EE2703 : Applied Programming Lab Assignment 10

Potta Muni Asheesh
EE19B048

May 24, 2021

1 Introduction

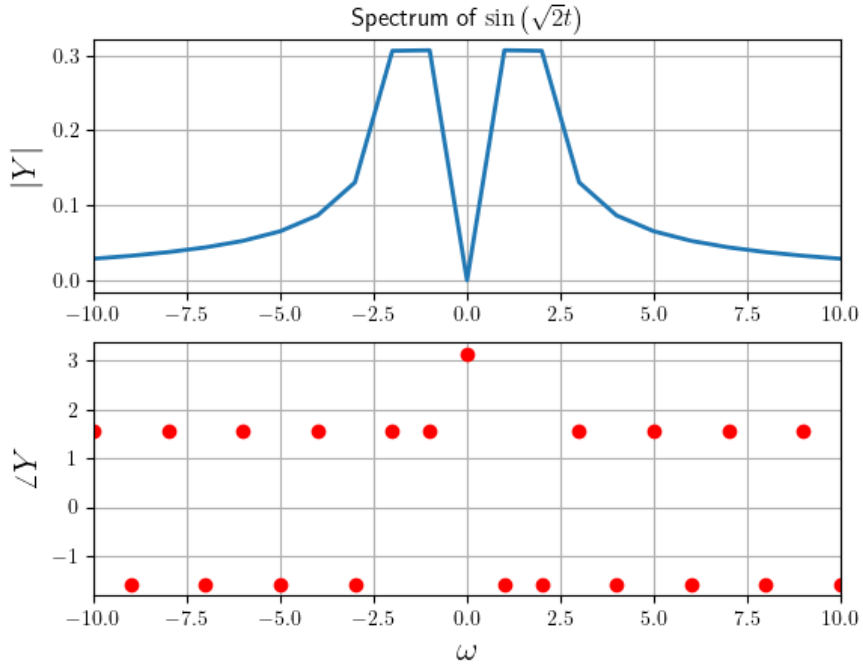
In this assignment, the computation of spectra of non-periodic signals is explored.

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

Consider the signal $y(t) = sin(\sqrt{2}t)$ sampled over the interval $[-\pi, \pi)$ at sampling rate $F_s = 64/2\pi$ and $y(-t_{max})$ is set to 0, to maintain the purely imaginary nature of spectra of sine waves.

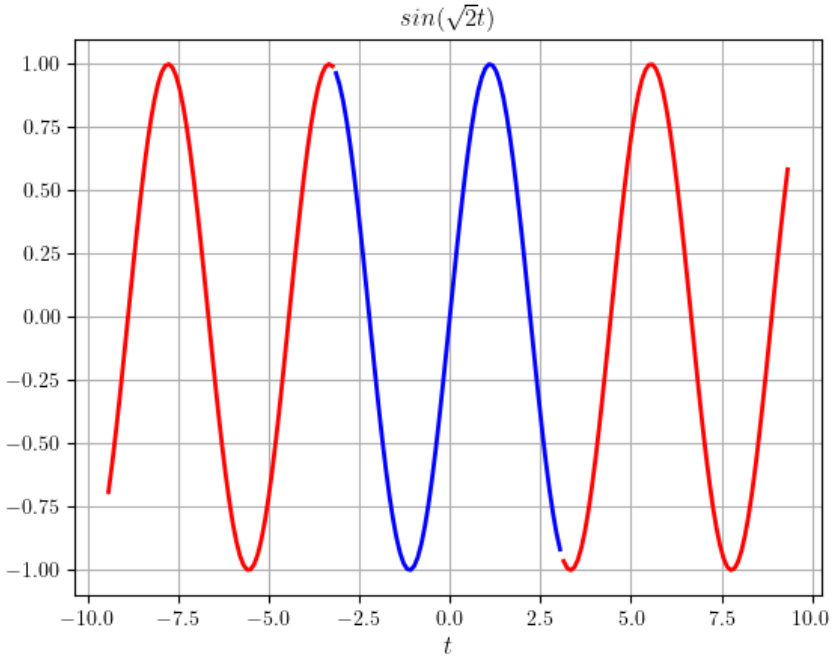
```
def ex1():
    t=np.linspace(-np.pi,np.pi,65);t=t[:-1]
    dt=t[1]-t[0];fmax=1/dt
    y=np.sin(np.sqrt(2)*t)
    y[0]=0 # the sample corresponding to -tmax should be set zero
    y=fftshift(y) # make y start with y(t=0)
    Y=fftshift(fft(y))/64.0
    w=np.linspace(-np.pi*fmax,np.pi*fmax,65);w=w[:-1]
```

