Phylogenetic Inference using `RevBayes`

MCMC algorithms

Sebastian Höhna, Mike R. May and Brian R. Moore

This tutorial is currently under construction/revision.

1 Overview

This tutorial demonstrates how to diagnose the performance of MCMC simulations, such as the output from `RevBayes`. You will create a phylogenetic model for the evolution of DNA sequences under a JC, HKY85, GTR, GTR+Gamma and GTR+Gamma+I substitution model. For all these models you will perform an MCMC run to estimate phylogeny and other model parameters.

Requirements

We assume that you have completed the following tutorials:

- RB_MCMC_Tutorial

2 Exercise: Assessing Performance of MCMC Simulations

"*You can never be absolutely certain that the MCMC is reliable, you can only identify when something has gone wrong.*" Andrew Gelman

Model-based inference is, after all, based on the model. Careful research means being vigilant both regarding the choice of model and rigorously assessing our ability to estimate under the chosen model. The first issue—model specification, which actually entails three closely related issues—is critically important for the simple reason that unbiased estimates can only be obtained under a model that provides a reasonable description of the process that gave rise to our data. *Model selection* entails assessing the *relative fit* of our dataset to a pool of candidate models. Rankings are based on model-selection methods that compare the relative fit of candidate modes based either on their maximum-likelihood estimates (which measures the fit of the data to the model at a single point in parameter space), or on marginal likelihood of the candidate models (which measures the average fit of the candidate models to the data). *Model adequacy*—an equally important but relatively neglected issue—assesses the absolute fit of the data to the model. *Model uncertainty* is related to the common (and commonly ignored) scenario when multiple candidate models provide a similar fit to the data: in this scenario, conditioning on *any single* model (even the best) will lead to biased estimates, and so model averaging is required to accommodate uncertainty in the choice of model.

Much less concern is given to the second aspect of model-based inference: the ability to obtain reliable estimates under the chosen model(s). The implicit assumption, it seems, is that if a model is implemented correctly, and if that implementation is successfully used to obtain an estimate from a given dataset, then we must have performed valid inference under the model. This would be perfectly sound reasoning if inferences were based on analytical methods. Owing to the complexity of the models, however, it is not possible to estimate phylogenetic parameters analytically. Instead, parameter estimates are based on numerical methods. In the case of maximum-likelihood estimation, these are typically hill-climbing algorithms that attempt to search the profile likelihood to identify the vector of point estimates of all phylogenetic model parameters that jointly maximize the likelihood of observing the data under the model. The reliability of these algorithms can (and should) be assessed by comparing estimates obtained from repeated analyses that are initiated from random points in parameter space. Because there is only one *maximum* likelihood estimate, the terminal values estimated by replicate runs should be identical (within the precision of computer memory).

In the Bayesian statistical framework, inferences focus on the joint posterior probability density of phylogenetic model parameters, which is approximated by Markov chain Monte Carlo (MCMC) algorithms. It may be comforting to know that, in theory, an *appropriately constructed* and *adequately run* MCMC simulation is guaranteed to provide an arbitrarily precise description of the joint posterior probability density. In practice, however, even a given MCMC algorithm that provides reliable estimates in *most* cases will nevertheless fail in *some* cases and is not guaranteed to work for any given dataset. This raises an obvious question: "When do we know that an MCMC algorithm provides reliable estimates for a given empirical analyses". The answer is simple: *Never.*

Fortunately, this problem is not unique to the field of phylogenetics. Much of Bayesian inference outside our field also relies on MCMC algorithms to approximate the joint posterior probability density of parameters: similar concerns regarding the reliability of those inferences has motivated the development of a suite of diagnostic tools to assess MCMC performance. The trick is learning how to use these tools effectively and rigorously, especially for analyses that entail complex phylogenetic models and/or large datasets.

2.1 MCMC Basics

The ability to rigorously diagnose MCMC performance requires familiarity with some basic concepts from probability theory (discussed last time) and a strong intuitive understanding of the underlying mechanics—we need to know how the algorithms work in order to understand when they are not working. In this installment we'll briefly cover the mechanics of the Metropolis Hastings MCMC algorithm.

Recall that Bayesian inference is focused on the posterior probability density of parameters. The posterior probability of the parameters can, in principle, be solved using Bayes' theorem. However, (most) phylogenetic problems cannot be solved analytically, owing mainly to the denominator of Bayes' theorem—the marginal likelihood requires solving multiple integrals (for all of the continuous parameters, such as branch lengths, substitution rates, stationary frequencies, etc.) for each tree, and summing over all trees.

