lightning.py

Release v2024.1

Erik B. Monson

Aug 13, 2024

DOCUMENTATION

1 Installation **3**

2 Getting Started **5**
2.1 Setup . . . 5
2.2 Inference . . . 6

3 Choosing a Model **9**
3.1 Stellar Model and Star Formation History . . . 9
3.2 Attenuation Models . . . 10
3.3 Dust Emission Model . . . 10
3.4 UV-IR AGN Model . . . 10
3.5 X-ray Models . . . 10

4 Choosing a Solver **11**

5 Examples **13**
5.1 Metallicity Fit - NGC 628 . . . 13
5.2 Line ratio grids . . . 45
5.3 Stellar & Nebular Emission - PEGASE . . . 56
5.4 Stellar & Nebular Emission - BPASS . . . 60
5.5 AGN Emission . . . 66
5.6 Dust Attenuation and Emission . . . 71

6 Nebular Emission Recipe **81**

7 Filter Profiles **83**
7.1 Directory Structure . . . 83
7.2 File Format . . . 83
7.3 Addition of New Filters . . . 83
7.4 List of Filters in Lightning . . . 84

8 API Reference **105**
8.1 Lightning . . . 105
8.2 Attenuation Curves . . . 115
8.3 SFH Models . . . 118
8.4 Stellar Model . . . 123
8.5 Dust Model . . . 141
8.6 AGN Model . . . 144
8.7 X-ray Models . . . 146
8.8 Prior Distributions . . . 160
8.9 Plotting . . . 163

8.10 Postprocessing 170
8.11 Posterior Predictive Checks 172

9 Attribution **175**

10 License **177**

11 Acknowledgments **179**

12 Indices and tables **181**

Python Module Index **183**

Index **185**

`lightning.py` is the python version of the Lightning SED-fitting code. The goal of the project is to maintain Lightning's primary design philosophy of simplicty and physically-based models, while taking advantage of Python's object-oriented nature and the wide array of pre-existing astronomical Python code to improve modularity and user-friendliness.

The IDL version of `Lightning` is now considered the "legacy" version, and will not see further development. Its documentation will remain available at lightning-sed.readthedocs.io, and the source code can still be downloaded from github.com/rafaeleufrasio/lightning.

This new python version contains all the features of IDL `Lightning` with the current notable exception of the Doore+(2021) inclination-dependent attenuation model. Users interested in the properties of highly-inclined disk galaxies are encouraged to continue to use IDL `Lightning` for their analyses, and to let the authors know if it would be nice to access that model through `lightning.py`.

CHAPTER

ONE

INSTALLATION

To install Lightning, all you need to do is clone the git repository to a location of your choosing and then add that location to your `PYTHONPATH` system variable.

Dependencies must be handled manually because Erik has never looked up how `setuptools` works:

```
- numpy
- scipy
- emcee
- corner
- h5py
- astropy
```

That said, you probably have compatible versions of all of the above already, and if you don't they can be installed through `conda` very easily.

CHAPTER

TWO

GETTING STARTED

2.1 Setup

The primary interface to `lightning.py` (hereafter, just `lightning`) is the `Lightning` class.

```
>>> from lightning import Lightning
```

This class holds all the information about the models and observations that are needed to perform inference with your chosen models. In its most basic configuration for inference, `Lightning` requires you to specify a set of bandpasses, a distance indicator (either a distance in Mpc or a redshift), and fluxes and uncertainties corresponding to those bandpasses (in mJy).

```
>>> import numpy as np
>>> filter_labels = ['SDSS_u', 'SDSS_g', 'SDSS_r', 'SDSS_i', 'SDSS_z']
>>> dist = 7.2
>>> fnu = np.array([237.147, 606.260, 929.869, 1205.353, 1337.25])
>>> fnu_unc = np.array([24.045, 61.632, 94.810, 122.585, 135.481])
>>> lgh = Lightning(filter_labels, lum_dist=dist, flux_obs=fnu, flux_obs_unc=fnu_unc)
```

Note that `filter_labels` is the first and only positional argument, since it's the only argument that's always required: we could specify redshift rather than distance, and it's possible to not specify fluxes and uncertainties (e.g., if you're simulating fluxes). Note also that the `flux_obs` and `flux_unc` keywords should be set by numpy arrays.

The setup here will adopt the defaults: 5 age bins spaced from 0.0-13.4 Gyr, modified Calzetti extinction (as in Noll+2009) and no dust emission (or AGN emission). All of these choices can be modified or made explicit by setting the appropriate keywords when initializing `Lightning`.

We can print the full set of model parameters with

```
>>> lgh.print_params()
['psi_1', 'psi_2', 'psi_3', 'psi_4', 'psi_5']
['Zmet']
['mcalz_tauV_diff', 'mcalz_delta', 'mcalz_tauV_BC']

Total parameters: 9
```

The `verbose` option to `print_params` additionally describes the parameters and gives their allowed ranges.

2.2 Inference

The last step before we can perform Bayesian inference is to define priors for each parameter. In `lightning` priors are passed to the `fit` method as a list of objects from the `lightning.priors` module.

```
>>> from lightning.priors import UniformPrior, ConstantPrior
>>> priors = [UniformPrior([0,10]),
...           UniformPrior([0,10]),
...           UniformPrior([0,10]),
...           UniformPrior([0,10]),
...           UniformPrior([0,10]),
...           ConstantPrior([0.020]),
...           UniformPrior([0,3]),
...           ConstantPrior([0.0]),
...           ConstantPrior([0.0])]
```

Note that the priors are (and must be) in the same order as the parameter list produced by `lgh.print_params`: the first 5 correspond to the SFH coefficients, the next is the stellar metallicity, and the final three are the parameters of the attenuation curve.

To fit with `emcee` we must define an initial state. We can sample this state from the priors of the model:

```
>>> Nwalkers = 32
>>> const_dim = np.array([pr.model_name == 'constant' for pr in priors])
>>> p0 = np.stack([pr.sample(Nwalkers) for pr in priors], axis=-1)
>>> print(p0.shape)
(32,9)
```

We can now fit:

```
>>> mcmc = lgh.fit(p0,
...                method='emcee',
...                priors=priors,
...                const_dim=const_dim,
...                Nwalkers=Nwalkers,
...                Nsteps=10000,
...                progress=True)
```

The `const_dim` keyword is a holdover from previous versions predating `ConstantPrior`. When the MCMC sampling has finished, we can construct the final MCMC chain with the `` `get_mcmc_chains` `` method:

```
>>> chain, logprob_chain, tau = lgh.get_mcmc_chains(mcmc,
...                                                 discard=3000,
...                                                 thin=500,
...                                                 Nsamples=1000,
...                                                 const_dim=const_dim,
...                                                 const_vals=p0[0,const_dim])
```

The above command can be read as "discard the first 3000 samples from each walker, retain only every 500th sample after that, and then return the last 1000 samples after merging the separate chains from each walker." The `const_dim` and `const_vals` keywords allow us to add back in the constant dimensions of the model (since they aren't sampled, the `mcmc` sampler object returned by `emcee` doesn't know anything about them) such that the `chain` array has shape (1000, 9). The `tau` returned by the above command is the estimate of the integrated autocorrelation time for each parameter. If the chains were run too short, the estimate may be unreliable (or not available). In that case the MCMC sampling can be continued from its stopping point with `mcmc.run_mcmc(Nsteps)` and then the above command can

be run again to construct the chains. In practice the discard and thin factors can be chosen based on the autocorrelation time (~a few times and ~0.5-1 times, are decent choices, respectively), but arbitrarily chosen scales are often fine as long as one inspects the chains for correlated behavior. That can be done with a chain plot:

```
>>> fig, ax = lgh.chain_plot(chain)
```

The corner plot, best fitting SED, and residuals can also be plotted:

```
>>> fig1, ax1 = lgh.corner_plot(chain, titles=True)
>>> fig2, ax2 = lgh.sed_plot_bestfit(chain, logprob_chain)
>>> fig3, ax3 = lgh.sed_plot_delchi(chain, logprob_chain)
```

SHOW PLOTS

CHAPTER

THREE

CHOOSING A MODEL

3.1 Stellar Model and Star Formation History

Modeling in `lightning.py` is biased toward the use of piecewise-constant (sometimes called "non-parametric") star formation histories (SFH). These are adopted as the default choice for fitting galaxy SEDs. Several other functional SFH forms are provided for toy models and simulations.

3.1.1 PEGASE

The default stellar population model remains the `PEGASE` models used in IDL `Lightning`, with a slightly expanded range of metallicity. These population models include an analytic prescription for nebular emission. The IMF is locked to the Kroupa et al. (2001) IMF.

3.1.2 PEGASE-A24

This release includes a new set of `PEGASE` models including a nebular emission model derived from single-cloud `Cloudy` simulations run by Amirnezam Amiri (in 2024, hence 'A24'). See Nebular Emission Recipe for a description of the procedure. The IMF remains Kroupa et al. (2001).

3.1.3 BPASS

The `BPASS` v2.1 release, including *the BPASS collaboration's* `Cloudy` *simulations*. The IMF is the `imf_135_300` option from `BPASS`.

3.1.4 BPASS-A24

`BPASS` v2.1 models including a nebular emission model derived from single-cloud `Cloudy` simulations run by Amirnezam Amiri (in 2024, hence 'A24'). See Nebular Emission Recipe for a description of the procedure. The IMF is Chabrier et al. (2003).

3.1.5 BPASS-ULX-G24

BPASS v2.1 models including a nebular emission model derived from single-cloud Cloudy simulations run by Amirnezam Amiri following the same recipe as above. However, the source SEDs are drawn from the Garofali et al. (2024, hence 'G24') "Simple X-ray Population" models, which match the BPASS UV-IR spectra with a ULX model.

3.2 Attenuation Models

The default attenuation curve in Lightning is the Noll et al. (2009) modification of the Calzetti curve, which adds a Drude profile at 2175 Å, and a parameter δ which controls the deviation from the Calzetti slope in the UV. Note that the IDL Lightning implementation of birth cloud attenuation is no longer included – the tauV_BC parameter is nonfunctional and should be left constant at 0 in any fits. The option to include dust grains in the HII regions adds some element of extra attenuation for young stars, though it shouldn't be applied incautiously, especially at low metallicity.

The featureless Calzetti curve is also preserved as an option.

3.3 Dust Emission Model

The only option for dust emission is the Draine and Li (2007) model, with 5 possible free parameters. **Energy balance between the dust emission and attenuation models is always enforced.**

3.4 UV-IR AGN Model

We implement the SKIRTOR model grid from Stalevski et al. (2016), with 3-4 free parameters, fixing the opening angle of the torus at 40 degrees. There are several quirks to our implementation:

- The UV-IR AGN model sets the normalization of the X-ray model *unless using the* QSOSED *X-ray AGN model* in which case the situation is reversed: the X-ray AGN model sets the normalization of the UV-IR model. For practical purposes this means the UV-IR normalization should be held constant when fitting the QSOSED model.
- When the polar dust model is selected the attenuation of the AGN and stellar population are decoupled - the accretion disk is attenuated only by the torus and polar dust. When the polar dust model *isn't* used, the accretion disk is subject to the same attenuation as the stellar population.

3.5 X-ray Models

TKTKTK

CHAPTER

FOUR

CHOOSING A SOLVER

Fitting in `Lightning` is biased toward the Bayesian exploration of parameter space rather than maximum likelihood estimation. As such the majority of our plotting and post-processing functions assume that you've used `emcee` for fitting.

We do however include an option for fitting the SED model with the L-BFGS-B minimization algorithm, results from which are compatible with the `lightning_postprocess` function. There is additionally an option to solve the problem with L-BFGS-B and then, if it converges, perform a brief exploration of parameter around the best fitting solution (converting any preexisting bounds on the parameters to uniform priors). This method is potentially much faster than a brute force fit with `emcee`, though it risks getting stuck in a local minimum and is not possible when the solver fails to converge. The results from this MCMC followup are naturally compatible with the MCMC plotting functions.

CHAPTER

FIVE

EXAMPLES

The following notebooks give a few use cases for lightning: fitting a normal galaxy and generating grids of line ratios for complex star formation histories.

5.1 Metallicity Fit - NGC 628

Fit NGC 628 with the new BPASS and PEGASE + nebular models.

5.1.1 Imports

[1]:
```
import numpy as np
import h5py
rng = np.random.default_rng()
import astropy.units as u
from astropy.table import Table
from corner import corner
import matplotlib.pyplot as plt
plt.style.use('ebm-dejavu')
%matplotlib inline

from lightning import Lightning
from lightning.priors import UniformPrior, NormalPrior
```

5.1.2 Setup

[28]:
```
cat = Table.read('../photometry/ngc628_dale17_photometry.fits')

# Some annoyance with converting table columns to flat numpy arrays and text encoding:
# strings come in as bytestrings (unencoded) by default python wants UTF-8, I think.
# The labels are also padded with spaces.
filter_labels = np.array([s.decode().strip() for s in cat['FILTER_LABELS'].data[0]])
fnu_obs = cat['FNU_OBS'].data[0]
fnu_unc = cat['FNU_UNC'].data[0]
dl = cat['LUMIN_DIST'].data[0]

print('BPASS setup:')
```

(continues on next page)

(continued from previous page)

```
agebins = [0.0] + list(np.logspace(7, np.log10(13.4e9), 7))

lgh = Lightning(filter_labels,
                lum_dist=dl,
                ages=agebins,
                nebula_lognH=3.5,
                nebula_dust=True, # What even happens to the continuum fit?
                stellar_type='BPASS-A24',
                SFH_type='Piecewise-Constant',
                atten_type='Modified-Calzetti',
                dust_emission=True,
                model_unc=0.10,
                print_setup_time=True)


lgh.flux_obs = fnu_obs * 1e3
lgh.flux_unc = fnu_unc * 1e3

# lgh_pg.save_json('ngc337_config.json')
# lgh_bp.save_json('ngc337_config_BPASS.json')
```

```
BPASS setup:
0.010 s elapsed in _get_filters
0.001 s elapsed in _get_wave_obs
1.972 s elapsed in stellar model setup
0.000 s elapsed in dust attenuation model setup
0.127 s elapsed in dust emission model setup
0.000 s elapsed in agn emission model setup
0.000 s elapsed in X-ray model setup
2.110 s elapsed total
```

[3]:
```
lgh.print_params(verbose=True)
```

```
============================
Piecewise-Constant
============================
Parameter  Lo  Hi              Description
--------- --- --- ------------------------
    psi_1 0.0 inf SFR in stellar age bin 1
    psi_2 0.0 inf SFR in stellar age bin 2
    psi_3 0.0 inf SFR in stellar age bin 3
    psi_4 0.0 inf SFR in stellar age bin 4
    psi_5 0.0 inf SFR in stellar age bin 5
    psi_6 0.0 inf SFR in stellar age bin 6
    psi_7 0.0 inf SFR in stellar age bin 7

============================
BPASS-Stellar-A24
============================
Parameter                    Lo                Hi                                ␣
↪           Description
--------- -------------------- ------------------ ------------------------------------
```

(continues on next page)

(continued from previous page)

```
↪-----------------------
    Zmet 0.0006471873203208087 0.0257649912911017 Metallicity (mass fraction, where␣
↪solar = 0.020 ~ 10**[-1.7])
    logU                -4.0              -1.5                              log10 of␣
↪the ionization parameter

============================
Modified-Calzetti
============================
     Parameter   Lo  Hi                                                  Description
-------------- ---- --- ------------------------------------------------------------
mcalz_tauV_diff  0.0 inf                          Optical depth of the diffuse ISM
    mcalz_delta -inf inf      Deviation from the Calzetti+2000 UV power law slope
  mcalz_tauV_BC  0.0 inf Optical depth of the birth cloud in star forming regions

============================
DL07-Dust
============================
      Parameter     Lo       Hi                                                    ␣
↪   Description
-------------- ------ -------- ------------------------------------------------------------
↪--------------
dl07_dust_alpha  -10.0      4.0                  Radiation field intensity distribution␣
↪power law index
dl07_dust_U_min    0.1     25.0                                          Radiation field␣
↪minimum intensity
dl07_dust_U_max 1000.0 300000.0                                          Radiation field␣
↪maximum intensity
dl07_dust_gamma    0.0      1.0 Fraction of dust mass exposed to radiation field␣
↪intensity distribution
dl07_dust_q_PAH 0.0047   0.0458                             Fraction of dust mass␣
↪composed of PAH grains

Total parameters: 17
```

[20]:
```
p0_seed = np.array([5,5,5,0,0,0,0,
                    0.014, -2.0,
                    0.1, -1.0, 0.0,
                    2, 3, 3e5, 0.01, 0.02])

priors = 7 * [UniformPrior([0,20])] + \
             [NormalPrior([0.013, 0.001]), NormalPrior([-2.5,0.75])] + \
             [UniformPrior([0,3]), UniformPrior([-1.5, 0.3]), None] + \
             [None, UniformPrior([0.1, 25]), None, UniformPrior([0,1]), UniformPrior([0.
↪0047, 0.0458])]
const_dim = 7 * [False] + \
            [False, False] + \
            [False, False, True] + \
            [True, False, True, False, False]
const_dim = np.array(const_dim)
const_vals = p0_seed[const_dim]
```

(continues on next page)

(continued from previous page)

```
Nwalkers = 64
# p0 = p0_seed[None, :] + rng.normal(loc=0, scale=1e-5, size=(Nwalkers, len(p0_seed)))
# p0[:, const_dim] = p0_seed[const_dim]

p0s = [pr.sample(Nwalkers) if pr is not None else np.zeros(Nwalkers) for pr in priors]
p0 = np.stack(p0s, axis=-1)
p0[:, const_dim] = const_vals[None,:]
# Metallicity and logU might sample out of bounds
p0[p0[:,7] < 0.001, 7] = 0.001
p0[p0[:,7] > 0.02, 7] = 0.02
p0[p0[:,8] < -3, 8] = -3
p0[p0[:,8] > -1.5, 8] = -1.5


print('BPASS Model:')
mcmc = lgh.fit(p0,
               method='emcee',
               priors=priors,
               const_dim=const_dim,
               Nwalkers=Nwalkers,
               Nsteps=30000,
               progress=True)

# mcmc = l.fit(p0, method='emcee', Nwalkers=Nwalkers, Nsteps=20000, priors=priors, const_
↪dim=const_dim)
# print(res_bp)
```

```
BPASS Model:
```

```
100%|##########################################################| 30000/30000 [17:19<00:00,␣
↪28.85it/s]
```

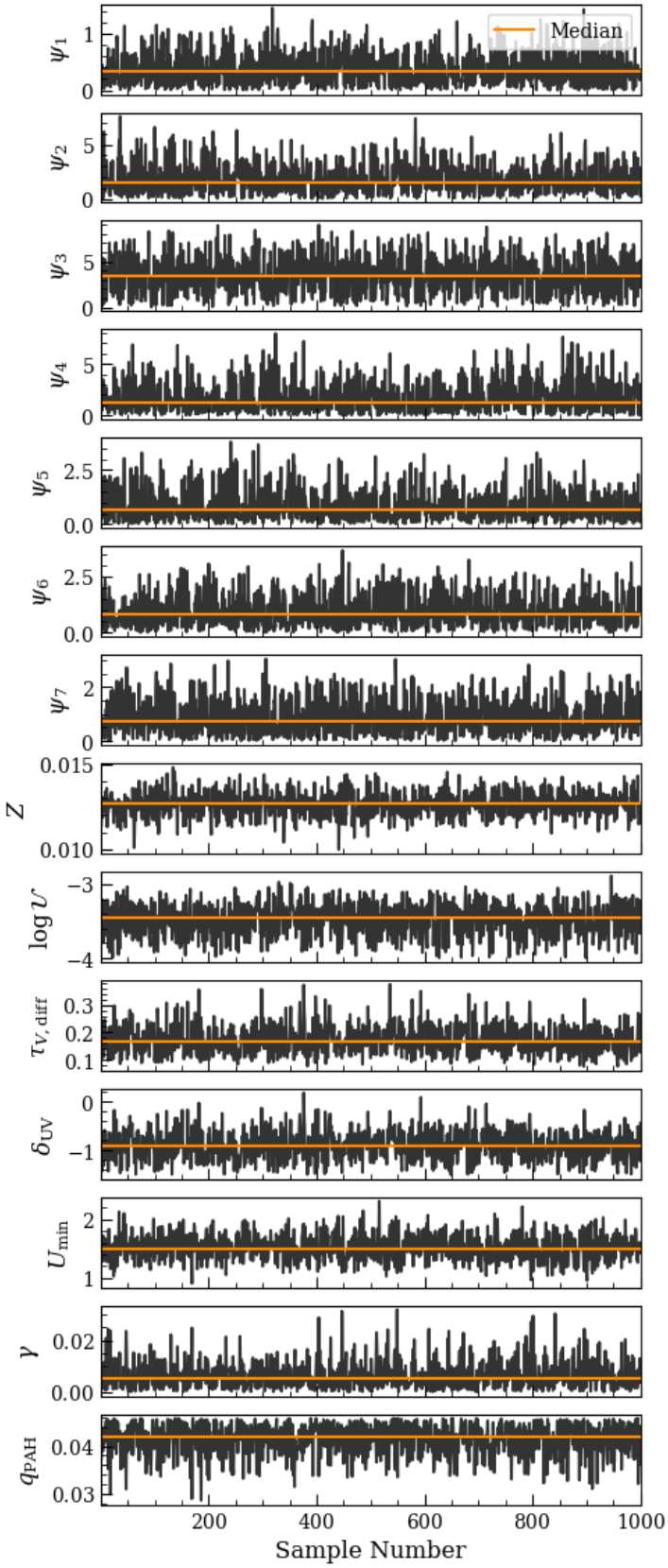
[21]:
```
print('MCMC mean acceptance fraction: %.3f' % (np.mean(mcmc.acceptance_fraction)))
```

```
MCMC mean acceptance fraction: 0.208
```

[22]:
```
chain, logprob_chain, tau_ac = lgh.get_mcmc_chains(mcmc, discard=15000, thin=500, const_
↪dim=const_dim, const_vals=p0_seed[const_dim])
```

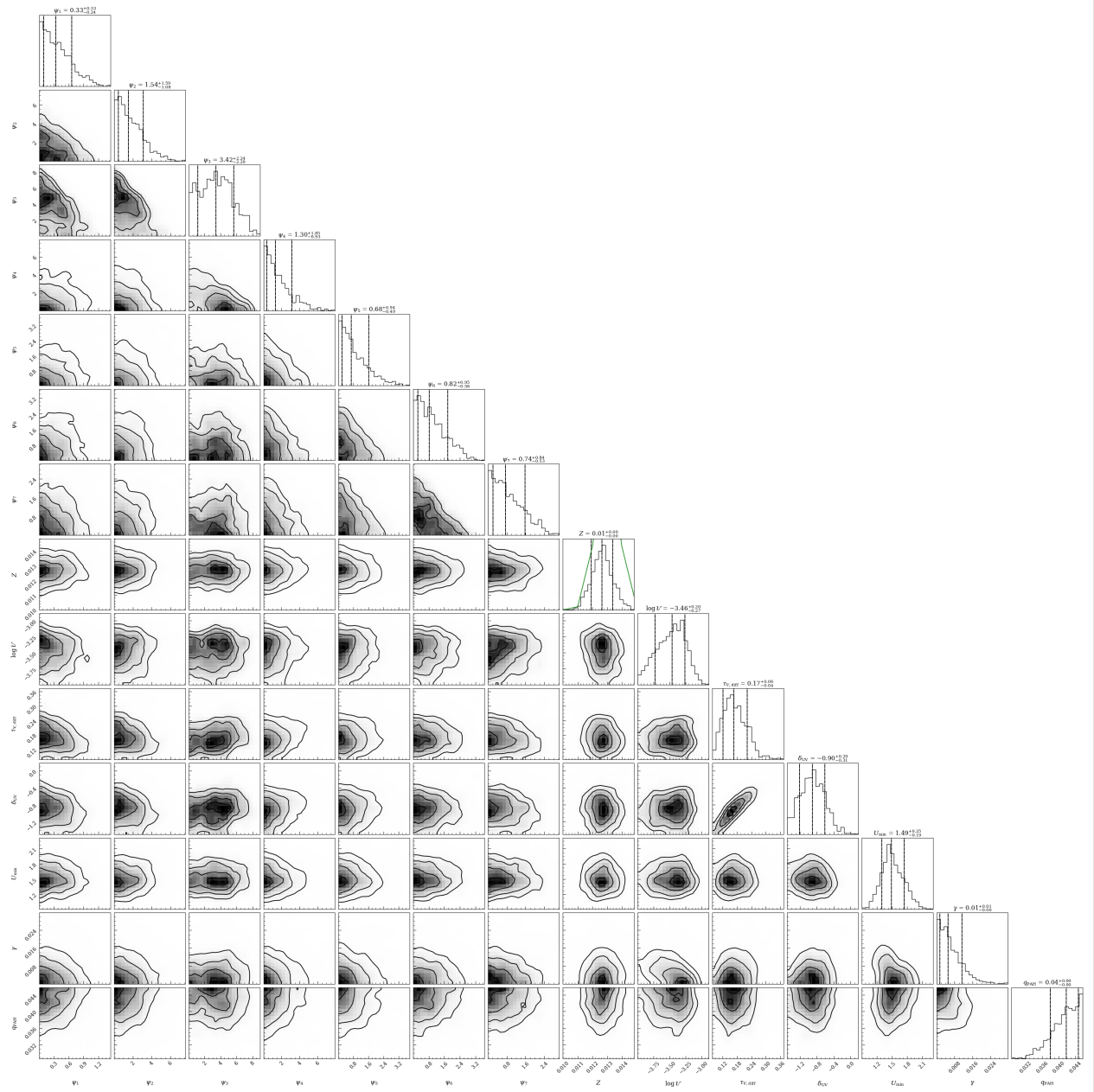
```
WARNING: The integrated autocorrelation time is longer than N/50.
         The autocorrelation estimate may be unreliable.
```

[6]:
```
fig, axs = lgh.chain_plot(chain, color='k', alpha=0.8)
```


Median
ψ_1
ψ_2
ψ_3
ψ_4
ψ_5
ψ_6
ψ_7
Z
$\log U$
$\tau_{V,\mathrm{diff}}$
δ_{UV}
U_{min}
γ
q_{PAH}
Sample Number

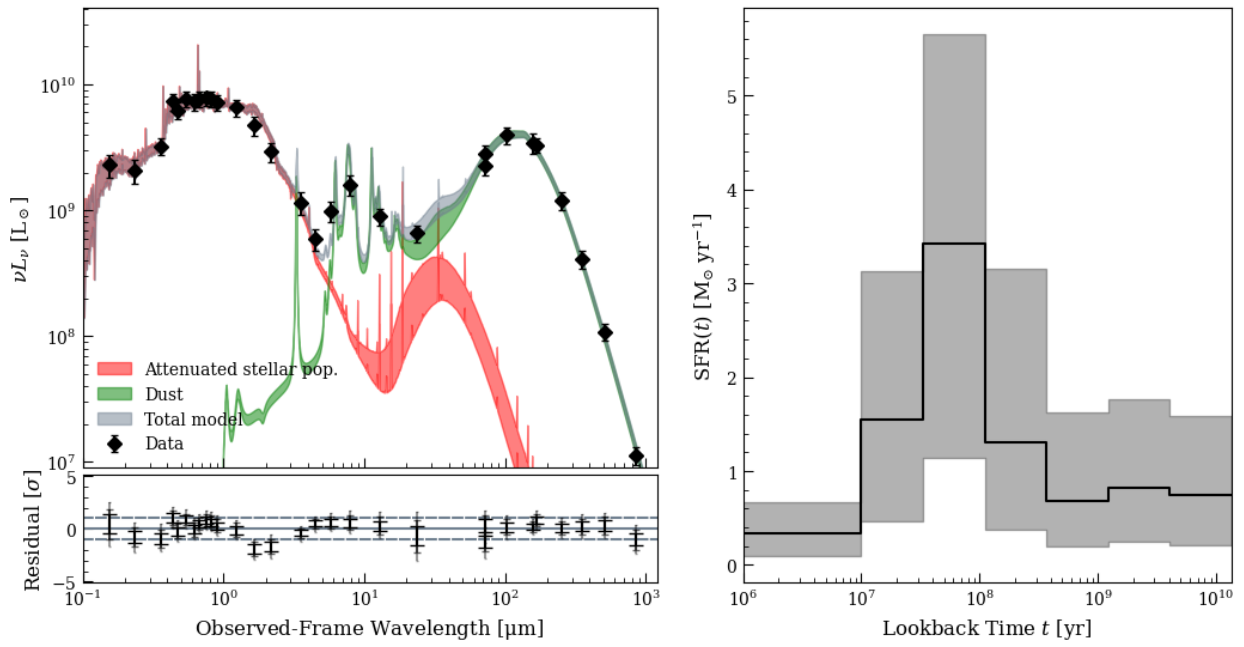
[7]:
```
fig = lgh.corner_plot(chain,
                      quantiles=(0.16, 0.50, 0.84),
                      smooth=1,
                      levels=None,
                      show_titles=True)

ZZ = np.linspace(0.001, 0.03, 30)
Zprior = lambda Z, mu, s: 1 / np.sqrt(2 * np.pi * s**2) * np.exp(-1 * (Z - mu)**2 / s**2)
axs = (np.array(fig.axes)).reshape(14,14)
yy = Zprior(ZZ, 0.013, 0.001)
axs[7,7].plot(ZZ, yy, color='forestgreen')
```

[7]:
```
[<matplotlib.lines.Line2D at 0x18029bdc0>]
```



[8]:
```
from lightning.plots import sed_plot_morebayesian, sed_plot_delchi_morebayesian, sfh_plot

fig5 = plt.figure(figsize=(12,6))
ax51 = fig5.add_axes([0.1, 0.26, 0.4, 0.64])
ax52 = fig5.add_axes([0.1, 0.1, 0.4, 0.15])
ax53 = fig5.add_axes([0.56, 0.1, 0.34, 0.8])

fig5, ax51 = sed_plot_morebayesian(lgh, chain, logprob_chain,
                                   plot_components=True,
                                   ax=ax51,
                                   legend_kwargs={'loc': 'lower left', 'frameon': False})
ax51.set_xticklabels([])
fig5, ax52 = sed_plot_delchi_morebayesian(lgh, chain, logprob_chain, ax=ax52)
fig5, ax53 = lgh.sfh_plot(chain, ax=ax53)
```

```
/Users/eqm5663/miniconda3/envs/ciao-4.15/lib/python3.10/site-packages/numpy/lib/
↪nanfunctions.py:1217: RuntimeWarning: All-NaN slice encountered
  return function_base._ureduce(a, func=_nanmedian, keepdims=keepdims,
/Users/eqm5663/miniconda3/envs/ciao-4.15/lib/python3.10/site-packages/numpy/lib/
↪nanfunctions.py:1563: RuntimeWarning: All-NaN slice encountered
  return function_base._ureduce(a,
```



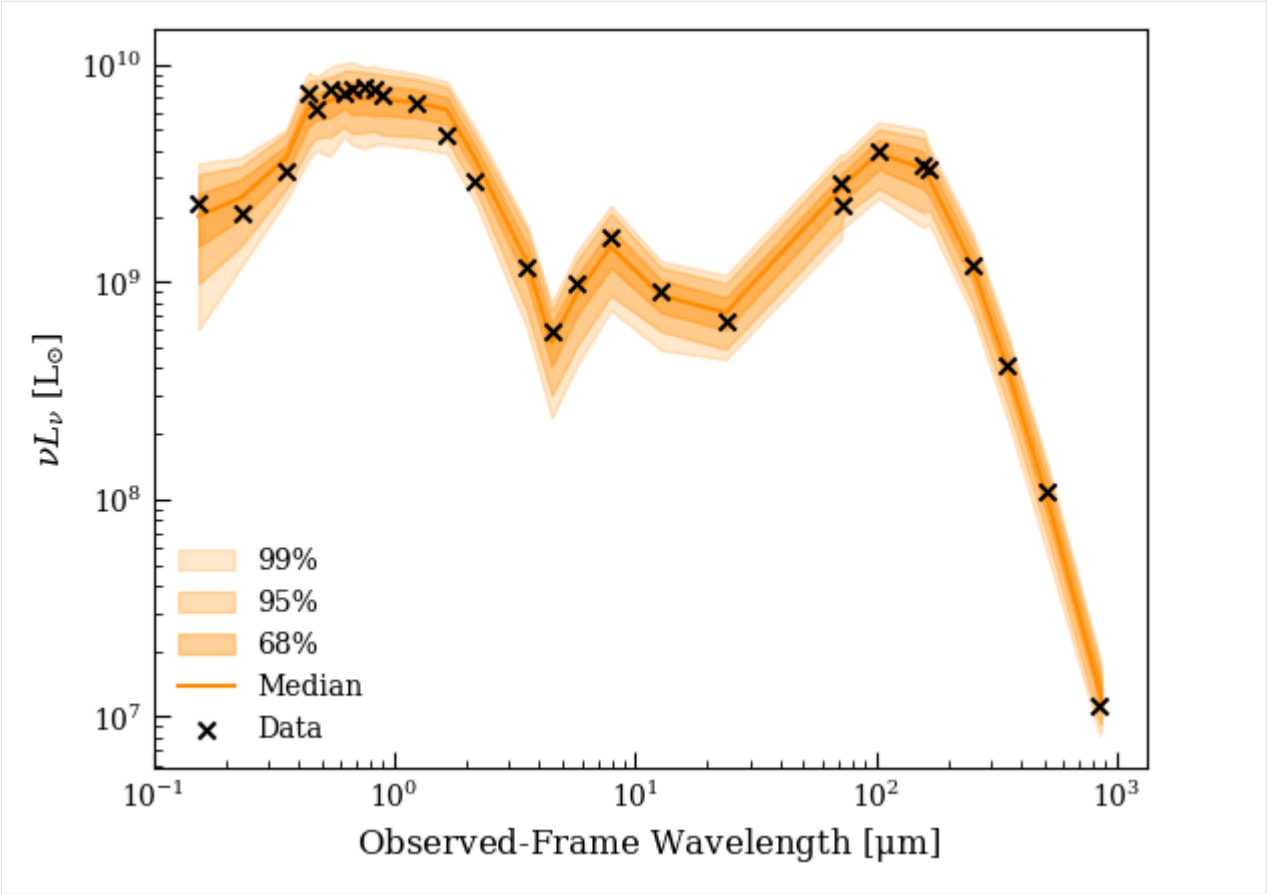
[9]:
```
from lightning.ppc import ppc, ppc_sed

# I'm not thrilled about the way this works;
# the PPC needs all the parameters, even the constant
# ones. It's probably past time to make the MCMC fit function
# return a more user-friendly data structure that includes the
# constant values.
# param_arr = np.zeros((chain.shape[0], l.Nparams))
# param_arr[:, const_dim] = p[None, const_dim]
# param_arr[:, var_dim] = chain
```

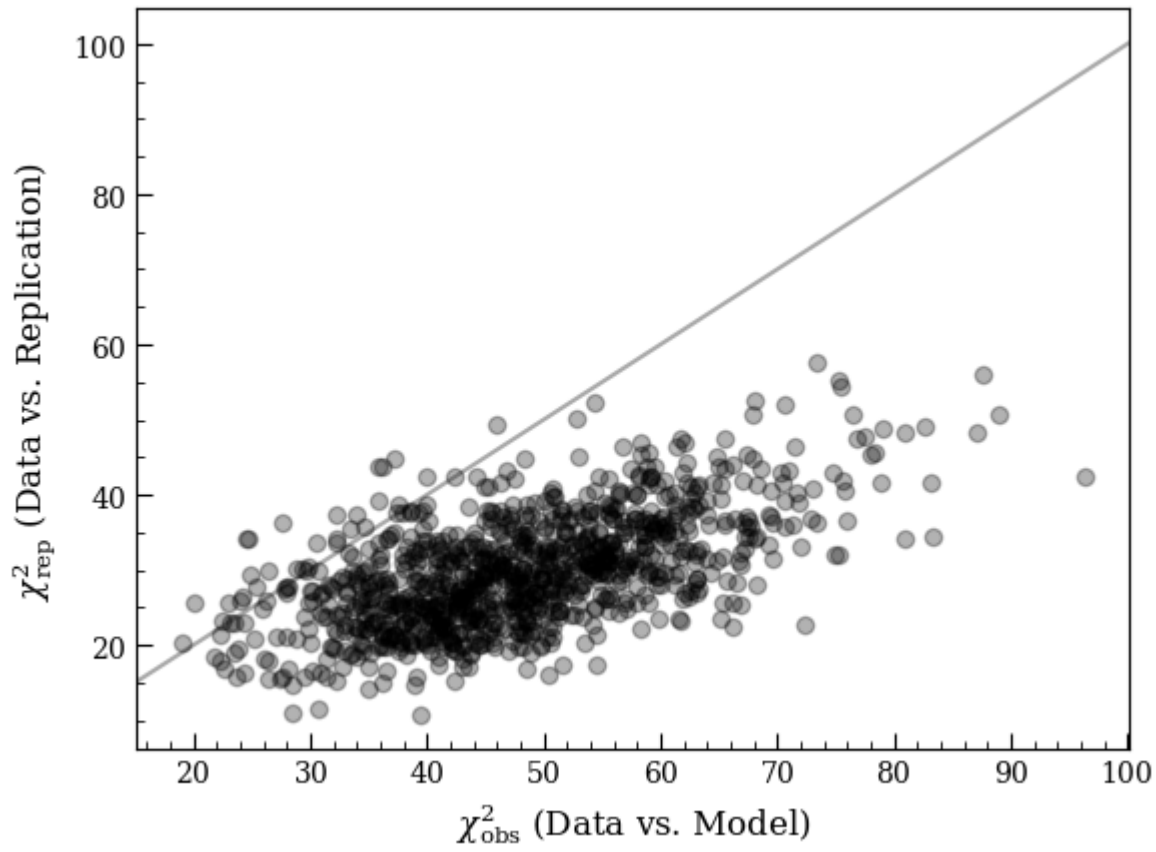
(continues on next page)

(continued from previous page)

```
pvalue, chi2_rep, chi2_obs = ppc(lgh, chain,
                                 logprob_chain,
                                 Nrep=1000,
                                 seed=12345)
fig, ax = ppc_sed(lgh, chain,
                  logprob_chain,
                  Nrep=1000,
                  seed=12345,
                  normalize=False)

fig2, ax2 = plt.subplots()

ax2.scatter(chi2_obs,
            chi2_rep,
            marker='o',
            alpha=0.3)

xlim = ax2.get_xlim()
ax2.plot(xlim, xlim, linestyle='-', color='darkgray', zorder=-1)
ax2.set_xlim(xlim)

ax2.set_xlabel(r'$\chi_{\rm obs}^2$ (Data vs. Model)')
ax2.set_ylabel(r'$\chi_{\rm rep}^2$ (Data vs. Replication)')


print('p = %.3f' % (pvalue))
```

```
p = 0.032
```


10^{10}
10^{9}
10^{8}
10^{7}
νL_ν [$L_\odot$]
99%
95%
68%
Median
Data
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
Observed-Frame Wavelength [μm]



5.1.3 Line ratios

[26]:
```
import corner
# Convenience functions I made to plot BPT-like diagnostic regions
from python_snippets import k06_NIIplot, k06_SIIplot, k06_OIplot

from astropy.table import Table
groves = Table.read('~/Research/010_PHANGS_Chandra/groves2023_nebula_catalog.fits')
mask = groves['gal_name'] == 'NGC0628'

OIIIHbeta_obs = np.log10(groves['OIII5006_FLUX_CORR'][mask] / groves['HB4861_FLUX_CORR
↪'][mask])
NIIHalpha_obs = np.log10(groves['NII6583_FLUX_CORR'][mask] / groves['HA6562_FLUX_CORR
↪'][mask])
SIIHalpha_obs = np.log10((groves['SII6716_FLUX_CORR'][mask] + groves['SII6730_FLUX_CORR
↪'][mask]) / groves['HA6562_FLUX_CORR'][mask])
OIHalpha_obs = np.log10(groves['OI6300_FLUX_CORR'][mask] / groves['HA6562_FLUX_CORR
↪'][mask])

# linelum, linelum_intr = lgh.stars.get_model_lines(lgh.sfh, chain[:,:7], chain[:,7:9])
linelum, linelum_intr = lgh.get_model_lines(chain)
# print(lineratios.shape)
```

(continues on next page)

(continued from previous page)

```
# print(lineratios[0,:])

OIIImask = lgh.stars.line_labels == 'O__3_500684A'
Halphamask = lgh.stars.line_labels == 'H__1_656280A'
Hbetamask = lgh.stars.line_labels == 'H__1_486132A'
NIImask = lgh.stars.line_labels == 'N__2_658345A'
SII6717mask = lgh.stars.line_labels == 'S__2_671644A'
SII6730mask = lgh.stars.line_labels == 'S__2_673082A'
OImask = lgh.stars.line_labels == 'BLND_630000A'

print(NIImask)
# print(OIIImask)

OIIIHbeta = np.log10(linelum[:,OIIImask] / linelum[:,Hbetamask])
NIIHalpha = np.log10(linelum[:,NIImask] / linelum[:,Halphamask])
SIIHalpha = np.log10((linelum[:,SII6717mask] + linelum[:,SII6730mask]) / linelum[:,
↪Halphamask])
OIHalpha = np.log10(linelum[:,OImask] / linelum[:,Halphamask])

OIIIHbeta_intr = np.log10(linelum_intr[:,OIIImask] / linelum_intr[:,Hbetamask])
NIIHalpha_intr = np.log10(linelum_intr[:,NIImask] / linelum_intr[:,Halphamask])
SIIHalpha_intr = np.log10((linelum_intr[:,SII6717mask] + linelum_intr[:,SII6730mask]) /␣
↪linelum_intr[:,Halphamask])
OIHalpha_intr = np.log10(linelum_intr[:,OImask] / linelum_intr[:,Halphamask])

fig, axs = plt.subplots(1,3, figsize=(12,4))

# print(np.median(OIIIHbeta))
# print(np.median(NIIHalpha))
# print(np.median(SIIHalpha))
# print(np.median(OIHalpha))

corner.hist2d(NIIHalpha, OIIIHbeta, smooth=1, color='darkorange', levels=[0.50, 0.95, 0.
↪99], ax=axs[0])
corner.hist2d(NIIHalpha_intr, OIIIHbeta_intr, smooth=1, color='dodgerblue', levels=[0.50,
↪ 0.95, 0.99], ax=axs[0])
k06_NIIplot(ax=axs[0])
axs[0].scatter(NIIHalpha_obs, OIIIHbeta_obs, marker='.', color='slategray', alpha=0.2,␣
↪label='Groves+(2023)')

# OIIIHbeta_M06 = np.log10(275 / 152.700)
# NIIHalpha_M06 = np.log10(4.7 / 455.00)
# axs[0].scatter(NIIHalpha_M06, OIIIHbeta_M06, marker='*', s=100, color='forestgreen',␣
↪label='Moustakas+(2006)')

axs[0].set_xlim(-2.2,1)
axs[0].set_ylim(-1,2)
axs[0].set_xlabel(r'$\rm \log([N II] / H\alpha)$')
axs[0].set_ylabel(r'$\rm \log([O III] / H\beta)$')
axs[0].legend(loc='best')

corner.hist2d(SIIHalpha, OIIIHbeta, smooth=1, color='darkorange', levels=[0.50, 0.95, 0.
```

(continues on next page)

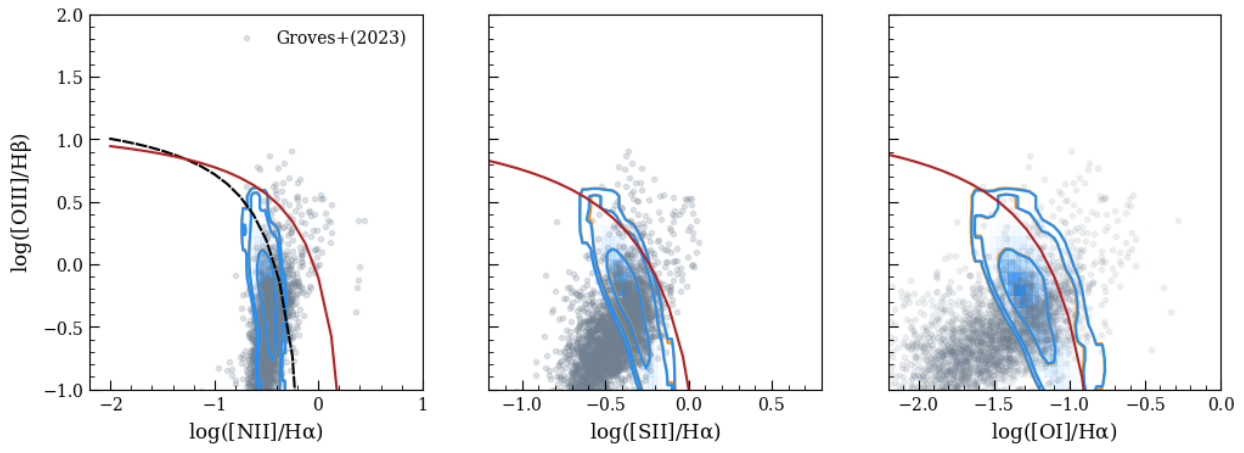
(continued from previous page)

```
↪99], ax=axs[1])
corner.hist2d(SIIHalpha_intr, OIIIHbeta_intr, smooth=1, color='dodgerblue', levels=[0.50,
↪ 0.95, 0.99], ax=axs[1])
axs[1].scatter(SIIHalpha_obs, OIIIHbeta_obs, marker='.', color='slategray', alpha=0.2)
k06_SIIplot(ax=axs[1])
axs[1].set_xlim(-1.2,0.8)
axs[1].set_ylim(-1,2)
axs[1].set_yticklabels([])
axs[1].set_xlabel(r'$\rm \log([S II] / H\alpha)$')

corner.hist2d(OIHalpha, OIIIHbeta, smooth=1, color='darkorange', levels=[0.50, 0.95, 0.
↪99], ax=axs[2])
corner.hist2d(OIHalpha_intr, OIIIHbeta_intr, smooth=1, color='dodgerblue', levels=[0.50,␣
↪0.95, 0.99], ax=axs[2])
axs[2].scatter(OIHalpha_obs, OIIIHbeta_obs, marker='.', color='slategray', alpha=0.1)
k06_OIplot(ax=axs[2])
axs[2].set_xlim(-2.2,0.0)
axs[2].set_ylim(-1,2)
axs[2].set_yticklabels([])
axs[2].set_xlabel(r'$\rm \log([O I] / H\alpha)$')
```

```
[False False False False False  True False False False False False]
```

```
/var/folders/p6/qlk1ytxd09vf64nrq0vfw06wgg_lcz/T/ipykernel_76875/1871759201.py:9:␣
↪RuntimeWarning: divide by zero encountered in log10
  OIIIHbeta_obs = np.log10(groves['OIII5006_FLUX_CORR'][mask] / groves['HB4861_FLUX_CORR
↪'][mask])
/var/folders/p6/qlk1ytxd09vf64nrq0vfw06wgg_lcz/T/ipykernel_76875/1871759201.py:12:␣
↪RuntimeWarning: divide by zero encountered in log10
  OIHalpha_obs = np.log10(groves['OI6300_FLUX_CORR'][mask] / groves['HA6562_FLUX_CORR
↪'][mask])
```

[26]:
```
Text(0.5, 0, '$\\rm \\log([O I] / H\\alpha)$')
```



5.1.4 Try the older BPASS+Cloudy models

To compare the line ratios especially.

