EE2703 Applied Programming Lab - Assignment 10

Name: Rajat Vadiraj Dwaraknath
Roll Number: EE16B033

April 24, 2018

1 Introduction

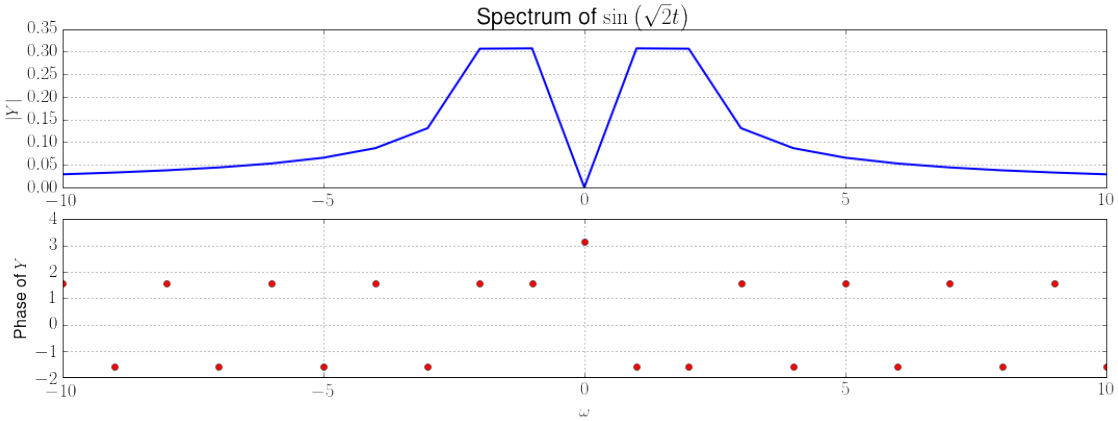
In this assignment, we continue our analysis of signals using Fourier Transforms. This time, we focus on finding transforms of functions which are discontinuous when periodically extended. An example of this is $\sin(\sqrt{2}t)$. The discontiuity causes fourier components in frequencies other than the sinusoids frequency which decay as $\frac{1}{\omega}$, due to Gibbs phenomenon. We resolve this problem using the process of windowing. In this assignment, we focus on one particular type of window - the Hamming window. We use this windowed transform to analyse signals known to contain a sinusoid of unknown frequencies and extract its phase and frequency. We then perform a sliding DFT on a chirped signal and plot a spectrogram or a time-frequency plot.

2 Worked examples

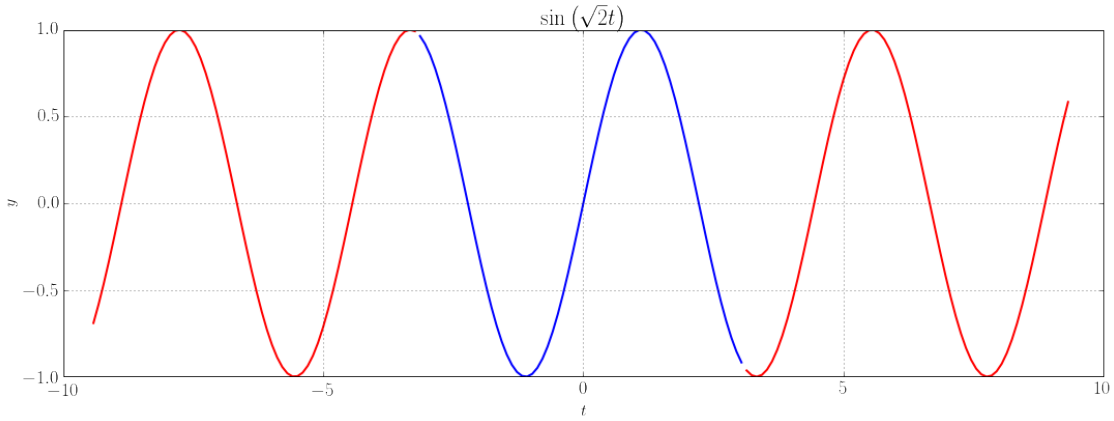
The examples given are worked through below:

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

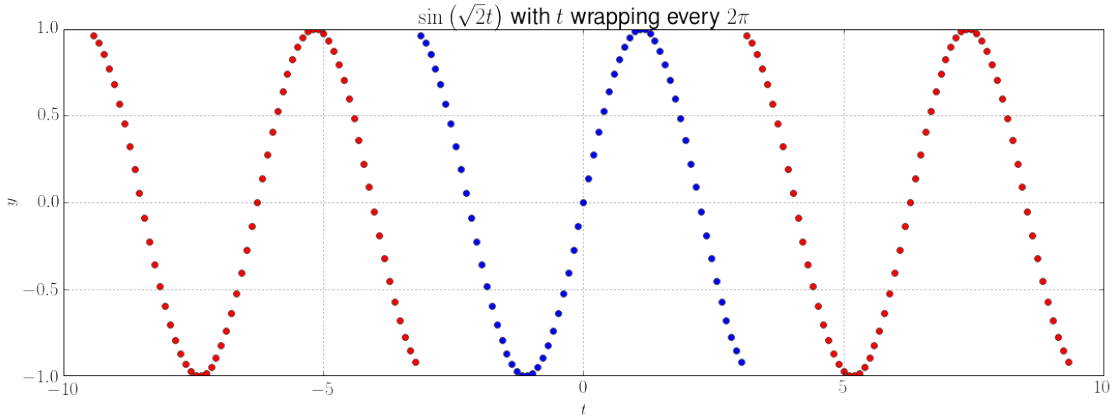
The spectrum of $\sin(\sqrt{2}t)$ is plotted below:



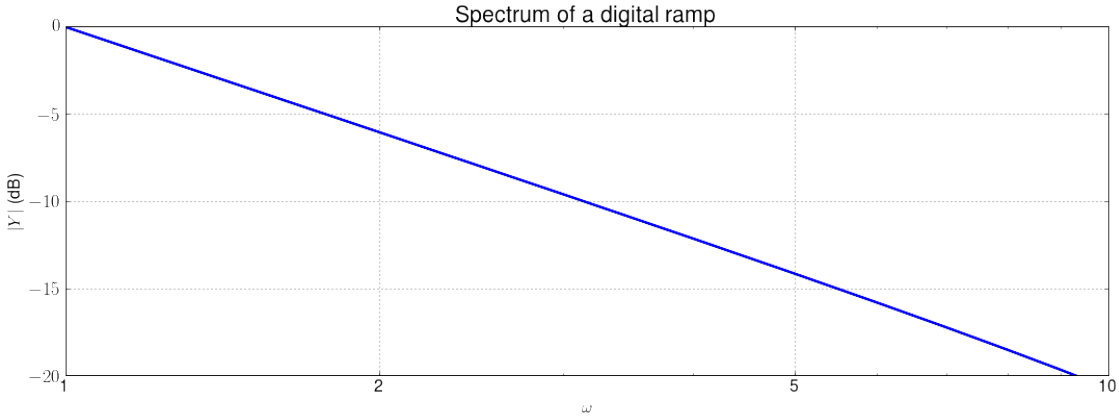
This is the function for which we want to find the DFT:



However, when we calculate the DFT by sampling over a finite time window, we end up calculating the DFT of the following periodic signal:



This results in discontinuites in the signal. These discontinuities lead to spectral components which decay as $\frac{1}{\omega}$. To confirm this, we plot the spectrum of the periodic ramp below:



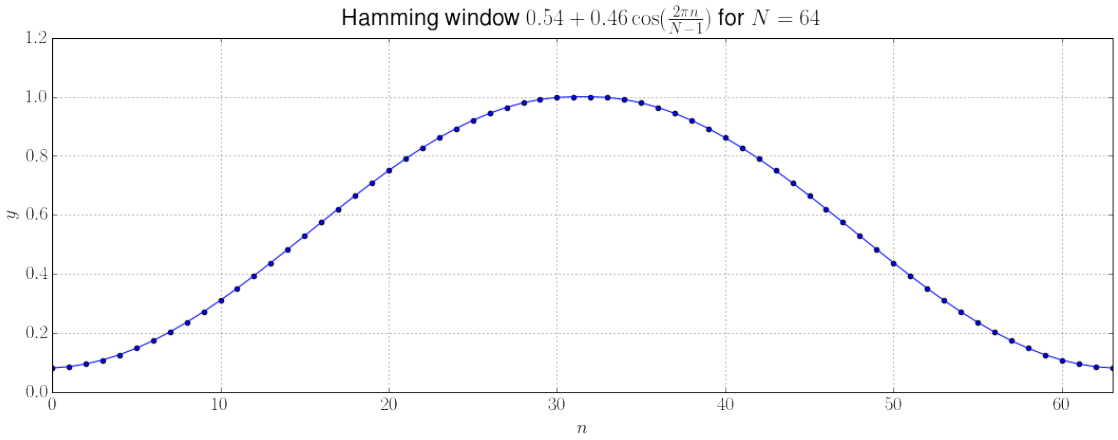
2.2 The Hamming window

We resolve the problem of discontinuities by attenuating the signal near the endpoints of our time window, to reduce the discontinuities caused by periodically extending the signal. This is done by multiplying by a so called windowing function. In this assignment we use the Hamming window of size N:

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

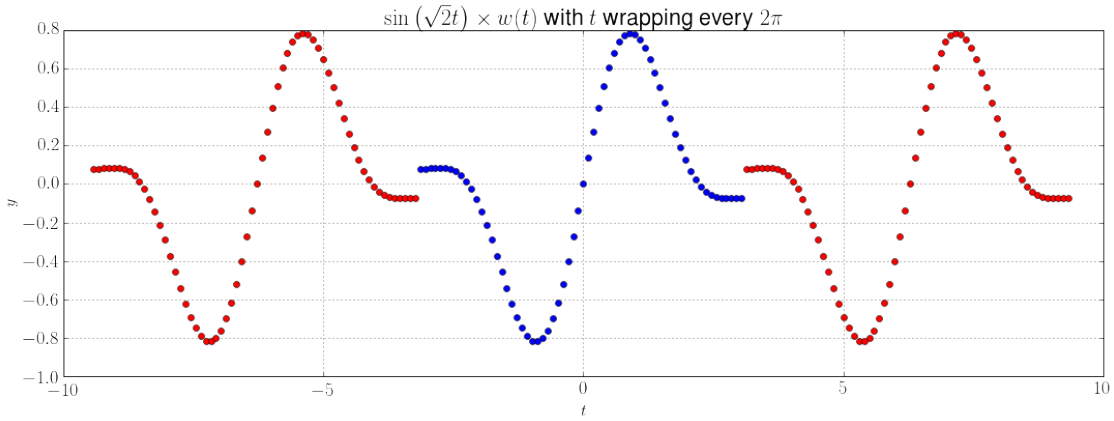
```
def hamming(n):
    n = array(n)
    N = n.shape[0]
    return 0.54+0.46*cos(2*pi*n/(N-1))
```

