COMS W3101–2

Programming Languages: MATLAB

Lecture 1

Spring 2010
Instructor: Michele Merler

Course Information – Instructor

- Michele Merler
  - Email: mmerler@cs.columbia.edu
  - Office: 624 CEPSR
  - Office Hours: TDB

- 3rd year PhD Student in CS Department

- Research Interests:
  - Image & Video Processing
  - Multimedia
  - Computer Vision
Course Information – TA

- Daniel Miau
  - Email: dm2701@columbia.edu
  - Office: TA room
  - Office Hours: Mon 10am – 12pm

- Rohit Sethi
  - Email: rs2990@columbia.edu
  - Office: TA room
  - Office Hours: Wed 3.30pm – 5.30pm
Course Information – Goals

Learn how to use MATLAB for:

- Solve problems in Science and Engineering
- Perform Matrix and Vector Operations
- Compute Complex Mathematical Functions
- Plotting and Visualization
- Perform Simulations and Prototyping
Course Information – Syllabus

- Week 1 – March 2
  - Data Structures (Variables, Vectors, Matrices)
  - Types (int, double, single)
  - Operators
  - Basic Plotting
  - Scripts

- Week 2 – March 9
  - Plotting (continued)
  - Control flow (if_else, for, while, loops)
Course Information – Syllabus

- **Week 3 – March 16 March 23**
  - I/O (from files, images, loading/saving variables)
  - User input
  - Advanced data structures (cell, struct)
  - Debugging
  - Functions

- **Week 4 – March 30**
  - Figures
  - Images
  - Videos
Course Information – Syllabus

- Week 5 – April 6
  - Math and Linear Algebra
  - Solving Equations, basic statistics

- Week 6 – April 13
  - Final Useful things
  - Object Oriented Programming
  - GUI
  - Simulink & other Toolboxes
Course Information – Grading

- 5 Homeworks (15%, 15%, 15% , 15% , 15%)
- 1 Midterm Quiz (25%)  In class March. 30
Technical Details

- Download Xming and Putty (for Windows)
  - [http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html](http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html)
Technical Details

- Launch Xming
- Open a session in putty with Host Name
  - cunix.cc.columbia.edu
Make sure the X11 option of the SSH category is enabled
Technical Details

- Enter your cunix credentials

- Type
  - $ matlab &
What is MATLAB?

- Programming Environment
- Calculator
- Programming Language
- The solution to all your problems
MATLAB® is a high-level language and interactive environment that enables you to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and Fortran.

http://www.mathworks.com/products/matlab/
What can you do with Matlab?

- Design
- Compute
- Visualize
What can you do with Matlab?

- Design
- Compute
- Visualize
What can you do with Matlab?

- Design
- Compute
- Visualize
What can you do with Matlab?

- Design
- Compute
- Visualize
MATLAB Interface

Command Window

```
>> a = [3 4 5 6 7];

>>
```

Command History

```
axis([0 14 0 700]);

-- 9/27/09  8:46 PM --

clc
-1/((3.89*0.3/4+1)^4)  
-1/((3.89*0.3/4+1)^4))
-1/((3.89*0.3/4+1)^4))
(150 - sqrt(300))*pi
```
MATLAB records in the workspace and command history everything you write in the command window, so:

- `clear variable`
  - deletes variable from memory (and workspace)

- `clear all`
  - deletes all variables from memory (and workspace)

- `clc`
  - cleans command window
MATLAB’s command window works like a Linux terminal

Some example commands:
  - `cd`
  - `mkdir, rmdir`
  - `ls`
  - ...
Some commands used to interact with MATLAB

- **what**
  - returns the MATLAB files (.m, .mat) in the current directory

- **who**
  - returns the variables in your workspace

- **whos**
  - returns the variables in the workspace with additional info (size, dimensions)
Meet your best friend…

- Start → Help
- Press ? in interface
- Type `doc name_function`

... what about `help name_function`?
MATLAB does not use explicit type initialization like other languages.

Just assign some value to a variable name, and MATLAB will automagically understand its type:

- int \( x \)
  - \( x = 3 \)
  - \( x = 'hello' \)

Most common types:

- double
- char

We can assign mathematical expressions to directly create a variable:

- \( x = (3 + 4)/2 \)

; operator prevents the variable to be printed in the command window:

- \( x = 3; \)

disp prevents ans = from being displayed:

- disp(x)
MATLAB does not use explicit type initialization like other languages.

