PyParadise User Manual

B. Husemann[1], O. S. Choudhury[2], C. J. Walcher[2]

[1]*European Southern Observatory, Karl-Schwarzschild-Str. 2, 85748 Garching b. München, Germany*
[2]*Leibniz-Institut für Astrophysik Potsdam (AIP), An der Sternwarte 16, 14482 Potsdam, Germany*

19 November 2019

ABSTRACT

1 INSTALLATION

1.1 Software dependencies

PyParadise depends on the following software:

- python version 2.7 or 3.X
- numpy, version 1.12.1 or later
- scipy, version 0.19.0 or later
- astropy, version 2.X or later

For the MCMC algorithm two options are available of which only one needs to be installed:

- pymc, version 2.3.6 or later
- emcee, version 2.2.1 or later

While different combination of package version may work as well, we suggest to create a dedicated python environment with miniconda (https://docs.conda.io/en/latest/miniconda.html). All packages can be easily install within the conda environment using the command

```
conda install PACKAGE=VERSION
```

This allows the user to select the necessary specific versions described above to be installed under a common environment. This ensures stable operations of PyParadise. Running PyParadise with different version than the once listed above is at the users risk.

1.2 Package Installation

In order to install PyParadise, you can obtain the code from BitBucket, and install by running the following command from the main directory (ideally within the prepared conda environment)

```
python setup.py install
```

If the user does not want to create a dedicate environment with conda, you need to consider running the command with superuser-permissions or to change the command to either of the following commands

```
python setup.py install --user
python setup.py install --home
```

while the preferred option is clearly the first, the second option requires a correct setup of the `PYTHONPATH` environment variable.

1.3 Spectral stellar population templates library and example files

PyParadise is shiped with several spectral stellar population template libraries and some example files with the purpose to get the user familiar with the necessary parameter files, input files and output files.

Important: The installation procedure **will not** copy the template files nor the example files to a new location, but will stay in the `templates` and `examples` directory. The user is free to move thes files to any location. The location of the template files will be specified in the parameter files and is totally independent of the internal PyParadise installation directory.

2 QUICK START

Please read this section if you want to start working with PyParadise quickly. This section provides the minimum information to run PyParadise successfully on a certain data set. It is based on the example files provided with PyParadise and we recommend that new users first run PyParadise on the example files and have a careful look at them before applying PyParadise to their own data. More details on the algorithms and full explanations on all parameters and functionality is given in Sect. **??**. For the most common mistakes we refer to the toubleshooting Sect. 9 for further details.

2.0.1 List of distributed files and their purpose

2.1 Running PyParadise

PyParadise can be applied on single spectra, a collection of spectra (i.e. row-stacked spectra or RSS), and a 3-dimensional cube from integral field spectroscopy. The necessary data format for all three cases is explained in Sect. 6. Below we provide the sequence of commands to process the example RSS file `NGC2691.COMB.RSS.fits`.
The **stellar continuum fit**

```
ParadiseApp.py NGC2691.COMB.RSS.fits NGC2691 6.0 --SSP_par parameters_stellar --parallel 1
```

The **emission line fit** in the residual spectra

```
ParadiseApp.py NGC2691.COMB.RSS.fits NGC2691 6.0 --line_par parameters_eline --parallel 1
```

The **bootstrap run to construct proper errors** for all parameters

```
ParadiseApp.py NGC2691.COMB.RSS.fits NGC2691 6.0 --SSP_par parameters_stellar --line_par
    parameters_eline --bootstraps 100 --parallel 4
```

In the command lines above `NGC2691` is a output file prefix, `6.0` is the spectral resolution of the input spectra in unit wavelength (FWHM), `parameter_stellar` and `parameters_eline` are setup parameter files. The number of CPU cores to be used is given by the `--parallel` and the size of bootstrap sample is given by the `--bootstraps` command line parameters.

2.2 Basic content of output files

Once the user ran the commands as indicated in the above example several output file will occur for the three stages.

2.2.1 Stellar continuum fit result

The best-fit stellar continua spectra and the residual spectra are stored as a FITS image in the files `<PREFIX>.cont_model.fits` and `<PREFIX>.cont_res.fits` with the same dimension as in the input data. An example for the resulting stellar continuum model for the test case `NGC2691` is shown in Fig. 1.
In addition, a FITS table `<PREFIX>.stellar_table.fits` is stored which lists inferred measurements from the best-fit for each spectrum. It contains the stellar kinematic parameters including errors from the MCMC run and various stellar population parameters such as luminosity-weighted age (`lum_age_total`) or luminosity-weighted iron abundance (`lum_[Fe/H]_total`). We highly recommend to use the software TOPCAT (http://www.star.bris.ac.uk/∼mbt/topcat/) to quickly visualize the output table content.

2.2.2 Emission line fit result

The best-fit emission-line model and the residual spectra are stored as a FITS image in the files `<PREFIX>.eline_model.fits` and `<PREFIX>.eline_res.fits` with the same dimension as in the input data. An example of the resulting emission line model for the test case `NGC2691` is shown in Fig. 1.
In addition, a FITS table `<PREFIX>.eline_table.fits` is stored which lists inferred measurements from the best-fit for each spectrum. For all fitted lines it lists the line flux `<LINE>_flux`, line velocity `<LINE>_vel` in km/s, and intrinsic line width `<LINE>_fwhm` in km/s.

