MATLAB Course

Adriana Hera
ahera@wpi.edu
Computing & Communications Center

- October 2010-

Part of the notes are from Matlab documentation

MATLAB Refresher Course

1. Variables, Operators
2. Matrices
3. Matlab Functions
4. Relational operators & Loops (Flow Control)
5. Scripts
6. User Defined Functions
7. Visualization
8. Curve fitting: Polynomial curve fitting
9. Interpolation
10. Publishing a script to HTML
What is Matlab?

MATLAB® is a high-performance language for technical computing.

It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation.

MATLAB stands for matrix laboratory.

MATLAB is an interactive system whose basic data element is an matrix (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.

MATLAB - Typical Uses

- Math and computation
- Algorithm development
- Data acquisition
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building
Starting Matlab

- **Windows:** Start menu → Matlab → Matlab
- **Unix:** Terminal window → type `matlab`

Matlab Help

1. Using HELP menu → MATLAB Help
   HELP → Using Help Browser

2. `>> helpdesk` Opens the Help browser.

3. `>> help commandname/toolboxname/functionname`
   Ex: `>> help sin`

4. `>> doc commandname/toolboxname/functionname`
   displays the detailed info in the Help browser.
   Ex: `>> doc sin`

Other commands:
5. `>> lookfor = helpdesk` → search
I. Matlab Programming

- Matlab Variables
- Numbers
- Operators
- Functions

.....

Matlab Variables

- A MATLAB variable is essentially a tag that you assign to a value in memory.
- MATLAB does not require any type declarations or dimension statements.
- When MATLAB encounters a new variable name, it automatically creates the variable and allocates the appropriate amount of storage.
- If the variable already exists, MATLAB changes its contents.
- Variable names consist of a letter, followed by any number of letters, digits, or underscores.
- MATLAB uses only the first 64 characters of a variable name.
- **MATLAB is case sensitive; it distinguishes between uppercase and lowercase letters.**
- MATLAB stores variables in a part of memory called workspace.
- To view what is stored in a variable type its name.

Types of Variables: MATLAB provides three basic types of variables:
- Local Variables
- Global Variables
- Persistent Variables
Matlab Variables

Rules for variable names:
- Make Sure Variable Names Are Valid
- Don't Use Function Names for Variables
- Check for Reserved Keywords
- Avoid Using i and j for Variables

Syntax:
```
varialeName=Value;
```

Example:
```
>> a=5;
>> b=7;
>> +b
```

```
>> c=a+b
```

```
>> method='linear'
```

How to remove a variable from workspace:
```
>> clear variableName
>> clear
- removes all variables from the workspace (!!!!)
```

ans = default variable, when the result is not assign to a variable

Exercise: 1. Define
```
a1=8 and b2=8, c1=a1+b2
```

2. Other commands:
```
variable= input('prompt') (>>help input)
>> a3=input('Please enter the value of a3:')
```

Operators

Expressions use familiar arithmetic operators and precedence rules.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+</code></td>
<td>Addition</td>
</tr>
<tr>
<td><code>-</code></td>
<td>Subtraction</td>
</tr>
<tr>
<td><code>*</code></td>
<td>Multiplication</td>
</tr>
<tr>
<td><code>/</code></td>
<td>Division</td>
</tr>
<tr>
<td><code>^</code></td>
<td>Power</td>
</tr>
<tr>
<td><code>'</code></td>
<td>Complex conjugate transpose</td>
</tr>
<tr>
<td><code>( )</code></td>
<td>Specify evaluation order</td>
</tr>
</tbody>
</table>
1. Standard elementary mathematical functions

>> help elfun

    Trigonometric (sin, cos)
    Exponential (exp, log)
    Complex (abs, angle)
    Rounding and remainder (round)

2. Elementary matrices and matrix manipulation.

>> help elmat

3. Specialized math functions.

>> help specfun

1. Built-in functions (Ex. sqrt, sin)

Some of the functions, like sqrt and sin, are built in.
Built-in functions are part of the MATLAB core
They are very efficient
The computational details are not readily accessible.
(you cannot see the code)

2. Function implemented in M-files (ex. factorial, mean, det)

You can see the code and even modify it, if you want.

