Nirdust

Release 0.2.0

Gaia Gaspar & Jose Alacoria

Feb 06, 2024

CONTENTS

NIRDust is a python package that uses K-band (2.2 micrometers) spectra to measure the temperature of the dust heated by an Active Galactic Nuclei (AGN) accretion disk.

CHAPTER

ONE

MOTIVATION

K-band nuclear spectral continuum of Type 2 AGNs is often composed of two components: the stellar population emission and a hot 800 - 1600 K dust component. Via substraction of the stellar emission dust component fitting can be performed to map its prescence and estimate its temperature.

CHAPTER

TWO

FEATURES

The package uses the modeling features of astropy to fit the hot dust component of a AGN K-band spectrum with black body functions. And provide a class with methods for spectrum manipulation and normalized-blackbody-fitting. Because NIRDust normalizes the spectra before fitting, is not necessary to flux-calibrate spectra to use it.

NIRDust needs a minimum of two spectra to run: a nuclear one, where the dust temperature will be determined, and an off-nuclear spectrum, where the emission is considered to be purely stellar. The off-nuclear spectrum will be used by NIRDust to subtract the stellar emission from the nuclear spectrum.

Footnote: the hot dust component may or may not be present in your type 2 nuclei, do not get disappointed if NIRDust finds nothing.

CHAPTER

THREE

USER DOCUMENTATION

3.1 Installation

This is the recommended way to install NIRDust.

3.1.1 Installing with pip

Make sure that you are using Python 3.8 or newer. The most convenient way to install NIRDust is within a virtual environment via the pip command.

After setting up and activating the virtualenv, run the following command:

```
$ pip install nirdust
```

Now NIRDust should be installed in your system along with all its dependencies.

3.1.2 Installing the development version

If you'd like to be able to update your NIRDust copy with the latest bug fixes and improvements, follow these instructions:

Make sure that you have Git installed and that you can run its commands from a shell. (Enter `git help` at a shell prompt to test this.)

Check out nirdust main development branch as follows:

```
$ git clone https://github.com/Gaiana/nirdust
```

This will create a directory *nirdust* in your current directory.

Then you can proceed to install with the commands

```
$ cd nirdust
$ pip install -e .
```

3.2 Tutorials

The following page contains a collection of tutorials and practical examples on how to use NIRDust. From reading your spectra from a file, to modelling the hot dust emission and visualizing the results.

3.2.1 Reading Spectra

NIRDust provides easy and quick functionalities to read your spectra. If you have your data in FITS format or just a common text ASCII file, you can use one of the following functions. If you don't have your data in any of these formats, by the end of this tutorial you can see how to load your data to create a NirdustSpectrum object directly.

Reading from a FITS file

Let's say you have your spectra in a FITS file. This format, in most cases, comes with a header and a science extension. The science extension stores the flux intesity of the spectrum and the respective pixel axis. The header has the information to transform the pixel axis into a spectral axis. The manual input of the redshift of the galaxy, which isn't included in most headers, is necesary to get the spectral axis in rest frame. The following code shows the easiest way to read a spectrum:

[2]:
```
import nirdust as nd

nuclear_spectrum = nd.read_fits("nuclear_spectrum.fits", z=0.00183)

# Show some information
nuclear_spectrum
```

[2]:
```
NirdustSpectrum(z=0.00183, spectral_length=1751, spectral_range=[18889.58-25106.71]␣
↪Angstrom)
```

If your FITS file has multiple extensions, the automatic detection of the science extension may fail. In such cases, you can manually indicate the correct extension by passing the parameter `extension=1` to the function (if 1 is the correct number).

Reading from an ASCII file

The function `read_table` can be used to read directly from a text file and supports multiple formats. In this case the file should contain these two columns: wavelength [A] and flux [arbitrary units]. If you have multiple columns you should indicate which columns correspond to the spectrum data. For a list of supported formats see this link.

[ ]:
```
spectrum = nd.read_table("file_name.csv", wavelength_column=2, flux_column=4, format="csv
↪")
```

[ ]:
```
```

3.2.2 Preprocessing utilities

work in progress...

3.2.3 Fitting a BlackBody model

work in progress...

3.2.4 Example of NIRDust application

The example imputs are two Flamingos-2 (Gemini South) K_{long} band longslit spectra of NGC 5128 (Centaurus A) a Seyfert 2 galaxy.

The spectra correspond to:

1) A nuclear spectrum extracted in an aperture of 1"x 1" (slit width = 1").
2) An off-nuclear spectrum extracted at 3.34" distance from the nucleus and in an aperture of 1.08" x 1".

These spectra can be found in the examples folder of the Git-Hub repository for Nirdust. No flux-calibration has been performed over these spectra.

For this simple aplication example we only need to import nirdust, and matplotlib to see the spectra at diferent steps:

[1]:
```
import nirdust as nd
import matplotlib.pyplot as plt
```

Read the spectra from the FITS files and store them in two separate NirdustSpectrum objects

Note that the redshift for this galaxy is 0.00183 (source: NED)

[2]:
```
target_spectrum = nd.read_fits("nuclear_spectrum.fits", z=0.00183)
reference_spectrum = nd.read_fits("external_spectrum.fits", z=0.00183)

target_spectrum
```

```
WARNING: FITSFixedWarning: RADECSYS= 'FK5 ' / R.A./DEC. coordinate system reference
the RADECSYS keyword is deprecated, use RADESYSa. [astropy.wcs.wcs]
```

[2]:
```
NirdustSpectrum(z=0.00183, spectral_length=1751, spectral_range=[18889.58-25106.71]␣
↪Angstrom)
```

The spectra before any pre-processing procedure look like this:

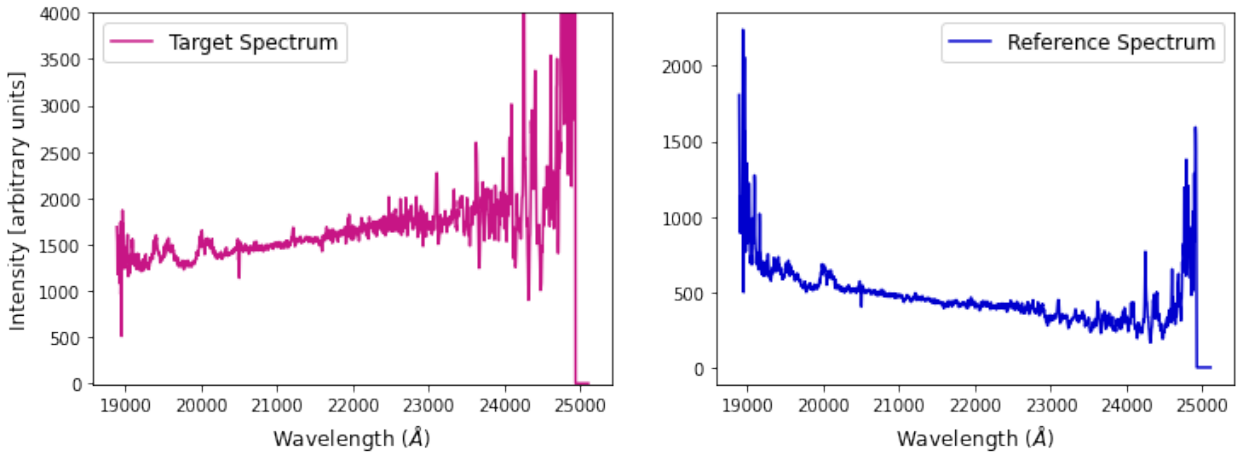
[3]:
```
fig, axs = plt.subplots(1, 2,figsize=(12, 4))
axs[0].plot(target_spectrum.spectral_axis, target_spectrum.flux, color="mediumvioletred",
↪ label="Target Spectrum")
axs[1].plot(reference_spectrum.spectral_axis, reference_spectrum.flux, color="mediumblue
↪", label="Reference Spectrum")
axs[0].legend(loc=2, fontsize=12)
axs[1].legend(loc=1, fontsize=12)
axs[0].set_xlabel("Wavelength ($\AA$)", fontsize=12 )
axs[1].set_xlabel("Wavelength ($\AA$)", fontsize=12)
axs[0].set_ylabel("Intensity [arbitrary units]", fontsize=12)
```

(continues on next page)

(continued from previous page)

```
axs[0].set_ylim(-10, 4000)  #setting ylim for better display, the noise rises higher␣
↪than the limit at the end of spectra
```

[3]:
```
(-10.0, 4000.0)
```



Cut the borders of the spectrum where the noise is too high

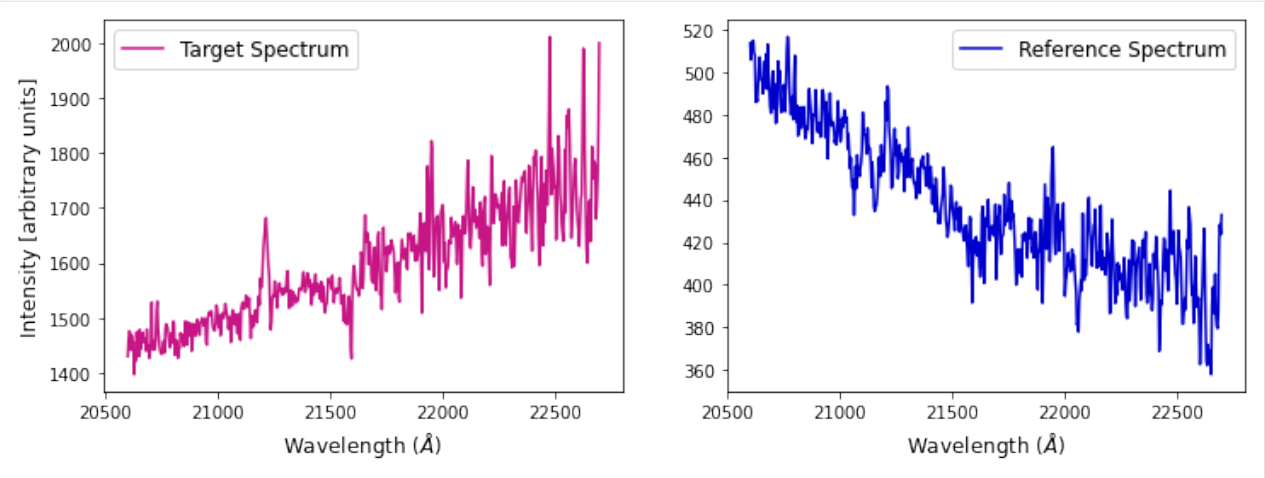
K-band spectra must be cut taking into account two factors: In one hand, spectrum quality often drops in the borders of the spectral range due to detector response and atmospheric transmission. Special attention must be payed to bad telluric absorption correction, which commonly affects the beginning of the K band between $\sim$ 1.95 and 2.1 μm. In the other hand, in K_{long} spectra, the beginning of the CO absorption band causes significant continuum absorption between 2.3 and 2.5 μm and hence the information of the dust presence is lost. Aditionally, in the case of this set of spectra, high levels of noise are present in the spectra beyond 2.4 μm.

[4]:
```
start, end = 20600, 22700  # wavelengths in A
cut_target = target_spectrum.cut_edges(start, end)
cut_reference = reference_spectrum.cut_edges(start, end)
```

Once the spectra are cut the obtained result is:

[5]:
```
fig, axs = plt.subplots(1, 2, figsize=(12, 4))
axs[0].plot(cut_target.spectral_axis, cut_target.flux, color="mediumvioletred", label=
↪"Target Spectrum")
axs[1].plot(cut_reference.spectral_axis, cut_reference.flux, color="mediumblue", label=
↪"Reference Spectrum")
axs[0].legend(loc=2, fontsize=12)
axs[1].legend(loc=1, fontsize=12)
axs[0].set_xlabel("Wavelength ($\AA$)", fontsize=12)
axs[1].set_xlabel("Wavelength ($\AA$)", fontsize=12)
axs[0].set_ylabel("Intensity [arbitrary units]", fontsize=12)
```

