Advanced Digital Signal Processing (ADSP) Lab - Python Lab Manual

Course Code: EEE G613
IC: Dr. Rajesh Kumar Tripathy
Research Scholar: Shaswati Dash
Lab Technician: Ramesh Pokanati

Experiment No. - 1

1. a) Consider a discrete time signal sequence of arbitrary size N (get the signal size as input from the user)

   b) Write your own code/program (without using built in matlab commands) to

   Calculate and Plot this signal's DFT.

   c) Compare your answers with the built in matlab 'fft' command.

Python Code:

```
#import libraries
import time
import numpy as np
import matplotlib.pyplot as plt
```

```
# Record the end time
end_time = time.time()
# Calculate the elapsed time
N=8
#discrete time sequence (DTS) (input from user)
x = np.array([1, 3, 2, 5, 6, 4])
print('Discrete Time sequence (x):',x) #Display the DTS x
N1 = len(x) #length of the DTS
x1 = np.append(x, np.zeros(N - N1))
wn = np.exp(-2j * np.pi / N)
X = np.zeros(N, dtype=complex)
print('DTS after zero padding (x1):',x1)
```

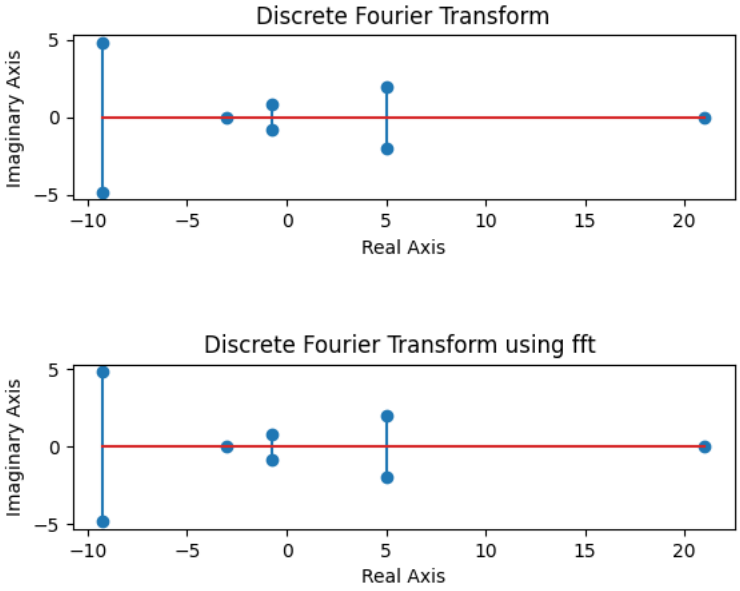
```
Discrete Time sequence (x): [1 3 2 5 6 4]
DTS after zero padding (x1): [1. 3. 2. 5. 6. 4. 0. 0.]
```

```
#Calculate the DFT
start_time = time.time()
for k in range(N):
  for n in range(N):
    X[k] += x1[n] * (wn ** (k * n))
Y = np.fft.fft(x1)
# Record the end time
end_time = time.time()
# Calculate the elapsed time
elapsed_time1 = end_time - start_time
print(f"For FFT elapsed time is:",elapsed_time1)
```

```
For FFT elapsed time is: 0.001302480697631836
```

```
#Plotting of DFT
X1 = np.real(X)
Y1 = np.imag(X)
plt.subplot(2, 1, 1)
plt.stem(X1, Y1)
plt.title('Discrete Fourier Transform')
plt.xlabel('Real Axis')
```

```
plt.ylabel('Imaginary Axis')
plt.subplots_adjust(hspace=1)
X2 = np.real(Y)
Y2 = np.imag(Y)
plt.subplot(2, 1, 2)
plt.stem(X2, Y2)
plt.title('Discrete Fourier Transform using fft ')
plt.xlabel('Real Axis')
plt.ylabel('Imaginary Axis')
plt.show()
```



