BOXFIT v1.0
User Guide

Hendrik van Eerten

March 15, 2016

Contents

Chapter 1

About boxfit

The BOXFIT gamma-ray burst afterglow fit and light curve generator code is a numerical implementation of the work described in Van Eerten et al. (2012). The code is capable of calculating light curves and spectra for arbitrary observer times and frequencies and of performing (broadband) afterglow model fits using the downhill simplex method combined with simulated annealing. The flux value for a given observer time and frequency is a function of various variables that set the explosion physics (energy of the explosion, circumburst number density and jet collimation angle), the radiative process (magnetic field generation efficiency, electron shock-acceleration efficiency and synchrotron power slope for the electron energy distribution) and observer position (distance, redshift and angle).

The dynamics of the afterglow blast wave have been calculated in a series of 19 high-resolution two-dimensional jet simulations performed with the RAM adaptive-mesh refinement relativistic hydrodynamics (RHD) code (Zhang & MacFadyen, 2006). The results of these calculations have been compressed and stored in a series of 'box' data files and BOXFIT calculates the fluid state for arbitrary fluid variables using interpolations between the data files and analytical scaling relations. End-users of BOXFIT do not need to perform RHD simulations themselves.

The code can be run both in parallel and on a single core. Because a model fit takes many iterations, this is best done in parallel. Single light curves and spectra can readily be done on a single core. Use of the code is completely free, but we request that Van Eerten et al. (2012) be cited in scientific publications using the fitting algorithms from BOXFIT. This also applies to derived code, plus possible additional citations depending on the modifications (see chapter 5).

For suggestions, questions about material not covered in the guide or comments, please contact the author at `hveerten@mpe.mpg.de` / `hveerten@gmail.com`. Notifications of bugs in the code or mistakes (spelling or other) in the manual are especially appreciated.

Chapter 2

Getting started

2.1 Downloading and compiling

BOXFIT is written for linux systems, and the instructions below should apply to the large majority of such systems. In principle the code should be portable to windows and mac systems but no support will be provided. There are four components to BOXFIT. These are the main executable, which has to be compiled locally from the `c++` source code, a simple text file containing all user settings, a set of 19 data files that contain the compressed results of the jet dynamics simulations and, in the case of model fitting, a text file with observational data. All these files can be downloaded from `http://cosmo.nyu.edu/afterglowlibrary`, including an example data set. The jet dynamics data files are in the HDF5 data format[1]. The whole set of dynamics data is approximately 1GB in size, so might take some time to download depending on the speed of the internet connection.

The source code is distributed as a tarfile. Upon extraction the directory `boxfit` will be created, along with the subdirectories `bin`, `data`, `settings` and `src`. The directory `src` will contain the complete set of source files (these are described in chapter 5, but knowledge of the content of these files is not required in order to succesfully run BOXFIT). A Makefile is also included, so after checking whether the paths and dependencies in the file `makefile` are correct, the executable BOXFIT can be created by simply issuing

```
make clean boxfit
```

In the makefile itself, the relevant environment variables are `CXX`, which sets the compiler that will be used. Common options are `g++` or `icpc` for the free GNU `c++` compiler and the proprietory Intel `c++` compiler, respectively. For use on a parallel computer, use `mpicxx`.

The environment variable `LDFLAGS` specifies which libraries the code depends on. We have kept these dependencies to a minimum for maximum compatibility, with the only nonstandard library being the HDF5 library which is sufficiently common that it is likely installed on most computer networks in science institutes. It is also available from the standard repositories in commonly used linux distro's such as Fedora and Ubuntu. `-lm` and `-lhdf5` refer to the math library and HDF5 library, respectively.

[1] See `http://www.hdfgroup.org/HDF5/` for more information on HDF5.

The directory following -L should include the HDF5 library files, in case these are not in a standard place such as /usr/lib.

The environment variable CPPFLAGS sets additional flags for compilation, such as code optimization level and compiler verbosity. In principle, these can be omitted but code optimization might improve performance.

After compilation the directory bin will contain the executable boxfit. Either copy this file to the directory where you wish to run boxfit, or set your PATH to include this directory.

The directory settings contains the template parameter file. By default this file is called boxfitsettings.txt and if BOXFIT is run without additional arguments it will assume the existence of a text file with this name in its current directory (so running BOXFIT from within the bin directory without arguments will result in an error as it cannot find the parameter file in its current dir). It is recommended that your problem-specific parameter file is copied to the directory where you wish to execute the program.

The directory data will only contain the template observational data file at first, but is suggested to contain the jet dynamics data files ('BOX' files). Because of their size, these are not part of the source code tarfile and need to be downloaded separately. Their filenames are box00.h5, box01.h5 etc. The path to the data files needs to be specified in the parameter file.

2.2 First run (calculating a light curve)

Assuming a local desktop computer with linux installed, a first run can proceed along the following steps:

- Download the tarfile boxfit.tar from http://cosmo.nyu.edu/afterglowlibrary and save into a local directory. Go to that directory in a terminal and extract the files using

  ```
  tar -zxvf boxfit.tar
  ```

- Download the files boxfit00.h5, ..., boxfit18.h5 and store these in the data subdirectory.

