

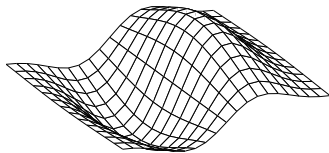
Matlab Tutorial

CS335 - Computational Methods in Business and Finance

Fall 2016

Outline

- Matlab Overview
- Useful Commands
- Matrix Construction and Flow Control
- Script/Function Files



- Basic Graphics

Getting to Matlab

- Everyone who is registered in CS335 should have an account in the CS undergrad environment
- This permits you to login to any machine (Macs, xterms) in the 2nd and 3rd floor of MC
- You can also login into the CS machines from home
`http://www.cs.uwaterloo.ca/cscf/student/`
- You will need to use a CS machine to access Matlab
- Your Watiam passwd and userid should work
- Problems: see consultants in MC3011

Getting to Matlab II

You can access Matlab on any of CS undergrad machines (labs in the second and third floor of MC). Alternatively, you can use MATLAB remotely. See Web for tutorials for doing so under Linux, Mac OS X and Windows.

Note that there are only a limited number of MATLAB licenses available in the student environment. Access to a MATLAB license may be difficult at peak usage times.

What is Matlab?

According to *The Mathworks*:

MATLAB is an integrated technical computing environment that combines numeric computation, advanced graphics and visualization, and a high-level programming language.

MATLAB includes hundreds of functions for:

- *Data analysis and visualization*
- *Numeric and symbolic computation*
- *Engineering and scientific graphics*
- *Modeling, simulation, and prototyping*
- *Programming, application development, and GUI design*

Getting Started

- Web resources
 - CS335 Course Web page (Matlab Primer)
 - www.mathworks.com
- Books
 - *Mastering Matlab 5/6/7/8/9*, D. Hanselman, B. Littlefield
 - *Introduction to Scientific Computing*, Van Loan
 - See also CS370 Web site for other sources

Running Matlab

- Macs/PCs (running Matlab locally)
 - `type: matlab`
- Using MATLAB remotely on Linux and OS X. This assumes you have the ssh binary in your path and an X server running. If you do not know what that means, do not worry! You probably do.
 - 1 Run `ssh -X USERNAME@linux.student.cs.uwaterloo.ca`, where USERNAME is your WatIAM username.
 - 2 You will be prompted to enter a password. This is your corresponding WatIAM password.
 - 3 Once you are authenticated, you will be presented with a shell. Run `matlab -nodesktop -nosplash`.
- Using MATLAB remotely on Windows
 - If you are already familiar with the Remote Desktop Client on Windows, you may consider using that to log on to `windows.student.cs.uwaterloo.ca`.
 - Otherwise, you can use the PuTTY terminal emulator (see Web)
- Use Matlab 6 or later for all assignments

How does Matlab work?

- Interactive environment
- Type commands at the prompt ('>>' typically)
- Case sensitive
- External programs/functions are in M-files (text files with a .m extension)
- Execute M-files by typing the filename (without the .m)
- Note: Almost everything in Matlab is an external function (use the which command to locate the source)

Basic Operations

- 'Matrix' is the only data type
- Vectors are $1 \times N$ or $N \times 1$ matrices
- Scalars are 1×1 matrices
- Addition and subtraction operate entry-wise, while
 $\quad * \quad ^ \quad \backslash \quad /$
 are matrix operations (unless preceded by a dot).
- Matrices and vectors are *1-offset*

Basic Example 1

```
>> A = [1 2 3 ; 4 5 6]
```

```
A =
```

1	2	3
4	5	6

```
>> test = A*A
```

```
??? Error using ==> *
```

```
Inner matrix dimensions must agree.
```

```
>> test = A*A'
```

```
test =
```

14	32
32	77

Basic Example 2

```
>> A = [1 2 ; 3 4]
```

```
A =
```

```
    1    2
    3    4
```

```
>> A^2
```

```
ans =
```

```
    7    10
   15    22
```

```
>> A.^2
```

```
ans =
```

```
    1    4
    9   16
```

Transposes

- Strictly, A' is *complex conjugate transpose* of A
- Usual (non-conjugate) transpose is $A.'$
- `>> A = [1+i, 2+2i, 3+3i]`