[31]:
```
cat = Table.read('../photometry/ngc628_dale17_photometry.fits')

# Some annoyance with converting table columns to flat numpy arrays and text encoding:
# strings come in as bytestrings (unencoded) by default python wants UTF-8, I think.
# The labels are also padded with spaces.
filter_labels = np.array([s.decode().strip() for s in cat['FILTER_LABELS'].data[0]])
fnu_obs = cat['FNU_OBS'].data[0]
fnu_unc = cat['FNU_UNC'].data[0]
dl = cat['LUMIN_DIST'].data[0]

print('BPASS setup:')

agebins = [0.0] + list(np.logspace(7, np.log10(13.4e9), 7))

lgh = Lightning(filter_labels,
                lum_dist=dl,
                ages=agebins,
                stellar_type='BPASS',
                SFH_type='Piecewise-Constant',
                atten_type='Modified-Calzetti',
                dust_emission=True,
                model_unc=0.10,
                print_setup_time=True)


lgh.flux_obs = fnu_obs * 1e3
lgh.flux_unc = fnu_unc * 1e3

# lgh_pg.save_json('ngc337_config.json')
# lgh_bp.save_json('ngc337_config_BPASS.json')
```

```
BPASS setup:
0.028 s elapsed in _get_filters
0.001 s elapsed in _get_wave_obs
2.792 s elapsed in stellar model setup
0.000 s elapsed in dust attenuation model setup
0.226 s elapsed in dust emission model setup
0.000 s elapsed in agn emission model setup
0.000 s elapsed in X-ray model setup
3.047 s elapsed total
```

[49]:
```
p0_seed = np.array([5,5,5,0,0,0,0,
                    0.014, -2.0,
                    0.1, -1.0, 0.0,
                    2, 3, 3e5, 0.01, 0.02])

priors = 7 * [UniformPrior([0,100])] + \
             [NormalPrior([0.013, 0.003]), NormalPrior([-3.0,0.75])] + \
             [UniformPrior([0,3]), UniformPrior([-1.5, 0.3]), None] + \
             [None, UniformPrior([0.1, 25]), None, UniformPrior([0,1]), UniformPrior([0.
```

(continues on next page)

(continued from previous page)

```
↪0047, 0.0458])]
const_dim = 7 * [False] + \
            [False, False] + \
            [False, False, True] + \
            [True, False, True, False, False]
const_dim = np.array(const_dim)
const_vals = p0_seed[const_dim]

Nwalkers = 64
# p0 = p0_seed[None, :] + rng.normal(loc=0, scale=1e-5, size=(Nwalkers, len(p0_seed)))
# p0[:, const_dim] = p0_seed[const_dim]

p0s = [pr.sample(Nwalkers) if pr is not None else np.zeros(Nwalkers) for pr in priors]
p0 = np.stack(p0s, axis=-1)
p0[:, const_dim] = const_vals[None,:]
# Metallicity and logU might sample out of bounds
p0[p0[:,7] < 0.001, 7] = 0.001
p0[p0[:,7] > 0.02, 7] = 0.02
p0[p0[:,8] < -3, 8] = -3
p0[p0[:,8] > -1.5, 8] = -1.5

print('BPASS Model:')
mcmc = lgh.fit(p0,
               method='emcee',
               priors=priors,
               const_dim=const_dim,
               Nwalkers=Nwalkers,
               Nsteps=30000,
               progress=True)

# mcmc = l.fit(p0, method='emcee', Nwalkers=Nwalkers, Nsteps=20000, priors=priors, const_
↪dim=const_dim)
# print(res_bp)
```

```
/Users/eqm5663/Research/code/plightning/lightning/stellar/bpass.py:473: RuntimeWarning:␣
↪divide by zero encountered in log10
  np.log10(np.transpose(self.Lnu_obs, axes=[1,2,0,3])),
```

```
BPASS Model:
```

```
  0%|                                                                                ␣
↪                                                                  | 0/30000 [00:
↪00<?, ?it/s]/Users/eqm5663/miniconda3/envs/ciao-4.15/lib/python3.10/site-packages/
↪emcee/moves/red_blue.py:99: RuntimeWarning: invalid value encountered in scalar␣
↪subtract
  lnpdiff = f + nlp - state.log_prob[j]
100%|##################################################################################
↪##################################################################| 30000/30000 [15:59<00:00,
↪ 31.27it/s]
```

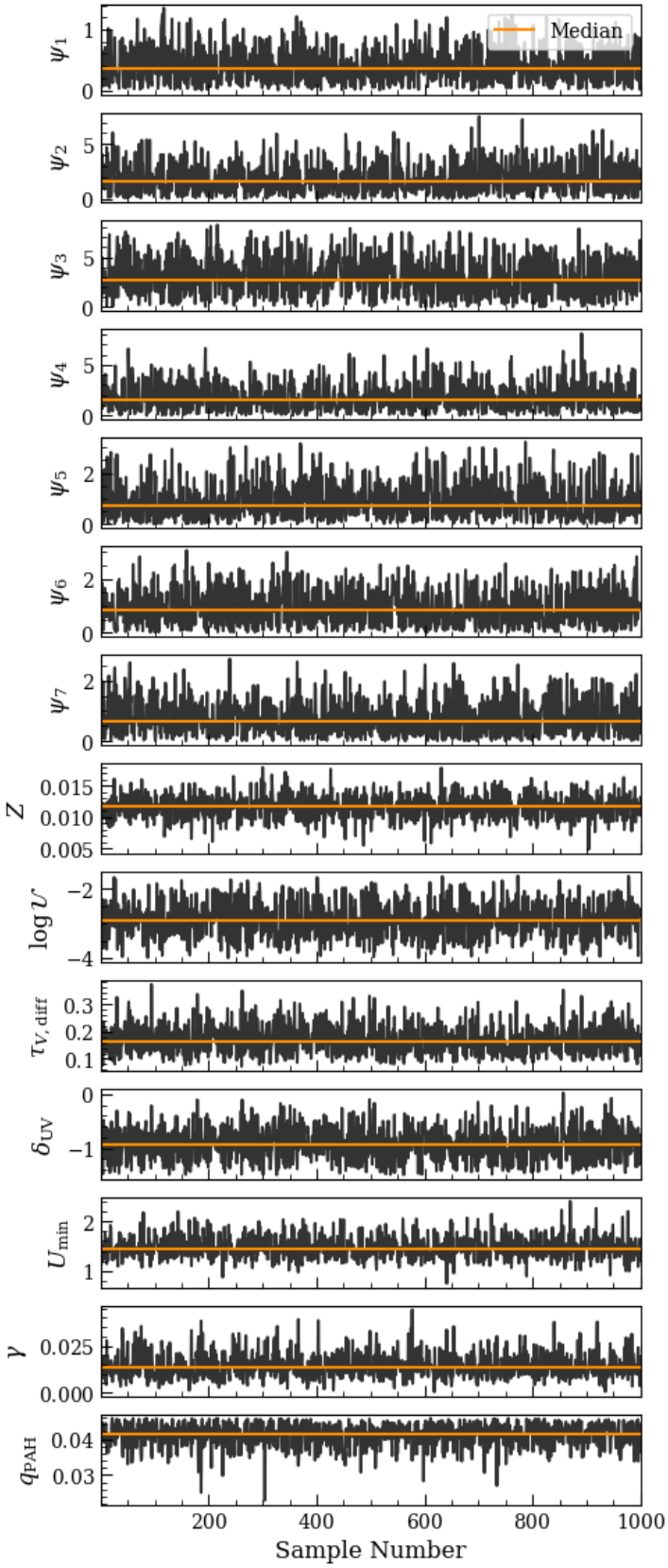
[50]:
```
print('MCMC mean acceptance fraction: %.3f' % (np.mean(mcmc.acceptance_fraction)))
```

```
MCMC mean acceptance fraction: 0.220
```

[52]:
```
chain, logprob_chain, tau_ac = lgh.get_mcmc_chains(mcmc, discard=2000, thin=500, const_
↪dim=const_dim, const_vals=p0_seed[const_dim])
```

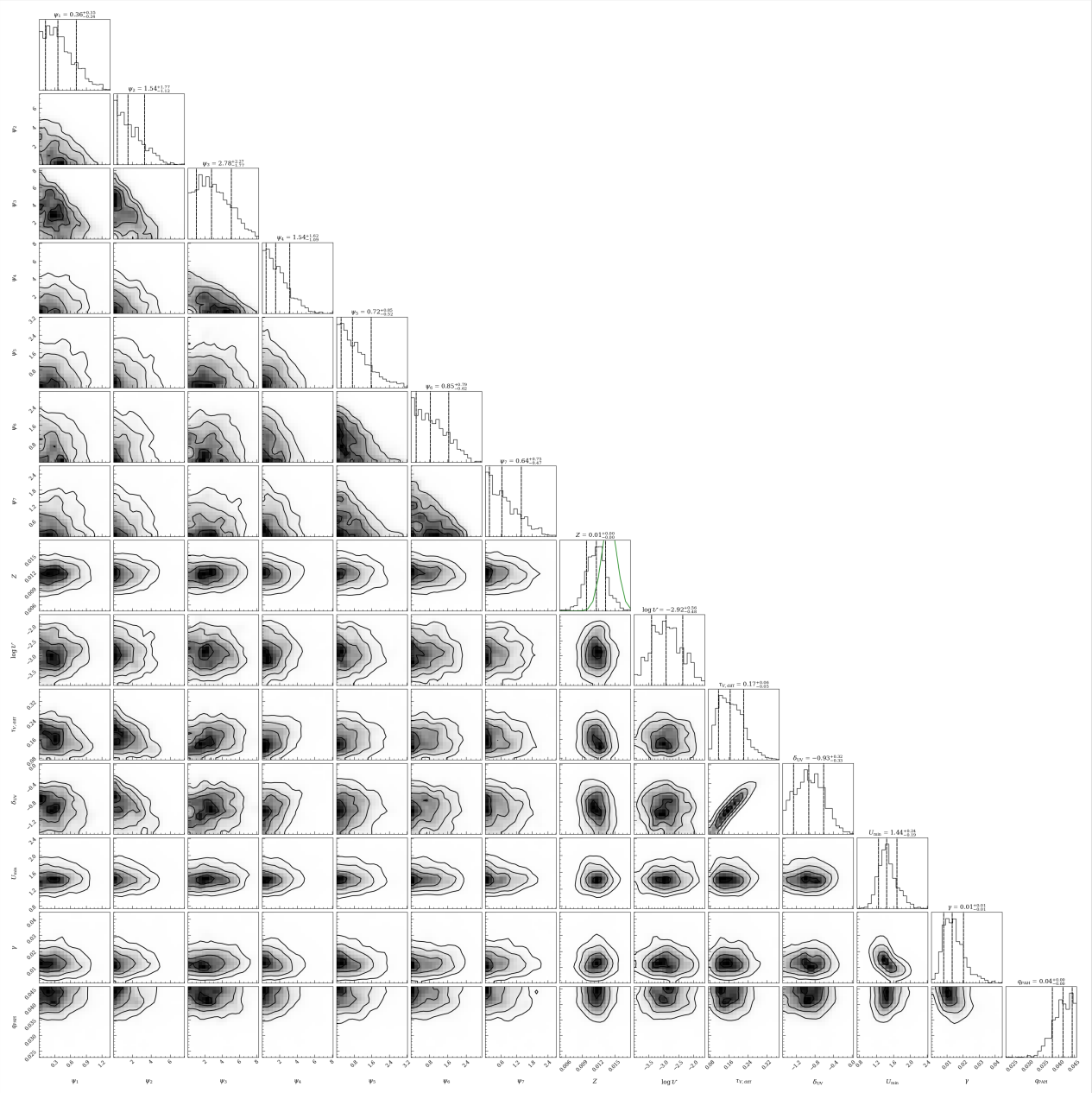
```
WARNING: The integrated autocorrelation time is longer than N/50.
         The autocorrelation estimate may be unreliable.
```

[53]:
```
fig, axs = lgh.chain_plot(chain, color='k', alpha=0.8)
```


Median
ψ_1
ψ_2
ψ_3
ψ_4
ψ_5
ψ_6
ψ_7
Z
$\log U$
$\tau_{V,\mathrm{diff}}$
δ_{UV}
U_{min}
γ
q_{PAH}
Sample Number

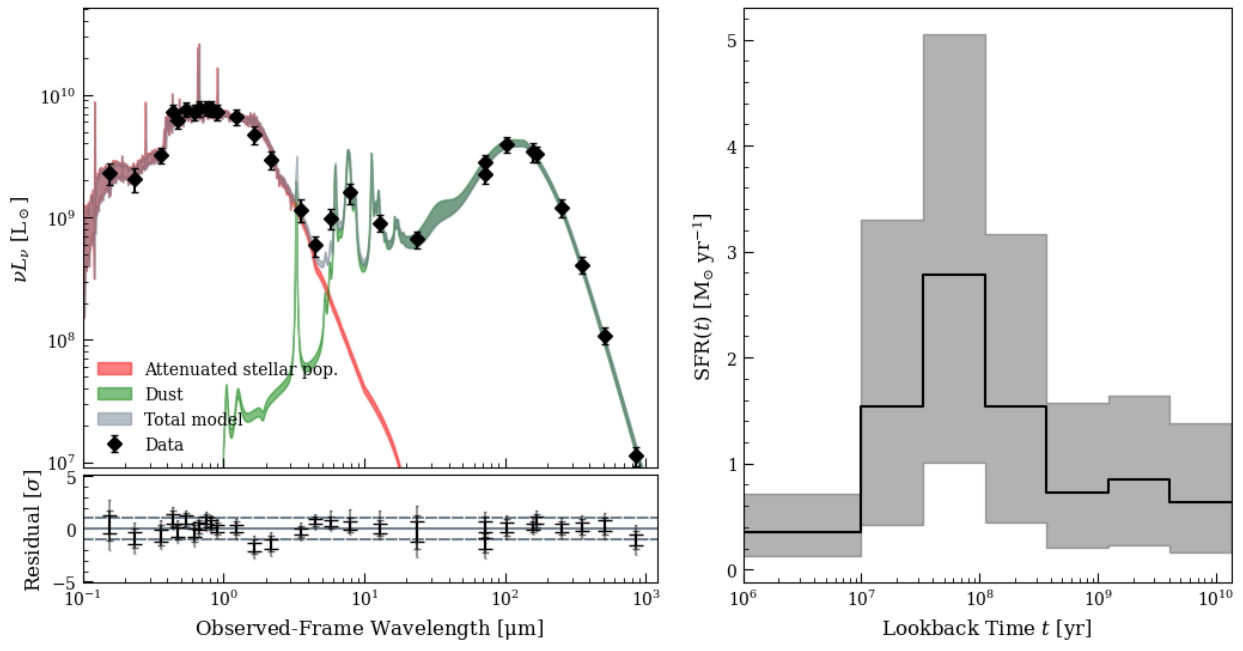
[54]:
```
fig = lgh.corner_plot(chain,
                      quantiles=(0.16, 0.50, 0.84),
                      smooth=1,
                      levels=None,
                      show_titles=True)

ZZ = np.linspace(0.001, 0.03, 30)
Zprior = lambda Z, mu, s: 1 / np.sqrt(2 * np.pi * s**2) * np.exp(-1 * (Z - mu)**2 / s**2)
axs = (np.array(fig.axes)).reshape(14,14)
yy = Zprior(ZZ, 0.014, 0.002)
axs[7,7].plot(ZZ, yy, color='forestgreen')
```

[54]:
```
[<matplotlib.lines.Line2D at 0x1964b66e0>]
```



[55]:
```
from lightning.plots import sed_plot_morebayesian, sed_plot_delchi_morebayesian, sfh_plot

fig5 = plt.figure(figsize=(12,6))
ax51 = fig5.add_axes([0.1, 0.26, 0.4, 0.64])
ax52 = fig5.add_axes([0.1, 0.1, 0.4, 0.15])
ax53 = fig5.add_axes([0.56, 0.1, 0.34, 0.8])

fig5, ax51 = sed_plot_morebayesian(lgh, chain, logprob_chain,
                                   plot_components=True,
                                   ax=ax51,
                                   legend_kwargs={'loc': 'lower left', 'frameon': False})
ax51.set_xticklabels([])
fig5, ax52 = sed_plot_delchi_morebayesian(lgh, chain, logprob_chain, ax=ax52)
fig5, ax53 = lgh.sfh_plot(chain, ax=ax53)
```

```
/Users/eqm5663/Research/code/plightning/lightning/stellar/bpass.py:473: RuntimeWarning:␣
↪divide by zero encountered in log10
  np.log10(np.transpose(self.Lnu_obs, axes=[1,2,0,3])),
/Users/eqm5663/Research/code/plightning/lightning/stellar/bpass.py:473: RuntimeWarning:␣
↪divide by zero encountered in log10
  np.log10(np.transpose(self.Lnu_obs, axes=[1,2,0,3])),
/Users/eqm5663/miniconda3/envs/ciao-4.15/lib/python3.10/site-packages/numpy/lib/
↪nanfunctions.py:1217: RuntimeWarning: All-NaN slice encountered
  return function_base._ureduce(a, func=_nanmedian, keepdims=keepdims,
/Users/eqm5663/miniconda3/envs/ciao-4.15/lib/python3.10/site-packages/numpy/lib/
↪nanfunctions.py:1563: RuntimeWarning: All-NaN slice encountered
  return function_base._ureduce(a,
```



[38]:
```
print(lgh.stars.line_names)
```

```
[b'Ly_A' b'CIII1548' b'HeII1640' b'OIII1666' b'CIII1909' b'OII3726'
 b'OII3729' b'HeII4685' b'Hbeta' b'OIII5007' b'Halpha' b'NII6584'
 b'SII6716' b'SII6730' b'SIII9068' b'SIII9530' b'HeI1.083']
```

[56]:
```
import corner
# Convenience functions I made to plot BPT-like diagnostic regions
from python_snippets import k06_NIIplot, k06_SIIplot, k06_OIplot

from astropy.table import Table
groves = Table.read('~/Research/010_PHANGS_Chandra/groves2023_nebula_catalog.fits')
mask = groves['gal_name'] == 'NGC0628'

OIIIHbeta_obs = np.log10(groves['OIII5006_FLUX_CORR'][mask] / groves['HB4861_FLUX_CORR
↪'][mask])
NIIHalpha_obs = np.log10(groves['NII6583_FLUX_CORR'][mask] / groves['HA6562_FLUX_CORR
↪'][mask])
SIIHalpha_obs = np.log10((groves['SII6716_FLUX_CORR'][mask] + groves['SII6730_FLUX_CORR
↪'][mask]) / groves['HA6562_FLUX_CORR'][mask])
OIHalpha_obs = np.log10(groves['OI6300_FLUX_CORR'][mask] / groves['HA6562_FLUX_CORR
↪'][mask])

lineratios = lgh.stars.get_model_lines(lgh.sfh, chain[:,:7], chain[:,7:9])
# print(lineratios.shape)
# print(lineratios[0,:])

OIIImask = lgh.stars.line_names == b'OIII5007'
Halphamask = lgh.stars.line_names == b'Halpha'
Hbetamask = lgh.stars.line_names == b'Hbeta'
NIImask = lgh.stars.line_names == b'NII6584'
SII6717mask = lgh.stars.line_names == b'SII6716'
SII6730mask = lgh.stars.line_names == b'SII6730'
# OImask = lgh.stars.line_names == b'BLND_630000A'

OIIIHbeta = np.log10(lineratios[:,OIIImask] / lineratios[:,Hbetamask])
NIIHalpha = np.log10(lineratios[:,NIImask] / lineratios[:,Halphamask])
SIIHalpha = np.log10((lineratios[:,SII6717mask] + lineratios[:,SII6730mask]) /␣
↪lineratios[:,Halphamask])
# OIHalpha = np.log10(lineratios[:,OImask] / lineratios[:,Halphamask])

fig, axs = plt.subplots(1,3, figsize=(12,4))

print(np.median(OIIIHbeta))
print(np.median(NIIHalpha))
print(np.median(SIIHalpha))
# print(np.median(OIHalpha))

corner.hist2d(NIIHalpha, OIIIHbeta, smooth=1, color='forestgreen', ax=axs[0])
k06_NIIplot(ax=axs[0])
axs[0].scatter(NIIHalpha_obs, OIIIHbeta_obs, marker='.', color='slategray', alpha=0.2,␣
↪label='Groves+(2023)')

# OIIIHbeta_M06 = np.log10(275 / 152.700)
# NIIHalpha_M06 = np.log10(4.7 / 455.00)
# axs[0].scatter(NIIHalpha_M06, OIIIHbeta_M06, marker='*', s=100, color='forestgreen',␣
↪label='Moustakas+(2006)')

axs[0].set_xlim(-2.2,1)
```

(continues on next page)

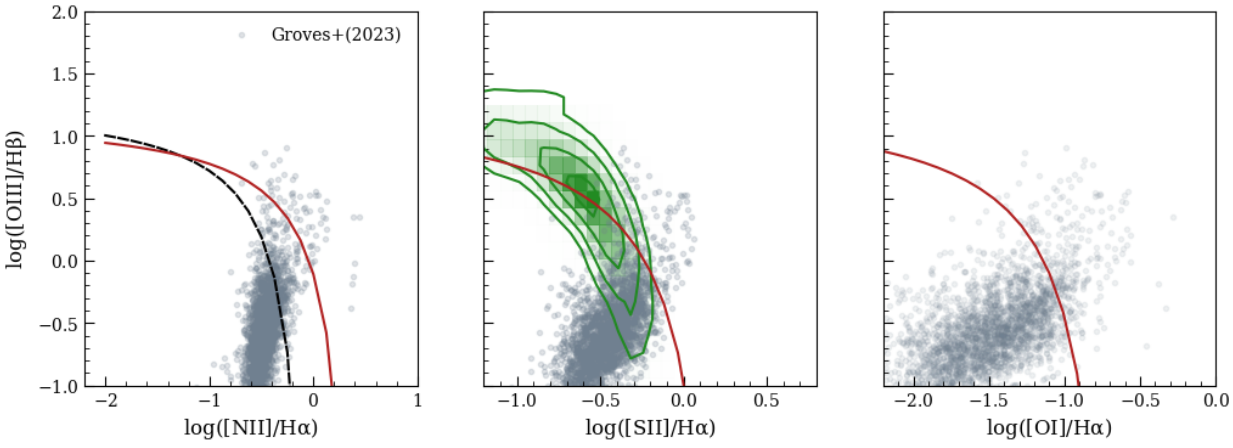
(continued from previous page)

```
axs[0].set_ylim(-1,2)
axs[0].set_xlabel(r'$\rm \log([N II] / H\alpha)$')
axs[0].set_ylabel(r'$\rm \log([O III] / H\beta)$')
axs[0].legend(loc='best')

corner.hist2d(SIIHalpha, OIIIHbeta, smooth=1, color='forestgreen', ax=axs[1])
axs[1].scatter(SIIHalpha_obs, OIIIHbeta_obs, marker='.', color='slategray', alpha=0.2)
k06_SIIplot(ax=axs[1])
axs[1].set_xlim(-1.2,0.8)
axs[1].set_ylim(-1,2)
axs[1].set_yticklabels([])
axs[1].set_xlabel(r'$\rm \log([S II] / H\alpha)$')

# corner.hist2d(OIHalpha, OIIIHbeta, smooth=1, color='forestgreen', ax=axs[2])
axs[2].scatter(OIHalpha_obs, OIIIHbeta_obs, marker='.', color='slategray', alpha=0.1)
k06_OIplot(ax=axs[2])
axs[2].set_xlim(-2.2,0.0)
axs[2].set_ylim(-1,2)
axs[2].set_yticklabels([])
axs[2].set_xlabel(r'$\rm \log([O I] / H\alpha)$')
```

```
0.55588670945865
-5.188020367183089
-0.5791833642589199
```

```
/var/folders/p6/qlk1ytxd09vf64nrq0vfw06wgg_lcz/T/ipykernel_2096/4006764960.py:9:␣
↪RuntimeWarning: divide by zero encountered in log10
  OIIIHbeta_obs = np.log10(groves['OIII5006_FLUX_CORR'][mask] / groves['HB4861_FLUX_CORR
↪'][mask])
/var/folders/p6/qlk1ytxd09vf64nrq0vfw06wgg_lcz/T/ipykernel_2096/4006764960.py:12:␣
↪RuntimeWarning: divide by zero encountered in log10
  OIHalpha_obs = np.log10(groves['OI6300_FLUX_CORR'][mask] / groves['HA6562_FLUX_CORR
↪'][mask])
/Users/eqm5663/Research/code/plightning/lightning/stellar/bpass.py:665: RuntimeWarning:␣
↪divide by zero encountered in log10
  np.log10(np.transpose(self.line_lum, axes=[1,2,0,3])),
```

[56]:
```
Text(0.5, 0, '$\\rm \\log([O I] / H\\alpha)$')
```



5.1.5 And finally, try the new PEGASE models:

[29]:
```
cat = Table.read('../photometry/ngc628_dale17_photometry.fits')

# Some annoyance with converting table columns to flat numpy arrays and text encoding:
# strings come in as bytestrings (unencoded) by default python wants UTF-8, I think.
# The labels are also padded with spaces.
filter_labels = np.array([s.decode().strip() for s in cat['FILTER_LABELS'].data[0]])
fnu_obs = cat['FNU_OBS'].data[0]
fnu_unc = cat['FNU_UNC'].data[0]
dl = cat['LUMIN_DIST'].data[0]

# print('BPASS setup:')

agebins = [0.0] + list(np.logspace(7, np.log10(13.4e9), 7))

lgh = Lightning(filter_labels,
                lum_dist=dl,
                ages=agebins,
                nebula_lognH=3.5,
                nebula_dust=True,
                stellar_type='PEGASE-A24',
                SFH_type='Piecewise-Constant',
                atten_type='Modified-Calzetti',
                dust_emission=True,
                model_unc=0.10,
                print_setup_time=True)


lgh.flux_obs = fnu_obs * 1e3
lgh.flux_unc = fnu_unc * 1e3

# lgh_pg.save_json('ngc337_config.json')
# lgh_bp.save_json('ngc337_config_BPASS.json')
```

```
0.011 s elapsed in _get_filters
0.001 s elapsed in _get_wave_obs
1.628 s elapsed in stellar model setup
0.000 s elapsed in dust attenuation model setup
0.128 s elapsed in dust emission model setup
0.000 s elapsed in agn emission model setup
0.000 s elapsed in X-ray model setup
1.768 s elapsed total
```

[4]:
```
lgh.print_params(verbose=True)
```

```
============================
Piecewise-Constant
============================
Parameter  Lo  Hi              Description
--------- --- --- ------------------------
    psi_1 0.0 inf SFR in stellar age bin 1
    psi_2 0.0 inf SFR in stellar age bin 2
```

(continues on next page)

(continued from previous page)

```
    psi_3 0.0 inf SFR in stellar age bin 3
    psi_4 0.0 inf SFR in stellar age bin 4
    psi_5 0.0 inf SFR in stellar age bin 5
    psi_6 0.0 inf SFR in stellar age bin 6
    psi_7 0.0 inf SFR in stellar age bin 7

============================
PEGASE-Stellar-A24
============================
Parameter                    Lo                 Hi                                  ␣
↪            Description
--------- -------------------- ------------------ ------------------------------------
↪-----------------------
     Zmet 0.0006471873203208087 0.0257649912911017 Metallicity (mass fraction, where␣
↪solar = 0.020 ~ 10**[-1.7])
     logU                 -4.0               -1.5                                log10 of␣
↪the ionization parameter

============================
Modified-Calzetti
============================
      Parameter   Lo                 Hi                                          ␣
↪                            Description
--------------- ---- ------------------ ------------------------------------------------
↪----------------------------------------
mcalz_tauV_diff  0.0                inf                                          ␣
↪         Optical depth of the diffuse ISM
    mcalz_delta -inf 0.4473684210526316 Deviation from the Calzetti+2000 UV power law␣
↪slope (Upper limit set by requiring Eb >= 0)
  mcalz_tauV_BC  0.0                inf                                Optical depth␣
↪of the birth cloud in star forming regions

============================
DL07-Dust
============================
      Parameter     Lo       Hi                                                    ␣
↪   Description
--------------- ------ -------- ------------------------------------------------------
↪--------------
dl07_dust_alpha  -10.0      4.0                  Radiation field intensity distribution␣
↪power law index
dl07_dust_U_min    0.1     25.0                                       Radiation field␣
↪minimum intensity
dl07_dust_U_max 1000.0 300000.0                                       Radiation field␣
↪maximum intensity
dl07_dust_gamma    0.0      1.0 Fraction of dust mass exposed to radiation field␣
↪intensity distribution
dl07_dust_q_PAH 0.0047   0.0458                           Fraction of dust mass␣
↪composed of PAH grains

Total parameters: 17
```

[30]:
```
p0_seed = np.array([5,5,5,0,0,0,0,
                    0.014, -2.0,
                    0.1, -1.0, 0.0,
                    2, 3, 3e5, 0.01, 0.02])

priors = 7 * [UniformPrior([0,100])] + \
             [NormalPrior([0.013, 0.001]), NormalPrior([-2.5,0.75])] + \
             [UniformPrior([0,3]), UniformPrior([-1.5, 0.3]), None] + \
             [None, UniformPrior([0.1, 25]), None, UniformPrior([0,1]), UniformPrior([0.
↪0047, 0.0458])]
const_dim = 7 * [False] + \
            [False, False] + \
            [False, False, True] + \
            [True, False, True, False, False]
const_dim = np.array(const_dim)
const_vals = p0_seed[const_dim]

Nwalkers = 64
# p0 = p0_seed[None, :] + rng.normal(loc=0, scale=1e-5, size=(Nwalkers, len(p0_seed)))
# p0[:, const_dim] = p0_seed[const_dim]
p0s = [pr.sample(Nwalkers) if pr is not None else np.zeros(Nwalkers) for pr in priors]
p0 = np.stack(p0s, axis=-1)
p0[:, const_dim] = const_vals[None,:]
# Metallicity and logU might sample out of bounds
p0[p0[:,7] < 0.001, 7] = 0.001
p0[p0[:,7] > 0.02, 7] = 0.02
p0[p0[:,8] < -3, 8] = -3
p0[p0[:,8] > -1.5, 8] = -1.5

print('PEGASE Model:')
mcmc = lgh.fit(p0,
               method='emcee',
               priors=priors,
               const_dim=const_dim,
               Nwalkers=Nwalkers,
               Nsteps=30000,
               progress=True)

# mcmc = l.fit(p0, method='emcee', Nwalkers=Nwalkers, Nsteps=20000, priors=priors, const_
↪dim=const_dim)
# print(res_bp)
```

```
PEGASE Model:
```

```
/Users/eqm5663/Research/code/plightning/lightning/stellar/pegase.py:1007: RuntimeWarning:
↪ divide by zero encountered in log10
  np.log10(np.transpose(self.Lnu_obs, axes=[1,2,0,3])),
/Users/eqm5663/miniconda3/envs/ciao-4.15/lib/python3.10/site-packages/scipy/interpolate/_
↪rgi.py:418: RuntimeWarning: invalid value encountered in multiply
  term = np.asarray(self.values[edge_indices]) * weight[vslice]
100%|##########################################################| 30000/30000 [17:02<00:00,␣
↪29.34it/s]
```

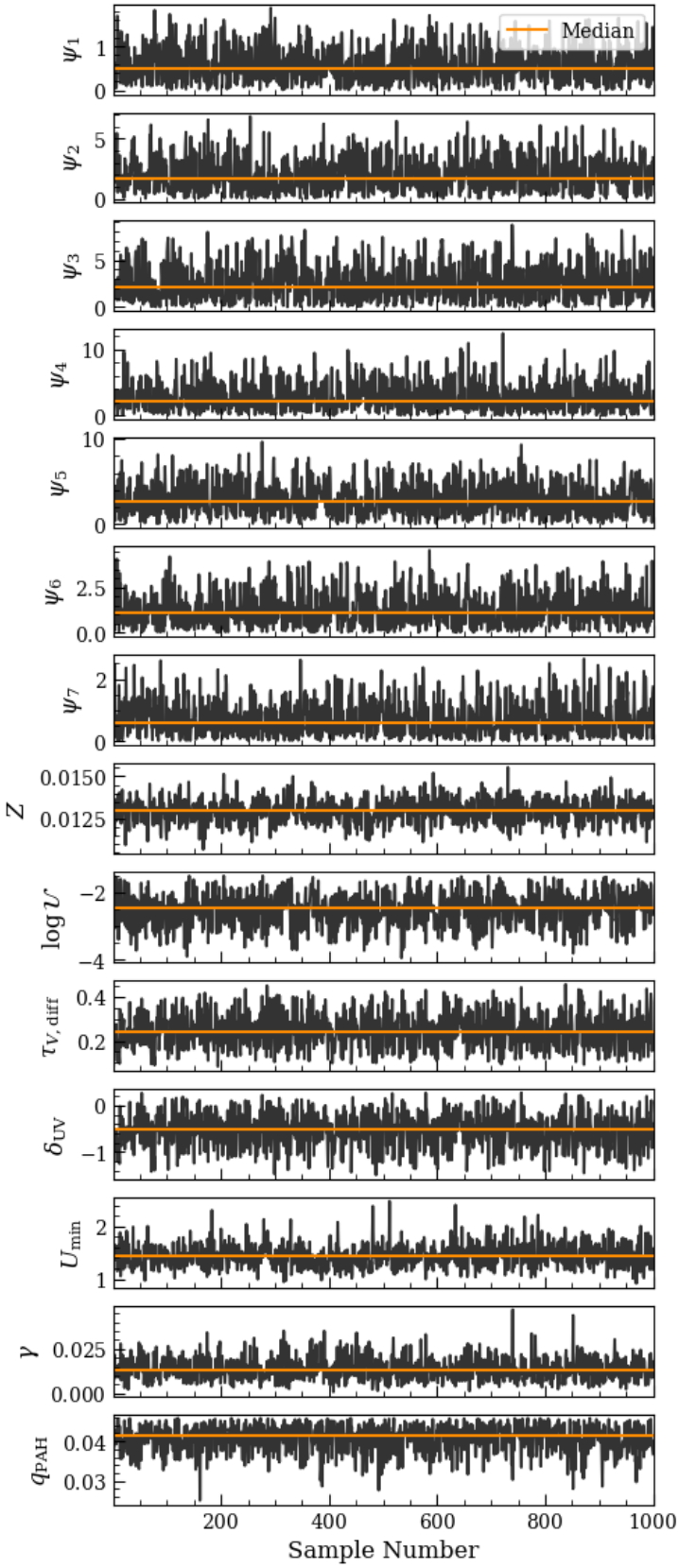
[4]:
```
print('MCMC mean acceptance fraction: %.3f' % (np.mean(mcmc.acceptance_fraction)))
```

```
MCMC mean acceptance fraction: 0.218
```

[31]:
```
chain, logprob_chain, tau_ac = lgh.get_mcmc_chains(mcmc, discard=2000, thin=500, const_
↪dim=const_dim, const_vals=p0_seed[const_dim])
```

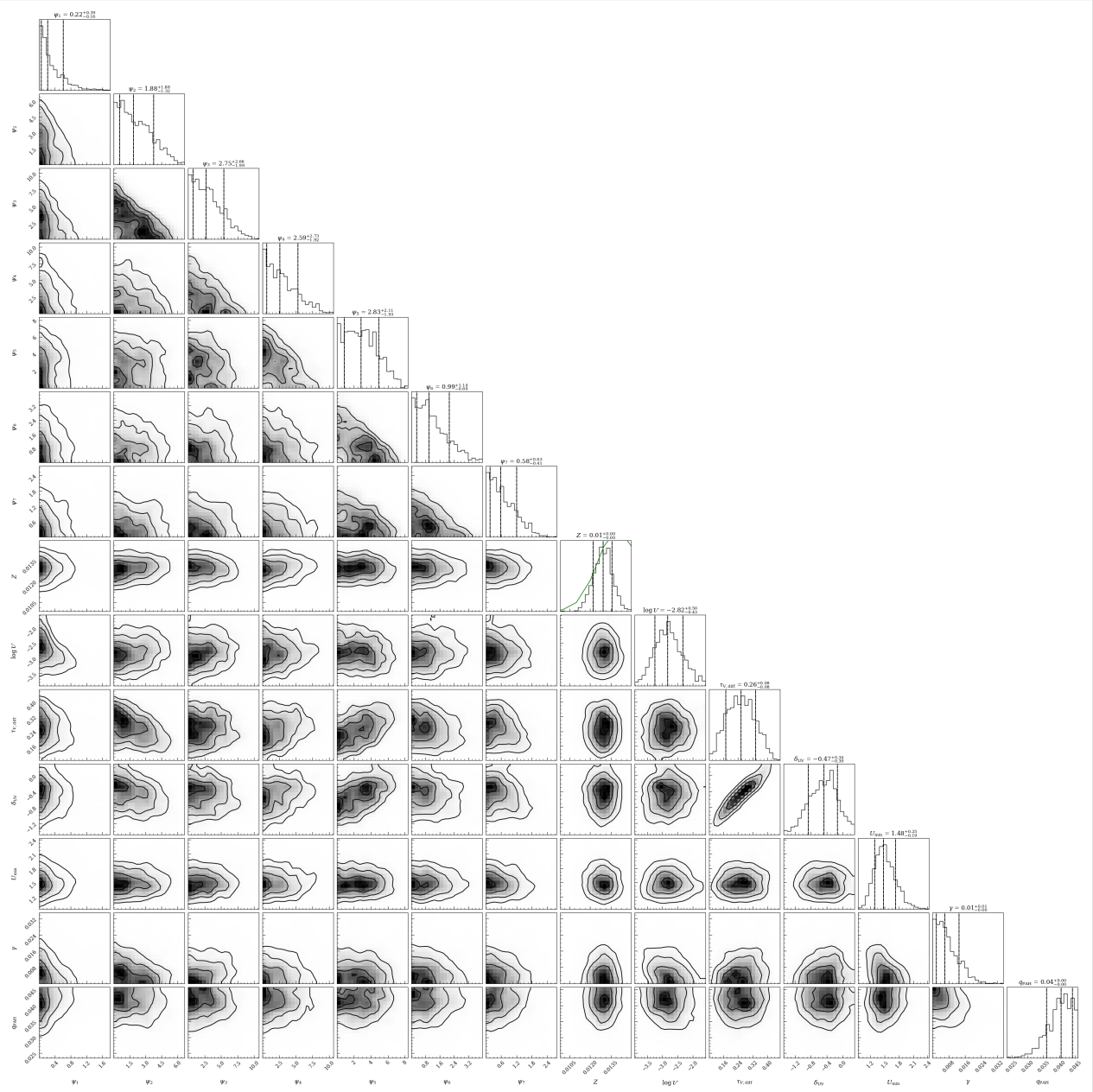
```
WARNING: The integrated autocorrelation time is longer than N/50.
         The autocorrelation estimate may be unreliable.
```

[6]:
```
fig, axs = lgh.chain_plot(chain, color='k', alpha=0.8)
```


Median
ψ_1
ψ_2
ψ_3
ψ_4
ψ_5
ψ_6
ψ_7
Z
$\log U$
$\tau_{V,\mathrm{diff}}$
δ_{UV}
U_{min}
γ
q_{PAH}
Sample Number

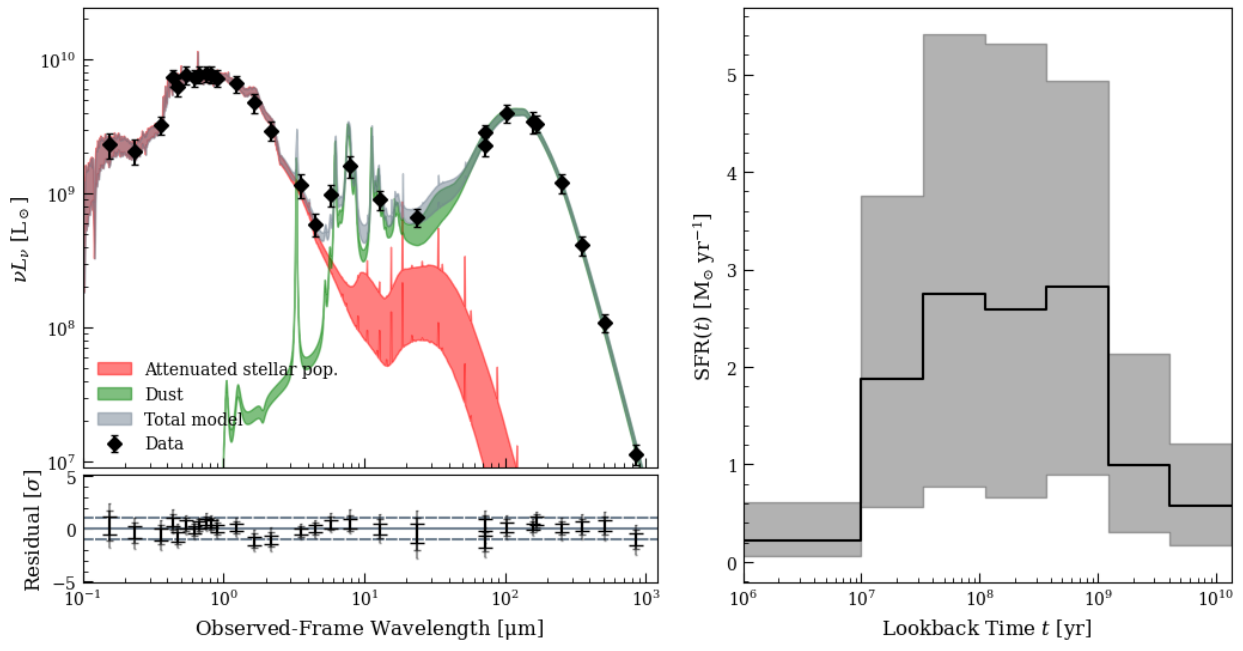
[35]:
```
fig = lgh.corner_plot(chain,
                      quantiles=(0.16, 0.50, 0.84),
                      smooth=1,
                      levels=None,
                      show_titles=True)

ZZ = np.linspace(0.001, 0.03, 30)
Zprior = lambda Z, mu, s: 1 / np.sqrt(2 * np.pi * s**2) * np.exp(-1 * (Z - mu)**2 / s**2)
axs = (np.array(fig.axes)).reshape(14,14)
yy = Zprior(ZZ, 0.014, 0.002)
axs[7,7].plot(ZZ, yy, color='forestgreen')
```

[35]:
```
[<matplotlib.lines.Line2D at 0x18a51f970>]
```



[36]:
```
from lightning.plots import sed_plot_morebayesian, sed_plot_delchi_morebayesian, sfh_plot

fig5 = plt.figure(figsize=(12,6))
ax51 = fig5.add_axes([0.1, 0.26, 0.4, 0.64])
ax52 = fig5.add_axes([0.1, 0.1, 0.4, 0.15])
ax53 = fig5.add_axes([0.56, 0.1, 0.34, 0.8])

fig5, ax51 = sed_plot_morebayesian(lgh, chain, logprob_chain,
                                   plot_components=True,
                                   ax=ax51,
                                   legend_kwargs={'loc': 'lower left', 'frameon': False})
ax51.set_xticklabels([])
fig5, ax52 = sed_plot_delchi_morebayesian(lgh, chain, logprob_chain, ax=ax52)
fig5, ax53 = lgh.sfh_plot(chain, ax=ax53)
```

```
/Users/eqm5663/Research/code/plightning/lightning/stellar/pegase.py:1007: RuntimeWarning:
↪ divide by zero encountered in log10
  np.log10(np.transpose(self.Lnu_obs, axes=[1,2,0,3])),
/Users/eqm5663/Research/code/plightning/lightning/stellar/pegase.py:1007: RuntimeWarning:
↪ divide by zero encountered in log10
  np.log10(np.transpose(self.Lnu_obs, axes=[1,2,0,3])),
/Users/eqm5663/miniconda3/envs/ciao-4.15/lib/python3.10/site-packages/numpy/lib/
↪nanfunctions.py:1217: RuntimeWarning: All-NaN slice encountered
  return function_base._ureduce(a, func=_nanmedian, keepdims=keepdims,
/Users/eqm5663/miniconda3/envs/ciao-4.15/lib/python3.10/site-packages/numpy/lib/
↪nanfunctions.py:1563: RuntimeWarning: All-NaN slice encountered
  return function_base._ureduce(a,
```



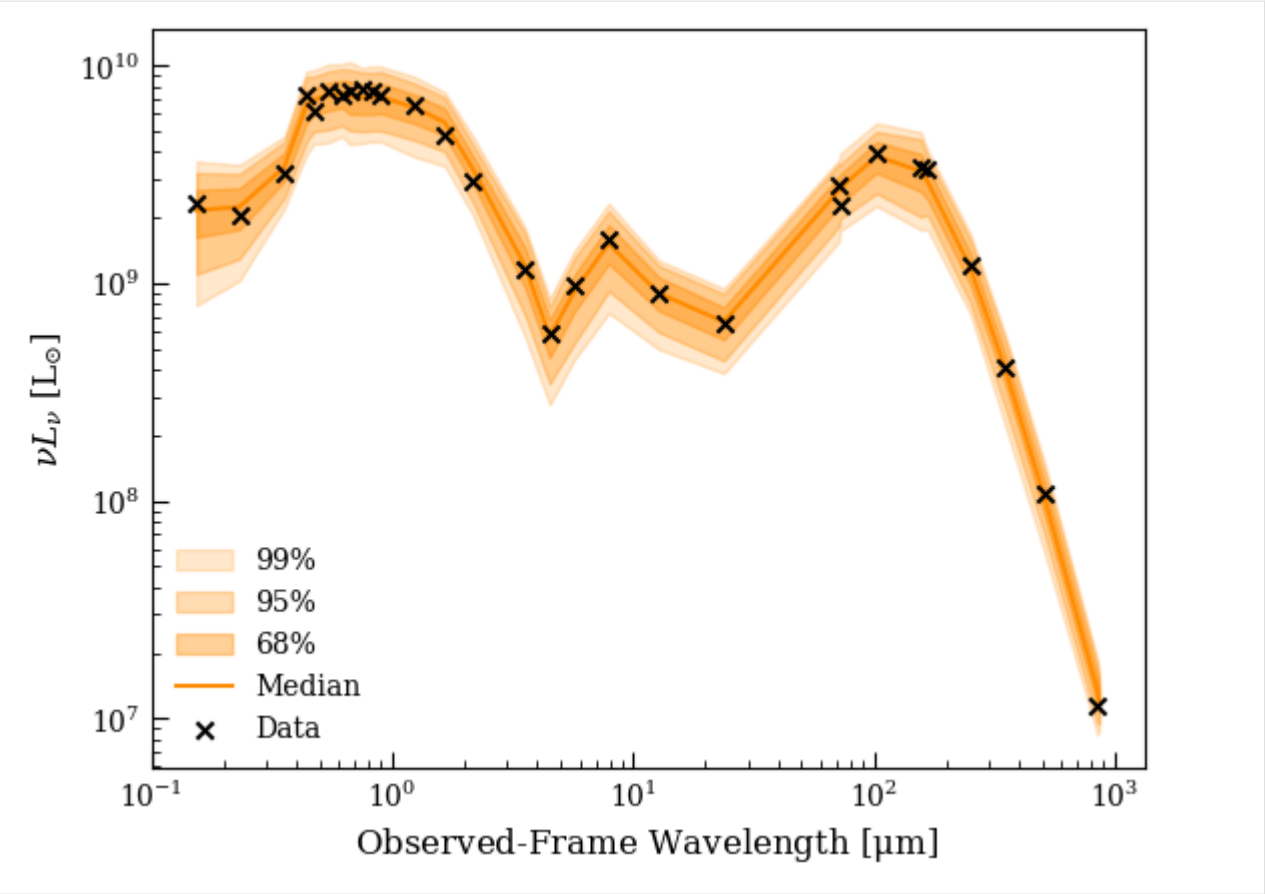
[9]:
```
from lightning.ppc import ppc, ppc_sed

pvalue, chi2_rep, chi2_obs = ppc(lgh, chain,
                                 logprob_chain,
```

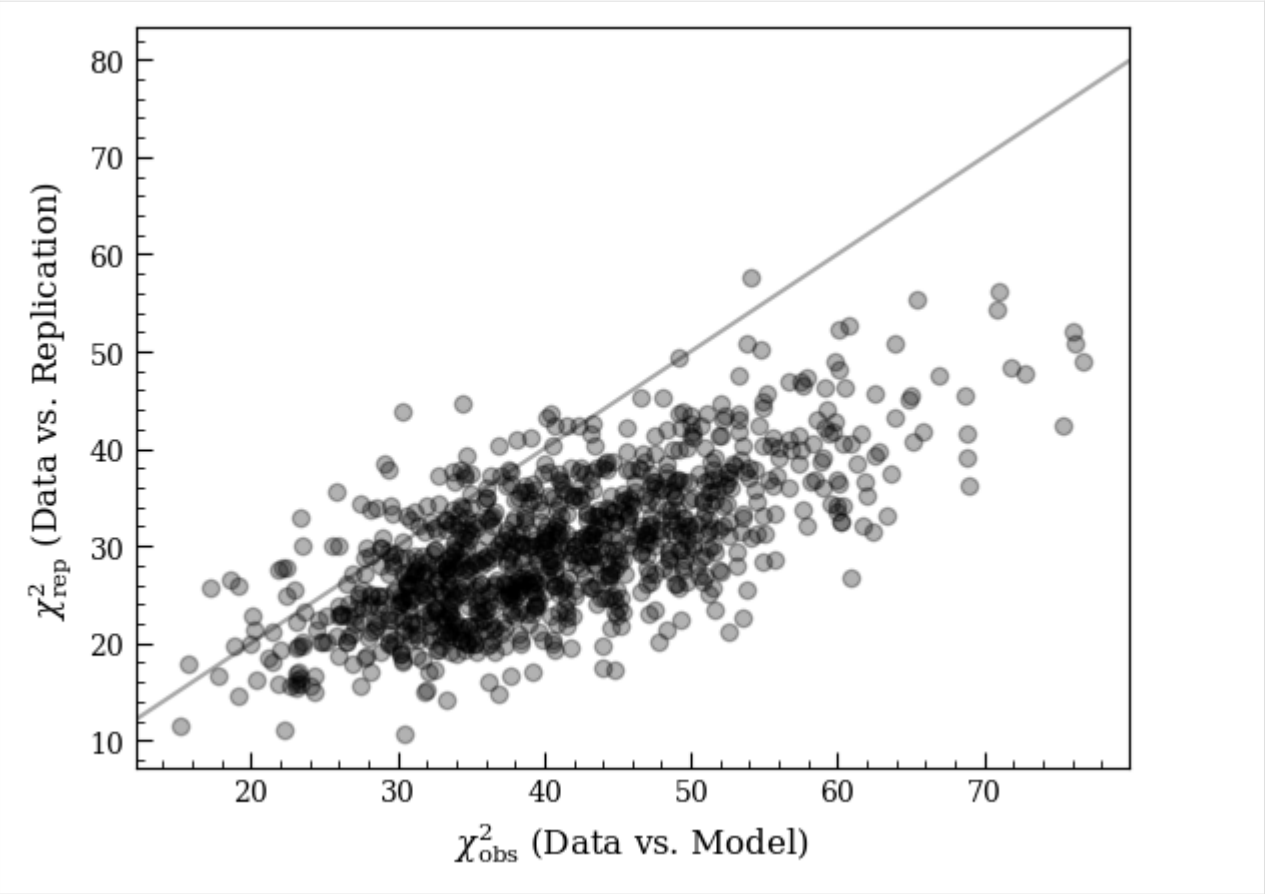
(continues on next page)

(continued from previous page)

```
                                Nrep=1000,
                                seed=12345)
fig, ax = ppc_sed(lgh, chain,
                  logprob_chain,
                  Nrep=1000,
                  seed=12345,
                  normalize=False)

fig2, ax2 = plt.subplots()

ax2.scatter(chi2_obs,
            chi2_rep,
            marker='o',
            alpha=0.3)

xlim = ax2.get_xlim()
ax2.plot(xlim, xlim, linestyle='-', color='darkgray', zorder=-1)
ax2.set_xlim(xlim)

ax2.set_xlabel(r'$\chi_{\rm obs}^2$ (Data vs. Model)')
ax2.set_ylabel(r'$\chi_{\rm rep}^2$ (Data vs. Replication)')


print('p = %.3f' % (pvalue))
```

```
p = 0.065
```


10^{10}
10^{9}
10^{8}
10^{7}
νL_ν [$L_\odot$]
99%
95%
68%
Median
Data
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
Observed-Frame Wavelength [μm]



