CENG 384 - Signals and Systems for Computer Engineers
Spring 2023
Homework 1

Göçer, Anıl Eren
e2448397@ceng.metu.edu.tr

Altuğ, Kadir
e2448108@ceng.metu.edu.tr

March 26, 2023

---

1. (a) $z = x + yj \longrightarrow \bar{z} = x - yj$

   $2(x + yj) + 5 = j - (x - yj)$

   $2x + 2yj + 5 = j - x + yj$

   $3x + yj = -5 + j$

   $x = \frac{-5}{3}$, $y = 1 \longrightarrow z = \frac{-5}{3} + j$

   $|z|^2 = z\bar{z} = (\frac{-5}{3} + j)(\frac{-5}{3} - j)$

   $= \frac{25}{9} + \frac{5j}{3} - \frac{5j}{3} - j^2 = \frac{25}{9} + 1 = \frac{34}{9}$

   So, $|z|^2 = \frac{34}{9}$
   Also z can be plotted on the complex plane as below:

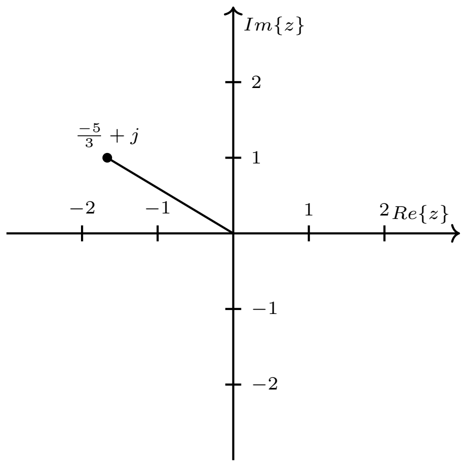

Figure 1: $z = \frac{-5}{3} + j$ on the complex number

   (b) $z = re^{j\theta} \longrightarrow z^5 = r^5 e^{5j\theta}$

   $z^5 = r^5 e^{5j\theta} = 32j$

   $\Longrightarrow r^5 = 32 \Longrightarrow \mathbf{r = 2}$

   $\Longrightarrow e^{5j\theta} = j = e^{j\frac{\pi}{2}} \Longrightarrow 5\theta = \frac{\pi}{2} + 2\pi k \Longrightarrow \theta = \mathbf{\frac{\pi}{10}} + \mathbf{\frac{\pi}{5}k}$

   Let's take k=0 where k∈Z;

   $\theta = \frac{\pi}{10}$

Thus, $< z = \frac{\pi}{10}$ rad, $|z| = 2$ and its polar form is found as follows:

$\mathbf{z = 2e^{j\frac{\pi}{10}}}$

(c) Since the expressions includes multiplication and division of complex numbers, it is a wise choice to write subexpression in polar form:

$z_1 = (1+j) = \sqrt{2}e^{j\frac{\pi}{4}}$

$z_2 = (\frac{1}{2} + \frac{\sqrt{3}}{2}j) = e^{j\frac{\pi}{3}}$

$z_3 = (j-1) = \sqrt{2}e^{j\frac{3\pi}{4}}$

$z = \frac{z_1 z_2}{z_3} = \frac{\sqrt{2}e^{j\frac{\pi}{4}}e^{j\frac{\pi}{3}}}{\sqrt{2}e^{j\frac{3\pi}{4}}} = e^{-j\frac{\pi}{6}}$

So its magnitude $|z| = 1$ and its angle $< z = \frac{-\pi}{6}$ rad$=-30°$

(d) It is known that $j = e^{j\frac{\pi}{2}}$

$z = je^{-j\frac{\pi}{2}} = e^{j\frac{\pi}{2}}e^{-j\frac{\pi}{2}} = e^0 = \mathbf{1}$

We know , $1 = e^{j2\pi}$

So , z=$e^{j2\pi}$, which is in polar form.

2. $x(t)$ can be defined in the form of a piecewise function as follows:

$$x(t) = \begin{cases} 0, & \text{if } -3 \leq t \leq -2 \\ t+2, & \text{if } -2 < t \leq -1 \\ 1, & \text{if } -1 < t \leq 1 \\ -t+2, & \text{if } 1 < t \leq 2 \\ 0, & \text{if } 2 < t \leq 3 \end{cases}$$

Then, replace t with $\frac{1}{2}t+1$ in the function above:

$$x(\frac{1}{2}t+1) = \begin{cases} 0, & \text{if } -3 \leq \frac{1}{2}t+1 \leq -2 \\ \frac{1}{2}t+3, & \text{if } -2 < \frac{1}{2}t+1 \leq -1 \\ 1, & \text{if } -1 < \frac{1}{2}t+1 \leq 1 \\ -\frac{1}{2}t+1, & \text{if } 1 < \frac{1}{2}t+1 \leq 2 \\ 0, & \text{if } 2 < \frac{1}{2}t+1 \leq 3 \end{cases}$$

This is equivalent to :

$$x(\frac{1}{2}t+1) = \begin{cases} 0, & \text{if } -8 \le t \le -6 \\ \frac{1}{2}t+3, & \text{if } -6 < t \le -4 \\ 1, & \text{if } -4 < t \le 0 \\ -\frac{1}{2}t+1, & \text{if } 0 < t \le 2 \\ 0, & \text{if } 2 < t \le 4 \end{cases}$$

In a more intuitive way, transformation applied to x(t) can be described as :

(i) Time scale: expanded by 2

(ii) Time shift: Shift by 2 to the left

Thus the signal $y(t) = x(\frac{1}{2}t+1)$ can be drawn as:

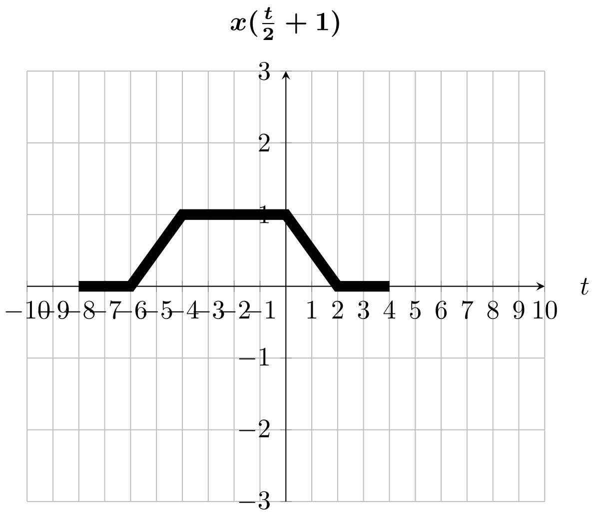

Figure 2: t vs. $y(t) = x(\frac{t}{2}+1)$.

3. (a) $x[-n]$:It is the reflected version of x[n] about y-axis.

x[2n-1]:To get this, we first shrink x[n] by 2, then shift to right by 1/2 and only take the integer n values , since it is discrete time signal.

At the end, we sum up these two signal, which are x[-n] and x[2n-1].