- In the source code directory, open the file environment.h in an editor and make sure the variable determining whether to compile the parallel or serial code is set to serial:

  ```
  #define PARALLEL_            DISABLED_
  ```

- Check the makefile. Set the compiler to g++ and make sure that the HDF5 directory is set correctly.

- In the source directory, compile the code using

  ```
  make clean boxfit
  ```

- create a directory for output, say boxfitoutput, on your system. Copy the executable and the parameter file into this directory.

- Open the parameter file, `boxfitsettings.txt`, in an editor and make sure that the path to the data files is listed correctly, for example to:

  ```
  box_path = "/home/username/boxfit/data/"
  ```

- In your output directory, run BOXFIT:

  ```
  ./boxfit boxfitsettings.txt | tee boxoutput.log
  ```

  The use of the argument `boxfitsettings.txt` is actually redundant, since this is the standard filename BOXFIT will look for. It is included here to illustrate how to use arbitrary parameter file names. The extra `| tee boxoutput.log` has the effect that the output will be written to a text file as well as to the screen.

- In the same parameter file, make sure that `eds_phi_res` is set to 1. We will first calculate the light curve for an on-axis observation, so the radiation intensity is circularly symmetric (the corresponding image on the sky would have been circular).

- Upon download, the standard settings in the template file are set to creating a light curve. After running BOXFIT a file `lightcurve.txt` should have been created. Open this file and check whether the output makes sense. It should contain the data for an optical afterglow light curve with standard explosion, radiation and observer parameters. The text file can easily plotted with a standard plotting tool like GNUPLOT, available in the repositories of most linux distro's:

  ```
  gnuplot
  set log
  plot "lightcurve.txt" u 2:4 with lines
  ```

  The resulting plot is shown in Fig. 2.1. The change in slope around 0.3 days is the jet break. The late time bump is the counterjet. The steep drop indicates the point where observer times are no longer fully covered by the simulation: at these times the flux is automatically set to zero. When a spectrum is calculated instead of a lightcurve, a file `spectrum.txt` is generated instead of `lightcurve.txt`

This should also offer a good starting point to experiment with settings that dictate the resolution and emission time coverage of the light curve calculation. These are `no_points`, `BM_start`, `eds_r_res`, `eds_phi_res`, `BM_res` and `box_res`. These settings and the other settings in the parameter file are all briefly explained in the following section.

2.3 The parameter file

All user settings and parameters for BOXFIT are set in a text file, `boxfitsettings.txt`, by default. Exclamation marks are used to separate comments from settings and everything following an exclamation mark will be ignored. There are no rules for indentation, but the variable names are case sensitive. Here we briefly describe each setting. Not all settings are needed for all BOXFIT functions (e.g. simplex settings are irrelevant when a single light curve is generated).

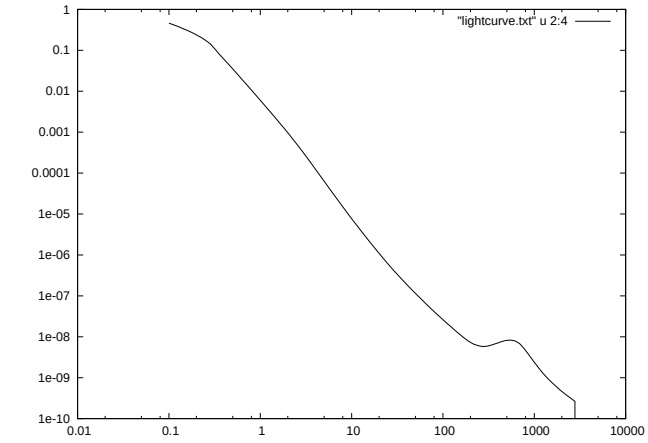

Figure 2.1: `gnuplot` output for a light curve calculated at 10^{14} Hz according to standard settings. Note the drop to zero flux at late times.

what_to_do

An integer number sets what BOXFIT should do when executed. The following options are possible:

-1. Perform some basic self tests of the code and exit the program. This should serve as a tool for debugging the code.

0. Calculate χ^2 once, for the parameters `theta_0`, ... , `ksi_N` and the datafile indicated by `data_filename`, then exit the program.

1. Calculate a single light curve from parameters `theta_0` etc. Observer timespan is set by `t_0` and `t_1`, observer frequency by `nu_0`.

2. Calculate a single spectrum from parameters `theta_0` etc. Observer frequency span is set by `nu_0` and `nu_1`.

3. Perform an iterative fit on the dataset, starting from a simplex with initial lower values determined by `theta_0` etc. and upper values by `theta_0_simplex_max` etc. The fit parameters are forced to remain within the boundaries set by `theta_0_min` and `theta_0_max` etc. Additional settings are discussed below.

4. Determine the errors on the fit parameters using a Monte Carlo approach, assuming that `theta_0` etc. represent the best fit values. In this settings the observations are fully recalculated for each Monte Carlo iteration, and this option is included for completeness but not supported.

