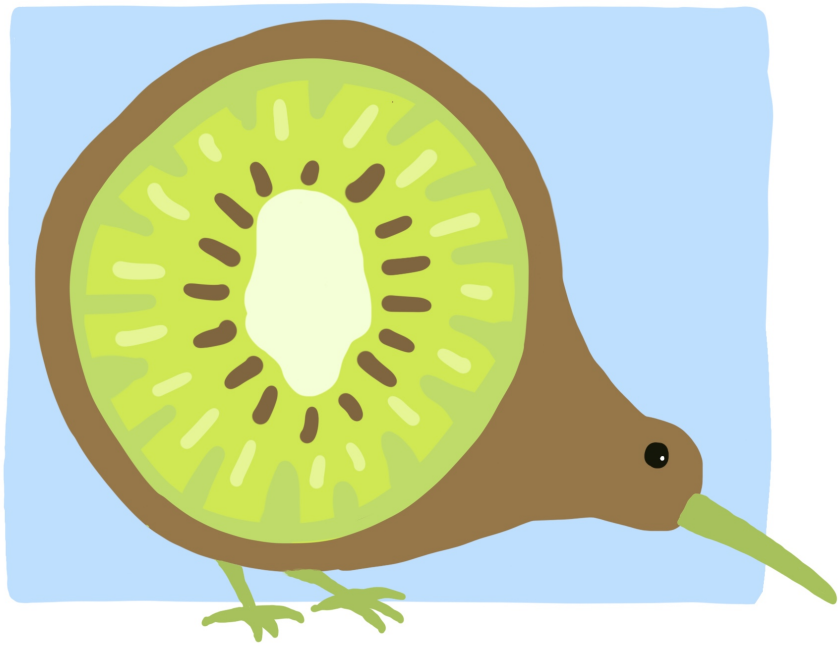
Kiwi-GA



Kiwi-GA manual

By Sarah Brands - November 23, 2021
s.a.brands@uva.nl

Contents

1 How to use this manual

KIWI-GA is a Genetic Algorithm paired up with FASTWIND that can be used for the analysis of massive star spectra. Before getting started, it is probably good to have a basic idea of how KIWI-GA works. This is very briefly described in Section 2.

In order to use KIWI-GA, you have to install some python packages, and make sure the cluster knows where to find the relevant files and packages (with cluster specific issues I can't help you, but some info is provided in Section 3). Then you can move on to Section 4: given that the installation went well, it should be straightforward to start a run with the example files provided (Section 4.1). You can then use the example input as a starting point for preparing your own run, following the detailed instructions in Section 5 and 6.

Section 7 describes the structure of the KIWI-GA output and how to plot it.

If you think you have encountered a bug, please check out Section 8.

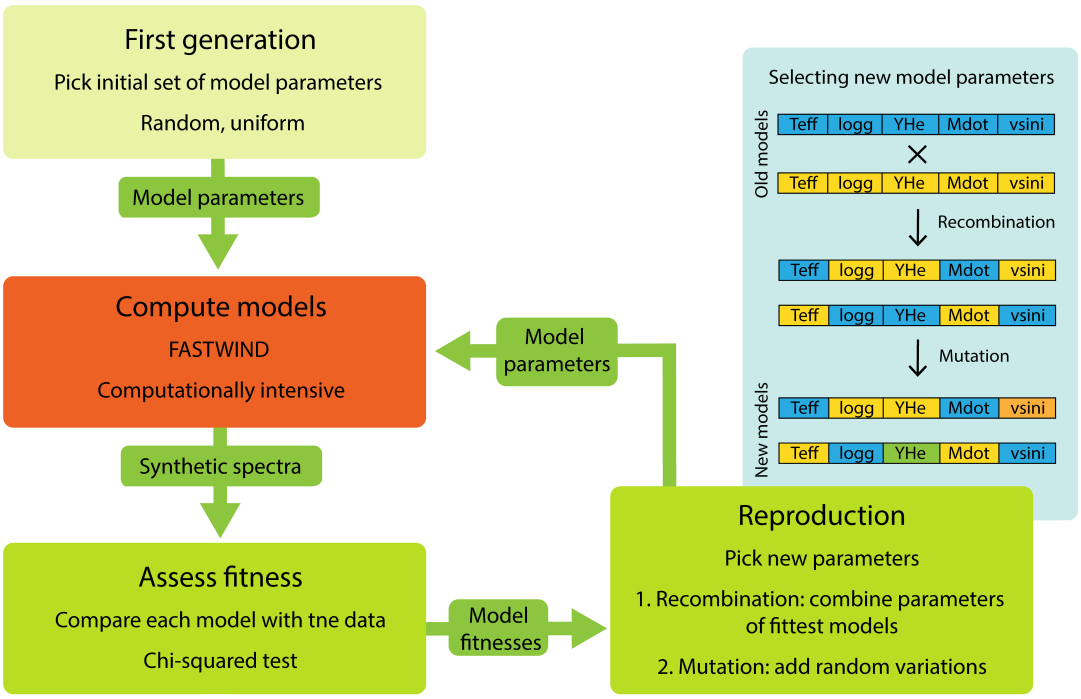


Figure 1. Schematic layout of a genetic algorithm used for fitting FASTWIND models.

2 A brief description of the algorithm

Genetic Algorithms are based on the concept of natural selection. Starting out with a generation of randomly drawn models, the fitness of each model assessed, and based on this, a new generation of models is selected. Then the process is repeated, until it converges to a set of parameters that fit the model well. This is illustrated in Figure 1. Many varieties of these algorithm exist. Here we describe briefly the design of KIWI-GA.

2.1 Recombination

KIWI-GA makes use of discrete recombination. This means that for each new individual, each of the parameters values is copied from either of the parents. The parameters are thus not altered by for example averaging or another operation. The probability of a gene being chosen is 50% for each parent, regardless of each of the parents' fitness.

2.2 Mutation

After recombination, each gene has a certain probability to undergo mutation (the mutation rate), i.e. has its value changed randomly. The typical amount with which a parameter is changed we call the mutation size. Strong mutation (high mutation rate and/or mutation size) can speed up the search and prevent premature convergence (getting stuck inside a local minimum), as parts of the parameter space that are not represented by the current population can be explored[1]. Too strong mutation, however, makes the algorithm behave like a random search, which is not efficient. It is thus a balance between speeding up the parameter search with the risk of no convergence, versus slowing down and/or getting stuck inside a local minimum.

[1]Note that when the recombination scheme is discrete (a new parameter value is equal to that of either one of the parents' values), then mutation is the only way to get new parameter values into the population.

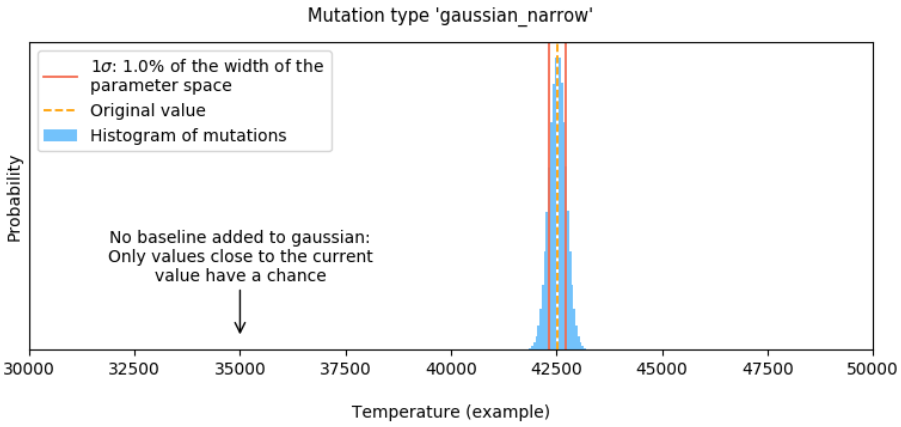


Figure 2. Distribution of mutated parameter values for KIWI-GA paramgauss_narrow. A step size of 50 for the temperature as used in this simulation. Note that if one chooses a step size that is larger than the width of the Gaussian, no mutation will occur.

In KIWI-GA mutation is implemented as follows. There are two types of mutation, Gaussian-broad and Gaussian-narrow. In both cases, the mutation is chosen from a Gaussian distribution around the current value of the parameter:

$$p(x) = b + \exp\left[\frac{-(x - x_c)^2}{2\sigma^2}\right], \tag{1}$$

where $p(x)$ is the (to be normalised) probability of a parameter with the current value x_c to mutate to x, σ is the width of the distribution, and b the baseline, that can be added to the Gaussian. For both types of mutation, the value for σ is related to the width of the search range of the parameter that is mutated, w:

$$\sigma = wf, \tag{2}$$

where f is the fraction of the width. Specific to the different types of mutation are the values of b and f.

Gaussian-narrow mutation This are small mutations that occur with a relatively high mutation rate, represented by a narrow Gaussian. Values far away from the current value have virtually no chance of occurring. The mutation rate for this type of mutation is fixed, by default to $m_{\mathrm{n}} = 0.5$. Inside KIWI-GA, the width of the Gaussian can be set as either a fraction of the width of the parameter space, as specified above, or in terms of step size. Note that if one chooses a step size that is larger than the width of the Gaussian, no mutation of this type will ever occur.

Gaussian-broad mutation This are potentially large mutations that occur with a relatively low mutation rate, represented by a broad Gaussian plus a baseline. Values closer to the current value have a larger chance of occurring, but there is a reasonable chance that values farther away are chosen. The mutation rate m_{b} can be set to be variable, see Section 5.5.2.

