Assignment No. 9

Srivenkat A (EE18B038)

May 6, 2020

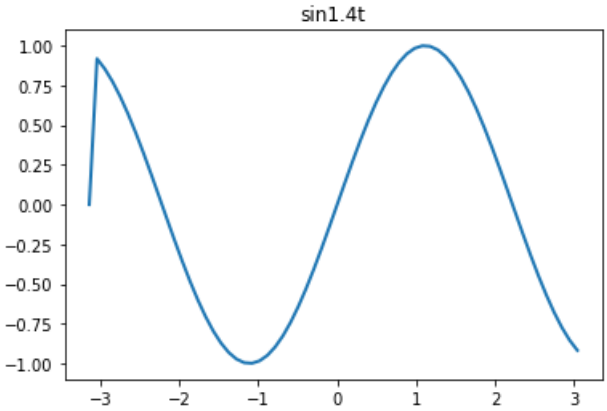
Introduction

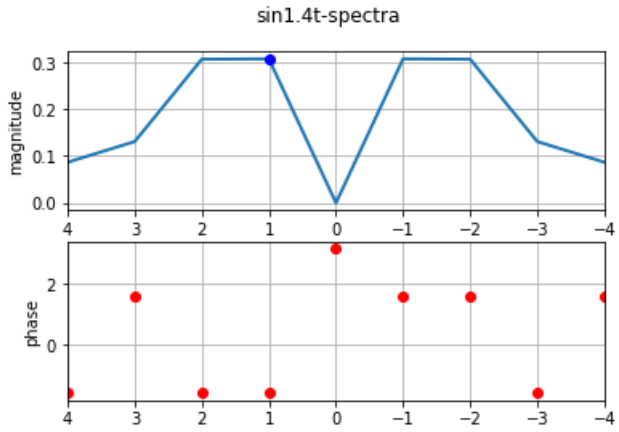
The assignment focuses on using numpy.fft() library in python and windowing techniques to calculate the Discrete fourier transform of non-periodic signals. The areas of focus are

- Plot spectra of nun-periodic signals
- Parameter estimation
- Analysing chirped signals

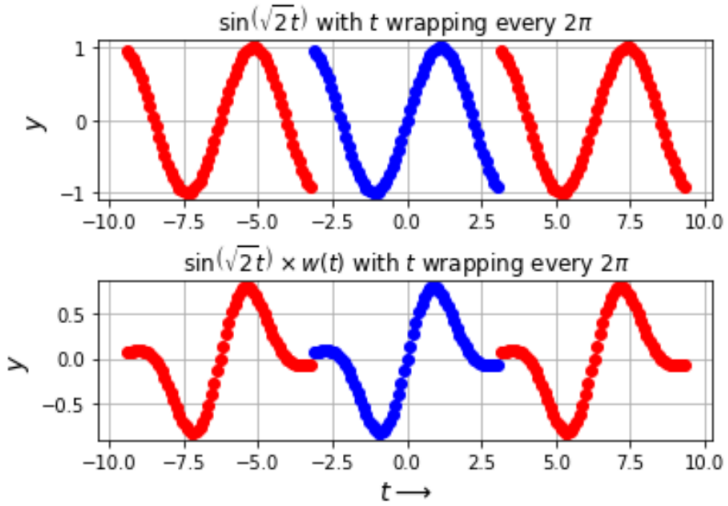
Spectra of $\sin(\sqrt{2}t)$

Though this is a periodic function in time, the range $-\pi$ to $+\pi$ doesn't include the principal component of the signal. So, if we extend it periodically, there occur large jumps at $-\pi$ and $+\pi$, resulting in gibbs phenomenon.





As, expected, the fundamental frequencies cannot be distinguished from the plot. The reason is evident in the below plot: The gaps at end of every



period need to be removed. The second plot shows the periodic extn. of a windowed version of the signal. This reduces the gap at the $-\pi$ and $+\pi$.

