EzGal Documentation

Release 2.0

Conor Mancone, Anthony Gonzalez

April 02, 2012

CONTENTS

Contents:

CHAPTER

ONE

FILE FORMATS

1.1 Model File Format

Three different model formats are accepted by `EzGal`: binary ised files, ASCII files, and fits files generated by `EzGal`. `EzGal` can read any of the binary ised files that come with the BC03 and CB07 models. This includes the SSP files as well as files generated by csp_galaxev.

`EzGal` can read in ASCII files, allowing it to work with any model set. By default `EzGal` expects SEDs in ASCII files to have wavelength units of angstroms and flux units of ergs/s/angstrom, with age units of Gyrs. The ASCII model file should be a simple text table with the number of rows equal to `number of wavelengths + 1` and the number of columns equal to `number of ages + 1`. The first row specifies the age of each column, and the first column specifies the wavelength of each row. Therefore the data value in the first row of the first column is ignored, although it must still contain a value (e.g., zero).

`EzGal` can also save and read models in fits format. The advantage of doing this is that as you calculate magnitude evolution for various filters and formation redshifts, `EzGal` stores this information and will write it out to the fits file when saved. When that fits file is later reloaded `EzGal` will also restore the stored magnitude evolution information, allowing you to quickly fetch magnitude evolution without having to recalculate it.

1.2 Filter File Format

When loading filter response curves `EzGal` expects filter response files to be two column text files, the first column containing wavelength, and the second fractional transmission. By default it expects the wavelengths to have units of angstroms. Units other than angstroms may be specified by passing a string giving the units to the `units` keyword (see `ezgal.utils.to_meters()` for list of available units).

CHAPTER

TWO

MODEL CALCULATIONS

2.1 Loading Models

To create an EzGal object in python you simply need to import the EzGal module and then create an object by passing the name of a model file. EzGal will search in three different places for the specified file: at the specified location, in a model directory specified with the EZGAL_MODELS environment variable, and in the EzGal model directory (data/models/ in the EzGal module folder). In python this is the calling sequence:

```
import ezgal
model = ezgal.model( 'filename' )
```

2.2 Adding Filters

Once you have created an EzGal object you can add filters with the ezgal.ezgal.add_filter() method. Simply pass the filename that contains the filter response curve (see *Filter File Format* for a description of the filter file format). EzGal will search for the filter file in one of three locations: at the specified location, in a filter directory specified with the EZGAL_FILTERS environment variable, and in the EzGal filter directory (data/filters/ in the EzGal module folder). The filename (without any path information) will then become the name of the filter which will be used later when calculating observables:

```
import ezgal
model = ezgal.model( 'model_filename' )
model.add_filter( 'filter_filename' )
```

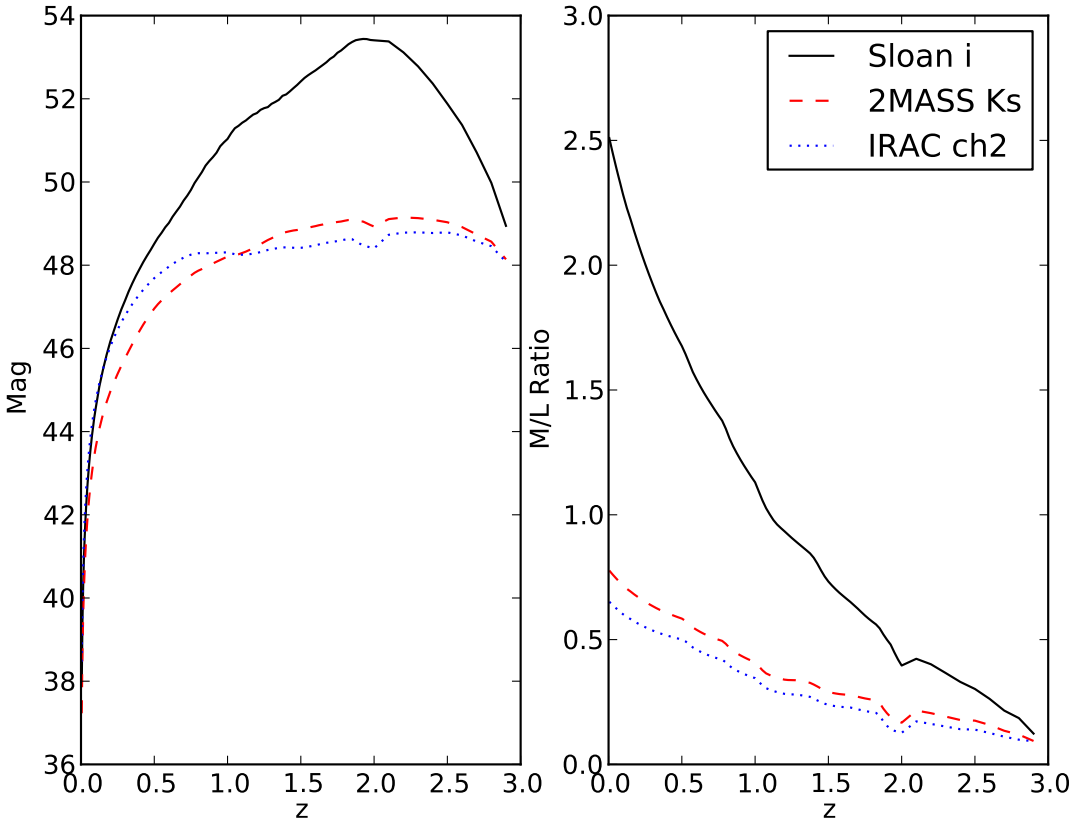
However, ezgal.ezgal.add_filter() does not often have to be called explicitly, as filters can be auto-loaded when calculating observables as long as they are in one of the EzGal filter directories. Instead, just pass the filename (without any path information) when calculating observables and EzGal will find and add the filter automatically. A number of filter files are distributed with EzGal in the filter directory, so these can always be auto-loaded. This method of loading filters is used in all of the examples below.

2.3 Calculating Observables

EzGal has a number of methods for fetching observables as a function of redshift. Most have the same calling sequence, which is to simply specify the formation redshift and optionally, filters and redshifts. The methods which have this calling sequence are: ezgal.ezgal.get_apparent_mags(), ezgal.ezgal.get_absolute_mags(), ezgal.ezgal.get_kcorrects(), ezgal.ezgal.get_ecorrects(), ezgal.ezgal.get_ekcorrects(), ezgal.ezgal.get_rest_ml_ratios(), ezgal.ezgal.get_observed_ml_ratios(),

`ezgal.ezgal.get_solar_observed_mags()`. If you do not pass a list of filters then it will return the observables for all the filters that have been loaded into EzGal. These methods all return numpy arrays of shape `(zs.size, filters.size)` when observables are being calculated for more than one filter. If observables are only being calculated for one filter then it will return a numpy array of shape `(zs.size,)` or just a scalar value if only one redshift is passed.

```
import ezgal
from pylab import *

# load ezgal model file
model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )

# desired formation redshift
zf = 3.0
# fetch an array of redshifts out to given formation redshift
zs = model.get_zs( zf )

# plot magnitude evolution versus redshift for three filters
subplot( 1,2,1 )
plot( zs, model.get_apparent_mags( zf, filters='sloan_i', zs=zs ), 'k-', label='Sloan i' )
plot( zs, model.get_apparent_mags( zf, filters='ks', zs=zs ), 'r--', label='2MASS Ks' )
plot( zs, model.get_apparent_mags( zf, filters='ch2', zs=zs ), 'b:', label='IRAC ch2' )
# and set labels
xlabel( 'z' )
ylabel( 'Mag' )

# plot mass-to-light ratio evolution versus redshift for the same filters
subplot( 1,2,2 )
plot( zs, model.get_rest_ml_ratios( zf, filters='sloan_i', zs=zs ), 'k-', label='Sloan i' )
plot( zs, model.get_rest_ml_ratios( zf, filters='ks', zs=zs ), 'r--', label='2MASS Ks' )
plot( zs, model.get_rest_ml_ratios( zf, filters='ch2', zs=zs ), 'b:', label='IRAC ch2' )
# and set labels
xlabel( 'z' )
ylabel( 'M/L Ratio' )

# how about a legend?
legend()

# all done
show()
```



There are many methods for fetching observables as a function of formation redshift (zf) and redshift(z). All such methods are explained, with examples, in the *EzGal API*. The methods are also summarized here:

Method	Returns
ezgal.ezgal.get_age()	Age as a function of zf and z
ezgal.ezgal.get_absolute_mags()	Rest-frame absolute magnitudes as a function of filter, zf, and z
ezgal.ezgal.get_apparent_mags()	Apparent magnitude as a function of filter, zf, and z
ezgal.ezgal.get_distance_moduli()	Distance moduli as a function of z
ezgal.ezgal.get_ecorrects()	e-corrections as a function of filter, zf, and z
ezgal.ezgal.get_ekcorrects()	e+k corrections as a function of filter, zf, and z
ezgal.ezgal.get_kcorrects()	k-corrections as a function of filter, zf, and z
ezgal.ezgal.get_mass_weighted_ages()	Mass weighted ages as a function of zf and z (for CSPs only)
ezgal.ezgal.get_masses()	Mass as a function of zf and z
ezgal.ezgal.get_normalization()	The model normalization, if set
ezgal.ezgal.get_observed_absolute_mags()	Observed-frame absolute magnitudes as a function of filter, zf, and z
ezgal.ezgal.get_observed_ml_ratios()	Observed-frame M/L ratios as a function of filter, zf, and z
ezgal.ezgal.get_rest_ml_ratios()	Rest-frame M/L ratios as a function of filter, zf, and z
ezgal.ezgal.get_sed()	SEDs as a function of age
ezgal.ezgal.get_sed_z()	SEDs as a function of zf and z
ezgal.ezgal.get_solar_observed_mags()	Observed-frame magnitude of the sun as a function of filter and z
ezgal.ezgal.get_solar_rest_mags()	Rest-frame magnitude of the sun as a function of filter
ezgal.ezgal.get_zs()	A list of redshifts out to some z

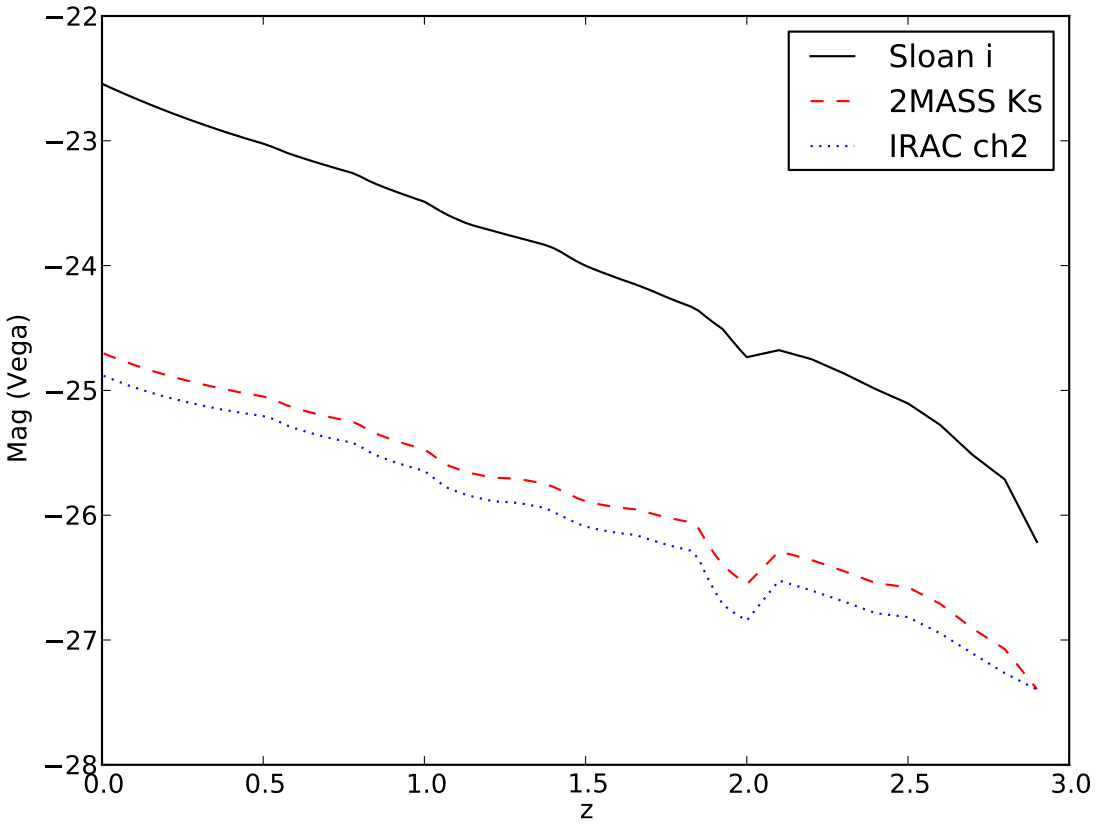
2.4 Normalizations and Vega Output

EzGal allows you to specify a model normalization. Just specify the desired magnitude of a galaxy at a particular redshift and through a loaded filter, and EzGal will calculate the model normalization and automatically apply it to the magnitude calculations for all other filters. This is done with ezgal.ezgal.set_normalization(), and by default it expects the normalization magnitude to be an absolute AB magnitude. If the normalization magnitude is in Vega mags or apparent mags then you can tell EzGal this by setting vega=True or apparent=True in the call to ezgal.ezgal.set_normalization().

To have EzGal output Vega magnitudes call the ezgal.ezgal.set_vega_output() method of any EzGal object. All subsequent calls to ezgal.ezgal.get_apparent_mags() or ezgal.ezgal.get_absolute_mags() will then return Vega magnitudes. You can later resume AB output by calling the ezgal.ezgal.set_ab_output() method. You can also call ezgal.ezgal.get_apparent_mags() or ezgal.ezgal.get_absolute_mags() with vega=True to retrieve Vega mags for an individual call.

```
import ezgal
from pylab import *

# load ezgal model file
model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )

# Tell the model to output Vega magnitudes (default is AB)
model.set_vega_output()

# set the model normalization to Dai et al 2009 (ApJ, 697, 506)
model.set_normalization( 'ch1', 0.24, -25.06, vega=True )

# desired formation redshift
zf = 3.0
```

```
# fetch an array of redshifts out to given formation redshift
zs = model.get_zs( zf )

# plot magnitude evolution versus redshift for three filters
plot( zs, model.get_absolute_mags( zf, filters='sloan_i', zs=zs ), 'k-', label='Sloan i' )
plot( zs, model.get_absolute_mags( zf, filters='ks', zs=zs ), 'r--', label='2MASS Ks' )
plot( zs, model.get_absolute_mags( zf, filters='ch2', zs=zs ), 'b:', label='IRAC ch2' )
# and set labels
xlabel( 'z' )
ylabel( 'Mag (Vega)' )

# how about a legend?
legend()
# all done
show()
```



2.5 Storing Observables

It is possible to calculate the complete redshift evolution of a model given a particular formation redshift and filter and then to store this information in a `EzGal` model file. When this same model file is loaded into `EzGal` later it will fetch the pre-calculated evolution, allowing you to quickly retrieve magnitude evolution without having to recompute it every time. This is how you would create and save such a model file:

```
# load model
import ezgal
model = ezgal.model( 'model_file' )

# add desired filters
model.add_filter( 'ch1' )
model.add_filter( 'ch2' )
model.add_filter( 'ch3' )
model.add_filter( 'ch4' )

# Calculate evolution for above filters with zf = 3 and 5
model.set_zfs( [3.0,5.0] ) # ~10 second delay while evolution is calculated for all filters and forma

# Save the model with evolution information
model.save_model( 'new_model.fits' )
```

This new model file can be loaded and used with `EzGal` just like any other model file, but without any delays for calculating model evolution.

