🚴

Signal Processing - LAB 6 Report

Sreeja Guduri (2021102007)

In this lab, we compute the Continuous-time Fourier Transform of signals and process periodic signals with LTI systems acting as filters. We also perform reconstruction using various interpolation methods and study various properties of audio signals and quantization.

6.1 - Continuous-time Fourier transform

a) We implement a function to compute the continuous time FT of a given signal x(t) using the formula below.

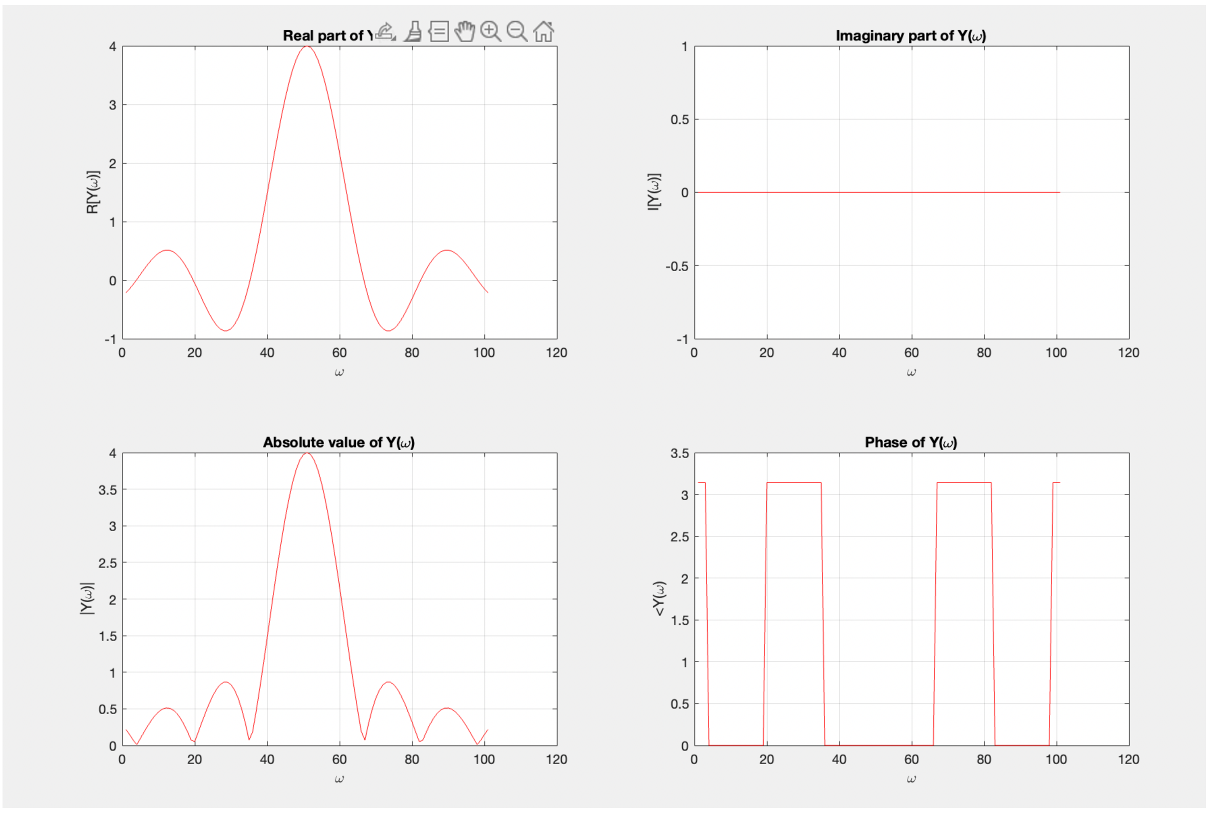
$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-i\omega t}\, dt$$

The function, *continuousFT(t,xt,a,b,w)* takes the following inputs:

- t → symbolic (syms) variable
- xt → input signal that is function of t
- a,b → [a,b] is the interval in which signal has non-zero values
- w → vector that contains the values of frequency where FT is to be computed

