Digital Communications Lab.02

Ali Emad 19016028 — Amr Ramadan 19016110 — Marwan Ahmed 19016605

March 23, 2023

1 Introduction

This lab report investigates the performance of the Pulse Code Modulation (PCM) system, which is a widely used digital communication technique. The objectives of this experiment are to investigate the various components of the PCM system, examine the impact of changing the number of levels, and evaluate the effects of oversampling, critical sampling, and undersampling. Additionally, the quantization error of the transmitted signal will be calculated to determine the accuracy of the system. The findings of this study will provide insights into the PCM system's behavior under different scenarios and help to enhance its performance.

2 Quantization Error

The provided MATLAB code investigates the reconstruction of a signal from oversampling in a Pulse Code Modulation (PCM) system. The signal being used is a cosine wave with a frequency of 2Hz sampled at 4kHz. The code iterates over a list of numbers and calculates the quantized signal with double fixed-point representation for each number. Then, it plots the quantized signal and the original signal for each number in a separate figure.
Furthermore, the code computes the absolute mean square error between the original signal and each quantized signal. The computed error is stored in the variable 'error'. Finally, it plots the error versus the number of levels used in the quantization.
The code also includes a function 'calculate˙abs˙mean˙square˙error', which calculates the absolute mean square error between two vectors. The function takes two input vectors, vector1 and vector2, and returns the absolute mean square error between them.

2.1 Matlab Program

```
clear
clc

% reconstruction from oversampling
t=0:(1/4000)*1:1;% time signal
x = cos(2*pi*2*t);
numlist = [3,4,5,10];
y = zeros(length(numlist),length(t));

for i = 1:length(numlist)
    y(i,:) = double(fi(x,1,2*numlist(i),numlist(i)));
    figure
    plot(t, y(i,:),'r' )
    hold on;
    plot(t, x,'b' )
    title('Quantized signal (8 bits)');
    xlabel('time')
    ylabel('quantized signal')
    hold off
end

error = zeros(1,length(numlist));

```

```
for i = 1:length(numlist)
    error(i) = calculate_abs_mean_square_error(x,y(i,:));
end
figure
plot(numlist, error,'r' )

function abs_mean_square_error = calculate_abs_mean_square_error(vector1, vector2)
% This function calculates the absolute mean square error between two vectors.

% Calculate the difference between the two vectors
difference = vector1 - vector2;

% Square each element of the difference vector
squared_difference = difference .^ 2;

% Calculate the mean of the squared difference vector
mean_squared_difference = mean(squared_difference);

% Calculate the square root of the mean squared difference
abs_mean_square_error = sqrt(mean_squared_difference);
end
```