CHAPTER

THREE

MODIFYING MODELS: CSPS AND INTERPOLATION

3.1 CSPs: Arbitrary Star Formation Histories

`EzGal` objects have one primary method, `ezgal.ezgal.make_csp()`, for generating a CSP with an arbitrary star formation history and dust law from any SSP. The primary inputs into `ezgal.ezgal.make_csp()` are a python function or callable object representing the star formation history as a function of time. This function will be passed an age or array of ages (in gyrs) and should return a weight (i.e. a relative star formation rate) at each age. `EzGal` will normalize the SEDs automatically, so the star formation history function does not have to be normalized. Additional arguments for `EzGal` to pass to the star formation history function should be passed to `ezgal.ezgal.make_csp()` with the 'args' parameter (a tuple).

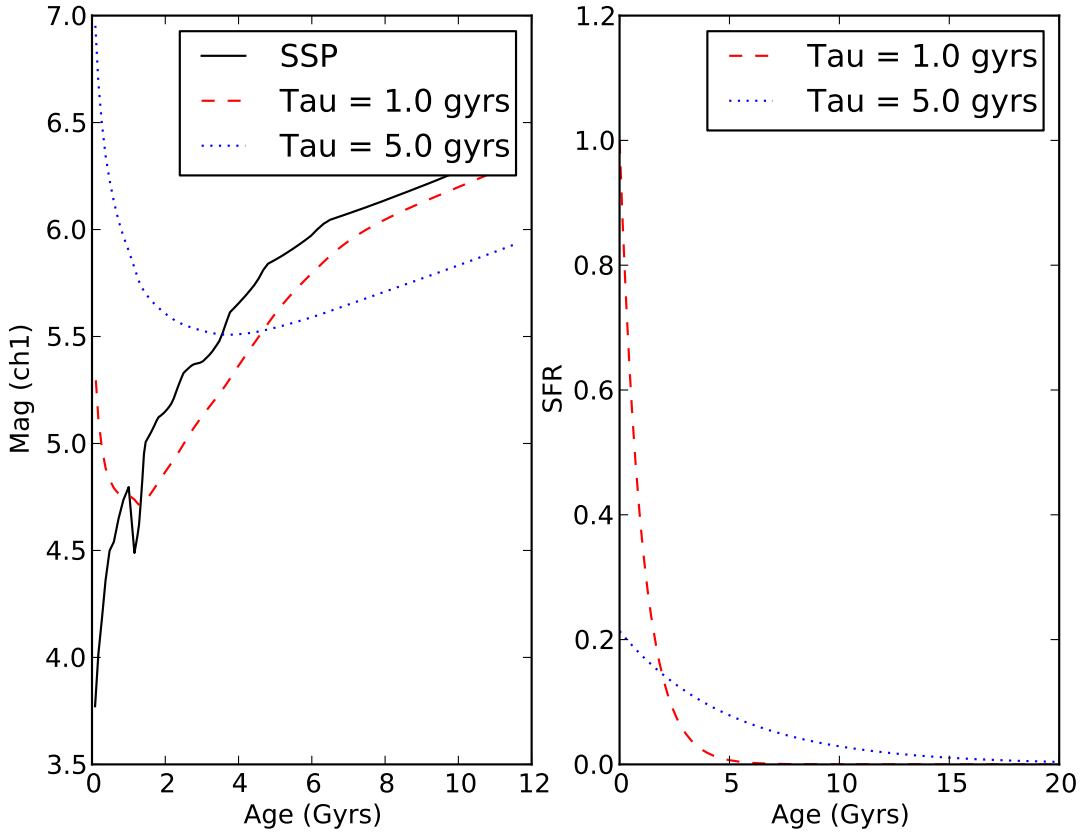
An arbitrary dust law can also be passed, and must once again be a python function or callable object, and extra parameters can be given to `ezgal.ezgal.make_csp()` to pass along to the dust function. The dust function will be given a list of ages (in gyrs) and a list of wavelengths (in angstroms) and must return the dimming factor at each combination of age and wavelength in an array of shape `(number of wavelengths, number of ages)`.

The `ezgal.ezgal.make_csp()` method returns an `EzGal` object with the new star formation history. The new `EzGal` object will also have an additional property, `sfh`, which is an array with a size of `model.ages` which gives the normalized results from the passed star formation history function as a function of age. This property is saved when the CSP model is saved to a fits file and automatically restored later. If an age-mass relationship is stored in the `EzGal` object than it will also be appropriately integrated and stored in the new CSP model object.

When calculating CSPs `EzGal` interpolates the model's age grid onto a finer grid to minimize errors from numeric integration. `EzGal` uses an automated method to decide on the level of sub-gridding to use given a desired level of accuracy. This is set by passing a maximum error (in magnitudes) to the `max_err` parameter of `ezgal.ezgal.make_csp()`. Before calculating the CSP `EzGal` performs the full integral at all ages at a wavlength of 3000, 8000, and 12000 Angstroms, with increasingly finer levels of age sampling. This process stops when the difference (in magnitudes) between two subsequent steps drops below `max_err` at all ages or when it has iterated `max_iter` number of times.

```
import ezgal
import numpy as np
from pylab import *

# load ezgal model file
model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )

# define an exponentially decaying sfh with a setable e-folding length
# ezgal will pass ages in Gyrs
def exponential_sfh( t, tau ):
```

```
        return np.exp( -1.0*t/tau )

# now generate two CSPs with tau = 1.0 and 5.0 (Gyrs)
exp_1 = model.make_csp( exponential_sfh, args=(1.0,) )
exp_5 = model.make_csp( exponential_sfh, args=(5.0,) )

# now plot age versus absolute mag
# ezgal does everything in redshift space, so we first have to get some redshifts
zf = 3.0
zs = model.get_zs( zf )
ages = model.get_age( zf, zs )

# and plot
subplot( 1,2,1 )
plot( ages, model.get_absolute_mags( zf, filters='ch1', zs=zs ), 'k-', label='SSP' )
plot( ages, exp_1.get_absolute_mags( zf, filters='ch1', zs=zs ), 'r--', label='Tau = 1.0 gyrs' )
plot( ages, exp_5.get_absolute_mags( zf, filters='ch1', zs=zs ), 'b:', label='Tau = 5.0 gyrs' )
# some labels
xlabel( 'Age (Gyrs)' )
ylabel( 'Mag (ch1)' )
legend()

# also plot the star formation history (only available for CSPs)
# the ages array for an ezgal model is in units of years - convert to gyrs
exp1_ages = ezgal.utils.convert_time( exp_1.ages, incoming='years', outgoing='gyrs' )
exp5_ages = ezgal.utils.convert_time( exp_5.ages, incoming='years', outgoing='gyrs' )
subplot( 1,2,2 )
plot( exp1_ages, exp_1.sfh, 'r--', label='Tau = 1.0 gyrs' )
plot( exp5_ages, exp_5.sfh, 'b:', label='Tau = 5.0 gyrs' )
# some labels
xlabel( 'Age (Gyrs)' )
ylabel( 'SFR' )
legend()

show()
```



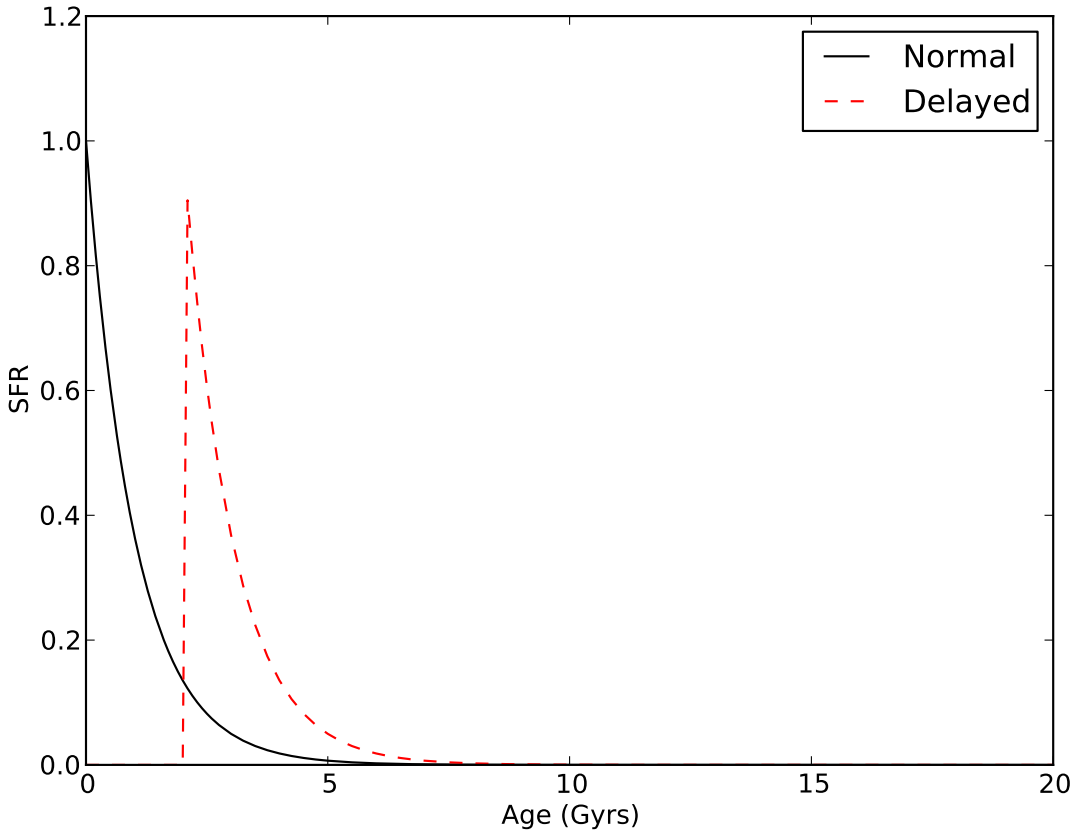
`EzGal` objects also have a few methods which are wrappers around the `ezgal.ezgal.make_csp()` method and implement simple star formation histories: `ezgal.ezgal.make_exponential()`, `ezgal.ezgal.make_burst()`, and `ezgal.ezgal.make_numeric()`. `ezgal.ezgal.make_exponential()` uses the same star formation history function as in the above example and is simply passed the e-folding time scale in gyrs. `ezgal.ezgal.make_burst()` returns CSPs corresponding to a burst of constant star formation of fixed length, and should be passed the length of star formation in gyrs. Finally, `ezgal.ezgal.make_numeric()` allows you to use an array of ages (in gyrs) and star formation rates to define an arbitrary star formation history:

```
import ezgal
model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )

exp_1 = model.make_exponential( 1.0 ) # same result as previous example

# generate the same CSP but now using an array of ages (in Gyrs) and star formation rates
new = model.make_numeric( exp_1.ages/1e9, exp_1.sfh )
```

3.2 Adding Delays Before Star Formation Begins

The `ezgal.ezgal.make_delayed()` method returns a new CSP with a delay before the onset of star formation. Simply pass the desired delay (in gyrs) before the onset of star formation. `ezgal.ezgal.make_delayed()` uses interpolation to move the SED grid to later ages

```
import ezgal
from pylab import *

# load a csp model: 1 gyr exponential burst
exp_1 = ezgal.model( 'bc03_exp_1.0_z_0.02_chab.model' )

# and then set a 2 gyr delay before the onset of star formation
delayed = exp_1.make_delayed( 2.0 )

# plot age/sfr relationship for both models
plot( exp_1.ages/1e9, exp_1.sfh, 'k-', label='Normal' )
plot( delayed.ages/1e9, delayed.sfh, 'r--', label='Delayed' )
xlabel( 'Age (Gyrs)' )
ylabel( 'SFR' )
legend()

show()
```

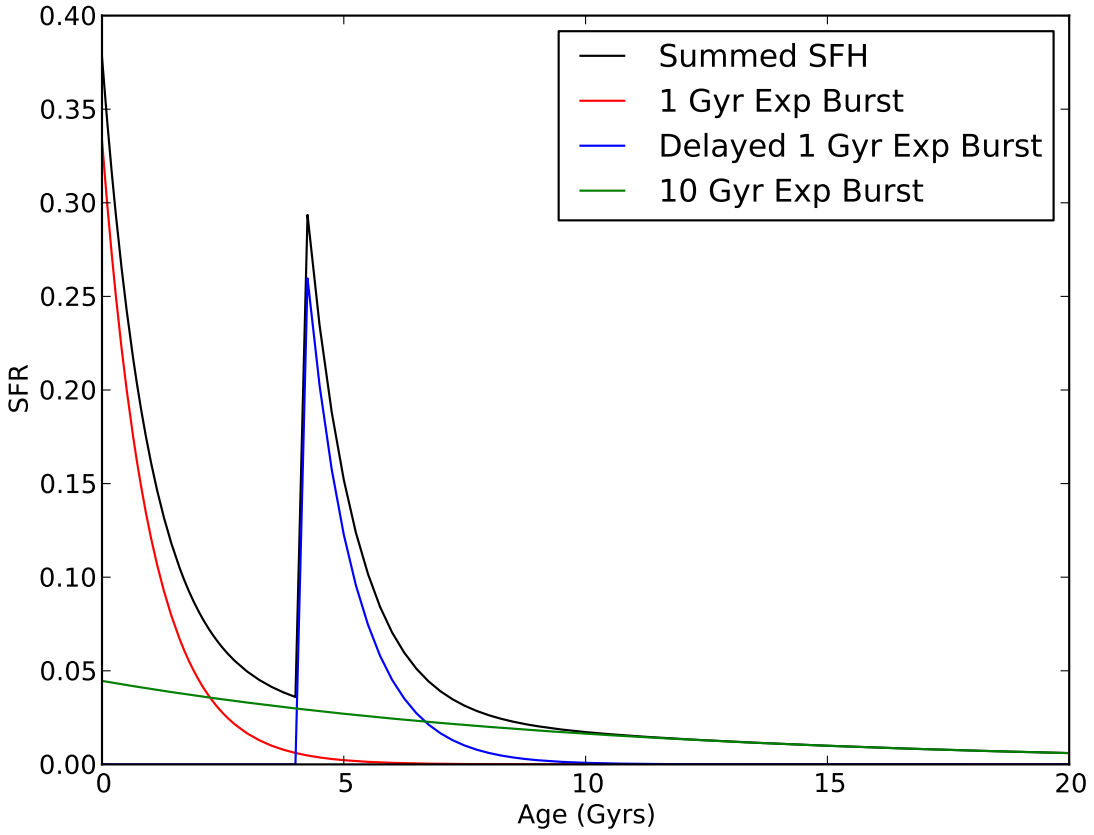


3.3 Combining Multiple Bursts of Star Formation

Models can be added together to make star formation histories with multiple bursts. This is done by loading or creating `EzGal` objects with the desired star formation histories, and then simply adding them together in python. Any number of models can be added together. Without weights specified each model will contribute equally to the final CSP, even

if they weren't added together in the same line of code. The new CSP models generated can be used like any other `EzGal` object: observables can be fetched (see *Calculating Observables*), the model can be saved, etc.

