gnssMPy

September 23, 2021

[1]:
```
import numpy as np
import scipy as sp
import pandas as pd
import matplotlib.pyplot as plt
import scipy.special as sc
import scipy.signal as sp
from IPython.display import clear_output
from tqdm import tqdm
```

[2]:
```
fileName = "data/2013_04_04_GNSS_SIGNAL_at_CTTC_SPAIN.dat"
```

0.1 PreProcessing

0.1.1 Constants

[3]:
```
## Satellite Frequency
FREQ1 = 1.57542e9        #L1_CA
FREQ2 = 1.22760e9       #  L2     frequency (Hz)
FREQ5 = 1.17645e9       #  L5/E5a frequency (Hz)
FREQ6 = 1.27875e9       #  E6/LEX frequency (Hz)
FREQ7 = 1.20714e9       #  E5b    frequency (Hz)
FREQ8 = 1.191795e9      #  E5a+b  frequency (Hz)
FREQ9 = 2.492028e9      #  S      frequency (Hz)
FREQ1_GLO  = 1.60200e9  #  GLONASS G1 base frequency (Hz)
DFRQ1_GLO  = 0.56250e6  #  GLONASS G1 bias frequency (Hz/n)
FREQ2_GLO  = 1.24600e9  #  GLONASS G2 base frequency (Hz)
DFRQ2_GLO  = 0.43750e6  #  GLONASS G2 bias frequency (Hz/n)
FREQ3_GLO  = 1.202025e9 #  GLONASS G3 frequency (Hz)
FREQ1_BDS  = 1.561098e9 #  BeiDou B1 frequency (Hz)
FREQ2_BDS  = 1.20714e9  #  BeiDou B2 frequency (Hz)
FREQ3_BDS  = 1.26852e9  #  BeiDou B3 frequency (Hz)
```

[4]:
```
#Sat frequency
acq_fs = FREQ1
### Raw signal Parameters
IF = 0
fs = 4e6
codeFreqBasis = 1.023e6
```

```
codeLength = 1023
samplesPerCode = round(fs/(codeFreqBasis/codeLength))

### Acquisition Settings
skipAcquisition = 0
acqSatelliteList = np.arange(0,32)
acqSearchBand = 14
acqThreshold = 2.5
acquisitionCohCodePeriods=2
acquisitionNonCodePeriods=2
pfa = 0.01
doppler_max = 10000
doppler_step = 250
CFAR = 1
#Filter settings
downSample = True
downSample_fs = 2e6
downSample_step = int(fs//downSample_fs) #Skip every 1 sample

fileType=2
dataOffset=80
```

[5]:
```
acqSatelliteList
```

[5]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31])
```

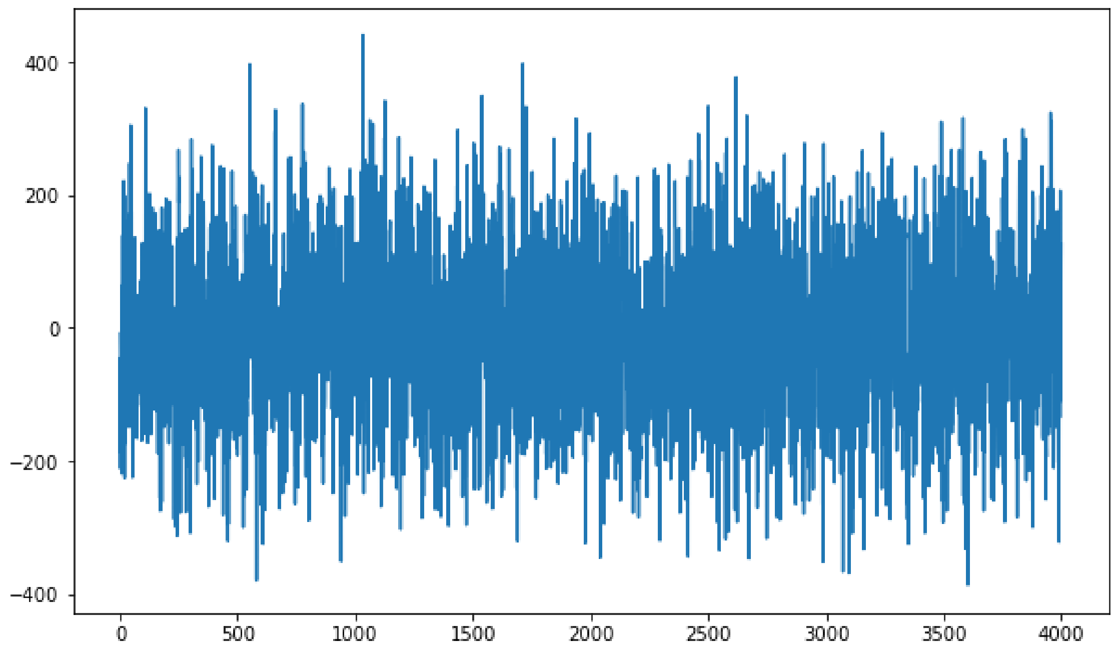
0.1.2 File I/O

[6]:
```
if fileType==1:
    dataType = np.complex64
elif fileType==2:
    dataType=np.int16
data= np.fromfile(fileName,dataType,offset=dataOffset,
count=30*(acquisitionCohCodePeriods*acquisitionNonCodePeriods)*samplesPerCode*6
)
data.shape
```

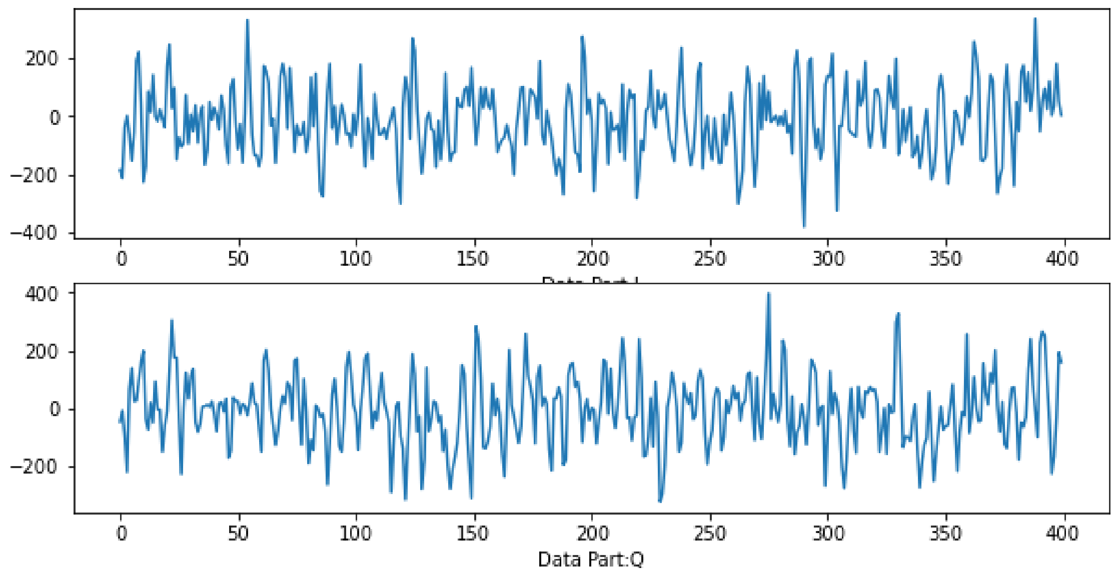
[6]:
```
(2880000,)
```

[7]:
```
fig = plt.figure(figsize=(10,6))
plt.plot(data[:samplesPerCode])
```

[7]:
```
[<matplotlib.lines.Line2D at 0x7f67d1180ac0>]
```



[8]:
```
if fileType==2:
    I = data[::2]
    Q = data[1::2]
    fig = plt.figure(figsize=(10,5))
    plt.subplot(211)
    plt.plot(I[:samplesPerCode//10])
    plt.xlabel('Data Part:I')
    plt.subplot(212)
    plt.plot(Q[:samplesPerCode//10])
    plt.xlabel('Data Part:Q')
```



[9]:
```
if fileType==2:
    signal = I + 1j*Q
else:
    signal = data
```

[10]:
```
print(np.where(signal==(-92-25j)))
print(np.where(signal==(53-71j)))
print(signal[2496])
```

```
(array([   2492,   86078,  599934,  620344,  732090,  784173,  812892,
        828476, 1169118, 1379064, 1399656]),)
(array([  16358,   28296,   52998,  272318,  393501,  516280,  802791,
        815645,  877841,  950292, 1051031, 1086451, 1101761, 1390511]),)
(-9+101j)
```

[ ]:

0.1.3 Signal Processing

Downsampling

[11]:
```
if downSample==True:
    signal = signal[2492::downSample_step]
    samplesPerCode = int(samplesPerCode//downSample_step)
    fs = int(fs//downSample_step)
    print('New Signal Shape:',signal.shape)
```

