EE2703 Applied Programming Lab - Assignment 9

Name: Rajat Vadiraj Dwaraknath
Roll Number: EE16B033

April 17, 2018

1 Introduction

In this assignment, we analyse signals using the Fast Fourier transform, or the FFT for short. The FFT is a fast implementation of the Discrete Fourier transform(DFT). It runs in $\mathcal{O}(n \log n)$ time complexity. We find the FFTs of various types of signals using the numpy.fft module. We also attempt to approximate the continuous time fourier transform of a gaussian by windowing and sampling in time domain, and then taking the DFT. We iteratively increase window size and number of samples until we obtain an estimate of required accuracy.

2 FFT and IFFT

We perform the FFT and then the IFFT on a random array to see how well the original signal can be reconstruced.

```
from numpy.fft import *
rcParams['figure.figsize'] = 18,6
rcParams['font.size'] = 18
rcParams['text.usetex'] = True
x=rand(10)
X= fft(x)
y= ifft(X)
print("Original and reconstructed signal values:")
print(c_[x,y])
print("Max absolute error in reconstruction: {}".format(abs(x-y).max()))
```

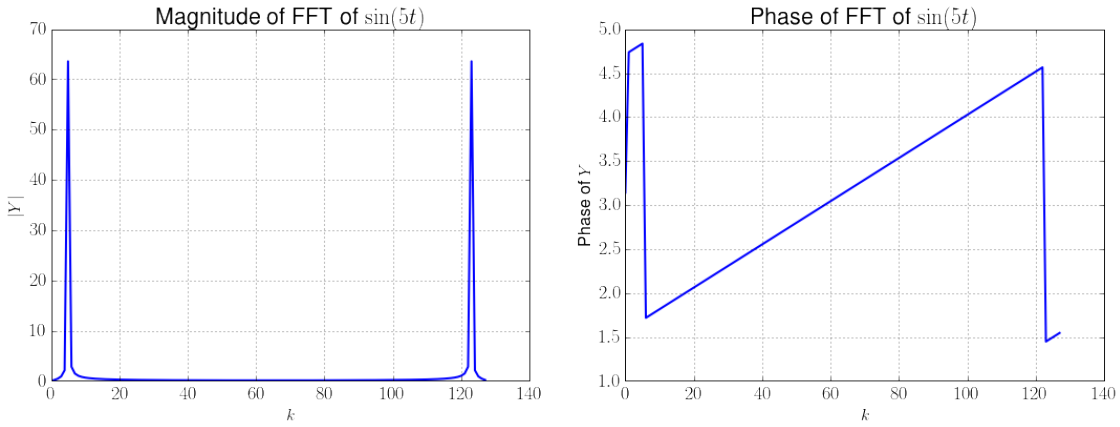
```
Original and reconstructed signal values:
[[ 0.12525382 +0.00000000e+00j  0.12525382 +0.00000000e+00j]
 [ 0.79069516 +0.00000000e+00j  0.79069516 +3.38618023e-16j]
 [ 0.46353347 +0.00000000e+00j  0.46353347 +9.55774994e-17j]
 [ 0.08177623 +0.00000000e+00j  0.08177623 -1.71584784e-16j]
 [ 0.53485467 +0.00000000e+00j  0.53485467 -4.41723755e-16j]
 [ 0.09175224 +0.00000000e+00j  0.09175224 -1.80948529e-16j]
 [ 0.11181523 +0.00000000e+00j  0.11181523 +2.11154472e-18j]
 [ 0.59301897 +0.00000000e+00j  0.59301897 +1.22315953e-16j]
 [ 0.11265585 +0.00000000e+00j  0.11265585 +1.09678038e-17j]
```

```
 [ 0.01678895 +0.00000000e+00j  0.01678895 +2.24666244e-16j]]
Max absolute error in reconstruction: 4.417237552752643e-16
```

An error of order 10^{-15} is present due to numerical inaccuracies in representations. We also observe that the reconstructed signal has some very small imaginary parts. Otherwise, we can see that the reconstruction is almost perfect.

3 Spectrum of sin(5t)

First, the examples given in the assignment are worked through:



To resolve the issues above like frequency scale and magnitude scale, a helper function to plot the magnitude and phase of the DFT of an arbitrary continuous function is written below. The function is first sampled to obtain the discrete periodic sequence whose DFT is then found.

