yig_learning

May 17, 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, 16])*1e9
yigDriver=yc.yigA
switch=yc.switchA


hardwareDescription='Yig filters moved to source side of transistor and 9v yig␣
 ↪voltage regulator. Added lower tempco resistors to ch4&5 added gapfiller to␣
 ↪cool all shunts except ch0'
hardwareRevision='2.3'

#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, 128, dtype=np.int16)
yigChs = range(6)

zeros = []
channels=[]
currentMtx = None;
for c in yigChs:
    yigDriver.set(c, wr[0])
    time.sleep(5);
    current=[]
    for w in wr:
        yigDriver.set(c, w);
        time.sleep(0.5)
        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});
```

[4]:
```
import scipy.io as sio
saveData('yig_filter_driver_current_data', {'zeros':zeros, 'channels':channels,␣
 ↪'wr':wr})
```

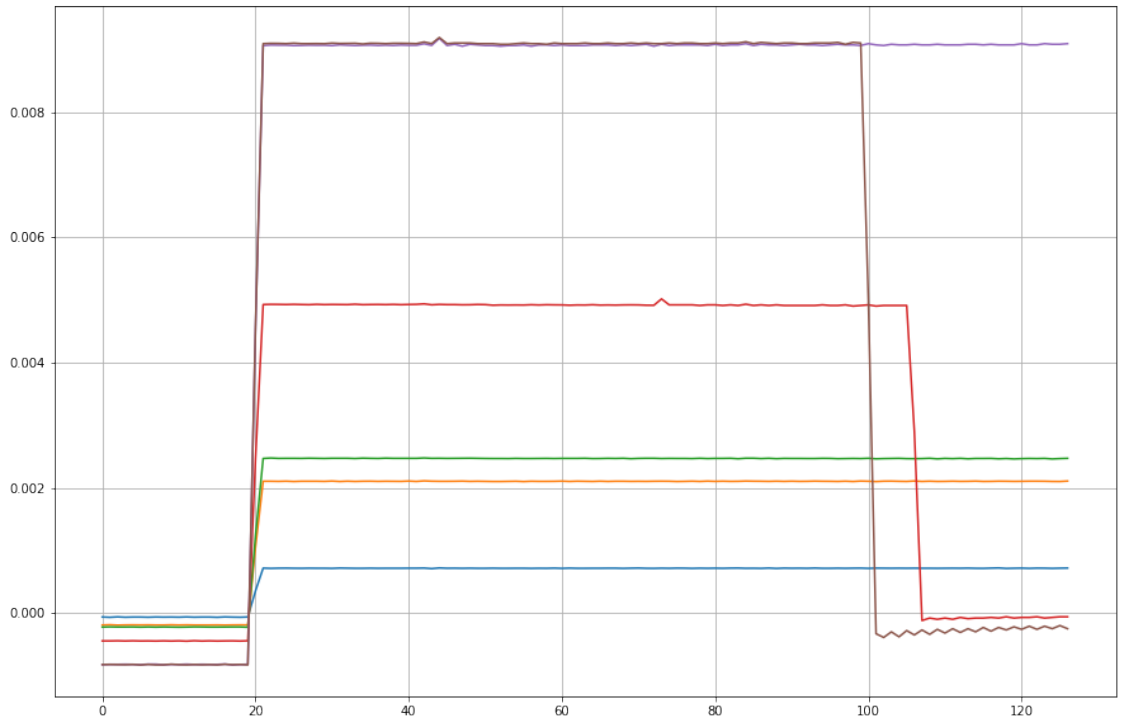
[5]:
```
d=loadData('yig_filter_driver_current_data')
zeros=d['zeros'][0]
channels=d['channels']
wr=d['wr'][0]
```

[6]:
```
print(zeros)
```

```
[-1.5e-05 -1.4e-05 -1.4e-05 -1.6e-05 -1.8e-05 -1.9e-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.plot(avgd)
    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 0.006476 to 0.081924 [A] in control word range -22448
- 32250 [LSB]
Filter 1 has current from 0.019050 to 0.240944 [A] in control word range -22448
- 32250 [LSB]
Filter 2 has current from 0.022368 to 0.282767 [A] in control word range -22448
- 32250 [LSB]
Filter 3 has current from 0.044618 to 0.465286 [A] in control word range -22448
- 21930 [LSB]
Filter 4 has current from 0.082102 to 1.038948 [A] in control word range -22448
- 32250 [LSB]
Filter 5 has current from 0.082379 to 0.805509 [A] in control word range -22448
- 18834 [LSB]
```

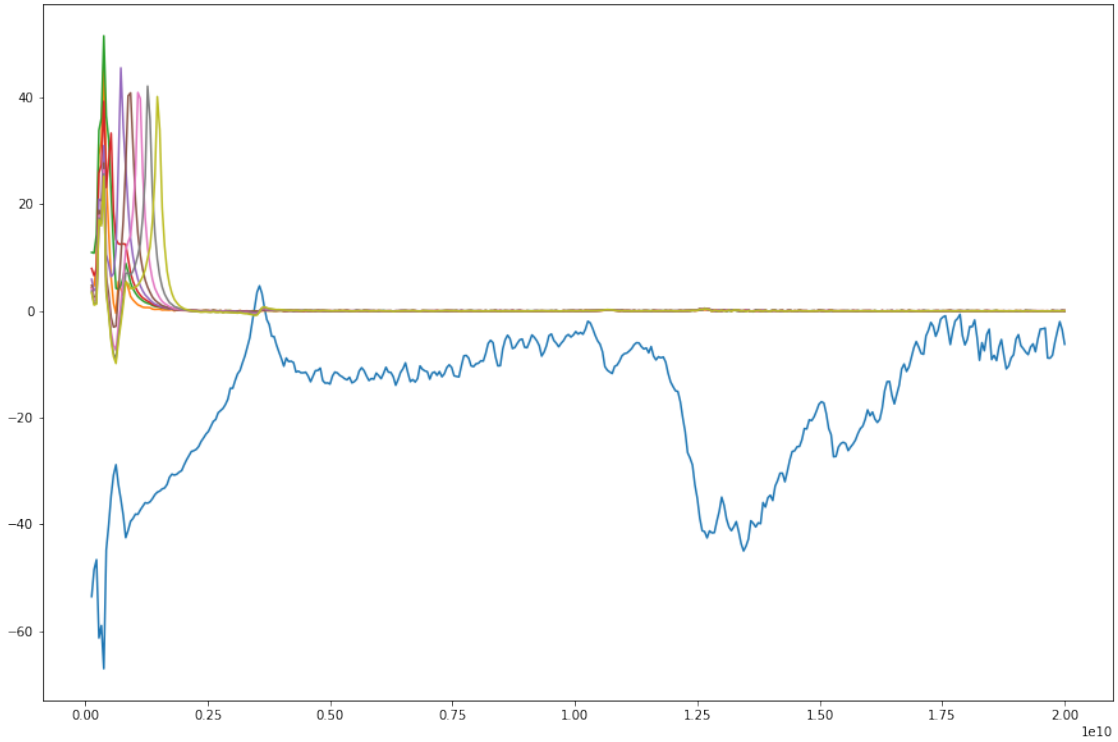


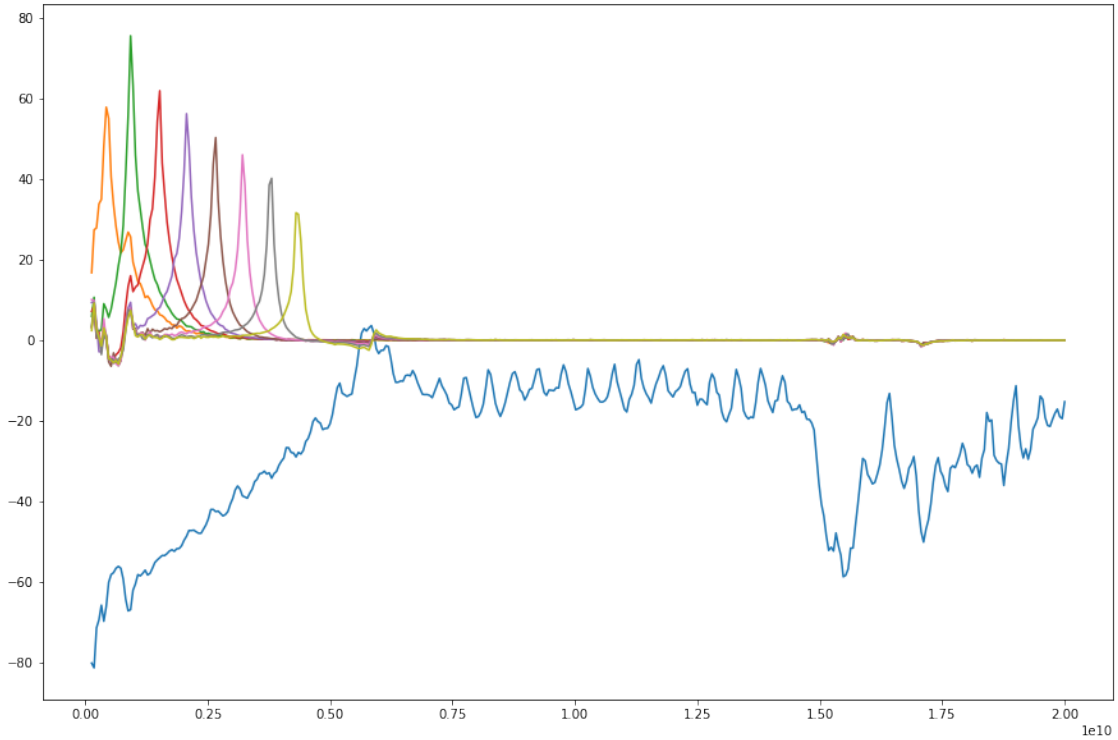
0.3 Coarse tuning

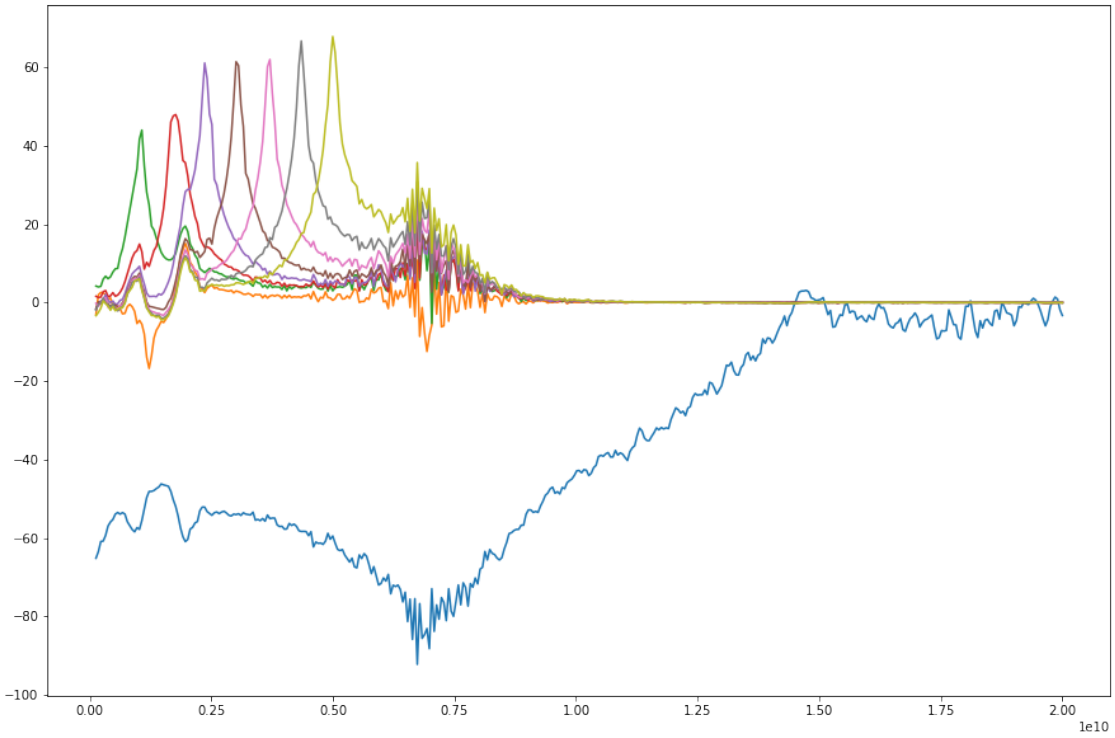
[8]:
```
vna.setStartFrequency(130e6)
vna.setStopFrequency(20e9)
vna.setPoints(401)

fRanges=[]
for i in range(len(zeros)):
    #print("Filter", i )
```

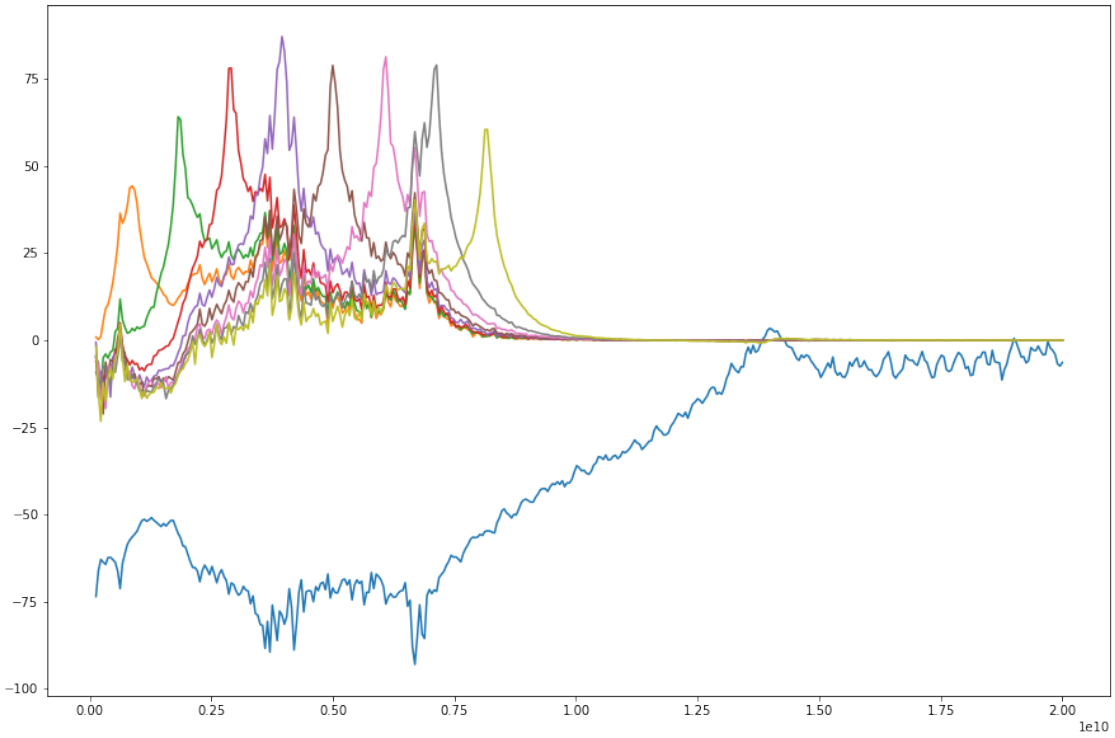
```
    fp = fParams[i]
    switch.set(i+1)
    yigDriver.set(6, 0)
    fax = vna.frequencies()
    base = vna.readSParameter('S21')
    plt.plot(fax, 20*np.log10(np.abs(base)))
    sr = np.linspace(fp.lw, fp.hw, 8)
    freqs=[]

    for w in sr:
        yigDriver.set(i, int(w))
        time.sleep(1)
        t = vna.readSParameter('S21')
        #traces.append(t)
        delt = np.abs(t)/np.abs(base)
        freqs.append(fax[np.argmax(delt)])
        plt.plot(fax, 20*np.log10(delt))
    mif=np.min(freqs)
    maf=np.max(freqs)
    #print(freqs)
    #print(np.diff(freqs))
    fRanges.append((mif, maf))

    plt.figure()

for fmi, fma in fRanges:
    print("%f to %f [GHz]"%(fmi/1e9, fma/1e9))

import scipy.io as sio
saveData('yig_filter_driver_coarse_frequency', {'fRanges':fRanges})
```

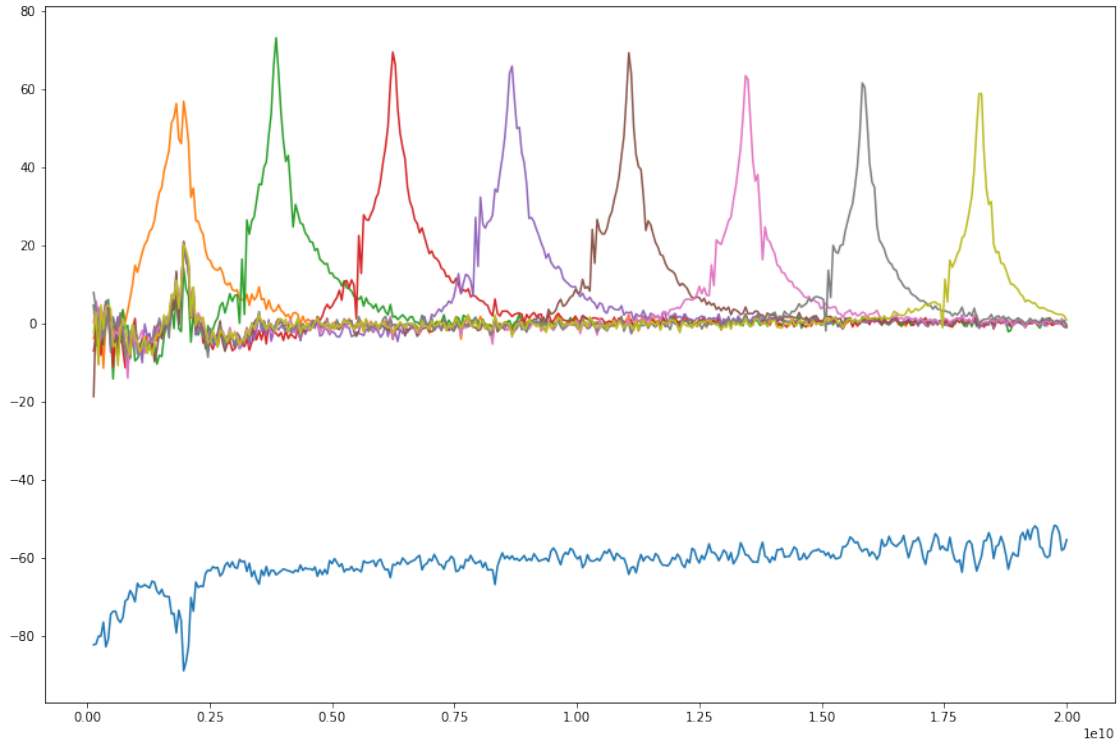
```
0.378375 to 1.471225 [GHz]
0.428050 to 4.302700 [GHz]
1.073825 to 4.998150 [GHz]
0.875125 to 8.177350 [GHz]
1.967975 to 18.261375 [GHz]
1.520900 to 16.771125 [GHz]
```

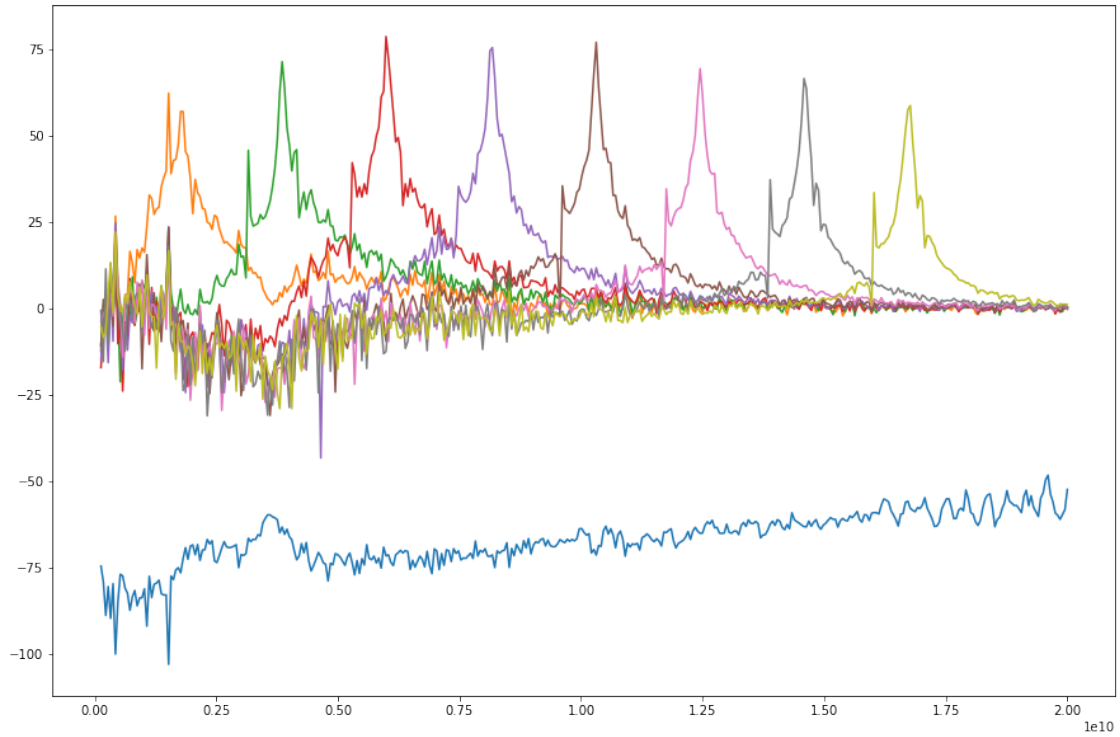

40
20
0
−20
−40
−60
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e10


80
60
40
20
0
−20
−40
−60
−80
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e10


60
40
20
0
−20
−40
−60
−80
−100
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e10


75
50
25
0
−25
−50
−75
−100
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e10


80
60
40
20
0
−20
−40
−60
−80
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e10

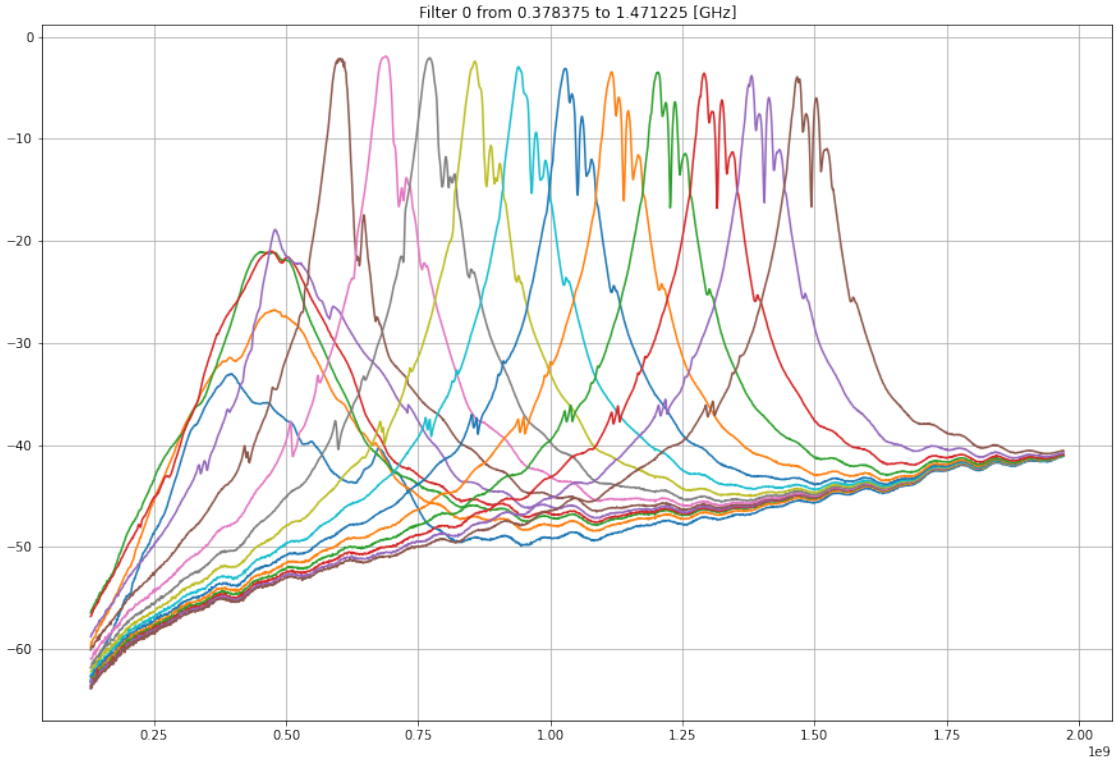

75
50
25
0
−25
−50
−75
−100
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e10

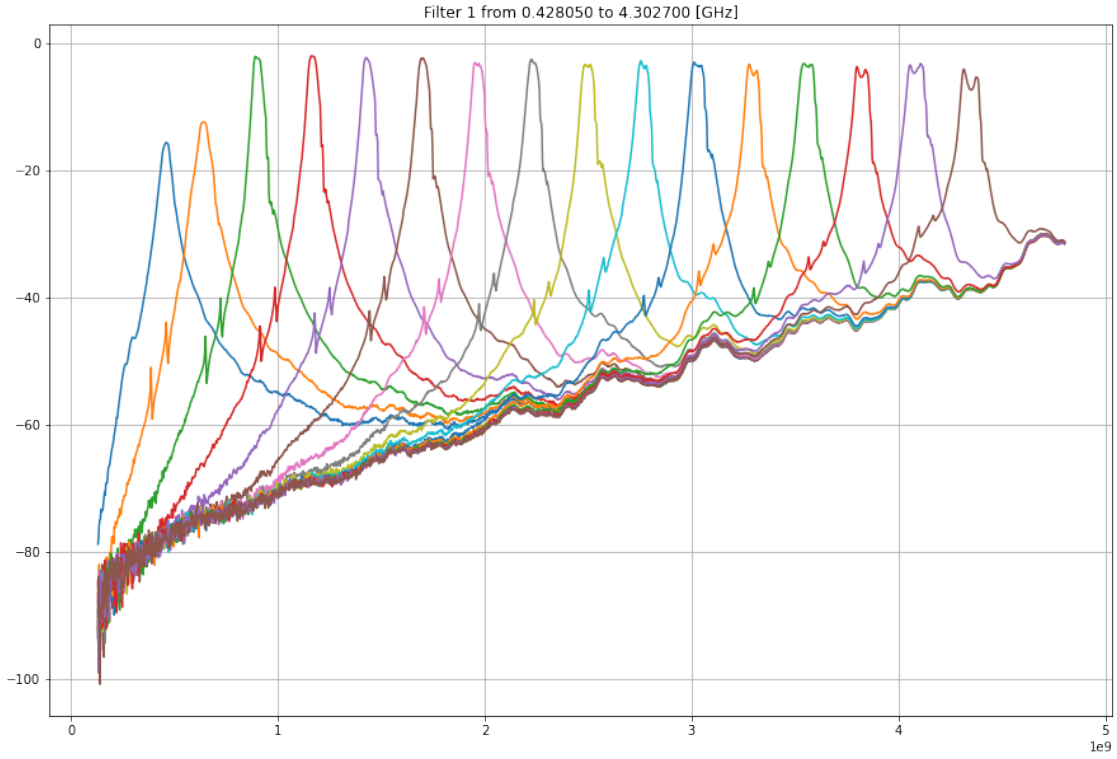
```
<Figure size 1080x720 with 0 Axes>
```

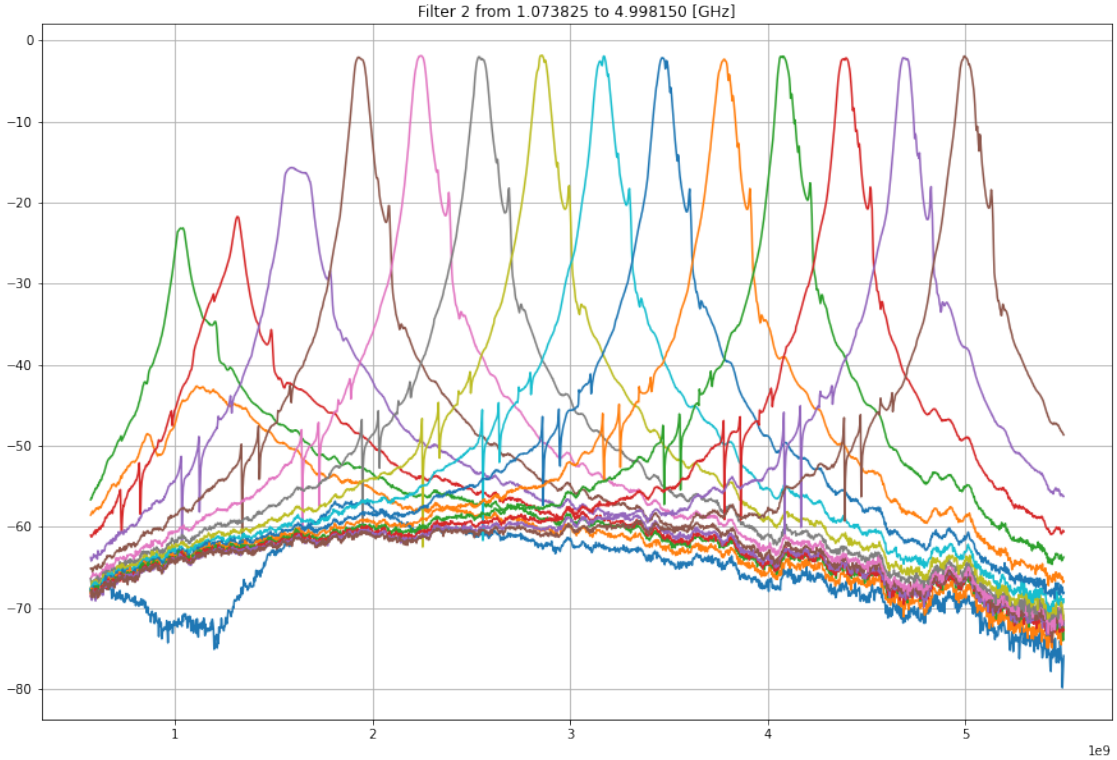
[9]:
```
#Get calibration data
import os.path

