Basic Instructions on running the MCMC_SD_v1 astrometry fitting codes

updated for v1, Publication -- May 2018, 2017

Serge Dieterich
sdieterich@carnegiescience.edu

BASIC IDEA AND GENERAL CONSIDERATIONS

The purpose of this Markov Chain Monte Carlo (MCMC) implementation is to inferr (solve) the complete astrometric motion of a binary star. The final product are the probability density functions for trigonometric parallax, 2 components of proper motion, and the 7 orbital elements, for a total of 10 parameters. The probability density function for each parameter will normally approximate a Gaussian, meaning that we can adopt the median and the standard deviation of each PDF as the value and uncertainty for those parameters. Any parameter can be set to zero or to some other known number if that parameter is not to be solved. For instance, setting parallax and proper motion to zero will solve the relative orbit of a visual binary.

From this information one can then plot orbits and/or solve for dynamical masses, depending on what other data are available. Note that this code itself will not perform these subsequent tasks.

Markov Chain Monte Carlo Works on the basis of Bayesian inference, where the posterior of a model is probed in a probabilistic way to find the probability density function. For a discussion of the astrometric model and the MCMC implementation see Dieterich et al. 2018.

ASTROMETRY DATA FILE FORMAT, SPACE SEPARATED

```
"JD_observation  RA_displacement_mas DEC_displacement_mas  RA_error    DEC_error
      RA_parallax_factor      DEC_parallax_factor"
```

Where the RA and DEC displacements should be measured from the earliest epoch of observation. The RA and DEC uncertainties can be the standard deviation of the images taken on each epoch. There is an opportunity to add a systemic error later on.

LIST OF ASTROMETRIC PARAMETER INDICES -- BEWARE OF UNITS!
This convention is used throughout the suite for referencing astrometric parameters as array dimensions.

0 - Parallax in mas
1 - RA proper motion in mas / day
2 - DEC proper motion in mas / day
3 - Semi-major axis in mas
4 - Period in days
5 - Eccentricity
6 - Epoch of periastron in days. Must be before first epoch and preferably within one period before it.
7 - OMEGA, position angle of the line of nodes E of N, in radians
8 - omega, position angle of periastron E of N, in radians
9 - inclination in radians

BRIEF OVERVIEW OF WHAT A REDUCTION RUN LOOKS LIKE.

```
1 - IDL> results = mcmc_sd_wrapper_v1([Keywords])

2 - IDL> combined_results = reconstruct_multi4(number of sessions)

3 - IDL> save,combined_results,filename="something.sav", description="what you want
         to remember about this run"

4 - IDL> finalresult_v1, combined_results.chains, "output_file.ps",[optional
         keywords]
```

THE PURPOSE OF EACH CODE --The codes you will see--

mcmc_sd_wrapper_v1.pro ---	The main interface that runs the MCMC. It also establishes the common block of variables used mutually by several codes.
reconstruct_multi4.pro --	Combines the results that were produced over 10 output files for each simultaneous IDL session in up to 4 IDL sessions. To run multiple simultaneous sessions sessions 2, 3, and 4 should be run from subdirectories called "./session2", "./session3", and "./session4". reconstruct_multi4 should then be run from the main directory with the only parameter being the number of sessions to be combined (see below). Note that reconstruct_multi4 will produce a structure that is not automatically saved to a .sav file; that must be done manually.
finalresult_v1.pro ---	Produces the probability distribution plots for the 10 astrometric parameters. These probabilities can then be trimmed to get rid of results stuck in local minima. Also prints the mean, median, and standard deviation of each parameter fit in the plot title lines. Optionally, saves the trimmed chains.

