MCMTpy

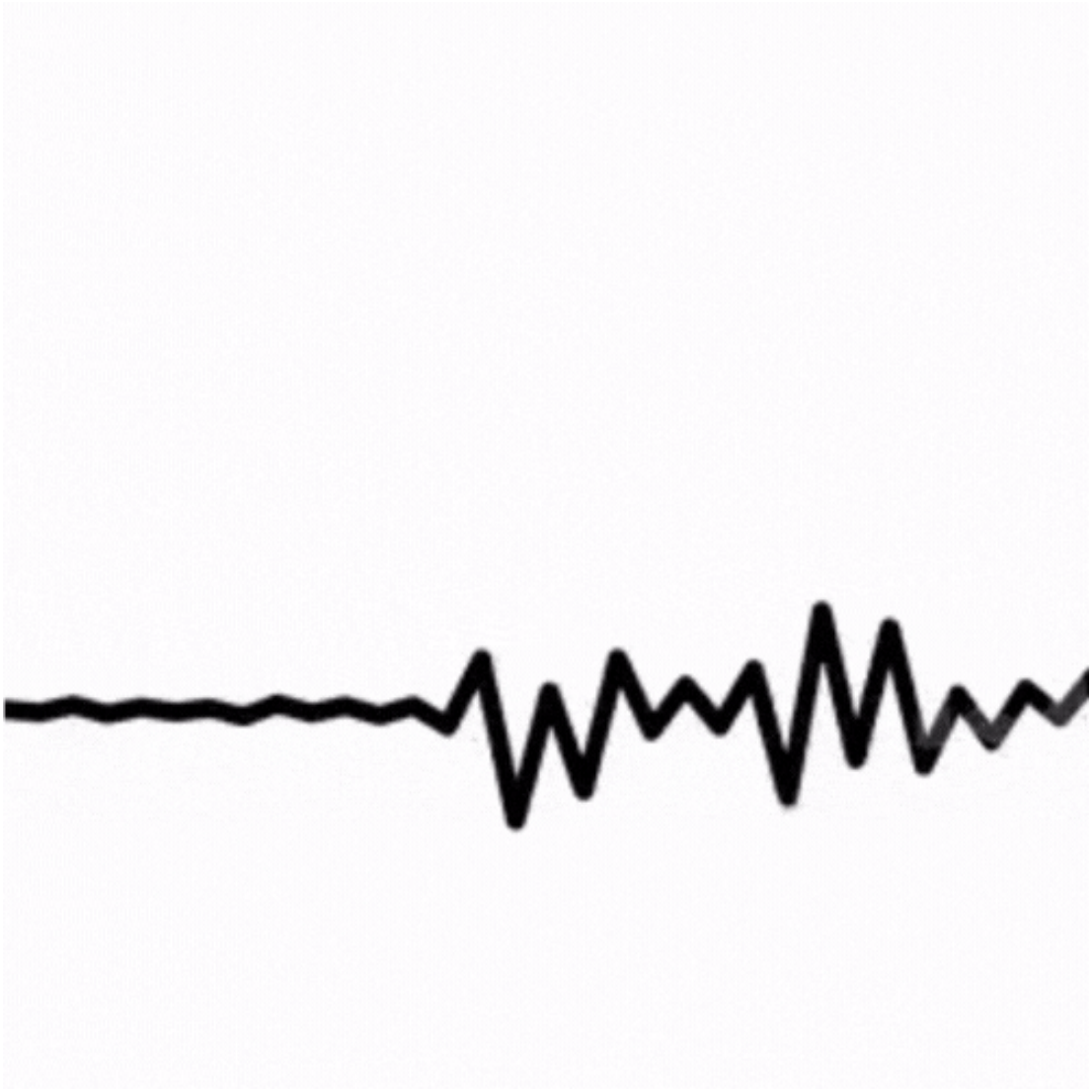
version 0.1.0a1

Fu Yin

六月 28, 2021

Contents

Welcome to MCMTpy-test's documentation!



Note

Version 0.1.0a1 is the currently release.

This is the documentation for the Python package of MCMTpy, which is python tool for seismic source study. For further information please see below website:

- Github repository of MCMTpy: https://github.com/OUCyf

If you use MCMTpy for your research and prepare publications, please citing MCMTpy:

- MCMTpy: A Python Package for Simultaneous Inversion of Source Location, Focal Mechanism, and Rupture Directivity. In prep for Seismological Research Letter.

Contents:

Installation

Note

☐ This package is still undergoing development. ☐

It is not recommended use Parallel installation, because it's somewhat complicated and prepared for the future version for parallel reading of data. Now use Quick installation or Source installation, you can also run MCMTpy in parallel. Please note that the test is performed on macOS Big Sur (11.2.1), so it could be slightly different for other OS.

Dependencies

The package MCMTpy runs on Unix-like systems including Mac and Linux. Package need Python 3.6 or greater, but python 3.9 is not supported for pyfk. It depends on the following Python modules:

- numpy>=1.14
- tqdm>=4.19.4
- matplotlib<=3.1.1
- mpi4py
- obspy
- pyfk
- pyasdf
- json5

For parallel:

- h5py

For data preprocessing and visualization:

- pygmt

We recommend to use anaconda as your python environment, and use conda and pip to install those librarys.

Quick installation

Firstly, make sure Anaconda has been installed, then:

Contents:

```
$ conda create -n MCMTpy  python=3.8 numpy=1.16 matplotlib=3.1.1 mpi4py obspy pyasdf json5 tqdm
$ conda activate MCMTpy
$ pip install pyfk
$ pip install MCMTpy
```

Some errors may occurred of that pyfk not support the new version of cysignals, please:

```
$ conda uninstall cysignals
$ pip install cysignals==1.10.2
$ pip install pyfk
```

Successful installation will generate an executable program MCMTpy under the anaconda environment path: /Users/user/opt/anaconda3/bin. And run following code to check the installation:

```
$ MCMTpy --help
```

Source installation

```
$ git clone https://github.com/OUCyf/MCMTpy.git
$ cd MCMTpy
$ python setup.py install
```

Parallel installation (for future version)

1. Install pyfk

Firstly, make sure Anaconda has been installed, then:

```
$ conda create -n MCMTpy  python=3.8 numpy=1.16 obspy
$ conda activate MCMTpy
$ pip install pyfk
```

Some errors may occurred of that pyfk not support the new version of cysignals, please:

```
$ conda uninstall cysignals
$ pip install cysignals==1.10.2
$ pip install pyfk
```

☐ Make sure the new environment does not include the following packages that we will install below: ☐

2. Install the HDF5 parallel version

- First, make sure openmpi or other MPI program has been installed, preferably source code installed. You can use which mpicc to get the installation's path.
- Activate the MCMTpy environment, and uninstall the mpi4py package in this environment (if it have). Then use pip install the mpi4py package not conda.

```
$ conda uninstall mpi4py
$ pip install mpi4py
```

Note

When you use pip install mpi4py, it will use the mpicc in your system. But if you use conda to do the installation, it resolves the dependent environment itself and install conda's mpicc in your environment which will cause problems for subsequent installations due to the not source code compilation of MPI. After installing mpi4py, you can use which mpicc to check whether it is a system mpicc or a conda mpicc.

- Then you need to install the parallel version of HDF5. Download the source package, compile and install the parallel version to your path. Your system may already have HDF5 installed. Software such as GMT relies on the HDF5 package, so use h5cc-showconfig to see more information about installation, note that h5cc is usually not shown in parallel. We do not want to uninstall the existing version of HDF5 on our system. This does not conflict with our subsequent installation of the parallel version of HDF5, as long as we specify the installation path of the parallel version.

```
$ chmod 777 ./configure  # enable read/write access
$ ./configure --enable-parallel --enable-shared --prefix=/your/path/of/hdf5_dir/
$ make
$ make check
$ make install
```

- After the installation, the executable program will be generated in ./bin path of your hdf5_dir directory. h5pcc will be generated due to the parallel installation, similar to the h5cc will be generated by the non-parallel installation. Check the installation version information:

```
$ h5pcc -showconfig
```

3. Install the h5py parallel version

- Before installing h5py, you need to uninstall both hdf5 and h5py in your environment, and then download h5py <https://github.com/h5py/h5py> and compile the source code. You need to specify the path of HDF5 and CC.

```
$ export CC=/path/to/mpicc  # do not use conda's mpicc
$ echo %CC
$ export HDF5_MPI="ON"
$ export HDF5_DIR="/your/path/of/hdf5_dir/"
$ pip install .
```

- Check the installation version information (download the demo2.py on h5py):

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from mpi4py import MPI
import h5py

rank = MPI.COMM_WORLD.rank  # The process ID (integer 0-3 for 4-process run)

f = h5py.File('parallel_test.hdf5', 'w', driver='mpio', comm=MPI.COMM_WORLD)

dset = f.create_dataset('test', (4,), dtype='i')
dset[rank] = rank

f.close()
```

Run the program:

```
$ mpiexec -n 4 python demo2.py
```

Looking at the file with h5dump:

```
$ h5dump parallel_test.hdf5
HDF5 "parallel_test.hdf5" {
GROUP "/" {
  DATASET "test" {
    DATATYPE  H5T_STD_I32LE
    DATASPACE  SIMPLE { ( 4 ) / ( 4 ) }
    DATA {
    (0): 0, 1, 2, 3
    }
  }
```

```
}
}
```

1. Install pyasdf parallel version

- After the above steps are completed sucessful, you can directly use pip to install pyasdf

```
$ pip install pyasdf
$ python -c "import pyasdf; pyasdf.print_sys_info()"  # (check for a parallel version)
```

5. Install MCMTpy parallel version

- Finally, you can install the parallel version of MCMTpy

```
$ pip install MCMTpy       # or python setup.py install
$ pip uninstall MCMTpy     # uninstall
```

Reference Installation Blog

- https://drtiresome.com/2016/08/23/build-and-install-mpi-parallel-hdf5-and-h5py-from-source-on-linux/
- https://gist.github.com/kentwait/280aa20e9d8f2c8737b2bec4a49b7c92
- https://github.com/h5py/h5py/issues/759
- https://seismicdata.github.io/pyasdf/installation.html
- https://noise-python.readthedocs.io/en/latest/installation.html
- https://docs.h5py.org/en/stable/mpi.html
- https://docs.h5py.org/en/stable/build.html
- https://github.com/conda-forge/hdf5-feedstock/issues/118
- https://distarray.readthedocs.io/en/latest/hdf5-notes.html
- https://forum.hdfgroup.org/t/installing-hdf5-ready-version-of-open-mpi/4998

Tutorial

In this tutorial, we will show some of the key steps in MCMTpy for Focal Mechanism Inversion, including DC (double couple) and MT (moment tensor) inversion (not in now).

About ASDF format

The users who are interested in the details of ASDF format are referred to the following publication. And the pyasdf Github repository https://github.com/SeismicData/pyasdf.

- Krischer, L., Smith, J., Lei, W., Lefebvre, M., Ruan, Y., de Andrade, E.S., Podhorszki, N., Bozdağ, E. and Tromp, J., 2016. An adaptable seismic data format. Geophysical Supplements to the Monthly Notices of the Royal Astronomical Society, 207(2), 1003-1011.

You can also find some useful examples about pyasdf in NoisePy.

About pyfk

The users who are interested in the details of pyfk are referred to the pyfk Github repository https://github.com/ziyixi/pyfk.

Short tutorial

MCMTpy stores all the parameter information in four JSON files: build_GFs.json, syn.json, sample.json and plot.json, and you can find those json-files in path ./MCMTpy-master/jsons/. For parameters choosing, please refer to Detailed tutorial. The steps to do inversion process are:

1. Calculate green function database:

```
$ MCMTpy build_GFs pyfk  -c ./build_GFs.json
$ mpirun -n 4 MCMTpy build_GFs pyfk  -c ./build_GFs.json        # parallel
```

2. Synthesize the test data:

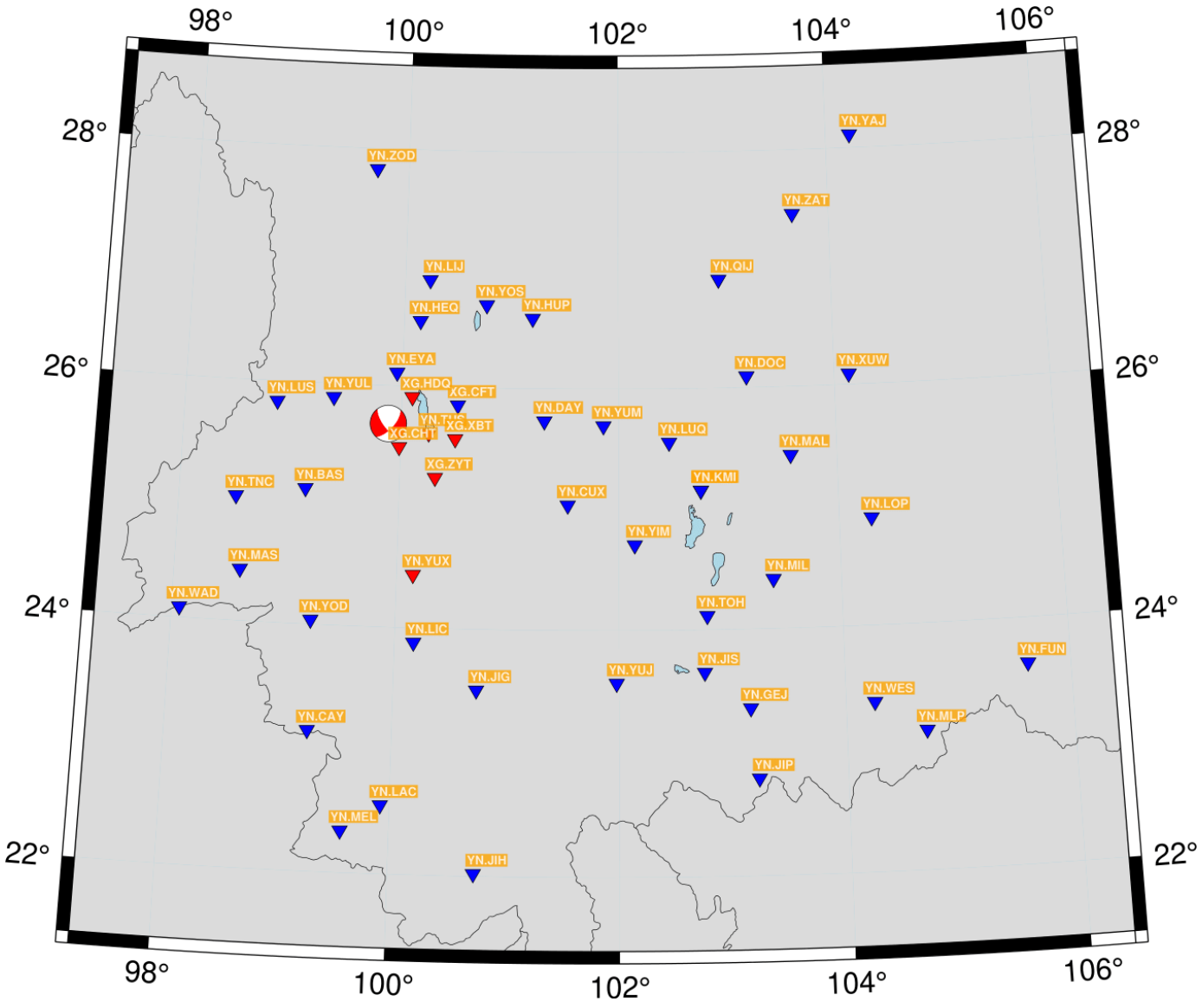
```
$ MCMTpy  syn pyfk  -c ./syn.json
```

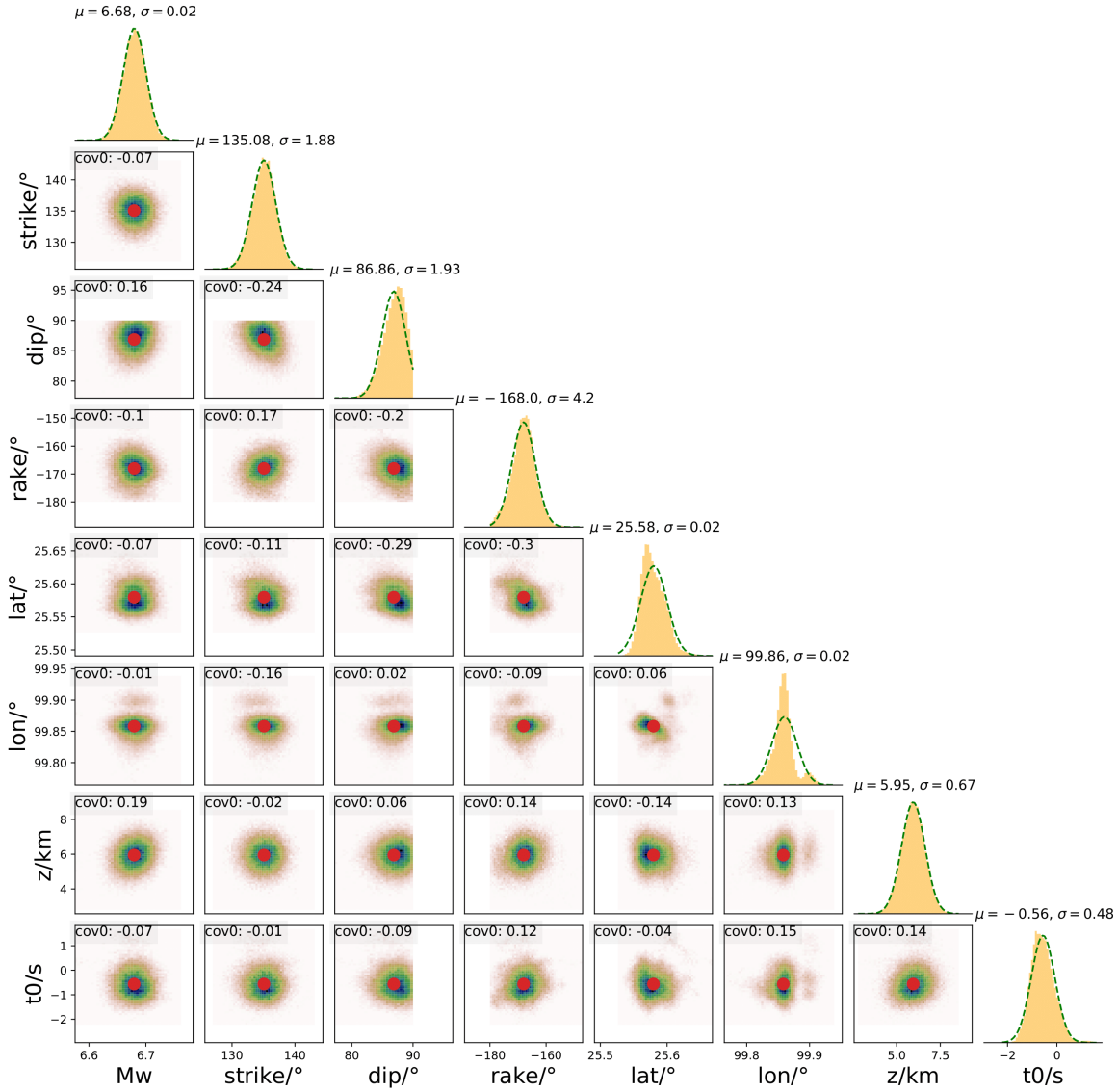
3. Inversion of focal mechanism:

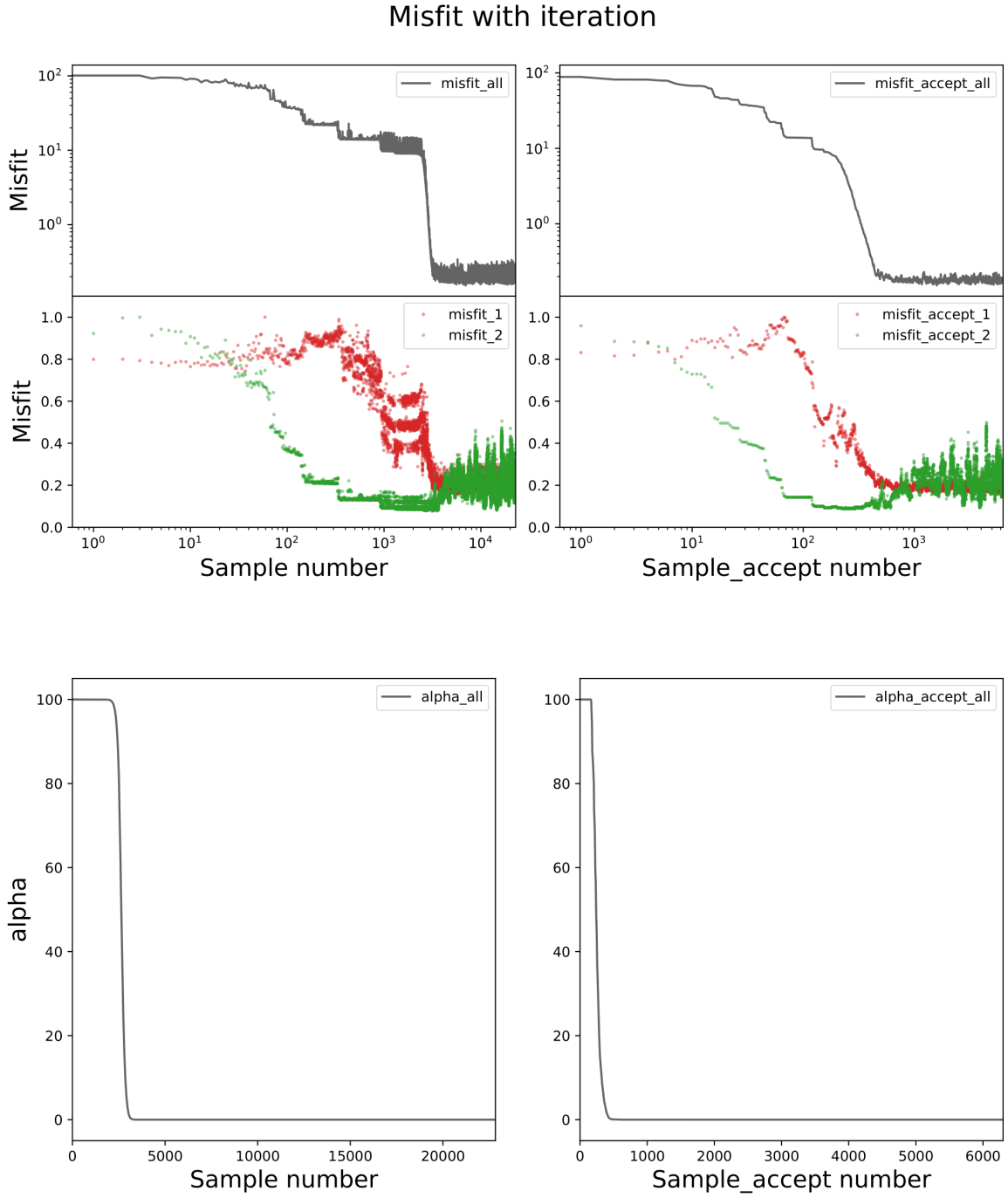
```
$ MCMTpy  sample MH  -c ./sample.json
$ mpirun -n 4  MCMTpy  sample MH  -c ./sample.json        # parallel
```

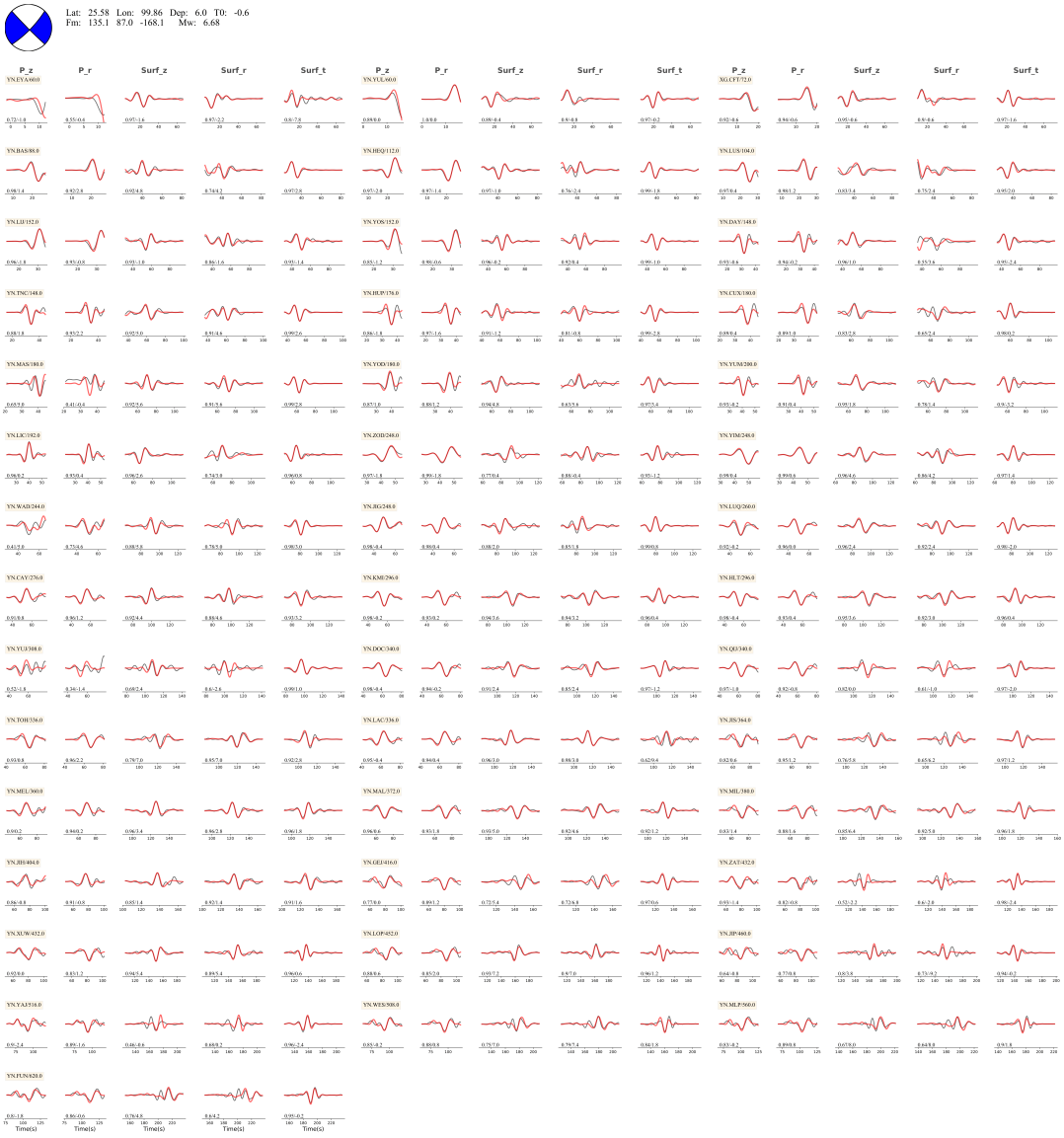
4. Result visualization:

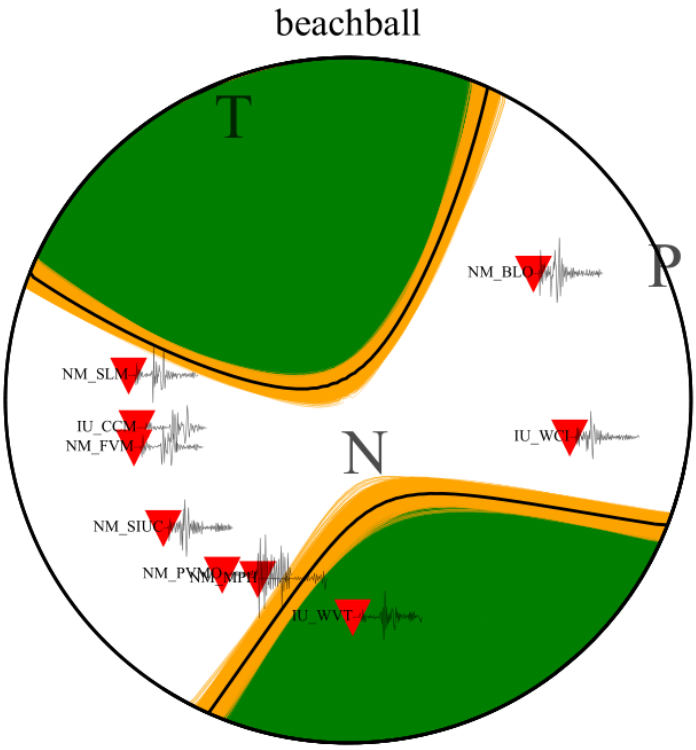
```
$ MCMTpy plot pyfk -c plot.json
```

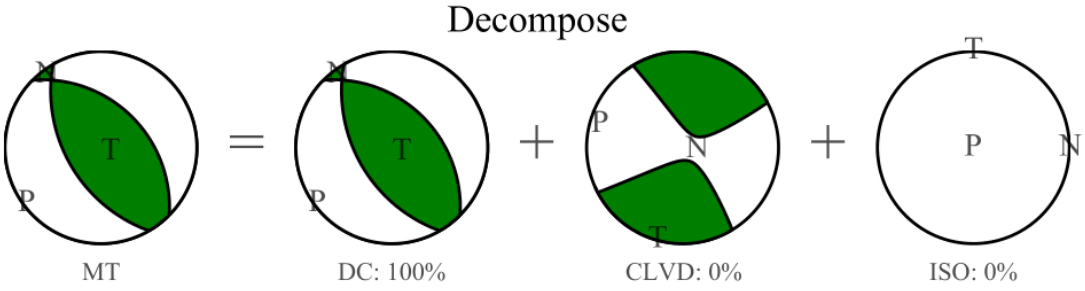



$\mu = 6.68, \sigma = 0.02$
$\mu = 135.08, \sigma = 1.88$
$\mu = 86.86, \sigma = 1.93$
$\mu = -168.0, \sigma = 4.2$
$\mu = 25.58, \sigma = 0.02$
$\mu = 99.86, \sigma = 0.02$
$\mu = 5.95, \sigma = 0.67$
$\mu = -0.56, \sigma = 0.48$
Mw
strike/°
dip/°
rake/°
lat/°
lon/°
z/km
t0/s


Misfit with iteration
misfit_all
misfit_accept_all
Misfit
misfit_1
misfit_2
misfit_accept_1
misfit_accept_2
Sample number
Sample_accept number
alpha_all
alpha_accept_all
alpha
Sample number
Sample_accept number


Lat: 25.58 Lon: 99.86 Dep: 6.0 T0: -0.6
Fm: 135.1 87.0 -168.1 Mw: 6.68
P_z
P_r
Surf_z
Surf_r
Surf_t
Time(s)


beachball
T
P
N
NM_BLO
NM_SLM
IU_CCM
NM_FVM
IU_WCI
NM_SIUC
NM_PWD
NM_MPH
IU_WVT

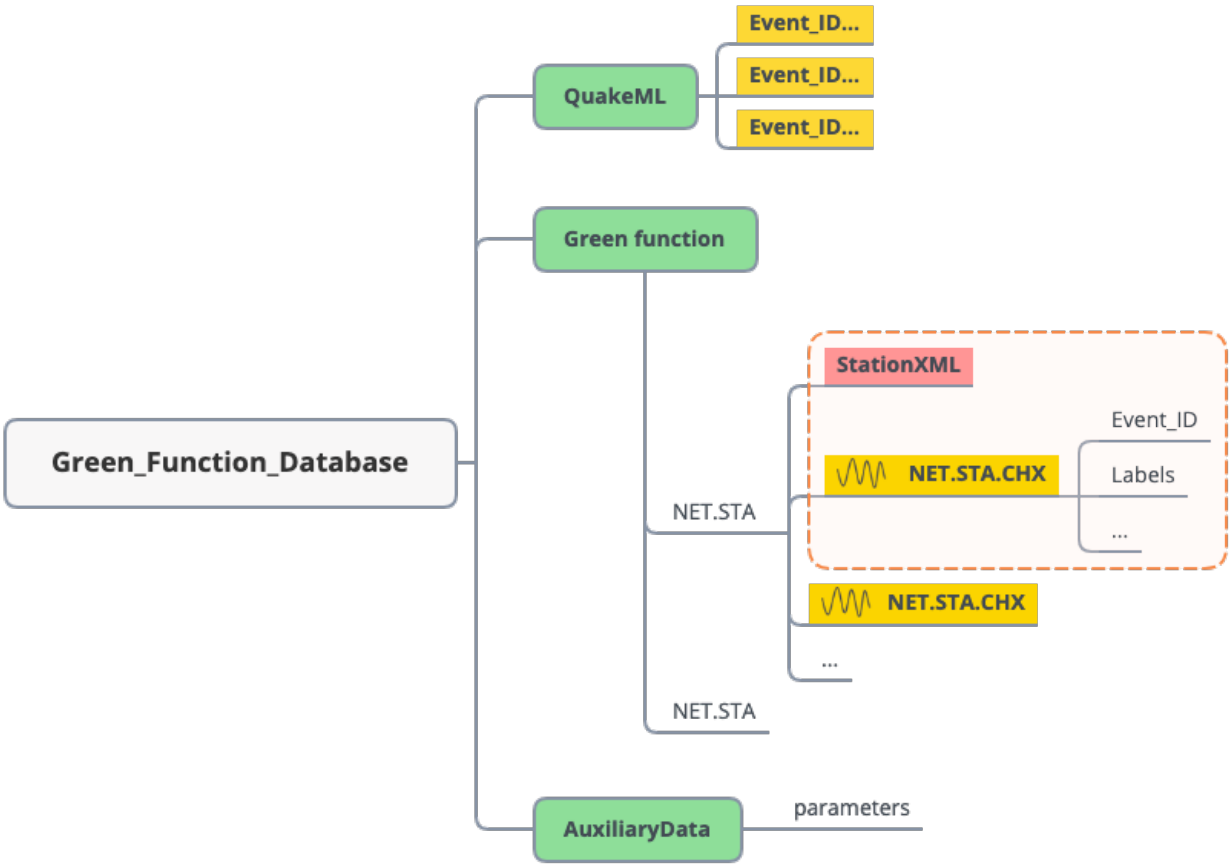

Decompose
T
P
=
T
P
+
P
N
T
+
T
P
N
MT
DC: 100%
CLVD: 0%
ISO: 0%

Tutorial_Detailed

More details on MCMTpy for data processing, parameters choosing can be found in following documentations, and the jupyter notebook can be find in ./MCMTpy-master/data/example_yunnan/Jupyter_notebook:

Calculate Green Function Database

- The first step of MCMTpy is to build Green's function database. All parameters are written into a JSON file named build_GFs.json. Some brief descriptions of the parameters are included here following the definination.
- We do not recommend using relative paths because of the possibility of errors. Absolute paths are preferred.
- We used ASDF seismic data format to manage Green Function database, which is convenient to locally build up a database of pre-processed waveforms. ASDF file greatly reduces the number of green functions files, and a single ASDF file can replace thousands of green function files which may threw us into confusion in the management process. We store the Green's functions corresponding to Event_ID (QuakeML) in "Waveforms" of ASDF data. In the "Labels" of "Waveforms" , we store the phase (P and S) travel times which can be computed in advance by a given velocity model.
- The general structure of MCMTpy stores Green's function with ASDF file format shown in the image below.



- You can also find some useful information about parameters in refers to pyfk.

Station and Source

Database_mode

- Set to be .true. when you want to compute the Green's function library for some region. For real examples, the GF databases may require up to several GBs of free disc space. In addition you need to set the following parameters correctly:

  Source_depth_min, Source_depth_max, Source_depth_spacing, Station_distance_radius_min, Station_distance_radius_max, Station_distance_spacing, Source_name, Network_name, Station_name, Station_depth_reference

- Set to be .false. when you want to calculate the Green's function for stations at which the source is given. You must provide an .txt file named Source_Station_info.txt, which including the longitude latitude and depth information of the source and station. Different sources should be separated by blank lines.

- Source_Station_info.txt format.

  [source_name] [source_lat] [source_lon] [source_depth]

  [network_name] [station_name] [station_lat] [source_lon] [station_depth]

```
1 source_1  25.58 99.86  1
2 YN EYA 26.108 99.947 0
3 YN YUL 25.885 99.371 0
4
5 source_2  25.58 99.86  2
6 YN EYA 26.108 99.947 0
7 YN YUL 25.885 99.371 0
8 XG CFT 25.848 100.517 0
```

```
 9 YN BAS 25.118 99.146 0
10
11 source_3  25.58 99.86  3
12 YN WES 23.372 104.253 0
13 YN MLP 23.128 104.702 0
14 YN FUN 23.624 105.620 0
```

Source_depth_min

- The minimum source depth in Green's function databases (km).

Source_depth_max

- The maximum source depth in Green's function databases (km).

Source_depth_spacing

- Interval of source depths in Green's function databases (km).

Station_distance_radius_min

- The minimum distance of the station in Green's function databases (km or degree, depend by degrees).

Station_distance_radius_max

- The maximum distance of the station in Green's function databases (km or degree, depend by degrees).

Station_distance_spacing

- Horizontal spacing interval in Green's function databases (km or degree, depend by degrees).

Source_name

- Source name in Green's function databases, default value is source, please do not change. Only lowercase letters are supported.

Network_name

- Network name in Green's function databases, default value is NET, please do not change.

Station_name

- Station name in Green's function databases, default value is STA, please do not change.

Station_depth_reference

- Depth of all virtual stations (km).

> **Note**
>
> When Database_mode = .false., depths can vary from station to station. But when set to be .true., all stations must use the same depth Station_depth_reference.

MPI

MPI_n

- CPU number. When Database_mode = .false., the number of call cores should be less than or equal to the number of source. But when set to be .true., there are no limits.

Path

DATADIR

- The output path of the Green function databases.

DATADIR_splits

- A subfolder under DATADIR for Green's function. Please keep the path relative to DATADIR.

SourceModel

source_mechanism

- Focal mechanism, not used. Please set it to .null..

srcType

- The type of Green's function, dc(double-couple) sf(single-couple) ep(explosion). Please set it to .dc..

SeisModel

Velocity_model

- Path to the velocity model file. And the format:

  thick(km) vs(km/s) vp(km/s) density(g/cm^3) Qs Qp

```
 1 5.01000000   3.17601610   5.41925121   2.50000000   600.00000000   1200.00000000
 2 5.01000000   3.36477536   5.67947907   2.55000000   600.00000000   1200.00000000
 3 5.01000000   3.52147424   5.93255556   2.60000000   600.00000000   1200.00000000
 4 5.01000000   3.55152174   6.00984300   2.65000000   600.00000000   1200.00000000
 5 5.01000000   3.54084541   6.02592110   2.70000000   600.00000000   1250.00000000
 6 5.01000000   3.58840419   6.14793478   2.75000000   650.00000000   1300.00000000
 7 5.01000000   3.75369968   6.48572061   2.80000000   700.00000000   1350.00000000
 8 5.01000000   3.94361514   6.86566667   2.90000000   750.00000000   1400.00000000
 9 10.01000000   4.12710548   7.20229388   3.00000000   800.00000000   1500.00000000
10 10.01000000   4.30825765   7.55049678   3.10000000   850.00000000   1600.00000000
11 10.01000000   4.38069968   7.72566425   3.20000000   850.00000000   1700.00000000
12 90.00000000   4.50000000   7.80000000   3.27000000   900.00000000   1800.00000000
```

flattening

- Set the flatten status of the velocity model, .true. or .false..

Config

The following parameters explanation refers to pyfk <https://github.com/ziyixi/pyfk>.

npt

- The sampling points is a multiple of 2.

dt

- Sampling interval in seconds

degrees

- Use degrees instead of km, defaults to .true.. Only when Database_mode = .true., it can be changed to .false.

taper

- Taper applies a low-pass cosine filter at fc=(1-taper)*f_Niquest, defaults to 0.3.

filter

- Apply a high-pass filter with a cosine transition zone between freq. f1 and f2 in Hz, defaults to (0, 0).

dk

- The non-dimensional sampling interval of wavenumber, defaults to 0.3.

smth

- Makes the final sampling interval to be dt/smth, defaults to 1.

pmin

- The min slownesses in term of 1/vs_at_the_source, defaults to 0.

pmax

- The max slownesses in term of 1/vs_at_the_source, defaults to 1.

kmax

- The kmax at zero frequency in term of 1/hs, defaults to 15.

rdep

- Not used now. Please keep the default values of the parameters, defaults to 0 in km.

updn

- The “up” for up-going wave only, “down” for down-going wave only, “all” for both “up” and “down”, defaults to “all”.

samples_before_first_arrival

- The number of points before the first arrival, defaults to 50.

Logging file

- MCMTpy will generates 2 logging files when it builds GFs datebase, that is prepro_para_info.txt and write_Source_Station_info_MPI. They record all the parameter information of GFs datebase. You can find them in DATADIR path.

Example

- The example_path need to de changed to your path, and run this notebook to set build_GFs.json file.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import os
import sys
import json5


# The root directory of the project
example_path = '/Users/yf/3.Project/8.MCMTpy/MCMTpy-master/data/example_yunnan'


#-----------------------------------------------------------#
# we expect no parameters need to be changed below
#-----------------------------------------------------------#
# 1. get notebook path
notebook_path = sys.path[0]
os.chdir(notebook_path)

# 2. change path to Green_Funcs
os.chdir('../Green_Funcs')
print(os.listdir())

# 3. show build_GFs.json
filename = 'build_GFs.json'
with open(filename, 'r',encoding='utf-8') as f1:
    gfs_json=json5.load(f1)
f1.close()

# 4. change parameters with absolute path
gfs_json['Source_Station_info'] = os.path.join(example_path,"Green_Funcs/Source_Station_info.txt")
gfs_json["DATADIR"] = os.path.join(example_path,"Green_Funcs/GFs1")
gfs_json["DATADIR_splits"] = os.path.join(example_path,"Green_Funcs1/GFs1/GFs_splits")
gfs_json["Velocity_model"] =  os.path.join(example_path,"v_model/v_model_yunnan.txt")
gfs_json_new = os.path.join(example_path,'Green_Funcs/build_GFs_new.json')

```

```
with open(gfs_json_new,'w') as f2:
    json5.dump(gfs_json, f2, indent=2)
f2.close()

# !mpirun -n 4 MCMTpy build_GFs pyfk -c build_GFs_new.json  > gfs.log
```

- Now you can run MCMTpy in bash:

```
$ mpirun -n 4 MCMTpy build_GFs pyfk -c build_GFs_new.json  > gfs.log
```

Note

Please follow the above parameter instructions and set all parameters correctly before running the program. Otherwise, it is easy to report an error!

Synthesize the Test Data

- The synthesized test data can help us check whether the Green's function database is correctly calculated.
- We do not recommend using relative paths because of the possibility of errors. Absolute paths are preferred.

Path

GFs_json_file

- The path to the Json-file used to calculate Green Function Database.

Source

srcType

- The type of Source, mt(moment tensor) dc(double-couple) sf(single-couple) ep(explosion).

point

- The latitude longitude and depth of the source.

source_mechanism

- Focal mechanism.
- When srcType = dc: source_mechanism = [Mw, Strike, Dip, Rake]
- When srcType = mt: source_mechanism = [Mw, Mxx, Mxy, Mxz, Myy, Myz, Mzz]

Station

NET_STA

- The information contained in the two-dimensional list: [network_name] [station_name] [station_lat] [source_lon] [station_depth]

Source Time Function

Trapezoid_stf_mode

- Whether to use pyfk provided trapezoid shaped source time function. Set to be .true. you need to set the following parameters correctly:

  dura rise delta

dura

- Duration time (s).

rise

- Rise time (s).

delta

- The time interval (s).

Stf_file

- Stf_file should be an SAC file as the source time function when set Trapezoid_stf_mode to be .false..

Note

To read the data from the Stf_file file, keep the sampling rate of SAC data consistent with the data and GFs.

Filter

filter_mode

- Whether to filter.

freqmin

- The minimum frequency of filtering.

freqmax

- The maximum frequency of filtering.

Add Noise

Add_noise

- Whether add noise to syn.

noise_level_waveform

- Noise level added to the waveform.

noise_level_arrive_time

- Noise level added to the phase time as tp and ts.

Output

Output_mode

- Output data or not.

Output_path

- Path of output data

source_name

- Event name of output data, you can set it any way you want. Only lowercase letters are supported.

ASDF_filename

- The name of the output ASDF file

Example

- The example_path need to de changed to your path, and run this notebook to set syn.json file.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import os
import sys
import json5
import shutil

# The root directory of the project
example_path = '/Users/yf/3.Project/8.MCMTpy/MCMTpy-master/data/example_yunnan'

#-----------------------------------------------------------#
# we expect no parameters need to be changed below
#-----------------------------------------------------------#
# change path to Green_Funcs
notebook_path = sys.path[0]
os.chdir(notebook_path)
os.chdir('../syn')
print(os.listdir())

# show build_GFs.json
filename = 'syn.json'
with open(filename, 'r',encoding='utf-8') as f1:
    syn_json=json5.load(f1)

# change parameters with absolute path
syn_json["GFs_json_file"] = os.path.join(example_path,"Green_Funcs/build_GFs_new.json")
syn_json["Stf_file"] = os.path.join(example_path,"syn/Stf_file/Stf_file.sac")
syn_json["Output_path"] = os.path.join(example_path,"syn/Synthetic")
syn_json_new = os.path.join(example_path,'syn/syn_new.json')

with open(syn_json_new,'w') as f2:
    json5.dump(syn_json, f2, indent=2)
f2.close()

shutil.rmtree('./Synthetic')
# !MCMTpy syn pyfk  -c ./syn_new.json  > syn.log
```

- Now you can run MCMTpy in bash:

```
$ MCMTpy  syn pyfk  -c ./syn.json
```

- View syn waveform:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import os
import pyasdf


```

```
# The root directory of the project
example_path = '/Users/yf/3.Project/8.MCMTpy/MCMTpy-master/data/example_yunnan'

#-----------------------------------------------------------#
# we expect no parameters need to be changed below
#-----------------------------------------------------------#
## change the path
h5_path = os.path.join(example_path,'syn/Synthetic/SYN_test.h5')
source_name = 'source_syn'

### read h5 data --> Sta_data (dict)
ds_raw=pyasdf.ASDFDataSet(h5_path,mpi=False,mode='r')
Station_list = ds_raw.waveforms.list()
Station_num = len(Station_list)

Sta_data={}
for i in range(0,Station_num,1):
    info={}
    tags = ds_raw.waveforms[Station_list[i]].get_waveform_tags()
    if source_name in tags:
        raw_sac = ds_raw.waveforms[Station_list[i]][source_name]
        tp = float( ds_raw.waveforms[Station_list[i]][source_name][0].stats.asdf['labels'][1] )
        ts = float( ds_raw.waveforms[Station_list[i]][source_name][0].stats.asdf['labels'][3] )
        info.update({ "tp": tp})
        info.update({ "ts": ts})
        info.update({ "data": raw_sac})
        Sta_data.update({ Station_list[i]: info})

# plot data
ax = Sta_data['YN.YUM']['data'].plot()
Sta_data['YN.YUM']['data'][0].stats
```

Note

Please follow the above parameter instructions and set all parameters correctly before running the program. Otherwise, it is easy to report an error!

Prepare Data for MCMTpy

Format of MCMTpy input data:

- ASDF file format, and the sampling rate dt is consistent with that of GFs database.
- Must be ENZ three-component data and the name of the channel must be ‘E’/’N’/’Z’ order.

Note

Pyasdf files are automatically sorted by name. So you must make sure that the order of the Trace is still ENZ after the automatic sorting, generally the name of the channel ‘BHE’/’BHN’/’BHZ’ will be fine.

- Trace.Stats.sac.t1/Trace.Stats.sac.t2 must be included in the trace header file to represent the P/S phase time.

- The trace header file must include the station and network name.
- The trace header file must include b/e/o/stla/stlo/stdp/evlo/evla/evdp/mag/dist/az/baz, time information, event longitude and latitude information, azimuth information.
- The trace header file must include starttime/endtime.
- All traces need to retain a uniform number of sampling points before T1, such as p_n0 = 50.

Example

- We have provided some useful scripts for data preprocessing, and the example_path need to de changed to your path, and run this notebook.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import os
import glob
import obspy
import shutil
from obspy import Stream
from obspy.taup import TauPyModel
from MCMTpy.utils.asdf_function import Add_waveforms,Add_stationxml,get_QuakeML,Add_quakeml
from MCMTpy.utils.asdf_function import get_StationXML

#-------------------------------------------------#
# 1.read raw data
def read_data(allfiles_path):

    allfiles = sorted( glob.glob(allfiles_path) )
    data_raw = Stream()
    for i in range(0,len(allfiles),1):
        try:
            tr = obspy.read(allfiles[i])
            data_raw += tr
        except Exception:
            print(allfiles[i],': no such file or obspy read error');continue
    data = data_raw.copy()

    return data

#-------------------------------------------------#
# 2.get_taup_tp_ts
def get_taup_tp_ts(model,depth,distance,degree=None):
    if degree==False:
        distance = distance/111.19

    time_p = model.get_travel_times(source_depth_in_km=depth,
                                    distance_in_degree=distance,
                                    phase_list=["p", "P"])

    time_s = model.get_travel_times(source_depth_in_km=depth,
                                    distance_in_degree=distance,
                                    phase_list=["s", "S"])

    ray_p = time_p[0].ray_param
    tp = time_p[0].time
    angle_p = time_p[0].incident_angle

    ray_s = time_s[0].ray_param
    ts = time_s[0].time
```

```
49    angle_s = time_s[0].incident_angle
50
51    return ray_p,tp,angle_p,ray_s,ts,angle_s
52
53 #-------------------------------------------------#
54 # 3.preprocess
55 def preprocess(data,freqmin,freqmax,samp_freq,p_n0,npts):
56
57    data.detrend(type='demean')
58    data.detrend(type='simple')
59    data.taper(max_percentage=0.05)
60
61    data.filter('bandpass', freqmin=freqmin, freqmax=freqmax, corners=4, zerophase=True)  # bandpass
62    data.resample(samp_freq,window='hanning',no_filter=True, strict_length=False)        # resample
63
64    for i in range(0,len(data),1):
65       b = data[i].stats.sac['b']
66       t1 = data[i].stats.sac['t1']
67       t_start = data[i].stats.starttime + (t1-b) - p_n0/samp_freq
68       t_end_npts = t_start+npts/samp_freq
69       t_endtime = data[i].stats.endtime
70       if t_end_npts > t_endtime:
71          t_end = t_endtime
72       else:
73          t_end = t_end_npts
74
75       data[i].trim(t_start, t_end)
76
77    for i in range(0,round(len(data)/3)):
78       t_start_1 = data[3*i].stats.starttime
79       t_start_2 = data[3*i+1].stats.starttime
80       t_start_3 = data[3*i+2].stats.starttime
81       t_end_1 = data[3*i].stats.endtime
82       t_end_2 = data[3*i+1].stats.endtime
83       t_end_3 = data[3*i+2].stats.endtime
84       t_start = max(t_start_1,t_start_2,t_start_3)
85       t_end = min(t_end_1,t_end_2,t_end_3)
86       data[3*i:3*i+3].trim(t_start, t_end)
87
88    # velocity(nm/s) to velocity(cm/s)
89    for i in range(0,len(data),1):
90       data[i].data = 1e-6*data[i].data
91
92 #-------------------------------------------------#
93 # 4.write_ASDF
94 def write_ASDF(data_enz,Output_path,source_name,ASDF_filename):
95    source_lat   = data_enz[0].stats.sac.evla
96    source_lon   = data_enz[0].stats.sac.evlo
97    source_depth = data_enz[0].stats.sac.evdp
98    source_time  = data_enz[0].stats.starttime + data_enz[0].stats.sac.b
99
100    Output_file_path = os.path.join(Output_path, ASDF_filename+'.h5')
101    catalog_create   = get_QuakeML(source_name, source_lat, source_lon, source_depth,source_time)
102    Add_quakeml(Output_file_path,catalog_create)
103
104    Station_num = round(len(data_enz)/3)
105    for i in range(0,Station_num,1):
106       net_sta_name     = data_enz[3*i].stats.network+'_'+data[3*i].stats.station
107       station_network  = data_enz[3*i].stats.network
108       station_name     = data_enz[3*i].stats.station
```

```
        station_lat      = data_enz[3*i].stats.sac.stla
        station_lon      = data_enz[3*i].stats.sac.stlo
        station_depth    = 0
        station_distance = data_enz[3*i].stats.sac.dist
        station_az       = data_enz[3*i].stats.sac.az
        station_baz      = data[3*i].stats.sac.baz
        tp               = data_enz[3*i].stats.sac.t1
        ts               = data_enz[3*i].stats.sac.t2
        stream           = data_enz[3*i:3*i+3].copy()
        starttime        = data_enz[3*i].stats.starttime
        time_before_first_arrival = p_n0/samp_freq

        inv=get_StationXML(station_network,
                           station_name,
                           station_lat,
                           station_lon,
                           station_depth,
                           'enz')

        Add_waveforms([stream],
                      catalog_create,
                      Output_file_path,
                      source_name,
                      tp,
                      ts,
                      station_distance,
                      station_az,
                      station_baz)

        Add_stationxml(inv,
                       Output_file_path)

        for j in range(0,3,1):
            CHN = stream[j].stats.channel
            sac_filename = os.path.join(Output_path,source_name+'.'+station_network+'.'+station_name+'.'+CHN+'.sac')
            stream[j].write(sac_filename, format='SAC')
#-----------------------------------------------------------#



#-----------------------------------------------------------#
# 0. main
# The root directory of the project
example_path = '/Users/yf/3.Project/8.MCMTpy/MCMTpy-master/data/example_yunnan'
freqmin      = 0.005                    # pre filtering frequency bandwidth (hz)
freqmax      = 2                        # note this cannot exceed Nquist frequency (hz)
samp_freq    = 5                        # targeted sampling rate (hz)
p_n0         = 50                       # number of sampling points before P wave arrives
npts         = 2048                     # data length
Output_path  = os.path.join(example_path,"YN.202105212148_Inv/YN.202105212148_MCMTpy")
source_name  = "source_enz"             # asdf's source_tag
ASDF_filename = "YN.202105212148"       # asdf filename



#-----------------------------------------------------------#
# we expect no parameters need to be changed below
```

```
#-----------------------------------------------------------#
# 1. read raw data
allfiles_path = os.path.join(example_path,'YN.202105212148_Inv/YN.202105212148_raw/*.SAC')
data = read_data(allfiles_path)

# 2. taup ray tracing
model_path = os.path.join(example_path,"v_model/v_model.npz")
model = TauPyModel(model=model_path)
for i in range(0,len(data),1):
    depth = data[i].stats.sac['evdp']
    distance = data[i].stats.sac['dist']
    ray_p,tp,angle_p,ray_s,ts,angle_s = get_taup_tp_ts(model,depth,distance,degree=False)
    data[i].stats.sac["t1"]=tp
    data[i].stats.sac["t2"]=ts

# 3. preprocess
preprocess(data,freqmin,freqmax,samp_freq,p_n0,npts)

# 4. output ASDF and SAC data
shutil.rmtree(Output_path)
os.mkdir(Output_path)
write_ASDF(data,Output_path,source_name,ASDF_filename)
```

Note

Please follow the above parameter instructions and set all parameters correctly before running the program. Otherwise, it is easy to report an error!

Inversion of Source

- This is the core of the MCMTpy, and all parameters are written into a JSON file named sample_dc.json. Some brief descriptions of the parameters are included here following the definination.
- We do not recommend using relative paths because of the possibility of errors. Absolute paths are preferred.

Path

GFs_json_file

- The path to the Json-file used to calculate Green Function Database.

GFs_file

- The path to the Green Function Database.

Raw_data_file

- The path to the Raw data ASDF file.

Source_tag

- Raw data ASDF file's source name which setted in the data preprocess function.

Raw_p_n0

- The number of points before the first arrival. Be consistent with the GFs' samples_before_first_arrival.

Dt

- Sampling rate of the raw data.

Output_path

- Inversion result output file path.

Source Time Function

Trapezoid_stf_mode

- Whether to use pyfk provided trapezoid shaped source time function. Set to be .true. you need to set the following parameters correctly:

  dura rise delta

dura

- Duration time (s).

rise

- Rise time (s).

delta

- The time interval (s).

Stf_file

- Stf_file should be an SAC file as the source time function when set Trapezoid_stf_mode to be .false..

Note

To read the data from the Stf_file file, keep the sampling rate of SAC data consistent with the data and GFs.

MPI and Chains_n

MPI_n

- CPU number

Chains_n

- A total of n Markov Chains are sampled on each core (CPU).

N

- Each Markov Chains' sampling number.

Misfit

Geometrical_spreading_mode

- Whether the geometric diffusion correction is turned on or not. Waveform normalization is used when it set to be .false., and please set the magnitude M0 not to be inverted in Fixed_FM. Only one of Geometrical_spreading_mode and Amplitude_normalization_mode can be .true..

Distance_0

- The reference epicentral distance when Geometrical_spreading_mode set to be .true..

Amplitude_normalization_mode

- Waveform normalization mode.Only one of Geometrical_spreading_mode and Amplitude_normalization_mode can be .true..

Use_file_mode

- When set it to be .false., don't need to provide Raw_data_inv_info file. Setting following parameters (Raw_data_inv_info_writed, NET_STA, P_inv_comp, S_inv_comp, Surf_inv_comp, P_win, S_win, Surf_win, Phase_weight, Distance_weight, P_maxlag, S_maxlag, Surf_maxlag, P_filter, S_filter, Surf_filter) correctly, MCMTpy will generate this file automatically, so you can change individual parameter values and use this file for inversion. Raw_data_inv_info_writed is the filename that generate this file automatically. When set it to be .true., Raw_data_inv_info is needed which is the path to inversion file.

Raw_data_inv_info

- The path to inversion parameters file. It includes the time window length, filtering range, weight and other parameters used for data. When set Use_file_mode to be .true., it must be provided.
- Raw_data_inv_info.txt format

```
 1 YN   EYA
 2 Lat_lon_depth       26.108801   99.947502   0.000000
 3 P_inv_comp       yes       yes       no
 4 S_inv_comp       no       no       no
 5 Surf_inv_comp       yes       yes       yes
 6 P_win              -10.00       3.40
 7 S_win              -5.00       25.20
 8 Surf_win             -12.00       50.20
 9 Phase_weight       1.0       1.0       1.0
10 Distance_weight       1.0
11 P_maxlag             5.0000
12 S_maxlag             15.0000
13 Surf_maxlag       10.0000
14 P_filter             0.010       0.20
15 S_filter             0.010       0.100
16 Surf_filter       0.005       0.100
17
18 YN   YUL
19 Lat_lon_depth       25.885401   99.371696   0.000000
20 P_inv_comp       yes       yes       no
21 S_inv_comp       no       no       no
22 Surf_inv_comp       yes       yes       yes
23 P_win              -10.00       6.20
24 S_win              -5.00       25.20
25 Surf_win             -12.00       50.20
26 Phase_weight       1.0       1.0       1.0
27 Distance_weight       1.0
28 P_maxlag             5.0000
29 S_maxlag             15.0000
30 Surf_maxlag       10.0000
31 P_filter             0.010       0.200
32 S_filter             0.010       0.100
33 Surf_filter       0.005       0.100
```

Raw_data_inv_info_writed

- Raw_data_inv_info_writed is the filename that generate this file automatically When set Use_file_mode to be .true..

NET_STA

- The information contained in the two-dimensional list: [network_name] [station_name] [station_lat] [source_lon] [station_depth]

P_inv_comp

- P wave Z/R/T component. ‘yes’: inversion. ‘no’: no inversion. Select the component you want to invert.

S_inv_comp

- S wave Z/R/T component. ‘yes’: inversion. ‘no’: no inversion. Select the component you want to invert.

Surf_inv_comp

- Surf wave Z/R/T component. ‘yes’: inversion. ‘no’: no inversion. Select the component you want to invert.

P_win

- P wave P_win=[-1,2] unit/s; The P wave fitting band, and ‘-1’ means 1 second before the P wave, ‘2’ means 2 second after the P wave

S_win

- S wave S_win…

Surf_win

- Surf wave Surf_win…

Phase_weight

- P/S/Surface wave inversion weight

Distance_weight

- The weight of each station in NET_STA

P_maxlag

- The maximum number of moveable sampling points to fitting waveform of P wave. It is generally half the period of the P phase waveform. unit/s.

S_maxlag

- The maximum number of moveable sampling points to fitting waveform of S wave.

Surf_maxlag

- The maximum number of moveable sampling points to fitting waveform of Surf wave.

P_filter

- P wave filter range, [freqmin, freqmax].

S_filter

- S wave filter range, [freqmin, freqmax].

Surf_filter

- Surf wave filter range, [freqmin, freqmax].

Inversion parameters

InvType

- The type of source, mt(moment tensor) dc(double-couple) sf(single-couple) ep(explosion).

FM0

- The initial value of the inversion, it’s a 1-d list.
- dc:[m0, str, dip rake, lat, lon, depth, t0] || mt:[m0, m_xx, m_xy, m_xz, m_yy, m_yz, m_zz, lat, lon, depth, t0] || ep:[m0]
- The default initial value of t0 is usually 0.

Fixed_FM

- The length of the list corresponds to FM0, and ‘constant’ means fixed to a constant and unchanged in the inversion process. ‘variable’ is represented as a variable and participates in the inversion.

FM_boundary

- A 2-d list, FM's boundary.

Noise_level_waveform

- Noise level estimation of raw data.

MISFIT0

- The initial error of the objective function, usually set to a very large value.

alpha_max

- The initial value of alpha.

alpha_min

- The final value of alpha.

a

- The alpha function's parameter.

b

- The alpha function's parameter.

sigma_mw

- The standard deviation of the new solution of Mw.

sigma_mt

- The standard deviation of the new solution of Moment Tensor.

sigma_dc

- The standard deviation of the new solution of Strike Dip Rake.

sigma_sf

- The standard deviation of the new solution of Single Couple (not test now).

sigma_ep

- The standard deviation of the new solution of Explosion (not test now).

sigma_lat_lon

- The standard deviation of the new solution of Latitude and Longitude (degree).

sigma_depth

- The standard deviation of the new solution of Source Depth (km).

sigma_t

- The standard deviation of the new solution of T0 (s).

Alpha

- The alpha function is shown in following script, you can test different parameters alpha_max alpha_min a b and plot thefigure.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import math
import numpy as np
import matplotlib.pyplot as plt

# parameters
N=5e3
a=1000              # 0.2*N
b=100               # 0.04*N
```

```
alpha_max=100
alpha_min=0

#-----------------------------------------------------------#
# we expect no parameters need to be changed below
#-----------------------------------------------------------#
tt=np.linspace(0, round(N), num=round(N))
yy=np.zeros(len(tt))
for i in range(0,len(tt)):
    yy[i]=(alpha_max-alpha_min)*(a+math.exp((tt[0]-a)/b))/(a+math.exp((tt[i]-a)/b))+alpha_min

fig, axs = plt.subplots(1,1)
axs.plot(tt,yy, linestyle='-', color='red', lw=3, alpha=0.6)
axs.set_xlabel("Sample Number",fontsize=17)
axs.set_ylabel("Alpha Value",fontsize=17)
fig.show()
```

Logging file

- MCMTpy will generates 1 logging files when it do Inversion, that is Inv_para_info.txt. They record all the parameter information of inversion process. You can find them in Output_path path.

Example Double-Couple Inv

- The example_path need to de changed to your path, and run this notebook to set sample_dc.json file.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import os
import sys
import json5


# The root directory of the project
example_path = '/Users/yf/3.Project/8.MCMTpy/MCMTpy-master/data/example_yunnan'


#-----------------------------------------------------------#
# we expect no parameters need to be changed below
#-----------------------------------------------------------#
# 1. get notebook path
notebook_path = sys.path[0]
os.chdir(notebook_path)
# change path to Green_Funcs
os.chdir('../YN.202105212148_Inv/dc_inv')

# 2.show build_GFs.json
filename = 'sample_dc.json'
with open(filename, 'r',encoding='utf-8') as f1:
    sample_dc_json=json5.load(f1)

# 3.change parameters with absolute path
sample_dc_json["GFs_json_file"] = os.path.join(example_path,"Green_Funcs/build_GFs_new.json")
sample_dc_json["GFs_file"] =os.path.join(example_path,"Green_Funcs/GFs/GFs_splits" )
sample_dc_json["Raw_data_file"] = os.path.join(example_path,"YN.202105212148_Inv/YN.202105212148_MCMTpy/YN.2
sample_dc_json["Output_path"] = os.path.join(example_path,"YN.202105212148_Inv/dc_inv/Output_YN.202105212148_d
sample_dc_json["Raw_data_inv_info"] = os.path.join(example_path,"YN.202105212148_Inv/dc_inv/Raw_data_inv_info.txt"
sample_dc_json["Raw_data_inv_info_writed"] = os.path.join(example_path,"YN.202105212148_Inv/dc_inv/Raw_data_inv_i
```

```
sample_dc_json_new = os.path.join(example_path,'YN.202105212148_Inv/dc_inv/sample_dc_new.json')

sample_dc_json["MPI_n"] = 4                    # CPU num
sample_dc_json["Chains_n"] = 4                 # Each MK-chains num
sample_dc_json["N"] = 1e1                      # each chain's search number

sample_dc_json["alpha_max"] = 100              # The initial value of alpha
sample_dc_json["alpha_min"] = 0                # The final value of alpha
sample_dc_json["a"] = 10
sample_dc_json["b"] = 10

with open(sample_dc_json_new,'w') as f2:
    json5.dump(sample_dc_json, f2, indent=2)
f2.close()

# 4.run MCMTpy in shell
os.chdir(notebook_path)
os.chdir('../YN.202105212148_Inv/dc_inv')
# !MCMTpy  sample MH  -c ./sample_dc_new.json #> sample.log
```

- Now you can run MCMTpy in bash:

```
$ MCMTpy  sample MH  -c ./sample_dc_new.json > sample.log
```

Note

Please follow the above parameter instructions and set all parameters correctly before running the program. Otherwise, it is easy to report an error!

Result Visualization

- This is the plotting script of the MCMTpy, and all parameters are written into a JSON file named plot.json. Some brief descriptions of the parameters are included here following the definination.
- We do not recommend using relative paths because of the possibility of errors. Absolute paths are preferred.

Path

plot_output_path,

- Plot result output file path.

Inv_json_file,

- The path to the Json-file used in inversion.

fig_format,

- Output figure format.

Hist

plot_hist

- Whether to plot hist-figure.

N_start

- The sequence number of all-FM Markov chain to plot in hist-figure.

N_start_accept

- The sequence number of all-accept-FM Markov chain to plot in hist-figure.

num_bins

- Number of grids to draw in hist-figure.

num_std

- Range of axes in each subgraph (mean +- several times standard deviation). num_std means n times standard deviation.

labels_name

- Labels' name, it's depend on the number parameters that you want to inverse.
- eg: dc–> ['mw','strike/°','dip/°','rake/°','x/km','y/km','z/km','t0/s'].
- eg: mt–> ['Mw', 'Mxx', 'Myy', 'Mzz', 'Mxy', 'Mxz', 'Myz', 'x/km', 'y/km', 'z/km', 't0/s'].

Misfit

plot_misfit

- Whether to plot misfit-figure.

MPI_n_st

- Which MPI's misfit do you want to draw in figure.

Chains_n_st

- Which Chains's misfit do you want to draw in figure.

Note

We can only specify a chain of an MPI to draw misfit-figure.

Waveform

plot_waveform

- Whether to plot waveform-figure.

NET_STA

- The information contained in the two-dimensional list: [network_name] [station_name] [station_lat] [source_lon] [station_depth]

FM_best

- The best focal mechanism, it's a 1-d list.
- dc:[m0, str, dip rake, lat, lon, depth, t0] || mt:[m0, m_xx, m_xy, m_xz, m_yy, m_yz, m_zz, lat, lon, depth, t0] || ep:[m0]

line_n_sta

- Draw several stations per row in waveform-figure.

max_p_ylim

- The maximum amplitude of P wave drawing in waveform-figure.

max_s_ylim

• The maximum amplitude of S wave drawing in waveform-figure.

max_surf_ylim

• The maximum amplitude of Surf wave drawing in waveform-figure.

plot_comp

• Which components of the waveform that you want to plot in waveform-figure.

• The 2-d list represent the three components of P, S and Surf, respectively ZRT.

plot_comp_name

• Labels' name, it's depend on the number parameters that you want to plot with plot_comp.

Example

• The example_path need to de changed to your path, and run this notebook to set sample_dc.json file.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import os
import sys
import json5

# The root directory of the project
example_path = '/Users/yf/3.Project/8.MCMTpy/MCMTpy-master/data/example_yunnan'

#-----------------------------------------------------------#
# we expect no parameters need to be changed below
#-----------------------------------------------------------#
# 1. get notebook path
notebook_path = sys.path[0]
os.chdir(notebook_path)

# 2.change path to Green_Funcs
os.chdir(notebook_path)
os.chdir('../YN.202105212148_Inv/dc_inv')
print(os.listdir())

# 3.read plot_dc.json
filename = 'plot_dc.json'
with open(filename, 'r',encoding='utf-8') as f1:
    plot_dc_json=json5.load(f1)

# 4.change parameters with absolute path
plot_dc_json["plot_output_path"] = os.path.join(example_path,"YN.202105212148_Inv/dc_inv/figure_dc")
plot_dc_json["Inv_json_file"] = os.path.join(example_path,"YN.202105212148_Inv/dc_inv/sample_dc_new.json")
plot_dc_json_new = os.path.join(example_path,'YN.202105212148_Inv/dc_inv/plot_dc_new.json')

# 5.hist
plot_dc_json["plot_hist"] =True
plot_dc_json["N_start"] = 0
plot_dc_json["N_start_accept"] = 1
plot_dc_json["num_bins"] =  50
plot_dc_json["num_std"] =5
plot_dc_json["labels_name"] = ['Mw','strike/°','dip/°','rake/°','z/km']

# 6. misfit
plot_dc_json["plot_misfit"] = True
plot_dc_json["MPI_n_st"] = 0
plot_dc_json["Chains_n_st"] = 0
```

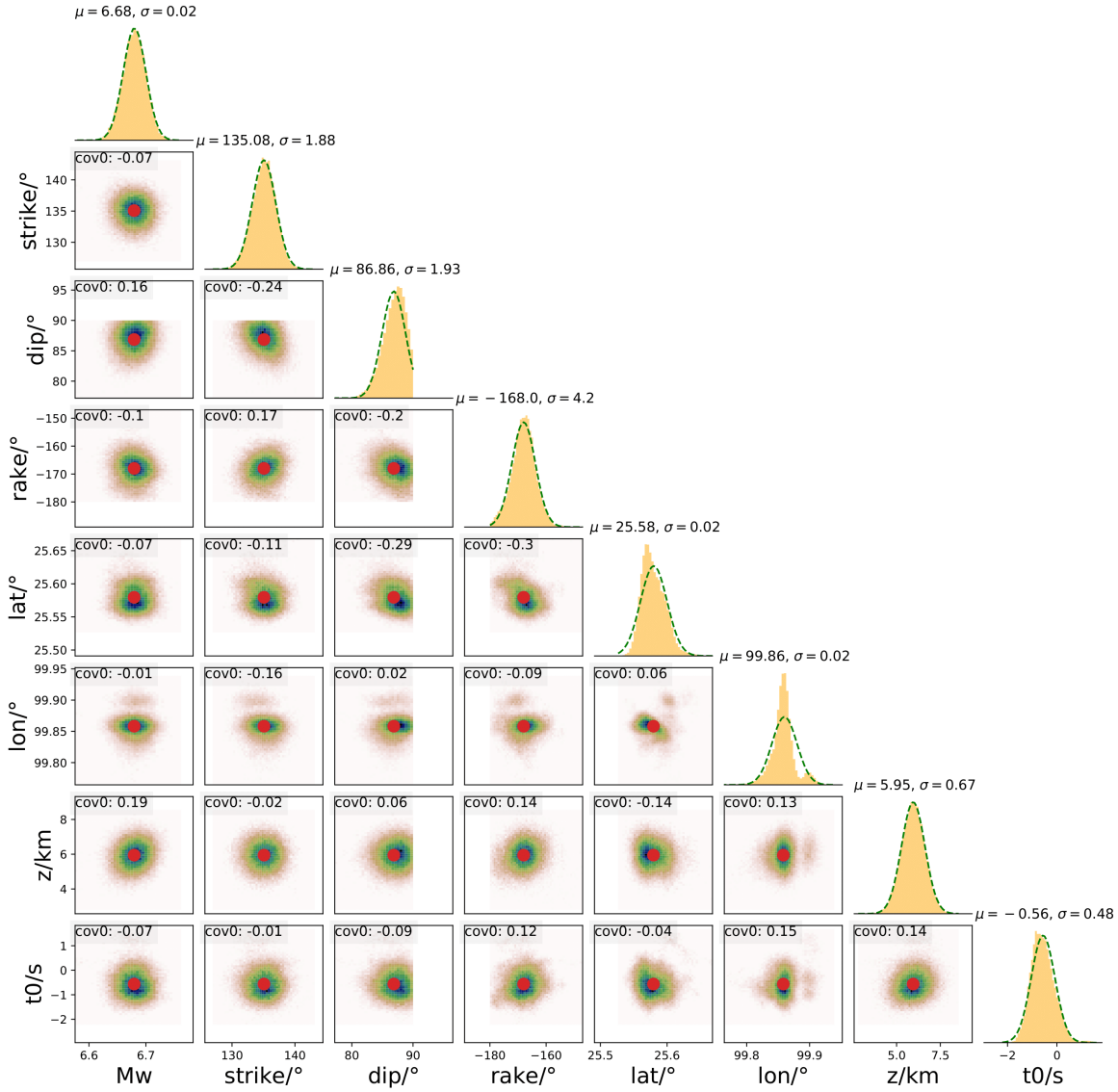
```
45
46 # 7. waveform
47 plot_dc_json["plot_waveform"] = True
48 plot_dc_json["FM_best"] = [6.68,  135.1,  87.0,  -168.1,    25.58,  99.86,  5.95,  -0.57 ]
49 plot_dc_json["line_n_sta"] = 3                          # Draw the data of several stations in a row
50 plot_dc_json["max_p_ylim"] = 1                           # max amp y-axis of p wave
51 plot_dc_json["max_s_ylim"] = 1.0
52 plot_dc_json["max_surf_ylim"] = 1
53 plot_dc_json["plot_comp"] = [[1,1,0],[0,0,0],[1,1,1]]          # What components do you want to draw? P、S、Surf's Z/R/T
54
55
56 with open(plot_dc_json_new,'w') as f2:
57     json5.dump(plot_dc_json, f2, indent=2)
58 f2.close()
59
60 # 8.run MCMTpy in shell
61 os.chdir(notebook_path)
62 os.chdir('../YN.202105212148_Inv/dc_inv')
63 # !MCMTpy plot pyfk -c plot_dc_new.json > plot.log
```

- Now you can run MCMTpy in bash:

```
$ MCMTpy plot pyfk -c plot_dc_new.json
```

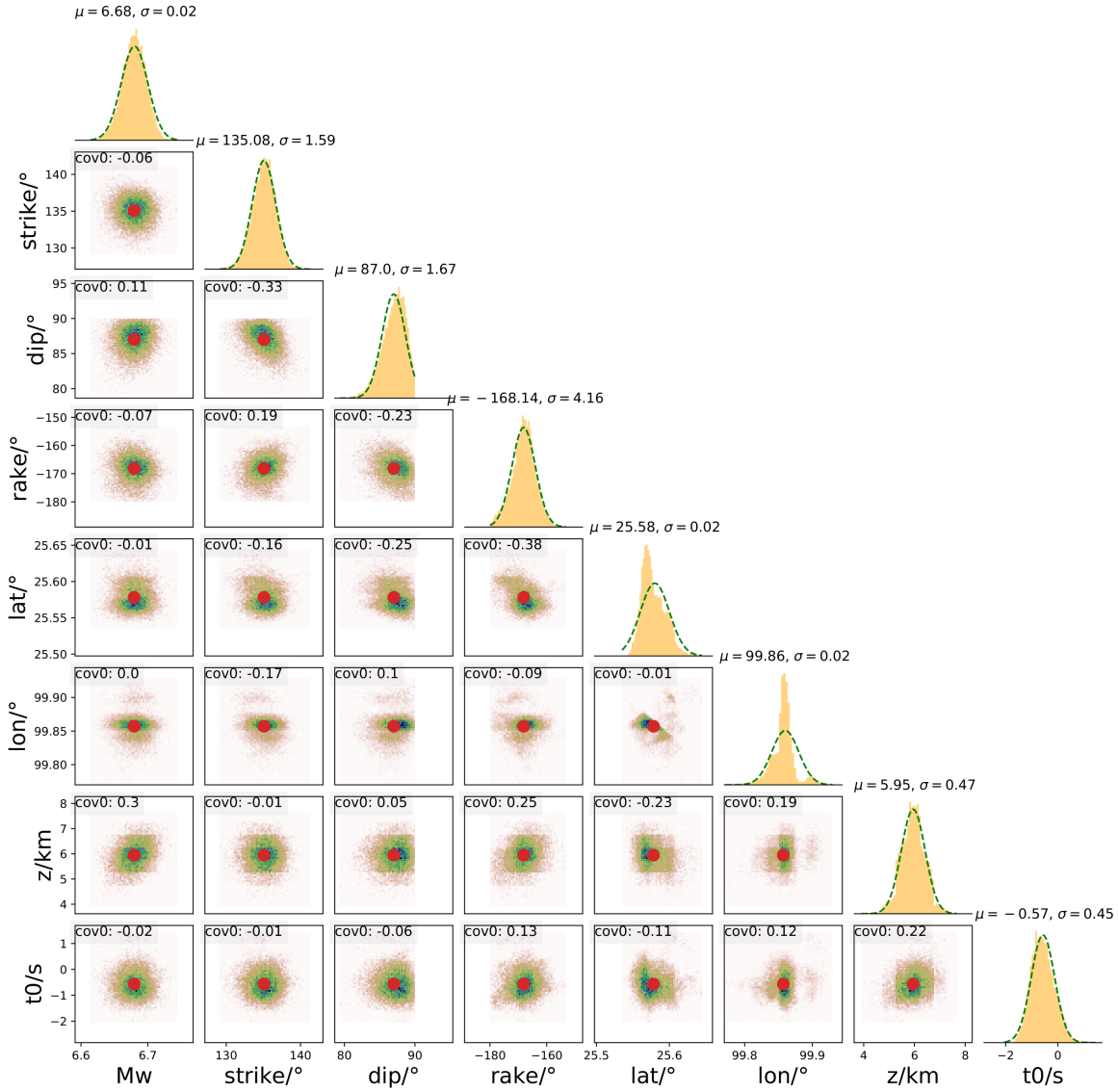
- Show hist in jupyter notebook:

```
fig_hist = os.path.join(plot_dc_json["plot_output_path"],'hist.jpg')
Image(filename = fig_hist, width=600)
```



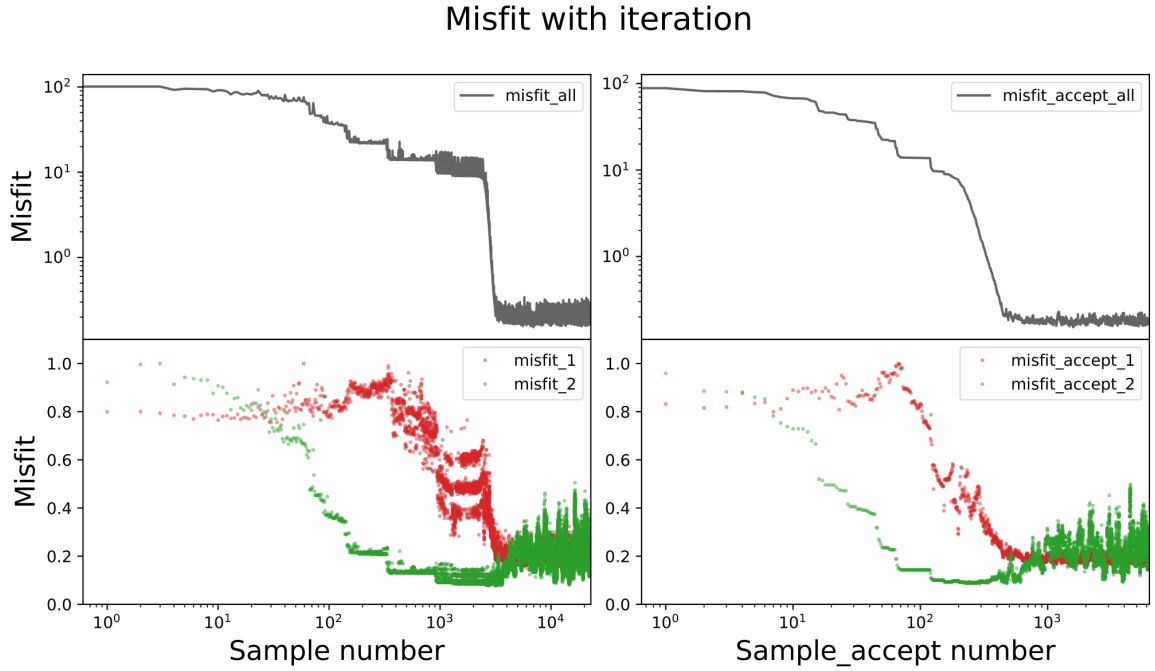
- Show hist_accept in jupyter notebook:

```
fig_hist_accept = os.path.join(plot_dc_json["plot_output_path"],'hist_accept.jpg')
Image(filename = fig_hist_accept, width=600)
```



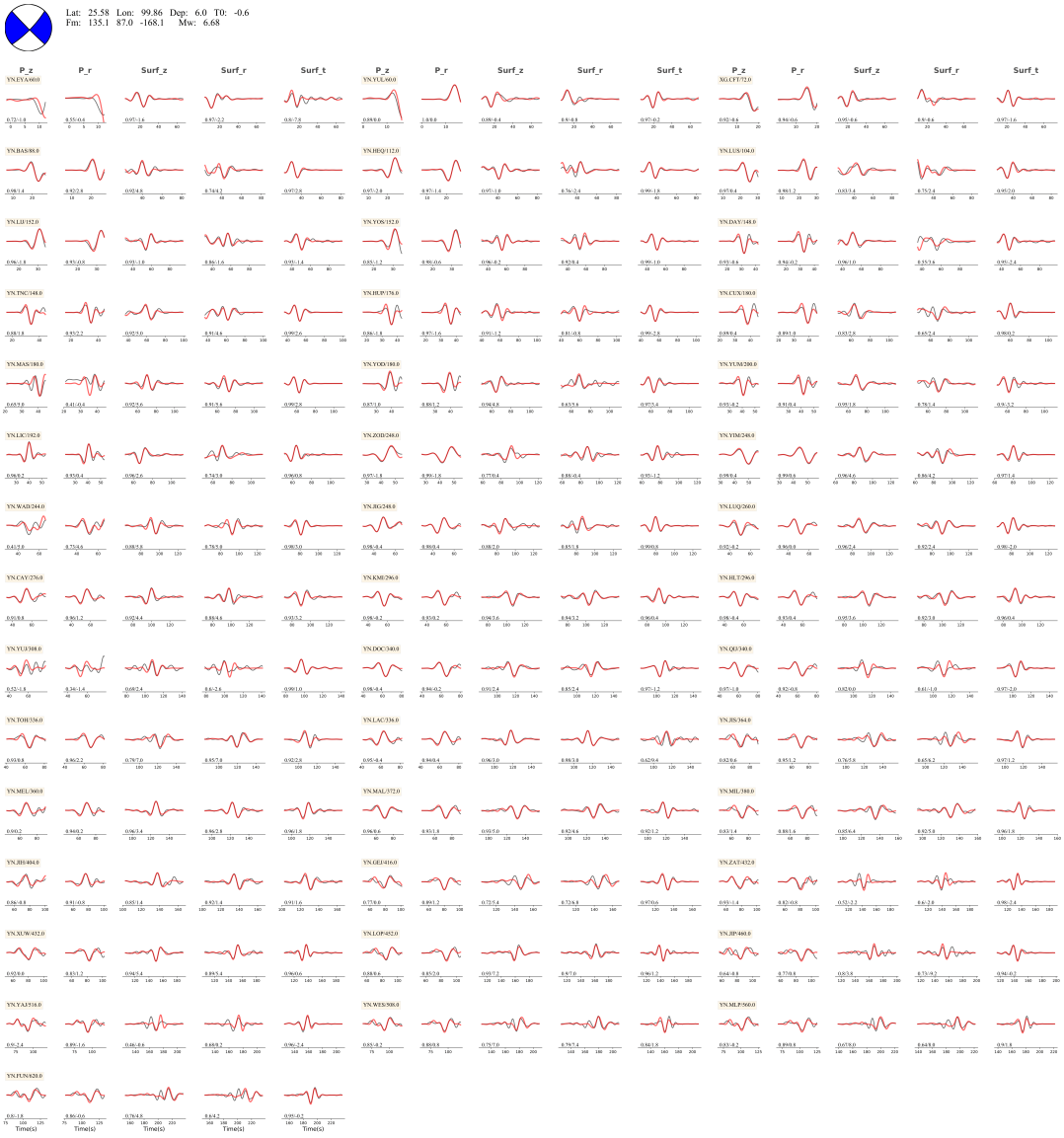
- Show misfit in jupyter notebook:

```
fig_misfit = os.path.join(plot_dc_json["plot_output_path"],'misfit.jpg')
Image(filename = fig_misfit, width=600)
```



- Show waveform in jupyter notebook:

```
fig_waveform = os.path.join(plot_dc_json["plot_output_path"],'waveform.jpg')
Image(filename = fig_waveform, width=1000)
```



Note

Please follow the above parameter instructions and set all parameters correctly before running the program. Otherwise, it is easy to report an error!

Conversion of Source Parameters

- The MCMTpy provides a series of scripts for source parameter calculation and conversion. The MomentTensor script refers to some theories and codes of Obspy MoPaD and Introduction to Seismology (Yongge Wan), mainly including:

- We do not recommend using relative paths because of the possibility of errors. Absolute paths are preferred.

The installation

- jupyter notebook:

```
# import MomentTensor
from MCMTpy import MomentTensor as MTpy

# get __doc__
MTpy.__doc__.strip().split("\n")
```

str_dip_rake to mt

- jupyter notebook:

```
# function: str_dip_rake2MT
MTpy.str_dip_rake2MT.__doc__.strip().split("\n")

# input
strike = 50
dip = 50
rake = 100

A = MTpy.str_dip_rake2MT(strike,dip,rake)
A.mt
```

The conversion between str_dip_rake and A/N vector

- jupyter notebook:

```
#-----------------------------------------------#
# function: str_dip_rake2AN
MTpy.str_dip_rake2AN.__doc__.strip().split("\n")

# str_dip_rake to A/N vector
# input
strike = 50
dip = 50
rake = 100

A,N = MTpy.str_dip_rake2AN(strike,dip,rake)
print('A = ', A)
print('N = ', N)

#-----------------------------------------------#
# function AN2str_dip_rake
MTpy.AN2str_dip_rake.__doc__.strip().split("\n")
```

```
# A/N vector to str_dip_rake
# input
A = np.array([0.37330426, -0.53992106, -0.75440651])
N = np.array([-0.58682409,  0.49240388, -0.64278761])

DD = MTpy.AN2str_dip_rake(A,N)
print('strike = ', DD.strike)
print('dip = ', DD.dip)
print('rake = ', DD.rake)
```

The conversion between A/N vector and P/T/N vector

- jupyter notebook:

```
#-----------------------------------------------#
# function AN2TPN
MTpy.AN2TPN.__doc__.strip().split("\n")

# Calculate the T-axis, P-axis and N-axis according to the slip vector (A) and fault plane direction vector (N)
# input
A = np.array([0.37330426, -0.53992106, -0.75440651])
N = np.array([-0.58682409,  0.49240388, -0.64278761])

T,P,Null = MTpy.AN2TPN(A,N)
print('T = ', T)
print('P = ', P)
print('Null = ', Null)

#-----------------------------------------------#
# function TP2AN
MTpy.TP2AN.__doc__.strip().split("\n")

# Calculate the slip vector (A) and fault plane direction vector (N) according to the T-axis and P-axis
# input
T = np.array([-0.15098132, -0.03359972, -0.98796544])
P = np.array([0.67891327, -0.72996397, -0.07892648])

A,N = MTpy.TP2AN(T,P)
print('A = ', A)
print('N = ', N)
```

mt to P/T/N vector

- jupyter notebook:

```
# function MT2TPN
MTpy.MT2TPN.__doc__.strip().split("\n")

# input
strike = 50
dip = 50
rake = 100

A = MTpy.str_dip_rake2MT(strike,dip,rake)
T, P, Null = MTpy.MT2TPN(A)
print('T = ', T)
print('P = ', P)
print('Null = ', Null)
```

P/T/N vector to P/T/N vector's stirke and dip

- jupyter notebook:

```
# function vector2str_dip
MTpy.vector2str_dip.__doc__.strip().split("\n")

# input
A = np.array([0.37330426, -0.53992106, -0.75440651])

CC = MTpy.vector2str_dip(A)
print('A.strike = ', CC.strike)
print('A.dip = ', CC.dip)
```

Describe_fault_plane with two str_dip_rake

- jupyter notebook:

```
# function describe_fault_plane
MTpy.describe_fault_plane.__doc__.strip().split("\n")

# input
strike = 50
dip = 50
rake = 100

A = MTpy.str_dip_rake2MT(strike,dip,rake)
CC = MTpy.describe_fault_plane(A.mt)
print('FM_1 = ', CC[0,:])
print('FM_2 = ', CC[1,:])
```

Plot Beachball with Station Projected

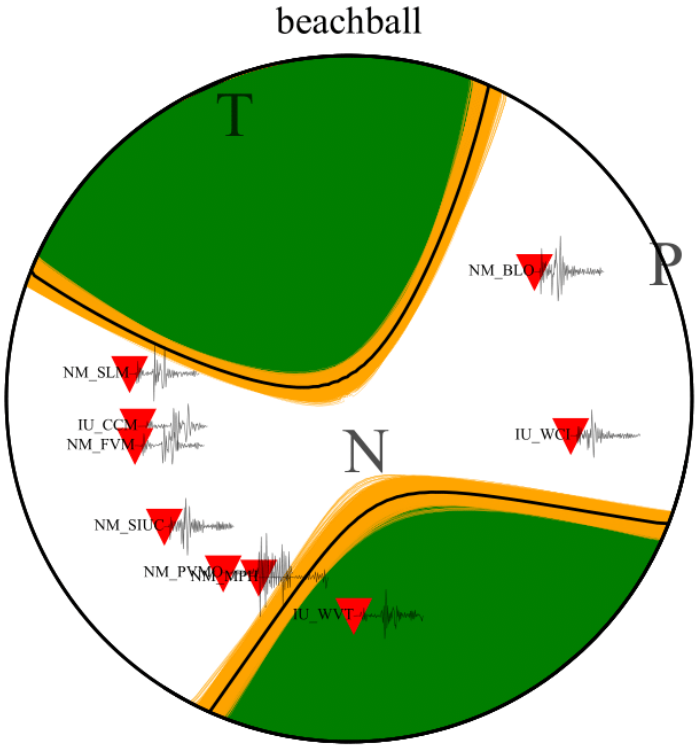
- The example path need to de changed to your path, and run this notebook.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import os
import glob
import obspy
import numpy as np
import matplotlib.pyplot as plt
from obspy import Stream
from obspy.taup import TauPyModel
from obspy.imaging.mopad_wrapper import beach
from MCMTpy import MomentTensor as MTpy

# Helper function
#-----------------------------------------------#
#%% 1.read raw data
def read_data(allfiles_path):

    allfiles = sorted( glob.glob(allfiles_path) )
    data_raw = Stream()
    for i in range(0,len(allfiles),1):
        try:
            tr = obspy.read(allfiles[i])
            data_raw += tr
        except Exception:
            print(allfiles[i],': no such file or obspy read error');continue
```

```
    data = data_raw.copy()

    return data

#-----------------------------------------------------#
#%% 2.taup ray trace
def get_taup_tp_ts(model,depth,distance,degree=None):
    if degree==False:
        distance = distance/111.19

    time_p = model.get_travel_times(source_depth_in_km=depth,
                                    distance_in_degree=distance,
                                    phase_list=["p", "P"])

    time_s = model.get_travel_times(source_depth_in_km=depth,
                                    distance_in_degree=distance,
                                    phase_list=["s", "S"])

    ray_p = time_p[0].ray_param
    tp = time_p[0].time
    angle_p = time_p[0].incident_angle

    ray_s = time_s[0].ray_param
    ts = time_s[0].time
    angle_s = time_s[0].incident_angle

    return ray_p,tp,angle_p,ray_s,ts,angle_s


#%%-----------------------------------------------------------#
# 0. set path
example_path = '/Users/yf/3.Project/8.MCMTpy/MCMTpy-master/data/example_yunnan'


#-----------------------------------------------------------#
# we expect no parameters need to be changed below
#-----------------------------------------------------------#
FM_path=os.path.join(example_path,"YN.202105212148_Inv/dc_inv/Output_YN.202105212148_dc/rank_0_output/chain_
allfiles_path = os.path.join(example_path,'YN.202105212148_Inv/YN.202105212148_raw/*.SAC')

## 1. read FM
N=3 # Three parameters are required to describe the focal mechanism
FM_all = np.loadtxt(FM_path)
FM_raw = FM_all[0:,1:N+1] # Define the number of solutions you want to plot
strike_np = FM_raw[:,0]
dip_np = FM_raw[:,1]
rake_np = FM_raw[:,2]
FM = np.vstack((strike_np, dip_np,  rake_np)).T
FM_mean=np.zeros(shape=(N))
for i in range(0,N,1):
    FM_mean[i]=np.mean(FM[0:,i])

## 2.read raw data
data = read_data(allfiles_path)
data.filter('bandpass', freqmin=0.005, freqmax=0.5, corners=4, zerophase=True)

## 3.ray trace with taup
model_path = os.path.join(example_path,"v_model/v_model.npz")
model = TauPyModel(model=model_path)    # "iasp91"  "prem"
for i in range(0,len(data),1):
```

```
    depth = data[i].stats.sac['evdp']
    distance = data[i].stats.sac['dist']
    ray_p,tp,angle_p,ray_s,ts,angle_s = get_taup_tp_ts(model,depth,distance,degree=False)
    data[i].stats.sac["user1"]=angle_p
    data[i].stats.sac["user2"]=angle_s

## 4.1 plot FM_mean
ax0 = plt.gca()
Length_Ball = 100
beach1 = beach(FM_mean, xy=(50, 50),  linewidth=1,width=Length_Ball-1, alpha=1,\
             facecolor='g',bgcolor='w', edgecolor='k',mopad_basis='NED',nofill=False,zorder=1 )
ax0.add_collection(beach1)
ax0.set_aspect("equal")

## 4.2 plot FM_all
for i in range(0,FM.shape[0],1):
    beach1 = beach(FM[i,:], xy=(50, 50),  linewidth=1,width=Length_Ball-1, alpha=1,\
                facecolor='b',bgcolor='w', edgecolor='orange',mopad_basis='NED',nofill=True,zorder=1 )
    ax0.add_collection(beach1)
    ax0.set_aspect("equal")

## 4.3 plot backgroud line
beach1 = beach(FM_mean, xy=(50, 50),  linewidth=1,width=Length_Ball-1, alpha=1,\
                facecolor='w',bgcolor='w', edgecolor='k',mopad_basis='NED',nofill=True,zorder=1 )
ax0.add_collection(beach1)
ax0.set_aspect("equal")

## 5.plot station and waveform
menthod='schmidt'   # 'schmidt' 'wulff'
for i in range(0,len(data),3):
    AZM = data[i].stats.sac['az']
    TKO = data[i].stats.sac['user1']
    net_sta_name = data[i].stats.network+'_'+data[i].stats.station
    X, Y = MTpy.project_beachball(AZM, TKO, R=Length_Ball/2, menthod=menthod)
    tt=np.linspace(X, X+10, num=len(data[i].data))
    ax0.plot(X, Y, "rv", ms=10,zorder=1)
    ax0.plot(tt, 5*data[i].data/2000000 + Y,  color='black',lw=0.2,alpha=0.6,zorder=1)
    ax0.text(X, Y,net_sta_name,horizontalalignment='right', verticalalignment='center',\
         fontsize=5, color='black',bbox = dict(facecolor = "r", alpha = 0.0),zorder=1)

## 6. plot P/T/N axis
MT = MTpy.str_dip_rake2MT(strike=FM_mean[0],dip=FM_mean[1],rake=FM_mean[2])
T_axis, P_axis, N_axis = MTpy.MT2TPN(MT)
T = MTpy.vector2str_dip(T_axis)
P = MTpy.vector2str_dip(P_axis)
N = MTpy.vector2str_dip(N_axis)

Tx, Ty = MTpy.project_beachball(AZM=T.strike, TKO=(90-T.dip), R=Length_Ball/2, menthod=menthod)
ax0.text(Tx,Ty,'T',horizontalalignment='center', verticalalignment='center',\
         fontsize=20, color='k',alpha=0.7,zorder=1)

Px, Py = MTpy.project_beachball(AZM=P.strike, TKO=(90-P.dip), R=Length_Ball/2, menthod=menthod)
ax0.text(Px,Py,'P',horizontalalignment='center', verticalalignment='center',\
         fontsize=20, color='k',alpha=0.7,zorder=1)

Nx, Ny = MTpy.project_beachball(AZM=N.strike, TKO=(90-N.dip), R=Length_Ball/2, menthod=menthod)
ax0.text(Nx,Ny,'N',horizontalalignment='center', verticalalignment='center',\
         fontsize=20, color='k',alpha=0.7,zorder=1)


```

```
146 ## 7. save figure
147 ax0.set_xlim(0,100)
148 ax0.set_ylim(0,100)
149 ax0.set_axis_off()
150 figurename=os.path.join('./S2_figure/beachball.pdf')
151 plt.savefig(figurename,dpi=800, format="pdf")
```



Moment Tensor Decompose

- The example path need to de changed to your path, and run this notebook.

```
 1 #!/usr/bin/env python3
 2 # -*- coding: utf-8 -*-
 3
 4 import os
 5 import glob
 6 import obspy
 7 import numpy as np
 8 import matplotlib.pyplot as plt
 9 from obspy import Stream
10 from obspy.taup import TauPyModel
11 from obspy.imaging.mopad_wrapper import beach
12 from MCMTpy import MomentTensor as MTpy
13
14 # Helper function
15 #----------------------------------------------------#
16 #%% 1.plot decompose mt
17 def plot_decompose(FM,FM_DC,FM_CLVD,FM_iso):
18     MT = MTpy.MTensor(FM)
19     Dec = MTpy.Decompose(MT)
```

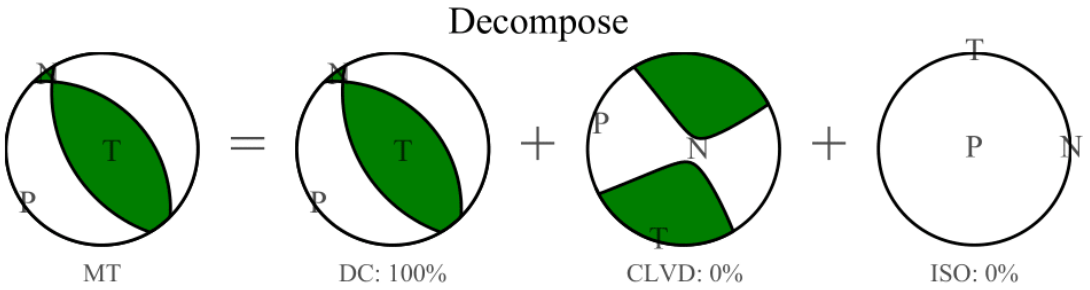
```
Dec.decomposition_iso_DC_CLVD()

fig2, ax2 = plt.subplots(1 ,1,  dpi=800)
Length_Ball = 100

###### MT
beach1 = beach(FM, xy=(50, 50),  linewidth=1,width=Length_Ball-1, alpha=1,\
            facecolor='g',bgcolor='w', edgecolor='k',mopad_basis='NED',nofill=False,zorder=1 )
ax2.add_collection(beach1)
ax2.set_aspect("equal")

menthod='schmidt'   # 'schmidt'  # 'wulff'
T_axis, P_axis, N_axis = MTpy.MT2TPN(MTpy.MTensor(FM))
T = MTpy.vector2str_dip(T_axis)
P = MTpy.vector2str_dip(P_axis)
N = MTpy.vector2str_dip(N_axis)

Tx, Ty = MTpy.project_beachball(AZM=T.strike, TKO=(90-T.dip), R=Length_Ball/2, menthod=menthod)
ax2.text(Tx,Ty,'T',horizontalalignment='center', verticalalignment='center',\
        fontsize=10, color='k',alpha=0.7,zorder=1)

Px, Py = MTpy.project_beachball(AZM=P.strike, TKO=(90-P.dip), R=Length_Ball/2, menthod=menthod)
ax2.text(Px,Py,'P',horizontalalignment='center', verticalalignment='center',\
        fontsize=10, color='k',alpha=0.7,zorder=1)

Nx, Ny = MTpy.project_beachball(AZM=N.strike, TKO=(90-N.dip), R=Length_Ball/2, menthod=menthod)
ax2.text(Nx,Ny,'N',horizontalalignment='center', verticalalignment='center',\
        fontsize=10, color='k',alpha=0.7,zorder=1)

###### DC
beach2 = beach(FM_DC, xy=(200, 50),  linewidth=1,width=Length_Ball-1, alpha=1,\
            facecolor='g',bgcolor='w', edgecolor='k',mopad_basis='NED',nofill=False,zorder=1 )
ax2.add_collection(beach2)
ax2.set_aspect("equal")

menthod='schmidt'   # 'schmidt'  # 'wulff'
T_axis, P_axis, N_axis = MTpy.MT2TPN(MTpy.MTensor(FM_DC))
T = MTpy.vector2str_dip(T_axis)
P = MTpy.vector2str_dip(P_axis)
N = MTpy.vector2str_dip(N_axis)

Tx, Ty = MTpy.project_beachball(AZM=T.strike, TKO=(90-T.dip), R=Length_Ball/2, menthod=menthod)
ax2.text(150*1+Tx,Ty,'T',horizontalalignment='center', verticalalignment='center',\
        fontsize=10, color='k',alpha=0.7,zorder=1)

Px, Py = MTpy.project_beachball(AZM=P.strike, TKO=(90-P.dip), R=Length_Ball/2, menthod=menthod)
ax2.text(150*1+Px,Py,'P',horizontalalignment='center', verticalalignment='center',\
        fontsize=10, color='k',alpha=0.7,zorder=1)

Nx, Ny = MTpy.project_beachball(AZM=N.strike, TKO=(90-N.dip), R=Length_Ball/2, menthod=menthod)
ax2.text(150*1+Nx,Ny,'N',horizontalalignment='center', verticalalignment='center',\
        fontsize=10, color='k',alpha=0.7,zorder=1)

####### CLVD
beach3 = beach(FM_CLVD, xy=(350, 50),  linewidth=1,width=Length_Ball-1, alpha=1,\
            facecolor='g',bgcolor='w', edgecolor='k',mopad_basis='NED',nofill=False,zorder=1 )
ax2.add_collection(beach3)
ax2.set_aspect("equal")

menthod='schmidt'   # 'schmidt'  # 'wulff'
```

```
T_axis, P_axis, N_axis = MTpy.MT2TPN(MTpy.MTensor(FM_CLVD))
T = MTpy.vector2str_dip(T_axis)
P = MTpy.vector2str_dip(P_axis)
N = MTpy.vector2str_dip(N_axis)

Tx, Ty = MTpy.project_beachball(AZM=T.strike, TKO=(90-T.dip), R=Length_Ball/2, menthod=menthod)
ax2.text(150*2+Tx,Ty,'T',horizontalalignment='center', verticalalignment='center',\
        fontsize=10, color='k',alpha=0.7,zorder=1)

Px, Py = MTpy.project_beachball(AZM=P.strike, TKO=(90-P.dip), R=Length_Ball/2, menthod=menthod)
ax2.text(150*2+Px,Py,'P',horizontalalignment='center', verticalalignment='center',\
        fontsize=10, color='k',alpha=0.7,zorder=1)

Nx, Ny = MTpy.project_beachball(AZM=N.strike, TKO=(90-N.dip), R=Length_Ball/2, menthod=menthod)
ax2.text(150*2+Nx,Ny,'N',horizontalalignment='center', verticalalignment='center',\
        fontsize=10, color='k',alpha=0.7,zorder=1)

######## iso
beach4 = beach(FM_iso, xy=(500, 50),  linewidth=1,width=Length_Ball-1, alpha=1,\
            facecolor='g',bgcolor='w', edgecolor='k',mopad_basis='NED',nofill=False,zorder=1 )
ax2.add_collection(beach4)
ax2.set_aspect("equal")

menthod='schmidt'   # 'schmidt'  # 'wulff'
T_axis, P_axis, N_axis = MTpy.MT2TPN(MTpy.MTensor(FM_iso))
T = MTpy.vector2str_dip(T_axis)
P = MTpy.vector2str_dip(P_axis)
N = MTpy.vector2str_dip(N_axis)

Tx, Ty = MTpy.project_beachball(AZM=T.strike, TKO=(90-T.dip), R=Length_Ball/2, menthod=menthod)
ax2.text(150*3+Tx,Ty,'T',horizontalalignment='center', verticalalignment='center',\
        fontsize=10, color='k',alpha=0.7,zorder=1)

Px, Py = MTpy.project_beachball(AZM=P.strike, TKO=(90-P.dip), R=Length_Ball/2, menthod=menthod)
ax2.text(150*3+Px,Py,'P',horizontalalignment='center', verticalalignment='center',\
        fontsize=10, color='k',alpha=0.7,zorder=1)

Nx, Ny = MTpy.project_beachball(AZM=N.strike, TKO=(90-N.dip), R=Length_Ball/2, menthod=menthod)
ax2.text(150*3+Nx,Ny,'N',horizontalalignment='center', verticalalignment='center',\
        fontsize=10, color='k',alpha=0.7,zorder=1)

######## plot '+' and '='
ax2.text(125,50,'=',horizontalalignment='center', verticalalignment='center',\
        fontsize=20, color='k',alpha=0.7,zorder=1)
ax2.text(275,50,'+',horizontalalignment='center', verticalalignment='center',\
        fontsize=20, color='k',alpha=0.7,zorder=1)
ax2.text(425,50,'+',horizontalalignment='center', verticalalignment='center',\
        fontsize=20, color='k',alpha=0.7,zorder=1)

######## plot percentage
MT_text = 'MT'
ax2.text(50,-15,MT_text,horizontalalignment='center', verticalalignment='center',\
        fontsize=8, color='k',alpha=0.7,zorder=1)

iso_text = 'ISO: '+str(round(Dec.M_iso_percentage,2))+'%'
ax2.text(500,-15,iso_text,horizontalalignment='center', verticalalignment='center',\
        fontsize=8, color='k',alpha=0.7,zorder=1)

DC_text = 'DC: '+str(round(Dec.M_DC_percentage,2))+'%'
ax2.text(200,-15,DC_text,horizontalalignment='center', verticalalignment='center',\
```

```
            fontsize=8, color='k',alpha=0.7,zorder=1)

    CLVD_text = 'CLVD: '+str(round(Dec.M_CLVD_percentage,2))+'%'
    ax2.text(350,-15,CLVD_text,horizontalalignment='center', verticalalignment='center',\
            fontsize=8, color='k',alpha=0.7,zorder=1)

    ######## set figure
    plt.title("Decompose")
    ax2.set_xlim(0,550)
    ax2.set_ylim(-30,100)
    ax2.set_axis_off()

    return fig2

#%%-----------------------------------------------------------#
# 1.set FM
# [Mxx_np,  Myy_np,  Mzz_np,  Mxy_np,  Mxz_np,  Myz_np] or [strike,dip,rake]
FM=[150,50,100]


#-----------------------------------------------------------#
# we expect no parameters need to be changed below
#-----------------------------------------------------------#
MT = MTpy.MTensor(FM)
Dec = MTpy.Decompose(MT)
Dec.decomposition_iso_DC_CLVD()   # Dec.help()
print(Dec.help())
print('\n\n*********************************\n\n')
Dec.print_self()

Mxx = Dec.M_iso[0,0]
Mxy = Dec.M_iso[0,1]
Mxz = Dec.M_iso[0,2]
Myy = Dec.M_iso[1,1]
Myz = Dec.M_iso[1,2]
Mzz = Dec.M_iso[2,2]
FM_iso = np.array((Mxx,  Myy,  Mzz,  Mxy,  Mxz,  Myz))

Mxx = Dec.M_DC[0,0]
Mxy = Dec.M_DC[0,1]
Mxz = Dec.M_DC[0,2]
Myy = Dec.M_DC[1,1]
Myz = Dec.M_DC[1,2]
Mzz = Dec.M_DC[2,2]
FM_DC = np.array((Mxx,  Myy,  Mzz,  Mxy,  Mxz,  Myz))

Mxx = Dec.M_CLVD[0,0]
Mxy = Dec.M_CLVD[0,1]
Mxz = Dec.M_CLVD[0,2]
Myy = Dec.M_CLVD[1,1]
Myz = Dec.M_CLVD[1,2]
Mzz = Dec.M_CLVD[2,2]
FM_CLVD = np.array((Mxx,  Myy,  Mzz,  Mxy,  Mxz,  Myz))

fig = plot_decompose(FM, FM_DC, FM_CLVD, FM_iso)
figurename=os.path.join('./S2_figure/Decompose.pdf')
plt.savefig(figurename,dpi=800, format="pdf")
```

- Decompose reuslt.

```
Incluing function:
self.M
self.M_iso
self.M_devi
self.M_DC
self.M_CLVD
self.M0
self.Mw
self.M_iso_percentage
self.M_DC_percentage
self.M_CLVD_percentage
self.eigen_val
self.eigen_vec
self.F
None


**********

self.M: [[-0.3576622  -0.48646688 -0.01115976]
 [-0.48646688 -0.61218411  0.20390851]
 [-0.01115976  0.20390851  0.96984631]]

self.M_iso: [[ -3.70074342e-17   0.00000000e+00   0.00000000e+00]
 [  0.00000000e+00  -3.70074342e-17   0.00000000e+00]
 [  0.00000000e+00   0.00000000e+00  -3.70074342e-17]]

self.M_devi: [[-0.3576622  -0.48646688 -0.01115976]
 [-0.48646688 -0.61218411  0.20390851]
 [-0.01115976  0.20390851  0.96984631]]

self.M_DC: [[-0.3576622  -0.48646688 -0.01115976]
 [-0.48646688 -0.61218411  0.20390851]
 [-0.01115976  0.20390851  0.96984631]]

self.M_CLVD: [[  3.33066907e-16   1.66533454e-16  -2.77555756e-17]
 [  1.66533454e-16  -2.22044605e-16   5.55111512e-17]
 [ -2.77555756e-17   5.55111512e-17   0.00000000e+00]]

self.M0: 1.0

self.Mw: -6.06666666667

self.M_iso_percentage: 0

self.M_DC_percentage: 100

self.M_CLVD_percentage: 0

self.eigen_val: [ -1.00000000e+00  -5.46437895e-17   1.00000000e+00]

self.eigen_vec: [[-0.60098212 -0.79705908 -0.0593069 ]
 [-0.79535596  0.58906868  0.14285304]
 [ 0.07892648 -0.13302222  0.98796543]]

self.F: -1.76363553391e-17
```

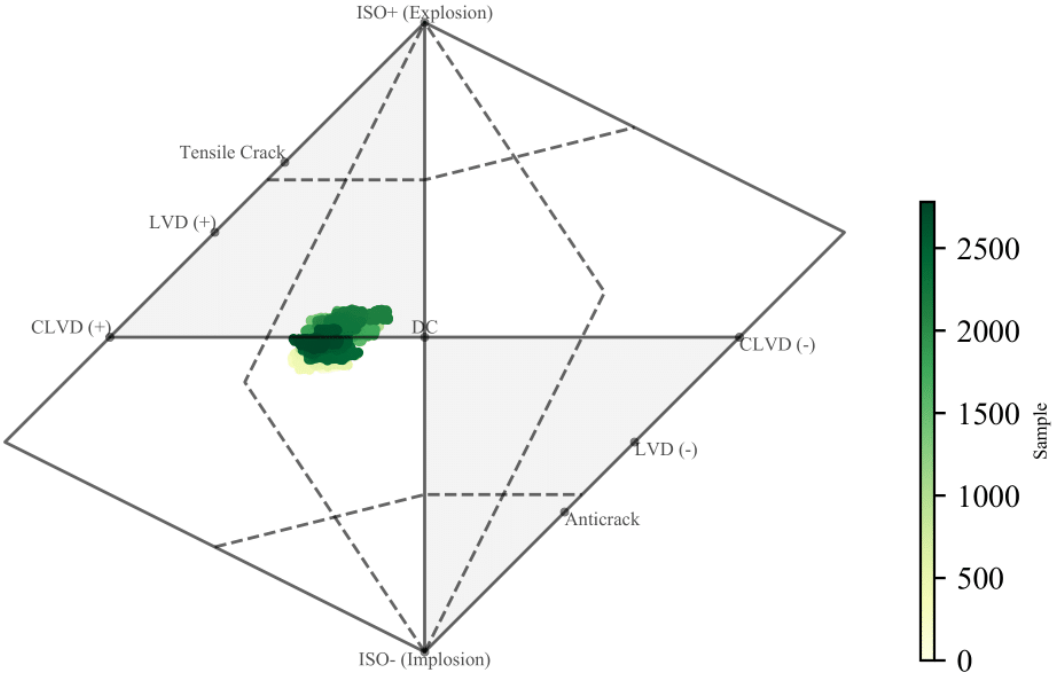


Huston Plot

- The example path need to de changed to your path, and run this notebook.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import os
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from MCMTpy import MomentTensor as MTpy

# Helper function
#-----------------------------------------------#
#%% 4. plot_Hudson_points
def plot_Hudson_points(FM):
    fig1, ax1 = plt.subplots(1 ,1,  dpi=800)
    plt.rc('font',family='Times New Roman')
    MTpy.Hudson_plot(ax=ax1)
    for i in range(0,len(FM)):
        MT = MTpy.MTensor(FM[i,:])
        M=MT.mt

        k,T = MTpy.M2kT_space(M)
        U,V = MTpy.kT2UV_space(k,T)

        map_vir = cm.get_cmap(name='YlGn')
        colors = map_vir(i/len(FM))

        ax1.scatter(U,V, color=colors,marker='o', s=5)

```

```
28
29      position=fig1.add_axes([0.85, 0.15, 0.01, 0.5])
30      font_colorbar = {'family' : 'Times New Roman',
31              'color'  : 'black',
32              'weight' : 'normal',
33              'size'   : 6,
34              }
35      sm = cm.ScalarMappable(cmap=map_vir)
36      sm.set_array(np.arange(0,len(FM)+1))
37      cb=plt.colorbar(sm,cax=position,orientation='vertical')
38      cb.set_label('Sample',fontdict=font_colorbar)
39
40      ax1.set_xlim(-4/3-0.1, 4/3+0.3)
41      ax1.set_ylim(-1-0.1, 1+0.1)
42
43      return fig1
44
45
46  #%%-----------------------------------------------------------#
47  ## 0. set path
48  example_path = '/Users/yf/3.Project/8.MCMTpy/MCMTpy-master/data/example_yunnan'
49
50
51  #-----------------------------------------------------------#
52  # we expect no parameters need to be changed below
53  #-----------------------------------------------------------#
54  FM_path=os.path.join(example_path,"YN.202105212148_Inv/mt_inv/Output_YN.202105212148_mt/rank_0_output/chain_
55  allfiles_path = os.path.join(example_path,'YN.202105212148_Inv/YN.202105212148_raw/*.SAC')
56  ## 1. read FM
57  N=6
58  FM_all = np.loadtxt(FM_path)
59  FM_raw = FM_all[0:,1:N+1]
60  Mxx_np = FM_raw[:,0]
61  Mxy_np = FM_raw[:,1]
62  Mxz_np = FM_raw[:,2]
63  Myy_np = FM_raw[:,3]
64  Myz_np = FM_raw[:,4]
65  Mzz_np = FM_raw[:,5]
66  FM = np.vstack((Mxx_np,  Myy_np,  Mzz_np,  Mxy_np,  Mxz_np,  Myz_np)).T
67
68  ## 2. plot Hudson
69  fig = plot_Hudson_points(FM)
70  figurename=os.path.join('./S2_figure/Hudson.pdf')
71  plt.savefig(figurename,dpi=800, format="pdf")
```



References

ASDF

The pyasdf Github repository https://github.com/SeismicData/pyasdf.

- Krischer, L., Smith, J., Lei, W., Lefebvre, M., Ruan, Y., de Andrade, E.S., Podhorszki, N., Bozdağ, E. and Tromp, J., 2016. An adaptable seismic data format. Geophysical Supplements to the Monthly Notices of the Royal Astronomical Society, 207(2), 1003-1011.

pyfk

The pyfk Github repository https://github.com/ziyixi/pyfk.

NoisePy

The NoisePy Github repository https://noise-python.readthedocs.io/en/latest/examples.html.

- Jiang, C., Yuan, C., and Denolle, M. NoisePy: a new high-performance python tool for seismic ambient noise seismology.

MoPaD

The MoPaD website http://www.larskrieger.de/mopad/

- Lars Krieger and Sebastian Heimann. MoPad —Moment Tensor Plotting and Decomposition: A Tool for Graphical and Numerical Analysis of Seismic Moment Tensors.

BEAT

The BEAT website https://hvasbath.github.io/beat/index.html

- Vasyura-Bathke, H.,J. Dettmer, A. Steinberg, S. Heimann, M. P. Isken, O. Zielke, P. M. Mai, H. Sudhaus, and S. Jónsson (2020). The Bayesian Earthquake Analysis Tool, Seismol. Res. Lett. 91, 1003–1018, doi: 10.1785/0220190075.

Other references adding…

Working on that…

Indices and tables

- genindex
- search