rcppbugs – native MCMC for R

Whit Armstrong
armstrong.whit@gmail.com

KLS Diversified Asset Management

May 12, 2012

What is this rcppbugs thing?

Or as Paul Johnson put it recently on r-sig-mixed-models:

```
Hi, Whit:

I don't have the thread going back to the ancient times (:)), so I
can't tell what the point of this might be.  I wonder "what the
hell is he talking about?" and "why would I want a package called
rcppbugs if the project he describes doesn't have any C++ in it?"

I tried to install rcppbugs on Debian, but ended in failure,  so
obviously I'm missing some devel headers*.  While I track back to
find out WTF on that, how about you post an explanation of why
rcppbugs is worth the effort in the first place?
```

Short Answer

- rcppbugs is a new package that attempts to provide a pure R alternative to using OpenBUGS/WinBUGS/JAGS for MCMC
- it uses random walk Metropolis for sampling (Gibbs within Metropolis will be a future feature)
- The core of the package is c++, but the model specification is in R
- Did I mention it's pretty fast...?

Motivation

Why bother? There are lots of packages that already do MCMC...

- WinBugs is great.
- MCMC is easy.
- Besides if you need to go fast, there is MCMCpack[1]...

[1] just make sure you use a common model, nothing custom

Exhibit 1

I thought we were using R to avoid for loops...

```
model {
for (i in 1:n){
  y[i] ~ dnorm(y.hat[i], tau.y)
  y.hat[i] <- inprod(B[county[i],],X[i,])
  e.y[i] <- y[i] - y.hat[i]
}
tau.y <- pow(sigma.y, -2)
sigma.y ~ dunif(0, 100)
for (j in 1:J) {
    for(k in 1:K) {
      B[j,k] <- xi[k] * B.raw[j,k]
    }
    B.raw[j,1:K] ~ dmnorm(mu.raw[], Tau.B.raw[,])
}
for(k in 1:K) {
  mu[k] <- xi[k] * mu.raw[k]
  mu.raw[k] ~ dnorm(0,.0001)
  xi[k] ~ dunif(0,100)
}
Tau.B.raw[1:K,1:K] ~ dwish(W[,],df)
df <- K + 1
Sigma.B.raw[1:K,1:K] <- inverse(Tau.B.raw[,])
for(k in 1:K) {
  for(k.prime in 1:K) {
    rho.B[k,k.prime] <- Sigma.B.raw[k,k.prime] / sqrt(Sigma.B.raw[k,k] * Sigma.B.raw[k.prime,k.prime])
  }
  sigma.B[k] <- abs(xi[k]) * sqrt(Sigma.B.raw[k,k])
}
}
```

Motivation

- WinBugs **is** great... but slow
- MCMC is ~~easy~~hard, we need better tools to do it
- MCMC is already hard, and you have to set up another toolchain to do it, and then debug outside of R (WTF!)
- MCMCpack **is** great, but what if you need to fit a custom model...
- and like the Julia devs, I want it to go fast! (Doug Bates, come back!)

So, what can it do? (get on with it...)

Basic linear model

$$\beta \sim \mathcal{N}(0, 0.0001)$$
$$\tau_y \sim \Gamma(0.1, 0.1)$$
$$\hat{y} = \alpha + \beta \cdot X$$
$$y \sim \mathcal{N}(\hat{y}, \tau_y)$$

Basic linear model – data setup

X is 1000x2

```
require(rcppbugs)

NR <- 1e3L
NC <- 2L
intercept <- 10
y <- rnorm(NR,1) + intercept
X <- matrix(nr=NR,nc=NC)
X[,1] <- 1
X[,2] <- y - intercept + rnorm(NR)


lm.res <- lm.fit(X,y)
print(coef(lm.res))

##      x1      x2
## 10.5264  0.4866
```

Basic linear model – BUGS implementation

```
model {
 for (i in 1:N){
   y[i] ~ dnorm(y.hat[i], tau)
   y.hat[i] <- a + b * x[i]
 }
 a ~ dnorm(0, .0001)
 b ~ dnorm(0, .0001)
 tau ~ dgamma(0.1,0.1)
}
```