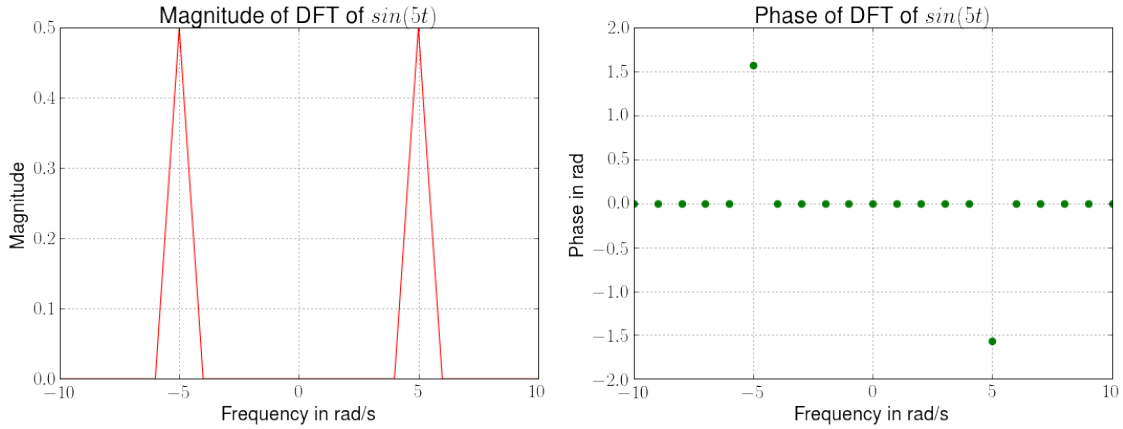
```
def plotFFT(func,t_range=(0,2*pi),points=128,tol=1e-5,
            func_name=None,unwrap_=True,wlim=(-10,10),scatter_size=40,
           iff=False):
    """Plot the FFT of the given continuous function.

    func : the continuous function
    t_range : the time range over which to sample the function,
        exclusive of the last value
    points : number of samples
    tol : tolerance for setting phase to 0 when magnitude is low
    func_name : name of the function
    unwrap : whether to unwrap phase
    wlim : range of frequencies for the plots, give None for all frequencies
    scatter_size : size of scatter plot points
    iff: whether to do an ifftshift on the time range

    Returns:
```

```
    numpy array containing the FFT, after being shifted and normalized.
    """

    # default name for function
    if func_name == None:
        func_name = func.__name__

    # time points to sample
    t = linspace(*t_range,points+1)[:-1]
    T = t_range[1]-t_range[0]
    samplingfreq = points/T

    if iff:
        t = ifftshift(t)

    # corresponding frequencies of the sampled signal
    w = linspace(-pi,pi,points+1)[:-1]
    w = w*samplingfreq

    # find fft
    y = func(t)
    Y =  fftshift( fft(y))/points

    # get phase
    ph = angle(Y)
    if unwrap_:
        ph = unwrap(ph)

    # get mag
    mag = abs(Y)

    # clean up phase where mag is sufficiently close to 0
    ph[where(mag<tol)]=0

    # plot
    fig,axes = subplots(1,2)
    ax1,ax2 = axes

    # magnitude
    ax1.set_title("Magnitude of DFT of {}".format(func_name))
    ax1.set_xlabel("Frequency in rad/s")
    ax1.set_ylabel("Magnitude")
    ax1.plot(w,mag,color='red')#,s=scatter_size)
    ax1.set_xlim(wlim)
    ax1.grid()

    # phase
    ax2.set_title("Phase of DFT of {}".format(func_name))
```

```
    ax2.set_xlabel("Frequency in rad/s")
    ax2.set_ylabel("Phase in rad")
    ax2.scatter(w,ph,color='green',s=scatter_size)
    ax2.set_xlim(wlim)
    ax2.grid()

    show()
    return w,Y
```

Let us use this function to find the DFT of $sin(5t)$:

```
def sin5(t):
    return sin(5*t)
```



