Introduction to Matlab

Simone Gasparini

How to get Matlab

- Computer-equipped classroom
  - Complete list available at http://www.cia.polimi.it/servizi/software.html (then select wished Matlab released)

- Download it from polimi-cdserver (Campus license)
  - http://www.cia.polimi.it/servizi/software.html?cdserv=1
  - It works only inside the polimi network

- Your own copy…

Matlab Documentation

- Several tutorial available on the Internet

- Help on line

Matlab overview

- MATrix LABoratory
- High-performance language for technical computing.
- It integrates computation, visualization, and programming in an easy-to-use environment

- Problems and solutions are expressed in familiar mathematical notation.

- Used for:
  - Math and computation
  - Algorithm development
  - Data acquisition
  - Modeling, simulation, and prototyping
  - Data analysis, exploration, and 2D-3D visualization
  - Scientific and engineering graphics
  - Application development, including GUI building
Matlab overview

Platforms

Matlab works on several platforms
- PCs powered by Windows (provided by www.cia.polimi.it)
- Unix/Linux Systems (provided by www.cia.polimi.it)
- Macintosh (not provided)

Similar GUI and interface

Toolboxes

MATLAB features a family of add-on application-specific solutions called toolboxes.
Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems.
They cover many areas:
- signal processing,
- control systems,
- neural networks,
- fuzzy logic,
- wavelets,
- simulation…

Matrix is the basic type
- Scalar values are 1x1 matrices
- Matrix is a rectangular array of numbers.

Other programming languages (eg C/C++, Java) work with numbers one at a time
MATLAB works with entire matrices quickly and easily.
Operations in MATLAB are designed to be as natural as possible.
- E.g.: >> A+B
- No matters if A or B are vectors, matrices or scalar values
- MATLAB automatically compute the sum
- Only dimensions matter in order to have consistent operations
  - A and B can’t be matrices with different dimensions

Command Window: Functions and commands can be entered here and executed. (matlab works like a calculator)
Workspace
All stored variables are shown here together with their current values. They can be also edited.

Command History
All executed command are listed here and can be recalled (re-executed) when needed by double clicking.

Current Directory
Current working directory,

Help
Help on line
Artificial Intelligence and Robotics Laboratory

**Variables**

- MATLAB variable is a tag assigned to a value while it remains in memory (workspace).
- A “value” can be:
  - A scalar value
  - A matrix
  - A string, which is a vector of char
- Names must begin with a letter, which may be followed by any combination of letters, digits, and underscores.
- MATLAB is case sensitive.
- Assign a value to a scalar
  - `>> Var_name = value`

**Matrices**

- There are several ways to enter a matrix:
  - Enter an explicit list of elements.
  - Load matrices from external data files.
  - Generate matrices using built-in functions.
  - Create matrices with your own functions in M-files.
- explicit list of elements

**Semicolon at the end of commands suppresses output**

**Variables are added to the workspace. The values are shown and can be manually edited by double-clicking**
Matrices

- Separate the elements of a row with blanks or commas.
- Use a semicolon, ; , to indicate the end of each row.
- Surround the entire list of elements with square brackets, [ ].

Vectors

- Vectors are mono-dimensional matrices

Working with matrices: Example

- ans is the default variable where the output is stored
- Results can be stored into a different variable by just assigning the result
Matrix elements

- To access to single elements of matrix use $A(r,c)$ notation, where $r$ is the index of row and $c$ index of column
- C-like programmers, watch out!
  - The first element (row or column) has index 1

```
Command Window
>> A = 
 16  3  11
 9  10  11
 6  9  7
>> A(1,3)
ans = 
3
>> A(1) = 14
>> A = 
16  3  14
 9 10 11
 6  9  7
>> A(1,3) = -10
A = 
16  3 13
 9 10 11
 6  9  7
>> A(1,:) = 100
A = 
100  3 100
 9 100 11
 6  9  7
```

Colon operator

- The colon `:` operator is the most important MATLAB operator
- Use `starting_value:increment:end_value`
- It returns a vector containing all values from `starting_value` to `end_value` with step `increment`

```
Command Window
>> v = 1:10
v = 
1     2     3     4     5     6     7     8     9    10
>> v = 1:10
v = 
1     2     3     4     5     6     7     8     9    10
>> v = 1:2:10
v = 
1     3     5     7     9
```

Colon operator

- Very useful to access to a part of a matrix
- end refers to the last element (row or column)

```
Command Window
>> k = 3
k = 
3
>> A(1:k,3)
ans = 
2
11
7
>> sum(A(1:k,3))
ans = 
20
>> sum(A(1,3))
ans = 
34
>> sum(A(:,end))
ans = 
34
```

