A guide for Bayesian analysis in ADMB

Cole C. Monnahan — monnahc@uw.edu

August 5, 2014

Abstract

The point of this document is.....

Contents

1 Bayesian inference

A short intro with citations to better papers. Maybe merge in what is currently shown?

2 MCMC

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 series of autocorrelated parameter vectors which can be thinned to produce independent samples from the posterior distribution of interest.

2.1 Workflow

The following steps outline the basic workflow typically used for conducting a MCMC in ADMB.

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` $N_{\texttt{save}}$ (among other options, see below). 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.

2.2 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 2.7 and 2.8). 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 (2.3) for details on how to do this.

2.3 Output files

2.3.1 Meta data: The hst file

2.3.2 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.

2.3.3 Derived quantity draws

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::app);
```

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;
  }
```

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. Naturally this code can be used anywhere in the procedure section, and this may be a useful diagnostic tool in some situations. New draws are appended to the `MCMCreport.csv` file so that it must be deleted in between MCMC runs.

2.4 Restarting a chain

How to restart a chain if you need more samples. How does this work?

2.5 Convergence diagnostics

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

2.6 Starting values and scaling

Where the algorithm starts from (MLE) and the scaling process (default and user options). Must discard these draws!

2.7 Metropolis-Hastings

The default MCMC algorithm used by ADMB is the Metropolis-Hastings (MH) algorithm. 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.

2.7.1 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[1] 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.

In ADMB there are options to modify the proposal function to achieve better efficiency.

2.7.2 MCMC Arguments

`-mcmc N`	Run N MCMC iterations
`-mcsave N`	Save every Nth MCMC iterations
`-mcscale N`	Rescale step size for first N iterations

Table 1: ADMB runtime arguments for Metropolis-Hastings MCMC

2.7.3 mcprobe

2.7.4 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

[1]Technically a bounded multivariate normal

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 follows.

$$\mathbf{\Sigma}_{\text{old}} = \begin{bmatrix} 1 & \cdots & \rho_{1,n} \\ \vdots & \ddots & \vdots \\ \rho_{n,1} & \cdots & 1 \end{bmatrix} \quad \text{The original correlation matrix}$$

$$\mathbf{L} = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ L_{n,1} & \cdots & L_{n,n} \end{bmatrix} \quad \text{Lower Choleski decomposition of } \mathbf{\Sigma}_{\text{old}}$$

$$\hat{\mathbf{L}} = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ L_{n,1}^{N/10} & \cdots & L_{n,n}^{N/10} \end{bmatrix} \quad \text{Raise elements to power user supplied } N$$

$$\tilde{\mathbf{L}} = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \frac{\hat{L}_{n,1}}{\left|\hat{L}_{n,\cdot}\right|} & \cdots & \frac{\hat{L}_{n,n}}{\left|\hat{L}_{n,\cdot}\right|} \end{bmatrix} \quad \text{Normalize rows of } \hat{L}$$

$$\mathbf{\Sigma}_{\text{rb}} = \tilde{\mathbf{L}}\tilde{\mathbf{L}}^T \quad \text{Calculate new correlation matrix}$$

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).

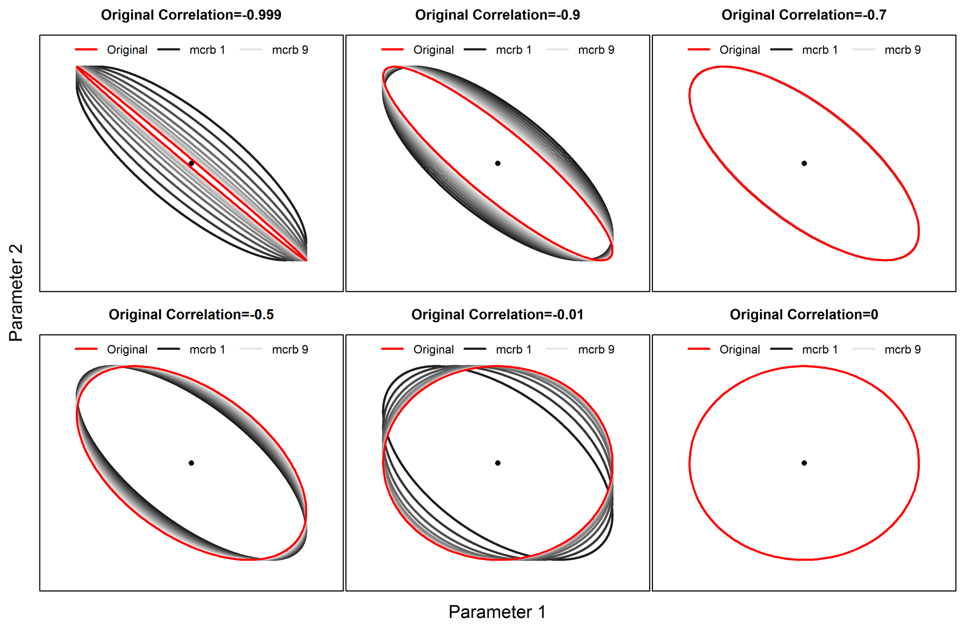

Figure 1: The effect of `mcrb` on a variety of correlations between two hypothetical parameters. Note that the effect of setting $N = 9$ depends on the original correlation.