5. Determine the partial derivatives and store them to disc.

6. Determine the errors on fit parameters via Monte Carlo procedure, assuming that `theta_0` etc. represent the best fit values. Use partial derivatives method with best fit light curve as starting point. The derivatives are loaded from disc and have to be calculated using `what_to_do = 5` previously.

box_path

Sets the path to the box files `box00.h5` etc. Double quotes need to be included. To set the path to the current directory, for example, use `"./"`.

no_boxes

The number of box files. In principle, this should be equal to 19.

data_filename

The filename of the dataset. This text file should have the following standard structure. The first line should list the total number of datapoints in the file as follows (for 123 datapoints, for example): `# datapoints = 123`. The following lines should contain four entries each, separated by comma's. For example: `2.08783`, `2.42E18`, `2.24163E-5`, `6.01974E-6`, for a data point at little over 2 days in X-rays (10^{18} Hz) with flux and 1 σ error expressed in mJy.

save_intermediate

A binary setting. If 1, then whenever the temperature is lowered during the simulated annealing fit procedure, a text file `intermediate0000.txt` is saved with `0000` incremented by 1 for each new intermediate file. The file contains the current best fit result in addition to the original dataset. If set to 0, no intermediate fit result files are saved.

start_from_simplex and simplex_filename

A binary setting. If 0, the fit procedure starts by constructing a simplex with initial lower values determined by `theta_0` etc. and upper values by `theta_0_simplex_max` etc. If 1, the complete specification of the simplex is written from disc. See 3.3 for further details. The filename of the simplex file should be specified in `simplex_filename`.

nu_0, nu_1, t_0, t_1

Float values that set the frequency and time for light curves and spectra when calculated with options 1 and 2 for `what_to_do`. Frequency should be in Hertz, time in days.

no_points

An integer value that sets the number of points that are calculated for the light curves and spectra of options 1 and 2 for `what_to_do`. The points are logarithmically spaced either in time or frequency.

d_L, z

Two floats setting the distance to the object. The redshift is set by `z`, the luminosity distance (in cm) by `d_L`. It is up to the user to ensure both values are consistent with a given cosmology.

self_absorption, electron_cooling

Two binary settings that determine the physics to include in the radiative transfer calculations. If `electron_cooling` is set to 1, electron cooling is included. Electron cooling is calculated using a global cooling time approximation following Sari et al. (1998) and affects high-frequency (i.e. X-ray) observations. If `self_absorption` is set to 1, synchrotron self-absorption is included, which mostly affects radio observations.

include_box, include_BM

Binary settings that determine the source of the fluid states. The program solves the radiative transfer equations for a number of rays through an evolving fluid. The fluid state can be generated either by the analytic Blandford-McKee self-similar solution of purely radial outflow in a conic jet (Blandford & McKee, 1976), or by the stored jet simulation data in the data files `boxfit00.h5` etc. The common setting should be both enabled, with `include_BM` setting the early-time Blandford-McKee coverage and `include_box` setting the simulation data coverage.

BM_start, BM_stop

Two float values setting the starting and stopping fluid Lorentz factor γ for the Blandford-McKee (BM) part of the jet evolution. In this solution, purely radial outflow is assumed with the initial opening angle set by the opening angle fit parameter and the local fluid variables are determined according to the analytically known self-similar BM profile. This profile also provided the initial conditions for each simulation stored in the boxfit data files. The BM profile can therefore be combined with the simulation results since the two smoothly connect. This has the advantage that earlier emission times are covered. The signal at a given observed time is the combined result from emission at many different emission times, and because the jet initially nearly keeps up with its radiation, a long emission timespan is needed to cover a small observer timespan. The difference between the two timespans is a factor γ^2, the Lorent factor of the fluid directly behind the shock front. Recommended value for `BM_start` is around 200. If `BM_stop` is set to a negative number, the starting Lorentz factors for the simulation are used (25). For a run using only the BM solution, a lower limit of around 2 is possible. Beyond that the value the BM solution, which assumes ultra-relativistic flow, is no longer accurate. The timespan corresponding to the bounding Lorentz factors can be calculated according to Blandford & McKee (1976).

eds_r_res, eds_phi_res

Integers setting the numerical resolution (i.e. number of separate rays) of the radiative transfer calculations. The rays are logarithmically spaced in the direction perpendicular to the observer. The total number of rays in this radial direction (radial in the plane with normal pointing to the observer) is set by `eds_r_res`. The total number of rays in the ϕ direction (the angle on the plane with the normal pointing to the observer) is set by `eds_phi_res`. For an on-axis observation, this value can be set equal to 1 due to symmetry (i.e. the image on the sky is circular). The observed flux is the area integral over the region covered by a total of `eds_r_res` $\times$ `eds_phi_res` rays. The larger the observer angle, the more important this resolution `eds_phi_res` becomes.

