Basics of Matlab

Statements and variables

Matrices

Graphics

Control flow

Scripts and Functions

Why Matlab?

- combines numerics, graphics, and programming
  - powerful
  - easy to use (?)
- toolboxes provide access to hundreds of useful routines
- widespread use in engineering education
- latest editions of many textbooks use Matlab
- many subjects at MIT use Matlab
- Matlab 5.x provides powerful programming features such as data structures and cell arrays

Statements and variables

Entering and displaying a matrix $\mathbf{A}$

```
>> A=[1 2; 4 6]

A =

     1     2
     4     6
```

Semicolon suppresses output:

```
>> A=[1 2; 4 6];
>> A=[1 2; 4 6]

A =

     1     2
     4     6
>>
```

Statements and variables

Matlab operators:

```
+ addition
- subtraction
* multiplication
/ division
^ power
```

You can use Matlab as a calculator:

```
>> 12.4/6.9

ans =

   1.7971
```

If no assigment takes place, the result is placed in the variable `ans`

Variable names

Matlab variables must begin with a letter

The rest of the characters can be letters, digits, or underscores.

Only the first 19 characters are significant

```
>> jleonardjleonardjleonard = 1

jleonardjleonardjle =

     1
>>
```

Matlab is case sensitve.

```
>> M = [1 2];
>> m = [3 5 7];
```

`M` and `m` are not the same.

Predifined variables

```
pi    Inf    NaN    i    j
```

```
>> z = 3+4*i

z =

   3.0000 + 4.0000i

>> inf

ans =

   Inf

>> 0/0

Warning: Divide by zero

ans =

   NaN

>>
```

Managing your workspace

The function `who` lists the variables in the workspace.

```
>> who
Your variables are:
A          M          ans        m          z
```

The function `whos` lists the size and memory allocation of your variables

```
>> whos
Name     Size    Elements  Bytes Density Complex
   A     2 by 2       4      32    Full   No
   M     1 by 2       2      16    Full   No
 ans     1 by 1       1       8    Full   No
   m     1 by 3       3      24    Full   No
   z     1 by 1       1      16    Full   Yes

Grand total is 11 elements using 96 bytes

>>
```

Managing your workspace

The command `clear` can be used to remove variables from the workspace

```
>> clear A
>> who

Your variables are:

M          ans        m          z

>>
```

`clear` with no arguments deletes all your variables

```
>> clear
>> who
Your variables are:

>>
```

Ouput formats

The function `format` changes the precision of the output

```
>> pi
ans =
    3.1416

>> format long; pi
ans =
   3.14159265358979

>> format short e; pi
ans =
   3.1416e+00

>> format long e; pi
ans =
     3.141592653589793e+00

>> format rat; pi
ans =
   355/113
```

Output formats

```
>> help format

FORMAT Set output format.
  All computations in MATLAB are done in double precision.
  FORMAT may be used to switch between different output
  display formats as follows:
    FORMAT         Default. Same as SHORT.
    FORMAT SHORT   Scaled fixed point format with 5 digits.
    FORMAT LONG    Scaled fixed point format with 15 digits.
    FORMAT SHORT E Floating point format with 5 digits.
    FORMAT LONG E  Floating point format with 15 digits.
    FORMAT HEX     Hexadecimal format.
    FORMAT +       The symbols +, - and blank are printed
                   for positive, negative and zero elements.
                   Imaginary parts are ignored.
    FORMAT BANK    Fixed format for dollars and cents.
    FORMAT COMPACT Suppress extra line-feeds.
    FORMAT LOOSE   Puts the extra line-feeds back in.
    FORMAT RAT     Approximation by ratio of small integers.
```

Creating matrices

```
>> A = [3 4; 7 6]
A =
     3     4
     7     6
>>

>> A = [1, -4*j, sqrt(2);
log(-1) sin(pi/2) cos(pi/3)
asin(0.5), acos(0.8) exp(0.8)]
A =
   1.0000                  0 - 4.0000i   1.4142
        0 + 3.1416i   1.0000                  0.5000
   0.5236                  0.6435                  2.2255
>>
```

