

yig_learning

June 7, 2022

0.1 Initialize HW and constants

```
[1]: import sys
sys.path.append("gpib_instrument_control")
import hp_3478a
import hp_8700_series_vna
import numpy as np
import time
import yig_controller_test
import matplotlib.pyplot as plt
import scipy.io as sio
import skrf.network
import yig_controller_test

plt.rcParams['figure.figsize'] = [15, 10]

#Instruments and devices
yigControllerPort='/dev/ttyUSB0'
vna = hp_8700_series_vna.Hp8753A()
curMeter = hp_3478a.Hp3478A()
yc = yig_controller_test.YigController(yigControllerPort)
```

Waiting for init... Done

```
[2]: #Constants and auxillary functions
fMin=np.array([0.6, 1, 2, 4, 8, 12])*1e9
fMax=np.array([1, 2, 4, 8, 12, 18])*1e9
yigDriver=yc.yiga
switch=yc.switchA

hardwareDescription='initial test of version 3 of pcb. Strapped mux2b since it was unconnected. Fan mounted correctly. Replaced u15 opamp that had gone bad.
..
hardwareRevision='3.1'

#reset yig filters
```

```

yc.yigB.set(0,0)
yc.yigA.set(6,0)
yc.yigB.set(7,0)

if yigDriver==yc.yigA:
    driverChannel='A'
if yigDriver==yc.yigB:
    driverChannel='B'

def revFileName(baseName):
    global hardwareRevision
    return '%s_channel_%s_rev_%s.mat'%(baseName, driverChannel,hardwareRevision)

def saveData(baseName, baseData):
    global hardwareDescription
    global hardwareRevision
    dataToSave=baseData;
    dataToSave['hardwareDescription']=hardwareDescription
    sio.savemat(revFileName(baseName), dataToSave)

def loadData(baseName):
    global hardwareRevision
    return sio.loadmat(revFileName(baseName))

```

0.2 Measure current tuning ranges

```

[3]: wr = np.linspace(-32768, 32767, 32, dtype=np.int16)
yigChs = range(6)
#zeros=[0, 0, 0, 0, 0, 0]
zeros = []
channels=[]
currentMtx = None;
for c in yigChs:
    print(c)
    yigDriver.set(c, wr[0])
    time.sleep(5);
    current=[]
    for w in wr:
        yigDriver.set(c, w);
        time.sleep(0.1)
        for i in range(3):
            curMeter.readValue()
            current.append(curMeter.readValue())
    channels.append(current)
    currentMtx = yig_controller_test.stackVector(currentMtx, current)

```

```

yigDriver.set(6, 0)
time.sleep(3)
zeros.append(curMeter.readValue())

saveData('current_sweep', {'current':currentMtx, 'yigChs':yigChs, ↴
    'tuningWordRange': wr});

```

0
1
2
3
4
5

[4]:

```

import scipy.io as sio
saveData('yig_filter_driver_current_data', {'zeros':zeros, 'channels':channels, ↴
    'wr':wr})

```

[5]:

```

d=loadData('yig_filter_driver_current_data')
zeros=d['zeros'][0]
channels=d['channels']
wr=d['wr'][0]

```

[6]:

```
print(zeros)
```

[-1.4e-05 -1.6e-05 -1.4e-05 -1.4e-05 -1.6e-05 -1.5e-05]

[7]:

```

charr = np.array(channels)
def moving_average(x, w):
    return np.convolve(x, np.ones(w), 'valid') / w

class FilterParameters:
    def __init__(self, lc, hc, lw, hw):
        self.lc=lc
        self.hc=hc
        self.lw=lw
        self.hw=hw

fParams=[]
for i in range(len(zeros)):
    #avgd = moving_average(np.diff(charr[i,:]), 10)
    avgd = np.diff(charr[i,:])
    plt.figure(1)
    plt.plot(wr[1::],avgd)
    plt.grid(True)
    plt.figure(2)

```

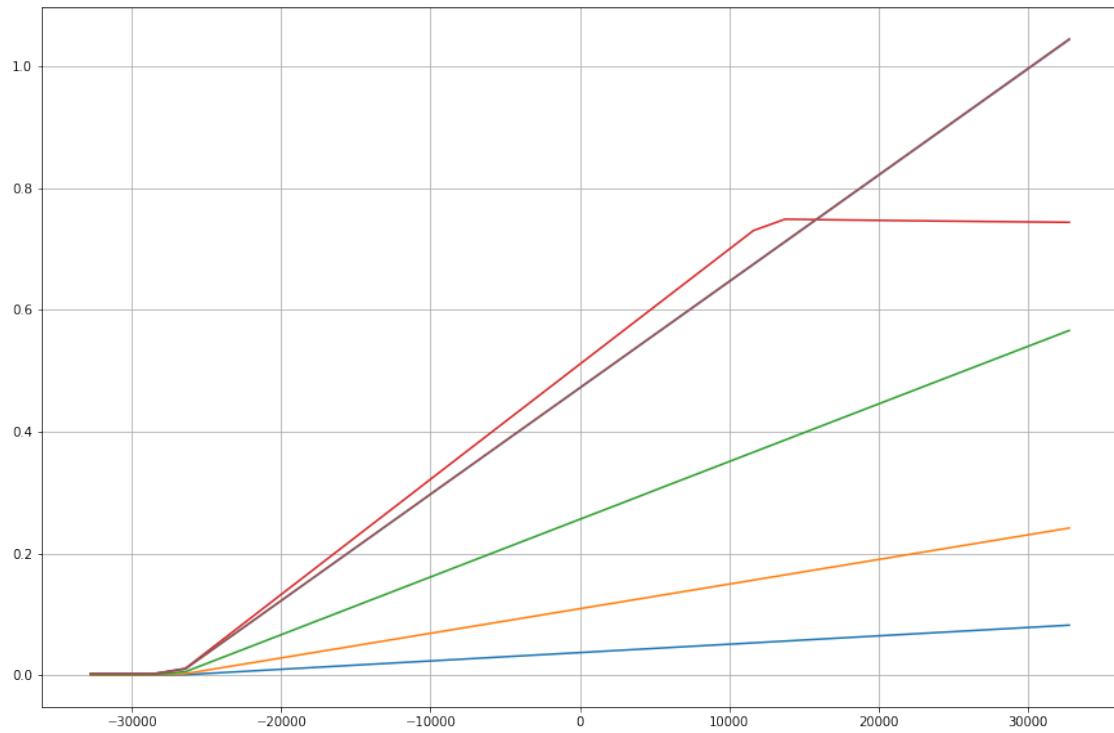
```

plt.plot(wr, charr[i,:])
plt.grid(True)
idxs = np.where(avgd>0.0002);
mii=idxs[0][0]
mai=idxs[0][-1]
zc = zeros[i]
mic=charr[i,mii]-zc
mac=charr[i,mai]-zc
#print(idxs[0])
print("Filter %i has current from %f to %f [A] in control word range %d -_"
    "->%d [LSB]"%(i, mic, mac, wr[mii], wr[mai]))
fParams.append(FilterParameters(mic, mac, wr[mii], wr[mai]))

```

Filter 0 has current from 0.000169 to 0.079196 [A] in control word range -28540
 - 30652 [LSB]
 Filter 1 has current from 0.000514 to 0.233025 [A] in control word range -28540
 - 30652 [LSB]
 Filter 2 has current from 0.001212 to 0.546064 [A] in control word range -28540
 - 30652 [LSB]
 Filter 3 has current from 0.002421 to 0.730184 [A] in control word range -28540
 - 11626 [LSB]
 Filter 4 has current from 0.002254 to 1.006536 [A] in control word range -28540
 - 30652 [LSB]
 Filter 5 has current from 0.002284 to 1.007675 [A] in control word range -28540
 - 30652 [LSB]





0.3 Coarse tuning

```
[8]: vna.setStartFrequency(130e6)
vna.setStopFrequency(20e9)
vna.setPoints(401)

fRanges=[]
for i in range(len(zeros)):
    #print("Filter", i )
    fp = fParams[i]
    switch.set(i+1)
    yigDriver.set(6, 0)
    fax = vna.frequencies()
    base = vna.readSParameter('S21')
    plt.plot(fax, 20*np.log10(np.abs(base)))
    sr = np.linspace(fp.lw, fp.hw, 8)
    freqs=[]

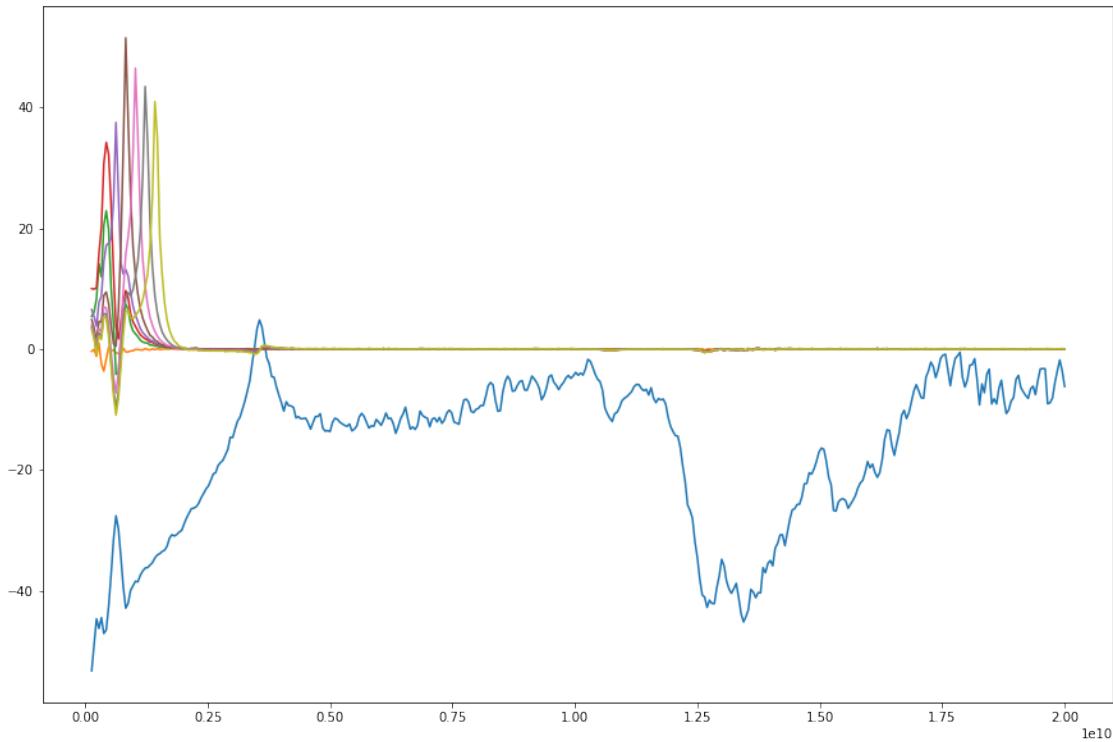
    for w in sr:
        yigDriver.set(i, int(w))
        time.sleep(1)
```

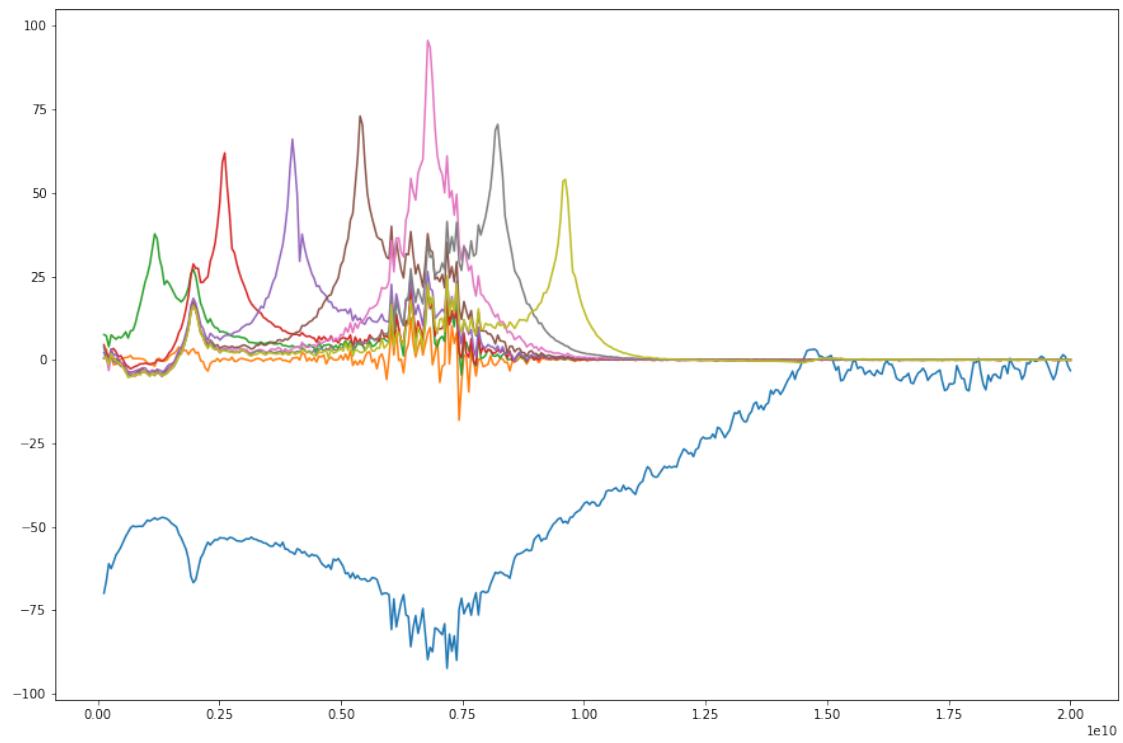
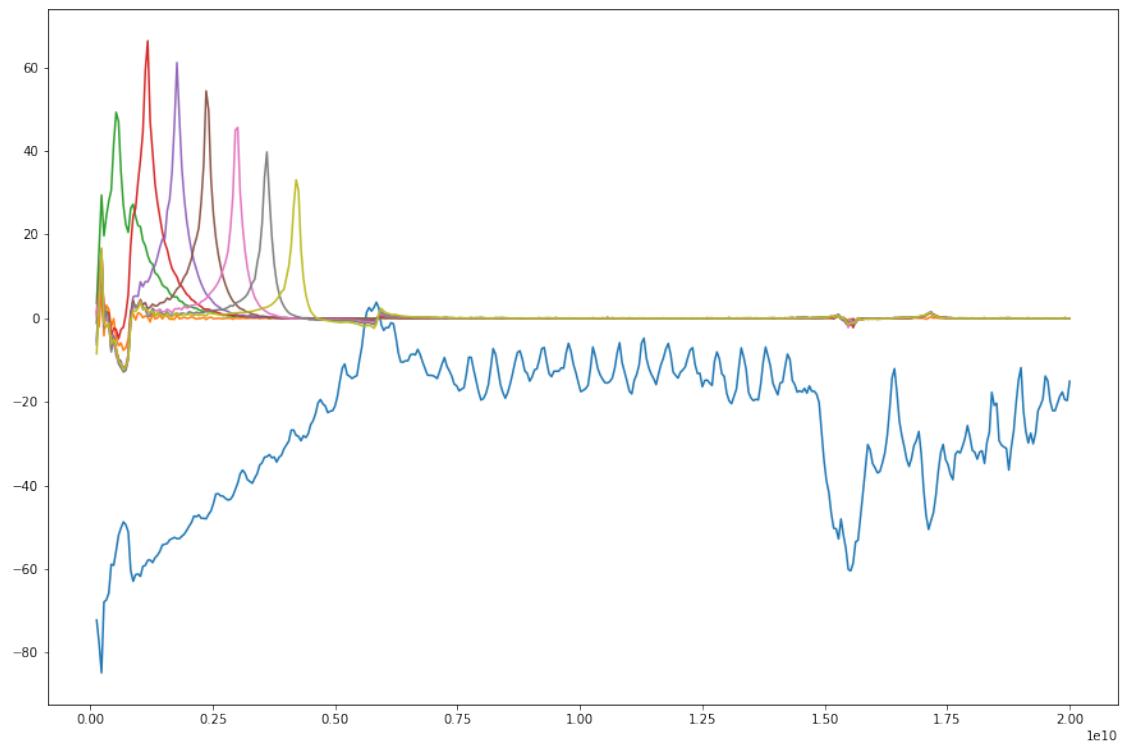
```

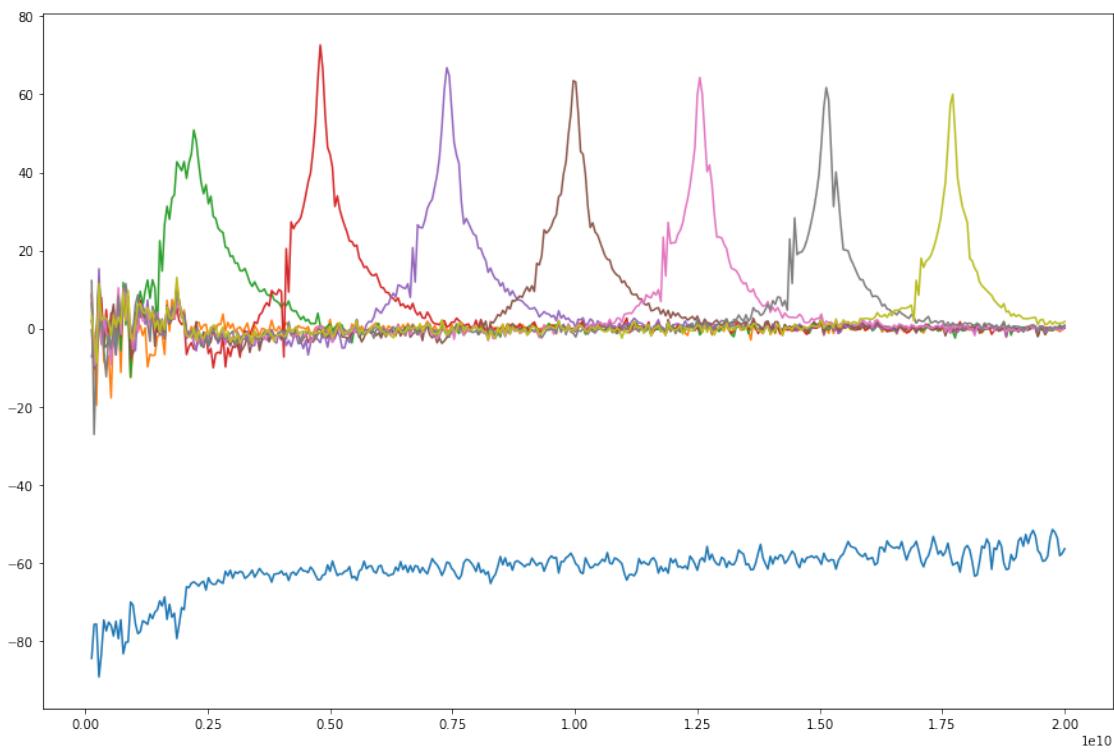
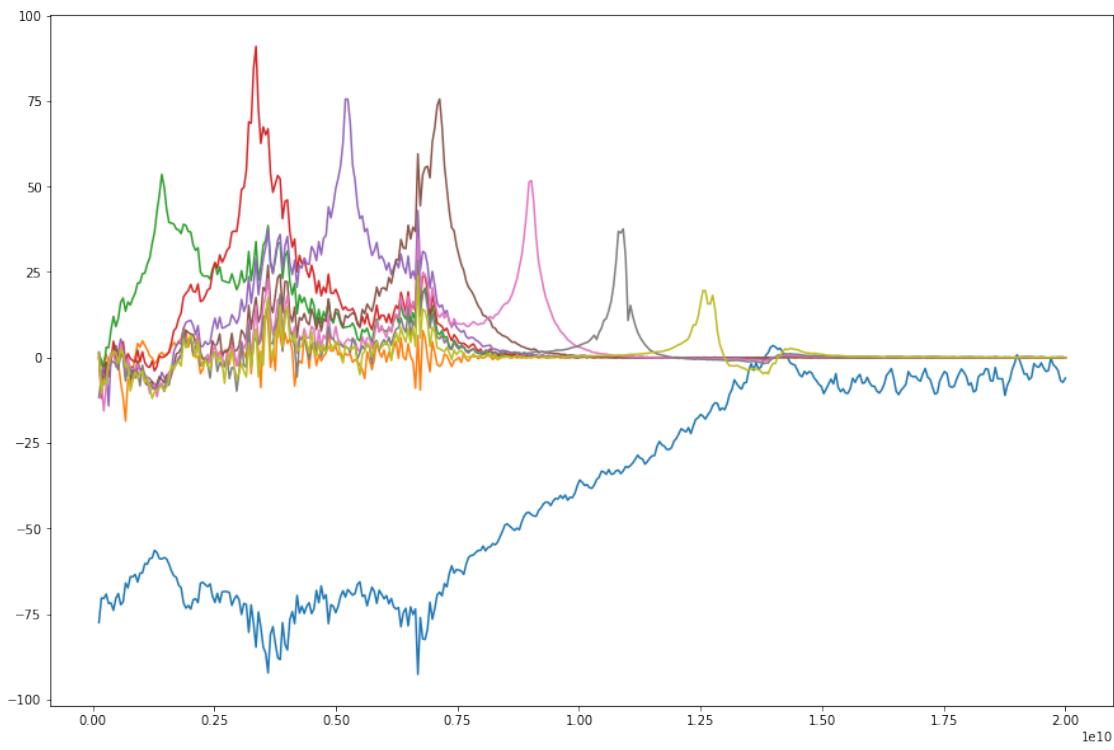
t = vna.readSParameter('S21')

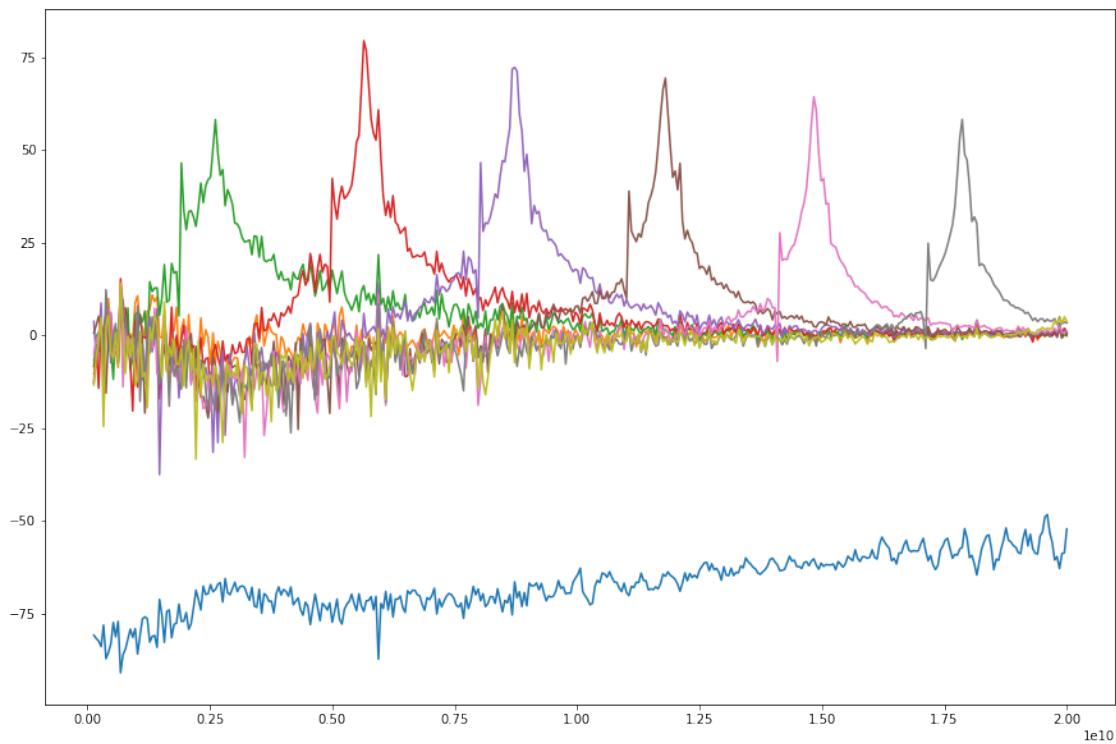