[5]:
```
Text(0, 0.5, 'Intensity [arbitrary units]')
```



Remove spectral features

The Reference Spectrum is quite featurless although is quite noisy, but the Target Spectrum presents some emission or noise features that could be auctomatically detected and removed. 'line_spectrum' will detect the features down to some certain 'noise_factor' and provide the mask and the spectrum of the detected lines:

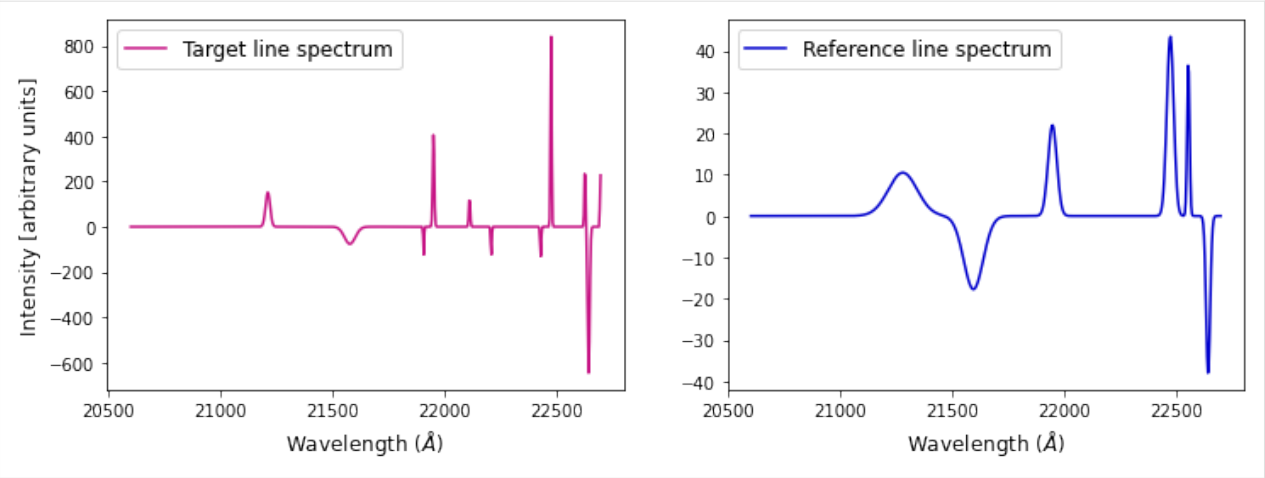
[6]:
```
target_lines, target_line_intervals = nd.line_spectrum(cut_target, noise_factor=5.5)
reference_lines, reference_line_intervals = nd.line_spectrum(cut_reference, noise_
↪factor=3.5)
```

```
WARNING: The fit may be unsuccessful; check fit_info['message'] for more information.␣
↪[astropy.modeling.fitting]
```

The spectrum of the detected features in each case is a NirdustSpectrum object and can be plotted for inspection:

[7]:
```
fig, axs = plt.subplots(1, 2, figsize=(12, 4))
axs[0].plot(target_lines.spectral_axis, target_lines.flux, color="mediumvioletred",␣
↪label="Target line spectrum")
axs[1].plot(reference_lines.spectral_axis, reference_lines.flux, color="mediumblue",␣
↪label="Reference line spectrum")
axs[0].legend(loc=2, fontsize=12)
axs[1].legend(loc=2, fontsize=12)
axs[0].set_xlabel("Wavelength ($\AA$)", fontsize=12 )
axs[1].set_xlabel("Wavelength ($\AA$)", fontsize=12)
axs[0].set_ylabel("Intensity [arbitrary units]", fontsize=12)
```

[7]:
```
Text(0, 0.5, 'Intensity [arbitrary units]')
```

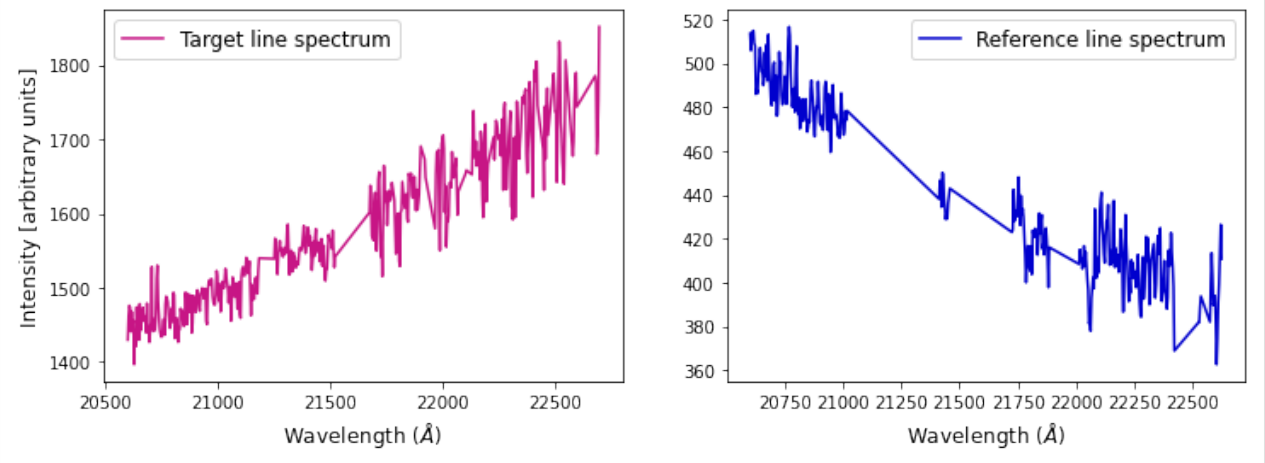


The mask for each spectrum is stored in the second output of line_spectrum and can be used as input for the class method "mask_spectrum":

[8]:
```
masked_target = cut_target.mask_spectrum(line_intervals=target_line_intervals)
masked_reference = cut_reference.mask_spectrum(line_intervals=reference_line_intervals)
```

