The pPXF Package v7.0 (10/JAN/2020)

Full Spectrum Fitting of Galactic and Stellar Spectra

This `pPXF` package contains a Python implementation of the Penalized PiXel-Fitting (`pPXF`) method to fit the stellar and gas kinematics, as well as the stellar population of galaxies. The method was originally described in Cappellari & Emsellem (2004) and was substantially upgraded in subsequent years and particularly in Cappellari (2017).

Attribution

If you use this software for your research, please cite at least Cappellari (2017). The BibTeX entry for the paper is:

```
@ARTICLE{Cappellari2017,
    author = {{Cappellari}, M.},
    title = "{Improving the full spectrum fitting method:
        accurate convolution with Gauss-Hermite functions}",
    journal = {MNRAS},
    eprint = {1607.08538},
    year = 2017,
    volume = 466,
    pages = {798-811},
    doi = {10.1093/mnras/stw3020}
}
```

Installation

install with:

```
pip install ppxf
```

Without write access to the global `site-packages` directory, use:

```
pip install --user ppxf
```

Usage Examples

To learn how to use the main program `pPXF` run the example programs in the `ppxf/examples` directory, within the main package installation folder inside `site-packages`, and read the

detailed documentation in the docstring in the file `ppxf.py`, on PyPi or as PDF from https://purl.org/cappellari/software.

pPXF Purpose

Extract galaxy stellar kinematics `(V, sigma, h3, h4, h5, h6,...)` or the stellar population and gas emission by fitting a template to an observed spectrum in pixel space, using the Penalized PiXel-Fitting (`pPXF`) method originally described in

Cappellari & Emsellem (2004)

and substantially upgraded in subsequent years and particularly in

Cappellari (2017)

The following key optional features are also available:

1) An optimal template, positive linear combination of different input templates, can be fitted together with the kinematics.
2) One can enforce smoothness on the template weights during the fit. This is useful to attach a physical meaning to the weights e.g. in terms of the star formation history of a galaxy.
3) One can fit multiple kinematic components for both the stars and the gas emission lines. Both the stellar and gas LOSVD can be penalized and can be described by a general Gauss-Hermite series.
4) Any parameter of the LOSVD (e.g. sigma) for any kinematic component can either be fitted, or held fixed to a given value, while other parameters are fitted. Alternatively, parameters can be constrained to lie within given limits or even tied by simple relations to other parameters.
5) One can enforce linear equality/inequality constraints on either the template weights or the kinematic parameters.
6) Additive and/or multiplicative polynomials can be included to adjust the continuum shape of the template to the observed spectrum.
7) Iterative sigma clipping can be used to clean the spectrum.
8) It is possible to fit a mirror-symmetric LOSVD to two spectra at the same time. This is useful for spectra taken at point-symmetric spatial positions with respect to the center of an equilibrium stellar system.
9) One can include sky spectra in the fit, to deal with cases where the sky dominates the observed spectrum and an accurate sky subtraction is critical.
10) One can derive an estimate of the reddening in the spectrum. This can be done independently for the stellar spectrum or the Balmer emission lines.
11) The covariance matrix can be input instead of the error spectrum, to account for correlated errors in the spectral pixels.
12) One can specify the weights fraction between two kinematics components, e.g. to model bulge and disk contributions.
13) One can use templates with higher resolution than the galaxy, to improve the accuracy of the LOSVD extraction at low dispersion.

Calling Sequence

```
from ppxf.ppxf import ppxf

pp = ppxf(self, templates, galaxy, noise, velscale, start,
          bias=None, bounds=None, clean=False, component=0,
          constr_templ={}, constr_kinem={}, degree=4, fixed=None,
          fraction=None, ftol=1e-4, gas_component=None, gas_names=None,
          gas_reddening=None, goodpixels=None, lam=None, linear=False,
          mask=None, method='capfit', mdegree=0, moments=2, plot=False,
          quiet=False, reddening=None, reddening_func=None,
          reg_dim=None, reg_ord=2, regul=0, sigma_diff=0, sky=None,
          templates_rfft=None, tied=None, trig=False,
          velscale_ratio=1, vsyst=0):

print(pp.sol)  # print best-fitting kinematics (V, sigma, h3, h4)
pp.plot()      # Plot best fit and gas lines
```

Input Parameters

templates: Vector containing the spectrum of a single template star or more commonly an array of dimensions `templates[nPixels, nTemplates]` containing different templates to be optimized during the fit of the kinematics. It has to be `nPixels >= galaxy.size`.

To apply linear regularization to the `weights` via the keyword `regul`, `templates` should be an array of two `templates[nPixels, nAge]`, three `templates[nPixels, nAge, nMetal]` or four `templates[nPixels, nAge, nMetal, nAlpha]` dimensions, depending on the number of population variables one wants to study. This can be useful to try to attach a physical meaning to the output `weights`, in term of the galaxy star formation history and chemical composition distribution. In that case the templates may represent single stellar population SSP models and should be arranged in sequence of increasing age, metallicity or alpha along the second, third or fourth dimension of the array respectively.

galaxy: Vector containing the spectrum of the galaxy to be measured. The star and the galaxy spectra have to be logarithmically rebinned but the continuum should *not* be subtracted. The rebinning may be performed with the `log_rebin` routine in `ppxf.ppxf_util`.

For high redshift galaxies, one should bring the spectra close to the restframe wavelength, before doing the `pPXF` fit. This can be done by dividing the observed wavelength by `(1 + z)`, where `z` is a rough estimate of the galaxy redshift, before the logarithmic rebinning. See Section 2.4 of Cappellari (2017) for details.

`galaxy` can also be an array of dimensions `galaxy[nGalPixels, 2]` containing two spectra to be fitted, at the same time, with a reflection-symmetric LOSVD. This is useful for spectra taken at point-symmetric spatial positions with respect to the center of an equilibrium stellar system. For a discussion of the usefulness of this two-sided fitting see e.g. Section 3.6 of Rix & White (1992).

IMPORTANT: (1) For the two-sided fitting the `vsyst` keyword has to be used. (2) Make sure the spectra are rescaled to be not too many order of magnitude different from unity, to avoid over or underflow problems in the calculation. E.g. units of `erg/(s cm^2 A)` may

cause problems!

noise: Vector containing the `1*sigma` error (per pixel) in the galaxy spectrum, or covariance matrix describing the correlated errors in the galaxy spectrum. Of course this vector/matrix must have the same units as the galaxy spectrum.

If `galaxy` is a `Nx2` array, `noise` has to be an array with the same dimensions.

When `noise` has dimensions `NxN` it is assumed to contain the covariance matrix with elements `sigma(i, j)`. When the errors in the spectrum are uncorrelated it is mathematically equivalent to input in `pPXF` an error vector `noise=errvec` or a `NxN` diagonal matrix `noise=np.diag(errvec**2)` (note squared!).

IMPORTANT: the penalty term of the `pPXF` method is based on the *relative* change of the fit residuals. For this reason, the penalty will work as expected even if no reliable estimate of the `noise` is available (see Cappellari & Emsellem (2004) for details). If no reliable noise is available this keyword can just be set to:

```
noise = np.ones_like(galaxy)  # Same weight for all pixels
```

velscale: Velocity scale of the spectra in km/s per pixel. It has to be the same for both the galaxy and the template spectra. An exception is when the `velscale_ratio` keyword is used, in which case one can input `templates` with smaller `velscale` than `galaxy`.

