PyQSOFit_SVL Manual

Jack Hon

May 2024

1 Introduction

PyQSOFit_SVL is a modified version of PyQSOFit from `https://github.com/legolason/PyQSOFit`. PyQSOFit is a spectrum analysis tool via gaussian line modelling that utilises least squares fitiing. It is made for SDSS active galactic nuclei (AGN) spectra but is not specialised for it, meaning that it can be generalised to any spectrum from any instrument with ease.

By default, PyQSOFit can decompose a spectrum into its host galaxy part and AGN part. It then models the AGN part for the continuum emission, and finally the emission lines. For continuum, PyQSOFit can handle the Balmer continuum, FeII emission complex, and the AGN powerlaw. For emission lines, PyQSOFit can fit narrow or broad lines with a normal Gaussian profile.

The reason for the modification to PyQSOFit_SVL is to make it better suited for my project needs. These modifications are:

- Skewed Gaussian profile for asymmetric feature modelling
- Voigt Gaussian profile for Narrow Line Seyfert 1 AGN modelling
- Negative flux for absorption feature modelling
- An alternate continuum modelling method for badly behaving continuum
- A line linking method that allows for more control for how linked lines interact
- A method for defining lines to model such that it is intuitive and more readable.

2 Line Linking

This is a method in PyQSOFit_SVL that grants control over how different lines can influence one another. It's main use is to simplify the number of fitted parameters. For example, the common [OIII]$\lambda\lambda 4959, 5007$ doublet is physically emitted from the same material and therefore should have the same line profile except [OIII]$\lambda 5007$ is about 3 times stronger than its counterpart. Line linking allows the user to define these two lines such that the model profile parameters of [OIII]$\lambda 4959$, which are sigma, skew, mean, and gamma, follows exactly from [OIII]$\lambda 5007$. It also allows the definition of [OIII]$\lambda 4959$ flux scale to be [OIII]$\lambda 5007 \times 0.33$.

This is a useful technique to constrain lines that are in noisy regions. For example, [OIII]$\lambda 5007$ is often stronger than the noise level of the spectrum, but [OIII]$\lambda 4959$ is not. Linking the lines allows [OIII]$\lambda 4959$ to be fitted in a more sensible manner, rather than having the model fit the noise.

It can also help deblend line complex that would have degeneracy otherwise. For example, the Hα line is blended with the [NII]$\lambda\lambda 6549, 6585$ doublet. But by linking Hα to Hβ, it helps constrain

the possible profiles of Hα. One can also link the [NII]$\lambda\lambda$6549, 6585 doublet to [OIII]λ5007 if it is sensible to do so.

Finally, line linking can help deblend and properly fit broad emission lines. For example, Hβ narrow emission lines are often much weaker than Hα. This means we can use Hα to constrain the profile of Hβ narrow line, and hence narrowing down the possibility of Hβ broad line + narrow line combinations. This method can also be used on [OIII]$\lambda\lambda$4959, 5007 to reveal AGN outflows.

3 Using PyQSOFit_SVL

I will demonstrate how I use PyQSOFit_SVL for my projects as an example. I always have two .py files, (1) Component_definitions.py to define the Gaussian profiles I want to model with, and (2) 6dfgs_fitting.py to open the 6dFGS spectrum and call PyQSOFit_SVL to process it.

File (1) has a strict format that it should follow because these parameters are stored and input into PyQSOFit_SVL when file (2) calls it. Whereas file (2) can be anything necessary for the user to tweak their spectrum into a format the PyQSOFit_SVL accepts.

While I present my example, I will keep the mentioned .py names for file (1) = Component_definitions.py and file (2) = 6dfgs_fitting.py.

3.1 Component_definitions.py

The purpose of this file is to define how to analyse a spectrum. We start by deciding how to section off the spectrum based on the lines we want to fit and link, then define the lines in that section, combine all sections together, and finally turn it into a .fits file with ASTROPY

Important to keep in mind that line linking only works of the linked lines are within the same section. Because PyQSOFit_SVL performs the line modelling section by section, if the linked lines are separated, the code will not work.

