

# Posterior Sampling with MCMC

Andy Shen, Devin Francom

LANL: CCS-6

## Overview

Say you have  $\mathbf{y}_1, \dots, \mathbf{y}_n \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$  where each  $\mathbf{y}_i$  is a vector of length  $p$ . Use  $n = 100$ ,  $p = 3$ ,  $\boldsymbol{\mu} = (1, 2, 3)$  and

$$\boldsymbol{\Sigma} = \begin{pmatrix} 1.0 & 1.4 & 2.1 \\ 1.4 & 4.0 & 4.2 \\ 2.1 & 4.2 & 9.0 \end{pmatrix}$$

to generate some data

**Task 1:** Use  $\mathbf{y}_1, \dots, \mathbf{y}_n \sim N(\boldsymbol{\mu}, \text{diag}(\sigma_1^2, \dots, \sigma_p^2))$  as your likelihood, with  $\boldsymbol{\mu} \sim N(\mathbf{m}, \mathbf{S})$  as your prior for  $\boldsymbol{\mu}$  and  $\sigma_i^2 \sim \text{InvGamma}(a, b)$  as your prior for  $\sigma_i^2$ . Use Gibbs sampling to sample the resulting posterior.

**Task 2:** Now use  $\mathbf{y}_1, \dots, \mathbf{y}_n \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$  as your likelihood, with  $\boldsymbol{\mu} \sim N(\mathbf{m}, \mathbf{S})$  as your prior for  $\boldsymbol{\mu}$  and the diagonal elements of  $\boldsymbol{\Sigma}$  as  $\sigma_i^2 \sim \text{InvGamma}(a, b)$  and each correlation parameter as  $\rho_{ij} \sim \text{Beta}(a, b)$ . Use Gibbs sampling and Metropolis-Hastings to sample the resulting posterior.

## Task 1

Our likelihood follows a multivariate normal distribution with mean  $\boldsymbol{\mu}$  and variance  $\text{diag}(\sigma_1^2, \dots, \sigma_p^2)$ .

We multiply the likelihood by both priors to get our posterior distribution,  $P(\boldsymbol{\mu}, \tilde{\boldsymbol{\Sigma}} \mid \alpha, \beta, \mathbf{m}, \mathbf{S})$ , where  $\tilde{\boldsymbol{\Sigma}} = \text{diag}(1, 4, 9)$ .

We get the following result:

$$P(\boldsymbol{\mu}, \tilde{\boldsymbol{\Sigma}} \mid \alpha, \beta, \mathbf{m}, \mathbf{S}) \propto \exp((\boldsymbol{\mu} - \mathbf{m})' \mathbf{S}^{-1} (\boldsymbol{\mu} - \mathbf{m})) \prod_{i=1}^{P=3} \{(\sigma_i^2)^{-\alpha-1} \exp(-\frac{\beta}{\sigma_i^2})\} \prod_{i=1}^{100} \{ \prod_{j=1}^3 [\sigma_j^{2-1/2}] \exp\{-\frac{1}{2}(\mathbf{y}_i - \boldsymbol{\mu})' \tilde{\boldsymbol{\Sigma}}^{-1} (\mathbf{y}_i - \boldsymbol{\mu})\} \}$$

Our  $\sigma_i^2$  values follow an inverse gamma distribution with parameters  $\alpha + 50$  and  $\beta + \frac{1}{2} \sum_{j=1}^{100} (y_{ji} - \mu_i)^2$

$$\sigma_i^2 \sim IG(\alpha + 50, \beta + \frac{1}{2} \sum_{j=1}^{100} (y_{ji} - \mu_i)^2)$$

Our  $\mu_i$  values follow a univariate normal distribution:

$$\mu_i \sim N(\frac{\sigma_i^2}{\sigma_i^2 + 100s_i^2} m_i + \frac{s_i^2 \sum_{j=1}^{100} y_{ji}}{\sigma_i^2 + 100s_i^2}, \frac{\sigma_i^2 s_i^2}{\sigma_i^2 + 100s_i^2})$$