Just assign some value to a variable name, and MATLAB will automagically understand its type:

- `x`  
- `x = 3` \textbf{double}  
- `x = 'hello'` \textbf{char}

Most common types:

- We can assign mathematical expressions to directly create a variable:
  - `x = (3 + 4)/2`

- `;` operator prevents the variable from being printed in the command window:
  - `x = 3;`

- `disp` prevents `ans` from being displayed:
  - `disp(x)`
Data Structures – Variables

- **Naming Conventions**
  - Letter case matters
    
    \[
    A = 2 \quad \text{and} \quad a = 4
    \]
    These are 2 different variables!

  - Avoid using functions names for variables
    
    Example:
    \[
    \sin = 2 \quad \text{and} \quad a = \sin(0.5)
    \]
    \sin cannot be used as a function any more!

- **Built-in Variables**
  - \(i\) and \(j\) indicate complex numbers
  - \(\pi = 3.1415926\ldots\)
  - \(\text{ans} = \) last unassigned value
  - \(\text{Inf}\) and \(\text{Inf} = \) positive and negative infinity
  - \(\text{NaN} = \) ‘Not a Number’
This is really what MATLAB is all about!

- **Row vectors**
  - \( r = [2 \ 3 \ 5 \ 7] \);
  - \( r = [2, \ 3, \ 5, \ 7] \);

- **Column vectors**
  - \( c = [2; \ 3; \ 5; \ 7] \);
  - \( c = [2 \ 3 \ 5 \ 7]' \);

Transpose operator

\[
\begin{bmatrix}
2 \\
3 \\
5 \\
7
\end{bmatrix}
\]

\[
\begin{bmatrix}
2 \\
3 \\
5 \\
7
\end{bmatrix}^T
\]
Data Structures – Vectors

- **Special Vectors Constructors**
  - `: operator`
    - \( x = 1:3:13; \)
      - [1x5]
        |   | 4 | 7 | 10 | 13 |
      - Spacing, default = 1
  - `linspace()`
    - \( x = \text{linspace}(0,10,100); \)
      - Creates a vector of 100 elements with values equally spaced between 0 and 10 (included)

- Equivalent notation with : operator?
Data Structures – Matrices

- **Explicit Definition**
  - \[ M = \begin{bmatrix} 2 & 4 \\ 3 & 6 \\ 8 & 12 \end{bmatrix}; \]

- **Concatenation of vectors**
  - \[ r_1 = \begin{bmatrix} 2 & 4 \end{bmatrix}; \]
  - \[ r_2 = \begin{bmatrix} 3 & 6 \end{bmatrix}; \]
  - \[ r_3 = \begin{bmatrix} 8 & 12 \end{bmatrix}; \]
  - \[ M = [r_1; r_2; r_3]; \]

- **Concatenation of vectors and matrices**
  - \[ r_1 = \begin{bmatrix} 2 & 4 \end{bmatrix}; \]
  - \[ m_1 = \begin{bmatrix} 3 & 6 \\ 8 & 12 \end{bmatrix}; \]
  - \[ M = [r_1; m_1]; \]

Dimensions and Type must coincide!
Some Predefined Matrix Creation Functions

- \( M = \text{zeros}(2,3); \)  [3x2] matrix of zeros
  - rows: 2, columns: 3

- \( M = \text{ones}(2,3); \)  [3x2] matrix of ones

- \( M = \text{eye}(2); \)  [2x2] identity matrix

- \( M = \text{rand}(2,3); \)  [2x3] matrix of uniformly distributed random numbers in range [0,1]

- \( M = \text{randn}(2,3) \)  [2x3] matrix of normally distributed random numbers (mean 0, std dev. 1)
## Data Structures – Matrices

- Replicating and concatenating matrices

**repmat**
- \( X = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \)
- \( Y = \text{repmat}(X, 2, 4) \)

**vertcat**
- \( x1 = [2 \ 3 \ 4] \)
- \( x2 = [1 \ 2 \ 3] \)
- \( X = \text{vertcat}(x1, x2) \)

**horzcat**
- \( x1 = [2; 3; 4] \)
- \( x2 = [1; 2; 3] \)
- \( X = \text{horzcat}(x1, x2) \)
Getting the size of the matrix

- \( M = \begin{bmatrix} 2 & 3 & 4; & 3 & 4 & 55 \end{bmatrix} \);
- \([r \ c] = \text{size}(M)\);
- \( r = \text{size}(M,1) \);
- \( c = \text{size}(M,2) \);
- \( r = 2 \);
- \( c = 3 \);
Accessing Elements of Matrix M

- Matrix indexing starts with 1!

