EE 338 Filter Design Assignment

Filter Number 82

Meet Pragnesh Shah | 13D070003
meetshah1995@ee.iitb.ac.in

1. Table of contents

- Design Process
  - Bandpass Filter 1.2
    - IIR Bandpass Filter 1.2.1
    - FIR Bandpass Filter 1.2.2
  - Bandstop Filter 1.3
    - IIR Bandstop Filter 1.3.1
    - FIR Bandstop Filter 1.3.2
- References 2
- Code
  - Bandpass Filter 3.1
    - IIR Bandpass Filter 3.1.1
    - FIR Bandpass Filter 3.1.2
  - Bandstop Filter 3.2
    - IIR Bandstop Filter 3.2.1
    - FIR Bandstop Filter 3.2.2

1.1. Filter Specifications

- m = 7
- q = 0
- r = 7

1.2. Filter 1

- Filter type: Band Pass.
- Passband tolerance = 0.15 (in magnitude).
- Stopband tolerance = 0.15 (in magnitude).
- Transition band = 2 KHz on either side of band.
- Pass band type = equiripple.
- Stop band type = monotonic.
- Sampling frequency = 100 kHz.
- Signal Bandlimit = 45 kHz
- Passband low limit, B_l = 18 kHz.
- Passband high limit, B_h = 28 kHz.

1.2.1. IIR Bandpass Filter

We obtain the normalized frequcnies using the following formulae:

$freq\_norm = \frac{freq*2*pi}{freq\_sampling}$

- Normalized Passband limits : $1.13097336, 1.75929189$
- Normalized Transitioned Passband limits : $1.00530965, 1.88495559$

We use the following transformation to get the analog filter specifications corresponding to the above digital filter.

- $\Omega = tan(\omega/2)$
  - Passband limits : $0.6346193, 1.20879235$
  - Stopband limits : $0.54975465, 1.37638192$

Since the passband is equiripple and stopband is monotonic we use the **Chebyschev** filter.

For Low Pass Filter

- $\Omega_p = 1$
- $\Omega_s = min(\frac{\Omega_{S1}^2-\Omega_0^2}{B*\Omega_{S1}}, \frac{\Omega_{S2}^2-\Omega_0^2}{B*\Omega_{S2}})$
- $\Omega_0^2 = \Omega_P1 * \Omega_P2 = 0.767122$
- $B = \Omega_P2 - \Omega_P1 = 0.574173$

The Transform used for transformation of analog low pass filter to analog band pass filter is:

- $\Omega_L = \frac{\Omega^2-\Omega_0^2}{B\Omega}$

The values for D1 and D2 hence obtained are :

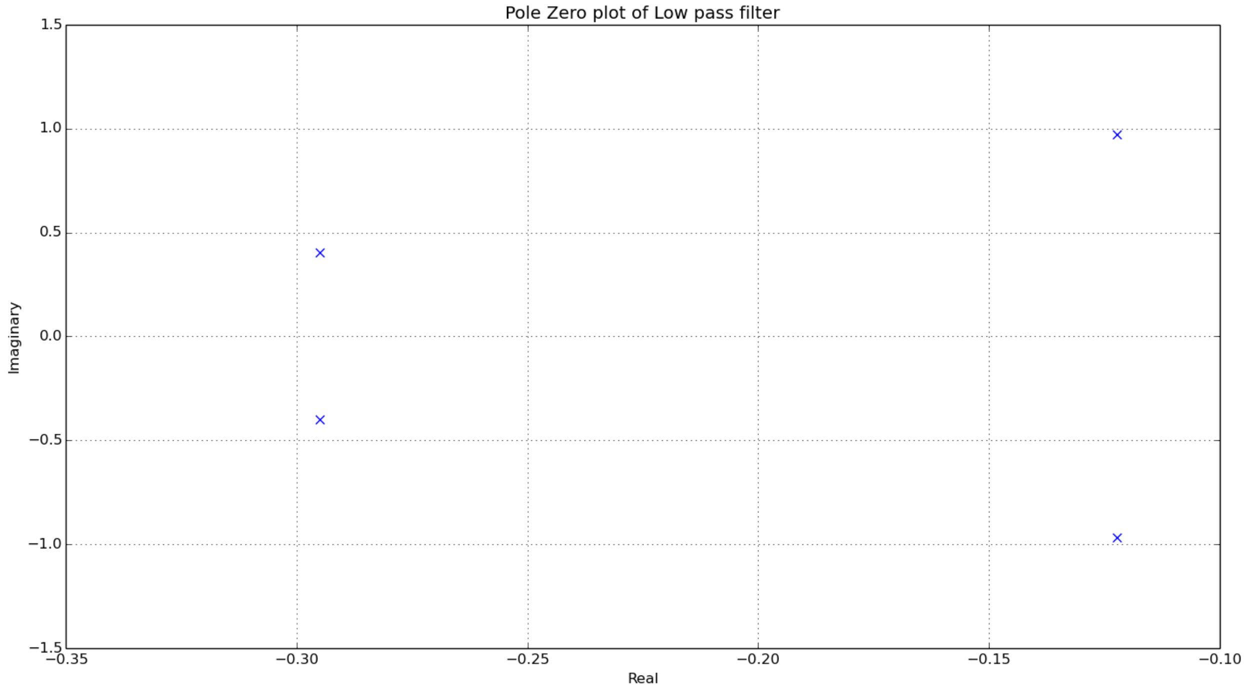
- D1 = 0.3840835
- D2 = 43.444444

Now the order is obtained using :

- N = $ceil(\frac{acosh(\sqrt{\frac{D2}{D1}})}{acosh(\frac{\Omega_{L}s}{\Omega_{L}p})}$
- N = 4

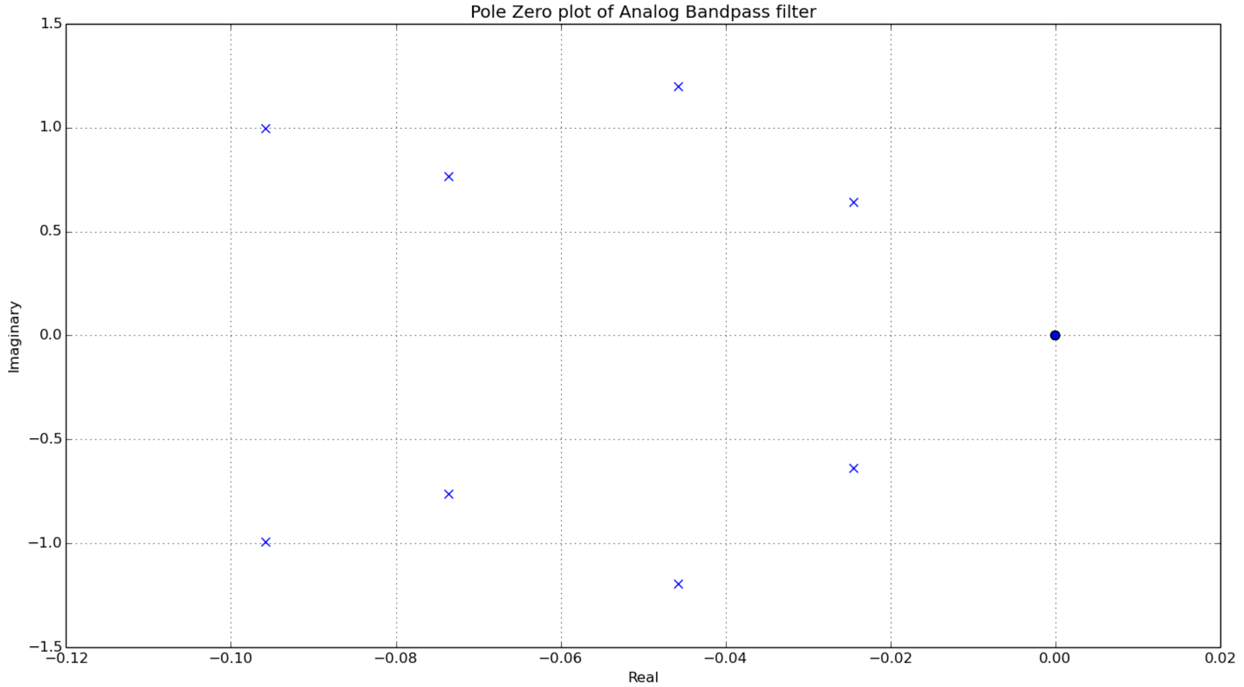
The lowpass filter transfer function is as follows :

- $H_{analog\_lowpass} = \frac{0.2372}{s^4+(0.8342s^3+1.348s^2+0.6243s+0.2373}$



The analog bandpass filter transfer function is as follows :

- $H_{analog\_bandpass} = \frac{0.02579s^4}{s^8+0.479s^7+3.513s^6+1.22s^5+4.238s^4+(0.9362s^3x+(2.067s^2+0.2162s+0.3463}$

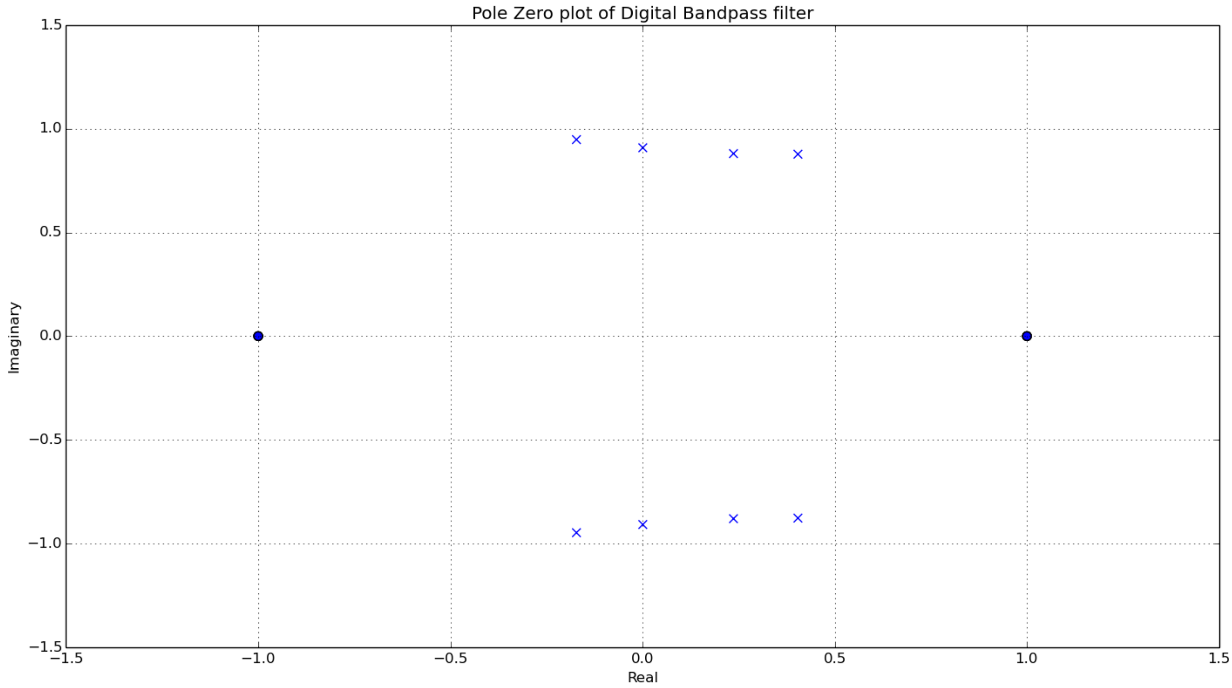


The transformation is used get digital filter from analog filter :

