CosmosPipeline Documentation

Gregory Walth

September 5, 2007

1 Introduction

The CosmosPipeline is a suite of Python scripts designed to simplify the interaction with COSMOS and the reduction of IMACS and LDSS3 data. The pipeline creates Makefiles that are dependent on the completion of certain stages. The Makefiles keep a record of the routines run as well as allow for modifications for particular data sets. The CosmosPipeline was created by Dan Kelson and Greg Walth.

The pipeline adds additional steps on top of the COSMOS package in which we also correct any scale errors, run 1d extraction on the data, flux the data with the box stars, output the data into a equispec format and measure redshifts.

The scale errors are corrected by fitting Gaussian to the spatial coordinate object positions. The 1d extraction uses the global trace of the brightest objects and extracts the data from a pre-defined aperture. Object spectra is fluxed by flatting and extracting a flat-field and then using a reference object(s) that can be used to fit a 2nd or 3rd degree Legendre polynomial. The redshifts of the objects are found by running a cross-correlation with Sloan templates.

2 Requirements

2.1 Data

The following data are needed to run the CosmosPipeline:

1. Science frames
2. Flat frames
3. Lamp frames
4. SMF file

2.2 Computing

The following computing requirements are needed to run the CosmosPipeline[1]:

1. Moderately fast computer:
   Faster or equivalent to a Pentium 4 (PC) or G4 (Mac).
2. Lots of RAM:
   More than 1GB.
3. Lots of hard drive space:
   For one science, flat and lamp about 3 GB of data are generated.
4. Linux or Mac OS X

3 Installation

3.1 Santa Barbara Street

To access all of the tasks needed by this pipeline all you need to type:

```
source /data1/chiaki/gwalth/Cosmos/CosmosSetup
```

This can also be added into your .cshrc or cshdef file.

3.2 Elsewhere

The following needs to be downloaded in order to run the CosmosPipeline on your own system:

1. COSMOS
   http://www.ociw.edu/Code/cosmos
2. Python Distribution
   - Redhat 9, PowerPC G5 Mac OS X
     http://www.ociw.edu/Code/python/kelson/
   - Intel Mac OS X
     http://www.ociw.edu/Code/python/burns-python-scripts/
3. Python Scripts
   http://www.ociw.edu/Code/python/walth

[1] As of now the pipeline works only with csh/tcsh.

For the Python Distribution, you will need to download the file Python.*System-Date*.tar.gz or Python.*System-Date*.dmg The Python Distribution has been compiled for Redhat 9 and Mac OS X 10.4 for the G4, G5 and Intel. The Redhat 9 distribution should work on most Linux systems with a similar CPU. The Mac distributions are available as dmg files.

In order to install the Python Distribution and Python Scripts you will need to untar the two archives and add the following lines to your .cshrc/.tcshrc or entered in your csh/tcsh terminal[2]:

```
setenv PYTHONHOME /dir/to/python/distribution/Python.`uname`
source $PYTHONHOME/PythonSetup

setenv PYTHONHOME1 /dir/to/python/root
setenv PATH ${PYTHONHOME1}/scripts:${PATH}
setenv PYTHONPATH ${PYTHONHOME1}/scripts:${PYTHONPATH}
```

You should be able to run things immediately. If you run into problems you may need to re-compile the ATLAS, along with Numeric and Scipy. More extreme cases, you may need to re-compile everything, for say an OS like Debian.

4 Before starting

The CosmosPipeline will create directories and files in the current directory you are working in. It is usually a good idea to create a directory (parent directory) to generate and run the pipelines in.

You will need to copy all SMF files into parent pipeline directory. Also check the Makefiles inside of each target directory to ensure that the SMF file names are consistent.

5 Setting up the database

The script **Cosmosdb.py** generates a database file by reading the headers of the fits files.

In order to create a database, type the following command:

```
Cosmosdb.py -d DATADIR -ins INSTRUMENT
```

Where *DATADIR* is either a relative or absolute path to the directory containing your fits files and *INSTRUMENT* is either ldss3 or imacs.

