

TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND
MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND
AUTOMATION

Laboratory Work No. 2

Convolution of two sequences and its properties

Author:

Catlabuga Corneliu,
st. gr. FAF-213

Checked By:

Railean Serghei,
university lecturer

Chisinau 2024

Objective:

Research on the convolution of two sequences and its properties.

Theoretical considerations:**Research on the convolution of two sequences and its properties.**

Linear systems analysis techniques.

There are two techniques for analyzing linear systems for the purpose of finding the response of the system to the given input signal. The first method is based on the direct solution of the input-output equation of the system, which has the form:

$$y(n) = F[y(n-1), y(n-2), \dots, y(n-N), x(n), x(n-1), \dots, x(n-M)] \quad (1)$$

For LTI systems, the general form of the input-output equation a system is:

$$y(n) = - \sum_{k=1}^N a_k y(n-k) + \sum_{k=0}^M b_k x(n-k) \quad (2)$$

where $\{a_k\}$ and $\{b_k\}$ are the constant parameters that specify the system and are independent of $x(n)$ and $y(n)$.

The second method of analyzing the behavior of the system at a given signal at input is based on decomposing the input signal into a sum of elementary signals. The elementary signals are chosen in such a way as the answer system to each elementary signal is easy to find. Then using linearity property of the system, responses of the system to signals elements are summed to find the total response of the system to the input signal.

Let the input signal $x(n)$ be decomposed into a scaled sum of elementary signals $\{x_k(n)\}$ as follows:

$$x(n) = \sum_k c_k x_k(n) \quad (3)$$

where $\{c_k\}$ is a set of amplitudes (scaling coefficients) in the decomposition of the signal $x(n)$. Let the response of the system to the elementary signal $x_k(n)$ be $y_k(n)$. Then:

$$y_k(n) = T[x_k(n)] \quad (4)$$

and the response of the system to $c_k x_k(n)$ is $c_k y_k(n)$, as a consequence of the property of scaling of the linear system.

Finally, the total response of the system to the input signal $x(n)$ is:

$$y(n) = \mathcal{T}[x(n)] = \mathcal{T}\left[\sum_k c_k x_k(n)\right] = \sum_k c_k \mathcal{T}[x_k(n)] = \sum_k c_k y_k(n) \quad (5)$$

In (5) we used the additivity property of the linear system.

If the input signal $x(n)$ is periodic with period N , it is more mathematically convenient to choose elementary signals:

$$x_k(n) = e^{j\omega_k n}, \quad k = 0, 1, \dots, N-1 \quad (6)$$

where the frequencies $\{\omega_k\}$ are associated (harmonically related) in the following way:

$$\omega_k = \left(\frac{2\pi}{N}\right)k, \quad k = 0, 1, \dots, N-1 \quad (7)$$

The frequency $2\pi / N$ is called the fundamental frequency.

For the resolution of the input signal into a weighted sum of units sample sequences, we must first determine the response of the system to a unit sample sequence and then use the scaling and multiplicative properties of the linear system to determine the formula for the output given any arbitrary input. This development is described in detail as follows.

Decomposition of the signal into elementary signals

Let us have an elementary signal $x(n)$ that we will decompose into a sum of pulses-unit. We will choose the elementary signal (pulse-unit) $x_k(n)$ in the murmuring way:

$$x_k(n) = \delta(n - k) \quad (8)$$

where k represents unit-impulse displacement.

Let's multiply the sequences $x(n)$ and $\delta(n - k)$. The sequence $\delta(n - k)$ is zero anywhere except $n = k$, where its value is equal to one. The outcome multiplication and another sequence, is zero everywhere except $n = k$, where its value is $x(k)$, as illustrated in figure 1. Then:

$$x(n)\delta(n - k) = x(k)\delta(n - k) \quad (9)$$

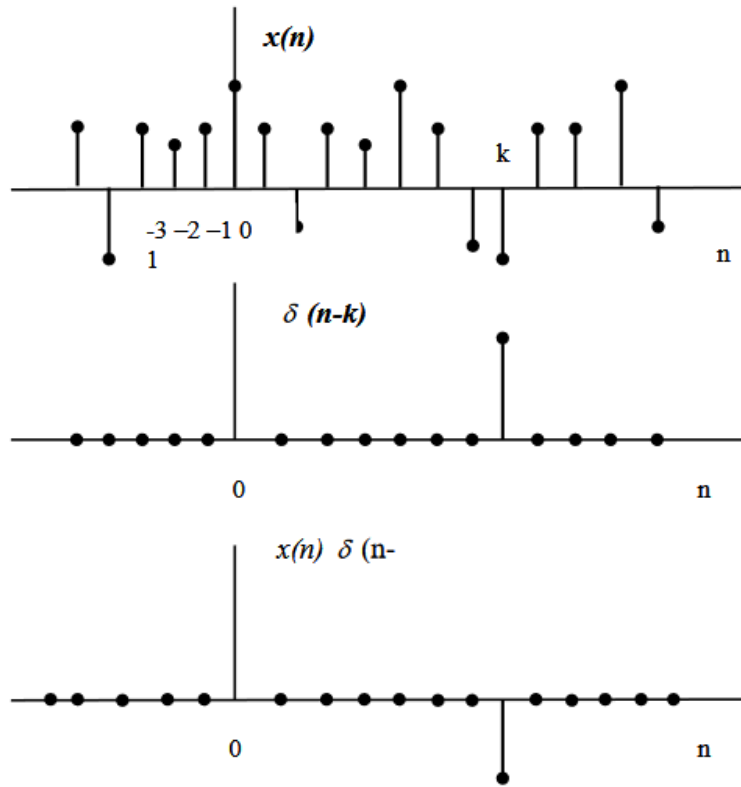


Figure 1

is a sequence that is zero everywhere except $n = k$, where its value is $x(k)$. If we repeat the multiplication of the signal $x(n)$ by $\delta(n - m)$, where m is another shift ($m \neq k$), the result will be another sequence that is zero everywhere except $n = m$, and its value will be $x(m)$. so

$$x(n)\delta(n - m) = x(m)\delta(n - m) \quad (9th)$$

In other words, each multiplication of the signal $x(n)$ with the impulse-unit at a displacement k , $[\delta(n - k)]$, locates the value $x(k)$ of the signal a of the $x(n)$ signal at the displacement where the unit impulse is nonzero.