```
New Signal Shape: (718754,)
```

low_pass(double gain, double sampling_freq, double cutoff_freq, double transition_width, win_type window = WIN_HAMMING, double beta = 6.76)

0.2 Acquisition

[12]:
```
%%latex
\begin{equation}
\label{xin} x_\text{IN}[k] = A(t)\tilde{s}_{T}(t - \tau(t))e^{j \left( 2\pi␣
↪f_D(t) t + \phi(t) \right) } \Bigr \rvert_{t=kT_s} + n(t) \Bigr␣
↪\rvert_{t=kT_s}
\end{equation}
```

$$x_{\text{IN}}[k] = A(t)\tilde{s}_{T}(t - \tau(t))e^{j \left(2\pi f_D(t) t + \phi(t) \right) } \Bigr \rvert_{t=kT_s} + n(t) \Bigr \rvert_{t=kT_s} \tag{1}$$

[13]:
```
signal1 = signal[0:samplesPerCode]
#signal1 = signal1 - np.mean(signal1)
signal2 = signal[samplesPerCode:2*samplesPerCode]
testSignal_2s = signal[samplesPerCode*2:samplesPerCode*3]
signal0DC = signal-np.mean(signal)
ts = 1/fs
```

[14]:
```
testSignal_2s.shape
```

[14]: (2000,)

0.2.1 CA Code Generation

$$G_1 = x^3 + x^{10}$$

$$G_2 = x^2 + x^3 + x^6 + x^8 + x^{10}$$

[15]:
```
def generateCACode(prn):
    g2s = [  5,   6,   7,   8,  17,  18, 139, 140, 141, 251,
       252, 254, 255, 256, 257, 258, 469, 470, 471, 472,
       473, 474, 509, 512, 513, 514, 515, 516, 859, 860,
       861, 862,863,950,947,948,950]

    g2shift = g2s[prn]

    g1 = np.zeros((1023))
    reg = -1*np.ones((10))
    for i in range(codeLength):
        g1[i]      = reg[9]
        saveBit    = reg[2]*reg[9]
        reg[1:10]   = reg[0:9]
        reg[0]     = saveBit
```

```
    g2 = np.zeros((1023))
    reg = -1*np.ones((10))
    for i in range(codeLength):
        g2[i] = reg[9]
        saveBit     = reg[1]*reg[2]*reg[5]*reg[7]*reg[8]*reg[9]
        reg[1:10]   = reg[0:9]
        reg[0]      = saveBit

    g2 = np.concatenate([g2[1023-g2shift:],g2[0:1023-g2shift]])
    CAcode = -1*np.multiply(g1,g2)
    return CAcode

def makeCATable():
    caCodesTable = np.zeros((acqSatelliteList.shape[0],samplesPerCode))
    ts = 1/fs
    tc = 1/codeFreqBasis

    for i in acqSatelliteList:
        caCode = generateCACode(i)
        cvi = ts * np.arange(1,samplesPerCode+1)/ tc
        codeValueIndex = np.ceil(cvi-1)

        # Correct the last index (due to number rounding issues) -----------
        codeValueIndex[-1] = 1022
        codeValueIndex = list(map(int,list(codeValueIndex)))
        # Make the digitized version of the C/A code -----------------------
        #The "upsampled" code is made by selecting values form the CA code
        # chip array (caCode) for the time instances of each sample.
        caCodesTable[i, :] = caCode[codeValueIndex]


    return caCodesTable
```

0.2.2 Threshold Calculation

Reference: Incomplete Gamma Functions

$$\Gamma(s,x) = \int_x^\infty t^{s-1} * e^{-t}\, dt$$

[16]:
```
d_num_doppler_bins = (doppler_max-(-doppler_max))//doppler_step
effective_fft_size = samplesPerCode
num_doppler_bins = d_num_doppler_bins
num_bins = int(effective_fft_size * num_doppler_bins)
d_threshold = 2*sc.gammaincinv(2,np.power(1-pfa,1/(num_bins)))
```

```
print("Threshold calculated with number of bin:{} is:{}".
 ↪format(num_bins,d_threshold))
```

```
Threshold calculated with number of bin:160000 is:39.217592689170104
```

[17]:
```
phasePoints = (np.arange(0,samplesPerCode))*2*np.pi/fs
numberOfFrqBins = round(acqSearchBand*2)+1
caCodesTable = makeCATable()
results = np.zeros((num_doppler_bins,samplesPerCode))
frqBins = np.zeros((num_doppler_bins))
```

0.2.3 Metrics Calculation

[18]:
```
def nextpow2(x):
    return 1 if x == 0 else 2**np.ceil(np.log2(abs(x)))
```

$$P_{IN} = \frac{1}{K} * \sum_{k=0}^{K} |x_{in}[k]|^2$$

[19]:
```
def input_power(INsignal,effective_fft_size=0):
    s_in = INsignal[:effective_fft_size]
    power = np.sum(np.power(np.abs(s_in),2))
    K = s_in.shape[0]
    return power/K
```

[20]:
```
inputPower = input_power(signal1,effective_fft_size)
print('Input Signal Power is:',inputPower)
```

```
Input Signal Power is: 30607.2565
```

Max Power to Input Power

[21]:
```
carrFreq = np.zeros((acqSatelliteList.shape[0]))
codePhaseRes = np.zeros((acqSatelliteList.shape[0]))
peakMetric = np.zeros((acqSatelliteList.shape[0]))
magnitudeGrid = np.zeros((num_doppler_bins,samplesPerCode))
```

[22]:
```
def MP_IP_acqResults(Insignal,effective_fft_size):
    x_in = Insignal[0:effective_fft_size]
    for i in acqSatelliteList[:]:
        caCodesT = caCodesTable[i]
        caCodeFreqDom = np.conj(np.fft.fft(caCodesT,effective_fft_size))
        for j in range(num_doppler_bins):
            fd = IF - doppler_max + doppler_step * j#doppler wipeoff
            signalCarr = np.exp(1j*(fd*phasePoints))
            # "Remove carrier" from the signal ------------------------------
            x_k = np.multiply(x_in,signalCarr)#I1+Q1*1j
            x_K = np.fft.fft(x_k,effective_fft_size)
```

```
            Y_k = np.multiply(x_K,caCodeFreqDom)
            Y_K= np.fft.ifft(Y_k,effective_fft_size)
            magnitudeGrid[j,:] = np.multiply((1/(effective_fft_size))
                                             ,Y_K)

        magGrid = np.power(np.abs(magnitudeGrid),2)

        #max_to_inputPower = magnitudeGrid/inputPower
        np.savetxt('results/results_{}.csv'.format(str(i)), magGrid,␣
↪delimiter=',', fmt='%s')
        #np.savetxt('results/input_{}.csv'.
↪format(str(i)),x_in,delimiter=',',fmt="%s")
        # Looking for correlation peak
        frequencyBinIndex_MI = np.max(np.argmax(magGrid,axis=0))
        peakSize_MI = np.max(np.max(magGrid))
        #print(peakSize,frequencyBinIndex)

        # Find code phase of the same correlation peak ---------------------
        codePhase_MI = np.max(np.argmax(magGrid,axis=1))

        #print(peakSize,codePhase)
        # Find 1 chip wide C/A code phase exclude range around the peak ----
        samplesPerCodeChip   = round(fs /codeFreqBasis)
        excludeRangeIndex1 = codePhase_MI - samplesPerCodeChip
        excludeRangeIndex2 = codePhase_MI + samplesPerCodeChip

        # Correct C/A code phase exclude range if the range includes array
        #boundaries
        if excludeRangeIndex1 < 2:
            codePhaseRange = np.arange(excludeRangeIndex2 ,(samplesPerCode +␣
↪excludeRangeIndex1-1))

        elif excludeRangeIndex2 >= samplesPerCode:
            codePhaseRange = np.arange(excludeRangeIndex2 -␣
↪samplesPerCode,excludeRangeIndex1-1)
        else:
            codePhaseRange = np.hstack((np.arange(1,excludeRangeIndex1),
            np.arange(excludeRangeIndex2 ,samplesPerCode-1)))

        GLRT_variance = 2*effective_fft_size*peakSize_MI/inputPower
        #print(GLRT_variance)

        # Store result ---------------------------------------------------
        peakMetric[i] = GLRT_variance

        # If the result is above threshold, then there is a signal
        if GLRT_variance > (d_threshold):
```

```
            caCode = generateCACode(i)
            codeValueIndex = np.floor((ts * np.arange(1,10*samplesPerCode)) /
                                      (1/codeFreqBasis))
            #print(codeValueIndex)
            #print(np.remainder(codeValueIndex,1023) + 1)
            longCaCode = caCode[list((np.remainder(codeValueIndex,1023).
↪astype(np.int8) + 1))]

            # Remove C/A code modulation from the original signal ----------
            # (Using detected C/A code phase)
            xCarrier = np.multiply(signal[codePhase_MI:(codePhase_MI +␣
↪10*samplesPerCode-1)]
                , longCaCode)

