Report - Assignment 9

Akshay Anand (EE16B046)

April 10, 2018

Abstract

In this week's assignment, the digital fourier transform of some given functions were found out using python functions for obtaining the same like *numpy.fft.fft*(). Then, the values found by this function were normalized to their actual numerical values using *numpy.fft.fftshift*(), after which the phase and magnitude plots of the resulting transforms were plotted.

Libraries Used

```
import numpy as np
from numpy import fft as f
import matplotlib.pyplot as plt
import math
```

1 Prerequisite Functions Defined

1.1 Function to calculate the DFT

The following function, when given a mathematical function, calculates the DFT of that function and returns this calculated value.

```
def calc_fft(func, steps):
    ''' The parameters for this function:
        func - The python function, which returns the calculated values for the
           mathematical function for which DFT is to be calculated.
        steps - The number of steps taken in the time interval.
    '''
    x=np.linspace(-8*math.pi,8*math.pi,steps+1);
    x=x[:-1]
    y=func(x)
    Y=f.fftshift(f.fft(f.ifftshift(y)))/steps
    return Y
```

1.2 Function to plot the graphs

The function defined below, takes the parameters, as mentioned in the docstring, and plots the phase and magnitude plot of the passed DFT of some function.

```
def plot_fft(Y,x_lim,y_lim,function,steps,offset,show_data_points):
    ''' The parameters accepted by the function to do the plotting:
         Y - The DFT of the function, to be plotted, whose magnitude and phase
            constitute the 2 plots.
         x_lim - The magnitude of maximum frequency to be shown in both the plots.
         y_lim - The magnitude of maximum phase to be shown in the phase plot.
         function - The function to be plottes (in string form to show in graph)
         steps - No. of steps to take for the frequency array.
         offset - The maximum offset for the labels in the graph.
```

```
        show_data_points - A boolean, for displaying the data values in the graph.
    '''
    w=np.linspace(-64,64,steps+1)
    w = w[:-1]
    ctr = 0
    fig, axes = plt.subplots(2, 1, figsize=(15, 7), sharex = True)

    plt.suptitle("The␣DFT␣plots␣for␣" + function, fontsize=18)
    # The magnitude plot is plotted
    axes[0].plot(w,abs(Y),lw=2)
    if show_data_points:
        for xy in zip(w, abs(Y)):
            if xy[1] > 1e-3:
                axes[0].annotate('(%s,␣%s)' % xy, xy=xy, textcoords='data') # To
                    mark the points where the impulse occurs
    axes[0].set_xlim([-x_lim,x_lim])
    axes[0].set_ylabel(r"$|Y|$",size=16)
    axes[0].set_title("Spectrum␣of␣" + function, fontsize=14)
    axes[0].grid(True)

    # The phase plot is plotted
    ii=np.where(abs(Y)>1e-3)
    axes[1].plot(w[ii],np.angle(Y[ii]),'go',lw=2)
    if show_data_points:
        for xy in zip(w[ii], np.angle(Y[ii])):
            axes[1].annotate('(%0.2f,␣%0.2f)' % xy, xy=(xy[0],xy[1]+((-1)**ctr)*
                offset), textcoords='data')     # To mark the phase at which the
                impulse occurs
            ctr = ctr + 1
    axes[1].set_xlim([-x_lim,x_lim])
    axes[1].set_ylim([-y_lim,y_lim])
    axes[1].set_ylabel(r"Phase␣of␣$Y$",size=16)
    axes[1].set_title("Phase␣Plot␣of␣" + function, fontsize=14)
    axes[1].set_xlabel(r"$k$",size=16)
    axes[1].grid(True)
    plt.show()
```

