Manual for RADMC
Version 3.1
Manual Version 3.1-2

C.P. Dullemond

Chapter 1

Overview and Quick-Start

1.1 Introduction

The RADMC package is a code package for doing calculations of continuum radiative transfer in 3-D axisymmetric (i.e. effectively 2-D) circumstellar dust configurations around an illuminating central source (e.g. a star).

This package is not public domain software. Every use of this code package, or any part of it, is only to be done with explicit permission of the author, C.P. Dullemond.

This package actually contains *two* main codes, and a number of smaller ones. The main radiative transfer code is a Monte Carlo code, which is the actual code called RADMC. The algorithm that is used for the Monte Carlo radiative transfer is an enhanced version of the algorithm of Bjorkman & Wood (2001, ApJ, 554, 615). This code will compute the dust temperature and the scattering source function everywhere. It will also produce tentative (but noisy) spectra at various inclinations, but they are often too noisy to be used. To produce smooth spectra (SEDs) and/or smooth images one does a post-processing with a ray tracing code called RAYTRACE, also part of this package. There are also a couple of codes in this package which have support purposes but which are of less direct importance.

In addition to the codes this package also contains a number of example models, which are in fact so general that they can be used as black-box ready-to-use models for various purposes (although we advise against the purely black-box use of these models; we prefer to call these models: template models).

This manual explains first how to get quick results using these template models, just so that you can get used to the model and its output. Then the model explains all the input files and output files of the code, including tips how to use them. The manual will also explain more details of the various template models.

1.2 For which kind of modeling can this code be used?

This code package is meant for continuum radiative transfer of dust configurations that are axisymmetric and mirror-symmetric in the equatorial plane. Typically they are illuminated by a central stellar source, but this is not necessary. They can also be illuminated by external radiation (interstellar radiation field). And finally, the code also allows for internal heating in two ways: a) for galaxy simulations by a smooth population of stars and b) for disk simulations by internal dissipation of heat. In conjunction with another code (PAHCODE), the RADMC also allows for inclusion of quantum-heated grains such as PAHs.

Typically the code is used for modeling protoplanetary disks, circumstellar envelopes, post-AGB stars, planetary nebulae etc.

1.3 Requirements

This package runs under linux/unix/MacOSX, but has not been tested under Windows. The following pre-installed software is required:

- A F77 compiler
  Preferably the GNU g77 compiler (which the current installation assumes is present on the system). Web site:

http://www.gnu.org/software/fortran/fortran.html (NOTE: Mac users can install g77 via the fink installation tool, see http://finkproject.org/). But g95 also works. Web site: http://www.g95.org/. Other compilers may work, but have not been tested yet.

- The IDL package (Interactive Data Language)
  IDL is a software package similar to MatLab, and it is not free. The website for IDL is: http://www.ittvis.com/. If IDL is not present on your system, and your system administrators cannot install this package due to lack of funding, you can use an open source clone called GDL (Gnu Data Language) which can be readily downloaded from the web. This GDL package misses some libraries and features, but the RADMC code can be used with GDL.

Note that the Monte Carlo code RADMC itself is in Fortran. Only the creation of the input files (and hence the problem definition) and the analysis of the output files is done in IDL. The user is of course invited to use other ways to create the input files for RADMC if he/she is not able to use IDL nor GDL. Therefore IDL/GDL are not strictly required for the use of this code.

1.4 The package

1.4.1 The run_* directory(ies)

The package contains one or more directories starting with `run_`. These are templates for models that can be calculated with this package. We will describe them in detail below. But a few remarks beforehand:

- It is advisable to keep the current template intact. To experiment with RADMC, just type e.g. `cp -r run_1 run_1_mytest`, and then modify the files in that directory.
- The philosophy of this package is that each run_* directory contains *one* model run. So if you want to make different models, the way to do this is to make different directories. For instance, if one wants to make three models of T Tauri disks, one could make the directories run_ttauri_1, run_ttauri_2 and run_ttauri_3.
- Each run directory contains a series of IDL program files, named problem_*.pro (where * stands for the various different files). These form a setup package that writes out the RADMC input files. Of course the user can also create these RADMC input files him/her-self using other software. *These IDL routines are only there to give the user already a reasonably start, but they do not strictly belong to the core RADMC package!!!* The user is allowed (and even expected) to modify these IDL routines to suit his/her needs. But in case the user wants to simply use the current setup without modifications, then the user will simply (and only) edit the file problem_params.pro, which contains a list of model parameters.

1.4.2 The source directory

The directory `source/` contains the sources of the actual code package. The `source/` directory contains the following subdirectories:

- radmc/
  The main code RADMC. This is the Monte Carlo code that computes the dust temperature everywhere and also computes the scattering source functions (for isotropic scattering).
- raytrace/
  The main code RAYTRACE. Once the RADMC code has finished, spectra and images can be created using this ray tracer called RAYTRACE. This used to be part of the RADICAL code, which is, however, now not used anymore.
- chopdens/
  Some disky dust density configurations can be so extremely optically thick that the code is a bit slow. By setting a maximum optical depth (by writing a file chopdens.inp with a certain format) and calling the chopdens routine, the density configuration can be moderated such that a bit of mass is removed from the regions of highest optical depths. This should not affect the optical/infrared appearance of the disk too much, but one must be careful with this method, and always check the final results against a run in which this chopping is not used.

- idlroutines/
  Some of the analysis of the results of the RADMC and RAYTRACE codes can be done best with some subroutines in IDL, which are located here for the convenience of the user.

- makeopac/
  (At present not used!)
  The makeopac routine is a simple code for Mie calculations based on some lists of optical constants which are also located in this directory.

- pahcode/
  RADMC itself cannot deal directly with PAHs (i.e. with quantum heated grains). But it can be made to cooperate with the PAHCODE, which can deal with PAHs. This code reads in information from RADMC about the energy absorbed by the PAH molecules, and computes therewith the excitation of these molecules, which can then again be read into RADMC to make sure that the other (thermal) dust grains also 'see' this emission. Finally RAYTRACE will then read in both the data for the thermal grains as well as for the PAH molecules and produces a spectrum. **NOTE: This is not yet tested with RAYTRACE. Only for RADICAL this is tested. Will be tested later.**

The compilation of all these codes is organized in each of the run_* directories, and we will come to this lateron.

1.4.3 The bin directory (will be created)

Once the compilation is done there will also be a bin/ directory. This will then contain all the binary codes of the programs in the source/ directory.

1.5 Compilation

1.5.1 How to compile the codes

The codes must first be compiled before use. But since these codes are still programmed in Fortran-77, which has fixed array sizes, the compilation may/will depend on the problem at hand. Therefore the compilation is fully organized within each run_* directory. So suppose you want to run the model in the run_1 directory, here's how to compile the codes accordingly:

```
cd run_1
idl
IDL> .r problem_compilecodes
```

where the `IDL>` stands for the IDL prompt (and therefore this `IDL>` should not be typed; only the text behind it). The execution of the problem_compilecodes should compile all codes, create a bin/ directory if this does not yet exist, and put all executables in that directory. It should end with a message

```
  ====== ALL COMPILATIONS SUCCESFUL ======
```

if all compilations worked well. If problems occured during compilation, then this message will not appear. Check your compiler (currently set to g77) and if necessary change this in the makefile or Makefile in the source directories of the codes in source/ directory. Once the compilation is done, you can either stay in IDL, or exit:

```
IDL> exit
```

Note that these compilations are done such that the array sizes of the Fortran-77 codes are all precisely consistent with the problem that is defined by the problem_params.pro file. In particular the following parameters will affect the array sizes, and hence the compilation of the code (at least for the particular setup in run_1):

nr	Nr of radial grid points
nt	Nr of theta grid points
fresmd	An index of the frequency resolution mode used (see later)
ab_ab0	Nr of elements of this array determines nr of dust species

1.5.2 The automatic creation of a bin directory in home

It can be very useful to be able to use radmc and raytrace directly from the prompt without having to write the path of the code. For that reason the compilation routines will make sure that there is a bin/ directory in your home directory in which two little scripts are placed called radmc and raytrace which in fact only link to the most recently compiled version of these codes. If you put the /bin in your tcsh or bash shell (for Linux or MacOSX) then you will be able to simply type `radmc` or `raytrace` in the shell without a path.

Caution: If you have multiple versions of the code or multiple compilations for different models, and you want to make sure your model uses the local copy of these codes, then the fulll path is necessary.

1.6 A quick 'howto' for the template model(s)

The complete code is quite complex, so it is probably best to start simply from the template models and try to understand their workings. A first result can be obtained in the following way (including the compilation which is already described above):

```
cd run_1
idl
IDL> .r problem_compilecodes.pro
IDL> .r problem_setup.pro
IDL> exit
nice ../bin/radmc
nice ../bin/raytrace spectrum incl 45
idl
IDL> .r ../sources/idlroutines/analyze.pro
IDL> s=read_spectrum()
IDL> plot_spectrum,s
```

This should compile all codes, set up the model, run the Monte Carlo code RADMC, run the ray tracing code RAYTRACE, read the spectrum and plot the spectrum on the screen.

Note: It can be convenient to set the IDL_PATH in the csh or tcsh (or equivalently in bash) to the ../source/analyze.pro path, so that one can type `.r analyze` instead of `.r ../sources/idlroutines/analyze.pro`. That is easier.

1.7 Warning

The author of this code (C.P. Dullemond) does not take any responsibility for the use of this code. Codes of this kind are quite complex and despite many rounds of testing there can always appear unexpected problems. The most common problems are a) wrong use of the code, b) bugs introduced by the author after modifications of the code (such as incompatibilities of new options with other older options, or simply that new options have not yet been tested well enough) and c) use of the code in ranges of parameter space where it has not yet been tested well enough. Problem a) can be only avoided by thorough reading of this manual, regular testing of the code and self-checking ('do I really understand what this option really does? How can I test that I indeed do?'). The danger of problem b) can be suppressed by always testing any new version of the code against older versions on the same test problems. Problem c) cannot really be 100% avoided, but a careful checking of the check-lists of Chapter 5 are a good start. Of course there is also the risk that there are still undiscovered bugs in the code. All of these problems are best avoided by *thoughtful, patient and careful modeling, with lots of checks and tests*. Typically a modeler should spend a serious, perhaps even dominant part of his/her time on checking the results ('can this result be correct?', 'how can I check that this result is correct?', 'when I change *this*, then one should get *that*...' etc).

In spite of these sombre notes, we wish the user lots of fun and good results with this code. Should there be problems or bug reports, please contact Cornelis Dullemond (`dullemon@mpia.de`).

Chapter 2

The codes RADMC and RAYTRACE

The core of the code package is the code tandem RADMC and RAYTRACE. The first is the actual Monte Carlo code that does the real work. The second is a post-processing tool to create spectra and images. The input files to these codes are mostly the same for both codes, with the exception of the file radmc.inp which is only for RADMC and raytrace.inp which is only for raytrace. The template models (the problem_*.pro IDL routines in the run_* directories) create all the input files for the user, and the model parameters in those files (in particular in the problem_params.pro file) are parameters of the model but not always of the codes.

