ELEN 3084: Signals and Systems Lab
Lab IV: Sampling

1 Introduction

The theory of sampling is the mathematical interface between our analog world and the discrete world of digital computers. The purpose of this lab is to expose us to the practical side of this theory by presenting some simple examples in MATLAB.

In the first part of the lab we are going to use elementary signals in order to recreate results similar to the ones that appear in section *5.1 The Sampling Theorem* of the Lathi course textbook. This way we will become familiar with the notions of perfect reconstruction and aliasing.

In the second part of the lab we will use a real music sample and we will experiment with different sampling frequencies. We will then empirically evaluate the effect of aliasing.

Before we begin we will start again the diary function which will capture all our commands. Go the `lab4` directory using the `cd` command the same way as in UNIX. Then type:

```
>> diary on;
```

Note that you should reproduce all the examples that will be presented below. By receiving your diary file at the end of the lab session we will have a simple way to see that you successfully completed the lab. Don't forget to turn off the diary function at the end of the lab.

2 Sampling of elementary signals

In this first part of the lab we will focus on a couple of simple examples. Assuming that one has already been exposed to the basic theory of sampling the execution of this part of the lab is going to help us digest the concepts of sampling, reconstruction and aliasing.

We will now construct a sinusoid and we will sample it using various sampling rates. In order to visualize our signals we will also make use of some of the functions we wrote in lab 3.

To Do 1

Generate an `analog` sinusoid and sample it. Plot the signals involved, show the LA your figures and save the plots in .eps format. Notice that our initial signal `m` which we create using the provided function `makecos()` is already sampled but at a very high sampling rate.

We consider this a *virtually* analog signal since we don't want to use symbolic variables in order to create continuous-time signals.

```
% Make our initial, "analog" signal
% (frequency 20 Hz)
[m,t] = makecos(20);
```

Now we will sample this signal with an adequate sampling frequency. We know that our initial signal is a cosine with frequency 20 Hz. This means that the nyquist rate for this signal is 2*20=40Hz. In the first sampling scenario we use a sampling rate of 50Hz which is more than enough for perfect reconstruction. This is the case of oversampling (p321). So we have:

```
% Let's make an impulse train for sampling our signal
% (sampling frequency 50 Hz)
[it1,ts1] = makeimp(50);

% Now sample our original signal
ms1 = sampleit1(t,m,ts1);

% Plot all the signals to visualize the sampling process
c1 = 'r'; % color for the first case
smpl_plot(t,m,ts1,it1,ms1,c1);
```

Notice that the discrete plots featuring arrows denote the integral of the dirac delta function. Now we will sample the same *analog* signal but with a sampling rate less than the nyquist rate. This is the case of undersampling (p321) and we choose 30Hz.

```
% Now make a second impulse train
% (sampling frequency 30 Hz less than nyquist)
[it2,ts2] = makeimp(30);

% Now sample our original signal with the new sampling rate
ms2 = sampleit1(t,m,ts2);

% Plot all the signals to visualize the sampling process
c2 = 'g'; % color for the second case
smpl_plot(t,m,ts2,it2,ms2,c2);
```

According to the theory using the first sampled signal `ms1` we should be able to reconstruct the original whereas using the second `ms2` aliasing should occur. Let's now reconstruct using the function `interpsinc()` that we provide you with and plot the waveforms in both time and frequency in order to verify the sampling theory.

```
% Now we reconstruct from the two sampled versions
mr1 = interpsinc(ms1,ts1,t);
mr2 = interpsinc(ms2,ts2,t);

% Plot original and reconstructed to compare
recon_plot(t,m,ts1,ms1,mr1,c1);
recon_plot(t,m,ts2,ms2,mr2,c2);
```

Notice how the first reconstructed cosine has the same frequency as the original while the second has a different frequency. Clearly in the second reconstructed signal we don't have perfect reconstruction. Following the theory in pages 326-327 we can calculate that the new, *folded* frequency is 10Hz. This is because the original frequency of 20Hz is being *folded* over $30/2 = 15$ Hz.

Since we are examining frequencies we can directly evaluate our calculations in the frequency domain.

```
% Now lets look at the spectrum to examine the aliasing
% Make the frequency index for plotting
f = (-5000/2):(1/2):(5000/2);

% Use the functions we wrote in lab3 to calculate spectra
M   = am_spectrum(m);
MR1 = am_spectrum(mr1);
MR2 = am_spectrum(mr2);

% Plot the spectra to compare
am_plot(f,M,MR1,MR2,0.02);
```

Verify the frequency of the original and the two reconstructed signals, notice the aliasing.