```

0.279025 to 1.421550 [GHz]
0.229350 to 4.203350 [GHz]
1.173175 to 9.617925 [GHz]
1.421550 to 10.909475 [GHz]
1.868625 to 17.714950 [GHz]
0.676425 to 17.863975 [GHz]









<Figure size 1080x720 with 0 Axes>

```
[9]: #Get calibration data
import os.path

if not os.path.exists('cal_through.s2p'):
    calPar=vna.getHighResolutionNetwork(130e6, 20e9, 1e6)
    calPar.plot_s_db()
    calPar.write_touchstone('cal_through.s2p')
```

```
[10]: import skrf.network

calPar=skrf.network.Network('cal_through.s2p')
d=loadData('yig_filter_driver_current_data')
zeros=d['zeros'][0]
channels=d['channels']
wr=d['wr'][0]

d2=loadData('yig_filter_driver_coarse_frequency')
fRanges=d2['fRanges']
#print(fRanges)
```

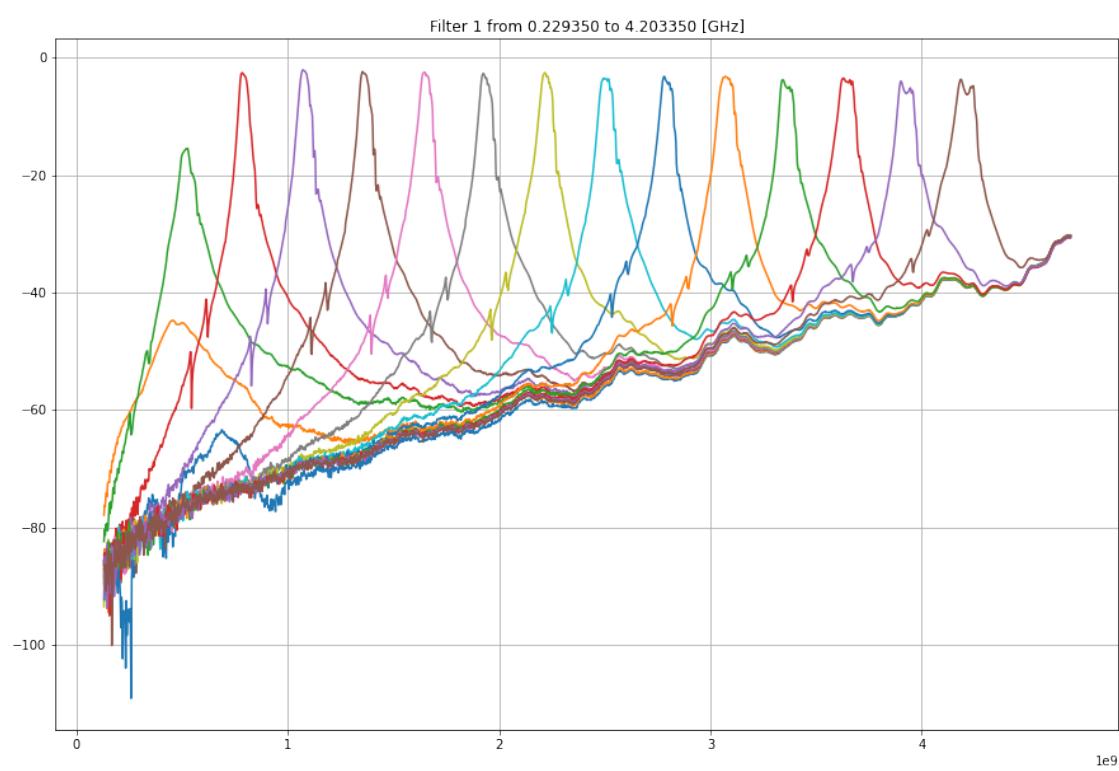
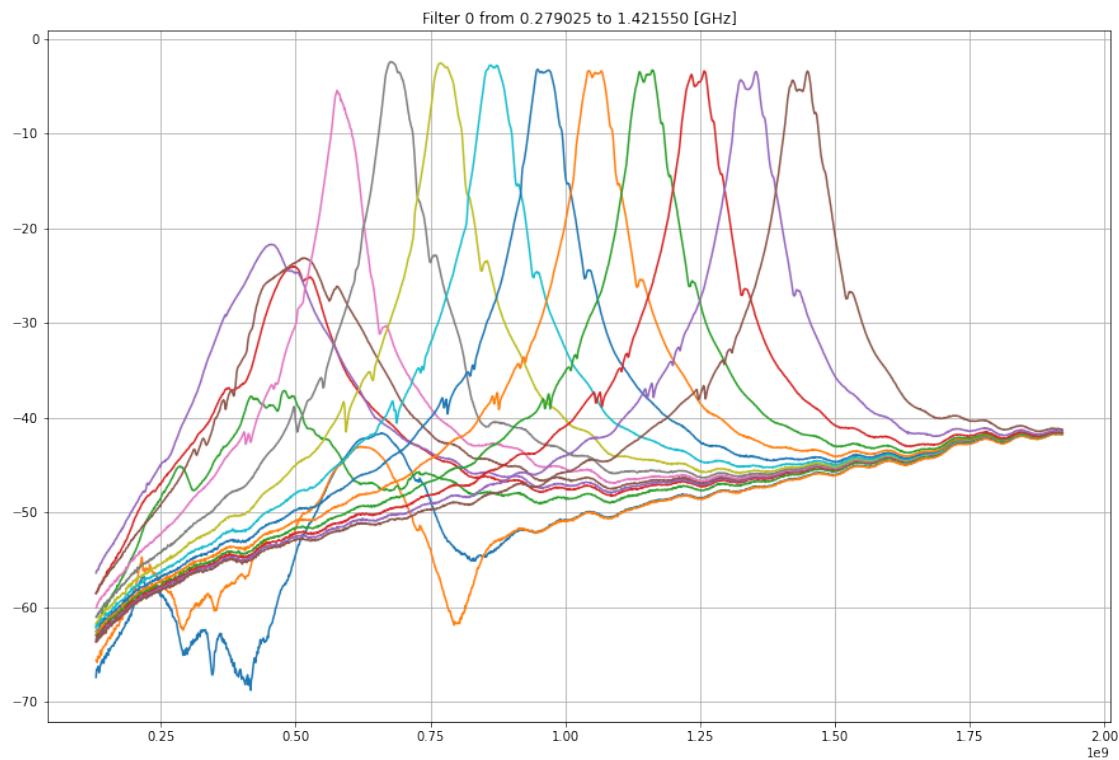
```

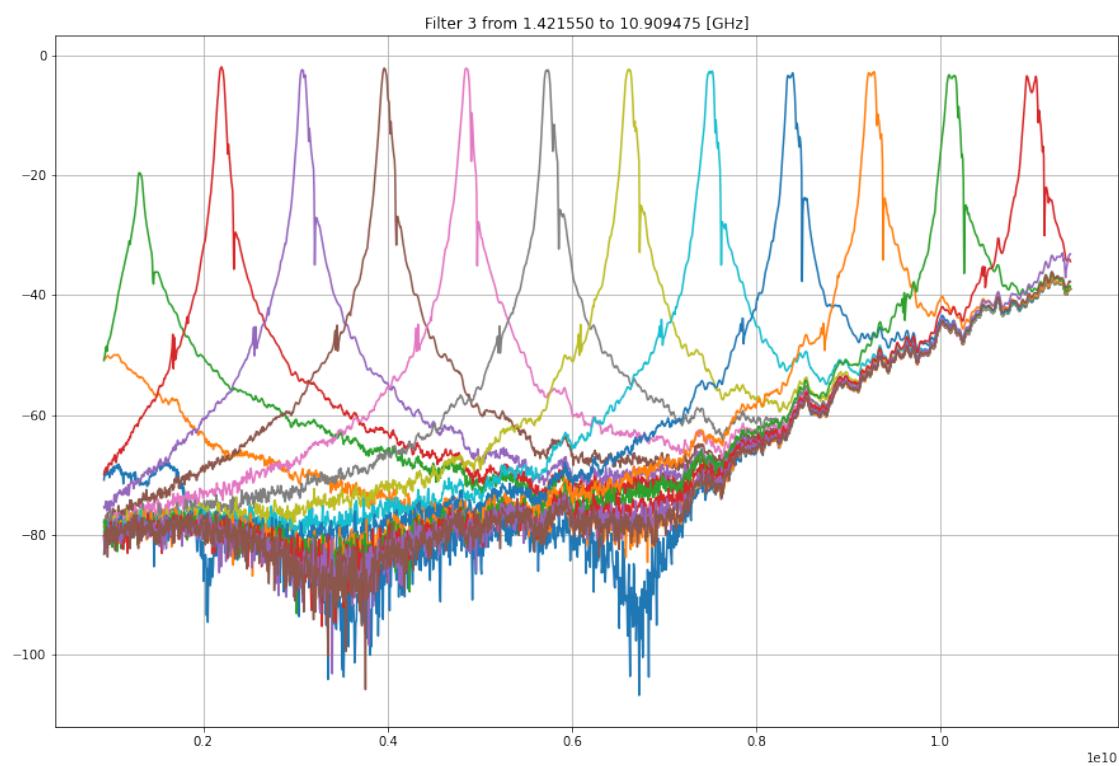
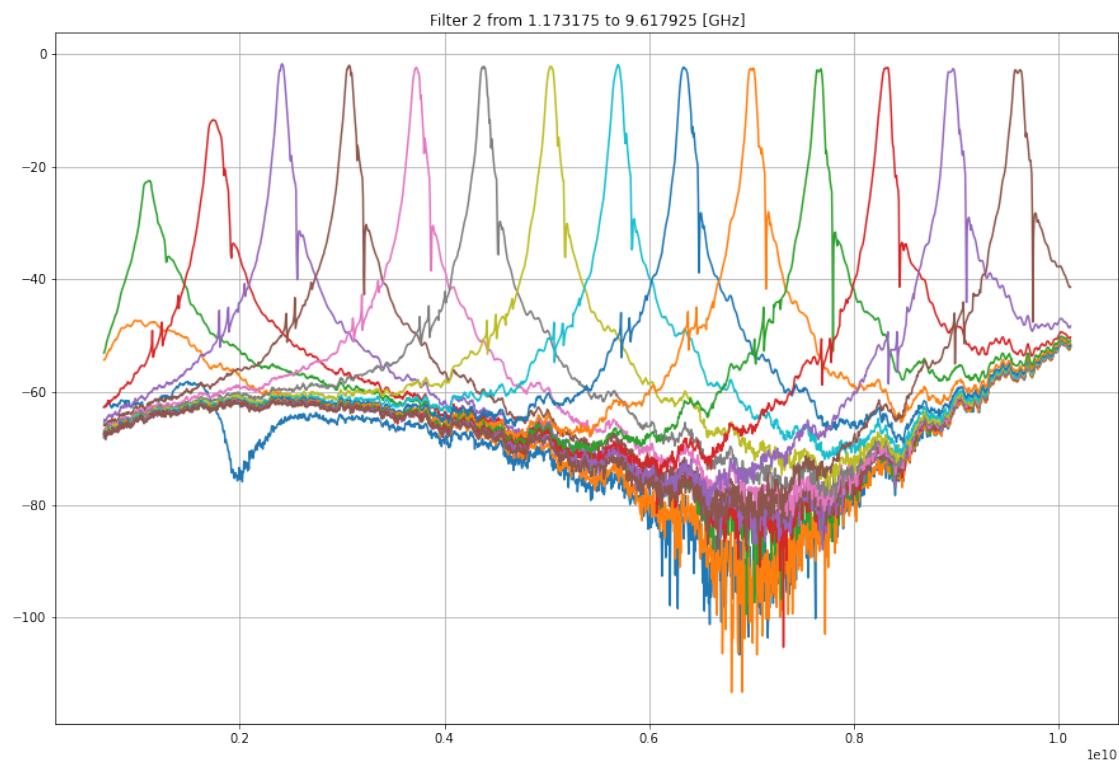
#print(channels)
#print(wr)

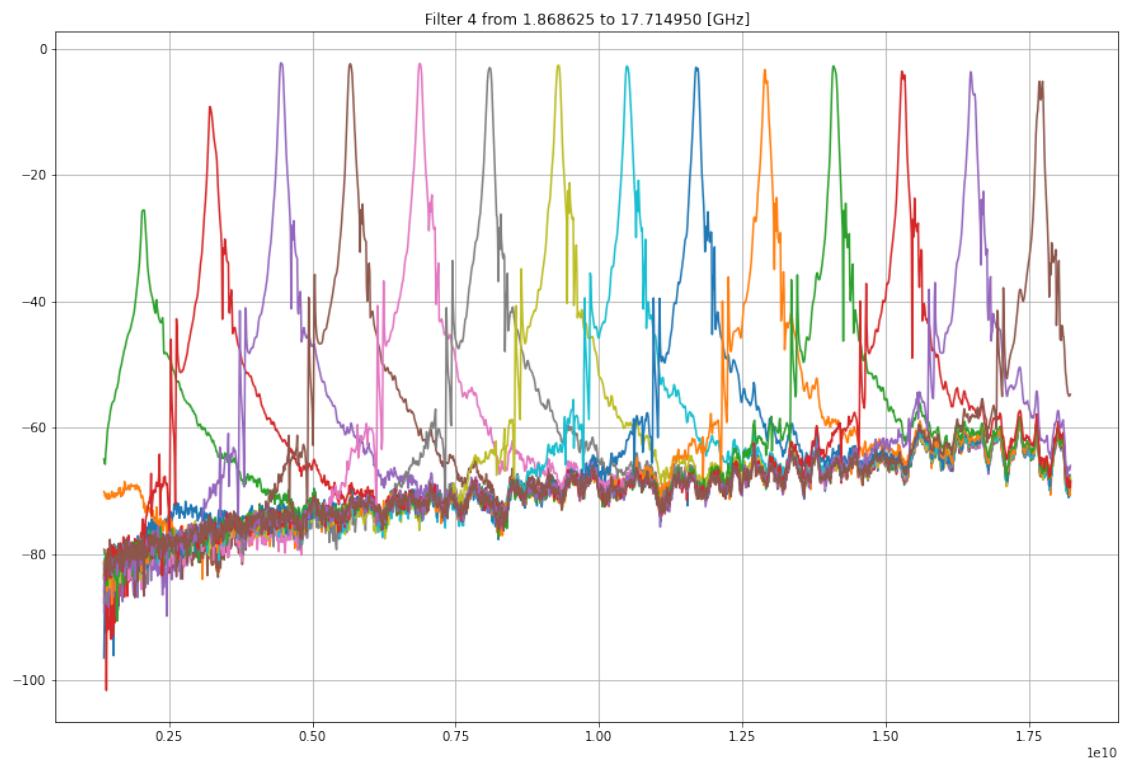
def inter(fd, fa, a):
    afd=np.interp(fd, fa, a)
    return afd

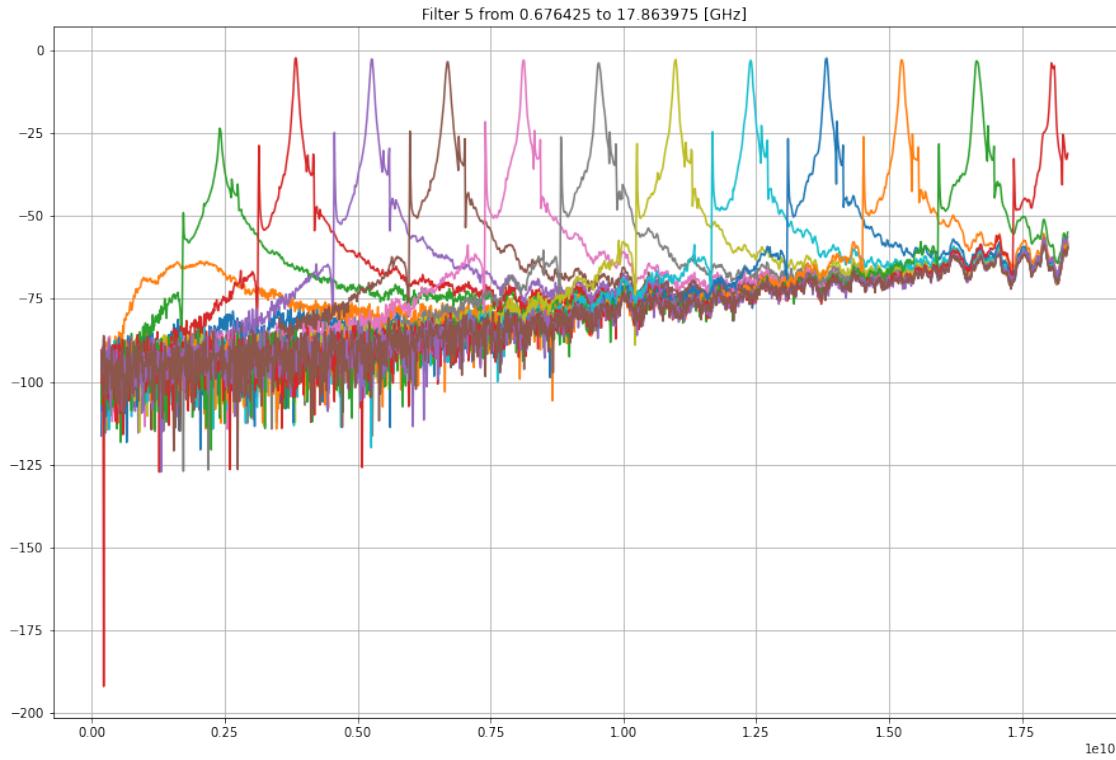
filterMeas={}
for i in range(len(zeros)):
    switch.set(i+1)
    fsta=fRanges[i,0]-500e6
    fsta=np.max((fsta, 130e6))
    fsto=fRanges[i,1]+500e6
    fsto=np.min((fsto, 20e9))
    vna.setStartFrequency(fsta)
    vna.setStopFrequency(fsto)
    vna.setPoints(1601)
    vwr = np.linspace(fParams[i].lw, fParams[i].hw, 16)
    plt.figure()
    plt.title("Filter %d from %f to %f [GHz]"%(i, fRanges[i,0]/1e9, fRanges[i,1]/1e9))
    thisFilterMeas={}
    thisFilterMeas['words']=[]
    thisFilterMeas['traces']=[]
    thisFilterMeas['frequencies']=[]
    thisFilterMeas['calData']=[]
    for w in vwr:
        yigDriver.set(i, int(w))
        time.sleep(0.5)
        tr = vna.readSParameter('S21')
        fr = vna.frequencies()
        cai=inter(fr, calPar.f, np.abs(calPar.s[:,1,0]))
        #nf, npar = norm(calPar.f, np.abs(calPar.s[:,0,1]), fr, np.abs(tr))
        #plt.plot(nf, npar)
        plt.plot(fr, 20*np.log10(np.abs(tr)/cai))
        thisFilterMeas['words'].append(w)
        thisFilterMeas['traces'].append(tr)
        thisFilterMeas['frequencies'].append(fr)
        thisFilterMeas['calData'].append(cai)
        #plt.plot(fr, cai)
    filterMeas[i] = thisFilterMeas
    plt.grid(True)
    plt.show()
    yigDriver.set(6, 0)

```









```
[11]: words = []
frequencies = []
calData = []
traces = []
for i in filterMeas:
    f = filterMeas[i]
    words.append(f['words'])
    frequencies.append(f['frequencies'])
    calData.append(f['calData'])
    traces.append(f['traces'])

saveData('coarse_tuning_data', {'words':words, 'frequencies':frequencies,
                                'calData':calData, 'traces':traces})
```

```
[12]: kvec=[]
mvec=[]
for i in range(len(zeros)):
    thisFilter=filterMeas[i]
    words = thisFilter['words']
    traces = thisFilter['traces']
    frequencies = thisFilter['frequencies'][0]
    cals = thisFilter['calData']
    fildata=[]
```

```

for trace, w, cal in zip(traces, words, cals):
    tDat = 20*np.log10(np.abs(trace)/np.abs(cal))
    i=np.argmax(tDat)
    if(tDat[i]>-5):
        fildat.append((int(w), frequencies[i]/1e6, tDat[i]))

j=int(len(fildat)/5.0)
jj=int(4*len(fildat)/5.0)
m=int((j+jj)/2)
dw=fildat[jj][0]-fildat[j][0]
df=fildat[jj][1]-fildat[j][1]
filK=df/dw
ms=[]
for m in range(len(fildat)):
    filM=fildat[m][1]-fildat[m][0]*filK
    ms.append(filM)
filM=np.mean(ms)
print("Filter parameter for filter %d is k %.3f [MHz/LSB] m is %.
      ↵3f[LSB]"%(i, filK, filM))
kvec.append(filK)
mvec.append(filM)

coarseFilter={'k':kvec, 'm':mvec}
saveData('coarse_filter_parameters', coarseFilter)

```

Filter parameter for filter 1178 is k 0.025 [MHz/LSB] m is 689.895[LSB]
 Filter parameter for filter 1419 is k 0.072 [MHz/LSB] m is 1989.367[LSB]
 Filter parameter for filter 1517 is k 0.167 [MHz/LSB] m is 4531.730[LSB]
 Filter parameter for filter 1569 is k 0.331 [MHz/LSB] m is 8961.722[LSB]
 Filter parameter for filter 1554 is k 0.305 [MHz/LSB] m is 8356.187[LSB]
 Filter parameter for filter 1561 is k 0.361 [MHz/LSB] m is 9871.338[LSB]

0.4 Measure drift

```
[16]: import time
coarseFilter=loadData('coarse_filter_parameters')
mvec=coarseFilter['m'][0]
kvec=coarseFilter['k'][0]

def computeWord(fTarget, k, m):
    return (fTarget/1e6-m)/k

s21Drift=None
currentDrift=None
```

```

dataDict={}

def measureDrifts():
    for a in range(3):
        curMeter.readValue()
    t0=time.time()
    t00=t0
    fmax=[]
    curr=[]
    phase=[]
    sFreq = None
    sMat = None
    timeVec = []
    for j in range(10):
        t0=t00+10*j
        while time.time() < t0:
            pass
        f=vna.frequencies()
        curr.append(curMeter.readValue())
        t=vna.readSParameter('S21')
        fmax.append(f[np.argmax(np.abs(t))])
        phase.append(np.angle(t[int(len(t)/2)]))
        sFreq = f;
        sMat = yig_controller_test.stackVector(sMat, t)
        timeVec.append(t0)
    return timeVec, f, curr, phase, sMat, fmax

for i in range(6):
    print("Measuring filter", i)
    k = kvec[i]
    m = mvec[i]
    switch.set(i+1)
    keyBase = 'yigFilter%d'%(i)
    for tf, keySub in zip([fMin[i], fMax[i], fMin[i]], ['Low', 'High', ↴'LowAgain']):
        yigDriver.set(i, int(computeWord(tf, k, m)))
        vna.setStartFrequency(tf-100e6)
        vna.setStopFrequency(tf+100e6)
        timeVec, f, curr, phase, sMat, fmax = measureDrifts()
        dataDict[keyBase+keySub+'Current']=curr
        dataDict[keyBase+keySub+'Time']=timeVec
        dataDict[keyBase+keySub+'Frequency']=f
        dataDict[keyBase+keySub+'Phase']=phase
        dataDict[keyBase+keySub+'SMatrix']=sMat
        dataDict[keyBase+keySub+'MinimumLossFrequency']=fmax