- **Explicit access**
  - `element = M(2,3);`
  - `element = M(5);`

- **:` operator**
  - `element = M(1,1:2);`
  - `element = M(:,1);`

- **`end` operator**
  - `element = M(1,2:end);`
Data Structures – Matrices

- Accessing Elements of Matrix M
  - Matrix indexing starts with 1!
  - Explicit access
    - `element = M(2,3);`
    - `element = M(5);`
  - `:` operator
    - `element = M(1,1:2);`
    - `element = M(:,1);`
  - `end` operator
    - `element = M(1,2:end);`

```
M
-1.2  -0.86   0.1
1.256  0.435 -1.33
```
Accessing Elements of Matrix M

Matrix indexing starts with 1!

- Explicit access
  - `element = M(2,3);`
  - `element = M(5);`

- `:` operator
  - `element = M(1,1:2);`
  - `element = M(:,1);`

- `end` operator
  - `element = M(1,2:end);`
Accessing Elements of Matrix M

- Matrix indexing starts with 1!

- Explicit access
  - `element = M(2,3);`
  - `element = M(5);`

- `:` operator
  - `element = M(1,1:2);`
  - `element = M(:,1);`

- `end` operator
  - `element = M(1,2:end);`

<table>
<thead>
<tr>
<th></th>
<th>-1.2</th>
<th>-0.86</th>
<th>0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>1.256</td>
<td>0.435</td>
<td>-1.33</td>
</tr>
</tbody>
</table>
Accessing Elements of Matrix M

Matrix indexing starts with 1!

- Explicit access
  - `element = M(2,3);`
  - `element = M(5);`

- `: operator`
  - `element = M(1,1:2);`
  - `element = M(:,1);`

- `end` operator
  - `element = M(1,2:end);`

```
M
[-1.2, -0.86, 0.1]
[1.256, 0.435, -1.33]
```
Accessing Elements of Matrix M

Matrix indexing starts with 1!

- Explicit access
  - element = M(2,3);
  - element = M(5);

- : operator
  - element = M(1,1:2);
  - element = M(:,1);

- end operator
  - element = M(1,2:end);
# Types

<table>
<thead>
<tr>
<th>Type name</th>
<th>bits</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>64</td>
<td>x = 32</td>
</tr>
<tr>
<td>char</td>
<td>16</td>
<td>x = ‘as’</td>
</tr>
<tr>
<td>(u)int8</td>
<td>8</td>
<td>x = (u)int8(32)</td>
</tr>
<tr>
<td>(u)int16</td>
<td>16</td>
<td>x = (u)int16(32)</td>
</tr>
<tr>
<td>(u)int32</td>
<td>32</td>
<td>x = (u)int32(32)</td>
</tr>
<tr>
<td>(u)int64</td>
<td>64</td>
<td>x = (u)int64(32)</td>
</tr>
<tr>
<td>single float</td>
<td>32</td>
<td>x = single(32)</td>
</tr>
<tr>
<td>complex</td>
<td>128</td>
<td>x = complex(2,1)</td>
</tr>
<tr>
<td>logical</td>
<td>1</td>
<td>x = true, x = logical([1 0 1])</td>
</tr>
</tbody>
</table>

- Note on complex numbers:
  - $x = 3 + 4j$;
  - $x = \text{complex}(3,4)$;
Operators

- **Basic Mathematical Operators**
  - + - * / \ ^

- **Some more complex mathematical functions**
  - sqrt()
  - log(), exp()
  - sin(), cos(), tan(), atan()
  - abs(), angle()
  - round(), floor(), ceil()
  - conj(), imag(), real()
  - sign()

- **Logical Operators**
  - & | ~

- **Relational Operators**
  - > < >= <= == ~=
Operators on matrices

- \( X = \begin{bmatrix} 2 & 3 & 4; \\ 5 & 4 & 6 \end{bmatrix} \);
- \( Y = \begin{bmatrix} 1 & 2 & 3; \\ 3 & 3 & 3 \end{bmatrix} \);

