redrock Documentation

Release 0.15.4

DESI

Nov 17, 2022

Contents

1 Contents 1

2 Indices and tables 21

Python Module Index 23

Index 25

CHAPTER 1

Contents

1.1 redrock Change Log

1.1.1 0.15.5 (unreleased)

- No changes yet.

1.1.2 0.15.4 (2022-02-28)

- Add redrock.templates.eval_model convenience routine (PR #206).

1.1.3 0.15.3 (2022-02-11)

- Propagate SURVEY and PROGRAM keywords from input files (PR #203).

1.1.4 0.15.2 (2022-01-23)

- Propagate spec group keywords from input files (PR #202).

1.1.5 0.15.1 (2022-01-20)

- add dependency keywords to redrock output (PR #200).
- set zwarn LITTLE_COVERAGE for badamp/badcol (PR #201).

1.1.6 0.15.0 (2021-07-14)

Note: Major changes to output formats; requires desispec >= 0.45.0

- Split FIBERMAP into FIBERMAP (coadded) and EXP_FIBERMAP (per-exposure) (PR #196).
- Add additional ZWARN bit masking for known bad input data (PR #196).
- Rename zbest -> redrock output, update rrdesi option names (PR #198).

1.1.7 0.14.6 (2021-07-06)

- reserve ZWARN bits 16-23 for end-user; redrock will not set these.
- Add tophap prior option (PR #194).
- Switch to github actions for testing (PR #195).

1.1.8 0.14.5 (2021-02-15)

- Use temporary files + rename to avoid partially written files with the final name in case of timeout (PR #186).

1.1.9 0.14.4 (2020-08-03)

- Re-enable ability for templates to specify their redshift range (one line update to master).

1.1.10 0.14.3 (2020-04-07)

- Allow `redrock.external.boss.read_spectra()` to receive a string as well as a list of files (PR #173).
- Support coadds that don't have EXPID in fibermap (master update).

1.1.11 0.14.2 (2019-10-17)

- Bug fix for specfiles of different sizes (PR #167).
- Fix plotting subset of input spectra (PR #168).
- Add *–no-mpi-abort* option (PR #170)

1.1.12 0.14.1 (2019-08-09)

- Minor code cleanup (PRs #162, #164).
- Add *and_mask* option for BOSS (PR #165).

1.1.13 0.14.0 (2018-12-16)

- Adds optional cosmic ray rejection during coadds (PR #156).
- No longer requires BRICKNAME (PR #157).
- Fix interactive plotspec window disappearing (PR #161).

1.1.14 0.13.2 (2018-11-07)

Version used for 18.11 software release.

- Codacy style recommendations (PR #155).
- Optional redshift prior (PR #152).

1.1.15 0.13.1 (2018-09-26)

- Fixed problem with new format of `make_templates` (PR #153).
- Update code based on codacy recommendations (PR #154).

1.1.16 0.13.0 (2018-08-31)

- Lower galaxy z_min from +0.005 to -0.005 (PR #136).
- Support for simutaneous fits of multiple e/BOSS spPlates (PR #137, #141, #147).
- Bug fix when using subset of targetids (PR #139).
- Small interface useability updates (PR #142, #143).
- Fix R normalization cut bug impacting tags 0.12.0 and 0.12.1 (PR #144).
- Mask sky lines 5577 and 9793.5 (PR #146).
- Standarize ZBEST output format for easier concatenating tables (PR #149).

1.1.17 0.12.1 (2018-07-26)

- Update DELTACHI2 column definition to match how it is used in ZWARN flag, i.e. excluding other candidates with nearby redshifts (PR #134).

1.1.18 0.12.0 (2018-07-18)

- Adds optional archetypes (PR #119).
- Include blank fibers in output with ZWARN NODATA flag (PR #123).
- Include template name in output (PR #124).
- Include template and archetype version numbers in zbest output (PR #126, #128, and #131).
- Update travis testing to astropy=2 python=3 (PR #127).
- Increase QSO redshift range to z=6 (PR #130).
- rrplot option for a subset of targetids (PR #132).

1.1.19 0.11.0 (2018-05-10)

- Catch LinAlgErrors from bad input data (PR #109).
- Add –nminima option (PR #113).
- Improve spectra reading speed (PR #114).

- hdf5 file locking workaround (PR #116).
- Fix MPI version of LyA transmission correction (PR #117).
- WD DA and DB templates (PR #118).

1.1.20 0.10.1 (2018-03-30)

- Default QSO redshift range 0.05-4.0 instead of 0.5-4.0 (PR #107).

1.1.21 0.10.0 (2018-03-29)

- Correct QSO template for LyA during zscan (PR #104).

