# `CODE NAME` [1] User's Manual

Marco Sortino

March 12, 2024

[1] Name still to be choosen

# Contents



## Overview

`CODE NAME` is a tool that allows user to translate FLASH simulated AGNs Jets outputs in spectra. It is capable of elaborate the Spectral Energy Distribution (SED) produced by each blob and how the light interacts with each sphere. It

comes with the option to define the distance and the angle of view of the source with respect to the observer, as well as the radiative processes to consider.

Given the nature of FLASH and the block structure it comes out with, `CODE NAME` allows to select which block should take place in the emission process. For this reason `CODE NAME` comes with a small routine that produces histograms and contour plots based on density, pressure, temperature and energy of each block.

The currently available processes are:

- Non-Thermal Bremmstrahlung;
- Synchrotron;
  - Synchrotron Self Absorption;
  - Synchrotron Self Compton;
- External Compton;

The External Compton interacting region available are:

- Cosmic Microwave Background (CMB)
- Shakura Sunyaev Disc (SSDisc)

There are two codes provided: `main` and `plotting`.
The first one is the one used in order to compute the SED. The settings parameters are stored in `config.txt`. More info on how to customize it is found in section 1.3.2.
The latter is used to produce histograms and contour plots. The respective settings parameters are stored in `config_plot.txt`.
They are briefly explained in the following sections.

Soon to be available are the Dust Torus (DT) and the Broad Line Region (BLR).
This code wouldn't have seen the light of the day if not for the `agnpy` library made by Nigro et al (2022).

### 0.0.1 Changelog

On this date (December 08, 2023) the code has been modified:

- Fixed the previous issues on External Compton, CMB ray tracing and Disc energy loss.
- Added a convex hull algorithm to properly track the blobs subjected to CMB and External Compton

On this date (March 06, 2024) the code has been modified:

- Added `plotting.py`.
- Added RayTracing algorithm in C.
- Overhauled the general structure and collocation of libraries.

## Issues

On this date (October 20, 2023), the current known issues are:

- External Compton and Synchrotron Self - Compton
  Blobs and photon interaction is not correctly tracked, resulting the source to appear more bright in the high frequencies range. —- FIXED
- Ray tracing
  The current algorithm is too slow. —- FIXED
- Disc energy loss
  Currently deactivated as it brings issues with the computation —- FIXED
- CMB ray tracing
  Disabled as there are some issues on how to correctly take the opacity into account. —- FIXED

# Chapter 1

# How to Use

## 1.1 Installation

`CODE NAME` currently works on `python3.7` or above. The following libraries need to be installed: `numpy, matplotlib, astropy, agnpy, sherpa, gammapy`.

It has been correctly ran and tested on Ubuntu 23.02. It does **not** work on Windows since `sherpa` is not supported. Any tests on Mac has not been made yet, hence the owner cannot guarantee the correct functioning of the code.

The main directory display the following structure:

- `main.py`
- `plotting.py`
- `config.txt`
- `config_plot.txt`
- `lib/`
- `compiler.sh`

The latter is a bash script and must be run when installing for the first time. On Linux, it can be launched via terminal using:

```
sh compiler.sh
```

It will simply compile a C code that is needed for the LightRayIntersection routine that is used later. The script must be launched via terminal from the same directory in which it is located. Alternatively one can simply compile it directly in `lib/c\_routines/codes/` using

```
gcc -g -Wall raytrace.c -o raytrace -lm
```

and then move the output to `lib/c\_routines/exec/`.

## 1.2 How to use it

In order to compute the radiative processes we are interested in, it is necessary to edit `config.txt` and select: which radiative process must be computed, which blobs should contribute to the total SED and which structure are present. It is also necessary to specify the input file path and the directory path in which the output must be stored. `CODE NAME` automatically determines the name of the output.

Before proceeding with the computation, `CODE NAME` will print a brief summary of the starting conditions. In those cases in which some contradictory inputs are given it will ask the user whether to proceed or not.

## 1.3 `main` settings

`main` allows the user to translate FLASH output in SED graphs. The settings parameters are stored in `config.txt`. How it is structured and how to set it is described in 1.3.2. FLASH outputs structure to be given as inputs to `main` are described in 1.3.1.

### 1.3.1 Input structure

`CODE NAME` accepts as input HDF5 format files and assumes that each parameter is labeled and to be structured as:

| label | (array size) |
|---|---|
| temp | $8 * 8 * 8$ |
| dens | $8 * 8 * 8$ |
| pres | $8 * 8 * 8$ |
| ener | $8 * 8 * 8$ |
| velx | $8 * 8 * 8$ |
| vely | $8 * 8 * 8$ |
| velz | $8 * 8 * 8$ |
| block size | $1 * 3$ |
| coordinates | $1 * 3$ |

It also looks for the time between the current input simulation time and the next one in `real runtime parameters` section and it assumes to be labeled as `dt`.

The unit measure of each of these parameters need to be stated in `config` under the Unit Settings section.