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

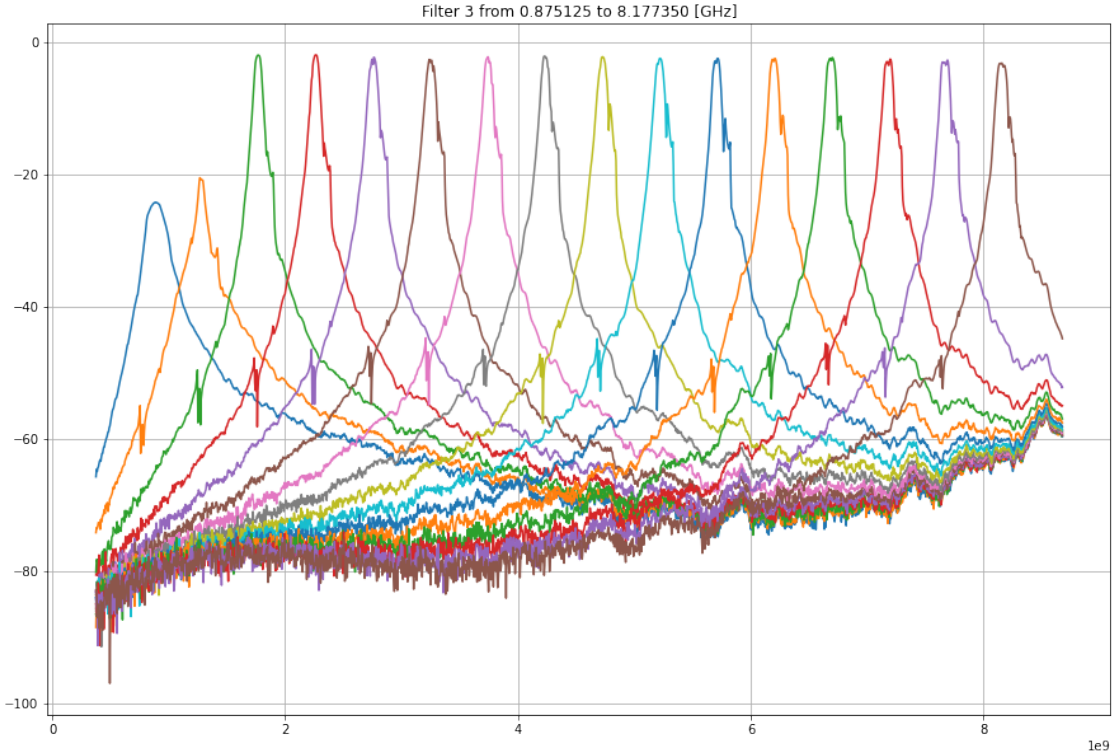
[10]:
```
import skrf.network

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']=[]
```

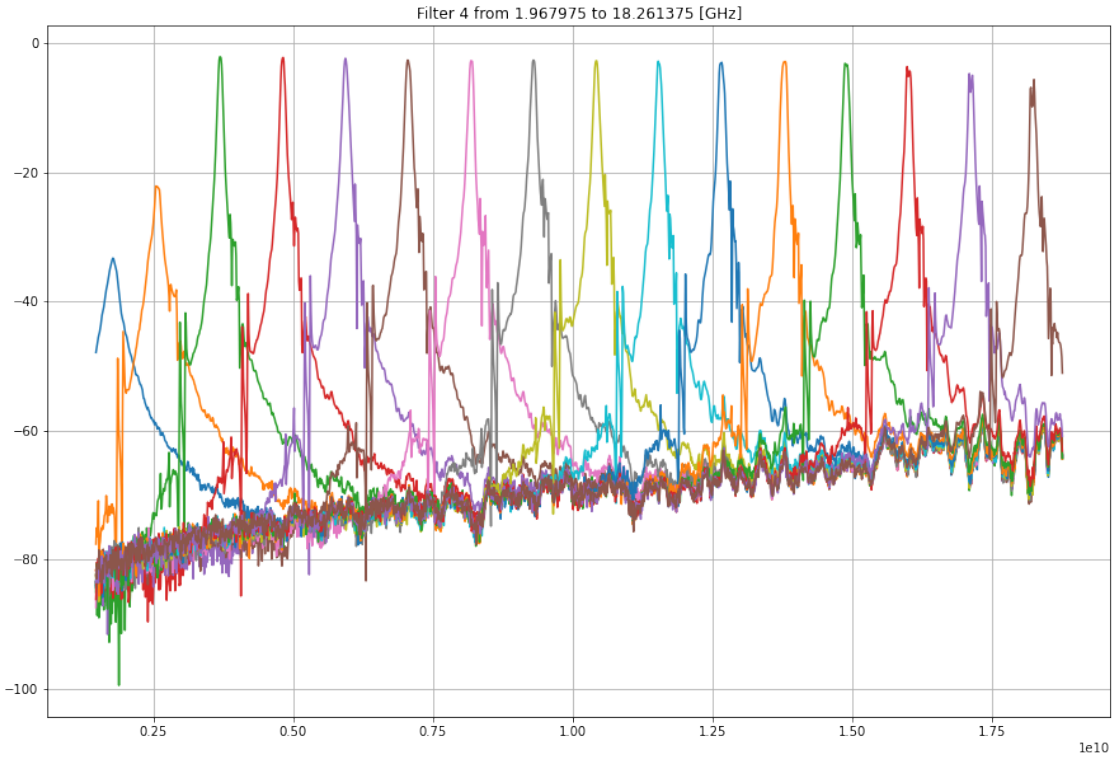
```
    for w in vwr:
        yigDriver.set(i, int(w))
        time.sleep(0.5)
        tr = vna.readSParameter('S21')
        fr = vna.frequencies()
        cai=inter(fr, calPar.f, np.abs(calPar.s[:,1,0]))
        #nf, npar = norm(calPar.f, np.abs(calPar.s[:,0,1]), fr, np.abs(tr))
        #plt.plot(nf, 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)
```

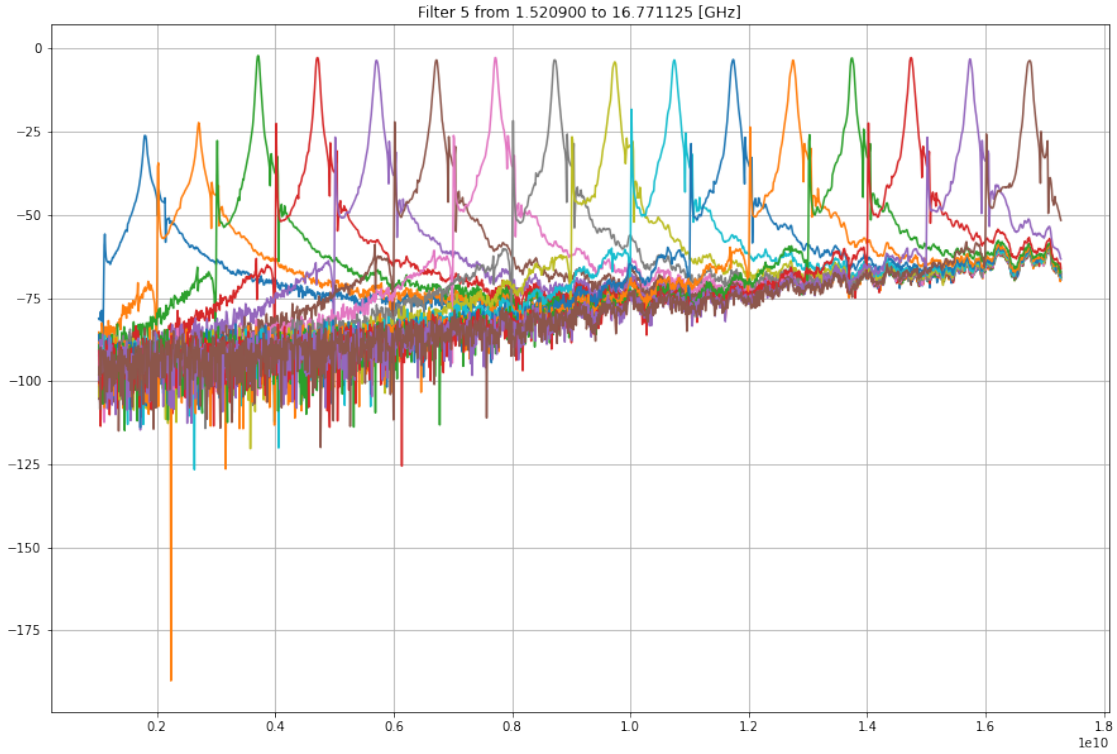



Filter 1 from 0.428050 to 4.302700 [GHz]
0
−20
−40
−60
−80
−100
0
1
2
3
4
5
1e9


Filter 2 from 1.073825 to 4.998150 [GHz]
0
−10
−20
−30
−40
−50
−60
−70
−80
1
2
3
4
5
1e9


Filter 3 from 0.875125 to 8.177350 [GHz]
0
−20
−40
−60
−80
−100
0
2
4
6
8
1e9


Filter 4 from 1.967975 to 18.261375 [GHz]
0
−20
−40
−60
−80
−100
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1e10



[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']
    fildat=[]
    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 1162 is k 0.024 [MHz/LSB] m is 699.073[LSB]
Filter parameter for filter 1434 is k 0.072 [MHz/LSB] m is 2000.848[LSB]
Filter parameter for filter 1437 is k 0.085 [MHz/LSB] m is 2277.881[LSB]
Filter parameter for filter 1500 is k 0.167 [MHz/LSB] m is 4523.037[LSB]
Filter parameter for filter 1553 is k 0.306 [MHz/LSB] m is 8348.592[LSB]
Filter parameter for filter 1548 is k 0.365 [MHz/LSB] m is 9886.749[LSB]
```

0.4 Measure drift

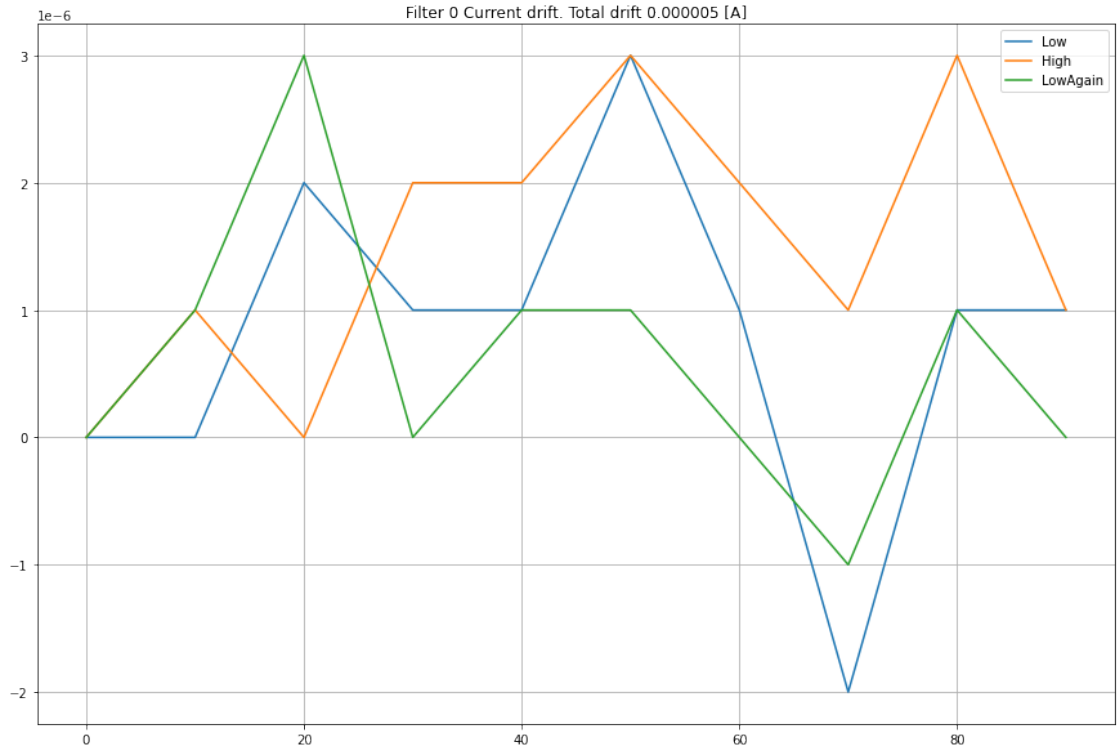
[14]:
```
import time
coarseFilter=loadData('coarse_filter_parameters')
mvec=coarseFilter['m'][0]
kvec=coarseFilter['k'][0]


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

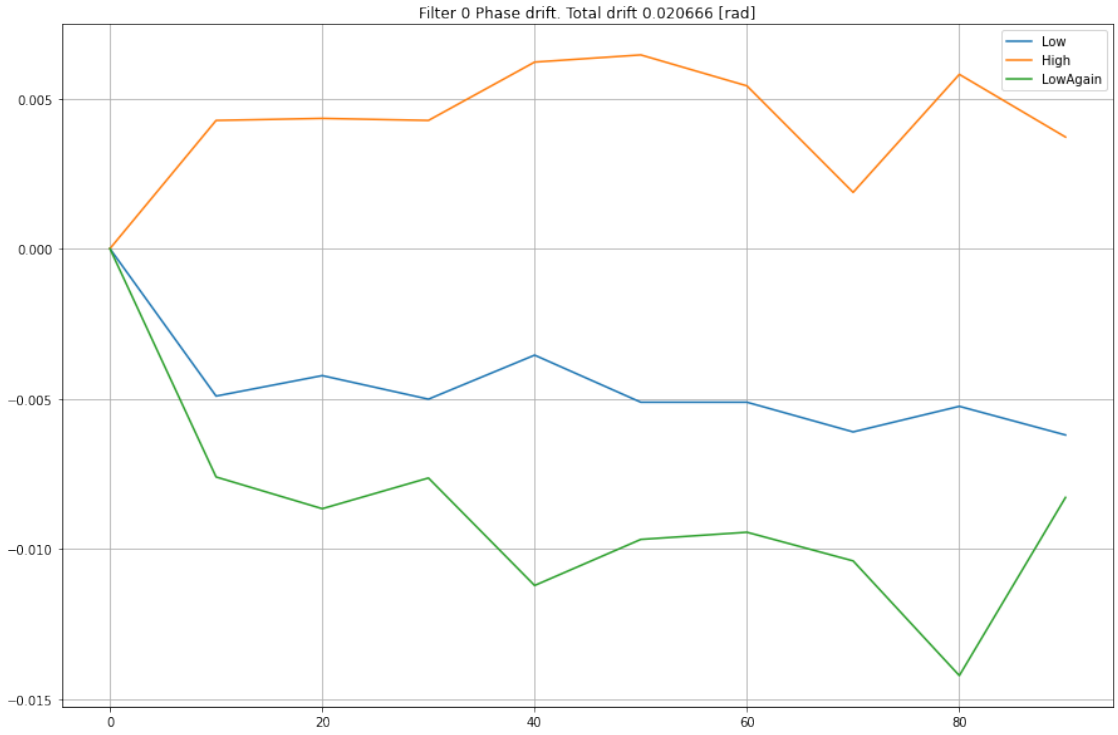
```
s21Drift=None
currentDrift=None

dataDict={}

def measureDrifts():
    for a in range(10):
        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
```

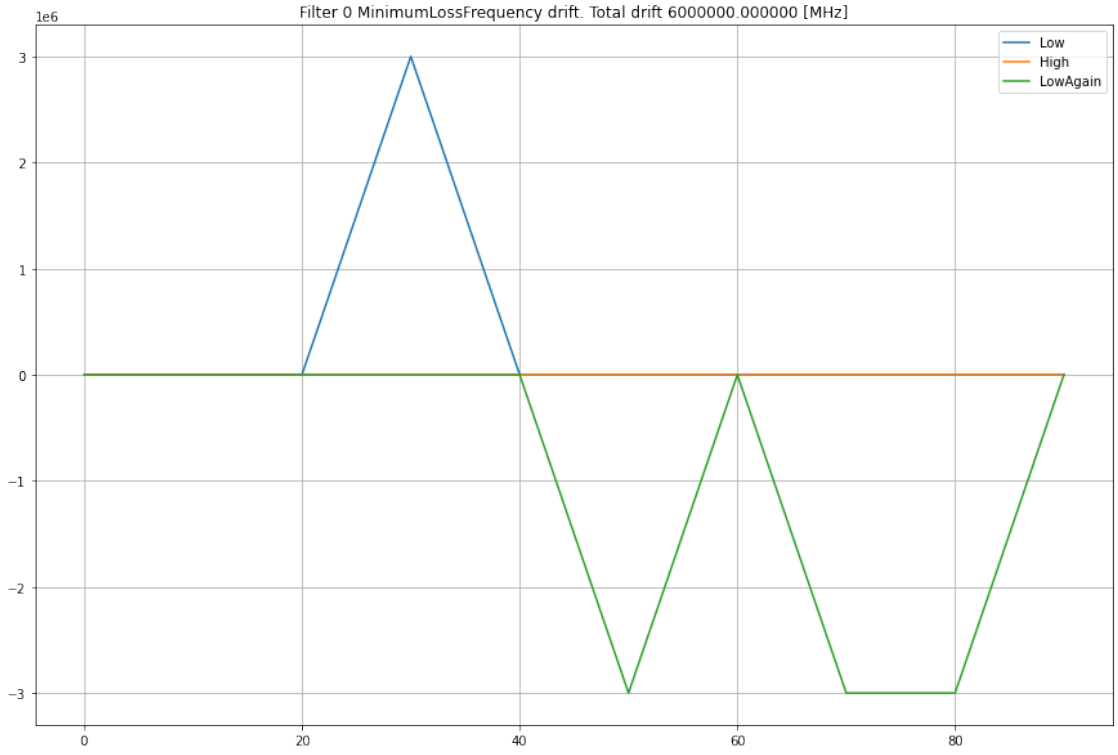
```
        dataDict[keyBase+keySub+'Phase']=phase
        dataDict[keyBase+keySub+'SMatrix']=sMat
        dataDict[keyBase+keySub+'MinimumLossFrequency']=fmax


saveData('filter_drift', dataDict)
```