Accordingly, Bayesian inference of phylogeny typically resorts to numerical methods that approximate the posterior probability density. There are many flavors of Markov chain Monte Carlo (MCMC) algorithms—Gibbs samplers, Metropolis-coupled and reversible-jump MCMC, etc.—we will consider the Metropolis Hastings (MH) algorithm because it is commonly used for phylogenetic problems, and because it is similar to many other variants (which we will cover elsewhere). Note that MCMC and Bayesian inference are distinct animals: they have a relationship similar to that between 'optimization algorithms' and âĂŸmaximum-likelihood estimation.' Some Bayesian inference can be accomplished without MCMC algorithms, and MCMC algorithms can be used to solve problems in non-Bayesian statistical frameworks.

To introduce the MH algorithm, we will imagine a robot that is programed to explore an area. Specifically, the goal of our robot is to generate a topographic map of an unknown terrain. This terrain has a total surface area of one hectare. We deploy our robot by parachute at a random location within the terrain. We have programmed the robot with three simple rules:

1. If a proposed step will take the robot uphill, it automatically takes the proposed step.
2. If a proposed step will take the robot downhill, it divides the elevation of the proposed location by the elevation of the current location: we call this quotient R. It then generates a uniform random number, $U[0, 1]$. If $U < R$, the robot takes the proposed step; otherwise, it stays put.
3. The distribution for proposing steps is symmetrical. That is, the probability of proposing a step (but not necessarily accepting a proposed step) from point A to point B is equal to the probability of proposing a step from point B to point A.

We allow our little robot to wander through the terrain following these three simple rules. At prescribed intervals (*e.g.*, every 10 proposed steps), the robot records the details of his position (his elevation, latitude, longitude, etc.) in a log. If we allow our robot to wander long enough, his log is guaranteed to provide an arbitrarily precise description of the topography of the terrain.

Let's consider a slightly more formal description of the Metropolis-Hastings MCMC algorithm. First some preliminaries. This algorithm entails simulating a Markov chain that has a stationary distribution that is the joint posterior probability density of phylogenetic model parameters. The 'state' of the chain is a set of parameter values that fully specify phylogenetic model: *e.g.*, a tree topology, a vector of branch lengths, and a set of parameters specifying the stochastic model of trait change (*e.g.*, for the GTR + Γ substitution model, this comprises a vector of values for the four stationary frequencies, a vector of values for the six exchangeability parameters, and the value of the alpha parameter describing the shape of the gamma-distributed among-site rate variation). Under the robot analogy, the state of the chain corresponds to a unique point in the terrain. The Markov property of the MCMC reflects the fact that the next state of the chain only depends on the current state of the chain, but not on previous states—that is, the past affects the future only through the present. The Monte Carlo aspect of the MCMC reflects the fact that it is a simulation: it is a numerical method that relies on repeated random sampling.

Now, on to the MH algorithm:

1. Initialize the chain with values for all parameters, including the tree topology, τ, the vector of branch lengths ν, and substitution model parameters π_i, q_i, α. We will call the set of model parameters θ. The initial parameter values might be specified arbitrarily, or might be drawn from the corresponding prior probability density for each parameter.

2. Select a single parameter (or set of parameters) to alter according to its proposal probability. For example, here are the default proposal probabilities used by `MrBayes`:

```
The MCMC sampler will use the following moves:
   With prob. Chain will use move
      1.00 %  Dirichlet(Revmat)
      1.00 %  Slider(Revmat)
      1.00 %  Dirichlet(Pi)
      1.00 %  Slider(Pi)
      2.00 %  Multiplier(Alpha)
     10.00 %  ExtSPR(Tau,V)
     10.00 %  ExtTBR(Tau,V)
     10.00 %  NNI(Tau,V)
     10.00 %  ParsSPR(Tau,V)
     40.00 %  Multiplier(V)
     10.00 %  Nodeslider(V)
      4.00 %  TLMultiplier(V)
```

   We can see that $\sim 1\%$ of the time (*i.e.*, with probability ~ 0.01), the MCMC will propose changes to the exchangeability ('revmat') and stationary frequency ('pi') parameters using a 'Dirichlet' proposal mechanism, and an equal effort proposing changes to the same parameters using something called a 'Slider' proposal mechanism. Conversely, $\sim 10\%$ of the time (*i.e.*, with probability ~ 0.1), the MCMC will propose changes to the tree and branch lengths ('Tau and V') using the 'extending SPR' proposal mechanism, and with equal probability using the 'extending TBR', 'extending NNI' and the 'parsimony SPR' proposal mechanisms.