1.1.22 0.9.0 (2018-02-23)

- ivar=0 for edge pix with integral(resolution)<0.99 (PR #94)
- Restore –ncpu option (PR #95)
- Adds wrap-redrock MPI wrapper script (PR #97)
- Robust to input NaN and Inf (PR #99)
- Adds WD templates (PR #101)

1.1.23 0.8.0 (2018-01-30)

- Major restructure of MPI and multiprocessing dataflow (PR #67, #73, #76).
- Fully support desiInstall and DESI infrastructure generally (PR #65).
- Fix import errors that were preventing RTD builds (PR #91).
- Add seed to template generation; increase number of stars used (PR #93).
- Add rrplot script to be called from ipython (PR #90).

1.1.24 0.7.0 (2017-12-20)

- no ZWARN SMALL_DELTA_CHI2 between same spectype (PR #47)
- rrdesi –templates can now be folder not just file (PR #44)
- Allow templates to optionally include redshift range (PR #41)
- API CHANGE: redrock.io.read_templates() returns dict not list (PR #41)
- set ivar = 0 where mask != 0 (PR #42)
- Add NUMEXP and NUMTILE to zbest output (PR #59)
- Propagate input fibermap into output zbest (PR #59)

1.1.25 0.6.0 (2017-11-10)

- adds rrboss to process boss spectra (PR #37)
- refactors multiprocessing parallelism to use less memory (PR #37)

1.1.26 0.5.0 (2017-09-29)

- adds optional MPI parallelism (PR #34)

1.1.27 0.4.2 (2017-08-14)

- refactored multiprocessing parallelism to use explicit shared memory (PR #31)

1.1.28 0.4.1 (2017-06-16)

- add support for new DESI spectra format

1.1.29 0.4 (2017-02-03)

- add optional truth input to plotspec
- Fix bug when first target is missing a channel of data
- external.desi.read_bricks allow glob for list of brick files
- external.desi.read_bricks read subset of targetids from bricks
- add support for stars and template subtypes
- limit galaxy redshift scan to z<1.7

1.1.30 0.3 (2017-01-23)

- added this file
- python3 updates
- refactor internal data object wrappers
- fit and store multiple minima in chi2 vs. z
- refactor parallelism
- add option to fit coadd instead of individual spectra
- add plotspec
- experimental: penalize GALAXY template fits with negative [OII] flux

1.1.31 0.2 (2016-03-05)

- tag for DESI zdc1

1.2 Priors format in Redrock

1.2.1 Implementation

The implementation was done in (PR #152) and updated in (PR #194)

1.2.2 Prior form available

- gaussian –> the amplitude is set to one. I (Edmond) noticed it is not always sufficient –> in this case use tophat prior
- lorentzien –> same remark than above
- tophat –> useful if you really trust the region where you want to search. Use for the clustering QSO catalog in DESI

1.2.3 Use prior in RR

- Build a prior file as explained above. The prior_file as to contain a prior for every targetid on which redrock will be run
- Use the flag : –priors filename_priors during the execution of rrdesi

1.2.4 Prior file

- I (Edmond) give here a minimal function to write in a correct way the prior file (see raw file –> problem of render with github...):

```
import numpy as np
import fitsio

def write_prior_for_RR(targetid, z_prior, filename_priors):
    """
        Minimal fonction to write prior file for redrock.

        targetid : must be the array of targetid list given to redrock in the rrdesi␣
↪command
        z_prior  : array of size targetid.size containing the prior value of the␣
↪redshift for the considered targetid. For instant value from QuasarNet.
        filename_priors : name of the prior file which will be given to the rrdesi␣
↪command

    """

    # need to be the same for every target
    # only function[0] will be read in the prior class !
    function = np.array(['tophat'] * z_prior.size)

    # can be different for every target (I set it constant here)
    sigma = 0.1*np.ones(z_prior.size)

    # save
    out = fitsio.FITS(filename_priors, 'rw', clobber=True)
```

(continues on next page)

(continued from previous page)

```
    data, names, extname = [targetid, function, z_prior, sigma], ['TARGETID',
↪'FUNCTION', 'Z', 'SIGMA'], 'PRIORS'
    out.write(data, names=names, extname=extname)
    out.close()

    print(f'     Write prior file for RR with {z_prior.size} objetcs: {filename_
↪priors}')
```

1.3 Full redrock API Reference

This is used to include docstrings from modules. See the autodoc documentation.

If you're loading a module here, and don't see some functions, try adding the `:imported-members:` option.

1.3.1 redrock

Redrock redshift fitter.

1.3.2 redrock.constants

Set constants used by the rest of the package.

1.3.3 redrock.external

This provides interfaces to actual data files.

1.3.4 redrock.external.boss

redrock wrapper tools for BOSS

redrock.external.boss.**read_spectra**(*spplates_name*, *targetids=None*, *use_frames=False*, *fiberid=None*, *coadd=False*, *cache_Rcsr=False*, *use_andmask=False*)

Read targets from a list of spectra files

Parameters

- **spplates_name** (*list or str*) – input spPlate files or pattern to match files.
- **targetids** (*list*) – restrict targets to this subset.
- **use_frames** (*bool*) – if True, use frames.
- **fiberid** (*int*) – Use this fiber ID.
- **coadd** (*bool*) – if True, compute and use the coadds.
- **cache_Rcsr** (*bool*) – pre-calculate and cache sparse CSR format of resolution matrix R
- **use_andmask** (*bool*) – sets ivar = 0 to pixels with and_mask != 0

Returns (targets, meta) where targets is a list of Target objects and meta is a Table of metadata (currently only BRICKNAME).

