multilightning

Release v2024.1

Erik B. Monson

Aug 16, 2024

DOCUMENTATION

`multilightning` provides an object oriented interface to jointly fit spectral energy distributions of multiple regions of a galaxy, using the python version of the `Lightning` SED-fitting code.

For nearby galaxies, we often have a subsets of bands in which the galaxy is resolved and unresolved (e.g., the far-IR). We can thus choose to either discard the spatial information of the resolved bands and fit the integrated light of the galaxy, or fit without the unresolved data, potentially losing valuable constraints on the star formation rate.

Here, we define a joint likelihood for fitting multiple resolved regions, under the assumption that the unresolved photometry is described by the sum of the models for each resolved region. By way of example, this method is applied to the nearby galaxy NGC 628, jointly fitting the galactic disk and a small, crowded region at its center. This is a very simple case, however, as the center region is small and contributes little to the overall IR SED.

This method could be extended to:

- Larger numbers of regions, to provide a computationally cheaper option to computing physical property maps.
- Separately modeling the nucleus and disk in nearby AGN.
- Determining the ages of individual star formation regions which are not separated in IR imaging?

CHAPTER

ONE

BACKGROUND

The original motivation for `multilightning` is fairly simple:

- The sub-arcsecond to arcsecond-scale resolution of UV-optical imaging of nearby ($\lesssim 40$ Mpc) galaxies contains detailed information about the spatial variation of star formation and dust obscuration.
- The comparatively worse $\gtrsim 10 - 20$ arcsecond-scale resolution of IR imaging loses much of that spatial information, but IR data provide a valuable handle on the overall normalization of the star formation history (SFH), and the inclusion of IR data in spectral energy distribution (SED) fits can allow us to use more complicated dust models.

We must therefore balance the need for spatial information with the need for an IR constraint on the SFH.

In the case of Lehmer et al. (2024), we required good constraints on the SFH over only the disks of some galaxies, excluding the nuclear regions: the nuclear regions of star-forming galaxies suffer from crowding of X-ray sources, such that individual sources can't be counted. However, these galaxies are not resolved in the available far-IR imaging. Thus, instead of attempting to fit the SED and estimate the SFH using only the subset of fluxes that resolve the disk, we forward model all the available data, using the IR data as a constraint on the SFH of both the disk and the nuclear region.

We have three sets of fluxes:

- Two sets of $N_{\rm res}$ fluxes, which resolve the two regions, $\{F_{\nu,i}^{\rm disk}\}_{i=1}^{N_{\rm res}}$ and $\{F_{\nu,i}^{\rm center}\}_{i=1}^{N_{\rm res}}$
- One set of $N_{\rm unres}$ fluxes which cover the entire galaxy, $\{F_{\nu,j}^{\rm total}\}_{j=1}^{N_{\rm unres}}$.

Each underlying region is modeled with a separate star formation history. We then have three χ^2 terms:

$$\chi^2_{\rm disk} = \sum_{i=1}^{N_{\rm res}} \frac{(F_{\nu,i}^{\rm disk} - \hat{F}_{\nu,i}^{\rm disk})^2}{(\sigma F_{\nu,i}^{\rm disk})^2}$$

$$\chi^2_{\rm center} = \sum_{i=1}^{N_{\rm res}} \frac{(F_{\nu,i}^{\rm center} - \hat{F}_{\nu,i}^{\rm center})^2}{(\sigma F_{\nu,i}^{\rm center})^2}$$

$$\chi^2_{\rm total} = \sum_{j=1}^{N_{\rm unres}} \frac{(F_{\nu,j}^{\rm total} - \hat{F}_{\nu,j}^{\rm total})^2}{(\sigma F_{\nu,j}^{\rm total})^2}$$

where

$$\hat{F}_{\nu,j}^{\rm total} = \hat{F}_{\nu,j}^{\rm disk} + \hat{F}_{\nu,j}^{\rm center}.$$

We then have

$$\chi^2 = \chi^2_{\rm disk} + \chi^2_{\rm center} + \chi^2_{\rm total}$$

In *the included Jupyter notebook*, we demonstrate this in practice for NGC 628.

CHAPTER

TWO

EXAMPLE: TWO REGION FIT IN NGC 628

This notebook outlines a similar procedure as followed in Lehmer et al. (2024) to fit the SFHs of galaxies in two regions, using NGC 628 as an example.

Briefly, the central regions of spiral galaxies suffer from crowding in Chandra imaging, preventing us from counting individual X-ray, such that the X-ray binary luminosity function can only be constructed outside the central region. In Lehmer et al. (2024), where our goal was to connect the XLF to the SFH of the galaxy, this meant that we needed to exclude the center of many galaxies from the estimation of the SFH. However, deriving the SFH by fitting only the SED of the outer region would force us to ignore IR data which does not resolve the separate regions. To preserve the valuable IR constraints on the SFH, we hacked together this method for jointly fitting the central and outer regions of the galaxy, such that the sum of the two SED models is constrained by the IR measurements.

2.1 Imports

[1]:
```
import h5py
import sys
from pprint import pprint
import numpy as np
from astropy.table import Table

from lightning import Lightning
from lightning.priors import UniformPrior
from multi_lightning import MultiLightning
from multi_lightning.priors import FixedConnection, NormalConnection, UniformConnection

import corner
import matplotlib.pyplot as plt

%matplotlib inline
```

