```python
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
import sys
from datetime import datetime, timedelta

"""
#getRinexTime function creates python datetime object from a Rinex line
"""
#
def getRinexTime(line):
    hour = int(line[10:12])
    minute = int(line[13:15])
    second = int(line[16:18])
    return datetime(1, 1, 1, hour, minute, second)

"""
#getNmeaTime function creates python datetime object from nmea line where:

0 = hour
1 = minutes
2 = seconds
tuc = leapseconds to tuc
"""
def getNmeaTime(nmea, tuc):
    time = datetime(1, 1, 1, nmea[0], nmea[1], nmea[2])
    return time - timedelta(hours=0) + timedelta(seconds = tuc);

"""
#rinexSat2nmeaSat function adjusts RINEX satellite number to match NMEA satellite number format
"""
def rinexSat2nmeaSat(line):
    result = []
    for i in range(0, len(line)/3): # Loops in bundles of 3 due to GXX RXX satellite number format
        c = line[i*3] # G for GPS, R for GLONASS and E for GALILEO
        d = line[i*3+1] # Tenths
        u = line[i*3+2] # Units
        if not (c in ['g','G', 'r', 'R','e','E']):
            break
        elif c in ['g','G']:
            result.append( int(d)*10 + int(u) )
        elif c in ['r', 'R']:
            result.append( int(d)*10 + int(u) + 64) #64 is added to match NMEA numbering for GLONAS
        elif c in ['e','E']:
            result.append( int(d)*10 + int(u) + 100) #64 is added to match NMEA numbering for GLONA
    return result

"""
#getRinexEpoch function compares RINEX observed satellites to NMEA's, removing the ones that do not
"""
def getRinexEpoch(line, rsats, nsats):
    result = line # inserts first 29 spaces related to the epoch
    sats = []
    for rsat in rsats: # Checking each RINEX satellite
        for nsat in nsats: # checks each NMEA satellite
            if rsat == nsat:
                sats.append(rsat) # Includes RINEX satellite in the list
    for nsat in nsats:
```

```
        if not (nsat in sats): # Checks if all NMEA satellites were also observed in RINEX list
            pass
#            print "Alert: Satellite ", nsat," in NMEA not found in RINEX.", line
    result = result + '%3d' % len(sats) # Adds total number of satellites that match
    for i, sat in enumerate(sats):
        if (i > 0) and (i % 12 == 0): # Checks to see if line has 12 satellites
            result = result + '\n' + (' ' * 32) # Adds new line with 32 blank spaces
        if sat > 64 and sat < 100:
            Rsat = sat - 64 # Restores RINEX numbering for GLONASS satellite
            result = result + 'R%02d' % Rsat # Concatenates GLONASS satellite
        elif sat > 100:
            Esat = sat - 100 # Restores RINEX numbering for GALILEO satellite
            result = result + 'E%02d' % Esat # Concatenates GALILEO satellite
        else:
            result = result + 'G%02d' % sat # Concatenates GPS satellite
    result = result + '\n'
    return result

"""
#getSatellites function reads RINEX satellites for a given epoch
"""
def getSatellites(line):
    result = []
    for i in range(0, len(line)/3): # Loops in bundles of 3 due to GXX RXX satellite number format
        c = line[i*3] # G for GPS and R for GLONASS
        d = line[i*3+1] # Tenths
        u = line[i*3+2] # Units
        result.append(c+d+u)
    return result

"""
#smoothRange function filters pseudoranges by weighing in doppler measurement to the pseudorange
"""
def smoothRange(measuredRange,lastsmoothedRange,doppler,weight,wlength):
    deltaW = 0.20 # how much weight is added to each epoch
    Wrange = 1 - weight*deltaW # reduces weight for measured range
    if Wrange < 0.01:
        Wrange = 0.01 # minimum weight is 1%

    Wdoppler = 0 + weight*deltaW # increases weight for smoothed range
    if Wdoppler > 0.99:
        Wdoppler = 0.99 # maximum weight is 99%
    result = Wrange*measuredRange + Wdoppler * (lastsmoothedRange-doppler*wlength) #formula for dop
    return result

"""
readNMEA é função para leitura de fonte de dados NMEA e listagem dos satélites
          usados em cada época, cuja solução foi considerada fixada.

