EXPERIMENT NUMBER : 9

EXPERIMENT NAME : RAISED-COSINE SPECTRUM

DATE : 19/12/2022, MONDAY

* AIM :

Implement raised-cosine filter using Python and study its properties by plotting the impulse response with various roll-off factors.

* SOFTWARE REQUIRED :

1. Anaconda3 2021.11 (Python 3.9.7 64-bit)
2. Spyder 5.1.5, Integrated Development Environment (IDE)

* THEORY :

The raised-cosine filter is a filter frequently used for pulse-shaping in digital modulation due to its ability to minimise intersymbol interference (ISI).

Its name stems from the fact that the non-zero portion of the frequency spectrum of its simplest form ($\beta=1$) is a cosine function, 'raised up' to sit above the f (horizontal) axis.

The raised-cosine filter is an implementation of a low-pass Nyquist filter, i.e., one that has the property of vestigial symmetry. This means that its spectrum exhibits odd symmetry about $1/2T$, where T is the symbol-period of the communication system.

PTO

The impulse response of such a filter is given by,

$$h(t) = \begin{cases} \frac{\pi}{4T} \operatorname{sinc}\left(\frac{1}{2\beta}\right), & t = \pm \frac{T}{2\beta} \\ \frac{1}{T} \operatorname{sinc}\left(\frac{t}{T}\right) \frac{\cos\left(\frac{\pi \beta t}{T}\right)}{1 - \left(\frac{2\beta t}{T}\right)^2}, & \text{otherwise} \end{cases}$$

in terms of the normalised sinc function.
Here, this is the "communications sinc" $\sin(\pi x)/(\pi x)$ rather than the mathematical one.

→ **Roll-off factor** -
The roll-off factor, β, is a measure of the excess bandwidth of the filter, i.e., the bandwidth occupied beyond the Nyquist bandwidth of $1/2T$. Some authors use $\alpha = \beta$.

The graph shows the amplitude response as β is varied between 0 and 1, and the corresponding effect on the impulse response. As can be seen, the time-domain ripple level increases as β decreases. This shows that the excess bandwidth of the filter can be reduced, but only at the expense of an elongated impulse response.

→ **$\beta = 0$** -
As β approaches 0, the roll-off zone becomes infinitesimally narrow, so the impulse response approaches $h(t) = \frac{1}{T} \operatorname{sinc}\left(\frac{t}{T}\right)$.

Hence, it converges to an ideal or brick-wall filter in this case.

→ **$\beta = 1$** -
When $\beta = 1$, the non-zero portion of the spectrum is a pure raised cosine.

→ Bandwidth -

The bandwidth of a raised cosine filter is most commonly defined as the width of the non-zero frequency-positive portion of its spectrum, i.e.,

$$BW = \frac{R_s}{2}(\beta+1), \quad (0<\beta<1)$$

As measured using a spectrum analyzer, the radio bandwidth B in Hz of the modulated signal is twice the baseband bandwidth BW, i.e.,

$$B = 2BW = R_s(\beta+1), \quad (0<\beta<1)$$

→ Application -

When used to filter a symbol stream, a Nyquist filter has the property of eliminating ISI, as its impulse response is zero at all nT (where n is an integer), except n=0.

Therefore, if the transmitted waveform is correctly sampled at the receiver, the original symbol values can be recovered completely.

However, in many practical communications system, a matched filter is used in the receiver, due to the effects of white noise. For zero ISI, it is the net response of the transmit and receive filters that must equal H(f):

$$H_R(f) \cdot H_T(f) = H(f)$$

And therefore:

$$|H_R(f)| = |H_T(f)| = \sqrt{|H(f)|}$$

These filters are called root-raised-cosine filters. Raised cosine is a commonly used apodization filter for fiber Bragg gratings.

