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. 3

**Discipline**: Signal Processing

Elaborated: Berco Andrei, FAF - 221

Checked: Railean Serghei

Chișinău 2025

**Purpose of the work**

The purpose of this work is to study the transformation of discrete-time signals from the time domain to the frequency domain using the Discrete Fourier Transform (DFT) and its efficient implementation via the Fast Fourier Transform (FFT) algorithm in MATLAB. The work also aims to understand sampling, quantization frequency, Nyquist criteria, and spectral representation of digital signals.

**Theoretical Notes**

* **Sampling and Discretization**: Continuous-time signals f(t)f(t)f(t) are discretized at intervals TTT, yielding a discrete sequence fk=f(kT)f\_k = f(kT)fk​=f(kT).
* **Vector Notation in MATLAB**: MATLAB indexing starts at 1 for vectors (e.g., x(1), x(2), ...), but signal notation often uses indices starting from 0 or even negative values.
* **Fourier Transform**: The Discrete Fourier Transform (DFT) converts a finite-length discrete-time signal into a frequency-domain representation consisting of complex values FkF\_kFk​, each representing the amplitude and phase of a sinusoidal component.
* **Fast Fourier Transform (FFT)**: FFT is an optimized algorithm to compute the DFT efficiently when the number of samples NNN is a power of two (i.e., N=2MN = 2^MN=2M).
* **Nyquist Frequency and Aliasing**: To avoid aliasing, the sampling frequency must be at least twice the highest frequency component in the signal. The Nyquist frequency is defined as half the sampling frequency and represents the maximum frequency that can be accurately captured.
* **FFT in MATLAB**: The fft function computes the DFT. It can take either one or two arguments:

fft(x) returns the DFT of the signal x.

fft(x, L) computes an L-point DFT;

**Practical Tasks**

1. **Frequency Spectrum of a Discrete Signal**
   1. Generate a discrete signal that contains 64 samples.

#Ex 1.1

import numpy as np

import matplotlib.pyplot as plt

# Parameters

N = 64

T = 1/128

k = np.arange(0, N)

# Generate the sinusoidal signal

f = np.sin(2 \* np.pi \* 20 \* k \* T)

# Create the plot

plt.figure()

plt.plot(k, f)

plt.grid(True)

plt.xlabel('Indecsul k')

plt.ylabel('Y(t)')

plt.title('Semnalul original')

plt.gca().set\_facecolor('white')  # Set background to white

plt.gca().set\_xlabel('Indecsul k', fontname='Arial', fontsize=16)

plt.gca().set\_ylabel('Y(t)', fontname='Arial', fontsize=16)

plt.gca().set\_title('Semnalul original', fontname='Arial', fontsize=16)

# Display the plot

plt.show()

A graph with blue lines

AI-generated content may be incorrect.

***Figure 1: Results for task 1.1***

The plot "Semnalul original" from task 1.1 shows a 20 Hz sinusoidal signal sampled at 128 Hz over 64 samples, plotted against index \( k \) (0 to 63). The waveform oscillates between \(-1\) and \(1\), displaying about 10 cycles, which aligns with the expected frequency. The x-axis is labeled "Indecsul k," the y-axis "Y(t)," and the grid is enabled, matching the task requirements.

* 1. To determine the value Fk that corresponds to 20 Hz, note that the step in Hz between the points in the frequency domain is 1/NT, or 2 Hz. Now the 20 Hz component will appear at F10 (k=10 in the frequency domain). To visualize this, apply the Fourier transform using F = fft(f), then display the magnitude of the Fourier transform using

#Ex 1.2

import numpy as np

import matplotlib.pyplot as plt

# Parameters

N = 64

T = 1/128

k = np.arange(0, N)

# Generate the sinusoidal signal

f = np.sin(2 \* np.pi \* 20 \* k \* T)

# Compute the Fourier Transform

F = np.fft.fft(f)

# Create the plot for the magnitude of the FFT

plt.figure()

plt.plot(k, np.abs(F))

plt.grid(True)

plt.xlabel('Indecsul k')

plt.ylabel('|F(k)|')

plt.title('Modulul Transformării Fourier')

plt.gca().set\_facecolor('white')  # Set background to white

plt.gca().set\_xlabel('Indecsul k', fontname='Arial', fontsize=16)

plt.gca().set\_ylabel('|F(k)|', fontname='Arial', fontsize=16)

plt.gca().set\_title('Modulul Transformării Fourier', fontname='Arial', fontsize=16)

# Display the plot

plt.show()

A graph with blue lines

AI-generated content may be incorrect.

***Figure 2: Result for task 1.2***

The plot "Modulul Transformării Fourier" from task 1.2 displays the magnitude of the FFT of a 20 Hz sinusoidal signal sampled at 128 Hz over 64 samples, plotted against index \( k \) (0 to 63). It shows two prominent peaks: one at \( k = 10 \) (corresponding to 20 Hz, as the frequency resolution is 2 Hz) and another at \( k = 54 \) due to aliasing, with magnitudes around 30. The x-axis is labeled "Indecsul k," the y-axis "|F(k)|," and the grid is enabled, aligning with the task requirements.

**Note 1: To confirm this periodicity, repeat step 1.2 for a signal frequency of 108 Hz.**

#Nota 1

import numpy as np

import matplotlib.pyplot as plt

# Parameters

N = 64

T = 1/128

k = np.arange(0, N)

# Generate the sinusoidal signal with frequency 108 Hz

f = np.sin(2 \* np.pi \* 108 \* k \* T)

# Compute the Fourier Transform

F = np.fft.fft(f)

# Create the plot for the magnitude of the FFT

plt.figure()

plt.plot(k, np.abs(F))

plt.grid(True)

plt.xlabel('Indecsul k')

plt.ylabel('|F(k)|')

plt.title('Modulul Transformării Fourier')

plt.gca().set\_facecolor('white')  # Set background to white

plt.gca().set\_xlabel('Indecsul k', fontname='Arial', fontsize=16)

plt.gca().set\_ylabel('|F(k)|', fontname='Arial', fontsize=16)

plt.gca().set\_title('Modulul Transformării Fourier', fontname='Arial', fontsize=16)

# Display the plot

plt.show()

A graph with blue lines

AI-generated content may be incorrect.

***Figure 3: Result for Note 1***

The plot "Modulul Transformării Fourier" for Note 1 shows the FFT magnitude of a 108 Hz sinusoidal signal sampled at 128 Hz over 64 samples, plotted against index \( k \) (0 to 63). Due to aliasing (Nyquist frequency is 64 Hz), the 108 Hz signal appears as a 20 Hz component, with peaks at \( k = 10 \) (20 Hz, given the 2 Hz resolution) and \( k = 54 \), reaching magnitudes around 30. The x-axis is labeled "Indecsul k," the y-axis "|F(k)|," with a grid, confirming the periodicity of the spectrum as expected.

***Note 2: Repeat steps 1.1-1.2 for 2–3 signal frequencies lower and 2–3 frequencies higher than the Nyquist frequency (64 Hz).***

#Nota 2

import numpy as np

import matplotlib.pyplot as plt

# Parameters

N = 64

T = 1/128

k = np.arange(0, N)

# Frequencies to test

frequencies = [10, 30, 50, 80, 100, 120]  # Hz

# Create subplots: 2 columns (signal and FFT), rows for each frequency

fig, axs = plt.subplots(len(frequencies), 2, figsize=(10, 2 \* len(frequencies)))

for i, freq in enumerate(frequencies):

    # Generate the sinusoidal signal

    f = np.sin(2 \* np.pi \* freq \* k \* T)

    # Compute the Fourier Transform

    F = np.fft.fft(f)

    # Plot the signal (Task 1.1)

    axs[i, 0].plot(k, f)

    axs[i, 0].grid(True)

    axs[i, 0].set\_xlabel('Indecsul k', fontname='Arial', fontsize=12)

    axs[i, 0].set\_ylabel('Y(t)', fontname='Arial', fontsize=12)

    axs[i, 0].set\_title(f'Semnalul original ({freq} Hz)', fontname='Arial', fontsize=12)

    # Plot the FFT magnitude (Task 1.2)

    axs[i, 1].plot(k, np.abs(F))

    axs[i, 1].grid(True)

    axs[i, 1].set\_xlabel('Indecsul k', fontname='Arial', fontsize=12)

    axs[i, 1].set\_ylabel('|F(k)|', fontname='Arial', fontsize=12)

