DBSP_DRP

Release 1.0.0.post4.dev5+g75b1b25

Milan Sharma Mandigo-Stoba

Mar 22, 2023

GETTING STARTED:

Version: 1.0.0.post4.dev5

DBSP_DRP is a Data Reduction Pipeline for Palomar's workhorse spectrograph DBSP. It is built on top of PypeIt. DBSP_DRP automates the reduction, fluxing, telluric correction, and combining of the red and blue sides of one night's data. It adds several GUIs to allow for easier control of your reduction:

- select which data to reduce, and verify the correctness of your FITS headers in an editable table GUI
- manually place traces for a sort of manually "forced" spectroscopy with the `-m` option
- after manually placing traces, manually select sky regions and tweak the FWHM of your manual traces

DBSP_DRP also provides a quicklook script for making real-time decisions during an observing run, and can open a GUI displaying a minimally reduced exposure in under 15 seconds.

CHAPTER

ONE

STATEMENT OF NEED

Palomar Observatory, located near San Diego, CA, is a multinational observatory with a broad user base. Users come from large and small institutions, and their observing experience ranges from novice to expert. One responsibility for serving such a diverse user base is to provide software data reduction pipelines for the most frequently used instruments, such as the Palomar Double Spectrograph (DBSP). Although DBSP was commissioned in 1982, it remains the workhorse instrument of the 200" Hale Telescope. It is used on 42% of the nights in a year, comprising nearly all of the valuable "dark" (moonless) time. In previous years, standard astronomical practice left the data reduction up to the user. However, attitudes in instrument building have shifted since DBSP was built. The pipeline is now considered an indispensable component of the astronomical instrument. In fact, the difference between a good pipeline and a great pipeline means the difference between counting some of the photons vs. counting all of the photons.

Spectroscopy is a severe bottleneck in time-domain astronomy; currently less than 10% of discoveries are spectroscopically classified. Without a pipeline, data reduction is a difficult process and the standard method without a pipeline is to use IRAF, a 35 year old program on which development and maintenance was discontinued in 2013 and whose use is discouraged by many in the field. Needless to say, data reduction sans pipeline is extremely time-consuming. There is a clear need for a modern and stable automated data reduction pipeline for DBSP.

During observing runs, one would like to be able to quickly inspect data as it is taken, in order to ensure that it is of sufficient quality to do the desired science with. For objects whose brightness may have changed between a previous observation and the observing run, the observer may have uncertainties regarding how long of an exposure is needed to produce quality data. For very faint objects or objects in crowded fields, the observer may not even be sure that the telescope is pointed at the right object! A quicklook functionality, that can do a rudimentary reduction to correct for instrumental signatures and subtract light from the sky, revealing the spectra of the objects observed, can answer questions of exposure time and whether the object observed is the right one.

1.1 Community Guidlines

1.1.1 Contributing

Contributions come in many different kinds, spanning bug fixes and new features to suggestions for improved documentation or ways to make your DBSP_DRP workflow easier for you and more efficient. All types of contributions from the community are greatly welcomed and much appreciated!

If you want to fix a bug or contribute a new feature, please create a fork of DBSP_DRP on GitHub, make changes to your fork, and then create a Pull Request when your changes are ready to be reviewed for inclusion in DBSP_DRP. Maintainers will review your proposed changes and give you feedback. Once all parties are satisfied with the proposed changes, your code will be merged. Please create pull requests to the develop branch of DBSP_DRP, as this is where active development occurs.

If you have suggestions for how to improve DBSP_DRP, please open an Issue on GitHub describing how you want DBSP_DRP to be improved. Maintainers and developers will comment on the Issue, discussing your suggestions in

a collaborative manner, developing and maturing it with you until it is ready to be implemented (and someone has volunteered to do so).

1.1.2 Reporting Issues

If you run into an error while running DBSP_DRP, identify a bug, documentation deficiency, or potential enhancement, please file an Issue on GitHub. For bugs or errors, please try to be as descriptive as possible when writing what went wrong – it's hard to provide too much detail! At a bare minimum, please include the versions of Python, DBSP_DRP, PypeIt, NumPy, and Astropy that were installed when you encountered the error. If you can create a "minimum working example" for the error, the smallest set of input files and command line invocation or Python code that still manifests this error, and share this with your Issue, it will greatly assist anybody who tries to fix your error.

1.1.3 Getting Support

If you are having trouble running DBSP_DRP, you can open a Discussion on GitHub to ask the community for help. There are no silly questions – though we try to make DBSP_DRP work as well as possible, computers are finnicky creatures and entropy is always working against us.

1.2 Installing DBSP_DRP

Conda is the recommended method for installing `DBSP_DRP`.

1.2.1 Using conda

Install `DBSP_DRP` from conda by running

```
$ conda install -c conda-forge dbsp_drp
```

1.2.2 Using pip

First download the provided environment.yml file

Now use the environment.yml file to create a conda environment with the required dependencies.

```
$ cd /path/to/Downloads
$ conda env create -f environment.yml
$ conda activate dbsp_drp
```

Now use `pip` to install DBSP_DRP

```
$ pip install dbsp-drp
```

1.2.3 From Source

```
$ git clone https://github.com/finagle29/DBSP_DRP.git
$ cd DBSP_DRP
$ conda env create -f environment.yml
$ conda activate dbsp_drp
$ pip install -e .
```

This performs an editable install, which allows you to make modifications to the code and immediately see their effects. Importantly, this can be used in combination with `git` branches to test features in development.

1.3 Post-Install

The telluric correction code provided by PypeIt relies on a large (5 GB) atmospheric model file (TellFit_Lick_3100_11100_R10000.fits), which can be downloaded here and must be installed into the `pypeit/data/telluric/atm_grids` directory of your PypeIt installation.

To determine the location of your PypeIt installation, open the Python interpreter and run

```
>>> import pypeit
>>> import os
>>> print(os.path.dirname(pypeit.__file__))
/Users/me/anaconda3/envs/dbsp_drp/lib/python3.7/site-packages/pypeit
```

An easier alternative is to download and run this script, which will perform the download and install it into the current PypeIt installation.

```
$ cd /path/to/Downloads
$ chmod +x download_tellfile
$ ./download_tellfile
```

If you have multiple PypeIt installations on the same machine, you can create a hard link from the one PypeIt installation to the others so you can reuse the atmospheric model file.

```
$ ln /path/to/stable/pypeit/data/telluric/atm_grids/TellFit_Lick_3100_11100_R10000.fits /
↪path/to/other/pypeit/data/telluric/atm_grids/TellFit_Lick_3100_11100_R10000.fits
```

Make sure to use the same filename in both PypeIt installations. If you're not sure where your PypeIt installations are, run the previous Python snippet in each `conda` or `venv` environment you want to use `DBSP_DRP` in.

1.3.1 Testing your installation

Make sure your PypeIt installation was successful

```
$ run_pypeit -h
```

The expected output of this command is a usage/help message for PypeIt, which confirms that PypeIt is installed correctly.

Run some built-in tests for `DBSP_DRP`, including verification that the quicklook script works

```
$ pytest --pyargs dbsp_drp
```

Warning: the DBSP_DRP built-in tests take 5-15 minutes to run, depending on the speed of your computer, and will raise a large number of warnings that can be safely ignored.

