REPORT

Zajęcia: Analog and digital electronic circuits

Teacher: prof. dr hab. Vasyl Martsenyuk

Lab 1

Date: 23.02.2024

Topic: "Spectral Analysis of Deterministic Signals"

Variant: 13

Agnieszka Białecka

Informatyka II stopień,

stacjonarne,

2 semestr,

Gr.1a

1. Problem statement

The objective is to use discrete Fourier transform and its implementation with the help of matrix multiplication.

2. Theoretical introduction

2.1. Input Signal

Let us first define a "complex-valued signal" *x[k]* of a certain block length *N* ranging from $0 \leq k \leq N-1$. We will now perform an DFT of *x[k]* since we are interested in the frequency spectrum of it.

2.2. DFT Definition

The discrete Fourier transform pair for a discrete-time signal *x[k]* with sample index *k* and the corresponding DFT spectrum *X[μ]* with frequency index *μ* is given as

$$\mathrm{DFT}: X[\mu] = \sum_{k=0}^{N-1} x[k] \cdot \mathrm{e}^{-\mathrm{j}\frac{2\pi}{N}k\mu}$$

$$\mathrm{IDFT}: x[k] = \frac{1}{N} \sum_{\mu=0}^{N-1} X[\mu] \cdot \mathrm{e}^{+\mathrm{j}\frac{2\pi}{N}k\mu}$$

2.3. DFT and IDFT with Matrix Multiplication

Now we do a little better: We should think of the DFT/IDFT in terms of a matrix operation setting up a set of linear equations.

For that we define a column vector containing the samples of the discretetime signal *x[k]*:

$$\mathbf{x}_k = (x[k=0], x[k=1], x[k=2], \ldots, x[k=N-1])^{\mathrm{T}}$$

and a column vector containing the DFT coefficients *X[μ]*

$$\mathbf{x}_\mu = (X[\mu=0], X[\mu=1], X[\mu=2], \ldots, X[\mu=N-1])^{\mathrm{T}}$$

Then, the matrix operations

$$\mathrm{DFT}: \mathbf{x}_\mu = \mathbf{W}^* \mathbf{x}_k$$

$$\mathrm{IDFT}: \mathbf{x}_k = \frac{1}{N} \mathbf{W} \mathbf{x}_\mu$$

$()^T$ is the transpose, $()^*$ is the conjugate complex.

The *N* × *N* Fourier matrix is defined as (element-wise operation ⊙)

$$\mathbf{W} = \mathrm{e}^{+\mathrm{j}\frac{2\pi}{N}\odot\mathbf{K}}$$

using the so called twiddle factor (note that the sign in the exp() is our convention)

$$W_N = \mathrm{e}^{+\mathrm{j}\frac{2\pi}{N}}$$

and the outer product

$$\mathbf{K} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ \vdots \\ N-1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 2 & \cdots & N-1 \end{bmatrix}$$

containing all possible products *k μ* in a suitable arrangement.

For the simple case *N = 4* these matrices are

$$\mathbf{K} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 \\ 0 & 2 & 4 & 6 \\ 0 & 3 & 6 & 9 \end{bmatrix} \rightarrow \mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & +\mathrm{j} & -1 & -\mathrm{j} \\ 1 & -1 & 1 & -1 \\ 1 & -\mathrm{j} & -1 & +\mathrm{j} \end{bmatrix}$$

2.4. Fourier Matrix Properties

The DFT and IDFT basically solve two sets of linear equations, that are linked as forward and inverse problem.

This is revealed with the important property of the Fourier matrix

$$\mathbf{W}^{-1} = \frac{\mathbf{W}^{\mathrm{H}}}{N} = \frac{\mathbf{W}^{*}}{N},$$

the latter holds since the matrix is symmetric.

Thus, we see that by our convention, the DFT is the inverse problem (signal analysis) and the IDFT is the forward problem (signal synthesis)

$$\begin{aligned} \mathrm{DFT}\colon\ & \mathbf{x}_\mu = \mathbf{W}^{*}\mathbf{x}_k \rightarrow \mathbf{x}_\mu = N\mathbf{W}^{-1}\mathbf{x}_k \\ \mathrm{IDFT}\colon\ & \mathbf{x}_k = \frac{1}{N}\mathbf{W}\mathbf{x}_\mu. \end{aligned}$$

The occurrence of the *N*, *1/N* factor is due to the prevailing convention in signal processing literature.