Distributions of values after mutation are shown in Figure 2 and 3

2.3 Reinsertion

Reinsertion of new individuals into the parent population is done according to an elitist-fitness scheme. Here the parent population may be larger than the the amount of individuals of the new generation. The exact ratios can be adjusted inside the KIWI-GA control file, and can also be done such that the new parent population is exactly the same as the new offspring population (pure reinsertion), with the exception of the best model, which is always kept. The reinsertion scheme is illustrated in Figure 4.

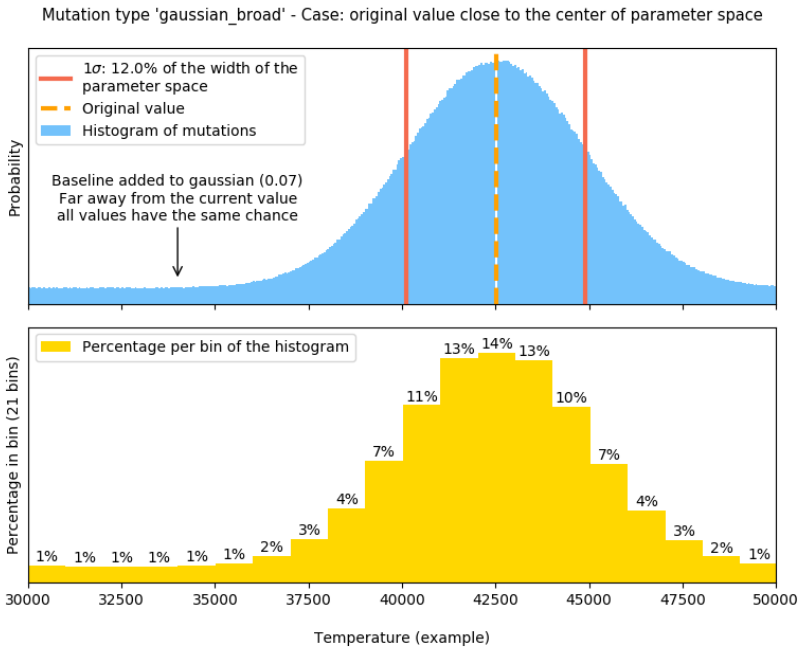


Figure 3. Distribution of mutated parameter values for Gaussian-broad mutation, in the case of the current value lying somewhere near the center of the parameter range. There is a probability of about 14% that the new value will be very close to the original value (mimicking the behaviour of Gaussian-narrow mutation. Chances that a parameter is chosen outside the Gaussian-narrow range, but inside the 1σ of the width are about 47%, and the chance of picking a parameter even further away from the current value is around 39%. These values change when the parameters for f or b are changed – these values can be varied by altering meta parameters of KIWI-GA. Note that percentages are rounded to ensure readability, so may not add up to exactly 100.

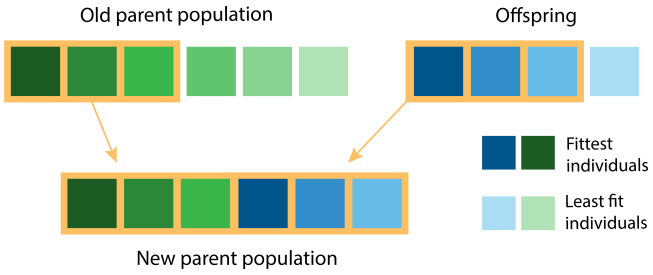


Figure 4. Schematic view of the used reinsertion scheme is a combination of elitist insertion and fitness based insertion. The actual numbers and fractions of replaced individuals, can be varied through altering meta parameters of KIWI-GA. A number of parents n_{parent} generates an number of offspring n_{ind}, where $n_{\text{parent}} > n_{\text{ind}}$. After the n_{ind} models are calculated, the best individuals of the offspring are replacing the worst individuals of the previous parent population. Based on the scheme described in `http://www.geatbx.com/docu/algindex-05.html`, following their Figure 6-1.

3 Installation

You find the code of KIWI-GA here: `https://github.com/sarahbrands/Kiwi-GA`.

KIWI-GA is tested with Python 3.6.6 on Linux, in combination with FASTWIND version 10.3 and FASTWIND version 11.

3.1 Python packages

- Make sure you have installed the following Python 3 packages: `os`, `sys`, `numpy`, `math`, `matplotlib`, `glob`, `collections,`, `warnings`, `scipy`, `argparse`, `functools`, `schwimmbad`, `PyAstronomy`, `PyPDF2`.
- When working on a cluster, the installations should be accessible both during the run, and in interactive mode (head node). For the Cartesius Surfsara cluster for example, one can do this by loading a python module after login, and then installing each package using `pip install --user package_name`.

3.2 Fastwind

For the analysis an installation of FASTWIND is required[2]. Steps for preparing the FASTWIND installation for usage with KIWI-GA:

1. Install FASTWIND on the cluster as usual (outside the KIWI-GA directory): compile in the directory `fastwind` and afterwards copy the `.eo` files to the directory `inicalc`. For versions 10.x with x < 5 and v11, you need the intel compiler `ifort` for this. For 10.5 compilation with `gfortran` should be possible.
2. Inside the main KIWI-GA directory, create a folder called `vxx_inicalc`. The first three characters of this name should denote the main version of FASTWIND: either v10 or v11. The rest can be named freely, a descriptive name can be used to distinguish between different versions of FASTWIND, if applicable.
3. Copy the complete content of `inicalc` in the original FASTWIND directory to `vxx_inicalc` inside the KIWI-GA main directory.

3.3 Testing the MPI-implementation

After compiling FASTWIND and installing the python packages listed in Section 3.1 the code is now ready for use, but you would want to make sure that the implementation of `schwimmbad` and `MPIPool` works on your cluster by doing a test run (Section 4.1).

[2] I do not provide the FASTWIND code. For this, please ask Jo Puls or Jon Sündqvist.

4 Running KIWI-GA

4.1 Example input files

Example input files including an example spectrum are provided in the folder `example_files`.

4.2 Starting a run

You start a run by preparing the input files, and executing the job script. The main thing that happens in the job script, is essentially running the file kiwiGA.py. The steps for doing this are as follows:

1. Prepare the input files (Section 5) and put them in the right location. They should be located in `input/<run_name>`.
2. Make sure the input files do have the right format. For this run `pre_run_check.py` (Section 6).
3. Navigate to the main KIWI-GA.
4. Execute the job script, or if you want and your cluster allows this, manually do the following:
   - Load the relevant modules (`ifort`, `python3`)
   - Copy the files to the right location
   - Start a run with the command:
     `srun -n <ncpu> python3 kiwiGA.py <run_name>`,
     where <ncpu> is the amount of CPUs you want to use, and <run_name> is the name of the run, corresponding to the directory with input files inside the directory `input`. Make sure that <ncpu> is a multiple of the amount of CPUs in one node at your cluster, so that you use the full capacity of the nodes assigned to you, and that this in turn corresponds to the amount of individuals as in `control.txt`, lowered by 1. Note that alternatives for the `srun` command, such as `mpiexec`, exist. Which one you need to use might depend on the cluster.

4.3 Restarting a (partially completed) run

After a run has finished or is interrupted, it can be restarted, so that continues at the point where it was ended. This means that it continues with a parent population as was there after the last generation before the run stopped. For this purpose, after each generation files ending on `_cont.txt` are stored in the output directory. You restart a run by adding the argument `-c` to the srun command as presented in Section 4.2:
`srun -n <ncpu> python3 kiwiGA.py <run_name> -c`
For this to work, at least one generation of the run should have been completed previously and be still stored in the right location, and in principle, the input files of the run should not be changed. As with starting the run, this would usually be taken care of in the job script.

4.3.1 Checking on run progress

After each generation, the output files are updated. During the run you can do the following checks:

- Just after the start of the run (say 5-10 minutes in), check if nothing is run by counting the amount of lines in the main output file `chi2.txt`: `grep chi2.txt | wc -l`. This file contains one line per finished model (see Section 7.1 for details), and thus early in the run this file should not contain more lines than about half of the number of individuals in a run. If it does, then typically something is going wrong with running the FASTWIND model (see Section 4.3.2).
- After you expect the first generation to be completed, you can check the `best_chi2.txt`. This file lists the lowest $\chi^2_{\rm red}$ in the run so far and the latest value is added after each generation. If all is ok, then after one generation this file should have one line with the best $\chi^2_{\rm red}$ so far.

- In later stages of the run, you can check the `best_chi2.txt` for tracking run progress. If the lowest $\chi^2_{\rm red}$ stays (nearly) constant over many generations this points to convergence. Note that very early in the run the lowest $\chi^2_{\rm red}$ may stay constant over several generations too, this happens when in an early generation by chance a good model was found.
- For an in depth check of run progress, generate a report of the run with `GA_analysis.py`.
- If you want to stop the run prematurely, set `ngen` to 0 in `output/input_copy/control.txt` (see also Section 4.3.3).
- For changing control parameters during the run, see Section 4.3.3.