### 1.3.2 `config`

`config` allows user to indicate to `CODE NAME` the location of the input file, where to store the output and how to interpret the data. An example of `config.txt` is displayed below.

```
# ***** File paths:
file_path = /home/user/path/to/file.hdf5
file_saving_path = ''
# ***** if you want to launch multiple session put id_launch = y
id_launch = n
file_list_path = ''
# ***** Indexing settings:
pres_Min = 1e-100 cm-1 g s-2
pres_Max = 1e100 cm-1 g s-2
dens_Min = 1e-100 g cm-3
dens_Max = 1e100 g cm-3
energy_Min = 1 erg
energy_Max = 1e50 erg
T_Min = 1 K
T_Max =  1e20 K
# ***** Jet settings:
# PowerLaw -> p
# # BrokenPowerLaw -> p1 p2 gamma_B
# n_e = BrokenPowerLaw
# p1 = 1.1
# p2 = 2.0
# gamma_B = 2
n_e = PowerLaw
p = 1.10
gamma_Min = 1
B = 1e-5 G
Z = 1.22
mu_0 = 1.4
E_released = 1
f_delta = 1
# ***** Processes to compute:
id_brem = n
id_syn = y
id_ssa = y
id_ssc = y
id_ec = n
# --- CMB:
id_cmb = n
# ***** Disc Settings:
id_disc = n
# M_BH = 1e45 g
a = 1
P_jet = 1e45 erg s-1
L_disc = 2e46 erg s-1
eta = 0.083
R_g_unit = y
R_in = 6
R_out = 200
# ***** Plot settings:
# nu_Min and nu_Max are the magnitude number, hence they must be
    float.
# e.g. nu_Min = 8 -> 10^8 Hz
nu_Min = 0
nu_Max = 29
ObserverDistance = 1e27 cm
Obs_theta = 0 deg
Obs_phi = 90 deg
```

```
# ***** Unit settings:
DistanceUnit = 4.3460247626638975e+24 cm
TimeUnit = 2.898016026760564e+17 s
MassUnit = 1.4644322980212338e+43 kg
GridUnit = 70 kpc
```

The file is divided in 7 sections:

- File paths;
- Indexing settings;
- Jet Settings;
- Processes settings;
- Emitting Regions settings;
- Plot settings;
- Unit settings;

Each section is explained in detail below.
`config` has the following structure:

```
    parameter_name = parameter_value
```

The parameter values are classified in four categories:

- numbers;
- units;
- 'yes/no';
- path ;

Before proceeding with the computation, the code checks if the parameter values are in the correct form. An explanation on what is the correct form is given in the following paragraphs. Lines can be commented by placing `#` at the beginning of the line.

**Units** These inputs must be composed by a value (e.g. 1, 3.14) and an `astropy.units.Quantity`. The correct syntax for the units is the following: the exponential value of a unit must be written after the unit itself, e.g.

$$\frac{g}{s\ cm^3} \rightarrow \texttt{g s-1 cm-3} \tag{1.1}$$

The user is free to express the input using different units of measure although they must still retain the same physical quantity, e.g.:

$$erg \longleftrightarrow J \longleftrightarrow \frac{kg\ m^2}{s^2} \ \texttt{or even}\ \frac{kg\ m^2\ Hz}{s^2\ Hz} \tag{1.2}$$

The scientific notation is valid, i.e. '.' to indicate decimal quantities:

$$3.14,\ 1.27,\ 5.748 \tag{1.3}$$

and the $e$ symbol to indicate the magnitude order, e.g.:

$$\texttt{1e3} \longleftrightarrow 1000,\ \texttt{1e-3} \longleftrightarrow 0.001 \tag{1.4}$$

For more information on these units, the user is referred to check the `astropy` doc. Example:

```
B = 1e-5 G
```

**Numbers** These parameters are the same of 'Unit' except that they don't come with an `astropy.units.Quantity`. Example:

```
p = 1.1
```

**'yes/no'** Some input must indicate for an affirmative or negative response. In order to indicate an affirmative response the user must insert `y` or `yes`. In order to indicate a negative response the user can reply with `n` or `no`. Please note that the code is case-insensitive. Any other answer leads the code to assume a negative response. However it induces to print a brief message notifying the user that the input is not clear. Example:

```
id_brem = y
id_syn = n
id_ssa = yep  <- would count as no
```

**path** These inputs needs to indicate the path required. The path must always be complete, i.e.: `/home/user/write/your/directory/path` The code will not assume to be based on the current directory. Example:

```
file_path = /home/user/path/to/file
```

### File Paths

As a brief recap it is composed by:

```
# ***** File paths :
  file_path = /home/user/path/to/file.hdf5
  file_saving_path = ''
# ***** if you want to launch multiple session -> id_launch = y
  id_launch = n
  file_list_path = ''
```

**file_path** If **id_launch** is set as 'no', put the file path of the simulation file to run. If it is set to 'yes', put the path to the directory containing the list of files to launch. Check the id_launch paragraph for more information.

**file_saving_path** put the directory path where the user want to save the final plot. If the user don't want to specify any path then it must be leaved as:

```
    file_saving_path = ''
```

, `CODE NAME` will automatically create a `plots/seds/` directory and will store the output file there.

**id_launch** It is a 'yes/no' input. Put 'yes' if the user wants to launch multiple instance of the code with multiple different files coming from the same directory. In this case it is necessary to write in **file_list_path** the name of a `.txt` file containing the name of all the files to run. The input **file_path** must now indicate the *directory* containing the simulation files. Finally `main.py` must be launched specifying the number of the row as argument. An **example** is depicted below:
Suppose to have the directory `runs` containing the files: `runA.hdf5`, `runB.hdf5`, `runC.hdf5`.
The eventual `'list.txt'` file would be:

```
    runA.hdf5
    runB.hdf5
    runC.hdf5
```

and the `config.txt` file would be:

```
# ***** File paths:
file_path = /home/user/path/to/runs
file_saving_path = ''
# --- if you want to launch multiple session put id_launch = y
id_launch = y
file_list_path = list.txt
```

. Let's say we want to study `runB.hdf5` then we would issue to the terminal the command:

```
> python3 main.py 2
```

This is quite useful when dealing with multiple simulation files. In this case one simply needs to launch a shell script like the pseudo-code below:

```
    n = lenght(list.txt)
    for i in range(1, n)
        > launch python3 main.py i
```

