Digital Filter Design in Python Based on Convex Optimization

Shahrokh Hamidi
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada
Email: shahrokh.hamidi@uwaterloo.ca

I. Digital Filter Design

This section has been dedicated to Finite Impulse Response (FIR) digital filter design. The approach is based on the Chebyshev design which is described as

$$\begin{aligned} \min_{\mathbf{h},\delta_2} \quad & \delta_2 \\ \text{s.t.} \quad & \frac{1}{\delta_1} \leq H(\omega_k) \leq \delta_1, \quad 0 \leq \omega_k \leq \omega_p, \\ & -\delta_2 \leq H(\omega_k) \leq \delta_2, \quad \omega_s \leq \omega_k \leq \pi, \end{aligned} \tag{1}$$

where ω_p, ω_s are the pass-band and stop-band cut-off frequencies, respectively. In addition, $\pm 20\log_1 0\delta_1$ is the pass-band ripple. Furthermore, $-20\log_1 0\delta_2$ is the stop-band attenuation.

Finally $H(\omega) = \sum_{n=1}^{N} h[n]\exp(-j\omega n/N)$.

Digital filter design problem can be written in different forms. The convex optimization problem given in (1) is based on minimizing the stop-band attenuation.

A. Low Pass Filter

The Low Pass Filter (LPF) design is performed based on the following min-max optimization problem

$$\begin{aligned} \min_{\mathbf{h}} \quad & \max |\mathbf{H}(\omega_{\mathbf{i}})| \\ \text{s.t.} \quad & \frac{1}{\delta_1} \leq \mathbf{H}(\omega_{\mathbf{k}}) \leq \delta_1, \quad 0 \leq \omega_k \leq \omega_p, \\ & \omega_s \leq \omega_i \leq \pi. \end{aligned} \tag{2}$$

The code for the filter design has been given as follows.

```
#Low pass filter

import numpy as np
import matplotlib.pyplot as plt
import sys
import cvxpy as cp
import matplotlib
```

```
import warnings
warnings.filterwarnings('ignore')



#%matplotlib qt


class Params:


    N  = 256
    Fs = 40
    fp = 3/Fs
    fs = 9/Fs
    filter_order = 10
    delta = 0.5
    freq = np.linspace(0,1,N)
    def __init__(self):

        pass






class  Opt(Params):

    def __init__(self):

        super(Opt, self).__init__()
        self.utils()


    def utils(self):

        A = 2*np.cos(2*np.pi*Params.freq.reshape(-1,1)*np.arange(0,Params.filter_order))
        index_p = np.where(Params.freq<Params.fp)[0].tolist()
        index_s = np.where(((Params.fs<Params.freq) & (Params.freq<1/2.)))[0].tolist()
        A[:,0] = 1
        self.Ap = A[index_p,:]
        self.As = A[index_s,:]



    def _cost(self, hp):

        cost = cp.max(cp.abs(self.As@hp))
        return cost

    def _constr(self, hp):

        constr = []
        constr += [10**(-Params.delta/20) <= self.Ap @ hp]
        constr += [self.Ap @ hp  <=  10**(Params.delta/20)]

        return constr


    def _run(self):

        hp = cp.Variable(Params.filter_order)
```

```
        prob = cp.Problem(cp.Minimize(self._cost(hp)), self._constr(hp))
        prob.solve()

        return hp.value


class Filter(Params):

    def __init__(self):
        pass


    @staticmethod
    def impulse_response(h):

        h = np.hstack((h[:0:-1], h))

        return Filter.frequency_response(h)

    def frequency_response(h):

        H = np.fft.fft(h, Params.N)

        return h, H



class Display:


    def __init__(self, H, h, msg):

        self.msg = msg
        self.display_IPR(h)
        self.display_IPR_H(H)
        self.display_phase(H)

    def display_IPR(self, h):

        plt.figure()
        plt.stem(h)
        plt.plot(h, 'r--')
        plt.grid()
        plt.xlabel('$Samples$', FontSize = 16)
        plt.ylabel('$IPR$', FontSize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        #plt.savefig('FIR_LPF.png')
        plt.tight_layout()
        plt.show()

    def display_IPR_H(self, H):

        H = abs(H)
        plt.figure()
        plt.plot(Params.freq*Params.Fs, 20*np.log10(H), 'k', lw = 3)
        #plt.plot(Params.freq*Params.Fs, 0.5*np.ones(len(H)), 'r--', lw = 1)
        #plt.plot(Params.freq*Params.Fs, -0.5*np.ones(len(H)), 'r--', lw = 1)
        plt.grid()
        plt.xlabel('$Frequency\;\; [Hz]$', FontSize = 16)
```

```
        plt.ylabel('$20\;log\;|H(f)|\;\;[dB]$', FontSize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        plt.xlim(0,Params.Fs//2)
        #plt.savefig('FIR_LPF.png')
        plt.tight_layout()
        plt.show()



    def display_phase(self, H):

        plt.figure()
        plt.plot(Params.freq*Params.Fs, np.unwrap(np.angle(H)), 'k', lw = 3)
        #plt.plot(Params.freq*Params.Fs, 0.5*np.ones(len(H)), 'r--', lw = 1)
        #plt.plot(Params.freq*Params.Fs, -0.5*np.ones(len(H)), 'r--', lw = 1)
        plt.grid()
        plt.xlabel('$Frequency\;\; [Hz]$', FontSize = 16)
        plt.ylabel('$Phase\;\;[rad]$', FontSize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{self.msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        plt.xlim(0,Params.Fs//2)
        #plt.savefig('FIR_LPF.png')
        plt.tight_layout()
        plt.show()



if __name__ == '__main__':

    msg = 'Low'
    Params()
    opt = Opt()
    h = opt._run()
    h, H = Filter.impulse_response(h)

    Display(H, h, msg)
```

