Computer Vision / Lab Exercises

– Introduction to MatLab –

Colin Doert, Kai Lienemann

University of Dortmund
Intelligent Systems Group
12. Februar 2007
What is MatLab

- A numerical computing environment
  - Designed for matrix computation (Matrix Laboratory)
  - Offers toolboxes for various application areas

- A programming language comprising:
  - powerful visualising features
  - debugger & profiler
  - compiler
  - GUI development features
What is MatLab

- A numerical computing environment
  - Designed for matrix computation (Matrix Laboratory)
  - Offers toolboxes for various application areas
- A programming language comprising:
  - powerful visualising features
  - debugger & profiler
  - compiler
  - GUI development features
Getting started

Everything is a matrix:

- **Entering Matrices:** 
  \[
  A = \begin{bmatrix}
  1 & 6 & 4 \\
  7 & 3 & 9 
  \end{bmatrix}
  \]

- **Accessing Elements:** 
  \[B = A(1,3) \Rightarrow B = 4\]

- **Colon Operator:**
  - Produces a sequence of numbers:
    \[C = 1:2:5 \Rightarrow C = \begin{bmatrix}
    1 & 3 & 5 
  \end{bmatrix}\]
  - Used to access a range of indices:
    \[C = A(1,1:3) \Rightarrow C = \begin{bmatrix}
    1 & 6 & 4 
  \end{bmatrix}\]
  - Get one dimension:
    \[C = A(1,:) \Rightarrow C = \begin{bmatrix}
    1 & 6 & 4 
  \end{bmatrix}\]
  - Get all elements of a matrix in a vector:
    \[C = A( :) \Rightarrow C = \begin{bmatrix}
    1;7;6;3;9;4 
  \end{bmatrix}\]

- **Generating special matrices:**
  - `zeros(rows, columns)` *(all zero matrix)*
  - `ones(rows, columns)` *(all ones matrix)*
  - `rand(rows, columns)` *(uniformly distributed random elements)*
Getting started

Everything is a matrix:

- **Entering Matrices**: \( A = \begin{bmatrix} 1 & 6 & 4 \\ 7 & 3 & 9 \end{bmatrix} \)

- **Accessing Elements**: \( B = A(1,3) \Rightarrow B = 4 \)

- **Colon Operator**:
  - Produces a sequence of numbers:
    \( C = 1:2:5 \Rightarrow C = \begin{bmatrix} 1 & 3 & 5 \end{bmatrix} \)
  - Used to access a range of indices:
    \( C = A(1,1:3) \Rightarrow C = \begin{bmatrix} 1 & 6 & 4 \end{bmatrix} \)
  - Get one dimension:
    \( C = A(1,:) \Rightarrow C = \begin{bmatrix} 1 & 6 & 4 \end{bmatrix} \)
  - Get all elements of a matrix in a vector
    \( C = A(:) \Rightarrow C = \begin{bmatrix} 1;7;6;3;9;4 \end{bmatrix} \)

- **Generating special matrices**:
  - `zeros(rows, columns)` *(all zero matrix)*
  - `ones(rows, columns)` *(all ones matrix)*
  - `rand(rows, columns)` *(uniformly distributed random elements)*
Operators for scalars and matrices

A = [ 1 2 ; 4 5 ];
B = [ 9 8 ; 6 5 ];

%% matrix and scalar
C = (2.6 * A) + 9.7;

%% matrix and matrix
D = (A * B) / C;

%% matrix element-wise
E = D ./ (A .* C);

▪ matrix with scalar
  +, -, *, /, ^

▪ matrix with matrix
  * (matrix product), / (division)

▪ matrix with matrix element-wise
  +, -, .* , ./
Useful general functions

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

\[
[x \ y] = \text{size}(A);
\]

\[
[d \ e \ f] = \text{sum}(A);
\]

\[
s = \text{sort}(A);
\]

\[
\text{ind} = \text{find}(A == 1);
\]

\[
I = \text{inv}(A);
\]

- **size**  Size of an array
- **sum**  Sum along one dimension of an array
- **sort**  Sorts along one dimension of an array
- **find**  Determines indices that meet a logical condition
- **inv**  Inverse of a matrix
if (count > limit)
    fprintf('true\n');
else
    fprintf('false\n');
end;

for i=0:0.5:100
    A(i)=sin(i*5)+1;
end;

while (count < limit)
    count = count + 1;
end;

**Conditional structure:**

- **if and else** Execute a statement or block only if a condition is fulfilled.

**Iteration structures (loops)**

- **for-loop** Repeat the enclosed block and increase a variable, as long as the condition is fulfilled.
- **while-loop** Repeat a statement or block, as long as the conditional expression is true.
Programming in MatLab – Control Structures

if (count > limit)
    fprintf('true\n');
else
    fprintf('false\n');
end;

for i=0:0.5:100
    A(i)=sin(i*5)+1;
end;

while (count < limit)
    count = count + 1;
end;

Conditional structure:

    if and else     Execute a statement or block only if a condition is fulfilled.

