# MOSFiT Documentation

***Release 0.7.12***

**&**

**Sep 07, 2017**

# Contents

Welcome to the documentation for MOSFiT (The Modular Open Source Fitter for Transients), a Python 2.7/3.x package for fitting, sharing, and estimating the parameters of transients via user-contributed transient models.

Please see the links below for information on how to install and run the package.

CHAPTER 1

# Using MOSFiT

## Installation

Several installation methods for `MOSFiT` are outlined below. If you run into issues, open a new issue on GitHub.

### Setting up MOSFiT with Anaconda

**Platforms:** MacOS X, Linux, and Windows

We recommend using the Anaconda Python distribution from Continuum Analytics (or the related Miniconda distribution) as your Python environment.

After installing conda, `MOSFiT` can be installed via:

```
conda install -c conda-forge mosfit
```

### Installing with pip

**Platforms:** MacOS X, Linux, and Windows

Installing `MOSFiT` with pip is straightforward:

```
pip install mosfit
```

## Getting started

Once installed, `MOSFiT` can be run from any directory, and it's typically convenient to make a new directory for your project.

```
mkdir mosfit_runs
cd mosfit_runs
```

MOSFiT can be invoked either via either `python -m mosfit` or simply `mosfit`. Then, to run MOSFiT, pass an event name to the program via the `-e` option:

```
mosfit -e LSQ12dlf
```

The above command will prompt the user to choose a model (of those distributed with MOSFiT) to fit against the data, using the event's claimed type to provide a list of suggested models. A specific model can be fit to transients using the model option `-m`:

```
mosfit -e LSQ12dlf -m slsn
```

Multiple events can be fit in succession by passing a list of names separated by spaces (names containing spaces can be specified using quotation marks):

```
mosfit -e LSQ12dlf SN2015bn "SDSS-II SN 5751"
```

The code outputs JSON files for each event/model combination that each contain a set of walkers that have been relaxed into an equilibrium about the posterior parameter distributions. This output is visualized via an example Jupyter notebook (`mosfit.ipynb`), which is copied to the `products` folder in the run directory, and by default shows output from the last MOSFiT run.

## Parallel execution

MOSFiT is parallelized and can be run in parallel by prepending `mpirun -np #`, where `#` is the number of processors in your machine +1 for the master process. So, if you computer has 4 processors, the above command would be:

```
mpirun -np 5 mosfit -e LSQ12dlf
```

MOSFiT can also be run without specifying an event, which will yield a collection of light curves for the specified model described by the priors on the possible combinations of input parameters specified in the `parameters.json` file. This is useful for determining the range of possible outcomes for a given theoretical model:

```
mpirun -np 5 mosfit -m magnetar
```

## Using your own data

MOSFiT has a built-in converter that can take input data in a number of formats and convert that data to the Open Catalog JSON format. Using the converter is straightforward, simply pass the path to the file(s) using the same `-e` option:

```
mosfit -e my_ascii_data_file.csv
```

MOSFiT will convert the files to JSON format and immediately begin processing the new files (append `-i 0` to immediately exit after conversion). For more information, please see the *Private data* section.

## Producing outputs

All outputs (except for converted observational data) are stored in the `products` directory, which is created by MOSFiT automatically in the current run directory. By default, a single file with the transient's name, e.g.

`LSQ12dlf.json`, will be produced; this file contains all of the information originally available in the input JSON file and the results of the fitting. An exact copy of this file is stored under the name `walkers.json` for convenience.

Additional outputs can be produced via some optional options that can be passed to `MOSFiT`. Please see the *arbitrary outputs* section.

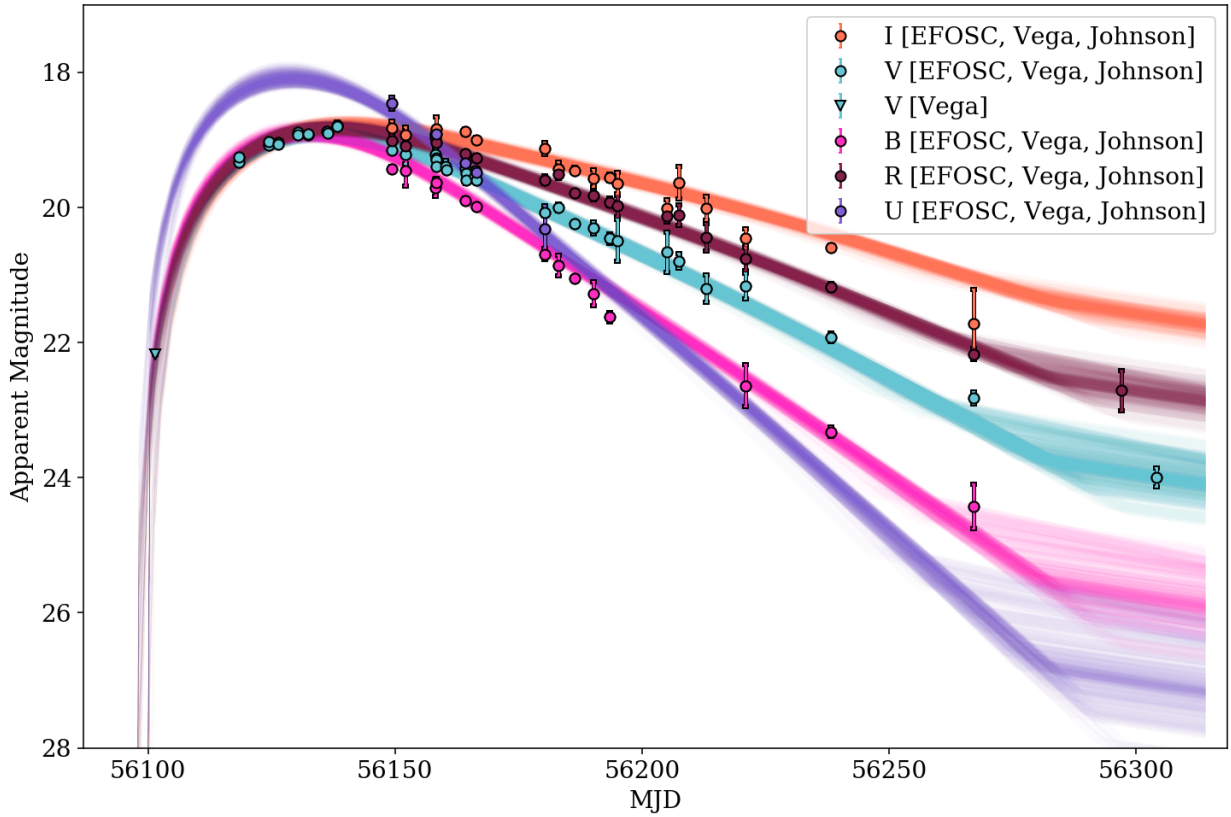
## Visualizing outputs

The outputs from `MOSFiT` can be visualized using the Jupyter notebook `mosfit.ipynb` copied by the code into a `jupyter` directory within the current run directory. This notebook is intended to be a simple demonstration of how to visualize the output data, and can be modified by the users to their own needs.

First, the user should make sure that Jupyter is installed, then execute the Jupyter notebook from the run directory:

```
jupyter notebook jupyter/mosfit.ipynb
```

In this notebook, there are four cells which should require minimal editing to visualize your results; the cells should be evaluated in order. The first cell imports modules and loads the data output from the last run (store in `walkers.json`). The second cell displays the ensemble of light curve fits and the data the model was fitted to:




The third cell shows X-ray observations, if the transient had any.

The fourth cell shows the evolution of free parameters as a function of time (the Monte Carlo chain).

The last cell produces a corner plot using the corner package.

### Sharing data and outputs with the community

To upload fits back to the Open Catalogs, users can simply pass the `-u` option. The first time `-u` is used, MOSFiT will request a Dropbox token, which is provided on the Open Astronomy Catalogs on the pages describing `MOSFiT`, e.g. https://sne.space/mosfit/. These tokens can be revoked at any time, so a user may be occaisionally asked to enter a new token if the old one has expired.

Upon completing the fitting process, and if the fits satisfy some quality checks, the model fits will uploaded to the Open Catalogs, where they will be ingested and available approximately 24 hours after their submission.

If the data was read from a file (rather than from one of the Open Astronomy Catalogs), and if the `-u` option was provided, `MOSFiT` will offer the user the option of uploading the *observed* data to the Open Catalogs as well, in addition to the model fits. Because of the possibility that local data passed to `MOSFiT` is private, the user will be asked if they wish to upload each event before they are uploaded. Users should immediately contact the maintainers of `MOSFiT` if they believe they have uploaded private data in error (because of the 24 hour waiting period, the inadvertantly uploaded data can be purged before becoming public if the maintainers are given enough notice).

### Troubleshooting

If you are having trouble getting `MOSFiT` working, please first consult our *Frequently Asked Questions* page, which addresses many common issues. If the answers there do not answer your questions, feel free to join our MOSFiT Slack channel and ask for assistance.

## Models

`MOSFiT` has been designed to be modular and easily modifiable by users to alter, combine, and create physical models to approximate the behavior of observed transients. On this page we walk through some basics on how a user might alter an existing model shipped with the code, and how they could go about creating their own model.

### List of built-in models

| Model name | Description | Reference(s) |
|---|---|---|
| `default` | Nickel-cobalt decay | 1994ApJS...92..527N |
| `csm` | Interacting CSM-SNe | 2013ApJ...773...76C |
| `csmni` | CSM + NiCo decay | See `default` & `csm` |
| `exppow` | Analytical engine | |
| `ia` | NiCo decay + I-band | |
| `ic` | NiCo decay + radio | |
| `magnetar` | Magnetar engine w/ simple SED | 2017arXiv170600825N |
| `magni` | Above + NiCo decay | |
| `rprocess` | Kilonovae | 2016arXiv161009381M |
| `slsn` | Magnetar + modified SED + constraints | 2017arXiv170600825N |
| `tde` | Tidal disruption events | Mockler+ 2017 |

### Altering an existing model

For many simple alterations to a model, such as adjusting input priors, setting variables remain free and which should be fixed, and adding/removing modules to the call stack, the user only needs to modify copies of the model JSON files. For these sorts of minor changes, no Python code should need to be modified by the user!

For this example, we'll presume that the user will be modifying the `slsn` model. First, the user should create a new run directory and run `MOSFiT` there once to copy the model JSON files to that run directory:

```
mkdir slsn_run
cd slsn_run
python -m mosfit -m slsn
```

After running, the user will notice four directories will have been created in the run directory: `models`, `modules`, `jupyter`, and `products`. `models` will contain a clone of the `models` directory structure, with the `parameters.json` files for each model copied into each model folder, `modules` contains a clone of the `modules` directory structure, etc.

