3D-RT: 3D Radiative Transfer Code Documentation

Frederik De Ceuster

Abstract

This report gives an overview of some technical aspects of the 3D-RT code. The goal is to motivate why and explain how some things are coded the way they are.

Contents

1 General

3D-RT 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. 3D-RT 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 3D-RT is mainly based on 3D-PDR [1]. The main difference is that in 3D-RT 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 general structures

2.1 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

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

4 Radiative transfer

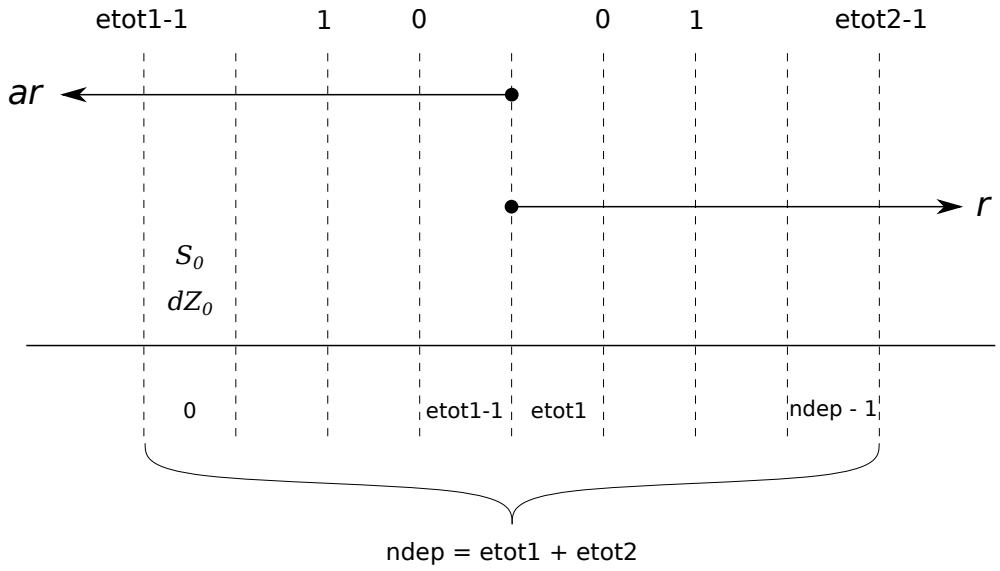

Figure 1: Construction of the ray along wich the radiative transfer is solved.

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.