2 Plots of some simple functions

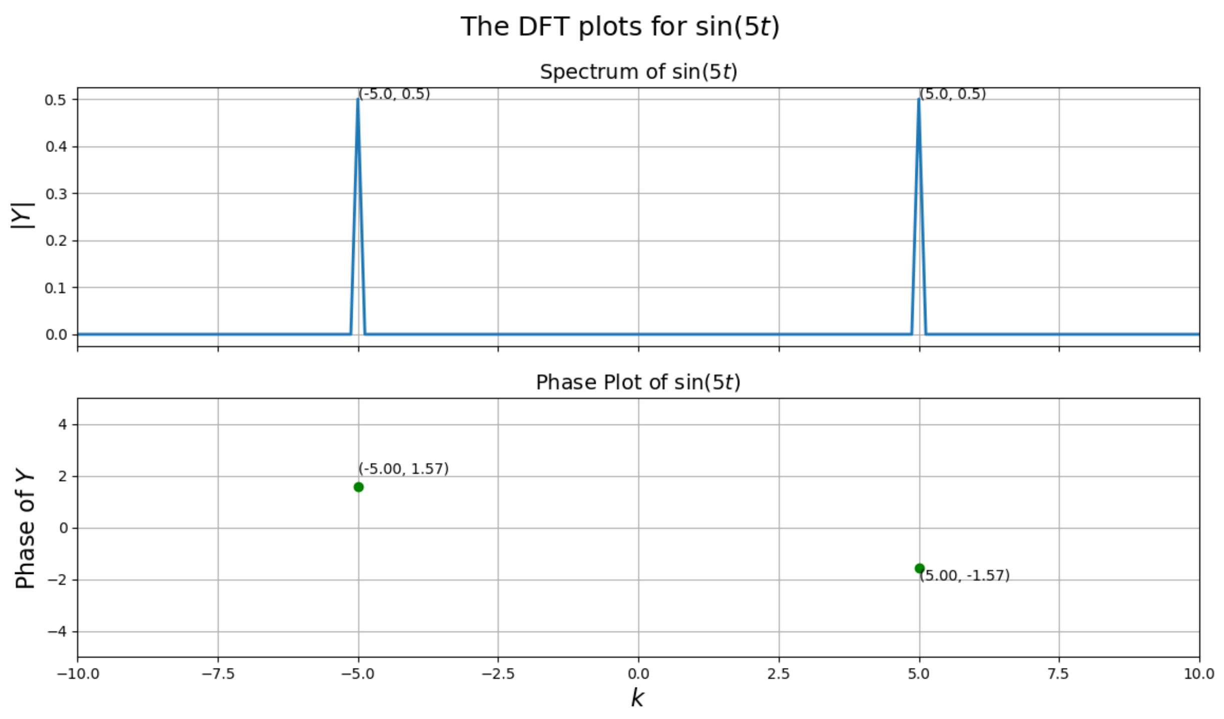
In this section, the transform plots of some simple functions are done so as to verify the functioning of the python plots, to see if the plots correspond to the expected transforms of these functions, after which, more complicated functions' plots would plotted.

2.1 $\sin 5t$ spectrum

One of the simplest functions available, $\sin 5t$ is used, and its digital fourier transform plots (magnitude and phase) are plotted.

```
def f1(x):     # The python function representing sin(5t) is declared.
    return np.sin(5*x)

no_of_steps = 1024
Y = calc_fft(f1, no_of_steps)    # calc_fft is called to calculate and return the
    calculated DFT.
plot_fft(Y,10,5,r"$\sin(5t)$",no_of_steps,0)
```

The output plot obtained is:



Thus, from the plots, it is seen that the impulse occurs at frequencies of $+5$ and -5, as expected as the input sine wave has a frequency of 5. Also, the phase at these points are $+\frac{\pi}{2}$ and $-\frac{\pi}{2}$, which is also expected, as the complex representation of $\sin(5t)$ is

