

User guide for WEAVEify (v1)

WEAVE-QSO Collaboration



Contents

1	Overview of WEAVEIFY	1
1.1	Purpose of the code	1
1.2	Installation	1
1.3	Introduction	2
2	Quick Start	2
2.1	H I Lyman series forest	2
2.2	Adding DLA to forest	4
2.3	Adding Metals to forest	4
2.4	Continuum Placement	5
2.5	Noise Addition	5
2.6	Optical depth calculations	6
3	Automating WEAVEIFY for large number of spectra	7
3.1	Single core WEAVEIFY automation	7
3.2	Multi core WEAVEIFY automation using MPI	8
4	Reading WEAVEIFY output	9
4.1	Output file names	9
4.2	Data format	9
4.3	Header format	13
4.4	Script to read WEAVEIFY output	13

5	Modifying WEAVEIFY parameters	14
5.1	DLA parameters	14
5.2	WEAVE resolution mode	15
5.3	Instrumental broadening parameters	16
5.4	Wavelength sampling parameters	17
5.5	Telescope throughput parameters	17
5.6	Metal properties	19
5.6.1	Method 1 : Cloudy Interpolation Tables	19
5.7	Continuum placement parameters	23
5.8	Noise variation parameter	24
6	Detail description of free parameters in WEAVEIFY	28
6.1	Cosmological mocks	28
6.2	QSO parameters	28
6.3	DLA parameters	30
6.4	Metal parameters	30
6.5	Wavelength sampling parameters	30
6.6	Continuum placement parameters	31
6.7	Noise parameters	31
6.8	Cosmological parameters	31
6.9	MPI job submission parameters	32
6.10	Input / Output file saving parameters	32
6.11	Miscellaneous parameters	33
7	Acknowledgement and Citations	34
8	Frequently Asked Questions (FAQs)	34

1 Overview of WEAVEIFY

1.1 Purpose of the code

WEAVEIFY is developed to forward-model the QSO absorption spectra in simulations that look similar (in a statistical way) to those that will be observed by the WEAVE instrument. WEAVEIFY can efficiently generate large number of ($\sim 10^4 - 10^6$) mock QSO absorption spectra that include modeling of H I Lyman series forest, metal and DLA absorption systems. The code also accounts for realistic noise, continuum placement and instrumental resolution effects of WEAVE instrument. Even though WEAVEIFY is written to model spectra for WEAVE survey, the code is modular and flexible enough to mock the spectra for variety of instruments by changing parameters. We refer the reader to the paper describing WEAVEIFY for details. The code is publicly available at <https://bitbucket.org/prakashgncra/weaveify>

1.2 Installation

- User can download the code using following command. The size of the zip file is ~ 50 Mb.

```
git clone https://bitbucket.org/prakashgncra/weaveify.git
```

- Extract the zip file at your location of choice. The typical file structure would look like shown in Fig. 1

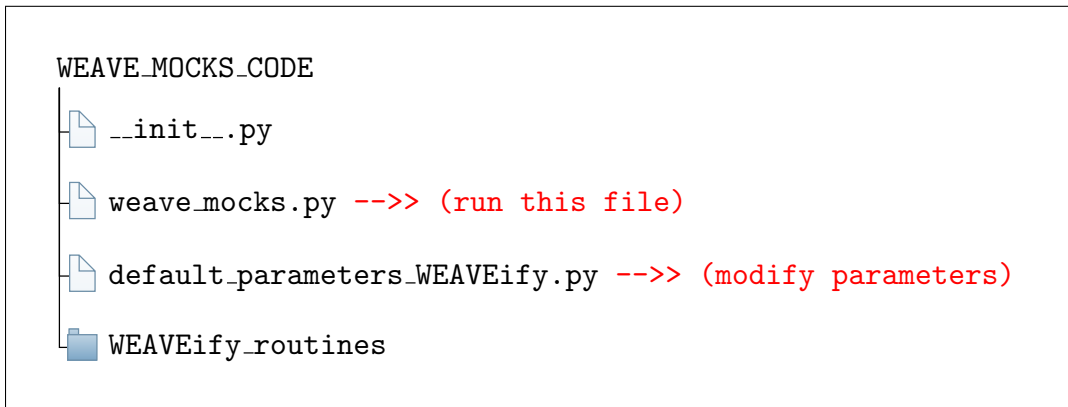


Figure 1: Basic structure of the code directory. Users need to run the file `weave_mock.py`. All the parameters of the code are specified in the file `default_parameters_WEAVEify.py`

- WEAVEIFY is written in python and is compatible with both *python2* and *python3*
- Following packages are required
 - *numpy*
 - *scipy*
 - *matplotlib*
 - *astropy* (Optional) package is needed if the desired output format is **fits**.
 - *h5py* (Optional) package is required if the desired output format is **hdf5**.
 - Either *astropy* or *h5py* should be installed.
 - *mpi4py* package is required if spectra needs to be generated on multi-core machines in parallel. *mpi4py* is not needed if users want to run the code for few spectra on single core.

- Users can refer to §8 if they have any difficulty in installing these python packages.
- All the parameters needed to mock the spectra are given in the file `default_parameters_WEAVEify.py`. User may modify this parameter file later for their custom needs (see §3 for details). We recommend users to make a copy of this file before modifying it.

1.3 Introduction

- To execute the code, go to directory `WEAVE MOCKS CODE` and run the python script `weave_mock.py` (see Fig. 1)
- WEAVEIFY can be run either with command like interface or by modifying the parameter file `weave_mock.py`.

```
python weave_mock.py <argument-list> # Command like Interface
```

- We recommend to use the command like interface to test the code behaviour for small number of mocks (~ 100 spectra).
- While for generating large mock spectra, we recommend to modify `weave_mock.py` file.

```
python weave_mock.py # Script like Interface for Automation
```

- We provide few examples of `weave_mock.py` file to illustrate how automation can be achieved with script approach.
- We refer users to §3 on how to use Message Passing Interface (MPI) with script.

2 Quick Start

- To begin, we provide few test cosmological mock fields with the code. The test examples are kept in directory `WEAVE` directory (see Fig. 2)
- In this directory there are 2 subdirectories `INPUT` and `OUTPUT`. `INPUT` directory contains cosmological mock fields i.e., overdensity (Δ), temperature (T), line of sight velocity (v) and neutral hydrogen fields (f_{HI}) from Sherwood simulation suite.
- For the following tutorial, the code output, e.g., optical depths, mock spectrum and plots, would be stored in `OUTPUT/SERIAL_SINGLE_SPECTRUM/` directory.
- After each successful execution of code, user can view mock spectrum from `Quick_Plot_Mock_Spectra-0.pdf`

2.1 H I Lyman series forest

- To begin, we first generate mock spectra for H I ($\text{Ly}\alpha + \text{Ly}\beta$) forest only. That is we exclude metals and DLA option. To do this, we run following command in the terminal.

```
python weave_mock.py flag_absorber=HI
```

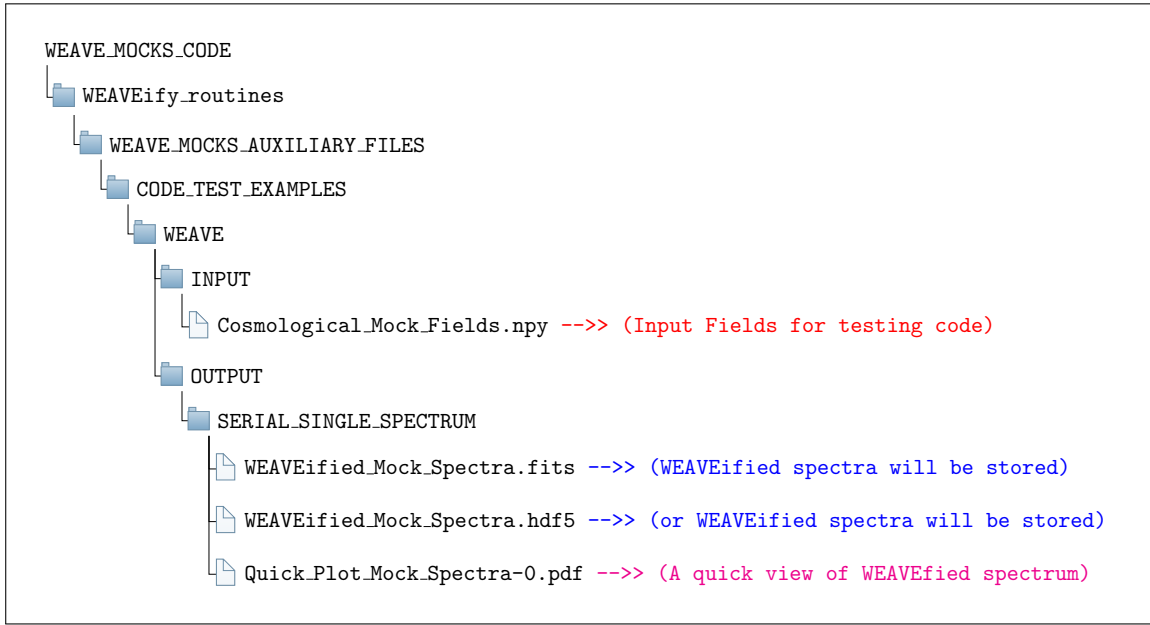


Figure 2: Location of input and output directory for test examples of the code.

- Note that sequence of argument is “not” important. However it is important to not leave empty spaces before and after “=” equal to sign while specifying arguments. We caution that if you copy-paste above command in terminal, there may be extra spaces between “=” sign because of clipboard copy format and code will not work.
- The code will start to calculate H I flux from H I optical depths. The flag `absorber=HI` tells code to calculate normalize flux from only H I optical depth.
- Once the code is executed successfully, we can go to directory `OUTPUT/SERIAL_SINGLE_SPECTRUM`

The code has generated several files with fits extension and a pdf file showing the mock spectrum. The main output mock spectrum is in file `WEAVEified_Mock_Spectra.fits`. We can view part of the mock spectrum by opening file `Quick_Plot_Mock_Spectra-0.pdf`

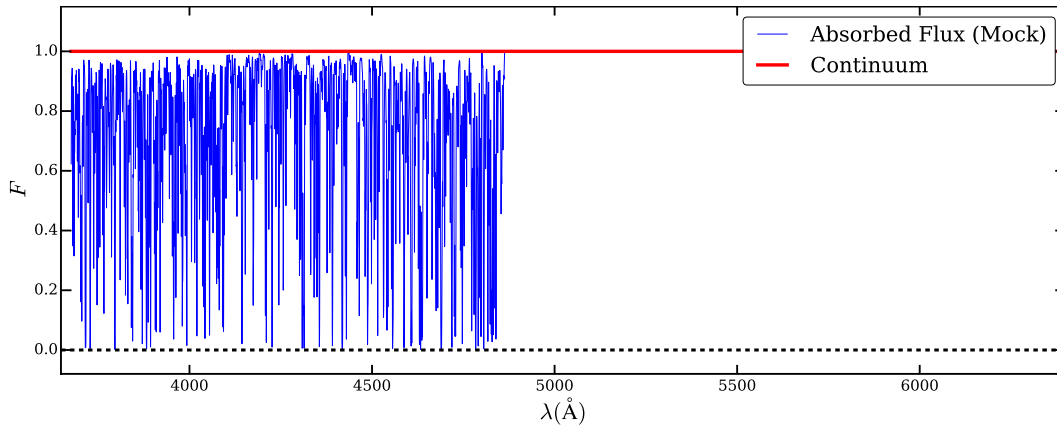


Figure 3: Test example of first mock spectrum H I ($\text{Ly}\alpha + \text{Ly}\beta$) forest only

2.2 Adding DLA to forest

- We can now add DLA to the mock spectra. To do this we can run command,

```
python weave_mocks.py flag_absorber=HI_DLA
```

Output:Quick_Plot_Mock_Spectra-0.pdf.