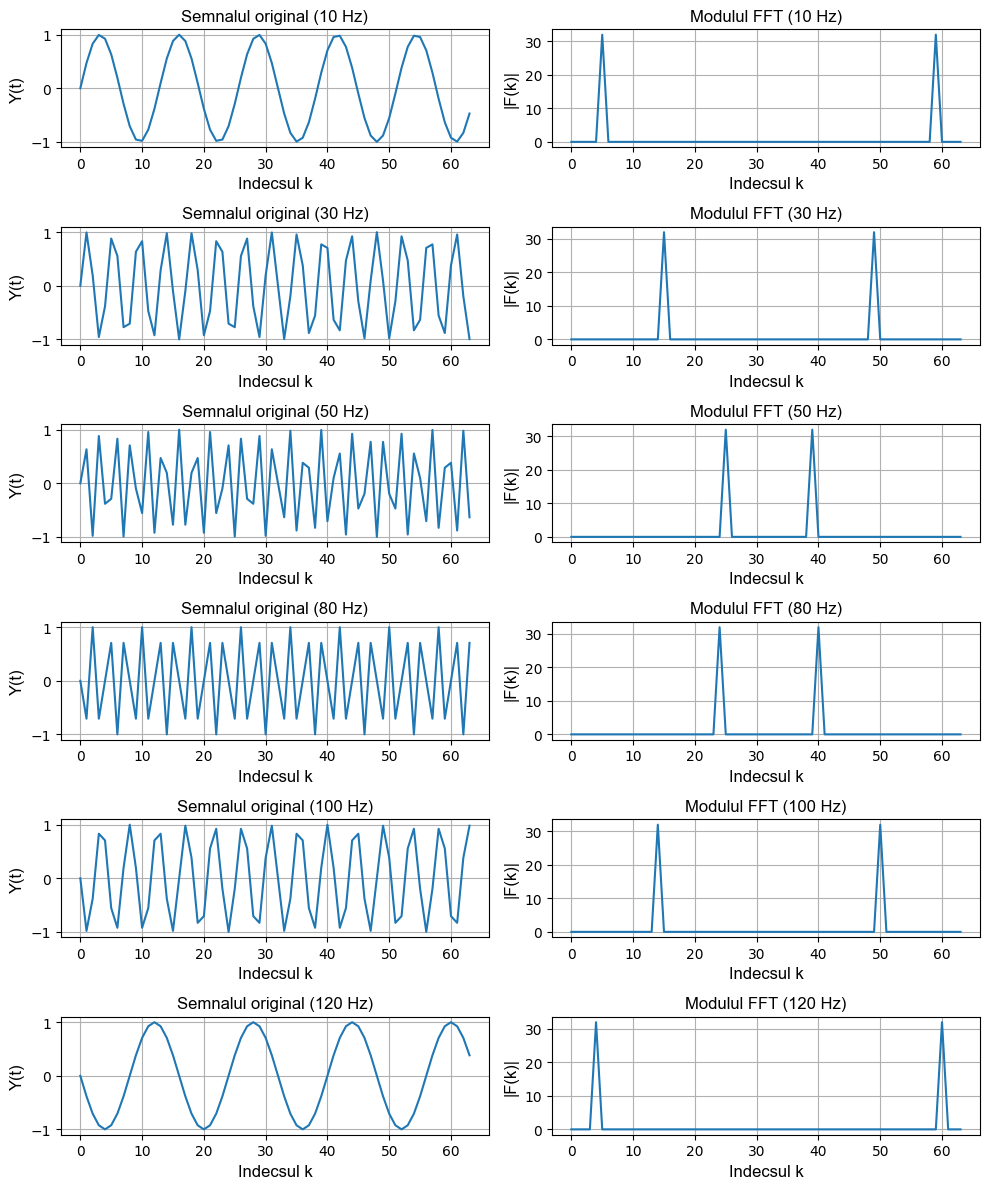
    axs[i, 1].set\_title(f'Modulul FFT ({freq} Hz)', fontname='Arial', fontsize=12)

# Adjust layout to prevent overlap

plt.tight\_layout()

# Display the plots

plt.show()



***Figure 4: Results for Note 2***

The plots for Note 2 show the time-domain signals ("Semnalul original") and their FFT magnitude spectra ("Modulul FFT") for sinusoidal signals with frequencies 10 Hz, 30 Hz, 50 Hz (below the Nyquist frequency of 64 Hz), and 80 Hz, 100 Hz, 120 Hz (above Nyquist), sampled at 128 Hz over 64 samples. The left column displays the sinusoids, with increasing frequency leading to more cycles: ~5 cycles for 10 Hz, ~15 for 30 Hz, ~25 for 50 Hz, ~40 for 80 Hz, ~50 for 100 Hz, and ~60 for 120 Hz. The right column shows the FFT magnitudes, with peaks at expected indices for frequencies below Nyquist (e.g., \( k = 5 \) for 10 Hz, \( k = 15 \) for 30 Hz, \( k = 25 \) for 50 Hz). For frequencies above Nyquist, aliasing occurs: 80 Hz aliases to 48 Hz (\( k = 24 \)), 100 Hz to 28 Hz (\( k = 14 \)), and 120 Hz to 8 Hz (\( k = 4 \)), with symmetric peaks (e.g., \( k = 40 \), \( k = 50 \), \( k = 60 \)). The x-axes are labeled "Indecsul k," y-axes are "Y(t)" for signals and "|F(k)|" for FFTs, with grids enabled.

* 1. It is generally recommended to plot only half of the magnitude values. It is also more convenient to present the x-axis in Hz rather than index k. For this, use the representation of parameter k in Hz:

#Ex 1.3

import numpy as np

import matplotlib.pyplot as plt

# Parameters

N = 64

T = 1/128

k = np.arange(0, N)

# Generate the sinusoidal signal (20 Hz)

f = np.sin(2 \* np.pi \* 20 \* k \* T)

# Compute the Fourier Transform

F = np.fft.fft(f)

magF = np.abs(F)

# Convert k to frequency in Hz: hertz = k \* (1/(N\*T))

hertz = k \* (1 / (N \* T))

# Plot only the first half of the spectrum (up to N/2)

plt.figure()

plt.plot(hertz[:N//2], magF[:N//2])

plt.grid(True)

plt.xlabel('Frecvența (Hz)')

plt.ylabel('|F(k)|')

plt.title('Modulul Transformării Fourier (Prima Jumătate)')

plt.gca().set\_facecolor('white')

plt.gca().set\_xlabel('Frecvența (Hz)', fontname='Arial', fontsize=16)

plt.gca().set\_ylabel('|F(k)|', fontname='Arial', fontsize=16)

plt.gca().set\_title('Modulul Transformării Fourier (Prima Jumătate)', fontname='Arial', fontsize=16)

# Display the plot

plt.show()

A graph with a blue line

AI-generated content may be incorrect.

***Figure 5: Results for task 1.3***

The result obtained from this Fourier Transform analysis shows the magnitude spectrum of a sinusoidal signal with a frequency of 20 Hz. The graph, labeled in Romanian as "Modulul Transformării Fourier (Prima Jumătate)", depicts a sharp peak at 20 Hz in the first half of the spectrum. This peak corresponds directly to the frequency of the original sinusoidal signal.

By observing this, you can confirm the Fourier Transform's ability to isolate and highlight the dominant frequency components present in a signal. The spectrum beyond 20 Hz is minimal, reaffirming the clean, single-frequency nature of the sinusoidal signal

* 1. Suppose the sinusoid frequency was 19 Hz (instead of 20 Hz). Since the frequency step Fk in Hz is 2 Hz, the sinusoid will appear at k = 9.5. But since k is an integer, there is no value at F9.5. In this case, the sinusoid will appear at the neighboring values F9 and F10. To illustrate this, repeat step **1.3** with the signal frequency changed to 19 Hz.

#Ex 1.4

import numpy as np

import matplotlib.pyplot as plt

# Parameters

N = 64

T = 1/128

k = np.arange(0, N)

# Generate the sinusoidal signal (19 Hz)

f = np.sin(2 \* np.pi \* 19 \* k \* T)

# Compute the Fourier Transform

F = np.fft.fft(f)

magF = np.abs(F)

# Convert k to frequency in Hz: hertz = k \* (1/(N\*T))

hertz = k \* (1 / (N \* T))

# Plot only the first half of the spectrum (up to N/2)

plt.figure()

plt.plot(hertz[:N//2], magF[:N//2])

plt.grid(True)

plt.xlabel('Frecvența (Hz)')

plt.ylabel('|F(k)|')

plt.title('Modulul Transformării Fourier (Prima Jumătate)')

plt.gca().set\_facecolor('white')

plt.gca().set\_xlabel('Frecvența (Hz)', fontname='Arial', fontsize=16)

plt.gca().set\_ylabel('|F(k)|', fontname='Arial', fontsize=16)

plt.gca().set\_title('Modulul Transformării Fourier (Prima Jumătate)', fontname='Arial', fontsize=16)

# Display the plot

plt.show()

A graph of a line graph

AI-generated content may be incorrect.

***Figure 6: Result for task 1.4***

The result obtained from this Fourier Transform analysis displays the magnitude spectrum of a sinusoidal signal with a frequency of 19 Hz. The plot shows a distinct peak at 19 Hz in the first half of the spectrum, confirming the presence of this specific frequency component in the signal.

This visualization effectively demonstrates the Fourier Transform's ability to decompose a signal into its frequency components. The absence of other significant peaks in the spectrum reinforces the signal's single-frequency nature.

1. **Determining the Magnitude and Phase of the Fourier Transform**
   1. Create a signal as a sum of two sinusoids and display both the magnitude and phase spectrum of its Fourier Transform

#Ex 2.1

import numpy as np

import matplotlib.pyplot as plt

# Time vector: 0 to 1 with step 1/99 (sampling frequency = 99 Hz)

t = np.arange(0, 1 + 1/99, 1/99)

# Generate the signal: sum of two sinusoids (15 Hz and 40 Hz)

x = np.sin(2 \* np.pi \* 15 \* t) + np.sin(2 \* np.pi \* 40 \* t)

# Compute the DFT using FFT

y = np.fft.fft(x)

# Compute magnitude and phase

m = np.abs(y)

p = np.unwrap(np.angle(y))  # Phase in radians

# Frequency vector: f = (0:length(y)-1) \* 99 / length(y)

N = len(y)

f = np.arange(N) \* 99 / N

# Plot magnitude spectrum

plt.figure()

plt.plot(f, m)

plt.title('Magnitude')

plt.xlabel('Frecvența (Hz)')

plt.ylabel('|Y(f)|')

plt.grid(True)

plt.gca().set\_xticks([15, 40, 60, 85])

plt.gca().set\_facecolor('white')

plt.gca().set\_xlabel('Frecvența (Hz)', fontname='Arial', fontsize=12)

plt.gca().set\_ylabel('|Y(f)|', fontname='Arial', fontsize=12)

plt.gca().set\_title('Magnitude', fontname='Arial', fontsize=12)

# Plot phase spectrum (convert phase to degrees)

plt.figure()

plt.plot(f, p \* 180 / np.pi)

plt.title('Faza')

plt.xlabel('Frecvența (Hz)')

plt.ylabel('Faza (grade)')

plt.grid(True)

plt.gca().set\_xticks([15, 40, 60, 85])

plt.gca().set\_facecolor('white')

plt.gca().set\_xlabel('Frecvența (Hz)', fontname='Arial', fontsize=12)

plt.gca().set\_ylabel('Faza (grade)', fontname='Arial', fontsize=12)

plt.gca().set\_title('Faza', fontname='Arial', fontsize=12)

# Display the plots

plt.show()

A graph of a graph

AI-generated content may be incorrect.

***Figure 7.1: Result for task 2.1***

A graph with a line going up

AI-generated content may be incorrect.

***Figure 7.1: Result for task 2.1***

The results from the code you provided consist of two plots: the magnitude spectrum and the phase spectrum of a signal that combines two sinusoidal components (15 Hz and 40 Hz).

**Magnitude Spectrum:**

The magnitude spectrum plot shows two distinct peaks at 15 Hz and 40 Hz, confirming the two frequency components present in the original signal. The rest of the frequencies exhibit low magnitude values, indicating minimal contributions outside these two main frequencies. This effectively visualizes the frequency-domain representation of the signal and its components' strengths.