No formato NMEA são inseridas diversas linhas (sentenças) com formatos
distintos, dos quais nos interessam os formatos a saber:

Formato da sentença $GPGSA
     Posição:     Domínio:
     0        $GPGSA => Satellite status
     1        A or M (for Automatic or Manual) selection of 2D or 3D fix
     2        3D fix - values include: 1 = no fix
```

```
                                        2 = 2D fix
                                        3 = 3D fix
     3 - 14   PRNs of satellites used for fix (space for 12)
     15       PDOP (dilution of precision)
     16       Horizontal dilution of precision (HDOP)
     17       Vertical dilution of precision (VDOP)
     18       The checksum data, always begins with *

Formato da sentença $GPGGA
     Posição:     Domínio:
     0            $GPGGA => Global Positioning System Fix Data
     1            Fix taken at UTC   (123519 => 12:35:19)
     2            Latitude  ( 4807.038 => 48 deg 07.038')
     3            (N or S)
     4            Longitude (01131.000 => 11 deg 31.000')
     5            (E or W)
     6            Fix quality: 0 = invalid
                               1 = GPS fix (SPS)
                               2 = DGPS fix
                               3 = PPS fix
                              4 = Real Time Kinematic
                              5 = Float RTK
                               6 = estimated (dead reckoning) (2.3 feature)
                              7 = Manual input mode
                              8 = Simulation mode
     7            Number of satellites being tracked
     8            Horizontal dilution of position
     9            Altitude, Meters, above mean sea level
     10           Metric unit
     11           Height of geoid (mean sea level) above WGS84 ellipsoid
     12           Metric unit
     13           Time in seconds since last DGPS update
     14           DGPS station ID number
     15           The checksum data, always begins with *
"""
def readNMEASimple(fileName):
    result = []
    searchFix = True
    searchSat = False
    numberSat = 0
    satellites = []
    hTime = 0
    mTime = 0
    sTime = 0
    with open(fileName, 'r') as nmeaFile:
        for line in nmeaFile:
            line = line.translate(None, '$')
            dataArray = line.split(',')
            if len(dataArray) == 0: # Pula linhas vazias
                continue
            sentenceId = dataArray[0]
            if searchFix and sentenceId == "GPGGA":
                fixQuality = int(dataArray[6])
                if fixQuality == 1:
                    numberSat = int(dataArray[7])
                    hTime = int(dataArray[1][0:2])
                    mTime = int(dataArray[1][2:4])
```

```
                    sTime = int(dataArray[1][4:6])
                    searchSat = True
                    searchFix = False
            if searchSat and sentenceId == "GPGSA":
                fixType = int(dataArray[2])
                if fixType in [2, 3]:
                    satellites = [int(x) for x in dataArray[3:15] if x is not '']
                    if numberSat != len(satellites):
                        print "Incoherent satellite count at the line ", \
                        len(result), "\n"
                    epoch = (hTime, mTime, sTime, numberSat, satellites)
                    result.append( epoch )
                    searchSat = False
                    searchFix = True
        nmeaFile.close()
    return result



def readNMEA(fileName):
    result = []
    searchFix = True
    searchSat = False
    numberSat = 0
    satellites = []
    checkGPSsat = []
    hTime = 0
    mTime = 0
    sTime = 0
    nlines = 0
    with open(fileName, 'r') as nmeaFile:
        for line in nmeaFile:
            line = line.translate(None, '$')
            dataArray = line.split(',')
            if len(dataArray) == 0: # Skip blank lines
                continue
            sentenceId = dataArray[0]
            if searchFix and sentenceId == "GPGGA": # Searching for GNSS fixes
                fixQuality = dataArray[6]
                if fixQuality[0] == '1': # Check if solution exists
                    numberSat = int(dataArray[7]) # Number of satellites in fix
                    hTime = int(dataArray[1][0:2]) # Hour of fix
                    mTime = int(dataArray[1][2:4]) # Minute of fix
                    sTime = int(dataArray[1][4:6]) # Second of fix
                    searchSat = True # Start looking for fix satellites
                    searchFix = False
                    nlines = 0
                    satellites = []
            if searchSat and sentenceId == "GNGSA": # Searching for GPS and GLONASS satellites in j
                fixType = int(dataArray[2]) # Checks if fix exists
                if fixType in [2, 3]:
                    satellites += [int(x) for x in dataArray[3:15] if x is not ''] # Add observed s

            if searchSat and sentenceId == "GAGSA": # Searching for Beidou satellites in fix
                fixType = int(dataArray[2]) # Checks if fix exists
                if fixType in [2, 3]:
                    satellites += [int(x) for x in dataArray[3:15] if x is not ''] # Add observed s
```

```
            if searchSat and sentenceId == "GPRMC":
                epoch = (hTime, mTime, sTime, len(satellites), satellites)
                result.append( epoch )
                searchSat = False
                searchFix = True # Starts looking for next fix
        nmeaFile.close()
    return result

"""
readRINEX is a function that reads RINEX data and lists the observed satellites for each epoch.