BM_res, box_res

Two integers setting the resolution of the radiative transfer line integrals. While the number of rays is set by `eds_r_res` and `eds_phi_res`, these values set the resolution along the rays. `BM_res` sets the step size between `BM_start` and `BM_stop`. `box_res` sets the value for the range covered by the simulations. Values of a few thousands will lead to a converged result. Much smaller values can be used for test runs.

temp

A float value that sets the starting temperature for the simulated annealing procedure. In a simulated annealing approach to the downhill simplex method the simplex is a set of points in fit parameter space that should move closer to the best fit value over the course of many fit iterations. When during an iteration an attempt is made to move the simplex point with the highest χ^2 closer to the best fit value, a random value is added to the χ^2 values of each point during the selection of the worst fit. The scale of this random component is set by the fit temperature, so the 'hotter' the simplex, the more often an attempt is made to improve a point that is not the worst of the set. As a result the algorithm is able to escape local minima in fit parameter space. The initial annealing temperature set by `temp` should be at least equal to the initial χ^2 values of the points in the simplex, something that can be checked directly after the code starts to run as these are part of the output of the code. If the initial simplex values are not chosen to resemble the data set under consideration, as a rule-of-thumb the initial temperature should exceed the χ^2 for the dataset with all fit flux values set to zero, i.e. as the sum of the squared flux values divided by their squared errors. This is strongly dataset-dependent, but initial temperatures of 10^5 are no exception. Further information about simulated annealing and downhill simplex can be found in Press et al. (1986); Nelder & Mead (1965); Kirkpatrick et al. (1983).

iter

An integer number setting the number of iterations the fit code should perform at a given temperature. At least a handful of iterations should be performed at fixed temperature before the temperature is lowered again.

temp_factor

A float value < 1, the fraction that the temperature should be lowered after `iter` fits have been performed at fixed temperature. This value should remain close to 1 (for example, 0.99), for if too low, the effect of high temperatures is lost and the fit algorithm might not be able to avoid local minima completely.

temp_lowest

A float value that sets the lower bound on the annealing temperature. Because the temperature is decreased according to a mulitplicative factor, it will approach zero asymptotically and a nonzero lower bound is required. After this temperature is reached, the code will perform a final set of iterations at zero temperature. The lower temperature

bound therefore has to be smaller than the best fit χ^2. Typical good fits will have reduced χ^2 on the order of a few, so the lower temperature bound should be significantly lower than the total number of datapoints.

MC_runs

An integer value that sets how many Monte Carlo runs should be calculated when the error on the best fit values is determined. This should be a very large number in order for the Monte Carlo procedure to converge, for example 10000, or 1000 at least. During the Monte Carlo analysis of the fit parameter errors the flux values of the dataset are perturbed `MC_runs` times with random shifts that have an amplitude set by the size of the error bar. After each perturbation, the best fit value is recalculated.

Fit parameter values theta_0, ...

The fit variables are a floating point numbers that set explosion physics, radiation physics and observer angle. `theta_0` is the jet half opening angle in radians (typically 0.1 for a long GRB), `E` is the explosion energy in erg (10^{52}), `n` is the circumburst number density in cm^{-3} (1), `theta_obs` the observer angle in radians, `p` the synchrotron slope (2.5), `epsilon_B` the fraction of downstream internal energy in the shock-generated magnetic field (0.01), `epsilon_E` the fraction of downstream internal energy in the shock-accelerated electrons (0.1), `ksi_N` the fraction of electrons being accelerated (1.0). In case a single spectrum or light curve is calculated, these parameters set the values for that calculation. In case of a data fit, these provide the lower bounding values for the initial simplex (unless a simplex from disc is used and `start_from_simplex` is 1). In case of a Monte Carlo data fit error estimation, these provide the best fit results (so make sure that these are updated correctly when calculating the error on the fit parameters after finding a best fit).

Frozen parameters theta_0_frozen, ...

A set of binary parameters, `theta_0_frozen`, `E_frozen`, `n_frozen`, `theta_obs_frozen`, `p_frozen`, `epsilon_B_frozen`, `epsilon_E_frozen`, `ksi_N_frozen`, that determine which values are included when fitting. If a parameter (e.g. `theta_0_frozen` is set to 1 the corresponding fit parameter is 'frozen' and is not included in the fit. All fits will then use the fit parameter value (e.g. `theta_0`) from the settings file and the variable with be included when determining the reduced χ^2.

Fit parameter ranges theta_0_min, theta_0_max, ...

A set of float values that set the lower and upper bound of parameter space. The lower bounds are set by `theta_0_min`, `E_min`, `n_min`, `theta_obs_min`, `p_min`, `epsilon_B_min`, `epsilon_E_min`, `ksi_N_min`. The upper bounds are set by `theta_0_max`, `E_max`, `n_max`, `theta_obs_max`, `p_max`, `epsilon_B_max`, `epsilon_E_max`, `ksi_N_max`. Important lower boundaries are 0.045 for `theta_0_min` (this is the smallest opening angle covered by the simulations) and 2.0 for `p_min` (for internal consistency of the physical model, if it is lower the integrated energy of the accelerated electrons diverges). Important upper boundaries are 0.5 for `theta_0_max` (the largest angle covered by simulations) and 1.0 for `espilon_B_max`, `epsilon_E_max` and `ksi_N_max` (since these are fractions).

Initial simplex range theta_0_simplex_max, ...

A set of float values that determine the range of the initial simplex, `theta_0_simplex_max`, `E_simplex_max`, `n_simplex_max`, `theta_obs_simplex_max`, `p_simplex_max`, `epsilon_B_simplex_max`, `epsilon_E_simplex_max`, `ksi_N_simplex_max`. When the initial simplex is created using these values as upper limits and the fit parameters `theta_0`, ... as lower limits, a set of $N+1$ simplex points is generated, where N the number of parameters included in the fit. The first simplex point is given by the fit parameters, and for each subsequent point one of the original fit parameters is replaced by its upper limit. In the case of `what_to_do = 5`, when partial derivatives with respect to the best fit values are calculated, the differences between the variables `theta_0_simplex_max` etc. and `theta_0` etc. determine the stepsize used in the derivative.