`velscale` is *defined* in `pPXF` by `velscale = c*Delta[np.log(lambda)]`, which is approximately `velscale ~ c*Delta(lambda)/lambda`. See Section 2.3 of Cappellari (2017) for details.

start: Vector, or list/array of vectors `[start1, start2, ...]`, with the initial estimate for the LOSVD parameters.

When LOSVD parameters are not held fixed, each vector only needs to contain `start = [velStart, sigmaStart]` the initial guess for the velocity and the velocity dispersion in km/s. The starting values for h3-h6 (if they are fitted) are all set to zero by default. In other words, when `moments=4`:

```
start = [velStart, sigmaStart]
```

is interpreted as:

```
start = [velStart, sigmaStart, 0, 0]
```

When the LOSVD for some kinematic components is held fixed (see `fixed` keyword), all values for `[Vel, Sigma, h3, h4,...]` can be provided.

Unless a good initial guess is available, it is recommended to set the starting `sigma >= 3*velscale` in km/s (i.e. 3 pixels). In fact when the sigma is very low, and far from the true solution, the `chi^2` of the fit becomes weakly sensitive to small variations in sigma (see `pPXF` paper). In some instances, the near-constancy of `chi^2` may cause premature convergence of the optimization.

In the case of two-sided fitting a good starting value for the velocity is `velStart = 0.0` (in this case `vsyst` will generally be nonzero). Alternatively on should keep in mind that velStart refers to the first input galaxy spectrum, while the second will have velocity `-velStart`.

With multiple kinematic components `start` must be a list of starting values, one for each different component.

EXAMPLE: We want to fit two kinematic components. We fit 4 moments for the first component and 2 moments for the second one as follows:

```
component = [0, 0, ... 0, 1, 1, ... 1]
moments = [4, 2]
start = [[V1, sigma1], [V2, sigma2]]
```

Optional Keywords

bias: This parameter biases the `(h3, h4, ...)` measurements towards zero (Gaussian LOSVD) unless their inclusion significantly decreases the error in the fit. Set this to `bias=0` not to bias the fit: the solution (including `[V, sigma]`) will be noisier in that case. The default `bias` should provide acceptable results in most cases, but it would be safe to test it with Monte Carlo simulations. This keyword precisely corresponds to the parameter `lambda` in the Cappellari & Emsellem (2004) paper. Note that the penalty depends on the *relative* change of the fit residuals, so it is insensitive to proper scaling of the `noise` vector. A nonzero `bias` can be safely used even without a reliable `noise` spectrum, or with equal weighting for all pixels.

bounds: Lower and upper bounds for every kinematic parameter. This is an array, or list of arrays, with the same dimensions as `start`, except for the last one, which is two. In practice, for every element of `start` one needs to specify a pair of values `[lower, upper]`.

EXAMPLE: We want to fit two kinematic components, with 4 moments for the first component and 2 for the second (e.g. stars and gas). In this case:

```
moments = [4, 2]
start_stars = [V1, sigma1, 0, 0]
start_gas = [V2, sigma2]
start = [start_stars, start_gas]
```

then we can specify boundaries for each kinematic parameter as:

```
bounds_stars = [[V1_lo, V1_up], [sigma1_lo, sigma1_up],
                [-0.3, 0.3], [-0.3, 0.3]]
bounds_gas = [[V2_lo, V2_up], [sigma2_lo, sigma2_up]]
bounds = [bounds_stars, bounds_gas]
```

component: When fitting more than one kinematic component, this keyword should contain the component number of each input template. In principle every template can belong to a different kinematic component.

EXAMPLE: We want to fit the first 50 templates to component 0 and the last 10 templates to component 1. In this case:

```
component = [0]*50 + [1]*10
```

which, in Python syntax, is equivalent to:

```
component = [0, 0, ... 0, 1, 1, ... 1]
```

This keyword is especially useful when fitting both emissions (gas) and absorption (stars) templates simultaneously (see the example for `moments` keyword).

constr_templ: This is specified by the following dictionary, where `A_ineq` and `A_eq` are arrays (have `A.ndim = 2`), while `b_ineq` and `b_eq` are vectors (have `b.ndim = 1`). Either the `_eq` or the `_ineq` keys can be omitted if not needed:

```
constr_templ = {"A_ineq": A_ineq, "b_ineq": b_ineq, "A_eq": A_eq, "b_eq": b_eq}
```

It enforces linear constraints on the template weights during the fit. The resulting pPXF solution will satisfy the following linear matrix inequalities and/or equalities:

```
A_ineq @ pp.weights <= b_ineq
A_eq @ pp.weights == b_eq
```

Inequality can be used e.g. to constrain the fluxes of emission lines to lie within prescribed ranges. Equalities can be used e.g. to force the weights for different kinematic components to contain prescribed fractions of the total weights.

EXAMPLES: We are fitting a spectrum using four templates, the first two templates belong to one kinematic component and the rest to the other. This implies we have:

```
component=[0, 0, 1, 1]
```

then we can set the constraint that the sum of the weights of the first kinematic component is equal to `fraction` times the total as follows [cfr. equation 30 of Cappellari (2017)]:

```
A_eq = [[fraction - 1, fraction - 1, fraction, fraction]]
b_eq = [0]
constr_templ = {"A_eq": A_eq, "b_eq": b_eq}
```

An identical result can be obtained in this case using the `fraction` keyword, but `constr_templ` additionally allows for general linear constraints for multiple kinematic components.

We can constrain the ratio of the first two templates weights to lie in the interval `ratio_min <= w[0]/w[1] <= ratio_max` as follows:

```
A_ineq = [[-1, ratio_min, 0, 0],    # -w[0] + ratio_min*w[1] < 0
          [1, -ratio_max, 0, 0]]    # +w[0] - ratio_max*w[1] < 0
b_ineq = [0, 0]
constr_templ = {"A_ineq": A_ineq, "b_ineq": b_ineq}
```

constr_kinem: This is specified by the following dictionary, where `A_ineq` and `A_eq` are arrays (have `A.ndim = 2`), while `b_ineq` and `b_eq` are vectors (have `b.ndim = 1`). Either the `_eq` or the `_ineq` keys can be omitted if not needed:

```
constr_kinem = {"A_ineq": A_ineq, "b_ineq": b_ineq, "A_eq": A_eq, "b_eq": b_eq}
```

It enforces linear constraints on the template weights during the fit. The resulting pPXF solution will satisfy the following linear matrix inequalities and/or equalities:

```
params = np.ravel(pp.sol)  # Unravel for multiple components
A_ineq @ params <= b_ineq
A_eq @ params == b_eq
```

IMPORTANT: the starting guess `start` must satisfy the constraints, or in other words, it must lie in the feasible region.

Inequalities can be used e.g. to force one kinematic component to have larger velocity or dispersion than another one. This is useful e.g. when extracting two stellar kinematic components or when fitting both narrow and broad components of gas emission lines.

