MIDDLE EAST TECHNICAL UNIVERSITY
Mechanical Engineering Department

2025-2026Fall

ME461 - Mechatrinic Components and Instruments

[01/11/2025]

Shannon's Sampling Theorem

Author:

Buğra ÇINAROĞLU - 2577849

Ogan Altuğ OKUTAN - ....

Abdullah Naci BODUR - ......

Contents

1 Sampling

We live in an analog world. Whether recording sounds, capturing images, or processing an electromagnetic wave, many sources of information are of analog or continuous-time (CT) nature. We usually want to store, exchange, or manipulate this information using digital computers, microprocessors, and so on. **Sampling:** The process of converting a CT signal to a discrete-time (DT) signal (i.e., a discrete sequence of numbers)

$$x(t) \longrightarrow \ldots, x(-2T), x(-T), x(0), x(T), x(2T), \ldots$$

Sampling theorem states that*under certain conditions* a CT signal can be completely represented by its values ('samples") at points equally spaced in time. That is, no information is lost in the sampling process.

Why is sampling theorem very important/useful?

- It provides a bridge between continuous-time signals and discrete-time signals.
- Sampling theorem allows us to represent continuous-time signals with discrete samples.
- With the development of digital technology, it became much easier to process discrete-time signals.

2 Shannon Sampling Theorem

2.1 Definitions

The **Shannon Sampling Theorem** states that a continuous-time signal $x(t)$ that is *bandlimited* to a maximum angular frequency ω_M can be *completely reconstructed* from its samples if the sampling frequency ω_s satisfies

$$\omega_s > 2\omega_M \quad \text{or equivalently} \quad f_s > 2f_M$$

where f_s is the sampling frequency and f_M is the highest frequency component of $x(t)$.

This condition ensures that the replicas of the signal spectrum in the frequency domain do not overlap, thereby preventing *aliasing*. When this criterion is satisfied, the original signal $x(t)$ can be exactly recovered from its discrete samples $x(nT)$ using an ideal low-pass filter.

Definition: A signal $x(t)$ is bandlimited if

$$X(j\omega) = 0 \quad \text{for } |\omega| > \omega_m \text{ for some } \omega_m$$

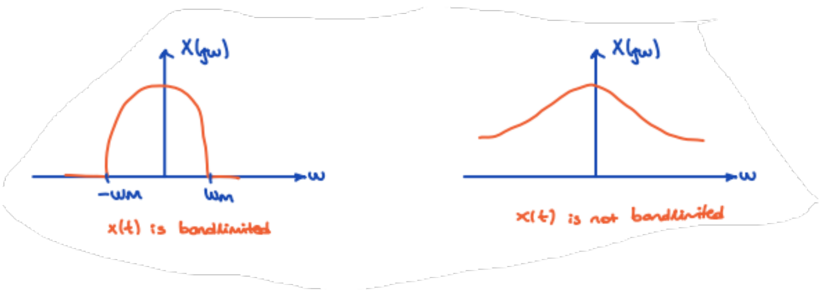

Figure 1: Example for Bandlimited Signal

Ideal impulse-train sampling:
Periodic impulse train

$$p(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT)$$

Sampling of a CT signal $x(t)$:

$$x_p(t) = x(t)\,p(t)$$

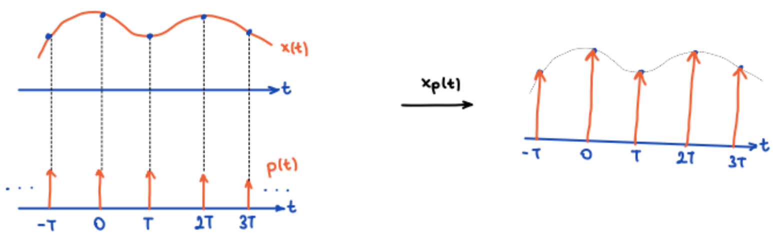

Figure 2: Example for an Impulse Train

$$x_p(t) = \sum_{n=-\infty}^{\infty} x(nT)\,\delta(t-nT)$$

Find the spectrum of $x_p(t)$:

$$p(t) = \sum_{n=-\infty}^{\infty} \delta(t-nT) \quad \Longleftrightarrow \quad P(j\omega) = \frac{2\pi}{T}\sum_{k=-\infty}^{\infty} \delta(\omega - k\omega_s)$$

where

$$\omega_s = \frac{2\pi}{T} \quad \text{(sampling frequency)}, \qquad T : \text{sampling period.}$$

T: sampling period

$$\omega_s = \frac{2\pi}{T} \qquad : \text{sampling frequency}$$

$$X_p(j\omega) = \frac{1}{2\pi}X(j\omega) * P(j\omega) = \frac{1}{2\pi}X(j\omega) * \frac{2\pi}{T}\sum_{k=-\infty}^{\infty}\delta(\omega - k\omega_s) = \frac{1}{T}\sum_{k=-\infty}^{\infty} X(j(\omega - k\omega_s))$$

where $X(j(\omega - k\omega_s))$ represents shifted copies of $X(j\omega)$

When $\omega_s > 2\omega_M$ (sampling frequency greater than twice the largest frequency in the signal)

$x(j\omega)$ can be recovered from $x_p(j\omega)$ if $\omega_s > 2\omega_M$

$x(t)$ can be recovered from $x_p(t)$

Assume $x(t)$ is bandlimited with maximal frequency ω_M. When $\omega_s > 2\omega_M$,

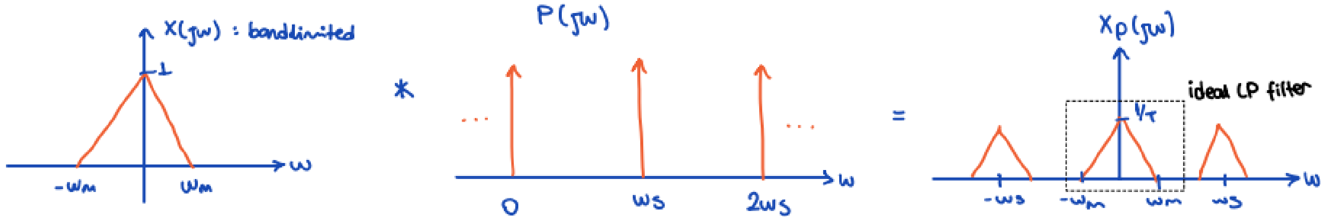

Figure 3: Example of the Method

By using a low-pass (LP) filter, we can get $X(j\omega)$ from $X_p(j\omega)$.