            # Compute the magnitude of the FFT, find maximum and the
            #associated carrier frequency
            nextPow2=np.ceil(np.log2(abs(len(xCarrier)))).astype('int')
            # Find the next highest power of two and increase by 8x --------
            fftNumPts = 8*(2**(nextPow2))

            # Compute the magnitude of the FFT, find maximum and the
            #associated carrier frequency
            fftxc = abs(np.fft.fft(xCarrier, fftNumPts))


            uniqFftPts = np.ceil((fftNumPts + 1) / 2)
            [fftMax, fftMaxIndex] = np.max(fftxc),np.argmax(fftxc)
            fftFreqBins = np.arange(0 ,uniqFftPts-1) *fs/fftNumPts

            if (fftMaxIndex > uniqFftPts): #%and should validate using complex␣
↪data
                if ((fftNumPts%2)==0):  #even number of points, so DC and Fs/2␣
↪computed
                    fftFreqBinsRev=-fftFreqBins[(uniqFftPts-1):-1:2]
                    fftMax, fftMaxIndex = np.max(np.
↪arange(fftxc[(uniqFftPts+1),len(fftxc)])),
                    np.argmax(fftxc[(uniqFftPts+1),len(fftxc)])
                    carrFreq[i]  = -fftFreqBinsRev[fftMaxIndex]
                else:  #%odd points so only DC is not included
                    fftFreqBinsRev=-np.flip(fftFreqBins[1:(uniqFftPts)])
                    fftMax, fftMaxIndex = np.amax(fftxc[(uniqFftPts+1):
↪len(fftxc)]),
                    np.argmax(fftxc[(uniqFftPts):len(fftxc)])
                    carrFreq[i] = doppler_max-frequencyBinIndex_MI*0.
↪25e3#fftFreqBinsRev[fftMaxIndex]
```

```
            else:
                carrFreq[i]  =doppler_max-frequencyBinIndex_MI*0.25e3␣
↪#(-1)**(fileType-1)*fftFreqBins[fftMaxIndex]


            codePhaseRes[i] = codePhase_MI
            print(i+1,end =" ")
        else:
            # No signal with this PRN --------------------------------------
            print('. ',end =" ")
    return peakMetric,codePhaseRes,carrFreq,magGrid
```

[23]:

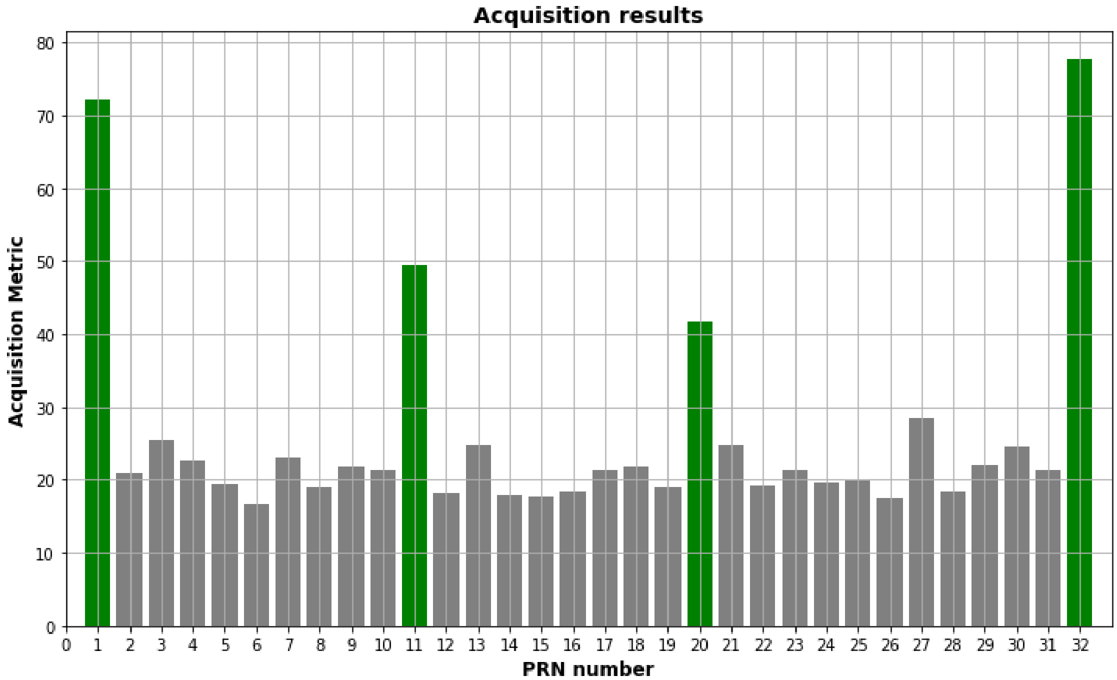
```
if CFAR==1:
    MP_peakMetric,codePhases,frequencyInd,MP_results =␣
↪MP_IP_acqResults(signal1,effective_fft_size)
```

```
/tmp/ipykernel_211143/1061446624.py:14: ComplexWarning: Casting complex values
to real discards the imaginary part
  magnitudeGrid[j,:] = np.multiply((1/(effective_fft_size))
```

```
1 .  .  .  .  .  .  .  .  .  11 .  .  .  .  .  .  .  .  20 .  .  .  .  .  .  .
.  .  .  .  32
```

[24]:

```
if CFAR==1:
    fig = plt.figure(figsize=(12,7))
    colors = np.where(MP_peakMetric>d_threshold,'green','grey')
    plt.bar(list(range(1,acqSatelliteList.
↪shape[0]+1)),MP_peakMetric,color=colors)
    plt.grid()
    plt.title ( 'Acquisition results',fontdict={'fontsize':14,'fontweight':
↪'bold'})
    plt.xlim([0,acqSatelliteList.shape[0]+1])
    #plt.yticks(np.arange(0,max(MP_peakMetric),2))
    plt.xticks(np.arange(0,acqSatelliteList.shape[0]+1,1))
    plt.xlabel( 'PRN number',
               fontdict={'fontsize':12,'fontweight':'bold'})
    plt.ylabel( 'Acquisition Metric',fontdict={'fontsize':12,'fontweight':
↪'bold'})
```



[25]:
```
from scipy.io import savemat
savemat("matlab_matrix.mat", {"results":MP_results})
```

First Peak to Second Peak

[26]:
```
if CFAR==0:
    carrFreq = np.zeros((acqSatelliteList.shape[0]))
    codePhaseRes = np.zeros((acqSatelliteList.shape[0]))
    frequencyRes = np.zeros((acqSatelliteList.shape[0]))
    peakMetric = np.zeros((acqSatelliteList.shape[0]))
```

[27]:
```
def acqResults():
    for i in acqSatelliteList:
        caCodesT = caCodesTable[i]
        caCodeFreqDom = np.conj(np.fft.fft(caCodesT))
        for j in range(num_doppler_bins):

            # Generate carrier wave frequency grid (0.5kHz step) -----------
            frqBins[j] = IF - (acqSearchBand/2) * 1000 + 0.25e3 * j

            # Generate local sine and cosine -------------------------------
            sigCarr = np.exp(1j*frqBins[j]*phasePoints)

            # "Remove carrier" from the signal -----------------------------
            I1      = np.real(sigCarr * signal1)
```

```
        Q1      = np.imag(sigCarr * signal1)
        I2      = np.real(sigCarr * signal2)
        Q2      = np.imag(sigCarr * signal2)

        # Convert the baseband signal to frequency domain --------------
        IQfreqDom1 = np.fft.fft(I1 + 1j*Q1)
        IQfreqDom2 = np.fft.fft(I2 + 1j*Q2)

        # Multiplication in the frequency domain (correlation in␣
↪time%domain)
        convCodeIQ1 = np.multiply(IQfreqDom1, caCodeFreqDom)
        convCodeIQ2 = np.multiply(IQfreqDom2 , caCodeFreqDom)

        # Perform inverse DFT and store correlation results ------------
        acqRes1 =np.power(abs(np.fft.ifft(convCodeIQ1)),2)
        acqRes2 = np.power(abs(np.fft.ifft(convCodeIQ2)),2)

        # Check which msec had the greater power and save that, will
        # 1st and 2nd msec but will correct data bit issues
        if (max(acqRes1) > max(acqRes2)):
            results[j, :] = acqRes1
        else:
            results[j, :] = acqRes2


    # Looking for correlation peaks
    peakSize= np.max(np.max(results,axis=0))
    frequencyBinIndex = np.max(np.argmax(results,axis=0))#np.amax(np.
↪where(results[j,:]==peakSize))

    # Find code phase of the same correlation peak ---------------------
    codePhase = np.max(np.argmax(results,axis=1))

    #print(peakSize,codePhase)
    # Find 1 chip wide C/A code phase exclude range around the peak ----
    samplesPerCodeChip   = round(fs /codeFreqBasis)
    excludeRangeIndex1 = codePhase - samplesPerCodeChip
    excludeRangeIndex2 = codePhase + samplesPerCodeChip

