BU-ALI SINA UNIVERSITY

SIGNAL PROJECT

Signal Project Report

Student:

Mehrdad Shahidi

supervised by
Dr.Mahdi Abbasi

February 15, 2020

Contents

1 Introduction **2**

2 Question1 **2**

2.1 Q1.1 2

2.1.1 Unit 2

2.1.2 Ramp 3

2.2 Q1.2 5

2.3 Q1.3 7

2.4 Q1.4 8

3 Question2 **10**

3.1 Q2.1 10

3.2 Q2.2 11

3.3 Q2.3 13

4 Question3 **14**

4.1 Q3.1 14

4.2 Q3.2 15

4.3 Q3.3 16

5 Question4 **17**

5.1 Q4.1 17

5.2 Q4.2 17

6 Question5 **18**

6.1 Q5.1 18

6.2 Q5.2 18

1 INTRODUCTION

This article is a report for singal project. we have used **Numpy , scipy, matplotlib** for this project.

2 QUESTION1

2.1 Q1.1

In this section, we implement Unit and Ramp function with different frequencies and compare them.

2.1.1 Unit

expression **2.1** represents a unit step function and **Listing 1** is an equivalent code in python for this function that get an array of float numbers and return an array with the same size of input as output that has the same logic as expression **2.1**

$$u(t) = \begin{cases} 0 & : t < 0 \\ 1 & : t \geq 0 \end{cases} \tag{2.1}$$

```python
def unit(t): return np.array([1 if i >= 0 else 0 for i in t])
```

Listing 1: Unit_step function

As you can see in **Listing 2** first we plot the unit step with the sample rate 10 per second and then with the rate 100 per second and **Figure 2.1** shows both plots

```python
gridsize = (2, 1)
fig = plt.figure(figsize=(20, 15))
# sampling with rate: 10 sample per second
t1 = np.linspace(-0.5, 0.5, 10)
# settings for axes1
ax1 = plt.subplot2grid(gridsize, (0, 0))
ax1.set_xticks(np.arange(-1, 1, step=0.1))
ax1.set_title("$u(t)$ with sampling rate of $F_s =10$", fontsize=20)

# sampling with rate: 100 sample per second
t2 = np.linspace(-0.5, 0.5, 100)
# settings for axes2
ax2 = plt.subplot2grid(gridsize, (1, 0))
ax2.set_xticks(np.arange(-1, 1, step=0.1))
ax2.set_title("$u(t)$ with sampling rate of $F_s =100$", fontsize=20)

#plot and show
ax1.plot(t1, unit(t1), '.-')
```

```
ax2.plot(t2, unit(t2), '.-')
#plt.savefig('doc/images/unit.pgf')
fig.show()
```

Listing 2: Code for plotting unit_step

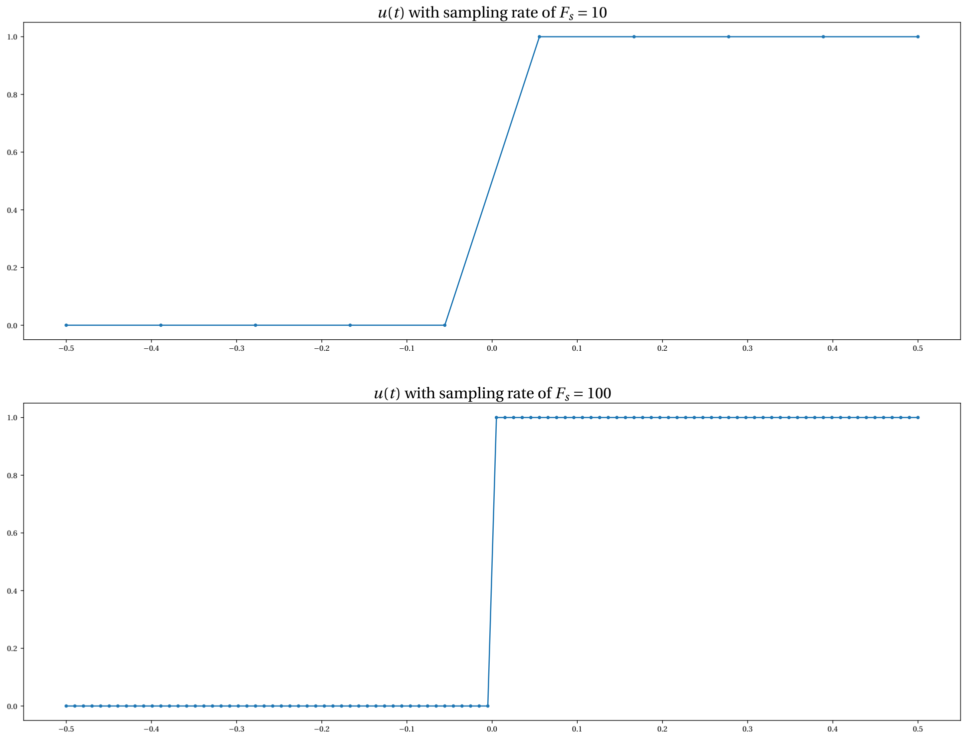

Figure 2.1: Result plot of **Listing 2**

2.1.2 Ramp

Like the previous section we write the **Listing 3** python code that represents **expression 2.2**

$$r(t) = \begin{cases} t & : t \geq 0 \\ 0 & : t < 0 \end{cases} \tag{2.2}$$

```
def ramp(t): return np.array([i if i >= 0 else 0 for i in t])
```

