Matlab tutorial
Outline of Matlab Tutorial

• Data Types
  – arrays: char, numeric, struct, logical, cell, others

• Operators
  – arithmetic, relational, logical

• Flow Control
  – conditionals, case, while, etc.

• M-functions
  – syntax
  – examples of simple functions
  – writing and debugging a simple MATLAB function

• Graphics
Data Types in MATLAB

Array

Logical

Char
  ‘a’

Numeric

Uint8
  (8 bit, from 0 to 255, e.g., pixel values)

Double
  e.g., 3.2567
  (8 bytes)

Structure
  image.width = 120
  image.name = ‘face1’

Others

Others
I prefer the non-fancy command-line interface
Images

>> im = imread('arnold.jpg');

>> size(a)
ans =
   450 313 3

>> imshow(im);

imshow(im) works for color images (3D arrays) and grayscale images (2D matrices)
Uint8 and Doubles

- Double
  - almost all MATLAB functions
    - expect doubles as arguments
    - return doubles
**Uint8 and Doubles**

- **Double**
  - almost all MATLAB functions
    - expect doubles as arguments
    - return doubles
    - e.g.,

```matlab
>> a = 1:10
a =
    1     2     3     4     5     6     7     8     9    10

>> b = uint8(a)

>> whos
Name      Size         Bytes  Class
a         1x10            80  double array
b         1x10            10  uint8 array

>> b*500
255  255  255  255  255  255  255  255  255  255

>> double(b)*500
ans =
500  1000  1500  2000  2500   3000  3500 4000 4500 5000
```
## uint8 and Doubles

- **Double**
  - almost all MATLAB functions
    - expect doubles as arguments
    - return doubles
    - e.g.,

- need to convert uint8 to double before performing any math operations

```matlab
» a = 1:10
a =
1 2 3 4 5 6 7 8 9 10
» b = uint8(a)
b =
1 2 3 4 5 6 7 8 9 10
» whos
Name       Size         Bytes  Class
a          1x10         80  double array
b          1x10         10  uint8 array

» b*500
255 255 255 255 255 255 255 255 255 255

» double(b)*500
ans =
500 1000 1500 2000 2500 3000 3500 4000 4500 5000
```
Struct Data Type

\[ \text{im.index} = [4 \ 10 \ 3; \ 12 \ 12 \ 2; \ 10 \ 4 \ 3]; \]
Struct Data Type

```matlab
>> im.index = [8 10 2; 22 7 22; 2 4 7];
>> im.map = [0 0 0; 0 0 .1; 0 .1 0;.....]

» whos
   Name        Size         Bytes  Class
   im       1x1            590  struct

Grand total is 18 elements using 590 bytes

This is how gif images are stored internally
Arrays of Structures

```matlab
» im.index = [8 10 2; 22 7 22; 2 4 7];
» im.map = [0 0 0; 0 0 .1; 0 .1 0;.....]
» ims(1) = im;
» ims(2).index = [12 3 2; 23 3 3; 23 12 1];
» ims(2).map = ims(1).map;
» whos
    Name      Size         Bytes  Class          attributes
    -----------  -----------    ------  -----------
    ims    1x2            894  struct array

Grand total is 28 elements using 894 bytes

» ims

ims =

1x2 struct array with fields:
    index
    map
```
Array of structures (example)

Useful for enumerating all frames in a directory (eg, homeworks)
Plotting an image

- Use imshow.m
  >> help imshow

IMSHOW(I) displays the grayscale image I
IMSHOW(RGB) displays 3D array as color image
IMSHOW(BW) displays the binary image BW. IMSHOW display pixels with the value 0 (zero) as black and pixels with the value 1 as white.
IMSHOW(X,MAP) displays the indexed image X with the colormap MAP. X takes on values from 1 to “k”, and colormap is a matrix of size “k by 3”

- For matrices containing doubles, color intensities are on the range [0.0, 1.0]. For uint8 and uint16 matrices, color intensities are on the range [0, 255].
• “im2double(im)” converts image to double and scales values from 0 to 1. It will try to guess the correct format of the input image “im”.

• you can use “double(im)”, but then you need to explicitly scale to visualize

• “im2uint8(im)” converts image to uint8 values (0-255). It will try to guess the correct format of the input image “im”.
Arithmetic Operators

- Transpose, $a'$

- Power, $a^{^2}$

- Addition, multiplication, division
  - $a(1)*b(2)$
  - $a*b$
    - works if $a$ and $b$ are matrices with appropriate dimensions $(\text{columns}(a) = \text{rows}(b))$
  - $a.*b$ (element by element)

- except for matrix operations, most operands must be of the same size, unless one is a scalar
Which will work?

» a = [2 3];
» b = [4 5];
» a(1)*b(2)
   
   ans =
   
   10

» a*b

??? Error using ==> *
   Inner matrix dimensions must agree.

» a*b'

   ans =
   
   23

» a.*b

   ans =
   
   8    15

» b/2

   ans =
   
   2.0000    2.5000
Vectorized arithmetic operators

- Matlab functions can be applied on each element of an array

\[
\begin{align*}
\text{>> } a &= [1 \ 4 \ 5 \ 3]; \\
\text{>> } \sin(a)
\end{align*}
\]

- Arithmetic operations over sets of numbers can be vectorized

\[
\begin{align*}
\text{>> } a &= 1 + 2 + 3 + 4 + 5; \\
\text{>> } a &= \text{sum}(1:5);
\end{align*}
\]

- Such operations can be applied on each column of a 2D array (image)

\[
\begin{align*}
\text{>> } a &= [1 \ 2; \ 3 \ 4]; \\
\text{>> } \text{sum}(a)
\end{align*}
\]

- What about applying on each row?
Relational Operators

- `<`, `<=`, `>`, `>=`, `==`, `~=`

- compare corresponding elements of arrays with same dimensions

- if one is scalar, one is not, the scalar is compared with each element

- result is of type Logical
  - element by element 1 or 0
a = [2 3; 4 1]

b = [4 5; 2 2]

a > b

b > a

a > 2
Vectorization of Computation

tic
for i=1:100000
    y(i) = log(i);
end
toc
Vectorization of Computation

tic
for i=1:100000
    y(i) = log(i);
end
toc

elapsed_time = 168.78 seconds
Vectorization of Computation

tic
for i=1:100000
    y(i) = log(i);
end
toc

tic
i=1:100000;
z = log(i);
toc
Vectorization of Computation

