RADLite manual (version 1.1)

Klaus M. Pontoppidan (pontoppi@gps.caltech.edu)

June 15, 2010

1 RADLite philosophy

Welcome to RADLite. RADLite is a raytracer optimized for rendering large numbers of lines for dusty axisymmetric (2D) structures over, in principle, the entire thermal wavelength range UV-millimeter. It is *not* a stand-alone code, as it requires the input of an axisymmetric dust temperature structure. Rather, it is designed to take advantage of the general availability of dust radiative transfer codes, and does not attempt to re-invent the wheel. Currently, RADLite is a fully functional backend to the well-tested and much used dust code RADMC by C. P. Dullemond. However, any axisymmetric dust code can provide input to RADLite, although doing so will probably require that a simple interface is developed in order to produce a dust temperature and scattering source function input array in the correct format. I imagine that RADMC will serve the needs of nearly every RADLite user, in particular since it is a freely available code.

RADLite is essentially a highly stripped-down version of the very general code RADICAL (Dullemond & Turolla, 2000), but with certain fundamental additions that aid in the speedy rendering of line spectra and images. RADICAL itself also spawned the code RAYTRACE, which produces continuum SEDs and images with input from RADMC. In this way, RADLite can be thought of as a sister code to RAYTRACE, only more general, since it also includes lines.

1.1 When to use RADLite?

There are at least two distinct approaches when it comes to modeling line observations in astrophysics, in particular molecular line modeling: 1) Some like to understand chemistry and line/dust excitation processes on a fundamental level and may develop highly advanced and elaborate models that attempt to predict properties of various astrophysical objects – for instance, protoplanetary disks. Because they are so elaborate, such codes tend to be very specialized for a specific geometry or set of parameters, and may be difficult and cumbersome to use for someone who just wants to interpret their set of observations. 2) Others, usually more observationally based, want to place their data into a relative context, perhaps by deriving gas temperatures, determine average abundances, etc of a sample. Such studies may be critized for being over-simplified, for instance, not taking into account complex geometries when deriving basic parameters from unresolved spectra. RADLite is a code that may be used to bridge those two approaches. Given a chemical code, RADLite will be able to rapidly render spectra that can be used to predict observations. Conversely, given data, RADLite is fast enough that it

can be used to search a parameter space to fit abundances and abundance structures that in turn can be used to test chemical models.

1.2 RADLite in the infrared

There are other codes that accomplish similar tasks as RADLite, but these are generally optimized for use with submillimeter/millimeter molecular lines. One example is SKY (as part of RATRAN) by Michiel Hogerheijde and Floris van der Tak. RADLite, on the other hand, was developed for use in the infrared, and therefore includes are more complete treatment of dust absorption/emission and scattering. It also takes explicit steps to deal with a very wide range in velocity space in a stable way. For instance, for protoplanetary disks in the infrared, one may have emission components that are 100 km/s wide, but with absorption components that are ¡1 km/s wide. Since radiative transfer is generally a highly non-local problem, this can cause problems in codes that do not explicitly try to deal with gridding issues in structures with large velocity gradients. More details on what this means can be found in Pontoppidan et al. (2009), and there is no reason to repeat it here. Finally, the infrared spectra of, in particular, molecules are very rich. Water has 1000s of strong lines in the 3-200 μm region. RADLite is designed to be able to efficiently render that many lines and to construct the resulting line+dust spectra.

1.3 Line excitation

RADLite is not an excitation code. It is a raytracer, so will need input level populations to run. It is straight forward to set level populations to LTE, using either the dust temperature from RADMC, or an input gas temperature. This can be done using the built-in RADLite setup script. If the user wishes to calculate the detailed balance level populations, this will have to be supplied by an external excitation code in a single file with a specific format. I envision that use can be made of the various existing excitation codes by the development of various interfaces. Thus far, we have an interface to the the excitation code beta3D as described in Meijerink et al. (2009). This is an aspect that we are actively working on, so stay tuned. For now, RADLite is a most effective tool for thermalized lines.

1.4 Caveat emptor

As these things tend to be, RADLite is developed on borrowed time. So while I try to make it as stable and user friendly as possible, I expect most users to run into some problems getting it to run properly. I have only tested the code for a small range of applications and platforms. There are a number of known bugs, usually the kind that causes the code to crash, not the kind that produces wrong results – the code is so far passes the benchmark tests that I have been able to do (more details in Pontoppidan et al., 2009).

2   Installation

I am running RADLite on Intel Mac platforms, and I am using the Intel Fortran compiler. So I do not yet consider the code "portable", because it has not been tested yet on other platforms. For a while, the main limitation was the non-availability of a 64-bit Gnu compiler, but since that seems to have been solved, it should be more portable.