Consequently, if we repeat the multiplication at all possible displacements, $-\infty < k < \infty$, and we add all the product-sequences, this result will be equal to the sequence $x(n)$, so

$$x(n) = \sum_{k=-\infty}^{\infty} x(k)\delta(n-k) \quad (9-b)$$

The right side of equality (9b) is a sum of an infinite number of unit pulses $\delta(n - m)$, which have amplitudes $x(k)$. So the right side of equality (9b) represents the decomposition of the arbitrary signal $x(n)$ into a scaled sum of shifted unit impulses.

Convolution sum.

Having the decomposition of the arbitrary signal $x(n)$ into a scaled sum of shifted unit-impulses, we can determine the response of the system to anything input signal. First we note the impulse response $y(n, k)$ of the system unity at $n = k$ through a special symbol y the special $h(k)$, $-\infty < k < \infty$. so,

$$y(n, k) \equiv h(k) = \mathcal{T}[\delta(n - k)] \quad (10)$$

In (10) we denoted by n the time index and by k the parameter which shows the location of the input pulse-unit. If the drive pulse is scaled with $ck = x(k)$, the system response will also be scaled:

$$c_k h(k) = x(k) h(n, k) \quad (11)$$

Finally, if the input is an arbitrary signal $x(n)$ and is expressed by a scaled sum of unit-pulses:

$$x(n) = \sum_{k=-\infty}^{\infty} x(k) \delta(n - k) \quad (12)$$

then the response of the system to $x(n)$ is correspondingly a sum of responses scaled:

$$y(n) = \mathcal{T}[x(n)] = \mathcal{T}\left[\sum_{k=-\infty}^{\infty} x(k) \delta(n - k)\right] = \sum_{k=-\infty}^{\infty} x(k) \mathcal{T}[\delta(n - k)] = \sum_{k=-\infty}^{\infty} x(k) h(n, k) \quad (13)$$

Expression (13) is the response of the linear system to an arbitrary signal of input $x(n)$. This expression is a function of both $x(n)$ and $\delta(n, k)$ – the response of the system to the unit impulse $\delta(n - k)$ for $-\infty < k < \infty$.

If the system is time invariant, expression (13) reduces considerable. If the response of the LTI system to the unit-impulse $\delta(n)$ is $h(n)$, then:

$$h(n) = \mathcal{T}[\delta(n)] \quad (14)$$

and according to the time-invariant property, the response of the system to the impulse-unit shifted $\delta(n - k)$ will be:

$$h(n - k) = \mathcal{T}[\delta(n - k)] \quad (15)$$

Consequently, formula (13) reduces to:

$$y(n) = \sum_{k=-\infty}^{\infty} x(k) h(n - k) \quad (16)$$

We note that the LTI system is completely characterized by the function $h(n)$, called its unit-impulse response $\delta(n)$.

Expression (16) shows the response $y(n)$ of the LTI system as a function of the signal of input $x(n)$ and the unit impulse response $h(n)$ and this expression is called the convolution sum. The input $x(n)$ is said to be in convolution with the answer $h(n)$ to find the answer $y(n)$.

Tasks:

Task 1:

```
import matplotlib.pyplot as plt

def main():
    a = [-2, 0, 1, -1, 3]
    b = [1, 2, 0, -1]
    d = 5
    n = [i for i in range(1, d + 1)]
    c = 4
    l = [i for i in range(1, c + 1)]

    plt.stem(n, a)
    plt.title('Sequence a')
    plt.xlabel('Time index n')
    plt.ylabel('Amplitude')
    plt.savefig('plots/task1a.png')

    plt.clf()

    plt.stem(l, b)
    plt.xlabel('Time index n')
    plt.ylabel('Amplitude')
    plt.title('Sequence b')
    plt.savefig('plots/task1b.png')

if __name__ == "__main__":
    main()
```

