MATLAB Workshop
English Version

Ben Glocker, Tobias Sielhorst, Jörg Traub
Dr. Selim Ben Himane
Chair for Computer Aided Medical Procedures
adapted from the MATLAB(c) Documentation
3D Computer Vision

October 19, 2007

1 Basics

MATLAB is a numerical mathematics environment. Its easy-to-learn interface and script language supports rapid prototyping of algorithms. It is also a powerful environment for image processing and vision.

1.1 Environment

After starting MATLAB is ready to use by its command line within the Command Window. After entering formulas are solved and the answer `ans` is given immediately.

```
>> 2+2

ans =
     4
```

Besides the Command Window there are two important windows. The MATLAB Workspace shows you all variables that are currently in the MATLAB memory. By clicking on the variables names the data can be directly accessed via a dialog window. The third window is the Command History. All commands that were typed in are listed here.

2 Matrices

2.1 Scalars, Vectors Matrices

In general MATLAB stores numeric data in matrices. Vectors are stored in matrices with only one row or column, scalars are just 1-by-1 matrices. So first of all we will learn how to handle matrices.

2.1.1 Entering data

Matrices can be entered in several different ways:

- 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.

To enter a matrix explicitly, simply type in the Command Window

```
>> A = [1 2 3; 4 5 6; 7 8 9]
```

or

```
>> A = [1,2,3; 4,5,6; 7,8,9]
```

Matrices are given line by line. To change the colon, use a *space* or a *comma*, to change the line use a *semicolon*. MATLAB displays the matrix you just entered:

```
A =
      1   2   3
      4   5   6
      7   8   9
```

Once a matrix has been entered, it is automatically remembered in the MATLAB workspace. It can be simply refered by typing `A`.

2.1.2 Subscripts

The element in the row `i` and the column `j` of `A` is denoted by `A(i,j)`.
Typing

```
>> A(1,3) + A(3,2)
```

will give the result

```
ans =
     11
```

If you assign a value to a position that is out of the range of the matrix, MATLAB increases the size automatically.

```
>> X = A;

>> X(3,4) = 10

X =
      1   2   3    0
      4   5   6    0
      7   8   9   10
```

2.1.3 Colon operator

The colon `:` is one of the most important MATLAB operators. For example the expression

```
>> 1:10
```

is a row vector containing the integers from 1 to 10:

```
ans =
      1   2   3   4   5   6   7   8   9   10
```

To obtain nonunit spacing, specify an increment:

```
>> 15:-2.5:5

ans =
      15   12.5   10   7.5   5
```

Subscript expressions involving colons refer to portions of a matrix:

```
>> A(1:k,j)
```

is the first `k` elements of the `j`th column of `A`. The colon by itself refers to all the elements in a row or column of a matrix and the keyword `end` refers to the last row or column.

```
>> A(:,end)
```

represents the last column vector of `A`:

```
ans =
      3
      6
      9
```

2.2 Expressions

Like most other programming languages, MATLAB provides mathematical expressions, but unlike most programming languages, these expressions involve entire matrices. The building blocks of expressions are

- Varibales
- Numbers
- Operators
- Functions

2.2.1 Variables

MATLAB does not require any type declarations or dimension statements.

```
>> num_students = 25
```

creates a 1-by-1 matrix named `num_students` and stores the value 25 in its element. Variable names consist of a letter, followed by any number of letters, digits, or underscores. MATLAB is case sensitive.

2.2.2 Numbers

MATLAB uses conventional decimal notation, with an optional decimal point and leading plus or minus sign, for numbers.

```
3           -99           0.0001
9.6397238   1.60210e-20   6.02252e23
1i          -3.14159j     3e5i
```

All numbers are stored internally using the long format specified by the IEEE floating-point standard. Floating-point numbers have a finite precision of roughly 16 significant decimal digits and a finite range of roughly 10^{-308} to 10^{+308}.

2.2.3 Operators

Expressions use familiar arithmetic operators and precedence rules.

`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`\`	Left Division
`^`	Power
`'`	Complex conjugate transpose
`( )`	Specify evaluation order

2.2.4 Functions

MATLAB provides a large number of standard elementary mathematical functions, including `abs`, `sqrt`, `exp`, and `sin`. For a list of the elementary mathematical functions, type

```
>> help elfun
```

For a list of more advanced mathematical and matrix functions, type

```
>> help specfun

>> help elmat
```

2.3 Working with matrices

2.3.1 Generating

MATLAB provides different functions that generate basic matrices.

