EXPERIMENT 7 – EFFECTIVE COMPUTATION OF DFT:

CODE:

# -\*- coding: utf-8 -\*-

"""

Created on Tue Apr 16 11:20:58 2024

@author: sanja

"""

import numpy as np

import matplotlib.pyplot as plt

def dft(x):

N = len(x)

n = np.arange(N)

k = n.reshape((N, 1))

e = np.exp(-2j \* np.pi \* k \* n / N)

return np.dot(e, x)

def dft1(x):

N = len(x)

n = np.arange(N)

k = n.reshape((N, 1))

e = np.exp(-2j \* np.pi \* -k \* n / N)

return np.dot(e, x)

def plot\_sequence(x,t,title):

plt.stem(t,x)

plt.title(title)

plt.xlabel('n')

plt.ylabel('Amplitude')

plt.grid(True)

plt.show()

x3=np.zeros(8,dtype=complex)

x3=np.array([1,2,1,2,3,4,3,4])

t3=np.arange(0,8)

plt.figure(1)

plot\_sequence(x3,t3,"Input sequence")

plt.figure(2)

plt.subplot(1,2,1)

x3e=x3[0:8:2]

plot\_sequence(x3e,np.arange(0,len(x3e)),"Even indexed sequence")

plt.subplot(1,2,2)

x3o=x3[1:8:2]

plot\_sequence(x3o,np.arange(0,len(x3o)),"Odd indexed sequence")

x=np.add(x3e, 1j\*x3o)

print("Complex sequence is :", x)

plt.figure(3)

DFT=dft(x)

plt.subplot(1,2,1)

plt.title("DFT of complex with loop function (Magnitude)")

plt.xlabel("k")

plt.ylabel("Ampitude")

plt.grid(True)

plt.stem(np.real(DFT))

plt.subplot(1,2,2)

plt.title("DFT of complex with loop function (Phase)")

plt.xlabel("k")

plt.grid(True)

plt.ylabel("Angle")

plt.stem(np.imag(DFT))

DFT1=dft1(x)

CDFT=np.conjugate(DFT1)

plt.figure(4)

x1=(DFT+CDFT)/2

plt.subplot(2,2,1)

plt.title("DFT Coefficients for X1 (Magnitude)")

plt.xlabel("k")

plt.ylabel("Ampitude")

plt.grid(True)

plt.stem(x1)

plt.subplot(2,2,3)

plt.title("DFT Coefficients for X1 (Imaginary component)")

plt.xlabel("k")

plt.ylabel("Angle")

plt.grid(True)

plt.stem(np.imag(x1))

x2=(DFT-CDFT)/2j

plt.subplot(2,2,2)

plt.title("DFT Coefficients for X2 (Magnitude)")

plt.xlabel("k")

plt.ylabel("Ampitude")

plt.grid(True)

plt.stem(x2)

plt.subplot(2,2,4)

plt.title("DFT Coefficients for X2 (Imaginary component)")

plt.xlabel("k")

plt.ylabel("Angle")

plt.grid(True)

plt.stem(np.imag(x2))

def fft(x):

N = len(x)

if N <= 1:

return x

else:

X\_even = fft(x[::2])

X\_odd = fft(x[1::2])

factor = np.exp(-2j \* np.pi \* np.arange(N) / N)

X = np.concatenate([X\_even + factor[:N//2] \* X\_odd,

X\_even + factor[N//2:] \* X\_odd], axis=0) # Ensure correct shape

return X

res=fft(x3)

print("\nCalculated coefficient values from algorithm:",res)

plt.figure(5)

plt.subplot(2,2,1)

plt.title("Final result after performing twiddle factor multiplication (Magnitude)")

plt.xlabel("k")

plt.ylabel("Ampitude")

plt.grid(True)

plt.stem(res)

plt.subplot(2,2,3)

plt.title("Final result after performing twiddle factor multiplication (Imaginary component)")

plt.xlabel("k")

plt.ylabel("Angle")

plt.grid(True)

plt.stem(np.imag(res))

res2=np.fft.fft(x3)

print("\nCoefficient values from built-in function DFT:",res2)

plt.subplot(2,2,2)

plt.title("DFT from in-built function (Magnitude)")

plt.xlabel("k")

plt.ylabel("Ampitude")

plt.grid(True)

plt.stem(res2)

plt.subplot(2,2,4)