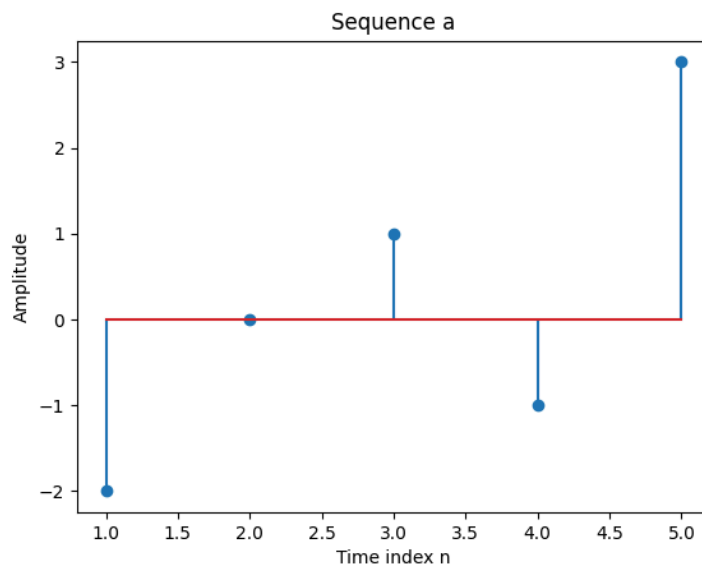



Figure 2: Task 1a

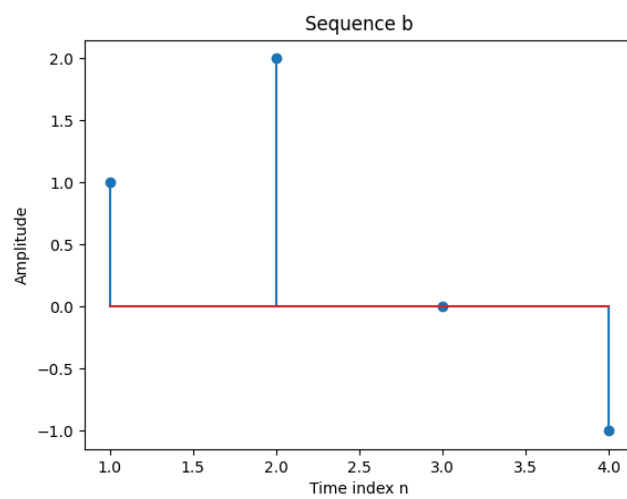


Figure 3: Task 1b

Task 2:

```
import matplotlib.pyplot as plt
from scipy.signal import convolve

def main():
    a = [-2, 0, 1, -1, 3]
    b = [1, 2, 0, -1]
    d = 5
    c = convolve(a, b)
    k = [i for i in range(1, 8 + 1)]

    plt.stem(k, c)
    plt.title('a and b convolved')
    plt.xlabel('Time index')
    plt.ylabel('Amplitude')
    plt.savefig('plots/task2.png')

if __name__ == "__main__":
    main()
```

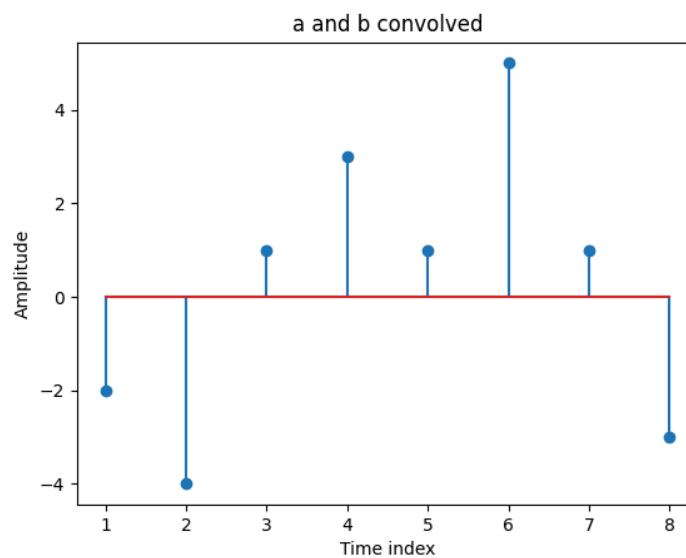


Figure 4: Task 2 a and b

Task 3:

```
import matplotlib.pyplot as plt
from numpy.fft import fft, ifft

def main():
    a = [-2, 0, 1, -1, 3]
    b = [1, 2, 0, -1]
    m = 8
    k = [i for i in range(1, m + 1)]
    AE = fft(a, m)
    BE = fft(b, m)
    p = AE * BE

    plt.stem(k, p)
    plt.xlabel('Index')
    plt.ylabel('Amplitude')
    plt.title('FFT of a * b')
    plt.savefig('plots/task3.png')

if __name__ == "__main__":
    main()
```

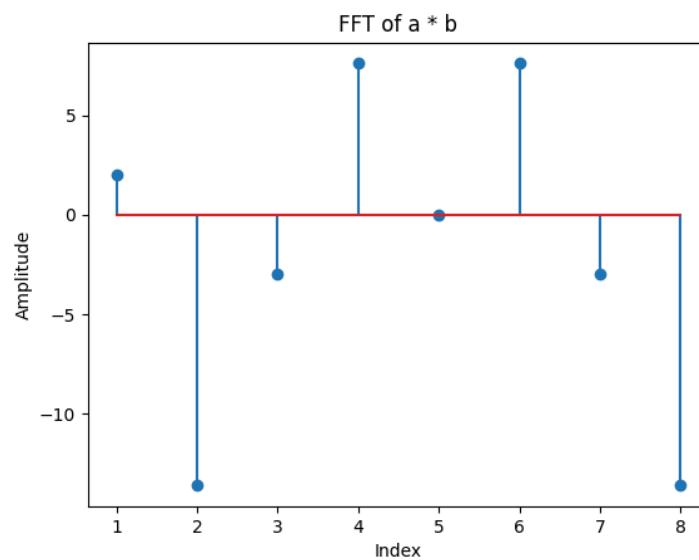


Figure 5: Task 3 FFT a*b

Task 4:

```
import matplotlib.pyplot as plt
from numpy.fft import fft, ifft

def main():
    a = [-2, 0, 1, -1, 3]
    b = [1, 2, 0, -1]
    m = 8
    k = [i for i in range(1, m + 1)]
    AE = fft(a, m)
    BE = fft(b, m)
    p = AE * BE
    y1 = ifft(p, m)

    plt.stem(k, y1)
    plt.xlabel('Index')
    plt.ylabel('Amplitude')
    plt.title('IFFT of a * b')
    plt.savefig('plots/task4.png')

if __name__ == "__main__":
    main()
```