Optionally run some built-in tests for PypeIt (note that these will take upwards of 30 minutes to run and similarly raise a large numbr of ignorable warnings).

```
$ pytest --pyargs pypeit
```

1.4 Using DBSP_DRP

After *Installing DBSP_DRP*, you are ready to reduce some DBSP data!

```
$ dbsp_reduce --help
usage: dbsp_reduce [-h] [-i] -r ROOT -d OUTPUT_PATH [-a {red,blue}] [-m]
                   [--debug] [-j N] [-p PARAMETER_FILE] [-t] [-c]
                   [--splicing-interpolate-gaps]

Automatic Data Reduction Pipeline for P200 DBSP

optional arguments:
    -h, --help            show this help message and exit
    -i, --no-interactive  Interactive file-checking?
    -r ROOT, --root ROOT  File path+root, e.g. /data/DBSP_20200127
    -d OUTPUT_PATH, --output_path OUTPUT_PATH
                            Path to top-level output directory.  Default is the current␣
↪working directory.
    -a {red,blue}, --arm {red,blue}
                            [red, blue] to only reduce one arm (null splicing)
    -m, --manual-extraction
                            manual extraction
    --debug               debug
    -j N, --jobs N        Number of processes to use
    -p PARAMETER_FILE, --parameter-file PARAMETER_FILE
                            Path to parameter file. The parameter file should be␣
↪formatted as follows:

                            [blue]
                            ** PypeIt parameters for the blue side goes here **
                            [red]
                            ** PypeIt parameters for the red side goes here **
                            EOF

                            The [red/blue] parameter blocks are optional, and their␣
↪order does not matter.
    -t, --skip-telluric   Skip telluric correction
    -c, --null-coadd      Don't coadd consecutive exposures of the same target.
                            By default consective exposures will be coadded.
    --splicing-interpolate-gaps
                            Use this option to linearly interpolate across large gaps
                            in the spectrum during splicing. The default behavior is to
                            only use data from one detector in these gaps, which results
                            in a slightly noisier spliced spectrum.
```

The basic usage of DBSP_DRP is as follows:

```
$ dbsp_reduce -r /path/to/data/DBSP_YYYYMMDD/ -d /output/path/DBSP_YYYYMMDD_redux
```

When reducing a night of data on a machine with multiple CPU cores, it is highly recommended to add the `-j N` flag to use N jobs for the telluric correction.

Note: the default setting is to open a GUI that allows you to verify that the header data is correct for all of the files. Add the option `-i` or `--no-interactive` to turn this behavior off.

If you only want to reduce a subset of your data, you can select unwanted files in the header validation table GUI and right click to delete them from the current reduction run. Be sure to always keep the standard stars you need for fluxing in the data reduction.

If you want to only reduce the red arm or the blue arm, add the flag `--arm red` or `-a blue` for short.

If you want lots of intermediate debugging plots displayed to the screen, add the flag `--debug`.

If you want to check that your target traces have been correctly identified, and manually select them if they were missed, add the flag `--manual-extraction` or `-m` for short. Check out *Using the Manual Tracing and Manual Aperture GUIs* for more details.

If you want to fine-tune the reduction parameters, create a parameter file like so:

```
# params.cfg
[blue]
# PypeIt reduction parameters to control the blue side reduction
[reduce]
    [[skysub]]
        no_local_sky = True
    [[extraction]]
        use_2dmodel_mask = False
[red]
# PypeIt parameters to control the red side reduction
[reduce]
    [[skysub]]
        no_local_sky = True
    [[extraction]]
        use_2dmodel_mask = False
```

and use the option `-p params.cfg` or its longer form `--parameter-file params.cfg`.

See PypeIt Parameters for more details on the full set of available parameters.

1.5 DBSP Quicklook

During an observing run, in order to make time-sensitive decisions about what to observe, a quicklook script is provided.

1.5.1 Usage

```
$ dbsp_ql --help
usage: dbsp_ql [-h] [--no-show] fname

Quicklook for P200 DBSP

positional arguments:
  fname       file to take a quick look at, or else red/blue
              to just perform rough calibrations

optional arguments:
  -h, --help  show this help message and exit
  --no-show   Set this flag to suppress opening of plots
```

First, navigate to a directory you want the output data in.

```
$ cd /path/to/workdir
```

Then, with at least one arc frame and at least one flat frame in `/path/to/calibs` run

```
$ dbsp_ql /path/to/calibs/red
$ dbsp_ql /path/to/calibs/blue
```

to perform quick calibrations for the red and blue sides, respectively.

Then once you have a science frame (either in the same directory, or elsewhere) in `/path/to/science/data`, run

```
$ dbsp_ql /path/to/science/data/red0030.fits
$ dbsp_ql /path/to/science/data/blue0030.fits
```

This step should be very quick (about 15 seconds for the red side, 8 seconds for the blue side), and each command will pop up a ginga window for you to inspect the sky-subtracted frame. Also, a GUI will pop up to display the fluxed spectrum for each object identified in the frame.

The quicklook script produces PypeIt 2D Spectrum and 1D Spectrum files, described in *Data Reduction Outputs*.

1.6 Data Reduction Outputs

Assuming you ran DBSP_DRP with `dbsp_reduce -r $RAW_PATH -d $OUTPUT_PATH`, then in the `$OUTPUT_PATH/Science` directory you will find:

PypeIt 2D Spectrum files `spec2d_redNNNN-target_DBSPr_obstimestamp.fits` described in more detail here. Briefly, these files hold flat-fielded 2D spectral images, as well as sky, noise, object, wavelength, and bad pixel mask images. These files can be visually inspected using the command `pypeit_show_2dspec SPEC2D_FILE`.

PypeIt 1D Spectrum files `spec1d_redNNNN-target_DBSPr_obstimstamp.fits` described in more detail here. Briefly, these files hold 1D spectra for each object that was traced and extracted from the raw frame. Each object's spectrum is stored in a separate extension of the FITS file, and the extension names are of the form `SPATNNNN-SLITMMMM-DET01` where the SPAT number describes the spatial pixel coordinate of the object and the SLIT number is only useful for spectrographs with multiple slits (and the DET number is only useful for spectrographs with multiple detectors for the same arm). These files can be visually inspected using the command `pypeit_show_1dspec --exten N SPEC1D_FILE` to view extension N of the file.

PypeIt 1D Coadd files `redNNNN-redNNNN_target_SPATNNNN.fits` described in more detail here. These files exist to separate out multiple objects on the same frame into their own file, and to coadd consecutive exposures of the same frame. These files can be visually inspected using the command `lt_xspec COADD_FILE`. The filename contains the first and last raw data file and the medial spatial pixel coordinate of the object. This was done to make the coadd filename of constant length, instead of scaling with the number of coadded frames, since all operating systems/file systems have maximum allowed filename lengths.

Telluric-corrected 1D Coadd files have `_tellcorr` appended to the base filename of the coadd file.

In the directory `$OUTPUT_PATH/spliced` are the spliced final data products. The final data product of DBSP_DRP is a FITS file named `target_char.fits` with structure described below, where `char` is a one-letter designation of which object along the slit it is.

Table 1: table

Extension Number	Name	Header	Data
0	PRIMARY	Version info about DBSP_DRP, PypeIt, NumPy and Astropy	None
1	RED0031	Header from raw red side FITS file	Fluxed (not telluric-corrected) red side spectrum
2	RED0032	Header from raw red side FITS file	Fluxed (not telluric-corrected) red side spectrum
3	BLUE0032	Header from raw blue side FITS file	Fluxed blue side spectrum
4	BLUE0032	Header from raw blue side FITS file	Fluxed blue side spectrum
5	RED	Header from raw red side FITS file	Fluxed (and telluric-corrected) red side spectrum
6	BLUE	Header from raw blue side FITS file	Fluxed blue side spectrum
7	SPLICED	Empty	Final spliced spectrum

The header on the 0th extension also contains cards named `B_COADD` and `R_COADD` which contain the filename of the blue and red coadd files, respectively, that were spliced together. This is useful for determining which traces a particular final output file corresponds to. The 0th header also contains an `INTERP_GAPS` cards, noting whether or not detector gaps were interpolated over during splicing. If the splicing has been manually adjusted (see *Manual Splicing* for more details) then a `RED_MULT` card will also be present, recording the factor multiplied into the red coadd spectrum before splicing with the blue coadd.

If *n* red side files were coadded and *m* blue side files were coadded, then extensions 1 through *n* would contain the red side raw headers and fluxed spectrum from each individual file, extensions 1+*n* through 1+*n*+*m* would contain the blue side raw headers and fluxed spectra from each individual file, and the last 3 extensions are the red coadd, blue coadd, and final spliced spectrum.

If an object was not observed in the blue (red) then there will be no raw blue (red) frames, and the BLUE (RED) extension would still exist, but contain no data.

Each of the data tables contain columns for *wave*, *flux*, and *sigma*, with wavelength in Angstroms and both flux and sigma in units of 10^{-17}erg/s/cm^2/Ang.