Chapter 3

Model fitting

Creating single light curves and spectra is very straightforward and basically covered in chapter 2. Here we provide additional instructions on how to apply BOXFIT to fitting datasets.

3.1 Running on a cluster

Fitting a model to a dataset in general requires the use of a computer cluster. The parallelization in BOXFIT works as follows: the data points in the dataset are divided over the available cores and for each fit iteration, each core performs the radiative transfer calculation for the observer time and frequency corresponding to its data point. Once the calculation is finished, a core will request a new data point. This approach has a direct implication for the ideal number of cores that is used when fitting a dataset: equal to the number of data points plus one (a 'master' core that handles the bookkeeping and I/O of during the run). Any additionally added cores will remain idle throughout the run, since they will never receive a data point to work on. Please check with your local computer cluster administrator for details regarding how to submit parallel computation jobs. These is typically down via a *script* file that is likely to resemble the following:

```
# script
#PBS -l nodes=11:ppn=8
#PBS -N boxfit

cd $PBS_O_WORKDIR

# start your program
mpiexec ./boxfit > out
```

In this particular example 11 nodes are requested and 8 processors per node, making for a total of 88 processors or cores. In a dataset of, for example, 82 datapoints, 5 cores will remain idle throughout the calculation. The job name is set to 'boxfit' and when the script is executed it changes directory to the current directory and executes a local copy of BOXFIT, while the output is piped to a text file `out`.

3.2 Output files from a model fit

When the parameter `what_to_do` is set to 3 and all settings are prepared for a model fit, the code will generate a number of text files during execution. Each fit iteration result is appended to a file `fit.txt`. Progress of the model fit is quickly checked by looking at this file. It also shows the χ^2 results for the initial simplex, and these should be checked against the initial annealing temperature. After `iter` fits have been performed at a given temperature, the temperature is decreased and two files are generated before fitting continues at lower temperature. The file `currentsimplex.txt` contains the complete simplex at that stage. This is useful in case the fit procedure is interrupted and fitting has to be restarted from an intermediate stage. In addition a file `intermediate0000.txt` is written to disc, with '`0000`' increased by 1 after each decrease in temperature. This file contains the complete data set plus the fit results and can be plotted quickly (for example in GNUPLOT) to compare the data and model light curves by eye. If in the parameter file the setting `save_intermediate` is set to zero, no intermediate files are stored. The total number of intermediate files depends on the starting temperature, the stopping temperature and the temperature decrease factor and might become very large (> 1000).

Once the fit is completed the results are stored in a filename `bestfit.txt`. The content of this file is similar in structure to that of the intermediate fit result files. Note that the best file values are not written to the file but are part of the screen output of BOXFIT.

During the fit routine there is output is also written to screen (`stdout`), such as user settings and intermediate fit results after each temperature decrease. It is therefore strongly recommended that the output be piped to a text file as well (using e.g. `> out` as in the script file of the preceding section or `| tee boxoutput.log` as in the first run section).

3.3 Starting from a simplex file

The initial simplex can be defined in two ways. Either by specifying its range through `theta_0`, ... and `theta_0_simplex_max`, ... and setting `start_from_simplex = 0`, or by loading a simplex from disc. This is done by specifying a simplex filename in `simplex_filename` and setting `start_from_simplex = 1`. An example simplex file might look as follows:

```
# current annealing temperature = 8.007314e+02
# number of fit parameters = 2
2.827129e-01, 5.228039e+01, 8.075746e+02
4.594679e-01, 5.201461e+01, 1.082095e+03
2.807668e-01, 5.215024e+01, 9.812374e+02
4.516597e-01, 5.217091e+01, 1.186542e+03
4.079544e-01, 5.202525e+01, 1.201930e+03
3.706876e-01, 5.233105e+01, 1.282199e+03
3.068600e-01, 5.238077e+01, 7.280832e+02
3.968382e-01, 5.202685e+01, 1.199974e+03
```

A user-specified simplex file should follow the exact same pattern, including the two header lines. The actual number used for the current annealing temperature will have no effect on the actual starting temperature of the fit, since this is decided by

`temp` in the parameter file. Also, the χ^2 results for the simplex points (the last table in the simplex file) will not be read from disc but recalculated. Both starting temperature and χ^2 values are included in the file layout in order to have an identical layout to the `currentsimplex.txt` file that is updated after each temperature change (where their inclusion is informative). This makes it simpler to continue a fit from a `currentsimplex.txt` file after a fit is interrupted. It is recommended just to set start temperature and χ^2 results equal to -1.0 for clarity, a value that would not occur otherwise. It is also very important to ensure that the thawed parameters match the simplex fit parameters, since this information is not included in the simplex file. Finally, most fit parameters are actually first translated to their base 10 logarithms before fitting. This should also be done when manually specifying the simplex (e.g., the second column in the example above is the ^{10}log of the energy). The following variables are in log form: `E`, `n`, `epsilon_B`, `epsilon_E` and `ksi_N`, but not when `simplex_max` in the parameter file is used.