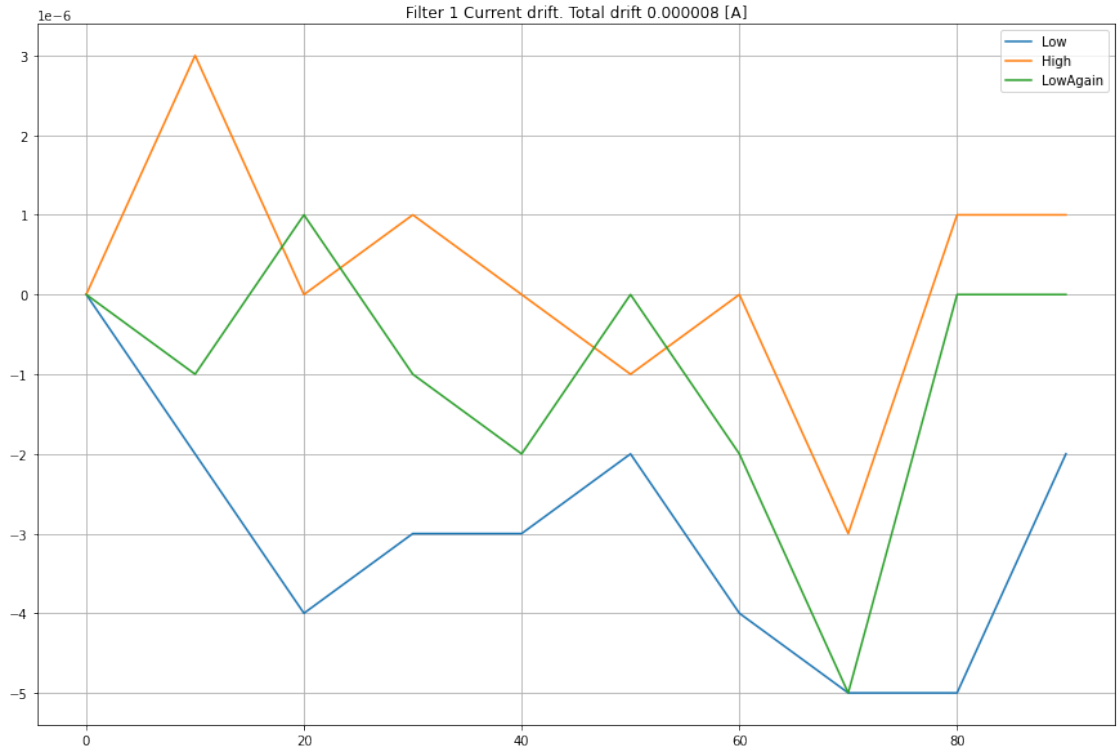
```
Measuring filter 0
Measuring filter 1
Measuring filter 2
Measuring filter 3
Measuring filter 4
Measuring filter 5
```

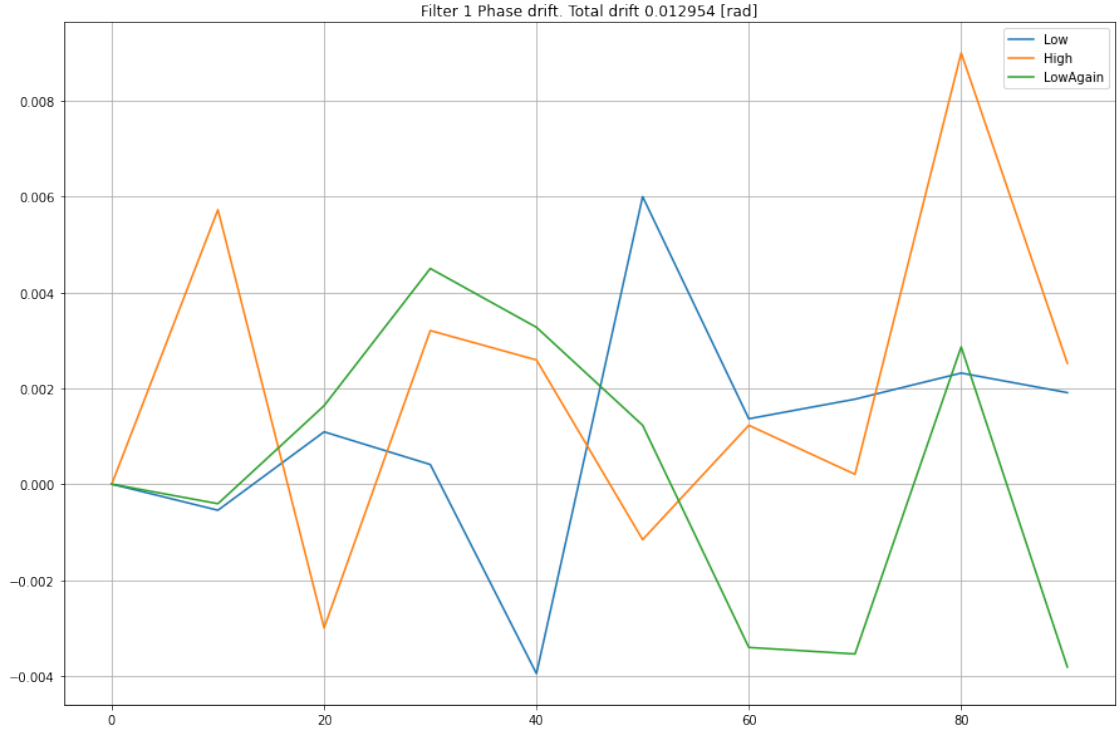
[15]:
```
dd = loadData('filter_drift')

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')
```

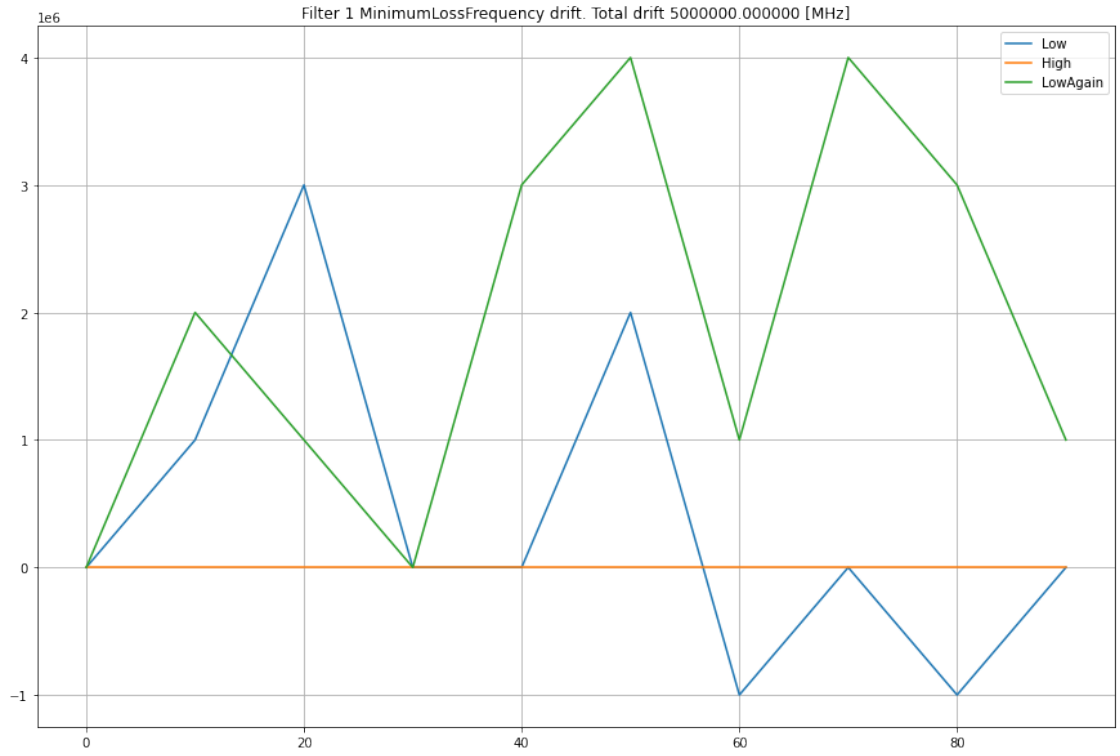

Filter 0 Current drift. Total drift 0.000005 [A]
1e−6
3
2
1
0
−1
−2
0
20
40
60
80
Low
High
LowAgain

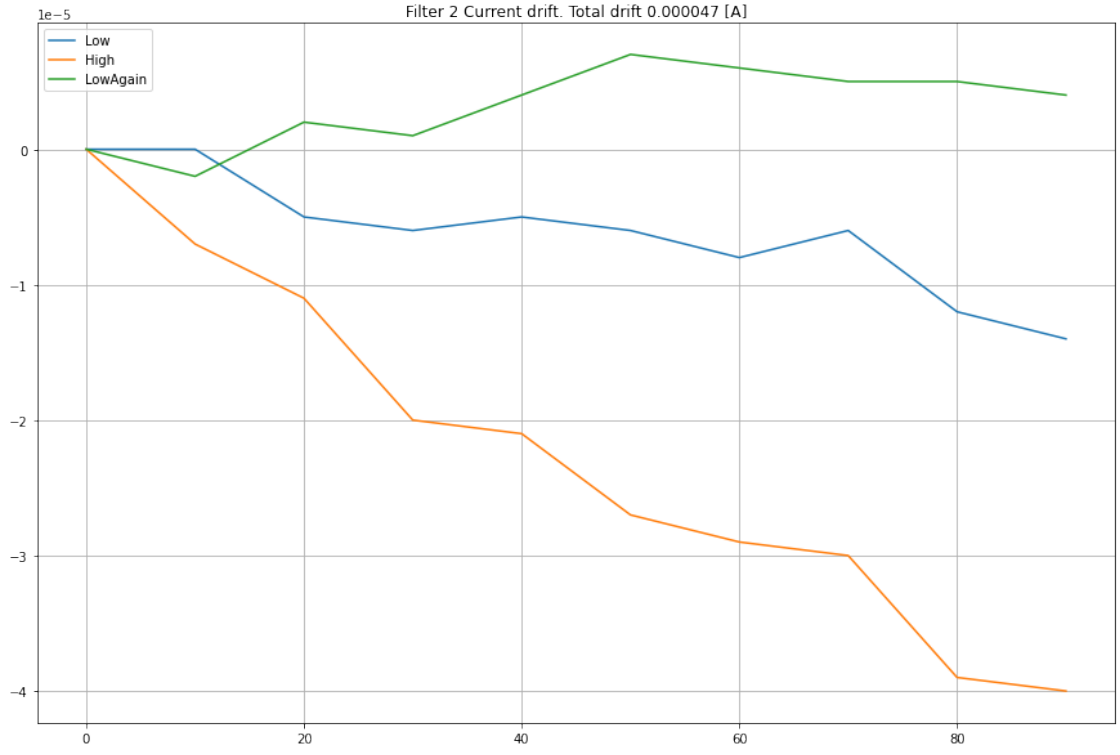

Filter 0 Phase drift. Total drift 0.020666 [rad]
Low
High
LowAgain
0.005
0.000
−0.005
−0.010
−0.015
0
20
40
60
80

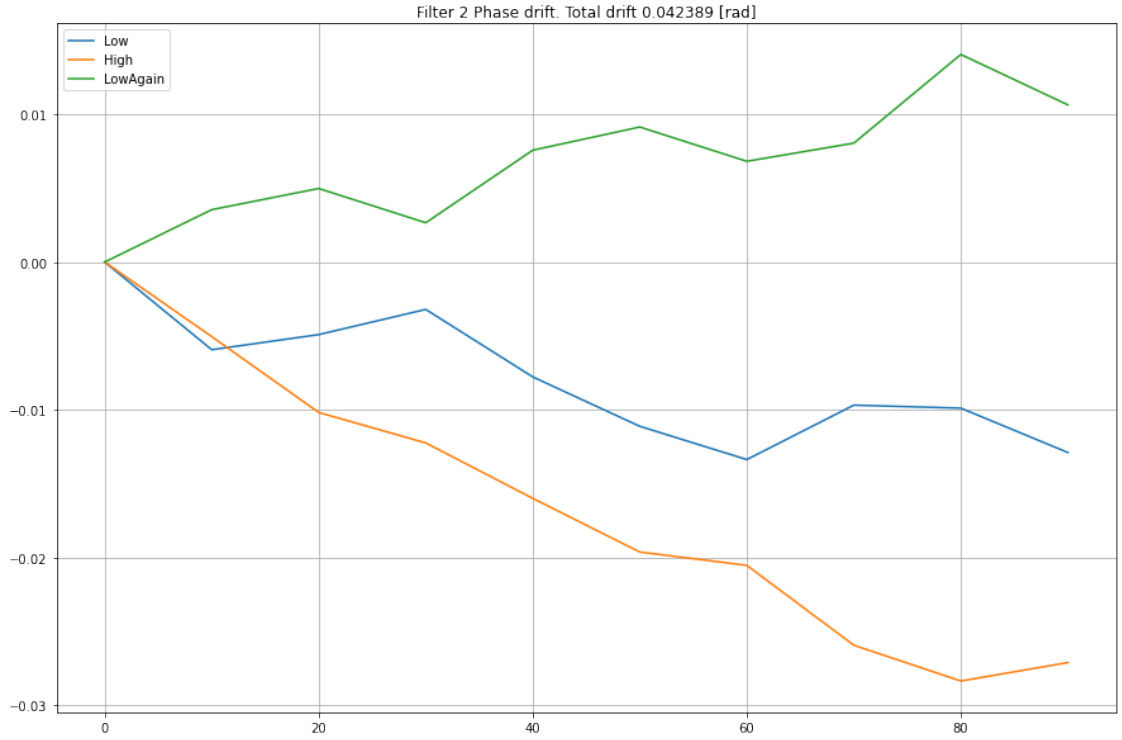

Filter 0 MinimumLossFrequency drift. Total drift 6000000.000000 [MHz]
1e6
Low
High
LowAgain
3
2
1
0
−1
−2
−3
0
20
40
60
80

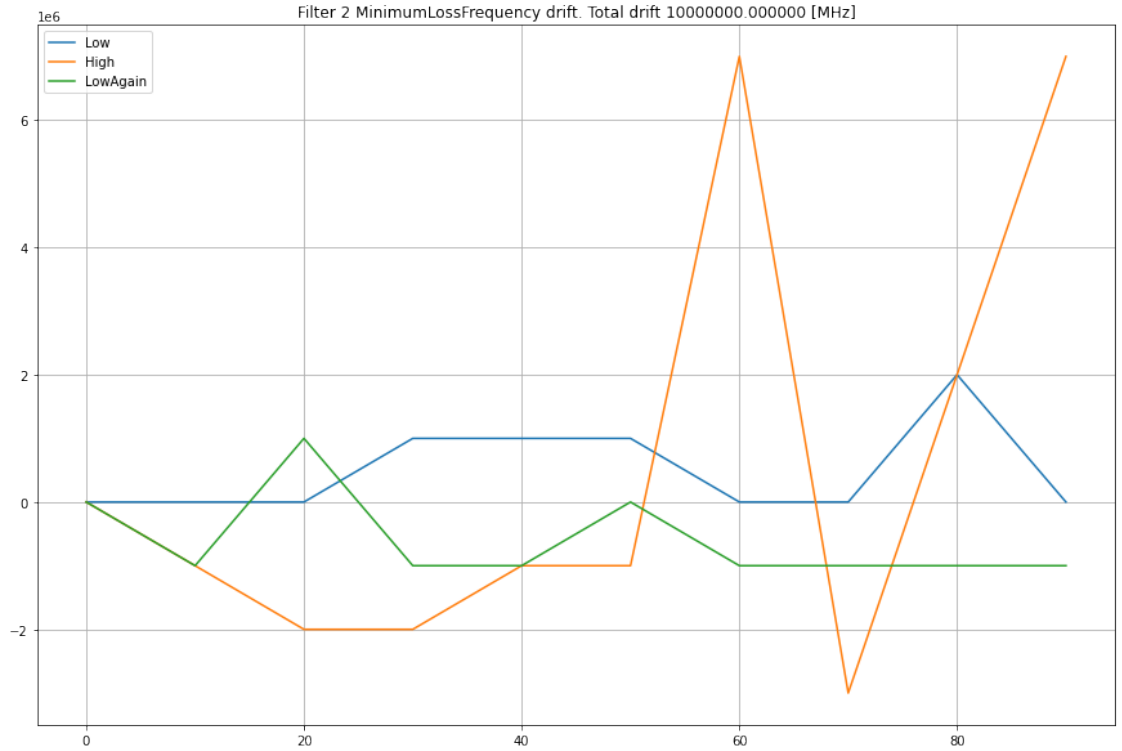

Filter 1 Current drift. Total drift 0.000008 [A]
1e−6
3
2
1
0
−1
−2
−3
−4
−5
0
20
40
60
80
Low
High
LowAgain

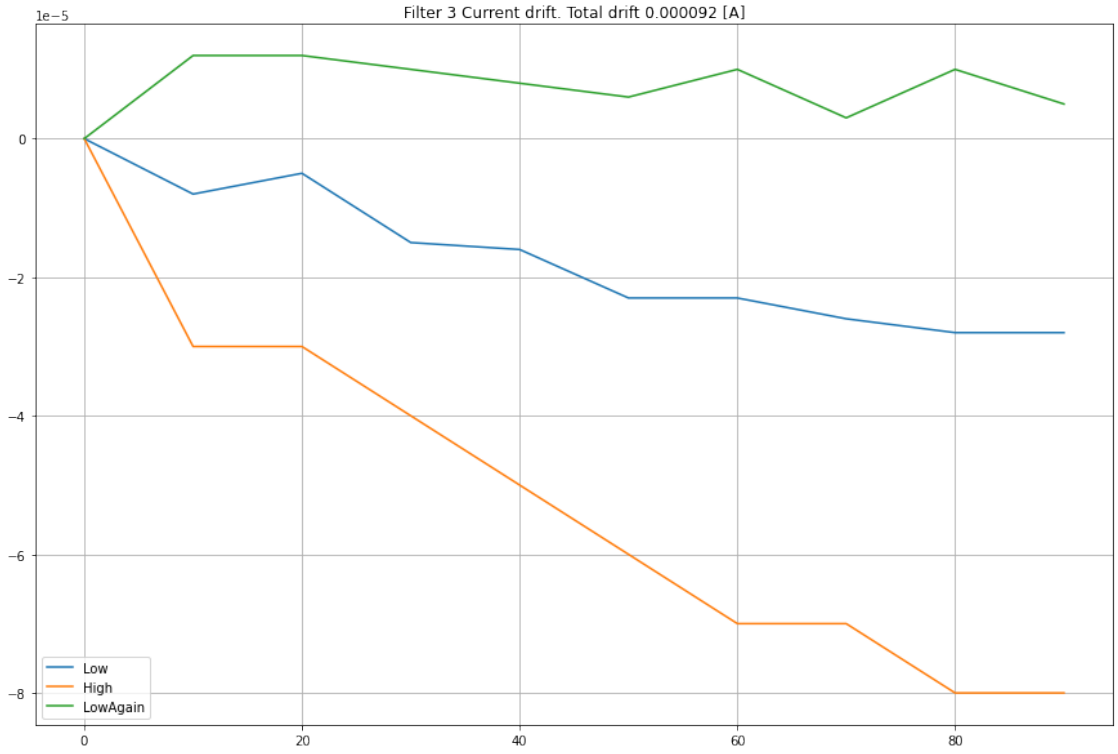

Filter 1 Phase drift. Total drift 0.012954 [rad]
0.008
0.006
0.004
0.002
0.000
−0.002
−0.004
0
20
40
60
80
Low
High
LowAgain

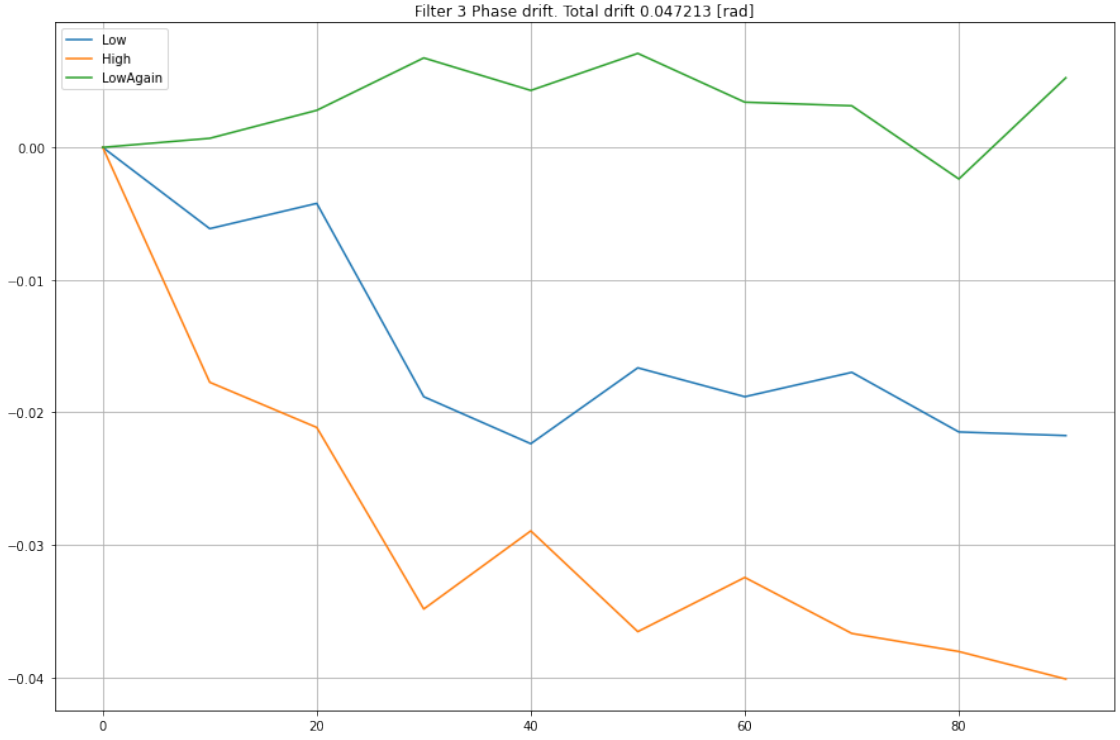

Filter 1 MinimumLossFrequency drift. Total drift 5000000.000000 [MHz]
1e6
Low
High
LowAgain
4
3
2
1
0
−1
0
20
40
60
80