4.3.2 Trouble shooting

Several problems you could encounter and possible approaches to solve them:

- All completed models have a $\chi^2_{\rm red}$ of 999999999. Probably something is something going wrong with running the FASTWIND model. In that case all models seem to "finish" almost immediately, $\chi^2_{\rm red}$ values of 999999999 are assumed for all models, and a next generation will be started. If this happens, look into the job-output to see what is the FASTWIND error. The error could be related to FASTWIND itself (e.g. something went wrong with compilation, or the control file is pointing to the wrong version), or to the input files that KIWI-GA created for FASTWIND (e.g. the format of the INDAT.DAT file is not appropriate, or a FORMAL_INPUT is missing a line that is listed in `line_list.txt`).
- Run hangs forever on the first generation. This happens when an error occurs in the execution of python code handling one specific model. In that case, if you are sure this does not have to do with an incompatible input file, the bug should be fixed inside the KIWI-GA code (please report to s.a.brands@uva.nl, see also Section 8.2).

4.3.3 Changing control parameters

During the run the `control.txt` parameters can be changed, as reading in this file is repeated at the start of every generation. If you would like to change parameters during a run, change the version that is found in the run directory at `output/input_copy/control.txt`, changes in the original `input/` directory have no effect.

In principle any of the parameters in `control.txt` can be changed during the run, but in practice you would like them to be fixed during a run and starting a new run in case of a mistake is the preferred option. The most likely parameter that you indeed would want to change is the number of generations `ngen`. By setting `ngen` to 0 (or any other value that is lower than the current generation), the run will stop after completing its current generation of models. Furthermore, one could manually adapt the mutation parameters during the run, but again, better keep them fixed for a given run.

5 Input files

All input files of a run should be located in the directory KIWI-GA/input/run_name with run_name the name of the run. KIWI-GA needs six input files:

- spectrum.norm: contains the observed spectrum.
- parameter_space.txt: specifies the free parameters and their ranges.
- line_list.txt: specifies the spectral regions that needed to be fitted.
- radius_info.txt: specifies how the radius should be set.
- control_info.txt: specifies control parameters of the algorithm.
- defaults_fastwind.txt: contains the default values of FASTWIND

The content of each file is described in the subsections below. After preparing the input files, their contents should be checked using the script pre_run_check.py (see Section 6).

5.1 File: spectrum.norm

The file spectrum.norm contains the observed spectrum that you want to analyse. The file should have three columns that specify:

1. Wavelength in Angstroms.
   - The wavelengths should in principle be be rest wavelengths, though one can correct for this in the line_list.txt input file (Section 5.3).
   - For lines in v10.x that have $\lambda < 2000$ this are wavelengths in vacuum, other wavelengths are in air.
   - For "CMF wavelength regions" in v11 this is always in vacuum.
2. Normalised flux. A correction on this can in principle be carried out by GA, you should specify this in the line_list.txt input file (Section 5.3).
3. Error bars on the measurements (1σ).

When preparing the file, mind the following:

- All spectral regions should be merged into one file.
- The exact regions that will be fitted are further specified elsewhere. It is thus no problem if this file includes spectral regions that you do not want to include in the fit.

An example of a spectrum.norm file is shown below.

```
1.16583034876492018e+03 8.91009419534686153 4e-01 5.707910408589814827e-02
1.16641293067846527 3e+03 8.98045901240675958 9e-01 5.753844931770196736e-02
1.16699651187488098 0e+03 8.88465920848207968 4e-01 5.801333258407799509e-02
1.16758009307129668 7e+03 8.78766774998624211 6e-01 5.850360358896115193e-02
1.16816267498484171 5e+03 7.96010466723501286 5e-01 5.900903581727302144e-02
...
```

Listing 1. The first few lines of a spectrum.norm file.

5.2 File: parameter_space.txt

This file contains a list of parameters, one line each, in the following format:

parameter_name lower_bound upper_bound step_size with:

- parameter_name: the name of the parameter. This should be matching the name of the parameter as in KIWI-GA, a list is given in Section 5.2.2.
- lower_bound and upper_bound: lower and upper bounds of the parameter space that will be searched, for parameter_name. This means that KIWI-GA will search the range lower_bound $\leq$ x $\geq$ upper_bound for the given parameter x. It is advisable to choose a range that exceeds the expected value for the parameter.
- step_size: to prevent the computation of models that are very similar, the parameter space for each parameter (see previous point) is not continuous but is divided into steps of a certain step_size. What is a reasonable value for each parameter depends on the data quality.

When preparing the file, mind the following:

- The step_size should be chosen such, that the range as specified with lower_bound and upper_bound can be divided in an integer amount of chunks of size step_size. See e.g. the input line for teff in the example file below.
- If lower_bound = upper_bound a fixed value equal to lower_bound and upper_bound will be assumed for this parameter. We call this parameter a fixed parameter. The parameter will thus not be free/fitted. See for example vinf in the example file below, which is fixed at 2100.
  - If a parameter is fixed, the step size does not matter.
  - If a parameter is fixed in parameter_space.txt but also present in defaults_fastwind.txt, the value as in parameter_space.txt will be used.
  - For some parameters fixing them at a specific value has a different meaning (Section 5.2.1).
  - For parameters without such a special meaning, setting lower_bound = upper_bound = -1 is equivalent to removing the parameter from the file: a default value as specified in defaults_fastwind.txt will be used.
- When a parameter is not in parameter_space.txt but a value is needed for the parameter to compute a model, a default value as specified in defaults_fastwind.txt is used.
- No default values are specified for Teff, logg, mdot, yhe, vinf, vrot, macro, fvel. These parameters should thus always be present in the parameter_space.txt file either as a free parameter, or as a fixed parameter. They cannot have the value -1 unless they are added to defaults_fastwind.txt[3]. Exception is fvel, which does only need to be specified when fic > 0, but can be -1 otherwise (see also Section 5.2.1).

An example of a parameter_space.txt file is shown below.

```
teff 35000 57000 500
logg 3.25 4.75 0.025
mdot -7.5 -5.0 0.05
yhe 0.095 0.095 0.005
vinf 2100.0 2100.0 00.0
vrot 110.0 110.0 5.0
macro -1.0 -1.0 0.0
micro 15.0 15.0 1.0
beta 0.7 2.0 0.025
C 7.0 8.2 0.025
N 6.9 9.0 0.05
O 8.00 8.00 0.00
Si 7.10 7.10 0.00
```

Listing 2. A parameter_space.txt file for a fit with 7 free parameters.

[3]It is easy to implement this, but it simply makes no sense to have default values for these parameters.

5.2.1 Special parameter values

When preparing `parameter_space.txt`, keep in mind that certain values of a parameter have a special meaning. These parameters and their special values are listed below. The definition and units of the parameters mentioned here are listed in Table 1.

Broadening parameters

- `macro` $= -1$: No macro turbulence included, i.e. macro turbulent velocity is 0.
- `vrot` $= -1$: No rotational broadening included, i.e. rotational broadening is 0.

X-ray parameters

- `fx` > 0: include X-rays. The maximum value of `fx` is be 16 (see Carneiro et al., 2016, for details).
- `logfx` $< \log(16)$: include X-rays, and explore this parameter in log-space. Note: when `logfx` is used, set `fx` < 0. They cannot both be a free parameter.
- `fx` < 0 and `logfx` $<$ log(16): do not include X-rays. In this case do not set any other X-ray parameters as a free parameter.
- `fx` > 1000: estimate the volume filling fraction of each model based on its mass loss and terminal velocity, based on the observational relation found by Kudritzki, Palsa, Feldmeier et al. (1996, their Figure 6). The following equation is used:

$$\log(\texttt{fx}) = -5.45 - 1.05 * \log(\texttt{mdot}/\texttt{vinf}) \tag{3}$$

  In this case make sure `logfx` $>$ log(16), so that this parameter is ignored.

Note that X-rays are not implemented in FASTWIND for stars with `teff` < 25000 K. If you include X-rays in the KIWI-GA run and also allow `teff` to be lower than 25000 K, simply no X-rays will be included for the low `teff` models, while they will be included for the higher `teff` models.

Clumping parameters

- `fic` $= -1$: optically thin clumping. In this case, `fvel` cannot be a free parameter, as it is not used at all for computing the models.
- `fic` > 0: optically thick clumping. In this case `fvel` needs to be specified in `parameter_space.txt`, either as a fixed or a free parameter.
- `vclmax` $= -1.0$: a value of max(2·`vclstart`,1.0) is adopted for `vclmax`[4].
- `logfclump` < 3: explore the clumping parameter in log-space. Note: when `logfclump` is used, do not use `fclump`. They cannot both be a free parameter.

Note that optically thick clumping is not yet implemented in FASTWIND v11.

Terminal velocity

- `vinf` $= -1$: in this case the terminal velocity is determined per model, by computing the (non-effective) escape velocity and multiplying this by a factor (2.6 for `teff` $>$ 21000, 1.3 for 10000 $>$ `teff` $>$ 21000, and 0.7 otherwise).

[4]There are some other relations between vclstart and vclmax in the code of KIWI-GA, such as to add a fixed value, but currently there is no implementation for switching between them quickly.

Table 1. Parameters that can be fitted with KIWI-GA. The names as listed here should be used in `parameter_space.txt`. [a] Parameters that cannot be analysed with FASTWIND v11. Note that while `fx` is included in v11, but with different parameters than in v10, and is therefore not yet included in KIWI-GA. [b] Parameters that cannot be analysed with FASTWIND v10. [c] See Sundqvist & Puls (2018) for a precise definition. [d] See Carneiro et al. (2016) for a precise definition. [e] The value of this parameter in `parameter_space.txt` can have a special meaning (Section 5.2.1).

