Ministry of Education and Research of the Republic of Moldova

Technical University of Moldova

Department of Software and Automation Engineering

REPORT

Individual Work No. 2

Discipline: Signal Processing

Elaborated: Berco Andrei, FAF - 221

Checked: Railean Serghei

Chișinău 2025

Purpose of the work

Research on the convolution of two sequences and its properties.

Theoretical Notes

1. Definition of Convolution:
   Convolution is a fundamental operation used in signal processing to determine the output of a Linear Time-Invariant (LTI) system when an input signal is applied. It represents how the system responds over time to that input.

2. Mathematical Expression:
   The convolution of two discrete-time signals x(n) and h(n) is given by:

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

3. System Analysis Techniques:
   - Direct Method: Solving the system's input-output difference equation.
   - Decomposition Method: Expressing the input signal as a sum of scaled elementary signals (unit impulses) and summing the corresponding scaled responses.

4. Unit Impulse Response:
   The response of the system to a unit impulse δ(n) is called the impulse response h(n). Any system response to arbitrary input can be obtained using the convolution of input x(n) with h(n).

5. Properties of Convolution:
   - Commutative: $x(n)*h(n)=h(n)*x(n)$
   - Associative: $x(n)*(h(n)*g(n))=(x(n)*h(n))*g(n)$
   - Distributive: $x(n)*[h(n)+g(n)]=x(n)*h(n)+x(n)*g(n)$

6. Frequency Domain Interpretation:
   Convolution in the time domain corresponds to multiplication in the frequency domain.

   $Y(f)=X(f)\cdot H(f)$

7. Fast Convolution via FFT:
   - Step 1: Compute Fourier Transforms of both sequences.
   - Step 2: Multiply the transforms.
   - Step 3: Apply Inverse FFT to get convolution result.
   - Advantage: Significantly faster for large signals.

8. Block Convolution:
   - Used when input signals are too long.
   - The signal is split into blocks, each convolved separately, then results are merged.
   - Techniques: Overlap-Add or Overlap-Save methods.

Practical Tasks

1. **Generate two finite sequences** a(n) and b(n) with lengths 5 and 4 respectively.

```
#TASK 1
import numpy as np
import matplotlib.pyplot as plt

# Define the sequences
a = np.array([-2, 0, 1, -1, 3])
b = np.array([1, 2, 0, -1])

d = len(a)
c = len(b)

n = np.arange(1, d + 1)
l = np.arange(1, c + 1)

# Plot the sequences
plt.figure(figsize=(8, 6))

plt.subplot(2, 1, 1)
plt.stem(n, a)
plt.xlabel('Indexul de timp n')
plt.ylabel('Amplitudine')
plt.title('Secventa a')
plt.grid()

plt.subplot(2, 1, 2)
plt.stem(l, b)
plt.xlabel('Indexul de timp n')
plt.ylabel('Amplituda')
plt.title('Secventa b')
plt.grid()

plt.tight_layout()
plt.show()
```

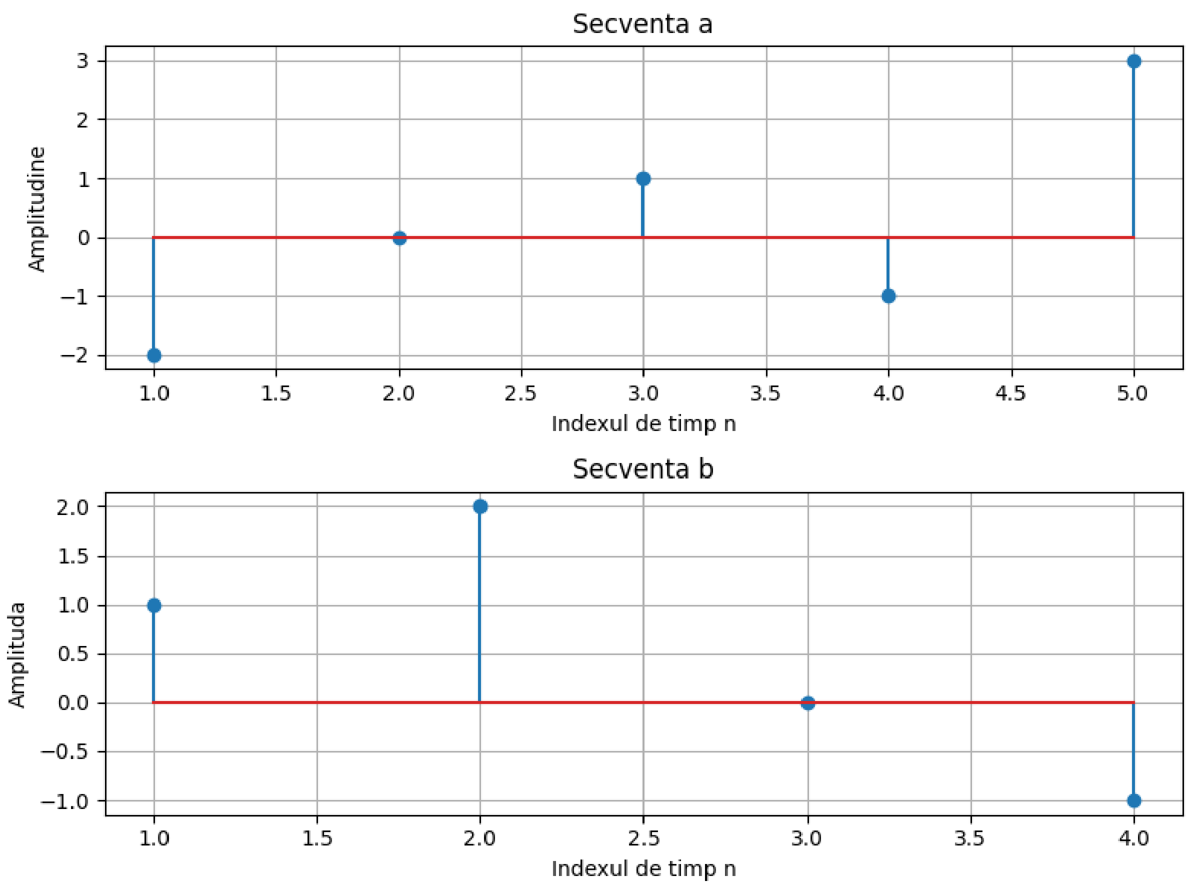

Figure 1: Result for task 1

The image displays two finite sequences: a(n) and b(n). These graphical representations effectively highlight the distinct amplitudes for a(n) and b(n) at their respective indices.

2. **Perform the convolution operation** of these two sequences using the conv function: c = conv(a, b);
   Display the signal in discrete form using the stem(k, c) function, specifying beforehand the length of the convolution as 8 (a + b – 1):
   k = 1:1:8; m = 8;.

```
#TASK 2

# Compute convolution
conv_result = np.convolve(a, b)

# Define length of convolution result
m = d + c - 1
k = np.arange(1, m + 1)

# Plot the sequences
plt.figure(figsize=(8, 8))
plt.stem(k, conv_result)
plt.xlabel('Indexul de timp k')
plt.ylabel('Amplituda')
plt.title('Convolutia secventelor a si b')
plt.grid()

plt.show()
```

