SPIF user guide v. 0.1

BY M. JUVELA AND D. THARAKKAL

Table of contents

1. Introduction

SPIF (spectrum iterative fitter) is a program that can be used to fit a set of spectra with one or more Gaussian profiles or using the hyperfine structure profile. It is similar to other software packages such as PySpeckit or Gausspy+. Its main use case is the fitting of a simple spectral model to a large number of independent spectra. Thus, it does not have all additional features, such as multi-transition fitting of hyperfine structure lines and the subsequent estimation of the kinetic temperatures, which can be found in PySpeckit. However, apart from fitting spectra, SPIF can also estimate the parameter uncertainties, either with a Monte Carlo simulation or with a Markov chain Monte Carlo (MCMC) run.

The main advantage compared to Python-based packages is the speed, which allows the fitting of samples of up to millions of spectra. The fits can be automatically repeated with many random initial conditions, thus making it more likely that the optimal solution is found. SPIF does not automatically select the number of velocity components to be fitted, unlike for example Gausspy+ does. The normal analysis is instead foreseen to include runs with different numbers of velocity components, and the most appropriate model is selected only afterwards, based on the results provided by SPIF.

The following sections describe the input files (Sect. 2), the run parameters (Sect. 3), and the output files (Sect. 4) of a SPIF runs. Further examples can be found in GitHub (see Sect. 5).

In this document, the input spectra are assumed to form a map of $N_y \times N_x$ spectra, with M velocity channels. The number of free parameters is referred to as N_P. The symbol # stands for an integeer argument (when not used as a comment sign inside and INI file).

2. Input files

The main inputs for SPIF are an INI file containing the parameters of the fitting run. In partic-

ular, that specifies the FITS files that contain the spectra and their error estimates. The only other potential input file is one that lists additional information for the fitting of hyperfine structure lines.

2.1. INI file

The INI file is a text file that consists of lines of the form `key = values`, for example

```
fits1  =  spectra.fits                    # spectra
dfits1 =  spectra_err.fits                # error estimates
y1     =  GAUSS(v1, x[0], x[1], x[2])     # model
prefix =  result
```

Here the three first keywords are mandatory. The keywords **`fits1`** and **`dfits1`** specify the FITS files that contain the spectra to be fitted and the error estimates of the measurements (see Sect. 2.2). Note the number "1" in the name of the keyword. If one fits two sets of spectra at the same time, one would also use the corresponding keywords with the number "2", to specify the FITS files for the second set of spectra (see examples below). The optional keyword **`prefix`** specifies the prefix for the FITS files that contain the results of the calculations (see Sect. 4). The default prefix is `res`.

In the example above, the third line specifies the model to be fitted, which is in this case a single Gaussian. The keyword **`y1`** refers to a spectrum read from the file specified with the keyword `fits1`, and the value **`v1`** refers to the velocity axis in those spectra. Again, when one is fitting two sets of spectra at the same time, the INI file would include a second model definition for the second spectra, where `y1` and `v1` are replaced with **`y2`** and **`v2`**. The free model parameters are written as `x[0]`, `x[1]`, etc. In the case of model **`GAUSS`**, the three parameters are the peak intensity (antenna temperature), the central velocity, and the FWHM linewidth.

The fitting of two sets of spectra at the same time is useful only if their models are linked. The following is an example of such a fit, where the velocities of the spectra are assumed to be the same.

```
fits1  =  13CO.fits                    # first set of spectra
dfits1 =  13CO_err.fits                #   with error estimates
y1     =  GAUSS(v1, x[0], x[1], x[2])  # first model
fits2  =  C18O.fits                    # second set of spectra
dfits2 =  C18O_err.fits                #   with error estimates
y2     =  GAUSS(v2, x[3], x[1], x[4])  # second model, same velocity
```

The keyword **`aux`** can appear several times, and it specifies each time another auxiliary FITS file. The values read from the aux file can be used in the definition of the model. If the spectrum files consist of $N_y \times N_x$ spectra, the auxiliary files must also contain an image of $N_y \times N_x$ values. Alternatively, the auxiliary file can have a third axis (FITS `NAXIS3`), and one selects one plane from the cube by appending `:#` to the name of the file. Here `#` stands for the index of the plane, and the suffix `:0` corresponds to *the first* plane. The following example takes *the second* plane `:1` from an auxiliary file (perhaps the result of some previous fit), and uses that as the values of the radial velocity, leaving only the intensity and thw FWHM as free parameters.

```
fits1  = spectra.fits
dfits1 = spectra_err.fits
aux    = previous_fit.fits:1
y1     = GAUSS(v1, x[0], aux0, x[1])  # aux0 for the first aux file !
```

Several auxiliary files can be added using the same keyword `aux`, but in the model definition one refers to the values as `aux0`, `aux1`, etc., in the order the files were listed in the INI file. Note that `aux0` is now a scalar value, the element of the input file that corresponds to the current spectrum.

The keyword **`init`** is used to set the initial values of the free parameters. Its value can be the name of a 3D FITS file, which then specifies the initial values separately for each spectrum and each of the free parameters (`NAXIS1`=N_y, `NAXIS2`=N_x, and `NAXIS3`=N_P). Alternatively, the value can be a list of expressions, one per free parameter. Each expression can be either a constant (a floating point number) or a value that is calculated based on the spectra and for each spectrum separately. For the latter option, the possibilities are `y1:tmax` (maximum value in the spectrum), `y1:vmax` (velocity of the channel with the maximum value), `y1:vmean` (emission-weighted mean velocity of the spectrum), and `y1:fwhm` (FWHM calculated from the channel values). The keywords (i.e. `tmax`, `vmax`, `fwhm`) are case insensitive. If the values are to be calculated based on the spectra in the second set of input spectra, `y1` is replaced with `y2`. Finally, the values computed based on a spectrum can be further modified using the `+`, `-`, `*`, and `/` operators. Thus, the full expression for the initial value of a parameter could be for example "`0.5`" or "`y1:tmax`" or "`y1:tmax*0.7`" (without the quotation marks).

