ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE
School of Computer and Communication Sciences

Handout 5 Signal Processing for Communications
Homework 3 March 7, 2011, INF 213 - 10:15-12:00

We expect you to have read the matlab primer!

Problem 1 (Starting with Matlab)**.**

Open a terminal, and start Matlab. Fom the main application, go to Window Menu and explore the following items

- Command Window (CW)
- Command History
- Current Folder
- Workspace
- Help
- Editor

Though you can type your commands directly on the CW, you will mostly use the Editor to write either script files or m-functions. Once you have written and saved your file, you can run your script and make calls to m-functions in either other script files or from the CW. Now we create an m-function "MYSQUARE.m" which squares the input elements and call the function from another script file "MYSQUARE_TEST.m".

To open a new file on the Editor, simply type on the CW:

```
edit
```

Create and save the two files as shown below.

```
% MYSQUARE.m
function [out] = MYSQUARE(in)
out = in.^2
```

```
% TEST_SQUARE.m
out_value = MYSQUARE(2)
out_vector = MYSQUARE([1 2 3])
```

Note that for the m-function to work, you have to save the file containing the m-function with the convention "filename = functionname". Run the script and check the results from the CW. You should have already discovered the benefits of using the '.' operator. Finally, you can clear the workspace, and close figures typing:

```
clear;
close all;
```

Problem 2 (Signal Generation & Plotting)**.**

In matlab, a one dimensional signal can be represented as either a row or column vector of N elements. The values can be real or complex valued. You can define a row vector typing directly its values on the CW:

```
[1 2 3 4]
```

Similarly, you can define a column vector typing:

```
[1; 2; 3 + i; 4+i]
```

You can interoperate between these two using the transpose operator for real valued signals:

```
[1 2 3 4]'
```

or in case of complex valued signals the following operator:

```
[1; 2; 3 + i; 4+i].' % This avoids taking conjugate transpose
```

Two useful functions which creates vectors of zeros and ones of the specified size are:

```
zeros(20, 1)  % Column vector of length 20
ones(30, 1)
```

Try generating impulse and unit step functions using them.

For more complex signals, we often start creating a time vector which represents the time instances corresponding to the regularly spaced samples of the signal. In case of unit increments you can simply type:

```
n = [-5:5]'
```

In other cases, you can either specify explicitly the increments:

```
n = [0:0.001:1]' % Increments of 0.001
```

or use the linspace function which generates linearly spaced samples in the desired interval:

```
n = linspace(-5, 5, 100)' % n = 100 samples are generated in [-5, 5]
```

You can now start generating signals using built-in functions such as 'sin', 'cos', 'sinc', 'exp' and visualize them. For that purpose, we investigate two functions 'plot' and 'stem'. Create a script file to run the below code.

```
N = 16; % Length of signal
n = 0:N-1; % Time vector
x = cos((3/16)*2*pi.*n);
figure; % Open a new figure
plot(n, x);
figure;
stem(n, x);

L = 140;
l = 0:L-1;
y= cos((3/20)*2*pi.*l);
figure;
plot(l, y);
figure;
stem(l, y);
```

Which command do you think is more appropriate in each case, and why?

Finally, we can add information on the plot and use the 'subplot' command to draw multiple plots in a single figure. We illustrate this for a complex valued signal where we plot the magnitude and phase components of the signal in the same figure:

```
z = y*(1+i);
m = abs(z); % magnitude vector
p = angle(z) ; % phase vector
figure
subplot(2,1,1), plot(l, m), % Divide area into 2 * 1 subplots
title('Magnitude Plot of z[n]'), xlabel('n'), ylabel('|z[n]|');
subplot(2,1,2), plot(l, p),
title('Phase Plot of z[n]'), xlabel('n'), ylabel('Phi(z[n])');
grid on;
hold on;
```

Problem 3 (Circular convolution).

