The Digital Fourier Transform

Course: EE2703 - Applied Programming Lab

Date: 10/04/2018

Abstract

In this week, we looked upon the way python helps us to obtain the fourier transform of a particular signal using the in-built 'fft' function. So, we are exploring the 'fft' function in this assignment and also how to recover the analog Fourier Tranform for some known functions by the proper sampling of the function.

1 INTRODUCTION

A time function of the type f(t)u_0(t) that grows slower than $e^{\alpha t}$ for some α has a Laplace transform, F(s). The Laplace transform has an inverse transform which involves integrating vertically in the complex plane along a path that is to the right of all poles.

A finite energy function of the type f(t) has a Fourier Transform (and its inverse) of the following type,

$F(j\varpi) = \int_{-\infty}^{\infty} f(t)e^{-j\varpi t}dt$

$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(j\varpi)e^{j\varpi t}d\varpi$

If f (t) is periodic with period 2π, the Fourier transform collapses to the Fourier series,

$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{jnt}$

where,

$c_n = \frac{1}{2\pi}\int_{t_0}^{t_0+2\pi} f(t)e^{-jnt}dt$

for any t_0. For conveniece,

$t_0 = 0$.

In case of discrete time, we have the following equations,

$F(z) = \sum_{n=-\infty}^{\infty} f[n]z^{-n}$

For $z = e^{j\theta}$, we have,

$F(e^{j\theta}) = \sum_{n=-\infty}^{\infty} f[n]e^{-jn\theta}$

So, clearly F(z) is like the periodic time function that gives rise to the fourier series whose coefficients are the samples f [n].

$F(e^{j\theta})$ is called the Digital Spectrum of the samples f [n]. It is also called the DTFT of f [n].

$F(e^{j\theta})$ is continuous and periodic.

f [n] is discrete and aperiodic.

Suppose now f [n] is itself periodic with a period N,

i.e., f[n + N] = f[n] $\forall$n

Then, it should have samples for its DTFT. This is true, and leads to the Discrete Fourier Transform or the DFT.

Suppose f [n] is a periodic sequence of samples, with a period N.

Then the DTFT of the sequence is also a periodic sequence F[k] with the same period N. So,

F[k] = cf[n]$e^{-\frac{2\pi nkj}{N}}$ = $\sum_{n=0}^{N-1}$f[n]W^{nk}

f[n] = $\frac{1}{N}\sum_{k=0}^{N-1}$F[k]$W^{-nk}$

(W = $e^{-\frac{2\pi j}{N}}$)

The values F[k] are what remains of the Digital Spectrum F($e^{j\theta}$).

We can consider them as the values of F($e^{j\theta}$) for $\vartheta = \frac{2\pi k}{N}$, since the first N terms in the expression for F($e^{\frac{2\pi kj}{N}}$) yield,

F($e^{\frac{j2\pi k}{N}}$) = $\sum_{n=0}^{N-1}$f[n]e($\frac{-2\pi knj}{N}$) + ...

which is same as the DFT expression. The remaining terms are repetitions of the above sum and help build up the delta function that is needed to take us from a continuous transform to discrete impulses. The DFT is a sampled version of the DTFT, which is the digital version of the analog Fourier Transform.

2 QUESTIONS

2.1 Working out the examples from the given pdf,

2.1.1 Calculating dft in python for a random function :

```
#Importing modules and libraries

from pylab import *
import matplotlib.pyplot as plt
```

In the above code, I have imported all the modules and libraries I will require in this assignment.

There are two commands in Python, one to compute the forward fourier transform and the other to compute the inverse transform. They are,

numpy.fft.fft()

(for forward fourier transform)

numpy.fft.ifft()

(for inverse fourier transform)

Below is one example where we have calculated the fourier and inverse fourier transform,

```
#Calculating fourier and inverse fourier transform

x = rand(100)
X = fft(x)
```

```
#forward fourier transform of x
y = ifft(X)
#inverse fourier transform of X
c_[x,y]
#creating a matrix with x and y as columns
print abs(x-y).max()
```

The starting vector, x, and the vector got by running fft and ifft on x are essentially the same vector. The command c_[x,y] tells Python to make a 100 $\times$ 2 matrix by using x and y as the columns. This is a quick and dirty way to print both out side by side. And what this shows is that x is pure real while y is very slightly complex. This is due to the finite accuracy of the CPU so that the ifft could not exactly undo what fft did.

