atelier

Release 0.1a0

Jan-Torge Schindler

Aug 16, 2023

CONTENTS:

Version: 0.1a0

The "atelier" is my personal compilation of custom written software designed for my astrophysical research. Please feel welcome to use the python package and its modules. However, I would like to emphasize that the software is provided 'as is' and no warranty is given for its functionality (BSD 3-Clause license).

If you would like to contribute to the software please contact Jan-Torge Schindler via github.

CHAPTER

ONE

ATELIER.LUMFUN - INTRODUCTION

1.1 Introduction to the Atelier's luminosity function classwith a strong emphasis on quasar luminosity functions

[1]:
```
# General imports
import numpy as np
import matplotlib.pyplot as plt
from astropy.cosmology import FlatLambdaCDM

# Importing the luminosity function module
from atelier import lumfun
```

1.1.1 1) Introduction to the luminosity function object

1.2 1.1) Instantiating the luminosity function object

ADS reference: https://ui.adsabs.harvard.edu/abs/2013ApJ...768..105M/abstract The quasar luminosity function in the paper is defined using a broken double power law with the luminosity in magnitudes. An implementation of a magnitude based broken double power law luminosity function already exists in the *lumfun* module.

[2]:
```
print(lumfun.DoublePowerLawLF.__doc__)
```

```
Luminosity function, which takes the functional form of a double
   power law with the luminosity in absolute magnitudes.

   The luminosity function has four main parameters:

   - "phi_star": the overall normalization
   - "lum_star": the break luminosity/magnitude where the power law slopes
     change.
   - "alpha": the first power law slope
   - "beta": the second power law slope
```

The four main parameters need to be specified either as parameters or as functions. In this example only the source density normalization is a function, which depends on redshift (redsh), a z=6 normalized value of the source density *log_phi_star* and a parameter *k*.

[3]:
```
# Define the source density normalization as in McGreer+2013
def phi_star(redsh, log_phi_star, k):
    return 10 ** (log_phi_star + k * (redsh - 6))


log_phi_star = lumfun.Parameter(-8.94, 'log_phi_star', one_sigma_unc = [0.24, 0.20])
k = lumfun.Parameter(-0.47, 'k', vary=False)
```

We begin with defining the function for *phi_star* and its parameters. Redshift (redsh) and luminosity (lum) are always arguments of the luminosity function and do not require further definition. However, we need to define and provide *log_phi_star* and *k*.

For these parameters we use the **lumfun.Parameter** class which has the following attributes:

[4]:
```
print(lumfun.Parameter.__doc__)
```

```
 A class providing a data container for a parameter used in the
    luminosity function class.

    Attributes
    ----------
    value : float
        Value of the parameter
    name : string
        Name of the parameter
    bounds : tupler
        Bounds of the parameter, used in fitting
    vary : bool
        Boolean to indicate whether this parameter should be varied, used in
        fitting
    one_sigma_unc: list (2 elements)
        1 sigma uncertainty of the parameter.
```

In the next step we define the other three main parameters and instantiate the broken double power law luminosity function (*atelier.lumfun.DoublePowerLawLF*).

[5]:
```
lum_star = lumfun.Parameter(-27.21, 'lum_star', one_sigma_unc = [0.33, 0.27])
alpha = lumfun.Parameter(-2.03, 'alpha', vary=True, one_sigma_unc = [0.14, 0.15])

# The beta slope does not have an uncertainty as it set to -4 in McGreer+2013
beta = lumfun.Parameter(-4, 'beta', bounds=[-8, -3])

# Wrapping the parameters in a dictionary
parameters = {'log_phi_star': log_phi_star, 'k': k, 'lum_star': lum_star,
              'beta': beta, 'alpha': alpha}
# Wrapping the parameter functions (param_functions) in a dictionary
param_functions = {'phi_star': phi_star}

# The luminosity function type attribute is used in specific calculations based on the␣
↪luminosity function
# only lum_type = 'M1450' has a special meaning at the moment.
lum_type = 'M1450'
```

(continues on next page)

(continued from previous page)

```
mcgreer2013 = lumfun.DoublePowerLawLF(parameters, param_functions, lum_type=lum_type)
```

```
[INFO]------------------------------------------------
[INFO] Performing initialization checks
[INFO]------------------------------------------------
[INFO]------------------------------------------------
[INFO] Main parameter phi_star is described by a function.
[INFO] The function parameters are: ['redsh', 'log_phi_star', 'k']
[INFO] All parameters are supplied.
[INFO] Parameters "lum" and "redsh" were ignored as they are luminosity function␣
↪arguments.
[INFO]------------------------------------------------
[INFO] Main parameter lum_star is supplied as a normal parameter.
[INFO]------------------------------------------------
[INFO] Main parameter alpha is supplied as a normal parameter.
[INFO]------------------------------------------------
[INFO] Main parameter beta is supplied as a normal parameter.
[INFO]------------------------------------------------
[INFO] Initialization check passed.
[INFO]------------------------------------------------
```

Important With the instantiation of the luminosity function object a verbose check is carried out in which the code automatically checks if all *main parameters* have been either defined as parameters or as a function of parameters.

1.3 1.2) Basic calculations

1.4 1.2.1) Calculating the luminosity function

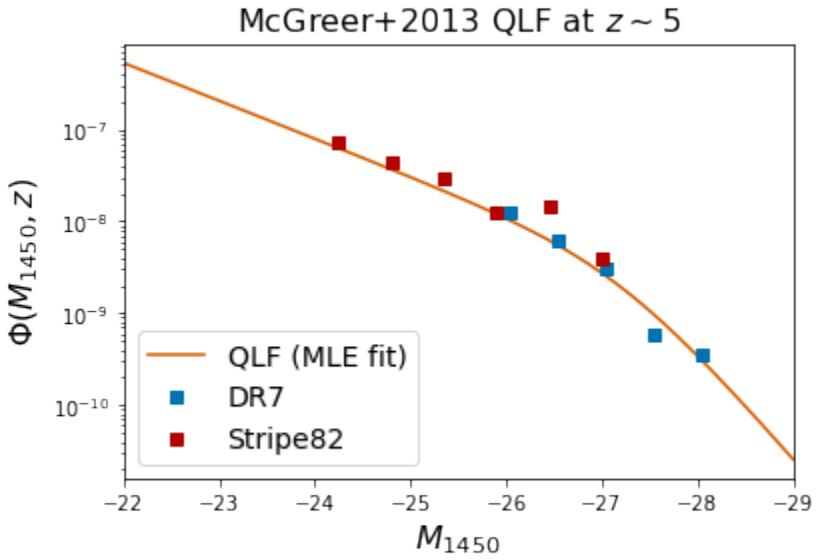
To retrieve values of the luminosity function one can either directly call it supplying a luminosity and redshift (e.g., *mcgreer2013(M1450,redsh)*) or use the *.evaluate(lum,redsh)* function. In both cases either the luminosity or the redshift can also be specified as a numpy.ndarray instead of a single float value.

[6]:
```
# Defining a magnitude array
M1450 = np.arange(-29, -22, 0.01)
# Defining a redshift array
z= np.arange(4, 6, 0.1)

# Calling the luminosity function with a luminosity and a redshift value produces one␣
↪output value
print(mcgreer2013(-27, 4.9))

# Calling the luminosity function with a np.ndarray in luminosity and a redshift value␣
↪produces a np.ndarray
print(mcgreer2013(M1450, 4.9).shape)

# Calling the luminosity function with a np.ndarray in redshift and a magnitude value␣
↪produces a np.ndarray
print(mcgreer2013(-27, z).shape)
```

```
2.7377613276931382e-09
(700,)
(20,)
```

Let us now display the McGreer+2013 quasar luminosity function in comparison to the binned values from the paper, basically reproducing Fig.16 of the publication.

[7]:
```
# Binned QLF data from the publication

dr7 = {'M1450':np.array([-28.05, -27.55, -27.05, -26.55, -26.05]),
       'logPhi':np.array([-9.45, -9.24, -8.51, -8.20, -7.9]),
       'simga_logPhi':np.array([0.21, 0.26, 0.58, 0.91, 1.89])*1e-9}

str82 = {'M1450':np.array([-27.0, -26.45, -25.9, -25.35, -24.8, -24.25]),
       'logPhi':np.array([-8.4, -7.84, -7.9, -7.53, -7.36, -7.14]),
       'simga_logPhi':np.array([2.81, 6.97, 5.92, 10.23, 11.51, 19.9])*1e-9}



# Color definition for plotting
vermillion = (213/255., 94/255., 0)
dblue = (0, 114/255., 178/255.)
dred = (179/255., 0, 0)

qlf = mcgreer2013(M1450, 4.9)

plt.plot(M1450, qlf, lw=1.5, color=vermillion, label='QLF (MLE fit)')
plt.plot(dr7['M1450'], 10**dr7['logPhi'], 's', color=dblue, label='DR7')
plt.plot(str82['M1450'], 10**str82['logPhi'], 's', color=dred, label='Stripe82')
plt.xlim(-22, -29)

# Make the plot nice
plt.semilogy()
plt.ylabel('$\Phi(M_{1450},z)$', fontsize=16)
plt.xlabel('$M_{1450}$', fontsize=16)
plt.title('McGreer+2013 QLF at $z\sim5$', fontsize=16)
plt.legend(loc=3, fontsize=14)
plt.show()
```



1.5 1.2.2) Calculating the number of quasars in a redshift and luminosity range (per steradian)

The built-in class function *.integrate_over_lum_redsh* performs the double integral of the quasar luminosity function using the provided input parameters. For this calculation we need to specify a cosmology as the differential comoving solid volume element, which can be provided directly, depends on it.

[8]:
```
print(mcgreer2013.integrate_over_lum_redsh.__doc__)
```

```
Calculate the number of sources described by the luminosity function
        over a luminosity and redshift interval in units of per steradian.

        Either a cosmology or dVdzdO have to be supplied.

        :param lum_range: Luminosity range
        :type lum_range: tuple
        :param redsh_range: Redshift range
        :type redsh_range: tuple
        :param dVdzdO: Differential comoving solid volume element (default =
            None)
        :type dVdzdO: function
        :param selfun: Selection function (default = None)
        :type selfun: atelier.selfun.QsoSelectionFunction
        :param cosmology: Cosmology (default = None)
        :type cosmology: astropy.cosmology.Cosmology
        :param kwargs:
        :return: :math:`N = \int\int\Phi(L,z) (dV/(dz d\Omega)) dL dz`
        :rtype: float
```

[11]:
```
cosmology = FlatLambdaCDM(H0=70, Om0=0.272)

n_qso_per_steradian = mcgreer2013.integrate_over_lum_redsh([-29,-24],[5,5.5],␣
↪cosmology=cosmology)
print('There are {:.5f} quasars per steradian within z=5-5.5 and with M1450=-29 to -24␣
↪(using romberg integration)'.format(n_qso_per_steradian))

n_qso_per_steradian = mcgreer2013.integrate_over_lum_redsh_simpson([-29,-24],[5,5.5],␣
↪cosmology=cosmology)
print('There are {:.5f} quasars per steradian within z=5-5.5 and with M1450=-29 to -24␣
↪(using Simpsons rule on a grid)'.format(n_qso_per_steradian))
```

```
There are 936.08722 quasars per steradian within z=5-5.5 and with M1450=-29 to -24␣
↪(using romberg integration)
There are 936.08637 quasars per steradian within z=5-5.5 and with M1450=-29 to -24␣
↪(using Simpsons rule on a grid)
```

1.6 1.2.3) Calculating the quasar number density over a luminosity range (per Mpc^3)

The built-in class function *.integrate_lum* performs this integral using the provided input parameters.

[10]:
```
print(mcgreer2013.integrate_lum.__doc__)
```

```
Calculate the volumetric source density described by the luminosity
        function at a given redshift and over a luminosity interval in units of
        per Mpc^3.

        :param redsh: Redshift
        :type redsh: float
        :param lum_range: Luminosity range
        :type lum_range: tuple
        :param kwargs:
        :return: :math:`\int \Phi(L,z) dL`
        :rtype: float
```

[11]:
```
n_qso_per_Mpc3 = mcgreer2013.integrate_lum(4.9,[-29,-24])
print('The quasar density (in Mpc^-3) over M1450=-29 to -24 is {:.2e} at redshift z=4.9'.
↪format(n_qso_per_Mpc3))
```

```
The quasar density (in Mpc^-3) over M1450=-29 to -24 is 8.53e-08 at redshift z=4.9
```

1.7 1.2.4) Calculating the quasar number density over a luminosity range (per steradian)

The built-in class function *.redshift_density* performs this integral using the provided input parameters. This quantity is rarely used in normal calculations based on the quasar luminosity function.

