R package `mcll` for Monte Carlo local likelihood estimation

Minjeong Jeon
University of California, Berkeley

Cari Kaufman
University of California, Berkeley

Sophia Rabe-Hesketh
University of California, Berkeley

February 4, 2013

Abstract

`mcll` is an R package for Monte Carlo local likelihood (MCLL) estimation of generalized linear mixed models with crossed random effects. `mcll` implements the nested maximizations in the MCLL algorithm (Step 2), given posterior samples of model parameters for a particular set of priors (Step 1). `mcll` also provides standard error estimates for the MCLL parameter estimates. This paper describes how the MCLL algorithm works with the package `mcll`. The widely-used salamander mating data are used for illustration.

1 Monte Carlo Local Likelihood Method

Monte Carlo local likelihood (MCLL) is an approximate maximum likelihood method for estimating generalized linear mixed models (GLMM) with crossed random effects. MCLL initially treats model parameters as random variables and samples them from the posterior for a particular prior. The likelihood function is approximated up to a constant by fitting a density to the posterior samples and dividing it by the prior. The posterior density is approximated using local likelihood density estimation (Loader, 1996), where the log-likelihood is locally approximated by a polynomial function. For details on MCLL, see Jeon et al. (2012).

Here we describe the procedure of MCLL for parameter estimation. Specifically, assuming a d-dimensional parameter space $\boldsymbol{\theta}$ with observed data vector $\mathbf{y}$, the MCLL algorithm involves the following two steps:

Step 1 Choose a prior $p(\boldsymbol{\theta})$ and use a MCMC method to obtain samples from the posterior $p(\theta|\mathbf{y})$

$$p(\boldsymbol{\theta}|\mathbf{y}) = \frac{L(\mathbf{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{C_s},$$

where the normalizing constant is $C_s = \int L(\mathbf{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}$. is the likelihood and

Step 2 Maximize an approximation to the likelihood defined up to constant C_s by

$$\hat{L}(\mathbf{y}|\boldsymbol{\theta}) = \frac{1}{p(\boldsymbol{\theta})}\mathrm{Ps}_p(\boldsymbol{\theta}), (\theta),$$

where $\text{Ps}_p(\boldsymbol{\theta})$ is the local likelihood estimate of the posterior density. Specifically, for a given value of $\boldsymbol{\theta}$, this is obtained by assuming that the log-posterior density can be approximated by a polynomial function $P_{\boldsymbol{a}}(\mathbf{u}-\boldsymbol{\theta})$ p with parameters $\boldsymbol{a}$. For example, in the three dimensional case (d=3), the log-posterior can be locally approximated by a quadratic function

$$\begin{aligned} P_{\boldsymbol{a}}(\mathbf{u}-\boldsymbol{\theta}) = a_0 &+ a_1(u_1-\theta_1) + a_2(u_2-\theta_2) + a_3(u_3-\theta_3) \\ &+ \frac{1}{2}a_4(u_1-\theta_1)^2 + \frac{1}{2}a_5(u_2-\theta_2)^2 + \frac{1}{2}a_6(u_3-\theta_3)^2 \\ &+ a_7(u_1-\theta_1)(u_2-\theta_2) + a_8(u_1-\theta_1)(u_3-\theta_3) \\ &+ a_9(u_2-\theta_2)(u_3-\theta_3), \end{aligned}$$

where $\boldsymbol{a} = (a_0, a_1, ..., a_9)'$.

The $\boldsymbol{a}$ parameters are estimated for a particular $\boldsymbol{\theta}$ by maximizing a localized version of the log-likelihood, which in this case is

$$\hat{l}(\boldsymbol{\theta}, \boldsymbol{a}) = \sum_{j=1}^{m} K\left(\frac{\boldsymbol{\theta}^{(j)}-\boldsymbol{\theta}}{\mathbf{h}}\right) P_{\boldsymbol{a}}(\boldsymbol{\theta}^{(j)}-\boldsymbol{\theta}) - m\int K\left(\frac{\mathbf{u}-\boldsymbol{\theta}}{\mathbf{h}}\right)\exp\left(P_{\boldsymbol{a}}(\mathbf{u}-\boldsymbol{\theta})\right)d\mathbf{u}, \quad (1)$$

where $\{\boldsymbol{\theta}^{(j)}\}_{j=1}^{m}$ are the posterior sample points.

The MCLL method also provides a relatively simple way to compute standard errors. Specifically, we derive an alternative way of computing the Hessian matrix for MCLL by using the quadratic approximation of the log-posterior obtained using local likelihood density estimation, assuming the log-posterior can be well approximated by a quadratic polynomial in the neighborhood of the mode. For more details, see Section 3 in Jeon et al. (2012).

