radmc3dPy v0.26

Attila Juhász

The `radmc3dPy` package consist of four core modules (analyze, image, natconst, setup) for creating input files and post-processing the output of RADMC-3D and an ensemble of models to provide description for the physical variables in the model space as well as a few example models. For a detailed description of all classes and methods see the reference manual (radmc3dPy_refman.pdf).

1 Core modules

1.1 module: setup

This module is responsible mainly for setting up the model, but it also contains some functions to get some information on the existing models.

The setup module contains two separate functions to set up a dust and a gas model possibly independently from each other problemSetupDust() and problemSetupGas(). It depends on the actual model whether or not the are completely independent from each other. Those who are familiar with the IDL package of the predecessor code radmc this module is similar to 'problem_setup.pro'. These setup functions have one mandatory argument, the name of the model. The creation of the input files for radmc-3D is done in a very similar way than in the case of the predecessor code, radmc. A master parameter file, called 'problem_params.inp', contains all input parameters of the model (for the description of the file see Sec. 3). At the beginning of all model setup this file is read. Then any additional keyword argument passed to problemSetupDust() / problemSetupGas() is used to override/change the model parameters as read from the file. If a change/override occurs the input parameter file will be re-written to reflect the parameter setup for which the actual model was run. This way of parameter handling might be convenient when a grid of model is run and one or more of the parameters needs to be changed from model-to-model. Then after the parameters are finally set all necessary input files are created.

1.2 module: analyze

This module is responsible for I/O and some basic analytical functionality. It contains five classes (radmc3dData, radmc3dDustOpac, radmc3dGrid, radmc3dPar, radmc3dRadSources) and several functions. The radmc3dData class is responsible for the reading and writing of the variables (density, temperature, etc.), whatever radmc3dData can read it can also write it, both in formatted ASCII and in C-style binary format. It also contains functions to calculate continuum optical depths and surface density. The radmc3dDustopac class contains methods to read/write the master opacity file (dustopac.inp), read dust opacities and also to calculate them. For the dust opacity calculation it uses the code 'makedust' distributed with RADMC-3D (opac/dust_continuum/jena). The code in opac/dust_continuum/jena should be compiled and the directory should be added to the PATH environment variable such that it can be called/executed without the absolute path. The radmc3dGrid class has methods to create, read and write spatial and wavelength grid and to calculate the grid cell volumes. The radmc3dPar is the model parameter class. It can read and write the model parameter file (problem_params.inp). It has dictionaries to store the variables with the variable names as keys.

Important functions in the analyse module:

readData() - Reads variables (e.g. dust density, gas velocity, etc)

readGrid() - Reads the spatial and frequency grid

readOpac() - Reads the dust opacities

readParams() - Reads the parameter file (problem_params.inp)

writeDefaultParfile() - Writes the default parameters for a model

radmc3dData.getSigmadust() - Calculates the dust surface density in g/cm^2

radmc3dData.getSigmagas() - Calculates the gas surface density in molecule/cm^2

radmc3dData.getTau() - Calculates the continuum optical depth

radmc3dData.writeVTK() - Writes variables to a VTK format for visualisation with e.g. Paraview

1.3 module: image

The image module provides functionality to read/write images and do some simple manipulation of them. The base class is the radmc3dImage class. It can read images both in formatted ASCII and in C-style binary format and write images in FITS format. For the latter it uses the fits module of the astropy package, formerly known as PyFits (radmc3dPy can use any of them). Images can be convolved with an arbitrary 2D Gaussian beam using the imConv method of the radmc3dImage class. The images can be displayed with the plotImage() function, that uses matplotlib to display the image. A coronographic mask can also be simulated with the cmask() function, i.e. within a certain radius around the image center the pixel values will be set to zero. Some convenience/interface functions are also present (e.g. plotImage(), makeImage(), readImage(), etc.).

makeImage() - Calculates an image with RADMC-3D (both dust continuum and channel maps)

plotImage() - Plot the image / channel map

readImage() - Reads the image

radmc3dImage.imConv() - Convolve the image with a Gaussian beam

radmc3dImage.plotMomentmap() - Plots moment map for a 3d image cube

radmc3dImage.writeFits() - Writes the image to a FITS file with CASA compatible header