[12]:
```
print(mcgreer2013.redshift_density.__doc__)
```

```
Calculate the volumetric source density described by the luminosity
        function at a given redshift and over a luminosity interval in units of
        per steradian per redshift.

        :param redsh: Redshift
        :type redsh: float
        :param lum_range: Luminosity range
        :type lum_range: tuple
        :param dVdzdO: Differential comoving solid volume element (default =
            None)
        :type dVdzdO: function
        :param kwargs:
        :return: :math:`\int \Phi(L,z) (dV/dz){d\Omega} dL`
        :rtype: float
```

[13]:
```
dVdzdO = lumfun.interp_dVdzdO([4.0,6.0], cosmology)


n_qso_per_steradian_per_z = mcgreer2013.redshift_density(4.9,[-29,-24],dVdzdO)
print('The quasar density (per steradian and per redshift(!)) over M1450=-29 to -24 is {:
↪.2e} at redshift z=4.9'.format(n_qso_per_steradian_per_z))
```

```
The quasar density (per steradian and per redshift(!)) over M1450=-29 to -24 is 2.80e+03␣
↪at redshift z=4.9
```

1.7.1 1.3) Calculating the ionizing emissivity at 1450A (or 912A)

Another built-in class function, which is only defined for broken double power laws and single power laws (both based on magnitude) and for quasar luminosity functions with *lum_type = 'M1450'* is *.calc_ionizing_emissivity_at_1450A*. To properly test this function and compare it with published results on the ionizing emissivity we use the pre-defined quasar luminosity function (single power law parametrization) of Wang+2019.

ADS reference: https://ui.adsabs.harvard.edu/abs/2019ApJ...884...30W/abstract

[14]:
```
wang2019spl = lumfun.WangFeige2019SPLQLF()
```

```
[INFO]----------------------------------------------------
[INFO] Performing initialization checks
[INFO]----------------------------------------------------
[INFO]----------------------------------------------------
[INFO] Main parameter phi_star is supplied as a normal parameter.
[INFO]----------------------------------------------------
[INFO] Main parameter alpha is supplied as a normal parameter.
[INFO]----------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
[INFO] Main parameter lum_ref is supplied as a normal parameter.
[INFO]------------------------------------------------------
[INFO] Initialization check passed.
[INFO]------------------------------------------------------
```

We use the same integration boundaries (M1450=-30 to -28) at a redshift of z=6.7

[15]:
```
e1450 = wang2019spl.calc_ionizing_emissivity_at_1450A(6.7, [-30,-18])

# Adopting a power law slope of -0.6 we scale the ionizing emissivity at 1450A to 912A␣
↪(Lyman break)
e912 = e1450*(1450./912)**-0.6
print('Calculated ionizing emissivity at 1450A e1450={:.2e}'.format(e1450))
print('Rescaled ionizing emissivity to 912A e912={:.2e}'.format(e912))
print('Ionizing emissivity at 912A calculated in Wang2019 for the single power law QLF␣
↪e912=2.10e+23')
```

```
Calculated ionizing emissivity at 1450A e1450=2.77e+23
Rescaled ionizing emissivity to 912A e912=2.10e+23
Ionizing emissivity at 912A calculated in Wang2019 for the single power law QLF e912=2.
↪10e+23
```

1.7.2 2) Sampling from the luminosity function

A built-in class function allows to sample sources from a luminosity function. The sampling is done in a two step process. 1) The redshift range is divided into small bins and the number of expected sources per bin is calculated. 2) A cumulative distribution of the quasar luminosity function is calculated at the redshift bin middle and the number of expected sources for the redshift bin is taken. Sources for all bins are concatenated and returned.

Note! This sampling routine is in large part inspired by https://github.com/imcgreer/simqso/blob/master/simqso/lumfun.py , lines 219 and following.

[16]:
```
print(mcgreer2013.sample.__doc__)
```

```
Sample the luminosity function over a given luminosity and
            redshift range.

        This sampling routine is in large part adopten from
        https://github.com/imcgreer/simqso/blob/master/simqso/lumfun.py
        , lines 219 and following.

        If the integral over the luminosity function does not have an
        analytical implementation, integrals are calculated using
        integrate.romberg, which can take a substantial amount of time.

        :param lum_range: Luminosity range
        :type lum_range: tuple
        :param redsh_range: Redshift range
        :type redsh_range: tuple
        :param cosmology: Cosmology (default = None)
        :type cosmology: astropy.cosmology.Cosmology
        :param sky_area: Area of the sky to be sampled in square degrees
```

(continues on next page)

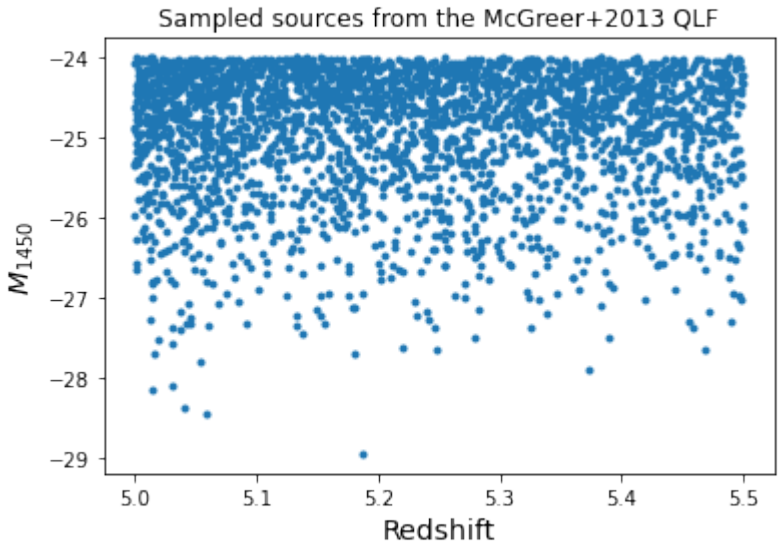
(continued from previous page)

```
        :type sky_area: float
        :param seed: Random seed for the sampling
        :type seed: int
        :param lum_res: Luminosity resolution (default = 1e-2, equivalent to
            100 bins)
        :type lum_res: float
        :param redsh_res: Redshift resolution (default = 1e-2, equivalent to
            100 bins)
        :type redsh_res: float
        :param verbose: Verbosity
        :type verbose: int
        :return: Source sample luminosities and redshifts
        :rtype: (numpy.ndarray,numpy.ndarray)
```

[17]:
```
Mrange = [-29, -24]
zrange = [5.0, 5.5]
sky_area = 10000

sample_M, sample_z = mcgreer2013.sample(Mrange, zrange, cosmology, sky_area)
```

```
[INFO] Integration returned 2851 sources
```

[18]:
```
plt.plot(sample_z, sample_M, '.')
plt.xlabel('Redshift', fontsize=14)
plt.ylabel('$M_{1450}$', fontsize=14)
plt.title('Sampled sources from the McGreer+2013 QLF')
plt.show()
```



[ ]:

CHAPTER

TWO

ATELIER.LUMFUNFIT - INTRODUCTION

2.1 Introduction to the Atelier's luminosity function fit classwith a strong emphasis on quasar luminosity functions

This notebook leads through an example of using the luminosity functio fit class to determine the free parameters of a luminosity function.

The luminosity function fit uses MCMC (emcee) to sample the free parameters space minimizing the negative logarithmic likelihood defined in Marshall et al. (1983) Equation 3.

ADS reference: https://ui.adsabs.harvard.edu/abs/1983ApJ...269...35M/abstract

[1]:
```
# General imports
import corner
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from astropy.cosmology import FlatLambdaCDM

# Importing the atelier modules
from atelier import lumfun
from atelier import lumfunfit
from atelier import survey
```

2.1.1 1) Setting up the mock data

We start by generating a mock sample of quasars adopting the McGreer+2018 quasar luminosity function at z~5. Using the lumfun class method ".sample" we generate quasars in 10000 square degrees over the defined luminosity and redshift range.

Note: The sample method relies on a large number of computational operations and has not been optimized, yet. So executing the function will take time.

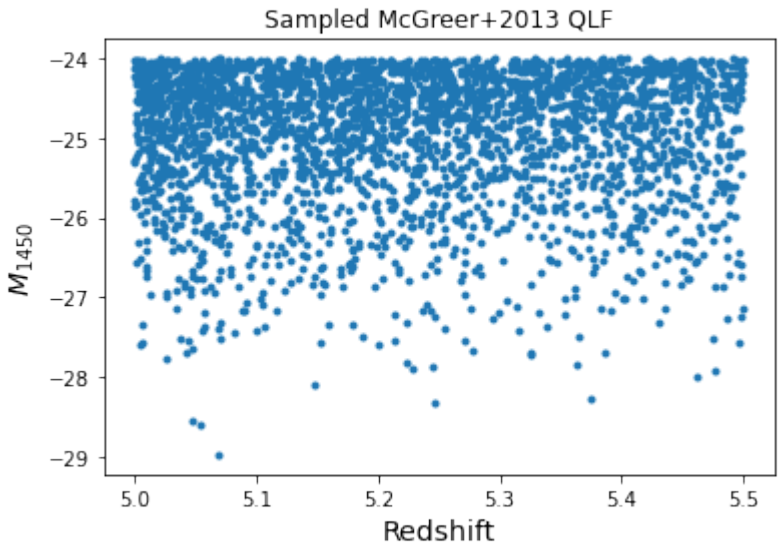
[2]:
```
cosmology = FlatLambdaCDM(H0=70, Om0=0.272)

lum_range = [-29, -24]
redsh_range = [5.0, 5.5]
sky_area = 10000

mcgreer2018 = lumfun.McGreer2018QLF()
```

(continues on next page)

(continued from previous page)

```
sample_M, sample_z = mcgreer2018.sample(lum_range, redsh_range, cosmology, sky_area)
mock_df = pd.DataFrame(data={'absmag':sample_M, 'redsh':sample_z})

plt.plot(sample_z, sample_M, '.')
plt.xlabel('Redshift', fontsize=14)
plt.ylabel('$M_{1450}$', fontsize=14)
plt.title('Sampled McGreer+2013 QLF')
plt.show()
```

```
[INFO]-----------------------------------------------------
[INFO] Performing initialization checks
[INFO]-----------------------------------------------------
[INFO]-----------------------------------------------------
[INFO] Main parameter phi_star is described by a function.
[INFO] The function parameters are: ['redsh', 'log_phi_star_z6', 'k', 'z_ref']
[INFO] All parameters are supplied.
[INFO] Parameters "lum" and "redsh" were ignored as they are luminosity function␣
↪arguments.
[INFO]-----------------------------------------------------
[INFO] Main parameter lum_star is supplied as a normal parameter.
[INFO]-----------------------------------------------------
[INFO] Main parameter alpha is supplied as a normal parameter.
[INFO]-----------------------------------------------------
[INFO] Main parameter beta is supplied as a normal parameter.
[INFO]-----------------------------------------------------
[INFO] Initialization check passed.
[INFO]-----------------------------------------------------
[INFO] Integration returned 3013 sources
```



2.1.2 2) Setting up the survey class

The mock quasar sample in M1450 and redshift will now be stored in the "Survey" class of the *survey* module. The "Survey" class is a container for important information of an astronomical survey such as the luminosity and redshift of it's sources and the overall survey area. As a keyword argument a selection function (see *selfun* module) can also be passed.

[3]:
```
mock_survey = survey.Survey(mock_df, 'absmag', 'redsh', 10000)
```

2.1.3 3) Fitting the mock survey to a luminosity function model

2.2 3.1) Instantiating the luminosity function fit class

We begin by instantiating a LuminosityFunctionFit object before. The luminosity and redshift range for the fit are specified at this instance along with the cosmology and the list of surveys, whose data should be considered in the fit. For this example we only pass the mock_survey object we generated above.

[4]:
```
fit = lumfunfit.LuminosityFunctionFit([-29, -24],[5.0,5.5], cosmology, [mock_survey])
```

2.3 3.2) Setting up the luminosity function model for the fit

In the next step we define the quasar luminosity function model used in the fit. Our goal is to reproduce the values of the McGreer+2018 quasar luminosity function. So we begin by instantiating a new version of it.