2.2 Figures

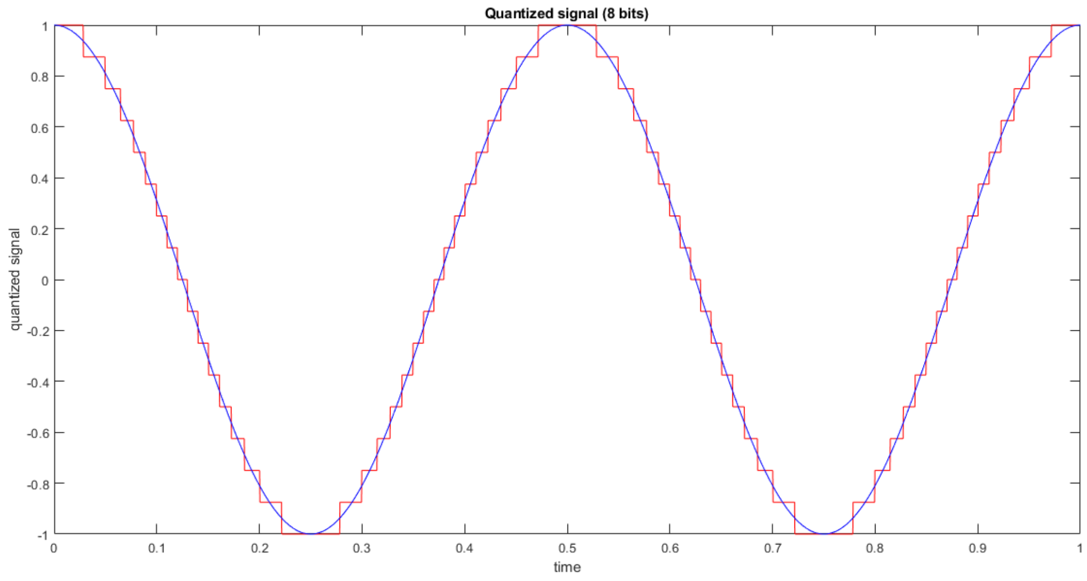

Figure 1: m = 3

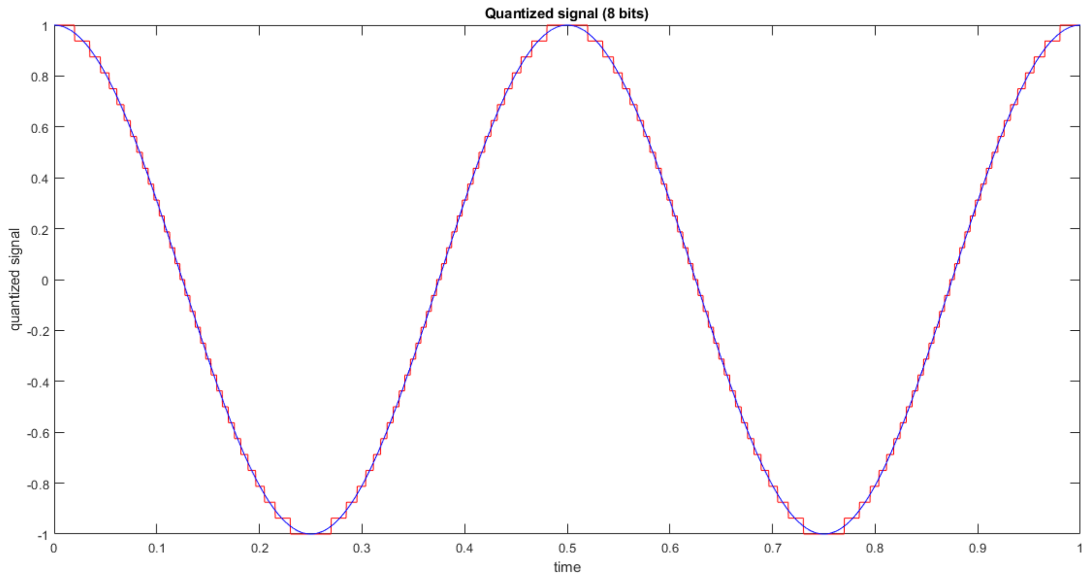

Figure 2: m = 4

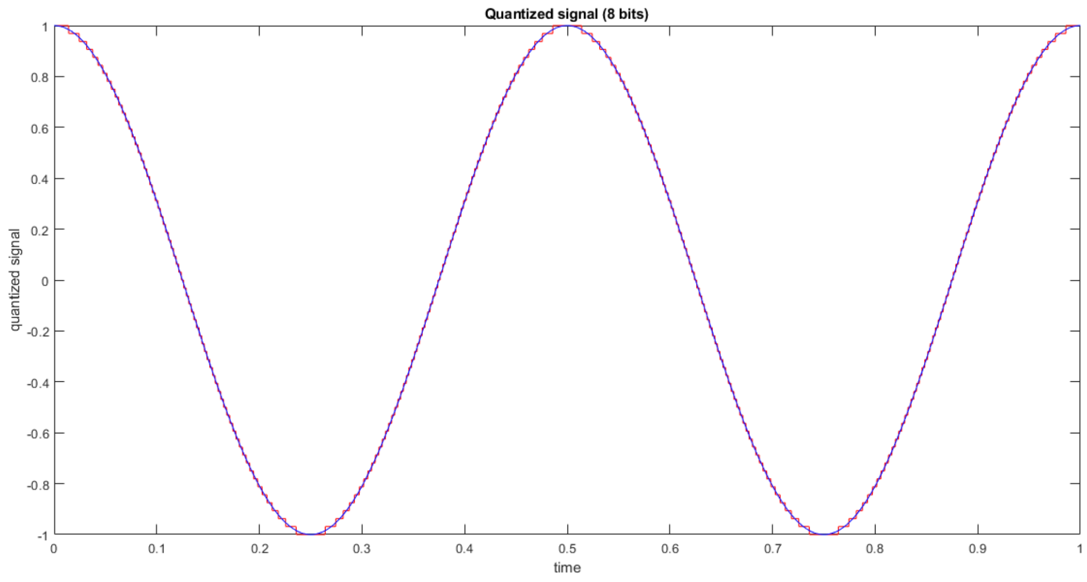

Figure 3: m = 5

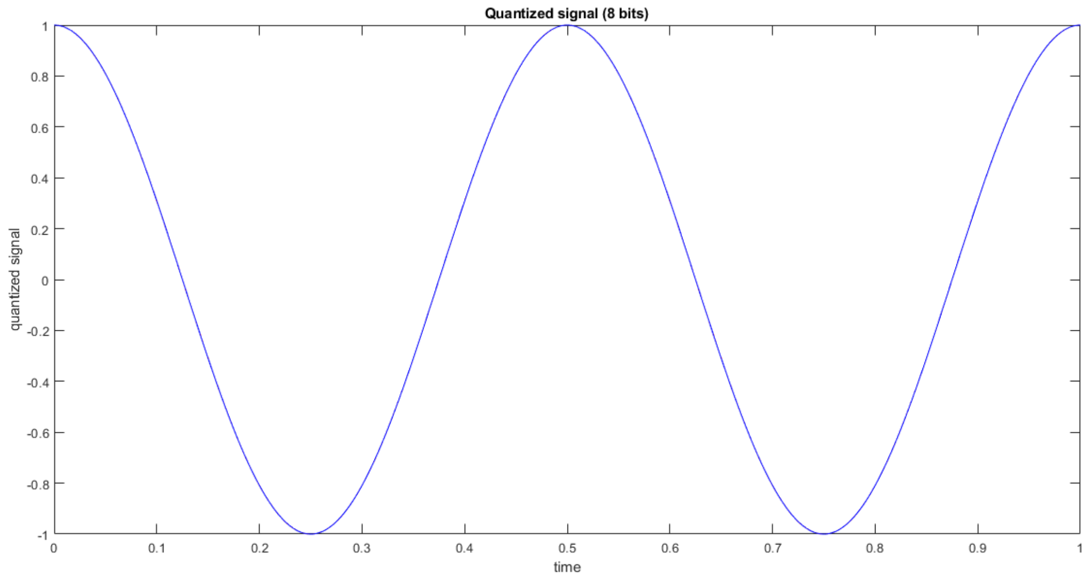

Figure 4: m = 10

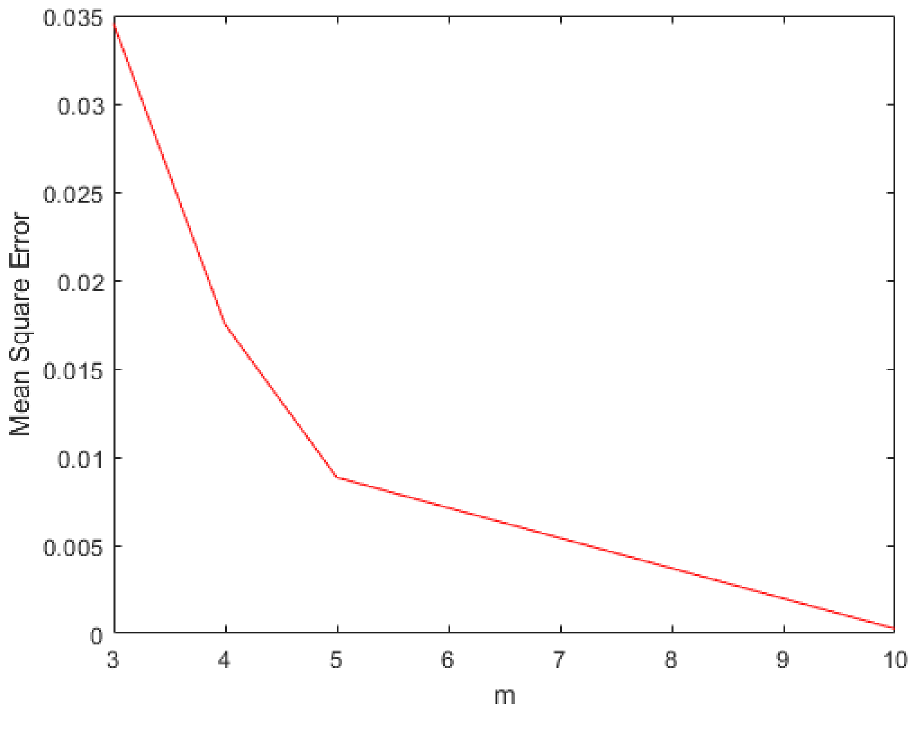

Figure 5: Error vs m

3 Sampling Distortion

The given MATLAB code demonstrates the reconstruction of signals from oversampling, minimum sampling, and undersampling. The code utilizes the Butterworth low-pass filter to filter the over-sampled signal, the minimum-sampled signal, and the under-sampled signal.

The code starts by clearing the variables and the command window. It then creates a signal with a frequency of 5

