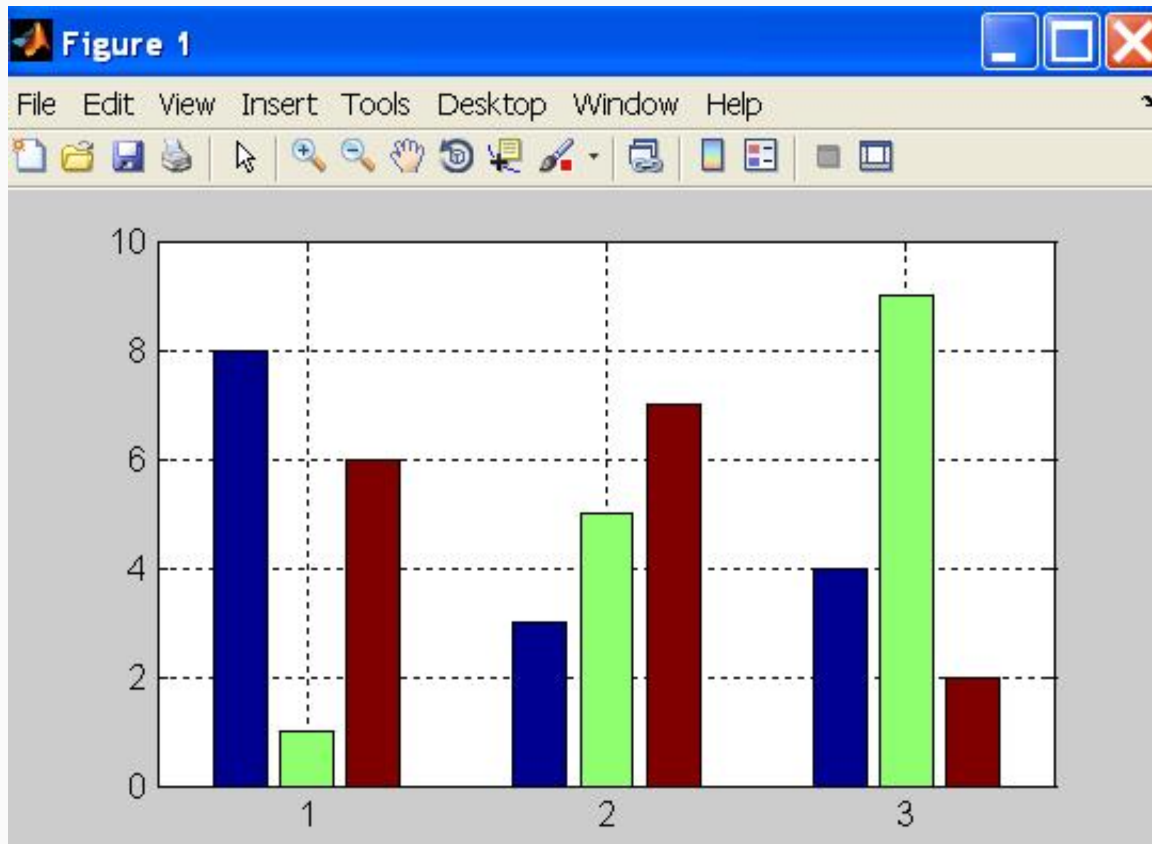
Customizing Graphical Effects

Generally, MATLAB's default graphical settings are adequate which make plotting fairly effortless. For more customized effects, use the *get* and *set* commands to change the behavior of specific rendering properties.

```
>> hp1 = plot(1:5)              % returns the handle of this line plot
>> get(hp1)                     % to view line plot's properties and their values
>> set(hp1, 'lineWidth')        % show possible values for lineWidth
>> set(hp1, 'lineWidth', 2)     % change line width of plot to 2
>> gcf             % returns current figure handle
>> gca             % returns current axes handle
>> get(gcf)        % gets current figure's property settings
>> set(gcf, 'Name', 'My First Plot')      % Figure 1 => Figure 1: My First Plot
>> get(gca)        % gets the current axes' property settings
>> figure(1)       % create/switch to Figure 1 or pop Figure 1 to the front
>> clf             % clears current figure
>> close           % close current figure; "close 3" closes Figure 3
>> close all       % close all figures
```

2D Bar Graph

```
>> x = magic(3);     % generate data for bar graph
>> bar(x)            % create bar chart
>> grid              % add grid for clarity
```



Use MATLAB Command or Function ?

