ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE
School of Computer and Communication Sciences

Handout 6 Signal Processing for Communications
Solution 3 March 7, 2011, INF213 – 10:15pm-12:00

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

Everything is in the homework handout.

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

As we can see from the two figures, the 'stem' command plots the discrete values of the signal. The 'plot' command provides a "continuous" representation joining the successive values of the signal by a line (linear interpolation).

Hence, the 'stem' plot is useful when we would like to emphasize the discrete nature of the representation. On the other hand, whenever we have enough samples, we can get an accurate representation for a continuous function by using 'plot' (Though you should not forget that the plot only makes sense for discrete values, i.e. for $n = 1, 2, \ldots$).

Problem 3 (Circular convolution)**.**

1) Here is the commented code with the implementation of the circular convolution :

```
function [out] = MYCCONV(x, h, N)
% y[n] = sum x[m] h[(n - m)) mod N]

L_x = size(x, 1); % Get length of the column vector x
L_h = size(h, 1);
x_m = x;
h_m = h;

if(L_x < N) % Check if length of x is shorter than N
    x_m = [x; zeros(N-L_x, 1)]; % Apply zero -padding to make length
    % equal to N
elseif(L_x > N) % Check if length of x is greater than N
    q = floor(L_x/N); % L_x = qN + r
    r = rem(L_x, N);
    x_m = [x; zeros(N-r, 1)]; % Add zeros to make length a multiple of N
    x_m = reshape(x_m, N, q+1); % Reshape the zero padded vector into a
    % matrix such that each block of N components of the vector
    % constitutes a column of the matrix
    x_m = sum(x_m.').'; % Add all the columns of the matrix.
end

if(L_h < N)
    h_m = [h; zeros(N-L_h, 1)];
elseif(L_h > N)
    q = floor(L_h/N); % L_x = qN + r
    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
h_m = flipud(h_m); % Reverse the order of the elements.
for i = 1:N
    H_m(:, i) = circshift(h_m, i); % Circulary shift the flipped vector
    % from 1 to N and put each shifted version into the columns
    % of a matrix H_m
end
out = (x_m.'*H_m)'; % Compute the circular convolution
```

2) We can evaluate the linear convolution using MYCONV by setting the input $N = length(x) + length(h) - 1$.

3) Let $z_{lc}[n]$ and $z_{pc}[n]$ denote the linear and circular convolutions of $x[n]$ and $h[n]$ respectively, i.e.

$$z_{lc}[n] = \sum_{m=-\infty}^{\infty} x[m]h[n-m]$$
$$z_{pc}[n] = \sum_{m=0}^{N-1} x[m]h[(n-m)\text{mod}N]$$

Then one can show that

$$z_{pc}[n] = \begin{cases} \sum_{r=-\infty}^{\infty} z_{lc}[n-rN] & \text{if } 0 \le n \le N-1 \\ 0 & \text{otherwise} \end{cases}$$

Therefore, the circular convolution of two finite length sequences corresponds to their linear convolution with time aliasing. See "Discrete-Time Signal Processing" Chapter 8 for more details.

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

1) To compute all N DFT coefficients, N^2 complex multiplications and $N(N-1)$ complex additions are required in this case.

Here is the code for MYDFT.m :

```
function out_vector=MYDFT(in_vector)
%---------------------------------
% Computation of DFT using for.
%---------------------------------
L = length(in_vector);
for k=1:L
out_vector(k) = 0;
for j=1:L
temp = in_vector(j) * exp(-i*2*pi*(j-1)*(k-1)/L);
```

```
out_vector(k) = out_vector(k) + temp;
end
end
```

2) To compute all N DFT coefficients using the butterfly operation requires at most $N + 2(N/2)^2$ complex multiplications and approximately the same amount of complex additions.

Here is the code for MYmodDFT.m :

```
function out_vector=MYmodDFT(in_vector)
%-----------------------------------
% Computation of DFT using butterfly method.
%-----------------------------------
L=length(in_vector);
% add a zero entry if the length of in vector is odd
if (mod(L,2)≠0)
input=zeros(1,L+1);
input(1:end-1)=in_vector;
in vector=input;
L=L+1;
end
% separate odd and even parts
in_even=in_vector(1:2:end-1);
in_odd=in_vector(2:2:end);
% take DFT from both parts
out_even=MYDFT(in_even);
out_odd=MYDFT(in_odd);
% compute out-vector considering correct weights
W=exp(-i*2*pi/L);
weights=W.^(0:(L/2)-1); % half of the weights are sufficient.
out_vector_1 = out_even + weights .* out_odd;
out_vector_2 = out_even + W^(L/2)* weights .* out_odd;
out_vector=[out_vector_1, out_vector_2];
%-----------------------------------
```

3) Here is a script to evaluate performance :

```
close all;
clc;
N= 9:13 ;
t=zeros(3,5);
for i = 9:13
x=rand(2^i,1);
tic;
X=fft(x);
t(1,i-8) = toc
tic;
X=MYmodDFT(x);
t(2,i-8) = toc
tic;
X=MYDFT(x);
t(3,i-8) = toc
end
hold on;
```

```
plot(t(1,:),'r')
plot(t(2,:),'b')
plot(t(3,:),'g')
```

4) The DFT without 'for' loops :

```
function out_vector=MYDFT2(in_vector)

N = length(in_vector);
n = 0:N-1;
k = 0:N-1;
W = exp(-j*2*pi/N*k'*n);
out_vector = in_vector'*W;
```

Problem 5 (DFT Properties).