2.1.1 Read data and construct flux arrays

[2]:
```
phot = Table.read('NGC0628-photometry.fits')

filter_labels = np.array([s.strip() for s in phot['FILTER_LABELS']])
filter_labels[filter_labels == ['2MASS_K']] = '2MASS_Ks'

Nfilters = len(filter_labels)

#ascii.write(phot['FILTER_LABELS','WAVELENGTH'], format='fixed_width_two_line')

fnu = phot['FNU_OBS'] * 1e3 # in mJy
fnu_cent = phot['FNU_CENT_OBS'] * 1e3 # in mJy
# By the construction of this catalog, the annulus photometry is the difference of the␣
↪total and center regions.
fnu_out = fnu - fnu_cent


# For this example we assume that the uncertainty in the flux is
# dominated by the absolute flux calibration, and we ignore the
# background.
cal_unc = np.array([0.15, 0.15, # GALEX
                    0.05, 0.05, 0.05, 0.05, 0.05, # SDSS
                    0.05, 0.05, 0.05, # 2MASS
                    0.05, 0.05, 0.05, 0.05, # IRAC
                    0.05, # MIPS 24 um
                    0.05, 0.05, 0.05, # PACS
                    0.15, 0.15, 0.15, # SPIRE
                    0.07, 0.07, 0.07, 0.07 # WISE
                    ])

fnu_unc = cal_unc * fnu
fnu_out_unc = cal_unc * fnu_out
fnu_cent_unc = cal_unc * fnu_cent

fnu_unc = cal_unc * fnu
fnu_out_unc = cal_unc * fnu_out
fnu_cent_unc = cal_unc * fnu_cent

unresolved = (fnu != 0) & (fnu_cent == 0)
resolved = ~unresolved

fnu[fnu == 0] = np.nan
fnu_out[fnu_cent == 0] = np.nan
fnu_cent[fnu_cent == 0] = np.nan

# The flux array has Nregions+1 rows, where the extra (first) row
# corresponds to the "total" fluxes, from the bands that
# do not resolve the regions.
fnu_arr = np.zeros((3, Nfilters))
fnu_arr[1,:] = fnu_cent
fnu_arr[2,:] = fnu_out
fnu_arr[0,unresolved] = fnu[unresolved]
```

(continues on next page)

(continued from previous page)

```
fnu_arr[0,resolved] = np.nan

# Likewise for the uncertainty array.
fnu_unc_arr = np.zeros((3, Nfilters))
fnu_unc_arr[1,:] = fnu_cent_unc
fnu_unc_arr[2,:] = fnu_out_unc
fnu_unc_arr[0,unresolved] = fnu_unc[unresolved]
fnu_unc_arr[0,resolved] = 0.0
```

2.2 Initialize the Lightning and MultiLightning Objects

Things to notice: - The first positional argument for `MultiLightning` is itself a `Lightning` object or list of objects defining the models to use for each region. - Here, we only need to define one `Lightning` object, since we use the same model for both regions. In some scenarios you can imagine having different models, for example if you need to include an AGN component in the nuclear region.

[3]:
```
reg_names = ['center', 'disk']

redshift = 0.0
DL = 7.3 # Mpc - collected from Moustakis+2010, originally measured by Sharina+(1996)␣
↪from supergiant stars.

lgh = Lightning(filter_labels,
                lum_dist=DL,
                atten_type='Calzetti',
                dust_emission=True)

# lgh.save_json('NGC0628_config.json')


mlgh = MultiLightning(lgh,
                      fnu_arr,
                      fnu_unc_arr,
                      model_unc=0.10,
                      Nregions=2,
                      reg_names=reg_names)

print('Priors are still given as ordered lists, so it is useful to use `print_params` to␣
↪see what that order should be.')
lgh.print_params(verbose=True)
```

```
Priors are still given as ordered lists, so it is useful to use `print_params` to see␣
↪what that order should be.

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

(continues on next page)

(continued from previous page)

```
    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

============================
Pegase-Stellar
============================
Parameter    Lo  Hi                                          Description
--------- ----- --- -------------------------------------------------
     Zmet 0.001 0.1 Metallicity (mass fraction, where solar = 0.020)

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

============================
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: 12
```

2.3 Fit the model

[4]:
```
# This is no longer a mean for the initialization now that
# we initialize from the priors. However, it's currently
# still required in order to handle any constant
# dimensions -- MultiLightning doesn't understand the ConstantPrior in Lightning
# yet.
p = {'center': np.array([0.1,0.1,0.1,0.1,0.1,
                         0.02,
                         0.1,
                         2,1,3e5,0.01,0.02]).reshape(1,-1),
```

(continues on next page)

(continued from previous page)

```
    'disk': np.array([1,1,1,1,1,
                      0.02,
                      0.1,
                      2,1,3e5,0.01,0.02]).reshape(1,-1)
    }

# We can make the prior process a little less obnoxious with
# list arithmetic.
# The prior `FixedConnection` takes two arguments: a region name and a parameter index.␣
↪Here it fixes the
# qPAH of the center region to the qPAH of the disk.
# priors_cen = 5 * [UniformPrior([0,1])] + \
#                  [UniformPrior([0,3])] + \
#                  [None, UniformPrior([0.1, 25]), None, UniformPrior([0,1]),␣
↪FixedConnection('disk', -1)]
# The prior `NormalConnection` takes three arguments: a [mu, sigma] iterable, a region␣
↪name, and a parameter index.
# Here it loosely fixes the qPAH of the center region to the qPAH of the disk.
priors_cen = 5 * [UniformPrior([0,1])] + \
                 [None] + \
                 [UniformPrior([0,3])] + \
                 [None, UniformPrior([0.1, 25]), None, UniformPrior([0,1]),␣
↪NormalConnection([0,0.01], 'disk', -1)]
priors_disk = 5 * [UniformPrior([0,10])] + \
                  [None] + \
                  [UniformPrior([0,3])] + \
                  [None, UniformPrior([0.1, 25]), None, UniformPrior([0,1]),␣
↪UniformPrior([0.0047, 0.0458])]

priors = {'center': priors_cen,
          'disk': priors_disk
         }
mcmc = mlgh.fit(p,
                priors,
                progress=True)
```