In the following example one fits one set of spectra, each spectrum with two Gaussians. The initial values of the first component are calculated based on the spectrum. The initial values of the second velocity component are modified versions of those set for the first component.

```
fits1   =  spectra.fits
dfits2  =  spectra_err.fits
y1      =  GAUSS(v1, x[0], x[1], x[2]) + GAUSS(v1, x[3], x[4], x[5])
init    =  y1:tmax y1:vmax y1:fwhm  y1:tmax/2 y1:vmax+0.5 y1:fwhm*0.8
```

In the case of hyperfine fits, one must specify in the ini file also a file that contains the parameters of the hyperfine components (see Sect. 2.3). This is done using the keyword **`hfs1`**. The fitted model is called **`HFS`**, and it has four parameters: the excitation temperature T_{ex}, the radial velocity (of the reference velocity chosen in the `hfs1`parameter file), the FWHM linewidth, and the total optical depth τ. Below is an example of the INI file for a fit of hyperfine structure line.

```
fits1    =   N2H+.fits
dfits    =   N2H+_err.fits
hfs1     =   N2H+.HFS                  # name of the HFS parameter file
y1       =   HFS(v1, x[0], x[1], x[2], x[3]) # HFS spectral model
init     =   9.0  y1:vmean  1.0   0.5  # Tex=9, <v>, fwhm=1.0 tau=0.5
```

Note again that the since it is the first (and here the only) set of spectra that has the hyperfine structure, its parameterfile is specified using the keyword `hfs1`. Whe the hyperfine model applies for a second set of spectra (model for `y2`), the parameter file is specified with the keyword `hfs2`.

The model can be modified by adding penalty function, to guide the optimisation or to avoid unphysical solutions. The keyword is **`penalty`**, and its argument is any valid c-language expression. The expression can include references to the free parameters (i.e. `p[0]`, `p[1]`, etc.) as well

as values read from auxiliary files (keyword `aux`). For example, to carry out the fit of a single Gaussian where negative intensities are penalised and the radial velocity is forced to remain close to the value read from an auxiliary file, the INI file could be

```
fits1   =   13CO.fits
dfits1  =   13CO_err.fits
y1      =   GAUSS(v1, x[0], x[1], x[2])
aux     =   velocity.fits
penalty =   ((x[0]<0.0)?(-x[0]/0.1):0.0) + pow(x[1]-aux,2)
```

The penalty is added directly to the calculated χ^2 (not the reduced χ^2), and this needs to be taken into account when planning the magnitude of the penalties. In addition to general c-language statements, one can use the following pre-defined functions

- `SQUARE(x,a)`, equal to penalty $(x/a)^2$ (value of x should be close to zero)
- `POS(x,a)`, equal to penalty `(x<=0) ? (-x/a) : 0.0` (x should be positive)
- `NEG(x,a)`, equal to penalty `(x>0) ? (x/a):0.0` (x should be negative)

In the expressions x should be an expression that contains one or more free parameters (`x[0]` etc) and a is some constant (floating point number or reference to aux files, `aux0` etc.)

For example, if parameter `x[0]` should get values larger than 2.73 and the total χ^2 should double for every 0.1 units that `x[0]` is below the threshold, one can add the penalty as `MAX(x[0]-2.73, 0.1)`. Similarly, if parameter `x[1]` should be close to 5.0, penalty could be for example `SQUARE(p[1]-5.0, 1.0)`, or a less stringent `SQUARE(p[1]-5.0, 3.0)`. Because the `penalty` is added directly to the χ^2 value, it does not need to correspond to any valid probability distribution (its integral need not be equal to one or even finite). As in the example above, the penalties of different parameters should all be written on the same line and *added* together.

In MCMC calculations one can use a somewhat similar keyword **`prior`**. While the keyword `penalty` modified the χ^2, the keyword `prior` is used to specify the prior probabilities $p_{\rm prior}$ for the fitted parameters. On a practical level, since SPIF is formally optimising χ^2 values, the target function is modified as $\chi^2 \leftarrow \chi^2 - 2\ln p_{\rm prior}$. The expression following the `prior` keyword can again be any c-language expression, but it should correspond to some probability distribution. If the keyword `prior` is followed by priors for several parameters (all on the same line), those probabilities should be combined by *multiplication* (unlike in the case of `penalties`, where the individual penalties were *added* together). One can also make use of the following pre-defined functions

- `NORMAL(x,s)`, equal to $p_{\rm prior} = \frac{1}{\sqrt{2\pi}\,s}\, e^{-\frac{1}{2}(x/s)^2}$
- `BOX(x,a,b)`, equal to

$$p_{\rm prior} = \begin{cases} 0, \text{ if } x < a \text{ or } x > b \\ \frac{1}{b-a} \times 1, \text{ if } a \leqslant x \leqslant b \end{cases}$$

The constant normalisation factors (such as $\frac{1}{\sqrt{2\pi}s}$ and $\frac{1}{b-a}$ above) could be ignored in the specification of the priors. Those would shift the χ^2 values returned by SPIF but (being constants) do

not affect the location of the χ^2 minimum, the values of the Monte Carlo error estimates, or even the MCMC error estimates. In fact, those normalisation factors (greyed out above) are dropped from the definition of `NORMAL` and `BOX`, because that would make it less clear what the relative weights of the likelihood and the priors are. For example, now `BOX` does not change the returned χ^2 value as long as the condition $a \leqslant x \leqslant b$ is fulfilled, and the same applies to `NORMAL` if $x \sim 0$.

When a SPIF run includes fits that are repeated with different initial values (command line argument **-iter**, see Sect. 3 and Table 2), the perturbations of the initial values are by default to normally distributed with a standard deviation 30% of the initial initial value. However, the amount of dispersion can be specified in the INI file with the keyword **`delta`**. There should be one expression for each of the free parameters (and in the same order), and each expressions is either similar to **`r:0.2`**, for relative dispersion equal to $\sigma \sim 20\%$ of the initial value, or **`a:1.5`**, for absolute dispersion of $\sigma \sim 1.5$ units.