Fig. 1 illustrates the result of the code for the LPF design.

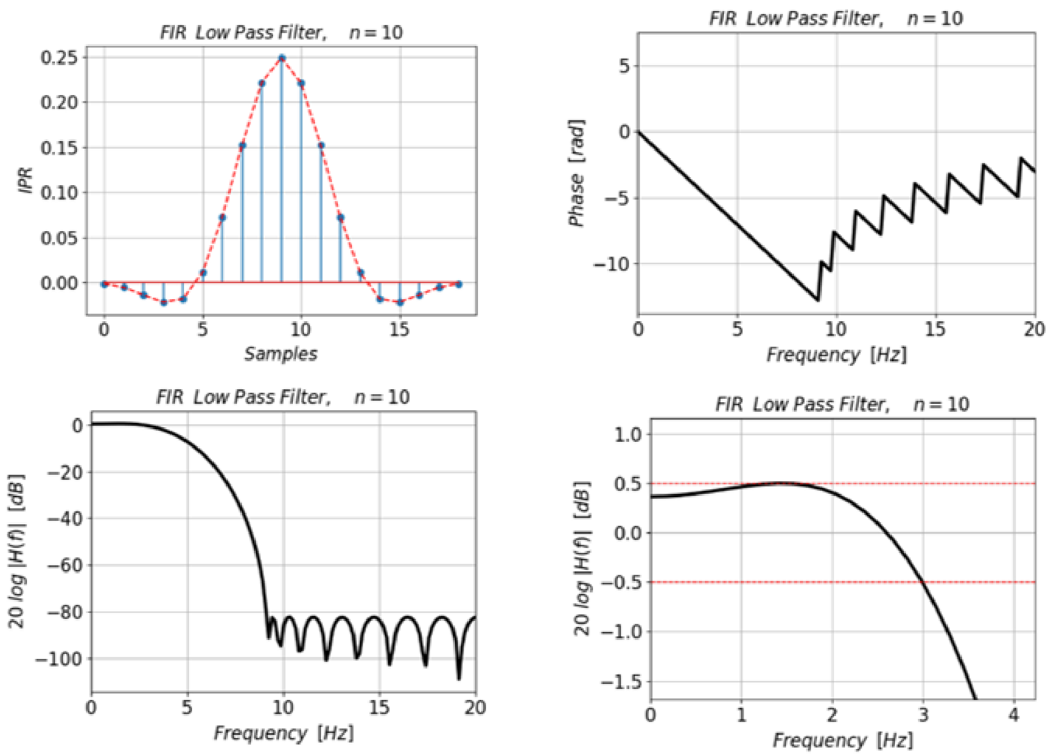

Fig. 1. The result for the LPF design.