If the matrix is normalised as $\frac{W}{\sqrt{N}}$, as so called unitary matrix results, for which the important property

$$(\frac{\mathbf{W}}{\sqrt{N}})^{\mathrm{H}}(\frac{\mathbf{W}}{\sqrt{N}}) = \mathbf{I} = (\frac{\mathbf{W}}{\sqrt{N}})^{-1}(\frac{\mathbf{W}}{\sqrt{N}})$$

holds, i.e. the complex-conjugate, transpose is equal to the inverse $\left(\frac{W}{\sqrt{N}}\right)^{H} = \left(\frac{W}{\sqrt{N}}\right)^{-1}$ and due to the matrix symmetry also $\left(\frac{W}{\sqrt{N}}\right)^{*} = \left(\frac{W}{\sqrt{N}}\right)^{-1}$ is valid.

This tells that the matrix $\frac{W}{\sqrt{N}}$, is **orthonormal**, i.e. the matrix spans a orthonormal vector basis (the best what we can get in linear algebra world to work with) of N normalized DFT eigensignals.

So, DFT and IDFT is transforming vectors into other vectors using the vector basis of the Fourier matrix.

2.5. Check DFT Eigensignals and -Frequencies

The columns of the Fourier matrix W contain the eigensignals of the DFT. These are

$$w_{\mu}[k] = \cos(\frac{2\pi}{N}k\mu) + \mathrm{j}\sin(\frac{2\pi}{N}k\mu)$$

since we have intentionally set up the matrix this way.

The eigensignals for $0 \leq \mu \leq N-1$ therefore exhibit a certain digital frequency, the so called DFT eigenfrequencies.

The nice thing about the chosen eigenfrequencies, is that the eigensignals are "orthogonal".

This choice of the vector basis is on purpose and one of the most important ones in linear algebra and signal processing.

We might for example check orthogonality with the "complex" inner product of some matrix columns.

2.6. Initial Example: IDFT Signal Synthesis for N=8

Let us synthesize a discrete-time signal by using the IDFT in matrix notation for $N = 8$.

The signal should contain a DC value, the first and second eigenfrequency with different amplitudes, such as

$$\mathbf{x}_{\mu} = [8, 2, 4, 0, 0, 0, 0, 0]^{\mathrm{T}}$$

3. Input data (Variant)

This report was created with base of variant 13:

$$\mathbf{x}_{\mu} = [6, 2, 4, 3, 4, 5, 0, 0, 0, 0]^{\mathrm{T}}$$

GitHub repository:

4. Course of actions

The first step was to upload the libraries that were used during the implementation of the code.