Syntax:

>> outputArgs = functionName(inputArgs)

Related commands: edit
Constants, inf, NaN

Constants

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pi</td>
<td>3.14159265...</td>
</tr>
<tr>
<td>i</td>
<td>Imaginary unit, $\sqrt{-1}$</td>
</tr>
<tr>
<td>1</td>
<td>Same as i</td>
</tr>
<tr>
<td>eps</td>
<td>Floating-point relative precision, $\varepsilon = 2^{-52}$</td>
</tr>
</tbody>
</table>

inf

Infinity: division by zero and overflow, which lead to results too large to represent as conventional floating-point values.

ex: 1/0, 1.e1000

NaN

Not-a-Number: a result of mathematically undefined operations like 0.0/0.0 and inf-inf.

Related commands: edit

I. Matlab Programming

We talked about:

✓ Matlab Variables
✓ Numbers
✓ Operators
✓ Functions

II. Matlab Programming
I. Matlab Programming

- Matrices
- Operators
- Functions

Matrix & basic matrix functions

Define a matrix:

1. Type the matrix
2. Use Specialized Matrix Functions

Matrix Manipulation

Matrix Functions

\[
\begin{array}{cccc}
(1,1) & (1,2) & (1,3) & (1,4) \\
(2,1) & (2,2) & (2,3) & (2,4) \\
(3,1) & (3,2) & (3,3) & (3,4) \\
(4,1) & (4,2) & (4,3) & (4,4) \\
\end{array}
\]
Matrix: Define a matrix

1. Type the matrix

- 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, [].

\[
S = \begin{bmatrix}
3 & -10 & 0 \\
-10 & 0 & 30 \\
0 & 30 & -27
\end{bmatrix}
\]

1. >> \texttt{S=[3 -10 0; -10 0 30; 0 30 -27]}

Basic matrix information: \texttt{size} (size of a matrix)

\[
>> \texttt{[m,n] = size(X)}
\]

Matrix: Define a matrix

2. Use Specialized Matrix Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{ones}</td>
<td>Create a matrix or array of all ones.</td>
</tr>
<tr>
<td>\texttt{zeros}</td>
<td>Create a matrix or array of all zeros.</td>
</tr>
<tr>
<td>\texttt{eye}</td>
<td>Create a matrix with ones on the diagonal and zeros elsewhere.</td>
</tr>
<tr>
<td>\texttt{diag}</td>
<td>Create a diagonal matrix from a vector.</td>
</tr>
<tr>
<td>\texttt{rand}</td>
<td>Create a matrix or array of uniformly distributed random numbers.</td>
</tr>
<tr>
<td>\texttt{randn}</td>
<td>Create a matrix or array of normally distributed random numbers and arrays.</td>
</tr>
</tbody>
</table>

\[
>> \texttt{B=eye(3)}
\]

\[
B = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

\[
>> \texttt{D=diag([23,0,47])}
\]
Matrix: Accessing Matrix Elements

- **individual element**
  \[ \text{>> } S(2,2) \]
  \[
  \begin{bmatrix}
  3 & -10 & 0 \\
  -10 & 0 & 30 \\
  0 & 30 & -27 
  \end{bmatrix}
  \]

- **column**
  \[ \text{>> } S(:,2) \]
  \[
  \begin{bmatrix}
  -10 \\
  0 \\
  30 
  \end{bmatrix}
  \]

- **row**
  \[ \text{>> } S(2,:) \]
  \[
  \begin{bmatrix}
  -10 \\
  0 \\
  30 
  \end{bmatrix}
  \]

- **group of elements**
  \[ \text{>> } S(3,1:2) \]
  \[
  \begin{bmatrix}
  0 \\
  30 
  \end{bmatrix}
  \]

Define a vector

\[ v=[1:0.5:20]; \]
\[ v=[1, 1.5, 2, 2.5, \ldots, 19, 19.5, 20] \]

**linspace**

\[ y = \text{linspace}(a,b,n) \]

generates a row vector \( y \) of \( n \) points linearly spaced between and including \( a \) and \( b \).

