A guide for Bayesian analysis in ADMB

Cole C. Monnahan — monnahc@uw.edu

July 18, 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 (3), 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

[1] Technically a bounded multivariate normal

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 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.

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

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

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

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

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

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).

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

2.8.1 Algorithm

Quick explanation of this.

2.8.2 Arguments

Only show those different than the MH algorithm.

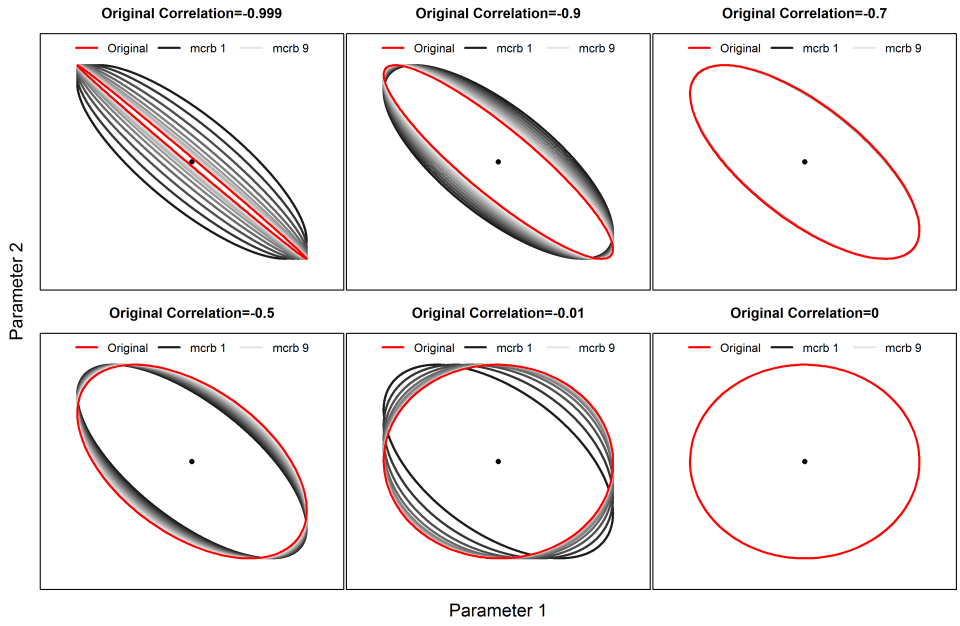

Figure 1: The effect of `mcrb` on a variety of correlations between two hypothetical parameters. Note that the effect of setting $N = 9$ depends on the original correlation.

3 SIR

3.0.3 Algorithm

3.0.4 Example

Use same model as the other example and show how to do SIR in R, then compare the two posteriors to show the same.

4 Running analyses in R

How to create top-level functions for running analyses programmatically.

