dense basis

Release v 0.1.9

Jul 14, 2022

General Usage:

dense_basis is an implementation of the Dense Basis method tailored to SED fitting - in particular, the task of recovering accurate star formation history (SFH) information from galaxy spectral energy distributions (SEDs). The current code is being adapted from its original use-case (simultaneously fitting specific large catalogs of galaxies) to being a general purpose SED fitting code, and acting as a module to compress and decompress SFHs and other time-series.

CHAPTER 1

Installation

The current version of the *dense_basis* module has a few dependencies (see this) that need to be set up before running this package. Once these are met, the package can be installed as follows:

```
git clone https://github.com/kartheikiyer/dense_basis.git
cd dense_basis
python setup.py install
```

The code will default to looking for filter lists in a *filters/* directory, and will build and store atlases in a *pregrids/* directory within the current working directory. If you would like to supply your own paths, please provide either the relative or absolute paths as inputs to the relevant functions using the *filt_dir* or *path* arguments.

CHAPTER 2

Dependencies

- FSPS and python-FSPS (v.0.3.0+): The current implementation of the *dense_basis* method uses a backend based on Flexible Stellar Population Synthesis (FSPS; Conroy, Gunn, & White 2009, ApJ, 699, 486; Conroy & Gunn 2010, ApJ, 712, 833) to generate spectra corresponding to a set of stellar population parameters. Since this is originally a Fortran package, we use the python-FSPS (Foreman-Mackey, Sick and Johnson, 2014) set of bindings to call FSPS from within python. Installation instructions for these packages can be found at their respective homepages: FSPS and python-FSPS.
- Astropy (v.3.2.1+): For redshift and distance calculations based on different cosmologies.
- George (v.0.3.1+): We use the George package (Ambikasaran et al. 2014) to implement Gaussian processes.
- Scikit-Learn (v.0.21.2+): can be used as an alternative to George, although it doesn't perform as well.
- Corner (v.2.0.1+): Foreman-Mackey (2016) is used to plot prior and posterior distributions.
- Numpy, Scipy, Matplotlib
- This code is written in Python 3.8.

CHAPTER 3

Features

- GP-SFH module: A module that can convert SFH-tuples to smooth curves in SFR vs time space, and vice-versa. This module can also be used to create GP-approximations for any input SFH curve (e.g., SFHs from simulations).
- Prior atlas generator - A set of functions that can generate SEDs corresponding to input stellar population parameters using the FSPS backend. Taken in conjunction with the db.Priors() class, this can be used to sample prior distributions and trade space for time complexity while fitting SEDs.
- SED fitter module - Functions for evaluating the goodness-of-fit given an observed SED with uncertainties, and plotting posterior distributions in parameter- and SFH-space.

CHAPTER 4

The GP-SFH module

4.1 Creating different shapes using SFH-tuples

The `dense_basis` code contains a module for creating smooth star formation history from a tuple consisting of (M:math:_*, SFR, $\{t_X\}$) - the stellar mass, star formation rate, and a set of lookback times at which the galaxy forms N equally spaced quantiles of its stellar mass.

This parametrization comes with a lot of flexibility, and allows us to create a large range of SFH shapes even with a small number of parameters. Here we show a few examples, showing how we create a variety of different SFH shapes with just 2 free parameters - the SFR and the t_{50}.

[1]:
```
import dense_basis as db
import numpy as np
import matplotlib.pyplot as plt
```

```
Starting dense_basis. please wait ~ a minute for the FSPS backend to initialize.
```

[2]:
```
Nparam = 1
redshift = 1.0
logMstar = 10.0
```

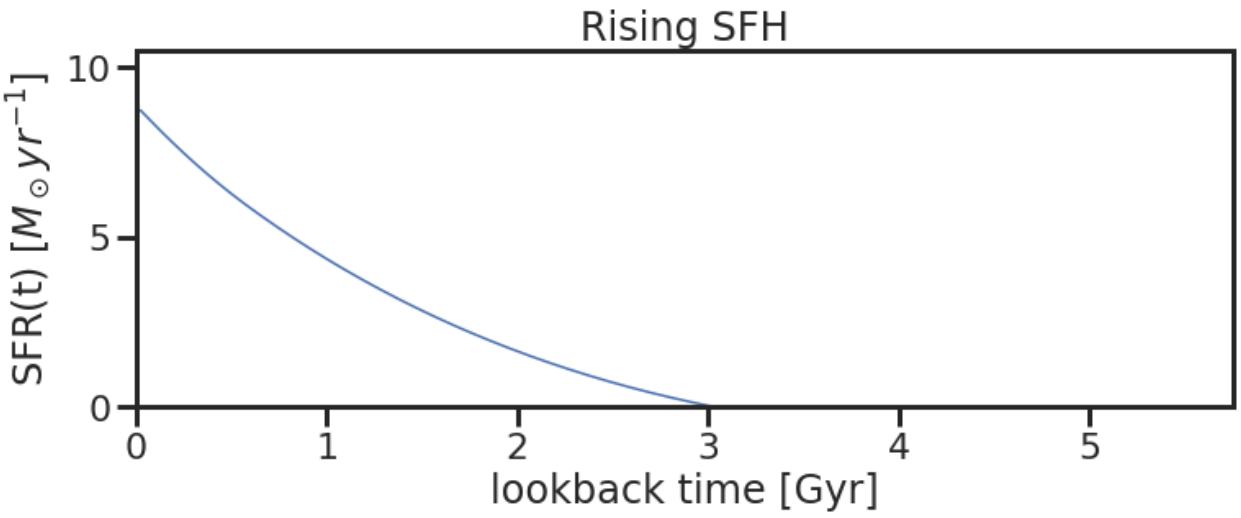
Let's start with an `SFH that is rising` throughout a galaxy's lifetime, such as may be expected for high-redshift star forming galaxies. Since we are considering a galaxy with $M_* = 10^{10} M_\odot$ at z=1, we choose a reasonably high SFR of 10 $M_\odot/yr$. Since the SFR is rising, we also choose a short t_{50}, since it is rapidly building forming its stars. Running this through the model, we get:

[3]:
```
logSFR = 1.0
t50 = 0.6 # t50, lookback time, in Gyr

sfh_tuple = np.hstack([logMstar, logSFR, Nparam, db.scale_t50(t50,redshift)])
sfh, timeax = db.tuple_to_sfh(sfh_tuple, redshift)

fig = db.plot_sfh(timeax, sfh, lookback=True)
```

(continues on next page)

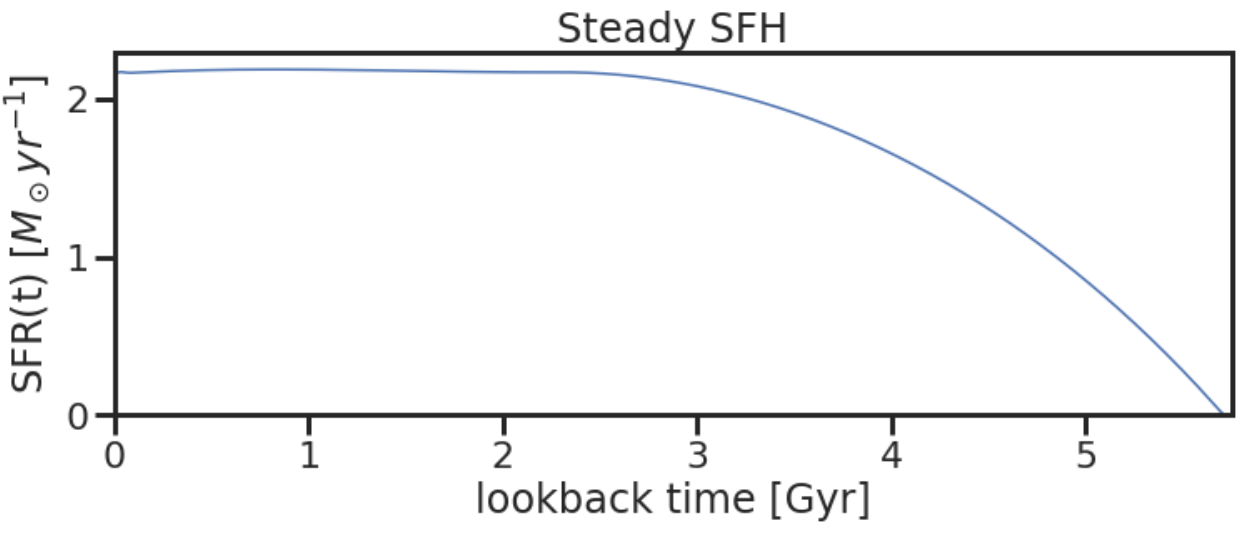
(continued from previous page)

```
plt.title('Rising SFH')
plt.show()
```



We next consider the case of reasonably `steady star formation`. This is different from constant star formation, because SFR goes to 0 smoothly as we approach the big bang. In this case, we choose an SFR closer to the expected lifetime average for a massive galaxy at z=1, and a t_{50} close to half the age of the universe at the redshift of observation. Doing this gives us:

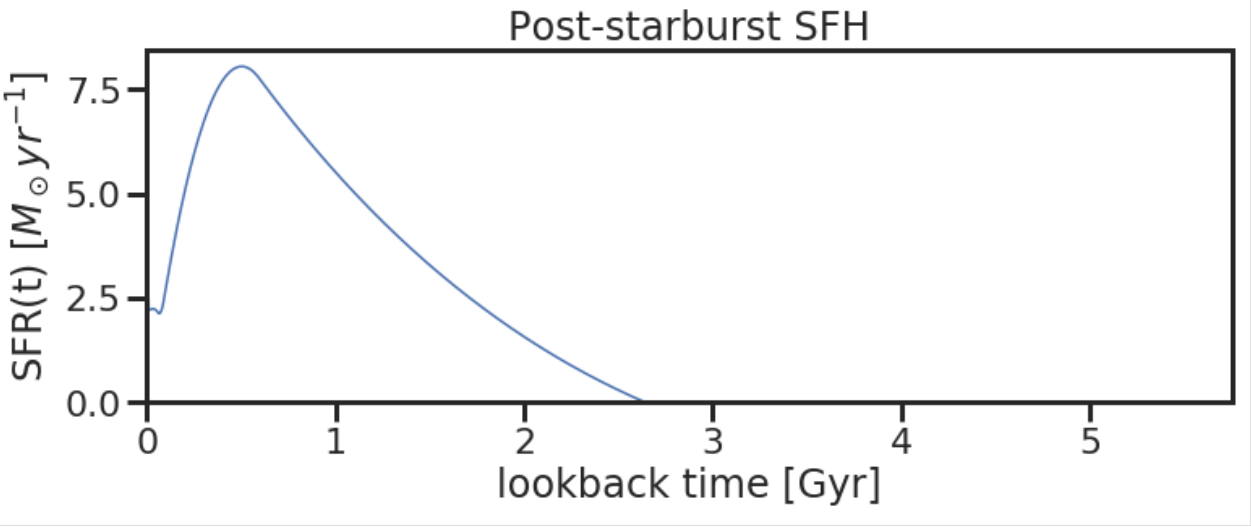
[4]:
```
logSFR = 0.335
t50 = 2.3 # t50, lookback time, in Gyr

sfh_tuple = np.hstack([logMstar, logSFR, Nparam, db.scale_t50(t50,redshift)])
sfh, timeax = db.tuple_to_sfh(sfh_tuple, redshift)

fig = db.plot_sfh(timeax, sfh, lookback=True)
plt.title('Steady SFH')
plt.show()
```



We now look at the class of quenched and quenching galaxies.

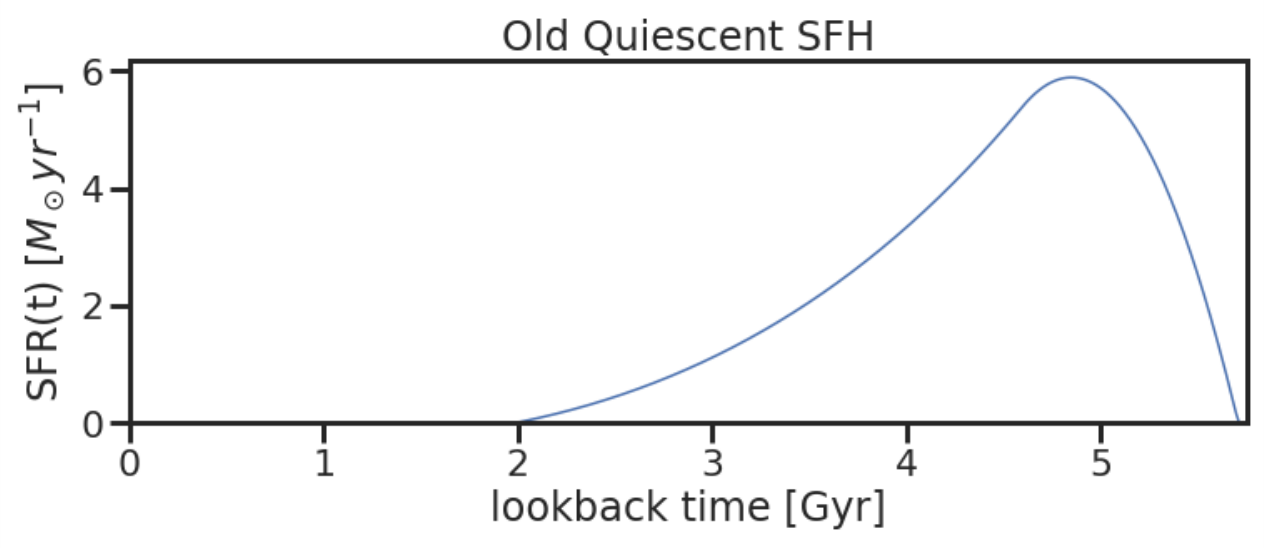
For the `post-starburst SFH`, we create a similar setup to the rising SFH, but with a low SFR at the time of observation. Since the galaxy still formed a lot of stars in the recent past but is not doing so now, this creates the distinctive post-starburst shape.

[5]:
```
logSFR = 0.5
t50 = 0.6 # t50, lookback time, in Gyr

sfh_tuple = np.hstack([logMstar, logSFR, Nparam, db.scale_t50(t50,redshift)])
sfh, timeax = db.tuple_to_sfh(sfh_tuple, redshift)

fig = db.plot_sfh(timeax, sfh, lookback=True)
plt.title('Post-starburst SFH')
plt.show()
```

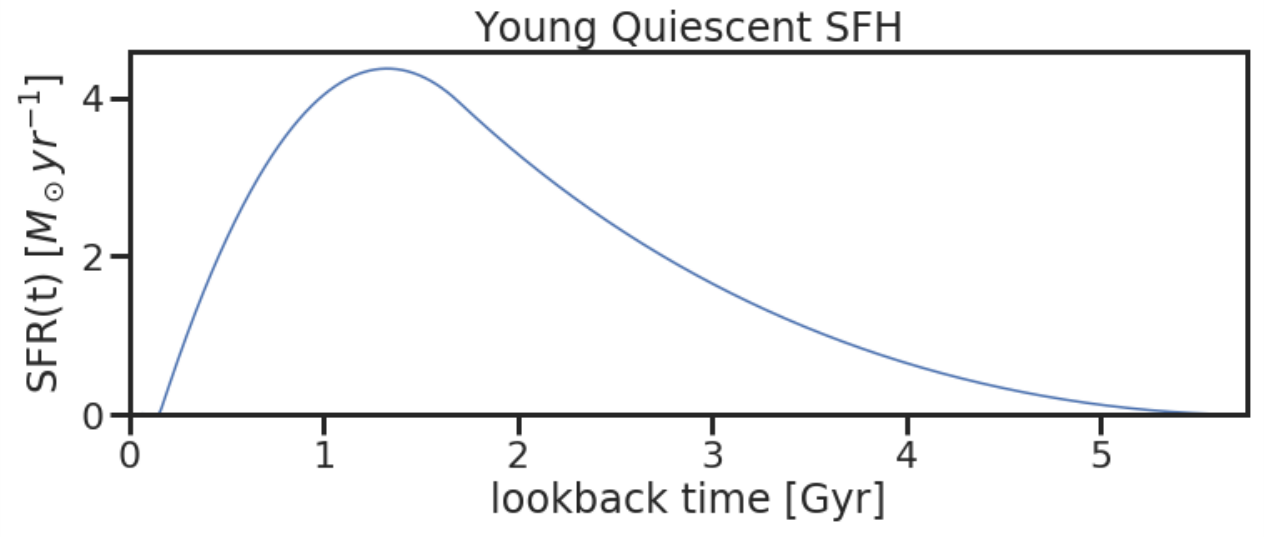


We also consider two simple types of `quenched galaxies`, obtained easily by setting the recent SFR to a very low value. To consider the different possible shapes for a quenched SFH, we use a recent and an older value for the t_{50}, to obtain SFHs that quenched either gradually or aburptly.

[6]:
```
logSFR = -3.0
t50 = 4.6 # t50, lookback time, in Gyr

sfh_tuple = np.hstack([logMstar, logSFR, Nparam, db.scale_t50(t50,redshift)])
sfh, timeax = db.tuple_to_sfh(sfh_tuple, redshift)

fig = db.plot_sfh(timeax, sfh, lookback=True)
plt.title('Old Quiescent SFH')
plt.show()
```



[7]:
```
logSFR = -3.0
t50 = 1.7 # t50, lookback time, in Gyr

sfh_tuple = np.hstack([logMstar, logSFR, Nparam, db.scale_t50(t50,redshift)])
sfh, timeax = db.tuple_to_sfh(sfh_tuple, redshift)

fig = db.plot_sfh(timeax, sfh, lookback=True)
plt.title('Young Quiescent SFH')
plt.show()
```



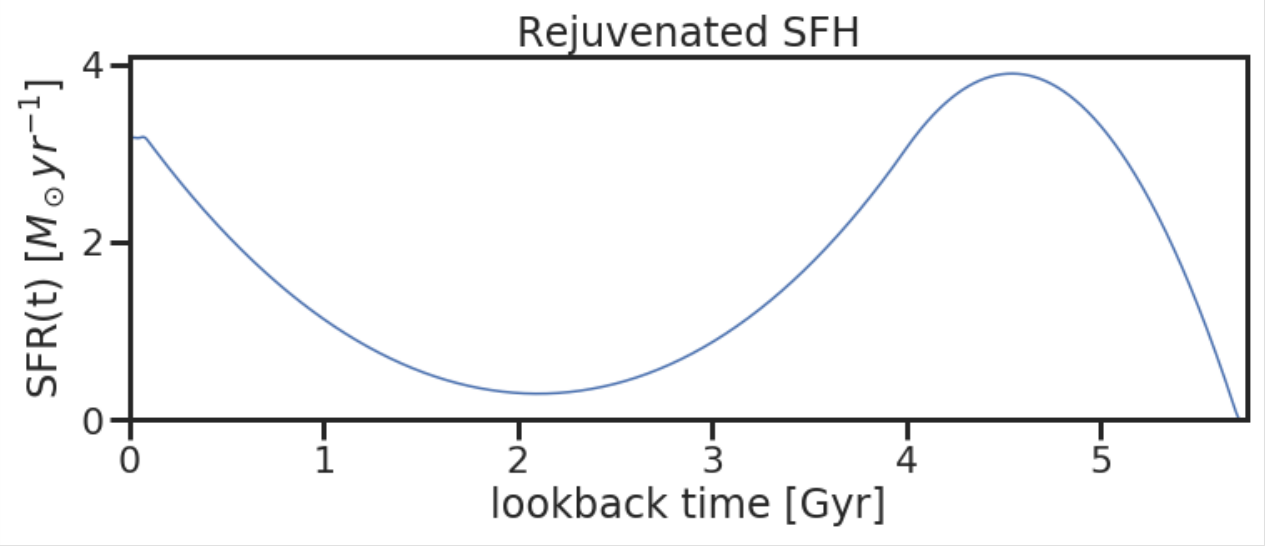
Finally, we also consider the case of a `rejuvenated SFH`, which had a significant lull between two periods of active star formation. To create an example of this kind of SFH, we use a reasonably large t_{50}, which tells the GP-SFH module that the galaxy formed 50% of its stars early on. Coupled with an SFR that indicates active star formation at the time of observation, this means that there had to be a period between these two when the galaxy did not form a lot of mass, leading to this distinctive shape.