B. *Band Pass Filter*

The Band Pass Filter (BPF) design is performed based on the following min-max optimization problem

$$\begin{aligned} \min_{\mathbf{h}} \quad & \max |\mathbf{H}(\omega_\mathbf{i})| \\ \text{s.t.} \quad & \frac{1}{\delta_1} \leq \mathbf{H}(\omega_\mathbf{k}) \leq \delta_1, \quad \omega_{p1} \leq \omega_k \leq \omega_{p2}, \\ & 0 \leq \omega_i \leq \omega_{s1} \cup \omega_{s2} \leq \omega_i \leq \pi. \end{aligned} \tag{3}$$

The code for the filter design is given as follows.

```
# Band pass filter

import numpy as np
import matplotlib.pyplot as plt
import sys
import cvxpy as cp
import matplotlib
import warnings
warnings.filterwarnings('ignore')



#%matplotlib qt


class Params:


    N  = 256
```

```python
    Fs = 40
    fp1 = 4/Fs
    fp2 = 7/Fs
    fs1 = 2/Fs
    fs2 = 9/Fs
    filter_order = 10
    delta = 0.5
    freq = np.linspace(0,1,N)
    def __init__(self):

        pass




class  Opt(Params):

    def __init__(self):

        super(Opt, self).__init__()
        self.utils()


    def utils(self):

        A = 2*np.cos(2*np.pi*Params.freq.reshape(-1,1)*np.arange(0,Params.filter_order))
        index_p = np.where(((Params.fp1<Params.freq) & (Params.freq<Params.fp2)))[0].tolist()
        index_s1 = np.where(((0<Params.freq) & (Params.freq<Params.fs1)))[0].tolist()
        index_s2 = np.where(((Params.fs2<Params.freq) & (Params.freq<1/2.)))[0].tolist()
        index_s = np.hstack((index_s1, index_s2))
        A[:,0] = 1
        self.Ap = A[index_p,:]
        self.As = A[index_s,:]



    def _cost(self, hp):

        cost = cp.max(cp.abs(self.As@hp))
        return cost

    def _constr(self, hp):

        constr = []
        constr += [10**(-Params.delta/20) <= self.Ap @ hp]
        constr += [self.Ap @ hp  <=  10**(Params.delta/20)]

        return constr


    def _run(self):

        hp = cp.Variable(Params.filter_order)
        prob = cp.Problem(cp.Minimize(self._cost(hp)), self._constr(hp))
        prob.solve()

        return hp.value


class Filter(Params):
```

```
    def __init__(self):
        pass


    @staticmethod
    def impulse_response(h):

        h = np.hstack((h[:0:-1], h))

        return Filter.frequency_response(h)

    def frequency_response(h):

        H = np.fft.fft(h, Params.N)

        return h, H



class Display:


    def __init__(self, H, h, msg):

        self.msg = msg
        self.display_IPR(h)
        self.display_IPR_H(H)
        self.display_phase(H)

    def display_IPR(self, h):

        plt.figure()
        plt.stem(h)
        plt.plot(h, 'r--')
        plt.grid()
        plt.xlabel('$Samples$', FontSize = 16)
        plt.ylabel('$IPR$', FontSize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        #plt.savefig('FIR_LPF.png')
        plt.tight_layout()
        plt.show()

    def display_IPR_H(self, H):

        H = abs(H)
        plt.figure()
        plt.plot(Params.freq*Params.Fs, 20*np.log10(H), 'k', lw = 3)
        #plt.plot(Params.freq*Params.Fs, 0.5*np.ones(len(H)), 'r--', lw = 1)
        #plt.plot(Params.freq*Params.Fs, -0.5*np.ones(len(H)), 'r--', lw = 1)
        plt.grid()
        plt.xlabel('$Frequency\;\; [Hz]$', FontSize = 16)
        plt.ylabel('$20\;log\;|H(f)|\;\;[dB]$', FontSize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        plt.xlim(0,Params.Fs//2)
        #plt.savefig('FIR_LPF.png')
```

```
        plt.tight_layout()
        plt.show()



    def display_phase(self, H):

        plt.figure()
        plt.plot(Params.freq*Params.Fs, np.unwrap(np.angle(H)), 'k', lw = 3)
        #plt.plot(Params.freq*Params.Fs, 0.5*np.ones(len(H)), 'r--', lw = 1)
        #plt.plot(Params.freq*Params.Fs, -0.5*np.ones(len(H)), 'r--', lw = 1)
        plt.grid()
        plt.xlabel('$Frequency\;\; [Hz]$', FontSize = 16)
        plt.ylabel('$Phase\;\;[rad]$', FontSize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{self.msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        plt.xlim(0,Params.Fs//2)
        #plt.savefig('FIR_LPF.png')
        plt.tight_layout()
        plt.show()




if __name__ == '__main__':

    msg = 'Band'
    Params()
    opt = Opt()
    h = opt._run()
    h, H = Filter.impulse_response(h)

    Display(H, h, msg)
```