The spectrum obtained is

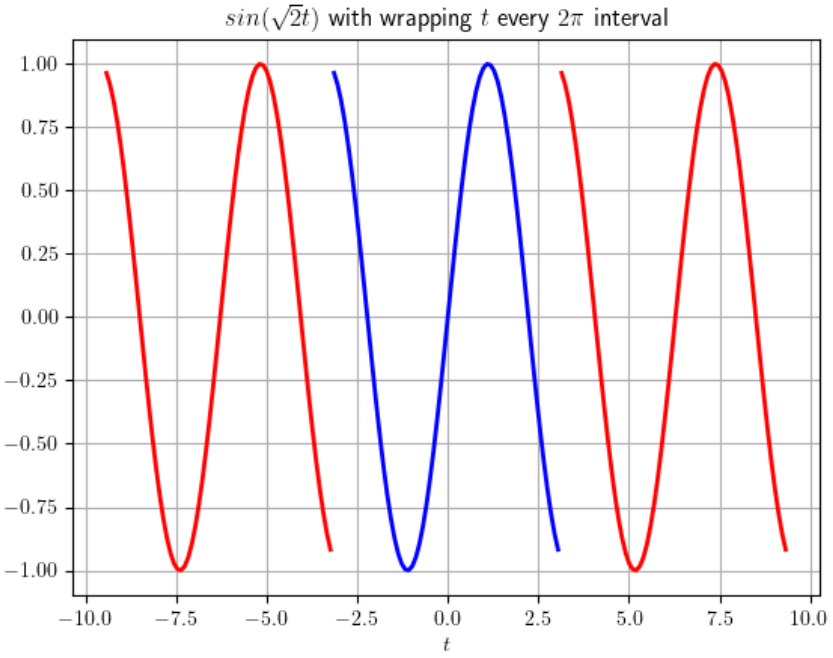


Two spikes were expected. There are two spikes, but they are broad and the magnitude decays after the spikes. The phase corresponding to the spikes is correctly obtained.

To understand what went wrong. Consider the signal over the interval $[-3\pi, 3\pi)$.



It can be observed that the signal is periodic, but the part between $-\pi$ and π is not the repeating part. Consider the signal obtained by wrapping $sin(\sqrt{2}t)$ every 2π, which is the signal for which the spectrum has been obtained above.

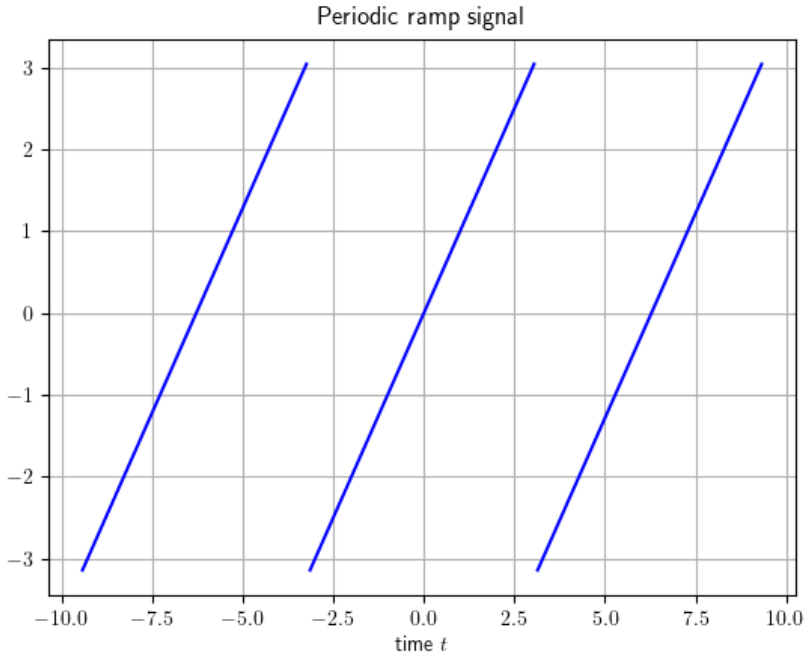


It can be observed that there are jump discontinuities at every period and due to these jumps, the magnitude spectrum sees a gradual decay after the spikes. This is called **Gibbs phenonmenon**.

2.1 Gibbs phenomenon

It is observed in the spectra of signal which have discontinuities. Consider the periodic ramp signal for example.

