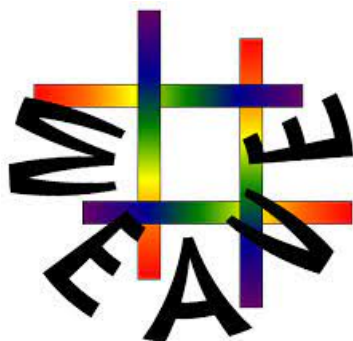


User guide for ASTRO-MOCKS (v2)

WEAVE-QSO Collaboration



Contents

1	Overview of ASTRO-MOCKS	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	4
2.2	Adding DLA to forest	5
2.3	Adding Metals to forest	6
2.4	Multiple options	7
3	Automating ASTRO-MOCKS for large number of spectra	7
3.1	Single core ASTRO-MOCKS automation	7
3.2	Multi core ASTRO-MOCKS automation using MPI	8
4	Reading ASTRO-MOCKS output	9
4.1	Output file names	9
4.2	Data format	11
4.3	Header format	13
4.4	Script to read ASTRO-MOCKS output	13

5	Modifying ASTRO-MOCKS parameters	14
5.1	DLA parameters	14
5.2	Wavelength sampling parameters	14
5.3	Metal properties	16
5.3.1	Method 1 : Cloudy Interpolation Tables	16
6	Detail description of free parameters in ASTRO-MOCKS	19
6.1	Cosmological mocks	19
6.2	QSO parameters	19
6.3	DLA parameters	21
6.4	Metal parameters	21
6.5	Wavelength sampling parameters	21
6.6	Cosmological parameters	22
6.7	MPI job submission parameters	22
6.8	Input / Output file saving parameters	23
6.9	Miscellaneous parameters	23
7	Acknowledgement and Citations	24
8	Frequently Asked Questions (FAQs)	24

1 Overview of ASTRO-MOCKS

1.1 Purpose of the code

ASTRO-MOCKS (together with 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. ASTRO-MOCKS 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. A separate code, WEAVEIFY, accounts for realistic noise, continuum placement and instrumental resolution effects of WEAVE instrument. ASTRO-MOCKS is written to model the H I forest, metal and DLA in the cosmological N-body or hydrodynamical simulations. The code is modular and flexible enough to mock the spectra for variety of simulations. We refer the reader to the paper describing ASTRO-MOCKS for details. The code is publicly available at <https://ingbitbucket.ing.iac.es/projects/WVQSO/repos/astromocks/>

1.2 Installation

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

```
git clone ssh://git@ingbitbucket.ing.iac.es:7999/wvqso/astromocks.git
```

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

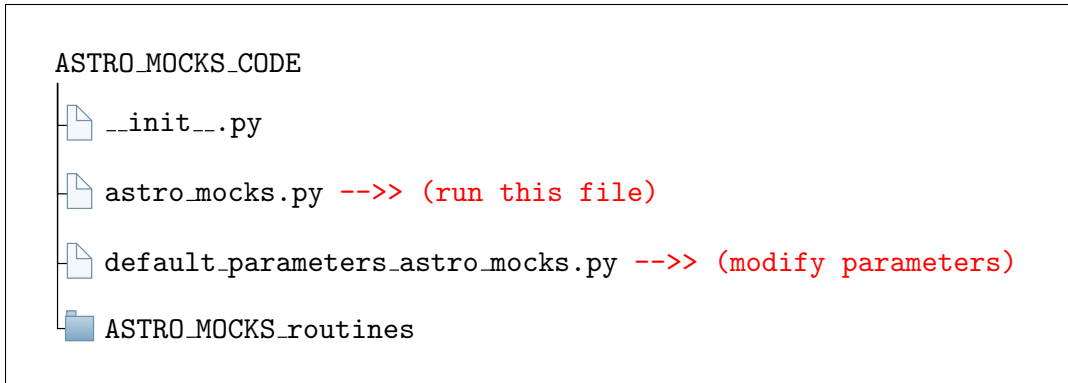


Figure 1: Basic structure of the code directory. Users need to run the file `weave_mock.py`. All the parameters of the code are specified in the file `default_parameters_astromocks.py`

- ASTRO-MOCKS 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_astromocks.py`. User may modify this parameter files later for their custom needs (see §3 for details). We recommend users to make a copy this file before modifying it.