Installation procedure:

- First make sure that you have a 64 bit processor and a 64 bit Fortran compiler. You will also need a reasonably new version of IDL (¿6.0). RADLite should run fine on a 2 core Intel processor, but 4 cores are recommended for optimal performance. You will also need at least 2GB of memory, 8GB recommended. Finally, you will need the full IDL astronomy library installed to get the most out of the post-processing IDL scripts: `http://idlastro.gsfc.nasa.gov/`

- Install RADMC as instructed on the RADMC website: (`http://www.mpia.de/homes/dullemon/radtrans/radmc/`) Note that you will only *need* the RADMC executable, not the setups that come with RADMC. You could use those just fine, but RADLite comes with some ready-to-use RADMC setups, so no need to get them mixed up! Once you are more familiar with how everything works, you can experiment with adding your own setups.

- gunzip and untar the RADLite tarball in the folder of your choice:

  `gunzip RADLite_XXX.tar.gz`
  `tar -xvf RADLite_XXX.tar`

  This should make a main folder called `RADLITE_VX.X`

- If you have an Intel Mac, you can skip the next two compilation steps. The RADLite distribution comes with a pre-compiled executable for Macs.

- Go to the Fortran RADLite folder: `RADLITE_VX.X/RADLITE/` and edit the Makefile to use the compiler you want. You will have to uncomment the appropriate compiler line and comment the others. Right now you can choose between ifort and gfortran.

- Now compile RADLite by typing `./INSTALL` from the `RADLITE_VX.X/RADLITE/` folder. This should generate two executables, one for spectroscopy, and one for imaging.

- RADLite currently comes with two example model setups, one for a protoplanetary disk, and one for an infalling Shu envelope. Each has its own subfolder in the main `RADLITE_VX.X/` folder. Head to the `RADLITE_VX.X/RUN_EXAMPLE_PPDISK/` folder. Each model run folder contains two input parameter files; one for RADMC and one for RADLite. For historical reasons, the RADMC input file is called `problem_params.pro` (this is the same for any regular RADMC run). At the end of the `problem_params.pro` file there are a number of paths that you may need to edit to match your preferences. At a minimum, you will need to point to the Kurucz folder in the RADLite main folder if you want to include Kurucz input stellar spectra. Most users will only need to make this change. The RADLite input parameter file is called `line_params.ini`. Open this in your editor and edit the 'main_path' keyword to point to your `RADLITE_VX.X` folder.

- Finally(!), you need to add the folder containing all the RADLite IDL scripts to your IDL path. You can do this in your .cshrc or .bashrc, depending on your shell. My .cshrc line is: `setenv IDL_PATH $IDL_PATH:+/Users/pontoppi/WORK/RADLITE_V1.1/PRO`. Don't forget to source the .cshrc for the change to take effect!

- That's it! You should now be good to go.

3 Running the code

If you are already familiar with RADMC, the way RADLite functions should be familiar. RADLite takes the place of RAYTRACE to generate spectra and images. The normal sequence of a model run is:

- You will always run the model from within the run folder. In our case, that would be RADLITE_VX.X/RUN_EXAMPLE_PPDISK/. Now start IDL.

- The RADMC setup is generated using:
  IDL> .r problem_setup

- You can now exit idl and run RADMC, or simply run it from within IDL using:
  IDL> $radmc

- Once RADMC has completed its run, you are ready to actually do something with RADLite. You should now edit the parameters in line_params.ini to suit your needs. There are a lot of parameters, and not all are really useful, a few are no longer used at all. However, the parameter file is commented, so most things should be self-explanatory. One important technical parameter is the number of processor cores to use ncores. Obviously, this is system-dependent, and you do not want to use 3 cores if you only have 2! Essentially, this is a highly simplified parallelization. Modern operating systems running on multiple-core processors will assign processes to different cores to even out the load. This means that on a 4 core system (like my own), I can start three simultaneous RADLite processes that will automatically each run on a separate core, leaving one core free for other tasks (such as reading the news while I wait for my run to complete). RADLite is thus made to start, say, 3 parallel processes at any one time. Each process will render subN lines. Each process resulting in the rendering of subN lines is called a subrun. The specific lines that are rendered is determined by setting a wavelength range (using keywords min_mu and max_mu), as well as which molecule to actually render by specifying the hitran isotope code with isot. If the total number of lines is 98 and subN is 20, RADLite will generate 5 separate runs; 4 with 20 lines and 1 with 18. With 3 cores made available, these runs will take up 3 cores for the first three runs, followed by 2 cores for the final runs.

