Assignment-10 : Spectra of non-periodic signals

Sai Gautham Ravipati - EE19B053

May 24, 2021

Contents

1 Introduction

In the previous assignment, the spectra of periodic signals has been looked upon, where the following approach was used.

1. Sample the signal so that $f_{Nyquist}$ is met, and so that Δf is small enough. Generate the frequency axis from -$\frac{f_{max}}{2}$ to +$\frac{f_{max}}{2}$, taking care to drop the last term.
2. Ensure that the signal starts at $t = 0+$ and ends at $t = 0$ -.
3. Use 2^k samples
4. Obtain the DFT. Rotate the samples so that they go from f = -$\frac{f_{max}}{2}$ to +$\frac{f_{max}}{2} - \Delta f$.

When the same procedure is adopted for the case of non-periodic spectra, the sampled signals tend to have a large discontinuity, which introduce frequency components that decay as $\frac{1}{\omega}$ as first derivative is impulsive, put alternatively by the Gibbs Phenomena. Windowing approach is introduced to resolve the same, where frequency domain convolution is done with response of that of the hamming window, and this makes the spectrum much cleaner. The rest of the analysis follows.

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

Before starting of with the analysis, the following functions have been used to generate the spectrum and to plot it. Several aspects of the function are not required at the start, and shall be used as we move further. The inputs and outputs of the function are mentioned in commented part.

1. Generation of Spectrum :

```
def spectrum_gen(t_range, N, f, wnd_flag, fft_flag, noise):

    """
    This function generates Y,w of the spectrum for various input functions
    and returns them. The inputs to the function are :
    t_range   : [low_t,up_t] where low_t is the lower limit and up_t is the
                upper limit of t.
    N         : Number of points for which the function is calculated.
    f         : Name of the function which has t as argument.
    wnd_flag  : Boolean value, 1 if hamming window is to be used orwise 0
                is passed.
    fft_flag  : Boolean value, 1 to perform 'y[0] = 0' and fftshift(y),
                orwise  0 is passed.
    noise     : Boolean value, 1 is passes to add Gaussian noise of A = 0.1
                is to be added, orwise 0 is passed.

    Output - Y : DFT Spectrum points which is of the dimension N.
             w : The corresponding values of w, which map to values of Y.
    """
    t  = np.linspace(t_range[0], t_range[-1], N+1)
    t  = t[:-1]
    dt = t[1]-t[0]
    w  = np.linspace(-np.pi, np.pi, N+1)/dt
    w  = w[:-1]

```

```
    wnd = 1
    if wnd_flag :
        n   = np.arange(N)
        wnd = np.fft.fftshift(0.54+0.46*np.cos(2*np.pi*n/(N-1)))

    y = f(t)*wnd

    if noise :
      y = y + 0.1*np.random.randn(N)

    if fft_flag :
        y[0] = 0
        y = np.fft.fftshift(y)

    Y = np.fft.fftshift(np.fft.fft(y))/N
    return Y,w
```

2. Plotting of Spectrum :

```
def spectrum_plot(Y, w, y_title, x_range, plt_flag, file_name, thresh):
    """
    This function takes in Y,w and plots the spectrum various input
    functions and then saves them in the directory EE2703_ASN10. The
    inputs to the function are :
    Y         : DFT Spectrum points which is of the dimension N.
    w         : The corresponding values of w, which map to values of Y.
    y_title   : Title to be added to the plot, displayed as 'Spectrum of
                <y_ttl>'
    x_range   : Range of x axis data-points to be displayed in the plot.
    plt_flag  : 1 - Plots only those points having mag > thresh in phase
                spectrum.
                else - Plots all phase points in red with point mentioned
                in 1 as green dots.
    file_name : Output file name of the image
    thresh    : Phase of points with mag < thresh is taken as zero if
                plt_flag is 1.

    Output - None
    The plots shall be saved as EE2703_ASN10/<file_name>
    """
    plt.figure()
    plt.subplot(2,1,1)
    plt.plot(w, abs(Y), lw=2)
    plt.xlim(x_range)
    plt.ylabel(r"$|Y|$", size=12)
    plt.title(r"Spectrum of {}".format(y_title), size=14)
    plt.grid(True)

    plt.subplot(2,1,2)

    if plt_flag == 1 :
        ii = np.where(abs(Y) > thresh)
        plt.plot(w[ii], np.angle(Y[ii]), 'go', lw=2, markersize=5)

    else :
        plt.plot(w,np.angle(Y),'ro',lw=2, label = 'All Points')
        ii = np.where(abs(Y) > thresh)
```

```
39          plt.plot(w[ii], np.angle(Y[ii]), 'go', lw=2, markersize=5, label =
    'Mag > {}'.format(thresh))
40          plt.legend(loc = 'upper right')
41
42      plt.xlim(x_range)
43      plt.ylim([-4,4])
44      plt.ylabel(r"Phase of $Y$", size=12)
45      plt.xlabel(r"$\omega$", size=12)
46      plt.grid(True)
47
48      plt.savefig('EE2703_ASN10/' + file_name)
49
50      return 0
```

Please note that, unless mentioned plt_flag is taken as one, so only phase of the points with mag > thresh is being plotted.

Using the above functions the spectrum is generated, since we want to analyse the case without window at first, wnd_flag is set to 0, and noise is also set to 0, while fft_flag is set to 1, this is to make the sample corresponding to low_t to be set zero and fftshift(y) makes y start with y(t=0). This is because anti-symmetric function has a purely imaginary Fourier transform. But an anti-symmetric set of samples, need not have purely imaginary phase, but using fft_flag ensures the phase is purely imaginary. The spectrum of the function generated for $t \in [-\pi, \pi)$ with 64 samples being considered is represented in Fig. 1.

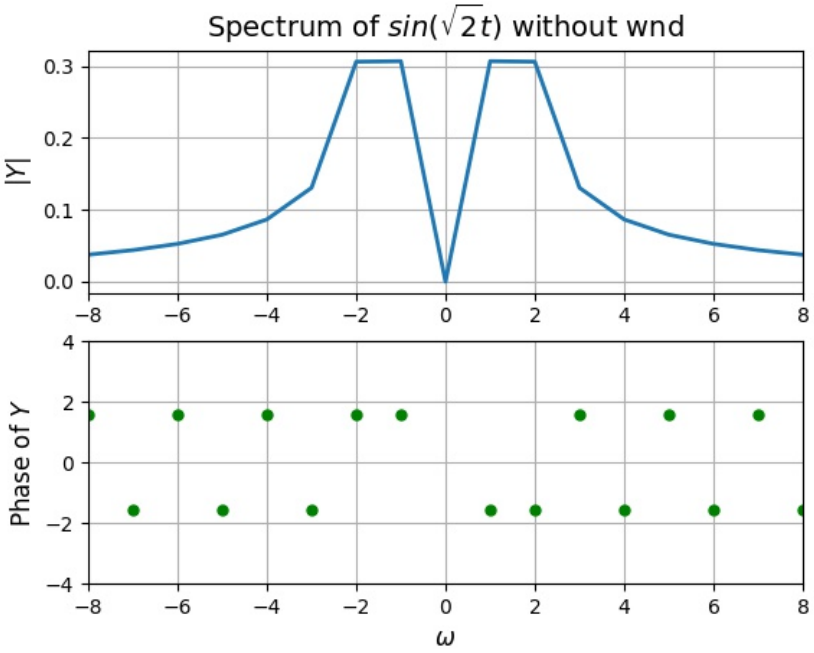

Figure 1: Spectrum of $sin(\sqrt{2}t)$ without hamming window

The phase is in accordance to what is expected, but the magnitude has a deviation from what is expected, where we get two broad peaks and the magnitude decays gradually, instead of getting two spikes. This is mainly because of the interval of function being considered. The plot of $sin(\sqrt{2}t)$ is given in Fig 2. The blue part represents the period which is being sampled. Although the function is periodic the interval which is being considered does not represent the

