

Lab 5 – Sampling and Reconstruction

XMUT220 Signals and Systems

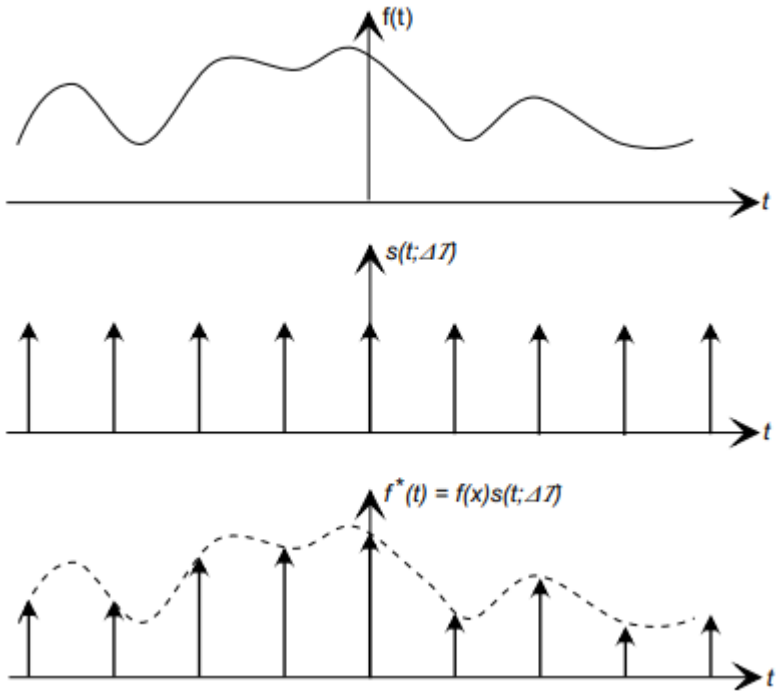
Overview

1. Sampling and reconstruction.
2. Aliasing of sampling signal.
3. Application of sampling and reconstruction.

A. Sampling of Signals

Sampling is is a process of conversion of continuous signal into discrete signal.

The output of the sampling technique which is a sampled signal represents the discrete version of its analogue signal.



The equation for sampling is as outlined in the expressions given as follow.

For a given continuous signal $x(t)$ to be sampled, the sampling function (also called a comb function):

$$x_s(t) = x(t) \cdot \mathrm{comb}(t) \qquad (Eq.1)$$

So, substitute the comb function above with equation below to the sampling equation

$$\mathrm{comb} = \sum_{m=-\infty}^{\infty} \delta(t - mt_0) \qquad (Eq.2)$$

By applying the comb function to our sampling process, this gives the following equation for sampling of signals:

$$x_s(t) = x(t) \cdot \sum_{m=-\infty}^{\infty} \delta(t - mt_0) \qquad (Eq.3)$$

To sample a signal in MATLAB, generate a time vector at the appropriate rate, and use this to generate the signal.

Since sampled signal $x_s(t)$ is a discrete signal, plot the discrete plot using the `stem()` function (See Appendix for further details of this function).

It is useful to plot the continuous time signal on the same plot.

All signals in MATLAB are essentially discrete-time, but they will look like continuous-time signals if the sampling rate is much higher than the Nyquist rate.