[38]:
```
import corner
# Convenience functions I made to plot BPT-like diagnostic regions
from python_snippets import k06_NIIplot, k06_SIIplot, k06_OIplot

from astropy.table import Table
groves = Table.read('~/Research/010_PHANGS_Chandra/groves2023_nebula_catalog.fits')
mask = groves['gal_name'] == 'NGC0628'

OIIIHbeta_obs = np.log10(groves['OIII5006_FLUX_CORR'][mask] / groves['HB4861_FLUX_CORR
↪'][mask])
NIIHalpha_obs = np.log10(groves['NII6583_FLUX_CORR'][mask] / groves['HA6562_FLUX_CORR
↪'][mask])
SIIHalpha_obs = np.log10((groves['SII6716_FLUX_CORR'][mask] + groves['SII6730_FLUX_CORR
↪'][mask]) / groves['HA6562_FLUX_CORR'][mask])
OIHalpha_obs = np.log10(groves['OI6300_FLUX_CORR'][mask] / groves['HA6562_FLUX_CORR
↪'][mask])

# linelum, linelum_intr = lgh.stars.get_model_lines(lgh.sfh, chain[:,:7], chain[:,7:9])
linelum, linelum_intr = lgh.get_model_lines(chain)
# print(lineratios.shape)
# print(lineratios[0,:])

OIIImask = lgh.stars.line_labels == 'O__3_500684A'
Halphamask = lgh.stars.line_labels == 'H__1_656280A'
```

(continues on next page)

(continued from previous page)

```
Hbetamask = lgh.stars.line_labels == 'H__1_486132A'
NIImask = lgh.stars.line_labels == 'N__2_658345A'
SII6717mask = lgh.stars.line_labels == 'S__2_671644A'
SII6730mask = lgh.stars.line_labels == 'S__2_673082A'
OImask = lgh.stars.line_labels == 'BLND_630000A'

OIIIHbeta = np.log10(linelum[:,OIIImask] / linelum[:,Hbetamask])
NIIHalpha = np.log10(linelum[:,NIImask] / linelum[:,Halphamask])
SIIHalpha = np.log10((linelum[:,SII6717mask] + linelum[:,SII6730mask]) / linelum[:,
↪Halphamask])
OIHalpha = np.log10(linelum[:,OImask] / linelum[:,Halphamask])

OIIIHbeta = np.log10(linelum_intr[:,OIIImask] / linelum_intr[:,Hbetamask])
NIIHalpha = np.log10(linelum_intr[:,NIImask] / linelum_intr[:,Halphamask])
SIIHalpha = np.log10((linelum_intr[:,SII6717mask] + linelum_intr[:,SII6730mask]) /␣
↪linelum_intr[:,Halphamask])
OIHalpha = np.log10(linelum_intr[:,OImask] / linelum_intr[:,Halphamask])

fig, axs = plt.subplots(1,3, figsize=(12,4))

# print(np.median(OIIIHbeta))
# print(np.median(NIIHalpha))
# print(np.median(SIIHalpha))
# print(np.median(OIHalpha))

corner.hist2d(NIIHalpha, OIIIHbeta, smooth=1, color='darkorange', levels=[0.50, 0.95, 0.
↪99], ax=axs[0])
corner.hist2d(NIIHalpha_intr, OIIIHbeta_intr, smooth=1, color='dodgerblue', levels=[0.50,
↪ 0.95, 0.99], ax=axs[0])
k06_NIIplot(ax=axs[0])
axs[0].scatter(NIIHalpha_obs, OIIIHbeta_obs, marker='.', color='slategray', alpha=0.2,␣
↪label='Groves+(2023)')

# OIIIHbeta_M06 = np.log10(275 / 152.700)
# NIIHalpha_M06 = np.log10(4.7 / 455.00)
# axs[0].scatter(NIIHalpha_M06, OIIIHbeta_M06, marker='*', s=100, color='forestgreen',␣
↪label='Moustakas+(2006)')

axs[0].set_xlim(-2.2,1)
axs[0].set_ylim(-1,2)
axs[0].set_xlabel(r'$\rm \log([N II] / H\alpha)$')
axs[0].set_ylabel(r'$\rm \log([O III] / H\beta)$')
axs[0].legend(loc='best')

corner.hist2d(SIIHalpha, OIIIHbeta, smooth=1, color='darkorange', levels=[0.50, 0.95, 0.
↪99], ax=axs[1])
corner.hist2d(SIIHalpha_intr, OIIIHbeta_intr, smooth=1, color='dodgerblue', levels=[0.50,
↪ 0.95, 0.99], ax=axs[1])
axs[1].scatter(SIIHalpha_obs, OIIIHbeta_obs, marker='.', color='slategray', alpha=0.2)
k06_SIIplot(ax=axs[1])
axs[1].set_xlim(-1.2,0.8)
axs[1].set_ylim(-1,2)
```

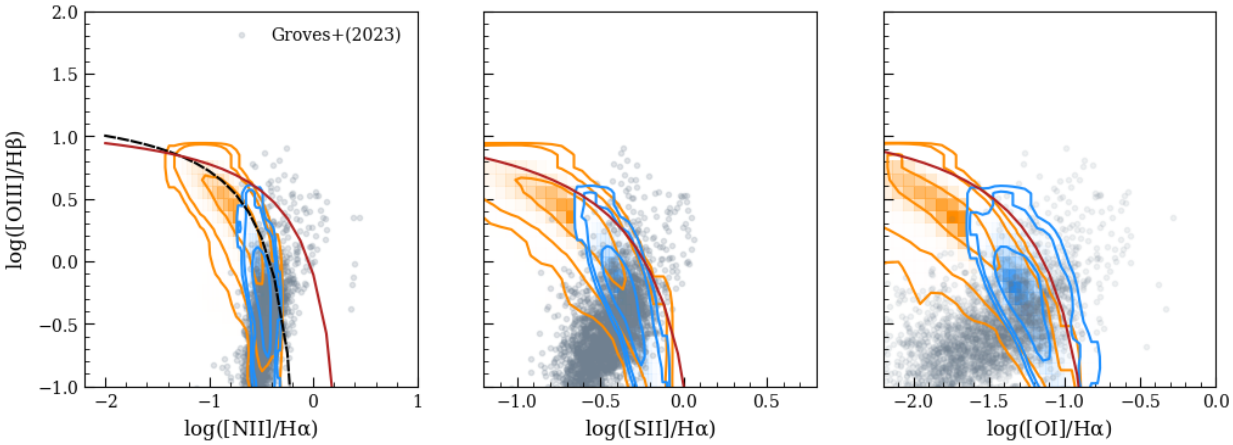
(continues on next page)

(continued from previous page)

```
axs[1].set_yticklabels([])
axs[1].set_xlabel(r'$\rm \log([S II] / H\alpha)$')

corner.hist2d(OIHalpha, OIIIHbeta, smooth=1, color='darkorange', levels=[0.50, 0.95, 0.
↪99], ax=axs[2])
corner.hist2d(OIHalpha_intr, OIIIHbeta_intr, smooth=1, color='dodgerblue', levels=[0.50,␣
↪0.95, 0.99], ax=axs[2])
axs[2].scatter(OIHalpha_obs, OIIIHbeta_obs, marker='.', color='slategray', alpha=0.1)
k06_OIplot(ax=axs[2])
axs[2].set_xlim(-2.2,0.0)
axs[2].set_ylim(-1,2)
axs[2].set_yticklabels([])
axs[2].set_xlabel(r'$\rm \log([O I] / H\alpha)$')
```

```
/var/folders/p6/qlk1ytxd09vf64nrq0vfw06wgg_lcz/T/ipykernel_76875/799415909.py:9:␣
↪RuntimeWarning: divide by zero encountered in log10
  OIIIHbeta_obs = np.log10(groves['OIII5006_FLUX_CORR'][mask] / groves['HB4861_FLUX_CORR
↪'][mask])
/var/folders/p6/qlk1ytxd09vf64nrq0vfw06wgg_lcz/T/ipykernel_76875/799415909.py:12:␣
↪RuntimeWarning: divide by zero encountered in log10
  OIHalpha_obs = np.log10(groves['OI6300_FLUX_CORR'][mask] / groves['HA6562_FLUX_CORR
↪'][mask])
/Users/eqm5663/Research/code/plightning/lightning/stellar/pegase.py:1215: RuntimeWarning:
↪ divide by zero encountered in log10
  np.log10(np.transpose(self.line_lum, axes=[1,2,0,3])),
```

[38]:
```
Text(0.5, 0, '$\\rm \\log([O I] / H\\alpha)$')
```



[ ]:

5.2 Line ratio grids

Produce and plot grids of line ratios for a variety of complex star formation histories.

5.2.1 Imports

[2]:
```
import numpy as np
import h5py
rng = np.random.default_rng()
import astropy.units as u
from astropy.table import Table
from corner import corner
import matplotlib.pyplot as plt
plt.style.use('ebm-dejavu')
%matplotlib inline

from lightning import Lightning
from lightning.priors import UniformPrior, NormalPrior
from lightning.plots import step_curve
from lightning.sfh import DelayedExponentialSFH
```

5.2.2 Setup

Create three Lightning objects, loading the Cloudy grids with and without dust grains, and with ULXs from Garofali+(2024) for BPASS stellar population models.

[5]:
```
filter_labels = ['SDSS_u'] # We aren't actually using these, 'None' just isn't an option

agebins = [0.0] + list(np.logspace(7, np.log10(13.4e9), 10))

lgh_nodust = Lightning(filter_labels,
                       lum_dist=15, # also isn't going to come up
                       ages=agebins,
                       nebula_lognH=2.0,
                       nebula_dust=False,
                       stellar_type='BPASS-A24',
                       SFH_type='Piecewise-Constant',
                       atten_type='Modified-Calzetti',
                       print_setup_time=True)
print()
lgh_dust = Lightning(filter_labels,
                     lum_dist=15,
                     ages=agebins,
                     nebula_lognH=2.0,
                     nebula_dust=True,
                     stellar_type='BPASS-A24',
                     SFH_type='Piecewise-Constant',
                     atten_type='Modified-Calzetti',
                     print_setup_time=True)
```

(continues on next page)

(continued from previous page)

```
print()
lgh_ULX = Lightning(filter_labels,
                    lum_dist=15,
                    ages=agebins,
                    nebula_lognH=2.0,
                    nebula_dust=True,
                    stellar_type='BPASS-ULX-G24',
                    SFH_type='Piecewise-Constant',
                    atten_type='Modified-Calzetti',
                    print_setup_time=True)
```

```
0.002 s elapsed in _get_filters
0.000 s elapsed in _get_wave_obs
1.105 s elapsed in stellar model setup
0.000 s elapsed in dust attenuation model setup
0.000 s elapsed in dust emission model setup
0.000 s elapsed in agn emission model setup
0.000 s elapsed in X-ray model setup
1.107 s elapsed total

0.001 s elapsed in _get_filters
0.000 s elapsed in _get_wave_obs
1.051 s elapsed in stellar model setup
0.000 s elapsed in dust attenuation model setup
0.000 s elapsed in dust emission model setup
0.000 s elapsed in agn emission model setup
0.000 s elapsed in X-ray model setup
1.052 s elapsed total

0.001 s elapsed in _get_filters
0.000 s elapsed in _get_wave_obs
0.997 s elapsed in stellar model setup
0.000 s elapsed in dust attenuation model setup
0.000 s elapsed in dust emission model setup
0.000 s elapsed in agn emission model setup
0.000 s elapsed in X-ray model setup
0.998 s elapsed total
```

5.2.3 Define SFH templates

For simplicity we'll convolve a delayed exponential SFH with the age bins.

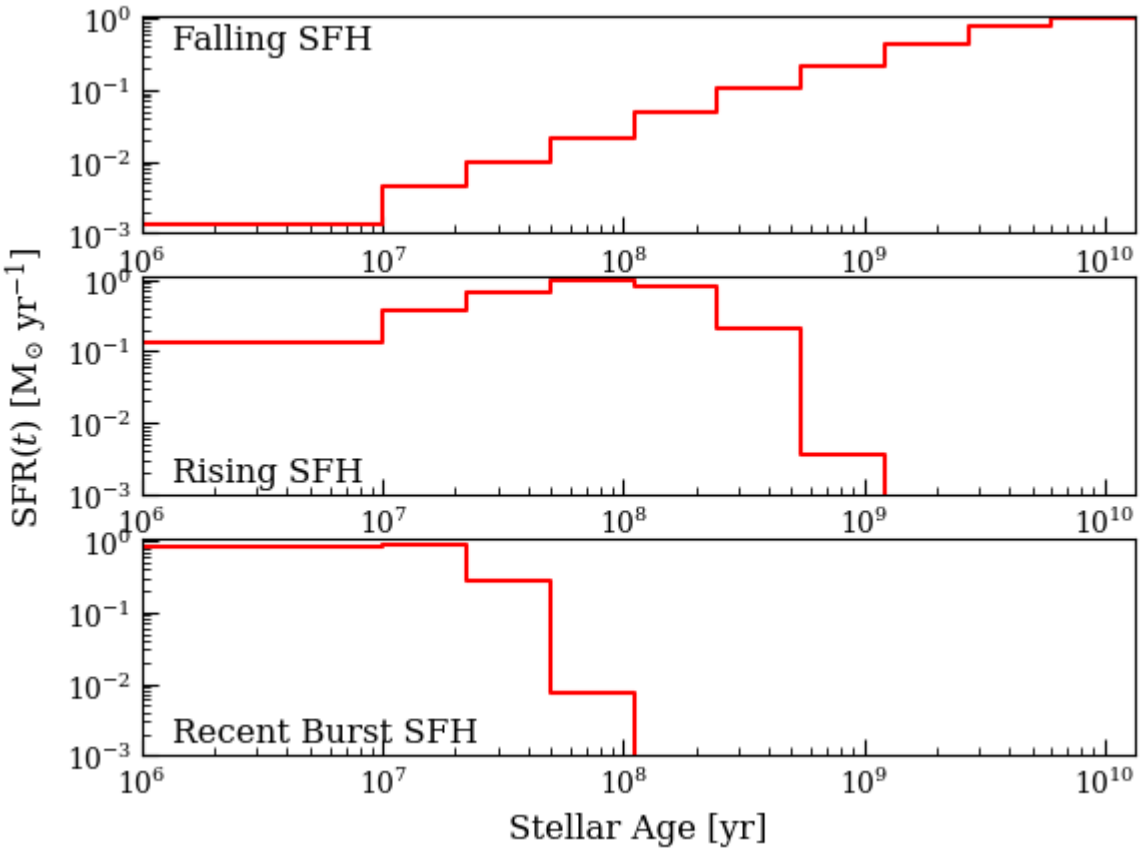
[4]:
```
agemid = 0.5 * np.array(agebins[:-1]) + 0.5 * np.array(agebins[1:])
sfh_exp = DelayedExponentialSFH(agemid)


falling_sfh =  sfh_exp.evaluate(np.array([np.exp(1), 1e10]))
rising_sfh = sfh_exp.evaluate(np.array([np.exp(1), 1e8]))
recent_sfh = sfh_exp.evaluate(np.array([np.exp(1), 1e7]))

fig, axs = plt.subplots(3,1)
```

(continues on next page)

(continued from previous page)

```
x,y = step_curve(agebins, falling_sfh)
axs[0].plot(x,y, color='red')

# x,y = step_curve(agebins, rising_sfh)
# axs[1].plot(x,y)
x,y = step_curve(agebins, rising_sfh)
axs[1].plot(x,y, color='red')

# x,y = step_curve(agebins, recent_sfh)
# axs[2].plot(x,y)
x,y = step_curve(agebins, recent_sfh)
axs[2].plot(x,y, color='red')

for i in [0,1,2]:
    axs[i].set_xscale('log')
    axs[i].set_yscale('log')
    axs[i].set_xlim(1e6, 13.4e9)
    axs[i].set_ylim(1e-3, 1.1)

axs[0].text(0.03, 0.97, 'Falling SFH', ha='left', va='top', transform=axs[0].transAxes)
axs[1].text(0.03, 0.03, 'Rising SFH', ha='left', va='bottom', transform=axs[1].transAxes)
axs[2].text(0.03, 0.03, 'Recent Burst SFH', ha='left', va='bottom', transform=axs[2].
↪transAxes)


axs[2].set_xlabel('Stellar Age [yr]')
axs[1].set_ylabel(r'${\rm SFR}(t)~[\rm M_{\odot}~yr^{-1}]$')
```

[4]:
```
Text(0, 0.5, '${\\rm SFR}(t)~[\\rm M_{\\odot}~yr^{-1}]$')
```



And now we'll derive grids of line ratios for our various faked SFHs:

5.2.4 Dusty grid

[29]:
```
from python_snippets import k06_NIIplot

Npoints = 10

logU_grid = np.linspace(-4, -1.5, Npoints)
Z_grid = np.logspace(np.log10(0.00075), np.log10(0.02), Npoints)

tauV_grid = np.array([0.0])

fig, axs = plt.subplots(2,3, figsize=(12,8))

linestyle=['-','--']

OIIImask = lgh_dust.stars.line_labels == 'O__3_500684A'
Halphamask = lgh_dust.stars.line_labels == 'H__1_656280A'
Hbetamask = lgh_dust.stars.line_labels == 'H__1_486132A'
NIImask = lgh_dust.stars.line_labels == 'N__2_658345A'

HeIImask = lgh_dust.stars.line_labels == 'HE_2_468568A'
```

(continues on next page)

(continued from previous page)

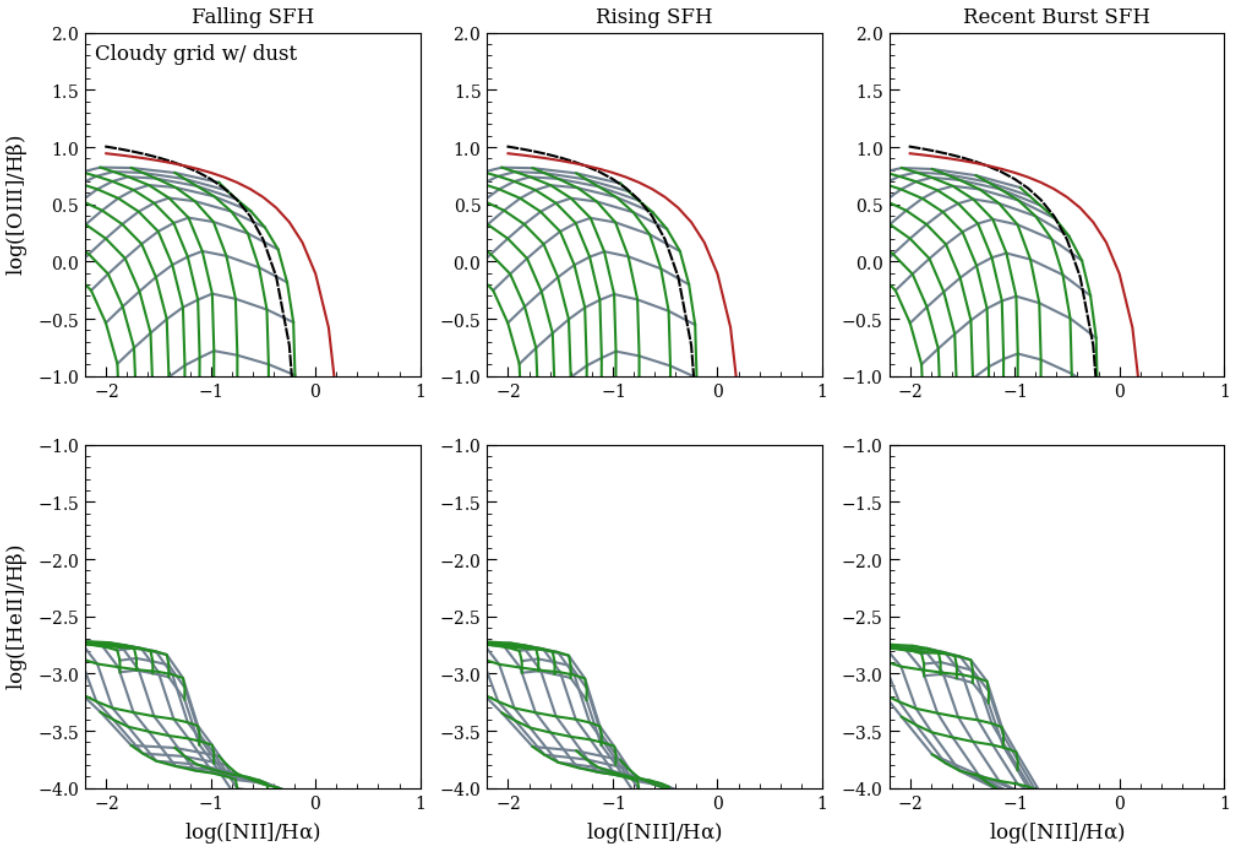
```
for m,sfh in enumerate([falling_sfh, rising_sfh, recent_sfh]):
    for k,tauV in enumerate(tauV_grid):

        param_array = np.zeros(15)
        param_array[:10] = sfh
        param_array[-3] = tauV

        line_grid = np.zeros((Npoints,Npoints,len(lgh_nodust.stars.line_labels)))

        for i,logU in enumerate(logU_grid):
            for j,Z in enumerate(Z_grid):

                param_array[10] = Z
                param_array[11] = logU

                lines_ext, lines_intr = lgh_dust.get_model_lines(param_array)

                line_grid[i,j,:] = lines_ext.flatten()

        o3hbeta_grid = np.log10(line_grid[:,:,OIIImask] / line_grid[:,:,Hbetamask])
        n2halpha_grid = np.log10(line_grid[:,:,NIImask] / line_grid[:,:,Halphamask])
        he2hbeta_grid =  np.log10(line_grid[:,:,HeIImask] / line_grid[:,:,Hbetamask])

        for i,_ in enumerate(logU_grid):
            axs[0,m].plot(n2halpha_grid[i,:], o3hbeta_grid[i,:], color='slategray',␣
↪linestyle=linestyle[k])
            axs[1,m].plot(n2halpha_grid[i,:], he2hbeta_grid[i,:], color='slategray',␣
↪linestyle=linestyle[k])

        for j,_ in enumerate(Z_grid):
            axs[0,m].plot(n2halpha_grid[:,j], o3hbeta_grid[:,j], color='forestgreen',␣
↪linestyle=linestyle[k])
            axs[1,m].plot(n2halpha_grid[:,j], he2hbeta_grid[:,j], color='forestgreen',␣
↪linestyle=linestyle[k])

        k06_NIIplot(ax=axs[0,m])
        axs[0,m].set_xlim(-2.2,1)
        axs[0,m].set_ylim(-1,2)

        axs[1,m].set_xlim(-2.2,1)
        axs[1,m].set_ylim(-4,-1)

axs[0,0].set_title('Falling SFH')
axs[0,1].set_title('Rising SFH')
axs[0,2].set_title('Recent Burst SFH')

axs[0,0].set_ylabel(r'$\rm \log([O III] / H\beta)$')
axs[1,0].set_ylabel(r'$\rm \log([He II] / H\beta)$')
for m in [0,1,2]:
    axs[1, m].set_xlabel(r'$\rm \log([N II] / H\alpha)$')
```

(continues on next page)

(continued from previous page)

```
axs[0,0].text(0.03, 0.97, 'Cloudy grid w/ dust', ha='left', va='top', transform=axs[0,0].
↪transAxes)
```

[29]:
```
Text(0.03, 0.97, 'Cloudy grid w/ dust')
```



5.2.5 Grid without dust

[28]:
```
from python_snippets import k06_NIIplot

Npoints = 10

logU_grid = np.linspace(-4, -1.5, Npoints)
Z_grid = np.logspace(np.log10(0.00075), np.log10(0.02), Npoints)

tauV_grid = np.array([0.0])

fig, axs = plt.subplots(2,3, figsize=(12,8))

linestyle=['-','--']

OIIImask = lgh_nodust.stars.line_labels == 'O__3_500684A'
Halphamask = lgh_nodust.stars.line_labels == 'H__1_656280A'
Hbetamask = lgh_nodust.stars.line_labels == 'H__1_486132A'
```

(continues on next page)

(continued from previous page)

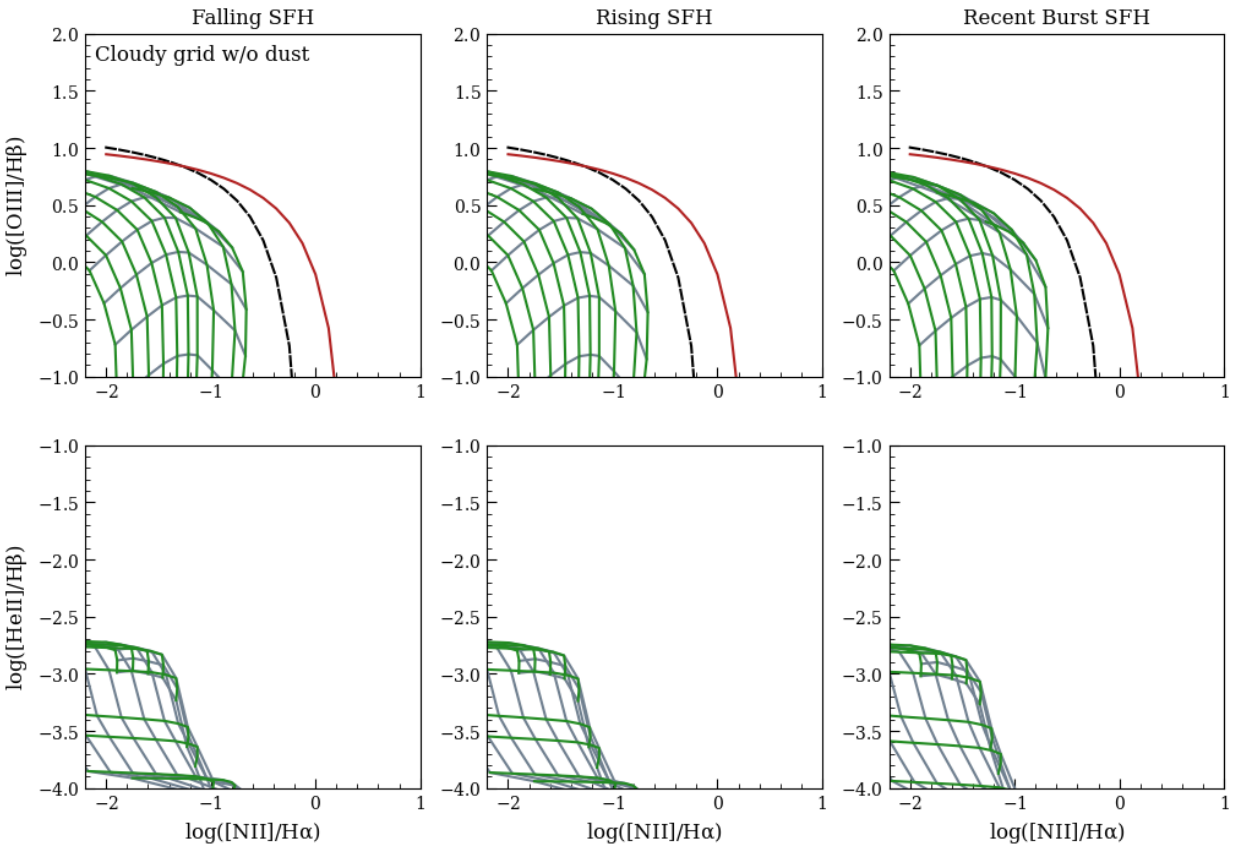
```
NIImask = lgh_nodust.stars.line_labels == 'N__2_658345A'

HeIImask = lgh_nodust.stars.line_labels == 'HE_2_468568A'

for m,sfh in enumerate([falling_sfh, rising_sfh, recent_sfh]):
    for k,tauV in enumerate(tauV_grid):

        param_array = np.zeros(15)
        param_array[:10] = sfh
        param_array[-3] = tauV

        line_grid = np.zeros((Npoints,Npoints,len(lgh_nodust.stars.line_labels)))

        for i,logU in enumerate(logU_grid):
            for j,Z in enumerate(Z_grid):

                param_array[10] = Z
                param_array[11] = logU

                lines_ext, lines_intr = lgh_nodust.get_model_lines(param_array)

                line_grid[i,j,:] = lines_ext.flatten()

        o3hbeta_grid = np.log10(line_grid[:,:,OIIImask] / line_grid[:,:,Hbetamask])
        n2halpha_grid = np.log10(line_grid[:,:,NIImask] / line_grid[:,:,Halphamask])
        he2hbeta_grid =  np.log10(line_grid[:,:,HeIImask] / line_grid[:,:,Hbetamask])

        for i,_ in enumerate(logU_grid):
            axs[0,m].plot(n2halpha_grid[i,:], o3hbeta_grid[i,:], color='slategray',␣
↪linestyle=linestyle[k])
            axs[1,m].plot(n2halpha_grid[i,:], he2hbeta_grid[i,:], color='slategray',␣
↪linestyle=linestyle[k])

        for j,_ in enumerate(Z_grid):
            axs[0,m].plot(n2halpha_grid[:,j], o3hbeta_grid[:,j], color='forestgreen',␣
↪linestyle=linestyle[k])
            axs[1,m].plot(n2halpha_grid[:,j], he2hbeta_grid[:,j], color='forestgreen',␣
↪linestyle=linestyle[k])

        k06_NIIplot(ax=axs[0,m])
        axs[0,m].set_xlim(-2.2,1)
        axs[0,m].set_ylim(-1,2)

        axs[1,m].set_xlim(-2.2,1)
        axs[1,m].set_ylim(-4,-1)

axs[0,0].set_title('Falling SFH')
axs[0,1].set_title('Rising SFH')
axs[0,2].set_title('Recent Burst SFH')

axs[0,0].set_ylabel(r'$\rm \log([O III] / H\beta)$')
axs[1,0].set_ylabel(r'$\rm \log([He II] / H\beta)$')
```

(continues on next page)

(continued from previous page)

```
for m in [0,1,2]:
    axs[1, m].set_xlabel(r'$\rm \log([N II] / H\alpha)$')

axs[0,0].text(0.03, 0.97, 'Cloudy grid w/o dust', ha='left', va='top', transform=axs[0,
↪0].transAxes)
```

[28]:
```
Text(0.03, 0.97, 'Cloudy grid w/o dust')
```



5.2.6 Grid with ULXs

[25]:
```
from python_snippets import k06_NIIplot

Npoints = 10

logU_grid = np.linspace(-4, -1.5, Npoints)
Z_grid = np.logspace(np.log10(lgh_ULX.stars.param_bounds[0,0]), np.log10(lgh_ULX.stars.
↪param_bounds[0,1]), Npoints)

tauV_grid = np.array([0.0])

fig, axs = plt.subplots(2,3, figsize=(12,8))

linestyle=['-','--']
```

(continues on next page)

(continued from previous page)

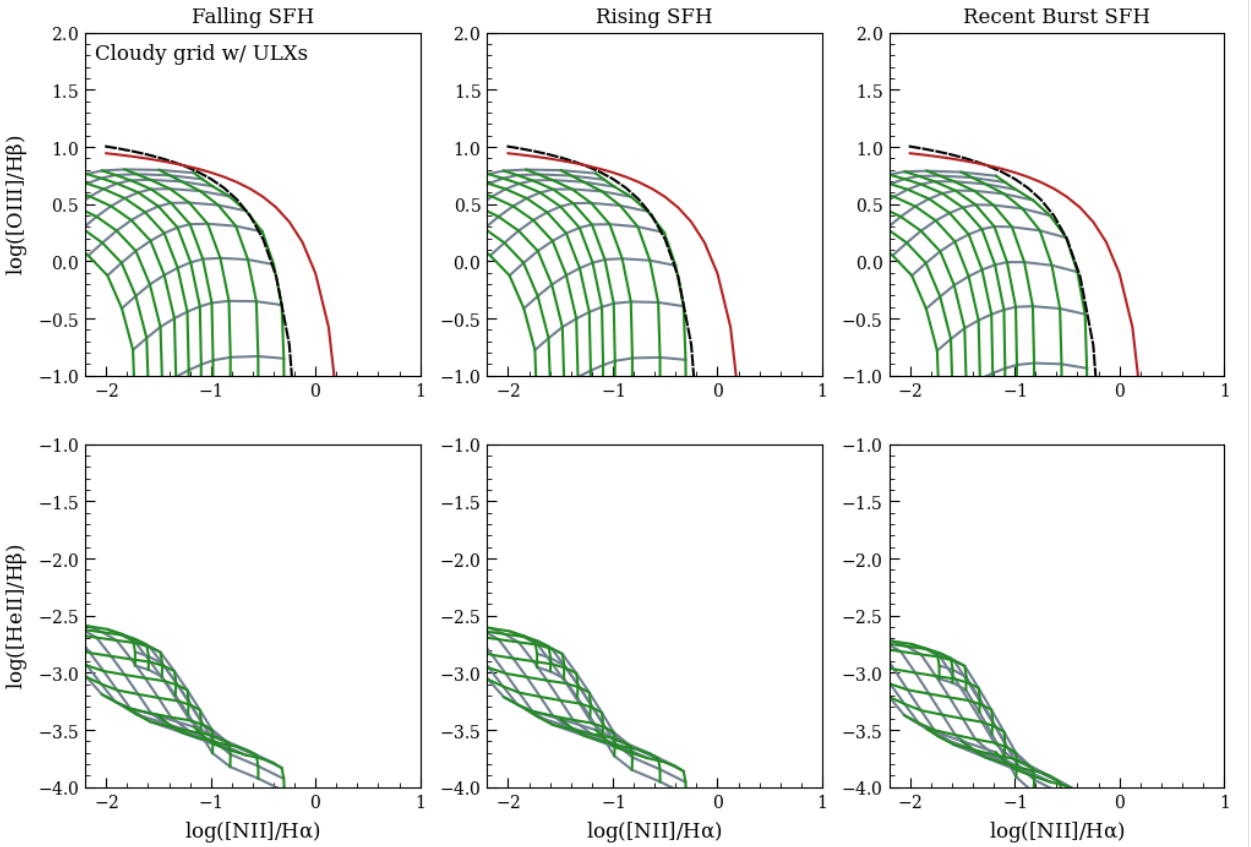
```
OIIImask = lgh_ULX.stars.line_labels == 'O__3_500684A'
Halphamask = lgh_ULX.stars.line_labels == 'H__1_656280A'
Hbetamask = lgh_ULX.stars.line_labels == 'H__1_486132A'
NIImask = lgh_ULX.stars.line_labels == 'N__2_658345A'

HeIImask = lgh_ULX.stars.line_labels == 'HE_2_468568A'

for m,sfh in enumerate([falling_sfh, rising_sfh, recent_sfh]):
    for k,tauV in enumerate(tauV_grid):

        param_array = np.zeros(15)
        param_array[:10] = sfh
        param_array[-3] = tauV

        line_grid = np.zeros((Npoints,Npoints,len(lgh_nodust.stars.line_labels)))

        for i,logU in enumerate(logU_grid):
            for j,Z in enumerate(Z_grid):

                param_array[10] = Z
                param_array[11] = logU

                lines_ext, lines_intr = lgh_ULX.get_model_lines(param_array)

                line_grid[i,j,:] = lines_ext.flatten()

        o3hbeta_grid = np.log10(line_grid[:,:,OIIImask] / line_grid[:,:,Hbetamask])
        n2halpha_grid = np.log10(line_grid[:,:,NIImask] / line_grid[:,:,Halphamask])
        he2hbeta_grid =  np.log10(line_grid[:,:,HeIImask] / line_grid[:,:,Hbetamask])

        for i,_ in enumerate(logU_grid):
            axs[0,m].plot(n2halpha_grid[i,:], o3hbeta_grid[i,:], color='slategray',␣
↪linestyle=linestyle[k])
            axs[1,m].plot(n2halpha_grid[i,:], he2hbeta_grid[i,:], color='slategray',␣
↪linestyle=linestyle[k])

        for j,_ in enumerate(Z_grid):
            axs[0,m].plot(n2halpha_grid[:,j], o3hbeta_grid[:,j], color='forestgreen',␣
↪linestyle=linestyle[k])
            axs[1,m].plot(n2halpha_grid[:,j], he2hbeta_grid[:,j], color='forestgreen',␣
↪linestyle=linestyle[k])

        k06_NIIplot(ax=axs[0,m])
        axs[0,m].set_xlim(-2.2,1)
        axs[0,m].set_ylim(-1,2)

        axs[1,m].set_xlim(-2.2,1)
        axs[1,m].set_ylim(-4,-1)

axs[0,0].set_title('Falling SFH')
axs[0,1].set_title('Rising SFH')
```

(continues on next page)

(continued from previous page)

```
axs[0,2].set_title('Recent Burst SFH')

axs[0,0].set_ylabel(r'$\rm \log([O III] / H\beta)$')
axs[1,0].set_ylabel(r'$\rm \log([He II] / H\beta)$')
for m in [0,1,2]:
    axs[1, m].set_xlabel(r'$\rm \log([N II] / H\alpha)$')

axs[0,0].text(0.03, 0.97, 'Cloudy grid w/ ULXs', ha='left', va='top', transform=axs[0,0].
↪transAxes)
```

[25]:
```
Text(0.03, 0.97, 'Cloudy grid w/ ULXs')
```



[10]:
```
lgh_ULX.print_params(verbose=True)
```

```
============================
Piecewise-Constant
============================
Parameter  Lo  Hi               Description
--------- --- --- -------------------------
    psi_1 0.0 inf  SFR in stellar age bin 1
    psi_2 0.0 inf  SFR in stellar age bin 2
    psi_3 0.0 inf  SFR in stellar age bin 3
    psi_4 0.0 inf  SFR in stellar age bin 4
    psi_5 0.0 inf  SFR in stellar age bin 5
```

(continues on next page)

(continued from previous page)

```
    psi_6 0.0 inf  SFR in stellar age bin 6
    psi_7 0.0 inf  SFR in stellar age bin 7
    psi_8 0.0 inf  SFR in stellar age bin 8
    psi_9 0.0 inf  SFR in stellar age bin 9
   psi_10 0.0 inf SFR in stellar age bin 10

============================
BPASS-Stellar-A24
============================
Parameter                   Lo                   Hi                                  ␣
↪             Description
--------- -------------------- -------------------- -----------------------------------
↪------------------------
     Zmet 0.00107151930523760 0.016982436524617443 Metallicity (mass fraction, where␣
↪solar = 0.020 ~ 10**[-1.7])
     logU                -4.0                 -1.5                                log10 of␣
↪the ionization parameter

============================
Modified-Calzetti
============================
      Parameter   Lo                 Hi                                            ␣
↪                            Description
--------------- ---- ------------------ ----------------------------------------------
↪-----------------------------------------
mcalz_tauV_diff  0.0                inf                                            ␣
↪         Optical depth of the diffuse ISM
    mcalz_delta -inf 0.4473684210526316 Deviation from the Calzetti+2000 UV power law␣
↪slope (Upper limit set by requiring Eb >= 0)
  mcalz_tauV_BC  0.0                inf                                Optical depth␣
↪of the birth cloud in star forming regions

Total parameters: 15
```

[14]:
```
np.log10(0.00107)
```

[14]:
```
-2.9706162223147903
```

[15]:
```
10**-2.97
```

[15]:
```
0.001071519305237606
```

[ ]:
```

```

The following notebooks show the variation of the the legacy PEGASE models, the BPASS models, the AGN models, and dust models, and are intended to give a quick reference as to where you may or may not have data to constrain a given model.

5.3 Stellar & Nebular Emission - PEGASE

With our Pégase SPS models nebular extinction and continuum emission are essentially built into the stellar emission model.

5.3.1 Imports

[2]:
```
import numpy as np
from scipy.interpolate import interp1d
from lightning.stellar import PEGASEModel as StellarModel
from lightning.sfh import PiecewiseConstSFH
import matplotlib.pyplot as plt
import matplotlib as mpl
plt.style.use('ebm-dejavu')
%matplotlib inline
```

5.3.2 Initialize Model

Here we'll initialize our sort of 'default' stellar population model, which is integrated over stellar age bins to be used with a piecewise-constant SFH. We'll do it twice, with and without the nebular component, to compare.

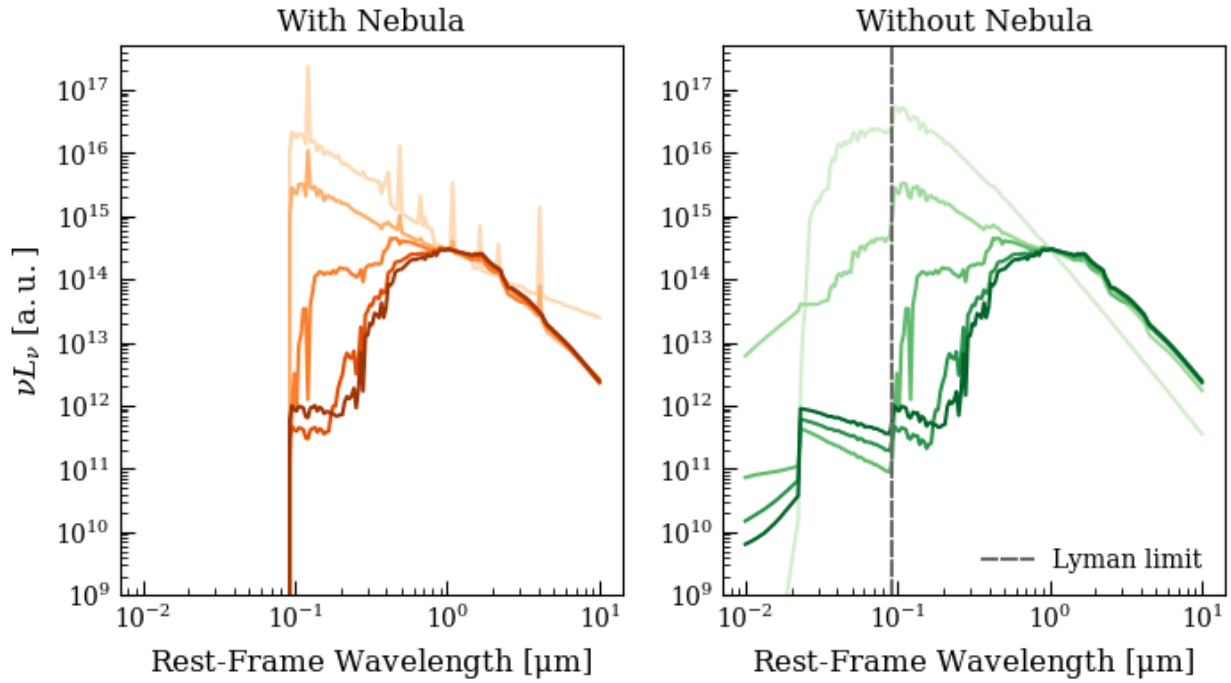
[3]:
```
wave_grid = np.logspace(np.log10(0.01),
                        np.log10(10),
                        200)
filter_labels = ['SDSS_u', 'SDSS_g', 'SDSS_r', 'SDSS_i', 'SDSS_z',
                 'MOIRCS_J', 'MOIRCS_H', 'MOIRCS_Ks',
                 'IRAC_CH1', 'IRAC_CH2', 'IRAC_CH3', 'IRAC_CH4']
redshift = 0.0
age = [0, 1e6, 1e8, 1e9, 5e9, 13.6e9]

stars_neb = StellarModel(filter_labels,
                         age=age,
                         redshift=redshift,
                         wave_grid=wave_grid,
                         nebular_effects=True)

stars_noneb = StellarModel(filter_labels,
                           age=age,
                           redshift=redshift,
                           wave_grid=wave_grid,
                           nebular_effects=False)
```

5.3.3 Simple Stellar Population Models

We've included the left-hand panel of this plot in basically every `Lightning`-related paper for years.

[4]:
```
fig, axs = plt.subplots(1,2, figsize=(8,4))

Nmod = len(age) - 1

cm_neb = mpl.colormaps['Oranges']
colors_neb = cm_neb(np.linspace(0.2, 0.9, Nmod))

cm_noneb = mpl.colormaps['Greens']
colors_noneb = cm_noneb(np.linspace(0.2, 0.9, Nmod))

for i in np.arange(Nmod):

    finterp = interp1d(stars_neb.wave_grid_rest, stars_neb.Lnu_obs[i,:])
    f1 = finterp(1)

    axs[0].plot(stars_neb.wave_grid_rest,
                stars_neb.nu_grid_obs * stars_neb.Lnu_obs[i,:] / f1,
                color=colors_neb[i])

    finterp = interp1d(stars_noneb.wave_grid_rest, stars_noneb.Lnu_obs[i,:])
    f1 = finterp(1)

    axs[1].plot(stars_noneb.wave_grid_rest,
                stars_noneb.nu_grid_obs * stars_noneb.Lnu_obs[i,:] / f1,
                color=colors_noneb[i])

axs[0].set_xscale('log')
axs[0].set_yscale('log')
axs[0].set_ylim(1e9, 5e17)

axs[0].set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
axs[0].set_ylabel(r'$\nu L_\nu\ [\rm a.u.]$')
axs[0].set_title('With Nebula')

axs[1].set_xscale('log')
axs[1].set_yscale('log')
axs[1].set_ylim(1e9, 5e17)
axs[1].axvline(0.0912, color='dimgray', linestyle='--', label='Lyman limit')
axs[1].legend(loc='lower right')

axs[1].set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
axs[1].set_title('Without Nebula')
```

[4]:
```
Text(0.5, 1.0, 'Without Nebula')
```



Where darker shades represent older ages. Note that nebular emission lines are only present for the two models with ages 100 Myr. The effect of free-free nebular continuum emission can be seen by comparing the lightest yellow curve on the left with the lightest green curve on the right.

5.3.4 Composite Stellar Population Models

To construct composite stellar populations we must of course assume a SFH for the population. Since we binned our simple stellar populations, we must use the `PiecewiseConstantSFH` model.

[5]:
```
sfh = PiecewiseConstSFH(age)
```

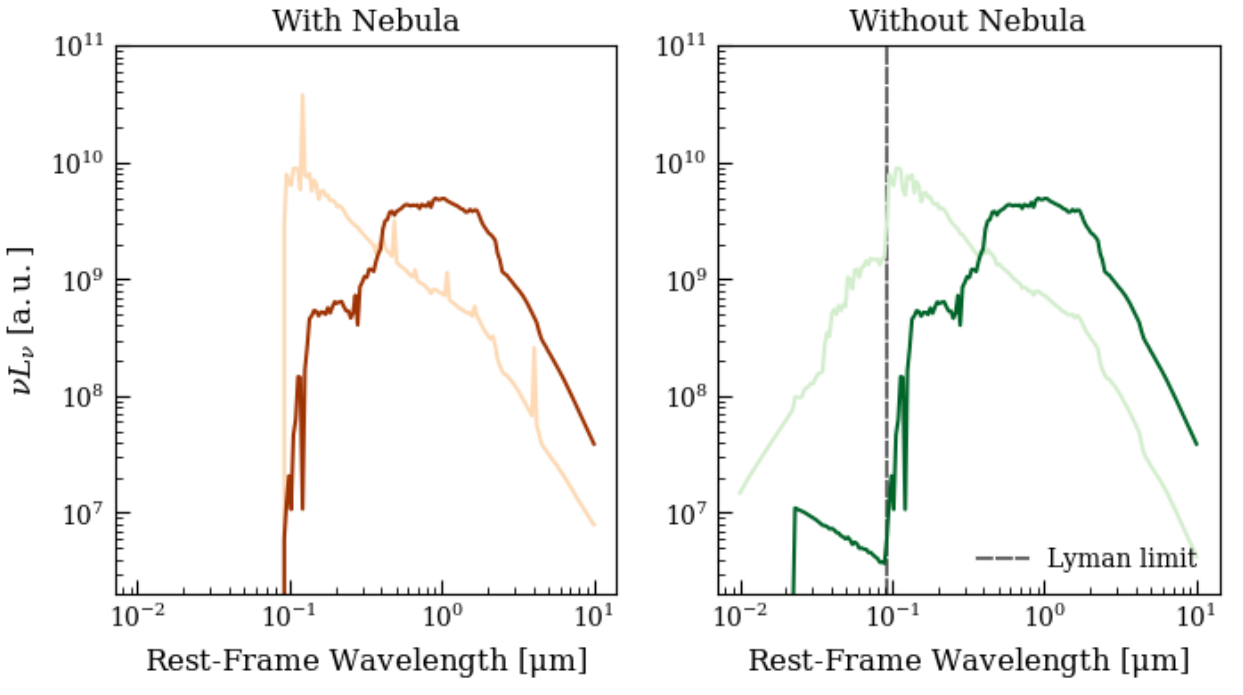
[15]:
```
fig, axs = plt.subplots(1,2, figsize=(8,4))

coeffs= np.array([[1,1,0,0,0],
                  [0,0,1,1,1]])

Nmod = coeffs.shape[0]

cm_neb = mpl.colormaps['Oranges']
colors_neb = cm_neb(np.linspace(0.2, 0.9, Nmod))

cm_noneb = mpl.colormaps['Greens']
colors_noneb = cm_noneb(np.linspace(0.2, 0.9, Nmod))

lnu_hires_neb,_,_ = stars_neb.get_model_lnu_hires(sfh, coeffs)
lnu_hires_noneb,_,_ = stars_noneb.get_model_lnu_hires(sfh, coeffs)

for i in np.arange(Nmod):
```

(continues on next page)

(continued from previous page)

```
    axs[0].plot(stars_neb.wave_grid_rest,
                stars_neb.nu_grid_obs * lnu_hires_neb[i,:],
                color=colors_neb[i])

    axs[1].plot(stars_noneb.wave_grid_rest,
                stars_noneb.nu_grid_obs * lnu_hires_noneb[i,:],
                color=colors_noneb[i])

axs[0].set_xscale('log')
axs[0].set_yscale('log')
axs[0].set_ylim(2e6, 1e11)

axs[0].set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
axs[0].set_ylabel(r'$\nu L_\nu\ [\rm a.u.]$')
axs[0].set_title('With Nebula')

axs[1].set_xscale('log')
axs[1].set_yscale('log')
axs[1].set_ylim(2e6, 1e11)
axs[1].axvline(0.0912, color='dimgray', linestyle='--', label='Lyman limit', zorder=-1)
axs[1].legend(loc='lower right')

axs[1].set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
axs[1].set_title('Without Nebula')
```

[15]:
```
Text(0.5, 1.0, 'Without Nebula')
```



Here the lighter colored populations are star forming, and the darker ones are quiescent. Note that even if you were to

use stellar populations without nebular extinction for your SED fitting in Lightning, the resulting galaxy would have no Lyman continuum leakage, as our ISM attenuation models are defined to be opaque to Lyman continuum radiation.

