DAO: User Guide

Yimin Huang [1]

[1]Center for Astronomy and Astrophysics, , Center for Field Theory and Particle Physics, , Department of Physics, Fudan University, Shanghai 200438, China , huangym23@m.fudan.edu.cn

October 11, 2025

Contents

1 Introduction **2**

2 Installation **3**
2.1 Requirements 4
2.2 Steps 4

3 Usage **5**
3.1 Model Parameters 5
3.1.1 Summary 5
3.1.2 Description 6
3.2 Model Output 11
3.2.1 op_spec 11
3.2.2 op_inte 12
3.2.3 op_temp 12
3.2.4 op_abund 12
3.2.5 op_log 13
3.2.6 op_emis 13
3.3 Running the Code with Python 13
3.3.1 Core Components 13
3.3.2 Basic Workflow 14
3.3.3 Key Features and Concepts 14

4 Examples **16**
4.1 Effect of ionization parameters 16
4.2 Emergent intensity 19
4.3 Ions Fractions 21
4.4 Heating-Cooling rate 22

5 Physics and math behind DAO **25**
5.1 Radiative transfer 25
5.2 Compton Scattering 28

6 Vesion **31**
6.1 v1.0 31
6.2 v1.1 31

7 Appendix **32**

Chapter 1

Introduction

`DAO` is a new public X-ray reflection model. It calculates the reprocessed radiation from disk–corona systems, similar to well-known reflection models such as `reflionx` (Ross & Fabian 2005) and `xillver` (García & Kallman 2010; García et al. 2013, 2014). The main purpose of `DAO` is to provide an accurate, flexible, and community-accessible reflection code that allows researchers to carry out studies according to their specific needs. In this document, I'll try to include all of this model (e.g., usage, approximation and physics) and to update this USERS GUIDE. please contact me if you have any problems.

We recommend the following steps to get started with `DAO`:

1. First, please read Chapter 3.1 (*Model Parameters*) and Chapter 3.2 (*Model Outputs*) to gain a comprehensive understanding of the code's capabilities and the physical parameters it uses.

2. Next, proceed to Chapter 3.3 (*Running the Code with Python*), which explains how to use the provided Python interface to set up and run simulations.

3. Finally, we encourage you to explore the practical examples provided in Chapter 4 (*Examples*). These demonstrate how to configure the model for common scientific use cases.

Chapter 2

Installation

`DAO` utilizes `XSTAR` to perform atomic process calculations, which requires that the `HEASoft` package is installed on your system. Please follow the instructions in the **Install** section to install it.

The `make` utility is required for the installation. If it is not already on your system, you can install it using the following command. For Debian-based systems (e.g., Ubuntu), run:

```
sudo apt-get install make
```

Atom data can directly influence the reflection spectrum (Ding et al. 2024). To allow for the flexible use of different atomic data table versions, `DAO` does not include the atomic data table or the Compton heating-cooling file from the `HEASoft` package. You are required to download your preferred versions from the `XSTAR Subroutine` repository and store them in a directory named `data`.

For example, to download the necessary files, execute the following commands:

```
mkdir data
cd data
wget ..../atdb.fits
wget ..../coheat.dat
```

Note: The folder name must be `data`. Do not change it unless you have also modified the corresponding path in the source code.

To accelerate modeling and conserve computational resources, this code utilizes pre-calculated Compton scattering data tables. We provide pre-calculated data for both a Gaussian-approximated and an exact redistribution function. Each table is structured with 70 temperature grids and a specific number of energy bins. It is critical that the number of energy bins in the data table matches the number of energy bins you have set for your model.

For convenience, we provide redistribution tables with two different energy resolutions:

- `kernel_gaus` and `kernel_exact`: 1000 energy bins
- `kernel_gaus5000` and `kernel_exact5000`: 5000 energy bins

If you require a different number of energy bins, you can generate your own redistribution tables using the routines located in the `get_redistribution` directory. The calculation routine for the exact redistribution function use the the code of García et al. (2020).

2.1 Requirements

- `make`
- `Python`: Version 3.10 or newer.
- Compilers: `gfortran`, `gcc`.
- `Python` Libraries: `f90nml`, `jinja2`, `hashlib`, `os`, `argparse`, `subprocess`, and `pathlib`
  `openmp`.

2.2 Steps

1. Download the source code from GitHub.
2. create mods file.

   ```
   mkdir mods
   ```

3. Change `HEASoft` path in `HEADS` in `Makefile`
   e.g., heasoft/heasoft-6.31.1/x86_64-pclinux-gnu-libc2.17
4. Compile model

   ```
   make
   ```

5. run model

   ```
   ./dao.x input
   ```

Chapter 3

Usage

3.1 Model Parameters

DAO need a input file to set physical parameters,output path,etc. A template has been exist in templates/inputṅmlj2

3.1.1 Summary

Table 3.1: Model parameters of DAO

Parameter	Type	Default	Description
model	int	0	Model selector [0: reflection, 1: pure scattering].
debug	int	2	Debug switch.
nmaxmain	int	15	Maximum number of main iteration steps.
nmaxrte	int	100	Maximum number of radiative transfer iteration steps.
ndrt	int	200	Number of optical depth grids.
nfrt	int	5000	Number of energy bins.
nmurt	int	10	Number of angle bins.
ity	int	2	Defines the type of angle bins and the angular integration method.
it-temp	int	99	Maximum iteration steps for thermal equilibrium calculation.
temp-gas	float	10^8	Initial gas temperature.
temp-gas-unit	str	K	Unit of the initial gas temperature (e.g., K).
nh	float	10^{15}	Hydrogen density in cm^{-3}.
inci-type	str	powerlaw	Type of corona illumination spectrum.
gamma	float	2.0	Photon index (Γ) for a power-law, or the Blackbody temperature in eV.

Continued on next page

Table 3.1 – continued from previous page

Parameter	Type	Default	Description
`hcut`	float	10^5	High-energy cut-off for the illumination in units of eV (e.g., 10^5 eV = 100 keV).
`inmu`	float	0.7	Cosine of the incidence angle.
`sbot`	int	0	Switch for bottom illumination (thermal disk radiation) [0: off, 1: on].
`ktbb`	float	350	Temperature of the disk blackbody radiation in eV.
`fx-frac`	float	10^{-10}	Ratio of the top illumination flux to the total flux.
`zeta`	float	3.0	Logarithm of the ionization parameter ($\log_{10}\xi$).
`Abundances`	float	–	Elemental abundances relative to Hydrogen for elements from H to Zn.
`inci-file`	str	–	The file name of the incident spectrum. This is required if `inci-type` is set to "file".
`kernelpath`	str	kernel_exact5000	Path to the Compton scattering redistribution function file. The number of energy bins must match the `nfrt` parameter.
`dataenv`	str	data	Path to the directory containing the atomic database and Compton heating-cooling files.
`output`	str	–	Path and name for the output file.

3.1.2 Description

`Model,Debug`

`DAO` support the pure scattering model with constant gas temperature along the vertical depth when `model` set at 1, and normal reflection model when `model` set at 0.

The simplt result of constant temperature - pure scattering model and reflection model as shown in Figure 3.1 and 3.2

Debug mode is opened when `debug` switch set at 1. Energy, angle and depth grids will be printed to a file when it's open.

`nmaxmain,nmaxrte`

In general, radiative transfer solution will get convergence after 100 iterations. The step of iterations depends on opacity $\chi(E)$ of plasma, i.e., when absorption opacity is large than scattering opacity (i.e., ionization of plasma is low enough and gas temperature is lower then 10^6 K), model will converge more faster. In contrast, Compton scattering is dominant process when gas temperature is high enough or ionization is high, while both opacity and the emissivity could be negligible in the condition of radiative equilibrium (see