"""
def filterRINEX(fileName, nmea):
    result = []
    onHeader = True
    currNmea = None
    nmeaTime = None
    currTime = None
    searchTime = True
    ignoreLines = 0
    extendLines = 0
    numSat = 0
    epoch = 0
    tuc = 0
    sat = []
    out = []
    with open(fileName, 'r') as rinexFile:
        for line in rinexFile:
            if onHeader: # Keeps Header and gets leap seconds
                result.append(line)
                if "LEAP SECONDS" in line:
                    tuc = int(line[0:6]) # Gets Leap seconds
                if "END OF HEADER" in line:
                    onHeader = False
                    currNmea = nmea.pop(0) # List a file and read first line
            else: # Start reading epochs
                if searchTime:
                    if ignoreLines == 0:
                        currTime = getRinexTime(line)
                        nmeaTime = getNmeaTime(currNmea, tuc)
                        numSat = int(line[29:32]) # Number of satellites in epoch
                        extendLines =  (numSat - 1) / 12 # Check if epoch has more than 1 line of c
                        if (currTime < nmeaTime): # Checks if RINEX epoch matches NMEA epoch
                            ignoreLines = numSat + extendLines # Skip satellite observation lines
                        elif (currTime >= nmeaTime): # Checks if RINEX epoch matches NMEA epoch
                            while(currTime > nmeaTime): # Gets new NMEA line If RINEX epoch is ahec
                                if len(nmea) == 0:
                                    break
                                currNmea = nmea.pop(0) # Get new NMEA line
                                nmeaTime = getNmeaTime(currNmea, tuc)
                            if (currTime == nmeaTime): #Epochs match
                                epoch = line[0:29] # Reads epoch date
                                sat = rinexSat2nmeaSat(line[32:]) # List satellites for this epoch
                                searchTime = False # Start reading each satellite observation
                            else:
                                break
                    else:
                        ignoreLines -= 1 # Counts down lines that need to be skipped
```

```
                    continue
                else: # Start reading each satellite observation
                    if extendLines > 0: # Read extra observed satellite line
                        extendLines -= 1
                        sat = sat + rinexSat2nmeaSat(line[32:]) # concatenates second satellite lir
                    else:
                        if len(out) == 0:
                            out.append(getRinexEpoch(epoch, sat, currNmea[4])) # Compares RINEX obs
                        if sat[-numSat] in currNmea[4]: # Test Sat to append in out
                            out.append(line[:32])
                        numSat -= 1
                        if numSat == 0: # Finished reading all satellites
                            searchTime = True
                            while (len(out) > 0):
                                result.append(out.pop(0)) # pop out to result, resetting out
                            if len(nmea) == 0:
                                break
                            currNmea = nmea.pop(0) # Get new NMEA line
        rinexFile.close()
    return result

#with open('filteredRinex_Debug.19o', 'w') as outputFile:
#    for line in result:
#        outputFile.write(line)
#outputFile.close()
"""
filterRiod is a function that reads IBGE RINEX for Riod station and removes satellites
that don't match the NMEA file indicated. It also removes all the other observations
with the exception of C1 and S1 values.
