Convenient Fringe-Fit Data Storage in Python Dictionaries for VO2187 Experiment

Leonid Benkevitch, May 13, 2025

Table of Contents

Motivation

In Very Long Baseline Interferometry (VLBI), the correlated and fringe-fit data are stored in the Mark4 format, a 2-level directory tree. The top directory name is a 4-digit number, below it contains directories named `<doy>-<time>[<letter>]`. Each of these directories holds the data files for a scan: the cross- and auto-correlated data, and the fringe-fit data. For example:

```
2187
  ├── 187-1803b
      ├── 0458-020.3HJQAB
      ├── E..3HJQAB
      ├── EE..3HJQAB
      ├── G..3HJQAB
      ├── GE..3HJQAB
      ├── GE.X.6.3HJQAB
      ├── HE..3HJQAB
      ├── HE.X.2.3HJQAB
    .  .  .  .  .  .
```

The file names only have the baseline letters. The fringe-fit file names also have `".X."`. In order to access the information stored in the files the package HOPS is used. For example, here is how to access data items in a single file `GE.X.6.3HJQAB` located in the `"/home/benkev/Work/2187/scratch/Lin_I/2187/187-1803b/"` directory:

```
$ cd /home/benkev/Work/2187/scratch/Lin_I/2187/187-1803b/
```
In Python or IPython:
```
from vpal import fringe_file_manipulation as ffm
f_obj = ffm.FringeFileHandle()
f_obj.load("GE.X.6.3HJQAB")
src = f_obj.source            # Celestial source
phase = f_obj.resid_phas      # Residual phase
dtec = f_obj.dtec             # Differential Total Electron Content
ttag = f_obj.time_tag         # Time or measurement, seconds
mbdelay = f_obj.mbdelay*1e6   # Multiband delay, us
sbdelay = f_obj.sbdelay*1e6   # Single-band delay, us
snr = f_obj.snr               # Signal to noise ratio
```

The data in a single file are for a specific polarization correlation product (for linear polarization, one of XX, XY, YX, YY, or I for pseudo Stokes I (or, further, pI) parameter). It can be found with the following code:

```
import hopstestb as ht
pp_list = \
    ht.get_file_polarization_product_provisional("GE.X.6.3HJQAB")
pp = pp_list[0]          # Polarization Product
```

With such an organization of information, retrieving multiple data items that meet several criteria (for example, the time interval of scanning a particular source for baselines that make up a triangular closure) becomes quite a non-trivial task. Directly using the Mark4 files is also inefficient because a lot of time is spent opening and reading multiple files.

The data needed for a particular analysis can be extracted from all of the fringe-fit file Mark4 database only once. The extracted data can be stored in data structures that provide convenient access. For example, using a good data structure would make it possible to access the whole data cluster related to a celestial source, or a time tag, or a baseline, or a baseline closure triangle.

Python has a built-in dictionary type, currently implemented as a hash table. "Multi-dimensional" dictionaries (or dictionaries of sub-dictionaries of sub-sub-dictionaries...) are ideal containers for storing Mark4 fringe-fit data and for easy access to it. Several Python dictionaries have been developed that are created once and written to disk using the Python `pickle` module. To access the fringe-fit data, one or more of the appropriate dictionaries need to be “unpickled” into memory as Python `dict` (i.e. “dictionary”) type variables.

Dictionaries to Study Effects of PolConversion

I was given the task of statistically studying the effects of PolConversion by comparing the original experiment VO2187 data with those transformed by PolConvert software. The original VO2187 data are obtained from various receivers with mixed polarization. I. Martí-Vidal *et al* (2016) wrote:
"*As we have already noted, ALMA uses receivers that record the signal on a linear (X/Y) basis, whereas VLBI stations mostly record the signals on a circular (R/L) basis.*".
"*The [PolConvert] program applies the calibration and conversion equations … for a phased array with linear-feed receivers. It … identifies the antenna(s) with linear feeds used in the observations; and converts the visibilities to a pure circular basis*"
Thus, the data resulted from the PolConversion are totally circularly-polarized. To distinguish the data before and after the PolConversion I (quite provisionally) call the former *linear* and the latter *circular.*

Since I have two Mark4 fringe-fit datasets, the linear and the PolConverted circular, I store their data in the pairs of dictionaries with slightly different names. The linear files and dictionaries are ended with the letter "l", while the circular files and dictionaries are ended with the letter "c". The files have also "I" letter to indicate that they contain pure pseudo-Stokes I data. Here are the pickle file lists for both polarizations:

Linear	**Circular**
`idx2187lI.pkl`	`idx2187cI.pkl`
`idxs2187lI.pkl`	`idxs2187cI.pkl`
`idxf2187lI.pkl`	`idxf2187cI.pkl`
`clos2187lI.pkl`	`clos2187cI.pkl`
`clot2187lI.pkl`	`clot2187cI.pkl`

The following two pickle files are common for both linear and circular datasets.