- \( Rplus = X + Y; \)
- \( Rminus = X - Y; \)

- \( Rmult = X \times Y; \)  
  **Error using \( ===> \) mtimes**  
  Inner matrix dimensions must agree.
- \( X2 = X'; \)
- \( Rmult = X2 \ast Y; \)

- \( Rpoint\_mult = X \ast \ast Y; \)

Some operators, like + and −, are always element wise!
Other operators, like * and /, must be disambiguated with . !
Operators on matrices

- \( R = X^2 \)
- \( X2 = \begin{bmatrix} 1 & 2 & 3; & 3 & 4 & 5; & 1 & 1 & 1 \end{bmatrix}; \)
- \( R_{\text{square}} = X2^2; \)

\[
R_{\text{square}} = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 10 & 13 & 16 \\ 20 & 27 & 34 \\ 5 & 7 & 9 \end{bmatrix}
\]

- \( R_{\text{dot}} = X.^2 \)

\[
R_{\text{dot}} = \begin{bmatrix} 4 & 9 & 16 \\ 25 & 16 & 36 \end{bmatrix}
\]
Special Functions for Matrices

° **sum()**, **prod()**
  - SumCols = \( \text{sum}(X) \);
  - SumRows = \( \text{sum}(X,2) \);
  - SumTot = \( \text{sum}(\text{sum}(X)) \);
  
  \[
  \begin{array}{c}
  \text{SumCols} \\
  \text{SumRows} \\
  \text{SumTot} = 21
  \end{array}
  \]

° **mean()**
  - MeanCols = \( \text{mean}(X) \);
  - MeanRows = \( \text{mean}(X,2) \);
  - MeanTot = \( \text{mean}(\text{mean}(X)) \);
  
  \[
  \begin{array}{c}
  \text{MeanCols} \\
  \text{MeanRows} \\
  \text{MeanTot} = 3.5
  \end{array}
  \]

° **max()**, **min()**
  - MaxVal = \( \text{max}(\text{max}(X)) \);
  - minCols = \( \text{min}(X) \);
  - minRows = \( [\text{min}(X(1,:)); \text{min}(X(2,:))]; \)
  - minRows2 = \( \text{min}(X,2) \equiv \text{min}(X,2*\text{ones(size}(X))) \)
  
  \[
  \begin{array}{c}
  \text{MaxVal} = 6 \\
  \text{minCols} \\
  \text{minRows} \\
  \text{minRows2}
  \end{array}
  \]
Operators

Special Functions for Matrices

- **max()**, **min()** – continued
  - \([\text{maxVal} \ \text{maxLoc}] = \text{max}(X);\) \hspace{1cm} \text{maxVal} = 13, \text{maxLoc} = 3
  
  MATLAB also tells us the location of the maximum value!

- **sort()** – orders the elements of a vector in ascending (default) or descending order
  - \(\text{xAsc} = \text{sort}(X);\)
  - \([\text{xDes} \ \text{order}] = \text{sort}(X, \text{’descend’});\)

- **find()**
  - \(\text{R} = \text{find}(X > 4);\) \hspace{1cm} \text{R} = 3 5 6
  - \(\text{R} = \text{find}(X == 13);\) \hspace{1cm} \text{R} = 3

\[
\begin{bmatrix}
1 & 2 & 13 \\
4 & 5 & 6
\end{bmatrix}
\]

- \(\text{R} = \text{find}(X >= 2 \ & \ X < 6)’;\)
  - \(\text{R} = \text{find}(X == 6);\)
  - \([r \ c] = \text{find}(X == 6);\)
  - \(r=2 \ \text{c}=3\)
Matrix indexing

- If we want to define the position of element 1 within the matrix M, we can do it with a single index or with the indexes of row and column
  - \( M = \begin{bmatrix} 2 & 4; 3 & 6; 5 & 1; 8 & 12 \end{bmatrix}; \)
  - \( \text{index} = \text{find}(M==1); \)

- \textbf{ind2sub}
  - \([r \ c] = \text{ind2sub}(\text{size}(M),\text{index});\)