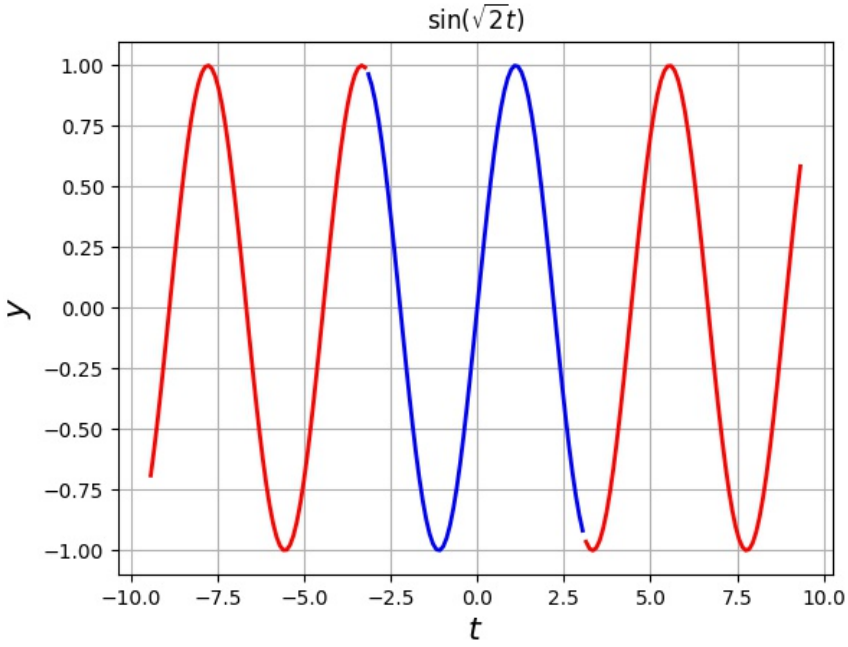

Figure 2: Plot of $sin(\sqrt{2}t)$

periodic interval and when the same is wrapped for length of the sampling period, gives the plot in Fig 3. The code used to generate this plot is given below.

```
def func_plot(t_range, N, f, y_title, wnd_flag, noise, file_name):

    """
    This function plots the sampled points, in the time period being considered
    in blue and extends it to two more periods after the given period and one
    before and plots those points in red. The inputs are :
    t_range   : [low_t,up_t] where low_t is the lower limit and up_t is the
                upper limit of t.
    N         : Number of points for which the function is calculated.
    f         : Name of the function which has t as argument.
    wnd_flag  : Boolean value, 1 if hamming window is to be used orwise 0 is
                passed.
    y_title   : Title to be added to the plot, displayed as 'Plot of <y_title>'.
    file_name : Output file name of the image.
    noise     : Boolean value, 1 is passes to add Gaussian noise of A - 0.1 is
                to be added, orwise 0 is passed.

    Output - None
    The plots shall be saved as EE2703_ASN10/<file_name>
    """
    t1 = np.linspace(t_range[0], t_range[-1], N+1)
    t1 = t1[:-1]
    T  = t_range[1] - t_range[0]
    t2 = np.linspace(t_range[0]-T,t_range[-1]-T, N+1)
    t2 = t2[:-1]
    t3 = np.linspace(t_range[0] + T,t_range[-1] + T, N+1)
    t3 = t3[:-1]

    wnd = 1
    if wnd_flag :
```

```
        n   = np.arange(N)
        wnd = np.fft.fftshift(0.54+0.46*np.cos(2*np.pi*n/(N-1)))

    y = f(t1)*wnd

    if noise :
      y = y + 0.1*np.random.randn(N)

    plt.figure()
    plt.plot(t1,y,'bo', lw=2, label = 'Sampled Points', markersize=3)
    plt.plot(t2,y,'ro', lw=2, label = 'Extension', markersize=3)
    plt.plot(t3,y,'ro', lw=2, markersize=3)
    plt.legend(loc = 'upper right')
    plt.ylabel(r"$y$", size=12)
    plt.xlabel(r"$t$", size=12)
    plt.title(r"Plot of sampled {}, wrapped".format(y_title), size=14)
    plt.grid(True)
    plt.savefig('EE2703_ASN10/' + file_name)

    return 0
```

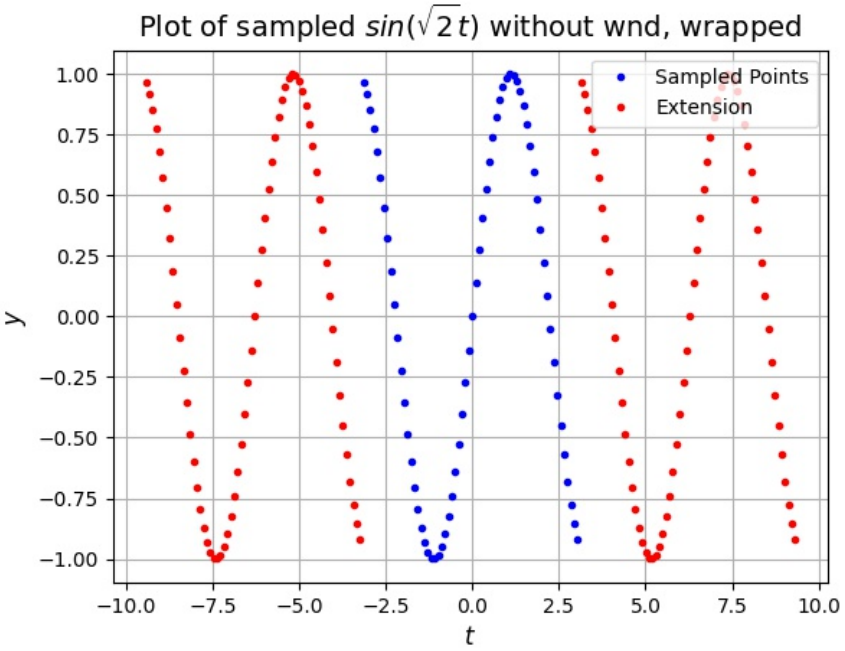

Figure 3: Sampled $sin(\sqrt{2}t)$, wrapped every period

Clearly there is a discontinuity at the end of the interval in the wrapped function, this leads to the so called Gibbs Phenomena and introduces components that decay as $\frac{1}{\omega}$, thus making the magnitude decay gradually, rather instantaneously like the case of impulse function. This can be verified by plotting the Fourier transform of the box function which decays as $\frac{1}{\omega}$ as magnitude plot shows a drop as -20 dB/decade. The same has been presented in Fig 4.

To compensate the previously mentioned effect, windowing approach is used which has been described below. The function is multiplied with the hamming window function which damps the function at the end of the periodic interval. Thus reducing the effect of Gibbs Phenomena.