`zeros`	All zeros
`ones`	All ones
`rand`	Uniformly distributed random elements
`randn`	Normally distributed random elements

Here are some examples:

```
>> Z = zeros(2,4)

Z =
      0   0   0   0
      0   0   0   0


>> F = 5*ones(3,3)

F =
      5   5   5
      5   5   5
      5   5   5


>> N = fix(10*rand(1,10))

N =
      9   2   6   4   8   7   4   0   8   4


>> R = randn(4,4)

R =
```

```
 0.6353   0.0860  -0.3210  -1.2316
-0.6014  -2.0046   1.2366   1.0556
 0.5512  -0.4931  -0.6313  -0.1132
-1.0998   0.4620  -2.3252   0.3792
```

2.3.2 Saving and loading

The `save` function saves variables that exists in the MATLAB Workspace. For example, typing the entries of a certain matrix:

```
>> A = [16.0, 3.0, 2.0, 13.0;...
5.0, 10.0, 11.0, 8.0;...
9.0, 6.0, 7.0, 12.0;...
4.0, 15.0, 14.0, 1.0]
```

will give the following answer:

```
A =
      16.0   3.0   2.0  13.0
       5.0  10.0  11.0   8.0
       9.0   6.0   7.0  12.0
       4.0  15.0  14.0   1.0
```

To save this variable in a binary file, type:

```
>> save matrix.mat A
```

The `load` function reads binary files containing matrices generated by earlier MATLAB sessions, or reads text files containing numeric data. For example loading a text file containing these four lines. The statement

```
>> load matrix.mat
```

reads the file `matrix.mat` and creates a variable `A` containing the example matrix.

2.3.3 Concatenation

Concatenation is the process of joining small matrices to make bigger ones. The pair of square brackets `[]` is the concatenation operator. For example:

```
>> v = [1;4;7];

>> A = [v v+1 v+2]

A =
      1  2  3
      4  5  6
      7  8  9
```

2.3.4 Deleting Rows and Columns

Complete rows and columns can be deleted by using just a pair of square brackets

```
>> X = A;

>> X(:,2) = []
```

This changes `X` to

```
X =
      1   3
      4   6
      7   9
```

2.4 Linear Algebra

The mathematical operations defined on matrices are one subject of linear algebra. In the following some operations will be presented.

2.4.1 Matrix Operations

Multiplication The multiplication symbol `*` denotes the matrix multiplication involving inner products between rows and columns:

```
>> A*A'
```

Transpose The matrix transpose can be applied by using `transpose` or, for real matrices, by using `'`. Adding a matrix to its transpose produces a symmetric matrix:

```
>> A+A'

ans =
       2    6   10
       6   10   14
      10   14   18
```

Inverse The inverse can be get just by

```
>> inv(A)
```

2.4.2 Matrix Properties

Determinant The determinant of a matrix can be computed by

```
>> det(A)
```

Eigenvalue Decomposition To compute the Eigenvalues and the Eigenvectors

```
>> [V,D] = eig(A)
```

Singular Value Decomposition To compute the Singular Value Decomposition of a matrix `A`

```
>> [U,D,V] = svd(A)
```

2.5 Controlling Input and Output

2.5.1 Format Function

The `format` function controls the numeric format of the values displayed by MATLAB. The function affects only how numbers are displayed, not how MATLAB computes or saves them. Examples

```
>> x = [4/3 1.2345e-6]

>> format short

x =
       1.3333   0.0000

>> format short e

x =
       1.3333e+000   1.2345e-006

>> format short g

x =
       1.3333   1.2345e-006

>> format long

x =
       1.33333333333333   0.00000123450000

>> format rat

x =
       4/3   1/810045
```

In addition

```
>> format compact
```

suppresses many of the blank lines that appear in the output.

2.5.2 Suppressing Output

If you simply type a statement and press Return or Enter, MATLAB automatically displays the results on screen. However, if you end the line with a semicolon, MATLAB performs the computation but does not display any output. This is particularly useful when you generate large matrices. For example,

```
>> A = magic(100);
```

2.5.3 Entering Long Statements

If a statement does not fit on one line, use an ellipsis (three periods) ... followed by Return or Enter to indicate that the statement continues on the next line. For example,

```
>> s = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 ...
- 1/8 + 1/9 - 1/10 + 1/11 - 1/12;
```

Blank spaces around the =, +, and - signs are optional, but they improve readability.