    # Correct C/A code phase exclude range if the range includes array
    #boundaries
    if excludeRangeIndex1 < 2:
        codePhaseRange = np.arange(excludeRangeIndex2 ,(samplesPerCode +␣
↪excludeRangeIndex1-1))

    elif excludeRangeIndex2 >= samplesPerCode:
```

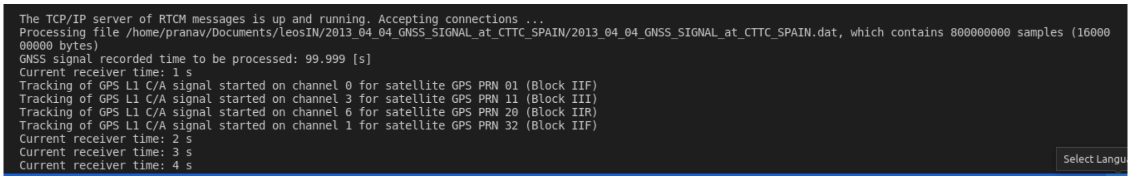
```
            codePhaseRange = np.arange(excludeRangeIndex2 -␣
↪samplesPerCode,excludeRangeIndex1-1)
        else:
            codePhaseRange = np.hstack((np.arange(1,excludeRangeIndex1),np.
↪arange(excludeRangeIndex2 ,samplesPerCode-1)))

        # Find the second highest correlation peak in the same freq. bin ---
        secondPeakSize = np.amax(results[frequencyBinIndex, codePhaseRange])
        #print(peakSize/secondPeakSize)

        # Store result -----------------------------------------------------
        peakMetric[i] = peakSize/secondPeakSize

        # If the result is above threshold, then there is a signal
        if (peakSize/secondPeakSize) > acqThreshold:
            caCode = generateCACode(i)
            codeValueIndex = np.floor((ts * np.arange(1,10*samplesPerCode)) /
                                  (1/codeFreqBasis))
            #print(codeValueIndex)
            #print(np.remainder(codeValueIndex,1023) + 1)
            longCaCode = caCode[list((np.remainder(codeValueIndex,1023).
↪astype(np.int8) + 1))]

            # Remove C/A code modulation from the original signal ----------
            # (Using detected C/A code phase)
            xCarrier = np.multiply(signal0DC[codePhase:(codePhase +␣
↪10*samplesPerCode-1)]
                , longCaCode)

            # Compute the magnitude of the FFT, find maximum and the
            #associated carrier frequency
            nextPow2=np.ceil(np.log2(abs(len(xCarrier)))).astype('int')
            # Find the next highest power of two and increase by 8x --------
            fftNumPts = 8*(2**(nextPow2))

            # Compute the magnitude of the FFT, find maximum and the
            #associated carrier frequency
            fftxc = abs(np.fft.fft(xCarrier, fftNumPts))


            uniqFftPts = np.ceil((fftNumPts + 1) / 2)
            [fftMax, fftMaxIndex] = np.max(fftxc),np.argmax(fftxc)
            fftFreqBins = np.arange(0 ,uniqFftPts-1) *fs/fftNumPts

            if (fftMaxIndex > uniqFftPts): #%and should validate using complex␣
↪data
```

```
                if (np.remainder(fftNumPts,2)==0):  #even number of points, so␣
↪DC and Fs/2 computed
                    fftFreqBinsRev=-fftFreqBins[(uniqFftPts-1):-1:2]
                    fftMax, fftMaxIndex = np.max(np.
↪arange(fftxc[(uniqFftPts+1),len(fftxc)])),np.argmax(np.
↪arange(fftxc[(uniqFftPts+1),len(fftxc)]))
                    np.argmax(fftxc[(uniqFftPts+1),len(fftxc)])
                    carrFreq[i]  = -fftFreqBinsRev[fftMaxIndex]
                else:  #%odd points so only DC is not included
                    fftFreqBinsRev=-np.flip(fftFreqBins[2:(uniqFftPts)])
                    [fftMax, fftMaxIndex] = np.amax(fftxc[(uniqFftPts+1):
↪len(fftxc)])
                    ,np.argmax(fftxc[(uniqFftPts+1):len(fftxc)])
                    carrFreq[i] = fftFreqBinsRev[fftMaxIndex]

            else:
                carrFreq[i]  = (-1)**(fileType-1)*fftFreqBins[fftMaxIndex]


            codePhaseRes[i] = codePhase
            frequencyRes[i] = frequencyBinIndex
            print(i+1,end =" ")
        else:
            # No signal with this PRN --------------------------------------
            print('. ',end =" ")
    return peakMetric,codePhaseRes,carrFreq,results
```

[28]:
```
if CFAR==0:
    peakMetric,codePhases,frequencyInd,results= acqResults()
```

[29]:
```
if CFAR==0:
    fig = plt.figure(figsize=(12,7))
    colors = np.where(peakMetric>acqThreshold,'green','teal')
    plt.bar(list(range(1,acqSatelliteList.shape[0]+1)),peakMetric,color=colors)
    plt.grid()
    plt.title ( 'Acquisition results',fontdict={'fontsize':14,'fontweight':
↪'bold'})
    plt.xlim([0,acqSatelliteList.shape[0]+1])
    plt.yticks(np.arange(0,max(peakMetric),0.5))
    plt.xticks(np.arange(0,acqSatelliteList.shape[0]+1,1))
    plt.xlabel( 'PRN number',
               fontdict={'fontsize':12,'fontweight':'bold'})
    plt.ylabel( 'Acquisition Metric',fontdict={'fontsize':12,'fontweight':
↪'bold'})
```

0.2.4 Expected Results



0.3 Channel

[30]:
```
numberOfChannels = 8
channelPRN = np.zeros((numberOfChannels),dtype=np.int16)
acquiredFreq = np.zeros((numberOfChannels))
codeChPhases = np.zeros((numberOfChannels),dtype=np.int16)
```

[31]:
```
sorted_peaks = np.sort(MP_peakMetric,axis=0,)[::-1]

sorted_peaks_ind = np.argsort(MP_peakMetric,axis=0)[::-1]
```

[32]:
```
if len(frequencyInd[frequencyInd!=0])<numberOfChannels:
    noOfChannels = len(frequencyInd[frequencyInd!=0])
    indexes = sorted_peaks_ind[:noOfChannels]
    channelPRN[:noOfChannels] = sorted_peaks_ind[:noOfChannels]+1
    acquiredFreq[:noOfChannels] = frequencyInd[indexes]
    codeChPhases[:noOfChannels] = codePhases[indexes]
satList = pd.DataFrame.from_dict({'Satellites':channelPRN,"Doppler":
 ↪acquiredFreq,'CodePhases':codeChPhases})
satList.head()
```

[32]:

	Satellites	Doppler	CodePhases
0	32	-9750.0	1984
1	1	-9750.0	1971
2	11	-9750.0	1971
3	20	-9750.0	1894
4	0	0.0	0

0.4 Tracking

[33]:
```
#Tracking loops settings ================================================
enableFastTracking     = 0

# Code tracking loop parameters
dllDampingRatio         = 0.707
dllNoiseBandwidth       = 15        #Hz
dllCorrelatorSpacing    = 0.5      #chips
# Carrier tracking loop parameters
pllDampingRatio         = 0.707
pllNoiseBandwidth       = 15       #db
```

```
fllDampingRatio         = 0.7
fllNoiseBandwidth       = 10      #Hz
```

[34]:
```
status          = '-'       # No tracked signal, or lost lock
msToProcess = len(signal)//samplesPerCode
numberOfChannels = 8

#Tracking Constants
accTime=0.001
enableVSM=0
PRM_K=200
PRM_M=20
MOMinterval=200
Plot = 1
VSMinterval = 400

# The absolute sample in the record of the C/A code start:
absoluteSample = np.zeros((numberOfChannels, msToProcess))

# Freq of the C/A code:
codeFreq        = np.full((numberOfChannels, msToProcess),np.inf)
rCodePhase= np.full((numberOfChannels, msToProcess),np.inf)#record codephase zsh

# Frequency of the tracked carrier wave:
carrFreq        = np.full((numberOfChannels, msToProcess),np.inf)
rCarrPhase= np.full((numberOfChannels, msToProcess),np.inf)#record carrier␣
 ↪phase zsh
```

[35]:
```
# Outputs from the correlators (In-phase):
IP              = np.zeros((numberOfChannels, msToProcess))
IE              = np.zeros((numberOfChannels, msToProcess))
IL              = np.zeros((numberOfChannels, msToProcess))

# Outputs from the correlators (Quadrature-phase):
QE              = np.zeros((numberOfChannels, msToProcess))
QP              = np.zeros((numberOfChannels, msToProcess))
QL              = np.zeros((numberOfChannels, msToProcess))

# Loop discriminators
dllDiscr        = np.full((numberOfChannels, msToProcess),np.inf)
dllDiscrFilt    = np.full((numberOfChannels, msToProcess),np.inf)
pllDiscr        = np.full((numberOfChannels, msToProcess),np.inf)
pllDiscrFilt    = np.full((numberOfChannels, msToProcess),np.inf)

#C/No
VSMValue = np.zeros((numberOfChannels,np.floor(msToProcess/VSMinterval).
 ↪astype(np.int16)))
```

```
VSMIndex = np.zeros((numberOfChannels,np.floor(msToProcess/VSMinterval).
↪astype(np.int16)))

PRMValue=0 #To avoid error message when
PRMIndex=0 #tracking window is closed before completion.