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

Picture 1. Uploaded libraries

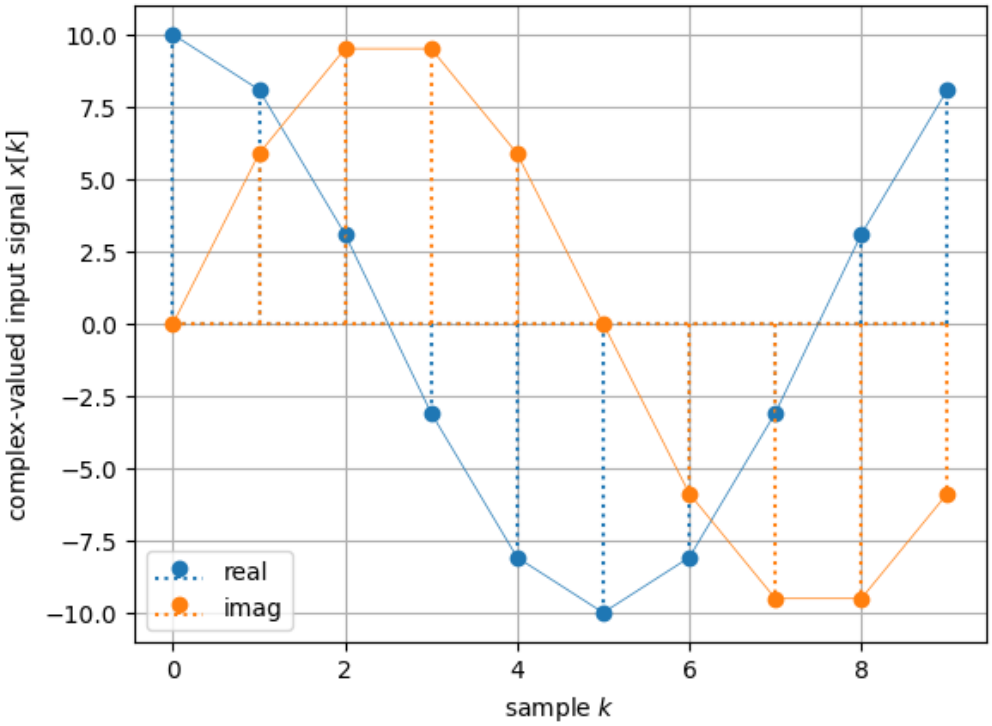
4.1. Input Signal

The next step was to define a "complex-valued signal" *x[k]* of a certain block length *N* ranging from $0 \leq k \leq N-1$, which can be seen below.

```
N = 10  # signal block length
k = np.arange(N)  # all required sample/time indices
A = 10  # signal amplitude

tmpmu = 2-1/2  # DFT eigenfrequency worst case
tmpmu = 1  # DFT eigenfrequency best case

x = A * np.exp(tmpmu * +1j*2*np.pi/N * k)

# plot
plt.stem(k, np.real(x), markerfmt='C0o',
         basefmt='C0:', linefmt='C0:', label='real')
plt.stem(k, np.imag(x), markerfmt='C1o',
         basefmt='C1:', linefmt='C1:', label='imag')
plt.plot(k, np.real(x), 'C0-', lw=0.5)
plt.plot(k, np.imag(x), 'C1-', lw=0.5)
plt.xlabel(r'sample $k$')
plt.ylabel(r'complex-valued input signal $x[k]$')
plt.legend()
plt.grid(True)
```

Picture 2. Implemented code

The variable "*tmpmu*" defines the frequency of the signal. For now, the *tmpmu=1*. This results in exactly one period of cosine and sine building the complex signal. If *tmpmu=2* we would get exactly two periods of *cos/sin*.



Picture 3. The result

4.2. DFT and IDFT with For-Loops

Another step was the implementation of the DFT and IDFT with for-loop handling. The DFT was created with an outer for-loop iterating over μ and an inner for loop summing over all k for a specific μ. A variable with _ subscript was used to maintain clear and organized variable names for matrix-based calculations.

```
# DFT with for-loop:
X_ = np.zeros((N, 1), dtype=complex)  # alloc RAM, init with zeros
for mu_ in range(N):  # do for all DFT frequency indices
    for k_ in range(N):  # do for all sample indices
        X_[mu_] += x[k_] * np.exp(-1j*2*np.pi/N*k_*mu_)
```

```
# IDFT with for-loop:
x_ = np.zeros((N, 1), dtype=complex)  # alloc RAM, init with zeros
for k_ in range(N):
    for mu_ in range(N):
        x_[k_] += X_[mu_] * np.exp(+1j*2*np.pi/N*k_*mu_)
x_ *= 1/N  # normalization in the IDFT stage
```

Picture 4. Implemented code

4.3. DFT and IDFT with Matrix Multiplication

The next step was the implementation of the DFT/IDFT in terms of a matrix operation setting up a set of linear equations.

```
# k = np.arange(N)  # all required sample/time indices, already defined above
# all required DFT frequency indices, actually same entries like in k
mu = np.arange(N)

# set up matrices
K = np.outer(k, mu)  # get all possible entries k*mu in meaningful arrangement
W = np.exp(+1j * 2*np.pi/N * K)  # analysis matrix for DFT
```