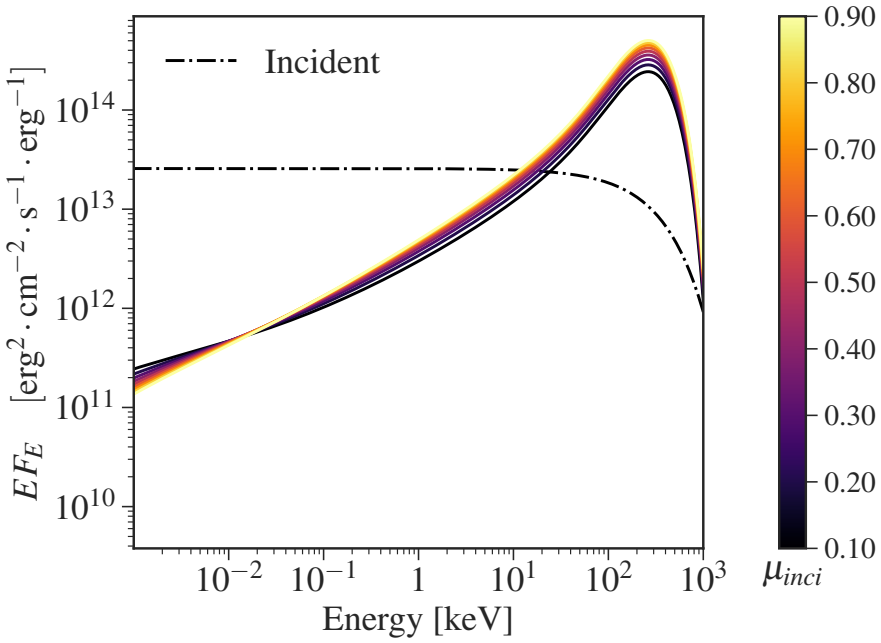

Figure 3.1: Pure scattering model for different incidence angle

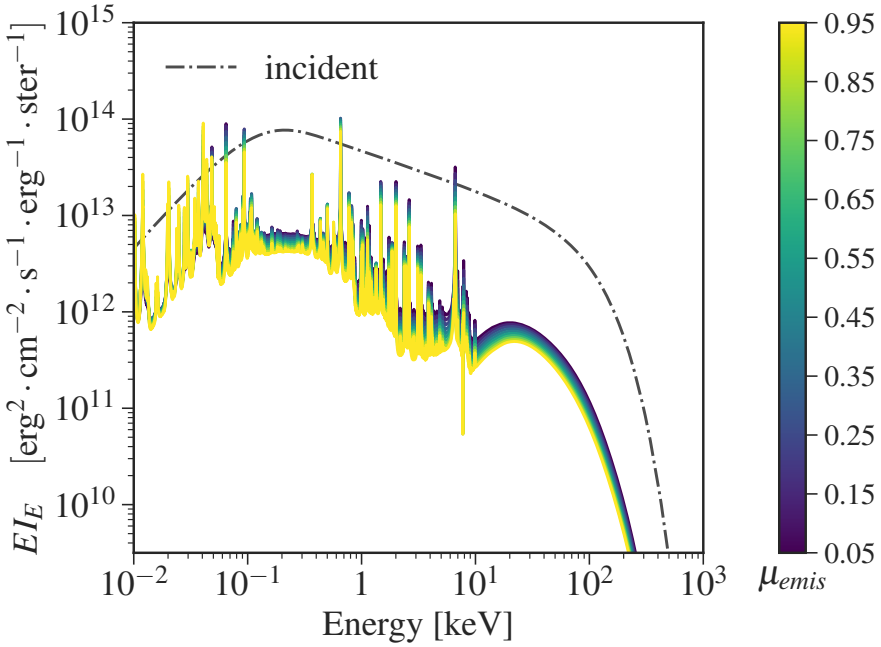

Figure 3.2: Reflection model for different emission angle

??, García & Kallman (2010)). Consequently, Lambda-iteration need to be performed more steps to get convergence.

`nmaxrte` controls the maximum iteration steps for radiative transfer. Considering the calculation time and convergence, we set 100 as default steps.

`nmaxmain` controls the maximum iteration steps between radiative transfer and XS-

TAR (see **??** for more details). We set 15 as default value.

ndrt,nfrt,nmurt

These parameters define the resolution of depth, energy, and angle grids. We don't recommend user reduce the resolution of depth and energies. The photons are more easier to be scattered in its own energy and the line emission haven't enough resolution when `nfrt` is not enough.

ity

`DAO` provides two methods for defining the angular grids and performing the corresponding angular integration. The desired method can be selected with the `ity` parameter.

1. **Gauss-Legendre Quadrature**
   This method uses angular grids and weights determined by the Gauss-Legendre quadrature rule. The mean intensity J_ν is calculated as a weighted sum:

$$J_\nu = \sum_{m=1}^{\texttt{nmurt}} I_\nu(\mu_m) w_m \tag{3.1}$$

   where μ_m represents the Gauss-Legendre points (nodes) and w_m are the corresponding Gauss-Legendre weights.

2. **Linear Grid with Trapezoidal Rule**
   This method uses a set of linearly spaced angular grids. The grid points are defined as:

$$\mu_m = \frac{m - 0.5}{\texttt{nmurt}} \quad (m = 1, 2, \ldots, \texttt{nmurt}) \tag{3.2}$$

   The integration is then performed using the trapezoidal rule to calculate the mean intensity:

$$J_\nu = \frac{1}{2} \sum_{m=2}^{\texttt{nmurt}} \left(I_\nu(\mu_m) + I_\nu(\mu_{m-1})\right) (\mu_m - \mu_{m-1}) \tag{3.3}$$

it-temp,temp-gas,temp-gas-unit

`XSTAR` need number of iterations to get thermal equilibrium, `it-temp` is same with `nlimd` in XSTAR. And `temp-gas`, `temp-gas-unit` define the initial gas temperature and its unit. In generl the latter is doesn't matter for final result.

nh

Hydrogen density in cm^{-3}. The `DAO` version 1.0 only support for density less than 10^{18} cm^{-3}. Specify atom data table will be needed when density large than that value Kallman et al. (2021); Mendoza et al. (2021).We will update as soon as possible when we get the atom data table.

inci-type,gamma,hcut,inmu,sbot,ktbb,fx-frac

The following parameters are used to define the properties of the incident radiation field.

1. `inci-type`: Defines the spectral shape of the incident illumination. Built-in options include `powerlaw`, `cutoff-powlaw`, and `blackbody`. To use a custom spectrum, set this parameter to `file` and provide the path in the `inci-file` parameter.

2. `gamma`: This parameter's function depends on the `inci-type`:

   - For `powerlaw` or `cutoff-powlaw`, it specifies the photon index (Γ).
   - For `blackbody`, it specifies the temperature in units of eV.
   - It is not used when `inci-type` is `file`.

3. `hcut`: Defines the high-energy cutoff in units of eV. This parameter is only used when `inci-type` is set to `cutoff-powlaw`.

4. `inmu`: Specifies the cosine of the incidence angle for the top illumination. A value greater than 1.0 is interpreted as an isotropic illumination field.

5. `sbot` and `ktbb`: These parameters control the bottom illumination, which should be included in scenarios where the disk itself is a significant source of radiation (e.g., in a thermally-dominated soft state).

   - Setting `sbot` to `1` enables the bottom illumination component.
   - The bottom illumination is modeled as an isotropic blackbody spectrum, and `ktbb` sets its temperature in units of eV.

6. `fx-frac`: This parameter is used when bottom illumination is active (`sbot = 1`). It sets the fraction of the total flux that originates from the top illumination. The total flux ($F_{\rm tt}$) is determined by the ionization parameter (ξ) and hydrogen density (n_h), and `fx-frac` (denoted as `frac` below) partitions this total flux between the top and bottom components:

$$F_{\rm tt} = \frac{\xi n_h}{4\pi} \tag{3.4}$$

$$F_{\rm top} = {\rm frac} \times F_{\rm tt} \tag{3.5}$$

$$F_{\rm bb} = (1 - {\rm frac}) \times F_{\rm tt} \tag{3.6}$$

`zeta`

Initial value of the log (base 10) of the model ionization parameter at the innermost shell. The Tarter et al. (1969) form is used:

$$\xi = \frac{4\pi F_x}{n_H} \tag{3.7}$$

This value will be re-calculated after radiation field update.

`Abundance`

Elemental abundances relative to Hydrogen for elements from H to Zn.

`inci-file`

Specifies the path to the file containing the custom incident illumination spectrum. This parameter is required only when `inci-type` is set to `"file"`.

The file must be a plain text file where the first three lines serve as a header, followed by the spectral data:

- **Line 1**: An integer specifying the number of energy grids.
- **Line 2**: An integer flag for the units of the radiation field:
  - `0`: erg/cm^2/s/erg
  - `1`: Photons/cm^2/s/erg
  - `2`: erg^2/cm^2/s/erg
- **Line 3**: An integer flag for the units of the energy grid:
  - `0`: eV
  - `1`: keV
- **Line 4 onwards**: The spectral data in two columns (Energy vs. Flux).

Note that the code internally normalizes the input spectrum using the ionization parameter, as shown in Equation 3.7, where the flux F_x is in units of erg/cm^2/s.
Warning: You must ensure that the energy unit flag on line 3 (`0` for eV or `1` for keV) correctly corresponds to the energy values in the data. A mismatch will result in significant errors in the calculation.

