Mon. Not. R. Astron. Soc. **000**, 000–000 (0000) Printed 2 March 2018 (MN LaTeX style file v2.2)

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*

2 March 2018

ABSTRACT

1 INTRODUCTION

There are a number of codes available for deriving stellar population properties from full-spectral fitting and/or emission lines, each with various advantages and disadvantages. In general, there are three different approaches to obtain derive the star-formation histories from stellar populations: codes that parametrise the star-formation histories, codes that regularise the star-formation histories and codes that do neither. For a full discussion of different codes and analysis, we refer to Walcher2011. List of codes: STARLIGHT, STECKMAP, ULYSS, PPXF, FIT3D.

STECMAP is developed by Ocvirk2006a to obtain the star-formation histories with fixed kinematics. There is an extension named STECKMAP Ocvirk2006b, which can also fit the kinematics. The codes use a penalised maximum-likelihood analysis to smooth the star-formation histories and applies Monte-Carlo simulations to derive uncertainties.

PPXF is a code developed by Cappellari2004 and is a powerful for obtaining the kinematics, but can also be used to obtain the star-formation histories and emission-line features. It penalises the kinematic derivation and regularises the star-formation histories by smoothing it as long as the fit quality stays constant. PPXF was originally developed for the ATLAS^{3D} project on early-type galaxies, which are believed to have steadily varying star-formation histories.

STARLIGHT is a Fortran 77 code written and uses Markov Chain Monte Carlo (MCMC) methods to derive weights, extinction, and kinematics from a spectrum. Of all the codes STARLIGHT's approach is the most comparable to PYPARADISE, with the exceptions that it simultaneously fits the stellar populations and the kinematics, and that there is not option to fit emission lines.

FIT3D is code that is based on the former lookup-reference

Probable strong advantage and disadvantage, high degree of freedom to the user, usable as a Python module and can therefore be integrated in other projects, decoupled fitting of kinematics and stellar populations, decoupled fitting of stellar populations and emission lines. Example with library usage needed, possibly with IPython Notebooks.

PARADISE, on which PYPARADISE is based on, is a code written in Fortran by C. J. Walcher, does not apply parametrisation or penalisation of the star-formation histories. It instead relies upon bootstrapping to obtain realistic uncertainties on the star-formation histories.

Unlike GANDALF, the SSP fitting and the emission line fitting, are two separate procedures. This is possible because regions with emission lines are masked out from the stellar population fitting.

Throughout the paper we assume a cosmological model with $H_0 = 70\,\mathrm{km\,s^{-1}\,Mpc^{-1}}$, $\Omega_\mathrm{m} = 0.3$, and $\Omega_\Lambda = 0.7$.

2 INSTALLATION

PYPARADISE depends on the following software:

- Python version 2.7 or 3.x. The code might work on earlier Python versions by supplying the argument `--parallel 1` to every call of `ParadiseApp.py`, although this is only tested with Python 2.6. Python 2.7 or higher is required in order for multi-threading to function properly.
- numpy, tested with versions …
- scipy, tested with versions …
- Either pyfits, tested with versions …, or astropy, tested with versions …. In case you are in doubt which package to install, it is recommended to install astropy as the development of pyfits ceased except for possible bug fixes.

- PyMC 2.x, tested with versions . . . . At the moment of writing, PyMC 3 is still in beta status and is a big overhaul with respect to PyMC 2.x. If an official release of PyMC 3 comes out, we recommend using PyMC 2.x, until further notice.
- emcee, tested with versions . . . Can be used instead of PyMC.

In order to install PYPARADISE, you can obtain the code from BitBucket, and install by running the following command from the main directory

```
python setup.py install
```

If you get permissions errors, please consider running the command with superuser-permissions, inside a virtual environment or to change the command to either of the following commands

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

with the first option being the preferred option and the second option requires a correct setup of the `PYTHONPATH` environment variable.

Note that the installation procedure **will not** copy the template files to a new location, but will stay in the `templates` directory. Feel free to move this files to any location, as the location of the template files need to be specified in the parameter-files.

3 QUICK START

Copy the example template files from the test directory and adjust the path to the template directory accordingly. Update the `vel_guess` parameter to a velocity close to the final velocity. Adjust the minimum and maximum values for the allowed ranges for the velocity v and velocity dispersion σ. Have a look if the `start_wave` and `end_wave` contains the wavelength range where you want to perform the fitting. Similarly, make sure that `min_x`, `max_x` and `min_y`, `max_y` cut up your cube/rss file correctly. You can now run the stellar population fitting by running

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

which will produce the following three output files `<prefix>.cont_model.fits`, `<prefix>.cont_res.fits` and `<prefix>.stellar_table .fits`.