[8]:
```
logSFR = 0.5
t50 = 4.0 # t50, lookback time, in Gyr
```

(continues on next page)

(continued from previous page)

```
sfh_tuple = np.hstack([logMstar, logSFR, Nparam, db.scale_t50(t50,redshift)])
sfh, timeax = db.tuple_to_sfh(sfh_tuple, redshift)

fig = db.plot_sfh(timeax, sfh, lookback=True)
plt.title('Rejuvenated SFH')
plt.show()
```



CHAPTER 5

A full SED-Fitting example

This tutorial goes through the things that need to be done to fit the observed spectral energy distributions (SEDs) of galaxies. This is different from SED fitting codes that use a sampler while fitting to create their posteriors, in the sense that we pre-sample the prior volume prior to fitting, trading space for time complexity. This results in a moderately lengthy initialization period where the method generates an atlas of parameters drawn from the priors and SEDs corresponding to these parameters with a user-specified filter-set, which can then be used to fit any number of SEDs in an extremely short amount of time. (Further iterations of the code are also planned to include variants with live samplers for edge-cases and objects with pathological likelihood surfaces.)

The atlas essentially provides a coarse mapping from the galaxy's stellar population parameters (stellar mass, SFR, star formation history, dust attenuation, metallicity, and redshift) to their corresponding SEDs.

Instantiate the module, making sure you have all the prerequisite packages (especially python-fsps and george) installed. Don't worry if the initial import takes a few minutes, because it's initializing its FSPS backend.

If you need to change any of the FSPS parameters, do so using `db.mocksp.params['key'] = value`, consulting the python-fsps API for reference.

5.1 Note: If you're using a version older than 0.1.4, the syntax for computing atlases has changed. Please update to the current version to be consistent with the tutorial and examples.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import dense_basis as db
print('DB version: ',db.__version__)
```

```
Starting dense_basis. please wait ~ a minute for the FSPS backend to initialize.
DB version:  0.1.8
```

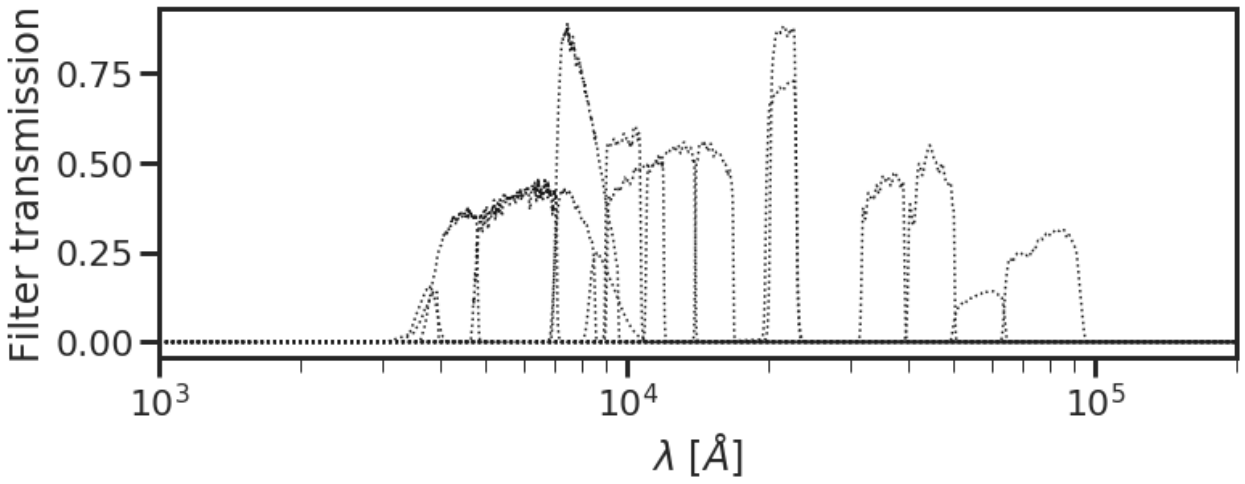
CHAPTER 6

Prerequisites to fitting:

If you're fitting photometry, put your photometric filter transmission curves in a folder somewhere and make a list of filter curves with the paths to each filter. You'll need to pass `filter_list` and `filt_dir` as arguments to the code to generate SEDs corresponding to a given parameter set.

The `db.plot_filterset()` function can be used to visualize the set of filter curves used to make SEDs. Let's load a filter list corresponding to the CANDELS GOODS-South photometric catalog for now:

[2]:
```
filter_list = 'filter_list_goodss.dat'
filt_dir = 'internal' # path to directory containing filter list
db.plot_filterset(filter_list = filter_list, filt_dir = filt_dir)
```

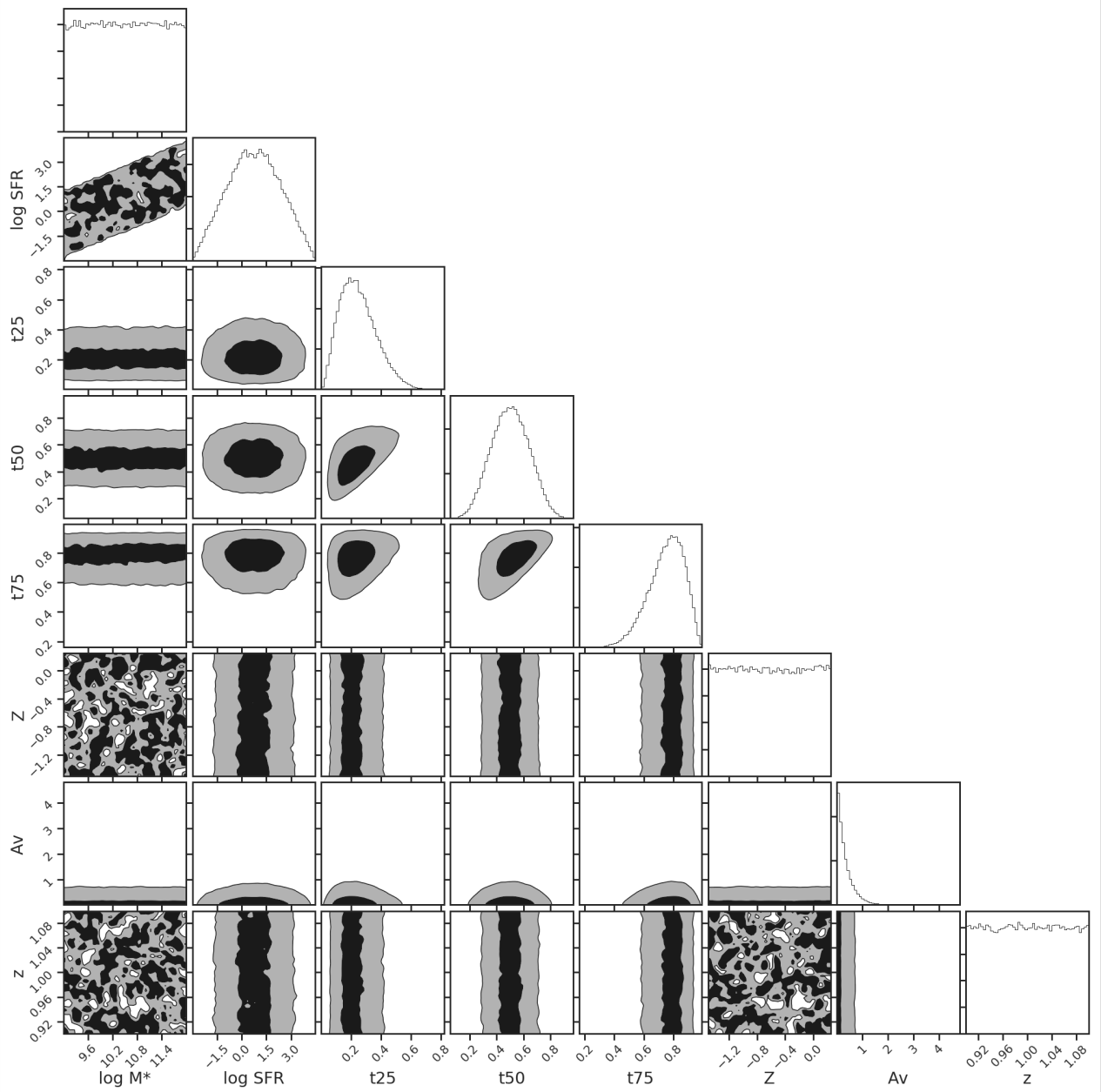


6.1 1. Instantiate a Priors() object

The next step is to generate a template atlas that you will use for fitting. The advantage of doing this is that we trade time-complexity for space, which is usually much more abundant, and it only needs to be done once to fit a large

dataset. Before doing this, however, we need a set of priors that we can draw from to generate this atlas. This is done using the `db.Priors()` class:

[3]:
```
priors = db.Priors()
priors.tx_alpha = 3.0
priors.plot_prior_distributions()
```



6.2 2. Generate an atlas of parameters drawn from the prior and their corresponding SEDs.

Now we can use the priors object to generate the atlas. The important arguments here are the size of the atlas (`N_pregrid`), which samples from the overall multidimensional prior distributions, and the number of SFH param-

eters (`priors.Nparam`). The generated atlas is then stored in a local `/pregrids` folder with the user-specified fname within the current working directory. Please specify a different path using the `path` argument if you would like it to be stored in a different place. The first few SEDs take time to compute because every time FSPS encounters a new metallicity value it needs to load a grid into memory. This happens only once, and when it is done, the atlas generation process speeds up considerably.

[4]:
```
fname = 'test_atlas'
N_pregrid = 10000
priors.Nparam = 3
path = 'pregrids/'
db.generate_atlas(N_pregrid = N_pregrid,
                  priors = priors,
                  fname = fname, store=True, path='pregrids/',
                  filter_list = filter_list, filt_dir = filt_dir)
```