- RADLite is executed using:
  IDL> .r line_run
  IDL> line_run, run_name='my_run'

  RADLite will now render lines in the specified sequence. When complete, IDL will issue a stop statement. This allows the user to check whether all processes are really complete (I use a top command to check on any active process). When you are sure that everything is finished, type:
  IDL> .c

  This will copy all run files into a time-stamped sub-folder in the run directory beginning with my_run (or whatever you specified when you ran line_run. That way, you will not be overwriting previous runs.
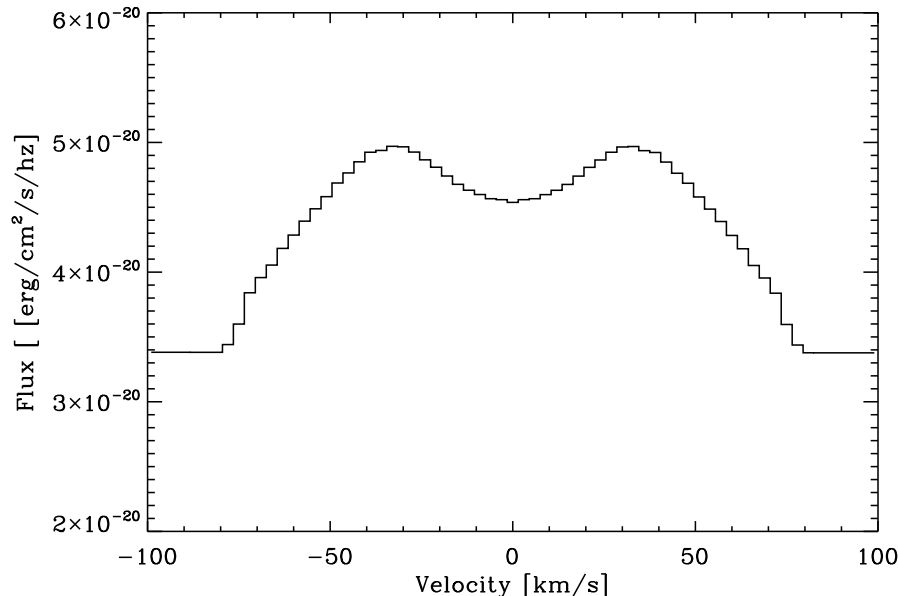
Figure 1: CO $v = 1 - 0$ P(7) line for the example protoplanetary disk as it appears when generated by RADlite and read by `read_line`.

4   Inspecting the result and generating spectra

Congratulations! When you reach this point, you have in fact rendered your first lines. However, they are at this point stored as ascii tables in velocity space, making them hard to visualize. There is an IDL script called `read_line()`, which allows you to read a set of lines into an IDL structure. Note that RADLite as default generates lines for a source at a distance of $D = 1$ pc. This makes it easy to scale the fluxes to whatever distance the user wishes using $\text{Flux}_{\text{scaled}} = \text{Flux}_{\text{RADLite}}/\text{D}^2$. The lines are rendered in velocity space, and so that is what `read_line` will return. I recommend that you always use `read_line` to inspect the lines you have created.

However, what you probably want to do, eventually, is to generate a full spectrum that can be compared directly to observations. To do this, you need to combine the lines. The way this is done is simple, but makes a significant assumption: Namely that lines do not interact. In the current version of RADLite, lines are rendered separately, so very densely spaced lines may not combine correctly. Currently, it is up to the user to determine when this is a problem. NOTE: Lines often overlap without interacting. In protoplanetary disks, emission lines in fact rarely interact. This only happens when the line spacing is less than the local line broadening, so for lines with rest frequencies that lie within a few km/s of eachother. This aspect is further discussed in Pontoppidan et al. (2009). My hope is that line overlaps can be included in the future, but for now, beware of this issue.

Once the RADLite run is complete, you can generate a full spectrum using the idl command:

```
IDL> genspec
```

Note that `genspec` right now requires that the line spectrum is wider than $\sim$1000 km/s, so do not try it for really narrow spectral ranges. The `genspec` tool allows for the simulation of various instrumental parameters, such as spectral resolving power and sampling, distance to the source (in parsec) and addition of noise. It also allows a user to combine lines from several different runs, with certain restrictions, using the `multiruns` keyword; this is a list of paths, e.g. `multiruns=['path1','path2','path3']`. Applications of `multiruns` can be to combine isotopologues of CO, or ortho- and para-water.