2 Models

While the core modules contain functionality how to read and write the input files the model modules should contain the rules what the physical structure of the model is, i.e. the distribution of density, velocity, turbulent velocity, etc. as a function of the spatial coordinates. Each model module must be named as 'model_NAME.py' where NAME stands for the name of the model (e.g. model_ppdisk.py). The files themselves must be located either in the current working directory or in the radmc3dPy directory wherever it is installed. Models are always tried to be imported from the current working directory first. Each model module can contain the following functions:

getModelDesc() - contains a one sentence description of the model

getDefaultParams() - contains the default parameters for this model

getDustDensity() - calculates the dust density, and returns as a numpy array

getDustTemperature() - if present it should return the dust temperature as a bumpy array

getGasAbundance() - returns the gas abundance

getGasDensity() - returns the gas density

getGasTemperature() - returns the gas temperature

getVelocity() - returns the velocity

getVTurb() - returns the microturbulent velocity

Each function that should return a variable gets the spatial grid and all model parameters as input arguments and they should return numpy arrays containing the variable.

3 Parameter file

The parameter file ('problem_params.inp') contains all input parameters for the model in an plain text format. Its structure resembles very closely the input parameter file of the predecessor code, radmc. Variables are grouped into blocks according to their meaning, e.g. radiation source parameters, grid parameters, dust opacity, etc. Each parameter in the file is given in the following format:

variableName = variableValue # Comment

The reader method of the radmc3dPar class will try to interpret each line according to this general recipe. However, there is some freedom in this formalism. The variable value can be any expression, also broken into multiple lines. The value expression of a variable can also include another variable defined anywhere *above* that line.

Dust continuum model setup

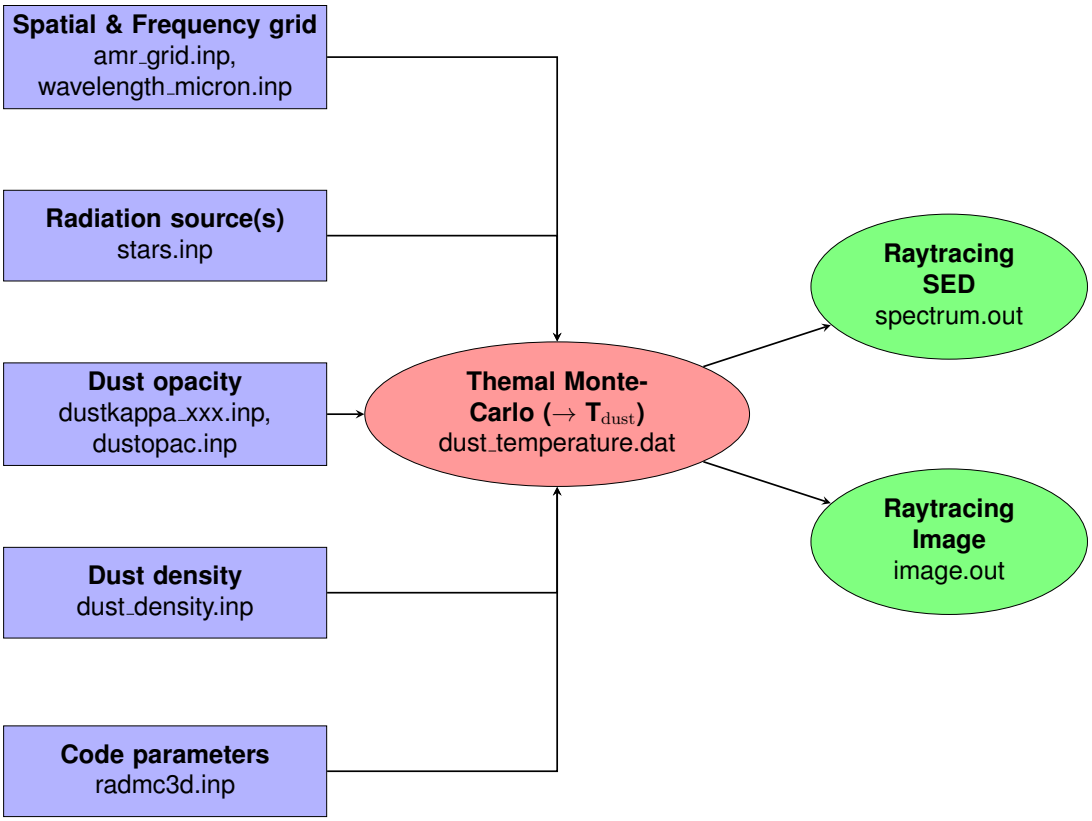

Figure 1: Structure of a dust continuum model. Inputs are marked with blue, intermediate calculation, data products are in red while green marks the output of the simulation.

radmc3dPy commands

First let us create a directory for our model. Then go to this directory and start python.