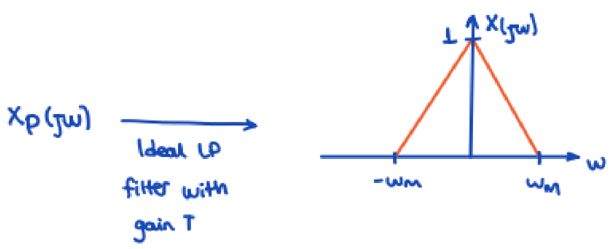

Figure 4: Recovery

So $X(j\omega)$ can be recovered from its samples if $\omega_s > 2\omega_M$.

2.2 Effect of Undersampling: Aliasing

The problem of resulting overlap in the frequency domain is referred to as **aliasing**. If $\omega_s < 2\omega_M$ (undersampling): We lose information $\Rightarrow$ **Aliasing** occurs if $\omega_s < 2\omega_M$. $X(j\omega)$ cannot be recovered from its samples when $\omega_s < 2\omega_M$.

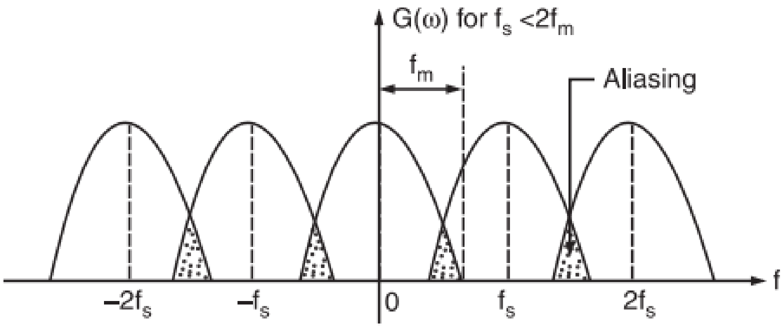

Figure 5: Aliasing

As the sampling frequency decreased the below results are obtained.

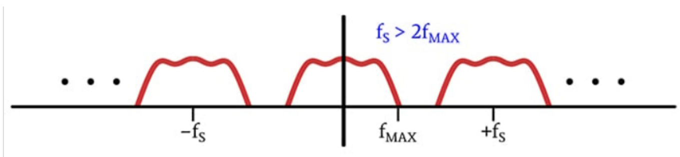

Figure 6: Aliasing Example

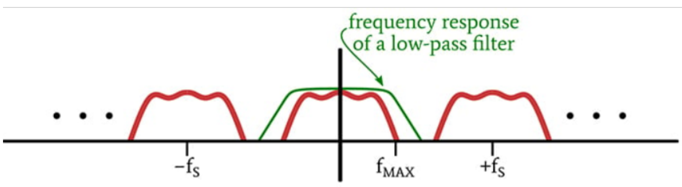

Figure 7: Aliasing Example

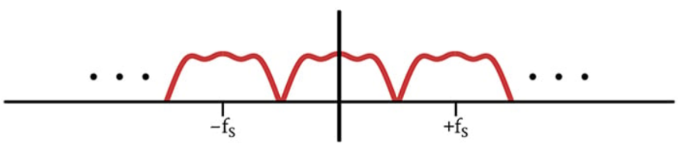

Figure 8: Aliasing Example

3 Examples

For the sampling frequency of $\omega_s = 7$ Hz

$$y(t) = \cos(2\pi \times 15\,\mathrm{Hz} \times t) \text{ where } \omega = 15\,\mathrm{Hz}$$

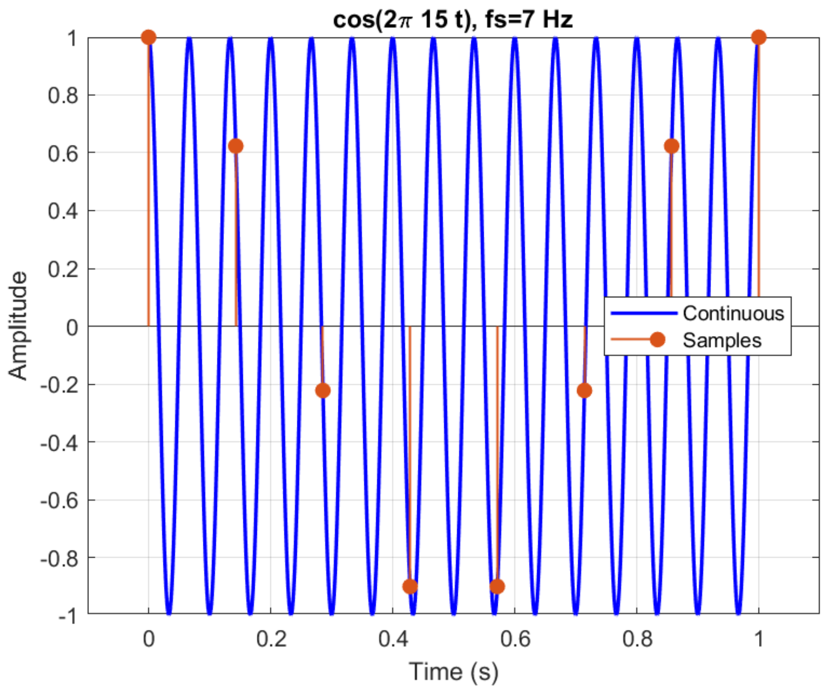

Figure 9: 15 Hz sampled with 7 Hz

If we fit reconstruct the signal from the sampled points.