#--- Copy initial settings for all channels -------------------------------
#trackResults = repmat(trackResults, 1, numberOfChannels)
trackingPRN = np.zeros((numberOfChannels))
# Initialize tracking variables ==========================================

codePeriods = msToProcess     #For GPS one C/A code is one ms

#--- DLL variables --------------------------------------------------------
# Define early-late offset (in chips)
earlyLateSpc = dllCorrelatorSpacing
PDIcarr = 0.001
PDIcode = 0.001

##Active Status
channelStatus = np.zeros((numberOfChannels))
activeChList = np.zeros((numberOfChannels),dtype=int)
```

0.4.1 Helper Functions

[36]:
```
def calculateLoopCoeff(noise_bw,damping_ration,k):
    Wn = noise_bw*8*damping_ration / (4*(damping_ration**2) + 1)
    # solve for t1 & t2
    tau1 = (Wn * Wn)
    tau2 = 2.0 * damping_ration * Wn
    return tau1,tau2
```

[37]:
```
tau1code, tau2code = calculateLoopCoeff(dllNoiseBandwidth,dllDampingRatio,0.25)
tau1carr, tau2carr = calculateLoopCoeff(dllNoiseBandwidth,dllDampingRatio,0.25)
print(tau1code,tau2code)
```

```
800.0805333300673 39.995972522467845
```

0.4.2 Tracking Process

[39]:
```
status=[]
for channelNr in tqdm(range(numberOfChannels),'Channel Progress'):

    # Only process if PRN is non zero (acquisition was successful)
    if (channelPRN[channelNr] != 0):
        # Save additional information - each channel's tracked PRN
        trackingPRN[channelNr]     = channelPRN[channelNr]
```

```
# Move the starting point of processing. Can be used to start the
# signal processing at any point in the data record (e.g. for long
# records). In addition skip through that data file to start at the
# appropriate sample (corresponding to code phase). Assumes sample
# type is schar (or 1 byte per sample)

# Get a vector with the C/A code sampled 1x/chip
caCode = generateCACode(channelPRN[channelNr]-1)
# Then make it possible to do early and late versions
caCode =np.hstack((caCode[-1],caCode,caCode[0]))
#print('CaCode shape'.format(caCode.shape),end=";\n")
#--- Perform various initializations ------------------------------

# define initial code frequency basis of NCO
codeFrq      = codeFreqBasis
# define residual code phase (in chips)
remCodePhase  = 0.0
# define carrier frequency which is used over whole tracking period
carrFrq     = acquiredFreq[channelNr]
carrFreqBasis = acquiredFreq[channelNr]
# define residual carrier phase
remCarrPhase  = 0.0

#code tracking loop parameters
oldCodeNco    = 0.0
oldCodeError = 0.0

#carrier/Costas loop parameters
oldCarrNco    = 0.0
oldCarrError = 0.0
dataOffset = 0
#C/No computation
vsmCnt  = 0
if (enableVSM==1):
    CNo='Calculating...'
else:
    CNo='Disabled'
#print("{}".format(channelNr),end='\n')
#=== Process the number of specified code periods =================
for loopCnt in range(codePeriods):

    Ln='\n'
    trackingStatus=['Tracking: Ch ', str(channelNr),
        ' of ', str(numberOfChannels),
        'PRN: ', str(channelPRN[channelNr]),
        'Completed ',str(loopCnt),
```

```
                ' of ', str(codePeriods), ' msec',
                'C/No: ',CNo,' (dB-Hz)']

            #if(np.remainder(loopCnt,50)==0):
            #    print(str(trackingStatus)

            # Read next block of data␣
↪-------------------------------------------------
            # Find the size of a "block" or code period in whole samples

            # Update the phasestep based on code freq (variable) and
            # sampling frequency (fixed)
            codePhaseStep = codeFrq / fs
            #print("CodePhaseStep",codePhaseStep,end=";\n")
            blksize = np.ceil((codeLength-remCodePhase) / codePhaseStep).
↪astype(int)-1
            #print("Block Size:{}".format(blksize),end=";\n")

            # Read in the appropriate number of samples to process this
            # interation
            rawSignal = np.fromfile(fileName,offset=dataOffset,
               count =int(fileType*blksize),dtype=np.int16)
            #rawSignal = rawSignal[2492:]
            samplesRead = rawSignal.shape[0]
            dataOffset = int(fileType*blksize)
            #print('Offset',fileType*blksize)

            if (fileType==2):
                rawSignal1 = rawSignal[::2]
                rawSignal2 = rawSignal[1::2]
                rawSignal = np.add(rawSignal1, 1j* rawSignal2)  #transpose␣
↪vector

                #rawSignal = rawSignal[::2]
                #print('Complex'dataOffset = 0,rawSignal.shape)
            # If did not read in enough samples, then could be out of
            # data - better exit
            if (samplesRead != fileType*blksize):
                print('Not able to read the specified number of samples  for␣
↪tracking, exiting!')


            # Set up all the code phase tracking information␣
↪-------------------------
            # Define index into early code vector
            tcode       = np.arange((remCodePhase-earlyLateSpc)-1,
```

```
                                   ␣
↪((blksize)*codePhaseStep+remCodePhase-earlyLateSpc)-1
                                   ,codePhaseStep)
          tcode2      = (np.ceil(tcode) ).astype(int)

          earlyCode   = caCode[tcode2]
          #print((((blksize)*codePhaseStep)+(remCodePhase+earlyLateSpc))-1)

          # Define index into late code vector
          tcode       = np.arange((remCodePhase+earlyLateSpc)-1,
                                   ␣
↪(((blksize)*codePhaseStep)+(remCodePhase+earlyLateSpc))-1,
                                   codePhaseStep)

          tcode2      = np.ceil(tcode).astype(int)
          #print('Iter:{} : tcode2:{}'.format(loopCnt,np.max(tcode2)),end=";
↪\n..........................\n")
          lateCode    = caCode[tcode2]

          # Define index into prompt code vector
          tcode       = np.
↪arange(remCodePhase-1,((blksize)*codePhaseStep+remCodePhase)-1
                                   , codePhaseStep)
          tcode2      = np.ceil(tcode).astype(int)
          promptCode  = caCode[tcode2]

          remCodePhase = (tcode[blksize-1] + codePhaseStep) - 1022

          # Generate the carrier frequency to mix the signal to baseband␣
↪-----------
          time    = np.arange(0,blksize)/ fs

          # Get the argument to sin/cos functions
          trigarg = ((carrFrq * 2.0 * np.pi)* time) + remCarrPhase
          remCarrPhase = np.remainder(trigarg[blksize-1], (2 * np.pi))

          # Finally compute the signal to mix the collected data to bandband
          carrsig = np.exp(1j* trigarg[0:blksize])

          # Generate the six standard accumulated values␣
↪---------------------------
          # First mix to baseband
          qBasebandSignal = np.real(carrsig * rawSignal)
          iBasebandSignal = np.imag(carrsig * rawSignal)

          # Now get early, late, and prompt values for each
```

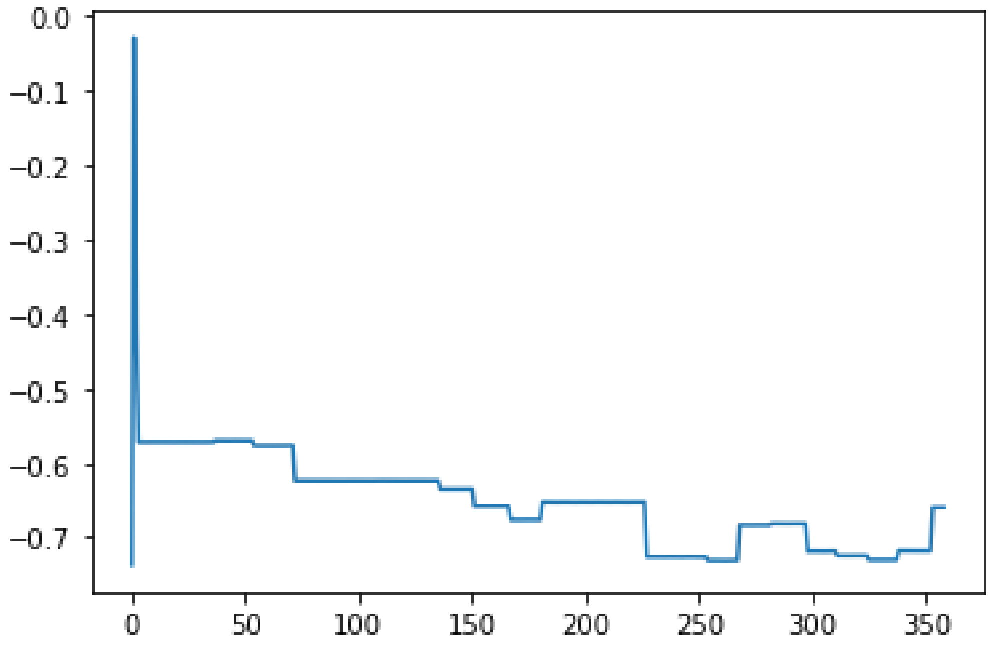
```
            I_E = np.sum(earlyCode  * iBasebandSignal)
            Q_E = np.sum(earlyCode  * qBasebandSignal)
            I_P = np.sum(promptCode * iBasebandSignal)
            Q_P = np.sum(promptCode * qBasebandSignal)
            I_L = np.sum(lateCode   * iBasebandSignal)
            Q_L = np.sum(lateCode   * qBasebandSignal)