We have used a Hamming window of the form

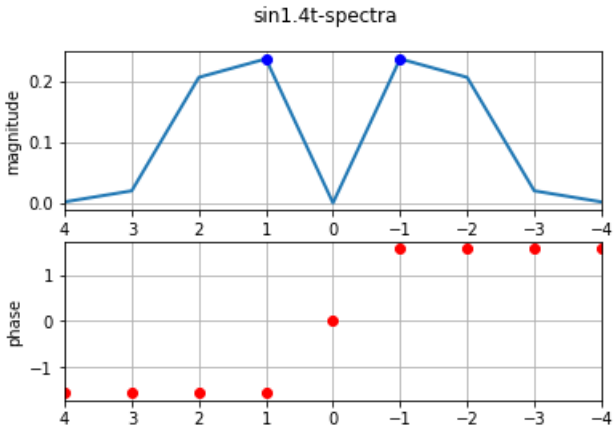
$$w[n] = \begin{cases} 0.54 + 0.46cos(\frac{2n\pi}{N-1}) if |n| < \frac{N-1}{2} \\ 0 otherwise \end{cases} \tag{1}$$

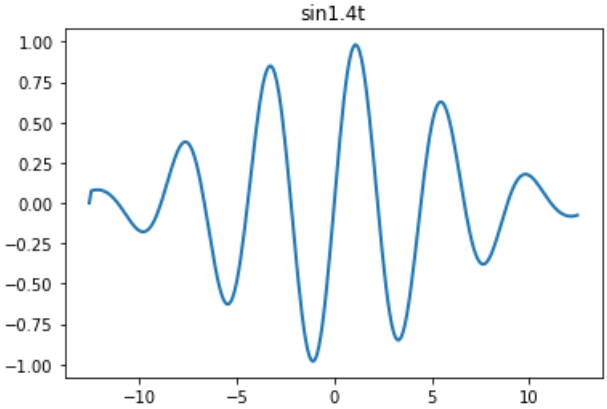
When we try to find the spectra after windowing,

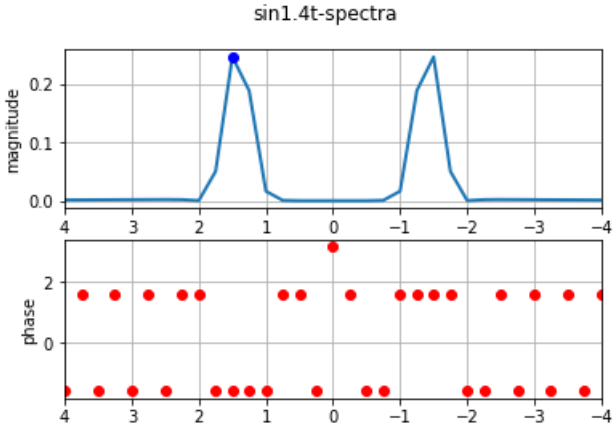
Now the peaks are more clearly distinguished. yet, as we have limited no. of points, we can't point out the fundamental component.

So, with increasing the no, of points and windowing, we get:

The peaks are more dominants and are close to 1.5 ($\sqrt{2} = 1.4$)*asexpected.*