`kernelpath`

Path to the Compton scattering redistribution function file. This file must contain the energy grids, temperature grids, and the redistribution data itself, with each component saved as an unformatted Fortran binary file.

We provide two pre-calculated redistribution files: `kernel_gaus5000` (using a Gaussian approximation) and `kernel_exact5000` (using the exact function).

If you wish to use a custom redistribution file, you must ensure the following:

- The number of energy bins in your file must be identical to the number set in the model parameter `nfrt`.
- Our default files use 70 temperature grids. If your file uses a different number of grids, you must update the `file_ntemp` parameter in the source code accordingly. Ensure the temperature range covered by your grids is sufficient for your expected results.

dataenv

Specifies the path to the directory containing the atomic data table and the Compton heating-cooling file. Different versions of the atomic data files can be downloaded from the official XSTAR webpage.
Note: You must ensure that the specified directory exists and that the required data files are present within it. The program will terminate with an error if the path is invalid or if the files cannot be found.

output

Determines the base path and filename prefix for all output files.

When executing the code via the main Python script, a unique output path is generated automatically based on the input parameters. However, if you are running the code manually, you must set this parameter explicitly.

The primary output files include:

- `op_spec`: The final emergent spectrum for each iteration.
- `op_emis`: The emissivity at first layer of the slab for last iteration (not use for now).
- `op_inte`: The angle-dependent radiation intensity for each iteration.
- `rf_abund`: Heating-cooling, ions fraction along depth fortlast iteration.
- `op_temp`: Temperature profile along the depth for each iteration.
- `op_log`: The main log file for the calculation run.

3.2 Model Output

`DAO` is a code designed to calculate the reprocessed X-ray radiation from an illuminated plasma slab. It solves for the physical and ionization structure of the plasma in equilibrium and provides a comprehensive suite of output products. Key outputs include the angle-averaged emergent spectrum, angle-dependent intensity, emissivity, opacity, temperature structure, ionic abundances, and detailed heating/cooling rates. A log file is also generated for each run.

3.2.1 op_spec

This file is a diagnostic output that records the state of the radiation field at each main iteration step.

The file consists of sequential data blocks, with each block corresponding to a single iteration. Within each block, the data is organized into the following columns:

- **Column 1**: Energy grid values [ev].
- **Column 2**: The incident illumination spectrum from the top source.
- **Column 3**: The incident illumination spectrum from the bottom source (if active).
- **Column 4**: The emergent specific intensity along the line of sight (defined by the cosine of the incidence angle, `inmu`).
- **Column 5**: The angle-averaged (flux) spectrum.

All spectral quantities in this file are given in units of erg/cm^2/s/erg. The data block for the first iteration appears first, followed by the second, and so on, until the final converged result.

3.2.2 op_inte

This file provides a detailed diagnostic record of the angle-dependent specific intensity, logging its state at each main iteration step.

- **Column 1**: The energy grid, with units of eV.
- **Columns 2 through 11**: The emergent specific intensity for each of the first 10 angle bins. Column 2 contains the intensity for the first angle (μ_1), Column 3 for the second angle (μ_2), and so on.

The specific intensity values are given in units of erg/cm^2/s/erg/ster. The data block for the first iteration appears at the beginning of the file, followed by the subsequent iterations, with the final converged result in the last block.

3.2.3 op_temp

Temperature profile along the vertical direction for each iteration step.

- **Column 1**: Thomson optical depth.
- **Columns 2**: Temperature value in K

3.2.4 op_abund

This file follows the standard format of the `xout_abund1.fits` file from the `XSTAR`, making it compatible with many existing analysis tools.

The file archives the detailed plasma state from the final, converged iteration step, containing ion abundances, column densities, and heating/cooling rates. The elements are ordered by increasing nuclear charge, and for each element, ions are ordered by their increasing ionization state.

The FITS file is structured into four primary extensions:

`ABUNDANCES` This extension contains the ionic abundances for each spatial zone (layer) of the slab. Each row corresponds to a single zone and reports the following physical properties:

- Radius and size of the zone
- Local ionization parameter (ξ)
- Electron fraction, density, pressure, and temperature
- The fractional balance of heating vs. cooling
- The abundance of each ion relative to its parent element

`COLUMNS` This extension has a format similar to `ABUNDANCES` but contains the integrated column density for each ion, with units of cm^{-2}. These values represent the abundances from the previous extension, integrated through the slab and weighted by the elemental abundances.

Note: The abundance of the fully stripped ion is not explicitly tabulated for any element. It can be calculated by subtracting the sum of all other tabulated ion fractions for that element from one.

`HEATING` **and** `COOLING` These final two extensions list the primary heating and cooling rates in units of erg cm^{-3} s^{-1}.

- The `HEATING` extension lists the Compton and total heating rates.
- The `COOLING` extension lists the Bremsstrahlung, Compton, and total cooling rates.

3.2.5 op_log

This file contains a detailed log of the entire calculation process, recording the status at each main iteration step and each subsequent radiative transfer iteration.

For each step, it typically records key physical quantities such as the ionization parameter, temperature. If the code encounters an error or fails to converge, this file will be the most critical resource for diagnosing the problem.

3.2.6 op_emis

Not use for now.

3.3 Running the Code with Python

Managing simulations with numerous parameters can be an error-prone task, particularly when testing many different configurations. To simplify this process and help organize the resulting outputs, we provide a Python interface designed to run the model and manage the data systematically.

3.3.1 Core Components

The Python interface is composed of three main files that work together:

Execution Script: `run.sh` This is the main script you will execute from the command line. It is a shell script where you can set execution parameters (like the `DIY` tag and run mode) and then launch the main Python script. **This is the primary file you will interact with to launch a new run.**

Physics Configuration: `mopar.py` This Python module serves as the central configuration file for the physical model. It contains all the Fortran model parameters and includes a dictionary that maps Python variable names to their Fortran counterparts.

Orchestration Script: `modelrun.py` This is the main Python script that orchestrates the entire simulation. It reads the parameters from `run.sh`, generates the necessary input files, executes the compiled `DAO` program, and organizes the output. You will typically not need to edit this file.

3.3.2 Basic Workflow

Running a simulation typically involves the following three steps:

1. **Define the Physics:** Open and edit `run.sh` to set your parameters.
2. **Configure the Run:** Edit `run.sh` to set your user-defined tag (`DIY`) and choose an execution mode (`exec`).
3. **Launch the Simulation:** Execute the script from your terminal: `./run.sh`.

3.3.3 Key Features and Concepts

The Python interface includes several powerful features designed to make your research workflow more organized and reproducible.

Automated Output Management

To prevent accidental overwriting of results and to keep simulations organized, the script implements an automated output management system based on two components:

User-Defined Tag (`DIY`) You can assign a human-readable tag to a group of simulations. This tag becomes the name of the main output subdirectory, allowing you to group related runs together (e.g., runs where you are testing the same physical parameter).

Auto-Generated Hash The script automatically generates a unique hash value based on all the core physical parameters. This ensures that any two runs with even slightly different physical setups will have unique identifiers, preventing data from being overwritten.

All output files are then stored and named using a combination of this tag and hash.

Example Workflow

Let's say your scientific goal is to test the effect of varying hydrogen density while using a model with 5000 energy grids.

1. You would set the user-defined tag in `run.sh`:

   ```
   DIY="dnh_ener5000"
   ```

2. The script would then create a main output directory: `out/dnh_ener5000/`
3. For each specific value of n_H, the script generates a unique hash (e.g., `aad11c7ab6`). All output files for that run are then named using this hash and tag:
   - `out/dnh_ener5000/spec_aad11c7ab6_dnh_ener5000.dat`
   - `out/dnh_ener5000/abund_aad11c7ab6_dnh_ener5000.fits`
   - ...and so on.

This system ensures that all your tests are neatly organized, while each individual run has a unique, reproducible identifier.

Slurm Cluster Integration

The script supports integration with the Slurm Workload Manager for running jobs on an HPC cluster. This is controlled by the `exec` parameter in `run.sh`.

`exec = 0` **(Generate-Only Mode)** Performs a "dry run." It generates all necessary input files but does not execute the simulation. This is useful for preparing a large batch of runs or for debugging.

`exec = 1` **(Slurm Submission Mode)** Automatically generates a Slurm submission script and submits the simulation as a job to the cluster using `sbatch`.

