Spectra of non-periodic signals

Course: EE2703 - Applied Programming Lab

Date: 28/04/2018

Abstract

In this week, we dealt with non-periodic signals and obtained their fourier spectra.

1 INTRODUCTION

For periodic signals our approach to find the spectra was,

1) Sample the signal so that $f_{Nyquist}$ is met, and Δf is small enough.

2) Generate the frequency axis from $-$f max /2 to +f max /2, taking care to drop the last term.

3) Ensure that the signal starts at t = 0^+ and ends at t = 0^-.

4) Use 2k samples

5) Obtain the DFT

6) Rotate the samples so that they go from f = $-$ f max /2 to f = + f max /2 $-$ Δ f .

7) Plot the magnitude and phase of the spectrum. Usually we would plot the magnitude in dB and the phase in degrees and the frequency axis would be logarithmic. This is to capture polynomial decay of the spectrum.

2 QUESTIONS

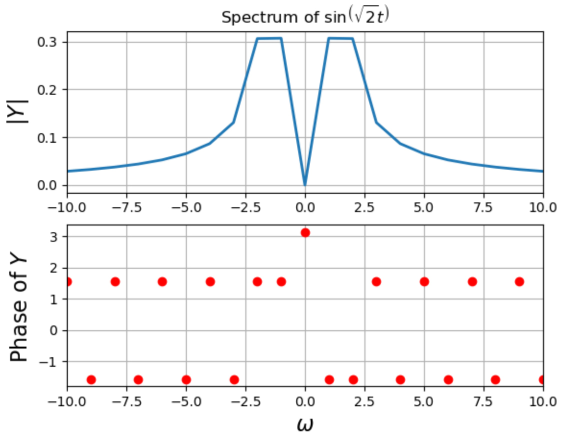
2.1 Working out the examples from the given pdf,

```
#Importing modules and libraries

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

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

2.1.1 Calculating dft in python for $\sin(\sqrt{2}t)$:

Now we want to look at non-periodic signals. Our signal is $\sin(\sqrt{2}t)$. After obtaining its spectrum over 0 to 2π with 64 samples, this function looks as follows:



Below is the code for the above graphs.

```
#Obtaining fourier transform of sin(sqrt(2*t))

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 sample corresponding to -tmax should be set zero
y = fftshift(y)
#make y start with y(t=0)
Y = fftshift(fft(y))/64.0
w = linspace(-pi*fmax,pi*fmax,65)
w = w[:-1]

#Plotting

plt.figure(0)
plt.subplot(2,1,1)
plt.plot(w,abs(Y),lw = 2)
plt.xlim([-10,10])
plt.ylabel(r"$|Y|$",size = 16)
plt.title(r"Spectrum of $\sin\left(\sqrt{2}t\right)$")
plt.grid(True)
plt.subplot(2,1,2)
plt.plot(w,angle(Y),'ro',lw = 2)
plt.xlim([−10,10])
plt.ylabel(r"Phase of $Y$",size = 16)
plt.xlabel(r"$\omega$",size = 16)
plt.grid(True)
plt.savefig("sine_root2t_fourier.png")
plt.show()
```

Inference:

1) We expected two spikes, but what we got were two peaks each with two values and a gradually decaying magnitude.

2) The phase is correct though.

There is one line in the code,

y[0] = 0

This is for the sample corresponding to -tmax should be set zero and an antisymmetric function has a purely imaginary fourier transform.

But what about an antisymmetric set of samples? Suppose,

y[0] = 0 (sin(0))

y[i] = -y[N-i] (i = 1,2,3 ... $\frac{N}{2} - 1$)

y[$\frac{N}{2}$] = sin(t$_{\frac{N}{2}}$) (sin(-t$_{max}$))

The DFT of this sequence gives us,

Y[k] = $\sum_{n=0}^{N-1}$y[n]e$^{\frac{2\pi jkn}{N}}$

Y[k] = $\sum_{n=1}^{\frac{N}{2}-1}$y[n]e$^{\frac{2\pi jkn}{N}}$ - e$^{\frac{*2\pi jkn}{N}}$ + y[$\frac{N}{2}$]e$^{\pi kj}$

= $\sum_{n=1}^{\frac{N}{2}-1}$-2jy[n]sin($\frac{2\pi kn}{N}$)+(-1)^{k}y[$\frac{N}{2}$]

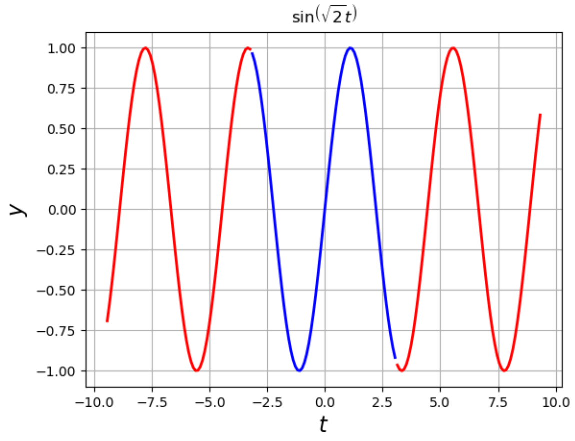
But this is no longer pure imaginary! Since we need that property, we set y[$\frac{N}{2}$] to zero. That gives us a purely imaginary phase.

2.1.2 Plotting for many time periods :

In this question, we plotted the graphs for three different time periods.