[ ]:

5.4 Stellar & Nebular Emission - BPASS

In the BPASS SPS models nebular extinction and continuum emission are added using Cloudy.

[1]:
```
import numpy as np
from scipy.interpolate import interp1d
from lightning.stellar import BPASSModel, PEGASEModel
from lightning.sfh import PiecewiseConstSFH
import matplotlib.pyplot as plt
import matplotlib as mpl
plt.style.use('ebm-dejavu')
%matplotlib inline
```

5.4.1 Initialize Model

Here we'll initialize our sort of 'default' stellar population model, which is integrated over stellar age bins to be used with a piecewise-constant SFH. We'll do it three times, with and without binary stellar evolution and the nebular component, to compare.

[2]:
```
wave_grid = np.logspace(np.log10(0.01),
                        np.log10(10),
                        200)
filter_labels = ['SDSS_u', 'SDSS_g', 'SDSS_r', 'SDSS_i', 'SDSS_z',
                 'MOIRCS_J', 'MOIRCS_H', 'MOIRCS_Ks',
                 'IRAC_CH1', 'IRAC_CH2', 'IRAC_CH3', 'IRAC_CH4']
redshift = 0.0
age = [0, 1e7, 1e8, 1e9, 5e9, 13.4e9]

stars_sin = BPASSModel(filter_labels,
                       age=age,
                       redshift=redshift,
                       wave_grid=wave_grid,
                       binaries=False,
                       nebular_effects=False)

stars_neb = BPASSModel(filter_labels,
                       age=age,
                       redshift=redshift,
                       wave_grid=wave_grid,
                       binaries=True,
                       nebular_effects=True)

stars_noneb = BPASSModel(filter_labels,
                         age=age,
                         redshift=redshift,
```

(continues on next page)

(continued from previous page)

```
                          wave_grid=wave_grid,
                          binaries=True,
                          nebular_effects=False)
```

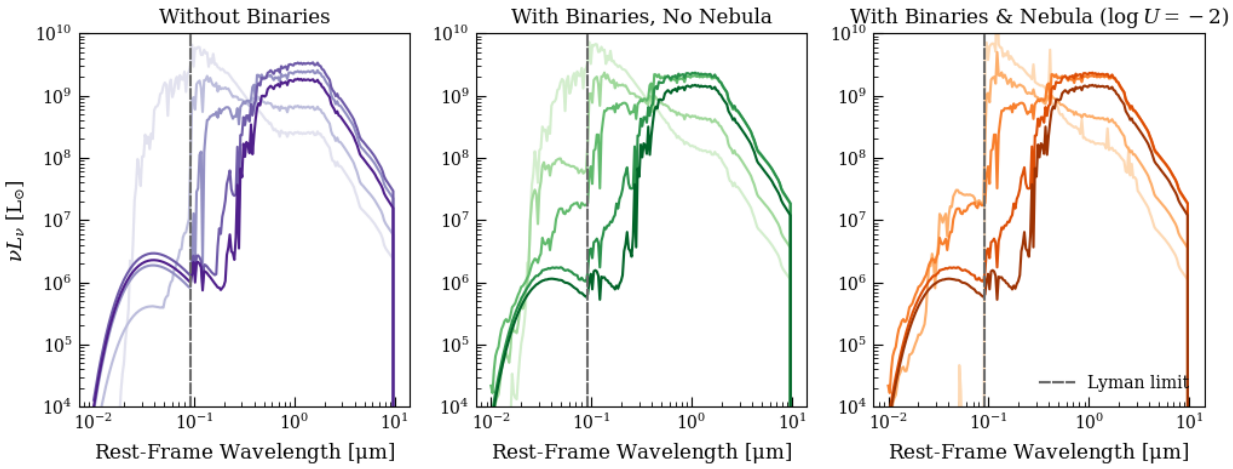
[14]:
```
fig, axs = plt.subplots(1,3, figsize=(12,4))

Nmod = len(age) - 1

cm_sin = mpl.colormaps['Purples']
colors_sin = cm_sin(np.linspace(0.2, 0.9, Nmod))

cm_neb = mpl.colormaps['Oranges']
colors_neb = cm_neb(np.linspace(0.2, 0.9, Nmod))

cm_noneb = mpl.colormaps['Greens']
colors_noneb = cm_noneb(np.linspace(0.2, 0.9, Nmod))

for i in np.arange(Nmod):

    finterp = interp1d(stars_sin.wave_grid_rest, stars_sin.Lnu_obs[i,:])
    # f1 = finterp(1)
    f1 = 1

    axs[0].plot(stars_sin.wave_grid_rest,
                stars_sin.nu_grid_obs * stars_sin.Lnu_obs[i,:] / f1,
                color=colors_sin[i])

    finterp = interp1d(stars_noneb.wave_grid_rest, stars_noneb.Lnu_obs[i,:])
    # f1 = finterp(1)
    f1 = 1

    axs[1].plot(stars_noneb.wave_grid_rest,
                stars_noneb.nu_grid_obs * stars_noneb.Lnu_obs[i,:] / f1,
                color=colors_noneb[i])

    finterp = interp1d(stars_neb.wave_grid_rest, stars_neb.Lnu_obs[i,2,:])
    # f1 = finterp(1)
    f1 = 1

    axs[2].plot(stars_neb.wave_grid_rest,
                stars_neb.nu_grid_obs * stars_neb.Lnu_obs[i,2,:] / f1,
                color=colors_neb[i])

axs[0].set_xscale('log')
axs[0].set_yscale('log')
axs[0].set_ylim(1e4, 1e10)
axs[0].axvline(0.0912, color='dimgray', linestyle='--', label='Lyman limit')
axs[0].set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
axs[0].set_ylabel(r'$\nu L_\nu\ [\rm L_{\odot}]$')
axs[0].set_title('Without Binaries')

axs[1].set_xscale('log')
```

(continues on next page)

(continued from previous page)

```
axs[1].set_yscale('log')
axs[1].set_ylim(1e4, 1e10)
axs[1].axvline(0.0912, color='dimgray', linestyle='--', label='Lyman limit')
# axs[1].legend(loc='lower right')

axs[1].set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
axs[1].set_title('With Binaries, No Nebula')

axs[2].set_xscale('log')
axs[2].set_yscale('log')
axs[2].set_ylim(1e4, 1e10)
axs[2].axvline(0.0912, color='dimgray', linestyle='--', label='Lyman limit')
axs[2].legend(loc='lower right')

axs[2].set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
axs[2].set_title(r'With Binaries & Nebula ($\log U=-2$)')
```

[14]:
```
Text(0.5, 1.0, 'With Binaries & Nebula ($\\log U=-2$)')
```



Zoom in on the differences between the mdoels with and without binary stellar evolution:

[6]:
```
fig, ax = plt.subplots(figsize=(4,4))

# cm_noneb = mpl.colormaps['Greens']
# colors = cm_noneb(np.linspace(0.2, 0.9, Nmod))

for i in np.arange(Nmod):

    # finterp = interp1d(stars_sin.wave_grid_rest, stars_sin.Lnu_obs[i,:])
    # f1 = finterp(1)

    ax.plot(stars_sin.wave_grid_rest,
            stars_noneb.Lnu_obs[i,:] / stars_sin.Lnu_obs[i,:])

ax.set_xscale('log')
ax.set_yscale('log')
ax.set_ylim(0.1, 10)
```

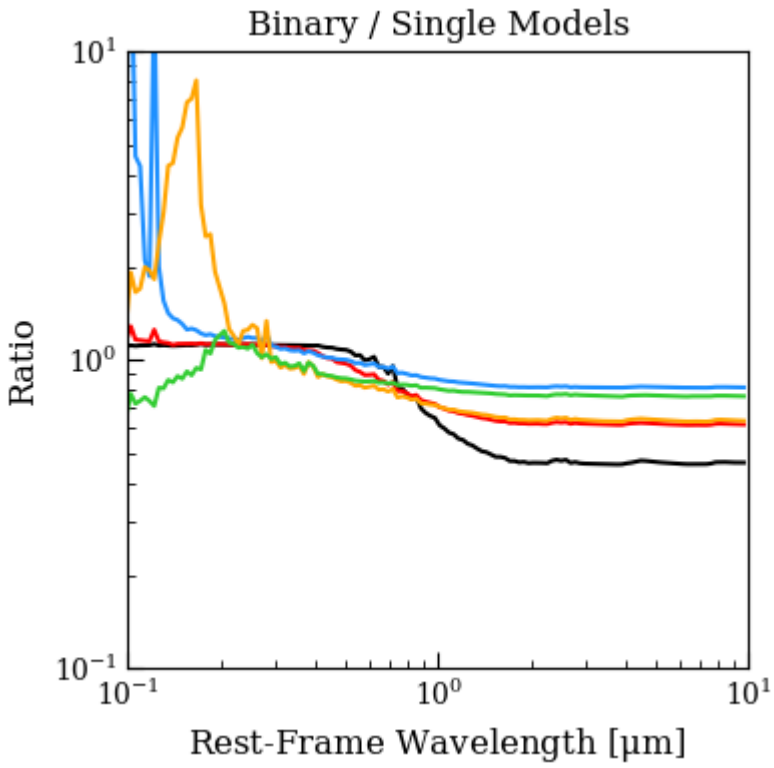
(continues on next page)

(continued from previous page)

```
ax.set_xlim(0.1, 10)

ax.set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
ax.set_ylabel(r'Ratio')
ax.set_title('Binary / Single Models')
```

```
/var/folders/p6/qlk1ytxd09vf64nrq0vfw06wgg_lcz/T/ipykernel_26571/2369984319.py:12:␣
↪RuntimeWarning: invalid value encountered in divide
  stars_noneb.Lnu_obs[i,:] / stars_sin.Lnu_obs[i,:])
```

[6]:
```
Text(0.5, 1.0, 'Binary / Single Models')
```



5.4.2 Compare Pegase Single Star Models to BPASS

Without any modification by the nebulae.

[7]:
```
bpass_sin = BPASSModel(filter_labels,
                       age=age,
                       redshift=redshift,
                       wave_grid=wave_grid,
                       binaries=False,
                       nebular_effects=False)

bpass_bin = BPASSModel(filter_labels,
                       age=age,
                       redshift=redshift,
```

(continues on next page)

(continued from previous page)

```
                        wave_grid=wave_grid,
                        binaries=True,
                        nebular_effects=False)

pegase_noneb = PEGASEModel(filter_labels,
                           age=age,
                           redshift=redshift,
                           wave_grid=wave_grid,
                           nebular_effects=False)
```

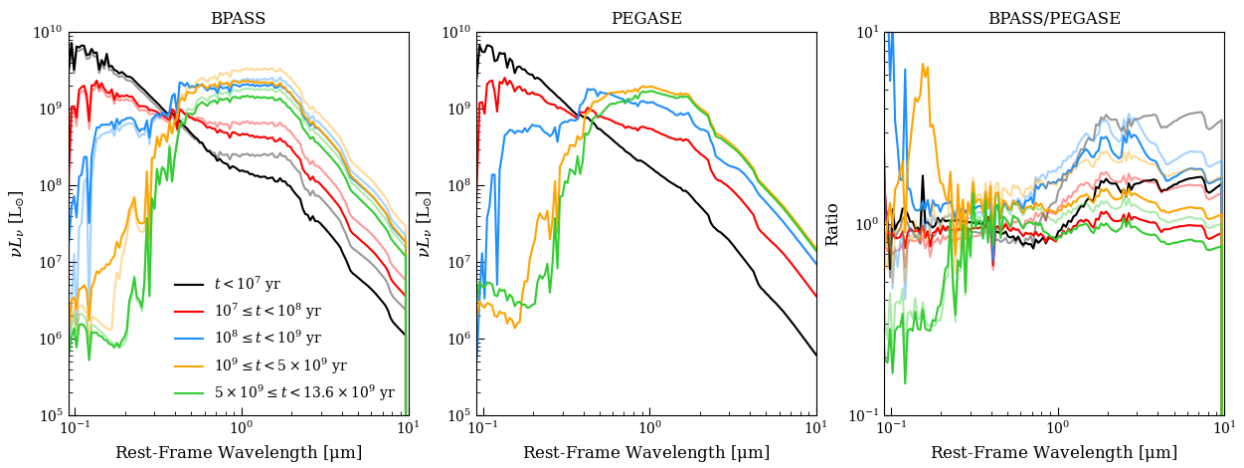
[13]:
```
fig, axs = plt.subplots(1,3,figsize=(15,5))

Nmod = len(age) - 1

labels = [r'$t < 10^7$ yr',
          r'$10^7 \leq t < 10^8$ yr',
          r'$10^8 \leq t < 10^9$ yr',
          r'$10^9 \leq t < 5 \times 10^9$ yr',
         r'$5 \times 10^9 \leq t < 13.6 \times 10^9$ yr']

for i in np.arange(Nmod):

    l,=axs[0].plot(bpass_sin.wave_grid_rest,
                   bpass_sin.nu_grid_obs * bpass_sin.Lnu_obs[i,:], alpha=0.4)
    axs[0].plot(bpass_bin.wave_grid_rest,
                   bpass_bin.nu_grid_obs * bpass_bin.Lnu_obs[i,:], color=l.get_color(),
                   label=labels[i])
    axs[1].plot(pegase_noneb.wave_grid_rest,
                  pegase_noneb.nu_grid_obs * pegase_noneb.Lnu_obs[i,:])
    l,=axs[2].plot(pegase_noneb.wave_grid_rest,
                  bpass_sin.Lnu_obs[i,:] / pegase_noneb.Lnu_obs[i,:], alpha=0.4)
    axs[2].plot(pegase_noneb.wave_grid_rest,
                  bpass_bin.Lnu_obs[i,:] / pegase_noneb.Lnu_obs[i,:], color=l.get_color())


axs[0].set_xscale('log')
axs[0].set_xlim(0.0912, 10)
axs[0].set_ylim(1e5, 1e10)
axs[0].set_yscale('log')
# axs[0].set_ylim(1e9, 5e17)
axs[0].legend(loc='lower right')

axs[0].set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
axs[0].set_ylabel(r'$\nu L_\nu\ [\rm L_{\odot}]$')
axs[0].set_title('BPASS')

axs[1].set_xscale('log')
axs[1].set_xlim(0.0912, 10)
axs[1].set_ylim(1e5, 1e10)
axs[1].set_yscale('log')
# axs[1].set_ylim(1e9, 5e17)
# axs[1].axvline(0.0912, color='dimgray', linestyle='--', label='Lyman limit')
```

(continues on next page)

(continued from previous page)

```
# axs[1].legend(loc='lower right')

axs[1].set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
axs[1].set_ylabel(r'$\nu L_\nu\ [\rm L_{\odot}]$')
axs[1].set_title('PEGASE')

axs[2].set_xscale('log')
axs[2].set_xlim(0.0912, 10)
# axs[2].set_ylim(1e5, 1e10)
axs[2].set_yscale('log')
# axs[2].set_ylim(1e9, 5e17)
# axs[2].axvline(0.0912, color='dimgray', linestyle='--', label='Lyman limit')
# axs[2].legend(loc='lower right')

axs[2].set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
axs[2].set_ylabel('Ratio')
axs[2].set_title(r'BPASS/PEGASE')
axs[2].set_ylim(0.1,10)
```

[13]:
```
(0.1, 10)
```



In the first and third panels, desaturated colors are single stars. The red excess we can see is driven by evolved massive stars; it's moderated by the binary evolution prescriptions (e.g. stars are stripped by their companion before they join the AGB) such that we see it primarily in the 100 Myr - 1 Gyr bin, where the AGB stars are really taking over.

[ ]:

5.5 AGN Emission

5.5.1 Imports

[2]:
```
import numpy as np
from lightning.agn import AGNModel
import matplotlib.pyplot as plt
import matplotlib as mpl
plt.style.use('ebm-dejavu')
%matplotlib inline
```

5.5.2 Initialize Model

[7]:
```
wave_grid = np.logspace(np.log10(0.0912),
                        np.log10(100),
                        200)
filter_labels = ['SDSS_u', 'SDSS_g', 'SDSS_r', 'SDSS_i', 'SDSS_z',
                 'MOIRCS_J', 'MOIRCS_H', 'MOIRCS_Ks',
                 'IRAC_CH1', 'IRAC_CH2', 'IRAC_CH3', 'IRAC_CH4',
                 'MIPS_CH1']

redshift = 0.0

agn = AGNModel(filter_labels,
               redshift=redshift,
               wave_grid=wave_grid,
               polar_dust=True)
```

[7]:
```
agn.print_params(verbose=True)
```

```
============================
SKIRTOR-AGN
============================
        Parameter Lo Hi                                        Description
----------------- -- -- ---------------------------------------------------
SKIRTOR_log_L_AGN  6 15    Integrated luminosity of the model in log Lsun
 SKIRTOR_cosi_AGN  0  1    Cosine of the inclination to the line of sight
   SKIRTOR_tau_97  3 11 Edge-on optical depth of the torus at 9.7 microns
  polar_dust_tauV  0  3 V-band optical depth of the polar dust extinction

Total parameters: 4
```

5.5.3 Plots with Varying Model Parameters

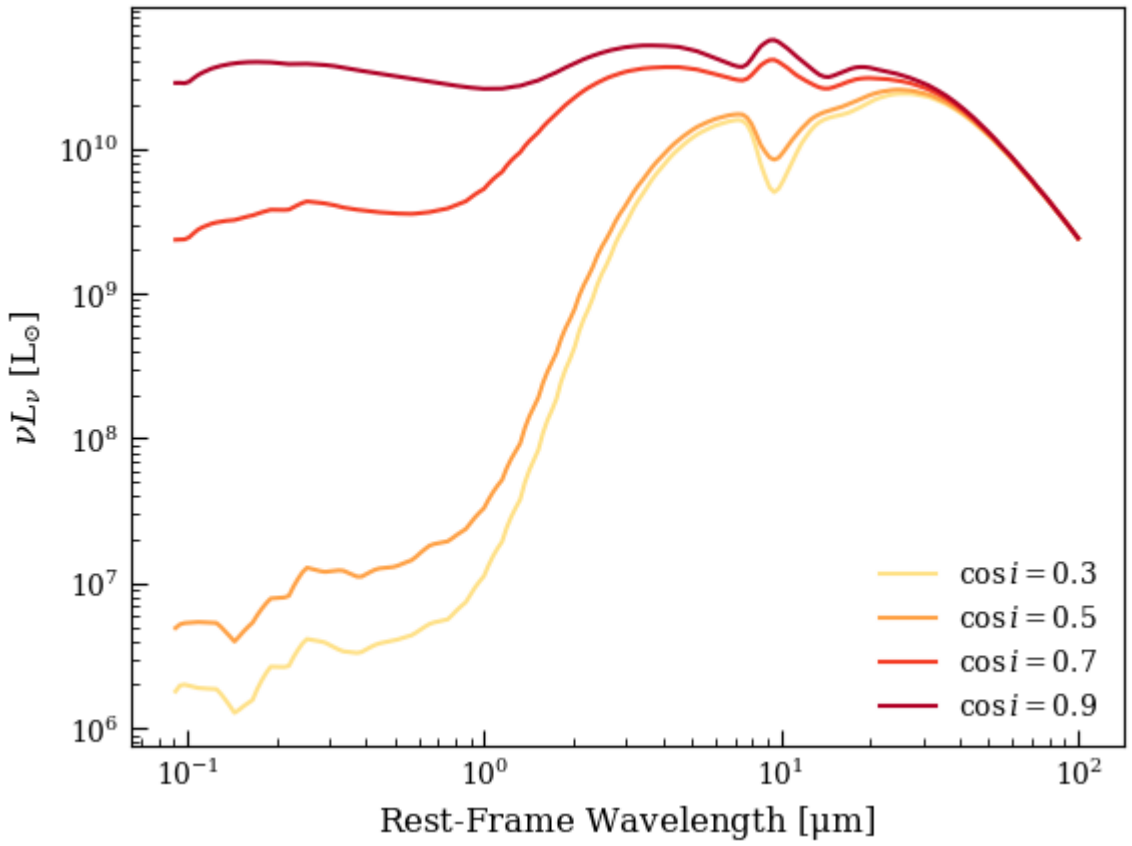
[4]:
```
def multi_model_plot(params, cmap='YlOrRd', labels=None, ax=None):

    Nmod = params.shape[0]
    if labels is None: labels = [None for i in np.arange(Nmod)]

    lnu = agn.get_model_lnu_hires(params)

    if ax is None:
        fig, ax = plt.subplots()
    else:
        fig = ax.figure

    cm = mpl.colormaps[cmap]
    colors = cm(np.linspace(0.2, 0.9, Nmod))

    for i in np.arange(Nmod):

        ax.plot(agn.wave_grid_rest,
                agn.nu_grid_obs * lnu[i,:],
                color=colors[i],
                label=labels[i])

    ax.set_xscale('log')
    ax.set_yscale('log')

    ax.set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
    ax.set_ylabel(r'$\nu L_{\nu}\ [\rm L_{\odot}]$')

    if labels is not None: ax.legend(loc = 'best')

    return fig, ax
```

Varying Inclination

The effects of the changing inclination are most pronounced at the sort of edge case, where $i \approx 90 - \Delta$, where Δ is the angle that the dusty torus cone extends upward from the plane of the accretion disk. It will often suffice to fit with inclination fixed or bounded to a small range (in fact this is the recommended strategy for e.g. Cigale) and compare the 'Type 1' and 'Type 2' fits.

[8]:
```
params = np.array([[11, 0.3, 7, 0.1],
                   [11, 0.5, 7, 0.1],
                   [11, 0.7, 7, 0.1],
                   [11, 0.9, 7, 0.1]])

multi_model_plot(params,
                 labels=[r'$\cos i = %.1f$' % p[1] for p in params])
```

[8]:
```
(<Figure size 640x480 with 1 Axes>,
 <Axes: xlabel='Rest-Frame Wavelength [$\\rm \\mu m$]', ylabel='$\\nu L_{\\nu}\\ [\\rm L_
↪{\\odot}]$'>)
```



Varying $\tau_{9.7}$ at Fixed Inclination

Unless we have *really* good data across the MIR it seems unlikely that we could constrain both the inclination and optical depth.

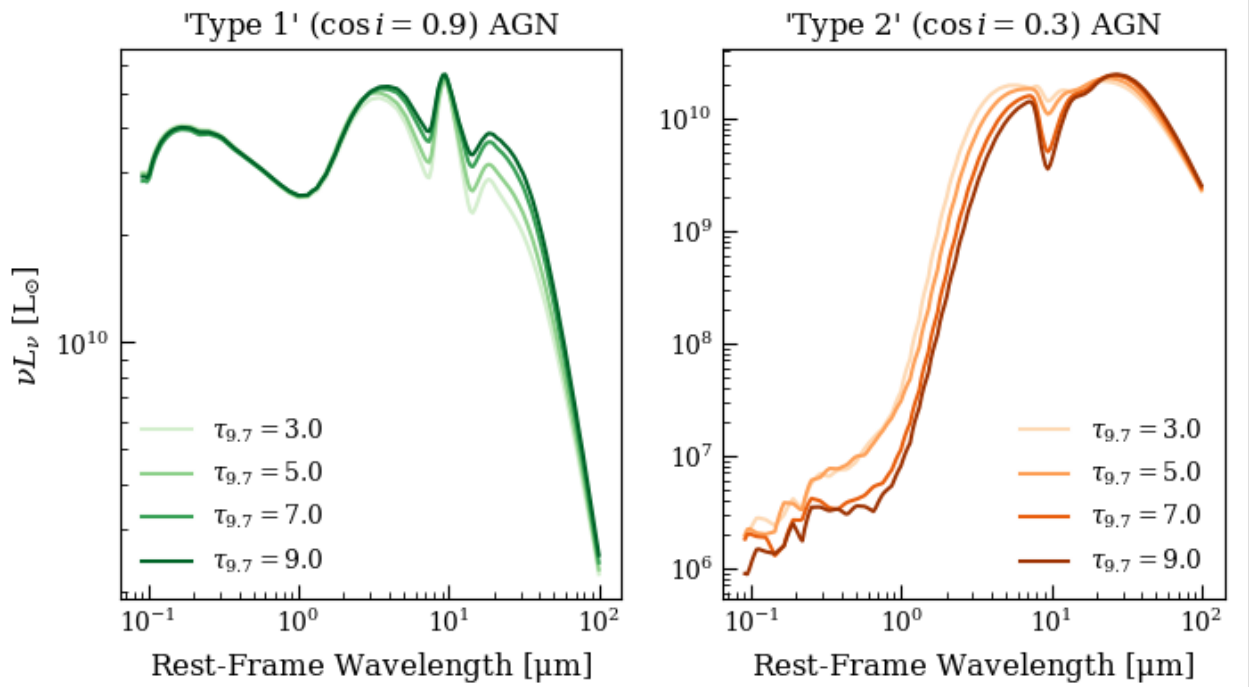
[9]:
```
params1 = np.array([[11, 0.9, 3, 0.1],
                    [11, 0.9, 5, 0.1],
                    [11, 0.9, 7, 0.1],
                    [11, 0.9, 9, 0.1]])

params2 = np.array([[11, 0.3, 3, 0.1],
                    [11, 0.3, 5, 0.1],
                    [11, 0.3, 7, 0.1],
                    [11, 0.3, 9, 0.1]])

fig, axs = plt.subplots(1,2, figsize=(8,4))

multi_model_plot(params1,
                 cmap='Greens',
                 labels=[r'$\tau_{9.7} = %.1f$' % p[2] for p in params1],
                 ax=axs[0]
                 )
```

(continues on next page)

(continued from previous page)

```
axs[0].set_title(r"'Type 1' ($\cos i = 0.9$) AGN")

multi_model_plot(params2,
                 cmap='Oranges',
                 labels=[r'$\tau_{9.7} = %.1f$' % p[2] for p in params2],
                 ax=axs[1]
                 )
axs[1].set_ylabel('')
axs[1].set_title(r"'Type 2' ($\cos i = 0.3$) AGN")
```

[9]:
```
Text(0.5, 1.0, "'Type 2' ($\\cos i = 0.3$) AGN")
```



Varying $\tau_V^{\rm pol}$ at Fixed Inclination

An optically-thick polar-dust obscurer can attenuate nearly all of the optical light from the face-on, 'Type 1' view. In the side-on, 'Type 2' view, we see none of that extra attenuation, but we do still see the isotropic, graybody dust re-emission of that attenuated optical light.

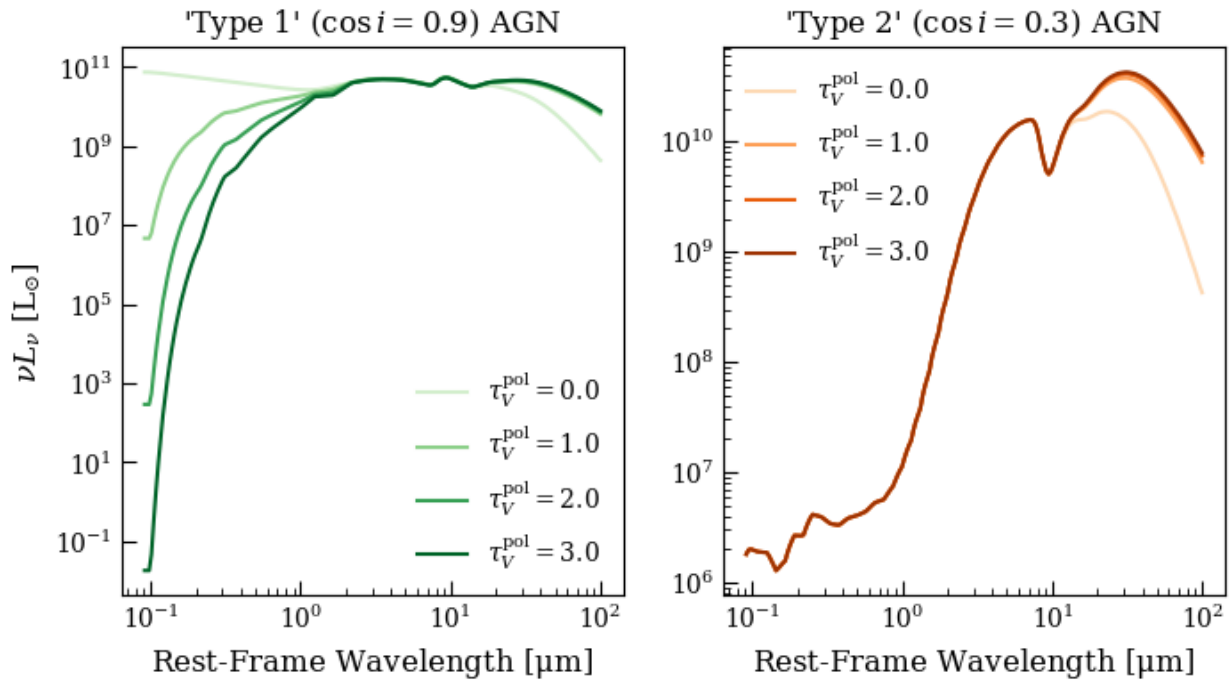
[10]:
```
params1 = np.array([[11, 0.9, 7, 0.0],
                    [11, 0.9, 7, 1],
                    [11, 0.9, 7, 2],
                    [11, 0.9, 7, 3]])

params2 = np.array([[11, 0.3, 7, 0.0],
                    [11, 0.3, 7, 1],
                    [11, 0.3, 7, 2],
                    [11, 0.3, 7, 3]])

fig, axs = plt.subplots(1,2, figsize=(8,4))
```

(continues on next page)

(continued from previous page)

```
multi_model_plot(params1,
                 cmap='Greens',
                 labels=[r'$\tau^{\rm pol}_{V} = %.1f$' % p[3] for p in params1],
                 ax=axs[0]
                 )

axs[0].set_title(r"'Type 1' ($\cos i = 0.9$) AGN")

multi_model_plot(params2,
                 cmap='Oranges',
                 labels=[r'$\tau^{\rm pol}_{V} = %.1f$' % p[3] for p in params2],
                 ax=axs[1]
                 )
axs[1].set_ylabel('')
axs[1].set_title(r"'Type 2' ($\cos i = 0.3$) AGN")
```

[10]:
```
Text(0.5, 1.0, "'Type 2' ($\\cos i = 0.3$) AGN")
```



In principle the "polar dust extinction" does not need to make many assumptions about the physical location of the obscuring dust. With a few changes to the shape of the attenuation curve and the temperature of the re-emission we could use this component to flexibly model "extra" ISM attenuation of a Type 1 AGN. Options for doing this right now are limited but would not be super difficult to expand.

5.6 Dust Attenuation and Emission

A fundamental assumption of our normal galaxy (i.e., non AGN) modeling in `Lightning` is energy balance: the optical power from starlight attenuated by ISM dust is re-radiated in the infrared. As such we present the dust attenuation and emission models side by side here.

5.6.1 Imports

[1]:
```
import numpy as np
from scipy.integrate import trapz

from lightning import Lightning
from lightning.attenuation import CalzettiAtten, ModifiedCalzettiAtten, SMC
from lightning.dust import Graybody, DL07Dust

import matplotlib.pyplot as plt
import matplotlib as mpl
plt.style.use('ebm-dejavu')
%matplotlib inline
```

5.6.2 Attenuation

`plightning` currently contains three attenuation options: Calzetti+(2000), "Modified Calzetti" (Noll+2009, including variable UV slope and a 2175 Å bump), and the SMC attenuation curve (Gordon+2003). Adding tabulated attenuation curves is pretty trivial (see `lightning.dust.TabulatedAtten`; the SMC curve is an instance of this class). The Doore+(2021) inclination-dependent attenuation curve is coming soon, pending some more testing of computational enhancements I implemented that might not actually be improvements.

[6]:
```
wave = np.logspace(np.log10(0.0912),
                   np.log10(5),
                   200)

calz = CalzettiAtten(wave)
modcalz = ModifiedCalzettiAtten(wave)
smc = SMC(wave)

calz.print_params(verbose=True)
modcalz.print_params(verbose=True)
smc.print_params(verbose=True)
```

```
============================
Calzetti
============================
     Parameter  Lo  Hi                       Description
-------------- --- --- -------------------------------
calz_tauV_diff 0.0 inf Optical depth of the diffuse ISM

Total parameters: 1

============================
```

(continues on next page)

(continued from previous page)

```
Modified-Calzetti
============================
      Parameter   Lo  Hi                                                    Description
--------------- ---- --- ------------------------------------------------------------
mcalz_tauV_diff  0.0 inf                          Optical depth of the diffuse ISM
    mcalz_delta -inf inf      Deviation from the Calzetti+2000 UV power law slope
  mcalz_tauV_BC  0.0 inf Optical depth of the birth cloud in star forming regions

Total parameters: 3

============================
smc
============================
    Parameter  Lo       Hi                       Description
------------ --- -------- --------------------------------
smc_tauV_diff 0.0 100000.0 Optical depth of the diffuse ISM

Total parameters: 1
```

5.6.3 SMC and Calzetti

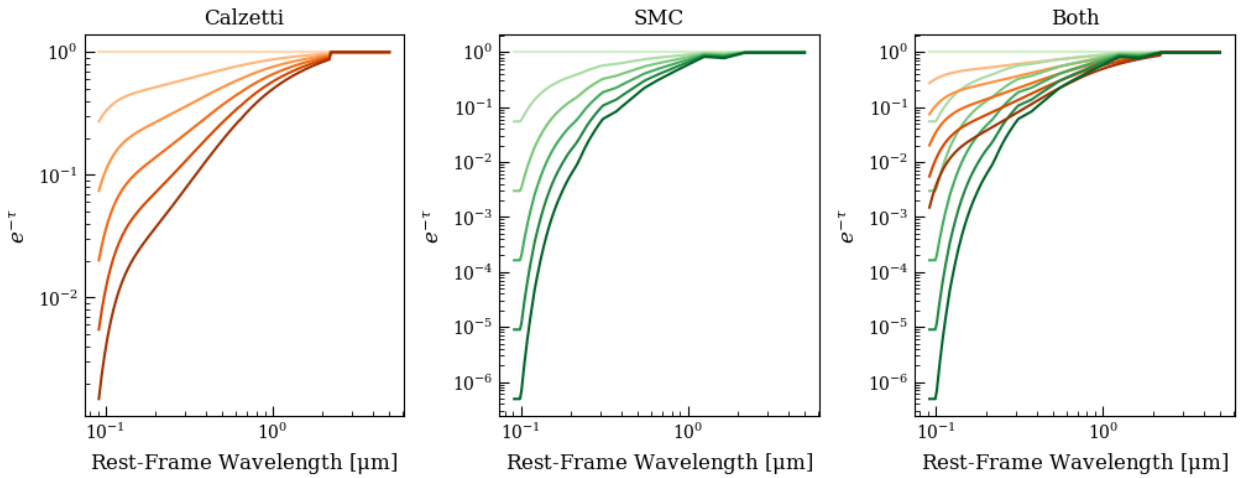
[12]:
```
fig, axs = plt.subplots(1,3, figsize=(12,4))
plt.subplots_adjust(wspace=0.3)

tauV_arr = np.linspace(0, 1.5, 6).reshape(-1, 1)
Nmod = len(tauV_arr)

cm_calz = mpl.colormaps['Oranges']
colors_calz = cm_calz(np.linspace(0.2, 0.9, Nmod))

cm_smc = mpl.colormaps['Greens']
colors_smc = cm_smc(np.linspace(0.2, 0.9, Nmod))

calz_models = calz.evaluate(tauV_arr)
smc_models = smc.evaluate(tauV_arr)

for i in np.arange(Nmod):
    axs[0].plot(wave, calz_models[i,:], color=colors_calz[i])
    axs[1].plot(wave, smc_models[i,:], color=colors_smc[i])
    axs[2].plot(wave, calz_models[i,:], color=colors_calz[i])
    axs[2].plot(wave, smc_models[i,:], color=colors_smc[i])

axs[0].set_xscale('log')
axs[0].set_yscale('log')
#axs[0].set_ylim(2e6, 1e11)

axs[0].set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
axs[0].set_ylabel(r'$e^{- \tau}$')
axs[0].set_title('Calzetti')
```

(continues on next page)

(continued from previous page)

```
axs[1].set_xscale('log')
axs[1].set_yscale('log')

axs[1].set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
axs[1].set_ylabel(r'$e^{- \tau}$')
axs[1].set_title('SMC')

axs[2].set_xscale('log')
axs[2].set_yscale('log')

axs[2].set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
axs[2].set_ylabel(r'$e^{- \tau}$')
axs[2].set_title('Both')
```

[12]:
```
Text(0.5, 1.0, 'Both')
```



The SMC curve is steeper at UV wavelengths for the same τ_V. It's used as the default curve for the AGN polar dust model.

5.6.4 Modified Calzetti

The main parameters here are the $V-$band optical depth (as above) and δ the deviation of the UV slope from the Calzetti value. The model also ostensibly allows for extra attenuation for young stars that have not yet blown away their birth cloud ($\tau_{V,\mathrm{BC}}$) but this is not currently implemented, so that parameter doesn't do anything. Working on coming up with a solution I like for flexibly allowing differential attenuation based on stellar age (this is also an issue that affects the Doore+2021 model).

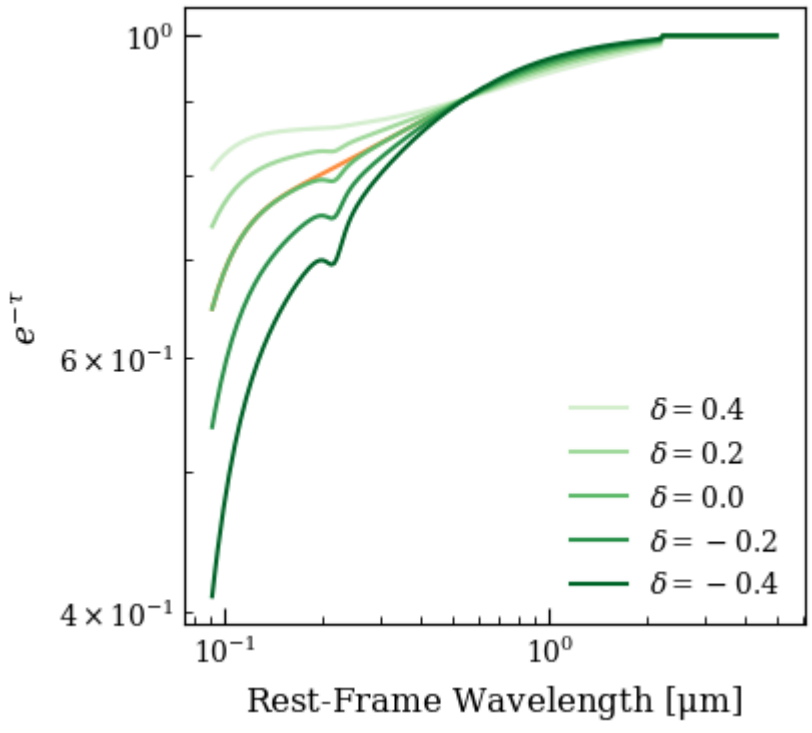
[41]:
```
fig, axs = plt.subplots(figsize=(4,4))
plt.subplots_adjust(wspace=0.3)

param_arr = np.array([[0.1, 0.4, 0.0],
                      [0.1, 0.2, 0.0],
                      [0.1, 0.0, 0.0],
                      [0.1, -0.2, 0.0],
                      [0.1, -0.4, 0.0]])
```

(continues on next page)

(continued from previous page)

```
Nmod = param_arr.shape[0]

cm_calz = mpl.colormaps['Oranges']
color_calz = cm_calz(0.5)

cm_modcalz = mpl.colormaps['Greens']
colors_modcalz = cm_smc(np.linspace(0.2, 0.9, Nmod))

calz_models = calz.evaluate(param_arr[:,0])
modcalz_models = modcalz.evaluate(param_arr)

axs.plot(wave, calz_models[0,:], color=color_calz)

for i in np.arange(Nmod):
    axs.plot(wave, modcalz_models[i,:], color=colors_modcalz[i], label=r'$\delta = %.1f$
↪' % (param_arr[i,1]))

axs.set_xscale('log')
axs.set_yscale('log')

axs.set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
axs.set_ylabel(r'$e^{- \tau}$')
axs.legend(loc='lower right')
```

[41]:
```
<matplotlib.legend.Legend at 0x17e960670>
```



The modified Calzetti model is in green, where lighter → darker colors indicate δ changing from positive to negative. The strength of the 2175 Å feature is tied to the UV slope parameter, δ. Note that the modified Calzetti model is not exactly equal to the Calzetti model with the same τ_V when $\delta = 0$, due to the presence of the 2175 Å absorption feature.

5.6.5 Emission Models

[2]:
```
wave_grid = np.logspace(np.log10(1),
                        np.log10(1000),
                        200)
filter_labels = ['IRAC_CH1', 'IRAC_CH2', 'IRAC_CH3', 'IRAC_CH4',
                 'MIPS_CH1',
                 'PACS_green', 'PACS_red',
                 'SPIRE_250', 'SPIRE_350', 'SPIRE_500']

redshift = 0.0

dl07 = DL07Dust(filter_labels,
                redshift=redshift,
                wave_grid=wave_grid)
gb = Graybody(filter_labels,
              redshift=redshift,
              wave_grid=wave_grid)

dl07.print_params(verbose=True)
gb.print_params(verbose=True)
```

```
============================
DL07-Dust
============================
      Parameter     Lo       Hi                                                  ␣
↪   Description
--------------- ------ -------- ------------------------------------------------------------
↪--------------
dl07_dust_alpha  -10.0      4.0                   Radiation field intensity distribution␣
↪power law index
dl07_dust_U_min    0.1     25.0                                        Radiation field␣
↪minimum intensity
dl07_dust_U_max 1000.0 300000.0                                        Radiation field␣
↪maximum intensity
dl07_dust_gamma    0.0      1.0 Fraction of dust mass exposed to radiation field␣
↪intensity distribution
dl07_dust_q_PAH 0.0047   0.0458                             Fraction of dust mass␣
↪composed of PAH grains

Total parameters: 5

============================
GrayBody-Dust
============================
Parameter   Lo     Hi                      Description
--------- ---- ------ --------------------------------
  gb_lam0 10.0 1000.0 Wavelength of unit optical depth
  gb_beta  0.1    5.0    Optical depth power law index
     gb_T 10.0  100.0                 Dust temperature

Total parameters: 3
```

Draine and Li (2017)

The Draine and Li model carries around its own wavelength grid due to some artifact of how the models were constructed in IDL `Lightning`. I should aee an option to interpolate it to some other grid for consistency with every other model class…

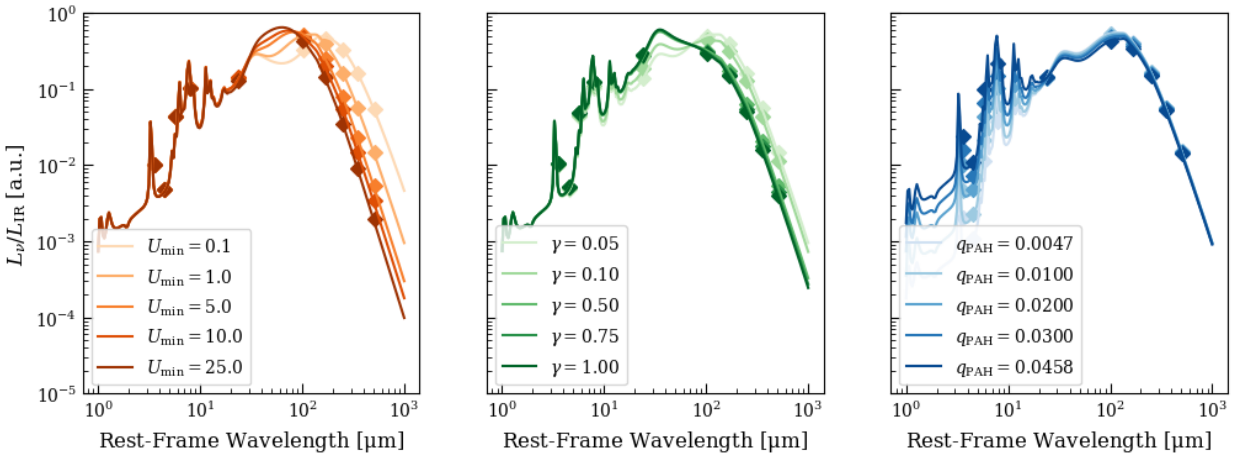
[4]:
```
fig, axs = plt.subplots(1,3, figsize=(12,4))

var_umin = np.array([[2, 0.1, 3e5, 0.05, 0.02],
                     [2, 1, 3e5, 0.05, 0.02],
                     [2, 5, 3e5, 0.05, 0.02],
                     [2, 10, 3e5, 0.05, 0.02],
                     [2, 25, 3e5, 0.05, 0.02]])

var_gamma = np.array([[2, 1, 3e5, 0.05, 0.02],
                     [2, 1, 3e5, 0.1, 0.02],
                     [2, 1, 3e5, 0.5, 0.02],
                     [2, 1, 3e5, 0.75, 0.02],
                     [2, 1, 3e5, 1.0, 0.02]])

var_qpah = np.array([[2, 1, 3e5, 0.05, 0.0047],
                     [2, 1, 3e5, 0.05, 0.01],
                     [2, 1, 3e5, 0.05, 0.02],
                     [2, 1, 3e5, 0.05, 0.03],
                     [2, 1, 3e5, 0.05, 0.0458]])

mod_var_umin, L_var_umin = dl07.get_model_lnu_hires(var_umin)
Lmod_var_umin, L_var_umin = dl07.get_model_lnu(var_umin)
mod_var_gamma, L_var_gamma = dl07.get_model_lnu_hires(var_gamma)
Lmod_var_gamma, L_var_gamma = dl07.get_model_lnu(var_gamma)
mod_var_qpah, L_var_qpah = dl07.get_model_lnu_hires(var_qpah)
Lmod_var_qpah, L_var_qpah = dl07.get_model_lnu(var_qpah)

Nmod = var_umin.shape[0]

cm_umin = mpl.colormaps['Oranges']
colors_umin = cm_umin(np.linspace(0.2, 0.9, Nmod))

cm_gamma = mpl.colormaps['Greens']
colors_gamma = cm_gamma(np.linspace(0.2, 0.9, Nmod))

cm_qpah = mpl.colormaps['Blues']
colors_qpah = cm_qpah(np.linspace(0.2, 0.9, Nmod))

for i in np.arange(Nmod):

    axs[0].plot(dl07.wave_grid_rest, dl07.nu_grid_rest * mod_var_umin[i,:] / L_var_
↪umin[i], color=colors_umin[i], label=r'$U_{\rm min} = %.1f$' % (var_umin[i,1]))
    axs[0].scatter(dl07.wave_obs, dl07.nu_obs * Lmod_var_umin[i,:] / L_var_umin[i],␣
↪color=colors_umin[i], marker='D')
    axs[1].plot(dl07.wave_grid_rest, dl07.nu_grid_rest * mod_var_gamma[i,:] / L_var_
↪gamma[i], color=colors_gamma[i], label=r'$\gamma = %.2f$' % (var_gamma[i,3]))
    axs[1].scatter(dl07.wave_obs, dl07.nu_obs * Lmod_var_gamma[i,:] / L_var_gamma[i],␣
```

(continues on next page)

(continued from previous page)

```
↪color=colors_gamma[i], marker='D')
    axs[2].plot(dl07.wave_grid_rest, dl07.nu_grid_rest * mod_var_qpah[i,:] / L_var_
↪qpah[i], color=colors_qpah[i], label=r'$q_{\rm PAH} = %.4f$' % (var_qpah[i,4]))
    axs[2].scatter(dl07.wave_obs, dl07.nu_obs * Lmod_var_qpah[i,:] / L_var_qpah[i],␣
↪color=colors_qpah[i], marker='D')

axs[0].set_xscale('log')
axs[0].set_yscale('log')
axs[0].set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
axs[0].set_ylabel(r'$L_\nu / L_{\rm IR}$ [a.u.]')
axs[0].set_ylim(1e-5, 1)
axs[0].legend(loc='lower left', frameon=True)
#axs[0].set_title(r'$U_{\rm min}=0.1\rightarrow25$')

axs[1].set_xscale('log')
axs[1].set_yscale('log')
axs[1].set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
axs[1].set_yticklabels([])
axs[1].set_ylim(1e-5, 1)
axs[1].legend(loc='lower left', frameon=True)
#axs[1].set_title(r'$\gamma=0.05\rightarrow1$')

axs[2].set_xscale('log')
axs[2].set_yscale('log')
axs[2].set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
axs[2].set_yticklabels([])
axs[2].set_ylim(1e-5, 1)
axs[2].legend(loc='lower left', frameon=True)
#axs[2].set_title(r'$q_{\rm PAH}=0.0047\rightarrow0.0458$')
```

[4]:
```
<matplotlib.legend.Legend at 0x17a179870>
```



In each of the above plots the changing parameter goes from lighter → darker lines. Note that in practice the normalization of the model is set by the attenuated power from starlight.

5.6.6 Graybody

This model is currently used in our modeling only to represent the re-emission component of the AGN polar dust model.

