SpecWizard User Guide

Joop Schaye
schaye@strw.leidenuniv.nl

Tom Theuns
tom.theuns@durham.ac.uk

Craig Booth
booth@strw.leidenuniv.nl

minor user guide updates by Nastasha Wijers
wijers@strw.leidenuniv.nl

August 7, 2020

Contents

Chapter 1

Installing SpecWizard

Installing SpecWizard is not entirely straightforward, and depends somewhat on what system you are installing it. However, here are some general instructions and steps that should be helpful. The makefiles, etc., included in this package are for the Cosma system at Durham, though other examples are also included.

1. Choose which C and fortran compilers you want to use (CC and FC in the makefiles, respectively). This should be consistent throughout the process.

2. Load the modules you need, if required on your system: hdf5, compilers (C and fortran90, should include MPI). It's useful to make a note somewhere on which ones you used.

3. Install the hdf5 wrapper SpecWizard needs, available at e.g. `https://github.com/galtay/sphray/tree/master/hdf5_wrapper`. Installation notes for this package can found found in its README file. One ingredient required here is the path where HDF5 is installed; this needs to be consistent with the hdf5 module you loaded. (You will need to install this if it is not installed already.) You will have to look up this location, or ask someone who knows.

4. Compile the files in the `eagle` directory in the SpecWizard package: type 'make' in that directory. (Check that the compilers in the Makefile are consistent with the ones in the previous step.)

5. Check the Makefile and make.SYSTEM.COMPILER files: at the top of the Makefile, you need to choose a make.SYSTEM.COMPILER file to use. You will likely need to make your own version of a make.SYSTEM.COMPILER file. This file has a number of lines to pay attention to. First, make sure the fortran compiler FC is consistent with the one used in the `eagle` directory and the hdf5 wrapper. Then, you

need to provide the information to link the `hdf5_wrapper` compiled files. The lines starting with `ISLIB` to `HDFW_LIB` are added to `OPTHDFW` and `OPTHDF` in the Makefile. Set them so that those lines look like the `LINKFLAGS_HDFW` in `/test/Makefile` in the `hdf5_wrapper`.

6. There are some other compile options in the Makefile; not using the MPI version (no `-DMPI`) should work fine. I (Nastasha) have only used SpecWizard with the `EAGLE` and `AliVersion`, and `READREGION` options, and the User Guide in its current state assumes those options are used.

7. Compile SpecWizard: type 'make' in the specwizard directory. You may need a few tries for this to work, e.g. if the hdf5 linking went wrong, compilers are inconsistent, etc.

8. You will need two more ingredients to actually run SpecWizard: simulation data and ionization tables. EAGLE simulation data is publicly released, although the LOS files used for the long spectra were not included in this release. The ionization tables typically used in EAGLE analysis are not publicly available, but other ionization tables are, and can be put into the format expected by SpecWizard.

Chapter 2

How to Run SpecWizard

SpecWizard can run in either serial or parallel mode. To run in serial mode, just run the executable with an optional command line parameter that specifies the parameter filename:

```
./specwizard [mypars.par]
```

To run in parallel mode, ensure that the code is compiled with the -DMPI flag set, and then run it like a standard MPI program:

```
mpirun -np 5 ./specwizard [mypars.par]
```

Specwizard has recently been used (and tested) using short spectra with snapshots lines of sight specified in text files, and long spectra using random sightline segments from LOS files.

Chapter 3

Parameters

By default specwizard looks for an ascii file called `specwizard.par` in the current directory and reads parameters from there. If the command line argument is not present then SpecWizard searches by default for a parameter file in the current directory named `specwizard.par`. This section contains a list and description of the parameters. Where units may be important for a parameter they are shown on the right hand side of the page, the subscript $\odot$ represents solar values. If any parameters remain uninitialized then SpecWizard will crash on startup.

Note that this default-name parameter file is not present in this repository. The parameter list is not fully complete. This version of SpecWizard has been tested using EAGLE data only, with short spectra generated from EAGLE snapshots, using specified lists of sightlines, and long spectra using random lines of sight from EAGLE LOS (line of sight) files. I have not used density-metallicity relations (this is unnecessary since EAGLE tracks element abundances), nor the instrument simulation options (noise and gaussian PSF) or the 'urchin' options (which are, to the best of my knowledge, not available for EAGLE).