```
tic
for i=1:100000
    y(i) = log(i);
end
toc

elapsed_time = 168.78 seconds```

```
tic
i=1:100000;
z = log(i);
toc

elapsed_time = 0.053 seconds```

First method calls the log function 100,000 times,
Second method only calls it once (much faster)
Subsets of matrices

```matlab
» b = [5 6 7 8];
» a = [1:4];
» c = [a;b];
» c(1,1)

» c(1,:)

» c(1:2,1:2)
```

How would you show the top half of an image?
Special of matrices

» ones(2,3)
ans =
1 1 1
1 1 1

» zeros(1,4)
ans =
0 0 0 0

» rand(3,3)
ans =
0.2176 0.4909 0.8985
0.4054 0.1294 0.5943
0.5699 0.5909 0.3020
Reshaping arrays

```matlab
» b = [1 2; 3 1];
» a = reshape(b,1,4);
» a

vectorized_ims = [a; a*2; a*3; a*4];
» size(vectorized_ims)
```

How to access the first image?
Row versus column major order?
Is there a way to construct `vectorized_ims` faster?
Saving your work

```
>> ls
>> whos
>> save work
>> clear
>> whos
>> load work
>> whos
```

Note: You can save and load variables in the workspace

Note: You can save the interactive commands in a file (e.g., script.m) and run the commands by typing “>> script”
Example of a MATLAB script

% script image_invert
% A simple script to invert an image
% Deva Ramanan, Jan 2015

Comment text: it is always important to put comments in your code. Comments at the top should clearly explain what the script or function does.
% script image_invert
% A simple script that reads in an image and inverts it
% Deva Ramanan, Jan 2015

im = imread('arnold.jpg');
[nrows,ncols] = size(im);
inv = zeros(nrows,ncols);

for i = 1:nrows,
    for j = 1:ncols,
        inv[i,j] = 1 - im[i,j];
    end
end

subplot(121);
imshow(im);

subplot(122);
imshow(inv);

Pre-allocation of variables always a good idea

What’s wrong with this?
Example of a MATLAB script

% script image_invert
% A simple script that reads in an image and inverts it
% Deva Ramanan, Jan 2015

im = imread('arnold.jpg');
im = im2double(im);
[nrows, ncols] = size(im);
inv = zeros(nrows, ncols);

for i = 1:nrows,
    for j = 1:ncols,
        for k = 1:3,
            inv[i, j, k] = 1 - im[i, j, k];
        end
    end
end

subplot(121);
imshow(im);
subplot(122);
imshow(inv);
Can we perform this faster?

```matlab
im = imread('arnold.jpg');
im = im2double(im);
inv = 1 - im;
```

This is the *real* benefit of matlab
Example of a MATLAB function

function [res] = invert_array(A);
Example of a MATLAB function

function [B] = invert_array(A);
% function [B] = invert_array(A);
% A function that inverts an array by subtracting
% each element from the maximum value in the array

% Deva Ramanan, Jan 2015

% INPUTS:
% A: array of any size

% OUTPUTS:
% B: inverted array

Compute maximum value
m = max(A(:));

% Subtract each element from max value
B = m - A;

Clear comments in function headers are very useful

Note the explicit statement explaining what
the inputs and outputs are (including their
dimensionality)