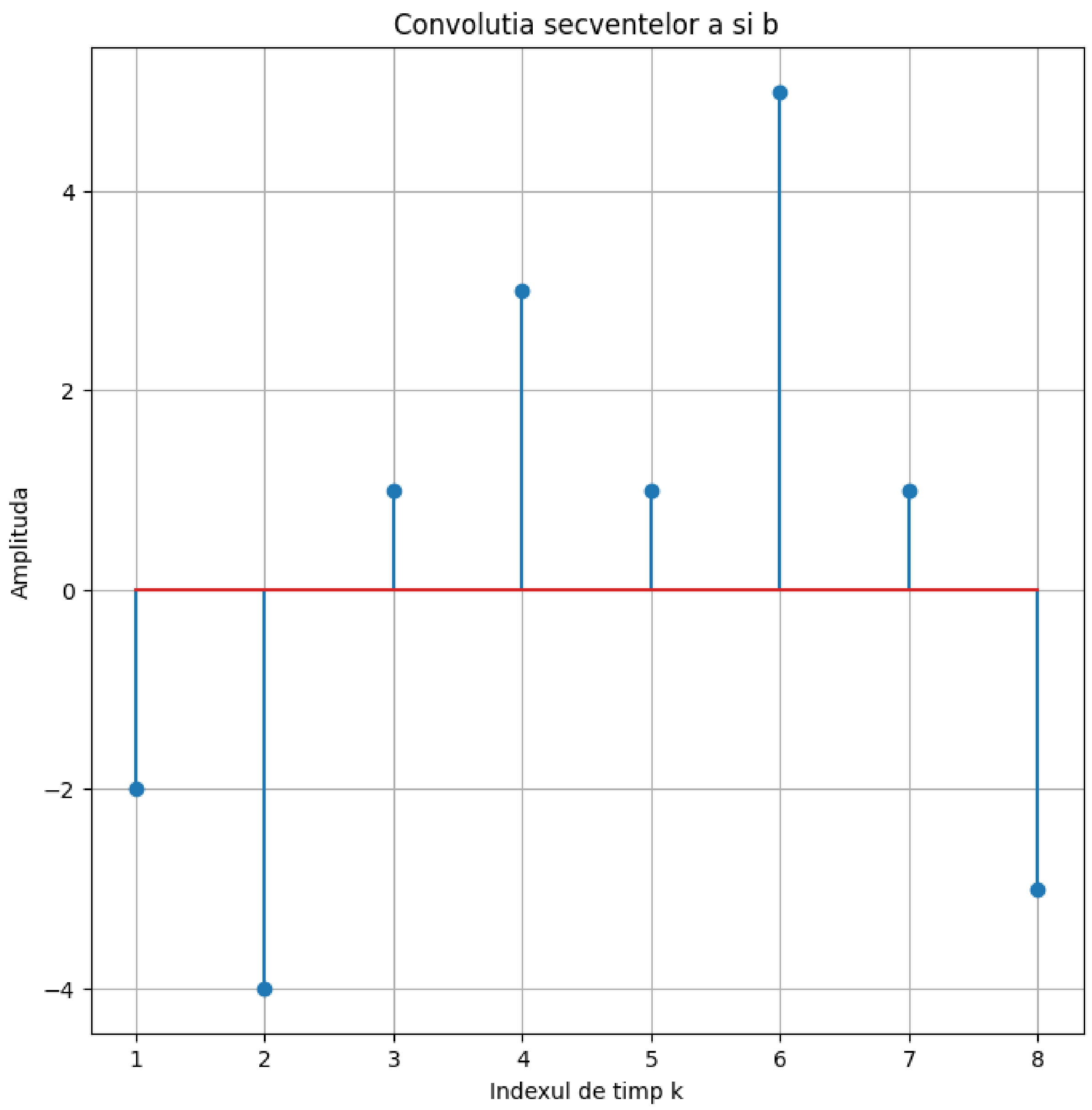

Figure 2. Result for Task 2

The image illustrates the convolution of the sequences a(n) and b(n), represented as a stem plot. This plot effectively displays the result of the convolution, a fundamental operation in signal processing that combines two sequences to generate a new one.

3. **Determine the Fourier Transform** of the sequences a(n) and b(n), then calculate the product of the obtained transforms:
   AE = fft(a, m); BE = fft(b, m); p = AE .* BE;
   This product is equal to the Fourier Transform of the convolution of signals a(n) and b(n). Display the signal in discrete form using stem(k, p).

```
#TASK 3

# Compute Fourier Transform of convolution result
conv_result_fft = np.fft.fft(conv_result, m)

# Compute Fourier Transform of sequences
AE = np.fft.fft(a, m)
BE = np.fft.fft(b, m)
p = AE * BE

# Plot the sequences
plt.figure(figsize=(8, 10))
plt.subplot(2, 1, 1)
plt.stem(k, conv_result_fft)
plt.xlabel('Indexul de timp k')
plt.ylabel('Amplituda')
plt.title('Transformarea Fourier a Convolutie secventelor a si b')
plt.grid()

plt.subplot(2, 1, 2)
plt.stem(k, np.real(p))
plt.xlabel('Indexul de timp k')
plt.ylabel('Amplituda')
plt.title('Produsul Transformarilor Fourier ale secventelor a si b')
plt.grid()


plt.tight_layout()
plt.show()
```

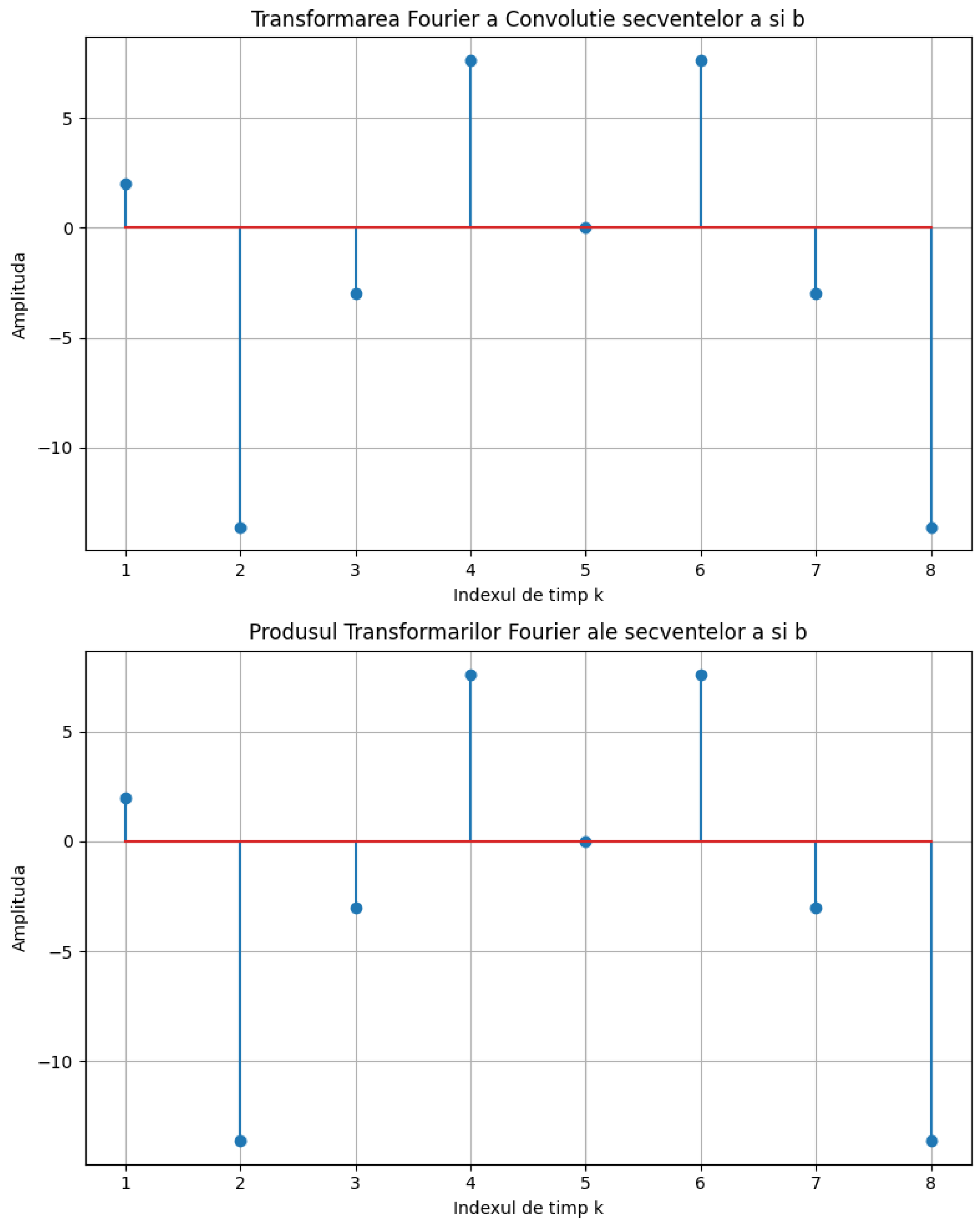

Figure 3. Result for Task 3

The image represents two distinct stem plots showcasing results related to Fourier Transform analysis and its application to convolution.

Fourier Transform of the Convolution:

- The first plot visually represents the Fourier Transform of the convolution of sequences a(n) and b(n).

- This serves as a verification step, illustrating how the Fourier domain captures the combined information of both sequences after convolution.

Product of Fourier Transforms:

- The second plot displays the product of the Fourier Transforms of the individual sequences a(n) and b(n).
- This directly corresponds to the fundamental theorem of convolution, linking the time and frequency domains: the convolution in the time domain equals the multiplication in the frequency domain.

The stem plots effectively demonstrate the amplitudes at various discrete points, emphasizing how the two processes interrelate. It's a precise validation of the mathematical equivalence between convolution and Fourier multiplication

4. **By calculating the Inverse Fourier Transform** of the product of the Fourier Transforms of a(n) and b(n) we obtain the convolution of the signals:
   y1 = ifft(p); Display the signal y1 in discrete form.

```
#Task 4

# Compute inverse Fourier Transform to verify
ifft_result = np.fft.ifft(p)

# Plot the sequences
plt.figure(figsize=(8, 10))

plt.stem(k, np.real(ifft_result))
plt.xlabel('Indexul de timp k')
plt.ylabel('Amplituda')
plt.title('Transformata Fourier Inversa a produsului')
plt.grid()

plt.tight_layout()
plt.show()
```

