TRADES

`TRADES` **v2.9.1 by Luca Borsato - 2016**

Most of the information can be found in the paper by Borsato et al. (2014) and at the webpage TRADES@ESPG.
Feel free to use or modify the code, but please cite Borsato et al. (2014).
Comments are welcome!
TRADES@github

WARNING:
only tested on a Unix/Linux machine (i.e., Centos Rocks 5.3, Ubuntu > 12.04 and derivatives)
Please use `gfortran` **version greater than** `4.8.1` **. Previous versions could fail to compile.**
`MPI/open-MPI` **required in order to compile the PolyChord library. In the future this will be fixed.**
`openMP` **required to compile parallel versions** `TRADES`

TRADES user guide

Introduction

We have developed a computer program (in Fortran 90, openMP, and MPI) for determining the possible physical and dynamical configurations of extra-solar planetary systems from observational data, known as `TRADES`, which stands for TRAnsits and Dynamics of Exoplanetary Systems.
The program `TRADES` models the dynamics of multiple planet systems and reproduces the observed transit times (T_0 , or mid-transit times) and radial velocities (RVs). These T_0s and RVs are computed during the integration of the planetary orbits.
Hereafter, I will use the reduced chi-squared ($\chi^2_\mathrm{r} = \chi^2/\mathrm{dof}$) as the fitness parameter, but it would be also possible to use a $\chi^2_{\mathrm{w,dof}}$ weighted/scaled by the number of each type of data set: transits and radial velocities.
We have developed `TRADES` from zero because we want to avoid black-box programs, it would be easier to parallelize it with openMP, and include additional algorithms.
To solve the inverse problem, `TRADES` can be run in different modes:

- **integration:**
  it runs a simple integration of the orbits of the planetary system calculating the T_0s and the RVs.

- **grid search:**
  TRADES samples the orbital elements of one perturbing body (all the parameters are allowed in the grid, but remember that the number of simulations will increase hugely). The grid parameters can be evenly sampled on a fixed grid by setting the number of steps, or the step size, or by a number of points chosen randomly within the parameter bounds. For any given set of values, the orbits are integrated, and the residuals between the observed and computed T_0s and RVs are computed. For each combination of the parameters, the `LM` algorithm can be called and the best case is the one with the lowest residuals (lowest reduced chi-squared = $\chi^2/\mathrm{dof} = \chi^2_\mathrm{r}$).

- **Levenberg-Marquardt (** `LM` **,** `lmdif` **from** `MINPACK` **) algorithm:**
  After an initial guess on the orbital parameters of the perturber, which could be provided by the previously described grid approach, the LM algorithm exploits the Levenberg-Marquardt minimization method to find the solution with the lowest residuals. The LM algorithm requires the analytic derivative of the model with respect to the parameters to be fitted. Since the T_0s are determined by an iterative method and the radial velocities are

computed using the numerical integrator, we cannot express these as analytic functions of fitting parameters. We have adopted the method described in Moré et al. (1980) to compute the Jacobian matrix, which is determined by a forward-difference approximation. The `epsfcn` parameter, which is the parameter that determines the first Jacobian matrix, is automatically selected in a logarithmic range from the machine precision up to 10^{-6} ; the best value is the one that returns the lower χ^2_r . This method has the advantage to be scale invariant, but it assumes that each parameter is varied by the same `epsfcn` value (e.g., a variation of 10% of the period has a different effect than a variation of the same percentage of the argument of pericenter).

- **genetic algorithm (`GA` or `PIK` , we used the implementation named `PIKAIA` , Charbonneau 1995):**
  the `GA` mode searches for the best orbit by performing a genetic optimization (e.g. Holland 1975; Goldberg 1989), where the fitness parameter is set to the inverse of the χ^2_r . This algorithm is inspired by natural selection which is the biological process of evolution. Each generation is a new population of *offspring* orbital configurations, that are the result of *parent* pairs of orbital configurations that are ranked following the fitness parameter. A drawback of the `GA` is the slowness of the algorithm, when compared to other optimizers. However, the `GA` should converge to a global solution (if it exists) after the appropriate number of iterations.
  At the moment if the `GA` after the predefine iteration has a fitness (or χ^2_r) higher than 1000, it will continue (it stops if it reaches the iteration step with fitness lower than 1000 or if it reaches fitness <= 1.).