Parameter name	Description	Units
`teff`	Effective temperature	K
`logg`	Gravitational acceleration	log [cgs]
`mdot`	Mass loss	log [$M_\odot$ yr^{-1}]
`yhe`	Helium fractional number density: $n_{\rm He}/n_{\rm H}$	-
`vrot`	Projected radial velocity $v \sin i$	km s^{-1}
`macro`[e]	Macro turbulent velocity	km s^{-1}
`micro`[e]	Micro turbulent velocity	km s^{-1}
`vinf`[e]	Terminal velocity of the wind	km s^{-1}
`windturb`	Wind turbulence velocity, as fraction of `vinf`	`vinf`
`beta`	Beta-law parameter (wind acceleration)	-
`fclump`	Clumping factor	-
`logfclump`[e]	Clumping factor (explore in log space)	-
`fic`[a,e]	Interclump density contrast	-
`fvel`[a]	Velocity-porosity clumping factor	-
`vclstart`	Onset velocity of clumping, as fraction of `vinf`	`vinf`
`vclmax`[e]	Velocity at which clumping reaches maximum as fraction of `vinf`	`vinf`
`fx`[a,e]	X-ray volume filling fraction	-
`logfx`[a,e]	X-ray volume filling fraction (explore in log space)	-
`uinfx`[a,d]	Maximum jump velocity of the shocks	km s^{-1}
`mx`[a,d]	Value that determines the onset of shocks	-
`Rinx`[a,d]	Value that determines the onset of shocks	$R_\odot$
`gamx`[a,d]	Power law exponent of shocks	-
`C`	Carbon abundance log $[n_{\rm C}/n_{\rm H}] + 12$	-
`N`	Nitrogen abundance log $[n_{\rm N}/n_{\rm H}] + 12$	-
`O`	Oxygen abundance log $[n_{\rm O}/n_{\rm H}] + 12$	-
`Si`	Silicon abundance log $[n_{\rm Si}/n_{\rm H}] + 12$	-
`Mg`	Magnesium abundance log $[n_{\rm Mg}/n_{\rm H}] + 12$	-
`P`	Phosphorus abundance log $[n_{\rm P}/n_{\rm H}] + 12$	-
`Fe`[b]	Iron abundance log $[n_{\rm Fe}/n_{\rm H}] + 12$	-
`S`[b]	Sulphur abundance log $[n_{\rm S}/n_{\rm H}] + 12$	-
`Na`[b]	Sodium abundance log $[n_{\rm Na}/n_{\rm H}] + 12$	-
`Ca`[b]	Calcium abundance log $[n_{\rm Ca}/n_{\rm H}] + 12$	-

5.2.2 List of parameter names

When defining the parameter space inside `parameter_space.txt`, the parameters should be specified as listed in Table 1. Mind that several parameters can have a different meaning when they are given a certain value (Section 5.2.1), and that some parameters can only be fitted using either FASTWIND v10 or v11.

5.3 Input file: `line_list.txt`

In the linelist you specify for which part of the spectrum the radiative transfer is being computed by FASTWIND, and which spectral regions exactly you want to fit. Furthermore, the resolution with which the spectra are convolved is specified here. Per spectral line/region, there is one line in the `line_list.txt` file. The columns are as following:

1. Line name:
   - v10: This should be the name of the line exactly as it is listed in the FORMAL_INPUT file.
   - v11: Has the following format: `UV_xxxxx_yyyyy`, where 'UV' indicates that it is a CMF line (it does not have to be a UV spectral region), `xxxxx` and `yyyyy` are the lower and upper limits of the wavelength range respectively, in Å (rounded to integer values), filled with leading zeros up to five digits. These lines do *not* have to be present in FORMAL_INPUT, they will be added to that file automatically during the run.
2. Resolving power R of the data at that spectral region.
3. Lower limit of the wavelength range that will be fitted.
4. Upper limit of the wavelength range that will be fitted.
5. Radial velocity correction of the line. A velocity > 0 means a blueward shift of the data, to compensate for the redward shift of the wavelengths due to the velocity of the star.
6. Renormalisation blue wavelength $\lambda_{\mathrm{rn,b}}$: Wavelength at which the deviation from 1 of the normalised continuum flux in `spectrum.norm` is analysed (see explanation below). In case no renormalisation is needed, set to a random wavelength, e.g. the lower limit of the wavelength range that will be fitted.
7. Renormalisation blue flux $\Delta F_{\mathrm{rn,b}}$: the value is the deviation from 1 of the normalised continuum flux in `spectrum.norm`, at the wavelength specified in the next column (see explanation below). In case no renormalisation is needed, set this value to 0.
8. As column 6, but then for the right (red) normalisation correction ($\lambda_{\mathrm{rn,r}}$).
9. As column 7, but then for the right (red) normalisation correction ($\Delta F_{\mathrm{rn,r}}$). In case no renormalisation is needed, set this value to 0.
10. Step size in Å, that determines the resolution at which the radiative transfer should be computed in the case of a v11 CMF line:
    - This should be compatible with the data: set it too high and the model cannot resolve features that you can resolve in the data, set it to low and the run will be unnecessarily slow. Also take into account the (expected) rotational broadening of the lines.
    - For v10 lines, this parameter is not used and should be set to 0.

Columns 7 to 8 can be used for correcting the normalisation of the spectrum within KIWI-GA. The correction is simply a linear fit to the two points ($\lambda_{\mathrm{rn,b}}$, $F_{\mathrm{rn,b}}$) and ($\lambda_{\mathrm{rn,r}}$, $F_{\mathrm{rn,r}}$), and then dividing the original spectrum by that line. Mathematically this comes down to:

$$b = (\Delta F_{\rm rn,r} - \Delta F_{\rm rn,b})/(\lambda_{\rm rn,r} - \lambda_{\rm rn,b}) \qquad q = 1 - \Delta F_{\rm rn,b} - b \cdot \lambda_{\rm rn,b}$$
$$l = q + b \cdot \lambda \qquad F_{\rm rn} = F_{\rm orig}/l,$$

where $F_{\rm orig}$ is the original normalised flux, $F_{\rm rn}$ the renormalised flux that will be used for fitting, and the other parameters as specified in the list above.

Parts of `line_list.txt` files are shown below.

```
NV1240    1250 1230.5  1249.56 0.0   1230.5   0.0  1249.56  0.0 1.0 0.0
OIV1340   1250 1340.00 1345.74 0.0   1337.00  0.0  1345.74  0.0 1.0 0.0
CIV1548   1250 1528.7  1566.97 0.0   1528.7   0.0  1566.97  0.0 0.5 0.0
HZETA     7000 3872.73 3903.25 800.0 3872.73  0.1  3903.25 -0.1 1.0 0.0
...
```

Listing 3. A `line_list.txt` file for v10 lines. Corresponds to the pre-run report shown in Figure 5.

```
UV_00900_01100 2500  900.51 1099.5  0.0  900.51 1099.5 0.0 1.0 0.1
UV_01200_01800 4000 1201.78 1798.5  0.0 1201.78 1798.5 0.0 1.0 0.1
UV_06000_07007 9000 6001.00 7006.0  0.0 6001.00 7006.0 0.0 1.0 0.1
...
```

Listing 4. A `line_list.txt` file for CMF v11 spectral regions.

5.4 Input file: `radius_info.txt`

In a KIWI-GA run the radius is not a free parameter. The user defines the radius inside `radius_info.txt`, which (apart from some comments at the top) only contains one line. There are two manners in which the radius can be defined:

1. Fixed to a certain value. In this case, the format of the input file is:
   `fixed_radius <radius in solar radii>`.
2. Fix the magnitude of the star in a certain band. In this case, before a model is computed, the radius is estimated based on the temperature and the magnitude anchor. The format of the input file is then:
   `<bandname> <magnitude> <zero point system>`.
   The magnitude must be the absolute, dereddened magnitude. Note that for this estimate not a computed SED is used, but an adapted Planck function. Therefore, after running KIWI-GA, one should correct for this by computing an SED of the best fit model, and checking how the magnitude of the relevant spectral band compares to the anchor magnitude. Then radius, as well as parameters that depend on that, such as luminosity, mass-loss rate and spectroscopic mass, should be scaled accordingly (see Mokiem et al., 2005, their Section 2.5).

An example `radius_info.txt` file is shown below.

```
# Specify radius or luminosity anchor
# If fixed radius type:
# fixed_radius <radius in solar radii>
# If luminosity anchor type:
# <bandname> <observed magnitude> <zero point system>
SPHERE_Ks -4.68 vega
```

Listing 5. A `radius_info.txt` file where an anchor magnitude is used.

5.5 Input file: `control.txt`

The file `control.txt` contains all parameters that determine the working of the algorithm, and that point KIWI-GA to the right FASTWIND directory. The most imporant parameters are listed in Table 2, while parameters you would probably want to keep at default are listed in Table 3.

An example of a `control.txt` file is shown below.

```
nind                    191             # number of models per generation
ngen                    60              # number of generations

# Detailed parameters controlling population size
f_gen1                  1.33            # nind in first gen = nind*f_gen1
ratio_po                1.33            # Ratio parent/offspring population
f_parent                0.25            # Fraction of parents kept
                                        # must satisfy: f_parent >= 1-1/ratio_po

# Specify fitting method and fastwind properties
fitmeasure              chi2            # fitness measure used for reproduction
                                        # choose between 'chi2' or 'fitness'
modelatom               A10HHeNCOPSi    # fastwind model atom
fw_timeout              35m             # maximum runtime of fastwind
inicalcdir              v10_HHeNCOPSi/  # relative path to inicalc master
...
```

Listing 6. Part of a `control.txt` file.

