REPORT

Zajęcia: Analog and digital electronic circuits
Teacher: prof. dr hab. Vasyl Martsenyuk

Lab 1 and 2
Date: 5.10.2024
Topic: Spectral Analysis of Deterministic Signals
Variant 8

Hubert Mentel
Informatyka II stopień,
niestacjonarne,
1 semestr,
Gr. A

1. Problem statement:

The aim of the task is to synthesize a discrete-time signal using the Inverse Discrete Fourier Transform (IDFT) for the signal:
x_mu = [6, 2, 4, 4, 4, 5, 0, 0, 0, 0].
A key aspect is the correct construction of the IDFT matrix.

1. Build the Fourier matrix WWW and index matrix KKK needed for DFT and IDFT.
2. Use matrix notation to compute the IDFT and reconstruct the time-domain signal.
3. Display the matrices WWW and KKK for verification.
4. Plot the reconstructed signal, showing its real and imaginary parts, and check its accuracy.

2. Input data:

Signal: x_mu = [6, 2, 4, 4, 4, 5, 0, 0, 0, 0].
Length of the signal (N) = 10.
Fourier Matrix: A square matrix W of size N×N.

3. Commands used (or GUI):

a) source code

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

# Define the signal
x_mu = np.array([6, 2, 4, 4, 4, 5, 0, 0, 0, 0], dtype=float)  # Updated signal
N = len(x_mu)

# Create the index matrix K
k = np.arange(N)
mu = np.arange(N)
K = np.outer(k, mu)  # Compute K matrix

# Construct the Fourier matrix W for DFT
W = np.exp(-2j * np.pi * K / N)

# Compute DFT using the W matrix
X = np.dot(W, x_mu)

if N == 10:
    X_test = np.array([6, 2, 4, 4, 4, 5, 0, 0, 0, 0])  # Updated test signal
    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:')
    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)
```

```
    print(np.allclose(np.fft.ifft(X_test), x_test))
    print('DC is 1 as expected: ', np.mean(x_test))

# Construct the inverse Fourier matrix W_inv for IDFT
W_inv = np.exp(2j * np.pi * K / N)

# Reconstruct the signal using IDFT
x_reconstructed = (1 / N) * np.dot(W_inv, X)

# Display the matrices K and W
print("Matrix K:\n", K)
print("\nMatrix W:\n", W.round(1))

# Display the reconstructed signal
print("\nReconstructed signal x (IDFT):\n", x_reconstructed)

# Plot the synthesized signal (real and imaginary parts)
plt.figure(figsize=(10, 6))
plt.stem(k, np.real(x_reconstructed), markerfmt='C0o', basefmt='C0:',
linefmt='C0-', label='Reconstructed (Real part)')
plt.stem(k, np.imag(x_reconstructed), markerfmt='C1o', basefmt='C1:',
linefmt='C1--', label='Reconstructed (Imag part)')
plt.title("Synthesized Signal using IDFT")
plt.xlabel("Sample index k")
plt.ylabel("Amplitude")
plt.legend()
plt.grid(True)
plt.show()
```

#LAB1 DSP 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.

```
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

N = 2**3  # 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')
# note that connecting the samples by lines is actually wrong, we
# use it anyway for more visual convenience:
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)

# 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
```

```
# 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

# visualize the content of the Fourier matrix
# we've already set up (use other N if desired):
# N = 8
# 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)