   For the moment, we won't worry about the details of these proposal mechanisms—they basically involve different ways of 'poking' the current parameter value, θ, to generate a new (proposed) parameter value, θ'. The important question at the moment is: Where do these proposal probabilities come from? The answer is: experience. We want to design the MCMC such that it invests effort in a given parameter in proportion to the difficulty of approximating that parameter. Note that the MCMC summarized above will spend $\sim 2\%$ of its time proposing changes to the exchangeability and stationary frequency parameters, but will invest $\sim 40\%$ of its time proposing changes to the topology parameter. Experience suggests that the tree topology is a more difficult parameter for the MCMC to approximate relative to the exchangeability and stationary frequency parameters (in fact, it appears that the developers of `MrBayes` determined from their experience that the topology is ~ 20 times harder to approximate).

3. Propose a change to the selected parameter using the parameter-specific proposal mechanism. Different parameters may naturally have different prior probability densities—for example, the stationary frequencies are proportions (ranging between 0 and 1), and so are conveniently described using a Dirichlet prior probability density, whereas the alpha-shape parameter can range between zero and infinity, so is better described using a uniform prior probability density. Different kinds of prior probability densities will have specific proposal mechanisms—for example, there is something called a 'Dirichlet' proposal mechanism that is used to propose new values for parameters described by a Dirichlet prior, and something called a 'Slider' proposal mechanism that is used to propose new values for parameters described by a uniform prior. (Again, we'll go into the gory details of these proposal mechanisms another time. The main idea for now is that the proposal mechanisms generate a new parameter value by poking the current parameter value.)

   The design of proposal mechanisms is something of a dark art—trial and error are used to guide the development of mechanisms that work well. Nevertheless, there are basic criteria that the proposal mechanisms must meet. All proposal mechanisms must be:

   (a) stochastic (new parameter values must be proposed 'randomly')
   (b) irreducible (all parameter values must be potentially accessible by the chain)
   (c) aperiodic (the proposal mechanism must not induce epicycles of the chain)

4. Calculate the probability of accepting the proposed change, R:

$$R = \min\left[1,\ \underbrace{\frac{P(X \mid \theta')}{P(X \mid \theta)}}_{\text{likelihood ratio}} \cdot \underbrace{\frac{P(\theta')}{P(\theta)}}_{\text{prior ratio}} \cdot \underbrace{\frac{P(\theta \mid \theta')}{P(\theta' \mid \theta)}}_{\text{proposal ratio}}\ \right] \tag{1}$$

   The acceptance probability, R, is either equal to one or the product of three ratiosâĂŤso long as that product is less than one (because R is a probability, so it cannot be greater than one). So, what are these three ratios?

   Likelihood ratio: the likelihood ratio is simply the likelihood of the observations given the proposed value of the parameter, divided by the likelihood of the observations given the current value of the parameter. We can calculate the likelihood for any given parameterization of the phylogenetic model using the pruning (Felsenstein) algorithm.

   Prior ratio: in Bayesian inference, each parameter is a random variable, and so is described by a prior probability density. Accordingly, we can 'look up' (calculate) the prior probability of any specific parameter value. The prior ratio is simply the prior probability of the proposed and current parameter values.

   Proposal ratio: the proposal ratio is the probability of proposing the current parameter value given the proposed parameter value, divided by the converse. This is also called the Hastings ratio. This is equivalent to the rule that forces our robot to propose steps symmetrically; *i.e.*, that the probability of proposing a step from point A to point B in the terrain is equal to the probability of proposing a step from point B to point A. For inference problems in which the dimensions of the model are static, the proposal ratio is usually equal to one (we'll discuss exceptions to this in the context of reversible-jump MCMC in a future post).

   Note that if the proposal ratio is equal to one, the acceptance probability, R is based only on the product of the likelihood and prior ratios. Does that ring a bell? [Hint: think of Bayes' theorem.]

[Hint 2: think of the right side of Bayes' theorem.] [Hint 3: think of the numerator of the right side of Bayes' theorem.] That is, the posterior probability is proportional to the product of the likelihood and prior probability. Accordingly, the acceptance probability is the posterior probability of the proposed state divided by the posterior probability of the current state. Just as we programmed our robot, if the ratio of the proposed and current elevations (posterior probabilities) is greater than 1 (*i.e.*, it is an uphill step), we always accept the proposed change. When the ratio of the proposed and current elevations is less than 1 (*i.e.*, it is an downhill step), we may or may not take the proposed step (stay tuned).