Matrix operators

```
>> A = [1 3; 5 9]; B = [4 -7; 10 0];
>> A+B
ans =
     5    -4
    15     9
>> A*B
ans =
    34    -7
   110   -35
>> b=[1;5];
>> A*b
ans =
    16
    50
>> A'
ans =
     1     5
     3     9
>>
```

Element-by-element array operators

```
.*  multiplication
./  division
.^  power
```

```
>> A=[1;2;3]; B=[-6;7;10];
>> A*B
??? Error using ==> *
Inner matrix dimensions must agree.

>> A.*B
ans =
    -6
    14
    30
>> A.^2
ans =
     1
     4
     9
>>
```

Colon notation

To create a vector `x` with initial value `xi`, increment `dx` , and final value `xf`, use the colon notation:

```
x = [xi: dx : xf];
```

Examples

```
>> i = 1:5
i =
    1     2     3     4     5
>> x = 0.1:0.1:1
x =
  Columns 1 through 4
    0.1000    0.2000    0.3000    0.4000
  Columns 5 through 7
    0.5000    0.6000    0.7000
  Columns 8 through 10
    0.8000    0.9000    1.0000
```

Understanding colon notation is essential to mastering matlab

Graphics

Basic plotting commands

Line types and colors

Enhancements to beautify your plots

Using `hold` and `subplot`

Setting the axis limits: `axis` and `zoom`

Basic plotting commands

Four types of 2-D plots:

`plot(x,y)` plots the vector x vs. y

`semilogx(x,y)` makes a plot with x-axis $\log_{10}$ and y axis linear

`semilogy(x,y)` makes a plot with x-axis linear and y axis $\log_{10}$

`loglog(x,y)` makes a plot with both axes $\log_{10}$

Line types and sizes

Various line types, plot symbols and colors may be obtained with `PLOT(X,Y,S)` where `S` is a 1, 2 or 3 character string made from the following characters:

y	yellow	.	point
m	magenta	o	circle
c	cyan	x	x-mark
r	red	+	plus
g	green	-	solid
b	blue	*	star
w	white	:	dotted
k	black	-.	dashdot
		--	dashed

For example, the following makes a plot of x vs. y using blue pluses

```
plot(x,y, 'b+')
```

Additional plotting commands

- `title('text')` — add title
- `xlabel('text')` — add xlabel
- `ylabel('text')` — add ylabel
- `text(p1, p2, 'text', 'sc')` — puts `'text'` at (`p1, p2`) in screen coordinates where (0.0, 0.0) is the lower left and (1.0, 1.0) is the upper right of the screen
- `subplot` — subdivides the window.

Additional plotting commands

- `axis` — change axes

- `axis('equal')` — equal aspect ratio

- `grid` — adds grid lines

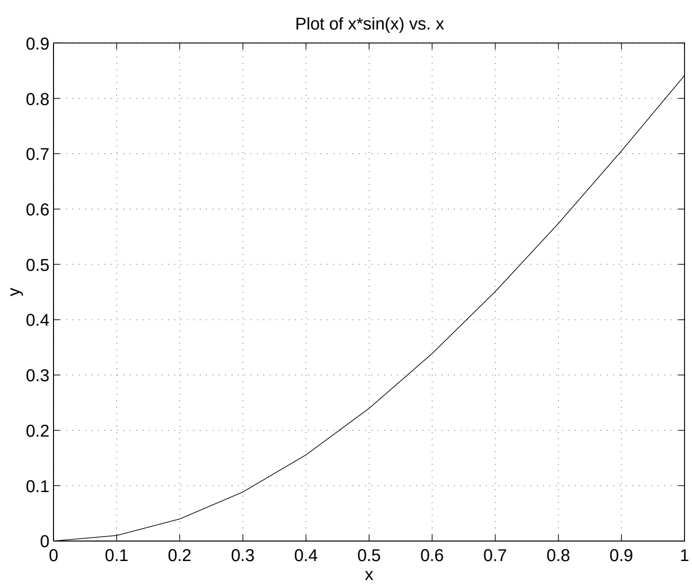
- `hold` — allows you to make multiple plots on the same subplot