$$\sin(5t) = \frac{e^{j5t}}{2j} - \frac{e^{-j5t}}{2j}$$

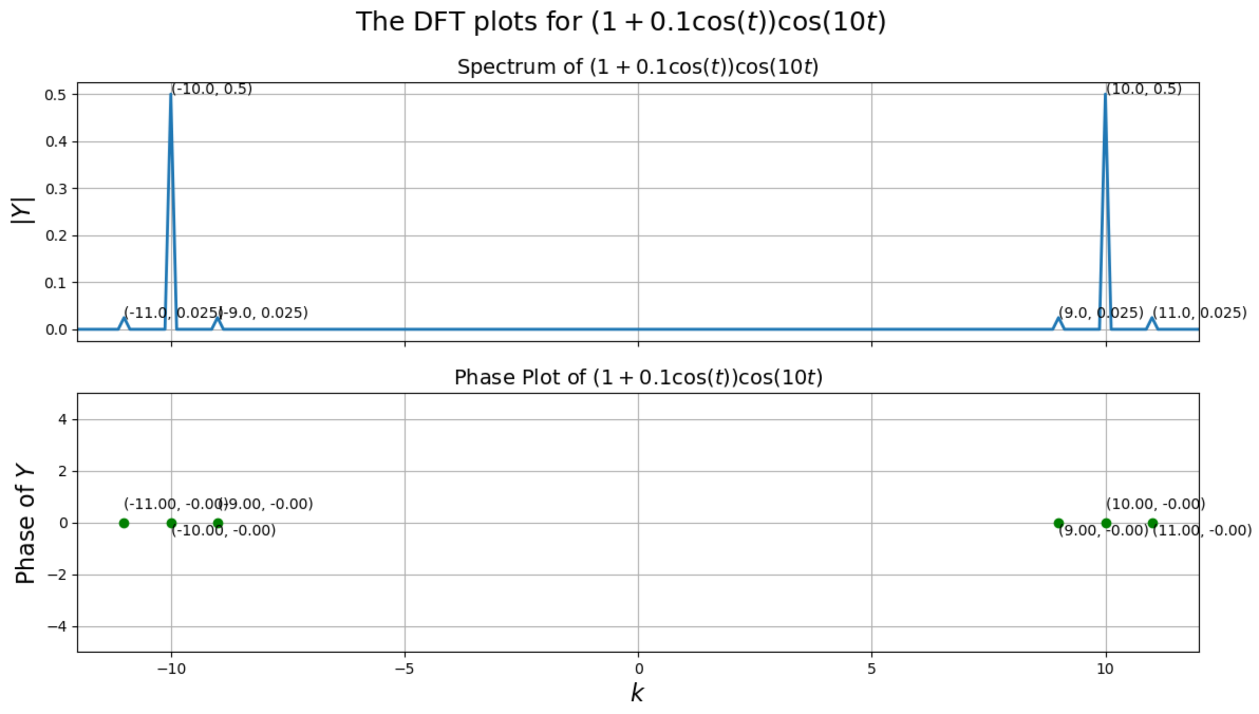
where, both elements are purely imaginary, with opposite signs and a magnitude of 0.5. Thus, the calculated graph is verified.

2.2 $(1 + 0.1\cos(t))\cos(10t)$ spectrum

In this example, spectrum of the the amplitude modulated wave $(1 + 0.1\cos(t))\cos(10t)$ is plotted, which is a slightly more complicated function than the one plotted above, and its obtained graphs are verified.

```python
def f2(x):
    return (1 + 0.1*np.cos(x))*np.cos(10*x)

no_of_steps = 1024
Y = calc_fft(f2, no_of_steps)
plot_fft(Y,12,5,r"$(1+0.1\cos(t))\cos(10t)$",no_of_steps,0.5)
```

The output plot is:



The complex representation of

$$f(t) = (1 + 0.1\cos(t))\cos(10t)$$

is

$$f(t) = 0.5\left(e^{j10t} + e^{-j10t}\right) + 0.025\left(e^{j11t} + e^{-j11t} + e^{j9t} + e^{-j9t}\right)$$