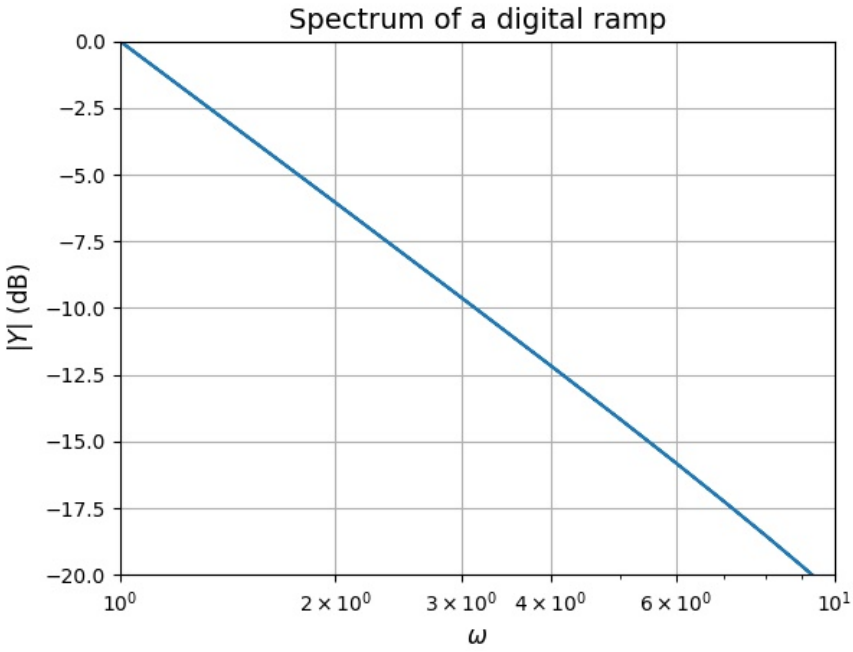

Figure 4: Fourier Spectrum of Digital Ramp

The window function is given by :

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

Plotting the previously considered function with using hamming window is given in Fig. 5.

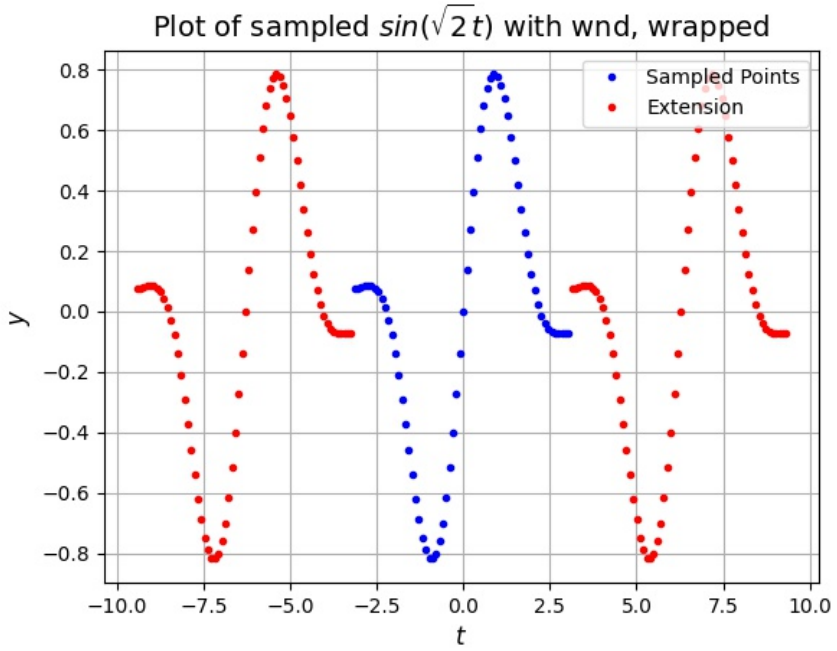

Figure 5: Sampled $sin(\sqrt{2}t)$, wrapped every period with window

Clearly the jump has been reduced and this helps to make the magnitude spectrum much spiky.

The spectrum with window being considered is given in Fig 6. and this can be obtained by passing the wnd_flag as 1.

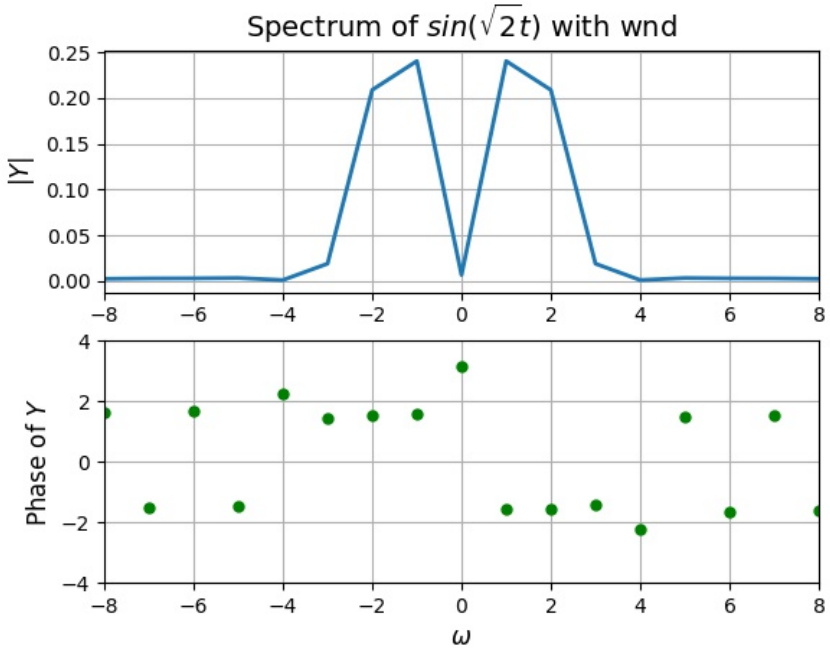

Figure 6: Spectrum of $sin(\sqrt{2}t)$ with hamming window

Clearly from the magnitude spectrum improved but still the spectrum is 2 samples wide, this can be improved further by considering a larger time-window. The plot of the spectrum for $t \in [-4\pi, 4\pi)$ with 256 samples being considered is presented in Fig 7.

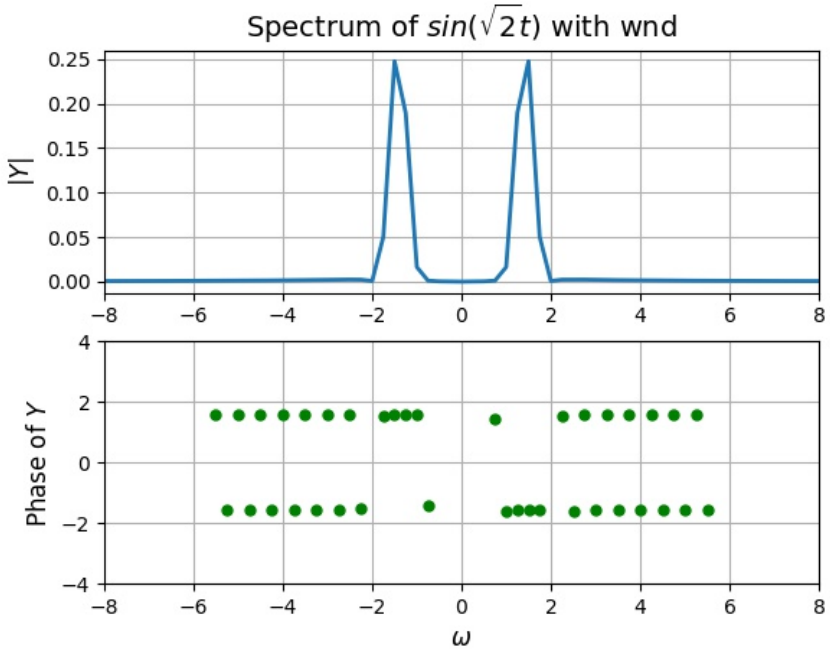

Figure 7: Spectrum of $sin(\sqrt{2}t)$ with hamming window and larger T

The spectrum is much more peaky when compared to that of Fig 6. However this is still not a single peak. This is because 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.