Running `genspec` results in the generation of a plot (in an .eps file), as well as a .fits file containing (in a .fits extension) the full rendered spectrum (called 'model.fits'). The model spectrum can be read into IDL using:

`IDL> spectrum = MRDFITS('model.fits',1)`

`spectrum` is now an IDL structure, which we recall can be inspected in IDL using:

`IDL> help, spectrum, /str`

5   Imaging with RADLite

As the spatial resolution of telescopes improve, in particular in the infrared, you may wish to figure out whether your line emission may be extended. RADLite provides the ability to generate image cubes and combine them with a simulated telescope to predict what a line imaging observation may look like. In fact, the spectra are generated by internally integrating image cubes. The images are not rendered with regularly spaced pixels, in the way that most people think of an image. Rather, the images are rendered using a two-dimensional circular grid with the same radial spacing as the input density grid. For instance, the RADMC grids are typically logarithmically spaced in the radial direction because this allows for an adequate sampling both the inner and outer regions of a protoplanetary disk. This is good for accurately calculating lines, but is not directly translatable to a computer screen. Imagine a typical disk around a T Tauri star with an inner radius of 0.1 AU and an outer radius of 100 AU. to properly sample the image of the inner disk, it may be necessary to use 0.01 AU pixels (or smaller!), resulting in a total image size of $>$10,000^2 pixels. The vast majority of those pixels would simply oversample the outer disk.

Hence, RADLite is equipped with resampling scripts that can convert the so-called 'circular images' to rectangular (flux-conserved) images. There are also tools available that can, with limitations, generate images as viewed by a telescope of a given size and instrument with a given sensitivity. The resampling tool is called `imcir_conv`. This IDL script will generate .fits image cubes that can be inspected with your favorite image viewer: ds9, skycat, gaia, etc. `imcir_conv` is great for inspecting your model, and also to check if there are sampling problems when rendering spectra; while RADLite tries to avoid some sampling pitfalls automatically, it is not failsafe, and still requires due diligence!

5.1   Running `imcir_conv`

First you must generate a circular image cube of your line(s). This is done by running RADLite with `image = 2` in line_params.ini. The only difference between this and the spectral rendering `image = 0` is that the circular image is saved to a file called `lineposvelcirc_moldata_X_Y.dat`, where X is the subrun number and Y is the line within the subrun. For the circular images, a separate file is generated for each line, as opposed to a spectral run in which a single file is generated for each subrun. The reason that the circular image files are not always saved

is simply that they are quite large (20-30 Mb per line), which could be an annoying problem when rendering a large number of lines.

An image can be generated by running `imcir_conv` on the relevant `lineposvelcirc` file. Say, we want to generate an image cube of the inner 0.5×0.5 AU with a sampling of 0.0005 AU and save the result in the file "test.fits", it can be done like this:

```
imcir_conv, 'run_Jan_1_22:34/lineposvelcirc_moldata_0_1.dat',$
xs=0.5, ys=0.5, nx=1000, ny=1000, saveit='test.fits'
```

`imcir_conv` will also plot the result to a postscript file using the `plotit` keyword:

```
imcir_conv, 'run_Jan_1_22:34/lineposvelcirc_moldata_0_1.dat',$
xs=0.5, ys=0.5, nx=1000, ny=1000, /plotit, scale=20
```

The -51 km/s channel intensity map as plotted by `imcir_conv` is shown in Figure 2 (the `scale` keyword is used to change the cuts for the scaling of the image). The astute reader will notice that the line emission has a little "rough" appearance. This is due to interpolation errors from the use of a 2D interpolation scheme on a grid that does not have a very fine ray (circular pixel) sampling (`imcir_conv` uses IDL TRIGRID). In general, images will require a significantly finer ray sampling than pure spectra. The user is invited to experiment, but will likely find that while a given ray sampling produces nice lines, images need a little more TLC[1]. You can ask `imcir_conv` to plot the rays on the images using the `plgrid` keyword. This is great for troubleshooting:

```
imcir_conv, 'run_Jan_1_22:34/lineposvelcirc_moldata_0_1.dat',$
xs=0.5, ys=0.5, nx=1000, ny=1000, /plotit, /plgrid
```

The result of this command is shown in the right panel of Figure 2. Each white dot represents the location of a ray, and consequently the calculation of a local intensity in units of $\mathrm{erg\,cm^{-2}\,s^{-1}\,Hz^{-1}}$ sterad. The rays must resolve local structure in the intensity map in order for the total flux to be correct. Plots such as those in Figure 2 allow the user to check for undersampling problems. Note that only the rays that include line emission are included. This is on purpose, as there is no need for RADLite to recalculate rays that are pure continuum for each velocity channel.

