Laboratório de Sinais e Sistemas

Amostragem

A teoria da amostragem fornece uma compreensão matemática clara da interface entre o mundo analógico e o mundo discreto dos computadores digitais. O objetivo deste laboratório é colocarmos em prática essa teoria, apresentando alguns exemplos simples no MATLAB.

Na primeira parte do laboratório, vamos usar sinais elementares para recriar os resultados semelhantes aos que aparecem no Capítulo referente ao Teorema da Amostragem do Lathi. Desta forma, nos familiarizaremos com as noções de perfeita reconstrução e aliasing.

Na segunda parte do laboratório, vamos usar uma amostra de música real e vamos experimentar frequências de amostragem diferentes. Em seguida, avaliaremos empiricamente o efeito do aliasing.

Amostragem de sinais elementares

Nesta primeira parte do laboratório, vamos nos concentrar em alguns exemplos simples. Assumindo que um já foi exposto à teoria básica de amostragem a execução desta parte do laboratório vai nos ajudar a digerir os conceitos de amostragem, reconstrução e aliasing.

Primeiramente, vamos construir uma sinusóide e amostrá-la usando várias taxas de amostragem.

Projeto 1: Gere uma sinusóide analógica e em seguida, amostre esse sinal. Plote os sinais envolvidos e salve os gráficos.

Observe que o sinal inicial criado usando a função makecos (função em anexo) já é amostrado, mas com uma taxa de amostragem muito alta. Consideramos isso praticamente um sinal analógico, já que não queremos usar variáveis simbólicas em para criar sinais de tempo contínuo.

```
% Faça o sinal analógico inicial
% (frequency 20 Hz)
[m,t] = makecos(20);
```

Agora vamos amostrar este sinal com uma frequência de amostragem adequada. Nós sabemos que nosso sinal inicial é um cosseno com frequência de 20 Hz. Isso significa que a taxa Nyquist para este sinal é 2 * 20 = 40Hz. No primeiro cenário de amostragem iremos usar uma taxa de amostragem de 50Hz que é mais que suficiente para uma reconstrução perfeita. Este é o caso da sobreamostragem, assim temos:

```
%Vamos fazer um trem de impulso para amostrar nosso sinal


% (frequência de amostragem de 50 Hz)
[it1,ts1] = makeimp(50);

% Agora amostre o sinal original
ms1 = sampleit1(t,m,ts1);

% Plote todos os sinais para visualizer o processo de amostragem

c1 = 'r'; % cor para o primeiro cenário
smpl_plot(t,m,ts1,it1,ms1,c1);
```

Observe que os gráficos discretos com setas denotam a integral da função dirac delta. Agora vamos amostrar o mesmo sinal analógico, mas com uma taxa de amostragem menor que a taxa de Nyquist. Este é o caso da subamostragem e escolhemos 30Hz.

```
% Agora faça um segundo trem de impulsos
% (frequência de amostragem de 30 Hz menor que a taxa de Nyquist)
[it2,ts2] = makeimp(30);
% Agora amostre o sinal original com a nova taxa de amostragem
ms2 = sampleit1(t,m,ts2);
% Plote todos os sinais para visualizer o processo de amostragem
c2 = 'g'; % cor para o Segundo cenário
smpl_plot(t,m,ts2,it2,ms2,c2);
```

De acordo com a teoria, para o primeiro sinal amostrado ms1, devemos ser capazes de reconstruir o original, enquanto para o segundo ms2 o aliasing deve ocorrer. Vamos agora reconstruir usando a função interpsinc (anexo) e plotar as formas de onda no tempo e na frequência para verificar a teoria da amostragem.

```
% Agora reconstrua as duas versões amostradas
mr1 = interpsinc(ms1,ts1,t);
mr2 = interpsinc(ms2,ts2,t);
% Plote o sinal original e o sinal reconstruido para comparar
recon_plot(t,m,ts1,ms1,mr1,c1);
```

```
recon_plot(t,m,ts2,ms2,mr2,c2);
```

Observe como o primeiro cosseno reconstruído tem a mesma frequência que o original, enquanto segundo tem uma frequência diferente. Claramente, no segundo sinal reconstruído, não temos reconstrução perfeita. Seguindo a teoria, podemos calcular que a frequência aparente é de 10Hz. Como estamos examinando frequências, podemos avaliar diretamente nossos cálculos na frequência domínio.

```
% Agora vamos obter o espectro para examinar o aliasing

% Faça o índice de frequência para plotagem

f = (-5000/2):(1/2):(5000/2);

% Use a função do anexo para calcular o espectro

M = am_spectrum(m);

MR1 = am_spectrum(mr1);

MR2 = am_spectrum(mr2);

% Plote o espectro para comparar

am_plot(f,M,MR1,MR2,0.02);
```

Verifique a frequência do sinal original e dos dois sinais reconstruídos, observe o aliasing.

TAREFAS:

Seguindo o procedimento descrito acima, verifique o aliasing e calcule a frequência dos co-senos reconstruídos nos seguintes casos:

• Cosseno: 30Hz, Amostragem: 50Hz

• Cosseno: 40Hz, Amostragem: 15Hz

• Cosseno: 10Hz, Amostragem: 50Hz

• Cosseno: 20Hz, Amostragem: 40Hz

Anexo

```
function [m,t] = makecos(f,len)

% MAKECOS Faz um sinal cossenoidal analógico

% [m,t] = makecos(f,len)

%
```

```
% f: frequência do cosseno em hertz
% len: comprimento em segundos
% m: o sinal cossenoidal
% t: the time vector on which m is defined
% valor default
if nargin < 2; len = 2; end
sr = 5000;
t = ((-len/2):1/sr:(len/2)).';  % vetor de tempo
m = cos(2*pi*f*t);  % nosso sinal
```

```
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;
```

```
function X = am_spectrum(x)
% AM_SPECTRUM Calculates the spectrum of a signal %
X = am_spectrum(x)
%
% x: signal
% X: spectrum of signal x
X = abs(fftshift(fft(x)));
```

```
function am_plot(idx,m,c,u,rng)
% AM_PLOT Plots the three modulation signals
%   am_plot(m,c,u,rng)
```

```
%
%   idx: x index (it can represent time or frequency)
%   m:   modulating signal
%   c:   carrier signal
%   u:   modulated signal
%   rng: range of x axis to plot (optional)

if nargin < 5; rng = 1; end % default value for rng

figure; % create a new figure so we don't overwrite an existing one
subplot(311); % split the figure in three subplots
plot(idx,m); grid on;
axis([rng*min(idx) rng*max(idx) min(m)-0.1 max(m)+0.1]);
subplot(312);
plot(idx,c); grid on;
axis([rng*min(idx) rng*max(idx) min(c)-0.1 max(c)+0.1]);
subplot(313);
plot(idx,u); grid on;
axis([rng*min(idx) rng*max(idx) min(u)-0.1 max(u)+0.1]);
```