Hz that is sampled at a rate of 100 kHz. Next, a Butterworth low-pass filter with a cutoff frequency of 1000 Hz is designed and applied to the oversampled signal. The oversampled signal is created by zero-padding the original signal by a factor of 10. The first 9 samples of the zero-added signal are removed to display the signal with a higher sampling rate without changing its spectrum.

In the next section, the minimum-sampled signal is constructed. A signal with a frequency of 5 Hz is created and sampled at a rate of 10 Hz. A Butterworth low-pass filter with a cutoff frequency of 0.1 is designed and applied to the signal. The signal is then zero-padded by a factor of 10 and filtered using the same filter coefficients. Finally, the Fourier transform of the signal is taken and plotted.

The last section constructs the under-sampled signal. A signal with a frequency of 5 Hz is created and sampled at a rate of 5 Hz. A Butterworth low-pass filter with a cutoff frequency of 0.2 is designed and applied to the signal. The signal is then zero-padded by a factor of 10 and filtered using the same filter coefficients. Finally, the Fourier transform of the signal is taken and plotted.

Note that the sampling rate used in the code is chosen based on the Nyquist rate. The step size used in the minimum-sampled signal and the under-sampled signal is chosen to avoid aliasing. The code utilizes the filter function to apply the Butterworth low-pass filter, which is designed using the BUTTER function. The filter function is a general-purpose function for applying digital filters to signals. It can be used with any filter design method, not just Butterworth filters.