Fig. 2 shows the result of the code for the BPF design.

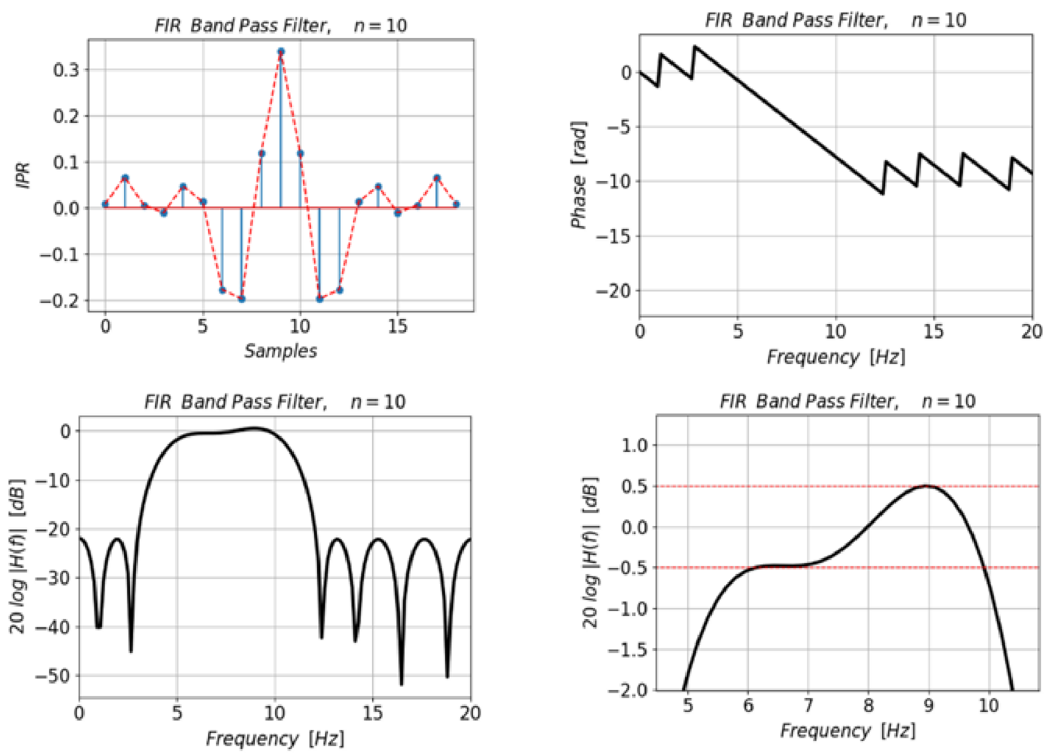

Fig. 2. The result for the BPF design.