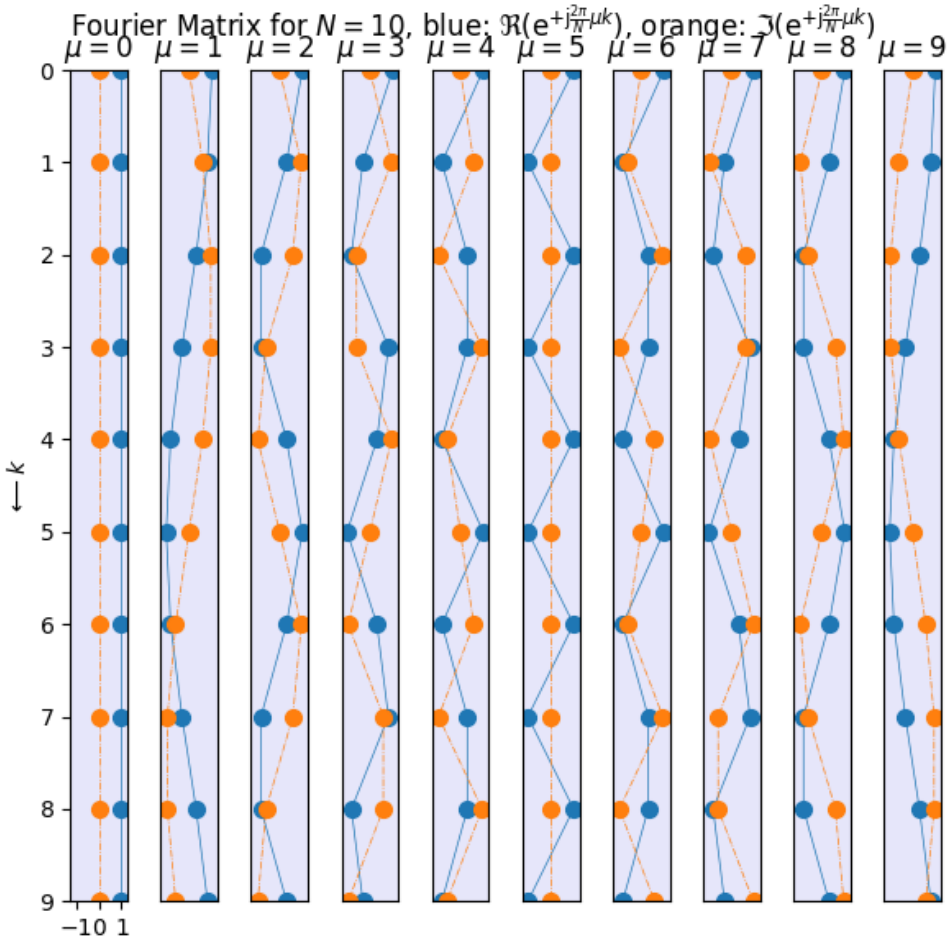
Picture 5. Implemented code

```
# visualize the content of the Fourier matrix
# we've already set up (use other N if desired):
# N = 10
# k = np.arange(N)
# mu = np.arange(N)
# W = np.exp(+1j*2*np.pi/N*np.outer(k, mu))  # set up Fourier matrix

fig, ax = plt.subplots(1, N)
fig.set_size_inches(6, 6)
fig.suptitle(
    r'Fourier Matrix for $N=$%d, blue: $\Re(\mathrm{e}^{+\mathrm{j} \frac{2\pi}{N} \mu k})$, orange: $\Im(\mathrm{e}^{+\mathrm{j} \frac{2\pi}{N} \mu k})$' % N)
```

Picture 6. Implemented code – continued

```python
for tmp in range(N):
    ax[tmp].set_facecolor('lavender')
    ax[tmp].plot(W[:, tmp].real, k, 'C0o-', ms=7, lw=0.5)
    ax[tmp].plot(W[:, tmp].imag, k, 'C1o-.', ms=7, lw=0.5)
    ax[tmp].set_ylim(N-1, 0)
    ax[tmp].set_xlim(-5/4, +5/4)
    if tmp == 0:
        ax[tmp].set_yticks(np.arange(0, N))
        ax[tmp].set_xticks(np.arange(-1, 1+1, 1))
        ax[tmp].set_ylabel(r'$\longleftarrow k$')
    else:
        ax[tmp].set_yticks([], minor=False)
        ax[tmp].set_xticks([], minor=False)
    ax[tmp].set_title(r'$\mu=$%d' % tmp)
fig.tight_layout()
fig.subplots_adjust(top=0.91)

fig.savefig('fourier_matrix.png', dpi=300)

# TBD: row version for analysis
```