- Many MATLAB utilities are available in both command and function forms.

- For this example, both forms produce the same effect:
- *>> print –djpeg mybar % print as a command*
- *>> print('-djpeg', 'mybar') % print as a function*

- For this example, the command form yields an unintentional outcome:
- *>> myfile = 'mybar'; % myfile is defined as a string*
- *>> print –djpeg myfile % as a command, myfile is treated as text*
- *>> print('-djpeg', myfile) % as a function, myfile is treated as a variable*
  % i.e., ‘mybar’ is passed into print

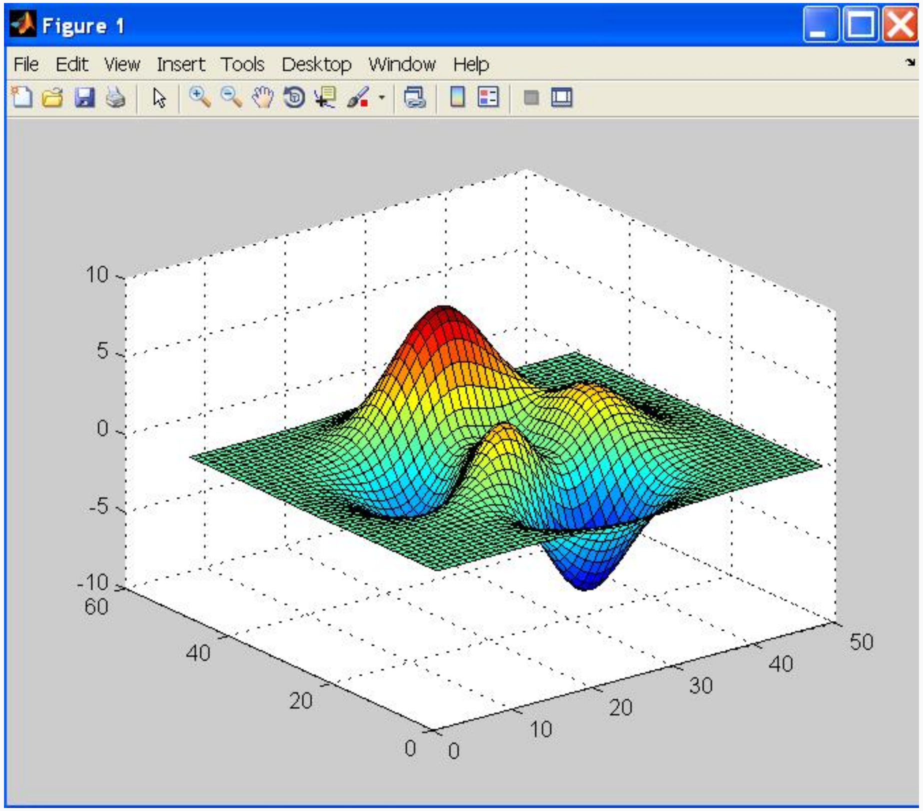
- Other frequently used utilities that are available in both forms are:
- *save, load*

Surface Plot

>> *Z = peaks;   % generate data for plot;   peaks returns function values*
>> *surf(Z)   % surface plot of Z*