3 Spectrum of $\cos^3(w_0 t)$

Using the approaches mentioned previously, the spectrum of the function $cos^3(w_0 t)$ is obtained for $w_0 = 0.86$. The following lines of code are used to generate the spectrum for this function. The previously mentioned function calls have been used for the same. The spectra obtained have been presented in Fig. 8,9 respectively. Further the samples along with the wrapped extension have been presented in Fig. 10,11. Note that the interval $t \in [-4\pi, 4\pi)$ with 256 sample is being considered.

```
f2 = lambda t : np.cos(0.86*t)**3
q2 = '$cos^3(0.86t)$ '

func_plot([-4*np.pi,4*np.pi], 256, f2, q2 + 'without wnd', 0, 0, 'q2.jpg')
func_plot([-4*np.pi,4*np.pi], 256, f2, q2 + 'with wnd', 1, 0, 'q2_wd.jpg')

Y2a,w2a = spectrum_gen([-4*np.pi,4*np.pi], 256, f2, 0, 1, 0)
Y2b,w2b = spectrum_gen([-4*np.pi,4*np.pi], 256, f2, 1, 1, 0)

spectrum_plot(Y2a, w2a, q2 + 'without wnd', [-8,8], 1, 'q2_dft.jpg', 1e-3)
spectrum_plot(Y2b, w2b, q2 + 'with wnd', [-8,8], 1, 'q2_dft_wd.jpg', 1e-3)
```

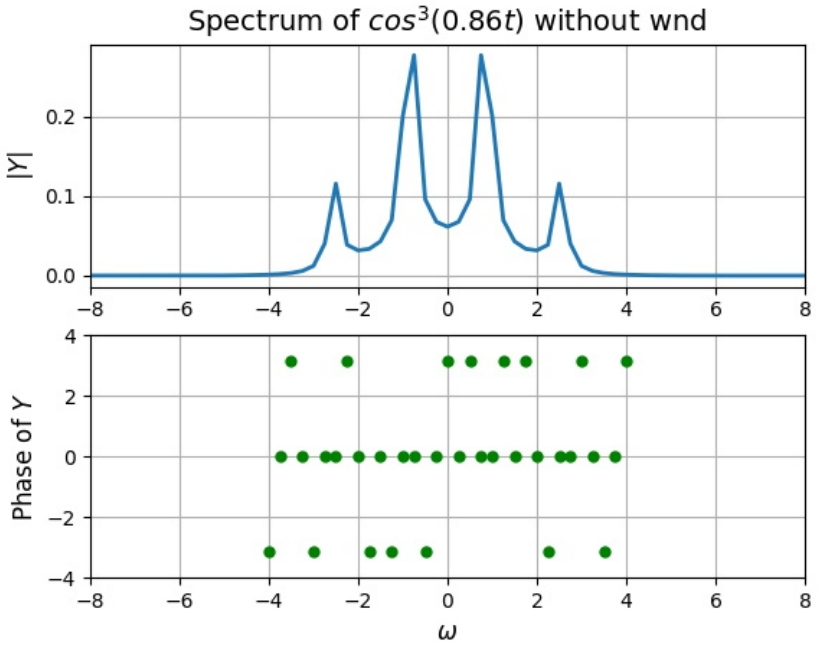

Figure 8: Spectrum of $cos^3(0.86t)$ without hamming window

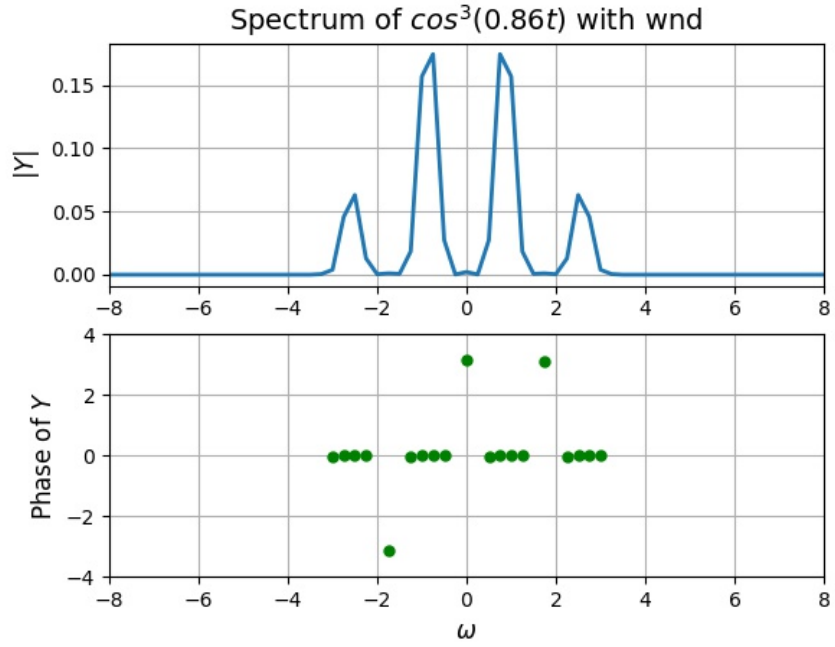

Figure 9: Spectrum of $cos^3(0.86t)$ with hamming window

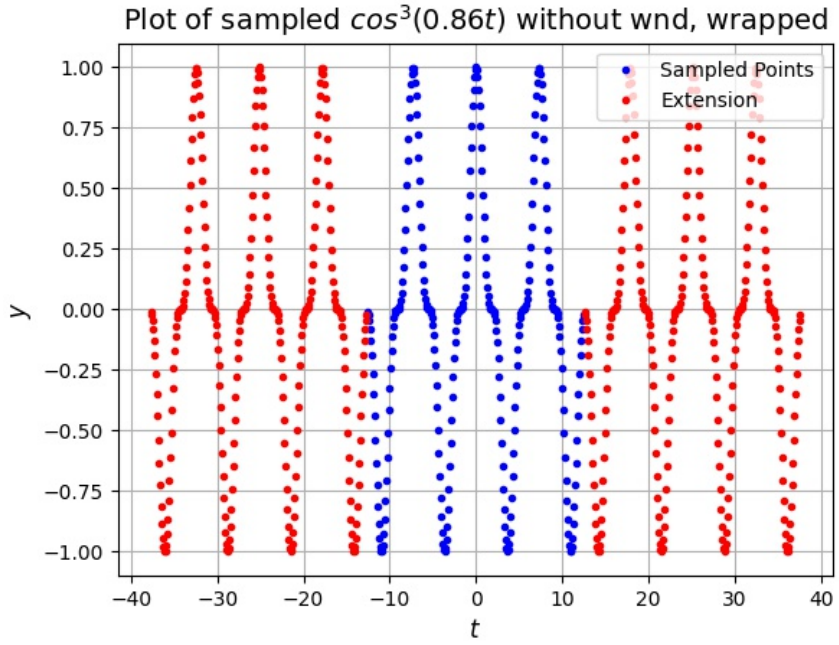

Figure 10: Sampled $cos^3(0.86t)$, wrapped every period without window

Although the first set of plots deviate from the what is expected, leading to the peaks being broader and not dropping to zero, clearly there is an improvement in the spectrum when window is used, where the curve tends to be more peaky. However, they are still broad when compared to the case of using impulse function, because of the reason mentioned previously.