1.7 Viewing Reduced Spectra

DBSP_DRP comes bundled with `dbsp_show`, an all-purpose interactive spectrum plotter for DBSP_DRP and intermediate `PypeIt` files, built using Matplotlib.

```
$ dbsp_show --help
usage: dbsp_show [-h] [--extension EXTENSION] [--BOX] [--COUNTS] fname

All-purpose interactive spectrum plotter for DBSP_DRP and intermediate PypeIt files.

positional arguments:
  fname                 path to FITS file

optional arguments:
  -h, --help            show this help message and exit
  --extension EXTENSION
                        Extension name or number
  --BOX                 Use boxcar extraction when plotting PypeIt spec1d files. By␣
↪default the optimal extraction is plotted.
  --COUNTS              Plot counts when plotting PypeIt spec1d files. By default flux␣
↪is plotted.
```

1.8 Trim Spectra By Wavelength

If you want to trim the wavelength range of your spliced spectra, e.g. to ignore data below the atmospheric cutoff, the command-line tool `dbsp_trim` is there to help you!

```
$ dbsp_trim --help
usage: dbsp_trim [-h] [--low LOW] [--high HIGH] fname [fname ...]

Trim spliced spectrum to wavelength range.

positional arguments:
fname        Final data output from DBSP_DRP to trim. Can be multiple files.

optional arguments:
  -h, --help   show this help message and exit
  --low LOW    Min wavelength after trimming. Default is 3300 Å.
  --high HIGH  Max wavelength after trimming. Default is 10500 Å.
```

To use `dbsp_trim` to trim all of your spliced spectra, located in the folder `spliced`, you can run the following command:

```
$ dbsp_trim --low 3200 --high 11000 spliced/*.fits
```

1.9 Using the Manual Tracing and Manual Aperture GUIs

1.9.1 Manual Tracing GUI

After the first round of reducing the red and/or blue sides, `dbsp_reduce` will open a Matplotlib window to display the sky-subtracted spectra, along with any object traces that were automatically detected.

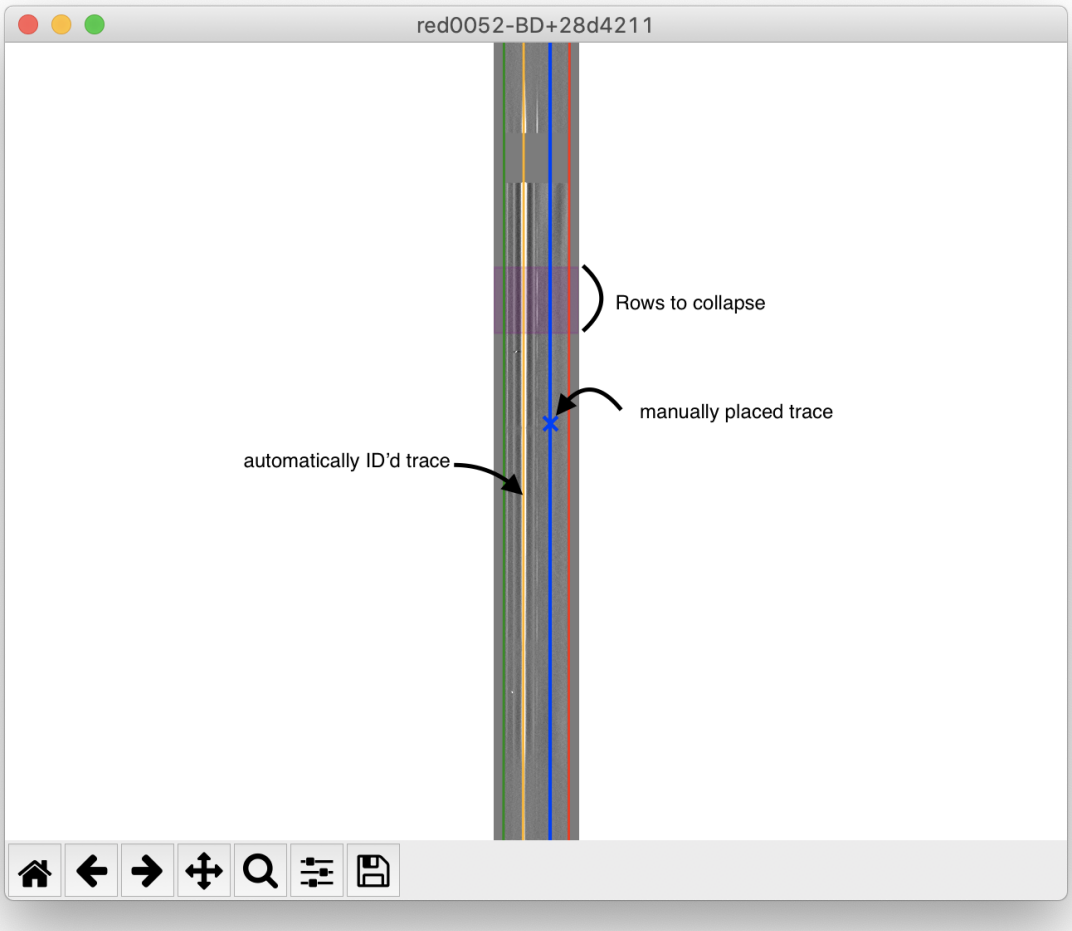
Using the left and right arrow keys, you can cycle through the spectra.

If your science target was not automatically detected, you can zoom in on the trace using the standard matplotlib zoom tool and then with your mouse over the trace, press the `m` key to mark that trace.

If you make a mistake, you can press `d` with your mouse over a previously marked trace to delete the trace.

To adjust the region of the spectrum that will be collapsed to select apertures and background regions, press `c`, then left click and drag to highlight the region to be collapsed in purple.

After you close this window, for each frame you marked with a manual trace, a window will pop up with a Manual Aperture and Sky Selection GUI.



1.9.2 Manual Aperture and Sky Selection GUI

This GUI shows the collapsed flux, along with any automatically identified traces (in orange) and your manually placed traces (in blue) along with the FWHM of each shaded in a lighter color.

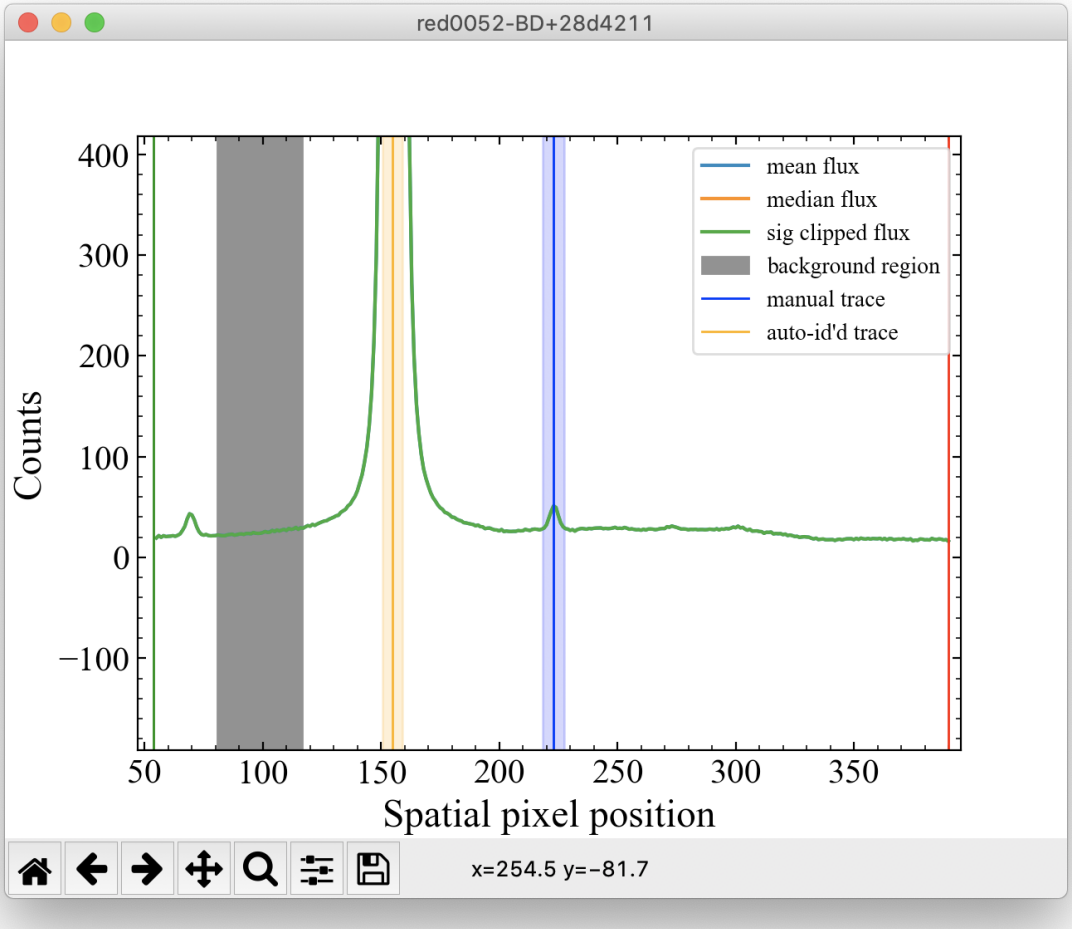
In this GUI, you can left click and drag your manual traces (in blue) to adjust their position.