### Indexing settings

These settings allows the user to indicate which blocks must be taking into account into the SED computation. Each one of them is an 'unit' input and refers to:

- `pres` - pressure;
- `dens` - density;

- `energy` - energy;
- `T` - temperature;

They are the quantity measured in each cell of the block in the simulation file. The indexing criteria is as it follows: for each block, the main quantities are computed. For each 8x8x8 block, the quantity is calculated by averaging the value of the central 2x2x2 cells and it is then associated to the block itself. Finally, only the blocks with respective value between the `'Min'/'Max'` values are selected. E.g.:

```
temp = np.mean(data.TempSet[i, 2:4,2:4,2:4])*u.K
if temp_Min < temp < temp_Max:
    do choose block;
else
    continue;
```

### Jet settings

These inputs describe the jet emitted by the AGN. Currently there is only one distribution function (d.f.) available: `PowerLaw`. Although present in code, `BrokenPowerLaw` is not available yet. However a short description is provided below.

The normalization constant $K_e$ and the maximum energy value of the d.f. are computed by ad-hoc routines taking into account Second Order Fermi Acceleration.

- `B` - (units) - Intensity of the IGM magnetic field;
- `Z` - (numbers) - Metallicity of the blocks;
- `mu_0` - (numbers) - Average atomic mass of swept-up material in units of $m_p$;
- `E_released` - (numbers) - Apparent isotropy energy release of the explosion in in rest mass solar unit;
- `f_delta` - (numbers) - Shell scale of swept up material;

For more information, refer to Dermer & Menon (2012) §12.

**PowerLaw** In this case the `config.txt` file must state:

```
n_e = PowerLaw
p = 1.1
gamma_Min = 1
```

where:

- `n_e` - (string) - Indicates which d.f. to use. It can be 'PowerLaw' or 'BrokenPowerLaw'.

- `p` - (numbers) - Exponent of the power law distribution function (d.f.);
- `gamma_Min` - (numbers) - Lower value of the energy in the distribution function in $m_e c^2$ units;

where the d.f. is:

$$N(E) = K_e E^{-p} \text{ where gamma\_Min } < E < \text{ gamma\_Max} \tag{1.5}$$

**BrokenPowerLaw (N.B. currently not available)** In this case the `config.txt` file must state:

```
n_e = BrokenPowerLaw
p1 = 1.1
p2 = 2.0
gamma_B = 5
gamma_Min = 1
```

where:

- `n_e` - (string) - Indicates which d.f. to use. It can be 'PowerLaw' or 'BrokenPowerLaw';
- `p1` - (numbers) - Exponent of the broken power law distribution function (d.f.);
- `p2` - (numbers) - Exponent of the broken power law distribution function (d.f.);
- `gamma_B` - (numbers) - energy value of the break in the d.f.. In $m_e c^2$ units;
- `gamma_Min` - (numbers) - Lower value of the energy in the distribution function in $m_e c^2$ units;

Where the d.f. is:

$$N(E) = K_e E^{-p*} \text{ where } p* = \begin{cases} p_1 \text{ if } E < E_{\text{break}} \\ p_2 \text{ if } E > E_{\text{break}} \end{cases} \tag{1.6}$$

### Processes to compute

This section allows the user to choose which emission process to consider. Each entry is a 'yes/no' input.

- `id_brem` refers to Non-Thermal Bremmstrahlung as computed by Blumenthal & Gould (1970).
- `id_syn` refers to Synchrotron emission and is computed using `agnpy` routines.
  - `id_ssa` refers to Synchrotron-Self Absorption and is computed using `agnpy` routines.

  - id_ssc refers to Synchrotron-Self Compton and is computed using agnpy routines. **High computational times.**

- id_ec refers to External Compton and is computed using agnpy routines. **High computational times.**

**Emitting Regions**

It contains all the entries for the emitting regions (i.e. CMB, Disc, Dust Torus, BLR). For the time being only CMB and Disc are available.

**CMB** id_cmb is a 'yes/no' input. For each block the code computes the respective redshift $z$ w.r.t. to the observer and computes the block interaction with the Cosmic Microwave Background Radiation.

**Disc** id_disc is a 'yes/no' input. If left as affirmative response it is then necessary to state all the following entries of the section. Not otherwise. The user can either provide the black hole mass - commented in the initial config.txt example - via the entry:

- M_BH - (units) - Black Hole Mass

or the dimensionless spin factor and the jet kinetic power:

- a - (numbers) - is the dimensionless spin factor;
- P_jet - (units) - is the jet kinetic power;

These are used in order to compute the Black Hole Mass $M_{BH}$ via the following equation:

$$M_{BH} = \frac{P_{jet} * 1e8 * M_{\odot}}{a^2 * 2e45 \texttt{erg s}^{-1}} \texttt{[g]} \tag{1.7}$$

Source: (A.R. King, J.E. Pringle 2021 [3], DOI 10.3847/2041-8213/ac19a1).

- L_disc - (units) - Disc Luminosity
- eta - (numbers) - Accretion efficiency

**R_g_unit** is a 'yes/no' input. In case of an affirmative input, the user provides the disc radius in units of the gravitational radius $R_g$ computed as:

$$R_g = \frac{2GM_{BH}}{c^2} \tag{1.8}$$

hence R_in and R_out are numbers. Otherwise they must be speficied as units.

