RWTH AACHEN UNIVERSITY

Computer Vision - Exercise 1

Introduction to MATLAB

Stefan Breuers
breuers@vision.rwth-aachen.de

RWTH Aachen University - Computer Vision Group
http://www.vision.rwth-aachen.de/

29th October, 2015

RWTH AACHEN UNIVERSITY

Outline

1. Obtain MATLAB
2. Basic usage
3. MATLAB is *different*
4. Speedup

RWTH AACHEN UNIVERSITY

Got MATLAB?

- Numerical Programming Environment
- `http://www.matlab.rwth-aachen.de/`
- CampusLicense
  - RWTH intern/VPN
  - “borrowing”
  - `matlab.internal.licensing.BorrowUI(true)`
- CIP Pool (E1 & E2 building)
- Use for exercise MATLAB 2014a +

Important Commands

help	guess. . .
doc	Open help browser
lookfor	Search for keyword
clear	Clears variable(s)
close	Closes current/all figures
clc	Clear command window
whos	List variables in current workspace
save	Save the current workspace
load	Load a saved workspace
keyboard	Enter debugging mode (until **dbquit**)

RWTH AACHEN UNIVERSITY

Useful things you should remember

- Index starts at 1 not 0
- Comments start with % not # or //
- Continue long lines with ...
- Semicolon ; suppresses output (no error)

Basic Operations

```
% Scalars
L = 2;
C = 3;

% basic operators
sum_ = L + C;
prod_ = L * C;

% functions
T = tan(L/C);
E = exp(L-C);
```

```
% For loop
sum_ = 0
for i = 1:100
  sum_ = sum_ + i;
end

% If statement
number = 13;
if isprime(number)
  disp('prime number');
else if odd(number)
  disp('odd number');
else
  disp('none of the above');
end
```

Everything is a Matrix

```
% Line vector
lv = [1 2 3];
lv = [1,2,3];
lv = 1:3; % from 1 to 3
lv = 1:1:3 % step size 1
lv = linspace(1, 3, 3);

% Column vector
cv = [1;2];
cv = (1:2)'; % transpose
size(cv)
ans =
   2   1
lv =
   1   2   3
cv =
   1
   2
```

```
% Different ways of
    defining the same
    matrix:
M = [1 2 3; 4 5 6];

L = 2;
C = 3;
M = zeros(2,3);
for l=1:L
  for c=1:C
    M(l,c) = ((l-1)*3)+c;
  end
end

M = reshape(1:L*C, C, L)';
M =
   1   2   3
   4   5   6
```

RWTH AACHEN UNIVERSITY

Accessing Elements

```
M = [1 2 3; 4 5 6]
M =
   1   2   3
   4   5   6

% M(line,column)
M(1,3)
ans =
   3
M(1,end)
ans =
   3
M(5)
ans =
   3
M(1,end:-1:1)
ans =
   3   2   1
M(1,1:3)
ans =
   1   2   3
M(1,1:end)
ans =
   1   2   3
M(1,:)
ans =
   1   2   3
M(1:2,1:2)
ans =
   1   2
   4   5
M(M > 4)
ans =
   5
   6
```

Manipulating Matrices

```
A = [1 4;0 3]
A =
   1   4
   0   3
B = [1 0;0 -1]
B =
   1    0
   0   -1
% Matrix multiplication
A*B
ans =
   1   -4
   0   -3
% Elementwise
A.*B
ans =
   1    0
   0   -3
% Concatenation
C = [A B]
C =
   1   4   1    0
   0   3   0   -1
C = [A;B]
C =
   1    4
   0    3
   1    0
   0   -1
C = repmat(A, 1, 2)
ans =
   1   4   1   4
   0   3   0   3
```

Manipulating Matrices (cont.)

```
A = [1 4;0 3]
A =
   1   4
   0   3
B = [1 0;0 -1]
B =
   1    0
   0   -1
A/B % same as A*inv(B)
ans =
   1   -4
   0   -3
% Elementwise division
% 1/0 = Inf, 0/0=NaN
R = A./B
R =
     1   Inf
   NaN    -3
```

```
isinf(R)
ans =
   0   1
   0   0
isnan(R)
ans =
   0   0
   1   0
% Replace NaN with 0
R(isnan(R))=0
R =
   1   Inf
   0    -3
% Find non-zero elements
find(R)
ans =
   1
   3
   4
```

Try to code in matrix ways

```
A = [1 2;3 4];
B = [1 1;2 2];

% With for loops
S = zeros(2);
for l = 1:2
  for c = 1:2
    S(l, c) = A(l, c)
        + B(l, c);
  end
end

% Better use matrix
C = A + B;
C =
  2   3
  5   6
```

```
% Matlab functions usually
    work on matrices, not
    only on scalars, for
    example:
C = A^2;
D = sqrt(B);
% Be careful which functions
     operate elementwise, on
    a line/column or on the
    whole matrix:
E = sum(A) % columnwise
E =
   4   6
E = sum(sum(A))
E =
   10
E = sum(A(:))
E =
   10
```