Additionally, you can right click and drag the shaded FWHM regions to adjust their extent.

To mark background regions, press `b` and then left click and drag to mark background regions by shading them in gray.

You can delete a background region by holding your mouse over the shaded background regions and pressing `d` to delete.

Once you are finished adjusting manual traces/FWHMs and marking background regions, close the window to be shown the same GUI for the next object you marked a manual trace on.



Tips

Make sure to select background regions on either side of your target for the best sky subtraction.

If you are dealing with faint sources, it is a good idea to re-mark in blue any orange (automatically identified) traces in case parameter changes lose these objects.

1.10 DBSP_DRP Quality Assurance (QA) Outputs

The QA folder has two main QA pages: `MF_A.html` and `Extraction.html`.

`MF_A.html` is automatically generated by PypeIt and includes a number of QA plots regarding the calibration data, which currently only encompass the wavelength calibration. You can read in more detail about PypeIt's QA plots here.

In `Extraction.html`, for each target, there is a page that shows various steps of the reduction for the red and blue sides. From left to right, the images are flat-fielded frame, sky model, sky-subtracted frame, sky-subtracted residuals, and sky- and object-subtracted residuals. Above each set of images, the raw filename, time of observation and the airmass is displayed. Slit edges are marked in red and green, object traces are marked in orange, and extraction FWHMs are highlighted in orange. If the extraction FWHM is very different from what you would expect, it is probably a good idea to use manual tracing on that target.

1.11 Manual Splicing

In the case that the association of objects/traces between the red and blue sides of DBSP is incorrect, you can use the script `dbsp_splice` to manually stitch together two coadded spectra.

You must provide the path to the raw files, so that the raw headers can be packaged into the final data product, as well as the filename to save the spliced spectrum to, and the red and blue coadds you wish to splice together.

```
$ dbsp_splice --help
usage: dbsp_splice [-h] [-r RED_FILE] [-b BLUE_FILE] raw_data_path outfile

Manually splice two coadded files together for DBSP.
After preparing the spliced spectrum, the dbsp_adjust_splicing GUI pops up to
allow for the splicing to be adjusted. To save the manually spliced spectrum,
you MUST press save in the GUI.

positional arguments:
  raw_data_path         Path to raw data from this reduction
  outfile               Destination of spliced spectrum.

optional arguments:
  -h, --help            show this help message and exit
  -r RED_FILE, --red_file RED_FILE
                        redNNNN-redMMMM_SPATXXXX.fits file.
  -b BLUE_FILE, --blue_file BLUE_FILE
                        blueNNNN-redMMMM_SPATXXXX.fits file.
```

Note that although the `red_file` and `blue_file` arguments are optional, you must supply at least one of them.

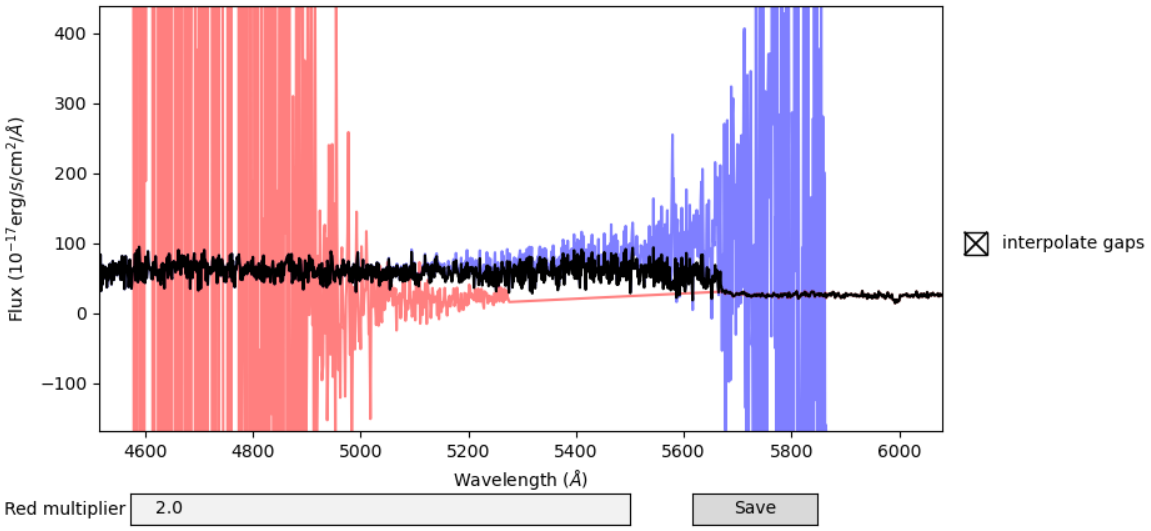
1.12 Adjusting splicing between arms

Typically the splicing between arms is quite good, and you only need to adjust it when the optimal extraction FWHM vary greatly between the red and blue sides. This can happen when an extended object is near another object on the slit. You can check the extraction FWHM visually by looking at the Extraction QA page.

```
$ dbsp_adjust_splicing --help
usage: dbsp_adjust_splicing [-h] fname [fname ...]

Red/Blue splicing adjustment for P200 DBSP

positional arguments:
  fname       Final data output from DBSP_DRP to adjust the splicing of. Can be one or␣
↪multiple files.

optional arguments:
  -h, --help  show this help message and exit
```

In this GUI, the blue and red spectra are plotted as well as the spliced spectrum. After using the usual Matplotlib tools to zoom and pan to the region of interest, you can adjust the red multiplier to multiply the red flux by a constant, and change if flux will be interpolated across detector gaps, as well as save your changes to the same file you opened. After you close the GUI, you will be presented with the next spectrum to adjust the splicing of.

The default splicing that DBSP_DRP does uses a red multiplier of 1, since the flux calibration provided by PypeIt is excellent.



1.13 DBSP_DRP Docker Image

Docker images are provided to allow for portable DBSP_DRP installations, or for easily maintaining an on-site DBSP_DRP installation. The Docker image `finagle29/dbsp_drp` contains a full installation of DBSP_DRP (i.e. including the telluric grid), and the Docker image `finagle29/dbsp_ql` contains a minimal installation of DBSP_DRP suitable for running the quicklook script. By appending a tag to the Docker image name, you can control which version of DBSP_DRP the downloaded Docker image contains:

Tag	Version	Example Name
(blank)	same as `latest` tag	`finagle29/dbsp_ql`
`latest`	latest released version (currently 1.0.0)	`finagle29/dbsp_ql:latest`
`main`	latest commit on `main` branch	`finagle29/dbsp_ql:main`
`edge`	latest commit on `develop` branch	`finagle29/dbsp_ql:edge`
`1.0.0`	version 1.0.0	`finagle29/dbsp_ql:1.0.0`

The shell script `run_docker.sh` is provided for ease of running DBSP_DRP in a Docker container. After installing Docker on your computer, with the `run_docker.sh` script in your working directory, you can start a Docker container with DBSP_DRP pre-installed using the following command-line invocation:

```
$ ./run_docker.sh -v /data/DBSP_20210919:/workdir/data finagle29/dbsp_drp:0.9.0
```

This will start the Docker container in the terminal window, and allow you to access the contents of the directory `/data/DBSP_20210919` on your computer in the directory `/workdir/data` in the Docker container.