[9]:
```
fig, axs = plt.subplots(1, 2, figsize=(12, 4))
axs[0].plot(masked_target.spectral_axis, masked_target.flux, color="mediumvioletred",␣
↪label="Target line spectrum")
axs[1].plot(masked_reference.spectral_axis, masked_reference.flux, color="mediumblue",␣
↪label="Reference line spectrum")
axs[0].legend(loc=2, fontsize=12)
axs[1].legend(loc=1, fontsize=12)
axs[0].set_xlabel("Wavelength ($\AA$)", fontsize=12)
axs[1].set_xlabel("Wavelength ($\AA$)", fontsize=12)
axs[0].set_ylabel("Intensity [arbitrary units]", fontsize=12)
```

[9]:
```
Text(0, 0.5, 'Intensity [arbitrary units]')
```



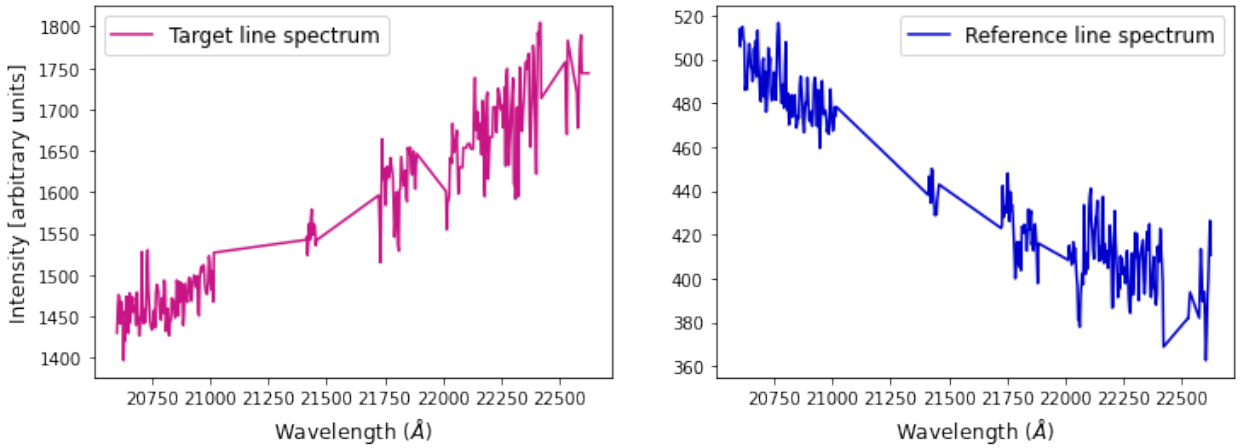
The last step in the pre-processing of the spectra is to match the spectral axes of both spectra to match the spectral resultution (not necessary in this example) and to homogenize the masked points:

[10]:
```
final_target, final_reference = nd.match_spectral_axes(masked_target, masked_reference)
```

The final spectra result:

[11]:
```
fig, axs = plt.subplots(1, 2, figsize=(12, 4))
axs[0].plot(final_target.spectral_axis, final_target.flux, color="mediumvioletred",␣
↪label="Target line spectrum")
axs[1].plot(final_reference.spectral_axis, final_reference.flux, color="mediumblue",␣
↪label="Reference line spectrum")
axs[0].legend(loc=2, fontsize=12)
axs[1].legend(loc=1, fontsize=12)
axs[0].set_xlabel("Wavelength ($\AA$)", fontsize=12)
axs[1].set_xlabel("Wavelength ($\AA$)", fontsize=12)
axs[0].set_ylabel("Intensity [arbitrary units]", fontsize=12)
```

[11]:
```
Text(0, 0.5, 'Intensity [arbitrary units]')
```



Compute the S/N ratio in order to asset the uncertainties of the result

After the spectral emission and noise features are removed, the noise can be computed in order to calculate the S/N ratio of each spectrum. Note thet the class method “compute_noise” will return a new NirdustSpectrum instance:

[12]:
```
low_lim, upper_lim = 20700, 22400  # wavelengths in A
noise_target = masked_target.compute_noise(low_lim, upper_lim)
noise_reference = masked_reference.compute_noise(low_lim, upper_lim)
noise_target.noise, noise_reference.noise
```

[12]:
```
(30.812629918132114, 10.996114393173345)
```

To compute the mean S/N of each spectra the value of the signal in the center of the band can be assumed as the mean signal. This step is not part of the pre-processing of the spectra but is helpful in order to asset the uncertainty of the fitting (see the publication of the package).

[13]:
```
central_index = masked_target.spectral_length // 2

target_signal = masked_target.flux[central_index]
reference_signal = masked_reference.flux[central_index]
```

(continues on next page)

(continued from previous page)

```
snr_target = target_signal/noise_target.noise
snr_reference = reference_signal/noise_reference.noise

snr_target.value, snr_reference.value
```

[13]:
```
(50.10917956601182, 37.422864914091654)
```

Obtain the temperature of the hot dust

Now the spectra are ready to perform the fitting and obtain the temperature. The 'bounds' parameter of 'fit_blackbody' can be determined in order to provide some physical meaningful constraints to the parameter fitting. Also, the 'gamma_target_fraction' parameter can be constrained to a low value if the spectra are background_substracted:

[14]:
```
# min and max bounds for T, alpha, beta, gamma
bounds = ((400, 2000), (0, 20), (6, 20), (-10, 10))

results = nd.fit_blackbody(final_target, final_reference, niter=700, gamma_target_
↪fraction=0.01,bounds=bounds)
```

```
/home/martin/Documents/Projects/nirdust/lib/python3.8/site-packages/scipy/optimize/_
↪optimize.py:284: RuntimeWarning: Values in x were outside bounds during a minimize␣
↪step, clipping to bounds
  warnings.warn("Values in x were outside bounds during a "
```

The resutl of the fitting is a NirdustResults object and can be displayed as a dictionary that contains many useful information as the values for the fitted parameters and the success status. The fitted values for the α, β, and γ parameters should not be in the borders of the intervals defined for the parameters in 'bounds', in that case, the fitting must be repeated with differents initial parameters.