The following combines all of the above (if not physically then at least technically a possible setup).

```
fits1   =   13co.fits
dfits1  =   13co_err.fits
fits2   =   n2h+.fits
dfits2  =   n2h+_err.fits
hfs2    =   N2H+.HFS
aux     =   velocity.fits  # velocities read from aux file
aux     =   fwhm.fits      # fwhm from aux file
y1      =   GAUSS(v1, x[0], aux0, x[1])       # v read from aux
y2      =   HPF(v2, x[2], aux0, x[1], x[3])   # same FWHM as in y1
init    =   y1:tmax  y1:fwhm*0.8 10.0  0.5    # four initial values
delta   =   r:0.3  a:2.0  a:0.2  a:0.2        # perturbation
# penalties to prefer FWHM>0.3 and tau>0.1
penalty =   POS(x[1]-0.3, 0.1) + POS(x[3]-0.1,0.05)
#  prior for Tex ~ N(10,3)
prior   =   NORMAL(x[2]-10.0, 3.0)
```

2.2. Input files for spectra and their error estimates

SPIF expects that the spectra are provided as a FITS cube. The first two axes of the cube correspond to the sky position (FITS header `NAXIS1` and `NAXIS2`). Since SPIF (currently) fits each spectrum separately, no accurate astrometric information needs to be included in the header . The third axis (`NAXIS3`) could be either velocity or frequency or just the channel index, and the fit will proceed assuming those units. For example, we referred to the second parameter of the Gaussian model as the velocity, but it could equally be just the index of the central channel, whatever is the definition in the input FITS file . In fits of hyperfine lines, the offsets of the hyperfine components (keyword `hfs1`; see Sect. 2.3) must correspond to the axis of the FITS cube (i.e. both as channels, velocity or frequency, in the same units). The header must contain the reference pixel (`CRPIX3`, the FITS standard specifying that the first channel is the channel number 1), the value at the reference pixel (`CRVAL3`), and the step between adjacent channels (`CDELT3`). A minimal header of for a FITS cube with dimensions $N_x = 32$, $N_y = 128$, and $M = 120$ would therefore be

```
SIMPLE  =                    T / conforms to FITS standard
BITPIX  =                  -32 / array data type
NAXIS   =                    3 / number of array dimensions
NAXIS1  =                  128 / map with 128 columns
NAXIS2  =                   32 / ... and 32 rows on the sky
NAXIS3  =                  120 / 120 velocity channels
CRPIX3  =                 60.5 / index of the reference channel
CRVAL3  =                  0.0 / velocity at the reference channel
CDELT3  =                  0.1 / step between channels (e.g. [km/s])
CTYPE3  = 'velocity'           / not needed...
EXTEND  =                    T
END
```

The actual values in the cube can again be in arbitrary units, and the results of the fit will be in the same units. The exception is the hyperfine fit, where the data must be given as radiation temperature in Kelvin degrees (as defined in Mangum & Shirley, PASP 127, 266).

The error estimates are currently assumed to be the same for all channels in a spectrum. Therefore, the error estimates are provided as a 2D FITS image, with a header similar to that of the spectrum file but without the third axis (`NAXIS3`).

When working with Python, it is good to remember that the order of the axes is reversed relative to the naming in the FITS header, and while FITS standard uses 1-offset indices, Python indexes arrays staring with the index zero. Thus, if the above 3D cube is read or written using a Python FITS object `F` (where the first header unit contains the cube), the last element of the array on the Python side is `F[0].data[119,31,127]`.

2.3. Parameter file for hyperfine fits

In hyperfine fits, the keywords `hfs1` and `hfs2` in the INI file (Sect. 2.1) refer to an additional text file. Its first line contains the frequency of the main transition in units of Hz. This is followed by lines, one per hyperfine component, where the first number gives the offset of the component and the second number its relative strength. The offsets must correspond to the third axis of the FITS cube that contains the spectra (i.e. the same quantity, the same unit). The normalisation of the relative strengths of the hyperfine components can be arbitrary, but it affects the interpretation of the fit results obtained for the optical depth. Below is an example of the file in the case of the N_2H^+(1-0) transition. The sum of the values in the second column is one, and the fit would thus provide the value for the sum of the peak optical depths of the components.

```
 93.171905e9           # frequency [Hz]
 5.98       0.037037   #  { velocity, strength }
 5.0282     0.185185
 4.5892     0.111111
 0.0        0.185185
-0.956      0.259259
-1.5669     0.111111
-8.9624     0.111111
```

3. Running SPIF

The SPIF script is run from the command line, using one or more command line arguments, given in arbitrary order. Only one of these is obligatory, the **`-ini`** option that specifies the INI file that was described in Sect. 2.1. The separation between the parameters specified in the INI file and the arguments used on the command line may seem arbitrary. However, the parameters in the INI file are related to the input data and the model description (*what* is being fitted), while the command line arguments are more concerned with the setup of the practical computations (*how* the fit is performed).

To increase the probability that one find the optimal fit for each spectrum, the fit can be repeated using different initial value. The initial values as well as the dispersion of the initial values during subsequent refits are determined by the INI file (keywords `init` and `delta`). The total number of fits (the initial and the refits) is given on the command line with the argument **`-iter`**.

The calculations can be performed using any of the OpenCL devices available on the computer. The device is selected using the arguments **`-gpu`** (value 0 for CPU, value 1 for GPU) and the numbers of the potential OpenCL platforms. By default the SPIF tries to find the device amoing the four first platforms. If several alternative platforms are listed after **`-platform`**, the must be separate by commas but without any spaces. Combining the above, a basic run with 20 retries for each fit on a GPU found among platforms 0-2 is made with

```
>  SPIF.py -ini my.ini -iter 20 -gpu 1 -platforms 0,1,2
```

The basic fit gives a result file, which is a 3D FITS cube with $N_x \times N_y$ spectra (corresponding to FITS axes `NAXIS1` and `NAXIS2`) and N_p planes (corresponding to N_P).

SPIF includes three alternative routines for the model fitting, which can be selected using the argument **`-method`**. The default is the Simplex routine (`-method 0`), but often the naive gradient descent (`-method 1`) or the conjugate gradient method (`-method 2`) are faster. However, the congugate gradient method requires analytical derivatives. These are implemented for the `GAUSS` and `HPF` models but not for general penalty functions or priors. If those additional constraints are used, the method should be `0` or `1`.

SPIF can estimate the uncertainty of the fitted parameters with Monte Carlo simulations. The number of Monte Carlo samples (different noise realisations of each spectrum) is given with the command line argument **`-mc #`**. The program does normal χ^2 optimisation for the original data, and uses the results as the initial values in the sunsequent fitting of the spectra with different noise realisations. Each fit is done by default only once. However, one can use the command line argument `-mciter #` so that each fit is repated # times, using randomised initial values and selecting the best fit. An example of a Monte Carlo run with 100 noise realisation per spectrum and 10 retries in the fitting of each noise realisation would then be

```
>  SPIF.py  -ini my.ini  -mc 100  -mciter 10
```

By default a Monte Carlo run results in a single FITS file `<prefix>_dres.fits` that contains only the standard deviations of the parameters, as estimated based on the Monte Carlo samples. With command line argument **`-fullsave 1`**, the individual Monte Carlo samples of the parameters will be save to a file `<prefix>_mc_samples.fits`.