Return type tuple

redrock.external.boss.**rrboss**(*options=None*, *comm=None*)
Estimate redshifts for BOSS targets.

This loads targets serially and copies them into a DistTargets class. It then runs redshift fitting and writes the output to a catalog.

Parameters

- **options** (*list*) – optional list of commandline options to parse.
- **comm** (*mpi4py.Comm*) – MPI communicator to use.

redrock.external.boss.**write_zbest**(*outfile*, *zbest*, *template_version*, *archetype_version*)
Write zbest Table to outfile

Parameters

- **outfile** (*str*) – output file.
- **zbest** (*Table*) – the output best fit results.

1.3.5 redrock.external.desi

redrock wrapper tools for DESI

class redrock.external.desi.**DistTargetsDESI**(*spectrafiles*, *coadd=True*, *targetids=None*, *first_target=None*, *n_target=None*, *comm=None*, *cache_Rcsr=False*, *cosmics_nsig=0*)
Distributed targets for DESI.

DESI spectral data is grouped by sky location, but is just a random collection of spectra for all targets. Read this into memory while grouping by target ID, preserving order in which each target first appears.

We pass through the spectra files once to compute all the book-keeping associated with regrouping the spectra by target. Then we pass through again and actually read and distribute the data.

Parameters

- **spectrafiles** (*str or list*) – a list of input files or pattern match of files.
- **coadd** (*bool*) – if False, do not compute the coadds.
- **targetids** (*list*) – (optional) restrict the global set of target IDs to this list.
- **first_target** (*int*) – (optional) integer offset of the first target to consider in each file. Useful for debugging / testing.
- **n_target** (*int*) – (optional) number of targets to consider in each file. Useful for debugging / testing.
- **comm** (*mpi4py.MPI.Comm*) – (optional) the MPI communicator.
- **cache_Rcsr** – pre-calculate and cache sparse CSR format of resolution matrix R
- **cosmics_nsig** (*float*) – cosmic rejection threshold used in coaddition

redrock.external.desi.**rrdesi**(*options=None*, *comm=None*)
Estimate redshifts for DESI targets.

This loads distributed DESI targets from one or more spectra grouping files and computes the redshifts. The outputs are written to a redrock scan file and a DESI redshift catalog.

Parameters

- **options** (*list*) – optional list of commandline options to parse.
- **comm** (*mpi4py.Comm*) – MPI communicator to use.

redrock.external.desi.**write_zbest**(*outfile*, *zbest*, *fibermap*, *exp_fibermap*, *tsnr2*, *template_version*, *archetype_version*, *spec_header=None*)

Write zbest and fibermap Tables to outfile

Parameters

- **outfile** (*str*) – output path.
- **zbest** (*Table*) – best fit table.
- **fibermap** (*Table*) – the coadded fibermap from the original inputs.
- **tsnr2** (*Table*) – table of input coadded TSNR2 values
- **exp_fibermap** (*Table*) – the per-exposure fibermap from the orig inputs.
- **template_version** (*str*) – template version used
- **archetype_version** (*str*) – archetype version used

Options: spec_header (dict-like): header of HDU 0 of input spectra

Modifies input tables.meta['EXTNAME']

1.3.6 redrock.fitz

Functions for fitting minima of chi^2 results.

redrock.fitz.**find_minima**(*x*)

Return indices of local minima of x, including edges.

The indices are sorted small to large.

Note: this is somewhat conservative in the case of repeated values: find_minima([1,1,1,2,2,2]) -> [0,1,2,4,5]

Parameters **x** (*array-like*) – The data array.

Returns The indices.

Return type (array)

redrock.fitz.**fitz**(*zchi2*, *redshifts*, *spectra*, *template*, *nminima=3*, *archetype=None*)

Refines redshift measurement around up to nminima minima.

Parameters

- **zchi2** (*array*) – chi^2 values for each redshift.
- **redshifts** (*array*) – the redshift values.
- **spectra** (*list*) – list of Spectrum objects at different wavelengths grids.
- **template** (*Template*) – the template for this fit.
- **nminima** (*int*) – the number of minima to consider.

Returns the fit parameters for the minima.

Return type Table

redrock.fitz.**get_dv**(*z*, *zref*)
Returns velocity difference in km/s for two redshifts

Parameters

- **z** (*float*) – redshift for comparison.
- **zref** (*float*) – reference redshift.

Returns the velocity difference.

Return type (float)

redrock.fitz.**minfit**(*x*, *y*)
Fits y = y0 + ((x-x0)/xerr)**2

See redrock.zwarning.ZWarningMask.BAD_MINFIT for zwarn failure flags

Parameters

- **x** (*array*) – x values.
- **y** (*array*) – y values.