### Changing parameter priors

From your run directory, navigate into the `models/slsn` directory and edit the `parameters.json` file in your favorite text editor:

```
cd models/slsn
vim parameters.json
```

You'll notice that `parameters.json` file is fairly bare-bones, containing only a list of model parameters and their allowed value ranges:

```
{
    "nhhost":{
        "min_value":1.0e16,
        "max_value":1.0e23,
        "log":true
    },
    "Pspin":{
        "min_value":1.0,
        "max_value":10.0
    },
    "Bfield":{
        "min_value":0.1,
        "max_value":10.0
    },
    "Mns":{
        "min_value":1.0,
        "max_value":2.0
    },
}
```

Now, change the range of allowed neutron star masses to something else:

```
{
    "Mns":{
        "min_value":1.5,
        "max_value":2.5
    },
}
```

**Congratulations!** You have just modified your first MOSFiT model. It should be noted that even this very minor change, which affects the range of a single parameter, would generate a completely different model hash than the default model, distinguishing it from any other models that might have been uploaded by other users using the default settings.

You can also use more complex priors within the same file. For example:

```
{
"Mns":{
    "class":"gaussian",
    "mu":1.4,
    "sigma":0.4,
    "min_value":0.1,
    "max_value":3.0,
    "log":false
}
}
```

A list of available priors is below; for all prior types, `min_value` and `max_value` specify the minimum and maximum allowed parameter values, and `log` will apply the prior to the log transform of the parameter.

| Prior name | Equation | Additional parameters |
|---|---|---|
| `parameter` | $\Pi \sim \text{constant}$ | |
| `gaussian` | $\Pi \sim \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$ | $\mu$ (`mu`), $\sigma$ (`sigma`) |
| `powerlaw` | $\Pi \sim x^{-\alpha}$ | $\alpha$ (`alpha`) |

### Swapping modules

Let's say you want to modify the SLSN model such that transform applied to the input engine luminosity is not diffusion, but instead viscosity (if the light of a SLSN was say filtered through an accretion disk rather than a dense envelope). To make this change, the user would want to swap out the `diffusion` module used by `slsn` for the `viscous` module. This can be accomplished by editing the `slsn.json` model file. The model files are not copied into the model directories by default (as they may change from version to version of `MOSFiT`), but a `README` file with the full path to the model is copied to all model folders to make it easy for the user to copy the relevant JSON files:

```
cd models/slsn
cp $(head -1 README)/* .
vim slsn.json
```

To swap `diffusion` for `viscous`, the user would remove the blocks of JSON that refer to the `diffusion` module:

```
{
    "kappagamma":{
        "kind":"parameter",
        "value":10.0,
        "class":"parameter",
        "latex":"\\kappa_\\gamma\\,({\\rm cm}^{2}\\,{\\rm g}^{-1})"
    },
    "diffusion":{
        "kind":"transform",
        "inputs":[
            "magnetar",
            "kappa",
            "kappagamma",
            "mejecta",
            "texplosion",
            "vejecta"
        ]
    },
```

```
    "temperature_floor":{
        "kind":"photosphere",
        "inputs":[
            "texplosion",
            "diffusion",
            "temperature"
        ]
    },
    "slsn_constraints":{
        "kind":"constraint",
        "inputs":[
            "mejecta",
            "vejecta",
            "kappa",
            "tnebular_min",
            "Pspin",
            "Mns",
            "diffusion",
            "texplosion",
            "redshift",
            "alltimes",
            "neutrino_energy"
        ]
    },
}
```

and replace them with blocks appropriate for `viscous`:

```
{
    "Tviscous":{
        "kind":"parameter",
        "value":1.0,
        "class":"parameter",
        "latex":"T_{\\rm viscous}"
    },
    "viscous":{
        "kind":"transform",
        "inputs":[
            "magnetar",
            "texplosion",
            "Tviscous"
        ]
    },
    "temperature_floor":{
        "kind":"photosphere",
        "inputs":[
            "texplosion",
            "viscous",
            "temperature"
        ]
    },
    "slsn_constraints":{
        "kind":"constraint",
        "inputs":[
            "mejecta",
            "vejecta",
            "kappa",
            "tnebular_min",
```

```
            "Pspin",
            "Mns",
            "viscous",
            "texplosion",
            "redshift",
            "alltimes",
            "neutrino_energy"
        ]
    },
}
```

As can be seen above, this involved removal of definitions of free parameters that only applied to `diffusion` (`kappagamma`), the addition of a new free parameter for `viscous` (`Tviscous`), and replacement of various `inputs` that depended on `diffusion` with `viscous`.

The user should also modify the `parameters.json` file to remove free parameters that are no longer in use:

```
{
    "kappagamma":{
        "min_value":0.1,
        "max_value":1.0e4,
        "log":true
    },
}
```

and to the define the priors of their new free parameters:

```
{
    "Tviscous":{
        "min_value":1.0e-3,
        "max_value":1.0e5,
        "log":true
    },
}
```

## Creating a new model

If users would like to create a brand new model for the `MOSFiT` platform, it is easiest to duplicate one of the existing models that most closely resembles the model they wish to create.

If you go this route, we highly recommend that you fork MOSFiT on GitHub and clone your fork, with development being done in the cloned `mosfit` directory:

```
git clone https://github.com/your_github_username/MOSFiT.git
cd mosfit
```

Copy one of the existing models as a starting point:

```
cp -R models/slsn models/my_model_that_explains_everything
```

Inside this directory are two files: a `model_name.json` file and a `parameters.json` file. We must edit both files to run our new model.

First, the `model_name.json` file should be edited to include your model's:

- Parameters
- Engine(s)

- Diffusion prescription
- Photosphere prescription
- SED prescription
- The photometry module

Optionally, your model file can also include an extinction prescription.

Then, you need to edit the `parameters.json` to include the priors on all ofyour model parameters. If no prior is specified, the variable will be set to a constant.

You can invoke the model using:

```
python -m my_model_that_explains_everything
```

If your model requires a new engine, you can create this engine by again copying an existing engine:

```
cp modules/engines/nickelcobalt.py my_new_engine.py
```

Then plug this engine into your model's json file.

# Fitting data

The primary purpose of `MOSFiT` is to fit models of transients to observed data. In this section we cover "how" of fitting, and how the user should interpret their results.

## Public data

`MOSFiT` is deeply connected to the Open Catalogs (The Open Supernova Catalog, the Open Tidal Disruption Catalog, etc.), and the user can directly fit their model against any data provided by those catalogs. The Open Catalogs store names for each transient, and the user can access any transient by any known name of that transient. As an example, both of the commands below will fit the same transient:

```
mosfit -m slsn -e PTF11dij
mosfit -m slsn -e CSS110406:135058+261642
```

While the Open Catalogs do their best to maintain the integrity of the data they contain, there is always the possibility that the data contains errors, so users are encouraged to spot check the data they download before using it for any scientific purpose. A common error is that the data has been tagged with the wrong photometric system, or has not been tagged with a photometric system at all and uses a different system from what is commonly used for a given telescope/instrument/band. Users are encouraged to immediately report any issues with the public data on the GitHub issues page assocated with that catalog (e.g. the Open Supernova Catalog's issue page).

## Private data

If you have private data you would like to fit, the most robust way to load the data into `MOSFiT` is to directly construct a JSON file from your data that conforms to the Open Catalog Schema. This way, the user can specify all the data that `MOSFiT` can use for every single observation in a precise way. All data provided by the Open Catalogs is provided in this form, and if the user open up a typical JSON file downloaded from one of these catalogs, they will find that each observation is tagged with all the information necessary to model it.

Of course, it is more likely that the data a user will have handy will be in another form, typically an ASCII table where each row presents a single (or multiple) observations. `MOSFiT` includes a conversion feature where the user can simply pass the path to the file(s) to convert:

```
mosfit -e path/to/my/ascii/file/my_transient.dat
mosfit -e path/to/my/folder/of/ascii/files/*.dat
```

In some cases, if the ASCII file is in a simple form with columns that match all the required columns, `MOSFiT` will silently convert the input files into JSON files, a copy of which will be saved to the current run directory. In most cases however, the user will be prompted to answer a series of questions about the data in a "choose your own adventure" style. If passed a list of files, `MOSFiT` will assume all the files share the same format and the user will only be asked questions about the first file.

If the user so chooses, they may *optionally* upload their data directly to the Open Catalogs with the `-u` option. This will make their observational data publicly accessible on the Open Catalogs:

```
mosfit -e path/to/my/ascii/file/my_transient.dat -u
```

Note that this step is completely optional, users do not have to share their data publicly to use `MOSFiT`, however it is the fastest way for your data to appear on the Open Catalogs. If a user believes they have uploaded any private data in error, they are encouraged to immediately contact the *maintainers*.

## Initialization

When initializing, walkers are drawn randomly from the prior distributions of all free parameters, unless the `-w` option was passed to initialize from a previous run (see *previous*). By default, any drawn walker that has a defined, non-infinite score will be retained, unless the `-d` option is used, which by default only draws walkers above the average walker score drawn so far, or the numeric value specified by the user (warning: this option can often make the initial drawing phase last a *long* time).

### Restricting the data used

By default, `MOSFiT` will attempt to use all available data when fitting a model. If the user wishes, they can exclude specific instruments from the fit using the `--exclude-instruments` option, specific photometric bands using the `--exclude-bands` option, and specific sources of data (e.g. papers or surveys) using `--exclude-sources`. The source is specified using the source ID number, visible on the Open Astronomy Catalog page for each transient as well as in the input file. For example

```
mosfit -e LSQ12dlf -m slsn --exclude-sources 2
```

will exclude all data from the paper that has the source ID number 2 on the Open Astronomy Catalog page.

To exclude times from a fit, the user can specify a range of MJDs that will be included using the `-L` option, e.g.:

```
mosfit -e LSQ12dlf -m slsn -L 55000 56000
```

will limit the data fitted for LSQ12dlf to lie between MJD 55000 and MJD 56000.

### Number of walkers

The sampler used in `MOSFiT` is a variant of `emcee`'s multi-temperature sampler `PTSampler`, and thus the user can pass both a number of temperatures to use with `-T` in addition to the number of walkers `-N` per temperature. If one temperature is used (the default), the total number of walkers is simply whatever is passed to `-N`, otherwise it is $N * T$.