The argument **`-mcmc`** tells SPIF to make a MCMC run, using either the basic Metropilis algorithm (for `-mcmc 1`) or the Robust Adaptive MCMC algorithm (RAM, for `-mcmc 2`.) This requires several additional arguments, **`-burnin #`** for the number of initial rejected MCMC samples, **`-samples #`** for the number of MCMC samples that are to be saved, and **`-thin #`** to specify thinning, registering only every #:th sample after the initial burnin. When one additionally gives

argument **`-preopt 1`**, SPIF does first a normal fit maximum likelihood fit and uses the obtained parameter values as the starting point for the subsequent MCMC calculation. One should also in this case include a burnin phase, because that allows the MCMC routine to first optimise the step size, before the stage where samples are saved The total number of MCMC steps is the length of the burnin phase plus the thinning factor times the finally stored MCMC samples. The result file `<prefix>_mcmc_res.fits` contains a cube that contains for each parameter the mean and the standard deviation of the MCMC samples (`NAXIS3` being equal to $2N_p$, the parameter and its standard deviation being located in concecutive planes). With the additional command line argument **`-fullsave 1`**, also the individual MCMC samples will be saved to a separate file `<prefix>_mcmc_samples.fits` (the axis `NAXIS3` running over parameters and `NAXIS4` over the stored samples).

Option	Description
`aux`	name of auxiliary FITS file, subsequent aux files providing access to variables aux0, aux1, aux2
`delta`	perturbation of initial values in repated fits, specified for each of the free parameters either as relative ("r:0.3") or absolute ("a:1.5") values
`dfits1`	2D FITS image of the error estimates for the first set of spectra (`y1`)
`dfits2`	2D FITS image of the error estimates for the optional second set of spectra (`y2`)
`fits1`	3D FITS cube containin the spectra for the first set of spectra
`fits2`	3d FITS cube containing the spectra for the optional second set of spectra
`hfs1`	text file describing the hyperfine structure for spectra in the first set of spectra (`y1`)
`hfs2`	text file describing the hyperfine structure for spectra in the optional second set of spectra (`y2`)
`init`	specification of the initial values for each of the free parameters, either as a FITS cube (`NAXIS3`=N_P) or expressions (such as "`1.5`" or "`y1:tmax`" or "`y2:fwhm*0.5`")
mask	2D FITS file with a mask; spectra corresponding to pixels with values <0 will not be fitted, and the corresponding returned parameters values will be zeros
`penalty`	c-language expression for penalty added to the minimised χ^2 function or function `POS`, `NEG`, or `SQUARE`
`prefix`	prefix for the output files
`prior`	c-language expression for additional terms of probability (especially for the priors of the free parameters) or function `BOX` or `NORMAL`
`step`	relative step sizes for the free parameters in MCMC calculations (optional)
threshold	floating point value; spectra with maximum intensity below the theshold value will not be fitted, and the parameter values for these spectra will be returned as zeros
`y1`	the model specification for the spectra in the first set of spectra
`y2`	the model specification for the spectra in the optional second set of spectra

Table 1. List of possible options in the INI file

Argument	Description
`-burnin #`	in MCMC runs, number of initial MCMC steps rejected
`-fullsave #`	for #>0, save individual samples in Monte Carlo and MCMC runs
`-gpu #`	choose either a CPU (`-gpu 0`) or a GPU (`-gpu 1`)
`-iter #`	number of times a spectrum is fitted with different initial values (during normal maximum likelihood fitting, not during Monte Carlo or MCMC)
`-mc #`	number of Monte Carlo samples (random realisation) used during Monte Carlo error estimation
`-mciter #`	during Monte Carlo error estimation, thenumber of fits (with different initial values) tried for each Monte Carlo sample
`-mcmc #`	in MCMC runs, specifies either the basic Metropolis (`-mcmc 1`) or with Robust Adaptive MCMC (`-mcmc 3`)
`-method #`	selects the fitting routine between Simplex (`-method 0`), naive gradient descent (`-method 1`), and the conjugate gradient method (`-method 2`)
-nan #	normally ("-nan 0") any NaN channel values lead to NaN fit results; with "-nan 1" the channels with NaN values are ignored, allowing fits to also these spectra
`-platform #`	selects the OpenCL platform, a single one (e.g. `-platform 0`) or a set of possible ones (e.g. `-platform 2,3,4`)
`-polak #`	for conjugate gradient optimiser, choose the Polak-Ribière instead of the Fletcher-Reeves implementation
`-preopt #`	in MCMC runs, start the chains at the maximum likelihood solution
`-samples #`	in MCMC runs, the number of samples to store
`-thin #`	in MCMC runs, save only every #:th sample after the initial burnin

Table 2. List of possible command line arguments for SPIF

4. Output files

Each SPIF run produces a single output file for which the default name is `res.fits` unless the prefix is set using the `-prefix` keyword of the INI file. The contents of the file are different between the basic runs (the calculation of the maximum likelihood solution), the Monte Carlo runs, and the MCMC runs. They may also depend further on some command line arguments.

The **maximum likelihood solution** is saved to a FITS file as a 3D data array. Two of the dimensions are the same as in the input spectrum file (`NAXIS1` and `NAXIS2`), and the third axis has the dimension equal to the number of free parameters plus one (`NAXIS3` equal to N_P+1). The final element along the last axis is the χ^2 value, which also includes any potential contributions from the use of penalty functions and priors. Since the header of the output file is based on the input spectrum file, it will also contain the same astrometric information.

In **Monte Carlo runs**, the third axis runs over the mean and standard deviation values of the parameters, with the data cube containing mean values and standard deviationscalculated based

on the Monte Carlo samples. The name of that file is `<prefix>_mc.fits`. The dimension of the third axis is thus twice the number of free parameters (`NAXIS3` equal to $2N_P$). The order along `NAXIS3` is $\langle x[0]\rangle, \sigma(x[0]), \langle x[1]\rangle, \sigma(x[1]), \ldots$. If the command line argument `-fullsave 1` is used, there will be a separate file `<prefix>_mc_samples.fits` that contains all the Monte Carlo samples of the parameter values (`NAXIS3` equal toto the number of Monte Carlo samples and `NAXIS4` equal to N_p).

