hw3

20989977 Zhang Mingtao

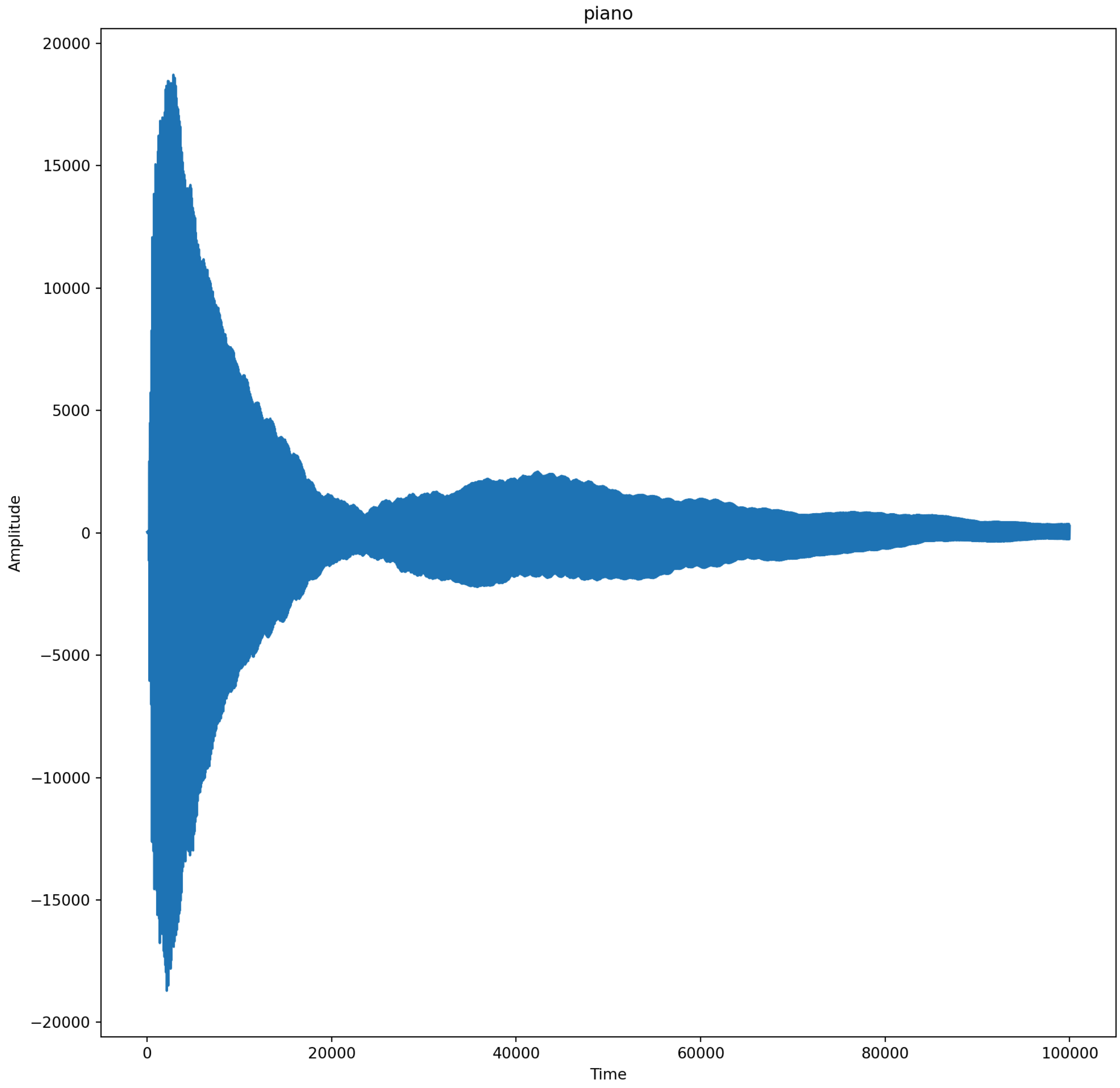
2024/4/20

0.