For low redshift AGN, I typically define 1 section that encompasses Hβ, [OIII]$\lambda\lambda$4959, 5007, Hα, and [NII]$\lambda\lambda$6549, 6585, with a section wavelength range of 4500Å to 6700Å. Doing so allows me to link Hα to Hβ, [OIII]λ4959, will have "[] to [OIII]λ5007, and [NII]λ6549 to [NII]λ6585. I can also change to have both [NII]$\lambda\lambda$6549, 6585 linked to [OIII]λ5007 if needed.

Alternatively, if one does not need to link Hβ to Hα, two sections can be defined. One contains Hβ and [OIII]$\lambda\lambda$4959, 5007 with a section range of 4500Å-5300Å. The other will have Hα and [NII]$\lambda\lambda$6549, 6585, from 6000Å-7000Å.

3.1.1 Section definition

To define a section, call the *Section* class from PyQSOFit_SVL. This class requires a *section_name*, *start_range*, and *end_range* to initialise.

For example, a two section definition as mentioned above can be:

```
from PyQSOFit.PyQSOFit_SVL import Section

hbo3_section = Section(section_name="O3", start_range=4500, end_range=5300)
hanl_section = Section(section_name="HA", start_range=6000, end_range=7000)
```

3.1.2 Line definition

To define a line for modelling, call the *LineDef* class from PyQSOFit_SVL. This class requires a name and centre wavelength for the line to initiate. A basic example for line definition would be:

```
from PyQSOFit.PyQSOFit_SVL import LineDef

line_hb_br1 = LineDef(l_name="Hb_br1", l_center=4861.33, scale=0.005,
                      default_bel=True)
line_hb_na = LineDef(l_name="Hb_na", l_center=4861.33, scale=0.002,
                     fwhm=(50, 700), voffset=1e-3, skew=(0,))
```

Then one can insert these lines into a section via:

```
hbo3_section.add_lines([line_hb_br1, line_hb_na])
```

The full description and keyword arguments of the line includes:

- l_name. A string for the name of the line. Must be unique.
- l_center. The rest wavelength for the line.
- fwhm. Accepts a python tuple with one or two values in km/s. One value input as (500,) will fix the fwhm of the modelling profile at 500. Two values input as (1200, 7000) will have the fwhm range from 1200, 7000. Values are not exact due to conversion to a log scale during modelling.
- profile_link. A string that tells PyQSOFit_SVL which line the profile of this one is linked to. For example, definition for [OIII]λ4959, will have "OIII5007*1" as profile_link. This overwrites fwhm, meaning fwhm can be ignored.
- skew. A python tuple with one or two values. One value input as (2.3,) will fix the skew of the modelling profile at 2.3. Two values input as (-10, 10) will have the skew range from -10 to 10.
- gamma. A string that controls the Voigt profile for the modelling. By default, it is an empty string and will turn off Voigt profile and use Gaussian profile instead. If specified "On", it will have a free gamma value and uses Voigt profile. If the string has "f0.53", it will fix the gamma at 0.53.
- voffset. A value in km/s for the range of velocity offset allowed from the centre wavelength.
- vmode. A string that controls how the voffset range is defined. Default is "", and means the offset is free on both ends. "+" restricts the range such that the velocity offset can only increase in wavelength. "-"" restricts the range such that the velocity offset can only decrease in wavelength. "." sets the velocity offset at the specified voffset value.
- scale. A value that controls the scale of the modelling profile.
- flux_link. A string that tells PyQSOFit_SVL which line the flux of this one is linked to. For example, definition for [OIII]λ4959, will have "OIII5007*0.33" as flux_link. This overwrites scale, meaning scale can be ignored.
- default_bel. A boolean. if True will set the fwhm to (1200, 7000), voffset to 5e-3, and skew to (-10, 10).
- default_nel. A boolean. if True will set the sig to (50, 1200), voffset to 1e-3, and skew to (0,).

3.1.3 Saving the file

To save the file, implement the following code:

```
newdata = np.concatenate([hbo3_section.lines, hanl_section.lines])
hdu = fits.BinTableHDU(data=newdata, header=Section.hdu_generate(), name='data')

# path definitions to save the component definitions file
path1 = os.getcwd() + "/PyQSOFit/"  # path of the source code file and qsopar.fits
hdu.writeto(path1 + "qsopar2.fits", overwrite=True) # leave the name as qsopar2
```

3.2 6dfgs_fitting.py

The purpose of this file is to load in spectra, clean and process it, then call PyQSOFit_SVL to fit it. As such, depending on what spectrum the user wants to fit, the code for loading, cleaning, and processing will be different. This manual will not focus on how to perform these operations, rather focus on the PyQSOFit_SVL part.