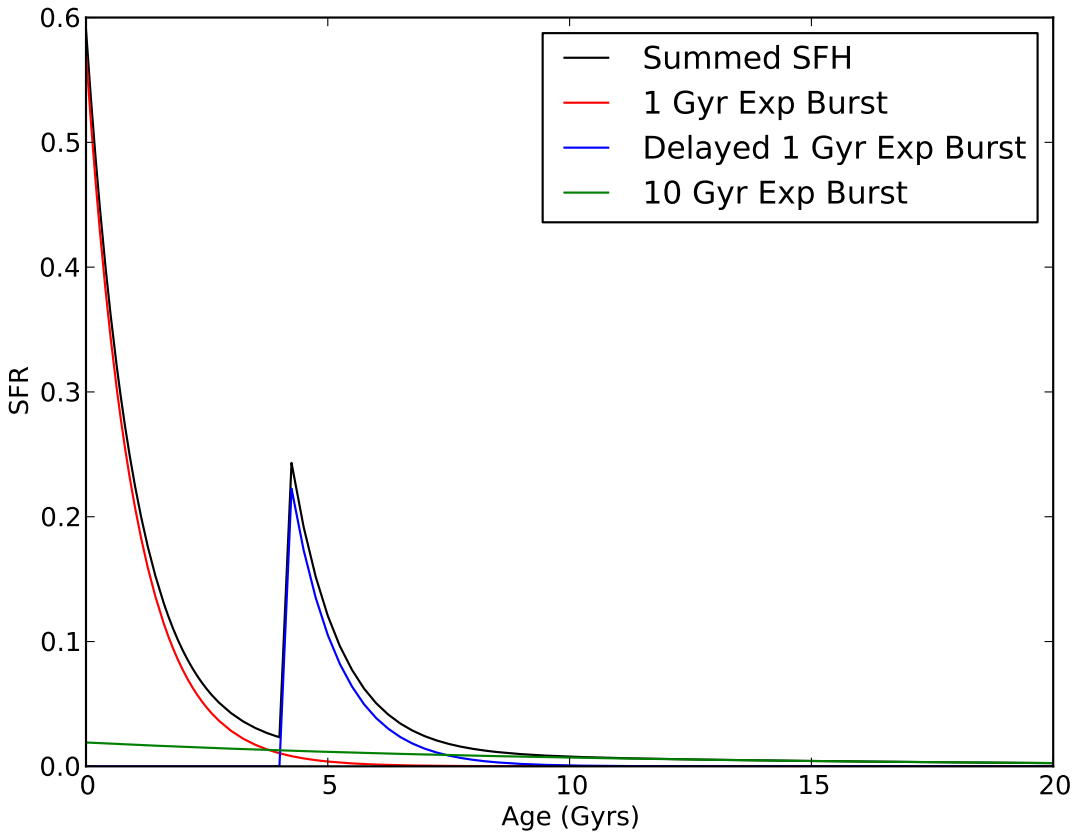
```
import ezgal
from pylab import *

exp_1 = ezgal.model( 'bc03_exp_1.0_z_0.02_chab.model' )
exp_10 = ezgal.model( 'bc03_exp_10.0_z_0.02_chab.model' )

# create a new model which is the sum of a 1 gyr burst and a delayed 1 gyr burst
delayed = exp_1.make_delayed( 4.0 )
complicated = exp_1 + delayed

# and the 10 gyr exponential burst in place
complicated += exp_10

# plot SFH of summed model
plot( complicated.ages/1e9, complicated.sfh, 'k-', label='Summed SFH' )

# plot SFH of individual models
# divide SFH of each by 3 because EzGal automatically weighted each model by 1/3
plot( exp_1.ages/1e9, exp_1.sfh/3, 'r-', label='1 Gyr Exp Burst' )
plot( delayed.ages/1e9, delayed.sfh/3, 'b-', label='Delayed 1 Gyr Exp Burst' )
plot( exp_10.ages/1e9, exp_10.sfh/3, 'g-', label='10 Gyr Exp Burst' )
xlabel( 'Age (Gyrs)' )
ylabel( 'SFR' )
legend()

show()
```



Weighting is done with the `weight` class, which is used to apply multiplicative factors during model addition. When creating a `weight` object simply pass the weight that a model should be multiplied by during addition. When adding models simply multiply them by `weight` objects to set the weights they should have in the final summed model.

```
import ezgal
from pylab import *

exp_1 = ezgal.model( 'bc03_exp_1.0_z_0.02_chab.model' )
exp_10 = ezgal.model( 'bc03_exp_10.0_z_0.02_chab.model' )

# created a delayed 1 gyr exponential burst
delayed = exp_1.make_delayed( 4.0 )

# add together bursts with various weights
complicated = ezgal.weight( 4.0 )*exp_1
complicated += ezgal.weight( 2.0 )*delayed
complicated += exp_10

# plot SFH of summed model
plot( complicated.ages/1e9, complicated.sfh, 'k-', label='Summed SFH' )

# plot SFH of individual models, weighted as above
plot( exp_1.ages/1e9, exp_1.sfh*4/7, 'r-', label='1 Gyr Exp Burst' )
plot( delayed.ages/1e9, delayed.sfh*2/7, 'b-', label='Delayed 1 Gyr Exp Burst' )
plot( exp_10.ages/1e9, exp_10.sfh/7, 'g-', label='10 Gyr Exp Burst' )
xlabel( 'Age (Gyrs)' )
```

```
ylabel( 'SFR' )
legend()

show()
```

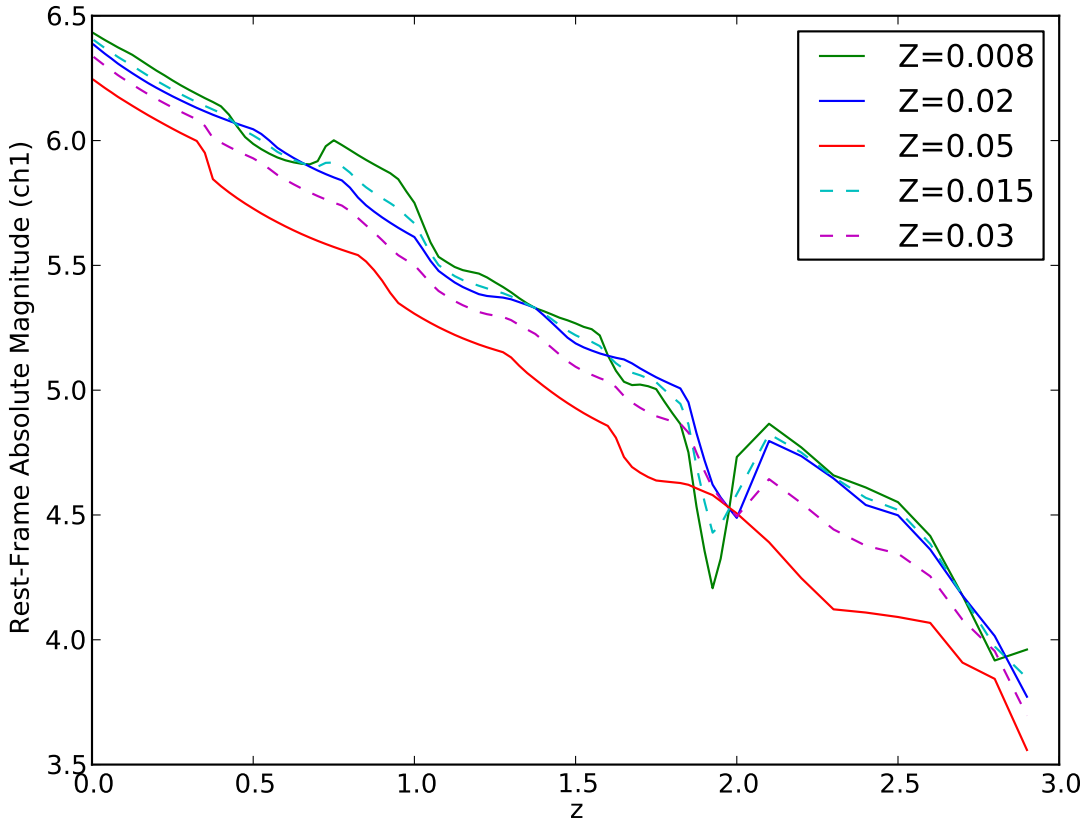


3.4 Interpolating Between Models

The `EzGal` module has a function, `interpolate`, which can be used to interpolate between models. The calling sequence for `interpolate` is the same as the interpolation functions in `np` and `scipi`. Pass a list of values to interpolate at, a list of values to interpolate between, and finally the corresponding models (either objects or filenames). `ezgal.interpolate` will return a list of models. `EzGal` uses numpy.interp to perform the interpolation.

```
import ezgal
from pylab import *

# load three different metallicty models: Z=0.008, Z=0.02, z=0.05
z008 = ezgal.model( 'bc03_ssp_z_0.008_chab.model' )
z02 = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
z05 = ezgal.model( 'bc03_ssp_z_0.05_chab.model' )

# interpolate between models to create Z=0.015 and Z=0.03 models
( z015, z03 ) = ezgal.interpolate( [0.015,0.03], [0.008,0.02,0.05], [z008,z02,z05] )

# plot absolute magnitude versus redshift in ch1 for original models
```

```
zs = z02.get_zs( 3.0 )
plot( zs, z008.get_absolute_mags( 3.0, filters='ch1', zs=zs ), 'g-', label='Z=0.008' )
plot( zs, z02.get_absolute_mags( 3.0, filters='ch1', zs=zs ), 'b-', label='Z=0.02' )
plot( zs, z05.get_absolute_mags( 3.0, filters='ch1', zs=zs ), 'r-', label='Z=0.05' )

# and then for the interpolated models
plot( zs, z015.get_absolute_mags( 3.0, filters='ch1', zs=zs ), 'c--', label='Z=0.015' )
plot( zs, z03.get_absolute_mags( 3.0, filters='ch1', zs=zs ), 'm--', label='Z=0.03' )

xlabel( 'z' )
ylabel( 'Rest-Frame Absolute Magnitude (ch1)' )
legend()

show()
```



If meta data (see `ezgal.ezgal.set_meta_data()`) has been set for the objects, then you can simply specify a meta data key for which `EzGal` should fetch the model values and use for interpolation. All `EzGal` model files downloaded off the website have meta data set. In particular, the metallicity is stored in the `met` key. Therefore the following code snippet will generate the exact same interpolate models as the above example:

```
import ezgal
( z015, z03 ) = ezgal.interpolate( [0.015,0.03], ['bc03_ssp_z_0.008_chab.model','bc03_ssp_z_0.02_chab
```

`ezgal.interpolate` makes use of *Handling Multiple Models* to handle the interpolation, so you can set `return_wrapper=True` to return the interpolated models in a wrapper object instead of a list.

CHAPTER

FOUR

HANDLING MULTIPLE MODELS

4.1 The Wrapper Class

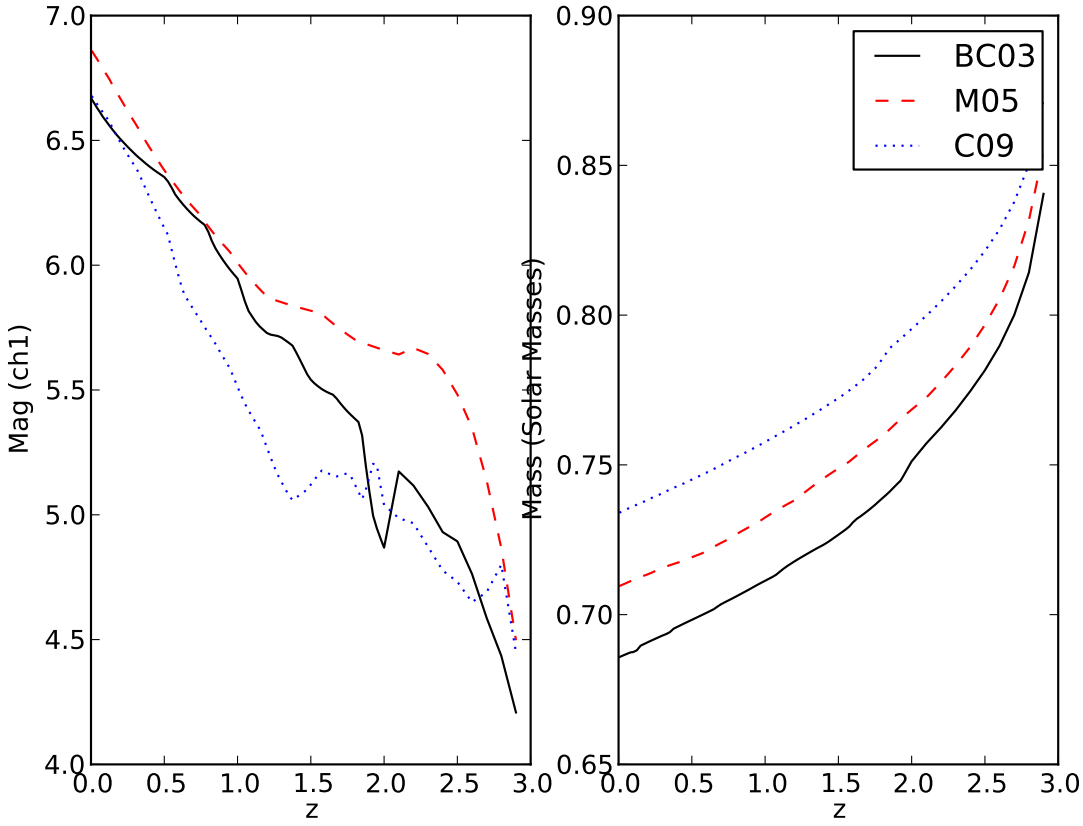
`ezgal.wrapper` is a class designed for working with multiple `EzGal` models simultaneously. It can calculate and return observables for many models in a large data cube with just one method call, sort model lists, search through models, interpolate between them, and return meta data. This is especially useful for calculating differences between models as a function of redshift, a fact used heavily in the plots for the EzGal paper. `ezgal.wrapper` objects are initialized with a list of EzGal-compatible model filenames or EzGal objects. All the corresponding model objects will then be stored in the `wrapper` object. Model objects can then be accessed by either looping over the `wrapper` object or through indexing. Indexing with lists is allowed. In addition, any meta data (see `ezgal.ezgal.set_meta_data()`) is copied into the wrapper object, and can be accessed by indexing the `wrapper` object with the meta data key. Model files downloaded through the `EzGal` website come with meta data already set, including 'model', 'met', 'sfh', and 'imf':

```
>>> import ezgal,os
>>> wrapper = ezgal.wrapper( ['bc03_ssp_z_0.008_chab.model','bc03_ssp_z_0.02_chab.model','bc03_ssp_z_
>>> len( wrapper )
3
>>> print os.path.basename( wrapper[0].filename )
bc03_ssp_z_0.008_chab.model

>>> for ezgal_object in wrapper:
>>>     print os.path.basename( ezgal_object.filename )
bc03_ssp_z_0.008_chab.model
bc03_ssp_z_0.02_chab.model
bc03_ssp_z_0.05_chab.model

>>> for ezgal_object in wrapper[[1,2,0]]:
>>>     print os.path.basename( ezgal_object.filename )
bc03_ssp_z_0.02_chab.model
bc03_ssp_z_0.05_chab.model
bc03_ssp_z_0.008_chab.model

>>> print wrapper.meta_data
{'imf': ['Chabrier', 'Chabrier', 'Chabrier'],
 'met': ['0.008', '0.02', '0.05'],
 'model': ['BC03', 'BC03', 'BC03'],
 'sfh': ['SSP', 'SSP', 'SSP']}

>>> print wrapper['met']
['0.008', '0.02', '0.05']
```

4.2 Calculating Observables

Most of the methods for fetching observables from `EzGal` objects are also available through the `wrapper` class (see *Wrapper API* for the full list). When called from a `wrapper` object a data cube is typically returned with a shape of `(len( wrapper ), len( filters ), len( zs ))`.