3.1 Basic Runtime Control

`do_long_spectrum` [LOGICAL]

If T then we generate a long spectrum by patching together spectra from different redshifts into one composite spectrum (long spectrum mode). If F then we generate a single spectrum along an individual LOS (short spectrum mode)

`nspec` [INTEGER]

Number of spectra to create. For short spectra this generates the first `nspec` spectra per LOS file. If there are insufficient sightlines in the LOS

file, nspec is reduced. For long spectra this is the total number of long spectra to generate.

`first_specnum` [INTEGER] 1

Number of the first spectrum to generate (should be ≥ 1). This is only used for randomly chosen sightlines. Its usefulness lies in the fact that the specwizard random number generator is seeded based on a (fixed) set of seeds plus the spectrum number. This means that (for long spectra, using the same redshift range, fzresol, etc.[1]) running specwizard with the same parameter file will generate the same spectra. Setting the first spectrum number is then useful to generate additional, different spectra. For short spectra, `first_specnum` - 1 is used and stored instead, because short spectra are numbered starting at 0, while long spectra start at 1 by default.

3.2 Files and Directories

`ibdir` [STRING]

Location of HDF5 file containing ionization fractions of each species as a function of redshift, density and temperature. The ion fractions for e.g. o8 should be contained in a file called o8.hdf5 in this directory.

`datadir` [STRING]

Location of the data files. See also `file_list`. OR if use_snapshot_file is true, this must be the directory containing the snapshot files (e.g. `<PATH>/L0100N1504/PE/REFERENCE/data/snapshot_027_z000p101/`).

`use_snapshot_file` [LOGICAL]

If this is true then instead of using LOS files, SpecWizard works with snapshot files

`snap` [INT]

If use_snapshot_file is true, then SW needs to know the snapshot number explicitly. (For EAGLE data, use `snap_base` instead.)

`snap_base` [STRING]

[1] What matters is that the zstart and zend variables, fzresol, and the LOS file list are the same. zstart and zend are the smallest redshift range given zabsmin, zabsmax, and the transition wavelengths and wavelength range.

If use_snapshot_file is true, and using EAGLE data, this is the part of the snapshot file names before the '.NUMBER.hdf5' part, e.g. `snap_027_z000p101`

`use_random_los` [LOGICAL]

If we are using a snapshot file then we have the option of specifying LOS coordinates. If this is T then LOS coordinates are random, if it is F then LOS coordinates are drawn from the file `los_coordinates_file`

`los_coordinates_file` [STRING]

An ascii file containing LOS coordinates for SW. The first line simply contains the number of coordinates in the file. The rest of the file has 5 columns: x y z phi theta. In Ali Rahmati's version (default in this branch), the file only has three columns: x, y, z (tab-separated). Lines of sight are always along the z axis, and z is assumed to be zero. The x and y coordinates are the (x, y)-postion of the sightline, in units of the box size. (So x and y are fractions of the total size of the box.)

`file_list` [STRING]

An ascii file called `file_list` must be present inside of `datadir`. This file must contain a list of LOS files, with one file per line. If this file is not present SpecWizard will crash. Update: this variable must contain the full path to the file containing the list of LOS files. It can be in `datadir` or somewhere else; both work. When using snapshot data, the value of this variable does not matter, and any dummy string will do.

`outputdir` [STRING]

Directory to write output data to

`output_frequency` [STRING]

If T then write frequencies to output files, if F then work with wavelengths. This is only relevant for long spectra, and in both cases, the spectra are evenly spaced in wavelength space.

`gimic` [LOGICAL]

If F then we are using OWLS simulations or Eagle; if T then we are using GIMIC

`wmap7` [LOGICAL]

The data is from an OWLS wmap7 run. The output do not have a few attributes the other OWLS files do have.

`overwrite` [LOGICAL]

If F then if the output file already exists SpecWizard will crash. If T then old files will be overwritten

`urchin` [LOGICAL]

If T then read the neutral fraction for H1 (Si2) from post-processed snapshot files rather than lookup the ionization balance. (Those files are not present for Eagle, to the best of my knowledge.)

`urchindir` [STRING]

If urchin: The directory containing the post-processed snapshot files

`use_urchin_temperature` [LOGICAL]

If urchin: If T then read the particle temperatures from post-processed snapshot files rather than use the simulation temperature

3.3 Output Control

`SpectrumFile` [STRING]

If stated, this output filename is used rather than automatic filename generation.