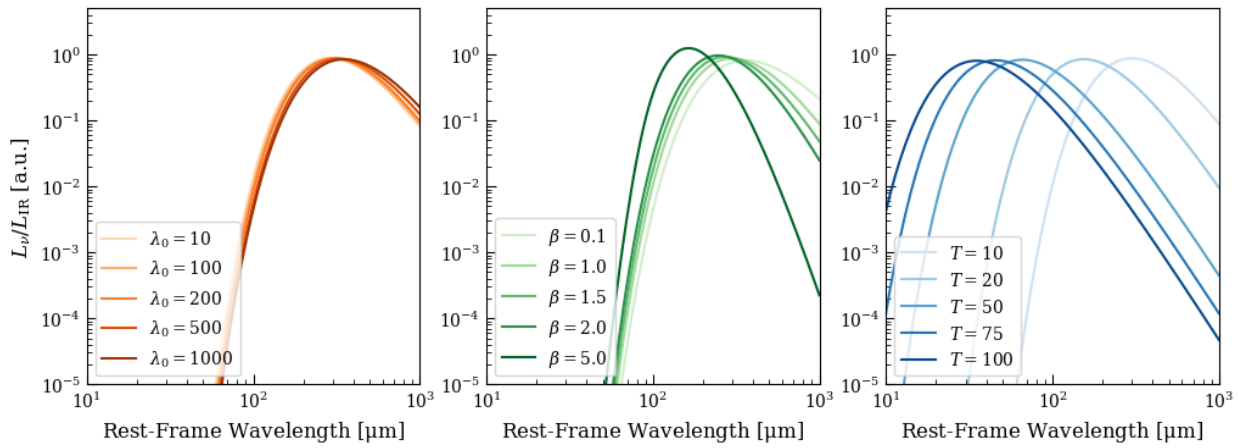
[50]:
```
fig, axs = plt.subplots(1,3, figsize=(12,4))

var_lam0 = np.array([[10, 1, 10],
                     [100, 1, 10],
                     [200, 1, 10],
                     [500, 1, 10],
                     [1000, 1, 10]])

var_beta = np.array([[100, 0.1, 10],
                     [100, 1, 10],
                     [100, 1.5, 10],
                     [100, 2, 10],
                     [100, 5, 10]])

var_temp = np.array([[100, 1, 10],
                     [100, 1, 20],
                     [100, 1, 50],
                     [100, 1, 75],
                     [100, 1, 100]])

mod_var_lam0 = gb.get_model_lnu_hires(var_lam0)
mod_var_beta = gb.get_model_lnu_hires(var_beta)
mod_var_temp = gb.get_model_lnu_hires(var_temp)

Nmod = var_umin.shape[0]

cm_lam0 = mpl.colormaps['Oranges']
colors_lam0 = cm_lam0(np.linspace(0.2, 0.9, Nmod))

cm_beta = mpl.colormaps['Greens']
colors_beta = cm_beta(np.linspace(0.2, 0.9, Nmod))

cm_temp = mpl.colormaps['Blues']
colors_temp = cm_temp(np.linspace(0.2, 0.9, Nmod))

for i in np.arange(Nmod):

    axs[0].plot(gb.wave_grid_rest, gb.nu_grid_rest * mod_var_lam0[i,:], color=colors_
↪lam0[i], label=r'$\lambda_0 = %d$' % (var_lam0[i,0]))
    axs[1].plot(gb.wave_grid_rest, gb.nu_grid_rest * mod_var_beta[i,:], color=colors_
↪beta[i], label=r'$\beta = %.1f$' % (var_beta[i,1]))
    axs[2].plot(gb.wave_grid_rest, gb.nu_grid_rest * mod_var_temp[i,:], color=colors_
↪temp[i], label=r'$T = %d$' % (var_temp[i,2]))

axs[0].set_xscale('log')
axs[0].set_yscale('log')
axs[0].set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
axs[0].set_ylabel(r'$L_\nu / L_{\rm IR}$ [a.u.]')
axs[0].set_ylim(1e-5, 5)
```

(continues on next page)

(continued from previous page)

```
axs[0].set_xlim(10, 1000)
#axs[0].set_title(r'$\lambda_{0}=10\rightarrow1000~\rm \mu m$')
axs[0].legend(loc='lower left', frameon=True)

axs[1].set_xscale('log')
axs[1].set_yscale('log')
axs[1].set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
axs[1].set_ylim(1e-5, 5)
axs[1].set_xlim(10, 1000)
#axs[1].set_title(r'$\beta=0.1\rightarrow5$')
axs[1].legend(loc='lower left', frameon=True)


axs[2].set_xscale('log')
axs[2].set_yscale('log')
axs[2].set_xlabel(r'Rest-Frame Wavelength [$\rm \mu m$]')
axs[2].set_ylim(1e-5, 5)
axs[2].set_xlim(10, 1000)
#axs[2].set_title(r'$T=10\rightarrow100~\rm K$')
axs[2].legend(loc='lower left', frameon=True)
```

[50]:
```
<matplotlib.legend.Legend at 0x17fc07040>
```



Again, for energy balance the model should be normalized to the total attenuated power.

CHAPTER

SIX

NEBULAR EMISSION RECIPE

Our custom nebular emission models are single cloud models, using `Cloudy` simulations assuming:

- An open geometry
- $\log U \in [-4, -1.5]$
- Gas-phase metallicity equal to the stellar metallicity (i.e. the metallicity of the youngest stars)
- Two densities – $\log n_H = 2$ and $\log n_H = 3.5$.
- Constant pressure.

For each spectral template, Q_0 is effectively fixed, and thus varying $\log U$ is equivalent to varying R. We convert the line and continuum intensities produced by `Cloudy` back into luminosities by solving for the appropriate R given Q_0, $\log U$, and $\log n_H$.

We include the option to add dust grains inside the H II region, which we note produces the effect of extra attenuation and warm dust emission in the emitted continuum in addition to affecting the emitted line ratios.

CHAPTER

SEVEN

FILTER PROFILES

A full list of available filters in Lightning can be found [below](filter_table). All filters are in terms of the spectral response per energy (i.e., equal-energy).

7.1 Directory Structure

The `filters/` directory contains all the current filter profiles available in Lightning. The structure of the directory follows the pattern `filters/observatory/instrument/instrument_band.txt`. In the case of survey specific filters (e.g., 2MASS, SDSS, etc.), the pattern is changed to `filters/survey/survey_band.txt`. Examples for HST WFC3/F125W and 2MASS J band would be `filters/HST/WFC3/WFC3_F125W.txt` and `filters/2MASS/2MASS_J.txt`, respectively.

7.2 File Format

Each filter profile file in Lightning has been edited to a common format. The original source files can be found in the linked URLs in the table [below](filter_table). Each formatted filter profile file has two, space delimited columns giving the wavelength (in microns) and throughput function (normalized to the maximum value). The values in both columns are formatted using the FORTRAN (C) formatting code `E13.7` (`%13.7e`).

For example, the first several lines for the 2MASS J band are:

```
# wave[microns]    norm_trans
  1.0620000E+00    0.0000000E+00
  1.0660000E+00    4.0706800E-04
  1.0700000E+00    1.5429300E-03
  1.0750000E+00    2.6701300E-03
  1.0780000E+00    5.5064300E-03
```

7.3 Addition of New Filters

New filters can be added manually, by formatting them as above and placing them in a subdirectory of the `filters/` directory. Filters must then be added to `filters/filters.json`, where the key is the 'filter_label' which will be used to specify the filter in lightning, and the value is the relative path to the filter.

> **Note**
>
> Adding new filters will make your local git repository out of sync with the remote. It is currently recommended to make a `filters/user/` directory, which you back up before updating the code and replace after updating. Note that paths to your filters need not follow the exact formula as the built-in filters, as long as the path in `filters/filters.json` is correct.

7.4 List of Filters in Lightning

The below table gives a list of all the filters currently available in Lightning, with links to the source and the corresponding filter labels.

> **Note**
>
> Filter labels in the below table may not render correctly in the PDF version of this manual. Check `filters/filters.json` for consistency.

Table 1: Filters available

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
2MASS		J, H, Ks	*2MASS_J*, *2MASS_H*, *2MASS_Ks*
Akari	FIS	N60, WIDE-S, WIDE-L, N160	*FIS_N60*, *FIS_WIDE-S*, *FIS_WIDE-L*, *FIS_N160*
Akari	IRC	N2, N3, N4, S7, S9W, S11, L15, L18W, L24	*IRC_N2*, *IRC_N3*, *IRC_N4*, *IRC_S7*, *IRC_S9W*, *IRC_S11*, *IRC_L15*, *IRC_L18W*, *IRC_L24*
ALMA		band3, band4, band5, band6, band7, band8, band9, band10	*ALMA_band3*, *ALMA_band4*, *ALMA_band5*, *ALMA_band6*, *ALMA_band7*, *ALMA_band8*, *ALMA_band9*, *ALMA_band10*
APEX	LABOCA	LABOCA	*LABOCA_LABOCA*
APEX	SABOCA	SABOCA	*SABOCA_SABOCA*
Apache Point Observatory	Broad	B, V, R, I	*Broad_B*, *Broad_V*, *Broad_R*, *Broad_I*
Apache Point Observatory	FNAL	V, R	*FNAL_V*, *FNAL_R*
Apache Point Observatory	Gunn	g, r, i, z	*Gunn_g*, *Gunn_r*, *Gunn_i*, *Gunn_z*
Apache Point Observatory	Hodge	6563, 6585, 6607, 6629	*Hodge_6563*, *Hodge_6585*, *Hodge_6607*, *Hodge_6629*

continues on next page

Table 1 – continued from previous page

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
Apache Point Observatory	York	5059, 5200, 5684, 5720, 5917, 6007, 6085, 6177, 6409, 6903, 6931, 7740, 8212, 8238, 8678, 9088	*York_5059*, *York_5200*, *York_5684*, *York_5720*, *York_5917*, *York_6007*, *York_6085*, *York_6177*, *York_6409*, *York_6903*, *York_6931*, *York_7740*, *York_8212*, *York_8238*, *York_8678*, *York_9088*
AstrosatUVIT	FUV	F148W, F148Wa, F154W, F169M, F172M	*FUV_F148W*, *FUV_F148Wa*, *FUV_F154W*, *FUV_F169M*, *FUV_F172M*
AstrosatUVIT	NUV	N219M, N242W, N245M, N263M, N279N	*NUV_N219M*, *NUV_N242W*, *NUV_N245M*, *NUV_N263M*, *NUV_N279N*
AstrosatUVIT	VIS	V347M, V391M, V420W, V435ND, V461W	*VIS_V347M*, *VIS_V391M*, *VIS_V420W*, *VIS_V435ND*, *VIS_V461W*
CFHT	CFH12k	B, Custom B, V, R, I, Zprime, HBeta off, HBeta on, OIII, HAlpha off, HAlpha on, TiO, CN, NB92	*CFH12k_B*, *CFH12k_Custom_B*, *CFH12k_V*, *CFH12k_R*, *CFH12k_I*, *CFH12k_Zprime*, *CFH12k_HBeta_off*, *CFH12k_HBeta_on*, *CFH12k_OIII*, *CFH12k_HAlpha_off*, *CFH12k_HAlpha_on*, *CFH12k_TiO*, *CFH12k_CN*, *CFH12k_NB92*
CFHT	MegaCam	U v1, U v3, G v1, G v3, R v1, R v3, I v1, I v2, I v3, Z v1, Z v3	*MegaCam_U_v1*, *MegaCam_U_v3*, *MegaCam_G_v1*, *MegaCam_G_v3*, *MegaCam_R_v1*, *MegaCam_R_v3*, *MegaCam_I_v1*, *MegaCam_I_v2*, *MegaCam_I_v3*, *MegaCam_Z_v1*, *MegaCam_Z_v3*
CFHT	MOCAM	B, V, CFHT V, R, Harris R, I	*MOCAM_B*, *MOCAM_V*, *MOCAM_CFHT_V*, *MOCAM_R*, *MOCAM_Harris_R*, *MOCAM_I*
CFHT	QUIRC	J, H, K, Kprime, Kshort, H+K, Hs, Hl, HeI, n1975, n2131, BrGamma, n2265, CO	*QUIRC_J*, *QUIRC_H*, *QUIRC_K*, *QUIRC_Kprime*, *QUIRC_Kshort*, *QUIRC_H+K*, *QUIRC_Hs*, *QUIRC_Hl*, *QUIRC_HeI*, *QUIRC_n1975*, *QUIRC_n2131*, *QUIRC_BrGamma*, *QUIRC_n2265*, *QUIRC_CO*

continues on next page

Table 1 – continued from previous page

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
CFHT	WIRCam	Y, J, J v2, H, H v2, Ks, Ks v2, K Cont, Low OH 2, Low OH 1, CH4 On, CH4 Off, H2, BrGamma	*WIRCam_Y*, *WIRCam_J*, *WIRCam_J_v2*, *WIRCam_H*, *WIRCam_H_v2*, *WIRCam_Ks*, *WIRCam_Ks_v2*, *WIRCam_K_Cont*, *WIRCam_Low_OH_2*, *WIRCam_Low_OH_1*, *WIRCam_CH4_On*, *WIRCam_CH4_Off*, *WIRCam_H2*, *WIRCam_BrGamma*
COBE	DIRBE	band1, band2, band3, band4, band5, band6, band7, band8, band9, band10	*DIRBE_band1*, *DIRBE_band2*, *DIRBE_band3*, *DIRBE_band4*, *DIRBE_band5*, *DIRBE_band6*, *DIRBE_band7*, *DIRBE_band8*, *DIRBE_band9*, *DIRBE_band10*
CTIO	ANDICAM	B, B YALO, V, R, R YALO, I, J, H, K	*ANDICAM_B*, *ANDICAM_B_YALO*, *ANDICAM_V* *ANDICAM_R*, *ANDICAM_R_YALO*, *ANDICAM_I*, *ANDICAM_J*, *ANDICAM_H*, *ANDICAM_K*
CTIO	DECam	g, r, i, z, Y	*DECam_g*, *DECam_r*, *DECam_i*, *DECam_z*, *DECam_Y*
CTIO	ISPI	Y 191A, Y 191B, Y 191C, I 81, J 183, J 186, J 192, J 40, J 82a, H 44, H 184, H 187, K 50, K 185, Ks 129, FeII 111, PAlpha 110, 203 134, HeI 135, CIV 136, H2 163, H2 89, H2 130, 214 164, 214w 131, 216 165, BrGamma 132, HeII 138, H2 125	*ISPI_Y_191A*, *ISPI_Y_191B*, *ISPI_Y_191C*, *ISPI_I_81*, *ISPI_J_183*, *ISPI_J_186*, *ISPI_J_192*, *ISPI_J_40*, *ISPI_J_82a*, *ISPI_H_44*, *ISPI_H_184*, *ISPI_H_187*, *ISPI_K_50*, *ISPI_K_185*, *ISPI_Ks_129*, *ISPI_FeII_111*, *ISPI_PAlpha_110*, *ISPI_203_134*, *ISPI_HeI_135*, *ISPI_CIV_136*, *ISPI_H2_163*, *ISPI_H2_89*, *ISPI_H2_130*, *ISPI_214_164*, *ISPI_214w_131*, *ISPI_216_165*, *ISPI_BrGamma_132*, *ISPI_HeII_138*, *ISPI_H2_125*

continues on next page

Table 1 – continued from previous page

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
CTIO	MOSAICII	U, B, Bj Tyson, V, V old, VR Stubbs, R, I, I Tyson, I old, u SDSS, g SDSS, r SDSS, i SDSS, z SDSS, C Wash, M Wash, D51, OII, OIII, Halpha, Halpha+80, SII	*MOSAICII_U*, *MOSAICII_B*, *MOSAICII_Bj_Tyson*, *MOSAICII_V*, *MOSAICII_V_old*, *MOSAICII_VR_Stubbs*, *MOSAICII_R*, *MOSAICII_I*, *MOSAICII_I_Tyson*, *MOSAICII_I_old*, *MOSAICII_u_SDSS*, *MOSAICII_g_SDSS*, *MOSAICII_r_SDSS*, *MOSAICII_i_SDSS*, *MOSAICII_z_SDSS*, *MOSAICII_C_Wash*, *MOSAICII_M_Wash*, *MOSAICII_D51*, *MOSAICII_OII*, *MOSAICII_OIII*, *MOSAICII_Halpha*, *MOSAICII_Halpha+80*, *MOSAICII_SII*
CTIO	SIO	U Bessel, B Bessel, V Bessel, R Bessel, I Bessel, u sdss, g sdss, r sdss, i sdss, z sdss, u Strom, v Strom, y Strom, HAlpha, SII, TiO, CN	*SIO_U_Bessel*, *SIO_B_Bessel*, *SIO_V_Bessel*, *SIO_R_Bessel*, *SIO_I_Bessel*, *SIO_u_sdss*, *SIO_g_sdss*, *SIO_r_sdss*, *SIO_i_sdss*, *SIO_z_sdss*, *SIO_u_Strom*, *SIO_v_Strom*, *SIO_y_Strom*, *SIO_HAlpha*, *SIO_SII*, *SIO_TiO*, *SIO_CN*
CTIO	Spartan	Y, J, H, K, HeI, FeII, Cont1, HeICIV, H2, Cont2, BrGamma, Cont3, CO	*Spartan_Y*, *Spartan_J*, *Spartan_H*, *Spartan_K*, *Spartan_HeI*, *Spartan_FeII*, *Spartan_Cont1*, *Spartan_HeICIV*, *Spartan_H2*, *Spartan_Cont2*, *Spartan_BrGamma*, *Spartan_Cont3*, *Spartan_CO*
CTIO	SPLUS	u, g, r, i, z, J0378, J0395, J0410, J0430, J0515, J0660, J0861	*SPLUS_u*, *SPLUS_g*, *SPLUS_r*, *SPLUS_i*, *SPLUS_z*, *SPLUS_J0378*, *SPLUS_J0395*, *SPLUS_J0410*, *SPLUS_J0430*, *SPLUS_J0515*, *SPLUS_J0660*, *SPLUS_J0861*
CTIO	Y4KCam	u, r, i, z, B, V, Rc, Ic	*Y4KCam_u*, *Y4KCam_r*, *Y4KCam_i*, *Y4KCam_z*, *Y4KCam_B*, *Y4KCam_V*, *Y4KCam_Rc*, *Y4KCam_Ic*
DENIS		I, J, Ks	*DENIS_I*, *DENIS_J*, *DENIS_Ks*
GAIA	DR2	DR2 G, Gbp, Grp	*DR2_G*, *DR2_Gbp*, *DR2_Grp*
GAIA	EDR3	EDR3 G, Gbp, Grp	*EDR3_G*, *EDR3_Gbp*, *EDR3_Grp*
GALEX		FUV, NUV	*GALEX_FUV*, *GALEX_NUV*

continues on next page

Table 1 – continued from previous page

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
Gemini	AcqCam	U-South, B-South, V-North, V-South, R-North, R-South, I-North, I-South	*AcqCam_U_S*, *AcqCam_B_S*, *AcqCam_V_N*, *AcqCam_V_S*, *AcqCam_R_N*, *AcqCam_R_S*, *AcqCam_I_N*, *AcqCam_I_S*
Gemini	Alopeke-Zorro	u XSDSS, g XSDSS, r XSDSS, i XSDSS, z XSDSS, EO 466, EO 562, EO 716, EO 832, HAlpha	*Alopeke-Zorro_u_XSDSS*, *Alopeke-Zorro_g_XSDSS*, *Alopeke-Zorro_r_XSDSS*, *Alopeke-Zorro_i_XSDSS*, *Alopeke-Zorro_z_XSDSS*, *Alopeke-Zorro_EO_466*, *Alopeke-Zorro_EO_562*, *Alopeke-Zorro_EO_716*, *Alopeke-Zorro_EO_832*, *Alopeke-Zorro_HAlpha*
Gemini	Flamingos-2	Y, J, H, Ks, K blue, K red, Jlow, JH, HK, Klong	*Flamingos-2_Y*, *Flamingos-2_J*, *Flamingos-2_H*, *Flamingos-2_Ks*, *Flamingos-2_K_blue*, *Flamingos-2_K_red*, *Flamingos-2_Jlow*, *Flamingos-2_JH*, *Flamingos-2_HK*, *Flamingos-2_Klong*

continues on next page

Table 1 – continued from previous page

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
Gemini	GMOS	u-North, u-South, g-North, g-South, r-North, r-South, i-North, i-South, z-North, z-South, Y-South, Zcap-South, ri-North, CaT-North, CaT-South, DS920-North, GG455-North, GG455-South, HaC-North, HaC-South, Ha-North, Ha-South, HeIIC-North, HeIIC-South, HeII-North, HeII-South, OG515-North, OG515-South, OIIIC-North, OIIIC-South, OIII-North, OIII-South, OVIC-North, OVIC-South, OVI-North, OVI-South, RG610-North, RG610-South, RG780-South, SII-North, SII-South	*GMOS_u_N*, *GMOS_u_S*, *GMOS_g_N*, *GMOS_g_S*, *GMOS_r_N*, *GMOS_r_S*, *GMOS_i_N*, *GMOS_i_S*, *GMOS_z_N*, *GMOS_Z_S*, *GMOS_Y_S*, *GMOS_Zcap_S*, *GMOS_ri_N*, *GMOS_CaT_N*, *GMOS_CaT_S*, *GMOS_DS920_N*, *GMOS_GG455_N*, *GMOS_GG455_S*, *GMOS_HaC_N*, *GMOS_HaC_S*, *GMOS_Ha_N*, *GMOS_Ha_S*, *GMOS_HeIIC_N*, *GMOS_HeIIC_S*, *GMOS_HeII_N*, *GMOS_HeII_S*, *GMOS_OG515_N*, *GMOS_OG515_S*, *GMOS_OIIIC_N*, *GMOS_OIIIC_S*, *GMOS_OIII_N*, *GMOS_OIII_S*, *GMOS_OVIC_N*, *GMOS_OVIC_S*, *GMOS_OVI_N*, *GMOS_OVI_S*, *GMOS_RG610_N*, *GMOS_RG610_S*, *GMOS_RG780_S*, *GMOS_SII_N*, *GMOS_SII_S*
Gemini	GNIRS	X, J, H, K, Y, J-MK, K-MK, H2, PAH	*GNIRS_X*, *GNIRS_J*, *GNIRS_H*, *GNIRS_K*, *GNIRS_Y*, *GNIRS_J-MK*, *GNIRS_K-MK*, *GNIRS_H2*, *GNIRS_PAH*
Gemini	GSAOI	Z, J, H, Kprime, Kshort, K, Jcont, Hcont, CH4short, CH4long, Kshort cont, Klong cont, HeI, PaBeta, PaGamma, FeII, H2O, HeI_2p2s, BrGamma, H2 1-0, H2 2-1, CO	*GSAOI_Z*, *GSAOI_J*, *GSAOI_H*, *GSAOI_Kprime*, *GSAOI_Kshort*, *GSAOI_K*, *GSAOI_Jcont*, *GSAOI_Hcont*, *GSAOI_CH4short*, *GSAOI_CH4long*, *GSAOI_Kshort_cont*, *GSAOI_Klong_cont*, *GSAOI_HeI*, *GSAOI_PaBeta*, *GSAOI_PaGamma*, *GSAOI_FeII*, *GSAOI_H2O*, *GSAOI_HeI_2p2s*, *GSAOI_BrGamma*, *GSAOI_H2_10*, *GSAOI_H2_21*, *GSAOI_CO*

continues on next page

Table 1 – continued from previous page

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
Gemini	Michelle	N, Nprime, Qa, Q, Si1, Si2, Si3, Si4, Si5, Si6	*Michelle_N*, *Michelle_Nprime*, *Michelle_Qa*, *Michelle_Q*, *Michelle_Si1*, *Michelle_Si2*, *Michelle_Si3*, *Michelle_Si4*, *Michelle_Si5*, *Michelle_Si6*
Gemini	NICI	ED283, ED286, ED299, ED379, ED381, ED449, ED451	*NICI_ED283*, *NICI_ED286*, *NICI_ED299*, *NICI_ED379*, *NICI_ED381*, *NICI_ED449*, *NICI_ED451*
Gemini	NIFS	ZJ, JH, HK	*NIFS_ZJ*, *NIFS_JH*, *NIFS_HK*
Gemini	NIRI	Y, Y cold, Y cold pk50, J, H, HKnotch G0236, Kprime, Kshort, K, Lprime, Mprime, Jcont G0239, HeI, PaGamma, Jcont G0232, PaBeta, PaBeta cold, Hcont, Hcont cold, CH4short, CH4short cold CH4long, CH4long cold, FeII, FeII cold, H2Oice G0242, Kcont G0217, Kcont G0217 cold, H2 1-0, H2 1-0 cold, BrGamma, BrGamma cold, H2 2-1, H2 2-1 cold, Kcont G0219, Kcont G0219 cold, CH4ice, CO2, CO2 cold, H2Oice G0230, Hydrocarbon, BrAlphaCont, BrAlpha, PK50	*NIRI_Y*, *NIRI_Y_cold*, *NIRI_Y_cold_pk50*, *NIRI_J*, *NIRI_H*, *NIRI_HKnotch_G0236*, *NIRI_Kprime*, *NIRI_Kshort*, *NIRI_K*, *NIRI_Lprime*, *NIRI_Mprime*, *NIRI_Jcont_G0239*, *NIRI_HeI*, *NIRI_PaGamma*, *NIRI_Jcont_G0232*, *NIRI_PaBeta*, *NIRI_PaBeta_cold*, *NIRI_Hcont*, *NIRI_Hcont_cold*, *NIRI_CH4short*, *NIRI_CH4short_cold*,`NIRI_CH4long`, *NIRI_CH4long_cold*, *NIRI_FeII*, *NIRI_FeII_cold*, *NIRI_H2Oice_G0242*, *NIRI_Kcont_G0217*, *NIRI_Kcont_G0217_cold*, *NIRI_H2_10*, *NIRI_H2_10_cold*, *NIRI_BrGamma*, *NIRI_BrGamma_cold*, *NIRI_H2_21*, *NIRI_H2_21_cold*, *NIRI_Kcont_G0219*, *NIRI_Kcont_G0219_cold*, *NIRI_CH4ice*, *NIRI_CO2*, *NIRI_CO2_cold*, *NIRI_H2Oice_G0230*, *NIRI_Hydrocarbon*, *NIRI_BrAlphaCont*, *NIRI_BrAlpha*, *NIRI_PK50*

continues on next page

Table 1 – continued from previous page

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
Gemini	TReCS	K, L, M, N, Nprime, Qa, Qb, Qwide, PAH1, PAH2, ArIII, SIV, NeII, NeII cont, Si1, Si2, Si3, Si4, Si5, Si6	*TReCS_K*, *TReCS_L*, *TReCS_M*, *TReCS_N*, *TReCS_Nprime*, *TReCS_Qa*, *TReCS_Qb*, *TReCS_Qwide*, *TReCS_PAH1*, *TReCS_PAH2*, *TReCS_ArIII*, *TReCS_SIV*, *TReCS_NeII*, *TReCS_NeII_cont*, *TReCS_Si1*, *TReCS_Si2*, *TReCS_Si3*, *TReCS_Si4*, *TReCS_Si5*, *TReCS_Si6*
Generic	Bessell	U, B, V, R, I, J, H, K, L, Lp, M	*Bessell_U*, *Bessell_B*, *Bessell_V*, *Bessell_R*, *Bessell_I*, *Bessell_J*, *Bessell_H*, *Bessell_K*, *Bessell_L*, *Bessell_Lp*, *Bessell_M*
Generic	Johnson	U, B, V, R, I, J, H, K, LI, LII, M, N	*Johnson_U*, *Johnson_B*, *Johnson_V*, *Johnson_R*, *Johnson_I*, *Johnson_J*, *Johnson_H*, *Johnson_K*, *Johnson_LI*, *Johnson_LII*, *Johnson_M*, *Johnson_N*
Herschel	PACS	blue, green, red	*PACS_blue*, *PACS_green*, *PACS_red*
Herschel	SPIRE	250, 350, 500	*SPIRE_250*, *SPIRE_350*, *SPIRE_500*
HST	ACS/HRC	F220W, F250W, F330W, F344N, F435W, F475W, F502N, F625W, F775W, 814W, F850LP, F892N, POL UV, POL V	*ACS_HRC_F220W*, *ACS_HRC_F250W*, *ACS_HRC_F330W*, *ACS_HRC_F344N*, *ACS_HRC_F435W*, *ACS_HRC_F475W*, *ACS_HRC_F502N*, *ACS_HRC_F625W*, *ACS_HRC_F775W*, *ACS_HRC_F814W*, *ACS_HRC_F850LP*, *ACS_HRC_F892N*, *ACS_HRC_POL_UV*, *ACS_HRC_POL_V*
HST	ACS/SBC	F115LP, F122M, F125LP, F140LP, F150LP, F165LP	*ACS_F115LP*, *ACS_F122M*, *ACS_F125LP*, *ACS_F140LP*, *ACS_F150LP*, *ACS_F165LP*

continues on next page

Table 1 – continued from previous page

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
HST	ACS/WFC	F435W, F475W, F502N, F550M, F555W, F606W, F625W, F658N, F660N, F775W, F814W, F850LP, F892N, POL UV, POL V	*ACS_F435W*, *ACS_F475W*, *ACS_F502N*, *ACS_F550M*, *ACS_F555W*, *ACS_F606W*, *ACS_F625W*, *ACS_F658N*, *ACS_F660N*, *ACS_F775W*, *ACS_F814W*, *ACS_F850LP*, *ACS_F892N*, *ACS_POL_UV*, *ACS_POL_V*
HST	WFC3/UVIS	F098M, F105W, F110W, F125W, F126N, F127M, F128N, F130N, F132N, F139M, F140W, F153M, F160W, F164N, F167N	*WFC3_F098M*, *WFC3_F105W*, *WFC3_F110W*, *WFC3_F125W*, *WFC3_F126N*, *WFC3_F127M*, *WFC3_F128N*, *WFC3_F130N*, *WFC3_F132N*, *WFC3_F139M*, *WFC3_F140W*, *WFC3_F153M*, *WFC3_F160W*, *WFC3_F164N*, *WFC3_F167N*
HST	WFC3/IR	F200LP, F218W, F225W, F275W, F280N, F300X, F336W, F343N, F350LP, F373N, F390M, F390W, F395N, F410M, F438W, F467M, F469N, F475W, F475X, F487N, F502N, F547M, F555W, F600LP, F606W, F621M, F625W, F631N, F645N, F656N, F657N, F658N, F665N, F673N, F680N, F689M, F763M, F775W, F814W, F845M, F850LP, F953N	*WFC3_F200LP*, *WFC3_F218W*, *WFC3_F225W*, *WFC3_F275W*, *WFC3_F280N*, *WFC3_F300X*, *WFC3_F336W*, *WFC3_F343N*, *WFC3_F350LP*, *WFC3_F373N*, *WFC3_F390M*, *WFC3_F390W*, *WFC3_F395N*, *WFC3_F410M*, *WFC3_F438W*, *WFC3_F467M*, *WFC3_F469N*, *WFC3_F475W*, *WFC3_F475X*, *WFC3_F487N*, *WFC3_F502N*, *WFC3_F547M*, *WFC3_F555W*, *WFC3_F600LP*, *WFC3_F606W*, *WFC3_F621M*, *WFC3_F625W*, *WFC3_F631N*, *WFC3_F645N*, *WFC3_F656N*, *WFC3_F657N*, *WFC3_F658N*, *WFC3_F665N*, *WFC3_F673N*, *WFC3_F680N*, *WFC3_F689M*, *WFC3_F763M*, *WFC3_F775W*, *WFC3_F814W*, *WFC3_F845M*, *WFC3_F850LP*, *WFC3_F953N*
JCMT	SCUBA2	450, 850	*SCUBA2_450*, *SCUBA2_850*
JWST	MIRI	F560W, F770W, F1000W, F1130W, F1280W, F1500W, F1800W, F2100W, F2550W	*MIRI_F560W*, *MIRI_F770W*, *MIRI_F1000W*, *MIRI_F1130W*, *MIRI_F1280W*, *MIRI_F1500W*, *MIRI_F1800W*, *MIRI_F2100W*, *MIRI_F2550W*

continues on next page

Table 1 – continued from previous page

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
JWST	NIRCam	F070W, F090W, F115W, F140M, F150W, F150W2, F162M, F164N, F182M, F187N, F200W, F210M, F212N, F250M, F277W, F300M, F322W2, F323N, F335M, F356W, F360M, F405N, F410M, F430M, F444W, F460M, F466N, F470N, F480M	*NIRCam_F070W*, *NIRCam_F090W*, *NIRCam_F115W*, *NIRCam_F140M*, *NIRCam_F150W*, *NIRCam_F150W2*, *NIRCam_F162M*, *NIRCam_F164N*, *NIRCam_F182M*, *NIRCam_F187N*, *NIRCam_F200W*, *NIRCam_F210M*, *NIRCam_F212N*, *NIRCam_F250M*, *NIRCam_F277W*, *NIRCam_F300M*, *NIRCam_F322W2*, *NIRCam_F323N*, *NIRCam_F335M*, *NIRCam_F356W*, *NIRCam_F360M*, *NIRCam_F405N*, *NIRCam_F410M*, *NIRCam_F430M*, *NIRCam_F444W*, *NIRCam_F460M*, *NIRCam_F466N*, *NIRCam_F470N*, *NIRCam_F480M*
Keck	DEIMOS	B, R, I, Z	*DEIMOS_B*, *DEIMOS_R*, *DEIMOS_I*, *DEIMOS_Z*
Keck	LRIS	B, V, Rs, R, I, GG495, OG570, RG850, NB4000, NB4325, NB5390, NB6741, NB8185, NB8560, NB9135, NB9148	*LRIS_B*, *LRIS_V*, *LRIS_Rs*, *LRIS_R*, *LRIS_I*, *LRIS_GG495*, *LRIS_OG570*, *LRIS_RG850*, *LRIS_NB4000*, *LRIS_NB4325*, *LRIS_NB5390*, *LRIS_NB6741*, *LRIS_NB8185*, *LRIS_NB8560*, *LRIS_NB9135*, *LRIS_NB9148*
Keck	MOSFIRE	Y, J, J2, J3, H, H1, H2, K, Ks	*MOSFIRE_Y*, *MOSFIRE_J*, *MOSFIRE_J2*, *MOSFIRE_J3*, *MOSFIRE_H*, *MOSFIRE_H1*, *MOSFIRE_H2*, *MOSFIRE_K*, *MOSFIRE_Ks*
Keck	NIRC2	J, H, Kp, Ks, K, Lp, Ms, Hcont, FeII, BrGamma, Kcont	*NIRC2_J*, *NIRC2_H*, *NIRC2_Kp*, *NIRC2_Ks*, *NIRC2_K*, *NIRC2_Lp*, *NIRC2_Ms*, *NIRC2_Hcont*, *NIRC2_FeII*, *NIRC2_BrGamma*, *NIRC2_Kcont*

continues on next page

Table 1 – continued from previous page

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
Keck	NIRSPEC	FeII, H2, KL, Lprime, Mwide, Mprime, Nirspec1, Nirspec2, Nirspec3, Nirspec4, Nirspec5, Nirspec6, Nirspec7, K, Kprime	*NIRSPEC_FeII*, *NIRSPEC_H2*, *NIRSPEC_KL*, *NIRSPEC_Lprime*, *NIRSPEC_Mwide*, *NIRSPEC_Mprime*, *NIRSPEC_Nirspec1*, *NIRSPEC_Nirspec2*, *NIRSPEC_Nirspec3*, *NIRSPEC_Nirspec4*, *NIRSPEC_Nirspec5*, *NIRSPEC_Nirspec6*, *NIRSPEC_Nirspec7*, *NIRSPEC_K*, *NIRSPEC_Kprime*
Keck	OSIRIS	Zbb IMAG, Hbb IMAG, Zn3 IMAG, Jn2 IMAG, Jn3 IMAG, Hn1 IMAG, Hn2 IMAG, Hn3 IMAG, Hn4 IMAG, Hn5 IMAG, Jn1 IMAG, Kn1 IMAG, Kn2 IMAG, Kn3 IMAG, Kn4 IMAG, Kn5 IMAG, Zbb SPEC, Jbb SPEC, Hbb SPEC, Kbb SPEC, Zn4 SPEC, Jn1 SPEC, Jn2 SPEC, Jn3 SPEC, Jn4 SPEC, Hn1 SPEC, Hn2 SPEC, Hn3 SPEC, Hn4 SPEC, Hn5 SPEC, Kn1 SPEC, Kn2 SPEC, Kn3 SPEC, Kn4 SPEC, Kn5 SPEC	*OSIRIS_Zbb_IMAG*, *OSIRIS_Hbb_IMAG*, *OSIRIS_Zn3_IMAG*, *OSIRIS_Jn2_IMAG*, *OSIRIS_Jn3_IMAG*, *OSIRIS_Hn1_IMAG*, *OSIRIS_Hn2_IMAG*, *OSIRIS_Hn3_IMAG*, *OSIRIS_Hn4_IMAG*, *OSIRIS_Hn5_IMAG*, *OSIRIS_Jn1_IMAG*, *OSIRIS_Kn1_IMAG*, *OSIRIS_Kn2_IMAG*, *OSIRIS_Kn3_IMAG*, *OSIRIS_Kn4_IMAG*, *OSIRIS_Kn5_IMAG*, *OSIRIS_Zbb_SPEC*, *OSIRIS_Jbb_SPEC*, *OSIRIS_Hbb_SPEC*, *OSIRIS_Kbb_SPEC*, *OSIRIS_Zn4_SPEC*, *OSIRIS_Jn1_SPEC*, *OSIRIS_Jn2_SPEC*, *OSIRIS_Jn3_SPEC*, *OSIRIS_Jn4_SPEC*, *OSIRIS_Hn1_SPEC*, *OSIRIS_Hn2_SPEC*, *OSIRIS_Hn3_SPEC*, *OSIRIS_Hn4_SPEC*, *OSIRIS_Hn5_SPEC*, *OSIRIS_Kn1_SPEC*, *OSIRIS_Kn2_SPEC*, *OSIRIS_Kn3_SPEC*, *OSIRIS_Kn4_SPEC*, *OSIRIS_Kn5_SPEC*

continues on next page

Table 1 – continued from previous page

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
KPNO	FLAMINGOS	J, JH, H, HK, Ks, K	*FLAMINGOS_J*, *FLAMINGOS_JH*, *FLAMINGOS_H*, *FLAMINGOS_HK*, *FLAMINGOS_Ks*, *FLAMINGOS_K*
KPNO	Mosaic	U, B, V, R, I, Ud, Un, Us, Bw, VR, Rs, C, M, r DECam, Z DECam, g SDSS, r SDSS, i SDSS, z SDSS, white, WR475, 337, 390, 420, 454, 493, 527, 579, 607, 666, 705, 755, 802, 815 v1, 815 v2, 823 v1, 823 v2, 848, 918, 918R, 973, CIII, CIV, D51, Gn, HAlpha, HAlpha12, HAlpha16, HAlpha4, HAlpha8, HeII, OII30, OIII	*Mosaic_U*, *Mosaic_B*, *Mosaic_V*, *Mosaic_R*, *Mosaic_I*, *Mosaic_Ud*, *Mosaic_Un*, *Mosaic_Us*, *Mosaic_Bw*, *Mosaic_VR*, *Mosaic_Rs*, *Mosaic_C*, *Mosaic_M*, *Mosaic_r_DECam*, *Mosaic_Z_DECam*, *Mosaic_g_SDSS*, *Mosaic_r_SDSS*, *Mosaic_i_SDSS*, *Mosaic_z_SDSS*, *Mosaic_white*, *Mosaic_WR475*, *Mosaic_337*, *Mosaic_390*, *Mosaic_420*, *Mosaic_454*, *Mosaic_493*, *Mosaic_527*, *Mosaic_579*, *Mosaic_607*, *Mosaic_666*, *Mosaic_705*, *Mosaic_755*, *Mosaic_802*, *Mosaic_815_v1*, *Mosaic_815_v2*, *Mosaic_823_v1*, *Mosaic_823_v2*, *Mosaic_848*, *Mosaic_918*, *Mosaic_918R*, *Mosaic_973*, *Mosaic_CIII*, *Mosaic_CIV*, *Mosaic_D51*, *Mosaic_Gn*, *Mosaic_HAlpha*, *Mosaic_HAlpha12*, *Mosaic_HAlpha16*, *Mosaic_HAlpha4*, *Mosaic_HAlpha8*, *Mosaic_HeII*, *Mosaic_OII30*, *Mosaic_OIII*
KPNO	NEWFIRM	J1, J2, J3, H1, H2, Ks	*NEWFIRM_J1*, *NEWFIRM_J2*, *NEWFIRM_J3*, *NEWFIRM_H1*, *NEWFIRM_H2*, *NEWFIRM_Ks*
KPNO	WIYN/0.9m	U, B, V, R, I, u Strom, b Strom, v Strom, y Strom, Ca, Hb narrow, Hb wide, HAlpha, HAlpha12, HAlpha16, HAlpha4, HAlpha8	*0.9m_U*, *0.9m_B*, *0.9m_V*, *0.9m_R*, *0.9m_I*, *0.9m_u_Strom*, *0.9m_b_Strom*, *0.9m_v_Strom*, *0.9m_y_Strom*, *0.9m_Ca*, *0.9m_Hb_narrow*, *0.9m_Hb_wide*, *0.9m_HAlpha*, *0.9m_HAlpha12*, *0.9m_HAlpha16*, *0.9m_HAlpha4*, *0.9m_HAlpha8*
KPNO	WIYN/ODI	u, g, r, i, z, NB422, NB659, NB695, NB746	*ODI_u*, *ODI_g*, *ODI_r*, *ODI_i*, *ODI_z*, *ODI_NB422*, *ODI_NB659*, *ODI_NB695*, *ODI_NB746*

continues on next page

Table 1 – continued from previous page

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
KPNO	WIYN/WHIRC	J, H, Ks, Low, HeI, CN, PaBeta, PaBeta45, FeII, FeII45, H2, BrGamma, BrGamma45, CO-n, CO-w	*WHIRC_J*, *WHIRC_H*, *WHIRC_Ks*, *WHIRC_Low*, *WHIRC_HeI*, *WHIRC_CN*, *WHIRC_PaBeta*, *WHIRC_PaBeta45*, *WHIRC_FeII*, *WHIRC_FeII45*, *WHIRC_H2*, *WHIRC_BrGamma*, *WHIRC_BrGamma45*, *WHIRC_CO-n*, *WHIRC_CO-w*
LaSilla	EMMI	Bb, V, R, I, Z, g Gunn, r Gunn, z Gunn, U Bessel, B Bessel, B Tyson, BG38, BG39, EUV, GG375, OG530, RG715, OII, OII10, OII15, OII5, HeII 652, HeII 671, HBeta cont, HBeta, OIII, OIII12, OIII15, OIII3, OIII6, OIII9, HAlpha, HAlpha0, HAlpha12, HAlpha15, HAlpha3, HAlpha6, HAlpha9, NII 653, NII 595, SII, SIII, NeV, Spec 656, Spec 672, Spec 673, Spec 723, Spec 765, Spec 766, Spec 767, Spec 768, Spec 795, Spec 796	*EMMI_Bb*, *EMMI_V*, *EMMI_R*, *EMMI_I*, *EMMI_Z*, *EMMI_g_Gunn*, *EMMI_r_Gunn*, *EMMI_z_Gunn*, *EMMI_U_Bessel*, *EMMI_B_Bessel*, *EMMI_B_Tyson*, *EMMI_BG38*, *EMMI_BG39*, *EMMI_EUV*, *EMMI_GG375*, *EMMI_OG530*, *EMMI_RG715*, *EMMI_OII*, *EMMI_OII10*, *EMMI_OII15*, *EMMI_OII5*, *EMMI_HeII_652*, *EMMI_HeII_671*, *EMMI_HBeta_cont*, *EMMI_HBeta*, *EMMI_OIII*, *EMMI_OIII12*, *EMMI_OIII15*, *EMMI_OIII3*, *EMMI_OIII6*, *EMMI_OIII9*, *EMMI_HAlpha*, *EMMI_HAlpha0*, *EMMI_HAlpha12*, *EMMI_HAlpha15*, *EMMI_HAlpha3*, *EMMI_HAlpha6*, *EMMI_HAlpha9*, *EMMI_NII_653*, *EMMI_NII_595*, *EMMI_SII*, *EMMI_SIII*, *EMMI_NeV*, *EMMI_Spec_656*, *EMMI_Spec_672*, *EMMI_Spec_673*, *EMMI_Spec_723*, *EMMI_Spec_765*, *EMMI_Spec_766*, *EMMI_Spec_767*, *EMMI_Spec_768*, *EMMI_Spec_795*, *EMMI_Spec_796*

continues on next page

Table 1 – continued from previous page

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
LaSilla	SOFI	Js, J, H, Ks, K, NB106, NB108, NB119, NB121, NB171, NB209, NB219, NB225, NB228, NB FeII J, NB Pbeta, NB FeII H, NB HeI K, NB H2, NB BrGamma, NB CO	*SOFI_Js*, *SOFI_J*, *SOFI_H*, *SOFI_Ks*, *SOFI_K*, *SOFI_NB106*, *SOFI_NB108*, *SOFI_NB119*, *SOFI_NB121*, *SOFI_NB171*, *SOFI_NB209*, *SOFI_NB219*, *SOFI_NB225*, *SOFI_NB228*, *SOFI_NB_FeII_J*, *SOFI_NB_Pbeta*, *SOFI_NB_FeII_H*, *SOFI_NB_HeI_K*, *SOFI_NB_H2*, *SOFI_NB_BrGamma*, *SOFI_NB_CO*
LaSilla	SUSI2	U, Uprime, B, B spare, V, R, I, Z, HeII, HBeta, OIII, OIII cont, HAlpha, Halpha cont, WB 490, IB 609, WB 655, IB 662	*SUSI2_U*, *SUSI2_Uprime*, *SUSI2_B*, *SUSI2_B_spare*, *SUSI2_V*, *SUSI2_R*, *SUSI2_I*, *SUSI2_Z*, *SUSI2_HeII*, *SUSI2_HBeta*, *SUSI2_OIII*, *SUSI2_OIII_cont*, *SUSI2_HAlpha*, *SUSI2_Halpha_cont*, *SUSI2_WB_490*, *SUSI2_IB_609*, *SUSI2_WB_655*, *SUSI2_IB_662*
LaSilla	TIMMI2	N79 OCLI, N1, N89 OCLI, N98 OCLI, N104 OCLI, SIV, N2, N119 OCLI, SiC, N129 OCLI, NEII, Q1, Q2, IRWBP	*TIMMI2_N79_OCLI*, *TIMMI2_N1*, *TIMMI2_N89_OCLI*, *TIMMI2_N98_OCLI*, *TIMMI2_N104_OCLI*, *TIMMI2_SIV*, *TIMMI2_N2*, *TIMMI2_N119_OCLI*, *TIMMI2_SiC*, *TIMMI2_N129_OCLI*, *TIMMI2_NEII*, *TIMMI2_Q1*, *TIMMI2_Q2*, *TIMMI2_IRWBP*

continues on next page

Table 1 – continued from previous page

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
LaSilla	WFI	U, U 38, B, B 99, V, Rc, I, Ic, Z+, Gieren 501, M Wash, MB416, MB445, MB461, MB485, MB513, MB516, MB518, MB531, MB549, MB571, MB604, MB620, MB646, MB679, MB696, MB721, MB753, MB770, MB790, MB815, MB837, MB856, MB884, MB914, NB396, NB504, NB665, NB810, NB817, NB824, OIII, OIII 2, Halpha, SIIr, white	*WFI_U*, *WFI_U_38*, *WFI_B*, *WFI_B_99*, *WFI_V*, *WFI_Rc*, *WFI_I*, *WFI_Ic*, *WFI_Z+*, *WFI_Gieren_501*, *WFI_M_Wash*, *WFI_MB416*, *WFI_MB445*, *WFI_MB461*, *WFI_MB485*, *WFI_MB513*, *WFI_MB516*, *WFI_MB518*, *WFI_MB531*, *WFI_MB549*, *WFI_MB571*, *WFI_MB604*, *WFI_MB620*, *WFI_MB646*, *WFI_MB679*, *WFI_MB696*, *WFI_MB721*, *WFI_MB753*, *WFI_MB770*, *WFI_MB790*, *WFI_MB815*, *WFI_MB837*, *WFI_MB856*, *WFI_MB884*, *WFI_MB914*, *WFI_NB396*, *WFI_NB504*, *WFI_NB665*, *WFI_NB810*, *WFI_NB817*, *WFI_NB824*, *WFI_OIII*, *WFI_OIII_2*, *WFI_Halpha*, *WFI_SIIr*, *WFI_white*
LBT	LBCB	U, B, V, gSloan, rSloan, USpec15, USpec40, USpec65	*LBCB_U*, *LBCB_B*, *LBCB_V*, *LBCB_gSloan*, *LBCB_rSloan*, *LBCB_USpec15*, *LBCB_USpec40*, *LBCB_USpec65*
LBT	LBCR	V, R, I, Y, rSloan, iSloan, zSloan, F972N20, CN, TiO	*LBCR_V*, *LBCR_R*, *LBCR_I*, *LBCR_Y*, *LBCR_rSloan*, *LBCR_iSloan*, *LBCR_zSloan*, *LBCR_F972N20*, *LBCR_CN*, *LBCR_TiO*
LBT	MODS	uMODS1, uMODS2, gMODS1, gMODS2, rMODS1, rMODS2, iMODS1, iMODS2, zMODS1, zMODS2	*MODS_uMODS1*, *MODS_uMODS2*, *MODS_gMODS1*, *MODS_gMODS2*, *MODS_rMODS1*, *MODS_rMODS2*, *MODS_iMODS1*, *MODS_iMODS2*, *MODS_zMODS1*, *MODS_zMODS2*
LSST		u, g, r, i, z, y	*LSST_u*, *LSST_g*, *LSST_r*, *LSST_i*, *LSST_z*, *LSST_y*