3 Programming

3.1 Flow Control

MATLAB has several flow control constructs. They are all known from standard programming languages. In the following only the syntax is described.

3.1.1 if

The `if` statement evaluates a logical expression and executes a group of statements when the expression is `true`. The optional `elseif` and `else` keywords provide for the execution of alternate groups of statements. An `end` keyword, which matches the `if`, terminates the last group of statements. The groups of statements are delineated by the four keywords–no braces or brackets are involved.

```
if A > B
   'greater'
elseif A < B
   'less'
elseif A == B
   'equal'
else
   error('Unexpected situation')
end
```

3.1.2 switch and case

The `switch` statement executes groups of statements based on the value of a variable or expression. The keywords `case` and `otherwise` delineate the groups. Only the first matching case is executed. There must always be an `end` to match the `switch`.

```
switch(value)
   case 0
      M = A
   case 1
      M = B
   otherwise
      M = C
end
```

Unlike the C language `switch` statement, MATLAB `switch` does not fall through. If the first case statement is true, the other case statements do not execute. So, `break` statements are not required.

3.1.3 for

The `for` loop repeats a group of statements a fixed, predetermined number of times. A matching `end` delineates the statements.

```
for i = 3:3:12
   A(i) = 5;
end
```

The result will be a 1-by-12 vector with every third value equals 5.

Remark: MATLAB `for` loop is very slow. Try to avoid it as often as possible. Instead of the example given above, use:

```
A(3:3:12) = 5
```

3.1.4 while

The `while` loop repeats a group of statements an indefinite number of times under control of a logical condition. A matching `end` delineates the statements.

```
while A > B
   A = A - 1;
end
```

3.1.5 continue

The `continue` statement passes control to the next iteration of the `for` or `while` loop in which it appears, skipping any remaining statements in the body of the loop. In nested loops, `continue` passes control to the next iteration of the `for` or `while` loop enclosing it.

3.1.6 break

The `break` statement lets you exit early from a `for` or `while` loop. In nested loops, `break` exits from the innermost loop only.

3.2 M-files

MATLAB is a powerful programming language as well as an interactive computational environment. Files that contain code in the MATLAB language are called M-files. You create M-files using a text editor, then use them as you would any other MATLAB function or command.

There are two kinds of M-files:

- Scripts, which do not accept input arguments or return output arguments. They operate on data in the workspace.
- Functions, which can accept input arguments and return output arguments. Internal variables are local to the function.

3.3 Scripts and Functions

3.3.1 Scripts

When you invoke a script, MATLAB simply executes the commands found in the file. Scripts can operate on existing data in the workspace, or they can create new data on which to operate. Although scripts do not return output arguments, any variables that they create remain in the workspace, to be used in subsequent computations. In addition, scripts can produce graphical output using functions like `plot`. Example script that generates 32 random values with `randn` and displays the values as bar graphic.

```
r = zeros(1,32);
for n = 1:32
   r(n) = randn;
end
r
bar(r)
```

If this code is saved in a file called `barrandn.m`, the script can be started by simply typing `barrandn` in the command line.

3.3.2 Functions

Functions are M-files that can accept input arguments and return output arguments. The name of the M-file and of the function should be the same. Functions operate on variables within their own workspace, separate from the workspace you access at the MATLAB command prompt. Example function `sumup` with one parameter `a` that sums up the values from 1 to `a`.

File `sumup.m`:

```
function r = sumup(a)
% Function that sums up the values from 1 to a
sum = 0;
for i = 1:a
   sum = sum + i;
end
r = sum;
```

Typed in the command line `sumup(5)` produces:

```
>> sumup(5)

ans =
      15
```

Function Handles You can create a handle to any MATLAB function and then use that handle as a means of referencing the function. A function handle

is typically passed in an argument list to other functions, which can then execute, or evaluate, the function using the handle.