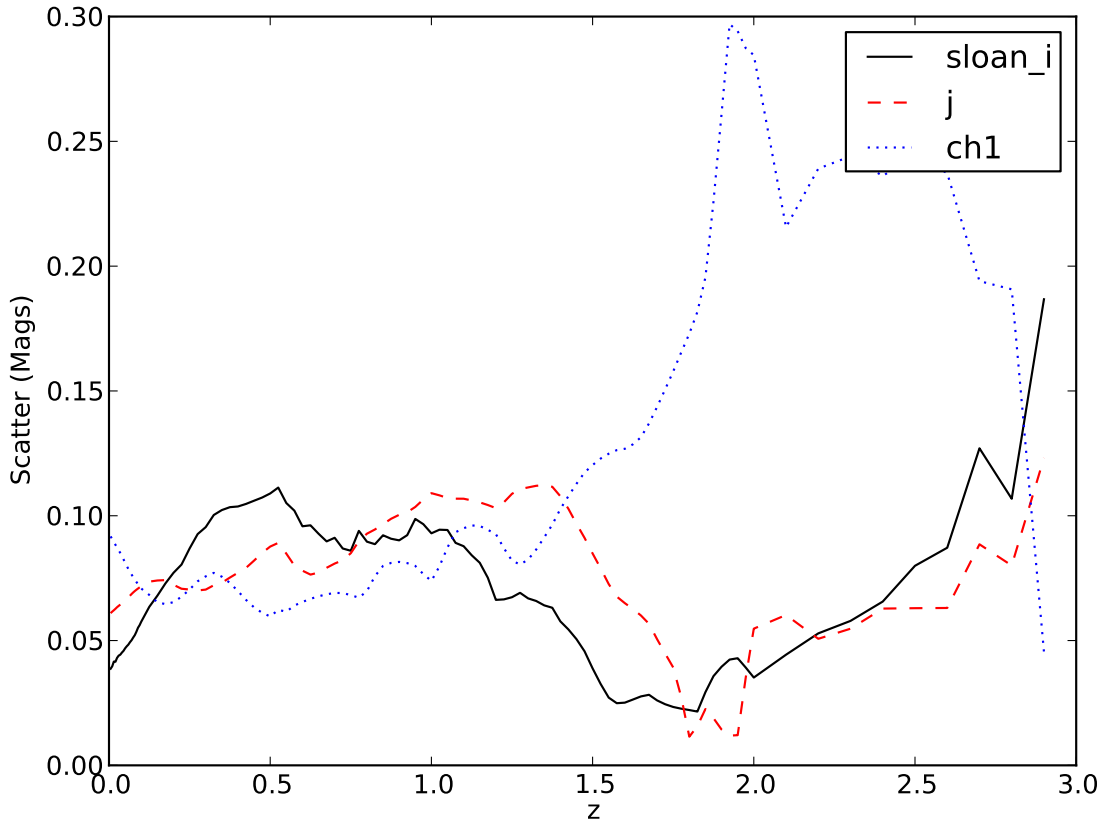
```
import ezgal
from pylab import *

# load ezgal model files
wrapper = ezgal.wrapper( ['bc03_ssp_z_0.02_salp.model','m05_ssp_z_0.02_salp.model','c09_ssp_z_0.02_sa

# desired formation redshift
zf = 3.0
# fetch an array of redshifts out to given formation redshift
zs = wrapper.get_zs( zf )

# fetch absolute mag vs redshift for all models in ch1
mags = wrapper.get_absolute_mags( zf, filters='ch1', zs=zs )

# set a line style for each model
syms = ['k-','r--','b:']
# and use the meta data for the line label
labels = wrapper['model']

# now loop through and plot
subplot( 1,2,1 )
for i in range( len( wrapper ) ):
        plot( zs, mags[i,:,0], syms[i], label=labels[i] )
xlabel( 'z' )
ylabel( 'Mag (ch1)' )

# also plot mass versus redshift
subplot( 1,2,2 )
masses = wrapper.get_masses( zf, zs )
for i in range( len( wrapper ) ):
        plot( zs, masses[i,:,0], syms[i], label=labels[i] )
xlabel( 'z' )
ylabel( 'Mass (Solar Masses)' )

# show the legend
legend()

# all done
show()
```



By returning everything in a data cube, it becomes easy to calculate differences between models, especially when making use of numpy functions for working with arrays. For instance, a couple lines can calculate the scatter between the predicted magnitudes of different models:

```
import ezgal
import numpy as np
from pylab import *

# load ezgal model files
wrapper = ezgal.wrapper( ['bc03_ssp_z_0.02_salp.model','m05_ssp_z_0.02_salp.model','c09_ssp_z_0.02_sa

# desired formation redshift
zf = 3.0
# fetch an array of redshifts out to given formation redshift
zs = wrapper.get_zs( zf )

# filters to calculate the scatter for
filters = ['sloan_i','j','ch1']
# and a plotting symbol for each filter
syms = ['k-','r--','b:']

# fetch the apparent magnitudes for all models through the above filters
mags = wrapper.get_apparent_mags( zf, filters=filters, zs=zs )

# now calculate the scatter between models through each filter and at each redshift
# numpy makes this very easy, as the zeroth axis of the data cube depends on model
```

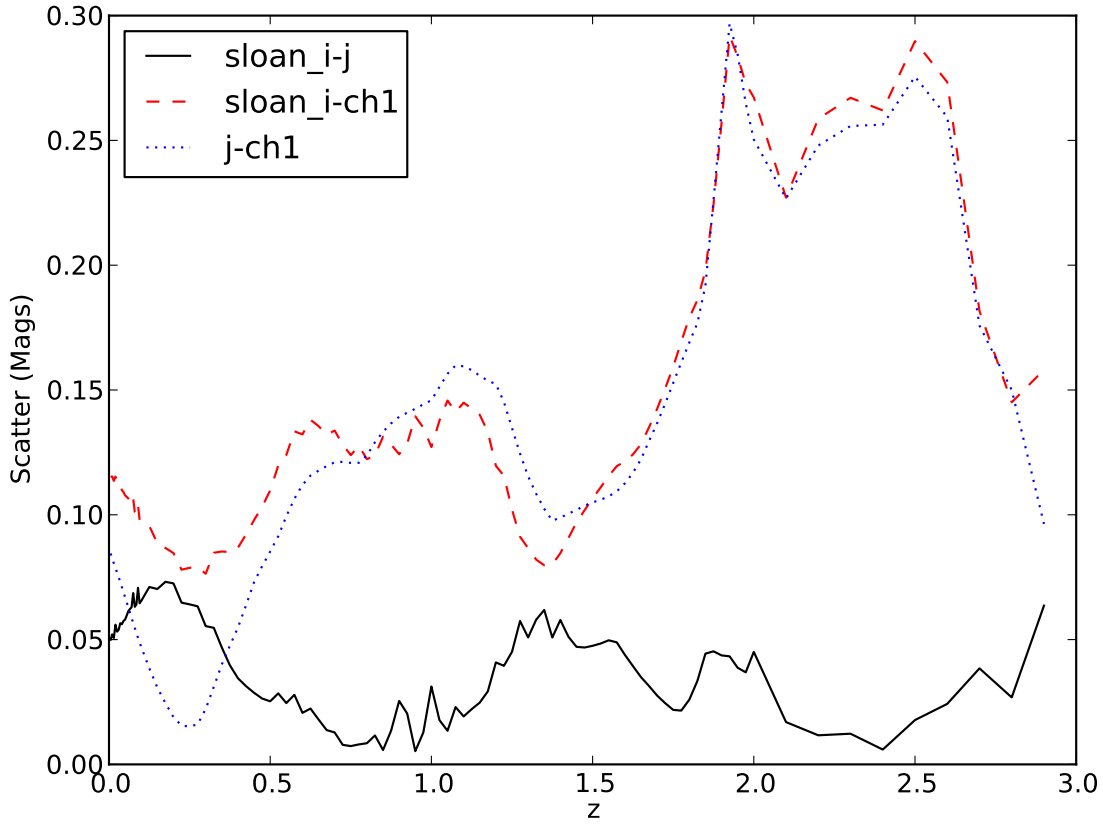
```
scatter = np.std( mags, axis=0 )

# now loop through and plot
for i in range( len( filters ) ):
        plot( zs, scatter[:,i], syms[i], label=filters[i] )
xlabel( 'z' )
ylabel( 'Scatter (Mags)' )

# show the legend
legend()

# all done
show()
```



Similarly for colors:

```
import ezgal
import numpy as np
from pylab import *

# load ezgal model files
wrapper = ezgal.wrapper( ['bc03_ssp_z_0.02_salp.model','m05_ssp_z_0.02_salp.model','c09_ssp_z_0.02_sa

# desired formation redshift
zf = 3.0
# fetch an array of redshifts out to given formation redshift
```

```
zs = wrapper.get_zs( zf )

# colors to calculate the scatter for
colors = [ ['sloan_i','j'],
           ['sloan_i','ch1'],
           ['j','ch1'] ]
# and a plotting symbol for each color
syms = ['k-','r--','b:']

# loop through each color
for ( filters, sym ) in zip ( colors, syms ):

        # calculate color for all models
        colors = wrapper.get_apparent_mags( zf, filters=filters[0], zs=zs ) - wrapper.get_apparent_ma

        # calculate scatter
        scatter = np.std( colors, axis=0 )

        # and plot
        plot( zs, scatter[:,0], sym, label='%s-%s' % (filters[0],filters[1]) )

# axis labels
xlabel( 'z' )
ylabel( 'Scatter (Mags)' )

# show the legend
legend( loc='upper left' )

# all done
show()
```



Most of the methods available for fetching observables from `EzGal` objects are also available for wrapper objects. Namely:

Method	Returns
`ezgal.wrapper.get_absolute_mags()`	Rest-frame absolute magnitudes as a function of filter, zf, and z
`ezgal.wrapper.get_age()`	Age as a function of zf and z
`ezgal.wrapper.get_apparent_mags()`	Apparent magnitude as a function of filter, zf, and z
`ezgal.wrapper.get_distance_moduli()`	Distance moduli as a function of z
`ezgal.wrapper.get_ecorrects()`	e-corrections as a function of filter, zf, and z
`ezgal.wrapper.get_ekcorrects()`	e+k corrections as a function of filter, zf, and z
`ezgal.wrapper.get_kcorrects()`	k-corrections as a function of filter, zf, and z
`ezgal.wrapper.get_masses()`	Mass as a function of zf and z
`ezgal.wrapper.get_normalization()`	The model normalization, if set
`ezgal.wrapper.get_observed_absolute_mags()`	Observed-frame absolute magnitudes as a function of filter, zf, and z
`ezgal.wrapper.get_observed_ml_ratios()`	Observed-frame M/L ratios as a function of filter, zf, and z
`ezgal.wrapper.get_rest_ml_ratios()`	Rest-frame M/L ratios as a function of filter, zf, and z
`ezgal.wrapper.get_solar_observed_mags()`	Observed-frame magnitude of the sun as a function of filter and z
`ezgal.wrapper.get_zs()`	A list of redshifts out to some z

4.3 Searching, Indexing, Adding, and Sorting

Because `wrapper` objects are fully index-able, it is possible to search on the meta data using standard numpy functions and return sorted or indexed wrapper lists. This could be particularly handy if you have a script that uses many models, but doesn't use them all at one time. You can also combine (using the addition operator) wrapper lists:

```
>>> import ezgal,os
>>> import numpy as np
>>> wrapper = ezgal.wrapper( ['bc03_ssp_z_0.008_chab.model','bc03_ssp_z_0.02_chab.model','bc03_ssp_z_
>>> wrapper += ezgal.wrapper( ['bc03_ssp_z_0.008_salp.model','bc03_ssp_z_0.02_salp.model','bc03_ssp_z
>>> wrapper += ezgal.wrapper( ['m05_ssp_z_0.008_salp.model','m05_ssp_z_0.02_salp.model','m05_ssp_z_0
>>> len( wrapper )
9

>>> # Use meta data to find all models with solar metallicity and a salpeter IMF
>>> w = np.where( (wrapper['met'] == '0.02') & (wrapper['imf'] == 'Salpeter') )
>>> solar = wrapper[w]
>>> for ezgal_object in solar:
>>>     print os.path.basename( ezgal_object.filename )
bc03_ssp_z_0.02_salp.model
m05_ssp_z_0.02_salp.model
```

`wrapper` objects have sort and argsort methods for sorting by meta data. Both methods assume that the meta data keys you are sorting by are floats, and convert the meta data to floats before sorting. The `ezgal.wrapper.sort()` method returns a sorted copy of the wrapper object, while the `ezgal.wrapper.argsort()` method returns a list of indexes for sorting the wrapper object. Both are passed the name of a meta key to sort by:

```
>>> import ezgal,os
>>> wrapper = ezgal.wrapper( ['bc03_ssp_z_0.03_chab.model','bc03_ssp_z_0.008_chab.model','bc03_ssp_z_
>>> wrapper.argsort()
array([1, 2, 0])

>>> print wrapper['met']
['0.03' '0.008' '0.02']
>>> new = wrapper.sort( 'met' )
>>> print new['met']
['0.008' '0.02' '0.03']
```

CHAPTER

FIVE

API

Contents:

5.1 EzGal API

5.1.1 EzGal Parameters

Every `EzGal` object has a number of available parameters which store model information. Those parameters are:

Parameter	Value
filename	The full path to the loaded model file
nages	The number of ages in the SED grid.
ages	An array giving the age (in years) of every column in `seds`.
nvs	The number of frequencies in the SED grid.
vs	An array giving the frequency (in Hertz) of every row in `seds`.
nls	The number of wavelengths in the SED grid. Note: nls == nvs.
ls	An array giving the wavelength (in Angstroms) of every row in `seds`.
seds	The SED grid. Stored as an array of shape `( nvs, nages )`.
has_masses	Whether or not the model contains an age-mass relationship.
masses	An array containing the stellar mass (in solar masses) for every age in `ages`.
has_sfh	Whether or not the model contains a star formation history.
sfh	For CSPs, an array containing the star formation rate for every age in `ages`.
zfs	An array of the current formation redshifts for which magnitude evolution will be precalculated when filters are added.
nzfs	`len( zfs )`
filters	A dictionary containg the `astro_filter` objects for each loaded filter.
filter_order	The list of current filters, in the order in which they are returned by the various methods for fetching observables.
meta_data	Ancillary model information, stored as a dictionary and set with `ezgal.ezgal.set_meta_data()`.
model_weight	The weight of the model when added with other models to make CSPs.