EXAMPLES: We want to fit two kinematic components, with two moments for both the first and second component. In this case:

```
moments = [2, 2]
start = [[V1, sigma1], [V2, sigma2]]
```

then we can set the constraint `sigma1 > 3*sigma2` as follows:

```
A_ineq = [[0, -1, 0, 3]]  # -sigma1 + 3*sigma2 < 0
b_ineq = [0]
constr_kinem = {"A_ineq": A_ineq, "b_ineq": b_ineq}
```

We can set the constraint `sigma1 > sigma2 + 2*velscale` as follows:

```
A_ineq = [[0, -1, 0, 1]]  # -sigma1 + sigma2 < -2*velscale
b_ineq = [-2]             # kinem. in pixels (-2 --> -2*velscale)!
constr_kinem =  {"A_ineq": A_ineq, "b_ineq": b_ineq}
```

We can set both the constraints `V1 > V2` and `sigma1 > sigma2 + 2*velscale` as follows:

```
A_ineq = [[-1, 0, 1, 0],   # -V1 + V2 < 0
          [0, -1, 0, 1]]   # -sigma1 + sigma2 < -2*velscale
b_ineq = [0, -2]           # kinem. in pixels (-2 --> -2*velscale)!
constr_kinem =  {"A_ineq": A_ineq, "b_ineq": b_ineq}
```

clean: Set this keyword to use the iterative sigma clipping method described in Section 2.1 of Cappellari et al. (2002). This is useful to remove from the fit unmasked bad pixels, residual gas emissions or cosmic rays.

IMPORTANT: This is recommended *only* if a reliable estimate of the `noise` spectrum is available. See also note below for `.chi2`.

degree: Degree of the *additive* Legendre polynomial used to correct the template continuum shape during the fit (default: 4). Set `degree=-1` not to include any additive polynomial.

fixed: Boolean vector set to `True` where a given kinematic parameter has to be held fixed with the value given in `start`. This is an array, or list, with the same dimensions as `start`.

EXAMPLE: We want to fit two kinematic components, with 4 moments for the first component and 2 for the second. In this case:

```
moments = [4, 2]
start = [[V1, sigma1, h3, h4], [V2, sigma2]]
```

then we can held fixed e.g. the sigma (only) of both components using:

```
fixed = [[0, 1, 0, 0], [0, 1]]
```

NOTE: Setting a negative `moments` for a kinematic component is entirely equivalent to setting `fixed = 1` for all parameters of the given kinematic component. In other words:

```
moments = [-4, 2]
```

is equivalent to:

```
moments = [4, 2]
fixed = [[1, 1, 1, 1], [0, 0]]
```

fraction: This keyword allows one to fix the ratio between the first two kinematic components. This is a scalar defined as follows:

```
fraction = np.sum(weights[component == 0])
         / np.sum(weights[component < 2])
```

This is useful e.g. to try to kinematically decompose bulge and disk.

The remaining kinematic components (`component > 1`) are left free, and this allows, for example, to still include gas emission line components. More general linear constraints, for multiple components at the same time, can be specified using the `constr_templ` keyword.

ftol: Fractional tolerance for stopping the non-linear minimization (default 1e-4).

gas_component: Boolean vector, of the same size as `component`, set to `True` where the given `component` describes a gas emission line. If given, `pPXF` provides the `pp.gas_flux` and `pp.gas_flux_error` in output.

EXAMPLE: In the common situation where `component = 0` are stellar templates and the rest are gas emission lines, one will set:

```
gas_component = component > 0
```

This keyword is also used to plot the gas lines with a different color.

gas_names: String array specifying the names of the emission lines (e.g. `gas_names=["Hbeta", "[OIII]",...]`, one per gas line. The length of this vector must match the number of nonzero elements in `gas_component`. This vector is only used by `pPXF` to print the line names on the console.

gas_reddening: Set this keyword to an initial estimate of the gas reddening `E(B-V) >= 0` to fit a positive gas reddening together with the kinematics and the templates. This reddening is applied only to the gas templates, namely to the templates with the corresponding element of `gas_component=True`. The fit assumes by default the extinction curve of Calzetti et al. (2000) but any other prescription can be passed via the `reddening_func` keyword.

goodpixels: Integer vector containing the indices of the good pixels in the `galaxy` spectrum (in increasing order). Only these spectral pixels are included in the fit.

IMPORTANT: in all likely situations this keyword *has* to be specified.

lam: When the keyword `reddening` or `gas_reddening` are used, the user has to pass in this keyword a vector with the same dimensions of `galaxy`, giving the restframe wavelength in Angstrom of every pixel in the input galaxy spectrum. If one uses my `log_rebin` routine to rebin the spectrum before the `pPXF` fit:

```
from ppxf.ppxf_util import log_rebin
specNew, logLam, velscale = log_rebin(lamRange, galaxy)
```

the wavelength can be obtained as `lam = np.exp(logLam)`.

When `lam` is given, the wavelength is shown in the best-fitting plot, instead of the pixels.

linear: Set to `True` to keep *all* nonlinear parameters fixed and *only* perform a linear fit for the templates and additive polynomials weights. The output solution is a copy of the input one and the errors are zero.

mask: Boolean vector of length `galaxy.size` specifying with `True` the pixels that should be included in the fit. This keyword is just an alternative way of specifying the `goodpixels`.

method: {'capfit', 'trf', 'dogbox', 'lm'}, optional. Algorithm to perform the non-linear minimization step. The default `'capfit'` is a novel trust-region implementation of the Levenberg-Marquardt method, extended to allow for general linear constraints in a rigorous manner, while also supporting tied or fixed variables. See documentation of `scipy.optimize.least_squares` for a description of the other methods.

mdegree: Degree of the *multiplicative* Legendre polynomial (with a mean of 1) used to correct the continuum shape during the fit (default: 0). The zero degree multiplicative polynomial is always included in the fit as it corresponds to the weights assigned to the templates. Note that the computation time is longer with multiplicative polynomials than with the same number of additive polynomials.

IMPORTANT: Multiplicative polynomials cannot be used when the `reddening` keyword is set, as they are degenerate with the reddening.

moments: Order of the Gauss-Hermite moments to fit. Set this keyword to 4 to fit `[h3, h4]` and to 6 to fit `[h3, h4, h5, h6]`. Note that in all cases the G-H moments are fitted (non-linearly) *together* with `[V, sigma]`.

If `moments=2` or `moments` is not set then only `[V, sigma]` are fitted.

If `moments` is negative then the kinematics of the given `component` are kept fixed to the input values. NOTE: Setting a negative `moments` for a kinematic component is entirely equivalent to setting `fixed = 1` for all parameters of the given kinematic component.

EXAMPLE: We want to keep fixed `component = 0`, which has a LOSVD described by `[V, sigma, h3, h4]` and is modelled with 100 spectral templates; At the same time, we fit `[V, sigma]` for `component = 1`, which is described by 5 templates (this situation may arise when fitting stellar templates with pre-determined stellar kinematics, while fitting the gas emission). We should give in input to `pPXF` the following parameters:

```
component = [0]*100 + [1]*5   # --> [0, 0, ... 0, 1, 1, 1, 1, 1]
moments = [-4, 2]
start = [[V, sigma, h3, h4], [V, sigma]]
```

velscale_ratio: Integer. Gives the integer `ratio >= 1` between the `velscale` of the `galaxy` and the `templates`. When this keyword is used, the templates are convolved by the LOSVD at their native resolution, and only subsequently are integrated over the pixels and fitted to `galaxy`. This keyword is generally unnecessary and mostly useful for testing.

Note that in realistic situations the uncertainty in the knowledge and variations of the intrinsic line-spread function become the limiting factor in recovering the LOSVD well below `velscale`.

plot: Set this keyword to plot the best fitting solution and the residuals at the end of the fit.

One can also call the class function `pp.plot()` after the call to `pp = ppxf(...)`.

quiet: Set this keyword to suppress verbose output of the best fitting parameters at the end of the fit.

reddening: Set this keyword to an initial estimate of the stellar reddening `E(B-V) >= 0` to fit a positive stellar reddening together with the kinematics and the templates. This reddening is applied only to the stellar templates, namely to the templates with the corresponding element of `gas_component=False`, or to all templates, if `gas_component` is not set. The fit assumes by default the extinction curve of Calzetti et al. (2000) but any other prescription can be passed via the `reddening_func` keyword.