- **particle swarm optimization (`PSO` , Tada 2007):**
  the `PSO` is another optimization algorithm that searches for the global solution of the problem; this approach is inspired by the social behavior of bird flock and fish school (e.g., Kennedy & Eberhart 1995; Eberhart 2007). The fitness parameter used is the same as the `GA` , the inverse of the χ^2_r . For each *particle*, the next step (or iteration) in the space of the fitted parameters is mainly given by the combination of three terms: random walk, best *particle* position (combination of parameters), and best *global* position (best orbital configuration of the all particles and all iterations).

- **PolyChord (`PC` , Handley et al., 2015):**
  PolyChord is a novel nested sampling algorithm tailored for high dimensional parameter spaces. In addition, it can fully exploit a hierarchy of parameter speeds such as is found in CosmoMC and CAMB. It utilises slice sampling at each iteration to sample within the hard likelihood constraint of nested sampling. It can identify and evolve separate modes of a posterior semi-independently and is parallelised using openMPI. PolyChord is available for download at PolyChord-CCPForge

In each mode, `TRADES` compares observed transit times ($T_{0,\rm obs}$) and radial velocities (RV $_{\rm obs}$) with the simulated ones ($T_{0,\rm sim}$ and RV $_{\rm sim}$). From version 1.1.2 of `TRADES` , it is possible to use different set of RV, with different RV offset (the so-called *gamma* point); `TRADES` will compute a γ for each RV data set.
The *grid* search is a good approach in case that we want to explore a limited subset of the parameter space or if we want to analyze the behavior of the system by varying some parameters, for example to test the effects of a growing mass for the perturbing planet.
`GA` and `PSO` are good methods to be used in case of a wider space of parameters. The orbital solution determined with the `GA` or the `PSO` method is eventually refined with the `LM` mode.
`PC` is well described in the paper by Handle et al. (2015), and it uses a Bayesian approach with the nested sampling. It works also on parameter bounds, but it would be better to limit the boundaries, and not used them as wide as those for `GA` and `PSO` .
For each mode, but `PC` , `TRADES` can perform a **bootstrap** analysis to calculate the interval of confidence of the best-fit parameter set. We generate a set of T_0s and RVs from the fitted parameters, and we add a Gaussian noise having the calculated value (of T_0s and RVs) as the mean and the corresponding measurement error as variance, scaled by the $\sqrt{\chi^2_{\rm reduced}}$. We fit each new set of observables with the `LM` with default options. We iterate the whole process thousands of times to analyze the distribution for each fitted parameter.

For the mathematical and computational description see Borsato et al. (2014).

Install and Compile

1. `TRADES` source is available at github.com/lucaborsato/trades.
   Download the .zip file or clone the repository (see github help).

2. Extract the .zip file in your drive or enter the cloned repository. It should contain a `README.md` file, `bin/`, `src/`, and `pytrades/` folders.
   The `src/` folder should countains the following f90 source files:

   - **Module source files:** `parameters.f90` `random_trades.f90` `convert_type.f90` `lin_fit.f90` `celestial_mechanics.f90` `init_trades.f90` `statistics.f90` `timing.f90` `rotations.f90` `sort.f90` `gls_module.f90` `eq_motion.f90` `output_files.f90` `numerical_integrator.f90` `radial_velocities.f90` `transits.f90` `ode_run.f90` `derived_parameters_mod.f90` `fitness_module.f90` `grid_search.f90` `lm.f90` `pikaia.f90` `util_sort.f90` `util_qmc.f90` `opti_pso.f90` `gaussian.f90` `bootstrap.f90` `PolyChord_driver.f90` `driver.f90`
   - **PolyChord folder** `PolyChord/` with source files: `utils.f90` `abort.F90` `settings.f90` `params.f90` `array_utils.f90` `priors.f90` `mpi_utils.F90` `calculate.f90` `random_utils.F90` `chordal_sampling.f90` `run_time_info.f90` `clustering.f90` `read_write.f90` `feedback.f90` `generate.F90` `ini.f90` `nested_sampling.F90`
   - **Old main** `TRADES` **file (DO NOT USED IT, to be converted into a library asap):** `trades.f90`
   - **simple integration+** `LM` **+bootstrap main:** `trades_int_lm_bootstrap.f90`
   - **simple grid main:** `trades_grid.f90`
   - **simple** `PIK` **and** `PSO` **main:** `trades_pik_pso.f90`
   - **simple** `PC` **main:** `trades_polychord.f90`
   - **Makefile:** `Makefile`
   - **python script to create example simulation folder:** `createSimFile.py`
     Look at section 6. for the `pytrades/` folder (install and execution).