For example:

```
>>> import ezgal
>>> model = ezgal.ezgal( 'bc03_ssp_z_0.02_chab.model' )
>>> ( model.nls, model.nages )
(6900, 221)
>>> model.seds.shape
(6900, 221)
```

```
>>> model.add_filter( 'ch1' )
>>> model.add_filter( 'ch2' )
>>> model.filter_order
['ch1', 'ch2']
>>> model.filters
{'ch1': <ezgal.astro_filter.astro_filter object at 0x1474d50>,
 'ch2': <ezgal.astro_filter.astro_filter object at 0x23703d0>}
```

5.1.2 ezgal.add_filter()

ezgal.**add_filter**(*file*, *name=None*, *units='a'*, *grid=True*)

Parameters

- **file** (*string*) – The filename containing the filter response curve
- **name** (*string*) – The name to store the filter as
- **units** (*string*) – The length units for the wavelengths in the file
- **grid** (*bool*) – Whether or not to calculate evolution information when first added

Add a filter for calculating models. Specify the name of the file containing the filter transmission curve. If the file is not found then `EzGal` will search for it in the directory specified by the `EZGAL_FILTERS` environment variable, and then in the `data/filters` directory in the `EzGal` module directory.

The filter file should have two columns (wavelength,transmission). Wavelengths are expected to be in angstroms unless specified otherwise with `units`. See `ezgal.utils.to_meters()` for list of available units.

Specify a name to refer to the filter as later. If no name is specified, the filename is used (excluding path information and extension) If a filter already exists with that name, the previous filter will be replaced.

If grid is True, then models will be generated for this filter at all set formation redshifts.

You can pass a numpy array directly, instead of a file, but if you do this you need to specify the name.

5.1.3 ezgal.copy()

ezgal.**copy**()

Returns A copy of the model

Return type `EzGal` model object

Example

```
>>> import ezgal
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> model_copy = model.copy()
```

Returns a copy of the `EzGal` object without any stored filter information.

5.1.4 ezgal.get_age()

ezgal.**get_age**(*z1*, *z2*, *units='gyrs'*)

Parameters

- **z1** (*int, float*) – The first redshift

- **z2** (*int, float, list, array*) – The second redshift
- **units** (*str*) – The units to return the time in

Returns Time between two redshifts

Return type int, float, list, array

Example

```
>>> import ezgal
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> print model.get_age( 3.0, [0.0,0.5,1.0] )
[ 11.55546768   6.49725355   3.76705514]
```

returns the time difference between z1 and z2. z2 can be a list or numpy array. Primarily used to get the age of a galaxy at redshift z2, given formation redshift z1. See `ezgal.utils.to_years()` for available age units.

5.1.5 ezgal.get_absolute_mags()

`ezgal.`**`get_absolute_mags`**(*zf*, *filters=None*, *zs=None*, *normalize=True*, *ab=None*, *vega=None*, *squeeze=True*)

Parameters

- **zf** (*int, float*) – The formation redshift of the galaxy
- **filters** (*string, list of strings*) – A list of filters to calculate magnitudes for
- **zs** (*int, float, list, array*) – A list of redshifts to calculate magnitudes at
- **normalize** (*bool*) – Whether or not to normalize the output
- **ab** (*bool*) – Whether or not to output in AB magnitues
- **vega** (*bool*) – Whether or not to output in Vega magnitudes

Returns Array of magnitudes

Return type array

Example

```
>>> import ezgal
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> model.add_filter( 'ch1' )
>>> model.add_filter( 'ch2' )
>>> model.add_filter( 'ch3' )
>>> model.add_filter( 'ch4' )
>>> model.get_absolute_mags( 3.0, zs=[0,1.0,2.0] )
array([[ 6.39000031  6.83946654  7.25469915  7.80111159]
       [ 5.61352032  6.07145866  6.49122232  7.04203427]
       [ 4.48814389  4.87218428  5.30970258  5.85536578]])
>>> model.get_absolute_mags( 3.0, filters=['ch1','ch2'], zs=[0,1.0,2.0] )
array([[ 6.39000031,  6.83946654],
       [ 5.61352032,  6.07145866],
       [ 4.48814389,  4.87218428]])
>>> model.get_absolute_mags( 3.0, filters='ch1', zs=[0,1.0,2.0] )
array([ 6.39000031  5.61352032  4.48814389])
>>> model.get_absolute_mags( 3.0, filters='ch1', zs=0 )
6.39000030832
```

Fetch the rest-frame absolute magnitudes for the given filter(s) at the given output redshifts (`zs`) for the given formation redshift (`zf`). If no filter is specified, then magnitudes will be fetched for all loaded filters. If the specified filters are not already loaded into `EzGal`, it will attempt to load them automatically according to the rules in `ezgal.ezgal.add_filter()`.

If normalize=True then the returned magnitudes will be normalized if a normalization has been set with `ezgal.ezgal.set_normalization()`.

Returns an array of shape `(len(zs),len(filters))` with the absolute magnitudes. If there is only one filter, then it returns an array of shape `(len(zs))`

Call with vega=True or ab=True to specify ab or vega output. If neither is set it will output AB mags unless `ezgal.ezgal.set_vega_output()` has been called.

Note: If no output redshifts are specified, then the redshifts in ezgal.zs will be used.

Note: This is the equiavlent to `M_AB_ev - k_cor_ev` from the output of the bruzual and charlot program `cm_evolution`

See Also:

`ezgal.ezgal.set_normalization()`, `ezgal.ezgal.set_vega_output()`, `ezgal.ezgal.set_ab_output()`

5.1.6 ezgal.get_apparent_mags()

`ezgal.`**`get_apparent_mags`**(*zf*, *filters=None*, *zs=None*, *normalize=True*, *vega=None*, *ab=None*, *squeeze=True*)

Same as `ezgal.ezgal.get_absolute_mags()` excepts returns apparent magnitudes.

Note: This is equivalent to the apparent magnitude field `m_AB_ev` found in the output of the bruzual and charlot program `cm_evolution`.

See Also:

`ezgal.ezgal.get_absolute_mags()`

5.1.7 ezgal.get_distance_moduli()

`ezgal.`**`get_distance_moduli`**(*zs=None*, *nfilters=None*, *squeeze=True*)

Parameters

- **zs** (*int, float, list, array*) – The redshifts to return distance moduli for
- **nfilters** (*int*) – The number of filters to return distance moduli for

Returns Distance Moduli

Return type array

Example

```
>>> import ezgal
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> print model.get_distance_moduli( [0.1, 0.5, 1.0], nfilters=1 )
array([ 38.30736787,  42.27018553,  44.1248392 ])
>>> print model.get_distance_moduli( [0.1, 0.5, 1.0], nfilters=2 )
array([[ 38.30736787,  38.30736787],
       [ 42.27018553,  42.27018553],
       [ 44.1248392 ,  44.1248392 ]])
```

Fetch the distance moduli for the given redshifts. Specify the number of filters to return an array of size (len(zs),nfilters). If nfilters is None, then the number of filters will be taken to be the number of filters loaded in the object. If there is only one filter, then it returns an array of shape (len(zs)). The distance moduli does not depend on filter so values are repeated when `nfilters>1`. This is done for consistency with the return values of methods like `ezgal.ezgal.get_apparent_mags()`.

Note: If no output redshifts are specified, then the redshifts in ezgal.zs will be used.

5.1.8 ezgal.get_ecorrects()

`ezgal.`**`get_ecorrects`**(*zf*, *filters=None*, *zs=None*, *squeeze=True*)

Parameters

- **zf** (*int, float*) – The formation redshift
- **filters** (*string, list of strings*) – The filters to calculate e-corrections for
- **zs** (*int, float, list, array*) – The redshifts to calculate e-corrections at

Returns The ecorrections

Return type array

Example

```
>>> import ezgal
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> model.add_filter( 'ch1' )
>>> model.add_filter( 'ch2' )
>>> model.add_filter( 'ch3' )
>>> model.add_filter( 'ch4' )
>>> model.get_ecorrects( 3.0, zs=[0,0.5,1.0] )
array([[ 0.        ,  0.        ,  0.        ,  0.        ],
       [-0.34465634, -0.32867488, -0.32408269, -0.32054988],
       [-0.77647999, -0.76800788, -0.76347683, -0.75907732]])
>>> model.get_ecorrects( 3.0, filters=['ch1','ch2'], zs=[0,0.5,1.0] )
array([[ 0.        ,  0.        ],
       [-0.34465634, -0.32867488],
       [-0.77647999, -0.76800788]])
>>> model.get_ecorrects( 3.0, filters='ch1', zs=[0,0.5,1.0] )
array([ 0.        , -0.34465634, -0.77647999])
>>> model.get_ecorrects( 3.0, filters='ch1', zs=0 )
0.0
```

Fetch the e-corrections for the given filter(s) at the given output redshifts (`zs`) for the given formation redshift (`zf`). If no filter is specified, then magnitudes will be fetched for all loaded filters. If the specified filters are not already loaded into `EzGal`, it will attempt to load them automatically according to the rules in `ezgal.ezgal.add_filter()`.

returns an array of shape `(len(zs),len(filters))` with the kcorrects. If there is only one filter, then it returns an array of shape `(len(zs))`

Note: If no redshifts are specified, then the redshifts in `ezgal.zs` will be used.

5.1.9 ezgal.get_ekcorrects()

`ezgal.`**`get_ekcorrects`**(*zf*, *filters=None*, *zs=None*)

Same as `ezgal.ezgal.get_ecorrects()` but returns the e+k corrections.

Note: This is the equiavlent to `e+k_cor` from the output of the bruzual and charlot program `cm_evolution`.

See Also:

`ezgal.ezgal.get_ecorrects()`

5.1.10 ezgal.get_kcorrects()

`ezgal.`**`get_kcorrects`**(*zf*, *filters=None*, *zs=None*, *squeeze=True*)

Same as `ezgal.ezgal.get_ecorrects()` but returns the kcorrections.

Note: This is the equiavlent to `k_cor_ev` from the output of the bruzual and charlot program `cm_evolution`.

See Also:

`ezgal.ezgal.get_ecorrects()`

5.1.11 ezgal.get_mass_weighted_ages()

`ezgal.`**`get_mass_weighted_ages`**(*zf*, *zs=None*, *units='gyrs'*)

Parameters

- **zf** (*int, float*) – The formation redshift
- **zs** (*int, float, list, array*) – The redshifts to calculate ages at
- **units** (*string*) – The units to return ages in

Returns The mass weighted ages

Return type int, float, array

Example

```
>>> import ezgal
>>> csp = ezgal.model( 'bc03_exp_10.0_z_0.02_chab.model' )
>>> csp.get_mass_weighted_ages( 3.0, [0, 1.0, 2.0, 2.5] )
array([ 10.6436012 ,   3.47012948,   1.079878  ,   0.44172018])
>>> # compare to the time between zf and zs:
>>> csp.get_age( 3.0, [0, 1.0, 2.0, 2.5] )
array([ 11.55546768,   3.76705514,   1.1586272 ,   0.47989398])
```

Returns the mass weighted age as a function of redshift and formation redshift. Set units of returned ages with `units`

Warning: Only works for CSP models generated with `EzGal`.

5.1.12 ezgal.get_masses()

`ezgal.`**`get_masses`**(*zf*, *zs*, *nfilters=None*, *normalize=True*, *squeeze=True*)

Parameters

- **zf** (*int, float*) – The formation redshift
- **zs** (*int, float, list, array*) – Redshifts to calculate masses at
- **nfilters** (*int*) – The number of filters to return masses for
- **normalize** (*bool*) – Whether or not to normalize the returned masses

Returns The mass in solar masses

Return type array

Example

```
>>> import ezgal
>>> model.add_filter( 'ch1' )
>>> model.add_filter( 'ch2' )
>>> model.add_filter( 'ch3' )
>>> model.add_filter( 'ch4' )
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> model.get_masses( 3.0, zs=[0, 1, 2] )
array([[ 0.50909797,  0.50909797,  0.50909797,  0.50909797],
       [ 0.54490264,  0.54490264,  0.54490264,  0.54490264],
       [ 0.60169807,  0.60169807,  0.60169807,  0.60169807]])
>>> model.get_masses( 3.0, zs=[0, 1, 2], nfilters=2 )
array([[ 0.50909797,  0.50909797],
       [ 0.54490264,  0.54490264],
       [ 0.60169807,  0.60169807]])
>>> model.get_masses( 3.0, zs=[0, 1, 2], nfilters=1 )
array([ 0.50909797,  0.54490264,  0.60169807])
>>> model.get_masses( 3.0, zs=0, nfilters=1 )
0.50909797135560608
```

Get the stellar mass (in solar masses) as a function of `zf` and `z`. Specify the number of filters to return an array of size `(len(zs),nfilters)`. If `nfilters` is None, then the number of filters will be taken to be the number of filters loaded in the object. If there is only one filter, then it returns an array of shape `(len(zs))`. The mass is independent of filter, so masses are repeated across filters. This is done for consistency with the return type of methods like `ezgal.ezgal.get_absolute_mags()`.

Note: If no output redshifts are specified, then the redshifts in ezgal.zs will be used.

5.1.13 ezgal.get_normalization

`ezgal.`**`get_normalization`**(*zf*, *flux=False*)

Parameters

- **zf** (*int, float*) – The formation redshift to assume
- **flux** (*bool*) – Wheter or not to return a multiplicative factor

Returns The normalization

Return type float

Example

```
>>> import ezgal
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> model.set_normalization( 'ch1', 0.24, -25.06, vega=True )
>>> model.get_normalization( 3.0 )
-28.45662555679969
>>> # when set, normalizations are always applied by default
>>> model.get_absolute_mags( 3.0, filters='ch1', zs=1.0 )
-22.843105234738616
>>> # manually apply normalization
>>> model.get_absolute_mags( 3.0, filters='ch1', zs=1.0, normalize=False ) + model.get_n
-22.843105234738616
>>> model.get_normalization( 3.0, flux=True )
241351622492.22711
>>> # calculate the mass (in solar masses) of the normalized galaxy at z=0.24 assuming z
>>> model.get_masses( 3.0, 0.24, nfilters=1 ) * model.get_normalization( 3.0, flux=True
124886846296.74387
```

Returns the model normalization given formation redshift. Returns normalization in units of magnitudes. Set flux=True to return a multiplicative factor for multiplying SEDs or masses. Returns 0 if no normalization has been set.

