Applied Bayesian Inference in R Using MCMCpack

Andrew D. Martin
Washington University in St. Louis

Kevin Quinn
Harvard University

http://mcmcpack.wustl.edu

June 16, 2006

MCMCpack Goals

- Free, open-source, easy-to-use software for Bayesian inference.
- Provide a development environment for easy implementation of non-standard statistical models.
- Provide a distribution mechanism for other researchers with a consistent user interface and documentation.

MCMCpack Design

- "R-like" user interface for Bayesian tools.
- Model-specific design.
- Most computation done in C++ (using the Scythe Statistical Library).
- coda `mcmc` objects for posterior sample storage and summarization.
- Modular design and hidden functions to ease implementing additional models.

Why Not WinBUGS, JAGS, OpenBugs, etc?

- The BUGS language is good for many things, including quickly developing models. We see it as a complementary tool to `MCMCpack`.

However . . .

However . . .

- . . . as noted in the WinBUGS manual:

> [P]otential users are reminded to be extremely careful if using this program for *serious statistical analysis*. . . If there is a problem, WinBUGS might just crash, which is not very good, but it might well carry on and produce answers that are wrong, which is even worse (p. 1).

However . . .

- . . . as noted in the WinBUGS manual:

  > [P]otential users are reminded to be extremely careful if using this program for *serious statistical analysis*. . . If there is a problem, WinBUGS might just crash, which is not very good, but it might well carry on and produce answers that are wrong, which is even worse (p. 1).

- The BUGS language can be slow (especially for large problems), and the WinBUGS engine does not work for certain problems.

However . . .

- . . . as noted in the WinBUGS manual:

  > [P]otential users are reminded to be extremely careful if using this program for *serious statistical analysis*. . . If there is a problem, WinBUGS might just crash, which is not very good, but it might well carry on and produce answers that are wrong, which is even worse (p. 1).

- The BUGS language can be slow (especially for large problems), and the WinBUGS engine does not work for certain problems.

- The various flavors of BUGS require the user to do some modest programming which increases the opportunities for something to go wrong.

Example 1: Inference for an Item Response Theory (IRT) Model

Consider:

$$y_{ij} \sim \mathcal{B}ernoulli(\pi_{ij}), \quad i = 1, \ldots, I \quad j = 1, \ldots, J$$

with

$$\pi_{ij} = \Phi(-\alpha_j + \beta_j \theta_i)$$

Typically, i indexes test-takers (subjects) and j indexes test items.

Prior distributions for the model parameters are:

$$(\alpha_j, \beta_j)' \sim \mathcal{N}(\mathbf{a}_0, \mathbf{A}_0^{-1}) \quad j = 1, \ldots, J$$

and

$$\theta_i \sim \mathcal{N}(0, 1) \quad i = 1, \ldots, I$$

To fit this model to some test data from
`http://work.psych.uiuc.edu/irt/downloads.asp`
using `MCMCpack` one can do the following.

```
post.irt <- MCMCirt1d(testdata,
                      burnin=5000, mcmc=100000, thin=10,
                      AB0=.001, store.item=TRUE, store.ability=TRUE,
                      verbose=5000,
                      theta.constraints=list(SUBJECT368="+",
                                             SUBJECT356="-"))
```