5. Generate a uniform random number between zero and one, $U[0,1]$. If $U < R$, accept the proposed change; otherwise, the current state of the chain becomes the next state of the chain. Downhill moves will be accepted 'randomly' in proportion to the difference in elevation.

6. Repeat steps $2-5$ an 'adequate' number of times.

That's it! Notice that the decision to accept or reject proposed steps in the MCMC (and thus to sample from the joint posterior probability density) is based exclusively on the likelihood and prior probability of the proposed and current states—two quantities that are easy to calculate. The beautiful, fantabulous achievement of the Metropolis-Hastings algorithm is the slaying of the beastly denominator of Bayes' theorem. Specifically, the algorithm allows us to do inference while entirely avoiding the need to calculate the (completely intractable) marginal likelihood!

Approximating the joint posterior probability density is based on samples from the MCMC: a chain following the simple rules outlined above will sample parameter values in proportion to their posterior probability. That is, the proportion of time that the chain spends in any particular state is a valid approximation of the posterior probability of that state (*e.g.*, if a clade is present in 87% of the samples drawn from the chain, then the posterior probability of that clade is estimated to be 0.87).

Why do we accept proposals to states with lower posterior probability? People familiar with maximum-likelihood estimation often misunderstand the purpose of accepting proposals to states with a lower posterior probability. In maximum-likelihood inference, the game is to simply climb relentlessly and mindlessly up the likelihood surface in search of the very tip of the globally highest peak. Accordingly, the only reason these zombie hill climbers might consider a downward move would be motivated by concerns that they were currently climbing up a local (rather than global) optimum. By contrast, Bayesians are not just interested in the elevation of the very tip of the absolute peak of the posterior probability surface, but rather are interested in exploring and estimating the entire joint posterior probability density—we want topographical map of the entire posterior probability, not the elevation of the tip of the very highest peak in parameter space.

This survey of the joint posterior probability density results in a more robust inference procedure (inferences are averaged over the joint posterior probability of all model parameters), and allows us to estimate and evaluate the marginal posterior probability density for each of the parameters (these can be viewed by querying the MCMC samples with respect to the parameter of interest—we will do this in future tutorials). Accordingly, it is not sufficient for the chain to reach the peak of the joint posterior probability density; we need the chain to mix over the entire stationary distribution, spending time at each location in proportion to its posterior probability.

Next to the specification of priors (which are inspired by much more philosophical considerations), performance of the MCMC algorithm used to approximate the joint posterior probability distribution is the greatest concern associated with Bayesian inference (it is certainly a more general concern, as it holds even when the priors are uncontroversial, and, in my opinion, is a more legitimate and practical concern). There are three closely related issues associated with MCMC performance: (1) convergence (is the robot sampling from the stationary distribution); (2) mixing (is the robot efficiently moving over the posterior probability density); and (3) adequacy (has the robot collected sufficient samples from the target distribution to describe it adequately).

3 Diagnosing MCMC performance

3.1 Running Markov chain Monte Carlo Simulations & Assessing Output

3.1.1 Data & Model

This is an analysis of 13 species within the genus *Fagus* (the beeches), based 2576 sites sampled from one nuclear gene region (ITS) and two chloroplast gene regions (rbcL and matK). Here we assume a GTR+Γ+I substitution model that is uniformly shared by all sites: we refer to this as the ‘uniform’ partition scheme, or PS0 for short. This is the same data and model as we used previously in the CTMC tutorial.

Look at Model file & MCMC file in a text editor.

3.1.2 Running MCMC

Let us now explore the behavior of our Markov chain. The aim here in these exercises is to visually inspect the output generated by `RevBayes`. You should look at the trace plots using `Tracer`. Watch out for badly mixing chains, poor performance of the MCMC and correlated parameter estimates.

The first analysis which we perform is specified as follows:

- uniform GTR+I+G
- single move per cycle
- large proposal for all parameters
- no-pre-burnin

You can run this analysis directly using

```
source("RevBayes_scripts/analysis/mcmc_run1_PS0.Rev")
mymcmc.operatorSummary()
```

The operator summary provides you with an overview of the moves that you have used for this MCMC run. It tells you what the weight for each move was, the variables it has been working on, the number of times the move was used, and how often the move was accepted. Good acceptance rates for continuous parameters are between 20% and 60%. For discrete characters, such as the phylogeny, there is no rule of thumb what good acceptance is. Instead, the acceptance rate of tree topology proposal strongly depends on the current data.

