# Assignment 9: Spectra of non-periodic signals

Abhishek Sekar
EE18B067

May 7, 2020

## 1 Introduction

- We explore digital fourier transform (DFT) with windowing. This is used to make the signal square integrable. Also done to ensure the function goes sufficiently rapidly towards 0. Also to make an infinitely long signal to a finite signal, since to take DFT we need a finite aperiodic signal.

- Windowing a simple waveform like $cos(\omega t)$, causes its fourier transform to develop non-zero value at frequencies other than $\omega$. This is called *Spectral Leakage*. This can cause in some applications the stronger peak to smear the weaker parts. So choosing proper windowing functions is essential. The windowing function we use here is called **Hamming window** which is generally used in $narrow-band\ applications$. The hamming window attenuates the high frequency components that cause the discontinuities. The hamming window function is given by

$$x[n] = 0.54 + 0.46cos(\frac{2\pi n}{N-1}) \tag{1}$$

## 2 Worked Examples

The worked examples in the assignment are given below: Spectrum of $sin(\sqrt{2}t)$ is given below

```
#worked examples
#sin(sqrt(2)t) attempt 1
t=linspace(-pi,pi,65)
t=t[:-1]
dt=t[1]-t[0]
fmax=1/dt
y=sin(sqrt(2)*t)
y[0]=0 #the centrepoint is set to 0
y=fftshift(y) #make y start from y(t=0)
Y=fftshift(fft(y))/64.0
w=linspace(-pi*fmax,pi*fmax,65)
w=w[:-1]

subplot(2,1,1)
plot(w, abs(Y), lw=2)
xlim([-10,10])
ylabel(r"$|Y| ->$", size=16)
title(r"Spectrum of $\sin\left(\sqrt{2}t\right)$")
grid()
subplot(2,1,2)
plot(w,angle(Y),'ro',lw=2)
xlim([-10,10])
ylabel(r"Phase of $Y$ ->",size=16)
xlabel(r"$\omega$ ->",size=16)
grid()
show()

#plotting the function over multiple time periods
t2=linspace(-3*pi,-pi,65);t2=t2[:-1]
t3=linspace(pi,3*pi,65);t3=t3[:-1]
```

```
plot(t, sin(sqrt(2)*t), color='red')
plot(t2,sin(sqrt(2)*t2), color='blue')
plot(t3,sin(sqrt(2)*t3), color='blue')
xlabel('time ->')
ylabel(r'$\sin\left(\sqrt{2}t\right)$ ->')
title(r'plot of $\sin(\sqrt{2}t)$')
grid()
show()

#replicating the blue portion alone over all the periods
plot(t,sin(sqrt(2)*t),'ro')
plot(t2,sin(sqrt(2)*t),'bo')
plot(t3,sin(sqrt(2)*t),'bo')
xlabel('time ->')
ylabel('y ->')
title(r'plot of $\sin(\sqrt{2}t)$ wrapping every 2$\pi$')
grid()
show()


#DFT of a ramp
z=t
z[0]=0
z=fftshift(z)
Z=fftshift(fft(z))/64
semilogx(abs(w),20*log10(abs(Z)),lw=2)
xlim([1,10])
ylim([-20,0])
xticks([1,2,5,10],["1","2","5","10"],size=16) #cool feature puts markers on t
ylabel(r"$|Y|$ (dB)",size=16)
title(r"Spectrum of a digital ramp")
xlabel(r"$\omega$",size=16)
grid()
show()