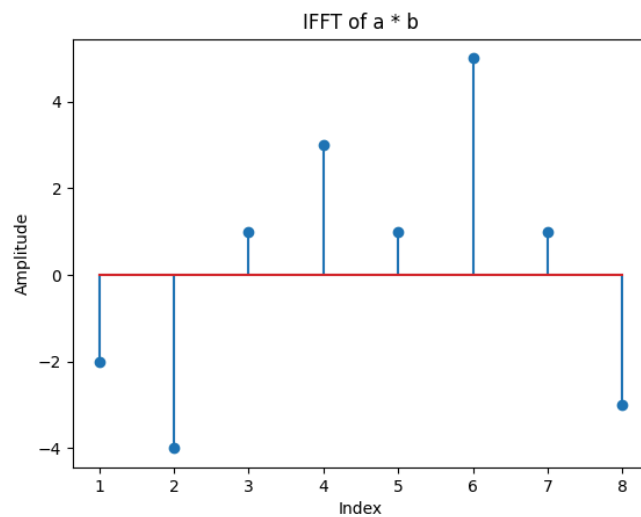


Figure 6: Task 4 IFFT a*b

Task 5:

```
import matplotlib.pyplot as plt
from scipy.signal import convolve
from numpy.fft import fft, ifft

def main():
    a = [-2, 0, 1, -1, 3]
    b = [1, 2, 0, -1]
    c = convolve(a, b)
    m = 8
    k = [i for i in range(1, m + 1)]

    AE = fft(a, m)
    BE = fft(b, m)
    p = AE * BE
    y1 = ifft(p, m)

    error = c - y1

    plt.stem(k, c, label='c')
    plt.xlabel('Index')
    plt.ylabel('Amplitude')
    plt.title('a and b convolved')
    plt.savefig('plots/task5a.png')

    plt.clf()

    plt.stem(k, y1, label='y1')
    plt.xlabel('Index')
    plt.ylabel('Amplitude')
    plt.title('IFFT of a * b')
    plt.savefig('plots/task5b.png')

    plt.clf()

    plt.stem(k, error, label='error')
    plt.xlabel('Index')
    plt.ylabel('Amplitude')
```

```
plt.title('Error')

plt.savefig('plots/task5c.png')

if __name__ == "__main__":
    main()
```

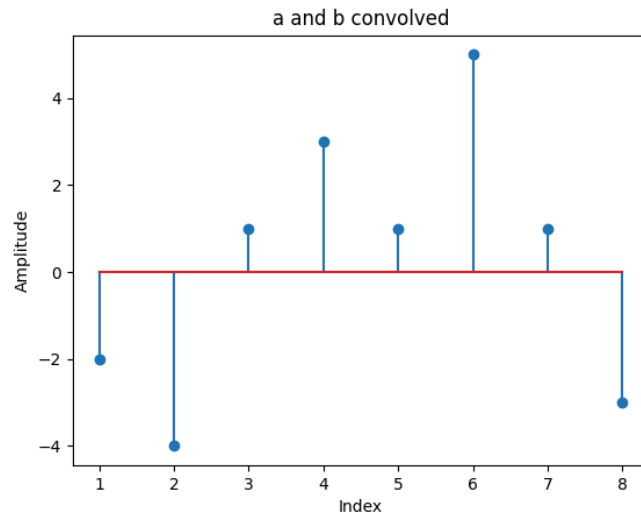


Figure 7: Task 5 conv a and b

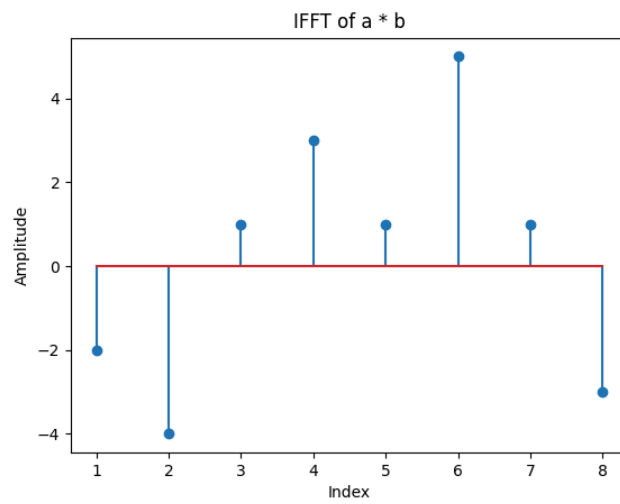


Figure 8: Task 5 IFFT a * b

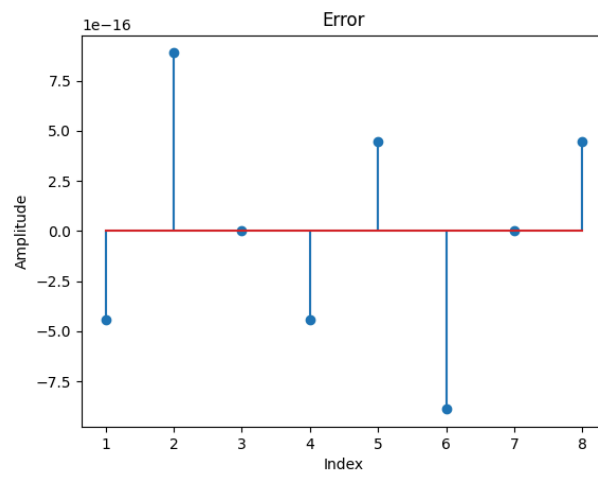


Figure 9: Task error in a and b

Task 6_1:

```
import matplotlib.pyplot as plt
from numpy import pi
from numpy.fft import fft, ifft
from scipy.signal import sawtooth, square, convolve

def main():
    d = 65536
    n = [i for i in range(1, d + 1)]
    a = 2.6 * square([28 * pi * i + 1 for i in n])

    c = 65536
    l = [i for i in range(1, c + 1)]
    b = 3.8 * sawtooth([22 * pi * i + 1 for i in l])

    m = d + c
    k = [i for i in range(1, m + 1)]

    plt.stem(n, a)
    plt.xlabel('Time index')
    plt.ylabel('Amplitude')
    plt.title('Signal a')
    plt.savefig('plots/task6.1a.png')

    plt.clf()

    plt.stem(l, b)
    plt.xlabel('Time index')
    plt.ylabel('Amplitude')
    plt.title('Signal b')
    plt.savefig('plots/task6.1b.png')

    plt.clf()

    c = convolve(a, b)

    plt.stem(k[0:-1], c)
    plt.title('a and b convolved')
```



```

plt.xlabel('Time index')
plt.ylabel('Amplitude')
plt.savefig('plots/task6.1c.png')

plt.clf()

AE = fft(a, m)
BE = fft(b, m)
p = AE * BE

plt.stem(k, p)
plt.xlabel('Index')
plt.ylabel('Amplitude')
plt.title('FFT of a * b')
plt.savefig('plots/task6.1d.png')

plt.clf()

y1 = ifft(p, m)

plt.stem(k, y1)
plt.xlabel('Index')
plt.ylabel('Amplitude')
plt.title('IFFT of a * b')
plt.savefig('plots/task6.1e.png')

if __name__ == "__main__":
    main()