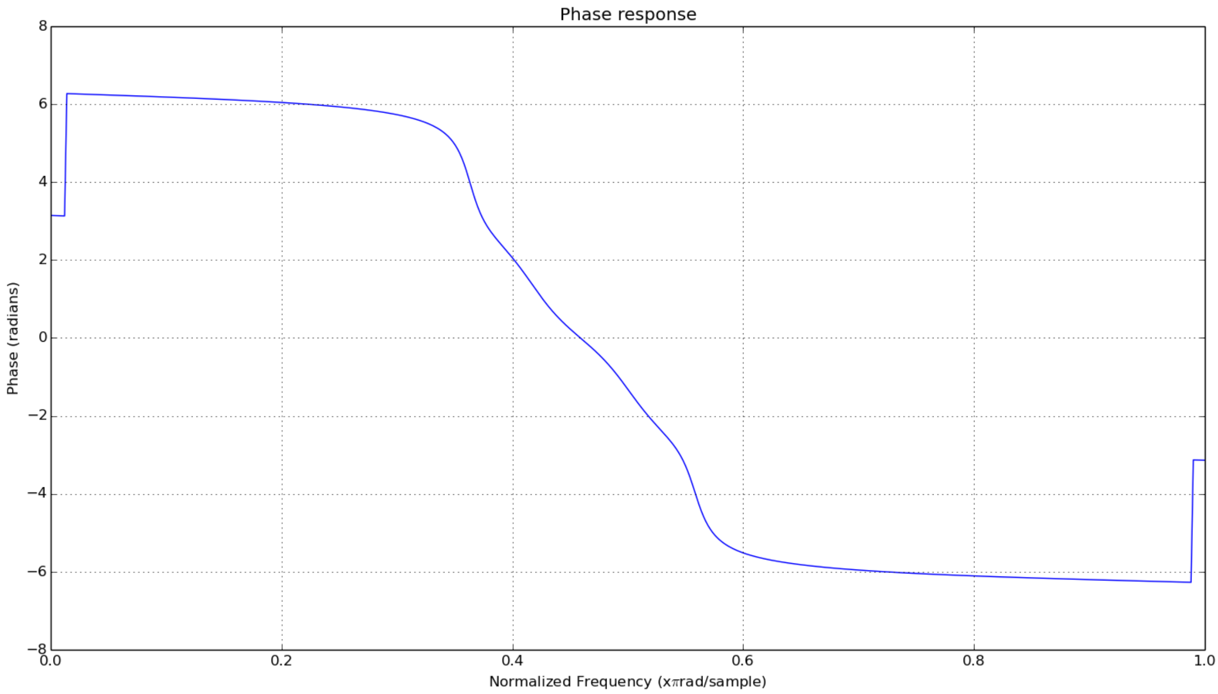
- $S = \frac{1-Z^{-1}}{1+Z^{-1}}$

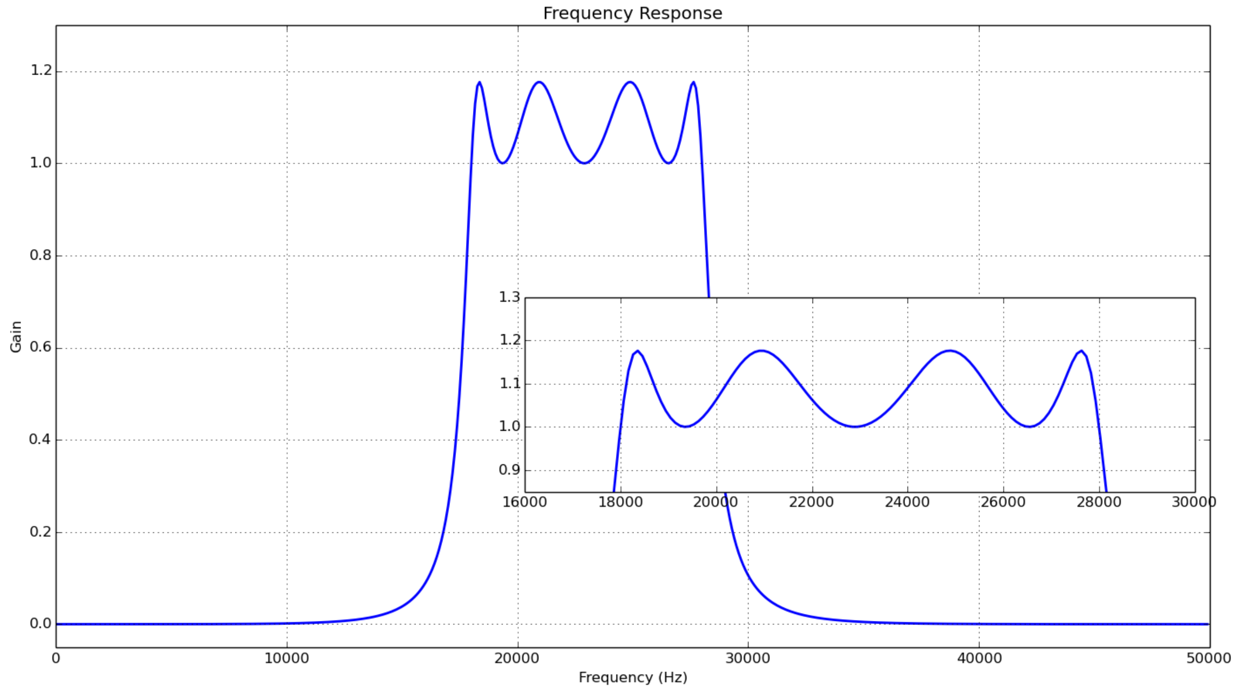
- $H_{digital\_bandpass}(Z) =$

$$\frac{0.0018 - 0.0073Z^{-2} + 0.0110Z^{-4} - 0.0073Z^{-6} + 0.0018Z^{-8}}{1 + -0.9386Z^{-1} + 3.4589Z^{-2} + -2.3398Z^{-3} + 4.5567Z^{-4} + -2.0583Z^{-5} + 2.6856Z^{-6} + -0.6326Z^{-7} + 0.5930Z^{-8}}$$



Phase and Frequency Response Plots of the filter





1.2.2. FIR Bandpass Filter

Order Calculation

- $(2 * N + 1) > 1 + ((A - 8)/2.285 * \Delta_\omega)$
- $\Delta_\omega = \omega_s - \omega_p$
- $A = -20 * log10(\delta_{tolerance})$

The ideal impulse response of the bandpass filter and cutoff frequencies is obtained from the normalized frequencies using :

- $H_{FIR}(n) = \frac{\omega_{c2} - \omega_{c1}}{\pi}, n = 0$
- $H_{FIR}(n) = \frac{sin(n\omega_{c2}) - sin(n\omega_{c1})}{n\pi} n! = 0$
- $n = [-N, N]$
- $\omega_{c1} = \frac{\omega_{s1} + \omega_{p1}}{2}$
- $\omega_{c2} = \frac{\omega_{s2} + \omega_{p2}}{2}$

We then multiply the ideal impulse response with a **Kaiser window** to get the FIR impulse response:

$H(z) =$

$0.0237 - 0.0072Z^{-1} - 0.0480Z^{-2} - 0.0033Z^{-3} + 0.0480Z^{-4}$

$+0.0116Z^{-5} - 0.0159Z^{-6} + 0.0053Z^{-7} - 0.0338Z^{-8} - 0.0596Z^{-9} + 0.0712Z^{-10}$

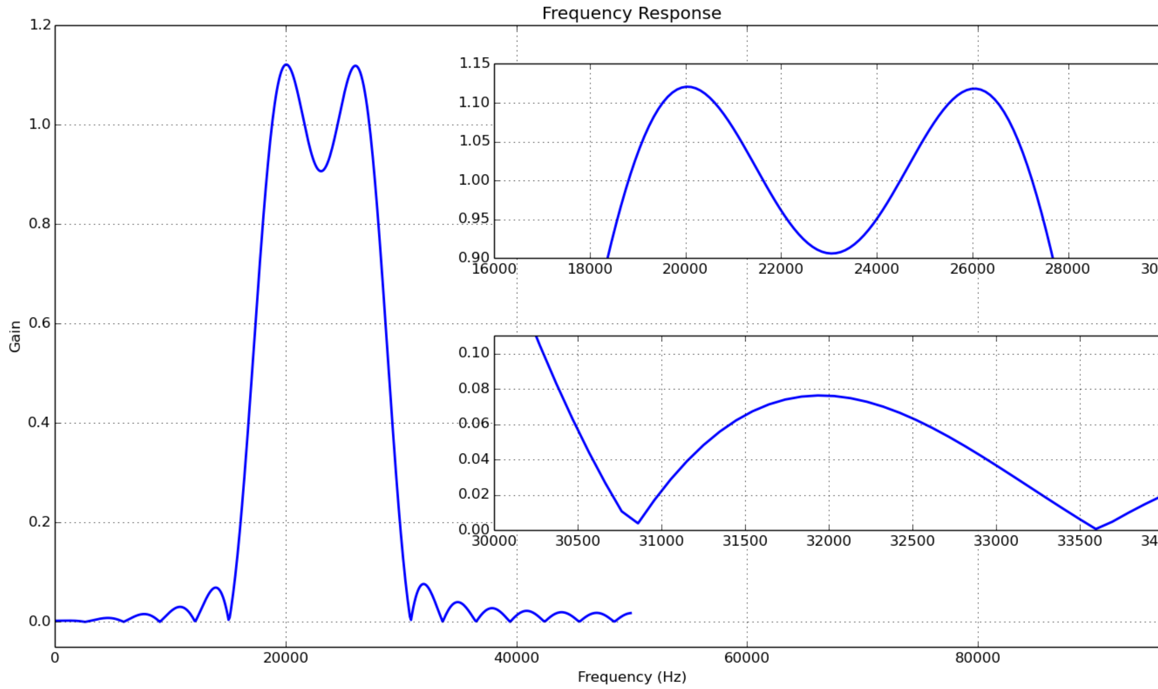
$+0.1392Z^{-11} - 0.0707Z^{-12} - 0.2111Z^{-13} + 0.0294Z^{-14} + 0.2400Z^{-15}$