Note that the initial simplex values for a given parameter also determine the initial step size along the dimension of parameter space for that parameter. It is therefore not possible to set the simplex difference between `simplex_max` and the initial value to zero for a thawed parameter, since this would effectively exclude that parameter from the fit algorithm. In this case, BOXFIT will issue an error message and quit.

3.4 A note on ξ_N

When fitting model parameters it is important to keep in mind that there is a degeneracy between ξ_N and the other parameters. This was first pointed out by Eichler & Waxman (2005). This means that a shift in ξ_N can be compensated for by a combined specific shift in E, n, ϵ_E and ϵ_B. In practice, it follows that it is pointless to keep both the latter parameters and ξ_N thawed during a fit and BOXFIT will issue a warning when this is done.

3.5 Determining the error bars on the fit parameters

In practice, determining the error bars using `what_to_do = 4` (full radiative transfer for each new light curve) is not feasible, since a full Monte Carlo procedure requires at least thousands of best fit calculations. The option `what_to_do = 4` is included for completeness, but not supported.

Alternatively the error bars on the best fit parameters can be calculated using the best fit results as starting point. As explained in Van Eerten et al. (2012), we determine the errors on the best fit parameters from the errors on the datapoints by perturbing the dataset using the flux errors and then finding a new best fit. This procedure is then repeated a large number of times. We then take the lowest 68.3% of the resulting χ^2 values and the extremes of the fit parameters within this subset determine the 1σ uncertainties of the fit parameters[1].

[1] The reason that we have chosen to perturb the dataset rather than using an arguably more elegant *bootstrap* method where the dataset is replaced by a randomly drawn subset of the original dataset before each Monte Carlo iteration, is that in a broadband dataset the number of datapoints in different bands can vary wildly and a sparsely populated band dropping out affects the fit constraints in a fundementally different way than shifting its values will, even if the error bars are large.

3.5.1 Setting the partial derivatives

The first step in determining the error bars is to calculate for each datapoint the partial derivatives around the best fit result with respect to the fit parameters. It is then possible to quickly calculate light curves in terms of deviations from the best fit light curve. Calculating the derivatives has to be done separately, using `what_to_do = 5`. The derivative for a given datapoint with best fit flux F_i with respect to a fit parameter x, so $\partial F_i/\partial x$, is calculated with a stepsize determined by the difference between the maximum simplex values and the best fit values. In the case of jet angle we get $\Delta x \equiv$ `theta_0_simplex_max - theta_0`, etc. The partial derivative is calculated from steps with size Δx, $\Delta x/2$, $-\Delta x/2$ and $-\Delta x$. It is up to the user to set the appropiate size and check the resulting derivatives. The derivatives are saved in a file name `partialdevs.txt`. Keeping in mind that many fit parameters are first transferred to logarithms, the following stepsizes are recommended:

- $\mathrm{d}\theta_0 \sim 0.05$
- $\mathrm{d}\log E_{iso} \sim 0.1-1.0$ (i.e. up to one order of magnitude)
- $\mathrm{d}\log n_0 \sim 0.1-1.0$ (i.e. up to one order of magnitude)
- $\mathrm{d}\theta_{obs} \sim 0.05$
- $\mathrm{d}p \sim 0.1$
- $\mathrm{d}\log \epsilon_B \sim 0.1$
- $\mathrm{d}\log \epsilon_E \sim 0.1$
- $\mathrm{d}\log \xi_N \sim 0.1$

Note that `E_simplex_max`, `E`, etc. are *not* represented as logarithms in the settings file, so the suggestions above should first be appriopately translated to the scale of the best fit results.

3.5.2 Monte Carlo procedure

Once the partial derivatives are known, the actual Monte Carlo procedure can be started using `what_to_do = 6`. The existence of a file `partialdevs.txt` in the current working directory is assumed.

Chapter 4

Things that can go wrong

In this chapter we list common things that can go wrong when using BOXFIT. There are various ways in which the code can produce incorrect results, some of them subtle. It is therefore worthwhile to check this list even when the output at first glance appears to be correct.

1. **Unphysical explosion, radiation or observer parameters.** The calculation might involve unphysical values for the fit parameters. Energies and densities should be positive. Note that the energy is not in terms of 10^{52} erg but in terms of erg. All fractions (`epsilon_B`, `epsilon_E`, `ksi_N` should lie between 0 (exclusive) and 1 (inclusive). The synchrotron power law slope `p` should be greater than 2.0. Angles should be positive and in radians. The possible jet angles are set by the simulations and should lie between 0.045 and 0.5 radians (inclusive).

2. **Wrong units in data file.** The data file, consisting of four columns separated by a comma, requires the following dimensions: observer time in days, observer frequency in Hertz, (monochromatic) flux in mJy and flux error also in mJy. The milliJansky unit is typically used in radio astronomy and is equal to 10^{-26} erg s^{-1} cm^{-2} Hz^{-1} in cgs units.