```
#Plotting for several time periods

t1 = linspace(−pi,pi,65)
t1 = t1[:−1]
t2 = linspace(−3∗pi,−pi,65)
t2 = t2[:−1]
t3 = linspace(pi,3∗pi,65)
t3 = t3[:−1]
# y=sin(sqrt(2)∗t)

#Plotting

plt.figure(1)
plt.plot(t1,sin(sqrt(2)∗t1),'b',lw = 2)
plt.plot(t2,sin(sqrt(2)∗t2),'r',lw = 2)
plt.plot(t3,sin(sqrt(2)*t3),'r',lw = 2)
plt.ylabel(r"$y$",size = 16)
plt.xlabel(r"$t$",size = 16)
plt.title(r"$\sin\left(\sqrt{2}t\right)$")
plt.grid(True)
plt.savefig("many_time_periods.png")
plt.show()
```

Above is the code we used. Following is the obtained plot,



The blue line connects the points whose DFT we took. Red lines show the continuation of the fucntion.

Inference:

1) Even though $\sin(\sqrt{2}t)$ is a periodic function, the portion between $-\pi$ and π is not the part that can be replicated to generate the function.

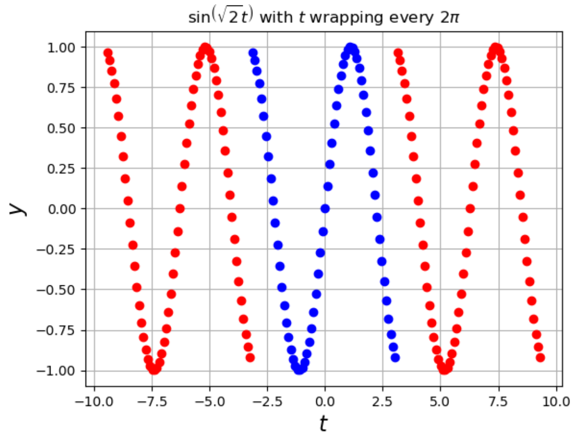
2.1.3 Replicating for the required part :

So which function is the DFT trying to fourier analyse? For that we have to replicate just the above blue points. And that is shown below,

```
#Replicating required part

t1 = linspace(-pi,pi,65)
t1 = t1[:-1]
t2 = linspace(-3*pi,-pi,65)
t2 = t2[:-1]
t3 = linspace(pi,3*pi,65)
t3 = t3[:-1]
y = sin(sqrt(2)*t1)