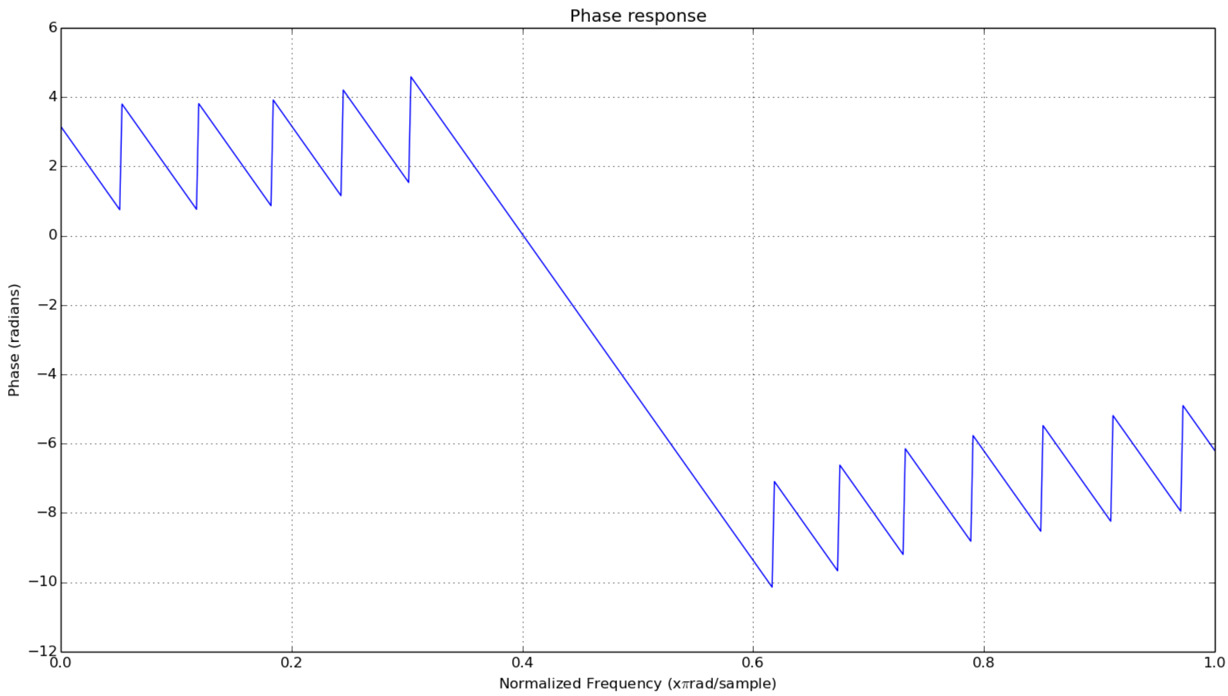
$+0.0294Z^{-16} - 0.2111Z^{-17} - 0.0707Z^{-18} + 0.1392Z^{-19} + 0.0712Z^{-20}$

$-0.0596Z^{-21} - 0.0338Z^{-22} + 0.0053Z^{-23} - 0.0159Z^{-24} + 0.0116Z^{-25}$

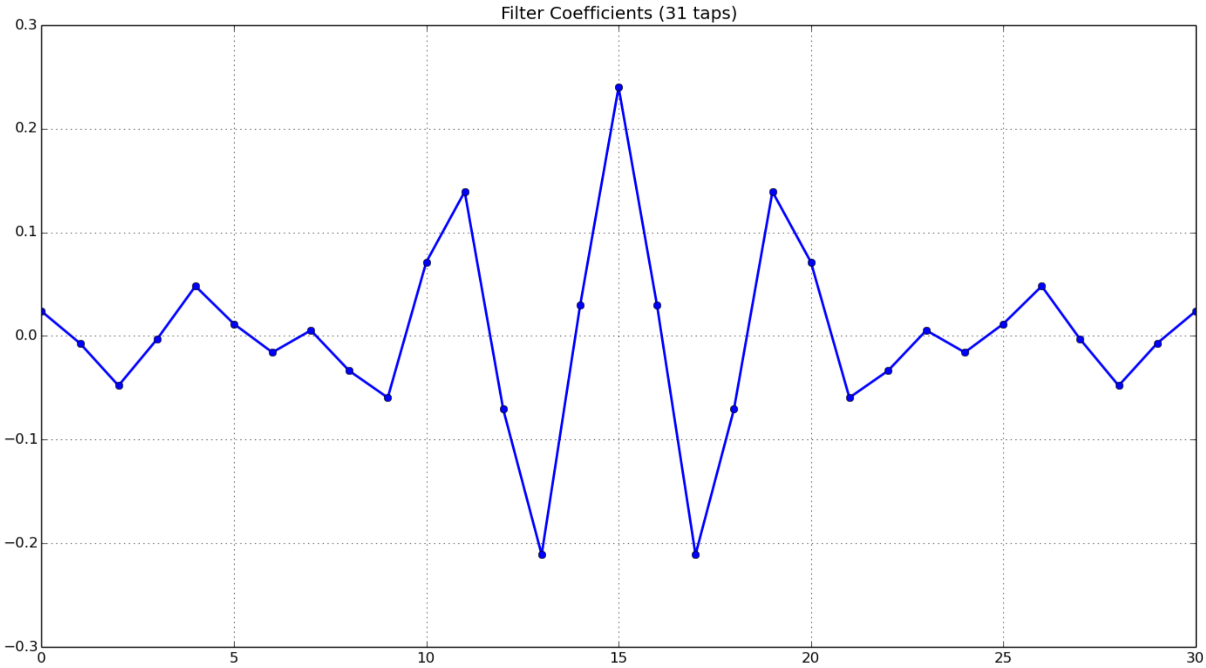
$+0.0480Z^{-26} - 0.0033Z^{-27} - 0.0480Z^{-28} - 0.0072Z^{-29} + 0.0237Z^{-30}$

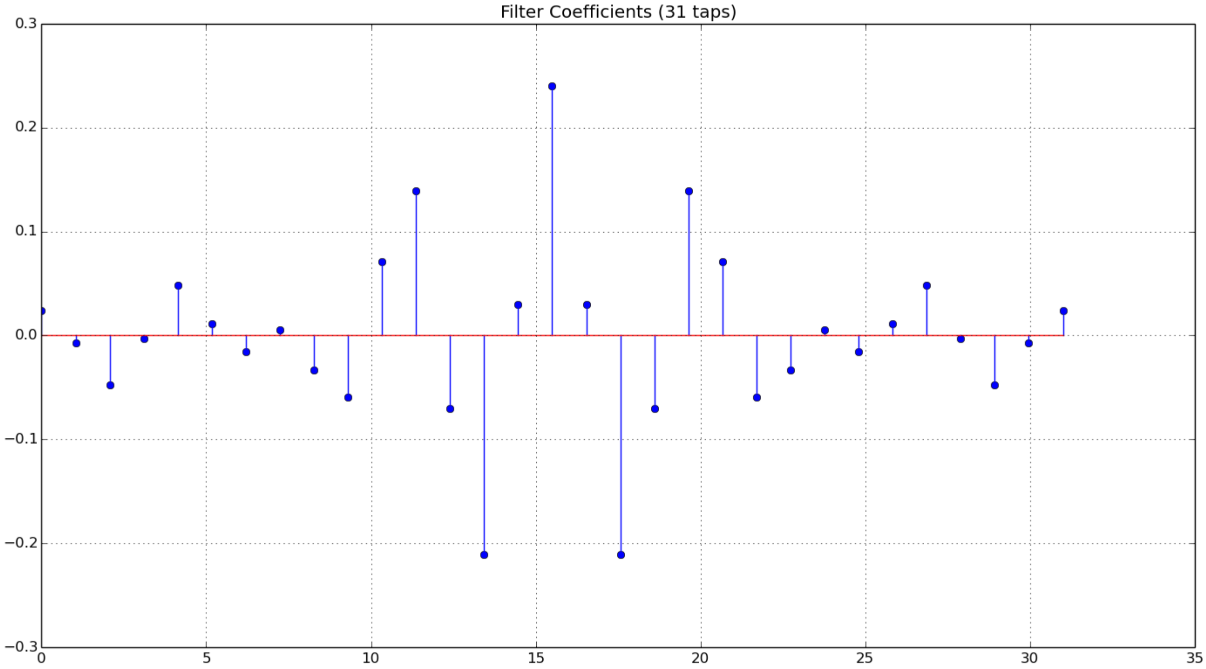
Phase and Frequency Response Plots of the filter





Filter coefficients Plots of the filter





1.3. Filter 2

- Filter type: Band Stop.
- Passband tolerance = 0.15 (in magnitude).
- Stopband tolerance = 0.15 (in magnitude).
- Transition band = 2 KHz on either side of band.
- Pass band type = monotonic.
- Stop band type = monotonic.
- Sampling frequency = 100 kHz.
- Signal Bandlimit = 45 kHz.
- Passband high limit, B_h = 18 kHz.
- Passband low limit, B_l = 28 kHz.

1.3.1. IIR Bandstop Filter

We obtain the normalized frequencies using the following formulae:

$freq\_norm = \frac{freq*2*pi}{freq\_sampling}$

- Normalized Stopband limits : $1.13097336, 1.75929189$
- Normalized Transitioned Stopband limits : $1.00530965, 1.88495559$

We use the following transformation to get the analog filter specifications corresponding to the above digital filter.

- $\Omega = tan(\omega/2)$
    - Passband limits : $0.6346193, 1.20879235$
    - Stopband limits : $0.54975465, 1.37638192$

Since the passband and stopband both are monotonic we use the **ButterWorth** filter.

For Low Pass Filter

- $\Omega_p = 1$
- $\Omega_s = min(\frac{B*\Omega_{S1}}{\Omega_0^2-\Omega_{S1}^2}, \frac{B*\Omega_{S2}}{\Omega_0^2-\Omega_{S2}^2})$
- Ω_0^2 $=\Omega_P1 * \Omega_P2$$= 0.767122$
- $B = \Omega_P2 - \Omega_P1 = 0.574173$

The Transform used for transformation of analog low pass filter to analog band stop filter is:

- $\Omega_L = \frac{B\Omega}{\Omega_0^2-\Omega^2}$

The values for D1 and D2 hence obtained are :