Basic linear model – rcppbugs implementation

$$\beta \sim \mathcal{N}(0, 0.0001)$$
$$\tau_y \sim \Gamma(0.1, 0.1)$$
$$\hat{y} = \alpha + \beta \cdot X$$
$$y \sim \mathcal{N}(\hat{y}, \tau_y)$$

```
b <- mcmc.normal(rnorm(NC),mu=0,tau=0.0001)
tau.y <- mcmc.gamma(runif(1),alpha=0.1,beta=0.1)
y.hat <- deterministic(function(X,b) { X %*% b }, X, b)
y.lik <- mcmc.normal(y,mu=y.hat,tau=tau.y,observed=TRUE)
m <- create.model(b, tau.y, y.hat, y.lik)
ans <- run.model(m, iterations=1e5L, burn=1e4L, adapt=1e3L, thin=10L)
```

```
## acceptance ratio:  0.4063
## credible interval for 'b':
##       [,1]   [,2]
## 10% 10.49 0.4660
## 50% 10.53 0.4863
## 90% 10.56 0.5071
```

Basic linear model – using 'linear' shortcut

```
b <- mcmc.normal(rnorm(NC),mu=0,tau=0.0001)
tau.y <- mcmc.gamma(runif(1),alpha=0.1,beta=0.1)
y.hat <- linear(X, b)
y.lik <- mcmc.normal(y,mu=y.hat,tau=tau.y,observed=TRUE)
m <- create.model(b, tau.y, y.hat, y.lik)
runtime <- system.time(ans <- run.model(m, iterations=1e5L, burn=1e4L,
adapt=1e3L, thin=10L))
```

```
## acceptance ratio:  0.3327
## runtime:  0.515
## credible interval for 'b':
##       [,1]   [,2]
## 10% 10.49 0.4663
## 25% 10.51 0.4763
## 50% 10.53 0.4867
## 75% 10.54 0.4975
## 90% 10.56 0.5070
```

A simple benchmark

```
iterations <- 1e5L
burn <- iterations
adapt <- 1e3L
thin <- 10L

lm.time <- system.time(lm.res <- lm.fit(X,y))
rcppbugs.time <- system.time(rcppbugs.ans <- run.model(m, iterations=iterations,
burn=burn, adapt=adapt, thin=thin))
rcppbugs.coefs <- apply(rcppbugs.ans[["b"]],2,mean)

mcmcpack.time <- system.time(mcmcpack.out <- MCMCregress(y ~
X.2,data=data.frame(y=y,X=X),burnin = burn, mcmc = iterations, thin=thin))
```

A simple benchmark – BUGS needs its own slide

```
linear.bug.fun <- function() {
    for (i in 1:NR){
        y[i] ~ dnorm(y.hat[i], tau.y)
        y.hat[i] <- inprod(b, X[i,])
    }
    for (j in 1:NC){
        b[j] ~ dnorm(0, .0001)
    }
    tau.y ~ dunif(0,100)
}

bug.file <- "linear.model.bug"
write.model(linear.bug.fun,con=bug.file)

jags.time1 <- system.time(
             jags <- jags.model(bug.file,
                                data=list(X=X,y=y,NR=NR,NC=NC),
                                n.chains = 1,
                                n.adapt = adapt,
                                quiet=TRUE
                                ))
jags.time2 <- system.time(update(jags, n.iter=burn, progress.bar="none"))
jags.time3 <- system.time(jags.trace <-
jags.samples(jags,c('b','tau.y'),n.iter=iterations,thin=thin,progress.bar="none"))

jags.time <- jags.time1 + jags.time2 + jags.time3
jags.coefs <- apply(jags.trace$b,1,mean)
```

benchmark results

```
print(round(all.times.ratio, 4))

##             time   ratio
## rcppbugs   0.889   1.000
## mcmcpack   2.386   2.684
## jags     165.435 186.091

print(coef.compare)

##             x1     x2
## lm.fit   10.53 0.4866
## rcppbugs 10.53 0.4870
## mcmcpack 10.53 0.4867
## jags     10.53 0.4864
```

Toolchain

Much of what I was able to do is because of the outstanding toolchain available in R and the fantastic software (outside of the R universe) that has already been written.