To be more precise:

n	x[n]
-1	0
0	0
1	-1
2	2
3	0
4	-4
5	0
6	0
7	3
8	0

n	x[−n]
-8	0
-7	3
-6	0
-5	0
-4	-4
-3	0
-2	2
-1	-1
0	0
1	0

n	x[2n − 1]
0	0
1	-1
2	0
3	0
4	3

Therefore;

$\underline{n}$	$\underline{x[-n]+x[2n-1]}$
-8	0
-7	3
-6	0
-5	0
-4	-4
-3	0
-2	2
-1	-1
0	0
1	-1
2	0
3	0
4	3

So x[-n]+x[2n-1] can be drawn as:

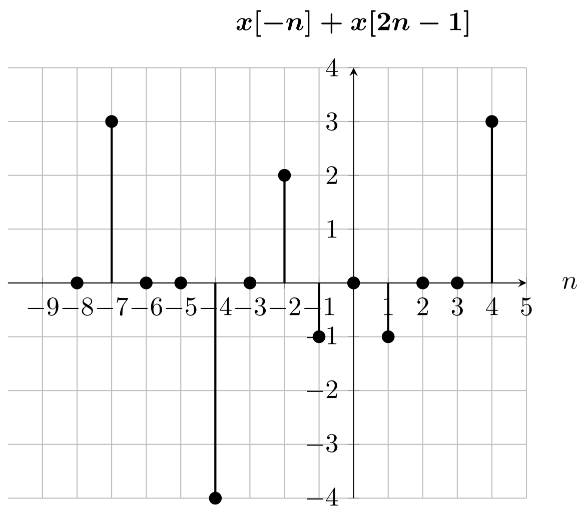

Figure 3: n vs. $x[-n]+x[2n-1]$.

(b) $x[-n]+x[2n-1]$=$3\delta[n+7]-4\delta[n+4]+2\delta[n+2]-\delta[n+1]-\delta[n-1]+3\delta[n-4]$

4. (a) *YES*. It is periodic.

$x(t)=5sin(3t-\frac{\pi}{4})$

$w_0=3\longrightarrow T_0=\frac{2\pi}{3}$

Therefore, it is periodic with fundamental period $T_0=\frac{2\pi}{3}$

(b) *YES*. It is periodic.
Let's say $x[n]=x_1[n]+x_2[n]$

$x_1[n]=cos(\frac{13\pi}{10}n)$ and $x_2=sin(\frac{7\pi}{10}n)$.

For x_1;

w_0=$\frac{13\pi}{10}\longrightarrow N_0=\frac{2\pi}{w_0}m=\frac{2\pi}{13\frac{\pi}{10}}m=\frac{20}{13}m.$

Put 13 into m $\longrightarrow N_0=\frac{20}{13}13=20$

For x_2;

w_0=$\frac{7\pi}{10}\longrightarrow N_0=\frac{2\pi}{w_0}m=\frac{2\pi}{7\frac{\pi}{10}}m=\frac{20}{7}m.$

Put 7 into m $\longrightarrow N_0=\frac{20}{7}7=20$

We need to find lowest common multiple of fundamental periods of x_1 and x_2 to find fundamental period of x[n]:

LCM(20,20)=20

So, this signal, x[n], is periodic with fundamental period $N_0 = 20$

(c) *NO*. It is not periodic.

$w_0 = 7 \longrightarrow N_0 = \frac{2\pi}{w_0}m = \frac{2\pi}{7}m$

There is no integer value of m which makes N_0 an integer.

We know that fundamental period of a discrete time signal must be an integer. However, we've seen that there is no integer value of m making N_0 integer.

Thus, this signal is not periodic.

5. (a) x(t)=u(t-1)-3u(t-3)+u(t-4)

(b) We know that $\frac{du(t-t_0)}{dt} = \delta(t - t_0)$

By using this fact, $\frac{dx(t)}{dt}$ can be found as follows:

$\frac{dx(t)}{dt} = \delta(t - 1) - 3\delta(t - 3) + \delta(t - 4)$.

It is drawn as below:

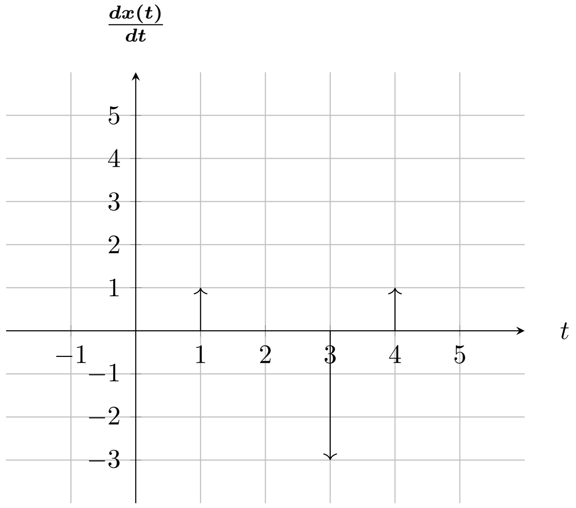

Figure 4: t vs. $\frac{dx(t)}{dt}$.

6. (a) **Memory**:*YES*. It has memory. It depends on future or past value of the input e.g.

y(2)=2x(7)

y(-5)=-5x(-7)

Stability:*NO*. It is not stable. Even if input, x(t), would be bounded ,y(t) may become unbounded because of the term,t, which is unbounded. So, y(t) is unstable.

Causality:*NO*. It is not casual. The outputs depends on future values of input. For example;

y(2)=2x(7)

Linearity:YES. It is linear.

Suppose we feed two inputs, x_1 and x_2 to the system. The corresponding outputs will be:

$$x_1(t) \longrightarrow y_1(t) = tx_1(2t+3)$$
$$x_2(t) \longrightarrow y_2(t) = tx_2(2t+3)$$

Output for the superposition of the inputs, $x_3 = a_1x_1 + a_2x_2$ is:

$$y_3(t) = t(a_1x_1 + a_2x_2) = a_1y_1 + a_2y_2$$

Superposition holds. Thus it is linear.

Invertibility: NO. It is not invertible.

$x(t) = \dfrac{y(\frac{t-3}{2})}{\frac{t-3}{2}}$, not defined when t=3.

$\mathbf{time-invariance}$: NO. It is time-varying(Not time invariant).

First,shift the input by t_0 and observe the corresponding output.

Corresponding output: $tx(2(t-t_0)+3) = tx(2t-2t_0+3)$.

Then, shift output by t_0:

$(t-t_0)x(2(t-t_0)+3) = (t-t_0)x(2t-2t_0+3)$

Lastly, observe that

$tx(2t-2t_0+3) \neq (t-t_0)x(2t-2t_0+3)$

Thus, it is time-varying.

(b) **Memory**:YES. It has memory. The output depends on past input values. For example;

y[0]=x[-1]+x[-2]+x[-3]+. . . . . . .

Stability:NO. It is not stable. Even if the input is bounded, the output is infinite sum of the input, i.e. y[n] is the sum of infinitely many number of bounded values. Therefore, the system is not stable.

Causality:YES. It is causal. The output does not depend on future values of the input. It only depends on past values of input. For example:

y[n]=x[n-1]+x[n-2]+. . . . . .

So, it is causal.