### Duration of fitting

The duration of the `MOSFiT` run is set with the `-i` option, unless the `-R` or `-U` options are used (see *convergence*). Generally, unless the model has only a few free parameters or was initialized very close to the solution of highest-likelihood, the user should not expect good results unless `-i` is set to a few thousand or more.

## Burning in a model

Unless the solution for a given dataset is known in advance, the initial period of searching for the true posterior distribution involves finding the locations of the solutions of highest likelihood. In `MOSFiT`, various `scipy` routines are employed in an alernating fashion with a Gibbs-like affine-invariant ensemble evolution, which we have found more robustly locates the true global likelihood minimas. The period of alternation between optimization (called "fracking" in `MOSFiT`) and sampling (called "walking" in `MOSFiT`) is controlled by the `-f` option, with the total burn-in duration being controlled by the `-b`/`-p` options. If `-b`/`-p` are not set, the burn-in is set to run for half the total number of iterations specified by `-i`.

As an example, the following will run the burn-in phase for 2000 iterations, the post burn-in for 3000 iterations more (for a total of 5000), fracking every 100th iteration:

```
mosfit -e LSQ12dlf -m slsn -f 100 -i 5000 -b 2000
```

All *convergence* metrics are computed *after* the burn-in phase, as the operations employed during burn-in do *not* preserve detailed balance. During burn-in, the solutions of highest likelihood are over-represented, and thus the posteriors should not be trusted until the *convergence* criteria are met beyond the burn-in phase.

## Input and output locations

The paths of the various inputs and outputs are set by a few different options in `MOSFiT`. The first time `MOSFiT` runs in a directory, it will make local copies of the `models` and `jupyter` folders distributed with the code (unless `--no-copy-at-launch` option is passed), and will *not* copy the files again unless they are deleted or the user passes the `--force-copy-at-launch` option.

By default, `MOSFiT` searches the local `models` folder copied to the run directory to find model JSON and their corresponding parameter JSON files to use for runs. If the user wishes to use custom parameter files for their runs instead, they can specify the paths to these files using the `-P` option.

`MOSFiT` outputs are always written to a local `products` directory, with the default filename being set to the name of the transient being fit (e.g. `LSQ12dlf.json` for LSQ12dlf). The user can append a suffix to the output filename using the `-s` option, e.g.:

```
mosfit -e LSQ12dlf -m slsn -s mysuffix
```

will write to the file `LSQ12dlf-mysuffix.json`. A copy of the output will also always be dumped to `walkers.json` in the same directory. The same suffix will applied to any additional outputs requested by the user, such as the `chain.json` and `extras.json` files.

## Fixing model parameters

Individual parameters can be locked to fixed values with the `-F` option, which will either assume the default specified in the model JSON file (if no value is provided):

```
mosfit -e LSQ12dlf -m slsn -F kappa
```

Or, will assume the value specified by the user:

```
mosfit -e LSQ12dlf -m slsn -F mejecta 3.0
```

Multiple fixed variables can be specified by chaining them together, with any user-prescribed variables following the variable names:

```
mosfit -e LSQ12dlf -m slsn -F kappa mejecta 3.0
```

If you have a prior for a given variable (not a single value), it is best to modify your local `parameters.json` file. For instance, to place a Gaussian prior on `vejecta` in the SLSN model, replace the default `parameters.json` snippet, which looks like this:

```
"vejecta":{
    "min_value":5.0e3,
    "max_value":2.0e4
},
```

with the following:

```
"vejecta":{
    "class":"gaussian",
    "mu":1.0e4,
    "sigma":0.5e3,
    "min_value":1.0e3,
    "max_value":1.0e5
},
```

Flat, log flat, gaussian, and power-law priors are available in `MOSFiT`; see the parameters_test.json file in the `default` model for examples on how to set each prior type.

### Initializing from previous runs

The user can use the ensemble parameters from a prior `MOSFiT` run to draw their initial conditions for a new run using the `-w` option. Assuming that `LSQ12dlf_mysuffix.json` contains results from a previous run, the user can draw walker positions from it by passing it to the `-w` option:

```
mosfit -e LSQ12dlf -m slsn -w LSQ12dlf_suffix.json
```

If the file contains more walkers than requested by the new run, walker positions will be drawn verbatim from the input file, otherwise walker positions will be "jittered" by a small amount so no two walkers share identical parameters.

## Outputs

The model structure used in `MOSFiT` makes it ammenable to producing outputs from models that need not be fit against any particular transient. In this section we walk through how the user can extract various data products.

### Light curve options

By default, `MOSFiT` will only compute model observations at the times a particular transient was observed using the instrument for which it was observed at those times. If a transient is sparsely sampled, this will likely result in a choppy light curve with no prediction for intra-observation magnitudes/fluxes.

### Smooth light curves

A smooth output light curve can be produced using the `-S` option, which when passed no argument returns the light curve with *every* instrument's predicted observation at all times. If given an argument (e.g. `-S 100`), `MOSFiT` will return every instrument's predicted observation at all times *plus* an additional $S$ observations between the first and last observation.

### Extrapolated light curves

If the user wishes to extrapolate beyond the first and last observations, the `-E` option will extend the predicted observations by $E$ days both before and after the first/last detections.

### Predicted observations that were not observed

The user may wish to generate light curves for a transient in instruments/bands for which the transient was not observed; this can be accomplished using the `--extra-bands`, `extra-instruments`, `extra-bandsets`, and `extra-systems` options. For instance, to generate LCs in Hubble's UVIS filter F218W in the Vega system in addition to the observed bands, the user would enter:

```
mosfit -e LSQ12dlf -m slsn --extra-instruments UVIS --extra-bands F218W --extra-
↪systems Vega
```

## Mock light curves in a magnitude-limited survey

Generating a light curve from a model in `MOSFiT` is achieved by simply not passing any event to the code with the `-e` option. The command below will dump out a default number of parameter draws to a `walkers.json` file in the `products` folder:

```
mosfit -m slsn
```

By default, these light curves will be the *exact* model predictions, they will not account for any observational error. If Gaussian Processes were used (by default they are enabled for all models), the output predictions will include an `e_magnitude` value that is set by the variance predicted by the GP model; if not, the `variance` parameter from maximum likelihood is used.

If the user wishes to produce mock observations for a given instrument, they should use the `-l` option, which sets a limiting magnitude and then randomly draws observations based upon the flux error implied by that limiting magnitude (the second argument to `-l` sets the variance of the limiting magnitude from observation to observation). For example, if the user wishes to generate mock light curves as they might be observed by LSST assuming a limiting magnitude of 23 for all bands, they would execute:

```
mosfit -m slsn -l 23 0.5 --extra-bands u g r i z y --extra-instruments LSST
```

## Saving the chain

Because the chain can be quite large (a full chain for a model with 15 free parameters, 100 walkers, and 20000 iterations will occupy ~120 MB of disk space), by default `MOSFiT` does not output the full chain to disk. Doing so is achieved by passing `MOSFiT` the `-c` option:

```
mosfit -m slsn -e LSQ12dlf -c
```

Note that the outputted chain includes both the burn-in and post-burn-in phases of the fitting procedure. The position of each walker in the chain as a function of time can be visualized using the included `mosfit.ipynb` Jupyter notebook.

Memory can be quite scarce on some systems, and storing the chain in memory can sometimes lead to out of memory errors (it is the dominant user of memory in `MOSFiT`). This can be mitigated to some extent by automatically thinning the chain if it gets too large with the `-M` option, where the argument to `-M` is in MB. Below, we limit the chain to a gigabyte, which should be sufficient for most modern systems:

```
mosfit -m slsn -e LSQ12dlf -M 1000
```

### Arbitrary outputs

Internally, `MOSFiT` is storing the outputs of each module in a single dictionary that is handed down through the execution tree like a hot potato. This dictionary behaves like a list of global variables, and when a model is executed from start to finish, it will be filled with values that were produced by all modules included in that module.

The user can dump any of these variables to a supplementary file `extras.json` by using the `-x` option, followed by the name of the variable of interest. For instance, if the user is interested in the spectral energy distributions and bolometric luminosities associated with the SLSN model of a transient, they can simply pass the `seds` and `dense_luminosities` keys to `-x`:

```
mosfit -m slsn -x seds dense_luminosities
```

## Using as a package

If you wish to produce light curves or other data products for a given model without using the fitting and evidence accumulation features of `MOSFiT`, functions within the code can be accessed by importing the `mosfit` package into your Python code.

### Produce model outputs

In the code snippet below, we fetch a supernova's data from the Open Catalogs using the `Fetcher` class, create a `Model` that initializes from the fetched data, and finally run the model:

```
import mosfit
import numpy as np

# Create an instance of the `Fetcher` class.
my_fetcher = mosfit.fetcher.Fetcher()

# Fetch some data from the Open Supernova Catalog.
fetched = my_fetcher.fetch('SN2009do')[0]

# Instantiatiate the `Model` class (selecting 'slsn' as the model).
my_model = mosfit.model.Model(model='slsn')

# Load the fetched data into the model.
my_model.load_data(fetched['data'], event_name=fetched['name'])

# Generate a random input vector of free parameters.
x = np.random.rand(my_model.get_num_free_parameters())

# Produce model output.
```

```
outputs = my_model.run(x)
print('Keys in output: `{}`'.format(', '.join(list(outputs.keys()))))
```

# Assessing models

## Convergence

Convergence in MOSFiT is assessed using the Gelman-Rubin statistic (or "potential scale reduction factor", abbreviated PSRF), which is a measure of the in-chain variance as compared to the between-chain variance. This metric is calculated for each free parameter, with the global PSRF score being derived by taking the maximum difference amongst all the individual parameter PSRFs. If a model is converged and well-mixed, these two values should be close to equal (PSRF ~ 1), and any significant deviance from equality suggests that the chains have yet to converge.

By default, MOSFiT will run for a small, fixed number of iterations (`-i 5000`), regardless of how well-converged a model is, to guarantee the total runtime is deterministic. If however the `-R` option is passed to MOSFiT, the code will continue to evolve the chains beyond the iteration limit specified by `-i` until the PSRF is less than a prescribed value (by default 1.1, unless the user sets another value using `-R`).

Another measure of convergence is the autocorrelation time $\tau_{\rm auto}$, estimated using the `acor` function embedded within `emcee`. Unfortunately, this metric usually does not give an indication of how close one is to convergence until one is already converged, as it fails to yield an estimate for the autocorrelation time if $\tau_{\rm auto} > i$, where $i$ is the number of iterations. We find that typically chains must run for significantly longer than what is required to converge according to the PSRF before `acor` will yield a numerical value (`-R 1.05` or less).