Now let us open the generated output file in `Tracer`. The file should be called output/fagus_ps0_posterior_run1.log.

What you may notice is that the ESS are very low and the MCMC is mixing very poor. This is not good. We'll try a new analysis but instead run multiple moves per iteration. The new setting is:

- uniform GTR+I+G
- *multiple moves per cycle*
- large proposal for all parameters
- flat & low proposal weights
- no-pre-burnin

You can run this analysis in `RevBayes` using

```
source("RevBayes_scripts/analysis/mcmc_run2_PS0.Rev")
mymcmc.operatorSummary()
```

Look at the output in `Tracer` again. You should see that the ESS values are still very low and the chain is still not mixing well.

One issue that you see is that the probability of a site being invariant is update not often enough. It is quite likely that our move proposed too many bad new parameters. So let us change the window size for the sliding window move applied on the probability of invariant sites from

```
moves[mi++] <- mvSlide(pinvar, delta=10.0, tune=false, weight=1.0)
```

to

```
moves[mi++] <- mvSlide(pinvar, delta=1.0, tune=false, weight=1.0)
```

This will give us now the following set-up:

- uniform GTR+I+G
- multiple moves per cycle
- *smaller proposals for Pinv*
- flat & low proposal weights
- no-pre-burnin

Run the analysis file and look how often the Sliding Move is now accepted.

```
source("RevBayes_scripts/analysis/mcmc_run3_PS0.Rev")
mymcmc.operatorSummary()
```

Open the output and look at it in `Tracer`. We are now getting better ESS values and better mixing of the MCMC.

That worked quite well but it seems to be cumbersome to adjust all the tuning parameters of the moves manually. Instead, let us allow `RevBayes` to auto-tune the moves. Just change **`tune=true`** for all moves where it was **`tune=false`** before. Now, also include a pre-burnin. The pre-burnin runs if you say **`mymcmc.burnin(generations=1000,tuningInterval=100)`**. This runs a chain for 1000 iterations and updates the tuning parameters, such as the window size, so that you achieve a good acceptance rate. That means, if previously a move accepted too few proposal it will decrease the window size.

Here is the new model set-up:

- uniform GTR+I+G
- multiple moves per cycle
- flat & low proposal weights
- *pre-burnin (with scale adjusted from previous)*

Run the analysis in `RevBayes`:

```
source("RevBayes_scripts/analysis/mcmc_run4_PS0.Rev")
mymcmc.operatorSummary()
```

Look at the output given by the operator summary. Notice how the tuning parameters have been updated. Also look at the output in `Tracer`. You will see that the ESS values are much better because the chain is mixing better.

In the next step we chain the proposal weights so that moves are used more often, especially for the parameters that are not mixing well. Our new setting is:

- uniform GTR+I+G
- multiple moves per cycle
- *variable proposal weights*
- pre-burnin (with scale adjusted from previous)

Run the `RevBayes` analysis.

```
source("RevBayes_scripts/analysis/mcmc_run5_PSO.Rev")
mymcmc.operatorSummary()
```

When you look at the output you will see that it now mixing fairly well. So let us focus on the approximated posterior distribution. Specifically, what is the posterior mean of the tree-length (`TL`)? This tree-length is surprisingly large. We could specify a prior distribution with a lower expectation (mean) on the tree-length.

- uniform GTR+I+G
- multiple moves per cycle
- variable proposal weights
- pre-burnin (with scale adjusted from previous)
- *branch length prior to* **`dnExponential(20.0)`** *(mean 0.05)*

Run the new analysis in `RevBayes`.

```
source("RevBayes_scripts/analysis/mcmc_run6_PSO.Rev")
mymcmc.operatorSummary()
```

If you look at the results from this analysis, you will notice that the tree-length estimates are better. However, if you look at the joint posterior density of the tree-length (`TL`) and the probability of a site being invariant (**`pinvar`**), then you will notice that there is a very high correlation between these two. It seems as if our model is over-parameterized. To resolve this issue we need to remove/fix some of our parameters. Open the data file and look how variable each site is. What is your impression on how many sites are invariant?

We remove the gamma distributed rate variation among sites because most sites either seem to be fixed or evolve under our GTR model. We also reset the branch length priors to **`dnExponential(10.0)`**.