As expected, there are two peaks at frequencies of 5 and -5, of height 0.5. The phases of the peaks correspond to the $\frac{1}{j}$ and $\frac{-1}{j}$ multiplying the exponentials in the expansion:

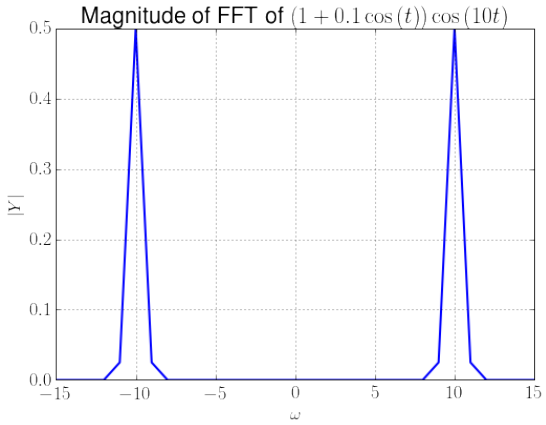
$$sin(5t) = 0.5(\frac{e^{5t}}{j} - \frac{e^{-5t}}{j})$$

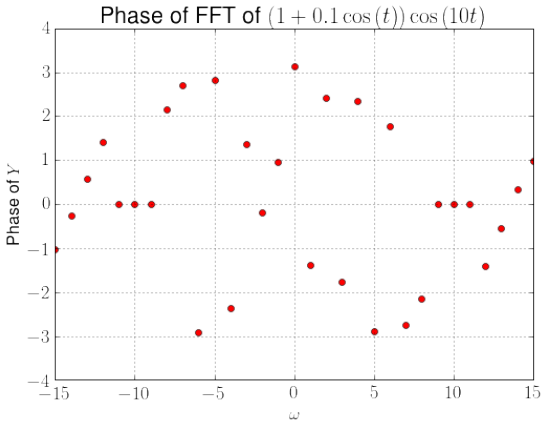
4 Amplitude modulation

Let's find the DFT of the AM modulated wave:

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

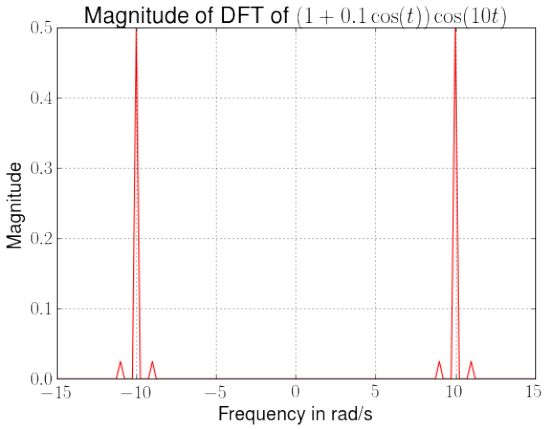
First the examples in the assignment are worked through:

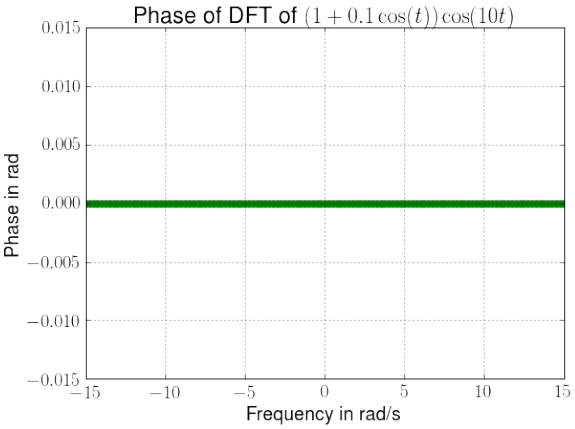




Using a finer frequency resolution:

```
def amMod(t):
    return (1+0.1*cos(t))*cos(10*t)
```





The phase is 0 everywhere as expected, as the terms are only cosine waveforms which have real DFTs. The magnitude has two large peaks at frequency of 10 corresponding to the carrier cosine wave. This wave is then multiplied by a low amplitude low frequency cosine in the time domain. This corresponds to convolution in the frequency domain. This results in the two sidebands next to each of the carrier bands.