Returns (x0, xerr, y0, zwarn) where zwarn=0 is good fit.

Return type (tuple)

1.3.7 redrock.plotspec

Visualization tools for plotting spectra.

1.3.8 redrock.rebin

Tools for binning data.

redrock.rebin.**_trapz_rebin**(*x*, *y*, *edges*, *results*)
Numba-friendly version of trapezoidal rebinning

See redrock.rebin.trapz_rebin() for input descriptions. *results* is pre-allocated array of length len(edges)-1 to keep results

redrock.rebin.**centers2edges**(*centers*)
Convert bin centers to bin edges, guessing at what you probably meant

Parameters **centers** (*array*) – bin centers,

Returns bin edges, lenth = len(centers) + 1

Return type array

redrock.rebin.**rebin_template**(*template*, *z*, *dwave*)
Rebin a template to a set of wavelengths.

Given a template and a single redshift, rebin the template to a set of wavelength arrays.

Parameters

- **template** (Template) – the template object
- **z** (*float*) – the redshift

- **dwave** (*dict*) – the keys are the "wavehash" and the values are a 1D array containing the wavelength grid.

Returns

The rebinned template for every basis function and wavelength grid in dwave.

Return type dict

redrock.rebin.**trapz_rebin**(*x*, *y*, *xnew=None*, *edges=None*)

Rebin y(x) flux density using trapezoidal integration between bin edges

Notes

y is interpreted as a density, as is the output, e.g.

```
>>> x = np.arange(10)
>>> y = np.ones(10)
>>> trapz_rebin(x, y, edges=[0,2,4,6,8])  #- density still 1, not 2
array([ 1.,  1.,  1.,  1.])
```

Parameters

- **x** (*array*) – input x values.
- **y** (*array*) – input y values.
- **edges** (*array*) – (optional) new bin edges.

Returns integrated results with len(results) = len(edges)-1

Return type array

Raises ValueError – if edges are outside the range of x or if len(x) != len(y)

Functions for reading and writing full redrock results to HDF5.

redrock.results.**read_zscan**(*filename*)

Read redrock.zfind results from a file.

Returns

(zbest, results) where zbest is a Table with keys TARGETID, Z, ZERR, ZWARN and results is a nested dictionary results[targetid][templatetype] with keys:

- z: array of redshifts scanned
- zchi2: array of chi2 fit at each z
- penalty: array of chi2 penalties for unphysical fits at each z
- **zbest: best fit redshift (finer resolution fit around zchi2** min)
- minchi2: chi2 at zbest
- zerr: uncertainty on zbest
- zwarn: 0=good, non-0 is a warning flag

Return type tuple

redrock.results.**read_zscan_redrock**(*filename*)

Read redrock.zfind results from a file to be reused by redrock itself.

Returns

dictionary of results for each local target ID. dic.keys() are TARGETID dic[tg].keys() are TEMPLATE dic[tg][ft].keys() are ['penalty', 'zcoeff', 'zchi2', 'redshifts']

Return type dict

redrock.results.**write_zscan**(*filename*, *zscan*, *zfit*, *clobber=False*)
Writes redrock.zfind results to a file.

The nested dictionary structure of results is mapped into a nested group structure of the HDF5 file:

/targetids[nt] /zscan/{spectype}/redshifts[nz] /zscan/{spectype}/zchi2[nt, nz] /zscan/{spectype}/penalty[nt, nz] /zscan/{spectype}/zcoeff[nt, nz, nc] or zcoeff[nt, nc, nz] ? /zfit/{targetid}/zfit table...

Parameters

- **filename** (*str*) – the output file path.
- **zscan** (*dict*) – the full set of fit results.
- **zfit** (*Table*) – the best fit redshift results.
- **clobber** (*bool*) – if True, delete the file if it exists.

Classes and functions for targets and their spectra.

class redrock.targets.**DistTargets**(*targetids*, *comm=None*)
Base class for distributed targets.

Target objects are distributed across the processes in an MPI communicator, but the details of how this data is loaded from disk is specific to a given project. Each project should inherit from this base class and create an appropriate class for the data files being used.

This class defines some general methods and the API that should be followed by these derived classes.

Parameters

- **targetids** (*list*) – the global set of target IDs.
- **comm** (*mpi4py.MPI.Comm*) – (optional) the MPI communicator.

local()
Return the local list of Target objects.

local_target_ids()
Return the local list of target IDs.

wavegrids()
Return the global dictionary of wavelength grids for each wave hash.

class redrock.targets.**DistTargetsCopy**(*targets*, *comm=None*, *root=0*)
Distributed targets built from a copy.

This class is a simple wrapper that distributes targets located on one process to the processes in a communicator.

Parameters

- **targets** (*list*) – list of Target objects on one process.
- **comm** (*mpi4py.MPI.Comm*) – (optional) the MPI communicator.
- **root** (*int*) – the process which has the input targets locally.

class redrock.targets.**Spectrum**(*wave*, *flux*, *ivar*, *R*, *Rcsr=None*)
Simple container class for an individual spectrum.