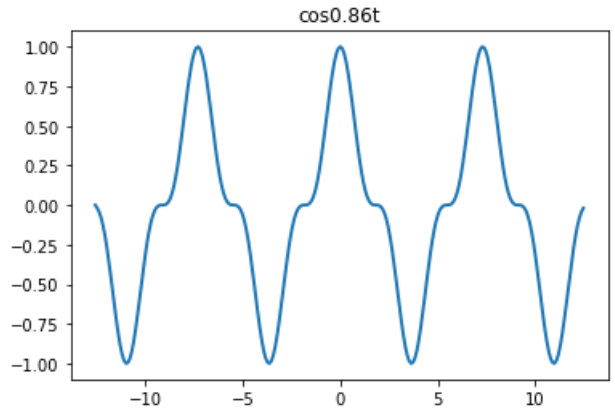


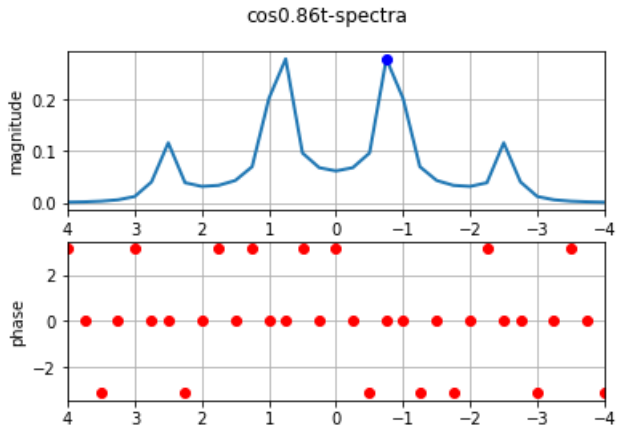


Spectra of $\cos(\mathbf{w}_0^3(t)$

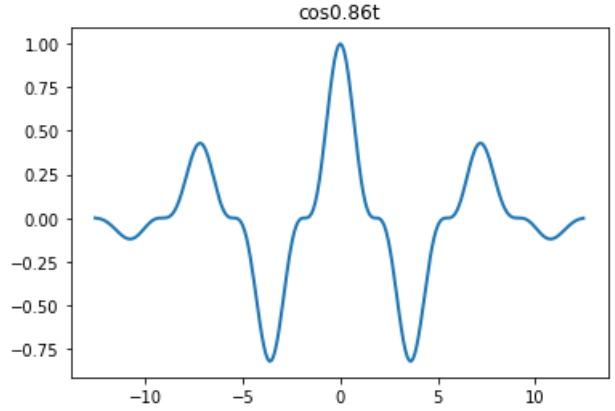
0.1 Pure sinusoid

Without using a hamming window, we get the original signal and spectra as:

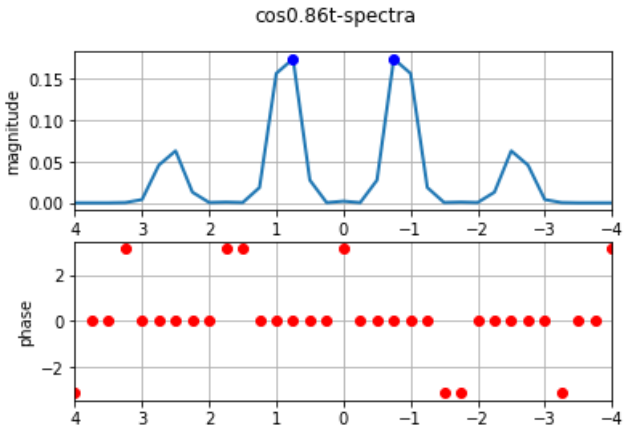




When we use a hamming window, we get



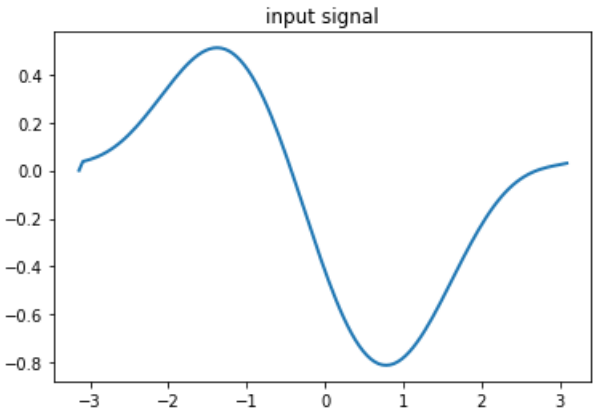
Again, as expected, when we use a window, the secondary components can also be distinguished and have identifiable peaks.