`A =`

1.0000 + 1.0000i 2.0000 + 2.0000i 3.0000 + 3.0000i

`>> A'`

`ans =`

1.0000 - 1.0000i

2.0000 - 2.0000i

3.0000 - 3.0000i

`>> A.'`

`ans =`

1.0000 + 1.0000i

2.0000 + 2.0000i

3.0000 + 3.0000i

More dots

```
>> A = [1 2; 3 5]
```

```
A =
```

```
    1    2
```

```
    3    5
```

```
>> B = [-5 2; 3 -1]
```

```
B =
```

```
   -5    2
```

```
    3   -1
```

```
>> A*B
```

```
ans =
```

```
    1    0
```

```
    0    1
```

```
>> A.*B
```

```
ans =
```

```
   -5    4
```

```
    9   -5
```

Basic Example 3 - Solving $Ax=b$

```
>> A = [1,15,4; 2,15,20; 3,30,9];  
>> b = [1;22;9];  
>> x=A\b
```

```
x =  
    6.0667  
   -0.5867  
    0.9333
```

```
>> x=inv(A)*b
```

```
x =  
    6.0667  
   -0.5867  
    0.9333
```

Useful commands

- `help` - Obtain help for a specific function
- `lookfor` - Keyword search of help text
- `more {on/off}` - Paging
- `clear` - Remove variables
- `close` - Close figure windows
- `whos` - List currently defined variables
- `format` - Set output format (e.g., number of digits)
- `%` - comment line in an M-file

help

- `help function` - Gives detailed information about 'function'
- Displays the comments at the top of the M-file
- Some of the help screens read like UNIX man pages
- Related items are listed at the end
- Despite the help text, all commands are lower case
- Useful command to use when you are stuck
- `help` - Provides a list of topics which can then be searched

lookfor

- First command to use when you are stuck
- `lookfor XYZ` - Searches the first comment line for the string XYZ
- Useful if you do not know the function name, but expect that the function exists
- Can be slow

more

- `more {on/off}`
- Turn screen paging on or off
- Works like the Unix `more` command

clear

- `clear X` - Remove the variable `X`
- `clear X*` - Remove all variables starting with string `X`
- `clear` - Remove all variables
- `clear all` - Removes everything (variables, functions, globals and MEX links)
- Often useful at the beginning of script files
- To clear command window: `clc`

close

- `close` - Close the current figure
- `close all` - Close all figure windows
- Useful at the start of script files

whos

- who - list all variables
- whos - list all variables, with size information

```
>> whos
```

Name	Size	Bytes	Class
ans	1x17	34	char array
x	14x21	2352	double array
y	14x22	2464	double array
z	14x21	2352	double array

Grand total is 913 elements using 7202 bytes

- Useful if you keep getting array size mismatches (remember that Matlab is 1-offset)

format

- ```
>> 1/3
ans =
 0.3333
```
- ```
>> format long
>> 1/3
ans =
    0.3333333333333333
```
- ```
>> format short e
>> 1/3
ans =
 3.3333e-01
```
- ```
help format
```

Command line tricks

- Up/Down arrow keys to cycle through commands
- Partially typing a command and hitting up arrow will search the command stack
- Can type multi-line commands, but each line is saved separately (ie. not very useful for re-entering loop commands)
- A command can span two lines by using ... at the end of the first line

Constructing Matrices

- Type in all the numbers directly (semi-colons or new lines create new rows)
- Use ones or zeros
- Use the colon notation
 - start:step:final (e.g. $3:2:7 = [3 \ 5 \ 7]$)
 - steps can be negative (e.g. $7:-2:3 = [7 \ 5 \ 3]$)
 - start:final assumes a step of 1
 - colon by itself means 'all' (eg. $A(1,:)$ is all entries in row 1)
- A variety of other methods exist (load, algebra, other functions)
- Note that vectors and arrays are dynamic

Example

```
>> m1 = zeros(1,3)
```

```
m1 =
```

```
    0    0    0
```

```
>> m2 = ones(3)
```

```
m2 =
```

```
    1    1    1  
    1    1    1  
    1    1    1
```

```
>> m3(2:3,:) = [m2(3,:); [1:1:3]]
```

```
m3 =
```