See Also:

`ezgal.ezgal.set_normalization()`

5.1.14 ezgal.get_observed_absolute_mags()

`ezgal.`**`get_observed_absolute_mags`**(*zf*, *filters=None*, *zs=None*, *normalize=True*, *vega=None*, *ab=None*, *squeeze=True*)

Same as `ezgal.ezgal.get_absolute_mags()` excepts returns observed-frame absolute magnitudes. The observed-frame absolute magnitude is the absolute magnitude of the model after being redshifted to the given redshift.

Note: This is the equiavlent to `M_AB_ev` from the output of the bruzual and charlot program `cm_evolution`

See Also:

`ezgal.ezgal.get_absolute_mags()`, `ezgal.ezgal.set_normalization()`, `ezgal.ezgal.set_vega_output()`, `ezgal.ezgal.set_ab_output()`

5.1.15 ezgal.get_observed_ml_ratios()

`ezgal.`**`get_observed_ml_ratios`**(*zf*, *filters=None*, *zs=None*)

Parameters

- **zf** (*int, float*) – The formation redshift
- **filters** (*string, list of strings*) – The filters to calculate M/L ratios for
- **zs** (*int, float, list, array*) – The redshifts to calculate M/L ratios at

Returns The M/L ratios

Return type array

Example

```
>>> import ezgal
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> model.add_filter( 'sloan_u' )
>>> model.add_filter( 'sloan_i' )
>>> model.add_filter( 'ch1' )
>>> model.get_observed_ml_ratios( 3.0, zs=[0,1.0,2.0] )
array([[ 7.64179498,  2.52235684,  0.6858418 ],
       [ 0.19372322,  2.24205801,  0.44824818],
       [         nan,  0.22833461,  0.28408678]])
```

Returns the observed-frame mass-to-light ratios as a function of `zf`, `z`, and `filters`. The observed-frame mass to light ratios is the stellar mass of the model given `zf` and `z` divided by the luminosity of the model in solar units. For observed-frame M/L ratios the observed-frame luminosity of the model and the observed-frame luminosity of the sun are used. The observed-frame luminosity of the sun is calculated by redshifting the solar spectrum to the given redshift and then projecting it through the filters. This returns an array of shape `(len(zs),len(filters))` with the observed-frame mass-to-light ratios. If there is only one filter, then it returns an array of shape `(len(zs))`.

Note: Returns `nan` when the redshifted solar spectrum doesn't fully cover the filter.

Warning: Only works if masses are set in the model.

5.1.16 ezgal.get_rest_ml_ratios()

`ezgal.`**`get_rest_ml_ratios`**(*zf*, *filters=None*, *zs=None*)

Same as `ezgal.ezgal.get_observed_ml_ratios()` excepts returns the rest-frame M/L ratios. Rest frame mass-to-light ratios are calculated using the rest-frame luminosity of the sun and the rest-frame luminosity of the model.

See Also:

`ezgal.ezgal.get_observed_ml_ratios()`

Warning: Only works if masses are set in the model.

5.1.17 ezgal.get_sed()

`ezgal.`**`get_sed`**(*age*, *age_units='gyrs'*, *units='Fv'*)

Parameters

- **age** (*int, float*) – The desired age of the SED

- **age_units** (*string*) – The units the age is in
- **units** (*string*) – The output units for the SED

Returns The SED

Return type array

Example

```
>>> import ezgal
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> sed = model.get_sed( 10, age_units='gyrs', units='Fl' )
>>> sed.size
6900
>>> model.ls.size, model.vs.size
(6900, 6900)
```

Returns the rest-frame sed of the galaxy at a given age. See `ezgal.utils.to_years()` for the available age units. Returns an array of size (model.nvs). The frequencies (wavelengths) of each point in the returned array correspond to the frequencies (wavelengths) in model.vs (model.ls).

Available output units are (case insensitive):

Name	Units
Fv	ergs/s/cm^2/Hz
Fl	ergs/s/cm^2/Angstrom
Flux	ergs/s/cm^2
Luminosity	ergs/s

Note: Uses linear interpolation between two nearest age models.

5.1.18 ezgal.get_sed_z()

`ezgal.`**`get_sed_z`**(*zf*, *z*, *units='Fv'*, *normalize=True*, *observed=False*, *return_frequencies=False*)

Parameters

- **zf** (*int, float*) – The formation redshift for the output SED
- **z** (*int, float*) – The redshift for the output SED
- **units** (*string*) – The output units for the SED
- **normalize** (*bool*) – Whether or not to normalize the output SED
- **observed** (*bool*) – Whether or not to output the observed-frame SED
- **return_frequencies** (*bool*) – Whether or not to return the corresponding frequencies

Returns The SED

Return type array

Example

```
>>> import ezgal
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> sed = model.get_sed_z( 3.0, 0.5, units='Fl' )
>>> sed.size
6900
```

```
>>> model.ls.size, model.vs.size
(6900, 6900)
>>> ( ls, sed ) = model.get_sed_z( 3.0, 0.5, units='Fl', return_frequencies=True )
```

Returns the rest-frame sed for a galaxy given its formation redshift (`zf`) and current redshift (`z`). If `normalize=True` and a normalization has been set with `ezgal.ezgal.set_normalization`'() then the sed will be normalized accordingly. If `observed=True` then the observed frame SED is returned. If `return_frequencies=True` then it also returns the frequencies of the points in the returned SED (or wavelength array in angstroms if `units='Fl'`). In this case a tuple is returned, with the first element being the list of frequencies, and the second element the SED. Otherwise, the frequencies (wavelengths) of each point in the returned array correspond to the frequencies (wavelengths) in model.vs (model.ls).

See Also:

See ezgal.get_sed() for available output units.

> **Warning:** The conversion to observed-frame has not been thoroughly tested, and I don't guarantee that I did it right. So please check it carefully before using.

5.1.19 ezgal.get_solar_observed_mags()

`ezgal.`**`get_solar_observed_mags`**(*zf*, *filters=None*, *zs=zs*, *ab=None*, *vega=None*, *squeeze=True*)

Parameters

- **zf** (*int, float*) – The formation redshift of the galaxy
- **filters** (*string, list of strings*) – A list of filters to calculate magnitudes for
- **zs** (*int, float, list, array*) – A list of redshifts to calculate magnitudes at
- **ab** (*bool*) – Whether or not to output in AB magnitues
- **vega** (*bool*) – Whether or not to output in Vega magnitudes

Returns Array of solar observed-frame magnitudes

Return type array

Example

```
>>> import ezgal
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> model.add_filter( 'ch1' )
>>> model.add_filter( 'ch2' )
>>> model.add_filter( 'ch3' )
>>> model.add_filter( 'ch4' )
>>> model.get_solar_observed_mags( 3.0, zs=[0,1.0,2.0] )
array([[ 6.06644384,  6.56615057,  7.0455159 ,  7.66857929],
       [ 4.05625676,  4.44971653,  4.89681114,  5.50676563],
       [ 3.35937661,  3.43865068,  3.72883098,  4.27894184]])
>>> model.get_solar_observed_mags( 3.0, filters=['ch1','ch2'], zs=[0,1.0,2.0] )
array([[ 6.06644384,  6.56615057],
       [ 4.05625676,  4.44971653],
       [ 3.35937661,  3.43865068]])
>>> model.get_solar_observed_mags( 3.0, filters='ch1', zs=[0,1.0,2.0] )
array([ 6.06644384,  4.05625676,  3.35937661])
>>> model.get_solar_observed_mags( 3.0, filters='ch1', zs=0 )
6.0664438415489164
```

Return the observed-frame absolute magnitude of the sun given `zf`, `filters`, and `zs`. Returns an array of size (zs.size,filters.size). The observed-frame magnitude of the sun is the absolute magnitude of the sun after being redshifted to the given redshift. As such the observed-frame solar magnitude of the sun does not actually depend on `zf`. However, `zf` is currently included in the calling sequence for consistency with other similar functions, such as `ezgal.ezgal.get_absolute_mags()`. Returns an array of shape `(len(zs),len(filters))`. If there is only one filter, then it returns an array of shape `(len(zs))`

Call with `vega=True` or `ab=True` to specify ab or vega output. If neither is set it will output AB mags unless `ezgal.ezgal.set_vega_output()` has been called.

Note: Returns NaN for undefined solar magnitudes.

Note: The solar observed-frame magnitude does not depend on `zf`, but this is included for consistency with other similar functions.

Warning: Requires `zf > zs`

5.1.20 ezgal.get_solar_rest_mags()

`ezgal.`**`get_solar_rest_mags`**(*nzs=None*, *filters=None*, *ab=None*, *vega=None*)

Parameters

- **nzs** – The number of redshifts to return solar rest-frame magnitudes for
- **filters** (*string, list of strings*) – The filters to calculate solar rest-frame magnitudes for
- **zs** (*int, float, list, array*) – The redshifts to calculate M/L ratios at
- **ab** (*bool*) – Whether or not to output in AB magnitues
- **vega** (*bool*) – Whether or not to output in Vega magnitudes

Returns Array of solar rest-frame magnitudes

Return type array

Example

```
>>> import ezgal
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> model.add_filter( 'ch1' )
>>> model.add_filter( 'ch2' )
>>> model.add_filter( 'ch3' )
>>> model.add_filter( 'ch4' )
>>> model.get_solar_rest_mags( nzs=3 )
array([[ 6.06644384,  6.56615057,  7.0455159 ,  7.66857929],
       [ 6.06644384,  6.56615057,  7.0455159 ,  7.66857929],
       [ 6.06644384,  6.56615057,  7.0455159 ,  7.66857929]])
>>> model.get_solar_rest_mags( nzs=3, filters=['ch1','ch2'] )
array([[ 6.06644384,  6.56615057],
       [ 6.06644384,  6.56615057],
       [ 6.06644384,  6.56615057]])
>>> model.get_solar_rest_mags( nzs=1, filters='ch1' )
6.0664438415489164
```

Return the rest-frame absolute magnitude of the sun through the given filters. Specify the number of redshifts to return an array of size `(nzs,len(filters))`. The solar rest-frame magnitude does not depend on redshift, so values are repeated in the column for each filter in the returned array. This is done to match the return type of other methods such as `ezgal.ezgal.get_solar_observed_mags()`.

If nzs is None then returns an array of size `len(filters)`. If no filters are specified then it will calculate solar rest-frame magnitudes for all filters which have already been loaded into the model.

Call with `vega=True` or `ab=True` to specify ab or vega output. If neither is set it will output AB mags unless `ezgal.ezgal.set_vega_output()` has been called.

Note: Returns NaN for undefined solar magnitudes.

5.1.21 ezgal.get_zs()

`ezgal.`**`get_zs`**`(z)`

Parameters **z** (*int, float*) – The redshift out which to return redshifts

Returns An array of redshifts

Return type array

Example

```
>>> import ezgal
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> model.get_zs( 0.1 )
array([ 0.005,  0.01 ,  0.015,  0.02 ,  0.025,  0.03 ,  0.035,  0.04 ,
        0.045,  0.05 ,  0.055,  0.06 ,  0.065,  0.07 ,  0.075,  0.08 ,
        0.085,  0.09 ,  0.095])
```

Fetch a redshift grid out to redshift `z`. Returned redshifts stop just short of `z`. Returns more closely spaced redshifts at low redhift.

5.1.22 ezgal.make_burst()

`ezgal.`**`make_burst`**`(`*length*, *dust_function=None*, *dust_args=()*, *max_err=0.001*, *max_iter=200*`)`

Parameters

- **length** (*float*) – Length of burst (in gyrs)
- **dust_function** (*function or callable object*) – The dust dimming factor as a function of age and wavelength
- **dust_args** (*tuple*) – A tuple with additional arguments to pass to dust_function
- **max_err** (*float*) – Maximum allowed integration error (in magnitudes)
- **max_iter** (*int*) – Maximum number of interations allowed when trying to get errors below `max_err`

Returns `EzGal` model object for new CSP

Return type ezgal object

Example

```
>>> import ezgal
>>> import numpy as np
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> # dust free 0.1 gyr constant burst
>>> short_burst = model.make_burst( 0.1 )
```

Returns a new EzGal object with a constant SFR from `t=0` to `t=length`. Length should be in gyrs.

See Also:

See `ezgal.ezgal.make_csp()` for description of other parameters.

Warning: A discontinuity is present for star formation histories which are a constant burst, as the star formation drops from some finite value to zero at the end of the burst. It is not yet clear if `ezgal.ezgal.make_csp()` is properly handling discontinuities (see the warning there for more info), so test results from this method carefully before using them.

5.1.23 ezgal.make_csp()

`ezgal.`**`make_csp`**(*sfh_function*, *args=()*, *dust_function=None*, *dust_args=()*, *break_points=None*, *meta_data={}*, *max_err=0.001*, *max_iter=200*)

Parameters

- **sfh_function** (*function or callable object*) – The star formation rate as a function of age
- **args** (*tuple*) – A tuple with additional arguments to pass to sfh_function
- **dust_function** (*function or callable object*) – The dust dimming factor as a function of age and wavelength
- **dust_args** (*tuple*) – A tuple with additional arguments to pass to dust_function
- **break_points** (*list, array*) – A list of discontinutites in the star formation history
- **meta_data** (*dict*) – A dictionary with meta information to be stored in the new model
- **max_err** (*float*) – Maximum allowed integration error (in magnitudes)
- **max_iter** (*int*) – Maximum number of interations allowed when trying to get errors below `max_err`

Returns `EzGal` model object for new CSP

Return type ezgal object

Example

```
>>> import ezgal
>>> import numpy as np
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> def exponential( t, tau ):
>>>     return np.exp( -1.0*t/tau )
>>> # make a csp with a dust free exponentially decaying star formation history with tau
>>> exp_1gyr = model.make_csp( exponential, (1.0,) )
```

Returns a new EzGal object which is a composite stellar population generated from an arbitrary SFH. Pass a python function or callable object representing the star formation history (`sfh_function`). The star formation history function will be passed an age (in gyrs) and should return a corresponding weight (relative star formation

rate). `args` is an optional tuple and is passed to sfh_function. The calling sequence for the star formation history function will be:

```
weight = sfh_function( time, *args )
```

dust is an optional parameter and can be any python function or callable object. It should represent the dust extinction and should accept a time (in gyrs) and an array of wavelengths (in Angstroms). It should return the dimming factor at all wavelengths in an array of size `lambdas.size`. If a tuple is passed to dust_args then these will be passed to dust_function as extra arguments. The calling sequence for dust_function is:

```
factor = dust_function( time, lambdas, *args )
```

If the SFH history or dust function has discontinuities in time then the integral can be split up. Use `break_points` to pass a list of ages (in Gyrs) to split the integral at.