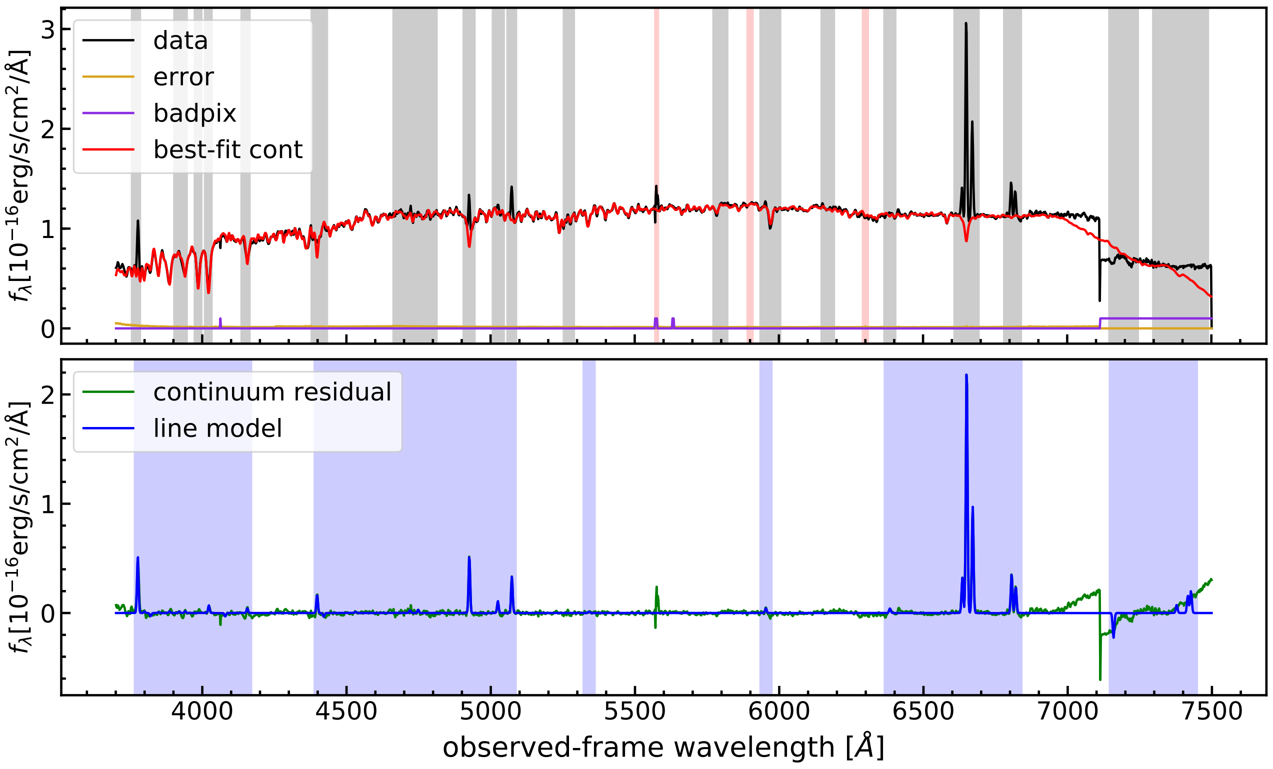

Figure 1. PyParadise results for the first spectrum in the provided RSS file NGC2619.COMB.RSS.fits. The upper panel shows the best fit stellar continuum with respect to the observed data and associated errors an bad pixel flag (==0 if good pixel). The grey and red area highlight wavelength regions masked out in the rest frame (for emission lines) and observed frame (for sky lines), respectively. The lower panel shows the best-fit emission line model with single Gaussians coupled in redshift and line dispersion. The blue area highlights the wavelength regions which are taken into accout to compute the χ^2 for the fitting of emission lines. Failure of providing a mask that does not cover all emission line components will results in wrong results.

2.2.3 Bootstrap error estimation result

The bootstraps run repeats the measurements for each spectrum N times as specified in the `--bootstraps <N>` parameter. In each repetition the input spectrum is randomly modulated within the associated error budget and only a random subsect from the template spectral library is use to model the spectrum.
Errors on each parameter are derived by taking a standard deviation of the repeated measurements and assumming a normal distribution. The tables `<PREFIX>.stellar_table.fits` and `<PREFIX>.eline_table.fits` are expanded with additional error columns for each parameter and columns names `<COLUMN>_err`.

2.3 Quick visualization of results

2.4 A quick look at the necessary parameter files

To successfully run PyParadise a correct setup of the necessary parameters are crucial. Most of the parameters may only be changed by expert users and can be left safely unchanged for most circumstances. We refer to Sect. **??** to gain deep insight into the underlying algorithms and purpose of all parameters that can be set to optimize PyParadise. Below we briefly explain the necessary files to run PyParadise and should be available in the example directory:

- `NGC2691.COMB.RSS.fits` $\rightarrow$ Input spectral data to fit
- `parameters_stellar` $\rightarrow$ Parameter file for stellar continuum fit
- `excl.cont` $\rightarrow$ Wavelength mask for continuum normalization
- `excl.fit` $\rightarrow$ Wavelength mask for continuum fitting
- `parameters_eline` $\rightarrow$ Parameter file for emission line fit
- `par.lines` $\rightarrow$ Definition of line component model
- `lines.fit` $\rightarrow$ Wavelength region considered for emission-line fit

Please check that all those files are available in the example directory of the downloaded package.

2.4.1 Example stellar continuum parameter file

The parameter file for the continuum fit is an ASCII file which sets several important parameters. Each line defines a certain parameter with a parameter name, a parameter value, and optionally followed by a comment separated with a '!' sign from the parameter. For the stellar contiuum fit an example parameter file `parameters_stellar` is available in the example directory. The most important parameters to be updated by the users are the following:

```
tmpldir ../Paradise/templates
tmplfile CB09_templates.fits
vel_guess 4040.0
vel_min 3900.0
vel_max 4200.0
start_wave 3701.0
end_wave 7499.0
```

These parameters usually need to be updated for every data set by the user. The absolute or relative path to the templates directory containing the stellar continuum template library `tmpldir` need to be set properly. Ideally a common absolute path is set and re-used for a given computer. The spectral library template file `tmplfile` need to be specified **and must exist in the specified template directory**. It is crucial to set a good guess `vel_guess` for the mean systemic redshift of the input spectra expressed in km/s as cz. The MCMC algorithm is only exploring the stellar kinematics within certain bondaries, so the minimum `vel_min` and maximum `vel_max` velocities around the systemic one needs to be chosen. Depending on the wavelength range covered and whether only a partial wavelength range should be modelled, the start `start_wave` and end `end_wave` wavelength. All other parameters can be left unchanged except the user is familiar with the details of parameters, explained in the Sect. **??**, and known the consequences.

2.4.2 Example emission line parameter file

PyParadise can also fit Gaussian emission line profiles to the residual spectra after the best-fit continuum model is subtracted. Another parameter file is needed for this second processing step which has a similar syntax as the parameter file for the continuum fit. An exmple parameter file `parameters_eline` is available in the example directory. From the file the most curcial lines to be updated by the users are:

```
eCompFile par_lines
vel_guess 4040.0
eguess_window 20
```

Those parameters usually need to be updated for every data set by the user. A dedicated file is needed to describe the emission line model to be fitted. The absolute path/file name of this emission line components file is registered in the `eCompFile` parameter. Similar to the stellar continuum model, the systemic redshift of the emission lines need to be set as an initial guess `vel_guess` expressed as cz in km/s. In most case the same initially velocity guess as for the stellar continuum model can be used. The non-linear emission line fitting needs good starting values to converge. Those starting values for each Gaussian emission line component will be empirically determined from the spectra in pixel window around the redshifted central pixel of each emission line in the spectrum. This windows may be adjusted depending on the expected width of lines given based on spectral resolution and spectral sampling.

2.4.3 Emission line model file

The associated model component file is a ASCII file that defines a list of different line components to be modelled. There is no limit on the number of components. Each component consistents of a block of several parameters depending on the specific component as described below. An example file `par.lines` is available in the example directory.

Each component block starts with a line that defines the component profile (i.e. Gaussian) and a unique component name (i.e. an identifier for the emission line). All parameters of the Gaussian such as flux, redshift (as velocity cz) and velocity dispersion in km/s, can also be tied to other lines based on the unique identifier. Also a factor to apply a fixed flux ratio is possible as indicated in case of the OIII doublet.

The example file `par.lines` lists a large number of potential emission lines of which a sub-sample of optical lines can be easily drawn for any specific case. For more complex situation and a full description of all features please read Sect. **??** for more details on the line modelling.