In **MCMC runs**, the output file is `<prefix>_mcmc.fits`. As in the case of a Monte Carlo run, this is a 3D cube of the mean values and standard deviations that are calculated over the MCMC step (over the samples specified by the command line argument `-samples`). If also command line argument `-fullsave 1` is used, the samples themselves will be stored to another FITS file with a 4D data cube. There the first two axes correspond to the position on the sky (`NAXIS1` and `NAXIS2`, identical those in the input spectrum file), the third axis runs over the samples (`NAXIS3` being equal to the argument of the `-samples` on the command line), and the last axis runs over the free patameters (`NAXIS4` equal to N_P).

5. Examples

The GitHub repository includes a test script that creates synthetic observations and uses SPIF to fit these data. The script includes examples of several cases (named E1-E9):

- spectra with a single Gaussian (single velocity component and with Monte Carlo or MCMC error estimates)

- spectra with two Gaussians (without and with penalty functions)

- two sets of spectra, both fitted with Gaussians

- hyperfine fit to one set of spectra, single velocity component (with MCMC error estimates and with bias terms)

The script will create and write to disk the spectrum files (the FITS cubes and the error maps) and the INI files used in these test cases. The script can be consulted also for examples of the command line options used. Those can be found in the script but are also written to text files E1-E9.sh when the script is run. The running of the script will produce plots that can be compared to the reference solutions included below. The images (e.g. for the error estimates) include all 64×64 spectra, the spectrum plots show selected 10×10 spectra over the same area.

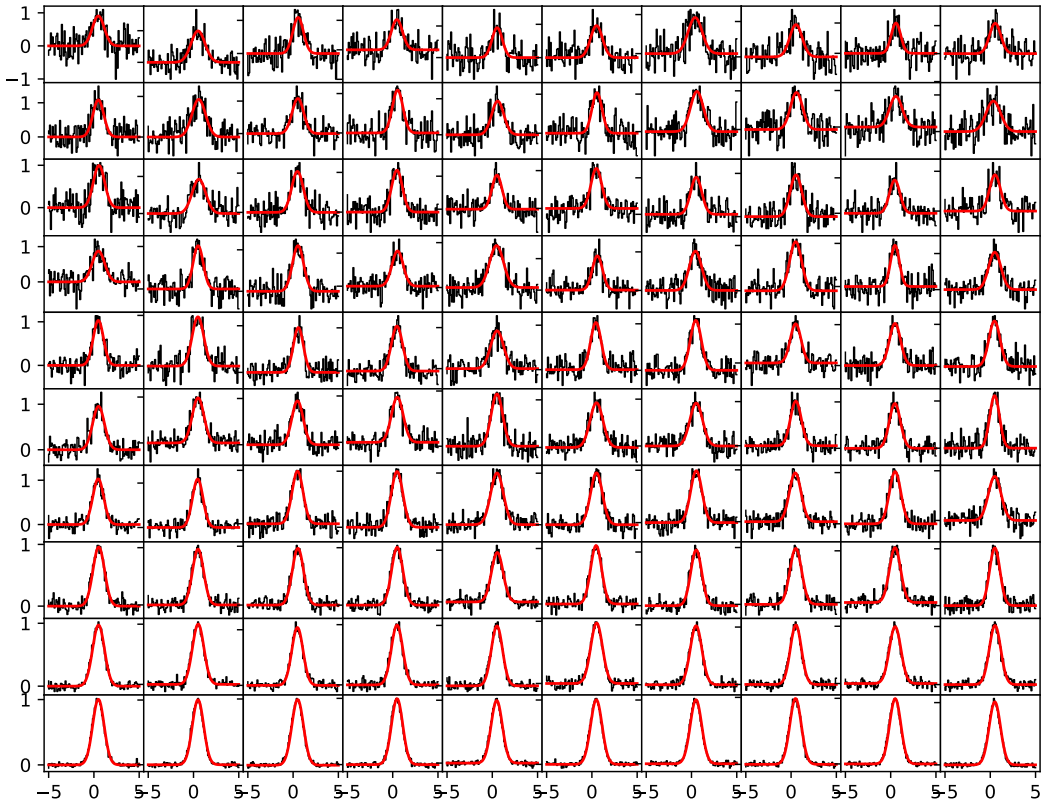

Figure 1. Test E1 with a single Gaussian line. Noise increases from 1% at the bottom to 30% at the top. The black lines show the synthetic observations, and the red lines show the fitted Gaussian profiles.

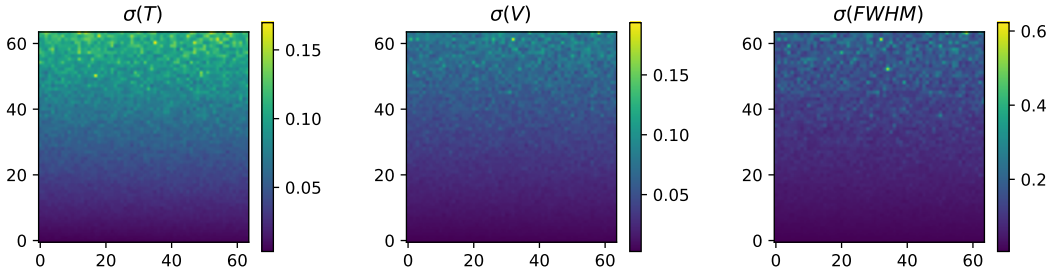

Figure 2. Test E2, Monte Carlo error estimates for the fits of the previous test E1.

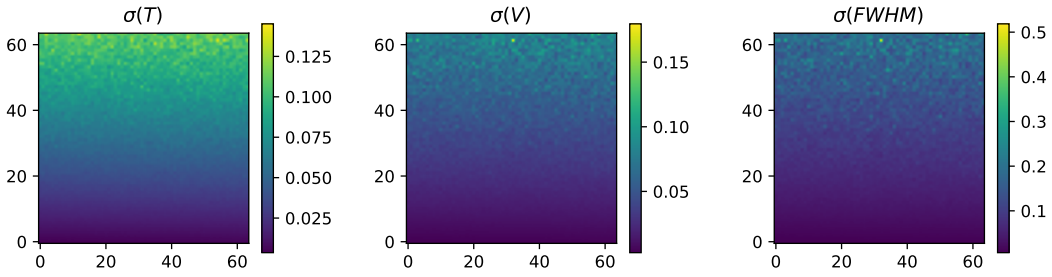

Figure 3. Test E3, MCMC error estimates for the fits from the test E1.