We plot it below for $N = 64$

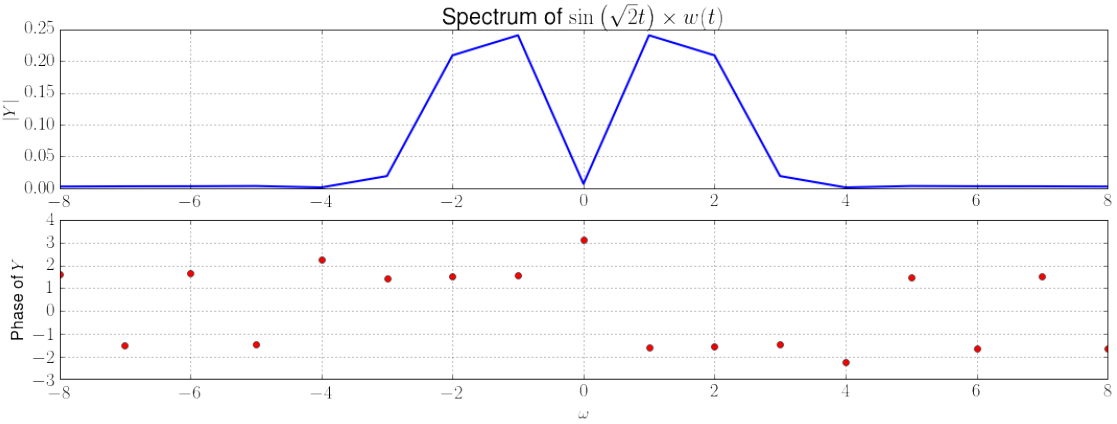


2.3 Spectrum after windowing

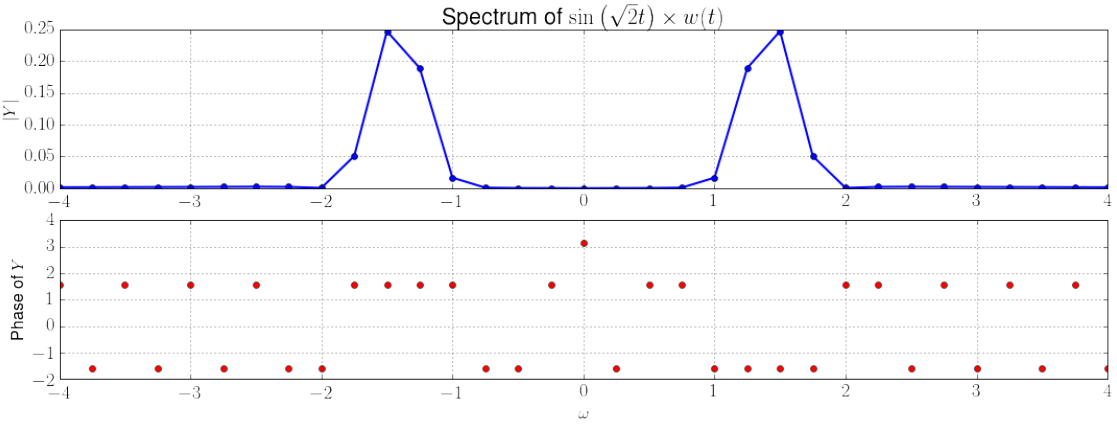
We now multiply our signal with the Hamming window and then periodically extend it:



The spectrum is found below using a window size of 2π:



And with a window size of 8π



We observe that the spectrum has much more components with a magnitude of zero. However, the peak at the frequency of $\sqrt{2}$ has been broadened due to spectral leakage.

3 FFT with hamming window

3.1 The function

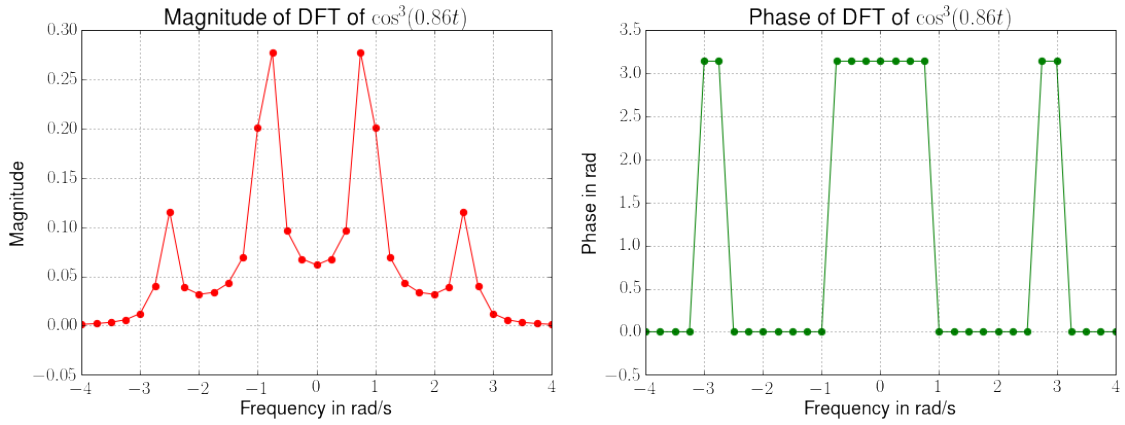
We first write a general function to find the windowed FFT of a given input signal:

```
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=True, window=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)
    if window:
```