```

```
#testing the hamming window
n=arange(64)
wnd=fftshift(0.54+0.46*cos(2*pi*n/63))
y=sin(sqrt(2)*t1)*wnd
plot(t1,y,'bo',lw=2)
plot(t2,y,'ro',lw=2)
plot(t3,y,'ro',lw=2)
ylabel(r"$y$",size=16)
xlabel(r"$t$",size=16)
title(r"$\sin\left(\sqrt{2}t\right)\times w(t)$ with $t$ wrapping every $2\pi$
grid()
show()

#DFT of this sequence
y[0]=0 # the sample corresponding to -tmax should be set zero
y=fftshift(y) # make y start with y(t=0)
Y=fftshift(fft(y))/64.0
subplot(2,1,1)
plot(w,abs(Y),lw=2)
xlim([-8,8])
ylabel(r"$|Y|$",size=16)
title(r"Spectrum of $\sin\left(\sqrt{2}t\right)\times w(t)$")
grid(True)
subplot(2,1,2)
plot(w,angle(Y),'ro',lw=2)
xlim([-8,8])
ylabel(r"Phase of $Y$",size=16)
xlabel(r"$\omega$",size=16)
grid(True)
show()

#using more samples to improve accuracy
t3=linspace(-4*pi,4*pi,257);t3=t3[:-1]
dt1=t3[1]-t3[0]
fmax1=1/dt1
```

```
n=arange(256)
wnd=fftshift(0.54+0.46*cos(2*pi*n/256))
y=sin(sqrt(2)*t3)
y=y*wnd
y[0]=0 # the sample corresponding to -tmax should be set zero
y=fftshift(y) # make y start with y(t=0)
Y=fftshift(fft(y))/256.0
w=linspace(-pi*fmax1,pi*fmax1,257);w=w[:-1]
subplot(2,1,1)
plot(w,abs(Y),'b',w,abs(Y),'bo',lw=2)
xlim([-4,4])
ylabel(r"$|Y|$",size=16)
title(r"Spectrum of $\sin\left(\sqrt{2}t\right)\times w(t)$")
grid(True)
subplot(2,1,2)
plot(w,angle(Y),'ro',lw=2)
xlim([-4,4])
ylabel(r"Phase of $Y$",size=16)
xlabel(r"$\omega$",size=16)
grid(True)
show()

#worked examples end here
```