1) Complete the below code, some lazy coder left unfinished, to implement the N-point circular convolution of two sequences $x[n]$ and $h[n]$ of arbitrary length. First read carefully the written part, and comment on the operations performed next to each '%' symbol (use Help). Then complete the code and test the function.

```
% MYCCONV.m
function [out] = MYCCONV(x, h, N)

L_x = size(x, 1); %
L_h = size(h, 1);
x_m = x;
h_m = h;

if(L_x < N) %
    x_m = [x; zeros(N-L_x, 1)]; %
elseif(L_x > N) %
    q = floor(L_x/N); %
    r = rem(L_x, N); %
    x_m = [x; zeros(N-r, 1)]; %
    x_m = reshape(x_m, N, q+1); %
    x_m = sum(x_m.').'; %
end

if(L_h < N)
    h_m = [h; zeros(N-L_h, 1)];
elseif(L_h > N)
    q = floor(L_h/N);
    r = rem(L_h, N);
    h_m = [h; zeros(N-r, 1)];
    h_m = reshape(h_m, N, q+1);
    h_m = sum(h_m.').';
end

% Implement circular convolution here.
%
%
%
%
```

2) Can you evaluate the linear convolution of $x[n]$ and $h[n]$ using the function 'MYCCONV'? If so, verify your result with the Matlab built-in function 'conv'.

3) Can you also think of a way of obtaining the circular convolution using linear convolution?

Problem 4 (From DFT towards FFT)**.**

1) Write a function "MYDFT.m", which takes a vector as its input and returns its DFT. Use the 'for' command to implement the DFT summation. How many complex operations (additions, multiplications) are performed in this case?

2) Write a function "MYmodDFT.m", which takes a vector as its input and returns its DFT. This time use the butterfly operation once, i.e. divide the signal into two parts, use MYDFT to compute the DFT of each part and then properly mix the output to form the DFT of the original signal. How many complex operations (additions, multiplications) are performed in this case?

3) Write a script file to evaluate performance of these algorithms and Matlab's built-in function 'fft' using the 'tic, toc' command (check help for usage). Does the experimental results match the theoretical expectations in complexity reduction?

Hint: You can create multiple random signals of length N using the 'rand' command:

```
x = rand(N, 1);
```

4) Write a function "MYDFT2.m", which takes a vector as its input and returns its DFT. This time you are not allowed to use the 'for' command. Evaluate also its performance.

Problem 5 (DFT Properties)**.**

1) Let $x[n]$ be a $\{1, -1\}$ valued square pulse of periodicity 4 of length $N = 32$, i.e. (1 1 1 1 -1 -1 -1 -1 1 1 1 1...), and h[n] a signal whose DFT is given by:

$$H[k] = \begin{cases} -j \text{ if } n = 1, \ldots, \dfrac{N}{2} - 1 \\ 0 \text{ if } n = 0, \dfrac{N}{2} \\ +j \text{ if } n = \dfrac{N}{2} + 1, \ldots, N - 1 \end{cases}$$

(i) Find an expression for h[n].

(ii) Write a script file which computes the 32-point circular convolution between $x[n]$ and $h[n]$ both in time domain and frequency domain, i.e start from $x[n]$ and $h[n]$ in each case. Plot all the signals and the resulting convolutions.

Hint: You can use 'ifft' to compute the inverse Fourier transform, 'real' and 'imag' to find the real and imaginary parts of signals.

2) Let a real signal be given as:

$$x[n] = \begin{cases} +1 \text{ if } 0 \leq n \leq 15 \\ -1 \text{ if } 32 \leq n \leq 63 \end{cases}$$

Let $X[k]$ be its 64-point DFT.

(i) Using the DFT symmetry properties you derived in HW2, deduce that the DFT of a real valued signal satisfies $X[k] = X^*[(-k) \mod N]$.

(ii) Plot the magnitude and phase components of $X[k]$, and $X^*[(-k) \mod N)]$. What can you infer from the plots?