Parameters

- **wave** (*array*) – the wavelength grid.

- **flux** (*array*) – the flux values.
- **ivar** (*array*) – the inverse variance.
- **R** (*scipy.sparse.dia_matrix*) – the resolution matrix in band diagonal format.
- **Rcsr** (*scipy.sparse.csr_matrix*) – the resolution matrix in CSR format.

sharedmem_pack()
Pack spectral data into multiprocessing shared memory.

sharedmem_unpack()
Unpack spectral data from multiprocessing shared memory.

class redrock.targets.**Target**(*targetid*, *spectra*, *coadd=False*, *cosmics_nsig=0.0*, *meta=None*)
A single target.

This represents the data for a single target, including a unique identifier and the individual spectra observed for this object (or a coadd).

Parameters

- **targetid** (*int or str*) – unique targetid
- **spectra** (*list*) – list of Spectrum objects
- **coadd** (*bool*) – compute and store the coadd at construction time. The coadd can always be recomputed with the compute_coadd() method.
- **cosmics_nsig** (*float*) – cosmic rejection threshold in compute_coadd
- **meta** (*dict*) – optional metadata dictionary for this Target.

compute_coadd(*cache_Rcsr=False*, *cosmics_nsig=0.0*)
Compute the coadd from the current spectra list.

Parameters

- **cache_Rcsr** – pre-calculate and cache sparse CSR format of resolution matrix R
- **cosmics_nsig** (*float*) – number of sigma for cosmic rejection.

This method REPLACES the list of individual spectra with coadds.

sharedmem_pack()
Pack all spectra into multiprocessing shared memory.

sharedmem_unpack()
Unpack all spectra from multiprocessing shared memory.

redrock.targets.**distribute_targets**(*targets*, *nproc*)
Distribute a list of targets among processes.

Given a list of Target objects, compute the load balanced distribution of those targets among a set of processes.

This function is used when one already has a list of Target objects that need to be distributed. This happens, for example, when creating a DistTargetsCopy object from pre-existing Targets, or when using multiprocessing to do operations on the MPI-local list of targets.

Parameters

- **targets** (*list*) – list of Target objects.
- **nproc** (*int*) – number of processes.

Returns

A list (one element for each process) with each element being a list of the target IDs assigned to that process.

Return type list

Classes and functions for templates.

class redrock.templates.**DistTemplate**(*template*, *dwave*, *mp_procs=1*, *comm=None*)
Distributed template data interpolated to all redshifts.

For a given template, the redshifts are distributed among the processes in the communicator. Then each process will rebin the template to those redshifts for the wavelength grids specified by dwave.

Parameters

- **template** (Template) – the template to distribute
- **dwave** (*dict*) – the keys are the "wavehash" and the values are a 1D array containing the wavelength grid.
- **mp_procs** (*int*) – if not using MPI, restrict the number of multiprocesses to this.
- **comm** (*mpi4py.MPI.Comm*) – (optional) the MPI communicator.

cycle()
Pass our piece of data to the next process.

If we have returned to our original data, then return True, otherwise return False.

Parameters **Nothing** –

Returns (bool): Whether we have finished (True) else False.

class redrock.templates.**DistTemplatePiece**(*index*, *redshifts*, *data*)
One piece of the distributed template data.

This is a simple container for storing interpolated templates for a set of redshift values. It is used for communicating the interpolated templates between processes.

In the MPI case, each process will store at most two of these simultaneously. This is the data that is computed on a single process and passed between processes.

Parameters

- **index** (*int*) – the chunk index of this piece- this corresponds to the process rank that originally computed this piece.
- **redshifts** (*array*) – the redshift range contained in this piece.
- **data** (*list*) – a list of dictionaries, one for each redshift, and each containing the 2D interpolated template values for all "wavehash" keys.

class redrock.templates.**Template**(*filename=None*, *spectype=None*, *redshifts=None*, *wave=None*, *flux=None*, *subtype=None*)
A spectral Template PCA object.

The template data is read from a redrock-format template file. Alternatively, the data can be specified in the constructor.

Parameters **filename** (*str*) – the path to the template file, either absolute or relative to the RR_TEMPLATE_DIR environment variable.

eval(*coeff*, *wave*, *z*)
Return template for given coefficients, wavelengths, and redshift

Parameters

- **coeff** – array of coefficients length self.nbasis
- **wave** – wavelengths at which to evaluate template flux
- **z** – redshift at which to evaluate template flux

Returns template flux array

Notes

A single factor of (1+z)^-1 is applied to the resampled flux to conserve integrated flux after redshifting.

full_type
subtype string.

Type Return formatted type

redrock.templates.**_mp_rebin_template**(*template*, *dwave*, *zlist*, *qout*)
Function for multiprocessing version of rebinning.

redrock.templates.**eval_model**(*data*, *wave*, *R=None*, *templates=None*)
Evaluate model spectra.