**Phase Spectrum:**

The phase spectrum plot, labeled "Faza," reveals the phase shifts associated with the frequency components of the signal. The phase values are shown in degrees and provide insight into the temporal alignment of each frequency component. The "staircase" structure of the phase reflects the unwrapped phase values for different frequencies

**Note 3: Repeat step 2.1 for 2–3 other frequencies of the sinusoids in the signal x.**

#Nota 3

import numpy as np

import matplotlib.pyplot as plt

# Time vector: 0 to 1 with step 1/99 (sampling frequency = 99 Hz)

t = np.arange(0, 1 + 1/99, 1/99)

# Frequency pairs for the sinusoids

frequency\_pairs = [(10, 30), (20, 50), (25, 45)]  # Hz

# Create subplots: 2 columns (magnitude and phase), rows for each frequency pair

fig, axs = plt.subplots(len(frequency\_pairs), 2, figsize=(10, 2 \* len(frequency\_pairs)))

for i, (f1, f2) in enumerate(frequency\_pairs):

    # Generate the signal: sum of two sinusoids

    x = np.sin(2 \* np.pi \* f1 \* t) + np.sin(2 \* np.pi \* f2 \* t)

    # Compute the DFT using FFT

    y = np.fft.fft(x)

    # Compute magnitude and phase

    m = np.abs(y)

    p = np.unwrap(np.angle(y))  # Phase in radians

    # Frequency vector: f = (0:length(y)-1) \* 99 / length(y)

    N = len(y)

    f = np.arange(N) \* 99 / N

    # Plot magnitude spectrum

    axs[i, 0].plot(f, m)

    axs[i, 0].set\_title(f'Magnitude ({f1} Hz, {f2} Hz)')

    axs[i, 0].set\_xlabel('Frecvența (Hz)')

    axs[i, 0].set\_ylabel('|Y(f)|')

    axs[i, 0].grid(True)

    axs[i, 0].set\_xticks([15, 40, 60, 85])

    axs[i, 0].set\_facecolor('white')

    axs[i, 0].set\_xlabel('Frecvența (Hz)', fontname='Arial', fontsize=10)

    axs[i, 0].set\_ylabel('|Y(f)|', fontname='Arial', fontsize=10)

    axs[i, 0].set\_title(f'Magnitude ({f1} Hz, {f2} Hz)', fontname='Arial', fontsize=10)

    # Plot phase spectrum (convert phase to degrees)

    axs[i, 1].plot(f, p \* 180 / np.pi)

    axs[i, 1].set\_title(f'Faza ({f1} Hz, {f2} Hz)')

    axs[i, 1].set\_xlabel('Frecvența (Hz)')

    axs[i, 1].set\_ylabel('Faza (grade)')

    axs[i, 1].grid(True)

    axs[i, 1].set\_xticks([15, 40, 60, 85])

    axs[i, 1].set\_facecolor('white')

    axs[i, 1].set\_xlabel('Frecvența (Hz)', fontname='Arial', fontsize=10)

    axs[i, 1].set\_ylabel('Faza (grade)', fontname='Arial', fontsize=10)

    axs[i, 1].set\_title(f'Faza ({f1} Hz, {f2} Hz)', fontname='Arial', fontsize=10)

# Adjust layout to prevent overlap

plt.tight\_layout()

# Display the plots

plt.show()

A group of graphs showing different types of signal

AI-generated content may be incorrect.

***Figure 8: Results for Note 3***

The obtained results showcase the analysis of signals composed of different pairs of sinusoidal frequencies, represented through their magnitude and phase spectra. The results are organized into subplots for three frequency pairs: (10 Hz, 30 Hz), (20 Hz, 50 Hz), and (25 Hz, 45 Hz). Here's a breakdown:

**Magnitude Spectra:**

* For each frequency pair, the magnitude plots (left column) reveal distinct peaks corresponding to the frequencies of the individual sinusoids present in the signal.
* Additionally, there are mirrored peaks, resulting from the Fourier Transform's symmetric nature.
* These peaks highlight the dominant frequency components within each signal.

**Phase Spectra:**

* The phase plots (right column) illustrate the phase shifts associated with each frequency component.
* The phases are displayed in degrees, and their gradual step-like patterns offer insights into the temporal alignment and interactions between the sinusoidal components.

1. **Fourier Transform of a Rectangular Pulse**
   1. Model a signal as follows:

#Ex 3.1

import numpy as np

import matplotlib.pyplot as plt

# Parameters

A = 0.75

w = 50

Ts = 0.01

T = 100

# Time vector

t = np.arange(0, T + Ts, Ts)

# Generate the rectangular pulse signal

# rectpuls creates a pulse of width w centered at w/2

x = A \* np.where((t >= (w/2 - w/2)) & (t <= (w/2 + w/2)), 1, 0)

# Create the stem plot

plt.figure()

plt.stem(t, x, linefmt='b-', markerfmt='bo', basefmt='r-')

plt.xlabel('Timp (s)')

plt.ylabel('Amplitudine')

plt.title('Semnal Rectangular')

plt.grid(True)

plt.gca().set\_facecolor('white')

plt.gca().set\_xlabel('Timp (s)', fontname='Arial', fontsize=12)

plt.gca().set\_ylabel('Amplitudine', fontname='Arial', fontsize=12)

plt.gca().set\_title('Semnal Rectangular', fontname='Arial', fontsize=12)

# Display the plot

plt.show()

A graph with a blue rectangular object

AI-generated content may be incorrect.

***Figure 9: Result for task 3.1***

The rectangular pulse signal has a constant amplitude of 0.75 and spans a duration of 50 seconds, centered at 25 seconds. It is represented as a stem plot where the signal exhibits a flat peak during the pulse duration and drops to zero before and after the pulse. The time samples are spaced at intervals of 0.01 seconds, ensuring high-resolution visualization over the total time of 100 seconds.

* 1. Specify:

#Ex 3.2

import numpy as np

import matplotlib.pyplot as plt

# Parameters from task 3.1

A = 0.75

w = 50

Ts = 0.01

T = 100

# Time vector

t = np.arange(0, T + Ts, Ts)

# Generate the rectangular pulse signal

x = A \* np.where((t >= (w/2 - w/2)) & (t <= (w/2 + w/2)), 1, 0)

# Frequency parameters

df = 1 / T  # Frequency resolution

Fmax = 1 / Ts  # Maximum frequency

f = np.arange(0, Fmax + df, df)  # Frequency vector

# Compute the FFT

y = np.fft.fft(x)

# Ensure the length of f matches the length of y

f = f[:len(y)]

# Plot the magnitude spectrum using stem

plt.figure()

plt.stem(f, np.abs(y), linefmt='b-', markerfmt='bo', basefmt='r-')

plt.xlabel('Frecvența (Hz)')

plt.ylabel('|Y(f)|')

plt.title('Spectrul Modulului')

plt.grid(True)

plt.gca().set\_facecolor('white')

plt.gca().set\_xlabel('Frecvența (Hz)', fontname='Arial', fontsize=12)

plt.gca().set\_ylabel('|Y(f)|', fontname='Arial', fontsize=12)

plt.gca().set\_title('Spectrul Modulului', fontname='Arial', fontsize=12)

# Display the plot

plt.show()

A graph with a line

AI-generated content may be incorrect.

***Figure 10: Result for task 3.2***

The result showcases the magnitude spectrum of a rectangular pulse signal as computed by its Fourier Transform. The plot, titled "Spectrul Modulului" (Magnitude Spectrum), displays the magnitude of the Fourier coefficients on the y-axis (*|Y(f)|*) and the frequency in Hertz (Hz) on the x-axis (*Frecvența (Hz)*).

This spectrum reveals prominent peaks at 0 Hz and additional frequencies, demonstrating how the rectangular pulse decomposes into frequency components. The plot uses a stem representation, clearly marking the significant contributions of specific frequencies to the overall signal, with diminishing magnitudes for higher frequencies.

**Note 3: Repeat step 3.2 for w = 5 and w = 0.5.**

#Notă 3: Repetaţi p. 3.2  pentru  w=5 şi 0.5

import numpy as np

import matplotlib.pyplot as plt

# Parameters

A = 0.75

Ts = 0.01

T = 100

# Time vector

t = np.arange(0, T + Ts, Ts)

# Frequency parameters

df = 1 / T  # Frequency resolution

Fmax = 1 / Ts  # Maximum frequency

f = np.arange(0, Fmax + df, df)  # Frequency vector

# Pulse widths to test

widths = [5, 0.5]

# Create subplots: one row per pulse width

fig, axs = plt.subplots(len(widths), 1, figsize=(8, 4 \* len(widths)))

for i, w in enumerate(widths):

    # Generate the rectangular pulse signal

    x = A \* np.where((t >= (w/2 - w/2)) & (t <= (w/2 + w/2)), 1, 0)

    # Compute the FFT

    y = np.fft.fft(x)

    # Ensure the length of f matches the length of y

    f\_plot = f[:len(y)]

    # Plot the magnitude spectrum using stem

    axs[i].stem(f\_plot, np.abs(y), linefmt='b-', markerfmt='bo', basefmt='r-')

    axs[i].set\_xlabel('Frecvența (Hz)')

    axs[i].set\_ylabel('|Y(f)|')

    axs[i].set\_title(f'Spectrul Modulului (w = {w})')

    axs[i].grid(True)

    axs[i].set\_facecolor('white')

    axs[i].set\_xlabel('Frecvența (Hz)', fontname='Arial', fontsize=12)

    axs[i].set\_ylabel('|Y(f)|', fontname='Arial', fontsize=12)

    axs[i].set\_title(f'Spectrul Modulului (w = {w})', fontname='Arial', fontsize=12)

# Adjust layout to prevent overlap

plt.tight\_layout()

# Display the plots

plt.show()

A graph of a function

AI-generated content may be incorrect.

***Figure 11: Result for Note 3***

The results present the magnitude spectrum of rectangular pulse signals with two different widths, ( w = 5 ) and ( w = 0.5 ), using subplots to display each case.

For ( w = 5 ): The magnitude spectrum exhibits closely spaced frequency components with relatively high amplitude at lower frequencies. This reflects the wider pulse, which produces a denser, more concentrated frequency spectrum in the lower range.