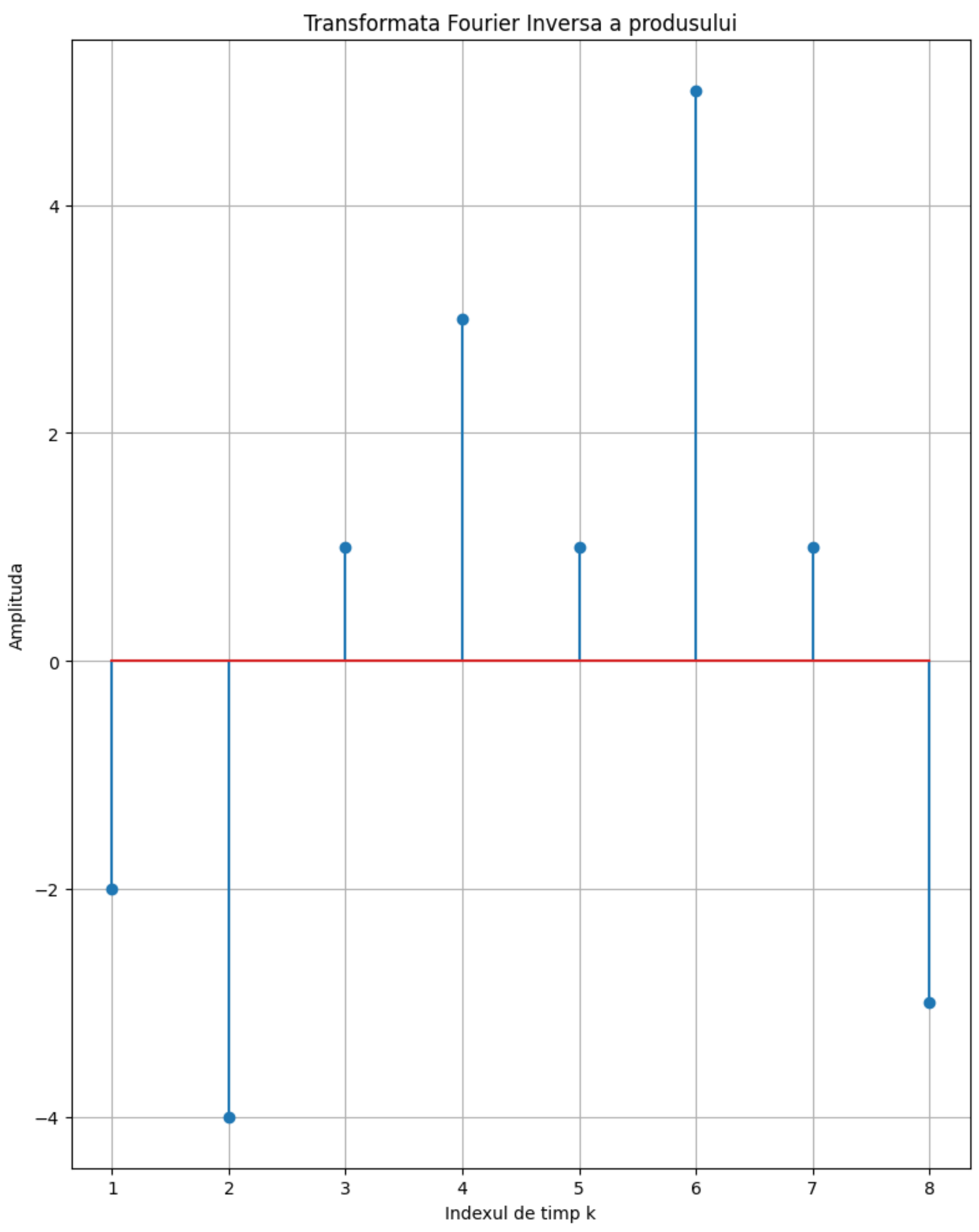

Figure 4: Result for Task 4

The image illustrates the result of calculating the Inverse Fourier Transform of the product of the Fourier Transforms of a(n) and b(n), which effectively reconstructs the convolution of these two signals. The displayed signal, y, is identical in form to the convolution result derived previously. This visualization confirms the fundamental relationship between the Fourier domain and the time domain in signal processing, showcasing the equivalence of convolution and frequency-domain multiplication. It provides a clear and discrete representation of the reconstructed convolution.

5. **Compare the obtained convolution** with the initial one. Calculate the error between the initial convolution and the one obtained:
   `error = c - y1;` Display the signals `c`, `y1`, and `error` in discrete form in three separate windows using: `subplot(3,1,1); stem(k, c)` for the first signal, then change the third parameter in `subplot` to 2 and 3 for the other two signals: `subplot(3,1,2);subplot(3,1,3);`.

```
#Task 5

# Compute error between convolution results
error = conv_result - np.real(ifft_result)

# Plot comparison of convolution results and error
plt.figure(figsize=(8, 9))

plt.subplot(3, 1, 1)
plt.stem(k, conv_result)
plt.xlabel('Indexul de timp k')
plt.ylabel('Amplituda')
plt.title('Convolutia originala')
plt.grid()

plt.subplot(3, 1, 2)
plt.stem(k, np.real(ifft_result))
plt.xlabel('Indexul de timp k')
plt.ylabel('Amplituda')
plt.title('Convolutia obtinuta prin Transformata Fourier Inversa')
plt.grid()

plt.subplot(3, 1, 3)
plt.stem(k, error)
plt.xlabel('Indexul de timp k')
plt.ylabel('Eroare')
plt.title('Eroarea dintre convolutii')
plt.grid()

plt.tight_layout()
plt.show()
```

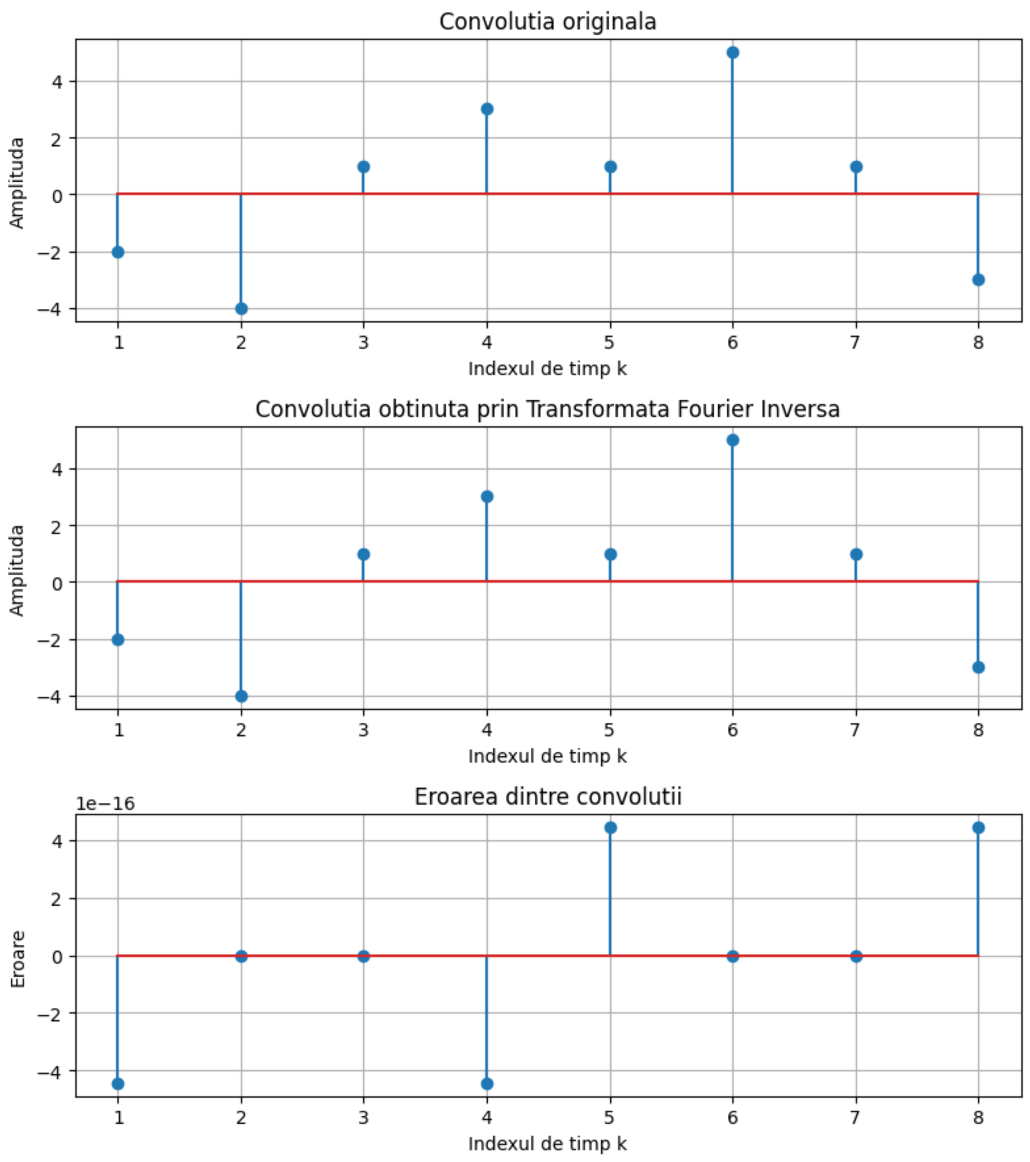

Figure 5: Result for task 5

The image contains three subplots showcasing the comparison of signals and the calculated error:

1. **Original Convolution Signal:** The first subplot represents the original convolution cc, displayed as a discrete signal. It retains its form as previously described, serving as the reference point for comparison.
2. **Reconstructed Convolution Signal:** The second subplot visualizes y_1, which is the convolution obtained through the Inverse Fourier Transform. It closely mirrors the original convolution, demonstrating the validity of the reconstruction process.
3. **Error Signal:** The third subplot illustrates the error between the original convolution and the reconstructed one. The error values are extremely small and appear negligible, essentially verifying that the reconstruction faithfully reproduces the original convolution.

This comparison confirms the accuracy of the Fourier Transform and its inverse for convolution operations, with the error signal being practically zero.

6. **Repeat steps 1–4 for two other signals with significantly longer lengths** in order to observe the time difference between the two methods.
7. **Repeat step 6 for two other even longer signals** in order to observe the time differences between the two methods for various signal lengths.