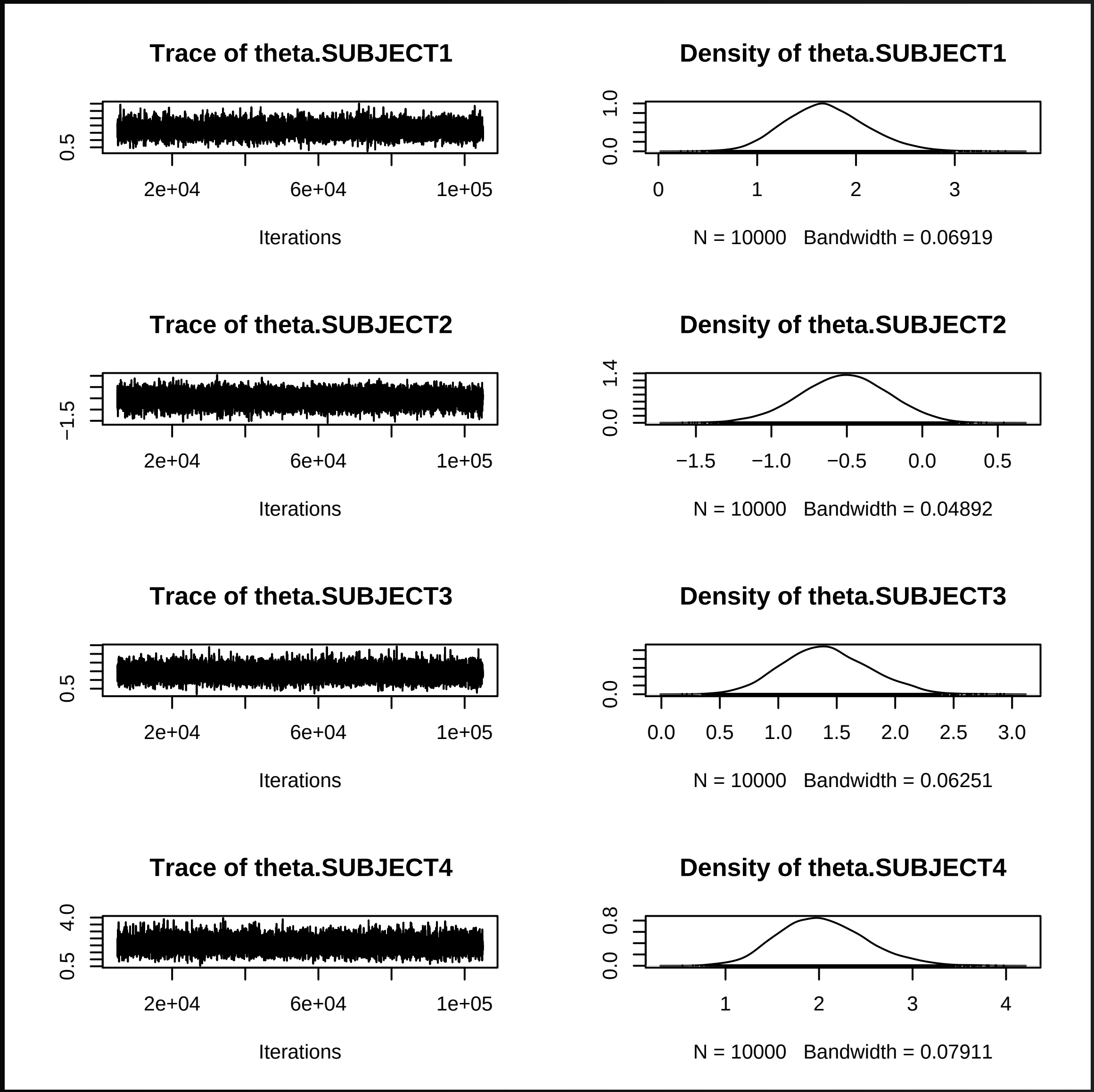
To fit this model to some test data from
`http://work.psych.uiuc.edu/irt/downloads.asp`
using `MCMCpack` one can do the following.

```
post.irt <- MCMCirt1d(testdata,
                      burnin=5000, mcmc=100000, thin=10,
                      AB0=.001, store.item=TRUE, store.ability=TRUE,
                      verbose=5000,
                      theta.constraints=list(SUBJECT368="+",
                                             SUBJECT356="-"))
```

To look at some traceplots and marginal density estimates one can use:

```
plot(post.irt)
```

which produces:



Trace of theta.SUBJECT1
0.5
2e+04
6e+04
1e+05
Iterations
Density of theta.SUBJECT1
0.0
1.0
0
1
2
3
N = 10000 Bandwidth = 0.06919
Trace of theta.SUBJECT2
−1.5
2e+04
6e+04
1e+05
Iterations
Density of theta.SUBJECT2
0.0
1.4
−1.5
−1.0
−0.5
0.0
0.5
N = 10000 Bandwidth = 0.04892
Trace of theta.SUBJECT3
0.5
2e+04
6e+04
1e+05
Iterations
Density of theta.SUBJECT3
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
N = 10000 Bandwidth = 0.06251
Trace of theta.SUBJECT4
0.5
4.0
2e+04
6e+04
1e+05
Iterations
Density of theta.SUBJECT4
0.0
0.8
1
2
3
4
N = 10000 Bandwidth = 0.07911

Subject 436 and subject 447 both got 75% of the test items correct. Which subject has higher ability?

The posterior probability that subject 436 has higher ability than subject 447 can be calculated with:

```
mean(post.irt[,436] > post.irt[,447])
```

which evaluates to 0.65.

We might also be interested in which test items do a good job of discriminating between high and low ability subjects. The β parameters provide this information.