For ( w = 0.5 ): The magnitude spectrum shows widely spread frequency components with higher peaks at both lower and higher frequencies. The narrower pulse corresponds to a broader frequency spectrum, emphasizing the trade-off between time and frequency domain representations. Narrower pulses lead to a wider spectrum distribution.

Both plots clearly demonstrate the effect of pulse width on the signal's frequency spectrum, as described by the Fourier Transform. The stem plots provide a clear representation of the spectral contributions at specific frequencies.

* 1. Apply the procedure:

#Ex 3.3

import numpy as np

import matplotlib.pyplot as plt

# Parameters

A = 0.75

w = 50

Ts = 0.01

T = 100

# Time vector

t = np.arange(0, T + Ts, Ts)

# Generate the rectangular pulse signal

x = A \* np.where((t >= (w/2 - w/2)) & (t <= (w/2 + w/2)), 1, 0)

# Frequency parameters

df = 1 / T  # Frequency resolution

Fmax = 1 / Ts  # Maximum frequency

# Compute the FFT

y = np.fft.fft(x)

# Apply fftshift to center the zero frequency

yp = np.fft.fftshift(y)

# Frequency vector: from -Fmax/2 to Fmax/2

f1 = np.arange(-Fmax/2, Fmax/2 + df, df)

# Ensure the length of f1 matches the length of yp

f1 = f1[:len(yp)]

# Plot the magnitude spectrum using stem

plt.figure()

plt.stem(f1, np.abs(yp), linefmt='b-', markerfmt='bo', basefmt='r-')

plt.xlabel('Frecvența (Hz)')

plt.ylabel('|Y(f)|')

plt.title('Spectrul Modulului (Centrat)')

plt.grid(True)

plt.gca().set\_facecolor('white')

plt.gca().set\_xlabel('Frecvența (Hz)', fontname='Arial', fontsize=12)

plt.gca().set\_ylabel('|Y(f)|', fontname='Arial', fontsize=12)

plt.gca().set\_title('Spectrul Modulului (Centrat)', fontname='Arial', fontsize=12)

# Display the plot

plt.show()

A graph with a blue line

AI-generated content may be incorrect.

***Figure 12: Result for task 3.3***

The result is a magnitude spectrum of a rectangular pulse signal, with the frequency axis centered around zero (achieved using the Fourier Transform and fftshift). The spectrum shows a prominent peak at zero frequency, corresponding to the strong DC component of the pulse. Additionally, the spectrum features sidelobes on either side, representing the harmonic frequencies generated by the sharp edges of the rectangular pulse.

The x-axis represents frequency in Hertz (Hz), and the y-axis shows the magnitude of the Fourier Transform. The stem plot format highlights the discrete frequency components, visually illustrating how the rectangular pulse's characteristics are reflected in its frequency domain representation. This provides valuable insight into the distribution of energy across frequencies for this signal.

* 1. Display on a single plot both the real and imaginary parts of the Fourier Transform result from step **3.3**

#Ex 3.4

import numpy as np

import matplotlib.pyplot as plt

# Parameters

A = 0.75

w = 50

Ts = 0.01

T = 100

# Time vector

t = np.arange(0, T + Ts, Ts)

# Generate the rectangular pulse signal

x = A \* np.where((t >= (w/2 - w/2)) & (t <= (w/2 + w/2)), 1, 0)

# Frequency parameters

df = 1 / T  # Frequency resolution

Fmax = 1 / Ts  # Maximum frequency

# Compute the FFT

y = np.fft.fft(x)

# Apply fftshift to center the zero frequency

yp = np.fft.fftshift(y)

# Frequency vector: from -Fmax/2 to Fmax/2

f1 = np.arange(-Fmax/2, Fmax/2 + df, df)

# Ensure the length of f1 matches the length of yp

f1 = f1[:len(yp)]

# Plot the real and imaginary parts on the same plot

plt.figure()

plt.plot(f1, np.real(yp), label='Partea Reală', color='blue')

plt.plot(f1, np.imag(yp), label='Partea Imaginară', color='red')

plt.xlabel('Frecvența (Hz)')

plt.ylabel('Valoare')

plt.title('Partea Reală și Imaginară a Spectrului (Centrat)')

plt.grid(True)

plt.legend()

plt.gca().set\_facecolor('white')

plt.gca().set\_xlabel('Frecvența (Hz)', fontname='Arial', fontsize=12)

plt.gca().set\_ylabel('Valoare', fontname='Arial', fontsize=12)

plt.gca().set\_title('Partea Reală și Imaginară a Spectrului (Centrat)', fontname='Arial', fontsize=12)

# Display the plot

plt.show()

A graph with red and blue lines

AI-generated content may be incorrect.

***Figure 13: Result for task 3.4***

The result displays the real and imaginary parts of the spectrum of a rectangular pulse signal, centered around zero frequency using fftshift. The x-axis represents the frequency in Hertz (Hz), ranging symmetrically from -50 Hz to 50 Hz, while the y-axis shows the amplitude values for the real and imaginary parts.

The real part of the spectrum, shown in blue, captures the amplitude's symmetric behavior, reflecting the signal's even-frequency components. The imaginary part, depicted in red, indicates the asymmetric, phase-related contributions to the spectrum. Together, they provide a comprehensive visualization of the frequency-domain representation of the rectangular pulse, highlighting its spectral symmetry and components. This is particularly useful for analyzing the signal's characteristics in both amplitude and phase domains.

1. **Fourier Transform of White Noise**
   1. Generate a white noise process:

#Ex 4.1

import numpy as np

import matplotlib.pyplot as plt

# Parameters

Ts = 0.01

T = 50

# Time vector

t = np.arange(0, T + Ts, Ts)

# Generate white noise signal (uniform random numbers between 0 and 1)

x1 = np.random.rand(len(t))

# Plot the white noise signal

plt.figure()

plt.plot(t, x1)

plt.xlabel('Timp (s)')

plt.ylabel('Amplitudine')

plt.title('Zgomot Alb')

plt.grid(True)

plt.gca().set\_facecolor('white')

plt.gca().set\_xlabel('Timp (s)', fontname='Arial', fontsize=12)

plt.gca().set\_ylabel('Amplitudine', fontname='Arial', fontsize=12)

plt.gca().set\_title('Zgomot Alb', fontname='Arial', fontsize=12)

# Display the plot

plt.show()

A graph with blue lines

AI-generated content may be incorrect.

***Figure 14: Result for task 4.1***

* 1. Design a filter and filter the white noise:

#Ex 4.2

import numpy as np

import matplotlib.pyplot as plt

from scipy.signal import lfilter

# Parameters

Ts = 0.01

T = 50

# Time vector

t = np.arange(0, T + Ts, Ts)

# Generate white noise signal

x1 = np.random.rand(len(t))

# Filter parameters

a = np.zeros(3)

b = np.zeros(1)

om0 = 2 \* np.pi

dz = 0.05

A = 1

oms = om0 \* Ts

# Compute filter coefficients

a[0] = 1 + 2 \* dz \* oms + oms\*\*2

a[1] = -2 \* (1 + dz \* oms)

a[2] = 1

b[0] = A \* 2 \* dz \* oms\*\*2

# Apply the filter

y1 = lfilter(b, a, x1)

# Plot the filtered signal

plt.figure()

plt.plot(t, y1)

plt.xlabel('Timp (s)')

plt.ylabel('Amplitudine')

plt.title('Zgomot Alb Filtrat')

plt.grid(True)

plt.gca().set\_facecolor('white')

plt.gca().set\_xlabel('Timp (s)', fontname='Arial', fontsize=12)

plt.gca().set\_ylabel('Amplitudine', fontname='Arial', fontsize=12)

plt.gca().set\_title('Zgomot Alb Filtrat', fontname='Arial', fontsize=12)

# Display the plot

plt.show()

A graph with blue lines

AI-generated content may be incorrect.

***Figure 15: Result for task 4.2***

* 1. Represent the magnitude of the Fourier Transform of the white noise at the input and output of the filter:

#Ex 4.3

import numpy as np

import matplotlib.pyplot as plt

from scipy.signal import lfilter

# Parameters

Ts = 0.01

T = 50

# Time vector

t = np.arange(0, T + Ts, Ts)

# Generate white noise signal

x1 = np.random.rand(len(t))

# Filter parameters

a = np.zeros(3)

b = np.zeros(1)

om0 = 2 \* np.pi

dz = 0.05

A = 1

oms = om0 \* Ts

# Compute filter coefficients

a[0] = 1 + 2 \* dz \* oms + oms\*\*2

a[1] = -2 \* (1 + dz \* oms)

a[2] = 1

b[0] = A \* 2 \* dz \* oms\*\*2

# Apply the filter

y1 = lfilter(b, a, x1)

# Frequency parameters

df = 1 / T  # Frequency resolution

Fmax = 1 / Ts  # Maximum frequency

f = np.arange(-Fmax/2, Fmax/2 + df, df)

# Compute FFT for input and output signals

Fu1 = np.fft.fft(x1)

Fu2 = np.fft.fft(y1)

# Apply fftshift to center the spectra

Fu1p = np.fft.fftshift(Fu1)

Fu2p = np.fft.fftshift(Fu2)

# Compute magnitudes

m = np.abs(Fu1p)

m1 = np.abs(Fu2p)

# Ensure the length of f matches the length of the FFT results

f = f[:len(m)]

# Plot the magnitude spectrum of the input signal (x1)

plt.figure()

plt.stem(f, m, linefmt='b-', markerfmt='bo', basefmt='r-')

plt.xlabel('Frecvența (Hz)')

plt.ylabel('|X1(f)|')

plt.title('Spectrul Modulului (Zgomot Alb - Intrare)')

plt.grid(True)

plt.gca().set\_facecolor('white')

plt.gca().set\_xlabel('Frecvența (Hz)', fontname='Arial', fontsize=12)

plt.gca().set\_ylabel('|X1(f)|', fontname='Arial', fontsize=12)

plt.gca().set\_title('Spectrul Modulului (Zgomot Alb - Intrare)', fontname='Arial', fontsize=12)

# Plot the magnitude spectrum of the output signal (y1)

plt.figure()

plt.stem(f, m1, linefmt='b-', markerfmt='bo', basefmt='r-')

plt.xlabel('Frecvența (Hz)')

plt.ylabel('|Y1(f)|')

plt.title('Spectrul Modulului (Zgomot Alb - Ieșire Filtrată)')

plt.grid(True)

plt.gca().set\_facecolor('white')

plt.gca().set\_xlabel('Frecvența (Hz)', fontname='Arial', fontsize=12)

plt.gca().set\_ylabel('|Y1(f)|', fontname='Arial', fontsize=12)

plt.gca().set\_title('Spectrul Modulului (Zgomot Alb - Ieșire Filtrată)', fontname='Arial', fontsize=12)

# Display the plots

plt.show()

A graph with a blue line

AI-generated content may be incorrect. A graph with a line in the middle

AI-generated content may be incorrect.