To not repeat the code, I combined this two tasks, giving an array of values for the signals length:

```
#Task 6, 7

import time

from scipy.signal import square, sawtooth

import matplotlib.pyplot as plt

import numpy as np



n_s = [16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768,
65536, 131072, 262144, 524288]

conv_time = []

fft_time = []



plt.figure(figsize=(20, 20))



for n in n_s:

    N = n

    L = n

    a = 2 * square(20 * np.pi * np.arange(N) + 1)

    b = 3 * sawtooth(20 * np.pi * np.arange(L) + 1)



    # Compute convolution time

    tic = time.time()
    conv = np.convolve(a, b)

    toc = time.time()

    print(f'Timpul pentru convolutie directa: {toc - tic} secunde pentru N
= {N} si L = {L}')

    conv_time.append(toc - tic)



    # Compute Fourier Transform time

    m = n + L - 1

    tic = time.time()

    AE = np.fft.fft(a, m)

    BE = np.fft.fft(b, m)

    p = AE * BE

    ifft = np.fft.ifft(p)

    toc = time.time()

    fft_time.append(toc - tic)

    print(f'Timpul pentru convolutie prin FFT: {toc - tic} secunde pentru N
= {N} si L = {L}')



    # Plot the results dynamically

    plt.clf()



    # Subplot for direct convolution result

    plt.subplot(2, 1, 1)

    plt.stem(np.arange(len(conv)), conv)

    plt.xlabel('Indexul de timp')

    plt.ylabel('Amplitudine')

    plt.title('Rezultatul convolutiei directe')

    plt.grid()
    # Subplot for execution times

    plt.subplot(2, 1, 2)

    plt.plot(n_s[:len(conv_time)], conv_time, label='Convolutie directa',
color='blue')

    plt.plot(n_s[:len(fft_time)], fft_time, label='Convolutie prin FFT',
color='red')

    plt.xlabel('Dimensiunea secventei')

    plt.ylabel('Timpul de executie (s)')

    plt.title('Timpul de executie pentru convolutie directa si prin FFT')

    plt.legend()

    plt.grid()



    plt.tight_layout()

    plt.pause(0.1)  # Pause to update the plot



plt.show()



# Plot the results

plt.figure(figsize=(12, 10))



# Subplot for direct convolution time

plt.subplot(2, 1, 1)

plt.plot(n_s, conv_time, label='Convolutie directa', color='blue')

plt.xlabel('Dimensiunea secventei')

plt.ylabel('Timpul de executie (s)')

plt.title('Timpul de executie pentru convolutie directa')

plt.legend()
plt.grid()


# Subplot for FFT convolution time

plt.subplot(2, 1, 2)

plt.plot(n_s, fft_time, label='Convolutie prin FFT', color='red')

plt.xlabel('Dimensiunea secventei')

plt.ylabel('Timpul de executie (s)')

plt.title('Timpul de executie pentru convolutie prin FFT')

plt.legend()

plt.grid()


plt.tight_layout()

plt.show()
```

Timpul pentru convolutie directa: 5.626678466796875e-05 secunde pentru N = 16 si L = 16

Timpul pentru convolutie prin FFT: 0.0002200603485107422 secunde pentru N = 16 si L = 16

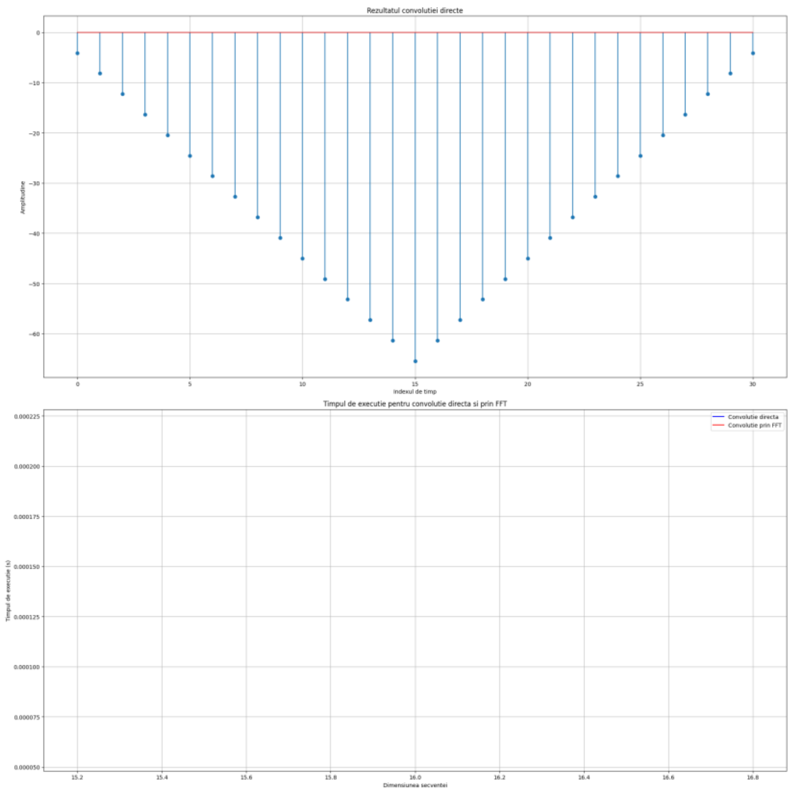

Figure 6.1: Results for length 16

Timpul pentru convolutie directa: 4.267692565917969e-05 secunde pentru N = 32 si L = 32

Timpul pentru convolutie prin FFT: 0.00030279159545898844 secunde pentru N = 32 si L = 32

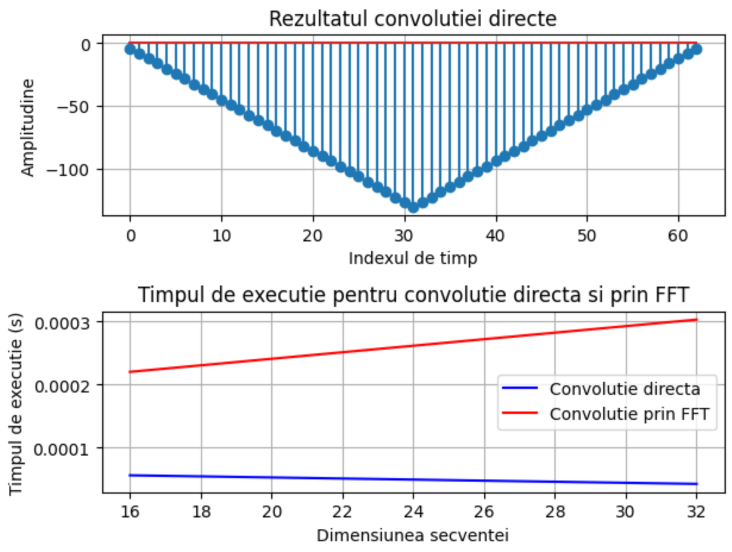

Figure 6.2: Results for length 32

Timpul pentru convolutie directa: 4.863739013671875e-05 secunde pentru N = 64 si L = 64

Timpul pentru convolutie prin FFT: 0.00029397010803222656 secunde pentru N = 64 si L = 64

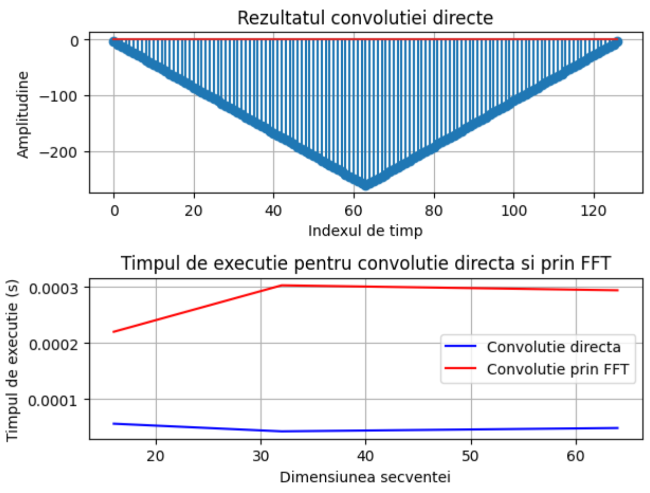

Figure 6.3: Result for length 64

Timpul pentru convolutie directa: 3.075599670410156e-05 secunde pentru N = 128 si L = 128

Timpul pentru convolutie prin FFT: 0.00014853477478027344 secunde pentru N = 128 si L = 128

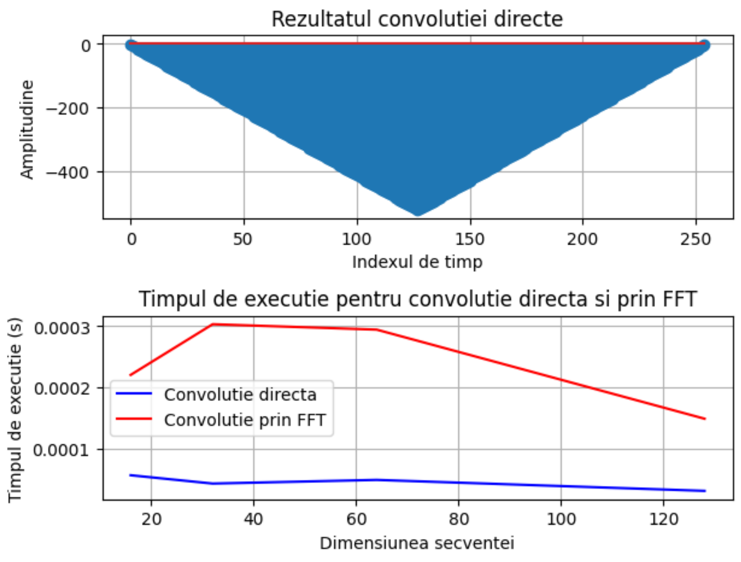

Figure 6.4: Results for length 128

Timpul pentru convolutie directa: 5.5789947509765625e-05 secunde pentru N = 256 si L = 256

Timpul pentru convolutie prin FFT: 0.0002422332763671875 secunde pentru N = 256 si L = 256