For example:

[2] PYTHONHOME1 needs to be set the directory containing the scripts directory, linelists and etc.

```
csh%> cd /data1/hidari/gwalth/IMACS
csh%> Cosmosdb.py -d examples/ms1054b -ins imacs
```

This will create a database file named *ms1054b.db*. Inside an IMACS database file you will find the following keywords: filename, date, start, instr, filter, disperser, g-angle, slitmask, exptime, dewarori, object

LDSS3 header keywords vary just slightly: filename, date, start, instr, filter, grism, aperture, exptime, object

The *object* keyword is supplied by the observer when the data is taken at the telescope. For combined image extractions the *object* keywords should all be the same. For separate image extractions the *object* keyword should be different. If inconsistencies exist, you should edit the database file such that the names are consistent, **CosmosPipeline.py** will be able to determine the name change. If the task become too great, another helpful program **CosmosdbFix.py** will allow you to fix various keywords, including the *object* keyword by supplying ranges and values.

If you have multiple directories that wish to reduce, all you need to do is supply the -d argument with each directory delimited by spaces.

```
Cosmosdb.py -d DATADIR1 DATADIR2 DATADIR3
```

There is also gzip file option *-gz*. Though it it probably more efficient to gunzip the files before running any of the database programs. The gzip file option will gunzip the files in a new location and preserving the gzipped files. This convention is adapted since not all data are in locations that can be unzipped.

6 Generating Pipelines

The simplest way to create a data pipeline is to type the following:

```
CosmosPipeline.py -db DATABASEFILE -ins INSTRUMENT
```

CosmosPipeline.py looks at the header keywords when generating the pipelines. The keyword of interest is the *object* keyword. There are certain keywords that **CosmosPipeline.py** looks for in the *object* keyword:

Flats:	flat, incan
Lamps:	lamp, arc, henear, he ne ar, he-ne-ar, he+ne+ar

In order to look for object spectra, I employ a blacklist method in order to filter the database. I have yet to have a problem with it but it may be possible to have some spectra that are missed or some that are categorized improperly.

Blacklist: Flats + Lamps + align, bias, mask, direct, image

Since I cannot possible guess every combination of what you may call your objects while observing it is usually a good idea to look through the database file. Usually the safest method of operation is to go through and comment out the files that you are not using with a #. You also must make sure that the *object* keywords are consistent. Fortunately you can change the *object* keyword inside the database file to whatever you like and **CosmosPipeline.py** will not barf.

If you have two objects with the same name and you want to do separate extractions, you will need to distinguish them without using spaces.

7 Overview

7.1 Stages

Running the pipeline until stage2[3] will provide you with almost the same output as if you were just running COSMOS. If you run the pipeline until stage3, you will end up with 2d and 1d extracted spectra. The remaining stages are optional stages for fluxing and finding redshifts for your data.

stage1
Runs COSMOS to **adjust-map**.

stage2
Runs COSMOS from **Sflats** to **extract-2dspec** producing 2d spectra in the extensions format.

stage3
Uses the 2d spectra from stage2, fits to the positions of the brightest objects and creates a new SMF file based on new corrected positions for each science frame. Separates the map file according to whether the object was in a *hole* or a *slit* in order to run **subsky** with different parameters for *holes* and *slits*. *holes* and *slits* are recombined and 1d and 2d spectra are produced in equispec format.

stage4
Fluxes the data based on a reference object (optional).

[3]Since HOLES and SLITS are both reduced in the CosmosPipline, values from the HOLES are used for **subsky** in stage2. This is for the astrometry corrections that occur in stage3. If you want stage2 to be a final product you will need to replace subsky-star.par with subsky-galaxy.par, this is so you are using the correct values for doing the sky-subtraction.

stage5
Calculates redshifts based on cross-correlation of template and object spectra (optional).

stage6
Converts the python object dictionaries to ASCII and HTML format (optional).

If all of the parameters are known, by typing *make* you can run all of the stages. stage3 has the most interactive parts in it but once the scale and position errors are known, it is possible to input the orders non-interactively.

