Digital Signal Processing
Lab 2: Discrete Time Systems

Downsampling

Taking one sample every M samples of a given sequence is an operation called **decimation of a factor M**. In practice it reduces the sampling frequency of a factor M (downsampling).

1) Consider the sequence $x[n] = \cos(0.1\pi n)$ for $-30 \leq n \leq 30$. Using the stem function plot
   (a) x[n] versus n.
   (b) A downsampled signal y[n] for M = 5.
   (c) A downsampled signal y[n] for M = 20.
   (d) How does the downsampled signal appear? Does it still represent correctly a sinusoidal sequence?

```
close all; clc
nx = -30:30;
x = cos(0.1*pi*nx);
M = 5; % Part (b)
%M = 20; % Part (c)
yind = mod(nx,M)==0;
y = x(yind);
ny = nx(yind)/M;

subplot(211)
stem(nx,x,'fill')
xlabel('nx'); ylabel('x[n]');
title('x[n]');
subplot(212)
stem(ny,y, 'fill')
xlabel('ny'); ylabel('y[n]');
title('Downsampling y[n]= x[nM]');
```

Before applying decimation, one should be sure that the final sampling frequency is high enough to avoid the phenomenon of aliasing (according to Shannon's sampling theorem), otherwise a prefilter should be added before decimation, in order to limit the bandwidth of the signal.
In the following example this anti-aliasing prefilter is omitted for simplicity, leading to a certain amount of acoustic distortion due to aliasing.

2) This problem uses the sound file "handel" available in MATLAB. This sound is sampled at Fs=8192 samples per second using 8-bits per sample.

(a) Load the sound waveform “handel” in an array x and listen to it using the sound function at the full sampling rate.
(b) Select every other sample in x which reduces the sampling rate by a factor of two. Now listen to the new sound array using the sound function at half the sampling rate.
(c) Select every fourth sample in x which reduces the sampling rate by a factor of four. Listen to the resulting sound array using the sound function at quarter the sampling rate.
(d) Save the generated sound in part (c) using the wavwrite function.

```
% Investigates the effect of downsampling using  audio file ‘handel’
close all; clc
load('handel.mat')
n = 1:length(y);
% Part (a): original sampling rate
sound(y,Fs); pause(1)
% Part (b): downsampling by a factor of two
y_ds2_ind = mod(n,2)==1;
sound(y(y_ds2_ind),Fs/2); pause(1)
% Part (c): downsampling by a factor of four
y_ds4_ind = mod(n,4)==1;
sound(y(y_ds4_ind),Fs/4)
% save the sound file
wavwrite(y(y_ds4_ind),Fs/4, 'handel_ds4')
```

Convolution

Matlab provides a built-in function called `conv` that computes the convolution between two finite-duration sequences `x` and `h`. The `conv` function assumes that the two sequences begin at *n=0* and is invoked by

```
y=conv(x,h);
```

The `conv` function neither provides nor accepts any timing information specifying the support of the sequences. We define a specifically modified function called `conv_m`, which performs the convolution of sequences with arbitrary support.

```
function [y,ny] = conv_m(x,nx,h,nh)
% Modified convolution routine for signal processing
% ---------------------------------------------------
% [y,ny] = conv_m(x,nx,h,nh)
% [y,ny] = convolution result
% [x,nx] = first signal
% [h,nh] = second signal
```

```
%
nyb = nx(1)+nh(1); nye = nx(length(x)) + nh(length(h));
ny = [nyb:nye];
y = conv(x,h);
```

Crosscorrelation

Given two real-valued sequences x[n] and y[n] of finite energy, the crosscorrelation of x[n] and y[n] is a sequence r_{xy} [l] defined as:

$r_{xy}[l] = \sum_{n=-\infty}^{\infty} x[n]\, y[n-l]$

which is used to determine the similarity of x[n] and y[n] depending on a lag l. By comparing this definition with the formula of the convolution, we find that the crosscorrelation can be also expressed as:

$r_{xy}[l] = x[l] * y[-l]$

Therefore the crosscorrelation can be computed using the `conv_m` function if sequences are of finite duration.