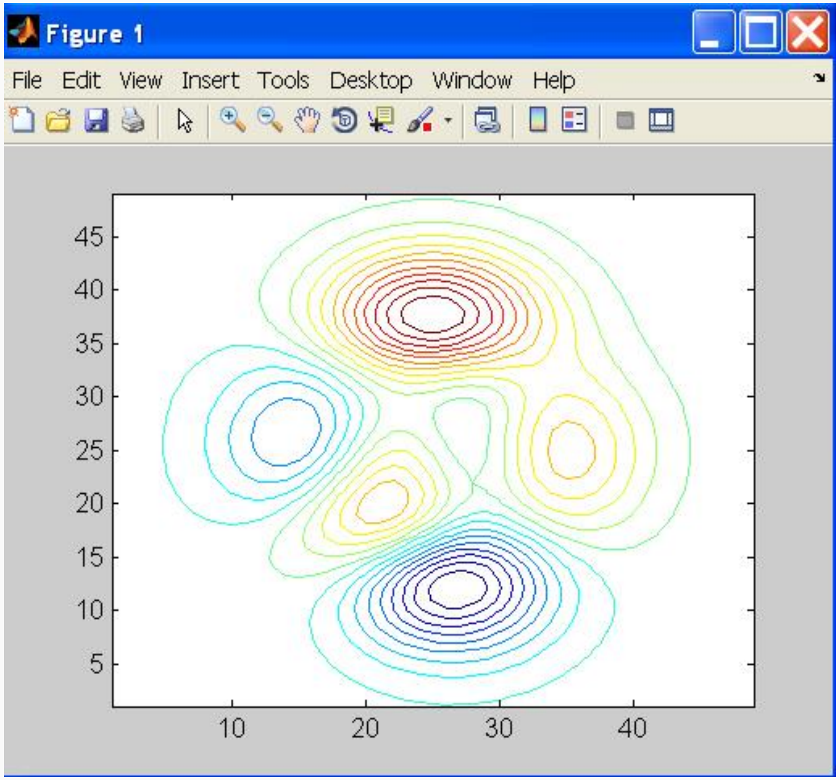
Try these commands also:

>> *shading flat*
>> *shading interp*
>> *shading faceted*
>> *grid off*
>> *axis off*
>> *colorbar*
>> *colormap('winter')*
>> *colormap('jet')*

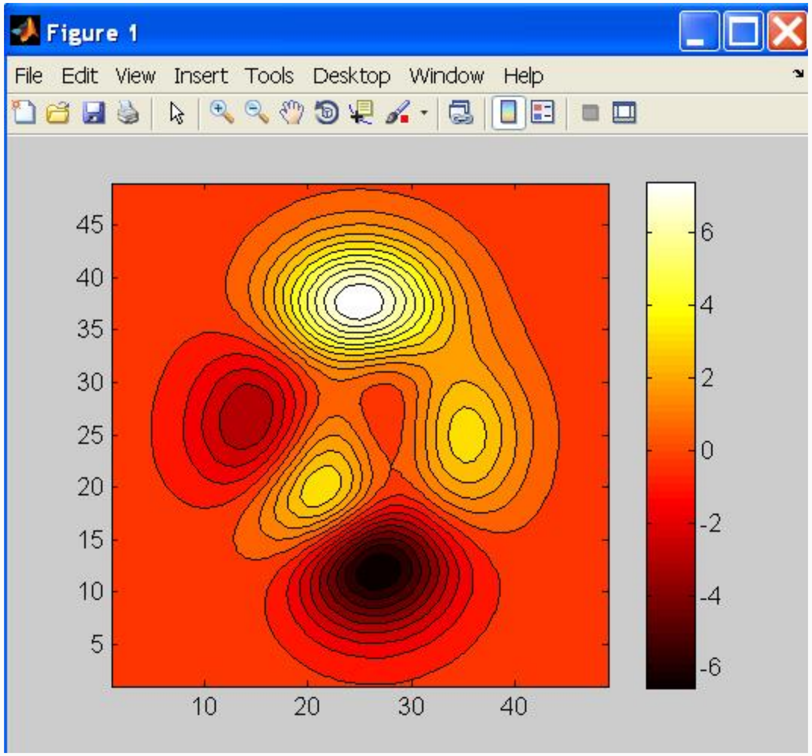


Contour Plots

```
>> Z = peaks;
>> contour(Z, 20)    % contour plot of Z with 20 contours
```