#Plotting
```

```
plt.figure(3)
plt.plot(t1,y,'bo',lw = 2)
plt.plot(t2,y,'ro',lw = 2)
plt.plot(t3,y,'ro',lw = 2)
plt.ylabel(r"$y$",size = 16)
plt.xlabel(r"$t$",size = 16)
plt.title(r"$\sin\left(\sqrt{2}t\right)
$ with $t$ wrapping every $2\pi$ ")
plt.grid(True)
plt.savefig("replication.png")
plt.show()
```

Below are the obatined graphs,



Inference:

1) Clearly this function is not $\sin(\sqrt{2}t)$ and that is why the DFT is not what we expect.

2.1.4 Discussing Gibbs Phenomenon :

The Fourier transform of the box function,

f(t) = 1 for $|t| <= t_0$

f(t) = 0 for $|t| > t_0$

is,

$F(\varpi) = \frac{2sin(\varpi t_0)}{\varpi_0}$

The spectrum of the box function decays very slowly, as $\frac{2}{\varpi_0}$.

Now our function is an odd function with a big jump.

So let us consider the periodic ramp:

f(t) = t for $-\pi < t < \pi$

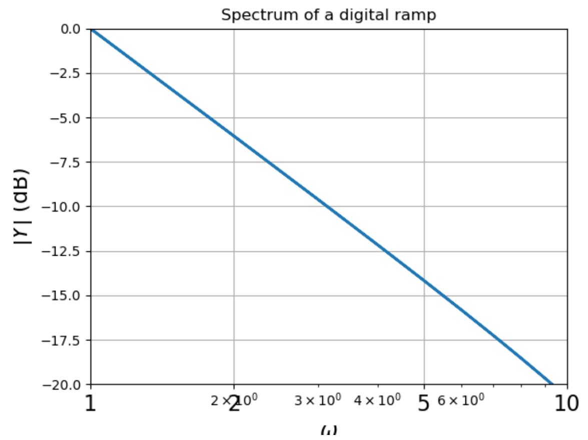
Then the fourier series of this ramp is,

$f(t) = 2(\frac{sint}{1} - \frac{sin2t}{2} + \frac{sin3t}{3} - ...)$

Again the coefficients decay very slowly. The DFT is just like the fourier series, except that both time and frequency are samples. So, if the time samples are like a ramp, the frequency samples will decay as $1/\omega$. Below is the code for it.

```
##Obtaining ramp

t = linspace(-pi,pi,65)
t = t[:-1]
dt = t[1]-t[0]
fmax = 1/dt
y = t
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
w = linspace(-pi*fmax,pi*fmax,65)
w = w[:-1]

#Plotting

plt.figure(3)
plt.semilogx(abs(w),20*log10(abs(Y)),lw = 2)
plt.xlim([1,10])
plt.ylim([-20,0])
plt.xticks([1,2,5,10],["1","2","5","10"],size = 16)
plt.ylabel(r"$|Y|$ (dB)",size = 16)
plt.title(r"Spectrum of a digital ramp")
plt.xlabel(r"$\omega$",size = 16)
plt.grid(True)
plt.savefig("ramp.png")
plt.show()
```

This is the graph I got.



Inference:

1) Clearly the spectrum decays as 20 dB per decade, which corresponds to 1/ω.

2) The big jumps at nπ force this slowly decaying spectrum, which is why we don't see the expected spikes for the spectrum of sin($\sqrt{2}$t).

2.1.5 Discussing windowing :

Well, the spikes happen at the end of the periodic interval. So we damp the function near there, i.e., we multiply our function sequence f[n] by a "window" sequence w[n]:

g(n) = f(n)w(n)

The new spectrum is got by convolving the two fourier transforms:

$G_k = \sum_{n=0}^{N-1} F_n W_{k-n}$

Suppose f_n is a sinusoid. Then F_k has two spikes. But the two spikes are now smeared out by W_k . So we expect to get broader peaks. But what this also does is to suppress the jump at the edge of the window. The window we will use is called the Hamming window:

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

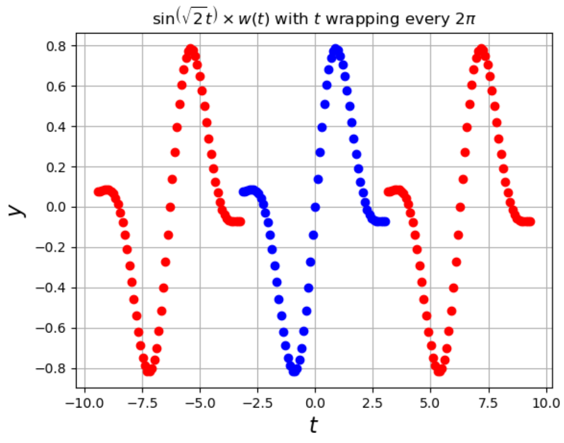
= 0 else

Let us look at our time sequence for sin($\sqrt{2}$t) now . . .

