REPORT

Zajęcia: Analog and digital electronic circuits

Teacher: prof. dr hab. Vasyl Martsenyuk, dr inż. Halyna Nahorniak

Lab 1

Date: 27.02.2026

Topic: Spectral Analysis of Deterministic Signals

Variant 11

Jakub Kołodziej

Informatyka II stopień,

stacjonarne,

2 semestr,

Gr.1b

1. Problem Statement

The objective of Lab 1 is to perform spectral analysis of deterministic signals using the Discrete Fourier Transform (DFT) and its implementation via matrix multiplication. The task is to synthesize a discrete-time signal by applying the Inverse DFT (IDFT) in matrix form, starting from a known DFT spectrum vector.

The required deliverables are:

- Construct the outer-product matrix **K** and the Fourier matrix **W** for a given N.
- Synthesize the time-domain signal **x[k]** using the IDFT matrix formula.
- Plot the synthesized signal and the DFT spectrum.
- Verify results against `numpy.fft.ifft()` and confirm the round-trip DFT.
- Show and discuss the DFT eigensignals (columns of W).

2. Input Data

For **Variant 11**, the DFT coefficient vector and block length are:

$$\mathbf{x}\mu = [6, 2, 4, 3, 4, 5, 0, 0, 0, 0]^T \qquad \mathbf{N = 10}$$

The spectrum contains a **DC component** at μ=0 (amplitude 6), five non-zero spectral components at μ=1…5 with amplitudes [2, 4, 3, 4, 5], and **zero-padding** for μ=6…9. Because the spectrum is not conjugate-symmetric, the synthesized signal will be **complex-valued**.

Table 1: DFT spectrum values X[μ]

μ	0	1	2	3	4	5	6	7	8	9
X[μ]	6	2	4	3	4	5	0	0	0	0

3. Commands and Source Code

The solution was implemented in **Python 3** using a **Jupyter Notebook** executed in **PyCharm Professional**. Key libraries: `numpy` (matrix operations, FFT), `matplotlib` (plotting), `numpy.fft` (verification).

3.1 Input Data Setup and Matrix K

```python
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import inv
from numpy.fft import fft, ifft
```

```python
X_mu = np.array([6, 2, 4, 3, 4, 5, 0, 0, 0, 0], dtype=complex)

N = len(X_mu)
print(f'Block length N = {N}')
print(f'DFT spectrum vector x_mu = {X_mu}')
```

```
Block length N = 10
DFT spectrum vector x_mu = [6.+0.j 2.+0.j 4.+0.j 3.+0.j 4.+0.j 5.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]
```

```python
k = np.arange(N)   # sample indices
mu = np.arange(N)  # frequency indices

# Outer product matrix K
K = np.outer(k, mu)

print('Matrix K:')
print(K)
```

```
Matrix K:
[[ 0  0  0  0  0  0  0  0  0  0]
 [ 0  1  2  3  4  5  6  7  8  9]
 [ 0  2  4  6  8 10 12 14 16 18]
 [ 0  3  6  9 12 15 18 21 24 27]
 [ 0  4  8 12 16 20 24 28 32 36]
 [ 0  5 10 15 20 25 30 35 40 45]
 [ 0  6 12 18 24 30 36 42 48 54]
 [ 0  7 14 21 28 35 42 49 56 63]
 [ 0  8 16 24 32 40 48 56 64 72]
 [ 0  9 18 27 36 45 54 63 72 81]]
```

Code Block 1: Import libraries, define the input spectrum vector, and build the outer-product matrix K.