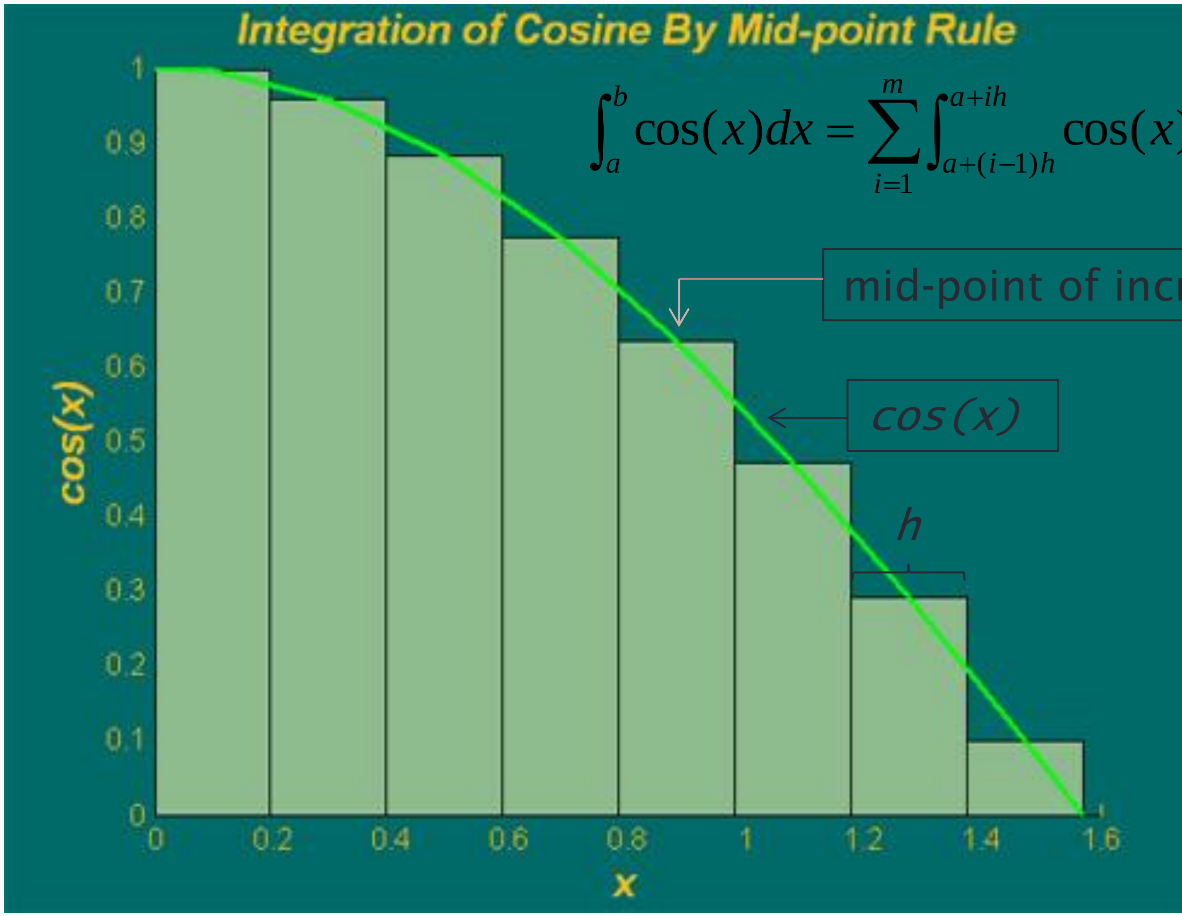



```
>> contourf(Z, 20);  % with color fill
>> colormap('hot')   % map option
>> colorbar      % make color bar
```



Integration Example

- Integration of cosine from 0 to $\pi/2$.
- Use mid-point rule for simplicity.

