BOXFIT v2
User Guide

Hendrik van Eerten

August 3, 2016

Contents

Chapter 1

About boxfit

This update of the BOXFIT code offers a significant number of changes and updates over the previous versions. To prevent unnecessary troubleshooting and frustration later on, please ensure to read the guide even if already familiar with BOXFIT. In particular, please keep in mind that

- Model fitting is still included, but deprecated.
- All times are in seconds (i.e. cgs units), no longer in days. Fluxes are still in mJy.
- Various settings in the parameter file have been renamed or added.
- Boosted frame BOX files require boosted frame compiled BOXFIT.
- Boosted frame BOX files do not cover late times, small boost wind case excepted. Conversely, lab frame simulations and small boost simulations do *not* properly cover very small angles ($\theta_0 < 0.045$ rad) at early times especially.
- synchrotron self-absorption is treated via a different algorithm by default.

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) 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 blast wave model starts from the self-similar relativistic solution for a spherical point explosion (Blandford & McKee, 1976), truncated at various opening angles and followed to late non-relativistic flow. The explosion can be set to occur either in a homogeneous environment (i.e. the ISM, interstellar-medium) or a progenitor-shaped stellar wind environment where density falls off with

radius ($\rho \propto r^{-2}$).

The dynamics of the afterglow blast wave have been calculated in a series of 114 high-resolution two-dimensional jet simulations performed with the RAM adaptive-mesh refinement relativistic hydrodynamics (RHD) code (Zhang & MacFadyen, 2006) both in the lab frame and in a Lorentz-boosted reference frame (van Eerten & MacFadyen, 2013). 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. As a side note, BOXFIT has not been optimized for memory management and each thread will separately load the BOX files. While this was fine for the smaller sample of 19 simulations in earlier versions, loading all BOX files (e.g. 34, for lab ISM and small boost wind) in memory simultaneously multiple times might require more memory than available on a node. In this case, either reduce the number of angles or the number of threads per node.

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 6).

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.

1.1 A brief summary of changes relative to v1

The original BOXFIT sample of dynamical templates has been greatly expanded for the re-release. The re-release of BOXFIT is part of the SCALEFIT project, that is currently being prepared for free public release. SCALEFIT is an afterglow model fit package in python that supersedes BOXFIT, making use of direct rescaling of simulation-derived synchrotron spectral templates (van Eerten & MacFadyen, 2012). This approach no longer requires on-the-fly radiative transfer computations, thus greatly increasing computational speed and allowing for model fitting on a single desktop machine. For this reason, the BOXFIT model fit routines have not been updated and are effectively deprecated. However, SCALEFIT does not offer direct access to the fluid state information provided by BOXFIT. While still usable for model fitting, the aim of the current re-release of BOXFIT is therefore to provide an extensive sample of multi-dimensional afterglow blast wave dynamical simulations and light curves that include both stellar-wind environments and very high outflow Lorentz factors (using a boosted Lorentz frame strategy, van Eerten & MacFadyen 2013). Below, the changes between v1 and v2 are summarized briefly.

- The sample of simulations has been increased from 19 simulations (ISM, opening angles from 0.045 rad to 0.5 rad) to 114, covering both ISM and stellar-wind environments. There are 34 simulations in the lab-frame and ISM, covering opening angles from 0.01 rad to $\pi/2$ (spherical explosion), and 23 boosted-frame simulations for ISM, covering opening angles from 0.01 to 0.5 rad. The stellar-

wind case also includes 34 + 23 simulations, with the difference that the 34 stellar-wind simulations were also done in a (lightly) boosted frame, for resolution reasons. Please keep in mind that ISM and wind simulations in lab frame and minor boost frame respectively are initially underresolved for $\theta_0 < 0.045$ rad (hence the complementary boosted frame set). These points are discussed further in section 1.1.1.

- The computation of synchrotron self-absorption has undergone a major change, and is described in more detail below in section 1.1.2.

- Command line support has been expanded. It is now also possible to run BOXFIT providing almost all settings directly via the command line. Command line parameters overrule those from the parameter file and typically have identical labels. For detailed description, see section 2.4.

- Error-handling and error messaging have been improved. The exiting procedure following error messages has been cleaned up and new error messages have been added. In principle, user mistakes (i.e. wrong or lacking input parameters, missing files) cannot crash the code but induce a clean exit.

- A new command-line utility is added to print the fluid states within a BOX file directly, in keeping with the new emphasis on the fluid data rather than model fitting. This tool is discussed in chapter 5.