3 Sampling of a musical signal

In this section we will use a music sample and we will examine the side-effects of aliasing acoustically. In order to do that we will compare two cases, sampling with and without an antialiasing filter.

To Do 2

Evaluate empirically the effect of antialiasing filter on the quality of the resampled sound. We will start by loading the music sample and looking at its spectrum.

```
% Load the music sample and its sampling frequency
```

```
[m,fs] = wavread('neneh32.wav');
% Select a small part of it
m = m(110000:230000);

% Listen to it
soundsc(m,fs);

% Look at the time signal
time_plot(m,fs);

% Look at the spectrum
figure;
subplot(311);
freq_plot(m,fs);
```

As we see in the plot, the spectrum extends up to 16kHz. This is to be expected since the sampling rate `fs` of the original signal is 32kHz. Note that once again we consider this as a *virtually analog* signal since there is no way to represent a *true analog* signal in MATLAB. If we now want to sample this signal with anything less than 32kHz we are bound to introduce aliasing. So lets use one eighth of the original sampling frequency that is 32/8=4kHz. This means that we will be able to represent frequencies only up to 2kHz. Indeed:

```
% Now resample it without antialiasing filter
%(sampling frequency is 4000 Hz, aliasing occurs)
newfs = 4000;
md1   = sampleit2(m,fs,newfs,0);

% Listen to it
soundsc(md1,newfs);

% Look at the spectrum
subplot(312);
freq_plot(md1,fs,newfs);
```

We notice that the signal now sounds *dull*, it lost its *brightness* since the high frequencies are gone. Actually those high frequencies are folded because of aliasing thus creating distortion. Now lets go over the same procedure but by using a lowpass antialiasing filter as described in page 327 of the lathi book.

```
% Lets try to resample it with an antialiasing filter
md2   = sampleit2(m,fs,newfs,1);
```

```
% Listen to it
soundsc(md2,newfs);

% Look at the spectrum
subplot(313);
freq_plot(md2,fs,newfs);
```

You should try to do an A/B listening comparison between the signals `md1` and `md2` preferably using headphones. You should be able to hear the difference. In fact we can listen to the difference with the following simple command:

```
soundsc(md1 - md2,newfs);
```

We can also notice this difference in the comparison of the three spectra. In the case of no antialiasing filter there is more energy on the 1-2kHz band (you can zoom in to see that) compared to the case with antialiasing filter. This is because the antialiasing filter prevents frequencies from being folded over.

Now a last, fun experiment. Try to change the playback rate of the original sound. You can do that by typing the command:

```
soundsc(m,fs/2);
```

for half the speed and the command:

```
soundsc(m,fs*2);
```

for double speed. This is a *practical* application of the scaling property of the fourier transform 4.3-3/p255.

4 Lab Problems

Following the proceedure described in the first part of the lab check for aliasing and calculate the frequency of the reconstructed cosines in the following cases:

- Cosine: 30Hz, Sampling: 50Hz
- Cosine: 40Hz, Sampling: 15Hz
- Cosine: 10Hz, Sampling: 50Hz
- Cosine: 20Hz, Sampling: 40Hz