Construct a function handle in MATLAB using the at sign, @, before the function name. The following example creates a function handle for the `sumup` function and assigns it to the variable fhandle.

```
>> fhandle = @sumup;
```

Evaluate a function handle using the MATLAB feval function. The function plot_fhandle, shown below, receives a function handle and data, and then performs an evaluation of the function handle on that data using `feval`.

```
function x = plot_fhandle(fhandle,data)
dimension = prod(size(data));
r = data;
for i = 1:prod(size(data))
   r(i) = feval(fhandle,data(i));
end
x = r
plot(data,r,'+')
```

Typing in the MATLAB command line

```
>> v = [ 1 2 3 4 5 ];

>> plot_fhandle(@sumup,v)
```

computes the results of the function `sumup` for the values within `v` and plots them.

Function Functions A class of functions called function functions works with nonlinear functions of a scalar variable. That is, one function works on another function. The function functions include

- Zero finding
- Optimization
- Quadrature
- Ordinary differential equations

MATLAB represents the nonlinear function by a function M-file. For example a simple nonlinear function:

```
function y = humps(x)
y = 1./((x-.3).^2 + .01) + 1./((x-.9).^2 + .04) - 6;
```

Evaluate this function at a set of points in the interval $0 \leq x \leq 1$

```
>> x = 0:0.002:1;

>> y = humps(x);
```

Then plot the function with

```
>> plot(x,y)
```

The graph in Figure 1 shows that the function has a local minimum near x = 0.6. The function `fminsearch` finds the minimizer, the value of x where the function takes on this minimum. The first argument to `fminsearch` is a function handle to the function being minimized and the second argument is a rough guess at the location of the minimum.

```
>> p = fminsearch(@humps,.5)

p =
      0.6370
```

To evaluate the function at the minimizer

```
>> humps(p)

ans =
      11.2528
```

3.4 Setting Paths

In order to have your own files, functions, scripts and data available in the MATLAB environment and on the command line, it is necessary to add your directories to MATLAB path variable. This can be done by opening the *Set Path...* option from the *File* menu.

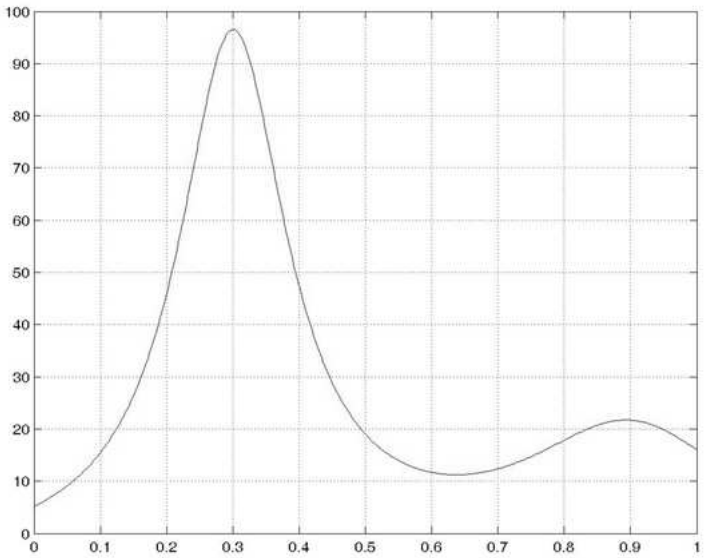

Figure 1: Function humps(x)

4 Graphics

4.1 Images

4.1.1 Loading an image

First of all an image has to be read into the MATLAB memory. This is done by using the command `imread`.

```
>> I = imread('image.jpg');
```

When loading images it is useful to write the semicolon at the end of the command, otherwise the whole image matrix is printed. The image data is saved in the variable I containing a MxN or MxNx3 matrix corresponding the number of channels.

For simplicity in this tutorial we will only work with 1-channel grayscale images. If the loaded image is grayscale but anyhow there are three channels (represented by a NxMx3 matrix) simply take just one of the channels by typing

```
>> I=I(:,:,1);
```

4.1.2 Displaying an image

Images can be displayed with the command

```
>> imshow(I);
```

The `imshow(I,N)` command displays the intensity image `I` with N discrete levels of gray. If N is omitted, `imshow` uses 256 gray levels on 24-bit displays, or 64 gray levels on other systems.
Every matrix of dimension MxN or MxNx3 can be displayed as an image by using the command

```
>> image(I);
```

If `I` is a 2-dimensional MxN matrix, the elements of `I` are used as indices into the current `colormap` to determine the color. For correct visualization of the grayscale image it is necessary to specify the `colormap` by typing

```
>> colormap(gray(256));
```

before displaying the image `I`. When `I` is a 3-dimensional MxNx3 matrix, the elements in `I(:,:,1)` are interpreted as red intensities, in `I(:,:,2)` as green intensities, and in `I(:,:,3)` as blue intensities.