IMPORTANT: The `mdegree` keyword cannot be used when `reddening` is set.

regul: If this keyword is nonzero, the program applies first or second order linear regularization to the `weights` during the `pPXF` fit. Regularization is done in one, two or three dimensions depending on whether the array of `templates` has two, three or four dimensions respectively. Large `regul` values correspond to smoother `weights` output. When this keyword is nonzero the solution will be a trade-off between the smoothness of `weights` and goodness of fit.

Section 3.5 of Cappellari (2017) gives a description of regularization.

When fitting multiple kinematic `component` the regularization is applied only to the first `component = 0`, while additional components are not regularized. This is useful when fitting stellar population together with gas emission lines. In that case, the SSP spectral templates must be given first and the gas emission templates are given last. In this situation, one has to use the `reg_dim` keyword (below), to give `pPXF` the dimensions of the population parameters (e.g. `nAge`, `nMetal`, `nAlpha`). A usage example is given in the file `ppxf_example_population_gas_sdss.py`.

The effect of the regularization scheme is the following:

- With `reg_ord=1` it enforces the numerical first derivatives between neighbouring weights (in the 1-dim case) to be equal to `w[j] - w[j+1] = 0` with an error `Delta = 1/regul`.
- With `reg_ord=2` it enforces the numerical second derivatives between neighboring weights (in the 1-dim case) to be equal to `w[j-1] - 2*w[j] + w[j+1] = 0` with an error `Delta = 1/regul`.

It may be helpful to define `regul = 1/Delta` and think of `Delta` as the regularization error.

IMPORTANT: `Delta` needs to be smaller but of the same order of magnitude of the typical `weights` to play an effect on the regularization. One quick way to achieve this is:

1. Divide the full `templates` array by a scalar in such a way that the typical template has a median of one:

   ```
   templates /= np.median(templates)
   ```

2. Do the same for the input galaxy spectrum:

   ```
   galaxy /= np.median(galaxy)
   ```

3. In this situation, a sensible guess for `Delta` will be a few percent (e.g. `0.01 --> regul=100`).

Alternatively, for a more rigorous definition of the parameter `regul`:

A. Perform an un-regularized fit (`regul=0`) and then rescale the input `noise` spectrum so that:

   `Chi^2/DOF = Chi^2/goodPixels.size = 1.`

   This is achieved by rescaling the input `noise` spectrum as:

   `noise = noise*sqrt(Chi**2/DOF) = noise*sqrt(pp.chi2);`

B. Increase `regul` and iteratively redo the `pPXF` fit until the `Chi^2` increases from the unregularized value `Chi^2 = goodPixels.size` by `DeltaChi^2 = sqrt(2*goodPixels.size)`.

The derived regularization corresponds to the maximum one still consistent with the observations and the derived star formation history will be the smoothest (minimum curvature or minimum variation) that is still consistent with the observations.

reg_dim: When using regularization with more than one kinematic component (using the `component` keyword), the regularization is only applied to the first one (`component=0`). This is useful to fit the stellar population and gas emission together.

In this situation, one has to use the `reg_dim` keyword, to give `pPXF` the dimensions of the population parameters (e.g. `nAge`, `nMetal`, `nAlpha`). One should creates the initial array of population templates like e.g. `templates[nPixels, nAge, nMetal, nAlpha]` and define:

```
reg_dim = templates.shape[1:]   # = [nAge, nMetal, nAlpha]
```

The array of stellar templates is then reshaped into a 2-dim array as:

```
templates = templates.reshape(templates.shape[0], -1)
```

and the gas emission templates are appended as extra columns at the end. An usage example is given in `ppxf_example_population_gas_sdss.py`.

When using regularization with a single component (the `component` keyword is not used, or contains identical values), the number of population templates along different dimensions (e.g. `nAge`, `nMetal`, `nAlpha`) is inferred from the dimensions of the `templates` array and this keyword is not necessary.

reg_ord: Order of the derivative that is minimized by the regularization. The following two rotationally-symmetric estimators are supported:

- `reg_ord=1`: minimizes the integral over the weights of the squared gradient:

  `Grad[w] @ Grad[w].`

- `reg_ord=2`: minimizes the integral over the weights of the squared curvature:

  `Laplacian[w]**2.`

sigma_diff: Quadratic difference in km/s defined as:

```
sigma_diff**2 = sigma_inst**2 - sigma_temp**2
```

between the instrumental dispersion of the galaxy spectrum and the instrumental dispersion of the template spectra.

This keyword is useful when the templates have higher resolution than the galaxy and they were not convolved to match the instrumental dispersion of the galaxy spectrum. In

this situation, the convolution is done by `pPXF` with increased accuracy, using an analytic Fourier Transform.

sky: vector containing the spectrum of the sky to be included in the fit, or array of dimensions `sky[nPixels, nSky]` containing different sky spectra to add to the model of the observed `galaxy` spectrum. The `sky` has to be log-rebinned as the `galaxy` spectrum and needs to have the same number of pixels.

The sky is generally subtracted from the data before the `pPXF` fit. However, for observations very heavily dominated by the sky spectrum, where a very accurate sky subtraction is critical, it may be useful *not* to subtract the sky from the spectrum, but to include it in the fit using this keyword.

templates_rfft: When calling `pPXF` many times with an identical set of templates, one can use this keyword to pass the real FFT of the templates, computed in a previous `pPXF` call, stored in the `pp.templates_rfft` attribute. This keyword mainly exists to show that there is no need for it...

IMPORTANT: Use this keyword only if you understand what you are doing!

tied: A list of string expressions. Each expression "ties" the parameter to other free or fixed parameters. Any expression involving constants and the parameter array `p[j]` are permitted. Since they are totally constrained, tied parameters are considered to be fixed; no errors are computed for them.

This is an array, or list of arrays, with the same dimensions as `start`. In practice, for every element of `start` one needs to specify either an empty string `''` implying that the parameter is free, or a string expression involving some of the variables `p[j]`, where `j` represents the index of the flattened list of kinematic parameters.

EXAMPLE: We want to fit three kinematic components, with 4 moments for the first component and 2 moments for the second and third (e.g. stars and two gas components). In this case:

```
moments = [4, 2, 2]
start = [[V1, sigma1, 0, 0], [V2, sigma2], [V3, sigma3]]
```

then we can force the equality constraint `V2 = V3` as follows:

```
tied = [['', '', '', ''], ['', ''], ['p[4]', '']]
```

or we can force the equality constraint `sigma2 = sigma3` as follows:

```
tied = [['', '', '', ''], ['', ''], ['', 'p[5]']]
```

NOTE: One could in principle use the `tied` keyword to completely tie the LOSVD of two kinematic components. However, this same effect is more efficient achieved by assigning them to the same kinematic component using the `component` keyword.

trig: Set `trig=True` to use trigonometric series as an alternative to Legendre polynomials, for both the additive and multiplicative polynomials. When `trig=True` the fitted series below has `N = degree/2` or `N = mdegree/2`:

```
poly = A_0 + sum_{n=1}^{N} [A_n*cos(n*th) + B_n*sin(n*th)]
```

IMPORTANT: The trigonometric series has periodic boundary conditions. This is sometimes a desirable property, but this expansion is not as flexible as the Legendre polynomials.

vsyst: Reference velocity in `km/s` (default 0). The input initial guess and the output velocities are measured with respect to this velocity. This keyword is generally used to account for the difference in the starting wavelength of the templates and the galaxy spectrum as follows:

```
vsyst = c*np.log(wave_temp[0]/wave_gal[0])
```

The value assigned to this keyword is *crucial* for the two-sided fitting. In this case `vsyst` can be determined from a previous normal one-sided fit to the galaxy velocity profile. After that initial fit, `vsyst` can be defined as the measured velocity at the galaxy center. More accurately `vsyst` is the value which has to be subtracted to obtain a nearly anti-symmetric velocity profile at the two opposite sides of the galaxy nucleus.