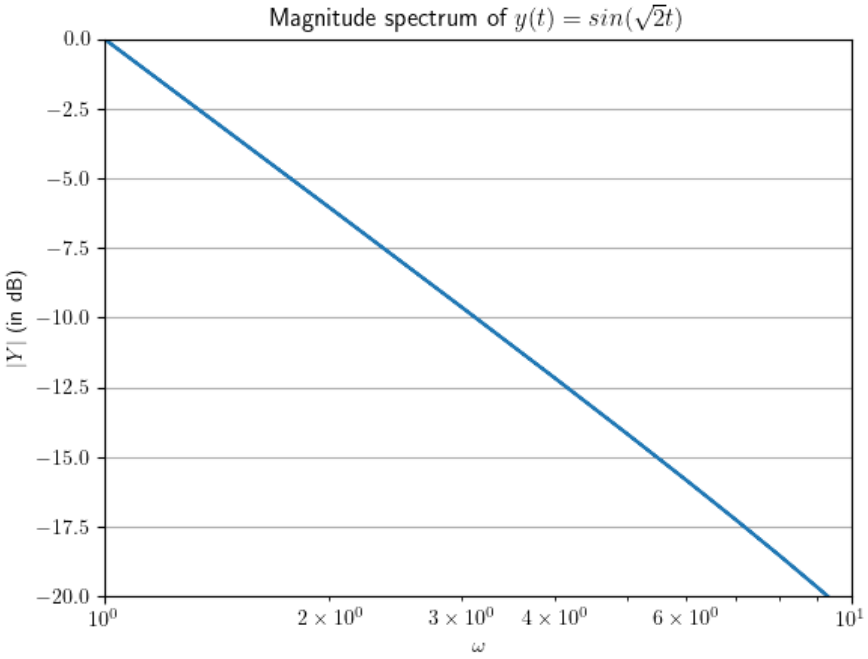
$$y(t) = t, -\pi < t < \pi$$



The fourier series of this signal is

$$y(t) = 2\left(\frac{sin(t)}{1} - \frac{sin(2t)}{2} + \frac{sin(3t)}{3} - ...\right)$$

So, the spectrum decays as $1/\omega$ and the same is obtained, i.e; a -20dB/decade decay.



```
def ex4():
    t = np.linspace(-np.pi, np.pi, 65); t = t[:-1]
    t2 = np.linspace(-3*np.pi, -np.pi, 65); t2 = t2[:-1]
    t3 = np.linspace(np.pi, 3*np.pi, 65); t3 = t3[:-1]
    dt = t[1]-t[0]
    fmax = 1/dt
    y = t
    y[0] = 0
    y = fftshift(y)
    y_spec = fftshift(fft(y))/64.0
    w = np.linspace(-np.pi*fmax, np.pi*fmax, 65); w = w[:-1]
```

So, in order to reduce the affect of this phenomenon, **windowing** is used.

2.2 Windowing

In order to suppress the jump, the signal is multiplied by another sequence called *window*.

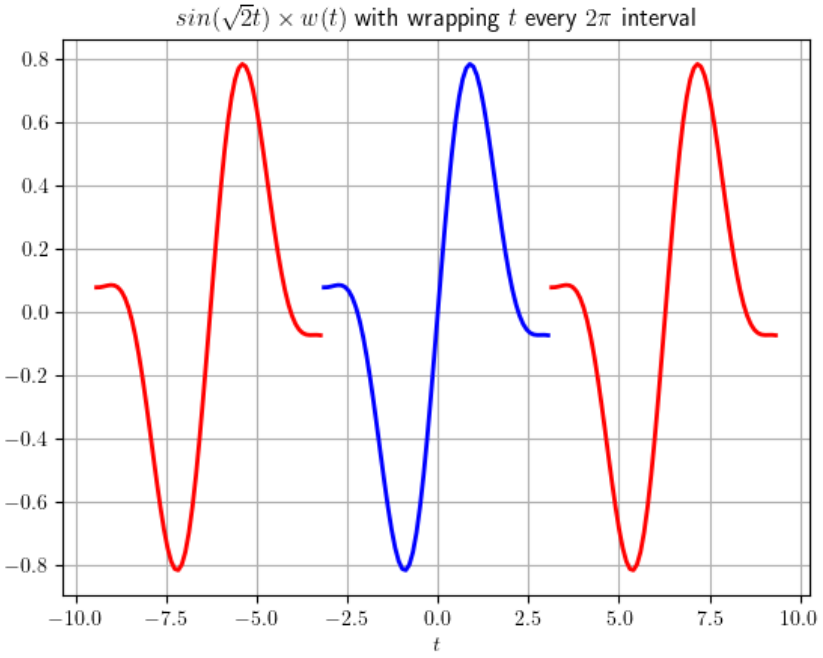
$$g(n) = f(n)w(n)$$

Multiplication in time domain results in convolution in frequency domain. The spectrum of the signal obtained is different from the actual spectrum required.