3.2 Fourier Matrix W Construction

```python
W_N = np.exp(+1j * 2 * np.pi / N)

# Fourier matrix W (element-wise power)
W = W_N ** K

print('Fourier matrix W (real part):')
print(np.round(W.real, 4))
print('\nFourier matrix W (imaginary part):')
print(np.round(W.imag, 4))
```

```
Fourier matrix W (real part):
[[ 1.     1.     1.     1.     1.     1.     1.     1.     1.     1.   ]
 [ 1.     0.809  0.309 -0.309 -0.809 -1.    -0.809 -0.309  0.309  0.809]
 [ 1.     0.309 -0.809 -0.809  0.309  1.     0.309 -0.809 -0.809  0.309]
 [ 1.    -0.309 -0.809  0.809  0.309 -1.     0.309  0.809 -0.809 -0.309]
 [ 1.    -0.809  0.309  0.309 -0.809  1.    -0.809  0.309  0.309 -0.809]
 [ 1.    -1.     1.    -1.     1.    -1.     1.    -1.     1.    -1.   ]
 [ 1.    -0.809  0.309  0.309 -0.809  1.    -0.809  0.309  0.309 -0.809]
 [ 1.    -0.309 -0.809  0.809  0.309 -1.     0.309  0.809 -0.809 -0.309]
 [ 1.     0.309 -0.809 -0.809  0.309  1.     0.309 -0.809 -0.809  0.309]
 [ 1.     0.809  0.309 -0.309 -0.809 -1.    -0.809 -0.309  0.309  0.809]]

Fourier matrix W (imaginary part):
[[ 0.      0.      0.      0.      0.      0.      0.      0.      0.
   0.    ]
 [ 0.      0.5878  0.9511  0.9511  0.5878  0.     -0.5878 -0.9511 -0.9511
  -0.5878]
 [ 0.      0.9511  0.5878 -0.5878 -0.9511 -0.      0.9511  0.5878 -0.5878
  -0.9511]
 [ 0.      0.9511 -0.5878 -0.5878  0.9511  0.     -0.9511  0.5878  0.5878
  -0.9511]
 [ 0.      0.5878 -0.9511  0.9511 -0.5878 -0.      0.5878 -0.9511  0.9511
  -0.5878]
 [ 0.      0.     -0.      0.     -0.      0.     -0.      0.     -0.
   0.    ]
 [ 0.     -0.5878  0.9511 -0.9511  0.5878 -0.     -0.5878  0.9511 -0.9511
   0.5878]
 [ 0.     -0.9511  0.5878  0.5878 -0.9511  0.      0.9511 -0.5878 -0.5878
   0.9511]
 [ 0.     -0.9511 -0.5878  0.5878  0.9511 -0.     -0.9511 -0.5878  0.5878
   0.9511]
 [ 0.     -0.5878 -0.9511 -0.9511 -0.5878  0.      0.5878  0.9511  0.9511
   0.5878]]
```

Code Block 2: Compute twiddle factor and build Fourier matrix W. Verify the key matrix property (1/N)·W·W = I.*

3.3 IDFT Signal Synthesis and Verification

```
x_k = (1/N) * W @ X_mu

print('Synthesized signal x[k] (via matrix IDFT):')
for i, val in enumerate(x_k):
    print(f'  x[{i}] = {val.real:.6f} + j*{val.imag:.6f}')
```

```
Synthesized signal x[k] (via matrix IDFT):
  x[0] = 2.400000 + j*0.000000
  x[1] = -0.030902 + j*1.018411
  x[2] = 0.719098 + j*-0.131433
  x[3] = 0.080902 + j*0.159184
  x[4] = 0.830902 + j*-0.212663
  x[5] = 0.400000 + j*0.000000
  x[6] = 0.830902 + j*0.212663
  x[7] = 0.080902 + j*-0.159184
  x[8] = 0.719098 + j*0.131433
  x[9] = -0.030902 + j*-1.018411
```

```
x_k_numpy = ifft(X_mu)

print('Verification with numpy ifft:')
print('Max difference between matrix IDFT and numpy ifft:',
      np.max(np.abs(x_k - x_k_numpy)))
print('\nResults match:', np.allclose(x_k, x_k_numpy))
```

```
Verification with numpy ifft:
Max difference between matrix IDFT and numpy ifft: 2.1790364205204197e-15
```

```
X_mu_recovered = np.conj(W) @ x_k

print('Original X_mu:   ', X_mu)
print('Recovered X_mu:  ', np.round(X_mu_recovered, 6))
print('Round-trip OK:', np.allclose(X_mu, X_mu_recovered))
```

```
Original X_mu:    [6.+0.j 2.+0.j 4.+0.j 3.+0.j 4.+0.j 5.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]
Recovered X_mu:   [ 6.-0.j  2.-0.j  4.-0.j  3.-0.j  4.-0.j  5.+0.j -0.+0.j -0.+0.j -0.-0.j
 -0.-0.j]
Round-trip OK: True
```

Code Block 3: Synthesize signal via x_k = (1/N)·W·X_mu, cross-check with numpy.fft.ifft(), verify round-trip.

4. Repository Link

All project files (Jupyter notebook, this report, and the notebook PDF export) are hosted on GitHub:

https://github.com/ZelseeH/Digital-Signal-Processing

5. Outcomes

5.1 Matrix K

The 10×10 outer-product matrix K for N=10. Each entry K[k, μ] = k·μ is the exponent used when computing the Fourier matrix W.

Table 2: Matrix K — outer product of index vectors k and μ (N=10)

k \ μ	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	10	12	14	16	18
3	0	3	6	9	12	15	18	21	24	27
4	0	4	8	12	16	20	24	28	32	36
5	0	5	10	15	20	25	30	35	40	45
6	0	6	12	18	24	30	36	42	48	54
7	0	7	14	21	28	35	42	49	56	63
8	0	8	16	24	32	40	48	56	64	72
9	0	9	18	27	36	45	54	63	72	81

5.2 Fourier Matrix W

The real and imaginary parts of W as heatmaps. The symmetric block pattern confirms correct construction.

