JOMO KENYATTA UNIVERSITY OF AGRICULTURE AND TECHNOLOGY

TELECOMMUNICATION SYSTEMS LAB REPORT

COURSE OF STUDY: BSc. TELECOMMUNICATION AND INFORMATION ENGINEERING

GROUP WORK: TELECOMMUNICATION LAB REPORT ETI 2506

LAB 3

STUDENT NAME	REGISTRATION NUMBER
1. PATIENCE SIMULI	ENE221-0110/2018
2. LEVIN WASIKE	ENE221-0300/2016
3. BENSON K. CHEGE	ENE221-0283/2016
4. WYCLIFFE WASONGA	ENE221-0122/2017

Introduction

Digital signal processing (DSP) techniques play a fundamental role in modern telecommunication systems, enabling the manipulation, analysis, and transmission of signals in various applications. One of the essential operations in DSP is the processing of discrete signals through techniques such as decimation and expansion. These operations allow us to modify the sampling rate and resolution of a signal while preserving its essential characteristics. This laboratory exercise focuses on the concepts of decimation and expansion and their practical implementation using MATLAB.

Decimation

Decimation involves the reduction of the number of samples in a discrete signal. The process is carried out by extracting every Nth sample from the original signal, resulting in a shorter sequence while maintaining the signal's integrity within the same domain range. This operation is particularly useful when aligning signals with varying resolutions to a common time domain. When applied to the discrete representation of a continuous real-world signal, decimation emulates the process of sampling the continuous signal to create a discrete sequence of samples.

MATLAB provides efficient methods to perform decimation. While a straightforward approach involves using a loop to extract samples with appropriate indexing, MATLAB's matrix operations offer a more efficient way to achieve this. The operation can be exemplified using functions like `linspace` and matrix indexing. The decimation process can be visualized by plotting the original continuous signal along with its corresponding discrete samples, showcasing the alignment of discrete samples within the same time domain.

Expansion

Contrary to decimation, expansion involves increasing the resolution of a signal. This is achieved by introducing intermediate values within the same domain. The simplest method of expansion is to insert zeros between the original samples. Alternatively, the same sample value can be replicated multiple times to create an expanded signal. For more sophisticated expansion,

interpolation techniques like spline interpolation can be employed, generating smooth intermediate values.

Objectives

The objectives of this laboratory exercise include:

1. To understand the concepts of decimation and expansion in digital signal processing.
2. To gain practical experience in implementing decimation and expansion operations using MATLAB.
3. To visualize the effects of different expansion techniques on signal resolution and characteristics.
4. To enhance skills in data visualization and interpretation through appropriate MATLAB plotting functions.

In the subsequent sections of this report, we will present the experimental procedure, results obtained from applying decimation and expansion techniques, and conclude with observations and insights drawn from the exercises. Through this hands-on exploration of decimation and expansion, a deeper understanding of signal processing concepts will be attained.

Methodology and Findings

The laboratory exercise was conducted using MATLAB to explore the concepts of decimation and expansion of discrete signals.

1. Exploring Decimation

Procedure

We started the session by exploring and understanding decimation from the examples given in the manual. The first was applying a decimation factor of 5 to a signal with values between 0 and 18. For this, the following code was used.

Step_5.m

/MATLAB Drive/Step_5.m

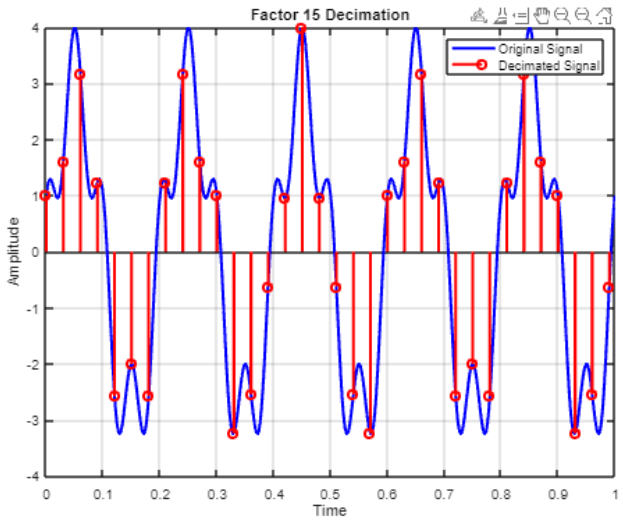
```
x = 0:18;
s = 5; %The decimation step value

new_signal = x(1: s: end);

disp("Input signal (x)");
disp(x);

disp("new_signal");
disp(new_signal);
```