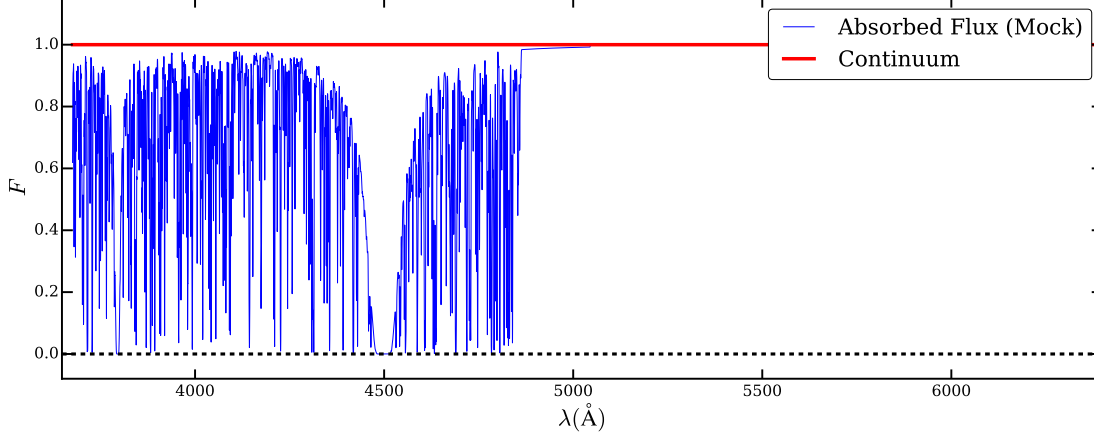


Figure 4: Example of mock spectrum with H I forest + DLA. Default parameter set for DLA in code are $\log N_{\text{HI}} = 21.0$, $b = 10 \text{ km s}^{-1}$ and $z_{\text{DLA}} = 2.7$.

2.3 Adding Metals to forest

- We can add metals to the mock spectra using cloudy interpolation tables (default).

```
python weave_mocks.py flag_absorber=HI_metal_DLA
```

Output:Quick_Plot_Mock_Spectra-0.pdf

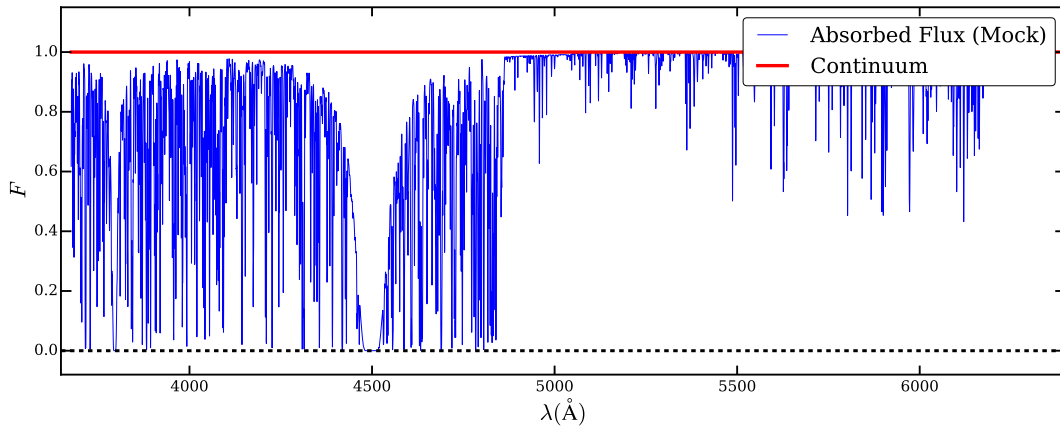


Figure 5: Example of mock spectrum H I (Ly α + Ly β) forest + Metals + DLA. Default IGM metallicity is $\log (Z_{\text{IGM}}/Z_{\odot}) = -1.5$

2.4 Continuum Placement

- We can now add continuum by specifying `flag_cont_method=PCA`. See §5.7 for more details on how continuum can be varied in WEAVEIFY.

```
python weave_mocks.py  flag_absorber=HI_metal_DLA
flag_cont_method=PCA
```

Output: Quick_Plot_Mock_Spectra-0.pdf

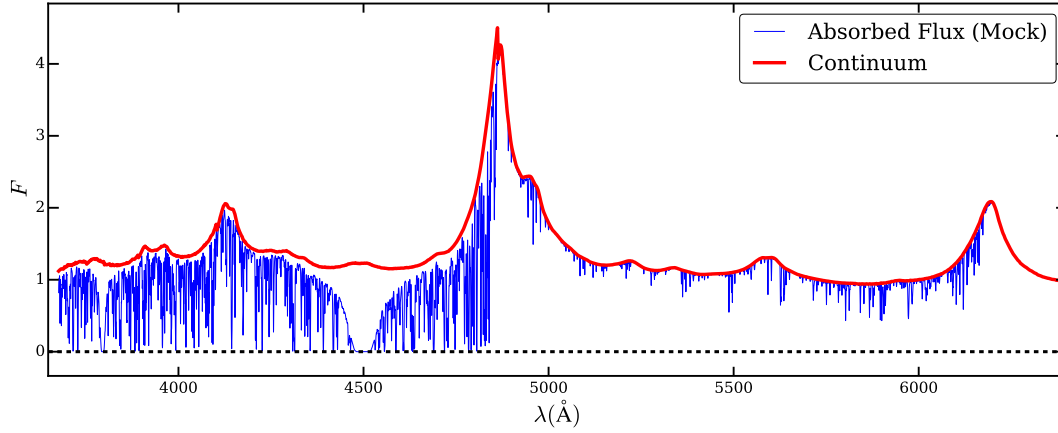


Figure 6: Mock spectrum with PCA generated continuum

2.5 Noise Addition

- We can now add realistic noise to the spectra using variable `flag_noise=OpR3`. Parameters describing the noise properties in spectra are explained in detail in §5.8

```
python weave_mocks.py  flag_absorber=HI_metal_DLA
flag_cont_method=PCA flag_noise=OpR3
```

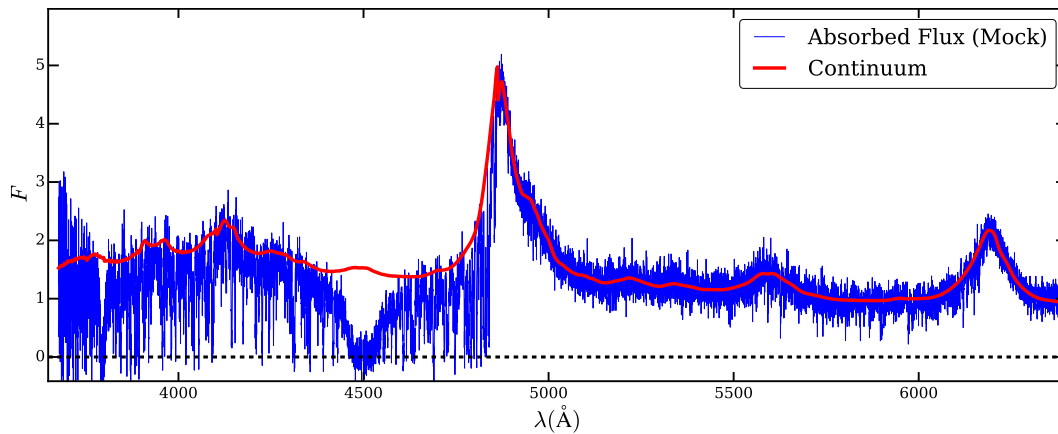


Figure 7: Example of mock spectrum contaminated by realistic WEAVE like noise.

2.6 Optical depth calculations

- In previous section, we generated the mock spectrum from optical depth data already calculated from cosmological fields i.e., Δ , T , v and f_{HI} .
- Often, one need to calculate the optical depths for various transitions (H I Ly α , H I Ly β or metal transitions) from cosmological fields. WEAVEIFY can calculate the optical depths for all specified transitions (i.e., H I Ly-series or metal transitions).
- In this section, we show how to calculate optical depths for various absorbers. To calculate optical depths for various absorbers we need to set `flag_tau` to the absorber names similar to `flag_absorber`. For example, `flag_tau=HI` would calculate optical depth for all the H I transitions specified in the code.
- `flag_tau=HI_DLA` would calculate optical depth of H I forest and DLA. `flag_tau=HI_metal_DLA` would calculate optical depth of H I forest, metal transitions and DLA.
- Note that `flag_tau` calculates optical depths independently of `flag_absorber`. This means if you specify `flag_tau=HI_metal` but specified `flag_absorber=HI`, then code would calculate optical depth for HI forest and metal but will not include metal in final mock spectrum because `flag_absorber` tells code to add contribution of only H I forest to the total flux.
- Separating `flag_tau` from `flag_absorber` is important because this provides flexibility to users. Sometime users may want to change only DLA properties keeping H I forest same. Then this allows user to calculate H I forest optical depths only once and then only calculate DLA optical depth for various parameters.
- To generate optical depth for all transitions, we can run following command

```
python weave_mocks.py flag_absorber=HI_metal_DLA
flag_tau=HI_metal_DLA flag_cont_method=PCA
```

- The code will start calculating optical depth for various species and transitions. This may take some time to compute optical depths. At the end output would be similar to Fig. 5.

Output:Quick_Plot_Mock_Spectra-0.pdf

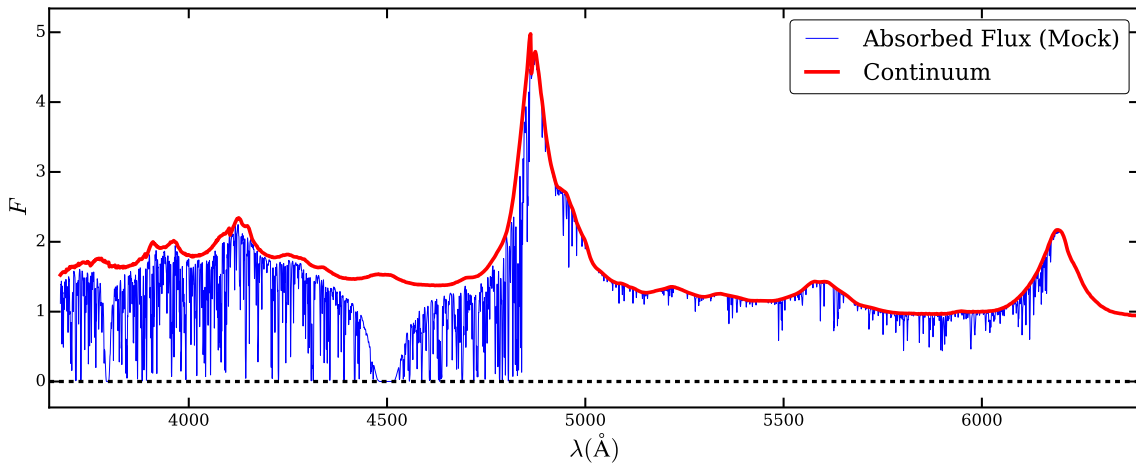


Figure 8: Example of mock spectrum after using optical depth generation option.

3 Automating WEAVEIFY for large number of spectra

The examples provided in §2 are suitable to produce few 10's of mock spectra. Often users would like to use WEAVEIFY to generate large number of ($\sim 10^4 - 10^6$) mock spectra. In this section we describe how WEAVEIFY can be used in script mode to generate spectra for WEAVE like survey. We provide two sample scripts to demonstrate how WEAVEIFY can easily be used for generating large number of mock spectra. The file `weave_mocks_serial_multiple_spectra.py` shows an example of automating the WEAVEIFY on single core while file `weave_mocks_parallel_mpi_multiple_spectra.py` shows an example of using WEAVEIFY on multi-core machines.

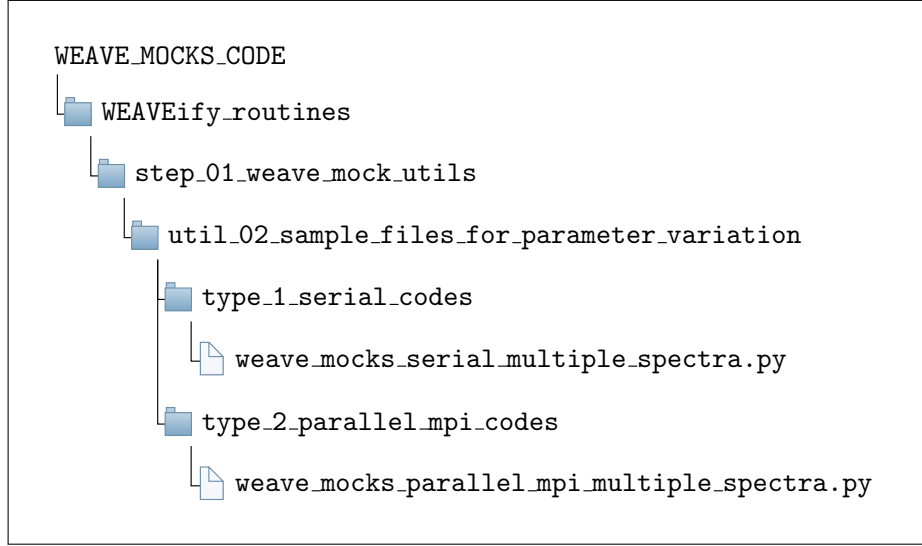


Figure 9: Location of sample codes to automate WEAVEIFY for large number of spectra. The file `weave_mocks_serial_multiple_spectra.py` shows an example of using single core machine while file `weave_mocks_parallel_mpi_multiple_spectra.py` illustrates use of WEAVEIFY on multi-core machine.