[5]:
```
lfmodel = lumfun.McGreer2018QLF()
lfmodel.print_parameters()
```

```
[INFO]------------------------------------------------------
[INFO] Performing initialization checks
[INFO]------------------------------------------------------
[INFO]------------------------------------------------------
[INFO] Main parameter phi_star is described by a function.
[INFO] The function parameters are: ['redsh', 'log_phi_star_z6', 'k', 'z_ref']
[INFO] All parameters are supplied.
[INFO] Parameters "lum" and "redsh" were ignored as they are luminosity function␣
↪arguments.
[INFO]------------------------------------------------------
[INFO] Main parameter lum_star is supplied as a normal parameter.
[INFO]------------------------------------------------------
[INFO] Main parameter alpha is supplied as a normal parameter.
[INFO]------------------------------------------------------
[INFO] Main parameter beta is supplied as a normal parameter.
[INFO]------------------------------------------------------
[INFO] Initialization check passed.
[INFO]------------------------------------------------------
Parameter log_phi_star_z6 = -8.97, bounds=None, vary=True, unc=[0.18, 0.15]
Parameter lum_star = -27.47, bounds=None, vary=True, unc=[0.26, 0.22]
Parameter alpha = -1.97, bounds=None, vary=True, unc=[0.09, 0.09]
Parameter beta = -4.0, bounds=None, vary=True, unc=None
```

(continues on next page)

(continued from previous page)

```
Parameter k = -0.47, bounds=None, vary=True, unc=None
Parameter z_ref = 6, bounds=None, vary=True, unc=None
```

However, we need to modify the luminosity function parameters for the luminosity function fit. We begin by excluding three parameters from the fitting, which were fixed in the original quasar luminosity function determination. Technically we should also be able to reproduce these parameters, but we want to keep the example simple.

Next, we set fitting bounds to the other three parameters and set their values to an arbitrary value within the fitting bounds for the initialization of the fit. Now we are ready to carry out the fit.

[6]:
```
# Set the parameters that are fixed in the model to not be varied during the fit.
lfmodel.parameters['z_ref'].vary=False
lfmodel.parameters['k'].vary=False
lfmodel.parameters['beta'].vary=False

# Set the parameter bounds for the fit parameters and set the starting value
lfmodel.parameters['log_phi_star_z6'].bounds = [-11,-7]
lfmodel.parameters['log_phi_star_z6'].value = -8
lfmodel.parameters['lum_star'].bounds = [-29.5, -25.5]
lfmodel.parameters['lum_star'].value = -28
lfmodel.parameters['alpha'].bounds = [-3, -1]
lfmodel.parameters['alpha'].value = -1.5

lfmodel.print_parameters()
```

```
Parameter log_phi_star_z6 = -8, bounds=[-11, -7], vary=True, unc=[0.18, 0.15]
Parameter lum_star = -28, bounds=[-29.5, -25.5], vary=True, unc=[0.26, 0.22]
Parameter alpha = -1.5, bounds=[-3, -1], vary=True, unc=[0.09, 0.09]
Parameter beta = -4.0, bounds=None, vary=False, unc=None
Parameter k = -0.47, bounds=None, vary=False, unc=None
Parameter z_ref = 6, bounds=None, vary=False, unc=None
```

2.4 3.3) Fitting the luminosity function model

To carry out the fit we call the *.run_mcmc* function supplying the luminosity function model and the number of steps for the MCMC chain. Default values for the MCMC are specified as attributes of the LuminosityFunctionFit class, are be set during initialization and can be modified at any time.

A bar indicates the progress of the MCMC sampling.

[ ]:
```
fit.run_mcmc(lfmodel,steps=5000)
```

The *sampler* attribute of the LuminosityFunctionFit class holds all the information on the emcee sampler including the chain of samples. We can visualize the result of the fit by using emcee's sampler function to return a flat chain from the MCMC.

Using the *corner* package we plot the results of the MCMC for the three fitted parameters highlighting the true input values of the McGreer+2018 QLF from which the mock sample was created.

[ ]:
```
# Plotting the covariance matrices of the final fit parameters

flat_samples = fit.sampler.get_chain(discard=100, thin=1, flat=True)
```

(continues on next page)

(continued from previous page)

```
print(flat_samples)
labels=['log_phi_star', 'lum_star', 'alpha']
fig = corner.corner(flat_samples, labels=labels, truths=[-8.97, -27.47, -1.97])
print(np.nanpercentile(flat_samples[:,0],[16,50,84]))
print(np.nanpercentile(flat_samples[:,1],[16,50,84]))
print(np.nanpercentile(flat_samples[:,2],[16,50,84]))
```

The fitting results encompass the true input values within 1 sigma, while being biased slightly higher in all three parameters. At present, we suspect the luminosity function method *.sample* to be at the root of this issue as changing the sampling resolution affects is know to bias the fitting results. However, investigations are still ongoing.

2.4.1 4) Fitting the mock survey to a luminosity function model (multiprocessing!)

Depending on the number of object in the survey and the complexity of the luminosity function model, the MCMC fitting can take a few minutes. To speed up the progress we have implemented a special version of the MCMC fitting, making use of emcee's native multiprocessing abilities.

The following cells provide a guide on how multiprocessing can be used to accelerate the fitting. The results initialized with the same default seed of the LuminosityFunctionFit class reproduce the result of the single core approach above.

[11]:
```
from multiprocessing import cpu_count

ncpu = cpu_count()
print("{0} CPUs".format(ncpu))
```

```
12 CPUs
```

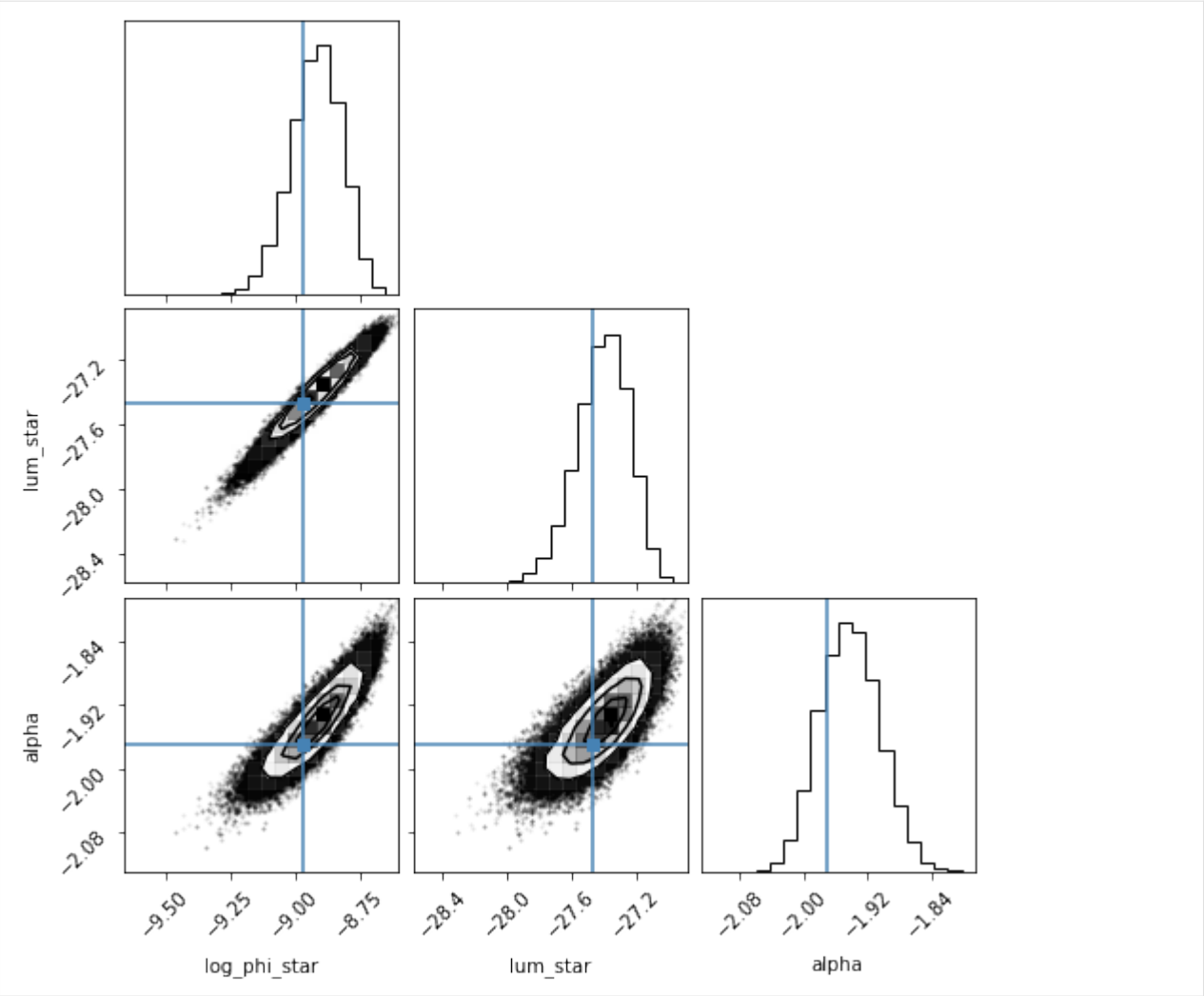
[12]:
```
# Reset the initial guess
initial_guess = [-8, -28, -1.5]
# Run the MCMC chain
fit.run_mcmc_multiprocess(lfmodel, steps=5000, initial_guess=initial_guess,
                          processes=6)
```

```
100%|| 5000/5000 [04:13<00:00, 19.72it/s]
```

```
[INFO] Multiprocessing took 253.6 seconds
```

[13]:
```
# Plotting the covariance matrices of the final fit parameters

flat_samples = fit.sampler.get_chain(discard=100, thin=1, flat=True)
labels=['log_phi_star', 'lum_star', 'alpha']
fig = corner.corner(flat_samples, labels=labels, truths=[-8.97, -27.47, -1.97])
print(np.nanpercentile(flat_samples[:,0],[16,50,84]))
print(np.nanpercentile(flat_samples[:,1],[16,50,84]))
print(np.nanpercentile(flat_samples[:,2],[16,50,84]))
```

```
[-9.01793936 -8.91553535 -8.82203563]
[-27.57611104 -27.40140614 -27.24798913]
[-1.97995996 -1.94293035 -1.90479171]
```



2.4.2 5) Fitting a luminosity function to multiple surveys

[ ]:
```
mock_survey1 = survey.Survey(mock_df[:int(mock_df.shape[0]/2)], 'absmag', 'redsh',
                             5000)
mock_survey2 = survey.Survey(mock_df[int(mock_df.shape[0]/2):], 'absmag', 'redsh',
                             5000)
```

[ ]:
```
fit2 = lumfunfit.LuminosityFunctionFit([-29, -24],[5.0,5.5], cosmology, [mock_survey1,
                                                                         mock_survey2])
```

[ ]:
```
initial_guess = [-8, -28, -1.5]
# Run the MCMC chain
fit2.run_mcmc(lfmodel, steps=5000, initial_guess=initial_guess)
```

[ ]:
```
# Plotting the covariance matrices of the final fit parameters

flat_samples = fit2.sampler.get_chain(discard=100, thin=1, flat=True)
```

(continues on next page)

(continued from previous page)

```
labels=['log_phi_star', 'lum_star', 'alpha']
fig = corner.corner(flat_samples, labels=labels, truths=[-8.97, -27.47, -1.97])
print(np.nanpercentile(flat_samples[:,0],[16,50,84]))
print(np.nanpercentile(flat_samples[:,1],[16,50,84]))
print(np.nanpercentile(flat_samples[:,2],[16,50,84]))
```

2.4.3 6) Fitting the mock survey to a luminosity function model (integration mode: "simpson")