- `zoom` — enables zoom (using mouse)

Note: `grid`, `hold`, and `zoom` operate like a "toggle" (successive calls turn the property on or off)

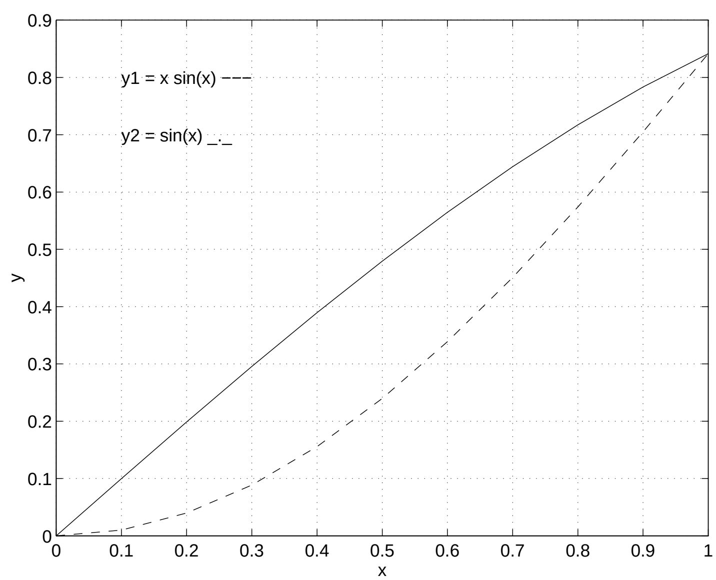
Example of a simple plot

```
>> x = [0:0.1:1]';
>> y = x.*sin(x);
>> clf                 % clears the current figure
>> plot(x,y)
>> title('Plot of x*sin(x) vs. x');
>> xlabel('x')
>> ylabel('y')
>> grid
>> print -deps plot1.eps
>> !lp plot1.eps
request id is test-878 (1 file(s))
```