`output_zspaceopticaldepthweighted_values` [LOGICAL]

For both long and short spectra. Output, separately for each ion, redshift space quantities weighted by contribution to the optical depth. The quantities that are written out are: Optical depth of strongest (first listed) transition, overdensity, temperature (K), and line-of-sight peculiar velocity (km/s).

`output_realspacemassweighted_values` [LOGICAL]

Only for short spectra, output real space, mass weighted quantities along the sightline. The quantities that are written out are: LOS peculiar velocity (km/s), metal mass fraction, overdensity and temperature (K)

`output_realspacenionweighted_values` [LOGICAL]

Only for short spectra, output real space N_{ion} weighted values separately for each ion. The quantities that are written out are: LOS peculiar velocity (km/s), ion density (cm^{-3}), overdensity and temperature (K)

`verbose` [LOGICAL]

If true, more information is written to the log files, e.g. timing information for different functions.

3.4 Control of Ions

`ibfactor` [REAL]

Factor to rescale ionizing background with: $I_{UV} = I_{UV} \times$ `ibfactor` (only used for calculation of the ionization balance). This rescaling is achieved in practice by rescaling the hydrogen number density used to calculate the ion balance, since the ion balance tables are pre-calculated with a fixed spectrum.

`use_fitted_ibfactor` [LOGICAL]

Calculate a redshift-dependent factor to rescale ionizing background, used in the same way as `ibfactor`, but overriding its value. See `function get_fitted_ibfactor(z)` in `specwizard_subroutines.F90` for the functional form.

`ibfactor_he_reionization` [LOGICAL]

Only used if `use_fitted_ibfactor` is true. This parameter sets the functional form for the rescaling factor as a function of redshift. See `function get_fitted_ibfactor(z)` in `specwizard_subroutines.F90` for the two options.

`doH1` [LOGICAL]

If T then consider H1 when calculating spectra. Other ions are switched on and off using the other flags: e.g. `doHe2`, `doC2`, `doC3`, `doC4`, `doN2`, `doN5`, `doO1`, `doO6`, `doO7`, `doO8`, `doNe8`, `doNe9`, `doMg2`, `doSi3`, `doSi4`, `doFe2`, `do21cm`. All of these parameters must be set (n.b. He2 = HeII=He^{+}). See `specwizard_modules.F90`, `module atomic_data` for the available ions, as well as the atomic data used for the transitions.

`doall` [LOGICAL]

If T then override all other ion flags and force SpecWizard to consider all currently implemented ions

`subtract_Hmol` [LOGICAL]

If urchin is T, then this option controls how to interpret the ionic fraction of Si2. Either (subtract_Hmol=T) X_Si2 = X_H1 or (subtract_Hmol=F) X_Si2 = X_H1 + X_Hmol

`setmaxt4sfgasl` [LOGICAL]

If true, set the temperature of all star-forming gas to 10^4 K.

`ignore_starforming` [LOGICAL]

If true, assume star-forming gas contributes no ions to the spectrum. (The mass of the particles is set to 0 in the calculations.)

`ionfracone` [LOGICAL]

Debugging option. If ion fractions are calculated from tables according to the other parameters, use a value of 1 for all these fractions instead.

`impose_eos` [LOGICAL]

If true, impose a power-law equation of state on the IGM: gas at temperatures $< 2 \times 10^4$ K and overdensity $<$ `imposed_eos_maxod` has its temperature set to `imposed_eos_T0` $\times$ overdensity$^{\texttt{imposed_eos_gamma}-1}$

`imposed_eos_T0` [REAL]

If `impose_eos` is true, this is the temperature to set overdensity $= 1$ gas to.

`imposed_eos_gamma` [REAL]

If `impose_eos` is true, this sets the slope of the density-temperature relation: $T \propto$ overdensity$^{\texttt{imposed_eos_gamma}-1}$

`imposed_eos_maxod` [REAL]

If `impose_eos` is true, this is the maximum overdensity for which the IGM equation of state is imposed.