1.14 dbsp_drp

1.14.1 dbsp_drp package

Subpackages

dbsp_drp.tests package

Submodules

dbsp_drp.tests.conftest module

dbsp_drp.tests.conftest.**files_path**() → str

dbsp_drp.tests.test_coadding module

dbsp_drp.tests.test_coadding.**test_group_coadds**()

dbsp_drp.tests.test_fluxing module

dbsp_drp.tests.test_fluxing.**test_archived_sensfunc_exist**()

dbsp_drp.tests.test_fluxing.**test_archived_sensfunc_read**()

dbsp_drp.tests.test_quicklook module

dbsp_drp.tests.test_quicklook.**quicklook_tester**(*wdir*, *path*, *fname*)

dbsp_drp.tests.test_quicklook.**test_quicklook_blue**(*files_path*, *working_dir*)

dbsp_drp.tests.test_quicklook.**test_quicklook_blue_calib**(*files_path*, *working_dir*)

dbsp_drp.tests.test_quicklook.**test_quicklook_red**(*files_path*, *working_dir*)

dbsp_drp.tests.test_quicklook.**test_quicklook_red_calib**(*files_path*, *working_dir*)

dbsp_drp.tests.test_quicklook.**working_dir**(*tmp_path_factory*)

dbsp_drp.tests.test_splicing module

dbsp_drp.tests.test_splicing.**interp_w_error_tester**(*xmin*, *xmax*, *xsize*, *xpmin*, *xpmax*, *xpsize*)

dbsp_drp.tests.test_splicing.**test_interp_w_error**()

dbsp_drp.tests.test_table_edit module

dbsp_drp.tests.test_table_edit.**astropy_table**(*pypeit_setup: PypeItSetup*) → Table

dbsp_drp.tests.test_table_edit.**pypeit_setup**(*tmp_path_factory*, *files_path*) → PypeItSetup

dbsp_drp.tests.test_table_edit.**table_model**(*astropy_table: Table*) → Callable[[tuple, list], *TableModel*]

dbsp_drp.tests.test_table_edit.**test_table_model_data**(*table_model:* TableModel) → None

dbsp_drp.tests.test_table_edit.**test_table_model_setdata**(*table_model:* TableModel) → None

dbsp_drp.tests.test_table_edit.**test_updating_fits_header**(*table_model:* TableModel) → None

Module contents

Submodules

dbsp_drp.adjust_splicing module

dbsp_drp.adjust_splicing.**adjust_splicing_GUI**(*hdul: HDUList*, *fname: str*)

Opens a Matplotlib GUI for users to manually adjust the matching of overall red/blue flux levels, as well as interpolate over detector gaps.

Parameters

- **hdul** (*fits.HDUList*) – HDUList read in from DBSP_DRP final data output file.
- **fname** (*str*) – Output path and filename to write adjusted spectrum to.

dbsp_drp.adjust_splicing.**entrypoint**()

dbsp_drp.adjust_splicing.**main**(*args: Namespace*)

dbsp_drp.adjust_splicing.**parse**(*options: List[str] | None = None*) → Namespace

dbsp_drp.coadding module

Automated coadding for P200 DBSP.

dbsp_drp.coadding.**coadd**(*grouped_spats_list: List[dict]*, *output_path: str*, *spectrograph: str*, *user_config_lines: List[str]*, *debug: bool = False*) → List[str]

Coadds objects specified in grouped_spats_list.

Parameters

- **grouped_spats_list** (*List[dict]*) – a list of dicts mapping `fnames` to a list of filenames and `spats` to a list of integer spatial pixel positions.
- **output_path** (*str*) – Coadded files will be written to `output_path/Science`
- **spectrograph** (*str*) – PypeIt name of spectrograph.
- **user_config_lines** (*List[str]*) – User-provided PypeIt configuration.
- **debug** (*bool, optional*) – Show debugging output/plots? Defaults to False.

Returns
List of filenames of coadded spectra.

Return type
List[str]

dbsp_drp.coadding.**coadd_one_object**(*spec1dfiles: List[str]*, *objids: List[str]*, *coaddfile: str*, *spectrograph: str*, *user_config_lines: List[str]*, *debug: bool = False*)

Coadds multiple 1D spectra of one object.

Parameters

- **spec1dfiles** (*List[str]*) – List of spec1d files to coadd.
- **objids** (*List[str]*) – List of object IDs to coadd.
- **coaddfile** (*str*) – Outpath path and filename for coadd file.
- **spectrograph** (*str*) – PypeIt name of spectrograph

- **user_config_lines** (*List[str]*) – User-provided PypeIt configuration
- **debug** (*bool, optional*) – Show debugging output/plots? Defaults to False.

dbsp_drp.coadding.**group_coadds**(*fname_to_spats: dict*)

Groups coadds. Destroys input.

Parameters
fname_to_spats (*dict*) – maps filenames to a list of integer spatial positions

Returns
List of dicts mapping 'fnames' to a list of filenames and 'spats' to a list of integer spatial positions.

Return type
List[Dict[str, Union[List[str], List[int]]]]

dbsp_drp.coadding.**make_coadd_filename**(*spec1d_filenames: List[str]*, *objids: List[str]*) → str

dbsp_drp.fix_headers module

dbsp_drp.fix_headers.**entrypoint**()

dbsp_drp.fix_headers.**main**(*path_prefix: str*, *throw_errors: bool*, *prompt_user: bool*) → List[str]

Fixes FITS files found in `path_prefix/*.fits` or `path_prefix*.fits` for common DBSP errors: empty file, extra bytes at the end, swapped GRATING/ANGLE values, missing/wrong RA/DEC/AIRMASS.

Parameters

- **path_prefix** (*str*) – Where to look for FITS files.
- **throw_errors** (*bool*) – Raise error when file cannot be automatically fixed?
- **prompt_user** (*bool*) – Prompt user to fix missing/bad headers?

Raises
ValueError – If `throw_errors`, raised when ANGLE and GRATING both cannot be parsed as angles.

Returns
list of files with headers good enough to start reduction.

Return type
List[str]

dbsp_drp.fluxing module

Automated fluxing for P200 DBSP.

dbsp_drp.fluxing.**build_fluxfile**(*spec1d_to_sensfunc: Dict[str, str]*, *output_path: str*, *spectrograph: str*, *user_config_lines: List[str]*) → str

Writes the fluxfile for fluxing.

Uses archived sensitivity function if no standard was reduced.

Parameters

- **spec1d_to_sensfunc** (*Dict[str,str]*) – maps spec1d filenames to the sensitivity function they should use
- **output_path** (*str*) – reduction output path

- **spectrograph** (*str*) – PypeIt name of spectrograph.
- **user_config_lines** (*List[str]*) – User-provided PypeIt configuration.

Returns
path to created fluxfile

Return type
str

dbsp_drp.fluxing.**flux**(*flux_file: str*, *output_path: str*, *debug: bool = False*) → None

Flux spectra.

Parameters

- **flux_file** (*str*) – Path to flux file
- **output_path** (*str*) – reduction output path
- **debug** (*bool, optional*) – Show debugging output/plots? Defaults to False.

dbsp_drp.fluxing.**make_sensfunc**(*standard_file: str*, *output_path: str*, *spectrograph: str*, *user_config_lines: List[str]*, *debug: bool = False*) → str

Makes a sensitivity function.

Parameters

- **standard_file** (*str*) – Filename of standard exposure.
- **output_path** (*str*) – Partial path to standard exposure.
- **spectrograph** (*str*) – PypeIt name of spectrograph.
- **user_config_lines** (*List[str]*) – User-provided PypeIt configuration.
- **debug** (*bool, optional*) – Show debugging output/plots? Defaults to False.

Returns
Filename of sensitivity function file, or empty string on failure.

Return type
str

dbsp_drp.gui_helpers module

Module for classes and functions shared between GUIs.

class dbsp_drp.gui_helpers.**HelpTool**(**args*, *helptext=''*, ***kwargs*)

Bases: `ToolBase`

Print help text.

default_keymap = 'h'

Keymap to associate with this tool.

String: List of comma separated keys that will be used to call this tool when the keypress event of `self.figure.canvas` is emitted.

description = 'Help'

Description of the Tool.