Inferences:

1) After running the above code, I got the max diference between the values of x as 1.03375543203e-15.

2.1.2 Calculating dft in python for sine function :

In this question, we took a sinusoidal function,

y = sin(x) = $\frac{e^{jx}-e^{-jx}}{2j}$

So,

Y(ϖ) =$\frac{1}{2j}$[δ(ϖ - 1) - δ(ϖ + 1)]

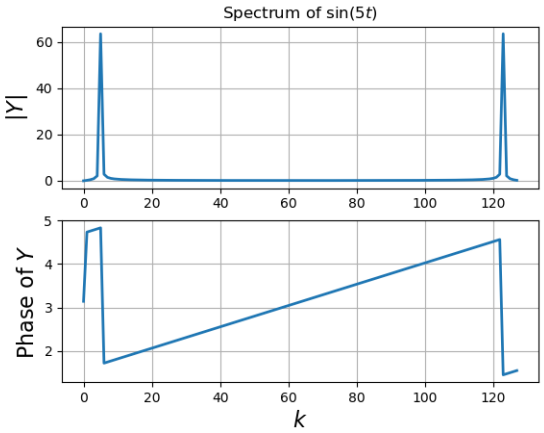
Below is the way, I have obtained the fourier transform of the sine function,

```
#Calculating fourier transform for sine function

x = linspace(0,2*pi,128)
y = sin(5*x)
Y = fft(y)
```

Below is the code for plotting,

```
#Plotting

plt.figure(0)
plt.subplot(2,1,1)
plt.plot(abs(Y),lw = 2)
plt.ylabel(r"$|Y|$",size = 16)
plt.title(r"Spectrum of $\sin(5t)$")
plt.grid(True)
plt.subplot(2,1,2)
plt.plot(unwrap(angle(Y)),lw = 2)
plt.ylabel(r"Phase of $Y$",size = 16)
plt.xlabel(r"$k$",size = 16)
plt.grid(True)
plt.savefig("sine_fourier.png")
plt.show()
```

Following is the output graph I obtained,



From the above graph, we can infer the following,

Inferences:

1) We see that we get spikes as expected, but not where we expected. The spikes have a height of 64, not 0.5. We should divide by N to use it as a spectrum. The phase at the spikes have a phase difference of π, which means they are opposite signs, which is correct. The actual phase at the spikes is near but not exactly correct.

2) We haven't yet got the frequency axis in place.

3) There is energy at nearby frequencies as well.

2.1.3 Determining the errors in the implementation and rectifying the fourier graphs for the sine function :

For the above fourier transform,

The problem is that the DFT treats the position axis as another frequency axis. So it expects the vector to be on the unit circle starting at 1. Our position vector started at 0 and went to 2π, which is correct. The fft gave an answer in the same value. So we need to shift the π to 2π portion to the left as it represents negative frequency. This can be done with a command called 'fftshift'. The second thing that is wrong is that both 0 and 2π are the same point. So our 128 points need to stop at the point just before 2π. The best way to do this is to create a vector of 129 values and drop the last one like,

x = linspace(0,2*pi,129);

x = x[:-1]

The ω axis has a highest frequency corresponding to N. This is because we have N points between 0 and 2π. So