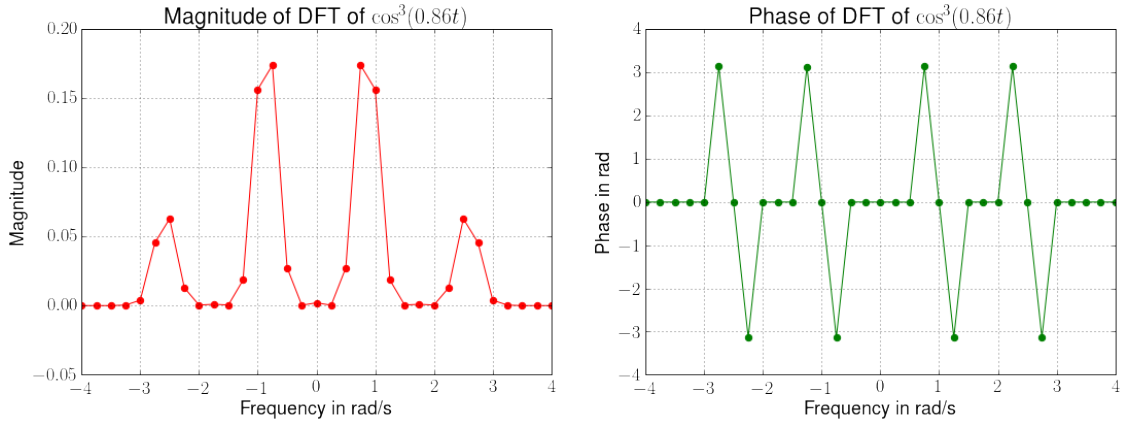
```
    wnd = fftshift(hamming(arange(points)))
    y = y*wnd
Y =  fftshift( fft(y))/points

if not plot:return w,Y
# 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')
ax1.scatter(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.plot(w,ph,color='green')
ax2.scatter(w,ph,color='green',s=scatter_size)
ax2.set_xlim(wlim)
ax2.grid()

show()
return w,Y
```

3.2 FFT of $\cos^3(0.86t)$

We first find the FFT without the Hamming window:



We repeat the process, but with Hamming window:




```
Estimated first peak frequency: 0.8597
```

- We observe that in the case without a Hamming window, a lot of energy in the spectrum was in frequencies other than those of the signal. This is because of the Gibbs phenomenon.
- We observe that the windowed transform is much better in terms of the magnitude spectrum. Only components near the frequencies of the input signal are present, while others are mostly 0. The reason that some frequencies near the actual peak are present is because multiplying by a window in the time domain corresponds to a convolution in the frequency domain with its fourier transform. This means that the delta functions in the frequency domain are smeared out by the spectrum of the Hamming window.

4 Estimating ω and δ

The task is to estimate the values of ω and δ given 128 samples of $\cos(\omega t + \delta)$ for $(-\pi, -\pi)$.

4.1 Estimator 1

The first approach is to take the windowed and non-windowed DFT of the 128 samples. We then find a weighted average of frequencies weighted by the magnitude of the DFT to obtain the peak frequency ω. To find δ, phase values around the peak frequency are averaged.

```
def estimateWD1(vec,l=6,window=True):
    """Estimate the value of omega and delta assuming vec contains
    128 samples of cos(omega*t + delta) in (-pi,pi). Uses the magnitude
    and phase spectra to estimate omega and delta respectively."""

    N = 128
    delta_t = 2*pi/N
    w_max = pi/delta_t
    delta_w = 2*w_max/N

    w = linspace(-w_max,w_max,N+1)[:-1]

    if window:
        vec_ = vec*fftshift(hamming(arange(N)))
    else:
        vec_ = vec

    y = fft(fftshift(vec_))/N
    mag = abs(y)

    points = mag[:l]
    ind = arange(l)
    omega = np.sum(points*ind)/np.sum(points)
    start=0
    if omega>1:
        start=1
    delta = mean(angle(y)[start:3])
    return omega,delta
```

A function to test a general estimator is written below:

```
def testEstimator(est,trials=100,noise = False):
    """Test an estimator of omega and delta."""
    t = linspace(-pi,pi,128+1)[:-1]

    oms=[]
    ompreds=[]

    dels = []
    delpreds = []

    for i in range(int(trials)):
        omega = 0.5+rand()
```

```
        oms.append(omega)
        delta = pi*(rand()-0.5)
        dels.append(delta)

        v = cos(omega*t+delta)
        if noise:
            v += 0.1*randn(128)
        om, de = est(v)

        ompreds.append(om)
        delpreds.append(de)

    oms = array(oms)
    dels = array(dels)
    ompreds = array(ompreds)
    delpreds = array(delpreds)
    omerr = mean(abs(oms-ompreds))
    delerr = mean(abs(dels-delpreds))
    print("MAE for omega: {:.4f}\tMAE for delta: {:.4f}".format(omerr,delerr))
    return omerr,delerr
```