```
#Windowing
t1 = linspace(-pi,pi,65)
t1 = t1[:-1]
t2 = linspace(-3*pi,-pi,65)
t2 = t2[:-1]
t3 = linspace(pi,3*pi,65)
t3 = t3[:-1]
n = arange(64)
wnd = fftshift(0.54+0.46*cos(2*pi*n/63))
y = sin(sqrt(2)*t1)*wnd
```

```
#Plotting

plt.figure(4)
plt.plot(t1,y,'bo',lw = 2)
plt.plot(t2,y,'ro',lw = 2)
plt.plot(t3,y,'ro',lw = 2)
plt.ylabel(r"$y$",size = 16)
plt.xlabel(r"$t$",size = 16)
plt.title(r"$\sin\left(\sqrt{2}t\right)\times w(t)$
with $t$ wrapping every $2\pi$ ")
plt.grid(True)
plt.savefig("windowing.png")
plt.show()
```

Below is the plot I got,



4

Inference:
1) The jump is still there, but it is much reduced.

2.1.6 Obtaining the spectrum of windowed function :

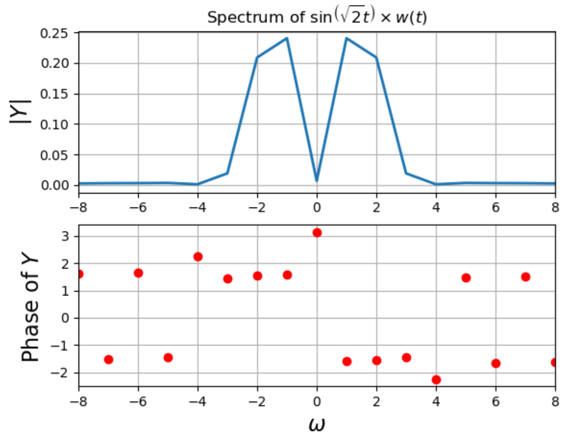
The jump is still there, but it is much reduced. Keeping a little bit of the jump gives us an extra 10 db of suppression. Now let us take the DFT of this sequence and see what we get.

Below is the code for it.

```
#Windowed fourier response

t = linspace(-pi,pi,65)
t = t[:-1]
dt = t[1]-t[0]
fmax = 1/dt
```

```
n = arange(64)
wnd = fftshift(0.54+0.46*cos(2*pi*n/63))
y = sin(sqrt(2)*t)*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))/64.0
w = linspace(-pi*fmax,pi*fmax,65)
w = w[:-1]

#Plotting

plt.figure(5)
plt.subplot(2,1,1)
plt.plot(w,abs(Y),lw = 2)
plt.xlim([-8,8])
plt.ylabel(r"$|Y|$",size = 16)
plt.title(r"Spectrum of $\sin\left(\sqrt{2}t\right)\times w(t)$")
plt.grid(True)
plt.subplot(2,1,2)
plt.plot(w,angle(Y),'ro',lw = 2)
plt.xlim([-8,8])
plt.ylabel(r"Phase of $Y$",size = 16)
plt.xlabel(r"$\omega$",size = 16)
plt.grid(True)
plt.savefig("windowed_fourier.png")
plt.show()
```

Below is the obtained plot.



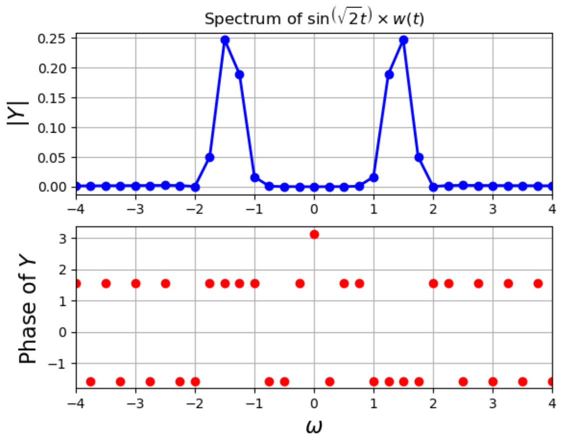
Inference:

1) Compare to our first plot and we can see that the magnitude is greatly improved.

2.1.7 Obtaining the spectrum with increased number of points :

We still have a peak that is two samples wide. But that is because $\sqrt{2}$ lies between 1 and 2, which are the two fourier components available. If we use four times the number of points we should get better results. Below is the code for above implementation.