- Armadillo
- Rcpp/RcppArmadillo
- WinBUGS
- PyMC

Now a brief interlude.

Anyone know who this is?

David Spiegelhalter... WinBUGS

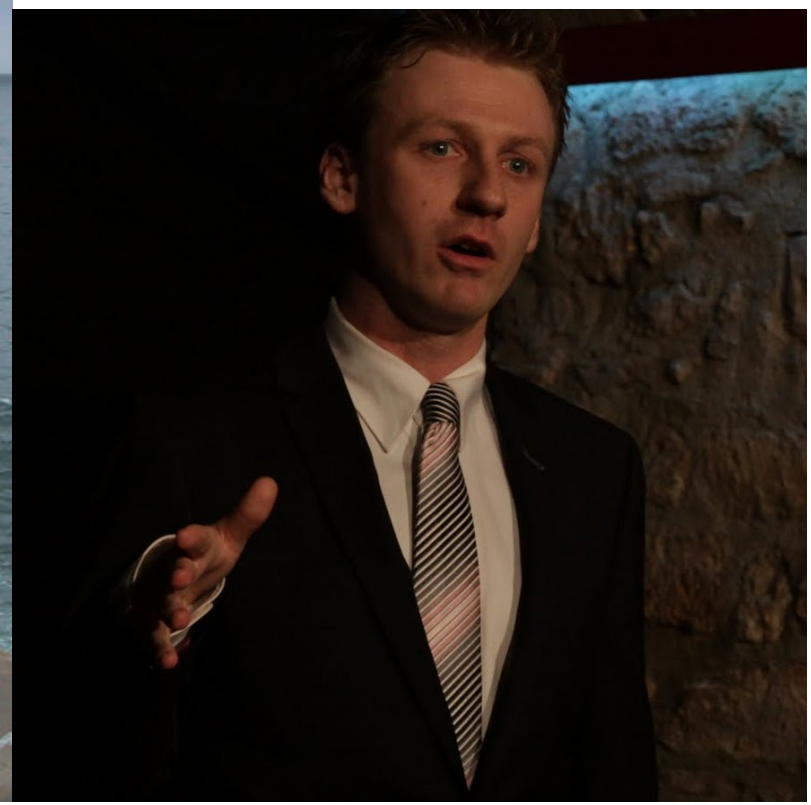
Anyone know who this is?

Chris Fonnesbeck... PyMC

Anyone know who this is?

Conrad Sanderson... Armadillo

How about these shady characters?



Dirk Eddelbuettel and Romain Francois... Rcpp/RcppArmadillo

Plumbing – What's going on?

```
b <- mcmc.normal(rnorm(NC),mu=0,tau=0.0001)
mcmc.normal

## function (x, mu, tau, observed = FALSE)
## {
##     if (missing(mu))
##         stop("required argument 'mu' missing.")
##     if (missing(tau))
##         stop("required argument 'tau' missing.")
##     if (length(mu) > length(x) || length(tau) > length(x)) {
##         stop("dimensions of hyperparmeters are larger than the stochastic variabl
##             "(is this really what you wanted to do?)")
##     }
##     check.dim.eq(x, mu)
##     check.dim.eq(x, tau)
##     attr(x, "distributed") <- "normal"
##     attr(x, "mu") <- substitute(mu)
##     attr(x, "tau") <- substitute(tau)
##     attr(x, "observed") <- observed
##     attr(x, "env") <- parent.frame()
##     class(x) <- "mcmc.object"
##     x
## }
## <environment: namespace:rcppbugs>
```

Multilevel Beta estimation

$$
\begin{aligned}
\mu_\alpha &\sim \mathcal{N}(0, 0.0001) \\
\tau_\alpha &\sim \Gamma(0.1, 0.1) \\
\alpha &\sim \mathcal{N}(\mu_\alpha, \tau_\alpha)
\end{aligned}
$$

$$
\begin{aligned}
\mu_\beta &\sim \mathcal{N}(0, 0.0001) \\
\tau_\beta &\sim \Gamma(0.1, 0.1) \\
\beta &\sim \mathcal{N}(\mu_\beta, \tau_\beta)
\end{aligned}
$$

$$
\begin{aligned}
\hat{y} &= \alpha + \beta \cdot X \\
\tau_y &\sim \Gamma(0.1, 0.1) \\
y &\sim \mathcal{N}(\hat{y}, \tau_y)
\end{aligned}
$$

