~~Measurement of cosmic ray energy spectra with data recorded by the TALE SD~~

Python modules for reading DST files

Ichiro Komae
Osaka Metropolitan Univ.
TADAML @ Academia Sinica, 2025/10/15

Why Read DST Files with Python?

- Traditionally, DST files have been read using JAVA or C.
- Recently, more students are performing data analysis in Python.
  - However, few people know how to read DST files directly in Python.
  - As a result, many students use inefficient workflows, such as:

    Dumping DST contents to text files with dstdump.
    → Extracting lines with grep.
    → Loading and analyzing the text files in Python.

  - This approach is simple but wastes time and disk space.
- By summarizing Python-based methods for reading DST files,
  I hope to help younger students spend more time on analysis,
  not on file conversion.

Tools for reading DST files in Python

- TDSTio
  - $TADIR/sources/c++/TDSTio
  - Developed by Dmitri?
- pydst
  - https://github.com/ranchantan/pydst
  - Developed by Omura
- dstparser
  - https://github.com/TA-DNN/dstparser.git
  - Developed by Anton?
- dstio (alpha version)
  - icrhome03:/home/ikomae/src/dstio_test/dstio
  - Developed by Komae

TDSTio

```
def main():
    # Name of an existing DST file, change as necessary
    fname="file_that_has_rusdraw_bank.dst";

    # Allocating a TDSTChain object (this object allows one
    # to iterate over all events found all DST files)
    dstfiles = TDSTChain()

    # Attempting to add one DST file to the chain, exiting
    # in the case of a failure (failure occurs if the DST file doesn't
    # exist or if it can't be opened for reading)
    if(not dstfiles.AddFile(fname)):
        exit(2)

    # If you want to iterate over more than one dst file, do
    # dstfiles.AddFile(fname1);
    # dstfiles.AddFile(fname2);
    # ...
    # or make an ASCII list file ('listFile.txt') that contains the
    # paths to DST files that you want to read and then use
    # dstfiles.AddFiles("listFile.txt");

    # Allocating the rusdraw class object because
    # we want to access rusdraw variables
    rusdraw = rusdraw_class()
```

- $TADIR/sources/c++/TDSTio
  - Developed by Dmitri?
  - PyROOT is required
- Write in a style similar to C++/ROOT
- We can probably read all dstbanks.
- Does not support banks related to TALE-SD.

```
# Reading all events in all files
while(dstfiles.ReadEvent()) :
    event_num = event_num + 1
    # (optional) skipping events that don't have the 'rusdraw' bank
    # (dstfiles.HaveBank("rusdraw") checks if the event that was
    # just read out has rusdraw bank in it )
    if not dstfiles.HaveBank('rusdraw'):
        sys.write.stderr('event %d doesn\'t have rusdraw bank\n' %
                         (event_num))
        continue
    # Load the rusdraw object with the variables from
    # the event that has been just read out.
    rusdraw.loadFromDST()
    # print the event date
    sys.stdout.write('event %d date = %06d\n' %
                     (event_num,rusdraw.yymmdd))
```

TDSTio

- Probably the only tool that can read all dstbanks in Python.
- Very convenient for PyROOT users.
  - PyROOT, however, does not seem widely used.
  - ROOT analysis is easier and faster in C++.
- Using PyROOT in a virtual environment is somewhat troublesome.
  - This tool may not be suitable for beginners.

pydst

- https://github.com/ranchantan/pydst
  - Developed by Omura
  - Using pycparser
  - The function convert_to_pyobject() converts data from C into Python objects that can be used with NumPy.
    - It supports only primitive and array types via type_.kind.
    - Structs are not supported, so data like tasdcalibev or fdraw cannot be read.

```
def convert_to_pyobject(obj):
    if isinstance(obj, dst.ffi.CData):
        type_ = dst.ffi.typeof(obj)
        if type_.kind == "primitive":
            return np.array(obj, dtype=table[type_.cname])
        elif type_.kind == "array":
            shape = tuple(int(m[1]) for m in re.finditer(r"\[(\d+)\]", type_.cname))
            return np.frombuffer(dst.ffi.buffer(obj), dtype=table[type_.cname.split("[")[0]]).copy().reshape(shape)
        else:
            raise NotImplementedError(type_.kind)
    else:
        return obj
```

dstparser

- https://github.com/TA-DNN/dstparser.git
  - Developed by Anton?
- Reads TA-SD dst files and returns data as a dictionary for machine learning.
  - Output can be easily converted to HDF5 format for DNN training.
- Likely to be introduced in tomorrow's hands-on session.
  - Therefore, features other than reading dst files are not covered here.
- The most convenient tool for TA-SD machine learning tasks.
  - (As far as I know) not usable for other purposes such as TALE-SD analysis or FD analysis.

```
def test_parser(dst_file):
    # The only function that are required to parsing
    # is parse_dst_file
    # paths to the directories with data is dstparser.paths module
    data = parse_dst_file(dst_file, xmax_reader=None)

    for key, val in data.items():
        if isinstance(val, np.ndarray):
            print(key, val.shape)
        else:
            print(key, len(val), val)

    print(f'energy = {data["energy"]}')
    print(f'xmax = {data["xmax"]}')
    print(data["time_traces_low"])
```