2.4.4 *Wavelength selection files*

Certain wavelength windows need to be masked or selected at various instances during the fitting. Such spectral windows can be defined in an ASCII file either as spectral windows in the rest-frame (e.g. for emission lines) or observed-frame (e.g. for sky line residuals or detector artefacts). Examples of such spectral window files (`excl.cont`, `excl.fit`, `lines.fit`) are available in the example directory.

PyParadise requires three wavelength selection files for different purposes: 1) exclusion of emission lines, sky line residuals and detector artefacts during the continuum normalization $\rightarrow$ `excl_cont` in `parameter_stellar`, 2) exclusion of non-continuum emission for the actual stellar continuum fit $\rightarrow$ `excl_fit` for `parameter_stellar`, and 3) wavelength windows covering ALL emission lines to be fit $\rightarrow$ `lines.fit` for `parameter_eline`. For most practical purposes `excl_cont` and `excl_fit` use the same spectral windows and can point to the same file for simplicity.

Important: Failure in providing correct and complete wavelength selection files will usually lead to incorrect fitting of the spectra. This will be recognized as certain artefacts in the results. Some examples to identify common problems in the masking will be described in Sect. **??**.

2.5 Available stellar template libraries

File name	spectra	coverage	sampling	resolution	type	Reference
CB09_templates.fits	85	3500Å–9500Å	0.9Å	2.5Å	SSP	
MIUSCAT_BaSTI_K_baseFe.fits	636	3465Å–9470Å	0.9Å	2.5Å	SSP	
RG04_templates.fits	39	3001Å–6998Å	0.3Å	0.6Å	SSP	
INDO_US_random_100.fits	100	3500Å–9400Å	0.4Å	1.0Å	stars	

3 STELLAR POPULATION MODELLING

3.1 PyParadise philosophie of fitting stellar continuum spectra

The fitting of galaxy continuum spectra is a complex process and usually needs optimization of a highly multi-dimensional parameter space. All galaxy spectra can be described by the super-position of stellar template spectra which are convolved with the kinematics of the stars as described by a radial velocity (including cosmological redshift) and a velocity dispersion. Optimizing the mix of template spectra and the kinematics together is a non-linear optimization process.
PyParadise is breaking up the non-linear kinematic optimization and the linear super-position of template spectra into 2 indpendent steps using an iterative procedure. This has the advantage that only the few non-linear parameters are fitted with non-linear optimization methods, while the super-position of template spectra are preformed with very fast linear optimization methods.
Since real spectra are never perfect and dust extinction may significantly affect the shape of the spectra and would not provide a good match to the any super-position of the template spectra. This issue is often solved by adding additional functions to the optimization, such as polynomials. However, even polynominals may not capture all the arteficial variation in the spectra correctly. Instead, PyParadise is normalizing the global continuum shape of the input spectra and template spectra in a non-parametric way before the algorithms is actually trying to match them.

Important: Due to the continuum normalization, PyParadise finds the best template super-position by reducing only the residuals of absorption line features and basically ignores the different continuum slopes of young and old stellar populations. Also the dust extinction cannot be determined during the fitting this way. Obviously this has clear advantages or disadvantages depending on the type of input spectra and the scientific objectives of the user.

3.2 Template broadening to match instrumental spectral resolution

The template library of continuum spectra has an intrinsic spectral resolution and spectral sampling, which is usually inhereted from the spectrograph the data are taken with. Before fitting the template spectra to the input data PyParadise therefore needs to match the spectral resolution by properly degrading the resolution of the template spectra, which is the first step in the process. Given that the input spectra are usually galaxy spectra with a significant redshift z, the redshifting of the templates need to be taken into account when computing the difference in spectral resolution of the templates ($\sigma_{\rm temp}$ in unit wavelength) and input data ($\sigma_{\rm data}(\lambda)$ in unit wavelength). The dispersion ($\sigma_{\rm smooth}$) of the Gaussian convolution kernel applied by PyParadise to the spectral template library is therefore given by

$$\sigma_{\rm smooth}(\lambda) = \sqrt{\sigma_{\rm data}(\lambda)^2 - \left(\left(1+\frac{v_{\rm guess}}{c}\right) \times \sigma_{\rm temp}\right)^2} \qquad (1)$$

Here, v_{guess} will be initially computed by PyParadise from the vel_guess parameter in the parameter_stellar stellar configuration file. While the templates spectra are usually generated such that the spectral resolution is constant with wavelength this may not be true for the input data. PyParadise is able to work with both cases and can adaptively adjust the width of the convolution kernel as a function of wavelength when smoothing the template library. When running PyParadiseApp.py the user needs to provide a value for the spectral resolution of the data which can either be a single float number for the spectral resolution expressed in terms of Full Width at Half Maximum (FWHM) which is $\mathrm{FWHM}_{\mathrm{data}} = 2.354 \times \sigma_{\mathrm{data}}$, or the path to an ASCII file which describes the wavelength-dependent spectral resolution. The format of the file is rather simple with just two columns for the wavelength and the spectral resolution (FWHM). When applying the wavelengt-dependent spectral resolution, PyParadise is linearly interpolating the provide wavelength grid to the redshifted wavelength grid of the template spectra. The wavelength coverage in the file therefore needs to cover the entire wavelength range of the input spectra. An exmaple file for MUSE based on the measured wavelength-dependent spectral resolution () is available in the example directory with the name MUSE_spec_res.txt.

Important: When the spectral resolution of the redshifted template spectra become worse than the input data, a correct machting of the spectra becomes impossible. This would lead to an incorrect determinations of the stellar kinematics and possible wrong mix of template spectra. PyParadise will refuse to fit but provides the minimum spectral resolution of the redshfit template spectra according to vel_guess. There are 2 options to solve that issue:

- select a template library of spectra with higher spectral resolution
- artifically set the input data spectral resolution to the redshifted spectral resolution of the templates

3.3 Continuum normalization of input data and template spectra

The next crucial step in the pre-fitting stage is the continuum normalization of the input data and the template spectra. This dependes crucially on masking all features that do not belong to the continuum such as emission lines, strong sky line residuals, detector artefact and bad pixels. While bad pixels should already be provided with the input data (see Sect. **??** for details of the input data format), wavelength regions covering emission lines and sky line residuals need to be defined by the user in the excl.cont file as described in Sect. **??** and available in the exmple directory. Here, wavelength windows can be defined in rest-frame and observed-frame seperately to aid the definition of masked wavelength regions.
The masked wavelength regions are then linearly interpolated to approximate a continuum singal in those regions. A running mean is then applied to the pure input spectra to create highly smooth version of the continuum shape. The with of the running mean average box size is set by the user in the nwidth_norm parameter in the parameter_stellar parameter file. The input spectra are then devided by their respective smoothed continuum spectra to achieve a normalization of the continuum close to 1 across the entire wavelength range. An example of the normalization process is shown in Fig. **??**. The template spectra are similarly normalized after redshifting to vel_guess and matching to the spectral resolution. Here, the box size of the running mean is automatically adjusted to match the same size in wavelength space based on the respective wavelength sampling of the input data and the template spectra.
While the subsequent fitting is done with the continuum normalized spectra, the normalization information is stored throughout the process so that the results can be de-normalized at the end of the process to go back to normal space.