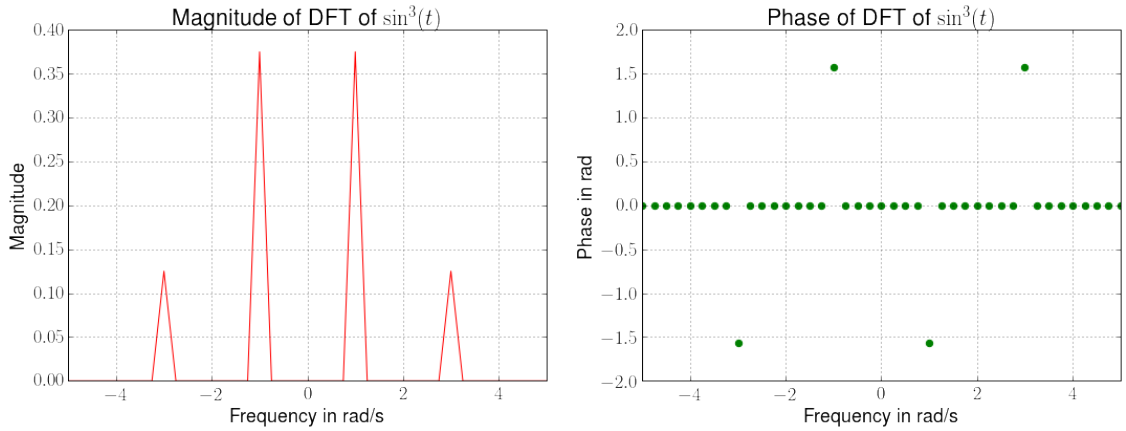
5 Spectrum of $\sin^3(t)$

Using the following identity:

$$\sin^3(t) = \frac{3}{4}\sin(t) - \frac{1}{4}\sin(3t)$$

We expect two sets of peaks at frequencies of 1 and 3, with heights corresponding to half of 0.75 and 0.25.

```
def sin3(t):
    return sin(t)**3
```



We observe the peaks in the magnitude at the expected frequencies of 1 and 3, along with the expected amplitudes. The phases of the peaks are also in agreement with what is expected(one is a positive sine while the other is a negative sine).

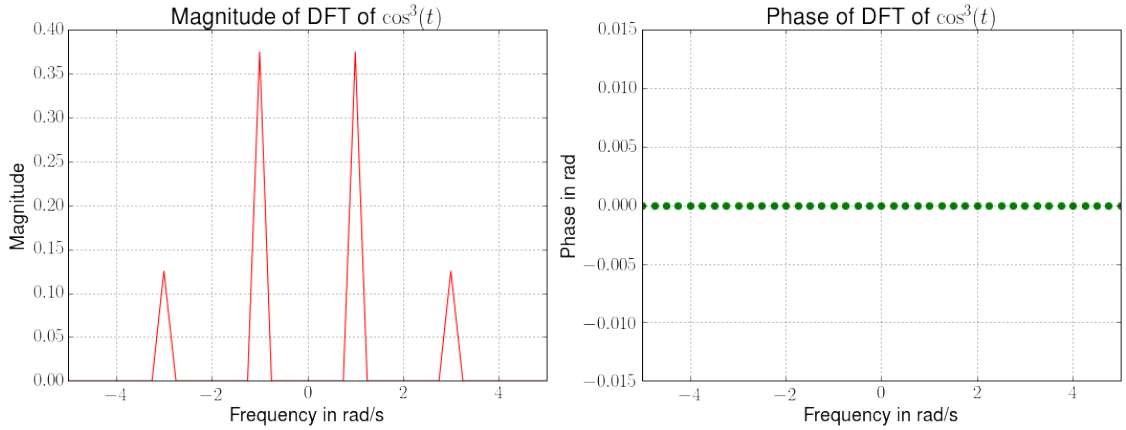
6 Spectrum of $\cos^3(t)$

Using the following identity:

$$\cos^3(t) = \frac{3}{4}\cos(t) + \frac{1}{4}\cos(3t)$$

We expect two sets of peaks at frequencies of 1 and 3, with heights corresponding to half of 0.75 and 0.25.

```
def cos3(t):
    return cos(t)**3
```