The code generated the following result.

```
>> Step_5
Input signal (x)
     0     1     2     3     4     5     6     7     8     9    10    11    12    13    14    15    16    17    18

new_signal
     0     5    10    15
```

Next, we sampled a real signal by decimation using the code:

```
dt = 0.002;
t = 0:dt:1;
x = 3* sin(2*pi*5*t) + cos(2*pi*20*t);

s = 15;
x_dec = x(1:s:end);

%Plotting
figure;
plot(t, x, 'b', 'Linewidth', 2);
hold on;

t_dec = t(1:s:end);
stem(t_dec, x_dec, 'r', 'Linewidth', 2, 'Marker', 'o');
xlabel('Time');
ylabel('Amplitude');
title('Factor 15 Decimation');
grid on;

legend('Original Signal', 'Decimated Signal');
```

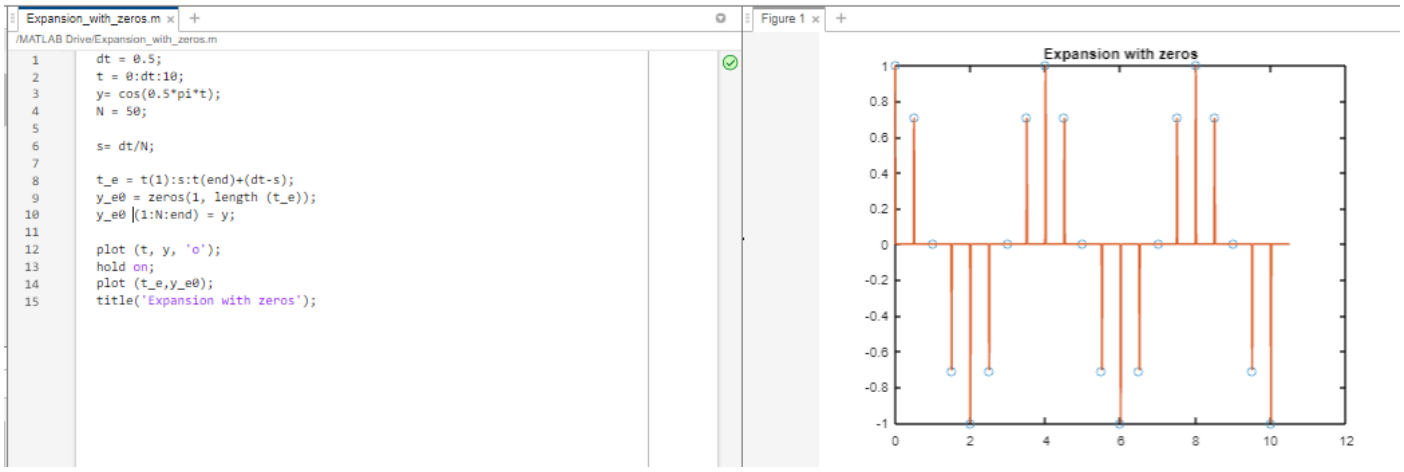


2. *Exploring Expansion*

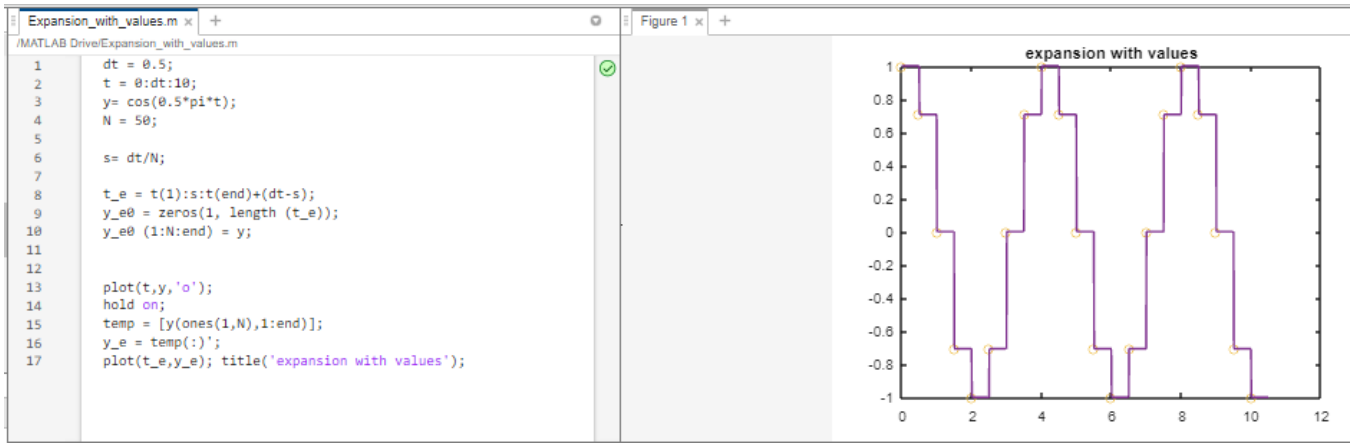
Next, we wrote code to generate the results presented in the laboratory manual for expansion with zeros, expansion with values, and expansion with spline. Code and corresponding results for these processes are shown below.

- Expansion with zeros

The code below generates the graph of expansion be zeros provided in the lab manual.