The relevant parameters for sampling of the circular images are `cir_np`, `b_per_r` and `b_extra`. `cir_np` is the number of rays in each circle, `b_per_r` is the number of rays per radial grid point and `b_extra` is the number of extra ray circles inside the inner radial grid point. For instance, the sampling can be improved by increasing `b_per_r` from 1 to 3. The result is shown in Figure 3.

References

Dullemond, C. P., & Turolla, R. 2000, A&A, 360, 1187

Meijerink, R., Pontoppidan, K. M., Blake, G. A., Poelman, D. R., & Dullemond, C. P. 2009, ApJ, 704, 1471
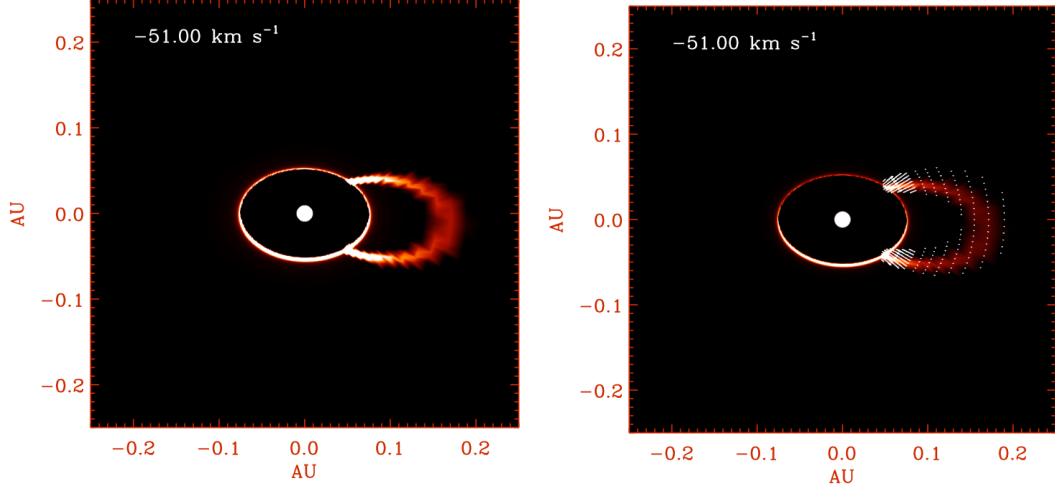
---

[1]Tender, loving care

Figure 2: CO $v = 1 - 0$ P(7) line for the example protoplanetary disk as it appears when generated by RADlite and imaged using `imcir_conv`. Left: Intensity map of the -51 km/s channel. Right: Intensity map with the ray configuration overlayed.
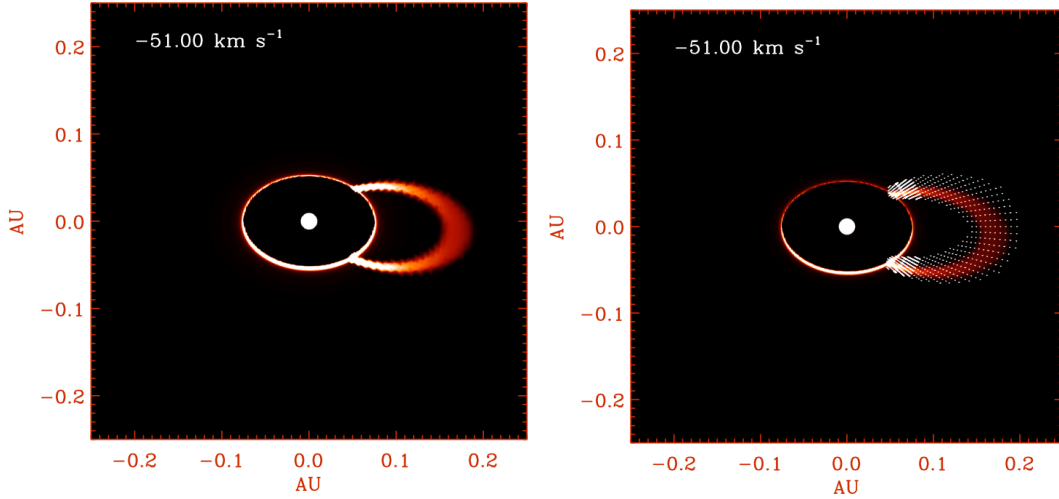

Figure 3: CO $v = 1 - 0$ P(7) line as shown in Figure 2, but with a finer radial sampling.

Pontoppidan, K. M., Meijerink, R., Dullemond, C. P., & Blake, G. A. 2009, ApJ, 704, 1482