Bayesian Analysis in AD Model Builder

Cole C. Monnahan — monnahc@uw.edu
Peter T. Kuriyama — ptrkrym@uw.edu
Melissa L. Muradian — murad17@uw.edu

September 29, 2014

Abstract

The goal of this guide is to outline and describe the steps needed to conduct a Bayesian analysis in AD Model Builder (ADMB). Included are general descriptions of Bayesian inference, priors, and two MCMC algorithms. We hope that the guide will enable users to take full advantage of the features of ADMB that are built-in, but not well-documented.

Contents

1 Introduction

The goal of this guide is to outline and describe the steps needed to conduct a Bayesian analysis in AD Model Builder (ADMB). Included are general descriptions of Bayesian inference, priors, and two MCMC algorithms. We hope that the guide will enable users to take full advantage of the features of ADMB that are built-in, but not well-documented.

AD Model Builder is the fastest, most powerful software for fitting general nonlinear statistical models. The "AD" refers to automatic differentiation that comes from a C++ library which implements reverse mode automatic differentiation.

-Add short paragraph about r2admb package

We encourage others to contribute to this guide and further document the Bayesian capabilites of ADMB. The contents of this document available on a github repository maintained by Cole Monnahan.

Something about about where to make code suggestions?

2 Bayesian inference

In general, Bayesian inference is used to quantify the full range of uncertainty around both models and parameter values and incorporate findings from historical studies. In a fisheries specific context, Bayesian inference allows scientists to investigate the consequences of alternative management strategies.

Markov chain Monte Carlo (MCMC) is a common algorithm used to sample and compute expectations with respect to complex and high-dimensional probability distributions. ADMB implements two different MCMC algorithms: the ubiqitous Metropolis-Hastings and a Hamiltonian (or "hybrid") sampler, both described in this section.

John Kruschke in "Doing Bayesian Data Analysis: A Tutorial with R and BUGS" has a simple example that illustrates the mechanics of the Metropolis-Hastings algorithm. A politician lives on a long chain of islands. The politician's goal is to stay in the public eye by traveling to each island proportional to their relative population. In other words, he will spend more time on highly populated islands and less time on less populated islands. He flips a fair coin to propose a move to an island east or west of his current location. If the proposed island has a higher population than his current island he moves. If the proposed island has a smaller population than his current island he will only move with a probability $P_{proposed}/P_{current}$ where $P_{proposed}$ is the population of the proposed island and $P_{current}$ is the population of the current island.

All MCMC algorithms work in this same general way. The user will start the algorithm at an arbitrary starting point, and the algorithm will propose new parameter sets or states. The algorithm will propose new states (i.e. parameter draws) and move to that state, or not, depending on its density relative to the current state. Just like the politician, the algorithm will spend most of the time in states with high densities.

MCMC algorithms typically need a "burn-in" period, especially if the starting point is in a region with low density. The goal of a MCMC algorithm is to explore areas with the highest probabilities in a distribution. It will perhaps take thousands of draws for the algorithm to find the peaks of the distribution, especially if it starts in a low density region. Discarding these intial draws allows the algorithm to explore the main regions of the target distribution.

Autocorrelated MCMC samples are close together and probably not a true represenation of the underlying distribution. Samples are typically "thinned" every nth-step to generate uncorrelated samples. The tradeoff is that excessive thinning increases run-time for the algorithm.

The R package `CODA` is a commonly used tool for checking MCMC diagnostics such as autocorrelation.

3 Priors

As with any Bayesian analysis, priors are a key component of the posterior surface and must be considered in ADMB as well. The posterior density is proportional to the product of the likelihood and priors. ADMB works in the negative log space, so the contribution of priors is incorporated into the model by adding the negative log of the prior density to the objective function. At this point the minimum point is no longer the maximum likelihood estimate, but rather what we refer to as the "maximum posterior mode"– i.e. the parameters corresponding to the highest point of the posterior. The difference between the two is subtle, but there are important philosophical differences: with Bayesian analysis ADMB is integrating across the parameter space, while the likelihood framework focuses on maximization.

When running an MCMC in ADMB, users need not explicitly define and incorporate priors into their model. In this case "implicit" priors are still affecting the posterior. In particular, unbounded parameters without explicit priors have infinitely-broad uniform priors (an example of an "improper" prior because it will not integrate to 1). Another example is a variance parameter estimated in log space and then exponentiated within the model to keep it above zero. An implicit uniform prior on the log parameter may not be uninformative on the natural scale. The implications of these issues are contentious and certainly beyond the scope of this guide. We caution users to carefully consider and explicitly acknowledge their priors – ADMB will not force you to.

Bounding a parameter incorporates a Uniform prior without changing the objective function calculation. Other forms of prior distributions must be added manually with, or without, any constants. For instance, a normal ($N(5,1)$) prior on a parameter a could be added as `f+=pow((a-5),2)/(2*1*1)`, where the constants have been dropped.

4 Markov chain Monte Carlo (MCMC) in ADMB

Markov chain Monte Carlo (MCMC) is a common algorithm used to sample from arbitrary, unscaled posterior distributions. ADMB implements two different MCMC algorithms: the ubiqitous Metropolis-Hastings and a Hamiltonian (or "hybrid") sampler, both described briefly below.

All MCMC algorithms work by proposing new states (i.e. parameter vectors) and moving to that state, or not, depending on its density relative to the current state. The algorithm thus generates a series of autocorrelated parameter vectors, which can be thinned to produce independent samples from the posterior distribution of interest.

4.1 Workflow

The R package `R2admb` contains some useful functions for streamlining the workflow of MCMC in ADMB. For users who wish to have more manual control, the following steps outline a typical workflow.

1. Build, run, and verify an ADMB model. This model must explicitly include the contribution of the priors to the objective function, such that the ADMB estimate is the posterior mode, rather than a maximum likelihood estimate.

2. Run an MCMC using the command line argument `-mcmc` N `-mcsave` $N_{\texttt{save}}$ (see Table 2 for more runtime options). The thinned draws are discarded, leaving a total of $N_{\text{out}} = N/N_{\text{save}}$ saved draws. For example `-mcmc 1e6 -mcsave 1000` will run 1 million draws but only save (i.e."thin") every 1000th, for a total kept of $Nout = 1000$.

3. After completion, run the model again with argument `-mceval`. This command tells ADMB to loop through the saved iterations (in the `.psv` file) and execute in the `mceval_phase()`.

4. Pull results into R or other program to ensure the sample is sufficiently thinned, either visually or with tools using, for example, the `CODA` package.

5. If necessary, rerun the chain with more thinning, drop the first part of the chain as a "burn-in," or run longer for more iterations.

6. Make whatever Bayesian inference is desired using the $Nout$ independent samples.

4.2 MCMC Phases

ADMB is designed with two phases that are used to produce MCMC output: (1) the `mcmc` phase and (2) `mceval` phase. While the use of these phases is not common (is this true??) among other MCMC software, and may be a source of confusion for new ADMB users, they provide a powerful and efficient framework for MCMC analyses.

The `mcmc` phase is the one with which most people are already familiar. This is where ADMB generates new parameter sets by proposing a set, and then determining whether to move there or stay at the current set. This process is repeated N times, and how sets are proposed depend on the algorithm used (see section 5 and 6). During this phase, the $Nout$ saved parameter values are written to a `.psv` file (described below). Note that if `-mcsave` *nsave* is not specified, ADMB will run the MCMC but no values will be saved.

The `mceval` is an optional phase that is designed to be run after the `.psv` file has been produced. During this phase, ADMB loops through the $Nout$ parameter combinations in the `.psv` file and reruns the `PROCEDURE_SECTION` in the `mceval()` phase. This phase is extremely powerful because it allows the user to minimize wasted calculations by parsing calculations into two groups: those that affect the objective function (i.e. posterior calculations for Bayesian analyses) and those that do not. Calculations done for discarded draws (which often is most iterations) simply slow down the analysis. Thus, an analysis can be made to run faster by minimizing calculations in the `mcmc` phase. For example, a user may want to extrapolate (e.g. project a time series into the future) or calculate values derived from the parameters and intermediate values. By putting these calculations inside an `mceval_phase` clause they are only done for saved draws and the MCMC will run faster. In practice, some chains need to be thinned significantly more than 1 in 1000, so the time saved can be substantial, especially if the `mceval_phase` calculations are time-consuming.