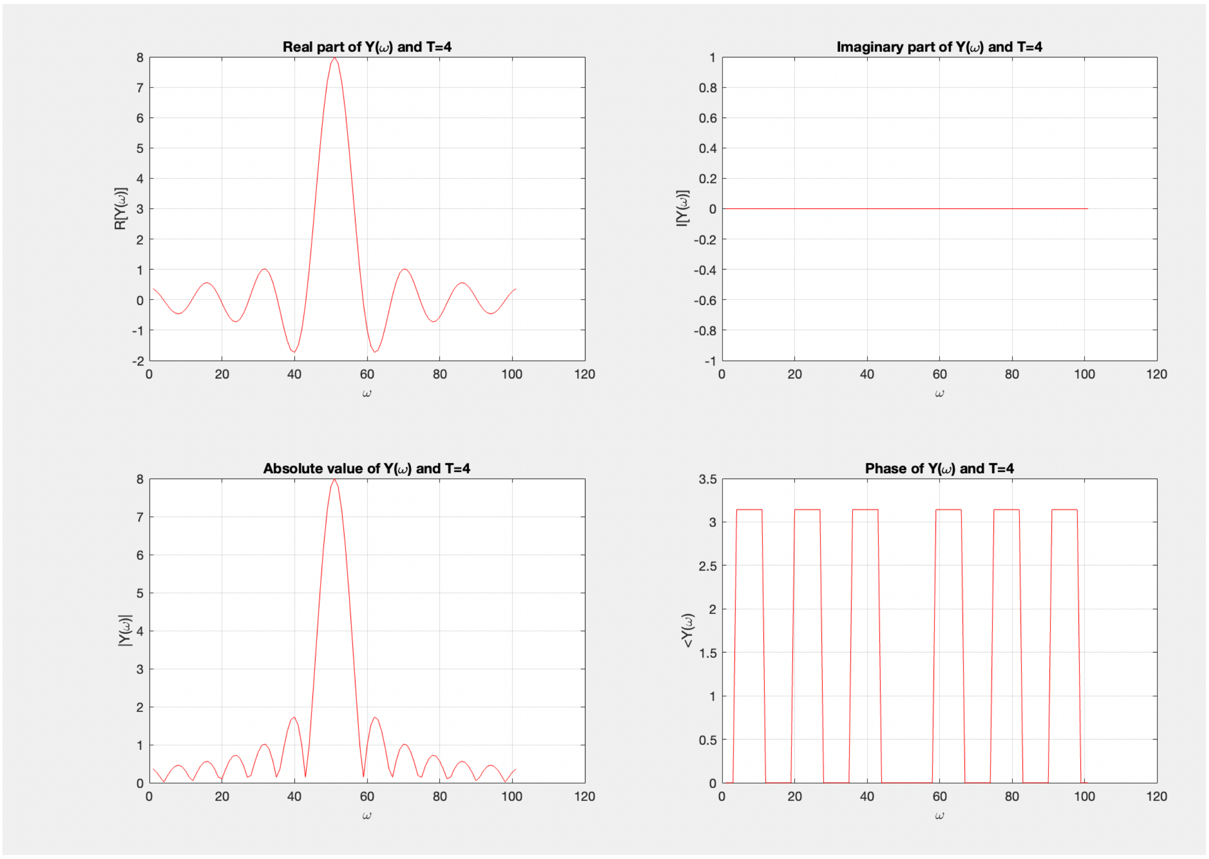
It returns a vector X which contains the FT of x(t) for each of the frequencies in the input vector ω

b) The function is called to perfom CTFT on a rectangular pulse of unit magnitude and the different graphs are plotted.



To compute the CTFT, we just integrate the exponential term from -T to T after which we plot the real, imaginary, absolute and phase part of the CTFT.

c) When the value of T is increased, we notice that there is an increase in the resolution of the Fourier transform. The property of the FT in use here is the time-scaling property.



d) Both the signals are expressed in terms of exponentials and hence, they will have similar FT plots, with small differences based on the powers of the exponents.