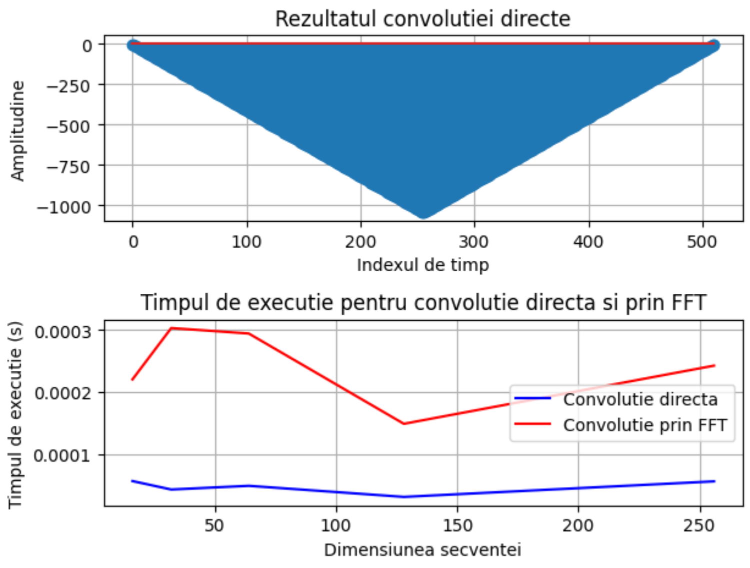

Figure 6.5: Result for length 256

Timpul pentru convolutie directa: 0.00017261505126953125 secunde pentru N = 512 si L = 512

Timpul pentru convolutie prin FFT: 0.03216052055358887 secunde pentru N = 512 si L = 512

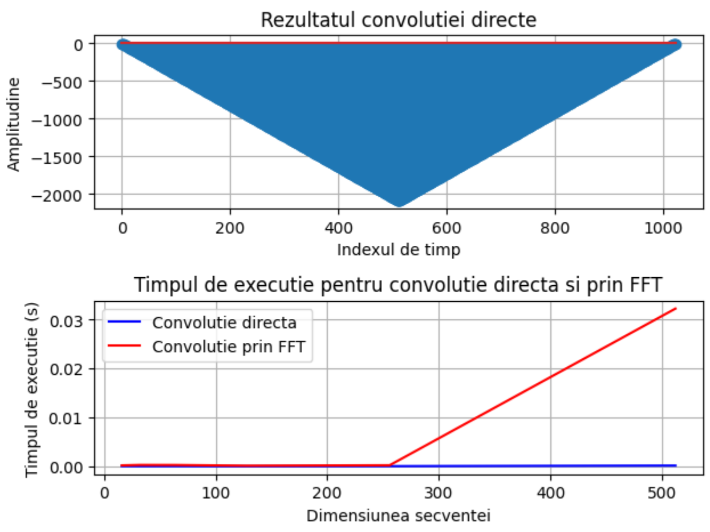

Figure 6.6: Results for length 512

Timpul pentru convolutie directa: 0.0002665519714355469 secunde pentru N = 1024 si L = 1024

Timpul pentru convolutie prin FFT: 0.0006117820739746094 secunde pentru N = 1024 si L = 1024

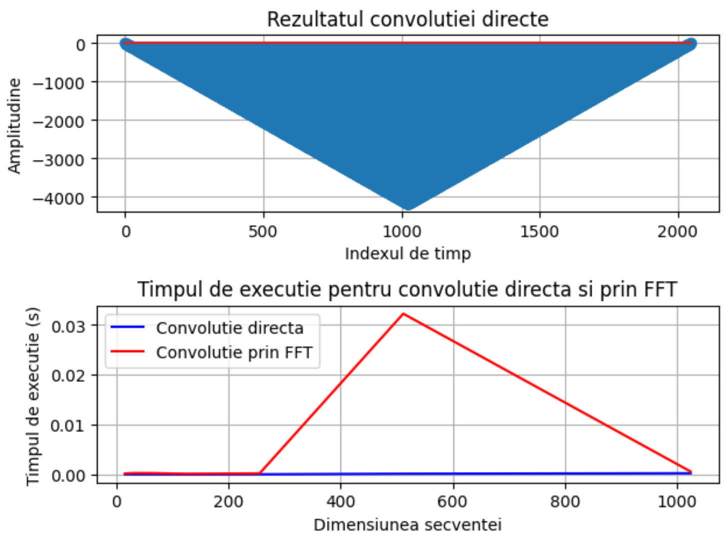

Figure 6.7: Results for length 1024

Timpul pentru convolutie directa: 0.0009107589721679688 secunde pentru N = 2048 si L = 2048

Timpul pentru convolutie prin FFT: 0.0005204677581787109 secunde pentru N = 2048 si L = 2048

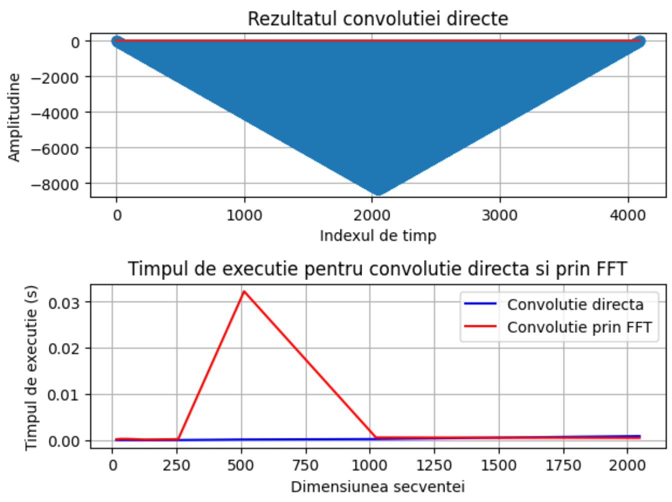

Figure 6.8: Result for length 2048

Timpul pentru convolutie directa: 0.0017452239990234375 secunde pentru N = 4096 si L = 4096

Timpul pentru convolutie prin FFT: 0.0015518665313720703 secunde pentru N = 4096 si L = 4096

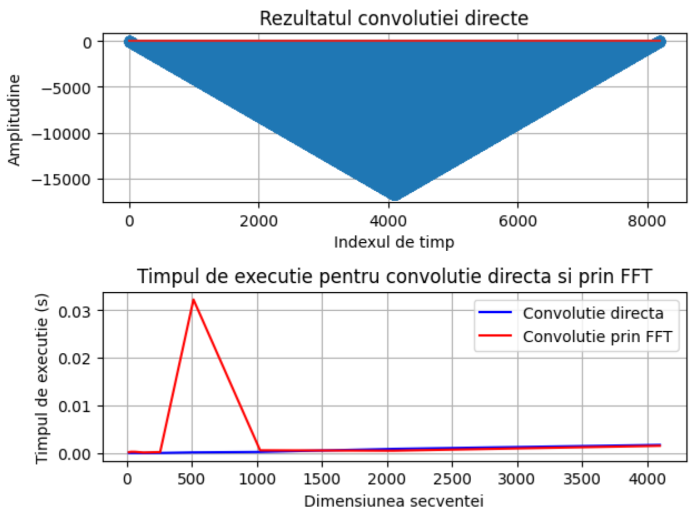

Figure 6.9: Results for length 4096

Timpul pentru convolutie directa: 0.006844282150268555 secunde pentru N = 8192 si L = 8192

Timpul pentru convolutie prin FFT: 0.0023183822631835938 secunde pentru N = 8192 si L = 8192

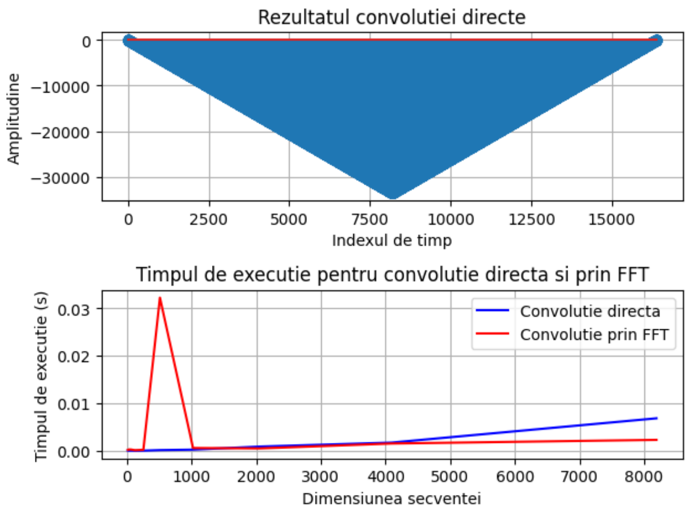

Figure 6.10: Results for length 8192

Timpul pentru convolutie directa: 2.0301363468170166 secunde pentru N = 16384 si L = 16384

Timpul pentru convolutie prin FFT: 0.006056308746337891 secunde pentru N = 16384 si L = 16384