3.5 Metal Abundance Modification

`modify_metallicity` [LOGICAL]

If this is true then modify the simulation metal abundances, in one of a number of ways described by the parameters in this section. If this is F then all further variables in this section are ignored, and the simulation abundances are used

`use_smoothed_abundance` [LOGICAL]

If true, reads the smoothed element abundance in place of the true element abundance. This will not work for the Eagle line-of-sight files, which do not contain smoothed abundances.

`maxz_rel` [REAL] $Z_\odot$

If either `impose_z_rho_relation` or `log_normal_scatter` are T, then limit the maximum metallicity with this parameter. $Z_\odot$ is hardcoded into the source file specwizard_modules.F90, and has a value of 0.0126637 (=$M_{Metal,\odot}/M_{tot,\odot}$

`scale_simulation_abundances` [LOGICAL]

Scale simulation metallicity by a scalar factor `z_rel`

`z_rel` [REAL] $Z_\odot$

Scalar factor by which to scale metallicity, used only if `scale_simulation_abundances` is T. $Z_\odot$ is hardcoded into the source file specwizard_modules.F90, and has a value of 0.0126637 (=$M_{Metal,\odot}/M_{tot,\odot}$

`ZC_rel` [REAL]

Scale carbon abundances by this linear value relative to their original abundance. Other metals are scaled by the parameters: `ZN_rel`,`ZO_rel`,`ZNe_rel`,`ZMg_rel`,`ZSi_rel`,`ZFe_rel`. All of which must be set. Used only if `modify_metallicity` is T. These parameters are used to change the relative abundances of different elements.

`impose_z_rho_relation` [LOGICAL]

impose metallicity $z = \texttt{z\_mean}(\rho/\rho_{mean})^{\texttt{z\_index}}$, up to maximum metallicity `maxz_rel`, ρ_{mean} is the mean baryonic density of the Universe

`z_index` [REAL]

Power-law index of ρ-Z relation, used only if `impose_z_rho_relation` is T

`z_mean` [REAL] $Z_\odot$

Metallicity at mean density, if we are imposing a ρ-Z relation, used only if `impose_z_rho_relation` is T

`log_normal_scatter` [LOGICAL]

If true, Divide computational volume in (`z_sig_bin`)^3 cells, and add lognormal metallicity with `z_sig_dex` scatter to particles in each cell

`z_sig_bin` [INTEGER]

Number of cells to divide the computational volume into, if we are imposing a lognormal metallicity scatter, used only if `log_normal_scatter` is T

`z_sig_dex` [REAL]

If we are adding a lognormal metallicity scatter to each particle, then in solar units, $\log(Z) \rightarrow \log(Z) + \Sigma$`z_sig_dex`, where Σ is a Gaussian deviate with mean 0, standard deviation=1, which is selected independently for each of the (`z_sig_bin`)^3 cells. Used only if `log_normal_scatter` is T

`read_part_ids_from_file` [LOGICAL]

If true, it appears simulation metallicities should be rescaled according to `ZC_rel`, etc. This seems to be intended to be applied to a set of flagged particles read in from a file, but the `particle_file_name` does not seem to be read in or defined anywhere.

`flagged_particle_metallicity` [REAL]

Unused or deprecated.

3.6 Technical options

`NoPecVel` [LOGICAL]

If true, set all peculiar velocities to zero (from the start) in the spectrum calculation.

`use_gaussian_kernel` [LOGICAL]

If true, assume a (3D) Gaussian gas distribution for each SPH particle. (Used for calculating ion densities along the line of sight from the number of ions per SPH particle.). If false, a cubic spline is used instead.

`integrate_kernel` [LOGICAL]

If true, the ion density in each cell along the line of sight is calculated by integrating the kernel over the line of sight in the cell. If false, the density is calculated by evaluating the kernel function at the cell center. This is currently only implemented for the Gaussian kernel option (`use_gaussian_kernel` is true).

`add_turbulence` [LOGICAL]

If true, absorption lines get turbulent broadening on top of the thermal broadening. This is only true for gas above the star formation density threshold, and the turbulence values depend on the density and the imposed equation of state. (I'm not sure exactly how the threshold density parameter is defined in EAGLE, since the threshold is depends on metallicity there.)

`do_periodic` [LOGICAL]

Deprecated; should not do anything.

