REPORT

Zajęcia: Analog and digital electronic circuits

Teacher: prof. dr hab. Vasyl Martsenyuk

Lab 1

Date: 11.10.2025

Topic: "Wprowadzenie do narzędzi i środowiska pracy w przetwarzaniu sygnałów cyfrowych: Python + biblioteki. Analiza sygnałów deterministycznych: implementacja podstawowych operacji na sygnałach czasowych."

Variant: 5

Andrzej Jasiński

Informatyka II stopień,

niestacjonarne,

1 semestr,

Gr. 1b

1. Problem statement

Synthesize a discrete-time signal by using the IDFT in matrix notation for different values of N. Show the matrices W and K. Plot the signal synthesized.

$x_\mu = [6, 4, 4, 5, 3, 4, 5, 0, 0, 0, 0]^T$

2. Input data:

- **Spectrum Vector :** $\mathbf{x_\mu = [6, 4, 4, 5, 3, 4, 5, 0, 0, 0, 0]^T}$
- **Signal Length (N):** The length of the vector $\mathbf{x_\mu}$ determines the block length N.
- **N** = 11
- **Objective:** Calculate the discrete-time signal vector x_k using the IDFT matrix equation: $x_k = (1/N) * W * x_\mu$

3. Commands used (or GUI):

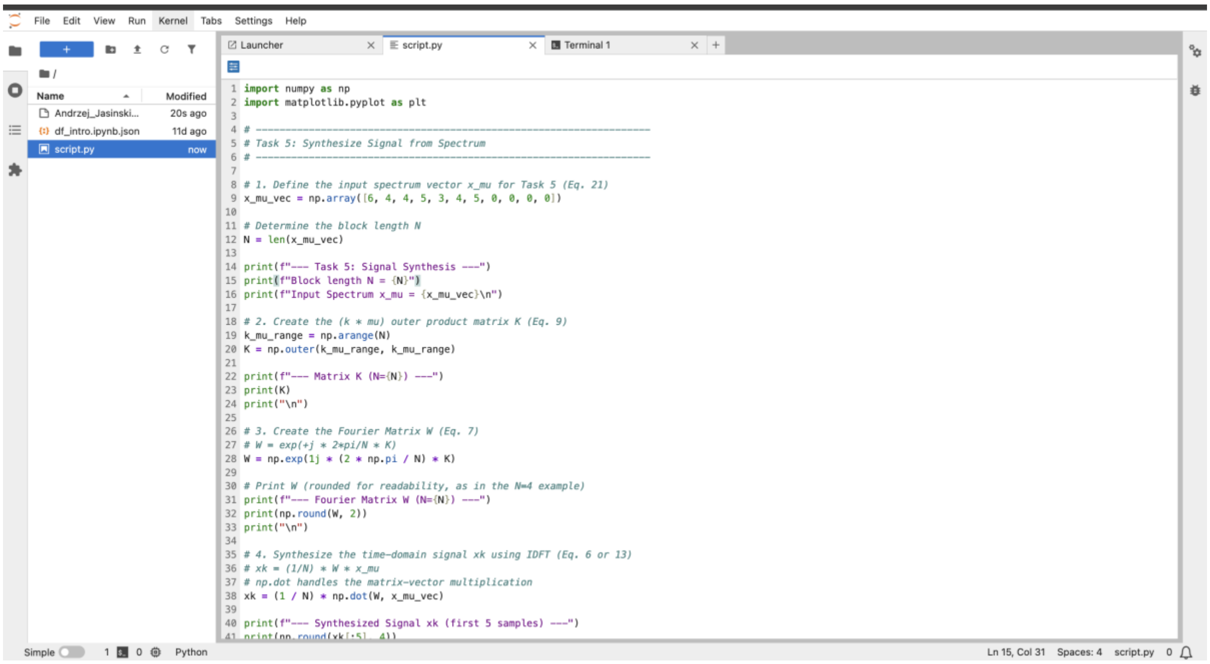
a) source code

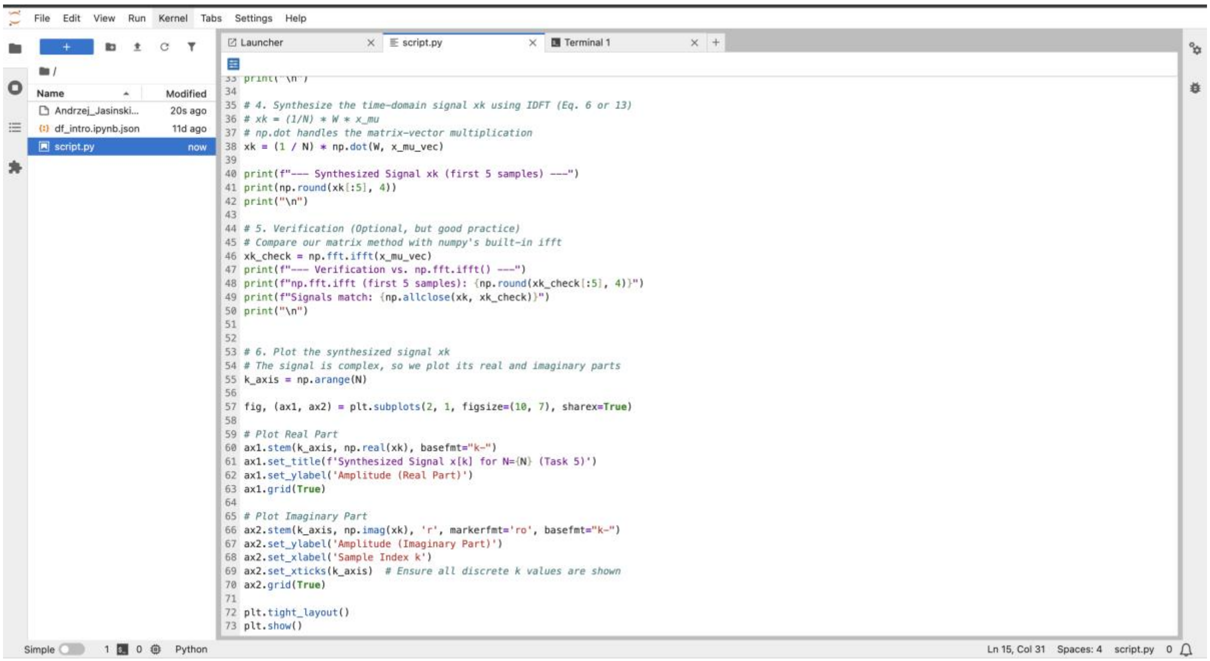
```
import numpy as np
import matplotlib.pyplot as plt

# ---------------------------------------------------------------
# Task 5: Synthesize Signal from Spectrum
# ---------------------------------------------------------------

# 1. Define the input spectrum vector x_mu for Task 5 (Eq. 21)
x_mu_vec = np.array([6, 4, 4, 5, 3, 4, 5, 0, 0, 0, 0])

# Determine the block length N
N = len(x_mu_vec)
```

```
print(f"--- Task 5: Signal Synthesis ---")
print(f"Block length N = {N}")
print(f"Input Spectrum x_mu = {x_mu_vec}\n")

# 2. Create the (k * mu) outer product matrix K (Eq. 9)
k_mu_range = np.arange(N)
K = np.outer(k_mu_range, k_mu_range)

print(f"--- Matrix K (N={N}) ---")
print(K)
print("\n")

# 3. Create the Fourier Matrix W (Eq. 7)
# W = exp(+j * 2*pi/N * K)
W = np.exp(1j * (2 * np.pi / N) * K)

# Print W (rounded for readability, as in the N=4 example)
print(f"--- Fourier Matrix W (N={N}) ---")
print(np.round(W, 2))
print("\n")

# 4. Synthesize the time-domain signal xk using IDFT (Eq. 6 or 13)
# xk = (1/N) * W * x_mu
# np.dot handles the matrix-vector multiplication
xk = (1 / N) * np.dot(W, x_mu_vec)

print(f"--- Synthesized Signal xk (first 5 samples) ---")
print(np.round(xk[:5], 4))
print("\n")

# 5. Verification (Optional, but good practice)
# Compare our matrix method with numpy's built-in ifft
xk_check = np.fft.ifft(x_mu_vec)
print(f"--- Verification vs. np.fft.ifft() ---")
print(f"np.fft.ifft (first 5 samples): {np.round(xk_check[:5], 4)}")
print(f"Signals match: {np.allclose(xk, xk_check)}")
print("\n")


# 6. Plot the synthesized signal xk
```

```python
# The signal is complex, so we plot its real and imaginary parts
k_axis = np.arange(N)

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 7), sharex=True)

# Plot Real Part
ax1.stem(k_axis, np.real(xk), basefmt="k-")
ax1.set_title(f'Synthesized Signal x[k] for N={N} (Task 5)')
ax1.set_ylabel('Amplitude (Real Part)')
ax1.grid(True)

# Plot Imaginary Part
ax2.stem(k_axis, np.imag(xk), 'r', markerfmt='ro', basefmt="k-")
ax2.set_ylabel('Amplitude (Imaginary Part)')
ax2.set_xlabel('Sample Index k')
ax2.set_xticks(k_axis)  # Ensure all discrete k values are shown
ax2.grid(True)

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

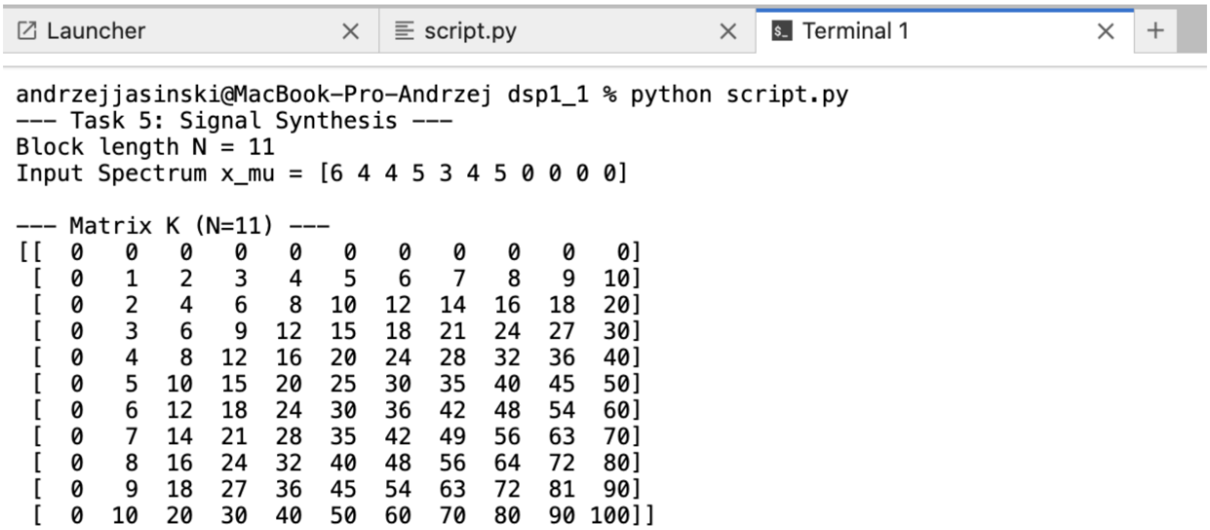
b) screenshots



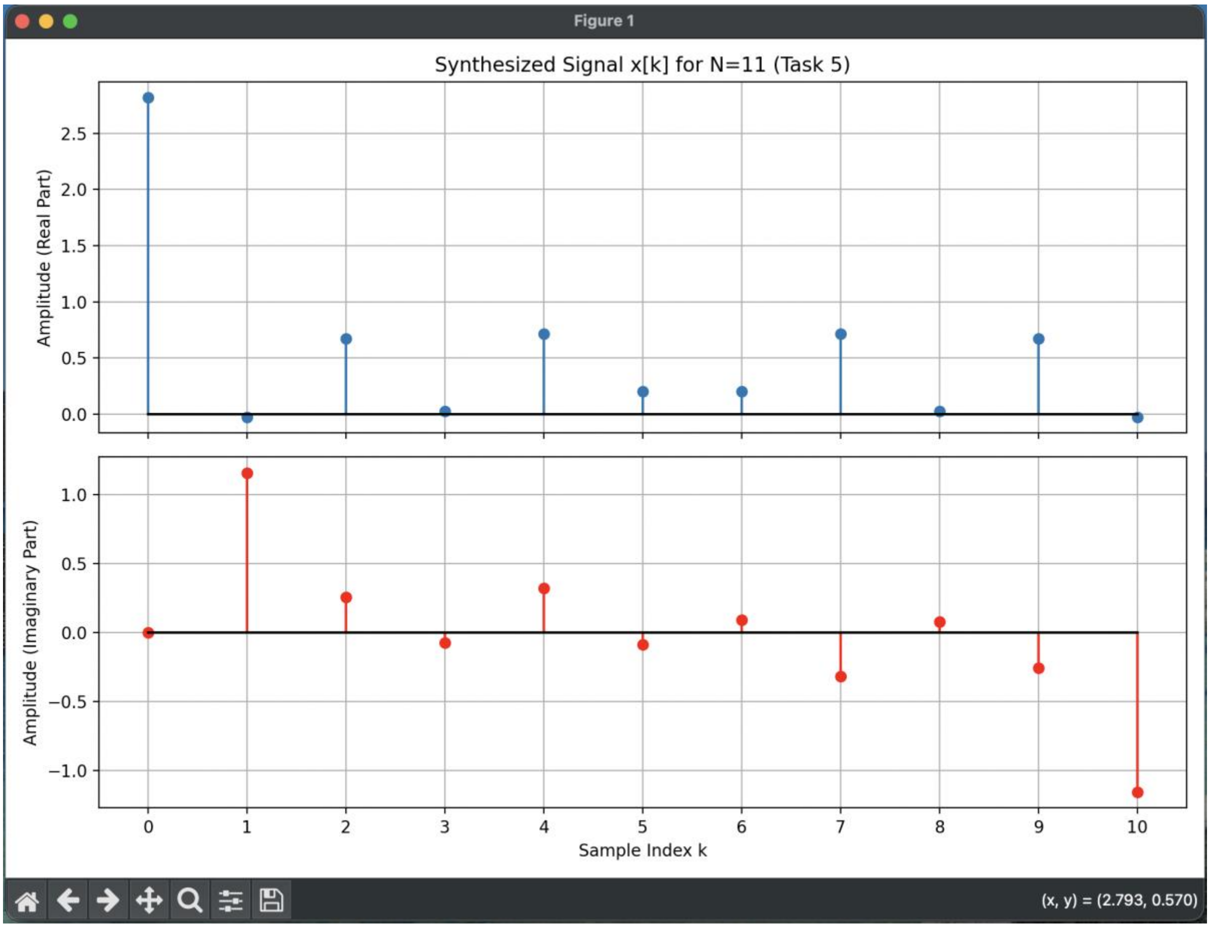


Link to remote repository: https://github.com/Thran34/dsp1_1

4. Outcomes:



```
--- Fourier Matrix W (N=11) ---
[[ 1.   +0.j    1.   +0.j    1.   +0.j    1.   +0.j    1.   +0.j    1.   +0.j
   1.   +0.j    1.   +0.j    1.   +0.j    1.   +0.j    1.   +0.j  ]
 [ 1.   +0.j    0.84+0.54j  0.42+0.91j -0.14+0.99j -0.65+0.76j -0.96+0.28j
  -0.96-0.28j -0.65-0.76j -0.14-0.99j  0.42-0.91j  0.84-0.54j]
 [ 1.   +0.j    0.42+0.91j -0.65+0.76j -0.96-0.28j -0.14-0.99j  0.84-0.54j
   0.84+0.54j -0.14+0.99j -0.96+0.28j -0.65-0.76j  0.42-0.91j]
 [ 1.   +0.j   -0.14+0.99j -0.96-0.28j  0.42-0.91j  0.84+0.54j -0.65+0.76j
  -0.65-0.76j  0.84-0.54j  0.42+0.91j -0.96+0.28j -0.14-0.99j]
 [ 1.   +0.j   -0.65+0.76j -0.14-0.99j  0.84+0.54j -0.96+0.28j  0.42-0.91j
   0.42+0.91j -0.96-0.28j  0.84-0.54j -0.14+0.99j -0.65-0.76j]
 [ 1.   +0.j   -0.96+0.28j  0.84-0.54j -0.65+0.76j  0.42-0.91j -0.14+0.99j
  -0.14-0.99j  0.42+0.91j -0.65-0.76j  0.84+0.54j -0.96-0.28j]
 [ 1.   +0.j   -0.96-0.28j  0.84+0.54j -0.65-0.76j  0.42+0.91j -0.14-0.99j
  -0.14+0.99j  0.42-0.91j -0.65+0.76j  0.84-0.54j -0.96+0.28j]
 [ 1.   +0.j   -0.65-0.76j -0.14+0.99j  0.84-0.54j -0.96-0.28j  0.42+0.91j
   0.42-0.91j -0.96+0.28j  0.84+0.54j -0.14-0.99j -0.65+0.76j]
 [ 1.   +0.j   -0.14-0.99j -0.96+0.28j  0.42+0.91j  0.84-0.54j -0.65-0.76j
  -0.65+0.76j  0.84+0.54j  0.42-0.91j -0.96-0.28j -0.14+0.99j]
 [ 1.   +0.j    0.42-0.91j -0.65-0.76j -0.96+0.28j -0.14+0.99j  0.84+0.54j
   0.84-0.54j -0.14-0.99j -0.96-0.28j -0.65+0.76j  0.42+0.91j]
 [ 1.   +0.j    0.84-0.54j  0.42-0.91j -0.14-0.99j -0.65-0.76j -0.96-0.28j
  -0.96+0.28j -0.65+0.76j -0.14+0.99j  0.42+0.91j  0.84+0.54j]]


--- Synthesized Signal xk (first 5 samples) ---
[ 2.8182+0.j     -0.0259+1.1578j  0.6717+0.2567j  0.0273-0.0772j
  0.7162+0.3202j]


--- Verification vs. np.fft.ifft() ---
np.fft.ifft (first 5 samples): [ 2.8182+0.j     -0.0259+1.1578j  0.6717+0.2567j  0.0273-0.0772j
  0.7162+0.3202j]
Signals match: True
```



5. Conclusions: For the reasons given, we conclude that the objective of synthesizing the discrete-time signal x_k from its spectrum x_u using the matrix IDFT formula, $x_k = 1/N * W_{xu}$ was successfully achieved. The required K and W matrices for N=11 were programmatically generated in Python to compute the complex-valued time-domain signal. The resulting x_k correctly reflected the input spectrum's properties, such as lacking high-frequency components where x_u was zero. The entire matrix-based method was validated as correct against Python's built-in np.fft.ifft() function, confirming this as an effective approach for signal synthesis.

Attention! In the case of several tasks, only one report must be prepared for the entire activity, which covers all tasks