- R_in - (numbers/units) - Internal disc radius
- R_out - (numbers/units) - External disc radius

### Plot settings

These entries define the plotting settings regarding the range of frequencies over which the SED must be computed and the observer parameters:

- `nu_Min` - (numbers) - indicates the magnitude number of the minimum value of the frequency range interval;
- `nu_Max` - (numbers) - indicates the magnitude number of the maximum, value of the frequency range interval;

The frequency array is defined as:

```
nu = np.logspace(nu_Min, nu_Max) * u.Hz
```

therefore both need to be numbers and to indicate the magnitude number. E.g.

$$\texttt{nu\_Min = 8} \rightarrow \nu_{min} = 10^8 \ Hz \tag{1.9}$$

- `ObserverDistance` - (units) - Distance of the observer from the source;
- `Obs_theta` - (units) - Azimuthal angle of the observer w.r.t. the source;
- `Obs_phi` - (units) - Polar angle of the observer w.r.t. the source;

Figure 1.1 shows the correct coordinates.

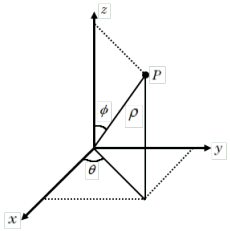


Figure 1.1: $\theta$ is the azimuthal angle, while $\phi$ is the polar one.

### Unit settings

Here the user indicates how the block attributes have been measured and the dimension of the Grid in the original simulation file.

- `DistanceUnit` - (units) - Distance unit used in the original simulation.
- `TimeUnit` - (units) - Time unit used in the original simulation.
- `MassUnit` - (units) - Mass unit used in the original simulation.
- `GridUnit` - (units) - Grid unit dimension in the original simulation grid.

It is customary to provide these units in function of the Hubble constant $H_0$. However, the code is not able to determine it and thus it would return as an error. Please take into consideration to provide the correct value.

### Summary

Before going on, the code will print out a short summary of the starting conditions showing the input file name, the emissions to be computed, the name of the output file and where it will be stored. An example is shown below:

```
    Initialization complete. Here a short summary:
------- Summary -------
File name:
-  run1.hdf5
Emission processes:
- Bremmstrahlung
- Synchrotron
- - self-absorption
- - self-Compton
- External Compton
- - CMB
- - Disc
The selected power law is: PowerLaw
The resulting plot will be called: run1_B_S_a_c_EC_cmb_disc.png
It will be saved in: /home/marco/coding/python/last_vers/plots
```

The output name is defined by the processes used to compute it. The structure is `name` + :

- `B` if brehmstrallung is to be considered.
- `S` if synchrotron emission is to be considered.
- `-a` if synchrotron self-absorption is to be considered.
- `-c` if synchrotron self-Compton is to be considered.
- `EC` if External Compton is to be considered.
- `-CMB` if CMB is to be considered.
- `- Disc` if the disc is to be considered.

Finally, if the user should indicate the presence of one the targets emitting sources which photons would be subject to External Compton without but the latter is set as off, the code warns the user and ask whether to continue or not:

```
    Attention, External Compton is set as off but the following
    processes are still set to be computed:
- CMB
- Disc
This may result in long computational times without any contribute
    to the final spectrum. Do you still want to proceed? (y/yes)
```

Please bear in mind that the processes of SSC and EC are time consuming.

## 1.4 `plotting` settings

`plotting` is used to produce the histograms and contour plots of the blobs physical quantities i.e. temperature, energy, density and pressure. It uses the same input as `main`. In order to properly set and decide which plot to produce, the user can modify `config\_plot.txt`. Before proceeding with the plots, `plotting` will print a brief summary of the starting conditions.