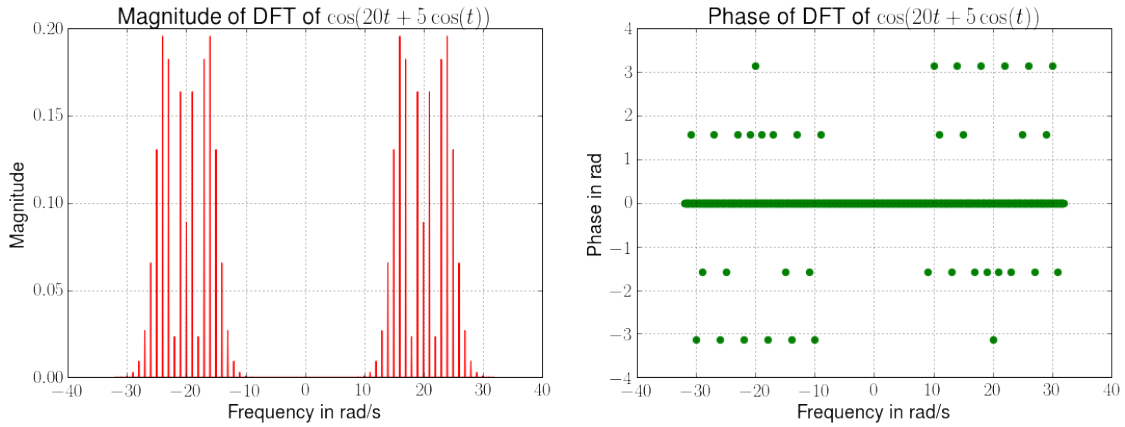
We observe the peaks in the magnitude at the expected frequencies of 1 and 3, along with the expected amplitudes. The phases of the peaks are also in agreement with what is expected(both are positive cosines).

7 Frequency modulation

We find the DFT of the following frequency modulated signal:

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

```
def fm(t):
    return cos(20*t + 5*cos(t))
```



There are many more sidebands compared to AM in the case of FM. Most of the energy of the signal is also present in these sidebands rather than the carrier band in the case of AM.

8 Continuous time Fourier Transform of Gaussian

The fourier transform of a signal $x(t)$ is defined as follows:

$$X(\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt$$

We can approximate this by the fourier transform of the windowed version of the signal $x(t)$, with a sufficiently large window. Let the window be of size T. We get:

$$X(\omega) \approx \frac{1}{2\pi}\int_{-\frac{T}{2}}^{\frac{T}{2}} x(t)e^{-j\omega t}dt$$

We can write the integral approximately as a Reimann sum:

$$X(\omega) \approx \frac{\Delta t}{2\pi}\sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} x(n\Delta t)e^{-j\omega n\Delta t}$$

Where we divide the integration domain into N parts (assume N is even), each of width $\Delta t = \frac{T}{N}$.

Now, we sample our spectrum with a sampling period in the frequency domain of $\Delta\omega = \frac{2\pi}{T}$, which makes our continuous time signal periodic with period equal to the window size T. Our transform then becomes:

$$X(k\Delta\omega) \approx \frac{\Delta t}{2\pi} \sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} x(n\Delta t) e^{-jkn\Delta\omega\Delta t}$$

Which simplifies to:

$$X(k\Delta\omega) \approx \frac{\Delta t}{2\pi} \sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} x(n\Delta t) e^{-j\frac{2\pi}{N}kn}$$

Noticing that the summation is of the form of a DFT, we can finally write:

$$X(k\Delta\omega) \approx \frac{\Delta t}{2\pi} DFT\{x(n\Delta t)\}$$

The two approximations we made were:

- The fourier transform of the windowed signal is approximately the same as that of the original.
- The integral was approximated as a Reimann sum.

We can improve these approximations by making the window size T larger, and by decreasing the time domain sampling period or increasing the number of samples N. We implement this in an iterative algorithm in the next part.