Important Configuration Step: Before using Slurm submission, you **must** customize the Slurm template to match your cluster's configuration:

`templates/submission.slurm.j2`

You will need to edit this file to set correct values for your system, such as the partition, account, job time limits, and memory requirements (`#SBATCH -partition=...`, `#SBATCH -account=...`, etc.). Failure to do so will result in job submission errors.

Automatic Metadata and Reproducibility

To ensure the traceability of every simulation, a comprehensive record is automatically generated and stored as a JSON file in the `metadata/` directory. The filename of the JSON record corresponds directly to the unique hash and tag of its simulation.

This file is a self-contained summary of the run, containing:

- A complete dictionary of all input physical parameters.
- The `DIY` tag and the auto-generated hash value.
- Relative paths to all generated output files.
- Execution metadata, such as the run ID, status (`exec`) and start time.

This feature is powerful for managing large sets of simulations, as you can easily write scripts to parse these JSON files to programmatically analyze, plot, or archive your results.

Chapter 4

Examples

4.1 Effect of ionization parameters

The Python interface is particularly powerful for running a grid of models on a cluster. The example `run.sh` script, shown in Listing follows, is configured to demonstrate this by looping over 16 different values of the ionization parameter, `zeta`.

```
#!/bin/bash

# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
#                                                                              #
#  Description: This is a runner script for modelrun.py, containing all       #
#               the input parameters for the model.                            #
#                                                                              #
#  Author: Yimin Huang, Fudan University (Updated based on mopar config)      #
#                                                                              #
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

# ======================================================================
# Script to run the model with a comprehensive set of parameters.
# Variable names are synchronized with the 'mopar' Python module.
# ======================================================================
PYTHON_SCRIPT="modelrun.py" # <-- Make sure this is your main Python script name
DIY="DAOguide"

# --- Execution & Model Flags ---
exec=1      # (0: Generate files only, 1: Submit with sbatch, 2: Run locally)
debug=0     # Debug switch
model=0     # (0: Reflection, 1: Pure scattering)
```

```
ity=2       # Type of angle bins and integral method

# --- Computational Grids ---
nmaxmain=15         # Number of maximum main iteration steps
nmaxrte=100         # Number of maximum radiative transfer iteration
   steps
ndrt=200            # Number of depth grids (tau)
nfrt=5000           # Number of energy grids [Must match kernel file]
nmurt=10            # Number of angle grids (mu)
ppemin=0.1          # Minimum energy of the grid [eV]
ppemax=9e5          # Before maximum energy of the grid [eV]
ppemax2=1e6         # Maximum energy of the grid [eV]
taumin=1e-4         # Minimum Thomson optical depth of the grid
taumaxrt=10.0       # Maximum Thomson optical depth of the grid
mumin=0.05          # Minimum mu value for the angular grid (> 0)
mumax=0.95          # Maximum mu value for the angular grid (< 1)

# --- Temperature & Equilibrium ---
it_temp=99          # Maximum iteration steps for thermal equilibrium
temp_gas=1e8        # Initial gas temperature
temp_gas_unit='k'  # Unit of the initial temperature ('k' or 'ev')

# --- Incident Spectrum Parameters ---
inci_type="file"               # Type of corona illumination ('powerlaw
   ', 'file', 'cutoff')
gamma=1e3                          # Photon index or Blackbody
   temperature in eV
hcut=300e3                       # High energy cut-off for cut-off
   power law illumination [eV]
inmu=0.7                         # Cosine of the incident illumination
   angle
inci_file="incident/nthcomp_g2.4_t60.txt" # Path to incident spectrum
   file

# --- Bottom Illumination (Disk) ---
sbot=0              # Switch for bottom illumination (0: off, 1: on)
ktbb=350.0          # Temperature of the bottom blackbody source [eV]
fx_frac=1e-10       # Flux ratio of top illumination to total
   illumination

# --- Atmosphere Properties ---
zeta=3.0            # Log10 of ionization parameter (xi)
nh=1e15             # Hydrogen number density [cm^-3]
fe=1.0              # Iron abundance relative to Hydrogen

# --- Paths ---
kernelpath="kernel_exact5000" # Path to the Compton scattering
   redistribution file

#
   ================================================================================

# --- Execute the Model ---
# The script will now call the Python program with all parameters
   defined above.
# The parameter names (--parameter-name) match the 'py_name' in mopar.
#
   ================================================================================
```

```
69
70 echo "----------------------------------------"
71 echo "Starting model run with DIY tag: ${DIY}"
72 echo "----------------------------------------"
73
74 # Example loop: You can loop over any parameter.
75 # Here, we loop over the incident angle 'inmu'.
76 for zeta in 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0 3.2 3.4 3.6 3.8
       4.0
77 do
78     echo "Running with nh = ${zeta}"
79
80     python ${PYTHON_SCRIPT} \
81         --exec ${exec} \
82         --debug ${debug} \
83         --model ${model} \
84         --ity ${ity} \
85         --nmaxmain ${nmaxmain} \
86         --nmaxrte ${nmaxrte} \
87         --ndrt ${ndrt} \
88         --nfrt ${nfrt} \
89         --nmurt ${nmurt} \
90         --ppemin ${ppemin} \
91         --ppemax ${ppemax} \
92         --ppemax2 ${ppemax2} \
93         --taumin ${taumin} \
94         --taumaxrt ${taumaxrt} \
95         --mumin ${mumin} \
96         --mumax ${mumax} \
97         --it-temp ${it_temp} \
98         --temp-gas ${temp_gas} \
99         --temp-gas-unit "${temp_gas_unit}" \
100        --inci-type "${inci_type}" \
101        --gamma ${gamma} \
102        --hcut ${hcut} \
103        --inmu ${inmu} \
104        --inci-file "${inci_file}" \
105        --sbot ${sbot} \
106        --ktbb ${ktbb} \
107        --fx-frac ${fx_frac} \
108        --zeta ${zeta} \
109        --nh ${nh} \
110        --fe ${fe} \
111        --kernelpath "${kernelpath}" \
112        --diy "${DIY}"
113
114    echo "----------------------------------------"
115 done
116
117 echo "Job submission script has finished."
118 echo "----------------------------------------"
```

After the script has finished, you will have a complete set of 16 distinct simulations either prepared or running on the cluster. The next step in the scientific workflow is to use these generated metadata files. As each JSON file contains the direct path to the output files for a specific run, you can easily write a separate analysis script to read these records, retrieve the results, and plot the final spectra as a function of the ionization parameter