continues on next page

Table 1 – continued from previous page

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
Pan-STARRS		open, gp1, rp1, ip1, zp1, yp1, wp1	*Pan-STARRS_open*, *Pan-STARRS_gp1*, *Pan-STARRS_rp1*, *Pan-STARRS_ip1*, *Pan-STARRS_zp1*, *Pan-STARRS_yp1*, *Pan-STARRS_wp1*
Paranal	VISTA/VIRCAM	Y, Z, J, H, Ks, NB980, NB990, NB118	*VIRCAM_Y*, *VIRCAM_Z*, *VIRCAM_J*, *VIRCAM_H*, *VIRCAM_Ks*, *VIRCAM_NB980*, *VIRCAM_NB990*, *VIRCAM_NB118*
Paranal	VLT/FORS	U BESS, B BESS, V BESS, R BESS, I BESS, u HIGH, b HIGH, g HIGH, v HIGH, U GUNN, G GUNN, V GUNN, R GUNN, Z GUNN, U SPECIAL, R SPECIAL, GG3751, GG3752, GG4351, GG4352, IF485, IF691, IF815, IF834, IF915, OG5901, OG5902	*FORS_U_BESS*, *FORS_B_BESS*, *FORS_V_BESS*, *FORS_R_BESS*, *FORS_I_BESS*, *FORS_u_HIGH*, *FORS_b_HIGH*, *FORS_g_HIGH*, *FORS_v_HIGH*, *FORS_U_GUNN*, *FORS_G_GUNN*, *FORS_V_GUNN*, *FORS_R_GUNN*, *FORS_Z_GUNN*, *FORS_U_SPECIAL*, *FORS_R_SPECIAL*, *FORS_GG3751*, *FORS_GG3752*, *FORS_GG4351*, *FORS_GG4352*, *FORS_IF485*, *FORS_IF691*, *FORS_IF815*, *FORS_IF834*, *FORS_IF915*, *FORS_OG5901*, *FORS_OG5902*
Paranal	VLT/HAWK-I	Y, J, H, Ks, CH4, H2, BrGamma, NB1060, NB1190, NB2090	*HAWK-I_Y*, *HAWK-I_J*, *HAWK-I_H*, *HAWK-I_Ks*, *HAWK-I_CH4*, *HAWK-I_H2*, *HAWK-I_BrGamma*, *HAWK-I_NB1060*, *HAWK-I_NB1190*, *HAWK-I_NB2090*

continues on next page

Table 1 – continued from previous page

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
Paranal	VLT/ISAAC	SZ, Js, J, H, Ks, L, M, NB1061, NB1083, NB1187, NB1215, NB1257, NB1282, NB1644, NB1710, NB2058, NB2090, NB2122, NB2188, NB2195, NB2248, NB2280, NB2336, NB3200, NB3280, NB3800, NB4050	*ISAAC_SZ*, *ISAAC_Js*, *ISAAC_J*, *ISAAC_H*, *ISAAC_Ks*, *ISAAC_L*, *ISAAC_M*, *ISAAC_NB1061*, *ISAAC_NB1083*, *ISAAC_NB1187*, *ISAAC_NB1215*, *ISAAC_NB1257*, *ISAAC_NB1282*, *ISAAC_NB1644*, *ISAAC_NB1710*, *ISAAC_NB2058*, *ISAAC_NB2090*, *ISAAC_NB2122*, *ISAAC_NB2188*, *ISAAC_NB2195*, *ISAAC_NB2248*, *ISAAC_NB2280*, *ISAAC_NB2336*, *ISAAC_NB3200*, *ISAAC_NB3280*, *ISAAC_NB3800*, *ISAAC_NB4050*
Paranal	VLT/MIDI	Nband, SiC, N87, ArIII, SIV, N113, NeII	*MIDI_Nband*, *MIDI_SiC*, *MIDI_N87*, *MIDI_ArIII*, *MIDI_SIV*, *MIDI_N113*, *MIDI_NeII*
Paranal	VLT/NACO	J, H, Ks, Lprime, Mprime, NB104, NB108, NB109, NB124, NB126, NB128, NB164, NB175, NB212, NB217, NB374, NB405, IB200, IB203, IB206, IB209, IB212, IB215, IB218, IB221, IB224, IB227, IB230, IB233, IB236, IB239, IB242, IB245, IB248	*NACO_J*, *NACO_H*, *NACO_Ks*, *NACO_Lprime*, *NACO_Mprime*, *NACO_NB104*, *NACO_NB108*, *NACO_NB109*, *NACO_NB124*, *NACO_NB126*, *NACO_NB128*, *NACO_NB164*, *NACO_NB175*, *NACO_NB212*, *NACO_NB217*, *NACO_NB374*, *NACO_NB405*, *NACO_IB200*, *NACO_IB203*, *NACO_IB206*, *NACO_IB209*, *NACO_IB212*, *NACO_IB215*, *NACO_IB218*, *NACO_IB221*, *NACO_IB224*, *NACO_IB227*, *NACO_IB230*, *NACO_IB233*, *NACO_IB236*, *NACO_IB239*, *NACO_IB242*, *NACO_IB245*, *NACO_IB248*

continues on next page

Table 1 – continued from previous page

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
Paranal	VLT/SPHERE/IRDIS	Y, J, H, Ks, Y23, J23, H23, ND_H23, H34, K12, HeI, CntJ, PaB, CntH, FeII, CntK1, H2, BrG, CntK2, CO	*IRDIS_Y*, *IRDIS_J*, *IRDIS_H*, *IRDIS_Ks*, *IRDIS_Y23*, *IRDIS_J23*, *IRDIS_H23*, *IRDIS_ND_H23*, *IRDIS_H34*, *IRDIS_K12*, *IRDIS_HeI*, *IRDIS_CntJ*, *IRDIS_PaB*, *IRDIS_CntH*, *IRDIS_FeII*, *IRDIS_CntK1*, *IRDIS_H2*, *IRDIS_BrG*, *IRDIS_CntK2*, *IRDIS_CO*
Paranal	VLT/SPHERE/ZIMP(	VBB, R PRIM, I PRIM, V, V S, V L, N R, NB730, N I, I L, KI, TiO, CH4, Cnt748, Cnt820, Ha B, Ha Cnt, HeI, OI 630, N Ha	*ZIMPOL_VBB*, *ZIMPOL_R_PRIM*, *ZIMPOL_I_PRIM*, *ZIMPOL_V*, *ZIMPOL_V_S*, *ZIMPOL_V_L*, *ZIMPOL_N_R*,`ZIMPOL_NB730`, *ZIMPOL_N_I*, *ZIMPOL_I_L*, *ZIMPOL_KI*, *ZIMPOL_TiO*, *ZIMPOL_CH4*, *ZIMPOL_Cnt748*, *ZIMPOL_Cnt820*, *ZIMPOL_Ha_B*, *ZIMPOL_Ha_Cnt*, *ZIMPOL_HeI*, *ZIMPOL_OI_630*, *ZIMPOL_N_Ha*
Paranal	VLT/VIMOS	U, B, V, R, I, z	*VIMOS_U*, *VIMOS_B*, *VIMOS_V*, *VIMOS_R*, *VIMOS_I*, *VIMOS_z*
Paranal	VLT/VISIR	M, J79, PAH1, J89, B87, ArIII, SIV 1, B97, SIV, B107, SIV 2, PAH2, B117, PAH2 2, J122, NeII 1, B124, NeII, NeII 2, Q1, Q2, Q3	*VISIR_M*, *VISIR_J79*, *VISIR_PAH1*, *VISIR_J89*, *VISIR_B87*, *VISIR_ArIII*, *VISIR_SIV_1*, *VISIR_B97*, *VISIR_SIV*, *VISIR_B107*, *VISIR_SIV_2*, *VISIR_PAH2*, *VISIR_B117*, *VISIR_PAH2_2*, *VISIR_J122*, *VISIR_NeII_1*, *VISIR_B124*, *VISIR_NeII*, *VISIR_NeII_2*, *VISIR_Q1*, *VISIR_Q2*, *VISIR_Q3*
Paranal	VST/OmegaCAM	B, V, V Strom, u, g, r, i, z, HAlpha, Cal	*OmegaCAM_B*, *OmegaCAM_V*, *OmegaCAM_V_Strom*, *OmegaCAM_u*, *OmegaCAM_g*, *OmegaCAM_r*, *OmegaCAM_i*, *OmegaCAM_z*, *OmegaCAM_HAlpha*, *OmegaCAM_Cal*
Planck	HFI	F100, F143, F217, F353, F545, F857	*HFI_F100*, *HFI_F143*, *HFI_F217*, *HFI_F353*, *HFI_F545*, *HFI_F857*

continues on next page

Table 1 – continued from previous page

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
Planck	LFI	F030, F044, F070	*LFI_F030*, *LFI_F044*, *LFI_F070*
SDSS		u, g, r, i, z	*SDSS_u*, *SDSS_g*, *SDSS_r*, *SDSS_i*, *SDSS_z*
SOFIA	FLITECAM	J, H, K, Lprime, L, M, PaAlpha, PaAlphaCont, Ice, PAH, LNarrow, MNarrow, Hwide, Kwide, Klong, KplusM	*FLITECAM_J*, *FLITECAM_H*, *FLITECAM_K*, *FLITECAM_Lprime*, *FLITECAM_L*, *FLITECAM_M*, *FLITECAM_PaAlpha*, *FLITECAM_PaAlphaCont*, *FLITECAM_Ice*, *FLITECAM_PAH*, *FLITECAM_LNarrow*, *FLITECAM_MNarrow*, *FLITECAM_Hwide*, *FLITECAM_Kwide*, *FLITECAM_Klong*, *FLITECAM_KplusM*
SOFIA	FORCAST	F054, F064, F066, F077, F111, F113, F197, F242, F315, F336, F348, F371	*FORCAST_F054*, *FORCAST_F064*, *FORCAST_F066*, *FORCAST_F077*, *FORCAST_F111*, *FORCAST_F113*, *FORCAST_F197*, *FORCAST_F242*, *FORCAST_F315*, *FORCAST_F336*, *FORCAST_F348*, *FORCAST_F371*
SOFIA	HAWC+	bandA, bandB, bandC, bandD, bandE	*HAWC+_bandA*, *HAWC+_bandB*, *HAWC+_bandC*, *HAWC+_bandD*, *HAWC+_bandE*
Spitzer	IRAC	CH1, CH2, CH3, CH4	*IRAC_CH1*, *IRAC_CH2*, *IRAC_CH3*, *IRAC_CH4*
Spitzer	IRS	BLUE-PeakUp, RED-PeakUp	*IRS_BLUE-PU*, *IRS_RED-PU*
Spitzer	MIPS	CH1, CH2, CH3	*MIPS_CH1*, *MIPS_CH2*, *MIPS_CH3*
SPT	SPT-SZ	150, 220	*SPT-SZ_150*, *SPT-SZ_220*
Steward	Bok/90Prime	U, uprime, B, V, R, I, zprime, Washington M	*90prime_U.txt*, *90prime_uprime.txt*, *90prime_B.txt*, *90prime_V.txt*, *90prime_R.txt*, *90prime_I.txt*, *90prime_zprime.txt*, *90Prime_Washington_M.txt*
Subaru	CIAO	J, H, K, Lprime, Mprime, PAH	*CIAO_J*, *CIAO_H*, *CIAO_K*, *CIAO_Lprime*, *CIAO_Mprime*, *CIAO_PAH*

continues on next page

Table 1 – continued from previous page

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
Subaru	CISCO-OHS	z, J, H, Kprime, K, N204, H2 1-0, N215, H2 2-1	*CISCO_OHS_z*, *CISCO_OHS_J*, *CISCO_OHS_H*, *CISCO_OHS_Kprime*, *CISCO_OHS_K*, *CISCO_OHS_N204*, *CISCO_OHS_H2_10*, *CISCO_OHS_N215*, *CISCO_OHS_H2_21*
Subaru	FOCAS	U, B, V, R, I, N373, N386, N487, N502, N512, N642, N658, N670	*FOCAS_U*, *FOCAS_B*, *FOCAS_V*, *FOCAS_R*, *FOCAS_I*, *FOCAS_N373*, *FOCAS_N386*, *FOCAS_N487*, *FOCAS_N502*, *FOCAS_N512*, *FOCAS_N642*, *FOCAS_N658*, *FOCAS_N670*
Subaru	HSC	g, r, r new, i, i new, z, Y, IB945, NB387, NB391, NB395, NB400, NB430, NB468, NB497, NB515, NB527, NB656, NB718, NB816, NB921, NB926, NB973, NB1010	*HSC_g*, *HSC_r*, *HSC_r_new*, *HSC_i*, *HSC_i_new*, *HSC_z*, *HSC_Y*, *HSC_IB945*, *HSC_NB387*, *HSC_NB391*, *HSC_NB395*, *HSC_NB400*, *HSC_NB430*, *HSC_NB468*, *HSC_NB497*, *HSC_NB515*, *HSC_NB527*, *HSC_NB656*, *HSC_NB718*, *HSC_NB816*, *HSC_NB921*, *HSC_NB926*, *HSC_NB973*, *HSC_NB1010*
Subaru	IRCS	J, H, Kprime, K, Lprime, Mprime, NB1189, CH4 long, CH4 short, FeII, NB1984, H2, BrGamma, H2Oice	*IRCS_J*, *IRCS_H*, *IRCS_K*, *IRCS_Kprime*, *IRCS_Lprime*, *IRCS_Mprime*, *IRCS_NB1189*, *IRCS_CH4_long*, *IRCS_CH4_short*, *IRCS_FeII*, *IRCS_NB1984*, *IRCS_H2*, *IRCS_BrGamma*, *IRCS_H2Oice*
Subaru	MOIRCS	Y, J, H, Ks, K, NB119, NB1550, NB1657, FeII, NB2071, NB2083, NB2095, H2, BrGamma, Kcont, CO, NB2315	*MOIRCS_Y*, *MOIRCS_J*, *MOIRCS_H*, *MOIRCS_Ks*, *MOIRCS_K*, *MOIRCS_NB119*, *MOIRCS_NB1550*, *MOIRCS_NB1657*, *MOIRCS_FeII*, *MOIRCS_NB2071*, *MOIRCS_NB2083*, *MOIRCS_NB2095*, *MOIRCS_H2*, *MOIRCS_BrGamma*, *MOIRCS_Kcont*, *MOIRCS_CO*, *MOIRCS_NB2315*

continues on next page

Table 1 – continued from previous page

Observatory/Survey	Instrument	Bands	Lightning Filter Labels
Subaru	SuprimeCam	B, V, Rc, Ic, gprime, iprime, rprime, zprime, Y, NA656, NB711, NB816, NB921	*SuprimeCam_B*, *SuprimeCam_V*, *SuprimeCam_Rc*, *SuprimeCam_Ic*, *SuprimeCam_gprime*, *SuprimeCam_iprime*, *SuprimeCam_rprime*, *SuprimeCam_zprime*, *SuprimeCam_Y*, *SuprimeCam_NA656*, *SuprimeCam_NB711*, *SuprimeCam_NB816*, *SuprimeCam_NB921*
Swift	UVOT	UVW2, UVM2, UVW1, U, B, V, white	*UVOT_UVW2*, *UVOT_UVM2*, *UVOT_UVW1*, *UVOT_U*, *UVOT_B*, *UVOT_V*, *UVOT_white*
UKIRT	WFCAM	Z, Y, J, H, K, H2, BrGamma	*WFCAM_Z*, *WFCAM_Y*, *WFCAM_J*, *WFCAM_H*, *WFCAM_K*, *WFCAM_H2*, *WFCAM_BrGamma*
WISE		W1, W2, W3, W4	*WISE_W1*, *WISE_W2*, *WISE_W3*, *WISE_W4*

CHAPTER

EIGHT

API REFERENCE

8.1 Lightning

class lightning.**Lightning**

The main interface for fitting a galaxy SED model.

Holds information about the filters, observed flux, type of model. Model components are set by keyword choices.

The only strictly required parameters are `filter_labels` and one of `redshift` or `lum_dist`. For inference/fitting, both `flux_obs` and `flux_unc` must also be set.

Parameters

filter_labels
[list, str] List of filter labels.

redshift
[float] Redshift of the model. If set, `lum_dist` is ignored. (Default: None)

lum_dist
[float] Luminosity distance to the model. If `redshift` is not set, this parameter must be. If a luminosity distance is provided instead of a redshift, the redshift is set to 0 (as we assume the galaxy is very nearby). (Default: None)

flux_obs
[np.ndarray, (Nfilters,) or (Nfilters, 2), float32, optional] The observed flux densities in *mJy*, or, optionally, the observed flux densities and associated uncertainties as a 2D array. (Default: None)

flux_unc
[np.ndarray, (Nfilters,), float32, optional] The uncertainties associated with `flux_obs`, in *mJy*. (Default: None)

wave_grid
[tuple (3,), or np.ndarray, (Nwave,), float32, optional] Either a tuple of (lo, hi, Nwave) specifying a log-spaced rest-frame wavelength grid, or an array giving the wavelengths directly. (Default: (0.1, 1000, 1200))

stellar_type
[{'PEGASE', 'PEGASE-A24', 'BPASS', 'BPASS-A24', 'BPASS-ULX-G24'}] String specifying the simple stellar population models to use. 'PEGASE' gives the stellar population models used in IDL Lightning (Doore+2023). (Default: 'PEGASE')

line_labels
[np.ndarray, (Nlines,), string, optional] Line labels in the format used by pyCloudy. See

lightning/models/linelist_full.txt for the complete list of available lines in the grid and their format. (Default: None)

line_flux
[np.ndarray, (Nlines,) or (Nlines, 2), float32, optional] Observed integrated fluxes of the lines in *erg cm-2 s-1* (unless the `line_index` keyword is set, in which case they should be normalized by the appropriate line). (Default: None)

line_flux_unc
[np.ndarray,(Nlines,), float32, optional] Uncertainties on the integrated line fluxes in *erg cm-2 s-1* (unless `line_index` is set, see above) (Default: None)

line_index
[string] Label for the line to index the line fluxes by. You would normally want this to be `'H__1_486132A'` or `'H__1_656280A'` for Hbeta and Halpha, respectively, but the highest SNR line could be a good choice. If this is set to `None` then the model line fluxes are not normalized for fitting. (Default: None)

nebula_lognH
[float] log of the Hydrogen density in cm-3 for the Cloudy grids. (Default: 2.0)

nebula_dust
[bool] If `True`, then dust grains are included in the Cloudy grids. (Default: False)

SFH_type
[{'Piecewise-Constant', 'Delayed-Exponential', 'Single-Exponential', 'Burst'}] String specifying the SFH type to use. (Default: 'Piecewise-Constant')

ages
[np.ndarray, (Nages,), float32] Array giving the stellar ages (or stellar age bins) of the stellar population models. Default depends on the model and redshift.

atten_type
[{'Modified-Calzetti', 'Calzetti'}] String specifying the dust attenuation model to use. (Default: 'Modified-Calzetti')

dust_emission
[bool] If `True`, a Draine & Li (2007) dust emission model is included, in energy balance with the attenuated power. (Default: False)

agn_emission
[bool] If `True`, a Stalevski et al. (2016) UV-IR AGN emission model is included. (Default: False)

agn_polar_dust
[bool] If `True`, AGN polar dust extinction and re-emission are implemented following the recipe from X-Cigale. Note that even if this keyword is set to `False`, the polar dust optical depth remains a parameter of the model - it just doesn't do anything, and should held at 0 in any fitting. (Default: False)

xray_mode
[{'flux', 'counts', 'None', None}] String specifying the mode for X-ray model fitting - fluxes are fit like any other point in the SED, while counts are folded through the instrumental response. Expect counts mode to be more flexible but harder to set up. If you are fitting X-ray data from multiple instruments simultaneously, you must set this to 'flux' (and should convert your instrumental countrates to fluxes assuming the same spectrum for every instrument). (Default: None)

xray_stellar_emission
[{'Stellar-Plaw', 'None', None}] String specifying the X-ray stellar model. Currently the only

available model is 'Stellar-Plaw', which uses the Gilbertson+(2022) LX-stellar age scaling relation to calculate the luminosity. (Default: None)

xray_agn_emission
[{'AGN-Plaw', 'QSOSED', 'None', None}] String specifying the X-ray AGN model. The 'AGN-Plaw' model is a power law sclaed by the Lusso & Risaliti (2017) empirical L2500-L2keV relationship; the QSOSED model is a theoretical accretion disk + comptonization model that sets the normalization of the whole X-ray-to-IR AGN model as a function of M and mdot. (Default: None)

xray_absorption
[{'tbabs', 'phabs', 'None', None}] String specifying the X-ray absorption model to use. Two instances are used, one in the rest frame describing the intrinsic absorption, and one in the observed frame with a constant Galactic NH. (Default: None)

xray_wave_grid
[tuple (3,), or np.ndarray, (Nwave,), float32, optional] Either a tuple of (lo, hi, Nwave) specifying a log-spaced rest-frame wavelength grid, or an array giving the wavelengths directly. At high redshift this should be constructed carefully to ensure that your bands are covered. (Default: (1e-6, 1e-1, 200))

xray_arf
[dict or astropy.table.Table or numpy structured array] A structure defining the anciliary response function (ARF) of your X-ray observations. The structure must have three keys, *'ENERG_LO'*, *'ENERG_HI'*, and *'SPECRESP'*, which given the energy bins and binned spectral response respectively. Only used if *xray_mode='counts'* `. (Default: None)

xray_exposure
[float or np.ndarray (Nfilters)] A scalar or array giving the exposure time of the X-ray observations. If an array, it should have the same length as `filter_labels`, with all non-X-ray bands having their exposure time set to 0. Note that you almost certainly don't need to give exposure time as an array, since the energy dependence of the effective area is explicitly given by the ARF. Only used if `xray_mode='counts'`. (Default: None)

galactic_NH
[float] A scalar giving the Galactic column density along the line of sight to the source in *1e20 cm-2*. (Default: 0.0)

lightning_filter_path
[str] Path to lightning filters. Not actually used.

print_setup_time
[bool] If `True`, the setup time will be printed. (Default: False)

model_unc
[np.ndarray, (Nfilters,), float, or float] Fractional (i.e. [0,1)) model uncertainty to include in the fits. If a scalar is provided, the same model uncertainty is applied to each filter. Alternately, model uncertainties can be provided as an array, one per filter. The smarter way to do this in the future may be to set model uncertainties per component, rather than per filter. (Default: None)

model_unc_lines
[np.ndarray, (Nlines,), float, or float] Fractional (i.e. [0,1)) model uncertainty to include in the line fits. (Default: None)

cosmology
[astropy.cosmology.FlatLambdaCDM] The cosmology to assume. Lightning defaults to a flat cosmology with `h=0.7 and Om0=0.3`.

uplim_handling
[{'exact', 'approx'}] A string specifying how upper limits are to be handled. In the case of 'exact' we treat them as derived in e.g. Appendix A of Sawicki et al. (2012). In the case of 'approx', upper limits are treated as measurements of 0 with a 1 sigma uncertainty equal to the 1 sigma flux limit. This option is provided for consistency with the handling of limits in IDL lightning. Our convention for inputting upper limits remains that they should be specified as a measurement of 0 with a 1 sigma uncertainty equal to the 1 sigma flux limit. (Default: 'exact')

Attributes

flux_obs
Observed flux-densities in mJy.

flux_unc
Uncertainties associated with `flux_obs`.

Lnu_obs
[None or numpy.ndarray, (Nfilters,), float32] Observed-frame luminosity densities, converted from the given fluxes.

Lnu_unc
[None or numpy.ndarray, (Nfilters,), float32] Uncertainties associated with `Lnu_obs`.

filter_labels
[list, str] List of filter labels.

filters
[dict, len=Nfilters, float32] A dict of numpy float32 arrays. The keys are *filter_labels* and the values correspond to the normalized transmission evaluated on *wave_grid*.

wave_obs
[numpy.ndarray, (Nfilters,), float32] Mean wavelength of the the supplied filters.

redshift
[float] Redshift of the model. Assumed to be 0 if *lum_dist* was set.

DL
[float] Luminosity distance to the model, in Mpc

wave_grid_rest
[numpy.ndarray, (Nwave,), float32] Unified rest-frame wavelength grid for the models.

wave_grid_obs
nu_grid_rest
nu_grid_obs
path_to_filters
[str] The path (relative or absolute, should just make it absolute) to the top filter directory.

model_unc

Methods

chain_plot(samples, **kwargs)	Make chain plot.
corner_plot(samples, **kwargs)	Make corner plot.
fit(p0, **kwargs)	Fit the model to the data.
from_json(fname)	Construct a Lightning object with the configuration specified by a json file.
get_mcmc_chains(sampler[, thin, discard, ...])	Reduce the emcee sampler object into chains.
get_model_components_lnu_hires(params[, ...])	Construct the individual components of the high-resolution spectral model.
get_model_likelihood(params[, negative])	Calculate the log-likelihood of the model under the given parameters.
get_model_lines(params[, stepwise])	Construct the absorbed and intrinsic model lines.
get_model_lnu(params[, stepwise])	Construct the low-resolution SED model.
get_model_lnu_hires(params[, stepwise])	Construct the high-resolution spectral model.
get_model_log_prob(params[, priors, ...])	Calculate the log-probability of the model under the given parameters.
get_xray_model_counts(params)	Construct the low-resolution X-ray instrumental SED.
get_xray_model_lnu(params)	Construct the low-resolution X-ray SED model.
get_xray_model_lnu_hires(params)	Construct the high-resolution X-ray spectral model.
print_params([verbose])	Print all the parameters of the current model.
save_json(fname)	Save the information needed to reconstruct this Lightning object to a json file.
save_pickle(fname)	Save this whole Lightning object to a pickle.
sed_plot_bestfit(samples, logprob_samples, ...)	Make plot of the best-fitting SED.
sed_plot_delchi(samples, logprob_samples, ...)	Make residual plot.
sfh_plot(samples, **kwargs)	Make SFH plot.

__init__(*filter_labels*, *redshift=None*, *lum_dist=None*, *flux_obs=None*, *flux_obs_unc=None*, *wave_grid=(0.1, 1000, 1200)*, *stellar_type='PEGASE'*, *line_labels=None*, *line_flux=None*, *line_flux_unc=None*, *line_index=None*, *nebula_lognH=2.0*, *nebula_dust=False*, *SFH_type='Piecewise-Constant'*, *ages=None*, *atten_type='Modified-Calzetti'*, *dust_emission=False*, *agn_emission=False*, *agn_polar_dust=False*, *xray_mode=None*, *xray_stellar_emission=None*, *xray_agn_emission=None*, *xray_absorption=None*, *xray_wave_grid=(1e-06, 0.1, 200)*, *xray_arf=None*, *xray_exposure=None*, *galactic_NH=0.0*, *lightning_filter_path=None*, *print_setup_time=False*, *model_unc=None*, *model_unc_lines=None*, *cosmology=None*, *uplim_handling='exact'*)

chain_plot(*samples*, ***kwargs*)

Make chain plot.

See lightning.plots.chain_plot

corner_plot(*samples*, ***kwargs*)

Make corner plot.

See lightning.plots.corner_plot

fit(*p0*, ***kwargs*)

Fit the model to the data.

Parameters

p0
[np.ndarray, (Nwalkers, Nparam) or (Nparam,), float32] Initial parameters. In the case of the affine invariant MCMC sampler, this should be a 2D array initializing the entire ensemble.

method
[{'emcee', 'optimize'}] Fitting method.

Returns

emcee ensemble sampler object or scipy.opt.minimize result, depending on `method`.

`property flux_obs`

Observed flux-densities in mJy.

Flux densities are converted to observed-frame luminosity densities.

`property flux_unc`

Uncertainties associated with `flux_obs`.

Flux densities are converted to observed-frame luminosity densities.

`static from_json`(*fname*)

Construct a Lightning object with the configuration specified by a json file.

This was a nice idea when the model was simple enough that it could just be `Lightning(**config)` but now it's kind of a nightmare. Should move to binary only. Or come up with a better scheme for ascii serialization, so that I don't have to update this function every time I change anything about the model.

`get_mcmc_chains`(*sampler*, *thin=None*, *discard=None*, *flat=True*, *Nsamples=1000*, *const_dim=None*, *const_vals=None*)

Reduce the emcee sampler object into chains.

Parameters

sampler
[emcee.EnsembleSampler] Sampler from completed MCMC run, as returned by lightning.fit(method='emcee')

thin
[int] Thin factor for chains. Note that *None* means that the thin factor will be determined from the autocorrelation time; if you instead want no thinning, use thin=1. (Default: None)

discard
[int] Number of trials (i.e. burn-in) to discard from the beginning of MCMC chains. Note that *None* means that the burn-in will be determined from the autocorrelation time; if you instead want no discard, use discard=0. (Default: None)

flat
[bool] If True, collapse the ensemble sampler to a single MCMC chain. Otherwise one chain per walker is retained (Default: True)

Nsamples
[int] Number of posterior samples to retain after discarding/thinning/flattening (Default: 1000)

const_dim
[np.array, boolean] An array of length Nparams showing which model parameters were constant (since emcee doesn't know anything about these dimensions, we must provide

them separately). If None, these dimensions will be excluded from the output (as if there were no constant dimensions).

const_vals
[np.array, float] An array giving the values of the constant model parameters, if any.

Returns

samples
[np.ndarray(Nsamples, Nparam)] Sampled posterior chain(s) for parameters.

logprob_samples
[np.ndarray(Nsamples, Nparam)] Sampled logprob chain(s).

t
[np.ndarray(Nparams,)] Autocorrelation time computed *before* thinning and discarding burn-in. Preserved as a diagnostic, since it sets the scale for thinning and burn-in.

`get_model_components_lnu_hires`(*params*, *stepwise=False*)

Construct the individual components of the high-resolution spectral model.

This function returns a dictionary (or an array, I haven't decided yet) containing the indiviudal model components interpolated to the same wavelength grid.

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

stepwise
[bool] If true, this function returns the spectral model as a function of stellar age.

Returns

hires_models
[dict] Keys are {'stellar_attenuated', 'stellar_unattenuated', 'attenuation', 'dust', 'agn'} where 'dust' and 'agn' are only included if the model includes these components. Each key points to a numpy array containing the high-resolution models.

`get_model_likelihood`(*params*, *negative=True*)

Calculate the log-likelihood of the model under the given parameters.

If `negative` flag is set (on by default), returns the negative log likelihood (i.e. chi2 / 2).

Parameters

params
[np.ndarray(Nmodels, Nparams)] An array of the parameters expected by the model. See `Lightning.print_params()` for details on the current model parameters.

negative
[bool] A flag setting whether the log probability or its opposite is returned (as e.g. when using a minimization method). (Default: `True`)

Returns

log_like
[np.ndarray(Nmodels,)] The log of the likelihood.

`get_model_lines`(*params*, *stepwise=False*)

Construct the absorbed and intrinsic model lines.

Only available if the model *has* lines

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

stepwise
[bool] If true, this function returns the spectral model as a function of stellar age.

Returns

lines_ext
[np.ndarray(Nmodels, Nlines) or np.ndarray(Nmodels, Nlines, Nages)] Model line luminosities in Lsun after multiplication by the galaxy attenuation curve.

lnu_intrinsic
[np.ndarray(Nmodels, Nlines) or np.ndarray(Nmodels, Nlines, Nages)] Intrinsic model line luminosities in Lsun.

`get_model_lnu`(*params*, *stepwise=False*)

Construct the low-resolution SED model.

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

stepwise
[bool] If true, this function returns the spectral model as a function of stellar age.

Returns

lnu_processed
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nmodels, Nwave, Nages)] Model including the effects of ISM dust, convolved with the filters.

lnu_intrinsic
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nmodels, Nwave, Nages)] Model not including the effects of ISM dust, convolved with the filters.

`get_model_lnu_hires`(*params*, *stepwise=False*)

Construct the high-resolution spectral model.

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

stepwise
[bool] If true, this function returns the spectral model as a function of stellar age.

Returns

lnu_processed
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nmodels, Nwave, Nages)] High resolution spectral model including the effects of ISM dust.

lnu_intrinsic
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nmodels, Nwave, Nages)] High resolution spectral model not including the effects of ISM dust.

`get_model_log_prob`(*params*, *priors=None*, *negative=True*, *p_bound=inf*)

Calculate the log-probability of the model under the given parameters.

If `negative` flag is set (on by default), returns the negative log probability (i.e. chi2 / 2 + log(prior)).

Parameters

params
[np.ndarray(Nmodels, Nparams)] An array of the parameters expected by the model. See `Lightning.print_params()` for details on the current model parameters.

priors
[list of Nparams callables] Priors on the parameters.

negative
[bool] A flag setting whether the log probability or its opposite is returned (as e.g. when using a minimization method). (Default: `True`)

p_bound
[float] The magnitude of the log probability for models outside of the parameter space. (Default: `np.inf`)

Returns

log_prob
[np.ndarray(Nmodels,)] The log of the probability, prior * likelihood.

`get_xray_model_counts`(*params*)

Construct the low-resolution X-ray instrumental SED.

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

Returns

counts_absorbed
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nmodels, Nwave, Nages)] Model including the effects of the chosen absorption model, convolved with the filters.

counts_unabsorbed
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nmodels, Nwave, Nages)] Model not including the effects of the chosen absorption model, convolved with the filters.

`get_xray_model_lnu`(*params*)

Construct the low-resolution X-ray SED model.

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters.

For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

Returns

lnu_absorbed
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nmodels, Nwave, Nages)] Model including the effects of the chosen absorption model, convolved with the filters.

lnu_unabsorbed
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nmodels, Nwave, Nages)] Model not including the effects of the chosen absorption model, convolved with the filters.

`get_xray_model_lnu_hires`(*params*)

Construct the high-resolution X-ray spectral model.

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

stepwise
[bool] If true, this function returns the spectral model as a function of stellar age.

Returns

lnu_absorbed: np.ndarray(Nmodels, Nwave) or np.ndarray(Nmodels, Nwave, Nages)
High resolution spectral model including the effects of the chosen absorption model.

lnu_unabsorbed
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nmodels, Nwave, Nages)] High resolution spectral model not including the effects of the chosen absorption model.

`property line_flux`

Observed line fluxes in erg cm-2 s-1.

Fluxes are converted to apparent luminosities.

`property line_flux_unc`

Observed line fluxes in erg cm-2 s-1.

Fluxes are converted to apparent luminosities.

`print_params`(*verbose=False*)

Print all the parameters of the current model.

If `verbose`, print a nicely formatted table of the models, their parameters, and the description of the parameters. Otherwise, just print the names of the parameters.

`save_json`(*fname*)

Save the information needed to reconstruct this Lightning object to a json file.

The Lightning object can be remade using `Lightning.from_json`.

The json configuration file is in theory human readable but note that the wavelength grids are reproduced in their entirety, so the file will likely be thousands of lines long.

`save_pickle`(*fname*)

Save this whole Lightning object to a pickle.

This will be larger than just saving the configuration to a json file, since it contains the whole object, all of the models, etc.

The normal caveats with pickles apply.

sed_plot_bestfit(*samples*, *logprob_samples*, ***kwargs*)

Make plot of the best-fitting SED.

See lightning.plots.sed_plot_bestfit

sed_plot_delchi(*samples*, *logprob_samples*, ***kwargs*)

Make residual plot.

See lightning.plots.sed_plot_delchi

sfh_plot(*samples*, ***kwargs*)

Make SFH plot.

See lightning.plots.sfh_plot

8.2 Attenuation Curves

The default attenuation model in lightning is the "modified Calzetti" curve from Noll+(2009) with a 2175 Angstrom bump and a variable UV slope linked to the bump strength. This model is implemented in the `ModifiedCalzettiAtten` class.

class lightning.attenuation.**ModifiedCalzettiAtten**

Bases: *AnalyticAtten*

The Noll+(2009) modification of the Calzetti+(2000) attenuation curve, including a Drude-profile bump at 2175 Å and a variable UV slope.

Linearly extrapolated from 1200 Å down to 912 Å.

Parameters

wave
[np.ndarray, (Nwave,), float] Rest frame wavelength grid to evaluate the model on.

Methods

evaluate(params)	Evaluate the attenuation as a function of wavelength for the given parameters.
get_AV(params)	Helper function to convert tauV -> AV
print_params([verbose])	If *verbose*, print a nicely formatted table of the models, their parameters, and the description of the parameters.

__init__(*wave*)

evaluate(*params*)

Evaluate the attenuation as a function of wavelength for the given parameters.

Model includes a featureless Calzetti law, with the addition of a UV bump at 2175 A and optionally extra birth cloud extinction. The same attenuation model used in most cases by Lightning; I ported it from the Lightning IDL source.

As of right now the birth cloud component should be ignored, it isn't really implemented properly at the moment – it'll be applied to all ages if you set tauV_BC > 0.

If I were willing to be a little more clever, I would define this more obviously as an extension of the Calzetti-Atten class.

Parameters

params
[np.ndarray, (Nmodels, 3) or (3,)] Parameters of the model.

Returns

expminustau
[(Nmodels, Nwave)]

get_AV(*params*)

Helper function to convert tauV -> AV

For toy models and instances where there is insufficient data to constrain the variable UV slope of the `ModifiedCalzettiAtten` model, we also implement the pure Calzetti featureless attenuation curve in the `CalzettiAtten` class.

class lightning.attenuation.**CalzettiAtten**

Bases: *AnalyticAtten*

Featureless Calzetti+(2000) attenuation curve.

Linearly extrapolated from 1200 Å down to 912 Å.

Parameters

wave
[np.ndarray, (Nwave,), float] Rest frame wavelength grid to evaluate the model on.

Methods

evaluate(params)	Evaluate the attenuation as a function of wavelength for the given parameters.
get_AV(params)	Helper function to convert tauV -> AV
print_params([verbose])	If *verbose*, print a nicely formatted table of the models, their parameters, and the description of the parameters.

__init__(*wave*)

evaluate(*params*)

Evaluate the attenuation as a function of wavelength for the given parameters.

Parameters

params
[np.ndarray, (Nmodels, 1) or (1,)] Values for tauV.

Returns

expminustau
[(Nmodels, Nwave)]

get_AV(*params*)

Helper function to convert tauV -> AV

The SMC extinction curve is also implemented in the SMC class for internal use by the polar dust attenuation model, as in X-Cigale.

class lightning.attenuation.**SMC**

Bases: *TabulatedAtten*

Small Magellanic Cloud extinction curve from Gordon et al. (2003)

Parameters

wave
[np.ndarray, (Nwave,), float] Rest frame wavelength grid to evaluate the model on.

References

- Gordon et al. (2003)
- https://www.stsci.edu/hst/instrumentation/reference-data-for-calibration-and-tools/astronomical-catalogs/interstellar-extinction-curves

Methods

evaluate(params)	Evaluate the attenuation as a function of wavelength for the given parameters.
print_params([verbose])	If *verbose*, print a nicely formatted table of the models, their parameters, and the description of the parameters.

evaluate(*params*)

Evaluate the attenuation as a function of wavelength for the given parameters.

Parameters

params
[np.ndarray, (Nmodels, 1) or (1,)] Values for tauV.

Returns

expminustau
[(Nmodels, Nwave)]

The above attenuation and extinction models extend the AnalyticAtten and TabulatedAtten classes.

class lightning.attenuation.base.**AnalyticAtten**

Base class for analytic (i.e., not tabulated) attenuation curves.

__init__(*wave*)

__new__(**args*, ***kwargs*)

class lightning.attenuation.base.**TabulatedAtten**

Base class for tabulated attenuation curves.

__init__(*wave=None*, *path_to_models=None*)

__new__(**args*, ***kwargs*)

8.3 SFH Models

SFH modeling in `Lightning` is biased toward the use of the "non-parametric" `PiecewiseConstSFH` as in every previous `Lightning` publication.

class lightning.sfh.**PiecewiseConstSFH**

Bases: `object`

Class for piecewise-constant star formation histories.

$$\psi(t) = \psi_i,\ t_i \leq t < t_{i+1}$$

Parameters

age
[array-like, (Nbins+1)] This array should define the edges of the stellar age bins.

Methods

evaluate(params)	Return the SFR as a function of time.
multiply(params, arr)	Multiply the SFR in each bin by a supplied array (to determine, e.g., the stellar mass in each bin).
sum(params, arr[, cumulative])	Multiply the SFR in each bin by a supplied array and sum along the age axis.

__init__(*age*)

__new__(**args*, ***kwargs*)

evaluate(*params*)

Return the SFR as a function of time.

For this piecewise constant SFH, it's just a pass-through for *params* after checking that it's the right shape.

Parameters

params
[array-like (Nmodels, Nbins)] SFR in each bin.

Returns

sfrt

multiply(*params*, *arr*)

Multiply the SFR in each bin by a supplied array (to determine, e.g., the stellar mass in each bin).

For normal internal use this function determines the appropriate broadcast shape of the output based on the shape of `arr` (which could be, e.g. stellar mass as a function of time, or luminosity density as a function of time *and* wavelength). If you find yourself using this function on its own, check the shape of the outputs carefully.

Parameters

params
[array-like (Nmodels, Nbins)] SFR in each bin.

arr
[array-like, (..., Nbins, ...) (up to three dimensions)] Array to multiply by the SFH.

Returns

res
[array-like] Product of sfh and `arr` broadcast to whatever shape was deemed appropriate.

sum(*params*, *arr*, *cumulative=False*)

Multiply the SFR in each bin by a supplied array and sum along the age axis.

For normal internal use this function determines the appropriate broadcast shape of the output based on the shape of `arr` (which could be, e.g. stellar mass as a function of time, or luminosity density as a function of time *and* wavelength). If you find yourself using this function on its own, check the shape of the outputs carefully.

Parameters

params
[array-like (Nmodels, Nbins)] SFR in each bin.

arr
[array-like, (..., Nbins, ...) (up to three dimensions)] Array to multiply by the SFH.

cumulative
[bool] If `True`, return the cumulative sum as a function of age.

Returns

res
[array-like, (Nmodels, ...)] Product of sfh and `arr`, summed along the age axis.

Several parametric SFH models are also included in the code for implementing toy models, simulating populations, reproducing literature results, etc.

class lightning.sfh.**DelayedExponentialSFH**

Bases: *FunctionalSFH*

Delayed exponential burst of star formation

$$\psi(t) = A(t/\tau)\exp(-t/\tau)$$

Methods

evaluate(params)	Returns the SFR as a function of time.
integrate(params, arr[, cumulative])	Multiply the SFR in each bin by a supplied array and integrate along the age axis.
multiply(params, arr)	Multiply the SFR in each bin by a supplied array (to determine, e.g., the stellar mass in each bin).

__init__(*age*)

__new__(**args*, ***kwargs*)

`evaluate`(*params*)

Returns the SFR as a function of time.

`integrate`(*params*, *arr*, *cumulative=False*)

Multiply the SFR in each bin by a supplied array and integrate along the age axis.

For normal internal use this function determines the appropriate broadcast shape of the output based on the shape of `arr` (which could be, e.g. stellar mass as a function of time, or luminosity density as a function of time *and* wavelength). If you find yourself using this function on its own, check the shape of the outputs carefully.

Parameters

params
[array-like (Nmodels, Nbins)] SFR in each bin.

arr
[array-like, (..., Nbins, ...) (up to three dimensions)] Array to multiply by the SFH.

cumulative
[bool] If `True`, return the cumulative intgral as a function of age.

Returns

res
[array-like, (Nmodels, ...)] Product of sfh and `arr`, integrated along the age axis.

`multiply`(*params*, *arr*)

Multiply the SFR in each bin by a supplied array (to determine, e.g., the stellar mass in each bin).

For normal internal use this function determines the appropriate broadcast shape of the output based on the shape of `arr` (which could be, e.g. stellar mass as a function of time, or luminosity density as a function of time *and* wavelength). If you find yourself using this function on its own, check the shape of the outputs carefully.

Parameters

params
[array-like (Nmodels, Nparams)] Model parameters.

arr
[array-like, (..., Nbins, ...) (up to three dimensions)] Array to multiply by the SFH.

Returns

res
[array-like] Product of sfh and `arr` broadcast to whatever shape was deemed appropriate.

`class lightning.sfh.SingleExponentialSFH`

Bases: *FunctionalSFH*

Exponentially decaying burst of star formation

$$\psi(t) = \begin{cases} A \exp[(t - t_{burst})/\tau]/k, & t \leq t_{burst} \\ 0, & t > t_{burst} \end{cases}$$

where

$$k = \tau * [1 - \exp(-t_{burst}/\tau)]$$

Methods

evaluate(params)	Returns the SFR as a function of time.
integrate(params, arr[, cumulative])	Multiply the SFR in each bin by a supplied array and integrate along the age axis.
multiply(params, arr)	Multiply the SFR in each bin by a supplied array (to determine, e.g., the stellar mass in each bin).

__init__(*age*)

__new__(**args*, ***kwargs*)

evaluate(*params*)

Returns the SFR as a function of time.

integrate(*params*, *arr*, *cumulative=False*)

Multiply the SFR in each bin by a supplied array and integrate along the age axis.

For normal internal use this function determines the appropriate broadcast shape of the output based on the shape of `arr` (which could be, e.g. stellar mass as a function of time, or luminosity density as a function of time *and* wavelength). If you find yourself using this function on its own, check the shape of the outputs carefully.

Parameters

params
[array-like (Nmodels, Nbins)] SFR in each bin.

arr
[array-like, (..., Nbins, ...) (up to three dimensions)] Array to multiply by the SFH.

cumulative
[bool] If `True`, return the cumulative intgral as a function of age.

Returns

res
[array-like, (Nmodels, ...)] Product of sfh and `arr`, integrated along the age axis.

multiply(*params*, *arr*)

Multiply the SFR in each bin by a supplied array (to determine, e.g., the stellar mass in each bin).

For normal internal use this function determines the appropriate broadcast shape of the output based on the shape of `arr` (which could be, e.g. stellar mass as a function of time, or luminosity density as a function of time *and* wavelength). If you find yourself using this function on its own, check the shape of the outputs carefully.

Parameters

params
[array-like (Nmodels, Nparams)] Model parameters.

arr
[array-like, (..., Nbins, ...) (up to three dimensions)] Array to multiply by the SFH.

Returns

res
[array-like] Product of sfh and `arr` broadcast to whatever shape was deemed appropriate.

New user-specific parametric SFHs can be added by defining classes that extend the `FunctionalSFH` class and define the `evaluate` method.

class lightning.sfh.base.**FunctionalSFH**

Base class for functional (delayed exponential, double exponential, etc.) star formation histories.

Parameters

age
[array-like] Grid of stellar ages on which to evaluate the SFH.

Methods

evaluate(params)	Return the SFR as a function of time.
integrate(params, arr[, cumulative])	Multiply the SFR in each bin by a supplied array and integrate along the age axis.
multiply(params, arr)	Multiply the SFR in each bin by a supplied array (to determine, e.g., the stellar mass in each bin).

__init__(*age*)

__new__(**args*, ***kwargs*)

evaluate(*params*)

Return the SFR as a function of time.

Parameters

params
[array-like (Nmodels, Nparams)] Model parameters.

Returns

sfrt
[array-like (Nmodels, Nages)]

integrate(*params*, *arr*, *cumulative=False*)

Multiply the SFR in each bin by a supplied array and integrate along the age axis.

For normal internal use this function determines the appropriate broadcast shape of the output based on the shape of `arr` (which could be, e.g. stellar mass as a function of time, or luminosity density as a function of time *and* wavelength). If you find yourself using this function on its own, check the shape of the outputs carefully.

Parameters

params
[array-like (Nmodels, Nbins)] SFR in each bin.

arr
[array-like, (..., Nbins, ...) (up to three dimensions)] Array to multiply by the SFH.

cumulative
[bool] If `True`, return the cumulative intgral as a function of age.

Returns

res
[array-like, (Nmodels, ...)] Product of sfh and `arr`, integrated along the age axis.

`multiply`(*params*, *arr*)

Multiply the SFR in each bin by a supplied array (to determine, e.g., the stellar mass in each bin).

For normal internal use this function determines the appropriate broadcast shape of the output based on the shape of `arr` (which could be, e.g. stellar mass as a function of time, or luminosity density as a function of time *and* wavelength). If you find yourself using this function on its own, check the shape of the outputs carefully.

Parameters

params
[array-like (Nmodels, Nparams)] Model parameters.

arr
[array-like, (..., Nbins, ...) (up to three dimensions)] Array to multiply by the SFH.

Returns

res
[array-like] Product of sfh and `arr` broadcast to whatever shape was deemed appropriate.

8.4 Stellar Model

Available stellar models in Lightning are based on the PEGASE and BPASS spectral population synthesis codes; models with the A24 suffix include a nebular component generated from custom Cloudy simulations run by Amirnezam Amiri in 2024 (hence, A24). The `PEGASEModel` class is the same set of stellar population models used in IDL `Lightning`, with a Kroupa IMF and Z = 0.001-0.1

`class lightning.stellar.PEGASEModel`

Bases: `BaseEmissionModel`

Stellar emission models generated using Pégase.

These models are either:

- A single burst of star formation at 1 solar mass yr-1, evaluated on a grid of specified ages
- A binned stellar population, representing a constant epoch of star formation in a specified set of stellar age bins. These models are integrated from the above.

Nebular extinction + continuum are included by default, lines are optional, added by hand.

Parameters

filter_labels
[list, str] List of filter labels.

redshift
[float] Redshift of the model.

age
[np.ndarray, (Nsteps + 1,) or (Nages,), float] Either the bounds of `Nsteps` age bins for a piecewise-constant SFH model, or `Nages` stellar ages for a continuous SFH model.

step
[bool] If `True`, `age` is interpreted as age bounds for a piecewise-constant SFH model. Otherwise `age` is interpreted as the age grid for a continuous SFH.

add_lines
[bool] If `True`, emission lines are added to the spectral models at ages < `1e7` yr.

wave_grid
[np.ndarray, (Nwave,), float, optional] If set, the spectra are interpreted to this wavelength grid.

Attributes

filter_labels
redshift
age
step
metallicity
mstar
[np.ndarray, (Nages,), float] Surviving stellar mass as a function of age.

Lbol
[np.ndarray, (Nages,), float] Bolometric luminosity as a function of age.

q0
[np.ndarray, (Nages,), float] Lyman-continuum photon production rate (yr-1) as a function of age.

wave_grid_rest
[np.ndarray, (Nwave,), float] Rest-frame wavelength grid.

wave_grid_obs
nu_grid_rest
nu_grid_obs
Lnu_obs
[np.ndarray, (Nages, Nwave), float] `(1 + redshift)` times the rest-frame spectral model, as a function of age.

Methods

get_model_lines(sfh, sfh_param, params[, ...])	Get the integrated luminosity of all the lines available to the nebular model.
get_model_lnu(sfh, sfh_param[, params, ...])	Construct the stellar SED as observed in the given filters.
get_model_lnu_hires(sfh, sfh_param, params)	Construct the high-res stellar spectrum.
get_mstar_coeff(Z)	Return the Mstar coefficients as a function of age for a given metallicity.
print_params([verbose])	If *verbose*, print a nicely formatted table of the models, their parameters, and the description of the parameters.

get_model_lines(*sfh*, *sfh_param*, *params*, *stepwise=False*)

Get the integrated luminosity of all the lines available to the nebular model.