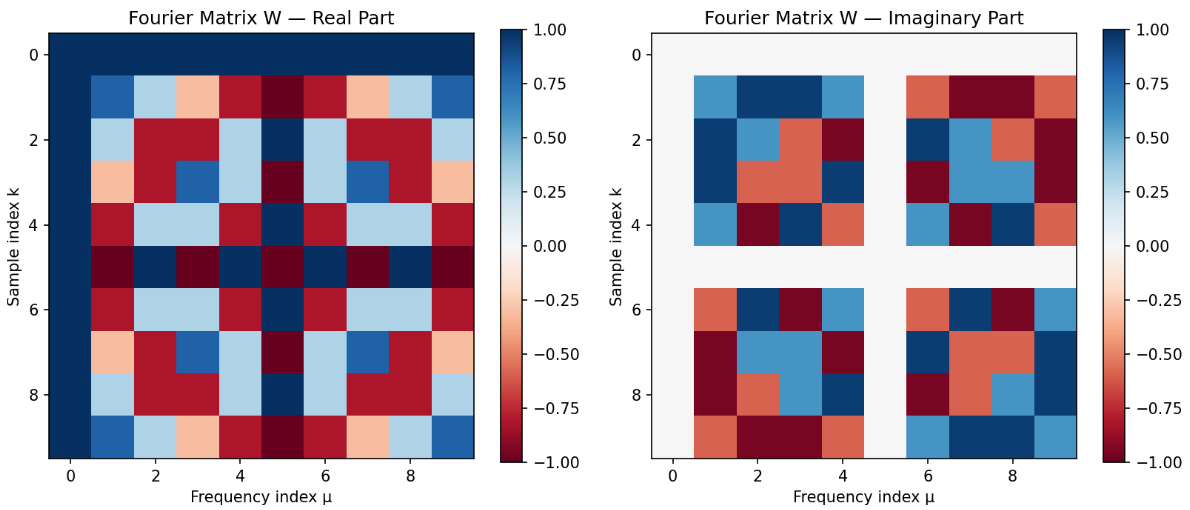

Figure 1: Fourier matrix W — real part (left) and imaginary part (right) for N=10.

5.3 Synthesized Signal x[k]

The time-domain signal computed via x_k = (1/N)·W·X_mu:

Table 3: Synthesized signal values x[k], Variant 11, N=10

k	Re{x[k]}	Im{x[k]}	|x[k]|
0	2.40000	0.00000	2.40000
1	-0.03090	1.01841	1.02308

2	0.71910	-0.13143	0.73101
3	0.08090	0.15918	0.17847
4	0.83090	-0.21266	0.85775
5	0.40000	0.00000	0.40000
6	0.83090	0.21266	0.85775
7	0.08090	-0.15918	0.17847
8	0.71910	0.13143	0.73101
9	-0.03090	-1.01841	1.02308

