ex_02_bepohl

November 15, 2022

1 Berechnung einer absoluten Positionierung mit Code-Messungen, Teil 2

1.1 Libraries and data cleaning

[ ]:
```
%%capture
# Requirements
# pip install git+https://github.com/GNSSpy-Project/gnsspy
# pip install pyunpack
# pip install georinex

# Libs
import gnsspy as gp
import numpy as np
import georinex
import geopandas
import matplotlib.pyplot as plt
import math
import pandas as pd

# Params
np.set_printoptions(formatter={'float': '{: 0.5f}'.format})
plt.rcParams['figure.dpi'] = 300
```

[ ]:
```
%%capture
# Dataimport
station = gp.read_obsFile("./data/ONSA0320.11O")
ephemerides = georinex.load("./data/G3_11032.PRE")
clock = gp.read_clockFile("./data/cod16212.clk")
```

[ ]:
```
# get dfs for each epoch
clock_epoch_0 = clock[clock.Epoch == "2011-02-01 00:00:00"]
clock_epoch_1 = clock[clock.Epoch == "2011-02-01 00:15:00"]
clock_epoch_2 = clock[clock.Epoch == "2011-02-01 00:30:00"]
clock_epoch_3 = clock[clock.Epoch == "2011-02-01 00:45:00"]
clock_epoch_4 = clock[clock.Epoch == "2011-02-01 01:00:00"]
```

```
epochs_clock = list((clock_epoch_0, clock_epoch_1, clock_epoch_2,␣
↪clock_epoch_3, clock_epoch_4))

for epoch in epochs_clock:
    epoch = epoch.drop("Epoch", axis=1)

ephemerides_epoch_0 = ephemerides.sel(time="2011-02-01T00:00:00.000000000")
ephemerides_epoch_1 = ephemerides.sel(time="2011-02-01T00:15:00.000000000")
ephemerides_epoch_2 = ephemerides.sel(time="2011-02-01T00:30:00.000000000")
ephemerides_epoch_3 = ephemerides.sel(time="2011-02-01T00:45:00.000000000")
ephemerides_epoch_4 = ephemerides.sel(time="2011-02-01T01:00:00.000000000")

epochs_ephemerides = list((ephemerides_epoch_0, ephemerides_epoch_1,␣
↪ephemerides_epoch_2, ephemerides_epoch_3, ephemerides_epoch_4))
```

[ ]:
```
# Clean all P1 / P2 for the epochs (only observed sats)
P1_epoch_0 = station.observation.P1["2011-02-01 00:00:00"]
P1_epoch_0 = P1_epoch_0.filter(like='G', axis=0).dropna()
P2_epoch_0 = station.observation.P2["2011-02-01 00:00:00"]
P2_epoch_0 = P2_epoch_0.filter(like='G', axis=0).dropna()

P1_epoch_1 = station.observation.P1["2011-02-01 00:15:00"]
P1_epoch_1 = P1_epoch_1.filter(like='G', axis=0).dropna()
P2_epoch_1 = station.observation.P2["2011-02-01 00:15:00"]
P2_epoch_1 = P2_epoch_1.filter(like='G', axis=0).dropna()

P1_epoch_2 = station.observation.P1["2011-02-01 00:30:00"]
P1_epoch_2 = P1_epoch_2.filter(like='G', axis=0).dropna()
P2_epoch_2 = station.observation.P2["2011-02-01 00:30:00"]
P2_epoch_2 = P2_epoch_2.filter(like='G', axis=0).dropna()

P1_epoch_3 = station.observation.P1["2011-02-01 00:45:00"]
P1_epoch_3 = P1_epoch_3.filter(like='G', axis=0).dropna()
P2_epoch_3 = station.observation.P2["2011-02-01 00:45:00"]
P2_epoch_3 = P2_epoch_3.filter(like='G', axis=0).dropna()

P1_epoch_4 = station.observation.P1["2011-02-01 01:00:00"]
P1_epoch_4 = P1_epoch_4.filter(like='G', axis=0).dropna()
P2_epoch_4 = station.observation.P2["2011-02-01 01:00:00"]
P2_epoch_4 = P2_epoch_4.filter(like='G', axis=0).dropna()

epochs_P1 = list((P1_epoch_0, P1_epoch_1, P1_epoch_2, P1_epoch_3, P1_epoch_4))
epochs_P2 = list((P2_epoch_0, P2_epoch_1, P2_epoch_2, P2_epoch_3, P2_epoch_4))
```

