Introduction to MATLAB

CS534 Fall 2016
Contact

Qisi Wang

Office: 1308CS
E-mail: qisi.wang@wisc.edu
Office hours:
  Tuesdays and Thursdays 11:45 a.m. - 12:45 p.m.
  and by appointment
What you'll be learning

- MATLAB basics (IDE, debugging)
- Operators
- Matrix
- Image I/O
- Image display, plotting
- A lot of demos
- ...

Accessing MATLAB

- Get a local copy from the [Campus Software Library](#).
- Available in the [Linux and Windows labs](#).
- Remotely accessible via ssh (instruction on [CSL website](#)).
  - Use your cs logins to login to instructional labs and ssh.
  - Note: On Linux, type `matlab` into the terminal.
Demo
MATLAB IDE
Introduction to the IDE

You can always change the layout here

Current Path

COMMAND WINDOW
Where you type commands

Workspace
List of your current variables

Filespace

Command History
List of previous commands
Your first MATLAB command

- Arithmetic operators: + - * /
- Assignment operator: =

```
% total = 1 + 1;

1. MATLAB computes what's 1 + 1
2. Value from (1) is assigned to this variable
3. Semicolon suppresses output

% This won't get executed

% marks for comments in matlab
Demo
Variable Types
Basic Types

- Numerics (double, int)
  - Numeric values are doubles by default
  - ex.
    - \( \text{var}_d = 1 \)
    - \( \text{var}_i = \text{uint8}(1) \)
  - other types exist: \text{uint16, int32, single}, etc

- Operations
  - +, -, *, /, ^, mod()
Basic Types

- Logical (boolean)
  - Can only be true or false
  - Mostly as the result of comparison operators
    - ==, <, >, <=, >=, !=
  - Support logical operations
    - ~, &&, ||
Demo
Basic Types

● Text (string, char)
  ○ Only ‘’ can be used to define strings/chars in Matlab
  ○ String are represented as char array
    ■ Strings can be indexed
      • str = ‘abc’; ← Note that the index starts from 1 in Matlab
      chr = str(1)
    ■ Strings can be concatenated
      • str1 = ‘Hello ’;
      str2 = ‘World’;
      hello = [str1, str2];
  ■ Some useful functions
    • str2num(), num2str(), strcmp()
Demo
Control Flow
if/elseif/else

if (boolean)
    ...
    ...
elseif (boolean)
    ...
    ...
else
    ...
    ...
end

Notice elseif is one word
while-loop

while expression
    statement
end

A = 0;
while A < 5
    disp(A);
    A = A + 1;
end
**for-loop**

```python
for i = values % values can be any array
    statements
end
```

- **Note:** for-"condition" overwrites changes to `i` within the loop!
for-loop

- \( i \) is assigned the value from the array directly. No need for indexing variables to iterate through the array.

```
% instead of
for i = 1:length(A)
    disp(A(i));
end
```
```matlab
for A = [1,1,2,3,5,8]
    disp(A);
end
```
**end keyword**

- Instead of using brackets for marking end of blocks, MATLAB uses "end"

```c
for(int a=0;a<=10;a++)
{
    if( a>3){
        ...
        ...
    }
}
```

```matlab
for (a=0:10)
    if (a>3)
        ...
        ...
    end
end
```

- In Command window, the control flow code block won’t be executed until an `end` was entered
Demo
MATLAB Files
There's two types of .m files

- **Scripts**
  - Contain a list of commands
  - Can be named anything
  - Usually for try things out (and you don’t want to type the same set of command again and again in the command line)

  Evaluate selection is a useful trick when you want to run part of

  ![Evaluate Selection](image)

  - Often used as drivers for functions you have implemented (Kind of like main in other languages)
There are two types of .m files

- Functions
  - If you want to run some command from other .m files with different parameters
  - Contain a function definition
  - FILE NAME MUST MATCH FUNCTION NAME

- Structure of a MATLAB function

  ```matlab
  function returnVar = FunctionName(input1,input2)
  %Adds two numbers
  returnVar = input1+input2;
  end
  ```

  - Start with function keyword
  - Return variable
  - Mark the end of the function. No return statement.
  - Return value is passed out by assigning to return variable(s)
  - Function name (Must match file name)
Writing MATLAB functions

- Functions Can Return Multiple values

```matlab
function [return1, return2] = FunctionName (input1,input2)
    return1 = input1+input2;
    return2 = 0;
end
```

- But you can suppress some of the returning value with ~

```matlab
return1 = FunctionName(input1, input2)
alternative
[return1, ~] = FunctionName (input1,input2)
```

  A place holder. return2 value not used

- Only the first function in the file can be called from other .m files

  You can have helper functions but they are not visible outside of the defining .m file
  The order you define helper functions doesn’t matter (unlike c++ or javascript)
Demo
Matrices/Arrays are effectively passed into functions "by value"

vector = [6 3 2 5 4 1];
disp(vector) % (1)
sort(vector);
disp(vector) % same output as (1)
Matrices are effectively passed into functions "by value"

