#!/usr/bin/env python

import sys import os import argparse from termcolor import colored import logging import json from datetime import datetime from pathlib import Path import pandas as pd from shutil import copyfile from typing import Tuple import pickle from nested_lookup import nested_lookup

from gfzrnx import gfzrnx_constants as gfzc from ampyutils import gnss_cmd_opts as gco

from ampyutils import am_config as amc from ampyutils import amutils from tle import tle_visibility, tleobs_plot from ltx import ltx_rnxobs_reporting

author = “amuls”

def treatCmdOpts(argv): """ Treats the command line options

```
:param argv: the options
:type argv: list of string
"""
baseName = os.path.basename(__file__)
amc.cBaseName = colored(baseName, 'yellow')

helpTxt = amc.cBaseName + ' analyses observation tabular file for selected GNSSs

# create the parser for command line arguments
parser = argparse.ArgumentParser(description=helpTxt)

parser.add_argument('--obstab', help='observation tabular file', type=str, requi

parser.add_argument('--prns', help='list of PRNs to examine (default {:s} (if PR

parser.add_argument('--freqs', help='select frequencies to use (out of {freqs:s}

parser.add_argument('--obstypes', help='select observation types(s) to use (out 

parser.add_argument('--snr_th', help='threshold for detecting variation in SNR l

parser.add_argument('--cutoff', help='cutoff angle in degrees (default {mask:s})

parser.add_argument('--plot', help='displays interactive plots (default False)',

parser.add_argument('--logging', help='specify logging level console/file (two o
# drop argv[0]
args = parser.parse_args(argv[1:])

# return arguments
return args.obstab, args.freqs, args.prns, args.obstypes, args.snr_th, args.cuto
```

def check_arguments(logger: logging.Logger = None): """ check arhuments and change working directory """ cFuncName = colored(os.path.basename(**file**), “yellow”) + ’ - ’ + colored(sys._getframe().f_code.co_name, “green”)

```
# check & change working dir
dTab['dir'] = os.path.dirname(Path(dTab['cli']['obstabf']).resolve())
dTab['obstabf'] = os.path.basename(dTab['cli']['obstabf'])

if not amutils.changeDir(dTab['dir']):
    if logger is not None:
        logger.error('{func:s}: changing to directory {dir:s} failed'.format(dir
    sys.exit(amc.E_DIR_NOT_EXIST)

# check accessibilty of observation statistics file
if not amutils.file_exists(fname=dTab['obstabf'], logger=logger):
    if logger is not None:
        logger.error('{func:s}: observation file {file:s} not accessible'.format
    sys.exit(amc.E_FILE_NOT_EXIST)

# create dir for storing the latex sections
dTab['ltx']['path'] = os.path.join(dTab['dir'], 'ltx')
if not amutils.mkdir_p(dTab['ltx']['path']):
    if logger is not None:
        logger.error('{func:s}: cannot create directory {dir:s} failed'.format(d
    sys.exit(amc.E_FAILURE)

# extract YY and DOY from filename
dTab['time']['YYYY'] = int(dTab['obstabf'][12:16])
dTab['time']['DOY'] = int(dTab['obstabf'][16:19])
# converting to date
```

```
dTab['time']['date'] = datetime.strptime('{year:04d}-{doy:03d}'.format(year=dTab
%j")

# check whether selected freq is available
for clifreq in dTab['cli']['freqs']:
    if clifreq not in dTab['info']['freqs']:
        if logger is not None:
            logger.error('{func:s}: selected frequency {clifreq:s} is not availa
        sys.exit(amc.E_NOAVAIL_FREQ)

# if 'E00' or 'G00' is selected, the list of PRNs to examine is all PRNs for tha
use_all_prns = False
for gnss in gfzc.lst_GNSSs:
    for prn in dTab['cli']['lst_prns']:
        if prn == gfzc.dict_GNSS_PRNs[gnss][0]:
            use_all_prns = True
            dTab['lst_prns'] = gfzc.dict_GNSS_PRNs[gnss]

if not use_all_prns:
    dTab['lst_prns'] = dTab['cli']['lst_prns']
```