"""


def filterRiod(fileName, nmeaFile):
    result = []
    onHeader = True
    currNmea = None
    nmeaTime = None
    currTime = None
    searchTime = True
    searchS1 = False
    ignoreLines = 0
    extendLines = 0
    numSat = 0
    epoch = 0
    tuc = 0
    sat = []
    out = []
    linecount = 0
    nmea  = readNMEA(nmeaFile)
    ignorecount = 0
    foundcount = 0
    with open(fileName, 'r') as rinexFile:
        for line in rinexFile:
            print line
            linecount += 1
            if onHeader: # Keeps Header and gets leap seconds
                result.append(line)
```

```
        if "LEAP SECONDS" in line:
            tuc = int(line[0:6]) # Gets Leap seconds
            print tuc
        if "END OF HEADER" in line:
            onHeader = False
            print 'End of header'
            currNmea = nmea.pop(0) # List a file and read first line
    else: # Start reading epochs
        if searchTime:
            if ignoreLines == 0:
                currTime = getRinexTime(line) # reads RINEX time
                nmeaTime = getNmeaTime(currNmea, tuc) # reads NMEA time and adds leap secor
                numSat = int(line[29:32]) # Number of satellites in epoch
                extendLines =  (numSat - 1) / 12 # Check if epoch has more than 1 line of c
                if (currTime < nmeaTime): # Checks if RINEX epoch matches NMEA epoch
                    ignorecount += 1
                    ignoreLines = numSat*4 + extendLines # Skip satellite observation lines
                elif (currTime >= nmeaTime): # Checks if RINEX epoch matches NMEA epoch
                    while(currTime > nmeaTime): # Gets new NMEA line If RINEX epoch is ahec
                        if len(nmea) == 0:
                            break
                        currNmea = nmea.pop(0) # Get new NMEA line
                        nmeaTime = getNmeaTime(currNmea, tuc)
                    if (currTime == nmeaTime): #Epochs match
                        foundcount += 1
                        epoch = line[0:29] # Reads epoch date
                        sat = rinexSat2nmeaSat(line[32:]) # List satellites for this epoch
                        searchTime = False # Start reading each satellite observation
                    else:
                        break
            else:
                ignoreLines -= 1 # Counts down lines that need to be skipped
                continue
        else: # Start reading each satellite observation
            if extendLines > 0: # Read extra observed satellite line
                extendLines -= 1
                sat = sat + rinexSat2nmeaSat(line[32:]) # concatenates second satellite lir
            else:
                if ignoreLines == 0 and numSat > 0: # read current line
                    if len(out) == 0: # In case this is the epoch line, out will be empty
                        out.append(getRinexEpoch(epoch, sat, currNmea[4])) # Creates epoch
                    if searchS1 == False and sat[-numSat] in currNmea[4]: # Test if current
                        out.append(line[:14]+26*' ') # adds L1 for current satellite and 26
                        ignoreLines = 3 # skips the 3 following lines
                        searchS1 = True # search for S1
                    else:
                        ignoreLines = 3 # skips the 3 following lines because current RINEX
                elif ignoreLines > 0: # satellite was found and we have the measurement!
                    if searchS1 and ignoreLines == 2: # if satellite was found, we need to
                        out.append(line[40:46]+'\n') # adds S1 if current RINEX satellite t
                        searchS1 = False # jump the rest of the lines
                    ignoreLines -=1 # one less line to read
                    continue # next line in list
                numSat -= 1 # count down one less satellite after it is read up there
                if numSat <= 0: # Finished reading all satellites
                    if searchS1:
                        if ignoreLines != 0:
```

```
                                    print line
                                    print linecount
#                                    out.append(line+'\n')
                                    ignoreLines -=1
                                    continue
#                                else:
#                                    print line
#                                    ignoreLines -=2
#                                    out.append(line+'\n') # adds S1 if current RINEX satellite is
#                                    searchS1 = False # jump the rest of the lines
#                             print linecount
                            searchTime = True
                            while (len(out) > 0):
                                result.append(out.pop(0)) # pop out to result, resetting out
                            if len(nmea) == 0:
                                break
                            currNmea = nmea.pop(0) # Get new NMEA line
        output = fileName[:-4]+'_filtered'+fileName[-4:]
        with open(output, 'w') as outputFile:
            for line in result:
                outputFile.write(line)
            outputFile.close()
        rinexFile.close()
    #return result
    print ignorecount
    print foundcount

"""
smoothRINEX is a function that reads a RINEX v2.11 from GEO++ RINEX Logger
and edits the pseudoranges through doppler smoothing technique.