Operators

- Arithmetic operators
  - + addition
  - - subtraction
  - * multiplication
  - / division
  - ^ power
  - ' transpose
Addition example

Command Window

```matlab
>> A
A =
16 3  2 13
 5 10 11  8
 9  6  7 12
 4 100 14  1
>> A + A
ans =
32  8 11 17
 8 12 17 109
11 17 14 25
17 100 26  2
>> A + 2
ans =
18  5  4 15
 7 12 13 10
11  6  9 14
 6 102 16  3
```

Sum correspondent elements

Sum the scalar value to each element

Multiplication example

Command Window

```matlab
>> A
A =
16 3  2 13
 5 10 11  8
 9  6  7 12
 4 100 14  1
>> A * A
ans =
256  261  260
261  961  209  265
285  3329  101  261
604  2105 1220 1021
>> A * 2
ans =
32  6  4  25
16 20 22 16
18 12 14 24
 8 200 28  2
>> A * (1:4)'
ans =
80
 90
 90
280
```

Classical matrix product (rows by columns)
Each element multiplied by 2
Matrix – vector product

Element-wise operators

- Operators that work on matrix elements
  - + sum
  - - subtraction
  - .* element by element multiplication
  - ./ element by element division
  - .^ element by element power

Example

Command Window

```matlab
>> A
A =
16 3  2 13
 5 10 11  8
 9  6  7 12
 4 100 14  1
>> A .* A
ans =
256  261  260
261  961  209  265
285  3329  101  261
604  2105 1220 1021
>> A .^ A
ans =
256  9  4 169
25 100 121 64
01  26 49 146
 1 10000 121  1
>> A ./ A
ans =
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
>> A ./ A
ans =
1.0000 0.6000 0.2312 1.2500
1.6667 1.0000 1.8333 0.8000
4.2857 0.3462 1.0000 0.8971
0.3077 12.3800 1.1467 1.0000
```

Classical matrix product (rows by columns)
Elementwise division
### Concatenation

- **Concatenation** is the process of joining small matrices to make bigger ones.
- The pair of square brackets, [], is the concatenation operator.

```matlab
B = [A(1:2, 3:4); A(2:3, 1:2); A(1:1, 1:1)]
>> C = [B; A(1, 1:4); A(2:1, 2:5)]
```

### Deleting row and columns

- The pair of square brackets "[]" can be used to delete rows or columns of a matrix.

```matlab
X = [16 2 13 0
    5 10 11 9
    6 7 12 0
    4 100 14 1
    0 0 0 0 -10];
>> X(:, 3) = [];
Delete 3rd column of X
>> X(2:1, 1) = [];
Delete 2nd row of X
```

### Other functions

- `max(A)` return an array with the maximum value of each column
  - Call `max(max(A))` to get the overall maximum of A
- `min(A)` like `max` return an array with the minimum value of each column
- `length(A)` return the value of the larger dimension of A
- `size(A)` return the values of the dimensions of A
  - `d = size(A)`
    - Put in matrix d the dimensions of A (d = [r c])
  - `[r, c] = size(A)`
    - Put in separate variables r and c the dimensions of A
  - `m = size(A, dim)`
    - Put in m the size of dimensions dim of A (dim = 1 means rows, 2 for columns)

### Example

```matlab
A = [16 3 2 13 0
    5 10 11 8 0
    9 6 7 12 0
    4 100 14 1 0
    0 0 0 0 -10];
>> max(A)
ans =
    16
>> length(A)
ans =
    5
>> length(A(1:1, 1:4))
ans =
    5
>> size(A)
size =
    5 5
>> [r, c] = size(A)
[r, c] =
    5
>> [r, c] = size(A)
c =
    5
>> r = size(A, 1)
r =
    5
>> c = size(A, 1)
c =
    5
```

---

This document provides an overview of basic operations in MATLAB, including matrix creation, concatenation, deletion of rows and columns, and the use of the `max`, `min`, `length`, and `size` functions. Examples are given to illustrate how these commands are used in practice.
Special matrices

- `eye(n)` returns the n-by-n identity matrix
- `zeros(n,m)` returns the n-by-m matrix of all zeros
- `ones(n,m)` returns the n-by-m matrix of all one
- `rand(n,m)` returns the n-by-m matrix of random numbers

Solving linear systems

- Linear systems in matrix form can be solved using operator "\"
- Given `Ax = b` with `x` vector of unknown, the system can be solved as `x = A\b` which corresponds to `x = inv(A) * b`

```
A = [2 5 6; 6 4 -12; 1 0 4]
b = [32; 6; 4]
x = A\b
```

