**Laboratory Report Cover Sheet**

|  |
| --- |
| SRM Institute of Science and Technology  College of Engineering and Technology  Department of Electronics and Communication Engineering |
| **18ECC204J DIGITAL SIGNAL PROCESSING**  **Fifth Semester, 2022-23 (Odd semester)** |

**Name :**

**Register No. :**

**Day / Session :**

**Venue :**

**Title of Experiment :**

**Date of Conduction :**

**Date of Submission :**

|  |  |  |
| --- | --- | --- |
| **Particulars** | **Max. Marks** | **Marks Obtained** |
| Pre lab and Post lab | 10 |  |
| Lab Performance | 10 |  |
| Simulation and results | 10 |  |
| Total | 30 |  |

**REPORT VERIFICATION**

**Staff Name :**

**Signature :**

**EXPERIMENT 13**

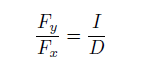
13a. INTERPOLATION IN TIME DOMAIN

Aim: To write code for interpolation of signal in SCILAB

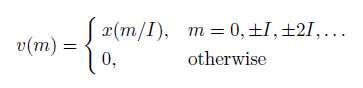
The idea of interpolation is a very familiar concept to most of us and has its origin in numerical analysis. Typically, interpolation is performed on a table of numbers representing a mathematical function. Such a table may be printed in a handbook or stored in a computer memory device. The interpolation, often simply linear (or straight line) approximation, creates an error called the *interpolation error*. The main difference between interpolation in digital signal processing and interpolation in numerical

analysis is that we will assume that the given data is bandlimited to some band of frequencies and develop schemes that are optimal on this basis, whereas a numerical analyst typically assumes that the data consists of samples of polynomials (or very nearly so) and develops schemes to minimize the resulting error. To motivate this concept of interpolation in signal processing, it is helpful to think of an underlying (or original) analog signal *x*a(*t*) that was sampled to produce the given discrete signal *x*(*n*). If the *x*a(*t*) was sampled at the minimum required rate, then, according to the sampling theorem, it can be recovered completely from the samples *x*(*n*). If we now sample this recovered analog signal, at say twice the old rate, we have succeeded in doubling the sampling rate or interpolating by a factor of 2 with zero interpolation error.

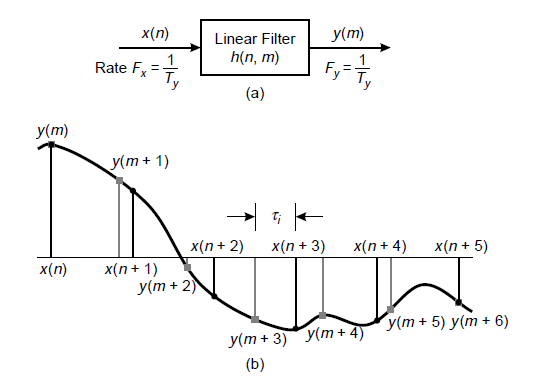
The process of sampling rate conversion in the digital domain can be viewed as a linear filtering operation, as illustrated in Figure. The input signal *x*(*n*) is characterized by the sampling rate *Fx* = 1*/Tx*, and the output signal *y*(*m*) is characterized by the sampling rate *Fy* =1*/Ty*, where *Tx* and *Ty* are the corresponding sampling intervals. In our treatment, the ratio *Fy/Fx* is constrained to be rational



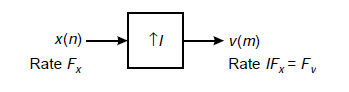
Let *v*(*m*) denote the intermediate sequence with a rate *Fy* = *IFx*, which is obtained from *x*(*n*) by adding *I −* 1 zeros between successive values of *x*(*n*). Thus,



and its sampling rate is identical to the rate of *v*(*m*). The block diagram of the upsampler is shown in Figure. Again, any system containing the upsampler is a time-varying system



**Figure1 :** Sampling rate conversion views as a linear filtering process



**Figure 2:** Interpolation by factor I

// PROGRAM FOR INTERPOLATION

// Program to do Interpolation of Signal

// clear work space variables

clf(); clc; clear; close;

// Set Initialization Variables

interp\_fac = 2;

samp\_freq = 100;

freq\_cycles = 3;

time\_index = 0:99;

// compute interpolation frequency with Sampling

inter\_samp\_freq = samp\_freq\*(interp\_fac);