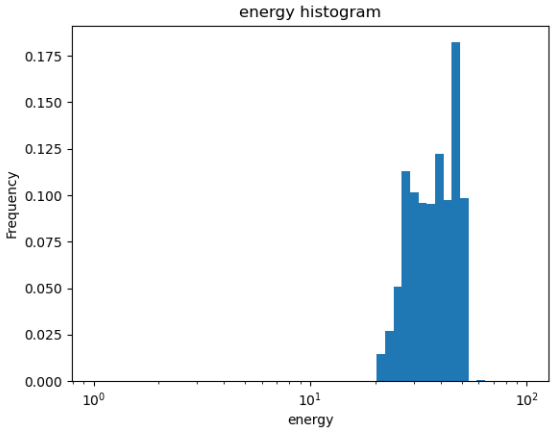


Figure 1.2: Example of histogram plot

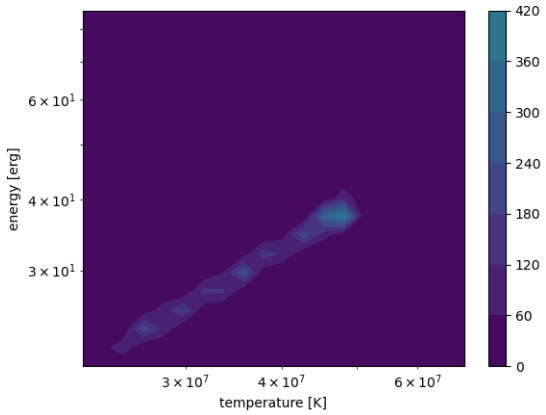


Figure 1.3: Example of contour plot

### 1.4.1 config_plot

`config_plot` allows user to specify the location of the input file, where to store the output, which plot to produce and which block to consider. An example is displayed below. It can be classified in file path info, graphs settings, interval settings and unit settings. The latter two works in a similar way to the one described in `config`. See 1.3.2 and 1.3.2.

```
#
file_path = /home/user/path/to/file.hdf5
file_saving_path = ''
# If the user wants to have the arrays printed:
display_arrays = yes
# AVAILABLE LABEL:
# -- 'density'
# -- 'pressure'
# -- 'energy'
# -- 'temperature
kind = histogram
#  AVAILABLE LABEL:
# -- 'histogram'
#       if 'kind' = histogram then:
#           bin_rule = (one between sqrt, rice or sturges)
#           label1 = parameter1
#           label2 = parameter2 etc.
#       example: label1 = density
# -- 'contour'
#       if kind = counter then the following must be stated:
    x_label and y_label must be stated.
#           - x_label (which quantity must be on x-axis)
#           - y_label (which quantity must be on y-axis)
#           - n (number of bins to consider)
#           - clim (max value to display on the color bar)
#       n number of bins must be stated aswell.
```

```
#       example: x_label = density
#       example: y_label = pressure
#       example:       n = 100
#
label1 = temperature
label2 = energy
bin_rule = sqrt
x_label = temperature
y_label = energy
n = 20
clim = 1000
# ****** Interval settings:
pres_Min = 1e-20 cm-1 g s-2
pres_Max = 1e-5 cm-1 g s-2
dens_Min = 1e-30 g cm-3
dens_Max = 2e-20 g cm-3
energy_Min = 1 erg
energy_Max = 1e2 erg
T_Min = 1e1 K
T_Max = 1e20 K
# ***** Unit settings:
DistanceUnit = 4.3460247626638975e+24 cm
TimeUnit = 2.898016026760564e+17 s
MassUnit = 1.4644322980212338e+43 kg
GridUnit = 70 kpc
```

### File path info

Here the user must specify the location of the input file and the directory in where to save it. Similar to `main`, if the user wish not to specify any path where to save the output file, the code will automatically create a directory in `/plots/graphs`.

### Graph settings

`plotting` comes up with the possibility to plot histograms or contour plot. This can be chosen by changing the parameter value `kind`. Each cases come with its own parameters to use. They are described below. The available label for the physical quantities are:

```
    'density'
    'pressure'
    'energy'
    'temperature'
```

There is no need to comment with # the unused parameters as they will be automatically ignored by the code.

With `display_arrays` the user can choose if print the name of the array printed or not on the graph.

**histogram** Example of histogram parameters:

```
kind = histogram
label1 = temperature
label2 = energy
bin_rule = sqrt
```

The user can choose how many label to define. In any case the rule to follow is 'label' + 'number' in increasing order. This will result in more graph to be produced. In the example above the code would produce two histograms. Each histogram is labeled accordingly.

Regarding the `bin_rule`, there are three currently laws available: `sqrt, rice, sturges`. Each of them specify the number of bin $n_{\text{bin}}$ to consider based on the number of the blocks $n_{\text{blocks}}$ rounded to the nearest integer.

- `sqrt`

$$n_{\text{bin}} = \texttt{round}(n_{\text{blocks}}^{1/2}) \tag{1.10}$$

- `rice`

$$n_{\text{bin}} = 2 * \texttt{round}(n_{\text{blocks}}^{1/3}) \tag{1.11}$$

- `sturges`

$$n_{\text{bin}} = \texttt{round}(\log_2(n_{\text{block}}) + 1) \tag{1.12}$$

**counter** Example of counter parameters:

```
kind = counter
x_label = density
y_label = pressure
n = 100
cmap = 1000
```

In this case it is necessary to specify the x and y labels, the number of bins `n` and the max value of the color map `cmap` to display on the colour bar.

### Summary

Before starting to compute the graphs, the code will print a brief summary showing which plot it's going to produce, from which file, where it will be stored and which name it will have.

The name etiquette is as it follows: `name_ + kind_ + (x_) label_ ( + y_label) +.png` Example:

```
file_contour_density_pressure.png
file_histogram_temperature.png
file_histogram_energy.png
```

# Chapter 2

# How it Works

The main `CODE NAME` idea is to build a `Block` class object for each block that compose the original input file. A list of these blocks is then feed to the various functions which will either remove some blocks or updates them.

Each block stores stores the information of the physical parameters of the original block and info on which emission process takes part into.

## 2.1 Data Management

Each process revolve around the concept of `Block` class. Each `Block` stores the information needed and which emission process should be computed with its values.

### 2.1.1 Data Read

`CODE NAME` assumes each input file to have the `HDF5` format and the data to be labeled and structured as such:

| label | (array size) |
|---|---|
| temp | $8 * 8 * 8$ |
| dens | $8 * 8 * 8$ |
| pres | $8 * 8 * 8$ |
| ener | $8 * 8 * 8$ |
| velx | $8 * 8 * 8$ |
| vely | $8 * 8 * 8$ |
| velz | $8 * 8 * 8$ |
| block size | $1 * 3$ |
| coordinates | $1 * 3$ |

It also looks for the time between the current input simulation time and the next one in `real runtime parameters` section and it assumes to be labeled as

`dt`.

### 2.1.2 Data reduction

FLASH computes data in a grid where each block is a $8^3$ structure. Since this is not feasible to handle in a long computation, `CODE NAME` computes the mean of the central $2^3$ cube. Hence, each `Block` is defined with a single quantity for attribute.

### 2.1.3 `Block` Definition

With the concept of `Block` we mean the fundamental unit which stores all the information and parameters that compose the jet. While Flash divides the space in a grid composed by $8^3$ blocks, `Block` is a reduced structure.

Each `Block` is composed by the following quantities.