Given a bunch of fits with coefficients COEFF, redshifts Z, and types SPECTYPE, SUBTYPE in data, evaluate the redrock model fits at the wavelengths wave using resolution matrix R.

The wavelength and resolution matrices may be dictionaries including for multiple cameras.

Parameters

- **data** (*table-like, [nspec]*) – table with information on each model to evaluate. Must contain at least Z, COEFF, SPECTYPE, and SUBTYPE fields.
- **wave** (*array [nwave] or dictionary thereof*) – array of wavelengths in angstrom at which to evaluate the models.
- **R** (*list of [nwave, nwave] arrays of floats or dictionary thereof*) – resolution matrices for evaluating spectra.
- **templates** (*dictionary of Template*) – dictionary with (SPECTYPE, SUBTYPE) giving the template corresponding to each type.

Returns model fluxes, array [nspec, nwave]. If wave and R are dict, then a dictionary of model fluxes, one for each camera.

redrock.templates.**find_templates**(*template_dir=None*)
Return list of redrock-*.fits template files

Search directories in this order, returning results from first one found:

- template_dir
- $RR_TEMPLATE_DIR
- <redrock_code>/templates/

Parameters **template_dir** (*str*) – optional directory containing the templates.

Returns a list of template files.

Return type list

redrock.templates.**load_dist_templates**(*dwave*, *templates=None*, *comm=None*, *mp_procs=1*)
Read and distribute templates from disk.

This reads one or more template files from disk and distributes them among an MPI communicator. Each process will locally store interpolated data for a redshift slice of each template. For a single redshift, the template is interpolated to the wavelength grids specified by "dwave".

As an example, imagine 3 templates with independent redshift ranges. Also imagine that the communicator has 2 processes. This function would return a list of 3 DistTemplate objects. Within each of those objects, the 2 processes store the interpolated data for a subset of the redshift range:

DistTemplate #1: zmin1 <— p0 —> | <— p1 —> zmax1 DistTemplate #2: zmin2 <– p0 –> | <– p1 –> zmax2 DistTemplate #3: zmin3 <— p0 —> | <— p1 —> zmax3

Parameters

- **dwave** (*dict*) – the dictionary of wavelength grids. Keys are the "wavehash" and values are an array of wavelengths.
- **templates** (*str or None*) – if None, find all templates from the redrock template directory. If a path to a file is specified, load that single template. If a path to a directory is given, load all templates in that directory.
- **comm** (*mpi4py.MPI.Comm*) – (optional) the MPI communicator.
- **mp_procs** (*int*) – if not using MPI, restrict the number of multiprocesses to this.

Returns a list of DistTemplate objects.

Return type list

Redrock utility functions.

redrock.utils.**distribute_work**(*nproc*, *ids*, *weights=None*)
Helper function to distribute work among processes.

This takes a list of unique IDs associated with each work unit, and a dictionary of weights for each ID. It returns a list of lists which contain the IDs assigned to each process.

Parameters

- **nproc** (*int*) – the number of processes.
- **ids** (*list*) – list of IDs
- **weights** (*dict*) – dictionary of weights for each ID. If None, use equal weighting.

Returns

A list (one element for each process) with each element being a list of the IDs assigned to that process.

Return type list

redrock.utils.**elapsed**(*timer*, *prefix*, *comm=None*)
Get and print the elapsed time.

If timer is None, compute the start time and return. Otherwise, find the elapsed time and print a message before returning the new start time.

Parameters

- **timer** (*float*) – time in seconds for some arbitrary epoch. If "None", get the current time and return.
- **prefix** (*str*) – string to print before the elapsed time.

- **comm** (*mpi4py.MPI.Comm*) – optional communicator.

Returns the new start time in seconds.

Return type float

redrock.utils.**encode_column**(*c*)
Returns a bytes column encoded into a string column.

Parameters **c** (*Table column*) – a column of a Table.

Returns an array of strings.

Return type array

redrock.utils.**get_mp**(*requested*)
Returns a reasonable number of multiprocessing processes.

This checks whether the requested value makes sense, and also whether we are running on a NERSC login node (and hence would get in trouble trying to use all cores).

Parameters **requested** (*int*) – the requested number of processes.

Returns the number of processes to use.

Return type int

redrock.utils.**getch**()
Return a single character from stdin.

redrock.utils.**mp_array**(*original*)
Allocate a raw shared memory buffer and wrap it in an ndarray.

This allocates a multiprocessing.RawArray and wraps the buffer with an ndarray.

Parameters

- **typecode** (*str*) – the type code of the array.
- **size_or_init** – passed to the RawArray constructor.

Returns; ndarray: the wrapped data.

redrock.utils.**native_endian**(*data*)
Convert numpy array data to native endianness if needed.

Returns new array if endianness is swapped, otherwise returns input data

By default, FITS data from astropy.io.fits.getdata() are not Intel native endianness and scipy 0.14 sparse matrices have a bug with non-native endian data.

Parameters **data** (*array*) – input array

Returns

original array if input in native endianness, otherwise a copy with the bytes swapped.