The fact that `acor` does not yield a value until the PSRF ~ 1 means that the number of independent draws from the posterior is significantly constrained unless the user chooses to run their chains for much longer. MOSFiT can be instructed to run until a certain number of independent samples are available via the `-U` option.

## Scoring

Model compatibility with a given dataset is measured using the "Watanabe-Akaike information criterion" (WAIC, also known as the "widely applicable information criterion", *[WAT2010]*), which is simply the score of the parameter combination with the highest likelihood minus the variance of the scores within the fully-converged posterior. Ideally, one prefers models with the fewest free parameters, the WAIC estimates the *effective* number of free parameters for a given model and adjusts the score accordingly. In principle, two models with the same score for their best fits may have wildly different WAIC scores depending on the distribution of scores within their posteriors. This criterion is less sensitive to overfitting than simply comparing the best scores yielded by two models, and should also provide a fair comparisson between models with different numbers of free parameters.

# Model errors

The choice of error model within a model can affect the score a given physical model receives; an error model that better treats the expected errors (either on the model or observation side) can thus enable a better evaluation of whether a model is a good match to a given set of observations. Commonly, no error modeling is done whatsoever, with a model's fitness being judged solely upon its deviance from the observations and their reported errors (i.e. reduced chi-square).

But what if the model itself has some uncertainty? For semi-analytical approximations of complicated phenomena, most assuredly the models possess some intrinsic error. These errors may be evident in a number of ways: Perhaps a given model cannot produce enough light at a particular frequency, or has an evolution that is not fully captured by the

approximation. As all semi-analytical models are prone to such issues, how do we compare two models with different (and unknown) deficiences to a given dataset?

## Maximum likelihood analysis

Maximum likelihood analysis (MLA) is a simple way to include the error directly in the modeling. In MLA, a variance parameter $\sigma$ is added to every observation. Because the chi-square metric includes $\sigma$ in its denominator, the increase of $\sigma$ comes with a cost to the overall score a model receives. As a result, optimizations of such a model will always trend towards solutions where $\chi^2_{\rm red} \to 1$. The output of MLA thus answers the question of "How much additional error do I need to add to my model/observations to make the model and observations consistent with one another?"

But MLA is rather inflexible, in order to match a model to observations, it must (by construction) increase the variance for *all* observations simultaneously. For most models, this is probably overkill: the models likely deviate in *some* colors, at *some* times. What's more, MLA only allows for the white noise component of the error to expand to accomodate a model, in reality there's likely to be systematic offsets between models and data that leads to *covariant* errors.

As described below, Gaussian processes are the default error model used in `MOSFiT`. To revert to MLA the user should fix the covariance parameters using `-F covariance`.

## Gaussian processes

Gaussian processes (GP) provides an error model that addresses these shortcomings of MLA. A white noise component, equivalent to MLA, is still included, but off-diagonal covariance is explicitly modeled by considering the "distance" between observations. `MOSFiT` by default uses GP as its error model, with a kernel structure that is described below.

### Kernel

The default kernel is chosen specifically to be ammenable to fitting photometric light curves. The kernel is constructed as a product of two exponential squared kernels, with the distance factors being the time of observation and the average wavelength of the filter used for the observation,

$$K_{ij} = \sigma^2 K_{ij,t} K_{ij,\lambda} + \mathrm{diag}(\sigma_i^2)$$
$$K_{ij,t} = \exp\left(-\frac{[t_i - t_j]^2}{2l_t^2}\right)$$
$$K_{ij,\lambda} = \exp\left(-\frac{[\lambda_i - \lambda_j]^2}{2l_\lambda^2}\right)$$

where $\sigma$ is the extra variance (analogous to the variance in MLA), $\sigma_i$ is the observation error of the $i\mathrm{th}$ observation, $t$ is the time of observation, and $\lambda$ is the mean wavelength of the observed band.

### Shortcomings

Gaussian processes can sometimes be too accomodating, explaining the entirety of the temporal evolution via random variation. Using the kernel described above, such a model match would present extremely long time and/or wavelength covariance lengths. This is often indicative that a given physical model is a poor representation of a given transient, or that a model is underconstrained (i.e. if the number of datapoints is comparable to the number of free parameters).

# Accessibility

## Language

MOSFiT can optionally translate all of its command line text into any language supported by Google translate. MOSFiT will use a user's $LANG environment variable to guess the language to use, if this variable is set. To accomplish this, the user must install the googletrans via pip:

```
pip install googletrans
```

Then, MOSFiT can be translated into one of the available languages using the --language option. When running for the first time for a new language, MOSFiT will pass all strings to the googletrans package one by one, which takes a few minutes to return the translated strings.

Note that Google's translation service is very approximate, and the translated text is only roughly equivalent to the original meaning.

# Frequently Asked Questions

## What do I do if MOSFiT or one of its requirements isn't installing?

We highly recommend using conda to install MOSFiT rather than pip, as conda will skip some compilation steps that are common sources of error in the install process. If you are still having issues installing MOSFiT even with conda, please ask us directly in the MOSFiT Slack channel.

## What can I try if MOSFiT won't run?

If MOSFiT is the first conda program you've used, and you previously used your system's built-in Python install, your shell environment may still be set up for your old Python setup, which can cause problems both for MOSFiT and your old Python programs. One common issue is that your PYTHON_PATH environment variable might be set to your build-in Python's install location, this will supercede conda's paths and potentially cause issues. Edit your .bashrc or .profile file to remove any PYTHON_PATH variable declarations, this will prevent path conflicts.

## Is MOSFiT using the correct data?

If private data is not provided to MOSFiT, it will draw data from the Open Astronomy Catalogs (the Open Supernova Catalog and the Open Tidal Disruption Catalog). These catalogs are constructed by combining data from hundreds of individual sources, any one of which could have had an issue when being imported into the OACs. If you suspect the data contained for a transient on one of these catalogs is incorrect, please open an issue on the appropriate catalog repository (links to the repositories are available on the AstroCats homepage) and the error will be corrected ASAP.

If you must correct the error immediately, feel free to copy the input file downloaded by MOSFiT (saved in a cache directory, the location of which is printed by MOSFiT when it runs) to your run directory and edit it on your own computer to fix the errors. But please *also* report the errors on the above issues pages so that the whole community will benefit!

## Can I fit private data with MOSFiT?

Yes! Simply pass your ASCII datafile to the -e flag instead of the name of the transient you wish to fit. Your data will remain private unless you choose to upload it with the optional -u flag, which will warn you before any data is

uploaded publicly. More info on fitting private data can be found *here*.

### How do I exclude particular instruments/bands/sources from my fit?

Excluding instruments can be accomplished by using the `--exclude-instruments` option, and excluding bands can be accomplished using the `--exclude-bands` option. All the data from a particular source (e.g. a paper or survey) can be excluded using `--exclude-sources` (see *here* for more information on restricting your dataset). More complicated exclusion rules (say ignoring a particular band from a particular instrument, but not for other instruments) are most easily accomplished by simply deleting the unwanted data from the input file; users should copy the cached version downloaded from the Open Astronomy Catalogs to their run directory and edit the files to remove the data.

## Command line arguments

Below are descriptions of the command line arguments available for `MOSFiT`.

### mosfit

Fit astrophysical transients.

```
usage: mosfit [-h] [--events EVENTS [EVENTS ...]] [--models [MODELS]]
                  [--parameter-paths PARAMETER_PATHS [PARAMETER_PATHS ...]]
                  [--walker-paths WALKER_PATHS [WALKER_PATHS ...]]
                  [--max-time MAX_TIME]
                  [--limiting-magnitude LIMITING_MAGNITUDE [LIMITING_MAGNITUDE ...]]
                  [--band-list BAND_LIST [BAND_LIST ...]]
                  [--band-systems BAND_SYSTEMS [BAND_SYSTEMS ...]]
                  [--band-instruments BAND_INSTRUMENTS [BAND_INSTRUMENTS ...]]
                  [--band-bandsets BAND_BANDSETS [BAND_BANDSETS ...]]
                  [--band-sampling-points BAND_SAMPLING_POINTS]
                  [--exclude-bands EXCLUDE_BANDS [EXCLUDE_BANDS ...]]
                  [--exclude-instruments EXCLUDE_INSTRUMENTS [EXCLUDE_INSTRUMENTS ...
↪]]
                  [--exclude-systems EXCLUDE_SYSTEMS [EXCLUDE_SYSTEMS ...]]
                  [--exclude-sources EXCLUDE_SOURCES [EXCLUDE_SOURCES ...]]
                  [--fix-parameters USER_FIXED_PARAMETERS [USER_FIXED_PARAMETERS ...]]
                  [--iterations [ITERATIONS]] [--smooth-times [SMOOTH_TIMES]]
                  [--extrapolate-time [EXTRAPOLATE_TIME [EXTRAPOLATE_TIME ...]]]
                  [--limit-fitting-mjds LIMIT_FITTING_MJDS LIMIT_FITTING_MJDS]
                  [--suffix SUFFIX] [--num-walkers NUM_WALKERS]
                  [--num-temps NUM_TEMPS] [--no-fracking] [--no-write] [--quiet]
                  [--cuda] [--no-copy-at-launch] [--force-copy-at-launch]
                  [--offline] [--frack-step FRACK_STEP] [--burn BURN]
                  [--post-burn POST_BURN] [--upload]
                  [--run-until-converged [RUN_UNTIL_CONVERGED]]
                  [--run-until-uncorrelated [RUN_UNTIL_UNCORRELATED]]
                  [--maximum-walltime MAXIMUM_WALLTIME]
                  [--maximum-memory MAXIMUM_MEMORY]
                  [--draw-above-likelihood [DRAW_ABOVE_LIKELIHOOD]] [--gibbs]
                  [--save-full-chain] [--print-trees]
                  [--set-upload-token [SET_UPLOAD_TOKEN]]
                  [--ignore-upload-quality] [--test]
                  [--variance-for-each VARIANCE_FOR_EACH [VARIANCE_FOR_EACH ...]]
```

```
                [--speak [SPEAK]] [--version] [--language [LANGUAGE]]
                [--extra-outputs EXTRA_OUTPUTS [EXTRA_OUTPUTS ...]]
                [--catalogs CATALOGS [CATALOGS ...]] [--open-in-browser]
                [--exit-on-prompt] [--download-recommended-data]
```

**-h, --help**
show this help message and exit

**--events** <events>, **-e** <events>
List of event names (or file names) to be fit, delimited by spaces. If an event name contains a space, enclose the event's name in double quote marks, e.g. "SDSS-II SN 5944". Files with *.json* extensions are presumed to be in Open Catalog format, whereas files with any other extension will be read as a list of event names.