1 Import `radmc3dPy` .

```
>>> import radmc3dPy
```

2 Check which models are available:

```
>>> radmc3dPy.setup.getModelNames()
['lines_nlte_lvg_1d_1', 'ppdisk', 'simple_1', 'spher1d_1', 'spher2d_1',
'test_scattering_1']
```

3 Create a parameter file with the default values

```
>>> radmc3dPy.analyze.writeDefaultParfile('ppdisk')
```

To change the parameters of the model the two most straightforward possibilities are the following:

a) Open the created 'problem_params.inp' file with a text editor and change the parameters if needed.

b) When in Step 4. the 'problemSetupDust()' method is called one can add keyword arguments with the parameter names, e.g.:

```
>>>radmc3dPy.setup.problemSetupDust('ppdisk', mdisk='0.01*ms])
```

The 'problemSetupDust()' method does the following in this case: Reads the problem_params.inp file. Then overwrites the value of the mdisk parameter and uses that the new value afterwards. It also re-writes the problem_params.inp file with the new values.

NOTE: If the value of the keyword argument is given as a string it will be written as a string unchanged to the 'problem_params.inp' file but will be interpreted and converted to double/float/int within the setup script. I.e. if mdisk='0.01*ms' is given as a keyword argument in the call of problemSetupDust() then in the problem_params.inp file it will appear in the exact same way: mdisk='0.01*ms'. However, if the keyword argument is given as mdisk=0.01*ms the problem_params.inp will contain mdisk = 1.9900000e+31.

4 Set up the model and create all necessary input files.

```
>>>radmc3dPy.setup.problemSetupDust('ppdisk')
```

Then we need to copy the dust opacity file called 'dustkappa_silicate.inp' from the python_examples/dataf directory within the distribution root directory to the current model directory.

5 Then run RADMC-3D from the shell with the Monte-Carlo simulation to calculate the dust temperature.

```
$>radmc3d mctherm
```

Alternatively we can also make a system call from within Python, e.g.:

```
>>>import os
>>>os.system('radmc3d mctherm')
```

6 After the thermal Monte-Carlo run has finished we can make an image from within pyton.

```
$>radmc3dPy.image.makeImage(npix=400, sizeau=200, wav=880., incl=45, posang=43.)
```

7 After RADMC-3D finished we can read the image and plot it.

```
>>>imag=radmc3dPy.image.readImage()
>>>radmc3dPy.image.plotImage(imag, arcsec=True, dpc=140., log=True, maxlog=5)
```

Here 'arcsec=True' sets the image axes to arc second that also requires the knowledge of the distance, which is set in parsec by the 'dpc=140.' keyword. The 'log=True', sets logarithmic stretch of the image. The 'maxlog=5' sets a clip of the displayed image by 10^{-5} below its maximum value, i.e. the displayed image values will be between max(image) and max(image)*10^{-5}.

8 We can also convolve the image with an arbitrary elliptical gaussian Gaussian beam

```
>>>conv_imag = imag.imConv(fwhm=[0.05, 0.1], pa=40., dpc=140.)
```

The fwhm of the Gaussian beam should be in arcsec, the positing angle of the major axis of the beam ellipse should be in degrees, and the distance to the source in pc (dpc keyword) should be given.