String: If the Tool is included in the Toolbar this text is used as a Tooltip.

image = 'help'

Filename of the image.

String: Filename of the image to use in the toolbar. If None, the *name* is used as a label in the toolbar button.

trigger(**args*, ***kwargs*)

Called when this tool gets used.

This method is called by *matplotlib.backend_managers.ToolManager.trigger_tool*.

Parameters

- **event** (*.Event*) – The canvas event that caused this tool to be called.
- **sender** (*object*) – Object that requested the tool to be triggered.
- **data** (*object*) – Extra data.

dbsp_drp.manual_aperture module

class dbsp_drp.manual_aperture.**ManualApertureGUI**(*figure: Figure*, *axes: Axes*, *target*, *spec: Spec2DObj*, *traces*, *edges*, *fitted_fwhms*, *manual_traces*, *collapse_region*)

Bases: object

GUI for manually selecting background and signal for object traces

How to use:

- to designate background regions, press (b)ackground and then click and drag
- left-click and drag blue (manual) traces around
- right-click and drag to change FWHM of blue (manual) traces
- press (d)elete to delete background regions under the cursor

delete(*event*)

Delete a background area under (or nearby) the mouse

The x-tolerance is hardcoded to 5 CCD pixels.

drag_draw_bg()

drag_draw_fwhm()

drag_draw_manual_trace()

get_background_areas()

get_fwhms()

get_manual_traces()

key_press_callback(*event*)

property mask

motion_notify_callback(*event*)

mouse_press_callback(*event*)

mouse_release_callback(*event*)

normalize_bgs()

Merges overlapping background areas.

Call only after the user is done interacting with the GUI

pick_callback(*event*)

plot()

dbsp_drp.manual_splice module

dbsp_drp.manual_splice.**entrypoint**()

dbsp_drp.manual_splice.**find_spec1ds_spats**(*history: History*)

dbsp_drp.manual_splice.**main**(*args: Namespace*)

dbsp_drp.manual_splice.**parse**(*options: List[str] | None = None*) → Namespace

dbsp_drp.manual_tracing module

class dbsp_drp.manual_tracing.**ManualTracingGUI**(*specs_dict*)

Bases: `object`

GUI for manually identifying object traces

Takes in specs_dict:

```
specs_dict[target_name] == {
    'spec': spectrum_2d,
    'edges': [all_left_edges, all_right_edges],
    'traces': traces,
    'fwhms': fwhms,
    'fwhmfit': automatically fit fwhms
}
```

How to use: Displays one spectrum at a time use left and right arrow keys to go back and forth between spectra press m to lay down a (m)anual trace under your cursor press d to (d)elete a manual trace near your cursor press c, then left-click and drag to set the region to view the (c)ollapsed flux

compute_sky_resid(*spec*)

draw_dragging_collapse_region()

property edges

property fwhmfit

helptext = 'GUI for manually identifying object traces\n Displays one spectrum at a time\n\n How to use:\n - use left and right arrow keys to go back and forth between spectra\n - press m to lay down a (m)anual trace under your cursor\n - press d to (d)elete a manual trace near your cursor\n - press c, then left-click and drag to set the region to view the (c)ollapsed flux\n '

key_press_callback(*event*)

property mask

motion_notify_callback(*event*)

mouse_press_callback(*event*)

mouse_release_callback(*event*)

plot()

plot_manual_traces()

property sky_resid

property spec

property target

property traces

dbsp_drp.p200_redux module

Automatic Reduction Pipeline for P200 DBSP.

dbsp_drp.p200_redux.**entrypoint**()

dbsp_drp.p200_redux.**interactive_correction**(*ps: PypeItSetup*) → None

Allows for human correction of FITS headers and frame typing.

Launches a GUI via dbsp_drp.table_edit, which handles saving updated FITS headers. table_edit depends on the current DBSP headers.

Parameters
ps (*PypeItSetup*) – PypeItSetup object created in dbsp_drp.reduction.setup

dbsp_drp.p200_redux.**main**(*args*)

dbsp_drp.p200_redux.**parser**(*options: List[str] | None = None*) → Namespace

Parses command line arguments

Parameters
options (*Optional[List[str]], optional*) – List of command line arguments. Defaults to sys.argv[1:].

Returns
Parsed arguments

Return type
argparse.Namespace

dbsp_drp.qa module

Quality Assurance for automated reduction of P200 DBSP.

dbsp_drp.qa.**save_2dspecs**(*qa_dict: dict*, *output_spec2ds: List[str]*, *output_path: str*, *spectrograph: str*) → dict

Saves PNG images of each spectra at various steps in the reduction process. In one PNG, displayed from left to right are science image, sky model, sky-subtracted image, sky-subtraction residuals, sky- and object-subtraction residuals.

Parameters

- **qa_dict** (*dict*) – Dict mapping target names to a dict containing lists of PNG filenames, airmasses, UTCs, and raw data filenames, from previous call of `save_2dspecs`, or empty dict for first call.
- **output_spec2ds** (*List[str]*) – List of spec2d filenames.
- **output_path** (*str*) – reduction output path.
- **spectrograph** (*str*) – PypeIt name of spectrograph.

Returns
updated `qa_dict`

Return type
dict

dbsp_drp.qa.**save_one2dspec**(*ax: Axes*, *spec: ndarray*, *edges: Tuple[ndarray, ndarray]*, *traces: List[ndarray]*, *fwhms: List[ndarray]*) → None

Displays a 2D spectrum on the input Matplotlib Axes, with slit edges and object traces and extraction FWHMs overplotted.

Parameters

- **ax** (*plt.Axes*) – Axes object for plotting.
- **spec** (*np.ndarray*) – 2D spectrum to plot.
- **edges** (*Tuple[np.ndarray, np.ndarray]*) – Left and right slit edges, as arrays of the spatial pixel position of the edge for each pixel in the spectral direction.
- **traces** (*List[np.ndarray]*) – Object traces as arrays of spatial pixel positions for each pixel in the spectral direction.
- **fwhms** (*List[np.ndarray]*) – Extraction FWHMs for each object trace.

dbsp_drp.qa.**write_extraction_QA**(*qa_dict: dict*, *output_path: str*) → None

Write Extraction.html page allowing for convenient viewing of PNGs saved in `save_2dspecs`.

Parameters

- **qa_dict** (*dict*) – Dict mapping target names to a dict containing lists of PNG filenames, airmasses, UTCs, and raw data filenames.
- **output_path** (*str*) – reduction output path

dbsp_drp.quicklook module

dbsp_drp.quicklook.**entrypoint**()

dbsp_drp.quicklook.**get_cfg_lines**(*spectrograph: str*) → List[str]

Get standard quicklook PypeIt configuration for `spectrograph`.

Parameters
spectrograph (*str*) – PypeIt name of spectrograph.

Returns
Standard PypeIt quicklook configuration for `spectrograph`.

Return type
List[str]

dbsp_drp.quicklook.**main**(*args: Namespace*)

dbsp_drp.quicklook.**parse**(*options: List[str] | None = None*) → Namespace

dbsp_drp.quicklook.**show_spec1d_helper**(*exten*, *file*)

dbsp_drp.quicklook.**show_spec2d_helper**(*file*)

dbsp_drp.reduction module

Automated reduction steps for P200 DBSP.

dbsp_drp.reduction.**delete_completely_masked_hdus**(*path: str*, *base_name: str = ''*)

Removes `SpecObj` s that are completely masked, i.e. no good pixels/data, from a FITS file containing a `SpecObjs` file.

Parameters

- **path** (*str*) – Path to FITS file.
- **base_name** (*str, optional*) – Name of file used for logging. Defaults to "".

dbsp_drp.reduction.**delete_duplicate_hdus_by_name**(*path: str*, *base_name: str = ''*)

Removes `SpecObj` s with identical names from a FITS file containing a `SpecObjs` object, leaving one `SpecObj` with the duplicate name behind.

Parameters

- **path** (*str*) – Path to FITS file.
- **base_name** (*str, optional*) – Name of file used for logging. Defaults to "".

dbsp_drp.reduction.**manual_extraction**(*output_spec2ds: List[str]*, *pypeit_file: str*, *output_path: str*) → list

wglwk

Parameters

- **output_spec2ds** (*List[str]*) – List of spec2d files to potentially manually extract.
- **pypeit_file** (*str*) – PypeIt reduction file for initial reduction.
- **output_path** (*str*) – reduction output path.