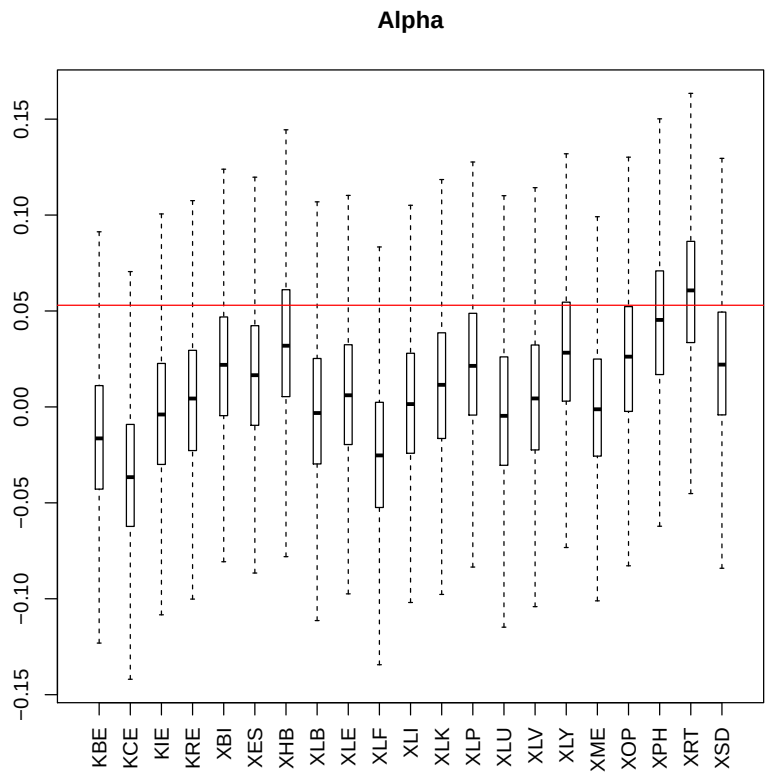
Multilevel Beta estimation – implementation

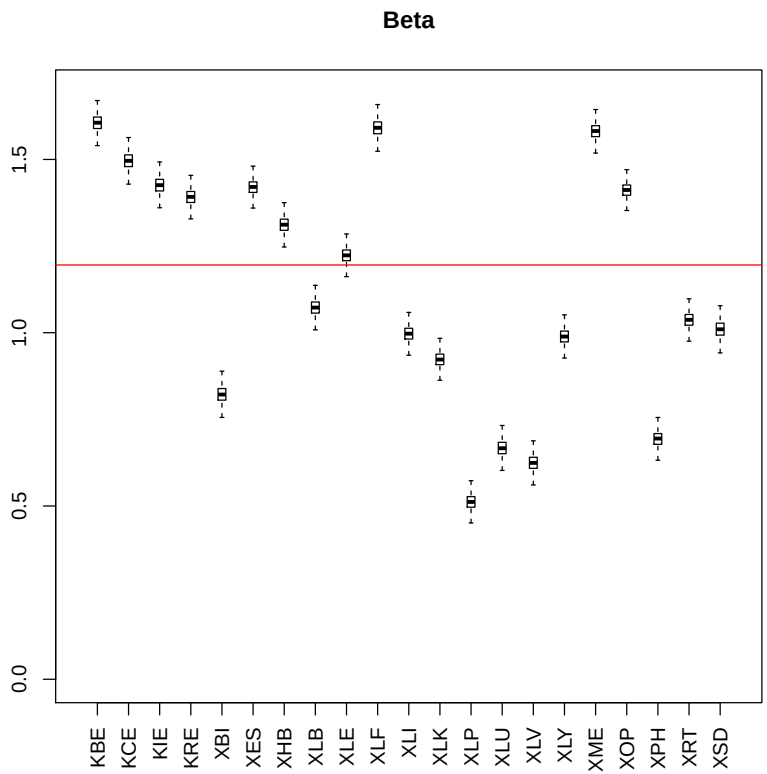
```
## hyperparms for a -- alpha
mu.a <- mcmc.normal(0,0,0.0001)
tau.a <- mcmc.uniform(runif(1),0,100)
a <- mcmc.normal(rnorm(J),mu.a,tau.a)

## hyperparams for b -- beta
mu.b <- mcmc.normal(1,1,0.0001)
tau.b <- mcmc.uniform(runif(1),0,100)
b <- mcmc.normal(rnorm(J),mu.b,tau.b)

tau.y <- mcmc.gamma(runif(1),0.01,0.01)
y.hat <- deterministic(function(a, b, group, SPX) { a[group] +  b[group] * SPX},
a, b, group, SPX)
y.lik <- mcmc.normal(y,mu=y.hat,tau=tau.y,observed=TRUE)
m <- create.model(mu.a, tau.a, a, mu.b, tau.b, b, tau.y, y.hat, y.lik)
```