```
  0%|          | 0/10000 [00:00<?, ?it/s]
```

```
generating atlas with:
3  tx parameters,  sSFRflat  SFR sampling custom  SFH treatment flat  met sampling␣
↪Calzetti  dust attenuation exp  dust prior False  SFR decoupling.
```

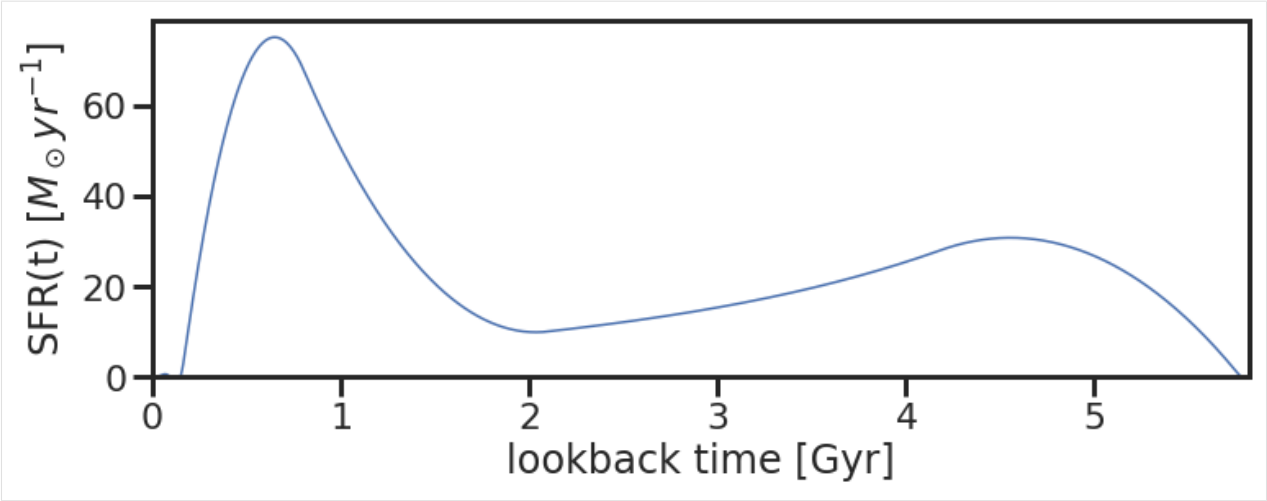
```
100%|| 10000/10000 [13:21<00:00, 12.48it/s]
```

```
Path exists. Saved atlas at : pregrids/test_atlas_10000_Nparam_3.dbatlas
```

6.3 3. Generate a mock SED to be fit.

To illustrate the SED fitting procedure, let's generate a mock star formation history (SFH) to recover. This can be done by sampling our priors for an SFH-tuple and then converting it to a SFR-vs-time curve using the `db.tuple_to_sfh()` command. We can then generate its corresponding spectrum, and multiply the spectrum with our prespecified filter-set to get the corresponding SED.

[5]:
```
# sample from the prior space to get parameters
rand_sfh_tuple, rand_Z, rand_Av, rand_z = priors.sample_all_params(random_seed = 1)

specdetails = [rand_sfh_tuple, rand_Av, rand_Z, rand_z]

# generate an SFH corresponding to the SFH-tuple and see how it looks:
rand_sfh, rand_time = db.tuple_to_sfh(rand_sfh_tuple, zval = rand_z)
fig = db.plot_sfh(rand_time, rand_sfh, lookback=True)
sfh_truths = [rand_time, rand_sfh]

# generate a corresponding spectrum and multiply by filter curves to get the SED:
obs_sed = db.makespec(specdetails, priors, db.mocksp, db.cosmo,
                              filter_list=filter_list,filt_dir=filt_dir,input_
↪sfh=False)
obs_err = obs_sed * 0.1 # S/N of 10

# store the true stellar mass and SFR
mstar_true = np.log10(db.mocksp.stellar_mass)
sfr_true = np.log10(db.mocksp.sfr)

sed_truths = (mstar_true, sfr_true, rand_sfh_tuple[3:], rand_Z, rand_Av, rand_z)
sed_truths = np.hstack(sed_truths)
```


SFR(t) [$M_\odot yr^{-1}$]
0
20
40
60
0
1
2
3
4
5
lookback time [Gyr]

CHAPTER 7

SED fitting with the dense_basis package:

7.1 4. Load the previously generated atlas:

Step 2 is extremely beneficial in fitting large datasets, since the atlas needs to be generated only once and can be used for fitting as many SEDs as needed using the brute-force Bayesian approach. Having generated this dataset, now an arbitrary SED (`obs_sed`, and its errors `obs_err`) can be fit using the previously generated atlas.

[6]:
```
# load the atlas
atlas = db.load_atlas(fname, N_pregrid = N_pregrid, N_param = priors.Nparam, path =␣
↪path)

# pass the atlas and the observed SED + uncertainties into the fitter,
sedfit = db.SedFit(obs_sed, obs_err, atlas, fit_mask=[])

# evaluate_likelihood returns the likelihood for each SED in the atlas and the norm␣
↪value to
# best match the observed SED with the atlas.
sedfit.evaluate_likelihood()

# evaluate_posterior_percentiles calculates the 16,50,84th percentiles for
# the physical parameters - stellar mass, SFR, tx, dust, metallicity and redshift
sedfit.evaluate_posterior_percentiles()
```

7.2 5. Fit the SED and visualize the parameter posteriors:

If we are interested in the full posteriors of the fit, this can be visualized by making the fitter return the chi2 array and then computing the full posteriors as prior*likelihood. Let's see how it compares to the truth:

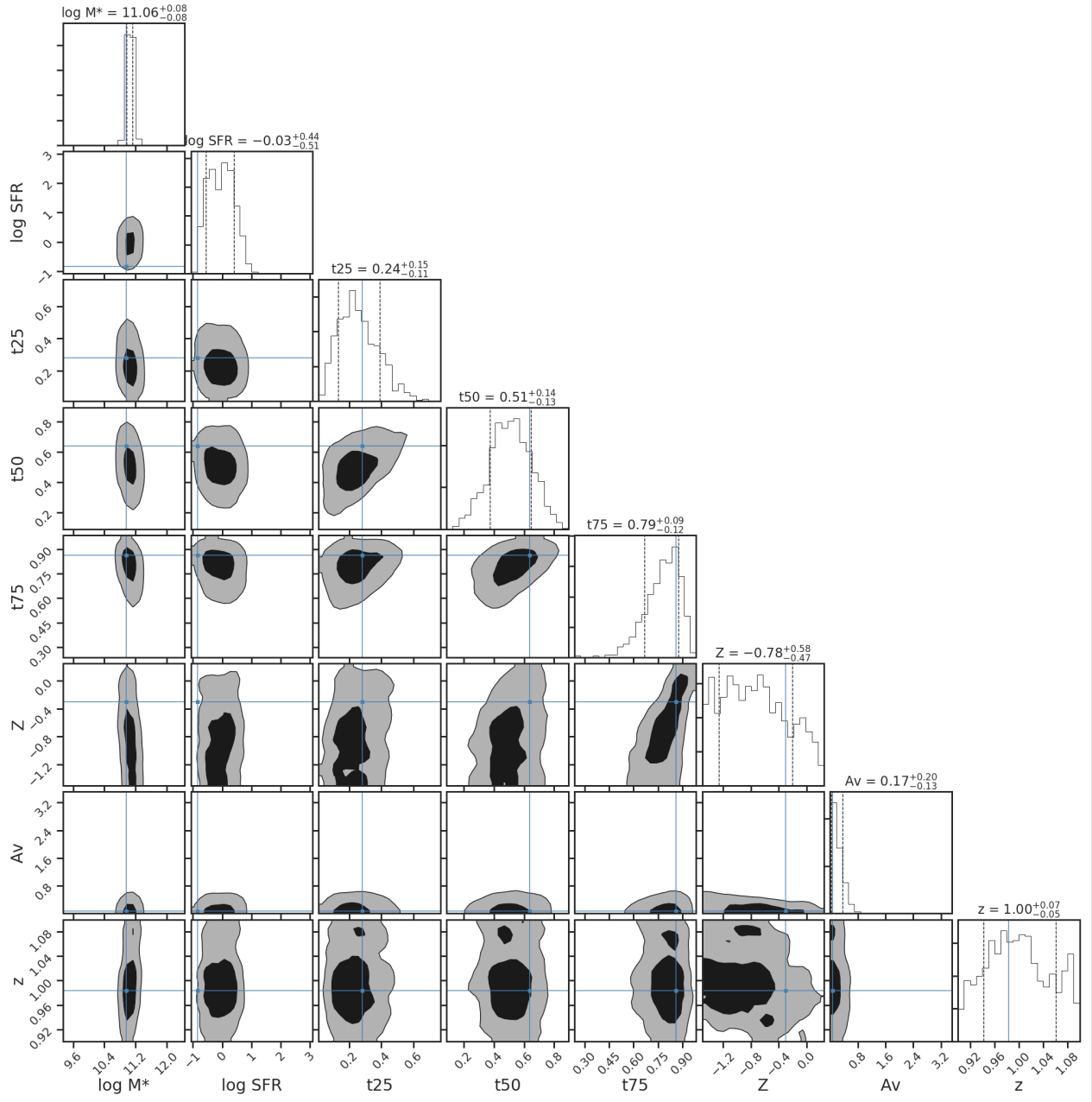
[7]:
```
%timeit sedfit.evaluate_likelihood()
```