PYTHON CODE:

```
import matplotlib.pyplot as plt # Provides an implicit way of plotting
import numpy as np # Support for large, multi-dimensional arrays and matrices
import warnings
warnings.filterwarnings('ignore') # Never print matching warnings

# Compute DFT coefficients using linear transformation method:
def DFT (x, plot_name):

    # Compute W(N) 1D Array:
    r1 = c1 = len(x)
    wn = []
    for i in range (r1):
        for j in range (c1):
            wn.append (np.exp (-2j * np.pi * i * j /len(x)))

    # numpy.reshape() is used to give a new shape to an array without changing its data.

    wn_multidim = np.reshape (wn, (r1, c1)) # An N*N W(N) matrix
    r2 = len(x); c2 = 1
    x_multidim = np.reshape (x, (r2, c2)) # An N*1 x(N) matrix

    # Compute X(N) = W(N) * x(N), an N*1 matrix
    fourier_transform_multidim = [[0] * c2] * r1 # NULL Multidimensional Array
    fourier_transform_l_t = []
    for i in range (r1):
        for j in range (c2):
            fourier_transform_multidim [i][j] = 0
```

```
for k in range (cl):
    fourier_transform_multidim[i][j] +=
wn_multidim[i][k] * float (x_multidim[k][j])
  fourier_transform_l_t.append(abs(fourier_
transform_multidim[i][j]))

plt.subplot (1,2,2)
plt.xlabel ("Frequency (f)");
plt.ylabel ("Frequency Response, H(f)")
plt.title (str (plot_name) + "\n" + "in Frequency
Domain")
plt.stem (np.arange (0,
len (fourier_transform_l_t)), fourier_transform_l_t)
plt.grid (True); plt.tight_layout(); plt.show()

# The overlap is fine, as long as your pulse shaping
filter meets this one criterion: all of the pulses must add
up to zero at every multiple of our symbol period T,
except for one of the pulses.

num_symbols = 10
sps = 8  # 8 samples per symbol
bits = np.random.randint (0, 2, num_symbols) # our
data to be transmitted, 1's and 0's.
x = np.array ([])

for bit in bits:
    pulse = np.zeros (sps)
    pulse [0] = bit * 2 - 1 # set the first value to either a
1 or -1
    x = np.concatenate ((x, pulse)) # Add the 8 samples
to the signal
```

```
plt.figure(); plt.plot(x, '.-')
plt.title("Pulse Train of Impulses")
plt.grid(True); plt.show()

"""

bits: [0, 1, 1, 1, 1, 0, 0, 0, 1, 1]
BPSK symbols: [-1, 1, 1, 1, 1, -1, -1, -1, 1, 1]
Applying 8 samples per symbol: [-1, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, ...]
"""


# Create our raised-cosine filter :-
beta = [0, 0.3, 0.6, 0.9]  # Range of β values
for i in range(len(beta)):

    Ts = sps  #Assume sample rate is 1 Hz, so sample
period is 1, and *symbol* period is 8
    t = np.arange(-50, 51)  # Remember it's not inclusive of
final number
    h = 1/Ts * np.sinc(t/Ts) * np.cos(np.pi * beta[i] * t/Ts) /
(1 - (2 * beta[i] * t/Ts)**2)


    plot_name = "For β = " + str(beta[i])
    plt.figure(); plt.subplot(1,3,1)
    plt.xlabel("Time (t)"); plt.ylabel("Impulse Response,
h(t)")
    plt.plot(t, h, '.'); plt.grid(True)
    plt.title(str(plot_name) + "\n" + "in Time Domain")

    DFT(h, plot_name)

"""

    from scipy.fft import fft  # Compute the 1-D discrete
Fourier Transform
```

```
y = fft(x); plt.subplot(1,3,2)

plt.xlabel("Frequency (f)"); plt.ylabel("Frequency
Response, H(f)")
plt.title(str(plot_name) + "\n" + "Fast Fourier
Transform")
plt.stem(np.arange(0, len(y)); plt.grid(True)

from scipy.fft import ifft #Compute the 1-D inverse
discrete Fourier transform
yinv = ifft(y); plt.subplot(1,3,3)

plt.xlabel("Frequency (f)"); plt.ylabel("Frequency
Response, H(f)")
plt.title(str(plot_name) + "\n" + "Inverse Fast
Fourier Transform")
plt.stem(np.arange(0, len(yinv)), yinv);
plt.grid(True); plt.tight_layout(); plt.show()

"""
```

RESULT:

Thus, implemented raised-cosine filter using Python and studied its properties. All the simulation results were verified successfully.