def read_obstab(obstabf: str, lst_PRNs: list, dCli: dict, logger: logging.Logger = None) -> Tuple[list, list, list, pd.DataFrame]: """ read_obstab reads the SNR for the selected frequencies into a dataframe """ cFuncName = colored(os.path.basename(**file**), “yellow”) + ’ - ’ + colored(sys._getframe().f_code.co_name, “green”)

```
# determine what the columnheaders will be
hdr_count = -1
hdr_columns = []
with open(obstabf) as fin:
    for line in fin:
        # print(line.strip())
        hdr_count += 1
        if line.strip().startswith('OBS'):
            break
        else:
            if line.strip() != '#HD,G,DATE,TIME,PRN':
                hdr_line = line.strip()
```

```
# split up on comma into a list
hdr_columns = hdr_line.split(',')
# print('hdr_columns = {!s}'.format(hdr_columns))
# print('hdr_columns[2:4] = {!s}'.format(hdr_columns[2:4]))
# print('hdr_count = {!s}'.format(hdr_count))

# keep the header columns selected by --freqs and --obstypes options
obstypes = hdr_columns[2:5]
obsfreqs = []
for freq in dTab['cli']['freqs']:
    for obst in dCli['obs_types']:
        obsfreq = '{obst:s}{freq:s}'.format(obst=obst, freq=freq)
        obsfreqs.append([obstid for obstid in hdr_columns[2:] if obstid.startswi
obstypes += obsfreqs

# created the possible navigation signals we have
nav_signals = list(set([obsfreq[1:] for obsfreq in obsfreqs]))
print(nav_signals)

logger.info('{func:s}: loading from {tab:s}: {cols:s}'.format(tab=obstabf, cols=

dfTmp = pd.read_csv(obstabf, delimiter=',', skiprows=hdr_count, names=hdr_column

# check whether the selected PRNs are in the dataframe, else remove this PRN fro
# print('lst_PRNs = {}'.format(lst_PRNs))
lst_ObsPRNs = sorted(dfTmp.PRN.unique())
print('lst_obsPRNs = {}'.format(lst_ObsPRNs))
print("dfTmp[dfTmp['PRN'] == 'E07'] = \n{}".format(dfTmp[dfTmp['PRN'] == 'E07'])

lst_CommonPRNS = [prn for prn in lst_ObsPRNs if prn in lst_PRNs]
# print('lst_CommonPRNS = {}'.format(lst_CommonPRNS))

if len(lst_CommonPRNS) == 0:
    logger.error('{func:s}: selected list of PRNs ({lstprns:s}) not observed. pr
    sys.exit(amc.E_PRN_NOT_IN_DATA)
else:
    logger.info('{func:s}: following PRNs examined: {prns:s}'.format(prns=', '.j
# apply mask to dataframe to select PRNs in the list
# print('lst_PRNs = {}'.format(lst_PRNs))
# print("dfTmp['PRN'].isin(lst_PRNs) = {}".format(dfTmp['PRN'].isin(lst_PRNs)))
dfTmp = dfTmp[dfTmp['PRN'].isin(lst_CommonPRNS)]

# # XXX TEST BEGIN for testing remove some lines for SV E02
# indices = dfTmp[dfTmp['PRN'] == 'E02'].index
# print('dropping indices = {!s}'.format(indices[5:120]))
# dfTmp.drop(indices[5:120], inplace=True)
# # END XXX TEST

if logger is not None:
    amutils.logHeadTailDataFrame(df=dfTmp, dfName='dfTmp', callerName=cFuncName,

return lst_CommonPRNS, nav_signals, obsfreqs, dfTmp
```