1.3 Introduction

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

```
python astro_mocks.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 `astro_mocks.py` file.

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

- We provide few examples of `astro_mocks.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 tests examples are kept in directory `WEAVE` directory (see Fig. 2)
- In this directory there are 2 subdirectories `INPUT` and `OUTPUT`. `INPUT` directory contains cosmological mocks 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`

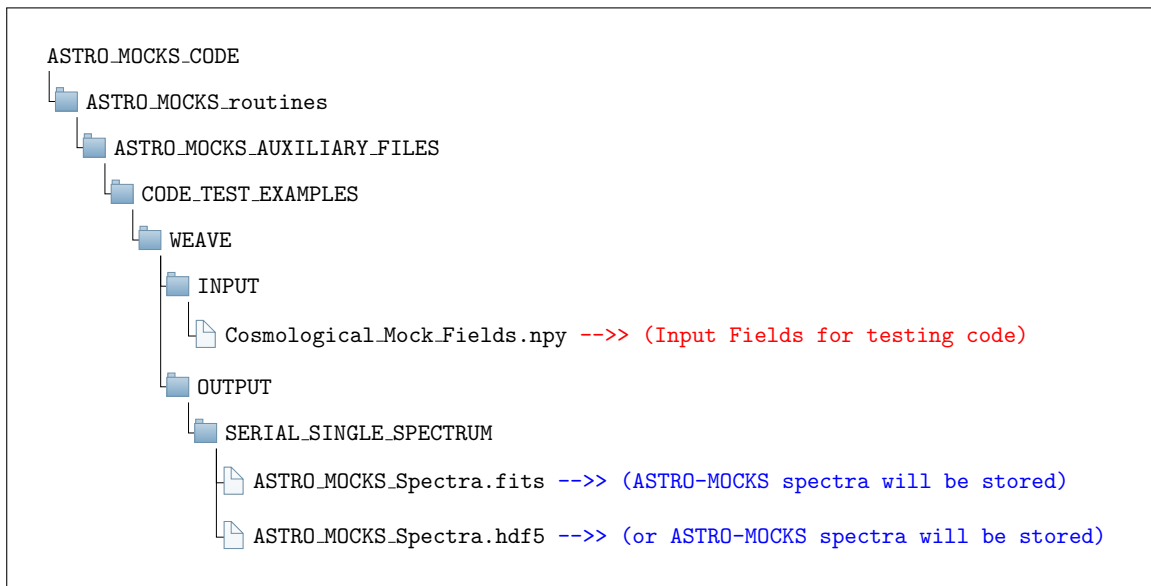


Figure 2: Location of input and output directory for test examples of the code.

2.1 H I Lyman series forest

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

```
python astro_mock.py flag_tau=HI
```

- 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.
- Next we generate the Ly series flux i.e., accounting for all the Ly-series transitions using following command,

```
python astro_mock.py flag_absorber=HI flag_make_quick_plot=y
```

- 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 `ASTRO MOCKS_Spectra.fits`. We can view part of the mock spectrum by opening file `Quick_Plot_Mock_Spectra-0.pdf`

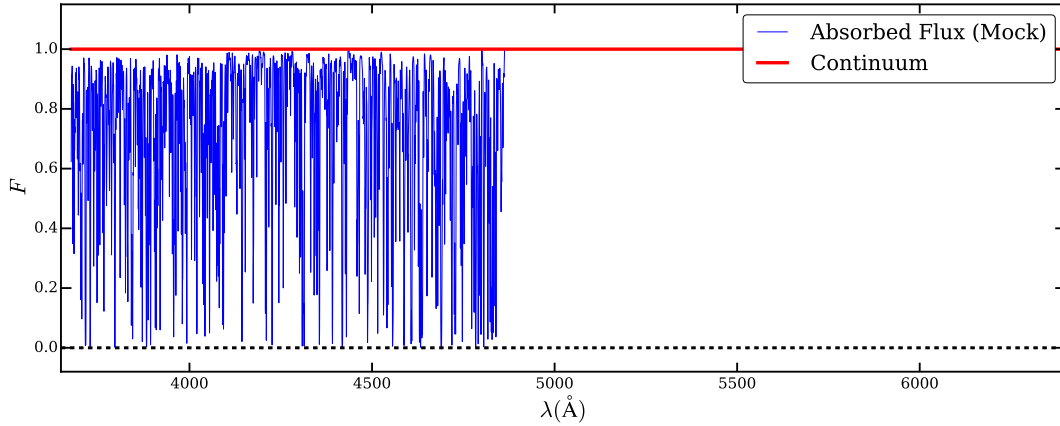


Figure 3: Test example of first mock spectrum H I ($\text{Ly}\alpha + \text{Ly}\beta$) forest only

2.2 Adding DLA to forest

- We can now add DLA to the mock spectra. To generate optical depth for DLA we can run command,