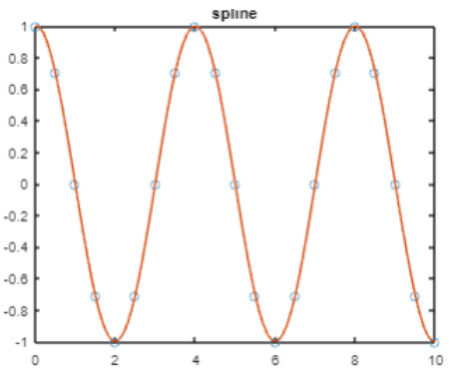


- Expansion with values



- Expansion with spline

```
dt = 0.5;
t = 0:dt:10;
y= cos(0.5*pi*t);
N = 50;

s= dt/N;

t_e = t(1):s:t(end)+(dt-s);
y_e0 = zeros(1, length (t_e));
y_e0 (1:N:end) = y;

t_sp = t(1):step:t(end);
y_sp = spline(t,y,t_sp);
plot(t,y,'o',t_sp,y_sp); title('spline');
```



EXERCISE 1

Exercise 1 Procedure

Plot the following x(t) real signals(generate matlab code). For each signal, select a suitable domain to spot the main signal features (e.g., edges, periodicity, values different from 0, etc.). Use the commands xlabel, axis, legend, axis, etc. to improve the graph presentation. Concerning the resolution, typically a value of 0.01 shall be adequate, although specific consideration shall be made on each signal (i.e. smaller resolution can be enough, or higher resolution is required)

$x(t) = 0.5\cos(15\pi t+\pi/8)u(t)$

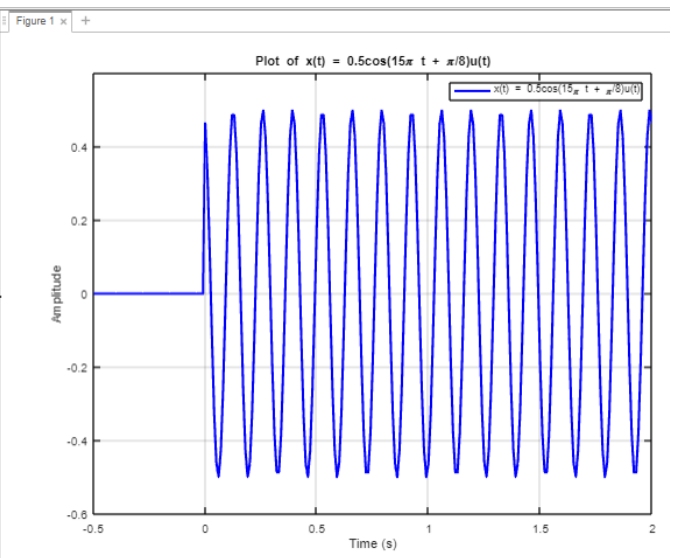
Results

The graph below is that of the signal $x(t) = 0.5\cos(15\pi t+\pi/8)u(t)$.