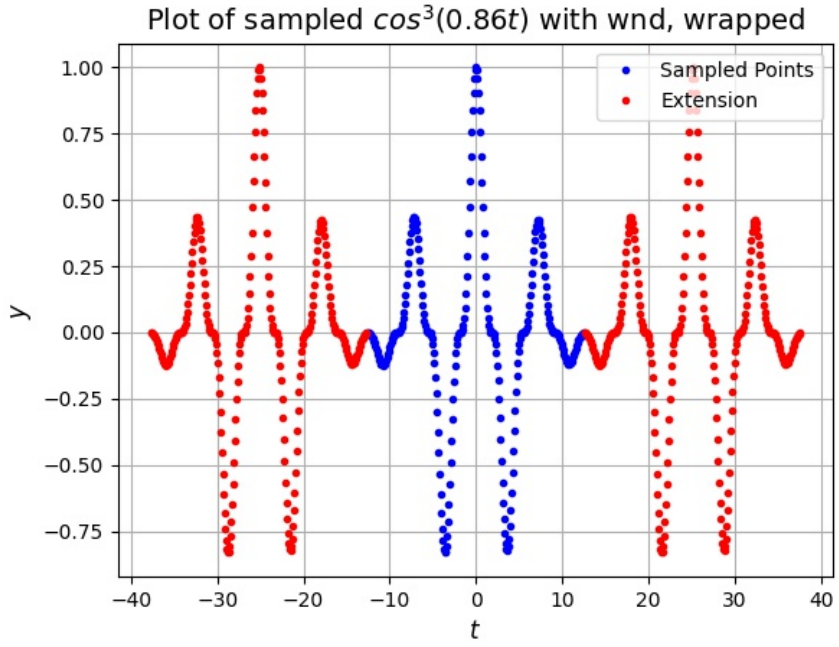

Figure 11: Sampled $cos^3(0.86t)$, wrapped every period with window

4 Spectrum of $\cos(\mathbf{w}_0 t + \delta)$

Given a 128 element vector known to containing $cos(w_0 t + \delta)$ for arbitrary δ and $0.5 < \omega_0 < 1.5$, where t goes from $-\pi$ to π, the task is to estimate ω_0 and δ from the DFT of these samples. This is done using the function below :

```
def wd_estm(w0, d, N, t_range, thresh, noise, dw=4, dd=2, p=1):

    """
    This function estimates w0 and delta for the case pf sinusoid from the DFT
    spectrum. The input arguments are :
    w0 : The true value of w0 used to generate the function points.
    d  : The true value of d  used to generate the function points.
    N          : Number of points for which the function is calculated.
    t_range    : [low_t,up_t] where low_t is the lower limit and up_t is the
                 upper limit of t.
    thresh     : Used in estimation of delta, where only points greater than
                 thresh are used.
    dw         : Radius of the Y points from 0, used in estimation w0.
    dd         : Radius of the Y points from 1, used in estimation d.
    p          : Weight of the norm used in estimation of w0.
    noise      : Boolean value, 1 is passes to add Gaussian noise of A = 0.1
                 is to be added, orwise 0 is passed.
    """
    f = lambda t : np.cos(w0*t + d)
    Y1,w1 = spectrum_gen(t_range, N, f, 1, 1, noise)
    ii  = np.where(w1>=0)
    Y,w = Y1[ii],w1[ii]
    w_estm = np.sum(abs(Y[:dw]**p)*w[:dw])/np.sum(abs(Y[:dw]**p))
    jj = np.where(abs(Y) > thresh)[0]
    d_estm = np.mean(np.angle(Y[jj[1:dd]]))
    return w_estm, d_estm, Y1, w1
```

The procedure used to estimate ω_0 is take the weighted norm of ω with weight p and considering only dw samples of Y and ω. This is done by line 22 in the above code. Similarly δ is estimated as mean phase of dd-1 samples starting from first value that has mag $>$ thresh after w $= 0$. This is done by line 25 in the above code. The optimal values of dd, dw and p are estimated using the below code, where 500 random values of ω_0 and δ are considered.

```
1  dd_range = np.arange(2,5)
2  dw_range = np.arange(2,10)
3  p_range  = np.arange(1,3,0.1)
4
5  we_dict = {}
6  de_dict = {}
7  noise   = 0
8  for dw in dw_range:
9      for p in p_range:
10         we_dict[dw,p] = 0
11
12 for dd in dd_range:
13     de = []
14     for dw in dw_range:
15       for p in p_range:
16           w_error = []
17           d_error = []
18           for i in range(500):
19               w0 = np.random.rand() + 0.5
20               d  = np.random.rand()*2*np.pi - np.pi
21               w_estm, d_estm,_,_ = wd_estm(w0, d, 128, [-np.pi,np.pi], 1e-5,
     noise, dw, dd, p)
22               w_error.append(abs(w_estm - w0))
                                 d_error.append(abs(d_estm - d))
23           we = np.mean(w_error)
24           de.append(np.mean(d_error))
25           we_dict[dw,p] += we
26     de_dict[dd] = np.mean(de)
27
28 dw,p  = min(we_dict, key=we_dict.get)
29 dd    = min(de_dict, key=de_dict.get)
30
31 print('The values of dw and giving minimum deviation is {},{}'.format(dw,p))
32 print('The value of dd which give minimum deviation is {}'.format(dd))
```

Running these lines of code for both the cases with and without noise returned dd to be 2, dw as 4 and p as 1. So the values of ω_0 and δ are estimated with these parameters being the default set and can be changed by the user.

4.1 Without noise

With the functions defined previously, the spectrum of $cos(w_0t+\delta)$ for $\omega_0 = 0.8$ and $\delta = 1$, has been presented in Fig. 12. Note that hamming window is being considered. Clearly there is an overlap between the two peaks. However, the previous mentioned approach of estimating is proved to be less biased, even in this case. This is because the weighted norm is being carried out across 4 samples, so although the peaks are diffuse the norm tends to be converging, unlike the case of estimation using considering only the frequency of peak values, which suffers much when ω_0 is close to 0.5 as it results in a peak at origin. The error in the values for this case is given by :