`use_maxdens_above_zmax` [LOGICAL]

For redshifts above the highest redshift in the ion balance tables, calculate ion balances using the highest-redshift table, assuming all gas is at the highest tabulated density (i.e., assuming collisional ionization equilibrium). If false, SpecWizard will crash if the redshift is outside the bounds of the ionization tables.

3.7 Noise Statistics

`generate_noise` [LOGICAL]

Generate a noise array? If this is true then in addition to the 'Flux' variable written for each spectrum (which always contains a noise-free spectrum), there are two additional arrays written out: 'Noise_Sigma' (standard deviation of the noise at each pixel), and 'Gaussian_deviate' (mean=0, sigma=1 Gaussian random number for each pixel)

`use_noise_file` [LOGICAL]

If T then use file describing sigma as a function of flux and wavelength. If F then use `sigtonoise` and `minnoise` to generate Gaussian noise. This option works in long spectrum mode only.

`noisefile` [STRING]

If we are using noise from a file, load in `noisefile`, an HDF5 file that describes the standard deviation of the noise as a function of wavelength and flux. Note that this noise file can be created from observations using the `noisestat.pro` IDL script included in the `Noise/` subdirectory of the SpecWizard distribution.

`sigtonoise` [REAL]

If `use_noise_file` is false then this is the signal to noise ratio for the Gaussian noise: sigma = `minnoise` + (1/`sigtonoise` - `minnoise`) * flux

`minnoise` [REAL]

If `use_noise_file` is false then this is the minimum noise level, normalized to the continuum, for the Gaussian noise: sigma = `minnoise` + (1/`sigtonoise` - `minnoise`) * flux

3.8 Accuracy Parameters

`minbother_blue` [REAL]

Accuracy parameter, minimum optical depth to consider for transitions with rest-frame wavelength shorter than that of HI Lyα. The smaller this number the more accurate the results, but the slower the code will run. This is the maximum optical depth allowed to be missed in a short spectrum pixel.

`minbother_red` [REAL]

Accuracy parameter, minimum optical depth to consider for transitions with rest-frame wavelength longer than that of HI Lyα. The smaller this number the more accurate the results, but the slower the code will run. This is the maximum optical depth allowed to be missed in a short spectrum pixel.

`limsigma` [LOGICAL]

Accuracy parameter. If true, the line profile is only calculated over the velocity range where the optical depth is above `minbother` divided by the number of pixels. This means that, at most, a total amount of absorption `minbother` is missed in each pixel. (The minbother results still affect the spectra if this is False: they also determine whether the maximum optical depth from a real space pixel is large enough to bother adding to the optical depth spectrum.)

`vpixsizekms` [REAL] km/s

Pixel size in km/s. For short spectra this represents the final output pixel size. For long spectra this represents the pixel size before rebinning into wavelength. SpecWizard will stop if this is too small compared with the other parameters. The smaller this number the more accurate the results, but the slower the code will run. Note that this also sets the resolution at which the position-space ion densities and ion-weighted temperatures and peculiar velocities are calculated. The shape and width of the spectral features can therefore be affected if this value is too low.

`integrate_thermprof_exactly` [LOGICAL]

When calculating optical depths, if true, the Gaussian thermal profile is integrated between the bounds of the cell in velocity space. If false, the Gaussian is evaluated at the cell center.

3.9 Parameters For Long Spectrum

`zqso` [REAL]

Redshift of the QSO

`minlambda` [REAL] Å

Minimum observed wavelength in final spectrum

`minlambda` [REAL] Å

Maximum observed wavelength in final spectrum

`zabsmin` [REAL]

Minimum allowed absorption redshift

`zabsmax` [REAL]

Maximum allowed absorption redshift. If `zabsmax` > `zqso` then `zabsmax` is set equal to `zqso`

`nlyman` [INTEGER]

Number of Lyman lines to include (1 = Ly-alpha, 2 = Ly-beta, etc.; neg. value = use nlyman_all = 31). Note that this also sets a maximum for other ions, so if you add many transitions for a different ion, you might also need to modify this.

`fzresol` [REAL]

Bin size for simulation LOS files: dz = fzresol*(1+z). Sight lines are drawn from files with z_file = z +/- `fzresol`(1+z)/2, where z is the current redshift.