C. Band Stop Filter

The Band Stop Filter (BSF) design is performed based on the following min-max optimization problem

$$
\begin{aligned}
\min_{\mathbf{h}} \quad & \max |\mathbf{H}(\omega_{\mathbf{i}})| \\
\text{s.t.} \quad & \frac{1}{\delta_1} \leq \mathbf{H}(\omega_{\mathbf{k}}) \leq \delta_1, \quad 0 \leq \omega_k \leq \omega_{s1} \cup \omega_{s2} \leq \omega_k \leq \pi, \\
& \omega_{p1} \leq \omega_i \leq \omega_{p2}.
\end{aligned}
\tag{4}
$$

The code for the filter design is given as follows.

```
# Band stop filter


import numpy as np
import matplotlib.pyplot as plt
import sys
import cvxpy as cp
import matplotlib
import warnings
warnings.filterwarnings('ignore')



#%matplotlib qt


class Params:
```

```
    N  = 256
    Fs = 40
    fs1 = 4/Fs
    fs2 = 7/Fs
    fp1 = 2/Fs
    fp2 = 9/Fs
    filter_order = 15
    delta = 0.5
    freq = np.linspace(0,1,N)
    def __init__(self):

        pass






class  Opt(Params):

    def __init__(self):

        super(Opt, self).__init__()
        self.utils()


    def utils(self):

        A = 2*np.cos(2*np.pi*Params.freq.reshape(-1,1)*np.arange(0,Params.filter_order))
        index_s = np.where(((Params.fs1<Params.freq) & (Params.freq<Params.fs2)))[0].tolist()
        index_p1 = np.where(((0<Params.freq) & (Params.freq<Params.fp1)))[0].tolist()
        index_p2 = np.where(((Params.fp2<Params.freq) & (Params.freq<1/2.)))[0].tolist()
        index_p = np.hstack((index_p1, index_p2))
        A[:,0] = 1
        self.Ap = A[index_p,:]
        self.As = A[index_s,:]



    def _cost(self, hp):

        cost = cp.max(cp.abs(self.As@hp))
        return cost

    def _constr(self, hp):

        constr = []
        constr += [10**(-Params.delta/20) <= self.Ap @ hp]
        constr += [self.Ap @ hp  <=  10**(Params.delta/20)]

        return constr


    def _run(self):

        hp = cp.Variable(Params.filter_order)
        prob = cp.Problem(cp.Minimize(self._cost(hp)), self._constr(hp))
        prob.solve()

        return hp.value


class Filter(Params):
```

```
    def __init__(self):
        pass


    @staticmethod
    def impulse_response(h):

        h = np.hstack((h[:0:-1], h))

        return Filter.frequency_response(h)

    def frequency_response(h):

        H = np.fft.fft(h, Params.N)

        return h, H



class Display:


    def __init__(self, H, h, msg):

        self.msg = msg
        self.display_IPR(h)
        self.display_IPR_H(H)
        self.display_phase(H)

    def display_IPR(self, h):

        plt.figure()
        plt.stem(h)
        plt.plot(h, 'r--')
        plt.grid()
        plt.xlabel('$Samples$', fontsize = 16)
        plt.ylabel('$IPR$', fontsize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        #plt.savefig('FIR_LPF.png')
        plt.tight_layout()
        plt.show()

    def display_IPR_H(self, H):

        H = abs(H)
        plt.figure()
        plt.plot(Params.freq*Params.Fs, 20*np.log10(H), 'k', lw = 3)
        #plt.plot(Params.freq*Params.Fs, 0.5*np.ones(len(H)), 'r--', lw = 1)
        #plt.plot(Params.freq*Params.Fs, -0.5*np.ones(len(H)), 'r--', lw = 1)
        plt.grid()
        plt.xlabel('$Frequency\;\; [Hz]$', fontsize = 16)
        plt.ylabel('$20\;log\;|H(f)|\;\;[dB]$', fontsize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        plt.xlim(0,Params.Fs//2)
```

```
        #plt.savefig('FIR_LPF.png')
        plt.tight_layout()
        plt.show()



    def display_phase(self, H):

        plt.figure()
        plt.plot(Params.freq*Params.Fs, np.unwrap(np.angle(H)), 'k', lw = 3)
        #plt.plot(Params.freq*Params.Fs, 0.5*np.ones(len(H)), 'r--', lw = 1)
        #plt.plot(Params.freq*Params.Fs, -0.5*np.ones(len(H)), 'r--', lw = 1)
        plt.grid()
        plt.xlabel('$Frequency\;\; [Hz]$', fontsize = 16)
        plt.ylabel('$Phase\;\;[rad]$', fontsize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{self.msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        plt.xlim(0,Params.Fs//2)
        plt.tight_layout()
        plt.show()




if __name__ == '__main__':

    msg = 'Band-Stop'
    Params()
    opt = Opt()
    h = opt._run()
    h, H = Filter.impulse_response(h)

    Display(H, h, msg)
```