1. By using a sampling signal at frequency of 1 kHz, perform sampling on the following continuous time signals.

   i. $x_1 = x(t) = 25\cos(500\pi t)$

   ii. $x_2 = 3\cos(50\pi t) + 10\sin(300\pi t)$

   iii. $x_3 = sinc\,(200\pi t)$

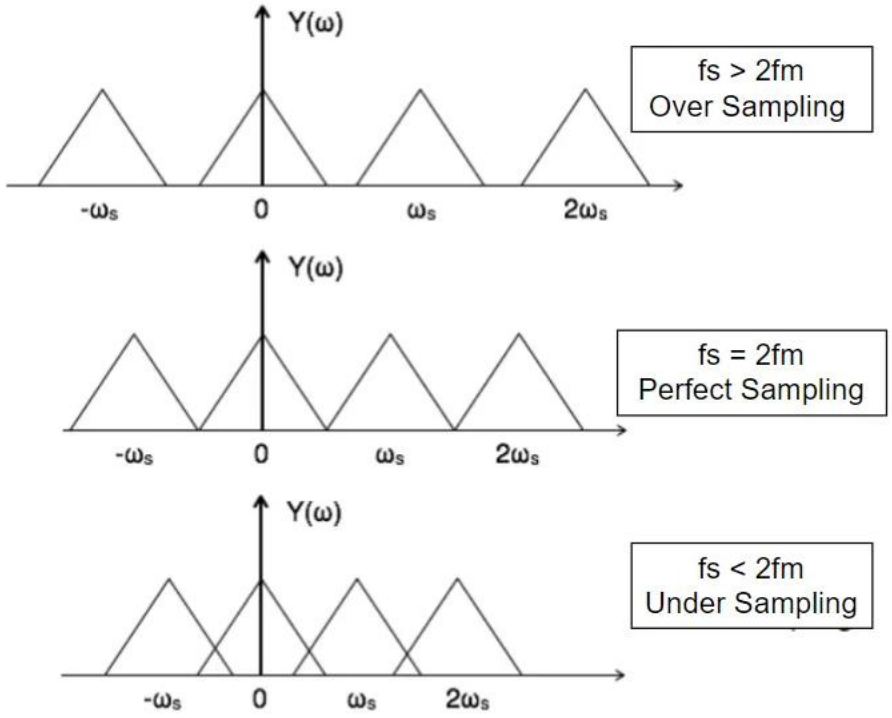
   a. Simulate the time domain plot of the signals in Matlab. [15 marks]

   b. Comment on the results of the sampling. [10 marks]

B. Aliasing in Signal Sampling

Aliasing is the effect in which overlapping of a frequency component takes place at the frequency higher than Nyquist rate i.e. $f_s > 2f_m$. Loss of signal may occur due to aliasing effect.

When the input frequency is greater than half the sample frequency, the sampled points do not adequately represent the input signal. Inputs at these higher frequencies are observed at a lower, aliased frequency.



Using the equation given below, whether a signal is experiencing the aliasing or not is determined as follow (e.g. f_s = sampling frequency and f_m = maximum frequency (component) of the signal to be sampled:

$$f_s > 2f_m \quad \text{Over sampling}$$
$$f_s = 2f_m \quad \text{Perfect sampling}$$
$$(Eqs. 4)$$
$$f_s < 2f_m \quad \text{Under sampling (i.e. aliasing)}$$

To analyse the signal at various sampling frequency as shown in the figure above, use `fft()` function in MATLAB to simulate the frequency spectrum of the signal.

A fast Fourier transform (FFT) is a highly optimized implementation of the discrete Fourier transform (DFT), which convert discrete signals from the time domain to the frequency domain.

Furthermore, FFT computations provide information about the frequency content, phase, and other properties of the signal.

When analysing the frequency components of signals, it can be helpful to shift the zero-frequency components to the centre.

You might need to use `fftshift()` function to shift zero-frequency component to centre of spectrum, so the frequency spectrum of the signal is centred like as shown in the figure above.

See further the Appendix the details of the implementation of these two functions in MATLAB.

2. You are given the following continuous time signal.

$$x(t) = 3\cos(50\pi t) + 10\sin(300\pi t) - \cos(100\pi t)$$

a. It is required that you carry out sampling process on the signal. In Matlab, plot and describe the frequency spectrum of the sampled signals at these sampling frequencies:

i. 10 Hz. [10 marks]

ii. 100 Hz. [10 marks]

iii. 1000 Hz. [10 marks]

b. Describe signal aliasing. Determine whether aliasing existed in part (a). [10 marks]

C. Decimation and Interpolation

Decimation or down sampling is the process of reducing the sample rate F_S in a signal processing system, and interpolation or up sampling is the opposite, increasing the sample rate F_S in a signal processing system.

In down sampling (decimation), the sampling rate is reduced from F_S to F_S/M by discarding $M-1$ samples for every M samples in the original sequence. Down sampling (decimation) factor (M) is simply the ratio of the input rate to the output rate.

$$M = \text{input rate/output rate} \qquad (Eq.5)$$

For M-fold decimation, or down-sampling, its equation is defined by:

$$y[n] = v[nM] = \sum_{k=-\infty}^{\infty} h[k]\, x[nM-k] \qquad (Eq.6)$$

Up sampling (interpolation) works by inserting $(L-1)$ zero-valued samples for each input sample. The sampling rate therefore increases from F_S to LF_S. The up sampling (interpolation) factor (L) is simply the ratio of the output rate to the input rate.

