Draft version March 10, 2022
Typeset using LaTeX default style in AASTeX631

# Gamma-ray Burst Prompt Emission Simulation Code Manual

Michael Moss[1, 2, 3]

[1] The Department of Physics, The George Washington University, 725 21st NW, Washington, DC 20052, USA
[2] Astrophysics Science Division, NASA Goddard Space Flight Center, Greenbelt, MD 20771, USA
[3] Sorbonne Université, CNRS, UMR 7095, Institut d'Astrophysique de Paris, 98 bis Arago, 75014 Paris, France

## Contents



Corresponding author: Michael Moss
mikejmoss3@gmail.com

## 1. INTRODUCTION

The code outlined in this document aims to simulate the Gamma-Ray Burst (GRB) prompt emission by assuming shells of material are ejected from a central engine with varying speeds and then using semi-analytical descriptions of emission process to calculate the expected spectra and light curves.

## 2. CODE STRUCTURE

The core of the simulation code resides in SynthGRB.cpp which is responsible for calculating the dynamics and emission of each simulation.

1. Classes
2. Classes methods
3. Schematic

## 3. PERFORMING GRB PROMPT SIMULATIONS

Below I will outline the necessary steps to generate the instantaneous jet dynamics parameters, a spectrum, and a light curve for a given set of input parameters..

In order to run a simulation, a user will edit main.cpp. The first thing to be done is to create a SynthGRB object. A SynthGRB object can be initialized three separate ways,

```
    // Definitions
    SynthGRB(float tw, float dte,
        double E_dot_iso, float theta, float r_open, float eps_th, float sigma,
        float eps_e_int, float eps_b_int, float zeta_int, float p_int,
        float eps_e_ext, float eps_b_ext, float zeta_ext, float p_ext,
        float k_med, double rho_not,
        std::string LorentzDist, std::string ShellDistParamsFile);
    SynthGRB(ModelParams * input_model_params);
    SynthGRB();

    // Example function calls
    example_grb = SynthGRB();
```

**Listing 1.** SynthGRB Constructors

The first constructor requires user input for the wind duration, time between shell launching, shell Lorentz distribution, and all microphysical parameters at the time of initialization. The second constructor will load all of this information from a given ModelParams object. The third constructor will take the default values for all input arguments, but these can immediately be reset by the user. To load all of the input parameters from a file a user can include the line,

```
    // Definitions
    SynthGRB::LoadJetParamsFromTXT(std::string file_name);

    // Example function calls
    example_grb.LoadJetParamsFromTXT("input-files/jet-params.txt");
```

**Listing 2.** Loading Jet Parameters from a Given Text File

In the example above, all input parameters are given in the "input-files/jet-params.txt" (where input-files/ is a directory within the simulation code directory). Here is what the parameter file looks like,

```
### tw, sec ###
30.
### dte, sec  ###
0.02
### E_dot_iso, erg/s  ###
1.e52
### theta, rad  ###
0.1
### r_open, cm ###
3.e6
```

```
### eps_th, fraction  ###
0.03
### sigma, dimensionless ###
0.1
### eps_e internal, fraction ###
0.33
### eps_b internal, fraction ###
0.33
### zeta internal, fraction ###
1.e-3
### p internal, dimensionless ###
2.2
### eps_e external, fraction ###
0.13
### eps_b external, fraction ###
1.e-3
### zeta external, fraction ###
1.
### p external, dimensionless ###
2.2
### k, wind parameter, i.e., 0, 2 ###
0
### rho_not, density normalization (for k = 0, g cm^-3, i.e., n0*mp = n0*1.672e-24 | for k = 2, g cm^-1, i.e., A_star * 5.e11 ) ###
1.672e-24
### LorentzDist, i.e., step, smoothstep, osci ###
gauss_inject
### ShellDistParamsFile, i.e., ./input-files/jet-shells-"LorentzDist".txt ###
./input-files/jet-shells-gauss_inject.txt
```

**Listing 3.** Jet Parameters Defined in a Text File

All lines that begin with the '#' character will be ignored when being read by SynthGRB. The parameters included in the text file must match the order of the parameters as defined in the first SynthGRB constructor list above. When a SynthGRB is initialized or when a new set of jet parameters are loaded from a file, the SynthGRB object will immediately uses the ShellDist class to calculate and store the initial radius, initial Lorentz factor, initial mass, ejection time of all shells which will be propagated in the jet dynamics simulation.