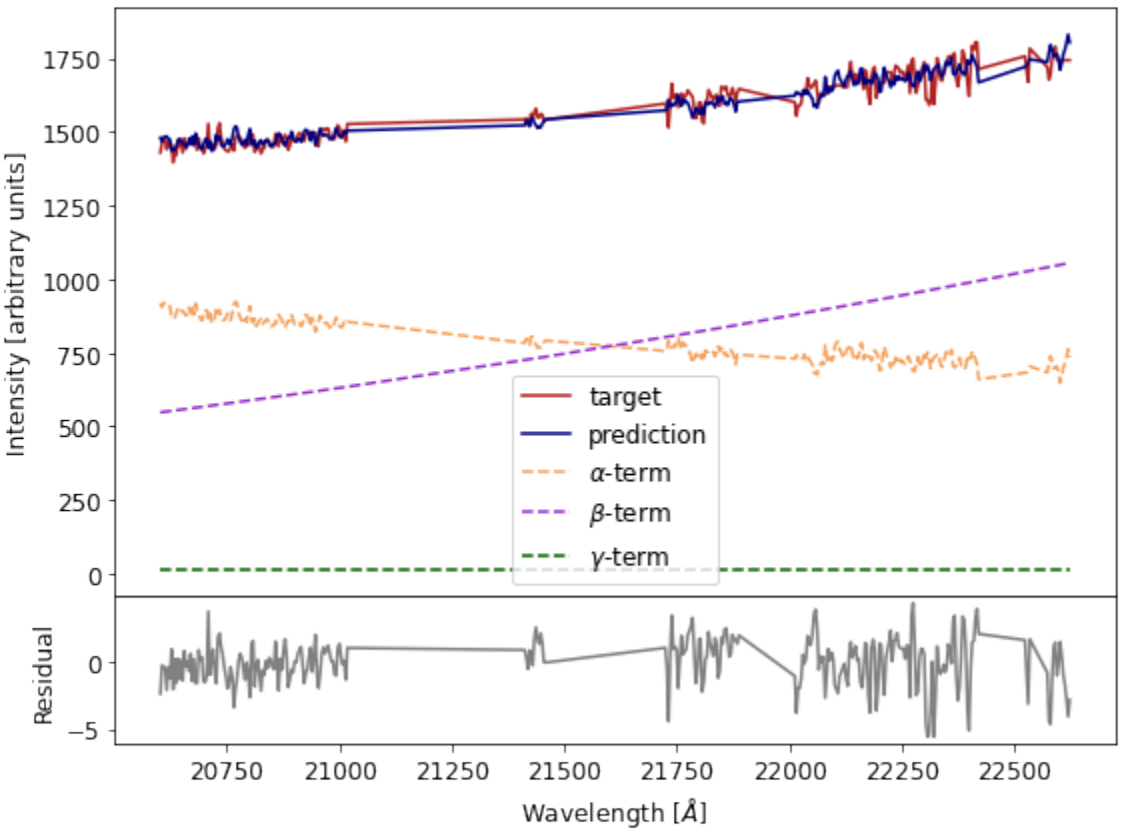
[15]:
```
results
```

[15]:
```
NirdustResults(temperature=666 K, alpha=1.79, beta=11.63, gamma=1.15,
minimizer_results=                        fun: 1728.147825028615
 lowest_optimization_result:      fun: 1728.147825028615
     jac: array([  -3.2599333 ,  124.4421125 , -503.72867213,   -5.27996959])
 message: 'Optimization terminated successfully'
    nfev: 773
     nit: 79
    njev: 78
  status: 0
 success: True
       x: array([666.49754838,   1.78847343,  11.6318601 ,   1.14510016])
                  message: ['requested number of basinhopping iterations completed␣
↪successfully']
     minimization_failures: 312
                    nfev: 274345
                     nit: 700
                    njev: 23641
                  success: True
                        x: array([666.49754838,   1.78847343,  11.6318601 ,   1.
↪14510016]))
```

Also, the NirudtsResults class has a class method to plot the data and the spectra obtained with the fitting:

[16]:
```
results.plot(show_components=True)
```

[16]:
```
(<AxesSubplot:ylabel='Intensity [arbitrary units]'>,
 <AxesSubplot:xlabel='Wavelength [$\\AA$]', ylabel='Residual'>)
```



3.3 API Reference

NIRDust: Near Infrared Dust Finder.

NIRDust is a python package that uses K-band (2.2 micrometers) spectra to measure the temperature of the dust heated by an Active Galactic Nuclei (AGN) accretion disk.

3.3.1 nirdust.core module

Core functionalities for NIRDust.

class nirdust.core.**NirdustSpectrum**(*spectral_axis*, *flux*, *z=0*, *metadata=_Nothing.NOTHING*, *noise=_Nothing.NOTHING*)

Bases: `object`

Class containing a spectrum to operate with Nirdust.

Stores the flux and wavelength axis of the spectrum in a Spectrum1D object and provides various methods for obtaining the dust component and perform blackbody fitting. The *flux* parameter recieves either flux-calibrated intensity or non flux-calibrated intensity, in both cases Nirdust will assign 'ADU' units to it.

Parameters

- **flux** (*~numpy.ndarray*, or *~astropy.units.Quantity*) – Spectral intensity.
- **frequency_axis** (*~numpy.ndarray*, or *~astropy.units.Quantity*) – Spectral axis in units of Hz.
- **z** (*float, optional*) – Redshift of the galaxy. Default is 0.
- **metadata** (*mapping, optional*) – Any dict like object. This is a good place to store the header of the fits file or any arbitrary mapping. Internally NirdustSpectrum wraps the object inside a convenient metadata object usefull to access the keys as attributes.

spec1d_

Contains the wavelength axis and the flux axis of the spectrum in unities of Å and ADU respectively.

Type
specutils.Spectrum1D object

noise

A value of the uncertainty representative of all the spectral range. If the value of noise is not provided, Nirdust will compute it by default using *noise_region_uncertainty* from *Astropy* inside the region 20650 - 21000 Å. The user can re-compute noise using the class method *compute_noise*.

Type
float.

compute_noise(*low_lim=20650*, *upper_lim=21000*)

Compute noise for the spectrum.