IMPORTANT: this value is generally *different* from the systemic velocity one can get from the literature. Do not try to use that!

Output Parameters

Stored as attributes of the `pPXF` class:

.apoly: Vector with the best fitting additive polynomial.

.bestfit: Vector with the best fitting model for the galaxy spectrum. This is a linear combination of the templates, convolved with the best fitting LOSVD, multiplied by the multiplicative polynomials and with subsequently added polynomial continuum terms or sky components.

.chi2: The reduced `chi^2` (namely `chi^2/DOF`) of the fit (where `DOF ~ pp.goodpixels.size`).

IMPORTANT: if `Chi^2/DOF` is not ~1 it means that the errors are not properly estimated, or that the template is bad and it is *not* safe to set the `clean` keyword.

.gas_bestfit: If `gas_component` is not `None`, this attribute returns the best-fitting gas spectrum alone. The stellar spectrum alone can be computed as `stellar_spectrum = pp.bestfit - pp.gas_bestfit`

.gas_flux: Vector with the integrated flux (in counts) of all lines set as `True` in the input `gas_component` keyword. This is the flux of individual gas templates, which may include multiple lines. This implies that, if a gas template describes a doublet, the flux is that of both lines. If the Balmer series is input as a single template, this is the flux of the entire series.

The returned fluxes are not corrected in any way and in particular, no reddening correction is applied. In other words, the returned `.gas_flux` should be unchanged, within the errors, regardless of whether reddening or multiplicative polynomials were fitted by `pPXF` or not. The fluxes are just raw values as one could measure (with lower accuracy) by summing the pixels, within the given gas lines, on the continuum-subtracted input galaxy spectrum.

IMPORTANT: `pPXF` makes no assumptions about the input flux units: The returned `.gas_flux` has the same units as the quantity one would obtain by just summing the values of the spectral pixels within the gas emission. This implies that, if the spectrum is in units of `erg/(cm^2 s A)`, the gas flux returned by `pPXF` should be multiplied by the pixel size in Angstrom at the line wavelength to obtain the integrated line flux in units of `erg/(cm^2 s)`.

NOTE: If there is no gas reddening and each input gas templates was normalized to `sum = 1`, then `pp.gas_flux = pp.weights[pp.gas_component]`.

When a gas template is identically zero within the fitted region, then `pp.gas_flux = pp.gas_flux_error = np.nan`. The corresponding components of `pp.gas_zero_template` are set to `True`. These `np.nan` values are set at the end of the calculation to flag the undefined values. They do *not* indicate numerical issues with the actual `pPXF` calculation, and the rest of the `pPXF` output is reliable.

.gas_flux_error: *Formal* uncertainty (`1*sigma`) for the quantity `pp.gas_flux`, in the same units as the gas fluxes.

This error is approximate as it ignores the covariance between the gas flux and any non-linear parameter. Bootstrapping can be used for more accurate errors.

These errors are meaningless unless `Chi^2/DOF ~ 1`. However if one *assumes* that the fit is good, a corrected estimate of the errors is:

```
gas_flux_error_corr = gas_flux_error*sqrt(chi^2/DOF)
                    = pp.gas_flux_error*sqrt(pp.chi2).
```

.gas_mpoly: vector with the best-fitting gas reddening curve.

.gas_reddening: Best fitting `E(B-V)` value if the `gas_reddening` keyword is set. This is especially useful when the Balmer series is input as a single template with an assumed theoretically predicted decrement e.g. using `emission_lines(..., tie_balmer=True)` in `ppxf.ppxf_util` to compute the gas templates.

.gas_zero_template: vector of size `gas_component.sum()` set to `True` where the gas template was identically zero within the fitted region. For those gas components `pp.gas_flux = pp.gas_flux_error = np.nan`.

.goodpixels: Integer vector containing the indices of the good pixels in the fit. This vector is a copy of the input `goodpixels` if `clean = False` otherwise it will be updated by removing the detected outliers.

.error: This variable contains a vector of *formal* uncertainty (`1*sigma`) for the fitted parameters in the output vector `sol`. This option can be used when speed is essential, to obtain an order of magnitude estimate of the uncertainties, but we *strongly* recommend to run bootstrapping simulations to obtain more reliable errors. In fact these errors can be severely underestimated in the region where the penalty effect is most important (`sigma < 2*velscale`).

These errors are meaningless unless `Chi^2/DOF ~ 1`. However if one *assumes* that the fit is good, a corrected estimate of the errors is:

```
error_corr = error*sqrt(chi^2/DOF) = pp.error*sqrt(pp.chi2).
```

IMPORTANT: when running Monte Carlo simulations to determine the error, the penalty (`bias`) should be set to zero, or better to a very small value. See Section 3.4 of Cappellari & Emsellem (2004) for an explanation.

.polyweights: This is largely superseded by the `.apoly` attribute above.

When `degree >= 0` contains the weights of the additive Legendre polynomials of order `0, 1,... degree`. The best fitting additive polynomial can be explicitly evaluated as:

```
from numpy.polynomial import legendre
x = np.linspace(-1, 1, len(galaxy))
apoly = legendre.legval(x, pp.polyweights)
```

When `trig=True` the polynomial is evaluated as:

```
apoly = pp.trigval(x, pp.polyweights)
```

When doing a two-sided fitting (see help for `galaxy` parameter), the additive polynomials are allowed to be different for the left and right spectrum. In that case, the output weights of the additive polynomials alternate between the first (left) spectrum and the second (right) spectrum.

.matrix: Prediction `matrix[nPixels, degree+nTemplates]` of the linear system.

`pp.matrix[nPixels, :degree]` contains the additive polynomials if `degree >= 0`.

`pp.matrix[nPixels, degree:]` contains the templates convolved by the LOSVD, and multiplied by the multiplicative polynomials if `mdegree > 0`.

.mpoly: Best fitting multiplicative polynomial (or reddening curve when `reddening` is set).

.mpolyweights: This is largely superseded by the `.mpoly` attribute above.

When `mdegree > 0` this contains in output the coefficients of the multiplicative Legendre polynomials of order `1, 2,... mdegree`. The polynomial can be explicitly evaluated as:

```
from numpy.polynomial import legendre
x = np.linspace(-1, 1, len(galaxy))
mpoly = legendre.legval(x, np.append(1, pp.mpolyweights))
```

When `trig = True` the polynomial is evaluated as:

```
mpoly = pp.trigval(x, np.append(1, pp.mpolyweights))
```

.reddening: Best fitting `E(B-V)` value if the `reddening` keyword is set.

.sol: Vector containing in output the parameters of the kinematics.

- If `moments=2` this contains `[Vel, Sigma]`
- If `moments=4` this contains `[Vel, Sigma, h3, h4]`
- If `moments=N` this contains `[Vel, Sigma, h3,... hN]`

When fitting multiple kinematic `component`, `pp.sol` contains a list with the solution for all different components, one after the other, sorted by `component`: `pp.sol = [sol1, sol2,...]`.

