Introduction to MATLAB

csc401/2511 - Natural Language Computing Tutorial, Winter 2016
My name = Hamed Heydari
My email = h.heydari@mail.utoronto.ca

Based on slides by Frank Rudzicz
Agenda

- Matlab Basics
- Matrices and vectors
- Displaying results
- Cell arrays and structures
- Miscellaneous topic
MATrix LABoratory

- Designed for matrix computations.
- High-level language
  - You don't see the details
  - Dynamic typing
- Basic Data type
  - 2- or 3-D double-precision matrix
- Most functions work on entire matrices.
MATrix LABoratory

Comprehensive and optimized toolboxes and libraries

- e.g., statistics and neural networks
Useful for prototyping and scientific work, but non-free and not arbitrarily flexible.

Available on cdf: greywolf:~% matlab

To use version with graphics from your own computer: ssh -X yourcdfaccount
Getting Help


Within matlab (>>help function-name):

```
>> help log
. LOG    Natural logarithm.
.   LOG(X) is the natural logarithm of the
.     elements of X.
.   Complex results are produced if X is not
.     positive.

.   See also log1p, log2, log10, exp, logm,
.     reallog.
```
Generate a “magic square” with equal row, column, and diagonal sums.
Assign the resulting matrix to variable A.

```
>> A = magic(3);
>> A = magic(3)
A =
     8     1     6
     3     5     7
     4     9     2
```

No semi-colon, so its result is shown. Useful for debugging, but NOT to be handed in.
Matrices in Matlab

Colon generates number sequence:

```matlab
>> 11:14
ans =
    11    12    13    14
```

```matlab
>> -1:1
ans =
    1     0     1
```

```matlab
>> 3:0
ans =
Empty matrix: 1-by-0
```

Second colon specifies step size:

```matlab
>> 1:3:12
ans =
    1     4     7    10
```

```matlab
>> 4:-1:1
ans =
    4     3     2     1
```

```matlab
>> 3:-0.5:2
ans =
    3.0000    2.5000    2.0000
```
Matrices in Matlab

>> A = magic(3)   A =
   8     1     6
   3     5     7
   4     9     2

Select submatrices

>> A(2,1)        ans =
                    3

>> A(1,:)        ans =
                        8     1     6

>> A(2:3,1:2)    ans =
                    3     5
                    4     9

>> A(1:5)        ans =
                        8     3     4     1     5

>> b=A(6:end)    ans =
                        9     6     7     2

>> size(b)       ans =
                        1     4
Matrices in Matlab

Build manually

Transpose of a matrix

zeros (1, 3)
Mask matrix elements

```matlab
>> find(A>5)
an =
    1
    6
    7
    8
>> A(find(A>5)) = 0
A =
    0    1    0
    3    5    0
    4    0    2
```

```matlab
>> A = magic(3)
A =
    8    1    6
    3    5    7
    4    9    2
```
Operators preceded by a period (.) perform the given operation in an *element-wise* fashion.
>> a = 5;
>> disp(a);
  5
>> a = [5 6 7];
>> disp(a);
  5 6 7
>> str = 'Array';
>> disp([str, ': ', num2str(a)]);
Array: 5 6 7

The disp command will be your primary way of reporting results. Leaving semicolon adds variable name, not always desirable.
Reporting in Matlab: plot

```matlab
>> x = 0:20;
>> y = 0.5 - 0.5*cos(2*pi*x/20);
>> stem(x,y);
>> title('20-point raised cosine');
```

```matlab
>> t = 0:0.1:10;
>> x = exp(t * (j - 1/3));
>> plot(t, real(x), t, imag(x));
>> grid;
>> legend('real', 'imaginary');
>> saveas(gcf, 'realImag.eps');
```
Good practice: vectorizing

Loops are slow in Matlab.

Matrix operations are very fast.

e.g., compute logarithms for elements of a matrix

BAD:

```matlab
P = randn(50,100);
for i=1:size(P,1)
    for j=1:size(P,2)
        P(i,j) = log( P(i,j) );
    end
end
```
Good practice: vectorizing

Loops are slow in Matlab.