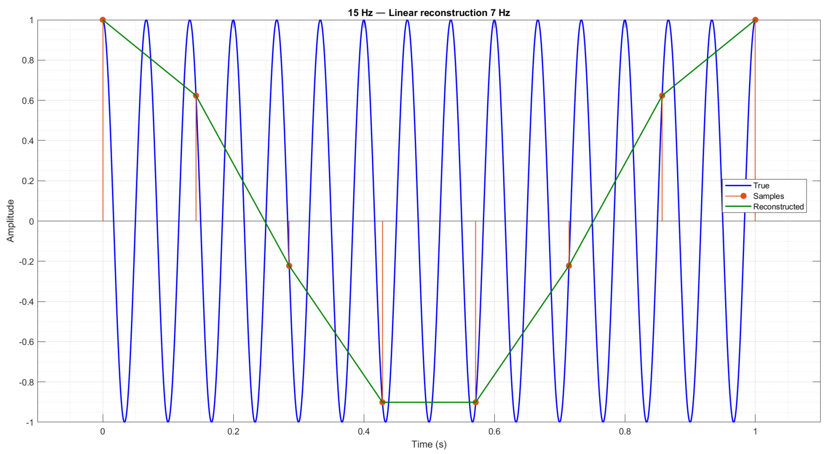

Figure 10: 15 Hz sampled with 7 Hz Reconstruction

$$y(t) = \cos(2\pi \times 13\,\text{Hz} \times t) \text{ where } \omega = 13\,\text{Hz}$$

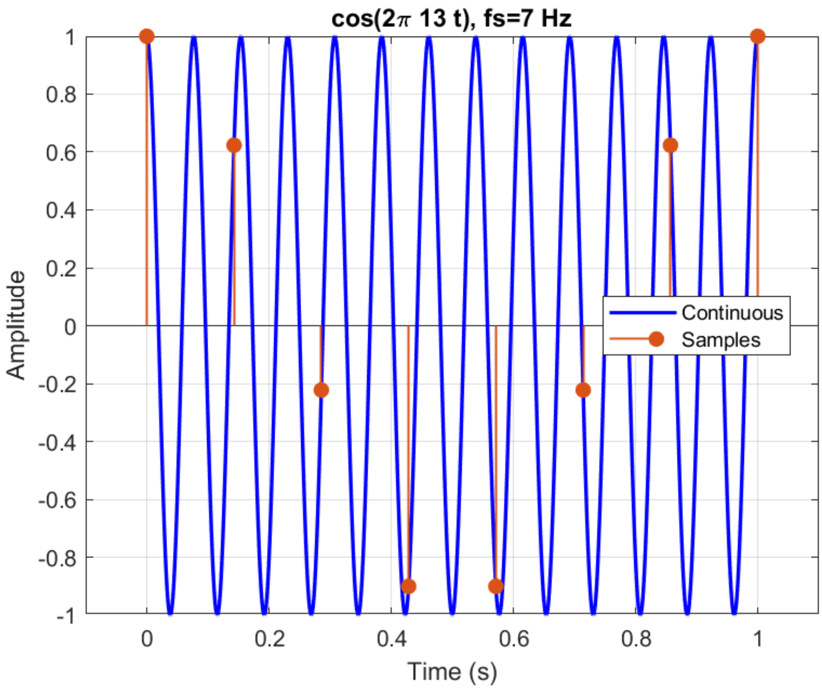

Figure 11: 13 Hz sampled with 7 Hz

If we fit reconstruct the signal from the sampled points.

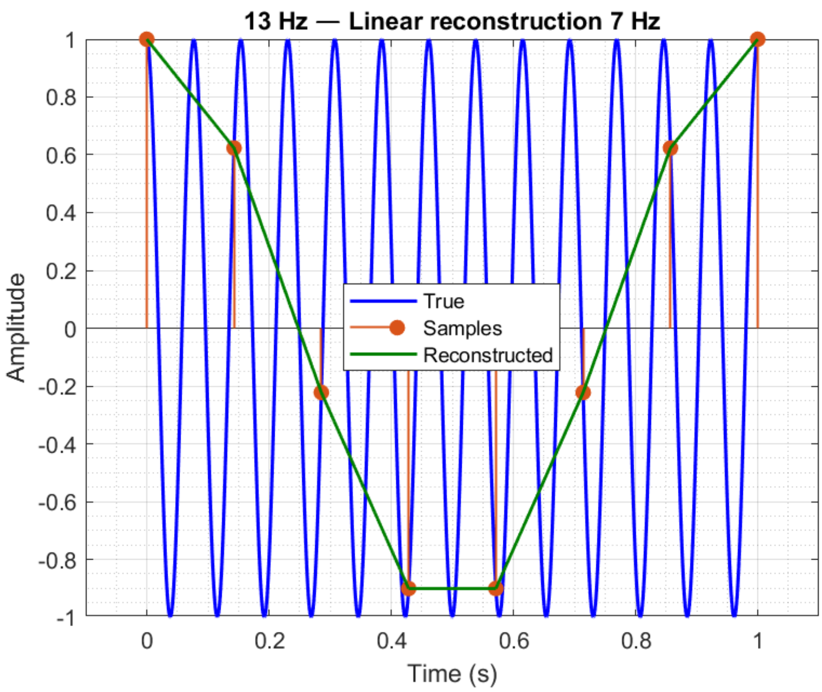

Figure 12: 13 Hz sampled with 7 Hz Reconstruction

$$y(t) = \cos(2\pi \times 8\,\mathrm{Hz} \times t) \text{ where } \omega = 8\,\mathrm{Hz}$$