While the `mceval` phase was designed specifically for MCMC analyses, it can be coopted for use in other types of analyses. In essence it is a convenient framework in which to get ADMB to quickly evaluate arbitrary sets of parameters, while only initializing once. Examples of alternative uses are the SIR algorithm, evaluating a grid of points for plotting and exploration of the posterior surface, or trying random parameter sets to investigate local minima. Getting ADMB to evaluate these parameter sets is as simple as writing them to the `.psv` file and then executing ADMB with the option `-mceval`. See section (4.3) for details on how to do this.

4.3 Output files

4.3.1 Meta data: The hst file

A file named `<model name>.hst` is produced which contains the marginal distribution of each object of type `sdreport` displayed as two columns under the header of the associated `sdreport` variable. This distribution is created by binning the observed values of each `sdreport` variable, over the entire MCMC chain, into a histogram and then scaling the area under the curve to equal 1. In the `.hst`, the first column reports the mid-point of each bin and the second column reports the associated density.

The structure of the `.hst` file is organized so that if a chain is restarted using the `-mcr` option, all values needed to bin the subsequent MCMC samples into the original histogram are automatically read, so that the values in the final `.hst` file reflect the observed values, and therefore the marginal distribution, of the complete chain. The firt value reported in the `.hst` file is the total number of iterations of the MCMC chain used to create the reported marginal distribution.

An important note about using the `.hst` file: The reported means and standard deviations are necessary to the internal algorithm that creates each `sdreport` variable's histogram and are calculated using only a small number of the initial iterations of the chain. Therefore these values are inappropriate to report as model results. The user has two ways to obtain the posterior median and credible intervals for each `sdreport` variable: the user can manually export the marginal distribution from the `.hst` file into an input file external to ADMB (e.g. `.csv` to `R`) and calculate the quantiles, etc., or the user can write each `sdreport` variable's (saved or total) draws during the `mceval` phase to a `.csv` using the information in the 'derived quantity draws' section. This `.csv` can then be read into `R` and manipulated, using the code below, to produce a histogram of the observed values.

Melissa todo: Include R code to re-create the histogram and report the median and CI's using the user defined .csv.

4.3.2 Parameter draws: The psv file

During the `mcmc` phase, saved parameter values, in bounded space, are written to a binary file called `<model name>.psv`. This file can be read into R using the following commands:

```
psv <- file("<model name>.psv", "rb")
nparams <- readBin(psv, "integer", n=1)
mcmc <- matrix(readBin(psv, "numeric", n=nparams*<Nout>), ncol=nparams, byrow=TRUE)
close(psv)
```

The first element in the `.psv` file is the number of active parameters in the model, which then tells R how to parse the following elements into parameter values. Note that the value of $Nout$ in `nparams*Nout` depends on `Nmcmc` and `mcsave` and must be specified manually. This is the main file that was designed to be used to extract MCMC draws from ADMB. However, this file only contains parameter values and not derived quantities or other quantities of interest (e.g. MSY or biomass trajectories) which often are of interest.

4.3.3 Derived quantity draws

Often the posterior distribution for quantities other than the parameters are desired. Examples of derived quantities are: functions of parameters, properties of the model, or model projections/extrapolations.

A simple way of extracting this information is to bypass the `psv` file altogether and use a `C++` function to write a `.csv` file containing whatever elements are desired. This can be accomplished inside the ADMB `.tpl` file with just a few lines of code. Inside the `DATA_SECTION` section use the following code to create an IO object that writes values to a `.csv` file, similar to the function `cout` which prints to screen.

```
  !!CLASS ofstream MCMCreport("MCMCreport.csv",ios::trunc);
```

Then, inside the `PROCEDURE_SECTION` the function can be used to write both parameters, derived quantities, or other information about the model.

```
  if(mceval_phase()){
    if(header==1) {
        MCMCreport << "a,b,NLL,ab" << endl;
        header=0;
    }
    MCMCreport << a <<"," << b << "," << NLL << "," << ab << endl;
  }
```

In this case the parameters a and b are saved, along with the negative log-likelihood and product of the model parameters. The `MCMCreport` object is used just like `cout` and is executed only during the `mceval` phase so that only saved values are written to the file as ADMB iterates through each saved parameter. The variable header is set to 1 in the `GLOBALS_SECTION` as `int header = 1;`. Naturally this code can be used anywhere in the procedure section, and this may be a useful diagnostic tool in some situations. The file is recreated at each execution if `ios::trunc` is used. Alternatively, new draws are appended to the bottom of the `.csv` with `ios::app`; this option is particularly useful when restarting an MCMC chain.

4.4 Convergence diagnostics

Burn-in and thinning, how to check this? What happens if not independent?

4.5 Resuming a chain

The MC resume command, `-mcr`, allows the user continue sampling along a previuosly run MCMC chain. Say that the user has run a chain, checked diagnostics, and concluded that the chain needs to thinned further as samples appear to be correlated. Rather than restarting a new chain, the user can pick up an existing chain where it left off with the command `-mcr`. If further thinning is needed, the user can continue an existing chain and thin after the chain is complete. Another benefit of `-mcr` is that the user can run a short chain, check diagnostics, then continue the chain if the thinning rate looks appropriate.

Resuming a chain is simple, and only requires adding `-mcr` to an MCMC statement. Let's save ten values from a chain, saving every 20th value with a seed of 30.

```
simple -mcmc 200 -mcsave 20 -mcseed 30
```

Now say that we want to resume the chain, still saving every 20 values until we have 20 saved parameter draws. In order to save ten additional draws we will run 200 more additional iterations by adding `-mcr` and `-nosdmcmc` (see Section 2.1 in the ADMB manual) will achieve this.

```
simple -mcmc 200 -mcsave 20 -mcr -nosdmcmc
```

One nuance of `-mcr` is that starting then resuming a chain does not result in the same samples as a long chain. For example if we run 400 iteration s of a chain, saving every 20, and setting the seed at 30 with the command:

```
simple -mcmc 400 -mcsave 20 -mcseed 30
```

we end up with different values as the started and resumed chain (Table 1). Users should note the differences between the last half of the resumed chain and the long chain. This disparity can have important implications for reproducibility.

Resuming chains with `-mcr` can reduce computation time. Adding the comm and `-noest` can save further computation time as minimization is unnecessary for resumed chains.

a-first	b-first	a-resumed	b-resumed	a-long	b-long
1.91	4.08	1.91	4.08	1.91	4.08
2.00	4.78	2.00	4.78	2.00	4.78
2.08	3.40	2.08	3.40	2.08	3.40
1.85	4.35	1.85	4.35	1.85	4.35
1.88	4.31	1.88	4.31	1.88	4.31
1.86	4.94	1.86	4.94	1.86	4.94
1.89	2.18	1.89	2.18	1.89	2.18
2.62	0.78	2.62	0.78	2.62	0.78
2.16	2.99	2.16	2.99	2.16	2.99
1.94	3.49	1.94	3.49	1.94	3.49
0.00	0.00	0.33	-1.25	2.10	2.71
0.00	0.00	2.31	1.02	1.90	4.38
0.00	0.00	1.93	3.36	1.91	4.00
0.00	0.00	1.73	5.22	2.08	4.00
0.00	0.00	1.59	6.24	1.59	5.48
0.00	0.00	2.16	3.08	1.72	5.47
0.00	0.00	1.83	3.24	1.66	5.69
0.00	0.00	1.95	4.12	2.26	3.02
0.00	0.00	1.76	4.55	1.74	4.69
0.00	0.00	1.82	5.21	2.38	2.51

Table 1: Parameter draws from the "simple" example. The first columns are from the first half of a chain. The resumed chains contain 10 values from the second half of a chain. The long chain is a long chain that has not been resumed with `-mcr`.

4.6 Convergence diagnostics

Burn-in and thinning, how to check this? What happens if not independent?