- When computing light curves or spectra, there is now the possibility to log how the flux at a single observer time is built up from multiple emission times. This is also discussed in chapter 5.

- This user guide has been updated with not only a description of changes between v1 and v2, but also an extended discussion of simulation resolution (in the appendix).

1.1.1 Lorentz-boosted simulations and BOX files

The BOX data set has been expanded with simulations run in a Lorentz boosted frame, using the method described in van Eerten & MacFadyen (2013). The Lorentz-boosted frame method allows for a huge effective increase in computational resolution, but has the drawbacks that counter jets have to be either ignored or included explicitly on the grid, and that the Lorentz boost becomes counterproductive at late times. For the ISM case, angles up to $\theta_0 = 0.5$ rad were simulated in a boosted frame, with blast wave tip starting Lorentz factor of 100 (in the lab frame) the point where the simulations pick up the evolution from the Blandford-McKee point-explosion solution. Together with the lab-frame simulations, these cover all stages of jet evolution at high numerical resolution. For the stellar wind case, no acceptable resolution was achieved using the lab frame even at late times (consistent with results reported in the literature; the simulations discussed by De Colle et al. 2012 are also of too low resolution to be usable in the BOX framework). This was resolved by running wind simulations up to spherical flow using a small Lorentz boost of 2 and with the counter jet explicitly included on the grid. The details of the simulations will be discussed in further detail in a forthcoming publication presenting simulation-derived templates for use with BOXFIT follow-up analysis code SCALEFIT (van Eerten et al, in prep.).

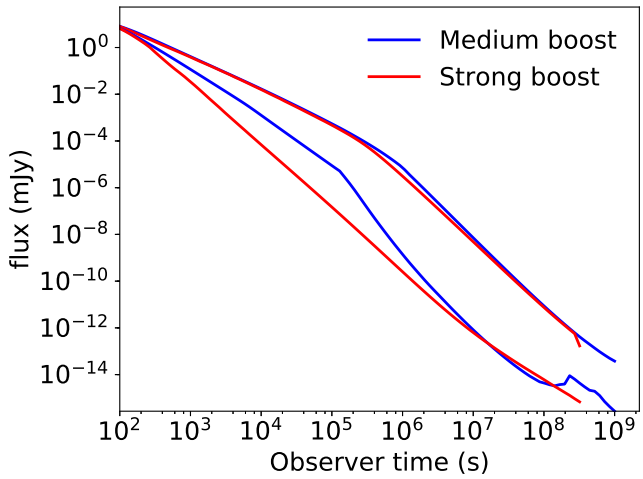

Figure 1.1: Light curves at $\nu = 10^{17}$ Hz, for a jets with opening angles of $\theta_0 = 0.01$ rad (lower curves) and $\theta_0 = 0.1$ rad (upper curves) viewed on-axis. Circumburst medium density is of stellar wind type with $n = 1$ cm^{-3} at reference distance 10^{17} cm, $E_{iso} = 10^{53}$ erg. Radiation settings are set to $p = 2.5$, $\epsilon_e = 10^{-1}$, $\epsilon_B = 10^{-2}$, $\xi_N = 1$. Observer location is set to $z = 0$, $d_L = 10^{28}$ cm.

Using boosted frame simulations with BOXFIT is straightforward. The filename of the correct BOX files should be indicated in the parameter file. BOXFIT needs to be compiled with boosted frame capabilities enabled. This setting can be found in `environment.h` in the source code, where the switch between serial and parallel computation can be found as well. Aside from pointing to the right files and setting the correct compilation switch, the user does not need to account for the boosted frame -user settings and model parameters remain unaffected. Also, it should be kept in mind that the strongly boosted frame simulations do not include a counter jet and therefore cannot produce light curves with counter jets -in fact, their output truncates at earlier time altogether.

The narrow jet simulations that have been added to the previous BOX set (which had $\theta_0 > 0.045$ rad) are at early times only fully resolved in the boosted frame. A comparison between different runs of the same θ_0 value is provided by Fig. 1.1. The figure shows wind simulations for both an average opening angle of $\theta_0 = 0.1$ rad and an ultra-narrow opening angle of $\theta_0 = 0.01$ rad. The wider jet shows mostly overlapping curves for boosted frame BOX files with a large Lorentz boost ($\gamma_S = 5$) and with small Lorentz boost ($\gamma_S = 2$). The small boost simulations pick up the Blandford-McKee solution at (lab frame) tip Lorentz factor γ of 15, and this lies fairly close to the expected onset of jet spreading for wind simulations with $\theta_0 = 0.1$ rad given by $\theta_0 \sim 1/\gamma$ (wind simulations start spreading at higher Lorentz factors than ISM simulations, see van Eerten 2013, 2015 for discussion of exact spreading times for wind and ISM). This explains the difference around the jet break for the two regular angle simulations shown in the plot, with jet spreading delayed for the top blue curve as a consequence of belated impact of the simulation initial conditions.