- the real plot has a sinusoidal shape
- the imaginary plot is 0 because the imaginary part doesn’t exist
- the absolute value plot has a positive sinusoidal shape
- the phase plot is in the form of rectangular pulses

e) The triangular pulse can be achieved by convolution of a rectangular pulse with itself. It can also be implemented using the piecewise function as:

```
xt = piecewise(abs(t)<1,1-abs(t),0);
```

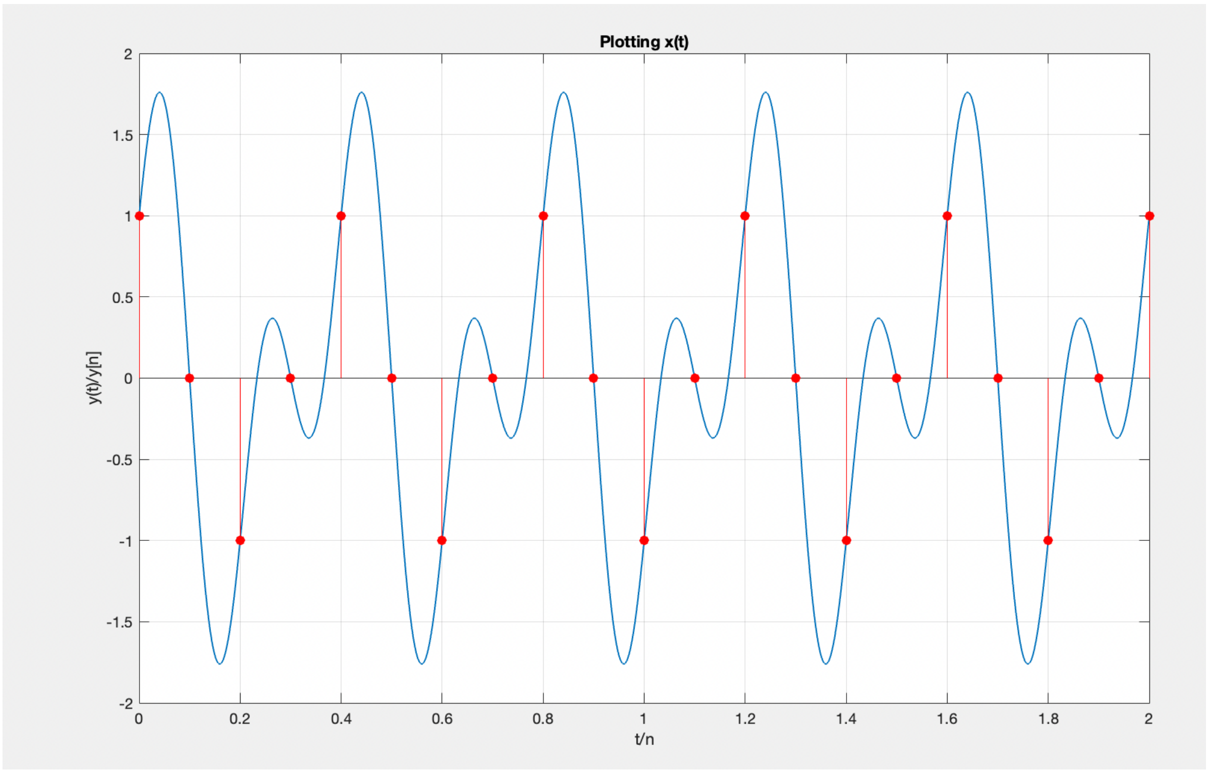
The FT plot is going to increase and then decrease by the same rate because of the difference in signs of the two halves of the triangle pulse.

6.2 - A signal and its samples

a) The signal $cos(5 * \pi * t) + sin(10 * \pi * t)$ is plotted on MATLAB. Since, it's a continuous-time signal that cannot be accurately represented on MATLAB, we use a fine time grid to plot the signal.

The signal is then sampled with a sampling interval of Ts = 0.1s

$x[n] = x(nT_s)$ and then is plotted using the *stem* command.



6.3 - Fast Fourier Transform (Radix-2)

a) We implement the *radix2fft* function which is used to find the FFT (using the decimation in time algorithm)

The function takes the following as input:

- x → N length vector which is the signal whose FFT is to be computed

This is a recursive function that returns the N-length FFT vector X.

When N=2, the FFT is just computed by multiplying the twiddle factor matrix with the signal x.