4.7 Starting values

For many Bayesian software platforms, the MCMC algorithms are started at user-chosen or arbitrary places. ADMB has the advantage that it can robustly estimate the posterior mode and the covariance at that point. This information is very valuable in initializing the MCMC chain.

Specifically, an MCMC chain starts from the posterior mode and uses the estimated covariance matrix in its proposed jumps (see the algorithm sections below). As such, ADMB chains typically do not need a long period to reach areas of high density.

The user can specify starting values via the command line option `mcpin <file>` which must point to a file with starting values for parameters.

5 Metropolis-Hastings MCMC

The default MCMC algorithm used by ADMB is the Metropolis-Hastings (MH) algorithm[1]. This algorithm has been around for decades, is simple to implement and used widely.

This algorithm will be most efficient when the posterior surface mimics a multivariate Normal distribution.

5.1 Algorithm

Let

$$\begin{aligned} f &= \text{the ADMB objective function} \\ c &= \text{an unknown normalization constant} \\ Xcur &= \text{current parameter vector} \\ Xprop &= \text{a proposed parameter vector} \\ U &= \text{a randomly drawn uniform value in [0,1]} \end{aligned}$$

Then

$$Xnew = \begin{cases} Xprop & \text{if} \quad U \leq \dfrac{cf(Xprop)}{cf(Xcur)} \\ Xcur & \text{otherwise} \end{cases} \tag{1}$$

The proposal (or "jump") function proposes new parameter vectors given the current set. The default behavior for ADMB is to use a multivariate normal distribution[2] centered at the current vector:

$$Xprop \sim MVN(Xcur, \Sigma)$$

where Σ is the covariance matrix obtained by inverting the Hessian at the posterior mode.

5.2 MCMC Arguments

ADMB has a suite of arguments available to the user at run time. We cover the most commonly used ones, and refer the user to the ADMB manual and other documentation for further options.

5.3 Console output

When running an MCMC chain from the console, the output similar to the following will appear.

```
Initial seed value -36519
-14.9642 -14.9642
 mcmc sim 1  acceptance rate 0 0
```

[1]Technically it has a symmetric proposal, a special case known as the Metropolis algorithm.

[2]Technically a bounded multivariate normal, operating in a transformed parameter space via the Cholesky decomposition of the covariance matrix, but we ignore that here for simplicity and clarity.

`-mcmc N`	Run N MCMC iterations
`-mcsave N`	Save every Nth MCMC iteration
`-mcscale N`	Rescale step size for first N iterations
`-mcmult N`	Rescale the covariance matrix
`-mcpin <file>`	Start algorithm from values in ¡file¿
`-mcrb N`	Reduce high parameter correlations (see 5.5.4)
`-mcprobe X`	Use a fat-tailed proposal distribution (see 5.5.3)
`-mcdiag`	Use a diagonal covariance matrix
`-mcnoscale`	Do not scale the algorithm during
`-mcscale N`	Scale the algorithm during the first N iterations
`-mcu`	Use a uniform distribution as proposal distribution.

Table 2: ADMB runtime arguments for the Metropolis-Hastings MCMC

```
-15.4843 -15.4843
 mcmc sim 201  acceptance rate 0.517413 0.52
increasing step size -1.32761
-15.7798 -15.7798
 mcmc sim 401  acceptance rate 0.428928 0.34
```

ADMB prints the random seed used, and then the negative objective function value twice. In this case scaling is increased after the first 200 iterations (proposing more distant values). This is followed by the log of the determinant of the Choleski decomposed covariance matrix. In some cases this information may be useful for troubleshooting, but for most practical purposes it can be ignored.

5.4 Example MCMC

We demonstrate these concepts using the "simple" model packaged with ADMB, which is a contrived two parameter linear model ($y = ax + b$). First we run the model without any thinning by specifying `mcsave`=1 (figure 1). The MCMC chain is clearly autocorrelated from the acf plots and the traces (not show). This chain needs to be tuned to achieve independent samples.

5.5 Tuning the MH algorithm

The goal of "tuning" the algorithm is to obtain independent samples from the posterior, with as few of calculations as possible (to keep runtime manageable). For the MH algorithm, there are several ways the user can tune a chain.

5.5.1 Thinning rate

For the MH algorithm, the most important tuning option available to the user is the thinning rate. This is the rate at which parameters are saved, such that thinning is effectively discarding draws. This tuning option is critical since this algorithm generates autocorrelated parameters, by design.

The user controls the thinning rate by the argument `mcsave N`. If $N = 1$, as in figure 1, every single draw is saved. As we saw with that example, the autocorrelation is high, suggesting the need to thin more (save fewer).

We now rerun the chain with `mcsave`=100 (figure 2), in effect running 100 times the samples and saving every 100^{th}. This helps reduce the autocorrelation and produces independent draws from the posterior of interest.

5.5.2 Opitmize the acceptance rate

Studies have shown that there is an optimal range for acceptance rate for the MH algorithm (e.g. [**?**]). If the proposal distribution generates values too close to the current state, the chain will accept them (high acceptance rate) but explore the posterior slowly and need more thinning. Alternatively, if proposals are too

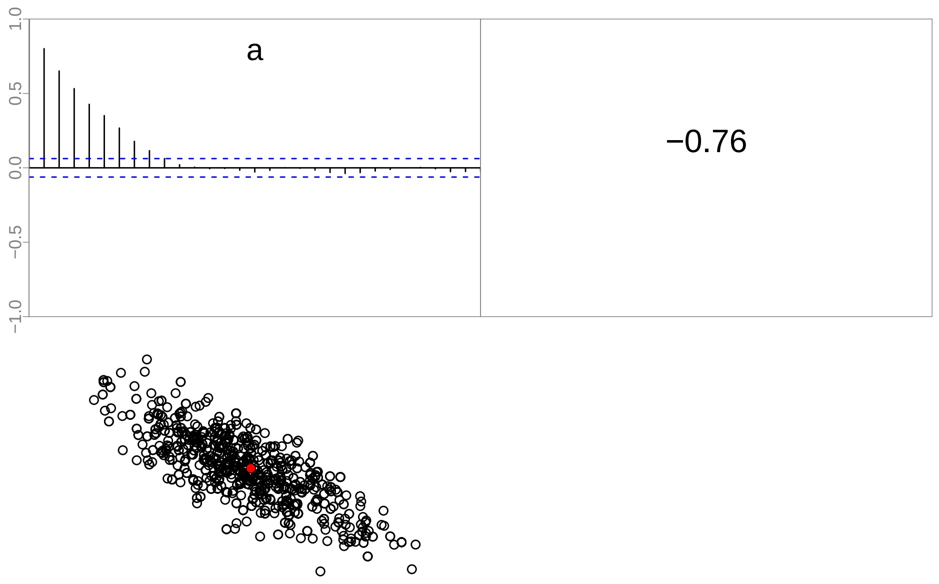

Figure 1: The samples from a simple model with `mcsave`=1. Note the high autocorrelation of both parameters.The red ellipses show the estimated pairwise parameter covariances, and the red point the MPD.

far away into regions of low density (low acceptance rate) the chain will not explore the space. The optimal acceptance rate varies by model, but is roughly 40%. The general advice is to tune the proposal distribution to achieve an efficient acceptance rate.

ADMB accomplishes this by "scaling" the covariance matrix up or down, depending on the current acceptance rate, during the first part of the chain. By default, it scales during the first 500 iterations (and prints this to screen), but the user can specify this with `mcscale N` or turn off scaling with `mcnoscale`. ADMB rescales the covariance matrix every 200 iterations until the acceptance rate is between 0.15 and 0.4, or the scaling period is exceeded.

In practice, the defaults work for many models, but the user may want to extend the scaling period for some models. Draws from this tuning phase should be discarded as part of the burn-in.

5.5.3 mcprobe

[if someone wanted to write this up more formally that'd be great] For some models, there may be concern of being "stuck" in a local minimum and simply never proposing a value far enough away to escape it and find other regions of high density. ADMB has a built-in algorithm which modifies the default proposal distribution so it occasionally proposes very distant parameters (i.e. "probes"). The `mcprobe X`[3] argument initiates this option.