`bls_2187.pkl`	List of the common baselines, used in both linear and circular datasets
`tribl_2187.pkl`	Dictionary of closure triangles pointing at the baseline triplets

The dictionaries, all or only some of them, can be unpickled into memory and assigned Python variable names using the commands:

```
import pickle
with open('idx2187lI.pkl', 'rb') as finp: idxl = pickle.load(finp)
with open('idx2187cI.pkl', 'rb') as finp: idxc = pickle.load(finp)

with open('idxs2187lI.pkl', 'rb') as finp: idxsl = pickle.load(finp)
with open('idxs2187cI.pkl', 'rb') as finp: idxsc = pickle.load(finp)
```

```
with open('idxf2187lI.pkl', 'rb') as finp: idxfl = pickle.load(finp)
with open('idxf2187cI.pkl', 'rb') as finp: idxfc = pickle.load(finp)

with open('clos2187lI.pkl', 'rb') as finp: closl = pickle.load(finp)
with open('clos2187cI.pkl', 'rb') as finp: closc = pickle.load(finp)

with open('bls_2187.pkl', 'rb') as finp: bls = pickle.load(finp)
with open('tribl_2187.pkl', 'rb') as finp: tribl = pickle.load(finp)
```

As can be seen, for myself I use the following names: `bls, tribl, idxl, idxc` and so on. The dictionaries created from the linear and the circular Mark4 databases have very similar (or the same) structures, so further I will use their "generic" names without the "l" or "c" endings:

Linear	Circular	Generic
idxl	idxc	idx
idxsl	idxsc	idxs
idxfl	idxfc	idxf
closl	closc	clos
clotl	clotc	clot

`bls`: List of common baselines

When I ran PolConvert, it by some reason omitted the 'Y' (i.e. 'Yj', Yebes) station, so it was not present in the baseline list of the PolConverted database. I have not examined the issue yet. Anyway, I have to use only the baselines common for both linear and circular databases. Also, the 'ST' baseline is excluded because the S (Oe, Onsala, north-east) and T (Ow, Onsala, south-west) stations are too close to each other. To avoid selecting the right baselines each time I use them, I prefer keeping the list on disk and read it into the `bls` variable. Here they are:

```
print(bls)
['GE', 'GH', 'GI', 'GM', 'GS', 'GT', 'HE', 'HM', 'HS', 'HT', 'IE',
 'IH', 'IM', 'IS', 'IT', 'ME', 'MS', 'MT', 'SE', 'TE']
```

`tribl`: Dictionary of Closure triangles

The `tribl` dictionary has 3-letter keys of the closure triangles composed of only the baselines in the `bls` list. Each triangle key has its value a tuple of the three baselines that make up the triangle:

```
tribl['HMS'] → ('HM', 'MS', 'HS')
```

Here is the whole dictionary:

```
tribl →
{'EGH': ('GH', 'HE', 'GE'),
'EGI': ('GI', 'IE', 'GE'),
'EGM': ('GM', 'ME', 'GE'),
'EGS': ('GS', 'SE', 'GE'),
'EGT': ('GT', 'TE', 'GE'),
'GHI': ('GI', 'IH', 'GH'),
'GHM': ('GH', 'HM', 'GM'),
'GHS': ('GH', 'HS', 'GS'),
'GHT': ('GH', 'HT', 'GT'),
'GIM': ('GI', 'IM', 'GM'),
'GIS': ('GI', 'IS', 'GS'),
'GIT': ('GI', 'IT', 'GT'),
'GMS': ('GM', 'MS', 'GS'),
'GMT': ('GM', 'MT', 'GT'),
'EHM': ('HM', 'ME', 'HE'),
'EHS': ('HS', 'SE', 'HE'),
'EHT': ('HT', 'TE', 'HE'),
'HMS': ('HM', 'MS', 'HS'),
'HMT': ('HM', 'MT', 'HT'),
'HIM': ('IH', 'HM', 'IM'),
'HIS': ('IH', 'HS', 'IS'),
'HIT': ('IH', 'HT', 'IT'),
'EIM': ('IM', 'ME', 'IE'),
'EIS': ('IS', 'SE', 'IE'),
'EIT': ('IT', 'TE', 'IE'),
'IMS': ('IM', 'MS', 'IS'),
'IMT': ('IM', 'MT', 'IT'),
'EMS': ('MS', 'SE', 'ME'),
'EMT': ('MT', 'TE', 'ME')}
```