```
/Users/eqm5663/Research/code/plightning/lightning/stellar/pegase.py:443: RuntimeWarning:␣
↪divide by zero encountered in log10
  finterp = interp1d(self.Zmet, np.log10(self.Lnu_obs), axis=1)
/Users/eqm5663/miniconda3_arm64/envs/ciao-4.16/lib/python3.11/site-packages/scipy/
↪interpolate/_interpolate.py:701: RuntimeWarning: invalid value encountered in subtract
  slope = (y_hi - y_lo) / (x_hi - x_lo)[:, None]
100%|##########################################################################################
↪######################| 30000/30000 [18:47<00:00, 26.60it/s]
```

construct the final chain:

[5]:
```
try:
    tau = mcmc.get_autocorr_time()
    print('tau = ', tau)
except:
    print('Chains too short to properly estimate autocorrelation time.')
    print('Run a longer chain. Using tau = 500. Inspect products carefully...')
```

(continues on next page)

(continued from previous page)

```
    tau = 500

burnin = int(4 * np.max(tau))
thin = int(1 * np.min(tau))

samples = mcmc.get_chain(discard=burnin, flat=True, thin=thin)[-1000:,:]
log_prob_samples = mcmc.get_log_prob(discard=burnin, flat=True, thin=thin)[-1000:]

# Put the final chains (including constant parameters) in an hdf5 file.
Nsamples_final = samples.shape[0]
Nparams_cen = len(p['center'].flatten())
Nparams_disk = len(p['disk'].flatten())

params = mlgh._params_vec2dict(samples, p, priors)
```

```
Chains too short to properly estimate autocorrelation time.
Run a longer chain. Using tau = 500. Inspect products carefully...
```

2.4 Plot some of the results

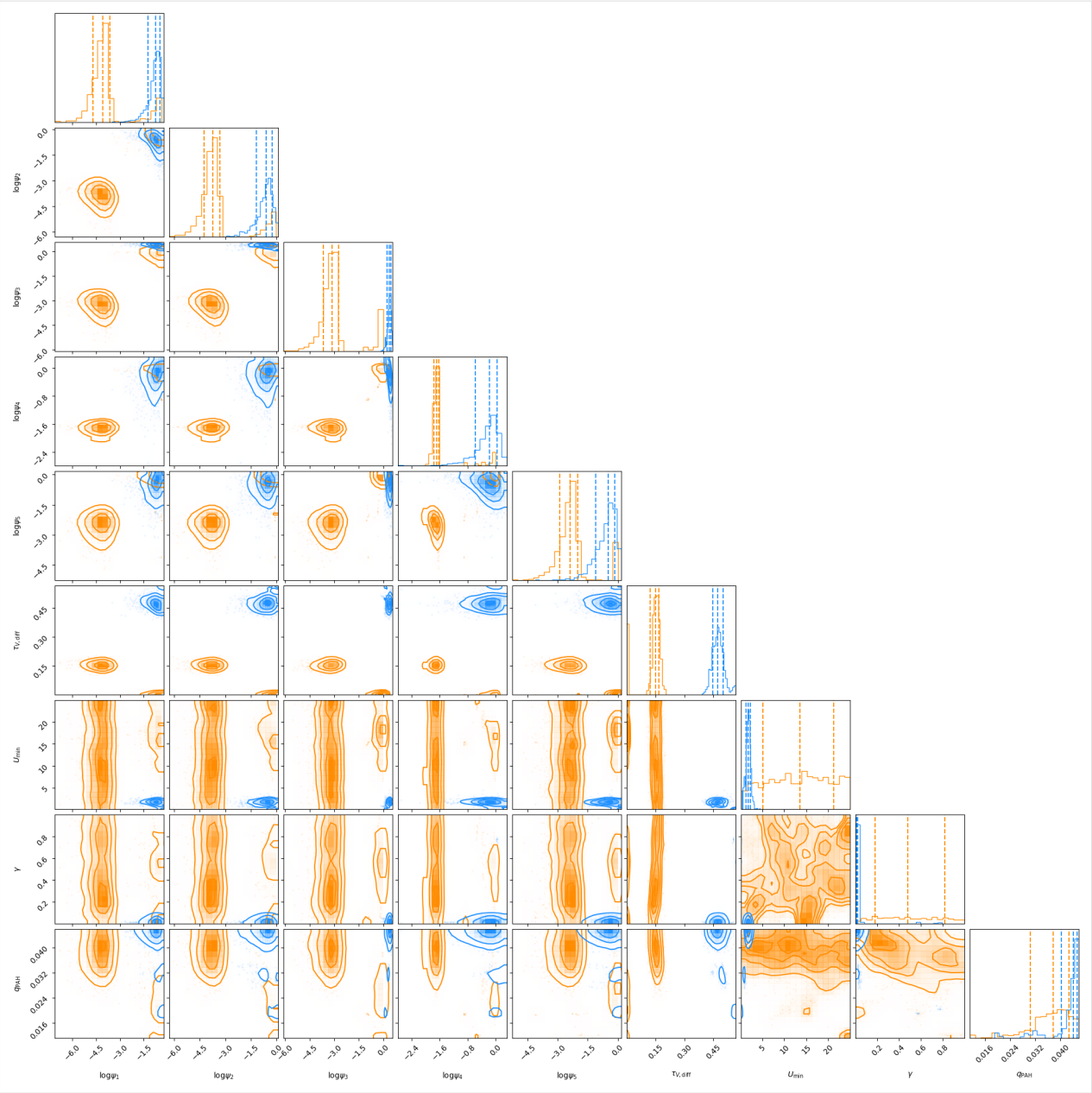
[6]:
```
param_labels = []
for mod in lgh.model_components:
    if mod is not None:
        param_labels = param_labels + mod.param_names_fncy

param_labels_log = ['log' + l if 'psi' in l else l for l in param_labels]
param_labels_log = np.array(param_labels_log)
param_labels = np.array(param_labels)

samples_center = params['center']
samples_disk = params['disk']
const_dim = np.var(samples_center, axis=0) < 1e-10
```

[7]:
```
sfh_center = samples_center[:,:5]
sfh_disk = samples_disk[:,:5]