In principle you can simply use the template models and modify them to your needs, and not be too much concerned with the input files they create and which are the true input to RADMC and RAYTRACE. But our advice is never to use the template models as black boxes. It is therefore important to understand how the codes work, which options they have and how their input files are structured.

2.1 The input files for RADMC and RAYTRACE

Here is a list of input files. Most files are input files for all codes. But each code also has one file specifically for that code to set options in that code.

2.1.1 General input files for all codes

- radius.inp
  A list of radial grid points, in centimeter. Space is mapped in spherical (polar) coordinates: (R, Θ), and this file contains the list of radial points. The first line gives the nr of grid points, then the list follows in ascending order. The best is to arrange the radial grid points logarithmically spaced, so that all scales of the problem are mapped. But near the inner edge the grid points typically need to be refined so as to make sure that the innermost cell is opticall thin.

- theta.inp
  A list of Θ grid points of the spherical coordinate system. First line is number of theta points and a dummy value which should be 1. Then the list of theta points comes. Note that $\Theta = 0$ means the polar axis while $\Theta = \pi/2$ is the equatorial plane. RADMC assumes mirror symmetry in the equatorial plane, so the coordinates should range between 0 and $\pi/2$, in ascending order (from pole to equator). Note that the Θ coordinates should not start exactly at 0, but a bit away from the pole. Also never make the last grid point exactly at $\pi/2$ but a tiny bit smaller than that. In configurations wich dense regions near the midplane (such as a protoplanetary disk or so), it is prudent to choose the Θ coordinates more finely spaced near this midplane than near the pole, so as to resolve the disk well. Moreover, it is useful to make sure that the fine spacing in Θ covers the disk up to where the optical depth *for radially outward moving stellar photons* is less than zero.

- frequency.inp
  A list of frequency points ($\nu \equiv c/\lambda$) in which the radiative transfer should be done, in Hertz. First line is the number of frequency points, then the list is given. This should be a list in ascending order in Hertz, and should cover both the wavelength domains in which the central illuminating source (the central star) radiates and the wavelength domains in which the dust radiates. In practice this means it should span from about 0.1μm to about 1000μm or longer wavelengths, meaning that ν ranges from 3×10^{11} Hz to 3×10^{15} Hz. A logarithmic

spacing is necessary to cover such a large range in frequencies. If required, one can choose refined frequency spacing in regions of particularly interesting (dust) features. Note that all radiative tables will be using this frequency grid (stellar spectrum, dust opacities, output spectrum).

- starspectrum.inp
  A list of two columns. First line is number of frequency points (which should match *exactly* the number of points in frequency.inp. First column is (again) the list of frequencies in Hz. These values should match *exactly* the values of frequency.inp. Second column is flux F_ν of the central illuminating source (central star) in units of erg/cm^2/s/Hz, as seen by an observer at 1 parsec distance. This file is the *input spectrum* of the problem: the spectrum of the central source.

- starinfo.inp
  A short file containing the parameters of the central star. First line: format number (=1), second line: radius of the star in cm, third line: mass of the star in gram (not used in RADMC), fourth line: effective temperature of the star in K (not used in RADMC, unless the starspectrum.inp file is not present).

- dustopac.inp
  The main control file for the dust opacities. RADMC can allow for multiple dust species that coexist at the same (or different) location. For each of these dust species an opacity must be given. The dustopac.inp file assigns which dust component gets which opacity. First line is a format number (which should be 2 in this version). Second line is the number of dust species (which must match the number of dust species in the file *dustdens.inp* below). Then a comment line (line 3) and then line 4: for the first dust species a control number telling how to read in this dust opacity (take it to be -1). Line 5 tells whether this is a normal grain (0) or a quantum-heated grain (¿0, but see Section 2.7). Then line 6 gives a number which points to the dust opacity file assigned to dust component 1. If this number is '1', then this points to the file dustopac_1.inp for this component. Finally there is again a comment line (line 7). Lines 4,5,6,7 are repeated for the next dust species (lines 8,9,10,11) in case the second line says '2' or higher. If the second line says 4, then lines 4,5,6,7 should be 4$\times$ repeated, but of course pointing to another dust opacity file each case. In this way each dust component is assigned to an opacity.

- dustopac_1.inp, dustopac_2.inp etc
  The dust opacity files. First line, first number: the number of frequency points which must match *exactly* the number in frequency.inp. Let us denote this number `nf`. Second number: a dummy that must be 1. Then a list of `nf` absorption opacities, in units of cm^2/gram (note: cross section per gram-of-*dust*). Then again a list of `nf` numbers: the scattering opacity, also in units of cm^2/gram. Note that scattering is, in this implementation, assumed to be isotropic scattering. This is not a perfect assumption, and it can be relaxed, but this is a bit complex and is not supported in this distribution yet.

- dustdens.inp
  The dust density distribution file. First line, first number: the number of dust species (must be *exactly* equal to the number of dust species in dustopac.inp. First line, second and third number: the number of radial and theta points respectively (which must be *exactly* equal to the number of points in radius.inp and theta.inp respectively. First line, fourth number: dummy, take it 1. Then follows a list of density values, in units of gram/cm^3 of dust. Inner loop: theta coordinate, from pole to equator, middle loop: radial coordinate, outer loop: index of dust species. If the number of dust species is, say, 2, then the outer loop goes from 1 to 2. The opacity belonging to these two dust components are given in dustopac.inp (see above). One sees that it is easy to have two different dust species simultaneously at the same location. That is perfectly fine, and can be useful to simulate various dust sizes coexisting at a same place, or different dust species coexisting at the same place. A few notes:

  - In older versions of RADMC the dust did not only have different species, but each species could have a powerlaw size distribution. In the code one therefore often finds a loop variable for species (`ispec`) *and* for size (`isize`). While this may still work, it is never used and it is not particularly useful. So if using size distributions, the user is advised just to take each size as a different dust species.
  - Each dust species is thermally decoupled from the other species. Each dust component therefore has its own dust temperature.
  - If the user wishes to model different dust species coexisting, one should ask oneself the question: do these different species in reality thermally decouple or are they (through a bit of coagulation perhaps)

thermally coupled? If the latter is the case, then one might prefer to mix the opacities beforehand, and treat them as if they are a single dust species. But this has the disadvantage that one has a globally constant mixing ratio of the components. If one wants to have the possibility to specify the density distribution of each dust component individually, BUT still wants thermal coupling (i.e. a single temperature for all dust species located at the same point in space), then one can set this with the itempdecoup parameter in the radmc.inp file (see below).

- external_meanint.inp
  If the model cloud is cool enough that the interstellar radiation field becomes an important heating mechanism, then this file is used to tell the codes that there is this interstellar radiation field, given in the form of a mean intensity. Note of caution: using this mode requires a good thought of the outer radius of the grid. Take it too big, and most of the CPU time is crunched on photons that enter and leave the grid without having had interactions with the model cloud.

- stellarsource.dat [only useful for galaxy simulations]
  If one wishes to model axisymmetric configurations of dusty galaxies, then use this file to put in smooth populations of stars as input of radiation into the system. Must compile codes with INCLUDE_EMISLOC to activate.

- quantsource.dat [only useful for PAH or other quantum grains]
  Input of source term from the PAHs or other quantum-heated grains. See Section 2.7 for details.

- qplus.inp [only useful for active accretion disks]
  For modeling active accretional dissipation of heat in the disk one can use this file to specify this heat insertion at each location. *This mode is not yet well tested, and it is very slow*.

2.1.2 Program-specific control input files

Here is a list of control files special for each specific code.

- radmc.inp [for RADMC code]
  Main control parameter file for RADMC. In Version 3.1 and higher of this package this file is a name list type (see Section 2.1.3 for an explanation of how to format such files). But it remains compatible with the old style (it simply checks if the first character is a 0-9, meaning old style, or a-z, meaning namelist style). The namelist variables are:

  - nphot: (default = 100000)
    The number of photon packages for the simulation. See Bjorkman & Wood (2001) for explanation. Of course, the more the better, but it makes the code slower.
  - iseed: (default = -17933201)
    A seed value for the random number generator.
  - imethod: (default = 2)
    An index for which method should be used for the Monte Carlo simulation.
  - ifast: (default = 0)
    If 1, then the recalculation of the dust temperature is not done always, but only when necessary. A bit less accurate, but faster. **WARNING:** There may be still a bug in the code when using ifast=0 but with the new super-efficient RADMC method. Consult author in case of doubt... (08.07.07)
  - enthres: (default = 1.d-2)
    If ifast.eq.1, then this gives the threshold for the energy increase in a cell before a new dust temperature is calculated. Could be taken to be 0.01 or so.
  - cntdump: (default = 10000000) [old default was 10000]
    After this many photons a safety dump is made. This was useful in the past, when the code was still so slow that it could take days before an answer. Safety dumps were useful, but it is now not important anymore. The high default value makes that it will not be used unless for extremely large photon numbers.
  - irestart: (default = 0)
    Again, in the past it was useful to be able to continue from a safety dump. By putting this to 1 it would do that.

- itempdecoup: (default = 1)
  If 1, then all dust species have their own dust temperature (this is default). If 0, then their temperatures are all locally coupled (but may still vary in space of course).
- iquantum: (default = 0)
  If quantum-heated grains are present, then you can put this to a value that is non-zero (depending on the way you wish this to be treated). If this value is 0, then, even if the dustopac.inp files says that one or more of the dust opacities belongs to a quantum-heated grain, the quantum heating will nevertheless be inactive. For more details about the quantum heating, see Section 2.7.
- istarsurf: (default = 1) [old default was 0]
  If 1, then the star is not treated as a point source, but treated as a true 3-D sphere. If 0, then the star emits light as a point source.
- nphotdiff: (default = 0)
  RADMC has a module for treating the low photon statistics near the midplane of a very optically thick disk. This is done using a diffusion algorithm. The number nphotdiff gives a limit such that if a cell is visited by fewer than this number of photon packages, it will participate in the diffusion trick and its temperature will be calculated using this diffusion recipe. See Section 2.5. If this value is 0, then the diffusion mode is not active and the full photon noise is present.
- errtol: (default = 1d-10)
  The tolerance for the error in the diffusion algorithm. See Section 2.5.
- nvstr: (default = 0)
  Apart from just doing continuum radiative transfer, this new version of RADMC also allows an iteration on the vertical hydrostatic pressure balance equation. If this value is set to 0 (default), then no such iteration is done, and the radiative transfer is done purely on the given input density field from dustdens.inp. If ¿1, then iterations are done, such that the vertically integrated density from dustdens.inp remains the same at each radius, but after the first Monte Carlo run the vertical density structure (along lines of varying Θ but constant R) is computed using the computed temperature profile. See Section 2.6.
- vserrtol: (default = 0.d0)
  Error tolerance for the vertical structure iteration. If 0.d0 (default) then the code will simply iterate nvstr times, period. **IMPORTANT:** Since photon noise usually spoils the perfect convergence, if one takes vserrtol > 0 then one should not take this too small.
- ivstrt: (default = 1)
  Index of dust species, the temperature of which is taken to be representative of the gas temperature for the vertical structure iteration.
- ntemp: (default = DB_NTEMP_MAX; see configure.h)
  Since the new algorithm of RADMC tabulates all the thermodynamic quantities in tables, this gives the size of this table. It can not exceed the DB_NTEMP_MAX hard compiled limit (see configure.h in the sources/radmc/src directory).
- temp0: (default = 0.01)
  The lower limit of the temperature in the temperature table mentioned above.
- temp1: (default = 1d5)
  The upper limit of the temperature in the temperature table mentioned above.