The package `mcll` consists of two main functions, `mcll_est` and `mcll_se`. `mcll_est` implements Step 2 in the MCLL procedure above, given the posterior samples obtained for a particular prior (in Step 1). It requires the values and posterior samples of model parameters on the real line. For example, log transformation of variance parameters is needed. `mcll_se` computes standard errors for the MCLL parameter estimates. In the next section, we illustrate how to implement the MCLL method (including Step 1) using a real data example.

2 Illustration

We use an example of a crossed random effects model using the salamander mating data (McCullagh, 1989, Section 14.5). This dataset is a benchmark that has been used to compare many different estimation methods for GLMMs with crossed random effects.

The salamander mating data consist of three separate experiments, each involving matings among salamanders of two different populations, called Rough Butt (RB) and White Side (WS). Sixty females and sixty males of two populations of salamander were paired by a crossed, blocked, and incomplete design in an experiment studying whether the two populations have developed generic mechanisms which would prevent inter-breeding. The response is a binary variable indicating whether mating was successful between female i and male j. We adopted model A used by Karim and Zeger (1992).

$$\text{logit}(p(y_{ij}=1|z_i^f, z_j^m)) = \beta_1 + \beta_2 x_{1i} + \beta_3 x_{2j} + \beta_4 x_{1i}x_{2j} + z_i^f + z_j^m, \quad (2)$$

where the covariates are dummy variables for White Side female (x_i), White Side male (x_j), and the interaction ($x_{1i}x_{2j}$). The two crossed random effects are random intercepts $z_i^f \sim N(0, \sigma_f^2)$ for females and $z_j^m \sim N(0, \sigma_m^2)$ for males. Each salamander participates in six matings, resulting in 360 matings in total.

Note that the two variance components in model (2) are reparameterized as $\tau_f = \log\sigma_f$ and $\tau_m = \log\sigma_m$.

2.1 Step 1: Obtain Posterior Samples

To obtain posterior samples for parameters, priors should be specified for model parameters. For model (2), we choose diffuse normal priors for the fixed effect parameters (with mean 0, standard deviation 100) and for the log-parameterized standard deviation parameters (with mean -0.98 and standard deviation 0.76). For details on this choice of priors, see Section 5.1 in Jeon et al. (2012).

Given the specified priors, the posterior samples can be obtained by any Markov chain Monte Carlo (MCMC) method. For example, the Bayesian software `WinBUGS` (Lunn et al., 2000) can be used together with the R package `R2WinBUGS` (Sturtz et al., 2005). In this example, we use three chains with relatively diffuse starting values. Each chain was run for 1,000 iterations after a 2,000 iteration burn-in period. For convergence assessment, the Gelman-Rubin statistic (Gelman and Rubin, 1992) is used in addition to graphical checks such as trace plots and autocorrelation plots. Here is an example code of using `R2WinBUGS` to run `WinBUGS`.

```
library(R2WinBUGS)
# dataset
data <- data(salamander)

# set up
n <- nrow(data) # all salamander
m <- length(unique(data$male)) # number of male
f <- length(unique(data$female))  # number of female
r <- data$y  # response (mating)
male <- data$male  # male id
female <- data$female # female id
wsm <- data$wsm  # WSM
wsf <- data$wsf  # WSF
ww <- data$ww  # WW

# mean and sd for the log variable
lmu <- -0.9870405  # mean
lsig <- 1/0.766672   # 1/sd (precision =1/sd^2)

# data set
sala.data <- list("n","m","f","r","male","female","wsm","wsf","ww","lmu","lsig")

# initial value set
sala.inits <- function() {
```

```
 list(rm= rnorm(m), rf=rnorm(f), b0 = rnorm(1), b1 = rnorm(1),
      b2 = rnorm(1), b3 = rnorm(1),
      tau0 =rnorm(1,lmu,sqrt(1/lsig)), tau1 =rnorm(1,lmu,sqrt(1/lsig)))

# parameter set
sala.parameters <- c("b0", "b1", "b2", "b3", "tau0", "tau1","sigma0","sigma1" )

# run WinBUGS (sala.bug) with 3 chains
post <- bugs(sala.data, sala.inits, sala.parameters, "sala.bug",
n.chains=3, n.iter=3000, n.burnin=2000,   # 1000 iterations after 2000 burn-in
n.thin=1, debug=F, bugs.directory="C:/Program Files/WinBUGS14/" )

# posterior samples
samp <- post$sims.matrix  # size 3000 x 6
```

Note that in WinBUGS, `lsig` is the inverse of the standard deviation of the log variable. The WinBUGS code "sala.bug" is is available in Appendix. The results of the running code above are stored in the package as object `samp`. One who do not want to run the code can use `samp`. Note that any other methods or software can be used to obtain the posterior samples.