# mstar_center = np.sum(lgh.stars.mstar[None,:] * sfh_center, axis=1)
# mstar_disk = np.sum(lgh.stars.mstar[None,:] * sfh_disk, axis=1)

sfh_center_lo, sfh_center_med, sfh_center_hi = np.quantile(sfh_center, q=(0.16, 0.50, 0.
↪84), axis=0)
sfh_disk_lo, sfh_disk_med, sfh_disk_hi = np.quantile(sfh_disk, q=(0.16, 0.50, 0.84),␣
↪axis=0)
# mstar_center_lo, mstar_center_med, mstar_center_hi = np.quantile(mstar_center, q=(0.16,
↪ 0.50, 0.84), axis=0)
# mstar_disk_lo, mstar_disk_med, mstar_disk_hi = np.quantile(mstar_disk, q=(0.16, 0.50,␣
↪0.84), axis=0)

# Plot log SFH so that we can fit both
# regions on the same corner plot
```

(continues on next page)

(continued from previous page)

```
tmp = samples_center[:,~const_dim]
tmp[:,:5] = np.log10(tmp[:,:5])
fig1 = corner.corner(tmp,
                     labels=param_labels_log[~const_dim],
                     quantiles=[0.16, 0.50, 0.84],
                     #show_titles=True,
                     smooth=1,
                     color='darkorange')
tmp = samples_disk[:,~const_dim]
tmp[:,:5] = np.log10(tmp[:,:5])
fig1 = corner.corner(tmp,
                     labels=param_labels_log[~const_dim],
                     quantiles=[0.16, 0.50, 0.84],
                     #show_titles=True,
                     smooth=1,
                     fig=fig1,
                     color='dodgerblue')
```



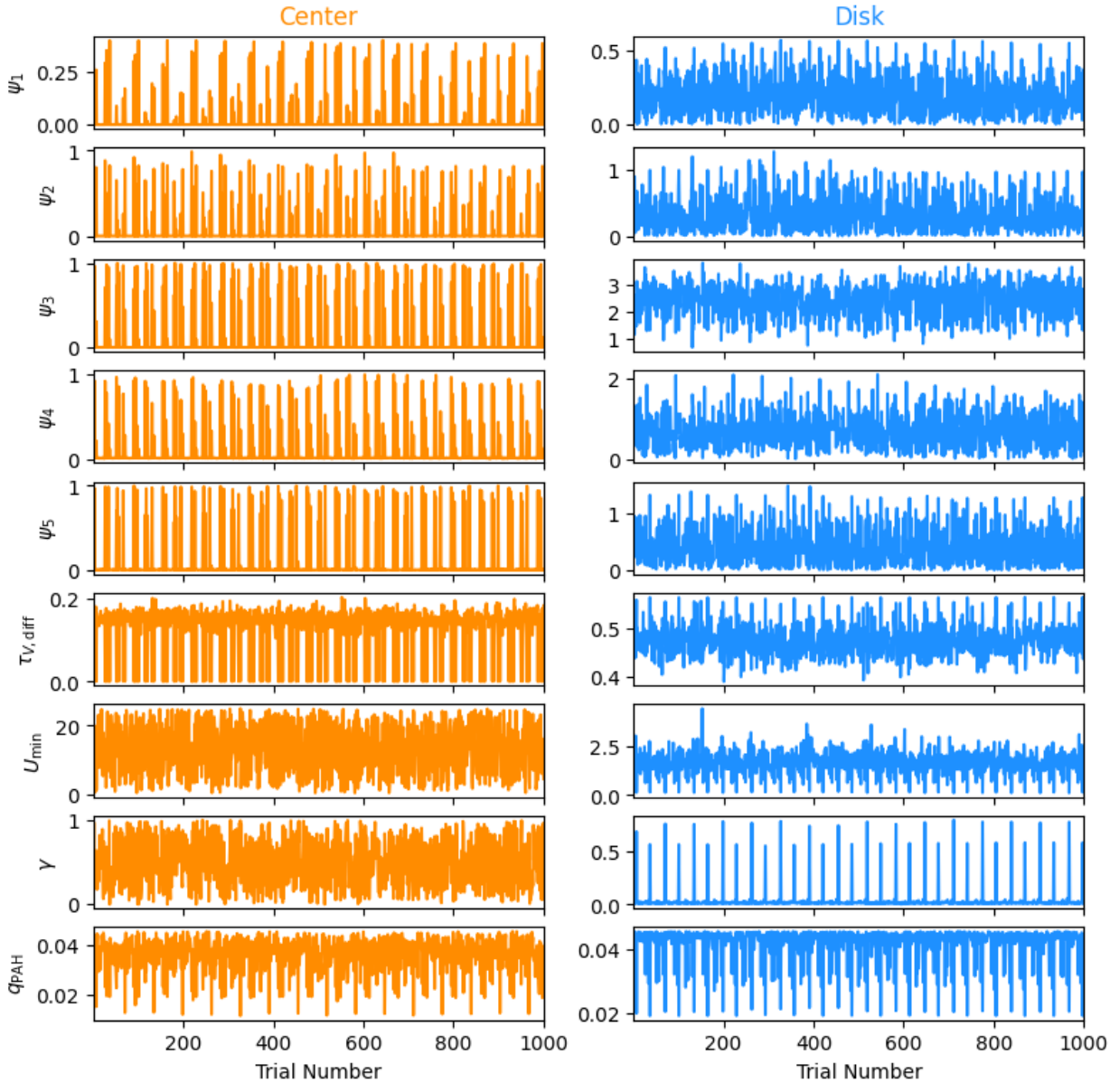
[8]:
```
fig3, axs = plt.subplots(9,2, figsize=(9,9))

t = 1 + np.arange(samples_center.shape[0])
for i, label in enumerate(param_labels[~const_dim]):
    axs[i,0].plot(t, samples_center[:,~const_dim][:,i], color='darkorange')
    axs[i,1].plot(t, samples_disk[:,~const_dim][:,i], color='dodgerblue')

    axs[i,0].set_ylabel(label)
    if i != 8:
        axs[i,0].set_xticklabels([])
        axs[i,1].set_xticklabels([])
```

(continues on next page)

(continued from previous page)

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

axs[0,0].set_title('Center', color='darkorange')
axs[0,1].set_title('Disk', color='dodgerblue')
axs[8,0].set_xlabel('Trial Number')
axs[8,1].set_xlabel('Trial Number')
```

[8]:
```
Text(0.5, 0, 'Trial Number')
```



Clearly still some correlated behavior. We would need longer chains to properly converge and get a better estimate of the autocorrelation time.