// Map time axis for before and after interpolation

time\_axis = time\_index/samp\_freq;

new\_time\_index = 0:1/interp\_fac:99;

new\_time\_axis = new\_time\_index / samp\_freq;

// Define the input signal

inp\_sig = sin(2\*%pi\*freq\_cycles\*time\_axis);

// Do the interpolation using interp1 function on SCILAB

out\_sig = interp1(time\_axis,inp\_sig,new\_time\_axis);

// Display the output

subplot(1,2,1)

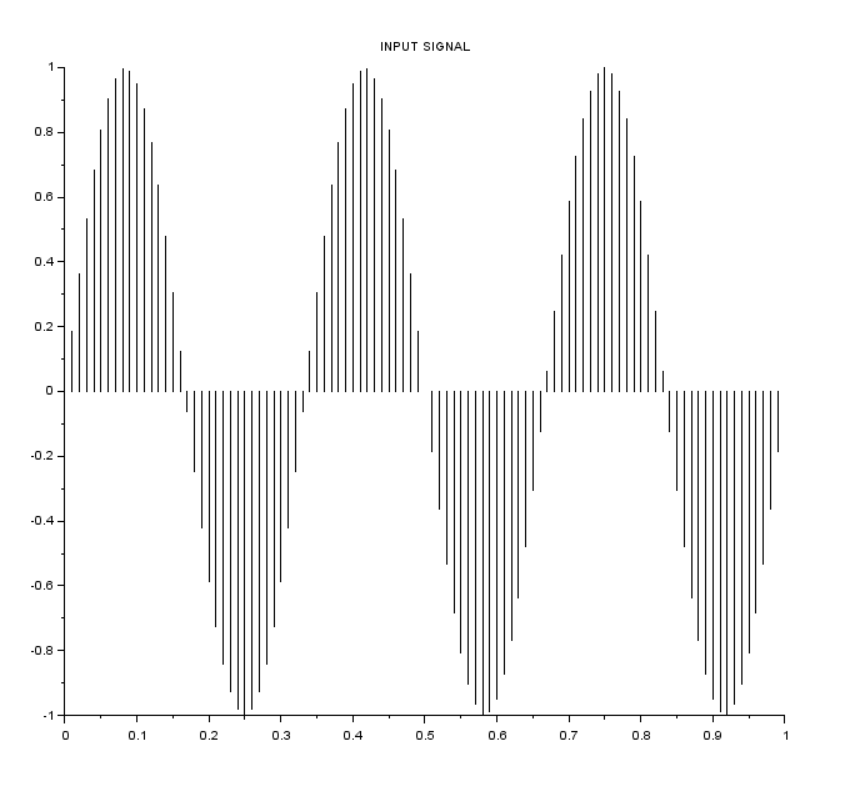
plot2d3(time\_axis,inp\_sig);

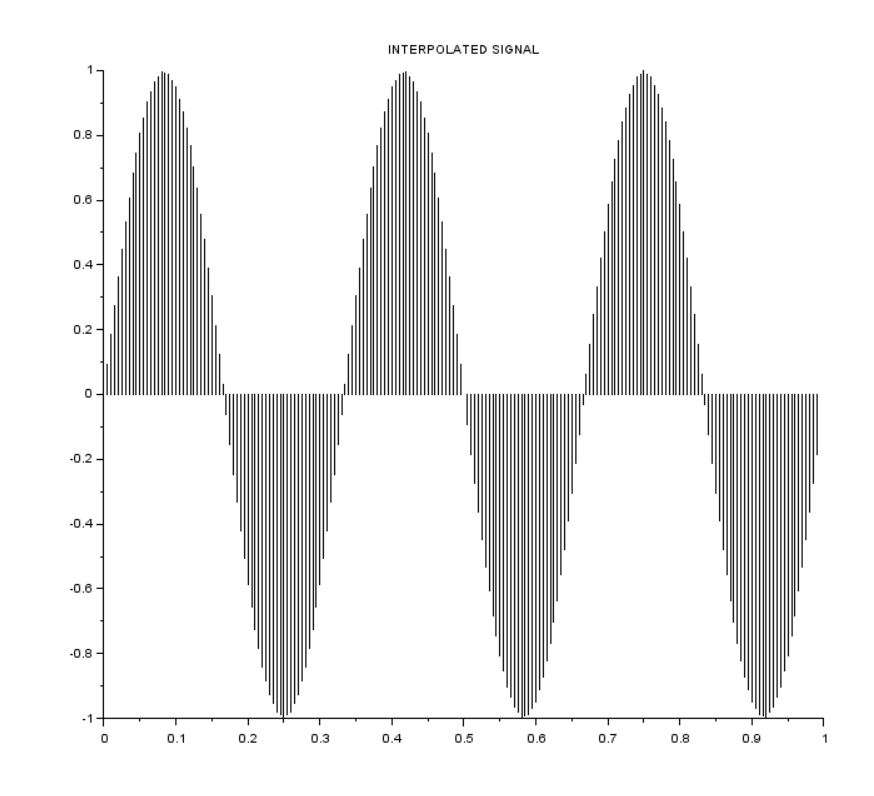
title('INPUT SIGNAL');