- raytrace.inp [for RAYTRACE code]
  A control file for the ray tracing program RAYTRACE. In Version 3.1 and higher of this package this file is a name list type (see Section 2.1.3 for an explanation of how to format such files). But it remains compatible with the old style (it simply checks if the first character is a 0-9, meaning old style, or a-z, meaning namelist style). The namelist variables are:
  - nrphiinf: (default = 32)
    For circular images and for spectra: the number of pixels per pixel circle.
  - nrrayextra: (default = -20)
    For circular images and for spectra: the absolute values of this number denotes the number of radial concentric pixel circles with impact parameter smaller than the inner radius of the model grid (excluding the possible refinement done with nrref).

  - imethod: (default = 0)
    If imethod=0 then only do the nrrayextra rays linearly spaced inward of inner grid radius. If imethod=1 then also add refinement of rays close to inner grid radius. This is useful (even crucial) for very optically thick disklike configurations with inner edge at the inner grid radius seen nearly (but not exactly) face-on. If imethod=1 then also check out the nrref (a good value is nrref=10).
  - nrref: (default = 10)
    Has meaning only if imethod=1. This tells how many refinement steps are done inward of the inner rim.
  - dbdr: (default = 1)
    The number of rays (i.e. the number of pixel circles) with impact parameter similar to one of the grid points. Normally we have 1 ray (pixel circle) per model radial grid point. If dbdr=2 then we have two per model radial grid point, meaning we have a better spatial resolution.
  - inclination: (default = 45)
    The default inclination for a spectrum, image or circular image if the inclination is not specified on the command line.

  See Section **??** for an explanation of the pixel arrangements for circular images and for spectra (i.e. the true meaning of nrphiinf, nrraysextra, imethod, nrref and dbdr.

- vstruct.inp [for RADMC code]
  If vertical structure iteration is used (see radmc.inp below and Section 2.6), then this file tells which of the dust species participates. First line: format number. Second line: number of dust species in total (must be *exactly* equal to number of dust species in dustopac.inp). Then follows a 0 or 1 for each dust species.

- aperture.inp [OPTIONAL for RAYTRACE code]
  NOTE: This is an *optional* file. If it does not exist, then no aperture is chosen. This file contains a list of aperture sizes for each frequency (same frequencies as in frequency.inp). First line: the number of frequency points (must be *identical* to that of frequency.inp); then follows the aperture for each frequency measured as *radius* in *arcsec*. Note that the aperture simply means that for the SED all emission coming from a distance larger than the aperture value from the center will be ignored. Apart from using the aperture.inp file you can also set a constant aperture (for all wavelengths the same) by including `aperture 10 arcsec` or `aperture 100 AU` in the shell call of RAYTRACE, see Section 2.2. *Important note:* If you do not specify the distance to the source when computing a spectrum (for instance if you type `raytrace spectrum incl 45`) then the distance to the source is automatically scaled to 1 parsec for the computation of the aperture stuff. In this case 1 AU == 1 arcsec. If instead you specify the distance using `raytrace spectrum incl 45 dpc 100`, then that distance (100 parsec) is used for the computation of the aperture stuff. *the output spectrum spectrum.dat is always normalized to a distance of 1 parsec, no matter what distance you type in the command line.* So the spectrum in spectrum.dat must always be scaled from distance = 1 parsec to the right distance.

- chopdens.inp [for CHOPDENS code]
  A control file for the ray tracing program RAYTRACE. In Version 3.1 and higher of this package this file is a name list type (see Section 2.1.3 for an explanation of how to format such files). But it remains compatible with the old style (it simply checks if the first character is a 0-9, meaning old style, or a-z, meaning namelist style). The namelist variables are:

  - taumax: (default = 1d6)
    The maximum allowed vertical optical depth toward the midplane of the (presumably) disk. The smaller value this is taken the stronger the density distribution is affected (bad) but the faster the RADMC Monte Carlo code is (good). A good balance between accuracy and speed is paramount. The default value at 1d6 means that it typically never does any smoothing. A value of 1d4 would mean that it only smoothes when the disk becomes rather heavily optically thick. Values much below 1d4 make the code very fast but the results inaccurate.
  - lambda: (default = 0.55)
    The wavelength in μm at which the vertical optical depth is measured.
  - smooth: (default = 1.0)
    The index by which the chopping is done smoothly. The lower this value the more gradual the chopping goes. The higher, the more abrupt it goes.

2.1.3 Namelist input files (radmc.inp, raytrace.inp, chopdens.inp)

Some of the input files have a free-format namelist style of input, namely:

1. radmc.inp
2. raytrace.inp
3. chopdens.inp

These are the files that contain the general program control parameters of the programs RADMC, RAYTRACE and CHOPDENS respectively. See Subsection 2.1.2 for a description of their input variables. In earlier versions of these codes these files were simply lists of numbers, and the user simply had to *know* which number means what. Now these files are namelists. An example of a namelist is (in this particular case the example is raytrace.inp):

```
; This is a comment line

nrphiinf     = 32      ; Some other comment, whatever you like
nrraysextra  = -20     ; Bla bla

; Here are some new parameters

imethod      = 1
nrref        = 10      ; Could also be 20, but why not 10
```

(which is in fact an example namelist for the file raytrace.inp, but that is irrelevant here as we are speaking about namelist files in general). As one can see, it is a list of <variable name> = <value> lines, more or less in free format. The ";" character denotes the start of comments and blank lines and spaces are ignored. The above file could also be written as

```
nrphiinf=32
nrraysextra=-20
imethod=1
nrref=10
```

with exactly the same meaning, as long as each line contains at maximum one identity. If a variable is *not* listed, then the code-internal default values are used. It is therefore not obligatory to list each parameter with a value. For instance, since `nrref` has per default the value of 10 in the code RAYTRACE the above input file could also contain just the first three lines and it would act precisely the same. If you choose the same values as the code-internal default, then you can choose to leave out that variable, but you can also leave it in if you give it precisely that value, as in the example above. An empty file would therefore just force the code to use all parameters as their default, and so would it if the file is non-existent altogether.

2.2 Command-line (csh/bash) options for RAYTRACE

. The program RAYTRACE is a program that accepts command-line options. In this version the program is written in F77 which does not support command-line options, so the actual raytrace program is in fact a PERL script that calls raytraceprog, which is the actual F77 program. The command line options are put as commands in a file called command which is then read by raytraceprog. But these are technical details. In effect one can consider raytrace as a program with command-line options. The first word after raytrace tells RAYTRACE what is should actually calculate:

- `raytrace spectrum`
  Makes an SED. See Section 2.9 for details. Further command-line options:
  - The inclination can be set by adding `incl 30` to the command line (i.e. in total: `raytrace spectrum incl 30`). If the inclination is not set, then the default inclination is used which is set in raytrace.inp, and if it is not set there, then the default inclination of the code is used.

  - One can also set an aperture (constant with wavelength) by adding `apert 10 arcsec` or `apert 100 AU` (to name a few possibilities). But beware that in most practical purposes one must set the aperture as a function of wavelength by adding a file aperture.inp. NOTE: the aperture is measured as *radius*. If you set the aperture, then one must also specify the distance to the source in parsec: dpc 100 (for 100 parsec). Note that from now on in RAYTRACE the dpc option will only affect the conversion of arcsec to cm in the comptutation of the aperture, but the spectrum normalization in the output file spectrum.dat will *always* be normalized to a distance of 1 parsec (also if the source itself is larger than 1 parsec; it is just a scaling factor).
  - By adding `nostar` you obtain a spectrum without the star spectrum, i.e. only the emission from the circumstellar matter will then be computed.

- `glob`
  Makes a series of spectra at increasing inclination with the aim of testing luminosity conservation. See Section 2.9.2 for details.
  - By adding `nincl 45` we specify that 45 regularly spaced inclinations between pole-on and edge-on are to be taken for this mode.
  -

- `raytrace image`
  Makes a rectangular image. See Section 2.9 for details. Further command-line options:
  -

2.3 The output files of RADMC

The RADMC code produces the following output files:

- dusttemp_final.dat
  This is the file containing the temperature of the dust as a function of location in the disk. First line, first number: the number of dust species; second number: nr of radial grid points; third number: nr of Θ points from pole to equator; fourth number: dummy. Then follows a data block for each dust species. Each data block starts with a first line with one number: this will for all practical purposes described here be 1. Then follows a list of numbers giving the temperature of this dust species at the grid points. The outer loop is ir=1,nr (radial grid point); the inner loop is it=1,nt (Θ-grid point).

- spectrum_all.dat
  This file contains rough estimates of the spectra as seen at a large distance at inclinations equal to the Θ_i grid points. The way these spectra are created is to simply collect the escaping photons (which eventually each photon will do) at $R = \infty$ at the various inclinations. The Θ-bins in which these photons are collected are bounded by the Θ_i grid points. So where the Θ_i grid is refined, the spectrum will have larger photon statistics noise, because the collection area is smaller. Simply put: the larger the collection area at infinity, the less photon noise, but the more uncertain which true inclination this spectrum belongs to. The spectrum_all.dat contains all these spectra between all Θ_i. First line, first number: nr of frequencies; second number: nr of Θ_i grid points. Then follows a number of data blocks, the number of which equals the nr of Θ_i grid points. Each data block contains two colums: first column is the frequency in Hz; second column is the spectrum as seen at a distance of 1 parsec in erg/s/cm^2/Hz.

- scatsource.dat
  The scattering source term. In this version of RADMC only isotropic scattering is included. That means that for each position (ir,it) and each frequency (inu) the Monte Carlo code must store how much light is being scattered per second in all directions. (This shows why only isotropic scattering is allowed at the moment: if we also include the angle (imu,iphi), then the scatsource.dat file would kill your hard disk). First line, first number: nr of frequency points; second number: nr of radial grid points; third number: nr of Θ grid points; fourth number: dummy. Then follows a table of scattering source function, with outer loop being frequency, then a Θ loop and then (as inner loop) a radius loop. The dimension of the source function is erg/cm^3/s/Hz/ster.

- photstat.info
  This gives for each (ir,it) position how many photon packages have visited that cell. Useful for detecting possible locations of bad photon noise. First line: format number (=1). Second line: nr and nt (radial and Θ grid size). Then the table with it in outer loop and ir in inner loop.

- dustdens.inp
  If (and only if!) the vertical structure mode is switched on (nvstr$>$0 in the radmc.inp file), then the new dust density structure is written in this file. See input files for the format of this file.