***Figure 16: Result for task 4.3***

**Input Signal (White Noise):**

The magnitude spectrum of the white noise shows a nearly uniform distribution across all frequencies, as expected for a white noise signal. The plot confirms the presence of all frequencies with roughly equal power, consistent with the characteristic flat spectral density of white noise

**Filtered Output Signal:**

The magnitude spectrum of the filtered signal demonstrates how the filter affects the noise. Certain frequencies are attenuated, while others are emphasized, depending on the filter's design. The overall shape of the spectrum reflects the filter's frequency response

1. **Investigation of the Continuous Fourier Transform**

* From the "MATLAB DEMO" menu, select **Toolboxes** > **Signal Processing** > **Transforms**, and launch the application by clicking **Continuous Fourier Transform**. The interface will appear on the screen.
* Change the value of the **modulation frequency** and examine its influence on the Fourier Transform Magnitude.
* Save information about the continuous Fourier Transform by clicking the **INFO** button in the modeling window.

***Because of the absence of the MatLab, the task was implemented in Python***

#Ex. 5

import numpy as np

import matplotlib.pyplot as plt

# Parameters

fs = 1000  # Sampling frequency (Hz)

T = 2  # Total time duration (seconds)

t = np.arange(-T/2, T/2, 1/fs)  # Time vector from -1 to 1 seconds

# Signal parameters

A = 1  # Amplitude

m = 0.5  # Modulation index

fc = 2  # Carrier frequency (Hz)

fm\_values = [1, 2.5, 4]  # Modulation frequencies to test (Hz)

# Frequency vector for FFT

N = len(t)

df = fs / N  # Frequency resolution

f = np.arange(-fs/2, fs/2, df)  # Frequency vector from -fs/2 to fs/2

# Create subplots: one row per modulation frequency, two columns (time and frequency)

fig, axs = plt.subplots(len(fm\_values), 2, figsize=(10, 3 \* len(fm\_values)))

for i, fm in enumerate(fm\_values):

    # Generate the modulated signal

    s = A \* (1 + m \* np.cos(2 \* np.pi \* fm \* t)) \* np.cos(2 \* np.pi \* fc \* t)

    # Compute the FFT and shift it

    S = np.fft.fft(s)

    S\_shifted = np.fft.fftshift(S)

    mag = np.abs(S\_shifted)

    # Plot the time-domain signal

    axs[i, 0].plot(t, s)

    axs[i, 0].set\_xlabel('Time (Seconds)')

    axs[i, 0].set\_ylabel('Waveform')

    axs[i, 0].set\_title(f'Waveform (fm = {fm} Hz)')

    axs[i, 0].grid(True)

    axs[i, 0].set\_xlim(-1, 1)

    axs[i, 0].set\_facecolor('white')

    # Plot the magnitude spectrum (limit frequency axis to -6 to 6 Hz)

    axs[i, 1].plot(f, mag)

    axs[i, 1].set\_xlabel('Frequency (Hertz)')

    axs[i, 1].set\_ylabel('Magnitude')

    axs[i, 1].set\_title(f'Magnitude (fm = {fm} Hz)')

    axs[i, 1].grid(True)

    axs[i, 1].set\_xlim(-6, 6)

    axs[i, 1].set\_facecolor('white')

# Adjust layout to prevent overlap

plt.tight\_layout()

# Display the plots

plt.show()

A group of graphs showing the waves

AI-generated content may be incorrect.

***Figure 17: Result for task 5***

The results illustrate the effects of amplitude modulation for three different modulation frequencies (( fm = 1 ), ( 2.5 ), and ( 4 ) Hz). Each row in the plots corresponds to a specific ( fm ) value, with the left column showing the time-domain waveforms and the right column depicting their magnitude spectra.

**Time-Domain Waveforms:**

As the modulation frequency (( fm )) increases, the number of oscillations (modulation cycles) within the same time frame also increases. The amplitude modulation produces a waveform with varying envelope sizes, determined by the modulation index (( m = 0.5 )).

**Frequency-Domain Magnitude Spectra:**

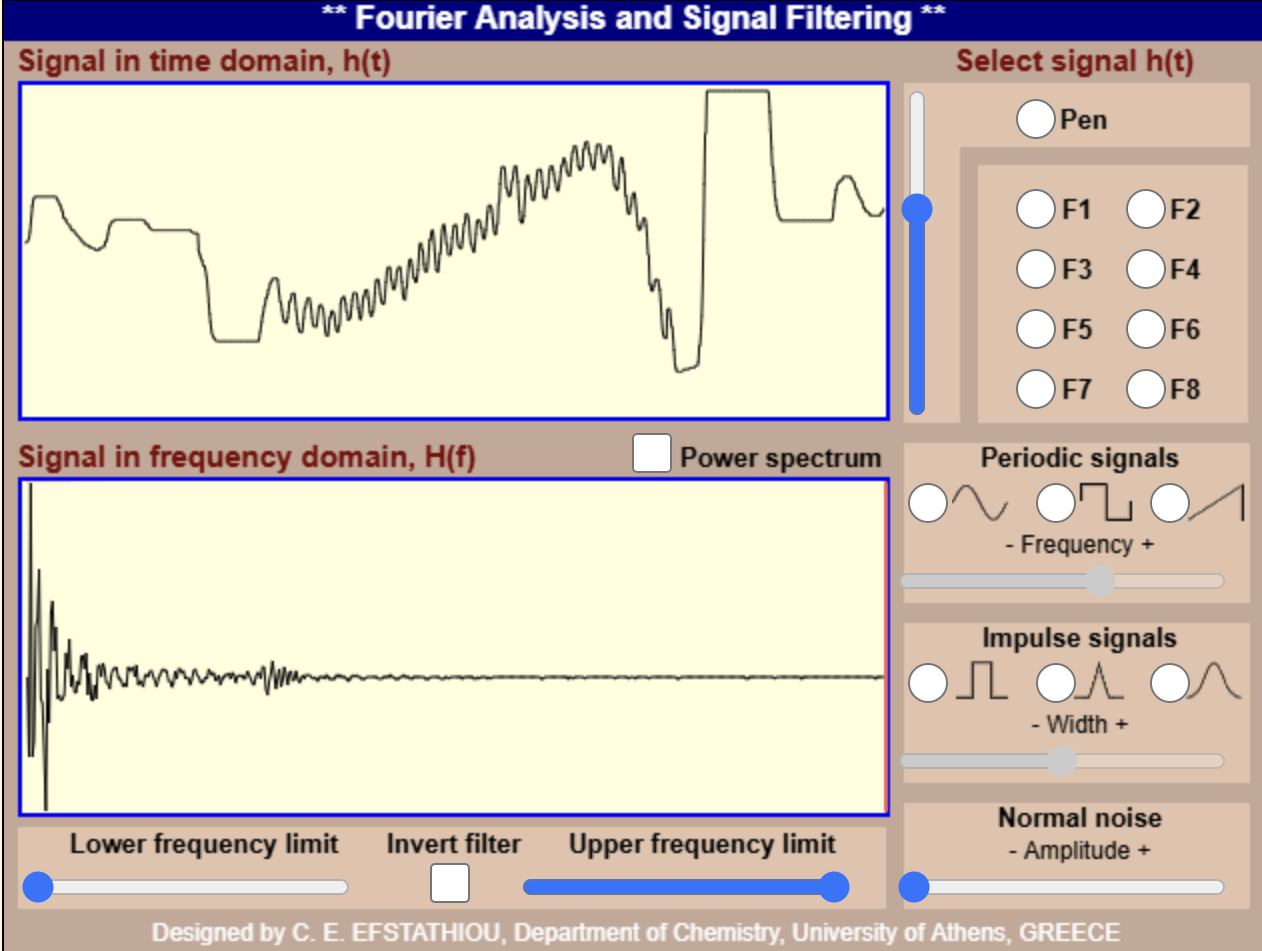
Each magnitude spectrum shows peaks at the carrier frequency (( fc = 2 ) Hz) and at sidebands offset by the modulation frequency (( fc \pm fm )). As ( fm ) increases, the sidebands shift further apart, which clearly reflects the relationship between modulation frequency and its impact on the signal's frequency content.

1. **Analysis and Synthesis of Signals**
   1. Open the demonstration app:  
      <http://195.134.76.37/applets/AppletFourAnal/Appl_FourAnal2.html>  
      Study and save the signal shapes and their spectrum — for 5–6 signals.