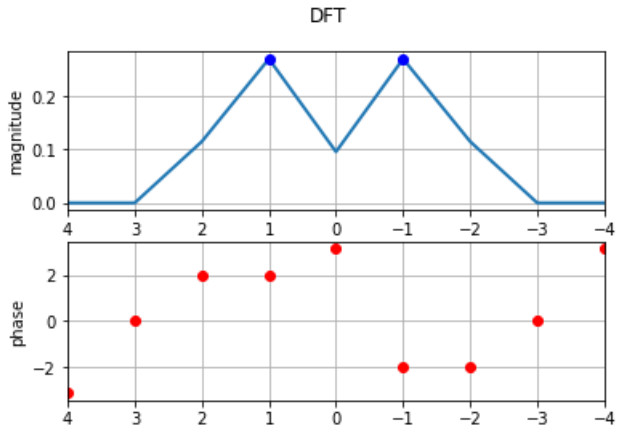
Parameter estimation for a sinusoid

Given a 128 element vector representing $cos(w_0 t + \delta)$ over a time range of $-\pi$ to π, we need to calculate the DFT and identify the parameters w_0 and δ

Checking the function for frequency $w_0 = 1$ and δ=2, first, we plot the time domain vector:



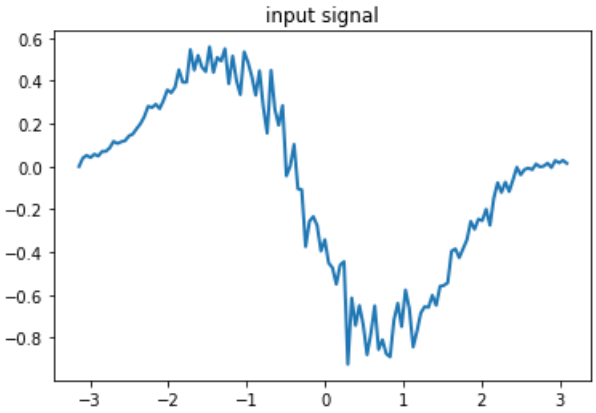
The, we calculate the DFT and identify the peaks. The peaks will be identified at $\pm w_0$.



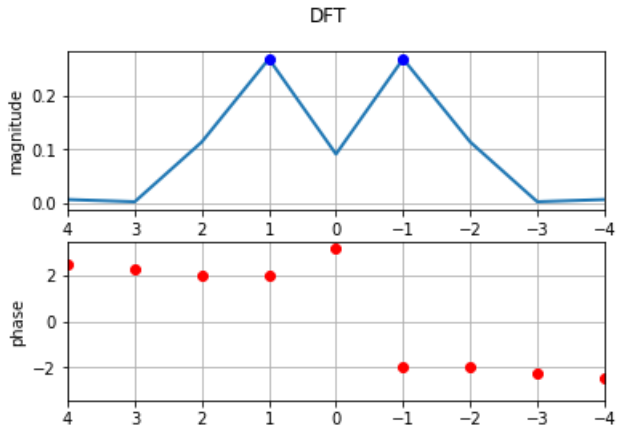
0.2 Sinusoid with white gaussian noise

We add a gaussian distribution of noise to the signal and again try to identify the parameters.

The time domail signal with noise:



The resultant spectrum comes out to be:



The noise doesn't affect the magnitude plot much and the peaks continue to remain clearly defined at $\pm w_0$. But the phase plot has undergone considerable change.

Then, for a sinusoid, the phase of the frequency plot at fundamental frequencies will be equal to the phase shift of the original signal.

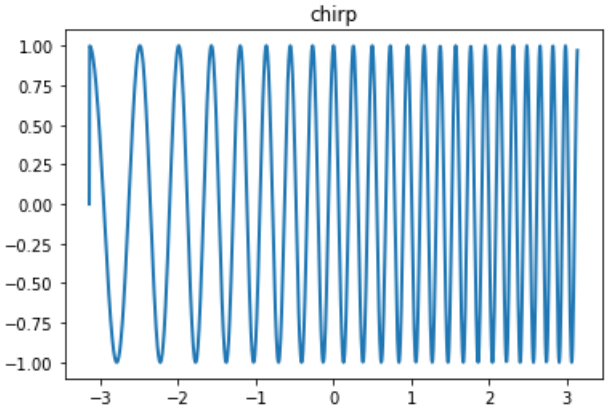
The **estimatevec()** function does the above calculations and returns the fundamental frequency and delta.