Note that for most part the output files of RADMC are meant as input to RAYTRACE (unless the dust temperature or dust density following from this code are used as input to other software you have created).

2.4 The output files of RAYTRACE

The RAYTRACE code produces the following output files:

- spectrum.dat [only after `raytrace spectrum` command]
  This is the spectrum (SED) of the object at either the default inclination or the inclination mentioned in the command line. For instance, after typing `raytrace spectrum incl 60`, the spectrum is at inclination of 60 degrees (measured from the pole). The file format is the same as that of starspectrum.inp. First line is number of frequency points. Then follows a data block. First column is the list of frequencies in Hz; they are identical to those of the frequency.inp file. Second column is flux F_ν of the complete object in units of erg/cm^2/s/Hz, as seen by an observer at 1 parsec distance. Note that one can also implement an aperture, see Section 2.1.2 and 2.2.

- spectrum_*.dat [only after `raytrace glob write` command]
  As spectrum.dat but now for a series of inclinations. See Section 2.9.2 for details.

- image.dat [only after `raytrace image` command]
  This file contains the image that has just been produced. First line, 3 integers: nr of pixels in horizontal direction, nr of pixels in vertical direction and number of frequencies for which the images are stored in this file (normally this is 1). Second line, 2 numbers: Size of each pixel in horizontal direction and size of each pixel in vertical direction (size in cm, i.e. corresponding to the local scales of the object to be imaged). Then follow the image(s). Outer loop (usually trivial because only 1 image): nr of image. Middle loop: iy index (vertical). Inner loop: ix index (horizontal). The image is given in units of erg/cm^2/s/Hz/ster, i.e. the typical CGS units of an *intensity*. Note 1: for an image there is no distance indicator necessary because the intensity is distance-independent. Note 2: there are resolution issues related to such rectangular images (see Section 2.9.3).

- imtau.dat [only after `raytrace image` command]
  As image.dat but this time with the optical depth throughout the source as a function of position. For diagnostic purposes only.

- circimage_sequence.dat [only after `raytrace image circu`]
  This file contains a series of circular images. Circular images are described in detail in Section 2.9.4. First line: nr of frequencies (i.e. nr of images) in this file. Second line = blank. Then follows a list of frequencies at which the images are taken. Then again a blank line. Then a line with 3 integers: nr of radial grid points of the image (= nr of pixel circles), nr of Φ grid points of the image (= nr of pixels along each circle), and again (sorry!) the number of frequencies. Then another blank line and a list of the radial grid points (pixel distances from central point in cm). Again blank. Then a list of the interfaces of the pixels in radius (i.e. same as before, but this time not pixel centers but pixel edges): this list is 1 longer than the list of pixel radii because the number of pixel edges is 1 more than the number of pixels. Then again a blank and a list of Φ points (angle of pixels along the circle), a blank and a list of Φ interfaces (again 1 more). Then finally after yet again a blank follows the image. Outer loop: radius, inner loop: position along circle. The image is given in units of erg/cm^2/s/Hz/ster, i.e. the typical CGS units of an *intensity*. Note 1: for an image there is no distance indicator necessary because the intensity is distance-independent. Note 2: although circular images are a bit more difficult to interpret than simple rectangular images, they are extremely useful because they resolve automatically all scales necessary (contrary to rectangular images, which require great caution to do correctly).

These files can be analyzed using the IDL routines in sources/idlroutines/ (See Chapter 4).

2.5 A fix for low photon statistics: the diffusion module

Any Monte Carlo code of the type of Bjorkman & Wood has the problem that photon packages tend to rarely visit cells that are very deep in a very optically thick region, for instance the midpane of a protoplanetary disk. This means that some cells may have poorly determined temperature (with large statistical noise) or even zero temperature. Since these regions are usually so deep inside the disk, they are not observable anyway. For purposes of making spectra and images this bad photon noise is therefore not particularly problematic.

However, if the temperature solution is to be used for an iteration of vertical structure, then the temperature must be a smooth function also in the deep interior of the disk. The only true solution to this problem is to take an exceptionally large number of photon packages (e.g. take `nphot=10000000` in radmc.inp, or equivalently in problem_params.pro. But this is very numerically costly, because high number of photons is anyway costly, but at high optical depths each photon package also takes up more CPU time.

A trick is to smooth these cells out using the equation of diffusion, which should be reasonably valid in very optically thick regions anyway. By specifying `nphotdiff` in radmc.inp to e.g. 30 one can say that each cell that is visited by fewer than 30 photon packages will participate in the diffusion equation. This will then be solved using a matrix equation solver.

NOTE: Using this diffusion module to prevent strong photon noise is crucial when using the vertical structure iteration module (see Section 2.6).

2.6 Vertical structure iteration

Normally RADMC is just a continuum radiative transfer code (if nvstr=0 in the radmc.inp file). But it has turned out to be exceedingly useful also to iterate on the vertical density structure, assuming that the gas has the same temperature as one of the dust species (which one is specified by ivstrt in the radmc.inp file). The vertical structure iteration is then computed after each Monte Carlo calculation, and at the very end a final Monte Carlo calculation is done for the final temperature and scattering source function.

The simulation starts from the dust density file dustdens.inp, which gives the 2-D/3-D density structure. It does one run of the Monte Carlo radiative transfer to obtain the temperature structure in 2-D/3-D. Now the surface density of each dust species is calculated. For the selected dust species (see below) the vertical density distribution is now replaced by a distribution consistent with the dust temperature distribution of dust species ivstrt (which is assumed to be the gas temperature). At each radius the surface density of the new dust density distribution is constructed to be the same as from the original density distribution. Once the new density distribution is found, the Monte Carlo radiative transfer is redone, and again the density can be modified. This procedure can be iterated multiple times (in fact nvstr times). This iteration stops if the largest difference is less than vserrtol. *Note: Due to photon noise one can presumably not get a very well converged solution.*.

Which of the dust species will be modified in this way is given in the file *vstruct.inp*. See Section 2.1.2 for explanation.

Note: To make sure that the vertical structure iteration has a temperature profile that is smooth enough, the diffusion method of Section 2.5 is important.

VERY IMPORTANT: For the moment never use this mode in conjunction with the CHOPDENS code, because the surface density is computed from the input density which would then have been modified. In fact, in the newest version of the code RADMC will simply refuse duty if the vertical structure mode is on while a chopdens.inp file is present.

2.7 Quantum-heated grains (PAHs)

This section is still under construction...

2.8 The use of RADMC

Once all the input files are there, and all the codes are compiled, the RADMC is very simple to use from the shell prompt:

```
nice ../bin/radmc
```

This will run the code, and the code will output its results. These results are not directly of practical use. But with a postprocessing using RAYTRACE one can produce spectra and images.

2.9 The use of RAYTRACE

Once RADMC is done, and new dust temperatures and scattering source functions have been produced (dusttemp_final.dat and scatsource.dat), then the ray tracing routine is required to produce the spectra and/or images. During compilation, RAYTRACE is installed such that it should be directly accessible, even without a path, by just typing 'raytrace XXXXXX' in the command line where 'XXXXXX' stands for various options and commands (see below).

2.9.1 Making an SED

To make an SED (spectrum) at some inclination, type

```
raytrace spectrum incl 45
```

on the unix/linux shell. If this does not work, then

```
../bin/raytrace spectrum incl 45
```

should do the trick. This command makes RAYTRACE produce the SED as seen at an inclination of 45 degrees (where 0 is face-on). The resulting spectrum is written to the file spectrum.dat, which has exactly the same format as starspectrum.inp (i.e. two columns with col 1: frequency in Herz and col 2: flux in erg/cm^2/s/Hz as seen at a distance of 1 parsec). With the analyze.pro routines you can read and plot the spectrum:

```
idl
IDL> .r ../sources/idlroutines/analyze.pro
IDL> s=read_spectrum()
IDL> plot_spectrum,s
```

You can also give various keywords to plot_spectrum, such as the usual keywords, but also things like /lsun or dpc=140.. See the analyze.pro routine for explanations.

2.9.2 Checking flux conservation; making many SEDs at once

RAYTRACE can make a systematic series of SEDs at a large number of inclinations in one command, even though this make take some time to compute. The usefulness of this `glob` mode is that it allows you to get a spectra view of the system at all inclinations, AND it will *check if the outcoming luminosity equals the input luminosity*. The latter thing is most important: it checks whether the results are self-consistent.

On the command line type

```
raytrace spectrum glob nincl 45 write
```

to obtain 45 spectra at inclinations 1, 3, 5 ... 89 degrees inclination. They will be in the files spectrum_1.dat, spectrum_2.dat ... spectrum_45.dat. Note that the index number of these files denote the *index* of the spectrum, not the inclination in degrees. So the spectrum at 5 degrees is in file spectrum_3.dat. The `write` command at the end of the above command line is to say that indeed each spectrum that is calculated must be written to a file. If `write` is not given, then RAYTRACE will compute all spectra, compute the luminosity conservation error, but will not write out any spectra. If `nincl 45` is not given, then RAYTRACE will take the θ-grid of the model as its inclinations.

The luminosity conservation error is calculated by integrating the SEDs at each inclination over frequency, and then integrating these fluxes over $d\cos i$ to get the total output luminosity:

$$L_{\rm out} = 2\pi d^2 \int_{-1}^{+1} d\cos i \int_0^\infty d\nu F_\nu(i) \tag{2.1}$$

where d is the distance of the observer from the source, which is standardized to $d = 1$pc for the F_ν coming out of RAYTRACE. Then this output luminosity is divided by the input luminosoty (L_*) assuming that the star is the only source of energy. This should ideally be 1. The error is calculated as the deviation from 1, but that number would anyway be very small if for instance the circumstellar matter is very optically thin, so that is not a good estimate of the true error. Instead this error is divided by the covering factor Ω of the circumstellar matter wrt to the star:

$$\text{Error} = \left(\frac{L_{\rm out}}{L_*} - 1 \right) \frac{1}{\Omega} \tag{2.2}$$

Typically this error should be of the order of 0.05 at maximum, but preferably less. If this error is bigger, then something is wrong. Either there is a bug in the code (please report to the author, giving the exact input data and configure.h of the compilation as information) or the gridding of the input problem is not done well.

Making a flux conservation check is time-consuming. My advice: Do this test in the beginning a few times when you are getting used to the code, just to make sure things are OK. Do the test only once-in-a-while during your production runs. And do the test for the final models that go into the paper.

2.9.3 Making images with RAYTRACE

RAYTRACE can make images in one of the wavelengths/frequencies of the frequency.inp frequency grid. An image is made by computing the ray-tracing along a set of NX×NY rays going through the object at some inclination. One can make images directly from the command line[1]:

```
raytrace image lambda 15. micron incl 45. size 160 AU npix 200
```

which makes a rectangular 200×200 image at an inclination of 45 degrees, with a size of 160 AU from left to right and top to bottom (i.e. 80 AU from center to right), at a wavelength of 15 micron (note that it will choose the frequency in frequency.inp closest to $10^4 c/\lambda$, i.e. it is not precisely at 15 micron). If we just write

```
raytrace image
```

then it will make an image at default wavelength, default inclination etc.