1.2 1. A priori receiver clock bias

1.2.1 Definition of functions

[ ]:
```
# Helper funcs for satellite Positions
# Consts (should be defined in func that uses it so it can get reassigned, just␣
↪for safety here)
omega_e = 7.292115e-5 #s^-1
c = 299792458 #m/s

def calculateSatPos(earth_fixed_coords, sat_velocities):
    omega_e = 7.292115e-5 #s^-1
    c = 299792458 #m/s

    earth_fixed_coords_si = earth_fixed_coords * 1000 # km to m
    sat_velocities_si = sat_velocities / 10 # dm/s to m/s

    tau = math.dist(np.array(station.approx_position), earth_fixed_coords_si) /␣
↪c
    sat_coords = np.array(earth_fixed_coords_si).T - np.array(␣
↪(sat_velocities_si + (omega_e * np.array([-earth_fixed_coords_si[1],␣
↪earth_fixed_coords_si[0], 0]))) * tau)

    return sat_coords

# Helper funcs for tropospherical correction
# Rotation matrices
def ry(a): return np.matrix([[np.cos(a), 0, -np.sin(a)], [0, 1, 0], [np.sin(a),␣
↪0, np.cos(a)]])
def rz(a): return np.matrix([[np.cos(a), np.sin(a), 0], [-np.sin(a), np.cos(a),␣
↪0], [0, 0, 1]])

def calculateLatLong(earth_fixed_coords):
    x,y,z = earth_fixed_coords

    lat = math.degrees(math.atan2(z, math.sqrt(x**2 + y**2)))
    lon = math.degrees(math.atan2(y, x))

    return lat, lon

# Topo Coords
def calculateTopoCoords(stat_coord_earth_fixed, earth_fixed_coords_at_send):
    lat_s, lon_s = calculateLatLong(stat_coord_earth_fixed)
    lat_s, lon_s = math.radians(lat_s), math.radians(lon_s)

    # Calculate N, E, U
    topo_coords = ry((math.pi / 2) - lat_s) @ rz(lon_s) @␣
↪(earth_fixed_coords_at_send - stat_coord_earth_fixed).T
    n = -topo_coords[0,0]
```

```python
    e = topo_coords[0,1]
    u = topo_coords[0,2]

    return (n, e, u)

# Zenitwinkel
def calculateZn(topo_coords):
    n, e, u = topo_coords
    return math.degrees(math.atan2(math.sqrt(n**2 + e**2), u))

# Tropo delay
def calculateTropDelay(angle):
    return 2.4 / math.cos(math.radians(angle))

# Relativistic delay
def calculateRelativistics(coord, velocity):
    velocity = velocity / 10 # dm/s to m/s
    return 2 * (coord @ velocity.T) / c
```