Listing 3: Unit_step function

So we plot the ramp function with 2 different sampling rate as above in **Listing 4**

```
gridsize = (2, 1)
fig = plt.figure(figsize=(20, 15))
# sampling with rate: 10 sample per second
t1 = np.linspace(-0.5, 0.5, 10)
ax1 = plt.subplot2grid(gridsize, (0, 0))
ax1.set_xticks(np.arange(-1, 1, step=0.1))
ax1.set_title('$r(t)$ with sampling rate of $F_s =10$', fontsize=20)

# sampling with rate: 100 sample per second
t2 = np.linspace(-0.5, 0.5, 100)
ax2 = plt.subplot2grid(gridsize, (1, 0))
ax2.set_xticks(np.arange(-1, 1, step=0.1))
ax2.set_title('$r(t)$ with sampling rate of $F_s =100$', fontsize=20)

#plot and show
ax1.plot(t1, ramp(t1), '.-')
ax2.plot(t2, ramp(t2), '.-')
fig.show()
```

Listing 4: Code for plotting ramp

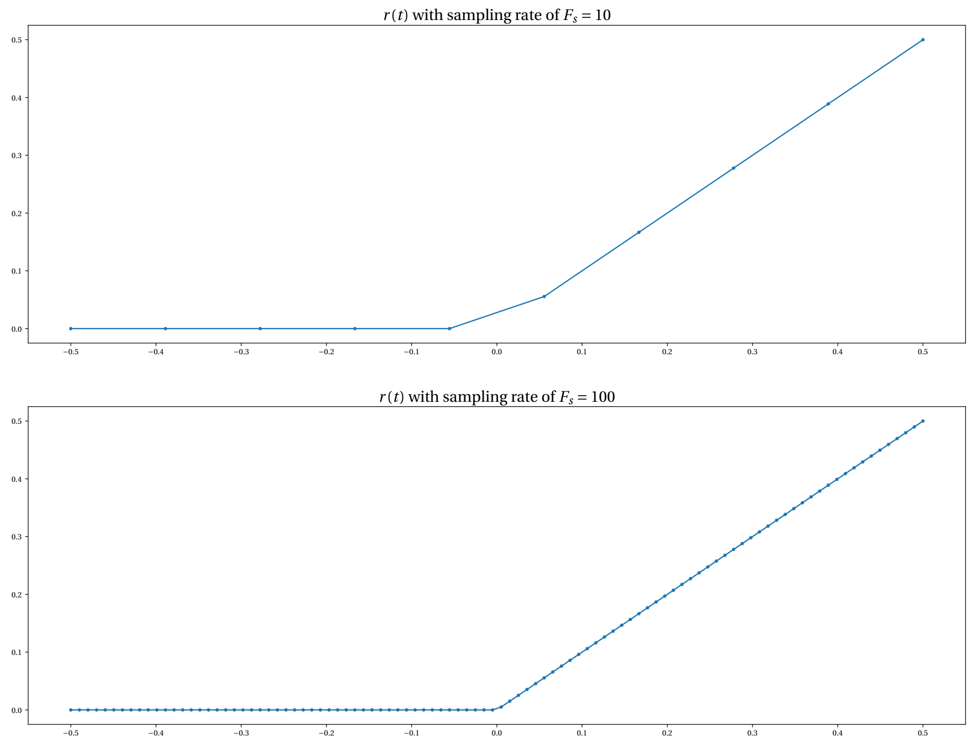

Figure 2.2: Result plot of **Listing 4**

2.2 Q1.2

In this section, we are going to plot the example in **Figure 2.3** , using ramp and unit_step that we implemented in the previous sections

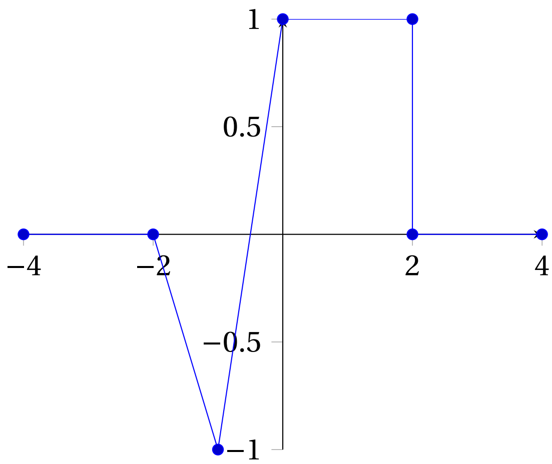

Figure 2.3: example graph of the siganl $x(t)$

first we break down the function to:
$x_1(t) = -r(t+2) * u(-t-1)$
$x_2(t) = (r(2*t+2)-1) * (u(t+1)-u(-t))$
$x_3(t) = u(t) - u(t-2)$
$x(t) = x_1(t) + x_2(t) + x_3(t)$
then we define a python function for each of them as you can see in **Listing 5** and in **Listing 6** we plot them

```
def x1(t): return -ramp(t+2) * unit(-t-1)


def x2(t): return (ramp(2*t+2) - 1) * (unit(t+1)-unit(t))


def x3(t): return unit(t) - unit(t-2)


def x(t): return x1(t)+x2(t)+x3(t)
```