`Vel` is the velocity, `Sigma` is the velocity dispersion, `h3 - h6` are the Gauss-Hermite coefficients. The model parameters are fitted simultaneously.

IMPORTANT: The precise relation between the output `pPXF` velocity and redshift is `Vel = c*np.log(1 + z)`. See Section 2.3 of Cappellari (2017) for a detailed explanation.

These are the default safety limits on the fitting parameters (they can be changed using the `bounds` keyword):

- Vel is constrained to be +/-2000 km/s from the first input guess
- `velscale/100 < Sigma < 1000` km/s
- `-0.3 < [h3, h4, ...] < 0.3` (extreme value for real galaxies)

In the case of two-sided LOSVD fitting the output values refer to the first input galaxy spectrum, while the second spectrum will have by construction kinematics parameters `[-Vel, Sigma, -h3, h4, -h5, h6]`. If `vsyst` is nonzero (as required for two-sided fitting), then the output velocity is measured with respect to `vsyst`.

.status: Contains the output status of the optimization. Positive values generally represent success (the status is defined as in `scipy.optimize.least_squares`).

.weights: Receives the value of the weights by which each template was multiplied to best fit the galaxy spectrum. The optimal template can be computed with an array-vector multiplication:

```
bestemp = templates @ weights
```

These weights do not include the weights of the additive polynomials which are separately stored in `pp.polyweights`.

When the `sky` keyword is used `weights[:nTemplates]` contains the weights for the templates, while `weights[nTemplates:]` gives the ones for the sky. In that case the best fitting galaxy template and sky are given by:

```
bestemp = templates @ weights[:nTemplates]
bestsky = sky @ weights[nTemplates:]
```

When doing a two-sided fitting (see help for `galaxy` parameter) *together* with the `sky` keyword, the sky weights are allowed to be different for the left and right spectrum. In that case the output sky weights alternate between the first (left) spectrum and the second (right) spectrum.

How to Set Regularization

The `pPXF` routine can give sensible quick results with the default `bias` parameter, however, like in any penalized/filtered/regularized method, the optimal amount of penalization generally depends on the problem under study.

The general rule here is that the penalty should leave the line-of-sight velocity-distribution (LOSVD) virtually unaffected, when it is well sampled and the signal-to-noise ratio (`S/N`) is sufficiently high.

EXAMPLE: If you expect a LOSVD with up to a high `h4 ~ 0.2` and your adopted penalty (`bias`) biases the solution towards a much lower `h4 ~ 0.1`, even when the measured `sigma > 3*velscale` and the S/N is high, then you are *misusing* the `pPXF` method!

THE RECIPE: The following is a simple practical recipe for a sensible determination of the penalty in `pPXF`:

1. Choose a minimum `(S/N)_min` level for your kinematics extraction and spatially bin your data so that there are no spectra below `(S/N)_min`;
2. Perform a fit of your kinematics *without* penalty (keyword `bias=0`). The solution will be noisy and may be affected by spurious solutions, however, this step will allow you to check the expected mean ranges in the Gauss-Hermite parameters `[h3, h4]` for the galaxy under study;
3. Perform a Monte Carlo simulation of your spectra, following e.g. the included `ppxf_example_simulation.py` routine. Adopt as `S/N` in the simulation the chosen value

`(S/N)_min` and as input `[h3, h4]` the maximum representative values measured in the non-penalized `pPXF` fit of the previous step;
4. Choose as the penalty (`bias`) the *largest* value such that, for `sigma > 3*velscale`, the mean difference delta between the output `[h3, h4]` and the input `[h3, h4]` is well within (e.g. `delta ~ rms/3`) the rms scatter of the simulated values (see an example in Fig. 2 of Emsellem et al. 2004).

Problems with Your First Fit?

Common problems with your first `pPXF` fit are caused by incorrect wavelength ranges or different velocity scales between galaxy and templates. To quickly detect these problems try to overplot the (log rebinned) galaxy and the template just before calling the `pPXF` procedure.

You can use something like the following Python lines while adjusting the smoothing window and the pixels shift. If you cannot get a rough match by eye it means something is wrong and it is unlikely that `pPXF` (or any other program) will find a good match:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.ndimage