Difference equations

An LTI discrete system can be described by a **linear constant coefficient difference equation** of the form

$$\sum_{k=0}^{N} a_k y[n-k] = \sum_{m=0}^{M} b_m x[n-m], \quad \forall n$$

A Matlab function called `filter` is available to solve difference equations numerically, given the input x[n] and the difference equation coefficients. In its simplest form this function is used as follows

```
b = [b0, b1, ..., bM]; a = [a0, a1, ..., aN];
y = filter(b,a,x)
```

There are two important observations concerning the function `y=filter(b,a,x)`:

-First, the feedback coefficients enter in the parameter vector a = [1 a1 . . . aN] with their sign reversed. This is because MATLAB assumes that all feedback terms are on the left hand side as the above equation.

-Second, the output sequence is computed at the same time interval as the input sequence.

To compute and plot the impulse response of the corresponding system, Matlab provides the function `impz`:

```
h = impz(b,a,n);
```

it computes samples of the impulse response at the sample indices given in `n`. When no output arguments are given, the `impz` function plots the response in the current figure window using the `stem` function.

Examples

1. The convolution between the sequences x[n] and h[n] defined as follows

```
x = [3, 11, 7, 0, -1, 4, 2]; nx = [-3:3];
h = [2, 3, 0, -5, 2, 1]; nh = [-1:4];
```

is computed by

```
[y,ny] = conv_m(x,nx,h,nh)
```

2. Given the following difference equation

$$y[n] - y[n-1] + 0.9y[n-2] = x[n] + x[n-1]; \quad \forall n$$

A. Calculate and plot the impulse response h[n] at n = −20, . . . , 100.
B. Calculate and plot the unit step response s[n] at n = −20, . . . , 100.
C. Is the system specified by h[n] stable?

```
%A.
n = [-20:100];
a = [1, -1, 0.9];
b = [1 1];
h = impz(b,a,n);
subplot(2,1,1); stem(n,h);
title('Impulse Response'); xlabel('n'); ylabel('h(n)')
%B.
x = stepseq(0,-20,100);
s = filter(b,a,x);
subplot(2,1,2); stem(n,s)
title('Step Response'); xlabel('n'); ylabel('s(n)')
```

```
%C
%compare partial sums till n=100, 1000, 10000 …
h=impz(b,a,[-20:1000]);
sum(abs(h))
h=impz(b,a,[-20:10000]);
sum(abs(h))
```

3. **Echo generation and reverberation**: When music is performed in a concert hall, a torrent of echoes from the various surfaces in the room strikes the ear, producing the impression of space to the listener. More specifically, the sound reaching the listener consists of several components: direct sound, early reflections, and reverberations. The early reflections correspond to the first few reflections off the wall, whereas the reverberation is composed of densely packed late reflections. Music recorded in an almost anechoic studio, using microphones placed close to the instruments, and played at home or in a car does

not sound natural. The typical solution to this problem is to create and add some amount of artificial reverberation to the original recording before distribution.
A single echo is easily generated using the FIR filter:
$y[n] = x[n] + ax[n - D], -1 < a < 1$
where x[n] is the original signal, D is the round-trip delay in number of sampling intervals, and a is the attenuation factor due to propagation and reflection. If the delay $\tau = D/Fs$ is greater than approximately 40 ms, an echo will be heard. A second echo will be given by $a^2x[n-2D]$, a third by $a^3x[n-3D]$, and so on. Therefore, a multiple echo generating FIR filter is given by
$y[n] = x[n] + ax[n - D] + a^2x[n - 2D] + a^3x[n - 3D] + \cdots$
which has impulse response
$h[n] = \delta[n] + a\delta[n - D] + a^2\delta[n - 2D] + a^3\delta[n - 3D] + \cdots$
This filter generates an infinite sequence of echoes having exponentially decaying amplitudes and spaced D sampling periods apart. A more efficient recursive implementation is
given by:
$y[n] = ay[n - D] + x[n]. - 1 < a < 1$
The condition $-1 < a < 1$ assures the stability of the system. Such simple filters provide the basic building blocks of more sophisticated digital reverberators.