- \textbf{sub2ind}
  - \( \text{newIndex} = \text{sub2ind}(\text{size}(M),r,c); \)
Matrix indexing

- If we want to define the position of element 1 within the matrix M, we can do it with a single index or with the indexes of row and column
  
  \[ M = \begin{bmatrix} 2 & 4 \\ 3 & 6 \\ 5 & 1 \\ 8 & 12 \end{bmatrix}; \]

  \[ \text{index} = \text{find}(M==1); \]

- **ind2sub**
  
  \[ [r \ c] = \text{ind2sub}(\text{size}(M), \text{index}); \]

  \[ 3 \ 2 \]

- **sub2ind**
  
  \[ \text{newIndex} = \text{sub2ind}(\text{size}(M), r, c); \]

  \[ 7 \]

It’s necessary to provide the size of the matrix!
Basic Plotting

- `plot()`
  - `x = [-1:0.1:1];`
  - `y = x.^2;`
  - `plot(y);`
  - `plot(x,y);`
Basic Plotting

- `plot()`
  - `x = [-1:0.1:1];`
  - `y = x.^2;`
  - `plot(y);`
  - `plot(x,y);`

- `plot(x,y,'--rd','LineWidth',2,...
  'MarkerEdgeColor','b',....
  'MarkerFaceColor','g',....
  'MarkerSize',10);`
Basic Plotting

- `plot()`
  - `x = [-1:0.1:1];`
  - `y = x.^2;`
  - `plot(y);`
  - `plot(x,y);`

- `plot(x,y,'--rd','LineWidth',2,...
  'MarkerEdgeColor','b',...
  'MarkerFaceColor','g',...
  'MarkerSize',10);`

- **Line style**  ```--```
- **Line color**  ```'red'```
- **Marker Type**  ```'diamond'```
Basic Plotting

- **bar()**
  - \( x = 100 \times \text{rand}(1,20); \)
  - \( \text{bar}(x); \)
  - \( \text{xlabel}('x'); \)
  - \( \text{ylabel}('values'); \)
  - \( \text{axis}([0 21 0 120]); \)
  - \( \text{xlim([0 21])}; \text{ylim([0 120])}; \)

- **pie()**
  - \( x = 100 \times \text{rand}(1,5); \)
  - \( \text{pie}(x); \)
  - \( \text{title}('My first pie!'); \)
  - \( \text{legend}('val1','val2',... 'val3','val4','val5'); \)
Basic Plotting

- **figure**
  - To open a new Figure and avoid overwriting plots
  - \( x = [-\pi:0.1:\pi]; \)
  - \( y = \sin(x); \)
  - \( z = \cos(x); \)
  - \( \text{plot}(x,y); \)
  - \( \text{figure} \)
  - \( \text{plot}(x,z); \)

- Close figures
  - \( \text{close 1} \)
  - \( \text{close all} \)

- Multiple plots in same Graph
  - \( \text{plot}(x,y); \)
  - \( \text{hold on} \)
  - \( \text{plot}(x,z,'r'); \)
  - \( \text{hold off} \)
Multiple plots in same Figure

- `figure(1)`
- `subplot(2,2,1)`
- `plot(x,y);`  
  `title('sin(x)');`
- `subplot(2,2,2)`
  `plot(x,z,'r');`  
  `title('exp(-x)');`
- `subplot(2,2,3)`
  `bar(x);`  
  `title('bar(x)');`
- `subplot(2,2,4)`
  `pie(x);`  
  `title('pie(x)');`
Scripts

- Like a notebook, but for code!
- M-files are MATLAB specific script files, they are called `namefile.m`
- You can open scripts from command window too, just type `open scriptname`
Adding comments to your code is a very healthy habit

Think about other people who have to read and understand 3000 lines of your code!

MATLAB comments, the `%` operator

- `x = [1 2 3 4];`
- `% this is a comment`
- `bar(x);`
- `title('bar(x)');`

When you type `help namefunction` in the command window, what you get is the comments on top of the `namefunction.m` script
Due at beginning of class, no exceptions

Put your code (.m files) and additional files in a single folder, name it `youruni_hw_X` and zip it

Upload the zipped folder to CourseWorks

Bring a printout of your code to class

Good luck and have fun !!!
MATLAB is also a philosopher!

Try typing why in the command window... you’ll get the answers!!!