```
python weave_mocks.py flag_tau=DLA
```

- Generating the optical depths for DLAs is usually fast. We generate the Ly series flux for DLA,

```
python astro_mocks.py flag_absorber=HI_DLA flag_make_quick_plot=y
```

- Note that we set `flag_absorber=HI_DLA` to ensure that we plot both Ly-series forest and DLA systems.

Output: Quick_Plot_Mock_Spectra-0.pdf.

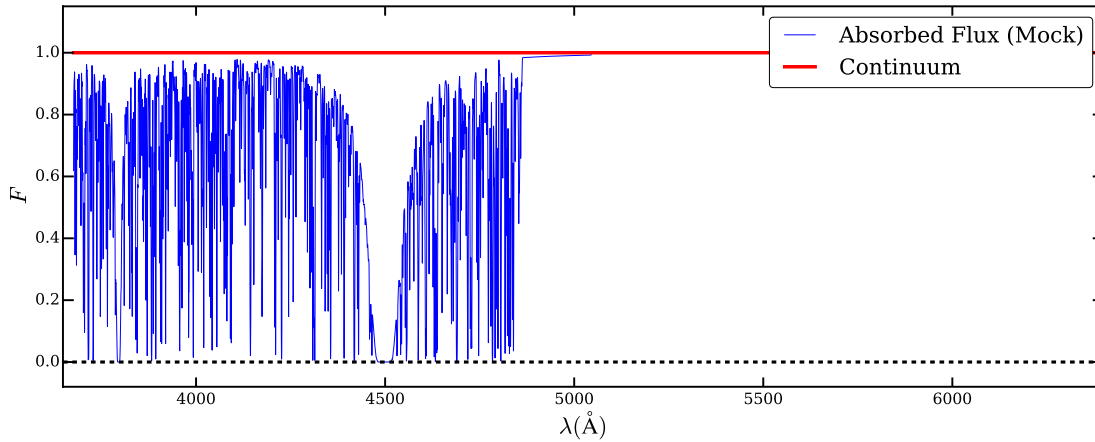


Figure 4: Example of mock spectrum with H I forest + DLA. Default parameter set for DLA in code are $\log N_{\text{HI}} = 21.0$, $b = 10 \text{ km s}^{-1}$ and $z_{\text{DLA}} = 2.7$.

2.3 Adding Metals to forest

- We can add metals to the mock spectra using cloudy interpolation tables (default).
To generate optical depth for metals we can use,

```
python weave_mocks.py flag_tau=metal
```

- Generating the optical depths for metals will take some time. We then generate the Ly series, DLA and metal flux,