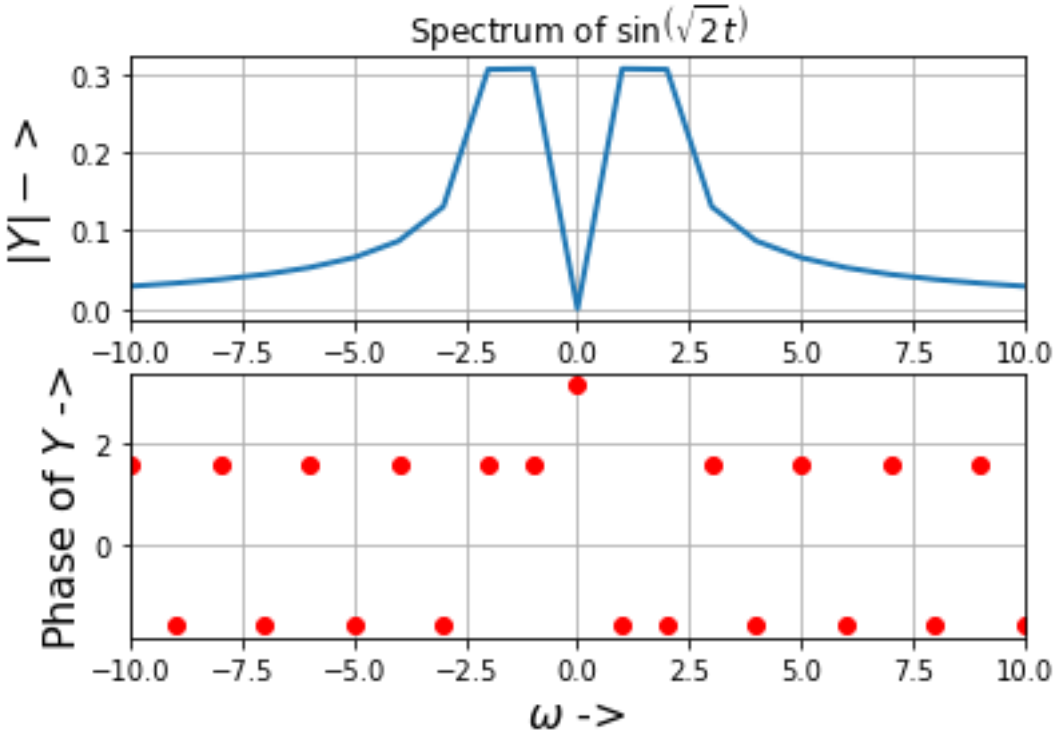


Figure 1: 'spectrum' of $\sin(\sqrt{2}t)$

Original function for which the DFT is required:

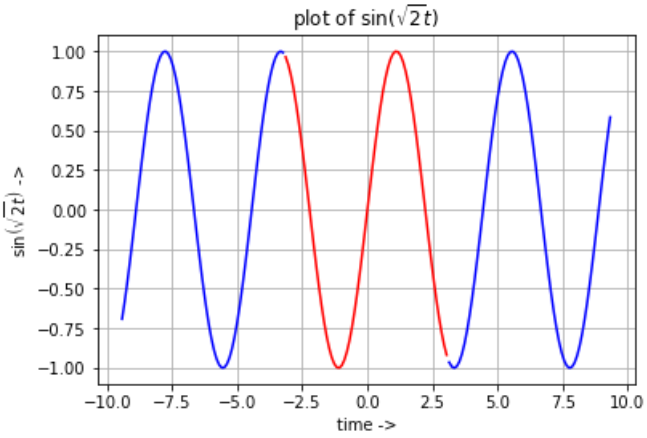


Figure 2: $\sin(\sqrt{2}t)$

Wrapped sine

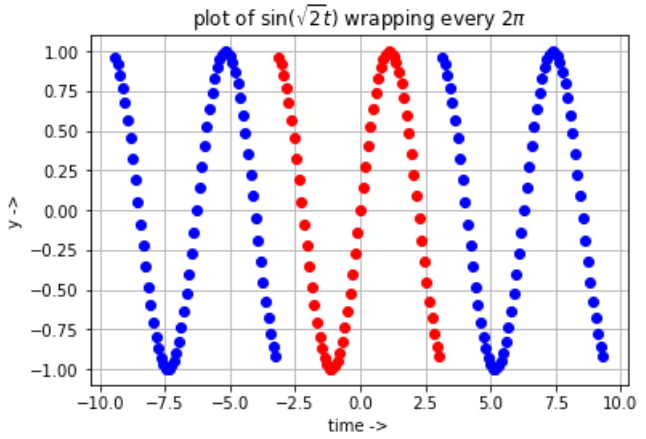


Figure 3: Wrapped $\sin(\sqrt{2}t)$

Discontinuities lead to non harmonic components in the FFT which decay as $\frac{1}{\omega}$. We can plot the spectrum of the periodic ramp to confirm this.

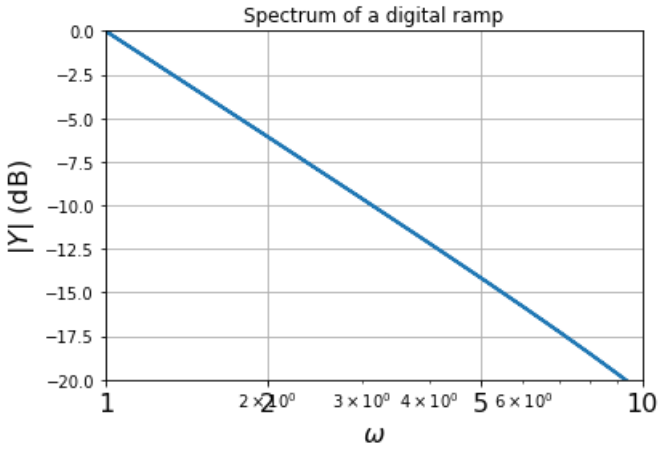


Figure 4: Spectrum of $\sin(\sqrt{2}t)$

The signal multiplied with the hamming window has its discontinuities nearly vanish as we expected.