Linearity:YES. It is linear. Suppose , we feed two inputs, x_1 and x_2, to the system.The corresponding outputs will be:

$x_1[n] \longrightarrow y_1[n] = \sum_{k=1}^{\infty} ax_1[n-k]$

$x_2[n] \longrightarrow y_2[n] = \sum_{k=1}^{\infty} ax_2[n-k]$

Output for the superposition of the inputs,

$x_3 = ax_1 + ax_2$
$x_3[n] \longrightarrow y_3[n] = \sum_{k=1}^{\infty} a_1 x_1[n-k] + a_2 x_2[n-k]$

$= a_1 \sum_{k=1}^{\infty} x_1[n-k] + a_2 \sum_{k=1}^{\infty} x_1[n-k]$

$= a_1 y_1[n] + a_2 y_2[n]$

Superposition holds. thus it is linear

Invertibility:YES. It is invertible.

$y[n+1] = x[n] + x[n-1] + x[n-2] + ...$

$y[n] = x[n-1] + x[n-2] + x[n-3] + ...$

If we substract y[n+1] and y[n]:

$y[n+1] - y[n] = x[n]$

We found $x[n] = y[n+1] - y[n]$. Thus the system is invertible.

time − invariance:YES. It is time-invariant.

First, shift the input by n_0 and observe the corresponding output:

$\sum_{k=1}^{\infty} x[n - n_0 - k]$

Then, shift the output by n_0 :

$\sum_{k=1}^{\infty} x[n - n_0 - k]$

As you can see, both of them are equal. A time shift in input creates identical shift on the output.

Thus it is time-invariant.

7. (a)
```python
# Question 7 - Part a
import matplotlib.pyplot as plt

MARGIN = 0

def load_data_part_a(filepath):
    file = open(filepath)
    data =list(map(lambda x: float(x), file.readline().split(",")))
    starting_index = data[0]
    signal = data[1:]
    time = list(range(int(starting_index), int(starting_index) + len(signal)))
    temp = fill_beyond_with_0(time, signal)
    time = temp[0]
    signal = temp[1]
    return (time, signal)

def fill_beyond_with_0(time, signal):
    zeros = [0] * MARGIN
    new_signal = zeros + signal + zeros
    lower_bound = time[0]
    upper_bound = time[-1]
    new_lower_bound = lower_bound - MARGIN
    new_upper_bound = upper_bound + MARGIN
    new_time = list(range(new_lower_bound, lower_bound)) + time + list(range(upper_bound+1,
    new_upper_bound+1))
    return (new_time, new_signal)

def reflect_about_y(time, signal):
    reflected_signal = signal[::-1]
    reflected_time = list(map(lambda x: -x, time[::-1]))
    return (reflected_time, reflected_signal)

def magnitude_scale(time, signal, k):
```

```python
    scaled_signal = []
    for element in signal:
        scaled_signal.append(k * element)
    return (time, scaled_signal)

################# PART A #################

# x(t) = [0.5 * (x(t) + x(-t))] + [0.5 * (x(t) - x(-t))]
# even part: [0.5 * (x(t) + x(-t))]
# odd part: [0.5 * (x(t) - x(-t))]

def even_odd_decomposition(filepath, option): # option = 0 -> even part , option = 1 -> odd part
    x_n = load_data_part_a(filepath) # x[n]
    x_minus_n = reflect_about_y(x_n[0], x_n[1]) # x[-n]

    x_n_time = x_n[0]
    x_minus_n_time = x_minus_n[0]

    x_n_signal = x_n[1]
    x_minus_n_signal = x_minus_n[1]

    x_n_lower_time_bound = x_n_time[0]
    x_n_upper_time_bound = x_n_time[-1]

    x_minus_n_lower_time_bound = x_minus_n_time[0]
    x_minus_n_upper_time_bound = x_minus_n_time[-1]

    new_time_lower_bound = min(x_n_lower_time_bound, x_minus_n_lower_time_bound)
    new_time_upper_bound = max(x_n_upper_time_bound, x_minus_n_upper_time_bound)

    i = new_time_lower_bound
    tmp1 = []
    c1 = 0
    while (i < x_n_lower_time_bound):
        c1+=1
        tmp1.append(i)
        i+=1

    i = x_n_upper_time_bound
    tmp2 = []
    c2 = 0
    while (i < new_time_upper_bound):
        c2+=1
        tmp2.append(i)
        i+=1

    x_n_time = tmp1 + x_n_time + tmp2
    x_n_signal = c1 * [0] + x_n_signal + c2 * [0]


    i = new_time_lower_bound
    tmp1 = []
    c1 = 0
    while (i < x_minus_n_lower_time_bound):
        c1+=1
        tmp1.append(i)
        i+=1

    i = x_minus_n_upper_time_bound
    tmp2 = []
    c2 = 0
    while (i < new_time_upper_bound):
        c2+=1
        tmp2.append(i)
        i+=1

    x_minus_n_time = tmp1 + x_minus_n_time + tmp2
    x_minus_n_signal = c1 * [0] + x_minus_n_signal + c2 * [0]


    if (option == 0):
        even_decomposition_time = x_n_time
        even_decomposition_signal = []

        for i in range(len(even_decomposition_time)):
            even_decomposition_signal.append(x_n_signal[i] + x_minus_n_signal[i])

        return magnitude_scale(even_decomposition_time, even_decomposition_signal, 0.5)

    elif (option == 1):
        odd_decomposition_time = x_n_time
        odd_decomposition_signal = []
```

```python

        for i in range(len(odd_decomposition_time)):
            odd_decomposition_signal.append(x_n_signal[i] - x_minus_n_signal[i])

        return magnitude_scale(odd_decomposition_time, odd_decomposition_signal, 0.5)


###########################################


################# PART A - Outputs #################
r = even_odd_decomposition("./chirp_part_a.csv", 0)
plt.stem(r[0], r[1])
plt.xlabel('n', fontsize=20)
plt.ylabel('Even(x[n])', fontsize=20)
plt.title('Even Part of chirp_part_a.csv', fontsize=20)
plt.savefig("Even Part of chirp_part_a.pdf", format="pdf", bbox_inches="tight")
plt.cla()
r = even_odd_decomposition("./chirp_part_a.csv", 1)
plt.stem(r[0], r[1])
plt.xlabel('n', fontsize=20)
plt.ylabel('Odd(x[n])', fontsize=20)
plt.title('Odd Part of chirp_part_a.csv', fontsize=20)
plt.savefig("Odd Part of chirp_part_a.pdf", format="pdf", bbox_inches="tight")
plt.cla()
r = even_odd_decomposition("./shifted_sawtooth_part_a.csv", 0)
plt.stem(r[0], r[1])
plt.xlabel('n', fontsize=20)
plt.ylabel('Even(x[n])', fontsize=20)
plt.title('Even Part of shifted_sawtooth_part_a.csv', fontsize=20)
plt.savefig("Even Part of shifted_sawtooth_part_a.pdf", format="pdf", bbox_inches="tight")
plt.cla()
r = even_odd_decomposition("./shifted_sawtooth_part_a.csv", 1)
plt.stem(r[0], r[1])
plt.xlabel('n', fontsize=20)
plt.ylabel('Odd(x[n])', fontsize=20)
plt.title('Odd Part of shifted_sawtooth_part_a.csv', fontsize=20)
plt.savefig("Odd Part of shifted_sawtooth_part_a.pdf", format="pdf", bbox_inches="tight")
plt.cla()

r = even_odd_decomposition("./sine_part_a.csv", 0)
plt.stem(r[0], r[1])
plt.xlabel('n', fontsize=20)
plt.ylabel('Even(x[n])', fontsize=20)
plt.title('Even Part of sine_part_a.csv', fontsize=20)
plt.savefig("Even Part of sine_part_a.pdf", format="pdf", bbox_inches="tight")
plt.cla()
r = even_odd_decomposition("./sine_part_a.csv", 1)
plt.stem(r[0], r[1])
plt.xlabel('n', fontsize=20)
plt.ylabel('Odd(x[n])', fontsize=20)
plt.title('Odd Part of sine_part_a.csv', fontsize=20)
plt.savefig("Odd Part of sine_part_a.pdf", format="pdf", bbox_inches="tight")
plt.cla()
####################################################
```