Δx = $\frac{2\pi}{N}$,

and the sampling frequency becomes N/2π. Thus,

ϖ_{max} = N

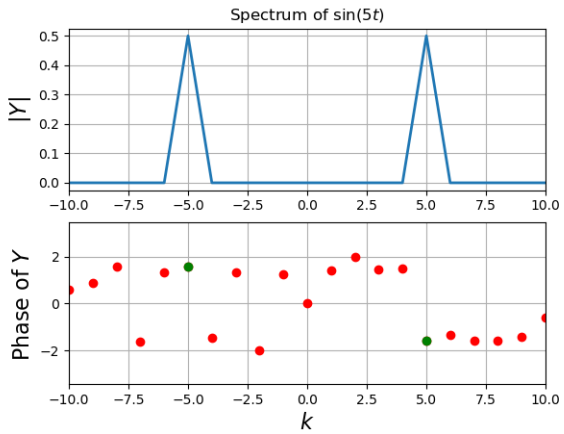
The actual frequencies go upto half that frequency only. Below is the update code for the above stuff,

```
#Correcting the errors and rectifying for the sine function

x = linspace(0,2*pi,129)
x = x[:-1]
y = sin(5*x)
Y = fftshift(fft(y))/128.0
w = linspace(-64,63,128)
```

Below is the code for plotting,

```
#Plotting

plt.figure(1)
plt.subplot(2,1,1)
plt.plot(w,abs(Y),lw = 2)
xlim([-10,10])
plt.ylabel(r"$|Y|$",size = 16)
plt.title(r"Spectrum of $\sin(5t)$")
plt.grid(True)
plt.subplot(2,1,2)
plt.plot(w,angle(Y),'ro',lw = 2)
ii = where(abs(Y)>1e-3)
plt.plot(w[ii],angle(Y[ii]),'go',lw = 2)
xlim([-10,10])
plt.ylabel(r"Phase of $Y$",size = 16)
plt.xlabel(r"$k$",size = 16)
plt.grid(True)
plt.savefig("sine_fourier2.png")
plt.show()
```

Shown below are the obatined plots for the updated code,



Now, things have improved. From the above graph, we see that,
Inferences:

1) The peaks are exactly at 0.5, and the remaining values are zero to machine precision. Their position along the x axis is correct as our function is sin (5x) and we expect a

spike at $\omega = \pm 5$.

2)The phase at the peak is meaningful and it is $\frac{\pi}{2}$ and $\frac{-\pi}{2}$ for the two peaks. This has been shown with green coour on the phase plot. The peak for $\omega = 5$ should be 0.5/ j, or

$0.5e^{-\frac{j\pi}{2}}$.

2.1.4 Obtaining the fourier plot for an amplitude modulated function :

We now look at AM modulation. The function we want to analyse is,

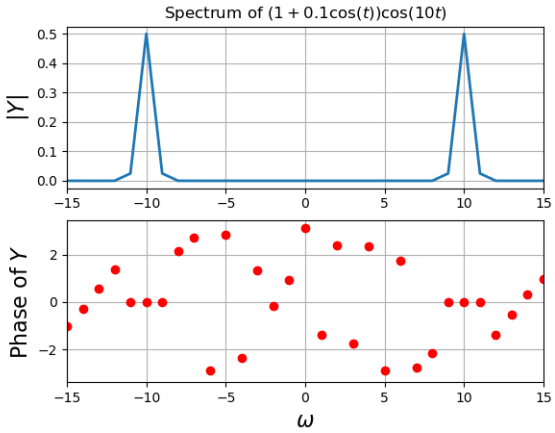
f(t) = (1 + 0.1 cos(t))cos (10t)

Again, we use 128 points and generate the spectrum using the above code, only changing the definition of f (t). Below code shows how we do it,

```
#Obtaining fourier plot for an amplitude
modulated signal

t = linspace(0,2*pi,129)
t = t[:-1]
y = (1+0.1*cos(t))*cos(10*t)
Y = fftshift(fft(y))/128.0
w = linspace(-64,63,128)
```