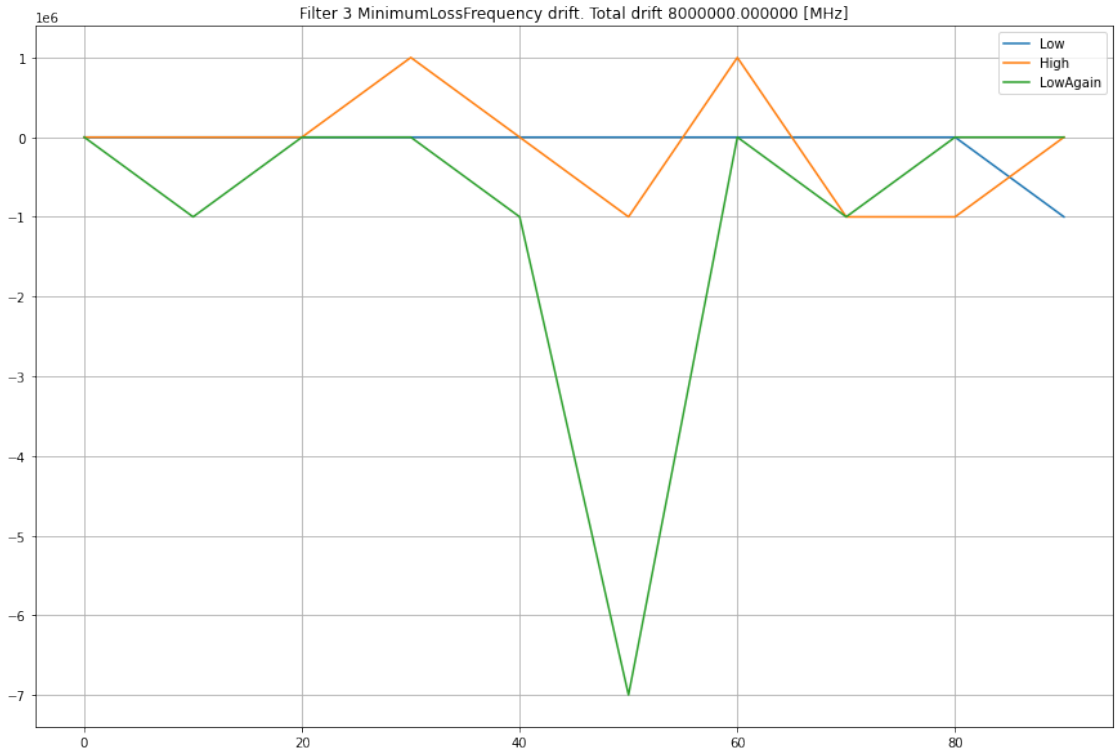

Filter 2 Current drift. Total drift 0.000047 [A]
1e−5
Low
High
LowAgain
0
−1
−2
−3
−4
0
20
40
60
80

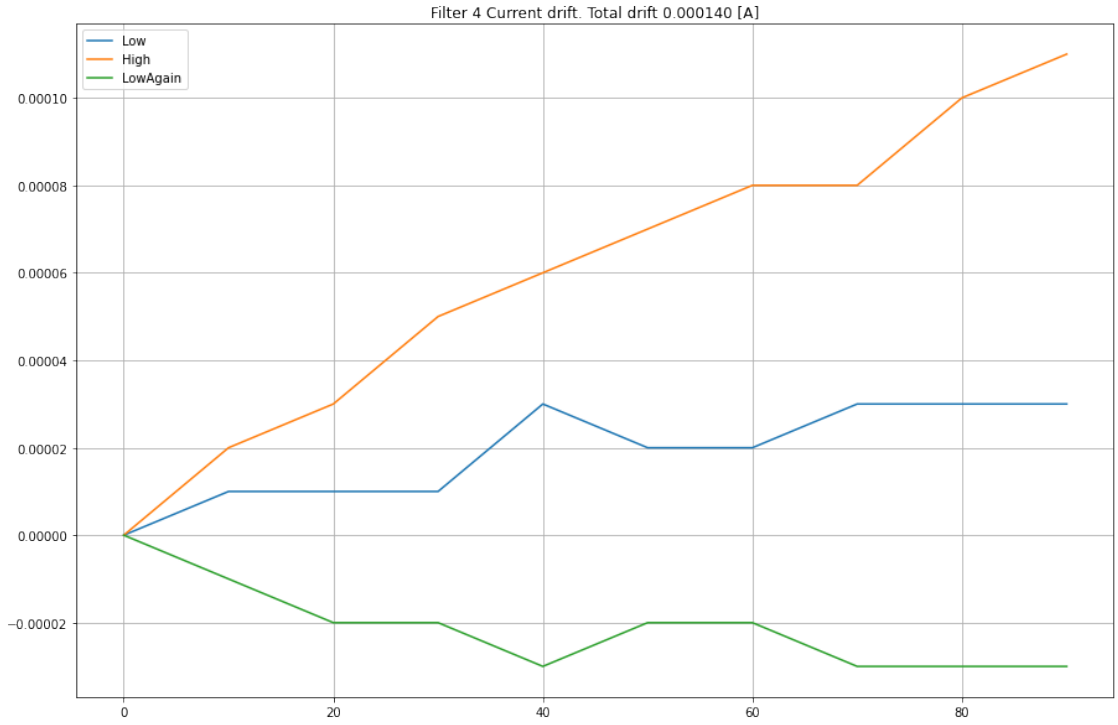

Filter 2 Phase drift. Total drift 0.042389 [rad]
Low
High
LowAgain
0.01
0.00
−0.01
−0.02
−0.03
0
20
40
60
80

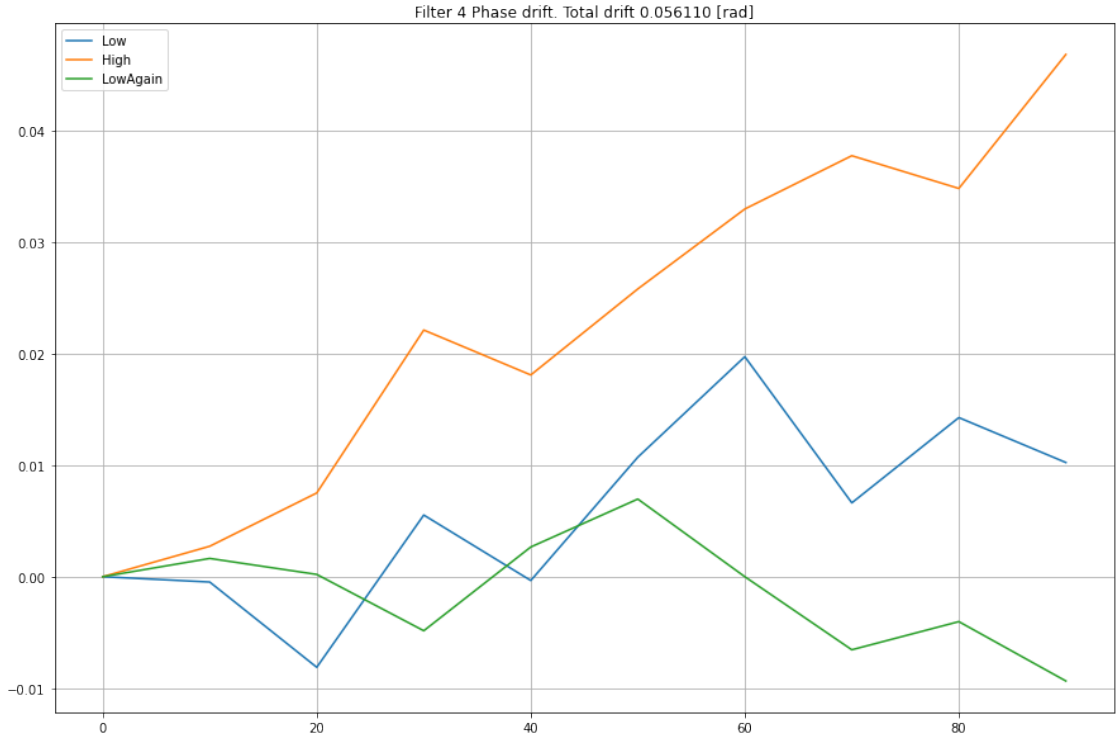

Filter 2 MinimumLossFrequency drift. Total drift 10000000.000000 [MHz]
1e6
Low
High
LowAgain
6
4
2
0
−2
0
20
40
60
80

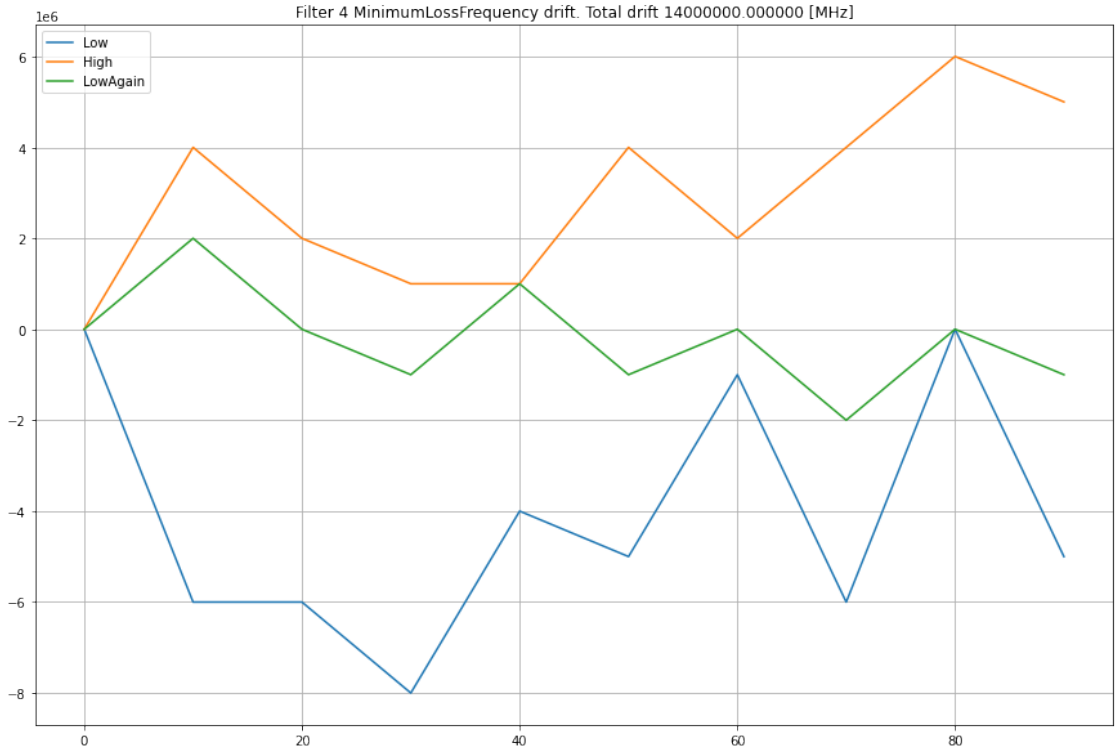

Filter 3 Current drift. Total drift 0.000092 [A]
1e−5
0
−2
−4
−6
−8
0
20
40
60
80
Low
High
LowAgain

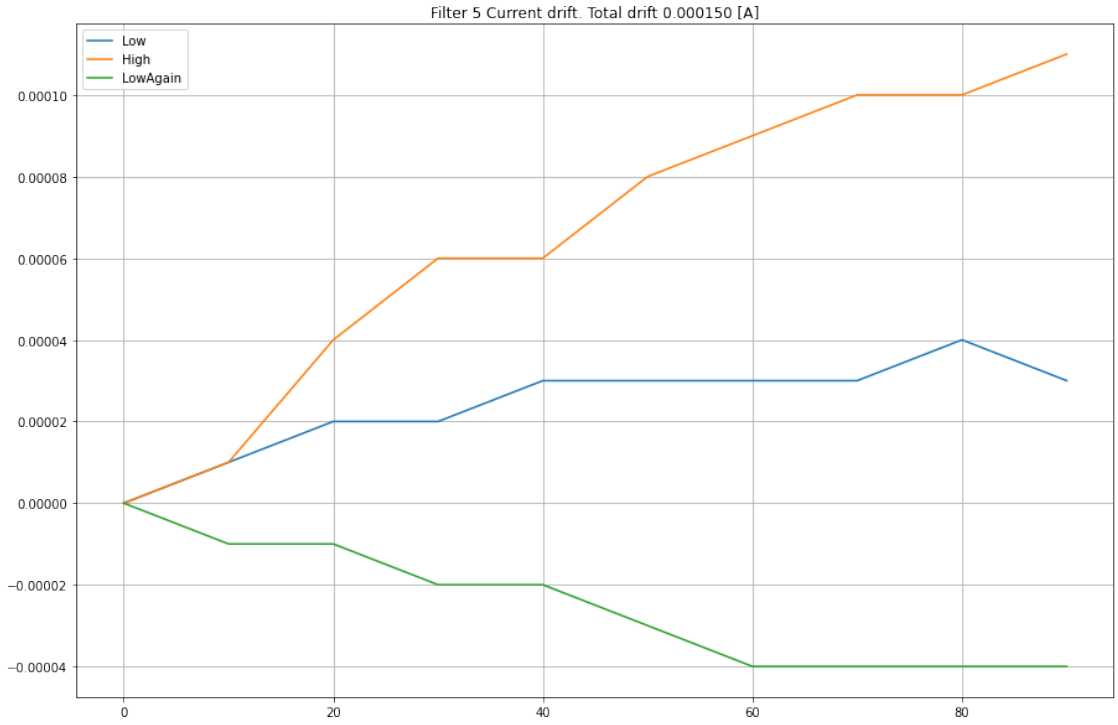

Filter 3 Phase drift. Total drift 0.047213 [rad]
0.00
−0.01
−0.02
−0.03
−0.04
0
20
40
60
80
Low
High
LowAgain

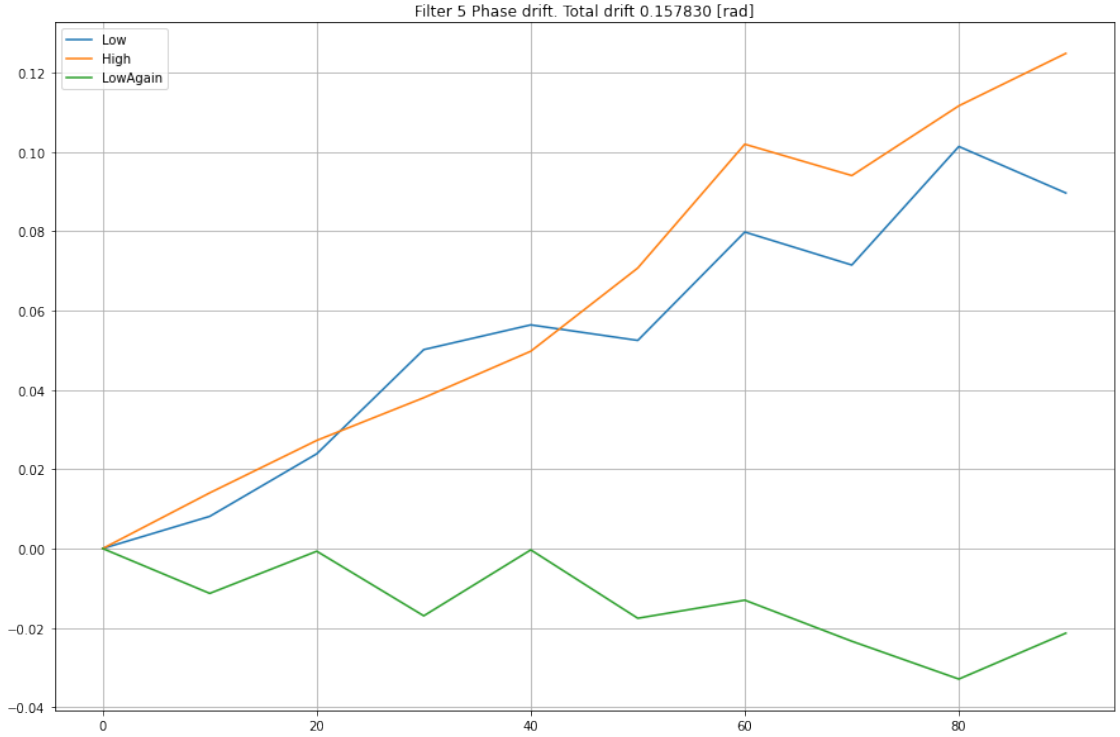

Filter 3 MinimumLossFrequency drift. Total drift 8000000.000000 [MHz]
1e6
Low
High
LowAgain
1
0
−1
−2
−3
−4
−5
−6
−7
0
20
40
60
80

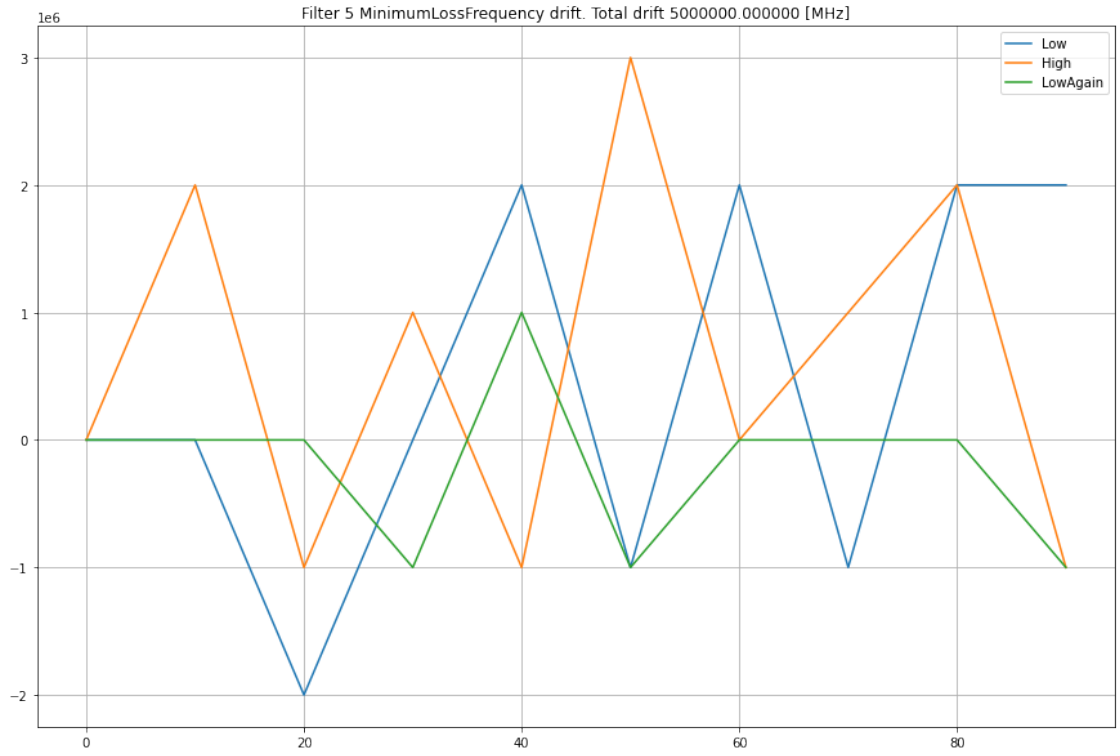

Filter 4 Current drift. Total drift 0.000140 [A]
Low
High
LowAgain
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
−0.00002
0
20
40
60
80


Filter 4 Phase drift. Total drift 0.056110 [rad]
Low
High
LowAgain
0.04
0.03
0.02
0.01
0.00
−0.01
0
20
40
60
80


Filter 4 MinimumLossFrequency drift. Total drift 14000000.000000 [MHz]
1e6
Low
High
LowAgain
6
4
2
0
−2
−4
−6
−8
0
20
40
60
80


Filter 5 Current drift. Total drift 0.000150 [A]
Low
High
LowAgain
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
−0.00002
−0.00004
0
20
40
60
80


Filter 5 Phase drift. Total drift 0.157830 [rad]
Low
High
LowAgain
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
0
20
40
60
80

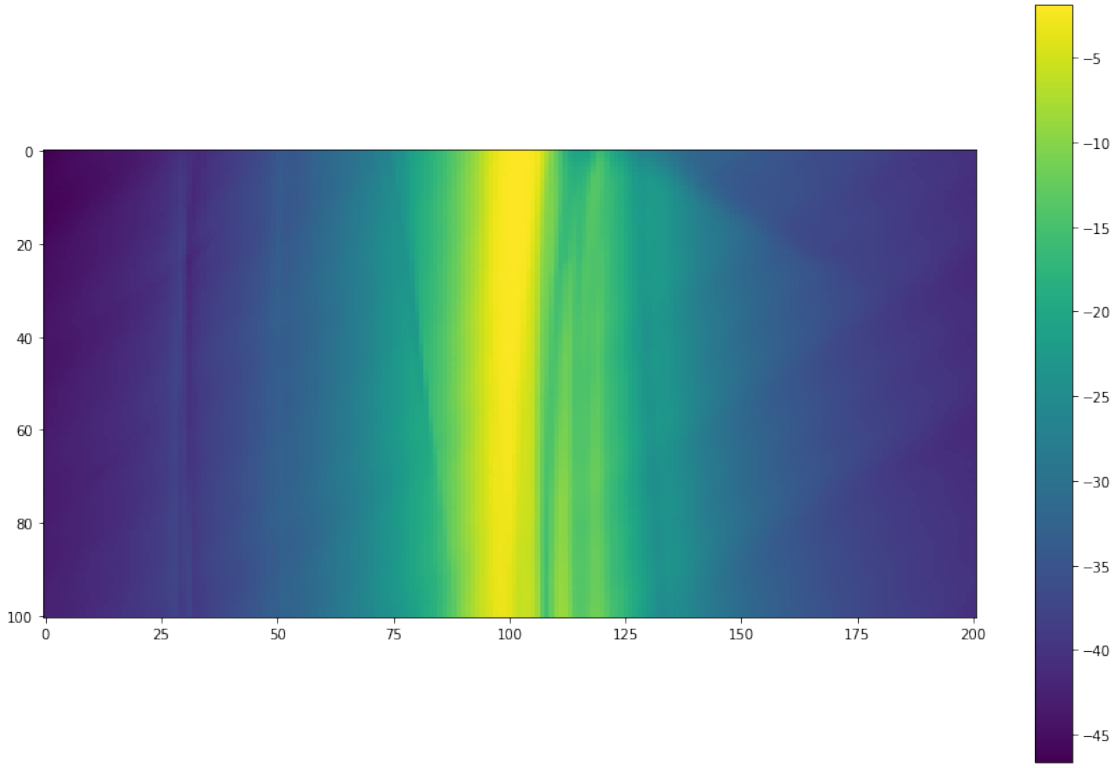


0.5 Fine tuning of filter tuning parameters

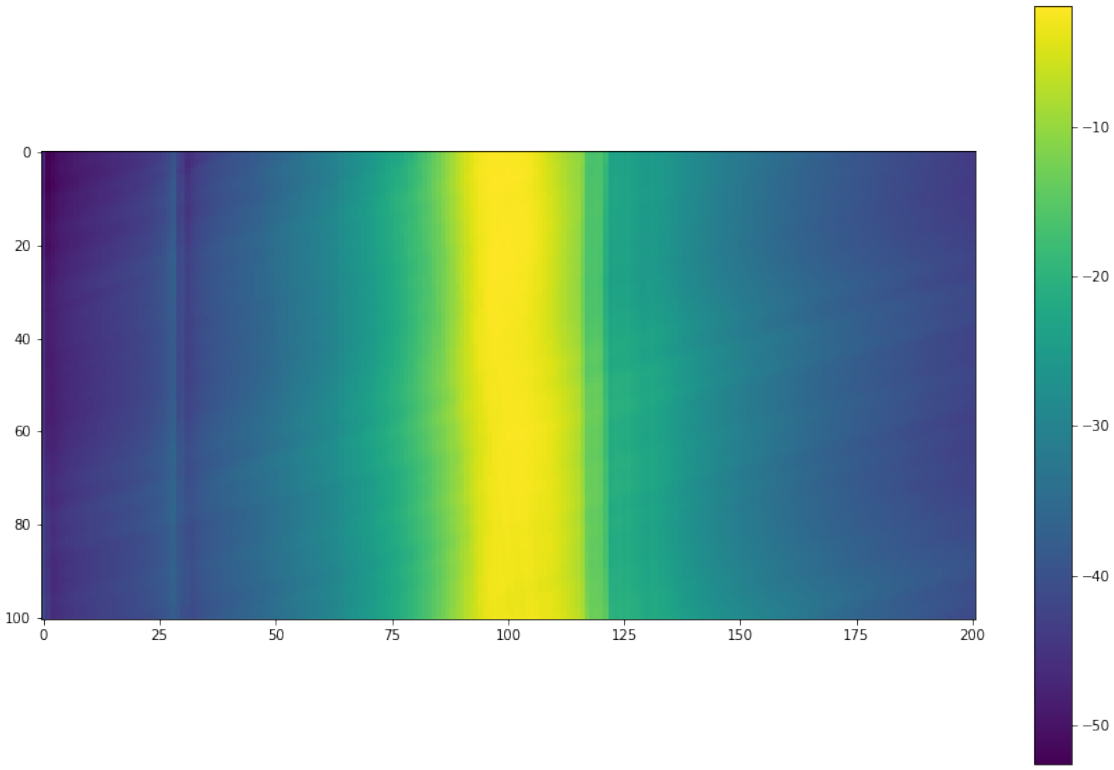
[3]:
```
a = loadData('coarse_filter_parameters')
import yig_controller_test
import yig_filter_model

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.computetTuningWord(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
```