2.2 Step 2: Obtain Parameter and Standard Error Estimates

Once the posterior samples for model parameters are obtained, `mcll_est` can be used to obtain parameter estimates. `mcll_est` uses a quadratic function (polynomial degree 2) and a tricube function for the weight function. A bandwidth is chosen at each data point so that the local neighborhood contains a specified number of points. Specifically, `mcll_est` requires a smoothing parameter α between 0 and 1, which is the nearest neighbor bandwidth with the kth smallest distance d where $k = \lfloor n\alpha \rfloor$ and $d(x, x_i) = |x - x_i|$ with the sample size n. Finally, a product kernel is used in (1) given the posterior samples (obtained in Step 1).

`mcll_est` requires a prior function which returns the log prior densities for parameter values. Specifically, the prior function should have as an argument a vector of parameter values ($vec.t$) and return value of the log prior density for those parameter values $vec.t$. For example, for the normal priors of the six parameters (β_1 to β_4 and τ_f and τ_m) in model (2), the prior function can be specified as

```
prior.func <- function(vec.t) {
    sum(dnorm(vec.t, m= c(0,0,0,0, -0.9870405, -0.9870405) ,
                sd=c(100,100,100,100, 1/0.766672, 1/0.766672) , log=T))
}
```

Here is an example of using `mcll_est` to estimate parameters for model (2).

```
library(mcll)
# posterior samples
data(samp)
# prior function
```

```
prior.func <- function(vec.t) {
    sum(dnorm(vec.t, m= c(0,0,0,0, -0.9870405, -0.9870405) ,
                sd=c(100,100,100,100, 1/0.766672, 1/0.766672) , log=T))
}
## parameter estimation
run1 <- system.time(
    result1 <- mcll_est(data=samp, prior.func= prior.func, alp=0.7,
        method = "BFGS", control= list(maxit=10000), use.locfit=TRUE )
)
```

If `use.locfit=TRUE`, the package `locfit` (Loader, 2012) is used to compute a local likelihood density estimate. `locfit` tends to be faster but can fail for high-dimensional problems. In these cases, a version of the local likelihood code is implemented in the package (use `use.locfit=FALSE`) and optimization methods can be chosen for finding the polynomial coefficients.

`mcll_est` returns the parameter estimates in the original scale as well as the usual output from `optim`.

```
result1
$par
            b0        b1         b2       b3       tau0     tau1
[1,] 0.9275766 -2.871686 -0.6488625 3.589313 0.08118962 0.148478

$convergence
[1] 0

$value
[1] -26.38284

$counts
function gradient
     112       15

$message
NULL
```

`value` is the unnormalized log-likelihood returned from the MCLL algorithm. It can be used to compute the Bayes factor. For more information on this, see Section 4 in Jeon et al. (2012). This parameter estimation took about 9 seconds on a Intel Pentium Dual-Core 2.5-GHz processor computer with 3.2 GB of memory.

For standard error estimation, the function `mcll_se` is used. `mcll_se` requires the Hessian matrix of the log prior **H.prior** evaluated at the MCLL parameter estimates. One can solve it analytically if a closed-form solution is available. For example, for the multivariate normal priors for the six parameters with zero mean and variance *p.var*, the Hessian matrix can be obtained as

```
p.var = c(100,100,100,100, 1/0.766672, 1/0.766672)^2
H.prior <- -diag(1/p.var)
```

Alternatively, one can use a numerical solution for the Hessian matrix using e.g., the `hessian` function in the `numDeriv` R package.

```
library(numDeriv)
log.prior.h <- hessian(prior.func, par)
```

Here is an example of using `mcll_se` to compute standard errors for the parameter estimates for model (2).

```
run2 <- system.time(
    result2 <- mcll_se(data=samp, par=par, H.prior = H.prior,  alp=0.7,
        method= "Nelder-Mead" , control=list(maxit=20000) )
)
result2
       b0        b1        b2        b3      tau0      tau1
0.4057844 0.5640063 0.4907643 0.6663096 0.3022842 0.2999727
```

`mcll_se` returns a vector of standard errors for the estimates for the model parameters (β_1 to β_4, τ_f and τ_m). Standard error estimation took about 122 seconds on a Intel Pentium Dual-Core 2.5-GHz processor computer with 3.2 GB of memory.