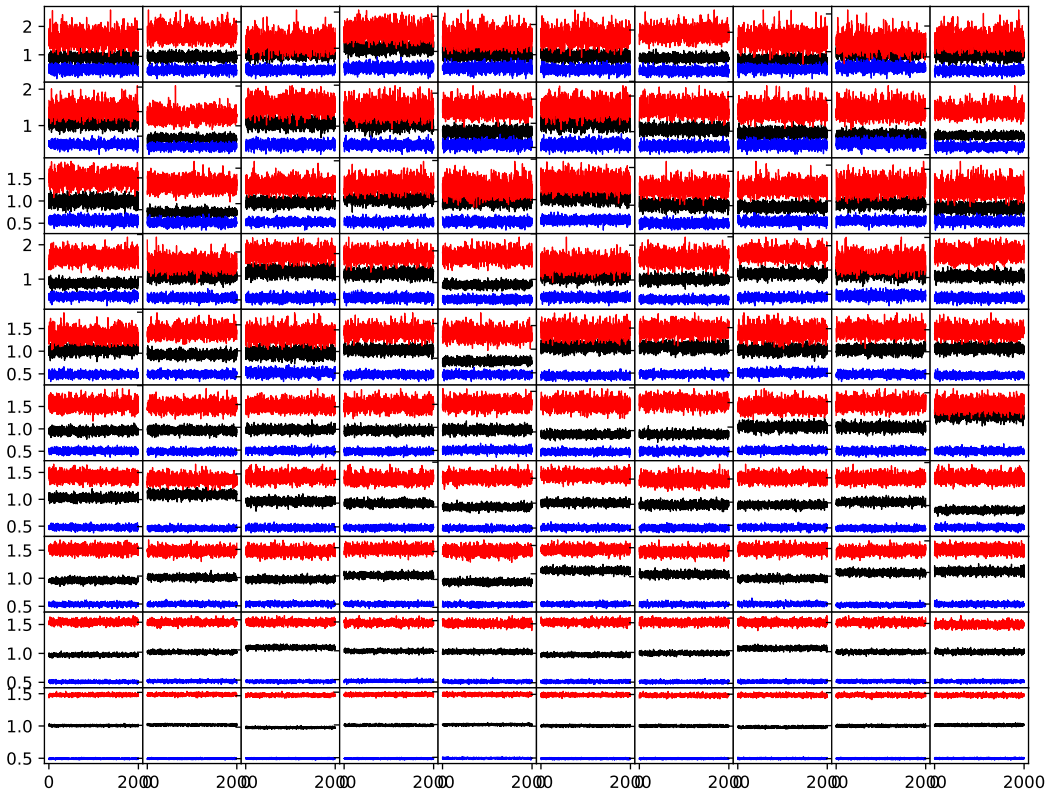

Figure 4. Test E3, saved 2000 MCMC samples (`-mcmc 1`) for the free parameters of peak intensity (black), velocity (blue), and FWHM (red). The MCMC chains are shown for the same 10×10 spectra as in Fig. 1.

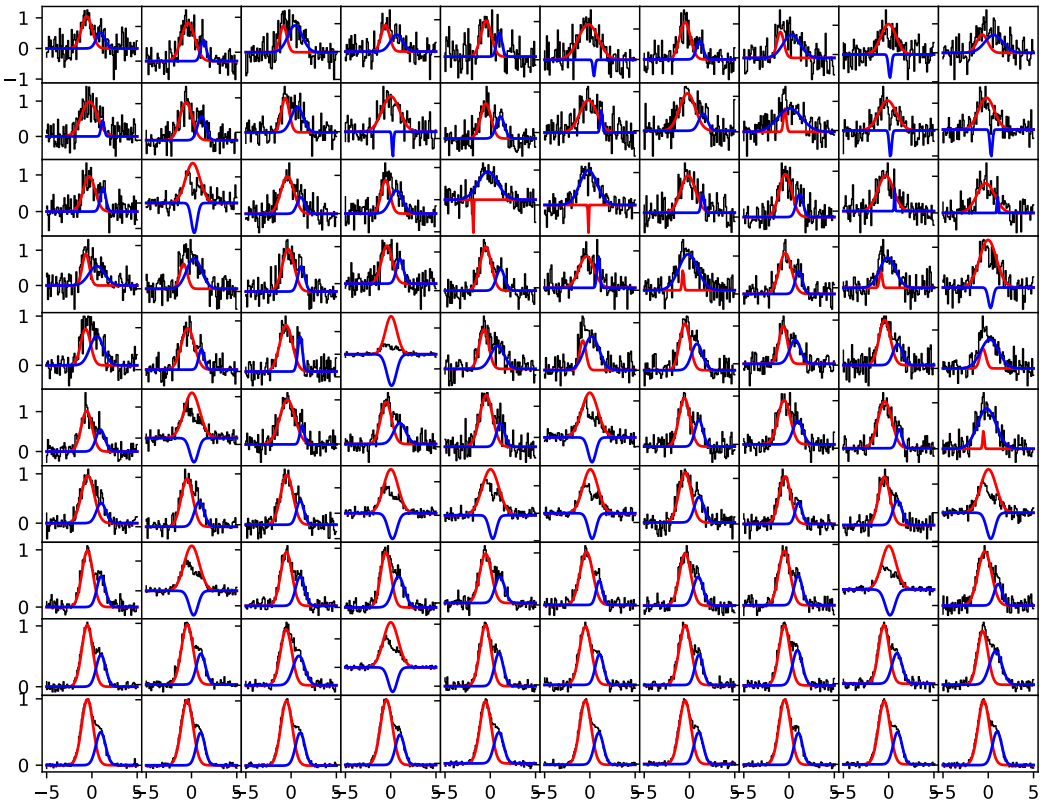

Figure 5. Sample fits from test E4, spectra with two Gaussians.