The modified proposal distribution is a mixture distribution of normal and Cauchy distributions. The argument `X` controls how the two distributions are mixed, with larger values being more Cauchy (fatter tails) and lower being thinner tails. The range of valid inputs is 0.00001 to 0.499, and if no value is supplied a default of 0.05 is used[4]

Figure 3 shows the shape of the proposal distribution. Note the extremely fat tails, and that the standard proposal itself is bounded.

[3]Previous versions of ADMB called this `mcgrope` – this is now deprecated

[4]See the function for more information: `http://admb-project.org/documentation/api/prmonte_8cpp_source.html#l00014`

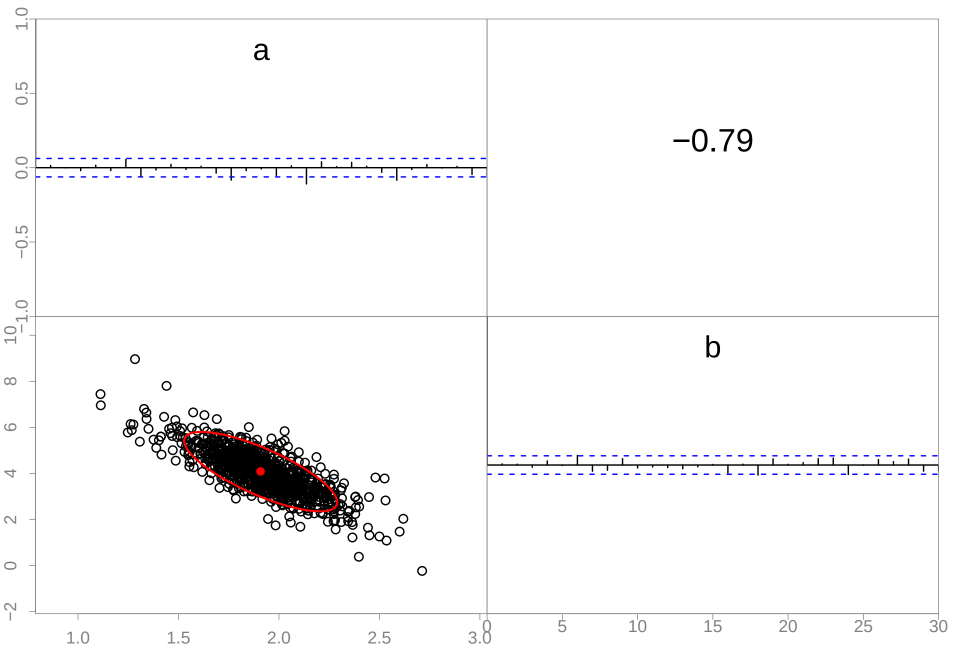

Figure 2: The samples from a simple model with `mcsave`=100. Note the negligible autocorrelation of both parameters.The red ellipses show the estimated pairwise parameter covariances, and the red point the MPD.

5.5.4 mcrb

The `-mcrb N` option (which stands for rescaled bounded) alters the covariance matrix used to propose new parameter sets in the MH algorithm. Its intended use is to create a more efficient MCMC sampler so the analyses run faster.

The option will be most effective under circumstances where the correlation between parameters at the MPD is higher than other regions of the parameter space. In this case, the algorithm may make efficient proposals at the MPD, but inefficient proposals in other parts of the space. By reducing the correlation using `mcrb` the proposal function may be more efficient on average across the entire parameter space and require less thinning.

The `mcrb` option is a set of calculations performed on the original correlation matrix, as

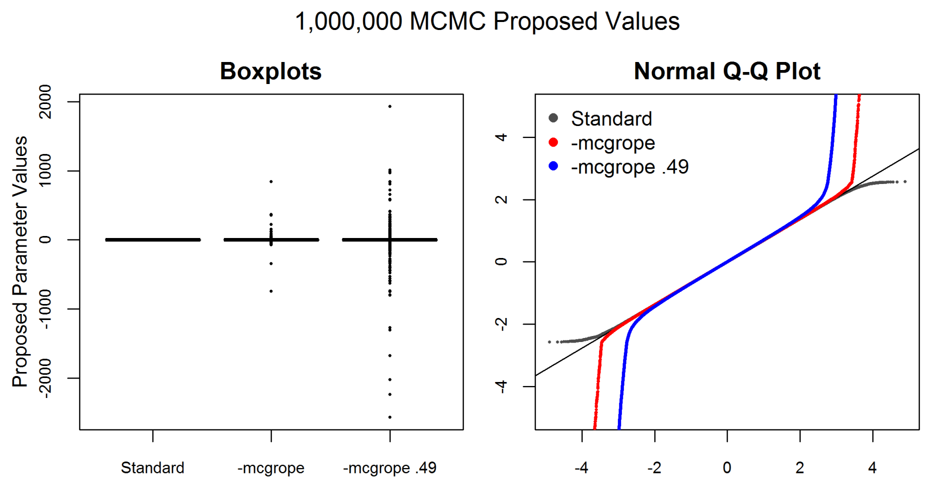

Figure 3: Example of fat-tailed proposal values for a parameter for the `mcprobe` option compared to the default proposal.

follows.

$$
\begin{aligned}
\mathbf{\Sigma}_{\text{old}} &= \begin{bmatrix} 1 & \cdots & \rho_{1,n} \\ \vdots & \ddots & \vdots \\ \rho_{n,1} & \cdots & 1 \end{bmatrix} && \text{The original correlation matrix} \\
\mathbf{L} &= \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ L_{n,1} & \cdots & L_{n,n} \end{bmatrix} && \text{Lower Choleski decomposition of } \mathbf{\Sigma}_{\text{old}} \\
\hat{\mathbf{L}} &= \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ L_{n,1}^{N/10} & \cdots & L_{n,n}^{N/10} \end{bmatrix} && \text{Raise elements to power user supplied } N \\
\tilde{\mathbf{L}} &= \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \frac{\hat{L}_{n,1}}{|\hat{L}_{n,\cdot}|} & \cdots & \frac{\hat{L}_{n,n}}{|\hat{L}_{n,\cdot}|} \end{bmatrix} && \text{Normalize rows of } \hat{L} \\
\mathbf{\Sigma}_{\text{rb}} &= \tilde{\mathbf{L}}\tilde{\mathbf{L}}^T \quad \text{Calculate new correlation matrix}
\end{aligned}
$$

By working with the Choleski decomposition of the correlation matrix, the algorithm ensures that the rescaled bounded matrix used in the MCMC remains a valid correlation matrix (i.e. positive definite). The impacts on a correlation matrix can be difficult to anticipate, but fortunately ADMB writes the resulting covariance matrix to the file `corrtest` (a text file with no ending) under the label "modified S". The user can plot this against the estimated covariance to visually gauge the impact of the `mcrb` algorithm.

5.5.5 User-supplied covariance matrix

The `mcrb` option is a quick way to try lowering the correlation between some parameters. A more flexible (and transparent) option is to provide ADMB with a covariance matrix chosen

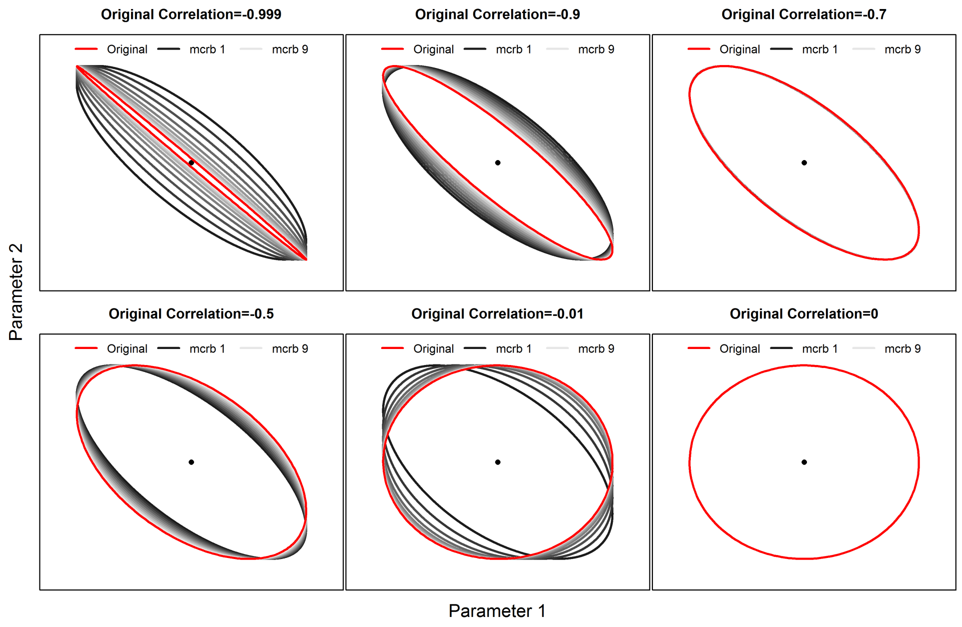

Figure 4: The effect of `mcrb` on a variety of correlations between two hypothetical parameters. Note that the effect of setting $N = 9$ depends on the original correlation.

by the user. By examining preliminary chains or using information gained from other simple models, it may be possible for the user to define a covariance matrix that is more efficient than the estimated one, or modified via the built-in options. We demonstrate the improved perofrmance using this option in the next section.