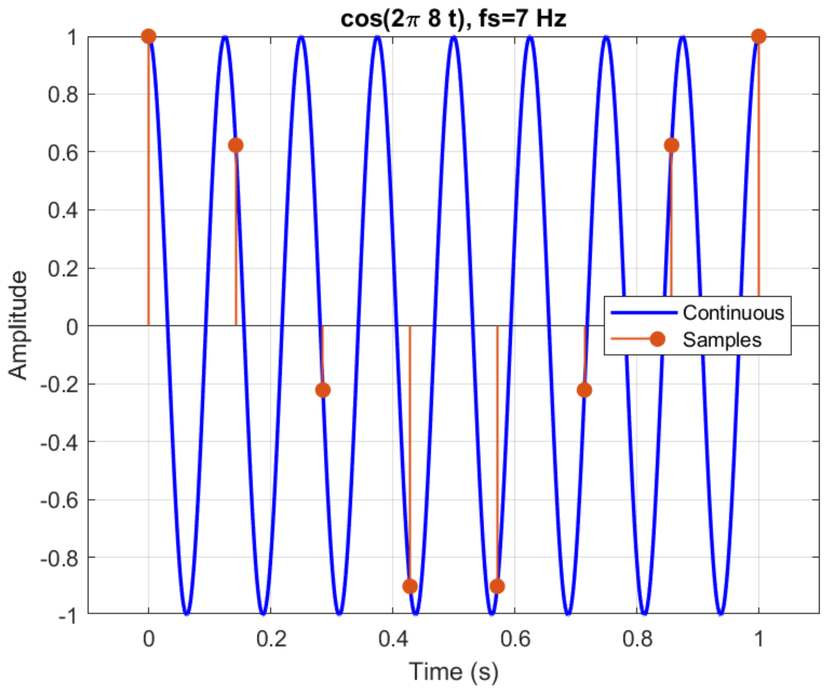

Figure 13: 8 Hz sampled with 7 Hz

If we fit reconstruct the signal from the sampled points.

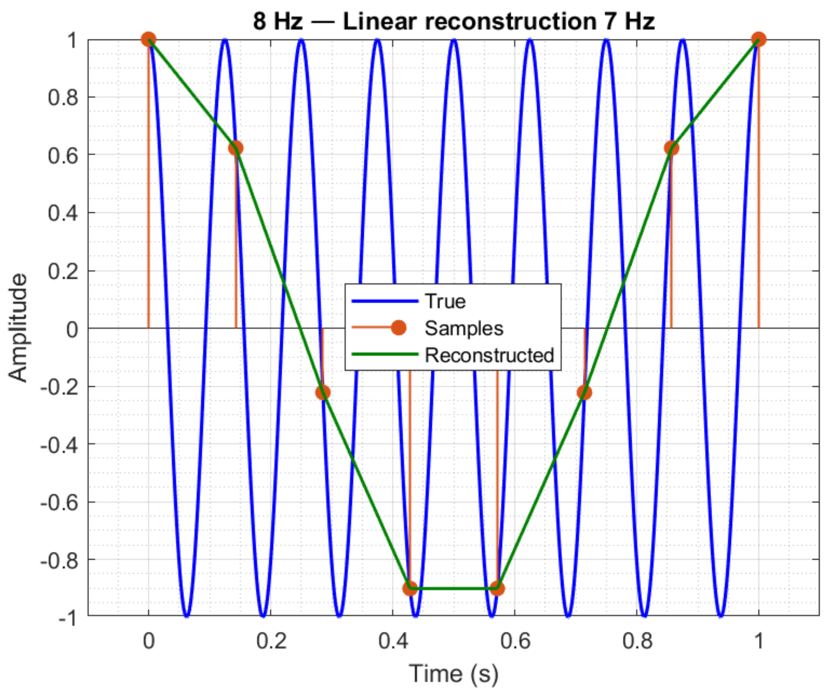

Figure 14: 8 Hz sampled with 7 Hz Reconstruction

$$y(t) = \cos(2\pi \times 6\,\text{Hz} \times t) \text{ where } \omega = 6\,\text{Hz}$$

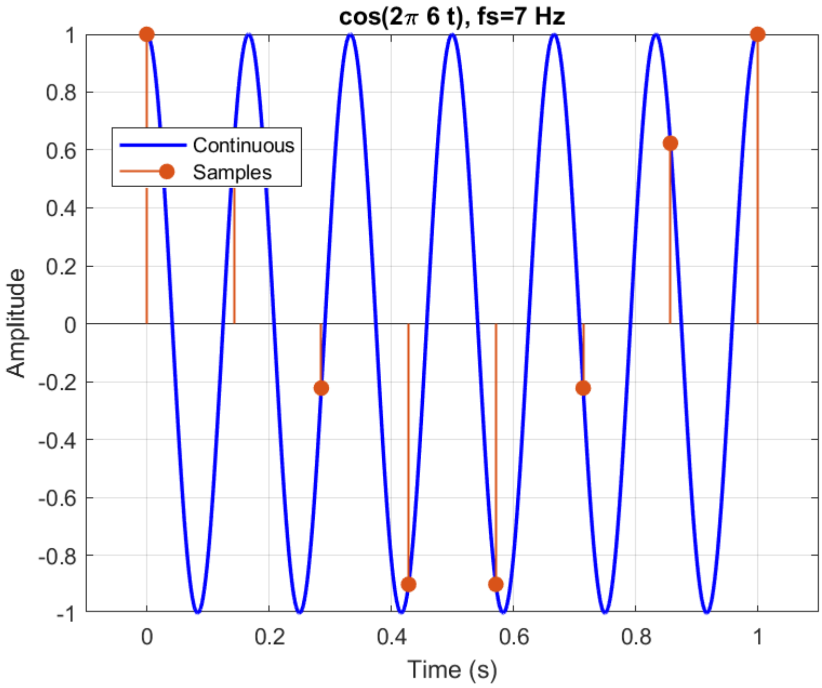

Figure 15: 6 Hz sampled with 7 Hz

If we fit reconstruct the signal from the sampled points.

