MAGRITTE: Multidimensional Accelerated General-purpose RadIaTive TransfEr

Code Documentation

Frederik De Ceuster

Abstract

This report gives an overview of some technical aspects of the Magritte code. The goal is to motivate why and explain how some things are coded the way they are.

Contents

1 General

Magritte is a multipurpose accelerated 3D radiative transfer code. Given a gas with a density and velocity distribution, an external radiation field and a provisional chemical composition, it selfconsistently calculates the temperature, level populations and chemical composition of the gas. Magritte is a ray-tracing code, meaning that to solve for the transport of electromagnetic radiation, the radiative transfer equation is solved along a set of rays.

The first version of Magritte is mainly based on 3D-PDR [1]. The main difference is that in Magritte the radiative transfer is solved exactly and not in the Sobolev or large velocity gradient (LVG) approximation.

The code is mainly written in C with some features of C++. For the ray tracing the discretization scheme of the unit sphere is used from HEALPix[1]. The chemical rate equations are solved using the CVODE solver provided in SUNDIALS[2]. Most of the linear algebra is done using the Eigen[3] library.

2 Some structures

2.1 (HEALPix) rays

To model the propagation of light through the grid, we need to trace rays through it i.e. determine which grid points you pass when you travel along a straight line in a certain direction. The first step is to discretize all directions. This is done using HEALPix. The algorithm considers a unit sphere and devides it into `NRAYS` segments. Afterwards it asigns unit vectors from the center of the sphere to the center each segments. These vectors form a discretization of all directions in 3D space. To facillitate the calculations of the radiative transfer, we require a discretization in which for each ray there is an antipodal ray (i.e. a ray in the oposite direction). This can be obtained by demanding that the `NRAYS` is of the form $12n^2$ where n is an integer. We consider `NRAYS` rays (as defined by the HEALPix vectors) around each grid point.

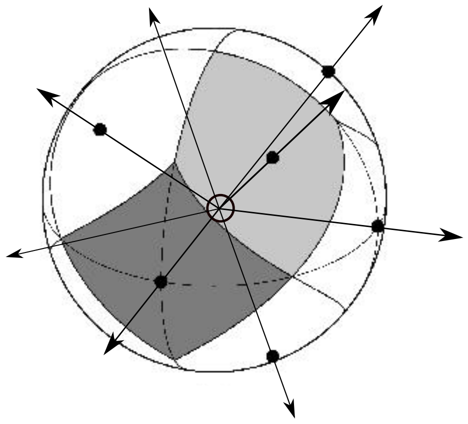
Figure 1: HEALPix rays through segment centers that discretize the sphere.

2.2 New types `GRIDPOINT`, `EVALPOINT`, `SPECIES` and `REACTION`

The code defines four new types: `GRIDPOINT`, `EVALPOINT`, `SPECIES` and `REACTION`. Each of these represents a type of objects that is used extensively in the code.

[1]https://healpix.jpl.nasa.gov
[2]https://computation.llnl.gov/projects/sundials
[3]http://eigen.tuxfamily.org

$$\texttt{GRIDPOINT gridpoint.}\begin{cases}\texttt{double x, y, z} & \text{position}\\ \texttt{double vx, vy, vz} & \text{gas velocity}\\ \texttt{double density} & \text{gas density}\end{cases}\tag{1}$$

`GRIDPOINT` stores all informtion that comes with the input gird. This includes the three components of the position vector of the grid point, the three components of the velocity and the density of the gas at that point. These parameters are the typical output of a hydrodynamical simulation. As one could expect, there is one `GRIDPOINT` object for each point in the gird.

$$\texttt{EVALPOINT evalpoint.}\{\tag{2}$$

`EVALPOINT` represents the so-called evaluation points which store the information on the rays passing through the grid points. An evaluation point is the projection of a grid point on a ray. Since integrations are often done along a ray, the physical quantities are evaluated at the projections of the grid points these rays. Hence the name: evaluation points. The most important parameters stored for an evaluations points are the distance between consecutive points and the projection of the gas velocity along the ray. For every grid point we have a set of rays. Ideally every other grid point lies on one of such rays. As a result we have (maximally) $\texttt{NGRID} * \texttt{NGRID}$ evaluation points.

$$\texttt{SPECIES species.}\begin{cases}\texttt{string sym} & \text{chemical symbol}\\ \texttt{double mass} & \text{molar mass}\\ \texttt{double abn[NGRID]} & \text{abundance}\end{cases}\tag{3}$$

`SPECIES` stores the properties of a chemical species. These include the symbol of the species, its abundance and its molar mass. All this data is read by the function `read_species` in the `read_chemdata.cpp` file.