Another simple plot

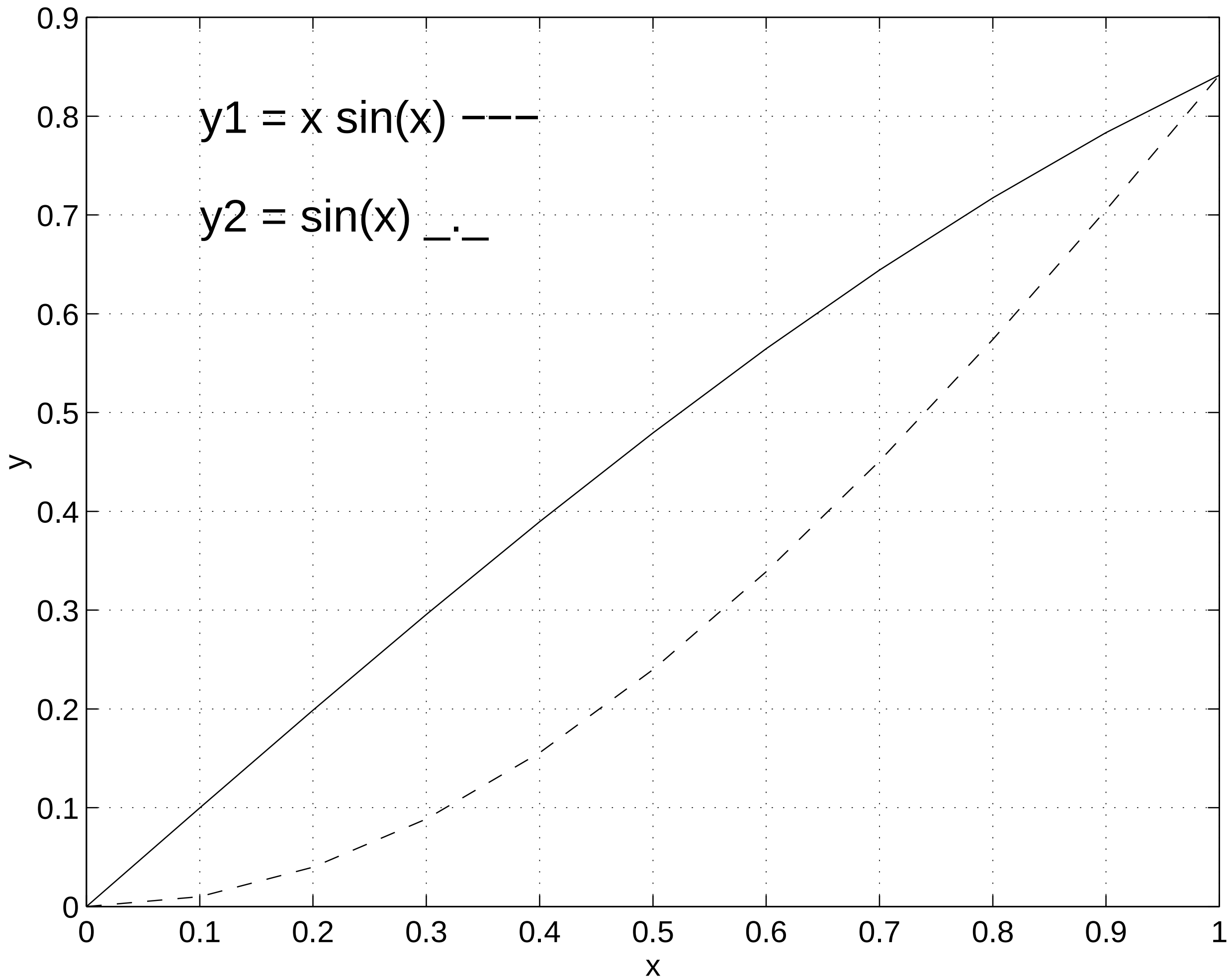
```
>> x = [0:0.1:1]';
>> y1 = x.*sin(x);
>> y2 = sin(x);
>> plot(x, y1, '--', x, y2, '-')
>> f1
>> text(0.1,0.8,'y1 = x sin(x) ---');
>> text(0.1,0.75,'y2 = sin(x) _._');
>> xlabel('x'); ylabel('y'); grid;
```



Using get and set

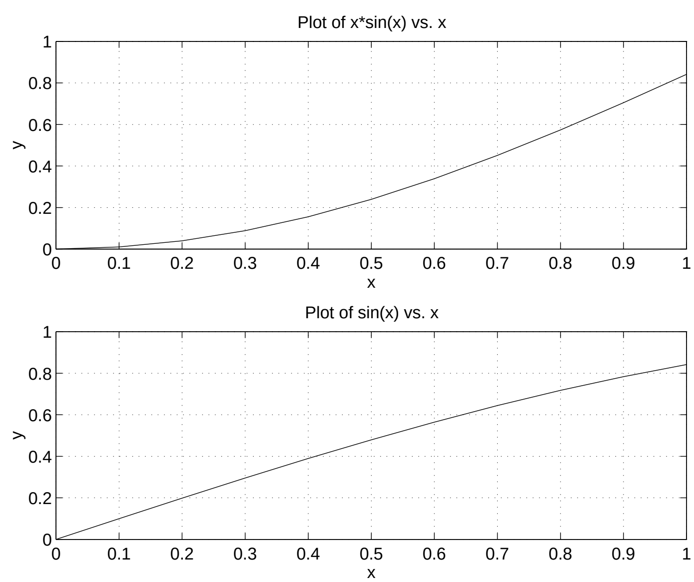
```
>> t1 = text(0.1,0.8,'y1 = x sin(x) ---');
>> t2 = text(0.1,0.7,'y2 = sin(x) _._');
>> get(t1)
        Color = [1 1 1]
        EraseMode = normal
        Extent = [0.0900693 0.730205 0.489607 0.117302]
        FontAngle = normal
        FontName = Helvetica
        FontSize = [18]
        FontWeight = normal
        HorizontalAlignment = left
        Position = [0.1 0.8 0]
        Rotation = [0]
        String = y1 = x sin(x) ---
        Units = data
        VerticalAlignment = middle

        ButtonDownFcn =
        Children = []
        Clipping = off
        Interruptible = no
        Parent = [58.0005]
        Type = text
        UserData = []
        Visible = on
>> set(t1, 'FontSize', 18)
>> set(t2, 'FontSize', 18)
```