`pixsize` [REAL] Å

Pixel size of final spectrum

`do_convolve_spectrum` [LOGICAL]

If T then convolve final spectrum with an instrumental broadening of `fwhm` km/s

`fwhm` [REAL] km/s

FWHM of instrumental broadening (Gaussian), used only if `do_convolve_spectrum` is T

Chapter 4

Structure of Output Files

Output files are HDF5, and so are organized hierarchically. At the top level there are four header groups (Units, Constants, Parameters, Header).

Units Conversion factors between original SPH simulation units and cgs. Whilst using SpecWizard you should not need anything in this group!

Constants Physical constants

Parameters Parameters from specwizard.par

Header The Header data from the simulation. Contains information on cosmology, etc.

At the top level there are in addition to the four header groups one additional group for each spectrum (for spectrum number N this is called `SpectrumN`), and a dataset that contains either Wavelengths (`Wavelength_Ang`. for long spectra), or Hubble expansion velocities, corresponding to a physical coordinate (`VHubble_KMpS` for short spectra). The contents of the individual spectrum groups will be treated separately for short and long spectra:

4.1 Short Spectra

Each spectrum group contains a sub-group for each ion (e.g. `/Spectrum1/c4/`), this group contains a scalar dataset (`LogColumnDensity`) containing the integrated column density of that ion, and an array, (`Optical Depth`), containing the optical depth of that ion as a function of hubble velocity.
If `output_realspacemassweighted_values` is true then each spectrum group contains a sub-group called `RealSpaceMassWeighted`, containing real space physical properties (i.e. as a function of Hubble velocity), weighted by mass. These arrays are

LOSPeculiarVelocity_KMpS

MetalMassFraction

OverDensity

Temperature_K

Note that the OverDensity is not the mass-weighted density in each pixel, but the density itself, used as a weight for the other datasets.
If `output_zspaceopticaldepthweighted_values` was set to true then there is an additional subgroup called `RedshiftSpaceOpticalDepthWeighted`, which contains

OverDensity

Temperature_K

LOSPeculiarVelocity_KMpS

These are arrays of densities and temperatures, with particle properties weighted by the optical depth they contribute to each pixel. Note that this is weighted by the optical depth of the strongest transition only. If `output_realspacenionweighted_values` is set then there is another additional subgroup, called `RealSpaceNionWeighted`, which contains the following arrays:

LOSPeculiarVelocity_KMpS

NIon_CM3 (in the ion group, not `RealSpaceNionWeighted`)

OverDensity

Temperature_K

These are real space quantities, weighted by Nion. If either category of real-space quantities is output, an additional array is stored: Redshift_RealSpace, which indicates the redshift (assuming no peculiar velocities) at which the mass- and ion-weighted quantities are calculated. Warning! If all optional output is generated then output dataset can become very large!

4.2 Long Spectra

Each spectrum group contains an array, `Flux`, containing the total normalized, transmitted flux as a function of wavelength. If noise has been specified then two additional arrays are present, called `Gaussian_deviate` and `Noise_Sigma`. An additional group, called `ShortSpectraInfo` contains information so that the exact lines of sight used from the LOS files

may be looked up. Finally, the spectrum group contains a sub-group for each individual ion, if `output_zspaceopticaldepthweighted_values` is true then this group contains a subgroup `RedshiftSpaceOpticalDepth`with the following arrays:

RedshiftSpaceOpticalDepthOfStrongestTransition (in the ion group, not the optical depth group; the weight itself, not optical-depth-weighted)

LogColumnDensity (in the ion group, not the optical depth group). Total value along the line of sight. Short spectrum contributions are printed in the log file if `verbose` is True.)

OverDensity

Temperature_K

LOSPeculiarVelocity_KMpS

Each of the arrays is of the same size as `/Wavelength_Ang`, and describes the overdensity, temperature, and peculiar velocity of each pixel, where gas particle properties have been weighted by their contribution to the optical depth.
If `output_realspacemassweighted_values` is True, the spectrum group contains a `RealSpaceMassWeighted` subgroup with the following arrays:

MetalMassFraction

OverDensity (the weight itself, not mass-weighted)

Temperature_K

LOSPeculiarVelocity_KMpS

If `output_realspacenionweighted_values` is True, the ion groups contain a `RealSpaceNionWeighted` subgroup with the following arrays:

NIon_CM3 (the weight itself, in the ion group)

OverDensity

Temperature_K

LOSPeculiarVelocity_KMpS

The real-space arrays show properties as a function of redshift along the line of sight (ignoring peculiar velocities), at the redshift resolution at the largest used wavelength in the spectrum. Depending on the redshift and wavelength ranges, these arrays can be very large. The redshifts these arrays correspond to are stored in the `Redshift_RealSpace` array in the spectrum group. (It could be moved to the root group along with the wavelengths.)