7.2 Input Files

These are the input files needed to run the pipeline, in addition to the files needed by COSMOS:

file	description
HeNeAr.dat	linelist used for **adjust-map**
afea_{imacs/ldss3}.dat	linelist used for **align-mask**
middle.dat	line used to re-center dewoff
parameters.dat	semi-universal parameters file containing COSMOS and CosmosPipeline parameters

Most of the parameters that can change are found in parameters.dat. Most of the changes mentioned in the "Running the Pipelines" section need to be done to the parameters.dat. In some rare cases one might have to go and change COSMOS parameter files. Remember grep is your friend:

```
grep "medbox" *.dat
grep "spline" *.par
```

7.3 Output Files

The various files that the CosmosPipeline outputs and their purposes:

file	stage	description
ccd*xxxx*_1spec_leastsq.ps	3	fits to objects in relation to the mask
ccd*yyyy*.ps	1,3	fits to objects in wavelength and spacial directions
ccd*yyyy*.rms	1,3	rms of the fits to objects in wavelength and spacial directions
index.html	6	HTML file generated for marked redshifts
*mask*_1spec.fits	4	1d spectra in equispec format, bands are in the following order: spectra, noise, flattened-flat, fluxed spectra, fluxed noise
*mask*_1spec_flux.ps	4	fit to the reference spectra for fluxing
*mask*_1spec_leastsq.ps	3	fits to objects in relation to the mask using 1st order in X and 0th order in Y
*mask*_2p_1spec.fits	3	1d spectra in equispec format, bands are in the following order: spectra, noise, flattened-flat
*mask*_big.fits	3	2d spectra in equispec format
*mask*_lig.fits	3	2d lamp in equispec format
*mask*_redshift.out	6	ASCII file generated for marked redshifts
*mask*_sig.fits	3	2d noise in equispec format
mask.fluxlevels	3	flux levels of 2d spectra combined in stage-sumspec
redshiftCC/*mask*_redshiftCC.out	5	ASCII file generated for all redshifts
orders.dat	3	orders of the fits to objects in relation to the mask

8 Running the Pipelines

It is strongly recommended that you first read through the COSMOS Cookbook provided with COSMOS before running the CosmosPipeline.

1. Determine wavelength range.

   If you have a particular wavelength range you wish to use or if you have overlapping slits this can be an important issue. Normally you can look at the object and see how far in the red and the blue the object goes. In non-overlapping case, the wavelength range is dependent on the instruments ability to collect flux in the extrema.

   If you are using a band pass filter you will need to restrict the wavelength range to the filter range otherwise the sky-subtraction will fail.

2. Calibrate mask alignment.

Type

```
make stage1
```

In stage-align-mask you will be prompted with the standard **align-mask** prompts:

```
XXX matches found. sigma =   XXX pixels. Continue?
Average scale =      XXXXXXX   Reset scale?
Average rotation angle = XXXXX   Reset angle?
Average X,Y Offsets =   XXX   XXX pixels  Apply shifts?
```

For **align-mask** you generally want to use a small line list (about 3 lines). You will be prompted for a *searchbox* size. The initial *searchbox* is set to 30. Each iteration you will want to enter a smaller *searchbox* value but larger than the sigma value. If the alignment does not converge to a smaller sigma, then it is possible your lamp lines are being confused. You may need to use a different set of lines. It is possible to use one line to align the dewar. In extreme case you may need to change the dewoff file in order for **align-mask** to find the right lines, the COSMOS program you would need to use in order to change the dewoff would be **adjust-offset**

3. Determine quality of fits from the output of **adjust-map**.

   adjust-map is now setup such that it outputs the fits in wavelength and spacial directions to a postscript file. So by looking at the fits you can better determine the values for *search_height* and *search_width*. You should keep the *search_height* and *search_width* values low in order to not include matches from other slits (this can be tricky in the cases when slits are tightly packed together). You should also not need to go over a 3rd order polynomial for the dispersion direction (*ord_disp*) and a 2nd order polynomial for the spacial direction (*ord_sag*); over-fitting can result in poor the calibrations.

4. Run COSMOS to **extract-2dspec**.

   Type

   ```
   make stage2
   ```

5. Determine *exclude*, *medbox*, *spline2d*, *siglimit*.

   There are two *exclude* and two *medbox* that have to be determined before running stage2. Since we reduce both the *holes* and the *slits*, the values may vary a bit between both.

   It is important to know roughly the smallest slit size (in pixels) for both the *holes* and *slits*.

In order to determine the *exclude* you can use the FWHM as a guide. In poor seeing conditions you may have to balance the *exclude* with the *medbox*. You do not want to go over the slit width when you add the total excluded area, edge area and the *medbox*. It is important to have enough pixels available to for **subsky** to fit the skylines. The purpose of the *exclude* is to not include the object when fitting for the skylines, that way the positions of the skylines will be easier to determine (and resulting in less skyline residuals).