Listing 5: Functions for $x_1(t)$,$x_2(t)$,$x_3(t)$,$x(t)$

```
# sampling with rate : 100 sample per second
t = np.linspace(-5, 5, 1000)
# Figure settings
gridsize = (3, 3)
fig = plt.figure(figsize=(10, 10))
axs = []
```

```
axs.append(plt.subplot2grid(gridsize, (0, 0), rowspan=2, colspan=3))
axs.append(plt.subplot2grid(gridsize, (2, 0)))
axs.append(plt.subplot2grid(gridsize, (2, 1)))
axs.append(plt.subplot2grid(gridsize, (2, 2)))
for i in range(4):
    axs[i].set_xticks(np.arange(-4, 5, step=1))
    axs[i].set_title(
        f'$x{"_"+str(i) if i>0 else "=x_1 + x_2 + x_3"}$', fontsize=16)

#plot and show
axs[0].plot(t, x(t))
axs[1].plot(t, x1(t))
axs[2].plot(t, x2(t))
axs[3].plot(t, x3(t))
fig.show()
```

Listing 6: Code for plotting functions $x_1(t)$,$x_2(t)$,$x_3(t)$,$x(t)$

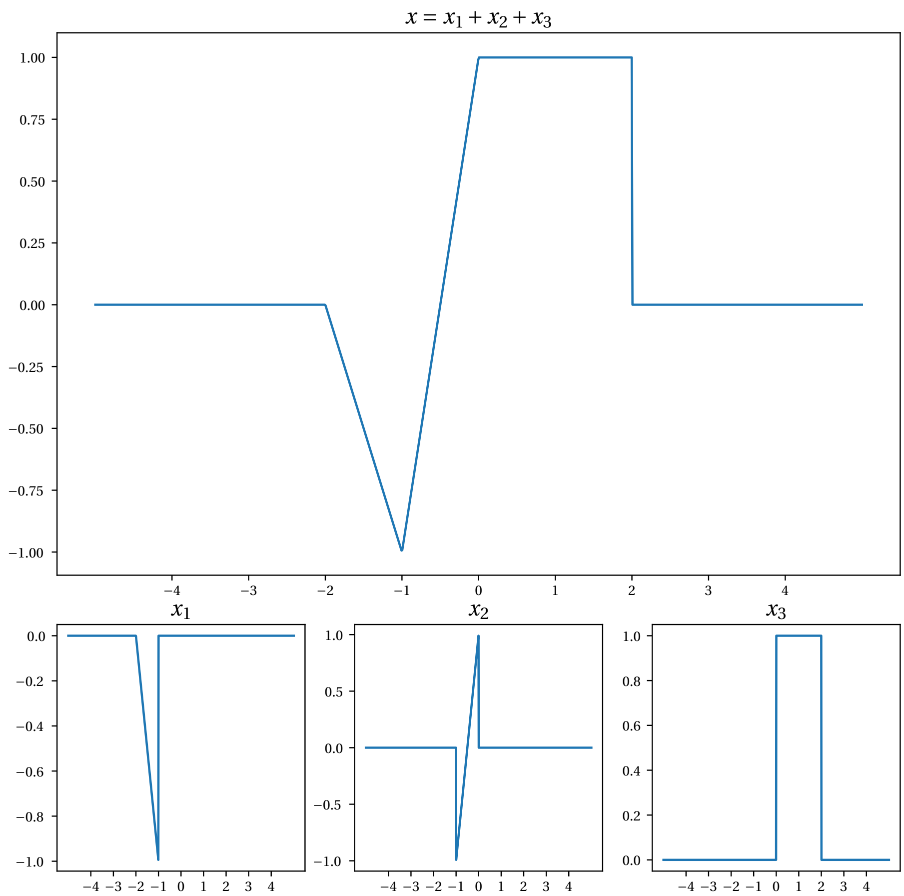

Figure 2.4: Result plot of **Listing 6**

2.3 Q1.3

Int this section, we are going to plot the $y_1(t)$ in the **Figure 2.5** by shifting and compressing $x(t)$ in the previous section

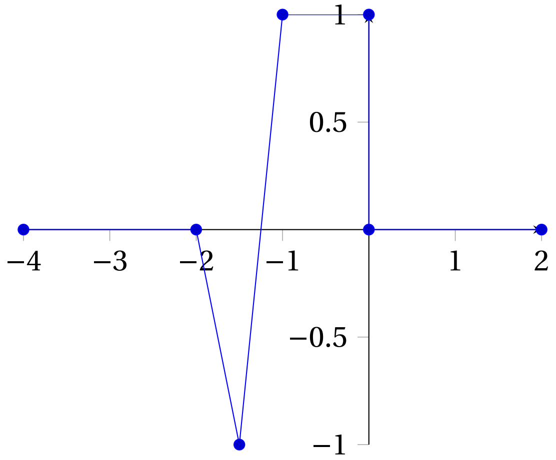

Figure 2.5: example graph of the siganl $y_1(t)$

first we shift $x(t)$, 2 unit left and then compress it with sacle of 2 as you see in **Listing 7** below

```python
# %%
def y1(t): return x(2*t+2)


fig, ax = plt.subplots(1, 1)

#plot and show
ax.plot(t, y1(t))
fig.show()
```