The analytical expression of the fourier transform of the gaussian:

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

Was found as:

$$X(j\omega) = \frac{1}{\sqrt{2\pi}} e^{\frac{-\omega^2}{2}}$$

We also compare the numerical results with the expected analytical expression.

```
def gauss(t):
    return exp(-t**2/2)

def expected_gaussFT(w):
    return 1/sqrt(2*pi) * exp(-w**2/2)
```

A function to estimate this continuous time fourier transform by windowing the input function, sampling, and then finding the DFT is written below. The function iteratively increases window size and sample number until the consecutive total absolute error between estimates reduces below a given threshold.

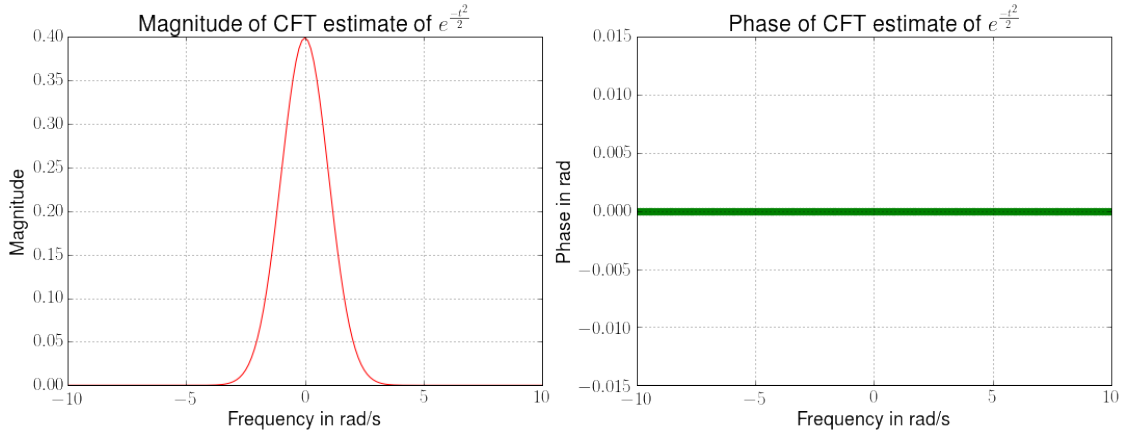
```
def estimateCTFT(func, tol=1e-6,time_samples=128, true_func=None,
                 func_name=None, wlim=None, scatter_size=40):
    """Estimate the continuous time Fourier Transform of the given function
    by finding the DFT of a sampled window of the function. The magnitude and
    phase of the estimate are also plotted.

    The window size and sample number are doubled until the consecutive
    total absolute error between two estimates is less than the given tolerance.

    time_samples : the initial number of samples to start with
    true_func : A function which is the analytical CTFT of the given function.
                Used to compare the estimate results with the true results.

    Returns frequencies and the CTFT estimate at those frequencies.
    """

    if func_name == None:
        func_name = func.__name__


    T = 8*pi
    N = time_samples
    Xold = 0
    error = tol+1
    iters=0

    while error>tol:

        delta_t = T/N # time resolution
        delta_w = 2*pi/T # frequency resolution

        W = N*delta_w # total frequency window size

        t = linspace(-T/2,T/2,N+1)[:-1] # time points
        w = linspace(-W/2,W/2,N+1)[:-1] # freq points

        x = func(t)

        # find DFT and normalize
        # note that ifftshift is used to prevent artifacts in the
        # phase of the result due to time domain shifting
        X = delta_t/(2*pi) * fftshift(fft(ifftshift(x)))

        error = sum(abs(X[::2]-Xold))

        Xold = X
        N *= 2 # number of samples
        T *= 2 # total time window size
```