The output of the function is verified by comparing with the inbuilt *fft* function plot

6.4 - Filtering of periodic signals with LTI systems

To find the fourier coefficients of the output signal, we multiply the frequency response $H(\omega)$ with the fourier coefficients of the input signal

a) Ideal low pass filter (LPF):

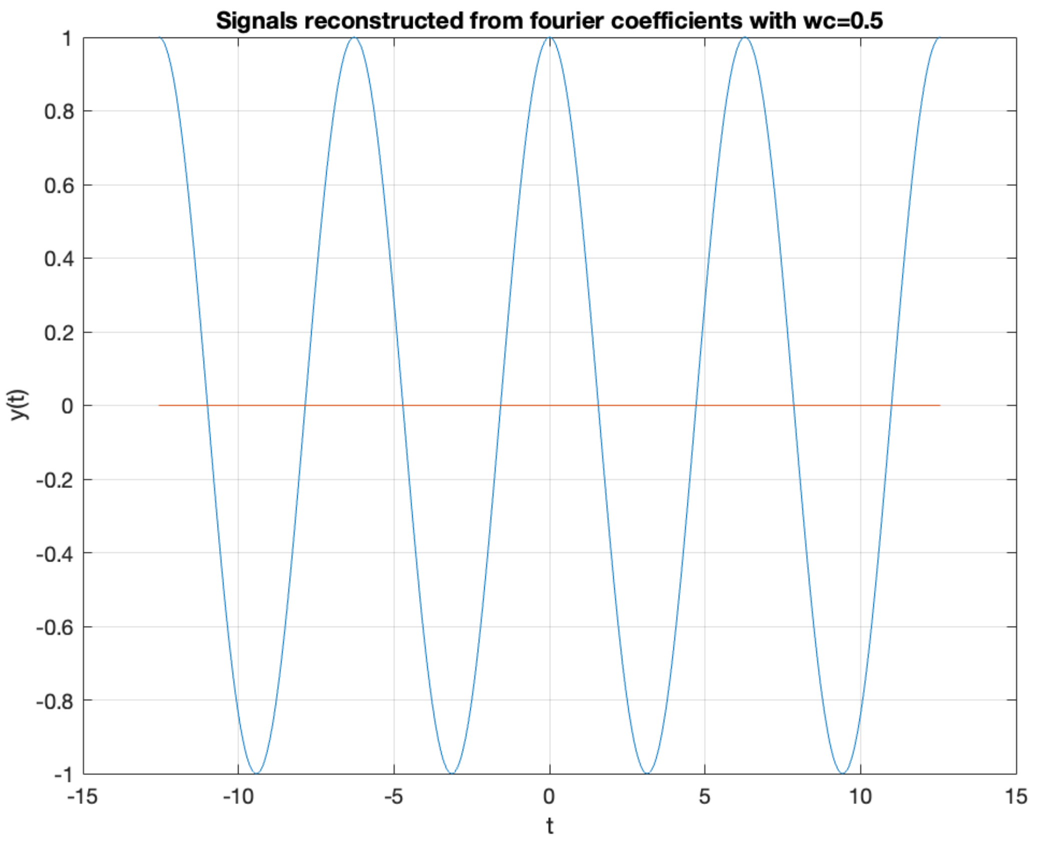
We implement the function *myLPF* that takes the following inputs:

- A → Fourier coefficients of the input signal
- w0_FS → fundamental frequency of input signal
- wc → cut-off frequency

The function should return the FS coefficients of the output signal

b) We use two previous functions we implemented called *partiralfouriersum* and *fourierCoeff* to reconstruct the signal from the fourier coefficients and to find the fourier coefficients of a signal respectively.

When we change the cutoff of the filter to 0.5 we notice that only one value of the FS coefficients of the signal that is below the cutoff is passed. However, since the value that is passed is zero, the output coefficients are also zero.



c) Ideal high pass filter (HPF):

We implement the function *myLPF* that takes the following inputs:

- A → Fourier coefficients of the input signal
- w0_FS → fundamental frequency of input signal
- wc → cut-off frequency

The function should return the FS coefficients of the output signal

The same script as the LPF is used to test the HPF implemented.

d) Non-ideal filter:

We implement the function *NonIdeal(A,w0_FS,G,a)* that takes the following inputs:

- A → Fourier coefficients of the input signal
- w0_FS → fundamental frequency of input signal

- G, a → parameters of the impulse response $H(\omega)$ of the filter

The function should return the FS coefficients of the output signal

The same script as the LPF is used to test the Non-ideal filter.