Iteration structures (loops)

    for-loop      Repeat the enclosed block and increase a variable, as long as the condition is fulfilled.

    while-loop    Repeat a statement or block, as long as the conditional expression is true.
if (count > limit)
    fprintf('true\n');
else
    fprintf('false\n');
end;

for i=0:0.5:100
    A(i)=sin(i*5)+1;
end;

while (count < limit)
    count = count + 1;
end;

**Conditional structure:**

**if and else** Execute a statement or block only if a condition is fulfilled.

**Iteration structures (loops)**

**for-loop** Repeat the enclosed block and increase a variable, as long as the condition is fulfilled.

**while-loop** Repeat a statement or block, as long as the conditional expression is true.
Programming in MatLab – Function Definition I

▶ Program files are named: *functionname.m*
▶ Start *MATLAB* editor: *edit functionname.m*

▶ M-File Structure:

```matlab
function result = multiply(arg1, arg2)

if (nargin ~= 2)
    error('No arguments given to function multiply!')
end;

result = arg1 * arg2;
```

Colin Doert, Kai Lienemann: Computer Vision / Lab Exercises
Programming in MatLab – Function Definition II

- All return values and arguments are optional.
- If special arguments are required, check the number of input arguments \(nargin\).
- Number of output arguments is determined by \(nargout\).

%function with 1 return value and 2 arguments
function ret = multiply(arg1, arg2)
ret = arg1 * arg2;

%function with 2 return values and several arguments
function [ret1,ret2] = f2(a1, a2, a3, a4, optargs)
ret1 = a1 + a2;
ret2 = a3 - a4;
fprintf('%0f optional arguments.
',length(optargs));
Data Types in MatLab

- **Matrices**: Store all values in a rectangular matrix.
  \[
  A = \begin{bmatrix}
  1 & 2 & 3 \\
  4 & 5 & 6
  \end{bmatrix};
  \]

- **Cell Array**: Store values of different types in one array.
  \[
  C = \{[1 3; 0 8], 'Anne'; 3, -\pi:\pi/4:\pi\};
  \]

- **Struct**: Structure array with specified fields and values of (possibly) different types.
  \[
  B = \text{struct('field1', values1, 'field2', values2)}
  \]

- **uint8, uint16, complex** ...
Efficiency

A = []; for i=1:1:500
    for j=1:1:500
        A(i,j)=1;
    end;
end;

% alternatives:
A(1:500,1:500) = 1;
A = ones(500,500);

% usage of find
A(1,1:30:500) = rand;
[row, col] = find(A ~= 1);}

- Loops reduce the speed of your functions! ⇒ Try avoiding them!

- Loops can be often expressed by matrix operations or alternative function calls (read the help pages!).

- Use find to do operations on all elements of a matrix that fulfill a special condition.
Efficiency

Loops reduce the speed of your functions!
⇒ Try avoiding them!

Loops can be often expressed by matrix operations or alternative function calls (*read the help pages!*).

Use `find` to do operations on all elements of a matrix that fulfill a special condition.

```matlab
A = []; for i=1:1:500     % alternatives: A(1:500,1:500) = 1; A = ones(500,500); end; for j=1:1:500
    A(i,j)=1;     % usage of find end; A(1,1:30:500) = rand; [row, col] = find(A ~= 1);}
```
Efficiency

A = [];  
for i=1:1:500  
    for j=1:1:500  
        A(i,j)=1;  
    end;  
end;  

% alternatives:  
A(1:500,1:500) = 1;  
A = ones(500,500);  

% usage of find  
A(1,1:30:500) = rand;  
[row, col] = find(A ~= 1);  

- Loops reduce the speed of your functions!  
  ⇒ Try avoiding them!  

- Loops can be often expressed by matrix operations or alternative function calls (read the help pages!).  

- Use find to do operations on all elements of a matrix that fulfill a special condition.
Example: Image Processing Toolbox

```matlab
% load and display a RGB-image
RGB = imread('landscape.jpg');
imshow(RGB);

% convert the image into a grayscale image and display it
BW = rgb2gray(RGB);
imshow(BW);

% write the grayscale image
imwrite(BW, 'landscape_bw.jpg');

% create and display a histogram
imhist(BW);
```
Help & Support

- `help` command on the **MATLAB**-prompt
- **MATLAB** Online Help
- Producer Homepage:
  www.MathWorks.com
  www.prenhall.com/gonzalezwoodseddins
Setting Up the System

▶ Log in with the distributed accounts.
▶ Download the course material from

```
ls12-www.cs.uni-dortmund.de/~fink/lectures/WS06/computervision.html
```

▶ Extract the archive in the personal directory.
▶ Start MATLAB by double-click on the desktop item.
▶ Add the (recursive) path of the extracted archive.

```
addpath(genpath('R:\Path\ToFiles\'))
```

Have fun!
Setting Up the System

- Log in with the distributed accounts.
- Download the course material from
  
  ls12-www.cs.uni-dortmund.de/~fink/lectures/WS06/computervision.html

- Extract the archive in the personal directory.
- Start MATLAB by double-click on the desktop item.
- Add the (recursive) path of the extracted archive.

  ```matlab
  addpath(genpath('R:\Path\ToFiles\'))
  ```

Have fun!
Setting Up the System

- Log in with the distributed accounts.
- Download the course material from
  
  `ls12-www.cs.uni-dortmund.de/~fink/lectures/WS06/computervision.html`

- Extract the archive in the personal directory.
- Start `MATLAB` by double-click on the desktop item.
- Add the (recursive) path of the extracted archive.

```matlab
addpath(genpath('R:\Path\ToFiles\'))
```

Have fun!
Setting Up the System

- Log in with the distributed accounts.
- Download the course material from
  
  ls12-www.cs.uni-dortmund.de/~fink/lectures/WS06/computervision.html

- Extract the archive in the personal directory.
- Start MATLAB by double-click on the desktop item.
- Add the (recursive) path of the extracted archive.

```matlab
addpath(genpath('R:\Path\ToFiles\'))
```

Have fun!