Important: The process crucially depends on propoer wavelength masks provided by the user. If artefact and emission lines are not properly masked in the input spectra, PyParadise will not be able to determine a good fit. It is therefore strongly encouraged, to review the definiation of the wavelength window by performing a test fit on a few characteristic spectra and use the visualization tools to check whether the provided wavelength masks are appropriate. General rule is that the wavelength regions should be broad enough to cover all non-continuum feature but small enough that enoug continuum signal remains to do a good continuum normalization. If the input spectra are nearly entirely dominated by non-continuum signal, PyParadise would not be the ideal software, but this is rarely the case.

3.4 Determination of stellar kinematics

A Markov Chain Monte Carlo (MCMC) lies on the basis of the determination of the kinematics. The constraints are the provided limits on the values that v and σ can take. A random value for v and σ are taken between these limits, followed by a random walk consisting of n steps. At each step of the random walk, the code evaluates whether the new kinematical values improves or worsens the fit. If the fit improves, the new values are accepted.
Stationarity is the state when the mean and variance for v and σ do not change when proceeding further in the walk. The first set of steps inside a walk still depend strongly on the initial value and the MCMC is therefore not stationary for those steps. These values need to be discarded, a process referred to as *burn-in*.

Even with a proper burn-in, there can be some auto-correlation as the v_i and σ_i are based on the v_{i-1} and σ_{i-1}. When the auto-correlation is weak-to-moderate, the results from a small walk might be affected by auto-correlation. For moderate-to-strong auto-correlation, the results from a longer walk might still be influenced by auto-correlation. To counteract the auto-correlation, the chain can be *thinned* by a factor k by only taking every k^{th} sample of the chain. This then produces a correction value to v and σ, which are the values for the derived kinematics at this step of the random walk.
From the sample, $(n-m)/k$ points will be used to derive the mean and the errors of the v and the σ.

3.5 Super-position of template spectra

Before the stellar populations are fitted, there are a few preparatory steps. Due to issues that often arise with in the data reduction and analysis of spectra, the continuum level is uncertain. To address this problem, PyParadise normalises the input spectra and the template library before fitting. To normalise the spectra a running mean is applied, with the width specified in the parameters-file. Another option is to use a n^{th}-order polynomial (not implemented yet), which is especially useful for data which has not been flux-calibrated yet.
The stellar population are fitted by applying a non-negative least squares (NNLS) algorithm when comparing a spectrum to the template library of model spectra. It is non-negative as so far only light-emitting stellar population have been observed. The template library does not consist of a set of spectra which are independent from each other, and the problem is referred mathematically as an ill-posed problem. The NNLS therefore does not provide a unique solution and the recovery of the star-formation history using an NNLS algorithm usually only provides at most dozen non-zero weights, e.g. it tells that at most a dozen SSPs construct the full star-formation history of a galaxy.
We therefore provided a bootstrapping procedure, which re-fits the spectra a number of times with a subset of models. If a contributing SSP in the fiducial fit is real and is removed during the bootstrapping procedure, it should be recovered by the similar SSPs during the bootstrapping. See the paper for more details on recovery of stellar parameters from simulated galaxy spectra and a validation of bootstrapping to recover trustworthy star-formation histories.

3.6 Best-fit sellar population parameters

3.7 Template handling

3.8 Masking

There are two different masking processes. As the spectra are first normalized before the stellar populations are fitted, one needs to mask out regions which can impact the normalisation, hereafter referred as continuum masking. What one wants to include for continuum masking are strong artefacts/residuals, strong emission lines and strong absorption lines. The second masking involves features in the spectra which are not due to stellar populations features: this would be the emission lines and any artificial structure in the spectra. The emission lines and artefacts are therefore masked twice, while the strong absorption lines are only masked out for the continuum determination.

3.9 Parameters file

- `tmpldir` (`string`) is the path to the directory in which the template file is located. This might be an absolute path or a relative path.
- `tmplfile` (`string`) file contains all the template spectra, and should be formatted in the FITS format. There are two HDUs in the template file: the first one contains the spectra and the second one the stellar population parameters. The spectra in the first HDU unit are saved in RSS format, with m wavelenegth elements along the first axis and the n spectra along the second axis. The first wavelength element is given by the header keyword CRVAL1, the step in wavelength space by the header keyword CDELT1.
In order to determine what the SSP parameters of each spectrum in the template file are, this information is located in the second HDU in table format. The j^{th} entry in the table has the SSP pararmeters corresponding to the j^{th} spectrum. This table should at least have the following columns:
  - `Luminosity`, which is the current luminosity of the SSP.
  - `Mass`, which is the current mass of the SSP, usually normalized such that the initial mass of the SSP is 1 $M_{\odot}$.
  - `Age`, which is the age of the SSP.
  - `[Fe/H]`, which is the iron abundance of the SSP.
  - `[A/Fe]`, which is the [α/Fe] abundance ratio of the SSP.
  - `mass-to-light`, which is used in the conversion of luminosity-weighted stellar population properties to mass-weighted stellar population properties.
- `tmplinitspec` (`int`) The initial spectrum to start the fitting with. A good guess can lead to a faster convergence, and a lower `iterations` value might therefore be sufficient.
- `vel_guess` (`float`). `vel_guess` is the best guess for the systemic velocity corresponding to cz for the object.

- vel_min, vel_max, disp_min, disp_max (float). The parameters vel_min and vel_max are the absolute upper and lower velocity boundaries expressed as cz_{min} and cz_{max} in km/s, respectively, which define the search range for systemtic velocity during MCMC iteration. The disp_min and disp_max values similarly define the absolute boundaries for σ in km/s.
- kin_fix (0 or 1) If the kin_fix parameter is set to 1, the kinematics will not be fitted, but will instead be read from <prefix>.kin_table .fits. This table is located as the second HDU unit, and should have the values for the kinematics of each spectrum in the columns named vel_fit and disp_fit.
- excl_fit (string) Path to the file which contains the wavelength windows to exclude during the stellar absorption line fitting. This may be an absolute path or a relative path.
- excl_cont (string) Path to the file which contains the wavelength windows to exclude during the determination of the stellar continuum. This may be an absolute path or a relative path.
- nwidth_norm (int) The width of the running mean in pixel space in which the continuum will be determined.
- start_wave, end_wave (float) These wavelength limits describe between where the spectra are fitted. One can also use the exclusion files for reducing the wavelength space, but the advantage of start_wave and end_wave is that the template spectra are truncated between these wavelength limits. As it reduces the wavelength range where the spectra are normalizes, it accelerates the computation and the reduced wavelength range reduces the memory footprint of the code.
- min_x, max_x, min_y, max_y If the input spectrum is in cube-format, these parameters extract a smaller cube and fit only those spectra. So if a cube has the shape (30, 40, 200), there are 30 spectra along the x direction times 40 spectra along the y direction, and each spectrum contains 2000 wavelength elements. If min_x and max_x are set to 10 and 15, respectively, and min_y and max_y are set to 3 and 5 respectively, only the spectra along (10:15, 3:5) are fitted.
- iterations (int) The fitting of the best (combination of) SSP(s) amd the kinematics are two different computational steps. At the first iteration, the tmplinitspec is used to fit the kinematics. The kinematics obtained from this is then applied to the template library and the best-ftting (combination of) SSP(s) are obtained, which gives the SSP parameters and the best-fitted spectrum.
- samples (int) The total number of random walks performed inside an MCMC chain. The longer the chain, the more reliable becomes the determination of v and σ.
- burn (int) An MCMC chain starts from a random value between vel_min and vel_max for v and a random value between disp_min and disp_max for σ. As there are a number of random walks required to converge to a value, the first m walks should be discarded for the determination of the v and σ values, which is usually referred to as the burn-in length of a chain. burn corresponds to this m walks that will be discarded.
- thin (int) A random walk means that the value at a new position $i + 1$ will be correlated to the value at position i. The thin parameter therefore takes only every k^{th} value inside a chain, to reduce the correlation between the values inside a chain. When plotting a chain, the graphs depict a high degree of correlation, we recommend to put this to a high value. The disadvantage of a high-value is that the total sample size needs to increase to obtain a statistically sufficient sample size.
- agebins (string) Currently implemented and being tested, but not distributed.