Returns
List of new PypeIt reduction files for the manually traced targets.

Return type
list

dbsp_drp.reduction.**manual_extraction_GUI**(*output_spec2ds: List[str]*, *output_path: str*) → dict

Runs the Manual Tracing GUI.

Parameters

- **output_spec2ds** (*List[str]*) – List of spec2d files to inspect in GUI.
- **output_path** (*str*) – reduction output path

Returns

Maps target names needing manual tracing to a dict specifying how
they should be manually traced.

Return type
dict

dbsp_drp.reduction.**parse_pypeit_parameter_file**(*parameter_file: str*, *spectrograph: str*) → List[str]

Grab user-provided PypeIt configuration for `spectrograph` from `parameter_file`, which contains user-provided PypeIt configuration for both arms of DBSP.

Parameters

- **parameter_file** (*str*) – User-provided file with their PypeIt config.
- **spectrograph** (*str*) – PypeIt name of spectrograph.

Returns
User-provided PypeIt configuration for `spectrograph`

Return type
List[str]

dbsp_drp.reduction.**re_redux**(*pypeit_files: List[str]*, *output_path: str*) → Tuple[set, set]

Runs multiple reductions, returns the combined sets of output spec1d, spec2d files.

Parameters

- **pypeit_files** (*List[str]*) – List of PypeIt reduction files to run.
- **output_path** (*str*) – reduction output path

Returns
set of filenames of reduced (spec1d, spec2d) files.

Return type
Tuple[set, set]

dbsp_drp.reduction.**redux**(*pypeit_file: str*, *output_path: str*, *reuse_masters: bool = True*, *show: bool = False*, *calib_only: bool = False*) → Tuple[set, set]

Runs the reduction

Parameters

- **pypeit_file** (*str*) – Path to PypeIt reduction file.
- **output_path** (*str*) – reduction output path
- **reuse_masters** (*bool, optional*) – Reuse master calibration files (if they exist). Defaults to True.
- **show** (*bool, optional*) – Show debugging/intermediate plots. Defaults to False.

- **calib_only** (*bool, optional*) – Only perform calibration? Defaults to False.

Returns
set of filenames of reduced (spec1d, spec2d) files.

Return type
Tuple[set, set]

dbsp_drp.reduction.**search_table_for_arc**(*row: Row*, *i: int*, *table: Table*, *step: int*, *max_sep: Angle*) → Tuple[int, int]

Searches table starting at row `i` in steps of `step` for frames within `max_separation` of `row`. If an arc frame is found, the arc's `calib` is assigned to `row`.

Parameters

- **row** (*Row*) – starting row
- **i** (*int*) – index of starting row
- **table** (*Table*) – table to search in and modify
- **step** (*int*) – direction to look in. 1 for forwards, -1 for backwards
- **max_sep** (*Angle*) – maximum separation allowed between frames pointing at the same object

Returns

(distance to arc frame, calib ID) or (-1, -1) if
no arc found.

Return type
Tuple[int, int]

dbsp_drp.reduction.**set_calibs**(*table: Table*)

Automagically set 'calib' column for .pypeit file.

Bias and flat frames get calib 'all' Arcs at airmass 1.0 get calib 0 (default for science/standards) A consecutive set of arcs at airmass > 1.0 get the same calib, starting at 1. Science and standards get the calib of the nearest arc frame, iff the telescope hasn't changed pointing between the arc and the science/standard. If there is no arc frame with the same pointing, science and standards are assigned calib 0.

Parameters
table (*Table*) – PypeItSetup.fitstbl.table

dbsp_drp.reduction.**setup**(*file_list: List[str]*, *output_path: str*, *spectrograph: str*) → Tuple[PypeItSetup, str]

Does PypeIt setup, without writing the .pypeit file

Parameters

- **file_list** (*List[str]*) – List of raw data files to reduce.
- **output_path** (*str*) – reduction output path
- **spectrograph** (*str*) – PypeIt name of spectrograph.

Returns
PypeItSetup object, reduction output path

Return type
Tuple[PypeItSetup, str]

dbsp_drp.reduction.**verify_spec1ds**(*output_spec1ds: List[str]*, *verification_counter: int*, *output_path: str*) → List[str]

Verifies validity of spec1d files, fixes some, and generates and returns a list of pypeit files for targets that need to be rerun.

Parameters

- **output_spec1ds** (*List[str]*) – List of spec1d filenames
- **verification_counter** (*int*) – number of times verification has been run
- **output_path** (*str*) – reduction output path

Returns
List of PypeIt files for targets that need to be rereduced.

Return type
List[str]

dbsp_drp.reduction.**write_manual_pypeit_files**(*old_pypeit_file: str*, *targets_list: List[List[str]]*, *manual_lines_fn: Callable[[List[str]], List[str]]*, *needs_std_fn: Callable[[str], bool]*) → List[str]

Writes pypeit files based on the default pypeit file for this reduction, but with filtered targets and additional manual parameter lines.

Parameters

- **old_pypeit_file** (*str*) – default pypeit file,
- **targets_list** (*List[List[str]]*) – [[target1, target2], [target3, target4]] will result in 1 & 2 being reduced together and 3 & 4 being reduced together. The target names are blueNNNN-ZTF21abcd.
- **manual_lines_fn** (*Callable[[List[str]], List[str]]*) – function mapping [target1, target2] to the cfg lines they need.
- **needs_std_fn** (*Callable[[str], bool]*) – function mapping target1 -> bool. True if target1 needs a standard star to be reduced alongside it.

Returns

List of new PypeIt reduction files for the manually traced
targets.

Return type
List[str]

dbsp_drp.reduction.**write_setup**(*context: Tuple[PypeItSetup, str]*, *cfg_split: str*, *spectrograph: str*, *user_config_lines: List[str]*) → List[str]

Writes the .pypeit file

Parameters

- **context** (*Tuple[PypeItSetup, str]*) – PypeItSetup object, reduction output path
- **cfg_split** (*str*) – [description]
- **spectrograph** (*str*) – PypeIt name of spectrograph.
- **user_config_lines** (*List[str]*) – User-provided PypeIt configuration.

Raises
RuntimeError – Raised if files were not assigned a frame type.

Returns
List of PypeIt files generated.

Return type
List[str]

dbsp_drp.show_spectrum module

All-purpose interactive spectrum plotter for DBSP_DRP and intermediate PypeIt files.

dbsp_drp.show_spectrum.**entrypoint**()

dbsp_drp.show_spectrum.**main**(*args: Namespace*) → None

Attempts to parse input FITS file as `OneSpec`, `SpecObjs`, and DBSP_DRP final data output.

Parameters
args (*argparse.Namespace*) – input arguments

Raises

- **LookupError** – Raised if the input extension is not found in the input FITS file.
- **IndexError** – Raised if the input integer extension is outside of the valid range for indexing the input FITS file's extensions.

dbsp_drp.show_spectrum.**parser**(*options: List[str] | None = None*) → Namespace

dbsp_drp.show_spectrum.**plot**(*wave: ndarray*, *flux: ndarray*, *err: ndarray*, *title: str*) → None

Plots spectrum and error with sensible y-scale limits.

Flux and error have units of $10^{-17}\mathrm{erg/s/cm^2/\mathring{A}}$.

Parameters

- **wave** (*np.ndarray*) – Wavelengh array
- **flux** (*np.ndarray*) – Flux array
- **err** (*np.ndarray*) – Flux error array
- **title** (*str*) – Plot title.

dbsp_drp.show_spectrum.**sensible_ylim**(*wave: ndarray*, *flux: ndarray*) → Tuple[float, float]

Returns a tuple of sensible y-axis limits for plotting spectra.

Usage:

```
>>> plt.step(wave, flux)
>>> plt.ylim(sensible_ylim(wave, flux))
>>> plt.show()
```

Parameters

- **wave** (*np.ndarray*) – wavelength array
- **flux** (*np.ndarray*) – flux array

Returns
bottom, top

Return type
Tuple[float, float]

dbsp_drp.splicing module

Automated splicing for P200 DBSP.

dbsp_drp.splicing.**adjust_and_combine_overlap**(*spec_b: FITS_rec*, *spec_r: FITS_rec*, *interpolate_gaps: bool*, *red_mult: float = 1.0*) → Tuple[Tuple[ndarray, ndarray, ndarray], Tuple[ndarray, ndarray, ndarray], Tuple[ndarray, ndarray, ndarray]]

Takes in red and blue spectra, adjusts overall flux level by red_mult, and combines overlap region.