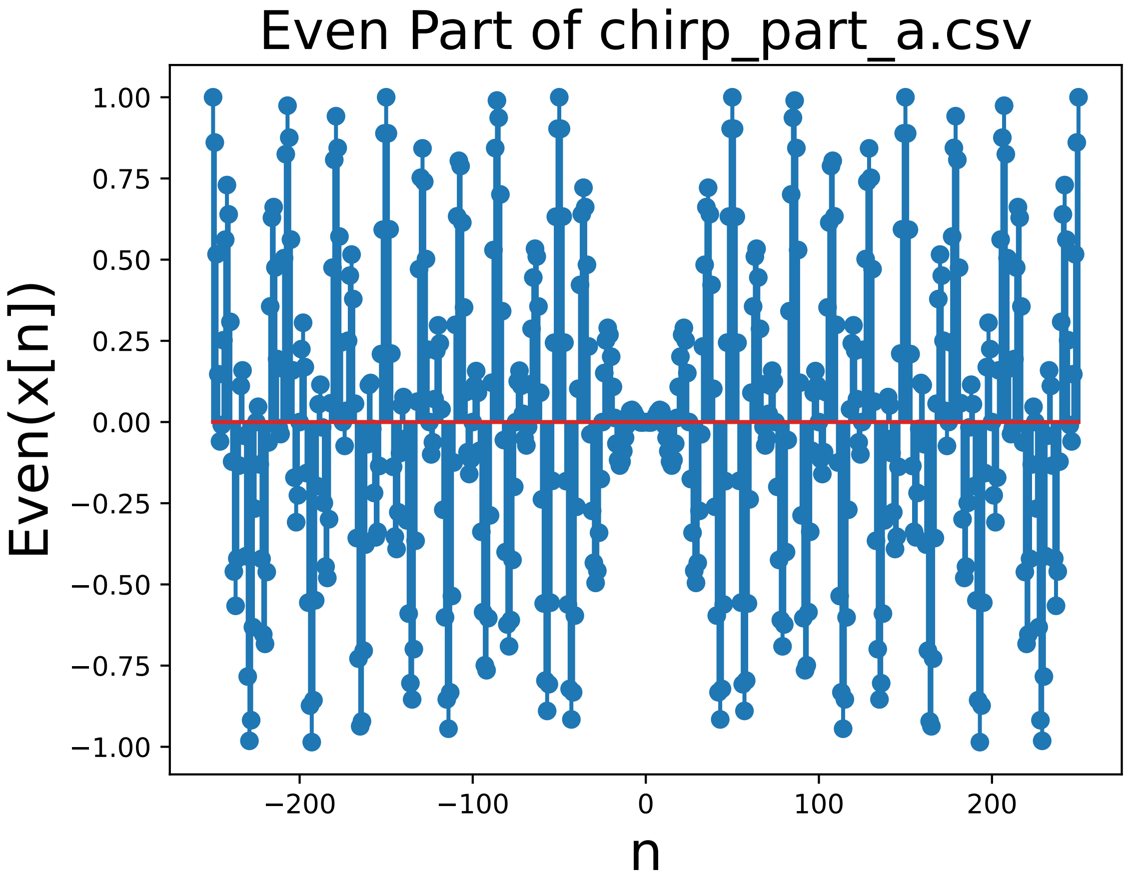

Figure 5: Even Part of chirp_part_a.csv

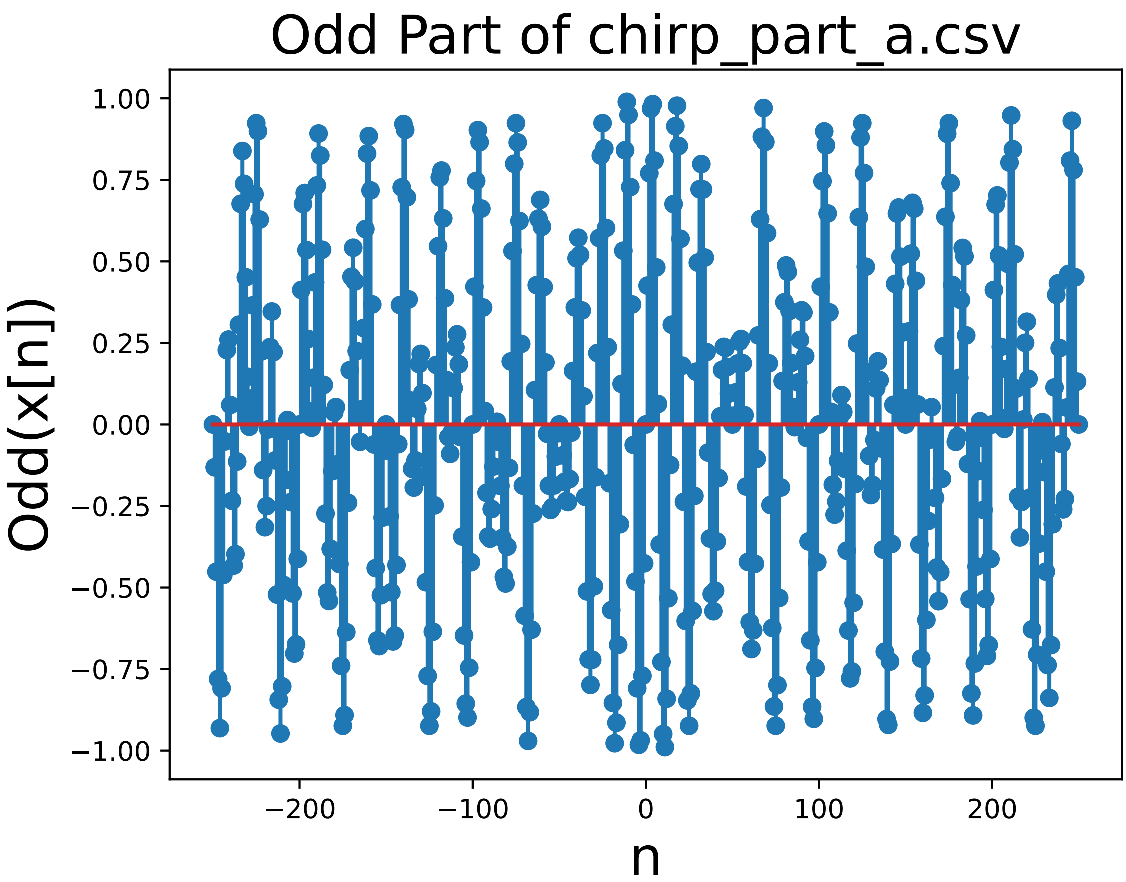

Figure 6: Odd Part of chirp_part_a.csv

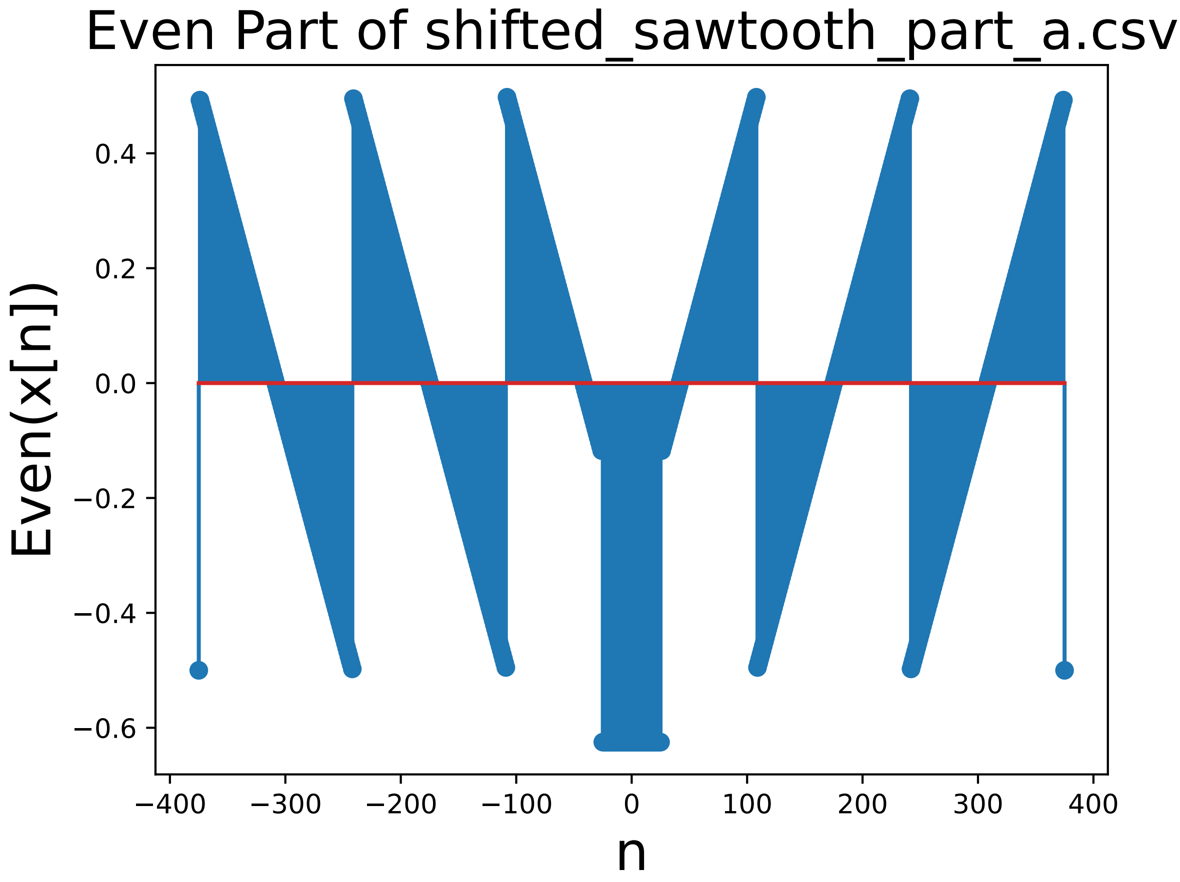

Figure 7: Even Part of shifted_sawtooth_part_a.csv

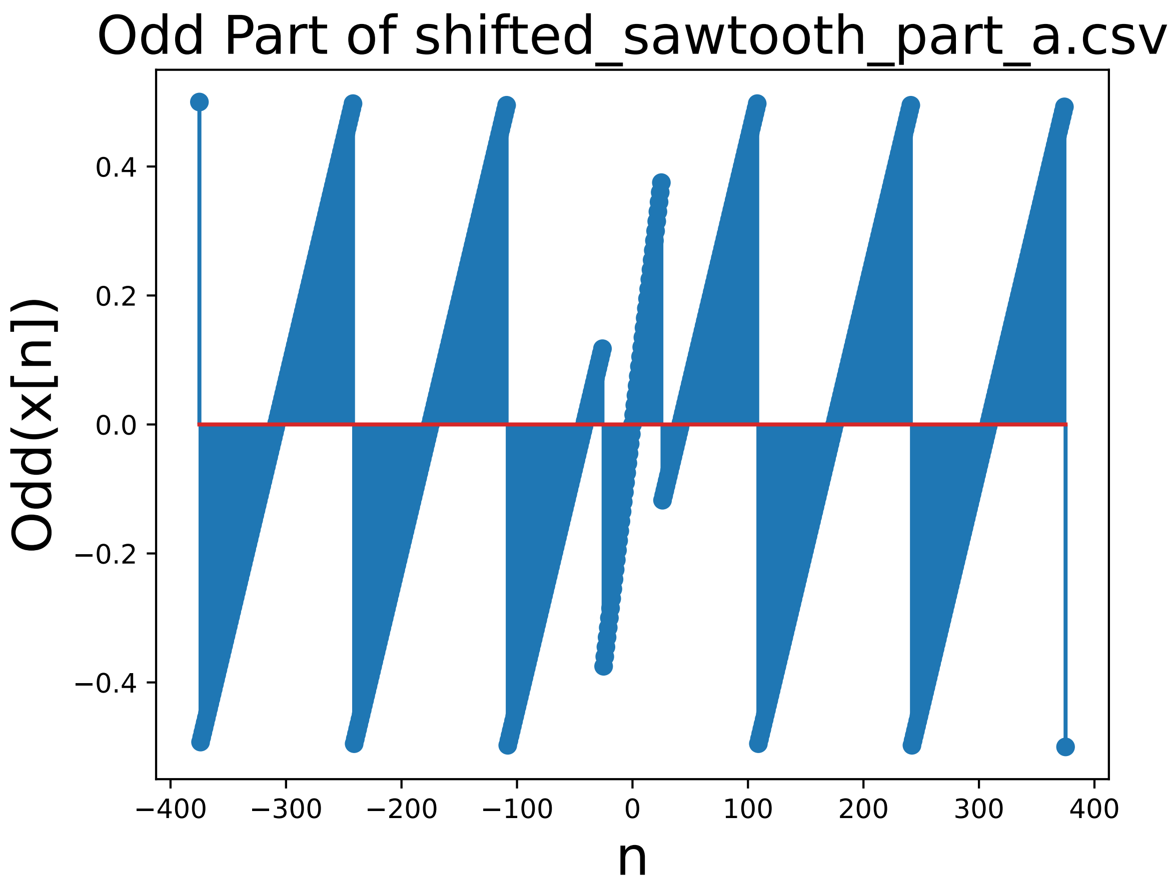

Figure 8: Odd Part of shifted_sawtooth_part_a.csv

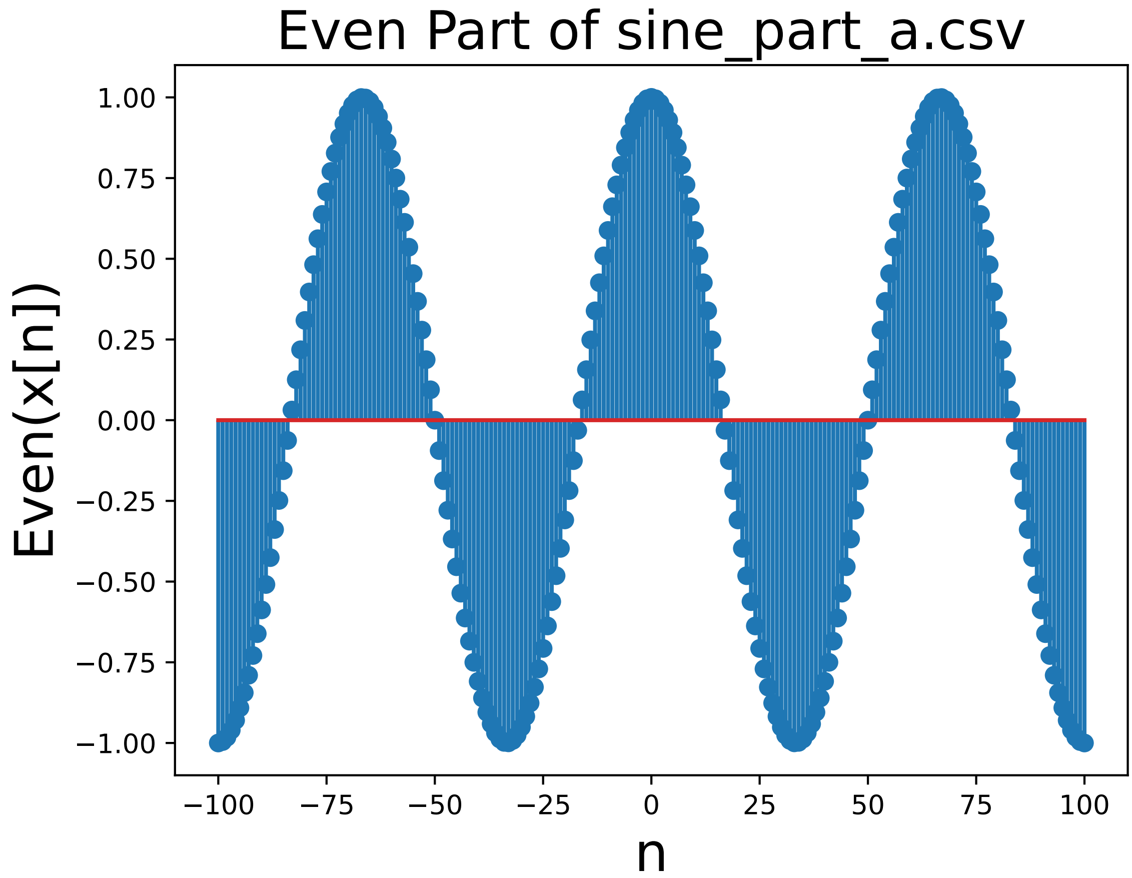

Figure 9: Even Part of sine_part_a.csv

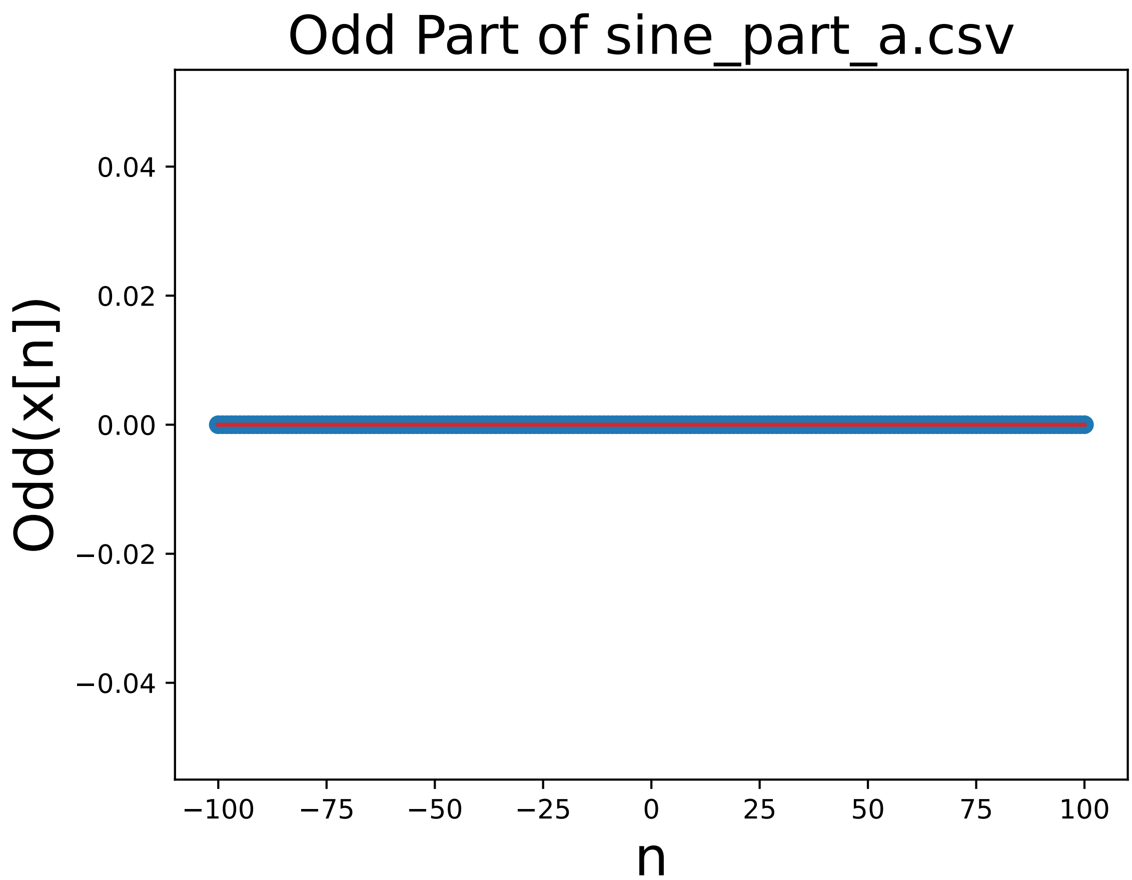

Figure 10: Odd Part of sine_part_a.csv

(b)
```python
# Question 7 - Part b
import matplotlib.pyplot as plt

MARGIN = 0

def load_data_part_b(filepath):
    file = open(filepath)
    data =list(map(lambda x: float(x), file.readline().split(",")))
    starting_index = data[0]
    a = data[1]
    b = data[2]
    signal = data[3:]
    time = list(range(int(starting_index), int(starting_index) + len(signal)))
    temp = fill_beyond_with_0(time, signal)
    time = temp[0]
    signal = temp[1]
    return (time, signal, a, b)

def fill_beyond_with_0(time, signal):
    zeros = [0] * MARGIN
    new_signal = zeros + signal + zeros