We can calculate the posterior expection of each of the discrimination parameters with:

```
beta.post <- post.irt[,seq(from=452, to=530, by=2)]
print(sort(colMeans(beta.post)))
```

which produces:

```
 beta.ITEM18  beta.ITEM28  beta.ITEM16  beta.ITEM37  beta.ITEM19  beta.ITEM32
-0.033332789  0.007374421  0.084904972  0.217837674  0.276428999  0.315174596
  beta.ITEM8  beta.ITEM20  beta.ITEM40  beta.ITEM21   beta.ITEM1  beta.ITEM34
 0.331350234  0.411436261  0.418098884  0.419030043  0.439532906  0.460505549
  beta.ITEM7  beta.ITEM10  beta.ITEM22  beta.ITEM14  beta.ITEM35  beta.ITEM31
 0.462356437  0.472680964  0.481691248  0.482167728  0.514685111  0.564987099
 beta.ITEM39  beta.ITEM36  beta.ITEM33   beta.ITEM3   beta.ITEM4  beta.ITEM27
 0.601600337  0.623693132  0.627042964  0.634003442  0.641052025  0.672391363
 beta.ITEM29   beta.ITEM2   beta.ITEM6  beta.ITEM11  beta.ITEM30  beta.ITEM13
 0.689009747  0.706951188  0.723142300  0.762149494  0.772004859  0.779142756
  beta.ITEM9  beta.ITEM38   beta.ITEM5  beta.ITEM23  beta.ITEM25  beta.ITEM17
 0.851232006  0.901117855  0.902609937  0.969587117  1.039535127  1.041296658
 beta.ITEM12  beta.ITEM26  beta.ITEM24  beta.ITEM15
 1.099500195  1.217643770  1.282921879  1.377932523
```

We can also calculate the posterior expected ranks of the discrimination parameters:

```
beta.post.ranks <- matrix(NA, nrow(beta.post), ncol(beta.post))
colnames(beta.post.ranks) <- colnames(beta.post)
for (i in 1:nrow(beta.post)){
  beta.post.ranks[i,] <- rank(beta.post[i,])
}
print(sort(colMeans(beta.post.ranks)))
```

```
beta.ITEM18 beta.ITEM28 beta.ITEM16 beta.ITEM37 beta.ITEM19 beta.ITEM32
     1.4852      1.9182      2.8395      4.7514      5.8596      7.0272
 beta.ITEM8 beta.ITEM20 beta.ITEM21 beta.ITEM40  beta.ITEM1 beta.ITEM34
     7.5302     10.7339     11.1222     11.1408     12.1650     13.1280
 beta.ITEM7 beta.ITEM10 beta.ITEM22 beta.ITEM14 beta.ITEM35 beta.ITEM31
    13.2174     13.6764     14.0879     14.1981     15.8325     18.3422
beta.ITEM39 beta.ITEM36 beta.ITEM33  beta.ITEM3  beta.ITEM4 beta.ITEM27
    20.1938     21.2814     21.4698     21.7877     22.1645     23.6409
beta.ITEM29  beta.ITEM2  beta.ITEM6 beta.ITEM11 beta.ITEM30 beta.ITEM13
    24.4503     25.2602     25.9368     27.5372     27.9766     28.2097
 beta.ITEM9 beta.ITEM38  beta.ITEM5 beta.ITEM23 beta.ITEM25 beta.ITEM17
    30.6817     32.0941     32.1787     33.7927     35.0096     35.2362
beta.ITEM12 beta.ITEM26 beta.ITEM24 beta.ITEM15
    36.2589     37.8847     38.5471     39.3517
```

Example 2: Calculating Bayes Factors for Model Comparison

Suppose that the observed data $\mathbf{y}$ could have been generated under one of two models $\mathcal{M}_1$ and $\mathcal{M}_2$.

A natural thing to ask from the Bayesian perspective is, "what is the posterior probability that $\mathcal{M}_1$ is true (assuming either $\mathcal{M}_1$ or $\mathcal{M}_2$ is true)?"

How can we calculate this? Well, using Bayes theorem we can write:

$$\Pr(\mathcal{M}_k|\mathbf{y}) = \frac{p(\mathbf{y}|\mathcal{M}_k)\Pr(\mathcal{M}_k)}{p(\mathbf{y}|\mathcal{M}_1)\Pr(\mathcal{M}_1) + p(\mathbf{y}|\mathcal{M}_2)\Pr(\mathcal{M}_2)}, \quad k = 1, 2$$

The key quantities here are $p(\mathbf{y}|\mathcal{M}_1)$ and $p(\mathbf{y}|\mathcal{M}_2)$ which are called *marginal likelihoods* or *integrated likelihoods*.