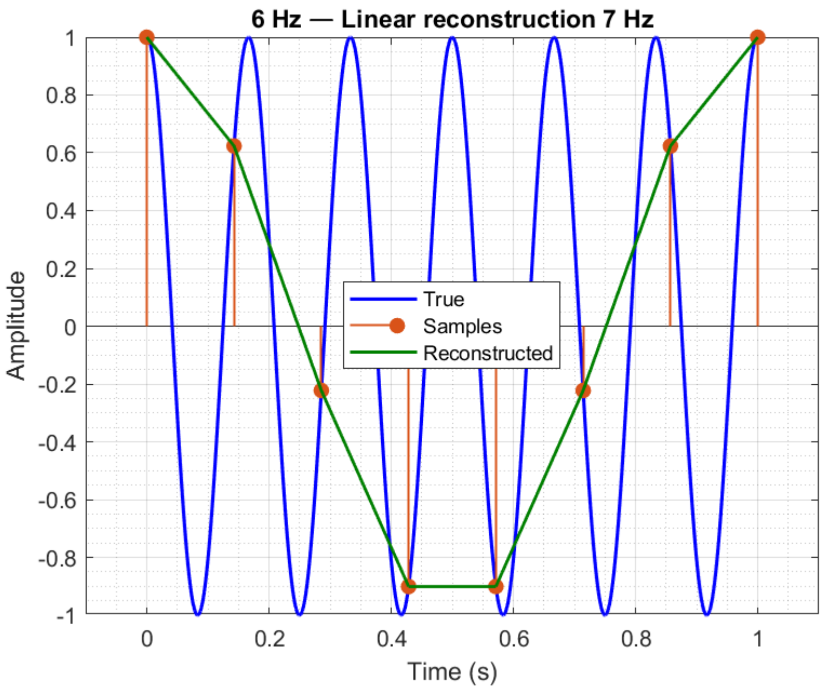

Figure 16: 6 Hz sampled with 7 Hz Reconstruction

$$y(t) = \cos(2\pi \times 3\,\text{Hz} \times t) \text{ where } \omega = 3\,\text{Hz}$$

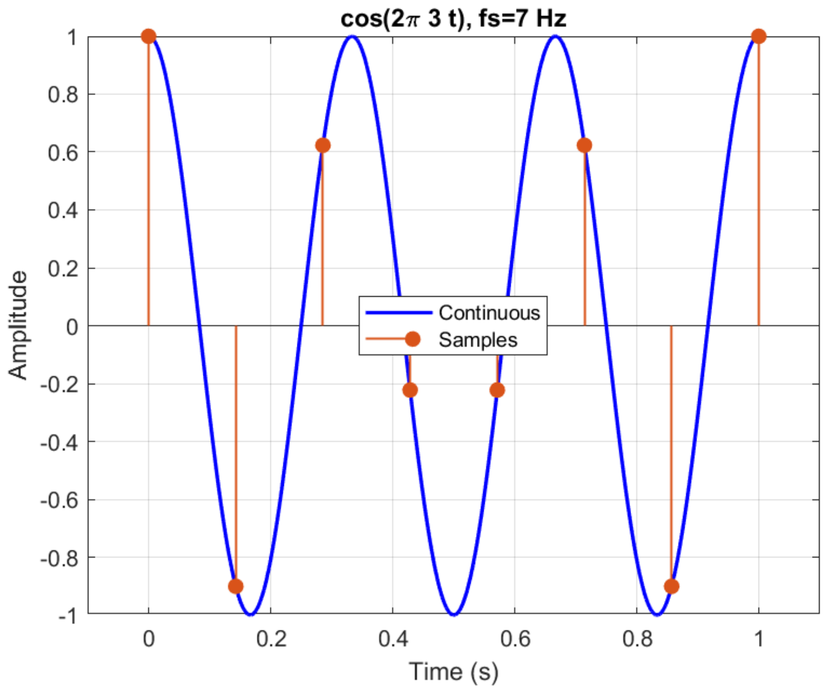

Figure 17: 3 Hz sampled with 7 Hz

If we fit reconstruct the signal from the sampled points.