3.1 Matlab Program

```
% this part below must be copied to your m file and complete the %
clear
clc

% reconstruction from oversampling
t=0:0.001:1;% time signal
y=2*cos(2*pi*5*t);

[B,A] = butter(3,1000/100000,'low' ); % butter fly filter
%a Butterworth low-pass filter is designed using the BUTTER function.
%The filter has a order of 3 and a cutoff frequency of 1000 Hz (normalized with
    respect to the sampling frequency of 100 kHz).
%The filter coefficients are stored in the vectors B and A.
zero_added_signal=zeros(1,length(y)*10); %zero padding like dsp

for i=1:length(y)
    zero_added_signal(i*10)=y(i); %the new signal (padded signal)
end

zero_added_signal(1:9)=[]; %To display the signal with a higher sampling rate
    without changing its spectrum,
                              %the first 9 samples of the zero-added signal are
                                  removed

% Adding zeros enhances the signal display and don't change the
%spectrum,it changes sampling freq. only (just higher resolution)

t=linspace(0,1,length(zero_added_signal)); %new time vector - 9
filtered_signal = filter(B,A,zero_added_signal); % pass the filter coefficients A B
     and the over sampled signal
plot(t,filtered_signal,'r' )
% XLABEL('time')
% YLABEL('oversampled signals')
```

```
%The filter function is a general-purpose function for applying digital filters to
    signals.
%It can be used with any filter design method, not just Butterworth filters.
%The BUTTER function, on the other hand, is a specific function for designing
    Butterworth filters.


%% construction from minimum sampling
figure
t=0:0.1:1; % replace ?? with the suitable number
%considering the Nyquist rate in the signal to avoid aliasing. In this case, the
    signal has a frequency of 5 Hz, so the Nyquist rate is 10 Hz.
%Therefore, the step size should be no larger than 0.1 seconds

y=2*cos(2*pi*5*t);
[B,A] = butter(10,0.1,'low'); %cutoff = 0.1 (normalized wrt sampling frequency)
zero_added_signal=zeros(1,length(y)*10);
for i=1:length(y)
zero_added_signal(i*10)=y(i);
end
zero_added_signal(1:9)=[];
t=linspace(0,1,length(zero_added_signal));
filtered_signal = filter(B,A,zero_added_signal);
plot(t,filtered_signal,'r')
% XLABEL('time')
% YLABEL('minimum sampled signals')

s=fft(filtered_signal);
s=fftshift(s);
fs=100; % why 100?? Write your comments in the m file
%If the original signal has a frequency of 10 Hz and it was resampled by a factor
    of 10,
% then the resulting signal would have a sampling frequency of 100 Hz


freq=linspace(-fs/2,fs/2,length(s)); %length of spectrum
figure
plot(freq,abs(s))
% XLABEL('freq')
% YLABEL('magnitude of minimum sampled signals')
%% construction from undersampling sampling
figure
t=0:0.2:1;
y=2*cos(2*pi*5*t);
[B,A] = butter(10,0.2,'low');
% complete this part as shown in the construction from minimum sampling
%and do the necessary changes , you have to do low pass filtering and %

zero_added_signal=zeros(1,length(y)*10);

for i=1:length(y)
    zero_added_signal(i*10)=y(i);
end

zero_added_signal(1:9)=[];
```

```
% Adding zeros enhances the signal display and don't change the
%spectrum,it changes sampling freq. only

t=linspace(0,1,length(zero_added_signal));
filtered_signal = filter(B,A,zero_added_signal);
plot(t,filtered_signal,'r' )

s=fft(filtered_signal);
s=fftshift(s);
fs=100;


freq=linspace(-fs/2,fs/2,length(s));

figure
plot(freq,abs(s))
 xlabel('freq')
 ylabel('magnitude of under sampled signals')
```