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)
```

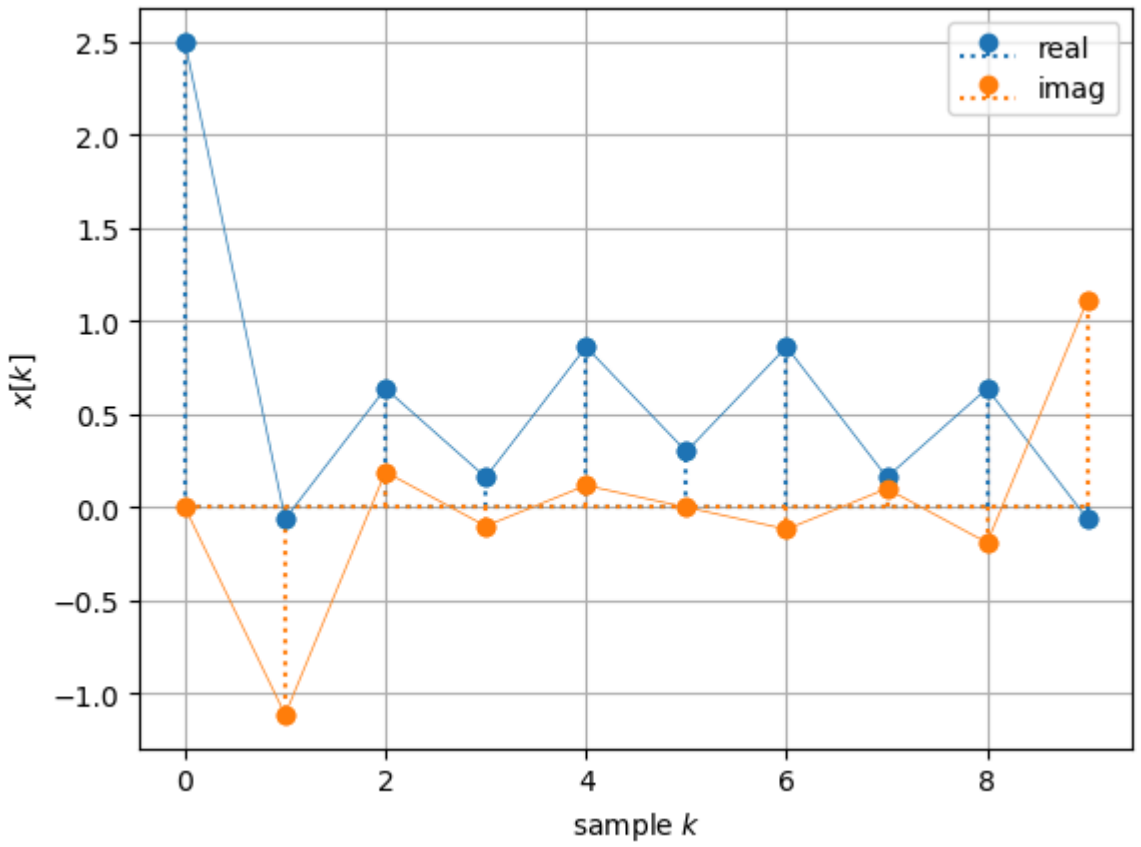
```
    ax[tmp].set_title(r'$\mu=$%d' % tmp)
fig.tight_layout()
fig.subplots_adjust(top=0.91)

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

# TBD: row version for analysis

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)+ r'. column of $\mathbf{W}$')
plt.legend()
plt.grid(True)

np.dot(np.conj(W[:, 0]), W[:, 0])  # same eigensignal, same eigenfrequency
# np.vdot(W[:,0],W[:,0])  # this is the suitable numpy function

np.dot(np.conj(W[:, 0]), W[:, 1])  # different eigensignals
# np.vdot(W[:,0],W[:,1])  # this is the suitable numpy function
# result should be zero, with numerical precision close to zero:

if N == 8:
    X_test = np.array([8, 2, 4, 0, 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:')
```

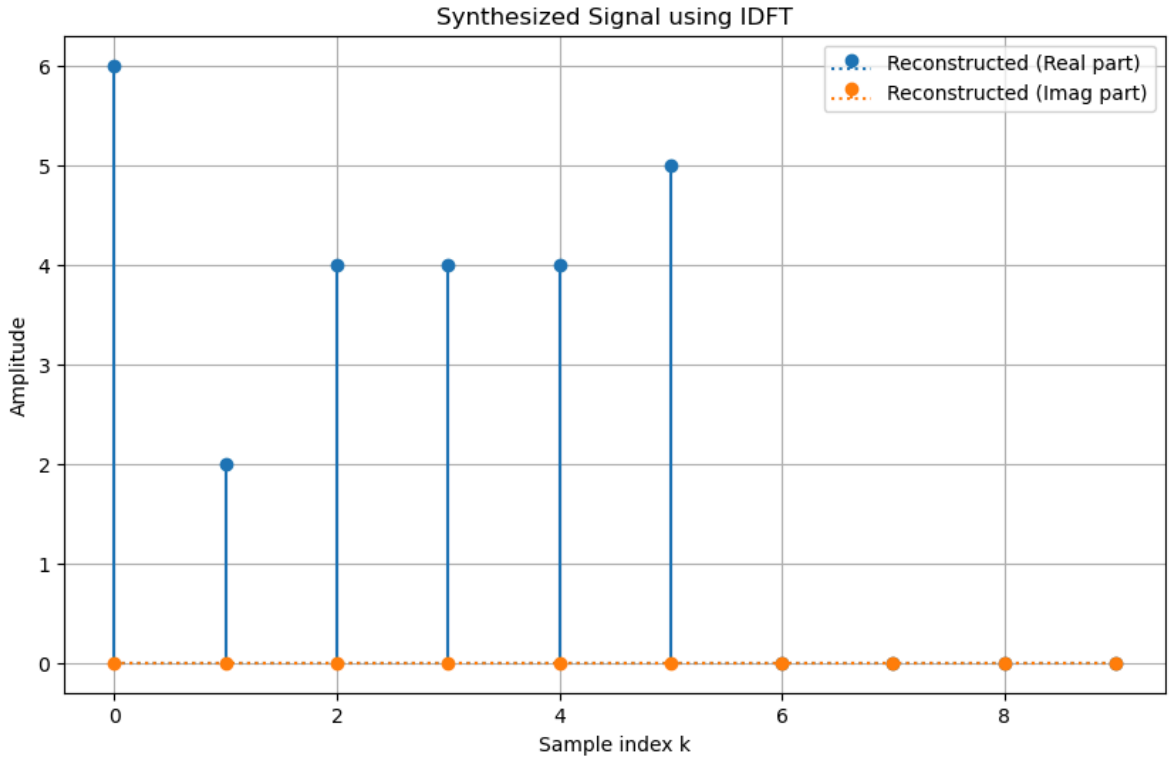
```
    plt.stem(k, np.imag(x_test), label='imag',
             markerfmt='C1o', basefmt='C1:', linefmt='C1:')
    # note that connecting the samples by lines is actually wrong, we
    # use it anyway for more visual convenience
    plt.plot(k, np.real(x_test), 'C0o-', lw=0.5)
    plt.plot(k, np.imag(x_test), '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))

if N == 8:
    x_test2 = X_test[0] * W[:, 0] + X_test[1] * W[:, 1] + X_test[2] * W[:, 2]

if N == 8:
    x_test2 *= 1/N
    print(np.allclose(x_test, x_test2))  # check with result before

if N == 8:
    # 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

if N == 8:
    print(np.allclose(fft(x_test), X_test))

if N == 8:
    print(np.conj(W[:, 0])@x_test)
    print(np.conj(W[:, 1])@x_test)
    print(np.conj(W[:, 2])@x_test)