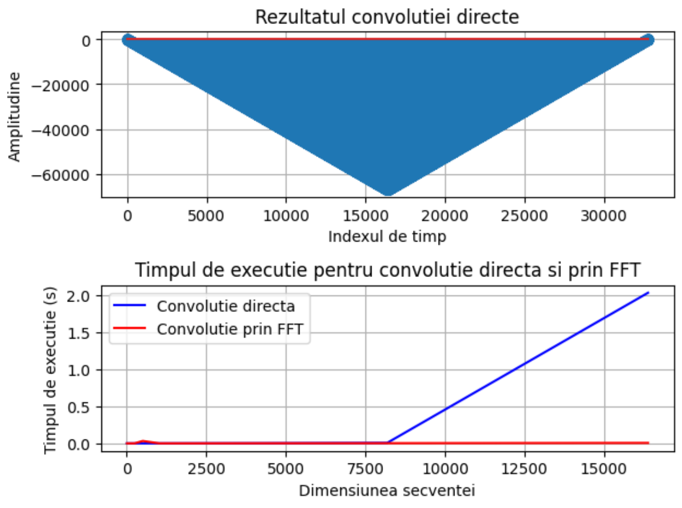

Figure 6.11: Results for length 16384

Timpul pentru convolutie directa: 9.21169900894165 secunde pentru N = 32768 si L = 32768

Timpul pentru convolutie prin FFT: 0.034094810485839844 secunde pentru N = 32768 si L = 32768

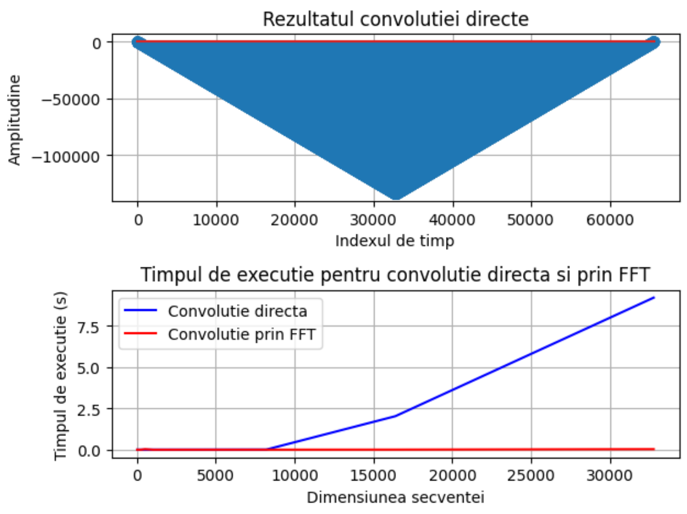

Figure 6.12: Results for length 32768

Timpul pentru convolutie directa: 22.035715103149414 secunde pentru N = 65536 si L = 65536

Timpul pentru convolutie prin FFT: 0.05417776107788086 secunde pentru N = 65536 si L = 65536

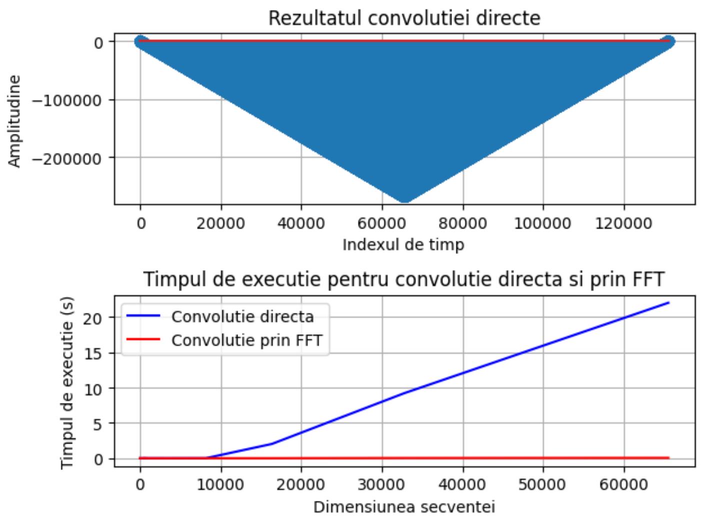

Figure 6.13: Results for length 65536

Timpul pentru convolutie directa: 56.987982988357544 secunde pentru N = 131072 si L = 131072

Timpul pentru convolutie prin FFT: 0.04238271713256836 secunde pentru N = 131072 si L = 131072

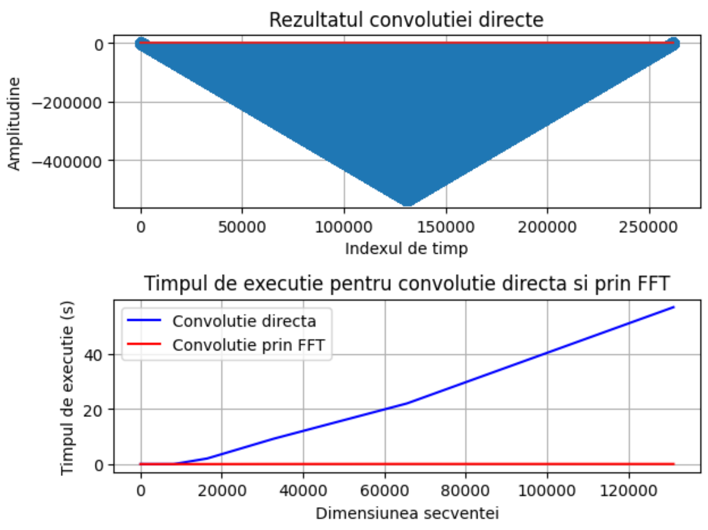

Figure 6.14: Results for length 131072

Timpul pentru convolutie directa: 135.88212823867798 secunde pentru N = 262144 si L = 262144

Timpul pentru convolutie prin FFT: 0.3767204284667969 secunde pentru N = 262144 si L = 262144

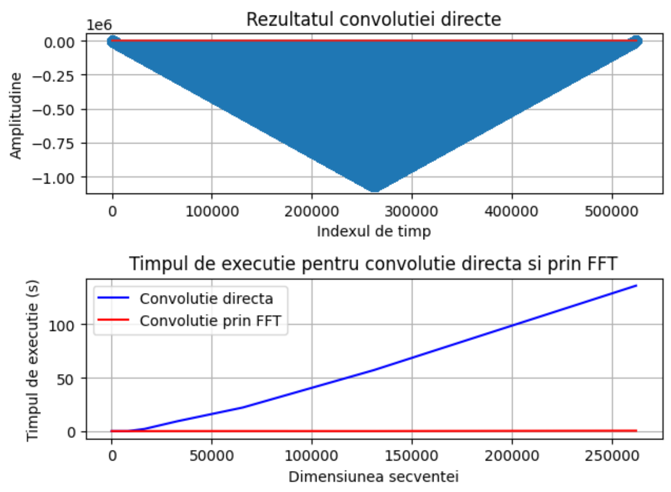

Figure 6.15: Results for length 262144

Timpul pentru convolutie directa: 205.45371532440186 secunde pentru N = 524288 si L = 524288

Timpul pentru convolutie prin FFT: 0.20438766479492188 secunde pentru N = 524288 si L = 524288

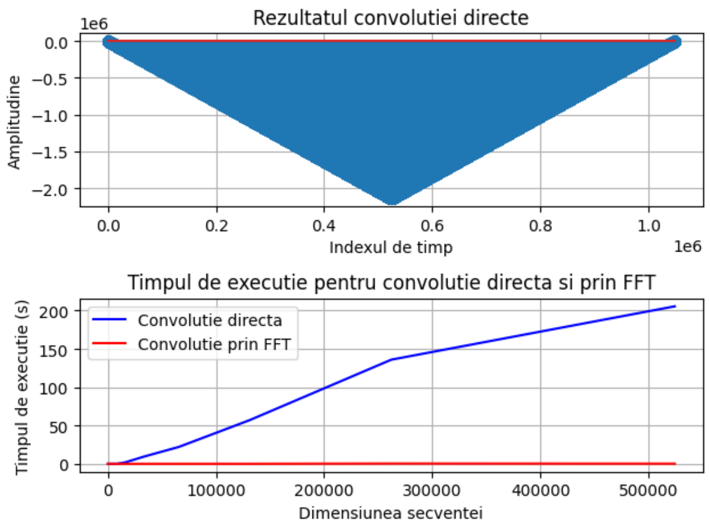

Figure 6.16: Result for length 524288

As a conclusion from the obtained results:

- For small values of the lengths of the signals, the time for both methods is pretty similar
- As values for lengths increase, the FFT method is much faster, not even half of second. But at the same length, the time for direct convolution increases a lot, almost to 4 minutes