3.9.1 *Choosing parameter-values*

The key to a succesful run of PyParadise and to reduce computational time is a careful look at the parameter-files. Choosing an initial template tmplinitspec which comes close to the final fitted average might lead to a value of iterations which is a bit lower and therefore makes the calculations faster. In case you believe that tmplinitspec does not differ strongly from the input spectrum, a value of 2 for iterations will likely suffice. That this value needs be higher than 3 or 4 should be rare.

The value for vel_guess is important as the template library gets shifted to this velocity and *afterwards* resampled to the instrumental dispersion. The derived values for σ is therefore an additional broadening applied to the instrumental dispersion. This value can therefore be set to 3000 km/s, instead of the true 4000 km/s without having significant effect on the fitting, but 10000 km/s would become more problematic. These numbers depend on the situation PyParadise is applied to, with more accurate values becoming more critical towards higher resolution spectroscopy and lower velocity dispersion.

For vel_min, vel_max, disp_min, and disp_max we recommend values which are not too far away from the real value as it may take longer to converge or the MCMC routine might try for out unrealistic values. We also do not recommend values that are too close to the real value, since PyParadise assumes that the distribution of v and σ in the final MCMC sample is normal distributed. For σ, the assumption that distribution of values is normal might break down in the case that the values come close to the minimum values. We therefore recommend in such a case to put disp_min to something like $v_{\mathrm{step}}/2 - \sigma_\sigma$, with v_{step} the sampling of the spectrum in velocity space and σ_σ the uncertainty of the velocity dispersion. Insert figure to illustrate this.

In case you are using kinematical information from a third-source, or just simply re-running PyParadise without the computationally expensive kinematic determination, you can fix the kinematics by setting kin_fix to 1. We strongly recommend this setup in case iterations is set to 0.

For the MCMC, we recommend a test run where you save the full chain to the disk (or a plot, needs implementation first though). An example is illustrated in Figure .... From this test run, one can determine the proper size of the burn-in length should be around 50/70 as the walk is still in the process of converging. At the moment, we do not have clear cases where thinning a chain would be beneficial. Some chains do show correlations between the $i^{\rm th}$ and the $(i+1)^{\rm th}$ steps of the chain, but the combination of multiple chains which run longer than the $\sim$ 10 times the largest autocorrelation value should provide enough values. We therefore recommend letting the value of `thin` to be set to 1 or 2. The autocorrelation value (not outputted yet by PYPARADISE), should be multiplied by at least 10 to obtain accurate uncertainties on v and σ.

4 EMISSION-LINE MODELLING

PYPARADISE can also fit emission lines after the continuum subtraction has been performed. Hence, emission lines are not fitted simultaneously with the stellar continuum, which has certain advantages and disadvantges. Below we describe the adopted algorithm, available line shapes and how to setup the configuration files for running the emission-line modelling.

4.1 Algorithm

Fitting emission lines is a non-linear process so we employ standard χ^2 minimization algorithms to find the best-fit parameter set for a given model. In a spectrum of a galaxiy usually several emission lines of interest are present so that a typical model is usually of a superposition of several lines across the spectrum. In PYPARADISE we offer two different but widely adopted non-linear minimization algorithms. The user can chose between a Downhill-Simplex algorithm or the Levenberg-Marqudart algorithms.
The issue with fast χ^2 algorithms is that they require a good starting point for exploring the χ^2 surface to find the global minimum and not being trapped in a local minimum. Therefore, reasonable starting values are necessary are crucial for successful non-linear emission-line modelling. PYPARADISE is optimized for automated analysis of a large number of spectra, with a similar redshift, so that an individual tuning of starting values by hand become infeasible. Our strategy is that users only need to make a good guess for the redshift in all spectra and set a pixel window size so that the local spectral shape and integrated flux of each redshifted line can be empirically evaluated to obtain starting values. This procedure works best for isolated lines, but usually provides reasonable starting values for blended lines in most cases to allow a good fit. In very complex cases, however, this approach may not be sufficient.

4.2 Line profiles

Only simple **Gaussian** line profiles are available in the current version of PYPARADISE. It is planed to expand this further to Gauss-Hermite expansions of lines to take asymmetric line shapes into account. If asymmetric lines are present in the data, currently two separate Gaussian profiles need to be used to capture the asymmetry of lines implicitly.

4.2.1 Gaussian line

A Gaussian line is well described by a central wavelength, a line disperion, and an integrated line flux. In PYPARADISE the central wavelength is computed from a provided *rest-frame wavelength* for a specific line together with a *redshift expressed in velocity space as* $v = c \times z$ *in units of km/s*. Also the line disperion is expressed by an *intrinsic velocity dispersion* from which the line dispersion in unit wavelength at a given redshift is computed and enlarged in quadrature by the provided instrumental spectral resolution. Hence, four parameters (of which only three are free parameters to be optimized) need to be provided for each Gaussian component in the model.
A key feature of PYPARADISE is that coupling redshift, line dispersion and fixing certain line ratios is easy. The user can simply indicate which lines should have the same redshift or intrinsic velocity diserpsion which can significantly reduce the number of free parameters and makes the optimization process more robust. Fixing line ratios is meant for known multiplets of lines with a-prior priors on their ratios.

4.3 Configuration files for emission line fitting

Using the emission line modelling in PYPARADISE requires to prepare three configuration files as described below. In most cases they can be re-used for different spectra and purposes.