[10]:
```
# Best-fit SED plot
fig4 = plt.figure()
ax41 = fig4.add_axes([0.1, 0.4, 0.8, 0.5])
ax42 = fig4.add_axes([0.1, 0.1, 0.8, 0.3])

# samples
# log_prob_samples

# mlgh.lnu_obs[1,:]
# mlgh.lnu_unc[1,:]
# mlgh.lnu_obs[2,:]
# mlgh.lnu_unc[2,:]
# mlgh.lnu_obs[0,:]
# mlgh.lnu_unc[0,:]

bestfit = np.argmax(log_prob_samples)
disk_lnu_best = lgh.get_model_components_lnu_hires(samples_disk[bestfit,:])
center_lnu_best = lgh.get_model_components_lnu_hires(samples_center[bestfit,:])
disk_lnu_best_total,_ = lgh.get_model_lnu_hires(samples_disk[bestfit,:])
center_lnu_best_total,_ = lgh.get_model_lnu_hires(samples_center[bestfit,:])
disk_lmod_best_total,_ = lgh.get_model_lnu(samples_disk[bestfit,:])
center_lmod_best_total,_ = lgh.get_model_lnu(samples_center[bestfit,:])
# total_total
total_lmod_best_total = disk_lmod_best_total + center_lmod_best_total

delchi_disk = (mlgh.lnu_obs[2,:] - disk_lmod_best_total) / np.sqrt(mlgh.lnu_unc[2,:]**2␣
↪+ (0.10 * disk_lmod_best_total)**2)
delchi_center = (mlgh.lnu_obs[1,:] - center_lmod_best_total) / np.sqrt(mlgh.lnu_unc[1,:
↪]**2 + (0.10 * center_lmod_best_total)**2)
delchi_unresolved = (mlgh.lnu_obs[0,:] - total_lmod_best_total) / np.sqrt(mlgh.lnu_unc[0,
↪:]**2 + (0.10 * center_lmod_best_total)**2)

ax41.plot(lgh.wave_grid_obs,
          lgh.nu_grid_obs*disk_lnu_best_total,
          color='dodgerblue',
          label='Disk',
          alpha=0.5)
ax41.plot(lgh.wave_grid_obs,
          lgh.nu_grid_obs*center_lnu_best_total,
          color='darkorange',
          label='Center',
          alpha=0.5)
ax41.plot(lgh.wave_grid_obs,
          lgh.nu_grid_obs*(center_lnu_best_total + disk_lnu_best_total),
          color='black',
          label='Total',
          alpha=0.5)
ax41.errorbar(lgh.wave_obs,
              lgh.nu_obs * mlgh.lnu_obs[2,:],
              yerr=lgh.nu_obs * mlgh.lnu_unc[2,:],
              marker='D',
              linestyle='',
              color='dodgerblue',
```

(continues on next page)

(continued from previous page)

```
              markerfacecolor='dodgerblue',
              capsize=2)
ax41.errorbar(lgh.wave_obs,
              lgh.nu_obs * mlgh.lnu_obs[1,:],
              yerr=lgh.nu_obs * mlgh.lnu_unc[1,:],
              marker='D',
              linestyle='',
              color='darkorange',
              markerfacecolor='darkorange',
              capsize=2)
ax41.errorbar(lgh.wave_obs,
              lgh.nu_obs * mlgh.lnu_obs[0,:],
              yerr=lgh.nu_obs * mlgh.lnu_unc[0,:],
              marker='D',
              linestyle='',
              color='k',
              markerfacecolor='k',
              capsize=2)

ax41.set_xscale('log')
ax41.set_yscale('log')
ax41.set_ylabel(r'$\nu L_{\nu}~[\rm L_{\odot}]$')
ax41.legend(loc='lower left')

ax42.axhline(-1, color='slategray', linestyle='--')
ax42.axhline(0, color='slategray', linestyle='-')
ax42.axhline(1, color='slategray', linestyle='--')

ax42.errorbar(lgh.wave_obs,
              delchi_disk,
              yerr=np.ones_like(delchi_disk),
              marker='D',
              color='dodgerblue',
              markerfacecolor='dodgerblue',
              linestyle='',
              capsize=2.0)
ax42.errorbar(lgh.wave_obs,
              delchi_center,
              yerr=np.ones_like(delchi_center),
              marker='D',
              color='darkorange',
              markerfacecolor='darkorange',
              linestyle='',
              capsize=2.0)
ax42.errorbar(lgh.wave_obs,
              delchi_unresolved,
              yerr=np.ones_like(delchi_unresolved),
              marker='D',
              color='k',
              markerfacecolor='k',
              linestyle='',
              capsize=2.0)
```