9 We can also write the image into a fits file:

```
>>>imag.writeFits(fname='image.fits', dpc=140., coord='03h0m0s -29d0m0s')
```

Gas model setup

radmc3dPy commands

1 Follow the instructions for the dust models from Step 1. to 5.

6 Create the necessary input files for the gas simulations:

```
>>>radmc3dPy.setup.problemSetupGas('ppdisk')
```

7 Calculate a channel map at a single frequency/wavelength

```
$>radmc3d image npix 400 sizeau 200 incl 45.  phi 0.  posang 43.  iline 3 vkms 1.0
```

6 This command calculates a single channel map at

```
>>>imag=radmc3dPy.image.readImage()
>>>radmc3dPy.image.plotImage()
```

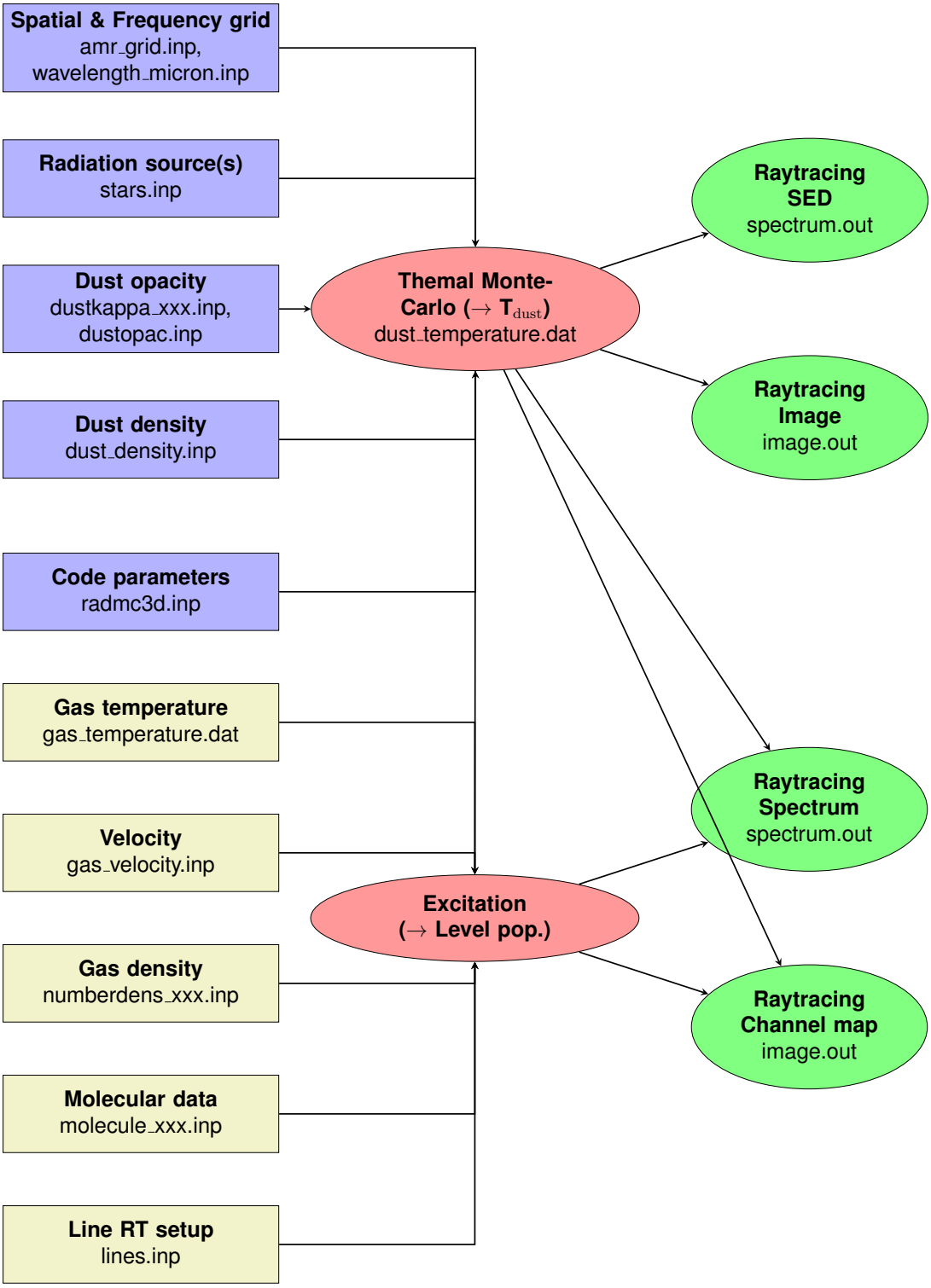

Figure 2: Structure of a dust + gas model. Dust inputs are marked with blue, gas inputs are marked with yellow, intermediate calculation, data products are in red while green marks the output of the simulation.