Return type array

redrock.utils.**nersc_login_node**()
Returns True if we are on a NERSC login node, else False.

redrock.utils.**transmission_Lyman**(*zObj*, *lObs*)
Calculate the transmitted flux fraction from the Lyman series This returns the transmitted flux fraction: 1 -> everything is transmitted (medium is transparent) 0 -> nothing is transmitted (medium is opaque) :param zObj: Redshift of object :type zObj: float :param lObs: wavelength grid :type lObs: array of float

Returns transmitted flux fraction

Return type array of float

1.3.9 redrock.zfind

Redshift finding algorithms.

redrock.zfind.**_mp_fitz**(*chi2*, *target_data*, *t*, *nminima*, *qout*, *archetype*)
Wrapper for multiprocessing version of fitz.

redrock.zfind.**calc_deltachi2**(*chi2*, *z*, *dvlimit=None*)
Calculate chi2 differences, excluding candidates with close z

Parameters

- **chi2** – array of chi2 values
- **z** – array of redshifts

Options: dvlimit: exclude candidates that are closer than dvlimit [km/s]

Note: The final target always has deltachi2=0.0 because we don't know what the next chi2 would have been. This can also occur for the last N targets if all N of them are within dvlimit of each other.

redrock.zfind.**zfind**(*targets*, *templates*, *mp_procs=1*, *nminima=3*, *archetypes=None*, *priors=None*, *chi2_scan=None*)
Compute all redshift fits for the local set of targets and collect.

Given targets and templates distributed across a set of MPI processes, compute the redshift fits for all redshifts and our local set of targets. Each process computes the fits for a slice of redshift range and then cycles through redshift slices by passing the interpolated templates along to the next process in order.

Note: If using MPI, only the rank 0 process will return results- all other processes with return a tuple of (None, None).

Parameters

- **targets** (DistTargets) – distributed targets.
- **templates** (*list*) – list of DistTemplate objects.
- **mp_procs** (*int, optional*) – if not using MPI, this is the number of multiprocessing processes to use.
- **nminima** (*int, optional*) – number of chi^2 minima to consider. Passed to fitz().
- **archetypes** (*str, optional*) – file or directory containing archetypes to use for final fitz choice of best chi2 vs. z minimum.
- **priors** (*str, optional*) – file containing redshift priors
- **chi2_scan** (*str, optional*) – file containing already computed chi2 scan

Returns

(allresults, allzfit), where "allresults" is a dictionary of the full chi^2 fit information, suitable for writing to a redrock scan file. "allzfit" is an astropy Table of only the best fit parameters for a limited set of minima.

Return type tuple

1.3.10 redrock.zscan

Algorithms for scanning redshifts.

redrock.zscan.**_mp_calc_zchi2**(*indx*, *target_ids*, *target_data*, *t*, *qout*, *qprog*)
Wrapper for multiprocessing version of calc_zchi2.

redrock.zscan.**_zchi2_one**(*Tb*, *weights*, *flux*, *wflux*, *zcoeff*)
Calculate a single chi2.

For one redshift and a set of spectral data, compute the chi2 for template data that is already on the correct grid.

redrock.zscan.**calc_zchi2**(*target_ids*, *target_data*, *dtemplate*, *progress=None*)
Calculate chi2 vs. redshift for a given PCA template.

Parameters

- **target_ids** (*list*) – targets IDs.
- **target_data** (*list*) – list of Target objects.
- **dtemplate** (DistTemplate) – distributed template data
- **progress** (*multiprocessing.Queue*) – optional queue for tracking progress, only used if MPI is disabled.

Returns

(zchi2, zcoeff, zchi2penalty) with:

- **zchi2[ntargets, nz]: array with one element per target per** redshift
- **zcoeff[ntargets, nz, ncoeff]: array of best fit template** coefficients for each target at each redshift
- **zchi2penalty[ntargets, nz]: array of penalty priors per target** and redshift, e.g. to penalize unphysical fits

Return type tuple

redrock.zscan.**calc_zchi2_one**(*spectra*, *weights*, *flux*, *wflux*, *tdata*)
Calculate a single chi2.

For one redshift and a set of spectra, compute the chi2 for template data that is already on the correct grid.

Parameters

- **spectra** (*list*) – list of Spectrum objects.
- **weights** (*array*) – concatenated spectral weights (ivar).
- **flux** (*array*) – concatenated flux values.
- **wflux** (*array*) – concatenated weighted flux values.
- **tdata** (*dict*) – dictionary of interpolated template values for each wavehash.

Returns chi^2 and coefficients.

Return type tuple

redrock.zscan.**calc_zchi2_targets**(*targets*, *templates*, *mp_procs=1*)
Compute all chi2 fits for the local set of targets and collect.

Given targets and templates distributed across a set of MPI processes, compute the coarse-binned chi^2 fit for all redshifts and our local set of targets. Each process computes the fits for a slice of redshift range and then cycles through redshift slices by passing the interpolated templates along to the next process in order.