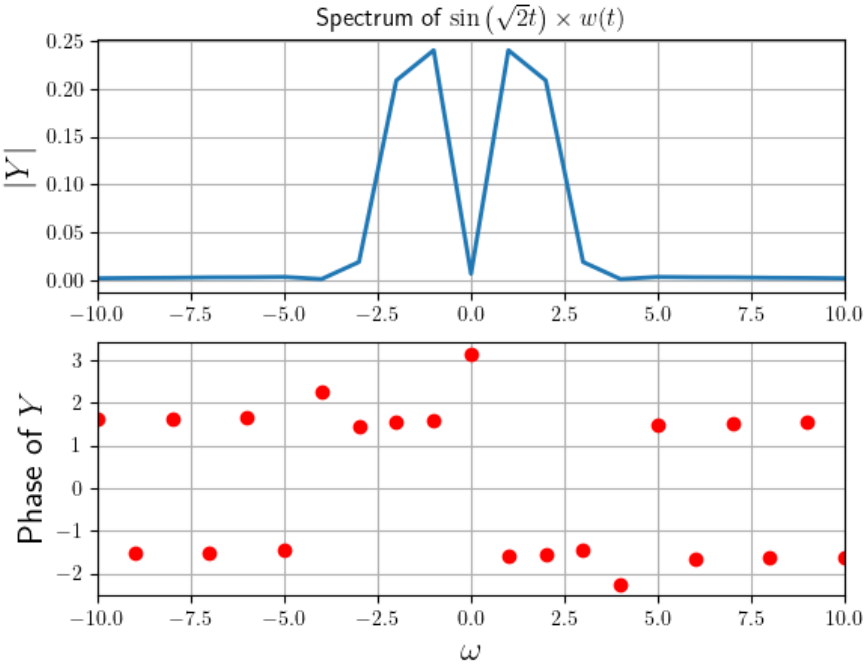
One of the window sequence is the *hamming window*. It is given as

```
def hamming_window(N):
    n = np.arange(N)
    return fftshift(0.54 + 0.46*np.cos(2*np.pi*n/(N-1)))
```

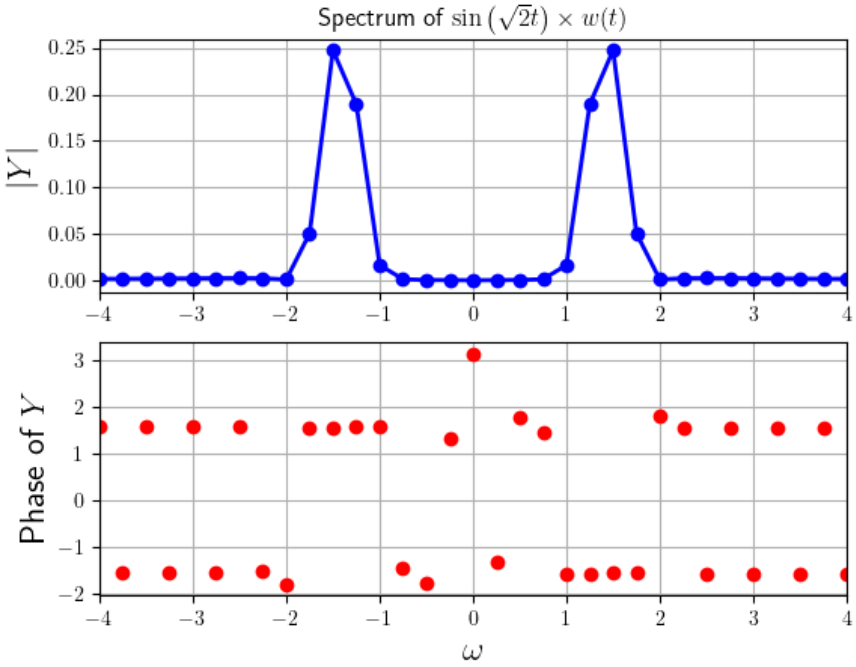
The plot of $sin(\sqrt{2}t) \times w(t)$ wrapped every 2π time is shown below.