5.5.1 Use of the fitness measure $\mathcal{F}$

There are two reasons one wants to assess the goodness-of-fit of each model:

1. During the run the models need to be ordered by their goodness-of-fit to the data, in order to assign them a probability for being selected as a parent model for the next generation. This is the essence of GAs (the best models reproduce, i.e. survival of the fittest), and the parameter `fitmeasure` specifies the method for doing this.

2. In post-processing of the run output, one wants to assess which model has the best fit parameters and what are the uncertainties thereof. This is the end product of a GA run.

The χ^2 statistical framework is an often used method for determining goodness-of-fit, and this is (and always has been) the obvious goodness-of-fit measure for the post-processing of GA runs (second point). For the first point (regulating GA exploration), one needs exactly the same information, as GA needs to know how well parameters match the data in order to select the fittest parents for reproduction. Therefore χ^2 (denoted as `chi2` in `control.txt`) seems a suitable choice for `fitmeasure`.

In the past however, a goodness-of-fit measure called fitness $\mathcal{F}$ was introduced, to account for inhomogeneous data, especially different amounts of data points in the different spectral lines. $\mathcal{F}$ is the inverse of the average of the $\chi^2_{\rm red}$ value of each individual spectral line $\chi^2_{{\rm red},l}$, and should thus be independent of the amount of data points in a given line. Furthermore, line weights w_l were introduced to account for missing physics in the FASTWIND models:

$$\mathcal{F} = \left(\frac{\sum_l w_l \chi^2_{{\rm red},l}}{N_l} \right)^{-1} \tag{4}$$

with N_l the total number of spectral lines.

While weighing by $\mathcal{F}$ might in some aspects be a more 'fair' manner to account for different parts of the data that χ^2, it has its own disadvantages. This includes the effect that lines with low and high signal-to-noise get the same weight, while one could argue that higher signal-to-noise data gives more, or at least more certain, information. Furthermore, it is not so obvious that each spectral line should carry the same weight in the first place: a line with fewer data points might actually carry less information, and also each line has different behaviour with respect to stellar and wind properties, so assessment of the importance of each line will always be arbitrary to some extent. Lastly, using $\mathcal{F}$ instead of χ^2 results in an inconsistency with the determination of the best fit parameters and uncertainties, which are computed using χ^2. Namely, the GA searches according to one goodness-of-fit measure, and afterwards another goodness-of-fit measure is used to determine the final

Table 2. Most important parameters in `control.txt`. Table 3 lists the other parameters.

Parameter	Default	Comment
nind	71-192	Number of models per generation. The amount needed depends on the number of free parameters, where for 6 free parameters I typically set 71, and for 12 I set 191. Choose this number such that it equals an integer multiple of the amount of CPUs in one node, minus 1, this way maximizing CPU efficiency. (One CPU is reserved for managing the processes on the other CPUs)
ngen	30-80	Number of generations. The amount needed depends on the number of free parameters where more free parameters require a larger amount of generation to converge. Related to this, with a larger nind (relative to the number of free parameters) you might need less parameters for convergence + exploration of the uncertainty regions (because during convergence uncertainty regions are already explored, and this goes quicker with a large nind). For the runs I did with KIWI-GA so far, I needed 20-30 generations for convergence with 6 free parameters, and 30-80 for convergence with 12 free parameters.
f_gen1	1.33	can be used to compute more models for the first generation, with nind in first gen = nind*f_gen1. In the first generation typically several models crash in an early stage, these CPUs then have no job but to wait for the other CPUs to finish their model and can be occupied by enlarging the first generation.
ratio_po	1.33	The parent population (the pool of models that is used to pick new models) can be larger than the offspring population (which, because of the amount of CPUs assigned to a run, should always equal nind). Setting this value to > 1 allows the pool of parent models to include both models of the last generation, as well as models of even earlier generations.
f_parent	0.25	Fraction of the models of the original parent population, to be kept for the next parent population, after a generation has completed (and the offspring become the new parents). Keeping a certain fraction of 'old parents'[5] in the new parent population creates a fitter parent population, as the models of the offspring population (now becoming parents) are not necessarily fitter than their parents. This parameter must satisfy f_parent $\geq$ 1-1/ratio_po, because otherwise ratio_po cannot be satisfied.
fitmeasure	chi2	Either chi2 (strongly recommended) or fitness. See also the explanation in Section 5.5.1.
modelatom	A10HHe...	FASTWIND model atom to be used. This should correspond to the .eo files inside inicalcdir.
fw_timeout	35m	After this amount of time, the program pnlte_xx.eo is killed, regardless of convergence. In this time, you would want most of the models to be converged. The few that are not are prematurely cut off, because if one model takes exceptionally long, the other nind CPUs are waiting and doing nothing. The time it takes for most models to complete depends on the cluster, FASTWIND version, and potentially on the part of parameter space that is being explored. It turned out that 35m (35 minutes) is a good time for FASTWIND v10, for v11 this should be longer. One could start a run with a long fw_timeout and look how the amount of models inside the run directory changes over time, to investigate what is a reasonable value for this parameter[6].
inicalcdir	v10_...	Name of the inicalc directory inside the main KIWI-GA directory. This directory name should start with either v10_ or v11_ in order to indicate which version of FASTWIND is contained in the directory.
mut_rate_init	$1/n_{\rm free}$	Mutation rate of the large mutations at start of the run. Recommended to be se to $1/n_{\rm free}$ where $n_{\rm free}$ is the number of free parameters of the run, corresponding to on average one large mutation per model.

Table 3. Continuation of the parameters of `control.txt`. Table 2 lists the other parameters. The parameters regarding a variable mutation rate are described in the text.

Parameter	Default	Comment
p_value	0.05	Determines which models lie in the uncertainty region. Currently only relevant for post-processing, but may in the future be needed for assessing convergence.
clone_fraction	0.0	recommended to be set to 0. If > 0 this fraction of the new models is not picked by combining two parent models, but by just choosing the same parameters (for the model to be actually computed, a mutation then has to occur, because it is prevented that duplicate models are computed). The advantage of including closes is not obvious, therefore the recommended value for clone_fraction is 0.
w_guass_br	0.10	Sets the width (standard deviation) of the distribution (gaussian) that describes the sizes of the infrequent (large) mutations, expressed in a fraction of width of the parameter space (i.e. the difference between the maximum and minimum value for each parameter, as described parameter_space).
b_guass_br	0.10	Sets the baseline of the distribution of the infrequent (large) mutations.
w_guass_na	1.25	Sets the width (standard deviation) of the distribution (gaussian) that describes the sizes of the frequent (small) mutations, expressed in either a fraction of width of the parameter space, or an amount of steps, corresponding the step_size for each parameter as defined in parameter_space, which measure is used is specified by narrow_type.
b_guass_na	0.0	Sets the baseline of the distribution of the frequent (small) mutations.
narrow_type	step	Either step or frac, determines whether the width of the infrequent (small) mutation is set in step size or in width of the parameter space.
broad_type	frac	Either step or frac, determines whether the width of the frequent (large) mutation is set in step size or in width of the parameter space.
doublebroad	no	Instead of one gaussian centered at the current parameter value, the infrequent (large) mutation distribution consists of two gaussians symmetry around the current parameter value. Set to either yes or no.
mut_rate_na	0.5	Mutation rate of the small mutations. This concerns the chance that each parameter undergoes a mutation (i.e. a fraction of mut_rate_na of the parameters of a model undergoes this type of mutation), and because the mutations are small, this fraction should be high.
mut_adjust_type	constant	Choose if and if so how to adjust the mutation rate. Options are constant, charbonneau and autocharb. Recommended: constant. The extra settings for the other options are explained in Section 5.5.2.
use_string	no	Choose yes for using string mutations instead of the Gaussians (not recommended).
sigs_string	2	Only relevant when when use_string=yes. Relates to the number of digits that decodes one parameter. Do not change.
fracdouble_string	0.5	Only relevant when when use_string=yes. Relates to cross-over. Do not change.
be_verbose	True	Print some output during the run.

values. In the worst case this discrepancy is large and the peak of the χ^2 distribution for a given parameter lies at a different point than the peak in the $\mathcal{F}$ distribution: it that case, one has told GA to search for the best models in some place (using $\mathcal{F}$) whereas after exploration you define to be your best models to be somewhere else (using χ^2). Therefore, while technically in KIWI-GA it is possible to use $\mathcal{F}$ for exploration by setting `fitmeasure` to `fitness`, I strongly recommend to use `chi2`.

It should be noted that a third possibility would be to use $\mathcal{F}$ for both exploration and post-processing. In that case, you would get a consistent framework as is the case when using χ^2 for both aspects. However, to my knowledge, no such statistical framework exists at the moment, and furthermore, the other objections to using $\mathcal{F}$ as a goodness-of-fit measure that were pointed out before, are still there.

5.5.2 Variable mutation rate

Omitted from Table 3 are parameters that relate to a variable mutation rate. It is recommended to set `mut_adjust_type` to `constant`, as a variable mutation rate seems not to add much to the exploration, and varying it is at the cost of transparency about the process. Nonetheless we explain here the options for varying the mutation rate (of the large mutations only) during the run, and how they work.