```
python astro_mocks.py flag_absorber=HI_DLA_metal flag_make_quick_plot=y
```

Output:Quick_Plot_Mock_Spectra-0.pdf

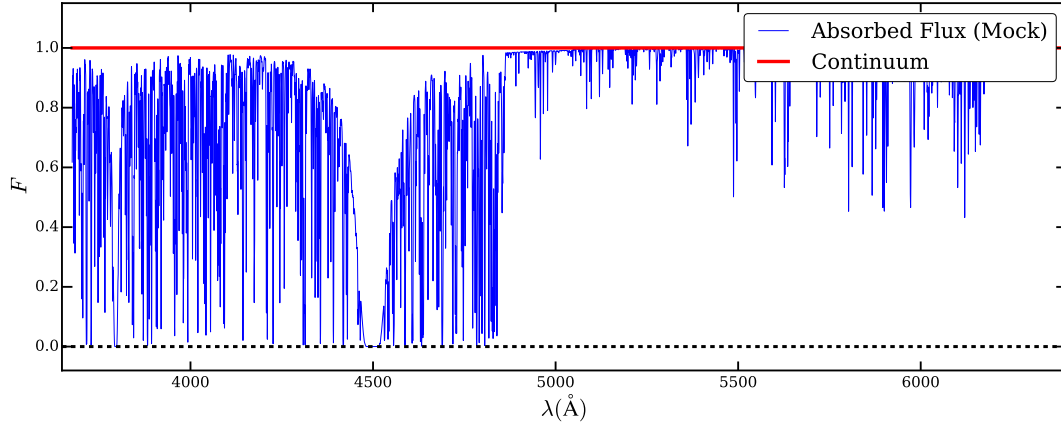


Figure 5: Example of mock spectrum H I ($\text{Ly}\alpha + \text{Ly}\beta$) forest + Metals + DLA. Default IGM metallicity is $\log(Z_{\text{IGM}}/Z_{\odot}) = -1.5$

2.4 Multiple options

- In previous section, we generated the optical depth for each systems individually like HI or DLA or metal.
- But it is possible to generate the optical depths for all systems simultaneously. For this one can set `flag_tau` variable to `HI_DLA_metal` or `HI_DLA` or `HI_metal` etc. For example `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 and DLA systems.
- Similar to `flag_tau`, one can also set multiple systems for `flag_absorber` variable. 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_make_quick_plot=y
```

- 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 The output would be similar to that shown in Fig. 5.

3 Automating ASTRO-MOCKS 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 ASTRO-MOCKS to generate large number of ($\sim 10^4 - 10^6$) mock spectra. In this section we describe how ASTRO-MOCKS can be used in script mode to generate spectra for WEAVE like survey. We provide two sample scripts to demonstrate how ASTRO-MOCKS can easily be used for generating large number of mock spectra. The file `astro_mocks_serial_multiple_spectra.py` shows an example of automating the ASTRO-MOCKS on single core while file `astro_mocks_parallel_mpi_multiple_spectra.py` shows an example of using ASTRO-MOCKS on multi-core machines.

3.1 Single core ASTRO-MOCKS automation

In this section, we describe the steps to automate ASTRO-MOCKS 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_astromocks.py` to see list of all variable names. For example, if we want to vary emission redshift of QSO, then the corresponding variable name is `z_em`.
2. ASTRO-MOCKS code uses one single main function `astromocks_function_calls_main`. This function has been imported in all `astro_mocks.py` files. We need to supply the

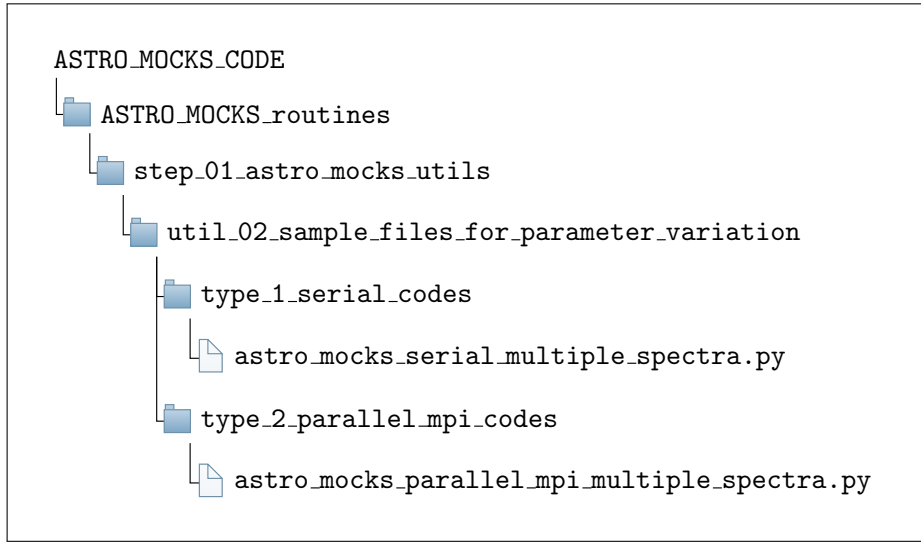


Figure 6: Location of sample codes to automate ASTRO-MOCKS for large number of spectra. The file `astro_mock_serial_multiple_spectra.py` shows an example of using single core machine while file `astro_mock_parallel_mpi_multiple_spectra.py` illustrates use of ASTRO-MOCKS on multi-core machine.