sigma = 2       # Velocity dispersion in pixels
shift = -20     # Velocity shift in pixels
template = np.roll(ndimage.gaussian_filter1d(template, sigma), shift)
plt.plot(galaxy, 'k')
plt.plot(template*np.median(galaxy)/np.median(template), 'r')
```

Changelog

V7.0.0: MC, Oxford, 10 January 2020

- `capfit`: New general linear least-squares optimization function `lsqlin` which is now used to solve the quadratic subproblem.
- `capfit`: Allow for linear inequality/equality constraints `A_ineq`, `b_ineq` and `A_eq`, `b_eq`.
- `ppxf`: Use (faster) `capfit.lsqlin` for the linear fit.
- `ppxf`: Use updated `capfit.capfit` for the non-linear optimization.
- `ppxf`: Allow for linear equalities/inequalities for both the template weights and the kinematic parameters with the `constr_templ` and `constr_kinem` optional keywords.
- `ppxf`: New `set_linear_constraints` function.
- `ppxf`: Updated documentation.

V6.7.17: MC, Oxford, 14 November 2019

- `capfit`: Written complete documentation.
- `capfit`: Improved print formatting.
- `capfit`: Return `.message` attribute.
- `capfit`: Improved `xtol` convergence test.
- `capfit`: Only accept final move if `chi2` decreased.
- `capfit`: Strictly satisfy bounds during Jacobian computation.

V6.7.16: MC, Oxford, 12 June 2019

- `capfit`: Use only free parameters for `xtol` convergence test.
- `capfit`: Describe in words convergence status with nonzero `verbose`.
- `capfit`: Fixed program stop when `abs_step` is undefined.
- `capfit`: Fixed ignoring optional `max_nfev`.

V6.7.15: MC, Oxford, 7 February 2019

- Removed unused `re` import.
- Removed Scipy's `next_fast_len` usage due to an issue with odd padding size. Thanks to Eric Emsellem (ESO) for a clear example illustrating this rare and subtle bug.

V6.7.14: MC, Oxford, 27 November 2018

- Print used `tied` parameters equalities, if any.
- Return `.ndof` attribute.
- Do not remove `fixed` or `tied` parameters from the DOF calculation. Thanks to Joanna Woo (Univ. of Victoria) for the correction.
- Replaced `normalize`, `min_age`, `max_age` and `metal` keywords with `norm_range`, `age_range` and `metal_range` in `ppxf.miles_util.miles`.
- Fixed `clock` `DeprecationWarning` in Python 3.7.

V6.7.13: MC, Oxford, 20 September 2018

- Expanded documentation of `reddening` and `gas_reddening`. Thanks to Nick Boardman (Univ. Utah) for the feedback.
- `capfit` now raises an error if one tries to tie parameters to themselves. Thanks to Kyle Westfall (Univ. Santa Cruz) for the suggestion.
- `capfit` uses Python 3.6 f-strings.

V6.7.12: MC, Oxford, 9 July 2018

- Allow for `velscale` and `vsyst` to be Numpy arrays rather than scalars.
- Improved criterion for when the Balmer series is within the fitted wavelength range in `ppxf.ppxf_util.emission_lines`. Thanks to Sam Vaughan (Univ. of Oxford) for the feedback.
- Included `width` keyword in `ppxf.ppxf_util.determine_goodpixels`. Thanks to George Privon (Univ. of Florida) for the suggestion.
- Expanded `.gas_flux` documentation.

V6.7.11: MC, Oxford, 5 June 2018

- Formatted `ppxf.py` docstring in reStructuredText.
- Removed CHANGELOG from the code and placed in a separate file.
- Modified `setup.py` to show help and CHANGELOG on PyPi page.
- Included `ppxf.__version__`.

V6.7.8: MC, Oxford, 21 May 2018

- Moved package to the Python Package Index (PyPi).

- Dropped legacy Python 2.7 support.

V6.7.6: MC, Oxford, 16 April 2018

- Changed imports for the conversion of pPXF to a package. Thanks to Joe Burchett (Santa Cruz) for the suggestion.

V6.7.5: MC, Oxford, 10 April 2018

- Fixed syntax error under Python 2.7.

V6.7.4: MC, Oxford, 16 February 2018

- Fixed bug in `reddening_cal00()`. It only affected NIR lam $>$ 1000 nm.

V6.7.3: MC, Oxford, 8 February 2018

- Plot wavelength in nm instead of Angstrom, following IAU rules.
- Ensures each element of `start` is not longer than its `moments`.
- Removed underscore from internal function names.
- Included `ftol` keyword.

V6.7.2: MC, Oxford, 30 January 2018

- Included dunder names as suggested by Peter Weilbacher (Potsdam).
- Fixed wrong `.gas_reddening` when `mdegree > 0`.
- Improved formatting of documentation.

V6.7.1: MC, Oxford, 29 November 2017

- Removed import of `misc.factorial`, deprecated in Scipy 1.0.

V6.7.0: MC, Oxford, 6 November 2017

- Allow users to input identically-zero gas templates while still producing a stable NNLS solution. In this case, warn the user and set the .gas_zero_template attribute. This situation can indicate an input bug or a gas line which entirely falls within a masked region.
- Corrected `gas_flux_error` normalization, when input not normalized.
- Return `.gas_bestfit`, `.gas_mpoly`, `.mpoly` and `.apoly` attributes.
- Do not multiply gas emission lines by polynomials, instead allow for `gas_reddening` (useful with tied Balmer emission lines).
- Use `axvspan` to visualize masked regions in plot.
- Fixed program stop with `linear` keyword.
- Introduced `reddening_func` keyword.

V6.6.4: MC, Oxford, 5 October 2017

- Check for NaN in `galaxy` and check all `bounds` have two elements.
- Allow `start` to be either a list or an array or vectors.

V6.6.3: MC, Oxford, 25 September 2017

- Reduced bounds on multiplicative polynomials and clipped to positive values. Thanks to Xihan Ji (Tsinghua University) for providing an example of slightly negative gas emission lines, when the spectrum contains essentially just noise.
- Improved visualization of masked pixels.

V6.6.2: MC, Oxford, 15 September 2017

- Fixed program stop with a 2-dim templates array and regularization. Thanks to Adriano Poci (Macquarie University) for the clear report and the fix.

V6.6.1: MC, Oxford, 4 August 2017

- Included note on `.gas_flux` output units. Thanks to Xihan Ji (Tsinghua University) for the feedback.

V6.6.0: MC, Oxford, 27 June 2017

- Print and return gas fluxes and errors, if requested, with the new `gas_component` and `gas_names` keywords.

V6.5.0: MC, Oxford, 23 June 2017

- Replaced `MPFIT` with `capfit` for a Levenberg-Marquardt method with fixed or tied variables, which rigorously accounts for box constraints.

V6.4.2: MC, Oxford, 2 June 2017

- Fixed removal of bounds in solution, introduced in V6.4.1. Thanks to Kyle Westfall (Univ. Santa Cruz) for reporting this.
- Included `method` keyword to use Scipy's `least_squares()` as alternative to MPFIT.
- Force float division in pixel conversion of `start` and `bounds`.

V6.4.1: MC, Oxford, 25 May 2017

- `linear_fit()` does not return unused status any more, for consistency with the corrospinding change to `cap_mpfit`.

V6.4.0: MC, Oxford, 12 May 2017

- Introduced `tied` keyword to tie parameters during fitting.
- Included discussion of formal errors of `.weights`.

V6.3.2: MC, Oxford, 4 May 2017

- Fixed possible program stop introduced in V6.0.7 and consequently removed unnecessary function `_templates_rfft()`. Many thanks to Jesus Falcon-Barroso for a very clear and useful bug report!

V6.3.1: MC, Oxford, 13 April 2017

- Fixed program stop when fitting two galaxy spectra with reflection-symmetric LOSVD.

V6.3.0: MC, Oxford, 30 March 2017

- Included `reg_ord` keyword to allow for both first and second order regularization.

V6.2.0: MC, Oxford, 27 March 2017

- Improved curvature criterion for regularization when `dim > 1`.

V6.1.0: MC, Oxford, 15 March 2017

- Introduced `trig` keyword to use a trigonometric series as alternative to Legendre polynomials.

V6.0.7: MC, Oxford, 13 March 2017

- Use `next_fast_len()` for optimal `rfft()` zero padding.
- Included keyword `gas_component` in the `.plot()` method, to distinguish gas emission lines in best-fitting plots.
- Improved plot of residuals for noisy spectra.
- Simplified regularization implementation.

V6.0.6: MC, Oxford, 23 February 2017

- Added `linear_fit()` and `nonlinear_fit()` functions to better clarify the code structure. Included `templates_rfft` keyword.
- Updated documentation. Some code simplifications.

V6.0.5: MC, Oxford, 21 February 2017

- Consistently use new format_output() function both with/without the `linear` keyword. Added `.status` attribute. Changes suggested by Kyle Westfall (Univ. Santa Cruz).

V6.0.4: MC, Oxford, 30 January 2017

- Re-introduced `linear` keyword to only perform a linear fit and skip the non-linear optimization.