It is instructive to look at the posterior odds in favor of one model (say $\mathcal{M}_1$):

$$\frac{\Pr(\mathcal{M}_1|\mathbf{y})}{\Pr(\mathcal{M}_2|\mathbf{y})} = \frac{p(\mathbf{y}|\mathcal{M}_1)}{p(\mathbf{y}|\mathcal{M}_2)} \times \frac{\Pr(\mathcal{M}_1)}{\Pr(\mathcal{M}_2)}$$

What this means is that if we want to move from the prior odds in favor of $\mathcal{M}_1$ to the posterior odds in favor of $\mathcal{M}_1$ we simply multiply the prior odds by:

$$B_{12} = \frac{p(\mathbf{y}|\mathcal{M}_1)}{p(\mathbf{y}|\mathcal{M}_2)}$$

which is called the *Bayes factor* for $\mathcal{M}_1$ relative to $\mathcal{M}_2$.

In words,

$$\text{posterior odds} = \text{Bayes factor} \times \text{prior odds}$$

Since the posterior odds equal the Bayes factor when the models are equally likely *a priori*, the Bayes factor is a measure of how much support is available in the data for one model relative to another.

It is now possible to calculate Bayes factors and posterior probabilities of models with MCMCpack.

```
data(birthwt)
model1 <- MCMCregress(bwt~age+lwt+as.factor(race) + smoke + ht,
                       data=birthwt,
                       b0=c(2700, 0, 0, -500, -500, -500, -500),
                       B0=c(1e-6, .01, .01, 1.6e-5, 1.6e-5, 1.6e-5,
                            1.6e-5), c0=10, d0=4500000,
                       marginal.likelihood="Chib95", mcmc=50000)

model2 <- MCMCregress(bwt~age+lwt+as.factor(race) + smoke,
                       data=birthwt,
                       b0=c(2700, 0, 0, -500, -500, -500),
                       B0=c(1e-6, .01, .01, 1.6e-5, 1.6e-5, 1.6e-5),
                       c0=10, d0=4500000,
                       marginal.likelihood="Chib95", mcmc=50000)
```

```
model3 <- MCMCregress(bwt~as.factor(race) + smoke + ht,
                       data=birthwt,
                       b0=c(2700, -500, -500, -500, -500),
                       B0=c(1e-6, 1.6e-5, 1.6e-5, 1.6e-5,
                         1.6e-5), c0=10, d0=4500000,
                       marginal.likelihood="Chib95", mcmc=50000)

BF <- BayesFactor(model1, model2, model3)

mod.probs <- PostProbMod(BF)
```

```
print(BF)

The matrix of Bayes Factors is:
       model1 model2 model3
model1  1.000  14.08  7.289
model2  0.071   1.00  0.518
model3  0.137   1.93  1.000

The matrix of the natural log Bayes Factors is:
       model1 model2 model3
model1   0.00  2.645  1.986
model2  -2.64  0.000 -0.658
model3  -1.99  0.658  0.000
```

```
print(mod.probs)

    model1     model2     model3
0.82766865 0.05878317 0.11354819
```

Example 3: Single Block Metropolis Sampling for a User-Defined Model

`MCMCpack` also has some facilities for fitting user-specified models.

The `MCMCmetrop1R()` function uses a random walk Metropolis algorithm to sample from a user-defined log-posterior density.

Suppose one is interested in fitting a Bayesian negative binomial regression to the Ornstein data in the `car` package.

