How to use Python for typical applications at AIUB

R. Dach; D. Arnold

April 23, 2019

1. Background

At AIUB, we focus our research on orbit modeling and the best possible representation of observations from Global Navigation Satellite Systems (GNSS). The most popular of the GNSS is the American Global Positioning System (GPS); but there are also other global or regional systems around (GLONASS from Russia, Galileo from Europe, etc.). We are developing our own software to process the related observations: the *Bernese GNSS Software* (BSW) that is written in Fortran with a QT-based user interface.

Currently we are lacking of a library that is able to produce more or less basic plots based on the results. We plan to use Python for this purpose. We try to start after visiting the introductory courses *Python for Science*. In this context the question about the optimal usage arises: where to use dataframes, dictionaries or simple arrays in order to have an optimal performance but also the comfort that a structured data management offers.

For that reason we have compiled a list with typical tasks that we plan to manage with Python. Based on these examples we ask for some advise/recommendations on the best usage of Python.

2. Typical Tasks

2.1. Satellite visibility

Input data:

- Input files are located at `ftp://ftp.aiub.unibe.ch/users/darnold/Python`
- Precise orbit files for 60 days with satellite positions in an Earth fixed Cartesian coordinate system (`RM_yyddd.PRE.bz2`, with `yy` two digit year and `ddd` day of

year).
Below the header the positions (line starts with `P`) for all the individual satellites per epoch (date and time stamp behind the $*$) are given in kilometers. The satellite is identified by a so-called PRN after a character indicating the system (`G`: American GPS; `R`: Russian GLONASS; `E`: European Galileo; `C`: Chinese BeiDou; `J`: Japanese regional QZSS).

- Coordinates of the GNSS tracking stations from a BSW-formatted coordinate file (`IGS14.CRD`).
For each station one line containing the Cartesian coordinates in meters (Potential further information per line are not relevant). It can be assumed that these coordinates are in the same system as the satellite positions.

What and how to compute/plot:

1. The station coordinates for a specific station name (e.g., `ZIMM 14001M004`) have to be extracted from the coordinate file.

2. The station position need to be transferred into geodetic coordinates (latitude, longitude on a ellipsoid): see function `xyzell` in Appendix; use mode `0` in order to obtain latitude and longitude in radians.

3. The positions of a selected subset of satellites (e.g., all GPS satellites) shall be read for a specific interval (e.g., one day).

4. First the difference between the satellite and station coordinates in geocentric coordinates are computed:
$$\Delta\vec{X}_{geo} = \vec{X}_{sat} - \vec{X}_{sta} \tag{1}$$
This vector has to be transformed into a local horizon system at the observing station $\Delta\vec{X}_{loc} = (\Delta x, \Delta y, \Delta z)$: see function `eccell` in appendix.

5. The elevation e and the azimuth a of the satellite as seen by the station need to be computed:
$$e = \arctan \frac{\Delta z}{\sqrt{\Delta x^2 + \Delta y^2}} \tag{2}$$
(Take care on the quadrant). If $e > 5^\circ$ the satellite is assumed to be visible; otherwise ignore it.
The azimuth a is computed by
$$a = \arctan \frac{\Delta x}{\Delta y} \tag{3}$$
(Take care on the quadrant).

6. Plot the trace of each satellite on the sky above the station in an elevation-azimuth diagram (see Fig. 1 for an example).

The selections and options may be put at the beginning of the script. Colors for satellites and systems shall be defined.

Target plot:

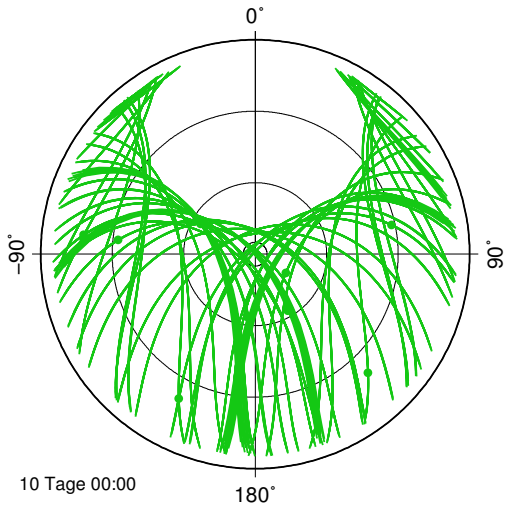

Figure 1: Example for an elevation-azimuth diagram.