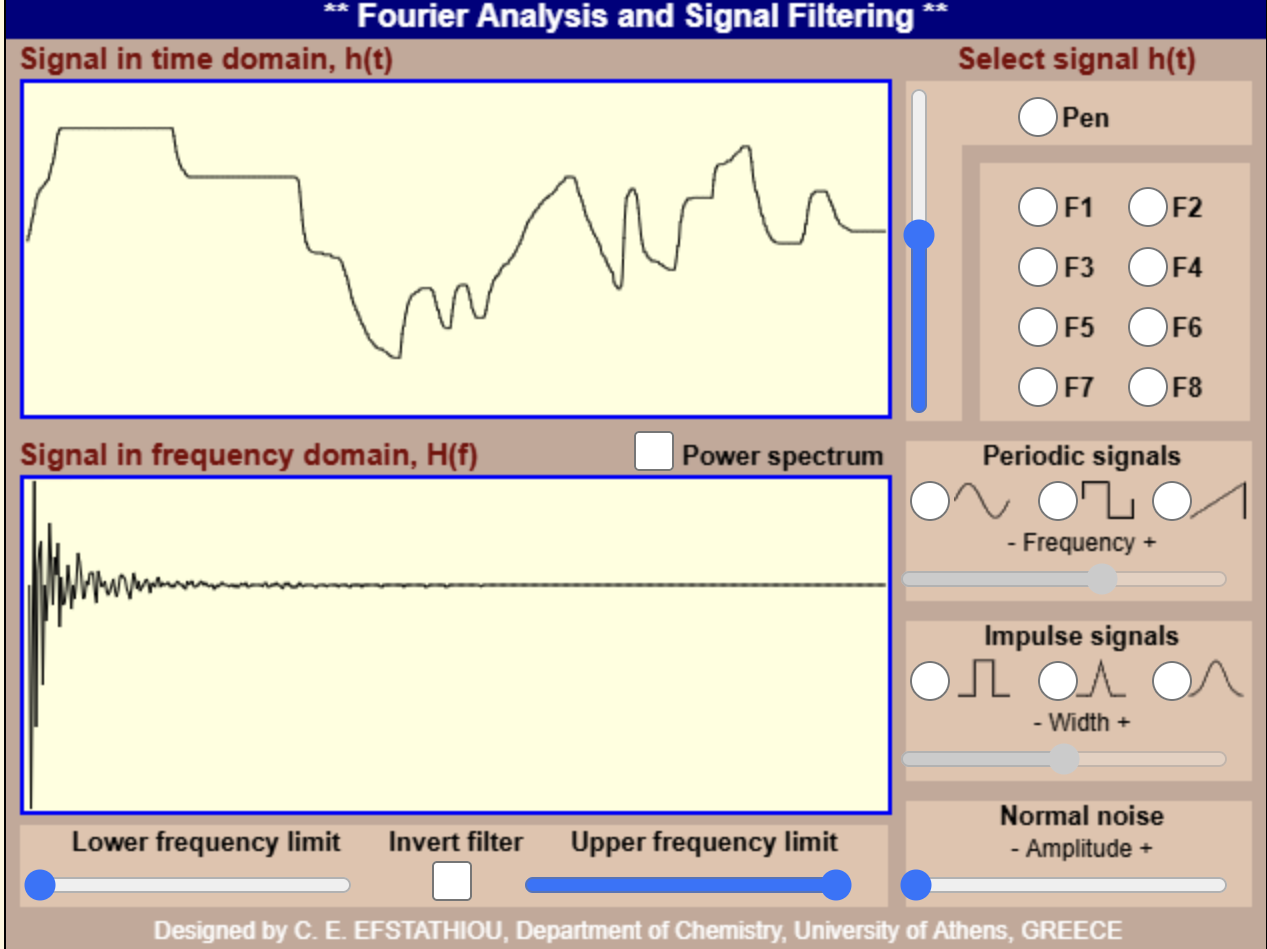
A screenshot of a computer

AI-generated content may be incorrect.