3.2.1 Basic fitting

Once the spectrum is ready, initiate PyQSOFit_SVL by using the following

```
path1 = os.getcwd() + "/PyQSOFit/"  # path to PyQSOFit directory
q = QSOFit(lam=wave, flux=flux, err=err, z=0.053, path=path1)
```

wave is the wavelength of the spectrum, flux is the flux, err can be the variance spectrum of the flux. If the variance spectrum does not exist, try using np.ones_like(flux).

To fit, call the following function

```
q.Fit(decomposition_host=True, PL=True, poly=True, Fe_uv_op=False, BC=True,
      nsmooth=1, redshift=True, linefit=True, plot_fig=True, save_fig=False,
      save_fig_path=None, save_result=False, save_fits_path=None, save_fits_name=None
    )
```

In this instance, the arguments are:

- decomposition_host. If True will perform principal component analysis (PCA) to fit host galaxy and separate from AGN.
- PL. If True, will include a power-law with index from -5 to 3 for AGN continuum fitting
- poly. If True, will include a 3rd order polynomial for AGN continuum fitting
- Fe_uv_op. If True, will include a FeII template for fitting
- BC. If True, will model the Balmer continuum
- nsmooth. Value for the gaussian smoothing sigma
- redshift. If True, will correct for the redshift of spectrum
- linefit. If True, will fit the emission lines

The remaining ones are self-explanatory.

3.2.2 Obtaining fitting results

Once the fitting has completed, the results can be obtain in a few ways. If the plot is all you need, ensure the save_fig_path is defined, plot_fig and save_fig both set to True.

If the machine format will be saved into a fits file is save_result is True, and save_fits_path and save_fits_name are defined. It can also be printed in the console with

```
q.full_result_print()
```

If only specific line properties are desired, use the line_result_output function like so:

```
HB_result = q.line_result_output("Hb_br", to_print=True)
print(HB_result["area"])
```

If the function is called with to_print as True, it will print on the console a neat table. Alternatively, one can set it to false, save it into a variable such as HB_result, then call the specific measurement from this list: fwhm, sigma, skew, ew, peak, area

While it is up to the user how they wish to save the results, I personally created the SixDFGSFitter.save_result function to do it.

```
a_spec = SixDFGSFitter(file_path=a_path, z=xz)

a = q.line_result_output("Hb_br")
b = q.line_result_output("Hb_na")
c = q.line_result_output("OIII5007c")
d = q.line_result_output("Ha_br")
e = q.line_result_output("NII6549c")

a_spec.save_result(spec_id=22878, result=[a, b, c, d, e],
                   save_properties=["fwhm", "skew", "peak", "area"],
                   save_path="xx.txt", save_error=False)
```

This code runs PyQSOFit_SVL, then saves the five lines into a, b, c, d, and e. It then calls the .save_result function, where I specify the row name with spec_id, the list of result I want to save, the measured properties I want, and the path. save_error if False will save the measured values, and if True, will save the errors only instead (if errors are measured).

3.2.3 Error Estimation

PyQSOFit_SVL estimates fitting uncertainty by bootstrapping. This can be turned on with

```
q.Fit(decomposition_host=True, PL=True, poly=True, Fe_uv_op=False, BC=True,
      linefit=True, MC=True, n_trials=50)
```

The MC argument set to True will turn on the bootstrapping, and n_trials will dictate how many iteration the bootstrapping will perform. Each iteration is essentially fitting the spectrum one more time, so this operation might be time consuming.

At each iteration, the spectrum is randomly varied before re-fitting. The spectrum is varied by (1) sampling random points of the residual, (2) randomly shifting those points according to a normal distribution, (3) fitting a 3rd order polynomial to smooth out the randomness, (4) applying this to the continuum subtracted line. This has the net effect of fitting under a slightly different continuum subtraction. This provides a more accurate error estimation of the emission line measurements because the main systematic uncertainty for the fitting results arises from continuum fitting.