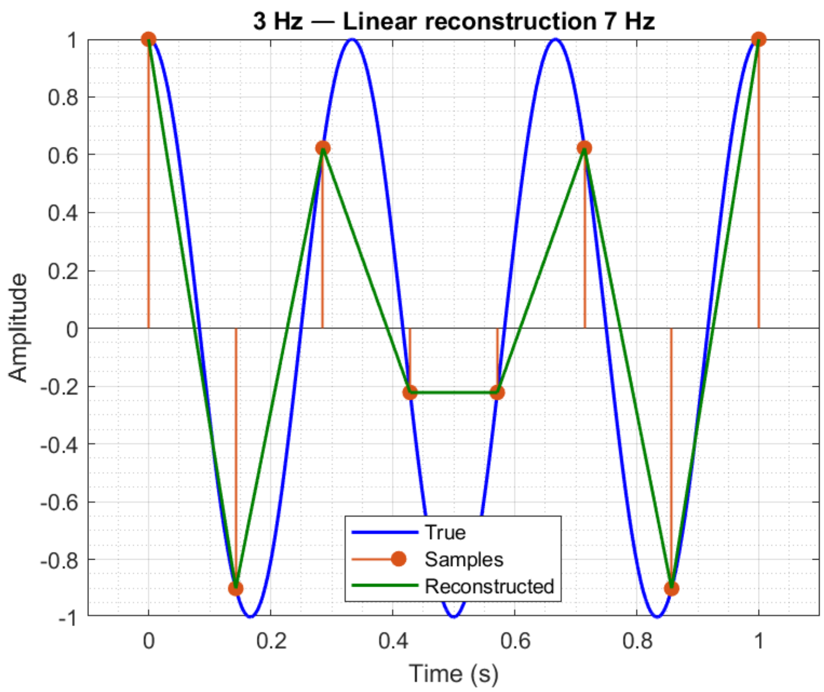

Figure 18: 3 Hz sampled with 7 Hz Reconstruction

$$y(t) = \cos(2\pi \times 1\,\mathrm{Hz} \times t) \text{ where } \omega = 1\,\mathrm{Hz}$$

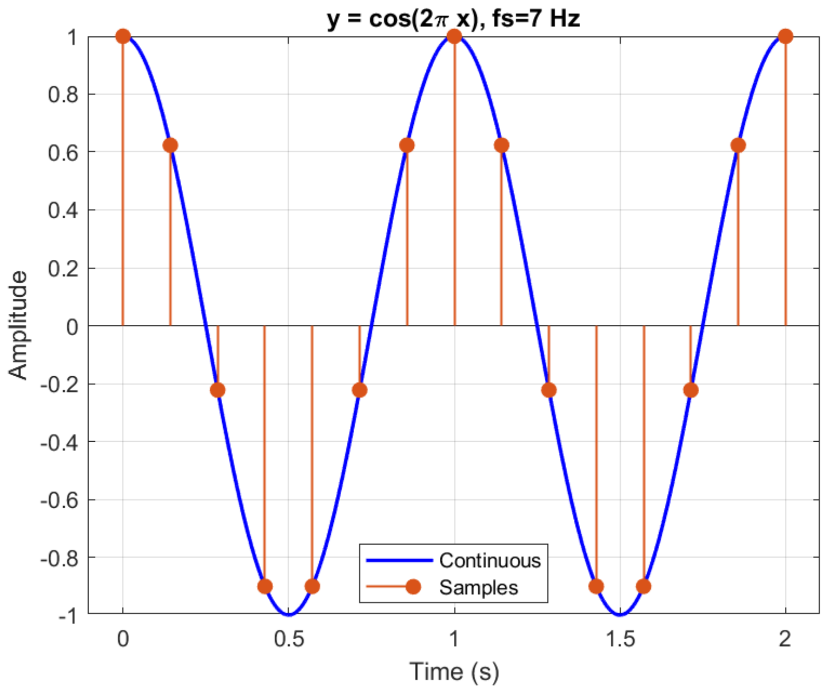

Figure 19: 1 Hz sampled with 7 Hz

If we fit reconstruct the signal from the sampled points.