Examples on implementation of reverberation:

3.1: A recursive implementation of reverberation is given below:
$y[n] = x[n] + ay[n - D]$,
where $D = \tau Fs$ is the delay in sampling interval given the delay τ in seconds and sampling rate Fs and a is an attenuation factor. To generate digital reverberation we will use the sound file handel which is recorded at Fs = 8192 samples per second.
(a) For $\tau = 50$ ms and a = 0.7, obtain a difference equation for the digital reverberation and process the sound in handel. Comment on its audio quality.
(b) Repeat (a) for $\tau = 100$ ms.
(c) Repeat (a) for $\tau = 500$ ms.
(d) Which implementation sounds natural?

```
close all; clc
load('handel.mat')
n = 1:length(y);
a = 0.7; % specify attenuation factor
tau = 50e-3; % Part (a)
% tau = 100e-3; % Part (b)
%tau = 500e-3; % Part (c)
D = floor(tau*Fs); % compute delay
yd = filter(1,[1 zeros(1,D-1),-a],y);
sound(yd,Fs);
```

3.2. Convolution using a recorded impulse response to simulate reverberation

```
% Room reverb simulation
[h,sr] = audioread('hlwy16.wav');  % Recording of clap in hallway
soundsc(h,sr)
[d,sr] = audioread('mpgr1_sx419.wav'); % Speech example
soundsc(d,sr)
% Convolve them (SLOW because many thousands of points)
y=(conv(d,h));
% Listen to simulated reverberation
soundsc(y,sr)
```

4. In this example we will demonstrate one application of the cross-correlation sequence. Let x[n] = [3, 11, 7,0,−1, 4, 2] be a prototype sequence, and let y[n] be its noise-corrupted-and-shifted version: y[n] = x[n – 2] + w[n]
where w[n] is a Gaussian sequence with mean 0 and variance 1. Compute the crosscorrelation between y[n] and x[n].

```
x = [3, 11, 7, 0, -1, 4, 2]; nx=[-3:3]; % given signal x[n]
subplot(311); stem(nx,x);title('x[n]');
[y,ny] = sigshift(x,nx,2); % obtain x[n-2]
w = randn(1,length(y)); nw = ny; % generate w[n]
[y,ny] = sigadd(y,ny,w,nw); % obtain y[n] = x[n-2] + w[n]
subplot(312); stem(ny,y);title('y[n]');

[x,nx] = sigfold(x,nx); % obtain x[-n]
[rxy,nrxy] = conv_m(y,ny,x,nx); % crosscorrelation
subplot(3,1,3);stem(nrxy,rxy)
xlabel('lag variable l')
ylabel('rxy');title('Crosscorrelation between x and y')
```

We observe that the crosscorrelation indeed peaks at l= 2, which implies that y[n] is similar to x[n] shifted by 2. This approach can be used in applications like radar signal processing in identifying and localizing targets.
Note that the signal-processing toolbox in MATLAB also provides a function called **xcorr** for sequence correlation computations. In its simplest form:
>> xcorr(x,y)
computes the crosscorrelation between vectors x and y, while
>> xcorr(x)
computes the autocorrelation of vector x. It generates results that are identical to the one obtained from the proper use of the `conv_m` function. However, the `xcorr` function cannot provide the timing (or lag) information (as done by the `conv_m` function), which then must be obtained by some other means.

5. Write a MATLAB code fragment to compute and plot the output of the discrete-time system. y[n] = 5y[n − 1] + x[n], y[−1] = 0, for x[n] = u[n], 0 ≤ n ≤ 1000. Based on these results can you make a statement regarding the stability of the system? Hint: What happens when n increases, e.g check the value y[600].

```
close all; clc
n = 0:1e3;
x = ones(1,length(n));
y = filter(1,[1 -5],x);
% Plot:
stem(n,y,'fill')
xlabel('n');
title('y[n] = 5y[n-1]+x[n], y[-1]=0')
```

6. A system is implemented by the statements
y1=conv(ones(1,5),x);
y2=conv([1 -1 -1 -1 1],x);
y=conv(ones(1,3),y1+y2);
(a) Determine the impulse response of the equivalent system y=conv(h,x).
(b) Compute and compare the step responses of the two equivalent system representations.

```
close all; clc
n1 = 0:4;
h1 = ones(1,5);
h2 = [1 -1 -1 -1 1];
n2 = 0:2;
h3 = ones(1,3);
[h n] = conv_m(h1+h2,n1,h3,n2);
un = ones(1,5);
[ytemp1 nyt] = conv_m(h1,n1,un,n1);
[ytemp2 nyt]= conv_m(h2,n1,un,n1);
[y1 ny1] = conv_m(h3,n2,ytemp1+ytemp2,nyt);
[y2 ny2] = conv_m(h,n,un,n1);
%Plot
subplot(2,2,1)
stem(n,h, 'fill')
xlabel('n');
ylabel('h[n]');
title('System h[n]');
subplot(2,2,2)
stem(n1,un,'fill')
xlabel('n');
ylabel('u[n]');
title('Unit Step u[n]');
subplot(2,2,3)
```

```
stem(ny1,y1, 'fill')
xlabel('n');
ylabel('y_1[n] ');
title('Step Response of System I');
subplot(2,2,4)
stem(ny2,y2, 'fill')
xlabel('n'); ylabel('y_2[n]');
title('Step Response of System II');
```