3.1 Single core WEAVEIFY automation

In this section, we describe the steps to automate WEAVEIFY for multiple spectra.

1. First we identify the parameter that needs to be varied. Identify the variable name corresponding to this parameter. Refer §6 or file `default_parameters_WEAVEify.py` to see list of all variable names. For example, if we want to vary magnitude of sources, then the corresponding variable name is `m_source`.
2. WEAVEIFY code uses one single main function `WEAVEify_function_calls_main`. This function has been imported in all `weave_mocks.py` files. We need to supply the variable name users want to change to the function `WEAVEify_function_calls_main` as a keyword argument (`**kwargs`). *In this step we need to set the variable `flag_mpi='n'` to tell the code that it is a single core script.*
3. All other parameters would take default value as set in file `default_parameters_WEAVEify.py`.
4. In the final step, we need to call this function multiple times in `for` loop in which the parameter is changed. In this way WEAVEIFY can generate multiple number of spectra.

5. We provide a more detailed example of automating WEAVEIFY on single core machine in file `weave_mock_serial_multiple_spectra.py` (see Fig. 9).
6. Users can copy the file `weave_mock_serial_multiple_spectra.py` to `WEAVE MOCKS_CODE` directory and run following command to test how WEAVEIFY works

```
python weave_mock_serial_multiple_spectra.py
```

Note that the `spec_id` is one of the most important variable that should be varied within for loop. This variable is used in WEAVEIFY to create unique hdf5 group or fits extension name in the final output files.

3.2 Multi core WEAVEIFY automation using MPI

Now we use the same example given above, and show how to automate WEAVEIFY on multi-core machines using Message Passing Interface (MPI). The basic idea behind automation is same as that of single-core machine except that the `for` loop is executed simultaneously on different cores. This essentially reduces the number of iterations that the `for` loop performs. A prerequisite to use multi-core functionality is that the `mpi4py` is installed. The steps to automate WEAVEIFY using parallel MPI libraries is as follows

1. *First we modify the `weave_mock.py` file for single core machine as explained in previous section. We recommend to test if the code is working on single core. The code that runs on single-core machine can simply be run on multi-core machine. For this we just need to change variable `flag_mpi='y'`*
2. We provide a more detailed example of automating WEAVEIFY on multi core machine in file `weave_mock_parallel_mpi_multiple_spectra.py` (see Fig. 9).
3. We recommend reader to compare the content of file `weave_mock_parallel_mpi_multiple_spectra.py` with `weave_mock_serial_multiple_spectra.py`. All the variables are identical except `flag_mpi` and `out_path`. We changed `out_path` to save the output at different location. However, users do not need to change this.
4. Users can copy the file `weave_mock_parallel_mpi_multiple_spectra.py` to `WEAVE MOCKS_CODE` directory and run the program. The syntax for running MPI codes is as follows.

```
mpirun -np <number-of-processors> python <file-name.py>
```

Following command would work with the examples provided.

```
mpirun -np 5 python weave_mock_parallel_mpi_multiple_spectra.py
```

Note that syntax to submit the MPI job may differ on different machines especially if the HPC uses Slurm resource management systems. HPC system admin team usually provides instruction on how to submit the slurm scripts. Users do not need to worry about distributing the equal number of jobs on different processors. Even if the number of spectra to mock are not divisible by number of processors, WEAVEIFY code would try to distribute them as evenly as possible.

4 Reading WEAVEIFY output

In this section we describe the basic format of WEAVEIFY output files. For fits format, the data is arranged in extension name and BinaryHDU tables. The data of each mock spectrum is stored under different extension name. All data arrays are stored with single precision accuracy to optimize the file sizes. A header is attached to each extension separately. For hdf5 format, the data is arranged in groups, dataset and Header blocks. Similar to fits format, the data of each mock spectrum is stored under different hdf5 group. A header block is explicitly added to each group. *The output data structure and naming convention is same for fits and hdf5 file format.*

4.1 Output file names

By default, the WEAVEIFY outputs (i.e., optical depths and fluxes) are saved in file starting with prefix ‘WEAVEified_Mock_Spectra’. The variables `out_file_flux_base` and `out_file_tau_base` controls the prefix of the output files. Both variables currently set to `WEAVEified_Mock_Spectra`. As a result optical depths and fluxes are saved in the same file. If users want to change the prefix of the output files, they need to change these two variables. It is also possible to store the optical depths and fluxes in two different files. In such case, users need to choose different values of the two variables. When WEAVEIFY is run in parallel, the outputs from different processors (ranks) are stored in different files and a suffix ‘Rank_<processor-number>’ is attached to each file. This way of storing the file optimizes the performance of resources while maintaining the output file size. The extension of output files is controlled by variable `flag_output_format`. WEAVEIFY supports two file format fits or hdf5 which requires different python modules to be installed (see 1.2 for details).

```
For flag_mpi = 'n', flag_output_format = 'fits',  
Output File Name: WEAVEified_Mock_Spectra.fits  
  
For flag_mpi = 'y', rank = 5, flag_output_format = 'hdf5',  
Output File Name: WEAVEified_Mock_Spectra_Rank_5.hdf5
```

4.2 Data format

Fig. 10 shows the basic data structure of typical WEAVEIFY output file (fits or hdf5). The WEAVEIFY output file contains following fits extension (hdf5 group) names, BinaryHDU tables (hdf5 dataset) and headers.

- **COMMON_FIELDS:** (Fits Extension or HDF5 Group, Block, String)
This extension / group contains the arrays common to all the spectra. For example, for a given resolution mode the wavelength array of a flux would remain same for all the spectra. Hence to optimize the output file size it is important to save the array only once in file.
- **Wavelength_For_Flux:** (BinaryHDU Table or HDF5 Dataset, Array, Float32)
This table/dataset contains wavelength array for the fluxes. The shape of this array could be equal to or smaller than the `Wavelength_For_Optical_Depth` table/dataset.
- **Wavelength_For_Optical_Depth:** (BinaryHDU Table or HDF5 Dataset, Array, Float32)
This table/dataset contains wavelength array for the optical depths. The shape of this array could be equal to or greater than the `Wavelength_For_Flux` table/dataset.

```

- COMMON_FIELDS -->> (Fits Extension or HDF5 Group)
  - Wavelength_For_Flux -->> (BinaryHDU Table or HDF5 Dataset)
  - Wavelength_For_Optical_Depth -->> (BinaryHDU Table or HDF5 Dataset)
- MOCK_SPECTRA-0 -->> (Data for mock spectra with spec_id=0)
  - Header -->> (All the variables used for this spectrum)
  - Flux -->> (WEAVEified mock flux, Final output)
  - Flux_Without_Noise -->> (Flux without noise)
  - Continuum_Template -->> (Continuum template used in the spectrum)
  - SNR_per_pixel -->> (Array of Signal-to-noise ratio per pixel)
  - Optical_Depth_HI -->> (Total Optical Depth for HI Ly-series Forest)
  - Optical_Depth_DLA -->> (Optional, Total Optical Depth for HI Ly-series DLA)
  - Optical_Depth_METAL -->> (Optional, Total Optical Depth for all metals)
- MOCK_SPECTRA-1 -->> (Similar to MOCK_SPECTRA-0 except for spec_id=1)
- MOCK_SPECTRA-2 -->> (Similar to MOCK_SPECTRA-0 except for spec_id=2)
-   :   :   :   :   :
- MOCK_SPECTRA-N -->> (Similar to MOCK_SPECTRA-0 except for spec_id=N)

```

Figure 10: **The figure shows basic structure of WEAVEIFY output files.** Data is arranged in extension name (for fits format) and group name (hdf5 format). Each extension / group name contains its own header. The header stores all the input parameters supplied to WEAVEIFY. Note that header variables would be written for each mock spectrum separately. Data of optical depth for DLA and metals may not be stored if sightline does not contain DLA or metals as specified by users. Note that dimension of optical depth array and flux array need not be same. Hence separate wavelength array is provided for optical depth and flux. The output data structure is same for fits and hdf5 format. **Variable spec_id is used to create a unique extension / group name that distinguishes data of one mock spectrum from another.** We provide a script `read_weaveify_output_all.format.py` to read the WEAVEIFY output files.

- **MOCK_SPECTRA- $\langle spec_id \rangle$:** (Fits Extension or HDF5 Group, Block, String)
This extension / group contains the output flux and optical depth arrays for spectrum with unique identification variable `spec_id`. The mock spectra are distinguished from each other by `spec_id`. Hence it is important to vary `spec_id` for every spectrum.
- **Header:** (FITS or HDF5 Header, Dictionary, Mixed data type)
The header contains all the variables supplied to WEAVEIFY for generating the spectrum. Since the parameters varies from one spectrum to another, Header is specified for every mock spectrum separately to reflect changes. More details about Header format can found in §4.3.
- **Flux:** (BinaryHDU Table or HDF5 Dataset, Array, Float32)
This table/dataset contains final WEAVEIFY flux. The Flux table/dataset includes all the observational effects (such as continuum, noise, resolution mode etc.) that are specified by the user. The shape of this array is same as that of `Wavelength_For_Flux` array.
- **Flux_Without_Noise:** (BinaryHDU Table or HDF5 Dataset, Array, Float32)
This table/dataset contains noiseless flux. The Flux table/dataset includes all the observational effects (except noise) that are specified by the user. Subtraction of `Flux` table/dataset from `Flux_Without_Noise` gives noise estimate on each pixel. This field would be important for users if they want to test the effect of noise levels on properties of QSO absorption spectra. The shape of this array is same as that of `Wavelength_For_Flux` array.
- **Continuum_Template:** (BinaryHDU Table or HDF5 Dataset, Array, Float32)
This table/dataset contains continuum template used to generate the mock spectrum. The continua are generated randomly for each spectrum, unless random seed is specified. This field would be important for users if they want to test the effect of continuum estimate on properties of QSO absorption spectra. The shape of this array is same as that of `Wavelength_For_Flux` array.
- **SNR_per_pixel:** (BinaryHDU Table or HDF5 Dataset, Array, Float32)
This table/dataset contains Signal-to-Noise ratio used to add the noise the spectra. Since the sensitivity of spectra often changes with wavelength, the SNR per pixel changes with wavelength. The SNR per pixel array is used to generate a random realization of noise that is added to the final spectrum. The shape of this array is same as that of `Wavelength_For_Flux` array.
- **Optical_Depth_HI:** (BinaryHDU Table or HDF5 Dataset, Array, Float32)
This table/dataset contains total optical depths for H I Lyman series forest (as specified by variable `HI_trans_list`). This does not include the contribution of DLA. The shape of this array is same as that of `Wavelength_For_Optical_Depth` array.
- **Optical_Depth_DLA:** (BinaryHDU Table or HDF5 Dataset, Array, Float32)
This table/dataset contains total optical depths for H I DLA Lyman series (as specified by variable `HI_trans_list`). The properties of DLA set by variables described in §5.1. This does not include the contribution of H I Lyman series forest. Note that `Optical_Depth_DLA` extension/group may not exist if the DLA properties are not specified. In such cases, WEAVEIFY would assume that DLA is not present along the sightline. The shape of this array is same as that of `Wavelength_For_Optical_Depth` array.