3 Discussion

There are several things to be discussed regarding implementation of MCLL. First, we uses an orthogonal transformation of the posterior samples. This orthogonal transformation, also called data presphering (Wand and Jones, 1993) is useful in implementing MCLL because it simplifies the integral term in (1). Specifically, for multidimensional parameter $\boldsymbol{\theta}$, if the components are approximately independent in the posterior, then interactions terms in $P_{\boldsymbol{a}}(\mathbf{u}-\boldsymbol{\theta})$ can be dropped. In addition, a product kernel can be used, with

$$K\left(\frac{\mathbf{u}-\boldsymbol{\theta}}{\mathbf{h}}\right) = \prod_{i=1}^{d} K_0\left(\frac{u_i-\theta_i}{h_i}\right), \tag{3}$$

where K_0 is a one-dimensional kernel. With these two simplifications, the multidimensional integral can be factorized as a product of one-dimensional integrals due to the orthogonality of the parameter space. In addition, the orthogonal transformation standardizes a bandwidth choice by transforming the parameter space to be on the same scale. That is, the default choice for $\alpha = 0.7$ works in most applications.

Second, we use a log-transformation of variance parameters. This has several advantages: 1) it avoids need for a modified kernel in Step 2 to handle truncation of the density at zero, 2) the posterior distributions are closer to normal so that data presphering operation works better for a symmetric distribution, and 3)the log-posterior is better approximated by a quadratic function.

Third, for model (2), we used diffuse priors for the fixed and log standard deviation parameters. In this case, we have shown that the posterior mean estimates as well as MCLL estimates are also close to ML estimates (Jeon et al., 2012, Section 5). Note that even if priors are poorly specified,

the MCLL algorithm provides results close to the ML estimates while the posterior mean estimates are not. For details, see Section 6.2 in Jeon et al. (2012).

Finally, it is important to note that MCLL allows likelihood inference for any complex models for which ML estimation may be infeasible but MCMC methods are possible. For example, in addition to GLMMs with crossed random effects considered here, the MCLL algorithm could be used to fit models with higher dimensional latent variables such as spatial models for disease mapping. Therefore, when ML inference is desirable for highly complex models, the MCLL method is an effective and practical choice.

Appendix

Here is the WinBUGS code for model (2) for the salamander mating data. To use this for `bugs`, save this as "sala.bug" as shown in Section 2.1.

```
## crossed random effects model for salamander data

# n: number of salamanders
# m: number of female
# f: number of male
# rm: random effects for male
# rf: random effects for female
# mu0: mean for male
# mu1: mean for female
# zeta0: inverse variance for male
# zeta1: inverse variance for female

model
{

for (i in 1:n) {
        logit(p[i]) <- b0 + b1*wsf[i] + b2*wsm[i] + b3*ww[i]
        + rm[male[i]] + rf[female[i]]
        r[i] ~ dbern(p[i])
}

b0 ~ dnorm(0, .0001)
b1 ~ dnorm(0, .0001)
b2 ~ dnorm(0, .0001)
b3 ~ dnorm(0, .0001)

for (j in 1:m) {
    rm[j] ~ dnorm(0,zeta0)
}

zeta0 <- pow(exp(tau0),-2)
```

```
tau0 ~ dnorm(lmu,lsig) # normal
sigma0 <- exp(tau0)

for (k in 1:f) {
    rf[k] ~ dnorm(0,zeta1)
}

zeta1 <- pow(exp(tau1),-2)
tau1 ~ dnorm(lmu,lsig)
sigma1 <- exp(tau1)

}
```

References

[1] Gelman, A. and Rubin, D. (1992). Inference from iterative simulation using multiple sequences. Statistical Science 7, 457-472.

[2] Jeon, M., Kaufman, C., and Rabe-Hesketh, S. (2012). Monte Carlo local likelihood for estimating generalized linear mixed models. Submitted for publication.

[3] Karim, M. and Zeger, S. (1992). Generalized linear models with random effects: Salamander mating revisited. Biometrics 48, 631-644.

[4] Loader, C. (1996). Local likelihood density estimation. The Annals of Statistics 24, 1602-1618.

[5] Loader, C. (2012). locfit: local regression, likelihood, and density estimation. Downloadable from http://cran.r-project.org/web/packages/locfit/index.html.

[6] unn, D., Thomas, A., Best, N., and Spiegelhalter, D. (2000). WinBUGS - a Bayesian modelling framework: concepts, structure, and extensibility. Statistics and Computing 10, 325-337.

[7] McCullagh, P. and Nelder, J. (1989). Generalized Linear Models. Chapman and Hall, New York.

[8] Sturtz, S., Ligges, U., and Gelman, A. (2005). R2WinBUGS: A package for running WinBUGS from R. Journal of Statistical Software 12, 1-16.

[9] Wand, M. P. and Jones, M. C. (1993). Comparison of smoothing parameterizations in bivariate kernel density estimation. Journal of the American Statistical Association 88, 520-528.