For integration, the SEDs in the model will be interpolated onto a more finely spaced grid in age. Before integrating an iterative process is used at 3000, 8000, and 12000 angstroms to determine the proper level of age resampling. The full integral is completed for these wavelength with increasing age resampling and the process stops when the difference in magnitude at all wavelengths from one iteration to the next drops below `max_err` (in magnitudes) or until the iteration has run `max_iter` times.

In general finer age sampling (and therefore longer calculation times) are required for star formation histories with sharp bursts (i.e. short bursts of star formation).

See Also:

See `ezgal.ezgal.set_meta_data()` for information about meta data.

Note: `break_points` is intended to decrease the amount of age resampling required for models with discontinuities, making for shorter execution times. However, it is still under development and its utility is not yet established.

Warning: Discontinuties are always a source of trouble, and at the moment it is not clear whether or not `make_csp` is properly handling them. In fact, it seems not to be. `break_points` is my current attempt at dealing with this problem, but is still under development. If you do model a star formation history with discontinuities then expect `make_csp` to hit the maximum iteration limit, and expect your model to take a very long amount of time to calculate. Moreover, it still might not be correct. Test your results carefully.

5.1.24 ezgal.make_delayed()

`ezgal.`**`make_delayed`**(*delay*, *age_units='gyrs'*)

Parameters

- **delay** (*int, float*) – length of delay.
- **age_units** (*string*) – units of delay.

Returns `EzGal` model with delayed star formation history

Return type `EzGal` model object.

Example

```
>>> import ezgal
>>> import numpy as np
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> delayed_ssp = model.make_delayed( 1.0 )
```

Return a new model with a delay before the onset of star formation. Expects `delay` to be in units of `age_units`. See `ezgal.utils.to_years()` for available age units.

This is intended for generating CSPs with complicated star formation histories, as it allows you to add together bursts of star formation which begin at different times.

Note: Uses interpolation to add in delay.

5.1.25 ezgal.make_exponential()

`ezgal.`**`make_exponential`**(*tau*, *dust_function=None*, *dust_args=()*, *max_err=0.001*, *max_iter=200*)

Parameters

- **tau** (*float*) – Time scale for exponentially decaying burst of star formation (in gyrs).
- **dust_function** (*function or callable object*) – The dust dimming factor as a function of age and wavelength
- **dust_args** (*tuple*) – A tuple with additional arguments to pass to dust_function
- **max_err** (*float*) – Maximum allowed integration error (in magnitudes)
- **max_iter** (*int*) – Maximum number of interations allowed when trying to get errors below `max_err`

Returns `EzGal` model object for new CSP

Return type ezgal object

Example

```
>>> import ezgal
>>> import numpy as np
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> # dust free exponentially decaying burst with tau=1 gyr
>>> exp_1 = model.make_exponential( 1.0 )
```

Returns a new EzGal object which exponentially decaying SFH with a timescale of `tau`.

See Also:

See `ezgal.ezgal.make_csp()` for description of other parameters.

5.1.26 ezgal.make_numeric()

`ezgal.`**`make_numeric`**(*ages*, *sfr*, *age_units='gyrs'*, *dust_function=None*, *dust_args=()*, *break_points=None*, *max_err=0.001*, *max_iter=200*)

Parameters

- **ages** (*list, array*) – A list of ages for the numeric star formation history.
- **sfr** (*list, array*) – The star formation rate at each age in `ages`.
- **age_units** (*string*) – The age units for the `ages` array.
- **dust_function** (*function or callable object*) – The dust dimming factor as a function of age and wavelength
- **dust_args** (*tuple*) – A tuple with additional arguments to pass to dust_function

- **max_err** (*float*) – Maximum allowed integration error (in magnitudes)
- **max_iter** (*int*) – Maximum number of interations allowed when trying to get errors below `max_err`

Returns `EzGal` model object for new CSP

Return type ezgal object

Example

```
>>> import ezgal
>>> import numpy as np
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> # some arbitrary SFH
>>> ages = [0,1,2,5,10,12]
>>> sfr  = [0,2,5,8,20,0]
>>> csp = model.make_numeric( ages, sfr, age_units='gyrs' )
```

Returns a new EzGal object with a SFH determined from an arbitrary numeric star formation history. `ages` should be an array of ages and `sfr` should be an array with the corresponding star formation rates as a function of time. Expects the ages to be in units of gyrs, if not specify units with `age_units` (see `ezgal.utils.to_years()` for full list of available age units). The star formation history does not have to be normalized.

See Also:

See `ezgal.ezgal.make_csp()` for description of other parameters.

5.1.27 ezgal.save_model()

`ezgal.`**`save_model`**(*output_file*, *filter_info=True*, *filter_only=False*)

Parameters

- **model_file** (*str*) – Output filename
- **filter_info** (*bool*) – Whether or not to output calculated model evolution
- **filter_only** (*bool*) – If true, only output calculated model evolution.

Returns None

Example

```
>>> import ezgal
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> model.add_filter( 'ch1' )
>>> model.add_filter( 'ch2' )
>>> model.set_zfs( [3,4,5] )
>>> model.save_model( 'bc03_ssp_z_0.02_chab_with_models.model' )
>>> new_model = ezgal.model( 'bc03_ssp_z_0.02_chab_with_models.model' )
>>> new_model.get_absolute_mags( 3.0, zs=[0,1,2] )
array([[ 6.39000031,  6.83946654],
       [ 5.61352032,  6.07145866],
       [ 4.48814389,  4.87218428]])
```

Saves the `EzGal` model to a multi-extensions fits file which can be read back in later.

If `filter_info=True` then this fits file will also contain the calculated model evolution as a function of filter and formation redshift. This will be loaded back into the object later, so that you don't have to recalculate the models everytime.

if `filter_only=True` then the SEDs will not be stored - just the calculated filter data. This can later be loaded and return observables as a function of z for any formation redshifts and filters already computed. These files are smaller because the model grid is not included, but `EzGal` will not be able to calculate observables for any new formation redshifts or filters.

5.1.28 ezgal.set_ab_output()

`ezgal.`**`set_ab_output`**`()`

Example

```
>>> import ezgal
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> # AB mag for ch1, zf=3, z=1
>>> model.get_absolute_mags( 3.0, filters='ch1', zs=1.0 )
5.6135203220610741
>>> # set Vega output
>>> model.set_vega_output()
>>> # Vega mag for ch1, zf=3, z=1
>>> model.get_absolute_mags( 3.0, filters='ch1', zs=1.0 )
2.8262012027987748
>>> # and back to AB mags
>>> model.set_ab_output()
>>> model.get_absolute_mags( 3.0, filters='ch1', zs=1.0 )
5.6135203220610741
```

Set the default output for the ezgal object to AB mags for all methods that can return magnitudes in vega or AB units.

Note: By default all `EzGal` methods output AB magnitudes.

5.1.29 ezgal.set_cosmology()

`ezgal.`**`set_cosmology`**(*Om=0.272*, *Ol=0.728*, *h=0.704*, *w=-1*)

Set the cosmology. The default cosmology is from WMAP 7, Komatsu, et al. 2011, ApJS, 192, 18. If the cosmology changes, then all filters will be regrided and any stored evolution models will be discarded.

5.1.30 ezgal.set_masses()

`ezgal.`**`set_masses`**(*ages*, *masses*, *age_units='gyrs'*, *grid=True*)

Parameters

- **ages** (*list, array*) – A list of ages
- **masses** (*list, array*) – A list of corresponding masses
- **age_units** (*str*) – The units that `ages` are in

Set the age-mass relationship for the model. Pass a list of ages, the corresponding masses, and the units the ages are in. Once the age-mass relationship is set, masses and mass-to-light ratios can be fetched using all the standard methods. Also, age-mass relationships will be stored for any CSPs generated from the `EzGal` object.

Note: Masses are stored in the ezgal object and are gridded for every filter and formation redshift. This is technically unnecessary since mass is independent of filter. However, this guarantees that interpolation is done for masses exactly as for luminosity, thus removing potential errors from the mass-to-light ratios.

5.1.31 ezgal.set_normalization()

`ezgal.`**`set_normalization`**(*filter*, *z*, *mag*, *vega=False*, *apparent=False*)

Parameters

- **filter** (*str*) – The normalization filter
- **z** (*int, float*) – The normalization redshift
- **mag** (*float*) – The normalization magnitude
- **vega** (*bool*) – Whether or not the normalization is in Vega mags
- **apparent** (*bool*) – Whether or not the normalization is in apparent magnitudes

Returns None

Example

```
>>> import ezgal
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> model.get_absolute_mags( 3.0, filters=['ch1','ch2'], zs=0.24 )
array([[ 6.18394468,  6.63764649]])
>>> # normalize to match a Dai et al. 2009 M* galaxy.
>>> model.set_normalization( 'ch1', 0.24, -25.06, vega=True )
>>> # also set Vega output, to match normalization
>>> model.set_vega_output()
>>> model.get_absolute_mags( 3.0, filters=['ch1','ch2'], zs=0.24 )
array([[-25.06      , -25.07924584]])
```

Set the model normalization, given the rest-frame magnitude of a galaxy in a given filter at a given redshift. Assumes AB magnitudes unless vega=True. Assumes rest-frame absolute mag unless apparent=True.

Note: Once set, the normalization can be unset by calling again with mag=0.

5.1.32 ezgal.set_vega_output()

`ezgal.`**`set_vega_output`**()

Example

```
>>> import ezgal
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> # AB mag for ch1, zf=3, z=1
>>> model.get_absolute_mags( 3.0, filters='ch1', zs=1.0 )
5.6135203220610741
```

```
>>> # set Vega output
>>> model.set_vega_output()
>>> # Vega mag for ch1, zf=3, z=1
>>> model.get_absolute_mags( 3.0, filters='ch1', zs=1.0 )
2.8262012027987748
```

Set the default output for the ezgal object to Vega mags for all methods that can return magnitudes in vega or AB units.

Note: By default all `EzGal` methods output AB magnitudes.

5.1.33 ezgal.set_meta_data()

`ezgal.`**`set_meta_data`**(*data*)

Parameters **data** (*dict*) – A dictionary containing model information

Returns None

Example

```
>>> import ezgal
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> model.set_meta_data( {'model': 'BC03', 'sfh': 'SSP'} )
>>> model.meta_data
{'model': 'BC03', 'sfh': 'SSP'}
```

Pass a dictionary with key/value pairs containing model information. This information is stored along with the model data, and is restored when the model is loaded later. Mainly, this serves as a method for propogating model information throughout python sessions. It can also be used extensively with the `Wrapper` class for searching, sorting, and interpolating between models.

Warning: Meta data will be stored in the fits header so keep key length <= 8 characters. Keep value length <20 characters.

5.1.34 ezgal.set_zfs()

`ezgal.`**`set_zfs`**(*zfs*, *grid=True*)

Parameters **zfs** (*list, array*) – A list of formation redshifts

Returns None

Set a list of formation redshifts for the model. When filters are later added to the model, magnitude evolution will automatically be calculated for those filters at the set formation redshifts. Also, magnitude evolution will be calculated for any already added filters at the given formation redshifts unless `grid=False`.

5.1.35 ezgal.weight()

`ezgal.`**`weight`**(*weight*)

Parameters **weight** (*int, float*) – Weight to apply to model

Returns Weighted `EzGal` object

Return type EzGal model object

Example

```
>>> import ezgal
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> heavy = model.weight( 2 )
>>> heavier = heavy.weight( 4 )
>>> model.model_weight, heavy.model_weight, heavier.model_weight
(1, 2, 8)
```

Return a copy of the EzGal model weighted by an additional factor of weight.

5.2 Astro Filter API

5.2.1 Astro Filter Objects

The astro_filter class is the workhorse that EzGal uses for calculating observables. astro_filter objects calculate magnitudes, store calculated information, use interpolation to fetch information quickly, and also calculate a number of filter properties. EzGal creates one astro_filter object for every filter added to a model. However, most of the astro_filter methods are intended only for internal usage, or are best accessed through calls to EzGal methods. Therefore, most of the astro_filter functionality will be of little interest to the end user. However, some of the parameters have potentially useful information (such as filter properties), and there might be occasional use for the ezgal.astro_filter.calc_mag() method (which calculates magnitudes from an SED), so these items are documented below.

astro_filter objects are stored in the filters parameter of every EzGal object. They are stored in a dictionary with the key being the name of the filter. See ezgal.ezgal.add_filter() for details on the name assigned to filters.

5.2.2 Astro Filter Parameters

Parameter	Value
ls	Wavelengths of filter transmission curve.
tran_ls	Transmission of Filter at each point in ls.
vs	Frequencies of filter transmission curve.
tran	Transmission of filter at each point in vs.
npts	Number of points in filter transmission curve.
ab_flux	Flux of an AB source through filter.
vega_flux	Flux of Vega through filter.
to_vega	AB to Vega conversion for filter (Vega-AB).
solar	Rest-frame AB magnitude of sun through filter.
mean	Mean Wavelength of the filter, in Angstroms.
pivot	Pivot Wavelength of the filter, in Angstroms.
average	Average Wavelength of the filter, in Angstroms.
sig	Effective Dimensionless Gaussian Width of the filter.
width	Effective Width of the filter, in Angstroms.
equivalent_width	Equivalent width of the filter, in Angstroms.
rectangular_width	Rectangular width of the filter, in Angstroms.

For example:

```
>>> import ezgal
>>> model = ezgal.ezgal( 'bc03_ssp_z_0.02_chab.model' )
>>> model.add_filter( 'ch1' )
>>> model.add_filter( 'ch2' )
>>> model.filters['ch1'].pivot, model.filters['ch2'].pivot
(35569.639436017096, 45020.234074579392)
>>> model.filters['ch1'].to_vega, model.filters['ch2'].to_vega
(-2.7873191192622992, -3.2602667751643533)
>>> model.filters['ch1'].ls.size, model.filters['ch1'].npts
(505, 505)
```

5.2.3 astro_filter.calc_mag()

`astro_filter.`**`calc_mag`**(*vs*, *sed*, *z*)

Parameters

- **vs** (*list, array*) – List of sed frequencies.
- **sed** (*list, array*) – The SED, with units of ergs/s/cm^2/Hz
- **z** (*int, float*) – The redshift to redshift the SED to.

Returns Absolute AB magnitude

Return type float

Example

```
>>> import ezgal
>>> model = ezgal.model( 'bc03_ssp_z_0.02_chab.model' )
>>> model.add_filter( 'ch1' )
>>> zf = 3
>>> z = 1
>>> # Use ``EzGal`` to calculate the rest-frame ch1 mag given zf & z
>>> model.get_absolute_mags( zf, filters='ch1', zs=z )
5.6135203220610741
>>> # now calculate it directly
>>> sed = model.get_sed_z( zf, z )
>>> model.filters['ch1'].calc_mag( model.vs, sed, 0 )
5.6135203220610741
>>> # same for apparent mag
>>> model.get_apparent_mags( zf, filters='ch1', zs=z )
47.969095009830326
>>> model.filters['ch1'].calc_mag( model.vs, sed, z ) + model.get_distance_moduli( z, nf
47.969095009830326
```

Calculate the absolute AB magnitude of the given sed at the given redshift. Set `z=0` for rest-frame magnitudes. `vs` should give the frequency (in Hz) of every point in the SED, and the sed should have units of ergs/s/cm^2/Hz.

