rvspecfit

Sergey Koposov

Aug 29, 2023

CONTENTS:

class rvspecfit.make_interpol.**FakePool**

Bases: object

Methods

apply_async	
close	
join	

apply_async(*func*, *args*, *kwargs={}*)

close()

join()

class rvspecfit.make_interpol.**FakePoolResult**(*x*)

Bases: object

Methods

get	

get()

class rvspecfit.make_interpol.**Resolution**(*resol=None*, *resol_func=None*)

Bases: object

Methods

__call__(x)	Evaluate resolution at fixed wavelength

rvspecfit.make_interpol.**add_bool_arg**(*parser*, *name*, *default=False*, *help=None*)

rvspecfit.make_interpol.**extract_spectrum**(*logg*, *teff*, *feh*, *alpha*, *dbfile*, *prefix*, *wavefile*, *normalize=True*, *log_spec=True*)

Exctract a spectrum of a given parameters then apply the resolution smearing and divide by the continuum

Parameters

logg: real
Surface gravity

teff: real
Effective Temperature

feh: real
Metallicity

alpha: real
Alpha/Fe

dbfile: string
Path to the sqlite database

prefix: string
Prefix to the data files

wavefile: string
Path to the file with wavelengths

normalize: boolean
Normalize the spectrum by a linear continuum

rvspecfit.make_interpol.**get_line_continuum**(*lam*, *spec*)

Determine the extremely simple linear in log continuum to remove away continuum trends in templates

Parameters

lam: numpy array
Wavelength vector

spec: numpy array
spectrum

Returns

cont: numpy array
Continuum model

rvspecfit.make_interpol.**initialize_matrix_cache**(*mat*, *lamgrid*)

rvspecfit.make_interpol.**main**(*args*)

rvspecfit.make_interpol.**process_all**(*setupInfo*, *postf=''*, *dbfile='/tmp/files.db'*, *oprefix='psavs/'*, *prefix=None*, *wavefile=None*, *air=False*, *resolution0=None*, *normalize=True*, *revision=''*, *nthreads=8*)

class rvspecfit.make_interpol.**si**

Bases: object

Attributes

lamgrid
mat

lamgrid = None

mat = None

rvspecfit.make_nd.**execute**(*spec_setup*, *prefix=None*, *regular=False*, *perturb=True*, *revision=''*)

Prepare the triangulation objects for the set of spectral data for a given spec_setup.

Parameters

spec_setup: string
The spectroscopic configuration

prefix: string
The prefix where the data are located and where the triangulation will be stored

perturb: boolean
Boolean flag whether to perturb a little bit the points before doing a triangulation. This prevents issues with degenerate vertices and stability of triangulation. Without perturbation find_simplex for example may revert to brute force search.

Returns

None

rvspecfit.make_nd.**getedgevertices**(*vec*)

Given a set of n-dimentional points, return vertices of an n-dimensional cube that fully encompass/surrounds the data, This is sort of the envelope around the data

Parameters

vec: numpy (Ndim, Npts)
The array of input points

Returns

vec: numpy (Ndim, Nretpts)
The returned array of surrounding points

rvspecfit.make_nd.**main**(*args*)

rvspecfit.utils.**freezeDict**(*d*)

Take the input object and if it is a dictionary, freeze it (i.e. return frozendict) If not, do nothing

Parameters

d: dict
Input dictionary

Returns

d: frozendict
Frozen input dictionary

rvspecfit.utils.**get_default_config**()

Create a default parameter config ditctionary

Returns

ret: dict
Dictionary with config params

rvspecfit.utils.**read_config**(*fname=None*, *override_options=None*)

Read the configuration file and return the frozendict with it

Parameters

fname: string, optional
The path to the configuration file. If not given config.yaml in the current directory is used

override_options: dictionary, optional
Update the options

Returns

———-

config: frozendict
The dictionary with the configuration from a file

class rvspecfit.spec_inter.**GridInterp**(*uvecs*, *idgrid*, *vecs*, *dats*, *exp=True*)

Bases: object

Methods

`__call__`(p)	Compute the interpolated spectrum at parameter vector p

get_nearest

get_nearest(*p*)

class rvspecfit.spec_inter.**GridOutsideCheck**(*uvecs*, *vecs*, *idgrid*)

Bases: `object`

Methods

`__call__`(p)	Call self as a function.

class rvspecfit.spec_inter.**SpecInterpolator**(*name*, *interper*, *extraper*, *lam*, *mapper*, *parnames*, *revision=''*, *filename=''*, *creation_soft_version=''*, *logstep=None*)

Bases: `object`

Spectrum interpolator object

Methods

eval(param0)	Evaluate the spectrum at a given parameter
outsideFlag(param0)	Check if the point is outside the interpolation grid

eval(*param0*)

Evaluate the spectrum at a given parameter

outsideFlag(*param0*)

Check if the point is outside the interpolation grid

Parameters

param0: tuple
parameter vector

Returns
ret: float
> 0 if point outside the grid

class rvspecfit.spec_inter.**TriInterp**(*triang*, *dats*, *exp=True*)

Bases: `object`

Methods

`__call__`(p)	Compute the interpolated spectrum at parameter vector p

rvspecfit.spec_inter.**getInterpolator**(*HR*, *config*, *warmup_cache=False*)

Return the spectrum interpolation object for a given instrument setup HR and config. This function also checks the cache

Parameters

HR: string
Spectral configuration

config: dict
Configuration

warmup_cache: bool
If True we read the whole file to warm up the OS cache

Returns

ret: SpecInterpolator
The spectral interpolator

rvspecfit.spec_inter.**getSpecParams**(*setup*, *config*)