4.1 Code

Here is one I wrote for my thesis, which could be generalized easily. Use psv file here, or force user to write to csv? Let them use append or always overwrite files? Who actually restarts?

```
run.mcmc <- function(Nout, mcsave, burn.in, boot, cor.mat = NULL, init.pin = NULL,
    se.scale = NULL, mcscale = FALSE, mcdiag = FALSE, mcseed = NULL, mcrb = NULL,
    mcgrope = NULL, verbose = TRUE, extra.args = NULL) {
    ## This function runs an ADMB model MCMC, burns and thins, calculates
    ## effective sizes, and returns stuff depending on verbose.
    require(coda)
    model <- "logistic"
    thin.every <- 1  # setting to 1 for this project
    iterations <- (Nout + burn.in) * mcsave
    ## print(paste('Run started at', round(Sys.time())))
    if (iterations < 1)
        stop(paste0("Iterations too low: ", iterations))
```

```
  wd.old <- getwd()
  on.exit(setwd(wd.old))
  setwd(model.dir)
  ## Run to get MLE and covariance matrix if none provided
  if (is.null(init.pin))
      system(model, ignore.stdout = T) else {
      write.table(file = "init.pin", x = init.pin, row.names = F, col.names = F)
  }

  if (!is.null(se.scale))
      SetScale(se.scale)  # change the covariance matrix
  ## Clean up the old files
  file.list <- c("mc_abundances.csv", "mc_abundances_def.csv", "mc_abundances_mit.csv",
      "mc_abundances_none.csv", "mc_shipstrikes.csv", "mc_shipstrikes_def.csv",
      "mc_shipstrikes_mit.csv", "mc_shipstrikes_none.csv", "mc_parameters.csv",
      "mc_vessels.csv", "mc_equil.csv")
  suppressWarnings(file.remove(file.list))
  cmd <- paste(model, "-mcmc", iterations, "-nohess -noest")
  ## make the argument and run the MCMC
  if (!is.null(mcseed))
      cmd <- paste(cmd, "-mcseed", mcseed)
  cmd <- paste(cmd, "-mcsave", mcsave)
  if (mcscale == FALSE)
      cmd <- paste(cmd, "-mcnoscale")
  if (!is.null(mcrb))
      cmd <- paste(cmd, "-mcrb", mcrb)
  if (!is.null(mcgrope))
      cmd <- paste(cmd, "-mcgrope", mcgrope)
  if (mcdiag == TRUE)
      cmd <- paste(cmd, "-mcdiag")
  if (!is.null(extra.args))
      cmd <- paste(cmd, extra.args)
  if (!is.null(init.pin))
      cmd <- paste(cmd, "-mcpin init.pin")
  ## print(cmd) If user provided covar matrix, write it to file
  if (!is.null(cor.mat))
      writeADMBCovariance(cor.mat)
  system(cmd, ignore.stdout = T)
  system(paste(model, "-mceval -noest -nohess"), ignore.stdout = T)

  ## Read in the results
  df <- read.csv("mc_parameters.csv")
  vessels <- read.csv("mc_vessels.csv", head = T)
  S <- read.csv("mc_shipstrikes.csv")
  S.none <- read.csv("mc_shipstrikes_none.csv")
  S.mit <- read.csv("mc_shipstrikes_mit.csv")
  S.def <- read.csv("mc_shipstrikes_def.csv")
  N <- read.csv("mc_abundances.csv")
  N.none <- read.csv("mc_abundances_none.csv")
  N.mit <- read.csv("mc_abundances_mit.csv")
  N.def <- read.csv("mc_abundances_def.csv")
  D <- N/df$K
  D.none <- N.none/df$K
```

```
    D.mit <- N.mit/df$K
    D.def <- N.def/df$K
    equil <- read.csv("mc_equil.csv")
    ## Rows are mcmc iterations, cols are years except df which is parameters
    mcmc <- list(params = df, vessels = vessels, S = S, D = D, N = N, S.none = S.none,
        S.mit = S.mit, S.def = S.def, N.none = N.none, N.mit = N.mit, N.def = N.def,
        D.none = D.none, D.mit = D.mit, D.def = D.def, equil = equil)
    if (nrow(df) > burn.in)
        mcmc <- lapply(mcmc, function(x) x[-(1:burn.in), ]) else stop("Too few samples to thin/burn")
    ## Run effective sample size calcs from CODA, metric of convergence
    efsize <- data.frame(t(effectiveSize(df)/nrow(df)))
    names(efsize) <- paste0(names(df), "_efs")
    ## Grab admb fit and metrics
    fit <- read.admbFit("logistic")
    mle <- with(fit, data.frame(NLL = nloglike, maxgrad = maxgrad, rMSY.cor = cor[2,
        1], boot = boot))
    temp <- data.frame(t(fit$est), row.names = F)
    names(temp) <- paste0(fit$names, "_mle")
    mle <- cbind(mle, temp[, -ncol(temp)])
    temp <- data.frame(t(fit$std), row.names = F)
    names(temp) <- paste0(fit$names, "_se")
    mle <- cbind(mle, temp[, -ncol(temp)])
    fit.df <- cbind(mle, efsize)
    if (verbose)
        return(list(fit.df = fit.df, mcmc = mcmc)) else return(fit.df = fit.df)
}
```