The layout of the plot may also be in "Python-style".

2.2. Global map with number of visible satellites

Input data: as in Section 2.1

What and how to compute/plot:

1. Plot a global map with the number of visible satellites for one or more systems (e.g., GPS and GPS+GLONASS) for one given epoch.

2. Check the number of visible satellites ($e > 5°$) according to the description in Section 2.1.

Target plot:

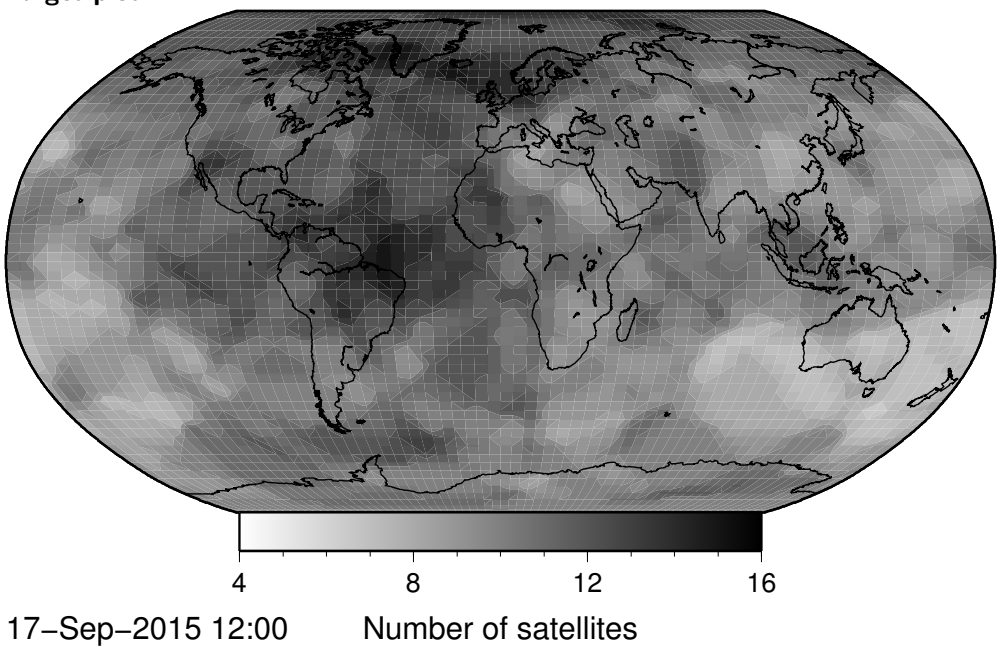

Figure 2: Number of visible GPS satellites.

The layout of the plot may also be in "Python-style".

2.3. Satellite geometry

Input data: as in Section 2.1

What and how to compute/plot:

1. Compute $\Delta\vec{X}_{geo}$ as above in Section 2.1
2. Setup a matrix $\mathbf{A}$ with one line per visible satellite and four columns

$$-\frac{\Delta x_{geo}}{|\Delta\vec{X}_{geo}|} \qquad -\frac{\Delta y_{geo}}{|\Delta\vec{X}_{geo}|} \qquad -\frac{\Delta z_{geo}}{|\Delta\vec{X}_{geo}|} \qquad 1 \tag{4}$$

   Compute the 4×4 matrix

$$\mathbf{C} = (\mathbf{A}^T\mathbf{A})^{-1} \tag{5}$$

3. The so-called PDOP (Position Dilution of Precision) is computed by

$$PDOP = \sqrt{C_{11} + C_{22} + C_{33}} \tag{6}$$

4. Plot a time series of PDOP values for a specific station over a certain interval, e.g., according to Figure 3.

5. Compute spectra of the PDOP series (e.g., Figure 4).

Target plot:

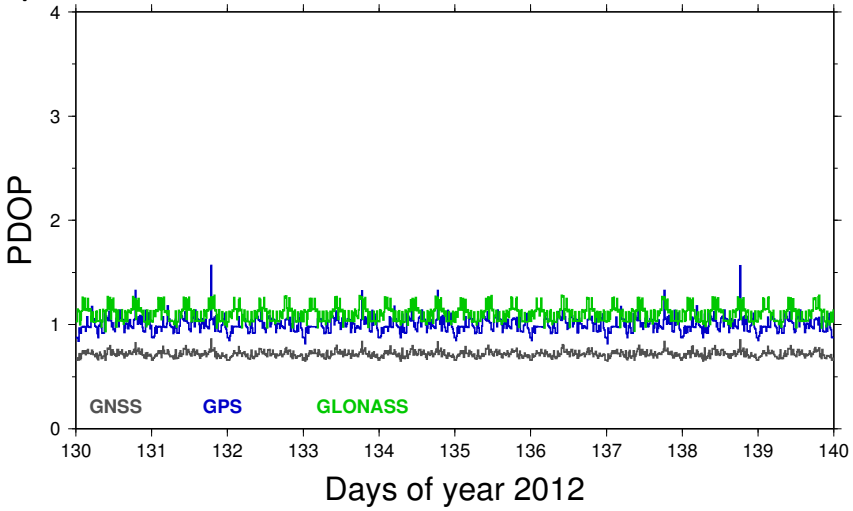

Figure 3: Example for a PDOP time series.

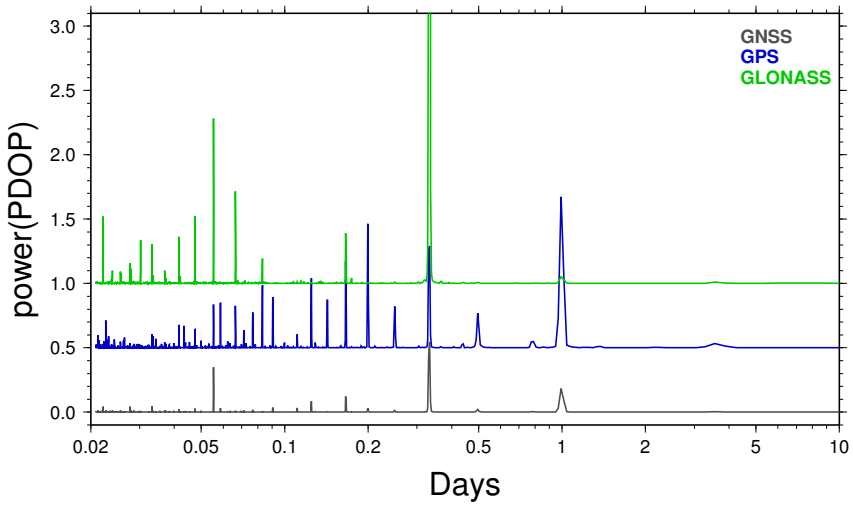

Figure 4: Example for PDOP spectra from 60 days.

2.4. Observation geometry for a LEO

Input data: as in Section 2.1
Instead of one fixed station coordinate the positions of the receiver onboard of a Low Earth Orbiting satellite (LEO) shall also be taken from separate precise orbit files (`RDAUyyddd0.PRE.bz2`). The identifier for the selected satellite is `L47` (Swarm-A). The lines starting with `V` can be ignored.

What and how to compute/plot:

1. Compute the PDOP according to the description in Section 2.3.
2. Plot time series and spectra as in Section 2.3.
3. The latitude and longitude of the LEO for a specific epoch can be computed using the function `xyzell` (mode `1` to obtain degree).
4. Put the PDOP values colorcoded on a global map
   a) as a scatter plot from one day,
   b) as a mean value per grid cell from 60 days.