```matlab
%my_corrected_script.m
vector = [6 3 2 5 4 1];
vector = sort(vector);
```
Demo
Debugging
Before you enter debugging mode

Click this to run program. Program pauses at checkpoints, if there's any.

Click along this column to set/remove breakpoints

Check this option for the program to pause once an error occurs

Error in **untitled** (line 1)
sdfsdf
Debugging mode works like any other IDE
Demo
Matlab Resources

- Matlab documentation
- Help command
- google
- Matlab resources form course webpage
Matrices
What is a matrix?

Terms: row, column, element, dimension
How are the dimensions arranged

First dimension

\[
\begin{pmatrix}
1 & 2 & 3 \\
4 & 5 & 6
\end{pmatrix}
\]

Second dimension

MxNxP matrix
Defining a matrix with literals

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

```
1  2  3
4  5  6
```
Defining a equally spaced vector

```
>> A = 1 : 5
A =
    1  2  3  4  5

>> A = 1 : 2 : 10
A =
    1  3  5  7  9
```

Colon creates regularly spaced vectors

Bonus: what if I have something impossible like
A = -1 : 2 :-5
Define matrix with built-in functions

- `zeros(M,N)`
- `ones(M,N)`
- `true(M,N)`
- `false(M,N)`
  - Create matrices with all 0/1/true/false’s
  - M, N are number of rows and cols respectively
  - can have more dims
- `linespace(start, end, number)`
  - Create linearly spaced vector ranging from start to end (inclusive)
  - number specifies the length of the vector

Bonus: How do you get a matrix of all 5?
Demo
Matrix Operations
size()

>> A = [1 2 3; 4 5 6];
>> size(A, 1)
ans =
    2
>> size(A, 2)
ans =
    3

asks for first dimension

asks for second dimension
size() cont'd

>> A = [1 2 3; 4 5 6];
>> [height, width] = size(A)
height =
    2
width =
    3
Demo
Concatenation

- \( M = [A, B; C, D] \)

Dimension must match; mark the next row

\[
\begin{bmatrix}
  A & B \\
  \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} & \begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix} \\
  C & D \\
  \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 \end{bmatrix} & \begin{bmatrix} 1 & 2 \end{bmatrix}
\end{bmatrix}
\]
Concatenation in higher dims

- \texttt{cat(A, B, n)}

Operand matrices \quad Dimension to work on

The length of dimensions other than \( n \) of \( A \) and \( B \) must match
Demo
Linear Algebraic Operations

- **+** Addition (dimensions match exactly)
- **−** Subtraction (dimensions match exactly)
- **∗** Matrix Multiplication (MxN-matrix * NxP-matrix)
- **^** Matrix Power (must be square matrix)
- **'** Transpose
- **\** Left Matrix Division (Solves A*x=B)
- **/** Right Matrix Division (Solves x*A=B)
How Operations Work

\[
A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad B = \begin{bmatrix} 3 & 1 \\ 5 & 6 \end{bmatrix}
\]

\[
A + B = \begin{bmatrix} 4 & 3 \\ 8 & 10 \end{bmatrix} \quad A - B = \begin{bmatrix} -2 & 1 \\ -2 & -2 \end{bmatrix}
\]

\[
A \times B = \begin{bmatrix} 13 & 13 \\ 29 & 27 \end{bmatrix} \quad A^2 = \begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}
\]

\[
A \backslash B = \begin{bmatrix} -1 & 4 \\ 2 & 1.5 \end{bmatrix} \text{ solves } A \times x = B \quad B \backslash A = \begin{bmatrix} -0.3077 & 0.3846 \\ 0.1538 & 0.6923 \end{bmatrix} \text{ solves } x \times A = B
\]
Transpose

\[
C = \begin{pmatrix}
1 & 3 \\
5 & 7 \\
9 & 11 \\
13 & 15
\end{pmatrix}
\]

\[
C' = \begin{pmatrix}
1 & 5 & 9 & 13 \\
3 & 7 & 11 & 15
\end{pmatrix}
\]
Elementwise Operations

- dimensions need to match exactly
- usually use . to distinguish from their linear-algebraic counterparts

- + Addition
- - Subtraction
- .* Element by Element Multiplication
- ./ Element by Element Division
- .^ Element by Element Power
  - A.^2 vs. A^2 vs. A.^B
Element-wise operations

\[ A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad B = \begin{bmatrix} 3 & 1 \\ 5 & 6 \end{bmatrix} \]

Note the 2 operand matrix for element-wise operations must match

\[ A ./ B = \begin{bmatrix} 0.333 & 2 \\ 0.6 & 0.666 \end{bmatrix} \]

\[ A .* B = \begin{bmatrix} 3 & 2 \\ 15 & 24 \end{bmatrix} \]

\[ A .^ B = \begin{bmatrix} 1 & 2 \\ 243 & 4096 \end{bmatrix} \]

\[ A .^ 2 = \begin{bmatrix} 1 & 4 \\ 9 & 16 \end{bmatrix} \]
Demo
Logical operators

- `==` is equal to
- `< > <= >=` less/greater than
- `~` not
- `~=` not equal to
- `&` elementwise logical AND (for matrices)
- `|` elementwise OR (for matrices)
- `~` negation

To be distinguished from

- `&&` short-circuit AND (for logical expressions)
- `||` short-circuit OR (for logical expressions)
Two useful commands

- `all()`
- `any()`
  - both work along one dimension of the matrix
  - by default compare along first dimension
  - use an optional second parameter to specify the dimension to work on
  - help to shrink a logical matrix to a logical scalar
  - then you can use `||` or `&&`
Demo
fin.