4.3.1 Model file

The model file is a ASCII file that defines a list of different line components to be modelled. There is no limit on the number of components. Each component consistents of a block of several lines depending on the specific component as described below. **A typical parameter file looks like this**:

```
Gauss: OIII5007
restwave 5006.84
flux 50.0 1
```

```
vel 10350.0 1
disp 100.0 1

Gauss: OIII4960
restwave 4958.9
flux OIII5007:0.33
vel OIII5007
disp OIII5007
```

The structure for each component is the same. The first line of each component always starts with a line TYPE: NAME, where TYPE is a component type, i.e. Gauss for Gaussian, and NAME is a unique identifier WITHOUT spaces for the emission line to be fitted. The second line for a Gaussian is the fixed rest-frame wavelength in the same wavelength units as the spectrum to be fitted. What follows are the settings for the three free parameters of a Gaussian to be fitted. The from for each line is PAR VALUE FLAG, where PAR is either flux, vel or disp to discribe the flux, centroid or disperion of the line. Those are starting values for the fitting which may be updated automatically by PyParadise. Flux are in the same units as the spectra multiplied by the wavelength unit, and vel and disp are in units of km/s as described above for the Gaussian component. The FLAG can be set to 1 or 0. In case of 1, the parameter is free to vary and will be fitted, and in case of 0 the parameter will be fixed to the provided value.
As shown for the second component some parameters of Gaussian can be coupled to other lines in various ways. The syntax for coupling is PAR NAME, where PAR is the parameter to be coupled and NAME is Gaussian component to which this parameter will be tight together. No FLAG parameter is needed in this case as this will be inherreted from the FLAG parameter of the component it is thight with. For the flux parameter it is additionally allowed to add a fixed multiplicative factor for the coupling which is important for known multiplets with fixed ratios. The syntax in this case is then `flux NAME:FACTOR`, where NAME is again the component the flux value will be replaced and FACTOR is a multiplicative factor for the coupled flux.
Not all the parameters of a Gaussian need to be coupled and a mixture of free, fixed and coupled parameters are all allowed for a component. However, an important issue is that the component with the primary parameters always need to be listed before the coupled parameters are defined as in the example case above. Since free parameters are usually updated automatically based on the empirical data, it is recommended that the brightest line is set with free parameter in case coupling between several lines is desired. This will provide the best guess value for coupled parameters.

4.3.2 *Wavelength range file*

Usually the emission lines are not spread over the full wavelength range of the input spectra. Computation speed and robustness is gained if the modelling of the emission lines is restricted to the wavelength range the emission line actually contribute signnificantly to. Hence, it is highly recommended to provide a file that defines certain wavelength intervals for which the model is actually evluated and a χ^2 value is computed. The syntax of this file is the same as for the wavelength masking for the stellar continuum fitting, but only wavelength ranges defined in the rest-frame will be used. For the simple case of fitting the [OIII] double as defined in the Model file example above the wavelength range file would simply look like:

```
[rest-frame] 4930 5030
```

This means that only the wavelength range from 4930 to 5030 (in Angstrom) would be fitted and the rest of the spectrum would be ignored. Of course, the unit of the wavelength ranges needs to match with that of the spectrum and adjusted as necessary.

4.3.3 *Parameter file*

All the setups for the emission line fitting will be combined in a parameter file which is provided to PyParadise at the start-up call. A typical example file looks like this:

```
eCompFile modelfile.txt !name of the line parameter file, None if no emission lines to be fitted
vel_guess 10300.0 !rough velocity guess for the object in km/s as cz (float)
line_fit_region lines.fit !Wavelength regions considered during the fitting (string)
efit_method leastsq !method for line parameter fitting (leastsq/simplex)
efit_ftol 1e-8 !ftol convergence parameter
efit_xtol 1e-8 !xtol convergence parameter
eguess_window 20 !size of the spectral window in pixel to estimate initial guesses for each line
min_x 1 !Minimum x dimension
```

```
max_x 500 !Maximum x dimension
min_y 1 !Minimum y dimension
max_y 500 !Maximum y dimension
```

Each line follows the usual convention for PyParadise parameter files, i.e. PAR VALUE !Description. The detailed description of the necessary parameters in this files are
`eCompFile` (`string`) Full path to the model file which is in ASCII format as described in Sect. 4.3.1 above.
`vel_guess` (`float`) Best guess of mean redshift z of all spectra to model expressed in velocity (km/s) via $c \times z$.
`line_fit_region` (`string`) Full path to the wavelength range file in ASCII format as described in Sect. 4.3.2 above.
`efit_method` (`string`) Method to find the minimal χ^2 value. Only two options available now, either leastsq or simplex for the Levenberg-Marquardt or Downhill-simplex method, respectively.
`efit_ftol` (`float`) and `efit_ftol` (`float`) are the relative errors desired for the χ^2 and the in the approximate solution, respectively, which set stopping limits for the χ^2 minimization alrogithms. Do not change numbers if unsure.
`eguess_window` (`integer`) Based on the mean redshift defined by `vel_guess` and rest-frame wavelength of all lines defined in the model file, empirial measurements on line shape such as integrated flux, line centroid and line shape will be performe from all pixels within the given wavelength window in pixels. If this number is set to 0, no empirical estimates for the starting values will be done and the values in the model file will be used instead (CHECK IF TRUE!?).
`min_x` (`integer`) till `max_y` (`integer`) define limits for spectra to be modelled in spatial domain. Indexing starts with 1. The syntax is the same as for the continuum modelling as described in Sect. **??**.

5 BOOTSTRAP ERRORS

5.1 The importance of error estimation

The SSP fitting only provides the mean values for the SSP parameters and no errors. Bootstrap provides a technique to obtain the errors on the SSP parameters. As we already obtained the errors on the kinematics when we fit the spectra for the first time, the kinematics stay constant and the bootstrapping is therefore performed relatively fast. The BOOTSTRAPS command-line argument gives the number of bootstraps that should be performed. The bootstrapping is only performed together when the MODKEEP command-line argument is provided too. During the bootstrapping a random number of spectra is discarded. The command-line argument MODKEEP gives the percentage of spectra that will be kept during each bootstrapping step. It is recommended to keep this value relatively high, e.g. 80%.

6 DATA FORMATS

Like any software package PyParadise also requires that data is provided in certain pre-defined formats to run successfully. Failure of the program may often be related to invalid input. Therefore, users are strongly encourage to read this section carefully before using this software for their analysis.

6.1 Spectral data

PyParadise can be feed with three basic types of spectral data. These are 1) a single spectrum, 2) row-stacked spectra (hereafter RSS), and 3) 3D data cubes. The minimum information they all contain are the wavelength grid and the corresponding flux densities that makes up the spectrum. Optimal but often important information are the associated errors and known bad pixels. All three types have different technical specifications to store those data. They are automatically identified by PyParadise if the format is consistent with the specifications.

6.1.1 General structure of spectral FITS files

There is no standard for storing spectral information in FITS files. The format used by PyParadise is described in the section, and we note to the users of the Fortran code of PARADISE that the format is slightly different. In this part, the general description of FITS files is given and the variation due to the different formats are given in §6.1.4 for FITS files containing single spectra, in §6.1.2 for RSS files and in §6.1.3 for cubes. §6.1.4 also contains information regarding the structure of a spectrum which is stored in FITS table format or ASCII format.
A FITS file consists of several HDUs, Header Data Units. The spectra (`DATA`), the errors (`ERROR`), and the pixel mask (`BADPIX`) are stored in different HDUs.
The `DATA` HDU should be the first HDU and include a few header values regarding the wavelength information. Any other information is not ignored (`DC-FLAG` still needs to be made consistent). In the header of `ERROR`, the header keyword `EXTNAME` should be set to `ERROR`. Similarly, in the header of `BADPIX`, the header keyword `EXTNAME` should be set to `BADPIX`. No other header information will be read in from `ERROR` and `BADPIX`.