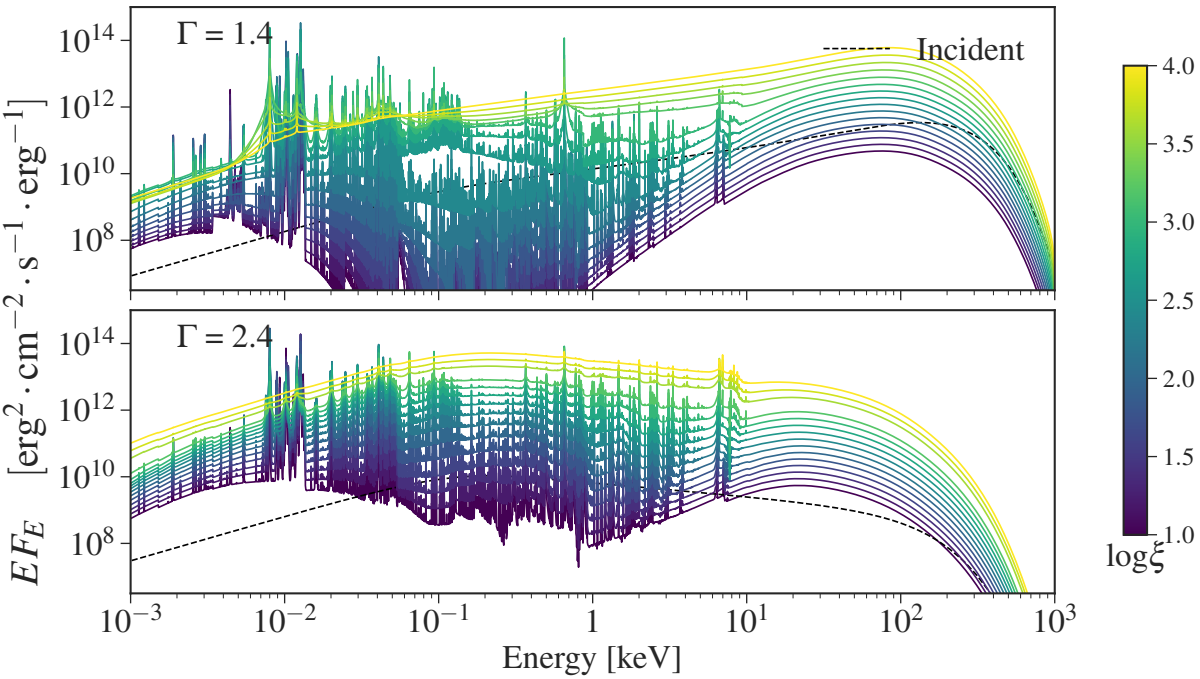

Figure 4.1: Reflection spectrum for different ionization parameters. Top panel: `nthcomp` incident with $\Gamma = 1.4$. Bottom panel: `nthcomp` incident with $\Gamma = 2.4$.

```
METADATA_DIR = 'metadata'
if not os.path.isdir(METADATA_DIR):
    raise FileNotFoundError(f" '{METADATA_DIR}' not exist")
for data_filename in datas:
    with open(os.path.join(METADATA_DIR, data_filename), 'r') as f:
        try:
            data = json.load(f)
            params = data['parameters']
            if params.get('diy_tag') == 'dxi5000':
                spec= params['op_spec']
        except (json.JSONDecodeError, KeyError, TypeError) as e:
            print(f"Warning: Jump {data_filename},: {e}")
            continue

.......

plot_data_list.sort(key=lambda item: item['zeta'])
zeta_values = [item['zeta'] for item in plot_data_list]

.......

PLOT
```

The results are shown in Figure 4.1

4.2 Emergent intensity

For quick tests or single simulations, manually editing the Fortran namelist input file can be a straightforward alternative to using the Python interface. This method gives you

direct control over all model parameters without the overhead of the wrapper script.

An example of a partially configured input file, which we can name `input.dat`, is shown in Listing 4.2. Note how file paths and physical parameters are set directly.

```
&input_params
    ...
    rfstinci = 'file'
    rfinci_file = 'incident/nthcomp_g2.4_t60.txt'
    ...
    rfcomp_file = 'kernel_exact5000'
    rfdataenv = 'data'

    ! --- Output file paths ---
    rfop_spec = 'out/spec_inputout.dat'
    rfop_temp = 'out/temp_inputout.dat'
    rfop_inte = 'out/inte_inputout.dat'
    rfop_log = 'out/model_inputout.log'
    rfop_emis = 'out/emis_inputout.dat'
    rfabdfits = 'out/abund_inputout.fits'

    rfmodel = 0
    ...
    rfafe = 1.0
    ...
/
```

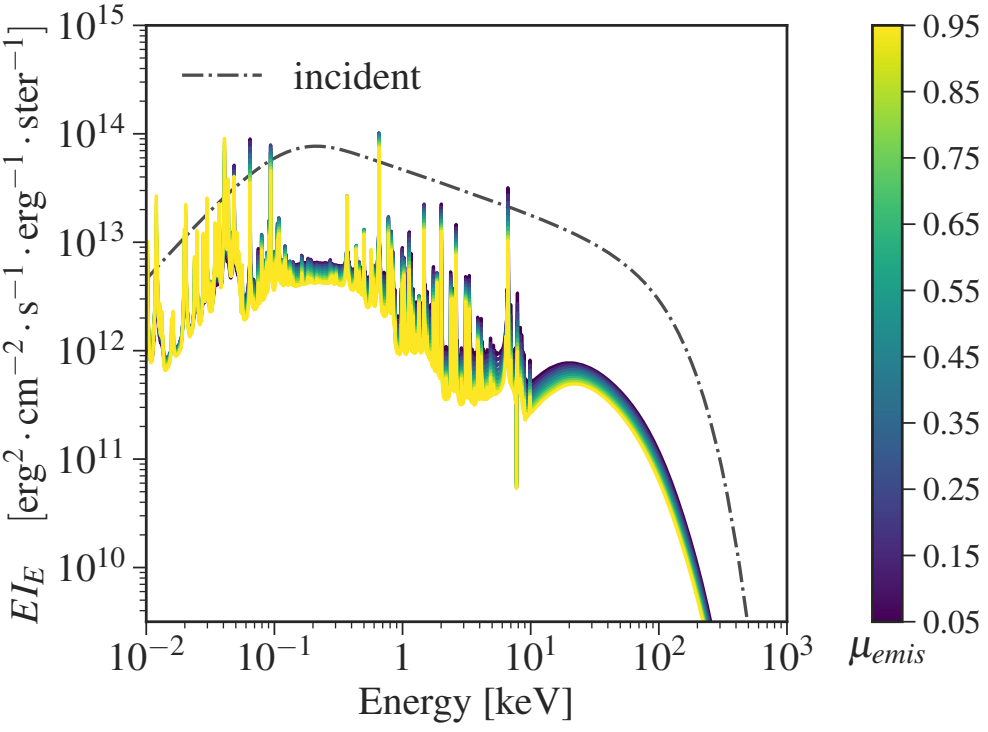

Figure 4.2: The emergent intensity profile produced by the simulation using the manual input parameters shown in Listing 4.2.

After preparing this file, you can execute the compiled Fortran program directly from the command line. The program will read the parameters from this file and write the

results to the specified output paths.

The emergent intensity from this specific run, which is written to the file defined by `rfop_inte`, is plotted in Figure 4.2.

4.3 Ions Fractions

This example demonstrates how to use the `DAO` Python interface to investigate the effect of the ionization parameter (ξ) on the physical structure of the plasma.

Simulation Setup

We will run two simulations with different ionization parameters to compare the resulting ionic fractions. The primary physical parameters are set as follows:

- **Ionization Parameter** ($\log \xi$): Two separate runs are performed, one with `zeta = 3.0` and another with `zeta = 4.0`.
- **Incident Spectrum**: The illumination is an `nthcomp` spectrum, defined in an external file, with a photon index $\Gamma = 2.0$ and an electron temperature $kT_e = 60$ keV.

After running the `run.sh` script with these settings, we proceed to analyze the output FITS files.

Analysis and Results

```python
import numpy as np
from astropy.io import fits
# Note: The 'to_roman' function is a helper you would need to define, e.g.,
# def to_roman(n): return ["", "I", "II", ...][n]

def loadabund(path):
    hdul= fits.open(path)
    abund=hdul[1].data
    return abund

def get_ionfrac(abund,ionname):
    names = [name for name in abund.columns.names if name.startswith(ionname)]
    ionfrac = np.zeros((len(names),len(tau)))
    for i,name in enumerate(names):
        ionfrac[i] = abund[name]
    ionfrac /= ionfrac.sum(axis=0)
    return ionfrac
def getinfo(abund):
    fefrac = get_ionfrac(abund,'fe_')
    ofrac = get_ionfrac(abund,'o_')
    cfrac = get_ionfrac(abund,'c_')
    ioninfo = {
        'fe':{
            "major_ions": [1, 5, 10, 15, 20, 26],
            "major_locs": [ion + 0.5 for ion in [1, 5, 10, 15, 20, 26]],
```

```
            "major_labels":[f"Fe {to_roman(i)}" for i in [1, 5, 10, 15, 20, 26]]
        },
        'o':{
            "major_ions": [1,2,3,4,5,6,7,8],
            "major_locs": [ion + 0.5 for ion in [1,2,3,4,5,6,7,8]],
            "major_labels":[f"O {to_roman(i)}" for i in [1,2,3,4,5,6,7,8]]
        },
        "c":{
            "major_ions": [1,2,3,4,5,6],
            "major_locs": [ion + 0.5 for ion in [1,2,3,4,5,6]],
            "major_labels":[f"C {to_roman(i)}" for i in [1,2,3,4,5,6]]
        }
    }
    return fefrac,ofrac,cfrac,ioninfo

# --- How to get the path programmatically (linking to previous section) ---
# This is a conceptual example. You would use the metadata parser from
# the previous section to get the actual file path for a given run.
# For example: path = get_path_from_metadata('...hash_for_zeta3...', 'ion_frac_test')
path_zeta3 = 'out/ion_frac_test/abund_...hash_zeta3..._ion_frac_test.fits'

# Optical depth has been stroed in op_temp file

# --- Main processing steps ---
abund = loadabund(path_zeta3)
fefrac,ofrac,cfrac,ioninfo = getinfo(abund)

# You can now use these arrays (e.g., fe_frac3) with Matplotlib's imshow
# to create the colormaps shown in the figure.
```

The resulting ionic fractions for Iron (Fe), Oxygen (O), and Carbon (C) are visualized as colormaps in Figure 4.3. The plots intuitively show the stratification of different ions as a function of Thomson optical depth. As expected, the higher ionization parameter ($\log \xi = 4.0$) clearly pushes the ionization balance towards more highly stripped ions, causing the transition zones for each element to shift to greater depths within the slab.