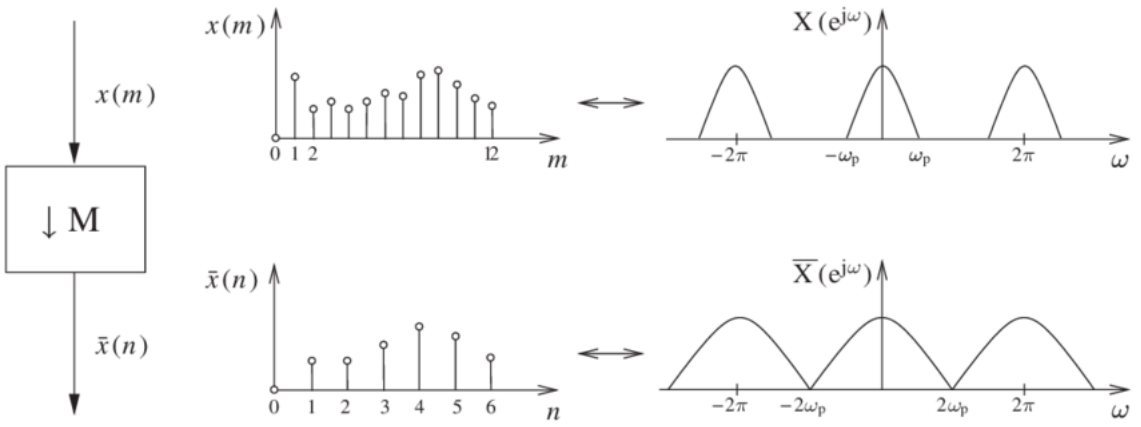
$$L = \text{output rate/input rate} \qquad (Eq.7)$$

CThe mathematical definition of L-fold interpolation is defined by:

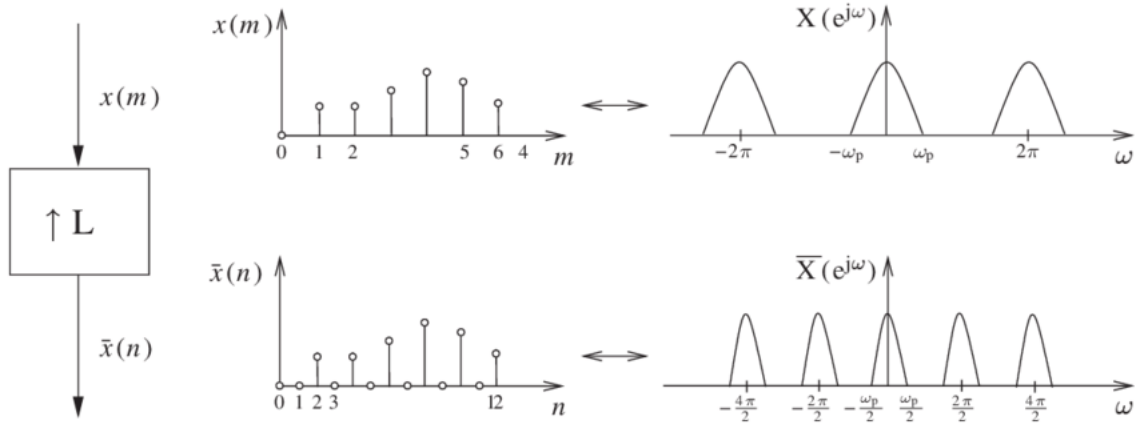
$$y[n] = L \sum_{k=-\infty}^{\infty} h[k]\, w[n-k] \qquad (Eq.8)$$

Where:

$$w[n] = \begin{cases} x[n/L], & \text{if } n/L \text{ is an integer} \\ 0, & \text{if } n/L \text{ is non} - \text{integer} \end{cases}$$



(a) Signal decimation with M = 2.



(b) Signal interpolation with L = 2.

In MATLAB, use `dec()` and `interp()` functions to perform decimation and interpolation in signal sampling respectively (see the details of these functions in the Appendix).

3. Consider a continuous time signal given in the following equation:

$$x(t) = 1.5\cos(2f_1 t) + \cos(2f_2 t)$$

if f_1 = 100 Hz and f_2 = 200 Hz and sampling frequency is 1 kHz, perform the following tasks and obtain the relevant plots in Matlab.

a. Sample the signal and get its discrete-time signal and describe the result. [10 marks]

b. Interpolate this signal by factor I = 3 and describe the result. [10 marks]

c. Decimate this interpolated signal by factor D = 3 and describe the result. [10 marks]

d. Based on the simulation results of parts (b) and (c), describe decimation and interpolation methods. How these methods are used in signal processing applications? [5 marks]

stem() – Plot Discrete Sequence and Signal function

Syntax

```
stem(Y)
stem(X,Y)
```

Description

`stem(Y)` plots the data sequence, Y, as stems that extend from a baseline along the *x*-axis. The data values are indicated by circles terminating each stem.