6.1.2 RSS files

A RSS file is the primary output of a multi-fiber spectrograph, either a MOS or IFU system. It is a simple stack of spectra that share a **common** wavelength grid. The stored data should have the structure of $m \times n$ pixels, where m is the number of wavelength elements and n the number of spectra. The wavelength information of RSS files is given by the header keywords `CRVAL1`, first wavelength element, and `CDELT1`, the wavelength difference between the i^{th} and $(i+1)^{\mathrm{th}}$) pixel, in units of Ångstrom per pixel. As any standard and valid FITS file, it should contain the header keyword `NAXIS1`, which equals m, and it should contain the header keyword `NAXIS2`, which is equal to n.

6.1.3 Data cubes

A cube is the primary output of an integral field unit (IFU), and can also be the end-products of radio telescope observations. It is a combination of an image, with at every spatial position (referred as spaxel) a spectrum. The stored data should have the structure of $x \times y \times z$ pixels, where x and y span the spatial space and z is the number of wavelength elements. The wavelength information of a cube is given by the header keywords `CRVAL3`, first wavelength element, and `CDELT3`, the wavelength difference between the i^{th} and $(i+1)^{\mathrm{th}}$) pixel, in units of Ångstrom per pixel. As any standard and valid FITS file, it should contain the header keyword `NAXIS3`, which equals z, and it should contain the header keyword `NAXIS1` and `NAXIS2`, which equal to x and y respectively.

6.1.4 Single Spectrum

There are various possible format to store a single spectrum. To facilitate the usage of PyParadise we implement a few common formats, the SDSS-DR7-like fits image format, the SDSS-DR10-like table format, and an ascii table format. They are widely distributed in the community. Since PyParadise does not require all the informations that are stored in SDSS spectra, a user can easily create a confrom spectrum that can be used with PyParadise only with the minimum content as described below.

6.2 Template spectra

The template spectra are a collection of spectra that should ideally be sufficient to describe the spectrum to be modelled as a linear superposition modolu the kinematic convolution kernel. There for the data format is similar to the RSS case for the input data. The big difference is that these data are usually theoretical or semi-empirical derived spectra. Thus, we assume them to be noise free and do not contain masked pixel. In addition, each template spectrum has associated additional information like stellar population age, luminosity, and metallicity that needs to be captures.

7 RUNNING PYPARADISE

To run PyParadise, one needs besides the input spectrum `<input_file>` a few configuration files. For the stellar populations one parameter file `<ssp_par.txt>` and two mask files are needed, and optionally a file describing the binning of stellar population parameters in different age ranges. To run PyParadise for extracting the stellar populations, execute the following command:

```
ParadiseApp.py <input_file> <prefix> <spec_res> --SSP_par <ssp_par.txt>
```

where `<prefix>` is the prefix that will be placed before the names of the output files, and `<spec_res>` is the spectral resolution of `<input_file>` in FWHM.
The emission line fitting requires the stellar population fitting to be performed first, after which the determination of the emission lines can be performed. For the emission line fitting, there is a parameter file `<line_par.txt>`, a mask file, and a file containing the emission line properties. To run PyParadise for extracting the emission-line properties, execute the following command:

```
ParadiseApp.py <input_file> <prefix> <spec_res> --line_par <line_par.txt>
```

with `<prefix>` should be the same prefix as used for the SSP derivation.
To obtain errors for the stellar population parameters and the emission line parameters, another run of PyParadise is required, where m bootstrap runs will be performed. During each bootstrap run, a certain percentage p of the template library will be used to obtain the best-fitting stellar population parameters. If bootstrapping for emission lines is requested, the best-fitted model at each bootstrap run will be subtracted from `<input file>` and the emission lines will be fitted to this spectrum. The bootstrapping will thus provide m different values for the stellar population and emission line parameters, from which a bootstrapped mean value and a standard deviation are derived. To perform the bootstrapping for the stellar population properties only or for the stellar population and emission line properties, run the following command, respectively

```
ParadiseApp.py <input_file> <prefix> <spec_res> --SSP_par <ssp_par.txt> --bootstraps <m> --modkeep <
    p>
ParadiseApp.py <input_file> <prefix> <spec_res> --SSP_par <ssp_par.txt> --line_par <line_par.txt> --
    bootstraps <m> --modkeep <p>
```

We do not recommend interrupting PyParadise call by hitting Ctrl+C on Linux systems, as it might happen that the program will not be interrupted.

8 OUTPUT FILES

8.1 Stellar population results

There are three files produced as result of the stellar population fitting: `<prefix>.cont_model.fits` (hereafter abbreviated with `<cont model>`), `<prefix>.cont_res.fits` (hereafter abbreviated as `<cont res>`) and `<prefix>.stellar_table.fits` (hereafter abbreviated as `<stellar table>`). The file `<cont model>` contains the best-fitted stellar population model spectra. The file `<cont res>` contains the residual spectra, constructed by subtracting the model spectra from the input spectra.
The header information from the `<input file>` will be copied into the first HDUs of the output files `<cont model>` and `<cont res>`. A possible exception is the wavelength information and the size of the data cubes depending on the values of the `start_wave` and `end_wave` parameters. If the `start_wave` value is higher than the first wavelength element, then the first HDU will have a different value of CRVAL1 compared to the `<input file>`. In the case of single spectrum file or RSS, the value of NAXIS1 and in the case of cubes the value of NAXIS3 will change too if `start_wave` and `end_wave` are different from the limits in `<input file>`. A change in the header values is linked to a different size of the data in the output files.
In case the fitting of a spectrum failed, the files will contain zero as the model spectrum and therefore the original spectrum as residual. One can check if the fitting of a spectrum failed by checking if the spectral index is missing from `<stellar table>`.
The kinematic and stellar population parameters, together with the parameters describing the quality of the fit, are saved in the `<stellar table>`.