```
#Increased points plotting

t = linspace(-4*pi,4*pi,257)
t = t[:-1]
dt = t[1]-t[0]
fmax = 1/dt
n = arange(256)
wnd = fftshift(0.54+0.46*cos(2*pi*n/256))
y = sin(sqrt(2)*t)
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*fmax,pi*fmax,257)
w = w[:-1]

#Plotting

plt.figure(6)
plt.subplot(2,1,1)
plt.plot(w,abs(Y),'b',w,abs(Y),'bo',lw = 2)
plt.xlim([-4,4])
plt.ylabel(r"$|Y|$",size = 16)
plt.title(r"Spectrum of $\sin\left(\sqrt{2}t\right)\times w(t)$")
plt.grid(True)
plt.subplot(2,1,2)
plt.plot(w,angle(Y),'ro',lw = 2)
plt.xlim([-4,4])
plt.ylabel(r"Phase of $Y$",size = 16)
plt.xlabel(r"$\omega$",size = 16)
plt.grid(True)
plt.savefig("many_points_windowed_fourier.png")
plt.show()
```



Inference:

1) It is quite bit better since we are now zoomed in and see a lot more detail. But it is not just a single peak. The reason for that is w(t). Multiplication in time is convolution in frequency and vice versa. So by multiplying with w(t), we got rid of the 1/ f decay. But the delta function is now replaced by the shape of the DFT of w[n]. That gives us a factor of two broadening over the peak when there is no window, which is why we still see a peak whose width is two samples. It is not because $\sqrt{2}$ is between 1.25 and 1.5. There is an alternate function in the above code, namely sin(1.25t). But this gives a broad peak as well. That is because of w[n].$\cos^3(\varpi_0 t)$

2.2 Obtaining the spectrum of $\cos^3(\varpi_0 t)$,

In this question, we have obtained the fourier transform of $\cos^3(\varpi_0 t)$,

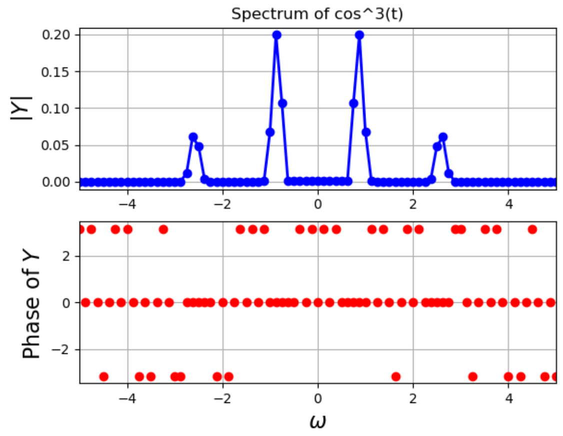
2.2.1 With Hamming window :

Below is the code for it,

```
#Spectrum of cos cubed t with Hamming window

t = linspace(-8*pi,8*pi,513)
t = t[:-1]
dt = t[1]-t[0]
fmax = 1/dt
n = arange(512)
wnd = fftshift(0.54+0.46*cos(2*pi*n/512))
y = cos(0.86*t)*cos(0.86*t)*cos(0.86*t)
y = y*wnd
y[0] = 0
y = fftshift(y)
Y = fftshift(fft(y))/512.0
```

```
w = linspace(-pi*fmax, pi*fmax, 513)
w = w[:-1]