**--models** <models>, **-m** <models>
List of models to use to fit against the listed events. The model can either be a name of a model included with MOSFiT, or a path to a custom model JSON file generated by the user.

**--parameter-paths** <parameter_paths>, **-P** <parameter_paths>
Paths to parameter files corresponding to each model file; length of this list should be equal to the length of the list of models

**--walker-paths** <walker_paths>, **-w** <walker_paths>
List of paths to Open Catalog format files with walkers from which to draw initial walker positions. Output data from MOSFiT can be loaded with this command. If some variables are not contained within the input file(s), they will instead be drawn randomly from the specified model priors.

**--max-time** <max_time>
Set the maximum time for model light curves to be plotted until.

**--limiting-magnitude** <limiting_magnitude>, **-l** <limiting_magnitude>
Assumed limiting magnitude of a simulated survey. When enabled, model light curves will be randomly drawn and assigned error bars. If passed one argument, that number will be used as the limiting magnitude (default: *20*). If provided a second argument, that number will be used for observation-to-observation variance in the limit.

**--band-list** <band_list>, **--extra-bands** <band_list>
List of additional bands to plot when plotting model light curves that are not being matched to actual transient data.

**--band-systems** <band_systems>, **--extra-systems** <band_systems>
List of photometric systems corresponding to the bands listed in *–band-list*.

**--band-instruments** <band_instruments>, **--extra-instruments** <band_instruments>
List of instruments corresponding to the bands listed in *–band-list*.

**--band-bandsets** <band_bandsets>, **--extra-bandsets** <band_bandsets>
List of bandsets corresponding to the bands listed in *–band-list*.

**--band-sampling-points** <band_sampling_points>
Number of wavelengths to sample in each band when modeling photometry.

**--exclude-bands** <exclude_bands>
List of bands to exclude in fitting.

**--exclude-instruments** <exclude_instruments>
List of instruments to exclude in fitting corresponding to the bands listed in *–exclude-bands*.

**--exclude-systems** <exclude_systems>
List of systems to exclude in fitting corresponding to the bands listed in *–exclude-bands*.

**--exclude-sources** <exclude_sources>
List of references to exclude data from when fitting. These are specified using the source ID number that is shown on the Open Astronomy Catalog page for each transient.

**--fix-parameters** <user_fixed_parameters>, **-F** <user_fixed_parameters>
Pairs of parameter names and values to fix for the current fit. Example: *-F kappa 1.0 vejecta 1.0e4* would fix the *kappa* and *vejecta* parameters to those values. If the second value is recognized to be an existing key, the whole list will be assumed to just be a list of keys and the default values specified in the model JSON files will be used. If the name is a parameter class (e.g. *covariance*), all variables of that class will be fixed.

**--iterations** <iterations>, **-i** <iterations>
Number of iterations to run emcee for, including burn-in and post-burn iterations. Setting this option to *0* (or providing no argument) will only draw walker positions and immediately exit.

**--smooth-times** <smooth_times>, **--plot-points** <smooth_times>, **-S** <smooth_times>
Add this many more fictitious observations between the first and last observed times. Setting this value to *0* (or providing no argument) will guarantee that all observed bands/instrument/system combinations have a point at all observed epochs, but no other times. A negative value will only yield model predictions at the observations but at no other times (faster but sparser light curves).

**--extrapolate-time** <extrapolate_time>, **-E** <extrapolate_time>
Extend model light curves this many days before/after first/last observation. Can be a list of two elements, in which case the first element is the amount of time before the first observation to extrapolate, and the second element is the amount of time before the last observation to extrapolate. Value is set to *0.0* days if option not set, *100.0* days by default if no arguments are given.

**--limit-fitting-mjds** <limit_fitting_mjds>, **-L** <limit_fitting_mjds>
Only include observations with MJDs within the specified range, e.g. *-L 54123 54234* will exclude observations outside this range. If specified without an argument, any upper limit observations before the last upper limit before the first detection in a given band will not be included in the fitting.

**--suffix** <suffix>, **-s** <suffix>
Append custom string to output file name to prevent overwrite

**--num-walkers** <num_walkers>, **-N** <num_walkers>
Number of walkers to use in emcee. When fitting, this must be set to at least twice the total number of free parameters within the model, not setting this parameter will set it to this minimum.

**--num-temps** <num_temps>, **-T** <num_temps>
Number of temperatures to use in the parallel-tempered emcee sampler. *-T 1* is equivalent to the standard EnsembleSampler.

**--no-fracking**
Setting this flag will skip the *fracking* step of the optimization process.

**--no-write**
Do not write any results to disk.

**--quiet**
Print minimal output upon execution. Don't display our amazing logo :-(

**--cuda**
Enable CUDA for MOSFiT routines. Requires the *scikit-cuda* package (and its dependencies) to be installed.

**--no-copy-at-launch**
Setting this flag will prevent MOSFiT from copying the user file hierarchy (models/modules/jupyter) to the current working directory before fitting.

**--force-copy-at-launch**
Setting this flag will force MOSFiT to overwrite the user file hierarchy (models/modules/jupyter) to the current working directory. User will be prompted before being allowed to run with this flag.

**--offline**
MOSFiT will only use cached data and will not attempt to use any online resources.

**--frack-step** <frack_step>, **-f** <frack_step>
Perform *fracking* every this number of steps while in the burn-in phase of the fitting process.

**--burn** <burn>, **-b** <burn>
Burn in the chains for this many iterations. During burn-in, global optimization ("fracking"), replacement, and a Gibbs variant of emcee are used to speed convergence. However, as none of these methods preserve detailed balance, the posteriors obtained during the burn-in phase are very approximate. No convergence information will be displayed during burn-in.

**--post-burn** <post_burn>, **-p** <post_burn>
Run emcee this many more iterations after the burn-in phase. The burn-in phase will thus be run for (i - p) iterations, where i is the total number of iterations set with *-i* and p is the value of this parameter.

**--upload, -u**
Upload results of MOSFiT to appropriate Open Catalog. If MOSFiT is only supplied with *-u* and no other arguments, it will upload the results of the latest run.

**--run-until-converged** <run_until_converged>, **-R** <run_until_converged>
Run each model until the autocorrelation time is measured accurately and chain has burned in for the specified number of autocorrelation times [Default: 10.0]. This will run beyond the specified number of iterations, and is recommended when the *–upload/-u* flag is set.

**--run-until-uncorrelated** <run_until_uncorrelated>, **-U** <run_until_uncorrelated>
Run each model until the autocorrelation time is measured accurately and chain has burned in for the specified number of autocorrelation times [Default: 10.0]. This will run beyond the specified number of iterations, and is recommended when the *–upload/-u* flag is set.

**--maximum-walltime** <maximum_walltime>, **-W** <maximum_walltime>
Total execution time (in seconds) constrained to be no greater than this value.

**--maximum-memory** <maximum_memory>, **-M** <maximum_memory>
Maximum memory MOSFiT is allowed to use, in megabytes. The memory use is roughly estimated, so it is best to set this number at least 1 GB below your system's actual memory limit per CPU.

**--draw-above-likelihood** <draw_above_likelihood>, **-d** <draw_above_likelihood>
When randomly drawing walkers initially, do not accept a draw unless a likelihood value is greater than this value. By default, any score greater than the likelihood floor will be retained.

**--gibbs, -g**
Using a Gibbs-sampling variant of emcee. This is not proven to preserve detailed balance, however it has much faster convergence than the vanilla emcee stretch-move. Use with caution.

**--save-full-chain, -c**
Save the full chain for each model fit.

**--print-trees**
Print the full dependency trees of each model.

**--set-upload-token** <set_upload_token>
Set the upload token. If given an argument, expects a 64-character token. If given no argument, MOSFiT will prompt the user to provide a token.

**--ignore-upload-quality**
Ignore all quality checks when uploading fits.

**--test**
Alters the printing of output messages such that a new line is generated with each message. Users are unlikely to need this parameter; it is included as Travis requires new lines to be produced to detected program output.

**--variance-for-each** <variance_for_each>
Create a separate *Variance* for each type of observation specified. Currently *band* is the only valid option.

**--speak** <speak>
Speak.

**--version**
Print code version info.

**--language** <language>
Language for output text.

**--extra-outputs** <extra_outputs>, **-x** <extra_outputs>
Extra keys to save alongside the default model outputs.

**--catalogs** <catalogs>, **-C** <catalogs>
Restrict data acquisition to the listed catalogs.

**--open-in-browser, -O**
Open the events listed with *-e* in the user's web browser one at a time.

**--exit-on-prompt**
Exit immediately if any user prompts are encountered (useful for batch jobs).

**--download-recommended-data**
Downloads any recommended data from the Open Catalogs if notprovided by the user (without prompting).

# API

## mosfit.modules Package

### Classes

| | |
|---|---|
| *Module*(name, model, **kwargs) | Base `Module` class. |
| *Diagonal*(**kwargs) | Calculate the diagonal/residuals for a model kernel. |
| *Kernel*(**kwargs) | Calculate the maximum likelihood score for a model. |
| *DenseTimes*(**kwargs) | Generate an evenly-spaced array of times for use in calculations. |
| *Array*(name, model, **kwargs) | Template class for arrays. |
| *RestTimes*(name, model, **kwargs) | This class converts the observed times to rest-frame times. |
| *AllTimes*(**kwargs) | Generate all times for which observations will be constructed. |
| *Constraint*(name, model, **kwargs) | Template class for constraints. |
| *CSMConstraints*(name, model, **kwargs) | CSM constraints. |
| *SLSNConstraints*(**kwargs) | SLSN constraints. |
| *MagnetarConstraints*(**kwargs) | Magnetar constraints. |
| *TDEConstraints*(**kwargs) | TDE constraints. |
| *Transient*(**kwargs) | Structure to store transient data. |
| *HomologousExpansion*(name, model, **kwargs) | Generate *vejecta* from *kinetic_energy* assuming homologous expansion. |
| *ThinShell*(name, model, **kwargs) | Generate *vejecta* from *kinetic_energy* if ejecta in thin shell. |
| *Energetic*(name, model, **kwargs) | Template class for energy/velocity conversions. |
| *RProcess*(**kwargs) | r-process decay engine. |