dstio

- icrhome03:/home/ikomae/src/dstio_test/dstio
  - Users with an ICRR server account can try the alpha version.
- A tool I am developing to help beginners easily read dst files in Python and perform efficient analysis.
- Reads dst files and returns data as a Python dictionary.
- With a basic understanding of Python dictionaries, it should be easy to start using.

```python
data = dstio.read_dst(dst_path)

print(f"len: {len(data)}")
print(f"keys: {data[0].keys()}")

for event in data:
    print("rusdraw.xxyy:")
    pprint.pprint(event["rusdraw"]["xxyy"])
```

```
len: 1
keys: dict_keys(['tasdcalibev', 'brraw', 'bsdinfo', 'rusdraw'])
rusdraw.xxyy:
(1413,
 1610,
 1618,
 1718,
 1801,
 1804,
 1805,
 1806,
 1808,
 1903,
 1904,
 1905,
 1905,
 2005,
 2006,
 2106,
 2106,
 2307)
```

dstio

- Rewriting the bankname_bank_to_common() function from bankname_dst.c for use in Python.
- Rewriting is straightforward for simple bank structures, but complex ones require many conditional branches.
  - As a result, automating the conversion for complex banks is difficult.

```
rcode +=
    dst_unpacki4_(&rusdmc_.event_num, &nobj, bank, &rusdmc_blen,
                  &rusdmc_maxlen);

rcode +=
    dst_unpacki4_(&rusdmc_.parttype, &nobj, bank, &rusdmc_blen,
                  &rusdmc_maxlen);
rcode +=
    dst_unpacki4_(&rusdmc_.corecounter, &nobj, bank, &rusdmc_blen,
                  &rusdmc_maxlen);
rcode +=
    dst_unpacki4_(&rusdmc_.tc, &nobj, bank, &rusdmc_blen,
                  &rusdmc_maxlen);
rcode +=
    dst_unpackr4_(&rusdmc_.energy, &nobj, bank, &rusdmc_blen,
                  &rusdmc_maxlen);
rcode +=
    dst_unpackr4_(&rusdmc_.height, &nobj, bank, &rusdmc_blen,
                  &rusdmc_maxlen);
rcode +=
    dst_unpackr4_(&rusdmc_.theta, &nobj, bank, &rusdmc_blen,
                  &rusdmc_maxlen);
rcode +=
    dst_unpackr4_(&rusdmc_.phi, &nobj, bank, &rusdmc_blen,
                  &rusdmc_maxlen);
```

```
rcode +=
    dst_unpacki4_py_(rusdmc_dict, "event_num", &nobj, bank, &rusdmc_blen,
                     &rusdmc_maxlen);

rcode +=
    dst_unpacki4_py_(rusdmc_dict, "parttype", &nobj, bank, &rusdmc_blen,
                     &rusdmc_maxlen);
rcode +=
    dst_unpacki4_py_(rusdmc_dict, "corecounter", &nobj, bank, &rusdmc_blen,
                     &rusdmc_maxlen);
rcode +=
    dst_unpacki4_py_(rusdmc_dict, "tc", &nobj, bank, &rusdmc_blen,
                     &rusdmc_maxlen);
rcode +=
    dst_unpackr4_py_(rusdmc_dict, "energy", &nobj, bank, &rusdmc_blen,
                     &rusdmc_maxlen);
rcode +=
    dst_unpackr4_py_(rusdmc_dict, "height", &nobj, bank, &rusdmc_blen,
                     &rusdmc_maxlen);
rcode +=
    dst_unpackr4_py_(rusdmc_dict, "theta", &nobj, bank, &rusdmc_blen,
                     &rusdmc_maxlen);
rcode +=
    dst_unpackr4_py_(rusdmc_dict, "phi", &nobj, bank, &rusdmc_blen,
                     &rusdmc_maxlen);
```

dstio

- The current status is that the following banks are supported.

bank id	bank name	bank id	bank name
12001	fraw1	13104	rusdgeom
12002	ftrg1	13105	rusdmc
12013	fscn1	13106	rusdmc1
12092	fdraw	13107	rufldf
12102	brraw	13109	sdtrgbk
12201	lrraw	13112	bsdinfo
13021	tasdcalibev	13207	talesdcalibev
13101	rusdraw	13208	talesdcalib
13103	rufptn		

Summary and ToDo

- Summary
  - Several tools now enable Python-based access to DST files.
  - Each has its own focus — from machine learning (dstparser) to general analysis (TDSTio, pydst, dstio).
  - dstio aims to provide a simple and beginner-friendly interface for efficient DST analysis.

- ToDo
  - Extend dstio support to more DST banks.
  - Make the repository publicly accessible via Git.
    - Ideally integrate dstio into the same repository as dstparser for easier collaboration and consistency.

BackUp