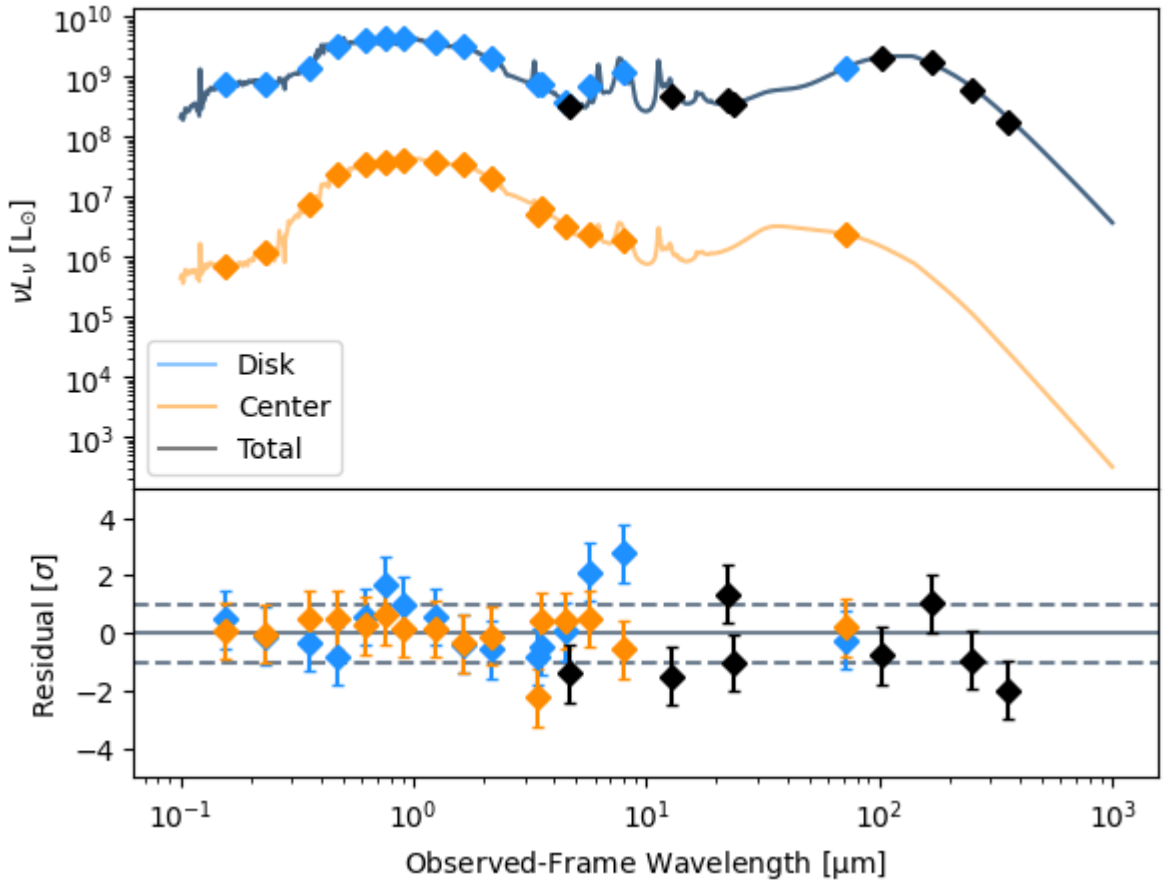
(continues on next page)

(continued from previous page)

```
ax42.set_xscale('log')
ax42.set_xlim(ax41.get_xlim())
ax42.set_xlabel(r'Observed-Frame Wavelength [$\rm \mu m$]')
ax42.set_ylim(-5,5)
ax42.set_ylabel(r'Residual [$\sigma$]')
```

```
/Users/eqm5663/Research/code/plightning/lightning/stellar/pegase.py:443: RuntimeWarning:␣
↪divide by zero encountered in log10
  finterp = interp1d(self.Zmet, np.log10(self.Lnu_obs), axis=1)
/Users/eqm5663/miniconda3_arm64/envs/ciao-4.16/lib/python3.11/site-packages/scipy/
↪interpolate/_interpolate.py:701: RuntimeWarning: invalid value encountered in subtract
  slope = (y_hi - y_lo) / (x_hi - x_lo)[:, None]
```

[10]:
```
Text(0, 0.5, 'Residual [$\\sigma$]')
```



We recover a pretty decent fit, all said.

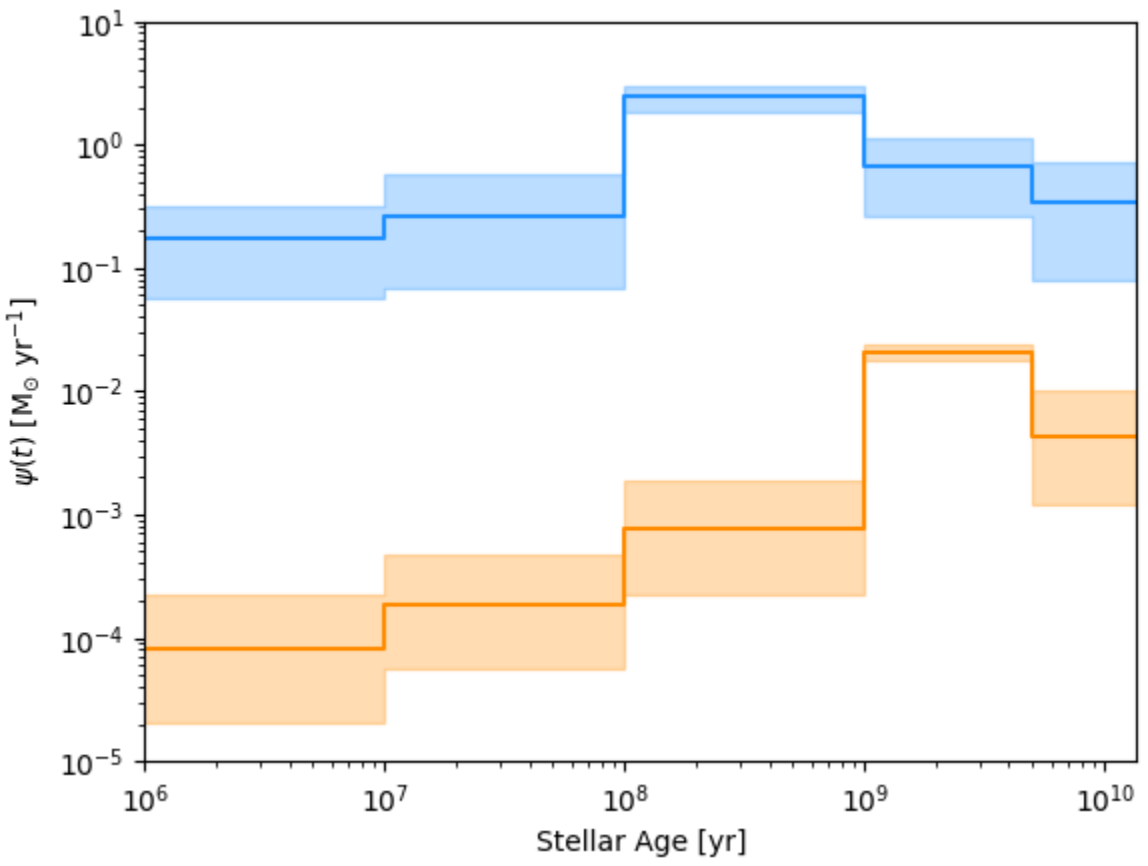
[11]:
```
# SFH plot -- Could also normalize the SFH somehow to better show the
# differences in the shapes.
fig5, ax5 = plt.subplots()
fig5, ax5 = lgh.sfh_plot(samples_center,
                         shade_kwargs={'color':'darkorange', 'alpha':0.3, 'zorder':0},
                         line_kwargs={'color':'darkorange', 'zorder':1},
                         ax=ax5)
```