Multilevel Beta estimation – results

```
runtime <- system.time(ans <- run.model(m, iterations=1e5L, burn=1e5L,
adapt=3e3L, thin=20L))
```

```
## runtime:  134.6
## acceptance ratio:  0.2161
```





Multilevel Beta estimation – alt. implementation

This alternative implementation uses a shortcut for grouped linear regression. However, the specification is a bit awkward because unlike R, there is no recycling in Armadillo. The dimension of b.mu and b.tau must match exacly the dimension of b.

```
SPX.1 <- cbind(1,SPX)

row.dup <- function(x,k) { as.matrix(rep(1,k)) %*% x }

b.mu <- mcmc.normal(c(0,rnorm(1)),mu=0,tau=1)
b.mu.rep <- deterministic(row.dup,b.mu,J)

b.tau <- mcmc.uniform(runif(2),lower=0,upper=100)
b.tau.rep <- deterministic(row.dup,b.tau,J)

b <- mcmc.normal(cbind(rnorm(J),rnorm(J)),b.mu.rep,b.tau.rep)
tau.y <- mcmc.gamma(runif(1),0.01,0.01)
y.hat <- linear.grouped(SPX.1,b,group)
y.lik <- mcmc.normal(y,mu=y.hat,tau=tau.y,observed=TRUE)
m <- create.model(b.mu, b.mu.rep, b.tau, b.tau.rep, b, tau.y, y.hat, y.lik)
```