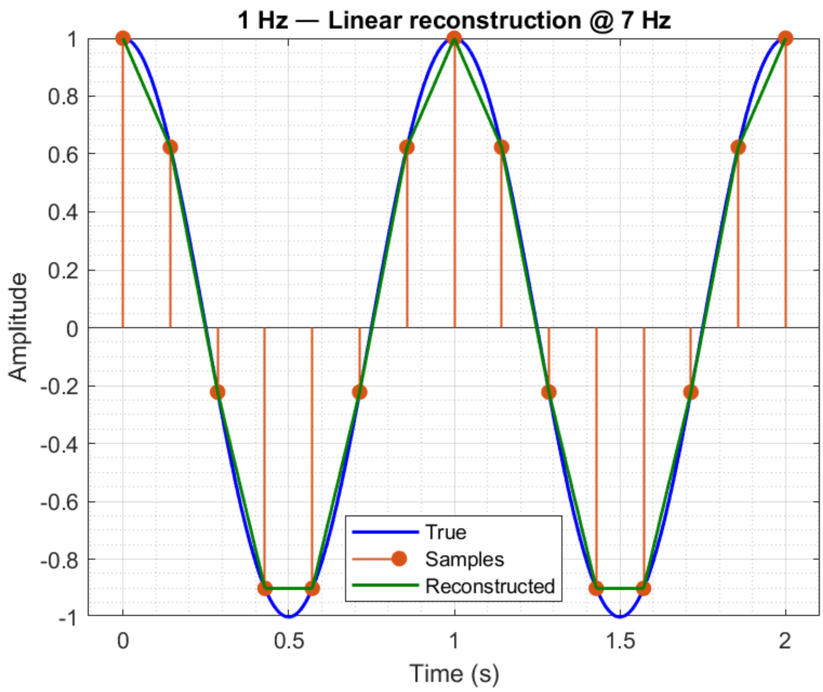

Figure 20: 1 Hz sampled with 7 Hz Reconstruction with Linear Fit

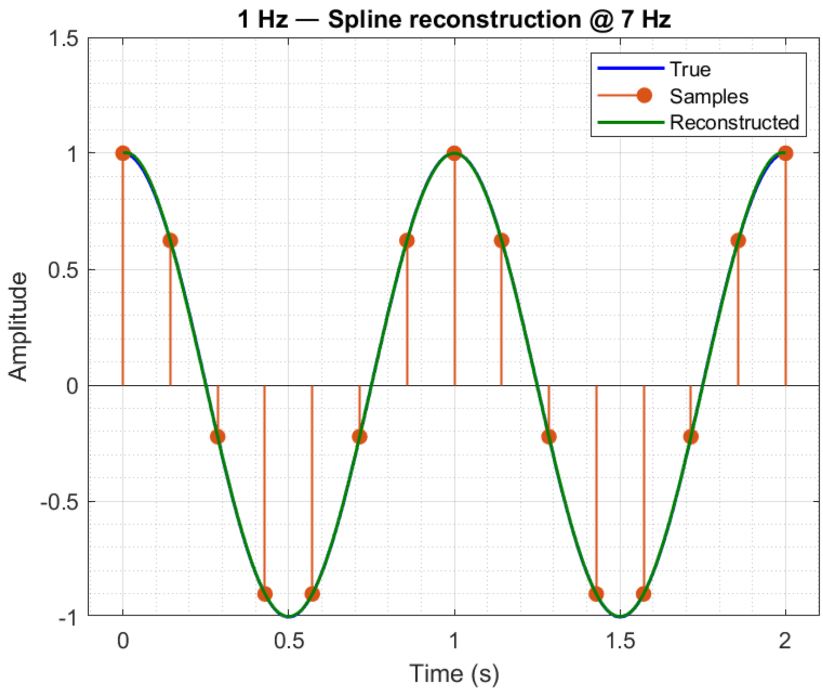

Figure 21: 1 Hz sampled with 7 Hz Reconstruction with Spline Fit

4 Appendix

4.1 Codes

```
function [hLine, hStem, t, ts, y, ys] = plot_true_with_samples(fun, fs, tspan, ...
    varargin)
    % ---------- Parse inputs ----------
    validateattributes(fs, {'numeric'}, {'scalar','real','positive'});
    validateattributes(tspan, {'numeric'}, {'numel',2,'real'});
    t0 = tspan(1); tf = tspan(2);
    if tf ≤ t0, error('tspan must satisfy [t0 tf] with tf > t0.'); end

    p = inputParser;
    addParameter(p, 'N', 4000, @(x) isnumeric(x) && isscalar(x) && x>0);
    addParameter(p, 'LineWidth', 1.6, @(x) isnumeric(x) && isscalar(x));
    addParameter(p, 'LineColor', [0 0 1], @(x) isnumeric(x) && numel(x)==3);
    addParameter(p, 'StemColor', [0.8500 0.3250 0.0980], @(x) isnumeric(x) && ...
        numel(x)==3);
    addParameter(p, 'Marker', 'filled', @(x) ischar(x) || isstring(x));
    addParameter(p, 'Title', '', @(x) ischar(x) || isstring(x));
    addParameter(p, 'XLabel', 'Time (s)', @(x) ischar(x) || isstring(x));
    addParameter(p, 'YLabel', 'Amplitude', @(x) ischar(x) || isstring(x));
    addParameter(p, 'Legend', true, @(x) islogical(x) && isscalar(x));
    parse(p, varargin{:});
    opt = p.Results;

    % ---------- Build time vectors ----------
    t  = linspace(t0, tf, opt.N);    % dense time for continuous curve
    Ts = 1/fs;
    % Include the endpoint tf if it aligns within numerical tolerance
    ts = t0:Ts:tf + 1e-12;           % sample times

    % ---------- Evaluate function ----------
    y  = fun(t);
    ys = fun(ts);

    % ---------- Plot ----------
    hold_state = ishold;
    hLine = plot(t, y, 'LineWidth', opt.LineWidth, 'Color', opt.LineColor); ...
        hold on
    hStem = stem(ts, ys, 'filled', 'LineWidth', 1.0, 'Color', opt.StemColor, ...
                 'BaseValue', 0);  %#ok<NASGU>  % vertical stems
    grid on
    xlabel(opt.XLabel); ylabel(opt.YLabel);

    if isempty(opt.Title)
        ttl = sprintf('Continuous function with samples @ f_s = %.3g Hz', fs);
    else
        ttl = opt.Title;
    end
    title(ttl);

    if opt.Legend
        legend({'Continuous','Samples'}, 'Location', 'best');
    end
```

```
    if ¬hold_state, hold off; end
end


fun = @(t) cos(2*pi*1*t);   % 1 Hz cosine
f_sampling = 7;
figure;
plot_true_with_samples(fun, f_sampling, [0 2], 'Title','y = cos(2\pi x), fs=7 ...
    Hz');


for f = [3 6 8 13 15]
    f_sampling = 7;
    figure;
    plot_true_with_samples(@(t) cos(2*pi*f*t), f_sampling, [0 1], 'Title', ...
        sprintf('cos(2\\pi %d t), fs=%d Hz', f,f_sampling));
end
```

```
    clc; clear; close all;
function [hTrue, hStem, hRec, t, ts, yTrue, yRec] = ...
    plot_reconstruction_with_samples(fun, fs, tspan, varargin)
    validateattributes(fs, {'numeric'}, {'scalar','real','positive'});
    validateattributes(tspan, {'numeric'}, {'numel',2,'real'});
    t0 = tspan(1); tf = tspan(2); if tf ≤ t0, error('tspan must satisfy tf > ...
        t0.'); end

    p = inputParser;
    addParameter(p, 'N', 4000, @(x) isnumeric(x) && isscalar(x) && x>0);
    addParameter(p, 'Method', 'linear', @(s) ...
        any(strcmpi(s,{'linear','spline','sinc'})));
    addParameter(p, 'SincHalfLobes', 8, @(x) isnumeric(x) && isscalar(x) && x>0);
    addParameter(p, 'TrueColor', [0 0 1], @(x) isnumeric(x) && numel(x)==3);
    addParameter(p, 'RecColor',  [0 0.5 0], @(x) isnumeric(x) && numel(x)==3);
    addParameter(p, 'StemColor', [0.8500 0.3250 0.0980], @(x) isnumeric(x) && ...
        numel(x)==3);
    addParameter(p, 'Title', '', @(x) ischar(x) || isstring(x));
    addParameter(p, 'Legend', true, @(x) islogical(x) && isscalar(x));
    parse(p, varargin{:});
    opt = p.Results;

    % Time and samples
    t  = linspace(t0, tf, opt.N);
    Ts = 1/fs;
    ts = t0:Ts:tf+1e-12;

    % True & samples
    yTrue = fun(t);
    ys    = fun(ts);

    % Reconstruction