1.2.2 Calculate satellite positions for all sats with corrections

[ ]:
```python
# Calculate Sat positions for each epoch for each satellite in Interval A
sat_coords_at_send = list()
velo_at_send = list()
for epoch in epochs_ephemerides: # for each epoch
    epoch_helper = list()
    velo_helper = list()
    for j, coord in enumerate(epoch.position): # for each coordinate
        epoch_helper.append(calculateSatPos(coord, epoch.velocity[j])) #
↪calculate sat position at send time
        velo_helper.append(np.array(epoch.velocity[j]))
    sat_coords_at_send.append(np.array(epoch_helper))
    velo_at_send.append(velo_helper)

# Calculate Tropo delay for each epoch for each satellite in Interval A
tropo_delay = list()
for epoch in sat_coords_at_send: # for each epoch
    epoch_helper = list()
    for i, coords in enumerate(epoch): # for each coordinate
        epoch_helper.
↪append(calculateTropDelay(calculateZn(calculateTopoCoords(station.
↪approx_position, coords))))
    tropo_delay.append(np.array(epoch_helper))

# Calculate Relativistic delay for each epoch for each satellite in Interval A
relativistic_delay = list()
for i, epoch in enumerate(sat_coords_at_send): # for each epoch
```

```
    epoch_helper = list()
    for j, coords in enumerate(epoch): # for each velocity
        epoch_helper.append(calculateRelativistics(coords, velo_at_send[i][j]))
    relativistic_delay.append(np.array(epoch_helper))
```

1.2.3 Calculate clock bias for each satellite

[ ]:
```
def calculateP3k(p_1, p_2):
    f_0 = 10.23 * 10**6
    f_1 = 154 * f_0
    f_2 = 120 * f_0
    return (f_1**2 * p_1 - f_2**2 * p_2) / (f_1**2 - f_2**2)

def calculateI0_i(stat_coord_earth_fixed, earth_fixed_coords_at_send,␣
 ↪clock_bias_i, epoch_i, index):
    c = 299792458 #m/s

    dist = math.dist(np.array(stat_coord_earth_fixed),␣
 ↪earth_fixed_coords_at_send)
    trop = tropo_delay[epoch_i][index]
    relat = relativistic_delay[epoch_i][index]

    return dist + trop + relat - clock_bias_i * c

# Calculate a-priori clock bias for each epoch for each satellite in Interval A
p3k_i = list()
I_0_i = list()
for i, epoch in enumerate(epochs_ephemerides): # for each epoch
    epoch_helper = list()
    epoch_helper_i0 = list()
    observed_satellites = epochs_P1[i].index.str.replace('G', '').astype(int)
    for j, coords in enumerate(epoch.position): # for each delta
        if j in observed_satellites:
            # calculate P3_i
            index = str(j)
            if j < 10: index = "0" + index
            p_1, p_2 = epochs_P1[i]["G" + index], epochs_P2[i]["G" + index]
            epoch_helper.append((index, calculateP3k(p_1, p_2)))

            # Calculate I0_i
            epoch_helper_i0.append((index, calculateI0_i(station.
 ↪approx_position, sat_coords_at_send[i][j - 1],␣
 ↪epochs_clock[i]["DeltaTSV"]["G" + index], i, j - 1)))
    p3k_i.append(pd.DataFrame(epoch_helper).rename(columns={0: "Satellite", 1:␣
 ↪"P3_k"}))
    I_0_i.append(pd.DataFrame(epoch_helper_i0).rename(columns={0: "Satellite",␣
 ↪1: "I_0"}))
```

1.2.4 Do fitting with all clock biases for epoch clock bias

[ ]:
```
# Ausgleichsrechnung
## Init I tilde
I_dash = list()
for i, p3k in enumerate(p3k_i):
    satellites = p3k.Satellite
    I_dash_i = p3k.P3_k - I_0_i[i].I_0
    concat = pd.DataFrame({"Satellite": satellites, "I_dash": I_dash_i})
    I_dash.append(concat)

## Initialize all P matrices
P_epochs = list()
zn = list()
for i, I_dash_i in enumerate(I_dash):
    zn_list = list()
    for j, satellite in enumerate(I_dash_i.I_dash):
        zn_list.append(calculateZn(calculateTopoCoords(station.approx_position,␣
↪sat_coords_at_send[i][I_dash[i].Satellite.astype(int)[j] - 1]))) # ZN of␣
↪satellite i
    P_epochs.append( np.diag(np.cos(np.radians((np.array(zn_list))))**2 ))

## Initialize A matrices
A_epochs = list()
for i, I_dash_i in enumerate(I_dash):
    A_epochs.append(np.full((len(I_dash_i), 1), c))

## do your thing
delta_tk_epochs = list()
for i in range(5):
    delta_tk_epochs.append(np.linalg.inv(A_epochs[i].T @ P_epochs[i] @␣
↪A_epochs[i]) @ A_epochs[i].T @ P_epochs[i] @ I_dash[i].I_dash)
```