def analyse_obsprn(marker: str, obstabf: str, navsig_name: str, dTime: dict, dfPrnNavSig: pd.DataFrame, dfPrnTle: pd.DataFrame, prn: str, navsig_obst_lst: dict, snrth: float, interval: int, show_plot: bool = False, logger: logging.Logger = None) -> Tuple[list, list, dict]: """ analyse_obsprn analyses the observations for the given PRN and determines a loss in SNR if asked. """ cFuncName = colored(os.path.basename(**file**), “yellow”) + ’ - ’ + colored(sys._getframe().f_code.co_name, “green”)

```
plots = {}
posidx_time_gaps = []
posidx_snr_posjumps = {}
posidx_snr_negjumps = {}

# get a list of time jumps for this navigation signal
# calculate the time difference between successive entries
dfPrnNavSig.insert(loc=dfPrnNavSig.columns.get_loc('DATE_TIME') + 1,
                   column='dt',
                   value=(dfPrnNavSig['DATE_TIME'] - dfPrnNavSig['DATE_TIME'].sh
amutils.logHeadTailDataFrame(df=dfPrnNavSig, dfName='dfPrnNavSig', callerName=cF

# check whether data is available for this PRN, if no data available, just retur
if dfPrnNavSig.shape[0] == 0:
    time_gaps = None
    time_reaqs = None
    # posidx_snr_posjumps = None
    # posidx_snr_negjumps = None
    plots = None

    return posidx_time_gaps, posidx_snr_posjumps, posidx_snr_negjumps, plots

idx_time_gaps = dfPrnNavSig.index[dfPrnNavSig.dt != interval].tolist()
# convert to positional indices
posidx_time_gaps = [dfPrnNavSig.index.get_loc(gap) for gap in idx_time_gaps]
# insert the first and last positional indices to get start and end time
if posidx_time_gaps[0] != 0:
    posidx_time_gaps.insert(0, 0)
if posidx_time_gaps[-1] != dfPrnNavSig.shape[0] - 1:
    posidx_time_gaps.append(dfPrnNavSig.shape[0] - 1)

print('{}: posidx_time_gaps = \n{}'.format(prn, posidx_time_gaps))
print('{}: posidx_time_gaps + 1 = \n{}'.format(prn, [x - 1 for x in posidx_time_
1]]))

time_reaqs = dfPrnNavSig.iloc[posidx_time_gaps]['DATE_TIME']
time_gaps = dfPrnNavSig.iloc[[x - 1 for x in posidx_time_gaps[1:-
1]]]['DATE_TIME']
print('{}: time_reaqs = \n{}'.format(prn, time_reaqs))
print('{}: time_gaps = \n{}'.format(prn, time_gaps))

# examine the requested observations for this navigation signal
print('{}: navsig_obst_lst = {}'.format(prn, navsig_obst_lst))
for navsig_obs in navsig_obst_lst:
    print('{}: navsig_obs = {}'.format(prn, navsig_obs))
    # select only the elements for this prn
    dfPrnNSObs = dfPrnNavSig[['DATE_TIME', 'dt', 'PRN', navsig_obs]].dropna()

    # add column which is difference between current and previous obst
    dfPrnNSObs.insert(loc=dfPrnNSObs.columns.get_loc(navsig_obs) + 1,
                      column='d{nso:s}'.format(nso=navsig_obs),
                      value=(dfPrnNSObs[navsig_obs] - dfPrnNSObs[navsig_obs].shi
    print('dfPrnNSObs = {}'.format(dfPrnNSObs))

    # find the exponential moving average
    # dfPrnNSObs['EMA05'] = dfPrnNSObs[navsig_obs].ewm(halflife='5 seconds', adj
    # dfPrnNSObs['WMA05'] = dfPrnNSObs[navsig_obs].rolling(5, min_periods=1).mea

    # for idx_gap in idx_time_gaps[1:]:
    #     pos_idx_gap = dfPrnNSObs.index.get_loc(idx_gap)
    #     print(dfPrnNSObs.iloc[pos_idx_gap - 2 * span:pos_idx_gap + int(span /

    # find the SNR differences that are higher than snrth (SNR threshold)
    if navsig_obs[0] == 'S':
        idx_snr_posjumps = dfPrnNSObs.index[dfPrnNSObs['d{nso:s}'.format(nso=nav
        # convert to poisionla indices
        posidx_snr_posjumps[navsig_obs] = [dfPrnNSObs.index.get_loc(jump) for ju
        print('posidx_snr_posjumps[navsig_obs] = {} #{}'.format(posidx_snr_posju

        idx_snr_negjumps = dfPrnNSObs.index[dfPrnNSObs['d{nso:s}'.format(nso=nav
snrth].tolist()
        # convert to poisionla indices
        posidx_snr_negjumps[navsig_obs] = [dfPrnNSObs.index.get_loc(jump) for ju
        print('posidx_snr_negjumps[navsig_obs] = {} #{}'.format(posidx_snr_negju
    else:
        posidx_snr_posjumps[navsig_obs] = None
        posidx_snr_negjumps[navsig_obs] = None

    # info to user
    if logger is not None:
        amutils.logHeadTailDataFrame(df=dfPrnNSObs, dfName='dfPrnNSObs', callerN

    # plot for each PRN and obstfreq
    plots[navsig_obs] = tleobs_plot.plot_prn_navsig_obs(marker=marker,
                                                        dTime=dTime,
                                                        obsf=obstabf,
                                                        prn=prn,
                                                        dfPrnObst=dfPrnNSObs,
                                                        dfTlePrn=dfPrnTle,
                                                        obst=navsig_obs,
                                                                  posidx_gaps=posidx_time_
                                                                  snrth=snrth,
                                                                  show_plot=show_plot,
                                                                  logger=logger)

print('plots = {}'.format(plots))
print('posidx_time_gaps = {}'.format(posidx_time_gaps))
print('posidx_snr_posjumps[navsig_obs] = {}'.format(posidx_snr_posjumps[navsig_o
print('posidx_snr_negjumps[navsig_obs] = {}'.format(posidx_snr_negjumps[navsig_o

# return posidx_time_gaps, posidx_snr_posjumps[navsig_obs], posidx_snr_negjumps[
return time_gaps.tolist(), time_reaqs.tolist()[1:-1], plots
```

