

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.

$$\mathbf{x}_\mu = [6, 4, 4, 5, 3, 4, 5, 0, 0, 0, 0]^T$$

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}_μ determines the block length N .
- $N = 11$
- **Objective**: Calculate the discrete-time signal vector \mathbf{x}_k using the IDFT matrix equation: $\mathbf{x}_k = (1/N) * W * \mathbf{x}_\mu$

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

```

```
# 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')
```

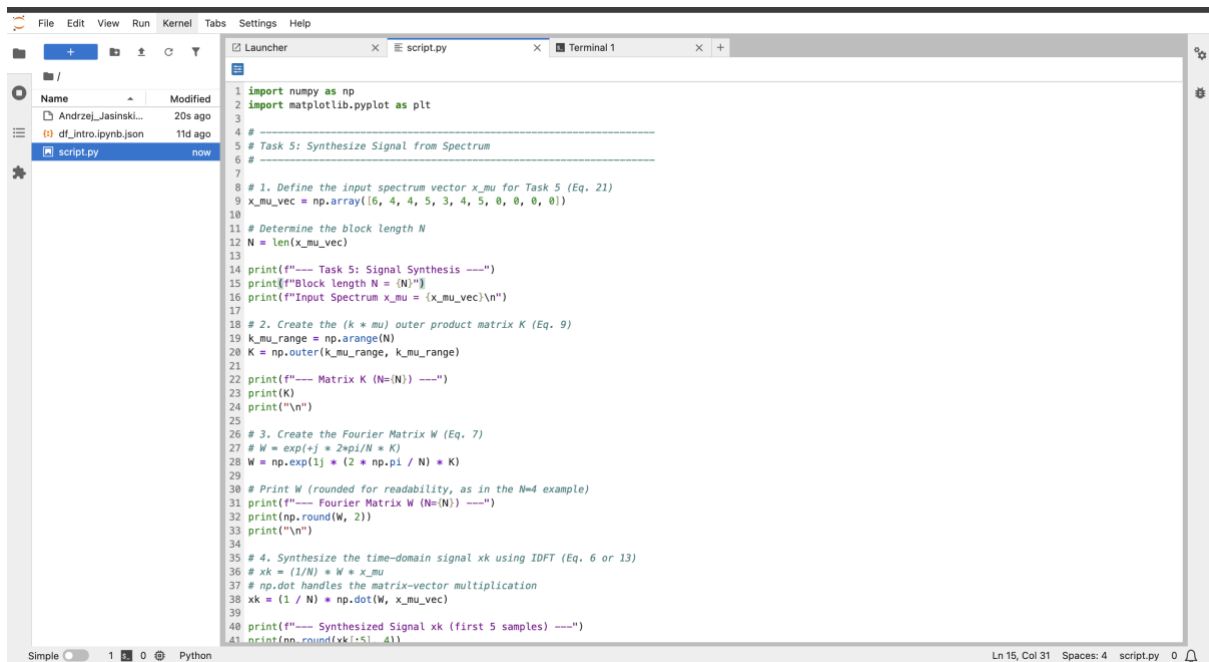
```
ax2.set_xticks(k_axis) # Ensure all discrete k values are shown
```