7. Consider the system $y[n] = y[n-1] + y[n-2] + x[n]$, $y[-1] = y[-2] = 0$.
(a) Compute and plot the impulse response, for $0 \le n \le 100$, using the function filter.
(b) Can you draw any conclusions about the stability of this system from the results in (a)?

```
close all; clc
% Part (a):
n = 0:100;
[x n]=impseq(n(1),n(1),n(end));
y = filter(1,[1 -1 -1],x);
% Plot:
stem(n,y, 'fill')
xlabel('n'); ylabel('y[n] ')
title('LCCDE: y[n] = y[n-1] + y[n-2] + x[n], y[-1] = y[-2] = 0')
```

8. Given the system specified by
$y[n] = 1.15y[n-1] - 1.5y[n-2] + 0.7y[n-3] - 0.25y[n-4] + 0.18x[n] + 0.1x[n-1] + 0.3x[n-2] + 0.1x[n-3] + 0.18x[n-4]$.

(a)Compute and plot the impulse response h[n], $0 \le n \le 100$ using the function h=impz(b,a,N).
(b)Compute and plot the output y[n], if x[n] = u[n], $0 \le n \le 100$ using the function y=filter(b,a,x).
(c)Compute and plot the output y[n], if x[n] = u[n], $0 \le n \le 100$ using the function y=conv(h,x).
(d)Compute and plot the output y[n], if x[n] = u[n], $0 \le n \le 100$ using the function y=filter(h,1,x).

```
close all; clc
n = 0:100;
b = [0.18 0.1 0.3 0.1 0.18];
a = [1 -1.15 1.5 -0.7 0.25];
% Part (a):
h = impz(b,a,length(n));
% Part (b):
%u = unitstep(n(1),0,n(end));
[u n]=stepseq(n(1),(n1),n(end));
u=double(u);  %convert from logical to double
y1 = filter(b,a,u);
% Part (c):
```

```
y2 = conv(h,u);
% Part (d):
y3 = filter(h,1,u);
%Plot
subplot(221)
stem(n,h, 'fill')
xlabel('n'); ylabel('h[n] ');
title('Impulse Response h[n] ');
subplot(222)
stem(n,y1, 'fill')
xlabel('n'); ylabel('y_1[n] ');
title('Unit Step Response: filter(b,a,x)');
subplot(223)
stem(0:2*n(end),y2,'fill')
xlabel('n'); ylabel('y_2[n]');
title('Unit Step Response: conv(h,x)');
subplot(224)
stem(n,y3,'fill')
xlabel('n'); ylabel('y_3[n]');
title('Unit Step Response: filter(h,1,x)');
```

9. Consider the finite duration sequences x[n] = u[n] − u[n − N] and h[n] = n(u[n] − u[n − M]), M ≤ N. Assume N = 10 and M = 5 and estimate the convolution y[n] between x[n] and h[n].

```
N = 10; M = 5;
n = 0:N-1;
x=ones(1,length(n));
h=n(1:M).*ones(1,M);
[y ny] = conv_m(h,0:M-1,x,n);
% Plot:
stem(ny,y, 'fill')
axis([ny(1)-1,ny(end)+1,min(y)-1,max(y)+1])
xlabel('n'); ylabel('y[n]');
title('y[n] = h[n]*x[n]')
```

10. A system is described by the difference equation y[n] = x[n] − 0.9y[n − 1], x[n] = u[n] – u[n – 10] What is the output of the system?

```
b = [1]; a = [1,0.9];
n = -5:50; x = stepseq(0,-5,50) - stepseq(10,-5,50);
y = filter(b,a,x);
stem(n,y); title('Output sequence')
xlabel('n'); ylabel('y(n)'); axis([-5,50,-0.5,8])
```