def pnt_available(dfPrnEvol: pd.DataFrame, logger: logging.Logger = None) -> dict: """ pnt_available deterimes the data_times corresponding to loss / reacquisition of PNT """ cFuncName = colored(os.path.basename(**file**), “yellow”) + ’ - ’ + colored(sys._getframe().f_code.co_name, “green”)

```
# # TEST CHECK IF START WITH NO PNT
# dfPrnEvol = dfPrnEvol.iloc[11:]
# dfPrnEvol.reset_index(inplace=True)
# # END TEST CHECK IF START WITH NO PNT

dPNT = {}
dPNT['loss'] = []
dPNT['reacq'] = []
dPNT['gap'] = []

print(dfPrnEvol)
print(dfPrnEvol.shape)

# check at first period whether we have PNT or not available
if dfPrnEvol.iloc[0]['PRNcnt'] >= 4:
    start_PNT = True
else:
    start_PNT = False

# indices used to find loss / reacquistion of PNT
idx_PNTloss = idx_PNTreacq = 0
```

```
while True:
    try:
        if start_PNT:  # have PNT at start
            # find index of first occurence that less than 4 satellites are trac
            idx_PNTloss = dfPrnEvol.iloc[idx_PNTreacq:].loc[dfPrnEvol.PRNcnt < 4
            # find when we have reacquired PNT
            idx_PNTreacq = dfPrnEvol.iloc[idx_PNTloss:].loc[dfPrnEvol.PRNcnt >=

            # get corresponding timings and PNT time gap
            dPNT['loss'].append(dfPrnEvol.iloc[idx_PNTloss - 1]['DATE_TIME'])
            dPNT['reacq'].append(dfPrnEvol.iloc[idx_PNTreacq]['DATE_TIME'])
        else:  # did not have PNT at start
            # lost PNT at start
            dPNT['loss'].append(dfPrnEvol.iloc[0]['DATE_TIME'])

            # find when we have reacquired PNT
            idx_PNTreacq = dfPrnEvol.iloc[idx_PNTloss:].loc[dfPrnEvol.PRNcnt >=
            dPNT['reacq'].append(dfPrnEvol.iloc[idx_PNTreacq]['DATE_TIME'])

            # we now have a new start where we have PNT
            start_PNT = True

        dPNT['gap'].append((dPNT['reacq'][-1] - dPNT['loss'][-1]).total_seconds(
        if logger is not None:
            logger.info('{func:s}: PNT loss @ {loss:s} => {reacq:s} for {gap:.1f
1].strftime('%H:%M:%S'),

1].strftime('%H:%M:%S'),

1],

    except IndexError:
        break

return dPNT
```