```
6.76 ms ± 37.4 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

[8]:
```
print('log Stellar Mass: %.2f' %sedfit.mstar[0])
print('log SFR: %.2f' %sedfit.sfr[0])
```

```
log Stellar Mass: 11.06
log SFR: -0.03
```

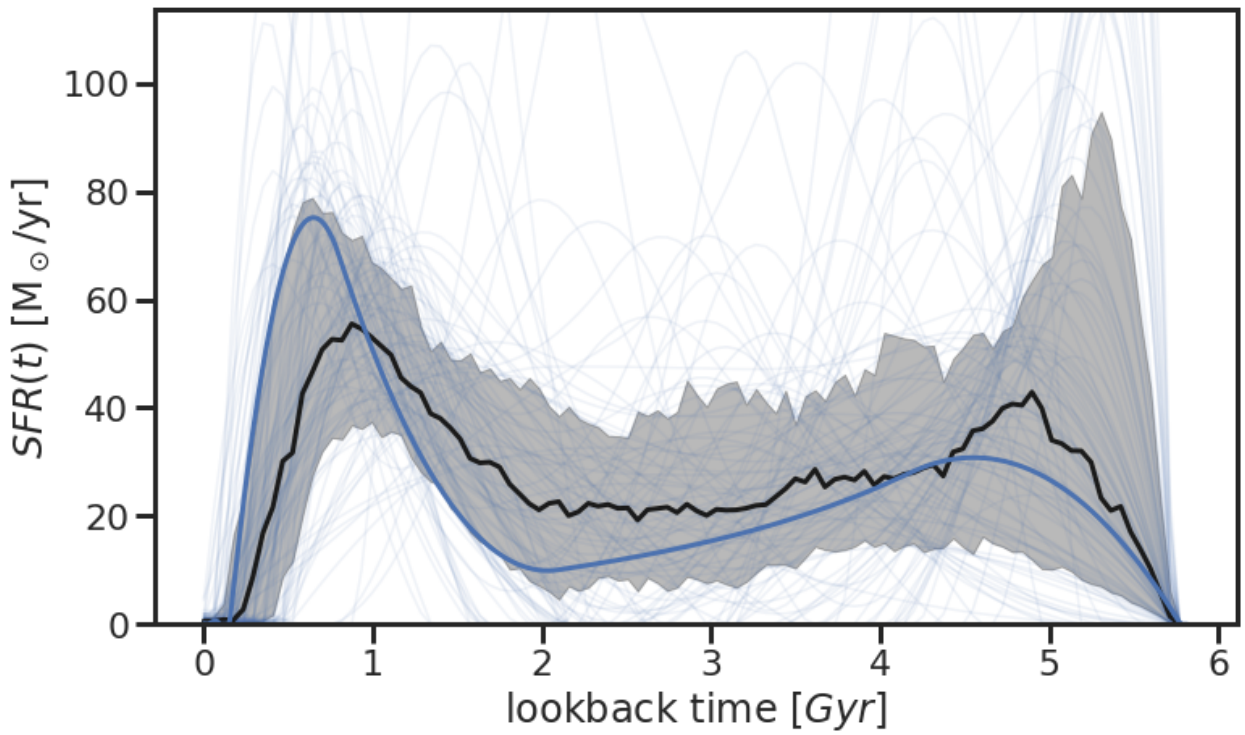
[9]:
```
sedfit.plot_posteriors(truths=sed_truths)
plt.show()
```



7.3 6. Visualize the posterior SFH and spectrum:

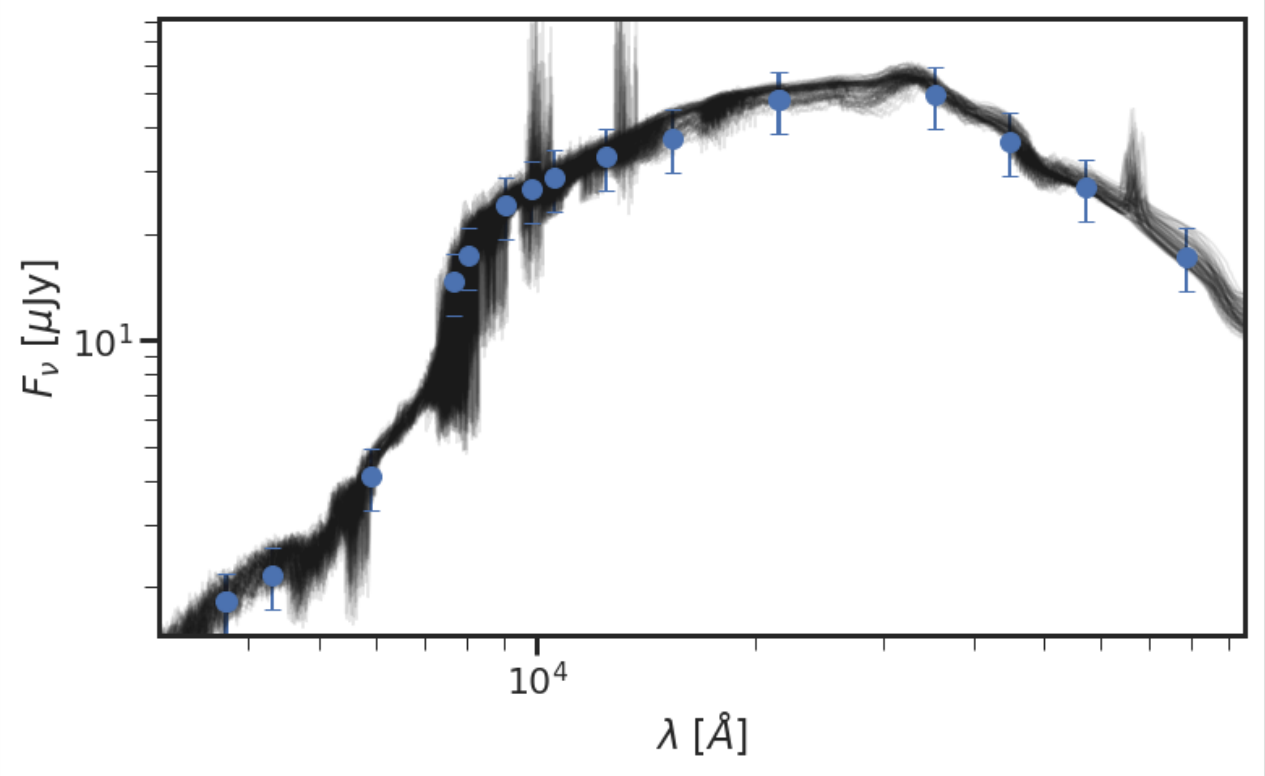
Finally, we can also plot the posterior SFH and see how it compares to the true SFH:

[10]:
```
sedfit.plot_posterior_SFH(sedfit.z[0])
plt.plot(np.amax(sfh_truths[0]) - sfh_truths[0],sfh_truths[1],lw=3)
plt.show()
```



[11]:
```
# deprecated code - use if you're using v.0.1.5 or older

# db.plot_SFH_posterior(chi2_array, norm_fac, obs_sed, atlas,
#                           truths = sfh_truths, sfh_threshold = 0.7)
```

[12]:
```
centers_goods_s = np.array([3734,3722,4317,5918,7693,8047,9055,9851,10550,12486,15370,
↪21605,21463,35508,44960,57245,78840])
sedfit.plot_posterior_spec(centers_goods_s, priors)
plt.show()
```