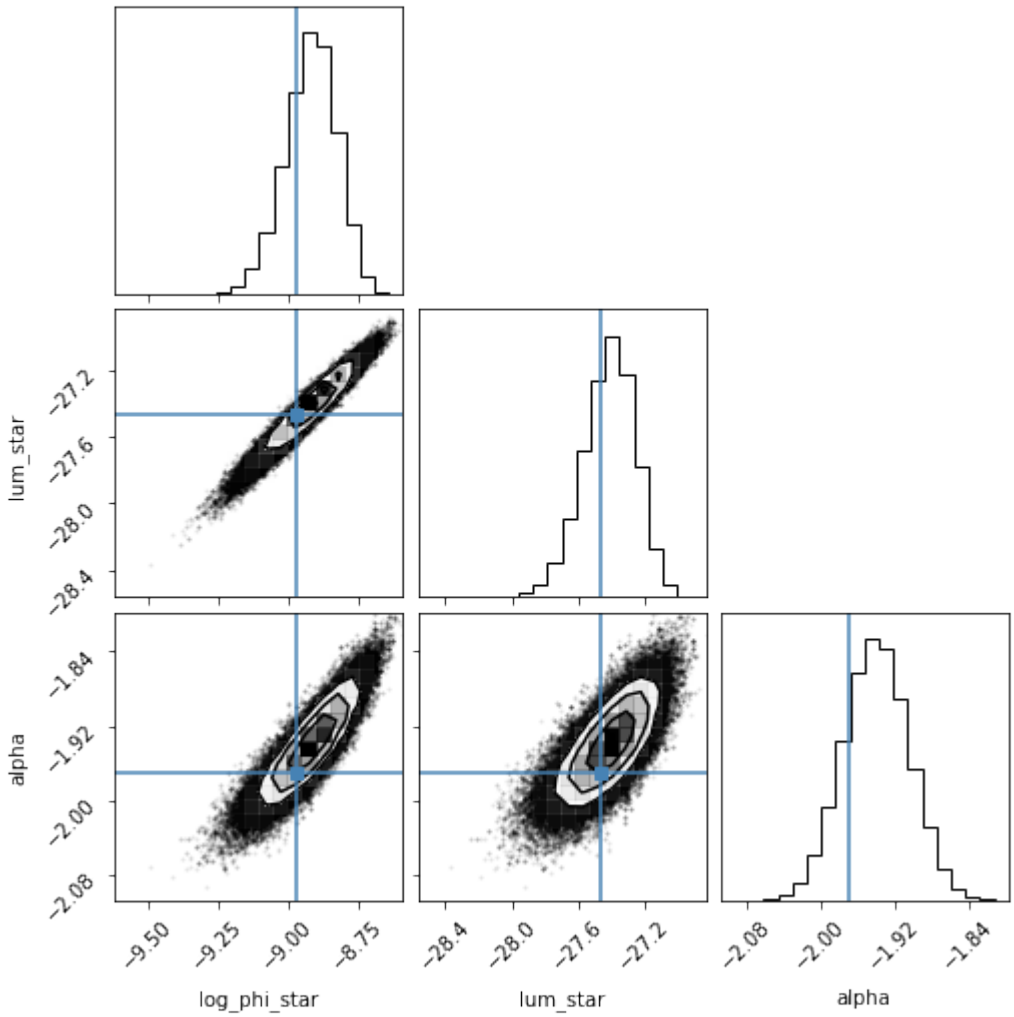
[7]:
```
fit = lumfunfit.LuminosityFunctionFit([-29, -24],[5.0,5.5], cosmology, [mock_survey])
```

[8]:
```
# Reset the initial guess
initial_guess = [-8, -28, -1.5]
# Run the MCMC chain
fit.run_mcmc(lfmodel, steps=5000, initial_guess=initial_guess, int_mode='simpson')
```

```
100%|| 5000/5000 [02:13<00:00, 37.38it/s]
```

[10]:
```
# Plotting the covariance matrices of the final fit parameters

flat_samples = fit.sampler.get_chain(discard=100, thin=1, flat=True)
# print(flat_samples)
labels=['log_phi_star', 'lum_star', 'alpha']
fig = corner.corner(flat_samples, labels=labels, truths=[-8.97, -27.47, -1.97])
print(np.nanpercentile(flat_samples[:,0],[16,50,84]))
print(np.nanpercentile(flat_samples[:,1],[16,50,84]))
print(np.nanpercentile(flat_samples[:,2],[16,50,84]))
```

```
[-9.01302699 -8.91593096 -8.82950664]
[-27.57408329 -27.40910229 -27.26364548]
[-1.97509396 -1.94016805 -1.90512569]
```


lum_star
alpha
−27.2
−27.6
−28.0
−28.4
−1.84
−1.92
−2.00
−2.08
−9.50
−9.25
−9.00
−8.75
log_phi_star
−28.4
−28.0
−27.6
−27.2
lum_star
−2.08
−2.00
−1.92
−1.84
alpha

[ ]:

CHAPTER

THREE

THE LUMINOSITY FUNCTION MODULE

class atelier.lumfun.**Akiyama2018QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Akiyama+2018 at z~4.

ADS reference: https://ui.adsabs.harvard.edu/abs/2018ApJ...869..150M/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

This implementation adopts the double power law fit presented in Table 5 ("Maximum Likelihood").

class atelier.lumfun.**Boutsia2021_QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Boutsia+2021 at z~3.9.

ADS reference: https://ui.adsabs.harvard.edu/abs/2021ApJ...912..111B/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

The fit in Boutsia+2021 includes QLF determinations of Fontanot+2007, Glikman+2011, Boutsia+2018, and Giallongo+2019 at fainter magnitudes.

This implementation adopts the double power law fit presented in Table 4.

class atelier.lumfun.**DoublePowerLawLF**(*parameters*, *param_functions*, *lum_type=None*, *cosmology=None*, *ref_cosmology=None*, *ref_redsh=None*, *verbose=1*)

Luminosity function, which takes the functional form of a double power law with the luminosity in absolute magnitudes.

The luminosity function has four main parameters:

- "phi_star": the overall normalization
- "lum_star": the break luminosity/magnitude where the power law slopes change.
- "alpha": the first power law slope
- "beta": the second power law slope

calc_ionizing_emissivity_at_1450A(*redsh*, *lum_range*, ***kwargs*)

Calculate the ionizing emissivity at rest-frame 1450A, ϵ_{1450}, in units of erg s^-1 Hz^-1 Mpc^-3.

This function integrates the luminosity function at redshift "redsh" over the luminosity interval "lum_range" to calculate the ionizing emissivity at rest-frame 1450A.

Calling this function is only valid if the luminosity function "lum_type" argument is "lum_type"="M1450".

Parameters

- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation

- **lum_range** (*tuple*) – Luminosity range

Returns
Ionizing emissivity (erg s^-1 Hz^-1 Mpc^-3)

Return type
float

evaluate(*lum*, *redsh*, *parameters=None*)

Evaluate the double power law as a function of magnitude ("lum") and redshift ("redsh").

Function to be evaluated: atelier.lumfun.mag_double_power_law()

Parameters

- **lum** (*float or numpy.ndarray*) – Luminosity for evaluation
- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **parameters** (*dict(atelier.lumfun.Parameters)*) – Dictionary of parameters used for this specific calculation. This does not replace the parameters with which the luminosity function was initialized. (default=None)

Returns
Luminosity function value

Return type
(numpy.ndarray,numpy.ndarray)

class atelier.lumfun.**Giallongo2019_4p5_QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Giallongo+2019 at z~4.5.

ADS reference: https://ui.adsabs.harvard.edu/abs/2019ApJ...884...19G/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

The fit in Giallongo+2019 (model 1) includes QLF determinations of Fontanot+2007, Boutsia+2018, and Akiyama+2018 at brighter magnitudes.

This implementation adopts the double power law fit presented in Table 3 (model 1).

class atelier.lumfun.**Giallongo2019_5p6_QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Giallongo+2019 at z~5.6.

ADS reference: https://ui.adsabs.harvard.edu/abs/2019ApJ...884...19G/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

The fit in Giallongo+2019 (model 4) includes data from CANDELS and SDSS.

This implementation adopts the double power law fit presented in Table 3 (model 4).

class atelier.lumfun.**Hopkins2007QLF**

Implementation of the bolometric quasar luminosity function of Hopkins+2007.

ADS reference: https://ui.adsabs.harvard.edu/abs/2007ApJ...654..731H/abstract

The luminosity function is described by Equations 6, 9, 10, 17, 19. The values for the best fit model adopted here are found in Table 3 in the row named "Full".

static alpha(*redsh*, *z_ref*, *gamma_one*, *kg1*)

Calculate the redshift dependent luminosity function slope alpha.

Equations 10, 17

Parameters

- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **z_ref** (*float*) – Reference redshift
- **gamma_one** (*float*) – Luminosity function power law slope at z_ref
- **kg1** (*float*) – Evolutionary parameter

Returns

static beta(*redsh*, *z_ref*, *gamma_two*, *kg2_1*, *kg2_2*)

Calculate the redshift dependent luminosity function slope beta.

Equations 10, 19 and text on page 744 (bottom right column)

Parameters

- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **z_ref** (*float*) – Reference redshift
- **gamma_two** (*float*) – Luminosity function power law slope at z_ref
- **kg2_1** (*float*) – Evolutionary parameter
- **kg2_2** (*float*) – Evolutionary parameter

Returns

evaluate(*lum*, *redsh*, *parameters=None*)

Evaluate the Hopkins+2007 bolometric luminosity function.

Function to be evaluated: atelier.lumfun.lum_double_power_law()

Parameters

- **lum** (*float or numpy.ndarray*) – Luminosity for evaluation
- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **parameters** (*dict(atelier.lumfun.Parameters)*) – Dictionary of parameters used for this specific calculation. This does not replace the parameters with which the luminosity function was initialized. (default=None)

Returns
Luminosity function value

Return type
(numpy.ndarray,numpy.ndarray)

static lum_star(*redsh*, *z_ref*, *log_lum_star*, *kl1*, *kl2*, *kl3*)

Calculate the redshift dependent break luminosity (Eq. 9, 10)

Parameters

- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **z_ref** (*float*) – Reference redshift
- **log_lum_star** (*float*) – Logarithmic break magnitude at z_ref

- **kl1** (*float*) – Function parameter kl1
- **kl2** (*float*) – Function parameter kl2
- **kl3** (*float*) – Function parameter kl3

Returns
Redshift dependent break luminosity

Return type
float

static phi_star(*log_phi_star*)

Calculate the break luminosity number density

Parameters
log_phi_star – Logarithmic break luminosity number density

Returns

class atelier.lumfun.**JiangLinhua2016QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Jiang+2016 at z~6.

ADS reference:

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

This implementation adopts the double power law fit described in Section 4.5

static phi_star(*redsh*, *phi_star_z6*, *k*, *z_ref*)

Calculate the redshift dependent luminosity function normalization.

Parameters

- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **phi_star_z6** (*float*) – Logarithmic source density at z=6
- **k** (*float*) – Power law exponent of density evolution
- **z_ref** (*float*) – Reference redshift

Returns

class atelier.lumfun.**Kim2020_QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Kim+2020 at z~5.

ADS reference: https://ui.adsabs.harvard.edu/abs/2020ApJ...904..111K/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

The fit in Kim+2020 includes data from Yang+2016 at brighter luminosities.

This implementation adopts the double power law fit presented in Table 6, Case 1.

static phi_star(*redsh*, *log_phi_star*)

Parameters

- **redsh** –
- **log_phi_star** –

Returns

class atelier.lumfun.**Kulkarni2019QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Kulkarni+2019 at z~1-6.

ADS reference: https://ui.adsabs.harvard.edu/abs/2019MNRAS.488.1035K/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

This implementation adopts the double power law fit presented in Table 3 ("Model 3").

static **alpha**(*redsh*, *c2_0*, *c2_1*)

Parameters

- **redsh** –
- **c2_0** –
- **c2_1** –

Returns

static **beta**(*redsh*, *c3_0*, *c3_1*)

Parameters

- **redsh** –
- **c3_0** –
- **c3_1** –

Returns

static **lum_star**(*redsh*, *c1_0*, *c1_1*, *c1_2*, *c1_3*)

Parameters

- **redsh** –
- **c1_0** –
- **c1_1** –
- **c1_2** –
- **c1_3** –

Returns

static **phi_star**(*redsh*, *c0_0*, *c0_1*, *c0_2*)

Parameters

- **redsh** –
- **c0_0** –
- **c0_1** –
- **c0_2** –

Returns

class atelier.lumfun.**LuminosityFunction**(*parameters*, *param_functions*, *main_parameters*, *lum_type=None*, *cosmology=None*, *ref_cosmology=None*, *ref_redsh=None*, *verbose=1*)

The base luminosity function class.