Listing 7: Code represents $y_1(t) = x(2t+2)$

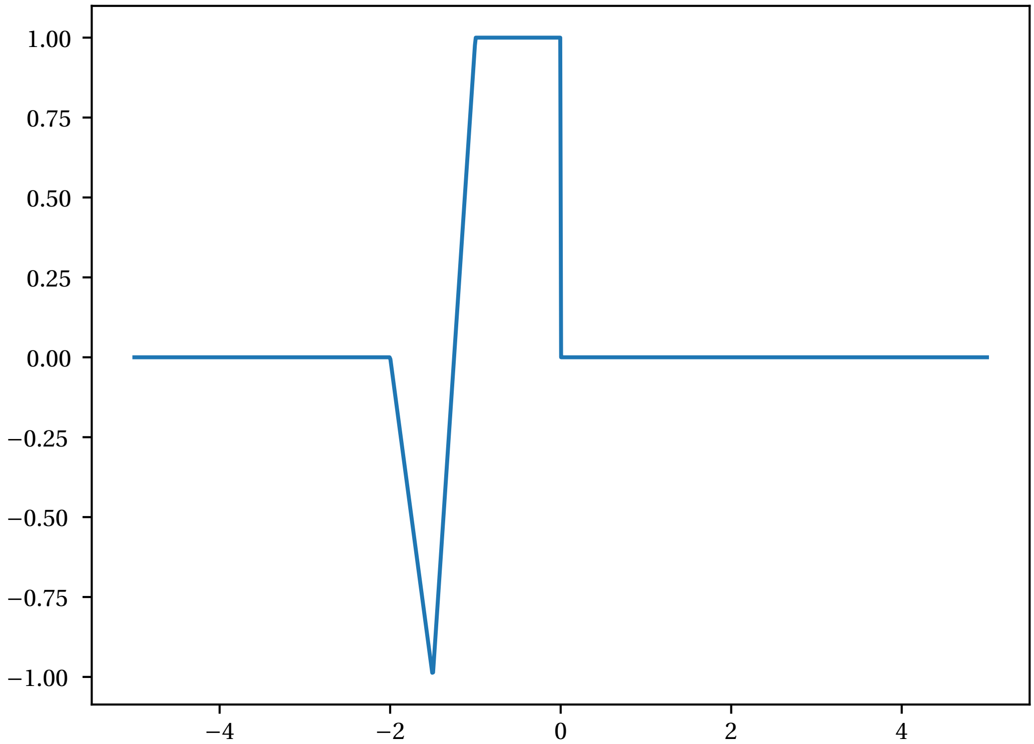

Figure 2.6: Result plot of **Listing 7**

2.4 Q1.4

In this section, we are going to do the same steps as previous section for $y_2(t)$ in **Figure 2.7**

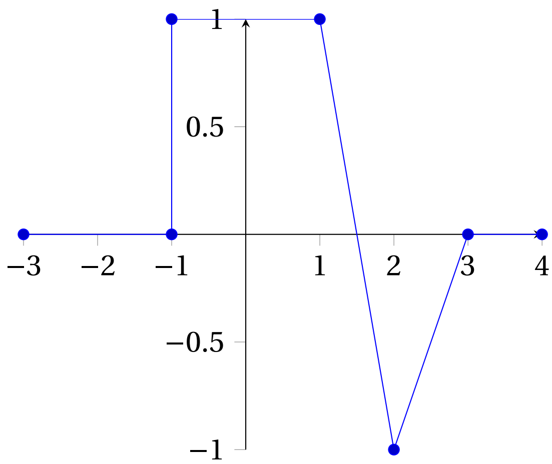

Figure 2.7: example graph of the siganl $y_2(t)$

as you see in **Listing 8**

```
# %%
def y2(t): return x(-t+1)

fig, ax = plt.subplots(1, 1)

#plot and show
ax.plot(t, y2(t))
#plt.savefig('doc/images/y2.pgf')
fig.show()
```

Listing 8: Code represents $y_2(t) = x(-t+1)$

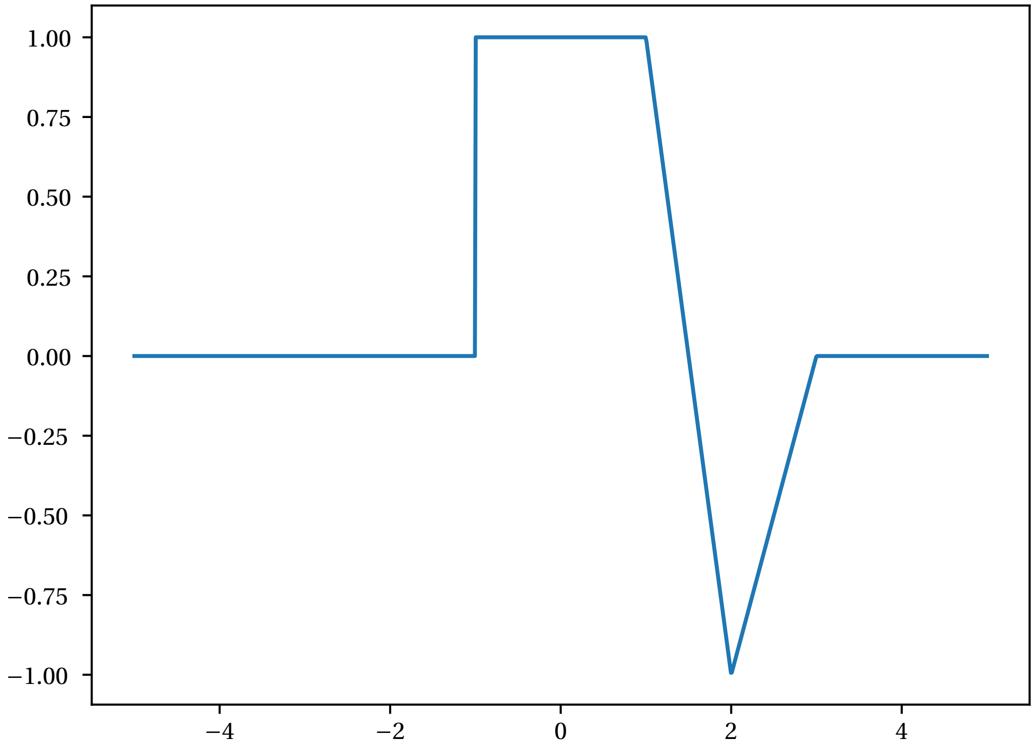

Figure 2.8: Result plot of **Listing 8**

3 QUESTION2

In this section, we are going to implement and plot these singals below.