```

```
saveData('filter_drift', dataDict)
```

```
Measuring filter 0
Measuring filter 1
Measuring filter 2
Measuring filter 3
Measuring filter 4
Measuring filter 5
```

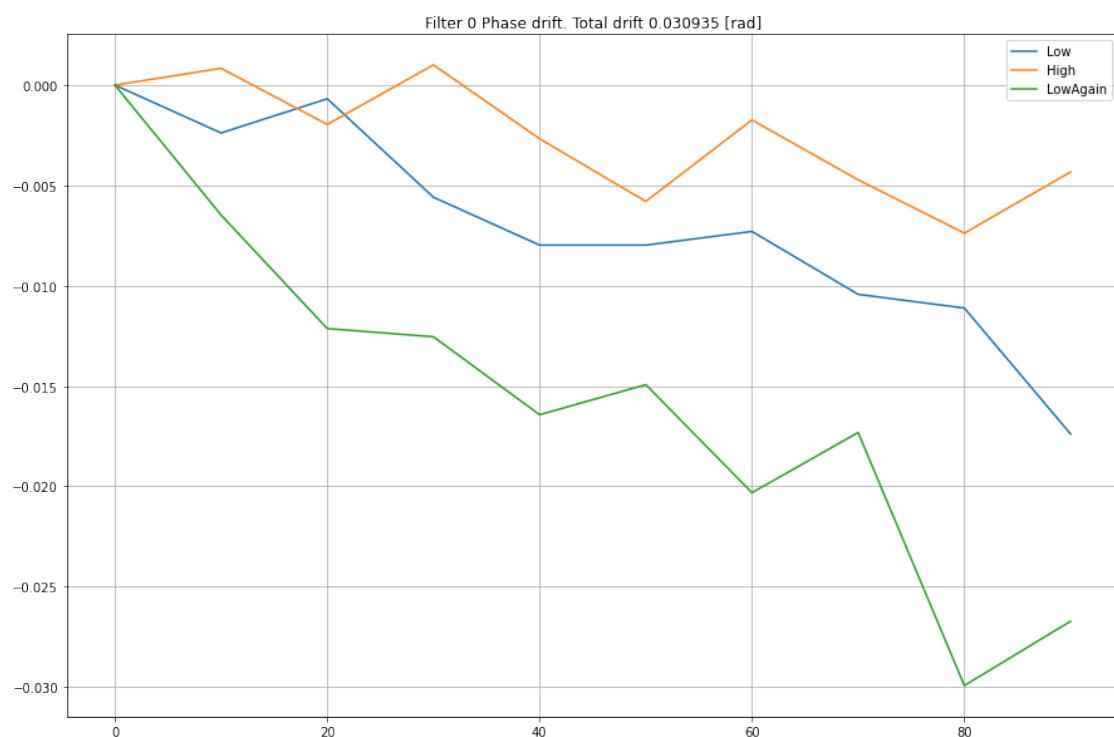
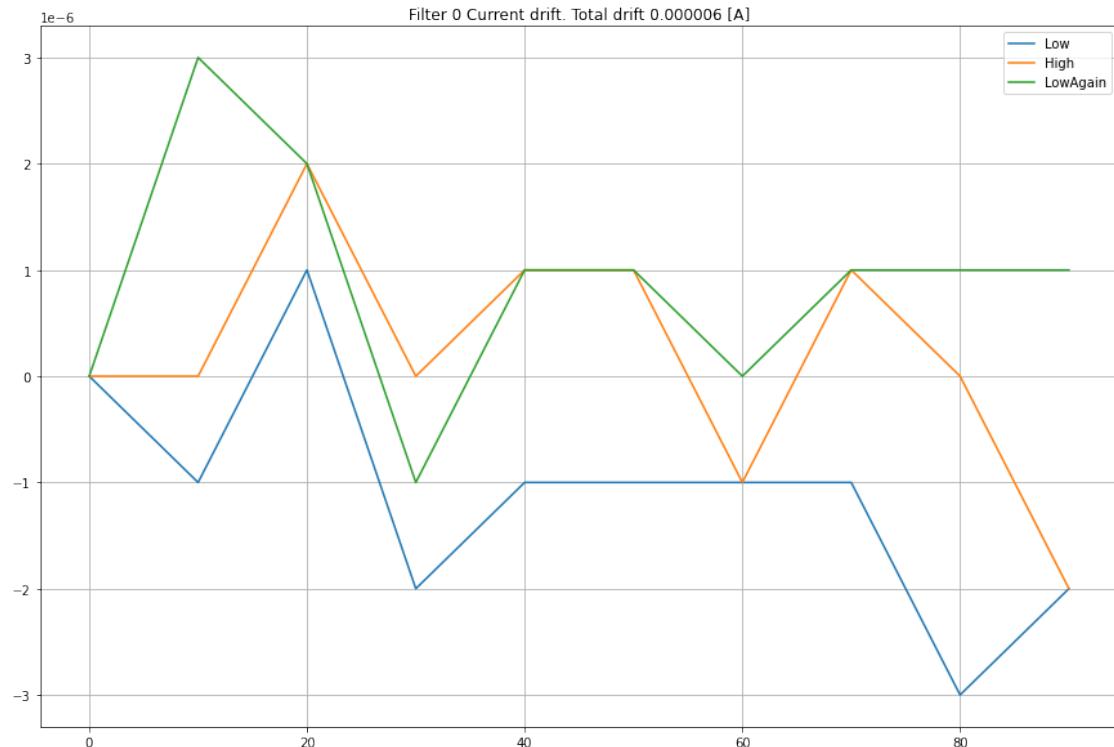
```
[17]: dd = loadData('filter_drift')

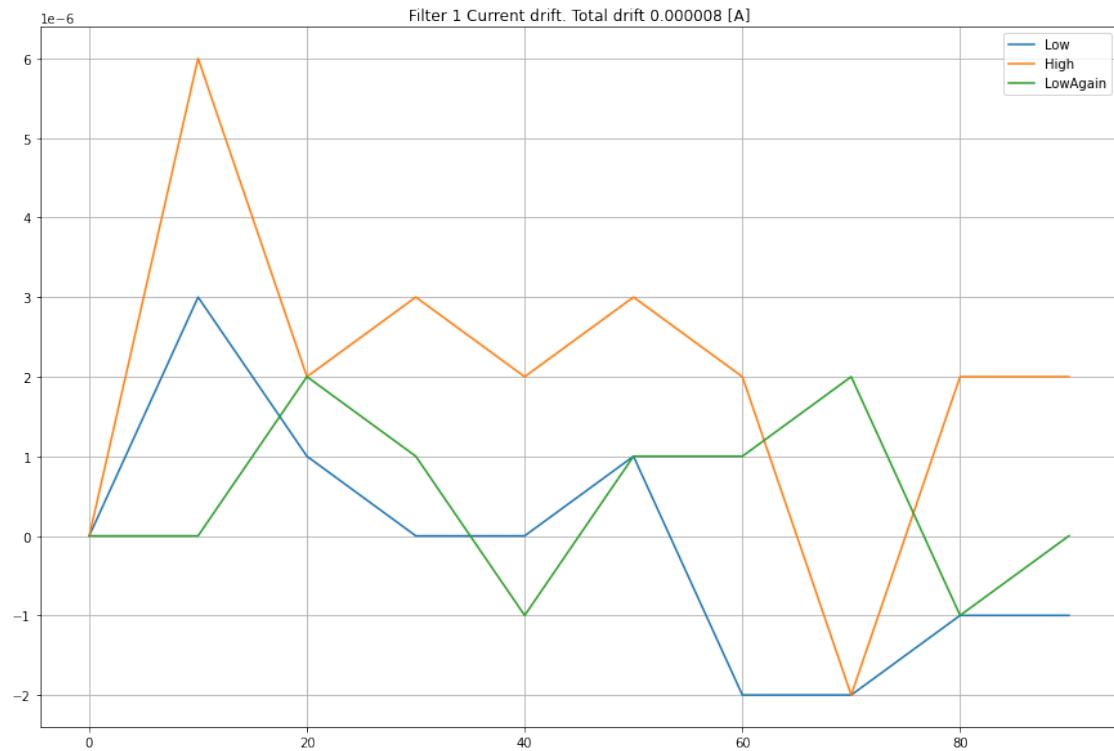
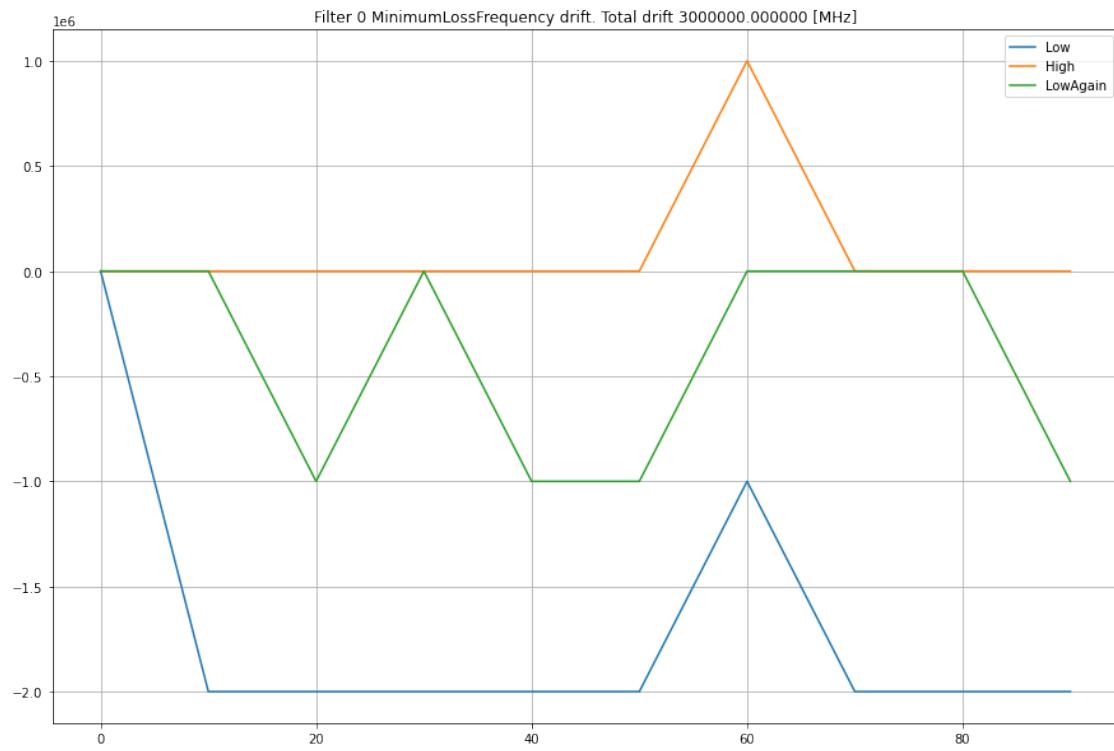
def plotParameter(i, par, unit):
    keyBase = 'yigFilter%d'%(i)
    plt.figure()

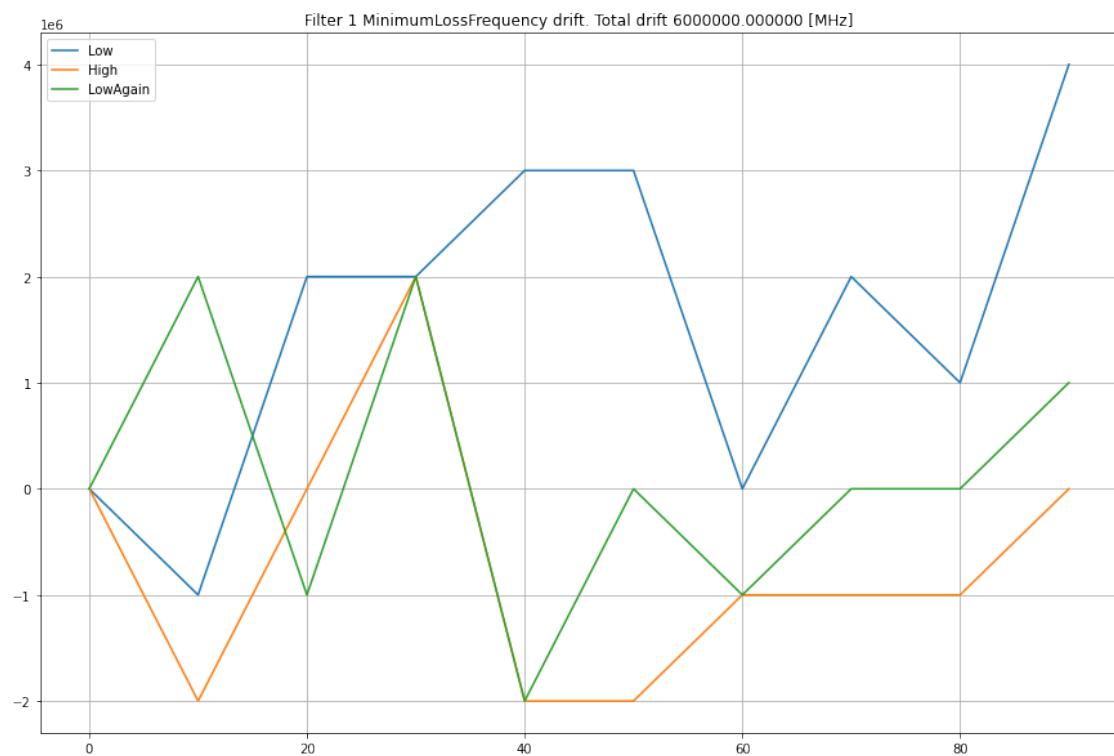
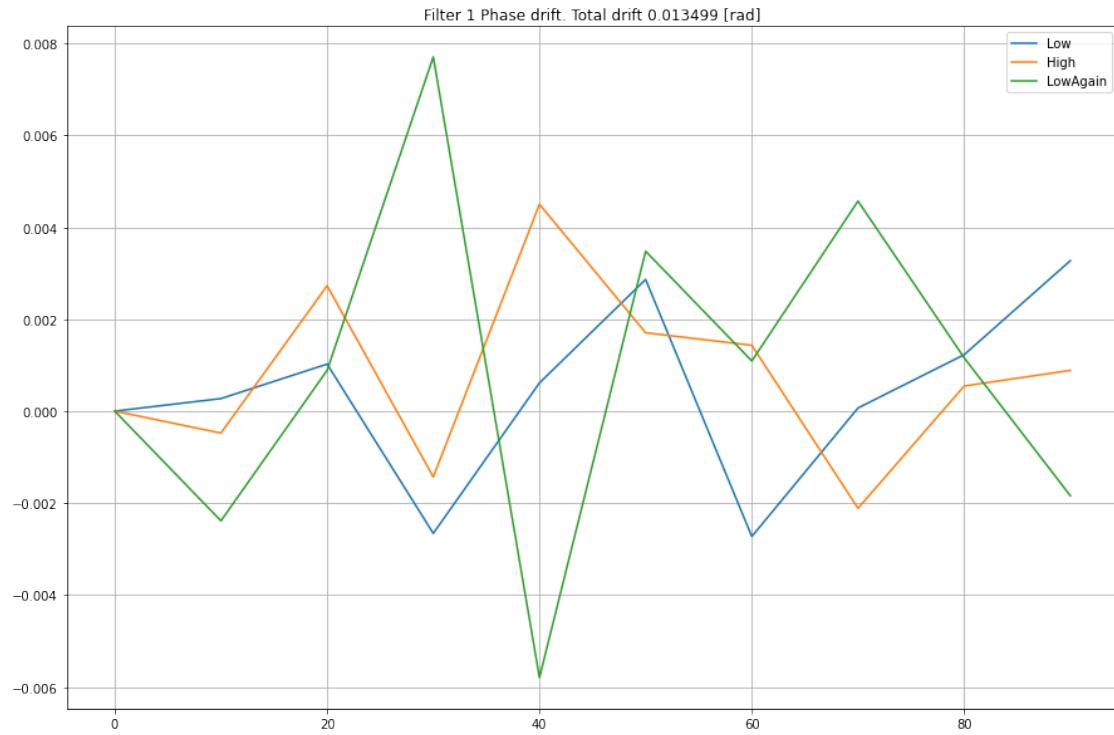
    minVal=None
    maxVal=None
    for keySub in ['Low', 'High', 'LowAgain']:
        data=dd[keyBase+keySub+par] [0]
        timeData=dd[keyBase+keySub+'Time'] [0]
        timeData=timeData-timeData[0]
        data=data-data[0]
        plt.plot(timeData, data, label=keySub)
        if minVal is None:
            minVal=min(data)
        else:
            minVal = min(minVal, min(data))
        if maxVal is None:
            maxVal=max(data)
        else:
            maxVal=max(maxVal, max(data))

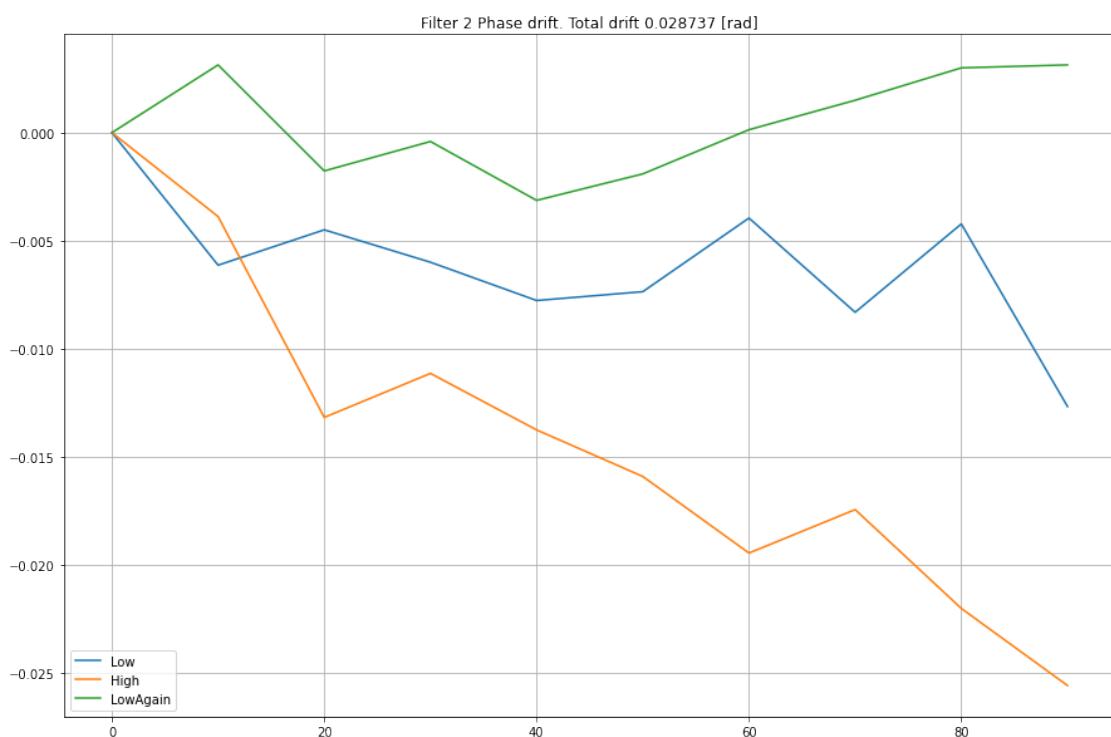
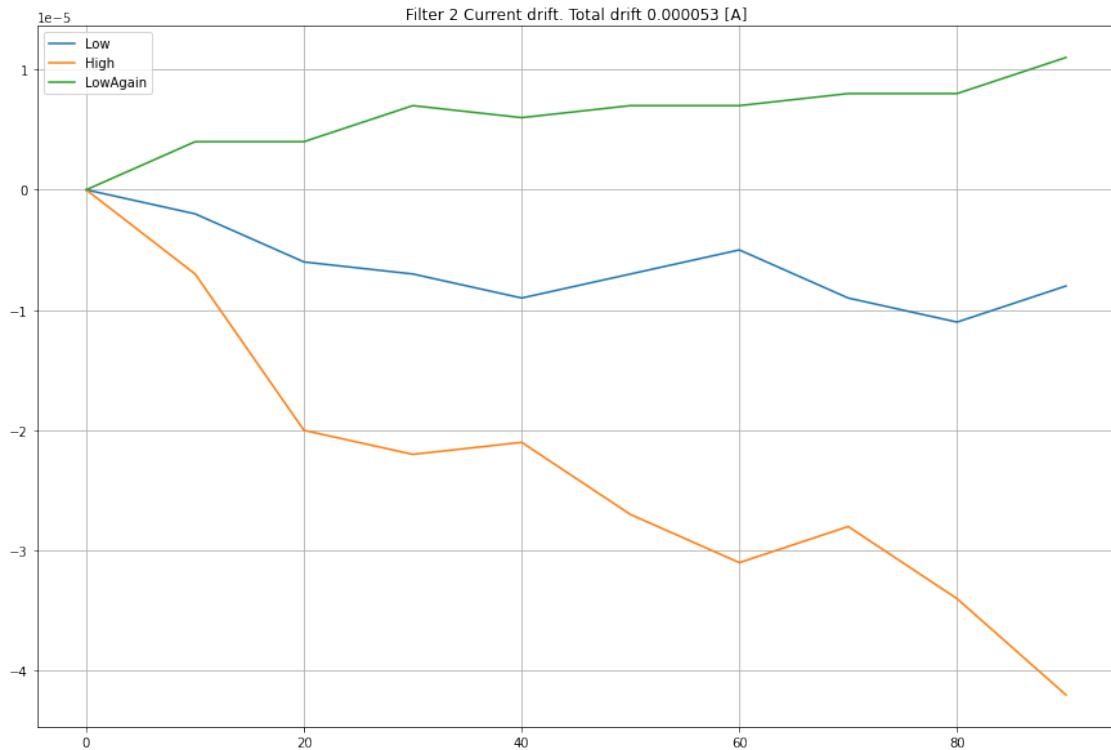
    plt.title("Filter %d %s drift. Total drift %f [%s]"%(i, par, maxVal-minVal, unit))
    plt.grid(True)
    plt.legend()
    plt.show()

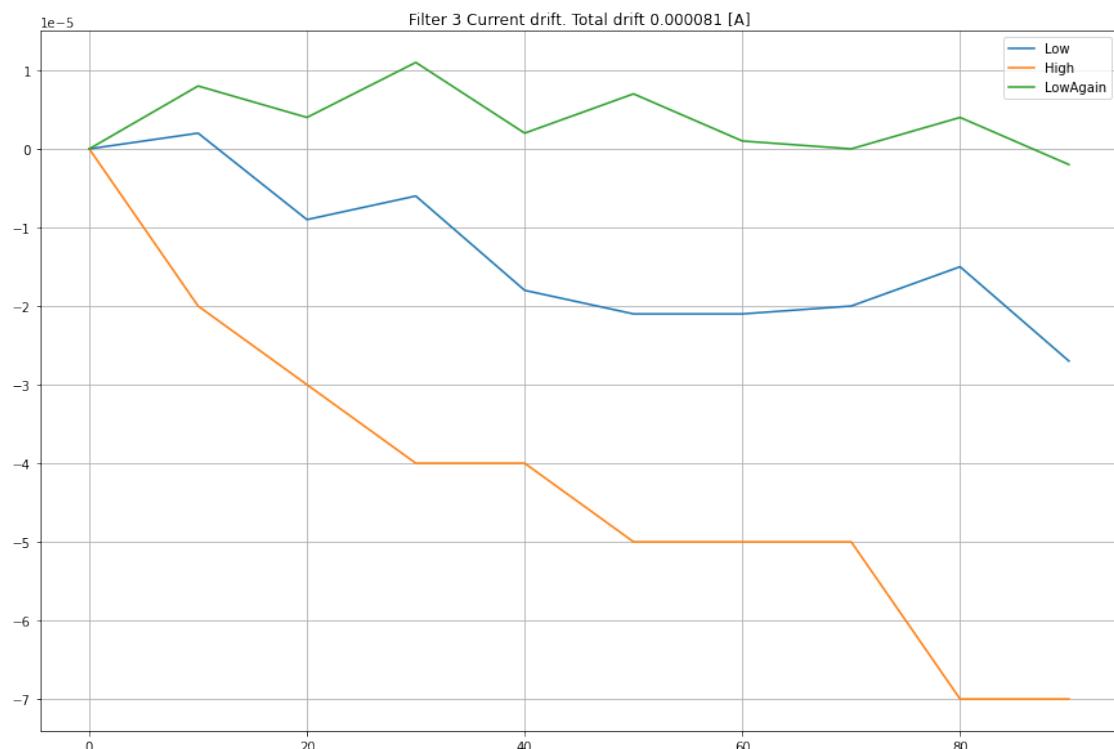
for i in range(6):
    plotParameter(i, 'Current', 'A')
    plotParameter(i, 'Phase', 'rad')
    plotParameter(i, 'MinimumLossFrequency', 'MHz')
```