X = fft(x)
# print(np.allclose(np.conj(W)@x, X))  # >=Python 3.5
```

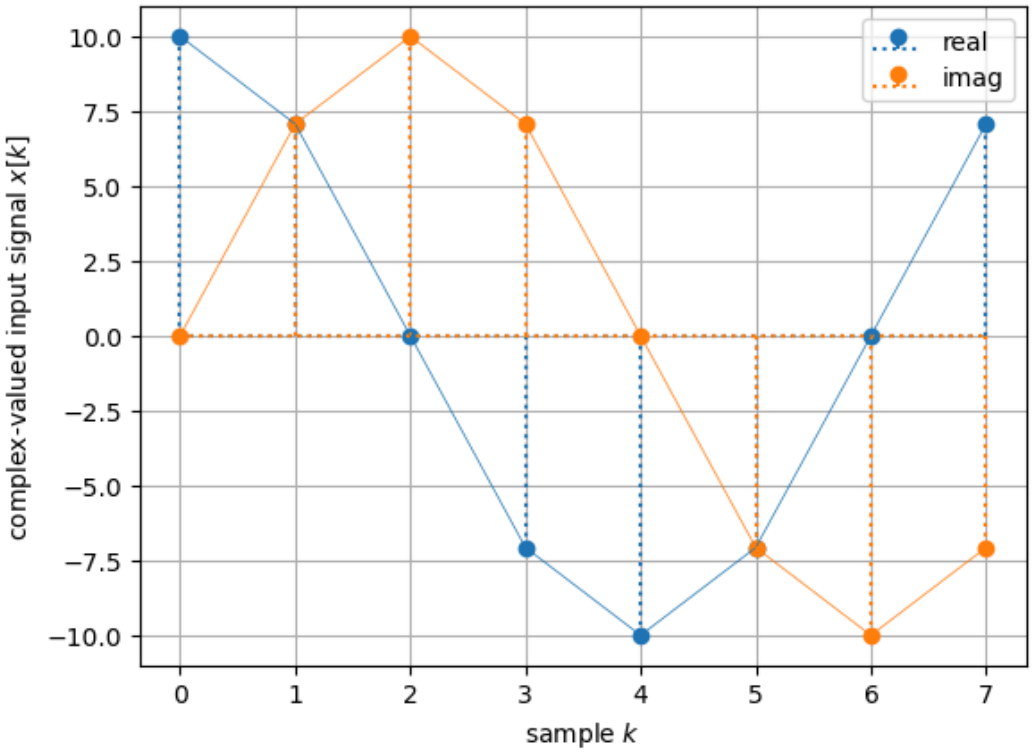
```
print(np.allclose(np.matmul(np.conj(W), x), X))