In this implementation a luminosity function is defined in terms of

- luminosity ("lum") and
- redshift ("redsh")
- a list of main parameters ("main_parameters")

The number of main parameters depend on the functional form and can themselves be functions ("param_functions") of luminosity, redshift or additional "parameters".

This general framework should facilitate the implementation of a wide range of luminosity functions without confining limits.

An automatic initialization will check whether the parameter functions and parameters define all necessary main parameters.

While the code does not distinguish between continuum luminosities, broad band luminosities or magnitudes, some inherited functionality is based on specific luminosity definitions. In order for these functions to work a luminosity type "lum_type" has to be specified. The following luminosity types have special functionality:

- "M1450" : Absolute continuum magnitude measured at 1450A in the rest-frame.

parameters

Dictionary of Parameter objects, which are used either as a main parameters for the calculation of the luminosity function or as an argument for calculating a main parameter using a specified parameter function "param_function".

Type
dict(*atelier.lumfun.Parameter*)

param_functions

Dictionary of functions with argument names for which the parameter attribute provides a Parameter or the luminosity "lum" or the redshift "redsh".

Type
dict(functions}

main_parameters

List of string providing names for the main parameters of the luminosity function. During the initialization the main parameters need to be either specified as a Parameter within the "parameters" attribute or as a function by the "param_functions" attribute.

Type
list(string)

lum_type

Luminosity type checked for specific functionality

Type
string (default=None)

verbose

Verbosity (0: no output, 1: minimal output)

Type
int

evaluate(*lum*, *redsh*, *parameters=None*)

Evaluate the luminosity function at the given luminosity and redshift.

:raise NotImplementedError

Parameters

- **lum** (*float or numpy.ndarray*) – Luminosity for evaluation
- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **parameters** (*dict(atelier.lumfun.Parameters)*) – Dictionary of parameters used for this specific calculation. This does not replace the parameters with which the luminosity function was initialized. (default=None)

evaluate_main_parameters(*lum*, *redsh*, *parameters=None*)

Evaluate the main parameters of the luminosity function

Parameters

- **lum** (*Luminosity for evaluation*) – float or numpy.ndarray
- **redsh** (*Redshift for evaluation*) – float or numpy.ndarray
- **parameters** (*dict(atelier.lumfun.Parameters)*) – Dictionary of parameters used for this specific calculation. This does not replace the parameters with which the luminosity function was initialized.

Returns

get_free_parameter_names()

Return a list of names with all parameters for which vary == True.

Returns
Names of parameters with vary == True

Return type
list(str)

get_free_parameters()

Return a dictionary with all parameters for which vary == True.

Returns
All parameters with vary == True

Return type
dict(*atelier.lumfun.Parameter*)

integrate_lum(*redsh*, *lum_range*, ***kwargs*)

Calculate the volumetric source density described by the luminosity function at a given redshift and over a luminosity interval in units of per Mpc^3.

Parameters

- **redsh** (*float*) – Redshift
- **lum_range** (*tuple*) – Luminosity range
- **kwargs** –

Returns
$\int \Phi(L, z) dL$

Return type
float

integrate_over_lum_redsh(*lum_range*, *redsh_range*, *dVdzdO=None*, *selfun=None*, *cosmology=None*, ***kwargs*)

Calculate the number of sources described by the luminosity function over a luminosity and redshift interval in units of per steradian.

Either a cosmology or dVdzdO have to be supplied.

Parameters

- **lum_range** (*tuple*) – Luminosity range
- **redsh_range** (*tuple*) – Redshift range
- **dVdzdO** (*function*) – Differential comoving solid volume element (default = None)
- **selfun** (atelier.selfun.QsoSelectionFunction) – Selection function (default = None)
- **cosmology** (*astropy.cosmology.Cosmology*) – Cosmology (default = None)
- **kwargs** –

Returns
$N = \int \int \Phi(L, z)(dV/(dzd\Omega))dLdz$

Return type
float

integrate_over_lum_redsh_appmag_limit(*lum_range*, *redsh_range*, *appmag_limit*, *kcorrection*, *dVdzdO=None*, *selfun=None*, *cosmology=None*, *initial_lum_bin_width=0.1*, *initial_redsh_bin_width=0.05*, *minimum_probability=0.001*, ***kwargs*)

Parameters

- **lum_range** –
- **redsh_range** –
- **appmag_limit** –
- **kcorrection** –
- **dVdzdO** –
- **selfun** –
- **cosmology** –
- **initial_lum_bin_width** –
- **initial_redsh_bin_width** –
- **minimum_probability** –
- **kwargs** –

Returns

integrate_over_lum_redsh_simpson(*lum_range*, *redsh_range*, *dVdzdO=None*, *selfun=None*, *cosmology=None*, *initial_lum_bin_width=0.1*, *initial_redsh_bin_width=0.05*, *minimum_probability=0.001*, ***kwargs*)

Calculate the number of sources described by the luminosity function over a luminosity and redshift interval in units of per steradian.

The integration is done on a grid using the Simpson rule.

This allows the selection function to be precalculated on the grid values for speed up of the integration process.

This code is in large part adopted from https://github.com/imcgreer/simqso/blob/master/simqso/lumfun.py lines 591 and following.

Either a cosmology or dVdzdO have to be supplied.

Parameters

- **lum_range** (*tuple*) – Luminosity range
- **redsh_range** (*tuple*) – Redshift range
- **dVdzdO** (*function*) – Differential comoving solid volume element (default = None)
- **selfun** (atelier.selfun.QsoSelectionFunction) – Selection function (default = None)
- **cosmology** (*astropy.cosmology.Cosmology*) – Cosmology (default = None)
- **kwargs** –

Returns
$N = \int \int \Phi(L, z)(dV/(dzd\Omega))dLdz$

Return type
float

integrate_to_luminosity_density(*lum_range*, *redsh*, ***kwargs*)

Parameters

- **lum_range** –
- **redsh** –
- **dVdzdO** –

Returns
$\int \Phi(L, z)L(dV/dz)d\Omega dL$

print_free_parameters()

Print a list of all free (vary==True) parameters.

print_parameters()

Print a list of all parameters.

redshift_density(*redsh*, *lum_range*, *dVdzdO*, ***kwargs*)

Calculate the volumetric source density described by the luminosity function at a given redshift and over a luminosity interval in units of per steradian per redshift.

Parameters

- **redsh** (*float*) – Redshift
- **lum_range** (*tuple*) – Luminosity range
- **dVdzdO** (*function*) – Differential comoving solid volume element (default = None)
- **kwargs** –

Returns
$\int \Phi(L, z)(dV/dz)d\Omega dL$

Return type
float

sample(*lum_range*, *redsh_range*, *cosmology*, *sky_area*, *seed=1234*, *lum_res=0.01*, *redsh_res=0.01*, *verbose=1*, ***kwargs*)

Sample the luminosity function over a given luminosity and
redshift range.

This sampling routine is in large part adopted from https://github.com/imcgreer/simqso/blob/master/simqso/lumfun.py , lines 219 and following.

If the integral over the luminosity function does not have an analytical implementation, integrals are calculated using integrate.romberg, which can take a substantial amount of time.

Parameters

- **lum_range** (*tuple*) – Luminosity range
- **redsh_range** (*tuple*) – Redshift range
- **cosmology** (*astropy.cosmology.Cosmology*) – Cosmology (default = None)
- **sky_area** (*float*) – Area of the sky to be sampled in square degrees
- **seed** (*int*) – Random seed for the sampling
- **lum_res** (*float*) – Luminosity resolution (default = 1e-2, equivalent to 100 bins)
- **redsh_res** (*float*) – Redshift resolution (default = 1e-2, equivalent to 100 bins)
- **verbose** (*int*) – Verbosity

Returns
Source sample luminosities and redshifts

Return type
(numpy.ndarray,numpy.ndarray)

update()

Update the lumfun class parameters after a manual input.

Returns

update_free_parameter_values(*values*)

Update all free parameters with new values.

Parameters
values (*list(float)*) – Values in the same order as the order of free parameters.

class atelier.lumfun.**Matsuoka2018QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Matsuoka+2018 at z~6.

ADS reference: https://ui.adsabs.harvard.edu/abs/2018ApJ...869..150M/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

This implementation adopts the double power law fit presented in Table 5 (“standard”).

static phi_star(*redsh*, *phi_star_z6*, *k*, *z_ref*)

Calculate the redshift dependent luminosity function normalization.

Parameters

- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation

- **phi_star_z6** (*float*) – Logarithmic source density at z=6
- **k** (*float*) – Power law exponent of density evolution
- **z_ref** (*float*) – Reference redshift

Returns

class atelier.lumfun.**Matsuoka2023QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Matsuoka+2023 at z~7.

ADS reference: https://ui.adsabs.harvard.edu/abs/2023arXiv230511225M/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

This implementation adopts the double power law fit presented in Table 7 (standard model).

static **phi_star**(*redsh*, *phi_star_z7*, *k*, *z_ref*)

Calculate the redshift dependent luminosity function normalization.

Parameters

- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **phi_star_z7** (*float*) – Source density at z=6
- **k** (*float*) – Power law exponent of density evolution
- **z_ref** (*float*) – Reference redshift

Returns

class atelier.lumfun.**McGreer2018QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of McGreer+2018 at z~5 (z=4.9).

ADS reference: https://ui.adsabs.harvard.edu/abs/2018AJ....155..131M/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

This implementation adopts the best maximum likelihood estimate fit from the second column in Table 2.

static **phi_star**(*redsh*, *log_phi_star_z6*, *k*, *z_ref*)

Calculate the redshift dependent luminosity function normalization.

Parameters

- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **log_phi_star_z6** (*float*) – Logarithmic source density at z=6
- **k** (*float*) – Power law exponent of density evolution
- **z_ref** (*float*) – Reference redshift

Returns

class atelier.lumfun.**Niida2020_QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Niida+2020 at z~5.

ADS reference: https://ui.adsabs.harvard.edu/abs/2020ApJ...904...89N/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

The fit in Kim+2020 includes data from SDSS at brighter luminosities.

This implementation adopts the double power law fit presented in Table 6, Case 1.

class atelier.lumfun.**Onken2022_Niida_4p52_QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Onken+2022 at z~4.52.

ADS reference: https://ui.adsabs.harvard.edu/abs/2021arXiv210512215O/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

The maximum likelihood fit in Onken+2022 includes constraints of Niida+2020 at fainter magnitudes.

This implementation adopts the double power law fit presented in Table 4.

class atelier.lumfun.**Onken2022_Niida_4p83_QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Onken+2022 at z~4.83.

ADS reference: https://ui.adsabs.harvard.edu/abs/2021arXiv210512215O/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

The maximum likelihood fit in Onken+2022 includes constraints of Niida+2020 at fainter magnitudes.

This implementation adopts the double power law fit presented in Table 4.

class atelier.lumfun.**PanZhiwei2022_3p8_QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Zhiwei Pan+2022 at z~3.8.

ADS reference: https://ui.adsabs.harvard.edu/abs/2022ApJ...928..172P/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

This implementation adopts the double power law fit presented in Table 5 denoted as (O+S+L) "Best fit".

static phi_star(*redsh*, *log_phi_star*)

Calculate the redshift dependent luminosity function normalization.

Parameters

- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **log_phi_star** (*float*) – Logarithmic source density at z=6

Returns

class atelier.lumfun.**PanZhiwei2022_4p25_QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Zhiwei Pan+2022 at z~4.25.

ADS reference: https://ui.adsabs.harvard.edu/abs/2022ApJ...928..172P/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

This implementation adopts the double power law fit presented in Table 5 denoted as (O+S+L) "Best fit".

static phi_star(*redsh*, *log_phi_star*)

Calculate the redshift dependent luminosity function normalization.

Parameters

- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **log_phi_star** (*float*) – Logarithmic source density at z=6

Returns

class atelier.lumfun.**PanZhiwei2022_4p7_QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Zhiwei Pan+2022 at z~4.7.

ADS reference: https://ui.adsabs.harvard.edu/abs/2022ApJ...928..172P/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

This implementation adopts the double power law fit presented in Table 5 denoted as (O+S+L) "Best fit".

static phi_star(*redsh*, *log_phi_star*)

Calculate the redshift dependent luminosity function normalization.