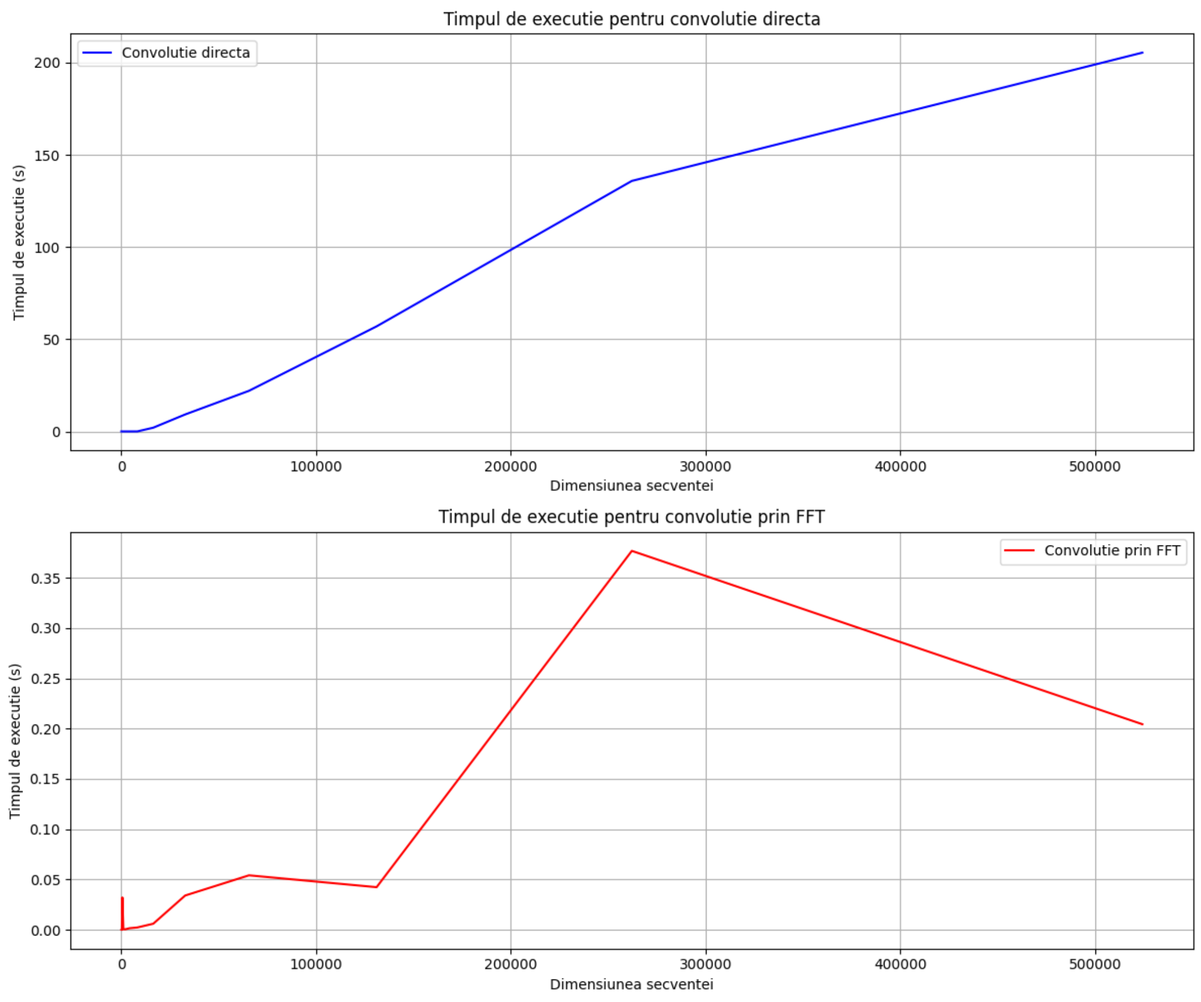

Figure 6.17: Comparison of time for direct convolution and FFT

In firs experiment, I took the values for length to be the same for all test cases. Next, I tried to mix the lengths, so they will be different. For example, N=16 and L=64 or N=2048 and L=131072. Below are some of the results for time, because in total there were 256 test cases:

- Timpul pentru convolutie directa: 2.5033950805664062e-05 secunde pentru N = 16 si L = 1024
- Timpul pentru convolutie prin FFT: 0.0012631416320800781 secunde pentru N = 16 si L = 1024
- Timpul pentru convolutie directa: 0.0003376007080078125 secunde pentru N = 32 si L = 16384
- Timpul pentru convolutie prin FFT: 0.0030112266540527344 secunde pentru N = 32 si L = 16384
- Timpul pentru convolutie directa: 0.002521991729736328 secunde pentru N = 128 si L = 65536
- Timpul pentru convolutie prin FFT: 0.07632708549499512 secunde pentru N = 128 si L = 65536

- Timpul pentru convolutie directa: 0.0010905265808105469 secunde pentru N = 1024 si L = 4096
- Timpul pentru convolutie prin FFT: 0.0024650096893310547 secunde pentru N = 1024 si L = 4096
- Timpul pentru convolutie directa: 0.004680156707763672 secunde pentru N = 8192 si L = 4096
- Timpul pentru convolutie prin FFT: 0.003312826156616211 secunde pentru N = 8192 si L = 4096

- Timpul pentru convolutie directa: 19.481027364730835 secunde pentru N = 32768 si L = 65536
- Timpul pentru convolutie prin FFT: 0.0406191349029541 secunde pentru N = 32768 si L = 65536
- Timpul pentru convolutie directa: 15.075796604156494 secunde pentru N = 65536 si L = 32768
- Timpul pentru convolutie prin FFT: 0.053049564361572266 secunde pentru N = 65536 si L = 32768

In conclusion, even for the cases when the lengths are different:

- For small lengths, either N or L, the time to calculate for both methods is small, not even 0.1 second
- As the values of lengths increase, the time to calculate using FFT remains small, not even 0.1 second, but the time for direct convolution increases significantly, in this case till to 15 seconds

8. The previously mentioned method performs **full convolution of two signals**.
   In some applications, it's necessary to perform convolution as the signal is being formed or measured. In such cases, **block convolution** is used, where the signal is divided into blocks and convolution is applied per block.

Suppose we have two signals — one is the input signal, and the second is the system's impulse response:

```
ini
CopiazăEditează
a = [1 4 2];
b = [1 2 3 4 5 4 3 3 2 2 1 1];
```

Determine the convolution of these signals (convolution length: 14).

```
#Task 8

def block_convolution(a, b, block_size):
    result = np.zeros(len(a) + len(b) - 1)
    for i in range(0, len(a), block_size):
        a_block = a[i:i + block_size]
        temp_result = np.convolve(a_block, b)
        result[i:i + len(temp_result)] += temp_result
    return result

# Define the sequences
a = np.array([1, 4, 2])
b = np.array([1, 2, 3, 4, 5, 4, 3, 3, 2, 2, 1, 1])

# Compute block convolution
conv_block = block_convolution(a, b, block_size=3)
print(f'Convolutie pe blocuri: {conv_block}')

# Plot the block convolution result
plt.figure(figsize=(10, 6))
plt.stem(np.arange(len(conv_block)), conv_block)
plt.xlabel('Indexul de timp')
plt.ylabel('Amplitudine')
plt.title('Rezultatul convolutiei pe blocuri')
plt.grid()
plt.show()
```

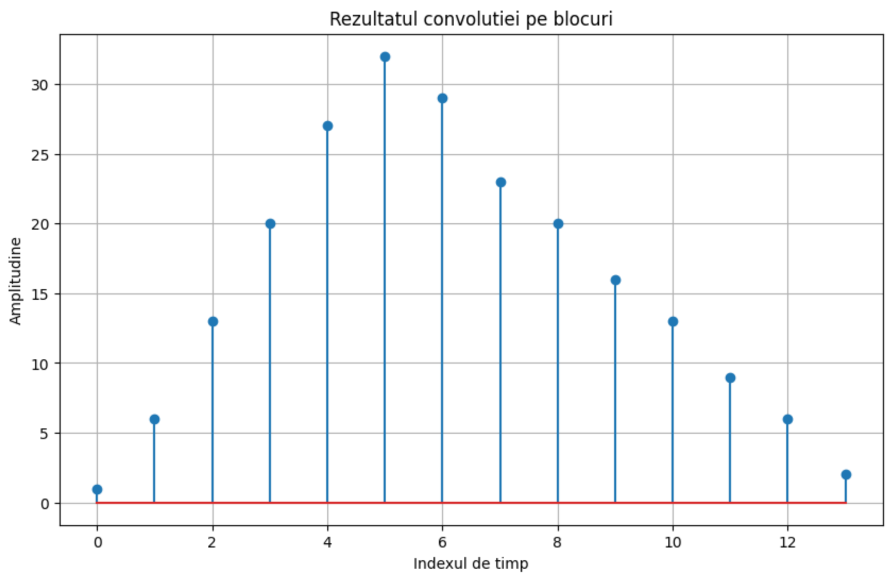

Figure 8: Result for task 8

The image represents the result of convolving two signals, performed using the block convolution approach. This method is suitable for applications where the signal is processed incrementally, as it is measured or formed.

The convolution result reflects how the input signal interacts with the system's impulse response over the specified length. The resulting amplitudes display a

structured pattern, starting with lower values, peaking, and then gradually decreasing, which is characteristic of the combined effects of the two signals.
This approach efficiently handles the convolution operation, ensuring accurate results while accommodating the constraints of real-time signal processing.