Picture 7. Implemented code- continued



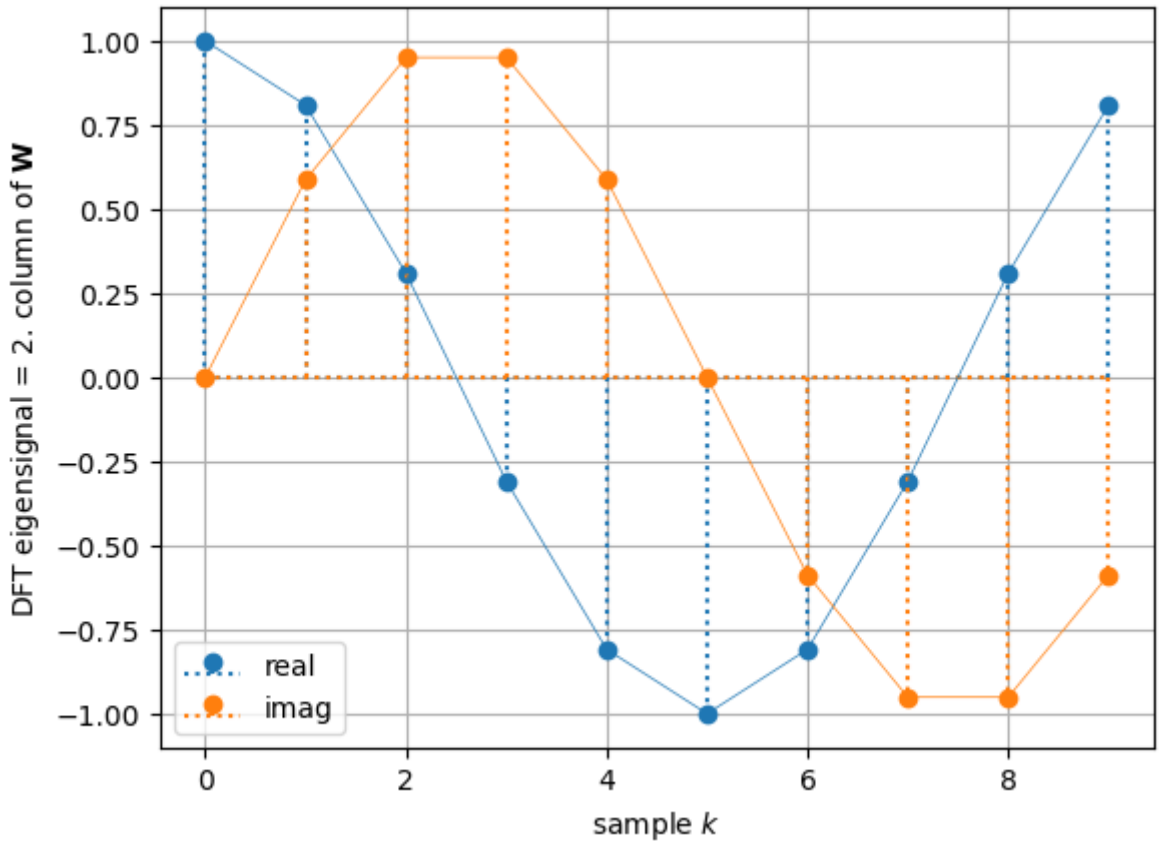
Picture 8. The result

4.4. Check DFT Eigensignals and -Frequencies

Another step was the implementation of the Fourier matrix. The plot below shows the eigensignal for $\mu = 1$, which fits again one signal period in the block length N. For $\mu = 2$ there should be two periods in one block.

```
tmpmu = 1  # column index

plt.stem(k, np.real(W[:, tmpmu]), label='real',
         markerfmt='C0o', basefmt='C0:', linefmt='C0:')
plt.stem(k, np.imag(W[:, tmpmu]), label='imag',
         markerfmt='C1o', basefmt='C1:', linefmt='C1:')
# note that connecting the samples by lines is actually wrong, we
# use it anyway for more visual convenience
plt.plot(k, np.real(W[:, tmpmu]), 'C0-', lw=0.5)
plt.plot(k, np.imag(W[:, tmpmu]), 'C1-', lw=0.5)
plt.xlabel(r'sample $k$')
plt.ylabel(r'DFT eigensignal = '+str(tmpmu+1)+'. column of $\mathbf{W}$')
plt.legend()
plt.grid(True)
```