```

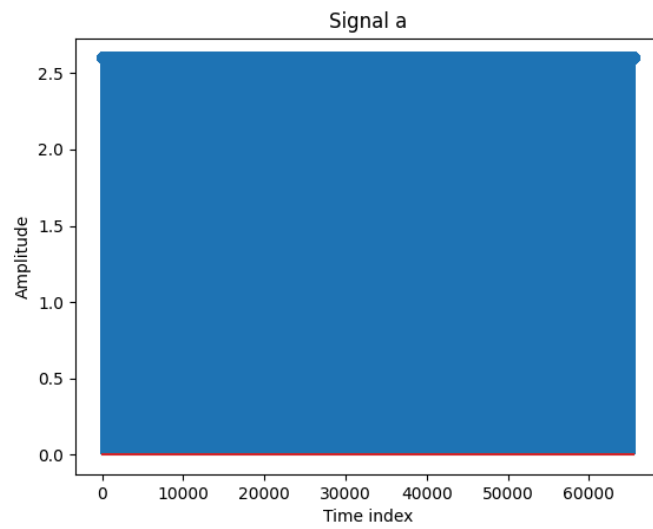


Figure 10: Task 6.1a

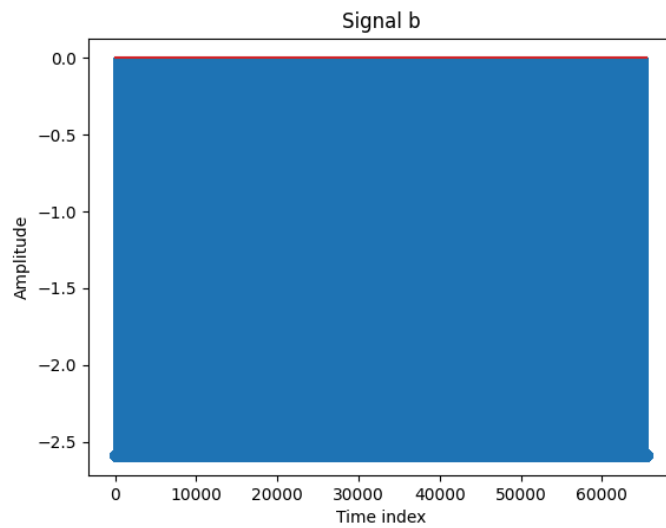


Figure 11: Task 6.1b

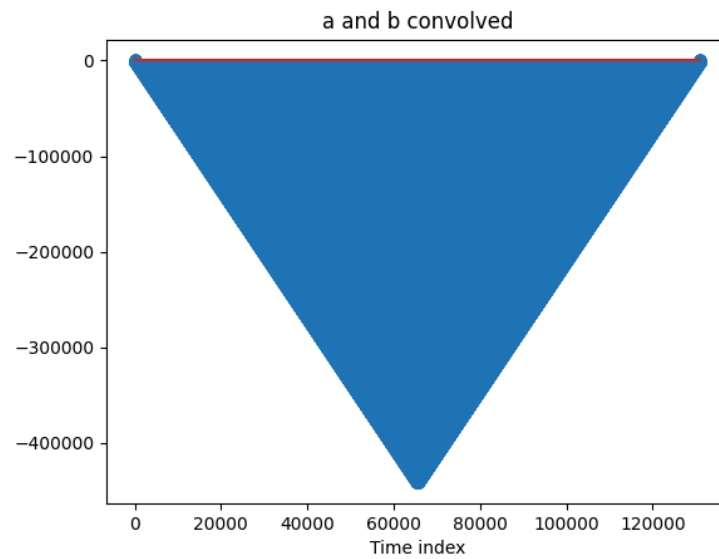


Figure 12: Task 6.1 conv a and b

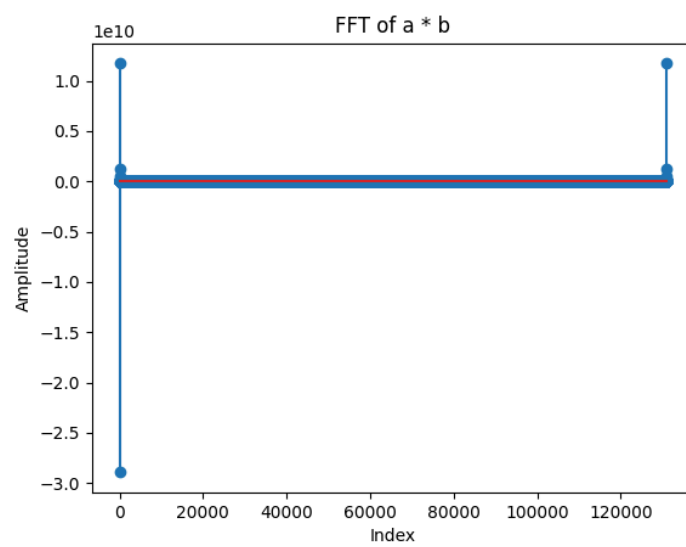


Figure 13: Task 6.1 FFT a * b

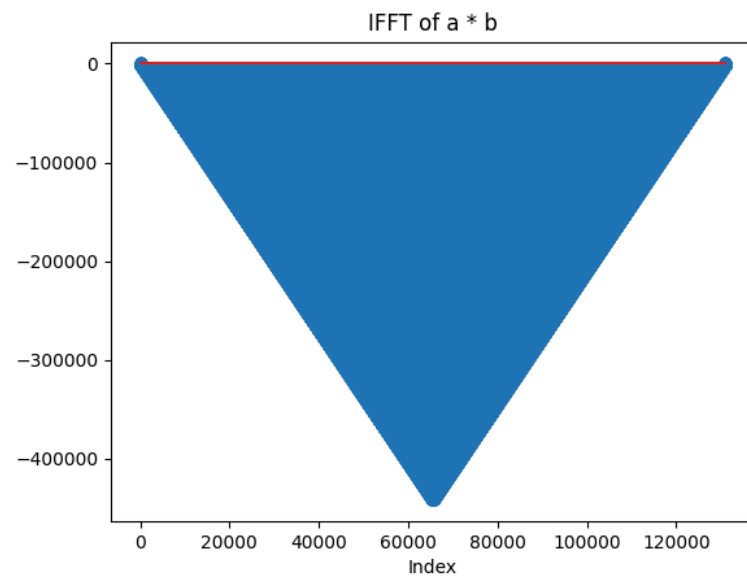


Figure 14: Task 6.1 IFFT $a * b$

Task 6_2:

```
import matplotlib.pyplot as plt
from numpy import pi
from numpy.fft import fft, ifft
from scipy.signal import sawtooth, square, convolve
from timeit import default_timer

def main():
    d = 65536
    n = [i for i in range(1, d + 1)]
    a = 2.6 * square([28 * pi * i + 1 for i in n])

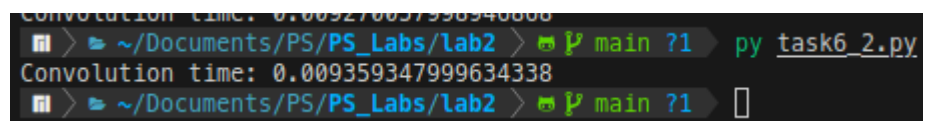
    c = 65536
    l = [i for i in range(1, c + 1)]
    b = 3.8 * sawtooth([22 * pi * i + 1 for i in l])

    m = d + c
    k = [i for i in range(1, m + 1)]

    start = default_timer()
    c = convolve(a, b)
    end = default_timer()

    print(f"Convolution time: {end - start}")

if __name__ == "__main__":
    main()
```



```
Convolution time: 0.009359347999634338
~/Documents/PS/PS_Labs/lab2 > main ?1 py task6_2.py
Convolution time: 0.009359347999634338
~/Documents/PS/PS_Labs/lab2 > main ?1
```

Figure 15: Task 6.2 conv time

Task 6_3:

```
import matplotlib.pyplot as plt
from numpy import pi
from numpy.fft import fft, ifft
from scipy.signal import sawtooth, square, convolve
from timeit import default_timer

def main():
    d = 65536
    n = [i for i in range(1, d + 1)]
    a = 2.6 * square([28 * pi * i + 1 for i in n])

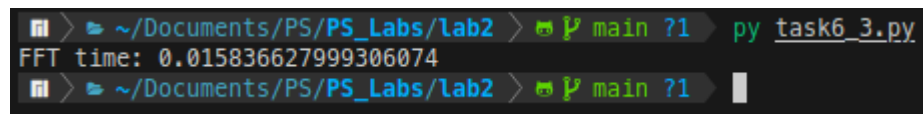
    c = 65536
    l = [i for i in range(1, c + 1)]
    b = 3.8 * sawtooth([22 * pi * i + 1 for i in l])

    m = d + c

    start = default_timer()
    AE = fft(a, m)
    BE = fft(b, m)
    p = AE * BE
    y1 = ifft(p, m)
    end = default_timer()

    print(f"FFT time: {end - start}")