1.2.5 Resulting epoch clock biases

[ ]:
```
print("Delta_tk bias for each epoch:")
for delta_tk in delta_tk_epochs:
    print(float(delta_tk))
```

```
Delta_tk bias for each epoch:
-2.607257225155549e-05
-2.607239480478024e-05
-2.6070772278066725e-05
-2.60707140000212e-05
-2.6069728000436512e-05
```

[ ]:
```
# Use this for indexing (eg. sats[1] = "G01")
sats = list()
```

```python
for i in range(40):
    if i < 10: i = "0" + str(i)
    sats.append("G" + str(i))

# Use this for indexing over epochs (eg observed_satellites_epochs[0][1] =␣
↪"G22")
observed_satellites_epochs = list()
for epoch in epochs_P1:
    observed_satellites_epochs.append(list(epoch.index))
```

1.3 2. Stationcoordinates

1.3.1 2a) Correct sat positions with new transmission delay based on clock bias

[ ]:
```python
# 2a) Correction Sat positions
##TODO not working yet, values way too big
def calculateSatPosTau(earth_fixed_coords, sat_velocities, tau):
    omega_e = 7.292115e-5 #s^-1
    c = 299792458 #m/s

    earth_fixed_coords_si = earth_fixed_coords * 1000 # km to m
    sat_velocities_si = sat_velocities / 10 # dm/s to m/s

    sat_coords = np.array(earth_fixed_coords_si).T - np.array(␣
↪(sat_velocities_si + (omega_e * np.array([-earth_fixed_coords_si[1],␣
↪earth_fixed_coords_si[0], 0]))) * tau)

    return sat_coords

# Calculate Sat positions for each epoch for each satellite in Interval A
sat_coords_at_send_corrected = list()
velo_at_send = list()
for i, epoch in enumerate(p3k_i): # for each epoch
    epoch_helper = dict()
    velo_helper = list()
    for j, p3kij in enumerate(epoch.values):
        #print(int(p3kij[0]))
        epoch_helper.update({"G" + p3kij[0]:␣
↪calculateSatPosTau(epochs_ephemerides[i].sel(sv=sats[int(p3kij[0])]).
↪position, epochs_ephemerides[i].sel(sv=sats[int(p3kij[0])]).velocity,␣
↪p3kij[1] / c + delta_tk_epochs[i])}) # calculate sat position at send time
        velo_helper.append(np.array(epochs_ephemerides[i].
↪sel(sv=sats[int(p3kij[0])]).velocity))
    sat_coords_at_send_corrected.append(epoch_helper)
    velo_at_send.append(velo_helper)

print("G25 Corrected Sat Positions:")
```

```
for epoch in sat_coords_at_send_corrected:
    print(epoch["G25"])
```

```
G25 Corrected Sat Positions:
[ 18619309.34432 -15884993.17540  10299699.57366]
[ 18131149.06977 -14641084.79629  12725495.90149]
[ 17588720.65841 -13139931.16253  14932097.42395]
[ 17031860.24737 -11396433.59762  16881478.97330]
[ 16498097.89783 -9433722.69675  18540047.10195]
```

[ ]:
```
observed_satellites_epochs = list()
for epoch in sat_coords_at_send_corrected:
    observed_satellites_epochs.append(list(epoch.keys()))
```