    lower_bound = time[0]
    upper_bound = time[-1]
    new_lower_bound = lower_bound - MARGIN
    new_upper_bound = upper_bound + MARGIN
    new_time = list(range(new_lower_bound, lower_bound)) + time + list(range(upper_bound+1,
    new_upper_bound+1))
    return (new_time, new_signal)

def reflect_about_y(time, signal):
    reflected_signal = signal[::-1]
    reflected_time = list(map(lambda x: -x, time[::-1]))
    return (reflected_time, reflected_signal)

def time_shift(time, signal, amount, direction): # direction : 0 -> left shift , direction : 1
    -> right shift
    shifted_time = []
    if (direction == 0):
        for moment in time:
            shifted_moment = moment - amount
            shifted_time.append(shifted_moment)
    else:
        for moment in time:
            shifted_moment = moment + amount
            shifted_time.append(shifted_moment)

    return (shifted_time, signal)

def time_scale(time, signal, a):
    if (a == 1):
        return (time, signal)
    elif (a == -1):
        return reflect_about_y(time, signal)
    elif (a > 0):
        scaled_time = []
        scaled_signal = []
        for i in range(0,len(time)):
            if (int(time[i] / a) == float(time[i] / a)):
                scaled_time.append(int(time[i] / a))
                scaled_signal.append(signal[i])
        return (scaled_time, scaled_signal)
    elif (a < 0):
        reflected_version = reflect_about_y(time, signal)
        time = reflected_version[0]
        signal = reflected_version[1]
        a = -a
        scaled_time = []
        scaled_signal = []
        for i in range(0,len(time)):
            if (int(time[i] / a) == float(time[i] / a)):
                scaled_time.append(int(time[i] / a))
                scaled_signal.append(signal[i])
        return (scaled_time, scaled_signal)

################# PART B #################
def x_an_b(filepath):
    data = load_data_part_b(filepath)
    time = data[0]
    signal = data[1]
    a = data[2]
    b = data[3]
    if (b == 0):
```

```
        return time_scale(time, signal, a)
    elif (b > 0):
        shifted_version = time_shift(time, signal, abs(b), 0)
