Sedona6 User Guide

Oct 05, 2018

Contents:

CHAPTER 1

Getting Started

1.1 Installing the Code

- Set the environment variable `SEDONA_HOME` to point to the base directory of sedona. In bash, for example, you can add to your `.bash_profile` or `.bashrc` the line:

```
export SEDONA_HOME=/Users/kasen/sedona6/
```

- Install lua. Source code can be found in the `src/external/` directory
- Download and install the Gnu Scientific Library (gsl) available at https://www.gnu.org/software/gsl/
- Download and install hdf5 from https://support.hdfgroup.org/downloads/index.html
- Set the environment variables `GSL_DIR`, `LUA_DIR`, and `HDF5_DIR`. `GSL_DIR` and `HDF5_DIR` should be set automatically if you're on a supercomputer and load GSL and HDF5 modules. *LUA_DIR* needs to be set manually and should point to the root lua directory. For example, for me it's set to `$SEDONA_HOME/src/external/lua-5.2.1`.
- Makefiles are in the `makefiles` directory. If your machine doesn't have a `Makefile.machine` file, use `Makefile.general` as a template. You'll need to set `CXX` to the C++ compiler you want to use, and `CXXFLAGS` to the compiler flags.
- Run `chmod +x ./install.sh` to make `install.sh` into something runnable.
- Compile the code by running the command `./install.sh MACHINE` where `MACHINE` corresponds to the `Makefile.MACHINE` you want to use. This will copy all source files into `build` and compile.
- Other options for `./install.sh` are: * `help`: prints usage statement * `clean`: deletes source, object, and binary files in `build` * `realclean`: deletes entire `build` directory
- If compilation is successful, the executable file `sedona6.ex` will appear in the `src/` directory. Copy this to the directory where you would like to run the code.

Python is currently used for plotting and testing scripts, but is not needed to build and run **sedona** itself (NB: Python may eventually replace lua for parameter files). Python 2.7 is being used. On linux to get the necessary packages try:

```
sudo apt-get install python-numpy python-scipy python-matplotlib python h5py
```

1.2 Running the Code

To get started with **sedona**, you can try running a simple transport calculation. Copy the executable file *sedona6.ex* into a directory where you want to run, for example type:

```
cp sedona6.ex examples/simple_examples/spherical_lightbulb/1D/
```

Change to that directory, and run the code by typing:

```
./sedona6.ex param.lua
```

where *param.lua* is the file name of the run time parameter file (if no file name is given, the name *param.lua* is assumed). See *Setting Runtime Parameters* for description of setting runtime parameters. At minimum, the parameter file must point to three files

- A model file (an ascii .mod file for 1D runs or a hdf5 file for multi-D) giving the physical conditions (e.g., density, composition) of the setup. See *Input Model Files* for description of formats.
- An atomic data file in hdf5 format. Such files are available in the `data` directory, with data compiled from various sources.
- A defaults file, giving the default settings for all runtime parameters. The standard is ‘‘ defaults/sedona_defaults.lau‘‘, although users can point to their own modified defaults file.

The code generates several files. The *plt_.?????.h5* files contain data in hdf5 format describing the grid properties (e.g., density, temperature, radiation field). If hdf5 tools are installed, type:

```
h5ls plt_00001.h5
```

to see the file contents. For 1D models, a *plt_?????.dat* gives some of this data in ascii format.

The code also generates a *spectrum_?.h5* hdf5 file (and for 1D models a *spectrum_?.dat* asci file) giving the output spectrum.

You will find in the `tools/` directory a python library file *sedona.py* that provides functions to read and analyze the data in the plot and spectrum files (NB: this needs to be developed)

CHAPTER 2

Basic Code Execution

Several example setups for different sorts of science runs are given in the `examples/` directory. Simpler tests are in the `tests/` directory.

sedona everywhere uses cgs units.

2.1 Input Model Files

Model file set the density, composition, temperature at each

2.2 Setting Runtime Parameters

Runtime parameters are currently set in the param file, by default is assumed to be called **param.lua**

The parameter files uses the Lua scripting language, which allows for using math expressions and function calls (may be replaced with a python option). One can also grab environment variables using, e.g.,:

```
sedona_home    = os.getenv('SEDONA_HOME')
```

which sets the local variable **sedona_home** based on the environment variable **$SEDONA_HOME**.

Default values of all parameters are set in a **defaults** file, which is also in the Lua language. The **param.lua** file must point to the defaults file using, e.g.,

```
defaults_file     = sedona_home.."/defaults/sedona_defaults.lua"
```

This points to the standard defaults file provided with **sedona**. (N.B. the .. notation means string concatination in Lua)

Scalar parameters are set as e.g.,:

```
tstep_time_stop  = 100.0
```

String parameters (such as filenames) are set using quotes, e.g.,:

```
transport_module = "monte_carlo"
```

Vectors parameters are currently set as e.g.,:

spectrum_nu_grid = {start,stop,delta}

where gives a uniform vector between values **start** and **stop** with spacing **delta**. To use a logarithmically spaced grid, we add an extra entry of 1:

```
spectrum_nu_grid   = {start,stop,delta,1}
```

where now the spacing between points is dx = x*delta

2.3 Controlling Time Evolution

Calculations in **sedona** can be run either as time evolving or steady-state models. This is controlled by time-stepping parameters