REFERENCES:

① PySDR: A Guide to SDR and DSP using Python - https://pysdr.org/content/pulse_shaping.html

② Raised-cosine filter. From Wikipedia, the free encyclopedia

③ Digital Communications, John G. Proakis, Masoud Salehi, Fifth Edition. McGraw-Hill Higher Education, 2008
ISBN-10: 0072957166
ISBN-13: 978-0072957167

DEMO GRAPHS :

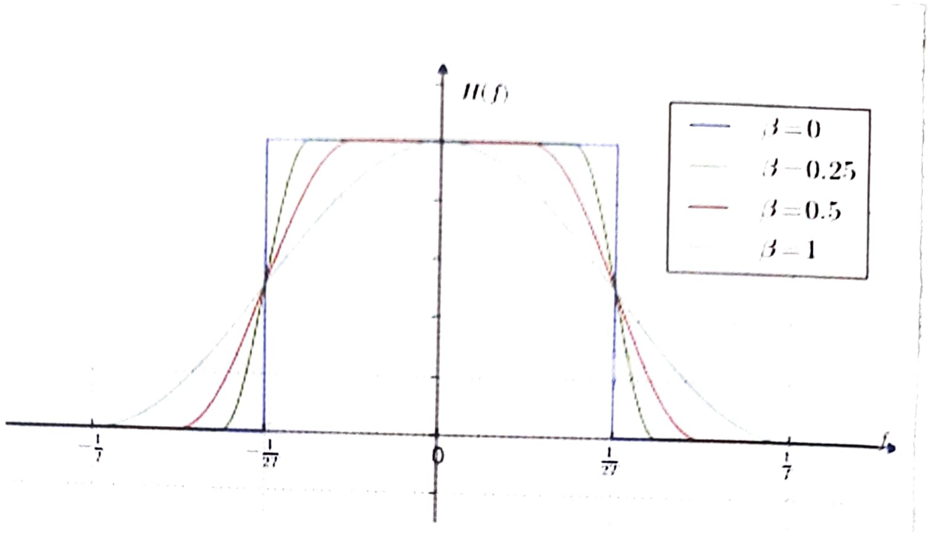


Figure 1.- Frequency response of raised-cosine filter with various roll-off factors

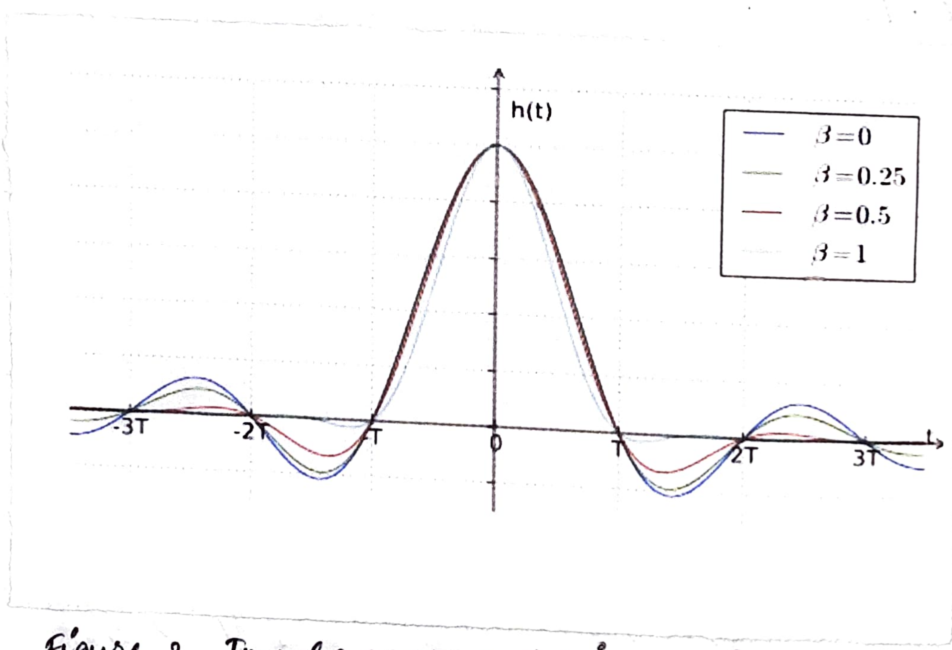


Figure 2 - Impulse response of raised-cosine filter with various roll-off factors