def main_obstab_analyse(argv): """ main_obstab_analyse analyses the created OBSTAB files and compares with TLE data. """

```
cFuncName = colored(os.path.basename(__file__), 'yellow') + ' -
' + colored(sys._getframe().f_code.co_name, 'green')

global dTab
dTab = {}
dTab['cli'] = {}
dTab['time'] = {}
dTab['ltx'] = {}
dTab['plots'] = {}
dTab['lock'] = {}
dTab['info'] = {}

dTab['cli']['obstabf'], dTab['cli']['freqs'], dTab['cli']['lst_prns'], dTab['cli

# detect used GNSS from the obstabf filename
dTab['info']['gnss'] = os.path.splitext(os.path.basename(dTab['cli']['obstabf'])
1]
dTab['info']['gnss_name'] = gfzc.dict_GNSSs[dTab['info']['gnss']]

# create logging for better debugging
logger, log_name = amc.createLoggers(baseName=os.path.basename(__file__), logLev

# read the observation header info from the Pickle file
dTab['obshdr'] = '{obsf:s}.obshdr'.format(obsf=os.path.splitext(dTab['cli']['obs
2])
try:
    with open(dTab['obshdr'], 'rb') as handle:
        dTab['hdr'] = pickle.load(handle)
    dTab['marker'] = dTab['hdr']['file']['site']
    dTab['time']['interval'] = float(dTab['hdr']['file']['interval'])
    dTab['info']['freqs'] = dTab['hdr']['file']['sysfrq'][dTab['info']['gnss']]
except IOError as e:
    logger.error('{func:s}: error {err!s} reading header file {hdrf:s}'.format(h
    sys.exit(amc.E_FILE_NOT_EXIST)

# verify input
```

```
check_arguments(logger=logger)

logger.info('{func:s}: Imported header information from {hdrf:s}\n{json!s}'.form

# determine start and end times of observation
dTab['time']['start'] = datetime.strptime(dTab['hdr']['data']['epoch']['first'].
dTab['time']['end'] = datetime.strptime(dTab['hdr']['data']['epoch']['last'].spl

logger.info('{func:s}: Project information =\n{json!s}'.format(func=cFuncName, j

# read obstab into a dataframe and select the SNR for the selected frequencies
dTab['lst_CmnPRNs'], dTab['nav_signals'], dTab['obsfreqs'], dfObsTab = read_obst




# get the observation time spans based on TLE values
dfTLE = tle_visibility.PRNs_visibility(prn_lst=dfObsTab.PRN.unique(),
                                       DTG_start=dTab['time']['start'],
                                       DTG_end=dTab['time']['end'],
                                       interval=dTab['time']['interval'],
                                       cutoff=dTab['cli']['mask'],
                                       logger=logger)

amutils.logHeadTailDataFrame(df=dfObsTab, dfName='dfObsTab', callerName=cFuncNam
amutils.logHeadTailDataFrame(df=dfTLE, dfName='dfTLE', callerName=cFuncName, log

logger.info('{func:s}: Project information =\n{json!s}'.format(func=cFuncName, j

sec_obstab = ltx_rnxobs_reporting.obstab_tleobs_ssec(obstabf=dTab['obstabf'],
                                                     lst_PRNs=dTab['lst_CmnPRNs'
                                                     lst_NavSignals=dTab['nav_si
                                                     lst_ObsFreqs=dTab['obsfreqs
                                                     dfTle=dfTLE)
dTab['ltx']['obstab'] = '{marker:s}_03_{gnss:s}_obs_tab'.format(marker=dTab['obs

lst_navsig_obst = {}
```

```
for nav_signal in dTab['nav_signals']:
    # dict for keeping the obtained info
    dTab['plots'][nav_signal] = {}
    dTab['lock'][nav_signal] = {}

    nav_signal_name = '{gnss:s}{navs:s}'.format(gnss=dTab['info']['gnss'], navs=
    logger.info('{func:s}: working on navigation signal {navs:s}'.format(navs=co

    # keep the observables for this navigatoion signal
    col_navsig = [column for column in dfObsTab.columns.tolist()[:2]]
    col_navsig += [column for column in dfObsTab.columns.tolist()[2:] if column.
    print('col_navsig = {}'.format(col_navsig))
    dfNavSig = dfObsTab[col_navsig].dropna()

    amutils.logHeadTailDataFrame(df=dfNavSig, dfName='dfNavSig', callerName=cFun

    # create dataframe for this navsig with count of PRN at each date_time
    dfNavSigPRNCount = dfNavSig.pivot_table(index=['DATE_TIME'], aggfunc='size')
    dfNavSigPRNCount.reset_index(inplace=True)
    dfNavSigPRNCount.rename(columns={0: 'PRNcnt'}, inplace=True)
    # add difference in PRNcnt and difference in DATE_TIME
    dfNavSigPRNCount["dPRNcnt"] = dfNavSigPRNCount["PRNcnt"].diff(1)
    dfNavSigPRNCount["dt"] = dfNavSigPRNCount["DATE_TIME"].diff(dTab['time']['in
    amutils.logHeadTailDataFrame(df=dfNavSigPRNCount, dfName='dfNavSigPRNCount',

    # create a dataframe containing the times where there is a change in PRNcnt
    idx_list = dfNavSigPRNCount.index[(dfNavSigPRNCount['dPRNcnt'] != 0) | (dfNa
    idx_prev_list = [x - 1 for x in idx_list if x > 0]
    idx_merged = idx_list + idx_prev_list
    idx_merged.sort()
    print('idx_list = {}'.format(idx_list))
    print('idx_prev_list = {}'.format(idx_prev_list))
    print('idx_merged = {}'.format(idx_merged))
    dfPRNEvol = dfNavSigPRNCount.iloc[idx_merged]
    dfPRNEvol.reset_index(drop=True, inplace=True)