Scripts and Functions

```
function p = pressure(V, n, T)
  R = 8.314462175 % gas constant, unit = J/(mol*K)
  p = n*R*T/V
end
```

- `*.m` files
- Can accept arguments
- Name of function **must** be the same as the filename (except local function)
- Execution by calling name of function

```
p = pressure(42, 18, 288.15)
p =
  1.026776689596964e+03
```

RWTH AACHEN UNIVERSITY

Image-specific functions

- `imshow` - Display an image
  - `imshow(img);`
  - `imshow(img, [low, high]);`
- **`image`** - Display a matrix as image
  - **`image`**`(mat);`
  - Elements of mat are indices to colormap
- **`imagesc`** - Display a matrix as scaled image
  - **`imagesc`**`(mat);`
  - scaling: full colormap is used
- **`colormap`** - Define the colormap to use

```
1 map = colormap;
2 reversed_map = map(end:-1:1, :);
3 colormap(reversed_map);
```

Some Nuisances

- Type issues
  - `imread` uses `uint8`
  - Use `im2double` to convert to double matrix
- Colormap issues
  - **`colormap`** works differently for different image formats!
- Functions can depend on return value!

```
m = min(abs(5-linspace(0, 10, 21)))
m =
   0
[m,am] = min(abs(5-linspace(0, 10, 21)))
m =
   0
am =
   11
```

Other Visualization Functions

- **figure** - Open a new window or select existing one

```
figure;
figure(1);
handle = figure;
figure(handle);
```

- **plot** - Plots 2D graph

```
plot(y);
plot(x, y);
plot(x, y, 'b.-');
plot(x1, y1, 'b.-', x2, y2, 'ro:');

figure;
hold on;
plot(x1, y1, 'b.-');
plot(x2, y2, 'ro:');
hold off;
```

Other Visualization Functions (cont.)

- Variations on plots
  - **plot3** - Plots 3D graphs
  - plotyy - 2D graph with two y-axes
  - **semilogx**, **semilogy**, **loglog** - logarithmic axes
- **bar** - Bar graphs
  - `bar(x, y);`
- scatter - Display a scatter plot
  - `scatter(x, y);`
  - `scatter(x, y, s, c);`

Advanced Functions

- Matrix operations
  - `inv(M)` - Create inverse of matrix
  - `ones(m, n)` - Create an $m \times n$ matrix with values 1
  - `zeros(m, n)` - Create an $m \times n$ matrix with values 0
  - `rand(m, n)` - Create an $m \times n$ matrix with values sampled uniformly from $[0, 1)$
  - `det(M)`, `eye(n)`, `norm(x)`, `rank(M)`, ...
- Common statistical functions
  - `mean(x)` - Arithmetic mean
  - `var(x)` - Variance
  - `hist(x)` - Create/plot a histogram of x
- Other useful functions
  - `[V,D] = eig(A)` - Eigenvalues and eigenvectors
  - Pre-defined filters: gaussian, laplacian, sobel, ...

MATLAB is different - Find the Mistake (1)

```
1 A = zeros(N, 1);
2 for n = 0:N-1
3   A(n) = n
4 end
```

MATLAB is different - Find the Mistake (1)

```
1 A = zeros(N, 1);
2 for n = 0:N-1
3   A(n) = n
4 end
```

Subscript indices must either be real positive integers or logicals.

MATLAB is different - Find the Mistake (1)

```
1 A = zeros(N, 1);
2 for n = 0:N-1
3   A(n) = n
4 end
```

Subscript indices must either be real positive integers or logicals.

```
1 A = zeros(N, 1);
2 for n = 1:N
3   A(n) = n
4 end
```

MATLAB is different - Find the Mistake (2)

```
A = 0;
for n = 1:N
  A(n) = n
end
```

MATLAB is different - Find the Mistake (2)

```
1 A = 0;
2 for n = 1:N
3   A(n) = n
4 end
```

Trick question! This **actually works**!

MATLAB is different - Find the Mistake (2)

```
1 A = 0;
2 for n = 1:N
3   A(n) = n
4 end
```

Trick question! This **actually works**!

```
1 A = zeros(N, 1);
2 for n = 1:N
3   A(n) = n
4 end
```

Memory will not be allocated in every iteration!

MATLAB is different - Find the Mistake (3)

```
% #students in lectures
students = [20;40;20];
% #teachers in lectures
teachers = [1;4;2];
% #students per teacher
ratio = students/teachers
```

MATLAB is different - Find the Mistake (3)

```
% #students in lectures
students = [20;40;20];
% #teachers in lectures
teachers = [1;4;2];
% #students per teacher
ratio = students/teachers
```

This is probably **not** what you want!

```
ratio =
   0    5   0
   0   10   0
   0    5   0
```

`B/A` solves $xA = B$.