**Example:** \[ \text{>> } y=\text{linspace}(2.1, 10, 9) \]
### Matrix: Operations

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>A+B</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>A-B</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>A*B</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>A/B</td>
</tr>
<tr>
<td>\</td>
<td>Left division (described in &quot;Matrices&quot;)</td>
<td>A \ B</td>
</tr>
<tr>
<td>^</td>
<td>Power</td>
<td>A^B</td>
</tr>
<tr>
<td>'</td>
<td>Complex conjugate transpose</td>
<td>A'</td>
</tr>
<tr>
<td>( )</td>
<td>Specify evaluation order</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>.*</td>
<td>Element-by-element multiplication</td>
<td>A.*B</td>
</tr>
<tr>
<td>./</td>
<td>Element-by-element division</td>
<td>A./B</td>
</tr>
<tr>
<td>.\</td>
<td>Element-by-element left division</td>
<td>A.B</td>
</tr>
<tr>
<td>.^</td>
<td>Element-by-element power</td>
<td>A.^B</td>
</tr>
<tr>
<td>.'</td>
<td>Unconjugated array transpose</td>
<td>A.'</td>
</tr>
</tbody>
</table>

#### Examples

A = \[
\begin{bmatrix}
1 & 2 & 3 \\
2 & 3 & 1 \\
2 & 2 & 2
\end{bmatrix}
\], \quad B = \[
\begin{bmatrix}
10 & 20 & 30 \\
11 & 21 & 31 \\
1 & 2 & 3
\end{bmatrix}
\]

\[\text{ans} = \begin{bmatrix}
35 & 68 & 101 \\
54 & 105 & 156 \\
44 & 86 & 128
\end{bmatrix}\]

\[\text{A} \ast \text{B} = \begin{bmatrix}
\begin{bmatrix}
a_{11} & a_{12} & a_{13}
\end{bmatrix} \cdot \begin{bmatrix}
b_{11} & b_{12} & b_{13}
\end{bmatrix} \\
\begin{bmatrix}
a_{21} & a_{22} & a_{23}
\end{bmatrix} \cdot \begin{bmatrix}
b_{21} & b_{22} & b_{23}
\end{bmatrix} \\
\begin{bmatrix}
a_{31} & a_{32} & a_{33}
\end{bmatrix} \cdot \begin{bmatrix}
b_{31} & b_{32} & b_{33}
\end{bmatrix}
\end{bmatrix}\]
Matrix: Functions

Few matrix functions:

- **det** - Determinant
- **linsolve** - Solve linear systems of equations (using LU factorization)

\[ X = A\backslash B \]

- Linear equation solution (\( X = A\backslash B \) is the solution to the equation \( AX = B \) computed by Gaussian elimination)

\[
\begin{align*}
3x + 2x + x &= 2 \\
-x + 2x + 3x &= 5 \\
x - 3x + 6x &= 7
\end{align*}
\]

\[
\begin{align*}
A &= \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 6 \end{bmatrix}; \\
b &= \begin{bmatrix} 2 \\ 5 \\ 7 \end{bmatrix};
\end{align*}
\]

1. \( x = A\backslash b \)
2. \( x = \text{linsolve}(A,b) \)
3. \( x = \text{inv}(A) \ast b \) Not recommended!!!

\( x = A\backslash b \) is the solution to the equation \( Ax = b \) computed by Gaussian elimination

**linsolve** - Solve linear systems of equations (using LU factorization)
Matlab Programming

We talked about:

- Matrices
- Operators
- Functions

IV. Matlab Programming

- How to write a program (M-files)
- Script
- Function
- How to plot data
M-files

- Files that contain code in the MATLAB language are called M-files.
- You create M-files using a text editor.
- Use a M-file as any other MATLAB function or command.
- A M-file is a plain text file.

Two kinds of M-files:

Scripts

do not accept input arguments or return output arguments
operate on data in the workspace.

Functions

can accept input arguments and return output arguments
internal variables are local to the function.

>> edit  fileName
>> edit exSwitch

M-files: Scripts

- do not accept input arguments or return output arguments
- operate on data in the workspace.

>> edit myScript
type the code
File/Save
>> myScript  Example
(to run the script type its name)

To practice:
* Use command window
• To present hw write scripts or functions (.m files)

% comments
clear; close; clc
tol=1e-2; x=1;
k=0; f=0; fk=1;
while fk>tol
    fk=x/(factorial(k));
    f=f+fk;
    k=k+1;
end
disp(['f=', num2str(f), '   k=',...
     num2str(k)])
plot

Syntax:
plot(y); plot(x,y); plot(x,y,s)

The plot function has different forms, depending on the input arguments.

If $y$ is a vector, \texttt{plot(y)} produces a piecewise linear graph of the elements of $y$ versus the index of the elements of $y$.

If you specify two vectors as arguments, \texttt{plot(x,y)} produces a graph of $y$ versus $x$.