    print('dfPRNEvol = \n{}'.format(dfPRNEvol))

    # create lists with DateTimes of loss / reacquisition of PNT
```

```
dTab['PNT'] = pnt_available(dfPrnEvol=dfPRNEvol, logger=logger)
print("dTab['PNT'] = {}".format(dTab['PNT']))

amutils.logHeadTailDataFrame(df=dfPRNEvol, dfName='dfPRNEvol', callerName=cF

# create plot with all selected PRNs vs the TLE part per navigation signal
dTab['plots'][nav_signal]['tle-obs'] = tleobs_plot.obstle_plot_arcs_prns(mar
                                                                         obs
                                                                         dTi
                                                                         nav
                                                                         lst
                                                                         dfN
                                                                         dfT
                                                                         log
                                                                         sho

# perform analysis of the observations done per PRN and per navigation signa
lst_navsig_obst[nav_signal] = [navsig_obs for navsig_obs in dTab['obsfreqs']

# create plot with all selected PRNs for the selected obstypes
dTab['plots'][nav_signal]['obst'] = tleobs_plot.obstle_plot_gnss_obst(marker
                                                                      obsf=d
                                                                      dTime=
                                                                      navsig
                                                                      lst_PR
                                                                      dfNavS
                                                                      dfNavS
                                                                      navsig
                                                                      dfTle=
                                                                      logger
                                                                      show_p

for prn in dTab['lst_CmnPRNs']:

    print('\nPRN = {} {}'.format(prn, nav_signal_name))
    # select the TLE row for this PRN
    dfTLEPrn = dfTLE.loc[prn]
```

```
        # select the TLE row for this PRN
        dfNavSigPRN = dfNavSig[dfNavSig['PRN'] == prn].dropna()
        print('dfNavSigPRN = \n{}'.format(dfNavSigPRN))

        # posidx_time_gaps, posidx_snr_posjumps[navsig_obs], posidx_snr_negjumps
        prn_loss, prn_reacq, prn_plots = analyse_obsprn(marker=dTab['marker'],
                                                        obstabf=dTab['obstabf'],
                                                        navsig_name=nav_signal_n
                                                        dTime=dTab['time'],
                                                        prn=prn,
                                                        dfPrnTle=dfTLEPrn,
                                                        dfPrnNavSig=dfNavSigPRN,
                                                        navsig_obst_lst=lst_navs
                                                        snrth=dTab['cli']['snrth
                                                        interval=dTab['time']['i
                                                        show_plot=show_plot,
                                                        logger=logger)

        dTab['plots'][nav_signal][prn] = prn_plots
        dTab['lock'][nav_signal][prn] = {}

        dTab['lock'][nav_signal][prn]['loss'] = prn_loss
        dTab['lock'][nav_signal][prn]['reacq'] = prn_reacq

print("dTab['plots'][nav_signal] = {}\n-------------------------
\n".format(dTab['plots'][nav_signal]))

logger.info('{func:s}: Project information =\n{json!s}'.format(func=cFuncName, j

ssec_tleobs = ltx_rnxobs_reporting.obstab_tleobs_overview(gnss=dTab['info']['gns
                                                          navsigs=dTab['nav_sign
                                                          navsig_plts=dTab['plot
                                                          navsig_obst_lst=lst_na
                                                          dPNT=dTab['PNT'])
# report to the user
sec_obstab.append(ssec_tleobs)

sec_obstab.generate_tex(os.path.join(dTab['ltx']['path'], dTab['ltx']['obstab'])
sys.exit(55)

# store the json structure
jsonName = os.path.join(dTab['dir'], '{scrname:s}.json'.format(scrname=os.path.s
with open(jsonName, 'w+') as f:
    json.dump(dTab, f, ensure_ascii=False, indent=4, default=amutils.json_conver

# clean up
copyfile(log_name, os.path.join(dTab['dir'], '{scrname:s}.log'.format(scrname=os
os.remove(log_name)
```

if **name** == "**main**": # Only run if this file is called directly main_obstab_analyse(sys.argv)