$$x_o[n] = \frac{x[n]-x[-n]}{2} \qquad x_e[n] = \frac{x[n]+x[-n]}{2}$$

$$x_i[n] = \begin{cases} x[n] & : n < 0 \\ 0 & : o.w \end{cases} \qquad x_r[n] = \begin{cases} x[n] & : n \geq 0 \\ 0 & : o.w \end{cases}$$

3.1 Q2.1

First, we define a python function for each of these signals above as you can see in **Listing 9** and a plotting them with $F_s = 100$ps in the code **Listing 10**

```
def x_o(n): return(x(n)-x(-n))/2


def x_e(n): return (x(n)+x(-n))/2


def x_i(n): return np.array([x(np.array([i]))[0] if i < 0 else 0 for i in n])


def x_r(n): return np.array([x(np.array([i]))[0] if i >= 0 else 0 for i in n])
```

Listing 9: Functions

```
# sampling with rate : 100 sample per second
t = np.linspace(-2.5, 2.5, 500)

# Figure settings
gridsize = (2, 2)
fig = plt.figure(figsize=(10, 10))
axs = []
axs.append(plt.subplot2grid(gridsize, (0, 0)))
axs.append(plt.subplot2grid(gridsize, (0, 1)))
axs.append(plt.subplot2grid(gridsize, (1, 0)))
axs.append(plt.subplot2grid(gridsize, (1, 1)))
for i in range(4):
    axs[i].set_xticks(np.arange(-4, 5, step=1))
axs[0].set_title(f'$x_o[n]$', fontsize=16)
axs[1].set_title(f'$x_e[n]$', fontsize=16)
axs[2].set_title(f'$x_i[n]$', fontsize=16)
axs[3].set_title(f'$x_r[n]$', fontsize=16)

#plot and show
axs[0].plot(t, x_o(t), 'ro', c='blue', markersize=1)
axs[1].plot(t, x_e(t), 'ro', c='red', markersize=1)
```

```
axs[2].plot(t, x_i(t), 'ro', c='green', markersize=1)
axs[3].plot(t, x_r(t), 'ro', c='purple', markersize=1)
fig.show()
```

Listing 10: Plotting

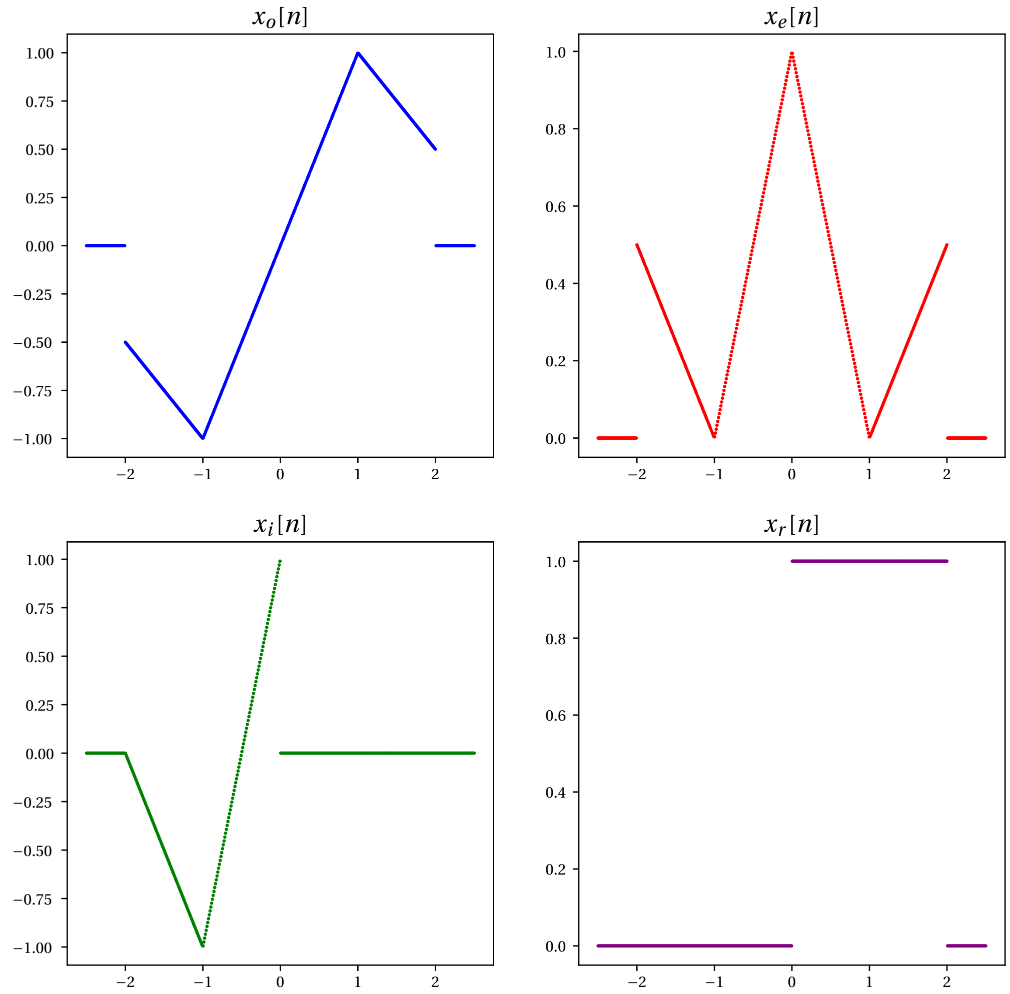

Figure 3.1: Result plot of **Listing 10**

3.2 Q2.2

We can rebuild $x[n]$ with $x_e[n]$ and $x_r[n]$. First we find the odd part in the negative side by subtracting $x_e[n]$ from $x_r[n]$ when n>=0 and time reverse and mirror respect to x axis we can get the odd part of negative side then sum up $x_e[n]$ with it and we get the $x[n]$