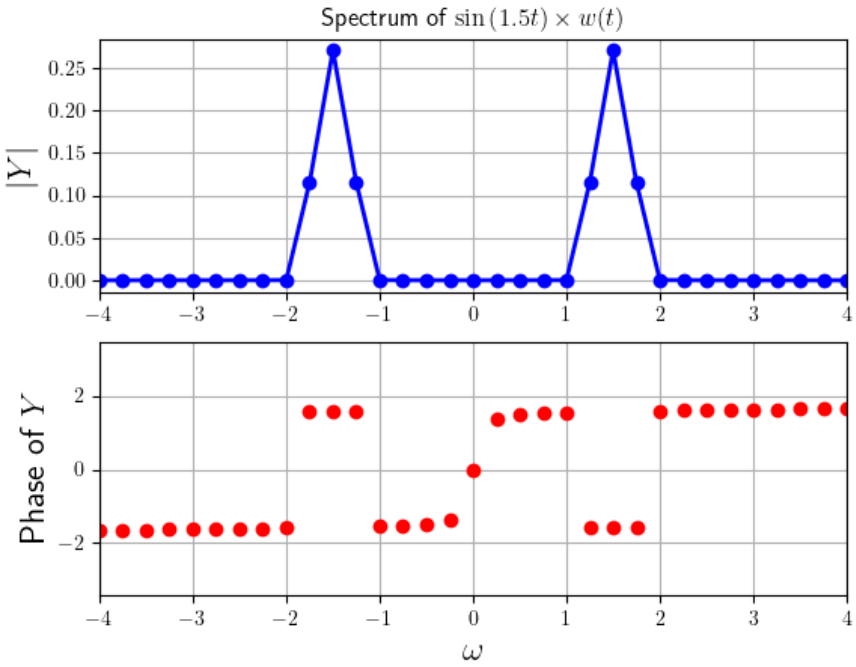
The spectrum of this sequence is given as



The gradual decay in magnitude is not observed, but peak is still broad.
Now, four times the number of points are considered to get better resolution.



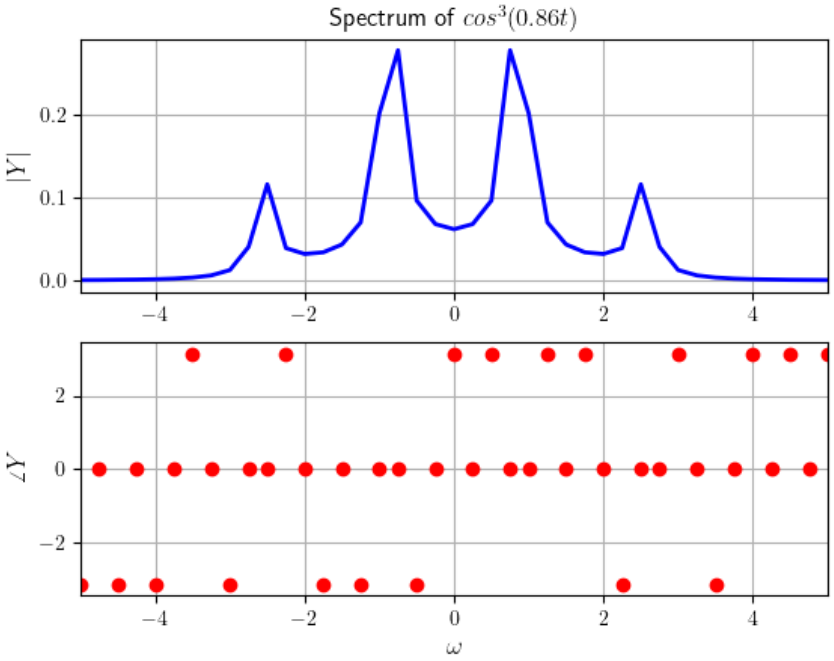
The resolution has certainly improved, but it should be noted that the peak is still broad and this is because of the windowing, not only because $\sqrt{2}$ is between 1 and 2. To confirm this, the spectrum of $sin(1.5t) \times w(t)$ is also broad.



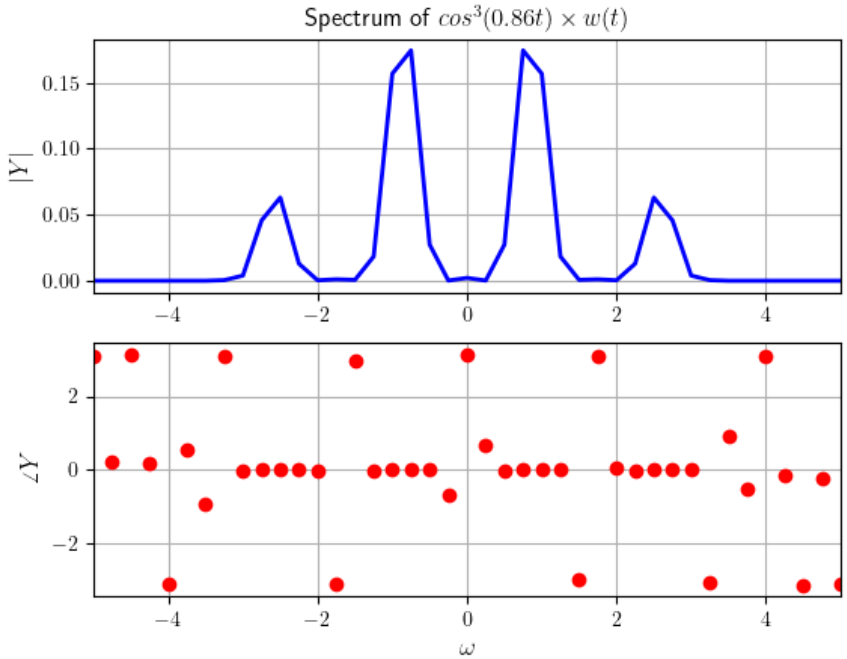
3 Spectrum of $cos^3(\omega_0 t)$

Consider the signal $cos^3(\omega_0 t)$ with $\omega_0 = 0.86$.