            ##Find PLL error and update carrier NCO␣
↪----------------------------------

            # Implement carrier loop discriminator (phase detector)
            carrError = np.arctan2(Q_P , I_P)

            # Implement carrier loop filter and generate NCO command
            carrNco = oldCarrNco + (tau2carr/tau1carr) * (carrError -␣
↪oldCarrError)
            + carrError * (PDIcarr/tau1carr)
            oldCarrNco   = carrNco
            oldCarrError = carrError

            # Modify carrier freq based on NCO command
            carrFrq = carrFreqBasis + carrNco

            carrFreq[channelNr,loopCnt] = carrFrq
            rCarrPhase[channelNr,loopCnt] = remCarrPhase
            # Find DLL error and update code NCO␣
↪-------------------------------------
            codeError = (np.sqrt(I_E * I_E + Q_E * Q_E) - np.sqrt(I_L * I_L +␣
↪Q_L * Q_L)) /(np.sqrt(I_E * I_E + Q_E * Q_E) + np.sqrt(I_L * I_L + Q_L *␣
↪Q_L))

            # Implement code loop filter and generate NCO command
            codeNco = oldCodeNco + (tau2code/tau1code) * (codeError -␣
↪oldCodeError) + codeError * (PDIcode/tau1code)
            oldCodeNco   = codeNco
            oldCodeError = codeError

            # Modify code freq based on NCO command
            codeFrq = codeFreqBasis - codeNco

            codeFreq[channelNr,loopCnt] = codeFrq
            rCodePhase[channelNr,loopCnt] = remCodePhase

            # Record various measures to show in postprocessing␣
↪----------------------
            # Record sample number (based on 8bit samples)
```

```
            absoluteSample[channelNr,loopCnt] = samplesRead/fileType-␣
↪remCodePhase/codePhaseStep

            dllDiscr[channelNr,loopCnt]       = codeError
            dllDiscrFilt[channelNr,loopCnt]   = codeNco
            pllDiscr[channelNr,loopCnt]   = carrError
            pllDiscrFilt[channelNr,loopCnt]   = carrNco

            IE[channelNr,loopCnt] = I_E
            IP[channelNr,loopCnt] = I_P
            IL[channelNr,loopCnt] = I_L
            QE[channelNr,loopCnt] = Q_E
            QP[channelNr,loopCnt] = Q_P
            QL[channelNr,loopCnt] = Q_L



            if (enableVSM==1):
                if (np.remainder(loopCnt,VSMinterval)==0):
                    vsmCnt=vsmCnt+1
                    CNoValue=CNoVSM(I_P[loopCnt-VSMinterval+1:loopCnt],
                        Q_P[loopCnt-VSMinterval+1:loopCnt],accTime)
                    VSMValue[vsmCnt]=CNoValue
                    VSMIndex[vsmCnt]=loopCnt
                    CNo=int(CNoValue)

            activeChList[channelNr] = channelNr+1

            # Evaluate the tracking results status here to ensure the
            # plotTracking to plot the results tracked so far
            # (In case the tracking update window is closed)
            #status  = channel[channelNr].status
        # for loopCnt

        # If we got so far, this means that the tracking was successful
        # Now we only copy status, but it can be update by a lock detector
        # if implemented
        #status  = channel(channelNr).status
```

Channel Progress: 100%|
| 8/8 [00:00<00:00,
11.83it/s]

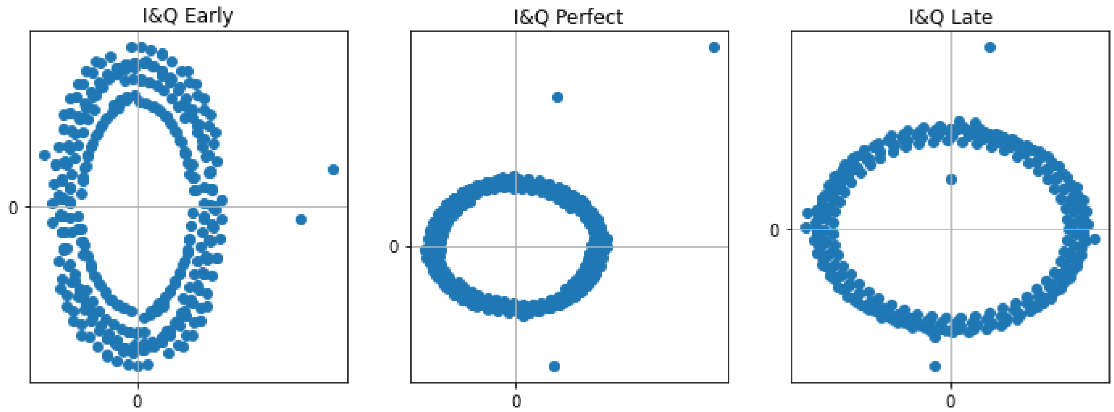
[40]:
```
plt.plot(dllDiscr[0,:])
```

[40]: [<matplotlib.lines.Line2D at 0x7f67d027ff10>]



[41]:
```
fig = plt.figure(figsize=(12,4))
plt.subplot(131)
plt.scatter(IE[0,:],QE[0,:])
plt.grid()
plt.title('I&Q Early')
plt.xticks([0])
plt.yticks([0])
plt.subplot(132)
plt.scatter(IP[0,:],QP[0,:])
plt.grid()
plt.title('I&Q Perfect')
plt.xticks([0])
plt.yticks([0])
plt.subplot(133)
plt.scatter(IL[0,:],QL[0,:])
plt.grid()
plt.title('I&Q Late')
plt.xticks([0])
plt.yticks([0])
```

[41]:
```
([<matplotlib.axis.YTick at 0x7f67d00ce1c0>], [Text(0, 0, '')])
```



0.5 Navigation Solution

[42]:
```
def findPreambles():
    searchStartOffset = 0

    #--- Initialize the firstSubFrame array -----------------------------------
    firstSubFrame = np.zeros(( numberOfChannels))

    #--- Generate the preamble pattern ----------------------------------------
    preamble_bits = [1 ,0,0,0, 1 ,0 ,1, 1]

    # "Upsample" the preamble - make 20 values per one bit. The preamble must be
    # found with precision of a sample.
    preamble_ms = np.kron(preamble_bits, np.ones((20)))

    #--- Make a list of channels excluding not tracking channels --------------
    activeChnList = activeChList[activeChList!=0]

    #=== For all tracking channels ...
    for channelNr in activeChnList:

    ## Correlate tracking output with preamble ================================
        # Read output from tracking. It contains the navigation bits. The start
        # of record is skiped here to avoid tracking loop transients.
        bits = IP[channelNr,0 + searchStartOffset : ]
        # Now threshold the output and convert it to -1 and +1
        bits[bits > 0]  =  1
        bits[bits <= 0] = 0

        # Correlate tracking output with the preamble
        tlmXcorrResult = sp.correlate(bits, preamble_ms)
        xcorrLength = (len(tlmXcorrResult)) //2
```

```
        tlmXCorr = tlmXcorrResult[xcorrLength-1: xcorrLength * 2 - 1]
        #print(tlmXCorr)
        #--- Find at what index/ms the preambles start ------------------------
        index = abs(tlmXCorr[tlmXCorr> 5])
        + searchStartOffset

        progressString = "Preamble Pattern for channel {}".format(channelNr)

        ## Analyze detected preamble like patterns␣
↪================================
        for i in tqdm(range(len(index)),progressString): # For each occurrence

            #--- Find distances in time between this occurrence and the rest of
            #preambles like patterns. If the distance is 6000 milliseconds (one
            #subframe), the do further verifications by validating the parities
            #of two GPS words

            index2 = index - index[i]-1
            if ((index2[index2 == 5999].shape[0]>0)):