9. **Split the input signal into two blocks** of 6 samples each (a1 = a(1:6); and a2 = a(7:end);), and process these blocks separately — calculate the convolution of each block: c1 and c2 (convolution length: 8).

```
#Task 9

# Define the sequences
a = np.array([1, 2, 3, 4, 5, 4, 3, 3, 2, 2, 1, 1])
b = np.array([1, 4, 2])

# Split input signal into two blocks
a1 = a[:6]
a2 = a[6:]

# Compute convolutions separately
c1 = np.convolve(a1, b)
c2 = np.convolve(a2, b)

print(f'Bloc 1: {a1}')
print(f'Convolutie bloc 1: {c1}\n')

print(f'Bloc 2: {a2}')
print(f'Convolutie bloc 2: {c2}')

# Plot the convolutions
plt.figure(figsize=(12, 6))

# Subplot for the first block convolution
plt.subplot(2, 1, 1)
plt.stem(np.arange(len(c1)), c1)
plt.xlabel('Indexul de timp')
plt.ylabel('Amplitudine')
plt.title('Convolutia blocului 1')
plt.grid()

# Subplot for the second block convolution
plt.subplot(2, 1, 2)
plt.stem(np.arange(len(c2)), c2)
plt.xlabel('Indexul de timp')
plt.ylabel('Amplitudine')
```

```python
plt.title('Convolutia blocului 2')
plt.grid()

plt.tight_layout()
plt.show()
```

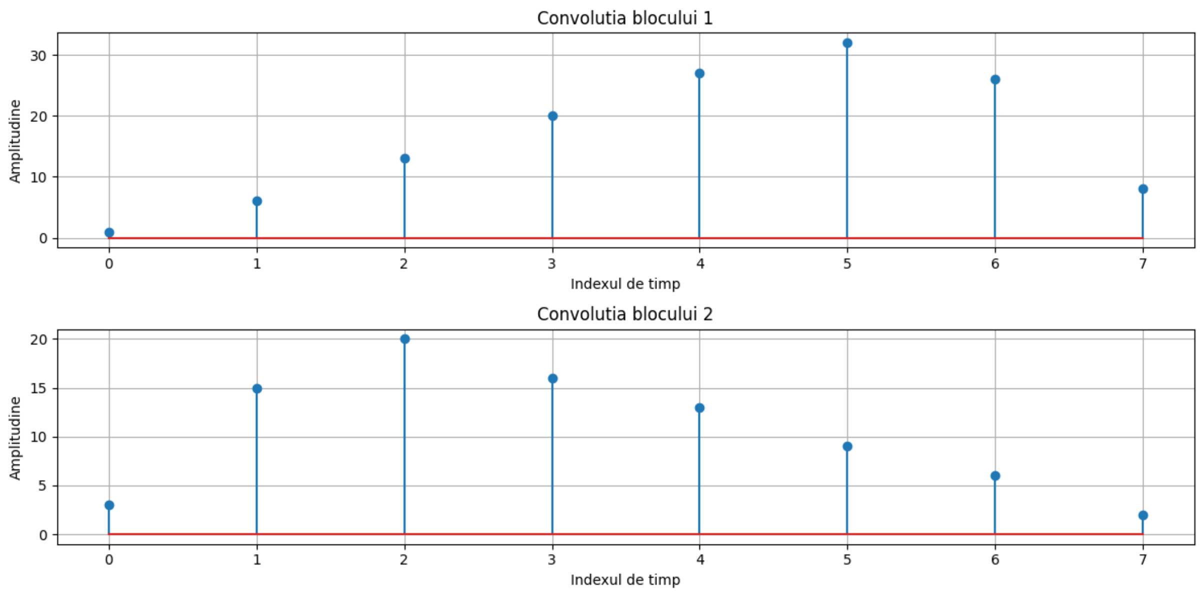

Figure 9: Results for task 9

Bloc 1: [1 2 3 4 5 4]

Convolutie bloc 1: [1 6 13 20 27 32 26 8]

Bloc 2: [3 3 2 2 1 1]

Convolutie bloc 2: [3 15 20 16 13 9 6 2]

10. **Determine the final convolution**:

    c_add = [c1(1:6), c1(7:8) + c2(1:2), c2(3:end)];

    Display the signal in discrete form using the stem(m, c_add) function.

```
# Compute final convolution

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

print(f'Convolutie finala: {c_add}')


# Display results

m = np.arange(len(c_add))

plt.stem(m, c_add)

plt.xlabel('Indexul de timp m')

plt.ylabel('Amplitudine')

plt.title('Convoluția finală')

plt.grid()


plt.show()
```

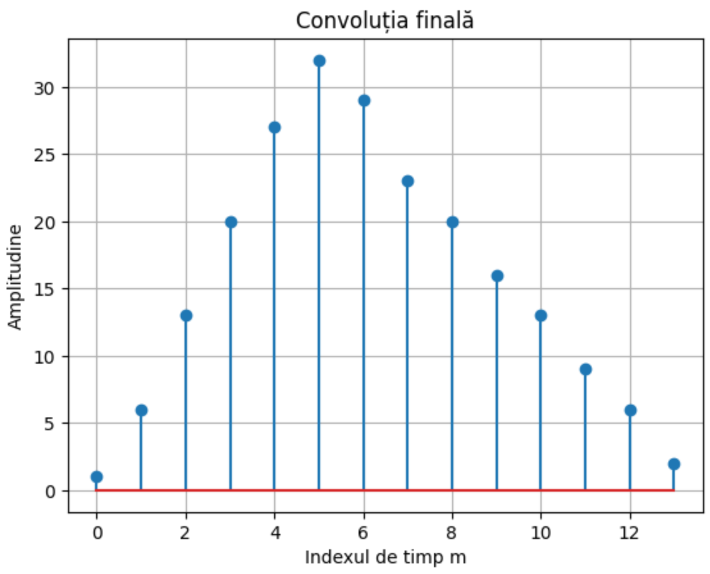

Figure 10: Result for task 10

```
Convolutie finala: [ 1  6 13 20 27 32 29 23 20 16 13  9  6  2]
```

11. **Open a block diagram modeling interface**: File → New → Model
(a similar interface was used in Lab #2). Open the individual block library
(In the Launch Pad window, go to **DSP Blockset → Block Library** or use the block diagram interface icon).
To build the block diagram below, find each block from the individual block library (by category, e.g., DSP Sources → DSP Constant, or search by name), and copy them into the open modeling interface.
Interconnect the found blocks, insert the values for signals 1 and 2 (DSP Constant and DSP Constant1 — other than those indicated here), and **run the block diagram simulation**.

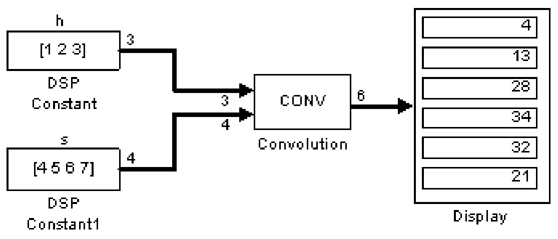

Figure 11.2: Block diagram for direct convolution

Because of not having access to Matlab and its functionalities, this task was done in Python

```
#Task 11

# Define the sequences
h = np.array([1, 2, 3])
s = np.array([4, 5, 6, 7])

d = len(h)
c = len(s)

n = np.arange(1, d + 1)
l = np.arange(1, c + 1)

# Compute convolution
conv_result = np.convolve(h, s)

print(f'Convolutia secventelor h si s: {conv_result}')
```

The result of this convolution is next:

Convolutia secventelor h si s: [4 13 28 34 32 21]

12. **Repeat step 6 using the block diagram** below (with the same signal values as in step 11 — DSP Constant and DSP Constant1 — but **add zeros to reach a number of values that is a power of 2**, a condition for using **FFT – Fast Fourier Transform**).

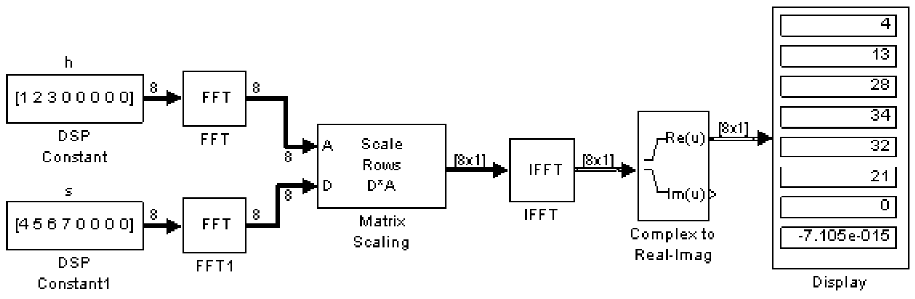

Figure 12: Block diagram for FFT convolution

As the previous task, it was done in Python

```
#Task 12

np.set_printoptions(suppress=True)  # Disable scientific notation

# Define input sequences
h = np.array([1, 2, 3, 0, 0, 0, 0, 0])  # Impulse response
s = np.array([4, 5, 6, 7, 0, 0, 0, 0])  # Input signal

# Compute FFT
H = np.fft.fft(h)
S = np.fft.fft(s)

# Perform element-wise multiplication in frequency domain
P = H * S

# Compute Inverse FFT (IFFT) to get the convolution result
conv_result = np.fft.ifft(P).real  # Taking real part

# Display the result
print("Convolution Result:", conv_result[0 : 6])
```

```
np.set_printoptions(suppress=False)  # Disable scientific notation
print(conv_result[7])
```

Results for the task are next:

```
Convolution Result: [ 4. 13. 28. 34. 32. 21. -3.552713678800501e-15]
```

Conclusions

In this laboratory work, we thoroughly explored the concept of convolution of two discrete-time sequences and its key properties. The theoretical and practical tasks helped in understanding the role of convolution in analyzing Linear Time-Invariant (LTI) systems and its equivalence with multiplication in the frequency domain.

Initially, convolution was performed using the direct method, providing a clear understanding of how two sequences interact over time. The same operation was then validated through the Fast Fourier Transform (FFT) approach, confirming the convolution theorem that states: *convolution in the time domain corresponds to multiplication in the frequency domain*.

Through practical tasks, it was demonstrated that for small-sized signals, both direct and FFT methods produce similar execution times. However, as the sequence length increases, the FFT-based method proves to be significantly faster and more efficient than the direct convolution approach. This efficiency becomes especially important in real-time signal processing applications involving large datasets.

Additionally, we explored **block convolution**, which is useful when signals are processed incrementally or in segments due to real-time constraints. The experiments confirmed that this method also provides correct and efficient results.

Finally, the use of block diagrams, although simulated in Python due to limited access to MATLAB tools, illustrated the theoretical understanding of system modeling and simulation using graphical representations.

Overall, the lab work successfully reinforced both the mathematical foundation and the practical implementation aspects of convolution, providing valuable insights into its applications in signal processing.