3. **Observer time not (fully) covered or covered twice.** An observer time can be covered twice if both analytical BM and simulation boxes are enabled and the lower BM Lorentz factor limit is set to a value smaller than the upper Lorentz factor where the simulations take over (25). Early time data points might not be fully covered, especially if the analytical BM solution is disabled. Note that the flux at a given observer time is determined by the combined emission from many different emission times. Very late observer times are also not covered by the simulation (or the BM solution, which is only semi-accurate down to Lorentz factors of ~ 2 under the assumption of purely radial flow). When an observer time is not fully covered, the code sets the flux to zero, rather than allowing the flux coverage to drop continuously. This should make it easier to check for late time coverage issues. This is not done for early times, because here relativistic beaming plays a strong role and coverage might not formally be complete but complete in practice (with an error that is effectively zero) because all observed emission comes from parts of the flow directed in the general direction of the observer.

4. **Incorrect path to jet dynamics data file or observer data file.** This will cause the code to crash, so should be easily recognizable.

5. **Not all data points are used.** The number of data points in a data file is set on the first line of the text file containing the data. This also provides an option to quickly select only a subset of the data: once the number of data points specified on the first line is included, further data points in the file are ignored. Make sure to restore this number to specify the complete dataset in case you wish to use the complete dataset again after working with a subset.

6. **Starting temperature for simulated annealing too low.** Make sure that the starting temperature for the simulated annealing is set to a numerical value higher than the initial unreduced χ^2 values of the starting simplex points. If there is a reason to believe that the starting simplex lies entirely within a local minimum, the starting temperature should be even higher (and / or the simplex points chosen differently).

7. **Cooling too fast.** As with the cooling of crystals, when it is done too quickly irregularities may remain and the end product might be stuck in a local rather than a global minimum. In order to prevent this from happening both `iter` and `temp_factor` should be sufficiently high (ideally comparable 100 and 0.99, respectively, but this might be numerically challenging in practice and can be lowered when fewer parameters are fitted).

8. **fit parameters not set to best fit values when calculating errors.** Once a fit has been performed and the best fit values are found, the errors on the fit parameters can be calculated using a Monte Carlo approach. To perform a Monte Carlo error bar determination, the fit parameters in the settings file should be replaced by the best fit values. If this is not done, the code will assume the wrong values for the best fit.

9. **Partial derivatives noisy.** When determining the partial derivatives of the model flux values with respect to the fit variables, proper care should be taken in determining the stepsize for the derivative. If a stepsize is chosen too small, the resulting partial derivative might end up reflecting numerical noise rather than parameter dependence. Please check the file `partialdevs.txt` and keep in mind that the flux and most fit parameters are represented on a logarithmic scale. As a rule, the radiation parameters are more robust against small stepsize than dynamical parameters.

10. **strange lower and upper limits to fit variable error bars.** This typically occurs when the number of Monte Carlo runs is set too small, or when the settings for the Monte Carlo runs don't converge well to a minimum. If only a very small number of Monte Carlo runs is done, the error bar for a given fit parameter might not even include the best fit value. This is fixed by increasing the number of Monte Carlo runs (ideally $\gg 1000$) and by checking whether the results have converged properly. Might also be related to noisy partial derivatives.

11. **fitting the wrong physics.** The BOXFIT code is inherently limited to the afterglow blast wave dynamics covered by the original two-dimensional jet simulations. These assume a decelerating blast wave in a homogeneous circumburst

environment. Specifically, the code can not be used to fit for flares or rebrightenings in the light curves or for a coasting phase (the expected physical origin of the plateau phase in early *Swift* data).

12. **Fit parameters degenerate.** If there are not enough data points or if the data points do not provide sufficient information (e.g. when they're all on a straight power law decline, nothing can be inferred about the jet or observer angle and a single band observation of a broken power law type light curve is not sufficient to break the degeneracy between explosion energy and circumburst density), the fit results can be misleading. This is a conceptual issue and cannot be solved by a fit algorithm: additional information in the form of observations at different wavelengths or additional constraints (e.g. assuming standard values for some of the fit variables and 'freezing' them) are required.

Chapter 5

The source code

The source code for BOXFIT consists of various files. In this chapter we briefly describe the different files. The main routine can be found in `boxfit.cpp`, that we describe first, followed by the other source files in alphabetical order. All code is written in `c++`. Combining synchrotron emission to simulation snapshots is first described in Van Eerten & Wijers (2009), the linear radiative transfer method is first described in Van Eerten et al. (2010b), the application to two-dimensional simulations in Van Eerten et al. (2011), the application to off-axis observers in Van Eerten et al. (2010a). The BOXFIT code is described in Van Eerten et al. (2012). When using the source code as basis for a separate project, the appropriate article(s) should be cited in resulting publications.

boxfit

The main routines are defined in `boxfit.cpp` and `boxfit.h`. When BOXFIT is run, parameters are read from file and memory is assigned. Control is handed over to a class `c_boxfit` that contains functions to perform the various tasks, like fitting a data set or creating a single light curve. The function to determine χ^2 is part of this class as well.

arraytools

The header file `arraytools.h` contains templates to define arrays up to three dimensions. The templates ensure that each array is allocated as a continuous block in memory, which is a requirement for quick file I/O with the HDF5 filestructure.