#Plotting

 plt.figure(7)
 plt.subplot(2,1,1)
 plt.plot(w, abs(Y), 'b', w, abs(Y), 'bo', lw = 2)
 plt.xlim([-5,5])
 plt.ylabel(r"$|Y|$", 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)
 plt.xlim([-5,5])
 plt.ylabel(r"Phase of $Y$", size = 16)
 plt.xlabel(r"$\omega$", size = 16)
 plt.grid(True)
 plt.savefig("cos_cubed_ham.png")
 plt.show()
```

Below is the graph I got,



Inference:

1) We see 4 peaks as expected with alternate positive and negative phase for sine and zero phase for cosine.

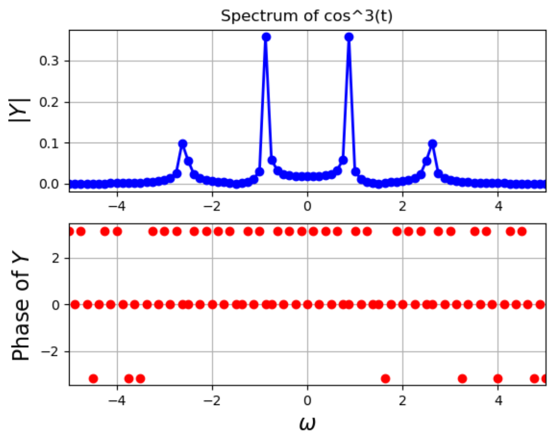
2) The phase at the peak is π.

2.2.2 Without Hamming window :

Below is the required code,

```
#Spectrum of cos cubed t without Hamming window

t = linspace(-8*pi,8*pi,513)
t = t[:-1]
dt = t[1]-t[0]
fmax = 1/dt
y = cos(0.86*t)*cos(0.86*t)*cos(0.86*t)
y[0] = 0
y = fftshift(y)
Y = fftshift(fft(y))/512.0
w = linspace(-pi*fmax,pi*fmax,513)
w = w[:-1]

#Plotting

plt.figure(8)
plt.subplot(2,1,1)
plt.plot(w,abs(Y),'b',w,abs(Y),'bo',lw = 2)
plt.xlim([-5,5])
plt.ylabel(r"$|Y|$",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)
plt.xlim([-5,5])
plt.ylabel(r"Phase of $Y$",size = 16)
plt.xlabel(r"$\omega$",size = 16)
plt.grid(True)
plt.savefig("cos_cubed_non-ham.png")
plt.show()
```

Below is the obtained plot.



Inference:

1) We see 4 peaks expected. But, as compared to the earlier case, the height of the peaks is quite larger.

2.3 Obtaining the spectrum of cos(ω_0t + δ),

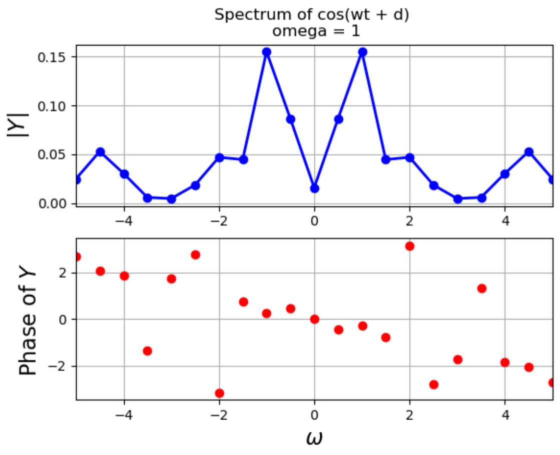
In this question, we had to write a program that will take a 128 element vector known to contain cos(ω_0t + δ) for arbitrary δ and $0.5 < \omega_0 < 1.5$. The values of t go from $-\pi$ to π. We had to extract the digital spectrum of the signal, find the two peaks at $\pm\omega_0$, and estimate ω_0 and δ.

Below is the code for it.

```
#Spectrum of cos(omega*t + delta)

t = linspace(-2*pi,2*pi,129)
t = t[:-1]
dt = t[1]-t[0]
fmax = 1/dt
delta = numpy.random.randn(len(t))
y = cos(t + delta)
#I took omega as 1
n = arange(128)
wnd = fftshift(0.54+0.46*cos(2*pi*n/128))
y = y*wnd
y[0] = 0
y = fftshift(y)
Y = fftshift(fft(y))/128.0
w = linspace(-pi*fmax,pi*fmax,129)
w = w[:-1]