2.7.5 User-supplied correlation matrix

If the MLE covariance is inefficient at proposing. Why do this? How? Show code for how to do this, and talk about positive definitness.

2.7.6 Example MCMC

Use a simple model to demonstrate some of these concepts. Especially the workflow and examining convergence properties, but also maybe restarting.

2.8 Hyrbid

The "hybrid" option in ADMB is an implementation of an MCMC algorithm based on Hamiltonian dynamics[2]. It is different from the MH algorithm in how it proposes new 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 practice, a well tuned hybrid chain will need less thinning, if any at all. The downside of the algorithm is that it is more difficult to tune than the MH algorithm.

2.8.1 Algorithm

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

A Hamiltonian system consists of two parameter vectors of equal length: "position" (qq) and "momentum" (pp). How these parameters change over time is described by the Hamiltonian function, $H(qq, pp)$. 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 surface through time, governed by H.

For use with MCMC, H is assumed to be $H(qq, pp) = U(qq) + K(pp)$, 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 multivariate normal distribution based on the estimated covariance. 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.

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

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

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

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 end in the same place. Examples of trajectories under different tuning parameters are given in figure 2. Note

[2]See [?], which provides a nice background to the algorithm

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.

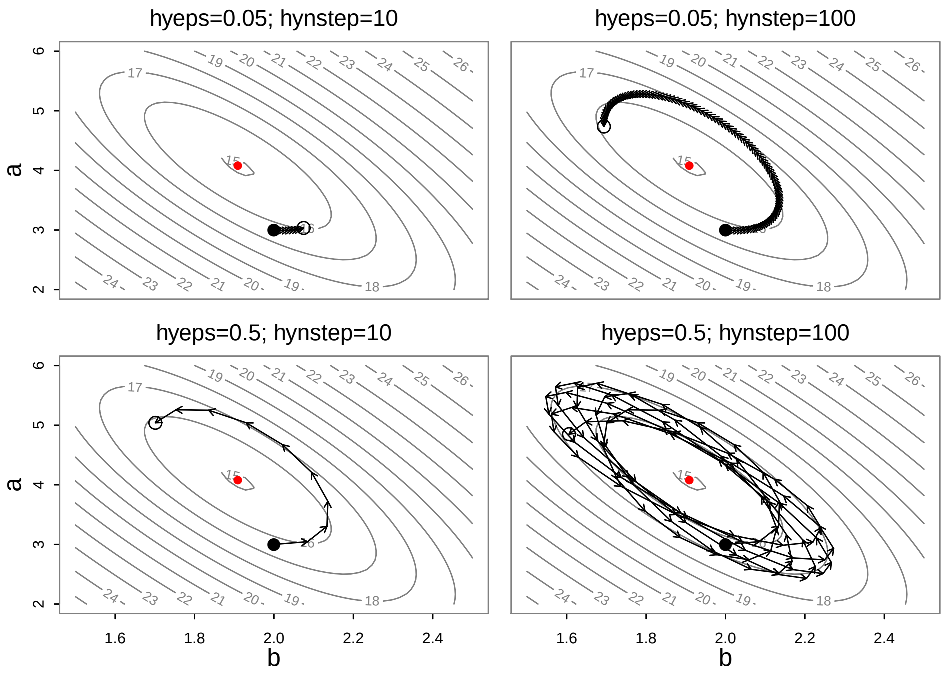

Figure 2: Leapfrog trajectories for different sets of tuning parameters. The posterior surface is shown as contours, and the posterior mode as a red point. The filled black point is the starting point, and the arrows show the trajectory of the leapfrog steps, ending at the open circle representing the proposed set of parameters.

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

1. **Propose new momentum variables.** New momentum values, q^* 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 p^* are generated with the leapfrog algorithm using `hynstep` 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.

A perhaps intuitive question might be: why bother with the momentum variables at all? 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)[**?**][3], which would require computing the determinant of the Jacobian matrix the mapping the dynamics defines. This Jacobian is not readily available and often computationally intensive. Thus the need to adopt the Hamilton dynamics.

The hybrid MCMC thus explores the joint posterior of these two vectors, and we do inference on the position variables while ignoring the momentum posteriors.

[3]among other issues

2.8.2 Arguments

Only show those different than the MH algorithm.

2.8.3 Tuning the hybrid algorithm