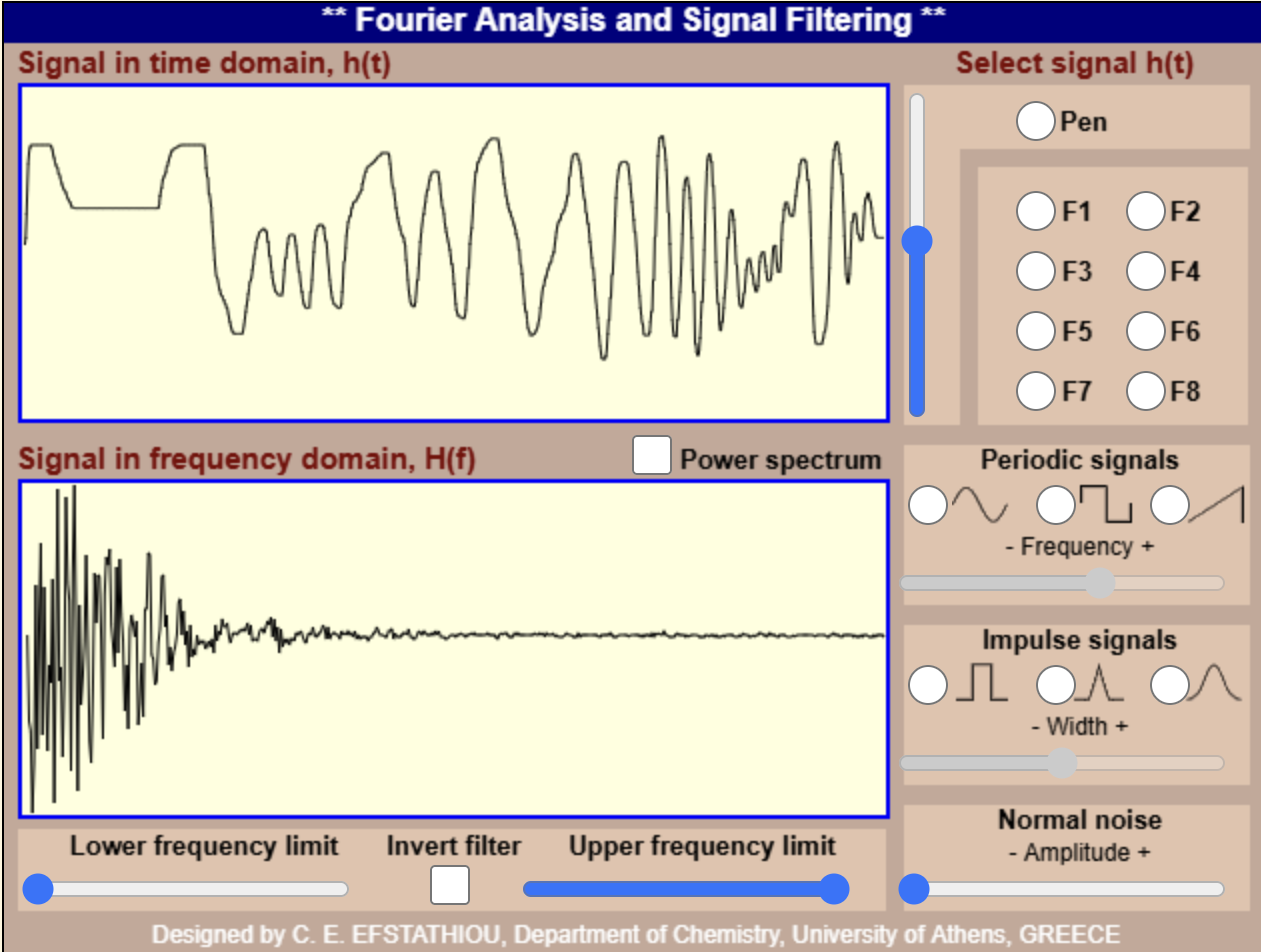
***Figure 18: Result for task 6.1***



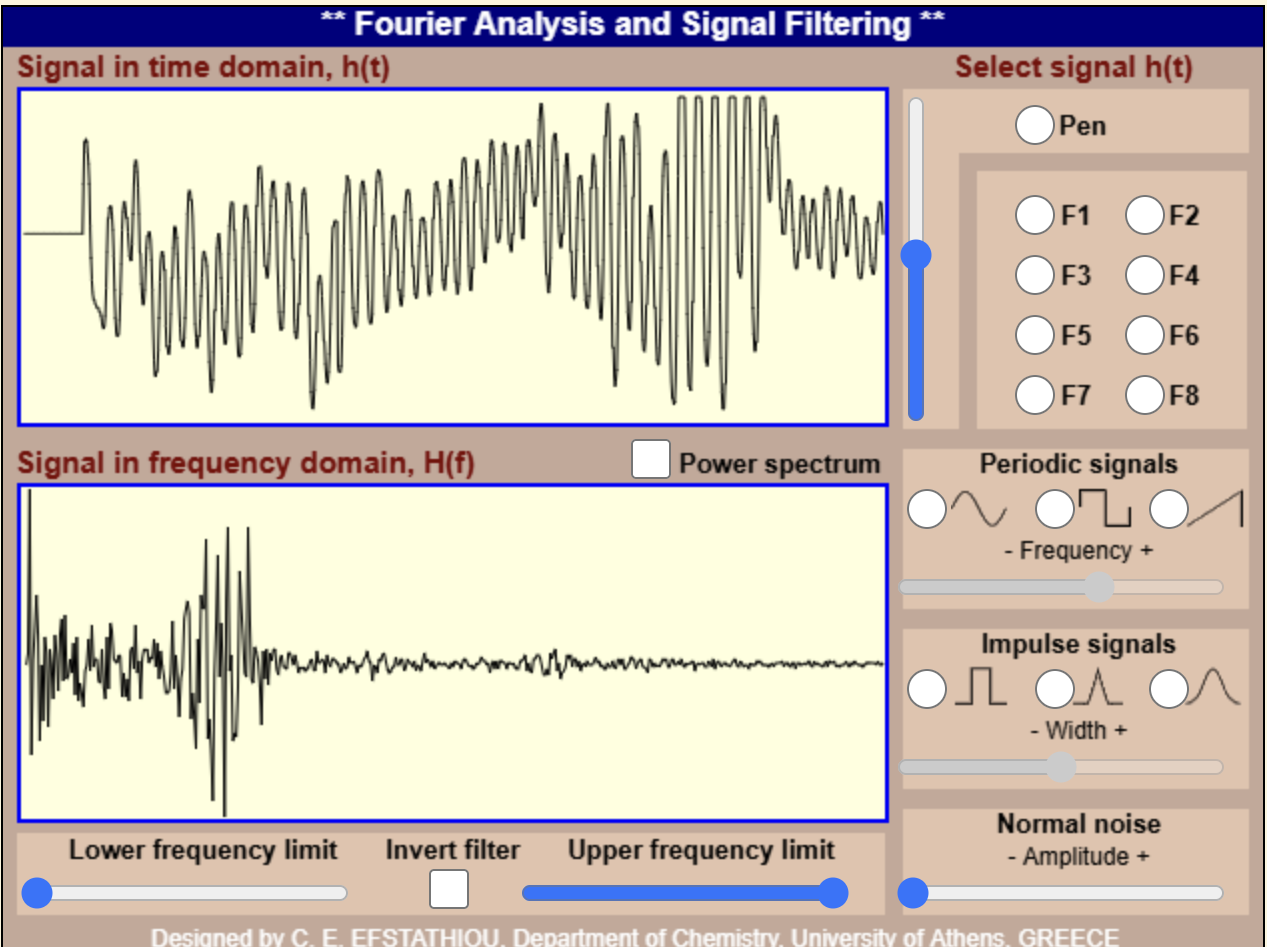
***Figure 19: Result for task 6.1***



***Figure 20: Result for task 6.1***

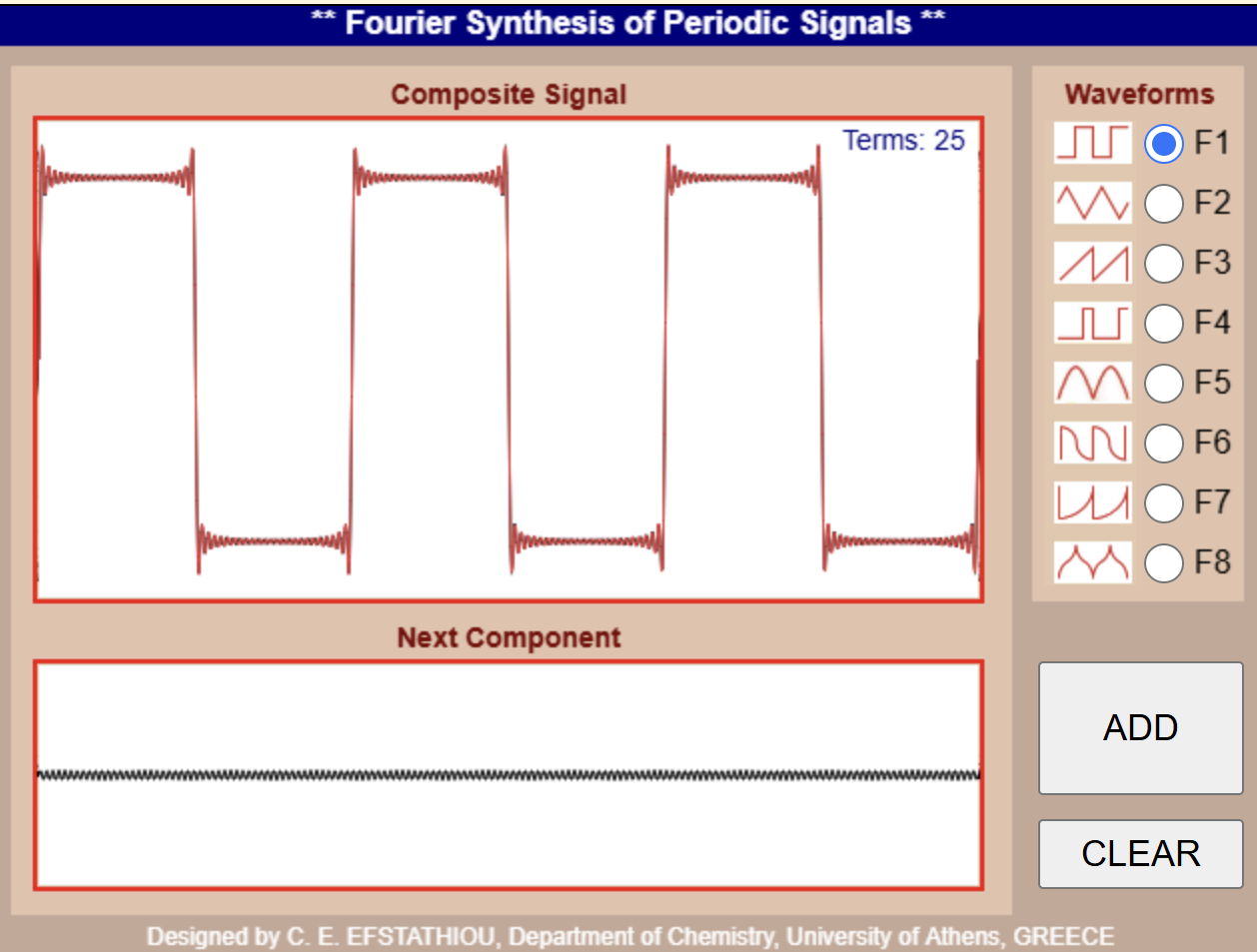
`

***Figure 21: Result for task 6.1***

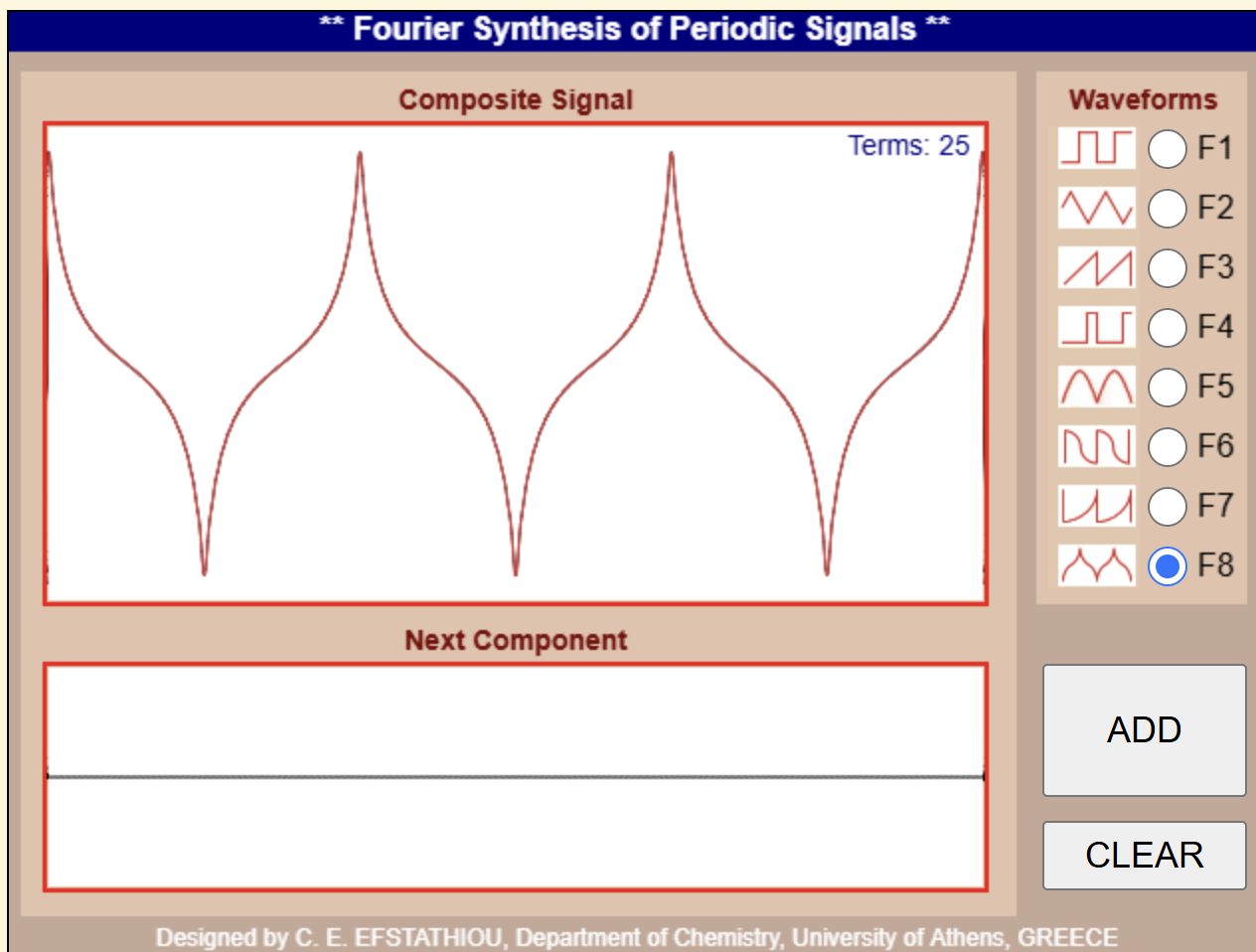


***Figure 22: Result for task 6.1***

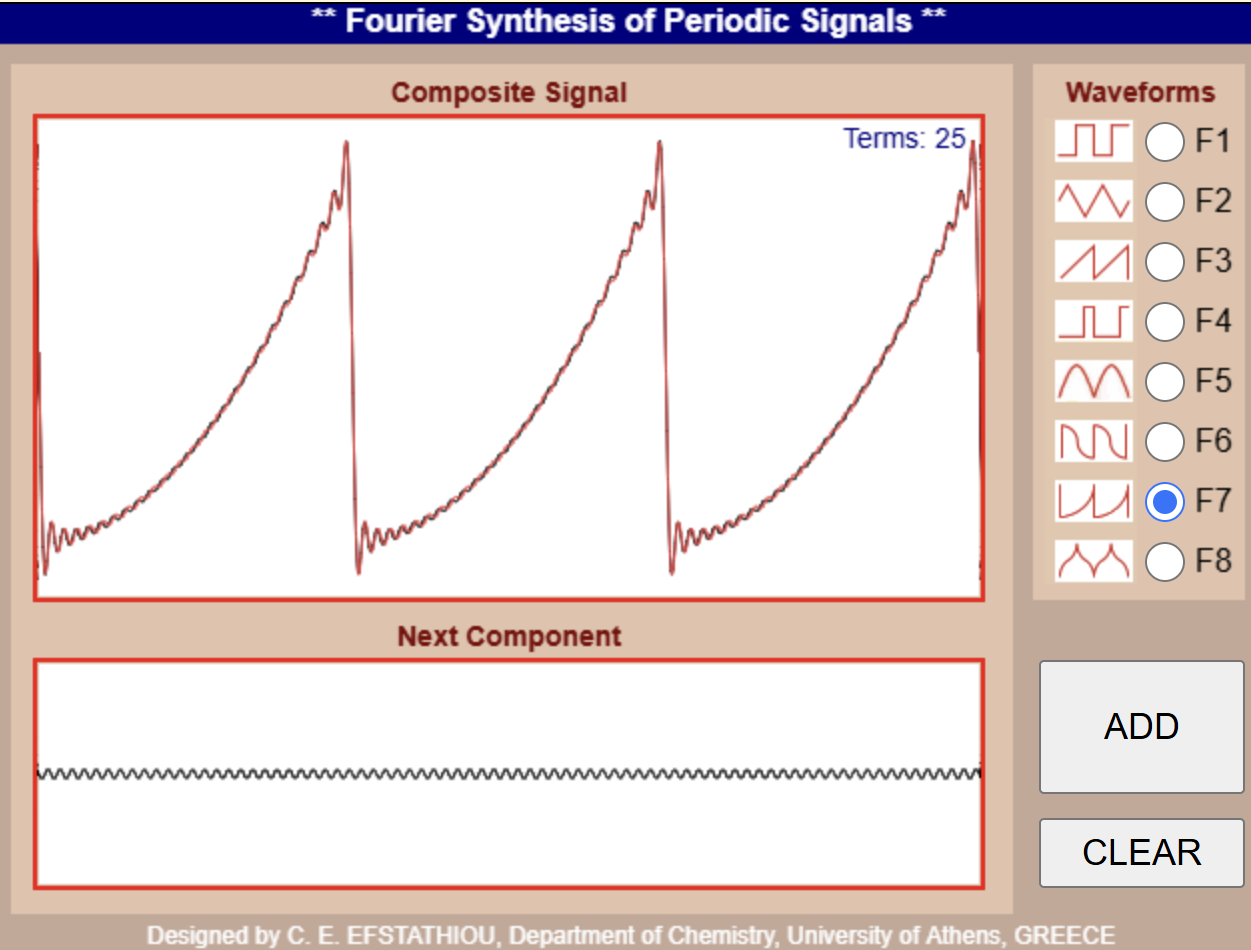
* 1. Open the demonstration app:  
     <http://195.134.76.37/applets/AppletFourier/Appl_Fourier2.html>  
     Study and save the synthesis of signals by progressively adding a different number of sinusoids (25–30 sinusoids) — for 5–6 signals.