In GA versions that employed only one mutation rate, the mutation rate was typically varying during the run, depending on a measure of variety of the population, defined as following:

$$\mathcal{V} = \frac{|\chi^2_{\rm best} - \chi^2_{\rm median}|}{\chi^2_{\rm best} + \chi^2_{\rm median}} \tag{5}$$

Then, if $\mathcal{V}$ for a given generation lies in a certain range of values between `fit_cutoff_min` and `fit_cutoff_max`, then the mutation rate stays constant. If $\mathcal{V}$ is lower than the defined minimum, then the mutation rate is be increased with a certain factor `mut_rate_factor`, and if $\mathcal{V}$ exceeds the defined maximum then the mutation rate is reduced by a `mut_rate_factor`. This scheme for altering mutation can still be used by setting `mut_adjust_type` to `charbonneau`. The idea of this method is that mutation is adjusted depending on variety in the population, where a diverse population would imply that the run is not yet near the right solution, and the mutation rate should be low. The downside is that the $\mathcal{V}$ is defined by χ^2 and not by the actual variety in the model parameters. The effect of this, is that for a given diversity in the model population, the ratio can differ depending on both $\rm SNR$ and stellar parameters. For example imagine the first generation of models of two different runs, one with high $\rm SNR$ and one with low $\rm SNR$. The parameter space of the runs is the same, meaning that the diversity in the first generation should be the same (in both cases: very high because it is completely random). Then in the high $\rm SNR$ case, the difference between the best fitting model and the median model χ^2 would be high, thus so would $\mathcal{V}$, and the mutation rate might be reduced. For a very low $\rm SNR$ stellar spectrum, many models will fit with a reasonably low χ^2 value, therefore the difference between the best fitting model and the median model χ^2 would be low, thus so would $\mathcal{V}$, and the mutation rate might be increased. The same holds for later in the run: runs with a lower $\rm SNR$ reach lower final values of $\mathcal{V}$ than runs with a higher $\rm SNR$.

To correct the mutation varying mechanism for the dependence on data, we introduced variable values for `fit_cutoff_min` and `fit_cutoff_max`, that are set based on the value of $\mathcal{V}$ after the first generation, $\mathcal{V}_{\rm gen1}$. $\mathcal{V}_{\rm gen1}$ turns out to be a good indicator as for the typical size of $\mathcal{V}$ in a run, which makes sense as this value represent the same initial diversity (completely random) for all runs. Based on $\mathcal{V}_{\rm gen1}$ the values of `fit_cutoff_min` and `fit_cutoff_max` are set automatically, and then the adjustment of the mutation rate works as explained above. This scheme for altering mutation can still be used by setting `mut_adjust_type` to `autocharb`. I won't expand on the detailed parameters of these setups here, if you are interested please ask me (s.a.brands@uva.nl).

5.6 Input file: `defaults_fastwind.txt`

The file `defaults_fastwind.txt` contains the default values for several FASTWIND parameters, as well as defaults that are related to KIWI-GA inputs, for example `logfclump`, which has a default value such that it

will be ignored by default. The values can be changed inside `defaults_fastwind.txt`, but for numerical values describing the star itself rather than the working of FASTWIND (such as `micro`), it is better practice to indicate them a s a fixed parameter inside `parameter_space.txt`, as to keep track of the values when setting up the parameter space.

```
...
enat_cor        T
expansion       F
set_first       1
set_step        2
fclump          1.0
logfclump       10.0
vclstart        0.15
vclmax          0.30
opthopf         F
do_iescat       0
windturb        0.1
fic             -1
hclump          1.0
fx              0.0
...
```

Listing 7. Part of a `defaults_fastwind.txt` file.

6 The script pre_run_check.py

The input format as described in the previous paragraphs is specific and vulnerable to mistakes. If not done correctly, KIWI-GA will crash, or give unwanted results and one should start a new run. The script pre_run_check.py is designed to catch mistakes before the run starts: it is strongly advised to check your input with this script and inspect the output, before staring a run.

1. Before running the script for the first time, make sure to adjust the paramter n_cpu_core at the top of the pre_run_check.py script to the amount of CPUs per core in the cluster on which KIWI-GA is used.

2. Run the following from the main KIWI-GA directory:
   `python pre_run_check.py run_name`

Here run_name should be the same as the directory with the input files (inside input/). Apart from checking the run input, the script also generates a job script that can be submitted to the cluster directly. This part of the pre_run_check.py script should be adapted by the user as to match the requirements of the specific cluster that KIWI-GA is executed on.

Note that while pre_run_check.py is designed to check the run input on all kind of inconsistencies, it is not guaranteed that it catches everything. If you would like to see something implemented or spotted a bug, please let me know (s.a.brands@uva.nl).

6.1 Terminal output

The script will output any issues to the terminal. In case of some major issues, the script will be aborted prematurely: fix the error and then run the script again. In other cases, the issues will be summarised at the end: please scroll up to check the details and solve the issues, before running the script again. If all issues are solved, move on to the pdf report (Section 6.2).

```
##############################################################################
##                                                                          ##
##   Summary of pre-run checks:                                             ##
##                                                                          ##
##     >  FW version: all ok.                                               ##
##     >  Mutation: all ok.                                                 ##
##     >  Line list: irregularities found --> CHECK INPUT!                  ##
##     >  Parameter space: all ok.                                          ##
##     >  Step size: all ok.                                                ##
##     >  FORMAL_INPUT: all ok.                                             ##
##     >  Reinsertion: all ok.                                              ##
##                                                                          ##
##############################################################################
```

Listing 8. Summary of pre-run checks: one issue is found (part of the output of pre_run_check.py).

6.2 Pre-run pdf-report

Apart from the terminal output, a pdf report will be generated and saved in the directory input/run_name. The pdf report consists of two or three pages. The first page shows a graphical overview of the free parameters, the choice of step size (each step is indicated with a blue vertical line), and the meta-parameters that control the mutation rate of the algorithm (see Figure 5, top row). If you accidentally have chosen a very small or large step size, this will be apparent in the plot. Furthermore you can see the width of the mutation distributions, and see how they compare to the total width of the parameter space.

The second page shows the parts of the spectrum that will be fitted, per line (see Figure 5, bottom row). Indicated are line the line boundaries used for the fit. The color of each plot turns read if not enough data points are available to compute a $\chi^2_{\rm red}$ value for an individual line. Furthermore, weight and resolution of the

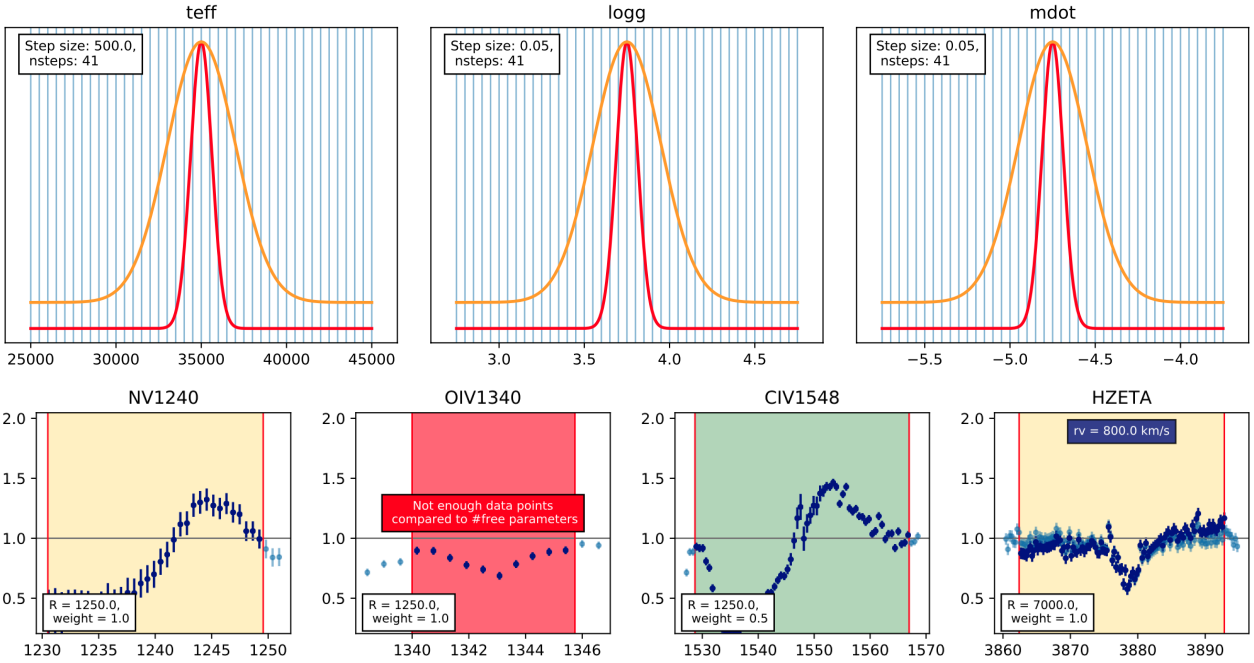


Figure 5. Top row: Part of page 1 of the pre-run report: graphical depiction of the parameter space. In blue, the step size is indicated, in red the 'narrow mutation' distribution (i.e. the one that will occur very frequently), in yellow the 'broad mutation' distribution (i.e. the one that will occur with a lower frequency). Bottom row: Part of page 2 of the pre-run report: lines/spectral regions to be fitted. By default (if all is well) lines or spectral regions will have a light yellow background. If something is wrong, this will turn red. If the weight of a line is altered, the background will be green, and if the radial velocity is corrected for inside the run, a blue note will show up. The renormalisation can be seen by the difference of the dark blue points (to be fitted) and the light blue points (original). In this case, the renormalisation is clearly worse than the original normalisation, and should not be applied during the run.

line are shown. If the weight is not equal to one, a signalling color is used. For convenience, the very wide spectral regions (broader than 100 Å) are plotted, again, on a third page. Different from page two is that there is a full row per spectral region, so the spectra are essentially repeated in a larger plot.