Table 1: Time Stepping Parameters

parameter	values	definition
tstep_max_steps	<integer>	Maximum number of time steps to take before exiting
tstep_time_start	<real>	Start time (in seconds)
tstep_time_stop	<real>	Stop time (in seconds)
tstep_max_dt	<real>	Maximum value of a time step (in seconds)
tstep_min_dt	<real>	Minimum value of a time step (in seconds)
tstep_max_delta	<real>	Maximum fractional size of a timestep – restricts dt to the specified value multiplied by the current time

Times are always in seconds.

2.4 Output Files

CHAPTER 3

Radiation Transport

3.1 Controlling Transport

Table 1: Radiation Transport Parameters

parameter	values	definition
transport_module	"monte_carlo"	What method to use for transport. Currently only monte carlo is implemented.
transport_nu_grid	<float vector>	Define frequency grid used for transport and opacities
transport_radiative_equilibrium	0 = no | 1 = yes	Whether to solve for radiative equilibrium
transport_steady_iterate	<integer>	Do a steady-state calculation with this number of iterations

Note: The last parameter should really be under tstep, since it controls time evolution not transport per se.

3.2 Controlling Output

Spectrum files and writeout files

3.3 Interaction Processes

3.3.1 Electron Scattering

3.3.2 Compton Scattering

3.3.3 Resonant Line Scattering

CHAPTER 4

Radiation Sources

4.1 Radioactivity

4.2 Thermal Emission

4.3 Radiating Core

Table 1: Radiating Core Parameters

parameter	values	definition
core_n_emit	<integer>	Number of particles to emit from core per time step (or iteration)
core_radius	<real>	Radius (in cm) of emitting spherical core
core_luminosity	<real> or <function>	Luminosity (in erg/s) emitted from core
core_temperature	<real>	Blackbody spectrum of core emission, if using blackbody emission
core_photon_frequency	<real>	Frequency of photons emitted from core, if using monochromatic emission
core_timescale	<real>	?
core_spectrum_file	<string>	filename of file to read to set spectrum of core emission
core_fix_luminosity	0 = no | 1 = yes	In steady state calculations, will rescale to fix output luminosity

4.4 Multiple Point Sources

CHAPTER 5

Opacity

5.1 Atomic Data Files

Atomic data is store in hdf5 files

5.2 Opacity Settings

Table 1: Opacity parameters

parameter	values	definition
opacity_grey_opacity	<real>	value of grey opacity to use (in cm^2/g). Will overide all other opacity settings
opacity_electron_scattering	0 = no | 1 = yes	include electron scattering opacity
opacity_bound_free	0 = no | 1 = yes	include bound-free (photoionization) opacity
opacity_free_free	0 = no | 1 = yes	include free-free opacity
opacity_bound_bound	0 = no | 1 = yes	include bound-bound (resolved line) opacity

Though we use the word "opacity", the code actually calculates and stores an extinction coefficient. The two are related by

$$\alpha = \kappa\rho$$

where...

5.3 Electron Scattering Opacity

This is calculated as

$$\alpha_{\rm es} = \sigma_t n_e$$

where n_e is the free electron density

Options for include Klein-Nishina corrections, Comptonization etc...

5.4 Bound-Free Opacity

5.5 Free-Free Opacity

5.6 Line Opacity

There are various options for treating lines. In general, only one of these approaches should be used to treat lines, unless one can be certain that line opacity is not be multiply counted.

5.6.1 Resolved Line Profiles

This approach is selecting by setting the runtime parameter:

```
opacity_bound_bound = 1
```

This approach treats the lines generally as Voigt profiles. The frequency good must be fine enough that there are multiple grid points across to resolve the line profile.

The widths of lines can be artifically broadened using the runtime paramater:

```
line_velocity_width = <real>
```

where <real> is a velocity (in cm/s) by which the lines should be Gaussian broadened.

5.6.2 Line Expansion Opacity

This approach is selected by setting the runtime parameter:

```
opacity_line_expansion = 1
```

This approach bins lines into frequency bins, assuming a homologous flow.

This approach implies the Sobolev approximation. For each line, the code calculates the Sobolev optical depth

$$\tau_{\rm sob} = \frac{\pi e^2}{m_e c} n_l f_{lu} \lambda_0 t_{\rm exp}$$

where...

The expansion opacity is then calculated by binning lines

$$\alpha_{\rm exp} = \frac{1}{c t_{\rm exp}} \frac{\lambda_0}{\Delta\lambda} \sum_i (1 - e^{-\tau_i})$$

5.6.3 Fuzz Expansion Opacity

This approach is selected by setting the runtime parameter:

```
opacity_fuzz_expansion = 1
```

The physics here is identical to that of line expansion opacity, it is just that the lines are read from an independent file.

5.6.4 Resonant Line Scattering

This is how we would treat scattering in strong individual lines like Lyman alpha.