1. Error in estimate of ω_0 is 0.010.
2. Error in estimate of δ is 0.016.

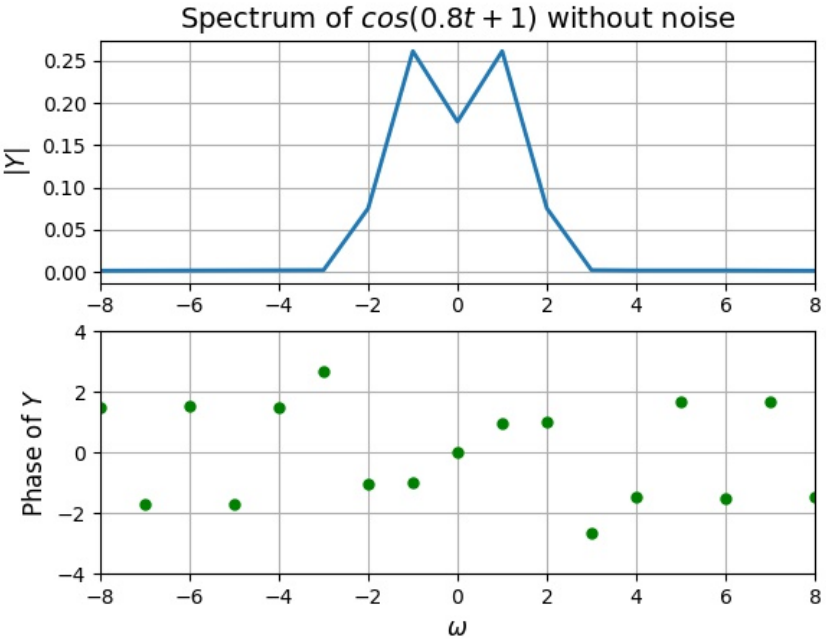

Figure 12: Spectrum of $cos(0.8t + 1)$ without noise

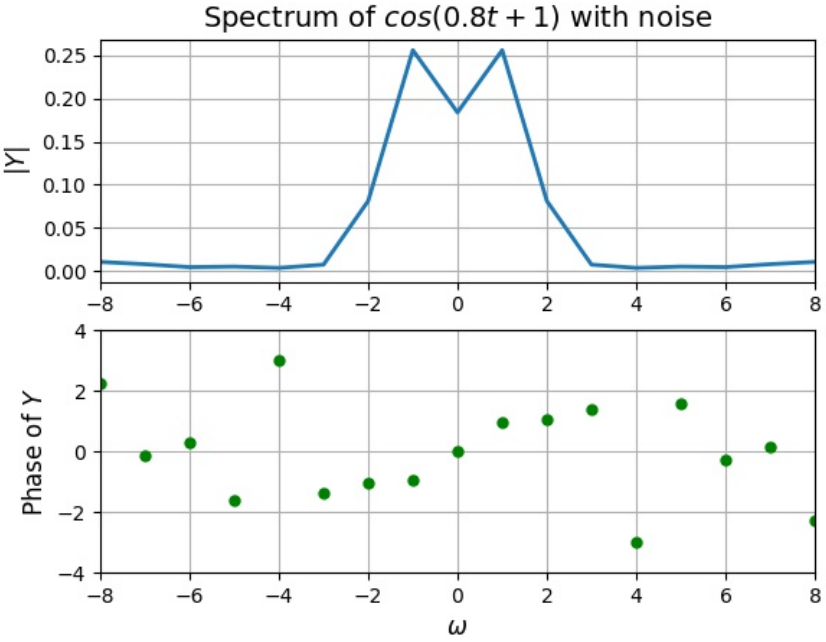

Figure 13: Spectrum of $cos(0.8t + 1)$ with noise

4.2 With noise

With the $\omega_0 = 0.8$ and $\delta = 1$, the spectrum with Gaussian noise with amplitude 0.1 being added (is done by setting noise to 1) has been presented in Fig. 13. Note that hamming window is being considered. Clearly lower frequencies remain unaffected in the spectrum, while those from $\omega > 3$ tend to have a increase in the magnitude. So although noise is added since only first 4 points are being considered to calculate the norm, the value of dw almost remains the same. The error in the values for this case is given by :

1. Error in estimate of ω_0 is 0.015.
2. Error in estimate of δ is 0.035.

Clearly the errors do not vary much even with noise being considered, and the reason for the same is the nature of Fourier spectrum of the function with noise being considered.

5 Spectrum of Chirped Signal

5.1 Using a single window

Using the functions defined previously the spectrum of the chirped signal which is given by :

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

are presented in Fig 14,15 which correspond to the case with and without window respectively. Note that $t \in [-\pi, \pi)$ has been used with 1024 samples being considered. The frequency of the signal continuously changes from 16 to 32 rad/s.

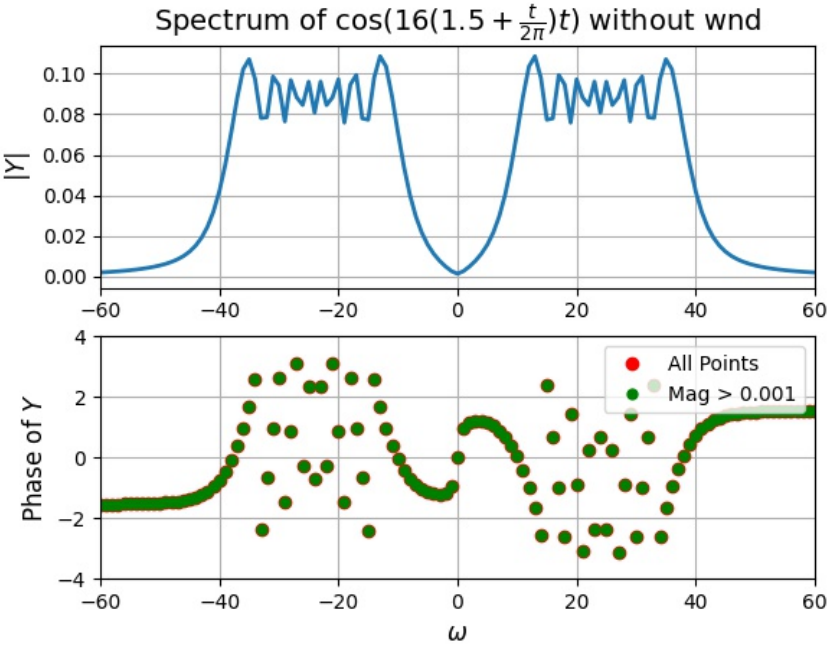

Figure 14: Spectrum of $\cos\big(16(1.5 + \frac{t}{2\pi})t\big)$ without window

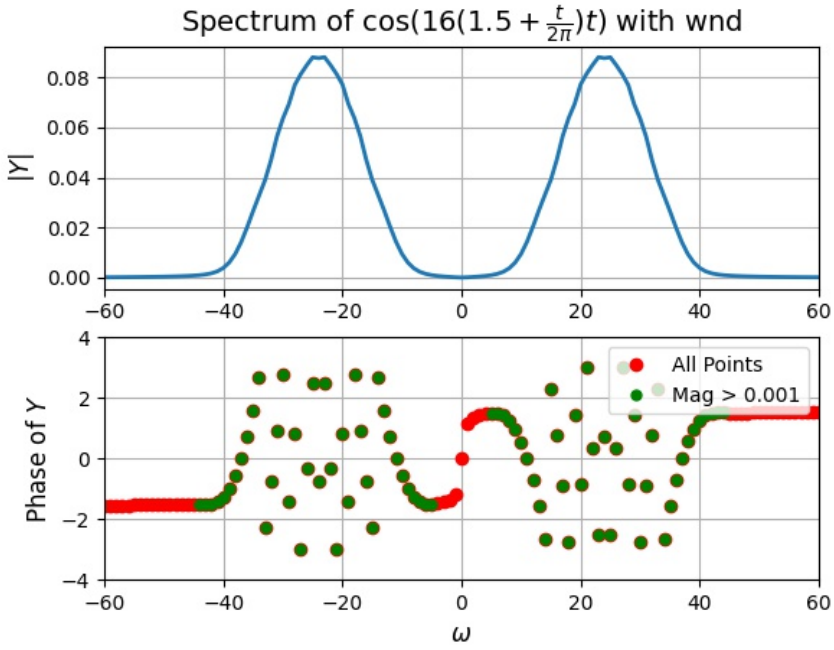

Figure 15: Spectrum of $\cos\left(16(1.5+\frac{t}{2\pi})t\right)$ with window

From the plots it can be inferred that using the window has reduced the spread of peaks, where it was from around 4-50 rad/s without using the window and it reduced to 16-32 rad/s after using it.