The narrow jet curves in the figure show a more extreme difference, with the large difference at earlier time effectively showing the difference between a spreading and a non-spreading jet -again due to the initial conditions in the small boost simulation,

where for extremely narrow jets conical flow is enforced until beyond the point of jet spreading. The curve shows another difference at very late times, with the narrow jet blue curve showing a feature akin to a counter jet. This is spurious and only shows up for very narrow small boost jets, and is a consequence of the poor initial resolution of the counter jet that was included explicitly on the grid. Wider angle jets show no counter jet appearing in the light curve in the wind case (again, consistent with the very wide ($\theta_0 = 0.2$ rad) jet result of De Colle et al. 2012). It is there in principle, but arrival time differences between jets and averaging over emission throughout the fluid when computing flux at a single arrival time render it invisible in practice.

1.1.2 Synchrotron self-absorption

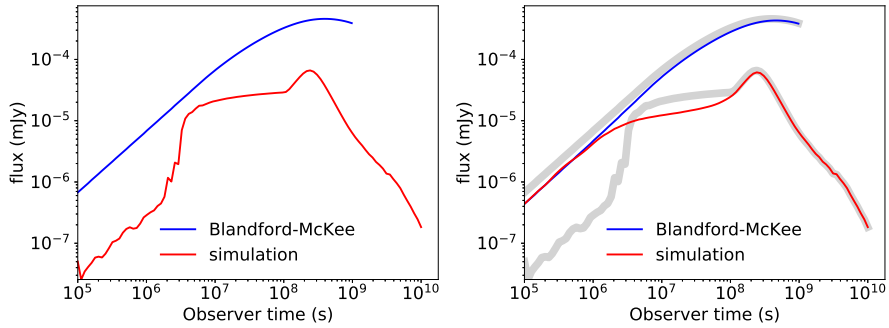

Figure 1.2: Light curves at $\nu = 1$ MHz, for a jet with opening angle $\theta_0 = 0.1$ rad viewed on-axis. Circumburst medium density is of stellar wind type with $n = 1$ cm^{-3} at reference distance 10^{17} cm, $E_{iso} = 10^{53}$ erg. Radiation settings are set to $p = 2.5$, $\epsilon_e = 10^{-1}$, $\epsilon_B = 10^{-2}$, $\xi_N = 1$. Observer location is set to $z = 0$, $d_L = 10^{28}$ cm. The figures illustrate the failure of the local synchrotron self-absorption algorithm to successfully capture early time emission for these parameters. The red lines show simulation (BOX) output, the blue lines show output from a conical flow with fixed opening angle $\theta_0 = 0.1$ rad obeying the Blandford-McKee solution at all times. In the right plot, a global algorithm for s.s.a. is used instead of a local one and the early time red light curve converges properly to the analytical solution in blue. The local computation from the left figure is repeated in grey, indicating both the systematic shift in self-absorption break ν_a between the two approaches and the crossing of ν_a through the observer band (after which the algorithm used for self-absorption becomes irrelevant).

Previously, synchrotron self-absorption was calculated by solving the equation of transfer simultaneously for a large number of rays using locally determined emission and absorption coefficients (Van Eerten et al., 2010b):

$$\frac{\mathrm{d}I_\nu}{\mathrm{d}z} = \epsilon_\nu - \alpha_\nu I_\nu. \tag{1.1}$$

This was numerically implemented via

$$\Delta I_\nu = \begin{cases} I_\nu \left(e^{-\Delta\tau} - 1\right) + S_\nu \left(1 - e^{-\Delta\tau}\right), & \text{if } \Delta\tau > 10^{-3}, \\ I_\nu \Delta\tau \left(\frac{\Delta\tau}{2} - 1\right) + \epsilon_e \Delta z \left(1 - \frac{\Delta\tau}{2}\right), & \text{otherwise,} \end{cases} \tag{1.2}$$

where $S_\nu \equiv \epsilon_\nu/\alpha_\nu$ the source function and $\Delta\tau \equiv \alpha_\nu \Delta z$ the optical depth across distance Δz (see also `eds_2D_regular.cpp` of the source code).