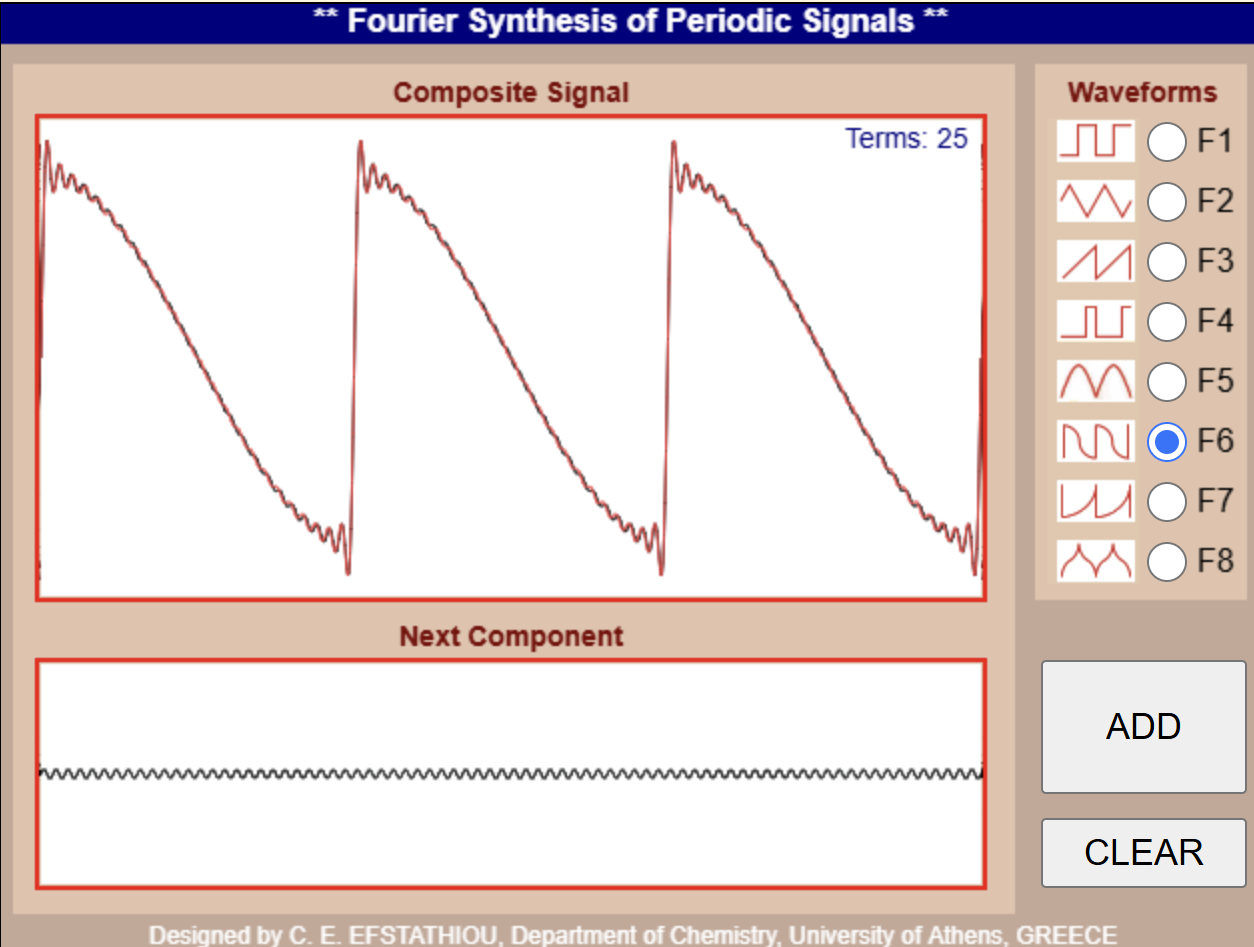
***Figure 23: Result for task 6.2***



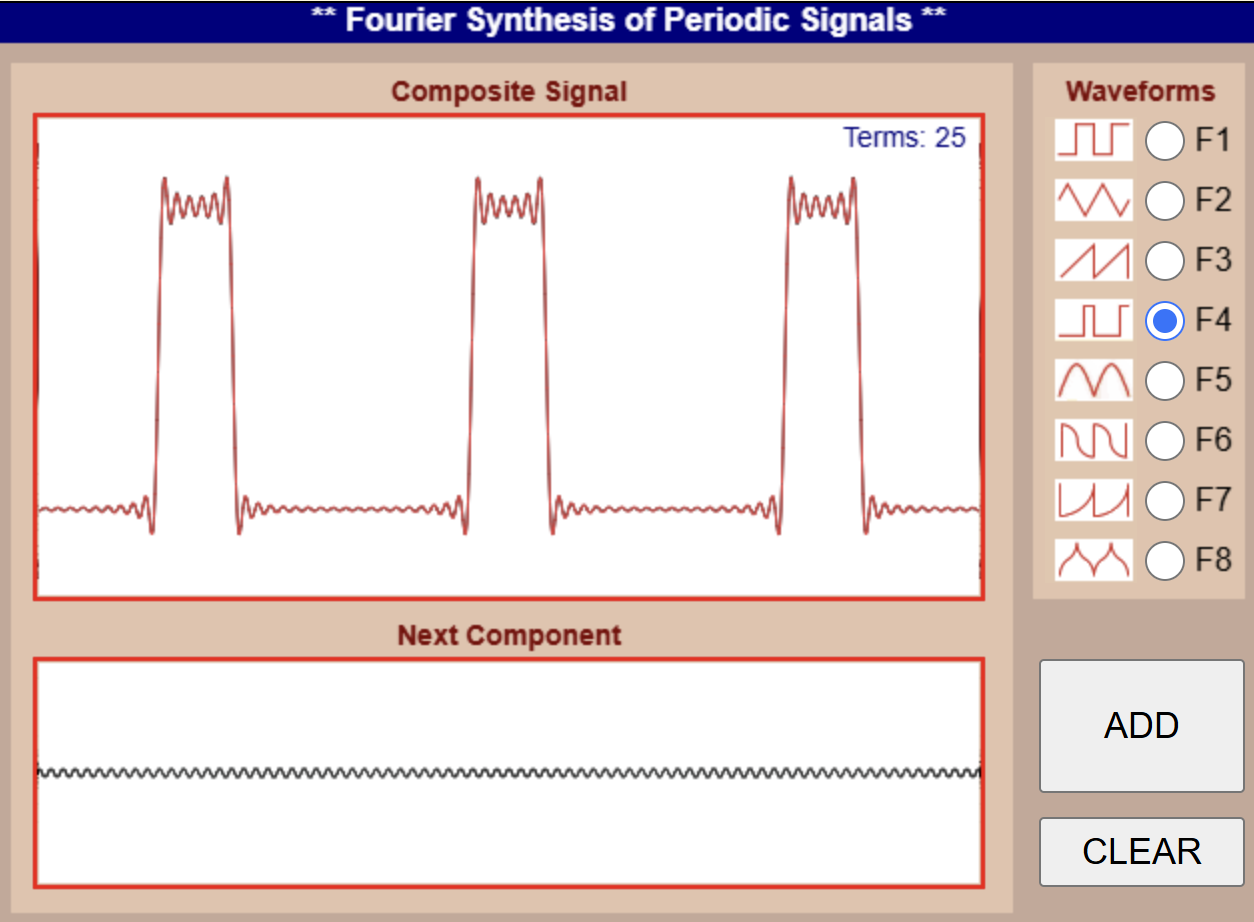
***Figure 24: Result for task 6.2***



***Figure 25: Result for task 6.2***



***Figure 26: Result for task 6.2***



***Figure 27: Result for task 6.2***

**Conclusions**

This work explored fundamental concepts of signal processing through a series of tasks implemented in Python, focusing on discrete signals, their Fourier Transforms, and filtering techniques. Task 1 analyzed a 20 Hz sinusoidal signal sampled at 128 Hz, confirming its frequency content via FFT and demonstrating aliasing effects at 108 Hz, as well as spectral leakage for a 19 Hz signal. Note 2 extended this analysis across frequencies below and above the Nyquist limit, illustrating aliasing for higher frequencies like 80 Hz, 100 Hz, and 120 Hz. Task 2 examined the FFT of a signal composed of two sinusoids, revealing distinct frequency peaks and phase behavior, with Note 3 reinforcing these observations for varied frequency pairs. Task 3 modeled a rectangular pulse, showing its sinc-shaped spectrum, with Notes 3 and 3.3-3.4 exploring the effects of pulse width and the real and imaginary components of its FFT. Task 4 generated white noise, filtered it with a low-pass filter, and compared the input and output spectra, highlighting the filter's attenuation of high frequencies. Finally, an interactive GUI was developed to simulate a MATLAB demo, enabling real-time analysis of modulation frequency effects on a signal's Fourier Transform. Collectively, these tasks provided a comprehensive understanding of signal representation, frequency analysis, and the practical implications of sampling and filtering in digital signal processing.