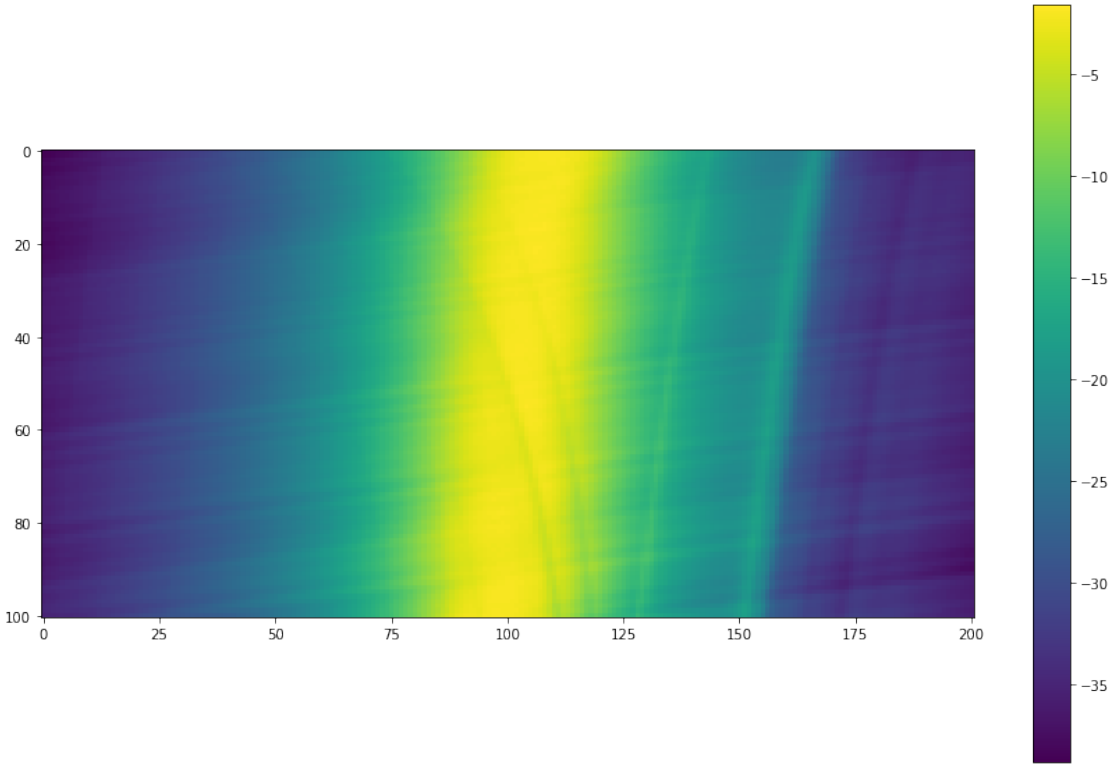
[14]:
```
filter0Map, filter0TuningWords, filter0Frequencies, filter0SpanMap =␣
↪measureYigFilter(filterBank[0])
```



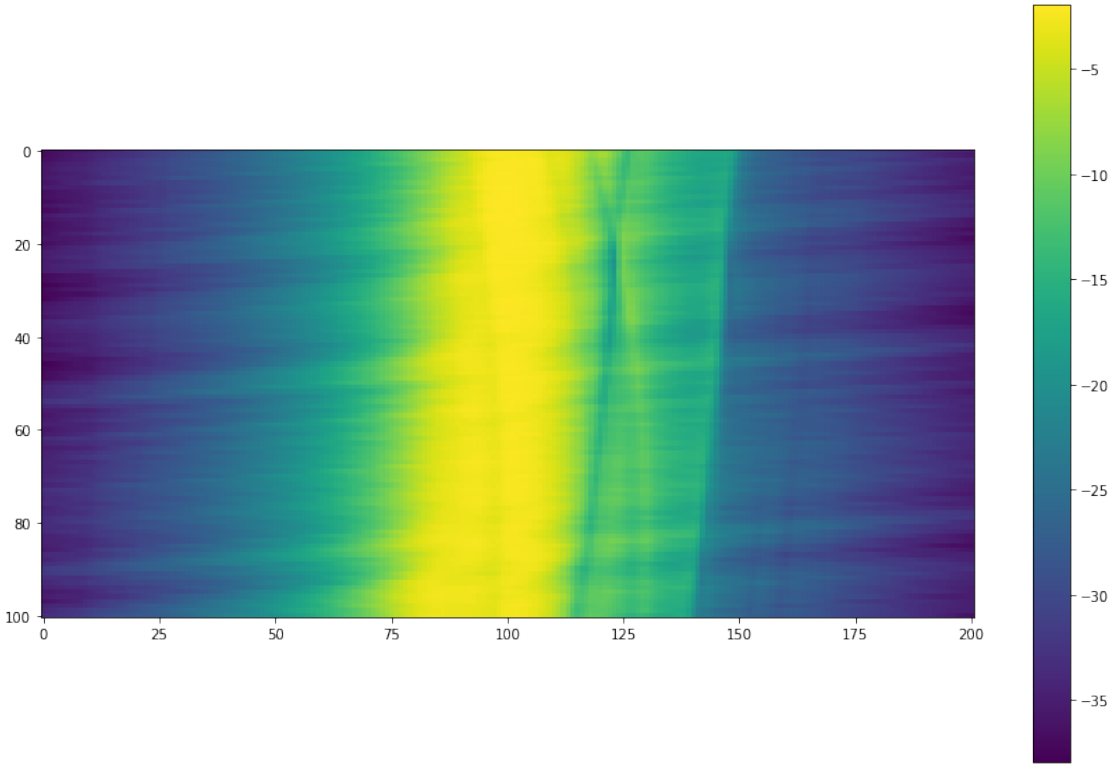
[15]:
```
filter1Map, filter1TuningWords, filter1Frequencies, filter1SpanMap =␣
↪measureYigFilter(filterBank[1])
```



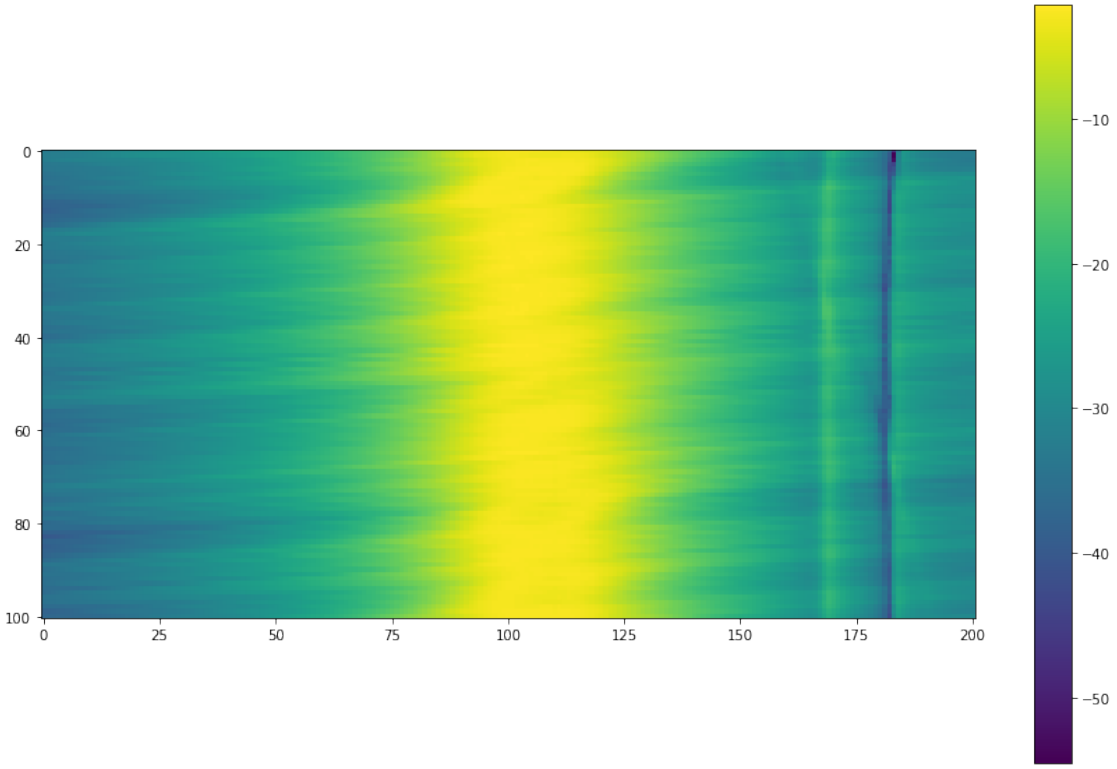
[16]:
```
filter2Map, filter2TuningWords, filter2Frequencies, filter2SpanMap =␣
↪measureYigFilter(filterBank[2])
```



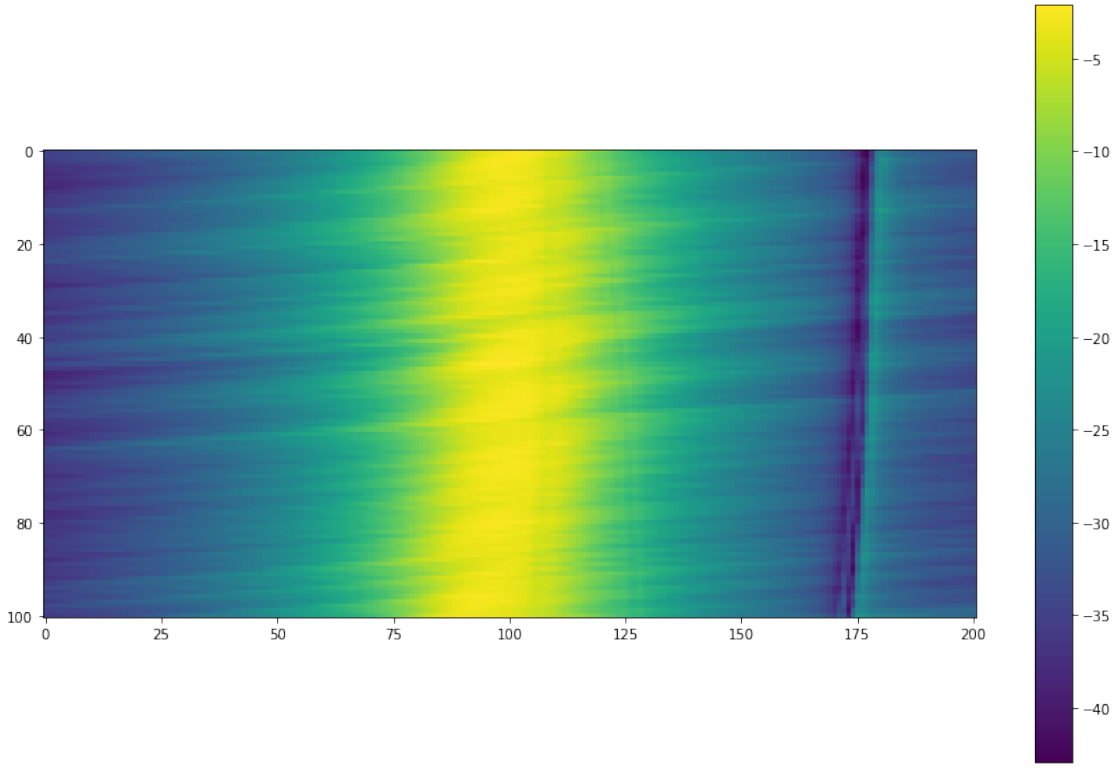
[17]:
```
filter3Map, filter3TuningWords, filter3Frequencies, filter3SpanMap =␣
↪measureYigFilter(filterBank[3])
```



[18]:
```
filter4Map, filter4TuningWords, filter4Frequencies, filter4SpanMap =␣
↪measureYigFilter(filterBank[4])
```



[19]:
```
filter5Map, filter5TuningWords, filter5Frequencies, filter5SpanMap =␣
↪measureYigFilter(filterBank[5])
```



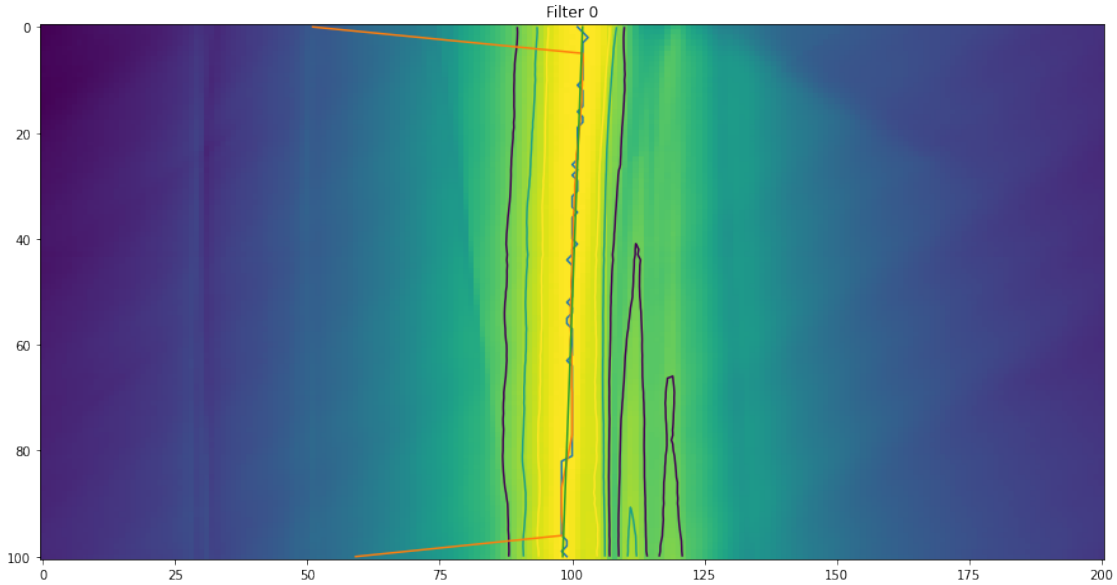
[20]:
```
saveDict={'filter0Map':filter0Map, 'filter0TuningWords':filter0TuningWords,␣
 ↪'filter0Frequencies':filter0Frequencies, 'filter0SpanMap':filter0SpanMap,
          'filter1Map':filter1Map, 'filter1TuningWords':filter1TuningWords,␣
 ↪'filter1Frequencies':filter1Frequencies, 'filter1SpanMap':filter1SpanMap,
          '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)
```

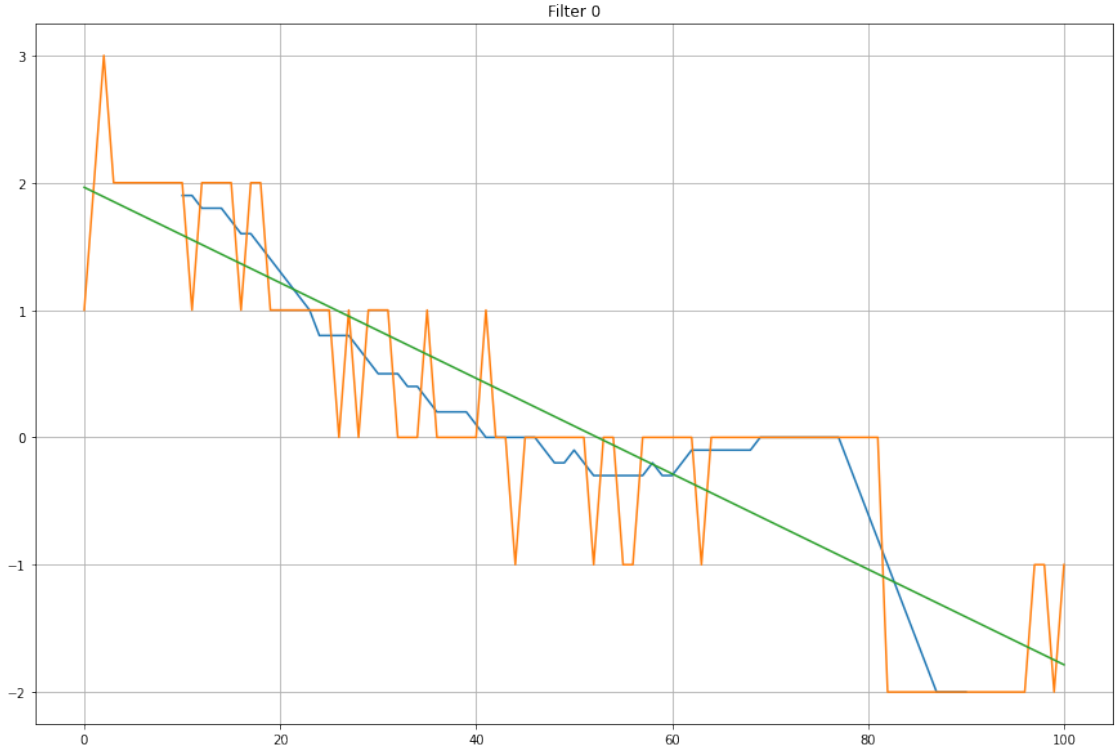
[21]:
```
def dB(data):
    return 20*np.log10(np.abs(data))

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

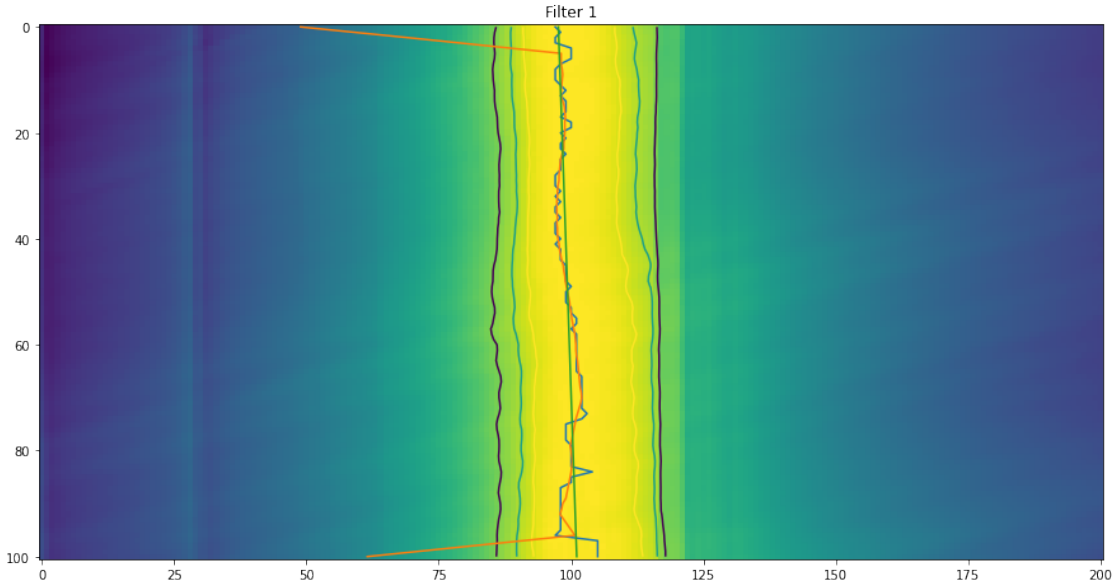
```
    spanMap = meas[baseKey+'SpanMap']
    dBFilt=dB(filterMap)
    peaksIdx=np.argmax(dB(filterMap),axis=1)
    peaksVal=np.max(dB(filterMap),axis=1)

    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), []
```

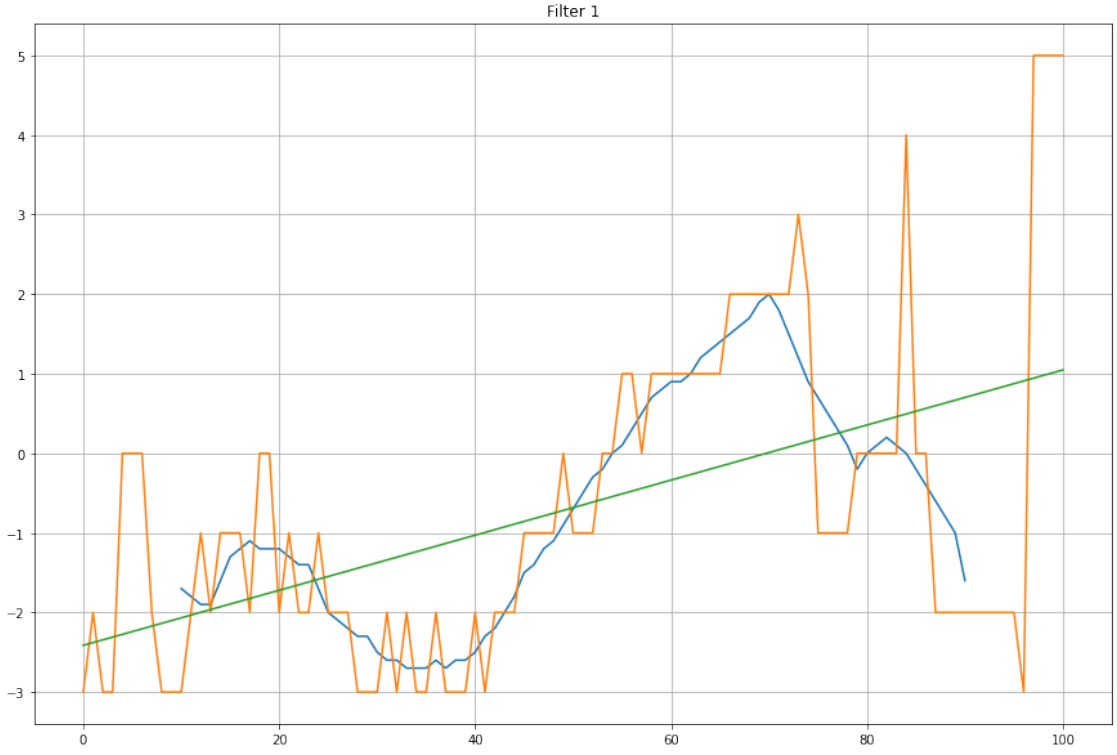
[24]:
```
ftm = loadData('coarse_tuning_fine_measurements')
coarseData = loadData('coarse_filter_parameters')