Parameters

- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **log_phi_star** (*float*) – Logarithmic source density at z=6

Returns

class atelier.lumfun.**Parameter**(*value*, *name*, *bounds=None*, *vary=True*, *one_sigma_unc=None*)

A class providing a data container for a parameter used in the luminosity function class.

value

Value of the parameter

Type
float

name

Name of the parameter

Type
string

bounds

Bounds of the parameter, used in fitting

Type
tupler

vary

Boolean to indicate whether this parameter should be varied, used in fitting

Type
bool

one_sigma_unc

1 sigma uncertainty of the parameter.

Type
list (2 elements)

class atelier.lumfun.**Richards2006QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Richards+2006 (z=1-5).

ADS reference:

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes Mi(z=2. We convert the Mi(z=2) magnitudes to M1450 using a simple conversion factor (see Ross+2013).

M1450(z=0) = Mi(z=2) + 1.486

This implementation adopts the double power law fit presented in Table 7 (variable power law).

evaluate(*lum*, *redsh*, *parameters=None*)

Evaluate the single power law as a function of magnitude ("lum") and redshift ("redsh") for the Richards 2006 QLF.

Function to be evaluated: atelier.lumfun.richards_single_power_law()

Parameters

- **lum** (*float or numpy.ndarray*) – Luminosity for evaluation
- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **parameters** (*dict(atelier.lumfun.Parameters)*) – Dictionary of parameters used for this specific calculation. This does not replace the parameters with which the luminosity function was initialized. (default=None)

Returns
Luminosity function value

Return type
(numpy.ndarray,numpy.ndarray)

static phi_star(*redsh*, *log_phi_star*)

Parameters

- **redsh** –
- **log_phi_star** –

Returns

class atelier.lumfun.**SchechterLF**(*parameters*, *param_functions*, *lum_type=None*, *ref_cosmology=None*, *ref_redsh=None*, *cosmology=None*, *verbose=1*)

Schechter luminosity function

evaluate(*lum*, *redsh*, *parameters=None*)

Evaluate the Schechter function as a function of magnitude ("lum") and redshift ("redsh").

Function to be evaluated: atelier.lumfun.mag_schechter

Parameters

- **lum** (*float or numpy.ndarray*) – Luminosity for evaluation
- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **parameters** (*dict(atelier.lumfun.Parameters)*) – Dictionary of parameters used for this specific calculation. This does not replace the parameters with which the luminosity function was initialized. (default=None)

Returns
Luminosity function value

Return type
(numpy.ndarray,numpy.ndarray)

class atelier.lumfun.**Schindler2019_2p9_QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Schindler+2019 at z~2.9.

ADS reference: https://ui.adsabs.harvard.edu/abs/2019ApJ...871..258S/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

The maximum likelihood fit in Schindler+2019 includes the quasar sample of Ross+2013 at fainter magnitudes.

This implementation adopts the double power law fit presented in Table 7.

class atelier.lumfun.**Schindler2019_3p25_QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Schindler+2019 at z~3.25.

ADS reference: https://ui.adsabs.harvard.edu/abs/2019ApJ...871..258S/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

The maximum likelihood fit in Schindler+2019 includes the quasar sample of Ross+2013 at fainter magnitudes.

This implementation adopts the double power law fit presented in Table 7.

class atelier.lumfun.**Schindler2019_3p75_QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Schindler+2019 at z~3.75.

ADS reference: https://ui.adsabs.harvard.edu/abs/2019ApJ...871..258S/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

The maximum likelihood fit in Schindler+2019 includes the quasar sample of Richards+2096 at fainter magnitudes.

This implementation adopts the double power law fit presented in Table 7.

class atelier.lumfun.**Schindler2019_4p25_QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Schindler+2019 at z~4.25.

ADS reference: https://ui.adsabs.harvard.edu/abs/2019ApJ...871..258S/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

The maximum likelihood fit in Schindler+2019 includes the quasar sample of Richards+2096 at fainter magnitudes.

This implementation adopts the double power law fit presented in Table 7.

class atelier.lumfun.**Schindler2019_LEDE_QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Schindler+2019 with LEDE evolution.

ADS reference: https://ui.adsabs.harvard.edu/abs/2019ApJ...871..258S/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

The LEDE model was fit to a range of binned QLF measurements using chi-squared minimzation.

This implementation adopts the LEDE double power law fit presented in Table 8.

static lum_star(*redsh*, *lum_star_z2p2*, *c2*)

Calculate the redshift dependent break magnitude.

Parameters

- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **lum_star_z2p2** (*float*) – Break magnitude at z=2.2
- **c2** (*float*) – Redshift evolution parameter

Returns

static phi_star(*redsh*, *log_phi_star_z2p2*, *c1*)

Calculate the redshift dependent luminosity function normalization.

Parameters

- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **log_phi_star_z2p2** (*float*) – Logarithmic source density at z=2.2
- **c1** (*float*) – Redshift evolution parameter

Returns

class atelier.lumfun.**Schindler2023QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Schindler+2023 at z~6.

ADS reference: https://ui.adsabs.harvard.edu/abs/2023ApJ...943...67S/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

This implementation adopts the double power law fit presented in Table 4 (first row).

static phi_star(*redsh*, *log_phi_star_z6*, *k*, *z_ref*)

Calculate the redshift dependent luminosity function normalization.

Parameters

- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **phi_star_z6** (*float*) – Logarithmic source density at z=6
- **k** (*float*) – Power law exponent of density evolution
- **z_ref** (*float*) – Reference redshift

Returns

class atelier.lumfun.**ShenXuejian2020QLF**

Shen+2020 bolometric quasar luminosity function; global fit B

static alpha(*redsh*, *a0*, *a1*, *z_ref*)

Parameters

- **redsh** –
- **a0** –
- **a1** –
- **z_ref** –

Returns

static **beta**(*redsh*, *b0*, *b1*, *b2*, *z_ref*)

Parameters

- **redsh** –
- **b0** –
- **b1** –
- **b2** –
- **z_ref** –

Returns

evaluate(*lum*, *redsh*, *parameters=None*)

Evaluate the Shen+2020 bolometric luminosity function.

Function to be evaluated: atelier.lumfun.lum_double_power_law()

Parameters

- **lum** (*float or numpy.ndarray*) – Luminosity for evaluation
- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **parameters** (*dict(atelier.lumfun.Parameters)*) – Dictionary of parameters used for this specific calculation. This does not replace the parameters with which the luminosity function was initialized. (default=None)

Returns
Luminosity function value

Return type
(numpy.ndarray,numpy.ndarray)

static **lum_star**(*redsh*, *c0*, *c1*, *c2*, *z_ref*)

Parameters

- **redsh** –
- **c0** –
- **c1** –
- **c2** –
- **z_ref** –

Returns

class atelier.lumfun.**SinglePowerLawLF**(*parameters*, *param_functions*, *lum_type=None*, *ref_cosmology=None*, *ref_redsh=None*, *cosmology=None*, *verbose=1*)

Luminosity function, which takes the functional form of a single power law with the luminosity in absolute magnitudes.

The luminosity function has three main parameters:

- "phi_star": the overall normalization
- "alpha": the first power law slope
- "lum_ref": the break luminosity/magnitude where the power law slopes change.

calc_ionizing_emissivity_at_1450A(*redsh*, *lum_range*, ***kwargs*)

Calculate the ionizing emissivity at rest-frame 1450A, ϵ_{1450}, in units of erg s^-1 Hz^-1 Mpc^-3.

This function integrates the luminosity function at redshift "redsh" over the luminosity interval "lum_range" to calculate the ionizing emissivity at rest-frame 1450A.

Calling this function is only valid if the luminosity function "lum_type" argument is "lum_type"="M1450".

Parameters

- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **lum_range** (*tuple*) – Luminosity range

Returns
Ionizing emissivity (erg s^-1 Hz^-1 Mpc^-3)

Return type
float

evaluate(*lum*, *redsh*, *parameters=None*)

Evaluate the single power law as a function of magnitude ("lum") and redshift ("redsh").

Function to be evaluated: atelier.lumfun.mag_single_power_law()

Parameters

- **lum** (*float or numpy.ndarray*) – Luminosity for evaluation
- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **parameters** (*dict(atelier.lumfun.Parameters)*) – Dictionary of parameters used for this specific calculation. This does not replace the parameters with which the luminosity function was initialized. (default=None)

Returns
Luminosity function value

Return type
(numpy.ndarray,numpy.ndarray)

class atelier.lumfun.**SmoothDoublePowerLawLF**(*parameters*, *param_functions*, *lum_type=None*, *cosmology=None*, *ref_cosmology=None*, *ref_redsh=None*, *verbose=1*)

Luminosity function, which takes the functional form of a double power law with the luminosity in absolute magnitudes.

The luminosity function has four main parameters:

- "phi_star": the overall normalization
- "lum_star": the break luminosity/magnitude where the power law slopes change.
- "alpha": the first power law slope
- "beta": the second power law slope
- "delta": the smoothing parameter

calc_ionizing_emissivity_at_1450A(*redsh*, *lum_range*, ***kwargs*)

Calculate the ionizing emissivity at rest-frame 1450A, ϵ_{1450}, in units of erg s^-1 Hz^-1 Mpc^-3.

This function integrates the luminosity function at redshift "redsh" over the luminosity interval "lum_range" to calculate the ionizing emissivity at rest-frame 1450A.

Calling this function is only valid if the luminosity function "lum_type" argument is "lum_type"="M1450".

Parameters

- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **lum_range** (*tuple*) – Luminosity range

Returns
Ionizing emissivity (erg s^-1 Hz^-1 Mpc^-3)

Return type
float

evaluate(*lum*, *redsh*, *parameters=None*)

Evaluate the double power law as a function of magnitude (“lum”) and redshift (“redsh”).

Function to be evaluated: atelier.lumfun.mag_double_power_law()

Parameters

- **lum** (*float or numpy.ndarray*) – Luminosity for evaluation
- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **parameters** (*dict(atelier.lumfun.Parameters)*) – Dictionary of parameters used for this specific calculation. This does not replace the parameters with which the luminosity function was initialized. (default=None)

Returns
Luminosity function value

Return type
(numpy.ndarray,numpy.ndarray)

class atelier.lumfun.**WangFeige2019DPLQLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Wang+2019 at z~6.7.

ADS reference: https://ui.adsabs.harvard.edu/abs/2019ApJ...884...30W/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

This implementation adopts the double power law fit described in Section 5.5

static phi_star(*redsh*, *phi_star_z6p7*, *k*, *z_ref*)

Calculate the redshift dependent luminosity function normalization.

Parameters

- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **phi_star_z6p7** (*float*) – Logarithmic source density at z=6
- **k** (*float*) – Power law exponent of density evolution
- **z_ref** (*float*) – Reference redshift

Returns

class atelier.lumfun.**WangFeige2019SPLQLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Wang+2019 at z~6.7.

ADS reference: https://ui.adsabs.harvard.edu/abs/2019ApJ...884...30W/abstract

The luminosity function is parameterized as a single power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

This implementation adopts the single power law fit described in Section 5.5

class atelier.lumfun.**Willott2010QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Willott+2010 at z~6.

ADS reference: https://ui.adsabs.harvard.edu/abs/2010AJ....139..906W/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

This implementation adopts the double power law fit presented in Section 5.2.

static phi_star(*redsh*, *phi_star_z6*, *k*, *z_ref*)

Calculate the redshift dependent luminosity function normalization.

Parameters

- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **phi_star_z6** (*float*) – Logarithmic source density at z=6
- **k** (*float*) – Power law exponent of density evolution
- **z_ref** (*float*) – Reference redshift

Returns

class atelier.lumfun.**YangJinyi2016QLF**(*cosmology=None*)

Implementation of the type-I quasar UV(M1450) luminosity function of Yang+2016 at z~5.

ADS reference: https://ui.adsabs.harvard.edu/abs/2016ApJ...829...33Y/abstract

The luminosity function is parameterized as a double power law with the luminosity variable in absolute magnitudes at 1450A, M1450.

The maximum likelihood fit in Yang+2016 includes the quasar samples of McGreer+2013 at fainter magnitudes (i.e., the SDSS DR7 and Stripe 82 samples).

This implementation adopts the double power law fit presented in Section 5.2.

static phi_star(*redsh*, *log_phi_star_z6*, *k*, *z_ref*)

Calculate the redshift dependent luminosity function normalization.