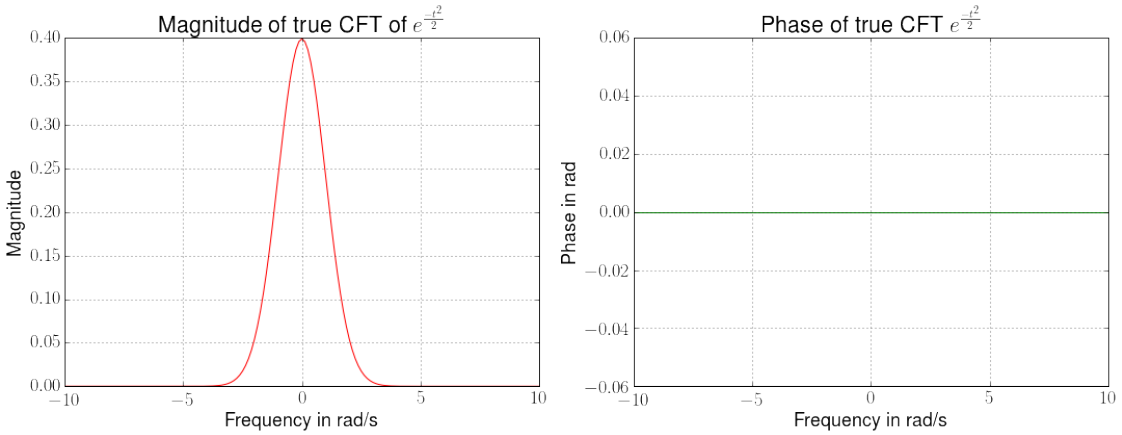
```
    iters+=1


print("Estimated error after {} iterations: {}".format(iters, error))
print("Time range : ({:.4f}, {:.4f})".format(-T/2,T/2))
print("Time resolution : {:.4f}".format(delta_t))
print("Frequency resolution : {:.4f}".format(delta_w))

if true_func != None:
    true_error = sum(abs(X-true_func(w)))
    print("True error: {}".format(true_error))

mag = abs(X)
ph = angle(X)
ph[where(mag<tol)]=0

# plot estimate
fig,axes = subplots(1,2)
ax1,ax2 = axes

# magnitude
ax1.set_title("Magnitude of CFT estimate of {}".format(func_name))
ax1.set_xlabel("Frequency in rad/s")
ax1.set_ylabel("Magnitude")
ax1.plot(w,mag,color='red')#,s=scatter_size)
ax1.set_xlim(wlim)
ax1.grid()

# phase
ax2.set_title("Phase of CFT estimate of {}".format(func_name))
ax2.set_xlabel("Frequency in rad/s")
ax2.set_ylabel("Phase in rad")
ax2.scatter(w,ph,color='green',s=scatter_size)
ax2.set_xlim(wlim)
ax2.grid()

show()

if true_func != None:

    X_ = true_func(w)

    mag = abs(X_)
    ph = angle(X_)
    ph[where(mag<tol)]=0

    # plot true
    fig,axes = subplots(1,2)
```

```
        ax1,ax2 = axes

        # magnitude
        ax1.set_title("Magnitude of true CFT of {}".format(func_name))
        ax1.set_xlabel("Frequency in rad/s")
        ax1.set_ylabel("Magnitude")
        ax1.plot(w,mag,color='red')#,s=scatter_size)
        ax1.set_xlim(wlim)
        ax1.grid()

        # phase
        ax2.set_title("Phase of true CFT {}".format(func_name))
        ax2.set_xlabel("Frequency in rad/s")
        ax2.set_ylabel("Phase in rad")
        ax2.plot(w,ph,color='green')
        ax2.set_xlim(wlim)
        ax2.grid()

        show()
    return w,X
```

We use the above function to estimate the fourier transform of the given gaussian upto a tolerance of 10^{-6}.

```
Estimated error after 2 iterations: 8.430912440960026e-15
Time range : (-50.2655, 50.2655)
Time resolution : 0.1963
Frequency resolution : 0.1250
True error: 3.9543195419547677e-14
```





9 Conclusions

- From the above pairs of plots, it is clear that with a sufficiently large window size and sampling rate, the DFT approximates the CTFT of the gaussian.
- This is because the magnitude of the gaussian quickly approaches 0 for large values of time. This means that there is lesser frequency domain aliasing due to windowing. This can be interpreted as follows:
- Windowing in time is equivalent to convolution with a sinc in frequency domain. A large enough window means that the sinc is tall and thin. This tall and thin sinc is approximately equivalent to a delta function for a sufficiently large window. This means that convolution with this sinc does not change the spectrum much.
- Sampling after windowing is done so that the DFT can be calculated using the Fast Fourier Transform. This is then a sampled version of the DTFT of the sampled time domain signal. With sufficiently large sampling rates, this approximates the CTFT of the original time domain signal.
- This process is done on the gaussian and the results are in agreement with what is expected.