THE PURPOSE OF EACH CODE -- Codes you don't normally interact with--

mcmc_sd_spidersolver_v1.pro ---	The MCMC "engine" that provides steps to the astrometric model, evaluates the results, decides whether or not to keep a step, and constructs the resulting Markov chains.
astromodel.pro ---	The astrometric model. It compares a given set of astrometric parameter values to the data and returns the natural log of the probability that those parameters are the right solution.
partials.pro ---	Evaluates the partial derivatives of the astrometric model at a given set of parameter values. This is used for step scaling. Note that these are partial derivatives with respect to overall RA and DEC displacement and not wrt ln(probability).

```
ecca.pro             ---   Solves Kepler's equation for the eccentric anomaly
                           numerically and returns indices of a solution array.

plothistograms.pro   ---   Produces histograms of partial results while the code is
                           running.


DETAILED CALLING OPTIONS FOR INTERACTIVE CODES
FOR MORE THOROUGH DOCUMENTATION SEE .pro FILE HEADERS.

mcmc_sd_wrapper_v1.pro --- ALL KEYWORDS NOT ENTERED IN THE CALL LINE WILL DEFAULT
                           TO POTENTIALLY INAPROPRIATE VALUES WRITEN IN THE CODE!
                           NOTE THAT KEYWORDS CAN BE ENTERED IN ANY ORDER.

IDL>results = mcmc_sd_wrapper_v1(astrofile='input_data_file',n_chains= ##, $
              n_iter= ##, n_runs= ##, ra_jitter= ##, $ dc_jitter= ##, $
              startvals=[previous.chains], scalingpar= ##, stepscale= ##, $
              stepmultiplier= ##, inputbounds= [2 by 10 array], $
              parbounds= [2 by 10 array], /tictoc, /commononly)

              Note that the results returned by this function (i.e. results = ....)
              contains only the last sub-run. Comprehensive results should be
              constructed using the reconstruct_multi4 procedure.

 INPUTS:
         astrofile = The input data file containing the astrometric data. Format:
                     "JD_observation   RA_displacement_mas  DEC_displacement_mas
                     RA_error  DEC_error   RA_parallax_factor  DEC_parallax_factor"
                     where RA and DEC displacement should be measured as offsets
                     from the first epoch of observation. JD_observation must be in
                     increasing order.

         n_chains =  The number of chains per session. We have been using a total
                     of about 50 chains for each result, meaning this number should
                     be 25 if running 2 simultaneous sessions or 17 if running 3
                     sessions (for a total of 51 in that case).
                     NOTE: All sessions must have identical calls if they are to be
                           combined by reconstruct_multi.pro

         n_iter   =  The number of steps per each sub-run. We have been using
                     10,000. n_iter x n_chains determines how much memory is
                     needed.

         n_runs   =  The number of sub-runs in a session. The last 10 sub-runs are
                     saved. Increasing the number of sub-runs allows one to
                     decrease n_iter and thereby decrease memory needs.

        ra_jitter =  Systemic RA error in milli-arcseconds to be added in
                     quadrature to the errors in the astrometry data file

        dc_jitter  = Systemic DEC error.

        startvals  = If specified it will make the run start from previous results
                     as opposed to random numbers. To start from previous results
                     in structure previousresults set startvals  =
                     previousresults.chains
```

```
       scalingpar =  The astrometric parameter used to set the step scale. We have
                     been using 0 (parallax), 3 (semi-major axis) also seems to
                     work.

       stepscale  =  Sets the scale of the step size distribution for the parameter
                     specified in scalingpar. We have been using 1, meaning steps
                     vary by about 1 mas, but see the next parameter.

   stepmultiplier =  Operates in stepscale to form a distribution. We have been
                     using stepmultiplier = 10, meaning that for every step a
                     random number between 0 and 10 is generated and there are
                     equal chances that stepscale will be multiplied or divided by
                     that number  and then multiplied by either 1 or -1.

     inputbounds   = A 10x2 array that indicates the interval from which starting
                     guesses should be drawn when initiating each chain. If
                     startvals is set then inputbounds is ignored. Example:
                     inputbounds = [[265d,285d],[10.7d,11d],[-6.8d,-6.5d],  $