    method = lower(opt.Method);
    switch method
        case 'linear'
            yRec = interp1(ts, ys, t, 'linear');
        case 'spline'
```

```
            yRec = interp1(ts, ys, t, 'spline');
        case 'sinc'
            yRec = local_sinc_reconstruct(t, ts, ys, opt.SincHalfLobes);
        otherwise
            error('Unknown Method.');
    end

    % Plot
    holdState = ishold;
    hTrue = plot(t, yTrue, 'Color', opt.TrueColor, 'LineWidth', 1.5); hold on
    hStem = stem(ts, ys, 'filled', 'Color', opt.StemColor, 'LineWidth', 1.0, ...
       'BaseValue', 0);
    hRec  = plot(t, yRec, 'Color', opt.RecColor, 'LineWidth', 1.4);
    grid on; grid minor;
    xlabel('Time (s)'); ylabel('Amplitude');

    ttl = opt.Title;
    if isempty(ttl), ttl = sprintf('Reconstruction: %s (f_s=%.3g Hz)', ...
       upper(method(1))+method(2:end), fs); end
    title(ttl)

    if opt.Legend
        lg = legend([hTrue hStem hRec], {'True','Samples','Reconstructed'}, ...
           'Location','best');
        set(lg, 'Interpreter','none');
    end
    if ¬holdState, hold off; end
end

% --- Local sinc interpolation (windowed, truncated) ---
function y = local_sinc_reconstruct(t, ts, ys, halfLobes)
    T = ts(2) - ts(1);
    y = zeros(size(t));
    % For each sample, add shifted, windowed sinc
    for k = 1:numel(ts)
        tau = t - ts(k);
        u = tau / T;
        % ideal sinc
        s = sinc(u);                    % MATLAB sinc(x) = sin(pi x)/(pi x)
        % cosine window to truncate around  halfLobes
        w = zeros(size(s));
        mask = abs(u) ≤ halfLobes;
        w(mask) = 0.5*(1 + cos(pi*u(mask)/halfLobes));
        y = y + ys(k) .* (s .* w);
    end
end


fun = @(t) cos(2*pi*1*t);
tspan = [0 2];
fs = 7;

figure;
plot_reconstruction_with_samples(fun, fs, tspan, 'Method','linear', ...
    'Title','1 Hz     Linear reconstruction @ 7 Hz');

figure;
```

```
plot_reconstruction_with_samples(fun, fs, tspan, 'Method','spline', ...
    'Title','1 Hz     Spline reconstruction @ 7 Hz');

figure;
plot_reconstruction_with_samples(fun, fs, tspan, 'Method','sinc', ...
    'SincHalfLobes', 8, ...
    'Title','1 Hz     Sinc reconstruction @ 7 Hz');


f_list = [3 6 8 13 15];
for fk = f_list
    funk = @(t) cos(2*pi*fk*t);
    fs = 7;
    figure;
    plot_reconstruction_with_samples(funk, fs, [0 1], 'Method','linear', ...
        'Title', sprintf('%d Hz     Linear reconstruction %d Hz', fk,fs));
end
```

```
    clc; clear; close all;
% --- SETTINGS ---
fs      = 1.5;              % sampling rate (Hz)
tspan   = [0 2];          % [t0 tf] seconds
N       = 4000;           % points for smooth plot

% Define functions (edit as needed)
fun1 = @(t) cos(2*pi*1*t);     % first function (e.g., 1 Hz)
fun2 = @(t) cos(2*pi*0.5*t);   % second function (e.g., 13 Hz)

% --- TIME & SAMPLING (only for first function) ---
t0 = tspan(1); tf = tspan(2);
t  = linspace(t0, tf, N);      % dense time for continuous curves
Ts = 1/fs;
ts = t0:Ts:tf + 1e-12;         % sampling instants (ONE set)

% --- EVALUATE ---
y1  = fun1(t);
y2  = fun2(t);
ys1 = fun1(ts);                % samples ONLY from first function

% --- FIGURE 1: first function + its samples ---
figure(1); clf; hold on
plot(t,  y1,  'b-', 'LineWidth', 1.6);              % blue line
stem(ts, ys1, 'r', 'filled', 'LineWidth', 1.0);     % red stems + filled red ...
   markers
grid on; xlabel('Time (s)'); ylabel('Amplitude');
title(sprintf('First function with samples, f_s = %g Hz', fs));
legend('f_1(t)', 'f_1 samples', 'Location', 'best');
hold off;

% FIGURE 2: first (blue), second (yellow), same red samples
figure(2); clf; hold on
plot(t, y1, 'b-', 'LineWidth', 1.6);                % blue
plot(t, y2, 'g-', 'LineWidth', 1.6);                % yellow
stem(ts, ys1, 'r', 'filled', 'LineWidth', 1.0);     % red stems reused
grid on; xlabel('Time (s)'); ylabel('Amplitude');
title(sprintf('f_1 , f_2 , and reused f_1 samples , f_s = %g Hz', fs));
```

```
legend('f_1(t)', 'f_2(t)', 'f_1 samples', 'Location', 'best');
hold off;
```