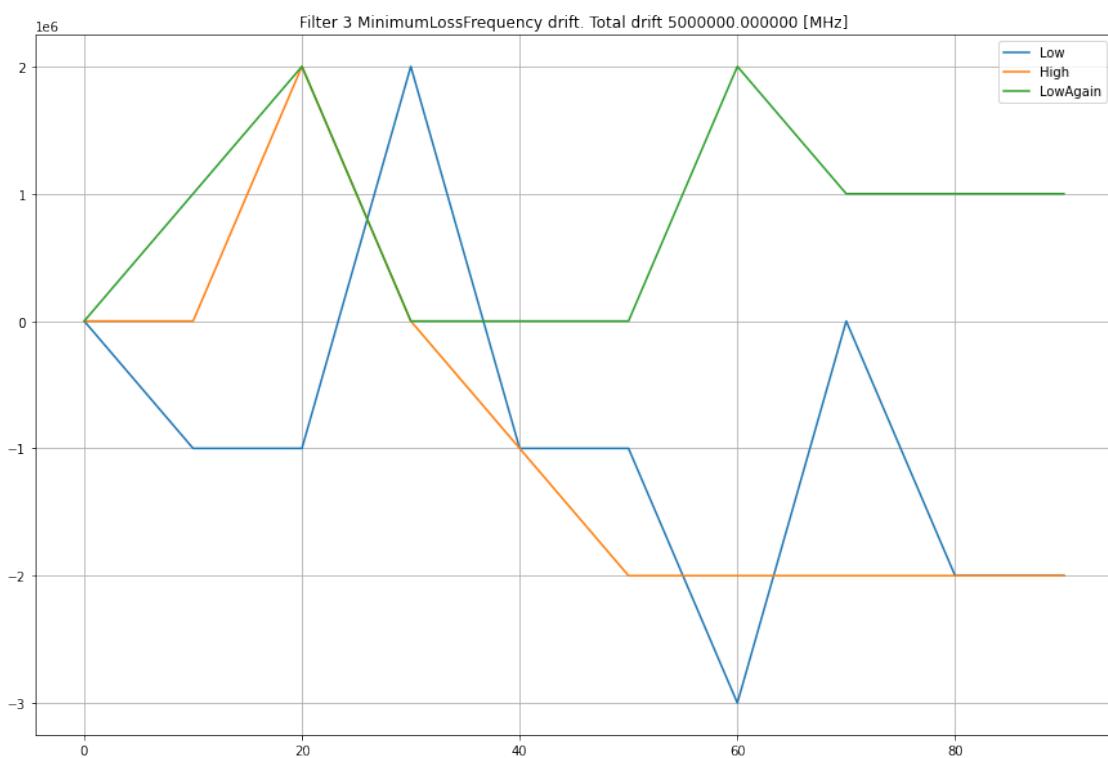
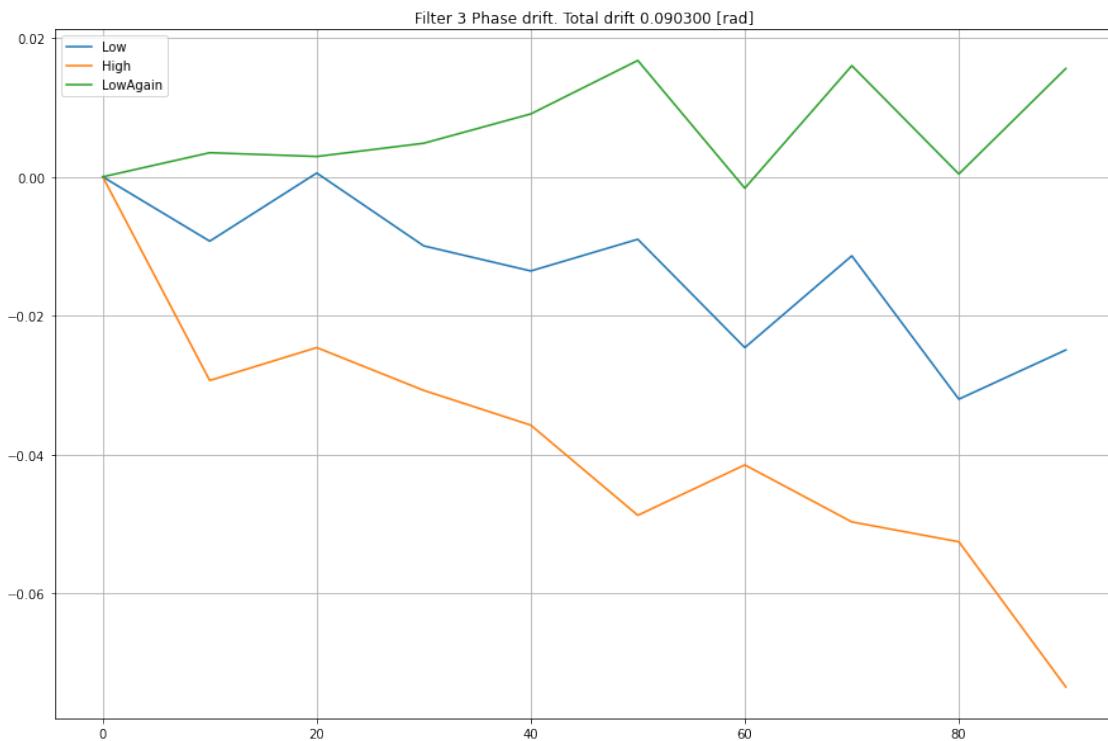


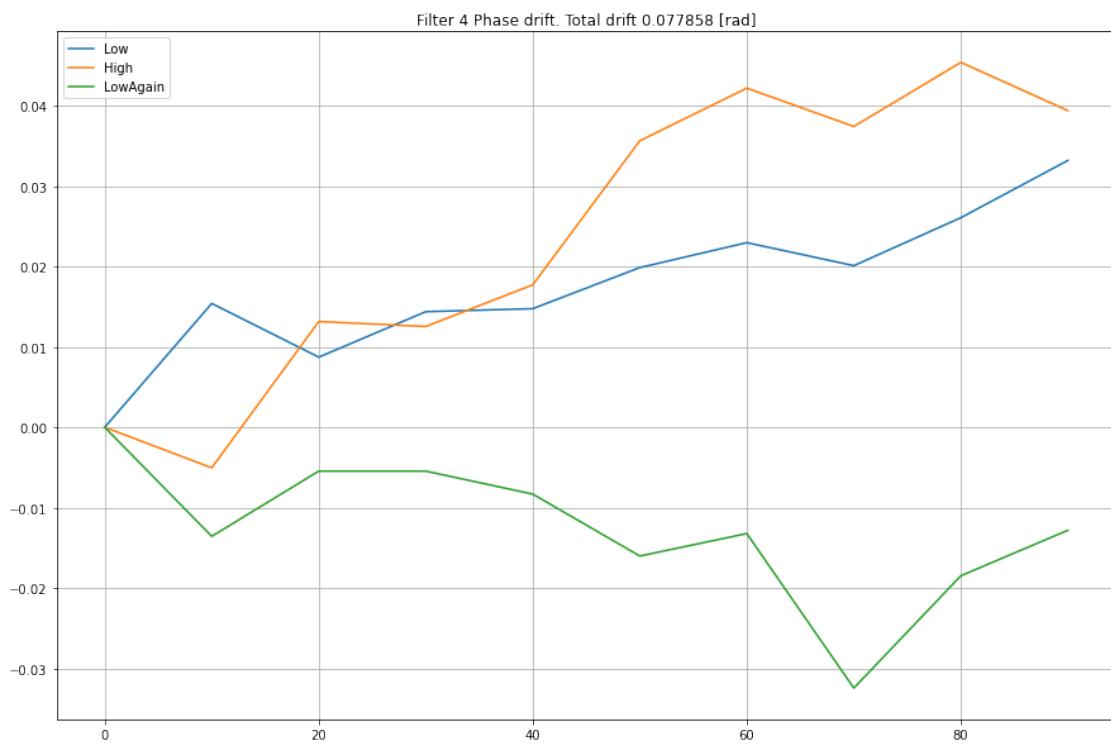
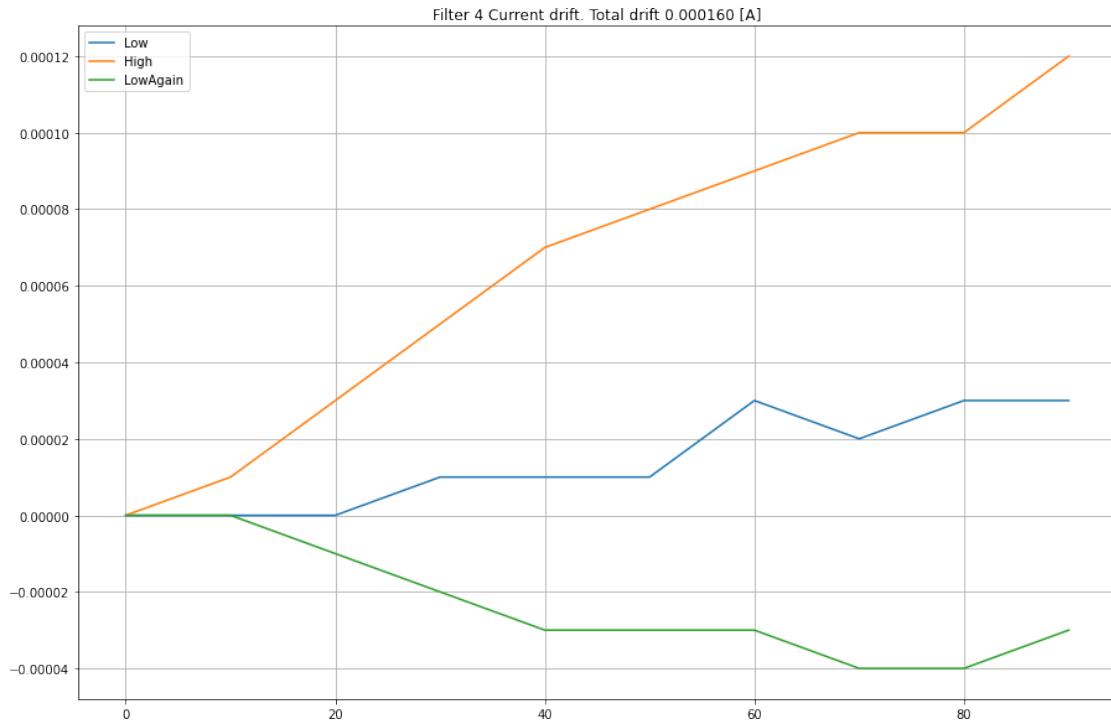


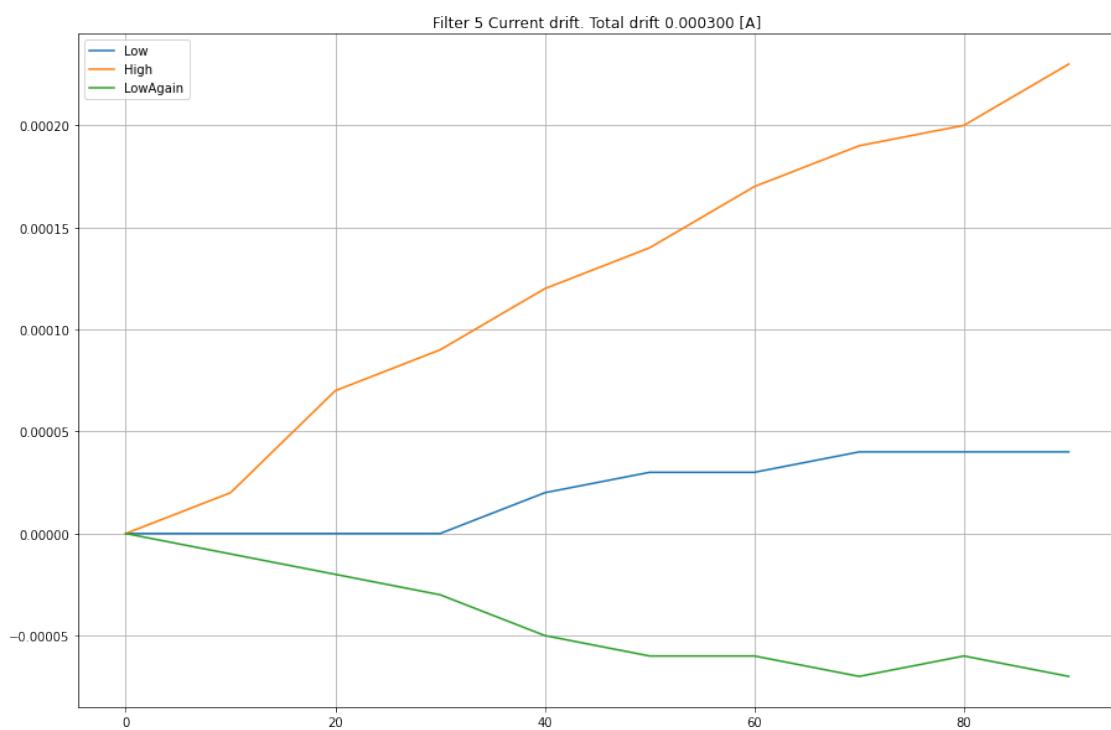
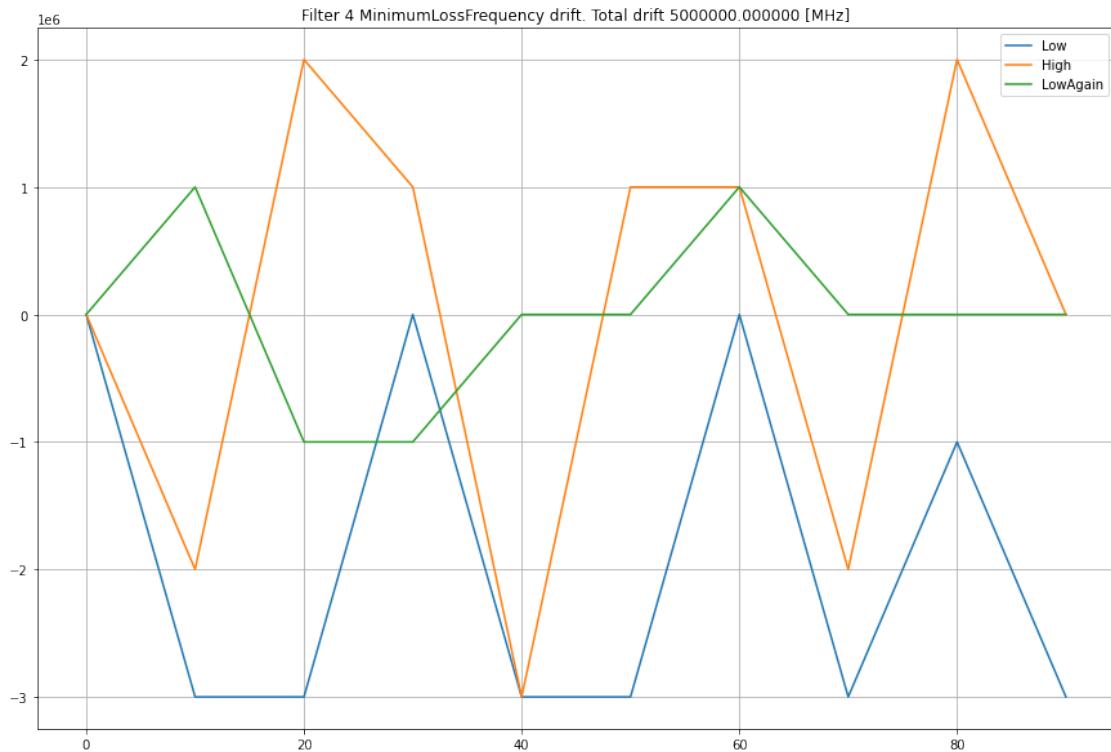


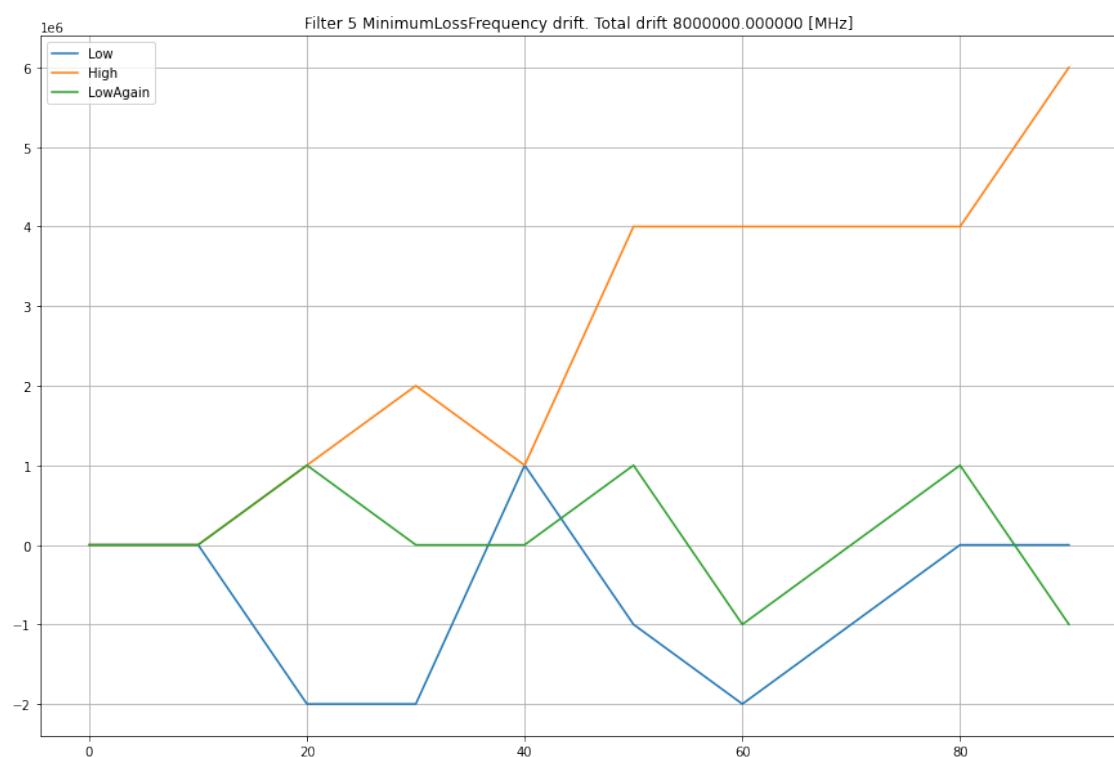
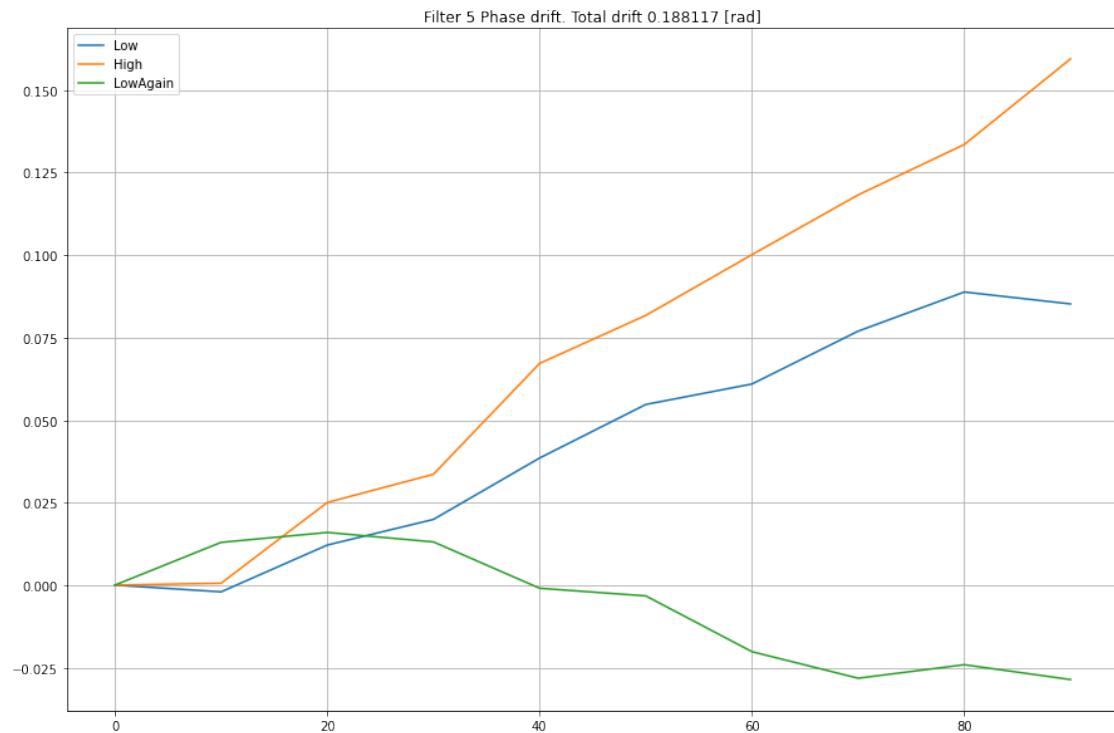