6.3 Job script

When both the terminal output and the pdf report show no issues, you can start the run by submitting the job script to the cluster. For a Slurm job scheduler: `sbatch jobscript.job`. A job script is created by `pre_run_check.py`, however please note that:

- This job script is optimised for the Surfsara Cartesius cluster, that uses the Slurm job scheduler.
- The user should alter the parts of `pre_run_check.py` that create the job script, as different clusters/users will require different job scripts.
- For users on Surfsara Cartesius, changing the parameter `username` at the top of `pre_run_check.py` to your own username might suffice, but others should also other adjust other aspects, such as paths to the relevant directories when the run is being executed, as well as loading relevant packages specific to the cluster. If the cluster is equipped with another job scheduler than Slurm, the top part of the job script should be adjusted.
- Make sure that your job script is such, that the amount of CPUs used for a job is one less than the amount of models in one generation.
- The maximum wall time is currently set at a fixed value for all jobs as a parameter in `pre_run_check.py`.
- What should be done in a job script is essentially starting a run, steps for which are listed in Section 4.2.

An example of a job script is shown below.

```
#!/bin/bash
#SBATCH --job-name=H35_UVopt_2012aa
#SBATCH -t 72:00:00
#SBATCH -N 8
#SBATCH -n 192
#SBATCH --no-requeue

runname=H35_UVopt_2012x
do_restart=no
ncpu=192
inidir=v10_HHeNCOPSi

echo Run ${runname}
echo Using $ncpu CPUs
echo Do restart? $do_restart

# Load modules
module load 2019
module load Python/3.6.6-intel-2018b

# Define paths
scratch=/scratch-shared/sbrands/${runname}/
homedir=/home/sbrands/Kiwi-GA/

echo Copying files

# Create and copy directories and files
mkdir -p $scratch
cp -r ${homedir}*.py $scratch
cp -r ${homedir}filter_transmissions $scratch
mkdir -p ${scratch}input/
mkdir -p ${scratch}input/${runname}/
cp -r ${homedir}input/${runname}/* ${scratch}input/${runname}/.
cp -r ${homedir}${inidir} $scratch

# Navigate to computation directory
cd $scratch

echo Starting run!
date

# Start run
if [ "$do_restart" == "yes" ]
then
    echo ...restarting run
    srun -n $ncpu python3 kiwiGA.py ${runname} -c
else
    echo ... creating output dir
    mkdir -p output
    echo ... starting run
    srun -n $ncpu python3 kiwiGA.py ${runname}
fi

date
echo ... Run ENDED!
```

Listing 9. A `run_kiwiGA.txt` file.

7 Output

The output of a run of 191 individuals that iterates for 60 generations is about 270 MB. The main output of KIWI-GA is stored in `chi2.txt` (∼7 MB), which contains all important information per computed model (Section 7.1). The bulk of the disk space is occupied by line profiles, that are stored for all computed models (Section 7.2). The output can be plotted and summarised using the script `GA_analysis.py` (Section 7.3).

7.1 The main output file: `chi2.txt`

All output of a KIWI-GA run is saved inside `chi2.txt`, with the exception are the line profiles (Section 7.2). The `chi2.txt` file contains a complete list of computed models, and lists for each model:

- Model identifiers:
  - `run_id`: ID denoting the generation and the model within that generation (e.g. 0000_0187, denoting the first generation, model number 187)
  - `gen`: generation (e.g. 0000)
- The fitness measures (all are computed and saved, regardless of which is used for the exploration):
  - `chi2`: Chi-quared value χ^2 (set to 999999999 if a model is failed).
  - `rchi2`: Reduced chi-quared value $\chi^2_{\rm red}$ (set to 999999999 if a model is failed).
  - `dof`: Degrees of freedom (set to -1 if a model is failed).
  - `fitness`: Fitness $\mathcal{F}$ (set to 0 if a model is failed).
- Meta information of the model computation: .
  - `maxit`: Amount of iterations of the FASTWIND model.
  - `maxcorr`: Maximum correction to level populations at the last iteration.
  - `cputime`: CPU-time needed to complete the model (set to 99999.9 if a model was cut-off before finishing. In that case, the computation time of the model is in fact equal to `fw_timeout` as set in `control.txt`).
- The model parameters used as input. Free parameters only, one column each, in the order in which they appeared in `parameter_space.txt`, and units as specified in Section 5.2.2.
- Derived parameters: model parameters that are not directly free parameters, but that can be derived from the input parameters, or can be derived from the model output after completion of the computation:
  - `radius`: in $\rm R_\odot$ (saved in all cases, so also if the radius was fixed throughout the run).
  - `xlum`: X-ray luminosity of the model, expressed in units of $L_{\rm bol}$.
  - H-I, He-I and He-II ionising fluxes: `logq0`, `logQ0`, `logq1`, `logQ1`, `logq2`, `logQ2`, where the small `logq0`, `logq1`, `logq2` denote logarithm of the flux per unit surface area capable of ionising H-I, He-I and He-II, respectively, and `logQ1`, `logq2`, `logQ2` those same fluxes, but integrated over the stellar surface (all units in cgs).
- $\chi^2_{\rm red}$ per individual line. One line each, in the order in the order in which they appeared in `line_list.txt`.

7.2 Models

The convolved line profiles of all computed models are saved inside `output/saved/xxxx/xxxx_mmmm.tar.gz`, where `xxxx` denotes the generation, and `mmmm` is the model identifier. Each line profile is stored in a file named LINENAME.prof.fin, containing two columns: one for the wavelength in Å, one for the normalised flux.

For each non-CMF line 150 data points are stored, resulting in a file of ∼3.2K (zipped this is about 1.2K), so that a model with for example 18 spectral lines results in a tar.gz of 23K, and if the KIWI-GA run would

consist of 191 individuals and 60 generations, the total size of the run would be about 270MB (also counting the `chi2.txt` file).

Apart from the line profiles the following is contained inside `xxxx_mmmm.tar.gz`:

- `INDAT.DAT`
- `formal.in`. This file contains three lines, that in turn contain the following:
  1. Model identifier.
  2. Micro turbulent velocity, turbulent velocity of the wind.
  3. Whether to include electron scattering (0 = no, 1 = yes).
- `broad.in`. This file contains two lines, that in turn contain the following:
  1. Rotational broadening (-1 = no rotational broadening included)
  2. Macro turbulent broadening (-1 = no macro turbulent broadening included)

7.3 The script `GA_analysis.py`

For analysing and summarising the content of Section 7.1 one can use the script `GA_analysis.py`. I'm not particularly proud on the way it is programmed, and hope to rewrite it soon. Furthermore note that this script is only tested with Python 2.7.

Run the script as follows:

1. Set the right paths inside `paths.py`:
   - `outpath_analysis`: Give a path to where the output of `GA_analysis.py` is stored. Inside this directory, `GA_analysis.py` automatically creates a directory with the name of the run. Inside this sub directory the output pdfs will be stored.
   - `datapath_analysis`: Give path to where the KIWI-GA output is stored. In this directory, each run should have its own folder named after the run name. Inside each run-directory, `chi.txt` and other files should be present, as well as a directory with the line profiles called `saved/`
2. Run `GA_analysis.py` from the terminal as following: `python GA_analysis.py <run name> -prof`.
   - The argument `-prof` is optional. By adding it, line profiles will be generated. You would always want this for creating a final report, but for checking run progress omitting it will save some time.
   - The argument `-set` specifies which report to make, choose between `short` (default), `long` (for final reports, create all plots possible, takes longer) or `manual` (create a subset of the plots, you can set these inside the script, search for "`if full_short_manual in ('manual'):`").
   - Run `python GA_analysis.py -h` for an overview.

The output of the script consists of:

- A pdf report. Two versions are compiled `short_report_runname.pdf` and `full_report_runname.pdf`, where the latter will be more extensive than the former, but only when `GA_analysis.py` has been run with the option `-set full`. Otherwise `full_report_runname.pdf` will still be created but will not be complete. I always open `full_report_runname.pdf` and use `short_report_runname.pdf` only to send to people. In Figure 6 and 7) examples are shown, with notes to each plot in the caption.
- A file `best_fit_parameters.txt` that summarises the best fitting parameters and the error regions.
- The best fitting line profiles plus the error regions. These can be handy for manually creating some plots afterwards.