To perform a jet dynamics simulation a user simply has to call the SynthGRB class method,

```
    // Definitions
    SynthGRB::SimulateJetDynamics();

    // Example function calls
    example_grb.SimulateJetDynamics();
```

**Listing 4.** Perform Jet Dynamics Simulation

All jet dynamics data will be stored in memory within the SynthGRB class. After the jet dynamics have been calculated, a spectrum and light curve can be calculated by calling the methods below,

```
    // Definitions
    SynthGRB::make_source_spectrum(float energ_min = 50., float energ_max = 350., int num_energ_bins = 50, float tmin = 0., float tmax = 30., std::string comp = "all");
    SynthGRB::make_source_light_curve(float energ_min, float energ_max, float Tstart, float Tend, float dt, std::string comp = "all", bool logscale = false);

    // Example function calls
    example_grb.make_source_spectrum();
    example_grb.make_source_light_curve();
```

**Listing 5.** Create Spectrum and Light Curve from Jet Simulation

A user can then write out all of the data using the functions below. These functions do not need to be called at the end, each data set can be written out to text as soon as the relevant calculation has been made (e.g., the jet parameters can be written out directly after the jet dynamics simulation).

```
    // Definitions
    ShellDist::WriteToTXT(std::string out_file_name);
```

```
    SynthGRB::write_out_jet_params(std::string dir_path_name);
    SynthGRB::WriteSpectrumToTXT(std::string out_file_name);
    SynthGRB::WriteLightCurveToTXT(std::string out_file_name);

    // Example function calls
    (*example_grb.p_jet_shells).WriteToTXT("data-file-dir/synthGRB_shell_dist.txt");
    example_grb.write_out_jet_params("./data-file-dir/");
    example_grb.WriteSpectrumToTXT("data-file-dir/synthGRB_spec_total.txt");
    example_grb.WriteLightCurveToTXT("data-file-dir/synthGRB_light_curve.txt");
```

**Listing 6.** Write Out Data to Text Files

## 4. HANDLING SIMULATION DATA

The plotting is done completely in Python and uses text files created by the C++ code described in Section 3. We can plot the initial Lorentz distribution like so,

```
# Definitions
def plot_lor_dist(file_name,ax=None,save_pref=None,xlabel=True,ylabel=True,label=None,fontsize=14,fontweight='bold',linestyle='solid', separator_string = "// Next step\n")

# Example function calls
plot_lor_dist('data-file-dir/synthGRB_shell_dist.txt')
```

**Listing 7.** Plotting Lorentz Distributions

The above method will plot the given Lorentz factor distribution saved in the text file with path name ”file˙name”. Multiple snapshots of the Lorentz distribution can be given in a single file. Each Lorentz distribution must be separated by a line with the string indicated by ”separator˙string”. If more than one snapshot is provided, the snapshots can be scrolled through with the left and right arrow keys. The Lorentz distribution file must contain the columns:

1. RADIUS - Radius of the shell
2. GAMMA - Lorentz factor of the shell
3. MASS - Mass of the shell
4. TE - Time of emission of the shell
5. STATUS - Status of the shell, this is used by the simulation code to indicate if a shell is still active or not.