Analysis of chirped signals

We analyse the spectra of the input signal:

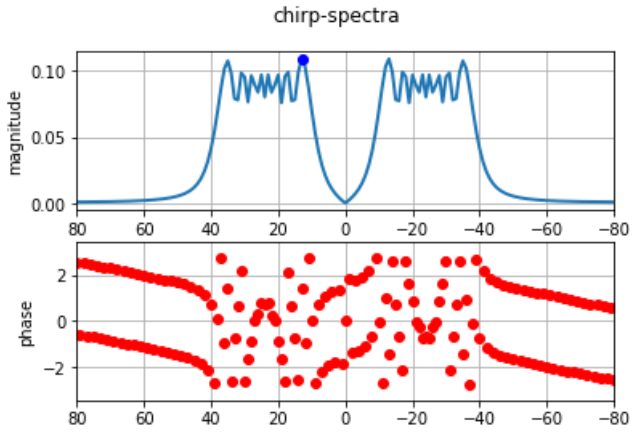
$$x(t) = cos(16(1.5 + \frac{t}{2\pi}))$$

The input signal is :

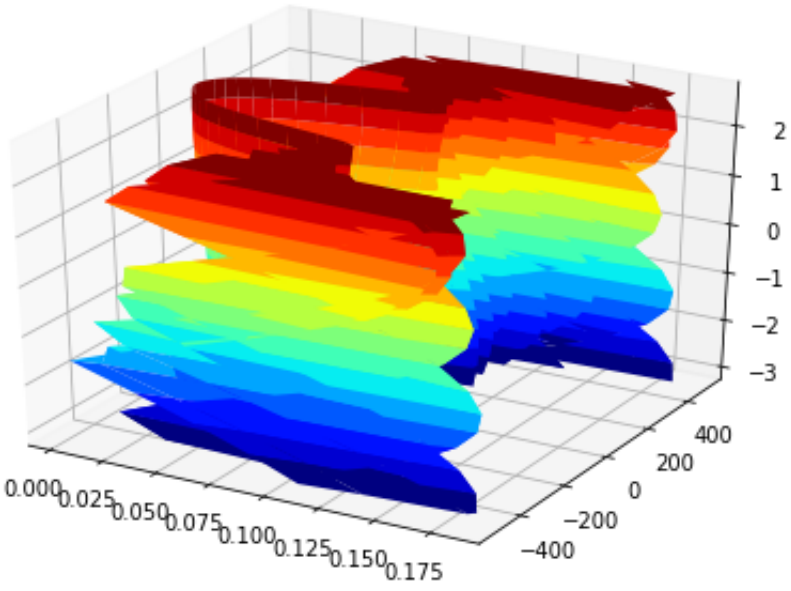


It has a time varying frequency component, changing from 16 to 32

radians per second. Its spectrum for a 1024 element vector in the time range of $-\pi$ to π is plotted as:



To analsye how the frequency changes over time, we plot a time surface plot:



It is calculated by breaking the 1024 element vector into 16 vectors of 64 elements each and finding the individual DFT components for those specific time ranges and combining them into 1 plot.