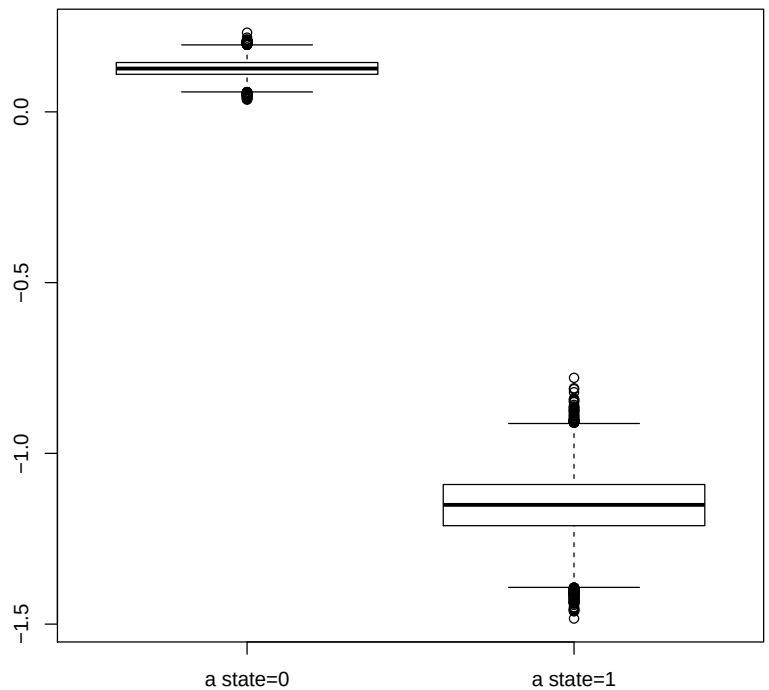
Regime switching model

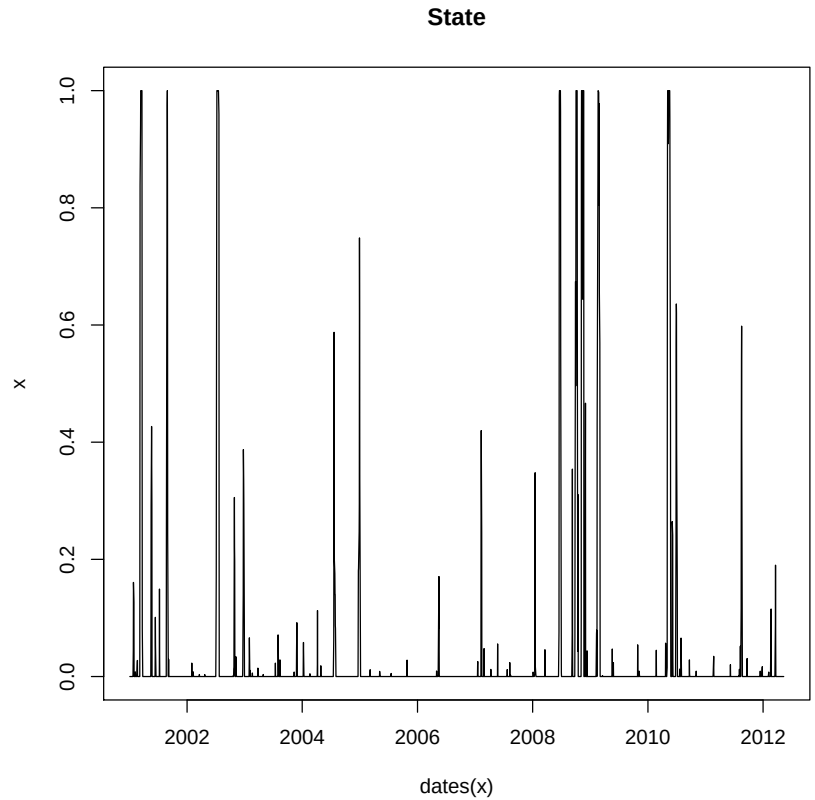
```
p <- mcmc.uniform(0.98, lower = 0.95, upper = 0.9999)
q <- mcmc.uniform(0.9, lower = 0.85, upper = p)
state <- as.double(runif(length(spx.ret)) > 0.95)

state.p <- deterministic(function(state, p, q) {
    c((1 - p), ifelse(state[-length(state)], q, 1 - p))
}, state, p, q)

state <- mcmc.bernoulli(state, p = state.p)
a <- mcmc.normal(rnorm(2), mu = c(1, -1), tau = 100)

y.hat <- deterministic(function(a, state) {
    a[state + 1]
}, a, state)

tau.y <- mcmc.gamma(runif(1), 0.01, 0.01)
y.lik <- mcmc.normal(spx.ret, mu = y.hat, tau = tau.y, observed = TRUE)

m <- create.model(p, q, state, state.p, a, y.hat, tau.y,
    y.lik)
```

Regime switching model – results

```
regime.trace <- run.model(m, iterations=2e5L, burn=5e5L, adapt=3e3L, thin=10L)
print(mean(regime.trace[["p"]]))

## [1] 0.9933

print(mean(regime.trace[["q"]]))

## [1] 0.8696

print(apply(regime.trace[["a"]],2,mean))

## [1]  0.127 -1.151
```

Regime switching model – state history





Conclusion

The next generation of R programmers needs something better than WinBUGS...
Please help me continue to build that tool.

Thanks!

Many people contributed ideas and helped debug work in progress as the package was being developed.

Conrad Sanderson for Armadillo.
Dirk Eddelbuettel and Romain Francois for Rcpp and RcppArmadillo.
Douglas Bates for feedback and ideas.
Martyn Plummer for feedback on JAGS.
Kurt Hornik and Uwe Ligges for putting up with my packaging.
John Laing and Thomas Harte for suffering through many discussions of bayesian methods.
Gyan Sinha for ideas and initial testing.
Yihui Xie for knitr.
Chris Fonnesbeck for PyMC.
Prof Brian Ripley for his extremely positive feedback on R-devl.
My wife for enduring many solitary hours while I built this package.