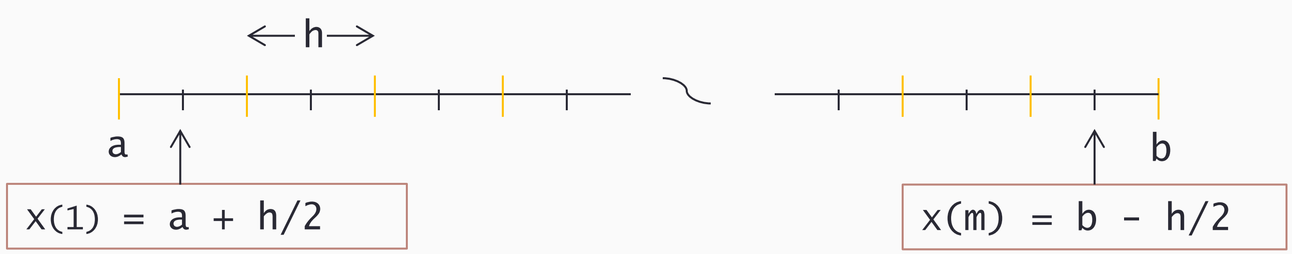


$$\int_a^b \cos(x)dx = \sum_{i=1}^{m} \int_{a+(i-1)h}^{a+ih} \cos(x)dx \approx \sum_{i=1}^{m} \cos(a + (i - \tfrac{1}{2})h)h$$

```
a = 0; b = pi/2;  % range
m = 8;  % # of increments
h = (b-a)/m;  % increment
```

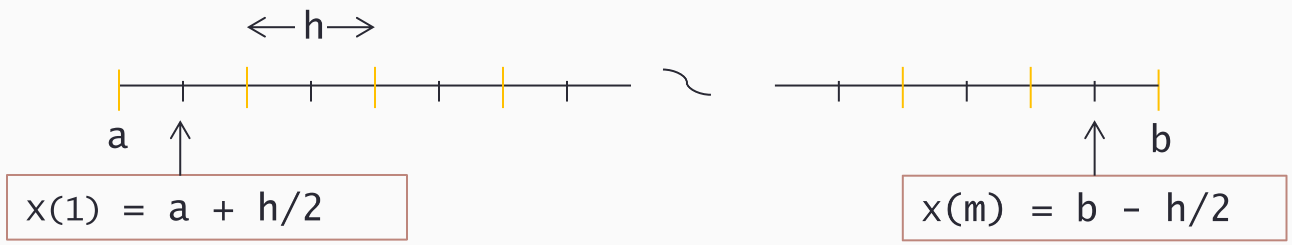
Integration Example — with for–loop

```
% integration with for-loop
tic
   m = 100;
   a = 0;                  % lower limit of integration
   b = pi/2;               % upper limit of integration
   h = (b - a)/m;          % increment length
   integral = 0;           % initialize integral
   for i=1:m
     x = a+(i-0.5)*h;      % mid-point of increment i
     integral = integral + cos(x)*h;
   end
toc
```



Integration Example — in vector form

```
% integration with vector form
tic
   m = 100;
   a = 0;                  % lower limit of integration
   b = pi/2;               % upper limit of integration
   h = (b - a)/m;          % increment length
   x = a+h/2:h:b-h/2;      % mid-point of m increments
   integral = sum(cos(x))*h;
toc
```