Providing user-supplied covariance is a more advanced technique that is not built in to ADMB. It requires a slightly more detailed understand of the underlying output file structure of ADMB. The steps to do this can be found in the document at `http://www.admb-project.org/examples/admb-tricks/covariance-calculations`.

6 Hyrbid MCMC

The "hybrid" option in ADMB is an implementation of an MCMC algorithm based on Hamiltonian dynamics. Here we provid a simplified overview [5] of the algorithm, with the aim of providing users an intuition about its behavior and properties and how to use it within ADMB. This section is based on [**?**], which provides a thorough review of the algorithm, including background motivation, proof of ergodicity, and illustrative examples[6].

The hybrid method is different from the MH algorithm in how it proposes new parameter values. Instead of proposing random states based on the current value, the hybrid method uses derivatives to follow a contour of the posterior surface. By doing so, it (in theory) only proposes states that are very likely to be accepted, and as such will have less autocorrelation.

In some models, a well tuned hybrid chain will need less thinning, if any at all, and run faster. The downside of the algorithm is that it is more difficult to tune than the MH algorithm.

[5]Ignoring, for example, the transformation of the parameter space via the Choleski decomposition used by ADMB
[6]This chapter is available at `http://www.admb-project.org/developers/workshop/la-jolla-2010/ham-mcmc.pdf/view`

6.1 Algorithm

The algorithm utilizes the properties of a physical system known as Hamiltonian dynamics. Hamiltonian dynamics, while based in physics, provides some extremely useful mathematical properties for Bayesian integration via MCMC.

A Hamiltonian system consists of two parameter vectors of equal length: "position" ($\mathbf{q}$) and "momentum" ($\mathbf{p}$). How these parameters change over time is described by the Hamiltonian function, $H(\mathbf{q}, \mathbf{p})$. This system can be conceptualized as a frictionless surface about which an object moves. At some time t an object has a certain height (position) and momemtum. The height of the surface is equal to the objective function of our model, and the momentum variables are introduced parameters to ensure the Hamiltonian dynamics are met. Samples from a posterior are generated by simulating the object moving about the joint surface through time, governed by H.

For use with MCMC, H is assumed to be $H(\mathbf{q}, \mathbf{p}) = U(\mathbf{q}) + K(\mathbf{p})$, where U is analagous to the potential energy and K kinetic energy. U is set equal to the ADMB objective function (i.e. the negative log of the posterior density) and K to a diagonal multivariate normal distribution. The hybrid MCMC algorithm samples from the joint posterior, H, but we are only interested in the posterior for U, so K is not saved by ADMB.

The time trajectory of the object through the joint probability space, H, is used to generate proposed parameter sets. Given the form for H above, the fundamental equations of motion are:

$$\frac{dq_i}{dt} = \frac{\partial K}{\partial p_i} \tag{2}$$

$$\frac{dp_i}{dt} = -\frac{\partial U}{\partial q_i} \tag{3}$$

ADMB uses the "leapfrog" method to discretize equations (2). The leapfrog method is more reliable than the well-known Euler method. It has two tuning parameters: the number of steps to take (`hynstep`), and the step size ε (`hyeps`). The following sequence of calculations shows a single iteration of the leapfrog method, for the i^{th} variable, and is repeated `hynstep` times sequentially.

$$p_i(t + \varepsilon/2) = p_i(t) - (\varepsilon/2)\frac{\partial U}{\partial q_i} q(t)$$

$$q_i(t + \varepsilon) = q_i(t) + \varepsilon \frac{p_i(t + \varepsilon/2)}{mi}$$

$$p_i(t + \varepsilon) = p_i(t + \varepsilon/2) - (\varepsilon/2)\frac{\partial U}{\partial q_i} q(t + \varepsilon)$$

The leapfrog algorithm moves deterministically through the joint surface along a contour of (appoximately) constant H. That is, for a given starting value of (q_0, p_0) and tuning parameters the trajectory will always end in the same place. Examples of trajectories under different tuning parameters for the leapfrog method are given in figure 5. Note that ADMB calculates the partial derivatives $\frac{\partial U}{\partial q_i}$ at each function evaluation using automatic differentiation. Thus these are available for any model and do not have to be determined analytically, which can be difficult or impossible for large, non-linear models.

Casting a posterior into a Hamiltonian system and discretizing it with the leapfrog method is simply a way to generate proposed sets of parameters in the larger, stochastic MCMC

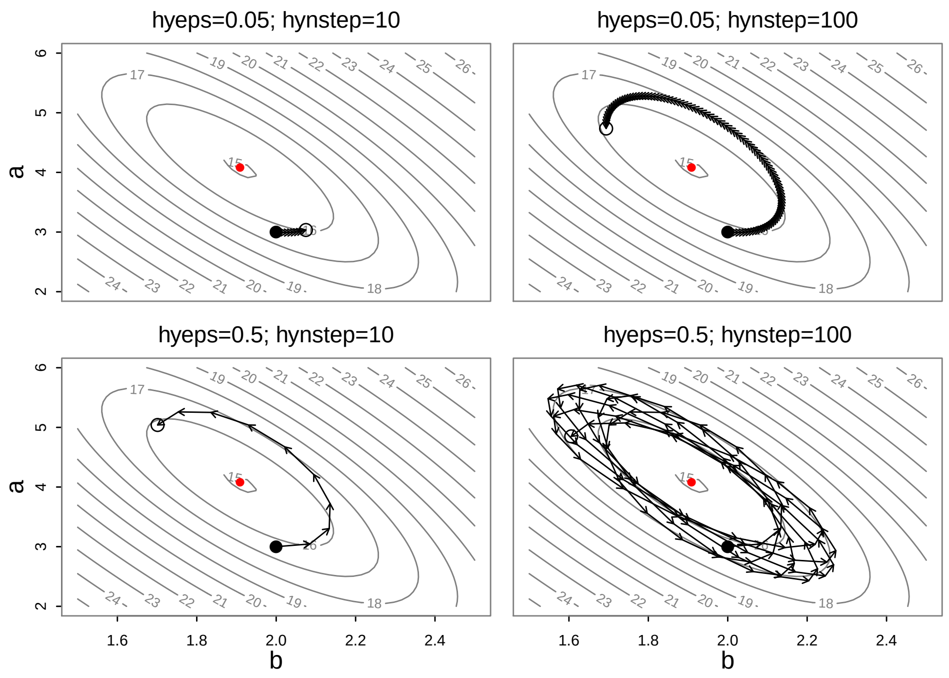

Figure 5: Leapfrog trajectories for different sets of tuning parameters. The posterior surface is shown as contours, and the posterior mode as a red point. The filled black point is the starting point, and the arrows show the trajectory of the leapfrog steps, ending at the open circle representing the proposed set of parameters.

algorithm. A single iteration of the hybrid MCMC algorithm, as implemented in ADMB, has three steps:

1. **Propose new momentum variables.** New momentum values, p^* are generated from a normal distribution based on the estimated covariance matrix, and independent of the current position variables.