•If Y is a vector, then the *x*-axis scale ranges from 1 to length(Y).

•If Y is a matrix, then stem plots all elements in a row against the same *x* value, and the *x*-axis scale ranges from 1 to the number of rows in Y.

`stem(X,Y)` plots the data sequence, Y, at values specified by X. The X and Y inputs must be vectors or matrices of the same size.

Additionally, X can be a row or column vector and Y must be a matrix with length(X) rows.

•If X and Y are both vectors, then stem plots entries in Y against corresponding entries in X.

•If X is a vector and Y is a matrix, then stem plots each column of Y against the set of values specified by X, such that all elements in a row of Y are plotted against the same value.

•If X and Y are both matrices, then stem plots columns of Y against corresponding columns of X.

`stem(___,"filled")` fills the circles. Use this option with any of the input argument combinations in the previous syntaxes.

`stem(___,LineSpec)` specifies the line style, marker symbol, and colour.

Examples

Plot Single Data Series

Create a stem plot of 50 data values between -2π and 2π.

```
figureY = linspace(-2*pi,2*pi,50);stem(Y)
```

Data values are plotted as stems extending from the baseline and terminating at the data value. The length of Y automatically determines the position of each stem on the *x*-axis.

Plot Multiple Data Series

Plot two data series using a two-column matrix.

```
figureX = linspace(0,2*pi,50)';Y = [cos(X),
0.5*sin(X)];stem(Y)
```

Each column of Y is plotted as a separate series, and entries in the same row of Y are plotted against the same *x* value. The number of rows in Y automatically generates the position of each stem on the *x*-axis.

fft() – Fast Fourier Transform function

Syntax

`Y = fft(X)`

Description

`Y = fft(X)` computes the discrete Fourier transform (DFT) of `X` using a fast Fourier transform (FFT) algorithm.

Example

Use Fourier transforms to find the frequency components of a signal. Specify the parameters of a signal with a sampling frequency of 1 kHz and a signal duration of 1.5 seconds.

```
Fs = 1000;              % Sampling frequency
T = 1/Fs;               % Sampling period
L = 1500;               % Length of signal
t = (0:L-1)*T;          % Time vector
```

Form a signal containing a 50 Hz sinusoid of amplitude 0.7 and a 120 Hz sinusoid of amplitude 1.

```
S = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
```

Now, take the Fourier transform of the original signal and retrieve single-sided amplitude spectrum of the exact amplitudes, 0.7 and 1.0.

```
Y = fft(S);P2 = abs(Y/L);P1 = P2(1:L/2+1);P1(2:end-1) =
2*P1(2:end-1);
```

fftshift() - Fast Fourier Transform shift function

Syntax

```
Y = fftshift(X)
```

Description

`Y = fftshift(X)` rearranges a Fourier transform `X` by shifting the zero-frequency component to the center of the array.

Example

When analysing the frequency components of signals, it can be helpful to shift the zero-frequency components to the centre.

Create a signal `S`, compute its Fourier transform, and plot the power.

```
fs = 100;                % sampling frequencyt =
0:(1/fs):(10-1/fs); % time vectorS = cos(2*pi*15*t);n =
length(S);X = fft(S);f = (0:n-1)*(fs/n);     %frequency
rangepower = abs(X).^2/n;    %powerplot(f,power)
```

Shift the zero-frequency components and plot the zero-centred power.

```
Y = fftshift(X);fshift = (-n/2:n/2-1)*(fs/n); % zero-
centered frequency rangepowershift = abs(Y).^2/n;     %
zero-centered powerplot(fshift,powershift)
```

decimate() - Decimation function

Syntax

```
y = decimate(x,r)
```

Description

`y = decimate(x,r)` reduces the sample rate of x, the input signal, by a factor of r.

Example

Create a sinusoidal signal sampled at 4 kHz. Decimate it by a factor of four.

```
t = 0:1/4e3:1;

x = sin(2*pi*30*t) + sin(2*pi*60*t);

y = decimate(x,4);
```

interp() - Interpolation function

Syntax

```
y = interp(x,r)
```

Description

`y = interp(x,r)` increases the sample rate of x, the input signal, by a factor of r.

Example

Create a sinusoidal signal sampled at 1 kHz. Interpolate it by a factor of four.

```
t = 0:1/1e3:1;
x = sin(2*pi*30*t) + sin(2*pi*60*t);
y = interp(x,4);
```