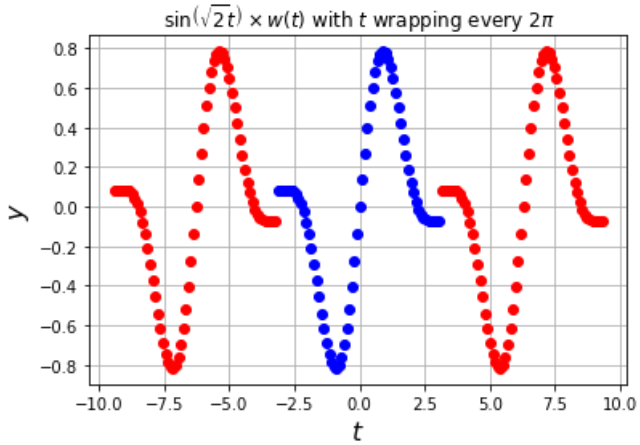


Figure 5: $\text{Sin}(\sqrt{2}t) * w(t)$

Spectrum obtained with a time period $2\pi$

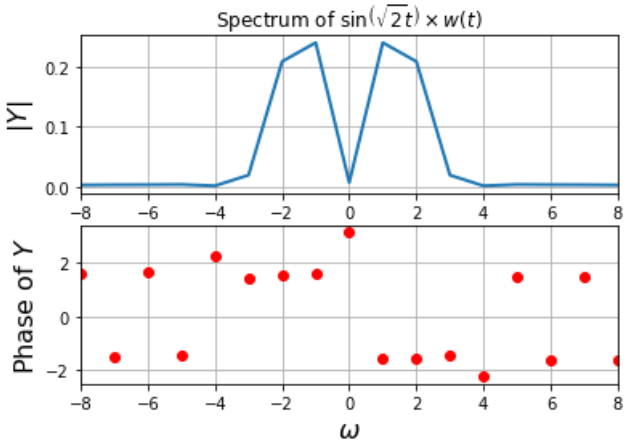


Figure 6: Spectrum of $\sin(\sqrt{2}t) * w(t)$

The spectrum that is obtained with a time period $8\pi$ has a slightly sharper peak as expected.