We test the first estimator for 1000 trials with random ω and δ. We then report the mean absolute error. We iterate over different averaging window sizes k, to find the window with the least error.

```
Mean absolute errors for different window sizes:
Estimator 1 without noise:
k=1         MAE for omega: 0.9815         MAE for delta: 0.2814
k=2         MAE for omega: 0.3537         MAE for delta: 0.2744
k=3         MAE for omega: 0.1263         MAE for delta: 0.1209
k=4         MAE for omega: 0.1101         MAE for delta: 0.1095
k=5         MAE for omega: 0.1173         MAE for delta: 0.1126
k=6         MAE for omega: 0.1162         MAE for delta: 0.1081
k=7         MAE for omega: 0.1220         MAE for delta: 0.1010
k=8         MAE for omega: 0.1287         MAE for delta: 0.0937
k=9         MAE for omega: 0.1457         MAE for delta: 0.0974
k=10         MAE for omega: 0.1539         MAE for delta: 0.0921


Mean absolute errors for different window sizes:
Estimator 1 with noise:
k=1         MAE for omega: 1.0072         MAE for delta: 0.2706
k=2         MAE for omega: 0.3392         MAE for delta: 0.2818
k=3         MAE for omega: 0.1227         MAE for delta: 0.1305
k=4         MAE for omega: 0.1143         MAE for delta: 0.1193
k=5         MAE for omega: 0.1092         MAE for delta: 0.1042
k=6         MAE for omega: 0.1132         MAE for delta: 0.0921
k=7         MAE for omega: 0.1421         MAE for delta: 0.0821
```

```
k=8        MAE for omega: 0.1789        MAE for delta: 0.0703
k=9        MAE for omega: 0.2548        MAE for delta: 0.0609
k=10        MAE for omega: 0.3170        MAE for delta: 0.0512
```

It is clear from the above results that a value of k around 4 or 5 works best. The estimator is not vastly affected by the noise added. This is because the noise mainly contains very high frequency components, which will not affect the low frequency parts of the spectrum. We try another approach to estimate delta below:

4.2 Estimator 2

Using the value of *omega* estimated by weighted averaging (say ω'), we then fit a model using least squares and estimate δ. In particular, we fit the 128 unwindowed samples to the model:

$$x(t) = A\cos(\omega' t) + Bsin(\omega' t)$$

We can then estimate the phase δ as follows:

$$\delta = -\tan^{-1}\left(\frac{B}{A}\right)$$

The estimator is written below:

```
def estimateWD2(vec,l=6,window=True):
    """Estimate the value of omega and delta assuming vec contains
    128 samples of cos(omega*t + delta) in (-pi,pi)."""

    N = 128
    delta_t = 2*pi/N
    w_max = pi/delta_t
    delta_w = 2*w_max/N

    w = linspace(-w_max,w_max,N+1)[:-1]

    if window:
        vec_ = vec*fftshift(hamming(arange(N)))
    else:
        vec_ = vec

    y = fft(vec_)/N
    mag = abs(y)

    points = mag[:l]
    ind = arange(l)
    omega = np.sum(points*ind)/np.sum(points)


    t = linspace(-pi,pi,N+1)[:-1]
    A = vstack((cos(omega*t),sin(omega*t))).T
```

```
    #print(vec.shape)
    a,b = lstsq(A,vec)[0]
    delta = -arctan(b/a)

    return omega,delta
```

We test it with varying weighted averging windows again:

```
Estimator 2 without noise:
k=1        MAE for omega: 1.0086        MAE for delta: 0.7789
k=2        MAE for omega: 0.3435        MAE for delta: 0.1186
k=3        MAE for omega: 0.1260        MAE for delta: 0.0332
k=4        MAE for omega: 0.1155        MAE for delta: 0.0322
k=5        MAE for omega: 0.1141        MAE for delta: 0.0220
k=6        MAE for omega: 0.1140        MAE for delta: 0.0268
k=7        MAE for omega: 0.1185        MAE for delta: 0.0316
k=8        MAE for omega: 0.1363        MAE for delta: 0.0417
k=9        MAE for omega: 0.1415        MAE for delta: 0.0356
k=10        MAE for omega: 0.1494        MAE for delta: 0.0353


Estimator 2 with noise:
k=1        MAE for omega: 1.0077        MAE for delta: 0.7626
k=2        MAE for omega: 0.3589        MAE for delta: 0.1351
k=3        MAE for omega: 0.1226        MAE for delta: 0.0406
k=4        MAE for omega: 0.1148        MAE for delta: 0.0359
k=5        MAE for omega: 0.1136        MAE for delta: 0.0488
k=6        MAE for omega: 0.1223        MAE for delta: 0.0538
k=7        MAE for omega: 0.1339        MAE for delta: 0.0345
k=8        MAE for omega: 0.1796        MAE for delta: 0.0527
k=9        MAE for omega: 0.2369        MAE for delta: 0.0725
k=10        MAE for omega: 0.3129        MAE for delta: 0.1288
```

We again observe that a value of 4 or 5 is best suited for the weighted averaging window size. We also observe that the error in estimating δ using the least squares approach is much less than using the phase of the DFT. The effect of noise is similar as in the first case. However, we must remember that the least squares approach is more computationally expensive compared to computing just one DFT.

5 Spectrogram analysis of a chirp

We now analyse the frequency spectrum of a particular signal called a chirp.