Parameters

- **redsh** (*float or numpy.ndarray*) – Redshift for evaluation
- **phi_star_z6** (*float*) – Logarithmic source density at z=6
- **k** (*float*) – Power law exponent of density evolution
- **z_ref** (*float*) – Reference redshift

Returns

atelier.lumfun.**interp_dVdzdO**(*redsh_range*, *cosmo*)

Interpolate the differential comoving solid volume element $(dV/dz)d\Omega$ over the specified redshift range zrange = (z_1, z_2).

This interpolation speeds up volume (redshift, solid angle) integrations for the luminosity function without significant loss in accuracy.

The resolution of the redshift array, which will be interpolated is $\Delta z = 0.025$.

Parameters

- **redsh_range** (*tuple*) – Redshift range for interpolation
- **cosmo** (*astropy.cosmology.Cosmology*) – Cosmology

Returns
1D interpolation function

atelier.lumfun.**lum_double_power_law**(*lum*, *phi_star*, *lum_star*, *alpha*, *beta*)

Evaluate a broken double power law luminosity function as a function of luminosity.

Parameters

- **lum** (*float or np.ndarray*) – Luminosity
- **phi_star** (*float*) – Normalization of the broken power law at a value of lum_star
- **lum_star** (*float*) – Break luminosity of the power law
- **alpha** (*float*) – First slope of the broken power law
- **beta** (*float*) – Second slope of the broken power law

Returns
Value of the broken double power law at a magnitude of M

Return type
float or np.ndarray

atelier.lumfun.**mag_double_power_law**(*mag*, *phi_star*, *mag_star*, *alpha*, *beta*)

Evaluate a broken double power law luminosity function as a function of magnitude.

Parameters

- **mag** (*float or np.ndarray*) – Magnitude
- **phi_star** (*float*) – Normalization of the broken power law at a value of mag_star
- **mag_star** (*float*) – Break magnitude of the power law
- **alpha** (*float*) – First slope of the broken power law
- **beta** (*float*) – Second slope of the broken power law

Returns
Value of the broken double power law at a magnitude of M

Return type
float or np.ndarray

atelier.lumfun.**mag_schechter_function**(*mag*, *phi_star*, *mag_star*, *alpha*)

Evaluate a Schechter luminosity function as a function of magnitude.

Parameters

- **mag** –
- **phi_star** –
- **mag_star** –
- **alpha** –

Returns

atelier.lumfun.**mag_single_power_law**(*mag*, *phi_star*, *mag_ref*, *alpha*)

Evaluate a power law luminosity function as a function as a function of magnitude

Parameters

- **mag** (*float or np.ndarray*) – Magnitude

- **phi_star** (*float*) – Normalization of the power law at a value of mag_ref
- **mag_ref** (*float*) – Reference magnitude of the power law
- **alpha** (*float*) – Slope of the power law

Returns
Value of the broken double power law at a magnitude of M

Return type
float or np.ndarray

atelier.lumfun.**mag_smooth_double_power_law**(*mag*, *phi_star*, *mag_star*, *alpha*, *beta*, *log_delta*)

Evaluate a smooth broken double power law luminosity function as a function of magnitude.

param mag
Magnitude

type mag
float or np.ndarray

param phi_star
Normalization of the broken power law at a value of mag_star

type phi_star
float

param mag_star
Break magnitude of the power law

type mag_star
float

param alpha
First slope of the broken power law

type alpha
float

param beta
Second slope of the broken power law

type beta
float

param delta
Smoothness parameter

type delta
float

return
Value of the broken double power law at a magnitude of M

rtype
float or np.ndarray

atelier.lumfun.**richards_single_power_law**(*mag*, *phi_star*, *mag_ref*, *alpha*)

Parameters

- **mag** –
- **phi_star** –

- **mag_ref** –
- **alpha** –

Returns

CHAPTER

FOUR

THE SELECTION FUNCTION MODULE

class atelier.selfun.**ClippedFunction**(*fun*, *minval*, *maxval*)

A helper class to clip output between two values.

class atelier.selfun.**CompositeQsoSelectionFunction**(*selfun_list*, *mag_range=None*, *redsh_range=None*)

Composite quasar selection function class

This class provides the capabilities for composite quasar selection functions and updates the base class evaluate and call methods accordingly.

selfun_list

List of quasar selection functions

Type
list(*atelier.selfun.QsoSelectionFunction*)

mag_range

Magnitude range over which the quasar selection function is evaluated.

Type
(tuple)

redsh_range

Redshift range over which the quasar selection function is evaluated.

Type
(tuple)

evaluate(*mag*, *redsh*)

Evaluate the composite selection function at magnitude and redshift.

Parameters

- **mag** (*float or np.ndarray*) – Magnitude at which the quasar selection function is evaluated.
- **redsh** (*float or np.ndarray*) – Redshift at which the quasar selection function is evaluated.

Returns
Value of selection function at given redshift and magnitude.

Return type
float or np.ndarray

class atelier.selfun.**QsoSelectionFunction**(*mag_range=None*, *redsh_range=None*)

Quasar selection function class.

This class provides the basic capabilities for quasar selection functions.

mag_range

Magnitude range over which the quasar selection function is evaluated.

Type
(tuple)

redsh_range

Redshift range over which the quasar selection function is evaluated.

Type
(tuple)

evaluate(*mag*, *redsh*)

Evaluate selection function at magnitude and redshift.

Parameters

- **mag** (*float or np.ndarray*) – Magnitude at which the quasar selection function is evaluated.
- **redsh** (*float or np.ndarray*) – Redshift at which the quasar selection function is evaluated.

Returns
Value of selection function at given redshift and magnitude.

Return type
float or np.ndarray

plot_selfun(*mag_res=0.01, redsh_res=0.01, mag_range=None, redsh_range=None, title=None, levels=[0.2, 0.5, 0.7], level_color='k', cmap=<matplotlib.colors.ListedColormap object>, vmin=0, vmax=1, sample_mag=None, sample_z=None, sample_color='red', sample_mec='k', sample_marker='D', save_name=None*)

Plot the selection function on a grid of redshift and magnitude.

To calculate the map of the selection function the selection function is evaluated using the .evaluate method.

If no magnitude and redshift range is provided the class attributes are used to determine the ranges.

Parameters

- **mag_res** (*float*) – Magnitude resolution (default = 0.01)
- **redsh_res** (*float*) – Redshift resolution (default = 0.01)
- **mag_range** (*tuple*) – Magnitude range over which the quasar selection function is evaluated. (default = None)
- **redsh_range** (*tuple*) – Redshift range over which the quasar selection function is evaluated. (default = None)
- **title** (*string*) – Title of the plot (default = None)
- **levels** (*list(float)*) – Values of contour levels to plot
- **level_color** (*string*) – Color for the level contours
- **cmap** (*Matplotlib colormap*) – Color map for the selection function
- **save_name** (*string*) – Populate with name to save the plot. Default='None'

Returns

class atelier.selfun.**QsoSelectionFunctionConst**(*value*)

Composite quasar selection function class

This class provides the capabilities for the constant quasar selection function and updates the base class evaluate and call methods accordingly.

value

Value of the constant quasar selection function

Type
float

evaluate(*mag*, *redsh*)

Evaluate the constant selection function at magnitude and redshift.

Parameters
redsh (*float or np.ndarray*) – Redshift at which the quasar selection function is evaluated.

Returns
Value of selection function at given redshift and magnitude.

Returns
Value of selection function at given redshift and magnitude.

Return type
float or np.ndarray

class atelier.selfun.**QsoSelectionFunctionGrid**(*mag_range=None*, *redsh_range=None*, *mag_bins=None*, *redsh_bins=None*, *selfungrid=None*, *clip=True*, *filename=None*, *format='csv'*)

Gridded quasar selection function class.

This class provides the capabilities for a quasar selection function calculated on a magnitude redshift grid of sources defined by a query of the source properties. It updates the base class evaluate and call methods accordingly.

mag_range

Magnitude range over which the quasar selection function is evaluated.

Type
(tuple)

redsh_range

Redshift range over which the quasar selection function is evaluated.

Type
(tuple)

mag_bins

Number of bins in magnitude

Type
(int)

redsh_bins

Number of bins in redshift

Type
(int)

selfungrid

Grid with selection function values

Type
(np.ndarray)

clip

Boolean to indicate whether to clip the output value of the selection function between 0 and 1.

Type
(bool)

calc_selfungrid_from_df(*df*, *mag_col_name*, *redshift_col_name*, *query=None*, *sel_idx=None*, *n_per_bin=None*, *verbose=1*)

Calculate the selection function grid from a pandas DataFrame

Parameters

- **df** (*pandas.DataFrame*) – Data frame to calculate the selection function from
- **query** (*string*) – A query describing the subsample that is compared to the full data set in order to calculate the selection function.
- **mag_col_name** (*string*) – Name of the magnitude column in the data frame.
- **redshift_col_name** (*string*) – Name of the redshift column in the data frame.
- **n_per_bin** (*int*) – Number of sources per bin (default = None). This keyword argument is used to check whether the redshift and magnitude ranges and the number of bins per dimension produce a uniform distribution of sources per bin.
- **verbose** (*int*) – Verbosity

Returns
None

evaluate(*mag*, *redsh*)

Evaluate the interpolation of the selection function grid at magnitude and redshift.

Parameters
redsh (*float or np.ndarray*) – Redshift at which the quasar selection function is evaluated.

Returns
Value of selection function at given redshift and magnitude.

Returns
Value of selection function at given redshift and magnitude.

Return type
float or np.ndarray

get_selfun_from_grid()

Calculate an interpolation (RectBivariateSpline) of the selection function grid to allow evaluations of the selection function at any point within the magnitude and redshift range.

This methods need to be called before the selection function can be evaluated.

Returns
None

load(*filename*, *format='csv'*)

Load a QsoSelectionFunctionGrid from a file.

Parameters

- **filename** (*string*) – Filename to load a saved QsoSelectionFunctionGrid from.
- **format** – File format of saved QsoSelectionFunctionGrid.

Currently implemented formats are: ".csv" :type format: string :return:

plot_grid(*title=None*)

Plot the selection function grid as a map of magnitude and redshift.

Parameters
title (*string*) – Title of the plot

Returns
None

save(*filename*, *format='csv'*)

Save a QsoSelectionFunctionGrid object to a file

Parameters

- **filename** (*string*) – Filename to save the QsoSelectionFunctionGrid to.
- **format** – File format of saved QsoSelectionFunctionGrid.

Currently implemented formats are: "csv" :type format: string :return:

CHAPTER

FIVE

THE SURVEY MODULE

class atelier.survey.**Survey**(*obj_df*, *lum_colname*, *redsh_colname*, *sky_area*, *selection_function=None*, *lum_range=None*, *redsh_range=None*, *conf_interval='poisson'*)

Survey class is a container to hold information on the sources of a particular astronomical survey.

It is used primarily to forward observational data in the calculation of luminosity function fits.

obj_df

Data frame with information on astronomical sources in the survey.

Type
pandas.DataFrame

lum_colname

The name of the data frame column holding the luminosity (absolute magnitude) information of the sources in the survey.

Type
string

redsh_colname

The name of the data frame column holding the redshift information of the sources in the survey.

Type
string

sky_area

Sky area the survey covers in square degrees

Type
float

selection_function

Selection function of the survey (default = None).

Type
lumfun.selfun.QsoSelectionFunction

calc_binned_lumfun_PC2000(*lum_edges*, *redsh_edges*, *cosmology*, *kcorrection=None*, *app_mag_lim=None*, ***kwargs*)

Calculation of the binned luminosity function based on the method outlined in Page & Carrera 2000

ADS Link: https://ui.adsabs.harvard.edu/abs/2000MNRAS.311..433P/abstract

This function is very similar to the other method for calculating the binned luminosity function below.

Parameters

- **lum_edges** –
- **redsh_edges** –
- **cosmology** –
- **kcorrection** –
- **app_mag_lim** –

Returns

atelier.survey.**return_poisson_confidence**(*n*, *bound_low=0.15865*, *bound_upp=0.84135*)

Return the Poisson confidence interval boundaries for the lower and upper bound given a number of events n.

The default values for ther lower and upper bounds are equivalent to the 1 sigma confidence interval of a normal distribution.