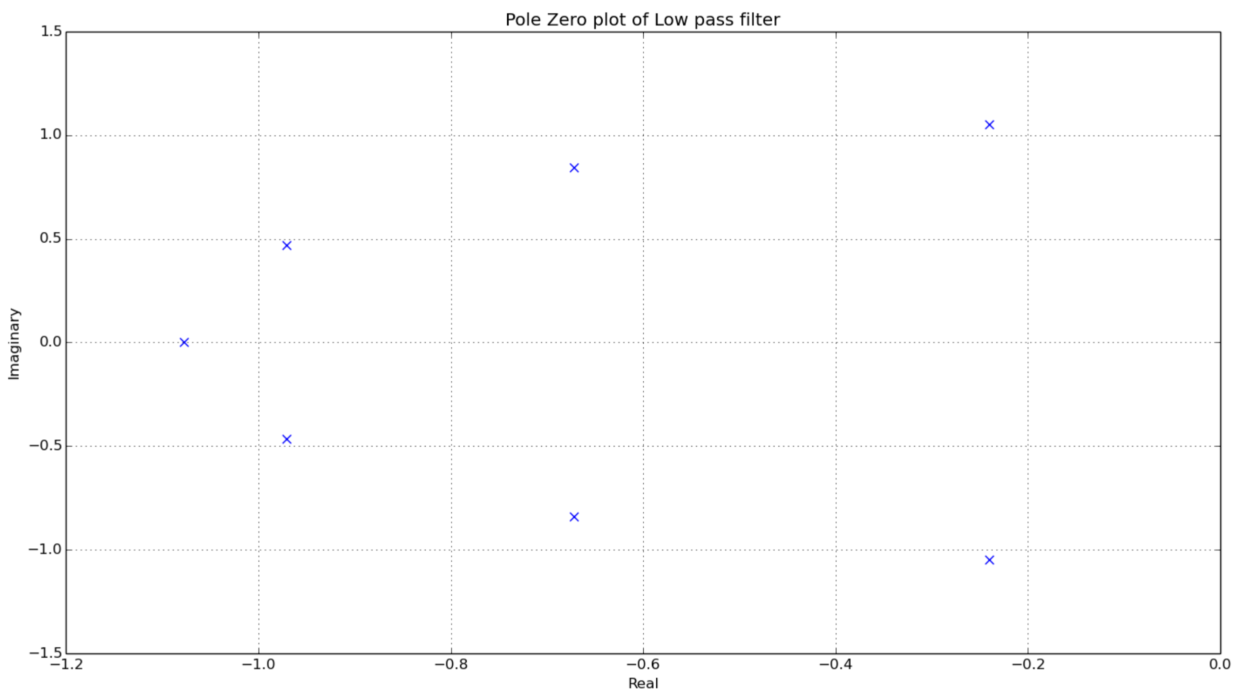
- D1 = 0.3840835
- D2 = 43.444444

Now the order is obtained using :

- N = $ceil(\frac{log(\sqrt{\frac{D2}{D1}})}{(log(\frac{\Omega_Ls}{\Omega_Lp})})$
- N = 7

The lowpass filter transfer function is as follows :

- $H_{analog\_lowpass} = \frac{1.68143395125}{s^7+4.84s^6+11.71s^5+18.23s^4+19.64s^3+14.64s^2+7.016s+1.681}$

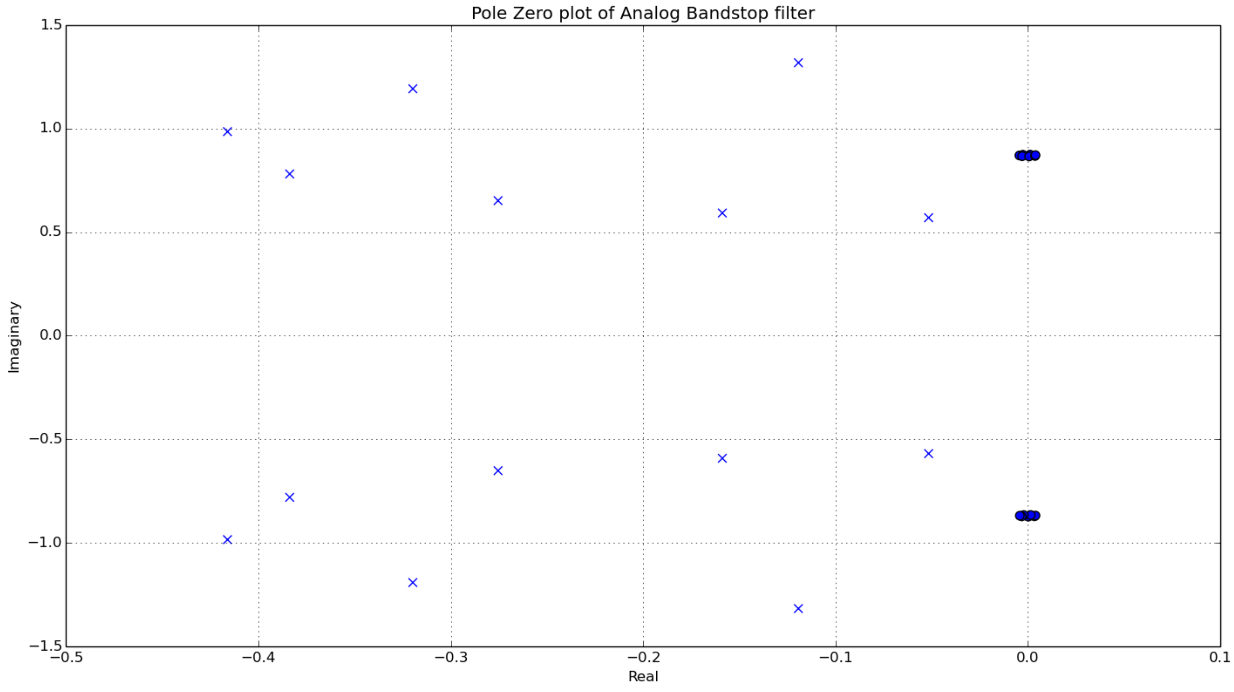


The analog bandpass filter transfer function is as follows :

Note : The coefficients are rounded to 2 numerics to fit here, the actual coefficients are upto 6 decimals

$H_{analog\_bandstop} =$

$\frac{1.681s^{14}+8.906s^{12}+20.22s^{10}+25.5s^8+19.29s^6+8.759s^4+2.209s^2+0.2388}{1.6s^{14}+5.7s^{13}+18s^{12}+37s^{11}+66s^{10}+87s^9+103s^8+95s^7+78s^6+50s^5+28s^4+12s^3+4.6s^2+1.0s+0.2}$

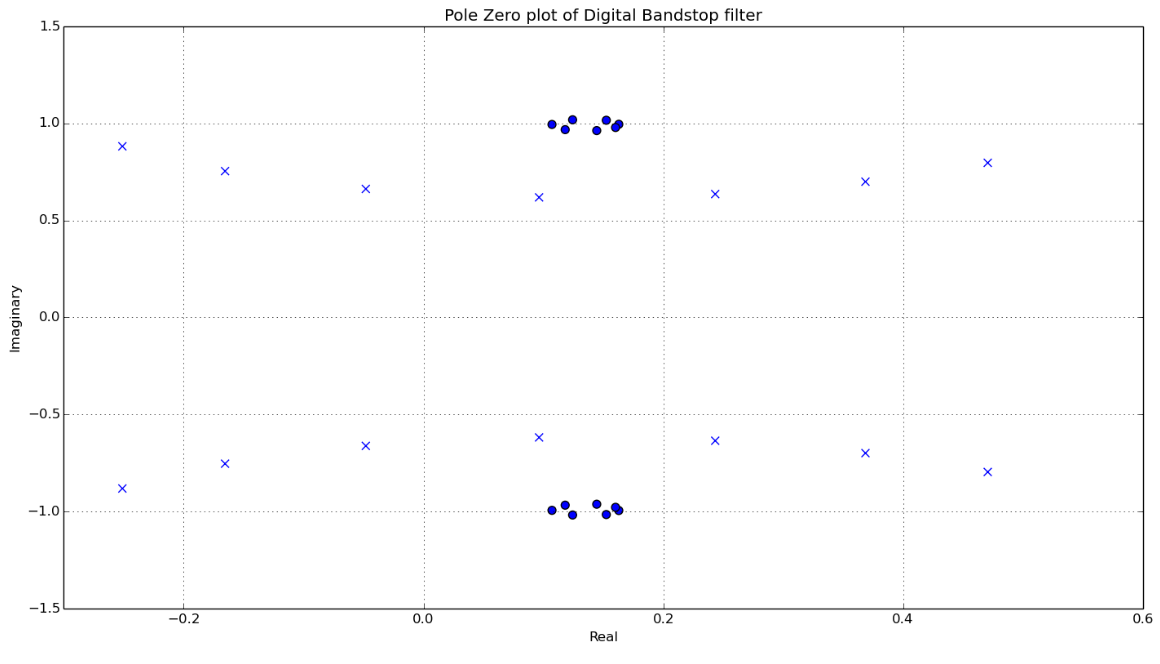


The transformation is used get digital filter from analog filter :

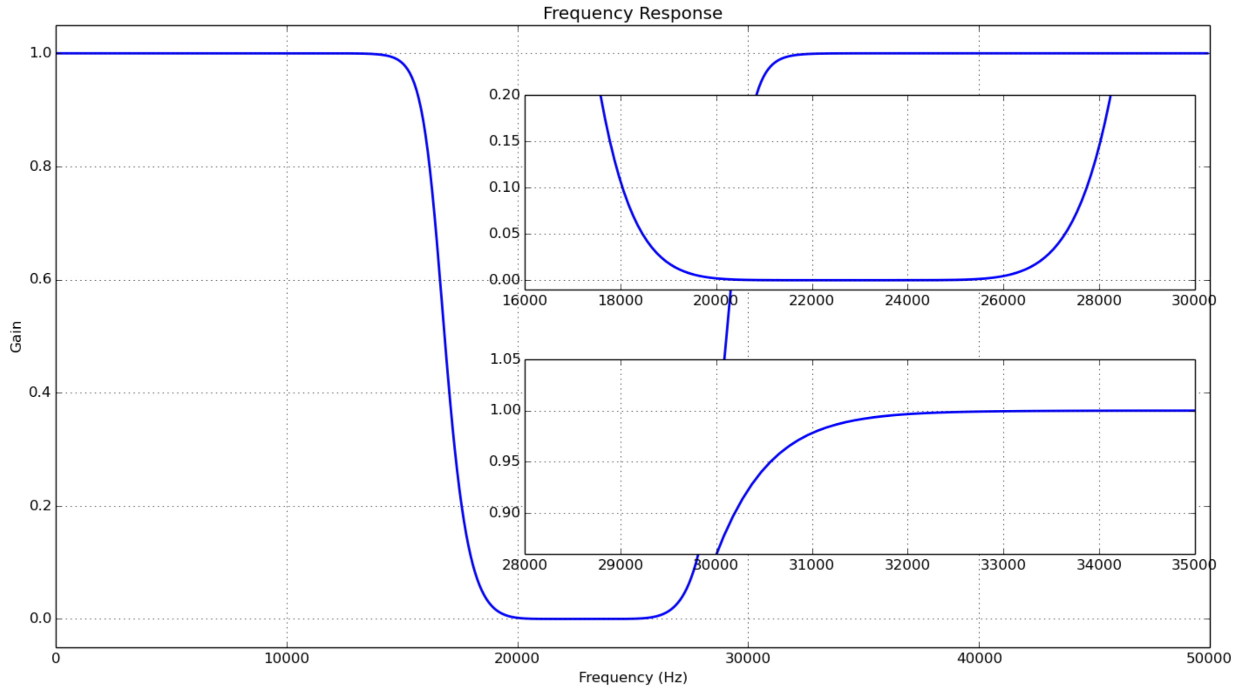
Note : The coefficients are rounded to 2 numerics to fit here, the actual coefficients are upto 6 decimals