Original_Signal.m

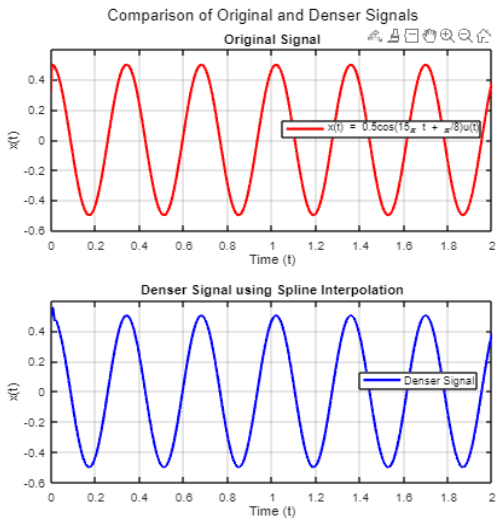
/MATLAB Drive/Original_Signal.m

```
% Define the time values
t = -0.5:0.01:2;

% Define the signal x(t)
x_t = 0.5 * cos(15 * pi * t + pi/8) .* (t >= 0);

% Plot the signal
plot(t, x_t, 'b', 'LineWidth', 2);
xlabel('Time (s)');
ylabel('Amplitude');
title('Plot of x(t) = 0.5cos(15\pi t + \pi/8)u(t)');
grid on;

% Set axis limits for better visualization
axis([min(t) max(t) -0.6 0.6]);

% Add a legend
legend('x(t) = 0.5cos(15\pi t + \pi/8)u(t)', 'Location', 'NorthEast');
```



The code below generates two graphs comparing the original and the dense signals.

```
% Define the original time range for the signal
t_original = -0.5:0.01:2;

% Calculate the original signal values
x_original = 0.5 * cos(15*pi*t_original*pi/8) .* (t_original >= 0);

% Define a denser time range for spline interpolation
t_dense = 0:0.001:2; % Smaller step size for denser points % Perform spline interpolation to gene
x_dense = spline(t_original, x_original, t_dense);

% Plot the original and denser signals figure;
subplot(2, 1, 1);
plot(t_original, x_original, 'r', 'Linewidth', 2);
xlabel('Time (t)');
ylabel('x(t)');
title("Original Signal");
grid on;
axis([0 max(t_original) -0.6 0.6]);
legend('x(t) = 0.5cos(15\pi t + \pi/8)u(t)', 'Location', 'best');

subplot(2, 1, 2);
plot(t_dense, x_dense, 'b', 'Linewidth', 2);
xlabel('Time (t)');
ylabel('x(t)');
title("Denser Signal using Spline Interpolation");
grid on;
axis([0 max(t_original) -0.6 0.6]);
legend('Denser Signal', 'Location', 'best');

sgtitle("Comparison of Original and Denser Signals");

%Adjust the figure layout
set(gcf, 'Position', [100, 100, 800, 800]);
```



Findings

Upon employing various expansion methods, we successfully augmented the resolution of the original signal. It became evident that the zero-padding technique introduced zeros in the intervals between samples, a straightforward approach that stretched the signal without altering its inherent shape. Conversely, the value replication method involved duplicating the existing signal values, thereby amplifying the signal's intensity.

The employment of spline interpolation yielded notably distinct outcomes. By employing this method, we generated seamless curves connecting the discrete samples. This sophisticated interpolation technique yielded a refined depiction of the continuous signal, capturing its nuances with greater precision.

Conclusion

In this laboratory exercise, we delved into the fundamental concepts of decimation and expansion in digital signal processing. Through hands-on experimentation using MATLAB, we

gained practical insights into how these operations can modify signal resolution while preserving essential characteristics.

Decimation enabled us to reduce the number of samples in a signal, aligning signals with varying resolutions to a common domain. The utilization of matrix operations in MATLAB provided an efficient approach to perform decimation. On the other hand, expansion techniques allowed us to increase resolution, and we observed that zero-padding, value replication, and spline interpolation each had distinct effects on the signal's appearance.

By investigating these operations and their outcomes, we gained a deeper understanding of their significance in telecommunication systems and signal processing applications. This exercise illuminated the crucial role of these techniques in manipulating and optimizing signals for various tasks, enriching our grasp of digital signal processing principles.