If your slits are less than 35 pixels, it might be a good idea to set the *spline2d* to 'n'. If left on the 2d spline may over fit the data leaving fitting patterns on the data.

We run an *edge* of 1 by default. If you have small slits and need the extra room for fitting the sky you may want to change that value.

e.g.

slit width = 35

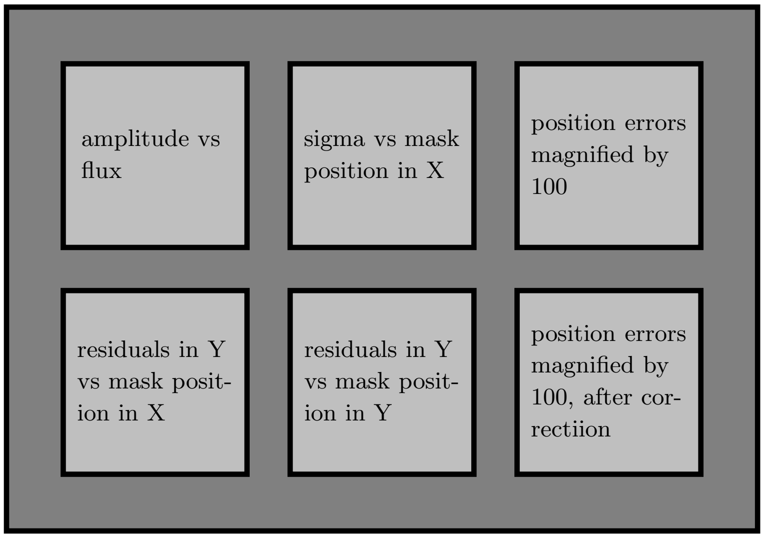
parameter	value	total
exclude	5	11
medbox	15	15
edge	2	4
total		30

6. Determine orders needed for position correction.

   Type

   ```
   make stage3
   ```

   In stage-2pass the initial fit in x-direction of the mask is a 1st order Legendre polynomial and 0th order polynomial in the y-direction. The PGPLOT window will display six graphs.



PGPLOT window for **imacsSpec1d.py**

You need to enter three values that are comma delimited.

```
<(ox,oy,cut)> =
```

ox is the order of the polynomial in the x-direction.

oy is the order of the polynomial in the y-direction.

cut is the amplitude cut for the objects (lower limit).

You want to enter values that minimize the position errors, without over-fitting. When you are finished, type "done".

7. Determine *slitnum* and *ordernum* (optional).

   In order to flux the data, you need a reference object to flux with. Typically a bright A-type star would work fine. It is important to look for a spectra that resembles a black body curve. Any stars with strong metal features will not flux too well (i.e. M-type star). You can flux to galaxies if you do not have any stars available.

   The spectra that you will use to flux with should not have chip gaps that fall on or between the ranges 6850-6960, 7570-7700, 7140-7360 and 9270-9770. These ranges block out the B-band, A-band, water band, and a very red band from the fitting. If a chip gap of the flux spectra falls within these ranges, you will lose that range, it effectively becomes zero.

   Sometimes the features on one star are good but its curve is not black body like. It is possible to set the *slitnum1* to a spectra with good absorption features and set *slitnum2* to a spectra with a good slope. If using different objects, *slitnum1* needs an absorption feature spectra and *slitnum2* needs a good slope spectra.

8. Flux spectra (optional).

   Type

   `make stage4`

9. Determine redshift objects, ranges, and masks (optional).

   Type

   `make stage-display`

   This marks the beginning of the redshift stages. This tool enables you to determine which objects should run the redshift cross-correlation, what areas should be masked out, direct redshift guesses, etc. This stage can be run as many times as needed. The goal of this stage is to cull out bad data for the cross-correlation. For more information on its use, see **imacsDisplay.py** documentation.

10. Run redshift cross-correlations (optional).

    Type

    `make stage5`

11. Determine quality of redshifts (optional).

    Type

    `make stage-redshift-display`

    This tool is used for after the completion of the cross-correlation. Its purpose to investigate the results, determine what needs to be run again and what is finished. For more information on its use, see **aRedshiftDisplay.py** documentation.