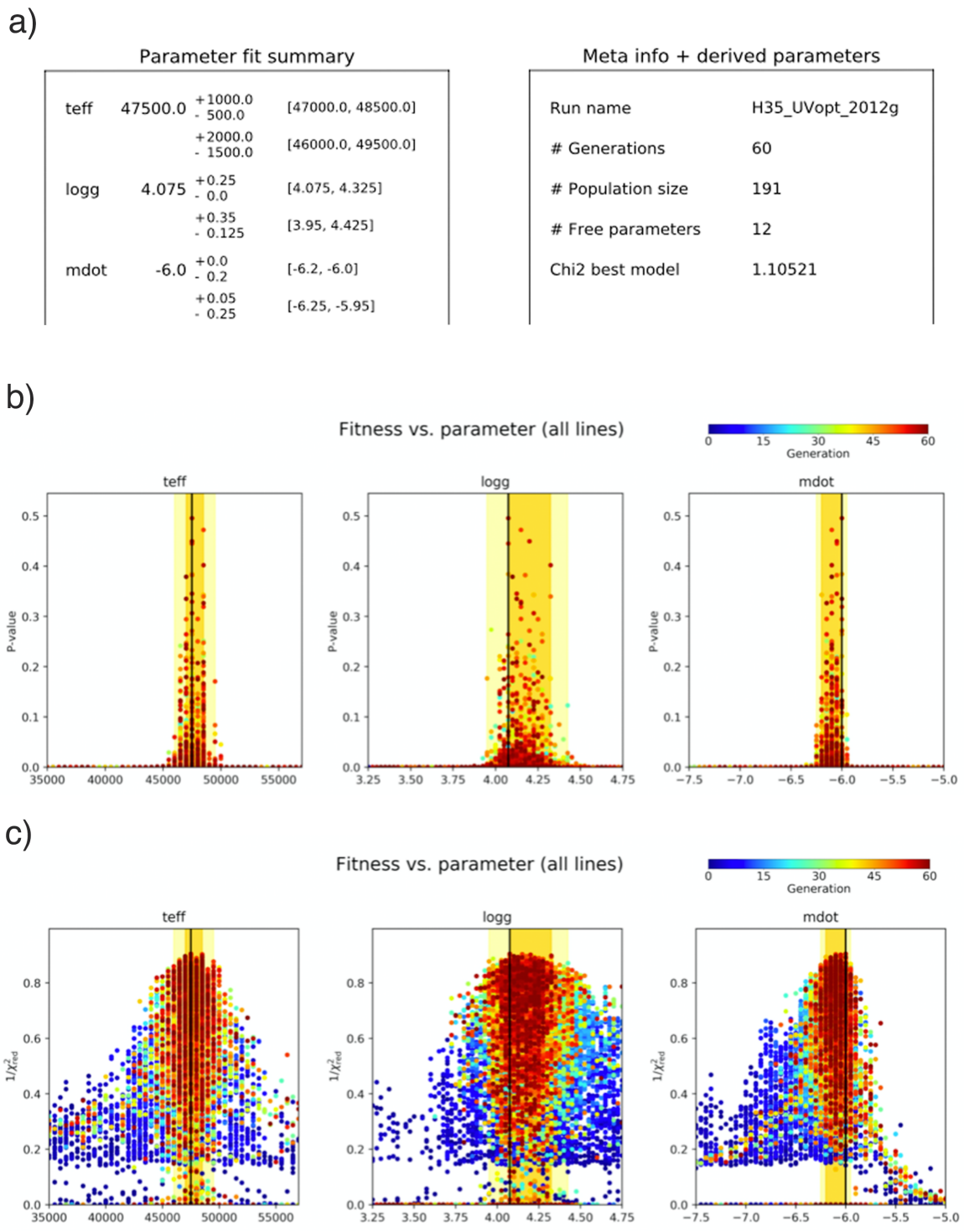


Figure 6. Example of the pdf output of `GA_analysis.py`: *a)* Run overview. Lists in the left column the values of the best fit parameters and their 1 and 2σ error bars. The values in brackets show the 1 and 2σ ranges. On the right, some meta information is shown, and at the bottom (not shown in the figure) the best fit values and error bars of derived parameters. *b)* Each plot shows the P-value of all models as a function of a parameter. Color of the dots denote of which generation the model was part. Dark and light shaded yellow regions denote the 1 and 2σ ranges. *c)* As b), but $1/\chi^2_{\rm red}$ instead of P-value.

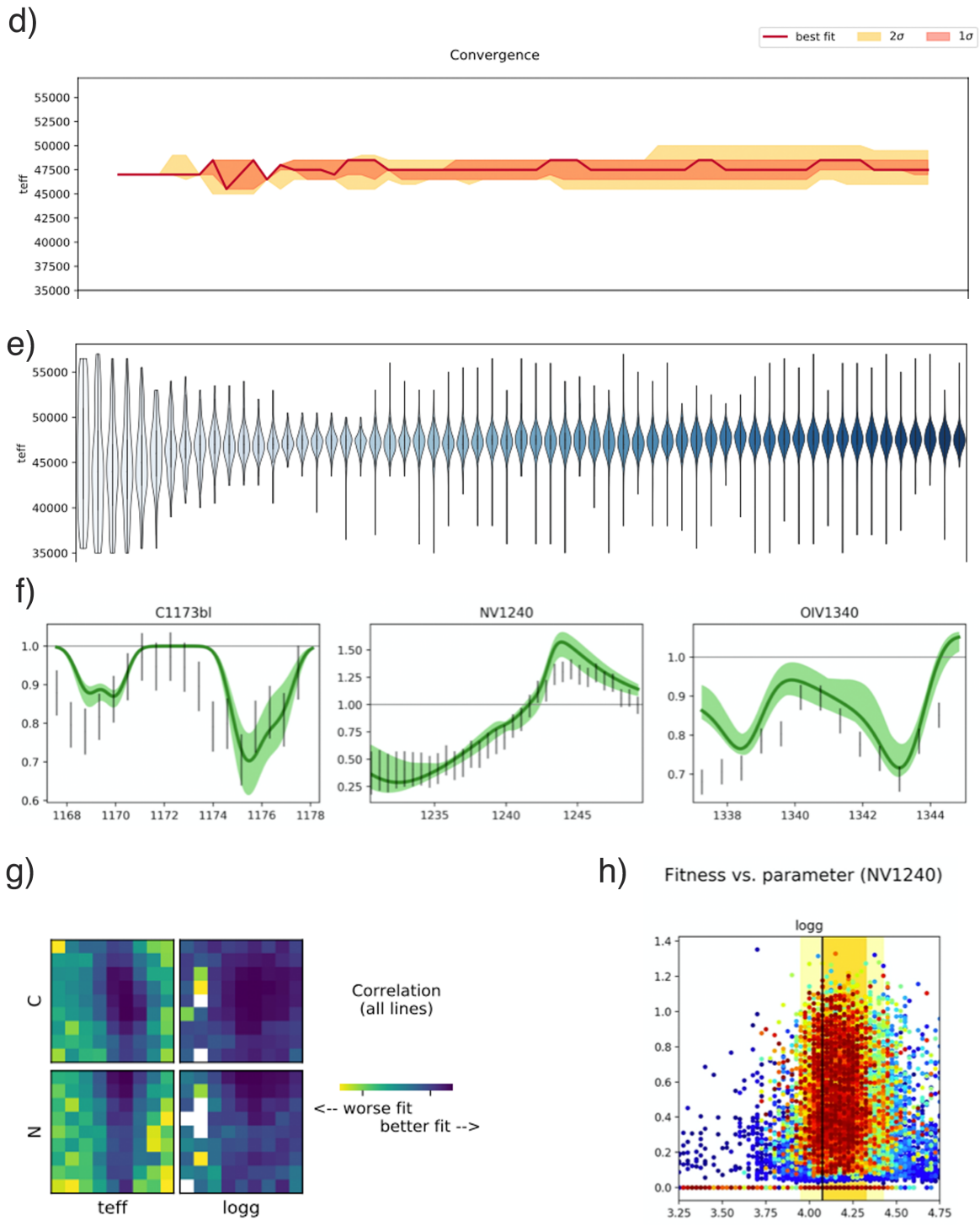


Figure 7. Example of the pdf output of `GA_analysis.py` continued from Figure 6. *d* Shows the best fit values and confidence regions for each parameter as a function of generation, in other words: the convergence of the run. *e* Shows for each parameter how many models with a certain value where computed in each generation. *f* Shows the data and the best fit model (dark green line) and 2σ error regions (light green shaded region). *g* Shows correlation between parameters. *h* As c) in Figure 6, but now the $1/\chi^2_{\rm red}$ for each individual line instead of that of the complete model. These line-by-line fitness plots can tell you which line is sensitive to what, but note that they can be misleading as they do not just reflect fitness of the line as a function of parameter: what you see is (strongly) biased towards where KIWI-GA explored based on the other lines.

8 Bugs

8.1 Reporting bugs

Please report bugs to s.a.brands@uva.nl.

8.2 Known bugs

Faulty header of chi2.txt *Encountered randomly in about 2 runs out of 100.*
Occasionally something goes wring in the order of writing the first lines of the `chi2.txt` file: the header is written to the second line, and the first model to the first line. This occurs probably when two CPUs write at the same time to the `chi2.txt` files, and the CPU that was supposed to go first (writing the header and the first model) is somehow slower than the other CPU. When this happens, after the run has finished, manually switch the two lines in the output file before running `GA_analysis.py`.

Nan value in chi2.txt *Encountered randomly in less then 1 runs out of 100.*
Model with a χ^2 value of 'nan' is written to `chi2.txt`, which results in an error in `GA_analysis.py`. I was not able to find out why this did happen (all models for which a χ^2 cannot be computed for whatever reason are supposed to return $\chi^2 = 999999999$), and in the end removed the line from the `chi2.txt` file.

Run hangs forever instead of stopping *Encountered when input files are inadequate.*
When an error occurs in the execution of python code handling one specific model, the particular CPU gets stuck, but this is not communicated to the main node that handles communication between nodes. The effect is that all CPUs will be waiting for their next instruction, but that instruction will never come. The run then just 'hangs' there forever, spending cluster CPU time while nothing is actually done. This can happen when user input is not provided in the right format. Therefore, always run `pre_run_check.py` on your input files and carefully inspect the output thereof, before starting a run. A run can also hang if there would be an actual error (or typo) in the KIWI-GA code. If you have the feeling this is the case, please inform me (s.a.brands@uva.nl).