Table 1: Header keywords / cards in WEAVEIFY output

Header Keyword	WEAVEIFY Variable	Description
OBJMAG	<code>m_source</code>	Source magnitude
OBJWBAND	<code>wave_band_source</code>	Wavelength filter of source magnitude
OBJTYPE	<code>source_type</code>	Type of source (Point or Extended)
INSTR	<code>instrument</code>	Instrument Name
RESMODE	<code>resolution_mode</code>	Resolution mode for WEAVE survey
SPECTYPE	<code>spectra_type</code>	Type of Spectra
DIATEL	<code>dia_telescope</code>	Diameter of telescope in meters
PIXSIZE	<code>ccd_pix_size_micron</code>	Size of CCD pixel in microns
FIBREDIA	<code>dia_fibre_arcsec</code>	Diameter of fibre in arcsec
CRVAL1	<code>CRVAL1</code>	Start of wavelength array in angstroms
CDEL1	<code>CDEL1</code>	Wavelength spacing in angstroms
NPIXEL	<code>N_pixel_obs</code>	Number of pixels in wavelength/flux array
WSCALE	<code>lambda_scale</code>	Wavelength spacing scale (LIN or LOG)
CONTINUM	<code>flag_cont_method</code>	Continuum generation method
LSFTYPE	<code>lsf_type</code>	Functional form of Line Spread Function
NOISTYPE	<code>flag_noise</code>	Type of noise added to spectra
SKYMAG	<code>m_sky</code>	Sky magnitude
SKYWBAND	<code>wave_band_sky</code>	Wavelength Filter of sky magnitude
EXPTIME	<code>t_exp</code>	Exposure time in seconds
SEEING	<code>fwhm_seeing</code>	Atmospheric Seeing
BEAMSIZE	<code>beamsize</code>	Beam size
DARKCURR	<code>dark_current</code>	Amount of Dark current
SEEPFLE	<code>seeing_profile</code>	Functional form of Seeing Profile
ROFFSET	<code>r_offset</code>	Offset of source from center of fibre in arcsec
READNOIS	<code>read_noise</code>	Amount of read noise
AIRMASS	<code>airmass</code>	Atmospheric airmass
BETAMOFT	<code>beta_moffat</code>	Power law index of MOFFAT Seeing profile
THRTYPE	<code>throughput_type</code>	Type of throughput
FLUXNORM	<code>flux_norm_(erg/s/cm^2/A)</code>	Normalization factor for flux
ZEM	<code>z_em</code>	Emission redshift of QSO
SIMNAME	<code>sim_name</code>	Name of the simulation
BOXSIZE	<code>L_box</code>	Size of simulation box in cMpc / h
NGRIDSIM	<code>N_Grid</code>	Number of grids in simulation sightline
SPECID	<code>spec_id</code>	Spectrum Identification number or String
HILIST	<code>HI_trans_list</code>	HI transition list to calculate OD and Flux
TAUFIELD	<code>flag_tau</code>	Optical Depth to be calculated for the specie
TAUSCALE	<code>flag_tau_each_trans</code>	If y, scaling relations are used to calculate OD
ABSTYPE	<code>flag_absorber</code>	Flux to be calculated for the specie
ZDLA	<code>z_DLA</code>	Redshift list of DLA
LNHIDLA	<code>log_N_HI_DLA</code>	Column Density list of DLA
BDLA	<code>b_DLA</code>	Line width parameter (b) list of DLA
METLTYPE	<code>flag_metal_method</code>	Method of adding metals to spectrum
METLLIST	<code>metal_trans_list</code>	Metal transition list for OD and flux calculation
LZMETAL	<code>log_Z_val_const</code>	Factor if metallicity is constant expressed in $\log Z_{\odot}$
METALUVB	<code>uvb_model_metal</code>	UVB used to calculate the metal specie abundances

- **Optical_Depth_METAL:** (BinaryHDU Table or HDF5 Dataset, Array, Float32)
This table/dataset contains total optical depths for all metal species (as specified by variable `metal_trans_list`). The properties of metals set by variables described in §5.6. This does not include the contribution of H I Lyman series forest or DLA. Note that `Optical_Depth_METAL` extension/group may not exist if the METAL properties are not specified. In such cases, WEAVEIFY would assume that Metals are not present along the sightline. The shape of this array is same as that of `Wavelength_For_Optical_Depth` array.

4.3 Header format

The header contains the information of all the variables used to generate the mock spectrum. Since the variables change from one spectrum to another, we save the given list of parameters for each spectrum individually. Fits file format does not allow the length of Header keyword (also known as cards) to be more than 8 characters. In Table 1, we provide the list of header keywords and their meaning. Even though hdf5 format allows user to store header keyword to be of any length, we follow the same procedure as fits format. This ensures the consistency of WEAVEIFY output irrespective fits or hdf5 format.

4.4 Script to read WEAVEIFY output

To facilitate users, we provide a python script to read the WEAVEIFY output files. The location of this file is shown in Fig. 11. The file contains a function `read_weaveify_output_fields`. Users can call this function in their python script. The function returns a dictionary that contains header, wavelength arrays and mock spectra arrays. We refer reader to the docstring of the function in file `read_weaveify_output_all_format.py` that explains the argument list in details. This data can be access using following commands,

```
from path.to.location.read_weaveify_output_all_format import \
read_weaveify_output_fields

out_dict = read_weaveify_output_fields(out_path,\
                                       flag_output_format='fits',flag_mpi='n',rank=None,\
                                       spec_id=None,flag_print_output='n')

# Syntax: out_dict[<extension or hdf5 group name>][<field name>]

# For example, to read flux from spec_id=0, we can use
flux_arr = out_dict['MOCK_SPECTRA-0']['Flux']

# To read wavelength from COMMON_FIELDS, we can use
wavelength_arr out_dict['COMMON_FIELDS']['Wavelength_For_Flux']

# To read header dictionary
header_dict = out_dict['MOCK_SPECTRA-0']['Header']
```

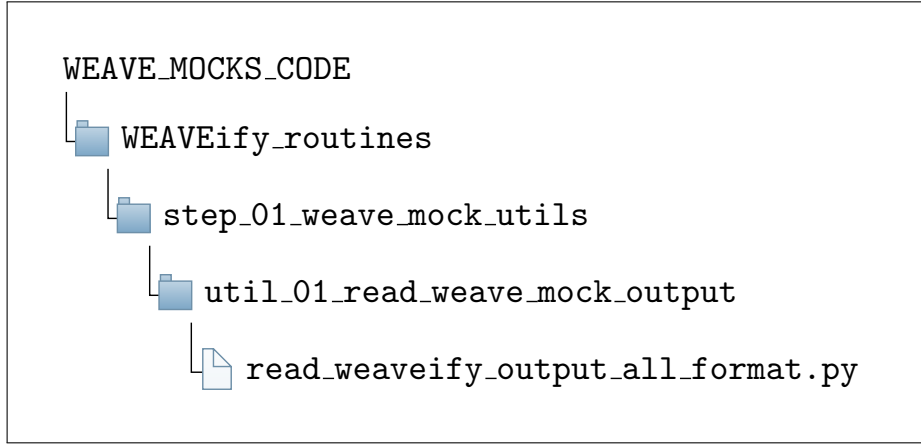


Figure 11: The location of sample python script to read the WEAVEIFY output files.

5 Modifying WEAVEIFY parameters

5.1 DLA parameters

- There are three parameters that set the properties of DLA in WEAVEIFY, (i) log of H I column density ($\log_{\text{N_HI_DLA}}$ in cm^{-2}), (ii) line width (doppler) parameter (b_{DLA} in km s^{-1}) and (iii) redshift of DLA system (z_{DLA})
- These parameters can be edited in `weave_mocks.py` or supplied as arguments as follow

```
python weave_mocks.py  flag_absorber=HI_metal_DLA z_DLA=2.65
flag_tau=DLA log_N_HI_DLA=20.5 b_DLA=5.0 flag_cont_method=PCA
```

Output:Quick_Plot_Mock_Spectra-0.pdf.

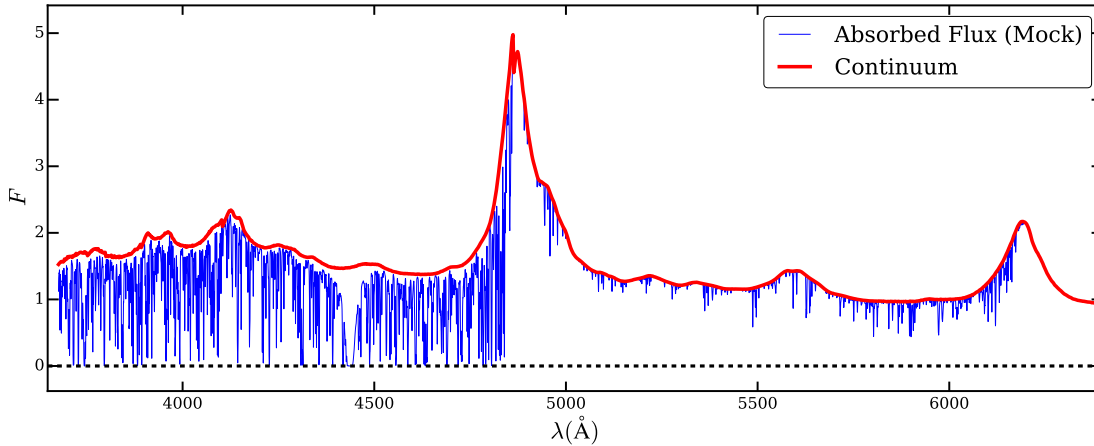


Figure 12: Example of mock spectrum with H I forest + DLA. The parameter of DLA are modified $\log N_{\text{HI}} = 20.5$, $b = 5.0 \text{ km s}^{-1}$ and $z_{\text{DLA}} = 2.65$.

- Here is an another example of different DLA properties

```
python weave_mocks.py  flag_absorber=HI_metal_DLA z_DLA=2.7
flag_tau=DLA log_N_HI_DLA=21.5 b_DLA=5.0 flag_cont_method=PCA
```

Output:Quick_Plot_Mock_Spectra-0.pdf.

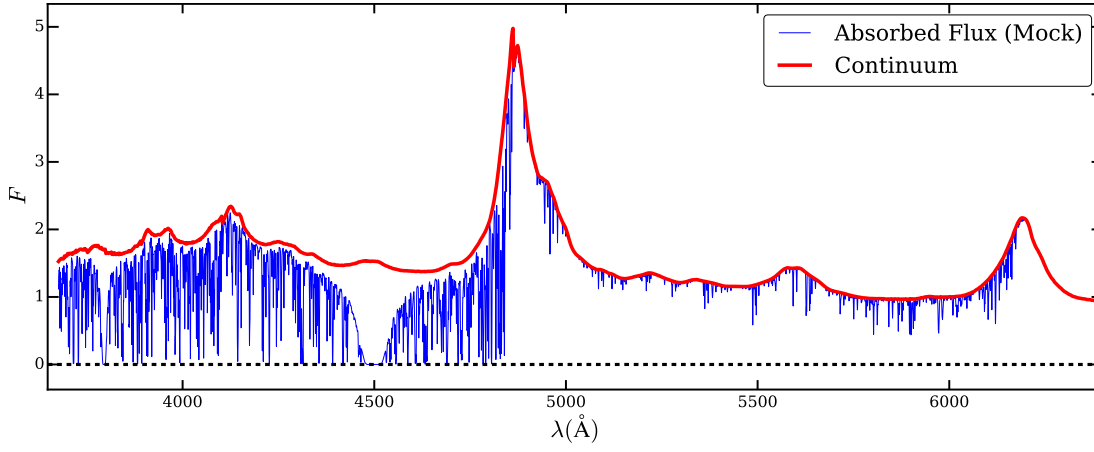


Figure 13: Example of mock spectrum with H I forest + DLA. Default parameter set for DLA in code are $\log N_{\text{HI}} = 21.5$, $b = 5 \text{ km s}^{-1}$ and $z_{\text{DLA}} = 2.7$.

5.2 WEAVE resolution mode

- In all the previous examples, WEAVEIFY modelled the mock spectra like low-resolution observations. If we instead want to model the high-resolution spectra we need to set variable `resolution_mode=HR`
- *It is important to note that users always need to calculate the optical depths when resolution is changed i.e., `flag_tau` variable needs to be specified when resolution mode is changed.*
- For high resolution WEAVE spectra we can use,

```
python weave_mocks.py  flag_absorber=HI_metal_DLA
flag_tau=HI_metal_DLA resolution_mode=HR flag_cont_method=PCA
```

Output:Quick_Plot_Mock_Spectra-0.pdf

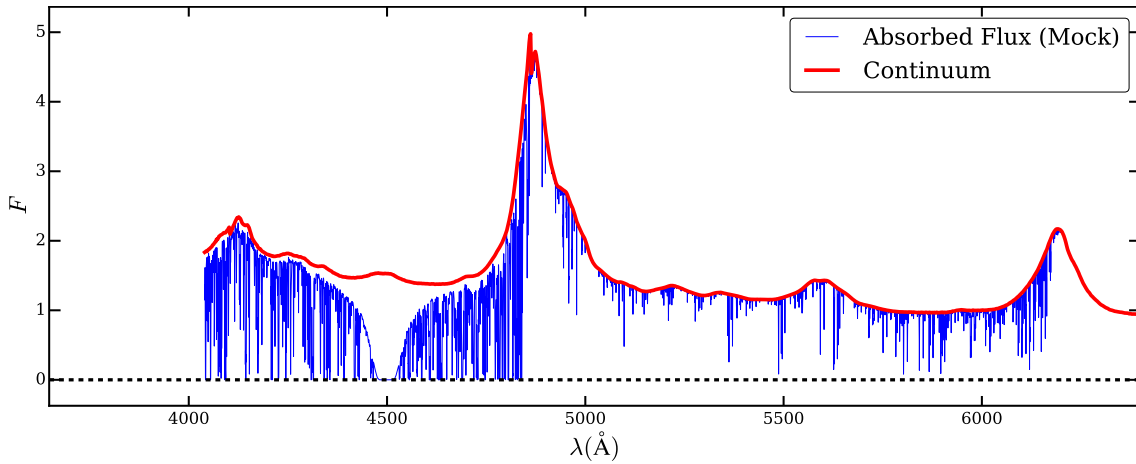


Figure 14: Mock spectrum with the wavelength resolution and wavelength sampling for WEAVE-HR ($R = 20000$) type observations (sampling parameters taken from OpR3 mocks)

5.3 Instrumental broadening parameters

- Many observations of quasar absorption spectra are taken from different instruments and/or telescopes. Often the instrumental broadening (also known as line spread function) of the absorption line is different for each instrument.
- For example, in WEAVE-LR sample the quasar absorption spectra would be obtained with $R = \lambda/\Delta\lambda \sim 5000$ while WEAVE-HR would have $R \sim 20000$.
- For standard WEAVE-LR and WEAVE-HR setting, we provide line spread function in files `instrumental_broadening_parameters_WEAVE_LR.txt` and `instrumental_broadening_parameters_WEAVE_HR.txt`
- The location of these files is shown in Fig. 15. Users can modify the files as per their needs.
- First two columns in these files are (lower and upper) wavelength ranges. The third column is FWHM of Gaussian LSF in velocity units i.e., km s^{-1} . We assume WEAVE LSF is Gaussian in shape (see Listing 1) for details.