Another way to make images is to use the readimage.pro/makeimage() routine. Have a look at this routine (in the sources/idlroutines/ directory) for further information.

Finally, one can also directly set the command file (the file that normally is produced by the PERL script raytrace to hand over information to raytraceprog). To see how this is done, just have a look in the readimage.pro/makeimage() file how that program does it.

- **IMPORTANT:** The pixels centered around the central star will often not be small enough to resolve the complex structure of the source at small radii. The flux of these pixels will then not be trustworthy because for each pixel only a single ray going though the center of the pixel is traced, which may miss important emitting material at small R that could/should contribute to the flux of that pixel. Example: a model disk ranging from 0.1 AU $\leq R \leq$ 1000 AU, where an image of 100×100 is produced with $x_{\rm max} = 1000$ AU. The pixel size is then 20 AU. This means that the central 2×2 pixels have rays with impact parameters of $b = 10\sqrt{2}$ AU away from the center, missing all the strong emission in the region within 15 AU. If that region produces most of the flux, then this is not picked up by the imager, and the central 2×2 pixels are far less bright as they should be. This is a typical *missing flux problem* of rectangular images. A quick-and-dirty fix, just to make sure that at least the stellar flux is correct, is to add the option addstar to the command line. But this only solves the stellar flux proble, not the flux from the unresolved inner disk regions. A more allround solution is by making consecutive 'Babushka' style images at ever increasing size and by replacing the central pixels of the bigger images by the integrated versions of the corresponding regions of the smaller (higher-resolution) images. Right now this is not yet automatized. Another solution is to use *circular images* (see Subsection 2.9.4).

This section is still under construction.

[1] There was an error in the manual. It said raytrace image lambda 15. micron incl 45. imagesize 160 AU npix 200 200, but that was incorrect. This error is now fixed.

2.9.4 Circular images

Another way of obtaining spatial information about radiation from the object, which does not suffer from the resolution problems of the rectangular images, is the use of circular images. The idea here is that we let go of our standard belief that images must always be sets of pixels in X- and Y- direction in a rectangular arrangement. The human eye sees images with 'pixels' in completely random arrangement and still we see perfectly orderly images. So the trick is to arrange the pixels of the image in such a way that they naturally resolve all scales of the problem automatically. In RAYTRACE they are arranged in concentric circles centered around the central star. The size of the circles (i.e. the impact parameters with respect to the star of the rays belonging to the pixels) are chosen to correspond to the radial grid of the model. Typically for each R_i of the model grid we will have a circle of pixels with the same radius (note that a distance-independent angular size on the image is in fact obtained if you measure it in cm, i.e. in object coordinates). The arrangement of pixels along the circle is regular, typically with 32 points along the circle (but this can be changed in raytrace.inp by setting the variable nrphiinf to the appropriate value, as long as it is a multiple of 4). The way to make circular images from the command line (csh/bash):

```
raytrace image circu incl 45 freqindex 5 40
```

which says that the circular images must be made at an inclination of 45 degrees and in this case 36 images are made, from `inu`=5 to including `inu`=40 (see frequency.inp to which frequencies this corresponds). NOTE: The `lambda 10 micron` option that works for the spectrum does not work here. Instead you must use the `freqindex 20 30` (or other indices) option to specify literally for which frequency indices you wish to make the circular image. Note that if you do not specify `freqindex 10 15` (or like that) then the circular images are made for *all* frequencies:

```
raytrace image circu incl 45
```

You can also use readcircimage.pro/makecircimage() (see sources/idlroutines/) which is an IDL routine that does it all for you.

Finally, you can also set the command file directly. See readcircimage.pro/makecircimage() how this is done.

Once the circular image(s) is/are made, then with readcircimage.pro/readcircimage() you can read in the circular image(s) into IDL into a structure.

NOTE: The routines in RAYTRACE that calculate the SED in fact make circular images at each wavelength and then integrate over these images. In this way it is guaranteed that the spatial scales (and thereby the fluxes) at all radii are included and the total flux is robust.

This section is still under construction.

Chapter 3

The template models

In principle the code package consists of the programs located in the sources/ directory. These are the fortran programs that perform the computations. But the full package also contains example model setups that are in fact rather advanced. In this chapter these packages are described. It is strongly encouraged that the user modifies these packages; they are there fore convenience, but they have their limitations.

3.1 Model directories

The way the current package is structured is such that each model has its own directory. The reason for this is that there are many input and output files for each model and it would be come quickly too messy if one directory contains more than one result. Therefore it is strongly encouraged to have *one model - one directory*. There are already example directories present:

- run_example_ppdisk: An example of a model of a protoplanetary disk. See Section 3.4.
- run_example_agn: An example of a simple AGN torus model. See Section 3.5.

It is advised to first make a copy of these example directories to e.g. run_ppdisk_1 (for the first example), so that you have your own model directory to play with, without accidently modifying the example model. After a succesful model, if you plan to continue to play around with parameters, it is advisable to keep the succesful model in its own directory, and simply do cp -r run_ppdisk_1 run_ppdisk_2 to create a new model and continue to play with run_ppdisk_2. In this way each model has its own directory, and can be easily re-run after a long time without having to ponder how the setup was done.

3.2 The problem_*.pro IDL modeling packages

Each model directory contains:

- A set of problem_*.pro files. These are the IDL model setup routines. See below for details.
- A set of *.Kappa files. These are the master opacity files. See Section 3.3

NOTE: Specific information for the example models can be found in separate sections below.

If you wish to use these example packages as-is, then you will likely just modify the problem_params.pro file, which contains all the model parameters (see below). The main routine is problem_setup.pro. So to set up the problem, you first edit the problem_params.pro and then you typically compile all codes by executing problem_compilecodes.pro:

```
idl
IDL> .r problem_compilecodes.pro
IDL> exit
```

and then you set up the problem by executing problem_setup.pro:

```
idl
IDL> .r problem_setup.pro
IDL> exit
```

This will create all the input files for the RADMC and RAYTRACE programs.

NOTE: The compilation of the codes only has to be done the first time you use the code OR if you changed the number of R-, Θ- or frequency-grid points of the model. If you only change parameters of the model setup but not the grid size, you only need to execute problem_setup.pro *to get the new model.*

Once you have done this, then you can call the fortran programs. For instance:

```
nice ../bin/radmc
nice ../bin/raytrace spectrum incl 45
```

See Chapter 2 for details of the codes. Then after the resulting spectrum has been written to spectrum.dat (or if you made an image with RAYTRACE, the image.dat has been written) you typically will want to analyze the results. You can do that either by reading in these output files in any way you want to do yourself. Or you can use the diagnostic tools described in Chapter 4. That is it!

3.2.1 A brief description of the generic problem_*.pro files

If you plan to modify the setup routines problem_*.pro to your own wishes, which you are very much encouraged to do, then you will need to know the functions of these files. Here is a very brief description of the files common to all example problems (specific example problems may contain additional problem_*.pro files):

- problem_params.pro
  This is the file that contains all the model parameters. It is the file that you (the user) will most often edit and change.
- problem_setup.pro
  This is the main routine for the model setup. This is the routine that you will execute from the IDL prompt to set up all the model input files for RADMC and RAYTRACE. This main routine calls all other routines and other problem_*.pro files.
- problem_compilecodes.pro
  This is the main routine for the automatic compilation of all necessary fortran codes, including RADMC and RAYTRACE.
- problem_grid.pro
  Contains routines for setting up the spatial grid (R and Θ).
- problem_subroutines.pro
  This file contains a number of generic (not model-specific) IDL subroutines used by the model setup.
- problem_makeopac.pro
  This file contains the routine useopac() that creates the dustopac_*.inp files from the *.Kappa opacity master files. See Section 3.3 for details.
- problem_mixopacities.pro
  This contains routines for mixing opacities. The mixing is done in a simplistic way by simply adding the opacities together multiplied by their respective abundances. See the parameters mixnames, mixspecs and mixabun in problem_params.pro and see Section 3.3.3.
- problem_natconst.pro
  Contains the values of several natural constants in CGS units.
- problem_kurucz.pro
  Contains routines for implementing a Kurucz model as stellar input spectrum (i.e. putting a Kurucz spectrum into starspectrum.inp. This routine only works if there exists a Kurucz model directory in a special format. Since this is a big package, this Kurucz model directory is typically not delivered along with this package. Please request it separately from me if you need it. To switch the use of the Kurucz model on, put kurucz=1 in problem_params.pro.

- problem_pah.pro
  Routines for implementing quantum-heated grains (PAHs) into the model. See Section 2.7 for details.
- problem_comp_files.pro
  This contains helper-routines for the problem_compilecodes.pro, i.e. it is the part which writes the appropriate configure.h files for the compilations.
- problem_files.pro, problem_build.pro, problem_disk.pro, problem_models.pro, problem_radmc_files.pro
  These are problem-specific files (and typically unique for each model). They are described in more detail in the model-specific sections of this chapter.

3.2.2 A brief description of generic problem_params.pro input parameters

Although each example model has its own model-specific parameters in the problem_params.pro file, there are a number of general parameters that are the same in all example models.

- nphot: Nr of photon packages used by RADMC. See Chapter 2.
- npdiff: This tells RADMC that if a cell is visited by fewer than this number of photon packages, then this cell will be included in the diffusion algorithm after the MC run (see Section 2.5).
- errtol: This is the error tolerance for the diffusion algorithm.
- ifast: If 1 then RADMC will only recompute the dust tempearture in the MC simulation when this is likely to have changed by some reasonable amount (Set to 1 for now).
- tauchop: For the chopping code (see Section 3.2.4).
- lmbchop: For the chopping code (see Section 3.2.4).
- idxchop: For the chopping code (see Section 3.2.4).
- xlevel: This sets the depth in the directory tree where this run directory is located with respect to the sources/ directory. Standard this is 1. But if for instance you wish to make a series of models in a deeper directory called e.g. series_1/, then the models series_1/run_1/ will be deeper in the directory tree than the run_example_ppdisk/ directory. In that case you set xlevel=2. This is important for the problem_compilecodes.pro to be able to find the sources/ directory and also for problem_setup.pro to find the chopdens code if requested.
- rstar: Radius of the central luminosity source
- mstar: Mass of the central luminosity source
- tstar: Effective temperature of the central luminosity source If kurucz=0 then this will be the blackbody temperature, but if kurucz=1 this will be the Kurucz effective temperature.
- fresmd: The frequency gridding mode. See Section 3.3.2.
- infile: Array of names of master opacity files to be used as dust opacities (see Sectionsec-master-opacity-files). The number of opacities specified in this way must be equal to the number of dust components used in the dust setup (for the ppdisk example this would be the number of elements of ab_ab0 *plus one*; for the AGN model this must be 1).
- pll: If the maximum wavelength in the master opacity files is smaller than the maximum wavelength used in the model (see Section 3.3.2), then use this power law for the extrapolation toward longer wavelengths. Must have same number of elements as infile. Typically this is taken to be either -1 or -2.
- scat: If 1, then include scattering into the modeling. If 0, then put the scattering opacities to 0.
- nr: Number of radial grid points of the model (when this is changed, you need to recompile the codes). The grid points are normally distributed logarithmically between rin and rout so that all spatial scales of the problem are well resolved. Note, however, that the inner gridpoints are refined (see rrefine below).