12. Convert database to ASCII and HTML tables (optional).

    Type

    `make stage6`

9 Diagnostics

9.1 Verifying the Wavelength Calibration

There are a couple of way to determine the wavelength calibration. **adjust-map** outputs a postscript file and a ASCII file. The postscript file shows the data and the fits, and the ASCII file shows the standard deviation and the rms of each fit.

Another way to check is by visual inspection of the lamps and mapfile.

```
make stage-display-lamp-0
```

10 Problems

10.1 align-mask

One of the most common problems with **align-mask** is when features are outside the *searchbox* region. In this case, **align-mask** will never converge. You will need to manually adjust the dewoff file by using the routine **adjust-offsets**.

Other common problems are using too large of a line list for **align-mask**. You should in general use no more than 3-4 isolated lines. If the lines are not isolated enough there may be confusion.

It is also important to make sure the lines used are actually seen in your lamps, sometimes features are not strong enough for **align-mask** to find them.

An accurate solution is not needed for **align-mask**, all it is doing is roughly finding the dewar orientation, the routine **adjust-map** will find the wavelength and spacial solutions.

10.2 Sky Subtraction Fails

There are several reasons why sky-subtraction can fail:

1. *exclude* is too large
2. *medbox* is too large
3. *medbox* is too small
4. 2d spline over fits the data
5. **adjust-map** failed to find lamp lines
6. 2nd order contamination

10.3 Cosmic ray clipping

Cosmic ray clipping occurs in the combining of the 2d spectra (stage-sumspec). To change the clipping for the combining you will want to change *clipping* in parameters.dat.

11 To Do

1. Add object keyword option to **CosmosPipeline.py** for sciences, flats and lamps.
2. Add ability to fork processes for multiple core, multiple processor.
3. Add Twisted client for queue based processing on multiple computers.
4. Re-write **CosmosPipeline.py**, using classes.
5. Finish documentation for interactive scripts.

12 Makefile

There are two types of Makefiles created by the pipeline; top level Makefile and individual target Makefiles.

12.1 Top level Makefile

The top level Makefile allows you better control your reductions on multiple targets. Some of the features includes making stages of all the interactive parts together. This is beneficial for stages such as stage1 and the beginning of stage3.

stage	description
all	make all targets, sequentially.
remake	re-generates the pipelines
rerun	re-generates the pipelines; skipping stage-align-mask, uses orders.dat in stage-2pass (non-interactive mode)
regenerate	re-generates the pipelines (overwrites parameter.dat, dewoff, obsdef, afea*.dat, middle.dat and HeNeAr.dat files)
stage1	makes all targets stage1
stage2	makes all targets stage2
stage3	makes all targets stage3
stage4	makes all targets stage4
stage5	makes all targets stage5
stage6	makes all targets stage6
display-2p_*target*	runs imacsDisplay.py on 1d and 2d extracted *target*
display_*target*	runs imacsDisplay.py on 1d and 2d extracted, fluxed *target*
targ_*target*	makes *target*
clean	cleans all targets
clean_*target*	cleans *target*
par-update	updates the COSMOS *.par files with parameters.dat
par_*target*	updates the COSMOS *.par files with parameters.dat for *target*

12.2 Individual target Makefiles

The target Makefiles can be broken up into three categories: main stages, tool stages and clean stages.

12.2.1 Main stages

The main stages are for the data reduction.

- **stage1**

  stage-par creates COSMOS *.par files, sets instrument specific parameters, updates COSMOS *.par files with parameters.dat

  stage-stitch (LDSS3 only) combines all LDSS3 frames

  stage-align-mask creates corrected dewoffs on all lamp frames

  stage-map-spectra creates mapfiles from for all lamp frames

  stage-adjust-map modifies mapfiles by fitting spacial and dispersion directions for all lamp frames

- **stage2**

  stage-sflats constructs spectroscopic flat-field frames

  stage-biasflat flat-field correction on science frames

  stage-subsky sky-subtraction of science frames

  stage-extract-2dspec extracts 2d spectra from science frames

- **stage3**

  stage-par-2 updates COSMOS *.par files with parameters.dat

  stage-2pass corrects for mask and astrometry errors, generates new SMF files

  stage-map-spectra-2 creates mapfiles from new SMF files for all lamp frames

  stage-adjust-map-2 modifies mapfiles by fitting spacial and dispersion directions for all lamp frames

  stage-subsky-2 sky-subtraction of both holes and slits with different parameters

  stage-extract-2dspec-2 extracts 2d spectra from science frames

  stage-sumspec combines 2d spectra

  stage-flat-flat creates blaze from flattening a flat-field

  stage-extract-1dspec extracts 1d spectra

- **stage4**

  stage-flux fluxes 1d spectra with blaze

- **stage5**

  stage-redshift runs redshift cross-correlation code on spectra (stage-display needs to be run first)