5.2 Using multiple slices

To gain further insight about the chirped signal, the spectrum is obtained by breaking the 1024 samples of time into 16 sub-samples of length 64 each and DFT of each sample is stored as rows of a 2D vector. The following lines of code has been used to obtain the 2D array for both the cases with and without using the window.

```
t6 = np.linspace(-np.pi,np.pi, 1025)
Y6a = np.zeros((16,64),dtype = 'complex_')
Y6b = np.zeros((16,64),dtype = 'complex_')

for i in range(16):
  Yia,w6 = spectrum_gen(t6[64*i:i*64 + 65], 64, f56, 1, 0, 0)
  Yib,w6 = spectrum_gen(t6[64*i:i*64 + 65], 64, f56, 0, 0, 0)
  Y6a[i][:] = Yia
  Y6b[i][:] = Yib

t6 = t6[:-1]
```

The 2D-vector is plotted as a surface plot and contour plot as well for better readability, note these plots are "time-frequency" plots, where we get localized DFTs which show how the spectrum evolves in time. The plots obtained are presented below in Fig. 16-23, where the phase of all the points has been considered even those with magnitude very less. The magnitude plot has been limited to $w \in [-120, 120]$ for the sake of better readability, and it has been verified that the magnitude after this interval is very low. From the time-evolution of magnitude plot it can be seen that, initially the frequencies are surrounded around 16, and as

we move with time they are surrounded around 32, and the same can be asserted by evaluating the function at t = -π and t = π. The transition is more smoother in case of using a window when compared to the case of not using a window and it is further linear in the former case. These are in-accordance to the spectra obtained previously. Further to add the phase spectrum is comparatively broader in the case of using a window.

6 Conclusion

Thus, spectra of non-periodic signals can be obtained using numpy.fft() with several other attributes like windowing added. The need of windowing has been analysed in in order to surpass the Gibbs Phenomena due to discontinuities in the extension of sampled signal. However, this impulse is being replaced by spectraum of the window, sharp peaks have not been obtained even in case of sinusoids. The spectra obtained of a phase-shifted sinusoid produced good-enough estimate of the frequency and phase of the sampled signal, and it has been observed that noise effects mainly higher frequency components. The chirped signal has been analysed through the time-evolution of the spectra where slices have been considered. The frequency components go from 16 to 32, as we move with time. The Fourier spectra provided various useful insights in all of these problems.