See self.line_names for a full list of lines. In the future we'll need to redden these lines to compare them to the observed lines, such that our line attenuation is consistent with the attenuation of the stellar population model broadly.

Parameters

sfh
[instance of lightning.sfh.PiecewiseConstSFH or lightning.sfh.FunctionalSFH] Star formation history model.

sfh_params
[np.ndarray, (Nmodels, Nparam) or (Nparam,), float32] Parameters for the star formation history.

params
[np.ndarray, (Nmodels, 1) or (Nmodels,)] Values for Z.

stepwise
[bool] If true, the lines are returned as a function of stellar age.

Returns

Lmod_lines
[np.ndarray, (Nmodels, Nlines) or (Nmodels, Nages, Nlines)] Integrated line luminosities, optionally as a function of age.

`get_model_lnu`(*sfh*, *sfh_param*, *params=None*, *exptau=None*, *exptau_youngest=None*, *stepwise=False*)

Construct the stellar SED as observed in the given filters.

Given a SFH instance and set of parameters, the corresponding high-resolution spectrum is constructed and convolved with the filters. Optionally, attenuation is applied and the attenuated power is returned.

Parameters

sfh
[instance of lightning.sfh.PiecewiseConstSFH or lightning.sfh.FunctionalSFH] Star formation history model.

sfh_params
[np.ndarray, (Nmodels, Nparam) or (Nparam,), float32] Parameters for the star formation history.

params
[None] Empty placeholder for compatibility.

exptau
[np.ndarray, (Nmodels, Nwave) or (Nwave,), float32] `exp(-tau)` as a function of wavelength. If this is 2D, the size of the first dimension must match the size of the first dimension of `sfh_params`.

exptau_youngest
[np.ndarray, (Nmodels, Nwave) or (Nwave,), float32] This doesn't do anything at the moment, until I figure out how to flexibly decide which ages to apply the birth cloud attenuation to.

stepwise
[bool] If true, the spectrum is returned as a function of stellar age.

Returns

lnu_attenuated
[np.ndarray, (Nmodels, Nfilters), (Nmodels, Nages, Nfilters), or (Nfilters,), float32] The stellar spectrum as seen after the application of the ISM dust attenuation model.

lnu_unattenuated
[np.ndarray, (Nmodels, Nfilters), (Nmodels, Nages, Nfilters), or (Nfilters,), float32] The intrinsic stellar spectrum.

L_TIR
[np.ndarray, (Nmodels,) or (Nmodels, Nages)] The total attenuated power of the stellar population.

`get_model_lnu_hires`(*sfh*, *sfh_param*, *params*, *exptau=None*, *exptau_youngest=None*, *stepwise=False*)

Construct the high-res stellar spectrum.

Given a SFH instance and set of parameters, the corresponding high-resolution spectrum is constructed. Optionally, attenuation is applied and the attenuated power is returned.

Parameters

sfh
[instance of lightning.sfh.PiecewiseConstSFH or lightning.sfh.FunctionalSFH] Star formation history model.

sfh_params
[np.ndarray, (Nmodels, Nparam) or (Nparam,), float32] Parameters for the star formation history.

params
[None] Empty placeholder for compatibility.

exptau
[np.ndarray, (Nmodels, Nwave) or (Nwave,), float32] `exp(-tau)` as a function of wavelength. If this is 2D, the size of the first dimension must match the size of the first dimension of `sfh_params`.

exptau_youngest
[np.ndarray, (Nmodels, Nwave) or (Nwave,), float32] This doesn't do anything at the moment, until I figure out how to flexibly decide which ages to apply the birth cloud attenuation to.

stepwise
[bool] If true, the spectrum is returned as a function of stellar age.

Returns

lnu_attenuated
[np.ndarray, (Nmodels, Nwave), (Nmodels, Nages, Nwave), or (Nwave,), float32] The stellar spectrum as seen after the application of the ISM dust attenuation model.

lnu_unattenuated
[np.ndarray, (Nmodels, Nwave), (Nmodels, Nages, Nwave), or (Nwave,), float32] The intrinsic stellar spectrum.

L_TIR
[np.ndarray, (Nmodels,) or (Nmodels, Nages)] The total attenuated power of the stellar population.

`get_mstar_coeff`(*Z*)

Return the Mstar coefficients as a function of age for a given metallicity.

Parameters

Z
[array-like (Nmodels,)] Stellar metallicity

Returns

Mstar
[(Nmodels, Nages)] Surviving stellar mass as function of age per 1 Msun yr-1 of SFR.

`class lightning.stellar.`**PEGASEModelA24**

Bases: `BaseEmissionModel`

Stellar emission models generated using PEGASE, including nebular emission calculated with Cloudy using a custom recipe with an ionization bounded nebula and an open geometry. See the README in the models folder for an outline of the Cloudy setup and links to more detailed documentation.

These models are either:

- A single burst of star formation at 1 solar mass yr-1, evaluated on a grid of specified ages
- A binned stellar population, representing a constant epoch of star formation in a specified set of stellar age bins. These models are integrated from the above.

Nebular extinction, lines, and continuum are included by default.

Parameters

filter_labels
[list, str] List of filter labels.

redshift
[float] Redshift of the model.

age
[np.ndarray, (Nsteps + 1,) or (Nages,), float] Either the bounds of `Nsteps` age bins for a piecewise-constant SFH model, or `Nages` stellar ages for a continuous SFH model.

step
[bool] If `True`, `age` is interpreted as age bounds for a piecewise-constant SFH model. Otherwise `age` is interpreted as the age grid for a continuous SFH.

binaries
[bool] If `True`, the spectra include the effects of binary stellar evolution. If `False`, the nebular model cannot be applied.

nebular_effects
[bool] If `True`, the spectra will include nebular extinction, continua, and lines.

line_labels
[np.ndarray, (Nlines,), string, optional] Line labels in the format used by pyCloudy. See *lightning/models/linelist_full.txt* for the complete list of lines in the grid and their format.

nebula_old
[bool] If `True`, the spectra will include nebular extinction and emission at ages older than 50 Myr, modeling the nebular contributions of the hot, stripped cores of evolved massive stars. While the the conditions we assume for the nebula are most appropriate for massive H II regions, Byler+(2017) found that nebular emission from post-AGB stars is not super sensitive to the geometry/density of the nebula, but it may be worth fitting your galaxies with and without this component if you're concerned.

dust_grains
[bool] If `True`, then dust grains are included in the Cloudy grids. (Default: False)

wave_grid
[np.ndarray, (Nwave,), float, optional] If set, the spectra are interpreted to this wavelength grid.

Attributes

filter_labels
redshift
age
step
metallicity
mstar
[np.ndarray, (Nages,), float] Surviving stellar mass as a function of age.

Lbol
[np.ndarray, (Nages,), float] Bolometric luminosity as a function of age.

q0
[np.ndarray, (Nages,), float] Lyman-continuum photon production rate (yr-1) as a function of age.

wave_grid_rest
[np.ndarray, (Nwave,), float] Rest-frame wavelength grid.

wave_grid_obs
nu_grid_rest
nu_grid_obs
Lnu_obs
[np.ndarray, (Nages, Nwave), float] `(1 + redshift)` times the rest-frame spectral model, as a function of age.

Methods

get_model_lines(sfh, sfh_param, params[, ...])	Get the integrated luminosity of all the lines available to the nebular model.
get_model_lnu(sfh, sfh_param[, params, ...])	Construct the stellar SED as observed in the given filters.
get_model_lnu_hires(sfh, sfh_param, params)	Construct the high-res stellar spectrum.
get_mstar_coeff(Z)	Return the Mstar coefficients as a function of age for a given metallicity.
print_params([verbose])	If *verbose*, print a nicely formatted table of the models, their parameters, and the description of the parameters.

get_model_lines(*sfh*, *sfh_param*, *params*, *exptau=None*, *stepwise=False*)

Get the integrated luminosity of all the lines available to the nebular model.

See self.line_names for a full list of lines. In the future we'll need to redden these lines to compare them to the observed lines, such that our line attenuation is consistent with the attenuation of the stellar population model broadly.

Parameters

sfh
[instance of lightning.sfh.PiecewiseConstSFH or lightning.sfh.FunctionalSFH] Star formation history model.

sfh_params
[np.ndarray, (Nmodels, Nparam) or (Nparam,), float32] Parameters for the star formation history.

params
[np.ndarray, (Nmodels, 2)] Values for Z and logU.

stepwise
[bool] If true, the lines are returned as a function of stellar age.

Returns

Lmod_lines
[np.ndarray, (Nmodels, Nlines) or (Nmodels, Nages, Nlines)] Integrated line luminosities, optionally as a function of age.

`get_model_lnu`(*sfh*, *sfh_param*, *params=None*, *exptau=None*, *exptau_youngest=None*, *stepwise=False*)

Construct the stellar SED as observed in the given filters.

Given a SFH instance and set of parameters, the corresponding high-resolution spectrum is constructed and convolved with the filters. Optionally, attenuation is applied and the attenuated power is returned.

Parameters

sfh
[instance of lightning.sfh.PiecewiseConstSFH or lightning.sfh.FunctionalSFH] Star formation history model.

sfh_params
[np.ndarray, (Nmodels, Nparam) or (Nparam,), float32] Parameters for the star formation history.

params
[np.ndarray, (Nmodels, 1) or (Nmodels,)] Values for logU, if the model includes a nebular component.

exptau
[np.ndarray, (Nmodels, Nwave) or (Nwave,), float32] `exp(-tau)` as a function of wavelength. If this is 2D, the size of the first dimension must match the size of the first dimension of `sfh_params`.

exptau_youngest
[np.ndarray, (Nmodels, Nwave) or (Nwave,), float32] This doesn't do anything at the moment, until I figure out how to flexibly decide which ages to apply the birth cloud attenuation to.

stepwise
[bool] If true, the spectrum is returned as a function of stellar age.

Returns

lnu_attenuated
[np.ndarray, (Nmodels, Nfilters), (Nmodels, Nages, Nfilters), or (Nfilters,), float32] The stellar spectrum as seen after the application of the ISM dust attenuation model.

lnu_unattenuated
[np.ndarray, (Nmodels, Nfilters), (Nmodels, Nages, Nfilters), or (Nfilters,), float32] The intrinsic stellar spectrum.

L_TIR
[np.ndarray, (Nmodels,) or (Nmodels, Nages)] The total attenuated power of the stellar population.

`get_model_lnu_hires`(*sfh*, *sfh_param*, *params*, *exptau=None*, *exptau_youngest=None*, *stepwise=False*)

Construct the high-res stellar spectrum.

Given a SFH instance and set of parameters, the corresponding high-resolution spectrum is constructed. Optionally, attenuation is applied and the attenuated power is returned.

Parameters

sfh
[instance of lightning.sfh.PiecewiseConstSFH or lightning.sfh.FunctionalSFH] Star formation history model.

sfh_params
[np.ndarray, (Nmodels, Nparam) or (Nparam,), float32] Parameters for the star formation history.

params
[np.ndarray, (Nmodels, 1) or (Nmodels,)] Values for logU, if the model includes a nebular component.

exptau
[np.ndarray, (Nmodels, Nwave) or (Nwave,), float32] `exp(-tau)` as a function of wavelength. If this is 2D, the size of the first dimension must match the size of the first dimension of `sfh_params`.

exptau_youngest
[np.ndarray, (Nmodels, Nwave) or (Nwave,), float32] This doesn't do anything at the moment, until I figure out how to flexibly decide which ages to apply the birth cloud attenuation to.

stepwise
[bool] If true, the spectrum is returned as a function of stellar age.

Returns

lnu_attenuated
[np.ndarray, (Nmodels, Nwave), (Nmodels, Nages, Nwave), or (Nwave,), float32] The stellar spectrum as seen after the application of the ISM dust attenuation model.

lnu_unattenuated
[np.ndarray, (Nmodels, Nwave), (Nmodels, Nages, Nwave), or (Nwave,), float32] The intrinsic stellar spectrum.

L_TIR
[np.ndarray, (Nmodels,) or (Nmodels, Nages)] The total attenuated power of the stellar population.

`get_mstar_coeff`(*Z*)

Return the Mstar coefficients as a function of age for a given metallicity.

Parameters

Z
[array-like (Nmodels,)] Stellar metallicity

Returns

Mstar
[(Nmodels, Nages)] Surviving stellar mass as function of age per 1 Msun yr-1 of SFR.

`class lightning.stellar.`**PEGASEBurstA24**

Bases: *PEGASEModelA24*

SFH-free model representing a single instantaneous burst of star formation with a given mass and age.

Methods

get_model_lines(params[, exptau])	
get_model_lnu(params[, exptau])	
get_model_lnu_hires(params[, exptau])	
get_mstar_coeff(Z)	Return the Mstar coefficients as a function of age for a given metallicity.
print_params([verbose])	If *verbose*, print a nicely formatted table of the models, their parameters, and the description of the parameters.

__init__(*filter_labels*, *redshift*, *wave_grid=None*, *age=None*, *lognH=2.0*, *cosmology=None*, *line_labels=None*, *dust_grains=False*)

Generic initialization. Actual model-building should be handled by implementing the *construct_model* and *construct_model_grid* methods.

get_model_lines(*params*, *exptau=None*)

Parameters

params
[array-like (Nmodels, 4)] The parameters are, in order, Mburst, tburst, Z, and logU. In practice we'll probably sample log(Mburst) and log(tburst).

exptau
[array-like (Nmodels, Nwave)] Attenuation curve(s) evaluated at model wavelengths; really it's exp(-tau). Currently unused, i.e. line luminosities returned are intrinsic, unreddened.

Returns

lmod_lines

get_model_lnu(*params*, *exptau=None*)

Parameters

params
[array-like (Nmodels, 4)] The parameters are, in order, Mburst, tburst, Z, and logU. In practice we'll probably sample log(Mburst) and log(tburst).

exptau
[array-like (Nmodels, Nwave)] Attenuation curve(s) evaluated at model wavelengths; really it's exp(-tau)

Returns

lmod_unattenuated
lmod_unattenuated
L_TIR

get_model_lnu_hires(*params*, *exptau=None*)

Parameters

params
[array-like (Nmodels, 4)] The parameters are, in order, Mburst, tburst, Z, and logU. In practice we'll probably sample log(Mburst) and log(tburst).

exptau
[array-like (Nmodels, Nwave)] Attenuation curve(s) evaluated at model wavelengths; really it's exp(-tau)

Returns

lnu_unattenuated
lnu_unattenuated
L_TIR

class lightning.stellar.**BPASSModel**

Bases: `BaseEmissionModel`

Stellar emission models generated using BPASS, including nebular emission calculated with Cloudy, as generated by the BPASS team. See the README in the models folder for an outline of the Cloudy setup and links to more detailed documentation. For the custom BPASS+Cloudy models used in e.g. Lehmer+(2024), see `BPASSModelA24`.

These models are either:

- A single burst of star formation at 1 solar mass yr-1, evaluated on a grid of specified ages
- A binned stellar population, representing a constant epoch of star formation in a specified set of stellar age bins. These models are integrated from the above.

Nebular extinction, lines, and continuum are included by default. IMF is only 'imf_135_300'; a Kroupa-like two-slope IMF with an upper mass cutoff at 300 Msun.

Parameters

filter_labels
[list, str] List of filter labels.

redshift
[float] Redshift of the model.

age
[np.ndarray, (Nsteps + 1,) or (Nages,), float] Either the bounds of `Nsteps` age bins for a piecewise-constant SFH model, or `Nages` stellar ages for a continuous SFH model.

step
[bool] If `True`, `age` is interpreted as age bounds for a piecewise-constant SFH model. Otherwise `age` is interpreted as the age grid for a continuous SFH.

binaries
[bool] If `True`, the spectra include the effects of binary stellar evolution. If `False`, the nebular model cannot be applied (and as a result the `dust_grains` switch has no effect).

nebular_effects
[bool] If `True`, the spectra at ages <1e7.5 years will include nebular extinction, continua, and lines.

dust_grains
[bool] If `True`, the Cloudy models include the effects of dust grain depletion. This option has no effect unless `nebular_effects` is True. By default, we set this option to `False`, for parity with the treatment of nebular emission in the Pégase stellar population models.

wave_grid
[np.ndarray, (Nwave,), float, optional] If set, the spectra are interpreted to this wavelength grid.

Attributes

filter_labels
redshift
age
step
metallicity
mstar
[np.ndarray, (Nages,), float] Surviving stellar mass as a function of age.

Lbol
[np.ndarray, (Nages,), float] Bolometric luminosity as a function of age.

q0
[np.ndarray, (Nages,), float] Lyman-continuum photon production rate (yr-1) as a function of age.

wave_grid_rest
[np.ndarray, (Nwave,), float] Rest-frame wavelength grid.

wave_grid_obs
nu_grid_rest
nu_grid_obs
Lnu_obs
[np.ndarray, (Nages, Nwave), float] `(1 + redshift)` times the rest-frame spectral model, as a function of age.

Methods

get_model_lines(sfh, sfh_param, params[, ...])	Get the integrated luminosity of all the lines available to the nebular model.
get_model_lnu(sfh, sfh_param, params[, ...])	Construct the stellar SED as observed in the given filters.
get_model_lnu_hires(sfh, sfh_param, params)	Construct the high-res stellar spectrum.
get_mstar_coeff(Z)	Return the Mstar coefficients as a function of age for a given metallicity.
print_params([verbose])	If *verbose*, print a nicely formatted table of the models, their parameters, and the description of the parameters.

get_model_lines(*sfh*, *sfh_param*, *params*, *stepwise=False*)

Get the integrated luminosity of all the lines available to the nebular model.

See self.line_names for a full list of lines. In the future we'll need to redden these lines to compare them to the observed lines, such that our line attenuation is consistent with the attenuation of the stellar population model broadly.

Parameters

sfh
[instance of lightning.sfh.PiecewiseConstSFH or lightning.sfh.FunctionalSFH] Star formation history model.

sfh_params
[np.ndarray, (Nmodels, Nparam) or (Nparam,), float32] Parameters for the star formation history.

params
[np.ndarray, (Nmodels, 2)] Values for Z and logU.

stepwise
[bool] If true, the lines are returned as a function of stellar age.

Returns

Lmod_lines
[np.ndarray, (Nmodels, Nlines) or (Nmodels, Nages, Nlines)] Integrated line luminosities, optionally as a function of age.

`get_model_lnu`(*sfh*, *sfh_param*, *params*, *exptau=None*, *exptau_youngest=None*, *stepwise=False*)

Construct the stellar SED as observed in the given filters.

Given a SFH instance and set of parameters, the corresponding high-resolution spectrum is constructed and convolved with the filters. Optionally, attenuation is applied and the attenuated power is returned.

Parameters

sfh
[instance of lightning.sfh.PiecewiseConstSFH or lightning.sfh.FunctionalSFH] Star formation history model.

sfh_params
[np.ndarray, (Nmodels, Nparam) or (Nparam,), float32] Parameters for the star formation history.

params
[np.ndarray, (Nmodels, 1) or (Nmodels,)] Values for logU, if the model includes a nebular component.

exptau
[np.ndarray, (Nmodels, Nwave) or (Nwave,), float32] `exp(-tau)` as a function of wavelength. If this is 2D, the size of the first dimension must match the size of the first dimension of `sfh_params`.

exptau_youngest
[np.ndarray, (Nmodels, Nwave) or (Nwave,), float32] This doesn't do anything at the moment, until I figure out how to flexibly decide which ages to apply the birth cloud attenuation to.

stepwise
[bool] If true, the spectrum is returned as a function of stellar age.

Returns

lnu_attenuated
[np.ndarray, (Nmodels, Nfilters), (Nmodels, Nages, Nfilters), or (Nfilters,), float32] The stellar spectrum as seen after the application of the ISM dust attenuation model.

lnu_unattenuated
[np.ndarray, (Nmodels, Nfilters), (Nmodels, Nages, Nfilters), or (Nfilters,), float32] The intrinsic stellar spectrum.

L_TIR
[np.ndarray, (Nmodels,) or (Nmodels, Nages)] The total attenuated power of the stellar population.

`get_model_lnu_hires`(*sfh*, *sfh_param*, *params*, *exptau=None*, *exptau_youngest=None*, *stepwise=False*)

Construct the high-res stellar spectrum.

Given a SFH instance and set of parameters, the corresponding high-resolution spectrum is constructed. Optionally, attenuation is applied and the attenuated power is returned.

Parameters

sfh
[instance of lightning.sfh.PiecewiseConstSFH or lightning.sfh.FunctionalSFH] Star formation history model.

sfh_params
[np.ndarray, (Nmodels, Nparam) or (Nparam,), float32] Parameters for the star formation history.

params
[np.ndarray, (Nmodels, 1) or (Nmodels,)] Values for logU, if the model includes a nebular component.

exptau
[np.ndarray, (Nmodels, Nwave) or (Nwave,), float32] `exp(-tau)` as a function of wavelength. If this is 2D, the size of the first dimension must match the size of the first dimension of `sfh_params`.

exptau_youngest
[np.ndarray, (Nmodels, Nwave) or (Nwave,), float32] This doesn't do anything at the moment, until I figure out how to flexibly decide which ages to apply the birth cloud attenuation to.

stepwise
[bool] If true, the spectrum is returned as a function of stellar age.

Returns

lnu_attenuated
[np.ndarray, (Nmodels, Nwave), (Nmodels, Nages, Nwave), or (Nwave,), float32] The stellar spectrum as seen after the application of the ISM dust attenuation model.

lnu_unattenuated
[np.ndarray, (Nmodels, Nwave), (Nmodels, Nages, Nwave), or (Nwave,), float32] The intrinsic stellar spectrum.

L_TIR
[np.ndarray, (Nmodels,) or (Nmodels, Nages)] The total attenuated power of the stellar population.

`get_mstar_coeff`(*Z*)

Return the Mstar coefficients as a function of age for a given metallicity.

Parameters

Z
[array-like (Nmodels,)] Stellar metallicity

Returns

Mstar
[(Nmodels, Nages)] Surviving stellar mass as function of age per 1 Msun yr-1 of SFR.

class lightning.stellar.**BPASSModelA24**

Bases: BaseEmissionModel

Stellar emission models generated using BPASS, including nebular emission calculated with Cloudy using a custom recipe with an ionization bounded nebula and an open geometry. See the README in the models folder for an outline of the Cloudy setup and links to more detailed documentation.

These models are either:

- A single burst of star formation at 1 solar mass yr-1, evaluated on a grid of specified ages
- A binned stellar population, representing a constant epoch of star formation in a specified set of stellar age bins. These models are integrated from the above.

Nebular extinction, lines, and continuum are included by default. IMF is Chabrier by default, with high mass cutoff at 300 Msun (chab300). If the grids based on the SSP+ULX SED files from Garofali+(2024) are selected, the IMF is instead 'imf_135_100', a Kroupa-like two-slope IMF with a high mass cutoff at 100 Msun.

Parameters

filter_labels
[list, str] List of filter labels.

redshift
[float] Redshift of the model.

age
[np.ndarray, (Nsteps + 1,) or (Nages,), float] Either the bounds of `Nsteps` age bins for a piecewise-constant SFH model, or `Nages` stellar ages for a continuous SFH model.

step
[bool] If `True`, `age` is interpreted as age bounds for a piecewise-constant SFH model. Otherwise `age` is interpreted as the age grid for a continuous SFH.

binaries
[bool] If `True`, the spectra include the effects of binary stellar evolution. If `False`, the nebular model cannot be applied.

ULX
[bool] If `True`, the spectra loaded will be the SXP+SSP models from Garofali+(2024), postprocessed with Cloudy following the same recipe as we typically adopt (see `lightning_models/models/BPASS_Cloudy/README.md`). Note that this fixes the IMF to the BPASS *imf_135_100* adopted in Garofali+. Note that this model doesn't extend fully to the X-rays yet, it just modifies the line emission due to the considerably larger Q(He II).

nebular_effects
[bool] If `True`, the spectra will include nebular extinction, continua, and lines.

line_labels
[np.ndarray, (Nlines,), string, optional] Line labels in the format used by pyCloudy. See *lightning/models/linelist_full.txt* for the complete list of lines in the grid and their format.

nebula_old
[bool] If `True`, the spectra will include nebular extinction and emission at ages older than 50 Myr, modeling the nebular contributions of the hot, stripped cores of evolved massive stars. While the the conditions we assume for the nebula are most appropriate for massive H II regions, Byler+(2017) found that nebular emission from post-AGB stars is not super sensitive to the geometry/density of the nebula, but it may be worth fitting your galaxies with and without this component if you're concerned.

dust_grains
[bool] If `True`, then dust grains are included in the Cloudy grids. (Default: False)

wave_grid
[np.ndarray, (Nwave,), float, optional] If set, the spectra are interpreted to this wavelength grid.

Attributes

filter_labels
redshift
age
step
metallicity
mstar
[np.ndarray, (Nages,), float] Surviving stellar mass as a function of age.

Lbol
[np.ndarray, (Nages,), float] Bolometric luminosity as a function of age.

q0
[np.ndarray, (Nages,), float] Lyman-continuum photon production rate (yr-1) as a function of age.

wave_grid_rest
[np.ndarray, (Nwave,), float] Rest-frame wavelength grid.

wave_grid_obs
nu_grid_rest
nu_grid_obs
Lnu_obs
[np.ndarray, (Nages, Nwave), float] `(1 + redshift)` times the rest-frame spectral model, as a function of age.

Methods

get_model_lines(sfh, sfh_param, params[, ...])	Get the integrated luminosity of all the lines available to the nebular model.
get_model_lnu(sfh, sfh_param[, params, ...])	Construct the stellar SED as observed in the given filters.
get_model_lnu_hires(sfh, sfh_param, params)	Construct the high-res stellar spectrum.
get_mstar_coeff(Z)	Return the Mstar coefficients as a function of age for a given metallicity.
print_params([verbose])	If *verbose*, print a nicely formatted table of the models, their parameters, and the description of the parameters.

get_model_lines(*sfh*, *sfh_param*, *params*, *exptau=None*, *stepwise=False*)

Get the integrated luminosity of all the lines available to the nebular model.

See self.line_names for a full list of lines. In the future we'll need to redden these lines to compare them to the observed lines, such that our line attenuation is consistent with the attenuation of the stellar population model broadly.

Parameters

sfh
[instance of lightning.sfh.PiecewiseConstSFH or lightning.sfh.FunctionalSFH] Star formation history model.

sfh_params
[np.ndarray, (Nmodels, Nparam) or (Nparam,), float32] Parameters for the star formation history.

params
[np.ndarray, (Nmodels, 1) or (Nmodels,)] Values for Z and logU

stepwise
[bool] If true, the lines are returned as a function of stellar age.

Returns

Lmod_lines
[np.ndarray, (Nmodels, Nlines) or (Nmodels, Nages, Nlines)] Integrated line luminosities, optionally as a function of age.

`get_model_lnu`(*sfh*, *sfh_param*, *params=None*, *exptau=None*, *exptau_youngest=None*, *stepwise=False*)

Construct the stellar SED as observed in the given filters.

Given a SFH instance and set of parameters, the corresponding high-resolution spectrum is constructed and convolved with the filters. Optionally, attenuation is applied and the attenuated power is returned.

Parameters

sfh
[instance of lightning.sfh.PiecewiseConstSFH or lightning.sfh.FunctionalSFH] Star formation history model.

sfh_params
[np.ndarray, (Nmodels, Nparam) or (Nparam,), float32] Parameters for the star formation history.

params
[np.ndarray, (Nmodels, 1) or (Nmodels,)] Values for logU, if the model includes a nebular component.

exptau
[np.ndarray, (Nmodels, Nwave) or (Nwave,), float32] `exp(-tau)` as a function of wavelength. If this is 2D, the size of the first dimension must match the size of the first dimension of `sfh_params`.

exptau_youngest
[np.ndarray, (Nmodels, Nwave) or (Nwave,), float32] This doesn't do anything at the moment, until I figure out how to flexibly decide which ages to apply the birth cloud attenuation to.

stepwise
[bool] If true, the spectrum is returned as a function of stellar age.

Returns

lnu_attenuated
[np.ndarray, (Nmodels, Nfilters), (Nmodels, Nages, Nfilters), or (Nfilters,), float32] The stellar spectrum as seen after the application of the ISM dust attenuation model.

lnu_unattenuated
[np.ndarray, (Nmodels, Nfilters), (Nmodels, Nages, Nfilters), or (Nfilters,), float32] The intrinsic stellar spectrum.

L_TIR
[np.ndarray, (Nmodels,) or (Nmodels, Nages)] The total attenuated power of the stellar population.

`get_model_lnu_hires`(*sfh*, *sfh_param*, *params*, *exptau=None*, *exptau_youngest=None*, *stepwise=False*)

Construct the high-res stellar spectrum.

Given a SFH instance and set of parameters, the corresponding high-resolution spectrum is constructed. Optionally, attenuation is applied and the attenuated power is returned.

Parameters

sfh
[instance of lightning.sfh.PiecewiseConstSFH or lightning.sfh.FunctionalSFH] Star formation history model.

sfh_params
[np.ndarray, (Nmodels, Nparam) or (Nparam,), float32] Parameters for the star formation history.

params
[np.ndarray, (Nmodels, 1) or (Nmodels,)] Values for logU, if the model includes a nebular component.

exptau
[np.ndarray, (Nmodels, Nwave) or (Nwave,), float32] `exp(-tau)` as a function of wavelength. If this is 2D, the size of the first dimension must match the size of the first dimension of `sfh_params`.

exptau_youngest
[np.ndarray, (Nmodels, Nwave) or (Nwave,), float32] This doesn't do anything at the moment, until I figure out how to flexibly decide which ages to apply the birth cloud attenuation to.

stepwise
[bool] If true, the spectrum is returned as a function of stellar age.

Returns

lnu_attenuated
[np.ndarray, (Nmodels, Nwave), (Nmodels, Nages, Nwave), or (Nwave,), float32] The stellar spectrum as seen after the application of the ISM dust attenuation model.

lnu_unattenuated
[np.ndarray, (Nmodels, Nwave), (Nmodels, Nages, Nwave), or (Nwave,), float32] The intrinsic stellar spectrum.

L_TIR
[np.ndarray, (Nmodels,) or (Nmodels, Nages)] The total attenuated power of the stellar population.

`get_mstar_coeff`(*Z*)

Return the Mstar coefficients as a function of age for a given metallicity.

Parameters

Z
[array-like (Nmodels,)] Stellar metallicity

Returns

Mstar
[(Nmodels, Nages)] Surviving stellar mass as function of age per 1 Msun yr-1 of SFR.

class lightning.stellar.**BPASSBurstA24**

Bases: *BPASSModelA24*

SFH-free model representing a single instantaneous burst of star formation with a given mass and age.

Methods

get_model_lines(params[, exptau])	
get_model_lnu(params[, exptau])	
get_model_lnu_hires(params[, exptau])	
get_mstar_coeff(Z)	Return the Mstar coefficients as a function of age for a given metallicity.
print_params([verbose])	If *verbose*, print a nicely formatted table of the models, their parameters, and the description of the parameters.

__init__(*filter_labels*, *redshift*, *wave_grid=None*, *age=None*, *lognH=2.0*, *cosmology=None*, *line_labels=None*, *dust_grains=False*, *ULX=False*)

Generic initialization. Actual model-building should be handled by implementing the *construct_model* and *construct_model_grid* methods.

get_model_lines(*params*, *exptau=None*)

Parameters

params
[array-like (Nmodels, 4)] The parameters are, in order, Mburst, tburst, Z, and logU. In practice we'll probably sample log(Mburst) and log(tburst).

exptau
[array-like (Nmodels, Nwave)] Attenuation curve(s) evaluated at model wavelengths; really it's exp(-tau).

Returns

lmod_lines

get_model_lnu(*params*, *exptau=None*)

Parameters

params
[array-like (Nmodels, 4)] The parameters are, in order, Mburst, tburst, Z, and logU. In practice we'll probably sample log(Mburst) and log(tburst).

exptau
[array-like (Nmodels, Nwave)] Attenuation curve(s) evaluated at model wavelengths; really it's exp(-tau)

Returns

lmod_unattenuated
lmod_unattenuated
L_TIR

`get_model_lnu_hires`(*params*, *exptau=None*)

Parameters

params
[array-like (Nmodels, 4)] The parameters are, in order, Mburst, tburst, Z, and logU. In practice we'll probably sample log(Mburst) and log(tburst).

exptau
[array-like (Nmodels, Nwave)] Attenuation curve(s) evaluated at model wavelengths; really it's exp(-tau)

Returns

lnu_unattenuated
lnu_unattenuated
L_TIR

8.5 Dust Model

Dust emission powered by the stellar radiation field is modeled in lightning using the Draine & Li (2007) models. Setting `dust_emission=True` when initializing `Lightning` automatically selects this model; we do not currently provide an alternative.

`class lightning.dust.DL07Dust`

Bases: `BaseEmissionModel`

An implementation of the Draine & Li (2007) dust emission models.

A fraction `gamma` of the dust is exposed to a radiation field such that the mass of dust `dM` exposed to a range of intensities `[U, U + dU]` is

$$dM = \text{const}\ U^{-\alpha} dU,$$

where `U` is in `[Umin, Umax]`. The remaining portion `(1 - gamma)` is exposed to a radiation field with intensity `Umin`.

Parameters

filter_labels
[list, str] List of filter labels.

redshift
[float] Redshift of the model.

wave_grid
[np.ndarray] Rest frame wavelength grid to interpolate the model on.

Attributes

Lnu_rest
[numpy.ndarray, (29, 7, Nwave), float] High-res spectral model grid. First dimension covers U, the second q_PAH, and the third covers wavelength.

Lnu_obs
[numpy.ndarray, (29, 7, Nwave), float] `(1 + redshift) * Lnu_rest`

mean_Lnu
[numpy.ndarray, (29, 7, Nfilters), float] The `Lnu_obs` grid integrated against the filters.

Lbol
[numpy.ndarray, (29, 7), float] Total luminosity of each model in the grid.

wave_grid_rest
[numpy.ndarray, (Nwave,), float] Rest-frame wavelength grid for the models.

wave_grid_obs
nu_grid_rest
nu_grid_obs

Methods

get_model_lnu(params)	Construct the dust SED as observed in the given filters.
get_model_lnu_hires(params)	Construct the high-resolution dust SED.
print_params([verbose])	If *verbose*, print a nicely formatted table of the models, their parameters, and the description of the parameters.

get_model_lnu(*params*)

Construct the dust SED as observed in the given filters.

Given a set of parameters, the corresponding high-resolution spectrum is constructed and convolved with the filters.

Parameters

params
[np.ndarray, (Nmodels, 5) or (5,) float32] The dust model parameters.

Returns

Lnu_obs
[np.ndarray, (Nmodels, Nfilters), (Nmodels, Nages, Nfilters), or (Nfilters,), float32] The dust spectrum as seen through the given filters

Lbol
[np.ndarray, (Nmodels,) or (Nmodels, Nages)] The total luminosity of the dust model.

get_model_lnu_hires(*params*)

Construct the high-resolution dust SED.

Given a set of parameters, the high-resolution spectrum is constructed.

Parameters

params
[np.ndarray, (Nmodels, 5) or (5,) float32] The dust model parameters.

Returns

Lnu_obs
[np.ndarray, (Nmodels, Nwave), (Nmodels, Nages, Nwave), or (Nwave,), float32] The dust spectrum as seen through the given filters

Lbol
[np.ndarray, (Nmodels,) or (Nmodels, Nages)] The total luminosity of the dust model.

The GrayBody class is used internally to add an additional cold dust component to the AGN emission spectra, balanced with the attenuated power of the "polar dust" component. It cannot currently be selected to replace the Draine & Li model to model the dust emission powered by attenuated stars. It is documented here for completeness; note that it can be initialized on its own and used to build your own custom models.

class lightning.dust.**Graybody**

Bases: BaseEmissionModel

A gray-body dust emission model.

The gray body is a modified blackbody accounting for variations in opacity and emissivity, such that (Casey et al. 2012):

$$L_\nu \propto [1 - \exp(-\tau(\nu))] B_\nu(T) = [1 - \exp(-\tau(\nu))] \frac{\nu^3}{\exp(h\nu/kT) - 1}$$

where the optical depth is taken to be a power law in frequency: `tau(nu) = (nu / nu0)^beta` for some `nu0` where the optical depth is 1. In practice this is usually assumed to be around 100-200 um. Beta is expected to range between ~1–2.5.

Parameters

filter_labels
[list, str] List of filter labels.

redshift
[float] Redshift of the model.

wave_grid
[np.ndarray] Rest frame wavelength grid to evaluate the model on.

Methods

get_model_lnu(params)	Construct the dust SED as observed in the given filters.
get_model_lnu_hires(params)	Construct the high-resolution dust SED.
print_params([verbose])	If *verbose*, print a nicely formatted table of the models, their parameters, and the description of the parameters.

get_model_lnu(*params*)

Construct the dust SED as observed in the given filters.

Note that the model Lnu is normalized to the total luminosity.

Parameters

params
[np.ndarray, (Nmodels, 3) or (3,) float32] The dust model parameters.

Returns

Lnu_obs
[np.ndarray, (Nmodels, Nfilters), (Nmodels, Nages, Nfilters), or (Nfilters,), float32] The dust spectrum as seen through the given filters

Lbol
[np.ndarray, (Nmodels,) or (Nmodels, Nages)] The total luminosity of the dust model.

`get_model_lnu_hires`(*params*)

Construct the high-resolution dust SED.

Note that the model Lnu is normalized to the total luminosity.

Parameters

params
[np.ndarray, (Nmodels, 3) or (3,) float32] The dust model parameters.

Returns

Lnu_obs
[np.ndarray, (Nmodels, Nwave), (Nmodels, Nages, Nwave), or (Nwave,), float32] The dust spectrum as seen through the given filters

Lbol
[np.ndarray, (Nmodels,) or (Nmodels, Nages)] The total luminosity of the dust model.

8.6 AGN Model

`class lightning.agn.AGNModel`

Bases: `BaseEmissionModel`

An implementation of the Stalevski (2016) SKIRTOR models.

The broken power law accretion disk model is the SKIRTOR default. Future versions may include an option to swap in the Schartman+(2005) power law model as in X-Cigale. Optionally includes the X-Cigale recipe for polar dust extinction.

Parameters

filter_labels
[list, str] List of filter labels.

redshift
[float] Redshift of the model.

wave_grid
[np.ndarray, (Nwave,), float, optional] If set, the spectra are interpreted to this wavelength grid.

polar_dust
[bool] If `True`, AGN polar dust extinction and re-emission are implemented following the recipe from X-Cigale Note that even if this keyword is set to `False`, the polar dust optical depth remains a parameter of the model - it just doesn't do anything, and should held at 0 in any fitting. (Default: True)

References

- https://ui.adsabs.harvard.edu/abs/2016MNRAS.458.2288S/abstract
- https://sites.google.com/site/skirtorus

Attributes

Lnu_rest
[numpy.ndarray, (29, 7, 132), float] High-res spectral model grid. First dimension covers U, the second q_PAH, and the third covers wavelength.

Lnu_obs
[numpy.ndarray, (29, 7, 132), float] `(1 + redshift) * Lnu_rest`

mean_Lnu
[numpy.ndarray, (29, 7, Nfilters), float] The `Lnu_obs` grid integrated against the filters.

Lbol
[numpy.ndarray, (29, 7), float] Total luminosity of each model in the grid.

wave_grid_rest
[numpy.ndarray, (132,), float] Rest-frame wavelength grid for the models.

wave_grid_obs
nu_grid_rest
nu_grid_obs

Methods

get_model_lnu(params[, exptau])	Produce the AGN model as observed in the given filters.
get_model_lnu_hires(params[, exptau])	Produce the high-resolution AGN spectral model.
print_params([verbose])	If *verbose*, print a nicely formatted table of the models, their parameters, and the description of the parameters.

get_model_lnu(*params*, *exptau=None*)

Produce the AGN model as observed in the given filters.

Given a set of parameters, produce the observed AGN SED, optionally attenuating it with the ISM dust attenuation model and a polar dust model, a là CIGALE.

Parameters

params
[np.ndarray, (Nmodels, 3) or (3,) float] The AGN model parameters.

exptau
[np.ndarray, (Nmodels, Nwave) or (Nwave,) float] The ISM attenuation curve.

Returns

lnu_hires
[np.ndarray, (Nmodels, Nfilters) or (Nfilters,) float] The high resolution AGN models.

`get_model_lnu_hires`(*params*, *exptau=None*)

Produce the high-resolution AGN spectral model.

Given a set of parameters, produce the AGN spectrum, optionally attenuating it with the ISM dust attenuation model and a polar dust model, a là CIGALE.

Parameters

params
[np.ndarray, (Nmodels, 3) or (3,) float] The AGN model parameters.

exptau
[np.ndarray, (Nmodels, Nwave) or (Nwave,) float] The ISM attenuation curve.

Returns

lnu_hires
[np.ndarray, (Nmodels, Nwave) or (Nwave,) float] The high resolution AGN models.

8.7 X-ray Models

8.7.1 X-ray Emission

The X-ray fitting module implements two simple power law models for XRB populations and AGN, as well as a version of the `QSOSED` model of Kubota & Done (2018).

One noteworthy point about the `StellarPlaw` and `AGNPlaw` models is that they both rely on knowledge of the corresponding UV-IR models to set their normalization. You'll thus see that the `StellarPlaw` `get_*` functions all require a stellar model and its parameters as input, while their equivalents for the `AGNPlaw` model require the AGN model and its parameters.

`class lightning.xray.StellarPlaw`

Bases: *XrayPlawExpcut*

Simple model for stellar X-ray emission.

Inclues a stellar-age parameterization of the luminosity, such that the model normalization is a function of the SFH. The high energy cutoff is fixed, but the photon index can vary.

The luminosity is determined from the SFH model based on the empirical Lx/M - stellar age relationship from Gilbertson+(2022).

Parameters

filter_labels
[list, str] List of filter labels.

arf
[dict or astropy.table.Table or numpy structured array] A structure defining the anciliary response function (ARF) of your X-ray observations. The structure must have three keys, *'ENERG_LO'*, *'ENERG_HI'*, and *'SPECRESP'*, which given the energy bins and binned spectral response respectively. Only used if *xray_mode='counts'* `.

exposure
[float or np.ndarray (Nfilters)] A scalar or array giving the exposure time of the X-ray observations. If an array, it should have the same length as `filter_labels`, with all non-X-ray bands having their exposure time set to 0. Note that you almost certainly don't need to give exposure time as an array, since the energy dependence of the effective area is explicitly given by the ARF. Only used if `xray_mode='counts'`.

redshift
[float] Redshift of the model. If set, `lum_dist` is ignored.

lum_dist
[float] Luminosity distance to the model. If not set, this will be calculated from the redshift and cosmology. (Default: None)

cosmology
[astropy.cosmology.FlatLambdaCDM] The cosmology to assume. Lightning defaults to a flat cosmology with `h=0.7 and Om0=0.3`.

path_to_models
[str] Path to lightning models. Not actually used in normal circumstances.

path_to_filters
[str] Path to lightning filters. Not actually used in normal circumstances.

wave_grid
[tuple (3,), or np.ndarray, (Nwave,), float32, optional] Either a tuple of (lo, hi, Nwave) specifying a log-spaced rest-frame wavelength grid, or an array giving the wavelengths directly. At high redshift this should be constructed carefully to ensure that your bands are covered. (Default: (1e-6, 1e-1, 200))

References

- Gilbertson et al. (2022)

Methods

get_model_countrate(params, stellar_model, ...)	Construct the bandpass-convolved SED in count-rate.
get_model_countrate_hires(params, ...[, exptau])	Construct the high-resolution countrate-density spectrum.
get_model_counts(params, stellar_model, ...)	Construct the bandpass-convolved SED in counts.
get_model_lnu(params, stellar_model, ...[, ...])	Construct the bandpass-convolved SED in Lnu.
get_model_lnu_hires(params, stellar_model, ...)	Construct the high-resolution spectrum in Lnu.
print_params([verbose])	If *verbose*, print a nicely formatted table of the models, their parameters, and the description of the parameters.

get_model_countrate(*params*, *stellar_model*, *stellar_params*, *sfh*, *sfh_params*, *exptau=None*)

Construct the bandpass-convolved SED in count-rate.

This function takes in the stellar and SFH models in order to use the stellar-age parametrization of Lx / M from Gilbertson et al. (2022).

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

stellar_model
[lightning.stellar model] The stellar model.

stellar_params
[np.ndarray(Nmodels, Nparams_st) or np.ndarray(Nparams_st)] The stellar model parameters (i.e. metallicity and possible logU).

sfh
[lightning.sfh.PiecewiseConstSFH or lightning.sfh.FunctionalSFH] The star formation history model.

sfh_params
[np.ndarray(Nmodels, Nparams_sfh) or np.ndarray(Nparams_sfh)] The parameters for the SFH.

exptau
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nwave)] The e(-tau) absorption curve.

Notes

- The absorption free count-rate is not returned by default. It can be accessed by setting exptau to `None`.

`get_model_countrate_hires`(*params*, *stellar_model*, *stellar_params*, *sfh*, *sfh_params*, *exptau=None*)

Construct the high-resolution countrate-density spectrum.

This function takes in the stellar and SFH models in order to use the stellar-age parametrization of Lx / M from Gilbertson et al. (2022).

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

stellar_model
[lightning.stellar model] The stellar model.

stellar_params
[np.ndarray(Nmodels, Nparams_st) or np.ndarray(Nparams_st)] The stellar model parameters (i.e. metallicity and possible logU).

sfh
[lightning.sfh.PiecewiseConstSFH or lightning.sfh.FunctionalSFH] The star formation history model.

sfh_params
[np.ndarray(Nmodels, Nparams_sfh) or np.ndarray(Nparams_sfh)] The parameters for the SFH.

exptau
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nwave)] The e(-tau) absorption curve.

Notes

- The absorption free count-rate spectrum is not returned by default. It can be accessed by setting exptau to `None`.

`get_model_counts`(*params*, *stellar_model*, *stellar_params*, *sfh*, *sfh_params*, *exptau=None*)

Construct the bandpass-convolved SED in counts.

This function takes in the stellar and SFH models in order to use the stellar-age parametrization of Lx / M from Gilbertson et al. (2022).

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

stellar_model
[lightning.stellar model] The stellar model.

stellar_params
[np.ndarray(Nmodels, Nparams_st) or np.ndarray(Nparams_st)] The stellar model parameters (i.e. metallicity and possible logU).

sfh
[lightning.sfh.PiecewiseConstSFH or lightning.sfh.FunctionalSFH] The star formation history model.

sfh_params
[np.ndarray(Nmodels, Nparams_sfh) or np.ndarray(Nparams_sfh)] The parameters for the SFH.

exptau
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nwave)] The e(-tau) absorption curve.

Notes

- The absorption free counts are not returned by default. They can be accessed by setting exptau to `None`.

`get_model_lnu`(*params*, *stellar_model*, *stellar_params*, *sfh*, *sfh_params*, *exptau=None*)

Construct the bandpass-convolved SED in Lnu.

This function takes in the stellar and SFH models in order to use the stellar-age parametrization of Lx / M from Gilbertson et al. (2022).

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

stellar_model
[lightning.stellar model] The stellar model.

stellar_params
[np.ndarray(Nmodels, Nparams_st) or np.ndarray(Nparams_st)] The stellar model parameters (i.e. metallicity and possible logU).

sfh
[lightning.sfh.PiecewiseConstSFH or lightning.sfh.FunctionalSFH] The star formation history model.

sfh_params
[np.ndarray(Nmodels, Nparams_sfh) or np.ndarray(Nparams_sfh)] The parameters for the SFH.

exptau
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nwave)] The e(-tau) absorption curve.

`get_model_lnu_hires`(*params*, *stellar_model*, *stellar_params*, *sfh*, *sfh_params*, *exptau=None*)

Construct the high-resolution spectrum in Lnu.

This function takes in the stellar and SFH models in order to use the stellar-age parametrization of Lx / M from Gilbertson et al. (2022).

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

stellar_model
[lightning.stellar model] The stellar model.

stellar_params
[np.ndarray(Nmodels, Nparams_st) or np.ndarray(Nparams_st)] The stellar model parameters (i.e. metallicity and possible logU).

sfh
[lightning.sfh.PiecewiseConstSFH or lightning.sfh.FunctionalSFH] The star formation history model.

sfh_params
[np.ndarray(Nmodels, Nparams_sfh) or np.ndarray(Nparams_sfh)] The parameters for the SFH.

exptau
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nwave)] The e(-tau) absorption curve.

`class lightning.xray.AGNPlaw`

Bases: *XrayPlawExpcut*

Simple model for AGN X-ray emission.

Uses the Lusso and Risaliti (2017) relationship to connect the intrinsic accretion disk luminosity at 2500 Angstroms to the X-ray luminosity at 2 keV. The high energy cutoff is fixed, but the photon index can vary.

The model includes a parameter representing the deviation from the LR17 relationship. Similar parameters are sometimes called 'x-ray weakness' - an underluminous X-ray spectrum compared to the prediction may be a result of mass loading in the corona providing an alternate source of cooling.