                     [150,350d],[3000d,7500d],[0.35d,0.6d],[2.45d6, min_jd], $
                     120d*!pi/180d,180d*!pi/180d],[0d,2d*!pi],[70d*!pi/180d, $
                     !pi/2d],[150d,350d]] means that starting values for parameter
                     0 (parallax) will be drawn from the 265 to 285 range, and so
                     on. Note that parameter 6, the epoch of periastron, must
                     always be earlier than the first epoch of observation. Because
                     inputbounds can be rather lengthy you may wish to either set
                     it in the code or set it to an array name that can then be
                     entered repeatedly instead of the explicit expression.

      parbounds   =  The interval in which each parameter will be allowed to
                     fluctuate. Same format as inputbounds. Parbounds should not be
                     too restrictive because that takes away the MCMC's ability to
                     find its way back to higher probabilities when a parameter
                     happens to be off by a large amount. A good rule is to enter
                     the plausible range given the observational setup without
                     taking into account specific knowledge about the system you
                     are fitting. If the upper and lower limits for a parameter are
                     identical then that parameter is considered a constant.


          tictoc  =  If /tictoc is included in the call line then the elapsed time
                     for the entire run will be output by the IDL tic toc command
                     at the end of the run. Be sure to check that your IDL
                     implementation supports tic toc:
                       IDL> tic
                       IDL> toc
                     should print the ellapsed time.

      commononly  =  If set (/commononly) then this wrapper code will establish the
                     common variable block and exit without running any codes. This
                     is useful in case you need to run codes individually.


--- reconstruct_multi4.pro ---

IDL>combined_results = reconstruct_multi4(number of sessions)

 INPUT:
        nsessions = 1, 2, 3, or 4.
                     The number of IDL sessions that were run in parallel to speed
```

```
                    up the computation. If nsessions > 1 then the other sessions
                    must have IDENTICAL calling sequences for
                    mcmc_sd_wrapper_v1.pro and must be run from subdirectories
                    ./session2, ./session3, and ./session4

 OUTPUT:
        A structure of the form ('chains', 'lnPchains', 'Matchepoch')where the
        number of chains has been combined from the multiple IDL sessions (if more
        than 1) and the length of the chains has been combined from the last 10
        sub-runs.

--- finalresult_v1.pro  ---

IDL>finalresult_v1, chains, outfile[, /savechains, chainsfile="string.sav", $
                    binning=(10 element double array), trimstatement="string"]

 INPUT:
         chains   = The array containing chains, generated by
                    reconstruct_multi4.pro and usually part of a structure:
                    combined_results.chains

         outfile  = String with the .ps file name to save plots: e.g.
                    "filename.ps"

       savechains = If set (/savechains) then the trimmed chains will be saved to
                    a .sav file.

       chainsfile = If savechains is set then chainsfile is a string with the .sav
                    file name where the trimmed chains will be saved: e.g.
                    "trimmedchains.sav"

        binning   = A 10 element double precision vector with the histogram
                    binning for every parameter, in the standard index order.
                    If not set then default values written in the code are used.

    trimstatement =  A string to be used as the condition for a where() statement
                    that trims bad samples. The format is, e.g, "chains0 gt 285"
                    to trim all samples with parallax (parameter 0) greater than
                    285. Any conditional statement allowed in a where( ) call can
                    be used. Use a very large or very small condition, or set
                    badindices = -1 in line 84 and don't set trimstatement in the
                    call for no trim.

 OUTPUT:
        .ps file with probability density function histogram plots labeled with
        mean, median, and standard deviation, in that order.


         If savechains is set (/savechains) a .sav file with the trimmed chains.
         Array format is dblarr(parameters, samples per chain, chains). Includes
         NaNs.


SEE DOCUMENTATION IN INDIVIDUAL .pro FILES FOR PARAMETERS OF NON-INTERACTIVE CODES.
```