Picture 9. Implemented code



Picture 10. The result

4.5. Initial Example: IDFT Signal Synthesis for N=10

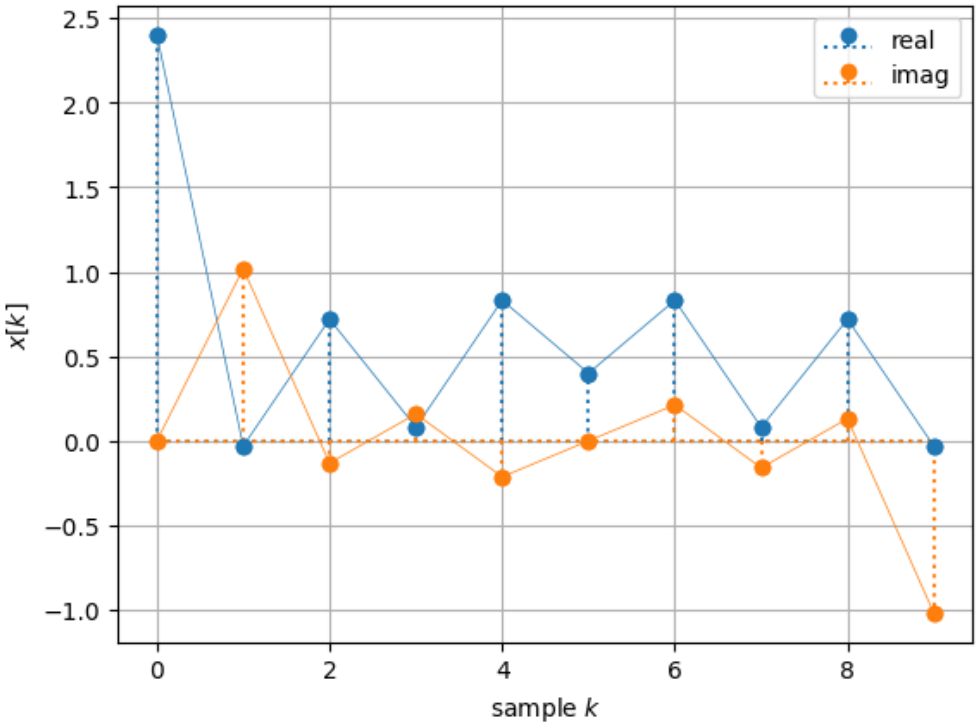
The next step was to synthesize a discrete-time signal by using the IDFT in matrix notation for *N=10.* A variable *„X_test"* represents a signal which was specified in the variant of task. In this case it was: $x_\mu = [6, 2, 4, 3, 4, 5, 0, 0, 0, 0]^T$.

```
if N == 10:
    X_test = np.array([6, 2, 4, 3, 4, 5, 0, 0, 0, 0])
    # x_test = 1/N*W@X_test  # >= Python3.5
    x_test = 1/N * np.matmul(W, X_test)

    plt.stem(k, np.real(x_test), label='real',
             markerfmt='C0o', basefmt='C0:', linefmt='C0:')
    plt.stem(k, np.imag(x_test), label='imag',
             markerfmt='C1o', basefmt='C1:', linefmt='C1:')
    # note that connecting the samples by lines is actually wrong, we
    # use it anyway for more visual convenience
    plt.plot(k, np.real(x_test), 'C0o-', lw=0.5)
    plt.plot(k, np.imag(x_test), 'C1o-', lw=0.5)
    plt.xlabel(r'sample $k$')
    plt.ylabel(r'$x[k]$')
    plt.legend()
    plt.grid(True)

    # check if results are identical with numpy ifft package
    print(np.allclose(ifft(X_test), x_test))
    print('DC is 1 as expected: ', np.mean(x_test))
```

Picture 11. Implemented code

```
True
DC is 1 as expected:  (0.6+6.66133814775094e-17j)
```

Picture 12. The result



Picture 13. The result - continued

Above we can see a linear combination of the Fourier matrix columns, which are the DFT eigensignals.

4.6. Initial Example: DFT Spectrum Analysis for N=10

Another step was to calculate the DFT of the signal *"x_test"*. The DFT vector was to be expected to be a result of calculation.

```
if N == 10:
    # X_test2 = np.conj(W)@x_test  # >= Python3.5
    X_test2 = np.matmul(np.conj(W), x_test)  # DFT, i.e. analysis
    print(np.allclose(X_test, X_test2))  # check with result before
```

```
True
```

Picture 14. Implemented code and the result

We can see that everything was alright, but it is advisable also to check against the numpy.fft implementation.

```
if N == 10:
    print(np.allclose(fft(x_test), X_test))
```

```
True
```

Picture 15. Implemented code and the result

From the picture 15 we can say that all results produce the same output.

```
if N == 10:
    print(np.conj(W[:, 0])@x_test)
    print(np.conj(W[:, 1])@x_test)
    print(np.conj(W[:, 2])@x_test)
```

```
(6+6.661338147750939e-16j)
(1.9999999999999998+4.440892098500626e-16j)
(3.9999999999999996+0j)
```

Picture 16. Implemented code and the result

5. Conclusions

Based on this lab we learn about DFT and IDFT, which decompose and reconstruct signals in the frequency domain. The Fourier Matrix serves as the basis for these transformations, exhibiting properties like orthonormality and connections to eigensignals and eigenfrequencies. DFT was also interpreted as a measure of correlation between the signal and different frequency components.