3.2 Figures

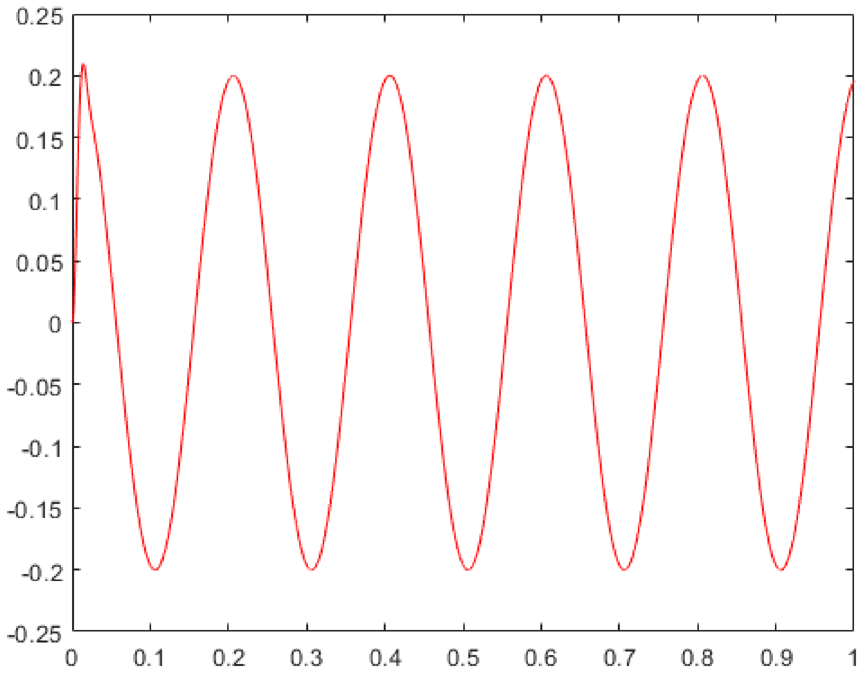

Figure 6: Reconstruction from oversampling

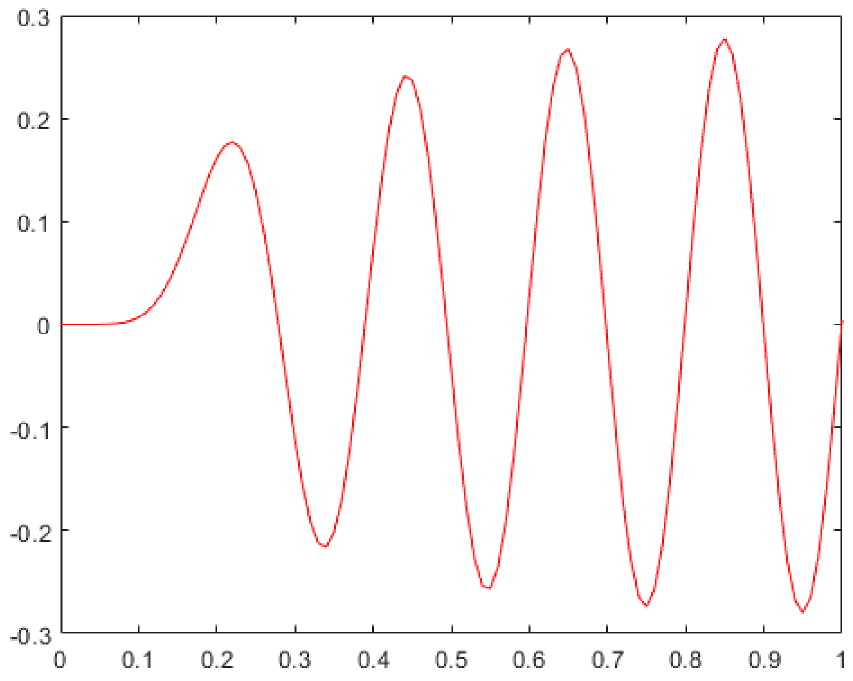

Figure 7: Reconstruction from critical sampling

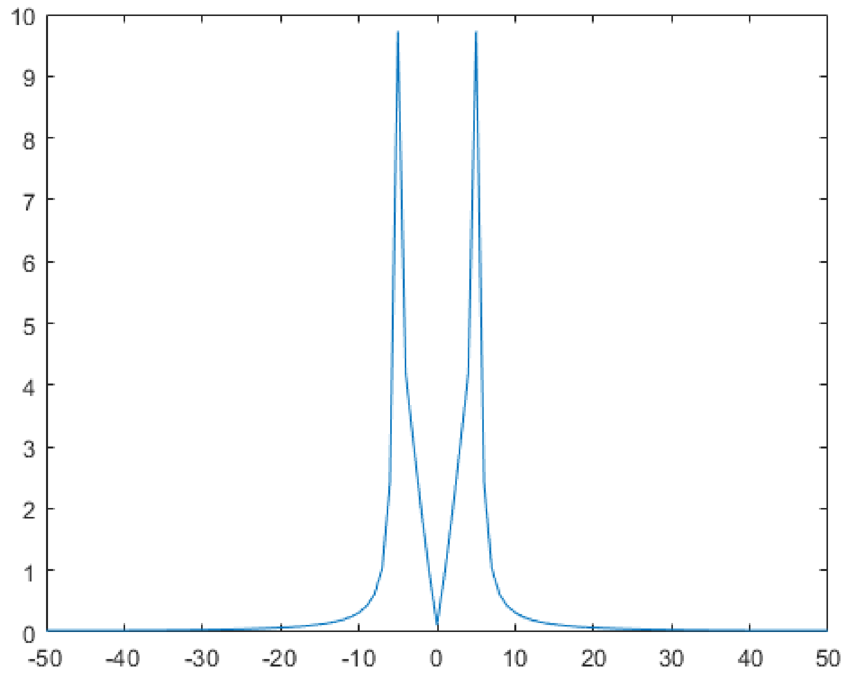