However, this approach is found to be insufficient when dealing with the early time regime opened up by boosted frame simulations and for the (early) stellar wind case. When self-absorption is computed locally, the outcome is highly sensitive to the local state of the fluid flow. Specifically, small deviations from the Blandford-McKee solution that arise temporarily after the Blandford-McKee solution is used to set the initial conditions of a numerical simulation end up having an exaggerated impact on the computed flux. This can be understood by reminding oneself that in the self-absorbed limit only the surface of the jet is seen, while in the optically thin case the entire volume is seen. In other words, a dimension of integration for computing the observed flux is effectively lost, as is its smoothing effect damping the impact of local fluid perturbations.

This issue is not unlike that between local and global cooling (see e.g. Van Eerten et al. 2010a for discussion on electron cooling). We have therefore switched to a different approach to self-absorption. As with global electron cooling, synchrotron self-absorption is now computed in a global manner, leading to a systematic offset in the self-absorption break position that can be compensated for a posteriori if desired. Emerging rays at z_{max} are now given by

$$I_\nu(z_{max}) = \tilde{S}_\nu \left(1 - e^{-\tau}\right), \tag{1.3}$$

where

$$\tau \equiv \int \mathrm{d}z \alpha_\nu, \tag{1.4}$$

and

$$\tilde{S}_\nu \equiv \frac{\langle \epsilon_\nu \rangle}{\langle \alpha_\nu \rangle}. \tag{1.5}$$

The average emission and absorption in turn are given by

$$\langle \epsilon_\nu \rangle \equiv \frac{\int \mathrm{d}z \epsilon_\nu}{\int \mathrm{d}z}, \tag{1.6}$$

and

$$\langle \alpha_\nu \rangle \equiv \frac{\int \mathrm{d}z \alpha_\nu}{\int \mathrm{d}z}. \tag{1.7}$$

In the optically thin limit the emerging intensity equation reduces to

$$I_\nu(z_{max}) = \int \mathrm{d}z \epsilon_\nu. \tag{1.8}$$

A comparison between the methods for the stellar wind environment case is shown in figure 1.2.

Chapter 2

Getting started, light curves and spectra

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 114 data files that contain the compressed results of the jet dynamics simulations and, in the case of data 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 9GB 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 6, 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

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

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. 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. These are not part of the source code tarfile because of their size, but need to be downloaded separately. Their filenames are `boxISM_00.h5`, `boxboostmedwind_00.h5` etc. The path to the datafiles 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 `boxfit*.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 OPEN_MPI_            DISABLED_
  ```

- in the same file `environment.h`, ensure that compilation for Lorentz boosted frame BOX files is disabled, since we will plot an ISM light curve from lab frame BOX files (`boxISM_xx.h5`; all other BOX files `boxboostISM_xx.h5`, `boxboostwind_xx.h5`, `boxboostmedwind.h5` are Lorentz-boosted):

  ```
  #define BOOST_             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 also be written to a textfile in addition 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 setting 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 5×10^4 s (0.6 days) is the jet break. The late time bump is the counter jet. 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.

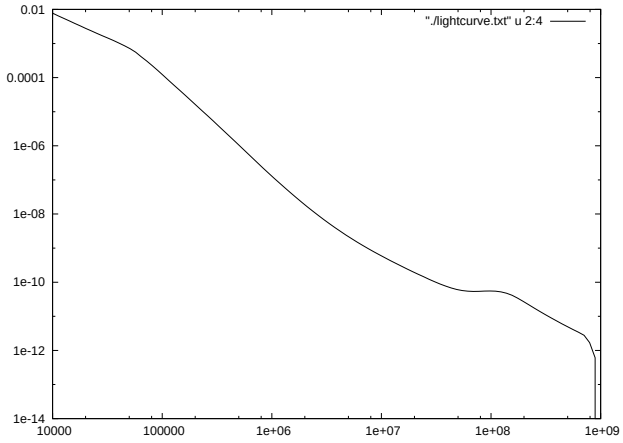

Figure 2.1: `gnuplot` output for a light curve calculated at 10^{17} Hz according to standard settings. Note the drop to zero flux at late times. Horizontal axis shows observer time in seconds, vertical axis monochromatic flux in mJy.

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).

what_to_do

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

-1. Perform same 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.