The spectrum of this signal without a hamming window is given as



Now, the spectrum of this signal with a hamming window is given as



It can be observed that the windowing has certainly improved the magnitude spectrum.

4 Estimating ω_0 and δ in $cos(\omega_0 t + \delta)$

A signal $cos(\omega_0 t + \delta)$ with arbitrary δ and $0.5 < \omega_0 < 1.5$ of length 128 samples can be generated as

```
def arbit_cos(noise=False):
    t = np.linspace(-np.pi, np.pi, 129); t = t[:-1]
    w0 = np.random.rand() + 0.5
    delta = np.random.rand()*2*np.pi
    # if delta-np.pi > 0:
    #     delta = 2*np.pi - delta
    if noise:
        return t, np.cos(w0*t + delta) + 0.1*np.random.randn(128), w0,
   ↪ delta
    return t, np.cos(w0*t + delta), w0, delta
```

The frequency corresponding to peak in the spectrum is estimate for ω_0 and the phase corresponding to this frequency is the estimate for δ.

```
def q3():
    t, y, w0, delta = arbit_cos()
    dt = t[1] - t[0]
    y[0] = 0
    y = fftshift(y)
    y_spec = fftshift(fft(y))/128
    y_spec_mag = np.abs(y_spec)
    y_spec_ang = np.angle(y_spec)
    w = np.linspace(-np.pi/dt, np.pi/dt, 129); w = w[:-1]
    ii = np.where(y_spec_mag == np.max(y_spec_mag))
    est_w0, est_delta = w[ii][-1], y_spec_ang[ii][-1]
    print('actual w0 and estimated w0 are {}, {}'.format(w0, est_w0))
    print('actual delta and estimated delta are {}, {}'.format(delta,
   ↪ est_delta))
```

Some of the estimates found are shown below.

- Actual ω_0 and estimated ω_0 are 0.5139319849138658, 1.0. Actual δ and estimated δ are 5.2899223447339185, -1.248008220373927.
- Actual ω_0 and estimated ω_0 are 1.0692624800488058, 1.0. Actual δ and estimated δ are 0.397228803727542, 0.37904375462634254.
- Actual ω_0 and estimated ω_0 are 0.5724358016556333, 0.0. Actual δ and estimated δ are 0.14840584305304158, 0.0

It can be observed that frequency estimate is not very accurate due to resolution of the spectrum, but the phase is better estimated.

Silimar method is used to estimate, when white gaussian noise is added to signal.

```
def q4():
    t, y, w0, delta = arbit_cos(noise=True)
    dt = t[1] - t[0]
    y[0] = 0
    y = fftshift(y)
    y_spec = fftshift(fft(y))/128
    y_spec_mag = np.abs(y_spec)
    y_spec_ang = np.angle(y_spec)
    w = np.linspace(-np.pi/dt, np.pi/dt, 129); w = w[:-1]
    ii = np.where(y_spec_mag == np.max(y_spec_mag))
    est_w0, est_delta = np.abs(w[ii][-1]), y_spec_ang[ii][-1]
    if est_delta < 0:
        est_delta += 2*np.pi
    print('Actual $\omega_0$ and estimated $\omega_0$ are {}, {}'.format(
   ↪ w0, est_w0))
    print('Actual $\delta$ and estimated $\delta$ are {}, {}'.format(
   ↪ delta, est_delta))
```

Some of the estimates found are shown below.

- Actual ω_0 and estimated ω_0 are 1.10588175592206, 1.0. Actual δ and estimated δ are 1.0784816041319503, 1.0277791200791555
- Actual ω_0 and estimated ω_0 are 1.3940422549339684, 1.0. Actual δ and estimated δ are 4.880580976127553, 1.3076057592874013
- Actual ω_0 and estimated ω_0 are 1.152667662320201, 1.0. Actual δ and estimated δ are 2.446344906917355, 2.500639994772733