Parameters

- **n** –
- **n_sigma** –

Returns

CHAPTER

SIX

THE LUMINOSITY FUNCTION FIT MODULE

class atelier.lumfunfit.**LuminosityFunctionFit**(*lum_range*, *redsh_range*, *cosmology*, *surveys*, *emcee_nwalkers=50*, *emcee_steps=1000*)

Class that carries out maximum likelihood fits to a luminosity function model using MCMC (emcee).

lum_range

Luminosity range for the luminosity function fit (2-element list).

Type
list(float, float)

redsh_range

Redshift range for the luminosity function fit (2-element list).

Type
list(float, float)

cosmology

Cosmology object

Type
astropy.cosmology.Cosmology

surveys

List of survey objects used in to fit the luminosity function model to.

Type
list(*survey.Survey*)

emcee_sampler

Emcee Ensemble sampler object. This attribute cannot be initialized but is internally populated during the .run_mcmc or .run_mcmc_multiprocess methods.

Type
emcee.EnsembleSampler

emcee_nwalker

Number of walkers for the emcee sampler.

Type
int (default = 50)

emcee_steps

Number of steps for the emcee sampler.

Type
int (default = 1000)

log_prior(*theta*, *parameters*)

Logarithmic prior function for luminosity function maximum likelihood estimation (MLE).

Parameters

- **theta** (*list of parameter values*) – parameter vector
- **parameters** (*dict(atelier.lumfun.Parameter)*) – Dictionary of parameters used in MLE. The Parameter.bounds attribute is used to keep the fit within the pre-defined parameter boundaries (bounds).

Returns
Logarithmic prior

Return type
float

log_probability(*theta*, *lumfun=None*, *use_prior=True*, *int_mode='romberg'*)

Logarithmic probability for luminosity function maximum likelihood estimation (MLE).

The logarithmic probability implemented here goes back to the definition of Marshall+1983 Equation 3.

ADS reference: https://ui.adsabs.harvard.edu/abs/1983ApJ...269...35M/abstract

Parameters

- **theta** (*list of parameter values*) – parameter vector
- **lumfun** (atelier.lumfun.LuminosityFunction *(or children classes)*) – Luminosity function model

Returns
Logarithmic probability

Return type
float

negative_log_likelihood(*theta*, *lumfun*)

Return the negative logarithmic likelihood for a set of parameters 'theta' and a luminosity function model 'lumfun' using the default logarithmic likelihood definition.

Parameters

- **theta** (*list of parameter values*) – parameter vector
- **lumfun** (atelier.lumfun.LuminosityFunction *(or children classes)*) – Luminosity function model

Returns
Negative logarithmic likelihood

Return type
float

run_mcmc(*lumfun*, *initial_guess=None*, *nwalkers=None*, *steps=None*, *int_mode='romberg'*)

Run the emcee MCMC EnsembleSampler to fit the luminosity function model to the survey data using maximum likelihood estimation.

Parameters

- **lumfun** (atelier.lumfun.LuminosityFunction *(or children classes)*) – Luminosity function model
- **initial_guess** (*list(float) (default = None)*) – Initial guess of the free parameters (ordered as in the luminosity function model.

- **nwalkers** (*int (default = None)*) – Number of walkers for the emcee sampler overriding (not overwriting) the class attribute emcee_nwalkers.
- **steps** (*int (default = None)*) – Number of steps for the emcee sampler overriding (not overwriting) the class attribute emcee_nwalkers.

Returns
None

run_mcmc_multiprocess(*lumfun*, *initial_guess=None*, *nwalkers=None*, *steps=None*, *processes=None*, *int_mode='romberg'*)

Run the emcee MCMC EnsembleSampler to fit the luminosity function model to the survey data using maximum likelihood estimation in multiprocessing mode.

Parameters

- **lumfun** (atelier.lumfun.LuminosityFunction *(or children classes)*) – Luminosity function model
- **initial_guess** (*list(float) (default = None)*) – Initial guess of the free parameters (ordered as in the luminosity function model.
- **nwalkers** (*int (default = None)*) – Number of walkers for the emcee sampler overriding (not overwriting) the class attribute emcee_nwalkers.
- **steps** (*int (default = None)*) – Number of steps for the emcee sampler overriding (not overwriting) the class attribute emcee_nwalkers.
- **processes** (*int (default = None)*) – Number of processes for multiprocessing. The default value of 'None' will use the maximum number determinded by the multiprocessing package.

Returns

atelier.lumfunfit.**log_prior**(*theta*, *parameters*)

Logarithmic prior function for luminosity function maximum likelihood
estimation (MLE) using emcee in multiprocessing mode.

Parameters

- **theta** (*list of parameter values*) – parameter vector
- **parameters** (*dict(atelier.lumfun.Parameter)*) – Dictionary of parameters used in MLE. The Parameter.bounds attribute is used to keep the fit within the pre-defined parameter boundaries (bounds).

Returns
Logarithmic prior

Return type
float

atelier.lumfunfit.**log_probability**(*theta*, *lumfun=None*, *lum_range=None*, *redsh_range=None*, *surveys=None*, *dVdzdO=None*, *log_prior=None*, *use_prior=True*, *int_mode='romberg'*)

Logarithmic probability for luminosity function maximum likelihood estimation (MLE) using emcee in multiprocessing mode.

The logarithmic probability implemented here goes back to the definition of Marshall+1983 Equation 3.

ADS reference: https://ui.adsabs.harvard.edu/abs/1983ApJ...269...35M/abstract

Parameters

- **theta** (*list of parameter values*) – parameter vector
- **lumfun** (atelier.lumfun.LuminosityFunction *(or children classes)*) – Luminosity function model
- **lum_range** (*tuple*) – Luminosity range
- **redsh_range** (*tuple*) – Redshift range
- **surveys** (*list(*atelier.survey.Survey*)*) – List of surveys
- **dVdzd0** (*function*) – Differential comoving solid volume element
- **log_prior** – Functiond describing the logarithmic prior
- **use_prior** (*bool*) – Boolean to indicate whether the logarithmic prior function will be used or strictly flat priors for all parameters will be adopted.

Returns
Logarithmic probability

Return type
float

CHAPTER

SEVEN

THE K-CORRECTION MODULE

class atelier.kcorrection.**KCorrection**(*cosmology*)

Base class for an astronomical K-correction.

cosmology

Cosmology

Type
astropy.cosmology.Cosmology

M2m(*mag*, *redsh*)

Calculate the apparent (observed) magnitude based on the rest-frame absolute magnitude and redshift using the K-correction and specified cosmology.

Parameters

- **mag** (*float*) – Absolute magnitude
- **redsh** (*float*) – Redshift

Returns
Apparent magnitude

evaluate(*mag*, *redsh*)

Evaluate the K-correction for an object of given apparent magnitude and redshift.

This method is not implemented and will be overwritten by children methods.

Parameters

- **mag** (*float*) – Magnitude
- **redsh** (*float*) – Redshift

Returns
None

m2M(*mag*, *redsh*)

Calculate the rest-frame absolute magnitude based on the apparent (observed) source magnitude and redshift using the K-correction and specified cosmology.

Parameters

- **mag** (*float*) – Magnitude
- **redsh** (*float*) – Redshift

Returns
Absolute magnitude

class atelier.kcorrection.**KCorrectionGrid**(*mag_range*, *redsh_range*, *mag_bins*, *redsh_bins*, *cosmology*, *kcorrgrid=None*, *min_n_per_bin=11*)

K-correction between the observed filter band and the rest-frame filter band determined on a grid of data.

Note that an astronomical K-correction is specific to the filter bands for which it was determined. The current implementation relies on the user's understanding of the K-correction in order to apply it properly.

The K-correction is calculated in redshift and rest-frame magnitude bins on a grid of data for which redshift, rest-frame magnitudes (either from the continuum or through a filter band) and observed frame apparent magnitudes through a filter band are provided. The bins are then interpolated to provide a smooth K-correction as a function of apparent magnitude and redshift.

mag_range

Magnitude range over which the quasar selection function is evaluated.

Type
(tuple)

redsh_range

Redshift range over which the quasar selection function is evaluated.

Type
(tuple)

mag_bins

Number of bins in magnitude

Type
(int)

redsh_bins

Number of bins in redshift

Type
(int)

cosmology

Cosmology

Type
astropy.cosmology.Cosmology

slope

Slope of the power law (in frequency)

Type
float

kcorrgird

Grid of K-correction values

Type
np.ndarray

calc_kcorrection_from_df(*df*, *redshift_col_name*, *mag_col_name*, *appmag_col_name*, *n_per_bin=None*, *statistic='median'*, *inverse=True*, *verbose=1*)

Calculate the K-correction grid from a data frame

Parameters

- **df** (*pandas.DataFrame*) – Data frame with redshift, absolute reference (filter band) magnitude, and apparent (filter band) magnitude
- **redshift_col_name** (*string*) – Name of the redshift column in the data frame.
- **mag_col_name** (*string*) – Name of the absolute reference (filter band) magnitude column in the data frame.
- **appmag_col_name** (*string*) – Name of the apparent (filter band) magnitude column in the data frame
- **n_per_bin** (*int*) – Number of sources per bin (default = None). This keyword argument is used to check whether the redshift and magnitude ranges and the number of bins per dimension produce a uniform distribution of sources per bin.
- **statistic** (*string (default = 'median')*) – A string indicating whether to use the 'median' or 'mean' of the K-correction value in the bin when calculating the grid.
- **inverse** – Boolean to indicate whether inverse kcorrection will

be calculated (default=True). :type inverse: bool :param verbose: Verbosity :type verbose: int :return: None

evaluate(*mag*, *redsh*, *inverse=False*)

Evaluate the K-correction for an object of given apparent magnitude and redshift.

Parameters

- **mag** (*float*) – Magnitude
- **redsh** (*float*) – Redshift

Returns
K-correction

Return type
float

get_inv_kcorrection_from_grid()

Interpolate the inverse K-correction grid (apparent magnitude, redshift) using a linear interpolation over a rectangular mesh

Returns
None

get_kcorrection_from_grid()

Interpolate the K-correction grid (absolute magnitude, redshift) using a linear interpolation over a rectangular mesh

Returns
None

plot_grid(*title=None*, *ylabel=None*, *show_mad=False*)

Plot the K-correction grid values using matplotlib.

Parameters

- **title** (*string*) – Title of the plot
- **ylabel** (*string*) – String for the y-label of the plot.
- **show_mad** (*bool (default = False)*) – Boolean to indicate whether to show the median absolute deviation of the K-correction instead of the mean/median values.

Returns
None

plot_kcorrection(*redsh_arr=None*, *mag_arr=None*, *ylabel=None*, *save=False*, *save_filename=None*, *mag_label='$M_{{1450}}={}$'*)

Plot the K-correction (interpolation) for a range of magnitudes over the given redshifts.

Parameters

- **redsh_arr** (*np.ndarray*) – Array of redshifts for which the K-correction is evaluated.
- **mag_arr** – Array of rest-frame (filter band) magnitudes for which the K-correction is evaluated over the given redshift array.
- **ylabel** (*string*) – String for the y-label of the plot.
- **save** (*bool*) – Boolean to indicate whether to save the plot.
- **save_filename** (*string*) – Path and name to save the plot.
- **mag_label** (*string*) – Label for the magnitude

Returns
None

class atelier.kcorrection.**KCorrectionPL**(*slope*, *cosmology*)

K-correction for a power law spectrum source with a specified spectral slope per unit frequency.

Note that an astronomical K-correction is specific to the filter bands for which it was determined. The current implementation relies on the user's understanding of the K-correction in order to apply it properly.

Only for the special case of a power law spectrum source (this case) this is independent of the astronomical filter bands.

cosmology

Cosmology

Type
astropy.cosmology.Cosmology

slope

Slope of the power law (in frequency)

Type
float

evaluate(*mag*, *redsh*, *inverse=False*)

Evaluate the K-correction for an object of given apparent magnitude and redshift.

The power law K-correction does not depend on magnitude. Hence the argument 'mag' and the keyword argument 'inverse' play no role.

Parameters

- **mag** (*float*) – Magnitude
- **redsh** (*float*) – Redshift

Returns
K-correction

Return type
float

CHAPTER

EIGHT

LICENSE

PYTHON MODULE INDEX

a

INDEX

G

H

I

J

K

L

M

N

O

P

Q

R

S

U

V

W

Y