- **stage6**

  stage-redshift-output outputs redshift results to ASCII and HTML (stage-redshift-display needs to be run first)

12.2.2 Tool stages

The tool stages are designed help with diagnoistics, redshifts and global parameters.

stage	description
stage-display-lamp-*num*	displays lamp *num* with vfits
stage-display-2p	displays extracted 1D and 2D spectra with imacsDisplay.py
stage-display	displays extracted 1D and 2D fluxed spectra with imacsDisplay.py
stage-redshift-display	displays redshift results from stage-redshift with aRedshiftDisplay.py
par-update	updates COSMOS *.par files with parameters.dat

12.2.3 Clean stages

The clean stages are designed to remove intermediate files with various levels of cleaning.

stage	description
clean	remove intermediate files
clean-stage*num*	removes stage files for stage*num*
very-clean[4]	removes all intermediate files; ccd*_2spec.fits, ccd*.dewoff*, *.map, *.ps, *.par*, *.xy, ccd*.obsdef, ccd*.SMF and all stage files

Use caution when using stage **very-clean**, only use this stage when you want to run the pipeline from the beginning!

12.3 Scripts

script	description
2specDisplay.py	Display tool for displaying COSMOS *2spec.fits files
align-mask.py	Better way of handling COSMOS **align-mask**
aRedshiftDisplay.py	Display tool for displaying **aRedshift.py** Cross-correlation output
aRedshiftOutput.py	Generates ASCII and HTML redshift reports
aRedshift.py	Cross-correlation tool for find redshifts with Sloan templates
Cosmosdb.py	Creates a database file containing information from the headers of fits files from a directory submitted to the command-line for IMACS and LDSS3 data
CosmosdbFix.py	Database keyword header fixing script
CosmosMarkLines.py	A revised version of Dan's **imacsMarkLines** that includes LDSS3 data
Cosmospar.py	Parses through COSMOS original parameter files and views, edits and creates the parameter files
CosmosPipeline.py	Generates the Makefiles from the database file (created by Cosmosdb.py) and sets up links and files necessary for starting a reduction pipeline.
fitsplot.py	Plotting program spawned from **2specDisplay.py**, similar to pfits
imacsDisplay.py	Display tool for setting up for running aRedshift.py
imacsSlitHoleCombine.py	Re-combines the *holes* and *slits* from the sky subtracted frames
imacsSlitHole.py	Separates *holes* and *slits* in the map files so *holes* and *slits* can be run through **subsky** with different parameters
imacsSpec1d.py	Multi-functioned script that generates new SMF files from fitting the positions on the objects from the extracted spectra from the first pass. It also creates 1D spectra from the extracted spectra of the second pass
imacsSpec2d.py	Generates equispec *big.fits from COSMOS *2spec.fits
imacsSpecFlux.py	Fluxes 1D spectra (created by **imacsSpec1d.py**) with a reference spectra and each of the flattened flat-fields
imacsSumSpec.py	A replacement for COSMOS **sumspec** program used only in the second pass. It was necessary to use this program to deal with situations where uneven arrays of objects were created and could not be summed together with COSMOS code

script	description
parUpdate.py	Updates parameters from parameters.dat to COSMOS parameter files
xy2lin.py	Converts coordinates from COSMOS **spectral-map** to Dan's lin type file for vfits and dfits

- all
  - stage1
    - stage-par
    - stage-stitch (LDSS3 only)
    - stage-align-mask
    - stage-map-spectra
    - stage-adjust-map
  - stage2
    - stage-sflats
    - stage-biasflat
    - stage-subsky
    - stage-extract-2dspec
  - stage3
    - stage-par-2
    - stage-2pass
    - stage-map-spectra-2
    - stage-adjust-map-2
    - stage-subsky-2
    - stage-extract-2dspec-2
    - stage-sumspec
    - stage-flat-flat
    - stage-extract-1dspec
  - stage4
    - stage-flux
  - stage5
    - stage-redshift
  - stage6
    - stage-redshift-output