Data Processing Script

To generate these plots, one must read the output FITS abundance files (`*.abund_...fits`). The following Python code snippet provides a practical example of how to load the data using the `Astropy` library and process it to extract the ionic fractions for specific elements.

4.4 Heating-Cooling rate

Same with Chapter 4.3, you only need to read ASCII file to get heating cooling rate. The example heating-cooling rate for different hydrogen density as shown in Figure 4.4 - 4.5

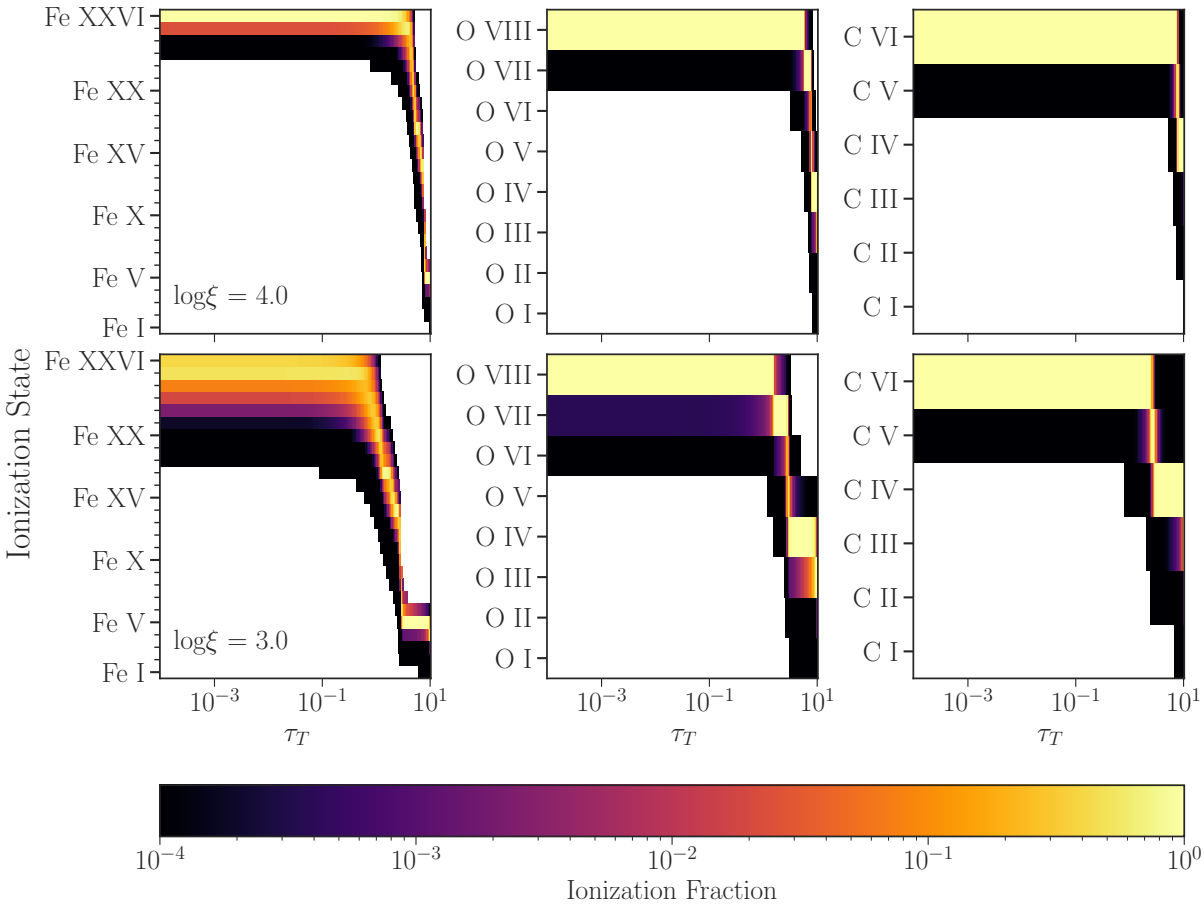

Figure 4.3: Comparison of the ionic fractions of Iron (Fe), Oxygen (O), and Carbon (C) as a function of Thomson optical depth for two different ionization parameters: $\log \xi = 3.0$ (top row) and $\log \xi = 4.0$ (top row).

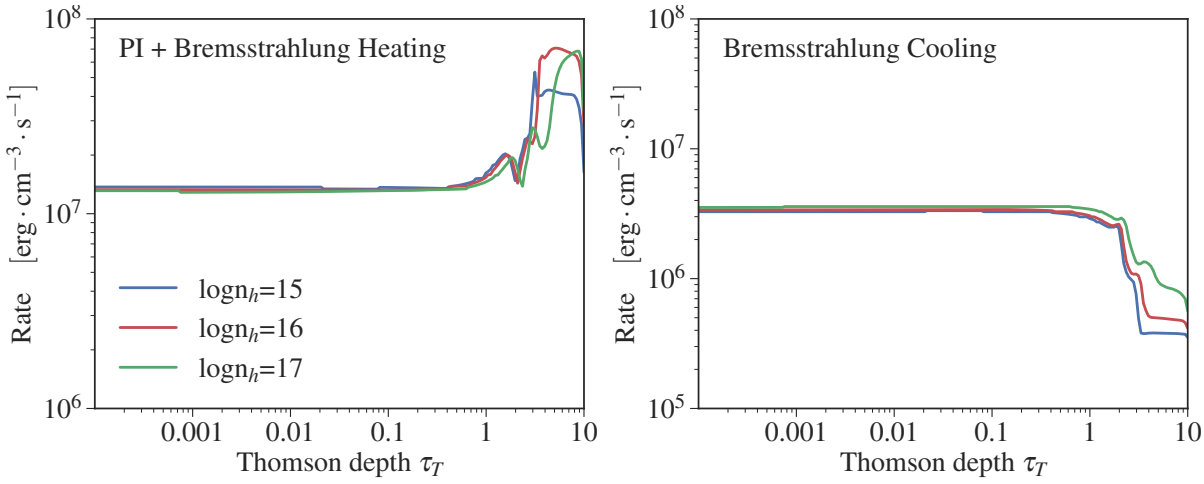

Figure 4.4: Photoionization + free-free heating and bremsstrahlung cooling

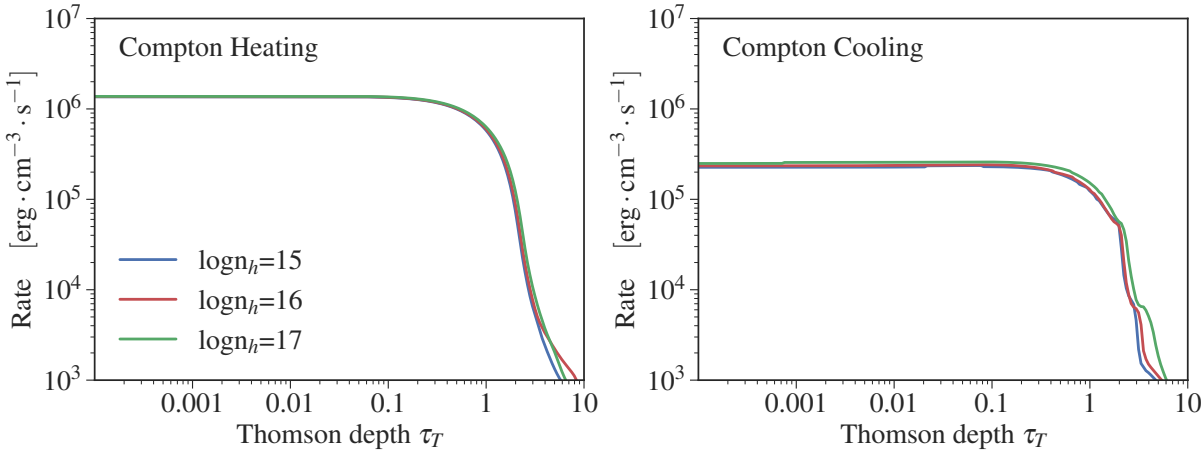

Figure 4.5: Compton heating-cooling rate

Chapter 5

Physics and math behind DAO

5.1 Radiative transfer

Radiative transfer is a basic problem in astrophysics particularly in atmosphere. The basic form of radiative transfer equation in plane-parallel geometry can be expressed by

$$\mu \frac{\partial I(z,\mu,\nu)}{\partial z} = \chi(z,\mu,\nu) I(z,\mu,\nu) - \eta(z,\mu,\nu) \tag{5.1}$$

where, z, μ, ν denote a specific depth, angle energy point in plasma. And I is specific intensity, χ is opacity and η is the emissivity. The photons travel in a plasma will interact with many elements in it which cause the absorption and emission in plasma. Due to artistic design of `XSTAR`, `DAO` can account for a wide range of atomic physic, it almost contains all possible process such as free-free, bound-free, three body recombination, collision, radiative recombination, etc. These process construct the opacity and emissivity in radiative transfer equation, in thing divided is Compton scattering.