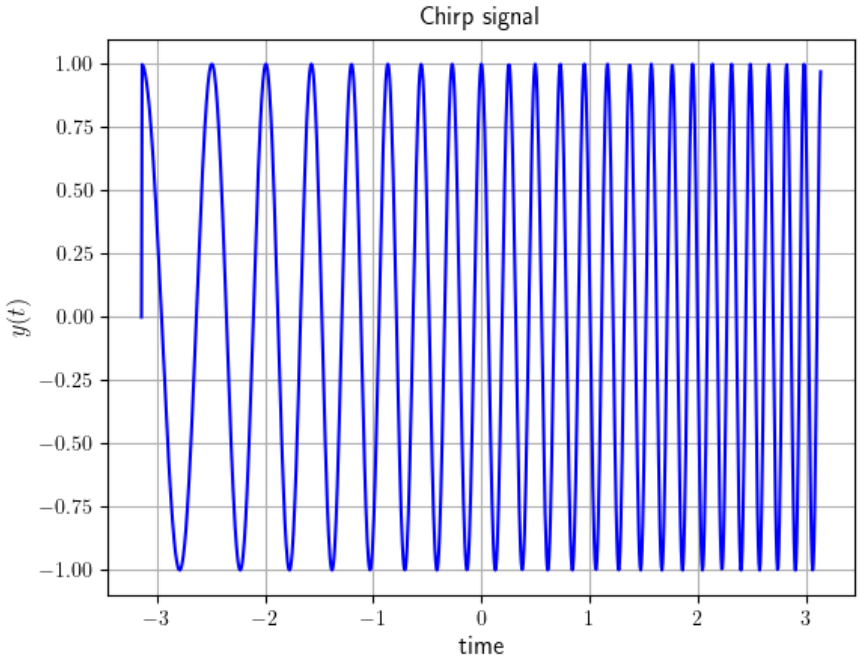
It can be seen that the estimate are not as accurate as they were when no noise is added. Hencen the noise makes estimation harder.

5 Spectrum of Chirp signal

Consider the signal

$$y(t) = cos\left(16\left(1.5 + \frac{t}{2\pi}\right)t\right)$$

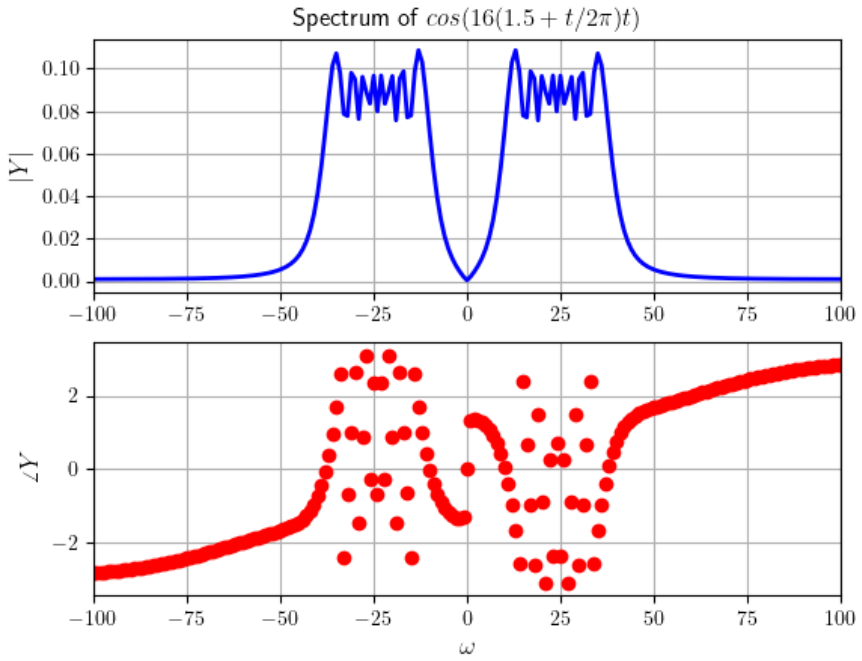
This signal is called *Chirped signal*.



It can observed from the plot and equation that angular frequency gradually increases from 16 rads/s to 32 rads/s as time goes from $-\pi$ to π