## Posterior Distribution

```
set.seed(123)
library(mvtnorm)
library(invgamma)
mu <- c(1, 2, 3)
sig <- cbind(c(1, 1.4, 2.1), c(1.4, 4.0, 4.2), c(2.1, 4.2, 9.0))
data <- rmvnorm(1000, mu, sig)
n <- nrow(data)
str(data)
```

```
## num [1:1000, 1:3] 1.234 1.943 0.542 1.295 1.085 ...
```

```
library(invgamma)
sig_tild <- diag(c(1,4,9))
mu <- mu
S <- diag(3)
M <- c(0,0,0)
```

```
## FULL CONDITIONAL DISTRIBUTIONS ##
```

```
p_mu <- function(m = M, ST = sig_tild, .S = S, dat = data) {
  sums <- colSums(dat) #update ST
  vals <- rep(NA, 3)
  for(i in seq_len(3)) {
    sig_i <- ST[i,i]
    si <- .S[i,i]
    mi <- m[i]
    mean <- ((sig_i * mi) / (sig_i + n * si)) +
      ((si * sums[i]) / (sig_i + n * si))
    #print(mean)
    var <- (sig_i * si) / (sig_i + n * si)
    vals[i] <- rnorm(1, mean, sqrt(var))
  }
  #return(mean)
  return(vals)
}
```

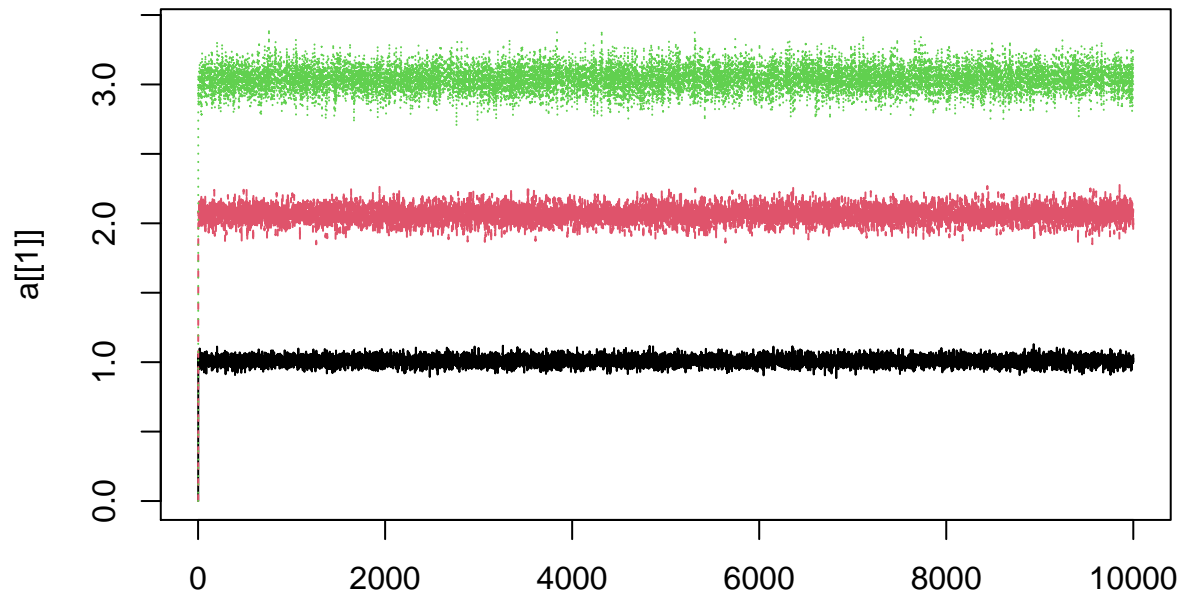
```
p_sig <- function(a = 2, b = 1, dat = data, .mu = mu) {
  centered <- t(t(dat) - .mu) #update .mu
  sums <- colSums(centered^2)
  vals <- rep(NA, 3)
  for(i in seq_len(3)) {
    beta <- 0.5 * sums[i] + b
    vals[i] <- 1/rgamma(1, shape = a + n/2, rate = beta)
  }
  return(vals)
}
```

```
gibbs <- function(its = 10000) {
  mat_sig <- matrix(NA, nrow = its, ncol = 3)
  mat_mu <- matrix(NA, nrow = its, ncol = 3)
  mat_sig[1,] <- rep(1, 3)
  mat_mu[1,] <- rep(0, 3)
```

```

for(i in 2:its) {
  # call p_sig with mu = mat_mu[i-1,]
  # call p_mu with updated value from p_sig
  mat_sig[i,] <- p_sig(.mu = mat_mu[i-1,])
  mat_mu[i,] <- p_mu(ST = diag(mat_sig[i,]))
}
list(mat_mu, mat_sig)
}
a <- gibbs()
matplot(a[[1]], type = "l")

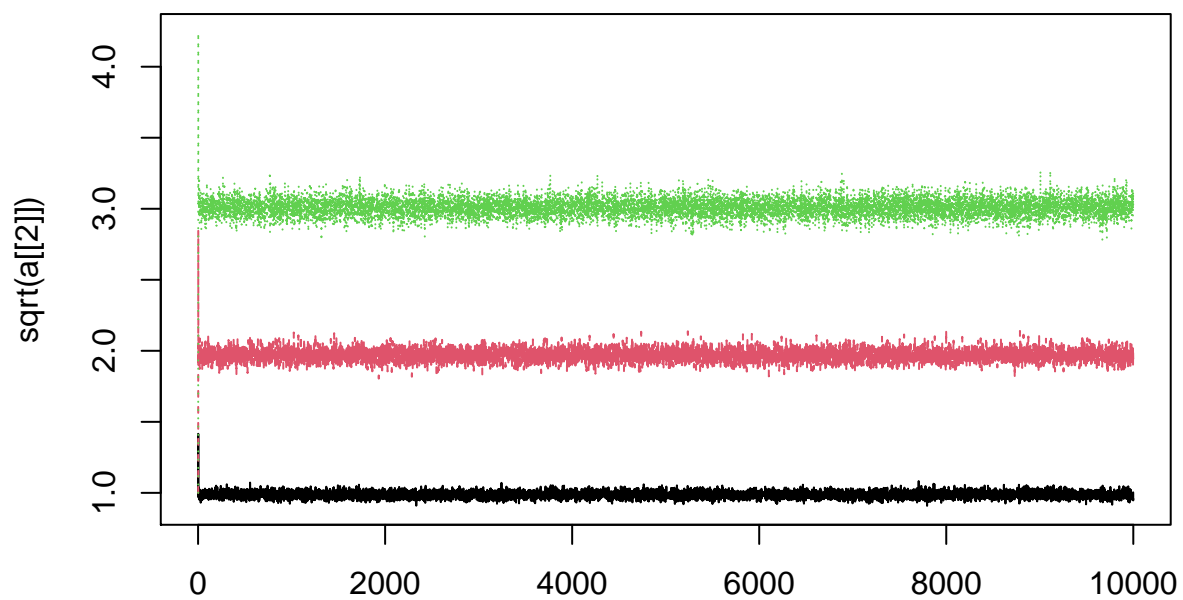
```



```

matplot(sqrt(a[[2]]), type = "l")