subplot(1,2,2)

plot2d3(new\_time\_axis,out\_sig);

title('INTERPOLATED SIGNAL')





Pre lab

1. What is difference between Upsampling & Interpolation?

2. Compare Decimation & Interpolation?

Post lab

1. Modify the program to work with 4 Cycle Cos Signal and present the code & output?
2. Modify the above program for interpolation factor I = 3 and present the code & output?

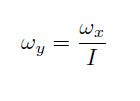
RESULT:

13b. INTERPOLATION IN THE FREQUENCY DOMAIN

Aim: To write code for interpolation of signal in frequency domain using SCILAB

Using fourier transform we will be able to do interpolation using zero padding in the frequency domain which will give exact bandlimited interpolation in the frequency domain. The DFT of the band limited signals for the entire non zero portions of x(n) extended by zeros yields exact interpolation of the complex spectrum.

Because fast fourier transform is so efficient, zero padding in FFT is highly practical method of interpolating spectra of finite duration signals and is used extensively in practice.



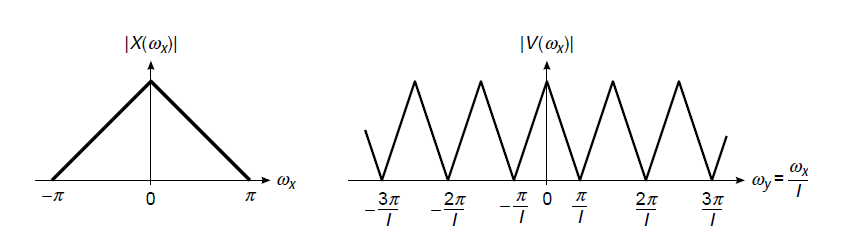


Figure 3: Frequency interpolation spectrum.

We observe that the sampling rate increase, obtained by the addition of *I −* 1 zero samples between successive values of *x*(*n*), results in a signal whose spectrum is an *I*-fold periodic repetition of the input signal spectrum.

// Program to do interpolation in the frequency domain

clf();clc;clear;close;

// Input sinusoidal signal

samp\_freq = 100;

sig\_freq = 3;

interp\_fac = 2;

// Time axis of the Sinusoidal Signal

time\_index = 0:99;

time\_axis = time\_index/samp\_freq;

inp\_sig = sin(2\*%pi\*sig\_freq\*time\_axis);

subplot(1,2,1)

plot2d3(time\_axis,inp\_sig );

// Frequency domain interpolation using zero padding

// Transpose of the signal

inp\_sig = inp\_sig(:);

// Take fft of the signal

sig\_fft = fft(inp\_sig);

// Do zero padding in frequency domain for interpolation

sig\_len = length(inp\_sig);

// compute how many zeros needed using the interpolation formula (N\*(I-1))

zeropad\_len = round(sig\_len\*(interp\_fac - 1));

// middle symmetry for splitting FFT in to two halves

mid\_symmetry = ceil((sig\_len+1)/2);

// Do zero padding here

frq\_interp\_sig = [sig\_fft(1:mid\_symmetry); zeros(zeropad\_len, 1); sig\_fft(mid\_symmetry+1:$)];

// Take inverse fourier transform

time\_interp\_sig = interp\_fac\* real(ifft(frq\_interp\_sig));