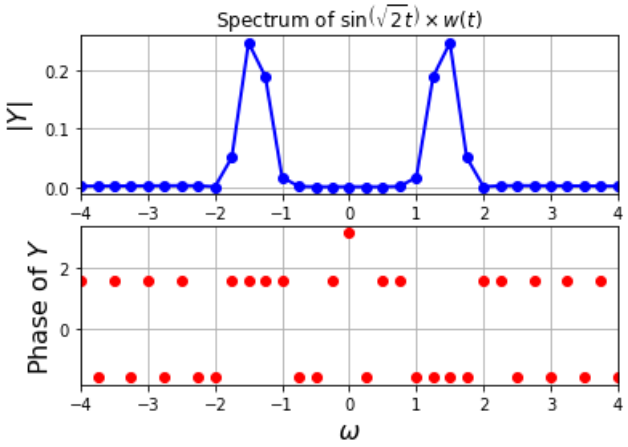


Figure 7: Spectrum of $\sin(\sqrt{2}t) * w(t)$

# 3 Python Code:Assignment Questions

## 3.1 Question 1:

- Consider the function $\cos^3(\omega_0 t)$. Obtain its spectrum for $\omega_0 = 0.86$ with and without a Hamming window.

```
  def selector(t,n,w=None,d=None):      #selector function
    if(n==1):
        return pow(cos(0.86*t),3)
    elif(n==2):
        return cos(16*(1.5+t/(2*pi))*t)
    elif(n==3):
        if(w!=None and d!=None):
            return cos(w*t+d)
    else:
            return pow(cos(0.86*t),3)

def window_fn(n,N):
    return (0.54+0.46*cos(2*pi*n/N))


'''
Function to find Discrete Fourier Transform
Arguments:
 low_lim,up_lim -> lower & upper limit for time
 no_points      -> Sampling rate
 n              -> function number
 wo             -> frequency
 d              -> phase difference
 noise_const    -> Gaussian contribution
 '''

def findFFT(low_lim,up_lim,no_points,n,window=True,wo=None,d=None,noise_const=
    t = linspace(low_lim,up_lim,no_points+1)[:-1]
    dt=t[1]-t[0]
    fmax=1/dt
    N = no_points
    y = selector(t,n,wo,d)+noise_const*randn(len(t)) #noise_const is only giv

    if(window):
```

```
35          n1=arange(N)
36          wnd=fftshift(window_fn(n1,N))
37          y=y*wnd
38
39      y[0]=0          # the sample corresponding to -tmax should be set zero
40      y=fftshift(y) # make y start with y(t=0)
41      Y=fftshift(fft(y))/N
42
43      w = linspace(-pi*fmax,pi*fmax,N+1)[:-1]
44      return t,Y,w
45  '''
46  Function to plot Magnitude and Phase spectrum for given function
47  Arguments:
48   t               -> time vector
49   Y               -> DFT computed
50   w               -> frequency vector
51   Xlims,Ylims     -> limits for x&y axis for spectrum
52   plot_title -> title of plot
53  '''
54
55  def plot_FFT(t,Y,w,Xlims,plot_title,dotted=False,Ylims=None):
56
57      subplot(2,1,1)
58
59      if(dotted):
60          plot(w,abs(Y),'bo',lw=2)
61      else:
62          plot(w,abs(Y),'b',lw=2)
63
64      xlim(Xlims)
65
66      ylabel(r"$|Y(\omega)| \to$")
67      title(plot_title)
68      grid(True)
69
```

```
    subplot(2,1,2)
    ii=where(abs(Y)>0.005)
    plot(w[ii],angle(Y[ii]),'go',lw=2)

    if(Ylims!=None):
        ylim(Ylims)

    xlim(Xlims)
    ylabel(r"$\angle Y(j\omega) \to$")
    xlabel(r"$\omega \to$")
    grid(True)
    show()

#Question 2 for cos^3((Wo)t)
t,Y,w = findFFT(-4*pi,4*pi,256,1,False)
Xlims = [-8,8]
plot_title = r"Spectrum of $\cos^3(\omega_o t)$ without windowing"
plot_FFT(t,Y,w,Xlims,plot_title)

t,Y,w = findFFT(-4*pi,4*pi,256,1,True)
Xlims = [-8,8]
plot_title = r"Spectrum of Windowed $\cos^3(\omega_o t)$"
plot_FFT(t,Y,w,Xlims,plot_title)
```