if __name__ == "__main__":
    main()
```



```
> ~/Documents/PS/PS_Labs/lab2 > P main ?1 py task6_3.py
FFT time: 0.015836627999306074
> ~/Documents/PS/PS_Labs/lab2 > P main ?1
```

Figure 16: Task 6.3 FFT time

Task 7_2:

```
import matplotlib.pyplot as plt
from numpy import pi
from numpy.fft import fft, ifft
from scipy.signal import sawtooth, square, convolve
from timeit import default_timer

def main():
    d = 262144
    n = [i for i in range(1, d + 1)]
    a = 2.6 * square([28 * pi * i + 1 for i in n])

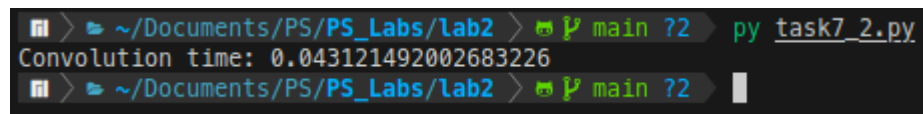
    c = 262144
    l = [i for i in range(1, c + 1)]
    b = 3.8 * sawtooth([22 * pi * i + 1 for i in l])

    m = d + c
    k = [i for i in range(1, m + 1)]

    start = default_timer()
    c = convolve(a, b)
    end = default_timer()

    print(f"Convolution time: {end - start}")

if __name__ == "__main__":
    main()
```



```
~> ~/Documents/PS/PS_Labs/lab2 > main ?2 py task7_2.py
Convolution time: 0.043121492002683226
~> ~/Documents/PS/PS_Labs/lab2 > main ?2
```

Figure 17: Task 7.2 conv time

Task 7_3:

```
import matplotlib.pyplot as plt
from numpy import pi
from numpy.fft import fft, ifft
from scipy.signal import sawtooth, square, convolve
from timeit import default_timer

def main():
    d = 262144
    n = [i for i in range(1, d + 1)]
    a = 2.6 * square([28 * pi * i + 1 for i in n])

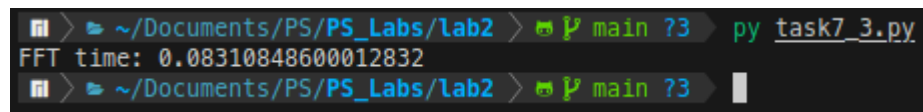
    c = 262144
    l = [i for i in range(1, c + 1)]
    b = 3.8 * sawtooth([22 * pi * i + 1 for i in l])

    m = d + c

    start = default_timer()
    AE = fft(a, m)
    BE = fft(b, m)
    p = AE * BE
    y1 = ifft(p, m)
    end = default_timer()

    print(f"FFT time: {end - start}")

if __name__ == "__main__":
    main()
```



```
~> ~/Documents/PS/PS_Labs/lab2 > P main ?3 py task7_3.py
FFT time: 0.08310848600012832
~> ~/Documents/PS/PS_Labs/lab2 > P main ?3
```