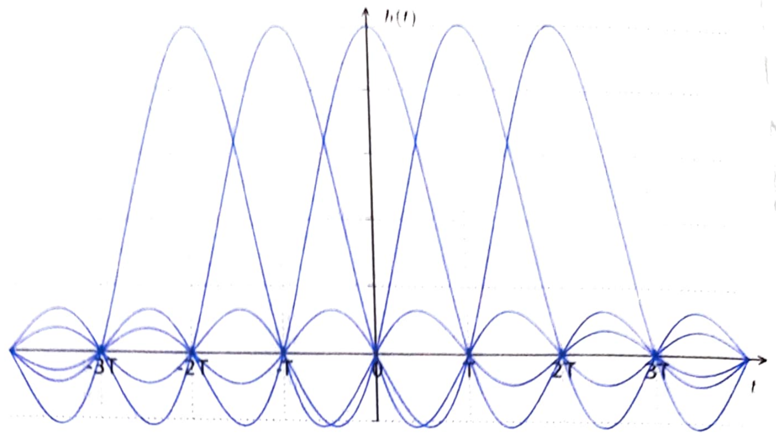


Figure 3 - consecutive raised-cosine impulses, demonstrating zero-ISI property

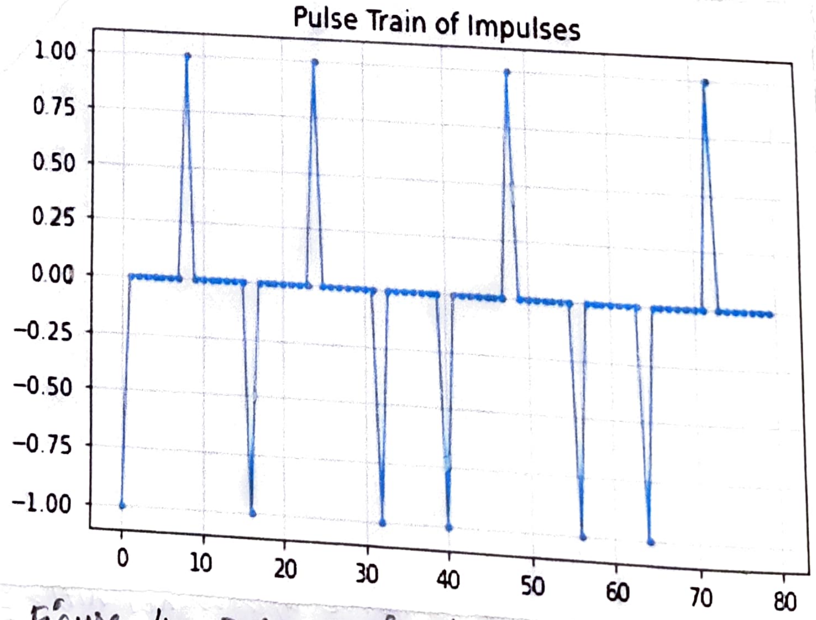
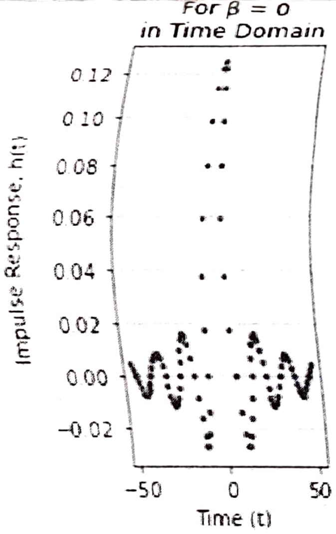


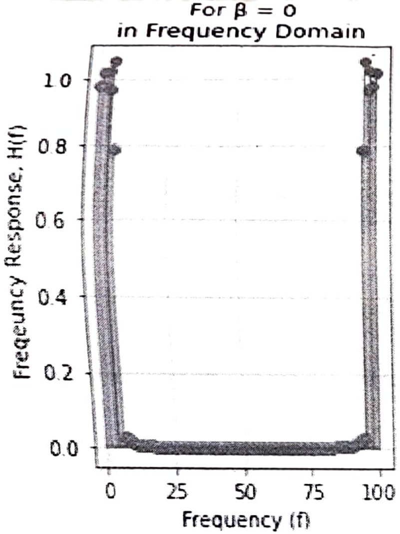
Figure 4- Pulse Train of Impulses

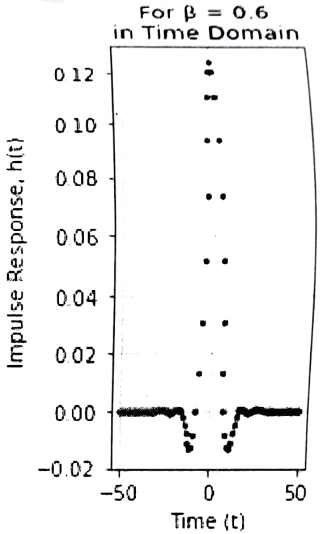
(i) our data to be transmitted, 1's & 0's.