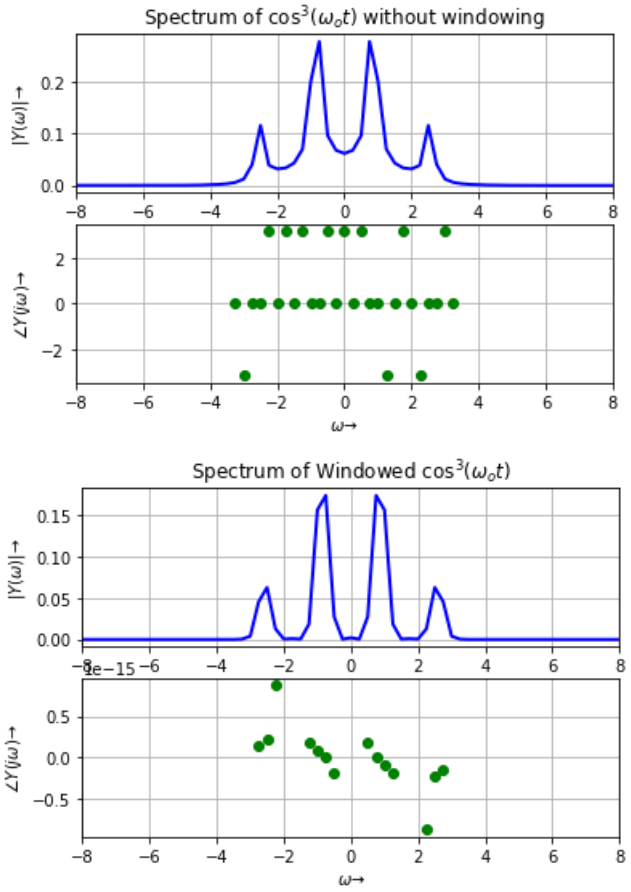


Figure 8: Spectrum of $\cos^3(\omega_o t)$ with and without windowing

### 3.1.1 Results and Discussion:

- Here we can see clear differences between the windowed fourier transform and fourier transform without the window function. Peak is being smeared by the windowing function but the high frequency components are attenuated by the window function. The *spectral leakage* can also be noticed.

### 3.2 Question 2:

- Write a program that will take a 128 element vector known to contain $\cos(\omega_0 t + \delta)$ for arbitrary $\delta$ and $0.5 < \omega_0 < 1.5$ where $\pi \leq t \leq \pi$.
- You have to extract the digital spectrum of the signal, find the two peaks at $\pm\omega_0$, and estimate $\omega_0$ and $\delta$.

#### 3.2.1 Estimating $\omega$ and $\delta$ from fourier spectrum

- According to the question the if the spectra is obtained, the resolution is not enough to obtain the $\omega_0$ directly. The peak will not be visible clearly because of the fact that resolution of the frequency axis is not enough. So a statistic is necessary to estimate value of $\omega_0$
- Which is essentially the weighted average of $\omega$ where $\omega$ is positive.
- Now, $\delta$ can be found by two ways :
- Least square fitting of

$$y(t) = A\cos(\omega_0 t) + B\sin(\omega_0 t) \tag{2}$$

- Minimizing L2-norm to find the coefficients $A, B$, we can compute $\delta$ by,

$$\delta = -\tan^{-1}(\frac{B}{A}) \tag{3}$$

- Another method which can be used is to find the phase of the discrete fourier transform at $\omega_o$ nearest to estimated $\omega$ using the above idea.
- This works because the phase of $\cos(\omega_o t + \delta)$ when $\delta = 0$ is 0, so when its not its $\delta$, so we can estimate it by this approach.
- The latter approach is used in this assignment.

---

```
#Question 3 and 4 estimating w and delta
def estimator(noise_const=0):   #function estimates the values of w and delta
    w_actual = np.random.uniform(0.5,1.5) #generates a random omega value
    delta_actual = (randn())              #random delta value
    t,Y,w = findFFT(-1*pi,1*pi,128,3,True,w_actual,delta_actual,noise_const)

```

```
    ii = where(w>=0)
    w_cal = sum(abs(Y[ii])**2*w[ii])/sum(abs(Y[ii])**2)   #do weighted averag
    i = abs(w-w_cal).argmin() #find w at which it is closest to the estimated
    delta_cal = angle(Y[i])   #find delta at that w0 since it'll be the phase
    print("Calculated value of w0 (to 4 decimal places) ",round(w_cal,4))
    print("Calculated value of delta (to 4 decimal places) ",round(delta_cal,4
    print("Actual value of w0 (to 4 decimal places) ",round(w_actual,4))
    print("Actual value of delta (to 4 decimal places)",round(delta_actual,4))
print("without noise")
estimator() #without noise
print("with noise")
estimator(0.1) #with noise
```