Parameters

filter_labels
[list, str] List of filter labels.

arf
[dict or astropy.table.Table or numpy structured array] A structure defining the anciliary response function (ARF) of your X-ray observations. The structure must have three keys, *'EN-*

ERG_LO', *'ENERG_HI'*, and *'SPECRESP'*, which given the energy bins and binned spectral response respectively. Only used if *xray_mode='counts'* `.

exposure
[float or np.ndarray (Nfilters)] A scalar or array giving the exposure time of the X-ray observations. If an array, it should have the same length as `filter_labels`, with all non-X-ray bands having their exposure time set to 0. Note that you almost certainly don't need to give exposure time as an array, since the energy dependence of the effective area is explicitly given by the ARF. Only used if `xray_mode='counts'`.

redshift
[float] Redshift of the model. If set, `lum_dist` is ignored.

lum_dist
[float] Luminosity distance to the model. If not set, this will be calculated from the redshift and cosmology. (Default: None)

cosmology
[astropy.cosmology.FlatLambdaCDM] The cosmology to assume. Lightning defaults to a flat cosmology with `h=0.7 and Om0=0.3`.

path_to_models
[str] Path to lightning models. Not actually used in normal circumstances.

path_to_filters
[str] Path to lightning filters. Not actually used in normal circumstances.

wave_grid
[tuple (3,), or np.ndarray, (Nwave,), float32, optional] Either a tuple of (lo, hi, Nwave) specifying a log-spaced rest-frame wavelength grid, or an array giving the wavelengths directly. At high redshift this should be constructed carefully to ensure that your bands are covered. (Default: (1e-6, 1e-1, 200))

References

- Lusso and Risaliti (2018)

Methods

get_model_countrate(params, agn_model, ...)	Construct the bandpass-convolved SED in count-rate.
get_model_countrate_hires(params, agn_model, ...)	Construct the high-resolution spectrum in count-rate density.
get_model_counts(params, agn_model, agn_params)	Construct the high-resolution spectrum in Lnu.
get_model_lnu(params, agn_model, agn_params)	Construct the bandpass-convolved SED in Lnu.
get_model_lnu_hires(params, agn_model, ...)	Construct the high-resolution spectrum in Lnu.
`print_params`([verbose])	If *verbose*, print a nicely formatted table of the models, their parameters, and the description of the parameters.

get_model_countrate(*params*, *agn_model*, *agn_params*, *exptau=None*)

Construct the bandpass-convolved SED in count-rate.

This function takes in the agn model to normalize the X-ray spectrum with the Lusso & Risaliti (2017) L2keV - L2500 relationship.

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

agn_model
[lightning.agn.AGNModel] The AGN model.

agn_params
[np.ndarray(Nmodels, Nparams_agn) or np.ndarray(Nparams_agn)] The parameters for the AGN model.

exptau
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nwave)] The e(-tau) absorption curve.

Notes

- The absorption free count-rate is not returned by default. It can be accessed by setting exptau to `None`.

`get_model_countrate_hires`(*params*, *agn_model*, *agn_params*, *exptau=None*)

Construct the high-resolution spectrum in count-rate density.

This function takes in the agn model to normalize the X-ray spectrum with the Lusso & Risaliti (2017) L2keV - L2500 relationship.

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

agn_model
[lightning.agn.AGNModel] The AGN model.

agn_params
[np.ndarray(Nmodels, Nparams_agn) or np.ndarray(Nparams_agn)] The parameters for the AGN model.

exptau
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nwave)] The e(-tau) absorption curve.

Notes

- The absorption free count-rate spectrum is not returned by default. It can be accessed by setting exptau to `None`.

`get_model_counts`(*params*, *agn_model*, *agn_params*, *exptau=None*)

Construct the high-resolution spectrum in Lnu.

This function takes in the agn model to normalize the X-ray spectrum with the Lusso & Risaliti (2017) L2keV - L2500 relationship.

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

agn_model
[lightning.agn.AGNModel] The AGN model.

agn_params
[np.ndarray(Nmodels, Nparams_agn) or np.ndarray(Nparams_agn)] The parameters for the AGN model.

exptau
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nwave)] The e(-tau) absorption curve.

Notes

- The absorption free counts are not returned by default. They can be accessed by setting exptau to `None`.

`get_model_lnu`(*params*, *agn_model*, *agn_params*, *exptau=None*)

Construct the bandpass-convolved SED in Lnu.

This function takes in the agn model to normalize the X-ray spectrum with the Lusso & Risaliti (2017) L2keV - L2500 relationship.

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

agn_model
[lightning.agn.AGNModel] The AGN model.

agn_params
[np.ndarray(Nmodels, Nparams_agn) or np.ndarray(Nparams_agn)] The parameters for the AGN model.

exptau
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nwave)] The e(-tau) absorption curve.

`get_model_lnu_hires`(*params*, *agn_model*, *agn_params*, *exptau=None*)

Construct the high-resolution spectrum in Lnu.

This function takes in the agn model to normalize the X-ray spectrum with the Lusso & Risaliti (2017) L2keV - L2500 relationship.

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

agn_model
[lightning.agn.AGNModel] The AGN model.

agn_params
[np.ndarray(Nmodels, Nparams_agn) or np.ndarray(Nparams_agn)] The parameters for the AGN model.

exptau
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nwave)] The e(-tau) absorption curve.

The `QSOSED` model, however, doesn't need to know about the UV-IR AGN model, because the opposite is true: the UV-IR AGN component is normalized by the `QSOSED` model, allowing the black hole mass and Eddington ratio to set the overall luminosity of the entire AGN model.

class lightning.xray.**Qsosed**

Bases: *XrayEmissionModel*

Physically motivated quasar model for X-ray emission.

Parametrized by the black hole mass and Eddington ratio (here called mdot). See Kubota & Done (2018) for details.

The model is available in XSpec, and the implementation here was generated using Sherpa.

Parameters

filter_labels
[list, str] List of filter labels.

arf
[dict or astropy.table.Table or numpy structured array] A structure defining the anciliary response function (ARF) of your X-ray observations. The structure must have three keys, *'ENERG_LO'*, *'ENERG_HI'*, and *'SPECRESP'*, which given the energy bins and binned spectral response respectively. Only used if *xray_mode='counts'* `.

exposure
[float or np.ndarray (Nfilters)] A scalar or array giving the exposure time of the X-ray observations. If an array, it should have the same length as `filter_labels`, with all non-X-ray bands having their exposure time set to 0. Note that you almost certainly don't need to give exposure time as an array, since the energy dependence of the effective area is explicitly given by the ARF. Only used if `xray_mode='counts'`.

redshift
[float] Redshift of the model. If set, `lum_dist` is ignored.

lum_dist
[float] Luminosity distance to the model. If not set, this will be calculated from the redshift and cosmology. (Default: None)

cosmology
[astropy.cosmology.FlatLambdaCDM] The cosmology to assume. Lightning defaults to a flat cosmology with `h=0.7 and Om0=0.3`.

path_to_models
[str] Path to lightning models. Not actually used in normal circumstances.

path_to_filters
[str] Path to lightning filters. Not actually used in normal circumstances.

wave_grid
[tuple (3,), or np.ndarray, (Nwave,), float32, optional] Either a tuple of (lo, hi, Nwave) specifying a log-spaced rest-frame wavelength grid, or an array giving the wavelengths directly. At high redshift this should be constructed carefully to ensure that your bands are covered. (Default: (1e-6, 1e-1, 200))

References

- Kubota and Done (2018)

Methods

get_model_L2500(params)	Calculate the intrinsic L2500
get_model_countrate(params[, exptau])	Construct the bandpass-convolved SED in count-rate.
get_model_countrate_hires(params[, exptau])	Construct the high-resolution spectrum in count-rate density.
get_model_counts(params[, exptau])	Construct the high-resolution spectrum in Lnu.
get_model_lnu(params[, exptau])	Construct the bandpass-convolved SED in Lnu.
get_model_lnu_hires(params[, exptau])	Construct the high-resolution spectrum in Lnu.
print_params([verbose])	If *verbose*, print a nicely formatted table of the models, their parameters, and the description of the parameters.

get_model_L2500(*params*)

Calculate the intrinsic L2500

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

get_model_countrate(*params*, *exptau=None*)

Construct the bandpass-convolved SED in count-rate.

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

exptau
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nwave)] The e(-tau) absorption curve.

Notes

- The absorption free count-rate is not returned by default. It can be accessed by setting exptau to `None`.

get_model_countrate_hires(*params*, *exptau=None*)

Construct the high-resolution spectrum in count-rate density.

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

exptau
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nwave)] The e(-tau) absorption curve.

Notes

- The absorption free count-rate spectrum is not returned by default. It can be accessed by setting exptau to `None`.

`get_model_counts`(*params*, *exptau=None*)

Construct the high-resolution spectrum in Lnu.

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

exptau
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nwave)] The e(-tau) absorption curve.

Notes

- The absorption free counts are not returned by default. They can be accessed by setting exptau to `None`.

`get_model_lnu`(*params*, *exptau=None*)

Construct the bandpass-convolved SED in Lnu.

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

exptau
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nwave)] The e(-tau) absorption curve.

`get_model_lnu_hires`(*params*, *exptau=None*)

Construct the high-resolution spectrum in Lnu.

Parameters

params
[np.ndarray(Nmodels, Nparams) or np.ndarray(Nparams)] An array of model parameters. For purposes of vectorization this can be a 2D array, where the first dimension cycles over different sets of parameters.

exptau
[np.ndarray(Nmodels, Nwave) or np.ndarray(Nwave)] The e(-tau) absorption curve.

The emission models above extend the `XrayPlaw` and `XrayEmissionModel` classes documented at the bottom of this page for the sake of completeness.

8.7.2 X-ray Absorption

Two absorption models are implemented: the Tubingen-Boulder absorption model (`tbabs`) which includes more atomic physics and extends to the UV, and the photoelectric absorption model from XSpec (`phabs`) which covers only the X-rays.

class lightning.xray.**Tbabs**

Bases: *TabulatedAtten*

Tubingen-Boulder absorption model.

Includes cross sections from gas phase ISM, grains, and molecular hydrogen. Atomic abundances are fixed to the default.

Parameters

wave
[np.ndarray, (Nwave,), float] Rest frame wavelength grid to evaluate the model on.

path_to_models
[str] Path to lightning models. Not actually used in normal circumstances.

References

- https://heasarc.gsfc.nasa.gov/xanadu/xspec/manual/XSmodelTbabs.html
- https://ui.adsabs.harvard.edu/abs/2000ApJ...542..914W/abstract

Methods

evaluate(params)	Evaluate the absorption as a function of wavelength for the given parameters.
print_params([verbose])	If *verbose*, print a nicely formatted table of the models, their parameters, and the description of the parameters.

evaluate(*params*)

Evaluate the absorption as a function of wavelength for the given parameters.

Parameters

params
[np.ndarray, (Nmodels, 1) or (1,)] Values for NH.

Returns

expminustau
[(Nmodels, Nwave)]

class lightning.xray.**Phabs**

Bases: *TabulatedAtten*

Photo-electric absorption model.

Abundances are fixed to the default.

Parameters

wave
[np.ndarray, (Nwave,), float] Rest frame wavelength grid to evaluate the model on.

path_to_models
[str] Path to lightning models. Not actually used in normal circumstances.

References

- https://heasarc.gsfc.nasa.gov/xanadu/xspec/manual/node264.html

Methods

evaluate(params)	Evaluate the absorption as a function of wavelength for the given parameters.
print_params([verbose])	If *verbose*, print a nicely formatted table of the models, their parameters, and the description of the parameters.

evaluate(*params*)

Evaluate the absorption as a function of wavelength for the given parameters.

Parameters

params
[np.ndarray, (Nmodels, 1) or (1,)] Values for NH.

Returns

expminustau
[(Nmodels, Nwave)]

8.7.3 X-ray Base Classes

class lightning.xray.**XrayPlawExpcut**

Bases: *XrayEmissionModel*

Power law emission model with an exponential cutoff at high energy.

Methods

get_model_countrate(params[, exptau])	Produce the mean model countrate density in the bandpass.
get_model_countrate_hires(params[, exptau])	Produce the high-resolution model countrate density in counts s-1 Hz-1.
get_model_counts(params[, exptau])	Produce the model counts in the bandpass.
get_model_lnu(params[, exptau])	Overwrite this method to provide the actual evaluation of the model as observed in the given filters.
get_model_lnu_hires(params[, exptau])	Overwrite this method to provide the actual evaluation of the high-res spectral model.
print_params([verbose])	If *verbose*, print a nicely formatted table of the models, their parameters, and the description of the parameters.

get_model_countrate(*params*, *exptau=None*)

Produce the mean model countrate density in the bandpass.

get_model_countrate_hires(*params*, *exptau=None*)

Produce the high-resolution model countrate density in counts s-1 Hz-1. Note that these could probably be implemented here; the only thing that's specific to each individual model is the high-resolution Lnu spectrum.

get_model_counts(*params*, *exptau=None*)

Produce the model counts in the bandpass.

get_model_lnu(*params*, *exptau=None*)

Overwrite this method to provide the actual evaluation of the model as observed in the given filters.

get_model_lnu_hires(*params*, *exptau=None*)

Overwrite this method to provide the actual evaluation of the high-res spectral model.

class lightning.xray.base.**XrayEmissionModel**

Bases: BaseEmissionModel

Base class for X-ray emission models.

Methods

get_model_countrate(params)	Produce the mean model countrate density in the bandpass.
get_model_countrate_hires(params)	Produce the high-resolution model countrate density in counts s-1 Hz-1.
get_model_counts(params)	Produce the model counts in the bandpass.
get_model_lnu(params)	Overwrite this method to provide the actual evaluation of the model as observed in the given filters.
get_model_lnu_hires(params)	Overwrite this method to provide the actual evaluation of the high-res spectral model.
print_params([verbose])	If *verbose*, print a nicely formatted table of the models, their parameters, and the description of the parameters.

__init__(*filter_labels*, *arf*, *exposure*, *redshift*, *lum_dist=None*, *cosmology=None*, *path_to_models=None*, *path_to_filters=None*, ***kwargs*)

Generic initialization. Actual model-building should be handled by implementing the *construct_model* and *construct_model_grid* methods.

get_model_countrate(*params*)

Produce the mean model countrate density in the bandpass.

get_model_countrate_hires(*params*)

Produce the high-resolution model countrate density in counts s-1 Hz-1. Note that these could probably be implemented here; the only thing that's specific to each individual model is the high-resolution Lnu spectrum.

get_model_counts(*params*)

Produce the model counts in the bandpass.

8.8 Prior Distributions

Priors in `Lightning` are built around two base classes: `AnalyticPrior` for priors with a functional form and `TabulatedPrior` for empirically derived priors. New user-specific priors can be added by defining classes that inherit from one of these base classes and define the `evaluate`, `quantile`, and `sample` methods.

In the course of normal use, the `sample` method is the only one likely to be used regularly, in initializing the mcmc.

class lightning.priors.base.**AnalyticPrior**

Base class for priors with a functional form.

Need only be overwritten if there's specific requirements for the prior parameters (i.e. b > a for the uniform prior).

Methods

evaluate(x)	This function must be overwritten by each specific prior, implementing the PDF.
quantile(q)	This function must be overwritten by each specific prior, implementing the quantile function/PPF (the inverse of the CDF).
sample(size[, rng, seed])	Sample from the prior.

__init__(*params*)

__new__(**args*, ***kwargs*)

evaluate(*x*)

This function must be overwritten by each specific prior, implementing the PDF.

quantile(*q*)

This function must be overwritten by each specific prior, implementing the quantile function/PPF (the inverse of the CDF).

sample(*size*, *rng=None*, *seed=None*)

Sample from the prior.

Parameters

size
[int] Number of samples to draw

rng
[numpy.random.Generator] Numpy object for random number generation; see `numpy.random.default_rng()`

seed
[int] Seed for random number generation. If you pass a pre-constructed generator this is ignored.

Returns

samples
[numpy array] Random samples

class lightning.priors.base.**TabulatedPrior**

Base class for tabulated priors.

Keywords are passed on to scipy.interpolate.interp1d.

Methods

evaluate(x)	Evaluate the scipy.interpolate.interp1d object representing the PDF on `x`.
quantile(q)	Evaluate the scipy.interpolate.interp1d object representing the quantile function on `q`.
sample(size[, rng, seed])	Sample from the prior.

__init__(*x*, *y*, ***kwargs*)

__new__(**args*, ***kwargs*)

evaluate(*x*)

Evaluate the scipy.interpolate.interp1d object representing the PDF on `x`.

quantile(*q*)

Evaluate the scipy.interpolate.interp1d object representing the quantile function on `q`.

sample(*size*, *rng=None*, *seed=None*)

Sample from the prior.

Parameters

size
[int] Number of samples to draw

rng
[numpy.random.Generator] Numpy object for random number generation; see `numpy.random.default_rng()`

seed
[int] Seed for random number generation. If you pass a pre-constructed generator this is ignored.

Returns

samples
[numpy array] Random samples

The `UniformPrior` and `NormalPrior` classes inherent from `AnalyticPrior` and define the uniform and normal distributions, respectively.

class lightning.priors.**UniformPrior**

Bases: *AnalyticPrior*

Uniform prior. Parameters are lower bound a and upper bound b, in that order.

PDF:

$$p(x) = \begin{cases} \frac{1}{b-a} & , \ x \in [a, b) \\ 0 & , \ \text{otherwise} \end{cases}$$

Quantile function:

$$x(q) = q(b - a) + a$$

__init__(*params*)

__new__(**args*, ***kwargs*)

class lightning.priors.**NormalPrior**

Bases: *AnalyticPrior*

Normal prior. Parameters are mu and sigma, in that order.

PDF:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp[-\frac{(x-\mu)^2}{\sigma^2}]$$

Quantile function:

$$x(q) = \mu + \sigma\sqrt{2}\text{erfinv}(2q - 1)$$

__init__(*params*)

__new__(**args*, ***kwargs*)

Constant parameters in Lightning can be fixed to a single value using the `ConstantPrior`, which implements a delta-function-like prior. In practice this isn't a true prior, but it tells `Lightning` not to bother sampling the given parameter and what its value should be.

class lightning.priors.**ConstantPrior**

Bases: *AnalyticPrior*

Delta function prior. The single parameter is value a.

PDF:

$$p(x) = \begin{cases} 1 & , x = a \\ 0 & , \text{otherwise} \end{cases}$$

Quantile function:

$$x(q) = a \forall q$$

Holding a parameter constant is not really a prior so much as a reduction in the dimensionality of the problem, so this is basically a dummy prior, which tells the sampler to hold a parameter constant and what its value should be.

__init__(*params*)

__new__(**args*, ***kwargs*)

8.9 Plotting

We have implemented a number of plotting functions in lightning to make visualizing results easy. These are typically implemented such that they take a `Lightning` object as their first argument, followed by an MCMC chain (and the chain of log-probability values, where necessary). Most plotting functions can accept dictionaries of keyword arguments describing the colors and styles of lines, markers, etc. These are passed through to the relevant `matplotlib` functions. All the plotting functions can also plot into existing axes by supplying the `ax` keyword (for `corner_plot`, set the `fig` keyword instead to a figure containing the appropriate number of axes).

Two convenience items are also implemented in this module: the `ModelBand` class adapted from `Ultranest`, and the `step_curve` function designed to make plotting step functions with shaded uncertainties easier. Both are mostly meant for internal use, but are documented below for completeness.

class lightning.plots.**ModelBand**

Bases: `object`

A class interface for storing a bunch of model realizations and plotting them, as shaded bands, quantiles, or individual realizations.

This is very much taken from UltraNest's `PredictionBand` class with some tweaks. Ultranest is (c) 2019 by Johannes Buchner, and is available under GPL v3. Find it here: https://johannesbuchner.github.io/UltraNest/

All plotting functions (`shade`, `line`, `realizations`) draw into the current axes unless the `ax` keyword is set. The plotting functions also all pass through keyword arguments, allowing you to modify color, alpha, etc.

Methods

add(y)	Add a realization to the band.
line([q, ax])	Draw a line at the specified quantile `q`.
realizations([num, replace, ax])	Plot `num` randomly chosen model realizations.
shade([q, ax])	Draw a shaded region between the quantiles specified by `q`.

__init__(*x*, *seed=None*)

add(*y*)

Add a realization to the band. Currently limited to one at a time.

line(*q=0.5*, *ax=None*, ***kwargs*)

Draw a line at the specified quantile `q`. Defaults to the median.

realizations(*num=1*, *replace=False*, *ax=None*, ***kwargs*)

Plot `num` randomly chosen model realizations. If `replace` is set, the same realization can be chosen multiple times.

shade(*q=(0.16, 0.84)*, *ax=None*, ***kwargs*)

Draw a shaded region between the quantiles specified by `q`. Defaults to the 68% interval.

lightning.plots.**chain_plot**(*lgh*, *samples*, *plot_median=True*, *median_color='darkorange'*, ***kwargs*)

Produce a chain plot of samples from an SED fit.

The chain plots are placed in a single matplotlib figure, all in a single column. Not that this may (will) be unwieldy in cases with many free parameters.

Parameters

lgh
[lightning.Lightning object] Used to assign names (and maybe units) to the sample dimensions.

samples
[np.ndarray, (Nsamples, Nparam), float] The sampled parameters. Note that this should also include any constant parameters.

plot_median
[bool] If true, draw a horizontal line at the median of the chain.

median_color
[string] The color to plot the median line with.

****kwargs**
[dict] Keyword arguments are passed on to *plt.plot* for the chain plot.

Returns

fig
[Matplotlib figure containing the plot]

axs
[List of axes in the plot]

lightning.plots.**corner_plot**(*lgh*, *samples*, ***kwargs*)

Produce a corner plot of samples from an SED fit.

This is just a thin wrapper around corner that just figures out which dimensions are constant and assigns parameter labels to the chains. See the link below for a complete listing of keywords understood by corner.

Parameters

lgh
[lightning.Lightning object] Used to assign names (and maybe units) to the sample dimensions.

samples
[np.ndarray, (Nsamples, Nparam), float] The sampled parameters. Note that this should also include any constant parameters.

****kwargs**
[dict] Keyword arguments are passed on to *corner.corner*.

Returns

fig
[Matplotlib figure containing the plot]

References

- https://corner.readthedocs.io/en/latest/api/

lightning.plots.**sed_plot_bestfit**(*lgh*, *samples*, *logprob_samples*, *plot_components=False*, *plot_unatt=False*, *ax=None*, *xlim=(0.1, 1200)*, *ylim=(9000000.0, None)*, *xlabel='Observed-Frame Wavelength $[\\rm \\mu m]$'*, *ylabel='$\\nu L_{\\nu}\\ [\\rm L_\\odot]$'*, *stellar_unatt_kwargs={'color': 'dodgerblue', 'label': 'Unattenuated stellar pop.'}*, *stellar_att_kwargs={'color': 'red', 'label': 'Attenuated stellar pop.'}*, *agn_kwargs={'color': 'darkorange', 'label': 'AGN'}*, *dust_kwargs={'color': 'green', 'label': 'Dust'}*, *total_kwargs={'color': 'slategray', 'label': 'Total model'}*, *data_kwargs={'capsize': 2, 'color': 'k', 'label': 'Data', 'linestyle': '', 'marker': 'D', 'markerfacecolor': 'k'}*, *show_legend=True*, *legend_kwargs={'frameon': False, 'loc': 'upper right'}*, *uplim_sigma=3*, *uplim_kwargs={'color': 'k', 'marker': '$\\downarrow$'}*)

Best-fit SED plot.

Selects the highest log-probability model from the chain. Individual components may be plotted.

Parameters

lgh
[lightning.Lightning object] Used to assign names (and maybe units) to the sample dimensions.

samples
[np.ndarray, (Nsamples, Nparam), float] The sampled parameters. Note that this should also include any constant parameters.

logprob_samples
[np.ndarray, (Nsamples,), float] Log-probability chain.

plot_components
[bool] If `True`, plot the components of the SED.

plot_unatt
[bool] If `True`, also plot the unattenuated stellar SED.

ax
[matplotlib.axes.Axes] Axes to draw the plot in. (Default: None)

xlim
[tuple]

ylim
[tuple]

xlabel
[str]

ylabel
[str]

stellar_unatt_kwargs
[dict]

stellar_att_kwargs
[dict]

agn_kwargs
[dict]

dust_kwargs
[dict]

total_kwargs
[dict]

data_kwargs
[dict]

show_legend
[bool]

legend_kwargs
[dict]

uplim_sigma
[int] How many sigma should upper limits be drawn at? (Default: 3)

uplim_kwargs
[dict] Each of the above `*_kwargs` parameters is a dict containing keyword arguments describing the color, style, label, etc. of the corresponding plot element, passed through to the appropriate `matplotlib` function.

Returns

fig
[Matplotlib figure containing the plot]

ax
[Axes containing the plot]

`lightning.plots.`**`sed_plot_delchi`**(*lgh*, *samples*, *logprob_samples*, *ax=None*, *xlim=(0.1, 1200)*, *ylim=(-5.1, 5.1)*, *lines=[0, -1, 1]*, *linecolors=['slategray']*, *linestyles=['-', '--', '--']*, *xlabel='Observed-Frame Wavelength $[\\rm \\mu m]$'*, *ylabel='Residual $[\\sigma]$'*, *data_kwargs={'capsize': 2, 'color': 'k', 'label': 'Data', 'linestyle': '', 'marker': 'D', 'markerfacecolor': 'k'}*, *uplim_sigma=3*, *uplim_kwargs={'color': 'k', 'marker': '$\\downarrow$'}*)

Delta-chi residuals for the best-fit SED.

$$\delta\chi = (L_\nu^{\rm obs} - L_\nu^{\rm mod})/\sigma$$

Selects the highest log-probability model from the chain.

Parameters

lgh
[lightning.Lightning object] Used to assign names (and maybe units) to the sample dimensions.

samples
[np.ndarray, (Nsamples, Nparam), float] The sampled parameters. Note that this should also include any constant parameters.

logprob_samples
[np.ndarray, (Nsamples,), float] Log-probability chain.

xlim
[tuple]

ylim
[tuple]

lines
[list or tuple] Values to draw horizontal guide lines at (in sigma). (Default: [-1,0,1])

linecolors
[list or str] Corresponding colors for the guide lines. (Default: 'slategray')

linestyles
[list or str] Corresponding styles for the guide lines. (Default: ['-', '–', '–'])

xlabel
[str]

ylabel
[str]

data_kwargs
[dict]

uplim_sigma
[int] How many sigma should upper limits be drawn at? (Default: 3)

uplim_kwargs
[dict] Each of the above `*_kwargs` parameters is a dict containing keyword arguments describing the color, style, label, etc. of the corresponding plot element, passed through to the appropriate `matplotlib` function.

Returns

fig
[Matplotlib figure containing the plot]

ax
[Axes containing the plot]

lightning.plots.**sed_plot_morebayesian**(*lgh*, *samples*, *logprob_samples*, *plot_components=False*, *plot_unatt=False*, *ax=None*, *xlim=(0.1, 1200)*, *ylim=(9000000.0, None)*, *xlabel='Observed-Frame Wavelength $[\\rm \\mu m]$'*, *ylabel='$\\nu L_{\\nu}\\ [\\rm L_\\odot]$'*, *stellar_unatt_kwargs={'alpha': 0.5, 'color': 'dodgerblue', 'label': 'Unattenuated stellar pop.'}*, *stellar_att_kwargs={'alpha': 0.5, 'color': 'red', 'label': 'Attenuated stellar pop.'}*, *agn_kwargs={'alpha': 0.5, 'color': 'darkorange', 'label': 'AGN'}*, *dust_kwargs={'alpha': 0.5, 'color': 'green', 'label': 'Dust'}*, *total_kwargs={'alpha': 0.5, 'color': 'slategray', 'label': 'Total model'}*, *data_kwargs={'capsize': 2, 'color': 'k', 'label': 'Data', 'linestyle': '', 'marker': 'D', 'markerfacecolor': 'k'}*, *show_legend=True*, *legend_kwargs={'frameon': False, 'loc': 'upper right'}*, *uplim_sigma=3*, *uplim_kwargs={'color': 'k', 'marker': '$\\downarrow$'}*)

More bayesian visualization of the fit, agnostic of the best fit. Better name TBD

Shows the 16th-84th percentile range of the model Lnu. Currently this doesn't show the median or best-fit, just shaded polygons indicating the range.

Parameters

lgh
[lightning.Lightning object] Used to assign names (and maybe units) to the sample dimensions.

samples
[np.ndarray, (Nsamples, Nparam), float] The sampled parameters. Note that this should also include any constant parameters.

logprob_samples
[np.ndarray, (Nsamples,), float] Log-probability chain.

plot_components
[bool] If `True`, plot the components of the SED.

plot_unatt
[bool] If `True`, also plot the unattenuated stellar SED.

ax
[matplotlib.axes.Axes] Axes to draw the plot in. (Default: None)

xlim
[tuple]

ylim
[tuple]

xlabel
[str]

ylabel
[str]

stellar_unatt_kwargs
[dict]

stellar_att_kwargs
[dict]

agn_kwargs
[dict]

dust_kwargs
[dict]

total_kwargs
[dict]

data_kwargs
[dict]

show_legend
[bool]

legend_kwargs
[dict]

uplim_sigma
[int] How many sigma should upper limits be drawn at? (Default: 3)

uplim_kwargs
[dict] Each of the above `*_kwargs` parameters is a dict containing keyword arguments describing the color, style, label, etc. of the corresponding plot element, passed through to the appropriate `matplotlib` function.

Returns

fig
[Matplotlib figure containing the plot]

ax
[Axes containing the plot]

`lightning.plots.`**`sfh_plot`**(*lgh*, *samples*, *xlabel='Lookback Time t [yr]'*, *ylabel='SFR(t) $[\\rm M_{\\odot}\\ yr^{-1}]$'*, *ax=None*, *shade_band=True*, *shade_q=(0.16, 0.84)*, *shade_kwargs={'alpha': 0.3, 'color': 'k', 'zorder': 0}*, *plot_line=True*, *line_q=0.5*, *line_kwargs={'color': 'k', 'zorder': 1}*, *ylim=None*)

A plot of the posterior SFR as a function of time.

The default behavior is to plot the median of the posterior along with the shaded 16th to 84th percentiles.

Parameters

lgh
[lightning.Lightning object] Used to assign names (and maybe units) to the sample dimensions.

samples
[np.ndarray, (Nsamples, Nparam), float] The sampled parameters. Note that this should also include any constant parameters.

xlabel
[str]

ylabel
[str]

ax
[matplotlib.axes.Axes] Axes to draw the plot in. (Default: None)

shade_band
[bool] Should the uncertainty band be shaded?

shade_q
[tuple] Quantiles to shade between. (Default: (0.16, 0.84))

shade_kwargs
[dict]

plot_line
[bool] Should a line be plotted?

line_q
Quantile to plot the line at (Default: 0.5)

line_kwargs
[dict] Each of the above `*_kwargs` parameters is a dict containing keyword arguments describing the color, style, label, etc. of the corresponding plot element, passed through to the appropriate `matplotlib` function.

ylim
[tuple]

Returns

fig
[Matplotlib figure containing the plot]

ax
[Axes containing the plot]

lightning.plots.**step_curve**(*bin_edges*, *y_values*, *anchor=False*)

From a histogram (bin edges and count values) generate a step curve for use with matplotlib's `plot` function.

This function takes an array of N + 1 bin edges and an array of N values and turns them into a list of 2N vertices so that you can make a step plot with the `plot` command rather than `step`, `hlines` and `vlines`, etc.

Parameters

bin_edges
[numpy array, (N+1), float] Array defining the edges of the histogram bins (i.e., the locations of the steps). Assumed to be a monotonically increasing sequence.

y_values
[numpy array, (..., N), float] Array containing the y values corresponding to bin_edges: the value of the function on (bin_edges[i], bin_edges[i+1]) is y_values[...,i]. Note that this can be a 2D array, where the first axis cycles among different curves.

anchor
[bool (default False)] If True, the returned curve will be anchored to 0 on both sides.

Returns

x
[np.ndarray]

y
[np.ndarray] Two numpy arrays containing the x- and y-values of the step curve, respectively. See note on size below.

Notes

If anchor is False, each curve has 2N points, where N is the number of y-values.

If anchor is True, each curve has 2N + 2 points.

This function actually doesn't do any plotting, it just makes arrays to feed into `plot`.

8.10 Postprocessing

lightning.postprocessing.**postprocess_catalog**(*res_filenames*, *model_filenames*, *solver_mode='mcmc'*, *model_mode='json'*, *names=None*, *catalog_name='postprocessed_catalog.hdf5'*)

Given lists of chain files and model files, merge the results into a postprocessed catalog.

This postprocessing script is for *samplers*, not for maximum likelihood methods. Luckily I haven't fully implemented any of the maximum likelihood methods yet.

This script uses h5py to produce an output file in HDF5 format. I've made this choice to allow for non-homogeneous model setups, e.g. different numbers of bandpasses and parameters per source. The structure and content of the HDF5 file is as follows (for each source):

```
└──sourcename
    ├── mcmc
    │   ├── logprob_samples (Nsamples)
    │   └── samples (Nsamples, Nparams)
    ├── parameters
    │   ├── modelname
    │   │   └── parametername
    │   │       ├── best ()
    │   │       ├── hi ()
    │   │       ├── lo ()
    │   │       └── med ()
    └── properties
        ├── filter_labels (Nfilters)
        ├── lnu (Nfilters)
        ├── lnu_unc (Nfilters)
        ├── lumdist ()
        ├── mstar
        │   ├── best ()
        │   ├── hi ()
        │   ├── lo ()
        │   └── med ()
        ├── redshift ()
        └── pvalue ()
```

The "modelname" and "parametername" groups under the "parameters" group repeat for every model and parameter. For piecewise-constant SFHs the "properties" group also contains the age bin edges for the SFH. Quantiles ("*/lo" and "*/hi") are computed at the 16 and 84th percentile.

For solver_mode='mcmc', the chains are also expected to be in HDF5 format, with the following structure:

```
mcmc
├── logprob_samples (Nsamples)
├── samples (Nsamples, Nparams)
└── autocorr (Nparams)
```

Whereas for solver_mode='mle', the results are expected to be formated as:

```
res
├── bestfit (Nparams)
└── chi2_best ()
```

I'll provide a function to make such HDF5 result files soon if I haven't already.

Parameters

res_filenames
[array-like, str] A list of filenames pointing to the result files.

model_filenames
[array-like, str] A list of filenames poitning to the model files (either json or pickles).

model_mode
[str] Method for model serialization, either "json" or "pickle". (Default: "json")

names
[array-like, str] Names for each of the galaxies. If None (default), we'll just guess based on the filenames of the chains assuming that they're named something like [NAME]_chain.h5.

catalog_name
[str] (Default: "postprocessed_catalog.hdf5")

Returns

None

Notes

TODO:

- Allow user-supplied quantiles.
- Add additional properties; allow user specification of properties? That might be a whole chore.

8.11 Posterior Predictive Checks

`lightning.ppc.ppc`(*lgh*, *samples*, *logprob_samples*, *Nrep=1000*, *seed=None*, *counts_dist='gaussian'*)

Compute the posterior predictive check p-value given a set of samples and a Lightning object

Parameters

lgh
[lightning.Lightning object] Used to compute chi2 values.

samples
[np.ndarray, (Nsamples, Nparam), float] The samples to compute the PPC with. Note that this must also include any constant parameters, since we need them to compute the model luminosities.

logprob_samples
[np.ndarray, (Nsamples,), float] The logprob corresponding to each sample. Used to weight samples.

Nrep
[int] Number of realizations of the data to compute from the model.

seed
[float] Seed for random number generation.

Returns

p-value
[float] A single p-value for the given chain. Extremely low p-values indicate underfitting, that the model is not flexible enough to produce the observed variation in the data (or that the uncertainties on the data are too small), while extremely high p-values (close to 1) may indicate over-fitting, where the model is *too* flexible (or the uncertainties overly conservative).

Notes

The implementation of PPC here is ported from IDL Lightning.

lightning.ppc.**ppc_sed**(*lgh*, *samples*, *logprob_samples*, *Nrep=1000*, *seed=None*, *ax=None*, *normalize=False*, *counts_dist='gaussian'*)

Make an SED plot representing the posterior predictive check.

The idea is that this can serve as a diagnostic plot showing where the model may be too inflexible to reproduce individual datapoints. This may show you if, e.g. your low p-value is driven by an individual band.

Parameters

lgh
[lightning.Lightning object] Used to compute chi2 values.

samples
[np.ndarray, (Nsamples, Nparam), float] The samples to compute the PPC with. Note that this must also include any constant parameters, since we need them to compute the model luminosities.

logprob_samples
[np.ndarray, (Nsamples,), float] The logprob corresponding to each sample. Used to weight samples.

Nrep
[int] Number of realizations of the data to compute from the model.

seed
[float] Seed for random number generation. Useful for matching your PPC p-value to the plot.

ax
[matplotlib.axis.Axes] Axes to draw the plot into.

normalize
[bool] If True, the data and quantiles of the reproduced data are divided by the median of the reproduced data before plotting.

Returns

PPC SED plot figure: an SED plot showing the quantile bands of the reproduced data, with the observed data overplotted.

CHAPTER

NINE

ATTRIBUTION

A paper describing `lightning.py` is forthcoming by Monson et al.; this page will be updated on submission. In the meantime, work using the new PEGASE+Cloudy and BPASS+Cloudy models are encouraged to also cite Lehmer et al. (2024), while work using models described in Doore et al. (2023) should also cite that paper:

```
@ARTICLE{2023ApJS..266...39D,
    author = {{Doore}, Keith and {Monson}, Erik B. and {Eufrasio}, Rafael T. and {Lehmer}
↪, Bret D. and {Garofali}, Kristen and {Basu-Zych}, Antara},
    title = "{Lightning: An X-Ray to Submillimeter Galaxy SED-fitting Code with␣
↪Physically Motivated Stellar, Dust, and AGN Models}",
    journal = {\apjs},
    keywords = {Extragalactic astronomy, Galaxy properties, Star formation, Spectral␣
↪energy distribution, 506, 615, 1569, 2129, Astrophysics - Astrophysics of Galaxies},
    year = 2023,
    month = jun,
    volume = {266},
    number = {2},
    eid = {39},
    pages = {39},
    doi = {10.3847/1538-4365/accc29},
    archivePrefix = {arXiv},
    eprint = {2304.06753},
    primaryClass = {astro-ph.GA},
    adsurl = {https://ui.adsabs.harvard.edu/abs/2023ApJS..266...39D},
    adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```

CHAPTER

TEN

LICENSE

`lightning.py` is available under the terms of the MIT license.

CHAPTER

ELEVEN

ACKNOWLEDGMENTS

`lightning.py` was primarily developed by Erik B. Monson, with significant contributions from Amirnezam Amiri, Keith Doore, and Rafael Eufrasio. Much of `lightning.py` is outright ported from or at least heavily based on code from IDL `Lightning`, which was developed by Rafael Eurfrasio, Keith Doore, and Erik B. Monson.

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

l

lightning.plots, 163
lightning.postprocessing, 170
lightning.ppc, 172

INDEX

Symbols

__init__() (*lightning.Lightning method*), 109
__init__() (*lightning.attenuation.CalzettiAtten method*), 116
__init__() (*lightning.attenuation.ModifiedCalzettiAtten method*), 115
__init__() (*lightning.attenuation.base.AnalyticAtten method*), 117
__init__() (*lightning.attenuation.base.TabulatedAtten method*), 117
__init__() (*lightning.plots.ModelBand method*), 163
__init__() (*lightning.priors.ConstantPrior method*), 163
__init__() (*lightning.priors.NormalPrior method*), 162
__init__() (*lightning.priors.UniformPrior method*), 162
__init__() (*lightning.priors.base.AnalyticPrior method*), 160
__init__() (*lightning.priors.base.TabulatedPrior method*), 161
__init__() (*lightning.sfh.DelayedExponentialSFH method*), 119
__init__() (*lightning.sfh.PiecewiseConstSFH method*), 118
__init__() (*lightning.sfh.SingleExponentialSFH method*), 121
__init__() (*lightning.sfh.base.FunctionalSFH method*), 122
__init__() (*lightning.stellar.BPASSBurstA24 method*), 140
__init__() (*lightning.stellar.PEGASEBurstA24 method*), 131
__init__() (*lightning.xray.base.XrayEmissionModel method*), 159
__new__() (*lightning.attenuation.base.AnalyticAtten method*), 117
__new__() (*lightning.attenuation.base.TabulatedAtten method*), 117
__new__() (*lightning.priors.ConstantPrior method*), 163
__new__() (*lightning.priors.NormalPrior method*), 162
__new__() (*lightning.priors.UniformPrior method*), 162
__new__() (*lightning.priors.base.AnalyticPrior method*), 160
__new__() (*lightning.priors.base.TabulatedPrior method*), 161
__new__() (*lightning.sfh.DelayedExponentialSFH method*), 119
__new__() (*lightning.sfh.PiecewiseConstSFH method*), 118
__new__() (*lightning.sfh.SingleExponentialSFH method*), 121
__new__() (*lightning.sfh.base.FunctionalSFH method*), 122

A

add() (*lightning.plots.ModelBand method*), 163
AGNModel (*class in lightning.agn*), 144
AGNPlaw (*class in lightning.xray*), 150
AnalyticAtten (*class in lightning.attenuation.base*), 117
AnalyticPrior (*class in lightning.priors.base*), 160

B

BPASSBurstA24 (*class in lightning.stellar*), 139
BPASSModel (*class in lightning.stellar*), 132
BPASSModelA24 (*class in lightning.stellar*), 135

C

CalzettiAtten (*class in lightning.attenuation*), 116
chain_plot() (*in module lightning.plots*), 164
chain_plot() (*lightning.Lightning method*), 109
ConstantPrior (*class in lightning.priors*), 162
corner_plot() (*in module lightning.plots*), 164
corner_plot() (*lightning.Lightning method*), 109

D

DelayedExponentialSFH (*class in lightning.sfh*), 119
DL07Dust (*class in lightning.dust*), 141

E

evaluate() (*lightning.attenuation.CalzettiAtten method*), 116
evaluate() (*lightning.attenuation.ModifiedCalzettiAtten method*), 115

evaluate() (*lightning.attenuation.SMC method*), 117
evaluate() (*lightning.priors.base.AnalyticPrior method*), 160
evaluate() (*lightning.priors.base.TabulatedPrior method*), 161
evaluate() (*lightning.sfh.base.FunctionalSFH method*), 122
evaluate() (*lightning.sfh.DelayedExponentialSFH method*), 119
evaluate() (*lightning.sfh.PiecewiseConstSFH method*), 118
evaluate() (*lightning.sfh.SingleExponentialSFH method*), 121
evaluate() (*lightning.xray.Phabs method*), 158
evaluate() (*lightning.xray.Tbabs method*), 157

F

fit() (*lightning.Lightning method*), 109
flux_obs (*lightning.Lightning property*), 110
flux_unc (*lightning.Lightning property*), 110
from_json() (*lightning.Lightning static method*), 110
FunctionalSFH (*class in lightning.sfh.base*), 122

G

get_AV() (*lightning.attenuation.CalzettiAtten method*), 116
get_AV() (*lightning.attenuation.ModifiedCalzettiAtten method*), 116
get_mcmc_chains() (*lightning.Lightning method*), 110
get_model_components_lnu_hires() (*lightning.Lightning method*), 111
get_model_countrate() (*lightning.xray.AGNPlaw method*), 151
get_model_countrate() (*lightning.xray.base.XrayEmissionModel method*), 160
get_model_countrate() (*lightning.xray.Qsosed method*), 155
get_model_countrate() (*lightning.xray.StellarPlaw method*), 147
get_model_countrate() (*lightning.xray.XrayPlawExpcut method*), 159
get_model_countrate_hires() (*lightning.xray.AGNPlaw method*), 152
get_model_countrate_hires() (*lightning.xray.base.XrayEmissionModel method*), 160
get_model_countrate_hires() (*lightning.xray.Qsosed method*), 155
get_model_countrate_hires() (*lightning.xray.StellarPlaw method*), 148
get_model_countrate_hires() (*lightning.xray.XrayPlawExpcut method*), 159
get_model_counts() (*lightning.xray.AGNPlaw method*), 152
get_model_counts() (*lightning.xray.base.XrayEmissionModel method*), 160
get_model_counts() (*lightning.xray.Qsosed method*), 156
get_model_counts() (*lightning.xray.StellarPlaw method*), 149
get_model_counts() (*lightning.xray.XrayPlawExpcut method*), 159
get_model_L2500() (*lightning.xray.Qsosed method*), 155
get_model_likelihood() (*lightning.Lightning method*), 111
get_model_lines() (*lightning.Lightning method*), 111
get_model_lines() (*lightning.stellar.BPASSBurstA24 method*), 140
get_model_lines() (*lightning.stellar.BPASSModel method*), 133
get_model_lines() (*lightning.stellar.BPASSModelA24 method*), 137
get_model_lines() (*lightning.stellar.PEGASEBurstA24 method*), 131
get_model_lines() (*lightning.stellar.PEGASEModel method*), 124
get_model_lines() (*lightning.stellar.PEGASEModelA24 method*), 128
get_model_lnu() (*lightning.agn.AGNModel method*), 145
get_model_lnu() (*lightning.dust.DL07Dust method*), 142
get_model_lnu() (*lightning.dust.Graybody method*), 143
get_model_lnu() (*lightning.Lightning method*), 112
get_model_lnu() (*lightning.stellar.BPASSBurstA24 method*), 140
get_model_lnu() (*lightning.stellar.BPASSModel method*), 134
get_model_lnu() (*lightning.stellar.BPASSModelA24 method*), 138
get_model_lnu() (*lightning.stellar.PEGASEBurstA24 method*), 131
get_model_lnu() (*lightning.stellar.PEGASEModel method*), 125
get_model_lnu() (*lightning.stellar.PEGASEModelA24 method*), 129
get_model_lnu() (*lightning.xray.AGNPlaw method*), 153
get_model_lnu() (*lightning.xray.Qsosed method*), 156
get_model_lnu() (*lightning.xray.StellarPlaw method*), 149

get_model_lnu() (*lightning.xray.XrayPlawExpcut method*), 159
get_model_lnu_hires() (*lightning.agn.AGNModel method*), 145
get_model_lnu_hires() (*lightning.dust.DL07Dust method*), 142
get_model_lnu_hires() (*lightning.dust.Graybody method*), 143
get_model_lnu_hires() (*lightning.Lightning method*), 112
get_model_lnu_hires() (*lightning.stellar.BPASSBurstA24 method*), 140
get_model_lnu_hires() (*lightning.stellar.BPASSModel method*), 134
get_model_lnu_hires() (*lightning.stellar.BPASSModelA24 method*), 138
get_model_lnu_hires() (*lightning.stellar.PEGASEBurstA24 method*), 131
get_model_lnu_hires() (*lightning.stellar.PEGASEModel method*), 125
get_model_lnu_hires() (*lightning.stellar.PEGASEModelA24 method*), 129
get_model_lnu_hires() (*lightning.xray.AGNPlaw method*), 153
get_model_lnu_hires() (*lightning.xray.Qsosed method*), 156
get_model_lnu_hires() (*lightning.xray.StellarPlaw method*), 150
get_model_lnu_hires() (*lightning.xray.XrayPlawExpcut method*), 159
get_model_log_prob() (*lightning.Lightning method*), 113
get_mstar_coeff() (*lightning.stellar.BPASSModel method*), 135
get_mstar_coeff() (*lightning.stellar.BPASSModelA24 method*), 139
get_mstar_coeff() (*lightning.stellar.PEGASEModel method*), 126
get_mstar_coeff() (*lightning.stellar.PEGASEModelA24 method*), 130
get_xray_model_counts() (*lightning.Lightning method*), 113
get_xray_model_lnu() (*lightning.Lightning method*), 113
get_xray_model_lnu_hires() (*lightning.Lightning method*), 114
Graybody (*class in lightning.dust*), 143

I

integrate() (*lightning.sfh.base.FunctionalSFH method*), 122
integrate() (*lightning.sfh.DelayedExponentialSFH method*), 120
integrate() (*lightning.sfh.SingleExponentialSFH method*), 121

L

Lightning (*class in lightning*), 105
lightning.plots
    module, 163
lightning.postprocessing
    module, 170
lightning.ppc
    module, 172
line() (*lightning.plots.ModelBand method*), 164
line_flux (*lightning.Lightning property*), 114
line_flux_unc (*lightning.Lightning property*), 114

M

ModelBand (*class in lightning.plots*), 163
ModifiedCalzettiAtten (*class in lightning.attenuation*), 115
module
    lightning.plots, 163
    lightning.postprocessing, 170
    lightning.ppc, 172
multiply() (*lightning.sfh.base.FunctionalSFH method*), 122
multiply() (*lightning.sfh.DelayedExponentialSFH method*), 120
multiply() (*lightning.sfh.PiecewiseConstSFH method*), 118
multiply() (*lightning.sfh.SingleExponentialSFH method*), 121

N

NormalPrior (*class in lightning.priors*), 162

P

PEGASEBurstA24 (*class in lightning.stellar*), 130
PEGASEModel (*class in lightning.stellar*), 123
PEGASEModelA24 (*class in lightning.stellar*), 126
Phabs (*class in lightning.xray*), 157
PiecewiseConstSFH (*class in lightning.sfh*), 118
postprocess_catalog() (*in module lightning.postprocessing*), 170
ppc() (*in module lightning.ppc*), 172
ppc_sed() (*in module lightning.ppc*), 173
print_params() (*lightning.Lightning method*), 114

Q

Qsosed (*class in lightning.xray*), 154
quantile() (*lightning.priors.base.AnalyticPrior method*), 160

quantile() (*lightning.priors.base.TabulatedPrior method*), 161

R

realizations() (*lightning.plots.ModelBand method*), 164

S

sample() (*lightning.priors.base.AnalyticPrior method*), 160
sample() (*lightning.priors.base.TabulatedPrior method*), 161
save_json() (*lightning.Lightning method*), 114
save_pickle() (*lightning.Lightning method*), 114
sed_plot_bestfit() (*in module lightning.plots*), 165
sed_plot_bestfit() (*lightning.Lightning method*), 115
sed_plot_delchi() (*in module lightning.plots*), 166
sed_plot_delchi() (*lightning.Lightning method*), 115
sed_plot_morebayesian() (*in module lightning.plots*), 167
sfh_plot() (*in module lightning.plots*), 169
sfh_plot() (*lightning.Lightning method*), 115
shade() (*lightning.plots.ModelBand method*), 164
SingleExponentialSFH (*class in lightning.sfh*), 120
SMC (*class in lightning.attenuation*), 117
StellarPlaw (*class in lightning.xray*), 146
step_curve() (*in module lightning.plots*), 170
sum() (*lightning.sfh.PiecewiseConstSFH method*), 119

T

TabulatedAtten (*class in lightning.attenuation.base*), 117
TabulatedPrior (*class in lightning.priors.base*), 161
Tbabs (*class in lightning.xray*), 157

U

UniformPrior (*class in lightning.priors*), 162

X

XrayEmissionModel (*class in lightning.xray.base*), 159
XrayPlawExpcut (*class in lightning.xray*), 158