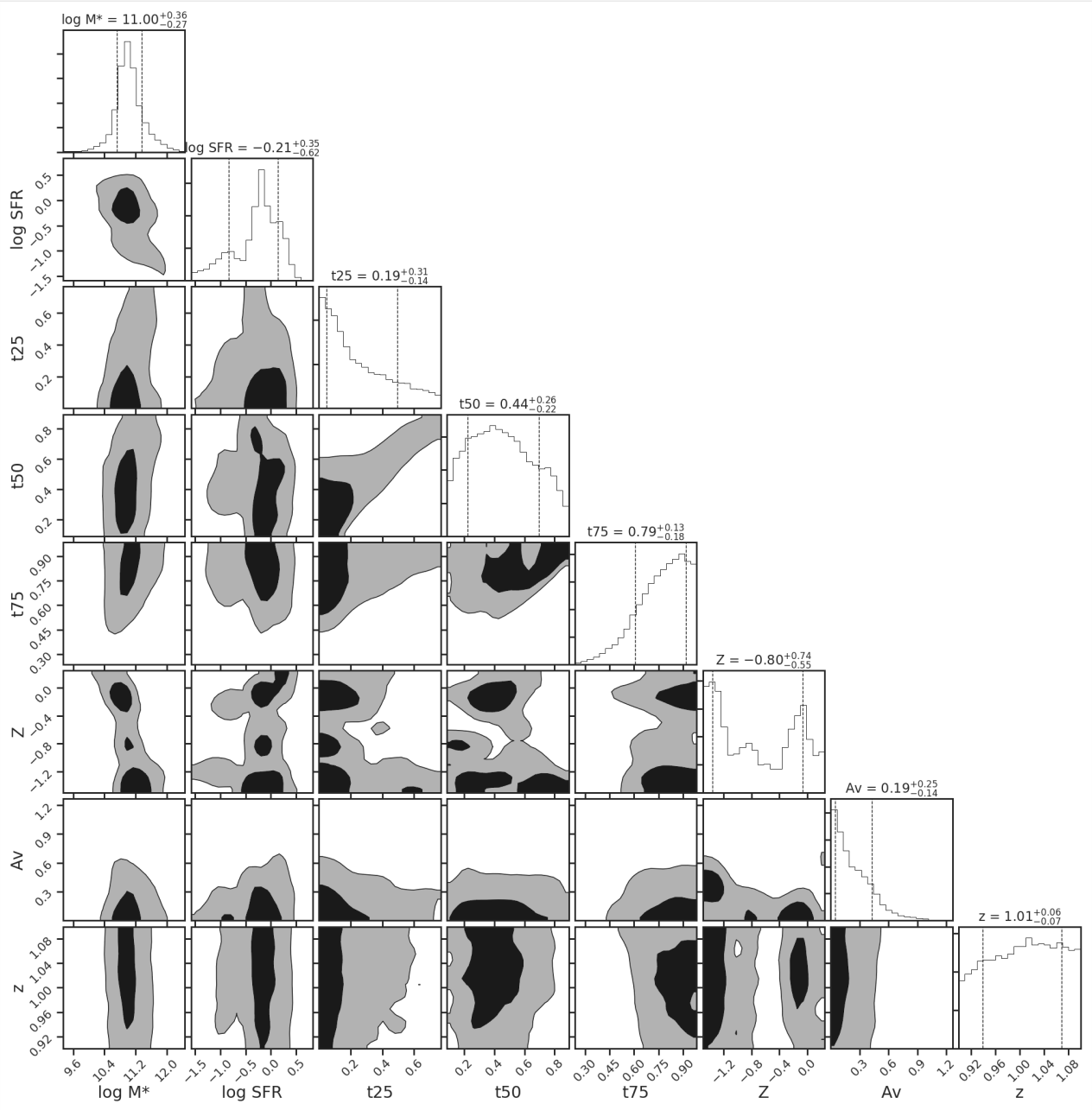


7.4 7. Compare with MCMC results

The dense_basis package also has an extremely fast MCMC sampler now available, for when the uncertainties need to be sampled more robustly, multi-modal solutions further investigated, or when SEDs might lie on the edge of the prior space. This uses the excellent emcee package (so make sure that's installed before you try to run the following), and returns an `emcee.sampler` object. Let's try running this with the galaxy above and see what we get for the posteriors.

[13]:
```
sampler = db.run_emceesampler(obs_sed, obs_err, atlas, epochs=10000, plot_
↪posteriors=True)
```

```
100%|| 10000/10000 [00:44<00:00, 225.26it/s]
```


log M* = 11.00$^{+0.36}_{-0.27}$
log SFR = −0.21$^{+0.35}_{-0.62}$
t25 = 0.19$^{+0.31}_{-0.14}$
t50 = 0.44$^{+0.26}_{-0.22}$
t75 = 0.79$^{+0.13}_{-0.18}$
Z = −0.80$^{+0.74}_{-0.55}$
Av = 0.19$^{+0.25}_{-0.14}$
z = 1.01$^{+0.06}_{-0.07}$
log M*
log SFR
t25
t50
t75
Z
Av
z

CHAPTER 8

Adding custom filter sets

The code now has the functionality to add custom filter response functions, along with the pre-built functionality to use the default filter sets for the five CANDELS fields. To see how this works, let's load up the module:

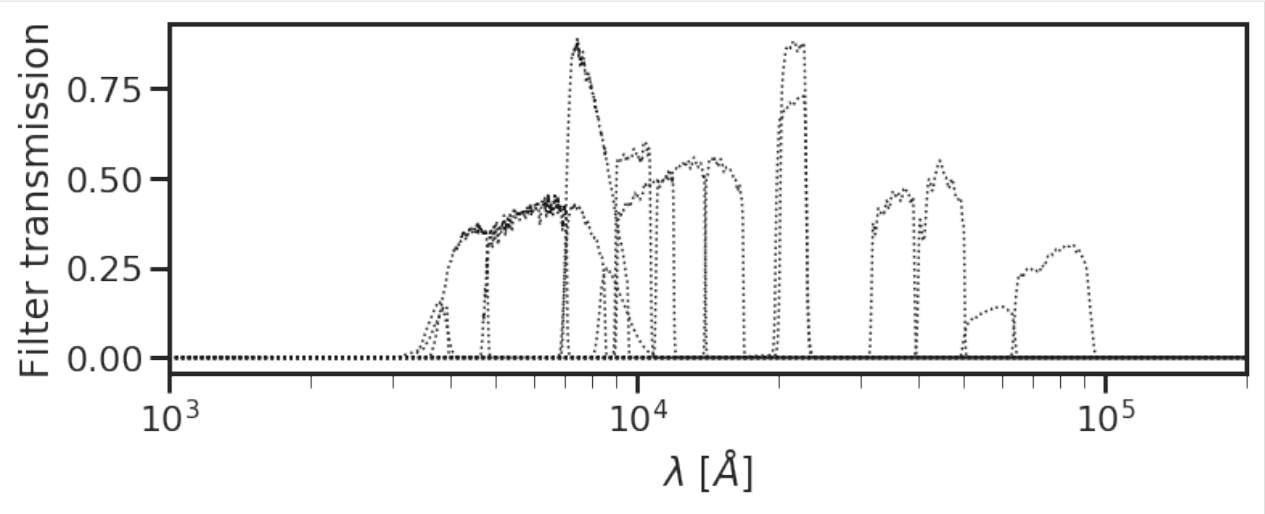
[1]:
```
import dense_basis as db
import os
```

```
Starting dense_basis. please wait ~ a minute for the FSPS backend to initialize.
Initialized stellar population with FSPS.
```

The set of filters is specified by two arguments: `filter_list`, which is a file that contains a list of filters along with relative paths specifying where the various filters are, and `filt_dir`, which specifies the absolute directory where the filter_list can be found.

To use one of the default available lists, simply set `filt_dir = 'internal'`, and specify one of the CANDELS fields as shown below. The `db.plot_filterset` command loads the filters and plots their transmission values as a function of wavelength.

[2]:
```
filter_list = 'filter_list_goodss.dat'
filt_dir = 'internal' # path to directory containing filter list
db.plot_filterset(filter_list = filter_list, filt_dir = filt_dir)
```



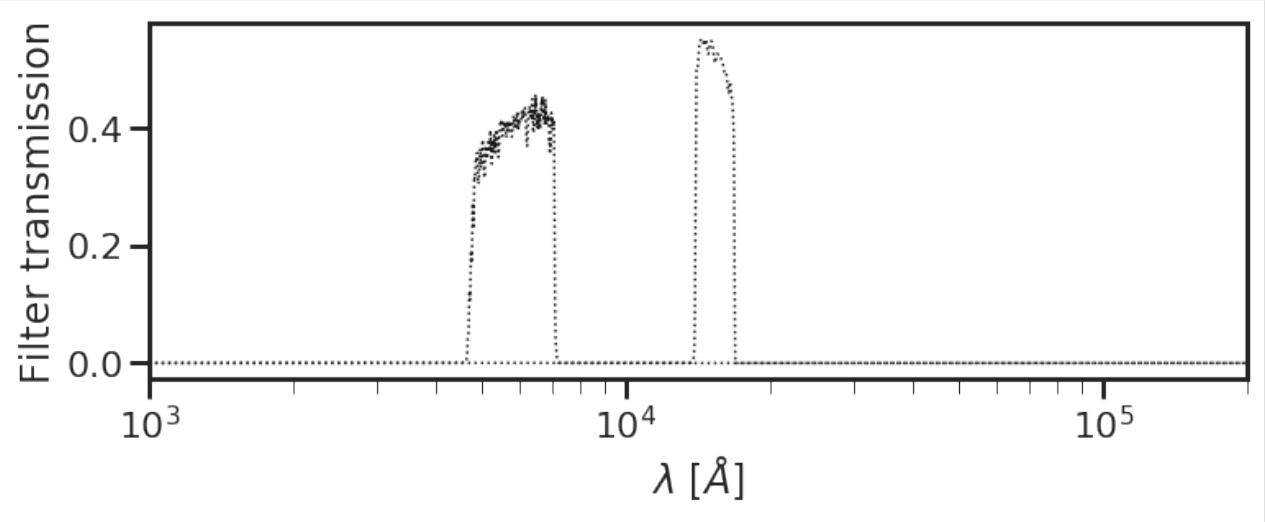
To specify your own custom set of filters and use them in the Dense Basis code, do the following:

1. Collect all of your individual filter transmission curves in one place. In the example below, I'm using two filters, named (wfc3_ir_160.txt) and (Filter04_ACS_f606w_2.dat). Both of these files contain a list of wavelengths and their corresponding filter response values that can be downloaded from sites like the SVO filter profile service. Put them in a folder if you'd like. For this example, I've kept mine in a folder named (filter_curves) within my working directory.
2. Create a `filter_list file`. In this case, I made a file in my working directory named `test_list.dat`, containing two lines, each of which specifies the relative path to a single filter:

[ ]:
```
filter_curves/wfc3_ir_160.txt
filter_curves/Filter04_ACS_f606w_2.dat
```

3. We're almost done. Now get the absolute path to the working directory (the one containing `test_list.dat`) using `db.get_path`. Pass these in as inputs, and plot them to confirm everything is working as expected:

[3]:
```
filter_list = 'test_list.dat'
filt_dir = db.get_path(filter_list) # path to directory containing filter list
db.plot_filterset(filter_list = filter_list, filt_dir = filt_dir)
```



[ ]:

CHAPTER 9

SFH reconstruction test suite

Now that we have started up the package and built an atlas, let's see how we do at fitting SFHs for various galaxy demographics - starforming, starbursting, quenched and rejuvenated galaxies. First thing, let's load the package:

[1]:
```
import numpy as np
import dense_basis as db
```

```
Starting dense_basis. please wait ~ a minute for the FSPS backend to initialize.
Initialized stellar population with FSPS.
```

Next we load the atlas that we built previously, with

[2]:
```
fname = 'test_atlas'
N_pregrid = 300000
Nparam = 3
path = 'internal'

pg_sfhs, pg_Z, pg_Av, pg_z, pg_seds, norm_method = db.load_atlas(fname, N_pregrid,␣
↪Nparam, path=path)
pg_theta = [pg_sfhs, pg_Z, pg_Av, pg_z, pg_seds]
```

9.1 Different SFH shapes:

We now define SFH-tuples corresponding to galaxies from different demographics, as follows:

[3]:
```
rising_sfh = np.array([10.0,1.5,3,0.5,0.7,0.85])
regular_sfg_sfh = np.array([10.0,0.35,3,0.3,0.55,0.8])
post_starburst_sfh = np.array([10.0,0.6,3,0.5,0.8,0.9])
old_quenched_sfh = np.array([10.0,-10.0,3,0.15,0.3,0.5])
double_peaked_SF_sfh = np.array([10.0,0.5,3,0.25,0.30,0.7])
double_peaked_Q_sfh = np.array([10.0,-1.0,3,0.1,0.6,0.7])
```

(continues on next page)

(continued from previous page)

```
sfh_list = [rising_sfh, regular_sfg_sfh, post_starburst_sfh, old_quenched_sfh, double_
↪peaked_SF_sfh, double_peaked_Q_sfh]
sfh_names = ['Rising/Starburst galaxy', 'Regular star-forming galaxy', 'Post-
↪Starburst galaxy', 'Quiescent SFH', 'Double-peaked SFH [SF]', 'Double-peaked SFH␣
↪[quiescent]']
```

9.2 Plotting posteriors after fitting these SFHs:

We then loop through these SFHs, and - generate corresponding SEDs with randomly sampled values for dust, metallicity, and redshifts (although $z \sim 1$) - fit these SEDs using the dense basis SED fitter, and - use the resulting likelihood surface to plot and SFH posterior, comparing it to the truth

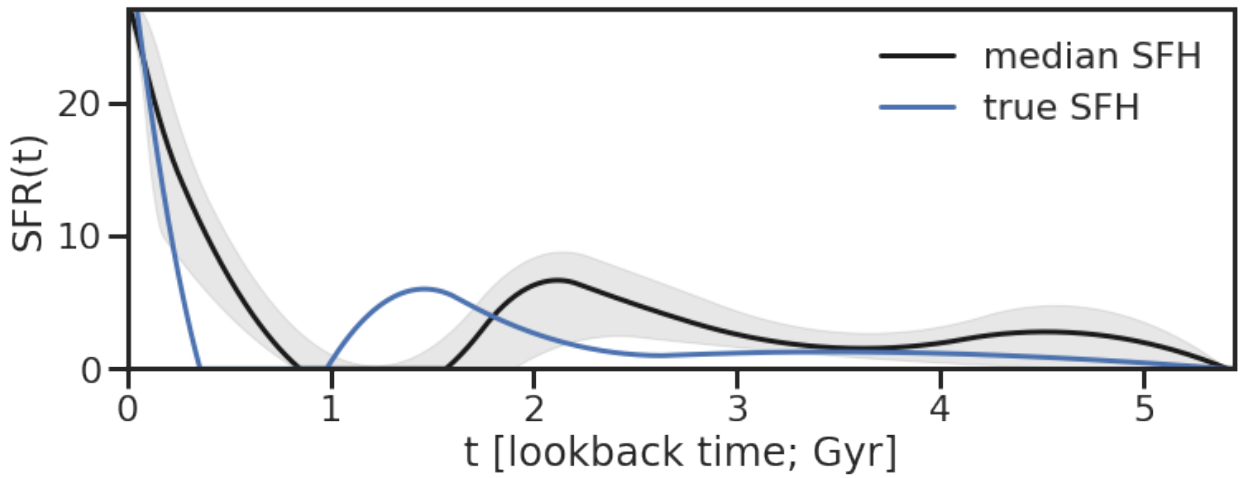
Since FSPS takes a while to load up the grids corresponding to different metallicities, the following cell might take a while when executed the first time, but subsequent runs should be fast.

[4]:
```
priors = db.Priors()
priors.Nparam = Nparam
priors.tx_alpha = 3.0
```

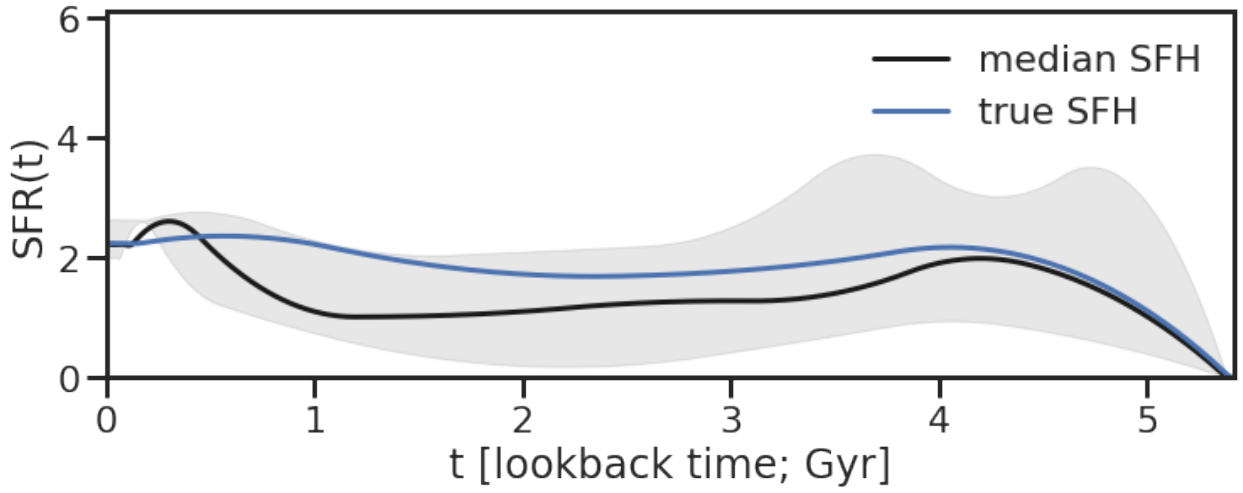
[5]:
```
def sfh_fitting_test(sfh_tuple, sfh_name, max_num = 1000):

    print('---------- '+sfh_name+' ------------')

    rand_sfh_tuple, rand_Z, rand_Av, rand_z = priors.sample_all_params_
↪safesSFR(random_seed = 7)
    rand_sfh_tuple = sfh_tuple

    # generate an SFH corresponding to the SFH-tuple and see how it looks:
    rand_sfh, rand_time = db.tuple_to_sfh(rand_sfh_tuple, zval = rand_z)
    # fig = db.plot_sfh(rand_time, rand_sfh, lookback=True)
    sfh_truths = [rand_time, rand_sfh]

    # generate a corresponding spectrum and multiply by filter curves to get the SED:
    _, sfr_true, mstar_true = db.make_spec(rand_sfh_tuple, rand_Z, rand_Av, rand_z,␣
↪return_ms = True)
    rand_spec, rand_lam = db.make_spec(rand_sfh_tuple, rand_Z, rand_Av, rand_z,␣
↪return_lam = True)

    filter_list = 'filter_list_goodss.dat'
    filt_dir = 'internal'
    obs_sed = db.calc_fnu_sed(rand_spec, rand_z, rand_lam, fkit_name = filter_list,␣
↪filt_dir = filt_dir)
    obs_err = obs_sed * 0.1 # S/N of 33
    sed_truths = (mstar_true, sfr_true, rand_sfh_tuple[3:], rand_Z, rand_Av, rand_z)
    sed_truths = np.hstack(sed_truths)

    chi2_array = db.fit_sed_pregrid(obs_sed, obs_err,
    pg_theta, return_val = 'chi2', norm_method=norm_method)

    db.plot_SFH_posterior_v3(chi2_array, obs_sed, pg_theta, truths=sfh_truths, sfh_
↪threshold=0.1, max_num = max_num)
```

[6]:
```
for i in range(len(sfh_list)):

    sfh_fitting_test(sfh_list[i], sfh_names[i])
```

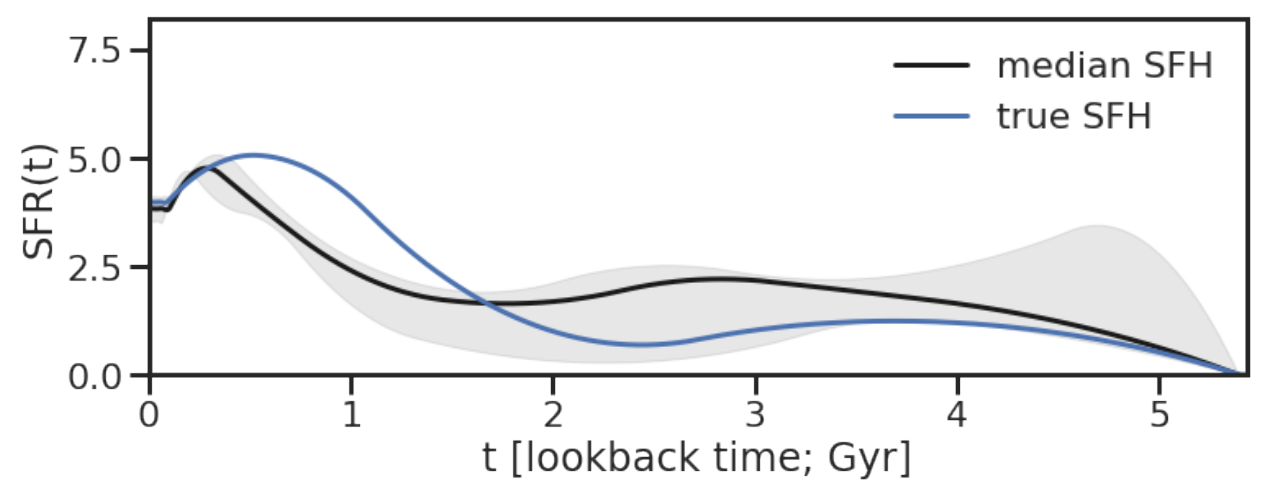
```
---------- Rising/Starburst galaxy ------------
truncated to 1000 SFHs to reduce computation time. increase max_num if desired.
```