In version 2, the fit routines are deprecated.

box_filename_base

Sets the base (i.e. excluding numbers and file extension `.h5`) file name of the BOX files, e.g. `boxISM_` etc. Double quotes need to be included. To set the path to the current directory, for example, use `"./"`.

box0, box1

Which BOX to load. To only plot a narrow jet with $\theta_0 = 0.01$ rad (the smallest available angle), for example, `box0 = 0`, `box1 = 1` would be sufficient. The flux at arbitrary angle θ_0 is computed as an interpolation between surrounding simulated angles. BOXFIT therefore requires at least two BOX files to be loaded into memory.

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 data points 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`.

save_emission_profile

A binary setting that has an effect only when a light curve or spectrum is calculated (i.e. `what_to_do` is 1 or 2). If 0, no additional log files are written. If 1, two data files are written for each observation. These contain $\Delta F_\nu / \Delta t_e$ for each emission time t_e in $F_\nu(t_{obs}) = \int \mathrm{d}t_e \left(\mathrm{d}F_\nu/\mathrm{d}t_e\right)$. The two data files describe the flux as computed for the tabulated θ_0 entries surrounding the requested θ_0. In case θ_0 lies very close to a tabulated entry, only one log file is written per observation.

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 seconds.

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 (`boxISM_00.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 BOX 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 Lorentz factor of the fluid directly behind the shock front. Recommended value for `BM_start` is around 300. If `BM_stop` is set to a negative number, the starting (lab frame, jet tip) Lorentz factors for the simulation are used (25 for ISM lab, 15 for medium boost wind, 100 for large boost simulations). For a run using only the BM solution, a lower limit of around 1.2 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 within 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`. The observed flux is the area integral over the region covered by a total of `eds_r_res` $\times$ `eds_phi_res` rays. For an on-axis observation, `eds_phi_res` value can be set equal to 1 due to symmetry (i.e. the image on the sky is circular). The larger the observer angle, the more important the ϕ-resolution becomes, while the *r*-resolution can be decreased.

fluid_res

Integer setting the emission time resolution of logarithmically spaces time steps in the radiative transfer line integrals, including both times covered by the BM solution and by BOX data. While the number of rays is set by `eds_r_res` and `eds_phi_res`, `fluid_res` sets the resolution along the rays. Values of a few thousands will lead to a converged result. Much smaller values can be used for test runs. For observers far off-axis, the time resolution needs to be increased again, because in these cases the rays spend less time within the emitting region (i.e. the effect of the blast wave nearly keeping up with its emission is gone for rays not moving along the blast wave direction of motion).

jet

Setting determining whether to include the forward jet only (use `forward`), the counter jet or receding jet only (use `receding`) or both jets (use `both`) when computing flux. Large boost BOX files do not include counter jet data and if `jet` is set to include counter jet emission in these cases, a warning is issued.

eds_ur_max

A float value that can be set to a positive number in order to overrule the default automatic algorithm computing the radial extent of the collection of rays on the plane

pointing to the observer. If set to a negative number, BOXFIT will determine by itself up to which maximum value of r the `eds_r_res` rays will be distributed. Best left to its default setting < 0.

t_e_0, t_e_1

Two float values that can overrule automatically determined settings when set to positive values, like `eds_ur_max`. If set to negative values (recommended default), the included emission times are set automatically and bound by what is covered by the BOX files and `BM_start` and `BM_stop` settings.

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 multiplicative 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 data points.

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 at reference distance 10^{17} cm in cm^{-3} (1 for ISM, 29.89 for wind), `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 step size used in the derivative.

2.4 Command line options

BOXFIT will always require the presence of a parameter file (as described above in section 2.3), either in the current directory and with filename `boxfitsettings.txt`, or under a filename provided as the first argument. Nevertheless, essentially all settings from the parameter file can be overruled at the command line. Below follows a list of command line options. Their syntax is chosen to match that in the parameter file, and the effect of the settings is the same between parameter file and command line.

