Fermi GRB Analysis

Memcys

Jul 29, 2020

CONTENTS:

This is the documentation for the repo https://github.com/Memcys/Fermi-GRB-Analysis.

CHAPTER

ONE

INTRODUCTION

The repo Fermi-GRB-Analysis aims to reproduce the results in the paper [Xu2018][1].

1.1 Goals

The main goals are:

1. Retrive the raw data (Fermi LAT/GRB photon events)
2. Do the calculus and statistics
3. Make plots

1.2 File Tree

- `data`: parent directory for all kinds of data; can be altered by modifying `grb/config/path.py` or assigned by running `setup.py`
- `demo`: demo scripts that demonstrate the functionalities of the package `grb` (**entrance for usage**). It recommended to convert all `.py` to `.ipynb` before viewing and running. (See Jupytext for help.)
  - `fits.py`: demo to load FITS data; call `grb/lat/timeutils.py` to convert time between UTC and MET
  - `main.py`: call `grb/lat/analysis.py` for data analysis and data visualization (goals 2 and 3)
  - `query-LAT.py`: call `grb/lat/query.py` to request *Fermi* LAT data (goal 1)
- `docs`: documentation for the repo
  - `source`: source file for sphinx
  - `_build`: documents built by `sphinx`
    - `html`: documents formatted in `html` built by `sphinx`. Please open `index.html` with any web browser you like
- `grb`: Python namespace package
  - `config`: submodule for configuration
    - 'path.py`: for unified paths for root, data, tables and images
    - `__init__.py`: necessary file to make `grb` a package
  - `lat`: submodule for *Fermi* LAT analysis

    * `analysis.py`: for data analysis and data visualization
    * `query.py`: for requests of *Fermi* LAT
    * `__init__.py`

- `mkconfig.py`: modify `grb/config/path.py` as needed; called by `setup.py`
- `setup.py`: setup script to install the packag `grb`

[1]: Xu, H., & Ma, B.-Q. (2018). Regularity of high energy photon events from gamma ray bursts. Journal of Cosmology and Astroparticle Physics, 2018(01), 050–050. https://doi.org/10.1088/1475-7516/2018/01/050, http://arxiv.org/abs/1801.08084

CHAPTER TWO

SETUP

Please ensure `Python` version **at least 3.8**. Note that the repo is developed and tested on Linux, not Windows.

```
python3 --version   # or python --version, if python=python3
```

Python virtual environment is recommended. The following assumes use of `` `venv `` module <https://docs.python.org/3/library/venv.html>`__.

```
# create a virtual environment (need only once)
python3 -m venv /path/to/new/virtual/environment
# activate a virtual environment (every time log in)
source /path/to/new/virtual/environment/bin/activate
# update pip itself
pip3 install -U pip
# install dependencies
pip3 install -r /path/to/requirements.txt
```

You may prefer to assign a near mirror before download/install, for example, TUNA.

```
pip install pip -U
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

If you prefer to use `conda`, please refer to https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html.

To install the custom package `pkg`, please run under the **root** directory of the local repo:

```
python setup.py install
```

- `ROOT` (absolute path) for the root directory, default to current working directory
- `FITS` (relative to `ROOT`) for data files formated in FITS
- `TABLE` (relative to `ROOT`) for generated dataframes or downloaded tables
- `IMAGE` (relative to `ROOT`) for generated images

Once installed, you could import it the same way as other modules, for example:

```
from pkg import config
```

You could also uninstall the package via

```
pip uninstall pkg
```

To find out outdated packages, please run

```
pip list --outdated
```

To update outdated package, you may run

```
pip install -U package1, package2, ...
```

or follow https://github.com/pypa/pip/issues/3819.

NOTE: one should avoid update all packages unless using virtual environments, especially in *nix systems.

CHAPTER THREE

JUPYTEXT

Scripts under the `demo` directory are uploaded as `.py` files for better version control. However, it is recommended to view (and run) these demo scripts as `.ipynb` files. Jupytext does the job well. Example usage:

```
# Turn notebook.ipynb into a paired ipynb/py notebook
jupytext --set-formats py,ipynb query-LAT.py
# Update whichever of notebook.ipynb/notebook.py is outdated
jupytext --sync query-LAT.py
```

or if you prefer to convert all `.py` under the working directory:

```
jupytext --to ipynb *.py
```

Please see the documentation for details.

CHAPTER

FOUR

FITS FILE HANDLING

Fermi provides events (photons and spacecrafts) files in FITS format. Astropy provides tools to deal with FITS files.

With `astropy.io.fits`, one can - retrieve information about a GRB, and - load recorded data.

FITS is a binary format. One may get weird results to *directly* convert data in FITS to popular data structures such as Numpy Array or Pandas DataFrame. Fortunately, `astropy.table.Table` makes it easy to load data and then convert to Pandas DataFrame.

`demo/fits.py` demonstrates how to make good use of Astropy to handle FITS files. One may prefer to first convert it to `.ipynb` via `jupytext`, see Jupytext.

4.1 General references

- Learn Astropy
  - FITS File Handling
  - Astropy Table
- FITS Standard Page

CHAPTER

FIVE

DATA SOURCE

5.1 Overview

Type	Usage	Source
GRB/GCN name	Request and downloading FITS files	See *below*
GRB red shift	Calculation	See *below*
First low peak time of GBM TTE	Reference time	GBM TTE FITS header
Observed time and energy of LAT photons	Calculation	LAT FITS header

5.2 LAT Data

This project requires information of photon events detected by *Large Area Telegram*. The data server currently provides Pass 8 (P8R3) data. There are at least two ways to query the required LAT FITS files: 1. Manually input information about a LAT GRB in the data server 2. Use the package `fermi` provided by `astroquery` to query the information.

`demo/query-LAT.py` is the demo script to query and download LAT FITS files making use of `astroquery`.

This webpage lists all Fermi LAT Gamma-Ray Bursts (GRBs) from 080825C to 180720B. Note that the field "Object name or coordinates" in the data server corresponds to the `GCN Name` in the table.

Downloaded FITS files are assumed to be saved under the directory assigned by `grb.config.path.FITS`, which is by default `data/fits/` relative to the root directory.

LAT FITS files that I used have been released here. Please download a release file and extraced to `data/fits/` such that it looks like: - data/fits/ - 080916/ - criteria.csv - info.csv - L200408135242F357373F23_PH00.fits - L200408135242F357373F23_SC00.fits - 090323/ - … - …

5.3 GBM TTE Data

This project does not perform a GBM analysis. There are, however, the reference time is choosen as the *trigger time* of GBM, and some *dirty* scripts that try to do such an anlysis, which assumes that GBM TTE FITS files for a certain GRB be located under its GRB directory. For example, - data/fits/ - 080916/ - criteria.csv - info.csv - L200408135242F357373F23_PH00.fits - L200408135242F357373F23_SC00.fits - TTE/ - glg_tte_b0_bn080916009_v01.fit - glg_tte_b1_bn080916009_v01.fit - glg_tte_n0_bn080916009_v01.fit - … - glg_tte_nb_bn080916009_v01.fit - …

`.fit` files can be equally treated as `.fits`.

To query or download the GBM TTE files, there are at least two ways: 1. Search in the Fermi GBM Trigger Catalog (fermigtrig). 2. Suppose the trigger name of a GRB is known. Then the FITS file may be downloaded in: https://heasarc.gsfc.nasa.gov/FTP/fermi/data/gbm/triggers/yyyy/bnyymmddfff/current/glg_tte_xN_bnyymmddfff_vzz.fit, where `yyyy` is the year, `yymmdd` the Year-Month-Date, `fff` the fraction of a day, `x` in `b` (for BGO) or `n` (for NaI), `N` the hexadecimal (`b`: 0-1, `n`: 0-b), `zz` the version (typically `00` or `01`). Note that `yymmddfff` corresponds to the column "GRB". See here for details.

To obtain the *trigger time*, one can

- convert the column `trigger time` in the query results of "fermigtrig" to Fermi MET (via xTime on HEASARC), or
- read from the header of one of the 14 TTE FITS files.

5.3.1 Time Utils

One may have to convert time between UTC and Fermi Mission Elapsed Time (MET). The module `grb.lat.timeutils` serves the purpose. (One can compare the results with those from xTime.)

Suppose one knows from the fermigtrig that bn160625945 has trigger time: 2016-06-25 22:40:16.275. The MET can be obtained with:

```
In [1]: from grb.lat.timeutils impot UTCMET
In [2]: utc = '2016-06-25 22:40:16.275'
        UTCMET.utc2met(utc)
Out[2]: <Quantity 4.8858722e+08 s>
```

Please see `demo/fits.py` for more examples.

5.4 Red shifts

Red shifts can be obtained in the GCN Circular Archive or here.

CHAPTER SIX

GUIDE USAGE

The detailed code of this guide can be seen in `demo/main.py`.

Suppose you have the required FITS data under the directory assigned by `grb.config.path.FITS`. Other required information

- GRB/GCN name
- GBM MET (reference time)
- first low peak time (relative to the reference time)
- red shift

are in `grb.config.path.TABLE / "query-results-latest.h5"` and named "table".

Follow these steps to perform the analysis.

1. Instantiate the `PHTable`.
2. Run the method `regPlot` to obtain the power parameter in the power law.
3. Manually assign the primed class IV (classIV_less). However, this primed class is actually ommitable in my analysis.
4. Run one of the method `randPlotI`, `randPlotII`, `randPlotIII`, or `randPlotIV` to generate required number (via `repeat`) of samples and plot the results.

Note that you may use the Python function `dir` to list all available attributes of any Python objects. For example,

```
from grb.lat.analysis import PHTable

dir(PHTable)
```

CHAPTER

SEVEN

BUILD THE DOCS

Requirements (can be installed via `pip` or `conda`):

```
sphinx
sphinx-rtd-theme
```

First, generate package API via (at ROOT directory)

```
sphinx-apidoc -f --implicit-namespaces -o docs/source grb
```

Then build (html format) via

```
sphinx-build -b html docs/source docs/_build/html
```

And all done.

To view the output documentation, please open the file `docs/_build/html/index.html` with any web browser you like, e.g.,

```
firefox docs/_build/html/index.html
```

CHAPTER

EIGHT

GRB.LAT PACKAGE

8.1 Submodules

8.2 grb.lat.analysis module

This module deals with LAT analysis: calculus, statistics, and data visualization.

class `grb.lat.analysis.`**`Colname`**

Bases: `object`

Provide unified column names

`COLS = ['ENERGY', 'TIME']`

`DTOBS = 'DTobs'`

`DTTSF = 'DTtsf'`

`EIN = 'Ein'`

`ENERGY = 'ENERGY'`

`KAPPA = 'kappa'`

`NAME = 'NAME'`

`TIME = 'TIME'`

`TOBS = 'Tobs'`

`TPEAK = 'Tpeak'`

`Z = 'z'`

class `grb.lat.analysis.`**`FITSLoad`**(*grbs*, *in_dir*)

Bases: *grb.lat.analysis.Colname*

Load data of given GRBs from in_dir.

Parameters

- **grbs** (*pandas.core.frame.DataFrame*) – Information of GRBs, including GCN-NAME, ENERGY and TIME.
- **in_dir** (*pathlib.Path or str*) – Input directory that contains GRB FITS files.

class `grb.lat.analysis.`**`PHTable`**(*grbs*, *in_dir*, *erange=<Quantity [1., 10., 20., 40., 150.] GeV>*)

Bases: *grb.lat.analysis.FITSLoad*, *grb.lat.analysis.Rand*

Load GRBs and do calculations and convertings.

Parameters

- **grbs** (*pandas.core.frame.DataFrame*) – Information of GRBs, including GCN-NAME, ENERGY and TIME.
- **in_dir** (*pathlib.Path or str*) – Input directory that contains GRB FITS files.
- **erange** (*astropy.units.quantity.Quantity*) – Array of 5 end points for 4 energy ranges.

`T(`*Set*, *rho: int*, *return_extr: bool = False*`)`

Calculate the test function.

Parameters

- **Set** (*astropy.table.table.Table*) – The set of all photon events.
- **rho** (*int*) – Constant parameter in the test function.
- **return_extr** (*bool, optional*) – Whether to return the extrema, by default False.

Returns values of the test function.

Return type numpy.ndarray

Note:

$$T_\rho = \frac{\sum_{i=1}^{N-\rho} \log[\bar{t}_\rho/(t_{i+\rho} - t_i)]}{N - \rho},$$

$$\bar{t}_\rho = \frac{\sum_{i=1}^{N-\rho} (t_{t+\rho} - t_i)}{N - \rho}.$$

See also:

Rand.Calc_Ts() Return the test function T of shape (row_len, sz).

`TEPlot(`*Set*, *rho: int*, *repeat: int*, *Emax: float*, *Emin: float*, *label: str*, *return_extr: bool*`)`

Test function versus Energy plot.

Parameters

- **Set** (*astropy.table.table.Table*) – All photon events.
- **rho** (*int*) – Constant parameter in the test function.
- **repeat** (*int*) – Number of random samples.
- **Emax** (*float*) – Maximum of the energy range.
- **Emin** (*float*) – Minimum of the energy range.
- **label** (*str*) – Label in the T-E plot.
- **return_extr** (*bool*) – Whether to return the extrama.

Returns return fig, and dict of extrama (if any).

Return type tuple of (matplotlib.figure.Figure, None or dict)

See also:

T() Test function.

`checkPower()` → bool

Check if the power has been assigned.

Returns True if self.POWER has been assigned; False otherwide.

Return type bool

Note: Power Law:

$$\frac{\mathrm{d}N}{\mathrm{d}E} \propto \left(\frac{E}{E_{\mathrm{ref}}}\right)^{-\alpha},$$

$$E_{\mathrm{ref}} = 1.0\mathrm{GeV}$$

`randPlotI`(*rho: int = 5*, *repeat: int = 100000*, *label: str = 'Data Set I'*, *return_extr: bool = False*)

T versus E plot for Data Set I.

Parameters

- **rho** (*int, optional*) – Constant parameter in the test function, by default 5.
- **repeat** (*int, optional*) – Number of random samples, by default 100000.
- **label** (*str, optional*) – Label in the plot, by default 'Data Set I'.
- **return_extr** (*bool, optional*) – Whether to return the extrama, by default False.

Returns return fig, and dict of extrama (if any).

Return type tuple of (matplotlib.figure.Figure, None or dict)

See also:

TEPlot() T versus E plot for individual data set.

`randPlotII`(*rho: int = 5*, *repeat: int = 100000*, *label: str = 'Data Set II'*, *return_extr: bool = False*)

T versus E plot for Data Set II.

Parameters

- **rho** (*int, optional*) – Constant parameter in the test function, by default 5.
- **repeat** (*int, optional*) – Number of random samples, by default 100000.
- **label** (*str, optional*) – Label in the plot, by default 'Data Set II'.
- **return_extr** (*bool, optional*) – Whether to return the extrama, by default False.

Returns return fig, and dict of extrama (if any).

Return type tuple of (matplotlib.figure.Figure, None or dict)

See also:

TEPlot() T versus E plot for individual data set.

randPlotIII(*rho: int = 5*, *repeat: int = 100000*, *label: str = 'Data Set III'*, *return_extr: bool = False*)

T versus E plot for Data Set III.

Parameters

- **rho** (*int, optional*) – Constant parameter in the test function, by default 5.
- **repeat** (*int, optional*) – Number of random samples, by default 100000.
- **label** (*str, optional*) – Label in the plot, by default 'Data Set III'.
- **return_extr** (*bool, optional*) – Whether to return the extrama, by default False.

Returns return fig, and dict of extrama (if any).

Return type tuple of (matplotlib.figure.Figure, None or dict)

See also:

TEPlot() T versus E plot for individual data set.

randPlotIV(*rho: int = 5*, *repeat: int = 100000*, *label: str = 'Data Set IV'*, *less: bool = False*, *return_extr: bool = False*)

T versus E plot for Data Set IV or IV' (IV_less).

Parameters

- **rho** (*int, optional*) – Constant parameter in the test function, by default 5.
- **repeat** (*int, optional*) – Number of random samples, by default 100000.
- **label** (*str, optional*) – Label in the plot, by default 'Data Set IV'.
- **less** (*bool, optional*) – Plot for Data Set IV_less or IV.
- **return_extr** (*bool, optional*) – Whether to return the extrama, by default False.

Returns return fig, and dict of extrama (if any).

Return type tuple of (matplotlib.figure.Figure, None or dict)

See also:

TEPlot() T versus E plot for individual data set.

regPlot(*binmethod=array([1.0, 1.04811313, 1.09854114, 1.1513954, 1.20679264, 1.26485522, 1.32571137, 1.38949549, 1.45634848, 1.52641797, 1.59985872, 1.67683294, 1.75751062, 1.84206997, 1.93069773, 2.02358965, 2.12095089, 2.22299648, 2.32995181, 2.44205309, 2.55954792, 2.6826958, 2.8117687, 2.9470517, 3.0888436, 3.23745754, 3.39322177, 3.55648031, 3.72759372, 3.90693994, 4.09491506, 4.29193426, 4.49843267, 4.71486636, 4.94171336, 5.17947468, 5.42867544, 5.68986603, 5.96362332, 6.25055193, 6.55128557, 6.86648845, 7.19685673, 7.54312006, 7.90604321, 8.28642773, 8.68511374, 9.10298178, 9.54095476, 10.0])*, *figname: str = ''*)

Plot the histogram of Data Set I and the regression line.

Parameters

- **binmethod** (*numpy.ndarray, optional*) – Bin method of the histogram plot, by default np.logspace(0., 1.0).

- **figname** (*str, optional*) – Output file name of the figure, by default '' (and not output to file).

save(*out_dir: str = '../data/', fname: str = 'ph-table-all', date: bool = True*)
Save the instance as pickle format, in order to reuse the instance without repeated calculations.

Parameters

- **out_dir** (*str, optional*) – Output directory, by default '../data/'
- **fname** (*str, optional*) – Output file name, by default 'ph-table-all'
- **date** (*bool, optional*) – Whether to include date in the file name, by default True

scatterPlot_(*ph_class, label: str, alpha: float = 0.75*)
Individual scatter plot for each data set.

Parameters

- **ph_class** (*astropy.table.table.Table*) – All photon events.
- **label** (*str*) – Label in the plot.
- **alpha** (*float, optional*) – Alpha of the plot, by default 0.75

Returns $\Delta t_{obs}/(1+z)$ versus κ plot.

Return type matplotlib.figure.Figure

scatterPlots(*labels=['Data Set I', 'Data Set II', 'Data Set III', 'Data Set IV', "Data Set IV'"]*)
Scatter plots of all data sets.

Parameters labels (*list of str, optional*) – Labels in the plots, by default ['Data Set ' + i for i in ['I', 'II', 'III', 'IV', "IV'"]]

Returns $\Delta t_{obs}/(1+z)$ versus κ plots.

Return type matplotlib.figure.Figure

See also:

scatterPlot_() Individual scatter plot for each data set.

class grb.lat.analysis.Rand(*Set, Emax: float, Emin: float, power: float, E_LVs, rho: int = 5, repeat: int = 100000, E_LV_sz: int = 1000*)
Bases: *grb.lat.analysis.Colname*

Generate random samples.

Parameters

- **Set** (*astropy.table.table.Table*) – The set of GRBs.
- **Emax** (*float*) – Maximum of the energy range.
- **Emin** (*float*) – Minimum of the energy range.
- **power** (*float*) – The parameter in the power law.
- **E_LVs** (*numpy.ndarray*) – Candidate Lorentz Violation parameters.
- **rho** (*int, optional*) – Constant rho in the test function, by default 5.
- **repeat** (*int, optional*) – Size of random samples, by default 100000.
- **E_LV_sz** (*int, optional*) – Size of random candidate Lorentz Violation parameters, by default 1000.

static Calc_Ts(*row_len: int*, *sz: int*, *rho: int*, *DTtsf*, *kappa*, *E_LVs*)
Return the test function T of shape (row_len, sz).

Parameters

- **row_len** (*int*) – Number of total candidate Lorentz Violation parameters; also T.shape[0].
- **sz** (*int*) – Number of photon events; also T.shape[1].
- **rho** (*int*) – constant rho in the test function T.
- **DTtsf** (*numpy.ndarray*) – observed time over (1 + z), i.e., $\Delta T_{obs}/(1+z)$.
- **kappa** (*numpy.ndarray*) – kappa of the same shape as DTtsf.
- **E_LVs** (*numpy.ndarray*) – candidate Lorentz Violation parameters

Returns Values of the test function

Return type numpy.ndarray

static Calc_kappa(*Eobs_repeat*, *z_repeat*, *Omega_m: float = 0.315*)
Return kappa caculated from observed energy and red-shift.

Parameters

- **Eobs_repeat** (*numpy.ndarray*) – Observed energy shape in (n, m), where n is the number of photon events, and m is the repeat time (m = 1 for non-repeat).
- **z_repeat** (*numpy.ndarray*) – red shift of the same shape as Eobs_repeat.
- **Omega_m** (*float, optional*) – matter density parameter in the ΛCDM model, by default 0.315, from[1].

Returns kappa of the same shape as z_repeat (and Eobs_repeat).

Return type numpy.ndarray

Note:

$$\kappa = s\frac{E_\mathrm{h} - E_\mathrm{l}}{H_0}\frac{1}{(1+z)}\int_0^z \frac{(1+z')\mathrm{d}z'}{\sqrt{\Omega_m(1+z')^3+\Omega_\Lambda}} \tag{8.1}$$

See[2].

References

randPlot()
Plot n-sigma regions, where n's are 1, 3, 5.

[1] Planck Collaboration, Aghanim, N., Akrami, Y., Ashdown, M., Aumont, J., Baccigalupi, C., Ballardini, M., Banday, A. J., Barreiro, R. B., Bartolo, N., Basak, S., Battye, R., Benabed, K., Bernard, J.-P., Bersanelli, M., Bielewicz, P., Bock, J. J., Bond, J. R., Borrill, J., … Zonca, A. (2020). Planck 2018 results. VI. Cosmological parameters. Astronomy & Astrophysics. https://doi.org/10.1051/0004-6361/201833910

[2] Xu, H., & Ma, B.-Q. (2018). Regularity of high energy photon events from gamma ray bursts. Journal of Cosmology and Astroparticle Physics, 2018(01), 050–050. https://doi.org/10.1088/1475-7516/2018/01/050

8.3 grb.lat.query module

This module deals with querying and downloading LAT FITS files.

`class grb.lat.query.Download`(*grbs*)

Bases: `object`

`DONE = 'DONE'`

`Download_fits`(*row: int*, *out_dir*, *timeout: float = - 1.0*)

Download fits files provided in urls and save to out_dir.

Parameters

- `row` (*int*) – Row index of the given GRB.
- `out_dir` (*[type]*) – Output directory for FITS file.
- `timeout` (*float, optional*) – Time for timeout in second, by default -1 (no timeout).

Returns self.DONE: if succeeded; self.MISSING: if failed

Return type self.DONE or self.MISSING

`Download_info`(*row*, *out_dir*, *pre='https://fermi.gsfc.nasa.gov/cgi-bin/ssc/LAT/QueryResults.cgi?id='*, *wait=2*, *timeout: float = - 1.0*)

Request query page in url, and save tables to out_dir.

Parameters

- `row` (*int*) – Row index of the given GRB.
- `out_dir` (*str*) – Output directory for FITS file.
- `pre` (*str, optional*) – prefix in url, by default INFOPRE
- `wait` (*int or float, optional*) – Wait time in second, by default WAIT
- `timeout` (*float, optional*) – Time for timeout in second, by default -1 (no timeout).

Returns self.DONE: if succeeded; self.MISSING: if failed

Return type self.DONE or self.MISSING

`static Filename`(*path*, *sep='/'*)

Retrieve the file name (without directory) from a full path.

Parameters

- `path` (*str*) – Path to a GRB FITS file.
- `sep` (*str, optional*) – Seperator of directory, by default '/' (in Unix).

Returns File name without directory.

Return type str

`GRB_record`(*row: int*, *col: str*, *value*)

Record information for the for the given grb.

Parameters

- `row` (*int*) – Row index of the given GRB.
- `col` (*str*) – Colume index of the given GRB.

- **value** – Any value to save to the unit.

`INFOPRE = 'https://fermi.gsfc.nasa.gov/cgi-bin/ssc/LAT/QueryResults.cgi?id='`

`MISSING = <NA>`

`Missing`(*row: int*, *col: str*)
Check if the data in the give unit is missing.

Parameters

- **row** (*int*) – Row index of the given GRB.
- **col** (*str*) – Colume index of the given GRB.

Returns True if missing; else, False.

Return type bool

`NAME = 'GCNNAME'`

`Query_url`(*row: int*, *period: float*, *E_MeV_i: float*, *E_MeV_f: float*, *trigtime: str*, *tpeak: str*, *timeout: float = - 1.0*)
Query urls for downloading.

Parameters

- **row** (*int*) – Row index of the given GRB.
- **period** (*float*) – Period after initial time in second.
- **E_MeV_i** (*float*) – Start energy in MeV.
- **E_MeV_f** (*float*) – End energy in MeV.
- **trigtime** (*str*) – Mission Elapsed Time in second.
- **tpeak** (*str*) – First low peak time in second.
- **timeout** (*float, optional*) – Time for timeout in second, by default -1 (no timeout).

`TBUFF = 30.0`

`TRIES = 3`

`Urls_resolve`(*row*)
Retrive urls from the given row.

Parameters **row** (*int*) – Row index of the given GRB.

Returns Urls for a given LAT GRB FITS photon (PH) and spacecraft (SC) files.

Return type list of str, or None

`WAIT = 2`

class `grb.lat.query.Query`(*grbs*, *out_dir*, *init=False*, *retry=True*, *timeout: float = - 1.0*)
Bases: *grb.lat.query.Download*

`EMAX = 500000.0`

`EMIN = 100.0`

`FAKE = 'fake'`

`Main_loop`(*outer_dir*)
Main loop to download all required data.

Parameters **outer_dir** (*pathlib.PosixPath*) – Output directory.

`PERIOD = 90.0`

`Requery()`
Remove queried urls and run Main_loop for missing grbs.

`Reset(`*init=False*`)`
Initialize urls of grbs with missing fits or info.

Parameters `init` (*bool, optional*) – Whether to initialize the table, by default False.

`Row_index(`*name*`)`
Return index of the given name

`TIMESYS = 'MET'`

`Which_missing()`
Find GRBs with missing information.

Returns GRBs with missing information.

Return type pandas.DataFrame or astropy.table.table.Table

`_Which_missing()`
Find locations of missing information.

Returns Where the information is missing, the location of it will be True.

Return type list of bool

8.4 grb.lat.timeutils module

This module provides utils to convert UTCMET between UTC and Fermi Mission Elapsed Time (MET).

`class grb.lat.timeutils.UTCMET`
Bases: `object`

Convert time between UTC and Fermi MET.

`static met2utc(`*met*`)`
Convert Fermi MET to UTC time.

Parameters `met` (*float or array-like*) – Fermi Mission Elapsed Time in second.

Returns Time object correspond to the input MET.

Return type astropy.time.core.Time

Examples

```
>>> met = 488587220.275348
>>> utc = UTCMET.met2utc(met)
>>> utc
<Time object: scale='utc' format='iso' value=2016-06-25 22:40:16.275>
```

To retrive the value: >>> utc.value ‘2016-06-25 22:40:16.275’

`t0 = <Time object: scale='utc' format='iso' value=2001-01-01 00:00:00.000>`

`static utc2met(`*utc*`)`
Convert UTC time to Fermi MET.

Parameters **utc** (*str, or list of str*) – UTC time formated like `yyyy-mm-dd hh:mm:ss' or so.

Returns Fermi MET time corresponds to the the input UTC time.

Return type astropy.units.quantity.Quantity

Examples

```
>>> utc = '2008-08-28T10:46:30.271448'
>>> met = UTCMET.utc2met(utc)
>>> met
<Quantity 2.41613191e+08 s>
>>> met.value
241613191.27144802
```

8.5 Module contents

This sub-package provides modules to query LAT FITS and perform LAT analysis.

CHAPTER NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

S

T

U

W

Z