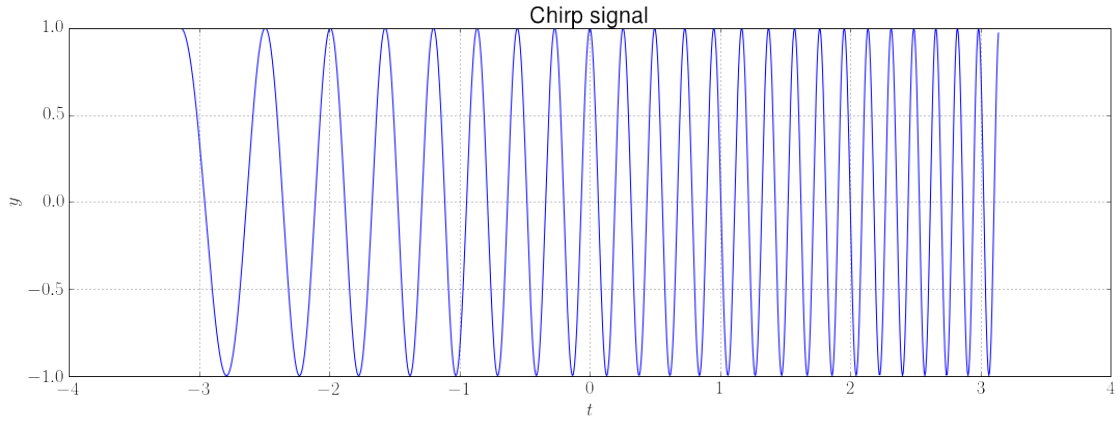
5.1 The chirp signal

The chirp signal is a sinusoid whose frequency varies linearly with time. In this assignment, we look at the following chirp:

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

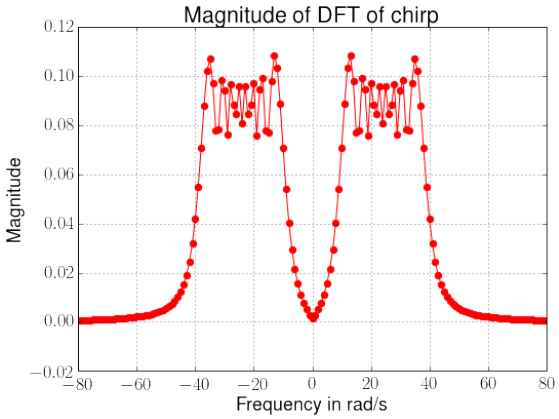
```
def chirp(t):
    return cos(16*t*(1.5 + t/(2*pi)))
```

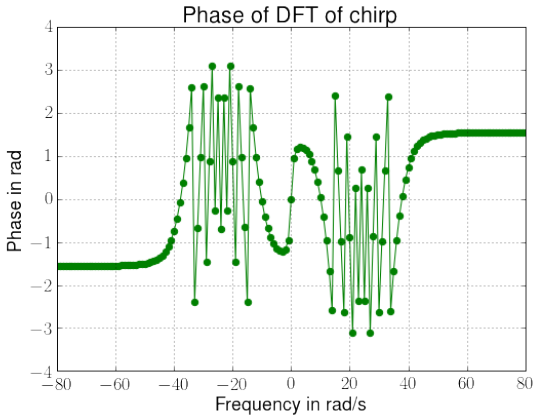
The chirp is plotted below:



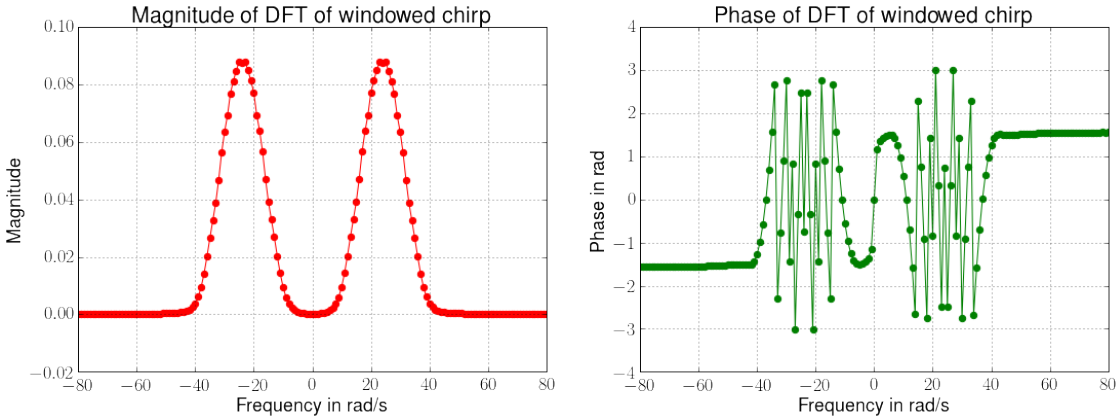
5.2 FFT of the chirp

We find the FFT of the chirp without windowing:





We observe that various frequencies in the range of 5-50 rad/s are present in the spectrum. This is because of the $1/\omega$ decay of the Gibbs phenomenon. Let us window the chirp and observe the differences:



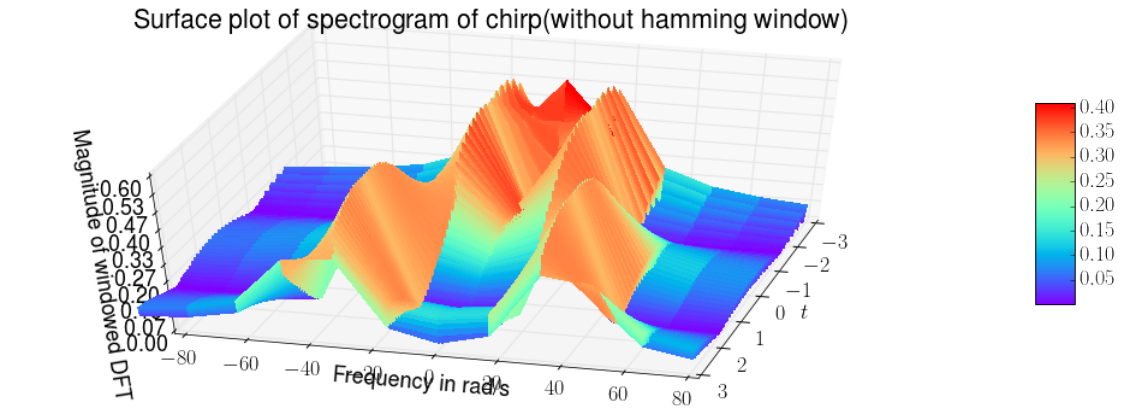
We now observe that the frequencies are more confined to the range between 16 and 32, as expected. The extra components due to the discontinuity have been suppressed.