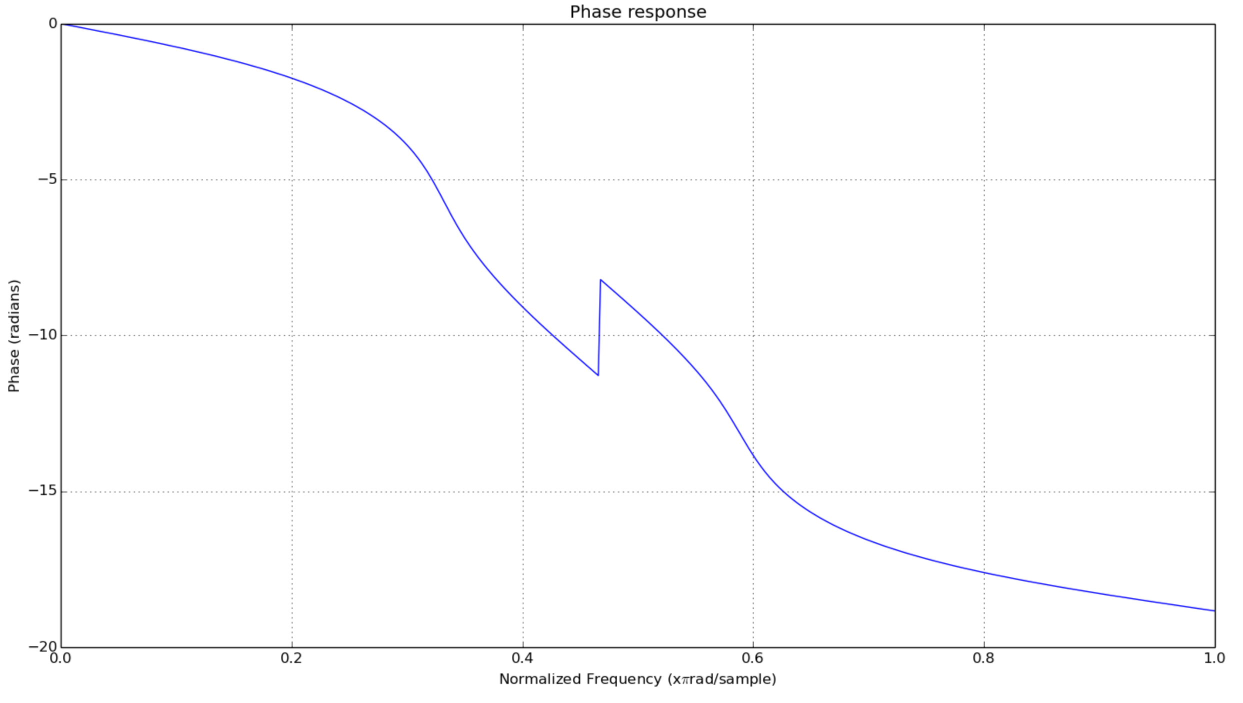
- $S = \frac{1-Z^{-1}}{1+Z^{-1}}$
- $H_{digital\_bandstop}(Z) =$

$$\frac{0.14-0.28Z^{-1}+1.2Z^{-2}-1.8Z^{-3}+4.2Z^{-4}-4.6Z^{-5}+7.5Z^{-6}-6.3Z^{-7}+7.5Z^{-8}-4.6Z^{-9}+4.2Z^{-10}-1.8Z^{-11}+1.2Z^{-12}-0.2Z^{-13}+0.14Z^{-14}}{1-1.43Z^{-1}+4.20Z^{-2}-4.51Z^{-3}+7.55Z^{-4}-6.33Z^{-5}+7.51Z^{-6}-4.90Z^{-7}+4.42Z^{-8}-2.18Z^{-9}+1.53Z^{-10}-0.52Z^{-11}+0.28Z^{-12}-0.05Z^{-13}+0.02Z^{-14}}$$



Phase and Frequency Response Plots of the filter





1.3.2. FIR Bandstop Filter

Order Calculation

- $(2 * N + 1) > 1 + ((A - 8)/2.285 * \Delta_\omega)$
- $\Delta_\omega = \omega_s - \omega_p$
- $A = -20 * log10(\delta_{tolerance})$

The ideal impulse response of the bandpass filter and cutoff frequencies is obtained from the normalized frequencies using :

- $H_{FIR}(n) = \frac{\omega_{c2} - \omega_{c1}}{\pi}, n = 0$
- $H_{FIR}(n) = \frac{sin(n\omega_{c2}) - sin(n\omega_{c1})}{n\pi} n! = 0$
- $n = [-N, N]$
- $\omega_{c1} = \frac{\omega_{s1} + \omega_{p1}}{2}$
- $\omega_{c2} = \frac{\omega_{s2} + \omega_{p2}}{2}$

We then multiply the ideal impulse response with a **Kaiser window** to get the FIR impulse response:

$H(z) =$

$-0.0237Z^{0} + 0.0072Z^{-1} + 0.0480Z^{-2} + 0.0033Z^{-3} - 0.0480Z^{-4} - 0.0116Z^{-5}$

$+0.0159Z^{-6} - 0.0053Z^{-7} + 0.0338Z^{-8} + 0.0596Z^{-9} - 0.0712Z^{-10}$

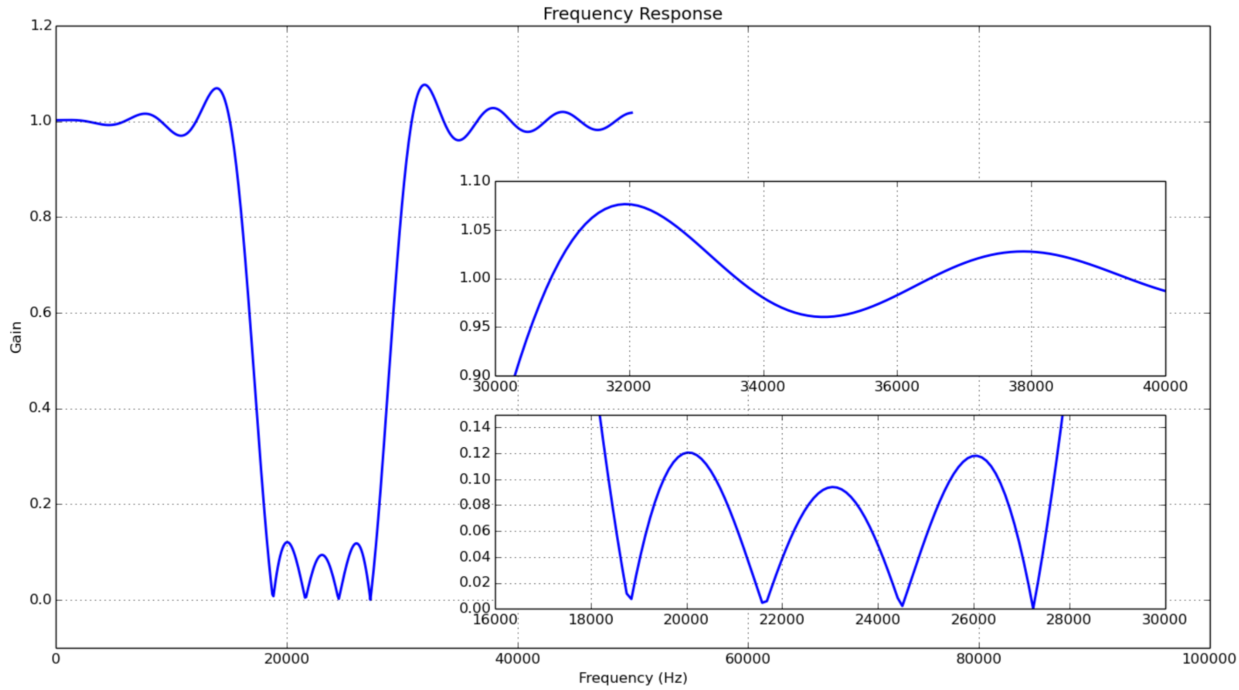
$-0.1392Z^{-11} + 0.0707Z^{-12} + 0.2111Z^{-13} - 0.0294Z^{-14} + 0.7600Z^{-15}$

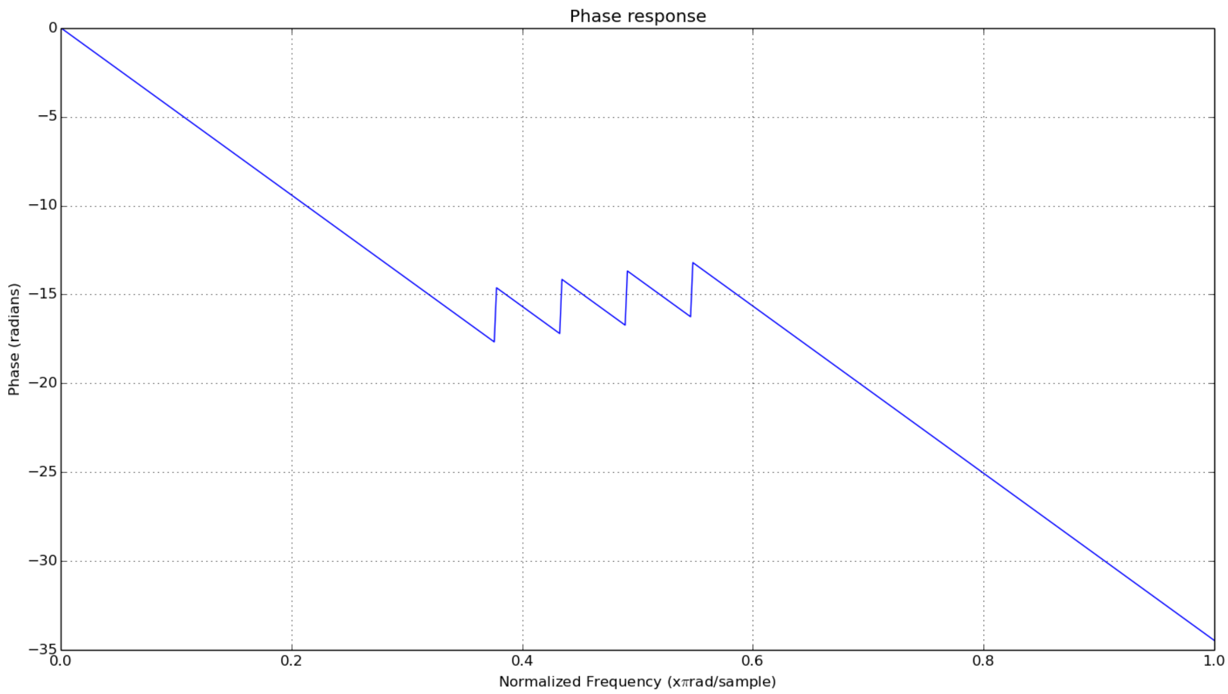
$-0.0294Z^{-16} + 0.2111Z^{-17} + 0.0707Z^{-18} - 0.1392Z^{-19} - 0.0712Z^{-20}$

