A blue and white logo

AI-generated content may be incorrect.

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:**

A black background with white symbols

AI-generated content may be incorrect.

1. **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.**
2. **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).**
3. **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)**
4. **Frequency Domain Interpretation:  
   Convolution in the time domain corresponds to multiplication in the frequency domain.**

**Y(f)=X(f)⋅H(f)**

1. **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.**
2. **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()

A graph of a number of numbers

AI-generated content may be incorrect.

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.

1. **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()

A graph with blue and red lines

AI-generated content may be incorrect.

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.

1. **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()

A graph of a diagram

AI-generated content may be incorrect.

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

1. **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()

A graph with lines and dots

AI-generated content may be incorrect.

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.

1. **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()

A graph of a graph

AI-generated content may be incorrect.

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 y1, 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.

Conclusions

In this individual work, we explored noise generation and filtering techniques using the Discrete-Time System, focusing on the M-point Moving Average System. The study involved generating white noise with different sampling intervals, analyzing its characteristics through histograms, and applying various filtering methods to enhance signal quality. These experiments provided a deeper understanding of how noise behaves in digital signal processing and how filtering techniques can be used to extract meaningful signals from noisy environments.

We began by generating white noise using a random function and visualizing it through time-domain plots. By modifying the sampling interval, we observed how different resolutions affect the representation of noise. We then represented the statistical properties of noise using histograms, which helped in understanding the uniform and Gaussian distributions of random processes.

To mitigate noise, we implemented digital filtering techniques. A second-order digital filter was applied to white noise, significantly reducing its randomness and producing a smoother signal. The impact of different filter parameters on signal clarity and smoothness was examined, demonstrating how filtering can enhance signal quality while preserving important features.

Further, we designed and applied a Moving Average Filter (MAF) to noisy signals. By adjusting the filter window size (M), we analyzed its effect on noise reduction. A smaller M provided less smoothing, preserving more details but allowing more noise to pass through, whereas a larger M resulted in a significantly smoother signal with reduced noise but also introduced a slight delay in signal response. These findings highlighted the trade-off between noise reduction and signal distortion in practical signal processing applications.

Additionally, we explored adaptive filtering techniques using the Least Mean Squares (LMS) and Recursive Least Squares (RLS) algorithms. These algorithms dynamically adjusted filter weights to minimize the error between the desired and output signals. Through various experiments, we found that RLS converged faster and provided more accurate filtering than LMS, making it more effective for real-time noise reduction. Increasing the filter length further improved performance for both algorithms, reducing noise and making the filtered signal more closely resemble the original.

Overall, this lab provided a comprehensive understanding of noise modeling, filtering techniques, and adaptive signal processing. The experiments demonstrated the importance of digital filtering in various applications, from audio processing to communication systems, where noise reduction is essential for improving signal clarity. By implementing and comparing different filtering methods, we gained valuable insights into how to effectively manage noise in real-world signal processing scenarios.