Fig. 3 shows the result of the code for the BSF design.

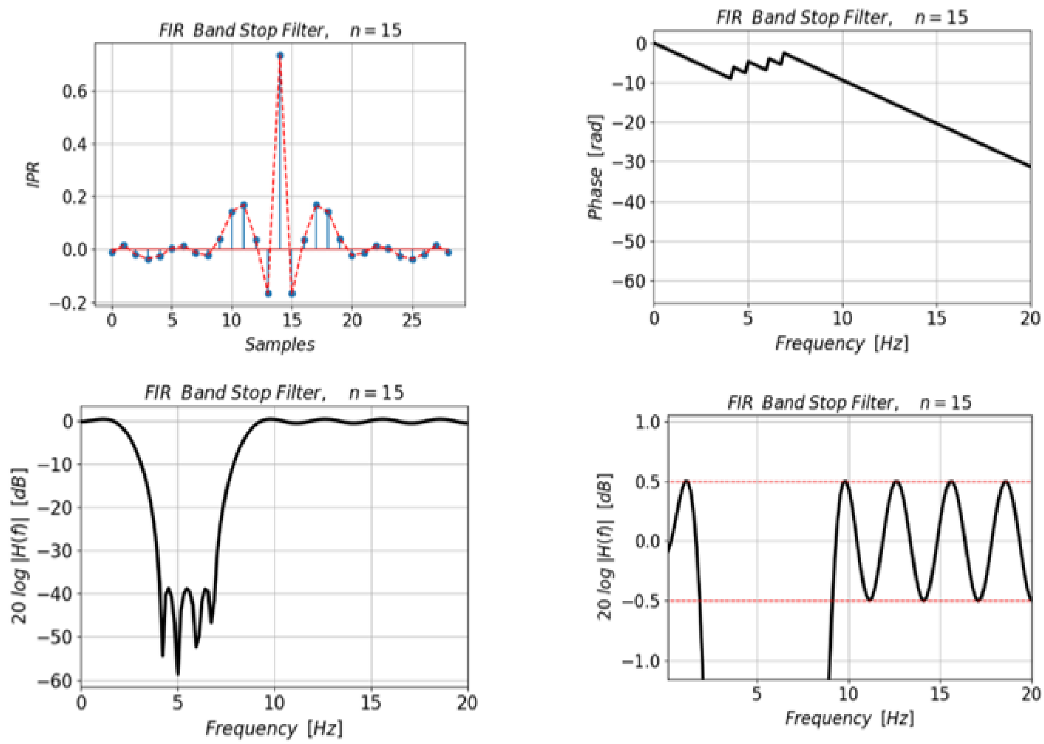

Fig. 3. The result for the BSF design.