variable name users want to change to the function `astromocks_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_astromocks.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 ASTRO-MOCKS can generate multiple number of spectra.
5. We provide a more detailed example of automating ASTRO-MOCKS on single core machine in file `astro_mock_serial_multiple_spectra.py` (see Fig. 6).
6. Users can copy the file `astro_mock_serial_multiple_spectra.py` to `ASTRO MOCKS CODE` directory and run following command to test how ASTRO-MOCKS works

```
python astro_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 ASTRO-MOCKS to create unique `hdf5` group or `fits` extension name in the final output files.

3.2 Multi core ASTRO-MOCKS automation using MPI

Now we use the same example given above, and show how to automate ASTRO-MOCKS 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 ASTRO-MOCKS using parallel MPI libraries is as follows

1. *First we modify the `astro_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 ASTRO-MOCKS on multi core machine in file `astro_mock_parallel_mpi_multiple_spectra.py` (see Fig. 6).
3. We recommend reader to compare the content of file `astro_mock_parallel_mpi_multiple_spectra.py` with `astro_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 `astro_mock_parallel_mpi_multiple_spectra.py` to `ASTRO 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 astro_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, ASTRO-MOCKS code would try to distribute them as evenly as possible.

4 Reading ASTRO-MOCKS output

In this section we describe the basic format of ASTRO-MOCKS 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 ASTRO-MOCKS outputs (i.e., optical depths and fluxes) are saved in file strating with prefix ‘‘ASTRO MOCKS_Spectra’’. The variables `out_file_flux_base` and `out_file_tau_base` controls the prefix of the output files. Both variables currently set to `ASTRO MOCKS_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 chose different values of the two variables. When ASTRO-MOCKS 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`. ASTRO-MOCKS 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: ASTRO MOCKS Spectra.fits
```

```
For flag_mpi = 'y', rank = 5, flag_output_format = 'hdf5',
Output File Name: ASTRO MOCKS Spectra_Rank_5.hdf5
```

```

├─ COMMON_FIELDS -->> (Fits Extension or HDF5 Group)
│   ├── WavelengthForFlux -->> (BinaryHDU Table or HDF5 Dataset)
│   └── WavelengthForOpticalDepth -->> (BinaryHDU Table or HDF5 Dataset)
├─ MOCK_SPECTRA-0 -->> (Data for mock spectra with spec_id=0)
│   ├── Header -->> (All the variables used for this spectrum)
│   ├── FluxWithoutNoise -->> ASTRO-MOCKS spectrum (Flux without noise)
│   ├── OpticalDepth_HI -->> (Total Optical Depth for HI Ly-series Forest)
│   ├── OpticalDepth_DLA -->> (Optional, Total Optical Depth for HI Ly-series DLA)
│   └── OpticalDepth_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 7: **The figure shows basic structure of ASTRO-MOCKS 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 ASTRO-MOCKS. 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_astromocks_output_all_format.py` to read the ASTRO-MOCKS output files.

4.2 Data format

Fig. 7 shows the basic data structure of typical ASTRO-MOCKS output file (fits or hdf5). The ASTRO-MOCKS 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.
- **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 ASTRO-MOCKS for generating the spectrum. Since the parameters varies from one spectrum to another, Header is specified for every mock spectrum seperately to reflect changes. More details about Header format can found in §4.3.
- **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, ASTRO-MOCKS 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
INSTR	<code>instrument</code>	Instrument Name
RESMODE	<code>resolution_mode</code>	Resolution mode for WEAVE survey
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)
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 §5.3. 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, ASTRO-MOCKS 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.
- **Flux_Without_Noise:** (BinaryHDU Table or HDF5 Dataset, Array, Float32)
This table/dataset contains noiseless flux, the final product of ASTRO-MOCKS. The shape of this array is same as that of `Wavelength_For_Flux` 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 ASTRO-MOCKS output irrespective fits or hdf5 format.

4.4 Script to read ASTRO-MOCKS output

To facilitate users, we provide a python script to read the ASTRO-MOCKS output files. The location of this file is shown in Fig. 8. The file contains a function `read_astromocks_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_astro_mocks_output_all_format.py` that explains the argument list in details. This data can be access using following commands,