One could do the following:

```
negbinlogpost<- function(theta, y, X, b0, B0, c0, d0){

   ## log of inverse gamma density
   logdinvgamma <- function(sigma2, a, b){
     logf <- a * log(b) - lgamma(a) + -(a+1) * log(sigma2) + -b/sigma2
     return(logf)
   }

   ## log of multivariate normal density
   logdmvnorm <- function(theta, mu, Sigma){
     d <- length(theta)
     logf <- -0.5*d * log(2*pi) - 0.5*log(det(Sigma)) -
       0.5 * t(theta - mu) %*% solve(Sigma) %*% (theta - mu)
     return(logf)
   }

   k <- length(theta)
   beta <- theta[1:(k-1)]
   alpha <- exp(theta[k])
   mu <- exp(X %*% beta)

   ## evaluate log-likelihood at (alpha, beta)
   log.like <- sum(
                   lgamma(y+alpha) - lfactorial(y) - lgamma(alpha) +
```

```
                   alpha * log(alpha/(alpha+mu)) +
                   y * log(mu/(alpha+mu))
                   )

   ## evaluate log prior at (alpha, beta)
   ## note Jacobian term necessary b/c of transformation
   log.prior <-  logdinvgamma(alpha, c0, d0) + theta[k] +
     logdmvnorm(beta, b0, B0)

   return(log.like+log.prior)
}

library(car)
data(Ornstein)
yvec <- Ornstein$interlocks
Xmat <- model.matrix(~sector+nation, data=Ornstein)


post.negbin <- MCMCmetrop1R(negbinlogpost, theta.init=rep(0,14),
                             X=Xmat, y=yvec,
                             thin=2, mcmc=50000, burnin=1000,
                             tune=rep(.7,14),
                             verbose=500, logfun=TRUE, optim.maxit=100,
                             b0=0, B0=diag(13)*1000, c0=1, d0=1)
```

```
summary(post.negbin)

Iterations = 1001:50999
Thinning interval = 2
Number of chains = 1
Sample size per chain = 25000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

                     Mean     SD  Naive SE Time-series SE
beta.(Intercept)  2.34860 0.1602 0.0010132       0.003964
beta.sectorBNK    1.70677 0.4014 0.0025385       0.010942
beta.sectorCON   -0.59210 0.5526 0.0034949       0.015653
beta.sectorFIN    0.99221 0.2653 0.0016778       0.007198
beta.sectorHLD    0.30736 0.4251 0.0026884       0.011383
beta.sectorMAN    0.12265 0.2226 0.0014077       0.005949
beta.sectorMER    0.23269 0.2728 0.0017253       0.007095
beta.sectorMIN    0.66688 0.2121 0.0013413       0.006118
beta.sectorTRN    0.89300 0.2779 0.0017578       0.006900
beta.sectorWOD    0.67976 0.2752 0.0017403       0.008080
beta.nationOTH   -0.07207 0.2985 0.0018882       0.008870
beta.nationUK    -0.56004 0.2736 0.0017304       0.008131
beta.nationUS    -0.84186 0.1511 0.0009559       0.004240
```

```
log(alpha)        0.07750 0.1070 0.0006764       0.003525

2. Quantiles for each variable:

                    2.5%      25%      50%     75%     97.5%
beta.(Intercept)  2.0484  2.23548  2.34587  2.4571  2.668864
beta.sectorBNK    0.9810  1.42833  1.68293  1.9583  2.555186
beta.sectorCON   -1.6220 -0.96827 -0.59939 -0.2372  0.538723
beta.sectorFIN    0.4877  0.81116  0.98990  1.1638  1.539247
beta.sectorHLD   -0.4815  0.02123  0.29320  0.5806  1.182072
beta.sectorMAN   -0.3233 -0.02640  0.12499  0.2735  0.557485
beta.sectorMER   -0.2974  0.05259  0.22863  0.4068  0.776510
beta.sectorMIN    0.2526  0.52381  0.66726  0.8065  1.088377
beta.sectorTRN    0.3814  0.70355  0.88244  1.0776  1.448740
beta.sectorWOD    0.1624  0.49518  0.67317  0.8621  1.244756
beta.nationOTH   -0.6499 -0.27512 -0.07392  0.1272  0.512434
beta.nationUK    -1.0953 -0.74036 -0.56804 -0.3831 -0.005212
beta.nationUS    -1.1399 -0.94211 -0.84336 -0.7415 -0.539916
log(alpha)       -0.1357  0.00676  0.07577  0.1511  0.287035
```

Future Plans

- add more models (generalized linear mixed models, Dirichlet process mixture models, dynamic linear models, other models suggested by users)
- add more options for prior specifications
- add additional flexible sampling engines (more complicated Metropolis-Hastings methods, slice sampling, etc.)
- improve performance
- add vignettes
- add enhancements to `coda` `mcmc` objects to allow for sampling from posterior and prior predictive distributions

Acknowledgements

National Science Foundation
Program in Methodology, Measurement, and Statistics
(Grants SES-0350646 and SES-0350613)

Washington University, Department of Political Science, and the
Weidenbaum Center on the Economy, Government, and Public Policy
http://wc.wustl.edu

Harvard University, Department of Government, and the
Institute for Quantitative Social Science
http://www.iq.harvard.edu