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

Dimensions of Matrices and Vectors

- `size(A)` for matrices, `length(x)` for vectors

- ```
>> A = [1 2 3; 4 5 6]
```

```
A =
```

```
 1 2 3
 4 5 6
```

```
>> [m n] = size(A)
```

```
m =
```

```
 2
```

```
n =
```

```
 3
```

```
>> x = [1 2 3 4]
```

```
x =
```

```
 1 2 3 4
```

```
>> length(x)
```

```
ans =
```

```
 4
```

# Control Structures

- For statements:

```
FOR I = 1:N,
 FOR J = 1:N,
 A(I,J) = 1/(I+J-1);
 END
END
```

- While loops

```
WHILE X > 1,
 X = X - 1;
END
```

## Control Structures (cont.)

- IF statements

```
IF expression
 statements
ELSEIF expression
 statements
.
.
.
ELSE
 statements
END
```

# Relational and Logical Operators

- Relational operators

< <= > >= == ~= (in C: !=)

- Logical operators

|     | Matlab | C  |
|-----|--------|----|
| AND | &      | && |
| OR  |        |    |
| NOT | ~      | !  |

- >> A = 1:9

A =

1      2      3      4      5      6      7      8      9

>> tf = (A>2)&(A<6)

tf =

0      0      1      1      1      0      0      0      0

## Vectorizing Loops

```
>> cs335marks = [24 36 11 42 33 55 30];
>> for i=1:length(cs335marks)
 cs335marks(i) = 10*cs335marks(i)^(1/2);
end
>> cs335marks
cs335marks =
 48.9898 60.0000 33.1662 64.8074 57.4456
 74.1620 54.7723

>> cs335marks = [24 36 11 42 33 55 30];
>> cs335marks = 10*cs335marks.^(1/2)
cs335marks =
 48.9898 60.0000 33.1662 64.8074 57.4456
 74.1620 54.7723
```

## Script files

- Matlab commands can be placed in text files with .m extensions
- The commands are interpreted/executed when the filename is typed at the Matlab prompt (no .m extension)
- The effect is identical to typing the commands (i.e. all new variables remain, all old variables are accessible)
- Convenient if the same set of commands need to be executed with minor changes
- Commonly used for 'driver' programs on assignments

## Script Example

```
clear all;
close all;

% Initial data
x = [9 8 7 3 1 1 2 5 8 7 5];
y = [4 2 1 2 5 7 9 11 9 8 7];
n = length(x);

% Initialize t
t = zeros(size(x));

% Choose t to be arclength
for i = 2:n
 dt = sqrt((x(i)-x(i-1))^2 + (y(i)-y(i-1))^2);
 t(i) = t(i-1) +dt;
end
```



## Function Files

- Defined in text files with .m extensions
- Called by typing the filename (no .m)
- Functions do not have access to existing variables (separate scope)
- Functions can accept/return zero or more values
- Control is lost when the end of the file is reached, or the command return is encountered

## Function Example

```
function [newmarks] = bell(oldmarks, method)
% Whatever appears here is displayed when the user
% types 'help bell'

% This line will not appear in the help text
if method == 1
 newmarks = 10*oldmarks.^(1/2);
elseif method == 2
 newmarks = oldmarks + 10*ones(1, length(oldmarks));
else
 newmarks = oldmarks;
end
return
```

## Function Example

```
>> help bell
```

Whatever appears here is displayed when the user types 'help bell'

```
>> m = [23 67 43 49 75 55];
```

```
>> bell(m,1)
```

```
ans =
```

```
47.9583 81.8535 65.5744 70.0000 86.6025 74.1620
```

```
>> m_new = bell(m,2)
```

```
m_new =
```

```
33 77 53 59 85 65
```

# Debugging

- See `help debug`
- Set a breakpoint with `dbstop`
- Trace through the execution with `dbstep`
- Show the execution stack with `dbstack`
- Continue execution with `dbcont`
- Quit debugging with `dbquit`

# Text Strings

- Use single quotes to define text: 'string'
- Use `disp` to display text without the associated variable name (also works for variables)
- Can have an array of strings if each string has the same length
- Can convert from numbers to strings using the `num2str` command

```
>> a = 1;
>> b = 5;
>> t = ['Plot ' num2str(a) ' of ' num2str(b)];
>> disp(t)
Plot 1 of 5
```