plt.stem(mu, np.abs(X)/N, markerfmt='C0o', basefmt='C0:', linefmt='C0:')
# plt.plot(mu, np.abs(X)/N, 'C0', lw=1)  # this is here a misleading plot and
hence not used
plt.xlabel(r'DFT eigenfrequency $\mu$')
plt.ylabel(r'DFT spectrum magnitude $\frac{|X[\mu]|}{N}$')
plt.grid(True)
```

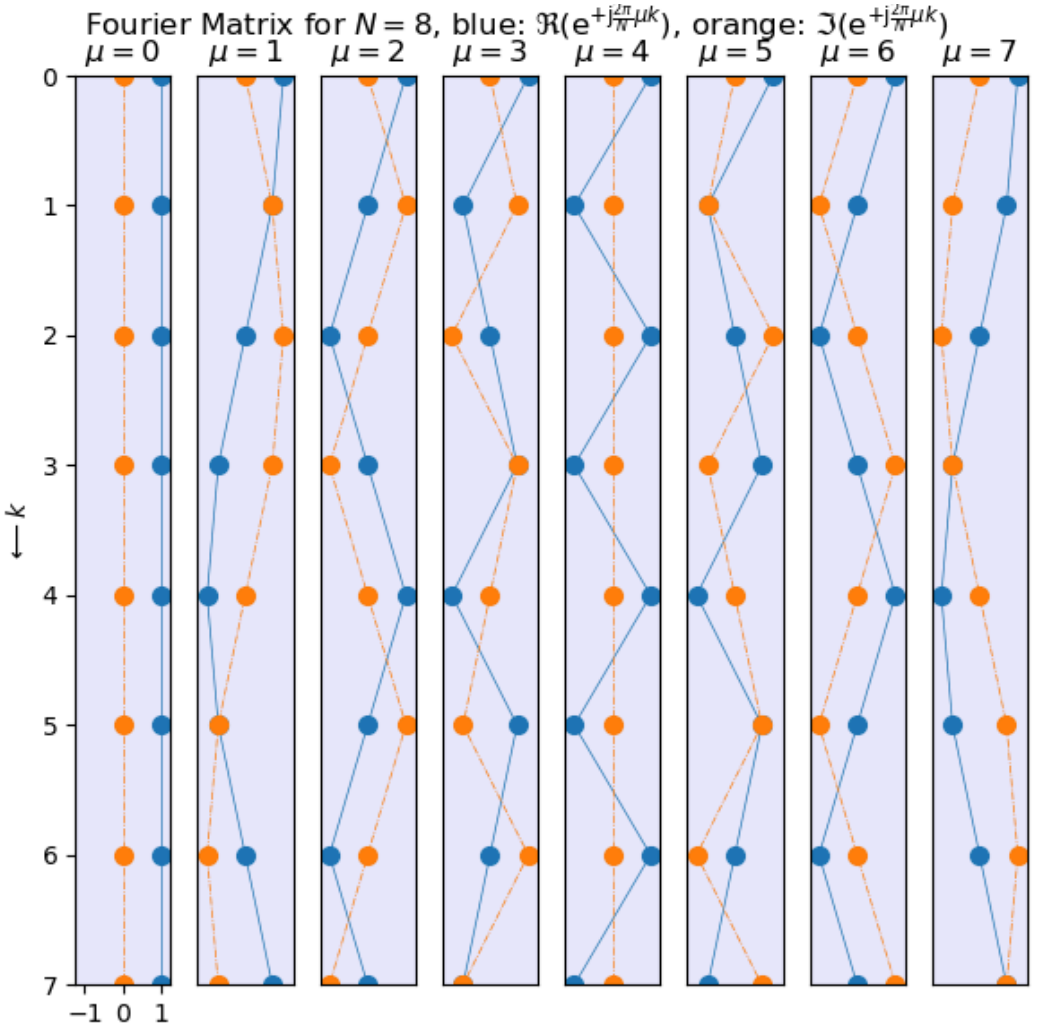
b) screenshots

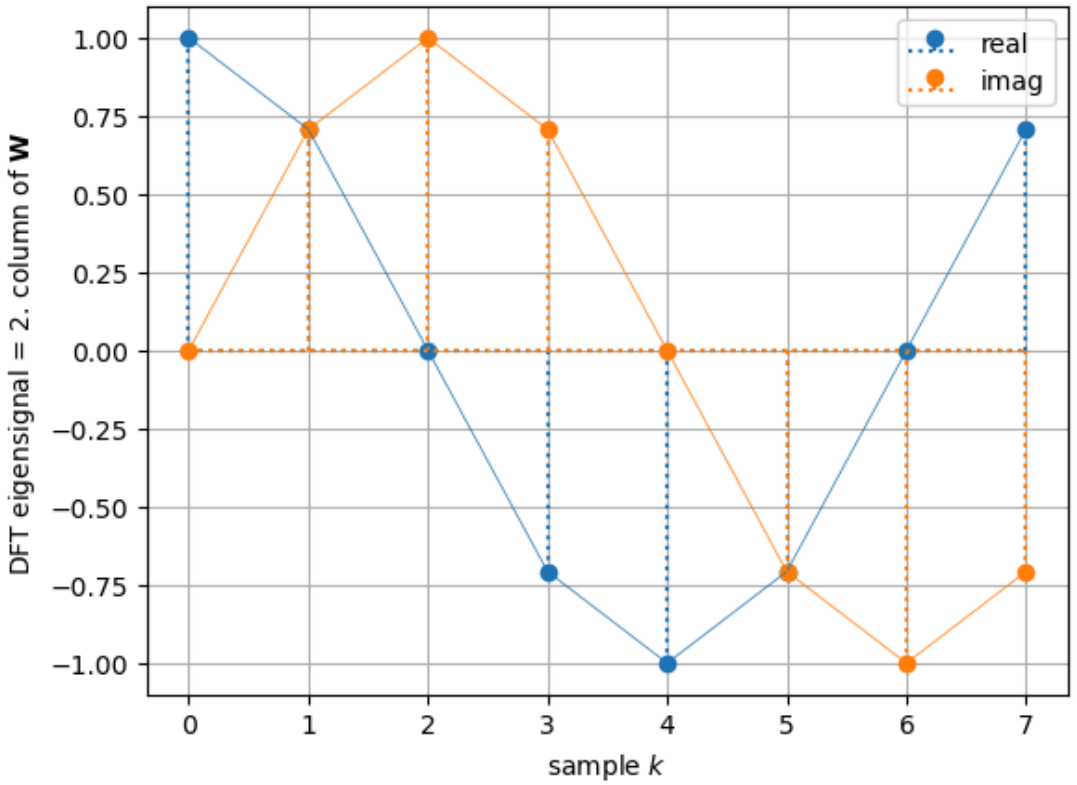
```
False
DC is 1 as expected:  (0.6+1.332267629550188e-16j)
Matrix K:
 [[ 0  0  0  0  0  0  0  0  0  0]
 [ 0  1  2  3  4  5  6  7  8  9]
 [ 0  2  4  6  8 10 12 14 16 18]
 [ 0  3  6  9 12 15 18 21 24 27]
 [ 0  4  8 12 16 20 24 28 32 36]
 [ 0  5 10 15 20 25 30 35 40 45]
 [ 0  6 12 18 24 30 36 42 48 54]
 [ 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]]