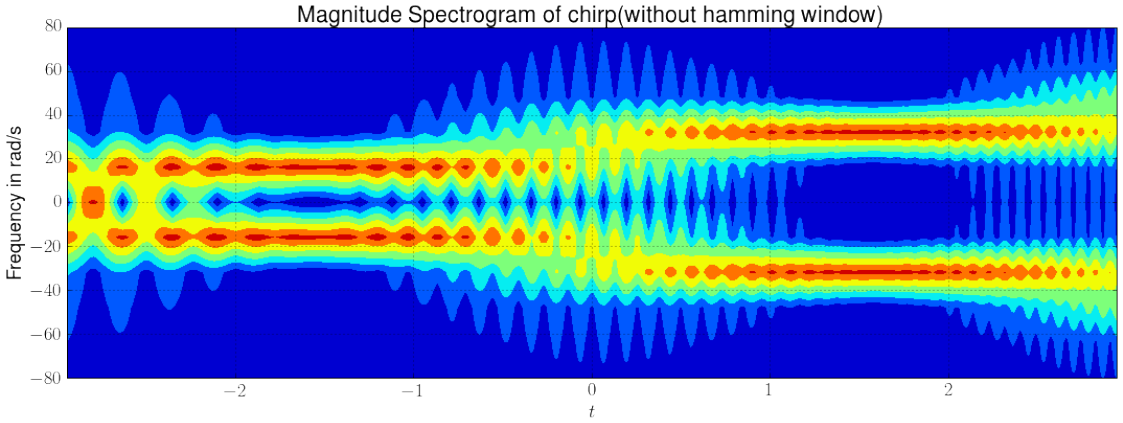
5.3 FFT over multiple windows

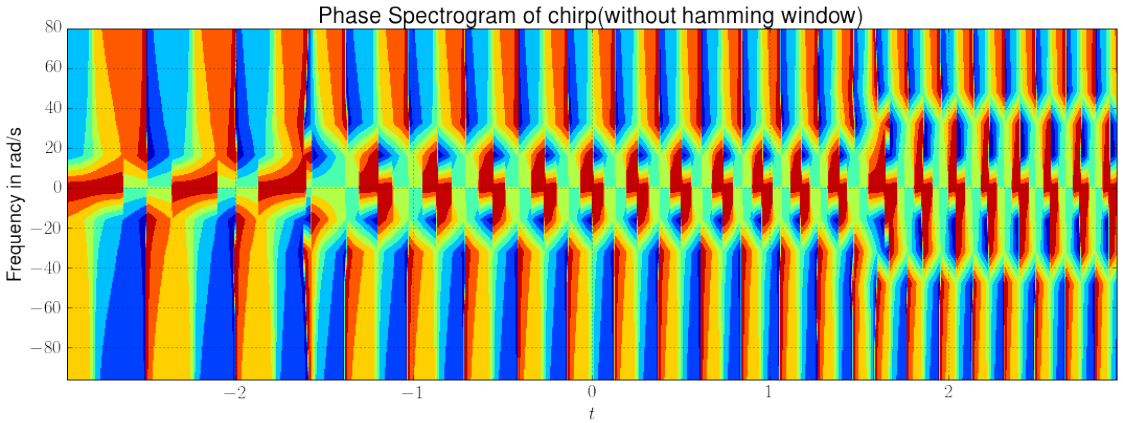
To obtain a better picture of what is going in the chirp signal, we take the DFT of a small window of samples around each time instant, and plot a 2D surface of the resulting spectra vs time. We initially find the DFTs without using a Hamming window:

```
N = 1024
window = 64
n_wins = int(N/window)
delta_t = 2*pi/N
w_max = pi/delta_t
delta_w = 2*w_max/N

t = linspace(-pi,pi,N+1)[:-1]
y = chirp(t)

ys=[]
T=26
for i in arange(0,N-window):
    y_ = y[i:i+window]
    Y = 1/window * fftshift(fft(y_))
    ys.append(Y[T:-T])

t = linspace(-pi,pi,N)[int(window/2):-int(window/2)]
w = linspace(-w_max,w_max,window+1)[:-1][T:-T]
tt,ww = meshgrid(t,w)
ys = array(ys)
```

We get the following spectrogram plots:





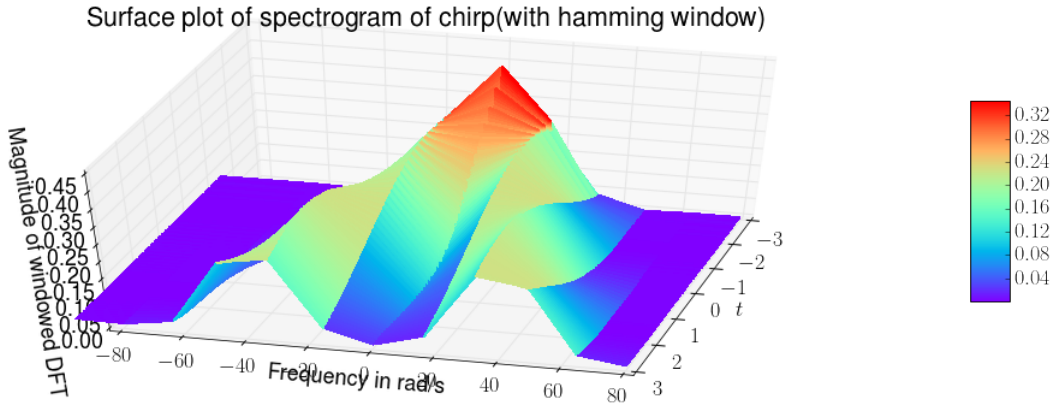


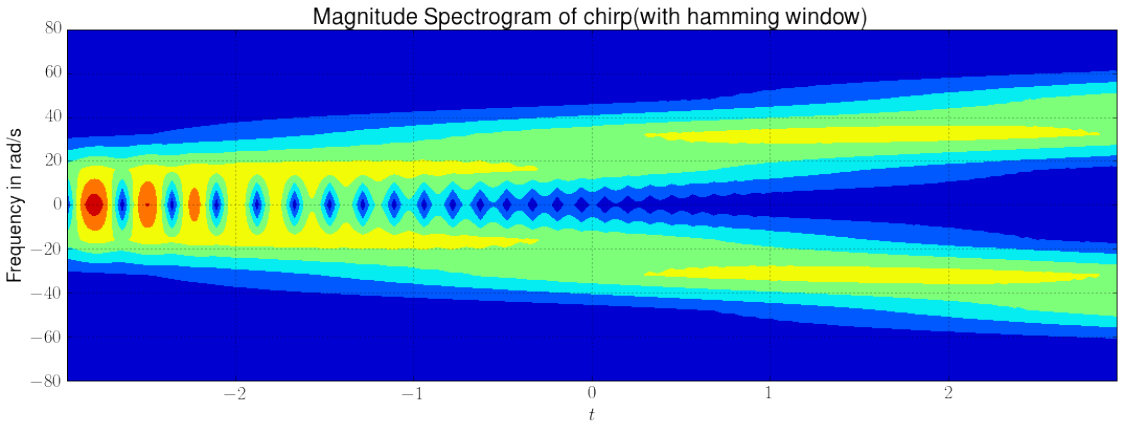
- We observe that the frequency components with high magnitude are concentrated around a frequency of 16 in the first half of the chirp, and then suddenly move to a frequency of 32 in

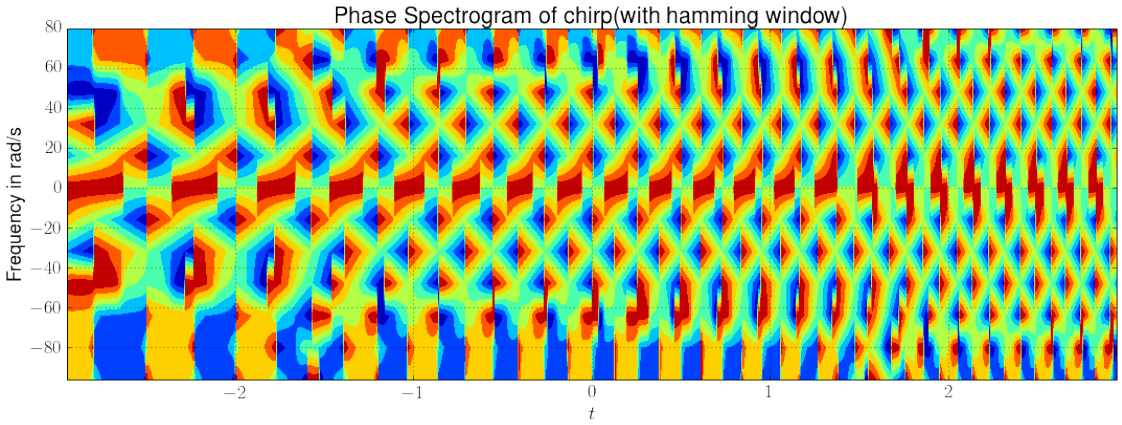
the second half. We expect only the inital and final frequencies to be 16 and 32 respectively, and not over the whole of the first and second halfs.

- This is because we have not windowed each section before taking the DFT. This results in the component with integer valued frequencies show up with higher amplitudes, and the remaining frequencies decay as $\frac{1}{\omega}$ due to the Gibbs phenomenon.

We repeat the process, but now we use a Hamming window before taking the DFT of each section of 64 samples.







- We observe a much more gradual change in the frequency in the signal.
- The change is linear from 16 to 32, matching with our expectations.

6 Conclusions

- From the above examples, it is clear that using a Hamming window before taking a DFT helps in reducing the effect of Gibbs phenomenon arising due to discontinuities in periodic extensions.
- However, this comes at the cost of spectral leakage. This is basically the blurring of the sharp peaks in the DFT. It occurs because of convolution with the spectrum of the windowing function. Deltas in the original spectrum are smoothed out and replaced by the spectrum of the windowing function.
- We used this windowed DFT to estimate the frequency and phase of an unknown sinusoid from its samples.
- By performing localized DFTs at different time isntants, we obtained a time-frequency plot which allowed us to better analyse signals with varying frequencies in time.