0.5 Fine tuning of filter tuning parameters

```
[3]: a = loadData('coarse_filter_parameters')
import yig_controller_test
import yig_filter_model

ks = a['k'][0]
ms = a['m'][0]
filterBank = []
for i in range(len(ks)):
    filterBank.append(yig_controller_test.YigFilter(fMin[i], fMax[i], ms[i]*1e6, ks[i]*1e6, yc, i))

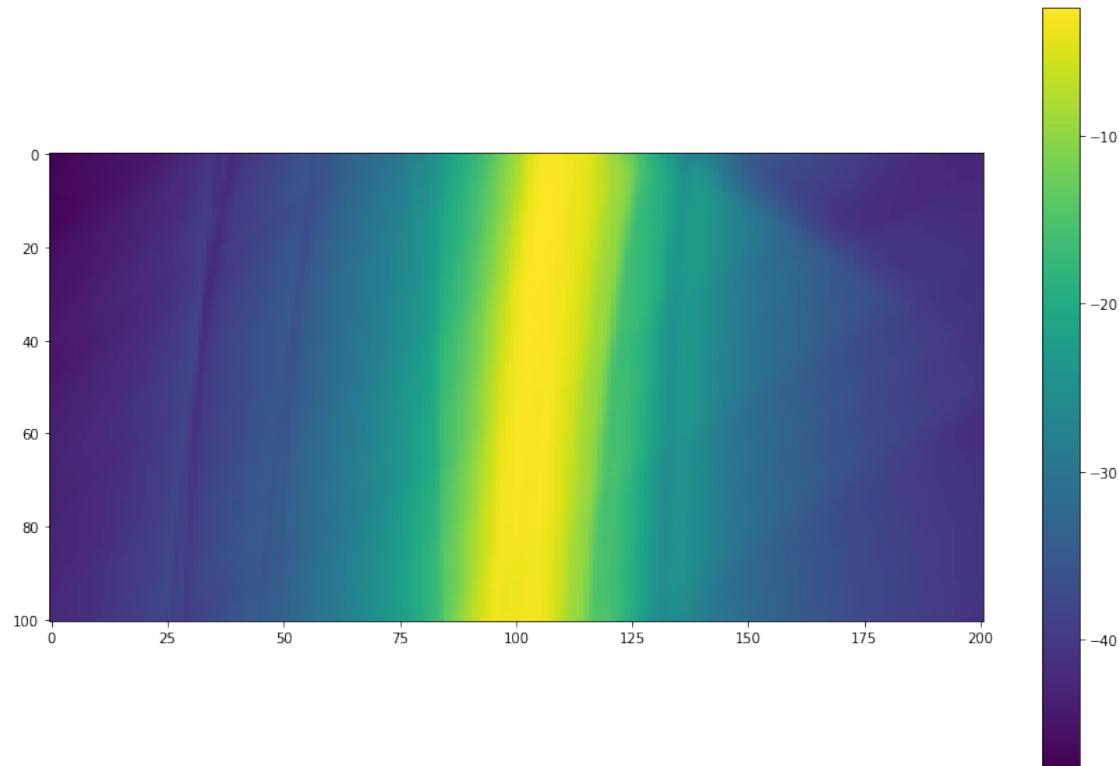
calMeas = skrf.network.Network('cal_through.s2p')

fix = yig_filter_model.SimpleS21Fixture(calMeas.f, calMeas.s[:, 1, 0])

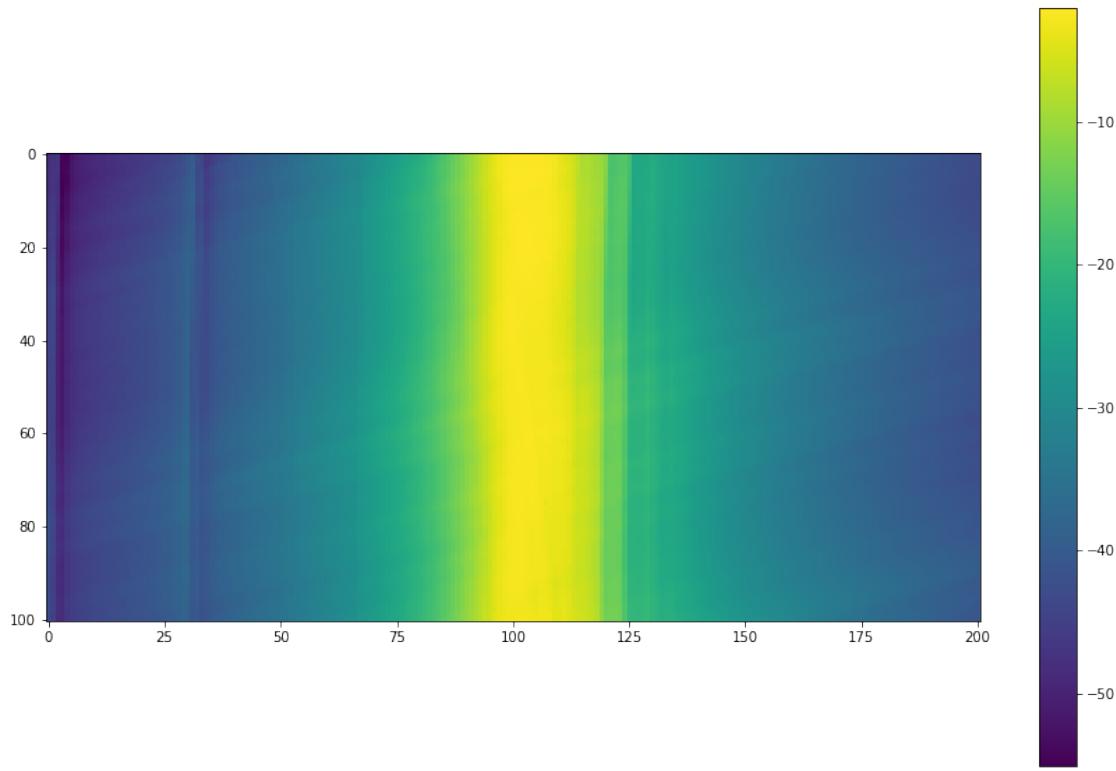
yc.yigB.set(0,0)
yc.yigA.set(6,0)
yc.yigB.set(7,0)

def measureYigFilter(f):
    frequencies = np.linspace(f.flow, f.fhigh, 101)
    vna.setPoints(201)
    filterMap=None
    spanMap=None
    tuningWords=[]
    for fr in frequencies:
        f.tuneTo(fr, channel=yigDriver)
        tuningWords.append(f.computeTuningWord(fr))
        vna.setStartFrequency(fr-250e6)
        vna.setStopFrequency(fr+250e6)
        spar=vna.readSParameter('S21')
        fax = vna.frequencies()
        spanMap = yig_controller_test.stackVector(spanMap, fax)
        dePar=fix.deembedFrom(fax, spar)
        filterMap=yig_controller_test.stackVector(filterMap, dePar)
    plt.figure()
    plt.imshow(20*np.log10(np.abs(filterMap)))
    plt.colorbar()
    plt.show()
    return filterMap, tuningWords, frequencies, spanMap
```

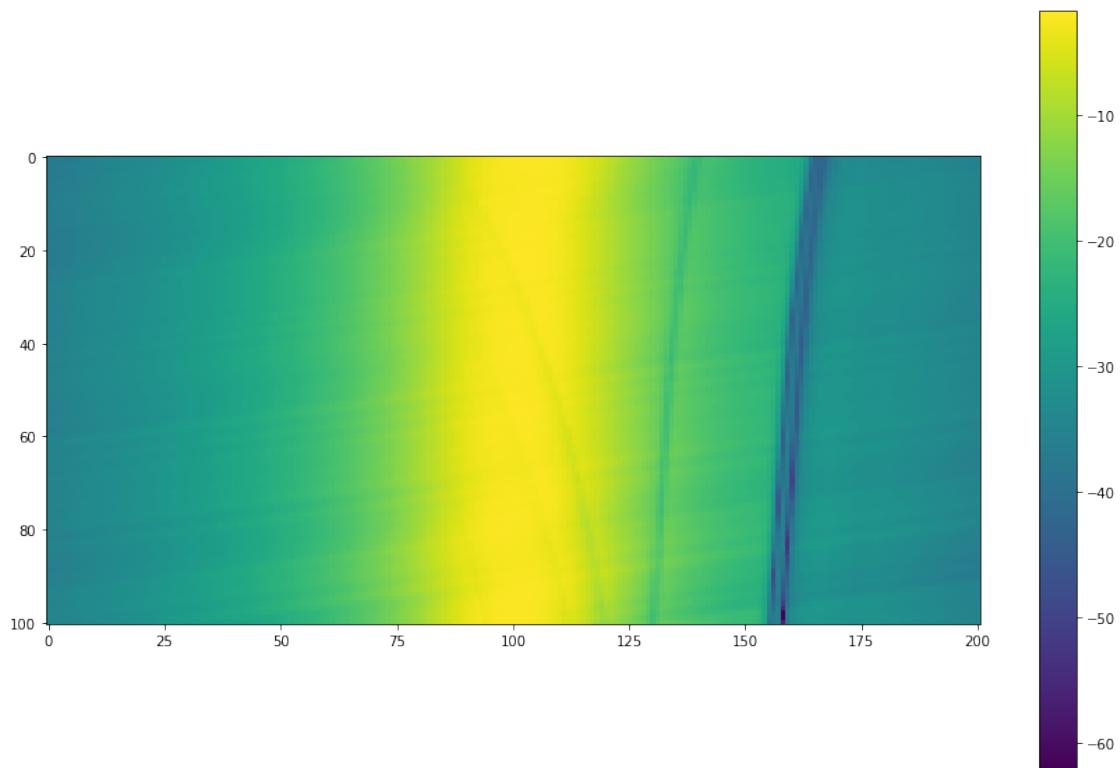
```
[10]: filter0Map, filter0TuningWords, filter0Frequencies, filter0SpanMap =  
      ↪measureYigFilter(filterBank[0])
```



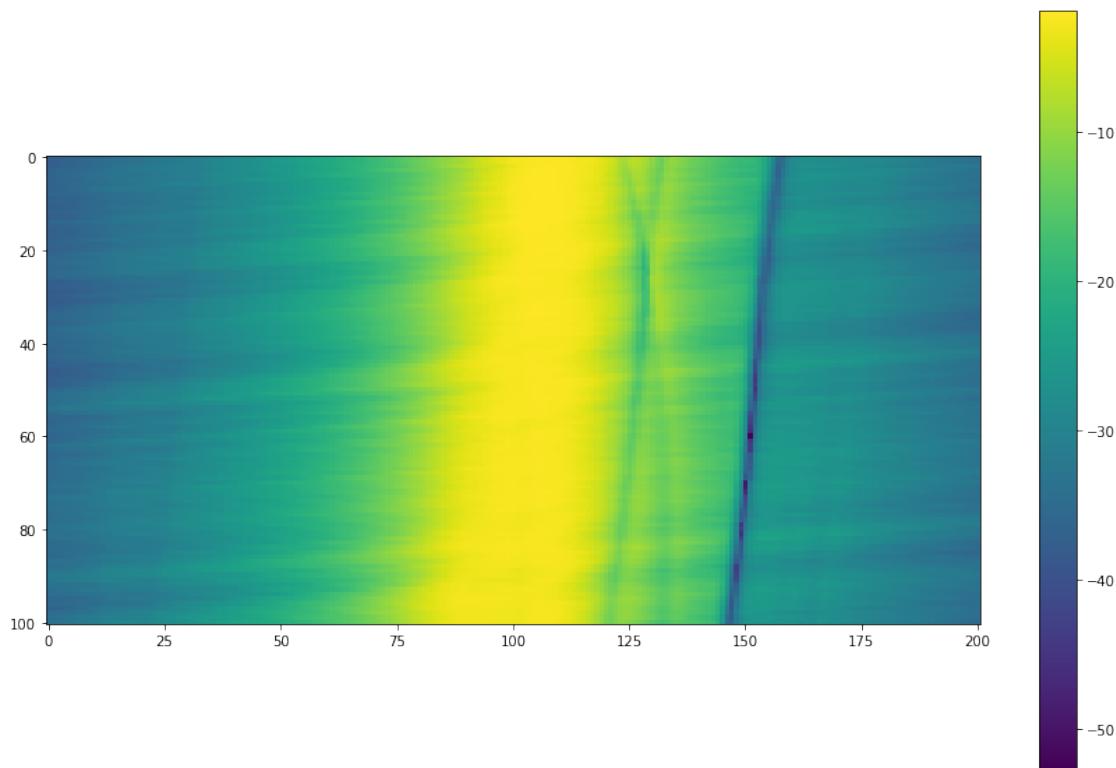
```
[11]: filter1Map, filter1TuningWords, filter1Frequencies, filter1SpanMap =  
      ↪measureYigFilter(filterBank[1])
```



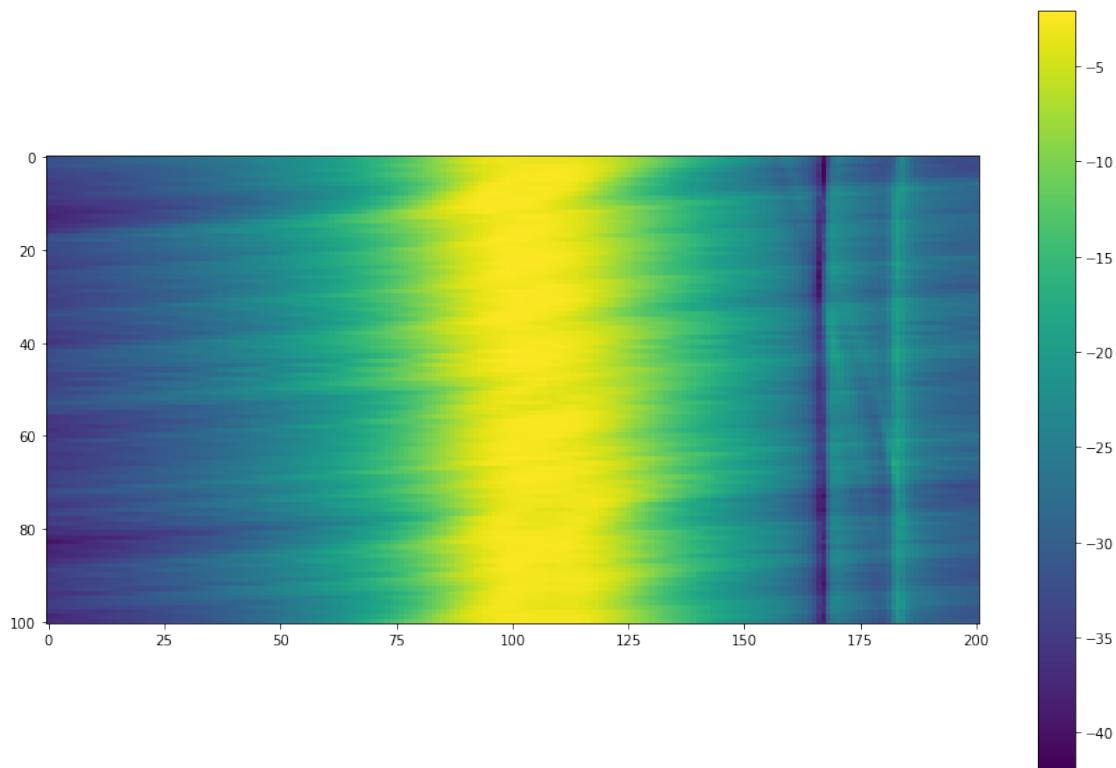
```
[5]: filter2Map, filter2TuningWords, filter2Frequencies, filter2SpanMap =  
      ↵measureYigFilter(filterBank[2])
```



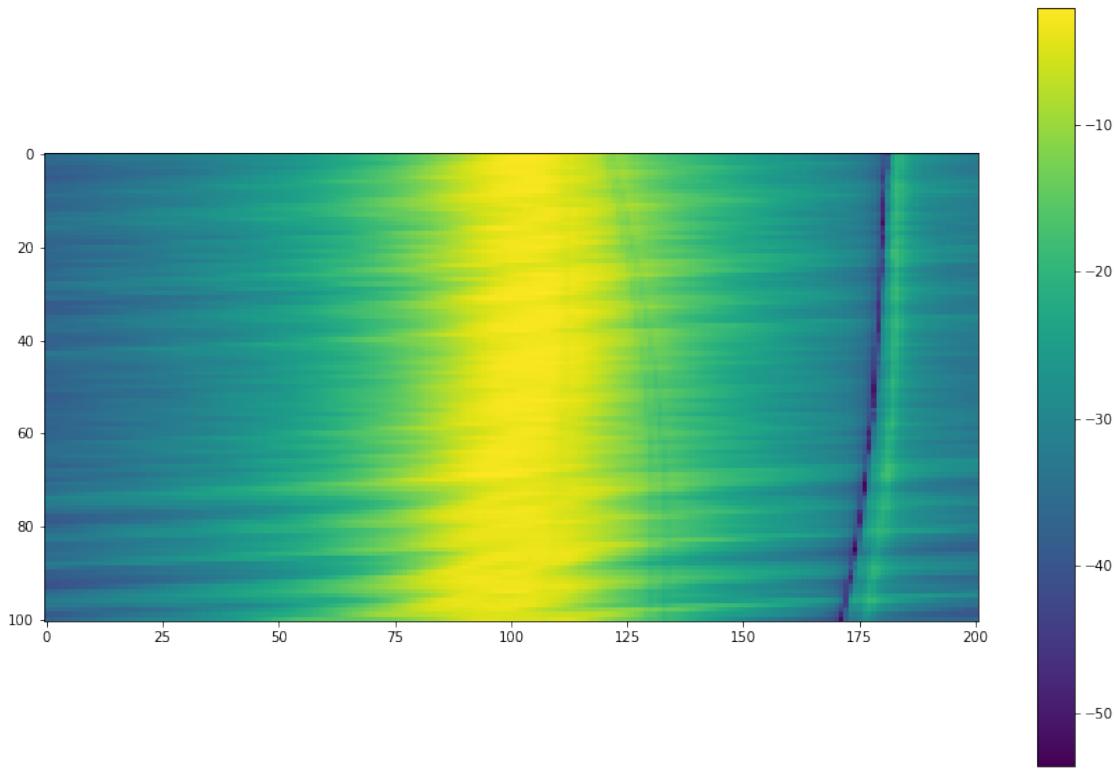
```
[6]: filter3Map, filter3TuningWords, filter3Frequencies, filter3SpanMap =  
      ↵measureYigFilter(filterBank[3])
```



```
[7]: filter4Map, filter4TuningWords, filter4Frequencies, filter4SpanMap =  
      ↵measureYigFilter(filterBank[4])
```



```
[8]: filter5Map, filter5TuningWords, filter5Frequencies, filter5SpanMap =  
      ↵measureYigFilter(filterBank[5])
```



```
[12]: saveDict={'filter0Map':filter0Map, 'filter0TuningWords':filter0TuningWords, □
    ↵'filter0Frequencies':filter0Frequencies, 'filter0SpanMap':filter0SpanMap,
    'filter1Map':filter1Map, 'filter1TuningWords':filter1TuningWords, □
    ↵'filter1Frequencies':filter1Frequencies, 'filter1SpanMap':filter1SpanMap,
    'filter2Map':filter2Map, 'filter2TuningWords':filter2TuningWords, □
    ↵'filter2Frequencies':filter2Frequencies, 'filter2SpanMap':filter2SpanMap,
    'filter3Map':filter3Map, 'filter3TuningWords':filter3TuningWords, □
    ↵'filter3Frequencies':filter3Frequencies, 'filter3SpanMap':filter3SpanMap,
    'filter4Map':filter4Map, 'filter4TuningWords':filter4TuningWords, □
    ↵'filter4Frequencies':filter4Frequencies, 'filter4SpanMap':filter4SpanMap,
    'filter5Map':filter5Map, 'filter5TuningWords':filter5TuningWords, □
    ↵'filter5Frequencies':filter5Frequencies, 'filter5SpanMap':filter5SpanMap, }
saveData('coarse_tuning_fine_measurements', saveDict)
```

```
[13]: def dB(data):
    return 20*np.log10(np.abs(data))

def fineTuneAnalysis(meas, i):
    baseKey='filter%d'%(i)
    filterMap = meas[baseKey+'Map']
    tuningWords = meas[baseKey+'TuningWords'][0]
    frequencies = meas[baseKey+'Frequencies'][0]
```

```

spanMap = meas[baseKey+'SpanMap']
dBfilt=dB(filterMap)
peaksIdx=np.argmax(dB(filterMap),axis=1)
peaksVal=np.max(dB(filterMap),axis=1)

x = np.arange(peaksIdx.shape[0])
bestLineCoeff=np.polyfit(x, peaksIdx-100, deg=1)
k, m =bestLineCoeff

span=spanMap[0]
deltaF=(max(span)-min(span))/float(len(span)-1)
deltaW =(max(tuningWords)-min(tuningWords))/float(len(tuningWords)-1)

kcorr=k*(deltaF/1e6)*(1/deltaW)
mcorr=m*deltaF/1e6
lineFunc=np.poly1d(bestLineCoeff)

filterNorm = (dBfilt.T-peaksVal).T;
plt.figure();
plt.title("Filter %d"%(i))

plt.imshow(filterNorm)

yr = np.array(range(filterMap.shape[0]))

plt.plot(peaksIdx, yr)
peaksIdxFilter=np.convolve(peaksIdx, np.ones((10,))/10, mode='same')

plt.plot(peaksIdxFilter, yr)
plt.plot(lineFunc(x)+100,x)
plt.contour(filterNorm, [-10, -6, -3])
plt.figure()
plt.title("Filter %d"%(i))
plt.plot(np.arange(10,91), peaksIdxFilter[10:-10]-100)
plt.plot(peaksIdx-100)
plt.grid(True)

plt.plot(x, lineFunc(x))
return kcorr, mcorr
#plt.contourf(db(filterMap), [])

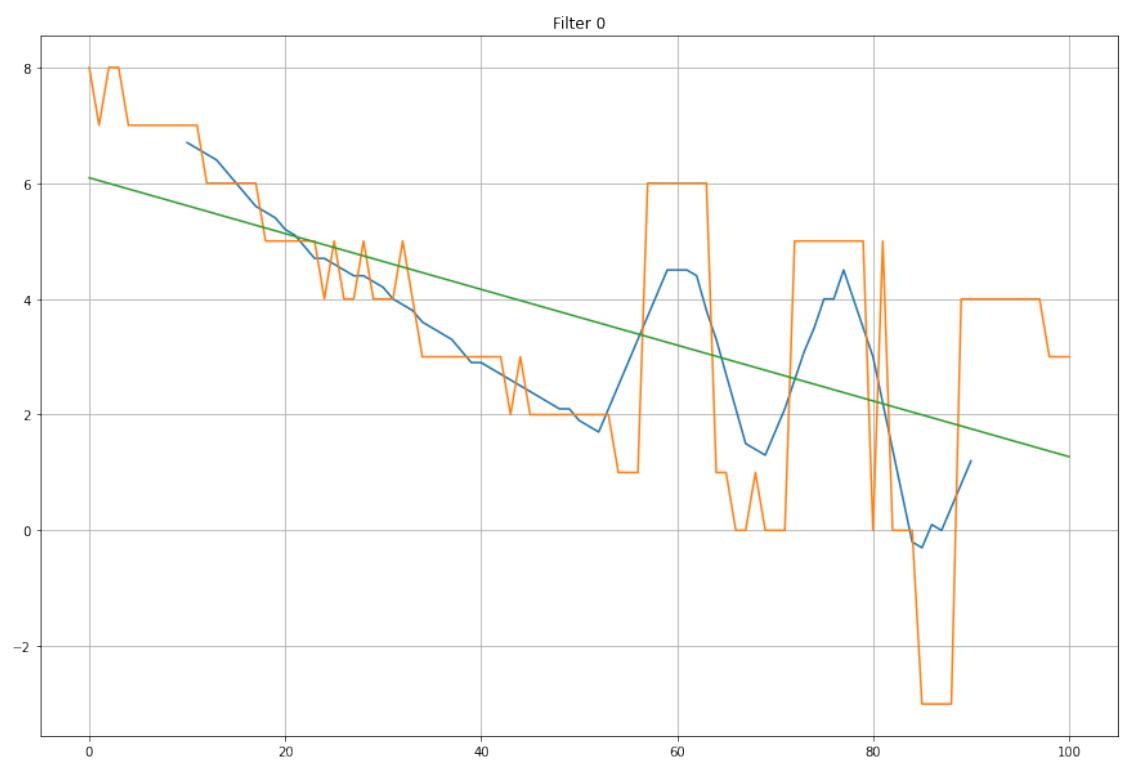
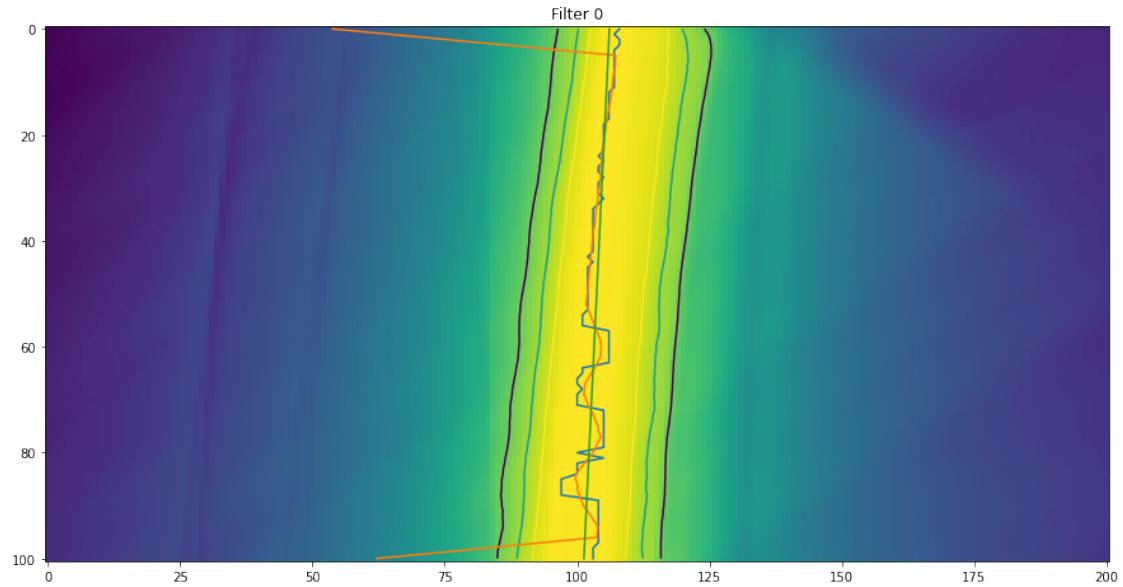
```