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)
```

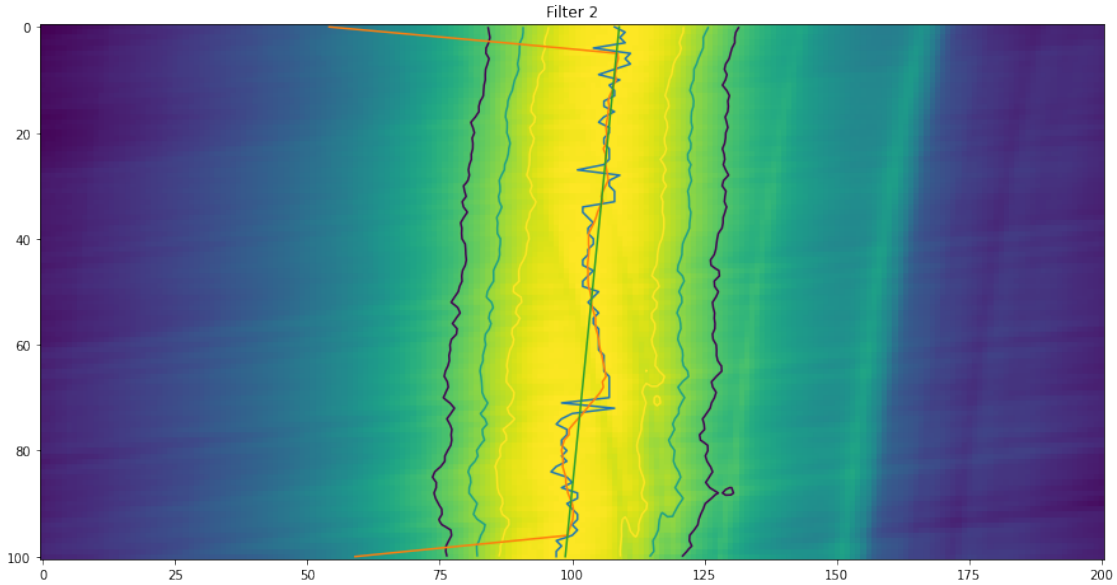
```
filter 0 Coarse k 0.02372116170186023 [MHZ/lsb] coarse m 699.0730360841161 [MHz]
filter 0 fine   k 0.023165242595050053 [MHZ/lsb] cfine  m 703.9827623508604
[MHz]
filter 1 Coarse k 0.07235017483316573 [MHZ/lsb] coarse m 2000.8478118760038
[MHz]
filter 1 fine   k 0.07297632373845435 [MHZ/lsb] cfine  m 1994.812575999475 [MHz]
filter 2 Coarse k 0.08496424368115545 [MHZ/lsb] coarse m 2277.880672173517 [MHz]
filter 2 fine   k 0.0838832026175793 [MHZ/lsb] cfine  m 2300.0836424705467 [MHz]
filter 3 Coarse k 0.16662916373239434 [MHZ/lsb] coarse m 4523.037301299044 [MHz]
filter 3 fine   k 0.1665224061419054 [MHZ/lsb] cfine  m 4524.7393979793005 [MHz]
filter 4 Coarse k 0.3064182278632889 [MHZ/lsb] coarse m 8348.592059966035 [MHz]
filter 4 fine   k 0.30672498315925734 [MHZ/lsb] cfine  m 8356.26823934868 [MHz]
filter 5 Coarse k 0.3645442209958414 [MHZ/lsb] coarse m 9886.749017341645 [MHz]
filter 5 fine   k 0.3634086439207302 [MHZ/lsb] cfine  m 9891.939757003845 [MHz]
```

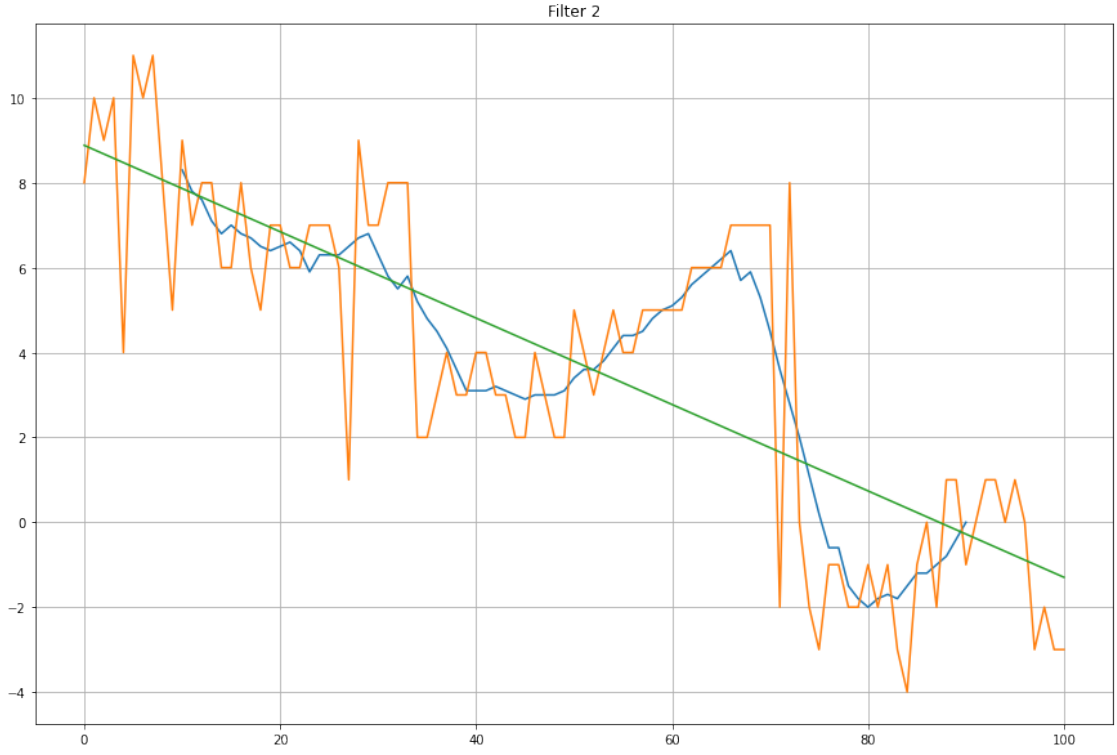

Filter 0
0
20
40
60
80
100
0
25
50
75
100
125
150
175
200

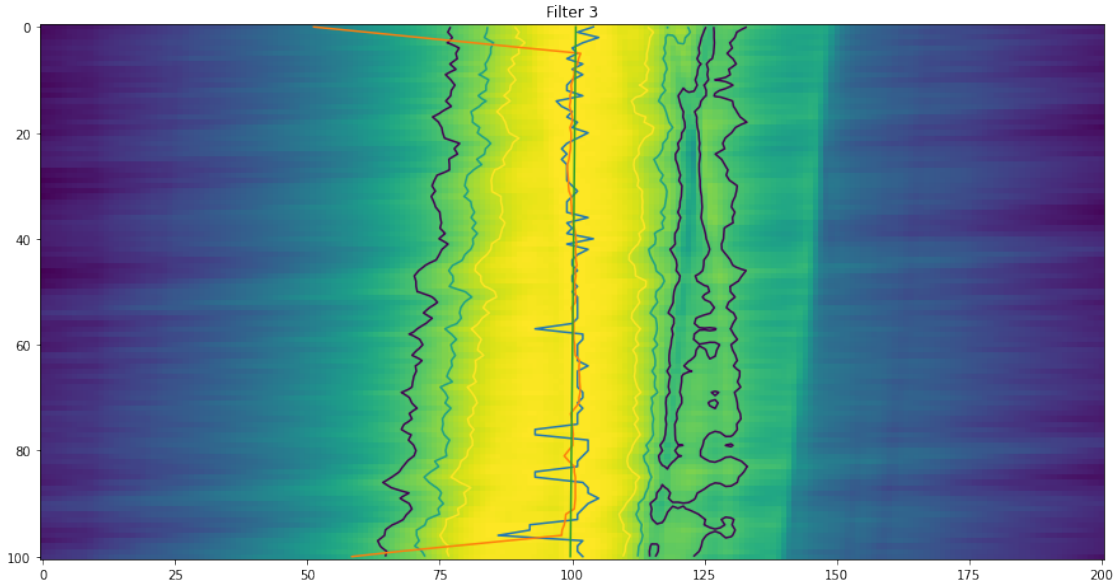

Filter 0
3
2
1
0
−1
−2
0
20
40
60
80
100

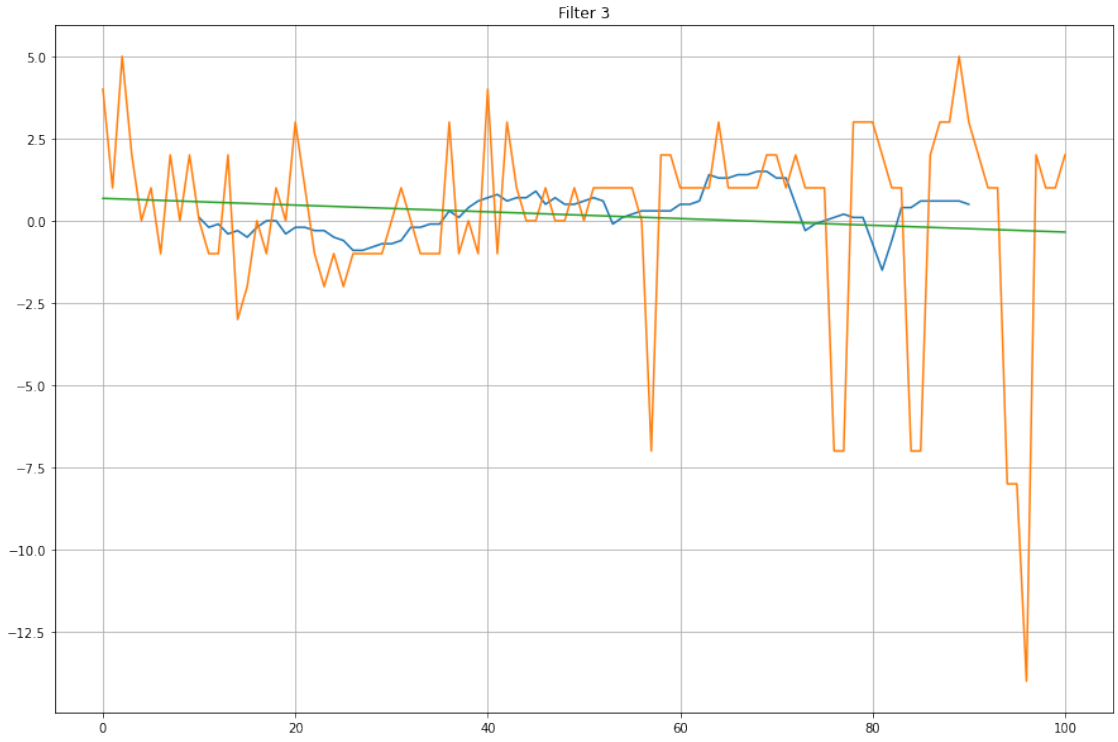

Filter 1
0
20
40
60
80
100
0
25
50
75
100
125
150
175
200

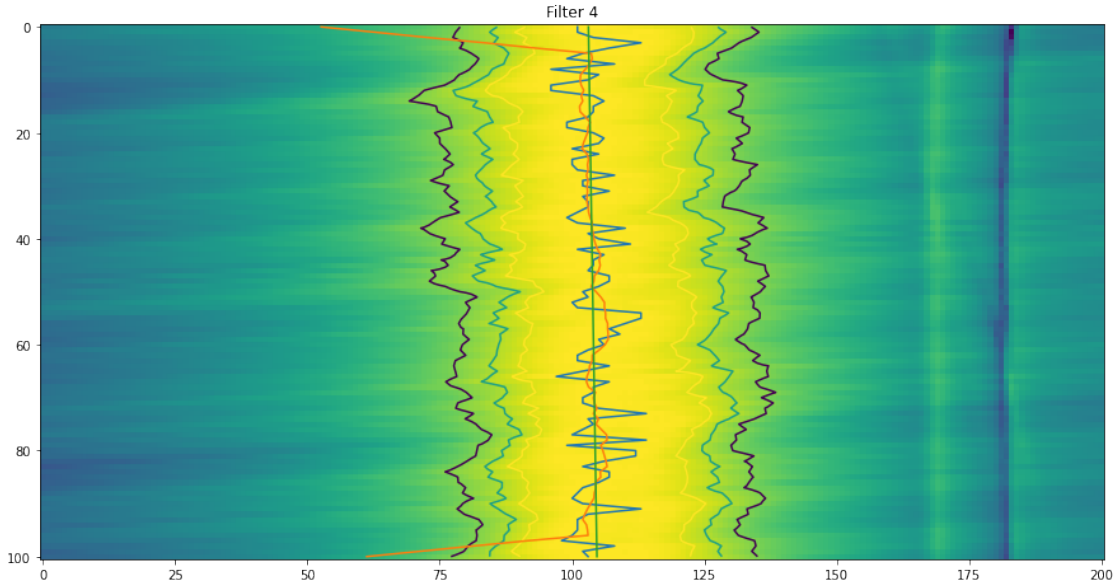

Filter 1
5
4
3
2
1
0
−1
−2
−3
0
20
40
60
80
100

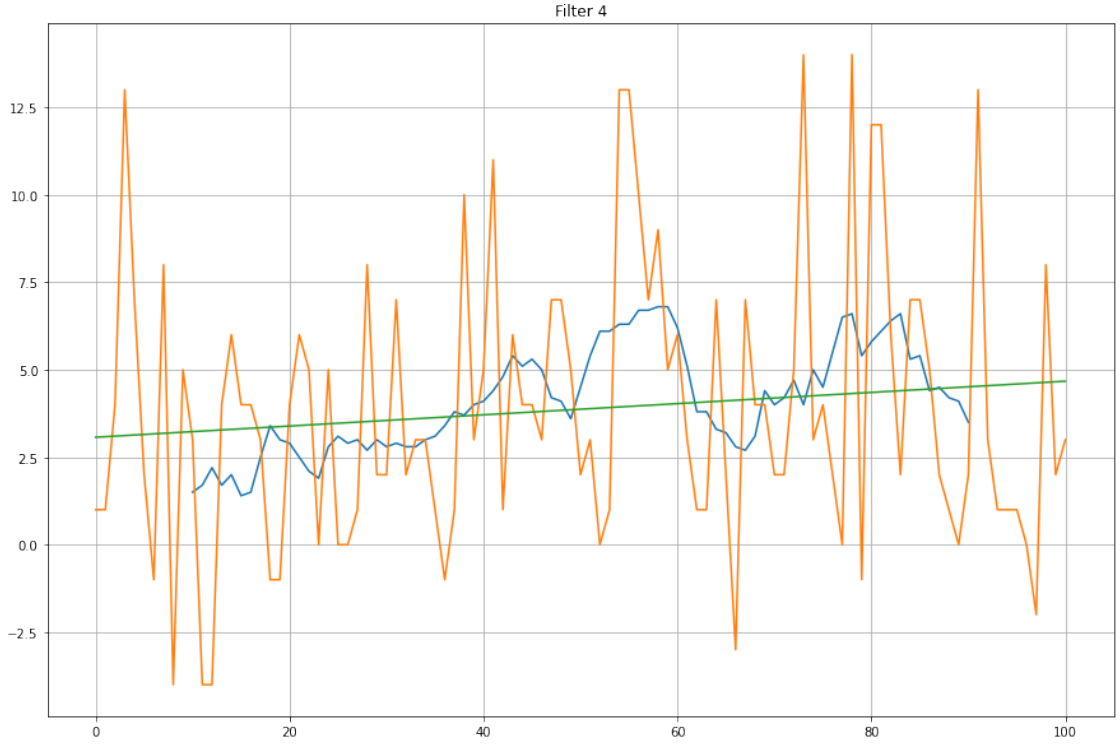

Filter 2
0
20
40
60
80
100
0
25
50
75
100
125
150
175
200

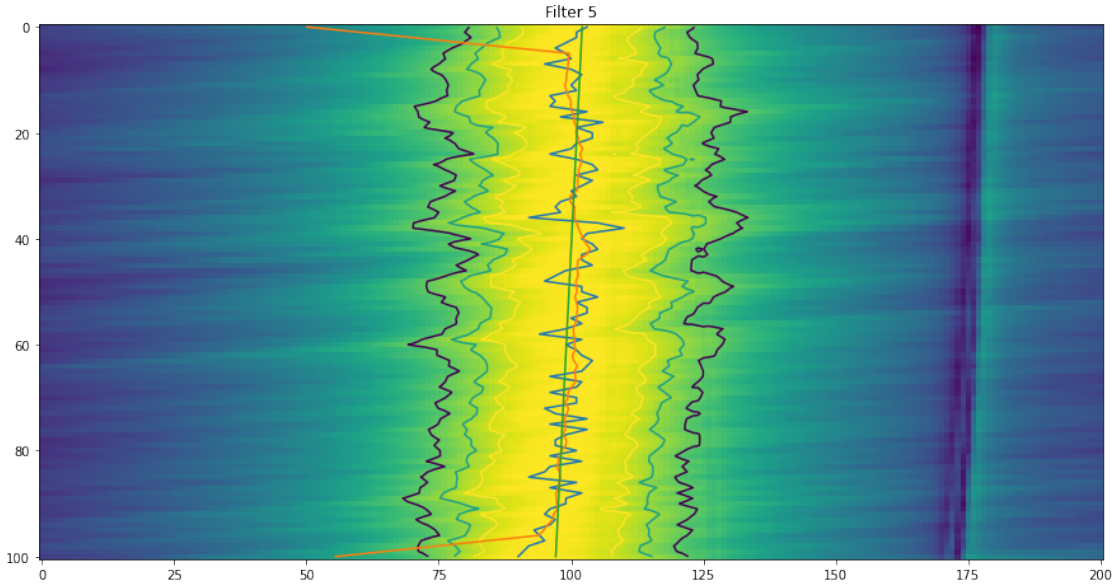

Filter 2
10
8
6
4
2
0
−2
−4
0
20
40
60
80
100

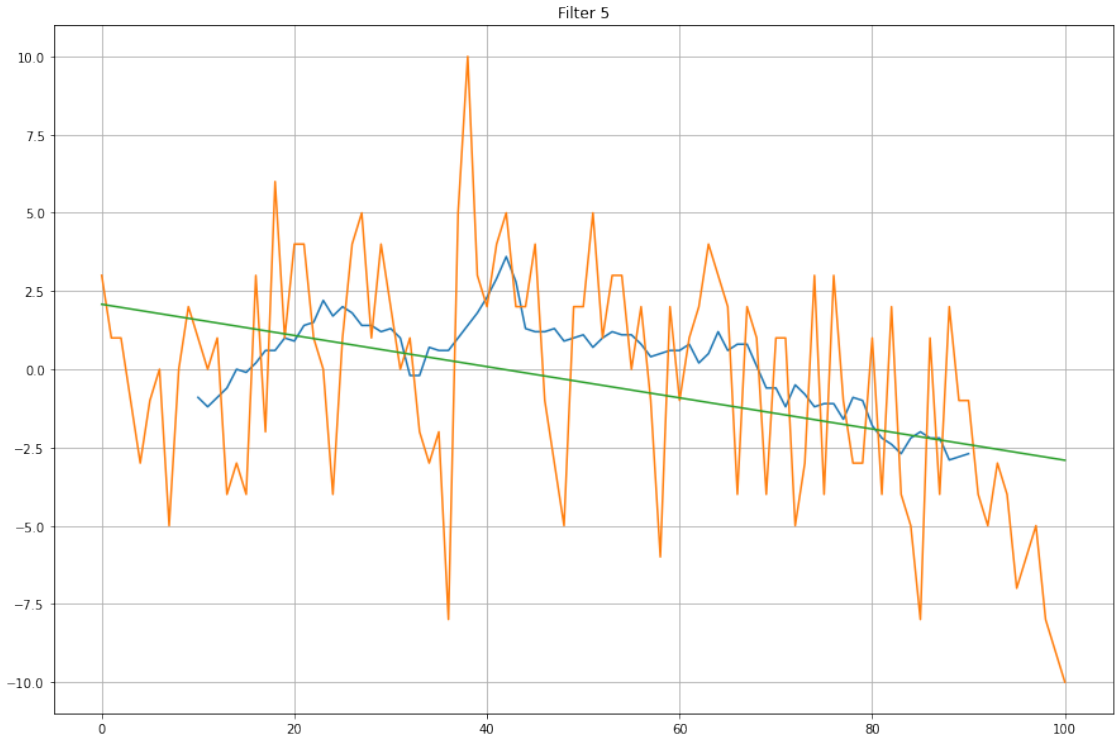

Filter 3
0
20
40
60
80
100
0
25
50
75
100
125
150
175
200


Filter 3
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
0
20
40
60
80
100


Filter 4
0
20
40
60
80
100
0
25
50
75
100
125
150
175
200


Filter 4
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
0
20
40
60
80
100





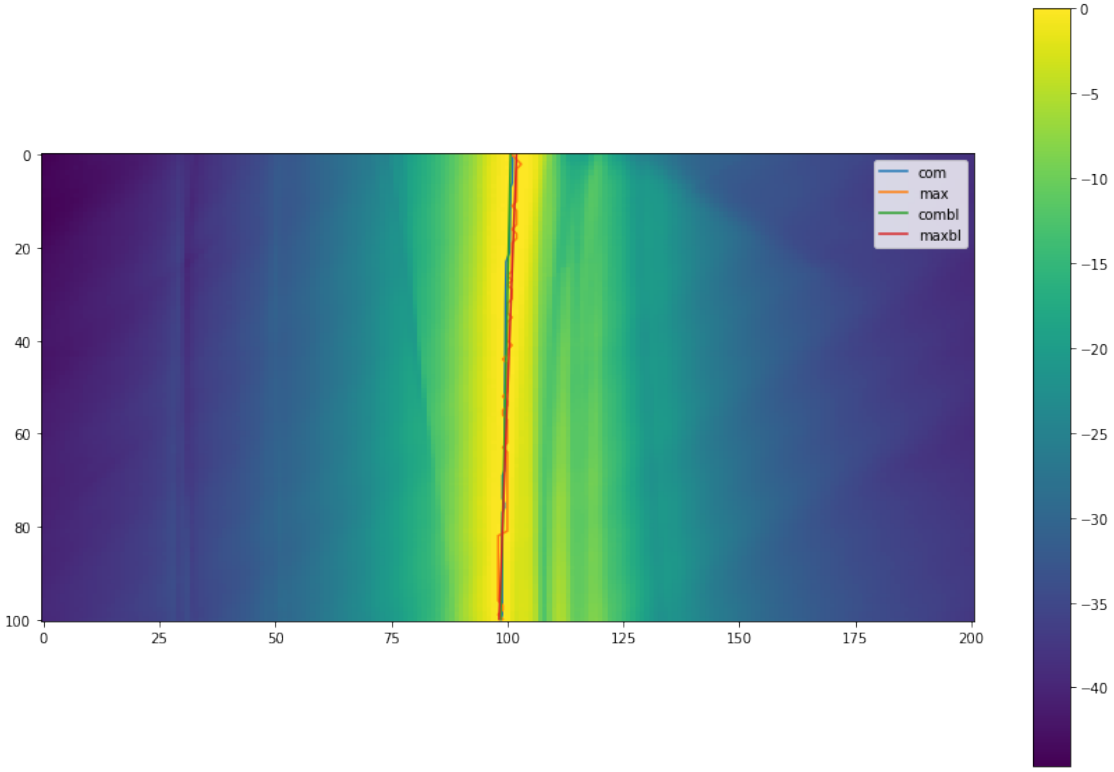
[25]:
```
ftm = loadData('coarse_tuning_fine_measurements')
coarseData = loadData('coarse_filter_parameters')

print(ftm.keys())
```

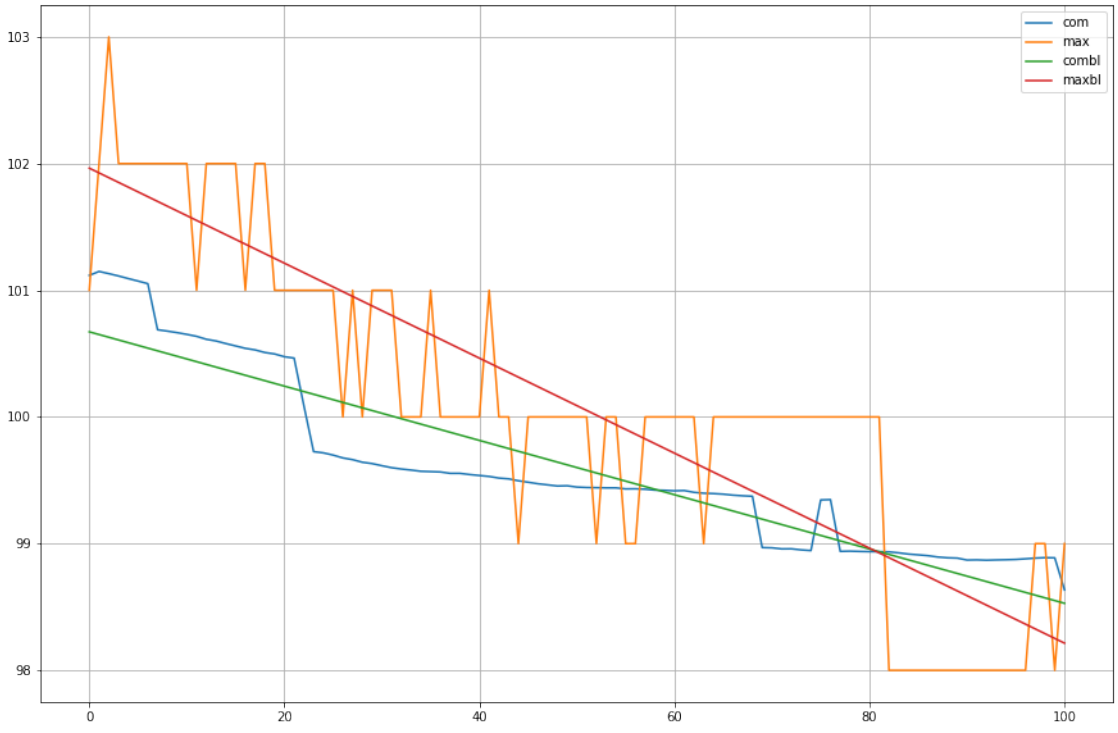
```
def computeCenterOfMass(s21Mag):
    ns=s21Mag.shape[1]
    massPositionProduct=np.multiply(s21Mag,np.arange(ns))
    massPositionProductSum=np.sum(massPositionProduct, axis=1)
    centerOfMass = massPositionProductSum/np.sum(s21Mag,axis=1)
    return centerOfMass