Parameters

- **targets** (DistTargets) – distributed targets.
- **templates** (*list*) – list of DistTemplate objects.
- **mp_procs** (*int*) – if not using MPI, this is the number of multiprocessing processes to use.

Returns dictionary of results for each local target ID.

Return type dict

redrock.zscan.**spectral_data**(*spectra*)

Compute concatenated spectral data products.

This helper function builds full length array quantities needed for the chi2 fit.

Parameters **spectra** (*list*) – list of Spectrum objects.

Returns

(weights, flux, wflux) concatenated values used for single redshift chi^2 fits.

Return type tuple

1.3.11 redrock.zwarning

Mask bit definitions for zwarning.

WARNING on the warnings: not all of these are implemented yet.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

redrock, 7
redrock.constants, 7
redrock.external, 7
redrock.external.boss, 7
redrock.external.desi, 8
redrock.fitz, 9
redrock.plotspec, 10
redrock.rebin, 10
redrock.results, 11
redrock.targets, 12
redrock.templates, 14
redrock.utils, 16
redrock.zfind, 18
redrock.zscan, 19
redrock.zwarning, 20

Index

Symbols

_mp_calc_zchi2() (*in module redrock.zscan*), 19
_mp_fitz() (*in module redrock.zfind*), 18
_mp_rebin_template() (*in module redrock.templates*), 15
_trapz_rebin() (*in module redrock.rebin*), 10
_zchi2_one() (*in module redrock.zscan*), 19

C

calc_deltachi2() (*in module redrock.zfind*), 18
calc_zchi2() (*in module redrock.zscan*), 19
calc_zchi2_one() (*in module redrock.zscan*), 19
calc_zchi2_targets() (*in module redrock.zscan*), 19
centers2edges() (*in module redrock.rebin*), 10
compute_coadd() (*redrock.targets.Target method*), 13
cycle() (*redrock.templates.DistTemplate method*), 14

D

distribute_targets() (*in module redrock.targets*), 13
distribute_work() (*in module redrock.utils*), 16
DistTargets (*class in redrock.targets*), 12
DistTargetsCopy (*class in redrock.targets*), 12
DistTargetsDESI (*class in redrock.external.desi*), 8
DistTemplate (*class in redrock.templates*), 14
DistTemplatePiece (*class in redrock.templates*), 14

E

elapsed() (*in module redrock.utils*), 16
encode_column() (*in module redrock.utils*), 17
eval() (*redrock.templates.Template method*), 14
eval_model() (*in module redrock.templates*), 15

F

find_minima() (*in module redrock.fitz*), 9
find_templates() (*in module redrock.templates*), 15
fitz() (*in module redrock.fitz*), 9
full_type (*redrock.templates.Template attribute*), 15

G

get_dv() (*in module redrock.fitz*), 10
get_mp() (*in module redrock.utils*), 17
getch() (*in module redrock.utils*), 17

L

load_dist_templates() (*in module redrock.templates*), 15
local() (*redrock.targets.DistTargets method*), 12
local_target_ids() (*redrock.targets.DistTargets method*), 12

M

minfit() (*in module redrock.fitz*), 10
mp_array() (*in module redrock.utils*), 17

N

native_endian() (*in module redrock.utils*), 17
nersc_login_node() (*in module redrock.utils*), 17

R

read_spectra() (*in module redrock.external.boss*), 7
read_zscan() (*in module redrock.results*), 11
read_zscan_redrock() (*in module redrock.results*), 11
rebin_template() (*in module redrock.rebin*), 10
redrock (*module*), 7
redrock.constants (*module*), 7
redrock.external (*module*), 7
redrock.external.boss (*module*), 7
redrock.external.desi (*module*), 8
redrock.fitz (*module*), 9
redrock.plotspec (*module*), 10
redrock.rebin (*module*), 10
redrock.results (*module*), 11
redrock.targets (*module*), 12

redrock.templates (*module*), 14
redrock.utils (*module*), 16
redrock.zfind (*module*), 18
redrock.zscan (*module*), 19
redrock.zwarning (*module*), 20
rrboss() (*in module redrock.external.boss*), 8
rrdesi() (*in module redrock.external.desi*), 8

S

sharedmem_pack() (*redrock.targets.Spectrum method*), 13
sharedmem_pack() (*redrock.targets.Target method*), 13
sharedmem_unpack() (*redrock.targets.Spectrum method*), 13
sharedmem_unpack() (*redrock.targets.Target method*), 13
spectral_data() (*in module redrock.zscan*), 20
Spectrum (*class in redrock.targets*), 12

T

Target (*class in redrock.targets*), 13
Template (*class in redrock.templates*), 14
transmission_Lyman() (*in module redrock.utils*), 17
trapz_rebin() (*in module redrock.rebin*), 11

W

wavegrids() (*redrock.targets.DistTargets method*), 12
write_zbest() (*in module redrock.external.boss*), 8
write_zbest() (*in module redrock.external.desi*), 9
write_zscan() (*in module redrock.results*), 12

Z

zfind() (*in module redrock.zfind*), 18