Uses *noise_region_uncertainty* from Astropy to compute the noise of the spectrum inside a *Spectral Region* given by the *low_lim* and *upper_lim* parameters.

Parameters

- **low_lim** (*float*) – A float containing the lower limit for the region where the noise will be computed. Must be in Å. Default is 20650.
- **upper_lim** (*float*) – A float containing the lower limit for the region where the noise will be computed. Must be in Å. Default is 21000.

Returns
A new instance of *NirdustSpectrum* class with the new noise parameter.

Return type
NirdustSpectrum object

convert_to_frequency()

Convert the spectral axis to frequency in units of Hz.

Returns
out – New instance of the *NirdustSpectrun* class containing the spectrum with a frquency axis in units of Hz.

Return type
object *NirsdustSpectrum*

cut_edges(*mini*, *maxi*)

Cut the spectrum in wavelength range.

Parameters

- **mini** (*float*) – Lower limit to cut the spectrum.

- **maxi** (*float*) – Upper limit to cut the spectrum.

Returns
out – Return a new instance of class *NirdustSpectrum* cut in wavelength.

Return type
NirsdustSpectrum object

property frequency_axis

Frequency axis access.

mask_spectrum(*line_intervals=None*, *mask=None*)

Mask spectrum to remove spectral lines.

Recives either a boolean mask containing *False* values in the line positions or a list with the line positions as given by the *line_spectrum* method of the *NirdustSpectrum* class. This method uses one of those imputs to remove points from the spectrum.

Parameters

- **line_intervals** (*python iterable*) – Any iterable object with pairs containing the beginning and end of the region were the spectral lines are. The second return of *line_spectrum* is valid.
- **mask** (*boolean array*) – array with same length as the spectrum containing boolean values with *False* values in the indexes that should be masked.

Returns
A new instance of *NirdustSpectrum* class with the especified intervals removed.

Return type
NirdustSpectrum object

normalize()

Normalize the spectrum to the unity using the mean value.

Returns
out – New instance of the *NirdustSpectrun* class with the flux normalized to unity.

Return type
NirsdustSpectrum object

property spectral_dispersion

Assume linearity to compute the spectral dispersion.

property spectral_length

Total number of spectral data points.

property spectral_range

First and last values of spectral_axis.

nirdust.core.**public_members_asdict**(*object*)

Thin wrapper around attr.asdict, that ignore all private members.

3.3.2 nirdust.preprocessing module

Collection of preprocessing utilities.

nirdust.preprocessing.**line_spectrum**(*spectrum*, *noise_factor=3*, *window=50*)

Construct the line spectrum.

Uses various *Specutils* features to fit the continuum of the spectrum, subtract it and find the emission and absorption lines. Then fits all the lines with gaussian models to construct the line spectrum using *astropy.models.Gaussian1D*.

Parameters

- **spectrum** (*NirdustSpectrum* object) – A spectrum stored in a *NirdustSpectrum* class object.
- **noise_factor** (*float*) – Same parameter as in *specutils.fitting.find_lines_threshold*. Factor multiplied by the spectrum's`uncertainty`, used for thresholding. Default is 3.
- **window** (*float*) – Same parameter as in *specutils.fitting.fit_lines*. Width of the region around each line of the spectrum to use in the fitting. If None, then the whole spectrum will be used in the fitting. *Window* is used in the Gaussian fitting of the spectral lines. Default is 50 (Å).

Returns

out – Returns in the first element a NirdustSpectrum of the same lenght as the original spectrum containing the fitted lines. In the 2nd position, returns the intervals where those lines were found determined by 3-sigma values around the center of the line.

Return type

NirdustSpectrum, Quantity

nirdust.preprocessing.**match_spectral_axes**(*first_sp*, *second_sp*, *scaling='downscale'*, *clean=True*)

Resample the higher resolution spectrum.

Spectrum_resampling uses the *spectral_axis* of one imput spectrum to resample the *spectral_axis* of the otherone, depending on the *scaling* parameter. To do so this function uses the *FluxConservingResampler* class of *Specutils*. The order of the input spectra is arbitrary and the order in the output is the same as in the input. It is recomended to run this function after the class methods 'cut_edges' and 'mask_spectrum'.

Parameters

- **first_sp** (*NirdustSpectrum* object) –
- **second_sp** (*NirdustSpectrum* object) –
- **scaling** (*string*) – If *downscale* the higher resolution spectrum will be resampled to match the lower resolution spectrum. If *upscale* the lower resolution spectrum will be resampled to match the higher resolution spectrum.
- **clean** (*bool*) – Flag to indicate if the spectrums have to be cleaned by *nan* values after the rescaling procedure. *nan* values occur at the edges of the resampled spectrum when it is forced to extrapolate beyond the spectral range of the reference spectrum.

Returns

out

Return type

NirdustSpectrum, *NirdustSpectrum*

3.3.3 nirdust.bbody module

Blackbody/temperature utilities.

class nirdust.bbody.**BasinhoppingFitter**(*target_spectrum*, *external_spectrum*, *basinhopping_kwargs*)

Bases: object

Basinhopping fitter class.

Fit a BlackBody model to the data using scipy modeling methods.

target_spectrum

Instance of NirdustSpectrum containing the central spectrum.

Type
NirdustSpectrum object

external_spectrum

Instance of NirdustSpectrum containing the external spectrum.

Type
NirdustSpectrum object

basinhopping_kwargs

Dictionary of keyword arguments to be passed to the scipy basinhopping routine. Read the documentation for a detailed description: docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.basinhopping.html

Type
dict

fit(*x0*, *minimizer_kwargs*)

Start fitting computation.

Parameters

- **x0** (*tuple*) – Vector indicating the initial guess values in order, i.e: (T, alpha, beta, gamma). Default: (1000.0, 8.0, 9.0, -5.0)
- **minimizer_kwargs** (*dict*) – Extra keyword arguments to be passed to the local minimizer.

Returns
results – Results of the fitting procedure.

Return type
NirdustResult object

property ndim_

Return number of fittable parameters.

run_model(*x0*, *minimizer_kwargs*)

Run fitter given an initial guess.