In the overlap region, the red spectrum is linearly interpolated to match the blue spectrum's wavelength spacing.

Parameters

- **spec_b** (*fits.FITS_rec*) – blue spectrum
- **spec_r** (*fits.FITS_rec*) – red spectrum.
- **interpolate_gaps** (*bool*) – Interpolate across gaps in wavelength coverage?
- **red_mult** (*float, optional*) – Factor multiplied into the red spectrum to match overal flux level with the blue spectrum. Defaults to 1.0.

Raises

ValueError – Raised when both *spec_b* and *spec_r* are empty or None.

Returns

Tuple[
Tuple[np.ndarray, np.ndarray, np.ndarray], Tuple[np.ndarray, np.ndarray, np.ndarray], Tuple[np.ndarray, np.ndarray, np.ndarray]

]: (blue, red, combined) spectra, where each spectrum is a tuple of
(wavelengths, flux, error)

dbsp_drp.splicing.**get_raw_hdus_from_spec1d**(*spec1d_list: List[Tuple[str, int]]*, *root: str*, *output_path: str*) → List[BinTableHDU]

Returns list of `fits.BinTableHDU` s, each containing the raw header and the 1D spectrum, of the input spec1d files.

Parameters

- **spec1d_list** (*List[Tuple[str, int]]*) – List of (spec1d filename, spatial pixel coordinate)
- **root** (*str*) – Path to raw data files, possibly including filename prefix.
- **output_path** (*str*) – reduction output path

Returns

List of raw data headers and data from input
spec1d files.

Return type
List[fits.BinTableHDU]

dbsp_drp.splicing.**interp_w_error**(*x: ndarray*, *xp: ndarray*, *yp: ndarray*, *err_yp: ndarray*, *interpolate_gaps: bool*) → Tuple[ndarray, ndarray]

Linearly interpolate the data points (`xp`, `yp`) with `err_yp` uncertainty onto the grid `x`.

Parameters

- **x** (*np.ndarray*) – destination x data

- **xp** (*np.ndarray*) – source x data
- **yp** (*np.ndarray*) – source y data
- **err_yp** (*np.ndarray*) – source y error data
- **interpolate_gaps** (*bool*) – Interpolate across gaps in xp?

Returns
Interpolated y and error.

Return type
Tuple[np.ndarray, np.ndarray]

dbsp_drp.splicing.**splice**(*splicing_dict: dict*, *interpolate_gaps: bool*, *root: str*, *output_path: str*) → None

Splices red and blue spectra together.

```
splicing_dict[target_name][position_along_slit][arm] = {
    'spec1ds': [(spec1d_filename_1, spatial_pixel_1), (spec1d_filename_2, spatial_
↪pixel_2)],
    'coadd': coadd_filename
}
```

Parameters

- **splicing_dict** (*dict*) – Guides splicing.
- **interpolate_gaps** (*bool*) – Interpolate across gaps in wavelength coverage?
- **root** (*str*) – Path to raw data files, possibly including filename prefix.
- **output_path** (*str*) – reduction output path

dbsp_drp.table_edit module

class dbsp_drp.table_edit.**Delegate**(*parent*, *cols: tuple*)

Bases: QStyledItemDelegate

createEditor(*self*, *parent: PySide2.QtWidgets.QWidget*, *option: PySide2.QtWidgets.QStyleOptionViewItem*, *index: PySide2.QtCore.QModelIndex*) → PySide2.QtWidgets.QWidget

setEditorData(*self*, *editor: PySide2.QtWidgets.QWidget*, *index: PySide2.QtCore.QModelIndex*) → None

setModelData(*self*, *editor: PySide2.QtWidgets.QWidget*, *model: PySide2.QtCore.QAbstractItemModel*, *index: PySide2.QtCore.QModelIndex*) → None

staticMetaObject = <PySide2.QtCore.QMetaObject object>

updateEditorGeometry(*self*, *editor: PySide2.QtWidgets.QWidget*, *option: PySide2.QtWidgets.QStyleOptionViewItem*, *index: PySide2.QtCore.QModelIndex*) → None

class dbsp_drp.table_edit.**MainWindow**(*data: Table*, *cols: tuple | None = None*, *del_files: list | None = None*)

Bases: QMainWindow

closeEvent(*self*, *event: PySide2.QtGui.QCloseEvent*) → None

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class dbsp_drp.table_edit.**TableModel**(*data: Table*, *cols: tuple | None = None*, *del_files: list | None = None*)

Bases: QAbstractTableModel

columnCount(*self*, *parent: PySide2.QtCore.QModelIndex = Invalid(PySide2.QtCore.QModelIndex)*) → int

data(*self*, *index: PySide2.QtCore.QModelIndex*, *role: int = PySide2.QtCore.Qt.ItemDataRole.DisplayRole*) → Any

flags(*self*, *index: PySide2.QtCore.QModelIndex*) → PySide2.QtCore.Qt.ItemFlags

frametype(*index: QModelIndex*) → str

headerData(*self*, *section: int*, *orientation: PySide2.QtCore.Qt.Orientation*, *role: int = PySide2.QtCore.Qt.ItemDataRole.DisplayRole*) → Any

removeRows(*self*, *row: int*, *count: int*, *parent: PySide2.QtCore.QModelIndex = Invalid(PySide2.QtCore.QModelIndex)*) → bool

rowCount(*self*, *parent: PySide2.QtCore.QModelIndex = Invalid(PySide2.QtCore.QModelIndex)*) → int

setData(*self*, *index: PySide2.QtCore.QModelIndex*, *value: Any*, *role: int = PySide2.QtCore.Qt.ItemDataRole.EditRole*) → bool

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class dbsp_drp.table_edit.**TableView**(*model: QAbstractTableModel*, *parent=None*)

Bases: QTableView

show_context_menu(*point: QPoint*) → None

staticMetaObject = <PySide2.QtCore.QMetaObject object>

dbsp_drp.table_edit.**get_zenith_ra_dec**(*time: str*) → SkyCoord

Returns RA and Dec of the zenith at Palomar at the given time

Parameters
time (*str*) – MJD time

Returns
object containing the zenith RA and Dec

Return type
astropy.coordinates.SkyCoord

dbsp_drp.table_edit.**main**(*table: Table*, *del_files: List[str]*)

Opens header/metadata editing table GUI.

Parameters

- **table** (*Table*) – table containing metadata
- **del_files** (*List[str]*) – List of files to be deleted. Mutated!

dbsp_drp.table_edit.**update_airmass**(*row: Row*) → None

Updates the `airmass` entry of `row` based on the `ra` `dec` and `mjd` entries.

Parameters
row (*astropy.table.Row*) – Row to be updated

dbsp_drp.telluric module

Telluric correction for P200 DBSP.

dbsp_drp.telluric.**picklable_telluric_correct**(*args*)

dbsp_drp.telluric.**telluric_correct**(*coadd: str*, *output_path: str*, *spectrograph: str*, *user_config_lines: List[str]*, *debug: bool = False*, *plot: bool = False*)

Telluric correct one coadd file.

Parameters

- **coadd** (*str*) – Coadd filename.
- **output_path** (*str*) – reduction output path
- **spectrograph** (*str*) – PypeIt name of spectrograph.
- **user_config_lines** (*List[str]*) – User-provided PypeIt configuration
- **debug** (*bool, optional*) – Show debugging output? Defaults to False.
- **plot** (*bool, optional*) – Show debugging plots? Defaults to False.

dbsp_drp.trim module

Command-line tool for tripping spliced spectrum to a specified wavelength range.

dbsp_drp.trim.**entrypoint**()

dbsp_drp.trim.**main**(*args: Namespace*)

dbsp_drp.trim.**parse**(*options: List[str] | None = None*) → Namespace

dbsp_drp.version module

Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX

E

F

G

H

I

K

M

N

P

Q

R

S

T

U

V

W