Matrix operations are very fast.

e.g., compute logarithms for elements of a matrix

**BAD:**

\[
P = \text{randn}(50, 100);
\text{for } i = 1: \text{size}(P,1)
  \quad \text{for } j = 1: \text{size}(P,2)
  \quad \quad P(i,j) = \log(P(i,j));
  \quad \text{end}
\text{end}
\]

**GOOD:**

\[
P = \text{randn}(50, 100);
P = \log(P);
\]
Some equations are secretly vector operations in disguise.

e.g., compute

\[
\sum_{n=1}^{1000} n(1000-n)
\]

BAD:

```plaintext
ssum = 0;
for n=1:1000
  ssum = ssum + n*(1000-n);
end
```
Some equations are secretly vector operations in disguise.

e.g., compute

\[ \sum_{n=1}^{1000} n(1000-n) \]

**BAD:**

```matlab
ssum = 0;
for n=1:1000
    ssum = ssum + n*(1000-n);
end
```

**GOOD:**

```matlab
a = 1:1000;
b = 1000 - a;
ssum = a*b';
```
>> C = cell(2,3)
C =
    []     []     []
    []     []     []

>> C{1,2} = 'hello';
>> C{1,1} = eye(2);
>> C{2,2} = 42
C =
   [2x2 double]    'hello'     []
       []    [ 42]     []

>> C{1,1}
ans =
   1     0
   0     1

>> C(1,1)
ans =
   [2x2 double]
Matlab Miscellany: Structures

Can act similarly to C's struct or to a dictionary.

```matlab
>> s.code = 401;
>> s.name = 'natural language computing';
>> s
s =
    code: 401
    name: 'natural language computing'
>> isfield(s, 'code')
ans =
    1
>> f = fieldnames(s)
f =
    'code'
    'name'
>> s('code')
ans =
    401
>> s.(f{2})
ans =
    natural language computing

>> s(2).code=777
>> s(2).name='NLP'
>> s(2)
ans =
    code: 777
    name: 'NLP'
```
The `dir` function gives a structure describing a directory.

```matlab
>> D = dir('.*PHN')
D =
3x1 struct array with fields:
    name
    date
    bytes
    isdir
    datenum

>> D(1).name
ans =
SX127.PHN
Matlab Miscellany: Files

>> type SX127.PHN
0 2231 h#
2231 2834 dh
2834 3757 iy
...
>> [t1, t2, phone] = 
textread('SX127.PHN', '%d %d %s');
>> t1(2)
ans =
    2231
>> t2(2)
ans =
    2834
>> phone(2)
ans =
    'dh'

>> mfcc = dlmread('unkn_14.mfcc');
>> size(mfcc)
ans =
    279    14

The textread and dlmread functions quickly parse files.
Regular Expressions

Pattern Matching
>> t = regexp('This assignment is awesome!', '(\w*)is', 'tokens');
>> t{1}
ans =
   'Th'

Replace String
>> t = regexprep('This assignment is awesome!', '(\w*)is', 'at')
t =
at assignment at awesome!
Creating a Function and a File

myfun.m

%Comment: \([y_1,\ldots,y_N]\) are the outputs,
%\texttt{myfun} is the function name
%(x_1,\ldots,x_M) are the inputs

function [y_1,\ldots,y_N] = myfun(x_1,\ldots,x_M)
    y_1 = \text{prod}(x_1);
    \ldots
    y_N = \text{prod}(x_N);
end

It is good practice to name the matlab file the same name as the first (and usually the only) function in it

You may create more than one function within a file, but any function other than the first will only be accessible from within their own file

You can call 'myfun' from the matlab terminal or from within another file by doing, for example:
\[
\text{>> myfun(var1, \ldots, varM)}
\]
Matlab Cheatsheet

\[ *, ^ \]
\% matrix multiply, exponent

\[ /, \backslash, \text{inv} \]
\% A/B = AB-1, A\backslash B = A^{-1}B, A^{-1}

\[ +, -, \ast , \ast /, \ast ^ \]
\% element-wise operations

\[ =, \neq, <, >, \leq, \geq \]
\% relations

\[ \text{length, size} \]
\% size of vectors, matrices
\% (row-first order)

\[ \text{zeros, ones, eye, diag} \]
\% all-0, all-1, identity, diagonal matrices

\[ \text{xlabel, ylabel, zlabel} \]
\% label plot axes

\[ \text{subplot(i,j,k)} \]
\% plot to the k\textsuperscript{th} spot of a i x j array of plots.

\[ \text{sum, prod, min, max} \]
\% aggregate rows, columns

\[ \text{find} \]
\% list non-zero indices

\[ \text{figure, saveas} \]
\% open, save figures

\[ \text{whos} \]
\% list name, size, and type of all workspace variables