        return time_scale(shifted_version[0], shifted_version[1], a)
    elif (b < 0):
        shifted_version = time_shift(time, signal, abs(b), 1)
        return time_scale(shifted_version[0], shifted_version[1], a)
###########################################

################# PART B - Outputs #################
r = x_an_b("./chirp_part_b.csv")
plt.stem(r[0], r[1])
plt.xlabel('n', fontsize=20)
plt.ylabel('x[an + b]', fontsize=20)
plt.title("chirp_part_b.csv", fontsize=20)
plt.savefig("chirp_part_b.pdf", format="pdf", bbox_inches="tight")

r = x_an_b("./shifted_sawtooth_part_b.csv")
plt.stem(r[0], r[1])
plt.xlabel('n', fontsize=20)
plt.ylabel('x[an + b]', fontsize=20)
plt.title("shifted_sawtooth_part_b.csv", fontsize=20)
plt.savefig("shifted_sawtooth_part_b.pdf", format="pdf", bbox_inches="tight")

r = x_an_b("./sine_part_b.csv")
plt.stem(r[0], r[1])
plt.xlabel('n', fontsize=20)
plt.ylabel('x[an + b]', fontsize=20)
plt.title("sine_part_b.csv", fontsize=20)
plt.savefig("sine_part_b.pdf", format="pdf", bbox_inches="tight")
####################################################
```