D. *High Pass Filter*

The Low Pass Filter (LPF) design is performed based on the following min-max optimization problem

$$\begin{aligned} \min_{\mathbf{h}} \quad & \max |\mathbf{H}(\omega_{\mathbf{i}})| \\ \text{s.t.} \quad & \frac{1}{\delta_1} \leq \mathbf{H}(\omega_{\mathbf{k}}) \leq \delta_1, \quad \omega_p \leq \omega_k \leq \pi, \\ & 0 \leq \omega_i \leq \omega_s. \end{aligned} \tag{5}$$

The code for the filter design has been given as follows.

```
# High pass filter

import numpy as np
import matplotlib.pyplot as plt
import sys
import cvxpy as cp
import matplotlib
import warnings
warnings.filterwarnings('ignore')



#%matplotlib qt


class Params:


    N  = 256
```

```
    Fs = 40
    fp = 15/Fs

    fs = 12/Fs
    filter_order = 10
    delta = 0.5
    freq = np.linspace(0,1,N)
    def __init__(self):

        pass




class  Opt(Params):

    def __init__(self):

        super(Opt, self).__init__()
        self.utils()


    def utils(self):

        A = 2*np.cos(2*np.pi*Params.freq.reshape(-1,1)*np.arange(0,Params.filter_order))

        index_s = np.where(((0<Params.freq) & (Params.freq<Params.fs)))[0].tolist()
        index_p = np.where(((Params.fp<Params.freq) & (Params.freq<1/2.)))[0].tolist()

        A[:,0] = 1
        self.Ap = A[index_p,:]
        self.As = A[index_s,:]



    def _cost(self, hp):

        cost = cp.max(cp.abs(self.As@hp))
        return cost

    def _constr(self, hp):

        constr = []
        constr += [10**(-Params.delta/20) <= self.Ap @ hp]
        constr += [self.Ap @ hp  <=  10**(Params.delta/20)]

        return constr


    def _run(self):

        hp = cp.Variable(Params.filter_order)
        prob = cp.Problem(cp.Minimize(self._cost(hp)), self._constr(hp))
        prob.solve()

        return hp.value


class Filter(Params):

    def __init__(self):
```

```
        pass


    @staticmethod
    def impulse_response(h):

        h = np.hstack((h[:0:-1], h))

        return Filter.frequency_response(h)

    def frequency_response(h):

        H = np.fft.fft(h, Params.N)

        return h, H




class Display:


    def __init__(self, H, h, msg):

        self.msg = msg
        self.display_IPR(h)
        self.display_IPR_H(H)
        self.display_phase(H)

    def display_IPR(self, h):

        plt.figure()
        plt.stem(h)
        plt.plot(h, 'r--')
        plt.grid()
        plt.xlabel('$Samples$', fontsize = 16)
        plt.ylabel('$IPR$', fontsize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        #plt.savefig('FIR_LPF.png')
        plt.tight_layout()
        plt.show()

    def display_IPR_H(self, H):

        H = abs(H)
        plt.figure()
        plt.plot(Params.freq*Params.Fs, 20*np.log10(H), 'k', lw = 3)
        #plt.plot(Params.freq*Params.Fs, 0.5*np.ones(len(H)), 'r--', lw = 1)
        #plt.plot(Params.freq*Params.Fs, -0.5*np.ones(len(H)), 'r--', lw = 1)
        plt.grid()
        plt.xlabel('$Frequency\;\; [Hz]$', fontsize = 16)
        plt.ylabel('$20\;log\;|H(f)|\;\;[dB]$', fontsize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        plt.xlim(0,Params.Fs//2)
        #plt.savefig('FIR_LPF.png')
```

```
        plt.tight_layout()
        plt.show()



    def display_phase(self, H):

        plt.figure()
        plt.plot(Params.freq*Params.Fs, np.unwrap(np.angle(H)), 'k', lw = 3)
        #plt.plot(Params.freq*Params.Fs, 0.5*np.ones(len(H)), 'r--', lw = 1)
        #plt.plot(Params.freq*Params.Fs, -0.5*np.ones(len(H)), 'r--', lw = 1)
        plt.grid()
        plt.xlabel('$Frequency\;\; [Hz]$', fontsize = 16)
        plt.ylabel('$Phase\;\;[rad]$', fontsize = 16)
        matplotlib.rc('font', size=16)
        matplotlib.rc('axes', titlesize = 16)
        #plt.rcParams['figure.dpi'] = 300
        #plt.rcParams['savefig.dpi'] = 300
        plt.title(f'$FIR\;\;{self.msg}\; Pass\; Filter, \;\;\;\; n = {Params.filter_order}$')
        plt.xlim(0,Params.Fs//2)
        #plt.savefig('FIR_LPF.png')
        plt.tight_layout()
        plt.show()




if __name__ == '__main__':

    msg = 'High'
    Params()
    opt = Opt()
    h = opt._run()
    h, H = Filter.impulse_response(h)

    Display(H, h, msg)
```