```
ax2.grid(True)
```

```
plt.tight_layout()
```

```
plt.show()
```

b) screenshots



The screenshot shows a Jupyter Notebook interface with a file explorer on the left and a code editor on the right. The code editor displays a Python script for signal synthesis. The script imports numpy and matplotlib, defines a spectrum vector, calculates the block length, creates the outer product matrix K, the Fourier matrix W, and finally synthesizes the time-domain signal xk using IDFT. The script includes several print statements for debugging and verification.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Task 5: Synthesize Signal from Spectrum
5
6 # 1. Define the input spectrum vector x_mu for Task 5 (Eq. 21)
7 x_mu_vec = np.array([6, 4, 4, 5, 3, 4, 5, 0, 0, 0])
8
9 # Determine the block length N
10 N = len(x_mu_vec)
11
12 print(f"--- Task 5: Signal Synthesis ---")
13 print(f"Block length N = {N}")
14 print(f"Input Spectrum x_mu = {x_mu_vec}")
15
16 # 2. Create the (k * mu) outer product matrix K (Eq. 9)
17 k_mu_range = np.arange(N)
18 K = np.outer(k_mu_range, k_mu_range)
19
20 print(f"--- Matrix K (N={N}) ---")
21 print(K)
22 print("\n")
23
24 # 3. Create the Fourier Matrix W (Eq. 7)
25 W = np.exp(1j * (2 * np.pi / N) * K)
26
27 # Print W (rounded for readability, as in the N=4 example)
28 print(f"--- Fourier Matrix W (N={N}) ---")
29 print(np.round(W, 2))
30 print("\n")
31
32 # 4. Synthesize the time-domain signal xk using IDFT (Eq. 6 or 13)
33 # xk = (1/N) * W * x_mu
34 # np.dot handles the matrix-vector multiplication
35 xk = (1 / N) * np.dot(W, x_mu_vec)
36
37 print(f"--- Synthesized Signal xk (first 5 samples) ---")
38 print(np.round(xk[:5], 4))
```

```

33 print('\n')
34
35 # 4. Synthesize the time-domain signal x_k using IDFT (Eq. 6 or 13)
36 x_k = (1/N) * W * x_mu
37 # np.dot handles the matrix-vector multiplication
38 x_k = (1 / N) * np.dot(W, x_mu_vec)
39
40 print(f"--- Synthesized Signal x_k (first 5 samples) ---")
41 print(np.round(x_k[:5], 4))
42 print("\n")
43
44 # 5. Verification (Optional, but good practice)
45 # Compare our matrix method with numpy's built-in ifft
46 x_k_check = np.fft.ifft(x_mu_vec)
47 print(f"--- Verification vs. np.fft.ifft() ---")
48 print(f"np.fft.ifft (first 5 samples): {np.round(x_k_check[:5], 4)}")
49 print(f"Signals match: {np.allclose(x_k, x_k_check)}")
50 print("\n")
51
52
53 # 6. Plot the synthesized signal x_k
54 # The signal is complex, so we plot its real and imaginary parts
55 k_axis = np.arange(N)
56
57 fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 7), sharex=True)
58
59 # Plot Real Part
60 ax1.stem(k_axis, np.real(x_k), basefmt="k-")
61 ax1.set_title(f'Synthesized Signal x[k] for N={N} (Task 5)')
62 ax1.set_ylabel('Amplitude (Real Part)')
63 ax1.grid(True)
64
65 # Plot Imaginary Part
66 ax2.stem(k_axis, np.imag(x_k), 'r', markerfmt='ro', basefmt="k-")
67 ax2.set_ylabel('Amplitude (Imaginary Part)')
68 ax2.set_xlabel('Sample Index k')
69 ax2.set_xticks(k_axis) # Ensure all discrete k values are shown
70 ax2.grid(True)
71
72 plt.tight_layout()
73 plt.show()

```

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

4. Outcomes:

```

andzejjasinski@MacBook-Pro-Andrzej dsp1_1 % python script.py
--- Task 5: Signal Synthesis ---
Block length N = 11
Input Spectrum x_mu = [6 4 4 5 3 4 5 0 0 0 0]

--- Matrix K (N=11) ---
[[ 0  0  0  0  0  0  0  0  0  0  0]
 [ 0  1  2  3  4  5  6  7  8  9 10]
 [ 0  2  4  6  8 10 12 14 16 18 20]
 [ 0  3  6  9 12 15 18 21 24 27 30]
 [ 0  4  8 12 16 20 24 28 32 36 40]
 [ 0  5 10 15 20 25 30 35 40 45 50]
 [ 0  6 12 18 24 30 36 42 48 54 60]
 [ 0  7 14 21 28 35 42 49 56 63 70]
 [ 0  8 16 24 32 40 48 56 64 72 80]
 [ 0  9 18 27 36 45 54 63 72 81 90]
 [ 0 10 20 30 40 50 60 70 80 90 100]]