Below is the code for plotting,

```
#Plotting

plt.figure(2)
plt.subplot(2,1,1)
plt.plot(w,abs(Y),lw = 2)
plt.xlim([-15,15])
plt.ylabel(r"$|Y|$",size = 16)
plt.title(r"Spectrum of $\left(1+0.1\cos\
left(t\right)\right)\cos\left(10t\right)$")
plt.grid(True)
plt.subplot(2,1,2)
plt.plot(w,angle(Y),'ro',lw = 2)
plt.xlim([-15,15])
plt.ylabel(r"Phase of $Y$",size = 16)
plt.xlabel(r"$\omega$",size = 16)
plt.grid(True)
plt.savefig("amp_modulation.png")
plt.show()
```



The above figure is my obtained plot. It is not what we expected. Following are the inferences we can draw,

Inferences:

1) There are three non-zero amplitudes on both sides, as expected. But not three spikes, just a broader single spike.

2) All have zero phase, since we have cosines.

2.1.5 Determining the errors in the implementation and rectifying the fourier graphs for the amplitude modulated function :

In above question, we did not allow for enough frequencies. Let us use 512 samples instead.

But, if we put more points between 0 and 2π,

t = linspace(0,2*pi,513);

t = t[:-1]

it will give the same spacing but a high sampling frequency.

What we need to do is to stretch the t axis as,

t = linspace(-4*pi,4*pi,513);

t = t[:-1]

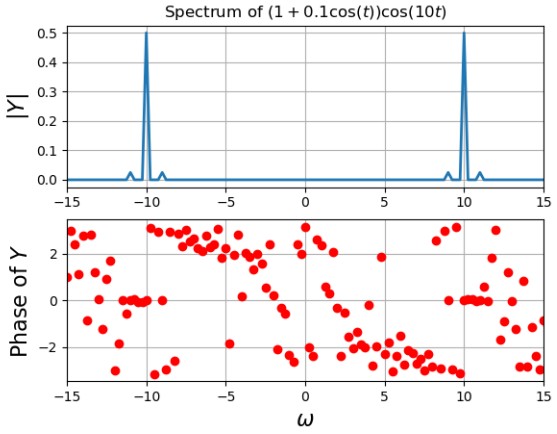
This will give us tighter spacing between frequency samples, but the same time spacing - i.e., the same sampling frequency. Below is the code for it,

```
#Correcting the errors and rectifying for the
amplitude modulated function

t = linspace(-4*pi,4*pi,513)
t = t[:-1]
y = (1+0.1*cos(t))*cos(10*t)
Y = fftshift(fft(y))/512.0
w = linspace(-64,64,513)
w = w[:-1]
```

Following is the way I plotted,

```
#Plotting

plt.figure(3)
plt.subplot(2,1,1)
plt.plot(w,abs(Y),lw = 2)
plt.xlim([-15,15])
plt.ylabel(r"$|Y|$",size = 16)
plt.title(r"Spectrum of $\left(1+0.1\cos\
left(t\right)\right)\cos\left(10t\right)$")
plt.grid(True)
plt.subplot(2,1,2)
plt.plot(w,angle(Y),'ro',lw = 2)
plt.xlim([-15,15])
plt.ylabel(r"Phase of $Y$",size = 16)
plt.xlabel(r"$\omega$",size = 16)
plt.grid(True)
plt.savefig("amp_modulation2.png")
plt.show()
```

After updating, I got the following plot,



From the above graph, we can draw the following inferences,

Inferences:

1) We have three clear spikes. The location of the spikes are 9, 10 and 11 radians per sec on the right side of zero. The height of the side bands come from,

0.1cos(10t)cos(t) = 0.05(cos(11t) + cos(9t))
= 0.025 (e^{11tj} + e^{9tj} + e^{-11tj} + e^{-9tj}

2) The phases at those spikes are zero to machine precision.

3) All the amplitudes are real and positive as seen in the phase plot.