```



## Task 2

Our likelihood follows a multivariate normal distribution with mean  $\boldsymbol{\mu}$  and variance  $\text{diag}(\sigma_1^2, \dots, \sigma_p^2)$ .

We multiply the likelihood by both priors to get our posterior distribution,  $P(\boldsymbol{\mu}, \boldsymbol{\Sigma} \mid \alpha, \beta, \mathbf{m}, \mathbf{S})$ , where

$$\boldsymbol{\mu} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

and

$$\boldsymbol{\Sigma} = \begin{pmatrix} 1.0 & 1.4 & 2.1 \\ 1.4 & 4.0 & 4.2 \\ 2.1 & 4.2 & 9.0 \end{pmatrix}$$

## Beta-Gamma-MVN Method

### Setting Up Data and Priors

We start by generating the same data from above:

```
set.seed(123)
library(mvtnorm)
library(invgamma)
mu <- c(1, 2, 3)
sig <- cbind(c(1, 1.4, 2.1), c(1.4, 4.0, 4.2), c(2.1, 4.2, 9.0))
data <- rmvnorm(100, mu, sig)
n <- nrow(data)
str(data)
```

```
##  num [1:100, 1:3] 1.234 1.943 0.542 1.295 1.085 ...
```

```
S <- diag(3)
Sinv <- solve(S)
m <- c(0,0,0)
```

We have that

$$\boldsymbol{\mu} \mid \cdot \sim \mathcal{N}((\mathbf{S}^{-1} + n\boldsymbol{\Sigma}^{-1})^{-1}(\mathbf{S}^{-1}\mathbf{m} + n\boldsymbol{\Sigma}^{-1}\bar{\mathbf{y}}), (\mathbf{S}^{-1} + n\boldsymbol{\Sigma}^{-1})^{-1})$$

which is the full conditional distribution for  $\boldsymbol{\mu}$ .

Recall that for our correlation coefficient  $\rho_i$ , we have that

$$\rho_{ij} = \rho_{ji} = \frac{\text{cov}(i, j)}{\sigma_i \sigma_j}$$

which we simplify to  $\rho_i$  and work with the upper and lower triangular portion of the matrices.

### Full Conditionals

```

p_mu <- function(n = nrow(data), ST) {
  mean <- solve(Sinv + n * solve(ST)) %*% (Sinv %*% m + n * solve(ST) %*% colMeans(data))
  covariance <- solve(Sinv + n * solve(ST))
  rmvnorm(1, mean, covariance)
}

p_sig <- function(alpha = 1, beta = 1, ST, sig_sq, .mu) { # Inverse Gamma Prior
  full_cond <- rep(NA, 3)
  prior <- (-alpha - 1) * log(sig_sq) + (-beta/sig_sq)

  mat <- t(t(data) - .mu)
  mat2 <- (mat %*% solve(ST)) * mat
  rsums <- rowSums(mat2)

  sum_all <- -0.5 * sum(rsums)
  like <- -0.5 * (determinant(ST, logarithm = TRUE)$modulus) + (sum_all)
  full_cond <- prior + like

  (full_cond)
}

p_sig2 <- function(alpha = 1, beta = 1, ST, sig_sq, .mu) { # Inverse Gamma Prior
  full_cond <- rep(NA, 3)
  prior <- (-alpha - 1) * log(sig_sq) + (-beta/sig_sq)
  in_sum <- rep(NA, n)
  for(i in seq_len(n)) {
    in_sum[i] <- t(data[i,] - .mu) %*% solve(ST) %*% (data[i,] - .mu)
  }
  sum_all <- -0.5 * sum(in_sum)
  like <- -0.5 * (determinant(ST, logarithm = TRUE)$modulus) + (sum_all)
  full_cond <- prior + like

  (full_cond)
}

p_rho <- function(rho, a = 2, b = 3, .mu, ST) { # Beta Prior
  full_cond <- rep(NA, 3)
  prior <- (a-1) * log(rho) + (b-1) * log(1-rho)
  mat <- t(t(data) - .mu)
  mat2 <- (mat %*% solve(ST)) * mat
  rsums <- rowSums(mat2)
  sum_all <- -0.5 * sum(rsums)
  like <- -0.5 * (determinant(ST, logarithm = TRUE)$modulus) + (sum_all)
  full_cond <- prior + like

  (full_cond)
}

system.time(p_sig(ST = sig, sig_sq = 1, .mu = c(0,0,0)))

##      user  system elapsed
##    0.027   0.001   0.027

```

```
system.time(p_sig2(ST = sig, sig_sq = 1, .mu = c(0,0,0)))
```

```
##      user  system elapsed  
## 0.021   0.000   0.021
```

## Metropolis-Hastings and Gibbs

```
set.seed(12)  
library(dplyr)
```

```
##  
## Attaching package: 'dplyr'  
  
## The following objects are masked from 'package:stats':  
##  
##      filter, lag  
  
## The following objects are masked from 'package:base':  
##  
##      intersect, setdiff, setequal, union
```

```
library(truncnorm)  
met_gibbs <- function(its) {  
  
  # For sigma, use truncated normal as proposal  
  propose_sig <- function(x, mn, sd = 0.1) {  
    #browser()  
    log(dtruncnorm(x, a = 0, mean = mn, sd = sd))  
  }  
  
  propose_rho <- function(x, mn, sd = 0.8) {  
    log(dtruncnorm(x, a = -1, b = 1, mean = mn, sd = sd))  
  }  
  
  met_mu <- matrix(NA, its, 3)  
  met.mu.start <- rep(0, 3)  
  met_mu[1,] <- met.mu.start  
  
  mat_sig <- matrix(NA, nrow = its, ncol = 3)  
  mat_rho <- matrix(NA, nrow = its, ncol = 3)  
  mat_sig[1,] <- rep(1, 3)  
  mat_rho[1,] <- rep(0, 3) #12, 13, 23  
  a_sig <- 0  
  a_rho <- 0  
  for(i in 2:its) {  
    # sample mu with mvn  
    #sample sigma given mu  
    #sample rho given sigma and mu & rtruncnorm between (-1,1)  
    current.mu <- met_mu[i-1,]  
  
    cor_to_cov <- c(  
      (mat_sig[i-1,1] * mat_sig[i-1,2]),  
      (mat_sig[i-1,1] * mat_sig[i-1,3]),  
      (mat_sig[i-1,2] * mat_sig[i-1,3])  
    )  
  }  
}
```

```

) #convert correlation to covariance

cov <- diag(mat_sig[i-1,])
cov[upper.tri(cov)] <- mat_rho[i-1,] * cor_to_cov
cov[lower.tri(cov)] <- mat_rho[i-1,] * cor_to_cov
#browser()
met_mu[i,] <- p_mu(ST = cov) #Done sampling mu

### SIGMA ###

for(j in 1:3) {
  cov2 <- cov
  candidate_sig <- rtruncnorm(
    1, a = 0, mean = met_mu[i,j], sd = 0.1
  )

  cov2[j,j] <- candidate_sig

  ratio_sig <-
    (p_sig(ST = cov2, sig_sq = cov2[j,j], .mu = met_mu[i,]) -
     propose_sig(candidate_sig, mn = cov[j,j]) ) -
    (p_sig(ST = cov, sig_sq = cov[j,j], .mu = met_mu[i,]) -
     propose_sig(cov[j,j], mn = candidate_sig ))

  pmove <- min(0, ratio_sig)
  #if(is.na(pmove)) browser()
  u <- log(runif(1))

  if(u < pmove) {
    mat_sig[i,j] <- candidate_sig
    a_sig = a_sig + 1
    cov <- cov2
  }
  else {
    mat_sig[i,j] <- mat_sig[i-1,j]
  }
}

### RHO ###
#browser()
cor_to_cov3 <- c(
  sqrt(mat_sig[i,1] * mat_sig[i,2]),
  sqrt(mat_sig[i,1] * mat_sig[i,3]),
  sqrt(mat_sig[i,2] * mat_sig[i,3])
) #convert correlation to covariance

#browser()
for(k in 1:3) {
  candidate_rho <- rtruncnorm(
    1, a = -1, b = 1, mean = met_mu[i,k], sd = 0.09
  )

