

Signal Processing

Lab 1

Ahmed Besbes *Panagiotis Koutsourakis* *Loic Simon*
Chaohui Wang

November 12, 2008

1 Introduction to Matlab - Basics

1.1 Matrices and Arrays

This first lab gives some basics about Matlab programming. We will use OCTAVE as an alternative to MATLAB because it is an open source and free software, and because it has almost the same syntax as MATLAB. Some of the missing functions in OCTAVE will be provided to you.

- 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`
- In MATLAB, the data is basically stored in n-dimensional arrays:
`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'; % true: 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]`
`d = linspace(1,7,4); % here one has d=[1 3 5 7] too`
`E = [1 2;3 4]; % a 2x2 matrix`
- The access to one entry of an array, 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`
- Particularly, the subscripting can be logical:
`d==5 % a boolean array of the same size of d ...`
`d(d==5) = 3; % replaces the values 5 in d by 3`

```
x = [2.1 1.7 1.6 1.5 NaN 1.9 1.8 1.5 5.1 Inf 1.4 2.2 1.6 1.8];
x = x(isfinite(x)) %get rid of the non finite elements of x
```

- 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 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
```
- 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
```

1.2 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.
```

1.3 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 & treatments ...
    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.

2 Exercises - Discrete Fourier Transform

You will be provided different `.wav` audio files that you can use to test your code. MATLAB function `wavread` and `wavwrite` enable you to read and write audio files from your disk. For more details, refer to the documentation or use the `help` function:

```
help wavread % how is wavread used ?
```

1. Write a function `X=DiscreteFourier(x)` that computes the DFT transform $X = DFT(x)$ of a discrete signal x . You may first implement it using loops, but try then to vectorize your code, and think the function in terms of matrix computing.
2. Let us consider now signals that have $N = 2^p$ points. For such signals, Cooley and Tuckey introduced first in 1965 the Fast Fourier Transform (FFT) algorithm. This recursive algorithm decomposes the problem into two FFTs of two signals of dimension 2^{p-1} . The k^{th} Fourier coefficient of x is :

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot W^{nk} \quad , \quad (1)$$

where $W = \exp(-\frac{2\pi j}{N})$. The previous equation 1 can be written as:

$$X(k) = \sum_{n=0}^{N/2-1} x(2n) \cdot W^{2nk} + \sum_{n=0}^{N/2-1} x(2n+1) \cdot W^{(2n+1)k} \quad (2)$$

$$= \sum_{n=0}^{N/2-1} x(2n) \cdot W^{2nk} + W^k \cdot \left(\sum_{n=0}^{N/2-1} x(2n+1) \cdot W^{2nk} \right) \quad . \quad (3)$$

Thus, $\forall k \in 0, \dots, N/2 - 1$:

$$X(k) = T_k^0 + W^k.T_k^1 \quad (4)$$

$$X(N/2 + k) = T_k^0 - W^k.T_k^1, \quad (5)$$

with $T_k^i = \sum_{n=0}^{N/2-1} x(2n+i).W^{2nk}$ being the FFT of signals of 2^{p-1} points. Write a function `X=FastFourier(x)` that implements the previous algorithm. Test and compare the speed of the two functions to the pre-built `fft` MATLAB function (note that `ifft` is the Inverse Fast Fourier Transform). You can use for this comparison some elementary discrete-time signals like:

- the unit sample, or unit impulse:

$$\delta(n) = \begin{cases} 1, & \text{for } n = 0 \\ 0, & \text{for } n \neq 0 \end{cases} \quad (6)$$

- the unit step:

$$u(n) = \begin{cases} 1, & \text{for } n \geq 0 \\ 0, & \text{for } n < 0 \end{cases} \quad (7)$$

- the unit ramp:

$$u_r(n) = \begin{cases} n, & \text{for } n \geq 0 \\ 0, & \text{for } n < 0 \end{cases} \quad (8)$$

- the unit square:

$$\Pi_N(n) = \begin{cases} 1, & \text{for } |n| \leq N \\ 0, & \text{for } |n| > N \end{cases} \quad (9)$$

3. You are given two `.wav` files `top.wav` and `paint_ball.wav`. By performing manipulations in the spectral domain, produce:
 - (a) A file that contains the “ball paint” sound.
 - (b) A file that contains the “pot” sound.
 - (c) A file that contains the “paint ball” sound, vanishing with reverberation.