1) (i) Let us compute $h[n]$:

$$\begin{aligned} h[n] &= \frac{1}{N}\sum_{k=0}^{N-1} H[k]e^{j\frac{2\pi}{N}kn} \\ &= \frac{1}{N}\sum_{k=0}^{\frac{N}{2}-1}(-j)e^{j\frac{2\pi}{N}kn} + \frac{1}{N}\sum_{k=\frac{N}{2}+1}^{N-1}(j)e^{j\frac{2\pi}{N}kn} \\ &= \frac{2}{N}\sum_{k=1}^{\frac{N}{2}-1}\sin\left(\frac{2\pi}{N}kn\right) \end{aligned}$$

(ii) Here is the complete code :

```
N = 32;
n = (0:N-1)';
k = (1:N/2-1)';
% H = [0; -1i*ones(N/2-1, 1); 0; 1i*ones(N/2-1, 1)];
h = sum(2/N*sin(2*pi/N*k*n'))';
H = fft(h);
m_H = abs(H);
p_H = angle(H);
x = [ones(N/8, 1); -ones(N/8, 1); ones(N/8, 1); -ones(N/8, 1);
    ones(N/8, 1);  -ones(N/8, 1); ones(N/8, 1); -ones(N/8, 1)]
X = fft(x, N);
y = cconv(x, h, N);
Y = X.*H;
y_2 = ifft(Y);

figure,
subplot(2,2,1), stem(n, real(x)),
title('Real Part of x[n]'), xlabel('n'), ylabel('Re{x[n]}');
subplot(2,2,2), stem(n, imag(x)),
title('Imaginary Part of x[n]'), xlabel('n'), ylabel('Im{x[n]} ');
subplot(2,2,3), stem(n, real(X)),
```

```
title('Real Part of X[k]'), xlabel('k'), ylabel('Re{X[k]}');
subplot(2,2,4), stem(n, imag(X)),
title('Imaginary Part of X[k]'), xlabel('k'), ylabel('Im{X[k]}');


figure,
subplot(2,2,1), stem(n, real(h)),
title('Real Part of h[n]'), xlabel('n'), ylabel('Re{h[n]}');
subplot(2,2,2), stem(n, imag(h)),
title('Imaginary Part of h[n]'), xlabel('n'), ylabel('Im{h[n]}');
subplot(2,2,3), stem(n, real(H)),
title('Real Part of H[k]'), xlabel('k'), ylabel('Re{H[k]}');
subplot(2,2,4), stem(n, imag(H)),
title('Imaginary Part of H[k]'), xlabel('k'), ylabel('Im{H[k]}');


figure,
subplot(3,2,1), stem(n, real(Y)),
title('Real Part of Y[k]'), xlabel('k'), ylabel('Re{Y[k]}');
subplot(3,2,2), stem(n, imag(Y)),
title('Imaginary Part of Y[k]'), xlabel('k'), ylabel('Im{Y[k]}');
subplot(3,2,3), stem(n, real(y)),
title('Real Part of y[n] computed using circular convolution'),
    xlabel('n'), ylabel('Re{y[n]}');
subplot(3,2,4), stem(n, imag(y)),
title('Imaginary Part of y[n] computed using circular convolution'),
    xlabel('n'), ylabel('Im{y[n]}');
subplot(3,2,5), stem(n, real(y_2)),
title('Real Part of y[n] computed using FFT'), xlabel('n'),
    ylabel('Re{y[n]}');
subplot(3,2,6), stem(n, imag(y_2)),
title('Imaginary Part of y[n] computed using FFT'), xlabel('n'),
    ylabel('Im{y[n]}');
```

2) (i) In the previous homework, you proved that $x^*[n] \xrightarrow{\text{DFT}} X^*[(-k)modN]$. Since $x[n]$ is real valued, we have $x^*[n] = x[n]$. Hence $X[k] = X^*[(-k)modN]$.

(ii) We observe that $|X[k]| = |X[(-k)modN]|$ and $\angle\{X[k]\} = -\angle\{X[(-k)modN]\}$

This can be seen using the following code :

```
N = 64;
n = (0:N-1)';
x = [-1*ones(N/2, 1) ; ones(N/2, 1)];
X = fft(x, N);
m = abs(X);
p = angle(X);
X_s = flipud(X');
m_s = abs(X_s);
p_s = angle(X_s);

figure,
subplot(4,1,1), stem(n, m),
title('Magnitude Plot of X[k]'), xlabel('k'), ylabel('|X[k]|');
subplot(4,1,2), stem(n, m_s),
title('Magnitude Plot of X[(-k) mod(N)]'), xlabel('n'),
    ylabel('|X[(-k) mod(N)]|');
```

```
subplot(4,1,3), stem(n, p),
title('Phase Plot of X[k]'), xlabel('k'), ylabel('Phi{X[k]}');
subplot(4,1,4), stem(n, p_s),
title('Phase Plot of X[(-k) mod(N)]'), xlabel('k'),
     ylabel('Phi{X[(-k) mod(N)]}');
grid on;
```