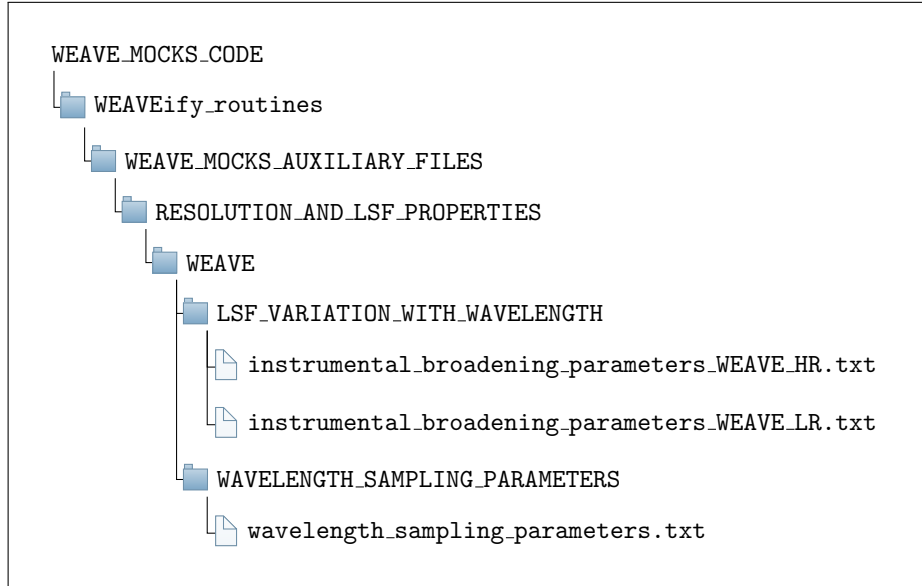


Figure 15: Location of instrumental broadening and wavelength sampling parameter files

Listing 1: Snippet of instrumental broadening parameter LR file

```

# Column-00 -->> Lower Wavelength (Angstroms) of wavelength range
# Column-01 -->> Upper Wavelength (Angstroms) of wavelength range
# Column-03 -->> FWHM of Gaussian

3000.00      3500.00      60.0
3500.00      4000.00      60.0
4000.00      4500.00      60.0
:            :            :
:            :            :
8500.00      9000.00      60.0
9000.00      9500.00      60.0
9500.00      10000.00     60.0

```

5.4 Wavelength sampling parameters

- Similar to instrumental broadening parameters, we provide wavelength sampling parameters in file `wavelength_sampling_parameters.txt` (see Fig. 15). The snippet of this file is shown in Listing 2
- Since wavelength coverage of WEAVE instrument is different in two resolution modes, we have tabulated the properties of each resolution mode and wavelength coverage separately in each row.
- In particular, users need to specify starting wavelength (CRVAL1), wavelength separation (CDEL1), Number of pixels, whether the wavelength are equally spaced in logarithmic or linear scale and central wavelength of the wavelength range.
- The default numbers specified in wavelength sampling parameters file are for WEAVE low resolution (LR) and high resolution (HR) mode. Users can modify these numbers as per their preferences.

Listing 2: Snippet of `wavelength_sampling_parameters.txt` file

```
# This file contains wavelength sampling parameters for WEAVE
# These parameters are taken from OpR3 Mocks
#
# Column-00 --> Resolution Mode (HR or LR)
# Column-01 --> Arm of the spectrograph Mode
# Column-02 --> CRVAL1 (Starting wavelength in angstroms)
# Column-03 --> CDEL1 (Wavelength separation in angstroms)
# Column-04 --> N_pixel_obs (Number of Pixel in the spectrum)
# Column-05 --> lambda_scale (LIN or LOG)
# Column-06 --> Central Wavelength of arm (Used to calculate
#               grating dispersion in noise addition)
#
# The wavelength array is obtained as
#
# if lambda_scale is LIN
# lambda_arr_obs = CRVAL1 + CDEL1 * np.arange(N_pixel_obs)
#
# if lambda_scale is LOG
# lambda_arr_obs = 10** (CRVAL1 + CDEL1 * np.arange(N_pixel_obs))
#
# -----
LR      Blue      3676.0      0.25      9649      LIN      4900.0
LR      Red       5772.0      0.25     15289      LIN      7950.0
# -----
HR      Blue      4040.0      0.05     12201      LIN      4250.0
HR      Green     4730.0      0.05     14401      LIN      5130.0
HR      Red       5948.0      0.05     18041      LIN      6450.0
# -----
```

5.5 Telescope throughput parameters

- Telescope throughputs specifies the efficiency of the telescope to capture the number of photons from source. Telescope throughput parameters are essential for realistic addition of the noise to the spectra.

- Since WEAVE instrument is still in installation phase, we have estimates of telescope throughputs. Thus telescope throughputs are subjected to change in future.
- The telescope throughputs are specified in file `telescope_throughput_V01.txt`. The telescope throughputs are taken from official WEAVE WEAVE-SYS-007 document. The location and sample content of this file is shown in Fig. 16 and Listing 3 respectively.

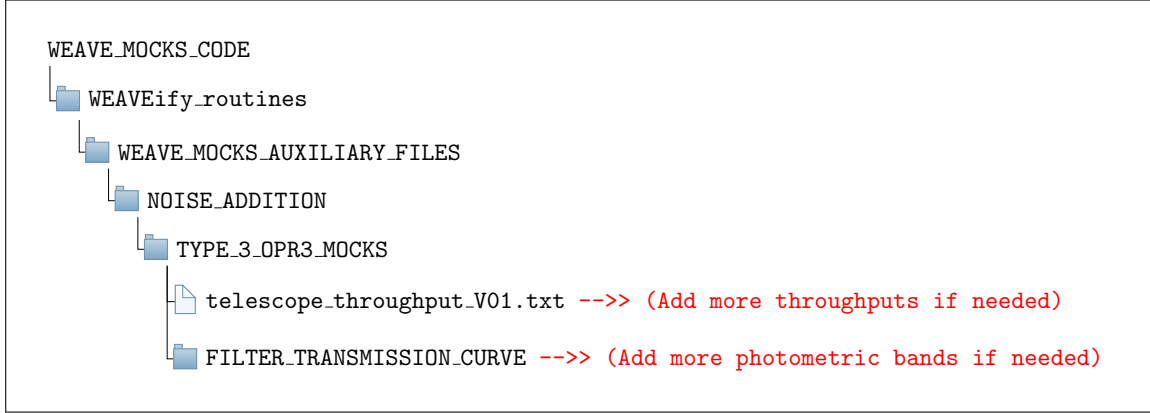


Figure 16: Location of telescope throughput file.

- There are 6 types of throughput to be specified in `telescope_throughput_V01.txt`, (i) Mirror reflectivity throughput, (ii) Central obscuration throughput, (iii) Prime focus corrector throughput, (iv) Fibre throughput, (v) Spectrograph throughput and (vi) CCD detector throughput
- It is important to note that wavelengths are specified in nano-meters while throughputs are always specified in percentages in this file. WEAVEIFY internally converts these quantities in to their standard units. The throughputs of various sources are multiplied to get effective throughput of the telescope.
- The ‘Essential’ type throughput refers to the minimum throughput that each system is required to meet. The ‘Optimal’ type throughput refers to the goal for each system to achieve this throughput. The noise routines of WEAVEIFY code currently use ‘Optimal’ throughput values.
- If users need to add more throughput they can append new rows to the table. In such case, one need to give an unique name to throughput type (Column-02). Once the new values are added in this file, users need to modify `throughput_type` variable to new unique name in `default_parameters_WEAVEify.py` file (see Fig. 1). This variable is currently set to ‘Optimal’.

Listing 3: Snippet of `telescope_throughput_V01.txt` file

```

# Column-00 -->> Resolution Mode (HR or LR)
# Column-01 -->> Arm of the spectrograph Mode
# Column-02 -->> Throughput type (Essential or Optimal or Observed)
# Column-03 -->> Wavelength in nano-meters
# Column-04 -->> M1 Reflectivity throughput (percent)
# Column-05 -->> Obscuration throughput (percent)
# Column-06 -->> Prime Focus Corrector throughput (percent)

```

```

# Column-07 -->> Fibres throughput (percent)
# Column-08 -->> Spectrograph throughput (percent)
# Column-09 -->> Detector throughput (percent)
#
# -----
LR   Blue   Essential   370.0   85.5   78.7   71.3   21.7   21.0   66.3
LR   Blue   Essential   400.0   85.8   78.7   79.1   53.7   46.3   92.1
:    :       :         :       :       :       :       :       :
:    :       :         :       :       :       :       :       :
HR   Red    Optimal    680.0   89.7   78.7   80.8   83.3   26.1   92.1
HR   Red    Optimal    685.0   89.6   78.7   80.8   83.4   21.0   91.4
# -----

```

5.6 Metal properties

There are three ways/options to add metals to mock spectra in WEAVEIFY, (i) using clody interpolation table, (ii) from standard library of metal profiles and (iii) pixel based metal addition (SDSS). Currently only cloudy interpolation table option is available in WEAVEIFY. We are working on implementing other two options with WEAVE-QSO group members.

5.6.1 Method 1 : Cloudy Interpolation Tables

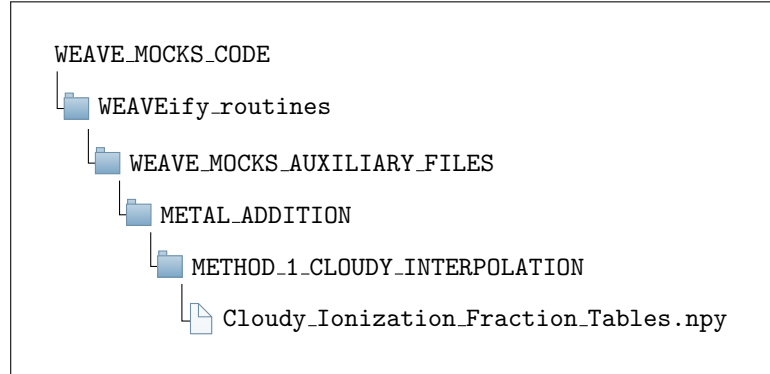


Figure 17: Location of cloudy interpolation tables.