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))
```

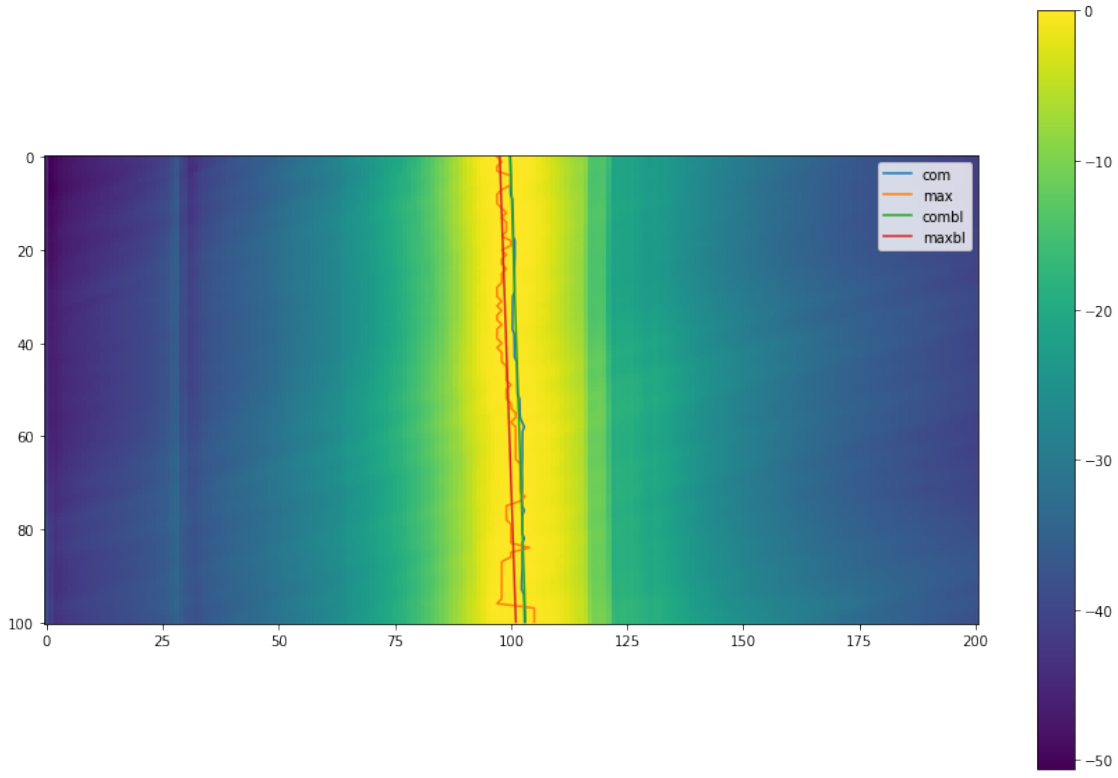
```
    plt.plot(centerOfMass.T, ya, label='com')
    plt.plot(peaks.T, ya, label='max')
    plt.plot(centerOfMassFunc(ya), ya, label='combl')
    plt.plot(peaksFunc(ya), ya, label='maxbl')
    plt.colorbar()
    plt.legend()
    plt.figure()
    plt.plot(ya, centerOfMass.T, label='com')
    plt.plot(ya,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