4 STELLAR POPULATION MODELLING

4.1 Algorithm

The kinematics and the stellar populations determination are computationally two different steps and use different methods. PYPARADISE iterates through the fitting of the kinematics and stellar populations, e.g. the kinematics of an input spectrum is determined from a certain template spectrum. These kinematics are then applied to the template library and the best-fitting combination of template spectra are determined.

4.1.1 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.

4.1.2 Stellar populations

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.

4.2 Template handling

4.3 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.

4.4 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_min`, `vel_max`, `disp_min`, `disp_max` (`float`). The `vel_min` and `vel_max` define the lower and the upper boundary between which the MCMC calculations will try to converge to the best-fitting v. The `disp_min` and `disp_max` values give the boundaries for σ.
- `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 $k^{\rm th}$ 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.
- `Ael` (`string`) Not implemented yet.
- `efflam` (`string`) Not implemented yet.
- `Rv` (`string`) Not implemented yet.

4.4.1 *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 $v_{\rm step}/2 - \sigma_\sigma$, with $v_{\rm step}$ 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 ~ 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.

5 EMISSION-LINE MODELLING

5.1 Algorithm

5.2 Line profiles

5.3 Configuration file

PyParadise can also fit emission lines. This is performed separately with respect to the stellar absorption fitting. From the observed spectrum, the model stellar absorption spectrum is subtracted. The emission lines are fitted to this stellar residual spectrum. One needs to provide estimates for the flux, velocity, and dispersion of every emission line. As this can be cumbersome, one can connect, e.g. the velocity of an emission line to the velocity of another emission line, and set the flux up such that it is at 1/3 the flux level of the first line. Note that the connected parameters of the line are not fitted simultaneously, e.g. the velocity of the first and second emission line are not fitted simultaneously. We therefore recommend letting the parameters of the strongest emission line be free, and that the parameters of weaker emission lines can be tied to the strong emission line.

6 BOOTSTRAP ERRORS

6.1 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%.

7 DATA INPUT 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.

7.1 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.

7.1.1 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 §7.1.4 for FITS files containing single spectra, in §7.1.2 for RSS files and in §7.1.3 for cubes. §7.1.4 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`.

7.1.2 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.

7.1.3 *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.

7.1.4 *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.

7.2 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.

8 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.

9 OUTPUT FILES

9.1 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).

9.2 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 §9.1
- `<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 .

10 USING PYPARADISE AS A PYTHON MODULE

PyParadise can be imported and used as a python module. In the case you simply want to fit a spectrum inside a data cube, print the kinematics and check the model parameters for convergence and χ^2 and plot the spectrum and the best-fit, running the following should suffice. There is some work required in transforming the library to the same resolution, sampling and wavelength grid as the input data.

```
from Paradise import *
import matplotlib.pyplot as plt

vel_guess = 1400.0  # Guessing velocity is 1400 km/s
z = vel_guess / 300000.
norm_window = 100   # Normalization window of 100 pixels

# Load in a spectrum from a cube
cube = loadCube('spectra.fits')
spec = cube.getSpec(20, 25).normalizeSpec(norm_window)

# Load and prepare the stellar population library
# assumed is that the instrumental dispersion for the cube is 3.0 FWHM angstrom
lib = SSPLibrary('templates.fits')
lib = lib.matchInstFWHM(3.0, vel_guess)
lib = lib.subWaveLibrary(spec.getWave()[0], spec.getWave()[-1])
lib = lib.resampleWaveStepLinear(spec.getWaveStep(), z)
lib = lib.normalizeBase(norm_window).rebinLogarithmic()

# Fit the data with the model
model = spec.fit_Kin_Lib_simple(templates, nlib_guess=20, vel_min=1200,
                                vel_max=1600, vel_guess=1400, disp_min=50,
                                disp_max=150)
model_spec = model[6]

# Print velocity + error + conversion value which should be close to 1
print('Fitted velocity = {} +/-'.format(model[0], model[1]))
print('Gelman-Rubin parameter for velocity conversion = {}'.format(mode[2]))

# Print velocity dispersion + error + conversion value which should be close to 1
print('Fitted velocity dispersion = {} +/-'.format(model[3], model[4]))
print('The Gelman-Rubin parameter for velocity dispersion conversion = {}'.format(mode[5]))

print('The chi^2 value from the fitting of the stellar population = {}'.format(mode[-1]))

plt.plot(spec.getWave(), spec.getData())                # Plot the data
plt.plot(model_spec.getWave(), model_spec.getData())    # Plot the model
```

11 GUIDELINES

11.1

11.2 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.

11.2.1 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.

12 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.

13 CONCLUSIONS

ACKNOWLEDGEMENTS