Using `get` and `set`



Use of subplot

```
>> subplot(211)
>> plot(x,y1);
>> xlabel('x'); ylabel('y'); grid;
>> title('Plot of x*sin(x) vs. x');
>> subplot(212)
>> plot(x,y2);
>> xlabel('x'); ylabel('y'); grid;
>> title('Plot of sin(x) vs. x');
>> print -deps plot3.eps
```



Control flow — decisions

Matlab commands for decisions:

`if`, `elseif`, `else`, and `end`

Example

```
d = date;
day = str2num(d(1:2));
if ((floor(day/2)*2) == day)
  disp('the day of the month for today is even');
else
  disp('the day of the month for today is odd');
end
```

Output

```
>> date
ans =
23-Feb-97
>> d = date;
>> day = str2num(d(1:2));
>> if ((floor(day/2)*2) == day)
  disp('the day of the month for today is even');
else
  disp('the day of the month for today is odd');
end
the day of the month for today is odd
```

Control flow — loops

Matlab commands for loops:

`for` and `while`

Examples

```
% compute factorial with for loop
n = 10;
factorial=1;
for i=1:n
    factorial = factorial * i;
end
factorial

% compute factorial with while loop
i=1;
factorial=1;
while (i < 10)
  i = i+1;
  factorial = factorial * i;
end
factorial
```

Scripts and functions

Matlab scripts and functions are called `M-files`, because they have the suffix ".m"

Scripts are text files that contain a sequence of matlab commands.

Functions are `M-files` that return values.

The biggest difference between scripts and functions is that variables created in functions are local variables, whereas variables created in scripts are global.

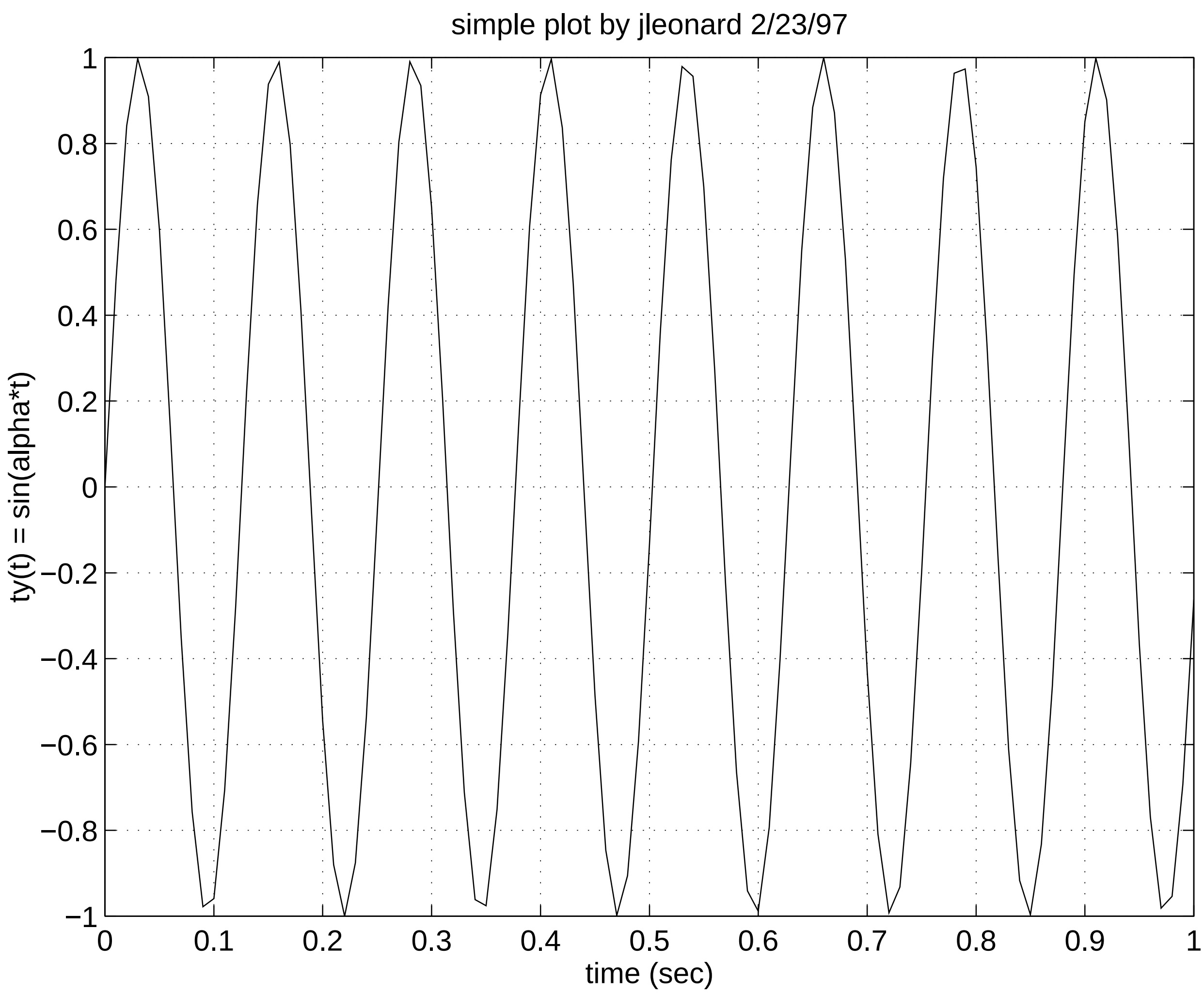
The matlab toolboxes are collections of useful `M-files`.