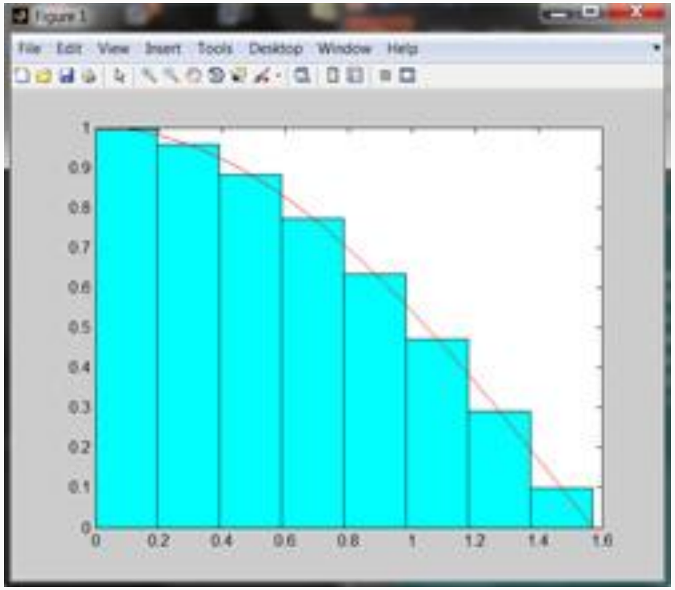
Integration Example — with function

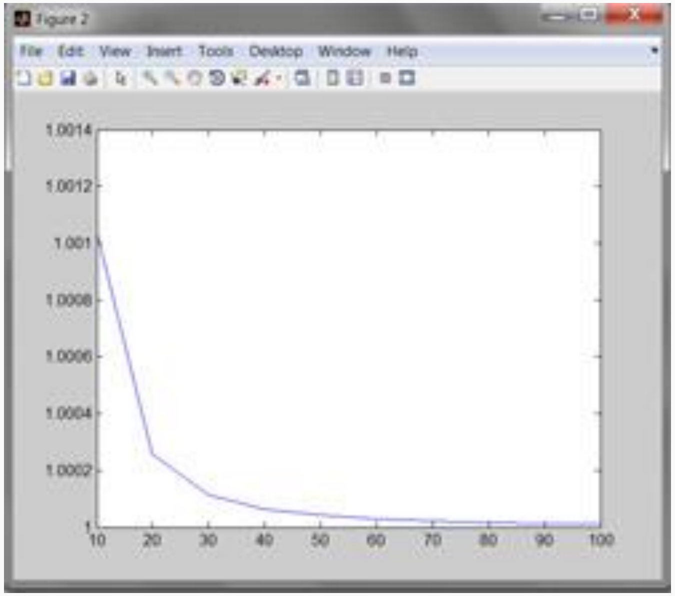
```
% integration with for-loop
tic
m = 100;      % number of intervals
a = 0;        % lower limit of integration
b = pi/2;     % upper limit of integration
integral = midpoint(a, b, m, @cos);  % integrand determined at runtime
toc
```

```
function int = midpoint(a, b, m, fct)
h = (b - a)/m;             % increment length
int = 0;                   % initialize int
  for i=1:m
    x = a+(i-0.5)*h;       % mid-point of increment i
    int = int + fct(x)*h;
  end
end
```

Hands On Exercise

1. Write a program (with editor) to generate the figure that describe the integration scheme we discussed. (Hint: use *plot* to plot the cosine curve. Use *bar* to draw the rectangles that depict the integrated value for each interval. Save as plotIntegral.m
2. Compute the cosine integrals, from 0 to pi/2, using 10 different increment sizes (*10, 20, 30, . . . , 100)*. Plot these *10* values to see how the solution converges to the analytical value of *1*.





Hands On Exercise Solution

```
a = 0;  b=pi/2;             % lower  and upper limits of integration
m = 8;                      % number of increments
h = (b-a)/m;                % increment size
x= a+h/2:h:b-h/2;           % m mid-points
bh = bar(x,cos(x),1,'c');   % make bar chart with bars full width (1) and cyan (‘c’)
hold                        % all plots will be superposed on same figure
x = a:h/10:b;               % use more points at which to evaluate cosine
f = cos(x);                 % compute cosine at x
ph = plot(x,f,'r');         % plots x vs f, in red
% Compute integral with different values of m to study convergence
for i=1:10
    n(i) = 10+(i-1)*10;
    h = (b-a)/n(i);
    x = a+h/2:h:b-h/2;
    integral(i) = sum(cos(x)*h);
end
figure   % create a new figure
plot(n, integral)
```

Useful RCS Info

- **RCS home page** (www.bu.edu/tech/research)
- **Resource Applications**
  www.bu.edu/tech/accounts/special/research/accounts
- **Help**
  - **System**
    - **help@scc.bu.edu, bu.service-now.com**
  - Web-based tutorials
    (www.bu.edu/tech/about/research/training/live-tutorials)
    (MPI, OpenMP, MATLAB, IDL, Graphics tools)
  - HPC consultations by appointment
    - Katia Oleinik (koleinik@bu.edu)
    - Yann Tambouret (yannpaul@bu.edu)
    - Kadin Tseng (kadin@bu.edu)