Parameters

- **x0** (*tuple*) – Vector indicating the initial guess values in order, i.e: (T, alpha, beta, gamma). Default: (1000.0, 1.0, 1.0, 1.0)
- **minimizer_kwargs** (*dict*) – Extra keyword arguments to be passed to the local minimizer.

Returns
results – Results of the local minimizer.

Return type
OptimizeResult object

exception nirdust.bbody.**ConvergenceWarning**

Bases: RuntimeWarning

Raised when the fitting procedure failed to converge.

class nirdust.bbody.**NirdustParameter**(*name*, *value*, *uncertainty=None*)

Bases: object

Parameter representation.

name

Parameter name.

Type
str

value

Expected value for parameter after fitting procedure.

Type
scalar, *~astropy.units.Quantity*

uncertainty

Uncertainties associated to the fitted value.

Type
scalar, *~astropy.units.Quantity*

class nirdust.bbody.**NirdustResults**(*temperature*, *alpha*, *beta*, *gamma*, *fitted_blackbody*, *target_spectrum*, *external_spectrum*, *minimizer_results*)

Bases: object

Create the class NirdustResults.

Storages the results obtained with BasinhoppingFitter plus the dust spectrum. The method plot() can be called to plot the spectrum and the blackbody model obtained in the fitting.

temperature

Parameter object with the expected blackbody temperature and its uncertainty.

Type
NirdustParameter

alpha

Parameter object with the expected alpha value and its uncertainty. Note: No unit is provided as the intensity is in arbitrary units.

Type
NirdustParameter

beta

Parameter object with the expected beta value and its uncertainty. Note: No unit is provided as the intensity is in arbitrary units.

Type
NirdustParameter

gamma

Parameter object with the expected gamma value and its uncertainty. Note: No unit is provided as the intensity is in arbitrary units.

Type
NirdustParameter

fitted_blackbody

BlackBody instance with the best fit value of temperature.

Type
~astropy.modeling.models.BlackBody

target_spectrum

Instance of NirdustSpectrum containing the central spectrum.

Type
NirdustSpectrum object

external_spectrum

Instance of NirdustSpectrum containing the external spectrum.

Type
NirdustSpectrum object

minimizer_results

Instance of OptimizeResult that generates in the fitting procedure.

Type
OptimizeResult object

plot(*axes=None*, *data_kws=None*, *model_kws=None*, *show_components=False*)

Build a plot of the fitted spectrum and the fitted model.

Parameters

- **axes** (tuple of `matplotlib.pyplot.Axis` objects) – Tuple with two objects of type Axes containing complete information of the properties to generate the image, by default it is None.
- **data_kws** (`dict`) – Dictionaries of keyword arguments. Passed to the data plotting function.
- **model_kws** (`dict`) – Dictionaries of keyword arguments. Passed to the model plotting function.
- **show_components** (`bool`) – Flag to indicate if the three components of the model should be plotted.

Returns
out – The axis where the method draws.

Return type
`matplotlib.pyplot.Axis` :

nirdust.bbody.**alpha_vs_beta**(*theta*, *target_spectrum*, *external_spectrum*)

Alpha term positivity relative to beta term.

Here we assume that: alpha * ExternalSpectrum > 10**beta * BlackBody in mean values.

Parameters

- **theta** (*array-like*) – Parameter vector: (temperature, alpha, beta, gamma).

- **target_spectrum** (*NirdustSpectrum object*) – Instance of NirdustSpectrum containing the total central spectrum.
- **external_spectrum** (*NirdustSpectrum object*) – Instance of NirdustSpectrum containing the external spectrum.

Returns
alpha_positivity – The difference between alpha term and beta term mean values, given the data; i.e.: mean(alpha * ExternalSpectrum) - (10**beta * BlackBody)

Return type
scalar

nirdust.bbody.**fit_blackbody**(*target_spectrum*, *external_spectrum*, *x0=None*, *bounds=None*, *gamma_target_fraction=0.05*, *seed=None*, *niter=200*, *stepsize=1*)

Fitter function.

Fit a BlackBody model to the data using Markov Chain Monte Carlo (MCMC) sampling of the parameter space using the emcee implementation. This function serves as a wrapper around the NirdustFitter class.

Parameters

- **target_spectrum** (*NirdustSpectrum object*) – Instance of NirdustSpectrum containing the nuclear spectrum.
- **external_spectrum** (*NirdustSpectrum object*) – Instance of NirdustSpectrum containing the external spectrum.
- **x0** (*tuple, optional*) – Vector indicating the initial guess values of temperature, alpha, beta and gamma.
- **bounds** (*tuple*) – Tuple of 4 pairs of values indicating the minimum and maximum allowed values of the fitted parameters. The order is: T, alpha, beta, gamma. Example: bounds = ((0, 2000), (0, 20), (6, 10), (-10, 0))
- **gamma_target_fraction** (*float*) – Maximum fraction of gamma vs target flux allowed to constraint the gamma value in the fitting procedure. Default: 0.05.
- **seed** (*int*) – Random number generation seed for the basinhopping algorithm.
- **niter** (*int*) – Number of basinhopping iterations. This numbers represents how many times the local minimizer will be excecuted. Default: 200.
- **stepsize** (*float*) – Maximum step size for use in the random displacement of x0 for each basinhopping iteration. Default: 1.0.

Returns
result – Instance of NirdustResults after the fitting procedure.

Return type
NirdustResults object

nirdust.bbody.**make_constraints**(*args*, *gamma_fraction*)

Make scipy minimizer constraints.

Parameters

- **args** (*tuple*) – Extra arguments to be passed to likelihood and model functions. args = (target_spectrum, external_spectrum)
- **gamma_fraction** (*float*) – Maximum fraction allowed to constraint the gamma value in the fitting procedure.

Returns
contraints – Constraints as required by the scipy SLSQP minimizer.

Return type
tuple

nirdust.bbody.**make_gamma_vs_target_flux**(*gamma_fraction*)

Encapsulate gamma_fraction for gamma constraint function.

Parameters
gamma_fraction (*scalar*) – Value between [0, 1] representing the maximum fraction of Target flux allowed for the gamma term.