The complex-valued nature of the LTI system frequency is apparent in the output signal as a shift in phase when compared to the original.

e) The same filters are used on a different input signal $x(t) = sin(2t) + cos(3t)$ and the output signals are plotted.

6.5 - Reconstruction methods

The input signal used is $x[n] = sin[n]$ with n going from 0 to 5π in intervals of π/4

a) To get zero-order interpolation using the inbuilt MATLAB function *interp1,* we give the method as '*previous'* which fills the points between two data points with the previous samples values.

b) Linear is the default method for the *interp1* function and it performs linear interpolation of the input signal.

c) Sinc interpolation is done using the formula

$$x_r(t) = \sum_{n=-\infty}^{\infty} T_s\, x(nT_s)\, \frac{\sin\big(\omega_c(t - nT_s)\big)}{\pi(t - nT_s)}$$

Since, the infinite sum can't be implemented on computer, we take the sum over the interval of [0,2].

We implement the function *sinc_recon(n,xn,Ts,t_fine)* that takes the following inputs:

- n → integer indexes of the samples of the signal
- xn → the sampled signal x[n]

- Ts → sampling interval
- t_fine → time vector for the reconstructed signal

The function should return the reconstructed signal xr

d) The quality of reconstruction is visually better in the case of the sinc interpolation when compared to the other methods. This is because the sinc interpolation functions involves the sinc function and since the input is sinusoid, the output signal is highly accurate.

e) When the maximum absolute error (MAE) of the all the three reconstructed signals are found with respect to the original signal, we found that:

- Zero-order MAE = 1.7071
- Linear interpolation MAE = 1.8536
- Sinc interpolation MAE = 1.7509

6.6 - Sampling non band-limited signals

We perform sinc interpolation of the specified traingular pulse (with base [-1,1] and height 1) using different sampling interval Ts values and then plot the outputs.

6.7 - Audio signals

a) The audio signals were downloaded and the bit-rate was found to be:

- 1MG.wav → 256 kbps
- 2MG.wav → 512 kbps
- 5MG.wav → 1411 kbps
- 10MG.wav → 1411 kbps

b) The signals were loaded into MATLAB using the *audioread* function which also helped find the sampling frequency of the signals, which were found to be:

- 1MG.wav → 8000 Hz
- 2MG.wav → 16000 Hz
- 5MG.wav → 44100 Hz
- 10MG.wav → 44100 Hz

c)

Length of the loaded signal = l

Sampling frequency = F_s

Thus, sampling interval $T_s = 1/F_s$

Hence, duration of the audio signal = $T_s * l$

Thus, the duration of each audio signal is found to be:

- 1MG.wav → 33.5296 s
- 2MG.wav → 33.5296 s
- 5MG.wav → 29.6287 s
- 10MG.wav → 58.9936 s

d)

Bit-rate = b

Sampling frequency = F_s

Length of the loaded signal = l

Thus, duration of the audio signal = $1/F_s * l$

Hence, number of bits = $b * 1/F_s * l$

Thus, the number of bits in each audio signal is found to be:

- 1MG.wav → 8583584 bits
- 2MG.wav → 17167168 bits
- 5MG.wav → 4.1806*10^7 bits

- 10MG.wav → 8.3240*10^7 bits

The number of levels of quatization that the ADC can perform is given my 2^m where m is the number of bits used.

e) The function *sound( )* is used to listen to the audio files

f) When the audio files are played using a lower sampling frequency, we notice that the audio sounds slower and we can tell that some of the audio is lost (because it's not being sampled often).

On the contrary, when a higher sampling frequency is used, the audio signal sounds faster as more points in the audio signal is being sampled.

g) The property of the FT that helps explain this phenomenon is frequency scaling. Since the frequency changes, we notice that there is a change in the time-domain also.

6.8 - Aliasing

a) The Nyquist rate is found using the relation

$f_s = 2 * f_m$

where f_m → maximum frequency in the signal

For $x(t) = cos(5 * \pi * t)$

we write $x(t) = cos(2\pi * 5/2 * t)$

Thus, the frequency $f_m = f = 5/2$

Hence, the Nyquist rate, $f_s = f_m * 2 = 5Hz$

b) The sinc interpolated signals are plotted on a 2x2 plot and observed. We notice that as the sampling interval T_s increases, the signal becomes more accurate.