3. Edit the `Makefile` with your Fortran 90 - MPI compiler, and with the needed compiling options.

   - flag `CC` for the compiler to use. From the implementation of `PC` the `mpif90` must be used;
   - flag `CFLAGS` for the compiler options;
     from the implementation of `PC` the `-ccp` preprocessor option must be used
     add options or uncomment following rows for other debugging options
   - flag `COPT` for the compiler optimization
     from the implementation of `PC` the `-ccp` preprocessor option must be used
   - flag `CFLAGS2` for the opemMP version
   - flag `TARGET_SER_X` is relative path and executable name for the serial program
   - flag `TARGET_OMP_X` is relative path and executable name for the `openMP` parallel program
   - flag `TARGET_MPIOMP` is relative path and executable name for the `MPI+openMP` parallel program (it compiles only old `trades.f90`)
   - flag `TARGET_PC` is relative path and executable name for the `PC` program

4. To compile:

   - serial-debug mode, type: `make serial_debug`
     It creates a executable file `trades_s_xxx` and `trades_polychord` in the `bin/` folder.
   - serial-release mode, type: `make serial_release`
     It creates a executable file `trades_s_xxx` and `trades_polychord` in the `bin/` folder.
   - openMP-debug mode, type: `make omp_debug`
     It creates a executable file `trades_o_xxx` and `trades_polychord` in the `bin/` folder.
   - openMP-release mode, type: `make omp_release`
     It creates a executable file `trades_o_xxx` and `trades_polychord` in the `bin/` folder.
   - openMP-MPI-debug mode, type: `make mpi_omp_debug`
     It creates a executable file `trades_mo` and `trades_polychord` in the `bin/` folder.

- openMP-MPI-release mode, type: `make mpi_omp_release`
  It creates a executable file `trades_mo` and `trades_polychord` in the `bin/` folder.

Remember to type `make clean` **to remove** `*.o` **and** `*.mod` **files before re-compiling** `TRADES` **.**
Remember to type `make clean_libchord` **to remove PolyChord files and library.**
Remember to type `make cleanall` **to remove** `*.o` **,** `*.mod` **, and all the executable files before re-compiling** `TRADES` **.**
To compile in parallel mode the `openMP` **libraries must be properly installed (as suggested by your Linux distribution) and the** `MPI` **(** `Open-MPI` **) libraries and compilers.**
2016-01-28 WARNING:
If the `MPI` compiler is different from the compiler used for the module `mpi.mod` , the program will fail to compile. Please, change properly your `Fortran-MPI` in the `Makefile` .

How to run TRADES

Different ways to launch TRADES:

- export the path of the executable (`trades_s` , `trades_o` , and `trades_mo`) in your ~/.bahsrc o ~/.profile:
  > export PATH=$PATH:/path/to/trades/executables

- it is possible to execute trades from the `bin/` folder by typing:
  ```
  ./trades_s_xxx
  ./trades_o_xxx
  ./trades_polychord
  ./trades_mo
  ```

WARNING:
Before running TRADES in parallel with `open-MP` (`trades_o_xxx`) remember to set the number of cpus (`Ncpu`) to use by exporting:
`OMP_NUM_THREADS=Ncpu`
`export OMP_NUM_THREADS`
Put this in a script or type it in a terminal; in short way:
`export OMP_NUM_THREADS=Ncpu`
In order to use trades with `MPI+openMP` remember to specify the `OMP_NUM_THREADS` and the MPI processes `mpiexec -n N_mpi_process trades_mo ...`

If TRADES has been launched without any arguments, it will search for the needed files in the current folder:
e.g.:

```
cd /home/user/Simulation/
trades_s_xxx
```

it is equal to type:

```
cd /home/user/Simulation/
trades_s_xxx .
```

or:

```
cd /home/user/
trades_s_xxx /home/user/Simulation/
```

In any of these three cases, `TRADES` will write the output files in the folder `/home/user/Simulation/`

Provide orbital elements (and boundaries) in the bodies file.