Output Plot

2. Consider the following linear equation of form AX=b

$$\begin{bmatrix} 1 & 0 & 2 & -1 \\ -1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Write your own code/program to do the following

a) Find the pseudoinverse matrix (check your answer with built-in matlab pseduoinverse command)
b) Solve the equation using least square approach
c) Calculate the error

Python Code:

```
#Import libraries
import time
import numpy as np
from scipy.linalg import pinv, lstsq
```

```
# Define matrices A and B
A = np.array([[1, 0, 2, -1],
[-1, 1, 0, 1]])
B = np.array([[1],[1]])
print('Matrix A=\n',A)
print('Matrix B=\n',B)
```

```
Matrix A=
 [[ 1  0  2 -1]
 [-1  1  0  1]]
```

```
Matrix B=
 [[1]
 [1]]
```

```
# Transpose of matrix A
At = A.T
# Compute Ai
Ai = np.dot(A, At)
# Calculate the inverse of Ai
Ainv = np.linalg.inv(Ai)
# Solve for X using the formula X = At * Ainv * B
X = np.dot(np.dot(At, Ainv), B)
# Solve for X using lsqr method
z = lstsq(A, B)[0]
# Solve for X1 using the pinv command
X1 = pinv(A).dot(B)
# Calculate the difference between X1 and z
E = X1 - z
print("X (Using At * Ainv * B):\n", X)
print("X (Using least square approach):\n", z)
print("X1 (Using pseudoinverse command):\n", X1)
print("Error(E): (between pinv & lsqr result):\n", E)
```

```
X (Using At * Ainv * B):
 [[-0.21428571]
 [ 0.57142857]
 [ 0.71428571]
 [ 0.21428571]]
X (Using least square approach):
 [[-0.21428571]
 [ 0.57142857]
 [ 0.71428571]
 [ 0.21428571]]
X1 (Using pseudoinverse command):
 [[-0.21428571]
 [ 0.57142857]
 [ 0.71428571]
 [ 0.21428571]]
Error(E): (between pinv & lsqr result):
 [[ 5.55111512e-17]
 [ 1.11022302e-16]
 [ 1.11022302e-16]
 [-2.77555756e-17]]
```

3. For the given matrix

$$A = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}$$

a. Calculate by hand eigen values and eigen vectors
b. check the calculated values in a) part by using matlab built-in-function

---

```
#Import libraries
import time
import numpy as np
```

```
# Define the matrix A
A = np.array([[1, -1, 0],
              [-1, 2, -1],
              [0, -1, 1]])
# Display the matrix A
print('Matrix A:\n',A)
```

```
Matrix A:
 [[ 1 -1  0]
 [-1  2 -1]
 [ 0 -1  1]]
```

```
# Calculate eigenvalues and eigenvectors
eigen_values, eigen_vectors = np.linalg.eig(A)
# Display the eigenvalues
print("Eigenvalues:\n",eigen_values)
# Display the eigenvectors
print("Eigenvectors:\n",eigen_vectors)
```

```
Eigenvalues:
 [ 3.00000000e+00  1.00000000e+00 -3.36770206e-17]
Eigenvectors:
 [[-4.08248290e-01 -7.07106781e-01  5.77350269e-01]
 [ 8.16496581e-01  2.61239546e-16  5.77350269e-01]
 [-4.08248290e-01  7.07106781e-01  5.77350269e-01]]
```

N.B. - Avoid Indentation Error in Python !

Mentioned below are some of the common causes of an indentation error in Python:

1. While coding you are using both the tab as well as space. While in theory both of them serve the same purpose, if used alternatively in a code, the interpreter gets confused between which alteration to use and thus returns an error.
2. While programming you have placed an indentation in the wrong place. Since python follows strict guidelines when it comes to arranging the code, if you placed any indentation in the wrong place, the indentation error is mostly inevitable.
3. Sometimes in the midst of finishing a long program, we tend to miss out on indenting the compound statements such as for, while and if and this in most cases will lead to an indentation error.
4. Last but not least, if you forget to use user defined classes, then an indentation error will most likely pop up.