4.3 Normalised Gaussians

We use error functions instead of the M4 kernel to obtain integrals over bins. To do so we replace the M4 SPH spline with a function which is a produce of 1D Gaussians,

$$G(x,y,z) = \left(\frac{1}{(2\pi\sigma^2)^{1/2}} \, \exp(-x^2/(2\sigma^2) \right) (x \to y)((x \to z) \, . \tag{4.1}$$

We choose the relation between σ and h such that the functions have the same value at zero lag, so that

$$\begin{aligned} (2\pi\sigma^2)^{3/2} &= \frac{\pi h^3}{8} \\ 2\sigma^2 &= \frac{h^2}{4\pi^{1/3}} \\ \frac{h}{2^{1/2}\sigma} &= 2\,\pi^{1/6} \, . \end{aligned} \tag{4.2}$$

Next we want the integral of G over a cube with side $2h$ to be unity. So the 1D 'truncated' Gaussian we actually use is

$$G(x) = \frac{\mathcal{N}}{(2\pi\sigma^2)^{1/2}} \exp(-x^2/(2\sigma^2)) \tag{4.3}$$

and we determine $\mathcal{N}$ such that $\int_{-h}^{h} G(x)\, dx = 1$, which yields

$$\mathcal{N} = \frac{1}{2\,\mathrm{erf}(2\,\pi^{1/6})} \, , \tag{4.4}$$

where the error function is

$$\mathrm{erf}(x) \equiv \frac{2}{\pi^{1/2}} \int_0^x \exp(-t^2)\, dt \, . \tag{4.5}$$

A 1D column density integral for a sightline at impact parameter b is then

$$\begin{aligned} \mathcal{C} &= \left(\frac{\mathcal{N}}{(2\pi\sigma^2)^{1/2}} \right)^3 \exp(-b^2/(2\sigma^2))\, 2 \int_0^{z_+} \exp(-z^2/(2\sigma^2))\, dz \\ &= \frac{\mathcal{N}^3}{(2\pi\sigma^2\, a^2)} \times \exp(-b^2/(2\sigma^2))\, \mathrm{erf}(z_+/(2\sigma^2)^{1/2}) \\ z_+ &\equiv (h^2 - b^2)^{1/2} \, . \end{aligned} \tag{4.6}$$

Note the appearance of the expansion factor a so that the result is a physical column-density (as opposed to a co-moving one).
The mass corresponding to square pixel is then

$$\begin{aligned} \mathcal{I} &= \left(\frac{\mathcal{N}}{(2\pi\sigma^2)^{1/2}} \right)^3 \left[\int_{x_1}^{x_2} \exp(-(x-x_0)^2/(2\sigma^2))\, dx \right] \times [x \to y] \left[\int_{-\infty}^{\infty} \exp(-(z-z_0)^2/(2\sigma^2))\, dz \right] \\ &= \mathcal{N}^2 \left[\frac{1}{2} \left(\mathrm{erf}((x_2 - x_0)/(2\sigma^2)^{1/2}) - \frac{1}{2} \mathrm{erf}((x_1 - x_0)/(2\sigma^2)^{1/2}) \right] \times [x \to y] \right] \, , \end{aligned} \tag{4.7}$$

and the corresponding mean column density is $\bar{\mathcal{C}} = \mathcal{I}/(a\,dx)^2$, again multiplying by the scale factor a to obtain a physical column density.

Chapter 5

How specwizard works

Here, we give an outline of the algorithms used by specwizard to calculate the spectra. This is useful for understanding the outputs, and should hopefully be helpful for modifying or debugging the code.

5.1 Short spectra

When calculating short spectra, the SPH particle data is first interpolated to a 1-dimensional grid along the line of site. This is then used to calculated the optical depth as a function of line of sight velocity. In these calculations, positions are periodic with the box size, as in the simulations being post-processed. Similarly, line-of-sight velocities are taken to be periodic, with a period equal to the Hubble flow across the box.