Algorithm selection in `arg.in` :
`progtype` :

1. grid
2. integration + `LM` + bootstrap
3. `PIKAIA` (+ `LM` + bootstrap)
4. `PSO` (+ `LM` + bootstrap)
5. PolyChord

Files needed by TRADES

In the `src/` folder you can find the `createSimFile.py` python script that allows to create all the files needed by `TRADES` . The files are based on Kepler-9 system, with the original data (T_0 and RV) from the discovery paper by Holman et al. (2010).

List of the files with explanation:
`arg.in bodies.lst star.dat b.dat c.dat lm.opt pikaia.opt pso.opt PolyChord.opt obsRV.dat NB2_observations.dat NB3_observations.dat`

1. `arg.in` : file with program arguments, needed for the integration, fitting type, output files.
   Example file `arg.in` .

2. `bodies.lst` : file with list of the files with the parameters for each body.
   The first column is always the file name of the body, followed by `0` or `1` for each parameter. `0` means do not fit it, `1` means fit it.
   The first row is always the star file with the Mass and Radius fitting parameter type.
   From the second row, each line is the body file name followed by the parameters to fit in this order:
   `mass radius period eccentricity argument_of_pericenter mean_anomaly inclination longitude_of_node` .
   Remember that the number of the lines of this file has to match the `NB` parameter in the `arg.in` file.
   Example file `bodies.lst`

   - `star.dat` : Mass (and sigma) and Radius (and sigma) of the star in Solar units. First row the Mass, second the Radius.
     In the code will be identified with the id == 1.
     Example file `star.dat`
   - `b.dat` : file with parameters of the planet in the second row of the bodies.lst, that is in the code will be identified with the id == 2.
     Each row is a different parameter (3 columns + flag for `grid`), in the order:
     ```
     mass min max [M_Jup]
     radius min max [R_Jup]
     period min max [days]
     semi-major_axis min max [au]
     eccentricity min max
     argument_of_pericenter min max [deg]
     mean_anomaly min max [deg]
     time_of_pericenter_passage min max [JD]
     inclination min max [deg]
     longitude_of_node min max [deg]
     ```
     When not specified the `min max` values are set by default (i.e. eccentricity < 1, angles between 0 and 360 deg, inclination between 0 and 180 deg, radius < 5 `R_Jup` , and mass < 1 `M_Sun` in 'M_Jup . `In the` grid `mode the mean of the columns is:` min max step type . `The` step `is used as true step, or step number, and so on accordingly to the fourth column` type . `Keywords for column 4 are:` ss `(means step size),` rn `(random number), and` sn `(step number). Alternatives:` semi-major axis `in au instead of` period `(set` period `greater than` 9000000. `, while setting semi-major axis`

equal to 999. will let you use the period); time of passage at pericentre is used if mean anomaly is greater than 999. , while set time of passage at pericentre greater than 9.e8 to use mean anomaly . Example file [b.dat`](trades_example/b.dat)

- c.dat : same as file b.dat , but with different parameter values for the body in the third row in bodies.lst , that is in the code will be identified with the id == 3.
  Example file c.dat

3. lm.opt : parameter options for the Levenberg-Marquardt algorithm; they are based on the original manual. Keep it as it for standard analysis.
   Example file lm.opt

4. pikaia.opt : parameter options for the GA algorithm. The most important parameters to tune are
   (1pik) the number of individuals (row 1, ctrl(1)), that is the number of set of parameters for each generation;
   (2pik) the number of generation (row 2, ctrl(2)), that is the number of iteration that GA has to perform, the last iteration returns the best set of parameters;
   (3pik) the seed (row 13), that is a integer number that defines the seed for the random generator, if you keep the same value it repeat the same analysis;
   (4pik) the wrtAll (row 14) is a parameter that defines if you want to that the GA writes all the individuals for each generation, set it to 1 to write, 0 not write;
   (5pik) the nGlobal (row 15) is the number of global search to perform with the GA , each search returns a solution, and the seed of each analysis is different (seed + i, i=1..nGlobal).
   Example file pikaia.opt

5. pso.opt : parameter options for the PSO algorithm. The rows 1, 2, 4, 5, and 6 are the same parameters explained for the pikaia.opt file.
   In particular:
   (1pso) row 1 and (2pik) 2 are exactly the same as in pikaia.opt ;
   (3pso) row 3 is an integer that specifies when write a summary during the PSO analysis;
   (4pso) row 4 is the same as (4pik) row 14 in pikaia.opt ;
   (5pso) row 5 is the same as (5pik) row 15 in pikaia.opt ;
   (6pso) row 6 is the same as (3pik) row 13 in pikaia.opt .
   Example file pso.opt