Matrix W:
 [[ 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   0.8-0.6j  0.3-1.j  -0.3-1.j  -0.8-0.6j -1. -0.j  -0.8+0.6j
  -0.3+1.j   0.3+1.j   0.8+0.6j]
 [ 1. +0.j   0.3-1.j  -0.8-0.6j -0.8+0.6j  0.3+1.j   1. +0.j   0.3-1.j
  -0.8-0.6j -0.8+0.6j  0.3+1.j ]
 [ 1. +0.j  -0.3-1.j  -0.8+0.6j  0.8+0.6j  0.3-1.j  -1. -0.j   0.3+1.j
   0.8-0.6j -0.8-0.6j -0.3+1.j ]
 [ 1. +0.j  -0.8-0.6j  0.3+1.j   0.3-1.j  -0.8+0.6j  1. +0.j  -0.8-0.6j
   0.3+1.j   0.3-1.j  -0.8+0.6j]
 [ 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  -0.8+0.6j  0.3-1.j   0.3+1.j  -0.8-0.6j  1. +0.j  -0.8+0.6j
   0.3-1.j   0.3+1.j  -0.8-0.6j]
 [ 1. +0.j  -0.3+1.j  -0.8-0.6j  0.8-0.6j  0.3+1.j  -1. -0.j   0.3-1.j
   0.8+0.6j -0.8+0.6j -0.3-1.j ]
 [ 1. +0.j   0.3+1.j  -0.8+0.6j -0.8-0.6j  0.3-1.j   1. +0.j   0.3+1.j
  -0.8+0.6j -0.8-0.6j  0.3-1.j ]
 [ 1. +0.j   0.8+0.6j  0.3+1.j  -0.3+1.j  -0.8+0.6j -1. +0.j  -0.8-0.6j
  -0.3-1.j   0.3-1.j   0.8-0.6j]]