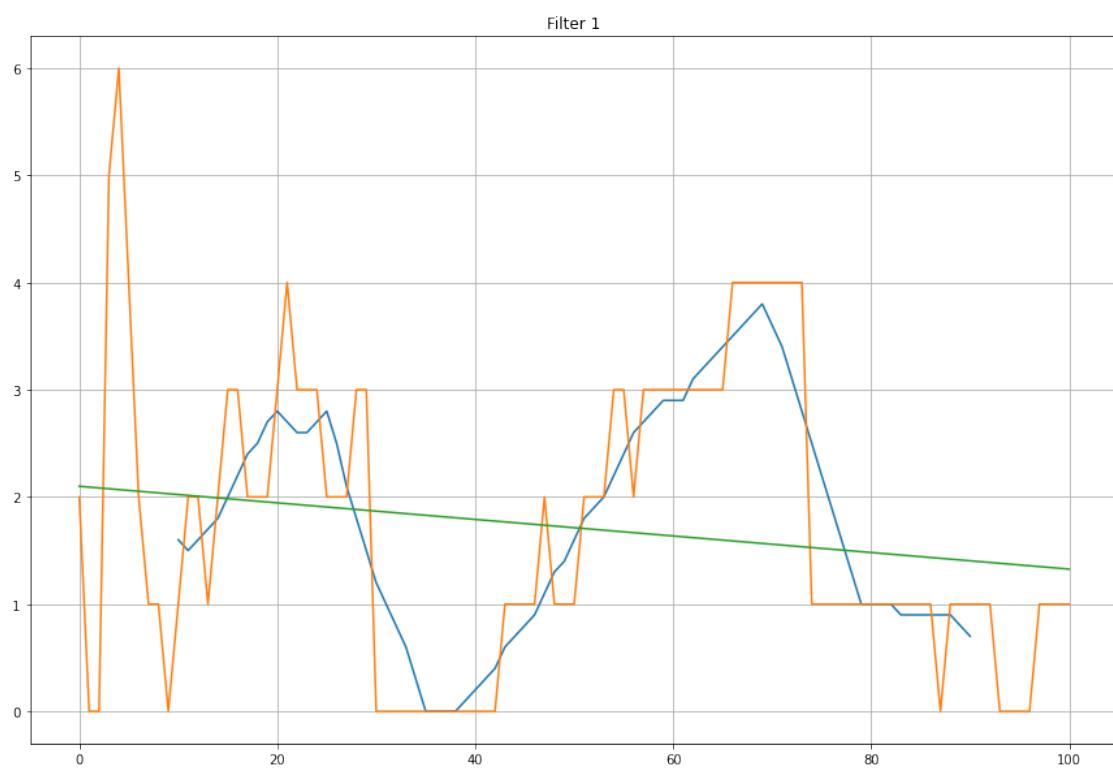
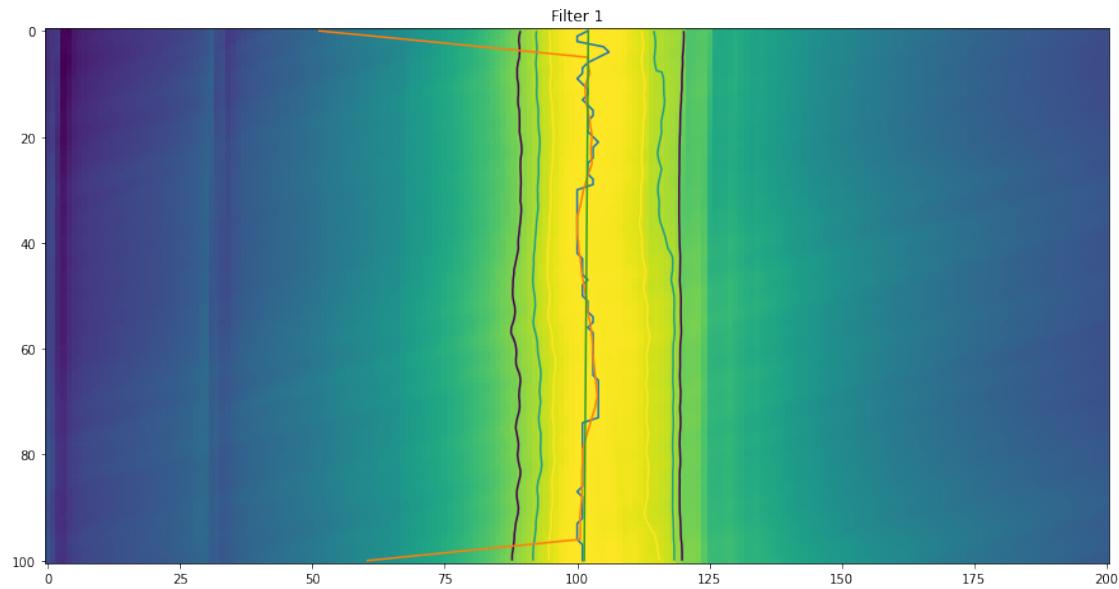
```
[14]: ftm = loadData('coarse_tuning_fine_measurements')
coarseData = loadData('coarse_filter_parameters')

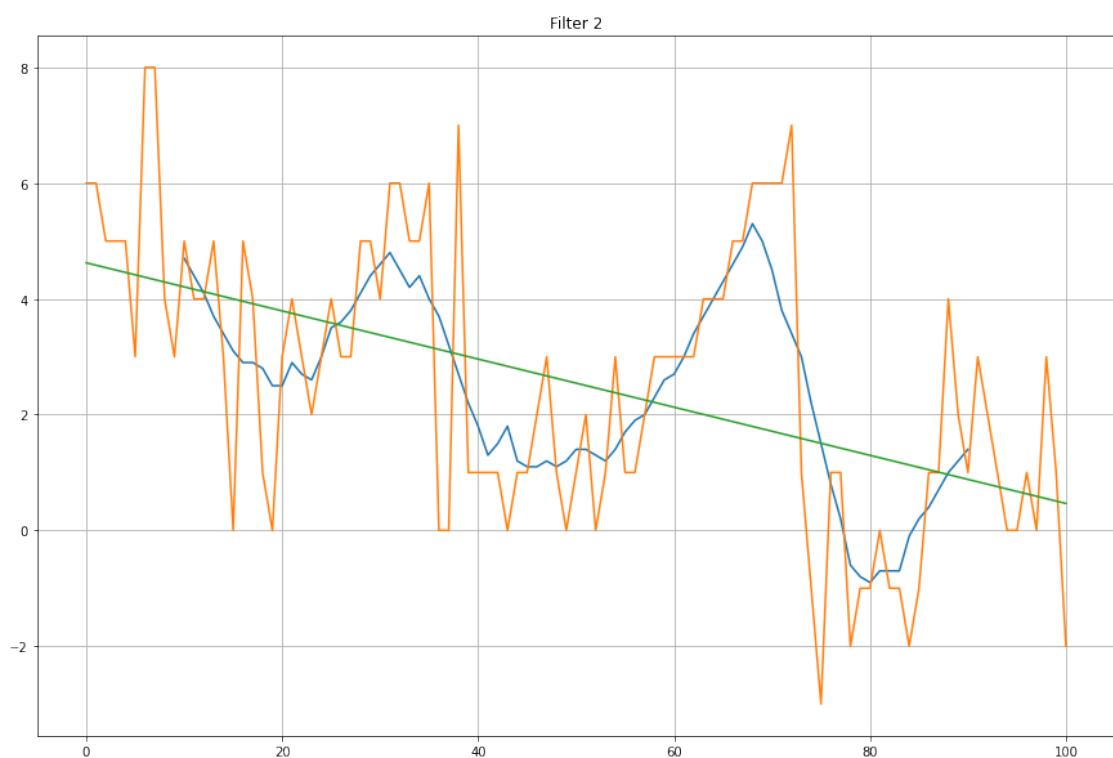
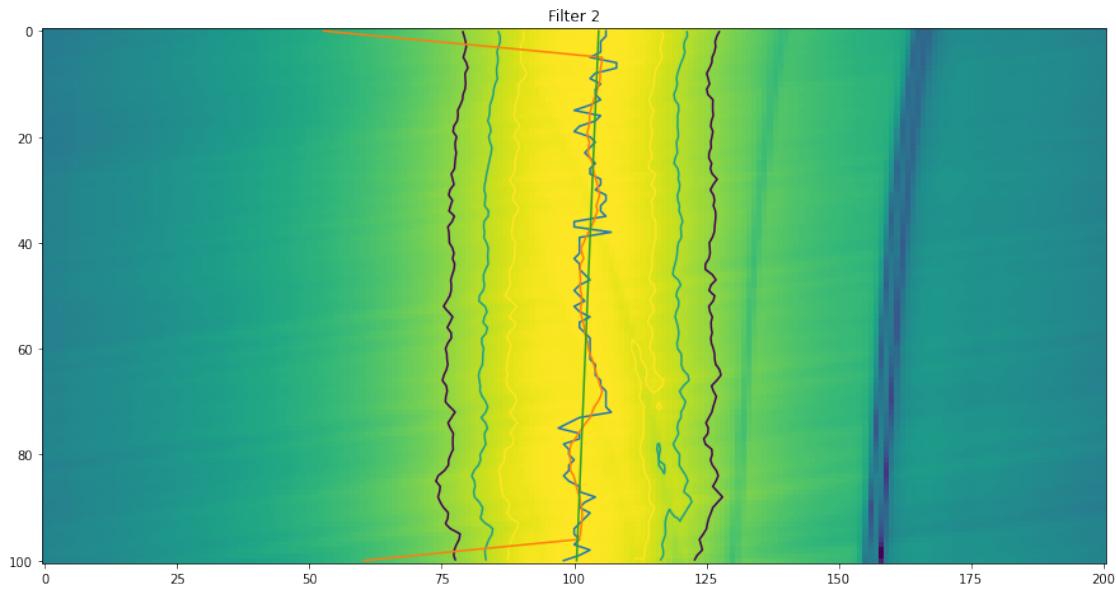
kcoarse = coarseData['k'][0]
mcoarse = coarseData['m'][0]
kfine = kcoarse
mfine = mcoarse

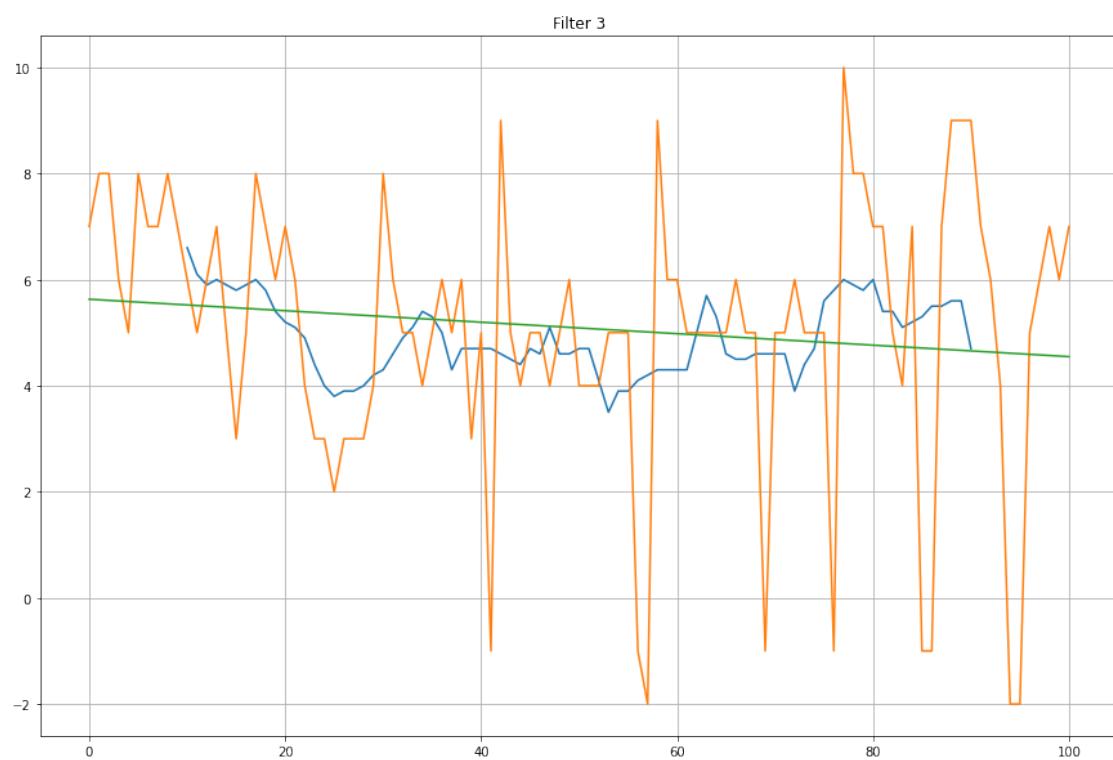
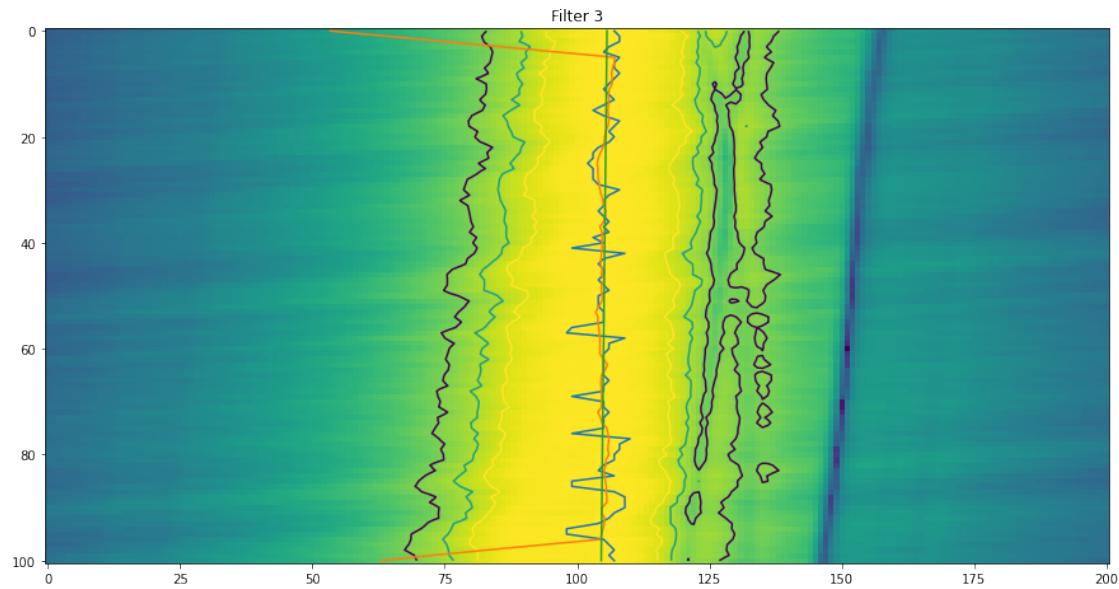
for i in range(6):
    k, m = fineTuneAnalysis(ftm, i)
    print("filter", i, "Coarse k", kcoarse[i], "[MHZ/lsb]", "coarse m", ↴
          ↴mcoarse[i], "[MHz]")
    kfine[i] = kcoarse[i] + k
    mfine[i] = mcoarse[i] + m
    print("filter", i, "fine k", kfine[i], "[MHZ/lsb]", "cfine m", mfine[i], ↴
          ↴"[MHz]")
fineTuned={'k':kfine, 'm':mfine}
saveData('fine_filter_parameters', fineTuned)
```

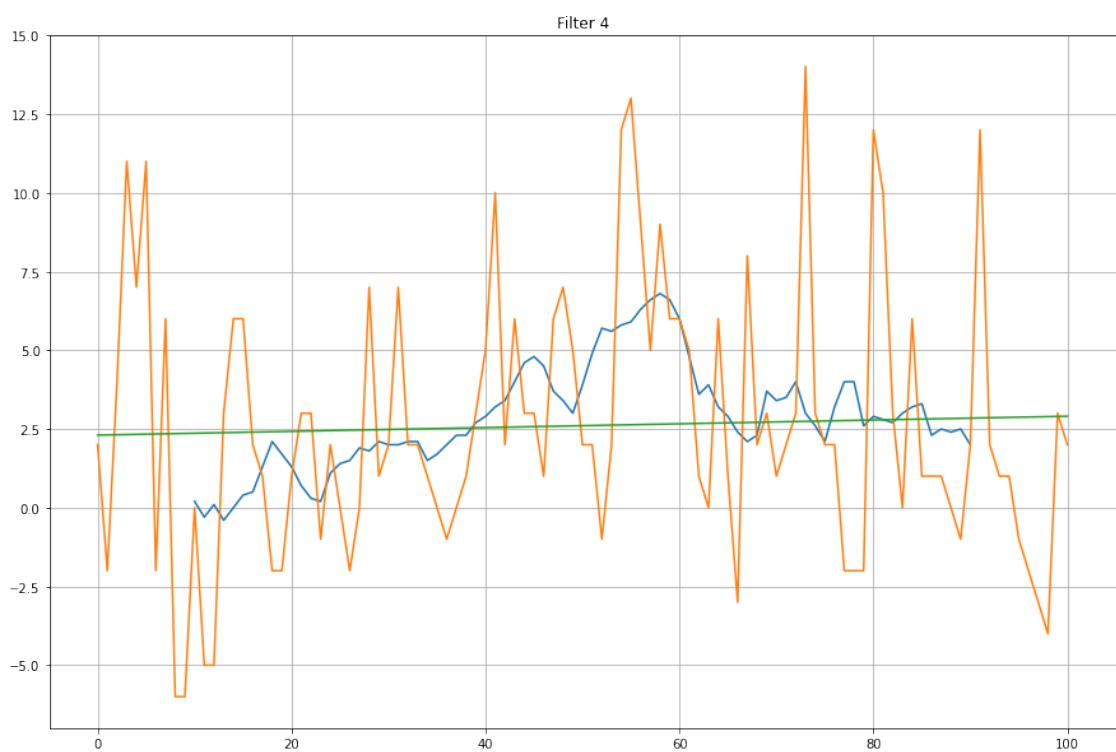
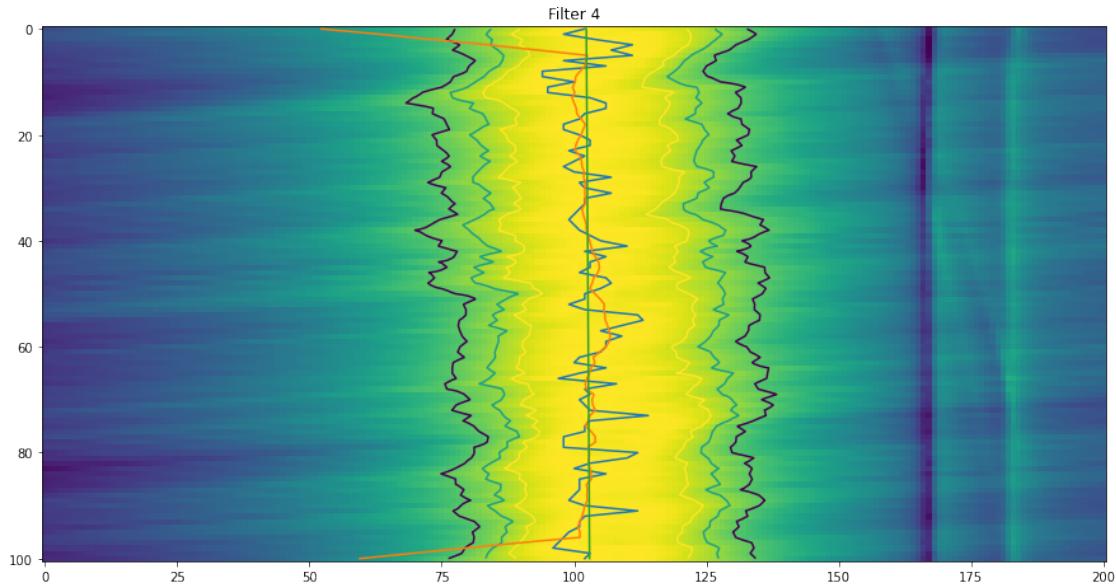
```
filter 0 Coarse k 0.024734452874218625 [MHZ/lsb] coarse m 689.8949186242279
[MHz]
filter 0 fine k 0.02398946645991117 [MHZ/lsb] cfine m 705.1264270691901 [MHz]
filter 1 Coarse k 0.07207430388367966 [MHZ/lsb] coarse m 1989.3674867148818
[MHz]
filter 1 fine k 0.07193514480415632 [MHZ/lsb] cfine m 1994.6150114673571
[MHz]
filter 2 Coarse k 0.16690386625696907 [MHZ/lsb] coarse m 4531.73029607712 [MHz]
filter 2 fine k 0.16603630057191854 [MHZ/lsb] cfine m 4543.289702017714 [MHz]
filter 3 Coarse k 0.3308892682917341 [MHZ/lsb] coarse m 8961.721804721628 [MHz]
filter 3 fine k 0.3306654863460182 [MHZ/lsb] cfine m 8975.797226969735 [MHz]
filter 4 Coarse k 0.3046263258415695 [MHZ/lsb] coarse m 8356.186718164145 [MHz]
filter 4 fine k 0.30474032406993945 [MHZ/lsb] cfine m 8361.948220784994 [MHz]
filter 5 Coarse k 0.36104698851157285 [MHZ/lsb] coarse m 9871.337649071173 [MHz]
filter 5 fine k 0.35989278475148645 [MHZ/lsb] cfine m 9885.581970562145 [MHz]
```