Return the ordered list of spectral parameters of a given spectroscopic setup

Parameters

setup: str
Spectral configuration

config: dict
Configuration dictionary

Returns

ret: list
List of parameters names

class rvspecfit.spec_inter.**interp_cache**

Bases: `object`

Singleton object caching the interpolators

interps = {}

class rvspecfit.read_grid.**ParamMapper**

Bases: `object`

Class used to map stellar atmospheric parameters into more suitable space used for interpolation

Methods

forward(vec)	Map atmospheric parameters into parameters used in the grid for interpolation.
inverse(vec)	Map transformed parameters used in the grid for interpolation back into the atmospheric parameters.

forward(*vec*)

Map atmospheric parameters into parameters used in the grid for interpolation. That includes logarithming the teff

Parameters

vec: numpy array
The vector of atmospheric parameters Teff, logg, feh, alpha

Returns

ret: numpy array
The vector of transformed parameters used in interpolation

inverse(*vec*)

Map transformed parameters used in the grid for interpolation back into the atmospheric parameters. That includes exponentiating the log10(teff)

Parameters

vec: numpy array
The vector of transformed atmospheric parameters log(Teff), logg, feh, alpha

Returns

ret: numpy array
The vector of original atmospheric parameters.

rvspecfit.read_grid.**apply_rebinner**(*mat*, *spec0*)

rvspecfit.read_grid.**gau_integrator**(*A*, *B*, *x1*, *x2*, *l1*, *l2*, *s*)

This computes the integral of (Ax+B)/sqrt(2pi)/s*exp(-1/2*(x-y)^2/s^2 for x=x1..x2 y=l1..l2

Here is the mathematica code FortranForm[Simplify[Integrate[(A*x + B)/Sqrt[2*Pi]/s* Exp[-1/2*(x - y)^2/s^2], {x, x1, x2}, {y, l1, l2}]]]

Parameters A: ndarray B: ndarray x1: ndarray x2: ndarray l1: ndarray l2: ndarray s: ndarray

Returns

ret: float
Integral of the function

rvspecfit.read_grid.**get_spec**(*logg*, *temp*, *met*, *alpha*, *dbfile='/tmp/files.db'*, *prefix='PHOENIX_PATH/v2.0/HiResFITS/PHOENIX-ACES-AGSS-COND-2011/'*, *wavefile='PHOENIX_PATH/v2.0/HiResFITS/WAVE_PHOENIX-ACES-AGSS-COND-2011.fits'*)

Returns individual spectra for a given spectral parameters

Parameters

logg: real
Surface gravity

temp: real
Temperature

met: real
[Fe/H]

alpha: real
[alpha/Fe]

dbfile: string
The pathname to the database sqlite file of templates

prefix: string
The prefix path to templates

wavefile: string
The filename of fits file with the wavelength vector

Returns

lam: ndarray
1-D array of wavelength

spec: ndarray
1-D array of spectrum

Example

```
>>> lam,spec=read_grid.get_spec(1,5250,-1,0.4)
    ..
```

rvspecfit.read_grid.**main**(*args*)

rvspecfit.read_grid.**make_rebinner**(*lam00*, *lam*, *resolution_function*, *resolution0=None*, *toair=True*)

Make the sparse matrix that convolves a given spectrum to a given resolution and new wavelength grid

Parameters

lam00: array
The input wavelength grid of the templates

lam: array
The desired wavelength grid of the output

resolution_function: function
The function that when called with the wavelength as an argument will return the resolution of the desired spectra (R=l/dl) I.e. it could be just lambda x: 5000

toair: bool
Convert the input spectra into the air frame

resolution0: float
The resolution of input templates

Returns

The sparse matrix to perform interpolation

rvspecfit.read_grid.**makedb**(*prefix='/PHOENIX-ACES-AGSS-COND-2011/'*, *dbfile='files.db'*, *keywords=None*, *mask=None*, *extra_params=None*)

Create an sqlite database of the templates

Parameters

prefix: str
The path to PHOENIX

dbfile: str
The output file with the sqlite DB

keywords: dict
The dictionary with the map of teff,logg,feh,alpha to keyword names in the headers

mask: string
The string how to identify spectral files, i.e. '*/*fits'

extra_params: dict or None
The dictionary of variable name vs FITS name to read from spectral files

rvspecfit.read_grid.**pix_integrator**(*x1*, *x2*, *l1*, *l2*, *s*)

Integrate the flux within the pixel given the LSF We assume that the flux before LSF convolution is given by linear interpolation from x1, x2. The l1,l2 are scalar edges of the pixel within which we want to compute the flux. s is the sigma of the LSF The function returns two values of weights for y1,y2 where y1,y2 are the values of the non-convolved spectra at x1,x2

Parameters

x1: ndarray
x2: ndarray
l1: ndarray
l2: ndarray
s: ndarray

rvspecfit.read_grid.**rebin**(*lam0*, *spec0*, *newlam*, *resolution*)

Rebin a given spectrum lam0, spec0 to the new wavelength and new resolution

Parameters

lam0: ndarray
1d numpy array with wavelengths of the template pixels

spec0: ndarray
1d numpy array with the template spectrum

newlam: ndarray
1d array with the wavelengths of the output spectrum

resolution: float
Resolution lam/dlam

Returns

spec: ndarray
Rebinned spectrum

CHAPTER

ONE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

INDEX

S

T