- `fiber` (`int`). This parameter is only written if the `<input file>` is a single spectrum or a RSS file. Values might be missing from this column in the case of a fiber where the fitting failed.
- `xcor`, `ycor` (`int`). These parameters are only written if the `<input file>` is a cube file. Values might be missing from this column in the case of a fiber where the fitting failed.
- `vel_fit`, `vel_fit_err` (`float`) The fitted value and the corresponding error for v in km/s.
- `Rvel` The Gelman-Rubin convergence value for v. For multiple runs of the MCMC chains, this value describes how close the derived v's are to each other. The closer this value is to 1, the better the convergence of multiple chains to the same value and the more confident one can be that derived value is correct.
- `disp_fit`, `disp_fit_err` (FLOAT) The fitted value and the corresponding error for σ in km/s.
- `Rdisp` Similar to `Rval`, but for σ.
- `chi2` (`float`). The χ^2 value of the fit. Use this value to evaluate the quality of the fit.
- `dof` (INT) Not implemented yet, but should be, as it makes diagnostic much easier.
- `base_coeff` (`array of floats`) This array gives the coefficients of the template library, and the size of the array equals the number of template spectra. One can reproduce the model spectrum in `<cont model>`, by multiplying the `base_coeff` with the template spectra, and correcting for the normalisation effects and `lum_coeff_frac_total`.
- `lum_coeff_frac_total` (`float`). When fitting a normalised spectrum, in the ideal case the sum of `base_coeff` value will equal 1. Due to noise and template mismatches, this value in normal cases is close to 1.
- `lum_<ssp_prop>_total` (`float`). The luminosity-weighted stellar population average of a stellar population property, where `<ssp_prop >` can be `age`, `[Fe/H]`, `[A/Fe]`.
- `mass_coeff_frac_total` (`float`). When fitting a normalised spectrum, in the ideal case the sum of `base_coeff` value will equal 1. Due to noise and template mismatches, this value in normal cases is close to 1. The mass-weighting approach will make this value slightly different from `lum_coeff_frac_total`.
- `mass_<ssp_prop>_total` (`float`). The mass-weighted stellar population average of a stellar population property, where `<ssp_prop>` can be `age`, `[Fe/H]`, `[A/Fe]`. To obtain the mass-weighted properties, a conversion is performed based on the `mass-to-light`-values inside the template library.
- More items will be added here, once the agebins file implementation is completed.

A number of additional parameters are saved in the file `<prefix>.stellar_table.fits` during the bootstrapping procedure. The columns with the appendix `*_err` contain the standard deviation of the corresponding value and the columns with the appendix `*_btmean` contain the arithmetic mean of the corresponding value.

- `lum_coeff_frac_total_err` (`float`).
- `lum_coeff_frac_total_btmean` (`float`).
- `lum_<ssp_prop>_total_err` (`float`).
- `lum_<ssp_prop>_total_btmean` (`float`).
- `mass_coeff_frac_total_err` (`float`).

- mass_coeff_frac_total_btmean (float).
- mass_<ssp_prop>_total_err (float).
- mass_<ssp_prop>_total_btmean (float).

8.2 Emission-line results

The emission-line fitting produces three files in a similar structure and style as the SSP fitting: <prefix>.eline_model.fits (hereafter referred as <eline_model>), <prefix>.eline_res.fits (hereafter referred as <eline_res>) and <prefix>.eline_table.fits (hereafter referred as <eline_table>). The <eline model> contains the emission-line fits, without the stellar populations, and should therefore be compared to <cont res>. Subtracting <eline model> from <cont res> produces the file <eline res>. The structure of the files <eline model> and <eline res> is the same as <cont model> and <cont res>.
The parameters of the emission lines are stored in the file <eline table>. The data is located as a FITS table in the second HDU, with the column structure as follows:

- fiber, x_cor, y_cor (int). Described in §8.1
- <line>_flux (float). The integrated flux of the emission line. The unit of this quantity depends on the unit of the original input file.
- <line>_vel (float). The velocity of the emission line.
- <line>_fwhm (float). The FWHM of the emission line.
  The last three columns are repeated for each emission line specified in the file.
  The following columns are only produced after the bootstrap run:
- <line>_flux_err (float). The error of the integrated flux of the emission line.
- <line>_vel_err (float). The error of the velocity of the emission line.
- <line>_fwhm_err (float). The error of the FWHM of the emission line.
  The last three columns are repeated for each emission line specified in the file .

9 TROUBLESHOOTING

9.1 Distinguishing bad from good fit results

A good starting point for determining the quality of the fit is to check whether the chi2 value is close to dof. This comparison is only reasonable to perform when the input data includes (realistic) errors. As usual in statistics, a ratio of chi2 / dof which is much higher than 1 is a bad fit but a value close to 1 does not imply that the fitted values are close to reality.
There can be several reasons for bad chi2 values, and plotting the input spectrum versus the fitted result can provide a good impression what might be going wrong.

- Incorrect positioning of the mask for masking out features for the determination of the continuum. An effect of what might happen is depicted in Figure todo. This can be easily corrected by adjusting the file excl.cont. In our example, we will need to adjust . . . to . . . .
- Besides features in the continuum mask, there might be an incorrect positioning with respect to the moment when the spectrum is fitted. This is illustrated in Figure todo. This can be easily corrected by adjusting the file excl.fit. In our example, we will need to adjust . . . to . . . .
- Incorrect determination of the continuum at the edges.
- Very young populations, which are missing from the template library.

9.1.1 Examining the stellar kinematic information

When either v or σ are located close the minimum and/or maximum values provided in the stellar parameters file, the limits can be erroneous and/or something else might be going wrong. If both the errors (vel_fit_err or disp_fit_err) and the Gelman-Rubin parameter (Rvel or Rdisp) are low, the issue will likely not be solved by altering the parameters sample, burn, and thin. This is because multiple random walks converged to the same value and within the chain there is not a lot of variation, indicated by the small errors.
When the errors are small but the values Rvel and/or Rdisp differ significantly from 1 (usually should be below 1.1 or 1.2), it is an indication that one or multiple chains converge to a different value for v and/or σ. There are two methods that might remedy this. One method involves making the limits vel_min, vel_max and/or disp_min, disp_max tighter to reduce the parameter space. Another method is to increase the values for length of the chain, which has the unfortunate side effect that the computation will take longer and that more memory will be allocated.

10 FREQUENTLY ASKED QUESTIONS

My velocity dispersion value is too high? If the fitted σ is too high, check the initial guess for the template spectrum. In case your initial guess is a relatively old model, but the expectation is a young stellar population, try adjusting the initial guess to a young stellar population. If this does not help immediately, try adjusting the the limits on `disp_min` and `disp_max`

PyParadise *crashed during a run, what do I do now?* First of all, if the error message that PyParadise prints to the screen contains both `multiprocessing/pool.py` and `raise self._value`, rerun PyParadise with additional command line argument `--parallel 1` and also consider adding `--verbose`. The error message indicates that something went wrong when PyParadise entered the part of the code which can run in parallel. Unfortunately due to the way that Python propagates the error message, it removes the indication to where actually the program ran into trouble and it is therefore not easy to understand where the code breaks.

My emission lines are badly fitted. What went wrong? First, check if the stellar continuum fit is ok. If this went wrong there are problem with the emission line fit, too. If this is ok, check if all the fitted emission lines defined in the file `par_lines` are in the fitted ranges defined in the file `lines.fit`. If the lines are not in the fitted range, the initial guesses for the lines are not adjusted and may therefore not fit the real lines.

The fitted stellar continuum equals 0. Why is that? Most likely there is a problem with the input data. First, check if the complete spectrum has a badpixel flag (in the second extension of your input data). This problem could also be caused by ”NaNs” in the spectrum. Another problem could be zeros (or very small numbers) in the error spectrum (first extension of the input data). Then the χ^2 value is not defined and no stellar continuum can be fitted.

11 CONCLUSIONS

ACKNOWLEDGEMENTS