`A\B` solves $Ax = B$.

MATLAB is different - Find the Mistake (3)

```
% #students in lectures
students = [20;40;20];
% #teachers in lectures
teachers = [1;4;2];
% #students per teacher
ratio = students/teachers
```

This is probably **not** what you want!

```
ratio =
   0    5   0
   0   10   0
   0    5   0
```

`B/A` solves $xA = B$.

`A\B` solves $Ax = B$.

```
% #students in lectures
students = [20;40;20];
% #teachers in lectures
teachers = [1;4;2];
% #students per teacher
ratio = students./teachers
```

You have to use `./` for elementwise matrix operations!

MATLAB is different - Find the Mistake (4)

```
figure;
imshow(img);
% mark detection
plot(box(1,:), box(2,:));
```

MATLAB is different - Find the Mistake (4)

```
1 figure;
2 imshow(img);
3 % mark detection
4 plot(box(1,:), box(2,:));
```

The image is gone!

MATLAB is different - Find the Mistake (4)

```
1 figure;
2 imshow(img);
3 % mark detection
4 plot(box(1,:), box(2,:));
```

The image is gone!

```
1 figure;
2 imshow(img);
3 % mark detection
4 hold on;
5 plot(box(1,:), box(2,:))
```

You have to use `hold on` to draw several things in the same figure!

MATLAB is different - Find the Mistake (5)

```
1 plot = figure;
2 imshow(img);
3
4 % mark detection
5 hold on;
6 plot(x, y, 'g');
7
8 % save result figure
9 imwrite(plot,'plot.png');
```

MATLAB is different - Find the Mistake (5)

```
1 plot = figure;
2 imshow(img);
3
4 % mark detection
5 hold on;
6 plot(x, y, 'g');
7
8 % save result figure
9 imwrite(plot,'plot.png');
```

Index exceeds matrix dimensions!

MATLAB is different - Find the Mistake (5)

```
1 plot = figure;
2 imshow(img);
3
4 % mark detection
5 hold on;
6 plot(x, y, 'g');
7
8 % save result figure
9 imwrite(plot,'plot.png');
```

Index exceeds matrix dimensions!

MATLAB is different - Find the Mistake (5)

```
1 plot = figure;
2 imshow(img);
3
4 % mark detection
5 hold on;
6 plot(x, y, 'g');
7
8 % save result figure
9 imwrite(plot,'plot.png');
```

Index exceeds matrix dimensions!

MATLAB is different - Find the Mistake (5)

```
1  plot = figure;
2  imshow(img);
3
4  % mark detection
5  hold on;
6  plot(x, y, 'g');
7
8  % save result figure
9  imwrite(plot,'plot.png');
```

Index exceeds matrix dimensions!

```
1  detection = figure;
2  imshow(img);
3
4  % mark detection
5  hold on;
6  plot(x, y, 'g');
7
8  % save result figure
9  imwrite(detection, ...
10   'detection.png');
```

You **can** overwrite functions! Better name variables appropriately!

MATLAB is different - Find the Mistake (6)

```
1 lines = n/m;
2 lines = int(lines);
3 vector = zeros(lines, 1);
```

MATLAB is different - Find the Mistake (6)

```
1 lines = n/m;
2 lines = int(lines);
3 vector = zeros(lines, 1);
```

Undefined function 'int' for input arguments of type 'double'.

MATLAB is different - Find the Mistake (6)

```
1 lines = n/m;
2 lines = int(lines);
3 vector = zeros(lines, 1);
```

Undefined function 'int' for input arguments of type 'double'.

```
1 lines = n/m;
2 lines = uint8(lines);
3 vector = zeros(lines, 1);
```

Use `uint8`, `uint16`, `uint32`, `int8`, `int16`, `int32` or `int64` for conversion to (unsigned) int.

MATLAB is different - Find the Mistake (7)

```
1 img = double( ...
2   imread(filename));
3 figure;
4 imshow(img);
```

MATLAB is different - Find the Mistake (7)

```
1 img = double( ...
2   imread(filename));
3 figure;
4 imshow(img);
```

imshow, **image** and **imagesc** expect an image with type double to consist of values between 0 and 1.

MATLAB is different - Find the Mistake (7)

```
1 img = double( ...
2   imread(filename));
3 figure;
4 imshow(img);
```

imshow, **image** and **imagesc** expect an image with type double to consist of values between 0 and 1.

```
1 img = im2double( ...
2   imread(name));
3 figure;
4 imshow(img);
```

```
4 % or rescale manually
5 imshow(img/255);
```

Use im2double or rescale the image manually.