- nt: Number of Θ grid points of the model (when this is changed, you need to recompile the codes). Normally the Θ grid is linearly spaced. Note that the Θ grid only spans from pole to equator: there is mirror symmetry in the equatorial plane.
- ntex: To make sure that a disklike structure is well resolve in Θ, the Θ grid is divided into two zones: a polar zone and an equatorial zone. The ntex variable sets the number of gridpoints in the polar zone. Typically one chooses fine resolution in the equatorial zone if a disk is present and less resolution in the polar zone. The number of grid points in the equatorial zone is then nt-ntex. The division line between the polar and equatorial region is set by hrgrid (see below). **NOTE:** In earlier version of this setup the total number of Θ grid points was nt+ntex. Now this is nt. Except when a special mode is used, namely zrefine, where a special very fine Θ grid is made near the midplane. But that latter mode is only expert-use because the codes will not automatically compile with the right number of Θ points in that case...
- rin: Inner edge of the radial grid in cm. If this parameter is set 0 and instead tin is specified, then tin will determine the inner radius instead.
- tin: If > 0 and rin=0, then this parameter sets the inner radius of the grid in a special way. It *estimates* the radius where the dust grains will have a temperature of tin and will take that to be the inner radius of the grid. **IMPORTANT:** This is just a very rough estimate based on the blackbody formulae of the Dullemond, Dominik & Natta (2001) paper. In reality the dust grains, if small grains, will be somewhat warmer than tin. Therefore use tin not as an exact inner temperature but rather a nice way to make sure that the inner radius always scales propertly in- or out-ward when you change the luminosity of the star. Do *not* trust that the inner disk temperature will indeed be equal to this tin.
- thintin: If set to 1, then use real dust opacities for computing rin from tin. This may make the estimate of rin from tin a bit more accurate.
- rout: Outer radius of the grid, in cm. Make sure that this is large enough that the entire model density distribution fits in.
- hrgrid: The boundary between the polar and equatorial Θ-grid region, measured as $\Theta_{\rm boundary} = \pi/2 - $hrgrid.
- hrgmax: The maximum of $\pi/2 - \Theta$. The closest to π, the closest the Θ grid comes to the polar axis. Note that the polar axis is a coordinate singularity and the closer hrgmax is chose to be near π, the easier it can become that numerical errors appear (in particular very low photon statistics in the first Θ grid cell).
- rrefine: This is a structure containing three parameters that control the radial grid refinement near the inner edge. First the radial grid is simply set up in a logarithmic way. The nspanr parameter now tells how many grid spacings from the inner edge need to be refined. nstepr tells how many sub grid points have to be inserted in each of these spacings. The nlevr tells how often this procedure is recursively repeated. This radial grid refinement is usually necessary to assure that the inner grid cell remains optically thin. NOTE: Unfortunately this refinement cannot be done indefinitely. The RADMC and RAYTRACE codes have limits to the fineness of the grid. This can be experimentally investigated.
- drsm: In case of protoplanetary disks the optical depth of the disk can be so enormous ($\gtrsim 10^6$ or more) that grid refinement using rrefine would need to be extreme. This requires a great many grid points and also may require so fine spacing that the code may complain. To prevent this the inner rim is smoothed out a bit automatically by the setup routines to guarantee that with the present grid refinement the inner cell remains optically thin. This smoothing is done over a range in radius from rin to rin*(1+drsm) (with rin being the inner radius of the grid).

For further parameters, which are model-specific, we refer to the model-by-model descriptions below (Sections 3.4, 3.5).

3.2.3 The diagnostic diagrams of the template models

At the end of each call of the problem_setup.pro there will be a popup diagram. This diagram is for diagnostic purposes and it can be switched off if it is annoying by setting show=0 in the problem_params.pro file.

The purpose of the upper panel of the diagram is to show the region in R, Θ space where the optical depth toward the central star at peak-stellar-wavelength is larger than 1 (red region). The rough shape of the parameterized

disk is therefore shown. The radial coordinate is the spherical (not cylindrical!) radius R. The vertical coordinate is $\pi/2 - \Theta$ which is a rough (though not exact) estimate of the dimensionless vertical coordinate Z/R. The bottom of this diagram is therefore the equatorial plane. Using this panel you can see how the grid resolves (or not resolves) the important structures you want to model. If the equatorial Θ-grid (having nt-ntex grid points) does not nicely cover all of the disk, then one might need to increase the hrgrid variable. If, on the other hand, the equatorial Θ-grid covers much more than the disk, then you can increase the Θ resolution around the disk not only by increasing nt and keeping ntex constant, but also by lowering hrgrid such that the equatorial Θ grid nicely covers the disk and not too much more.

Note that if vertical structure iteration is switched on, then the final disk structure could be very different. This also means that if the hrgrid parameter is chosen such that the equatorial grid nicely covers the initial disk, it may not nicely cover the final disk anymore. This requires some experimentation.

The purpose of the low panel is to show the cumulative radial optical depth at peak-stellar-wavelength along the equatorial plane ($\Theta = \pi/2$), in the inner few gridpoints at the inner edge. It shows whether the transition from optically thin to optically thick does not go too abruptly in the inner edge. The reason why this is a worry is that if the inner edge goes from $\tau \ll 1$ to $\tau \gg 1$ in one grid point, certain solid state features in the spectrum may be suppressed. In the case of the run_example_ppdisk, however, the setup routines attempt to avoid this automatically.

3.2.4 The chopdens program

If model density structures are extremely optically thick, then the code RADMC can become a bit slow. This is because some of the photon packages can then get stuck deep in the disk where they will ping-pong billions of times before they escape. This slows down the code considerably.

If one is only interested in the output spectra and images of such objects, and one does not use the vertical structure mode of RADMC (this is important!), then one may not mind too much if the very optically thick disk regions are not modeled absolutely correctly. The reason is that these regions are anyway unobservable. In that case one may wish to remove some matter from these very interior regions such that the total optical depth is not too extremely large anymore. This is a modification of the setup, but a modification that is not likely to affect the SEDs and images because they do not affect at all the surface regions of the object; only the interior regions which are unobservable anyway.

In this case the fortran program chopdens can help. It will be automatically called by the problem_setup.pro routine if the parameters tauchop, lmbchop and idxchop are set in the problem_params.pro file. This program will simply remove as much matter from the interior regions such that the vertical optical depth at wavelength lmbchop at each radial position is smaller or equal to tauchop. The idxchop determines how abruptly this chopping is done.

To see how chopdens does its work one can simply read in the file dustdens.inp after the execution of problem_setup.pro using the read_dustdens() routine from the analyze.pro program (see Chapter 4 for information about the various diagnostic tools provided in this package). By comparing the density distribution with and without chopping one can see what precisely the chopping does.

The advantage of chopping is that the RADMC code will execute faster. The disadvantage is that it is not guaranteed that the resulting SED and images are correct. In general the higher one sets tauchop the safer it is. The safest is not to use chopping at all. By setting tauchop=0 one can switch off the chopping altogether. By not defining tauchop (by commenting it out) one can also disable chopping, but beware that if you have used it before and have not exited IDL in the mean time, the tauchop variable will then still be set even after having commented it out in the problem_params.pro file. Exiting IDL and re-entering it would then be useful.

3.3 The master opacity files

The fortran programs RADMC and RAYTRACE read their opacity information from the dustopac_*.inp files. The problem with these files is that the opacities have to be sampled precisely on the same frequency (wavelength) points as all the other frequency-dependent data such as the stellar spectrum etc. If you wish to change the frequency resoltion/sampling of the opacities, then you would in principle have to create entirely new dustopac_*.inp files. *However, in the problem_*.pro setup routines this is all done automatically.* You merely have to specify in problem_params.pro which frequency resolution mode you wish to use, by setting the infile parameter appropriately, and the dustopac.inp files are automatically generated with the right frequency sampling.

In this section it is explained how this works, and where the original opacity data come from. So what happens is that the problem_setup.pro routine calls a subroutine called useopac() in the file problem_makeopac.pro.

This routine reads high-resolution opacity input files *.Kappa, the precise names of which are specified by the user in problem_params.pro. These are the *master opacity files* containing tables of the opacities at very high frequency resolution - much higher than you will typically use in the modeling. From these tables, using interpolation, the routine useopac() will create the dustopac_*.inp which will then contain the opacities mapped onto the globally used frequency grid of the model.

The *.Kappa files have their own frequency grid, independent of the frequency.inp file. Each *.Kappa can have a different frequency grid. The interpolation routine of useopac() will make sure that the opacities are then all mapped onto the same frequency grid (the one of frequency.inp) before they are written to the dustopac_*.inp files.

3.3.1 The file format of the master opacity files

The structure of these master opacity files is as follows. The first line contains the format number (see below for its meaning). The second line contains the number of frequency sampling points for this file (may be different for each *.Kappa file). Then follows the opacity data. First column: wavelength in micron. Second column: absorption opacity in units of cm^2/gram-of-dust. If the format number is 2 or larger then there is a third column: scattering opacity in units of cm^2/gram-of-dust. If the format number is 3 or larger then there is a fourth column: the Henyey-Greenstein g-factor for anisotropic scattering (only for the version of RADMC that has anisotropic scattering implemented; not this version).

3.3.2 Choosing the frequency resolution for the model

So how does one choose which frequency sampling to take for the model? There are a number of pre-defined frequency grids available, each with a integer label. They are defined in the problem_makeopac.pro file. There is a case statement in which the keyword fresmd determines in which way the frequency points are distributed. The way this is done here is that the frequency points are generally distributed logarithmically, but in three zones. A short-wavelength zone, a mid-wavelength zone and a long-wavelength zone. The boundaries of each zone are specified by setting four wavelength values, and the number of frequency points in each of these zones is specified using three integer numbers. There are a number of pre-defined settings for these values, and the fresmd keyword (which is set in the problem_params.pro file) chooses wich one to use. You are free to add more frequency distribution modes to problem_makeopac.pro and assign each its own unique label number.

3.3.3 Mixing the opacities

By specifying the mixnames, mixspecs and mixabun in the problem_params.pro file you request that certain master opacity files are mixed together. This is done automatically by the problem_setup.pro routine by calling the mixopacities() routine.

The master opacity files given by mixspecs are being mixed and a new (mixed) master opacity file given by mixnames is generated. Note that multiple mixtures can be generated by making the mixspecs array a 2-D array (e.g. mixspecs=[['a.Kappa','b.Kappa'],['c.Kappa','d.Kappa']], which mixes a and b together and c and d together), and making mixnames a 1-D array.

The mixing is simply done by adding the opacities times their abundances together to form a single 'dust grain' with an opacity that is the mix of the constituents. In the RADMC code this new dust grain (if then subsequently selected in the infile parameter) is then treated as if it is one single species. The RADMC code will then not know about the fact that it is actually am mixture.