**Note:** The quantities followed by an * are those which are stored dimensionless although they are computed with a certain units of measure.
Coordinate quantities:

- $x$* $(cm)$
- $y$* $(cm)$
- $z$* $(cm)$
- center* $(x, y, z)$

Average of physical quantities:

- temp $T$ $(K)$
- dens $\rho$ $(g\ cm^{-3})$
- pres $P$ $(g\ cm^{-1}\ s^{-2})$
- energy $E$ $(erg)$
- velx $v_i$ $km\ s^{-1}$
- vely $v_y$ $km\ s^{-1}$
- velz $v_z$ $km\ s^{-1}$

Computed quantities (more info on how they are computed in the following sections):

- radius $r$ $(cm)$
- distance (to the source) $d$ $(cm)$
- obs_distance (distance to the observer)

- redshift
- n_0 (number density) $n_0$ $(cm^{-3})$
- opacity $k$

### Radius

It is used to define the blob. We assume the blob to be a round sphere with radius equal to half of the block size. Hence:

$$r = 0.5 \ \texttt{(block size)}_x \ [\text{cm}] \tag{2.1}$$

### Distance to the source

We assume the source to be at $O \equiv (0,\ 0,\ 0)$ hence:

$$d = \sqrt{x^2 + y^2 + z^2} \tag{2.2}$$

### Distance to the observer

Supposing the observer to be in $P \equiv (P_x, P_y, P_z)$ then:

$$\text{obs}_{\text{distance}} = \sqrt{(x - P_x)^2 + (y - P_y)^2 + (z - P_z)^2} \ [\text{cm}] \tag{2.3}$$

### Redshift

Computes the redshift $z$ associated to the distance of the block from the observer. It uses the `astropy.Coordinates.Distance` module.

### Number Density $n_0$

Number density $n_0$ is used to compute opacity. Is defined as:

$$n_0 = \frac{\rho}{\mu_0 m_p} \ [\text{cm}^{-3}] \tag{2.4}$$

where $\rho$ is the density in the block, $m_p$ is the mass of the proton and $\mu_0$ is the average atomic mass of swept-up material in units of $m_p$.

### Opacity

See 2.4.1.

## 2.2 Inverse Compton Selection Criteria

In the Inverse Compton Scattering process, only the blobs which resides in the external layer take effectively part into the process. Although the inner layers may take part into the scattering process, the free mean path of the photons is quite diminished and it is more likely to interact with the blobs and to disperse their energy via different processes.

Although computing the Inverse Compton for each blob and then take into account the interaction of the photons with the other blobs is a legitimate idea, it soon become not much viable since the Inverse Compton has a computational overload not negligible.

Hence, it has been decided to consider only the blobs which compose the external layer. This is effectively a *convex hull* problem.

### 2.2.1 Convex Hull Problem

Suppose to have a set $X$ of $n$ points scattered in an Euclidean space. The convex hull of $X$ is the minimal convex set containing $X$.

The code uses the routine `ConvexHull` from `scipy`, which uses the program `QHull` [1].

## 2.3 Raytracing

In certain cases, it is necessary to quantify the photon flux from a point A to a point B. For example, when computing the Inverse Compton of the photons produced by the disk scattered by a certain blob $i$, we would like to quantify how many photons effectively reach the blob and how many 'get lost' interacting with the blobs that lie on the ray path. Allegedly, a blob which resides in the inner regions of the jet would experience a dimmer flux of the photons produced by the disk. Similarly, the photon flux produced by a blob may interact with the blobs that lie on the line of view. Hence it is necessary to take into account that.

### 2.3.1 Generalization of the problem

Suppose to have $n$ round spheres scattered in an empty space. Given an arbitrary point $P$, from each sphere's center starts a segment having as extremes respectively the center of the sphere and the point $P$. For each segment it is necessary to assess if it intersects with other spheres and if so, compute the fraction of the segment intersected with the sphere.

### 2.3.2 Algorithm

The idea as it follows:

1. Compute the line $l$ passing through two points ($P$ and the center $C_i$ of the $i$-sphere).
2. Compute the plane $\hat{A}_p$ perpendicular to the line $l$ and passing through $P$.
3. Compute the plane $\hat{A}_{C_i}$ perpendicular to the line $l$ and passing through $C_i$
4. Consider only the spheres lying between the two planes;
5. For each sphere $j \neq i$ that lies between the two planes, compute the distance between $C_j$ and $l$
6. If the distance $d(C_j, i) \leq r_j$ where $r_j$ is the radius of the sphere $j$, then $l$ intersects with the sphere $j$

### 2.3.3 Mathematical solution

Here is shown the mathematical approach to the algorithm. For formatting purposes, $C_i$ is indicated with $C$, all other block centers $C_j$ with $j \neq i$ are indicated with $K$:

1. The direction vector $\vec{l}$ of the line going from $P$ to $C$ is:
$$\vec{l} = (C_x - P_x, C_y - P_y, C_z - P_z) \tag{2.5}$$
2. Given the plane $\hat{A_C}$ passing through $C$ and $\perp \vec{l}$ we want to see in which side of the space $P$ resides. This translates to compute the distance point-plane $d(P, \hat{A})$. Since we are not interested in the whole number but only on the sign of the distance, we do the following:
$$\vec{r} = (C_x - P_x, C_y - P_y, C_z - P_z) \tag{2.6}$$
The distance should be computed as:
$$d_1 = d(P, \hat{A}_C) = \frac{\vec{l} \cdot \vec{r}}{\left\|\vec{l}\right\|} \tag{2.7}$$
but since we need only the sign of $d_1$ then:
$$d_C^{\text{sign}}(P) = \frac{\vec{l} \cdot \vec{r}}{d_1} \tag{2.8}$$
i.e. the sign of the distance of P with respect to the plane which passes through C.
3. We iterate 2. for each other block $K$. This is done both for the plane passing through C $\hat{A}_C$ and through $\hat{A}_P$. Therefore, we compute $d_C^{\text{sign}}(K)$ and $d_P^{\text{sign}}(K)$.