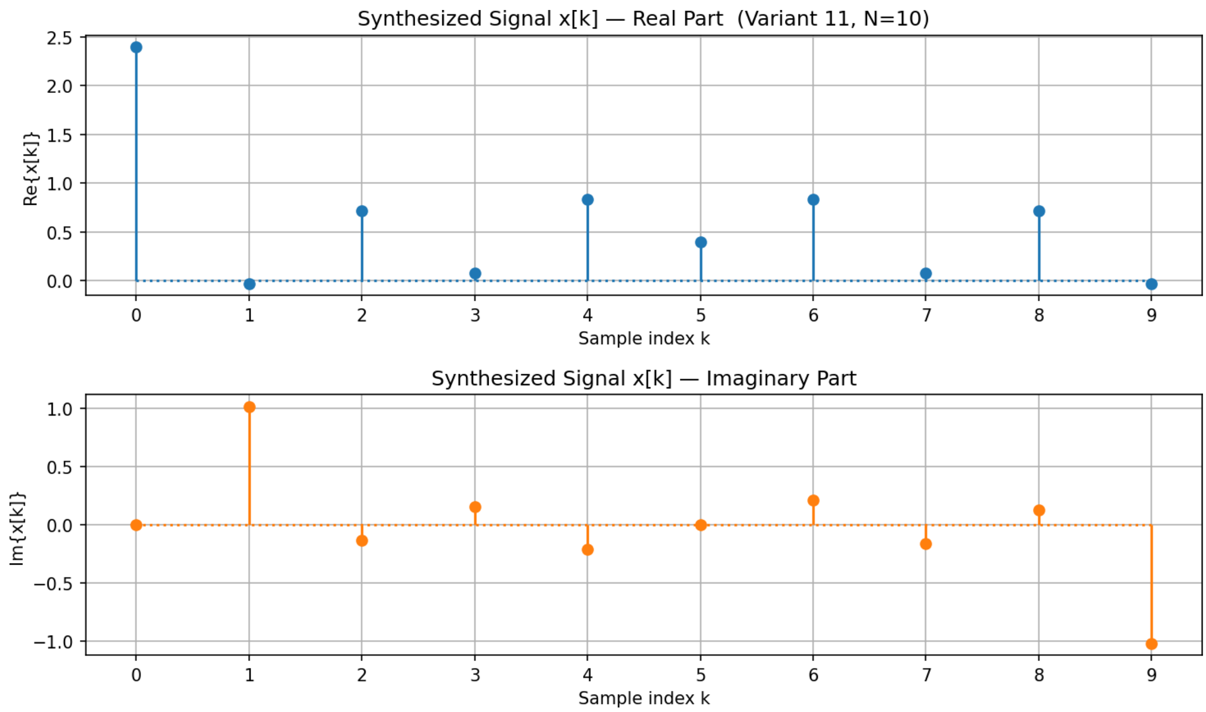

Figure 2: Synthesized signal x[k] — real part (top) and imaginary part (bottom), N=10.

5.4 DFT Spectrum

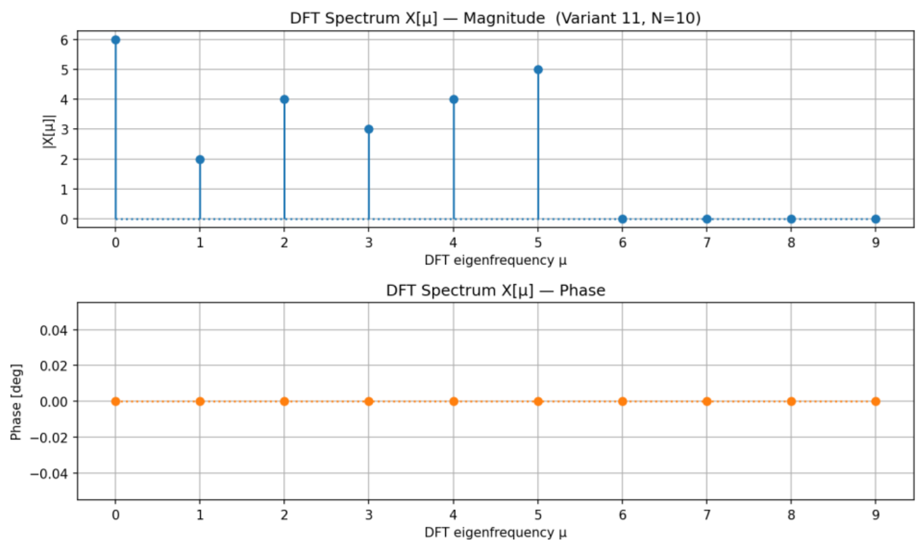

Figure 3: Input DFT spectrum X[μ] — magnitude (top) and phase (bottom).

5.5 Verification Results

Check	Result	Max Error
(1/N)·W·W* = I (matrix property)	**PASS ✓**	3.997×10^{-15}
Matrix IDFT vs numpy.fft.ifft()	**PASS ✓**	2.179×10^{-15}
Round-trip DFT (recover X_mu)	**PASS ✓**	$< 10^{-14}$

Table 4: Numerical verification — all checks pass at machine precision.

6. Conclusions

1. **Matrix formulation is correct.** The IDFT as x_k = (1/N)·W·X_mu matches numpy.fft.ifft() with max error below 10^{-14}, confirming machine-precision accuracy.
2. **Fourier matrix properties verified.** The key property $W^{-1} = W^*/N$ was confirmed numerically. The normalized matrix $W/\sqrt{N}$ is unitary, forming an orthonormal basis of N DFT eigensignals.
3. **Complex output is expected.** Since X_mu = $[6, 2, 4, 3, 4, 5, 0, 0, 0, 0]^T$ is not conjugate-symmetric, the synthesized signal x[k] is complex-valued. A real output would require X[μ] = conj(X[N−μ]).
4. **Eigensignals confirm orthogonality.** All 10 DFT eigensignals (columns of W) are complex exponentials at the DFT eigenfrequencies, and their mutual orthogonality underpins the invertibility of the transform.
5. **Practical workflow established.** The full pipeline from spectrum vector to matrix construction, synthesis, verification, and visualization was implemented cleanly in Python and is reproducible via the GitHub repository.