6. PolyChord.opt : parameter options for the PC algorithm. It is quite well self explained; for further information check the PolyChord webpage.
   Example file PolyChord.opt

7. obsRV.dat : list of radial velocities (RVs) data.
   Columns description:
   (1) RV observation time (JD, or time in same units of the integration time);
   (2) observed RVs in meter per seconds;
   (3) observed RV uncertainties in meter per seconds;
   (4) ID of the RV dataset, so if you have only one dataset set all column to 1 , else increas value untill the number of different datasets, i.e., 1 , 2 for 2 datasets (2 different facilities, or one facility before and after upgrade).
   Example file obsRV.dat

8. NB2_observations.dat : list of transit times (T_0s) observed for planet in the second row of bodies.lst , i.e., b.dat is planet 2.
   Columns description:
   (1) transit epoch (N), an integer number that identifies the transit w.r.t. a reference transit time $T_{\rm ref}$ and refined by a linear ephemeris of kind: $T_N = T_{\rm ref} + P_{\rm ref} \times N$;
   (2) the transit time (T_0) in JD or the same time unit of the integration/epoch/start time;
   (3) the uncertainty on the T_0 . Example file NB2_observations.dat

9. NB3_observations.dat : same kind of file NB2_observations.dat , but for the planet in the third row of

`bodies.lst`, i.e., `c.dat` is planet 3.
Example file `NB3_observations.dat`

10. `derived_boundaries.dat` : this file a special file.
    If you have some derived parameters (or other values) that can reduce your parameter space you have to create this file in your simulation folder.
    if the argument `secondary_parameters` in `arg.in` file is set to `0`, this file will not be used.
    If `secondary_parameters = 1`, but the file does not exist it will not be used (no derived parameters will be checked).
    The file should have a line for each parameter, the name in the first column (please keep it short), the min and the max value in the 2nd and 3rd column. Keep last line empty so the code can determine the end of file.
    `derived_boundaries.dat` example:

```
# name phase_min phase_max
ph2 34. 180.
ph3 180. 270.
```

    In order to use the derived parameters you have to modify by your own the `derived_parameters_mod.f90`. In fitness_module.f90 check which subroutines/fuctions will be called to 'check_derived' or 'fix_derived' parameters.

Output files by TRADES

Each algorithm will write different files, and depends on the flag used in the `arg.in` and in the `*.opt` files.
Each file *should* have an self-explaning header.

1. **integration:** depends only on `arg.in` file
   `#ID_#LM_rotorbit.dat`, `#ID_#LM_constants.dat`, `#ID_#LM_NB#_elements.dat`, `#ID_#LM_NB#_tra.dat`, `#ID_#LM_gls_output.dat`
   `#ID_#LM_rotorbit.dat`, if `wrtorb = 1`, where `#ID` is the simulation ID, `#LM` is the Levenberg-Marquardt flag (`lmon = 0` or `1`). Columns: 1 Time in JD; 2 Light-Time Travel Effect in days (LTE_d); 3:3+NB*6 {X,Y,Z,VX,VY,VZ} for each body (NB=number of bodies); last column is the radial velocity (RV) of the star due to the planets in m/s.
   `#ID_#LM_constants.dat`, if `wrtcon = 1`, naming convenction as previous. Columns: 1 Time in JD; 2 momentum; 3 delta between initial and current momentum; 4 Total Energy; 5 delta between initial and current Total Energy.
   `#ID_#LM_gls_output.dat`, it is the output of the General Lomb-Scargle (GLS) applied to the residuals: $\mathrm{res} = \mathrm{RV}_{\mathrm{obs}} - \mathrm{RV}_{\mathrm{sim}}$.
   `#ID_#LM_NB#_elements.dat`, if `wrtel = 1`, naming convenction as previous, plus the body id NB#, starting from 2 to the number of bodies used. Columns: 1 Time in JD; 2 Period in days, 3 semi-major axis in astronomical unit (au), 3 eccentricity, 4 inclination in degrees, 4 mean anomaly in degrees, 5 argument of the pericenter in degrees, 6 longitude of the node in degrees, 7 true anomaly in degrees, 8 difference between time of refence (epoch) and time of the passage of pericenter tau in days
   `#ID_#LM_NB#_tra.dat`, if `idtra > 0`, naming convenction as previous. Columns: 1 trasit time, 2 LTE, 3 firt contact time, 4 second contact time, 5 third contact time, 6 fourth contact time, 7:7+NB*6 state vector {X,Y,Z,VX,VY,VZ} for each body (NB=number of bodies).