MATLAB is different - Find the Mistake (8)

```
1  img = imread(name);
2
3  % compute center
4  x = size(img, 1)/2;
5  y = size(img, 2)/2;
6
7  % plot center
8  imshow(img);
9  hold on;
10 plot(x, y, '.');
```

MATLAB is different - Find the Mistake (8)

```
1  img = imread(name);
2
3  % compute center
4  x = size(img, 1)/2;
5  y = size(img, 2)/2;
6
7  % plot center
8  imshow(img);
9  hold on;
10 plot(x, y, '.');
```

Matrices are accessed by (row, column) i.e. (y,x)!

MATLAB is different - Find the Mistake (8)

```
1  img = imread(name);
2
3  % compute center
4  x = size(img, 1)/2;
5  y = size(img, 2)/2;
6
7  % plot center
8  imshow(img);
9  hold on;
10 plot(x, y, '.');
```

```
1  img = imread(name);
2
3  % compute center
4  x = size(img, 2)/2;
5  y = size(img, 1)/2;
6
7  % plot center
8  imshow(img);
9  hold on;
10 plot(x, y, '.');
```

Matrices are accessed by (row, column) i.e. (y,x)!

MATLAB is different - Find the Mistake (9)

```
for i = 1:n
  for j = 1:m
    A(i,j) = i*j;
  end
end
```

MATLAB is different - Find the Mistake (9)

```
1 for i = 1:n
2   for j = 1:m
3     A(i,j) = i*j;
4   end
5 end
```

`i` and `j` are constants for the imaginary unit i.

MATLAB is different - Find the Mistake (9)

```
1 for i = 1:n
2   for j = 1:m
3     A(i,j) = i*j;
4   end
5 end
```

`i` and `j` are constants for the imaginary unit *i*.

```
1 for row = 1:n
2   for col = 1:m
3     A(row, col) = row*col;
4   end
5 end
```

Better use appropriately named variables.

Speed up MATLAB: Vectorization (1)

slow:

```
for row = 1:n
  for col = 1:m
    A(row, col) = ...
      2*B(row, col);
  end
end
```

Speed up MATLAB: Vectorization (1)

slow:

```
for row = 1:n
  for col = 1:m
    A(row, col) = ...
      2*B(row, col);
  end
end
```

faster:

```
A = 2*B
```

Speed up MATLAB: Vectorization (2)

slow:

```
1 i = 0;
2 for t = 0:.01:10
3   i = i + 1;
4   y(i) = sin(t);
5 end
```

Speed up MATLAB: Vectorization (2)

slow:

```
1 i = 0;
2 for t = 0:.01:10
3   i = i + 1;
4   y(i) = sin(t);
5 end
```

faster:

```
1 t = 0:.01:10;
2 y = sin(t);
```

Speed up MATLAB: Vectorization (3)

slow:

```
1 for n = 1:size(V,1)
2   V(n) = pi*(D(n)^2)*H(n);
3 end
```

Speed up MATLAB: Vectorization (3)

slow:

```
1 for n = 1:size(V,1)
2   V(n) = pi*(D(n)^2)*H(n);
3 end
```

faster:

```
1 V = pi*(D.^2).*H;
```

Speed up MATLAB: Vectorization (4)

slow:

```
% clip data
for row = 1:n
  for col = 1:m
    if A(row, col)>255
      A(row, col) = 255;
    end
  end
end
```

Speed up MATLAB: Vectorization (4)

slow:

```
% clip data
for row = 1:n
  for col = 1:m
    if A(row, col)>255
      A(row, col) = 255;
    end
  end
end
```

faster:

```
% clip data
A(A > 255) = 255;
```

Some more useful stuff

- `tic`, `toc` for timing measurement
- `ls` also works within matlab environment
- Cell arrays
  - `c = cell(n)` creates an $n \times n$ cell array of empty matrices
  - `c = cell(m, n)` $m \times n$ cell array
  - access via `c{idx}`
  - elements of cell array can be of different size and type
- Structs
  - `st = struct('field1', values1, 'field2', values2, ...)`
  - `st.field1 = values1; st.field2 = values2; ...`
- Cell arrays and structs can be combined

```
c{1}.field1 = values1; c{1}.field2 = values2;
```

RWTH AACHEN UNIVERSITY

Useful MATLAB ressources

- MATLAB documentation
  `https://www.mathworks.com/help/index.html`
- File Exchange
  `https://www.mathworks.com/matlabcentral/fileexchange/`
- Code Vectorization Guide
  `https://www.mathworks.com/support/tech-notes/1100/1109.html`
- Writing Fast MATLAB code
  `https://www.mathworks.com/matlabcentral/fileexchange/5685`
- MATLAB array manipulation tips and tricks
  `http://home.online.no/~pjacklam/matlab/doc/mtt/index.html`