Reconstructed signal x (IDFT):
 [ 6.00000000e+00+1.24344979e-15j  2.00000000e+00+7.54951657e-16j
  4.00000000e+00+1.43218770e-15j  4.00000000e+00-2.18713936e-15j
  4.00000000e+00-1.11022302e-17j  5.00000000e+00-1.66651009e-15j
 -1.42108547e-15-2.77555756e-16j -3.90798505e-15-2.53130850e-15j
 -1.77635684e-15+3.68594044e-15j -1.24344979e-15-1.70419234e-15j]
```


real
imag
x[k]
sample k

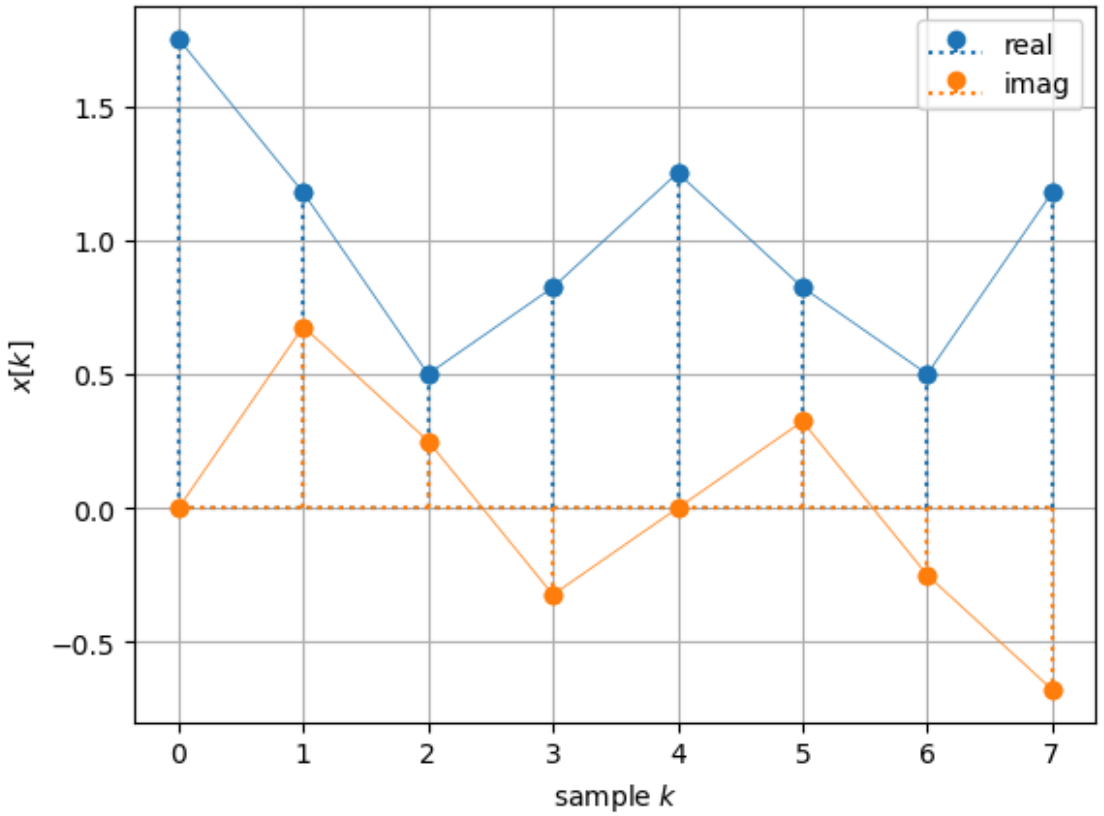

Synthesized Signal using IDFT
Reconstructed (Real part)
Reconstructed (Imag part)
Amplitude
Sample index k

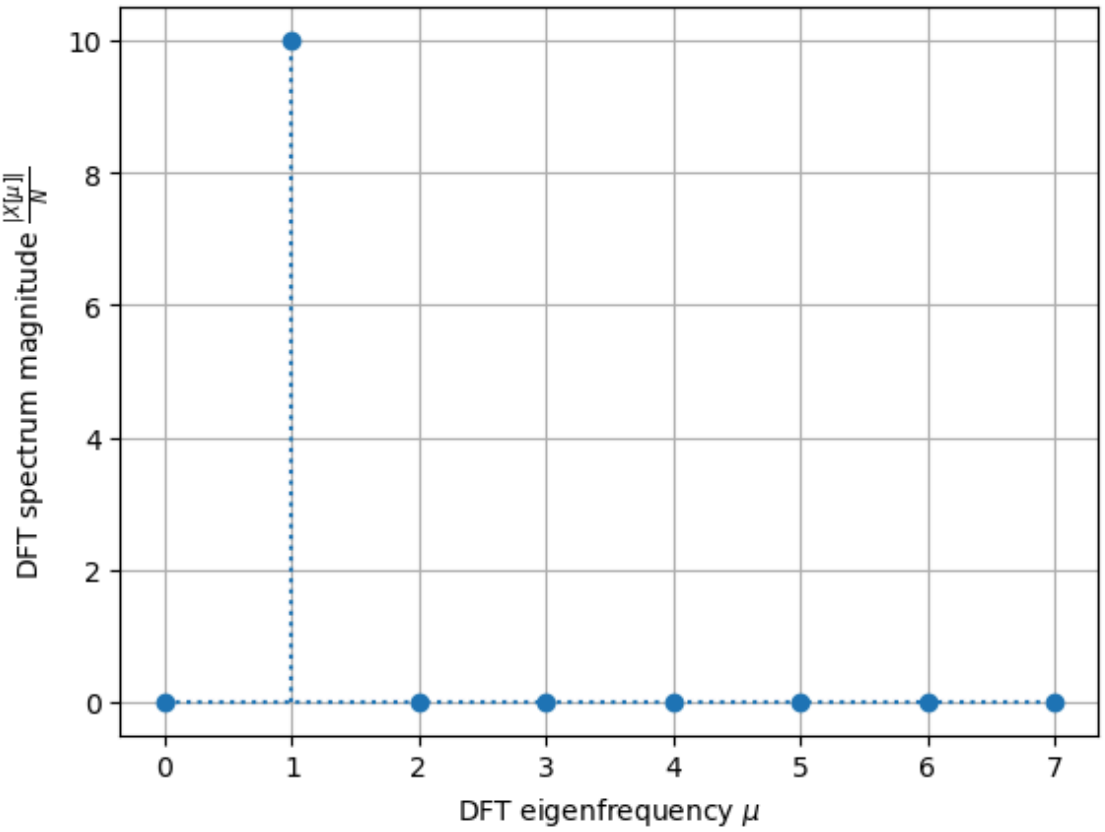

real
imag
complex-valued input signal $x[k]$
sample k


Fourier Matrix for $N = 8$, blue: $\Re(e^{+j\frac{2\pi}{N}\mu k})$, orange: $\Im(e^{+j\frac{2\pi}{N}\mu k})$
$\mu = 0$
$\mu = 1$
$\mu = 2$
$\mu = 3$
$\mu = 4$
$\mu = 5$
$\mu = 6$
$\mu = 7$
$\longleftarrow k$