Python Library

`TRADES` can now be used in python scripts, thanks to `f2py`.
In the `src/` folder there is the `pytrades_lib.f90` that is the library that interface `TRADES` to python.
Compile:

- `cd src/`
- debug mode: `make pyomp_debug`

- release mode: `make pyomp_release`

It creates the `pytrades_lib.so` and it copies it into the folder `../pytrades/`
Check if in the `src/` folder there is an hidden file `.f2py_f2cmap` that contains one row: `dict(real=dict(sp='float', dp='double'))`

Run scripts:

In folder `pytrades/` : `pytrades_lib.so` , `constants.py` , `ancillary.py` , `trades_pso2emcee.py` .
The script to run is `trades_pso2emcee.py` , that shows how to call the `pytrades_lib` and how to combine `PSO` algorithm with `emcee` .
Run as `python trades_pso2emcee.py -h` for instruction on the all the command line arguments to provide.
The `ancillary.py` file has different functions to write and read output files from `PSO` and `emcee` and other stuff helpful to manage the output of the simulations.

TO BE CONTINUED

Changes/Log

sorry, I will not be able to report all the small changes...

TRADES 2.9.1

Small bugfixes in check of the physical parameters.
Added optional argument for debug in the output subroutine in `ode_run` .

TRADES 2.9.0

Introduced flag in `arg.in` file: `oc_fit` .
`oc_fit = F` is false, it means that the difference between the observed transit times (`T0_obs`) and the simulated (`T0_sim`) is used to compute the χ^2 ;
`oc_fit = T` is true, it means that χ^2 is computed from the amplitude of the TTV, based on the O-C for the observed (`OC_obs`) and simulated data (`OC_sim`):
`OC_obs = T0_obs - T0_lin,obs` , where `T0_lin,obs` is the linear ephemeris of the observed data and
`OC_sim = T0_sim - T0_lin,sim` , where `T0_lin,sim` is the linear ephemeris of the simulated transit times.

TRADES 2.8.0

Python library of `TRADES` has been tested and now is public.

In `bodies.lst` file user can specify if a planet should transit or not.
At the end of each planet row (after 0/1 fit-flag and before the #), user adds a T (or leaves blank) if planet is allowed to transit, otherwise adds an F.
Example:
in `bodies.lst` row `b.dat 1 0 0 0 0 0 0 0 T` is equal to `b.dat 1 0 0 0 0 0 0 0` : planet b could transit.
in `bodies.lst` row `b.dat 0 0 0 1 0 0 0 0 F` : planet b could not transit (in case it discards the solution).

Converted all logical values in a consistent way.

TRADES 2.7.0

Fitting parameters examples:
in `bodies.lst` row `b.dat 1 0 0 0 0 0 0 0` : `mass_planet / mass_star`
in `bodies.lst` row `b.dat 0 0 0 1 0 0 0 0` : `eccentricity`
in `bodies.lst` row `b.dat 1 0 0 1 1 0 0 0` : `mass_planet / mass_star , e*cos(w) , e*sin(w)`
in `bodies.lst` row `b.dat 0 0 0 0 1 1 1 0` : `argument_of_pericenter , mean longitude (lambda) , inclination`
in `bodies.lst` row `b.dat 1 0 0 1 1 1 1 1` : `mass_planet / mass_star , e*cos(w) , e*sin(w) , lambda , i*cos(lN) , i*sin(lN)`

TRADES 2.6.0

The old main `trades.f90` has been replaced (still in the sources, it compiles) by small easier-to-understand mains. As for the version `2.5.1` the user has to change the `arg.in` accordingly to the purpose. See .

During `RV` fit, `TRADES` will run a `GLS` periodogram (Zechmeister and Kurster, 2009) and look for peaks close (`$\Delta P = \pm 0.5\ d`) to the periods of the planets of the simulated system. In case of a positive signal, it means the period has been induced (bad `RV` fit) and the fitness (χ^2_r) will be set to max value (bad fit).