4 Optional and Advanced usage

4.1 Continuum fitting tool

There are spectra that defy physical models and have a continuum that cannot be reproduced by any combinations available to PyQSOFit. These could be due to observational artefacts or errors from post-processing of the data. To handle these, PyQSOFit_SVL has a different continuum fitting tool (CFT), that approximates the continuum rather than modeling it.

CFT is similar to the online spectrum fitting tool known as Marz (`https://samreay.github.io/Marz/#/overview`). In Marz, pixels with high statistical significance are iteratively removed by estimating the continuum with a polynomial fitting of order 8. This polynomial line acts as the mean at each pixel to define the variance of the spectrum. The iterative process erodes away emission and absorption features until a close fit to the underlying continuum remains. For CFT, a Gaussian smoothing is used to increase the flexibility of the function. The downside to this is that broad emission features will not be perfectly removed, but the amount of area loss can be controlled and optimised by varying the strength (σ factor) of the smoothing. Narrow line features are unaffected by this issue.

To use CFT in PyQSOFit_SVL, call q.Fit as below:

```
q.Fit(CFT=True, CFT_smooth=75, linefit=True, MC=True, n_trials=50)
```

Since CFT ignores all other continuum components of PyQSOFit, those are not relevant parameters anymore. The default value of CFT_smooth is set to 75. A higher value will erode less emission/absorption features but will be less accurate to the continuum. A lower value improves accuracy, but will impact the emission/absorption features more. This is demonstrated in Figure 1.

4.2 Two component fitting

If two Gaussian components are required for a fit, typically for resolved double-peaked profile, this can be defined via

```
line_hb_br1 = LineDef(l_name="Hb_brR", l_center=4861.33, scale=0.005,
                      fwhm=(1200, 5000), voffset=5e-3, vmode="+")
line_hb_br2 = LineDef(l_name="Hb_brL", l_center=4861.33, scale=0.005,
                      fwhm=(1200, 5000), voffset=5e-3, vmode="-")
```

Instead of calling the default_bel as True, the two lines are defined manually with unique names of 'Hb_brR' and 'Hb_brL'. Because this example is to model a double-peak profile, 'Hb_brR' is set to have a vmode of '+' to restrict the velocity offset to only increase wavelength. Meanwhile, 'Hb_brL' is set to have a vmode of '-' to restrict velocity offset to only decrease wavelength.

Another example might be when the line should be fitted with multiple components, i.e. when the broad [OIII] outflow is visible with the narrow lines.

```
line_or = LineDef(l_name="OIII5007N", l_center=5006.843, scale=0.003, default_nel=
    True)
line_or = LineDef(l_name="OIII5007B", l_center=5006.843, scale=0.005,
                  fwhm=(1200, 3000), voffset=5e-3)
```

The point of setting the names to be similar in the first few characters is so that their total profile measurements can be easily calculated via:

```
a = q.line_result_output("Hb_br")
b = q.line_result_output("OIII5007")
```