`subroutine projectdata` Looping over the SPH particles, the ion content is calculated for each. The ionization balance is calculated from the density, temperature, and redshift, by interpolating the ionization tables. The total ion content then follows from the ionization fraction, mass, and element abundance. To calculate the ion density in each cell along the line of sight in position space, the chosen kernel is used, scaled with the smoothing length of the particle. Depending on the integration option, the kernel is either evaluated at the cell centre or integrated over the line of sight between cell edges (second option only available for a Gaussian kernel). Besides the ion densities themselves, the velocity along the line of sight and the temperature are needed for each cell. The ion-weighted temperatures and velocities are used for this. The number of cells in position space is equal to the number of pixels in velocity space.

`subroutine makespectra`, `subroutine computespectrum` From the ion densities, velocties, and temperatures in the cells, along with the cell sizes

and atomic data, the optical depth as a function of line of sight velocity is calculated. Looping over cell with position x along the line of sight, each cell is treated as a single absorbing cloud. Each cell's absorption is modelled as a single Gaussian line. (Damping wings are ignored, but can be added in further post-processing). The total (integrated) optical depth is set by the column density in the cell, and the width by thermal broadening. (Additional turbulent broadening should be possible, but is not recently tested.) It's centre is set at $xH(z) + v_p$, where v_p is the ion-weighted peculiar velocity in the cell. The Gaussian is either integrated over the cell or evaluated at the cell centre (`ntegrate_thermprof_exactly`).

5.2 Long spectra

To make long spectra, line-of-sight (los) files are needed in the current implementation. These are simulation outputs at smaller time intervals than the Eagle snapshots, containing all particles intersecting a set of 100 lines of sight through the box at each output time. These lines of sight are along the X or Z axis of the simulation, and the axes and positions are randomly chosen. Specwizard chooses lines of sight randomly from all files within some redshift difference $\Delta\, z$ of the end redshift of the last segment `zcurrent`, set by the `fzresol` parameter.
The lines of sight (files, LOS numbers, positions, and axes) used are stored in the specwizard output. The positions are in the same units as coordinates in the simulation; for Eagle, that's $\mathrm{cMpc\,h^{-1}}$.
The short spectra are calculated in the same way as in short spectrum mode, except that the densities are rescaled by the expansion factor at the redshift of the sightline segment (`acurrent/zcurrent`), rather than that of the line-of-sight file or snapshot. The same is true for cosmological calculations, such as local Hubble factors.
These short spectra are then interpolated onto a wavelength grid. The `output_frequency` options allows frequency outputs, but the calculations are still done in wavelength space; the output file simply contains the fixed-Δ-λ-grid spectra, but the spectrum bin array is converted to frequency units (MHz). The resolution at which the spectra are calculated is set by the pixel size chosen in velocity space.
The spectra are finally binned to that fixed-Δ-λ-grid, simply averaging the values that contributes to the same final pixel. This may cause small discrepancies between flux and optical depth spectra. For weighted average spectra, the binning respects the weighting. (The weighted quantities are multiplied by the weights before binning, then divided by the binned weights.)
Weighted line of sight properties are a function of line of sight position for the real space quantities, and wavelength for the redshift space quantities. Optical-depth-weighted averages account for contributions by the strongest (first listed) transition for each ion only.

`spline_interpolate` adds interpolated values to values already in the output array. Great for optical depths of different lines, but requires caution for averaged quantities.
Note that in the spectrum calculations, short spectra are calculated as a function of Δv, the velocity difference according to the Hubble flow and peculiar velocity with line of sight coordinate zero (assuming zero peculiar velocity there). For each pixel, this gives

$$1 + z_{\rm pix} = (1 + z_0)(1 + \Delta v_{\rm pix}/c), \tag{5.1}$$

in the limit of small Δv, with z_0 the redshift at the zero position of the line of sight. Using

$$\lim_{n\to\infty} (1 + x/n)^n = \exp(x), \tag{5.2}$$

$$1 + z_{\rm pix} = (1 + z_0)\exp(\Delta v_{\rm pix}/c) \tag{5.3}$$

for general Δv. This is used in the subroutine `insertspectra`, where the spectra are tabulated as a function of $\log$ wavelength, linear in Δv. This is also where the logs and exps in the setup of the long spectra come from.