### 3.2.2 Results and Discussion:

- Calculated value of w0 1.168418390564658
- Calculated value of delta -1.2029756072701416
- Actual value of w0 1.123602120604056
- Actual value of delta -1.1985712967313407
- As we observe that actual $\omega$ & $\delta$ are generated using uniformly distributed random functions, and using the above center of mass statistic we get estimated ones close with error in the range of 3%

### 3.3 Question 3:

- Now we add **white gaussian noise** to data in $Q_3$. This can be generated by $randn()$ in python. The extent of this noise is 0.1 in amplitude (i.e., $0.1 * randn(N)$, where $N$ is the number of samples).
- Repeat the problem and find the $\omega_0$ and $\delta$

#### 3.3.1 Results and Discussion:

- Calculated value of w0 (to 4 decimal places) 1.789
- Calculated value of delta (to 4 decimal places) -0.6021
- Actual value of w0 (to 4 decimal places) 1.3481
- Actual value of delta (to 4 decimal places) -0.5851
- Now we have added noise to the function and tried to estimate $\omega$ & $\delta$ from the spectrum.
- We follow same procedure as above case and we can observe that $\omega$ & $\delta$ are generated using uniformly distributed random functions, and using the above center of mass statistic we get estimated ones close with error in order of 10% in $\omega$ and 1% in $\delta$
- This error is a lot higher compared to the case without noise as we expected.

### 3.4 Question 4 - Analysis of Chirped Signal Spectrum

- Plot the $DFT$ of the function $\cos(16\ (1.5 + \frac{t}{2\pi})\ t)$ where $-\pi \leq t \leq \pi$ in 1024 steps. This is known as a *chirped* signal.
- Its frequency continuously changes from 16 to 32 radians per second. This also means that the period is 64 samples near $\pi$ and is 32 samples near $+\pi$.

---

```
#section 5 and 6
def chirp(t):
    return cos(16*(1.5+t/(2*pi))*t)
t = linspace(-pi,pi,1025)[:-1]
dt=t[1]-t[0]
fmax=1/dt
N = 1024
y = chirp(t)
Y=fftshift(fft(y))/N
w = linspace(-pi*fmax,pi*fmax,N+1)[:-1]
Xlims = [-100,100]
plot_title = r"Spectrum of Non-Windowed Chirped Signal"
plot_FFT(t,Y,w,Xlims,plot_title)
t = linspace(-pi,pi,1025)[:-1]
dt=t[1]-t[0]
fmax=1/dt
N = 1024
y = chirp(t)

n1=arange(N)
wnd=fftshift(window_fn(n1,N))
y=y*wnd
y[0] = 0
y = fftshift(y)
Y = fftshift(fft(y))/1024.0
w = linspace(-pi*fmax,pi*fmax,1025)[:-1]

Xlims = [-100,100]
```

```
29 plot_title = r"Spectrum of Windowed Chirped Signal"
30 plot_FFT(t,Y,w,Xlims,plot_title)
```