                #=== Re-read bit vales for preamble verification ==============
                # Preamble occurrence is verified by checking the parity of
                # the first two words in the subframe. Now it is assumed that
                # bit boundaries a known. Therefore the bit values over 20ms are
                # combined to increase receiver performance for noisy signals.
                # in Total 62 bits mast be read :
                # 2 bits from previous subframe are needed for parity checking
                # 60 bits for the first two 30bit words (TLM and HOW words).
                # The index is pointing at the start of TLM word.
                bits = IL[channelNr,np.arange(index[i]-40,index[i] + 20 * 60␣
↪-1)]

                #--- Combine the 20 values of each bit ------------------------
                bits = np.reshape(bits, 20, (bits.shape[0]// 20))
                bits = np.sum(bits)

                # Now threshold and make it -1 and +1
                bits[bits > 0]  = 1
                bits[bits <= 0] = -1

                #--- Check the parity of the TLM and HOW words ----------------
                if (navPartyChk[bits[0:31]] != 0) and (navPartyChk[bits[31:62]]␣
↪!= 0):
                    # Parity was OK. Record the preamble start position. Skip
                    # the rest of preamble pattern checking for this channel
                    # and process next channel.
```

```
                    firstSubFrame[channelNr]= index[i]
                    #print(index[i])
                    break

        # Exclude channel from the active channel list if no valid preamble was
        # detected
        if firstSubFrame[channelNr] == 0:

            # Exclude channel from further processing. It does not contain any
            # valid preamble and therefore nothing more can be done for it.
            activeChnList = np.setdiff1d(activeChnList, channelNr)

            print(['Could not find valid preambles in channel ',
                                                                  ␣
↪str(trackingPRN[channelNr]+1),'!'])
```

[43]:
```
findPreambles()
```

```
Preamble Pattern for channel 1: 100%|
                              | 250/250 [00:00<00:00,
75507.74it/s]

['Could not find valid preambles in channel ', '2.0', '!']

Preamble Pattern for channel 2: 100%|
                              | 251/251 [00:00<00:00,
77364.07it/s]

['Could not find valid preambles in channel ', '12.0', '!']

Preamble Pattern for channel 3: 100%|
                              | 244/244 [00:00<00:00,
69014.11it/s]

['Could not find valid preambles in channel ', '21.0', '!']

Preamble Pattern for channel 4: 0it [00:00, ?it/s]

['Could not find valid preambles in channel ', '1.0', '!']
```

[44]:
```
def postNavigation(trackResults, settings):
#Function calculates navigation solutions for the receiver (pseudoranges,
#positions). At the end it converts coordinates from the WGS84 system to
#the UTM, geocentric or any additional coordinate system.

#[navSolutions, eph] = postNavigation(trackResults, settings)
#
#   Inputs:
#       trackResults    - results from the tracking function (structure
#                         array).
```

```
#       settings        - receiver settings.
#   Outputs:
#       navSolutions    - contains measured pseudoranges, receiver
#                       clock error, receiver coordinates in several
#                       coordinate systems (at least ECEF and UTM).
#       eph             - received ephemerides of all SV (structure array).

#This program is free software; you can redistribute it and/or
#modify it under the terms of the GNU General Public License
#as published by the Free Software Foundation; either version 2
#of the License, or (at your option) any later version.

#This program is distributed in the hope that it will be useful,
#but WITHOUT ANY WARRANTY; without even the implied warranty of
#MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#GNU General Public License for more details.

## Check is there enough data to obtain any navigation solution ===========
# It is necessary to have at least three subframes (number 1, 2 and 3) to
# find satellite coordinates. Then receiver position can be found too.
# The function requires all 5 subframes, because the tracking starts at
# arbitrary point. Therefore the first received subframes can be any three
# from the 5.
# One subframe length is 6 seconds, therefore we need at least 30 sec long
# record (5 * 6 = 30 sec = 30000ms). We add extra seconds for the cases,
# when tracking has started in a middle of a subframe.
    tic
    if (settings.msToProcess < 36000) || (sum([trackResults.status] ~= '-') < 4)
        # Show the error message and exit
        disp('Record is to short or too few satellites tracked. Exiting!');
        navSolutions = [];
        eph          = [];
        svTimeTable  = [];
        activeChnList = [];
        return
    end

    ## Find preamble start positions ==========================================

    [subFrameStart, activeChnList] = findPreambles(trackResults, settings);
    ## Decode ephemerides =====================================================

    for channelNr = activeChnList

        #=== Convert tracking output to navigation bits =======================

        #--- Copy 5 sub-frames long record from tracking output ---------------
```

```
    navBitsSamples = trackResults(channelNr).I_P(subFrameStart(channelNr) -
↪40 : ...
        subFrameStart(channelNr) + (1500 * 20) -1)';

    #--- Group every 20 vales of bits into columns ------------------------
    navBitsSamples = reshape(navBitsSamples, ...
        20, (size(navBitsSamples, 1) / 20));

    #--- Sum all samples in the bits to get the best estimate -------------
    navBits = sum(navBitsSamples);

    #--- Now threshold and make 1 and 0 -----------------------------------
    # The expression (navBits > 0) returns an array with elements set to 1
    # if the condition is met and set to 0 if it is not met.
    navBits = (navBits > 0);

    #--- Convert from decimal to binary -----------------------------------
    # The function ephemeris expects input in binary form. In Matlab it is
    # a string array containing only "0" and "1" characters.
    navBitsBin = dec2bin(navBits);
    #=== Decode ephemerides and TOW of the first sub-frame ================
    [eph(trackResults(channelNr).PRN), TOW] = ...
        ephemeris(navBitsBin(3:1502)', navBitsBin(1),navBitsBin(2));
#     old version of ephemeris.m
#     [eph(trackResults(channelNr).PRN), TOW] = ...
#                              ephemeris(navBitsBin(3:1502)', navBitsBin(1));

    #--- Exclude satellite if it does not have the necessary nav data -----
    # If the satellite accuracy or health is not in reliable values, then
    # this satellite is excluded as well
    if (isempty(eph(trackResults(channelNr).PRN).IODC) || ...
            isempty(eph(trackResults(channelNr).PRN).IODE_sf2) || ...
            isempty(eph(trackResults(channelNr).PRN).IODE_sf3) || ...
            eph(trackResults(channelNr).PRN).accuracy >=3 ||...
            eph(trackResults(channelNr).PRN).health~=0)

        #--- Exclude channel from the list (from further processing) ------
        activeChnList = setdiff(activeChnList, channelNr);
        s=sprintf('PRN #d is excluded from the active channel',...
            trackResults(channelNr).PRN);
        disp(s);
    end
end

## Check if the number of satellites is still above 3 ====================
if (isempty(activeChnList) || (size(activeChnList, 2) < 4))
    # Show error message and exit
```

```
        disp('Too few satellites with ephemeris data for postion calculations.␣
↪Exiting!');
        navSolutions = [];
        eph          = [];
        svTimeTable  = [];
        activeChnList = [];
        return
    end

    ## Initialization ==========================================================
    # Set the satellite elevations array to INF to include all satellites for
    # the first calculation of receiver position. There is no reference point
    # to find the elevation angle as there is no receiver position estimate at
    # this point.
    satElev  = inf(1, settings.numberOfChannels);

    # Save the active channel list. The list contains satellites that are
    # tracked and have the required ephemeris data. In the next step the list
    # will depend on each satellite's elevation angle, which will change over
    # time.
    readyChnList = activeChnList;
    # Establish the transmitting time table
    svTimeTable.time=zeros(1,settings.msToProcess);
    svTimeTable.PRN=[];
    svTimeTable = repmat(svTimeTable, 1, max(activeChnList));

    # Establish the time table based on the TOW and its position
    for channelNr = activeChnList
        svTimeTable(channelNr).PRN=trackResults(channelNr).PRN;
        for i=1:settings.msToProcess
            svTimeTable(channelNr).time(i)=...
                TOW-subFrameStart(channelNr)*0.001+(i-1)*0.001;
        end;
    end
    #
    # svTimeTable=...
    #     transTimeTable(activeChnList,trackResults,subFrameStart,TOW,settings);
    # transmitTime = TOW;

    # Find the last sample number in the tracking results
    lastSample=inf(1,max(readyChnList));

    for channelNr = readyChnList
        lastSample(channelNr) = ...
            trackResults(channelNr).absoluteSample(end);
    end
    # Find the step size for navigation solution
```

```
navStep=settings.samplingFreq/settings.navSolRate;

###########################################################################
##   Do the satellite and receiver position calculations                 #
###########################################################################

## Initialization of current measurement ==================================
for currMeasNr =1:fix((min(lastSample) - ...
        settings.samplingFreq/settings.navSolRate-...
        settings.skipNumberOfSamples) /navStep);
    currMeasNr
# for currMeasNr =1:fix((settings.msToProcess - max(subFrameStart)) / ...
#                                                      (1000/settings.
↪navSolRate))
    # Exclude satellites, that are below elevation mask
    activeChnList = intersect(find(satElev >= settings.elevationMask), ...
        readyChnList);

    # Save list of satellites used for position calculation
    navSolutions.channel.PRN(activeChnList, currMeasNr) = ...
        [trackResults(activeChnList).PRN];