BM

The Blandford-McKee solution is encoded in `BM.cpp` and `BM.h`. The class `c_BM` includes routines to set the global variables of the BM solution and routines to obtain the local fluid state from the self-similar equations once the global state is set.

box

When the radiative transfer calculation needs to know the state of the fluid as it is determined by the compressed simulation data it will request the local fluid state from the `c_box` and `c_multibox` classes in `box.cpp` and `box.h`. These are derived

classed from the basic `fluid` class. All algorithms that interpolate within a given 'box' (i.e. a single compressed simulation with a given jet initial opening angle), including scaling of energies and densities can be found in `c_box`. The `c_multibox` class also includes interpolation between different jet collimation angles.

coords

This offers a simple struct `s_coordinates` that groups together a set of coordinates and provides some routines to translate between Cartesian and spherical coordinates.

eds_2D_regular

The bookkeeping of the radiative transfer calculation is done in the class `c_eds` in `eds_2D_regular.cpp` and `eds_2D_regular.h`. This class contains an array storing the regular grid in polar coordinates (logarithmically in the radial direction perpendicular to the observer direction) that contains the intermediate ray intensities for all rays, as well as emission and absorption coefficients during the ray update procedure. When an update is requested by the radiation calculation performed in `flux_from_box`, first all emission coefficients are set. This is done using by calling the radiation routine for synchrotron radiation contained in `radiation`, that in turn makes use of the generic fluid state interface in `fluid`. 'EDS' stands for 'equidistant surface' (Van Eerten et al., 2010b).

environment

In the header file `environment.h` a number of important pre-compiler switches are set. Most important of these is `PARALLEL_` which determines whether the code will be compiled for parallel execution on a cluster or for execution on a single core.

extramath

Some additional math routines and constants are found in `extramath.cpp` and `extramath.h`.

fluid

The files `fluid.cpp` and `fluid.h` contain the class `c_fluid` as well as labels referring to the different fluid variables. The fluid class does not contain functional fluid routines, but rather sets the framework for interaction with the analytical (BM) fluid states, the box-based fluid or the simulation grid fluid states (not included in the public release). Routines containing these all are derived classes based on the fluid class. This way a standard set of routines is defined that different parts of the code (such as the emission coefficient calculation) can call in order to probe a local fluid state.

fluid_special

The files `fluid_special.cpp` and `fluid_special.h` contain the class `c_fluid_special`, a derived class from `c_fluid` that uses the BM solution as implemented in `c_BM` to

return local fluid states according to the Blandford-McKee solution. It is the analytical analog of the c_box and c_multibox classes that deal with the compressed simulation results.

flux_from_box

The flux calculation for a datapoint is contained the class c_datapoint in files flux_from_box.cpp and flux_from_box.h. When a flux calculation is requested by main program, a loop is entered where the intensities are calculated simultaneously for a large number of rays. During this loop the local values of the emission and absorption coefficients are set using the bookkeeping code from class c_eds, following which the positions along the rays are increased by a small increment and the intensities are updated. The parallelization is done as follows: each core gets assigned a single datapoint and will separately calculate the radiative transfer for the observer time and frequency relevant to this datapoint. Once this calculation is finished the core will request a new datapoint from the master core, until all fluxes are calculated.

numerical

In numerical.cpp and numerical.h two classes are defined that deal with more extensive numerical routines not covered by extramath. These are c_random for random number generation using different underlying distribution functions and c_simplex, which implements the downhill simplex method with simulated annealing. The source code is based directly on the original scientific publications that describe them, but good descriptions of the algorithms can also be found in Press et al. (1986).

param_IO

All routines for reading the user parameter file boxfitsettings.txt are defined in param_IO.cpp and param_IO.h

physics

Various physics constants are defined in physics.h

radiation

The calculation of the emission and absorption coefficient is done within the class c_emab, defined in radiation.cpp and radiation.h. This class gets access to a fluid class (in the form of one of the two possible derived classes, either the analytical solution or the box data) via a pointer. A set of coordinates is provided using the s_coordinates struct. The c_emab routine is repeatedly called by c_eds when the emission and absorption coefficients need to be updated for the set of rays. A number of subroutines defining the synchrotron radiation spectrum are also part of c_emab, and c_emab is the starting point for implementing changes in the radiative process.

Bibliography

Blandford, R. D., & McKee, C. F. 1976, Physics of Fluids, 19, 1130

Eichler, D., & Waxman, E. 2005, ApJ, 627, 861

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. 1983, Science, 220, 671

Nelder, J. A., & Mead, R. 1965, Computer Journal, 7, 308

Press, W. H., Flannery, B. P., & Teukolsky, S. A. 1986, Numerical recipes. The art of scientific computing, ed. Press, W. H., Flannery, B. P., & Teukolsky, S. A.

Sari, R., Piran, T., & Narayan, R. 1998, ApJ, 497, L17+

Van Eerten, H., Zhang, W., & MacFadyen, A. 2010a, ApJ, 722, 235

Van Eerten, H. J., Leventis, K., Meliani, Z., Wijers, R. A. M. J., & Keppens, R. 2010b, MNRAS, 403, 300

Van Eerten, H. J., Meliani, Z., Wijers, R. A. M. J., & Keppens, R. 2011, MNRAS, 410, 2016

Van Eerten, H. J., van der Horst, A. J., & MacFadyen, A. I. 2012, ApJ, 749, 44

Van Eerten, H. J., & Wijers, R. A. M. J. 2009, MNRAS, 394, 2164

Zhang, W., & MacFadyen, A. I. 2006, ApJS, 164, 255