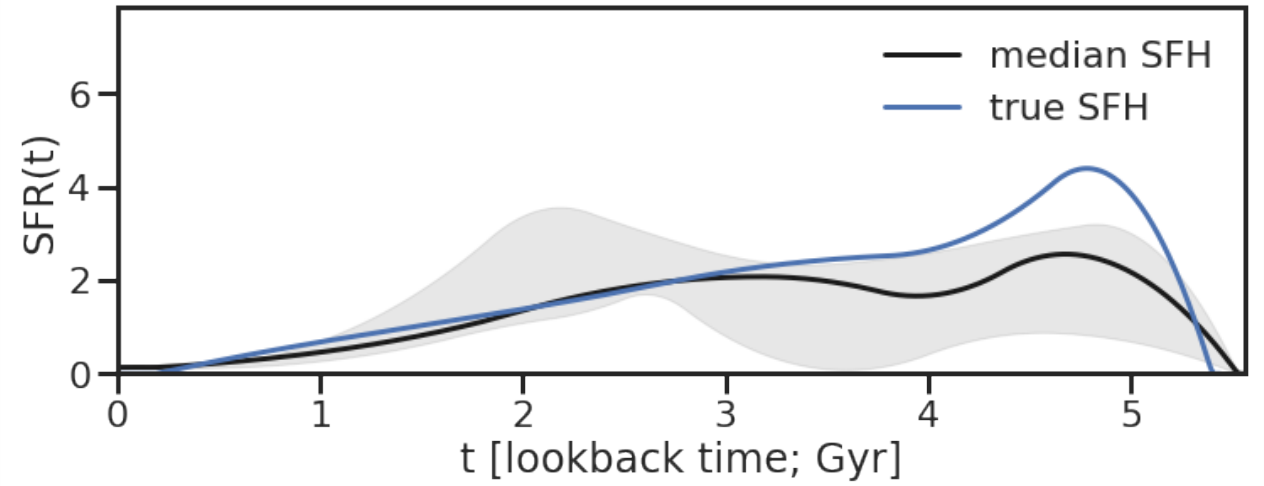

```
---------- Regular star-forming galaxy ------------
truncated to 1000 SFHs to reduce computation time. increase max_num if desired.
```



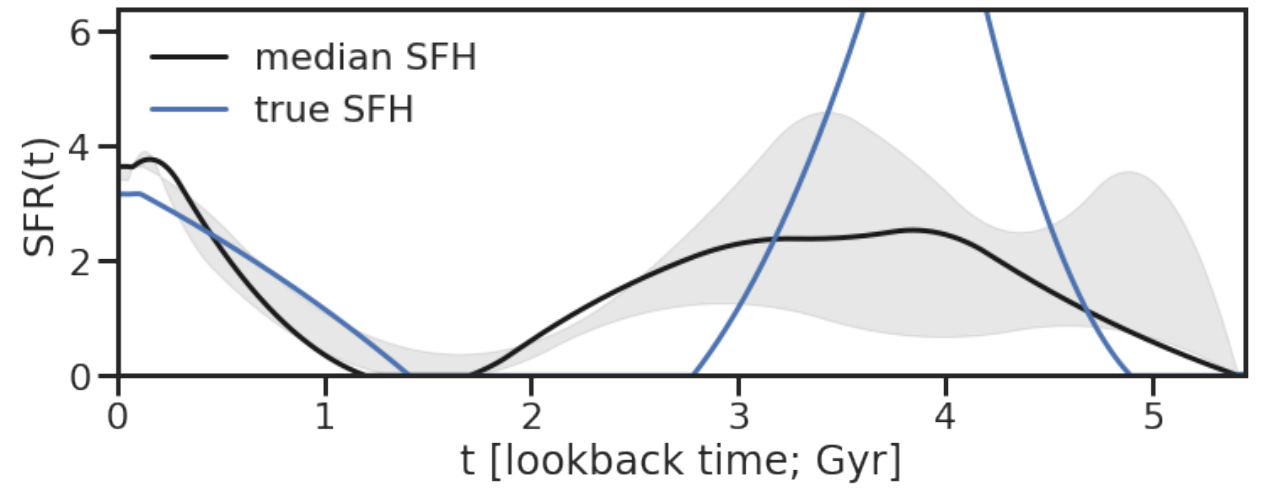

```
---------- Post-Starburst galaxy ------------
truncated to 1000 SFHs to reduce computation time. increase max_num if desired.
```



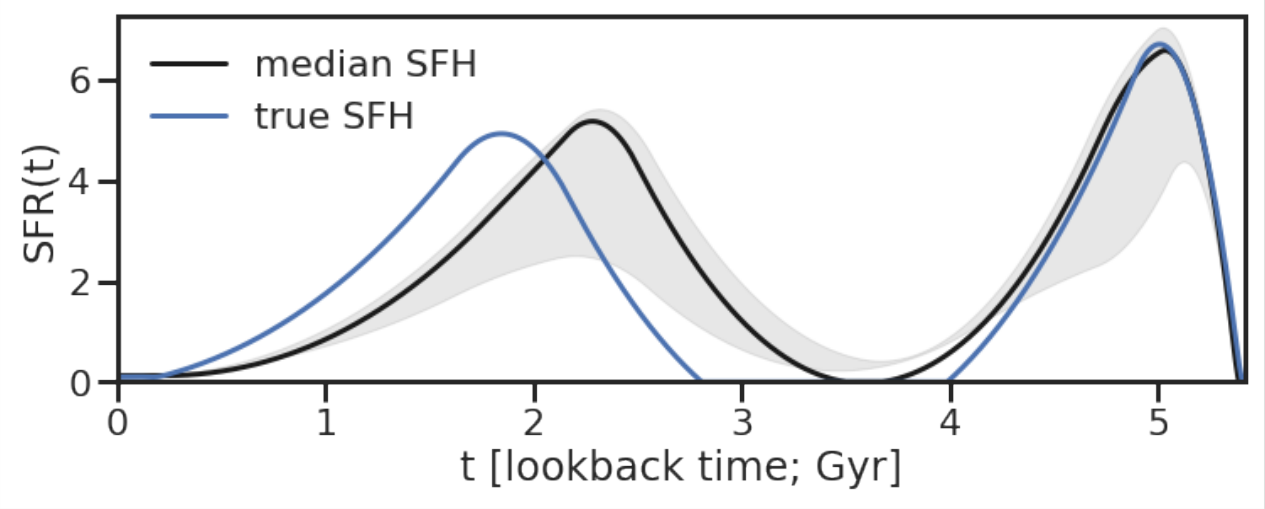

```
---------- Quiescent SFH ------------
truncated to 1000 SFHs to reduce computation time. increase max_num if desired.
```




```
---------- Double-peaked SFH [SF] ------------
truncated to 1000 SFHs to reduce computation time. increase max_num if desired.
```



```
---------- Double-peaked SFH [quiescent] ------------
truncated to 1000 SFHs to reduce computation time. increase max_num if desired.
```



We see that we do well in recovering the general shape of the SFHs, as well as derived quantities like stellar mass and recent SFR. The poorer reconstructions are discussed below in a bit more detail: - **Post-starburst galaxy**: While most of the true SFH is within the uncertainties, the reconstruction is not able to completely recover the recent SF prior to the quenching. This is mostly a prior issue, with post-starburst galaxies disfavoured by our adopted prior - in this case, going to larger Nparam or coupling the sSFR to the redshift allows for more variety in the SFH shapes and helps recover this population. - **Double-peaked quenched galaxy**: while this performs reasonably well, the peaks are poorly constrained due to low S/N from the old stellar populations. In this case, we actually find th older peak to be better constrained mainly because the SFH has to be 0 at the big bang. This also leads to the uncertainties not being estimated accurately, which is also usually mitigated by a larger atlas. - **Old quiescent galaxy and the double-peaked SF galaxy**: This is simply a S/N issue. The fitter detects that the galaxy is quiescent, but doesn't have enough information to accurately constrain the older stellar populations, leading to the posterior being prior rather than likelihood dominated.

The code is designed to be intuitive to use, and and consists of three steps to get you started doing SED fitting:

- defining your priors
- generating a model atlas (params <-> SEDs) to use while fitting
- actually fitting your data and visualizing the posteriors

More detailed descriptions of these modules can be found in the tutorials. If you are interested in going off the beaten track and trying different things, please let me know so that I can help you run the code as you'd like!

CHAPTER 10

Contribute

- Issue Tracker: https://github.com/kartheikiyer/dense_basis/issues
- Source Code: https://github.com/kartheikiyer/dense_basis

CHAPTER 11

Support

If you are having issues, please let me know at: kartheik.iyer@dunlap.utoronto.ca

CHAPTER 12

License & Attribution

dense_basis is being developed by Kartheik Iyer in a public GitHub repository. The source code is made available under the terms of the MIT license.

If you make use of this code, please cite the recent Dense Basis paper.

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

D