- There are 4 input parameters needed to add metals in mock spectrum, (i) overdensity (Δ), (ii) temperature (T), (iii) metallicity $\log Z$ and (iv) UVB spectrum.
- The first two inputs are anyway needed to mock H I forest so they are should be available from cosmological fields. User need to specify how the metallicity varies along sightline. In principle, if simulation include metals, this field can also be a part of cosmological fields.
- In case, if metallicity distribution is not available from simulations, users have a choice to supply metallicity evolution using an array `log_Z_arr`. This is log of metallicity of IGM expressed in units of solar metallicity ($Z_{\odot}$). For example, if all values of `log_Z_arr` are -2.0 then metallicity along sightline is $Z_{\text{IGM}} = 10^{-2} Z_{\odot}$. **Note that this option is only available by modifying the `weave_mock.py` script as supplying an array as an command line argument not feasible.**

- On the other hand, if users need constant metallicity along sightline, there is simple option included in WEAVEIFY. User just need to specify variable value `log_Z_val_const=-2.0`. **This variable can be set either in `weave_mocks.py` script or supply as an argument in command line.**
- The cloudy table has been generated for 3 different UVB spectrum Haardt & Madau 2012 (HM12), Puchwein+2019 (P19) and Faucher-Giguere+2020 (FG20). User need to set `uvb_model_metal` to either FG20, HM12 or P19.
- Cloudy interpolation table has been calculated and saved in `Cloudy_Ionization_Fraction_Tables.npy` (see Fig. 17) In this table, ionization fraction of specie H I, He I, He II, C II, C IV, O I, O VI, Si II, Si III, Si IV, Mg II, Fe II, N V, Ne VIII is stored
- To quickly test how to change metal properties, user can run following command,

```
python weave_mocks.py flag_absorber=HI_metal_DLA flag_tau=metal
log_Z_val_const=-2.0 uvb_model_metal=FG20 flag_cont_method=PCA
```

Output:Quick_Plot_Mock_Spectra-0.pdf

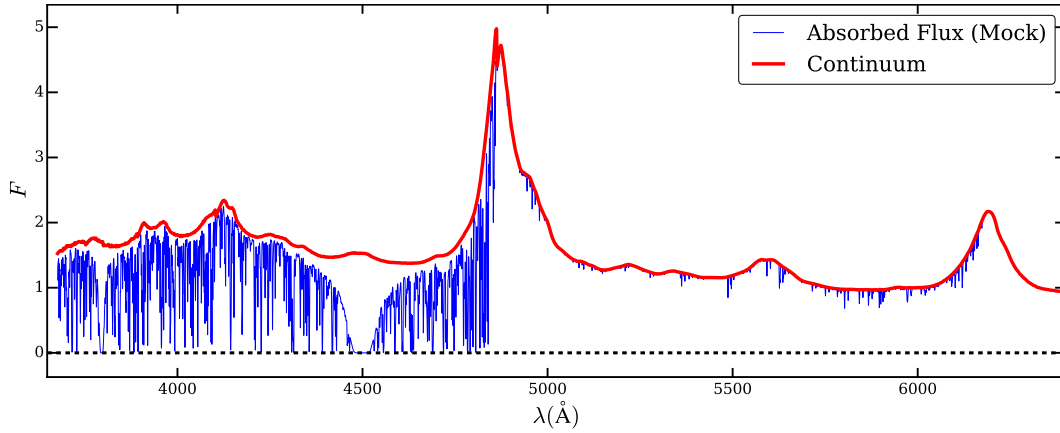


Figure 18: Example of mock spectrum H I ($\text{Ly}\alpha + \text{Ly}\beta$) forest + Metals + DLA. The IGM metallicity is $\log(Z_{\text{IGM}}/Z_{\odot}) = -2.0$

- Another example of change in metallicity,

```
python weave_mocks.py flag_absorber=HI_metal_DLA flag_tau=metal
log_Z_val_const=-1.0 uvb_model_metal=FG20 flag_cont_method=PCA
```

Output:Quick_Plot_Mock_Spectra-0.pdf

How do one add/modify metal species in mock spectrum?

- User may need to add specific transision of metal species in mock spectrum. This can be done in two steps.
- WEAVEIFY uses `<specie_name>-<transition_rest_wavelength>` format to recognize the particular transition of a given specie. For example, H I $\text{Ly}\alpha$ transision is represented by "HI-1215" whereas C IV doublets are represented by "CIV-1548" and "CIV-1550".
- A full list of all transition is given in file `specie_transition_properties.txt` (see Fig. 20 and Listing 4) WEAVEIFY uses the information given in this file to compute mock spetrum.

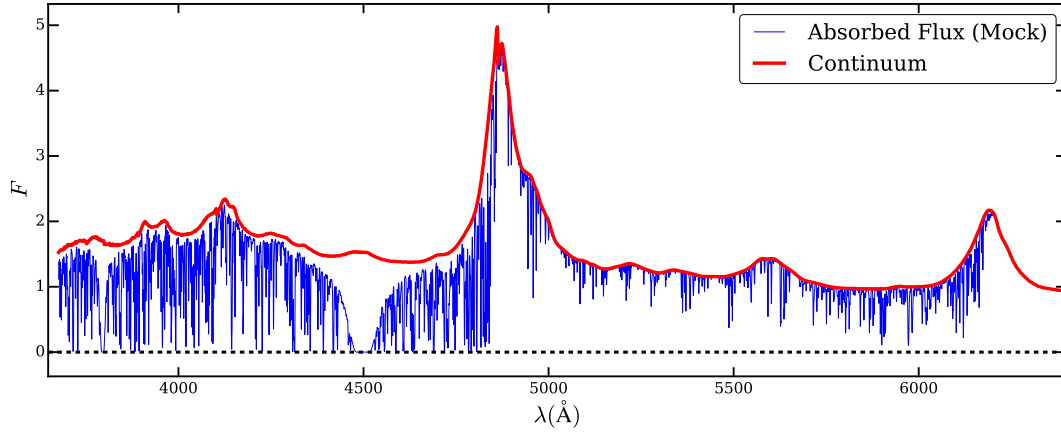


Figure 19: Example of mock spectrum H I ($\text{Ly}\alpha + \text{Ly}\beta$) forest + Metals + DLA. The IGM metallicity is $\log(Z_{\text{IGM}}/Z_{\odot}) = -1.0$

```
WEAVE_MOCKS_CODE
├── WEAVEify_routines
│   └── WEAVE_MOCKS_AUXILIARY_FILES
│       └── ABSORPTION_TRANSITION_LIST
│           ├── specie_transition_properties.txt -->> (File specifying all transitions)
│           ├── atom.dat -->> (Refer to this file for adding more transitions if needed)
│           └── Readme.txt
```

Figure 20: Location of file containing the list of transitions used in WEAVEIFY.

Listing 4: Snippet of specie_transition_properties.txt file

```
# Column-00 -->> Transition Alias or Name
# Column-01 -->> Number of protons in element
# Column-02 -->> Number of neutrons in element
# Column-03 -->> Rest wavelength in angstroms
# Column-04 -->> Oscillator strength of transition f_lu
# Column-05 -->> Quantum Mechanical Damping Coefficient
#-----
HI-1025      1    0    1025.72230    0.07912    1.897e8
HI-1215      1    0    1215.67010    0.41640    6.265e8
HI-972       1    0    972.053680    0.02900    8.127e7
HI-949       1    0    949.743100    0.01390    4.204e7
HI-937       1    0    937.803500    0.00780    2.450e7
#-----
HeI-584      2    2    584.334000    0.28500    1.799e9
HeI-537      2    2    537.029600    0.07520    5.663e8
:            :    :           :           :
:            :    :           :           :
:            :    :           :           :
SiIII-1526   14   14    1526.70698    0.13300    1.130e9
SiIII-1206   14   14    1206.50000    1.63000    2.480e9
SiIV-1393    14   14    1393.76018    0.51300    8.800e8
SiIV-1402    14   14    1402.77291    0.25400    8.620e8
#-----
```

- Once the transition is found in this file, user need to add the new transition in the python list `metal_trans_list` in file `weave_mocks.py`.
- For example, there are three main metal transition specified in `weave_mocks.py` file (by default) namely SiIV-1402, CIV-1548 and CIV-1550. Let us suppose user wants add one more transition say OI-1302.
- Then user need to change following line in `weave_mocks.py`,

```
metal_trans_list=["SiIV-1402","CIV-1548","CIV-1550"]
to
metal_trans_list=["SiIV-1402","CIV-1548","CIV-1550","OI-1302"]
```

- Similarly there are two main transition H I specified in `weave_mocks.py` file (by default) namely HI-1215 (H I Ly α) and HI-1025 (H I Ly β). Suppose user wants add H I Ly γ transition say HI-972. Then user need to change following line in `weave_mocks.py`,

```
HI_trans_list=["HI-1025","HI-1215"]
to
HI_trans_list=["HI-1025","HI-1215","HI-972"]
```

- After these changes user can continue to use WEAVEIFY code as mentioned in previous sections.

How do one add/modify metal species in mock spectrum if the transition is not listed in the `specie_transition_properties.txt` file?

- It is possible that some transition may not have listed in the file `specie_transition_properties.txt`. In such cases, one can add the transition to this file. However, we recommend users to use the same naming convention as described above.
- If users do know the value of oscillator strength and quantum mechanical damping coefficient, they can refer to file `atom.dat` (see Fig. 20)
- `atom.dat` file is taken from code VPFIT and contains list of (mostly) all known transition found in quasar absorption spectra. **Note that WEAVEIFY does not internally use `atom.dat` file. It is given just to facilitate users.**
- Once user add the new transition in file `specie_transition_properties.txt`, they can modify `weave_mock.py` file as discussed in previous section.

5.7 Continuum placement parameters

In WEAVEIFY, the continua are generated randomly using Principle Component Analysis (PCA) method. In this method, a random realization of continuum is generated from a composite continuum in red side (with respect to $\text{Ly}\alpha$ emission) of the QSO spectrum. Using the random continuum realization, composite spectrum and PCA components, the continuum is generated at all other wavelengths. There are 3 main variables in WEAVEIFY that controls the continuum placement which are described below.

- `flag_cont_method` : This variable tells WEAVEIFY a method to generate continuum. Currently only ‘PCA’ method of continuum generation is implemented in WEAVEIFY.
- `random_seed_cont` : As mentioned earlier, a realization of continuum is generated from composite spectrum. However, sometimes users may want to test the code using same realization of continuum. If `random_seed_cont` is set to some integer value then it would generate a pseudo-random numbers such that if same value of `random_seed_cont` is used again the continuum will be same. By default `random_seed_cont=None` indicating that the continua are generated using true random numbers.
- `alpha_cont` : It is possible that users would like to test the extreme cases of variation in continua in $\text{Ly}\alpha$ forest. The power-law index (α) of QSO spectra can be used to generate a large variation continuum $F_{\text{cont}} \propto \lambda^\alpha$. Setting the value of variable `alpha_cont` provides a convenient way to generate large variation in continua. The typical values of α varies from -0.5 to -3.0 but users can chose more extreme range like 0.0 to -5.0

We can now change continuum by specifying QSO power law index (`alpha_cont`)

```
python weave_mocks.py flag_test=weave flag_absorber=HI_metal_DLA
flag_cont_method=PCA alpha_cont=-0.5 random_seed_cont=123
```

Output:Quick_Plot_Mock_Spectra-0.pdf

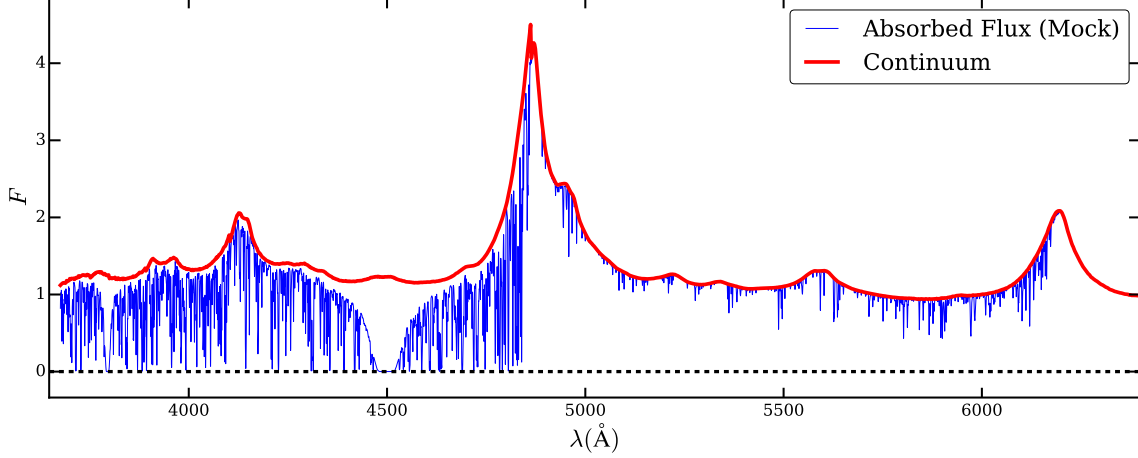


Figure 21: Mock spectrum with another continuum (Continuum templates are randomly generated from PCA). QSO continuum power-law index used in above example is $\alpha = -0.5$

```
python weave_mocks.py flag_test=weave flag_absorber=HI_metal_DLA
flag_cont_method=PCA alpha_cont=-3.0 random_seed_cont=123
```