satdict format:
"G13": [weight,pseudorange,doppler]
satdict[sat][0] = Grabs the weight value on the "sat" key
satdict[sat][1] = Grabs the pseudorange value on the "sat" key
satdict[sat][2] = Grabs the dopplerunder value on the "sat" key
"""

def smoothRINEXavgD(fileName):
    out = []
    onHeader = True
    searchTime = True
    extendLines = 0
    numSat = 0
    tuc = 0
    satdict = {} # Creates satellite dictionary for Sat number, weight and pseudorange
    sats = [] # Creates list to receive satellite for each epoch
    c = 299792458.0 # speed of light in m/s
    GPSL1length = c/1575420000
    GLONASSf0L1 = 1602000000 # initial value used to calculate GLONASS wavelength
    GLONASSdeltafL1 = 562500 # delta to be multiplied the RF channel
    GLONASSf = {"R01": 1, "R02": -4, "R03": 5, "R04": 6, "R05": 1, "R06": -4, "R07": 5, "R08": 6} #
    GLONASSf.update({"R09": -2, "R10": -7, "R11": 0, "R12": -1, "R13": -2, "R14": -7, "R15": 0, "R1
    GLONASSf.update({"R17": 4, "R18": -3, "R19": 3, "R20": 2, "R21": 4, "R22": -3, "R23": 3, "R24":
    epochnum = 0
    with open(fileName, 'r') as rinexFile:
        for line in rinexFile:
```

```
            if onHeader: # Keeps Header and gets leap seconds
                out.append(line)
                if "LEAP SECONDS" in line:
                    tuc = int(line[0:6]) # Gets Leap seconds
                if "END OF HEADER" in line:
                    onHeader = False
            else: # Start reading epochs
                if searchTime:
                    out.append(line)
                    numSat = int(line[29:32]) # Number of satellites in epoch
                    extendLines =  (numSat - 1) / 12 # Check if epoch has more than 1 line of obser
                    sats = getSatellites(line[32:]) # List satellites with timestamp for this epoch
                    searchTime = False
                elif extendLines > 0:
                    extendLines -= 1
                    sats = sats + getSatellites(line[32:]) # concatenates second satellite line to

                else: # Start reading each satellite observation
                    sat = sats[len(sats) - numSat]
                    pseudorange = float(line[0:14]) # Reads measured pseudorange
                    doppler = float(line[52:62]) # reads measured doppler
                    if epochnum == 0:
                        satdict[sat] = [0,pseudorange,doppler] # Set initial weight to zero and int
                        newRangeline = '  ' + str(format(pseudorange,'.3f')) + line[14:62] + '\n'
                    else:
                        if sat in satdict.keys():
                            satdict[sat][0] += 1 #adds one to the satellite weight
                            avgDoppler = (doppler + satdict[sat][2]) / 2.0
                            if sat in GLONASSf.keys():
                                wlength = c/(GLONASSf0L1 + GLONASSf[sat]*GLONASSdeltafL1)
                                newRange = smoothRange(pseudorange,satdict[sat][1],avgDoppler,satdi
                                satdict[sat][1] = newRange
                                satdict[sat][2] = doppler
                                newRangeline = '  ' + str(format(newRange,'.3f')) + line[14:62] + '
                            else:
                                wlength = GPSL1length
                                newRange = smoothRange(pseudorange,satdict[sat][1],avgDoppler,satdi
                                satdict[sat][1] = newRange
                                satdict[sat][2] = doppler
                                newRangeline = '  ' + str(format(newRange,'.3f')) + line[14:62] + '

                        else:
                            satdict[sat] = [0,pseudorange,doppler] # create an entry for new satell
                            newRangeline = '  ' + str(format(pseudorange,'.3f')) + line[14:62] + '\
                    remove = []
                    for key in satdict.keys():
                        if not (key in sats):
                            remove.append(key)
                    for key in remove:
                        del satdict[key]
                    out.append(newRangeline)
                    numSat -= 1
                    if numSat == 0: # END OF EPOCH: Finished reading all satellites
#                        print 'end of epoch ',epochnum
                        searchTime = True # Read new epoch
                        epochnum +=1
        output = fileName[:-4]+'doppleravg_Smoothed'+fileName[-4:]
```

```
        with open(output, 'w') as outputFile:
            for line in out:
                outputFile.write(line)
            outputFile.close()