$$z[n] = \begin{cases} x_r[n] & : t \geq 0 \\ x_e[n] - x_r[-n] + x_e[-n] & : o.w \end{cases} \tag{3.1}$$

python function equivalent to z[n] in **Listing 11** and plotting in **Listing 12**

```
def z(t):
    # negative t
    tn = np.array([i for i in t if i < 0])
    # positive t
    tp = np.array([i for i in t if i >= 0])

    neg = 2*x_e(tn) - x_r(-tn)
    pos = x_r(tp)
    return np.concatenate((neg, pos))
```

Listing 11: Function

```
# Figure settings
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))
ax1.set_title("$z[n]$")
ax2.set_title('$x[n]$')

#plot and show
ax1.plot(t, z(t), 'ro', c='red', markersize=1)
ax2.plot(t, x(t), 'ro', c='blue', markersize=1)
fig.show()
# %% 2.3
```

Listing 12: Plotting

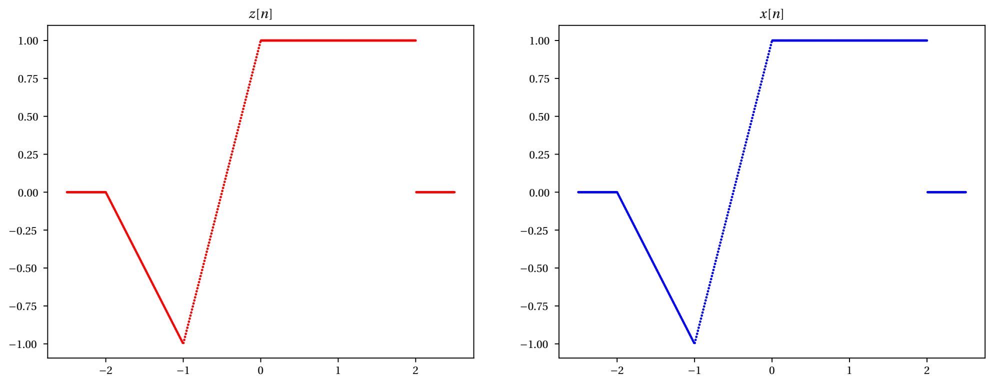

Figure 3.2: Result plot of **Listing 12**

3.3 Q2.3

we repeat the same steps as previous section for $x_o[n]$ and $x_i[n]$ and $q[n] = x[n]$ we show that with plotting them

$$q[n] = \begin{cases} x_i[n] & : t \leq 0 \\ 2x_o[n] + x_i[-n] & : o.w \end{cases} \tag{3.2}$$

```python
def q(t):
    # negative t
    tn = np.array([i for i in t if i < 0])
    # positive t
    tp = np.array([i for i in t if i >= 0])

    neg = x_i(tn)
    pos = 2*x_o(tp) + x_i(-tp)
    return np.concatenate((neg, pos))
```

Listing 13: Function

```python
# Figure settings
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))
ax1.set_title("$q[n]$")
ax2.set_title('$x[n]$')

#plot and show
ax1.plot(t, q(t), 'ro', c='green', markersize=1)
ax2.plot(t, x(t), 'ro', c='blue', markersize=1)
plt.savefig('doc/images/q2.3.pgf')
fig.show()
```

Listing 14: plotting

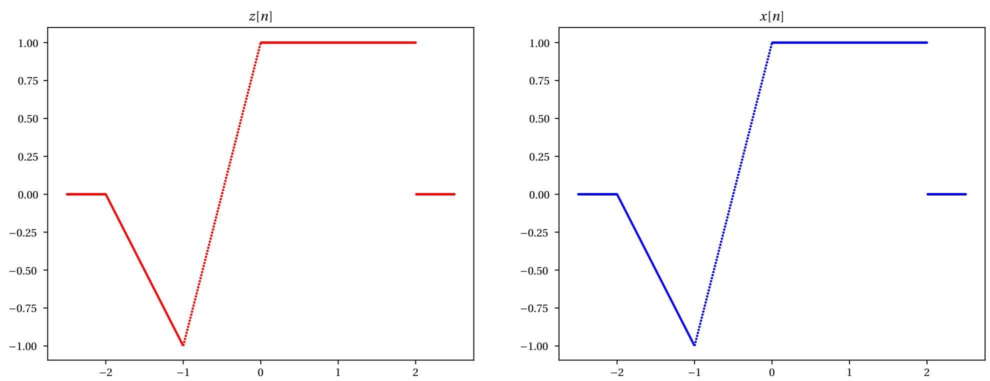

Figure 3.3: Result plot of **Listing 14**

4 QUESTION3

In this section we work with system below:

$$y(t) = \int_{-\infty}^{+\infty} e^{u-t} x(u-2)du \tag{4.1}$$

4.1 Q3.1

First we check the superposition property, $S\{a_1x_1(t) + a_2x_2(t)\}$ and $S\{a_1x_1(t)\} + S\{a_2x_2(t)\}$ have a same result because you can sperate a summation in integral and it's obvoius

$$x_1(t) = u(t) - u(t-2) \tag{4.2}$$
$$x_2(t) = u(t) - u(t-3) \tag{4.3}$$

First we define our System with this function below in **Listing 16** we use **scipy package** for integral

```
import scipy.integrate as integrate
```

Listing 15: scipy package

```
def S(x):
    def y(t):
        return np.array([integrate.quad(
            lambda u: x(u-2)*np.exp(u-i), -20, 20)[0] for i in t])
    return y
```

Listing 16: System fucntion

Then we define python functions for $x1(t)$, $x2(t)$, $s1(t) = S\{a_1x_1(t) + a_2x_2(t)\}$ and $s2(t) = S\{a_1x_1(t)\} + S\{a_2x_2(t)\}$ as you see in **Listing 17**

```
def x1(t): return unit(t) - unit(t-2)


def x2(t): return unit(t) - unit(t-3)


a1 = 3
a2 = 2


def s1(t): return S(lambda n: a1*x1(n) + a2*x2(n))(t)


def s2(t): return S(lambda n: a1*x1(n))(t) + S(lambda n: a2*x2(n))(t)
```

Listing 17: System fucntion

```
# sampling with rate 100 per second
t = np.linspace(-5, 5, 1000)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

ax1.plot(t, s1(t))
ax2.plot(t, s2(t))
```

Listing 18: plotting

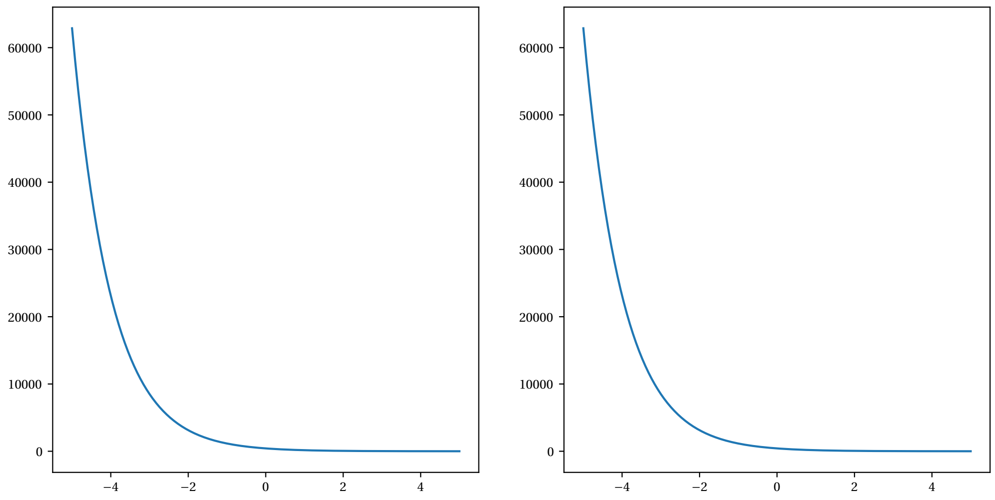

Figure 4.1: Result plot of **Listing 18**

4.2 Q3.2

In this section we define $y_1(t) = S\{x_1(t)\}$ and $y_2(t) = S\{x_1(t-3)\}$ in our code like **Listing 19** and plot them in **Listing 20**. In the **Figure 4.2** you can see both $y_1(t-3)$ and $y_2(t)$ are equal.

```
def y1(t): return S(x1)(t)


def y2(t): return S(lambda n: x1(n-3))(t)
```

Listing 19: Functions

```
# sampling with rate 100 per second
t = np.linspace(-5, 5, 1000)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

ax1.plot(t, y1(t-3))
ax2.plot(t, y2(t))
fig.show()
```

```
# %%
```

Listing 20: Plotting code

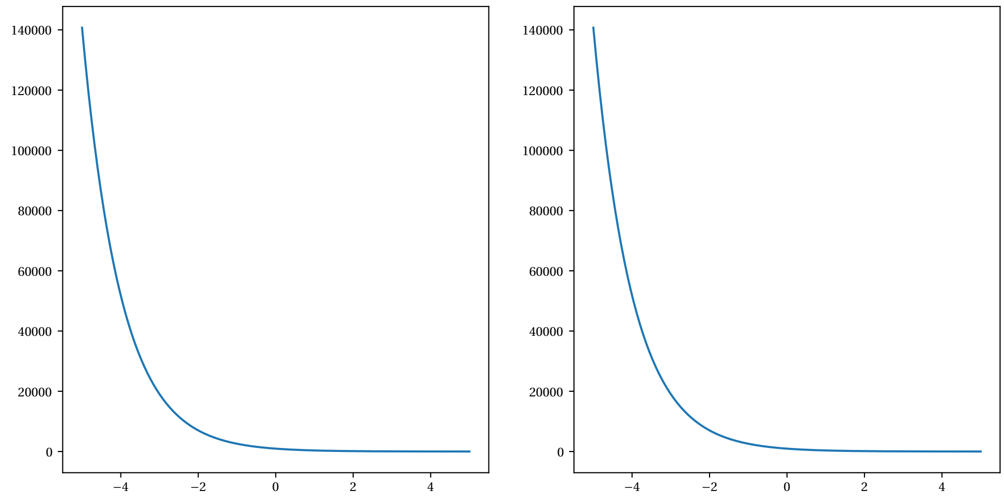

Figure 4.2: Result plot of **Listing 20**

4.3 Q3.3

in two previous sections, we examine two cases one for **linearity** and one for **time-invariant** property of the system. but we can prove them more generally that system is LTI.

5 QUESTION4

5.1 Q4.1

For implementing function for energy calculation of singal we use

$$\lim_{n\to\infty} \sum_{i=1}^{n} |x(t_i)|^2 \Delta t \tag{5.1}$$

this and a corresponding fucntion for this in python in **Listing 21**

```
def ct_energy(x, n, d): return sum([np.abs(x(i))**2 for i in n])*d
```

Listing 21: Energy function

5.2 Q4.2

We use the previous function and calcualte energy for $x(t)$, $y_1(t)$ and $y_2(t)$ as you see in **Listing 22** and output in **Listing 23**

```
d = .0001
n = np.arange(-4, 4, d)
x_en = ct_energy(x, n, d)
y1_en = ct_energy(y1, n, d)
y2_en = ct_energy(y2, n, d)
# endsection
print(f'x_en: {x_en}\ny_en: {y1_en}\ny2_en:{y2_en}')
```

Listing 22: Energy calculation

```
x_en: 2.6666166750070275
y_en: 1.3332833500056211
y2_en:2.6667166749880353
```

Listing 23: output

6 QUESTION5

Convolution of signals $x_1[n]$ and $x_2[n]$:

$$y[n] = \sum_{k=-\infty}^{+\infty} x_1[k]x_2[n-k] \tag{6.1}$$

6.1 Q5.1

We define a python function for convolution as below **Listing 24**

```
def Conv(x1, x2):
    def cfunc(n):
        return np.array([sum([x1(k) * x2(i-k) for k in n])for i in n])
    return cfunc
```

Listing 24: Convolution function

6.2 Q5.2

For each signals in the below we define a function and then we plot each of them and their convolution

$$x_1[n] = (\frac{1}{2^{-n+1}}).(u[n+2]-u[n-2])$$

$$x_2[n] = \begin{cases} \sum_{k=-\infty}^{n}(sin(2k)+e^{j\pi k}).(u[k+3]-u[k-5]) & : 0 < n < 7 \\ 0 & : O.W \end{cases}$$

```
def x1(n): return (2.0)**(n-1) * unit(n)


def s(i): return sum([(np.sin(2*k) + np.exp(2j*np.pi*k/2))*(unit(k+3)-unit(k-5))
                       for k in range(-3, i+1)])


def x2(n): return np.array([s(i) if i < 7 and i > 0 else 0 for i in n])\
    if isinstance(n, Iterable) else (s(n) if n < 7 and n > 0 else 0)
```

Listing 25: functions

```
x3 = Conv(x1, x2)

n = np.arange(-3, 20, 1)
gridsize = (3, 1)
fig = plt.figure(figsize=(16, 11))
ax1 = plt.subplot2grid(gridsize, (0, 0))
ax2 = plt.subplot2grid(gridsize, (1, 0))
```

```
ax3 = plt.subplot2grid(gridsize, (2, 0))

ax1.stem(n, x1(n), 'b', markerfmt='bo', use_line_collection=True, )
ax1.set_title("$x_1[n]$")
ax2.stem(n, x2(n).real, 'g', markerfmt='go', use_line_collection=True)
ax2.set_title("$|x_2[n]|$")
ax3.stem(n, x3(n).real,'r', markerfmt='ro', use_line_collection=True)
ax3.set_title('|x_1[n]*x_2[n]|')

fig.show()
```

Listing 26: Convolt and plot

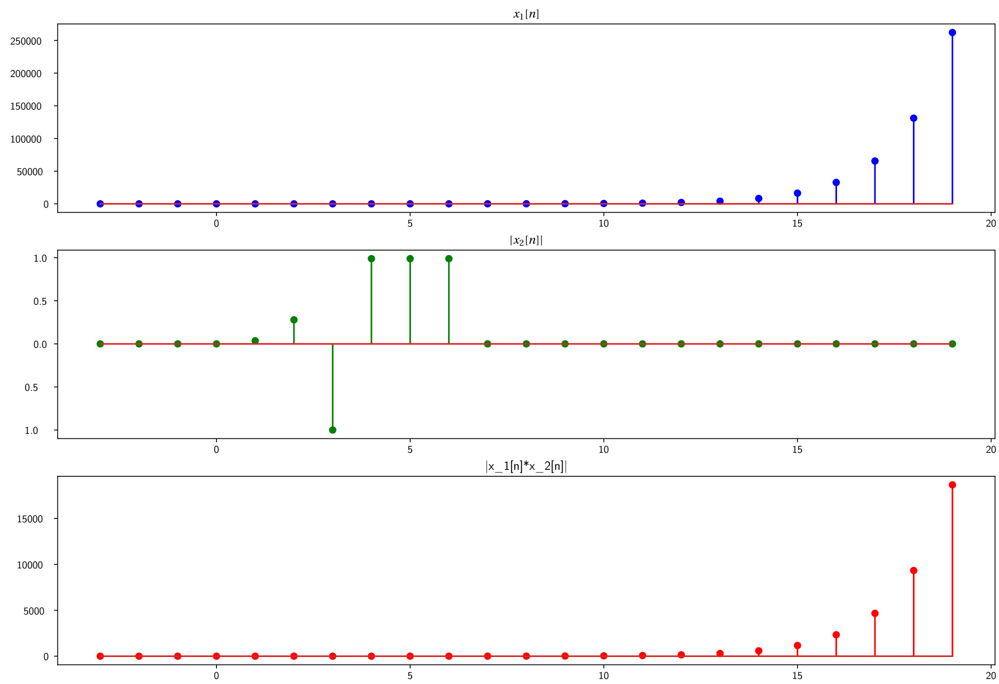

Figure 6.1: Result plot of **Listing 26**