V6.0.3: MC, Oxford, 1 December 2016

- Return usual `Chi**2/DOF` instead of Biweight estimate.

V6.0.2: MC, Oxford, 15 August 2016

- Improved formatting of printed output.

V6.0.1: MC, Oxford, 10 August 2016

- Allow `moments` to be an arbitrary integer.
- Allow for scalar `moments` with multiple kinematic components.

V6.0.0: MC, Oxford, 28 July 2016

- Compute the Fourier Transform of the LOSVD analytically:
- Major improvement in velocity accuracy when `sigma < velscale`.
- Removed `oversample` keyword, which is now unnecessary.
- Removed limit on velocity shift of templates.
- Simplified FFT zero padding. Updated documentation.

V5.3.3: MC, Oxford 24 May 2016

- Fixed Python 2 compatibility. Thanks to Masato Onodera (NAOJ).

V5.3.2: MC, Oxford, 22 May 2016

- Backward compatibility change: allow `start` to be smaller than `moments`. After feedback by Masato Onodera (NAOJ).
- Updated documentation of `bounds` and `fixed`.

V5.3.1: MC, Oxford, 18 May 2016

- Use wavelength in plot when available. Make `plot()` a class function. Changes suggested and provided by Johann Cohen-Tanugi (LUPM).

V5.3.0: MC, Oxford, 9 May 2016

- Included `velscale_ratio` keyword to pass a set of templates with higher resolution than the galaxy spectrum.
- Changed `oversample` keyword to require integers not Booleans.

V5.2.0: MC, Baltimore, 26 April 2016

- Included `bounds`, `fixed` and `fraction` keywords.

V5.1.18: MC, Oxford, 20 April 2016

- Fixed deprecation warning in Numpy 1.11. Changed order from 1 to 3 during oversampling. Warn if sigma is under-sampled.

V5.1.17: MC, Oxford, 21 January 2016

- Expanded explanation of the relation between output velocity and redshift.

V5.1.16: MC, Oxford, 9 November 2015

- Fixed potentially misleading typo in documentation of `moments`.

V5.1.15: MC, Oxford, 22 October 2015

- Updated documentation. Thanks to Peter Weilbacher (Potsdam) for corrections.

V5.1.14: MC, Oxford, 19 October 2015

- Fixed deprecation warning in Numpy 1.10.

V5.1.13: MC, Oxford, 24 April 2015

- Updated documentation.

V5.1.12: MC, Oxford, 25 February 2015

- Use `color=` instead of `c=` to avoid new Matplotlib 1.4 bug.

V5.1.11: MC, Sydney, 5 February 2015

- Reverted change introduced in V5.1.2. Thanks to Nora Lu"tzgendorf for reporting problems with `oversample`.

V5.1.10: MC, Oxford, 14 October 2014

- Fixed bug in saving output introduced in previous version.

V5.1.9: MC, Las Vegas Airport, 13 September 2014

- Pre-compute FFT and oversampling of templates. This speeds up the calculation for very long or highly-oversampled spectra. Thanks to Remco van den Bosch for reporting situations where this optimization may be useful.

V5.1.8: MC, Utah, 10 September 2014

- Fixed program stop with `reddening` keyword. Thanks to Masatao Onodera for reporting the problem.

V5.1.7: MC, Oxford, 3 September 2014

- Relaxed requirement on input maximum velocity shift.
- Minor reorganization of the code structure.

V5.1.6: MC, Oxford, 6 August 2014

- Catch an additional input error. Updated documentation for Python. Included templates `matrix` in output. Modified plotting colours.

V5.1.5: MC, Oxford, 21 June 2014

- Fixed deprecation warning.

V5.1.4: MC, Oxford, 25 May 2014

- Support both Python 2.7 and Python 3.

V5.1.3: MC, Oxford, 7 May 2014

- Allow for an input covariance matrix instead of an error spectrum.

V5.1.2: MC, Oxford, 6 May 2014

- Replaced REBIN with INTERPOLATE + /OVERSAMPLE keyword. This is to account for the fact that the Line Spread Function of the observed galaxy spectrum already includes pixel convolution. Thanks to Mike Blanton for the suggestion.

V5.1.1: MC, Dallas Airport, 9 February 2014

- Fixed typo in the documentation of `nnls_flags`.

V5.1.0: MC, Oxford, 9 January 2014

- Allow for a different LOSVD for each template. Templates can be stellar or can be gas emission lines. A PPXF version adapted for multiple kinematic components existed for years. It was updated in JAN/2012 for the paper by Johnston et al. (2013, MNRAS). This version merges those changes with the public PPXF version, making sure that all previous PPXF options are still supported.

V5.0.1: MC, Oxford, 12 December 2013

- Minor cleaning and corrections.

V5.0.0: MC, Oxford, 6 December 2013

- Translated from IDL into Python and tested against the original version.

V4.6.6: MC, Paranal, 8 November 2013

- Uses CAP_RANGE to avoid potential naming conflicts.

V4.6.5: MC, Oxford, 15 November 2012

- Expanded documentation of REGUL keyword.

V4.6.4: MC, Oxford, 9 December 2011

- Increased oversampling factor to 30x, when the /OVERSAMPLE keyword is used. Updated corresponding documentation. Thanks to Nora Lu"tzgendorf for test cases illustrating errors in the recovered velocity when the sigma is severely undersampled.

V4.6.3: MC, Oxford 25 October 2011

- Do not change TEMPLATES array in output when REGUL is nonzero. From feedback of Richard McDermid.

V4.6.2: MC, Oxford, 17 October 2011

- Included option for 3D regularization and updated documentation of REGUL keyword.

V4.6.1: MC, Oxford, 29 July 2011

- Use Coyote Graphics (http://www.idlcoyote.com/) by David W. Fanning. The required routines are now included in NASA IDL Astronomy Library.

V4.6.0: MC, Oxford, 12 April 2011

- Important fix to /CLEAN procedure: bad pixels are now properly updated during the 3sigma iterations.

V4.5.0: MC, Oxford, 13 April 2010

- Dramatic speed up in the convolution of long spectra.

V4.4.0: MC, Oxford, 18 September 2009

- Introduced Calzetti et al. (2000) PPXF_REDDENING_CURVE function to estimate the reddening from the fit.

V4.3.0: MC, Oxford, 4 Mach 2009

- Introduced REGUL keyword to perform linear regularization of WEIGHTS in one or two dimensions.

V4.2.3: MC, Oxford, 27 November 2008

- Corrected error message for too big velocity shift.

V4.2.2: MC, Windhoek, 3 July 2008

- Added keyword POLYWEIGHTS.

V4.2.1: MC, Oxford, 17 May 2008

- Use LA_LEAST_SQUARES (IDL 5.6) instead of SVDC when fitting a single template. Please let me know if you need to use PPXF with an older IDL version.

V4.2.0: MC, Oxford, 15 March 2008

- Introduced optional fitting of SKY spectrum. Many thanks to Anne-Marie Weijmans for testing.

V4.1.7: MC, Oxford, 6 October 2007

- Updated documentation with important note on penalty determination.

V4.1.6: MC, Leiden, 20 January 2006

- Print number of nonzero templates. Do not print outliers in /QUIET mode.

V4.1.5: MC, Leiden, 10 February 2005

- Verify that GOODPIXELS is monotonic and does not contain duplicated values. After feedback from Richard McDermid.

V4.1.4: MC, Leiden, 12 January 2005

- Make sure input NOISE is a positive vector.

V4.1.3: MC, Vicenza, 30 December 2004

- Updated documentation.

V4.1.2: MC, Leiden, 11 November 2004

- Handle special case where a single template without additive polynomials is fitted to the galaxy.

V4.1.1: MC, Leiden, 21 September 2004

- Increased maximum number of iterations ITMAX in BVLS. Thanks to Jesus Falcon-Barroso for reporting problems.
- Introduced error message when velocity shift is too big.
- Corrected output when MOMENTS=0.

V4.1.0: MC, Leiden, 3 September 2004

- Corrected implementation of two-sided fitting of the LOSVD. Thanks to Stefan van Dongen for reporting problems.

V4.0.0: MC, Vicenza, 16 August 2004

- Introduced optional two-sided fitting assuming a reflection symmetric LOSVD for two input spectra.

V3.7.3: MC, Leiden, 7 August 2004

- Corrected bug: keyword ERROR was returned in pixels instead of km/s.
- Decreased lower limit on fitted dispersion. Thanks to Igor V. Chilingarian.

V3.7.2: MC, Leiden, 28 April 2004

- Corrected program stop after fit when MOMENTS=2. Bug was introduced in V3.7.0.

V3.7.1: MC, Leiden, 31 March 2004

- Updated documentation.

V3.7.0: MC, Leiden, 23 March 2004

- Revised implementation of MDEGREE option. Nonlinear implementation: straightforward, robust, but slower.

V3.6.0: MC, Leiden, 19 March 2004

- Added MDEGREE option for multiplicative polynomials. Linear implementation: fast, works well in most cases, but can fail in certain cases.

V3.5.0: MC, Leiden, 11 December 2003

- Included /OVERSAMPLE option.

V3.4.7: MC, Leiden, 8 December 2003

- First released version.

V1.0.0: Leiden, 10 October 2001

- Created by Michele Cappellari.