- uniform GTR+I
- multiple moves per cycle
- variable proposal weights
- pre-burnin (with scale adjusted from previous)

Run the new script in `RevBayes`.

```
source("RevBayes_scripts/analysis/mcmc_run7_PSO.Rev")
mymcmc.operatorSummary()
```

Look again at output in `Tracer`. We seem to be doing a pretty good job with one chain. This is our visual method to inspect the MCMC output.

Next, let's compare it to the prior. Here we want to see how much information we actually have from the data and how much our prior influenced our posterior estimates. Run now an analysis that does not use the data. You can do this in `RevBayes` the exact same way as running an MCMC on the exact same model, you only need to call **`ymmcmc.run(1000,underPrior=true)`**.

```
source("RevBayes_scripts/analysis/mcmc_run8_1_prior_PSO.Rev")
mymcmc.operatorSummary()
```

Load both the MCMC output under the prior and the posterior into `Tracer`. Compare the approximated distributions on each parameter between both outputs.

Since we are doing fine for a single run, try multiple runs to check if the results are reproducible.

```
source("RevBayes_scripts/analysis/mcmc_run7_1_PSO.Rev")
source("RevBayes_scripts/analysis/mcmc_run7_2_PSO.Rev")
source("RevBayes_scripts/analysis/mcmc_run7_3_PSO.Rev")
```

...and the prior...

```
source("RevBayes_scripts/analysis/mcmc_run8_2_prior_PSO.Rev")
source("RevBayes_scripts/analysis/mcmc_run8_3_prior_PSO.Rev")
source("RevBayes_scripts/analysis/mcmc_run8_4_prior_PSO.Rev")
```

Apply multiple-run diagnostics:

- PSRF
- ASDSF
- comparetrees

4 Semi-automatic MCMC diagnosis using bonsai

Diagnosing MCMC performance manually can be extremely cumbersome, especially for more complex models. The R package bonsai performs a standard set of diagnostics on MCMC output and generates a report that highlights potentially pathological MCMC behaviors. To use bonsai, download the package source from https://github.com/mikeryanmay/bonsai, launch the R console, and install the package from source by following the instructions on the bonsai main page (making sure to install the packages that bonsai relies on as well):

```
install.packages(packages=c('coda','tools','RColorBrewer','entropy','xtable'),
    dependencies=TRUE)
install.packages('< path to bonsai >',repos=NULL,type='source')
library(bonsai)
```

Move the output files from `run7` to the folder `.../RB_MCMC_Tutorial/output/output_for_bonsai`.

Going back to the R console, provide a name for the project.

```
project <- 'Fagus run 7'
```

Point bonsai at the directory where we placed our log files.

```
path <- '.../RB_MCMC_Tutorial/output/output_for_bonsai'
```

Finally, we create the bonsai object and execute the `runBonsai()` function. It will generate a report that draws your attention to potential issues.

```
fagus_proj <- bonsai(project=project,path=path)
fagus_proj$runBonsai()
```

Check the flags generated by bonsai (Figure 1; your results may differ). Many of these flags may be innocuous; it is up to the user to decide whether a particular flag is truly problematic. For example, a correlation between `Likelihood` and `Posterior` is not problematic; however, a significant p-value for Geweke's diagnostic for `pi[3]` indicates that this parameter has not converged to its stationary distribution.

There are additional bonsai reports for more complex models demonstrating a wider array of MCMC pathologies in the `.../RB_MCMC_Tutorial/bonsai_pre_cooked` directory.

References

Version dated: August 8, 2017

2.2.1 Critical flags

- Run 2: Parameter `pi[3]` has critically low p-value for Geweke's diagnostic ($p = 0.002$)
- Parameters `Likelihood` and `Posterior` are strongly correlated ($\rho = 1$)
- Parameters `TL` and `Prior` are strongly correlated ($\rho = -1$)

2.2.2 Major flags

- Run 2: Parameter `Prior` has very low p-value for Geweke's diagnostic ($p = 0.023$)
- Run 2: Parameter `TL` has very low p-value for Geweke's diagnostic ($p = 0.023$)
- Parameters `Prior` and `Posterior` are correlated ($\rho = 0.292$)
- Parameters `TL` and `Posterior` are correlated ($\rho = -0.292$)
- Parameters `pinvar` and `Posterior` are correlated ($\rho = -0.268$)

Figure 1: Flags generated by bonsai.