Python code

```
import numpy as np
import numpy.fft as fft
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d.axes3d as p3

func_dict = {"cos0.86t":lambda t: np.cos(0.86*t)**3,
             "sin1.4t":lambda t: np.sin(np.sqrt(2)*t),
             "chirp":lambda t: np.cos(24*t + 8*t*t/np.pi)}

def findDFT(x,t_lim,N,wnd = True,disp=True,xlim=4):
    t = np.linspace(-t_lim,t_lim,N+1);t = t[:-1]
    dt = t[1] - t[0]
    fmax = 1/dt
    f = 0
    if type(x) is np.ndarray:
        y = x
    elif type(x) is str:
        y = func_dict[x](t)
        f = 1
    else:
        raise Exception("Enter a vector or a dictionary key")
    n = np.arange(N)
    if wnd:
        wnd = (0.54 + 0.46*np.cos(2*np.pi*n/(N)))
        y = y*fft.fftshift(wnd)
    y[0]=0
    Y = fft.fftshift(fft.fft(fft.fftshift(y)))/N
    if disp:
        maxima = abs(Y).max()
        wo = np.where(abs(Y)==maxima)
        plt.figure()
        plt.plot(t,y,lw=2)
        plt.title(x if f else "input signal")
        plt.show()
        w = np.linspace(-fmax*np.pi,fmax*np.pi,N+1);w = w[:-1]
        plt.figure()
        plt.suptitle("{}-spectra".format(x) if f else "DFT")
        plt.subplot(2,1,1)
        plt.plot(w,abs(Y),w[wo],abs(Y[wo]),"bo",lw=2)
        plt.xlim([xlim,-xlim])
        plt.ylabel("magnitude")
        plt.grid(True)
```

```
        plt.subplot(2,1,2)
        plt.plot(w,np.angle(Y),"ro",lw=2)
        plt.xlim([xlim,-xlim])
        plt.ylabel("phase")
        plt.grid(True)
        plt.show()
    return Y

def estimateVEC(x):
    Y = findDFT(x,np.pi,128,wnd=True,disp=True)
    dt = 2*np.pi/128
    fmax = 1/dt
    w = np.linspace(-np.pi*fmax,np.pi*fmax,129);w=w[:-1]
    ind = np.asarray(np.where(abs(Y)==abs(Y).max()))
    try:
        ind = ind[0][1]
    except:
        try:
            ind = ind[1]
        except:
            pass
    delta = np.angle(Y[ind])
    wo = w[ind]
    return wo,delta

findDFT("sin1.4t",np.pi,64,wnd=False,disp=True)
findDFT("sin1.4t",np.pi,64,wnd=True,disp=True)
findDFT("sin1.4t",4*np.pi,256,wnd=True,disp=True)

findDFT("cos0.86t",4*np.pi,256,wnd=True,disp=True)
findDFT("cos0.86t",4*np.pi,256,wnd=False,disp=True)

f = 1
d = 2
t = np.linspace(-np.pi,np.pi,129);t = t[:-1]
x = np.cos(f*t + d)
wo,delta = estimateVEC(x)
print("calculated {}, actual {}".format(wo,f))
print("calculated {}, actual {}".format(delta,d))

a = 0.1
x += a*np.random.randn(128)
wo,delta = estimateVEC(x)
print("calculated omega{}, actual omage {}".format(wo,f))
```

```
print("calculated delta{}, actual delta{}".format(delta,d))

findDFT("chirp",np.pi,1024,wnd=False,disp=True,xlim=80)
t = np.linspace(-np.pi,np.pi,1025);t = t[:-1];dt = t[1]-t[0]
x = func_dict["chirp"](t)
x_mat = x.reshape((64,16))
x_t = x_mat.transpose()

Y_t = np.zeros((16,64))
i = 0
for row in x_t:
    Y_t[i] = abs(findDFT(row,np.pi,64,wnd = False,disp=False))
    i += 1
Y = Y_t.transpose()
fmax = 1/dt
wx = np.linspace(-np.pi*fmax,np.pi*fmax,65);wx=wx[:-1]
wy = np.linspace(-np.pi,np.pi,17);wy=wy[:-1]
Wx,Wy = np.meshgrid(wy,wx)
ax = p3.Axes3D(plt.figure())
ax.plot_surface(Y,Wy,Wx,rstride=1,cstride=1,cmap='jet')
plt.show()
```

Conclusion

The discrete fourier tranform or DFT converts finite length sequence of equally-spaced samples of a function into a same-length sequence of equally-spaced samples of the discrete-time Fourier transform (DTFT) and can thus can be used for analysis of the spectra of time domain signals computationally. When the signals are not periodic, we use windowing to suppress the ripples and get a better estimate through the DFT.