$+0.0596Z^{-21} + 0.0338Z^{-22} - 0.0053Z^{-23} + 0.0159Z^{-24} - 0.0116Z^{-25}$

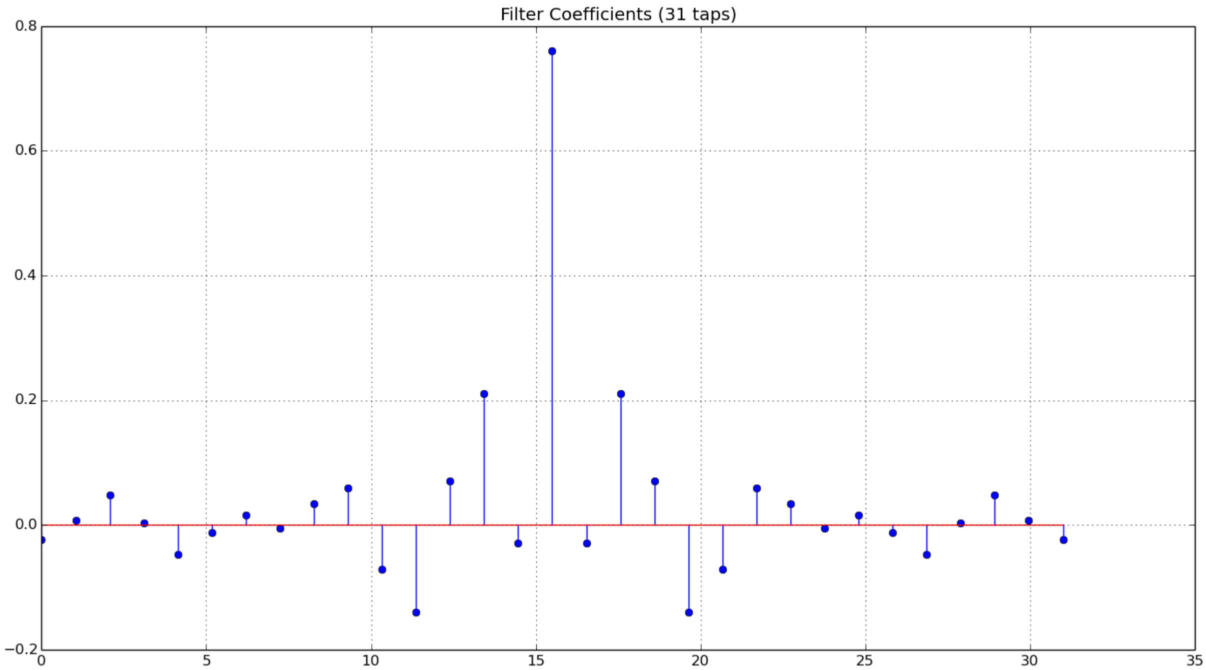
$-0.0480Z^{-26} + 0.0033Z^{-27} + 0.0480Z^{-28} + 0.0072Z^{-29} - 0.0237Z^{-30}$

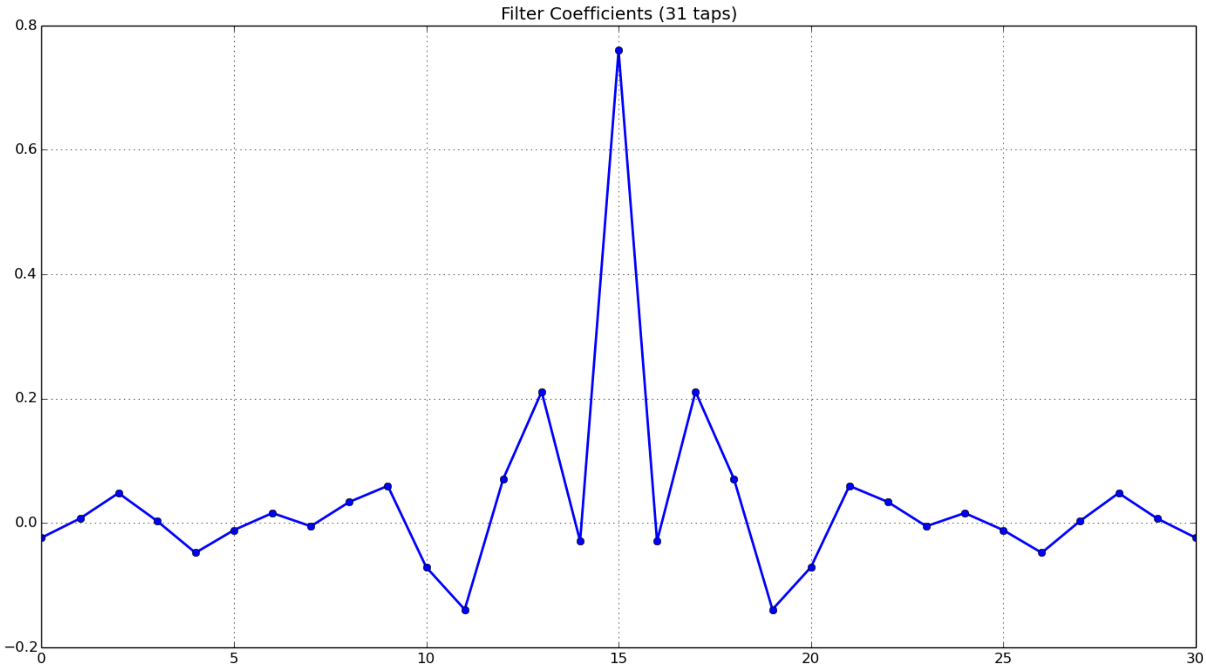
Phase and Frequency Response Plots of the filter





Filter coefficients Plots of the filter





2. References

- [1] : http://www.ece.uah.edu/courses/ee426/Chebyshev.pdf
- [2] : http://ocw.mit.edu/resources/res-6-008-digital-signal-processing/
- [3] : https://docs.scipy.org/doc/scipy-0.16.1/reference/generated/scipy.signal.kaiser.html
- [4] : http://docs.scipy.org/doc/numpy-1.10.1/reference/generated/numpy.poly1d.html

3. Code

3.1. Bandpass Filter

3.1.1. BP IIR Filter Code

```python
import numpy as np
import scipy as sp
import pylab as pl
import scipy.signal as sg
import matplotlib.pyplot as plt
import matplotlib.patches as pat

def printLatexPoly(coeffs):
    '''
    Utility function to print a numpy polynomial into Latex
    taking the coeffs of the polynomial (highest power first)
    as input
    '''
    latex_poly = ''
    for i in range(len(coeffs)):
        if i == 0:
            latex_poly += str( "%.4f" %coeffs[i]) + ' '
        else:
            if coeffs[i] > 0.0:
                latex_poly += ' + '
        latex_poly += ' ' + str( "%.4f" %coeffs[i]) + ' Z^{' + str(-i) + '} '
    print latex_poly

# Filter Specification Declaration
filter_number = 82
delta_stop    = 0.15
delta_pass    = 0.15
sampling_freq     = 100000
h_transistion = 2
l_transistion = 2
m,q,r         = 7,0,7

passband_lower_freq  = 4 + 0.7*q + 2*r
passband_higher_freq = passband_lower_freq + 10

# Analog Filter Frequencies initialization
omega_p1 = (passband_lower_freq)*1000.0
omega_p2 = (passband_higher_freq)*1000.0
omega_s1 = (passband_lower_freq-l_transistion)*1000.0
omega_s2 = (passband_higher_freq+h_transistion)*1000.0
analog_freq = np.array([omega_s1,omega_p1,omega_p2,omega_s2],dtype='f')

# Normalized digital frequencies
digital_freq = (analog_freq/sampling_freq)*2*np.pi

# Bilinear Transformation from (-pi,pi) to (-inf,inf)
equiv_analog_freq = np.tan(digital_freq/2)

# Values for frequency transformation
omega_z_s = equiv_analog_freq[1] * equiv_analog_freq[2]
f_B       = equiv_analog_freq[2] - equiv_analog_freq[1]

# Frequency Transformation for bandpass
equiv_analog_lowpass_freq = ((equiv_analog_freq**2)-omega_z_s)/(f_B*equiv_analog_freq)

# Values for Chebyschev low pass filter design
D_1   = (1/(1-delta_stop)**2)-1
D_2   = (1/delta_pass**2)-1
epsilon = np.sqrt(D_1)
abs_equiv_analog_lowpass_freq = abs(equiv_analog_lowpass_freq)

# Order Calculation and getting stringent omega
stringent_omega_s = min(abs_equiv_analog_lowpass_freq[0],abs_equiv_analog_lowpass_freq[3])
N = np.ceil(np.arccosh(np.sqrt(D_2)/np.sqrt(D_1)) / np.arccosh(stringent_omega_s/equiv_analog_lowpass_freq[2]))

# Pole calculation
omega_p  = equiv_analog_lowpass_freq[2]
poles    = np.zeros([2*N-1],dtype='complex64')
iterable = ((2*k+1)*np.pi/(2*N) for k in range(2*int(N)))
A_k = np.fromiter(iterable,float)
B   = np.arcsinh(1/epsilon)/N
poles = omega_p*np.sin(A_k)*np.sinh(B) + omega_p*np.cos(A_k)*np.cosh(B) * (1.j)

#Lowpass filter transfer function
a = 1+0.j
for c in poles:
  if(c.real< 0):
    a=np.poly1d([1,-c],r=0)*a
print "Low pass transfer function denominator\n",a

#Find gain for Chebyshev
chebyshev_k = 1
for k in range(int(N)):
    chebyshev_k = chebyshev_k*poles[k]
print "GAIN:",chebyshev_k

'''
For BPF when the transformation is applied to 1/(s-root) we get
Numerator = Bs
Denominator = s^2+omega_0^2-B*c*s
```

```
Using this basic result and finding numerator and denominator to get the bandpass transfer function
'''
analog_numer = chebyshev_k.real
analog_denom = 1+0.j
for c in poles:
  if(c.real<= 0):
    analog_numer = np.poly1d([f_B,0],r=0)*analog_numer
    analog_denom = np.poly1d([1,-f_B*c,omega_z_s],r=0)*analog_denom
print "Analog numerator\n",analog_numer
print "\nAnalog denominator\n",analog_denom

# Converting back to digital domain from transfer function
z,p,k=sg.tf2zpk(analog_numer,analog_denom)

plt.figure(5)
plt.grid(True)
plt.scatter(p.real,p.imag,s=50,c='b',marker='x')
plt.scatter(z.real,z.imag,s=50,c='b',marker='o')
plt.title('Pole Zero plot of Analog Bandpass filter')
plt.ylabel('Imaginary')
plt.xlabel('Real')

'''
For converting the bandpass filter to digital domain using
s = (1-z^-1)/(1+z^-1)
Numerator = B(z^2-1)
Denominator = (omega_0^2-B+1)z^2+(2*omega_0^2-2)z+(omega_0^2+B+1)
Using this basic result and finding numerator and denominator to get the bandpass transfer function
'''
digital_numer=chebyshev_k.real
digital_denom=1+0.j
for c in poles:
  if(c.real<= 0):
    digital_numer = np.poly1d([f_B,0,-f_B],r=0)*digital_numer
    digital_denom = np.poly1d([(omega_z_s-f_B*c+1),((2*omega_z_s)-2),(omega_z_s+f_B*c+1)],r=0)*digital_denom

z,p,k = sg.tf2zpk(digital_numer,digital_denom)

plt.figure(4)
plt.grid(True)
plt.scatter(p.real,p.imag,s=50,marker='x')
plt.scatter(z.real,z.imag,s=50,marker='o')
plt.title('Pole Zero plot of Digital Bandpass filter')
plt.ylabel('Imaginary')
plt.xlabel('Real')

print "Analog frequencies :",analog_freq
print "Digital frequencies :",digital_freq
print "Equivalent Digital frequencies : ",equiv_analog_freq
print "Equivalent Analog lpf freq :",equiv_analog_lowpass_freq
print "D1,D2 :",D_1,D_2
print "Poles :",poles
print "Order : ", N
print "stringent_omega :",stringent_omega_s
print "omega_z_s :",omega_z_s
print "B :",f_B
print "A-k :",A_k

print "Digital Numerator\n",digital_numer
print "\nDigital Denominator\n",digital_denom

# Plotting poles of low pass filter
plt.figure(1)
plt.grid(True)
neg_poles=np.zeros([0],dtype='complex64')
for c in poles:
  if(c.real<= 0):
    neg_poles=np.append(neg_poles,c)
plt.scatter(neg_poles.real,neg_poles.imag,s=50,marker='x')
plt.title('Pole Zero plot of Low pass filter')
plt.ylabel('Imaginary')
plt.xlabel('Real')

nmrz = (digital_numer.c).round(decimals=6).real
dmrz = (digital_denom.c).round(decimals=6).real

# Printing the latex digital polynomial
print "\nNormalized numerator Array:\n",nmrz/dmrz[0]
printLatexPoly(nmrz/dmrz[0])
print "\nNormalized denominator Array:\n",dmrz/dmrz[0]
printLatexPoly(dmrz/dmrz[0])

# Direct Form II for IIR
An=dmrz/dmrz[0]
Bn=nmrz/dmrz[0]
N=len(dmrz)
Kn=np.zeros(N)
Cn=np.zeros(N)
for i in np.arange(N-1,-1,-1):
    Kn[i]=An[i]
```

```
    Cn[i]=Bn[i]
    An_tilda=An[::-1]
    if(Kn[i] != 1):
        An=(An-(Kn[i]*An_tilda))/(1-(Kn[i]**2))
    Bn=(Bn-Cn[i]*An_tilda)
    An=np.delete(An,len(An)-1)
    Bn=np.delete(Bn,len(Bn)-1)

print "\nLattice Coefficients:Kn\n",Kn[1:]
print "\nLattice Coefficients:Cn\n",Cn

# Plot Frequency response
nyq_rate = sampling_freq/2
plt.figure(2)
plt.clf()
plt.grid(True)
w,h= sg.freqz(nmrz,dmrz,worN=512)
plt.plot((w/np.pi)*nyq_rate, np.absolute(h), linewidth=2)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Gain')
plt.title('Frequency Response')
plt.ylim(-0.05, 1.3)

# Zoomed plot
ax1 = plt.axes([0.44, 0.3, .45, .25])
plt.plot((w/np.pi)*nyq_rate, np.absolute(h), linewidth=2)
plt.xlim(16000.0,30000.0)
plt.ylim(0.85, 1.3)
plt.grid(True)

plt.figure(3)
plt.grid(True)
h_Phase = pl.unwrap(np.arctan2(np.imag(h),np.real(h)))
plt.plot(w/max(w),h_Phase)
plt.ylabel('Phase (radians)')
plt.xlabel(r'Normalized Frequency (x$\pi$rad/sample)')
plt.title(r'Phase response')
plt.show()
```