A. Useful Pyhton Source

Available at `ftp://ftp.aiub.unibe.ch/users/darnold/Python/staUtil.py`.

```
 # -*- coding: utf-8 -*-
"""
Created on Sat Mar 10 05:25:12 2018

@author: dach
"""

# Load general packages:
import math                                 # Trigonometric functions
import numpy as np                          # for matrix operations


# ----------------------------------------------------------------------------
# Coordinate conversion: XYZ -> ELL
def xyzell(Xcrd,Ycrd,Zcrd,mod):
    """
    Name:       staUtil@xyzell

    Purpose:    converts cartesian to ellipsoidal coordinates

    Usage:      (B, L, H) = xyzell(x, y, z, mod)

    Example:    xyzell($cor[0],$cor[1],$cor[2],1)

    Parameter: x    : X-coordinate
               y    : Y-coordinate
               z    : Z-coordinate
               mod : 0=radians
                     1=degrees

    Author:     R. Dach
    """

    # Changes:
    # DD-Mmm-YYYY NN: .....
    #
    # ----------------------------------------------------------------------------


    # pi, semi major and minor axis, eccentricities
    aell  = 6378137.000;
    bell  = 6356752.314;
    ef  = (aell**2-bell**2)/aell**2;
    es  = (aell**2-bell**2)/bell**2;

    blh = ( 0.0, 0.0, 0.0 )

    # geocenter or pole?
    if Xcrd == 0.0 and Ycrd == 0.0 and Zcrd == 0.0:
        return blh
    elif Xcrd == 0.0 and Ycrd == 0.0:
        blh = ( 90.0, 0.0, abs(Zcrd) - bell )
        if mod == 0:
            blh[0] = math.radians(blh[0])
        if Zcrd < 0:
            blh[0] = -blh[0]
        return blh

    # additional quantities
    XYRadius = math.sqrt(Xcrd**2+Ycrd**2)
    Tangle   = math.atan2(Zcrd*aell,XYRadius*bell)

```

```
63      # compute latitude and longitude
64      lam = math.atan2(Ycrd,Xcrd)
65      phi = math.atan2((Zcrd+es*bell*math.sin(Tangle)**3),(XYRadius-ef*aell*math.cos(Tangle)**3));
66
67  # additional quantity
68      ellRadius = aell**2/math.sqrt(aell**2*math.cos(phi)**2+bell**2*math.sin(phi)**2)
69
70      # compute height
71      hell = XYRadius/math.cos(phi)-ellRadius
72
73      if mod == 1:
74          phi = math.degrees(phi)
75          lam = math.degrees(lam)
76
77      blh = (phi,lam,hell)
78
79      # Return either degree or rad
80      return blh
81  # --------------------------------------------------------------------------
82
83
84  # --------------------------------------------------------------------------
85  # Conversion of a local vector:  dXYZ -> dNEU
86  def eccell(Bcrd,Lcrd,Hcrd,Xdel,Ydel,Zdel):
87      """
88      Name:       staUtil@eccell
89
90      Purpose:    compute local ellipsoidal eccentricities (north,east,up in meters)
91                  from global cartesian wgs-84 eccentricities (dx,dy,dz in meters)
92
93      Usage:      (dn, de, du) = eccell(B, L, H, dx, dy, dz)
94
95      Example:    eccell(cor[0],cor[1],cor[2],ecc[0],ecc[1],ecc[2])
96
97      Parameter: B    : latitude (radian)
98                 L    : longitude (radian)
99                 H    : el.. height (meters)
100                dx   : eccentricity in X
101                dy   : eccentricity in Y
102                dz   : eccentricity in Z
103
104     Author:     R. Dach
105     """
106
107     # Changes:
108     # DD-Mmm-YYYY NN: .....
109     #
110     # --------------------------------------------------------------------------
111
112     # SIN and COS functions
113     sphi=math.sin(Bcrd)
114     cphi=math.cos(Bcrd)
115     slmb=math.sin(Lcrd)
116     clmb=math.cos(Lcrd)
117
118     # compute rotation matrix
119     drmat = np.empty([3, 3])
120     drmat[0,0]=-sphi*clmb
121     drmat[0,1]=-sphi*slmb
122     drmat[0,2]= cphi
123     drmat[1,0]=     -slmb
124     drmat[1,1]=      clmb
125     drmat[1,2]= 0
126     drmat[2,0]= cphi*clmb
127     drmat[2,1]= cphi*slmb
128     drmat[2,2]= sphi
129
```

```
130
131     # rotate eccentricities
132     neu = np.empty([3])
133     neu[0]=drmat[0,0]*Xdel+drmat[0,1]*Ydel+drmat[0,2]*Zdel
134     neu[1]=drmat[1,0]*Xdel+drmat[1,1]*Ydel+drmat[1,2]*Zdel
135     neu[2]=drmat[2,0]*Xdel+drmat[2,1]*Ydel+drmat[2,2]*Zdel
136
137     return neu
138 # ----------------------------------------------------------------------------
```