4. We consider only those points $K$ positioned between the two planes. This means we consider the set $X$:

$$X = \{ \forall\ K : d_C^{\text{sign}}(K) = d_C^{\text{sign}}(P) \wedge d_P^{\text{sign}}(K) = -d_C^{\text{sign}}(P)\} \tag{2.9}$$

5. For each point in the set $X$ we consider the distance point-line $d(K, l)$ (line as defined by passing through two points). This is done in several step. First we compute the direction vector $\vec{D}$ of the line same as (2.5):

$$\vec{D} = C - P \tag{2.10}$$

Then the vector from $C$ to $K$, same as above:

$$\vec{E} = C - K \tag{2.11}$$

We then compute the projection of $\vec{E}$ on $\vec{D}$:

$$\vec{proj} = \frac{\vec{E} \cdot \vec{D}}{\vec{D} \cdot \vec{D}} D \tag{2.12}$$

Finally, it returns the distance between $\vec{E}$ and $\vec{proj}$:

$$d(K, l) = \sqrt{(E_x - proj_x)^2 + (E_y - proj_y)^2 + (E_z - proj_z)^2} \tag{2.13}$$

### 2.3.4 Code solution

The code can be divided in two structure: `Python` and in `C` . `Python` feeds the input to the `C` code and handles the output. `C` do the effective computation. This solution effectively reduces the computational times needed. The various inputs and outputs are in `txt` format and are stored in `/lib/tmp_files`.

After the computation, `Python` updates each block with the respective distance computed, i.e. the source of the Jet or the observer.

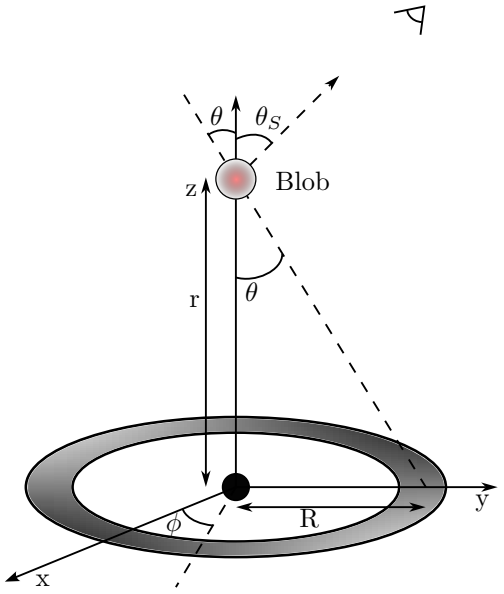


Figure 2.1: Geometry used for the energy density and external Compton scattering calculations in `agnpy`

## 2.4 Radiative Process

In the following paragraphs we will follow `agnpy` structure. Hence we consider the blob moving along the $z$ axis and the main angles $\theta$ and $\phi$ defined as in 2.1.

### 2.4.1 Opacity

In order to manage the photons interaction with the respective blocks, it is necessary to take into account the opacity.

In 2.4.1 we address the physical description of the process and the main computation. In 2.4.2 we describe the respective algorithm.

#### Opacity and cross section

Every medium encountered along the path of a photon flux interacts with the photons, leading to a reduction in the overall flux. This decrease in flux is determined by both the opacity of the medium and the cross section of interaction between the photons and the constituent particles of the medium. The opacity can be expressed as follows:

$$k_\nu = \frac{\sigma_c n_0}{\rho} \tag{2.14}$$


Figure 2.2: An "head-on" and "tail-on" collision.

where $\rho$ and $n_0$ are, respectively, the density and the number density of the gas. Finke (2016) [3] provides a way to compute the cross section: in the "head-on" approximation (see figure 2.2), the photons are scattered in the same direction as the incident electrons, thus:

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\epsilon_s \mathrm{d}\Omega_s} \approx \frac{\mathrm{d}\sigma}{\mathrm{d}\epsilon_s}\delta(\Omega_s - \Omega_e) \tag{2.15}$$

where $\epsilon_S$ is the scattered photon energy, $\Omega_s$ it is the direction that the delta ensured to be equal to the electron direction $\Omega_e$.

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\epsilon_s} = \int \mathrm{d}\Omega_s \frac{\mathrm{d}\sigma}{\mathrm{d}\epsilon_s \mathrm{d}\Omega_s} = \frac{3\sigma_T}{8\gamma\bar{\epsilon}}\Xi(\epsilon, y) H\left(\epsilon_s; \frac{\bar{\epsilon}}{2\gamma}, \frac{2\gamma\bar{\epsilon}}{1+2\bar{\epsilon}}\right) \tag{2.16}$$

with

$$\Xi(\bar{\epsilon}, y) = 1 - y + \frac{1}{1-y} - \frac{2y}{\bar{\epsilon}(1-y)} + \left[\frac{y}{\bar{\epsilon}(1-y)}\right]^2 \tag{2.17}$$

and

$$\bar{\epsilon} = \gamma\epsilon(1 - \cos\psi) \tag{2.18}$$

$$y = \frac{\epsilon_s}{\gamma} \tag{2.19}$$

Let $\phi$ and $\theta = \arccos(\mu)$ be, respectively, the azimuthal and polar angles of the incident photon (figure 2.1) and $\phi_s$, $\theta_s = \arccos(\mu_s)$ the angles of the scattered photons, then:

$$\cos\psi = \mu\mu_s + \sqrt{1-\mu^2}\sqrt{1-\mu_s^2}\cos(\phi - \phi_s) \tag{2.20}$$

The total Compton-scattering cross-section is:

$$\sigma(\bar{\epsilon}) = \int \mathrm{d}\epsilon_s \frac{\mathrm{d}\sigma}{\mathrm{d}\epsilon_s} = \sigma_T S_0(\bar{\epsilon}) \tag{2.21}$$

where:

$$S_0(x) = \frac{3}{8x^2}\left[4 + \frac{2x^2(1+x}{(1+2x)^2} + \frac{x^2 - 2x - x}{x}\ln(1+2x)\right] \tag{2.22}$$

with $x$ being the energy of the photon in $m_e c^2$ units.