```
from path.to.location.read_astro_mocks_output_all_format import \
read_astromocks_output_fields

out_dict = read_astromocks_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']
```

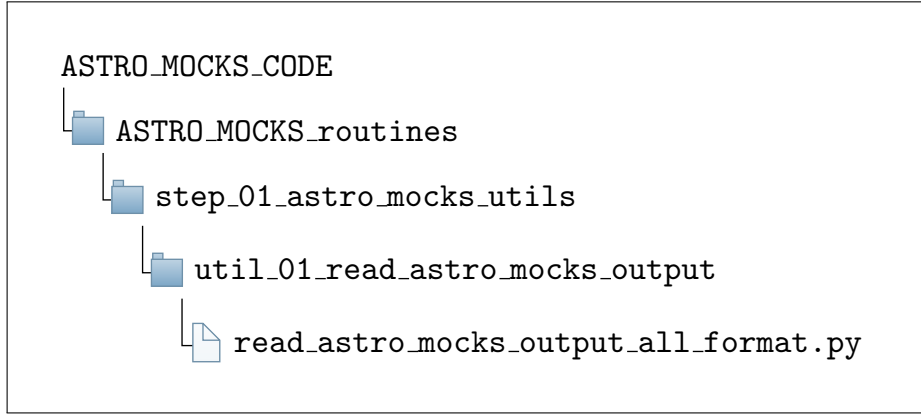


Figure 8: The location of sample python script to read the ASTRO-MOCKS output files.

5 Modifying ASTRO-MOCKS parameters

5.1 DLA parameters

- There are three parameters that set the properties of DLA in ASTRO-MOCKS, (i) log of H I column density ($\log_N_HI_DLA$ 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 `astro_mock.py` or supplied as arguments as follow

```
python astro_mock.py  flag_absorber=HI_metal_DLA  z_DLA=2.65
flag_tau=DLA log_N_HI_DLA=20.5 b_DLA=5.0
```

- Here is an another example of different DLA properties

```
python astro_mock.py  flag_absorber=HI_metal_DLA  z_DLA=2.7
flag_tau=DLA log_N_HI_DLA=21.5 b_DLA=5.0
```

5.2 Wavelength sampling parameters

- We provide wavelength sampling parameters in file `wavelength_sampling_parameters.txt` (see Fig. 9). The snippet of this file is shown in Listing 1
- 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 1: Snippet of `wavelength_sampling_parameters.txt` file

```
# This file contains wavelength sampling parameters for WEAVE
```

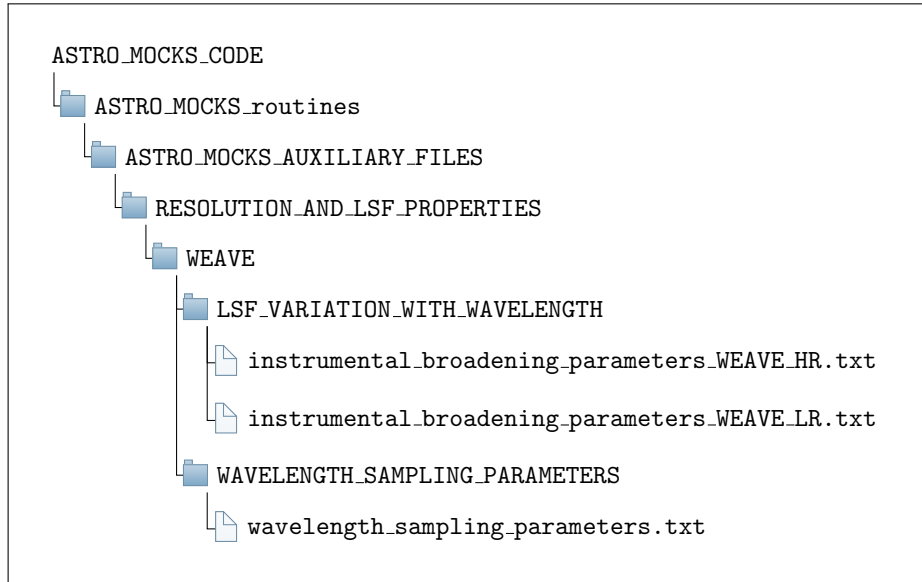



Figure 9: Location of instrumental broadening and wavelength sampling parameter files