Fig. 4 shows the result of the code for the HPF design.

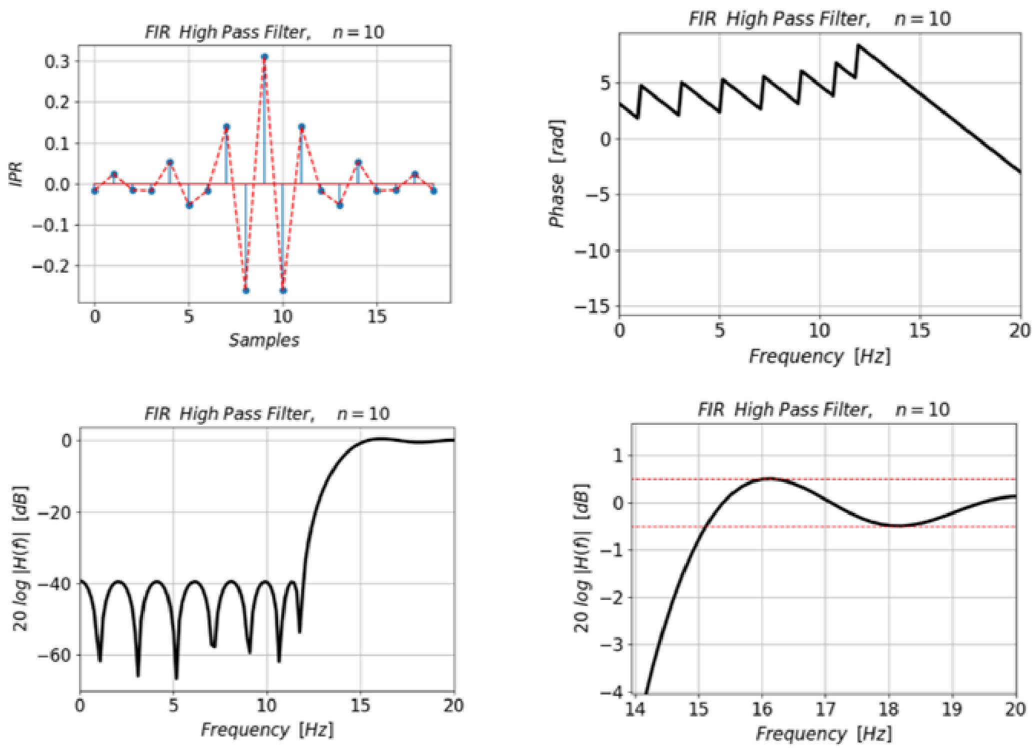

Fig. 4. The result for the HPF design.

Shahrokh Hamidi was born in Iran, in 1983. He received his B.Sc., M.Sc., and Ph.D. degrees all in Electrical and Computer Engineering. He is with the faculty of Electrical and Computer Engineering at the University of Waterloo, Waterloo, Ontario, Canada. His current research areas include statistical signal processing, mmWave imaging, Terahertz imaging, image processing, system design, multi-target tracking, wireless and digital communication systems, machine learning, optimization, and array processing.