2. **Propose new position variables.** Given the current state of the system, (q, p^*), new position variables q^* are generated with the leapfrog algorithm using `hynstep` [7] steps and a step size of `hyeps`.

3. **Accept or reject the new state.** The new state is then updated with a Metropolis step (i.e. accepted or rejected) in the same way as above. Acceptance is expected to be high because the proposed values (q^*, p^*) are rarely into regions of low density for a well tuned chain.

We can see the first two steps of the algorithm by randomly drawing values for U and doing a leapfrog projection to the proposed parameters (Fig. 6).

A perhaps intuitive question might be: why bother with the momentum variables at all? One issue is that without the momentum variables, and the Hamiltonian dynamics in general, we would need to account for changes in volume in the acceptance probability (step 3 above)[**?**]. This would require computing the determinant of the Jacobian matrix of the mapping defined by the dynamics. This Jacobian is not readily available and is

[7] Actually a random integers is generated around this value

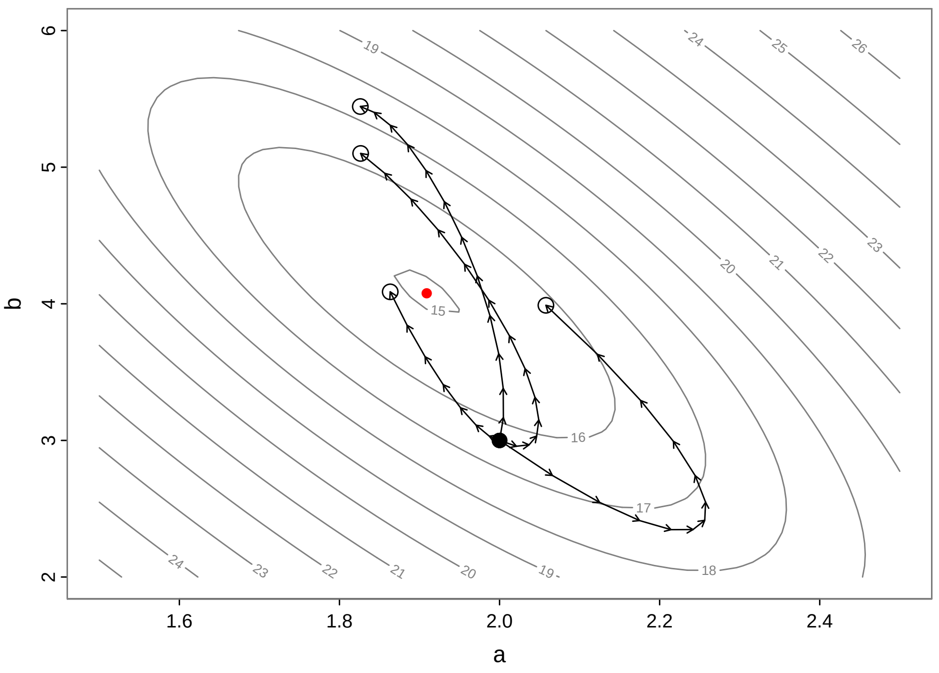

Figure 6: Leapfrog projections for different random draws from U.

often computationally intensive. Thus the need to adopt the Hamilton dynamics framework. Hamiltonian dynamics also provides other properties required for an ergodic Markov chain [?].

The hybrid MCMC thus samples from the joint posterior of the position and momentum vectors, but ADMB discards the momentum variables and returns only the position variables. The user can then do inference on those samples in the same was as the Metropolis-Hastings method, given they come from a properly converged (stationary) chain.

6.2 Arguments

Arguments for the hybrid algorithm are in general similar to those above, so we only discuss the differences. The main arguments are:

`-mcmc N`	Run N MCMC iterations
`-hybrid`	Use the hybrid method
`-hynstep N`	Mean number of steps for the leapfrog method
`-hyeps X`	The stepsize for the leapfrog method [X numeric and > 0]

Table 3: ADMB runtime arguments for the hybrid MCMC

Since the estimated covariance matrix is used in the algorithm, the `mcdiag`, `mcrb N`, and `mcmult` options above are also available. The `mcprobe` argument is not currently supported for the hybrid algorithm.

Note that `mcsave N` is **not** an argument for the hybrid and will be ignored. Each MCMC iteration of the hybrid algorithm is saved, such that the user may need to thin the chain manually after running.

6.3 Tuning the hybrid algorithm

A tuned hybrid MCMC algorithm often provides a more efficient (computationally) chain than the random walk behavior of the Metropolis-Hastings algorithm. However, the hybrid algorithm is often much more difficult to tune. A thorough review of tuning techniques is beyond the scope of this guide, and we refer users to [?] for further reading of more advanced tuning techniques and intuition about reasonable values[8].

In most practical applications the hybrid method is tuned via trial and error. That is, values for `hynstep` and `hyeps` are tried on a short chain, and the output diagnosed. ADMB uses default values of 0.1 and 10, respectively, and these may be good starting values for most problems. The end goal is to find a set of tuning parameters that produces a well mixing chain with the fewest leapfrog steps. Figure 7 shows the autocorrelation of a parameter from the simple model across different tuning parameters. Note that the top right and bottom left panels show very similar ACF patterns, which should not be a surprise when compared to the corresponding leapfrog trajectories in 5. Further, from that figure we can see that a chain with a `hynstep` of 100 cycles the surface many times, and although it mixes well, this value could probably be lowered and the runtime reduced without affecting mixing. This is somewhat analagous to having a higher thinning rate than is necessary.

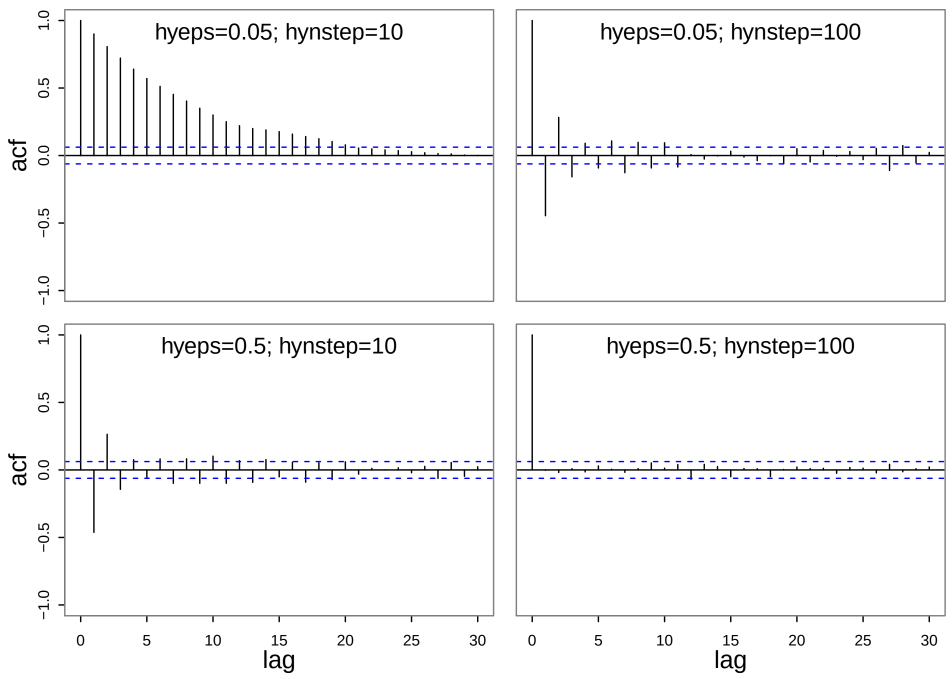

Figure 7: The autocorrelation of the parameter a from the simple model across different tuning parameters of the hybrid method.