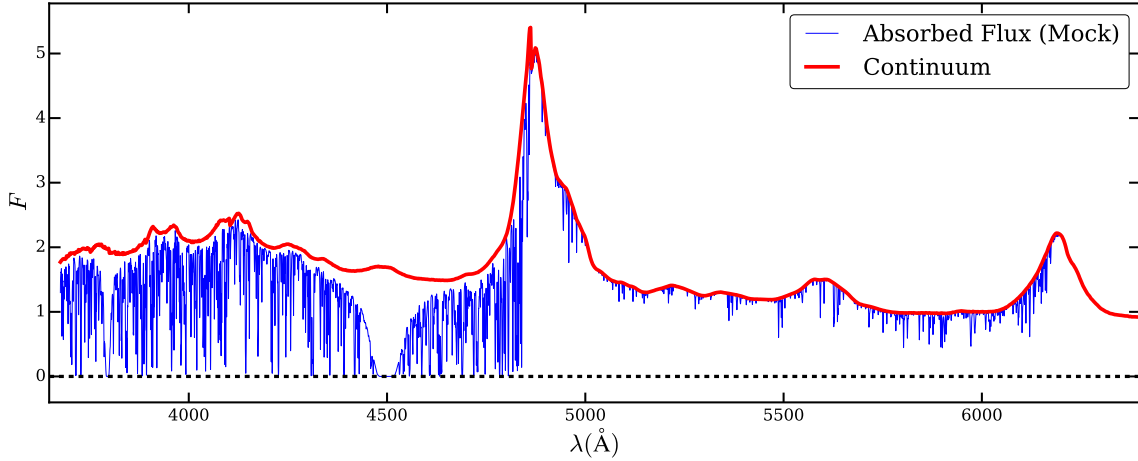


Figure 22: Mock spectrum with another continuum (Continuum templates are randomly generated from PCA). QSO continuum power-law index used in above example is $\alpha = -3.0$

5.8 Noise variation parameter

We estimate the noise for a given observation by first counting the total number of source photons received by the telescope as given below,

$$N_{\text{source}} = 10^{-m/2.5} \times N_{\text{photons, m=0}} \times f_{\text{atm}}(\text{airmass}) \times t_{\text{exp}} \times \eta_{\text{telescope}} \times A_{\text{unobst}} \quad (1)$$

where m is magnitude of source in given wave band, $N_{\text{photons, } m=0}$ is number of photons/sec/ $\text{\AA}/\text{m}^2$ received from source with $m = 0$ at the top of atmosphere, f_{atm} is transmission of atmosphere at a given airmass, t_{exp} is exposure time in seconds, $\eta_{\text{throughput}}$ is total throughput of telescope and A_{unobst} is unobstructed area of main mirror in m^2 .

We count the total number of sky photons (N_{sky}) similar to N_{source} by replacing the source brightness magnitude (m_{source}) by sky brightness magnitude m_{sky} . Since the phenomenon of counting the photons on CCD is governed Poisson statistics, we calculate the noise contribution of source (or sky) as $\sigma = \sqrt{N}$. Assuming the noises are uncorrelated, the total noise contribution and signal-to-noise ratio (SNR) is given by,

$$\sigma_{\text{tot}} = \sqrt{\sigma_{\text{source}}^2 + \sigma_{\text{sky}}^2 + \sigma_{\text{dark}}^2 + \sigma_{\text{read-out}}^2}$$

$$\text{SNR} = N_{\text{source}} / \sigma_{\text{tot}} \quad (2)$$

where $\sigma_{\text{dark}}, \sigma_{\text{read-out}}$ are noise due to dark current and CCD read-out respectively.

Some of the noise (miscounting of photons) can also come from thermal radiation of the detector. If $n_{\text{e,dark}}$ is number of ‘dark’ electrons detected per unit time and N_{pix} be the number of pixels used to observe an object then the Dark shot noise (σ_{dark}) is given by,

$$\sigma_{\text{dark}} = \sqrt{N_{\text{pix}} n_{\text{e,dark}} t_{\text{exp}}} \quad (3)$$

Read out noise is slightly different than the Shot noise. While reading the dectector, the amplifiers contribute some additional noise to the signal. This is a characteristic of the chip and of the read out mode used. Let $n_{\text{read-out}}$ be read-out-noise per read-out and N_{pix} be the number of pixels read in single exposure time. If there are N_{epoch} exposures then the total read-out noise ($\sigma_{\text{read-out}}$) is given by,

$$\sigma_{\text{read-out}} = \sqrt{N_{\text{pix}} N_{\text{epoch}}} \times n_{\text{read-out}} \quad (4)$$

Note that $n_{\text{read-out}}$ is not in the square root.

Following are the main variables in WEAVEIFY that can be changed to vary SNR of the mock spectra.

- **flag_noise** : This variable defines the method of adding noise to the spectra. The default value of **flag_noise**= ‘OpR3’ . If **flag_noise** is set to None, then noise will not be added to the spectra.
- **m_source** and **wave_band_source** : The source magnitude and the wavelength filter is defined by the variables **m_source** and **wave_band_source** respectively. WEAVEIFY assumes that the source magnitudes are specified in UBVRI photometric systems.
- **m_sky** and **wave_band_sky** : Similar to source, the sky magnitude and the wavelength filter is defined by the variables **m_sky** and **wave_band_sky** respectively. WEAVEIFY assumes that the sky magnitudes are specified in UBVRI photometric systems.
- **t_exp** : Total exposure time in seconds
- **airmass** : Airmass of the atmosphere. The default value is 1.107.
- **seeing_profile** : Atmospheric seeing effect is controlled by three parameters. **seeing_profile** defines the functional form of seeing. The options for **seeing_profile** is either ‘Gaussian’ or ‘Moffat’. The default seeing profile is Moffat.
- **beta_moffat** : The shape of the Moffat profile depends on the β parameter. The default value is 2.5. This variable is only used if seeing profile is set to Moffat.

- `fw hm_seeing` : The variable `fw hm_seeing` defines the full width at half maximum of seeing profile expressed in arcsec. The default value is 0.75 arcsec.
- `read_noise` : Read-out noise per electron. The default value is 3.3.
- `dark_current` : The noise due to the dark current. The default values is 0.0.
- `throughput_type` : WEAVE scientific instrument team has provided the variation of telescope throughput with wavelength for LR and HR resolution mode. Currently, there are two options to chose for `throughput_type` . The ‘Essential’ type throughput refers to the minimum throughput that each system is required to meet. The ‘Optimal’ type throughput refers to the goal for each system to achieve this throughput. The noise routines currently use ‘Optimal’ as default throughput values (see §5.5 for details).
- `dia_telescope` : Diameter of the telescope in meters. The default value is 4.2m for WHT.
- We show an example of changing source magnitude on noise properties of the WEAVEIFY mock spectrum.
- We can change the magnitude of the source to 18.5 as follows

```
python weave_mock s.py  flag_absorber=HI_DLA
flag_cont_method=PCA  flag_noise=0pR3 m_source=18.5
```

Output:Quick_Plot_Mock_Spectra-0.pdf.

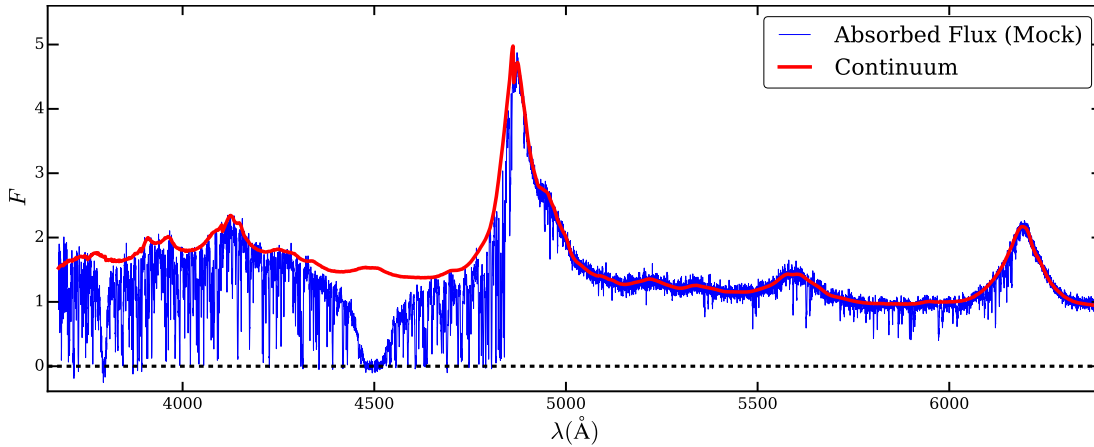


Figure 23: Example of mock spectrum with H I forest + DLA for a source magnitude of 18.5.

- Similarly, we can change the magnitude of the source to 20.5 as follows

```
python weave_mock s.py  flag_absorber=HI_DLA
flag_cont_method=PCA  flag_noise=0pR3 m_source=20.5
```

Output:Quick_Plot_Mock_Spectra-0.pdf.

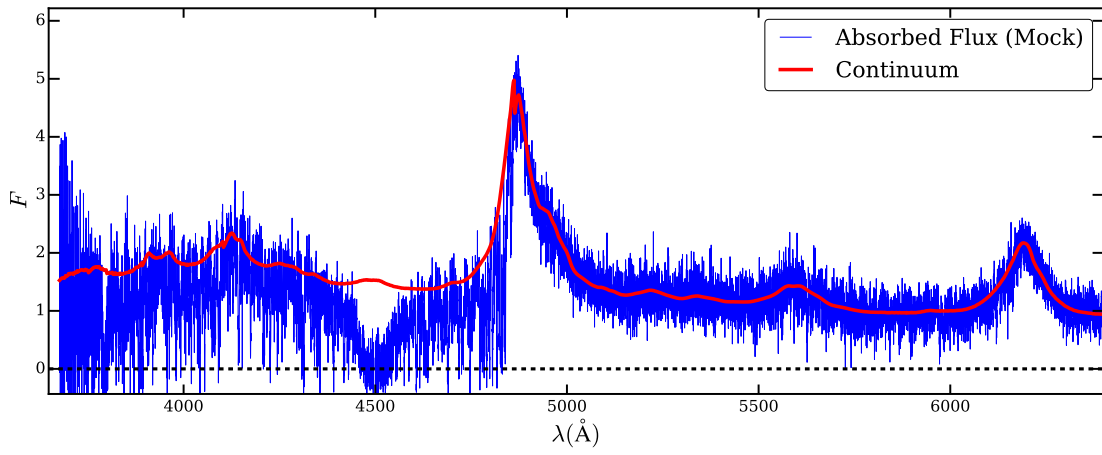


Figure 24: Example of mock spectrum with H I forest + DLA for a source magnitude of 20.5.

6 Detail description of free parameters in WEAVEIFY

In this sections, we describe all the parameters and fields that are required by WEAVEIFY to generate mock spectra. We have also provided the default value of the variables.

6.1 Cosmological mocks

Table 2: Cosmological mock fields parameters

WEAVEIFY Variable	Default Value	Description
<code>sim_name</code>	<code>SHERWOOD</code>	Name of the simulation suite
<code>L_box</code>	<code>80.0</code>	Length of simulation box in h^{-1} cMpc
<code>N_Grid</code>	<code>2048</code>	Number of pixels in one skewer
<code>Delta_arr</code>	<code>None</code>	Overdensity (Δ) field along sightline (1D array)
<code>T_gas_arr</code>	<code>None</code>	Temperature (T) field along sightline (1D array)
<code>v_gas_arr</code>	<code>None</code>	Velocity (v km s $^{-1}$) field along sightline (1D array)
<code>f_HI_arr</code>	<code>None</code>	H I fraction ($n_{\text{HI}}/n_{\text{H}}$) field along sightline (1D array)
<code>log_Z_arr</code>	<code>None</code>	(Optional) Metallicity (in $Z_{\odot}$) along sightline (1D array)
<code>z_arr</code>	<code>None</code>	Redshift along sightline (1D array)

WEAVEIFY needs cosmological mocks as input and calculates the optical depths for H I Lyman series forest, metal transitions and DLAs. These cosmological mocks are supplied through 5 variables which are 1d numpy arrays. The variables in WEAVEIFY and their descriptions are provided in Table 2. One can extract the 1d fields by extracting the fields along random skewers in cosmological simulation box. The variables length of the simulation box and number of grids per skewers are used in WEAVEIFY while calculating the optical depths.

6.2 QSO parameters

QSO parameters decides if the QSO absorption spectra contains only forest, has DLA and/or contains metal. A full list of variables and their description is provided in Table 3. The most important variables are `spec_id`, `flag_tau` and `flag_absorber`. The variable `spec_id` could be string or number. The data for given spectrum is saved in the output file under `MOCK_SPECTRA- < spec_id >`. Hence user always need to change this variable while making mocks for multiple spectra. The flag `flag_tau` describes the species for which optical depth need to be calculated. The options available for `flag_tau` variables are mentioned in Table 3. The flag `flag_absorber` is similar to `flag_tau` that describes the species for which flux $F = e^{-\tau}$ is calculated.

Very Important: If users specify DLA or METAL option in `flag_tau` but did not specify `flag_absorber`, then the final spectrum would not contain DLA or metals. On the other hand if users do not specify DLA or METAL option in `flag_tau` but specify in `flag_absorber`, then WEAVEIFY would look for DLA and METAL optical depths in file. If these fields are not present, then result would occur.