The spectrum of this signal is obtained as,

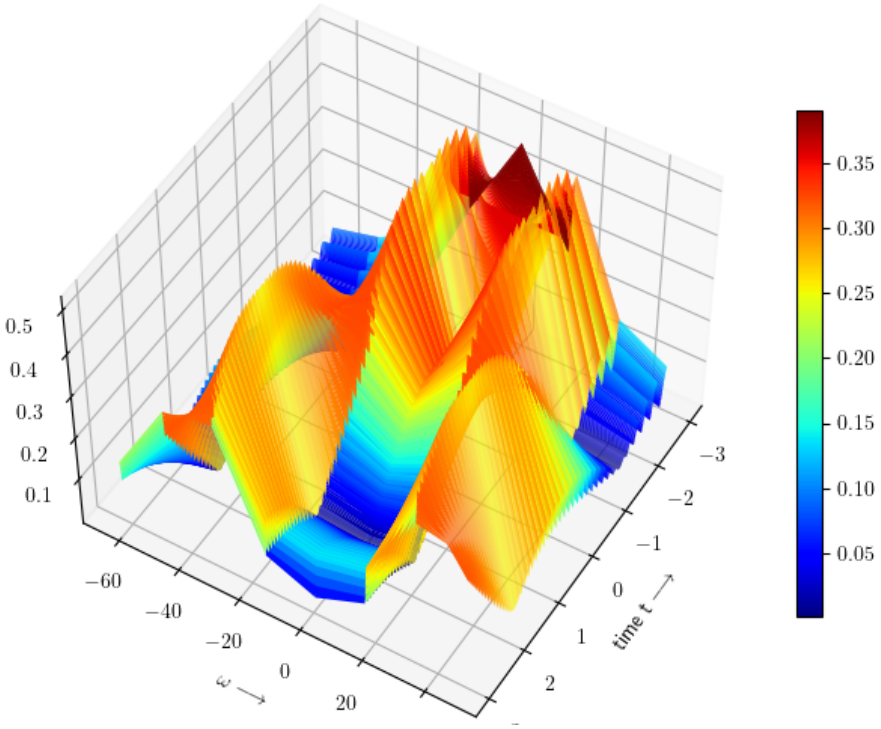
```
def q5():
    t = np.linspace(-np.pi, np.pi, 1025); t = t[:-1]
    dt = t[1] - t[0]
    y = np.cos(16*(1.5 + t/(2*np.pi))*t)
    y[0] = 0
    # y = fftshift(y)
    y_spec = fftshift(fft(y))/1024
    w = np.linspace(-np.pi/dt, np.pi/dt, 1025); w = w[:-1]
```



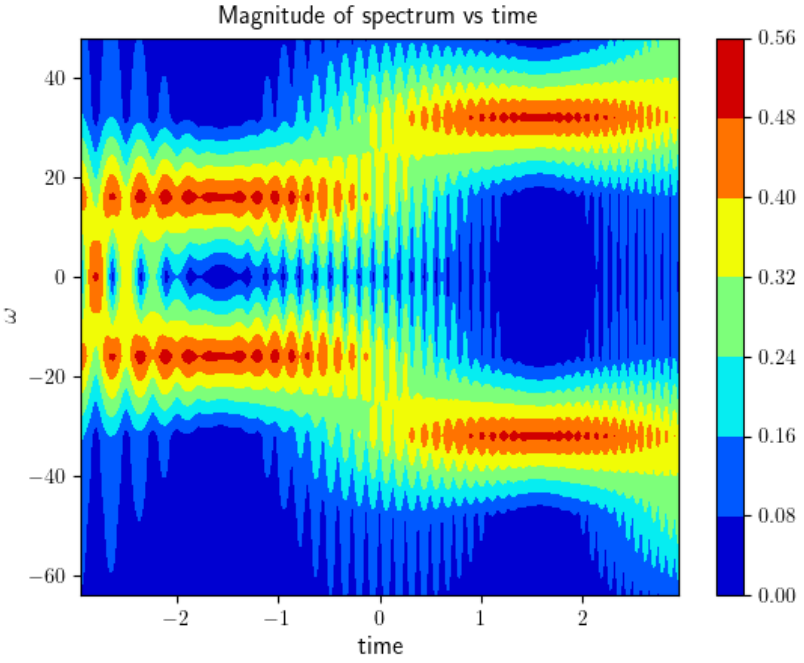
In the spectrum, a band is seen between 16 and 32 rads/s frequencies, but the gradual change of the frequency is not evident from the spectrum. So, a frequency - time plot is made to observe is change.

```
def q6():
    t = np.linspace(-np.pi, np.pi, 1025); t = t[:-1]
    dt = t[1] - t[0]
    y = np.cos(16*(1.5 + t/(2*np.pi))*t)
    y[0] = 0
    # y = fftshift(y)
    piece_length = 64
    freqvstime = np.zeros((piece_length, 1024-piece_length), dtype=
    ↪ complex)
    for i in range(1024-piece_length):
        piece_spec = fftshift(fft(y[i:i+piece_length]))/piece_length
        freqvstime[:,i] = piece_spec
    w = np.linspace(-np.pi/dt, np.pi/dt, piece_length+1); w = w[:-1]; w =
    ↪  w[28:-28]
    T, W = np.meshgrid(t[piece_length//2:-piece_length//2], w)
```

The surface plot of the frequency - time plot is shown below.



Instead of a surface plot, a filled contour plot gives better understanding what's happening here.



It can be seen that near $t = -\pi$, the peak frequency is around 16 rads/s, but this becomes 32 rads/s as t becomes positive and goes towards π