#Plotting
```

```
plt.figure(9)
plt.subplot(2,1,1)
plt.plot(w,abs(Y),'b',w,abs(Y),'bo',lw = 2)
plt.xlim([-5,5])
plt.ylabel(r"$|Y|$",size = 16)
plt.title('Spectrum of cos(wt + d) \n omega = 1')
plt.grid(True)
plt.subplot(2,1,2)
plt.plot(w,angle(Y),'ro',lw = 2)
plt.xlim([-5,5])
plt.ylabel(r"Phase of $Y$",size = 16)
plt.xlabel(r"$\omega$",size = 16)
plt.grid(True)
plt.savefig("cos(wt + delta).png")
plt.show()
```

I obtained the following graph.



`Inference:`

Here, the location of the peaks is at ϖ_0= -1 and ϖ_0= 1. From the phase plot, we can get an estimate of the value of δ. Phase is shifted by δ at these peaks. As the value of δ increases, the shift in phase increases more and more.

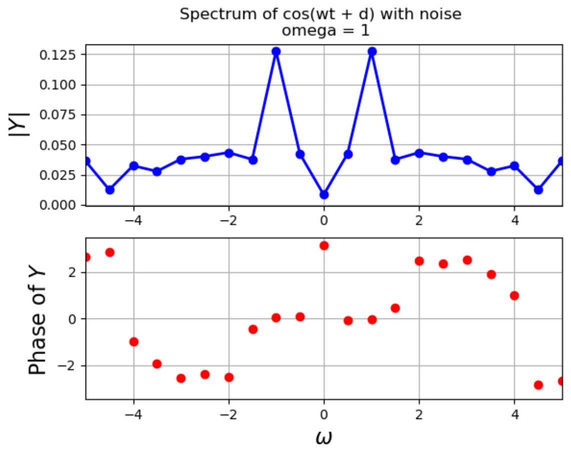
2.4 Obtaining the spectrum of cos(ω_0t + δ) with noise,

Now, we add "white gaussian noise" to the problem in question 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). Again we need to find $\varpi_0$and δ.

This is the code for it.

```
#Spectrum of cos(omega*t + delta) with white gaussian noise

t = linspace(-2*pi,2*pi,129)
t = t[:-1]
dt = t[1]-t[0]
fmax = 1/dt
delta = numpy.random.randn(len(t))
y = cos(t + delta) + 0.1*numpy.random.randn(len(t))
#I took omega as 1
n = arange(128)
wnd = fftshift(0.54+0.46*cos(2*pi*n/128))
y = y*wnd
y[0] = 0
y = fftshift(y)
Y = fftshift(fft(y))/128.0
w = linspace(-pi*fmax,pi*fmax,129)
w = w[:-1]

#Plotting

plt.figure(10)
plt.subplot(2,1,1)
plt.plot(w,abs(Y),'b',w,abs(Y),'bo',lw = 2)
plt.xlim([-5,5])
plt.ylabel(r"$|Y|$",size = 16)
plt.title('Spectrum of cos(wt + d) with noise \n omega = 1')
plt.grid(True)
plt.subplot(2,1,2)
plt.plot(w,angle(Y),'ro',lw = 2)
plt.xlim([-5,5])
plt.ylabel(r"Phase of $Y$",size = 16)
plt.xlabel(r"$\omega$",size = 16)
plt.grid(True)
plt.savefig("cos(wt + delta) with noise.png")
plt.show()
```

This is the obtained plot.



```
Inference:
```

Here, the location of the peaks is at ϖ_0= -1 and ϖ_0= 1. From the phase plot, we can get an estimate of the value of δ. As we can see, the magnitude plot is almost the same except for some noisy values instead of zeros, however the phase plot is quite dierent at the points other than the peaks.

2.5 Plotting the dft of chirped signal :

In this question, we have to plot the dft of 'chirped signal'. The function is,

$\cos(16(1.5 + \frac{t}{2\pi})t)$

for t going from $-\pi$ to π in 1024 steps.

This is known as a "chirped" signal, and its frequency continuously changes from 16 to 32 radians per second.

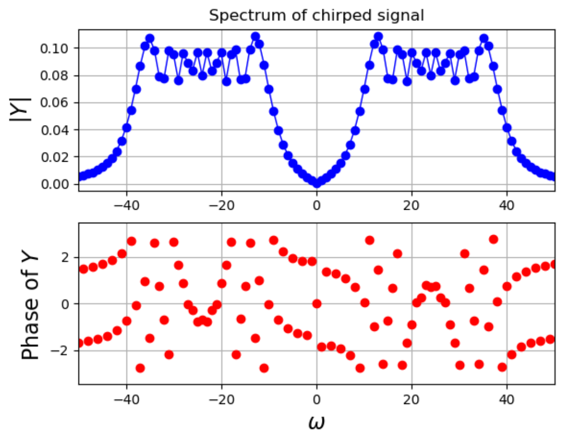
This also means that the period is 64 samples near $-\pi$ and is 32 samples near $+\pi$.

The code for the above implementation is as shown below.