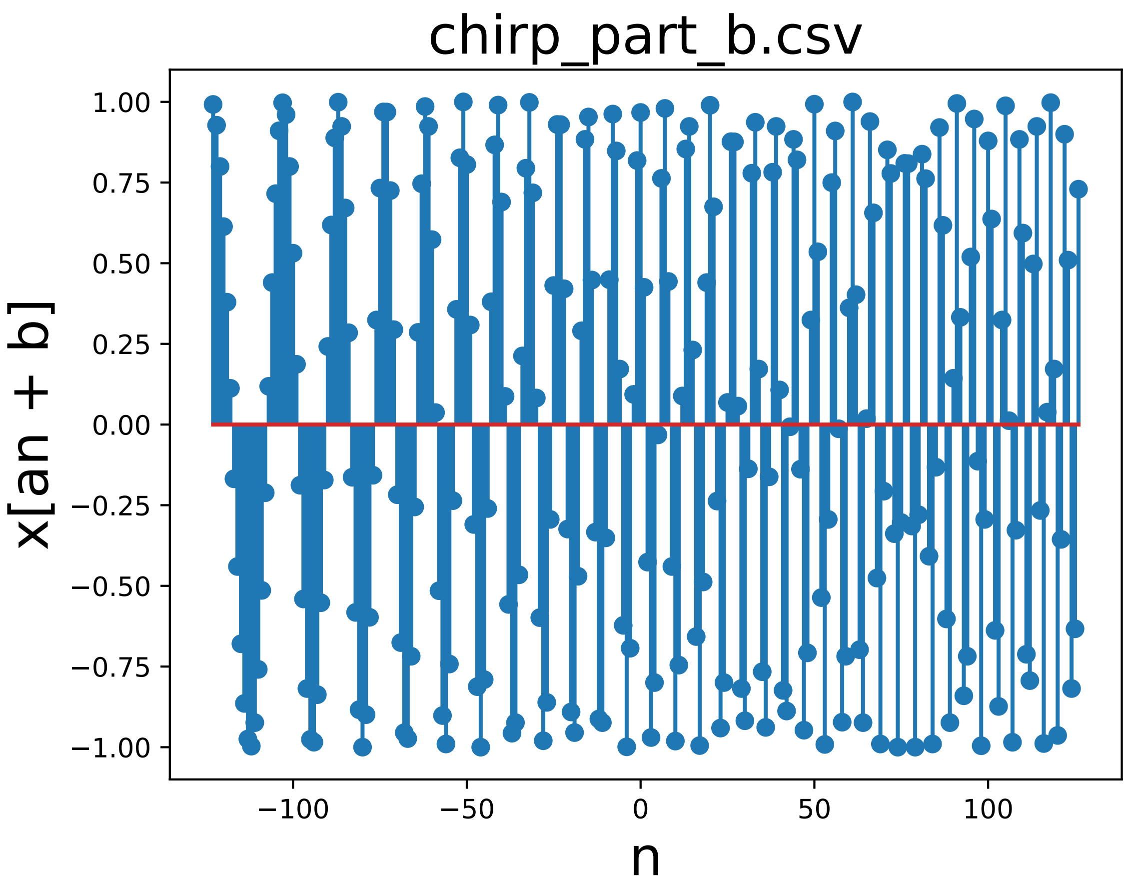

Figure 11: x[an+b] corresponding to chirp_part_b.csv

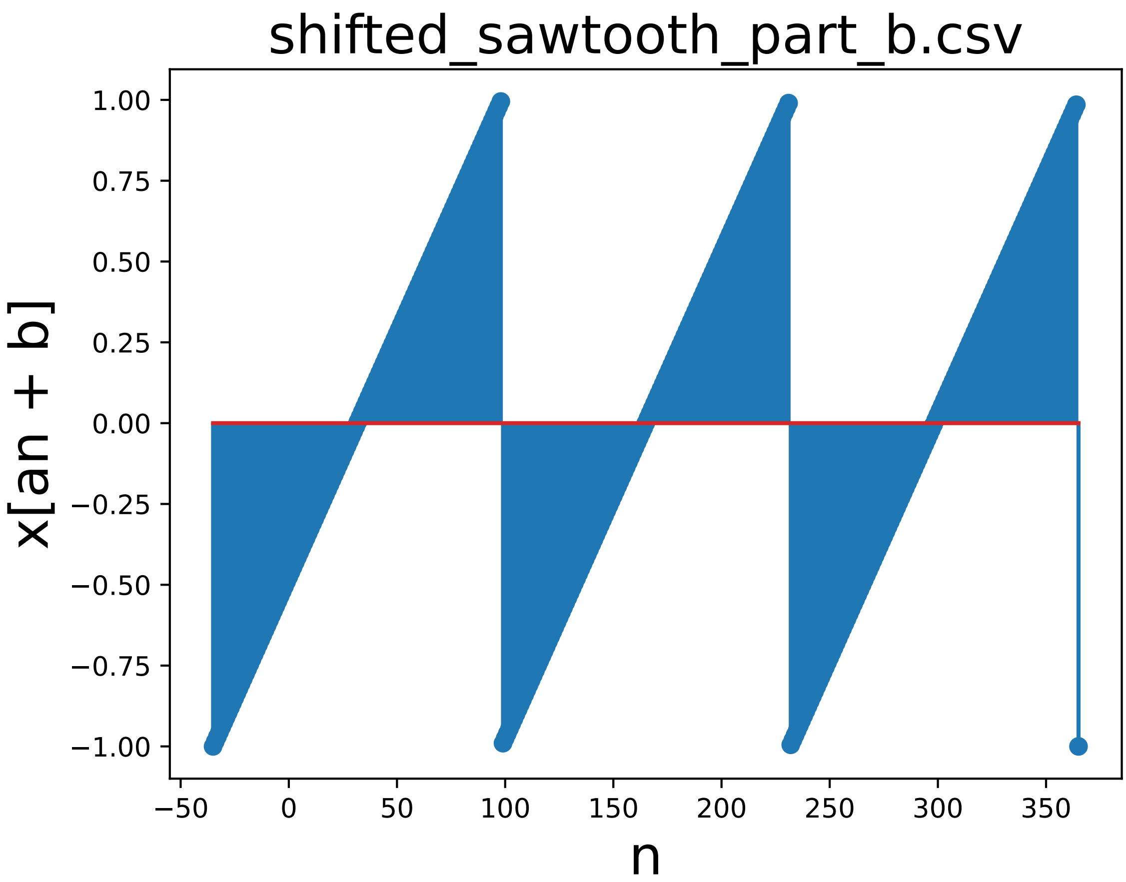

Figure 12: x[an+b] corresponding to shifted_sawtooth_part_b.csv

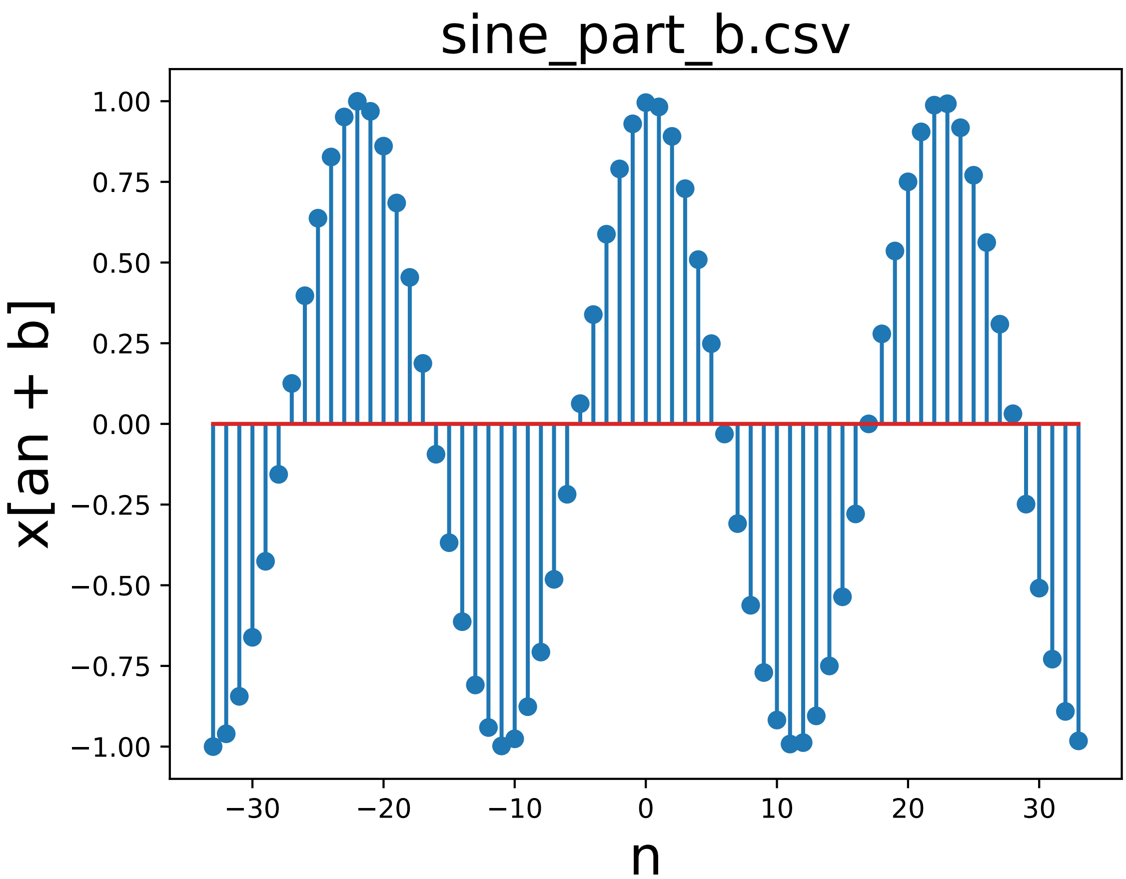

Figure 13: x[an+b] corresponding to sine_part_b.csv