2.2 Obtaining the spectrum of some cubic sinusoidal functions,

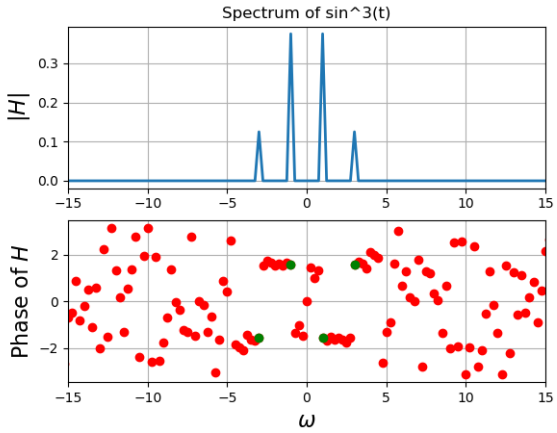
2.2.1 Obtaining the spectrum of $\sin^3 t$:

In this question we have $\sin^3 t$ as our function whose fourier transform I calculated with the following code,

```
#Obtaining fourier transform of sin^3(t)

t = linspace(-4*pi,4*pi,513)
t = t[:-1]
y = sin(t)*sin(t)*sin(t)
Y = fftshift(fft(y))/512.0
w = linspace(-64,64,513)
w = w[:-1]
```

Below is the way I have plotted,

```
#Plotting

plt.figure(4)
plt.subplot(2,1,1)
plt.plot(w,abs(Y),lw = 2)
plt.xlim([-15,15])
plt.ylabel(r"$|H|$",size = 16)
plt.title(r"Spectrum of sin^3(t)")
plt.grid(True)
plt.subplot(2,1,2)
plt.plot(w,angle(Y),'ro',lw = 2)
ii = where(abs(Y)>1e-3)
plt.plot(w[ii],angle(Y[ii]),'go',lw = 2)
plt.xlim([-15,15])
plt.ylabel(r"Phase of $H$",size = 16)
plt.xlabel(r"$\omega$",size = 16)
plt.grid(True)
plt.savefig("sine_cube_fourier.png")
plt.show()
```



Above is the plot for fourier transform of $\sin^3(t)$.

As we know,

$h(t) = \sin^3(t) = \frac{3sin(t)-sin(3t)}{4}$

$= \frac{3}{8j}(e^{jt}\text{-}e^{-jt})$ - $\frac{1}{8j}(e^{3jt}\text{-}e^{-3jt})$

Taking fourier transform of it, we get,

$H(j\varpi) = \frac{3}{8j}[\delta(\varpi - 1)$ - $\delta(\varpi + 1)]$ - $\frac{1}{8j}[\delta(\varpi - 3)$ - $\delta(\varpi + 3))]$

So in the graph, for $|H(j\varpi)|$, we got the peaks of the repsonse at ϖ = 1, -1, 3 and -3 as expected. And their magnitudes are 0.325, 0.325, 0.125 and 0.125 which was also found out from theory.

As far as the phase plot is concerned, we have,

1) A symmetric phase distribution corresponding to the symmetric function,

2) We also see that the phase is equal and opposite at the points of the peaks of the magnitude plot,

3) The other random points are obtained due to the python's fourier function.

So, we have got the plots as we had expected them to be.

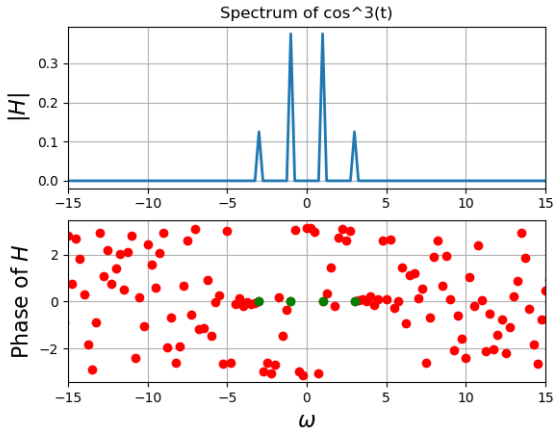
2.2.2 Obtaining the spectrum of $\cos^3(t)$:

In this question we have $\cos^3 t$ as our function whose fourier transform I calculated with the following code,

```
#Obtaining fourier transform of cos^3(t)

t = linspace(-4*pi,4*pi,513)
t = t[:-1]
y = cos(t)*cos(t)*cos(t)
Y = fftshift(fft(y))/512.0
w = linspace(-64,64,513)
w = w[:-1]
```

Below is the way I have plotted,

```
#Plotting

plt.figure(5)
plt.subplot(2,1,1)
plt.plot(w,abs(Y),lw = 2)
plt.xlim([-15,15])
plt.ylabel(r"$|H|$",size = 16)
plt.title(r"Spectrum of cos^3(t)")
plt.grid(True)
plt.subplot(2,1,2)
plt.plot(w,angle(Y),'ro',lw = 2)
ii = where(abs(Y)>1e-3)
plt.plot(w[ii],angle(Y[ii]),'go',lw = 2)
plt.xlim([-15,15])
plt.ylabel(r"Phase of $H$",size = 16)
plt.xlabel(r"$\omega$",size = 16)
plt.grid(True)
plt.savefig("cosine_cube_fourier.png")
plt.show()
```



Above is the plot for fourier transform of $\cos^3(t)$.

As we know,

$h(t) = \cos^3(t) = \frac{3cos(t)+cos(3t)}{4}$

$= \frac{3}{8}(e^{jt}+e^{-jt}) - \frac{1}{8}(e^{3jt}+e^{-3jt})$

Taking fourier transform of it, we get,

$H(j\varpi) = \frac{3}{8}[\delta(\varpi-1) + \delta(\varpi+1)] - \frac{1}{8}[\delta(\varpi-3) + \delta(\varpi+3)]$

So in the graph, for $|H(j\varpi)|$, we got the peaks of the repsonse at ϖ = 1, -1, 3 and -3 as expected. And their magnitudes are 0.325, 0.325, 0.125 and 0.125 which was also found out from theory.

As far as the phase plot is concerned, we have,

1) A symmetric phase distribution corresponding to the symmetric function,

2) Same phase at the peaks of the magnitude plot,
3) The other random points are obtained due to the python's fourier function.
So, we have got the plots as we had expected them to be.

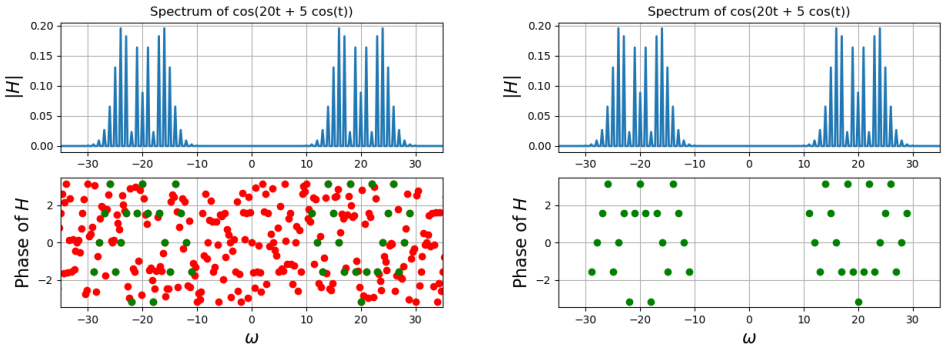
2.3 Obtaining the spectrum of cos(20t + 5cos(t)),

In this question, we have obtained the fourier transform of cos(20t + 5cos(t)),
Below is the code for it,

```
Obtaining the spectrum of cos(20t + 5 cos(t))

t = linspace(-4*pi,4*pi,513)
t = t[:-1]
y = cos(20*t + 5*cos(t))
Y = fftshift(fft(y))/512.0
w = linspace(-64,64,513)
w = w[:-1]
```