    # These two lines help the skyPlot function. The satellites excluded
    # do to elevation mask will not "jump" to possition (0,0) in the sky
    # plot.
    navSolutions.channel.el(:, currMeasNr) = ...
        NaN(settings.numberOfChannels, 1);
    navSolutions.channel.az(:, currMeasNr) = ...
        NaN(settings.numberOfChannels, 1);

    ## Calculate the current sample number, corresponding satellites ======
    ## tranmitting time and raw receiver time =============================
    sampleNum=currMeasNr*settings.samplingFreq/settings.navSolRate...
        +settings.skipNumberOfSamples;
    transmitTime=...
        findTransTime(sampleNum,activeChnList,svTimeTable,trackResults);
    rxTime=max(transmitTime)+settings.startOffset/1000;

    ## Find pseudoranges␣
↪=======================================================

    [navSolutions.channel.rawP(:, currMeasNr)] = calculatePseudoranges(...
        transmitTime,rxTime,activeChnList,settings);
    # old version of calculateP
#     navSolutions.channel.rawP(:, currMeasNr) = calculatePseudoranges(...
#             trackResults, ...
#             subFrameStart + 1000/settings.navSolRate * (currMeasNr-1), ...
```

```
  #            activeChnList, settings);

      ## Find satellites positions and clocks corrections␣
↪=======================

  #     [satPositions, satClkCorr] = satpos(transmitTime, ...
  #         [trackResults(activeChnList).PRN],eph);
          [satPositions, satClkCorr] =␣
↪satpos(transmitTime(find(transmitTime>0)), ...
          [trackResults(activeChnList).PRN],eph);

      ## Find receiver position␣
↪=================================================

      # 3D receiver position can be found only if signals from more than 3
      # satellites are available
      if length(activeChnList) > 3
  #     if size(activeChnList, 2) > 3

          #=== Calculate receiver position ==================================
          freqforcal=zeros(1,length(activeChnList));
          for ii=1:length(activeChnList)
              freqforcal(ii)=trackResults(1,activeChnList(ii)).
↪carrFreq(currMeasNr*1000/settings.navSolRate);
          end

          [xyzdt, ...
              navSolutions.channel.el(activeChnList, currMeasNr), ...
              navSolutions.channel.az(activeChnList, currMeasNr), ...
              navSolutions.DOP(:, currMeasNr)] = ...
              leastSquarePos(satPositions, ...
              navSolutions.channel.rawP(activeChnList, currMeasNr)' +␣
↪satClkCorr * settings.c, ...
              freqforcal,settings);

  #         [xyzdt, ...
  #             navSolutions.channel.el(activeChnList, currMeasNr), ...
  #             navSolutions.channel.az(activeChnList, currMeasNr), ...
  #             navSolutions.DOP(:, currMeasNr)] = ...
  #             leastSquarePos(satPositions, ...
  #             navSolutions.channel.rawP(activeChnList, currMeasNr)' +␣
↪satClkCorr * settings.c, ...
  #             settings);

          #--- Save results -------------------------------------------------
          navSolutions.X(currMeasNr)  = xyzdt(1);
```

```
            navSolutions.Y(currMeasNr)  = xyzdt(2);
            navSolutions.Z(currMeasNr)  = xyzdt(3);
            navSolutions.dt(currMeasNr) = xyzdt(4);

            navSolutions.Vx(currMeasNr)  = xyzdt(5);
            navSolutions.Vy(currMeasNr)  = xyzdt(6);
            navSolutions.Vz(currMeasNr)  = xyzdt(7);
            navSolutions.ddt(currMeasNr) = xyzdt(8);

            # Update the satellites elevations vector
            satElev = navSolutions.channel.el(:, currMeasNr);

            #=== Correct pseudorange measurements for clocks errors ===========
            navSolutions.channel.correctedP(activeChnList, currMeasNr) = ...
                navSolutions.channel.rawP(activeChnList, currMeasNr) + ...
                satClkCorr' * settings.c - navSolutions.dt(currMeasNr);

            ## Coordinate conversion
↪=================================================

            #=== Convert to geodetic coordinates ==============================
            [navSolutions.latitude(currMeasNr), ...
                navSolutions.longitude(currMeasNr), ...
                navSolutions.height(currMeasNr)] = cart2geo(...
                navSolutions.X(currMeasNr), ...
                navSolutions.Y(currMeasNr), ...
                navSolutions.Z(currMeasNr), ...
                5);

            #=== Convert to UTM coordinate system =============================
            navSolutions.utmZone = findUtmZone(navSolutions.
↪latitude(currMeasNr), ...
                navSolutions.longitude(currMeasNr));

            [navSolutions.E(currMeasNr), ...
                navSolutions.N(currMeasNr), ...
                navSolutions.U(currMeasNr)] = cart2utm(xyzdt(1), xyzdt(2), ...
                xyzdt(3), ...
                navSolutions.utmZone);

            # Compute the corrected receiver time
            navSolutions.rxTime(currMeasNr)=rxTime-navSolutions.dt(currMeasNr)/
↪settings.c;
            # Record the sample number and raw receiver time
            navSolutions.absoluteSample(currMeasNr) =sampleNum;
            navSolutions.rawRxTime(currMeasNr)=rxTime;
```

```
            #DMA add - get the precise time of the first sample and the avg
            #clock rate for the file (skip first 5 samples and should be
            #enough to get over transients)
            if (currMeasNr == fix((min(lastSample) - ...
                    settings.samplingFreq/settings.navSolRate-...
                    settings.skipNumberOfSamples) /navStep))
                dmaTime=polyfit(navSolutions.absoluteSample(5:end)-settings.
↪skipNumberOfSamples,navSolutions.rxTime(5:end),1);
                navSolutions.avgClock = 1/dmaTime(1);
                navSolutions.firstSampleTime = 1 * dmaTime(1) + dmaTime(2);
            end
        else # if size(activeChnList, 2) > 3
            #--- There are not enough satellites to find 3D position ----------
            disp(['   Measurement No. ', num2str(currMeasNr), ...
                ': Not enough information for position solution.']);

            #--- Set the missing solutions to NaN. These results will be
            #excluded automatically in all plots. For DOP it is easier to use
            #zeros. NaN values might need to be excluded from results in some
            #of further processing to obtain correct results.
            navSolutions.X(currMeasNr)           = NaN;
            navSolutions.Y(currMeasNr)           = NaN;
            navSolutions.Z(currMeasNr)           = NaN;
            navSolutions.dt(currMeasNr)          = NaN;
            navSolutions.DOP(:, currMeasNr)      = zeros(5, 1);
            navSolutions.latitude(currMeasNr)    = NaN;
            navSolutions.longitude(currMeasNr)   = NaN;
            navSolutions.height(currMeasNr)      = NaN;
            navSolutions.E(currMeasNr)           = NaN;
            navSolutions.N(currMeasNr)           = NaN;
            navSolutions.U(currMeasNr)           = NaN;
            navSolutions.rawRxTime               = NaN;
            navSolutions.absoluteSample          = NaN;
            navSolutions.rxTime                  = NaN;

            navSolutions.channel.az(activeChnList, currMeasNr) = ...
                NaN(1, length(activeChnList));
            navSolutions.channel.el(activeChnList, currMeasNr) = ...
                NaN(1, length(activeChnList));

            # TODO: Know issue. Satellite positions are not updated if the
            # satellites are excluded do to elevation mask. Therefore rasing
            # satellites will be not included even if they will be above
            # elevation mask at some point. This would be a good place to
            # update positions of the excluded satellites.
```

```
    return navSolutions, eph,svTimeTable,activeChnList
```

```
  File "/tmp/ipykernel_211143/574827455.py", line 38
    if (settings.msToProcess < 36000) || (sum([trackResults.status] ~= '-') < 4
                                       ^
SyntaxError: invalid syntax
```

1 References

- C. Fernández-Prades, J. Arribas, L. Esteve, D. Pubill, P. Closas, An Open Source Galileo E1 Software Receiver, in Proc. of the 6th ESA Workshop on Satellite Navigation Technologies (NAVITEC 2012), ESTEC, Noordwijk, The Netherlands, Dec. 2012. PDF
- J. Arribas, GNSS Array-based Acquisition: Theory and Implementation, PhD Thesis, Universitat Politècnica de Catalunya, Barcelona, Spain, June 2012.
- C. Fernández-Prades, J. Arribas, P. Closas, C. Avilés, and L. Esteve, GNSS-SDR: an open source tool for researchers and developers, in Proc. of the ION GNSS 2011 Conference, Portland, Oregon, Sept. 19-23, 2011.
- C. Fernández-Prades, C. Avilés, L. Esteve, J. Arribas, and P. Closas, Design patterns for GNSS software receivers, in Proc. of the 5th ESA Workshop on Satellite Navigation Technologies (NAVITEC'2010), ESTEC, Noordwijk, The Netherlands, Dec. 2010. DOI

[ ]: