**Machine Learning for Computer Vision 2014**

**Iasonas Kokkinos,** **iasonas.kokkinosecp.fr**

# Introduction to Matlab

## 1 Matrices and Arrays

- Data in Matlab are stored in n-dimensional arrays; the comma is used to indicate horizontal concatenation, the semicolon for vertical concatenetation.
```
a = 1; a = 2+1i; % real and complex numbers
b = [1 2 3 4]; % row vector
c = [1; 2; 3; 4]; % column vector
b=c'; %  b is the transpose of c
d = 1:7; % here one has d=[1 2 3 4 5 6 7]
d = 1:2:7; % here one has d=[1 3 5 7]
E = [1 2;3 4]; % a 2x2 matrix
```
- The acess of an array entry, or the selection of a sub-array is done by indexing:
```
size(d) % display the size
d(1) % display the first entry : indexing starts at 1
d(1:2) % display the sub-array containing entries 1 and 2
E(:); % flattens a matrix into a column vector
```
- Matlab provides functions to create pre-defined arrays:
```
% identity, 1 and random matrices
A = eye(4,4);
B = ones(4,4);
C = rand(4,4);
% transpose
c = b';
```
- Matlab uses the character % to write comments:
```
% this is a Matlab comment
```
- Ending an instruction with ; prevents from displaying the result in the command window:
```
a=1 % The result is displayed
a=1; % The result is not displayed
```
- Matlab provides many tools for matrix computing and manipulation. For instance, the multiplication operator `*` is used for matrix multiplication. The pointwise multiplication, i.e. multiplication of the corresponding entries of the matrices is obtained using the operator `.*`. Note that Matlab recognizes either `j` or `i` as the square root of $-1$, unless you have defined variables `j` or `i` with different values:
```
% note the difference
D = C*A % Matrix multiplication
D = C.*A % Pointwise multiplication
% You can apply functions to each entry of a matrix
E = A./C; % division
```

```
E = sin(A); % sinus is applied to each entry
E = abs(A + 1i*C); % modulus of each entry
```

- You can modify matrices and arrays in various ways:
```
b = sort(b); % sort values
b = b .* (b>2); % set to zeros (threshold) the values below 2
b(3) = []; % suppress the 3rd entry of a vector
B = [b; b]; % concatenation: create a matrix of size 2x4
c = B(:,2); % to access 2nd column
```

- It is possible to access directly the last entry of a vector using the keyword `end` in Matlab:
```
b(end-2:end) = 1; % to access the last entries
b = b(end:-1:1); % reverse a vector
```

- Matlab provides many other useful functions as:
```
sum(B(:)); % sum all values in B
min(B(:)); % the smallest value in B
max(B(:)); % the biggest value in B
```
  Type `help elmat` to see a list of such functions.

- To format and display text:
```
disp('Hello'); % display a text
x = 1.23456;
disp(sprintf('Value of x=%.2f',x)); % print a values with 2 digits
```

## 2 Graphics

- To create a plot in Matlab, run for instance:
```
x = 0:pi/100:2*pi;
y = cos(x);
plot(x,y)
```

- You can add labels to the axes and a title to your plot:
```
xlabel('x = 0:2\pi')
ylabel('Cosine of x')
title('Plot of the Cosine Function','FontSize',12)
```

- You can also include several plots in the same figure:
```
x = 0:pi/100:2*pi;
y1 = sin(x);
y2 = cos(x);
y3 = y1+y2;
plot(x,y1,x,y2,x,y3)
```

- Adding a legend is done through:
```
legend('sin(x)','cos(x)','sin(x)+cos(x)')
```

- Matlab enables also to plot complex data. When `Z` is a complex array, `plot(Z)` is equivalent to `plot(real(Z),imag(Z))`. An example below:
```
t = 0:pi/10:2*pi;
plot(exp(i*t),'-o')
axis equal
```

- It is also possible in Matlab to add a plot to an existing graph using the `hold` function:
```
[x,y,z] = peaks; % 3D peaks data
pcolor(x,y,z) % Pseudocolor plot
shading interp %setting the color properties
hold on % enables to add a plot
contour(x,y,z,20,'k') % adding the isolines to the plot
hold off % adding plot ends
```

- It is also possible to display multiple plots in the same figure:
```
x = 0:pi/100:2*pi;
y1 = sin(x);
y2 = cos(x);
subplot(2,2,1); plot(x,y1) % top left
subplot(2,2,4); plot(x,y2) % bottom right
```

- The following functions provide other ways to display plots:
```
t = [0.1 0.2 0.3 0.4];
 x = [1.0 8.0 4.5 9.7];
plot(t,x)
figure, stem(t,x)
figure, stairs(t,x)
fs = 1000;
ts = 0:1/fs:2;
f = 250 + 240*sin(2*pi*ts);
x = sin(2*pi*f.*ts);
strips(x,0.25,fs) % the file strips.m is needed
wavwrite(x',fs,'wave.wav') % writes a .wav audio file.
                           % the first argument must be a column vector.
```

## 3 Programming

- 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.

- *Functions* are M-files that can accept input arguments and return output arguments. The names 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. The syntax is as follows:
```
function [output1,...,outputN] = f(input1,...,inputN)
% This is a comment about the function
          instructions & tratments ...
   output1=...;
   outputN=...;
```

- Conditional control can be performed using the following syntax:
```
if condition 1
   treatment 1
elseif condition i
```

```
   treatment i
else
   treatment n
end
```

- ... or using the following:

```
switch expression
   case value 1
      treatment 1
   case value i
      treatment i
   otherwise
      treatment n
end
```

- Loop control can be performed using the following syntax:

```
for n = 1:N
   treatments ...
end
```

- ... or using the following:

```
while condition
   treatments...
end
```

- Vectorization: one way to make your MATLAB programs run faster is to vectorize the algorithms you use in constructing the programs. Example:
  instead of:

```
x = 0.01;
for k = 1:1001
   y(k) = log10(x);
   x = x + 0.01;
end
```

  use:

```
x = 0.01:0.01:10;
y = log10(x);
```

- Preallocation: you can make your `for` loops go faster by preallocating any vectors or arrays in which output results are stored:

```
A = zeros(10000,1); % Preallocating speeds up the algorithm
for n = 1:10000
    A(n) = n;
end
```

  You can try the code with an without preallocating, and notice its speed by adding the commands `tic` at the beginning and `toc` at the end.