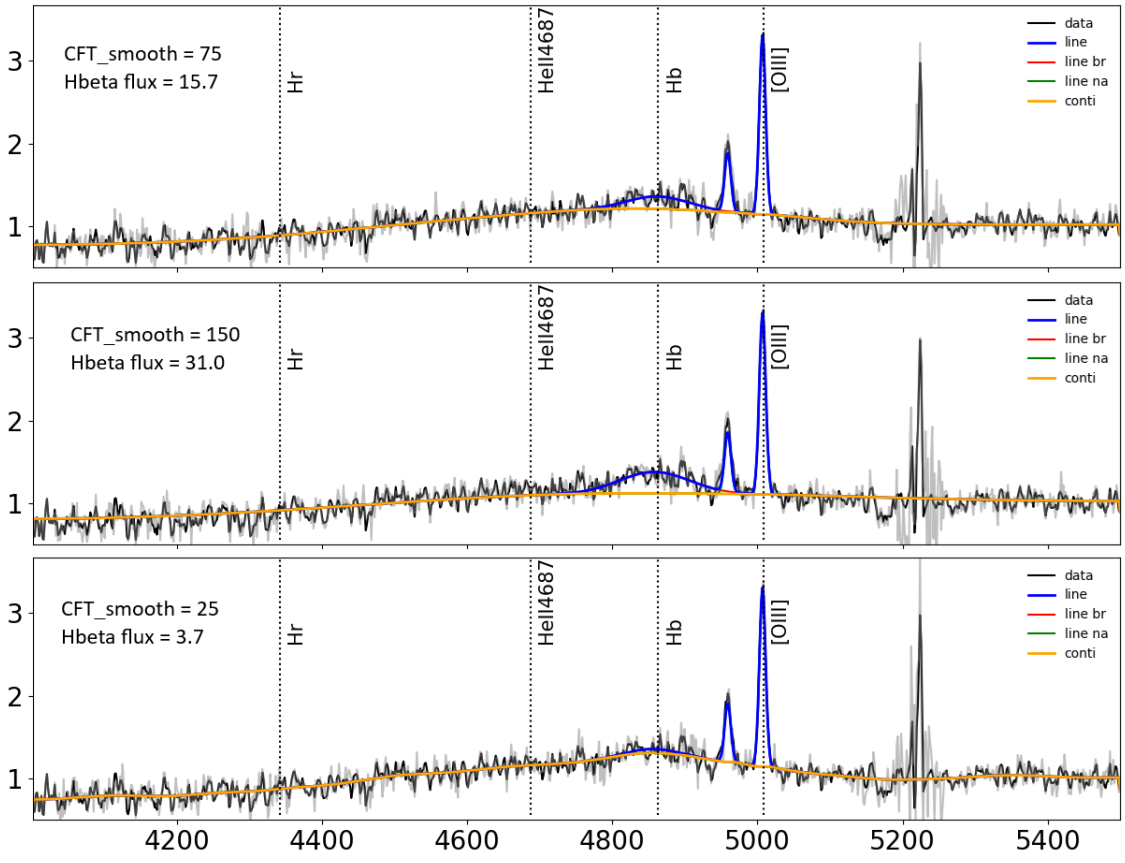

Figure 1: Figure showing an example of CFT over three plots with varying CFT_smooth and its impact on the broad emission flux.

By calling .line_result_output like so, the peak of the combined lines is given as the mean of the lines weighted by their area. The FWHM of the combined lines is the outer most edge of the combined profile. The skew is no longer applicable and is given as -999 with an error of -999.

While individual measurements can still be obtained via:

```
a = q.line_result_output("Hb_brR")
a = q.line_result_output("Hb_brL")
```

4.3 Voigt Profiles

While Voigt profiles is a feature in PyQSOFit_SVL, it is not fully tested. Using Voigt profiles might not produce consistent result with the FWHM that is set because the FWHM is calculated for Gaussian profiles.

One can define a line to use Voigt profile by setting gamma as "On" or if the exact value of gamma is known (to be 0.53 for example), gamma="f0.53":

```
line_hb_br1 = LineDef(l_name='Hb_br1', l_center=4861.33, scale=0.005,
                      fwhm=(1200, 5000), default_bel=True, gamma="On")
line_hb_br1 = LineDef(l_name='Hb_br1', l_center=4861.33, scale=0.005,
                      fwhm=(1200, 5000), default_bel=True, gamma="f0.53")
```

The value of gamma will not be reported from .line_output_result. If required, it will be in .full_result_print() instead.

4.4 Noise Area

To gauge if the line fitting has produced good results, one can look at the reduced chi-squared that is listed in .full_result_print(). However this often does not inform the user if the defined component has modelled an actual emission feature or just fitted noise that looked like a feature.

.average_noise_area is a value that provides an estimate to the area of a noise feature. By comparing the area of modelled features with this value, one can gauge if an emission feature truly exist or not. This value is calculated from the final subtracted residual with continuum and emission lines removed. The area of the residual is then averaged over the number of noise peaks that are $> 3\sigma$.

Narrow features can be compared directly. However broad features will require the flux to be scaled, with a factor that is relative to the FWHM between broad and narrow features, i.e.

$$\text{flux(if\_noise)} = \text{flux(broad)} \times \text{FWHM(narrow)/FWHM(broad)} \quad (1)$$

A comparison between broad features and noise area is not conclusive and requires intuition to understand if the modelled feautre is indeed noise. Use this as a guide carefully.