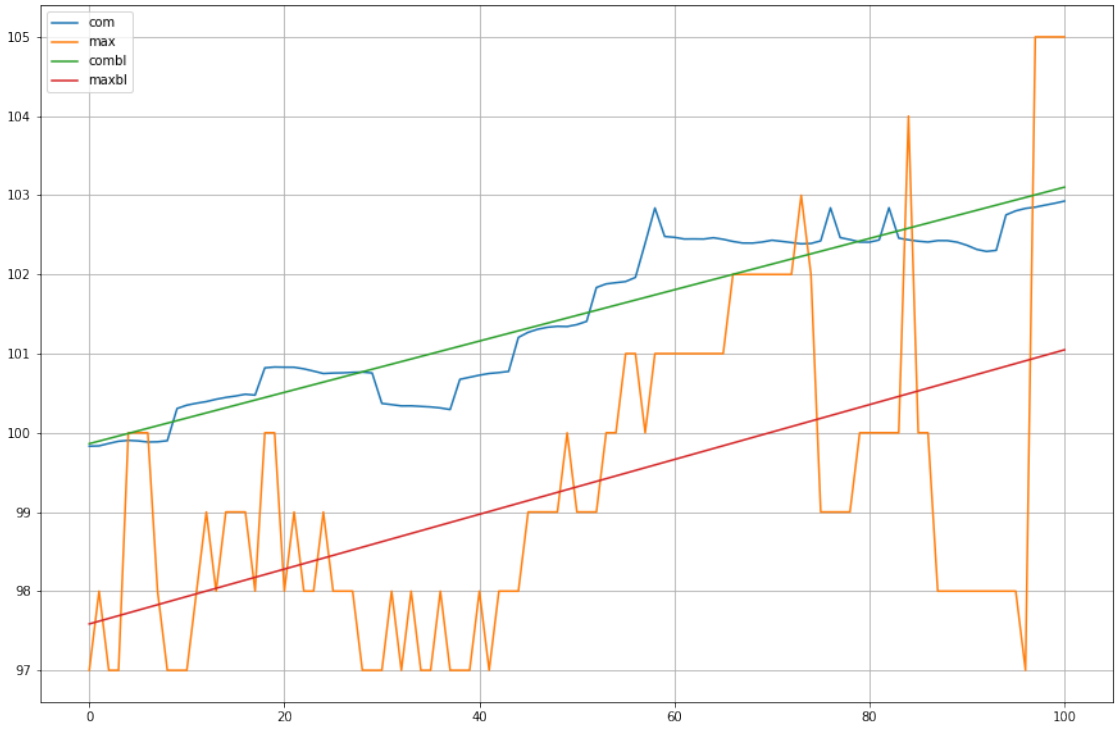
```
dict_keys(['__header__', '__version__', '__globals__', 'filter0Map',
'filter0TuningWords', 'filter0Frequencies', 'filter0SpanMap', 'filter1Map',
'filter1TuningWords', 'filter1Frequencies', 'filter1SpanMap', 'filter2Map',
'filter2TuningWords', 'filter2Frequencies', 'filter2SpanMap', 'filter3Map',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

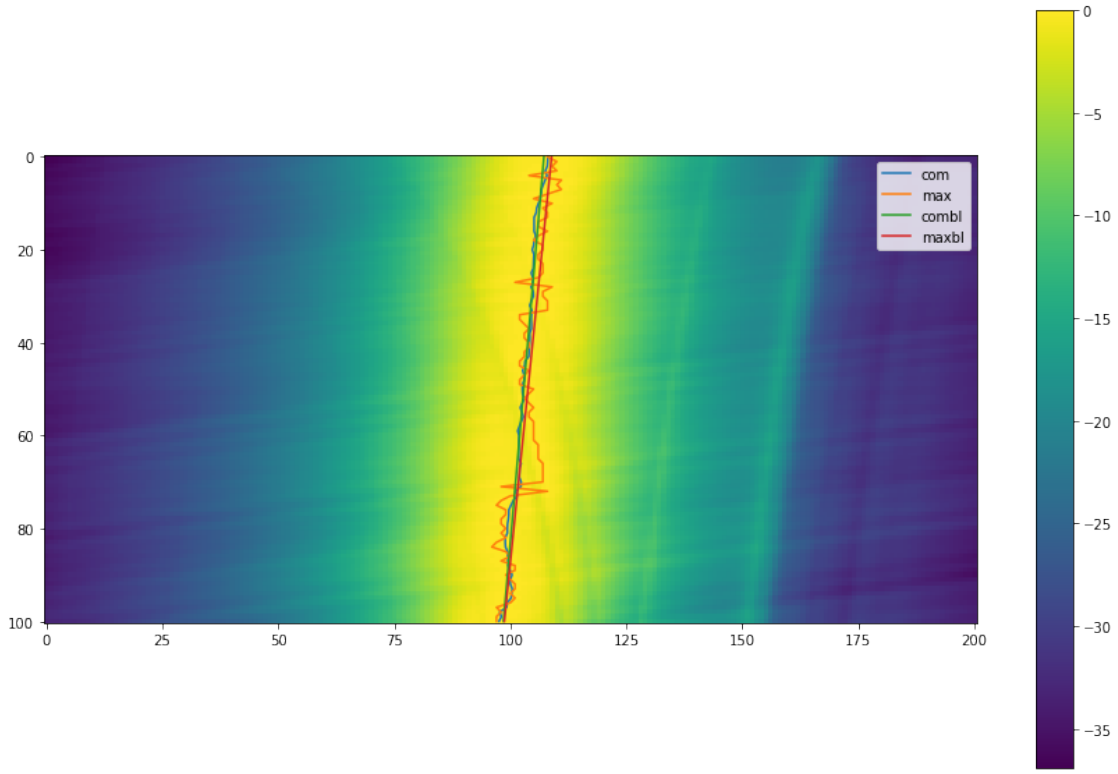
```
K 42.724714 [LSB/MHz] M -29882.627991 [LSB]
K 13.710514 [LSB/MHz] M -27538.639055 [LSB]
K 11.898843 [LSB/MHz] M -27283.148752 [LSB]
K 6.032053 [LSB/MHz] M -27296.960788 [LSB]
K 3.254860 [LSB/MHz] M -27203.797099 [LSB]
K 2.750581 [LSB/MHz] M -27223.782777 [LSB]
```

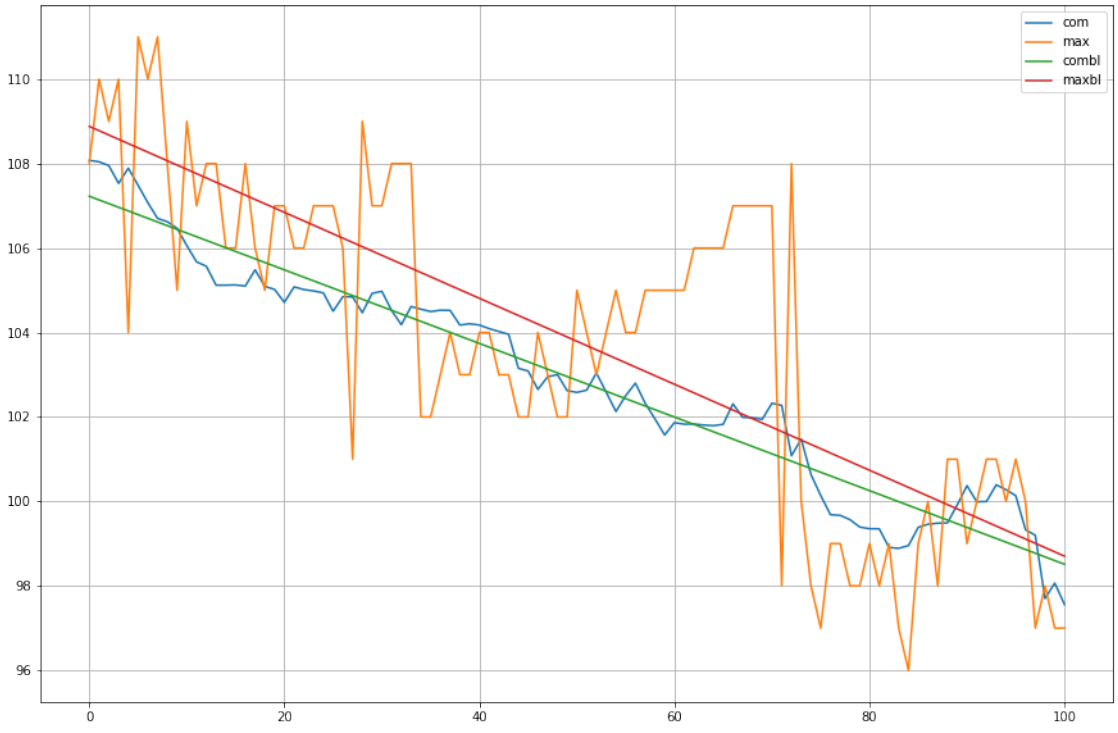


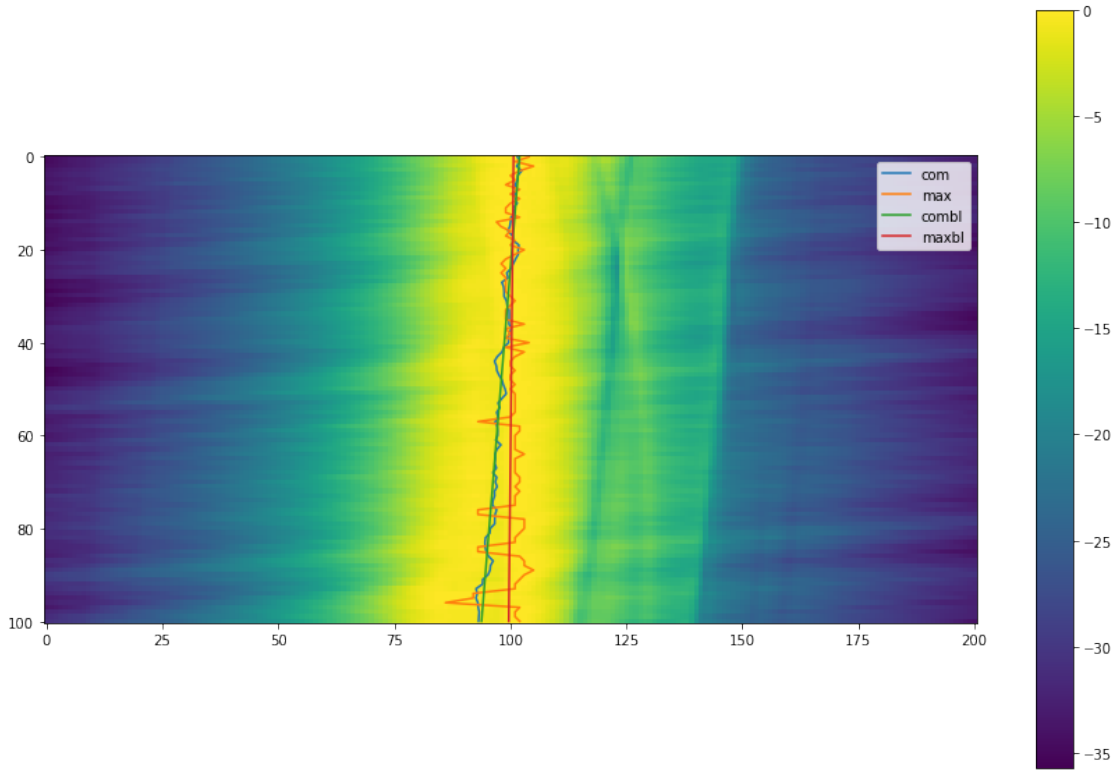

com
max
combl
maxbl
103
102
101
100
99
98
0
20
40
60
80
100

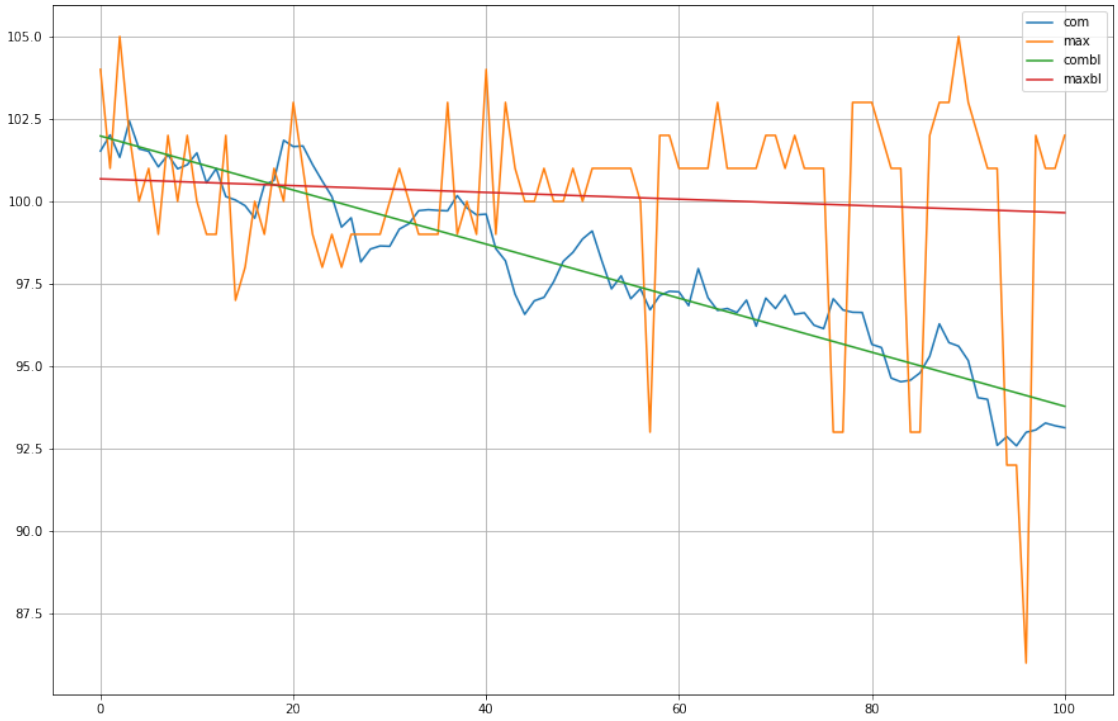

com
max
combl
maxbl
0
20
40
60
80
100
0
25
50
75
100
125
150
175
200
0
−10
−20
−30
−40
−50

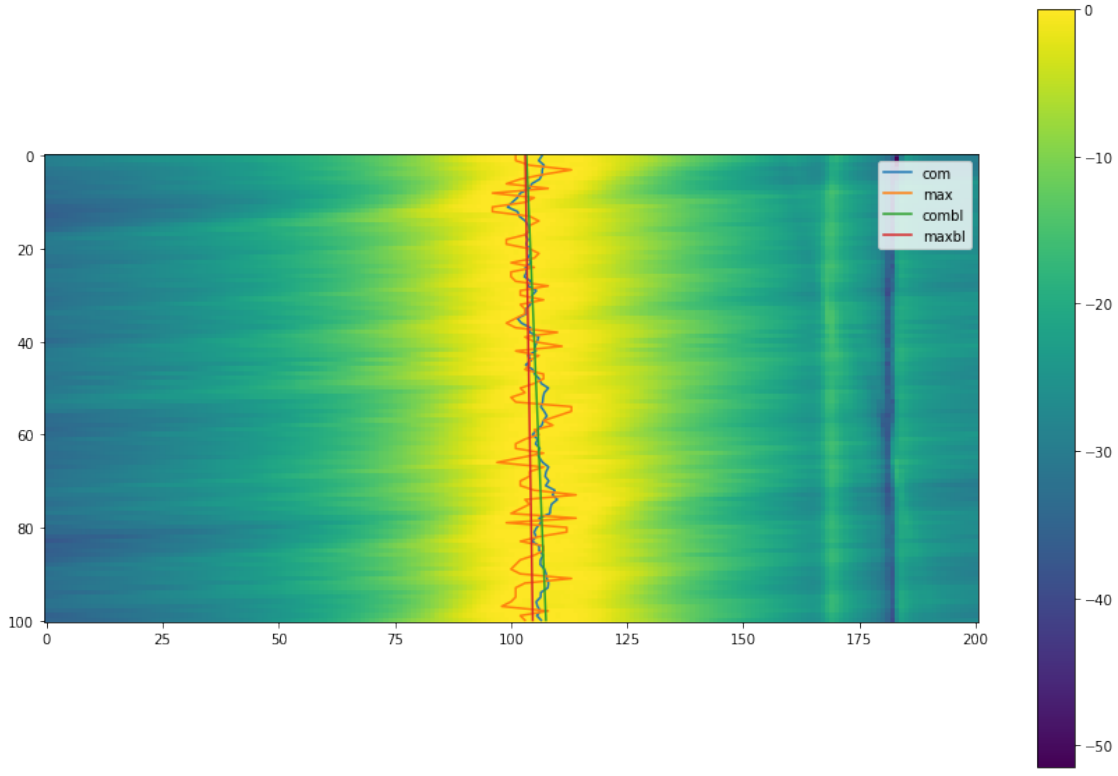

com
max
combl
maxbl
105
104
103
102
101
100
99
98
97
0
20
40
60
80
100

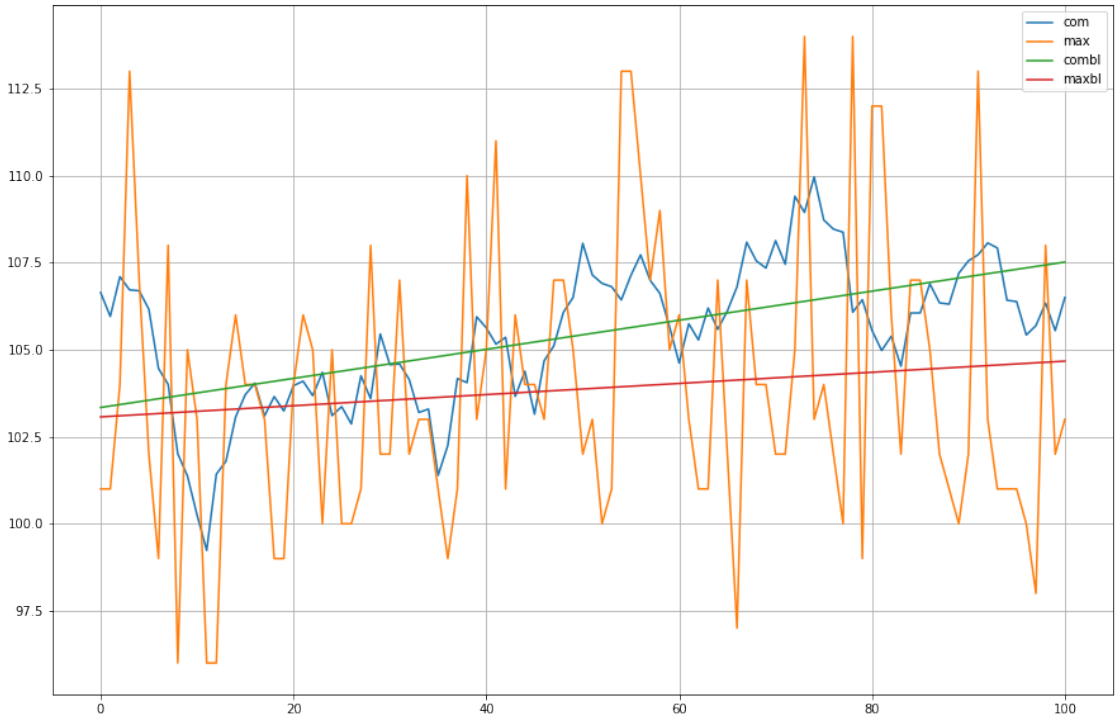

com
max
combl
maxbl
0
20
40
60
80
100
0
25
50
75
100
125
150
175
200
0
−5
−10
−15
−20
−25
−30
−35

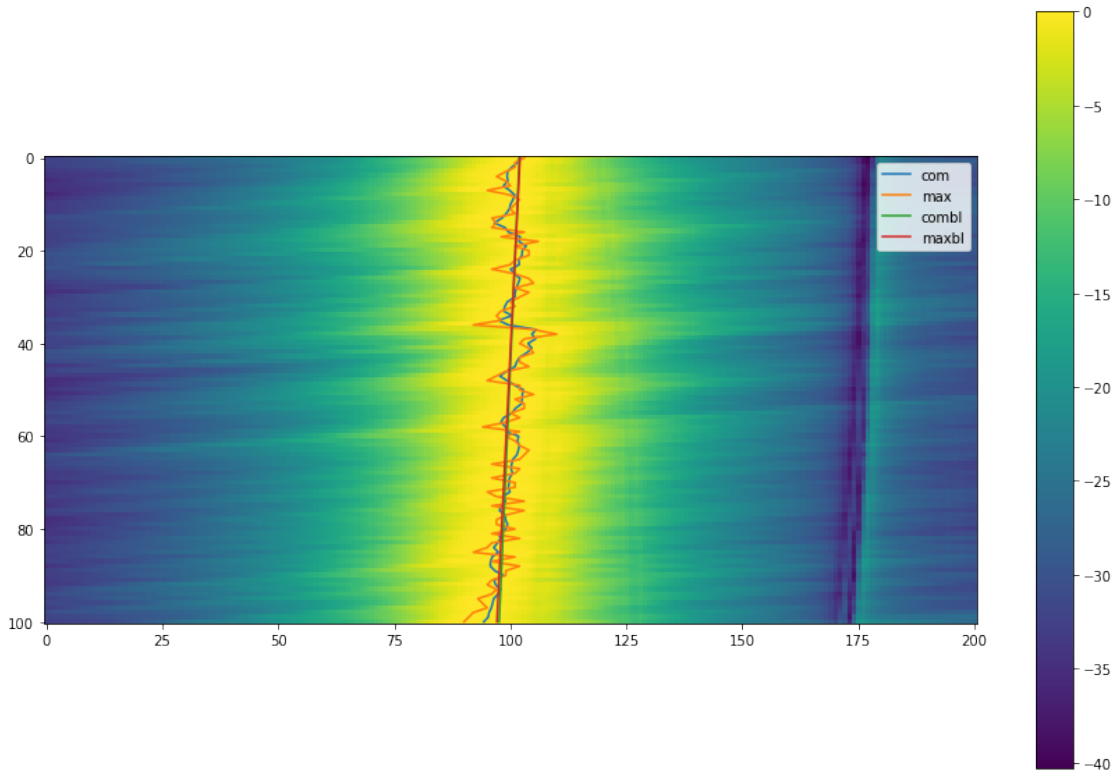

com
max
combl
maxbl
110
108
106
104
102
100
98
96
0
20
40
60
80
100

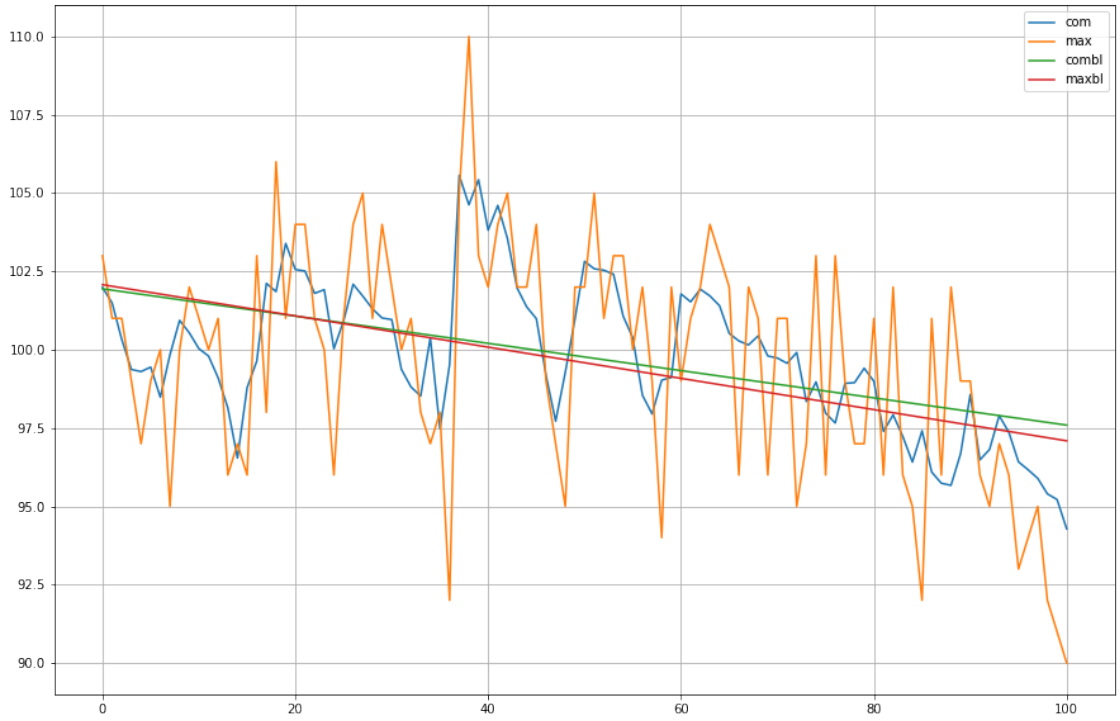

com
max
combl
maxbl
0
20
40
60
80
100
0
25
50
75
100
125
150
175
200
0
−5
−10
−15
−20
−25
−30
−35


com
max
combl
maxbl
105.0
102.5
100.0
97.5
95.0
92.5
90.0
87.5
0
20
40
60
80
100


com
max
combl
maxbl
0
20
40
60
80
100
0
25
50
75
100
125
150
175
200
0
−10
−20
−30
−40
−50


com
max
combl
maxbl
112.5
110.0
107.5
105.0
102.5
100.0
97.5
0
20
40
60
80
100

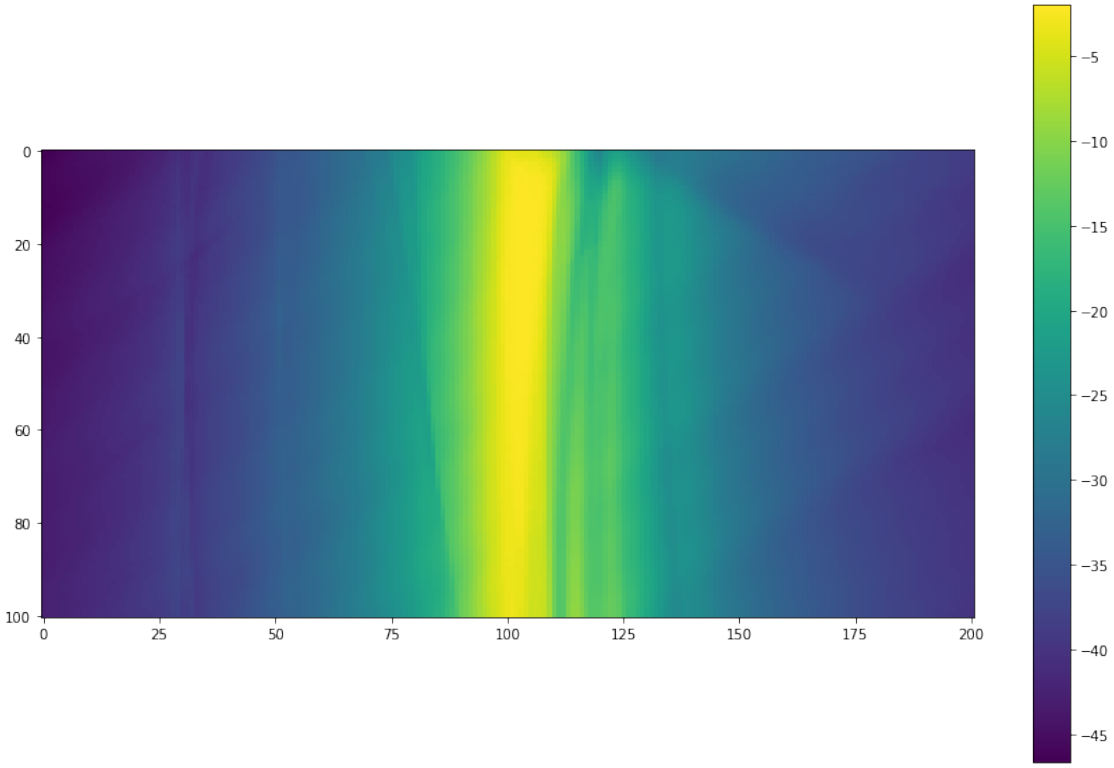

com
max
combl
maxbl
0
20
40
60
80
100
0
25
50
75
100
125
150
175
200
0
−5
−10
−15
−20
−25
−30
−35
−40



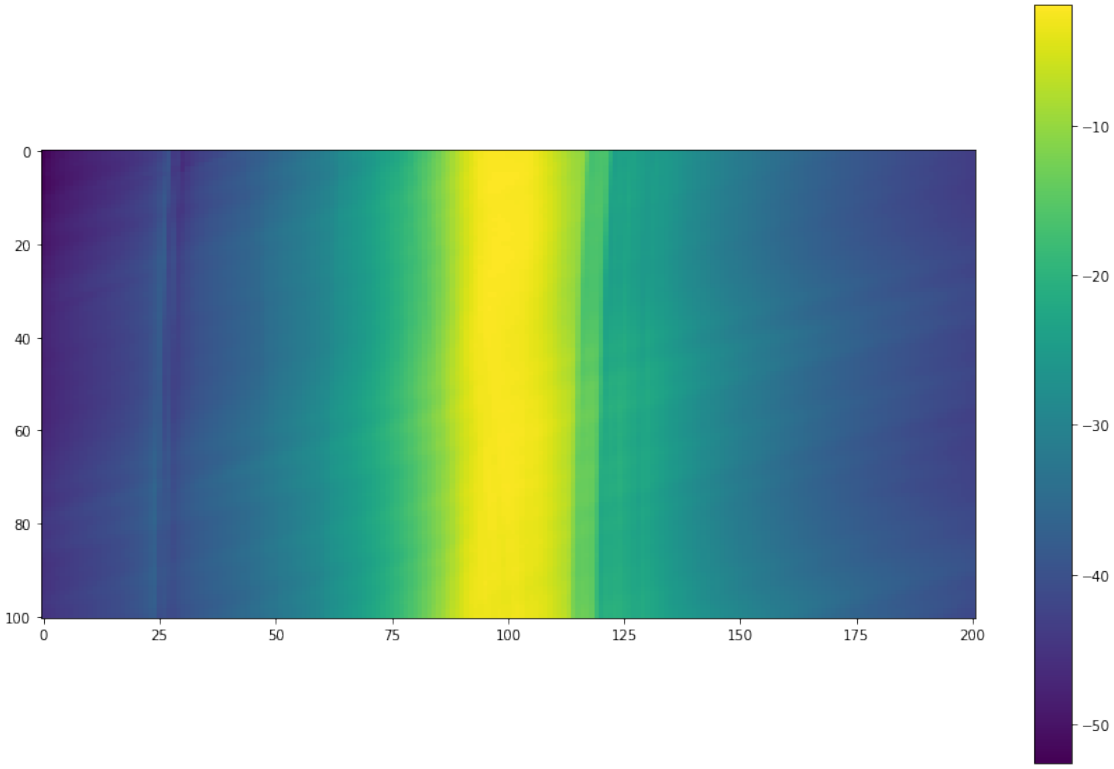
0.6 Check fine tuned parameters

[4]:
```
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])
```

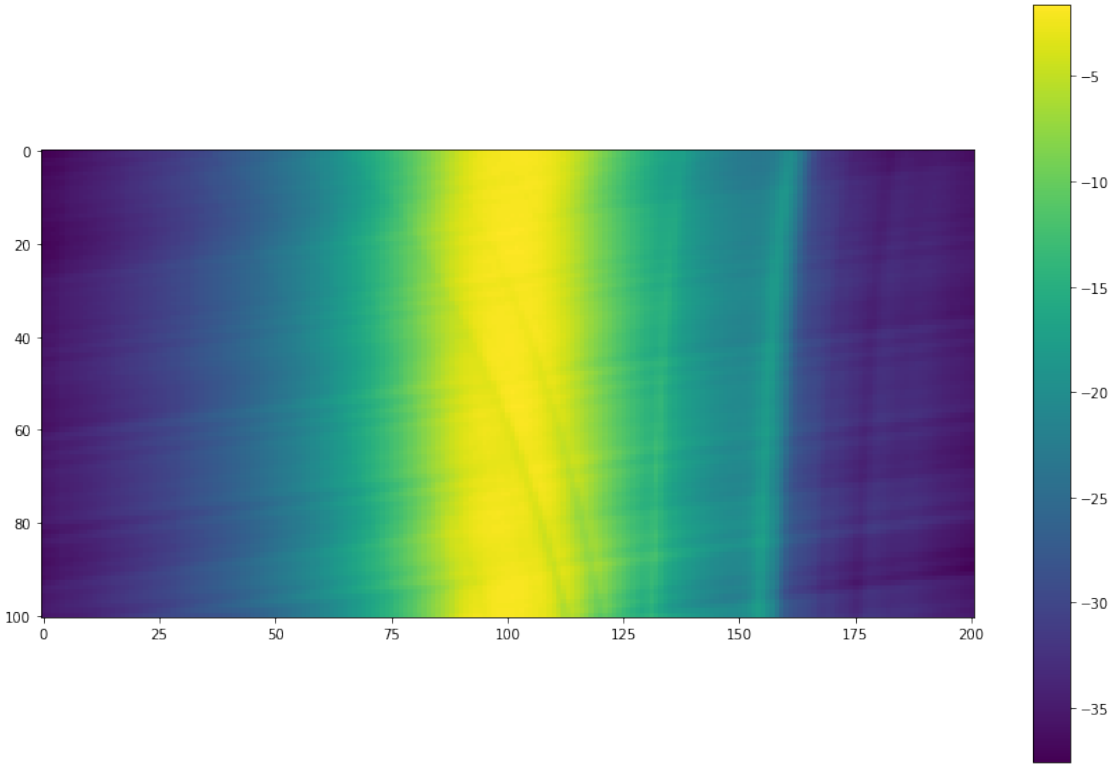
[6]:
```
filter0Map, filter0TuningWords, filter0Frequencies, filter0SpanMap =␣
 ↪measureYigFilter(filterBank[0])
```



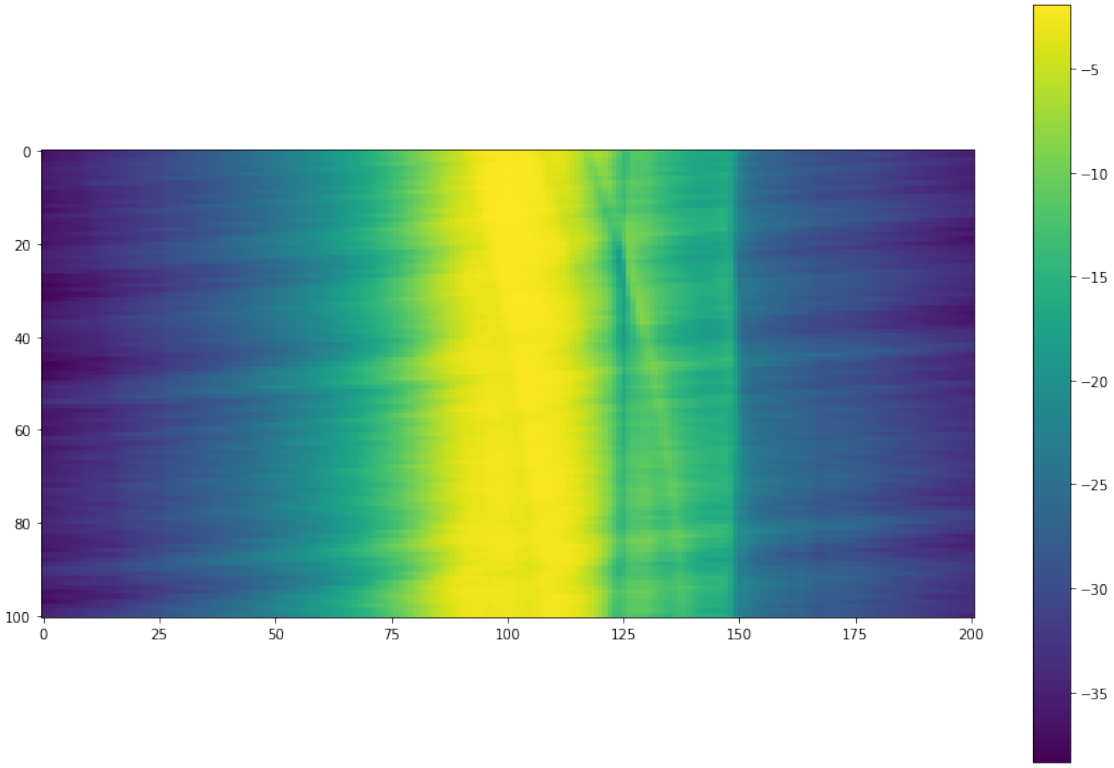
[7]:
```
filter1Map, filter1TuningWords, filter1Frequencies, filter1SpanMap =␣
↪measureYigFilter(filterBank[1])
```



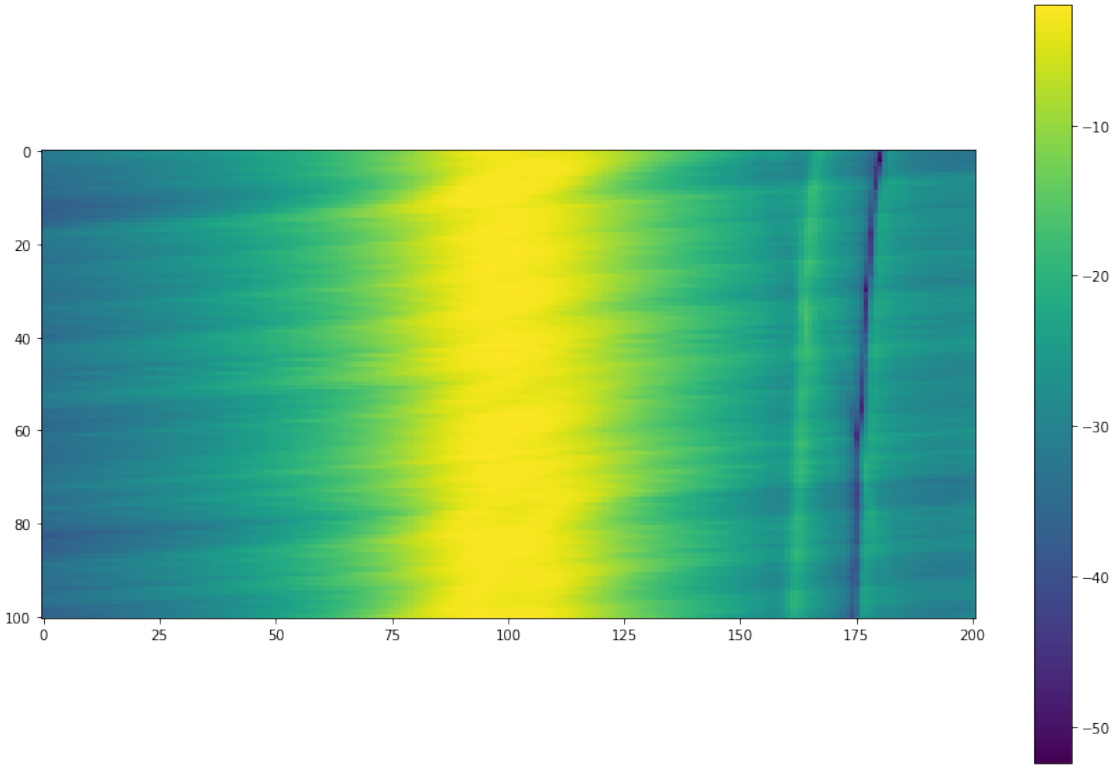
[8]:
```
filter2Map, filter2TuningWords, filter2Frequencies, filter2SpanMap =␣
↪measureYigFilter(filterBank[2])
```



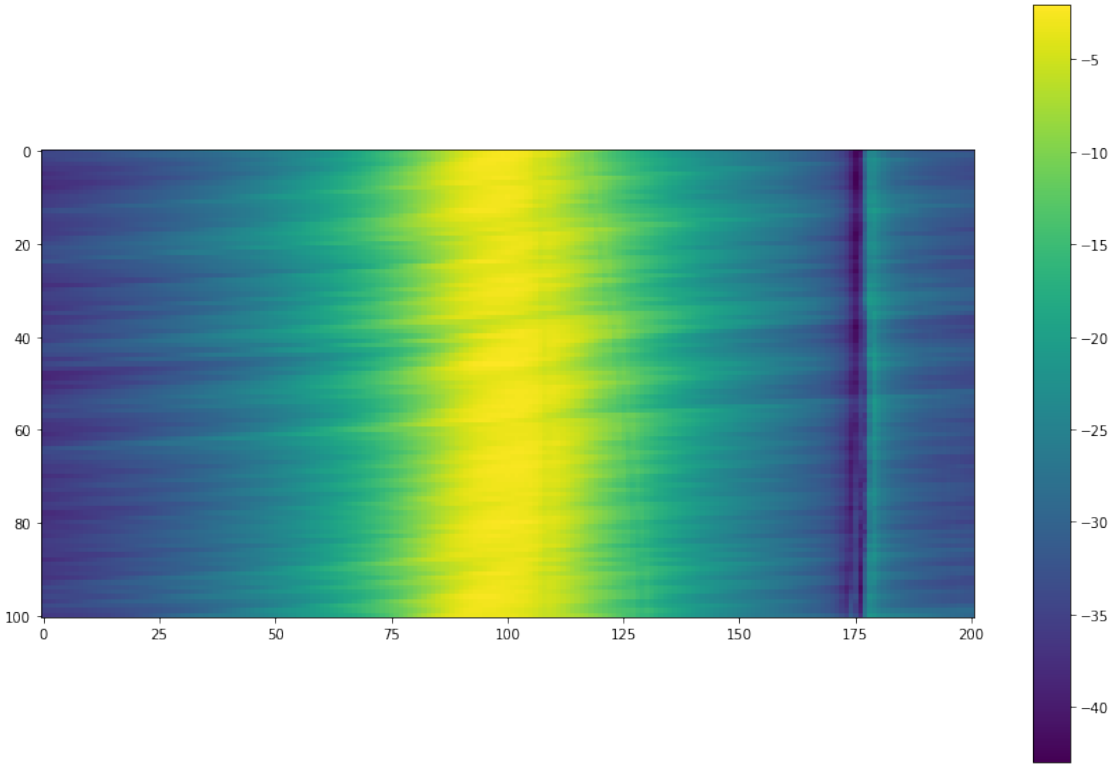
[9]:
```
filter3Map, filter3TuningWords, filter3Frequencies, filter3SpanMap =␣
↪measureYigFilter(filterBank[3])
```



[10]:
```
filter4Map, filter4TuningWords, filter4Frequencies, filter4SpanMap =␣
↪measureYigFilter(filterBank[4])
```



[11]:
```
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)
```

[13]:
```
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)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-13-2095a8ee9e46> in <module>
      9
     10 for i in range(6):
---> 11     k, m = fineTuneAnalysis(ftm, i)
     12     print("filter",i,"Coarse k", kcoarse[i], "[MHZ/lsb]", "coarse m",␣
 ↪mcoarse[i], "[MHz]")
     13     kfine[i] = kcoarse[i] + k

NameError: name 'fineTuneAnalysis' is not defined
```

0.7 Upload coefficients

[12]:
```
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)
```