If the equation `Ax = b` does not have a solution (and `A` is not a square matrix), `x = A\b` returns a least squares solution

Using MATLAB as a programming language

- Use M-file to create script or function that can be called from the command prompt
- Script are sequence of function/command call that can be executed in batch calling the m-file
- M-file is a normal text file that contains the sequence of commands to be execute
- To execute a M-file simply call the name of the m-file from command prompt
- M-file can be edited using the provide editor of MATLAB, but you can use any text editor you like
- MATLAB provide flow control structures (if, for, while ecc) that can be used in a m-file

MATLAB m-file editor
Example – Calling a m-file

```plaintext
37
Artificial Intelligence and Robotics Laboratory

Flow control – if...else and elseif

if expression1
    statements1
elseif expression2
    statements2
elseif expression3
    statements3
    ...
else
    statements4
end

If expression1 evaluates as true executes the one or more commands denoted here as statements1; else if expression2 is true executes statements2 and so on

Artificial Intelligence and Robotics Laboratory

Flow control – conditional expression

- Similar to C expressions
- Relational operators
  - <, <=, ==, >=, >, ~= ("not equal", ALT+0126 to get '\~')
- Logical operators
  - & ("and"), | ("or"), ~ ("not")

Artificial Intelligence and Robotics Laboratory

Flow control - switch

switch switch_expr
    case case_expr
        statement, ..., statement
    case {case_expr1, case_expr2, case_expr3, ...}
        statement, ..., statement
    otherwise
        statement, ..., statement
end

Artificial Intelligence and Robotics Laboratory

Flow control – if...else and elseif

if expression1
    statements1
elseif expression2
    statements2
elseif expression3
    statements3
    ...
else
    statements4
end

If expression1 evaluates as true executes the one or more commands denoted here as statements1; else if expression2 is true executes statements2 and so on

Artificial Intelligence and Robotics Laboratory

Flow control – conditional expression

- Similar to C expressions
- Relational operators
  - <, <=, ==, >=, >, ~= ("not equal", ALT+0126 to get '\~')
- Logical operators
  - & ("and"), | ("or"), ~ ("not")

Artificial Intelligence and Robotics Laboratory

Flow control - switch

switch switch_expr
    case case_expr
        statement, ..., statement
    case {case_expr1, case_expr2, case_expr3, ...}
        statement, ..., statement
    otherwise
        statement, ..., statement
end

Artificial Intelligence and Robotics Laboratory

Flow control – if...else and elseif

if expression1
    statements1
elseif expression2
    statements2
elseif expression3
    statements3
    ...
else
    statements4
end

If expression1 evaluates as true executes the one or more commands denoted here as statements1; else if expression2 is true executes statements2 and so on

Artificial Intelligence and Robotics Laboratory

Flow control – conditional expression

- Similar to C expressions
- Relational operators
  - <, <=, ==, >=, >, ~= ("not equal", ALT+0126 to get '\~')
- Logical operators
  - & ("and"), | ("or"), ~ ("not")

Artificial Intelligence and Robotics Laboratory

Flow control - switch

switch switch_expr
    case case_expr
        statement, ..., statement
    case {case_expr1, case_expr2, case_expr3, ...}
        statement, ..., statement
    otherwise
        statement, ..., statement
end

Artificial Intelligence and Robotics Laboratory
Flow control - for

```
for variable = startingval:step:endingval
    statements
end
```

- Example

```
[r c] = size(A)
for m = 1:r
    for n = 1:c
        A(m,n) = 1/(m+n -1);
    end
end
```

- The counter variables are managed automatically by MATLAB inside the loop!

Flow control - while

```
while expression
    statements
end
```

- Similar to C

- For and while inside the loop allow the use of:
  - `continue` to pass the control to the next iteration
  - `break` to exit early from the loop

Function

- Functions are stored in m-file named with the same name of the function
- Declaration:
  ```
  function [y1, ... , yn] = nome_funzione (x1, ... , xm)
  ```
- Where
  - `x1, ... , xm` are input params (optional)
  - `[y1, ... , yn]` are output params (optional)
- N.B. All input parameters are passed by value
  - They are not affected by change inside the function

Function example
Function example