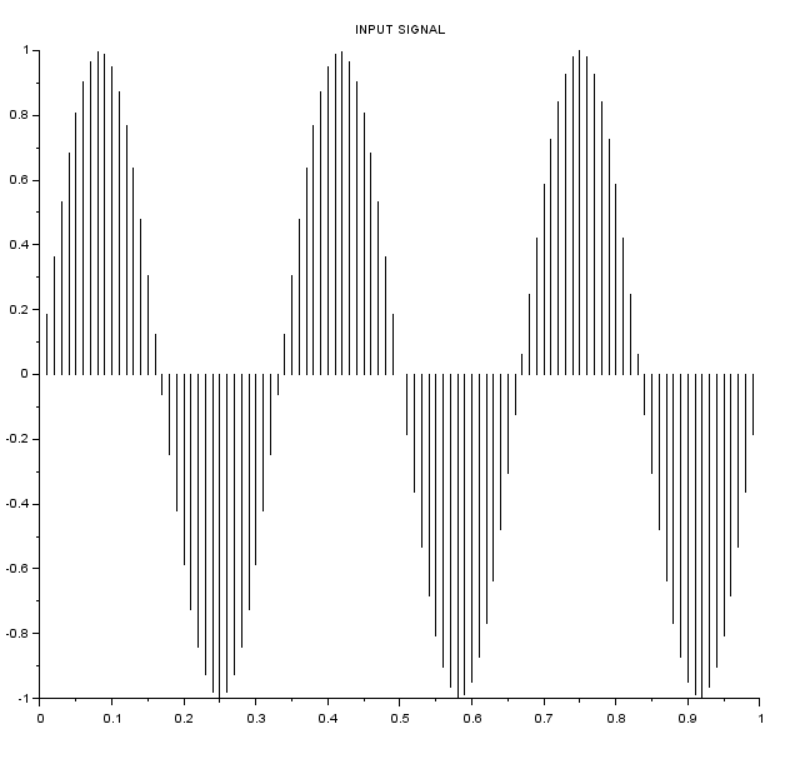
interp\_sig\_len = length(time\_interp\_sig);

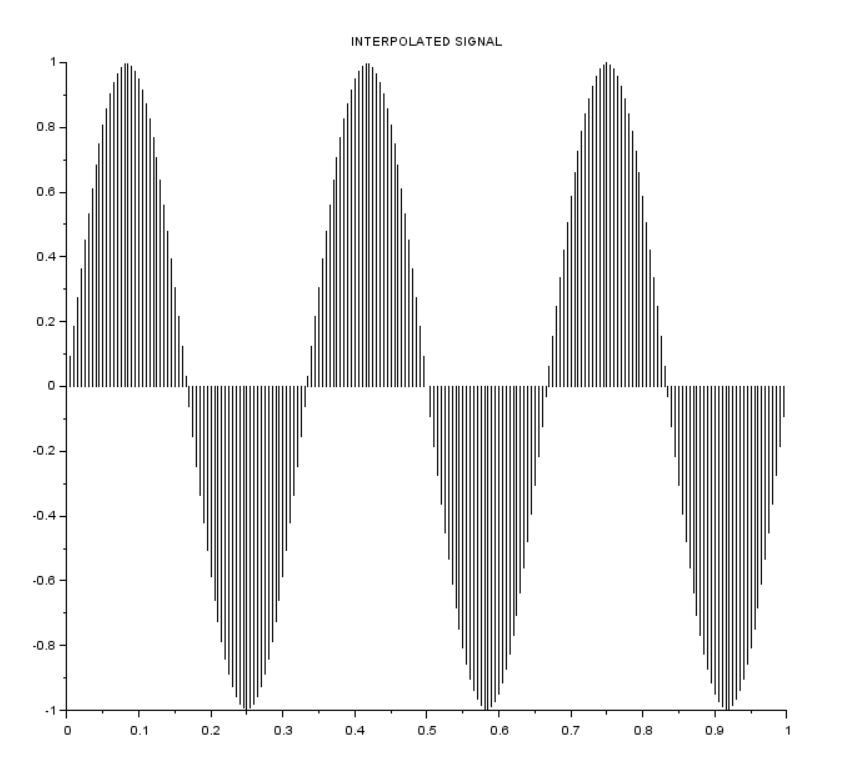
// compute new time axis

new\_time\_axis = (0:interp\_sig\_len-1)/(samp\_freq\*interp\_fac);

subplot(1,2,2)

plot2d3(new\_time\_axis, time\_interp\_sig)





PRE LAB:

1. Explain how the interpolation in frequency domain is advantageous over time domain interpolation?
2. Compare time domain & freq domain interpolation in coding logic?

POST LAB:

1. For the given code vary the length of the FFT to 256 and compute the output?
2. Change the amplitude of the output of the given code to [-2 2]?