idx: 3D Dictionary `idx[baseline][polprod][data_item]`

For any of the available baselines and any of the available polarization products, `idx` contains a sub-sub-dictionary of the data items, sorted in time-ascending order. I work with the pseudo-Stokes I data

only, so the only `polprod` key is `'I'`. Let's see which baselines are available in linear `idxl` and circular `idxc`:

```
idxl.keys() →
 dict_keys(['HE', 'MT', 'IM', 'ST', 'IT', 'MS', 'IS', 'GH', 'GM',
'IH', 'HM', 'GI', 'MY', 'GT', 'IE', 'EY', 'GE', 'TY', 'GY', 'SY',
'TE', 'SE', 'ME', 'IY', 'GS', 'HY', 'HS', 'HT'])

idxc.keys() →
 dict_keys(['HE', 'MS', 'IM', 'MT', 'IS', 'ST', 'IT', 'HM', 'GH',
'GI', 'GM', 'IH', 'GT', 'GS', 'TE', 'SE', 'ME', 'IE', 'GE', 'HS',
'HT'])
```

Polproducts for an arbitrary bl, say, `'HE'`:

```
idxl['HE'].keys() →
dict_keys(['I'])
```

The possible data items (the same set for any baseline and any polprod) are:

```
idxl['MS']['I'].keys() →
dict_keys(['time', 'source', 'dir', 'file', 'full_fname', 'mbdelay',
'sbdelay', 'snr', 'tot_mbd', 'tot_sbd', 'phase', 'dtec', 'time_tag',
'thour'])
```

All the numerical data items are Numpy float arrays; others are lists of the same size. The size is determined by the number of times the baseline was used in the course of observation.

`'time_tag':` time in seconds from an epoch (just the value from the fringe-fit file).
`'time':` time in seconds from the session start.
`'thour':` time in hours from the session start (just time/3600).
`'source':` celestial source, like '1803+784' or 'OJ287'.
`'mbdelay':` multiband delay in picoseconds.
`'sbdelay':` single-band delay in picoseconds.
`'phase':` residual phase (resid_phas value from the fringe-fit file).
`'tot_mbd':` total multiband delay
`'tot_sbd':` total single-band delay
`'dtec':` differential total electron content, dTEC
`'snr':` signal-to-noise ratio
`'full_fname':` absolute path to the fringe-fit file with these data items
`'dir':` Mark4 directory name <doy>-<time>[<letter>], like '187-2037b' or such.
`'file':` Mark4 file name, like 'MS.X.6.3HJQPD'

The beauty of this dictionary-based approach is the ease of tracing back any dubitable value: I can always see the source of the value, find the original Mark4 file and double-check it.

For example, we can print out several data items from the linear Mark4 dataset:

```
ixms = idxl['MS']['I']      # Set ixms to the ‘MS’ subdictionary

# Print time (h), source, mbd, phase, snr values from 20 to 24:
for i in range(20,25):
    print("%5.2f %8s %7.2f %6.2f %7.2f" % (ixms['thour'][i],
          ixms['source'][i], ixms['mbdelay'][i], ixms['phase'][i],
          ixms['snr'][i]))

2.35 0059+581  460.60 350.33  217.79
2.44 1849+670 -493.41 146.12   79.24
2.49 0613+570 1888.99 212.69  136.39
2.63    3C274 2499.34  49.08   41.27
2.81 1849+670 -203.81  44.78   72.11
```

Here are the fringe-fit files these data were extracted from:

```
ixms['full_fname'][20:25]
['/home/benkev/Work/2187/scratch/Lin_I/2187/187-2021/MS.X.7.3HJQNK',
 '/home/benkev/Work/2187/scratch/Lin_I/2187/187-2026/MS.X.4.3HJQO4',
 '/home/benkev/Work/2187/scratch/Lin_I/2187/187-2029/MS.X.12.3HJQOD',
 '/home/benkev/Work/2187/scratch/Lin_I/2187/187-2037b/MS.X.6.3HJQPD',
 '/home/benkev/Work/2187/scratch/Lin_I/2187/187-2048/MS.X.5.3HJQQE']
```

I. Martí-Vidal *et al*, Calibration of mixed-polarization interferometric observations. Tools for the reduction of interferometric data from elements with linear and circular polarization receivers, A&A, 2016