Note that any number of opacities can be mixed into one. BUT the first of the specified master opacity files will set the frequency grid! This is very important to realize: it is best to take the opacity with the finest frequency grid as the first. It is not impossible that this may cause a problem if none of the master opacity files have a frequency grid that spans the entire range of interest with sufficient grid points. In that case the mixing may have to be done by you in your own way, or the mixopacities routine will have to be adapted.

Once the mixing is done and a new (mixed) master file is created, you can use it as a master opacity file as any other master opacity file. The mixing is therefore to be seen as a pre-processing action.

3.4 The Protoplanetary Disk model package

This problem_setup.pro is in fact just a very short wrapper around a series of possible models given in problem_models.pro. You can therefore also modify problem_setup.pro to choose another model. You can, if you are sure of yourself, also modify the problem_models.pro file, or even add your own model setups there.

This manual will not outline all the details of the various models that are currently implemented. The description of the parameters in the problem_params.pro should be self-explanatory. Experimentation will certainly help. But a brief description of some of the model parameters in problem_params.pro is given here. For the generic model parameters in problem_params.pro we refer to Section 3.2.2.

- kurucz: If set to 1, then use a Kurucz spectrum for the star. See Section 3.2.1.
- gastodust: The gas-to-dust ratio used to calculate the mass of dust when given the mass in gas.
- rhofloor:: The lowest allowed dust density
- run: If 1, then automatically run RADMC and RAYTRACE from within the IDL package. Otherwise only do the setup.
- rdisk: The outer disk radius (must be smaller than rout). Note that the density distribution does not go do 0 abruptly beyond rdisk. It will go down with a very steep powerlaw: plsig2.
- sig0: If you specify sig0 and not mdisk, then you specify the surface density at r=rdisk.
- mdisk: If you specify mdisk and not sig0, then you specify the disk mass (gas+dust), and the sig0 is, so to speak, computed from mdisk.
- plsig1: the powerlaw of the surface density inward of rdisk.
- plsig2: the powerlaw of the surface density outward of rdisk. Take this -12 or -24 or so, so that the density drops so fast that rdisk is indeed effectively the outer disk radius. The reason why I do not simply put the density to zero outward of rdisk is to make sure that in images the disk does not have a really weird appearance.
- bgdens: you can add to this all a background density if you want to embed the disk into some interstellar medium.
- ab_r0: If you use more than 1 dust species, then the abundance for each additional species has to be specified. This is done by the ab_* variables. This is meant to mimic a simple radial mixing scenario of the kind of Gail et al. The ab_r0 parameter sets the point where the powerlaw mixing starts. For more than 2 dust species this becomes an array of nspec-1 elements.
- ab_ab0: The abundance inward of ab_r0. For more than 2 dust species this becomes an array of nspec-1 elements.
- ab_pl: The powerlaw for the mixing of this species. For more than 2 dust species this becomes an array of nspec-1 elements.
- ab_min: The floor value for the abundance of this species. For more than 2 dust species this becomes an array of nspec-1 elements.
- hrdisk: The parameterized pressure scale height of the disk at rdisk. Note that one must take care to choose this not too far from any realistic hydrostatic equilibrium if one does not use the vertical structure iteration mode.
- hrmin: If the pressure scale height of the disk becomes so small that the disk is squeezed in the single equatorial Θ grid cell, then evidently the grid resolution in Θ is not good enough. But another way of preventing this from happening is to prevent the disk from becoming thinner than some value. This is the hrmin parameter.

- plh: The parameterized powerlaw of the H_p/R of the disk. Here also, this is just a parameterized shape of the disk, and it may not reflect the real thing, unless the vertical structure iteration mode of RADMC is used (which erases the plh and hrdisk information after the first iteration). NOTE: Often people take $2/7$ because of the Chiang & Goldreich paper. But that estimate is based on grey opacities and constant surface density. So somewhat smaller flaring is usually more realistic.
- rpfrin: A parameterized way of mimicking a puffed-up inner rim which is often seen in the truly vertical structure models. In this case it is parameterized. This parameter tells how wide the inner rim must be, ranging from rin to rin*rpfrin.
- hrpuff: A parameterized way of mimicking a puffed-up inner rim. This gives the H_p/R ratio to be set at the inner rim. NOTE: This is a dangerous mode because one is too easily tempted to put the puffing-up too large. In reality the puffing-up is very subtle.
- nvstr: The number of iterations for the vertical structure calculation. If 0, then only the parameterized density distribution is used for the radiative transfer. If ≥ 1 then 1 or more iterations are done for the vertical structure.
- vserrt: The error tolerance for the vertical structute iteration to end. NOTE: Due to the photon noise the error typically remains rather high even after many iterations. Therefore this parameter is often of less use. If convergence is not formally reached, then the maximum number of iterations is nvstr.
- ivstrt: This specifies which of the dust components (species) will be used to represent the gas temperature for the vertical structure iteration.
- dostr: This is an array with a 0 or 1 for each dust component. If 0, then this component is not participating in the vertical structure iteration. If 1, then this dust component IS participating. In this way one can allow disk components to participate, while keeping envelope components untouched.

3.5 The AGN Torus model package

The AGN model is much simpler than the protoplanetary disk model. Here we model a simple geometry of dust around a central source of luminosity. The central source of luminosity is an actively accreting supermassive black hole that radiates a broad spectrum. We use the spectrum parameterized by Granato & Danese (1994).

NOTE: In this model setup the automatic smoothing of the inner rim is not (yet) included. This is because it is not usually so necessary because the optical depths here are not so large as in the disk case.

- rdisk: The outer radius of the torus in cm.
- hrdisk: The parameterized H/R height of the torus at rdisk.
- plh: The parameterized powerlaw of the H/R of the torus.
- sig0: If you specify sig0 and not mdisk, then you specify the surface density at r=rdisk.
- mdisk: If you specify mdisk and not sig0, then you specify the torus mass (gas+dust), and the sig0 is, so to speak, computed from mdisk.
- plsig1: the powerlaw of the surface density inward of rdisk.
- plsig2: the powerlaw of the surface density outward of rdisk. Take this -12 or -24 or so, so that the density drops so fast that rdisk is indeed effectively the outer torus radius. The reason why I do not simply put the density to zero outward of rdisk is to make sure that in images the torus does not have a really weird appearance.

THIS PART OF MANUAL NOT YET READY 28.07.07

Chapter 4

Diagnostic and executional tools in IDL

To make life easier with a complex program such as this one I provide a number of subroutines in the sources/idlroutines directory. Most are for diagnostic and plotting purposes only. Some are to help making models and/or images.

Here is a list of useful subroutines and the files in which they reside (written as <filename>/<subroutine>). To really understand how they work, please simply have a look into the respective codes, for it would go too far to explain all their functionality in this manual.

- analyze.pro/read_dustdens()
  Usage: `a=read_dustdens()`
  This routine reads the dust density distribution from the file dustdens.inp and returns this, together with other useful information, in a struct. This useful information includes the surface density of the dust (i.e. the ρ integrated vertically along Θ), the total mass of the dust, the radial and Θ grid etc.

- analyze.pro/read_dusttemp()
  Usage: `a=read_dusttemp()`
  Similar to the above, but now for the dust temperature in the file dusttemp_final.dat.

- analyze.pro/read_spectrum()
  Usage: `s=read_spectrum()`
  This reads the file spectrum.dat and returns a useful struct which can be inserted into the analyze.pro/plot_spectrum() routine.

- analyze.pro/plot_spectrum()
  Usage: `plot_spectrum,s`
  This plots the SED in various ways. This routine takes the resulting structure from read_spectrum() as input. See all the various keywords to this routine to obtain precisely the spectrum plot you wish to get. You can set the axis ranges (using xrange and yrange IDL keywords), you can switch off the log scale in either axis (by setting xlin and/or ylin), you can plot in Jansky instead of νF_ν (standard). You can plot the truly measured flux at some prescribed distance by setting the keyword dpc to the distance in parsec. If you do not set this then dpc is per default 1.0. You can also get the spectrum in νL_ν in units of $L_\odot$ by setting /lsun. Note that this luminosity is just the apparent luminosity as seen from this inclination under the (probably wrong) assumption that the source is a sphere. There are several other keywords (in addition to all the possible standard keywords for plotting in IDL), but a deeper understanding is best obtained by experimentation and/or inspection of the source code of analyze.pro/plot_spectrum().

- viewimage.pro
  Usage: `viewimage`
  A complete IDL GUI viewing program for viewing images at various angles, zoom factors and wavelengths. Just type `.r ../sources/viewimage.pro` and `viewimage,/au` (possibly with `,/small` added if viewed on a small screen) and you should get a GUI. This GUI can only work if RADMC has already done its work. It calls RAYTRACE to produce the images.

- readimage.pro
  Usage: `a=readimage()`
  This is a tool for reading rectangular images that have been made with RAYTRACE but also making them

(see routine makeimage()). Of course images can be made from the command line, but sometimes this makeimage routine can be more convenient.

- readcircimage.pro
  Usage: `a=readcircimage()`
  Similar to readimage.pro but now for the circular images.

- readopac.pro
  Usage: `o=readopac()`
  This is a file containing a routine readopac() which can read an opacity file (default: dustopac_1.inp, but by specifying e.g. o=readopac(nr=2) you can also read e.g. dustopac_2.inp) and a routine plotopac() which can plot this opacity in a useful way.

- visibility_circ.pro
  Usage: Read source code
  This is a set of routines to compute visibilities for interferometers. The important feature of this routine compared to most others in the literature is that, by using circular images (see Section 2.9.4), makes sure that all scales of the problem are properly resolved. When using rectangular images (and using more standard methods for computing the visibility) one can easily run into resolution problems. With the circular images this happens less easily. But even with circular images one can get into troubles: one can get aliasing effects for very high k values (i.e. for very high spatial frequencies, or in other words, very large baseline of the two telescopes). To prevent this aliasing the visibility_circ.pro has a new mode (imethod=2) which makes sure that the wave front is analytically exactly integrated over each pixel. Therefore the advise is to always use the new method.

Chapter 5

Tips and tricks

This chapter outlines a few tips for the user. In time more tips and tricks will be added in this chapter.

5.1 Typical beginners mistakes

1. **Where has my modification gone?**
   Modifying the input files for RADMC and RAYTRACE (as created by the problem_*.pro template tool kit) by hand is possible and may even yield the desired result. But beware that as soon as you make a new call to problem_setup.pro those files will be overwritten! This can sometimes cause confusion. It is therefore the best idea to make any modifications (unless temporary ones) directly in the problem_*.pro files. Since these models are anyway only meant as template models, you are invited to modify them to your wishes.

2. **'I use the disk model by Dullemond & Dominik (2004)'**
   The template disk models in this package are of course similar to the models of Dullemond & Dominik (2004) except that we now iterate the vertical structure using RADMC with the diffusion mode for treating the midplane, and not using the old RADICAL for that. This may cause some minor differences. Also beware that the Dullemond & Dominik (2004) models were based on vertical structure iteration. The template models have this also as a possibility, but if you switch this vertical structure iteration off and thus use just the parameterized density structure (which can be useful at times), then this is not the same as the Dullemond & Dominik (2004) models.