(continues on next page)

(continued from previous page)

```
fig5, ax5 = lgh.sfh_plot(samples_disk,
                         shade_kwargs={'color':'dodgerblue', 'alpha':0.3, 'zorder':0},
                         line_kwargs={'color':'dodgerblue', 'zorder':1},
                         ax=ax5)

ax5.set_ylabel(r'$\psi (t)~[\rm M_{\odot}~yr^{-1}]$')
ax5.set_xlabel(r'Stellar Age [yr]')
ax5.set_xscale('log')
ax5.set_yscale('log')
ax5.set_xlim(1e6,13.6e9)
ax5.set_ylim(1e-5, 10)
```

[11]: (1e-05, 10)



CHAPTER

THREE

MULTILIGHTNING

The main interface to `multilightning` is the `MultiLightning` class.

class multi_lightning.**MultiLightning**

A class interface to fit "multi-region" spectral energy distributions: SEDs where multiple regions are resolved in some, but not all of the available bandpasses.

Parameters

lgh
[Lightning object or list of Nregions Lightning objects] If a single Lightning object, the same model will be applied to each region (albeit with different parameters) and the `Nregions` keyword must be set. More generally, this can by a list of Lightning objects, one per region. Things like the redshift, luminosity distance, and filter set are expected to agree between different objects.

flux_obs
[np.ndarray, (Nregions+1, Nfilters), float] An array giving the observed fluxes in mJy: the first row should contain the unresolved "global" fluxes, and subsequent rows should contain the resolved fluxes for each region. Missing or NA entries should be set to NaN. As an example, suppose we have 2 regions with 4 resolved optical measurements and 2 unresolved IR measurements. Our flux array would be structured like:

```
                  Opt1   Opt2   Opt3   Opt4  IR1  IR2
unresolved     [[  NaN,   NaN,   NaN,   NaN, 0.2,   1],
region1         [0.003, 0.002, 0.004, 0.002, NaN, NaN],
region2         [  NaN, 0.001, 0.002, 0.001, NaN, NaN]]
```

where we've supposed that region2 is undetected in band Opt1.

flux_unc
[np.ndarray, (Nregions+1, Nfilters), float] An array giving the uncertainties on the fluxes in mJy. Here, missing and NA entries should be set to 0.0.

model_unc
[float] Fractional model uncertainty to apply to all bands.

Nregions
[int] Number of resolved regions. Only necessary if `lgh` is a single Lightning object.

reg_names
[list of str] Names for individual regions, e.g. 'core', 'disk', 'clumpA', 'clumpB', etc. Defaults to 'reg1', 'reg2', etc. if not set.

Methods

fit(p0, priors[, Nwalkers, Nsteps, ...])	Fit the multi-region model with emcee.
get_model_log_like(params)	Calculate model log-likelihood.
get_model_log_prior(params, priors)	Calculate prior probability of parameters.
get_model_log_prob(params, priors[, p_bound])	Calculate model log probability.
init_from_priors(priors, Nwalkers)	Sample an initial state vector for emcee from the priors.
print_params([verbose])	List parameters (or pretty-print, when verbose=True)

__init__(*lgh*, *flux_obs*, *flux_unc*, *model_unc=None*, *Nregions=None*, *reg_names=None*)

fit(*p0*, *priors*, *Nwalkers=64*, *Nsteps=30000*, *init_sigma=0.001*, *progress=True*, *savefile=None*)

Fit the multi-region model with emcee.

Parameters

priors
[dict] A dictionary keyed on `reg_names`, where each entry is a list of the `Nparamsx`-many prior functions for each region. Prior functions should be specified by lightning.priors objects. Any prior == None is assumed to indicate a constant parameter.

Nwalkers
[int] Number of MCMC samplers for the emcee affine-invariant algorithm (Default: 64)

Nsteps
[int] Number of steps to run the MCMC (default: 30000)

init_sigma
[float] Sigma for gaussian ball initialization (default: 1e-3) *UNUSED*

progress
[bool] If `True`, print a `tqdm` progress bar. (Default: True)

savefile
[str] Filename to use with the `emcee` HDF5 backend (if any). (Default: None)

Returns

sampler
[emcee.EnsembleSampler]

get_model_log_like(*params*)

Calculate model log-likelihood.

Parameters

params
[dict] A dictionary keyed on `reg_names`, where each entry is a numpy array with dimensions (Nmodels, Nparamsx), giving the parameters for each region. Note that in the most general case `Nparamsx` can be different for each region, while the first `Nmodels` axis is the vectorization axis, and should be the same for each region.

Returns

lnlike
[np.ndarray, (Nmodels,)] -0.5 * chi2

`get_model_log_prior`(*params*, *priors*)

Calculate prior probability of parameters.