Below is the way I plotted,

```
#Plotting

plt.figure(6)
plt.subplot(2,1,1)
plt.plot(w,abs(Y),lw = 2)
plt.xlim([-35,35])
plt.ylabel(r"$|H|$",size = 16)
plt.title(r"Spectrum of cos(20t + 5 cos(t))")
plt.grid(True)
plt.subplot(2,1,2)
plt.plot(w,angle(Y),'ro',lw = 2)
ii = where(abs(Y)>1e-3)
plt.plot(w[ii],angle(Y[ii]),'go',lw = 2)
plt.xlim([-35,35])
plt.ylabel(r"Phase of $H$",size = 16)
plt.xlabel(r"$\omega$",size = 16)
plt.grid(True)
plt.savefig("complex_sinusoid2.png")
plt.show()
```



The above plot shows the magnitude and phase response of cos(20t + 5 cos(t)). We have expanded the x-axis in this case. As seen from the figure, we have a huge density of peaks from -30 to -10 and from 10 to 30.

The above magnitude plot is expected as the magnitude of the fourier transform turns out to be like it. The peculiar characteristics of thr graph are,

1) The maximum density of the plot is at ϖ= 20 and ϖ = -20.

2) The maximum amplitude that we get is 0.2.

In the above figure, I have marked the phase points where the magnitude is greater than 10^{-3} in green colour. From the phase plot we see,

1) The phase plot shows us that we have same phase at the peak points.

2.4 Obtaining the spectrum of $e^{-\frac{t^2}{2}}$,

We are obtaining the fourier transform of $e^{-\frac{t^2}{2}}$ in this question. Following code shows the way we have got it,

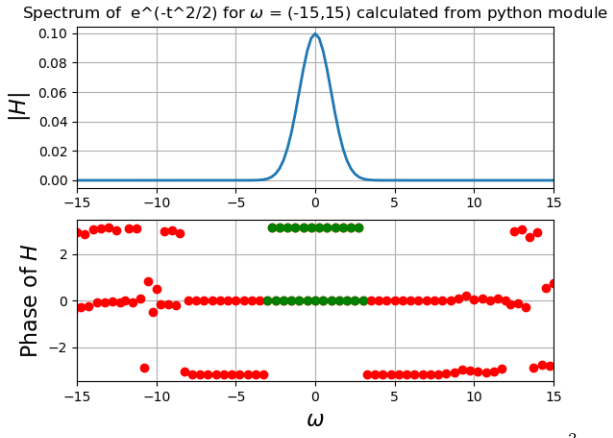
```
#From python fft function

N = 512
t = linspace(-(4*pi,4*pi,N+1)
t = t[:-1]
y = exp(-(t*t)/2)
Y = fftshift(fft(y))/N
w = linspace(-64,64,N+1)
w = w[:-1]
```

Below code shows the way we have plotted the spectrum for the Gaussian function using the in-built python module,

```
#Plotting

plt.figure(7)
plt.subplot(2,1,1)
plt.plot(w,abs(Y),lw = 2)
plt.xlim([-15,15])
```

```
plt.ylabel(r"$|H|$",size = 16)
plt.title(r"Spectrum of  e^(-t^2/2) for
$\omega$ = (-15,15) calculated from python module")
plt.grid(True)
plt.subplot(2,1,2)
plt.plot(w,angle(Y),'ro',lw = 2)
ii = where(abs(Y)>1e-3)
plt.plot(w[ii],angle(Y[ii]),'go',lw = 2)
plt.xlim([-15,15])
plt.ylabel(r"Phase of $H$",size = 16)
plt.xlabel(r"$\omega$",size = 16)
plt.grid(True)
plt.savefig("squared_exponential.png")
plt.show()
```

Following graph show the fourier spectrum of $e^{-\frac{t^2}{2}}$ calculated through the in-built function,



Following is the graph of fourier spectrum of $e^{-\frac{t^2}{2}}$ calculated manually,