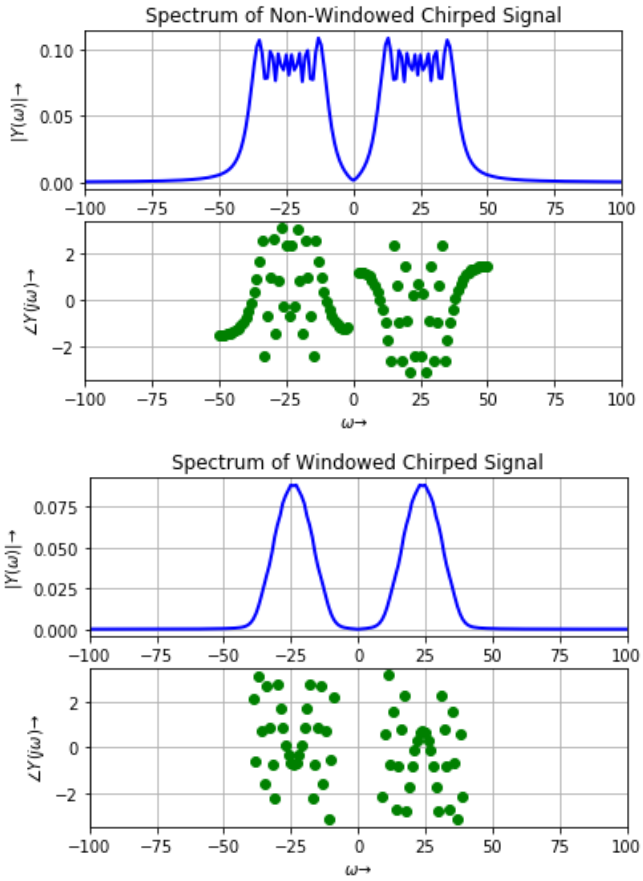


Figure 9: Spectrum of chirped signal with and without windowing

### 3.5 Question 5:

- For the same chirped signal, break the 1024 vector into pieces that are 64 samples wide. Extract the $DFT$ of each and store as a column in a 2D array.
- Then plot the array as a surface plot to show how the frequency of the signal varies with time.
- Plot and analyse the **time frequency** plot, where we get localized $DFTs$ and show how the spectrum evolves in time.
- From the 1024 dimensional vector, we take 64 dimensional sub-vector, and find the fourier transform and we will see how it evolves in time.This is known as **Short time Fourier Transform**

---

```
#the code for the time frequency surface plot(6th)
t_array = split(t,16)  #splitting time into 16
Y_mag = zeros((16,64))
Y_phase = zeros((16,64))

for i in range(len(t_array)):
    n = arange(64)
    wnd = fftshift(0.54+0.46*cos(2*pi*n/64))
    y = cos(16*t_array[i]*(1.5 + t_array[i]/(2*pi)))*wnd
    y[0]=0
    y = fftshift(y)
    Y = fftshift(fft(y))/64.0
    Y_mag[i] = abs(Y)
    Y_phase[i] = angle(Y)

t = t[::64]
w = linspace(-fmax*pi,fmax*pi,64+1); w = w[:-1]
t,w = meshgrid(t,w)

fig1=figure(1)
ax = fig1.add_subplot(111, projection='3d')
surf=ax.plot_surface(w,t,Y_mag.T,cmap='viridis',linewidth=0, antialiased=False
fig1.colorbar(surf, shrink=0.5, aspect=5)
```

```
24 ax.set_title('surface plot');
25 ylabel(r"$\omega\rightarrow$")
26 xlabel(r"$t\rightarrow$")
27 show()
```

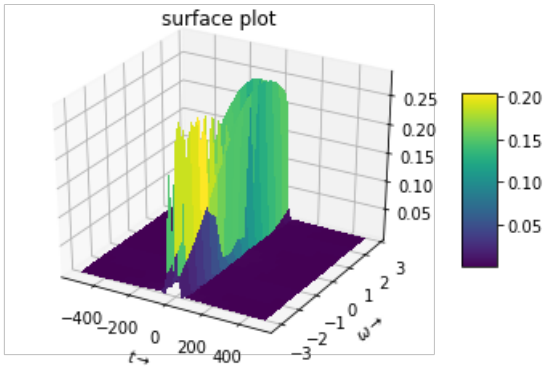


Figure 10: Surface plot of the magnitude of broken chirped signal

### 3.5.1 Results & Discussion:

- We observe that the magnitude of the fourier transform splits as time progresses as the frequency of the signal increases.
- In the surface plot of magnitude of spectrum vs time vs frequency, we observe strong peaks and as inferred from the contour plot of the magnitude spectrum we see 2 lobes whose separation increases as time increases.

# 4 Conclusion :

- Here in this assignment we implemented windowed fourier transform and also understood the need for windowing and also effects of windowing.
- Moreover,we implemented **Short time Fourier Transform** and witnessed how fourier transform evolves in time.