```matlab
% Normalize each set of points so that the origin is at centroid and
% mean distance from origin is sqrt(2). normalizeWeights also enforces the
% scale parameter is 1. Note that 'homography2d' will also call
% 'normalizeWeights' but the code is 'canned' that calls the distance
% function will not - so it is best that we normalize beforehand.
41: [x1, T1] = normalizeWeights(x1);
42: [x2, T2] = normalizeWeights(x2);
43: p = 4; % Minimum Np of points needed to fit a homography.
44: fittingData = Homography2d;
45: distData = @normalizeData;
46: degData = @isDegenerate;
47: % x1 and x2 are 'stacked' to create a 6xN array for canonic
48: [M, inliers] = canonic([x1; x2; fittingData, distData, degData, x, t1);
49: % Now do a final least squares fit on the data points considered to
50: % be inliers.
51: H = homography2d([x1;inliers], [x2;inliers]);
52: % Normalize
53: H = T1\H';
```

Graphics

- Function plot allow to create 2D graphics in MATLAB
  - `plot(Y)` plots the columns of Y versus their index
  ```matlab
  Command Window:
  >> t = linspace(0,2*pi);
  >> y = sin(t);
  >> plot(t,y)
  >>
  ```

- To plot different graphics on the same reference frame
  - `plot(x1,y1, x2,y2, x3,y3, x4,y4,...)`
  ```matlab
  >> t = linspace(0,2*pi);
  >> y = sin(t);
  >> y2 = sin(t - .4);
  >> y3 = sin(t - .8);
  >> y4 = sin(t + 1.3);
  >> plot(t,y,t,y2,t,y3,t,y4)
  ```

Graphics
Graphics

Plot style

\texttt{plot(x_1,..., 'options')}

Option is a string delimited by ' containing the specifiers for line styles

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Color</th>
<th>Line Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Red</td>
<td>Solid line (default)</td>
</tr>
<tr>
<td>--</td>
<td>Green</td>
<td>Dashed line</td>
</tr>
<tr>
<td>:</td>
<td>Blue</td>
<td>Dotted line</td>
</tr>
<tr>
<td>.</td>
<td>Magenta</td>
<td>Dash-dotted line</td>
</tr>
<tr>
<td>g</td>
<td>Yellow</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>Black</td>
<td></td>
</tr>
<tr>
<td>w</td>
<td>White</td>
<td></td>
</tr>
</tbody>
</table>

\texttt{plot(x_1,y_1, ':ro')}

Plot a red (r) dotted (: ) line placing a circle (o) on each point

Images

In MATLAB images are matrices where each element store the value of the correspondent pixel

Gray-scale images are bidimensional matrices

- \( I(i,j) \) where \( i \) and \( j \) define the location of the pixel and \( k \) specifies the channel (R, G or B)
- \( I(i,j) \) returns the graylevel of pixel at coordinate \( i,j \)

Color images (RGB) are tridimensional matrices

- \( I(i,j,k) \) where \( i \) and \( j \) define the location of the pixel and \( k \) specifies the channel (R, G or B)
- \( I(i,j,k) \) returns the value of channel \( k \) of pixel at coordinate \( i,j \)

load and display an image
Working with images - example

Fundamental matrix: example

\[
x^T F x = 0
\]

\[
x' x f_{11} + x' y f_{12} + x f_{13} + y' x f_{21} + y' y f_{22} + y f_{23} + x f_{31} + y f_{32} + f_{33} = 0
\]

separate known from unknown

\[
\begin{bmatrix}
x' x, x' y, x', y' x, y' y, x, y, 1 & f_{11}, f_{12}, f_{13}, f_{21}, f_{22}, f_{23}, f_{31}, f_{32}, f_{33}
\end{bmatrix}^T = 0
\]

(data) (linear)

\[
\begin{bmatrix}
x_{1}, y_{1}, x_{1}', y_{1}', x_{2}, y_{2}, \ldots, x_{n}, y_{n}, x_{n}', y_{n}', 1
\end{bmatrix}^T = 0
\]

(unknowns) (linear)

\[
F = 0
\]

\[
A f = 0
\]
Fundamental matrix: example

\[
\min_{F} \| Af \|^2 = 0
\]

since \( \| Af \|^2 = f^T A^T A f \)

this requires to find the eigenvector associated to the smallest eigenvalue of \( A^T A_{8x8} \) (e.g. svd)

Due to data noise the constraint rank(F)=2 is not enforced

Use svd on F and set the smallest singular value to 0

\[ \text{svd}(F) = Q D R \quad (D \text{ diagonal mat, } Q \text{ and } R \text{ orthogonal}) \]

Orders of magnitude difference
Between column of data matrix
→ least-squares yields poor results

Transform image to ~[−1,1]x[−1,1]

Heuristic: \(4 \times \text{image size} / \text{SDP size} \)