Continued on next page

Table 1.1 – continued from previous page

| | |
|---|---|
| *Engine*(**kwargs) | Generic engine module. |
| *Magnetar*(**kwargs) | Magnetar spin-down engine. |
| *CSM*(**kwargs) | CSM energy injection. |
| *ExpPow*(**kwargs) | A simple analytical engine. |
| *Fallback*(**kwargs) | A tde engine. |
| *NickelCobalt*(**kwargs) | Nickel/Cobalt decay engine. |
| *Likelihood*(name, model, **kwargs) | Calculate the maximum likelihood score for a model. |
| *Photometry*(**kwargs) | Band-pass filters. |
| *Write*(**kwargs) | Write keys to disk. |
| *Output*(name, model, **kwargs) | Template class for output Modules. |
| *LightCurve*(**kwargs) | Output a light curve to disk. |
| *Parameter*(**kwargs) | Model parameter that can either be free or fixed. |
| *Redshift*(**kwargs) | Redshift parameter that depends on luminosity distance. |
| *PowerLaw*(**kwargs) | Standard power law, alpha must be > 1. |
| *Variance*(**kwargs) | Model parameter that can either be free or fixed. |
| *Constant*(**kwargs) | Constant parameter. |
| *Gaussian*(**kwargs) | Parameter with Gaussian prior. |
| *Covariance*(**kwargs) | Model parameter that can either be free or fixed. |
| *LuminosityDistance*(**kwargs) | LuminosityDistance parameter that depends on luminosity distance. |
| *TemperatureFloor*(name, model, **kwargs) | Photosphere with a minimum allowed temperature. |
| *TdePhotosphere*(name, model, **kwargs) | Photosphere for a tidal disruption event. |
| *Photosphere*(name, model, **kwargs) | Template class for photosphere Modules. |
| *DenseCore*(name, model, **kwargs) | Photosphere with a dense core and a low-mass envelope. |
| *Blackbody*(**kwargs) | Blackbody spectral energy dist. |
| *Line*(**kwargs) | Line spectral energy distribution, modifies existing SED. |
| *MultiBlackbody*(**kwargs) | Generalized multiple blackbody spectral energy distribution. |
| *LOSExtinction*(**kwargs) | Adds extinction to SED from both host galaxy and MW. |
| *SED*(**kwargs) | Template class for SED Modules. |
| *Synchrotron*(**kwargs) | Synchrotron spectral energy distribution. |
| *BlackbodyCutoff*(**kwargs) | Blackbody SED with cutoff. |
| *Diffusion*(**kwargs) | Photon diffusion transform. |
| *DiffusionCSM*(**kwargs) | Photon diffusion transform for CSM model. |
| *Viscous*(**kwargs) | Viscous delay transform. |
| *Transform*(**kwargs) | Parent class for transforms. |

### Module

**class** mosfit.modules.**Module**(*name*, *model*, ***kwargs*)

Bases: object

Base Module class.

#### Methods Summary

| | |
|---|---|
| *dense_key*(key) | Manipulate output keys conditionally. |
| *get_bibcode*() | Return any bibcodes associated with the present Module. |

Continued on next page

Table 1.2 – continued from previous page

| | |
|---|---|
| *get_unset_recommended_keys*() | Return list of recommended keys that are not set. |
| *key*(key) | Substitute user-defined replacement key names for local names. |
| *name*() | Return own name. |
| *prepare_input*(key, **kwargs) | Prepare keys conditionally. |
| *process*(**kwargs) | Process module, should always return a dictionary. |
| *receive_requests*(**requests) | Receive requests from other `Module` objects. |
| *reset_preprocessed*(exceptions) | Reset preprocessed flag. |
| *reset_unset_recommended_keys*() | Null the list of unset recommended keys. |
| *send_request*(request) | Send a request. |
| *set_attributes*(task) | Set key replacement dictionary. |
| *set_event_name*(event_name) | Set the name of the event being modeled. |

### Methods Documentation

**dense_key**(*key*)
Manipulate output keys conditionally.

**get_bibcode**()
Return any bibcodes associated with the present `Module`.

**get_unset_recommended_keys**()
Return list of recommended keys that are not set.

**key**(*key*)
Substitute user-defined replacement key names for local names.

**name**()
Return own name.

**prepare_input**(*key*, ***kwargs*)
Prepare keys conditionally.

**process**(***kwargs*)
Process module, should always return a dictionary.

**receive_requests**(***requests*)
Receive requests from other `Module` objects.

**reset_preprocessed**(*exceptions*)
Reset preprocessed flag.

**reset_unset_recommended_keys**()
Null the list of unset recommended keys.

**send_request**(*request*)
Send a request.

**set_attributes**(*task*)
Set key replacement dictionary.

**set_event_name**(*event_name*)
Set the name of the event being modeled.

### Diagonal

**class** mosfit.modules.**Diagonal**(***kwargs*)
Bases: *mosfit.modules.Array*

Calculate the diagonal/residuals for a model kernel.

#### Attributes Summary

| |
|---|
| *MIN_COV_TERM* |

#### Methods Summary

| | |
|---|---|
| *preprocess*(**kwargs) | Construct arrays of observations based on data keys. |
| *process*(**kwargs) | Process module. |

#### Attributes Documentation

**MIN_COV_TERM = 1e-30**

#### Methods Documentation

**preprocess**(***kwargs*)
Construct arrays of observations based on data keys.

**process**(***kwargs*)
Process module.

### Kernel

**class** mosfit.modules.**Kernel**(***kwargs*)
Bases: *mosfit.modules.Array*

Calculate the maximum likelihood score for a model.

#### Attributes Summary

| |
|---|
| *MIN_COV_TERM* |

#### Methods Summary

| | |
|---|---|
| *preprocess*(**kwargs) | Construct kernel distance arrays. |
| *process*(**kwargs) | Process module. |
| *receive_requests*(**requests) | Receive requests from other `Module` objects. |

### Attributes Documentation

**MIN_COV_TERM = 1e-30**

### Methods Documentation

**preprocess**(***kwargs*)
Construct kernel distance arrays.

**process**(***kwargs*)
Process module.

**receive_requests**(***requests*)
Receive requests from other `Module` objects.

## DenseTimes

**class** mosfit.modules.**DenseTimes**(***kwargs*)
Bases: *mosfit.modules.Array*

Generate an evenly-spaced array of times for use in calculations.

This class ensures an even time-sampling between the time of explosion and the last datapoint, as many transients may lack regular candence data.

### Attributes Summary

| |
|---|
| *L_T_MIN* |
| *N_TIMES* |

### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

### Attributes Documentation

**L_T_MIN = -6**

**N_TIMES = 100**

### Methods Documentation

**process**(***kwargs*)
Process module.

### Array

**class** mosfit.modules.**Array**(*name*, *model*, ***kwargs*)
Bases: *mosfit.modules.Module*

Template class for arrays.

### RestTimes

**class** mosfit.modules.**RestTimes**(*name*, *model*, ***kwargs*)
Bases: *mosfit.modules.Array*

This class converts the observed times to rest-frame times.

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Methods Documentation

**process**(***kwargs*)
Process module.

### AllTimes

**class** mosfit.modules.**AllTimes**(***kwargs*)
Bases: *mosfit.modules.Array*

Generate all times for which observations will be constructed.

Create lists of observation times that associated with real observations and interpolations/extrapolations if such flags are passed to MOSFiT.

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |
| *receive_requests*(**requests) | Receive requests from other Module objects. |

#### Methods Documentation

**process**(***kwargs*)
Process module.

**receive_requests**(***requests*)
Receive requests from other Module objects.

### Constraint

**class** mosfit.modules.**Constraint**(*name*, *model*, ***kwargs*)
Bases: *mosfit.modules.Module*

Template class for constraints.

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Methods Documentation

**process**(***kwargs*)
Process module.

### CSMConstraints

**class** mosfit.modules.**CSMConstraints**(*name*, *model*, ***kwargs*)
Bases: *mosfit.modules.Constraint*

CSM constraints.

1.R0 <= Rph <= Rcsm. The photospheric radius is within the CSM

2.td < ts. The diffusion time is less than the shock crossing time.

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Methods Documentation

**process**(***kwargs*)
Process module. Add constraints below.

### SLSNConstraints

**class** mosfit.modules.**SLSNConstraints**(***kwargs*)
Bases: *mosfit.modules.Constraint*

SLSN constraints.

1.Kinetic energy cannot excede magnetar rotational energy

2.Ejecta remain optically thick to thermal photons for at least 100d

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Methods Documentation

**process** (***kwargs*)
Process module. Add constraints below.

### MagnetarConstraints

**class** mosfit.modules.**MagnetarConstraints** (***kwargs*)
Bases: *mosfit.modules.Constraint*

Magnetar constraints.

Kinetic energy cannot excede magnetar rotational energy

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Methods Documentation

**process** (***kwargs*)
Process module. Add constraints below.

### TDEConstraints

**class** mosfit.modules.**TDEConstraints** (***kwargs*)
Bases: *mosfit.modules.Constraint*

TDE constraints.

1. rp > rs –> the pericenter radius must be greater than the Schwarzschild radius or the bh will swallow the star whole (no disruption or flare)

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Methods Documentation

**process** (***kwargs*)
Process module. Add constraints below.

### Transient

**class** mosfit.modules.**Transient**(***kwargs*)
Bases: *mosfit.modules.Module*

Structure to store transient data.

#### Methods Summary

| | |
|---|---|
| *get_data_determined_parameters*() | Return list of parameters determined by data. |
| *process*(**kwargs) | Process module. |
| *send_request*(request) | Send requests to other modules. |
| *set_data*(all_data[, req_key_values, ...]) | Set transient data. |

#### Methods Documentation

**get_data_determined_parameters**()
Return list of parameters determined by data.

**process**(***kwargs*)
Process module.

**send_request**(*request*)
Send requests to other modules.

**set_data**(*all_data*, *req_key_values={}*, *subtract_minimum_keys=[]*, *smooth_times=-1*, *extrapolate_time=0.0*, *limit_fitting_mjds=False*, *exclude_bands=[]*, *exclude_instruments=[]*, *exclude_systems=[]*, *exclude_sources=[]*, *band_list=[]*, *band_telescopes=[]*, *band_systems=[]*, *band_instruments=[]*, *band_modes=[]*, *band_bandsets=[]*)
Set transient data.

### HomologousExpansion

**class** mosfit.modules.**HomologousExpansion**(*name*, *model*, ***kwargs*)
Bases: *mosfit.modules.Energetic*

Generate *vejecta* from *kinetic_energy* assuming homologous expansion.

Convert an input kinetic energy to velocity assuming ejecta in homologous expansion.

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Methods Documentation

**process**(***kwargs*)
Process module.

### ThinShell

**class** mosfit.modules.**ThinShell**(*name*, *model*, ***kwargs*)
Bases: *mosfit.modules.Energetic*

Generate *vejecta* from *kinetic_energy* if ejecta in thin shell.