5 Appendix

Here you can find the code for all the functions that we provide in this lab. You are free to experiment with them by opening them in the MATLAB editor.

```
function [m,t] = makecos(f,len)
% MAKECOS Makes a 'virtually' analog cosine
%   [m,t] = makecos(f,len)
%
%   f:   frequency of the cosine in hertz
%   len: length in seconds
%   m:   the cosine signal
%   t:   the time vector on which m is defined

% Default value
if nargin < 2; len = 2; end

sr  = 5000; % ignore this
t   = ((-len/2):1/sr:(len/2)).'; % time index
m   = cos(2*pi*f*t); % our signal
```

```
function [it,ts] = makeimp(fs,len)
% MAKEIMP Makes an impulse train for sampling
%   [it,ts] = makeimp(fs,len)
%
%   fs:  frequency of the impulse train (sampling frequency)
%   len: length in seconds
%   it:  the impulse train
%   ts:  the time vector of the impulses

% Default value
if nargin < 2; len = 2; end

ts  = ((-len/2):1/fs:(len/2)).'; % sampling times
it  = ones(size(ts)); % impulse train
```

```
function y = interpsinc(ms,ts,t)
```

```
% INTERPSINC Reconstructs a sampled signal using sinc interpolation
%   y = interpsinc(ms,ts,t)
%
%   ms: sampled signal
%   ts: sample times
%   t:  times at which to interpolate ms

% Infer the sampling rate
fs = 1/mean(diff(ts));

% Matrix form of the sinc convolution (hard)
y = sinc((t(:,ones(size(ts))) - ts(:,ones(size(t)))')*fs) * ms;
```

```
function [ms] = sampleit1(t,m,ts)
% SAMPLEIT1 Sample a signal (method 1 is for elementary signals)
%   [ms] = sampleit1(t,m,ts)
%
%   t:   the time vector on which m is defined
%   m:   the 'analog' signal
%   ts:  sampling interval for sampling
%   ms:  the sampled signal

% Sample the signal
ms = interp1(t,m,ts);
```

```
function ms = sampleit2(m,fs,newfs,bf)
% SAMPLEIT2 Sample a signal (method 2 is for music signals)
%   ms = sampleit2(m,fs,newfs,bf)
%
%   m:     the 'analog' signal
%   fs:    the sampling rate of the 'analog' signal
%   newfs: the desired sampling rate
%   bf:    should we use antialiasing filter (0 or 1)

% Specify tolerance to avoid really long integers
[N,D] = rat(fs/newfs,1);

if bf
```

```
    ms = resample(m,D,N,40);
else
    ms = resample(m,D,N,0);
end
```

```
function smpl_plot(t,m,ts,it,ms,c)
% SMPL_PLOT Plots the various signals in the sampling process
%   smpl_plot(t,m,ts,it,ms)
%
%   t:   the time vector on which m is defined
%   m:   the cosine signal
%   ts:  the time vector of the impulses
%   it:  the impulse train
%   ms:  the sampled signal
%   c:   color (eg. 'r' red, 'g' green)

figure;

% Plot the 'analog' signal
subplot(311);
plot(t,m);
grid on;
lim1 = [-.25 .25 -1.1 1.1];
axis(lim1);

% Plot the impulse train
subplot(312);
stem(ts,it,'fill',[c,'^']);
grid on;
lim2 = [-.25 .25 0 1.1];
axis(lim2);

% Plot the sampled signal
subplot(313);
idx = find(ms>=0);
stem(ts(idx),ms(idx),'fill',[c,'^']);
hold on;
idx = find(ms<0);
stem(ts(idx),ms(idx),'fill',[c,'v']);

grid on;
```

```
axis(lim1);
```

```
function recon_plot(t,m,ts,ms,mr,c)
% RECON_PLOT Plots original against reconstructed
%   recon_plot(t,m,ts,ms,mr,c)
%
%   t:   the time vector on which m is defined
%   m:   the cosine signal
%   ts:  the time vector of the impulses
%   ms:  the sampled signal
%   mr:  the reconstructed signal
%   c:   color (eg. 'r' red, 'g' green)

figure;

lim1 = [-.25 .25 -1.1 1.1];
lim2 = [-.25 .25  0   1.1];

subplot(311);
plot(t,m);
grid on;
axis(lim1);

subplot(312);
idx = find(ms >= 0);
stem(ts(idx),ms(idx),'fill',[c,'^']);
hold on;
idx = find(ms < 0);
stem(ts(idx),ms(idx),'fill',[c,'v']);
grid on;
axis(lim1);

subplot(313);
plot(t,mr,c);
grid on;
axis(lim1);
```

```
function time_plot(x,sr)
```

```
% TIME_PLOT Plots a signal in time
%   time_plot(x,sr)
%
%   x:   the signal
%   sr:  it's sampling frequency

% Look at the time signal
t = linspace(0,length(x)/sr,length(x));
plot(t,x);
grid on;
axis tight;
```

```
function freq_plot(x,fs,newfs)
% FREQ_PLOT Plots a signal in frequency (fourier) domain
%   freq_plot(x,fs,newfs)
%
%   x:     the signal
%   fs:    the sampling rate of the 'analog' signal
%   newfs: the desired sampling rate

% Default values
if nargin < 3; newfs = fs; end

% Look at the spectrum
X = am_spectrum(x);
% Normalize
X = X/max(X);
f = linspace(-newfs/2,newfs/2,length(x));
plot(f,X);
lim = [-fs/2 fs/2 0 1];
axis(lim);
grid on;
```
