TRADES

`TRADES` **v2.5.0 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 citeBorsato et al. (2014).
Comments are welcome!

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.
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 a perturbing body in a four-dimensional grid: the mass, M, the period, P (or the semi-major axis, a), the eccentricity, e , and the argument of the pericenter, ω . 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 χ^2).
  We have selected these four parameters for the grid search because they represent the minimal set of parameters required to model a coplanar system. In the future, we intend to add the possibility of making the grid search over all the set of parameters for each body.

- **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 . 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` , 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 . 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.

- **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 . 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 ($T_{0,\mathrm{obs}}$) and radial velocities (RV $_{\mathrm{obs}}$) with the simulated ones ($T_{0,\mathrm{sim}}$ and RV $_{\mathrm{sim}}$). 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 $\sqrt{\chi^2_{\mathrm{reduced}}}$. We fit each new set of observables with the `LM` . 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

WARNING: only tested on a Unix/Linux machine (i.e., Centos Rocks 5.3, Ubuntu > 12.04 and derivatives)

1. Download the tar.bz2 or .zip file from the link `NOT AVAILABLE`

2. Extract the tar.bz2 (or .zip) file in your drive. It should contain a `README.md` file, `bin/` and `src/` folders.
   The `src/` folder should countains the following f90 source files:

   - **Module source files:** `constants.f90` `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` `eq_motion.f90` `output_files.f90` `numerical_integrator.f90` `radial_velocities.f90` `transits.f90` `ode_run.f90` `grid_search.f90` `lm.f90` `pikaia.f90` `util_sort.f90` `util_qmc.f90` `opti_pso.f90` `gaussian.f90` `bootstrap.f90` `PolyChord_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`
  - **Main** `TRADES` **file:** `trades.f90`
  - **Makefile:** `Makefile`
  - **python script:** `createSimFile.py`

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` is relative path and executable name for the serial program
   - flag `TARGET_OMP` 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

4. To compile:

   - serial-debug mode, type: `make serial_debug`
     It creates a executable file `trades_s` in the `bin/` folder.
   - serial-release mode, type: `make serial_release`
     It creates a executable file `trades_s` in the `bin/` folder.
   - openMP-debug mode, type: `make omp_debug`
     It creates a executable file `trades_o` in the `bin/` folder.
   - openMP-release mode, type: `make omp_release`
     It creates a executable file `trades_o` in the `bin/` folder.
   - openMP-MPI-debug mode, type: `make mpi_omp_debug`
     It creates a executable file `trades_mo` in the `bin/` folder.
   - openMP-MPI-release mode, type: `make mpi_omp_release`
     It creates a executable file `trades_mo` 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.**

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
  > ./trades_o
  > ./trades_mo
  > ```
  > **WARNING:**
  > Before running TRADES in parallel with OPENMP (`trades_o`) remember to set the number of cpus (Ncpus) to use by exporting:
  > ```
  > OMP_NUM_THREADS=Ncpu
  > export OMP_NUM_THREADS
  > ```
  > Put this in a script o 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
```

it is equal to type:

```
cd /home/user/Simulation/
trades_s .
```

or:

```
cd /home/user/
trades_s /home/user/Simulation/
```

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

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 Radius 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, in the order: `mass radius period* semi-major_axis* eccentricity argument_of_pericenter mean_anomaly(or time_of_pericenter_passage) inclination longitude_of_node`
For the `radius` only one value in Jupiter radius has to be specified.
For `mass period* eccentricity argument_of_pericenter` the first column will be used as the value for the integration and `LM` fit, or as minimum value for the `grid` , `GA` , `PSO` , and `PC` .
The second column is the maximum value for `grid` , `GA` , `PSO` , and `PC` .
The third and fourth columns are used only with `grid` . The fourth column is the type of grid to create based on the value of the third columns.
Keywords for column 4 are: `ss` (means step size), `rn` (random number), and `sn` (step number).
Units of the parameters: `mass` in Jupiter mass;
`period` in days `*` (alternatively you can provide semi-major axis in au if period is set greater than `9000000.` , while setting semi-major axis equal to `999.` will let you use the period);
`argument_of_pericenter` , `mean_anomaly` , `inclination` , and `longitude_of_node` in degrees.
The third column of the `mean_anomaly` row is a flag that let you use alternatively:
`m` the `mean_anomaly` in degrees or `t` the `time_of_pericenter_passage` in JD.
Example file `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 use increasing 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_{ref} and refined by a linear ephemeris of kind: $T_N = T_{\mathrm{ref}} + P_{\mathrm{ref}} \times N$;
   (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`

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_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_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).

TO BE CONTINUED