Figure 18: Task 7.3 FFT time

Task 8:

```
import matplotlib.pyplot as plt
from scipy.signal import convolve

def main():
    a = [1, 4, 2]
    b = [1, 2, 3, 4, 5, 4, 3, 3, 2, 2, 1, 1]
    c = convolve(a, b)
    m = 14
    k = [i for i in range(1, m + 1)]

    plt.stem(k, c)
    plt.xlabel('Index')
    plt.ylabel('Amplitude')
    plt.title('Convolution of a and b')
    plt.savefig('plots/task8.png')

if __name__ == "__main__":
    main()
```

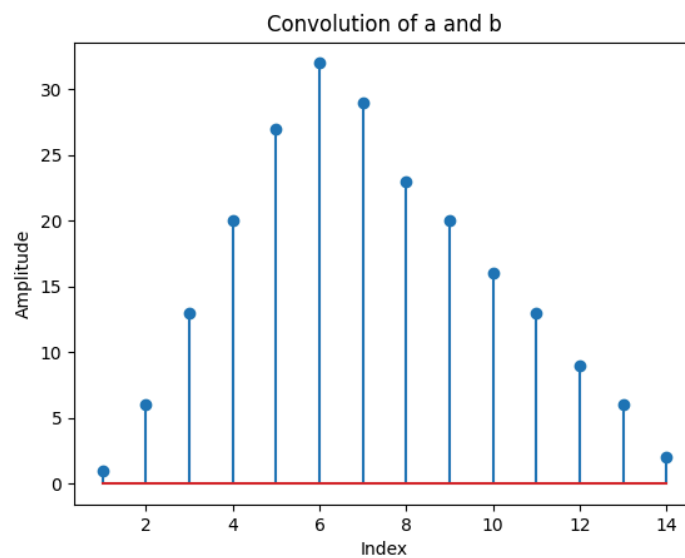


Figure 19: Task 8 conv a and b

Task 9:

```
import matplotlib.pyplot as plt
from scipy.signal import convolve

def main():
    a = [1, 4, 2]
    b = [1, 2, 3, 4, 5, 4, 3, 3, 2, 2, 1, 1]

    b1 = b[0:6]
    c1 = convolve(a, b1)

    b2 = b[6:12]
    c2 = convolve(a, b2)

    m = 8
    k = [i for i in range(1, m + 1)]

    plt.stem(k, c1)
    plt.xlabel('Index')
    plt.ylabel('Amplitude')
    plt.title('Convolution of a and b1')
    plt.savefig('plots/task9a.png')

    plt.clf()

    plt.stem(k, c2)
    plt.xlabel('Index')
    plt.ylabel('Amplitude')
    plt.title('Convolution of a and b2')
    plt.savefig('plots/task9b.png')

if __name__ == "__main__":
    main()
```

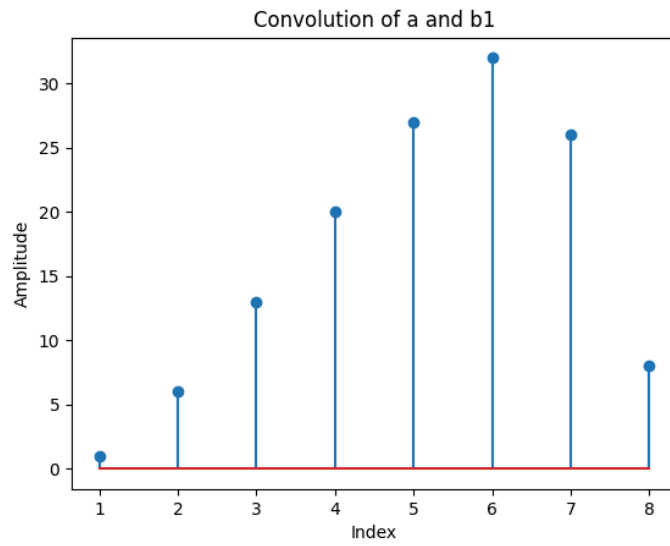


Figure 20: Task 9 conv a and b1

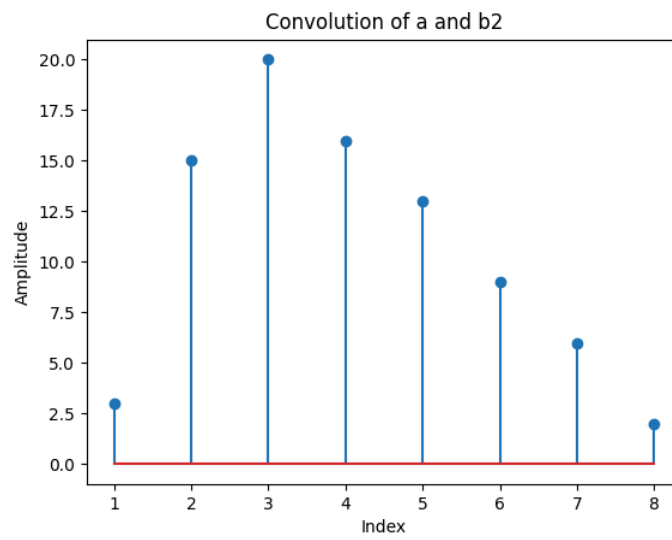


Figure 21: Task 9 conv a and b2

Task 10:

```
import matplotlib.pyplot as plt
from scipy.signal import convolve
import numpy as np

def main():
    a = [1, 4, 2]
    b = [1, 2, 3, 4, 5, 4, 3, 3, 2, 2, 1, 1]

    b1 = b[0:6]
    c1 = convolve(a, b1)

    b2 = b[6:12]
    c2 = convolve(a, b2)

    m = 14
    k = [i for i in range(1, m + 1)]

    c_add = np.concatenate((c1[0:6], c1[6:8] + c2[0:2], c2[2:]))

    plt.stem(k, list(c_add))
    plt.xlabel('Index')
    plt.ylabel('Amplitude')
    plt.title('Convolution of a and b')
    plt.savefig('plots/task10.png')

if __name__ == "__main__":
    main()
```