### 2.4.2 Opacity algorithm

For each block, the code computes respectively the cross-section $\sigma$ and the opacity $k_\nu$ using respectively (2.16) and (2.14).

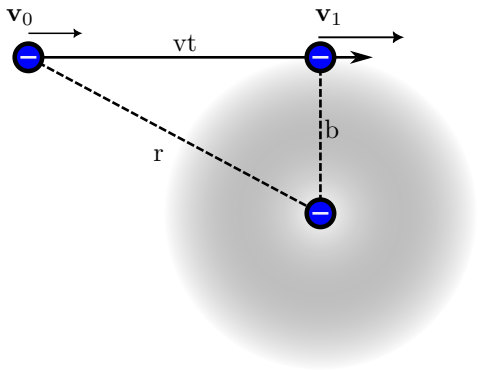


Figure 2.3: Example of moving charged particle interacting with the electromagnetic field produced by another charge. In this case the interaction results in an acceleration.

### 2.4.3 Bremsstrahlung

When a moving charged particle interacts with the potential field of the nearby particles, it gets accelerated resulting in emissions of photons. This process is called Bremsstrahlung or free-free emission and it is responsible of the emission in the X-ray.

In this section it is described how to compute the bremsstrahlung emission produced by a non-thermal population. Source [2].

#### Spectrum from a non-thermal distribution

In this case the emission comes from a non-thermal distribution of electrons the total bremsstrahlung spectrum is found from:

$$\frac{\mathrm{d}N_{\mathrm{tot}}}{\mathrm{d}t\,\mathrm{d}k} = \int \mathrm{d}E_i\, N_e(E_i)\left(\frac{\mathrm{d}N_i}{\mathrm{d}t\,\mathrm{d}k}\right) \tag{2.23}$$

where:

$$N_e(E_i) = K_e E_i^{-p} H(E_i; E_0, E_m) \tag{2.24}$$

with $H(E_i; E_0, E_m)$ is the Heaviside function and $K_e$ is the normalization constant:

$$K_e = \frac{\rho}{\left(\int_{E_0}^{E_m} N(E_i)\,\mathrm{d}E_i\right)\mu m_H} \tag{2.25}$$

with:

$$\int_{E_0}^{E_m} N(E_i)\,\mathrm{d}E_i = \frac{E_m^{-p+1} - E_0^{-p+1}}{-p+1} \tag{2.26}$$

Supposing that $k \ll E_m$ in order to simplify the computation, the lower limit is:

$$E_L = \max(k, E_0) \tag{2.27}$$

Using (**??**) and (**??**) one obtains:

$$\frac{\mathrm{d}N_{\mathrm{tot}}}{\mathrm{d}t\,\mathrm{d}k} = \alpha r_0 K_e k^{-1} \sum_s n_s \times \int_{E_L} \mathrm{d}E_i\, E_i^{-(p+2)} \left[ \left(2E_i^2 - 2E_i k + k^2\right)\phi_{1(s)} - \frac{2}{3} E_i (E_i - k)\phi_{2(s)} \right] \tag{2.28}$$

Referring to the whole integral as simply $I_s(k, E_L; p)$, in case of weak shielding, complete ionization, $\phi_1$ and $\phi_2$ are well represented by the value:

$$\phi_{\mathrm{weak}} = 4\left(Z^2 + Z_{\mathrm{el}}\right)\left\{ \ln\left[2E_i\left(\frac{E_i - k}{k}\right)\right] - \frac{1}{2} \right\} \tag{2.29}$$

thus reducing $I_s$ to:

$$I_s(\text{weak shielding}) \approx I' \bar{\phi}_{\mathrm{weak}} \tag{2.30}$$

where

$$\begin{aligned} I' &= \int_{E_L} \mathrm{d}E_i\, E_i^{-(p+2)} \left(\frac{4}{3}E_i^2 - \frac{4}{3}E_i k + k^2\right) \\ &= \frac{4}{3}\frac{E_L^{-(p-1)}}{p-1} - \frac{4}{3}\frac{k E_L^{-p}}{p} + \frac{k^2 E_L^{-(p+1)}}{p+1} \end{aligned} \tag{2.31}$$

For $E_L = E_0$ one can set $E_i = E_0$, on the other hand, if $E_L = E_0$ or $k \approx E_0$ one can set $E_i \approx 2k$ in (2.29), thus obtaining:

$$\bar{\phi}_{\mathrm{weak}} = \begin{cases} 4(Z^2 + Z_{el})\{\ln\left[2E_0(E_0 - k)/k\right] - 1/2\} \text{ for } k < E_0 \\ 4(Z^2 + Z_{el})[\ln(4k) - 1/2] \text{ for } k > E_0 \end{cases} \tag{2.32}$$

**Bremsstrahlung algorithm**

WIP

# Bibliography


[1] C. Bradford Barber and Hannu Huhdanpaa. *qhull*. URL: `http://www.qhull.org/`. (accessed: 11.03.24).

[2] George R Blumenthal and Robert J Gould. "Bremsstrahlung, synchrotron radiation, and compton scattering of high-energy electrons traversing dilute gases". In: *Reviews of modern Physics* 42.2 (1970), p. 237.

[3] Justin D Finke. "External compton scattering in blazar jets and the location of the gamma-ray emitting region". In: *The Astrophysical Journal* 830.2 (2016), p. 94.