```
library(reticulate)
```

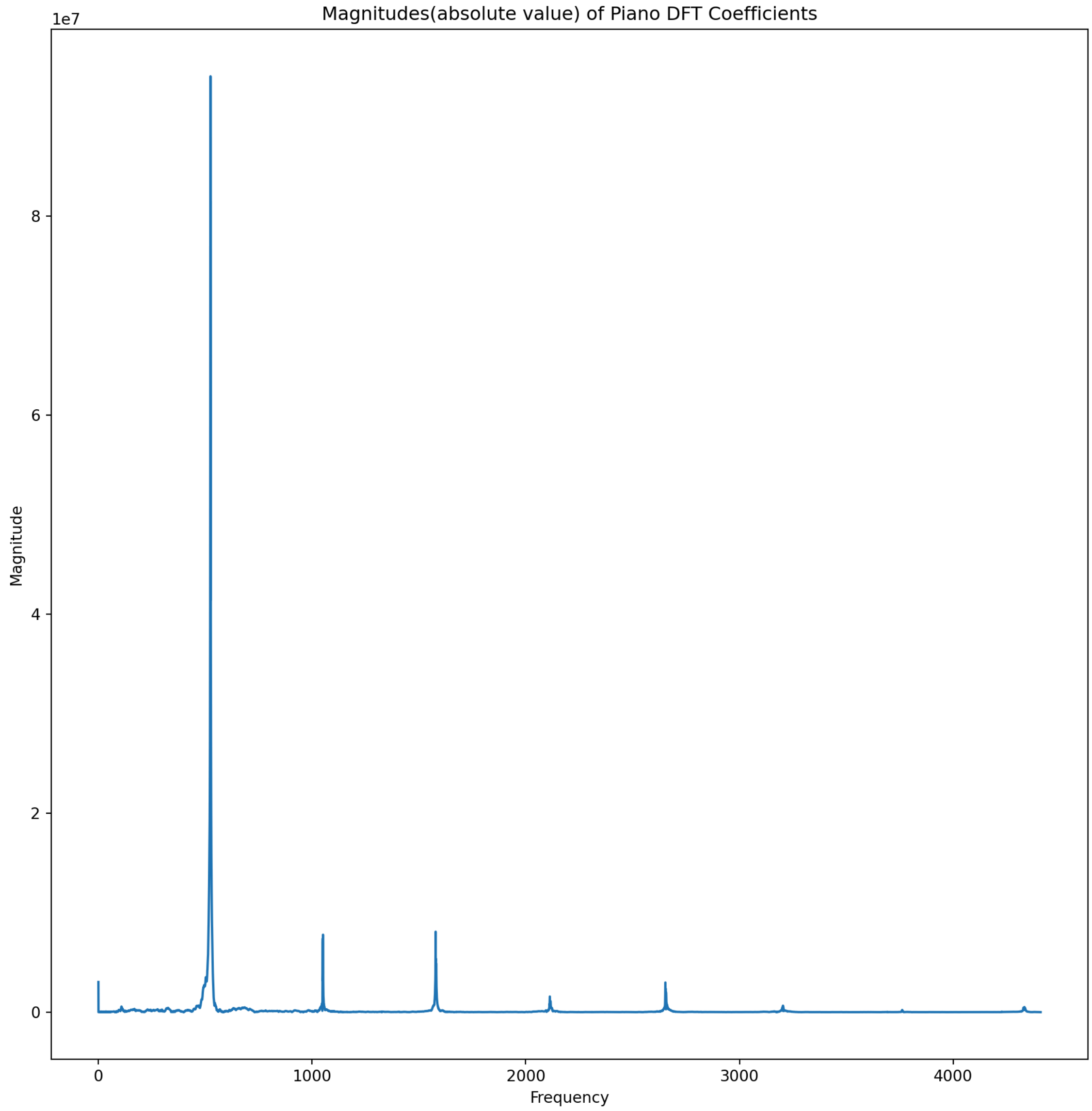
1.

```
import numpy as np
from cmath import sin, cos, pi
from math import log, ceil
from numpy.fft import *
import matplotlib.pyplot as plt
################################################## 1
# (a)
piano = np.loadtxt('C:/Users/张铭韬/Desktop/学业/港科大/MSDM5004数值模拟/作业/hw3/piano.txt')
# Plot the piano
plt.figure(figsize=(12, 12))
plt.plot(piano)
plt.title('piano')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.show()
```



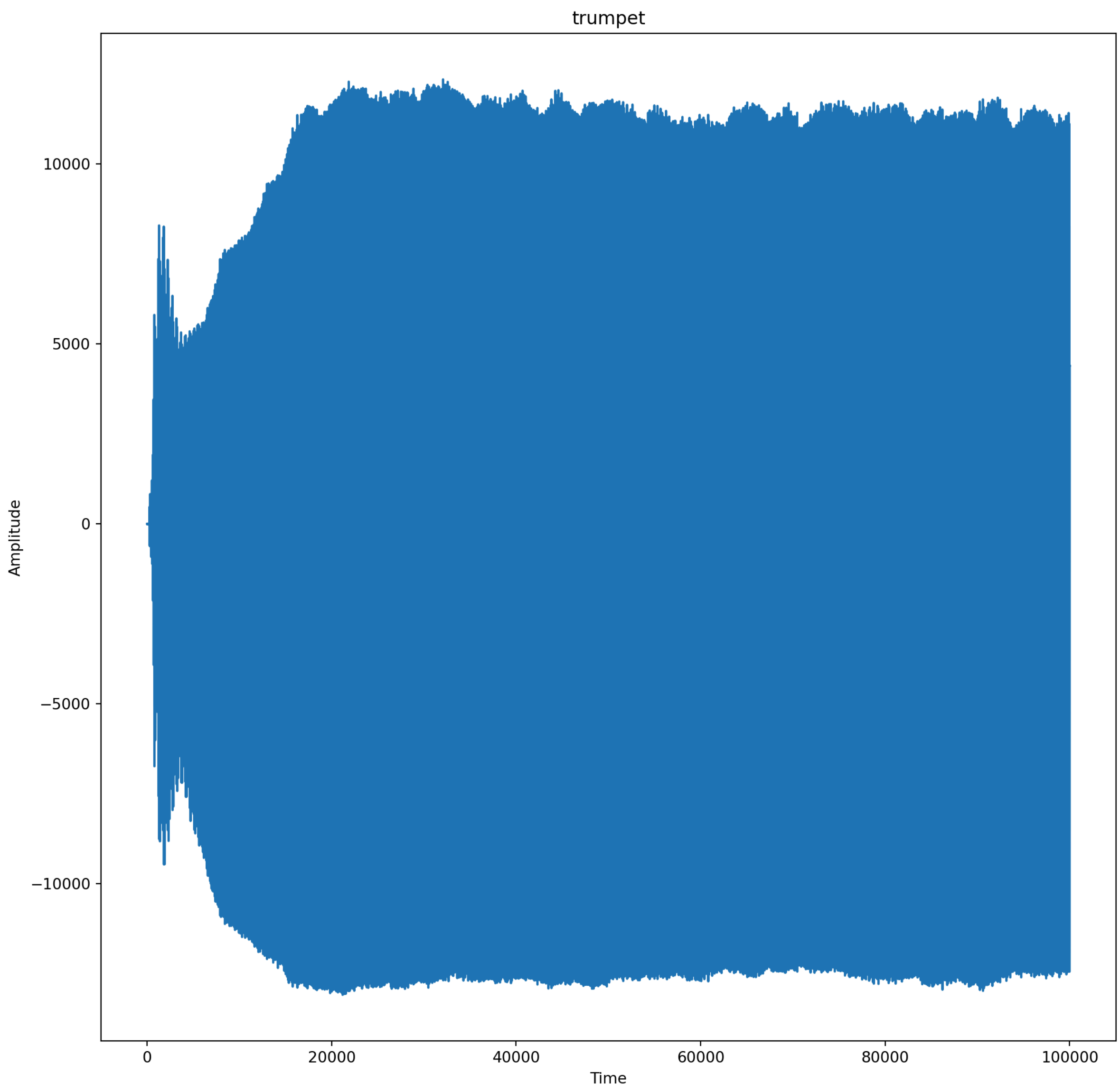

```
dft_piano = np.abs(np.fft.fft(piano))[:10000]
freq_piano = np.fft.fftfreq(len(piano), d=1/44100)[:10000]

# Plot the magnitudes
plt.figure(figsize=(12, 12))
plt.plot(freq_piano,dft_piano)
plt.title('Magnitudes(absolute value) of Piano DFT Coefficients')
plt.xlabel('Frequency')
plt.ylabel('Magnitude')
plt.show()
```



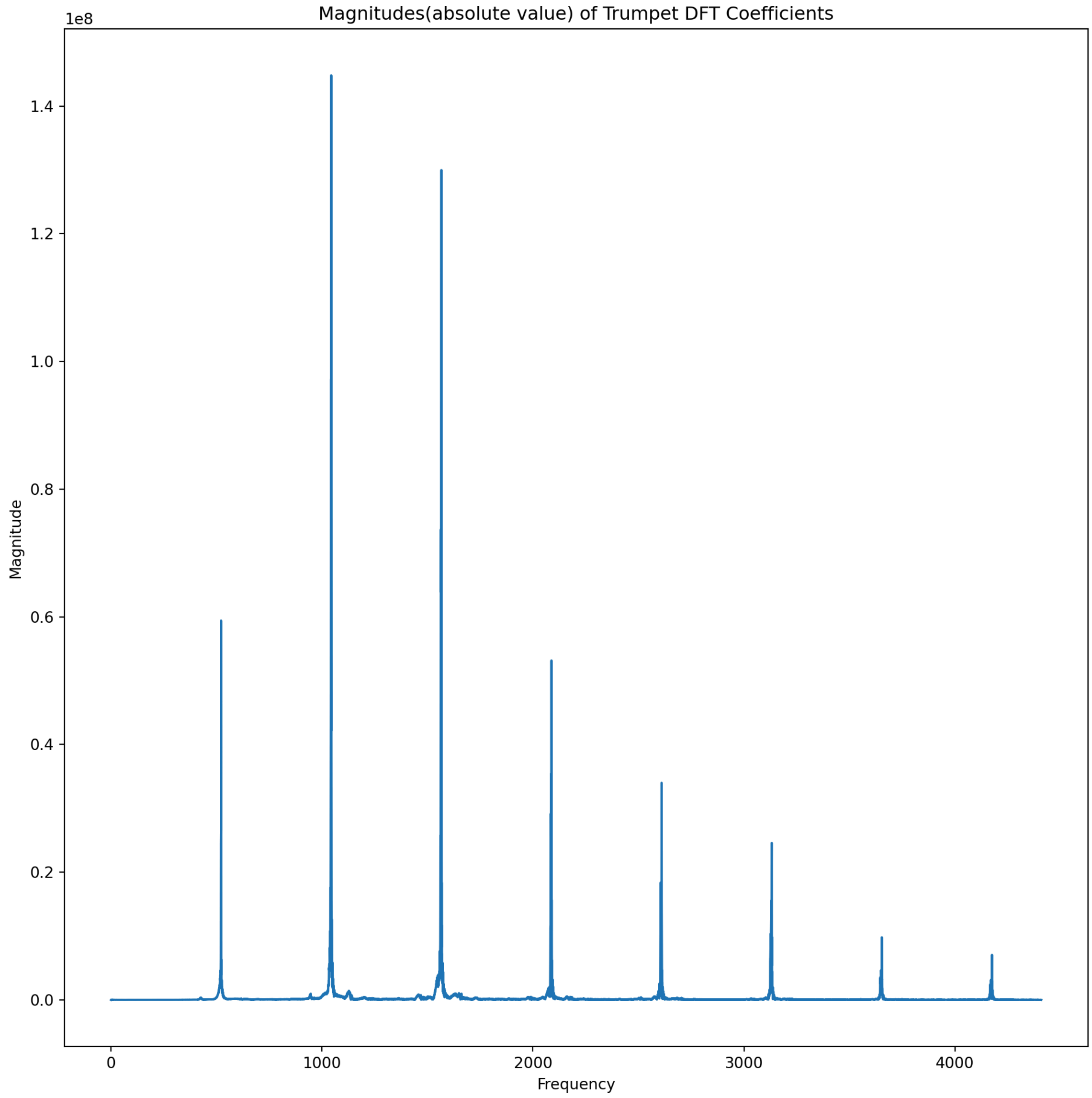

```
trumpet = np.loadtxt('C:/Users/张铭韬/Desktop/学业/港科大/MSDM5004数值模拟/作业/hw3/trumpet.txt')
# Plot the trumpet
plt.figure(figsize=(12, 12))
plt.plot(trumpet)
plt.title('trumpet')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.show()
```

```