# Graphics

- Matlab has excellent graphics support for experimenting with data
- Since the data is 'live', you can quickly and easily change plots and figures
- Figure windows can easily be saved and printed (as eps or pdf for assignments)
- Figures can be edited by clicking on edit in Figure Window

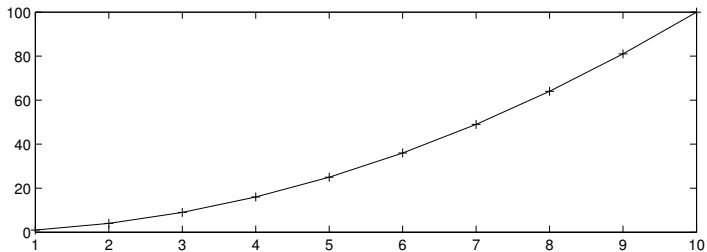
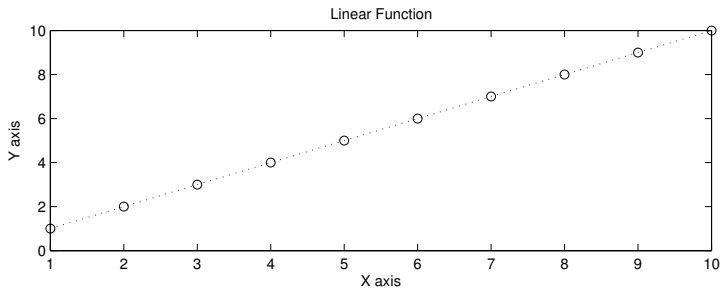
# Plots

- `plot(x,y)` - Basic plotting command
- `plot(x,y,'opts')` - `opts` specifies characteristics of the curve (color, style and data markers)
- `help plot` - Details on options available
- Can plot multiple curves on a single figure:  
`plot(x1,y1,'opt1',x2,y2,'opt2')`  
or use `hold on`
- Can add title, axis labels and legend with appropriate commands

## 2D plots

```
>> x = [1:1:10];
>> y_lin = x;
>> y_quad = x.^2;
>> subplot(2,1,1), plot(x,y_lin,'bo:')
>> title('Linear Function')
>> xlabel('X axis')
>> ylabel('Y axis')
>> subplot(2,1,2), plot(x,y_quad,'r+-')
>> print -deps fig1.eps
>> close
```