Figure 8: Critical sampling frequency spectrum

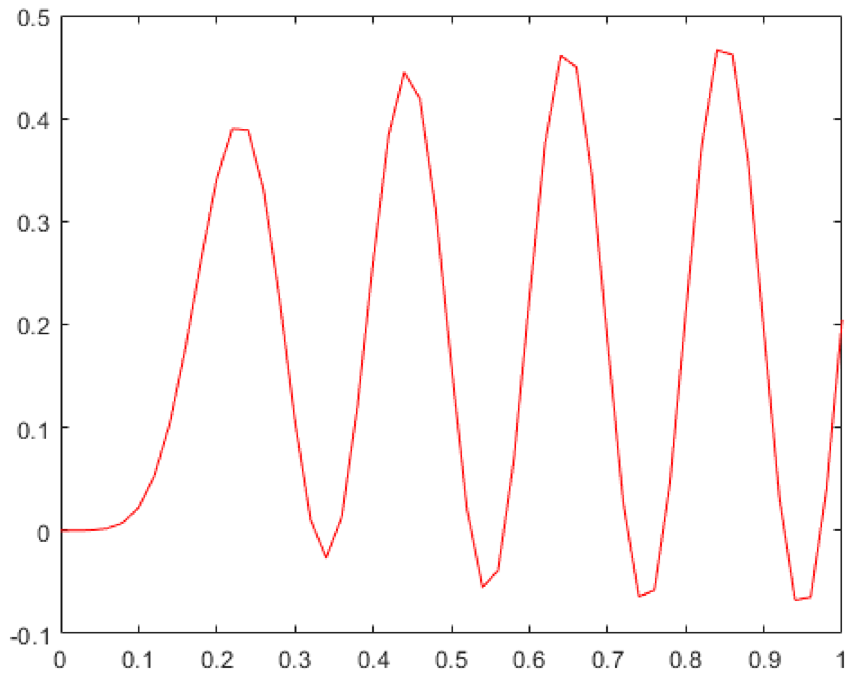

Figure 9: Reconstruction from undersampling

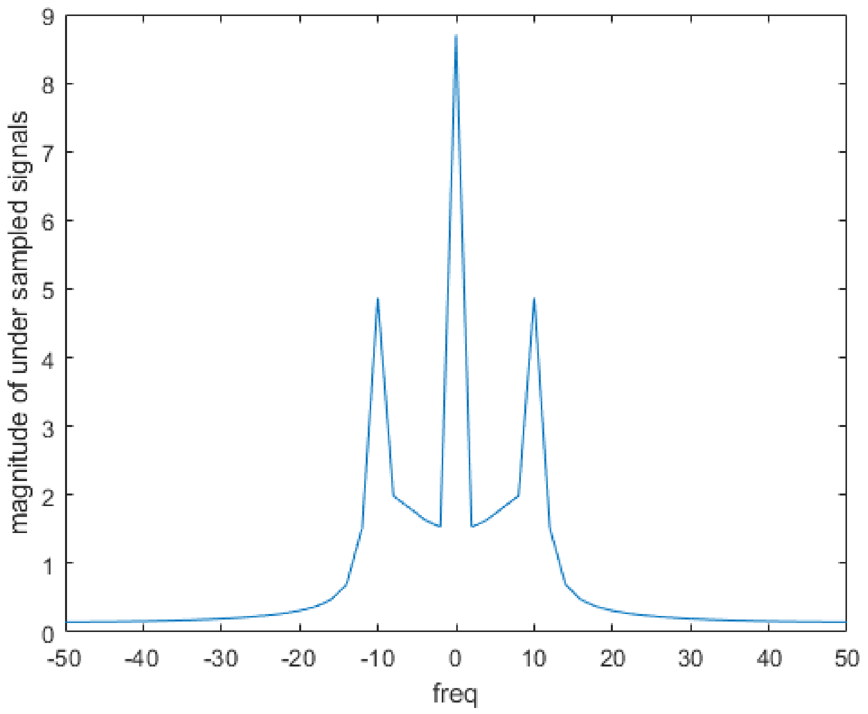

Figure 10: Critical sampling frequency spectrum (alliasing)