Parameters

params
[dict] A dictionary keyed on `reg_names`, where each entry is a numpy array with dimensions (Nmodels, Nparamsx), giving the parameters for each region. Note that in the most general case `Nparamsx` can be different for each region, while the first `Nmodels` axis is the vectorization axis, and should be the same for each region.

priors
[dict] A dictionary keyed on `reg_names`, where each entry is a list of the `Nparamsx`-many prior functions for each region. Prior functions should be specified by lightning.priors objects.

Returns

lnprior_prob
[np.ndarray, (Nmodels,)] Prior log-probabilities

`get_model_log_prob`(*params*, *priors*, *p_bound=inf*)

Calculate model log probability.

Parameters

params
[dict] A dictionary keyed on `reg_names`, where each entry is a numpy array with dimensions (Nmodels, Nparamsx), giving the parameters for each region. Note that in the most general case `Nparamsx` can be different for each region, while the first `Nmodels` axis is the vectorization axis, and should be the same for each region.

priors
[dict] A dictionary keyed on `reg_names`, where each entry is a list of the `Nparamsx`-many prior functions for each region. Prior functions should be specified by lightning.priors objects.

p_bound
[float] The magnitude of the log-probability of zero (default: np.inf)

Returns

log_prob
[np.ndarray, (Nmodels,)] Log probabilities

`init_from_priors`(*priors*, *Nwalkers*)

Sample an initial state vector for emcee from the priors.

Parameters

priors
[dict] A dictionary keyed on `reg_names`, where each entry is a list of the `Nparamsx`-many prior functions for each region. Prior functions should be specified by lightning.priors objects. Any prior == None is assumed to indicate a constant parameter.

Nwalkers
[int] Number of MCMC samplers for the emcee affine-invariant algorithm

Returns

x0
[np.ndarray, (Nwalkers, Ndim)] Sampled initial state for emcee

print_params(*verbose=False*)

List parameters (or pretty-print, when verbose=True)

CHAPTER

FOUR

PRIORS

`multilightning` is compatible with the priors defined in `lightning.priors`. However, the addition of multi-region fitting introduces the need for priors which *connect parameters together* between different models. `multilightning` implements three connections: Normal and Uniform connections, which assume the distance and absolute distance in the parameter values are normally and uniformly distributed, respectively; and a fixed connection, which pins the parameter to a reference value.

These are slightly hacked together, which you can tell by the weird way you initialize them. Each prior takes as arguments:

- Its own parameters: μ and σ for the normal connection, a and b for the uniform, nothing for the fixed connection.
- The name of the region to target, as a string.
- The index of the parameter to target.

Therefore `NormalConnection([0.0, 1.0], 'disk', 0)` assumes that the distance between the given parameter and the first parameter (i.e., the parameter at index 0) of the `'disk'` component is drawn from the standard normal distribution.

> **Note**
>
> This construction actually means you can use these priors to link parameters within the same region: the above construction could be used to link the second piecewise-constant SFH coefficient in the disk component to the first, for example.

class multi_lightning.priors.**NormalConnection**

Bases: `AnalyticPrior`

Treats the distance $\delta x = x_1 - x_2$ as a Normal random variable with $\delta x \sim N(\mu, \sigma)$

Parameters

params
[array-like, (2,)] Mean and sigma of the normal distribution.

target_name
[str] Name of region to target

param_idx
[int] Index of parameter in region model.

Methods

evaluate(x1, x2)	Calculate prior probability.
quantile(q)	Calculate quantile function:
sample(size[, rng, seed])	Sample from the prior.

__init__(*params*, *target_name*, *param_idx*)

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

evaluate(*x1*, *x2*)

Calculate prior probability.

Returns an array with the same shape as `x1` that is equal to

$$p = \frac{1}{\sigma\sqrt{2\pi}} \exp[-(\delta x - \mu)^2/\sigma^2)]$$

where

$$\delta x = x_1 - x_2$$

quantile(*q*)

Calculate quantile function:

Return an array with the same shape as `q` that is equal to

$$\delta x = \mu + \sigma\sqrt{2}\text{erfinv}(2q - 1)$$

where

$$\delta x = x_1 - x_2$$

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 multi_lightning.priors.**UniformConnection**

Bases: AnalyticPrior

Treats the absolute distance $dx = |x_1 - x_2|$ as a Uniform random variable with *dx ~ U(0, D_{max})*, i.e., x2 can be at most Dmax from x1.

Parameters

params
[array-like, (2,)] Lower and upper bound of the uniform distribution.

target_name
[str] Name of region to target

param_idx
[int] Index of parameter in region model.

Methods

evaluate(x1, x2)	Calculate prior probability.
quantile(q)	Calculate quantile function.
sample(size[, rng, seed])	Sample from the prior.

__init__(*param*, *target_name*, *param_idx*)

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

evaluate(*x1*, *x2*)

Calculate prior probability.

Return an array with the same shape as x1 that's equal to $1/D_{max}$ wherever dx is in $[a, b)$ and 0 elsewhere.

quantile(*q*)

Calculate quantile function.

Return an array with the same shape as q that's equal to qD_{max}.

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 multi_lightning.priors.**FixedConnection**

Dummy prior. We use this to hold a parameter value fixed to a parameter from another region.

Note that this prior doesn't fully implement `lightning.priors.AnalyticPrior` and can't be used to sample

Parameters

target_name
[str] Name of region to target

param_idx
[int] Index of parameter in region model.

Methods

evaluate(x1, x2)	Returns x1 == x2

__init__(*target_name*, *param_idx*)

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

evaluate(*x1*, *x2*)

Returns x1 == x2

CHAPTER

FIVE

ATTRIBUTION

multilightning was originally used in Lehmer et al. 2024:

```
@article{lehmer2024
         TKTKTKTKTKTK
        }
```

and can be cited using its ASCL identifier:

```
@software{multilightning
          TKTKTKTKTKTK
         }
```

CHAPTER

SIX

LICENSE

`multilightning` is released under the terms of the MIT license.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

E

F

G

I

M

N

P

Q

S

U