Writing your own scripts and functions makes it easier and more efficient to use matlab

A simple matlab script

```
% simple.m - a simple matlab script
%
% This script makes a simple plot of the sin
% function.  It assumes alpha is defined
% in the workspace before the script is called

t = [0:0.01:1];
y = sin(alpha * t);
plot(t,y)
xlabel('time (sec)');
ylabel('ty(t) = sin(alpha*t)');
title('simple plot by jleonard 2/23/97');
grid on

>> simple
>> help simple

  simple.m - a simple matlab script

  This script makes a simple plot of the sin
  function.  It assumes alpha is defined
  in the workspace before the script is called

>>
```

Graph produced by simple.m



Example: dolphin sonar beampattern

```
% script file to make beam pattern to distribute in class
% jleonard 10/20/96
f = 120000;
lambda = 1500/f;
theta = (-90:0.01:90) * pi / 180;
k = 2 * pi/lambda;
a = 0.037/2;
theta3 = 29.3 * lambda / (2 * a);

figure(1)
clf
bp = cpbeam(theta, k, a);
bp1 = bp;
plot(theta*180/pi,10 * log10(bp))
axis([-90 90 -80 0]);
set(gca, 'Xtick', [-90:30:90]);
grid on
hold on
plot([theta3 theta3], [-80 0], '-.')
plot([-theta3 -theta3], [-80 0], '-.')
title(['f = 120 kHz  D = ', num2str(2 * a), ...
      ' m  beamwidth = +-',num2str(theta3), ' deg']);
xlabel('theta (degrees)');
ylabel('Normalized source level (dB)');
```

cpbeam.m

```
function bp = cpbeam(theta,k,a)

% CPBEAM:  Beam pattern for circular piston transducer,
% using standard bessel function model.  There are two
% usages:
%    bp = cpbeam(theta,k,a)
%    bp = cpbeam(theta,ka)
%
% CPBEAM returns the normalized beam function for
% wavenumber k and transducer radius a at a boresight
% angle theta (radians).
% author: Bradley A. Moran, MIT Sea Grant, 1993

if nargin < 3, a = 1; end

reducedFreq = k*a*sin(theta);

bp = (2*bessel(1,abs(reducedFreq))./reducedFreq).^2;
```

Beam pattern