$$\texttt{REACTION reaction.}\begin{cases}\texttt{string R1, R2, R3} & \text{symbols of reactants}\\ \texttt{string P1, P2, P3, P4} & \text{symbols of reaction products}\\ \texttt{double alpha, beta, gamma} & \text{parameters determining reaction rate}\\ \texttt{double RT\_min, RT\_max} & \text{temperature range where parameters apply}\\ \texttt{double k[NGRID]} & \text{reaction rate at every grid point}\\ \texttt{int dup} & \text{number of duplicates in other T ranges}\end{cases}\tag{4}$$

`REACTION` contains the information on the chemical reactions. There is place for three reactants and four reaction products. For each of those the symbol of the species is stored. Furthermore there are three constants `alpha`, `beta` and `gamma` which will determine the reaction rate, and two temperatures giving the range where these constants apply. All parameters mentionned above are read by the `read_reactions` function in the `read_chemdata.cpp` file. Finally also the reaction rate is stored after it is calculated in the `reaction_rates` function.

2.3 Storing multi-dimensional arrays as lists

All multi-dimensional arrays in the code are stored as one-dimensional lists. On the lowest level, this is the case in every computer code. However, we chose to explicitly write the one-dimensional lists and define the relations between list index and the rows and columns.

3 Ray tracing

To be able to simulate 3-dimensional objects we need a way to represent them in computer language. In this first verison of the code space is dicretized into a truely unstructered grid. In a truely unstructered grid every point is stored separately, without any relation between different points. Physical quantities as the position, gas velocity and temperature are stored independent for every point. In a later version we aim to exploit more the possible structure of the grid.

Space is discretized into `ngrid` grid points.

3.1 Efficiently storing the evaluation points

THE ORIGIN IS NOT AN EVALUATION POINT!
`raytot[RINDEX(n,r)]` gives the total number of evaluation points on a ray `r` through a gridpoint `n`. Here the origin is not counted as an evaluation point. Otherwise we would store the origin each time as an evaluation point, resulting in `ngrid` times `NRAYS` unnecessary doubles. In the radiative transfer part of the code we do want to consider the origin as an eveluation point. Therefore we will systematically add one to `raytot` in that part of the code.

3.2 Equivalent rays

of

4 Radiative transfer

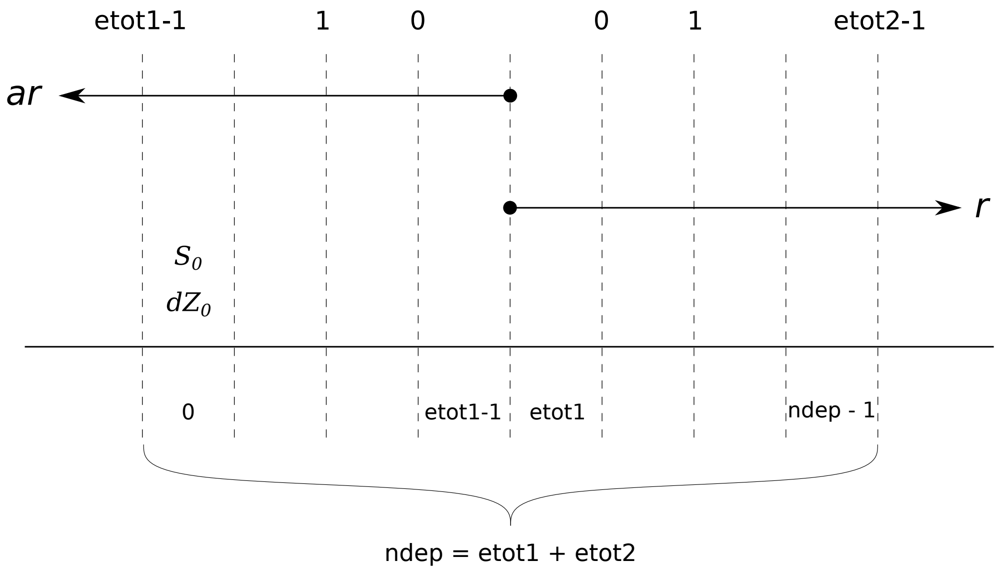

Figure 2: Construction of the ray along wich the radiative transfer is solved.

4.1 `exact_feautrier` solver

Based on the benchmarks, we can conclude that the `exact_feautrier` solver has a 9 digit precission.

References

[1] T. G. Bisbas, T. A. Bell, S. Viti, J. Yates, and M. J. Barlow, “3d-pdr: A new three-dimensional astrochemistry code for treating photodissociation regions,” *Mon. Not. R. Astron. Soc.*, vol. 427, no. 3, pp. 2100–2118, 2012.