```

# 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 -->> CDELTA1 (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 + CDELTA1 * np.arange(N_pixel_obs)
#
# if lambda_scale is LOG
# lambda_arr_obs = 10**(CRVAL1 + CDELTA1 * 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
# -----

```

5.3 Metal properties

There are three ways/options to add metals to mock spectra in ASTRO-MOCKS, (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 ASTRO-MOCKS. We are working on implementing other two options with WEAVE-QSO group members.

5.3.1 Method 1 : Cloudy Interpolation Tables

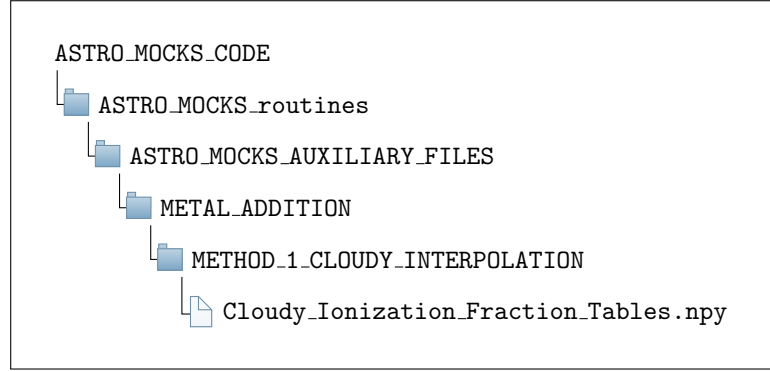


Figure 10: Location of cloudy interpolation tables.

- 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 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 `astro_mock.py` script as supplying an array as a command line argument not feasible.**
- On the other hand, if users need constant metallicity along sightline, there is a simple option included in ASTRO-MOCKS. User just need to specify variable value `log_Z_val_const=-2.0`. **This variable can be set either in `astro_mock.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. 10) In this table, ionization fraction of species 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 astro_mock.py  flag_absorber=HI_metal_DLA  flag_tau=metal
log_Z_val_const=-2.0  uvb_model_metal=FG20  flag_cont_method=PCA
```

- Another example of change in metallicity,

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

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.
- ASTRO-MOCKS uses <specie_name>-<transition_rest_wavelength> format to recognize the particular transition of a given specie. For example, H I Ly α transision is represented by "HI-1215" whereas C IV doublets are represnted by "CIV-1548" and "CIV-1550".
- A full list of all transition is given in file `specie_transition_properties.txt` (see Fig. 11 and Listing 2) ASTRO-MOCKS uses the information given in this file to compute mock spetrum.

```
ASTRO_MOCKS_CODE
├── ASTRO_MOCKS_routines
│   └── ASTRO_MOCKS_AUXILIARY_FILES
│       └── ABSOPRTION_TRANSITION_LIST
│           ├── specie_transition_properties.txt -->> (File specifying all transitions)
│           ├── atom.dat -->> (Refer to this file for addiing more transitions if needed)
│           └── Readme.txt
```

Figure 11: Location of file containing the list of transitions used in ASTRO-MOCKS.

Listing 2: 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
:	:	:	:	:	:
:	:	:	:	:	:
:	:	:	:	:	:
SiII-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 `astro_mocks.py`.
- For example, there are three main metal transition specified in `astro_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 `astro_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 `astro_mocks.py` file (by default) namely HI-1215 (H I $\text{Ly}\alpha$) and HI-1025 (H I $\text{Ly}\beta$). Suppose user wants add H I $\text{Ly}\gamma$ transition say HI-972. Then user need to change following line in `astro_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 ASTRO-MOCKS 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. 11)
- `atom.dat` file is taken from code VPFIT and contains list of (mostly) all known transition found in quasar absorption spectra. **Note that** ASTRO-MOCKS **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 `astro_mocks.py` file as discussed in previous section.