\texttt{plot(x,y,'s');} allows to plot: colors, symbols, different lines

<table>
<thead>
<tr>
<th>Color</th>
<th>Symbol</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>.</td>
<td>point</td>
</tr>
<tr>
<td>g</td>
<td>o</td>
<td>circle</td>
</tr>
<tr>
<td>r</td>
<td>x</td>
<td>x-mark</td>
</tr>
<tr>
<td>c</td>
<td>+</td>
<td>plus</td>
</tr>
<tr>
<td>m</td>
<td>*</td>
<td>star</td>
</tr>
<tr>
<td>y</td>
<td>s</td>
<td>square</td>
</tr>
<tr>
<td>k</td>
<td>d</td>
<td>diamond</td>
</tr>
</tbody>
</table>

\texttt{plot(x,y,'c+:')} plots a cyan dotted line with a plus at each data point;
clear
t=0:0.01:10; % time seconds
signalSin=sin(2*pi*t); % signal1 - frequency =1 Hz
signalCos=0.5*cos(2*pi*t); % signal2 - frequency =1 Hz

figure
plot(t,signalSin);
hold on
plot(t,signalCos, '-*r');

xlabel('time');  ylabel('signal');
legend('Sin', 'Cos');
title('Two Signals','FontSize',12)

Other commands: figure  xlabel  ylabel  legend,  title

Matlab - Plotting

Two Signals

Matlab - Plotting
Relational Operators

The relational operators are <, >, <=, >=, ==, and ~=.

Relational operators perform element-by-element comparisons between two arrays.

→ Logical array with elements set to logical 1 (true) or to logical 0 (false)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
</table>
| <        | Less than            | A < B  | \[
| <=       | Less than or equal to| A <= B | \[
| >        | Greater than         | A > B  | \[
| >=       | Greater than or equal to | A >= B | \[
| ==       | Equal to             | A == B | \[
| ~=       | Not equal to         | A ~= B | \[

Ex:

\[
\text{>> } v = \text{rand}(1, 10000);
\]

\[
\text{>> } v1 = \text{find}(v > 0.5);
\]

\[
\text{>> } v2 = \text{find}(v < 0.5);
\]

Loops (Flow Control)

MATLAB has several flow control commands:

- if, else, and elseif
- switch and case
- for
- while

- continue
- break
- return
Flow Control: **if** … *else*

**if**: conditionally executes statements

```
if relation
    statements 1
else
    statements 2
end
```

Example:

```
a=5; b=7;
if a>b
    disp('a greater than b');
else
    disp('b greater than a');
end
```

```
if expression1
    statements1
elseif expression2
    statements2
else
    statements3
end
```

Flow Control: **for**

The **for** loop executes a group of statements a number of times.

```
for variable = expression
    statements
end
```

Expression:
- first value: last value
- first value: step: last value

**file**: *exFor.m*

```
x=2; % exp(2)
for k = 1:5
    r(k) = x^(k-1)/(factorial(k-1));
end
f=sum(r);
disp(['f=', num2str(f)]);
disp(['r=',num2str(r)]);
```

Change of increment:

```
for k = 1:2:10; statement; end;
for k=10:-1:1; statement; end;
```
The **while** loop executes a group of statements until a logical test is false.

```matlab
tol=1e-2; x=2;
k=1; f=1; fk=1;
while fk>tol
    fk=x^k/(factorial(k))
    f=f+fk;
    k=k+1;
end
disp(['f=', num2str(f), '   k=', num2str(k)])
```

**File: exWhile.m**

\[
e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \cdots + \frac{x^k}{k!} + \cdots
\]

**Flow Control: switch, case**

Ex: Find the structure of the command.

```matlab
>> help switch
>> doc switch
```

My Example:

```matlab
clear
a=6; b=2;
method=input(' method=');
switch method
    case 1
        c=a+b;
    case 2
        c=a*b;
    case 3
        c=a/b;
    otherwise
        disp('no valid method')
end
```

**files: exSwitch.m exSwitch2.m**

Other commands: **input, disp**
Loops: Exit commands

**break**

Lets you exit early from a *for* loop or *while* loop.

In nested loops, break exits from the innermost loop only.

**return**

Terminates the current sequence of commands.

Returns control to the invoking function or to the keyboard.

Ctrl + C

Emergency exit

---

✓ III. Matlab - Programming

We talked about:

✓ Relational operators (*>, <, <=, >=*)

✓ Loops (Flow Control)

✓ *if*, *else*, and *elseif*

✓ *switch* and *case*

✓ *for*

✓ *while*

✓ *continue*

✓ *break*

✓ *return*
Functions are M-files that can accept input arguments and return output arguments.