Returns
gamma_vs_target_flux – Function that computes the gamma term positivity relative to the total target flux. Call signature: gamma_vs_target_flux(theta, target_spectrum, external_spectrum)

Return type
function

nirdust.bbody.**make_minimizer_kwargs**(*args*, *bounds*, *constraints*, *options=None*)

Make scipy minimizer keyword arguments.

Parameters

- **args** (*tuple*) – Extra arguments to be passed to likelihood and model functions. args = (target_spectrum, external_spectrum)
- **bounds** (*tuple*) – Tuple of 4 pairs of values indicating the minimum and maximum allowed values of the fitted parameters. The order is: T, alpha, beta, gamma. Example: bounds = ((0, 2000), (0, 20), (6, 10), (-10, 0))
- **contraints** (*tuple*) – Constraints as required by the scipy SLSQP minimizer.
- **options** (*dict*) – Extra options to be passed to the local minimizer through the *options* keyword. Default: {"maxiter": 1000}

nirdust.bbody.**negative_gaussian_log_likelihood**(*theta*, *target_spectrum*, *external_spectrum*)

Negative Gaussian logarithmic likelihood.

Compute the negative likelihood of the model represented by the parameter theta given the data. The negative sign is added for minimization purposes, i.e. finding the maximum likelihood parameters is the same as minimizing the negative likelihood.

Parameters

- **theta** (*array-like*) – Parameter vector: (temperature, alpha, beta, gamma).
- **target_spectrum** (*NirdustSpectrum object*) – Instance of NirdustSpectrum containing the total central spectrum.
- **external_spectrum** (*NirdustSpectrum object*) – Instance of NirdustSpectrum containing the external spectrum.

Returns
loglike – Negative logarithmic likelihood for parameter theta.

Return type
scalar

nirdust.bbody.**target_model**(*external_spectrum*, *T*, *alpha*, *beta*, *gamma*)

Compute the expected spectrum given a blackbody prediction.

Parameters

- **external_spectrum** (*NirdustSpectrum object*) – Instance of NirdustSpectrum containing the external spectrum.
- **T** (*float*) – BlackBody temperature in Kelvin.
- **alpha** (*float*) – Multiplicative coefficient for external_flux.
- **beta** (*float*) – Multiplicative coefficient for blackbody.
- **gamma** (*float*) – Additive coefficient.

Returns
prediction – Expected flux given the input parameters.

Return type
~numpy.ndarray

3.3.4 nirdust.io module

NIRDust Input/Output.

exception nirdust.io.**HeaderKeywordError**

Bases: KeyError

Raised when header keyword not found.

nirdust.io.**infer_fits_science_extension**(*hdulist*)

Auto detect fits science extension using the provided keywords.

Parameters
hdulist (*~astropy.io.fits.HDUList*) – Object containing the FITS extensions.

Returns
extensions – Array with the science extensions indeces in the hdulist.

Return type
~numpy.array

nirdust.io.**pix2wavelength**(*pix_arr*, *header*, *z=0*)

Transform pixel to wavelength.

This function uses header information to perform WCS transformation.

Parameters

- **pix_arr** (float or *~numpy.ndarray*) – Array of pixels values.
- **header** (*FITS header*) – Header of the spectrum.
- **z** (*float*) – Redshift of object. Use for the scale factor 1 / (1 + z).

Returns
wavelength – Array with the spectral axis.

Return type
~numpy.ndarray

nirdust.io.**read_fits**(*file_name*, *extension=None*, *z=0*)

Read a spectrum in FITS format and store as a *NirdustSpectrum* object.

Parameters

- **file_name** (*str*) – Path to where the fits file is stored.
- **extension** (*int or str*) – Extension of the FITS file where the spectrum is stored. If None the extension will be automatically identified by searching relevant header keywords. Default is None.
- **z** (*float*) – Redshift of the galaxy. Used to scale the spectral axis with the cosmological sacle factor 1/(1+z). Default is 0.

Returns
out – Returns an instance of the class *NirdustSpectrum*.

Return type
NirsdustSpectrum object

nirdust.io.**read_table**(*file_name*, *wavelength_column=0*, *flux_column=1*, *format='ascii'*, *z=0*, ***kwargs*)

Read a spectrum from a table and store it in a *NirdustSpectrum* object.

The table must contain two columns for the wavelength and the intensity/flux, the column number can be specified by parameters. It is assumed that the unit of the wavelength axis is Å.

Parameters

- **file_name** (*str*) – Path to where the fits file is stored.
- **wavelength_column** (*int*) – The positional number of the wavelengh column. Default is 0.
- **flux_column** (*int*) – The positional number of the intensity/flux column. Default is 1.
- **kwargs** – Args from``astropy.table.Table.read``.

Returns
out – Returns an instance of the class *NirdustSpectrum*.

Return type
NirsdustSpectrum object

nirdust.io.**spectrum**(*flux*, *header*, *z=0*)

Instantiate a *NirdustSpectrum* object from FITS parameters.

Parameters

- **flux** (*~astropy.units.Quantity*) – Intensity for each pixel in arbitrary units.
- **header** (*FITS header*) – Header of the spectrum.
- **z** (*float*) – Redshif of the galaxy.

Returns
spectrum – Return a instance of the class NirdustSpectrum with the entered parameters.

Return type
NirsdustSpectrum

3.4 Licence

CHAPTER

FOUR

REQUERIMENTS

You will need Python 3.8 or higher to run NIRDust.

CHAPTER

FIVE

REPOSITORY AND ISSUES

To view NIRDust source code visit the repository: https://github.com/Gaiana/nirdust
If you find any issues or bugs please let us know here: https://github.com/Gaiana/nirdust/issues

CHAPTER

SIX

AUTHORS

Gaia Gaspar (gaiagaspar@gmail.com)
Jose Alacoria (josealacoria@gmail.com)
Juan B. Cabral (jbc.develop@gmail.com)
Martin Chalela (tinchochalela@gmail.com)

PYTHON MODULE INDEX

n

INDEX

P

R

S

T

U

V