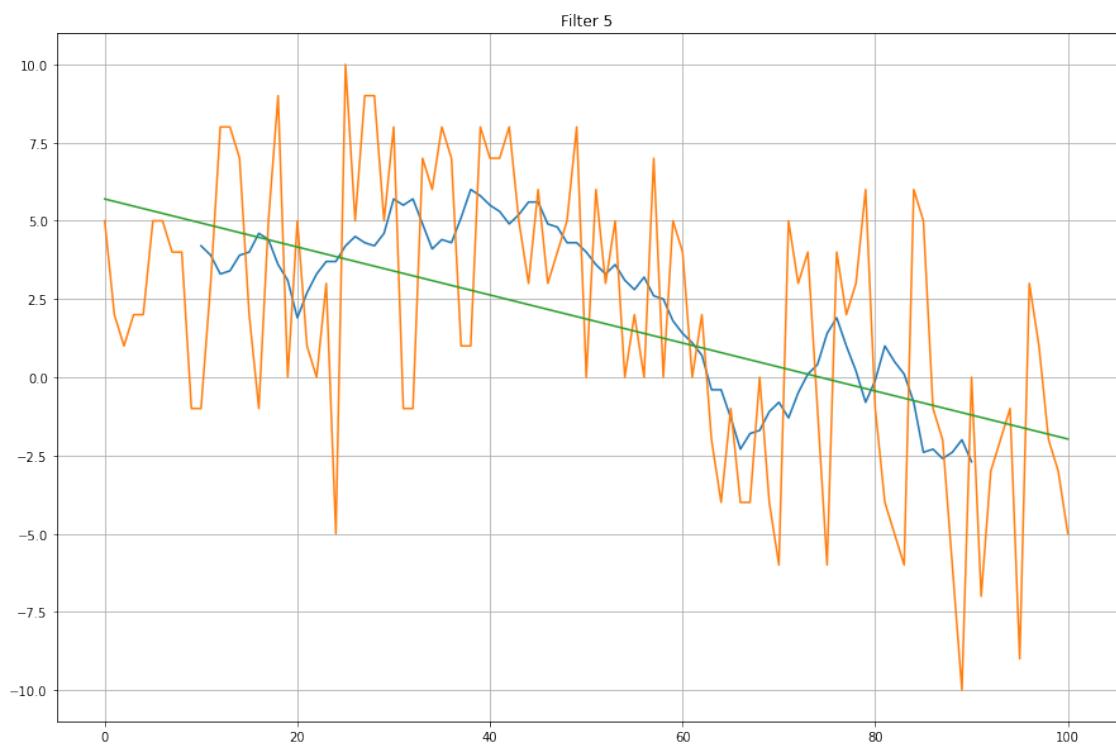
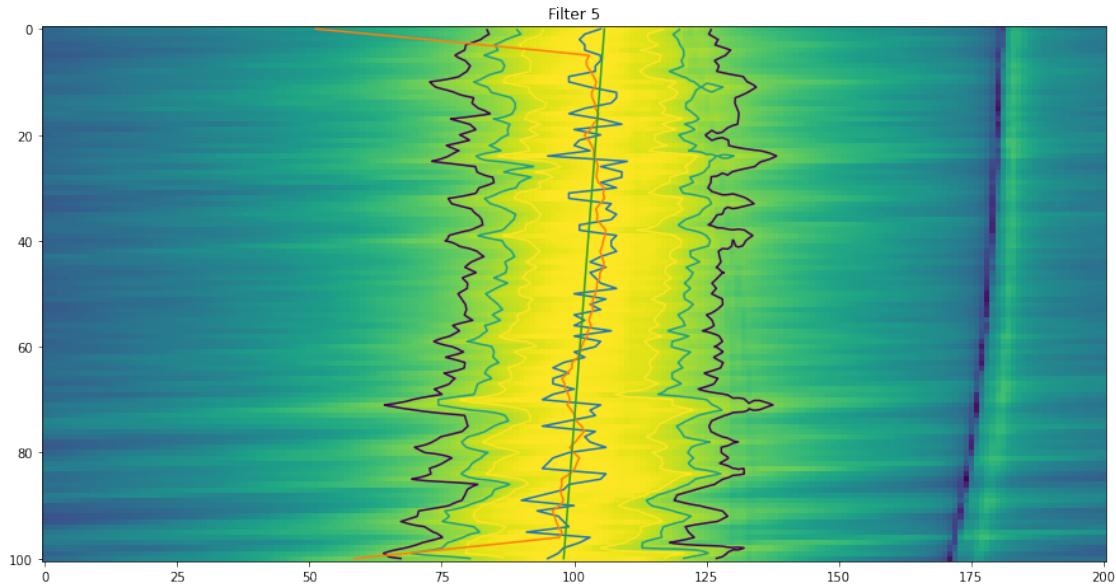












```
[15]: ftm = loadData('coarse_tuning_fine_measurements')
coarseData = loadData('coarse_filter_parameters')
```

```
print(ftm.keys())
```

```

def computeCenterOfMass(s21Mag):
    ns=s21Mag.shape[1]
    massPositionProduct=np.multiply(s21Mag,np.arange(ns))
    massPositionProductSum=np.sum(massPositionProduct, axis=1)
    centerOfMass = massPositionProductSum/np.sum(s21Mag, axis=1)
    return centerOfMass

def computeCenterOfMassLim(s21Mag, lim):
    rows, numSamp=s21Mag.shape
    centerOfMass=[]
    for r in range(rows):
        currentLine = s21Mag[r,:]
        valids = currentLine >= lim
        s21sel=currentLine[valids]
        idxSel=np.arange(numSamp)[valids]
        massPositionProduct=np.multiply(s21sel,idxSel)
        massPositionProductSum=np.sum(massPositionProduct)
        centerOfMass.append(massPositionProductSum/np.sum(s21sel))
    return np.array(centerOfMass)

def indexToFrequency(idx, fmap):
    rows, cols = fmap.shape
    iax = np.arange(cols)
    fs=[]
    for row in np.arange(rows):
        fs.append(np.interp(idx[row], iax, fmap[row,:]))
    return np.array(fs)

def analyzeFilterData(s21, freqMap, words):
    s21Mag=np.abs(s21)
    rows=s21Mag.shape[0]
    ya = np.arange(rows)

    peakValues = np.max(s21Mag, axis=1)
    s21Norm = s21Mag / peakValues[:,None]

    centerOfMass = computeCenterOfMassLim(s21Mag, 0.5)
    centerOfMassCoeff = np.polyfit(ya, centerOfMass, deg=1)
    centerOfMassFunc = np.poly1d(centerOfMassCoeff)

    #compute peak center
    peaks = np.argmax(s21Mag, axis=1)
    peaksCoeff = np.polyfit(ya, peaks, deg=1)
    peaksFunc = np.poly1d(peaksCoeff)

    plt.figure();
    plt.imshow(dB(s21Norm))

```

```

plt.plot(centerOfMass.T, ya, label='com')
plt.plot(peaks.T, ya, label='max')
plt.plot(centerOfMassFunc(ya), ya, label='combl')
plt.plot(peaksFunc(ya), ya, label='maxbl')
plt.colorbar()
plt.legend()
plt.figure()
plt.plot(ya, centerOfMass.T, label='com')
plt.plot(ya, peaks.T, label='max')
plt.plot(ya, centerOfMassFunc(ya), label='combl')
plt.plot(ya, peaksFunc(ya), label='maxbl')
plt.grid(True)
plt.legend()

centerOfMassFreqs = indexToFrequency(centerOfMass, freqMap)
linCoeff = np.polyfit(centerOfMassFreqs, words, deg=1)
k, m = linCoeff
print("K %f [LSB/MHz] M %f [LSB]"%(k*1e6, m))
return k, m

ks=[]
ms=[]

for a in range(6):
    baseKey='filter%d'%(a)
    s21=ftm[baseKey+'Map']
    freqMap = ftm[baseKey+'SpanMap']
    words = np.squeeze(ftm[baseKey+'TuningWords'])

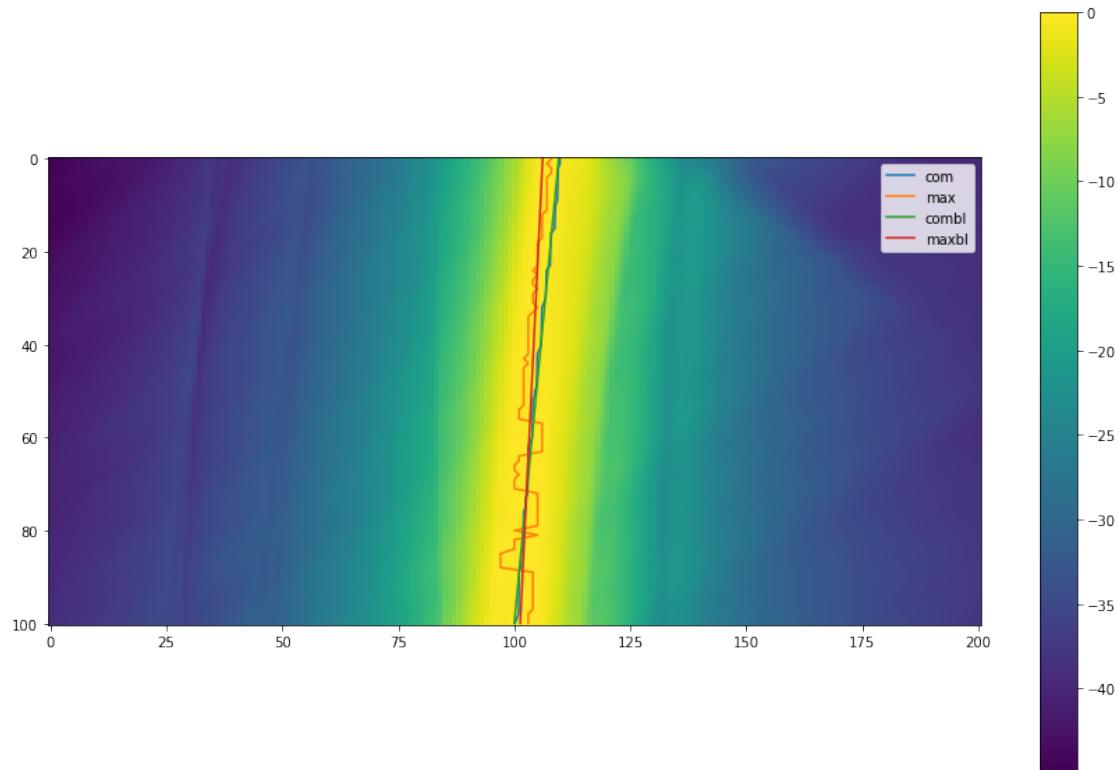
    k, m = analyzeFilterData(s21, freqMap, words)
    ks.append(k)
    ms.append(m)

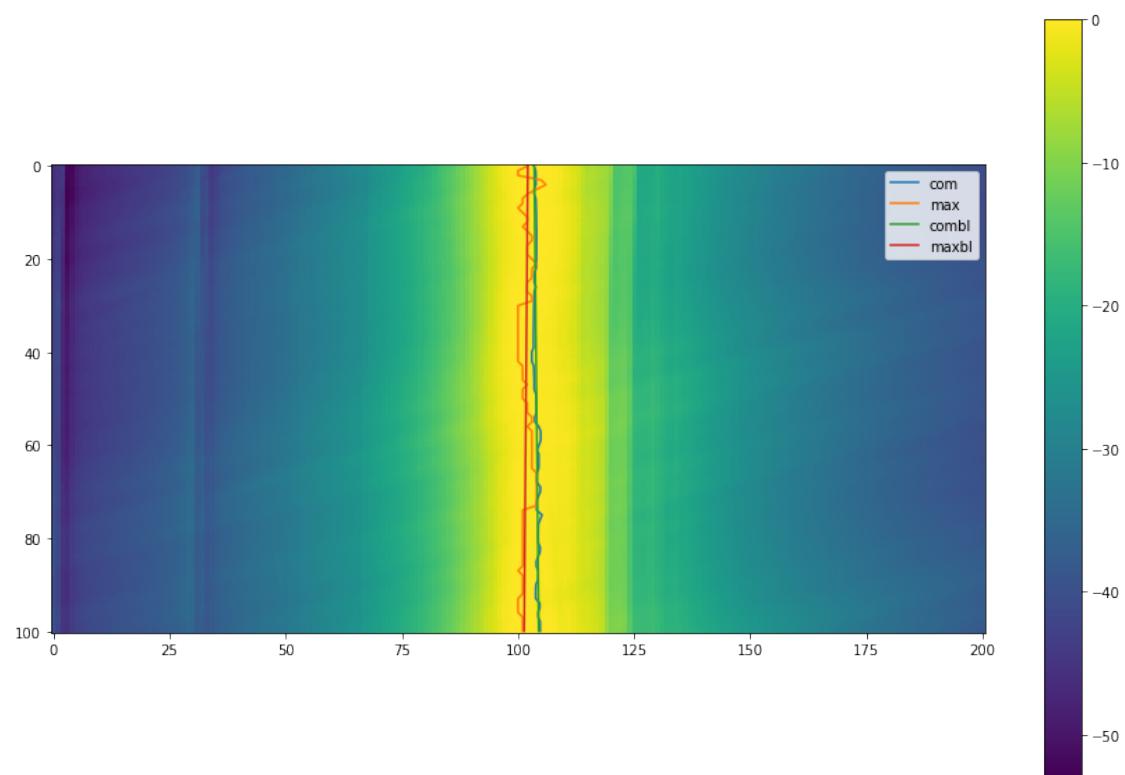
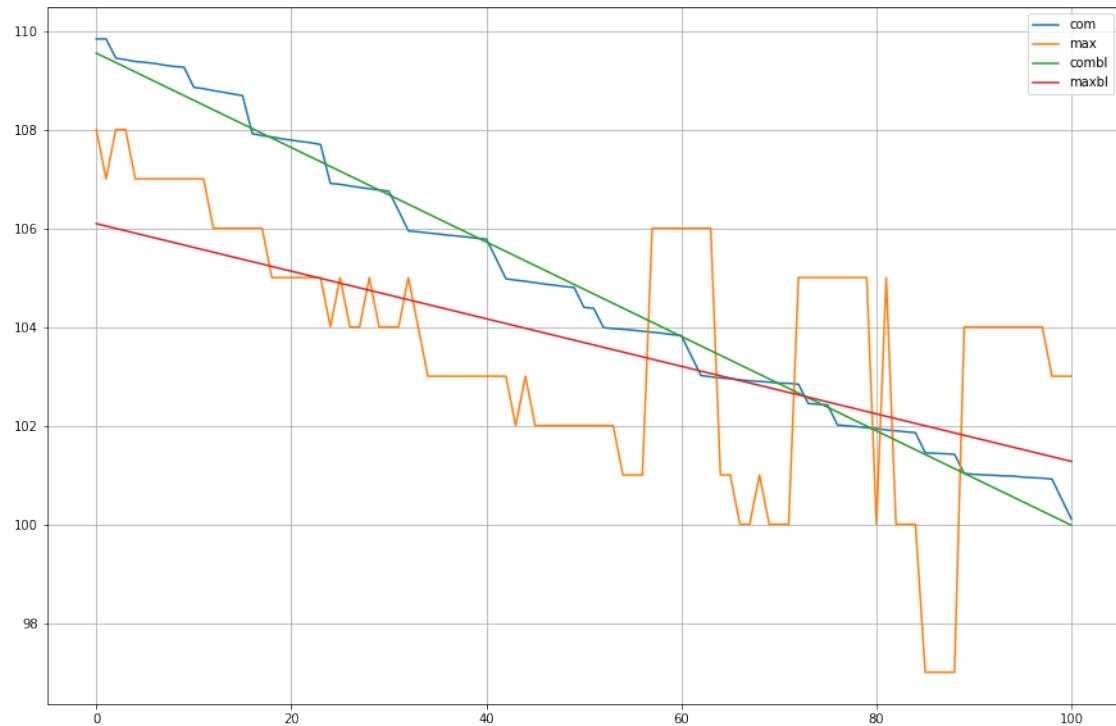
fineTuned={'k':ks, 'm':ms}
saveData('fine_filter_parameters2', fineTuned)

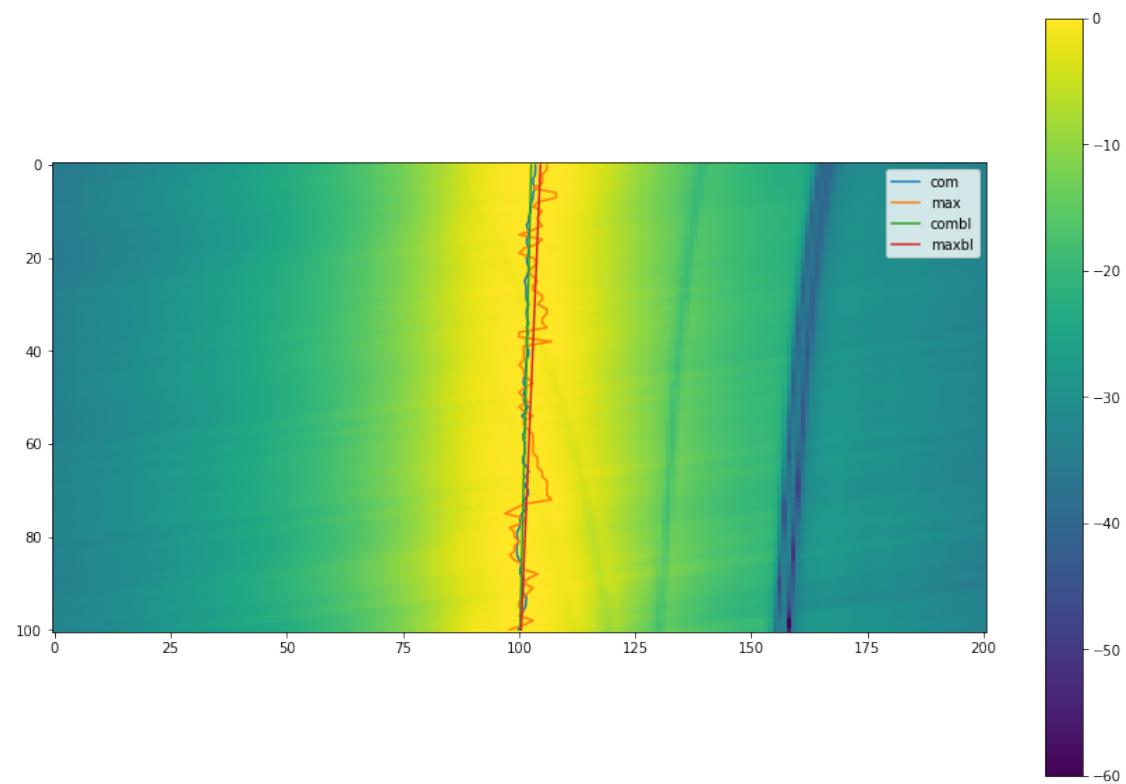
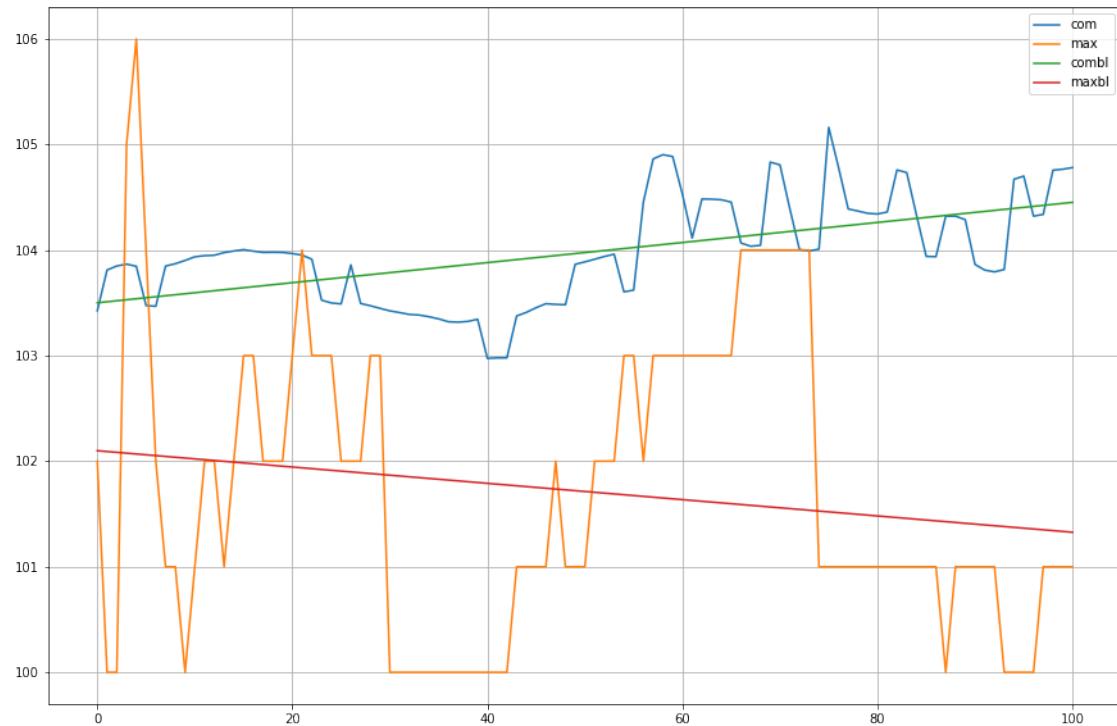
dict_keys(['__header__', '__version__', '__globals__', 'filter0Map',
'filter0TuningWords', 'filter0Frequencies', 'filter0SpanMap', 'filter1Map',
'filter1TuningWords', 'filter1Frequencies', 'filter1SpanMap', 'filter2Map',
'filter2TuningWords', 'filter2Frequencies', 'filter2SpanMap', 'filter3Map',
'filter3TuningWords', 'filter3Frequencies', 'filter3SpanMap', 'filter4Map',
'filter4TuningWords', 'filter4Frequencies', 'filter4SpanMap', 'filter5Map',
'filter5TuningWords', 'filter5Frequencies', 'filter5SpanMap',
'hardwareDescription'])

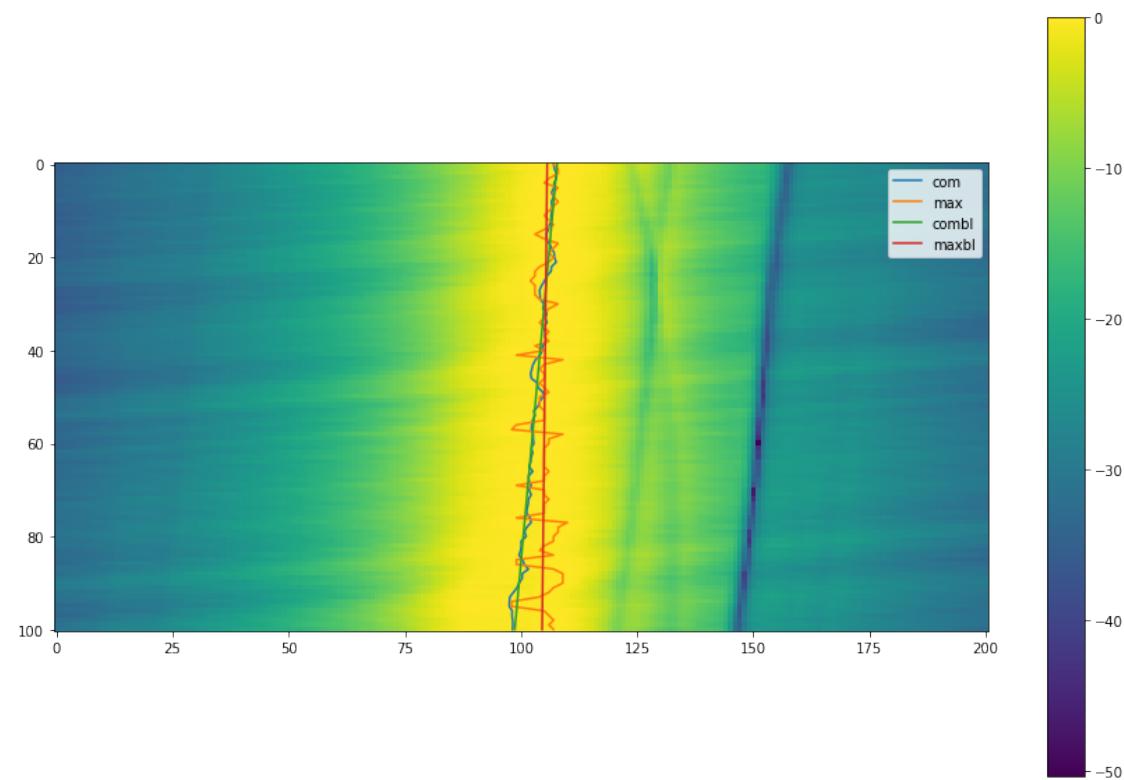
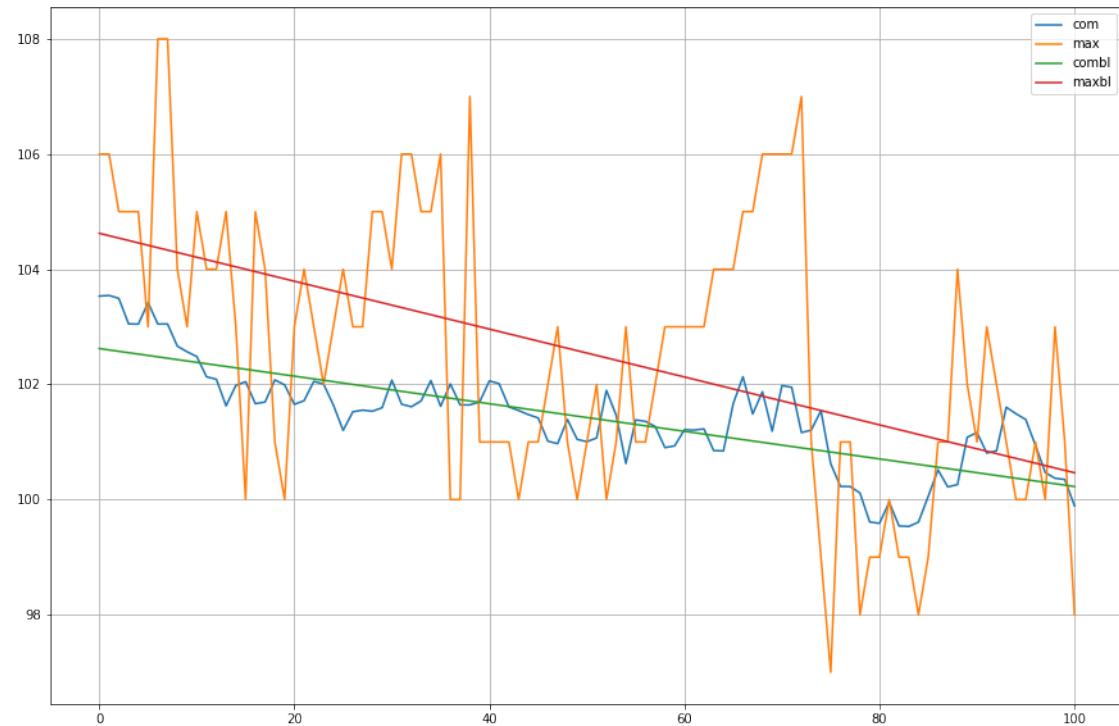
```