`-what_to_do=[int]`	which action to perform
`-data_filename=[string]`	filename containing burst data
`-box_filename_base=[string]`	BOX data filename base
`-save_intermediate=[0,1]`	save intermediate fit results
`-save_emission_profile=[0,1]`	save emission profile for `what_to_do` = 1,2
`-t_0=[float]`	time spectrum / earliest light curve time (days)
`-t_1=[float]`	latest light curve time (days)
`-nu_0=[float]`	light curve frequency / lowest spectrum frequency (Hz)
`-nu_1=[float]`	highest spectrum frequency (Hz)
`-d_L=[float]`	luminosity distance (cm)
`-z=[float]`	redshift
`-theta_0=[float]`	(initial fit) jet opening angle (rad)
`-E=[float]`	(initial fit) isotropic equivalent energy (erg)
`-n=[float]`	(initial fit) circumburst number density at 10^{17} cm (cm^{-3})
`-theta_obs=[float]`	(initial fit) observer angle (rad)
`-p=[float]`	(initial fit) synchrotron p
`-epsilon_B=[float]`	(initial fit) synchrotron ϵ_B
`-epsilon_E=[float]`	(initial fit) synchrotron ϵ_E
`-ksi_N=[float]`	(initial fit) synchrotron ξ_N
`-theta_0_max=[float]`	maximum fit range for opening angle
...	...
`-theta_0_min=[float]`	minimum fit range for opening angle
...	...

`-theta_0_frozen=[0,1]`	freeze opening angle during fit
...	...
`-theta_0_simplex_max=[float]`	initial simplex maximum range
...	...
`-self_absorption=[0,1]`	include synchrotron self-absorption
`-electron_cooling=[0,1]`	include electron cooling
`-include_BOX=[0,1]`	include BOX data
`-include_BM=[0,1]`	include Blandford-McKee solution
`-start_from_simplex=[0,1]`	whether to start from simplex.
`-fluid_res=[integer]`	emission time resolution.
`-eds_r_res=[integer]`	radial direction number or rays.
`-eds_phi_res=[integer]`	angular direction number of rays.
`-temp=[float]`	annealing starting temperature.
`-temp_factor=[float]`	annealing temperature decrease factor.
`-temp_lowest=[float]`	lowest annealing temperature.
`-iter=[integer]`	iterations per annealing temperature.
`-MC_runs=[integer]`	number of Monte Carlo runs.
`-box0=[integer]`	lowest BOX file index number.
`-box1=[integer]`	highest BOX file index number.
`-jet=[string]`	included jets, `forward`, `receding` or `both`.
`-no_points=[integer]`	number of light curve / spectrum points.
`-BM_start=[float]`	starting BM Lorentz factor.
`-BM_stop=[float]`	stopping BM Lorentz factor.
`-eds_ur_max=[float]`	perpendicular extent of rays.
`-t_e_0=[float]`	earliest included emission time.
`-t_e_1=[float]`	latest included emission time.

Chapter 3

Model fitting (deprecated)

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 models to datasets.

3.1 Running on a cluster

Fitting a model 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. This is typically done 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 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`, ldots 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 (see also source code file `numerical.cpp`):

```
# 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`.

3.4 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 data points 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].

3.4.1 Setting the partial derivatives

The first step in determining the error bars is to calculate for each data point 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 data point with best fit flux F_i with respect to a fit parameter x, so $\partial F_i/\partial x$, is calculated with a step size 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 appropriate 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:

- $\mathrm{d}\theta_0 \sim 0.05$

[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 data points in different bands can vary wildly and a sparsely populated band dropping out affects the fit constraints in a fundamentally different way than shifting its values will, even if the error bars are large.

- $\mathrm{d}\log E_{iso} \sim 0.1 - 1.0$ (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 appropriately translated to the scale of the best fit results.

3.4.2 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. **Calculation slows to a crawl.** Attempting to load too many BOX files into memory simultaneously. The parallel version of BOXFIT is quite dumb when it comes to memory management, with each thread loading the data separately. In this case, either reduce the number of angles or the number of threads per node.

2. **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 0.01 and 0.5 radians (inclusive) for large boost simulations and up to $\pi/2$ for lab and small boost simulations.

3. **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 or extended energy injection phase (the expected physical origin of the plateau phase in early *Swift* data).

4. **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 for lab ISM, 15 for small boost wind, 100 for large boost). 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.

5. **numerically noisy light curves, unexpected features, spurious wind case counter jet.** With the availability of both boosted and lab / small-boost simulations, make sure that they are applied within their regime of validity.

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

7. **Not all data points are used.** The number of datapoints 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 datapoints specified on the first line is included, further datapoints 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.

8. **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).

9. **Annealing 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).

10. **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.

11. **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 step size for the derivative. If a step size 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 step size than dynamical parameters.

12. **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.

13. **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

log files and fluid state dumps

The emphasis for BOXFIT has shifted from being a data analysis package to a versatile tool to explore the dynamics and emission of collimated relativistic blast waves. For this reason, the code has been extended with emission log file functionality and a separate command-line to for dumping fluid states.

5.1 Log files for light curves and spectra

When computing light curves or spectra, it is now possible to enable log file output for each measurement. This is done by setting the switch `save_emission_profile` to 1. Two data files are written for each observation. These contain $\Delta F_\nu / \Delta t_e$ for each emission time t_e in $F_\nu(t_{obs}) = \int \mathrm{d}t_e \left(\mathrm{d}F_\nu/\mathrm{d}t_e\right)$. The two data files describe the flux as computed for the tabulated θ_0 entries surrounding the requested θ_0. In case θ_0 lies very close to a tabulated entry, only one log file is written per observation.

The log files start with a header describing the user settings. An example header is:

```
##################################################
# Log file for processor 1. Data point 0
##################################################
# Observer settings:
#   r_obs = 1.000000e+28 cm
#   theta_obs = 0.000000e+00 rad
#   z = 0.000000e+00
#   nu_obs = 1.000000e+17 Hz
#   t_obs = 1.000000e+04 s (1.157400e-01 days)
# Radiation settings:
#   electron cooling is enabled
#   self-absorption is enabled
#   self-absorption computed globally
#   synchrotron slope p = 2.500000e+00
#   epsilon_E = 1.000000e-01
#   epsilon_B = 1.000000e-02
#   ksi_N = 1.000000e+00
# BOX and resolution settings:
#   Included emission time range: 2.101253e+06 - 3.325927e+08 (s)
#   Included radial extent emission image: 2.274036e+18 (cm)
```

```
#   ur_rays = 1000
#   uphi_rays = 1
#   Blandford-McKee analytical solution included
#   Initial BM Lorentz factor: 3.000000e+02
#   BOX solution included.
#   BOX filename: /home/hveerten/boxfit/data/boxISM_07.h5
#   Computing both forward jet and receding jet
#-----------------------------------------------
# i, t_e (sim frame), F(t_e), D F, D t_e, extent ur
```

This is followed by a series of output values grouped in 6 columns, with the meaning of the columns as indicated in the final line of the header above. Column (1) contains the step number along the rays, column (2) the emission time (in the simulation frame, so can be boosted with $\gamma_S = 2$, or $\gamma_S = 5$) for this step, column (3) the cumulative flux up to this point, column (4) the differential flux for this step, column (5) the differential emission time for this step and column (6) the current size-on-the-sky or radial extent within the plane perpendicular to the observer direction. The file closes with a concluding statement on emission times and radial extent, for example

```
#-----------------------------------------------
# Earliest contributing emission time estimate: 2.101253e+06 (s)
# Latest contributing emission time estimate: 2.962967e+07 (s)
# Radial extent of emission image: 2.253188e+16 (cm)
# extent found / extent set = 9.908319e-03
```

In this particular example, the automatic algorithm determining emission times started from `2.101253e+06`, identical to the finally reported estimate. This tells us that a flux contribution is observed immediately from this earliest included emission time onward. Apparently the rays were computed starting within the blast wave rather than behind it. We therefore have to be careful in interpreting the resulting flux value for this first measurement, since emission from earlier times that were not covered could have made a difference in total flux. For this particular case, this is not unexpected since the observer time is set to an early 10^4 seconds. Whether it means we need to set `BM_start` to a higher value for full coverage, or if there is no practical difference between initial BM Lorentz factor of 300 and a still higher one, can be determined by experimenting with different values. The automatically computed final emission time of `3.325927e+08` lies well beyond the reported final contributing time of `2.962967e+07`, so the rays have had the time to emerge from the blast wave.

The final size-on-the-sky for this measurement is `2.253188e+16` cm, well below the automatically computed `2.274036e+18` so the image has not been truncated and outer rays have passed the blast wave along the sides. Both time steps and radial ray distributions are logarithmically, which is why the two orders of magnitude difference between `2.253188e+16` and `2.274036e+18` is no cause for concern (and given by `9.908319e-03`).

Emission times and radial extent can be tweaked by the user with `t_e_0`, `t_e_1` and `eds_ur_max` as indicated previously in this manual. When tweaking, keep in mind that the values change between observer times, as can be verified by comparing log files for different observer times in a light curve.

5.2 Fluid state dumps

A separate command-line tool `dump_box` is provided to dump fluid states directly. Running `dump_box` without parameters prints a list of command line options:

```
use: "dump_box <filename> -t=[float]" or "dump_box <filename>
      -snapshot_time=[int]"
Overview of command-line settings:
  -t=[float]               BOX time in seconds, comoving frame
  -snapshot_time=[int]     BOX time, taken from stored BOX times,
      typically ranging from 0 - 100 (inclusive)
  -xmin=[float]            lower boundary Cartesian, lab frame x
      (cm), default is 0
  -xmax=[float]            upper boundary Cartesian, lab frame x
      (cm), default is c * t
  -ymin=[float]            lower boundary Cartesian, lab frame y
      (cm) ,default is 0
  -ymax=[float]            upper boundary Cartesian, lab frame y
      (cm), default is c * t
  -xres=[int]              resolution of BOX image x direction,
      default is 512
  -yres=[int]              resolution of BOX image y direction,
      default is 512
  -var=[int]               fluid variable to output, by number,
      default is 0 (rho)
  -varname=[string]        fluid variable to output, by name,
      default is rho
  -help                    print this message, and quit
  -output=[string]         name of output hdf5 file. Is set to
      dump.h5 by default
  -quiet, -q               avoid output to stdout. Disabled by
      default

The following command-line settings trigger a single slice dump
      at fixed theta and directly to stdout
  -slice, -s               Switch to 1D slice at fixed theta
  -rmin=[float]            lower boundary radial r (cm), default 0
  -rmax=[float]            upper boundary radial r (cm), default
      c*t
  -snapshot_rmax=[int]     upper boundary radial r, taken from
      stored BOX peak radii
  -fractional_rmin=[float] lower boundary radial r, as fraction
      of upper boundary
  -rres                    resolution r direction, default 100
  -slice_theta             angle theta of slice (rad), default 0

Fluid variables by name and number:
  0   rho    comoving density (gram cm^-3)
  1   eint   internal energy density (erg cm^-3)
  2   v_x    velocity component in generalized x-direction
```

```
    (i.e. r-direction), lab frame, units of c
3   v_y    velocity component in generalized y-direction
    (i.e. theta-direction), lab frame, units of c
4   v      magnitude of velocity, lab frame, units of c
5   lfac   Lorentz factor, lab frame
6   D      lab frame density (gram cm^-3)
9   N      comoving number density (cm^-3)
```

If 2D snapshots are printed, the output will be written in hdf5 format. These can then be plotted using standard tools in python (i.e. matplotlib). A detailed description of matplotlib plotting lies beyond the scope of this guide. Single slices at fixed angle can be plotted as well, these a written directly to `stdout` and can be plotted quickly using e.g. gnuplot. The datasets in the 2D hdf5 files are labeled and can be studied with hdf5 tools such as `hdfview` as well. The plotted fluid variable is labeled w, the Cartesian coordinates are labeled x, y. In the case of slices, two columns are printed containing radius and fluid value respectively.

Chapter 6

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 appriopate 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.

dump_box

a stand-alone tool for dumping fluid states. Contains a single class which reads input parameter files and accesses the BOX classes from `box.h` to read the local fluid state for the requested time and coordinates.

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 collects access to all BOX files in a single class. In this version of BOXFIT no interpolation between opening angle θ_0 entries takes place at the fluid level anymore, and fluxes are determined via interpolation at the flux level.

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 data point 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 data point and will separately calculate the radiative transfer for the observer time and frequency relevant to this data point. Once this calculation is finished the core will request a new data point from the master core, until all fluxes are calculated.

flux

The class in `flux_from_box` builds upon the base class provided by `flux.cpp` and `flux.h`. Although irrelevant for BOXFIT, this division between classes is motivated by different flux algorithms interacting with the radiation code (for example, an alternative to `flux_from_box` that computes flux directly from RHD simulation data dumps).

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).

observer

Provides a class `c_observer` that groups together observer coordinates, measurement time and frequency and some routines to transfer between different coordinate systems.

param_IO

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

parse

All routines for reading command-line arguments are defined here.

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 if for implementing changes in the radiative process.

Bibliography

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

De Colle, F., Ramirez-Ruiz, E., Granot, J., & Lopez-Camara, D. 2012, ApJ, 751, 57

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. 2013, paper 24 in eConf Proceedings C1304143. ArXiv: 1309.3869

van Eerten, H., & MacFadyen, A. 2013, ApJ, 767, 141

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

van Eerten, H. J. 2015, Journal of High Energy Astrophysics, 7, 23

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

van Eerten, H. J., & MacFadyen, A. I. 2012, ApJ, 747, L30

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