When we construct model, one convenient ways is represent this equation by the optical depth, where $\tau(E) = \int \chi(E) dz$. Then the Eq 5.1 can be expressed by

$$\mu \frac{\partial I(\tau_T,\mu,\nu)}{\partial \tau(\nu)} = I(\tau_T,\mu,\nu) - S(\tau_T,\mu,\nu) \tag{5.2}$$

where the second term in right hand is called source function it's

$$S_{\mu\nu} = \frac{\eta_{\mu\nu}}{\chi_{\mu\nu}} \tag{5.3}$$

we omit the dependence of optical depth. One could notice that in Eq 5.2, radiation field depend on Thomson optical depth τ_T, where $d\tau_T = n_e \alpha_T dz$ rather than τ. We set that because the model calculate in a confirmed depth grids, and it's the best choice for determining the depth grids. The same application can be found in many radiative transfer model such as `reflionx`.

There are many ways to solve the radiative transfer equation: Approximated, Directly way and iterative ways. We choose the Lambda-iteration with Feautrier method (hereafter FTM) to solve it by considering the accuracy and computational resource.

Feautrier Method

FTM is based on second order of radiative transfer equation. First if we consider symmetric and antisymmetric averages of the specific intensity at frequency ν propagating

along a ray in the $\pm\mu$ direction are

$$u_{\mu\nu} = \frac{1}{2}\left[I_\nu(\mu) + I_\nu(-\mu)\right] \quad \text{and} \quad v_{\mu\nu} = \frac{1}{2}\left[I_\nu(\mu) - I_\nu(-\mu)\right] \tag{5.4}$$

The transfer equation for two directions are

$$\mu\frac{\partial I_\nu(\mu)}{\partial \tau_\nu} = I_\nu(\mu) - S_\nu(\mu) \quad \text{and} \quad \frac{\partial I_\nu(-\mu)}{\partial \tau_\nu} = I_\nu(-\mu) - S_\nu(-\mu) \tag{5.5}$$

Adding Eq 5.4,

$$\mu\frac{\partial v_{\mu\nu}}{\partial \tau_\nu} = u_{\mu\nu} - S_\nu \quad \text{and} \quad \mu\frac{\partial u_{\mu\nu}}{\partial \tau_\nu} = v_{\mu\nu} \tag{5.6}$$

Then substituting the expression for $v_{\mu\nu}$, we obtain *Schuster's second-order transfer equation (Schuster 1905)*

$$\mu^2\frac{\partial^2 u_{\mu\nu}}{\partial \tau_\nu^2} = u_{\mu\nu} - S_\nu \tag{5.7}$$

where $0 \leq \mu \leq 1$.

To numerically solve that, the discrete form could be get by finite difference. We use m,n,d represent the angle, energy, and depth points.

$$\Delta\tau_{d+\frac{1}{2},n} \equiv \tau_{d+1,n} - \tau_{d,n} \tag{5.8}$$

Then the transfer funciton:

$$\frac{\mu u_{d-1}}{\Delta\tau_{d-\frac{1}{2}}\Delta\tau_d} - \frac{\mu^2 u_d}{\Delta\tau_d}\left(\frac{1}{\Delta\tau_{d-\frac{1}{2}}} + \frac{1}{\Delta\tau_{d+\frac{1}{2}}}\right) + \frac{\mu^2 u_{d+1}}{\Delta\tau_{d+\frac{1}{2}}\Delta\tau_d} = u_d - S_d \quad (d = 2, ..., ND-1) \tag{5.9}$$

where we omit the subscript of angle and energy for convenient and ND represent the maximum number of depth grids. In this expression, one need to take care for source function:

$$S_n = \frac{j_n}{\alpha_n + \alpha_n^c} \quad , \quad j_n = j_n^\alpha + \alpha_n^c J_n^c \tag{5.10}$$

which angle-independent. In the denominator, α and α^c represent opacity of absorption and scattering, respectively. And j_n^α represent the emissivity include all possible atomic process, J^c represent the scattered mean-intensity,

$$J_n^c = \sum_{n'} R_{n,n'} J_{n'} w_{n'} \tag{5.11}$$

where $R_{n,n}$ is redistribution function, $J_{n'}$ is mean-intensity before scattering and $w_{n'}$ is integral weight. Equation 5.11 describe how we deal with Compton scattering process, as a most important physics process in reflection. Redistribution we used in `DAO` is calculated by complete quantum mechanism and suitable for relativistic electrons, we will discuss later in Chapter 5.2

We consider external illumination at both of upper and bottom surface. At these boundaries, the second exact form from Auer (1967) is used:

$$\mu\frac{u_2 - u_1}{\Delta\tau_{3/2}} = u_1 + \Delta\tau_{3/2}(u_1 - S_1)/2\mu + I_{\text{inc}} \tag{5.12}$$

and the same expression could be found for lower surface.

The discretized radiative transfer equation Eq 5.9 has a block tridiagonal system:

$$-A_d u_{d-1} + B_d u_d - C_d u_{d+1} = L_d \tag{5.13}$$

$$B_1 u_1 - C_1 u_2 = L_2 \tag{5.14}$$

$$-A_{n_\tau} u_{n_\tau - 1} + B_{n_\tau} u_{n_\tau} = L_{n_\tau} \tag{5.15}$$

where A,B,C,L are diagonal in the problem we considering. Equation 5.13-5.15 could be solved by forward-elimination and back-substitution procedure. Once it's been solved, the new radiation field will been scattered and update the source function, which named Λ-iteration.

The complete iterative procedure is summarized as follows:

1. Obtain the initial estimates of the emissivity and absorption coefficients using a naive radiation field.
2. Compute the source function using Eq. (5.10).
3. Apply the Feautrier method to obtain an updated radiation field based on the source function from step 2.
4. Repeat steps 2–3 until convergence is achieved.

The convergence criterion adopted in `DAO` is defined as

$$E_\tau \left[\int \frac{S_n(E) - S_{n-1}(E)}{S_{n-1}(E)} \, dE \right] < \varepsilon, \qquad \varepsilon = 10^{-10}, \tag{5.16}$$

which ensures that the source function ceases to change between successive iterations. Additionally, radiative equilibrium is enforced, which, in its simplest form, can be expressed as

$$S = J \tag{5.17}$$

where J denotes the mean intensity. In fact, these two conditions are equivalent, since the Λ-iteration is expected to converge when the source function equals the mean intensity.

However, there are many advanced iteration process such as Accelerate-Lambda-Iteration (ALI). We didn't implement it because of huge memory cost by large number of energy grids. ALI need to solve linear equation such as

$$\mathbf{T} \cdot \mathbf{x} = \mathbf{b} \tag{5.18}$$

In general, $\mathbf{T}$ is a full matrix, while $\mathbf{x}$ and $\mathbf{b}$ are vectors. When the complete redistribution function $R_{\nu,\nu'}$ is considered, the problem becomes considerably more complex, as a $200 \times 5000 \times 5000$ matrix is introduced. At each depth point, it is necessary to compute the inverse of a 5000×5000 matrix, which scales with a computational complexity of $O(n^3)$. In contrast, the method implemented in `DAO` scales linearly with $O(n)$ complexity, although it still requires a significant amount of time to achieve convergence.

5.2 Compton Scattering

Essentially, Compton scattering is a stochastic process that alters both the propagation direction and the energy of photons, and it should be described using the exact quantum mechanical formalism. In addition, in high temperature gas, motion of relativistic electrons follow the Maxwellian distribution, which need to be taken into account in complete redistribution function.

Historically, many authors had investigated such unpolarized and quantum mechanism scattering redistribution function: Guilbert (1981) incorporated the relativistic Maxwellian velocity distribution for electrons into the redistribution function, but a computational error was present in the original paper. Subsequently, Nagirner & Poutanen (1993); Poutanen & Svensson (1996); Poutanen & Vurm (2010) investigated the redistribution function between photons and relativistic electrons using quantum electrodynamical methods. Later, Madej et al. (2017) corrected the earlier error of Guilbert and demonstrated consistency among these methods. Recently, García et al. (2020) showed the large difference between exactly redistribution function and Gaussian-approximated. The former is derived from Nagirner & Poutanen (1993) and the later is been widely use in reflection model (Ross et al. 1978; Ross & Fabian 2005; Nayakshin et al. 2000).

In `DAO` model, the parameter `kernelpath` allow user to set any type of redistribution function they want. Considering dependence of accuracy for physic probelm, we highly recommend user generate the exact redistribution function by using script in `get_redistribuion`, where we use the code of García et al. (2020) to calculate Nagirner & Poutanen (1993). The calculation routine as follows:

Exact redistribution

$$R_E(x, x_1, \mu, \gamma) = \frac{2}{Q} + \frac{u}{v}\left(1 - \frac{2}{q}\right) + u\frac{(u^2 - Q^2)(u^2 + 5v)}{2q^2v^3} + u\frac{Q^2}{q^2v^2}, \tag{5.19}$$

with x and x_1 the dimensionless photon energies before and after scattering, respectively, μ is the cosine of the scattering angle, and γ represents electron Lorentz factor. Here, $q = xx_1(1 - \mu)$ and $Q^2 = (x - x_1)^2 + 2q$, while

$$a_-^2 = (\gamma - x)^2 + \frac{1 + \mu}{1 - \mu}, \qquad a_+^2 = (\gamma + x_1)^2 + \frac{1 + \mu}{1 - \mu}, \tag{5.20}$$

$$v = a_-a_+, \qquad u = a_+ - a_-. \tag{5.21}$$

The redistribution function averaged over a relativistic Maxwellian distribution is

$$R(x, x_1, \mu) = \frac{3}{32\mu\Theta K_2(1/\Theta)} \int_{\left(x - x_1 + Q\sqrt{1+2/q}\right)/2}^{\infty} R(x, x_1, \mu, \gamma) \exp(-\gamma/\Theta), d\gamma, \tag{5.22}$$

where K_2 is the modified Bessel function of the second kind and $\Theta = kT/m_ec^2$. Currently we use angle-averaged redistribution so the resulting function is

$$R(x, x_1) = \int_{\mu_{\min}}^{\mu_{\max}} R(x, x_1, \mu) d\mu \tag{5.23}$$

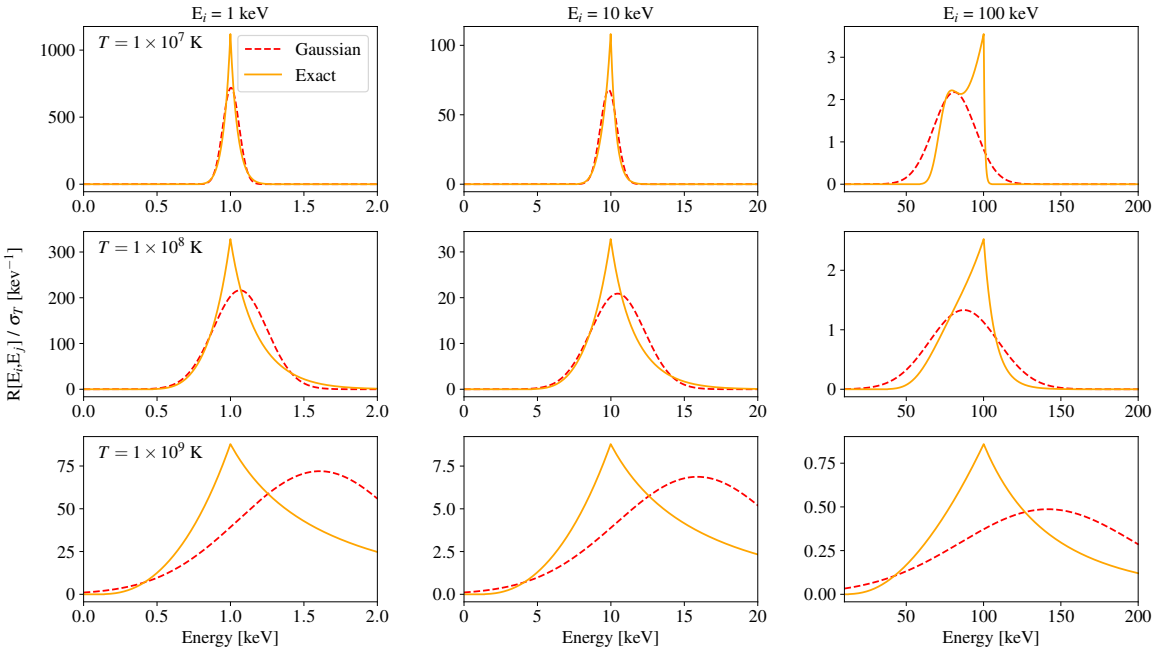

Figure 5.1: Gaussian (red dash line) and Exact redistribution (orange solid line) function with three initial energy: E_i = 1, 10, 100 keV and three different gas temperature: 10^7, 10^8, 10^9 K.

Gaussian redistribution

$$P(E_i, E_f) = \frac{1}{\sqrt{2\pi}\Sigma} \exp\left[-\frac{(E_f - E_c)^2}{2\Sigma^2}\right], \tag{5.24}$$

with central energy E_c and standard deviation Σ defined as

$$E_c = E_i \left(1 + \frac{4kT}{m_e c^2} - \frac{E_i}{m_e c^2}\right), \tag{5.25}$$

$$\Sigma = E_i \left[\frac{2kT}{m_e c^2} + \frac{2}{5}\left(\frac{E_i}{m_e c^2}\right)^2\right]^{1/2}. \tag{5.26}$$

In Figure 5.1, we shows the large deviation between exactly redistribution function and Gaussian-approximated function when gas temperature is high and photons have large energy. Same results have been reported in García et al. (2020). We show here just for convenient for readers to see differences.

Normalization

Scattered radiation field

In expression of source function Eq 5.10, the scattered mean-intensity J^c is seen as a thermal emission, where

$$J^c(E') = \int_{E_{min}}^{E_{max}} R(E', E) J(E) dE \tag{5.27}$$

Currently, angular redistribution is not included in DAO. If it were, Eq. 5.27 would take the form

$$I(\mu', E') = \int_{E_{min}}^{E_{max}} dE \int_{-1}^{1} d\mu R(E'; \mu', E; \mu) I(\mu, E) \tag{5.28}$$

where E' and μ' denote the photon energy and angle after scattering. In this case, the redistribution becomes more complex, depending on both the incident and scattered energies and angles. If one consider scattering angle η ($=\mu \cdot \mu'$), redistribution function change to a function of η, E', E, which is more convenient for calculation.

Chapter 6

Vesion

6.1 v1.0

Standard `DAO` model

6.2 v1.1

1. Add thermally comptonized continuum (`nthcomp`) as a type of incident spectrum. When incident type is `nthcomp`, `hcut` is used to represent corona temperature in keV. The seed photons' temperature is fixed at 0.1 keV.
2. TODO: Add isotropic illumination. i.e., set negative value for incidence angle to get isotropic illumination.

Chapter 7

Appendix

Bibliography

Auer, L. 1967, , 150, L53

Ding, Y., Garcıa, J. A., Kallman, T. R., et al. 2024, , 974, 280

García, J., Dauser, T., Lohfink, A., et al. 2014, , 782, 76

García, J., Dauser, T., Reynolds, C. S., et al. 2013, , 768, 146

García, J. & Kallman, T. R. 2010, , 718, 695

García, J. A., Sokolova-Lapa, E., Dauser, T., et al. 2020, , 897, 67

Guilbert, P. W. 1981, , 197, 451

Kallman, T., Bautista, M., Deprince, J., et al. 2021, , 908, 94

Madej, J., Różańska, A., Majczyna, A., & Należyty, M. 2017, , 469, 2032

Mendoza, C., Bautista, M. A., Deprince, J., et al. 2021, Atoms, 9, 12

Nagirner, D. I. & Poutanen, Y. J. 1993, Astronomy Letters, 19, 262

Nayakshin, S., Kazanas, D., & Kallman, T. R. 2000, , 537, 833

Poutanen, J. & Svensson, R. 1996, , 470, 249

Poutanen, J. & Vurm, I. 2010, , 189, 286

Ross, R. R. & Fabian, A. C. 2005, , 358, 211

Ross, R. R., Weaver, R., & McCray, R. 1978, , 219, 292

Schuster, A. 1905, , 21, 1

Tarter, C. B., Tucker, W. H., & Salpeter, E. E. 1969, , 156, 943