4.1.3 Manipulating image data

Most image data is of data type `uint8` so one byte containing values from 0 to 255. For further image processing and computations it is necessary to convert

the image data. Typing

```
>> I=double(I);
```

converts the matrix of data type `uint8` into data type `double`. Now it is possible to do all kind of computations on the image data. For example

```
>> image(I+50);
```

displays the image `I` where each pixel intensity is increased by 50. Manipulating the image data can be done directly on the matrix `I`. The next example shows how to set a half of the 10th row to the intensity 255.

```
>> for i=1:(size(I,2)/2)
I(10,i) = 255;
end
```

Typing again `image(I)` shows the result. In the last example the command `size(I,2)` is used to determine the number of columns of the matrix or the *width* of the image. Respectively gives `size(I,1)` the number of rows or the *height* of the image.

Remark: Sometimes (often!) it is better to use `imagesc` instead of `image` in order to make the data scaled to the full colormap.

4.2 Data Visualization

MATLAB provides a variety of functions for displaying vector data as line plots, as well as functions for annotating and printing these graphs.

4.2.1 Create Line Plots

The `plot` function has different forms depending on the input arguments. For example, if y is a vector, `plot(y)` produces a linear graph of the elements of `y` versus the index of the elements of `y`. If you specify two vectors as arguments, `plot(x,y)` produces a graph of `y` versus `x`.
For example the developing of the intensities in a single row of our image can be plotted

```
>> plot(I(100,:));
```

The graph shows the intensities varying along line 100. If we plot row 10 with `plot(I(10,:))` we get a constant graph representing the line we have manipulated before.

The command

```
>> grid on;
```

shows an underlying grid within the figure.

4.2.2 Adding Plots to an Existing Graph

You can add plots to an existing graph using the `hold` command. When you set hold to on, MATLAB does not remove the existing graph; it adds the new data to the current graph, rescaling if the new data falls outside the range of the previous axis limits.

```
>> plot(I(100,:));

>> hold on;

>> plot(I(10,:));
```

plots both graphs in one figure. The `hold` command can be turned off by typing `hold off`.

4.2.3 Specifying Data Point Markers, Line Styles and Colors

There are a lot of possibilities to specify the plotted graphs. You can define the colors and line styles. It is also possible to plot only markers at the data points. Here are some examples, for more information please read the MATLAB help.

```
>> plot(I(100,:),'r+');
```

plots a red '+' at each data point.

```
>> plot(I(100,1:20:end),'-rs','LineWidth',2,...
'MarkerEdgeColor','k',...
'MarkerFaceColor','g',...
'MarkerSize',10);}
```

plots a graph (Figure 2) of every 20th pixel of row 100 and marks each data point with a green rectangle.

4.2.4 Setting Axis Parameters

MATLAB selects axis limits based on the range of the plotted data. You can specify the limits manually using the `axis` command. Call `axis` with the new limits defined as a four-element vector.

```
axis([xmin,xmax,ymin,ymax])
```

By default, MATLAB displays graphs in a rectangular axes that has the same aspect ratio as the figure window. This makes optimum use of space available for plotting. MATLAB provides control over the aspect ratio with the `axis` command.
The command

```
axis square
```