A word of caution regarding the tuning of hybrid chains. For most values of `hyeps` the leapfrog trajectories will be "stable" in that they will cycle around a contour of H (this can be seen in the last panel in figure 5). However, for values of `hyeps` that are too large, the method becomes unstable and trajectories diverge (H is no longer constant), causing the

[8]The ADMB algorithm works in the transformed space, via the Cholesky decomposition of the estimated covariance matrix, making intuition about tuning even more difficult

algorithm to propose parameters with low density which are then rejected. Unfortunately for real problems, the value of `hyeps` that is "too big" can vary across the posterior space. Another issue that can arise is that certain combinations of tuning parameters can lead to near periodicity. That is, the leapfrog trajectory ends very near to where it began, after one or more steps. In pathological cases it could cycle forever and never be able to sample regions of the posterior space (i.e. not be ergodic). In practice near periodicity will make for a very slow mixing chain. ADMB mitigates this possibility by randomly drawing the number of leapfrog steps at each MCMC iteration. However, the user should still be aware of these potential issues and be vigilant in diagnosing the mixing behavior of a hybrid chain. Samples from an MCMC chain that is not in equilibrium, or has other issues, can lead to incorrect inference.

7 An example with non-linear posterior

The simple example used above had the characteristic that the parameters were correlated, but this correlation was constant across the entire posterior. Thus the same proposal distribution worked well throughout the parameter space.

Here we look at another example with two parameters, but where the correlation changes. This example uses contrived data and a discrete theta-logistic model. This is a classic example of a simple model which can lead to convergence issues for MCMC chains.

We first run it with the default settings and a thinning rate of `mcsave`=100.

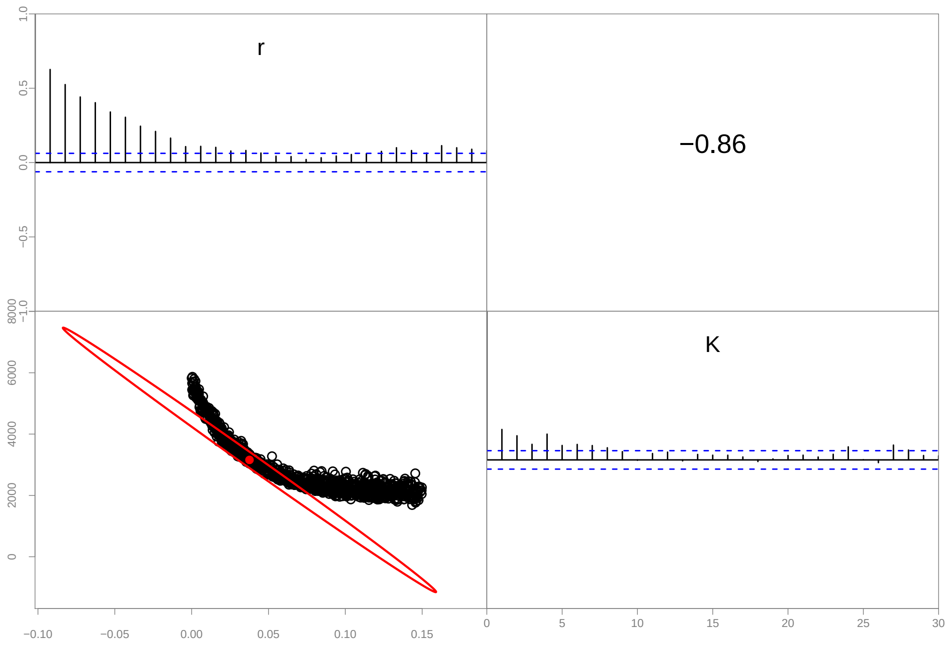

Figure 8: The logistic model with 1 in 100 samples saved using the MH algorithm and estimated covariance matrix (red ellipse).

We can immediately see the autocorrelation remains high even after thinning. One option would be to increase the thinning rate. Another is to try a lower correlation. From this initial run we calculate the empirical covariance matrix and rerun the chain with this option (see 5.5.5).

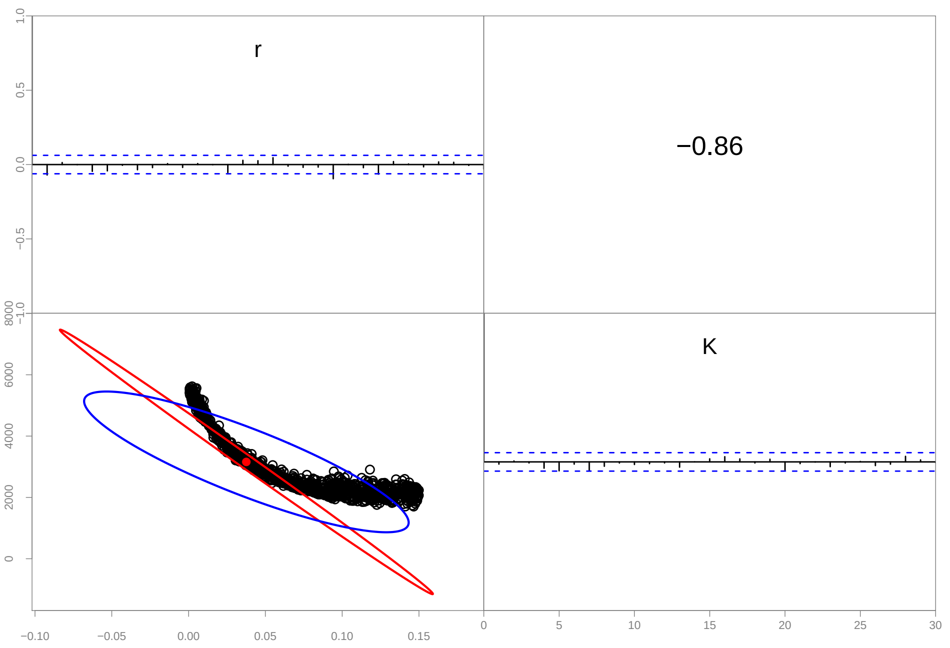

Figure 9: The logistic model with 1 in 100 samples saved using the MH algorithm and empirical covariance matrix (blue ellipse). Note the improvement in acf compared to 8.

Here we see a substantial improvement in the chain performance by changing the proposal distribution of the Metropolis-Hastings algorithm. This option is far superior to increasing the thinning rate because it takes no more time to run.

Here we see a substantial improvement in the chain performance by changing the proposal distribution of the Metropolis-Hastings algorithm. This option is far superior to increasing the thinning rate because it takes no more time to run.

¿¿¿¿¿¿¿ 155bf2dbf5b9ed4efac0735a0cb9e05388b4e584 We can also try the hybrid algorithm on this model, starting with the default parameters of `hyeps` and `hynstep` of 0.1 and 10, and using the estimated covariance matrix. The autocorrelation is high, suggesting we need to tune the chain. First we try the same as above, and instead use the empirical covariance matrix. This chain is performing better, but is still autocorrelated. A little trial and error the combination of `hyeps 0.05` and `hynstep 100` performed significantly better, and similarly to the tuned MH chain in 10.

Here the hybrid and MH perform similarly with the same number of function evaluations per sample (`hynstep` and `mcsave`, respectively). With a little trial and error, and visual exploration of the posterior surface it was fairly straightforward to improve on the default performance. It is likely possible to further refine the hybrid algorithm to outperform the MH, but it is not clear which of the parameters to start with. The difficulty in tuning

For many problems, the default MH algorithm will work well and require no tuning beyond a modest thinning rate. We encourage users to start with this algorithm, and explore alternatives only if computational efficiency is restrictive for the needs at hand. The hybrid method is more sophisticated, but much more difficult to tune, and we recommend it as an