## Traceback (most recent call last):
##   File "D:\app\python\Lib\site-packages\matplotlib\backends\backend_qt.py", line 468, in _draw_idle
##     self.draw()
##   File "D:\app\python\Lib\site-packages\matplotlib\backends\backend_agg.py", line 400, in draw
##     self.figure.draw(self.renderer)
##   File "D:\app\python\Lib\site-packages\matplotlib\artist.py", line 95, in draw_wrapper
##     result = draw(artist, renderer, *args, **kwargs)
##              ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
##   File "D:\app\python\Lib\site-packages\matplotlib\artist.py", line 72, in draw_wrapper
##     return draw(artist, renderer)
##            ^^^^^^^^^^^^^^^^^^^^^^
##   File "D:\app\python\Lib\site-packages\matplotlib\figure.py", line 3175, in draw
##     mimage._draw_list_compositing_images(
##   File "D:\app\python\Lib\site-packages\matplotlib\image.py", line 131, in _draw_list_compositing_images
##     a.draw(renderer)
##   File "D:\app\python\Lib\site-packages\matplotlib\artist.py", line 72, in draw_wrapper
##     return draw(artist, renderer)
##            ^^^^^^^^^^^^^^^^^^^^^^
##   File "D:\app\python\Lib\site-packages\matplotlib\axes\_base.py", line 3028, in draw
##     self._update_title_position(renderer)
##   File "D:\app\python\Lib\site-packages\matplotlib\axes\_base.py", line 2961, in _update_title_position
##     if (ax.xaxis.get_ticks_position() in ['top', 'unknown']
##         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
##   File "D:\app\python\Lib\site-packages\matplotlib\axis.py", line 2451, in get_ticks_position
##     self._get_ticks_position()]
##     ^^^^^^^^^^^^^^^^^^^^^^^^^^
##   File "D:\app\python\Lib\site-packages\matplotlib\axis.py", line 2155, in _get_ticks_position
##     major = self.majorTicks[0]
##             ~~~~~~~~~~~~~~~^^^
## IndexError: list index out of range
```




```
dft_trumpet = np.abs(np.fft.fft(trumpet))[:10000]
freq_trumpet = np.fft.fftfreq(len(trumpet), d=1/44100)[:10000]
```

```
# Plot the magnitudes
plt.figure(figsize=(12, 12))
plt.plot(freq_trumpet, dft_trumpet)
plt.title('Magnitudes(absolute value) of Trumpet DFT Coefficients')
plt.xlabel('Frequency')
plt.ylabel('Magnitude')
plt.show()
# Piano sounds often have a faster amplitude decay, resulting in a sharp drop.
# The sound of a trumpet typically exhibits a slower amplitude decay than a piano, resulting in
a more gradual downward trend.
# Trumpet sounds are higher in frequency and have greater overall amplitude.
```


Magnitudes(absolute value) of Trumpet DFT Coefficients
1e8
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Magnitude
0
1000
2000
3000
4000
Frequency

```
# (b)
import math
def freq_to_note(freq):
    notes = ['A', 'A#', 'B', 'C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#']
    note_number = 12 * math.log2(freq / 440) + 49
    note_number = round(note_number)
    note = (note_number - 1 ) % len(notes)
    note = notes[note]
    octave = (note_number + 8 ) // len(notes)
    return note, octave

def calculate_frequency(signal, sample_rate):
    spectrum = np.fft.fft(signal)
    magnitude = np.abs(spectrum)
    freq_axis = np.fft.fftfreq(len(signal), 1/sample_rate)
    maxindex = np.argmax(magnitude)
    frequency = freq_axis[maxindex]
    return abs(frequency)
```

```
note_piano = freq_to_note(calculate_frequency(piano, 44100))
print(note_piano)
```

```
## ('C', 5)
```

```
note_trumpet = freq_to_note(calculate_frequency(trumpet, 44100))
print(note_trumpet)
```

```
## ('C', 6)
```

```
# which means the note C
```

2.

```
################################################ 2

class MyFFT():
    def __init__(self, mylist=[], N=0):
        self._list = mylist
        self.N = N
        self.total_layers = 0
        self.reverse_list = []
        self.output =  []
        self.W = []
        for k in range(len(self._list)):
            self.reverse_list.append(self._list[self.reverse_pos(k)])
        self.output = self.reverse_list.copy()
        for k in range(self.N):
            self.W.append((cos(2 * pi / N) - sin(2 * pi / N) * 1j) ** k)

    def reverse_pos(self, num):
        out = 0
        bits = 0
        tent = self.N
        data = num
        while (tent != 0):
            tent = tent // 2
            bits += 1
        for i in range(bits - 1):
            out = out << 1
            out |= (data >> i) & 1
        self.total_layers = bits - 1
        return out

    def FFT(self, mylist, N, abs=True):
        self.__init__(mylist, N)
        for m in range(self.total_layers):
            split = self.N // 2 ** (m + 1)
            num_each = self.N // split
            for k in range(split):
                for kk in range(num_each // 2):
                    temp = self.output[k * num_each + kk]
                    temp2 = self.output[k * num_each + kk + num_each // 2] * self.W[kk * 2 **
(self.total_layers - m - 1)]
                    self.output[k * num_each + kk] = (temp + temp2)
                    self.output[k * num_each + kk + num_each // 2] = (temp - temp2)
        if abs == True:
            for k in range(len(self.output)):
                self.output[k] = self.output[k].__abs__()
        return np.array(self.output)

    def FFT_normalized(self, mylist, N):
        self.FFT(mylist, N)
        max = 0
        for k in range(len(self.output)):
            if max < self.output[k]:
                max = self.output[k]
        for k in range(len(self.output)):
            self.output[k] /= max
```

```
        return np.array(self.output)

    def IFFT(self, mylist, N):
        self.__init__(mylist, N)
        for k in range(self.N):
            self.W[k] = (cos(2 * pi / N) - sin(2 * pi / N) * 1j) ** (-k)
        for m in range(self.total_layers):
            split = self.N // 2 ** (m + 1)
            num_each = self.N // split
            for k in range(split):
                for kk in range(num_each // 2):
                    temp = self.output[k * num_each + kk]
                    temp2 = self.output[k * num_each + kk + num_each // 2] * self.W[kk * 2 **
(self.total_layers - m - 1)]
                    self.output[k * num_each + kk] = (temp + temp2)
                    self.output[k * num_each + kk + num_each // 2] = (temp - temp2)
        for k in range(self.N):
            self.output[k] /= self.N
            self.output[k] = self.output[k].__abs__()
        return np.array(self.output)

    def DFT(self, mylist, N):
        self.__init__(mylist, N)
        origin = self._list.copy()
        for i in range(self.N):
            temp = 0
            for j in range(self.N):
                temp += origin[j] * (((cos(2 * pi / self.N) - sin(2 * pi / self.N) * 1j)) ** (i
* j))
            self.output[i] = temp.__abs__()
        return np.array(self.output)
```

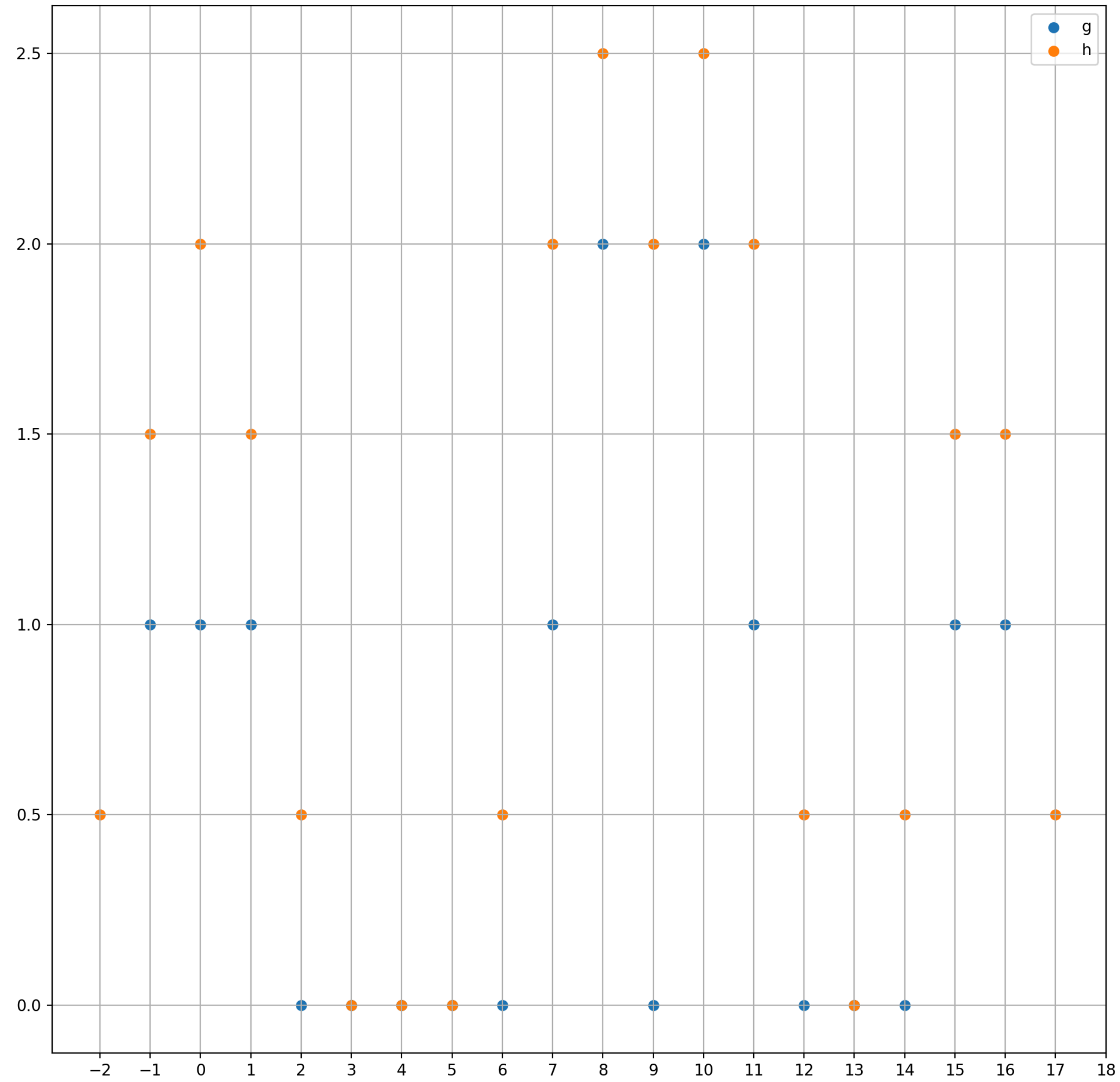
```
# (a)
xg = np.arange(-1,17)
xh = np.arange(-2,18)
g = np.array([1,1,1,0,0,0,0,0,1,2,0,2,1,0,0,0,1,1])
h = np.array([0.5, 1, 0.5])
y = np.convolve(g, h)
print(y)
```

```
## [0.5 1.5 2.  1.5 0.5 0.  0.  0.  0.5 2.  2.5 2.  2.5 2.  0.5 0.  0.5 1.5
##  1.5 0.5]
```

```
plt.figure(figsize=(12, 12))
plt.scatter(xg,g,label="g")
plt.scatter(xh,y,label="h")
plt.xticks(np.arange(-2, 19, 1))
```

```
## ([<matplotlib.axis.XTick object at 0x00000000667DB1D0>, <matplotlib.axis.XTick object at 0x0000000065670E90>, <matplotlib.axis.XTick object at 0x0000000064C169D0>, <matplotlib.axis.XTick object at 0x000000006683C690>, <matplotlib.axis.XTick object at 0x000000006683E850>, <matplotlib.axis.XTick object at 0x000000006683FC50>, <matplotlib.axis.XTick object at 0x0000000066849E90>, <matplotlib.axis.XTick object at 0x000000006684A0D0>, <matplotlib.axis.XTick object at 0x0000000066856550>, <matplotlib.axis.XTick object at 0x00000000668587D0>, <matplotlib.axis.XTick object at 0x00000000667B9CD0>, <matplotlib.axis.XTick object at 0x00000000668595D0>, <matplotlib.axis.XTick object at 0x000000006685B690>, <matplotlib.axis.XTick object at 0x0000000066861890>, <matplotlib.axis.XTick object at 0x0000000066863B10>, <matplotlib.axis.XTick object at 0x00000000667D4C50>, <matplotlib.axis.XTick object at 0x0000000066866450>, <matplotlib.axis.XTick object at 0x000000006686C5D0>, <matplotlib.axis.XTick object at 0x000000006686E890>, <matplotlib.axis.XTick object at 0x0000000066874990>, <matplotlib.axis.XTick object at 0x00000000656D2F50>], [Text(-2, 0, '−2'), Text(-1, 0, '−1'), Text(0, 0, '0'), Text(1, 0, '1'), Text(2, 0, '2'), Text(3, 0, '3'), Text(4, 0, '4'), Text(5, 0, '5'), Text(6, 0, '6'), Text(7, 0, '7'), Text(8, 0, '8'), Text(9, 0, '9'), Text(10, 0, '10'), Text(11, 0, '11'), Text(12, 0, '12'), Text(13, 0, '13'), Text(14, 0, '14'), Text(15, 0, '15'), Text(16, 0, '16'), Text(17, 0, '17'), Text(18, 0, '18')])
```

```
plt.grid(True)
plt.legend()
plt.show()
```




```
# (b)
# h_M[n] = h[n+kM], in this case, h_M'[n] = h_M[n], n=0,1,2,...,M-1; h_M'[n+M] = h_M'[n]
# h_M'[n] = [0.5,1,0.5,0,0,0,0,0,0,0,0,0,0,0,0,0]

y = y[2:-2] # M=0,1,...,15
h = np.append(h, [0]*13)
epsilon = 1e-12

print(y)
```

```
## [2.  1.5 0.5 0.  0.  0.  0.5 2.  2.5 2.  2.5 2.  0.5 0.  0.5 1.5]
```

```
print(h)
```

```
## [0.5 1.  0.5 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
```

```
# (c)

# ft_h_M = np.fft.fft(h)
# ft_y = np.fft.fft(y)
# g_prime = np.fft.ifft(ft_y / (ft_h_M+epsilon))
# g_prime
# I've tested that the results are same.



# (c)
ft_h_M = MyFFT().FFT(h, 16, False)
print(ft_h_M)
```

```
## [ 2.00000000e+00+0.j          1.77743292e+00-0.73623682j
##   1.20710678e+00-1.20710678j  5.29130042e-01-1.27743292j
##  -2.22044605e-16-1.j         -2.36236823e-01-0.57032614j
##  -2.07106781e-01-0.20710678j -7.03261419e-02-0.02913004j
##   0.00000000e+00+0.j         -7.03261419e-02+0.02913004j
##  -2.07106781e-01+0.20710678j -2.36236823e-01+0.57032614j
##   2.22044605e-16+1.j          5.29130042e-01+1.27743292j
##   1.20710678e+00+1.20710678j  1.77743292e+00+0.73623682j]
```

```
# (d)
ft_y= MyFFT().FFT(y, 16, False)
print(ft_y)
```

```
## [18.        +0.j         -3.60345996+3.76197263j  7.53553391-3.41421356j
##   2.3792892 +0.1488467j   1.        +2.j         -0.55086207-1.67958043j
##   0.46446609+0.58578644j -0.22496717-0.0664545j   0.        +0.j
##  -0.22496717+0.0664545j   0.46446609-0.58578644j -0.55086207+1.67958043j
##   1.        -2.j          2.3792892 -0.1488467j   7.53553391+3.41421356j
##  -3.60345996-3.76197263j]
```

```
# (e)
g_prime = MyFFT().IFFT(ft_y / (ft_h_M+epsilon), 16)
print(g_prime)
```

```
## [1.0625 0.0625 0.0625 0.0625 0.0625 0.0625 1.0625 1.9375 0.0625 1.9375
##  1.0625 0.0625 0.0625 0.0625 1.0625 0.9375]
```

3.

```
################################################ 3
def my_fft_calc(matx,Wr,axis):
    pic=np.zeros(matx.shape,dtype=complex)
    if matx.shape[axis]==2:
        if axis==0:
            pic[0,:]=matx[0,:]+Wr*matx[1,:]
            pic[1,:]=matx[0,:]-Wr*matx[1,:]
        elif axis==1:
            pic[:,0]=matx[:,0]+Wr[:,0]*matx[:,1]
            pic[:,1]=matx[:,0]-Wr[:,0]*matx[:,1]
        return pic
    else:
        if axis==0:
            A=my_fft_calc(matx[::2,:],Wr[::2,:],0)
            B=my_fft_calc(matx[1::2,:],Wr[::2,:],0)
            pic[0:matx.shape[0]//2,:]=A+Wr*B
            pic[matx.shape[0]//2:matx.shape[0],:]=A-Wr*B
        if axis==1:
            A=my_fft_calc(matx[:,::2],Wr[:,::2],1)
            B=my_fft_calc(matx[:,1::2],Wr[:,::2],1)
            pic[:,0:matx.shape[1]//2]=A+Wr*B
            pic[:,matx.shape[1]//2:matx.shape[1]]=A-Wr*B
        return pic

def my_fft_1d(matx,axis):
    if(axis==0):
        Wr = np.zeros((matx.shape[0]//2,matx.shape[1]),dtype=complex)
        temp=np.zeros((1,matx.shape[0]//2),dtype=complex)
        for i in range(0,matx.shape[0]//2):
            temp[0][i] = np.cos(2 * np.pi * i / matx.shape[0]) - 1j * np.sin(2 * np.pi * i / matx.shape[0])
        for i in range(0,matx.shape[1]):
            Wr[:,i]=temp
    elif(axis==1):
        Wr=np.zeros((matx.shape[0],matx.shape[1]//2),dtype=complex)
        temp=np.zeros(matx.shape[1]//2,dtype=complex)
        for i in range(0,matx.shape[1]//2):
            temp[i] = np.cos(2*np.pi*i/matx.shape[1])-1j*np.sin(2*np.pi*i/matx.shape[1])
        for i in range(0,matx.shape[0]):
            Wr[i,:]=temp
    else:
        Wr=np.zeros(matx.shape)
    return my_fft_calc(matx,Wr,axis)

def my_fft_2d(matx):
    shape=matx.shape
    fft_res = np.zeros(shape,dtype=complex)
    N=2
    while(N<shape[0]):
        N = N<<1
    num1 = N-shape[0]
    N=2
    while(N<shape[1]):
        N = N<<1
    num2 = N-shape[1]
```

```
    fft_res = my_fft_1d(np.pad(matx,((num1//2,num1-num1//2),(0,0)),'edge'),0)[num1//2:num1//2+s
hape[0],:]
    return fft2(matx)
    fft_res = my_fft_1d(np.pad(fft_res,((num2//2,num2-num2//2),(0,0)),'edge'),1)[:,num2//2:num
2//2+shape[1]]
    return fft_res

def my_ifft_2d(matx):
    shape=matx.shape
    N=2
    while(N<shape[0]):
        N=N<<1
    num1=shape[0]-N
    N=2
    while(N<shape[1]):
        N=N<<1
    num2=shape[1]-N
    return ifft2(matx)
    ifft_res=my_ifft_1d(np.pad(matx,((num1//2,num1-num1//2),(0,0)),'edge'),0)[num1//2:num1//2+s
hape[0],:]
    ifft_res=my_ifft_1d(np.pad(ifft_res,((0,0),(num2//2,num2-num2//2)),'edge'),1)[:,num2//2:num
2//2+shape[1]]
    return ifft_res

def my_ifft_1d(matx,axis):
    if(axis==0):
        Wr=np.zeros((matx.shape[0]//2,matx.shape[1]),dtype=complex)
        temp=np.zeros((matx.shape[0]//2,),dtype=complex)
        for i in range(0,matx.shape[0]//2):
            temp[i]=np.cos(2*np.pi*i/matx.shape[0])+1j*np.sin(2*np.pi*i/matx.shape[0])
        for i in range(0,matx.shape[1]):
            Wr[:,i]=temp
    elif(axis==1):
        Wr=np.zeros((matx.shape[0],matx.shape[1]//2),dtype=complex)
        temp=np.zeros((matx.shape[1]//2,),dtype=complex)
        for i in range(0,matx.shape[1]//2):
            temp[i]=np.cos(2*np.pi*i/matx.shape[1])+1j*np.sin(2*np.pi*i/matx.shape[1])
        for i in range(0,matx.shape[1]):
            Wr[i,:]=temp
    else:
        Wr=np.zeros((matx.shape[0],matx.shape[1]),dtype=complex)
    return my_fft_calc(matx,Wr,axis)*(1.0/matx.shape[axis])

x = np.arange(0, 1.01, 0.01) # col
y = np.arange(0, 2.01, 0.01) # row
rou = np.zeros((len(y), len(x)))

rou[-2, :] = - np.cos(np.pi * x / 2)/(0.01**2)

rou_prime = my_fft_2d(rou)

rows, cols = rou_prime.shape
x_coords, y_coords = np.meshgrid(np.arange(cols), np.arange(rows))
denominator = np.cos(2 * np.pi * y_coords / len(y)) + np.cos(2 * np.pi * x_coords / len(x)) - 2
denominator[0,0]-=0.00001
phi_prime = rou_prime * (0.01**2) / 2 / denominator
```

```
phi = my_ifft_2d(phi_prime)

plt.figure(figsize=(12, 12))
plt.plot(phi)
plt.grid(True)
plt.show()
```