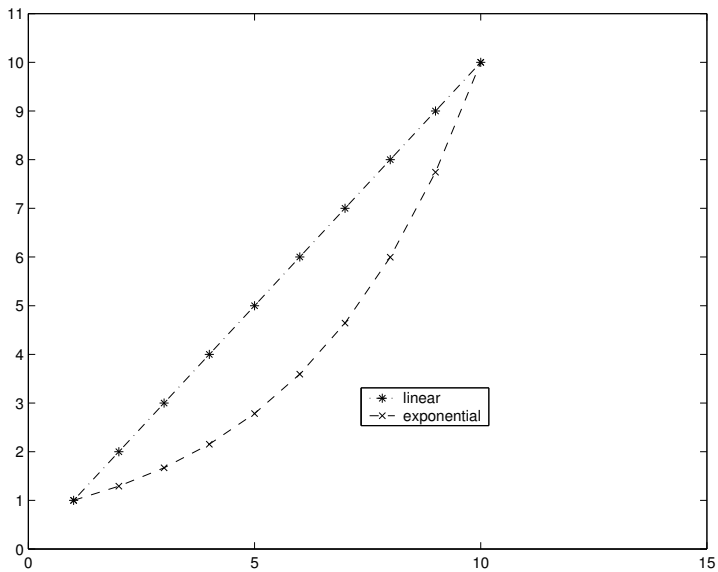




## 2D plots (cont.)

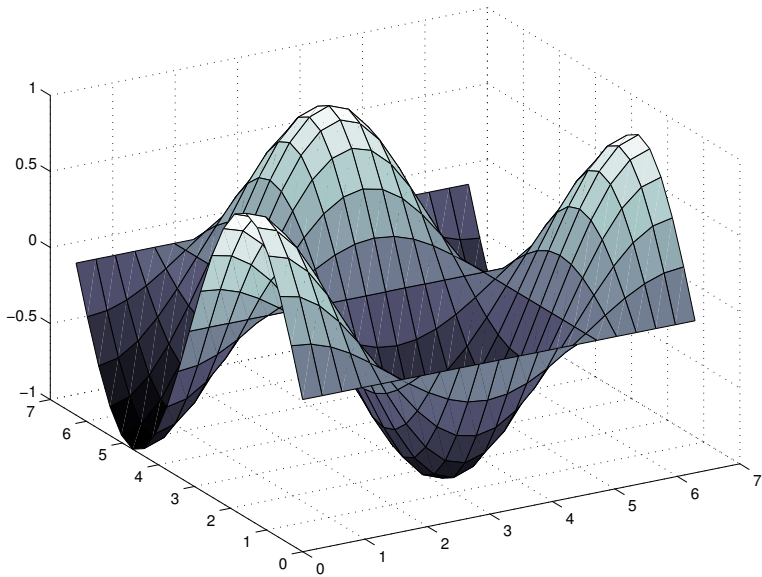
```
>> x=linspace(1,10,10);
>> y_lin = x
y_lin =
 1 2 3 4 5 6 7 8 9 10
>> y_log = logspace(0,1,10) % 10^[equally spaced 0..1]
y_log =
Columns 1 through 6
 1.0000 1.2915 1.6681 2.1544 2.7826 3.5938
Columns 7 through 10
 4.6416 5.9948 7.7426 10.0000

>> plot(x,y_lin,'*-.')
>> hold on
>> plot(x,y_log,'x--')
>> axis([0 15 0 11])
>> legend('linear', 'exponential')
```



## 3D plots

```
>> figure
>> x=[0:2*pi/20:2*pi];
>> y=x;
>> z=sin(x)'*cos(y);
>> surf(x,y,z)
>> colormap('bone')
>> view(-30,30)
>> print -deps mesh3d.eps
```



# Efficiency Issues

- Vectorize loops whenever possible
- Pre-allocate arrays whenever possible
- We will be checking for efficient code on assignments if we mention this specifically
- Otherwise, don't worry too much about this (but your code may take a long time (: )

## Vectorization Example: Monte Carlo Simulation

Slow code:

```
...
S_new = zeros(N_sim,1);

for m=1:N_sim % simulation loop
 S = S_init;

 for i=1:N % timestep loop
 S = S + S*(drift + sigma_sqrt_delt*randn(1,1));
 S = max(0.0, S);
 % check to make sure that S_new cannot be < 0
 end % timestep loop

 S_new(m,1) = S;

end % simulation loop
```

## Vectorization Example: Monte Carlo Simulation

Fast code:

```
...
 S_new = zeros(N_sim,1);
 S_old(1:N_sim,1) = S_init;

for i=1:N % timestep loop
 % now, for each timestep, generate info for
 % all simulations
 % now, only one explicit loop, second loop
 % replaced by vector commands

 S_new(:,1) = S_old(:,1) +...
 S_old(:,1).*(drift + sigma_sqrt_delt*randn(N_sim,1));

 S_new(:,1) = max(0.0, S_new(:,1));
 % check to make sure that S_new cannot be < 0

 S_old(:,1) = S_new(:,1);
end % timestep loop
```



## Once Again:: Matlab is Matrix Oriented

Most common source of errors

- All entities in Matlab are matrices by default
- A common cause of errors: size mismatch

```
>> a = 1;
>> size(a)
ans =
 1 1
```

This sometimes causes unexpected results when multiplying objects

- There is a difference between a row vector and a column vector!
- Usual rules for matrix multiplication must be followed

## Examples:

```
>> a = [1 2 3]; b = [4 5 6];
```

```
>> a'*b
```

```
ans =
```

```
 4 5 6
 8 10 12
 12 15 18
```

```
>> a*b'
```

```
ans =
```

```
 32
```

```
>> a*b
```

```
??? Error using ==> mtimes
```

```
Inner matrix dimensions must agree.
```

# Summary

- Use help and lookfor on a regular basis
- Use more on and semi-colons to maintain an intelligible display
- When interpreting error messages, remember that all variables are matrices
- Use script files and functions to automate repetitive tasks (anything over 5 lines should probably be in an M-file)
  - On assignments, you should hand in hard copy of all M-files used
- Try to use operations on vectors/matrices, instead of loop constructs