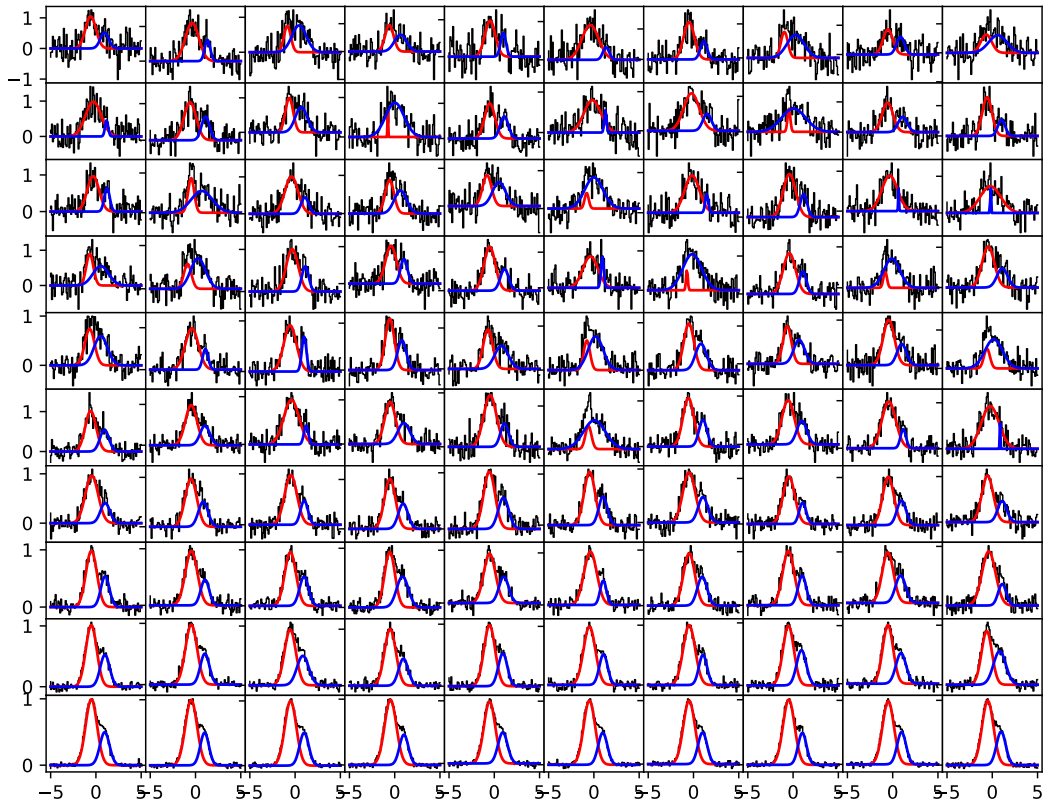

Figure 6. Sample fits from test E5, spectra with two Gaussians, and the fit including penalty for negative component intensities.

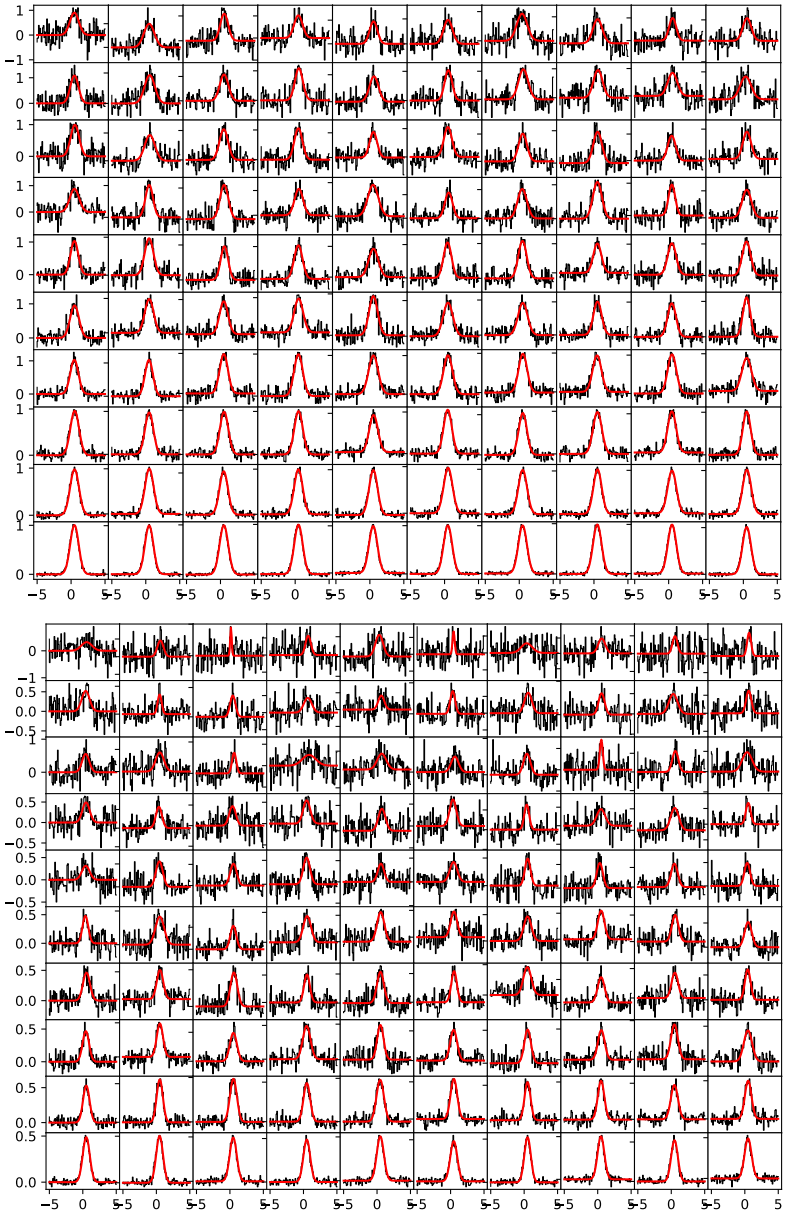

Figure 7. Sample fits from test E6, simultaneous fit to two sets of spectra, including a prior for a similar velocity (but not for the linewidth).