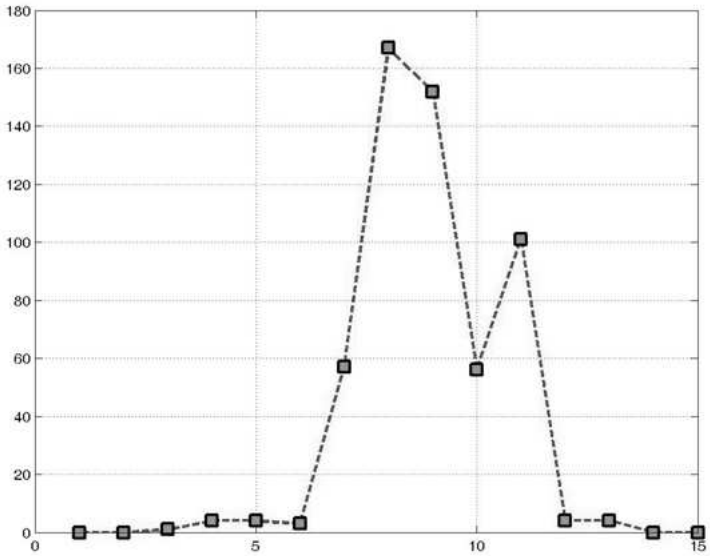

Figure 2: Plot of every 20th pixel of row 100 in Image I

makes the x- and y-axes equal in length. The square axes has one data unit in x to equal two data units in y. If you want the x- and y-data units to be equal, use the command

```
axis equal
```

This produces an axes that is rectangular in shape, but has equal scaling along each axis.
If you want the axes shape to conform to the plotted data, use the `tight` option in conjunction with `equal`.

```
axis equal tight
```

4.2.5 3D Plots

The `plot3` function displays a three-dimensional plot (Figure 3) of a set of data points. Example

```
>> t = 0:pi/50:10*pi;

>> plot3(sin(t),cos(t),t)

>> grid on

>> axis square
```

4.2.6 Mesh Plots

A *heightmap* is a 3D representation of an image where the pixels intensities are used as the height at each position. The visualization looks similar to an area with mountains. In MATLAB such a heightmap can be easily created with

```
>> mesh(I);
```

The `mesh` command creates 3-D surface plot of matrix data. If `I` is a matrix for which the elements I(i,j) define the height of a surface over an underlying (i,j) grid.

4.2.7 Figure Windows

Graphics functions automatically create new figure windows if none currently exist. If a figure already exists, MATLAB uses that window. If multiple figures exist, one is designated as the current figure and is used by MATLAB (this is generally the last figure used or the last figure you clicked the mouse in).

The `figure` function creates figure windows. For example,

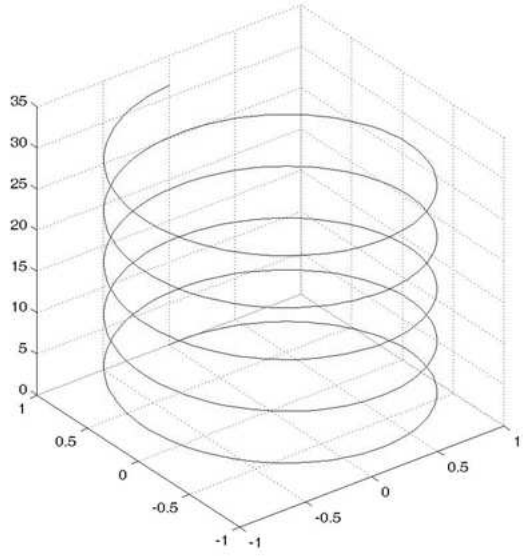

Figure 3: 3D Plot

```
>> figure
```

creates a new window and makes it the current figure. You can make an existing figure current by clicking on it with the mouse or by passing its handle (the number indicated in the window title bar), as an argument to figure.

```
>> figure(h)
```

5 Video editing

5.1 Loading Video Files

A video file of type AVI can be loaded in the MATLAB memory by typing

```
>> MOV=aviread('movie.avi');
```

The i-th frame can be loaded by specifying the frame index

```
>> FRAME=aviread('movie.avi',i);
```

MATLAB provides a data struct for holding movie frames in memory. It consists of two fields. The field `cdata` contains the image data of the frame, `colormap` contains information about the colormap that is used to display the frames. If the frames are truecolor images the field `colormap` will be empty. Typing

```
>> aviinfo('movie.avi')
```

the properties of the movie file will be printed.

5.2 Editing Frames

After loading a whole video sequence, single frames can be accessed by

```
>> FRAME=MOV(i);
```

The raw image data of a frame can be extracted easily by typing

```
>> IMAGE=FRAME.cdata;
```

That allows us to handle movie frames like single images. For example

```
>> for i=5:5:size(MOV,2)
I=MOV(i).cdata;
I(:,:,:)=255;
MOV(i).cdata = I;
end
```

will fill every fifth frame with intensity 255 (white).

5.3 Exporting

To export a MATLAB movie the command `movie2avi` is used.

```
>> movie2avi(MOV,'movie2.avi');
```

saves the movie in the file `movie2.avi` on the harddisk.