6 Detail description of free parameters in ASTRO-MOCKS

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

6.1 Cosmological mocks

Table 2: Cosmological mock fields parameters

ASTRO-MOCKS 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)

ASTRO-MOCKS 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 ASTRO-MOCKS 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 ASTRO-MOCKS 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 ASTRO-MOCKS 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

ASTRO-MOCKS 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

ASTRO-MOCKS 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 §5.3.

Table 5: Metal Addition Parameters

ASTRO-MOCKS Variable	Default Value	Description
<code>flag_metal_method</code>	<code>cloudy_interpolation</code>	Method to include metals in forest
<code>uvb_model_metal</code>	<code>FG20</code>	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>	<code>["SiIV-1402", "CIV-1548", "CIV-1550"]</code>	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.

Table 6: Wavelength sampling parameters

ASTRO-MOCKS Variable	Default Value	Description
<code>instrument</code>	<code>WEAVE</code>	Name of the instrument
<code>resolution_mode</code>	<code>LR</code>	Resolution mode HR or LR
<code>CRVAL1</code>	<code>None</code>	Start of wavelength array in angstroms
<code>CDEL1</code>	<code>None</code>	Wavelength spacing in angstroms
<code>N_pixel_obs</code>	<code>None</code>	Number of pixels in wavelength/flux array
<code>lambda_scale</code>	<code>None</code>	Wavelength spacing scale (LIN or LOG)

6.6 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_astromocks.py`. *Note: Cosmological parameters can only be changed by modifying the file `default_parameters_astromocks.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 7.

Table 7: Cosmological Parameters

ASTRO-MOCKS 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

6.7 MPI job submission parameters

The variables described in Table 8 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 ASTRO-MOCKS 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 8: MPI job submission Parameters

ASTRO-MOCKS 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.

6.8 Input / Output file saving parameters

The input / output directory and file names are set by variables as described in Table 9. 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. ASTRO-MOCKS provides a flexibility of saving the optical depths and fluxes in seperate 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 9: Parameters related to Input / Output

ASTRO-MOCKS 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>ASTRO_Mocks_Spectra</code>	Prefix of flux data file
<code>out_file_tau_base</code>	<code>ASTRO_Mocks_Spectra</code>	Prefix of optical depth data file. If OD needs to be saved in seperate 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

6.9 Miscellaneous parameters

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

7 Acknowledgement and Citations

8 Frequently Asked Questions (FAQs)

1. *I am running ASTRO-MOCKS 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. 12. Goto the above mentioned directory using cd command.

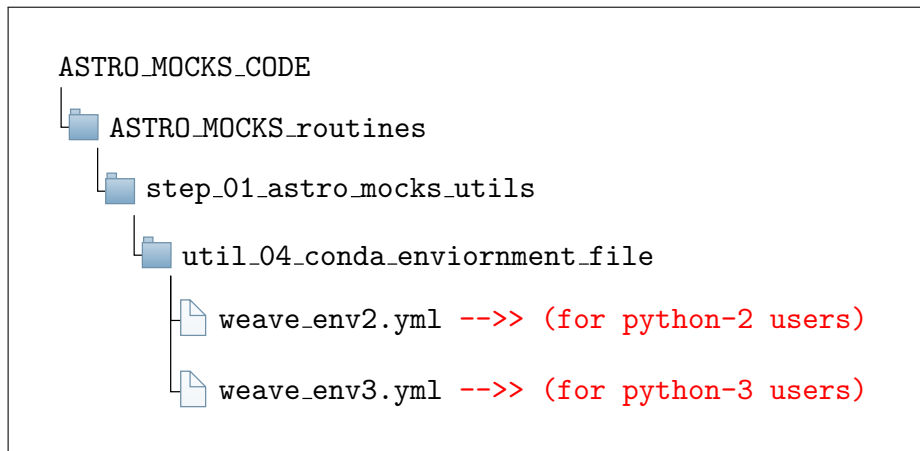


Figure 12: Location of conda environment files. The code has been tested for using all the packages mentioned in two yml files.

- (d) To create conda environment 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 associated packages
- (f) This will install all the packages needed to run ASTRO-MOCKS code.
- (g) To use the recently installed version of python, users need to activate conda environment 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.