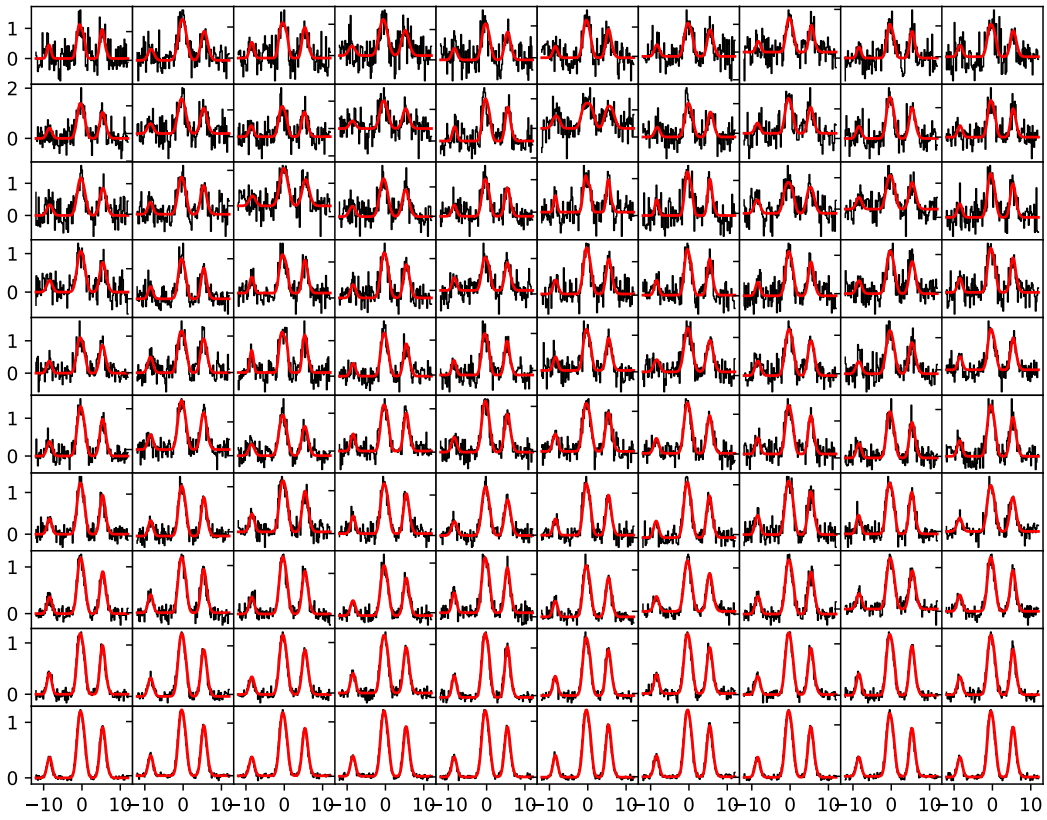

Figure 8. Example spectra from test E7, hyperfine spectra with a single velocity component.

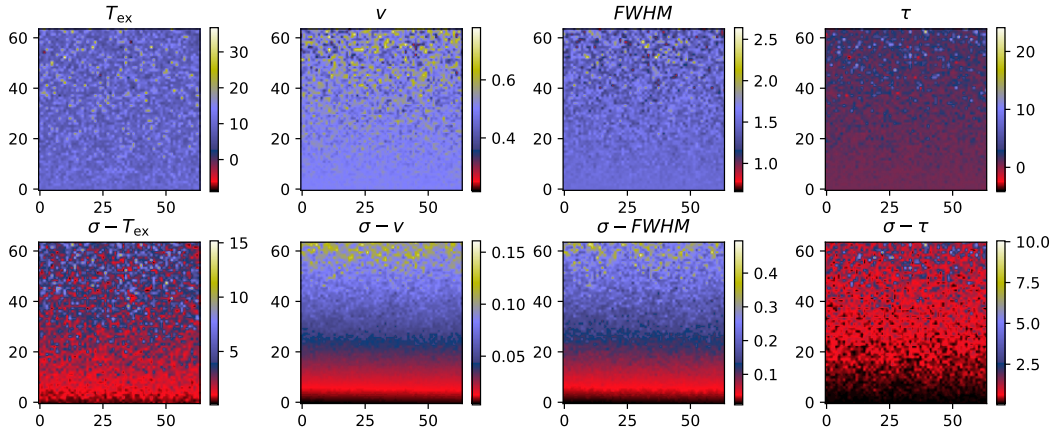

Figure 9. Results from test E8, MCMC calculations with hyperfine spectra with a single velocity component. The upper frames show the mean values of 2000 MCMC samples calculated for the four parameters and for all 64×64 fitted spectra. The bottom rows the corresponding standard deviations over the MCMC samples.

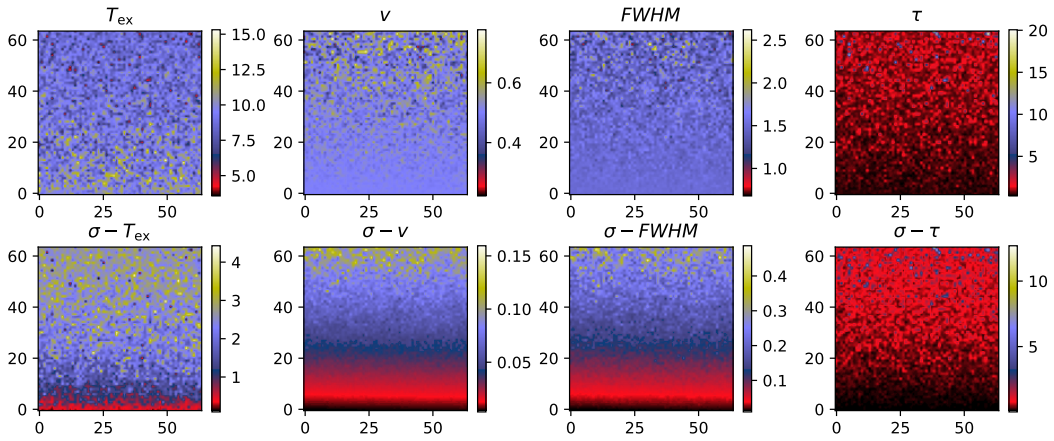

Figure 10. Results from test E9. The problem is the same as E8 (Fig. 9) but the fit includes a prior for the excitation temperature, $T_{ex} \sim N(10.0, 3.0)$ (i.e. `NORMAL(x[0]-10, 3)`).

Index