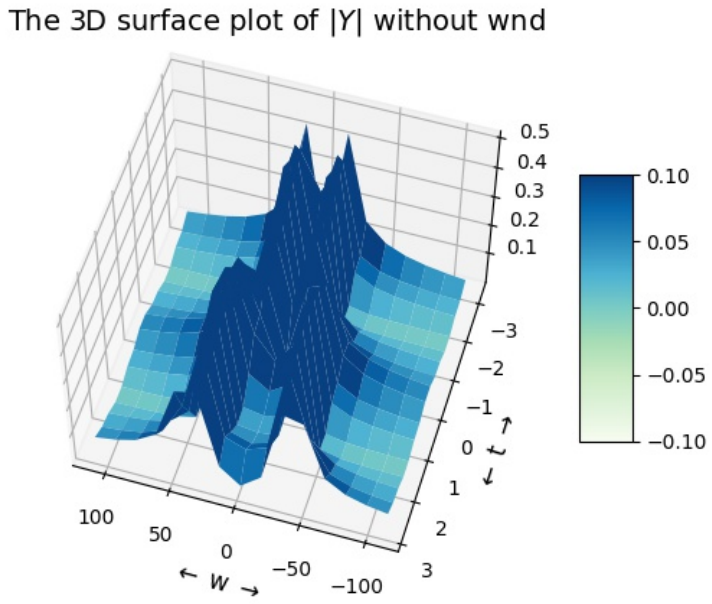

Figure 16: Surface Plot of $|Y|$, no window

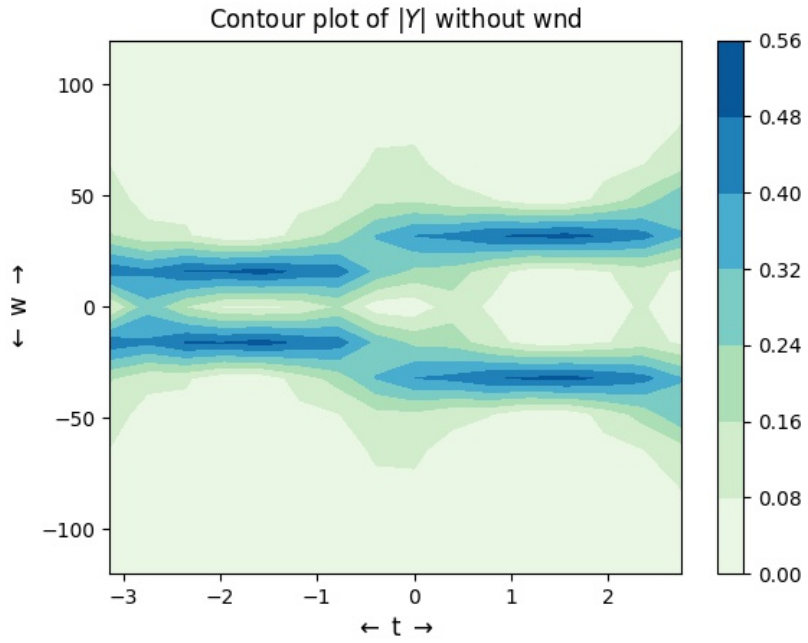

Figure 17: Contour Plot of $|Y|$, no window

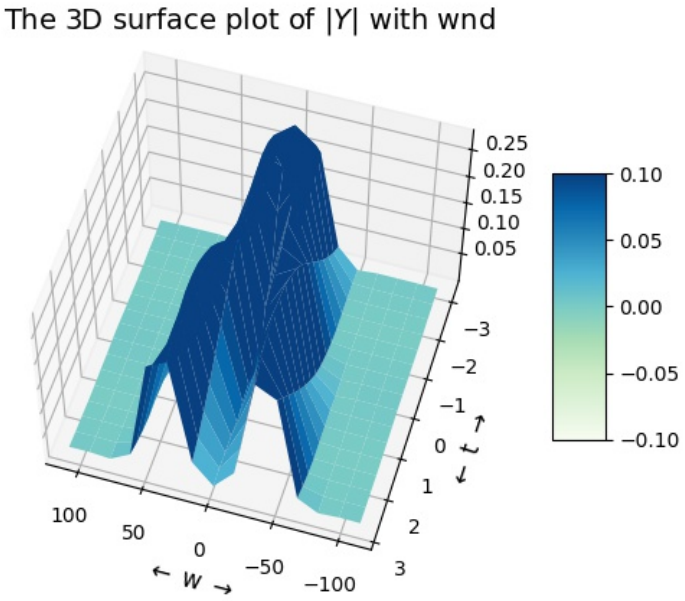

Figure 18: Surface Plot of $|Y|$, with window

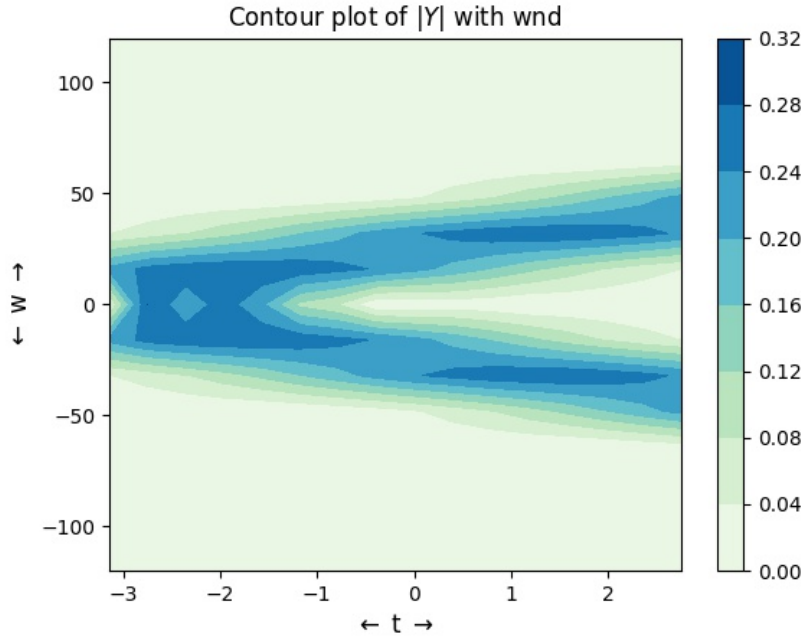

Figure 19: Contour Plot of $|Y|$, with window

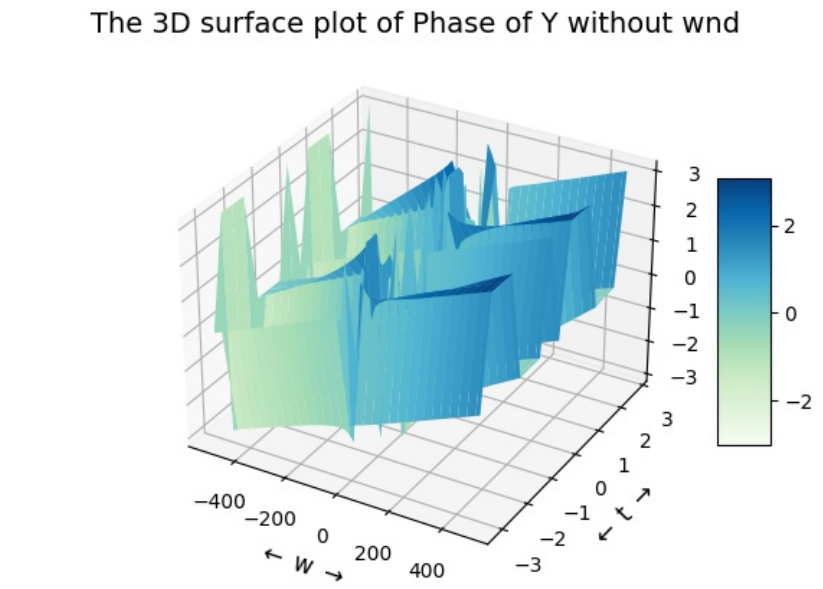

Figure 20: Surface Plot of Phase, no window

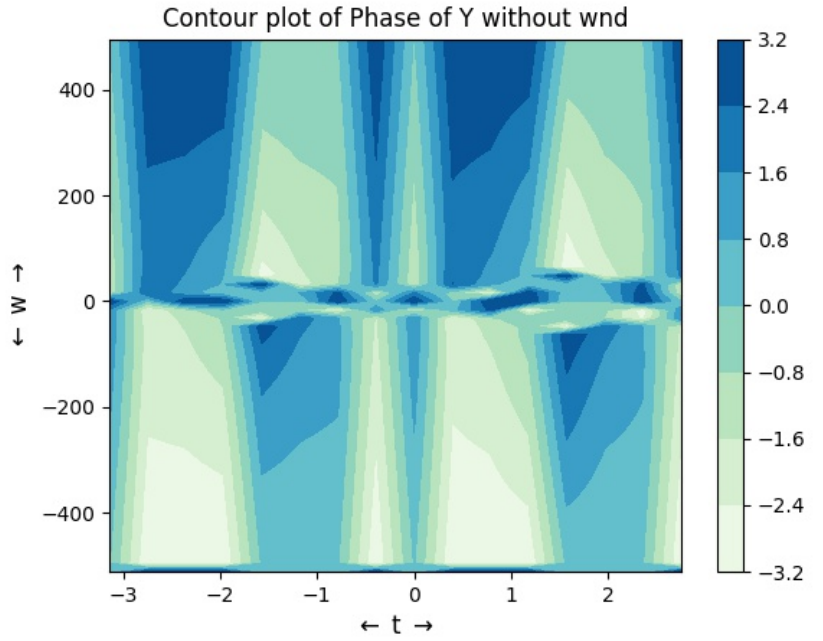

Figure 21: Contour Plot of Phase, no window

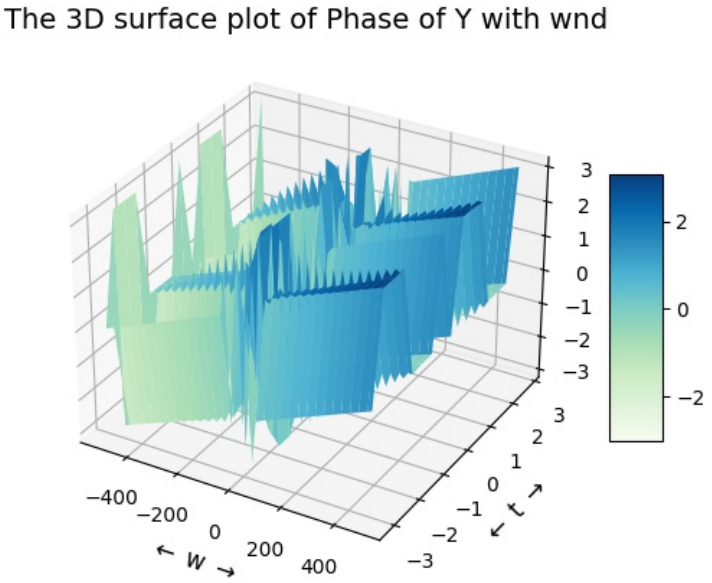

Figure 22: Surface Plot of Phase, with window

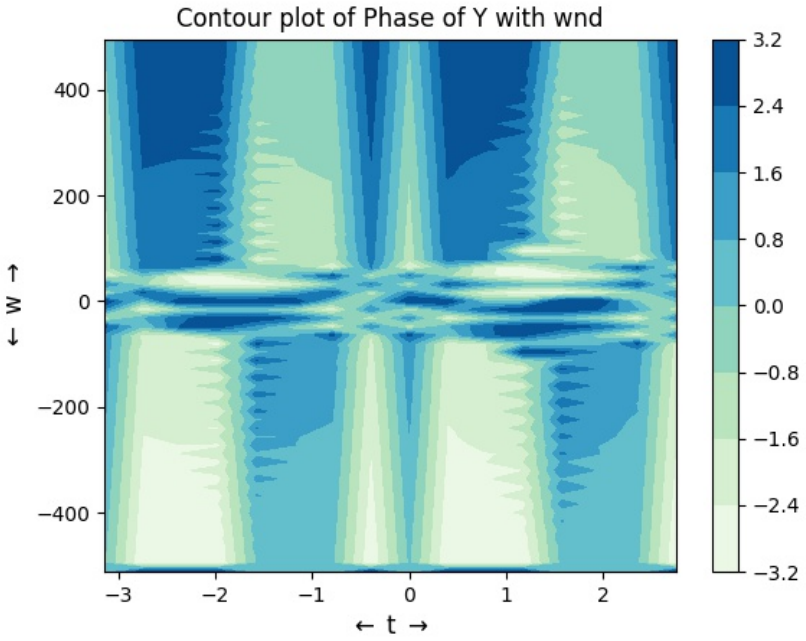

Figure 23: Contour Plot of Phase, with window