The M-file and functions should have the same name.

Each M-file function has an area of memory, called the function workspace.

Separate from the MATLAB base workspace, in which it operates.

Matlab functions can be found in:

C:\matlabR14sp2\toolbox\matlab\
FUNCTIONS

Function definition:

```
function [x, y] = day2Fun(A, alfa, f, t);
```

Calling the function:

```
>> [x1, y1] = day2Fun (A1, alfa1, f1, t1);
```

FUNCTIONS

Function definition:

```
function [f, k] = myExp(x, tol)
% function description

k = 1; f = 1; fk = 1;
while fk > tol
    fk = x / factorial(k);
    f = f + fk;
    k = k + 1;
end
disp(['f=', num2str(f), '   k=', num2str(k)])
```

Main code

```
>> [f1, k1] = myExp(x1, tol1)
```

The exponential function can be written as:

\[ e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \cdots + \frac{x^k}{k!} + \cdots \]
FUNCTIONS

Function: Inline functions

There are essentially two ways to create a new function in MATLAB:

1. in a command entered at run-time (inline)
2. or in a file saved to permanent storage.

Inline function, feval

\[ f = \text{inline}('x^2+y^2', 'x', 'y') \]

\[ f(x,y) = x^2+y^2 \]

>> m=f(1,2)

m = 5

Vector form

\[ f = \text{inline}('x.^2+y.^2', 'x', 'y') \]

>> m=f(1,2)

m = 5
**Curve fitting: Polynomial curve fitting**

**polyfit**

finds the coefficients of a polynomial $p(x)$ of degree $n$ that fits the data to

$$p(x) = p_1 x^n + p_2 x^{n-1} + \ldots + p_n x + p_{n+1}$$

**Syntax:**

```matlab
p = polyfit(x, y, n)
```

To see how good the fit is, evaluate the polynomial at the data points with

```matlab
f = polyval(p, x);
```

**Example:**

```matlab
%% generate data %%%%%%
>> x = (0: 0.1: 4)';
>> y = erf(x);
>> figure
>> plot(x, y, 'bo');
>> % % polynomial fitting % %
>> p = polyfit(x, y, 4)

p =
   -0.0135    0.1662   -0.7480    1.4538   -0.0271
```

```matlab
>> f = polyval(p, x);
>> plot(x, y, 'bo', x, f, 'g')
>> hold on
>> legend('original data', 'yfit', 'error')
```

**Advanced commands:**

To fit an arbitrary function, find the parameters by minimizing (fminsearch) the sum of squares of errors between the data and the given function.
Interpolation

**interp1** One-dimensional interpolation

\[ yi = \text{interp1}(x, y, xi, \text{'method'}) \]

Given \((x,y)\) and interpolates to find \(yi\) corresponding to \(xi\)

- **method**:
  - 'nearest' - nearest neighbor interpolation
  - 'linear' - linear interpolation
  - 'spline' - piecewise cubic spline interpolation (SPLINE)
  - 'cubic' - same as 'pchip'

### Interpolation: Example

```matlab
%% example for interpolation %
x=0:0.5:8;
y=sin(x);
xi=0:0.2:8;
yi=interp1(x,y,xi, 'linear');
plot(x, y, 'ob');
hold on;
plot(xi, yi, '*g');
legend('original data', 'data by interp');
```

*myInterp.m*
How to document a Matlab script

Define the boundaries of the cells in a MATLAB script using `%%`

% → comment

Publishing a script to HTML

Editor window → File Menu → Publish fileName

myFit.m
Publishing a script (in various formats)

1. Publish the script in **html format**

```matlab
f1 >> publish('myFit')
ans =
Y:
```

Note: Matlab saves the .html file in the folder html

2. Publish the script in **doc format**

```matlab
f2 >> publish('myFit', 'doc')
ans =
Y:
```

Note: Matlab saves the .doc file in the folder html

3. Publish the script in **pdf format**

```matlab
f3 >> publish('myFit', 'pdf')
ans =
Y:
```

Note: Matlab saves the .pdf file in the folder html
MATLAB Refresher Course

1. Variables, Operators
2. Matrices
3. Matlab Functions
4. Relational operators & Loops (Flow Control)
5. Scripts
6. User Defined Functions
7. Visualization
8. Publishing a script to HTML