3.1.2. BP FIR Filter Code

```
import numpy as np
import scipy as sp
import pylab as pl
import scipy.signal as sg
import matplotlib.pyplot as plt

def printLatexPoly(coeffs):
    '''
    Utility function to print a numpy polynomial into Latex
    taking the coeffs of the polynomial (highest power first)
    as input
    '''
    latex_poly = ''
    for i in range(len(coeffs)):
        if i == 0:
            latex_poly += str( "%.4f" %coeffs[i]) + ' '
        else:
            if coeffs[i] > 0.0:
                latex_poly += ' + '
        latex_poly += ' ' + str( "%.4f" %coeffs[i]) + ' Z^{' + str(-i) + '} '
    print latex_poly

# Filter Specification Declaration
filter_number    = 82
delta_stop     = 0.15
delta_pass       = 0.15
sampling_freq    = 100000
m,q,r        = 7,0,7
h_transistion    = 2
l_transistion    = 2

passband_lower_freq  = 4 + 0.7*q + 2*r
passband_higher_freq = passband_lower_freq+10

# Analog Filter Frequencies initialization
omega_p1 = (passband_lower_freq)*1000.0
omega_p2 = (passband_higher_freq)*1000.0
omega_s1 = (passband_lower_freq - l_transistion) *1000.0
omega_s2 = (passband_higher_freq + h_transistion)*1000.0
analog_freq = np.array([omega_s1,omega_p1,omega_p2,omega_s2],dtype='f')

# Normalized digital frequencies
digital_freq = (analog_freq/sampling_freq)*2*np.pi

# Kaiser window parameters
del_omega1 = digital_freq[3] - digital_freq[2]
del_omega2 = digital_freq[1] - digital_freq[0]
del_omega  = min(abs(del_omega1),abs(del_omega2))
A        = -20*np.log10(delta_stop)
```

```
if(A<21):
    alpha=0
elif(A<=50):
    alpha=0.5842*(A-21)**0.4+0.07886(A-21)
else:
    alpha=0.1102(A-8.7)

# Order Calculation
N = np.ceil((A-8)/(2*2.285*del_omega))

# Cutoff frequency calculation ideal impulse response
omega_c1 = (digital_freq[1]+digital_freq[0])*0.5
omega_c2 = (digital_freq[3]+digital_freq[2])*0.5

# Obtain the ideal bandpass impulse response
iterable   = ((np.sin(omega_c2*k)-np.sin(omega_c1*k))/(np.pi*k) for k in range(int(-N),int(N+1)))
h_ideal    = np.fromiter(iterable,float)
h_ideal[N] = ((omega_c2-omega_c1)/np.pi)
beta       = alpha/N

# Generate Kaiser window
h_kaiser = sg.kaiser(2*N+1,beta)
h_org    = h_ideal*h_kaiser

print "FIR Filter Coefficients:\n",h_org
print "\nH(Z) = :\n"
printLatexPoly(h_org)
print "\nHideal",h_ideal

# Plot the FIR filter coefficients.
nyquist_rate = sampling_freq/2
plt.figure(1)
plt.plot(h_org, 'bo-', linewidth=2)
plt.title('Filter Coefficients (%d taps)' % (2*N+1))
plt.grid(True)

# Plot Frequency response
plt.figure(2)
plt.clf()
plt.grid(True)
w,h= sg.freqz(h_org)
plt.plot((w/np.pi)*nyquist_rate, np.absolute(h), linewidth=2)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Gain')
plt.title('Frequency Response')
plt.ylim(-0.05, 1.2)
plt.xlim(0, 100000)

# Zoomed plot 1
ax1 = plt.axes([0.42, 0.6, .45, .25])
plt.plot((w/np.pi)*nyquist_rate, np.absolute(h), linewidth=2)
plt.xlim(16000.0,30000.0)
plt.ylim(0.9, 1.15)
plt.grid(True)

# Zoomed plot 2
ax2 = plt.axes([0.42, 0.25, .45, .25])
plt.plot((w/np.pi)*nyquist_rate, np.absolute(h), linewidth=2)
plt.xlim(30000.0, 34000.0)
plt.ylim(0.0, 0.11)
plt.grid(True)

plt.figure(3)
plt.grid(True)
h_Phase = pl.unwrap(np.arctan2(np.imag(h),np.real(h)))
plt.plot(w/max(w),h_Phase)
plt.ylabel('Phase (radians)')
plt.xlabel(r'Normalized Frequency (x$\pi$rad/sample)')
plt.title(r'Phase response')

#Stem Diagram
plt.figure(4)
y = pl.linspace(0,h_org.shape[0],h_org.shape[0])
plt.stem(y,h_org,linefmt='b-', markerfmt='bo', basefmt='r-')
plt.title('Filter Coefficients (%d taps)' % (2*N+1))
plt.grid(True)
plt.show()
```