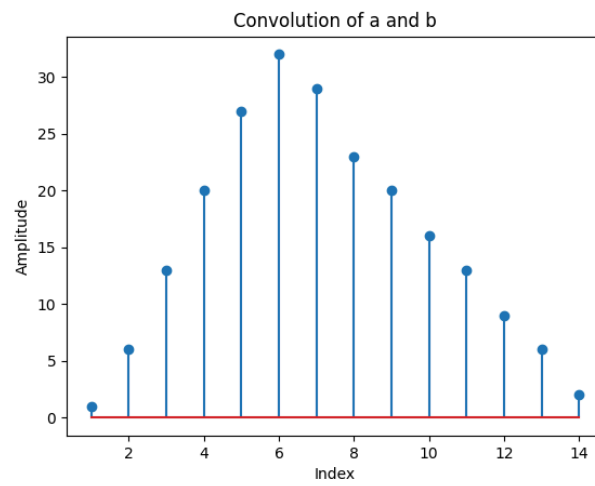


Figure 22: Task 10 conv a and b

Task 11:

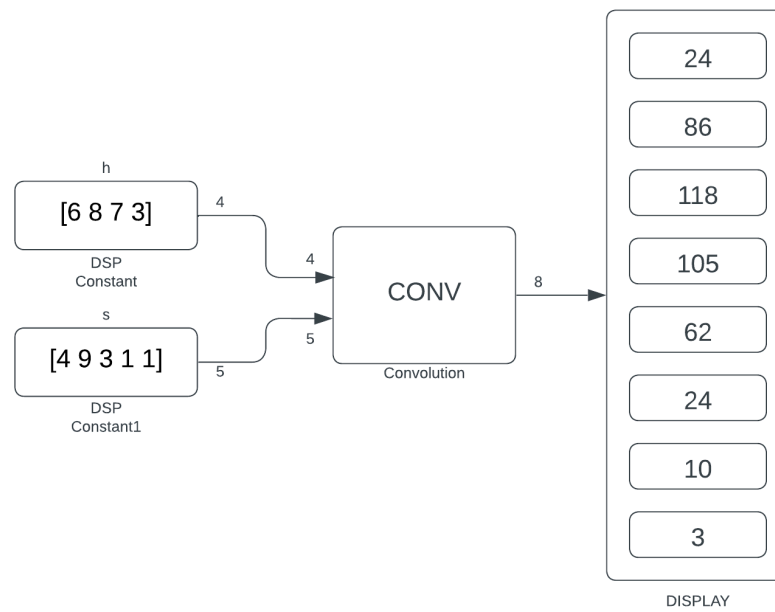


Figure 23: Task 11 block diagram of conv

Task 12:

```
import matplotlib.pyplot as plt
from numpy.fft import ifft
from numpy import array

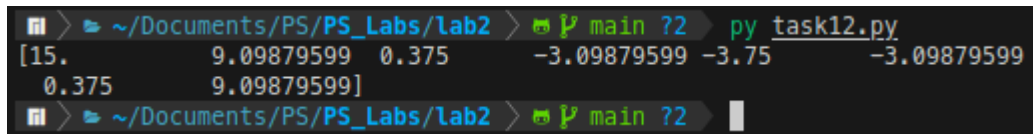
def main():
    d = 8
    n = [i for i in range(1, d + 1)]
    a = [6, 8, 7, 3, 0, 0, 0, 0]

    c = 8
    l = [i for i in range(1, c + 1)]
    b = [4, 9, 3, 1, 1, 0, 0, 0]

    ms = array(a) * array(b)
    ims = ifft(ms).real

    print(ims)

if __name__ == "__main__":
    main()
```

A terminal window with a dark background. The prompt is '~ /Documents/PS/PS_Labs/lab2 >'. The user enters 'python main ?2' and then 'py task12.py'. The output is displayed in two lines: '[15. 9.09879599 0.375 -3.09879599 -3.75 -3.09879599' and '0.375 9.09879599]'. The prompt is then '~ /Documents/PS/PS_Labs/lab2 > python main ?2' followed by a cursor.

```
~ /Documents/PS/PS_Labs/lab2 > python main ?2 py task12.py
[15.          9.09879599  0.375        -3.09879599 -3.75        -3.09879599
 0.375        9.09879599]
~ /Documents/PS/PS_Labs/lab2 > python main ?2
```

Figure 24: Task 12 procedural results

Conclusion:

The laboratory experiment focused on the application of convolution, Fast Fourier Transform (FFT), and Inverse Fast Fourier Transform (IFFT) on sequence pairs, with the aim of analyzing the output results graphically. Through systematic implementation, convolution highlighted the transformation effects on input sequences based on specified parameters such as kernel size, padding, and stride, showcasing its feature extraction and filtering capabilities. Additionally, FFT allowed for analysis of frequency components, revealing spectral characteristics and dominant frequencies in the input sequences, while IFFT facilitated reconstruction of the original signal from its frequency domain representation, validating the integrity of the FFT process.

Overall, the experiment underscored the utility of convolution, FFT, and IFFT in signal and image processing, offering insights into spectral properties and transformations applied to input sequences. Further exploration could delve into advanced techniques and practical applications across diverse fields, enhancing our understanding of these fundamental operations and their real-world impact in modern data processing systems.

References:

1. Github laboratory work repository:
https://github.com/muffindud/PS_Labs/tree/main/lab2