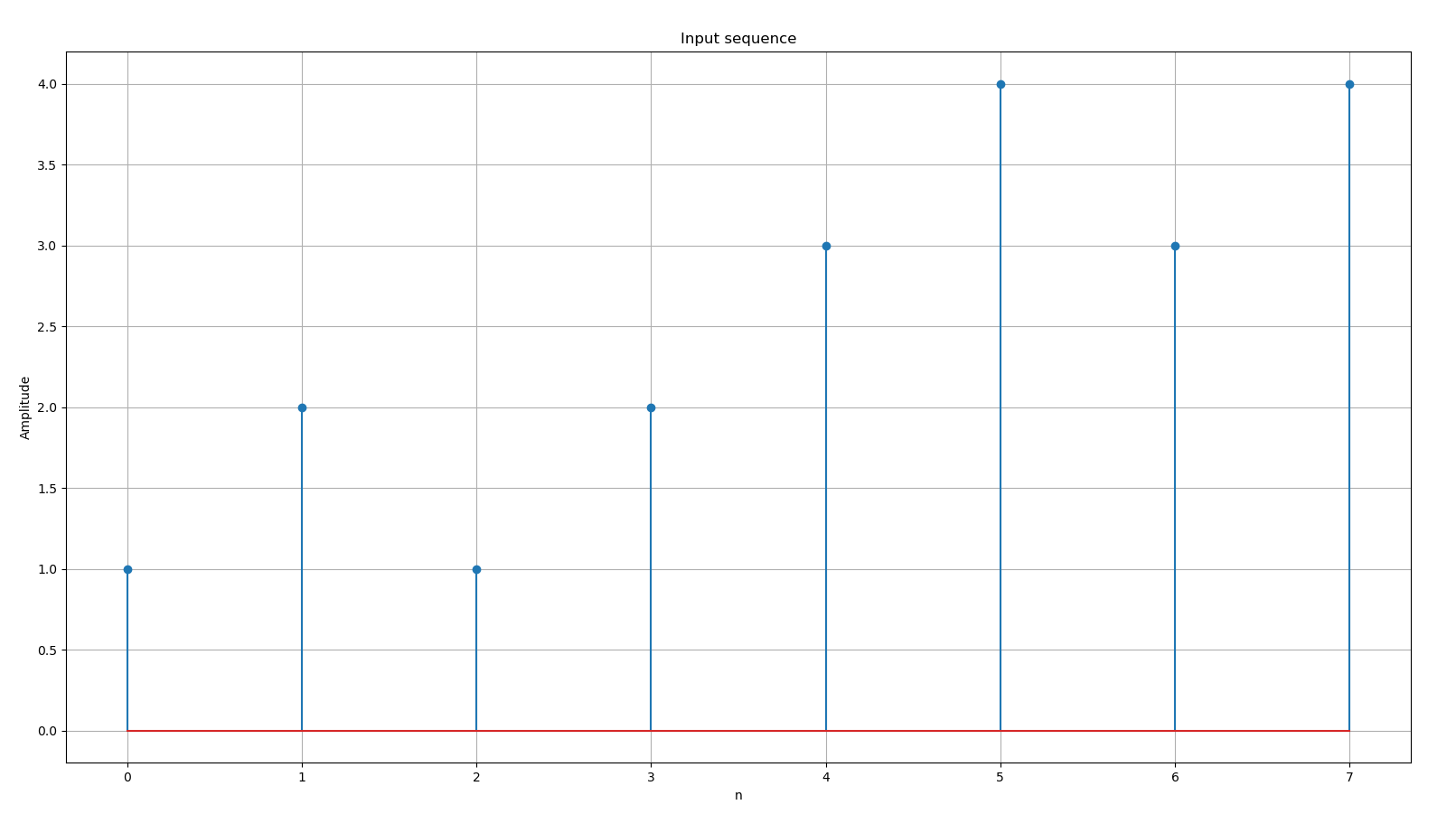
plt.title("DFT from in-built function (Imaginary component)")

plt.xlabel("k")

plt.ylabel("Angle")

plt.grid(True)

plt.stem(np.imag(res2))

OUTPUT:

