Wishart Multivaraite Response Linear Regression

John Tipton

December 13, 2015

Model

We begin with the model for the d-dimensional multivariate observation $\mathbf{y}_i$ is

$$\mathbf{y}_i \sim \mathrm{N}\left(X_i \boldsymbol{\beta}, \boldsymbol{\Sigma}\right),$$

where $\boldsymbol{\beta}$ is the p-dimensional set of regression coefficients for covariate X_i and the covariance matrix is

$$\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1^2 & \rho_{1,2}\sigma_1\sigma_2 & \cdots & \rho_{1,d}\sigma_1\sigma_d \\ \rho_{1,2}\sigma_1\sigma_2 & \sigma_2^2 & \cdots & \rho_{2,d}\sigma_2\sigma_d \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{1,d}\sigma_1\sigma_d & \rho_{2,d}\sigma_2\sigma_d & \cdots & \sigma_d^2 \end{pmatrix}.$$

We complete the model by assigning the priors

$$\begin{aligned} \boldsymbol{\beta} &\sim \mathrm{N}\left(\boldsymbol{\mu}_\beta, \boldsymbol{\Sigma}_\beta\right), \\ \rho_{j,k} &\sim \mathrm{Unif}\left(-1, 1\right), \\ \sigma_j &\sim \mathrm{Unif}\left(\sigma_l, \sigma_u\right), \end{aligned}$$

for $j, k = 1, \ldots, d$ and $k > j$.

Posterior

The posterior that we wish to sample is

$$
\begin{aligned}
[\boldsymbol{\beta}, \sigma_1, \ldots, \sigma_d, \rho_{1,2}, \ldots, \rho_{d-1,d} | \{\, \mathbf{y}_i, i = 1, \ldots, N \,\}] \propto & \left(\prod_{i=1}^{N} [\mathbf{y}_i | \boldsymbol{\beta}, \sigma_1, \ldots, \sigma_d, \rho_{1,2}, \ldots, \rho_{d-1,d}] \right) [\boldsymbol{\beta}] \left(\prod_{j=1}^{d} [\sigma_j] \right) \\
& \times \left(\prod_{j=1}^{d} \prod_{k=j+1}^{d} [\rho_{j,k}] \right) \\
\propto & \left(\prod_{i=1}^{N} |\boldsymbol{\Sigma}|^{-\frac{1}{2}} \exp\left\{ -\frac{1}{2} \left(\mathbf{y}_i - X_i \boldsymbol{\beta} \right)' \boldsymbol{\Sigma}^{-1} \left(\mathbf{y}_i - X_i \boldsymbol{\beta} \right) \right\} \right) \\
& \times |\boldsymbol{\Sigma}_\beta|^{-\frac{1}{2}} \exp\left\{ -\frac{1}{2} \left(\boldsymbol{\beta} - \boldsymbol{\mu}_\beta \right)' \boldsymbol{\Sigma}_\beta^{-1} \left(\boldsymbol{\beta} - \boldsymbol{\mu}_\beta \right) \right\} \\
& \times \prod_{j=1}^{d} I \left\{ \sigma_l < \sigma_j < \sigma_u \right\} \\
& \times \prod_{j=1}^{d} \prod_{k=j+1}^{d} I \left\{ -1 \leq \rho_{j,k} \leq 1 \right\}
\end{aligned}
$$

Full Conditionals

Full Conditional for β

$$
\begin{aligned}
[\boldsymbol{\beta} | \cdot] &\propto \prod_{i=1}^{N} [\mathbf{y}_i | \boldsymbol{\beta}, \boldsymbol{\Sigma}] \, [\boldsymbol{\beta}] \\
&\propto \prod_{i=1}^{N} \exp\left\{ -\frac{1}{2} \left(\mathbf{y}_i - \mathbf{X}_i \boldsymbol{\beta} \right)' \boldsymbol{\Sigma}^{-1} \left(\mathbf{y}_i - \mathbf{X}_i \boldsymbol{\beta} \right) \right\} |\boldsymbol{\Sigma}_\beta|^{-\frac{1}{2}} \exp\left\{ -\frac{1}{2} \left(\boldsymbol{\beta} - \boldsymbol{\mu}_\beta \right)' \boldsymbol{\Sigma}_\beta^{-1} \left(\boldsymbol{\beta} - \boldsymbol{\mu}_\beta \right) \right\} \\
&\propto \exp\left\{ -\frac{1}{2} \left(\boldsymbol{\beta}' \left(\sum_{i=1}^{N} \mathbf{X}_i' \boldsymbol{\Sigma}^{-1} \mathbf{X}_i + \boldsymbol{\Sigma}_\beta^{-1} \right) \boldsymbol{\beta} - 2 \boldsymbol{\beta}' \left(\sum_{i=1}^{N} \mathbf{X}_i' \boldsymbol{\Sigma}^{-1} \mathbf{y}_i + \boldsymbol{\Sigma}_\beta^{-1} \boldsymbol{\mu}_\beta \right) \right) \right\}
\end{aligned}
$$

which is $\mathrm{N}\left(\mathbf{A}^{-1} \mathbf{b}, \mathbf{A}^{-1} \right)$ with

$$
\begin{aligned}
\mathbf{A} &= \sum_{i=1}^{N} \mathbf{X}_i' \boldsymbol{\Sigma}^{-1} \mathbf{X}_i + \boldsymbol{\Sigma}_\beta^{-1} \\
\mathbf{b} &= \sum_{i=1}^{N} \mathbf{X}_i' \boldsymbol{\Sigma}^{-1} \mathbf{y}_i + \boldsymbol{\Sigma}_\beta^{-1} \boldsymbol{\mu}_\beta .
\end{aligned}
$$

Full Conditional for σ_j

For $j = 1, \ldots, d$,

$$
[\sigma_j | \cdot] \propto \prod_{i=1}^{N} |\boldsymbol{\Sigma}|^{-\frac{1}{2}} \exp\left\{ -\frac{1}{2} \left(\mathbf{y}_i - \mathbf{X}_i \boldsymbol{\beta} \right)' \boldsymbol{\Sigma}^{-1} \left(\mathbf{y}_i - \mathbf{X}_i \boldsymbol{\beta} \right) \right\} I \left\{ \sigma_l < \sigma_j < \sigma_u \right\}
$$

which can be sampled using Metropolis-Hastings

Full Conditional for $\rho_{j,k}$

For $j, k = 1, \ldots, d$ and $k > j$,

$$[\rho_{j,k}|\cdot] \propto \prod_{i=1}^{N} |\mathbf{\Sigma}|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\left(\mathbf{y}_i - \mathbf{X}_i\boldsymbol{\beta}\right)' \mathbf{\Sigma}^{-1}\left(\mathbf{y}_i - \mathbf{X}_i\boldsymbol{\beta}\right)\right\} I\left\{-1 \leq \rho_{j,k} \leq 1\right\}$$

which can be sampled using Metropolis-Hastings

Simulation and 'R' code

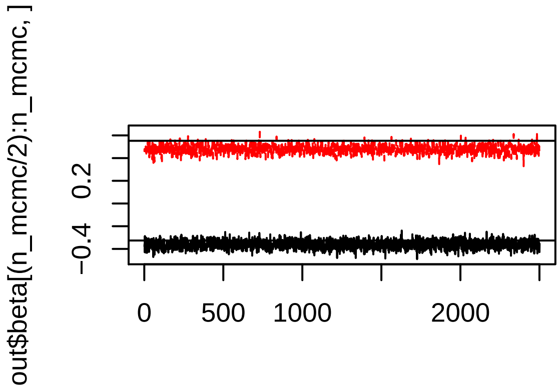
```
set.seed(101)
N <- 100
d <- 2
beta <- rnorm(d)
X <- array(rnorm(N * d^2), dim = c(N, d, d))
sigma_l <- 0
sigma_u <- 10
sigma <- runif(d, sigma_l, sigma_u)
rho <- runif(d * (d - 1)/2, -1, 1)

## Construct a generic dxd covariance matrix
makeCov <- function(d, sigma, rho) {
    sigmaMat <- sigma %*% t(sigma)
    rhoMat <- diag(d)
    rhoMat[lower.tri(rhoMat)] <- rho
    rhoMat <- rhoMat + t(rhoMat) - diag(d)
    # rhoMat[upper.tri(rhoMat)] <- t(rhoMat[lower.tri(rhoMat)])
    covMat <- sigmaMat * rhoMat
    return(covMat)
}

Sigma <- makeCov(d, sigma, rho)
## Simulate data
library(mvtnorm)
y <- matrix(0, N, d)
for (i in 1:N) {
    y[i, ] <- rmvnorm(1, X[i, , ] %*% beta, Sigma)
}


## MCMC function in R
mcmcR <- function(n_mcmc, y, mu_beta, Sigma_beta, sigma_l, sigma_u, sigma_tune = 1,
    rho_tune = 0.1) {

    library(mvtnorm)
    ## set up dimensions
```

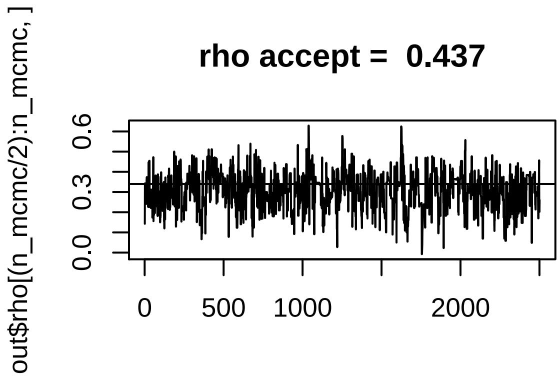
```r
N <- dim(y)[1]
d <- dim(y)[2]

## setup save variables
beta = matrix(0, n_mcmc, d)
sigma <- matrix(0, n_mcmc, d)
rho <- matrix(0, n_mcmc, d * (d - 1)/2)
Sigma_inv <- array(0, dim = c(n_mcmc, d, d))

## initialize values
beta[1, ] <- rmvnorm(1, mu_beta, Sigma_beta)
rho[1, ] <- runif(d * (d - 1)/2, -1, 1)
sigma[1, ] <- runif(d, sigma_l, sigma_u)
Sigma_inv[1, , ] <- solve(makeCov(d, sigma[1, ], rho[1, ]))
Sigma_beta_inv <- solve(Sigma_beta)
y_sum <- apply(y, 2, sum)
ty <- t(y)
sigma_tune <- rep(sigma_tune, d)
sigma_accept_tmp <- rep(0, d)
sigma_accept <- rep(0, d)
rho_tune <- rep(rho_tune, d * (d - 1)/2)
rho_accept_tmp <- rep(0, d * (d - 1)/2)
rho_accept <- rep(0, d * (d - 1)/2)


message(paste("Starting MCMC fit, will run for", n_mcmc, "iterations"))

## Start MCMC chain
for (k in 2:n_mcmc) {
    if (k%%500 == 0) {
        message(paste("Iteration", k))
    }

    ## sample beta
    A_inv <- solve(Reduce("+", lapply(seq_len(dim(X)[1]), function(i) {
        t(X[i, , ]) %*% Sigma_inv[k - 1, , ] %*% X[i, , ]
    })) + Sigma_beta_inv)
    b <- Reduce("+", lapply(seq_len(dim(X)[1]), function(i) {
        t(X[i, , ]) %*% Sigma_inv[k - 1, , ] %*% y[i, ]
    })) + Sigma_beta_inv %*% mu_beta
    beta[k, ] <- rmvnorm(1, A_inv %*% b, A_inv)

    ## sample sigma
    Sigma_inv[k, , ] <- Sigma_inv[k - 1, , ]
    sigma[k, ] <- sigma[k - 1, ]
    for (j in 1:d) {
        sigma_star <- sigma[k, ]
        sigma_star[j] <- rnorm(1, sigma[k, j], sigma_tune[j])
        if (sigma_star[j] > sigma_l && sigma_star[j] < sigma_u) {
            Sigma_inv_star <- solve(makeCov(d, sigma_star, rho[k - 1, ]))
            mh1 <- N * sum(log(diag(chol(Sigma_inv_star)))) - 0.5 * sum(unlist(lapply(seq_len(dim(X
              function(i) {
                t(y[i, ] - X[i, , ] %*% beta[k, ]) %*% Sigma_inv_star %*%
```

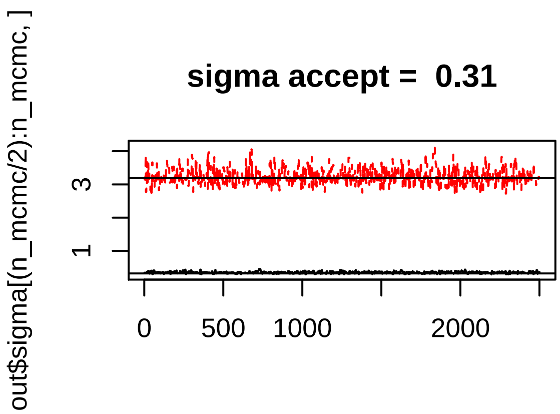
```r
                (y[i, ] - X[i, , ] %*% beta[k, ])
              })))
            mh2 <- N * sum(log(diag(chol(Sigma_inv[k, , ])))) - 0.5 * sum(unlist(lapply(seq_len(dim
              function(i) {
                t(y[i, ] - X[i, , ] %*% beta[k, ]) %*% Sigma_inv[k, , ] %*%
                  (y[i, ] - X[i, , ] %*% beta[k, ])
              })))
            mh <- exp(mh1 - mh2)
            if (mh > runif(1)) {
              sigma[k, ] <- sigma_star
              Sigma_inv[k, , ] <- Sigma_inv_star
              sigma_accept_tmp[j] <- sigma_accept_tmp[j] + 1/50
              sigma_accept[j] <- sigma_accept[j] + 1/n_mcmc
            }
        }
    }

    ## Update tuning
    if (k%%50 == 1) {
        for (j in 1:d) {
            if (sigma_accept_tmp[j] > 0.44) {
              sigma_tune[j] <- exp(log(sigma_tune[j]) + 1/sqrt(k))
            } else {
              sigma_tune[j] <- exp(log(sigma_tune[j]) - 1/sqrt(k))
            }
            sigma_accept_tmp[j] <- 0
        }
    }

    ## sample rho
    rho[k, ] <- rho[k - 1, ]
    for (j in 1:(d * (d - 1)/2)) {
        rho_star <- rho[k, ]
        rho_star[j] <- rnorm(1, rho[k, j], rho_tune[j])
        if (rho_star[j] >= -1 && rho_star[j] <= 1) {
            Sigma_inv_star <- solve(makeCov(d, sigma[k, ], rho_star))
            mh1 <- N * sum(log(diag(chol(Sigma_inv_star)))) - 0.5 * sum(unlist(lapply(seq_len(dim(X
              function(i) {
                t(y[i, ] - X[i, , ] %*% beta[k, ]) %*% Sigma_inv_star %*%
                  (y[i, ] - X[i, , ] %*% beta[k, ])
              })))
            mh2 <- N * sum(log(diag(chol(Sigma_inv[k, , ])))) - 0.5 * sum(unlist(lapply(seq_len(dim
              function(i) {
                t(y[i, ] - X[i, , ] %*% beta[k, ]) %*% Sigma_inv[k, , ] %*%
                  (y[i, ] - X[i, , ] %*% beta[k, ])
              })))
            mh <- exp(mh1 - mh2)
            if (mh > runif(1)) {
              rho[k, ] <- rho_star
              Sigma_inv[k, , ] <- Sigma_inv_star
              rho_accept_tmp[j] <- rho_accept_tmp[j] + 1/50
              rho_accept[j] <- rho_accept[j] + 1/n_mcmc
            }
```

```
            }
        }
        ## Update tuning
        if (k%%50 == 1) {
            for (j in 1:(d * (d - 1)/2)) {
                if (rho_accept_tmp[j] > 0.44) {
                  rho_tune[j] <- exp(log(rho_tune[j]) + 1/sqrt(k))
                } else {
                  rho_tune[j] <- exp(log(rho_tune[j]) - 1/sqrt(k))
                }
                rho_accept_tmp[j] <- 0
            }
        }
    }

    ## Output MCMC
    return(list(beta = beta, sigma = sigma, rho = rho, Sigma_inv = Sigma_inv,
        sigma_accept = sigma_accept, rho_accept = rho_accept))
}

## Define priors
mu_beta <- rep(0, d)
Sigma_beta <- 100 * diag(d)
sigma_l <- 0
sigma_u <- 10
n_mcmc <- 5000

## Run MCMC
out <- mcmcR(n_mcmc, y, mu_beta, Sigma_beta, sigma_l, sigma_u)
```

```
## Plot MCMC output (post burn-in)
layout(matrix(1:4, 2, 2))
matplot(out$beta[(n_mcmc/2):n_mcmc, ], type = 'l')
abline(h=beta)
matplot(out$sigma[(n_mcmc/2):n_mcmc, ], type = 'l', main=paste("sigma accept = ", round(mean(out$sigma_a
abline(h=sigma)
matplot(out$rho[(n_mcmc/2):n_mcmc, ], type = 'l', main=paste("rho accept = ", round(mean(out$rho_accept)
abline(h=rho)
```








```r
## Compare Estimates (post burn-in) to truth
beta
```

```
## [1] -0.3260365  0.5524619
```

```r
apply(out$beta[(n_mcmc/2):n_mcmc, ], 2, mean)
```

```
## [1] -0.3623669  0.4783602
```

```r
sigma
```

```
## [1] 0.3202498 3.1906751
```

```r
apply(out$sigma[(n_mcmc/2):n_mcmc, ], 2, mean)
```

```
## [1] 0.3518111 3.2408612
```

```r
rho
```

```
## [1] 0.3398204
```

```r
if(dim(out$rho)[2] == 1){
  mean(out$rho[(n_mcmc/2):n_mcmc, ])
} else {
  apply(out$rho[(n_mcmc/2):n_mcmc, ], 2, mean)
}
```

```
## [1] 0.3059642
```