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Methods Documentation

**process**(***kwargs*)
Process module.

### Energetic

**class** mosfit.modules.**Energetic**(*name*, *model*, ***kwargs*)
Bases: *mosfit.modules.Module*

Template class for energy/velocity conversions.

### RProcess

**class** mosfit.modules.**RProcess**(***kwargs*)
Bases: *mosfit.modules.Engine*

r-process decay engine.

input luminosity adapted from Metzger 2016: 2016arXiv161009381M

For 'red' kilonovae, use kappa ~ 10. For 'blue' kilonovae, use kappa ~1.

#### Attributes Summary

| |
|---|
| *M_sun* |

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Attributes Documentation

**M_sun = 1.99e+33**

#### Methods Documentation

**process**(**kwargs)
Process module.

### Engine

**class** mosfit.modules.**Engine**(**kwargs)
Bases: *mosfit.modules.Module*

Generic engine module.

### Magnetar

**class** mosfit.modules.**Magnetar**(**kwargs)
Bases: *mosfit.modules.Engine*

Magnetar spin-down engine.

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Methods Documentation

**process**(**kwargs)
Process module.

### CSM

**class** mosfit.modules.**CSM**(**kwargs)
Bases: *mosfit.modules.Engine*

CSM energy injection.

input luminosity calculation based on http://adsabs.harvard.edu/abs/2012ApJ...746..121C with coefficients from http://adsabs.harvard.edu/abs/1982ApJ...258..790C

There are two major changes in the input luminosity from Chatzopoulos, Wheeler & Vinko (2012): 1. ti is set to a small number, rather than the time it takes the ejecta to reach the csm shell 2. you can fit/choose an efficiency factor between KE and luminosity

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Methods Documentation

**process** (***kwargs*)
Process module.

### ExpPow

**class** mosfit.modules.**ExpPow** (***kwargs*)
Bases: *mosfit.modules.Engine*

A simple analytical engine.

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Methods Documentation

**process** (***kwargs*)
Process module.

### Fallback

**class** mosfit.modules.**Fallback** (***kwargs*)
Bases: *mosfit.modules.Engine*

A tde engine.

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Methods Documentation

**process** (***kwargs*)
Process module.

### NickelCobalt

**class** mosfit.modules.**NickelCobalt** (***kwargs*)
Bases: *mosfit.modules.Engine*

Nickel/Cobalt decay engine.

#### Attributes Summary

| |
|---|
| *CO56_LIFE* |
| *CO56_LUM* |
| *NI56_LIFE* |
| *NI56_LUM* |

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Attributes Documentation

**CO56_LIFE = 111.3**

**CO56_LUM = 1.45e+43**

**NI56_LIFE = 8.8**

**NI56_LUM = 6.45e+43**

#### Methods Documentation

**process** (***kwargs*)
Process module.

### Likelihood

**class** mosfit.modules.**Likelihood** (*name*, *model*, ***kwargs*)
Bases: *mosfit.modules.Module*

Calculate the maximum likelihood score for a model.

#### Attributes Summary

| |
|---|
| *MIN_COV_TERM* |

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Attributes Documentation

**MIN_COV_TERM = 1e-30**

### Methods Documentation

**process**(***kwargs*)
Process module.

## Photometry

**class** mosfit.modules.**Photometry**(***kwargs*)
Bases: *mosfit.modules.Module*

Band-pass filters.

### Attributes Summary

| |
|---|
| *ANG_CGS* |
| *C_CGS* |
| *FLUX_STD* |
| *H_CGS* |
| *H_C_ANG_CGS* |

### Methods Summary

| | |
|---|---|
| *abmag*(eff_fluxes, offsets) | Convert fluxes into AB magnitude. |
| *average_wavelengths*([indices]) | Return average wavelengths for specified band indices. |
| *bands*([indices]) | Return the list of unique band names. |
| *find_band_index*(band[, telescope, ...]) | Find the index corresponding to the provided band information. |
| *instruments*([indices]) | Return the list of instruments. |
| *load_bands*(band_indices) | Load band files. |
| *preprocess*(**kwargs) | Preprocess module. |
| *process*(**kwargs) | Process module. |
| *send_request*(request) | Send requests to other modules. |
| *set_variance_bands*(band_pairs) | Set band (or pair of bands) that variance will be anchored to. |

### Attributes Documentation

**ANG_CGS = 1.0000000000000002e-08**

**C_CGS = 29979245800.0**

**FLUX_STD = 0.1088546414998**

**H_CGS = 6.62607004e-27**

**H_C_ANG_CGS = 1.986445824171758e-08**

#### Methods Documentation

**abmag** (*eff_fluxes*, *offsets*)
Convert fluxes into AB magnitude.

**average_wavelengths** (*indices=None*)
Return average wavelengths for specified band indices.

**bands** (*indices=None*)
Return the list of unique band names.

**find_band_index** (*band*, *telescope=''*, *instrument=''*, *mode=''*, *bandset=''*, *system=''*)
Find the index corresponding to the provided band information.

**instruments** (*indices=None*)
Return the list of instruments.

**load_bands** (*band_indices*)
Load band files.

**preprocess** (***kwargs*)
Preprocess module.

**process** (***kwargs*)
Process module.

**send_request** (*request*)
Send requests to other modules.

**set_variance_bands** (*band_pairs*)
Set band (or pair of bands) that variance will be anchored to.

### Write

**class** mosfit.modules.**Write** (***kwargs*)
Bases: *mosfit.modules.Output*

Write keys to disk.

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Methods Documentation

**process** (***kwargs*)
Process module.

### Output

**class** mosfit.modules.**Output** (*name*, *model*, ***kwargs*)
Bases: *mosfit.modules.Module*

Template class for output Modules.

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Methods Documentation

**process** (***kwargs*)
Process module.

### LightCurve

**class** mosfit.modules.**LightCurve** (***kwargs*)
Bases: *mosfit.modules.Output*

Output a light curve to disk.

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Methods Documentation

**process** (***kwargs*)
Process module.

### Parameter

**class** mosfit.modules.**Parameter** (***kwargs*)
Bases: *mosfit.modules.Module*

Model parameter that can either be free or fixed.

#### Methods Summary

| | |
|---|---|
| *fix_value*(value) | Fix value of parameter. |
| *fraction*(value[, clip]) | Return fraction given a parameter's value. |
| *get_derived_keys*() | Return list of keys that should be generated by this parameter. |
| *is_log*() | Return if *Parameter*'s value is stored as log10(value). |
| *latex*() | Return the LaTeX representation of the parameter. |
| *lnprior_pdf*(x) | Evaluate natural log of probability density function. |
| *prior_cdf*(u) | Evaluate cumulative density function. |
| *process*(**kwargs) | Process module. |
| *receive_requests*(**requests) | Receive requests from other Module objects. |
| *value*(f) | Return the value of the parameter in parameter's units. |

### Methods Documentation

**fix_value**(*value*)
Fix value of parameter.

**fraction**(*value*, *clip=True*)
Return fraction given a parameter's value.

**get_derived_keys**()
Return list of keys that should be generated by this parameter.

**is_log**()
Return if *Parameter*'s value is stored as log10(value).

**latex**()
Return the LaTeX representation of the parameter.

**lnprior_pdf**(*x*)
Evaluate natural log of probability density function.

**prior_cdf**(*u*)
Evaluate cumulative density function.

**process**(***kwargs*)
Process module.

Initialize a parameter based upon either a fixed value or a distribution, if one is defined.

**receive_requests**(***requests*)
Receive requests from other `Module` objects.

**value**(*f*)
Return the value of the parameter in parameter's units.

## Redshift

**class** `mosfit.modules.`**Redshift**(***kwargs*)
Bases: *mosfit.modules.Parameter*

Redshift parameter that depends on luminosity distance.

### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |
| *receive_requests*(**requests) | Receive requests from other `Module` objects. |
| *send_request*(request) | Send requests to other modules. |

### Methods Documentation

**process**(***kwargs*)
Process module.

**receive_requests**(***requests*)
Receive requests from other `Module` objects.

**send_request**(*request*)

Send requests to other modules.

### PowerLaw

**class** mosfit.modules.**PowerLaw**(**kwargs)
Bases: *mosfit.modules.Parameter*

Standard power law, alpha must be > 1.

#### Methods Summary

| | |
|---|---|
| *lnprior_pdf*(x) | Evaluate natural log of probability density function. |
| *prior_cdf*(u) | Evaluate cumulative density function. |

#### Methods Documentation

**lnprior_pdf**(*x*)
Evaluate natural log of probability density function.

**prior_cdf**(*u*)
Evaluate cumulative density function.

### Variance

**class** mosfit.modules.**Variance**(**kwargs)
Bases: *mosfit.modules.Parameter*

Model parameter that can either be free or fixed.

### Constant

**class** mosfit.modules.**Constant**(**kwargs)
Bases: *mosfit.modules.Parameter*

Constant parameter.

*Parameter* that will throw an error if the user attempts to make the variable free.

### Gaussian

**class** mosfit.modules.**Gaussian**(**kwargs)
Bases: *mosfit.modules.Parameter*

Parameter with Gaussian prior.

If the parameter must be positive, set the *pos* keyword to True.

#### Methods Summary

| | |
|---|---|
| *lnprior_pdf*(x) | Evaluate natural log of probability density function. |
| *prior_cdf*(u) | Evaluate cumulative density function. |

#### Methods Documentation

**lnprior_pdf**(*x*)
Evaluate natural log of probability density function.

**prior_cdf**(*u*)
Evaluate cumulative density function.

### Covariance

**class** mosfit.modules.**Covariance**(***kwargs*)
Bases: *mosfit.modules.Parameter*

Model parameter that can either be free or fixed.

### LuminosityDistance

**class** mosfit.modules.**LuminosityDistance**(***kwargs*)
Bases: *mosfit.modules.Parameter*

LuminosityDistance parameter that depends on luminosity distance.

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |
| *receive_requests*(**requests) | Receive requests from other `Module` objects. |
| *send_request*(request) | Send requests to other modules. |

#### Methods Documentation

**process**(***kwargs*)
Process module.

**receive_requests**(***requests*)
Receive requests from other `Module` objects.

**send_request**(*request*)
Send requests to other modules.

### TemperatureFloor

**class** mosfit.modules.**TemperatureFloor**(*name*, *model*, ***kwargs*)
Bases: *mosfit.modules.Photosphere*

Photosphere with a minimum allowed temperature.

Photosphere that expands and cools with ejecta then recedes at constant final temperature.

#### Attributes Summary

| | |
|---|---|
| *RAD_CONST* | |
| *STEF_CONST* | |

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Attributes Documentation

**RAD_CONST = 8640000000.0**

**STEF_CONST = 0.0007125593324143198**

#### Methods Documentation

**process** (***kwargs*)
Process module.

### TdePhotosphere

**class** mosfit.modules.**TdePhotosphere** (*name*, *model*, ***kwargs*)
Bases: *mosfit.modules.Photosphere*

Photosphere for a tidal disruption event.

Photosphere that expands/recedes as a power law of Mdot (or equivalently L (proportional to Mdot) ).

#### Attributes Summary

| | |
|---|---|
| *RAD_CONST* | |
| *STEF_CONST* | |

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Attributes Documentation

**RAD_CONST = 8640000000.0**

**STEF_CONST = 0.0007125593324143198**

#### Methods Documentation

**process** (***kwargs*)
Process module.

### Photosphere

**class** mosfit.modules.**Photosphere** (*name*, *model*, ***kwargs*)
Bases: *mosfit.modules.Module*

Template class for photosphere Modules.

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Methods Documentation

**process** (***kwargs*)
Process module.

### DenseCore

**class** mosfit.modules.**DenseCore** (*name*, *model*, ***kwargs*)
Bases: *mosfit.modules.Photosphere*

Photosphere with a dense core and a low-mass envelope.

Expanding/receding photosphere with a dense core + low-mass power-law envelope.

#### Attributes Summary

| |
|---|
| *PL_ENV* |
| *STEF_CONST* |

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Attributes Documentation

**PL_ENV = 10.0**

**STEF_CONST = 0.0007125593324143198**

#### Methods Documentation

**process**(**kwargs)
Process module.

### Blackbody

**class** mosfit.modules.**Blackbody**(**kwargs)
Bases: *mosfit.modules.SED*

Blackbody spectral energy dist. for given temperature and radius.

#### Attributes Summary

| |
|---|
| *C_CONST* |
| *FLUX_CONST* |
| *STEF_CONST* |
| *X_CONST* |

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Attributes Documentation

**C_CONST = 29979245800.0**

**FLUX_CONST = 4.702049106843986e-12**

**STEF_CONST = 0.0007125593324143198**

**X_CONST = 1.4387773538277204**

#### Methods Documentation

**process**(**kwargs)
Process module.

### Line

**class** mosfit.modules.**Line**(**kwargs)
Bases: *mosfit.modules.SED*

Line spectral energy distribution, modifies existing SED.

#### Attributes Summary

| |
|---|
| *C_CONST* |

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Attributes Documentation

**C_CONST = 29979245800.0**

#### Methods Documentation

**process** (***kwargs*)
Process module.

### MultiBlackbody

**class** mosfit.modules.**MultiBlackbody** (***kwargs*)
Bases: *mosfit.modules.SED*

Generalized multiple blackbody spectral energy distribution.

#### Attributes Summary

| |
|---|
| *FLUX_CONST* |
| *STEF_CONST* |
| *X_CONST* |

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Attributes Documentation

**FLUX_CONST = 5.821090466093255e-46**

**STEF_CONST = 0.0007125593324143198**

**X_CONST = 4.79924466221135e-11**

#### Methods Documentation

**process** (***kwargs*)
Process module.

### LOSExtinction

**class** mosfit.modules.**LOSExtinction**(***kwargs*)

Bases: *mosfit.modules.SED*

Adds extinction to SED from both host galaxy and MW.

#### Attributes Summary

| |
|---|
| *ANG_CGS* |
| *C_CGS* |
| *H_CGS* |
| *H_C_CGS* |
| *KEV_CGS* |
| *LYMAN* |
| *MW_RV* |

#### Methods Summary

| | |
|---|---|
| *mm83*(nh, waves) | X-ray extinction in the ISM from Morisson & McCammon 1983. |
| *preprocess*(**kwargs) | Preprocess module. |
| *process*(**kwargs) | Process module. |

#### Attributes Documentation

**ANG_CGS = 1.0000000000000002e-08**

**C_CGS = 29979245800.0**

**H_CGS = 6.62607004e-27**

**H_C_CGS = 1.9864458241717581e-16**

**KEV_CGS = 1.6021766208000003e-09**

**LYMAN = 912.0**

**MW_RV = 3.1**

#### Methods Documentation

**mm83**(*nh*, *waves*)

X-ray extinction in the ISM from Morisson & McCammon 1983.

**preprocess**(***kwargs*)

Preprocess module.

**process**(***kwargs*)

Process module.

### SED

**class** mosfit.modules.**SED**(***kwargs*)
Bases: *mosfit.modules.Module*

Template class for SED Modules.

Modules that inherit from the SED class should produce a *seds* key, which contains a spectral energy distribution for each time. The units of the SED should be in ergs/steradian/cm^2/Hz/Angstrom.

#### Attributes Summary

| | |
|---|---|
| *C_OVER_ANG* | |

#### Methods Summary

| | |
|---|---|
| *add_to_existing_seds*(new_seds, **kwargs) | Add SED from module to existing `seds` key. |
| *receive_requests*(**requests) | Receive requests from other `Module` objects. |
| *send_request*(request) | Send a request. |
| *set_data*(band_sampling_points) | Set SED data. |

#### Attributes Documentation

**C_OVER_ANG = 2.9979245799999995e+18**

#### Methods Documentation

**add_to_existing_seds**(*new_seds*, ***kwargs*)
Add SED from module to existing `seds` key.

**Parameters** **new_seds** : array

The new SEDs to add to the existing SEDs.

**Returns** **new_seds** : array

The result of summing the new and existing SEDs.

**receive_requests**(***requests*)
Receive requests from other `Module` objects.

**send_request**(*request*)
Send a request.

**set_data**(*band_sampling_points*)
Set SED data.

### Synchrotron

**class** mosfit.modules.**Synchrotron**(***kwargs*)
Bases: *mosfit.modules.SED*

Synchrotron spectral energy distribution.

#### Attributes Summary

| |
|---|
| *ANG_CGS* |
| *C_CONST* |
| *FLUX_CONST* |
| *STEF_CONST* |
| *X_CONST* |

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Attributes Documentation

**ANG_CGS = 1.0000000000000002e-08**

**C_CONST = 29979245800.0**

**FLUX_CONST = 5.821090466093255e-46**

**STEF_CONST = 0.0007125593324143198**

**X_CONST = 4.79924466221135e-11**

#### Methods Documentation

**process**(**kwargs)
Process module.

### BlackbodyCutoff

**class** mosfit.modules.**BlackbodyCutoff**(**kwargs)
Bases: *mosfit.modules.SED*

Blackbody SED with cutoff.

Blackbody spectral energy dist. for given temperature and radius, with a linear absorption function bluewards of a cutoff wavelength.

#### Attributes Summary

| |
|---|
| *C_CONST* |
| *FLUX_CONST* |
| *F_TERMS* |
| *STEF_CONST* |
| *X_CONST* |

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Attributes Documentation

**C_CONST = 29979245800.0**

**FLUX_CONST = 4.702049106843986e-12**

**F_TERMS = 10**

**STEF_CONST = 0.0007125593324143198**

**X_CONST = 1.4387773538277204**

#### Methods Documentation

**process** (***kwargs*)
Process module.

### Diffusion

**class** mosfit.modules.**Diffusion** (***kwargs*)
Bases: *mosfit.modules.Transform*

Photon diffusion transform.

#### Attributes Summary

| |
|---|
| *DIFF_CONST* |
| *MIN_LOG_SPACING* |
| *N_INT_TIMES* |
| *TRAP_CONST* |

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Attributes Documentation

**DIFF_CONST = 9.682978142204928e+16**

**MIN_LOG_SPACING = -3**

**N_INT_TIMES = 100**

**TRAP_CONST = 4.747135373516629e+22**

#### Methods Documentation

**process**(***kwargs*)
Process module.

### DiffusionCSM

**class** mosfit.modules.**DiffusionCSM**(***kwargs*)
Bases: *mosfit.modules.Transform*

Photon diffusion transform for CSM model.

#### Attributes Summary

| |
|---|
| *MIN_EXP_ARG* |
| *N_INT_TIMES* |

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Attributes Documentation

**MIN_EXP_ARG = 50.0**

**N_INT_TIMES = 1000**

#### Methods Documentation

**process**(***kwargs*)
Process module.

### Viscous

**class** mosfit.modules.**Viscous**(***kwargs*)
Bases: *mosfit.modules.Transform*

Viscous delay transform.

#### Attributes Summary

| |
|---|
| *MIN_LOG_SPACING* |
| *N_INT_TIMES* |

#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Process module. |

#### Attributes Documentation

**MIN_LOG_SPACING = -3**

**N_INT_TIMES = 1000**

#### Methods Documentation

**process**(**kwargs)
Process module.

### Transform

**class** mosfit.modules.**Transform**(**kwargs)
Bases: *mosfit.modules.Module*

Parent class for transforms.

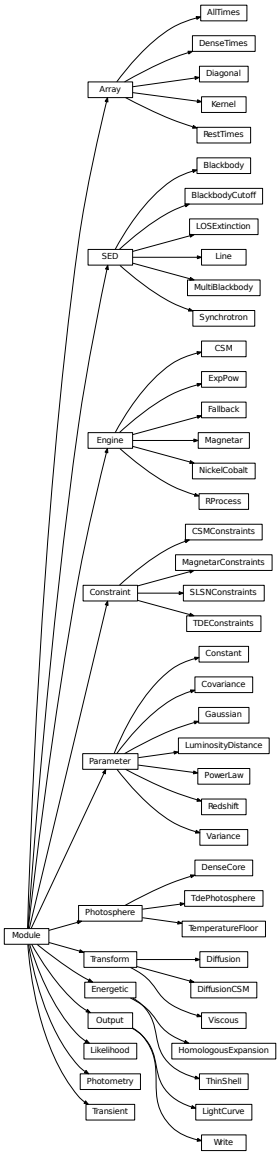
#### Methods Summary

| | |
|---|---|
| *process*(**kwargs) | Set *dense_** and **_since_exp* times/luminosities keys. |

#### Methods Documentation

**process**(**kwargs)
Set *dense_** and **_since_exp* times/luminosities keys.

## Class Inheritance Diagram



CHAPTER 2

# Indices and tables

- genindex
- modindex
- search

CHAPTER 3

# License & Attribution

# Bibliography

[WAT2010] Watanabe et al. 2010

# Python Module Index

## m

# Index

## Symbols

## A



## B



## C



## D



## E



## F

## N



## O



## P

## R



## S



## T



## V



## W



## X