```
#Spectrum of chirped signal

t = linspace(-pi, pi, 1025)
t = t[:-1]
dt = t[1]-t[0]
fmax = 1/dt
y = cos(16*(1.5 + t/(2*pi))*t)
y[0] = 0
y = fftshift(y)
Y = fftshift(fft(y))/1024.0
w = linspace(-pi*fmax, pi*fmax, 1025)
w = w[:-1]
```

```
#Plotting

plt.figure(11)
plt.subplot(2,1,1)
plt.plot(w,abs(Y),'b',w,abs(Y),'bo',lw = 1)
plt.xlim([-50,50])
plt.ylabel(r"$|Y|$",size = 16)
plt.title(r"Spectrum of chirped signal")
plt.grid(True)
plt.subplot(2,1,2)
plt.plot(w,angle(Y),'ro',lw = 2)
plt.xlim([-50,50])
plt.ylabel(r"Phase of $Y$",size = 16)
plt.xlabel(r"$\omega$",size = 16)
plt.grid(True)
plt.savefig("chirped signal.png")
plt.show()
```

Below shown is the plot for the above implementation.



`Inference:`

1) Here, the location of the peaks is at ϖ_0= -25 and ϖ_0= 25. The height of the peak is about 0.20.

2.6 Plotting some sections of the chirped signal and getting the surface plot :

In this question we were asked to the following:

1) For the same chirped signal, break the 1024 vector into pieces that are 64 samples wide.

2) Extract the DFT of each and store as a column in a 2D array.

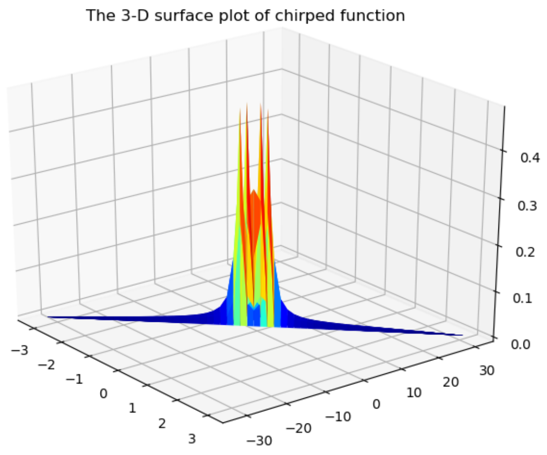
3) Then plot the array as a surface plot to show how the frequency of the signal varies with time. This is a "time-frequency" plot, where we get localized DFTs and show how the spectrum evolves in time.

Below code shows the way, I have splitted the time scale.

```
#Getting dft for a range of frequencies and
obtaining the surface plot

S = numpy.zeros((63, 63))
t_plot = linspace(-pi,pi,64)
t_plot = t_plot[:-1]
dt_plot = t_plot[1]-t_plot[0]
fmax_plot = 1/dt_plot
w_plot = linspace(-pi*fmax_plot,pi*fmax_plot,64)
w_plot = w_plot[:-1]

for k in range(0,15):
        t = linspace((-(8-k)*pi)/8,(-(7-k)*pi)/8, 64)
        t = t[:-1]
        y = cos(16*(1.5 + t/(2*pi))*t)
        y[0] = 0
        y = fftshift(y)
        Y = fftshift(fft(y))/64
        S[:,k] = Y.transpose()
```

Following code shows the way I plotted the surface plot.

```
#Plotting the surface plot

figure = figure(12)
plotting = axes.Axes3D(figure)
#Axes3D is the means to do a surface plot
plt.title('The 3-D surface plot of chirped function')
surf = plotting.plot_surface(t_plot,
w_plot, abs(S).T, rstride=1, cstride=1, cmap=cm.jet)
plt.show()
```



Above shown is the surface plot I obatined.

Inference:

1) From the plot, we see that there are two peaks in the plot.

2) The waveforms are decaying in both the directions.