To plot the evolution of the instantaneous jet dynamics we must first load the data. Once the data is loaded, the evolution of the thermal, internal shock, and external shock components can be plotted separately, but it is useful to view the internal and external shock components on the same plots.

```
# Definitions
def load_therm_emission(file_name)
def load_is_emission(file_name)
def load_fs_emission(file_name)
def load_rs_emission(file_name)

def plot_param_vs_time(emission_comp,param,frame="obs",ax=None,z=0, y_factor=1, label=None, Tmin=None, Tmax=None,save_pref=None,fontsize=14,fontweight='bold',disp_xax=True,disp_yax=True,
color='C0',marker='.',markersize=7,alpha=1)
def plot_evo_therm(thermal_emission,frame="obs",ax=None,z=0,Tmin=None, Tmax=None,save_pref=None,fontsize=14,fontweight='bold')
def plot_evo_int_shock(is_emission,frame="obs",ax=None,z=0,Tmin=None, Tmax=None,save_pref=None,fontsize=14,fontweight='bold')
def plot_together(is_data = None,fs_data=None, rs_data=None,frame="obs", z=0, Tmin=None, Tmax=None,save_pref=None,fontsize=14,fontweight='bold',markregime=True,markersize=10)

# Example function calls
th_data = load_is_emission("data-file-dir/synthGRB_jet_params_th.txt")
is_data = load_is_emission("data-file-dir/synthGRB_jet_params_is.txt")
fs_data = load_fs_emission("data-file-dir/synthGRB_jet_params_fs.txt")
rs_data = load_rs_emission("data-file-dir/synthGRB_jet_params_rs.txt")
```

```

plot_evo_therm(th_emission)
plot_evo_int_shock(is_data)
plot_evo_ext_shock(fs_data=fs_data,rs_data=rs_data)
fig0, fig1 = plot_together(is_data=is_data,fs_data=fs_data,rs_data=rs_data)
```

**Listing 8.** Plotting Instantaneous Jet Dynamics Parameters

A plot of the spectrum can be created simply by specifying the text file which contains the spectrum. The spectrum text file must contain three columns; (i) lower bound of energy bin, (ii) count rate, (iii) uncertainty in the count rate. The FermiGBM and SwiftBAT energy bands can also be added to these plots.

```
# Definitions
def plot_spec(file_name, z=0, joined=False, label = None, color="C0", ax=None, nuFnu=True, unc=False, Emin=None, Emax=None, save_pref=None,fontsize=14,fontweight='bold')
def add_FermiGBM_band(ax,fontsize=12,axis="x")

# Example function calls
ax_spec = plt.figure(figsize=(9,8)).gca()
plot_spec("data-file-dir/synthGRB_spec_total.txt",ax=ax_spec,z=z,label="Total",color="k")
add_FermiGBM_band(ax_spec)
```

**Listing 9.** Plotting Spectra

Similar to plotting a spectrum, a plot of a light curve can be created by specifying the text file which contains the light curve. The light curve text file must contain two columns, one for the time bin and the second for the count rate.

```
# Definitions
def plot_light_curve(file_name, z=0, label=None, ax=None, Tmin=None, Tmax=None, save_pref=None, color="C0", fontsize=14,fontweight='bold', logscale=False)

# Example function calls
ax_lc = plt.figure().gca()
plot_light_curve("data-file-dir/synthGRB_light_curve.txt",ax=ax_lc,z=z,label="Total",logscale=False,color="k")
```

**Listing 10.** Plotting Light Curves

Lastly, an interactive plot can be made to look at the light curves and spectra generated by the simulation code using the following lines of code. The initial times and energies will set the absolute maximum range of the times and energies that can be viewed. To increase the time or energy interval being plotted, the function should be called again and the initial time or energy interval should be increased.

```
# Definitions
def plot_light_curve_interactive(init_Tmin, init_Tmax, init_Emin, init_Emax, z=0, with_comps=False, label=None, ax=None, save_pref=None, fontsize=14,fontweight='bold',logscale=False)

# Example function calls
tbox = plot_light_curve_interactive(init_Tmin = 0, init_Tmax = 20, init_Emin = 8, init_Emax = 1e4,z=z,label="Total",with_comps=True)
```

**Listing 11.** Plotting Interactive Synthetic Spectra/Light Curves