5.3 Wrapper API

5.3.1 wrapper.add_model()

`wrapper.`**`add_model`**(*model*)

Parameters **model** (*filename, EzGal object*) – A filename/object to add to the wrapper object

Add an ezgal object to the wrapper object. `model` can be an ezgal model object or the filename of an ezgal compatible model file.

5.3.2 wrapper.argsort()

`wrapper.`**`argsort`**(*key*)

Parameters **key** (*string*) – The name of a meta keyword by which to sort

Returns List of indexes for soring wrapper object

Return type list

Example

```
>>> import ezgal
>>> wrapper = ezgal.wrapper( ['bc03_ssp_z_0.02_chab.model','bc03_ssp_z_0.008_chab.model'
>>> print wrapper.argsort( 'met' )
[1 0]
```

Return a numpy array of indexes to sort the models in the wrapper object. Sorting is done numerically by values in meta data keyword `key`. Data in given meta key must contain only numeric data.

5.3.3 wrapper.get_absolute_mags()

`wrapper.`**`get_absolute_mags`**(*zf*, *filters=None*, *zs=None*, *normalize=True*, *ab=None*, *vega=None*)

Parameters

- **zf** (*int, float*) – The formation redshift
- **filters** (*string, list*) – List of filters
- **zs** (*list, array, int, float*) – List of zs
- **normalize** (*bool*) – Normalize according to model normalization, if set
- **ab** (*bool*) – Whether or not to return AB mags
- **vega** (*bool*) – Whether or not to return vega mags

Returns Data cube with absolute magnitudes

Return type array

Example

```
>>> import ezgal
>>> wrapper = ezgal.wrapper( ['bc03_ssp_z_0.02_chab.model','bc03_ssp_z_0.008_chab.model'
>>> mags = wrapper.get_absolute_mags( 3.0, filters=['ch1','ch2','ch3'], zs=[0,1,2,2.5] )
>>> print mags.shape
( 2, 4, 3 )
>>> print mags
[[[ 6.39000031  6.83946654  7.25469915]
  [ 5.61352032  6.07145866  6.49122232]
  [ 4.48814389  4.87218428  5.30970258]
  [ 4.49875516  4.90041299  5.34425979]]
 [[ 6.43424729  6.92860144  7.35495047]
  [ 5.75053844  6.21216407  6.64700271]
```

```
[ 4.73228287  5.13875043  5.59090961]
[ 4.55125868  4.95278808  5.40624718]]]
```

Calls `ezgal.ezgal.get_absolute_mags()` on all stored model objects and returns results in a data cube of shape `(len( wrapper ),len( zs ),len( nfilters ))`

See Also:

`ezgal.ezgal.get_absolute_mags()`

Warning: If not already done, this will calculate the redshift evolution for all the specified filters at the given formation redshifts. If many `EzGal` objects have been loaded into the wrapper object, then this can take a lot of time. If working with many models it is often best to pre-calculate the redshift evolution for the filters and formation redshifts of interest, and save them in the `EzGal` model files.

5.3.4 wrapper.get_age()

`wrapper.`**`get_age`**(*z1*, *z2*, *units='gyrs'*)

Parameters

- **z1** (*int, float*) – The first redshift
- **z2** (*int, float, list, array*) – The second redshift
- **units** (*str*) – The units to return the time in

Returns Time between two redshifts

Return type int, float, list, array

Shortcut for wrapper[0].get_age(z1, z2, units=units)

See Also:

`ezgal.ezgal.get_age()`

5.3.5 wrapper.get_apparent_mags()

`wrapper.`**`get_apparent_mags`**(*zf*, *filters=None*, *zs=None*, *normalize=True*, *ab=None*, *vega=None*)

Same as `ezgal.wrapper.get_absolute_mags()` but calls `ezgal.ezgal.get_apparent_mags()`.

See Also:

`ezgal.wrapper.get_absolute_mags()`, `ezgal.ezgal.get_apparent_mags()`

5.3.6 wrapper.get_distance_moduli()

`wrapper.`**`get_distance_moduli`**(*zs=None*, *nfilters=None*)

Same as `ezgal.wrapper.get_absolute_mags()` but calls `ezgal.ezgal.get_distance_moduli()`.

See Also:

`ezgal.wrapper.get_absolute_mags()`, `ezgal.ezgal.get_distance_moduli()`

5.3.7 wrapper.get_ecorrects()

wrapper.**get_ecorrects**(*zf*, *filters=None*, *zs=None*)

Same as ezgal.wrapper.get_absolute_mags() but calls ezgal.ezgal.get_ecorrects().

See Also:

ezgal.wrapper.get_absolute_mags(), ezgal.ezgal.get_ecorrects()

5.3.8 wrapper.get_ekcorrects()

wrapper.**get_ekcorrects**(*zf*, *filters=None*, *zs=None*)

Same as ezgal.wrapper.get_absolute_mags() but calls ezgal.ezgal.get_ekcorrects().

See Also:

ezgal.wrapper.get_absolute_mags(), ezgal.ezgal.get_ekcorrects()

5.3.9 wrapper.get_kcorrects()

wrapper.**get_kcorrects**(*zf*, *filters=None*, *zs=None*)

Same as ezgal.wrapper.get_absolute_mags() but calls ezgal.ezgal.get_kcorrects().

See Also:

ezgal.wrapper.get_absolute_mags(), ezgal.ezgal.get_kcorrects()

5.3.10 wrapper.get_masses()

wrapper.**get_masses**(*zf*, *zs=zs*, *nfilters=None*)

Same as ezgal.wrapper.get_absolute_mags() but calls ezgal.ezgal.get_masses().

See Also:

ezgal.wrapper.get_absolute_mags(), ezgal.ezgal.get_masses()

5.3.11 wrapper.get_normalization()

wrapper.**get_normalization**(*zf*, *flux=False*)

Parameters

- **zf** (*int, float*) – The formation redshift to assume
- **flux** (*bool*) – Wheter or not to return a multiplicative factor

Returns The normalizations

Return type array

Example

```
>>> import ezgal
>>> wrapper = ezgal.wrapper( ['bc03_ssp_z_0.02_chab.model','bc03_ssp_z_0.008_chab.model'
>>> wrapper.set_normalization( 'ch1', 0.24, -25.06, vega=True )
>>> wrapper.get_normalization( 3.0 )
array([-28.45662556, -28.5216577 ])
```

```
>>> wrapper.get_normalization( 3.0, flux=True )
array([  2.41351622e+11,   2.56249531e+11])
```

Returns an array of size `len( wrapper )` containing the normalization for each model in the wrapper object.

See Also:

`ezgal.ezgal.get_normalization()`, `ezgal.wrapper.set_normalization()`

5.3.12 wrapper.get_observed_absolute_mags()

`wrapper.`**`get_observed_absolute_mags`**(*zf*, *filters=None*, *zs=None*, *normalize=True*, *ab=None*, *vega=None*)

Same as `ezgal.wrapper.get_absolute_mags()` but calls `ezgal.ezgal.get_observed_absolute_mags()`.

See Also:

`ezgal.wrapper.get_absolute_mags()`, `ezgal.ezgal.get_observed_absolute_mags()`

5.3.13 wrapper.get_observed_ml_ratios()

`wrapper.`**`get_observed_ml_ratios`**(*zf*, *filters=None*, *zs=None*)

Same as `ezgal.wrapper.get_absolute_mags()` but calls `ezgal.ezgal.get_observed_ml_ratios()`.

See Also:

`ezgal.wrapper.get_absolute_mags()`, `ezgal.ezgal.get_observed_ml_ratios()`

5.3.14 wrapper.get_rest_ml_ratios()

`wrapper.`**`get_rest_ml_ratios`**(*zf*, *filters=None*, *zs=None*)

Same as `ezgal.wrapper.get_absolute_mags()` but calls `ezgal.ezgal.get_rest_ml_ratios()`.

See Also:

`ezgal.wrapper.get_absolute_mags()`, `ezgal.ezgal.get_rest_ml_ratios()`

5.3.15 wrapper.get_solar_observed_mags()

`wrapper.`**`get_solar_observed_mags`**(*zf*, *filters=None*, *zs=None*, *ab=None*, *vega=None*)

Same as `ezgal.wrapper.get_absolute_mags()` but calls `ezgal.ezgal.get_solar_observed_mags()`.

See Also:

`ezgal.wrapper.get_absolute_mags()`, `ezgal.ezgal.get_solar_observed_mags()`

5.3.16 wrapper.set_normalization()

`wrapper.`**`set_normalization`**(*filter*, *z*, *mag*, *vega=False*, *apparent=False*)

Parameters

- **filter** (*str*) – The normalization filter
- **z** (*int, float*) – The normalization redshift
- **mag** (*float*) – The normalization magnitude

- **vega** (*bool*) – Whether or not the normalization is in Vega mags
- **apparent** (*bool*) – Whether or not the normalization is in apparent magnitudes

Returns None

Calls `ezgal.ezgal.set_normalization()` on all models loaded in the wrapper.

See Also:

`ezgal.ezgal.set_normalization()`, `ezgal.wrapper.get_normalization()`

5.4 Utils API

5.4.1 Utils Parameters

The utils module has three parameters of interest

Parameter	Value
au_per_pc	The number of AUs per parsec
c	The speed of light in m/s
m_per_au	The number of meters per astronomical unit

Which can be accessed from the ezgal module::

```
>>> import ezgal
>>> print ezgal.utils.c
299792458
```

5.4.2 ezgal.utils.convert_length()

`ezgal.utils.`**`convert_length`**(*to_convert*, *incoming='m'*, *outgoing='a'*)

converts a length from the incoming units to the outgoing units.

Parameters

- **to_convert** (*int, float*) – The length to convert
- **incoming** (*string*) – The units to convert the length from
- **outgoing** (*string*) – The units to convert the length to

Returns The converted length

Return type int, float

Example

```
>>> import ezgal
>>> ezgal.utils.convert_length( 1, incoming='pc', outgoing='au' )
206264.80624709636
```

See Also:

see `ezgal.utils.to_meters()` for available units.

5.4.3 ezgal.utils.convert_time()

ezgal.utils.**convert_time**(*to_convert*, *incoming='secs'*, *outgoing='gyrs'*)

Converts the given time from the incoming units to the outgoing units.

Parameters

- **to_convert** (*int, float*) – The length to convert
- **incoming** (*string*) – The units to convert the time from
- **outgoing** (*string*) – The units to convert the time to

Returns The converted time

Return type int, float

Example

```
>>> import ezgal
>>> ezgal.utils.convert_time( 1, incoming='years', outgoing='s' )
31536000.0
```

See Also:

see ezgal.utils.to_years() for available units.

5.4.4 ezgal.utils.to_hertz()

ezgal.utils.**to_hertz**(*to_convert*, *units='Angstroms'*)

Converts the given wavelength (in the given units) to hertz.

Parameters

- **to_convert** (*int, float*) – The wavelength to convert
- **units** (*string*) – The units the wavelength is in

Returns The converted frequency

Return type float

Example

```
>>> import ezgal
>>> ezgal.utils.to_hertz( 1000, units='a' )
2997924580000000.0
```

See Also:

see ezgal.utils.to_meters() for list of available units

Also see ezgal.utils.to_lambda()

5.4.5 ezgal.utils.to_lambda()

ezgal.utils.**to_lambda**(*to_convert*, *units='a'*)

Converts the given frequency to a wavelength in the given units.

Parameters

- **to_convert** (*int, float*) – The frequency to convert

- **units** (*string*) – The desired units of the output wavelength

Returns The converted wavelength

Return type float

Example

```
>>> import ezgal
>>> ezgal.utils.to_lambda( 2997924580000000.0, units='a' )
1000.0
```

See Also:

see `ezgal.utils.to_meters()` for list of available units

Also see `ezgal.utils.to_hertz()`

5.4.6 ezgal.utils.to_meters()

`ezgal.utils.`**`to_meters`**(*to_convert*, *units='a'*)

Converts a length from the given units to meters

Parameters

- **to_convert** (*int, float*) – The length to convert
- **units** (*string*) – The units to convert the length to

Returns The converted length

Return type int, float

Example

```
>>> import ezgal
>>> ezgal.utils.to_meters( 1e10, units='a' )
1.0
```

units

Available units are (case insensitive):

Name	Units
a,angstroms	Angstroms
nm,nanometers	Nanometers
um,microns	Microns
mm,millimeters	Millimeters
cm,centimeters	Centimeters
m,meters	Meters
km,kilometers	Kilometers
au	Astronomical Units
pc,parsecs	Parsecs
kpc, kiloparsecs	Kiloparsecs
mpc, megaparsecs	Megaparsecs

See Also:

`ezgal.utils.convert_length()`

5.4.7 ezgal.utils.to_years()

ezgal.utils.**to_years**(*to_convert*, *units='gyrs'*, *reverse=False*)

Converts the given time to years from the given units. If reverse=True then it converts from years into the given units

Parameters

- **to_convert** (*int, float*) – The time to convert
- **units** (*string*) – The units to convert the time to
- **reverse** (*bool*) – Converts from years if True

Returns The converted time

Return type int, float

Example

```
>>> import ezgal
>>> ezgal.utils.to_years( 1e-9, units='gyrs' )
1.0
```

units

Available units are (case insensitive):

Name	Units
gigayears,gyrs,gyr	Gigayears
megayears,myrs,myr	Megayears
years,yrs,yr	Years
days,day	Days
seconds,secs,sec,s	Seconds
log	log10(years)

See Also:

ezgal.utils.convert_time()

5.4.8 ezgal.utils.read_ised()

ezgal.utils.**read_ised**(*file*)

(seds, ages, vs) = ezgal.utils.read_ised(file)

Read a bruzual and charlot binary ised file.

Parameters **file** (*string*) – The name of the ised file

Returns A tuple containing model data

Return type tuple

Note: All returned variables are numpy arrays. ages and vs are one dimensional arrays, and seds has a shape of (vs.size,ages.size)

---

units Returns units of:

Return Variable	Units
seds	Ergs/s/cm**2/Hz
ages	Years
vs	Hz

5.4.9 ezgal.utils.rascii()

ezgal.utils.**rascii** (*filename*, *silent=False*)
Reads in numeric data stored in an ascii file into a numpy array.

Parameters

- **filename** (*string*) – The name of the ascii file
- **silent** (*bool*) – Whether or not to output basic file information

Returns A numpy array

Return type np.array()

Warning: Skips any lines that have any non-numeric data, and any data lines with a different number of columns than the first data line.

See Also:

ezgal.utils.wascii()

5.4.10 ezgal.utils.wascii()

ezgal.utils.**wascii** (*array*, *filename*, *formats*, *blank=False*, *header=False*, *names=None*)
Writes out a np array to a well formated file.

Parameters

- **array** (*a 2D numpy array*) – The numpy array to write out
- **filename** (*string*) – The name of the output file
- **formats** (*string,list*) – A list of python string formats (one for each column)
- **blank** (*string,list*) – Whether or not to output a blank line at the end of the file
- **header** (*string,list*) – A string or list of strings to write out as the header
- **names** – A list of column names with which to build a header

INDEX