1.3.2 2b) Do fitting based on new coords and biases for station coordinates

[ ]:
```
# Ausgleichsrechnung
## Init I tilde
I_dash = list()
for i, p3k in enumerate(p3k_i):
    satellites = p3k.Satellite
    I_dash_i = p3k.P3_k - I_0_i[i].I_0
    concat = pd.DataFrame({"Satellite": satellites, "I_dash": I_dash_i})
    I_dash.append(concat)

## Initialize all P matrices
P_epochs = list()
zn = list()
for i, I_dash_i in enumerate(I_dash):
    zn_list = list()
    for j, satellite in enumerate(I_dash_i.I_dash):
        zn_list.append(calculateZn(calculateTopoCoords(station.approx_position,␣
 ↪sat_coords_at_send[i][I_dash[i].Satellite.astype(int)[j] - 1]))) # ZN of␣
 ↪satellite i
    P_epochs.append( np.diag(np.cos(np.radians((np.array(zn_list))))**2) )

## Initialize A matrices
A_epochs = list()
for i, epoch in enumerate(observed_satellites_epochs):
    Ai_list = list()
    for j, satellite in enumerate(epoch):
        X_s, Y_s, Z_s = station.approx_position
        X_sat, Y_sat, Z_sat = sat_coords_at_send_corrected[i][satellite]
        dist = math.dist(station.approx_position,␣
 ↪sat_coords_at_send_corrected[i][satellite])
        Ai_list.append(-np.array([(X_sat - X_s) / dist, (Y_sat - Y_s) / dist,␣
 ↪(Z_sat - Z_s) / dist, -1]))
```

```python
    A_epochs.append(np.array(Ai_list))

## do your thing
stat_coord_epochs = list()
for i in range(5):
    delta_stat_coord_epochs = np.linalg.inv(A_epochs[i].T @ P_epochs[i] @␣
 ↪A_epochs[i]) @ A_epochs[i].T @ P_epochs[i] @ I_dash[i].I_dash
    stat_coord_epochs.append(np.array(station.approx_position -␣
 ↪delta_stat_coord_epochs[0:3]))
```

[ ]:
```python
print("Corrected station coordinates for each epoch:")
stat_coord_epochs
```

```
Corrected station coordinates for each epoch:
```

[ ]:
```
[array([ 3370657.62334,  711876.25165,  5349785.95224]),
 array([ 3370658.79049,  711875.77809,  5349787.88527]),
 array([ 3370658.34442,  711875.87220,  5349787.49884]),
 array([ 3370660.74713,  711876.03286,  5349787.16751]),
 array([ 3370657.20009,  711876.10735,  5349785.63052])]
```

1.3.3 2c) Show differences between calculated and given station coordinates

[ ]:
```python
# 2c) Abweichungen zwischen präzisen und berechneten Koordinaten
X_s_prec, Y_s_prec, Z_s_prec = 3370658.4552, 711877.2266, 5349787.0151
stat_prec = (X_s_prec, Y_s_prec, Z_s_prec)
lat_s_prec, lon_s_prec = calculateLatLong(stat_prec)

delta_stat_neu = list()
for i in range(5):
    delta_stat_neu.append(calculateTopoCoords(stat_prec, stat_coord_epochs[i]))

print("Differences for each epoch in North, East, Up coordinates to given␣
 ↪coordinates:")
delta_stat_neu
```

```
Differences for each epoch in North, East, Up coordinates to given coordinates:
```

[ ]:
```
[(0.2782328855576145, -0.782017519303059, -1.4433419595222399),
 (0.44696264559170573, -1.486535156952947, 0.7471603322803547),
 (0.5883360336008537, -1.3022725309985053, 0.19649593971454132),
 (-1.5954571758978595, -1.6415805303571782, 1.2086786301766779),
 (0.4772982662828332, -0.8357374743498829, -1.9541805485606965)]
```