```

```

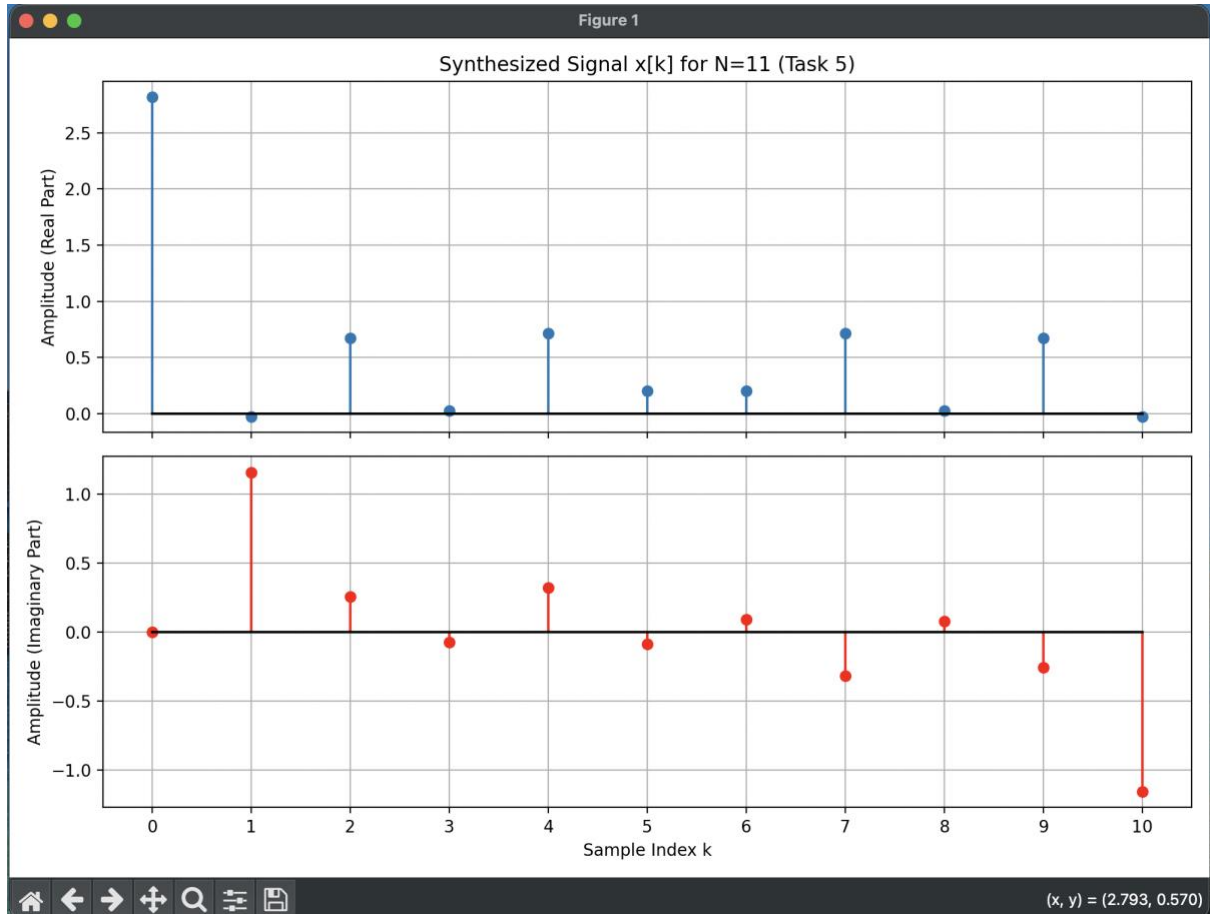
--- Fourier Matrix W (N=11) ---
[[ 1. +0.j    1. +0.j    1. +0.j    1. +0.j    1. +0.j    1. +0.j
  1. +0.j    1. +0.j    1. +0.j    1. +0.j    1. +0.j    1. +0.j ]
 [ 1. +0.j    0.84+0.54j  0.42+0.91j -0.14+0.99j -0.65+0.76j -0.96+0.28j
 -0.96-0.28j -0.65-0.76j -0.14-0.99j  0.42-0.91j  0.84-0.54j]
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