def smoothRINEX(fileName):
    out = []
    onHeader = True
    searchTime = True
    extendLines = 0
    numSat = 0
    tuc = 0
    satdict = {} # Creates satellite dictionary for Sat number, weight and pseudorange
    sats = [] # Creates list to receive satellite for each epoch
    c = 299792458.0 # speed of light in m/s
    GPSL1length = c/1575420000
    GLONASSf0L1 = 1602000000 # initial value used to calculate GLONASS wavelength
    GLONASSdeltafL1 = 562500 # delta to be multiplied the RF channel
    GLONASSf = {"R01": 1, "R02": -4, "R03": 5, "R04": 6, "R05": 1, "R06": -4, "R07": 5, "R08": 6} #
    GLONASSf.update({"R09": -2, "R10": -7, "R11": 0, "R12": -1, "R13": -2, "R14": -7, "R15": 0, "R1
    GLONASSf.update({"R17": 4, "R18": -3, "R19": 3, "R20": 2, "R21": 4, "R22": -3, "R23": 3, "R24":
    epochnum = 0
    with open(fileName, 'r') as rinexFile:
        for line in rinexFile:
            if onHeader: # Keeps Header and gets leap seconds
                out.append(line)
                if "LEAP SECONDS" in line:
                    tuc = int(line[0:6]) # Gets Leap seconds
                if "END OF HEADER" in line:
                    onHeader = False
            else: # Start reading epochs
                if searchTime:
                    out.append(line)
                    numSat = int(line[29:32]) # Number of satellites in epoch
                    extendLines =  (numSat - 1) / 12 # Check if epoch has more than 1 line of obser
                    sats = getSatellites(line[32:]) # List satellites with timestamp for this epoch
                    searchTime = False
                elif extendLines > 0:
                    extendLines -= 1
                    sats = sats + getSatellites(line[32:]) # concatenates second satellite line to

                else: # Start reading each satellite observation
                    sat = sats[len(sats) - numSat]
                    pseudorange = float(line[0:14]) # Reads measured pseudorange
                    doppler = float(line[52:62]) # reads measured doppler
                    if epochnum == 0:
                        satdict[sat] = [0,pseudorange,doppler] # Set initial weight to zero and int
                        newRangeline = '   ' + str(format(pseudorange,'.3f')) + line[14:62] + '\n'
                    else:
                        if sat in satdict.keys():
                            satdict[sat][0] += 1 #adds one to the satellite weight
                            avgDoppler = (doppler + satdict[sat][2]) / 2.0
                            if sat in GLONASSf.keys():
                                wlength = c/(GLONASSf0L1 + GLONASSf[sat]*GLONASSdeltafL1)
                                newRange = smoothRange(pseudorange,satdict[sat][1],doppler,satdict[
                                satdict[sat][1] = newRange
```

```
                                satdict[sat][2] = doppler
                                newRangeline = '  ' + str(format(newRange,'.3f')) + line[14:62] + '
                            else:
                                wlength = GPSL1length
                                newRange = smoothRange(pseudorange,satdict[sat][1],doppler,satdict[
                                satdict[sat][1] = newRange
                                satdict[sat][2] = doppler
                                newRangeline = '  ' + str(format(newRange,'.3f')) + line[14:62] + '

                        else:
                            satdict[sat] = [0,pseudorange,doppler] # create an entry for new satell
                            newRangeline = '  ' + str(format(pseudorange,'.3f')) + line[14:62] + '\
                    remove = []
                    for key in satdict.keys():
                        if not (key in sats):
                            remove.append(key)
                    for key in remove:
                        del satdict[key]
                    out.append(newRangeline)
                    numSat -= 1
                    if numSat == 0: # END OF EPOCH: Finished reading all satellites
#                        print 'end of epoch ',epochnum
                        searchTime = True # Read new epoch
                        epochnum +=1
        output = fileName[:-4]+'_Smoothed5sec'+fileName[-4:]
        with open(output, 'w') as outputFile:
            for line in out:
                outputFile.write(line)
            outputFile.close()

"""
RINEX Filtering
smoothRINEXavgD('20190115_Sta91500.19o')
smoothRINEXavgD('20190211_Sta91500.19o')
smoothRINEXavgD('20190212_Sta91500.19o')

smoothRINEX('20190115_Sta91500.19o')
smoothRINEX('20190211_Sta91500.19o')
smoothRINEX('20190212_Sta91500.19o')
"""
```