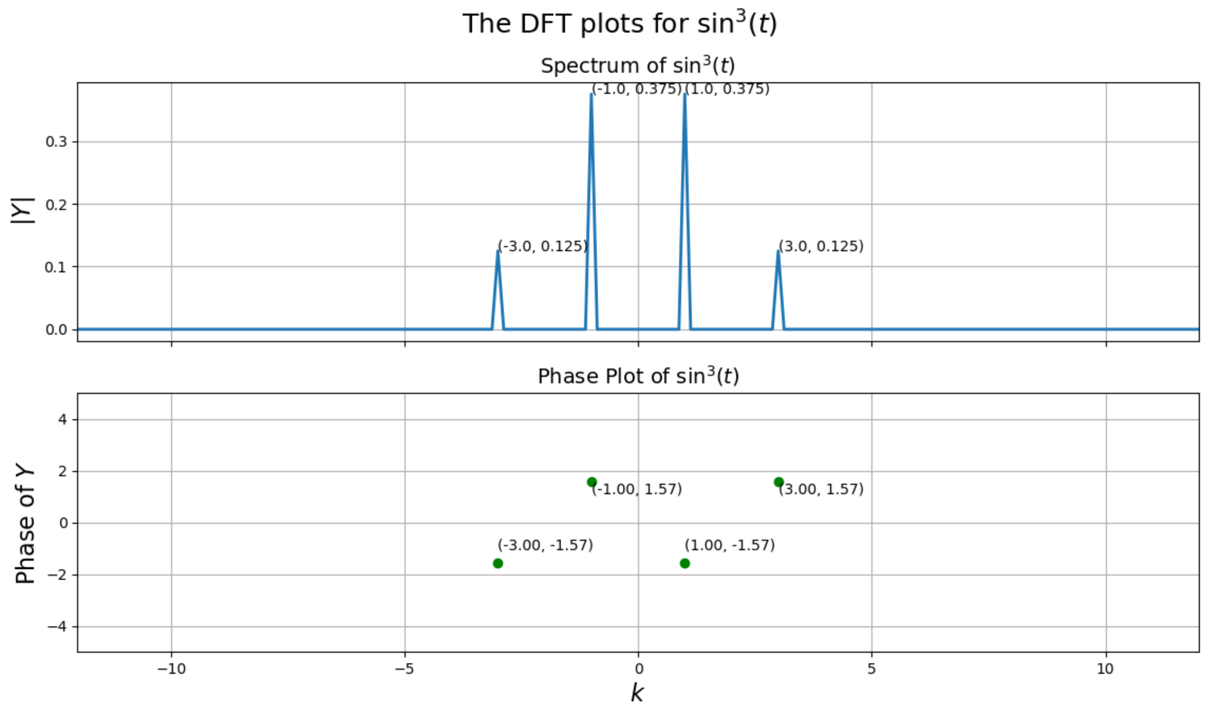
Thus, from the plots, it is seen that, all the phases are 0, which is expected, as only positive real coefficients are there in the complex representation, which occurs at frequencies of -9,-10,-11,9,10 and 11. Also, from the magnitude plot, it is confirmed that the coefficients of the 11 and 9 frequency components are indeed 0.025 and that of 10 is 0.5, according to the equation.

2.3 $\sin^3(t)$ Spectrum

In this, the spectrum of $\sin^3(t)$ is plotted and the graphs obtained are verified.

```
def f3(x):
    return np.sin(x)**3

no_of_steps = 1024
Y = calc_fft(f3, no_of_steps)
plot_fft(Y,12,5,r"$\sin^3(t)$",no_of_steps,0.5)
```

The graphs obtained are:



The complex representation of

$$f\left(t\right) = \sin^3\left(t\right)$$

is

$$f\left(t\right) = -j\left(0.375\left(e^{jt} - e^{-jt}\right) - 0.125\left(e^{j3t} - e^{-j3t}\right)\right)$$

using the formula $\sin(3t) = 3\sin(t) - 4\sin^3(t)$.

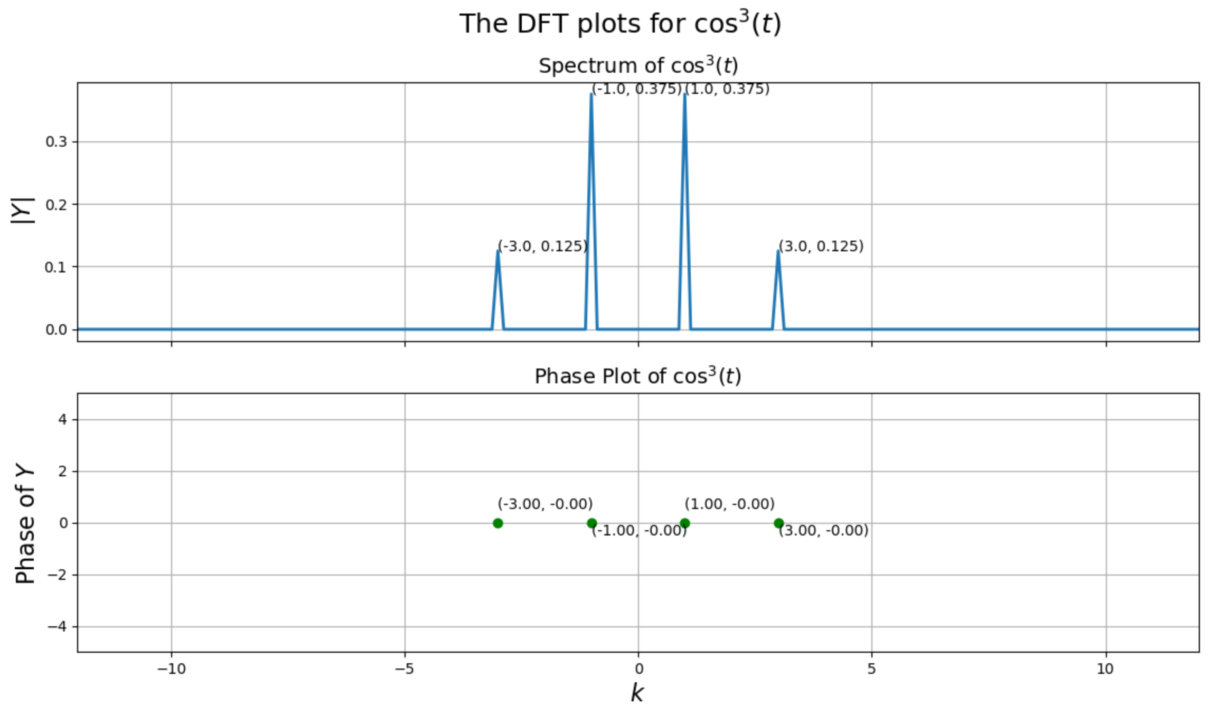
Thus, the observation from the graph is verified by the formula, in that, the magnitude of the peak at 3 and -3 are 0.125, and those at 1 and -1 are 0.375. Also, the phase at frequencies of -1 and 3 are $+\frac{\pi}{2}$, while those at -3 and 1 are $-\frac{\pi}{2}$, which also can be observed in the formula, as the coefficients of these terms are $+j$ and $-j$, respectively, when the equation is expanded.

2.4 $\cos^3(t)$ Spectrum

In this, the spectrum of $\cos^3(t)$ is plotted and the graphs obtained are verified.

```
def f4(x):
    return np.cos(x)**3

no_of_steps = 1024
Y = calc_fft(f4, no_of_steps)
plot_fft(Y,12,5,r"$\cos^3(t)$",no_of_steps,0.5)
```

The graphs obtained are:



Thus, similar to the previous example, the complex representation of

$$f\left(t\right) = \cos^3\left(t\right)$$

is

$$f\left(t\right) = 0.375\left(e^{jt} + e^{-jt}\right) + 0.125\left(e^{j3t} + e^{-j3t}\right)$$

using the formula $\cos(3t) = 4\cos^3(t) - 3\cos(t)$.

Thus, the observation from the graph is verified by the formula, in that, the magnitude of the peak at 3 and -3 are 0.125, and those at 1 and -1 are 0.375. Also, the phase at all frequencies (-1, 1, -3 and 3) are 0, which can also be observed from the formula, as the coefficients of all these terms are positive real numbers.

3 Plots of more complex functions

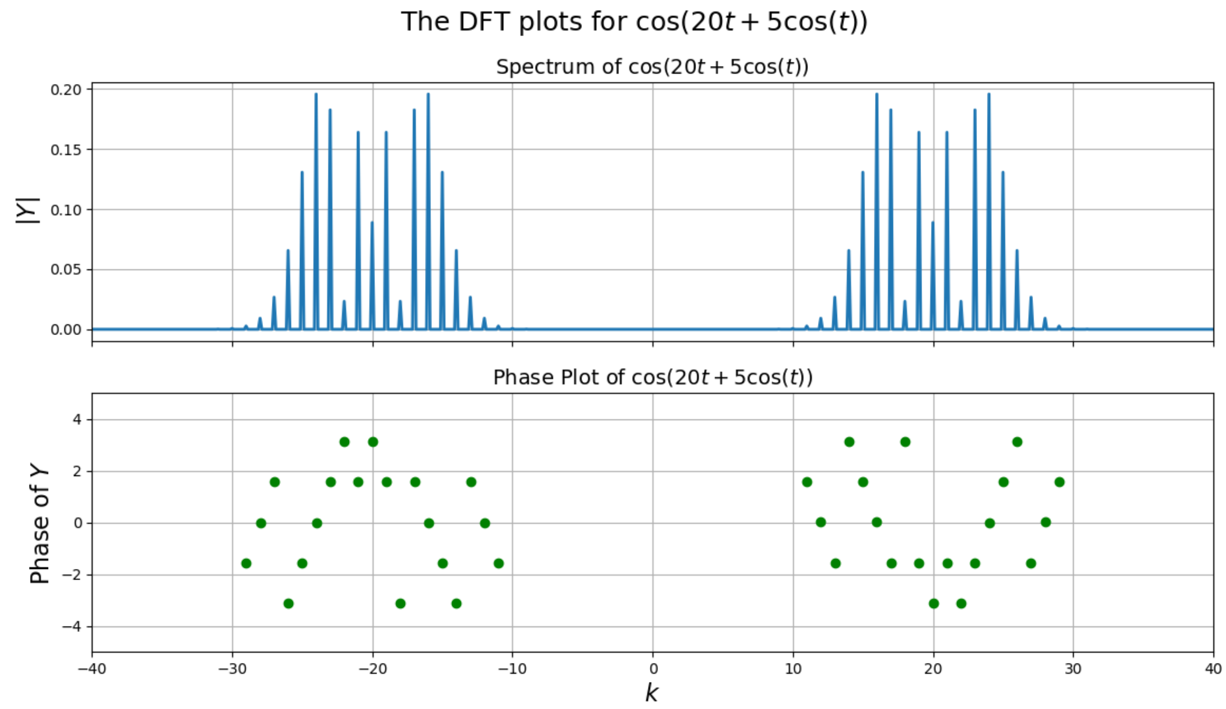
From the above plots, we have verified that what $numpy.fft.fft()$ returns actually matches the expected output that we predicted using the actual DFT formulas. So, in this section, the python fft is used to compute the fft of some more complex functions, from which some observations are made.

3.1 $\cos(20t + 5\cos(t))$ transform

In this section, the DFT plots for the Frequency modulated wave: $\cos(20t + 5\cos(t))$ is drawn, and the observations are noted.

```
def f5(x):
    return np.cos(20*x + 5*np.cos(x))

Y = calc_fft(f5, no_of_steps)
plot_fft(Y,40,5,r"$\cos(20t␣+␣5\cos(t))$",no_of_steps,0,False)
```

The output obtained is:



Thus, it can be seen that, in this example of a frequency modulated wave, the amplitude corresponding to each frequency, follows the amplitude curve of a bessel function from -30, with an initial rise at around -24, minima at around -22 reaching the median value at the centre frequency of -20 and following a mirror image path to -10. The same is followed in the Right Half Plane also, as the actual function is still a cosine wave, and therefore, the LHP and RHP have the same graphs.

In the phase plot, though, instead of getting the same phase diagrams, as other cosine waves in the 2 halves, the one on the right is the negative of the reflection along -20 line of the one on the left. This is because of the $5\cos(t)$ term in the phase of the original expression. This term can go to both positive and negative and therefore the phase will revolve around the central frequency of 20, as given by the first term in the phase $20t$. Also, when the complex representation of the equation is taken, then sign of $5\cos(t)$ would be opposite for both terms. Thus, the reflection around k = 20 is justified. The reflection around the phase = 0 line for the RHP diagram, as compared to the LHP diagram, can be attributed to the fact that, when the initial function is written in its complex form, the frequency of the first term would be $20t + 5\cos(t)$, whereas that of second would be negative of this. Thus, the entire phase is negative in the RHP, and therefore, the phase graph is as seen above.

3.2 $e^{\frac{-t^2}{2}}$ transform

In this, the transform of the gaussian - $e^{\frac{-t^2}{2}}$, is found out. This is also expected to be a gaussian itself, with different scaling. For the input gaussian as:

$$f(t) = e^{\frac{-t^2}{2}}$$

the fourier transform is also a gaussian, represented by:

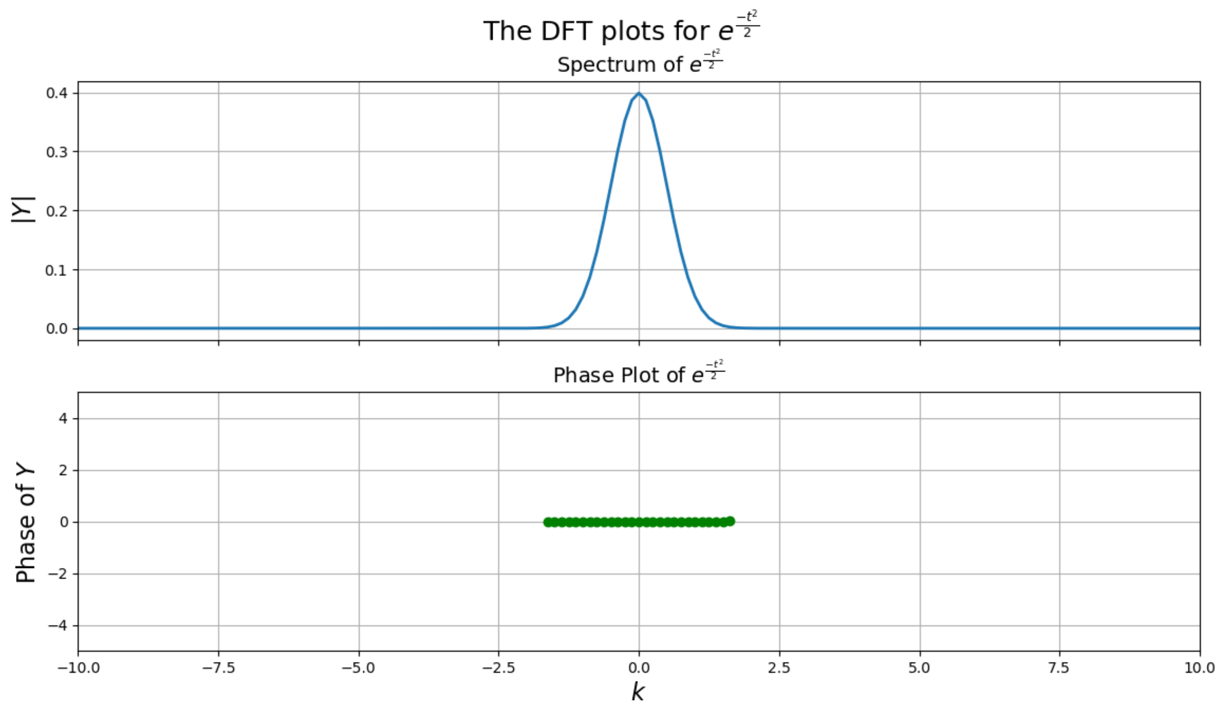
$$\mathscr{F}_t\left[e^{\frac{-t^2}{2}}\right](\omega) = \frac{1}{\sqrt{2\pi}}e^{-2\omega^2}$$

So, the values obtained via *numpy.fft.fft* compared with one obtained with this formula, and the maximum error is computed. For this, another calc_fft() function is declared. This is because, for this case, the scaling factor, to be multiplied with Y is different, as this is a non-periodic function. So, only the no. of steps in a single period of 0 to 2π is to be divided. So, in this particular case, that is the $\frac{1}{4^{th}}$ of the total number of steps. Also, here the interval is also different $(-4\pi, 4\pi)$, rather than $(-8\pi, 8\pi)$, as was done for the other functions. This is so as to scale the output of the python computed fft correctly, so as to match with the actual transform of the function.

```
def calc_fft_2(func, steps):
    x=np.linspace(-4*math.pi,4*math.pi,steps+1);
    x=x[:-1]
    y=func(x)
    Y=f.fftshift(f.fft(f.ifftshift(y)))/(steps/4)
    return Y

def gauss(w):
    return np.sqrt(1/(2*math.pi))*np.exp(-(w**2)*2)

def f6(x):
    return np.exp(-(x*x)/2)

no_of_steps=1024
Y = calc_fft_2(f6, no_of_steps)
w=np.linspace(-64,64,no_of_steps+1)
w = w[:-1]
print ("The␣maximum␣error␣between␣the␣expected␣and␣calculated␣gaussians␣are␣", max(
   np.absolute(gauss(w) - Y)))
plot_fft(Y,10,5,r"$e^{\frac{-t^2}{2}}$",no_of_steps,0,False)
```

Here, the output obtained is:

```
The maximum error between the expected and calculated gaussians are  1.80490527317e-15
```



Thus, it is proved from the graph, that the resulting transform is also a gaussian, with all phases as 0, as it is also only a real valued gaussian, with no imaginary components. It also matches very closely with the expected gaussian, as the maximum error that is returned is of the order of 10^{-15}.

4 Conclusion

Thus, from the initial plots, we verified the fourier transform that was calculated by python using *numpy.fft.fft*, as matching the expected ones, for those functions. Then, the python function was used to calculate the fourier transforms of more complex functions like gaussian and a frequency modulated wave, for which, calculating their actual expressions are difficult. So, this obtained fourier transform was analysed and verified with the known properties of the signal.