We keep optical depth and flux calculations separately because this provides more flexibility to users. For example, if users have already run a code for H I forest but later decided to add DLA in some spectra, then this task become much simpler. In such case, user need to set `flag_tau=DLA` and `flag_absorber=HI_DLA`

Table 3: List of QSO Parameters

WEAVEIFY Variable	Default Value	Description
<code>z_em</code>	<code>3.0</code>	Emission redshift of QSO
<code>spec_id</code>	<code>None</code>	Unique identification number or string to save data
<code>HI_trans_list</code>	<code>["HI-1025", "HI-1215", "HI-972", "HI-949"]</code>	List of HI Lyman series transition HI-1215 $\Rightarrow$ Ly α , HI-1025 $\Rightarrow$ Ly β HI-972 $\Rightarrow$ Ly γ , HI-949 $\Rightarrow$ Ly δ (Add more transitions if needed)
<code>flag_tau_each_trans</code>	<code>n</code>	If n, the Optical depths are calculated for one transition and scaled using oscillator strengths for other transitions. If y then optical depth is calculated for each transition seperately (usually computationally expensive).
<code>flag_tau</code>	<code>None</code>	Species for which optical depth is to be calculated <code>None</code> $\Rightarrow$ OD is not calculated <code>HI</code> $\Rightarrow$ OD for HI forest only <code>HI_metal</code> $\Rightarrow$ HI forest + Metals <code>HI_DLA</code> $\Rightarrow$ HI forest + DLA <code>HI_metal_DLA</code> $\Rightarrow$ HI forest + DLA + Metals
<code>flag_absorber</code>	<code>HI</code>	Species for which Flux is to be calculated <code>HI</code> $\Rightarrow$ Flux for HI forest only <code>HI_metal</code> $\Rightarrow$ HI forest + Metals <code>HI_DLA</code> $\Rightarrow$ HI forest + DLA <code>HI_metal_DLA</code> $\Rightarrow$ HI forest + DLA + Metals

6.3 DLA parameters

The variables changing the DLA properties are mentioned in Table 4. A more detail description of these parameters is given in §5.1.

Table 4: DLA Parameters

WEAVEIFY Variable	Default Value	Description
<code>log_N_HI_DLA</code>	21.5	Log HI Column density of DLA
<code>z_DLA</code>	2.7	Redshift of DLA
<code>b_DLA</code>	10.0	Line width in (km/s) parameter of DLA

6.4 Metal parameters

The variables changing the DLA properties are mentioned in Table 5. A more detail description of these parameters is given in §5.6.

Table 5: Metal Addition Parameters

WEAVEIFY Variable	Default Value	Description
<code>flag_metal_method</code>	<code>cloudy_interpolation</code>	Method to include metals in forest
<code>uvb_model_metal</code>	FG20	UVB model used to calculate metal abundances UVB model Options FG20: Faucher-Giguere+2020 HM12: Haardt & Madau 2012 P19: Puchwein+2019
<code>log_Z_val_const</code>	-1.5	Value of metallicity if it is constant along sightline
<code>metal_trans_list</code>	["SiIV-1402", "CIV-1548", "CIV-1550"]	Metal Transition list (Add more transitions if needed)

6.5 Wavelength sampling parameters

The variables changing the DLA properties are mentioned in Table 6. A more detail description of these parameters is given in §5.2, 5.3 and 5.4.

Table 6: Wavelength sampling parameters

WEAVEIFY Variable	Default Value	Description
<code>instrument</code>	WEAVE	Name of the instrument
<code>resolution_mode</code>	LR	Resolution mode HR or LR
<code>CRVAL1</code>	None	Start of wavelength array in angstroms
<code>CDEL1</code>	None	Wavelength spacing in angstroms
<code>N_pixel_obs</code>	None	Number of pixels in wavelength/flux array
<code>lambda_scale</code>	None	Wavelength spacing scale (LIN or LOG)
<code>lsf_type</code>	GAUSSIAN	Functional form of Line Spread Function

6.6 Continuum placement parameters

The variables changing the continuum placements are mentioned in Table 7. A more detail description of these parameters is given in §5.7.

Table 7: Continuum placement parameters

WEAVEIFY Variable	Default Value	Description
<code>flag_cont_method</code>	PCA	Method to generate random continuum
<code>random_seed_cont</code>	None	Seed to generate random continuum
<code>alpha_cont</code>	None	Power-law slope of the continuum $F_{\text{cont}} \propto \lambda^\alpha$

6.7 Noise parameters

The variables changing the noise properties are mentioned in Table 8. A more detail description of these parameters is given in §5.8 and §5.5.

Table 8: Noise parameters

WEAVEIFY Variable	Default Value	Description
<code>flag_noise</code>	OpR3	Type of noise to be added to spectra
<code>source_type</code>	point	Type of source point or extended
<code>dia_telescope</code>	4.2	Diameter of Telescope in meters
<code>dia_fibre_arcsec</code>	1.3045	Diameter of Fibre in arcsec
<code>m_source</code>	20.0	Apparent magnitude of source
<code>wave_band_source</code>	R	Wavelength filter of source magnitude
<code>m_sky</code>	20.92	Apparent magnitude of sky
<code>wave_band_sky</code>	B	Wavelength filter of sky magnitude
<code>t_exp</code>	3060.0	Exposure time in seconds
<code>airmass</code>	1.107	Airmass value for the atmosphere
<code>seeing_profile</code>	Moffat	Functional form of seeing Gaussian or Moffat
<code>beta_moffat</code>	2.5	Moffat profile index
<code>fwhm_seeing</code>	0.75	Full Width at half maximum of seeing in arcsec
<code>read_noise</code>	3.3	Read noise per electron
<code>dark_current</code>	0.0	Noise due to dark current
<code>ccd_pix_size_micron</code>	15.0	CCD pixel size in microns
<code>r_offset</code>	0.1	Offset of fibre from object center in arcsec
<code>throughput_type</code>	optimal	Type of telescope throughput

6.8 Cosmological parameters

Cosmological parameters used for generating mocks are usually fixed for a given simulation box. The cosmological parameters are initialized in function `cosmological_parameters` in file `default_parameters_WEAVEify.py`. ***Note: Cosmological parameters can only be changed by modifying the file default_parameters_WEAVEify.py. The cosmological parameters can not be changed by providing as an argument in command line..*** A more detail description of cosmological parameters along with their default values is given in Table 9.

Table 9: Cosmological Parameters

WEAVEIFY Variable	Default Value	Description
<code>omega_m</code>	0.308	Matter density parameter Ω_m
<code>omega_l</code>	0.692	Dark Energy density parameter Ω_Λ
<code>omega_r</code>	0.0	Radiation density parameter Ω_γ
<code>omega_k</code>	0.0	Curvature density parameter Ω_k
<code>omega_b</code>	0.0482	Baryon density parameter Ω_b
<code>hubble_param</code>	0.678	Hubble parameter h
<code>sigma_8</code>	0.829	Density fluctuations parameter σ_8
<code>Y</code>	0.24	Primordial Helium abundance by mass Y
<code>rho_c_by_hsq</code>	1.8791e-29	Critical density parameter ρ_c/h^2 , $\rho_c = 3H_0^2/(8\pi G)$
<code>ns</code>	0.961	Index of primordial power spectrum

6.9 MPI job submission parameters

The variables described in Table 10 are used to submit a parallel job using MPI processes. The most important parameter to submit a job in parallel is `flag_mpi`. Users need to set this parameter to 'y' if they want to run job in parallel. The variables `los_indx` and `N_los_total` are used to print the progress of the code. `los_indx` presents the spectrum on which WEAVEIFY is currently working on. While `N_los_total` is total number of spectra for which mock spectra needs to be generated. The details on how to submit a multi-core MPI job is described in §3.2. Note that there are two more variables `N_proc` and `rank` that are internally set when MPI job is submitted. Users do not need to modify these two variables (neither for serial or parallel jobs).

Table 10: MPI job submission Parameters

WEAVEIFY Variable	Default Value	Description
<code>flag_mpi</code>	n	Set this parameter to 'y' if you are using the code in parallel using MPI
<code>N_los_total</code>	1	Total number of mock spectra to be generated. This variable would be internally changed.
<code>los_indx</code>	1	Current mock spectra on which code is working. This variable would be internally changed.

6.10 Input / Output file saving parameters

The input / output directory and file names are set by variables as described in Table 11. Users always need to set the variable `out_path` to the directory where outputs will be stored. The prefix of the output files are set by `out_file_flux_base` and `out_file_tau_base` variables. Users can chose different names of output files. *Users need to make sure that the values of two variables are exactly the same if the optical depths and fluxes need to be saved in same file. WEAVEIFY provides a flexibility of saving the optical depths and fluxes in separte files. In such case, users need to set the value of out_file_tau_base different than the variable out_file_flux_base* If the value of variable `flag_make_quick_plot` is set to y, the output spectrum is plotted and saved in a file. The variable `out_file_plot_base` is prefix for output plot files. The output of spectrum

with `spec_id` would be saved in file `<out_file_plot_base>-<spec_id>.pdf` file.

Table 11: Parameters related to Input / Output

WEAVEIFY Variable	Default Value	Description
<code>out_path</code>	<code>None</code>	Path of directory where output needs to be stored
<code>out_file_flux_base</code>	<code>WEAVEified_Mock_Spectra</code>	Prefix of flux data file
<code>out_file_tau_base</code>	<code>WEAVEified_Mock_Spectra</code>	Prefix of optical depth data file. If OD needs to be saved in separate file, change this variable
<code>flag_output_format</code>	<code>fits</code>	Format of the output file fits or hdf5
<code>flag_make_quick_plot</code>	<code>n</code>	If y makes quick plot
<code>out_file_plot_base</code>	<code>Quick_Plot_Mock_Spectra</code>	Prefix of output plot file

6.11 Miscellaneous parameters

In addition to abover mentioned parameters / variables, there are additional variables described in file `default_parameters_WEAVEify.py`. These variables do not need to be change. We refer reader to `default_parameters_WEAVEify.py` file for more details.

7 Acknowledgement and Citations

8 Frequently Asked Questions (FAQs)

1. *I am running WEAVEIFY on single (or multi) core machine for 2 or more number of spectra. But the output is saved only for one spectrum (the last one)?*

You are not varying the variable spec_id in for loop. This variable tells the code to store the different spectrum in different blocks or HDUBinary Table. Note that spec_id can be a string or integer variable.

2. *How do I install the python packages mentioned in §1.2 ?*

If users have difficulty in installing python packages, they can use following steps to install packages.

- (a) Download and install miniconda using the scripts provided at <https://docs.conda.io/en/latest/miniconda.html>
- (b) In the next step we need to create a conda environment for installing the desired packages.
- (c) We have provided two *yml* files as shown in Fig. 25. Goto the above mentioned directory using cd command.

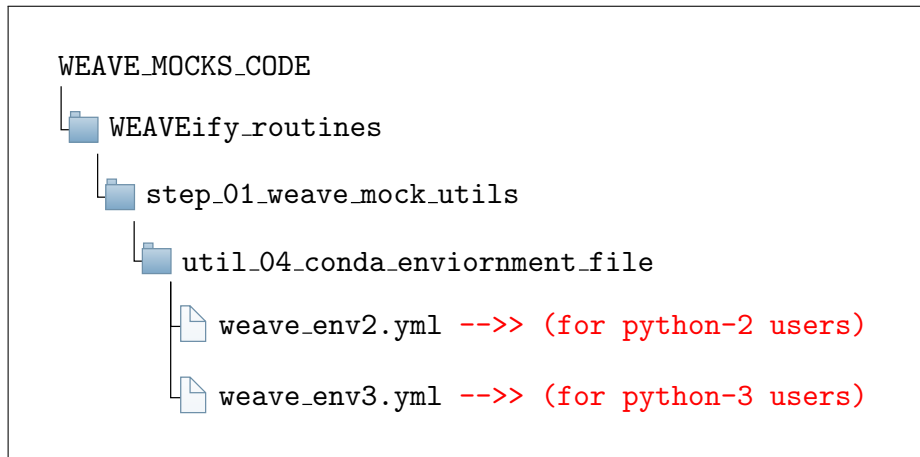


Figure 25: Location of conda enviornment files. The code has been tested for using all the packages mentioned in two yml files.

- (d) To create conda envrionment from the yml files, enter following command in terminal

```
conda env create --file weave_env3.yml -n weave_env3
```

- (e) Python2 users need use *weave_env2.yml* file in the above command to install python2 and associatated packages
- (f) This will install all the packages needed to run WEAVEIFY code.
- (g) To use the recently installed version of python, users need to activate conda enviornment using

```
conda activate weave_env3
```

(h) Similarly users can leave conda environment (after running the code) using

```
conda deactivate
```

(i) The code has been tested on both packages mentioned in two yml files and seems to perform well.