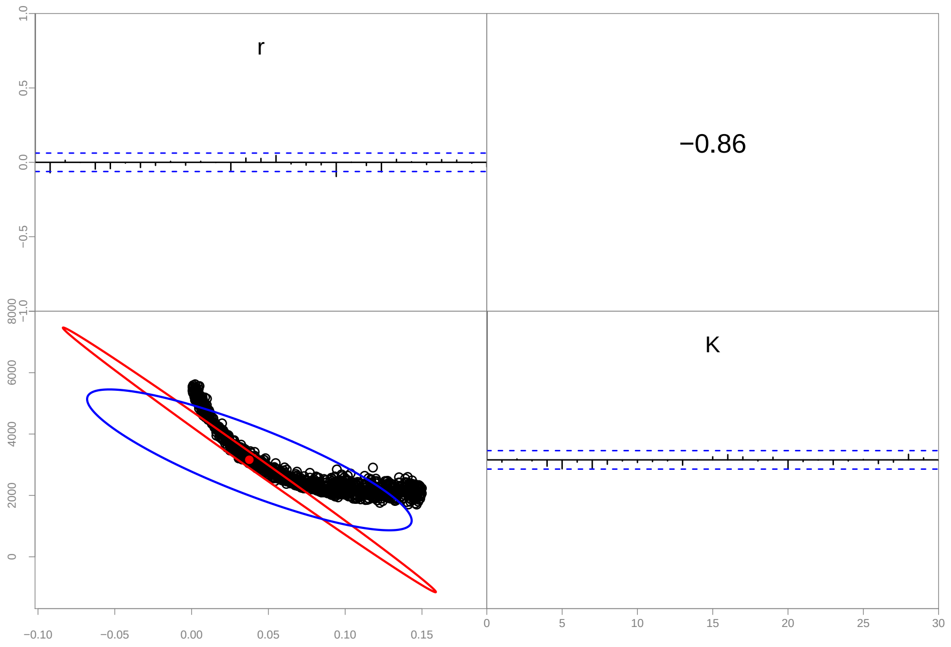

Figure 10: The logistic model with 1 in 100 samples saved using the MH algorithm and empirical covariance matrix (blue ellipse). Note the improvement in acf compared to 8.

option only when the MH is struggling.

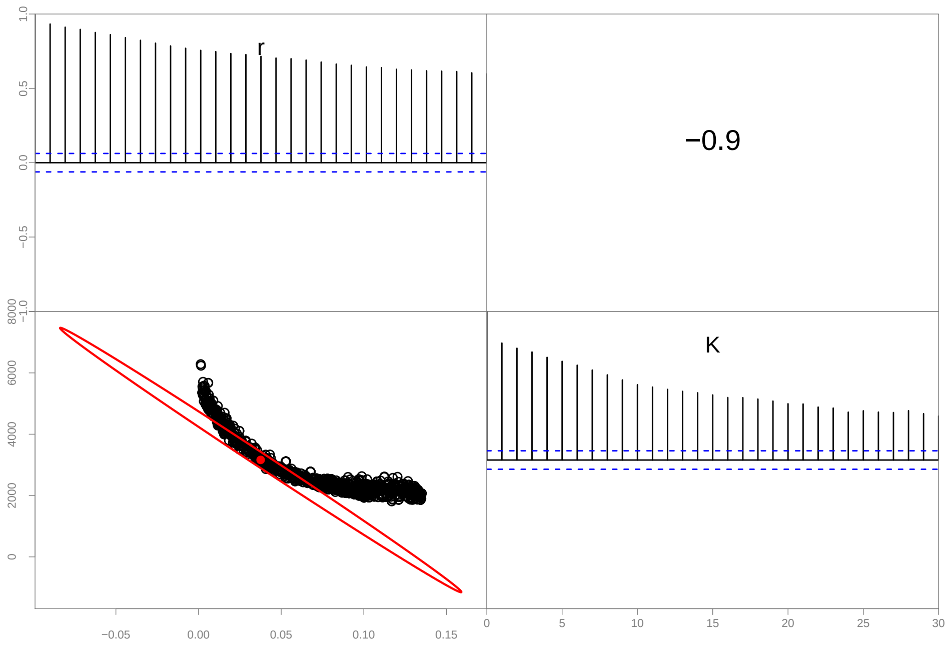

Figure 11: The logistic model using the hybrid method with default tuning parameters and the estimated covariance matrix (red ellipse).

References

[1] G. O. Roberts and J. S. Rosenthal. Optimal scaling for various metropolis-hastings algorithms. *Statistical Science*, 16(4):351–367, 2001.

[2] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of Markov Chain Monte Carlo*. CRC Press, 2011.

.

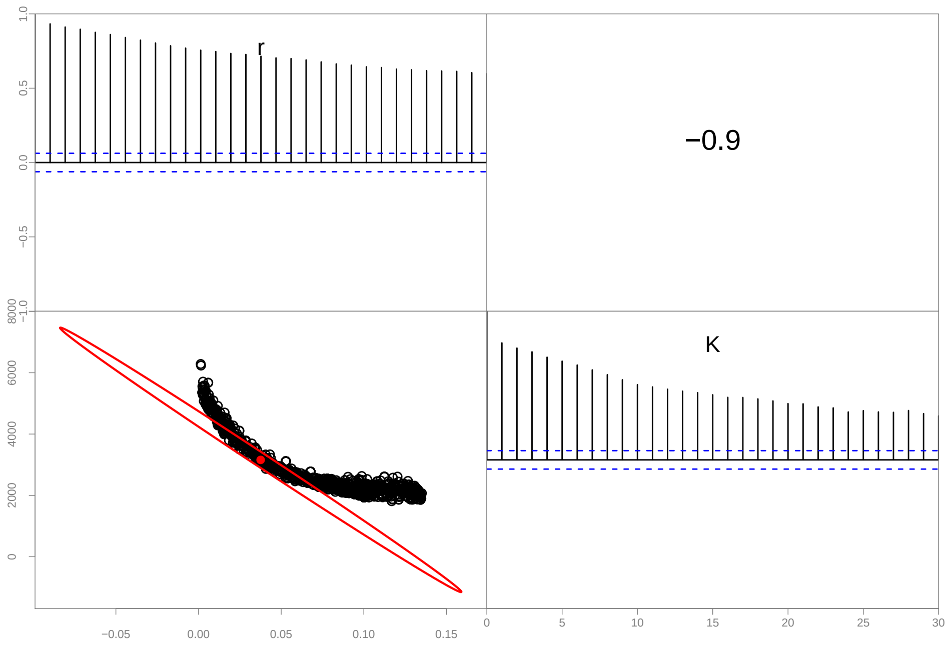

Figure 12: The logistic model using the hybrid method with default tuning parameters and the empirical covariance matrix (blue ellipse).

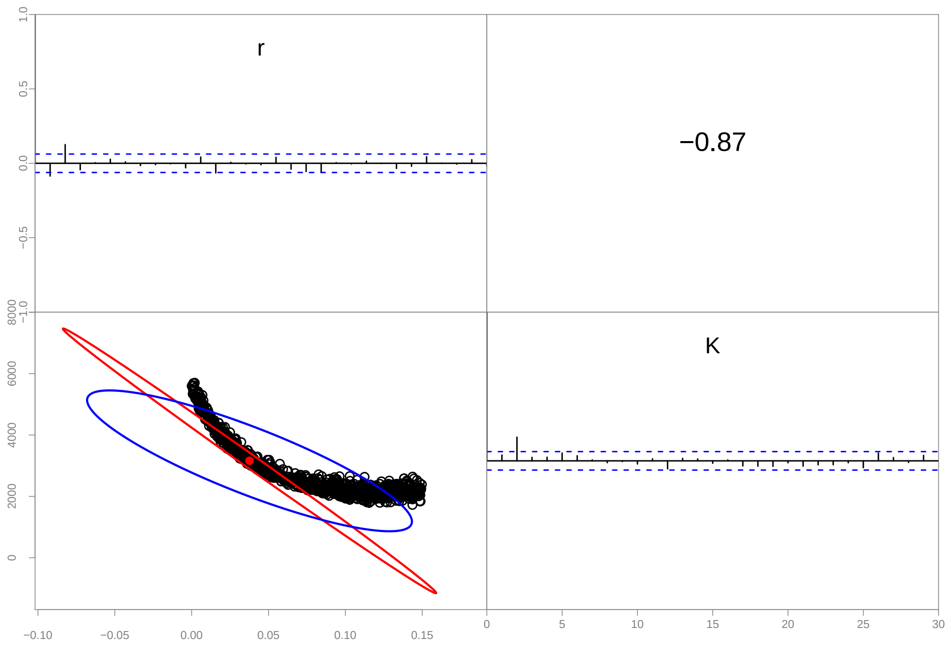

Figure 13: The logistic model using the hybrid method and `hyeps 0.05` and `hynstep 100` for tuning parameters and the empirical covariance matrix (blue ellipse).