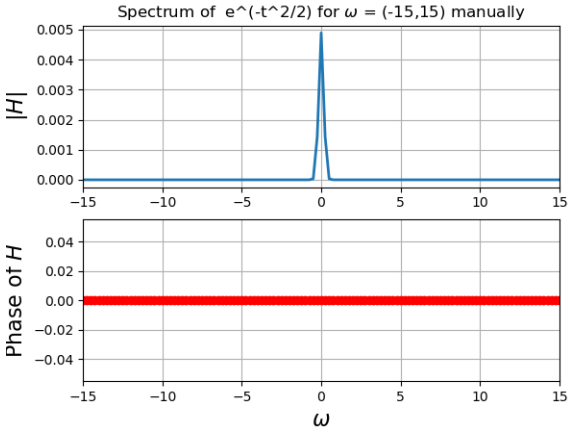
```
#Manually

def squared_exponential_fourier(m):
        return ((2*pi)**(0.5))*(e**(-2*pi*pi*m*m))/N

f = squared_exponential_fourier(w)
```

Below code shows the way we have plotted the spectrum for the Gaussian function using the in-built python module,

```
#Plotting

plt.figure(8)
plt.subplot(2,1,1)
```

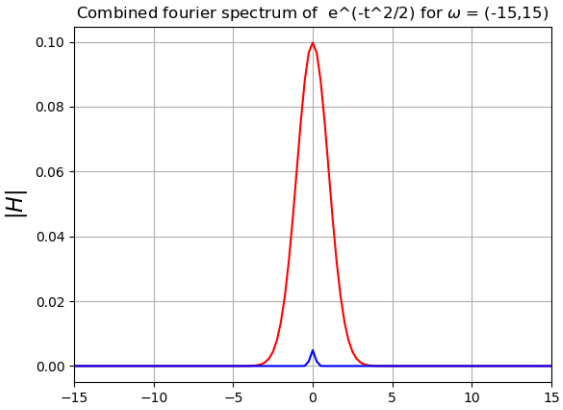
```
plt.plot(w,f,lw = 2)
plt.xlim([-15,15])
plt.ylabel(r"$|H|$",size = 16)
plt.title(r"Spectrum of  e^(-t^2/2)
for $\omega$ = (-15,15) manually")
plt.grid(True)
plt.subplot(2,1,2)
plt.plot(w,angle(f),'ro',lw = 2)
plt.xlim([-15,15])
plt.ylabel(r"Phase of $H$",size = 16)
plt.xlabel(r"$\omega$",size = 16)
plt.grid(True)
plt.savefig("squared_exponential2.png")
plt.show()
```

Following graph show the fourier spectrum of $e^{-\frac{t^2}{2}}$ calculated manually,

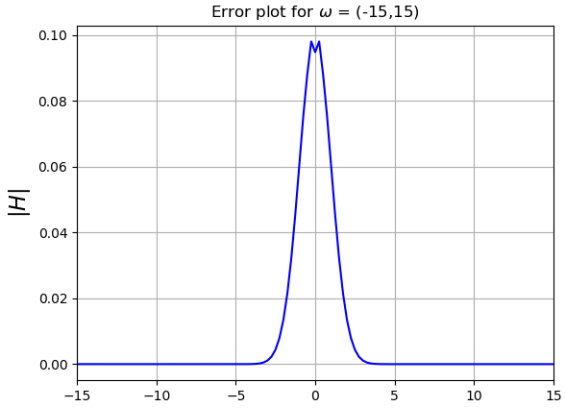


Now, I am comparing the magnitudes of the two Gaussian functions calculated using the two different methods,

```
#Combo plotting

plt.figure(9)
plt.plot(w,abs(Y),'r')
plt.plot(w,f,'b')
plt.xlim([-15,15])
plt.ylabel(r"$|H|$", size = 16)
plt.title(r"Combined fourier spectrum of  e^(-t^2/2)
for $\omega$ = (-15,15)")
plt.grid(True)
plt.savefig("squared_exponential_combo.png")
plt.show()
```



Above is the obtained plot for the function.
Below is the error plot I obtained for the above function,



Above is the plot I obtained for the error.
The maximum error I obtained was $\mathbf{1.354\ x\ 10^{-7}}$.