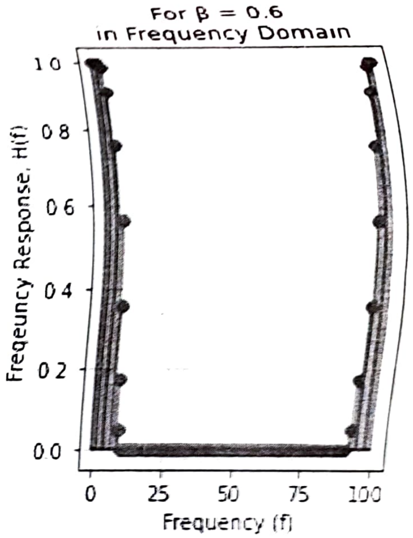
(ii) set the first value to either a 1 or -1.

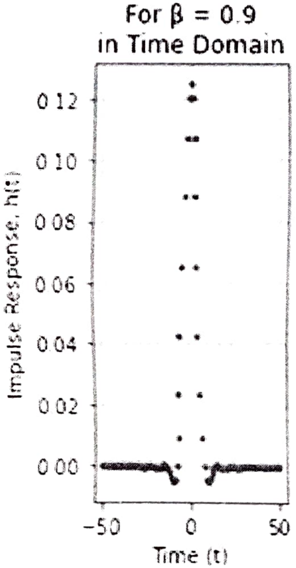
(iii) Add the 8 samples to the signal.

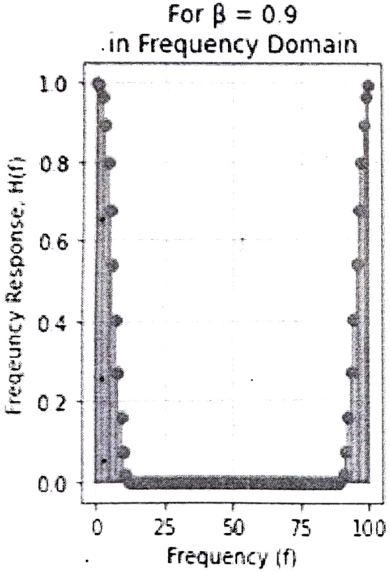

For β = 0
in Time Domain
Impulse Response, h(t)
Time (t)

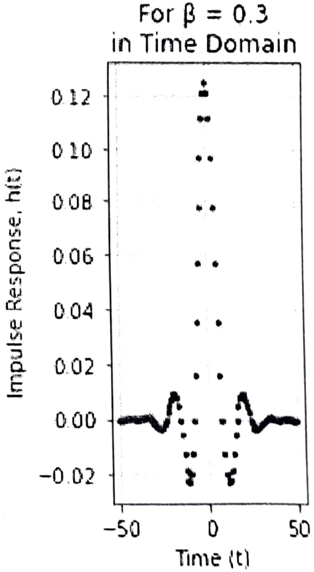

For β = 0
in Frequency Domain
Freqeuncy Response, H(f)
Frequency (f)

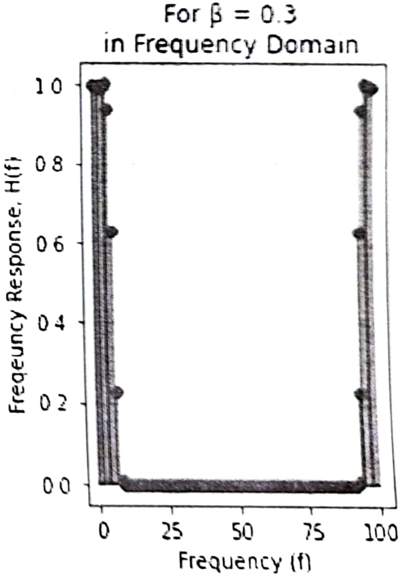

For β = 0.6
in Time Domain
Impulse Response, h(t)
Time (t)


For β = 0.6
in Frequency Domain
Freqeuncy Response, H(f)
Frequency (f)


For β = 0.9
in Time Domain
Impulse Response, h(t)
Time (t)


For β = 0.9
in Frequency Domain
Freqeuncy Response, H(f)
Frequency (f)


For β = 0.3
in Time Domain
Impulse Response, h(t)
Time (t)


For β = 0.3
in Frequency Domain
Freqeuncy Response, H(f)
Frequency (f)