3. **The model shows that... (1)**
   Using a complex model such as 2-D/3-D radiative transfer models means that one must be careful in jumping too eagerly to conclusions. The models have many parameters and it is therefore very well possible that the same results can be obtained by different geometries. In fact, if the only information available is the spectrum (SED) then it is well known that many vastly different geometries give the same SED. Typically one can test if an idea works, but it is much more difficult to prove than some other idea does *not* work. Such a proof (if at all possible) requires the scanning of parameter space, requiring hundreds of models, and can then only prove that within the setup that you have made the fit does not work. Typically it is also prudent to be able to explain in physical terms why it is so.

4. **The model shows that... (2)**
   Often people like to use models in a black-box fashion, because we all are short on time and need to publish our results quickly. Radiative transfer modeling is, unfortunately, not very well suited for quick-and-dirty methods. We therefore urge you to always follow the check-list tips of Section 5.2, at least during the times that you still need to get familiar with the code, but it also never hurts to re-read that list even when experienced.

5. **Why is my dustdens.inp suddenly different?**
   If the vertical structure mode of RADMC is switched on, then RADMC will overwrite the dustdens.inp file that it has read in with the new result for the density structure. If you want to restart the model fresh then you must rerun the problem_setup.pro of the template models (or your own model setup routine).

6. **Accidental old input files that are still around**
   Certain input files are optional, such as external_meaning.inp or aperture.inp. If these files are present in the model directory, then they *will* influence the result. If the user wishes to make models without these things but forgets that they are still there, then this could produce wrong results. Usually the code(s) warn when they read in such optional input files, but it nevertheless can be a cause of problems. If in doubt, start with a fresh model directory, copy only the (template) IDL problem files (problem_.pro) into the new directory, and set up and run the model.

7. **Why does viewimage produce a widget, but does not work?**
   You might want type .r problem_compilecodes.pro if this happens. It could be that the RAYTRACE code has been recompiled by you for another problem, and is not up to date anymore for the model you want to view here.

5.2 A check-list for setting up a good model

1. **Never trust any feature or result of the code blindly**
   Often errors happen because some feature or output of the code(s) is simply used without checking. Potential sources of error in that case are a) there is an obvious bug in that feature and you don't notice it even though a simple check could have brought it to light, b) the code is fine but you misinterpret the physical units of some quantity or the true meaning of that quantity, or the meaning of this code feature altogether. **It is always good practice (for any code that is not your own that you will ever use) to not trust any feature or any output at face value. Always think of some simple tiny test problems to check if you understand its meaning.** The truth is that *any code feature* can be tested to the degree that you are 100% sure that you at least understand what it should do. Whether a subtle bug is there, that cannot be checked, but basic functionality can! Example: if you are not sure about the units or meaning of the spectrum.dat file, then the easiest thing to do is to make a model without circumstellar material and a) integrate the spectrum over frequency and compare this to the stellar luminosity it should be and b) overplot the spectrum over what you know the spectrum should be. If you are unsure what the meaning of 'inclination' is (from pole to equator, or equator to pole?) then simply make some spectra of simple disklike configurations and compare them to what you qualitatively think they should look like at different inclinations. If you are unsure of the meaning of the aperture input parameter, then take a simple disk model with a precise outer edge, and check if the flux goes down precisely when you put the aperture smaller than that radius.

2. **Check for expected behavior**
   Though radiative transfer can produce anti-intuitive results, there are always tests that one can do to check if certain basic behavior works in the way you expect. For instance, try to change the abundance of a certain species and one should see that the dust features of that dust component in the spectrum should decrease in strength. Of course, this kind of testing is done usually best when you have already much experience with the code, so you know what is natural and what is not.

3. **New code versions: test against older versions**
   As the code is being updated and features are added, bugs can inadvertedly enter the code. A careful user will – if possible – rerun some current models with an older version of the code, just for checking. Of course one does this only occasionally: for instance, at the beginning when using a brandnew version of the code and perhaps (if one is very careful) with one of the final models before publishing, just to make sure.

4. **Check the density distribution by eye**
   Analyze the density distributions that the template models (or your own modifications of them) produce. With the IDL subroutines in the analyze.pro file in the sources/idlroutines/ directory you can read the density distribution into IDL very easily. Make vertical and radial cuts, or even plot the $\log(\rho)$ as a surface plot or so, just to be sure that you know *what precisely* you are modeling. For models of disks it is also useful to analyze the vertical density plots (i.e. ρ plotted agains $\pi/2 - \Theta$) with the vertical pressure balance in mind. If no vertical structure is computed (nvstr=0 in radmc.inp) then it is just prudent to check a-posteriori if your disk is not much thicker than what pressure balance allows. If vertical structure is computed (nvstr$>$0 in radmc.inp) then it is useful to make a rough check that all went well. Near the midplane the density is typically a Gaussian:

$$\rho(R,z) = \frac{\Sigma(R)}{\sqrt{2\pi}H_p(R)} \exp(-z^2/2H_p^2) \tag{5.1}$$

where z is defined as $z \equiv \pi/2 - \Theta$ and $H_p = \sqrt{kTR^3/\mu m_p GM_*}$ with $\mu = 2.3$ for molecular hydrogen-helium mixture and M_* is the stellar mass, $m_p = 1.6726 \times 10^{-24}$ gram the proton mass, $k = 1.3807 \times 10^{-16}$ erg/K the Boltzmann constant, $G = 6.672 \times 10^{-8}$ the gravitational constant and T the midplane temperature in K.

5. **Luminosity conservation test**
   Before fully trusting a result, it is useful to make a flux conservation (=luminosity conservation) check. It takes a bit of CPU time, but is useful before publishing any results:

   ```
   nice raytrace glob nincl 45
   ```

   It is very strongly urged that this is done, in particular in the beginning phase of modeling and shortly before publishing. This test does not say everything, but it is certainly a test that must be passed or something is seriously wrong.

6. **Beware of the proper gridding**
   The grid must be fine enough in the surface layers of a disk or any optically thick object. Here are a few potential issues:

   - For a disk with inner edge at the grid inner radius this means that the radial grid must be strongly refined near this inner radius. In the template models this is already taken care of. The lower panel of the diagnostic diagram that pops up at the end of a problem_setup.pro run shows the cumulative optical depth at the inner few radial grid cells at the largest Θ-grid-point (i.e. at the equatorial plane). This curve should cross $\tau = 1$ gradually, i.e. that the cell that crosses $\tau = 1$ isn't a cell that has optical depth larger than unity. In other words: the innermost cell of the disk must be optically thin. Often the optical depth of a disk is so rediculously large at the inner edge that it would require an immense grid refinement which is both CPU time consuming and may get to the limits of the mathmatic precision of RAYTRACE. Therefore in the template models there is also something like smoothing of the inner rim. The philosophy here is that since this $\tau = 1$ layer is anyway so spatially thin (sometimes of the order of $\Delta R/R_{\rm in} \simeq 10^{-6}$), it does not affect the result at all if the density near the inner rim is a bit reduced to make this $\tau = 1$ zone a bit geometrically thicker so that it can be resolved by the grid. This does not affect the result because in continuum radiative transfer the results only depend on τ (i.e. colum depth) and not on the density rho.
   - If you implement additional gaps in the disk (e.g. a gap due to a planet) or if the inner edge of your disk is not located at the inner radius of the grid, then you must take care that the proper grid refinement in R is made near that alternative inner rim radius. Or one must make that rim so smooth that it even works with the given grid resolution.
   - The Θ grid must be fine enough in the region where the disk resides, but may be taken courser near the polar regions (unless of course your model is special near the pole, for instance a conical cavity in an envelope or so). In the template models the Θ grid has two regions: a polar region and an equatorial region. You can set the number of grid points in both regions and you can set the Θ value which separates these two regions. You typically take more grid points in the equatorial region if you have a disk configuration. A reasonable location of the separation between these two regions is such that the disk is just about entirely within the equatorial region. In this way you have the best use of your grid. The decisive criterion that defines the upper part of the disk surface layer is the location where $\tau_{*,\rm radial}(R,\Theta) \simeq 0.1$, where $\tau_{*,\rm radial}(R,\Theta)$ is the optical depth between the origin of the coordinate system and the point (R,Θ), at a wavelength typical of the stellar radiation (i.e. somewhere in the V band for the sun, but dependent on stellar type for other stars). In the template models the $\tau_{*,\rm radial}(R,\Theta) = 1$ surface is shown in the pop-up diagram (see Section 3.2.3). The entire region of the disk below this surface should be in the refined Θ grid. Note that once the vertical structure iteration is used (nvstr>0 in the radmc.inp file), the thickness of the disk will vary, so it might be prudent to check a-posteriori if the Θ gridding is still fine. In that case it might be useful to take the equatorial Θ such that it could encompass any to-be expected disk thickness. A bit of experimentation is usually the way to do this. Note that if one models a thin disk (e.g. a disk in which the dust has sedimented to the midplane), then one must make sure that the Θ resolution is high near the midplane. This usually means that one chooses the transition between the equatorial- and the polar grid regions to lie at rather large Θ (i.e. near the midplane).

7. **Resolution of central regions of rectangular images**
   When making rectangular images it is extremely important to realize that the unresolved disk regions as well as the star itself could be undersampled. Unfortunately, contrary to CCDs in real life, the pixels do not represent the flux in that pixel divided by the pixel size. Instead it is the intensity corresponding to exactly the center of the pixel. The flux from the central 2×2 pixels in an image therefore does not correspond typically to the flux from the entire central region of the disk covered by these 2×2 pixels. For the star flux there is a simple fix: by adding the word `addstar` to the raytrace command on the command line the star flux is added to the central 2×2 pixels in a bit an artificial way. But better is to take images at varying scales and make one big image from it like a babushka/matrioshka doll.

Chapter 6

What is new, what has changed

Typically the code development of RADMC and RAYTRACE are done in such a way that they are always fully backward compatible with previous versions. They should produce the same results on old problems as they do on new problems. However, very occasionally a slight improvement may be built in, to make the code more stable or more reliable. In that case the code is still backward compatible, but may produce slightly different (and hopefully better!) results for old test problems. Changes of this kind will be listed in the Section **??**. Changes that only affect new features of the code or changes that will only take effect when explicitly switched on through a keyword in the radmc.inp or raytrace.inp will be listed in Section 6.1. I hope that I will be complete, but this cannot be fully guaranteed.

6.1 Special changes

6.2 Big changes

- The meaning of nt
  In earlier versions of this setup ($<$ Version 3.0) the total number of Θ grid points was nt+ntex. Now this is nt. So the total number of Θ grid points in the equatorial zone is now nt-ntex instead of nt.
- The radmc.inp and raytrace.inp and chopdens.inp.
  These input files now have a different format (although the programs are still backward compatible with the old one). Now they are namelists, so they are much more readable.

  THIS PART OF THE MANUAL IS NOT YET READY