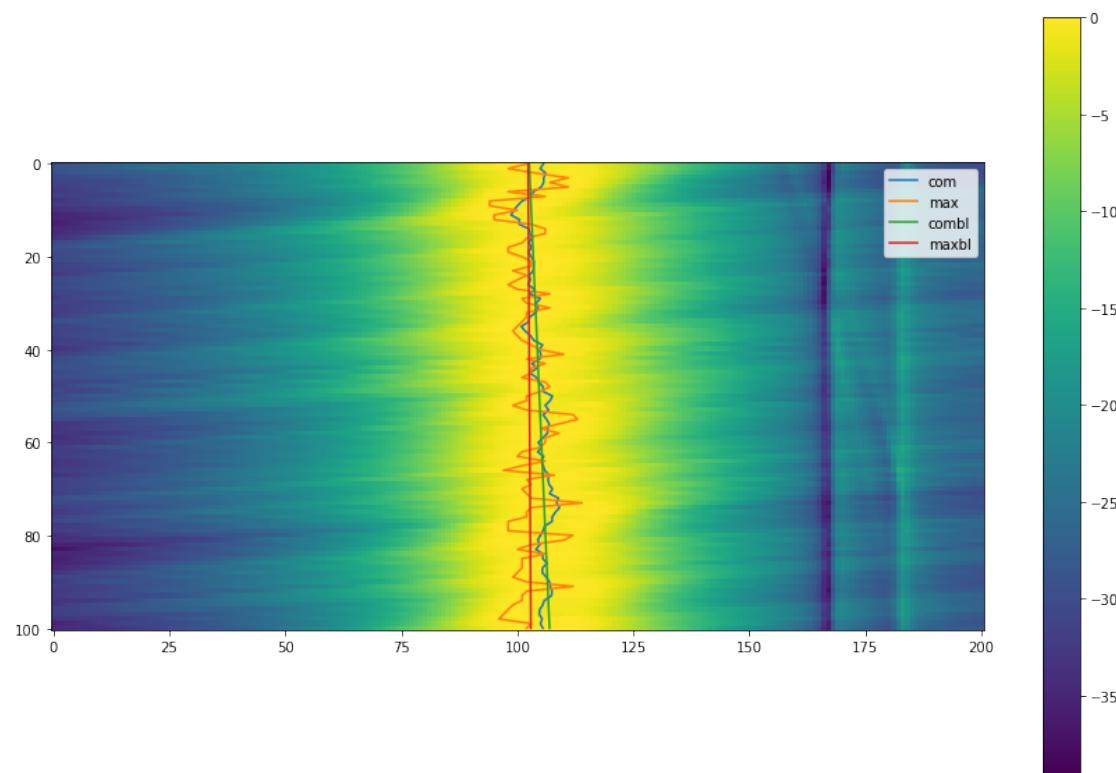
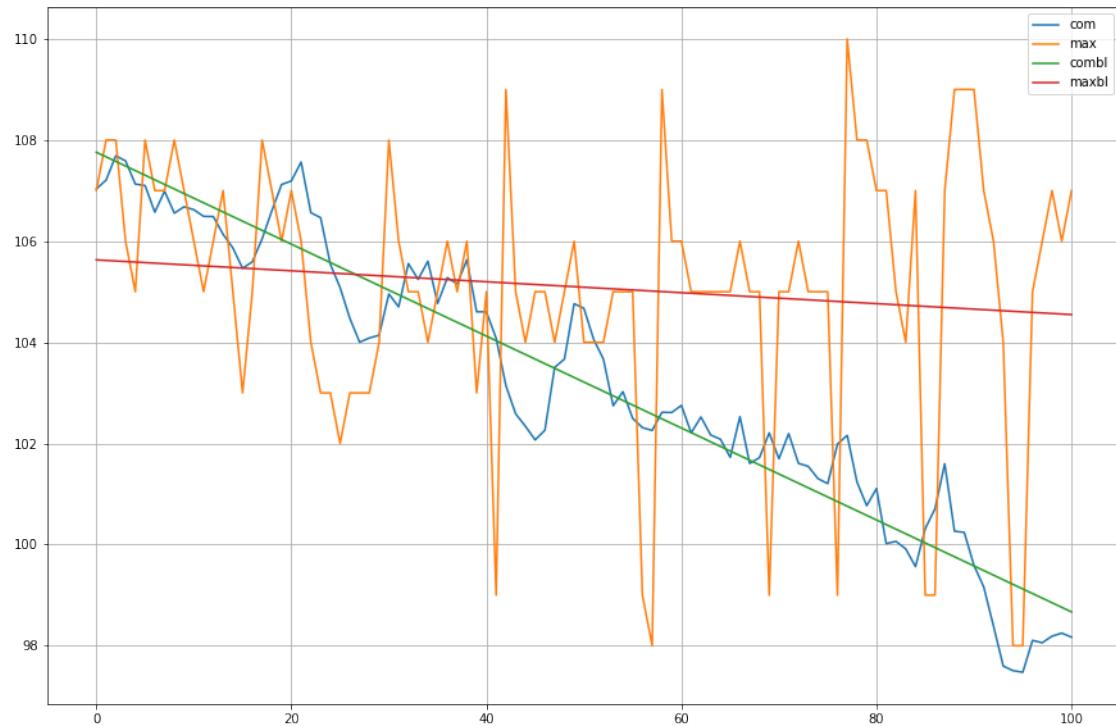
K 42.996213 [LSB/MHz] M -30457.848843 [LSB]
K 13.841439 [LSB/MHz] M -27689.030695 [LSB]
K 6.009456 [LSB/MHz] M -27226.581319 [LSB]
K 3.039428 [LSB/MHz] M -27211.280479 [LSB]
K 3.273613 [LSB/MHz] M -27379.200873 [LSB]
K 2.777058 [LSB/MHz] M -27469.594697 [LSB]

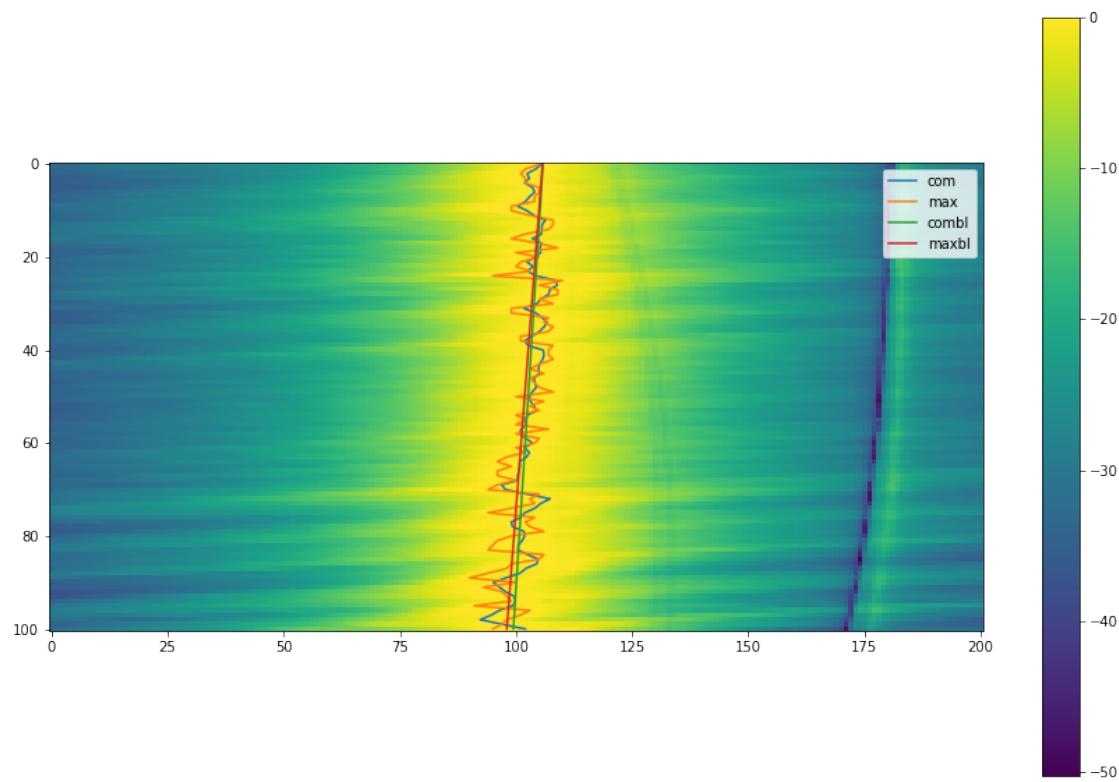
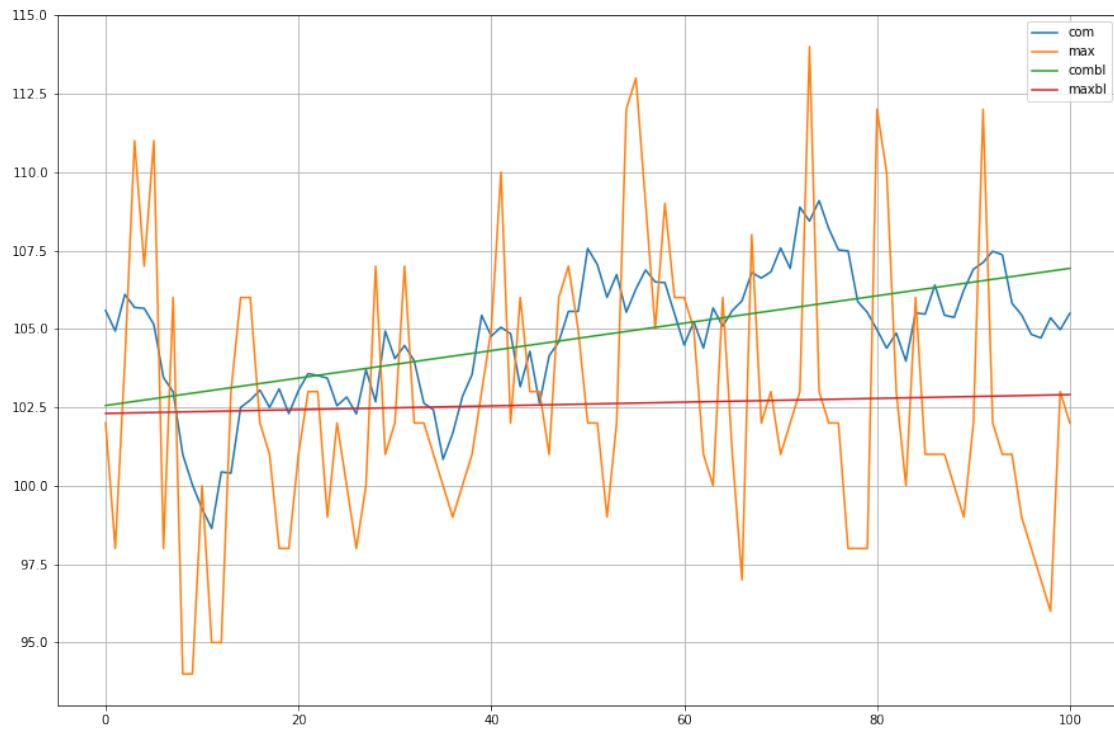


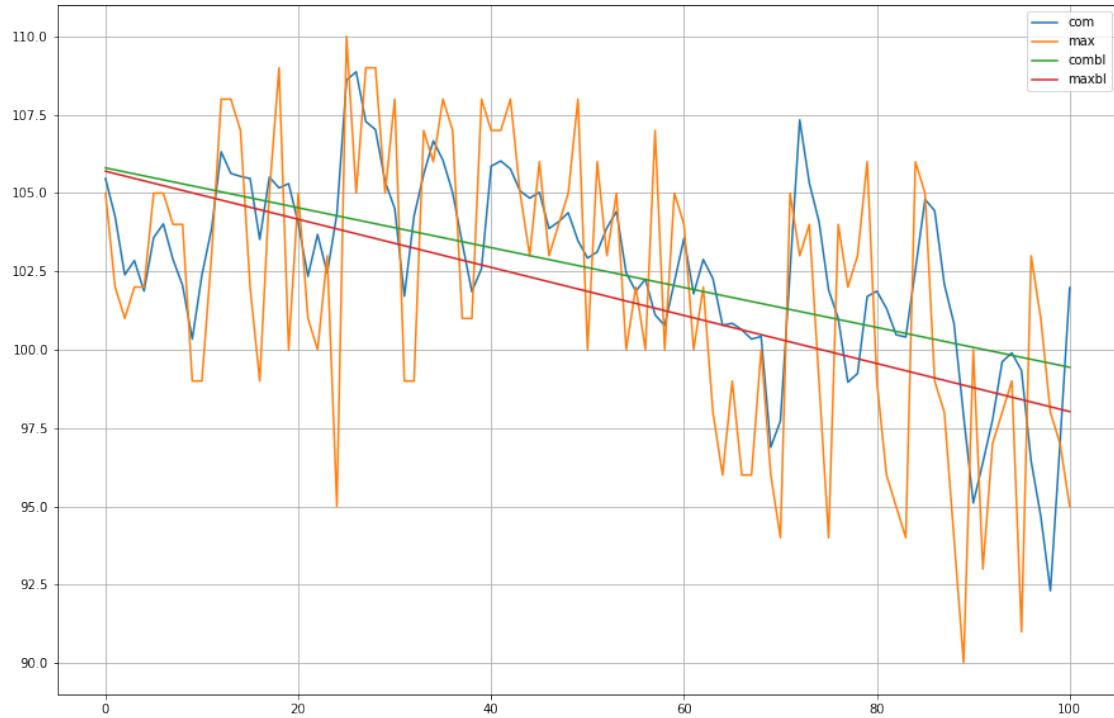












0.6 Check fine tuned parameters

```
[ ]: a = loadData('fine_filter_parameters2')
import yig_filter_model

ks = a['k'][0]
ms = a['m'][0]
filterBank = []
for i in range(len(ks)):
    filterBank.append(yig_controller_test.YigFilter(fMin[i], fMax[i], ms[i], ks[i], yc, i, new=True))

calMeas = skrf.network.Network('cal_through.s2p')

fix = yig_filter_model.SimpleS21Fixture(calMeas.f, calMeas.s[:, 1, 0])
```

```
[ ]: filter0Map, filter0TuningWords, filter0Frequencies, filter0SpanMap =
    measureYigFilter(filterBank[0])
```

```
[ ]: filter1Map, filter1TuningWords, filter1Frequencies, filter1SpanMap =
    measureYigFilter(filterBank[1])
```

```

[ ]: filter2Map, filter2TuningWords, filter2Frequencies, filter2SpanMap =_
    ↪measureYigFilter(filterBank[2])

[ ]: filter3Map, filter3TuningWords, filter3Frequencies, filter3SpanMap =_
    ↪measureYigFilter(filterBank[3])

[ ]: filter4Map, filter4TuningWords, filter4Frequencies, filter4SpanMap =_
    ↪measureYigFilter(filterBank[4])

[ ]: filter5Map, filter5TuningWords, filter5Frequencies, filter5SpanMap =_
    ↪measureYigFilter(filterBank[5])

[ ]: saveDict={'filter0Map':filter0Map, 'filter0TuningWords':filter0TuningWords,_
    ↪'filter0Frequencies':filter0Frequencies, 'filter0SpanMap':filter0SpanMap,
    'filter1Map':filter1Map, 'filter1TuningWords':filter1TuningWords,_
    ↪'filter1Frequencies':filter1Frequencies, 'filter1SpanMap':filter1SpanMap,
    'filter2Map':filter2Map, 'filter2TuningWords':filter2TuningWords,_
    ↪'filter2Frequencies':filter2Frequencies, 'filter2SpanMap':filter2SpanMap,
    'filter3Map':filter3Map, 'filter3TuningWords':filter3TuningWords,_
    ↪'filter3Frequencies':filter3Frequencies, 'filter3SpanMap':filter3SpanMap,
    'filter4Map':filter4Map, 'filter4TuningWords':filter4TuningWords,_
    ↪'filter4Frequencies':filter4Frequencies, 'filter4SpanMap':filter4SpanMap,
    'filter5Map':filter5Map, 'filter5TuningWords':filter5TuningWords,_
    ↪'filter5Frequencies':filter5Frequencies, 'filter5SpanMap':filter5SpanMap, }
saveData('fine_tuning_fine_measurements', saveDict)

[ ]: ftm = loadData('fine_tuning_fine_measurements')

coarseData = loadData('fine_filter_parameters')

kcoarse = coarseData['k'][0]
mcoarse = coarseData['m'][0]
kfine = kcoarse
mfine = mcoarse

for i in range(6):
    k, m = fineTuneAnalysis(ftm, i)
    print("filter",i,"Coarse k", kcoarse[i], "[MHZ/lsb]", "coarse m",_
    ↪mcoarse[i], "[MHz]")
    kfine[i] = kcoarse[i] + k
    mfine[i] = mcoarse[i] + m
    print("filter",i,"fine k", kfine[i], "[MHZ/lsb]", "cfine m", mfine[i],_
    ↪"[MHz]")

fineTuned={'k':kfine, 'm':mfine}
#saveData('fine_filter_parameters', fineTuned)

```

0.7 Upload coefficients

```
[ ]: a = loadData('fine_filter_parameters2')

ks = a['k'][0]
ms = a['m'][0]
#filterBank = []
for i in range(len(ks)):
    print
    yigDriver.writeFilterSlope(i, ks[i])
    time.sleep(0.2)
    yigDriver.writeFilterOffset(i, ms[i])
    time.sleep(0.2)
    yigDriver.writeFilterLowLim(i, fMin[i])
    time.sleep(0.2)
    yigDriver.writeFilterHighLim(i, fMax[i])
    time.sleep(0.2)
    #print(yigDriver.dev.read())
yigDriver.save()
time.sleep(0.2)
#filterBank.append(yig_controller_test.YigFilter(fMin[i], fMax[i], ms[i]*1e6, ks[i]*1e6, yc, i))

#fc = 0.7e9
#spanHalf=500e6/2;
#vna.setStartFrequency(fc-spanHalf)
#vna.setStopFrequency(fc+spanHalf)
#filterBank[0].tuneTo(fc, channel=yigDriver)
#switch.set(1)
#vna.readSParameter('S21')
#yc.yigA.set(6,32700)
```