3.2. Bandstop Filter

3.2.1. BS IIR Filter Code

```
import numpy as np
import scipy as sp
import scipy.signal as sg
import matplotlib.pyplot as plt
import pylab as pl
```

```python
def printLatexPoly(coeffs):
    '''
    Utility function to print a numpy polynomial into Latex
    taking the coeffs of the polynomial (highest power first)
    as input
    '''
    latex_poly = ''
    for i in range(len(coeffs)):
        if i == 0:
            latex_poly += str( "%.4f" %coeffs[i]) + ' '
        else:
            if coeffs[i] > 0.0:
                latex_poly += ' + '
        latex_poly += ' ' + str( "%.4f" %coeffs[i]) + ' Z^{' + str(-i) + '} '
    print latex_poly

# Filter Specification Declaration
filter_number = 82
delta_stop    = 0.15
delta_pass    = 0.15
sampling_freq = 100000
h_transistion = 2
l_transistion = 2
m,q,r         = 7,0,7

stopband_lower_freq  = 4 + 0.7*q + 2*r
stopband_higher_freq = stopband_lower_freq + 10

# Analog Filter Frequencies initialization
omega_p1 = (stopband_lower_freq)*1000.0
omega_p2 = (stopband_higher_freq)*1000.0
omega_s1 = (stopband_lower_freq-l_transistion)*1000.0
omega_s2 = (stopband_higher_freq+h_transistion)*1000.0
analog_freq = np.array([omega_s1,omega_p1,omega_p2,omega_s2],dtype='f')

# Normalized digital frequencies
digital_freq=(analog_freq/sampling_freq)*2*np.pi

# Bilinear Transformation from (-pi,pi) to (-inf,inf)
equiv_analog_freq=np.tan(digital_freq/2)

# Values for frequency transformation
omega_z_s=equiv_analog_freq[0]*equiv_analog_freq[3]
f_B=equiv_analog_freq[3]-equiv_analog_freq[0]

#Frequency Transformation for bandstop
equiv_analog_lowpass_freq=(f_B*equiv_analog_freq)/(omega_z_s-(equiv_analog_freq**2))

# Values for Butterworth low pass filter design
D_1     = (1/(1-delta_stop)**2)-1
D_2     = (1/delta_pass**2)-1
epsilon = np.sqrt(D_1)
mod_equiv_analog_lowpass_freq = abs(equiv_analog_lowpass_freq)

# Order Calculation and getting stringent omega
stringent_omega_s=min(mod_equiv_analog_lowpass_freq[1],mod_equiv_analog_lowpass_freq[2])
N = np.ceil(np.log(np.sqrt(D_2)/np.sqrt(D_1))/np.log(stringent_omega_s/equiv_analog_lowpass_freq[0]))

# Pole calculations
omega_p = equiv_analog_lowpass_freq[0]
omega_c = ((omega_p/(D_1**(1/(2*N))))+(stringent_omega_s/(D_2**(1/(2*N)))))/2
poles   = np.zeros([2*N-1],dtype='complex64')
iterable = ((2*k+1)*np.pi/(2*N) for k in range(2*int(N)))
xp = np.fromiter(iterable,float)
poles = (1.j)*omega_c*np.exp(1.j*xp)

#Lowpass filter transfer function
a=1+0.j
for c in poles:
  if(c.real< 0):
    a=np.poly1d([1,-c],r=0)*a
print "Low pass transfer function denominator\n",a

#Find gain for Butterworth
butter_k=omega_c**N
print "GAIN:",butter_k

'''
For BPF when the transformation is applied to 1/(s-root) we get
Numerator = s^2 + omega_0^2
Denominator = -c*s^2 + B*s -c*omega_0^2
Using this basic result and finding numerator and denominator to get the bandpass transfer function
'''
analog_numer = butter_k
analog_denom = 1+0.j
for c in poles:
  if(c.real<= 0):
    analog_numer = np.poly1d([1,0,omega_z_s],r=0)*analog_numer
    analog_denom = np.poly1d([-c,f_B,-c*omega_z_s],r=0)*analog_denom
print "Analog numerator\n",analog_numer
```

```
print "\nAnalog denominator\n",analog_denom

# Converting back to digital domain from transfer function
z,p,k=sg.tf2zpk(analog_numer,analog_denom)

plt.figure(5)
plt.grid(True)
plt.scatter(p.real,p.imag,s=50,marker='x')
plt.scatter(z.real,z.imag,s=50,marker='o')
plt.title('Pole Zero plot of Analog Bandstop filter')
plt.ylabel('Imaginary')
plt.xlabel('Real')

'''
For converting the bandpass filter to digital domain using
s = (1-z^-1)/(1+z^-1)
Numerator = (omega_0^2+1)z^2+2*(omega_0^2-1)z+(omega_0^2+1)
Denominator = (-B-c-c*omega_0^2)z^2+(2*c-2*c*omega_0^2)z+(-c*omega_0^2-c+B)
Using this basic result and finding numerator and denominator to get the bandpass transfer function
'''
digital_numer=butter_k
digital_denom=1+0.j
for c in poles:
  if(c.real<= 0):
    digital_numer=np.poly1d([(omega_z_s+1),2*(omega_z_s-1),(omega_z_s+1)],r=0)*digital_numer
    digital_denom=np.poly1d([(-f_B-omega_z_s*c-c),(2*c-2*c*omega_z_s),(-c+f_B-c*omega_z_s)],r=0)*digital_denom

print "Analog frequencies :",analog_freq
print "Digital frequencies :",digital_freq
print "Equivalent Digital frequencies : ",equiv_analog_freq
print "Equivalent Analog lpf freq :",equiv_analog_lowpass_freq
print "D1,D2 :",D_1,D_2
print "Poles :",poles
print "Order : ", N
print "stringent_omega :",stringent_omega_s
print "omega_z_s :",omega_z_s
print "B :",f_B

print "Digital Numerator\n",digital_numer
print "\nDigital Denominator:\n",digital_denom

# Plotting poles of low pass filter
plt.figure(1)
plt.grid(True)
neg_poles=np.zeros([0],dtype='complex64')
for c in poles:
  if(c.real<= 0):
    neg_poles=np.append(neg_poles,c)
plt.scatter(neg_poles.real,neg_poles.imag,s=50,marker='x')
plt.title('Pole Zero plot of Low pass filter')
plt.ylabel('Imaginary')
plt.xlabel('Real')
plt.xlim(-1.5,1.5)
plt.ylim(-1.5,1.5)

nmrz = (digital_numer.c).round(decimals=6)[::-1].real
dmrz = (digital_denom.c).round(decimals=6)[::-1].real

z,p,k = sg.tf2zpk(nmrz,dmrz)

plt.figure(4)
plt.grid(True)
plt.scatter(p.real,p.imag,s=50,marker='x')
plt.scatter(z.real,z.imag,s=50,marker='o')
plt.title('Pole Zero plot of Digital Bandstop filter')
plt.ylabel('Imaginary')
plt.xlabel('Real')

# Printing the latex digital polynomial
print "\nNormalized numerator:\n",nmrz/dmrz[0]
printLatexPoly(nmrz/dmrz[0])
print "\nNormalized denominator:\n",dmrz/dmrz[0]
printLatexPoly(dmrz/dmrz[0])

# Direct Form II for IIR
An=dmrz/dmrz[0]
Bn=nmrz/dmrz[0]
N=len(dmrz)
Kn=np.zeros(N)
Cn=np.zeros(N)
for i in np.arange(N-1,-1,-1):
    Kn[i]=An[i]
    Cn[i]=Bn[i]
    An_tilda=An[::-1]
    if(Kn[i] != 1):
        An=(An-(Kn[i]*An_tilda))/(1-(Kn[i]**2))
    Bn=(Bn-Cn[i]*An_tilda)
    An=np.delete(An,len(An)-1)
    Bn=np.delete(Bn,len(Bn)-1)
```

```
print "\nLattice Coefficients:Kn\n",Kn[1:]
print "\nLattice Coefficients:Cn\n",Cn

#Plot Frequency response
nyq_rate=sampling_freq/2
plt.figure(2)
plt.clf()
plt.grid(True)
w,h= sg.freqz(nmrz,dmrz,worN=512)
plt.plot((w/np.pi)*nyq_rate, np.absolute(h), linewidth=2)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Gain')
plt.title('Frequency Response')
plt.ylim(-0.05, 1.05)

# Zoomed plot 2
ax1 = plt.axes([0.44, 0.56, .45, .25])
plt.plot((w/np.pi)*nyq_rate, np.absolute(h), linewidth=2)
plt.xlim(16000.0,30000.0)
plt.ylim(-0.01, 0.2)
plt.grid(True)

# Zoomed plot 1
ax1 = plt.axes([0.44, 0.22, .45, .25])
plt.plot((w/np.pi)*nyq_rate, np.absolute(h), linewidth=2)
plt.xlim(28000.0,35000.0)
plt.ylim(0.86, 1.05)
plt.grid(True)

plt.figure(3)
plt.grid(True)
h_Phase = pl.unwrap(np.arctan2(np.imag(h),np.real(h)))
plt.plot(w/max(w),h_Phase)
plt.ylabel('Phase (radians)')
plt.xlabel(r'Normalized Frequency (x$\pi$rad/sample)')
plt.title(r'Phase response')
plt.show()
```

3.2.2. BS FIR Filter Code

```
import numpy as np
import scipy as sp
import scipy.signal as sg
import matplotlib.pyplot as plt
import pylab as pl

def printLatexPoly(coeffs):
    '''
    Utility function to print a numpy polynomial into Latex
    taking the coeffs of the polynomial (highest power first)
    as input
    '''
    latex_poly = ''
    for i in range(len(coeffs)):
        if i == 0:
            latex_poly += str( "%.4f" %coeffs[i]) + ' '
        else:
            if coeffs[i] > 0.0:
                latex_poly += ' + '
        latex_poly += ' ' + str( "%.4f" %coeffs[i]) + ' Z^{' + str(-i) + '} '
    print latex_poly

# Filter Specification Declaration
filter_number   = 82
delta_stop      = 0.15
delta_pass      = 0.15
sampling_freq   = 100000
m,q,r           = 7,0,7
h_transition   = 2
l_transition   = 2

stopband_lower_freq  = 4 + 0.7*q + 2*r
stopband_higher_freq = stopband_lower_freq+10

# Analog Filter Frequencies initialization
omega_s1=(stopband_lower_freq)*1000.0
omega_s2=(stopband_higher_freq)*1000.0
omega_p1=(stopband_lower_freq-l_transition)*1000.0
omega_p2=(stopband_higher_freq+h_transition)*1000.0
analog_freq=np.array([omega_p1,omega_s1,omega_s2,omega_p2],dtype='f')

# Normalized digital frequencies
digital_freq=(analog_freq/sampling_freq)*2*np.pi


# Kaiser window parameters
del_omega1 = digital_freq[3]-digital_freq[2]
del_omega2 = digital_freq[1]-digital_freq[0]
del_omega  = min(abs(del_omega1),abs(del_omega2))
A          = -20*np.log10(delta_stop)
```

```python
if(A<21):
    alpha=0
elif(A<=50):
    alpha=0.5842*(A-21)**0.4+0.07886(A-21)
else:
    alpha=0.1102(A-8.7)

# Order Calculation
N = np.ceil((A-8)/(2*2.285*del_omega))

# Cutoff frequency calculation ideal impulse response
omega_c1 = (digital_freq[1]+digital_freq[0])*0.5
omega_c2 = (digital_freq[3]+digital_freq[2])*0.5

# Obtain the ideal bandpass impulse response
iterable   = ((np.sin(omega_c1*k)-np.sin(omega_c2*k))/(np.pi*k) for k in range(int(-N),int(N+1)))
h_ideal    = np.fromiter(iterable,float)
h_ideal[N] = ((omega_c1-omega_c2)/np.pi)+1
beta       = alpha/N

# Generate Kaiser window
h_kaiser=sg.kaiser(2*N+1,beta)
h_org=h_ideal*h_kaiser

print "FIR Filter Coefficients:\n",h_org
print "\nH(Z) = :\n"
printLatexPoly(h_org)
print "\nHideal",h_ideal

# Plot the FIR filter coefficients
nyq_rate=sampling_freq/2
plt.figure(1)
plt.plot(h_org, 'bo-', linewidth=2)
plt.title('Filter Coefficients (%d taps)' % (2*N+1))
plt.grid(True)

# Plot Frequency response
plt.figure(2)
plt.clf()
plt.grid(True)
w,h= sg.freqz(h_org)
plt.plot((w/np.pi)*nyq_rate, np.absolute(h), linewidth=2)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Gain')
plt.title('Frequency Response')
plt.ylim(-0.1, 1.2)
plt.xlim(0,100000)

# Zoomed plot 1
ax1 = plt.axes([0.42, 0.45,.45, .25])
plt.plot((w/np.pi)*nyq_rate, np.absolute(h), linewidth=2)
plt.xlim(30000.0,40000.0)
plt.ylim(0.9, 1.1)
plt.grid(True)

# Zoomed plot 2
ax2 = plt.axes([0.42, 0.15, .45, .25])
plt.plot((w/np.pi)*nyq_rate, np.absolute(h), linewidth=2)
plt.xlim(16000.0, 30000.0)
plt.ylim(0.0, 0.15)
plt.grid(True)

plt.figure(3)
plt.grid(True)
h_Phase = pl.unwrap(np.arctan2(np.imag(h),np.real(h)))
plt.plot(w/max(w),h_Phase)
plt.ylabel('Phase (radians)')
plt.xlabel(r'Normalized Frequency (x$\pi$rad/sample)')
plt.title(r'Phase response')

#Stem Diagram
plt.figure(4)
y = pl.linspace(0,h_org.shape[0],h_org.shape[0])
plt.stem(y,h_org,linefmt='b-', markerfmt='bo', basefmt='r-')
plt.title('Filter Coefficients (%d taps)' % (2*N+1))
plt.grid(True)
plt.show()
```