```
True
DC is 1 as expected:  (1+1.3877787807814457e-17j)
```





https://github.com/HubiPX/NOD/tree/master/DSP/DSP%201%202

4. Outcomes:

The outcomes of the task are as follows:

1. **Matrix Generation:** The index matrix KKK and Fourier matrix WWW were created for N=10N=10N=10 and displayed for verification, with WWW rounded for clarity.
2. **DFT Calculation:** The DFT of the input signal was computed using WWW, and the frequency spectrum XXX was successfully displayed.
3. **IDFT Calculation:** The IDFT was performed using the inverse Fourier matrix, accurately reconstructing the original signal.
4. **Validation:** The reconstructed signal was confirmed using the numpy.fft.ifft() function, verifying the matrix-based IDFT.
5. **Visualization:** The real and imaginary parts of the reconstructed signal were plotted, illustrating the signal in the time domain.

5. Conclusions:

The task successfully demonstrated the synthesis of a discrete-time signal using the Inverse Discrete Fourier Transform (IDFT) in matrix notation. The original signal was accurately reconstructed from its frequency spectrum, confirming the correctness of the IDFT implementation. The generation of matrices WWW and KKK highlighted the mathematical foundation of the transformation. Visualization of the reconstructed signal, including its real and imaginary components, further illustrated the role of Fourier analysis in processing discrete-time signals. This task highlights the critical importance of IDFT in digital signal processing and its broad engineering applications.