5.7 LTE and NLTE settings

CHAPTER 6

Hydrodynamics

6.1 Homologous Expansion

6.2 Lagrangian Hydodynamics

CHAPTER 7

Plotting and Data Visualization

7.1 Generating Synthetic Light Curves

Sedona default outputs a file called *spectrum.h5* that gives the time series of the light curve $L_{\{\nu\}}(\{t\})$ at frequencies $\{\nu\}$ and output times $\{t\}$.

The bolometric luminosity is simply given as

$$L_{bol}(\{t\}) = \int_{\{\nu\}} L_{\{\nu\}}(\{t\})\, d\nu$$

Similarly, the absolute bolometric magnitude is given as

$$M_{bol} = -2.5 \log_{10} L_{bol} + 88.697425$$

In order to get light curves in certain filters, you have to convolve it with a given transmission curve.

If $T_b(\nu)$ is the transmission for a given filter band at frequency ν, then the luminosity convolved with the filter is expressed as

$$\mathcal{L}_\nu(b) = \frac{\int T_b(\nu) L_\nu \, d\ln\nu}{\int T_b(\nu)\, d\ln\nu}$$

The formula to convert this to an AB magnitude is

$$M_{AB}(b) = -2.5 \log_{10}\left(\frac{\mathcal{L}_\nu(b)}{4\pi d^2}\right) - 48.600$$

where $d = 10$ pc is the standardized distance to convert to a flux.

Note that $T_b(\nu)$ is here expressed as an energy-counting response.

A script that generates synthetic light curves from the Sedona *spectrum.h5* file is provided in the directory *tools/lightcurve_tools*. The python program *lcfilt.py* takes as input a *spectrum.h5* file and a list of filters/bands and converts the raw spectrum output to light curves.

To run, simply call:

```
python lcfilt.py -s <spectrum.h5> -b <band1,band2,...>
```

where *spectrum.h5* points to the Sedona spectrum file, and *<band1,band2,...> is a comma-separated list of filters that you wish to make light curves in. For example, if I wanted to generate synthetic light curves of a file */path/to/supernova_spectra.h5* in the LSST bands, then one would call:

```
python lcfilt.py -s /path/to/supernova_spectra.h5 -b LSST_u,LSST_g,LSST_r,LSST_i,LSST_
↪z,LSST_y
```

The light curve table is outputted in the file *lightcurve.out* and gives the time, bolometric luminosity, bolometric magnitude, and absolute magnitudes in the specified bands, using the AB magnitude system.

A full list of filters can be accessed by calling:

```
python lcfilt.py --bands
```

or by examining the file *FILTER_LIST*, which also contains references for the filters.

To add a filter not provided, add an entry to the end of *FILTER_LIST* and append the transmission curve (in Angstroms and relative response) to the **end** of *allfilters.dat*.

CHAPTER 8

All Runtime Parameters

sedona uses cgs units everywhere

Table 1: Radiating Core Parameters

parameter	values	definition
core_n_emit	<integer>	Number of particles to emit from core per time step (or iteration)
core_radius	<real>	Radius (in cm) of emitting spherical core
core_luminosity	<real> or <function>	Luminosity (in erg/s) emitted from core
core_temperature	<real>	Blackbody spectrum of core emission, if using blackbody emission
core_photon_frequency	<real>	Frequency of photons emitted from core, if using monochromatic emission
core_timescale	<real>	?
core_spectrum_file	<string>	filename of file to read to set spectrum of core emission
core_fix_luminosity	0 = no | 1 = yes	In steady state calculations, will rescale to fix output luminosity

Table 2: Time Stepping Parameters

parameter	values	definition
tstep_max_steps	<integer>	Maximum number of time steps to take before exiting
tstep_time_start	<real>	Start time (in seconds)
tstep_time_stop	<real>	Stop time (in seconds)
tstep_max_dt	<real>	Maximum value of a time step (in seconds)
tstep_min_dt	<real>	Minimum value of a time step (in seconds)
tstep_max_delta	<real>	Maximum fractional size of a timestep – restricts dt to the specified value multiplied by the current time

Table 3: Radiation Transport Parameters

parameter	values	definition
transport_module	"monte_carlo"	What method to use for transport. Currently only monte carlo is implemented.
transport_nu_grid	<float vector>	Define frequency grid used for transport and opacities
transport_radiative_equilibrium	0 = no | 1 = yes	Whether to solve for radiative equilibrium
transport_steady_iterate	<integer>	Do a steady-state calculation with this number of iterations

Table 4: Opacity parameters

parameter	values	definition
opacity_grey_opacity	<real>	value of grey opacity to use (in cm^2/g). Will overide all other opacity settings
opacity_electron_scattering	0 = no | 1 = yes	include electron scattering opacity
opacity_bound_free	0 = no | 1 = yes	include bound-free (photoionization) opacity
opacity_free_free	0 = no | 1 = yes	include free-free opacity
opacity_bound_bound	0 = no | 1 = yes	include bound-bound (resolved line) opacity

CHAPTER 9

Indices and tables

- genindex
- modindex
- search