```

```

cov3 <- cov
cov3[upper.tri(cov3)] <- candidate_rho * cor_to_cov3
cov3[lower.tri(cov3)] <- candidate_rho * cor_to_cov3
#browser()
ratio_cov <-
(p_rho(rho = candidate_rho, .mu = met_mu[i,], ST = cov3) -
 propose_rho(candidate_rho, mn = mat_rho[i-1,k])) -
(p_rho(rho=(cov3[upper.tri(cov3)][k])/(cor_to_cov3[k]), .mu = met_mu[i,], ST = cov3) -
 propose_rho(mat_rho[i-1,k], mn = candidate_rho))
#browser()
pmove <- min(0, ratio_cov)
#if(is.na(pmove)) browser()
u <- log(runif(1))
if(u < pmove) {
  #if(is.na(candidate_rho)) browser()
  mat_rho[i,k] <- candidate_rho
  a_rho = a_rho + 1
  cov <- cov3
}
else {
  mat_rho[i,k] <- mat_rho[i-1,k]
}
}
}
list(met_mu, mat_sig^2, mat_rho, a_sig, a_rho)
}

its <- 1000
a <- met_gibbs(its = its)

colMeans(a[[1]])

## [1] 0.9242297 1.7671005 2.9044598

colMeans(a[[2]])

## [1] 1.406738 4.721645 9.531596

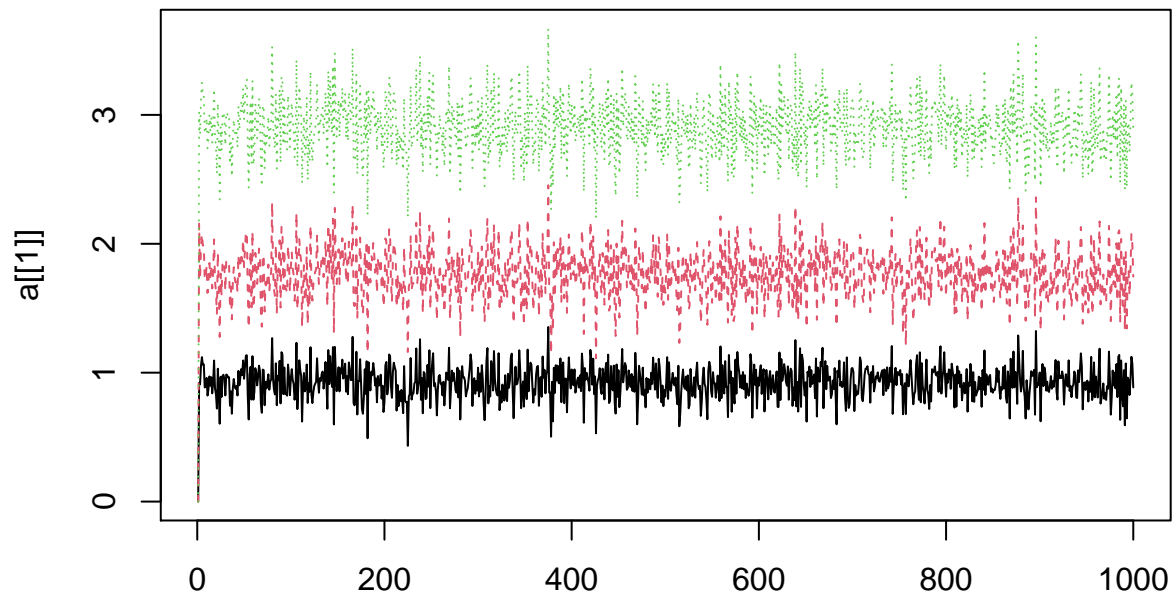
colMeans(a[[3]])

## [1] 0.8833487 0.9881884 0.9947751

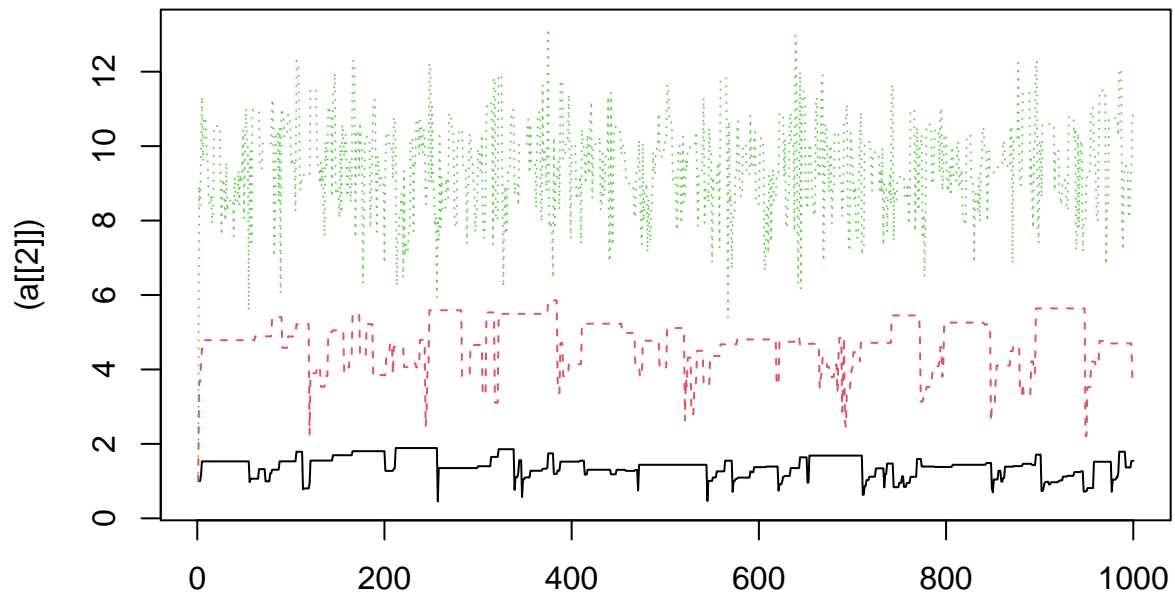
matplot(a[[1]], type = "l")

```

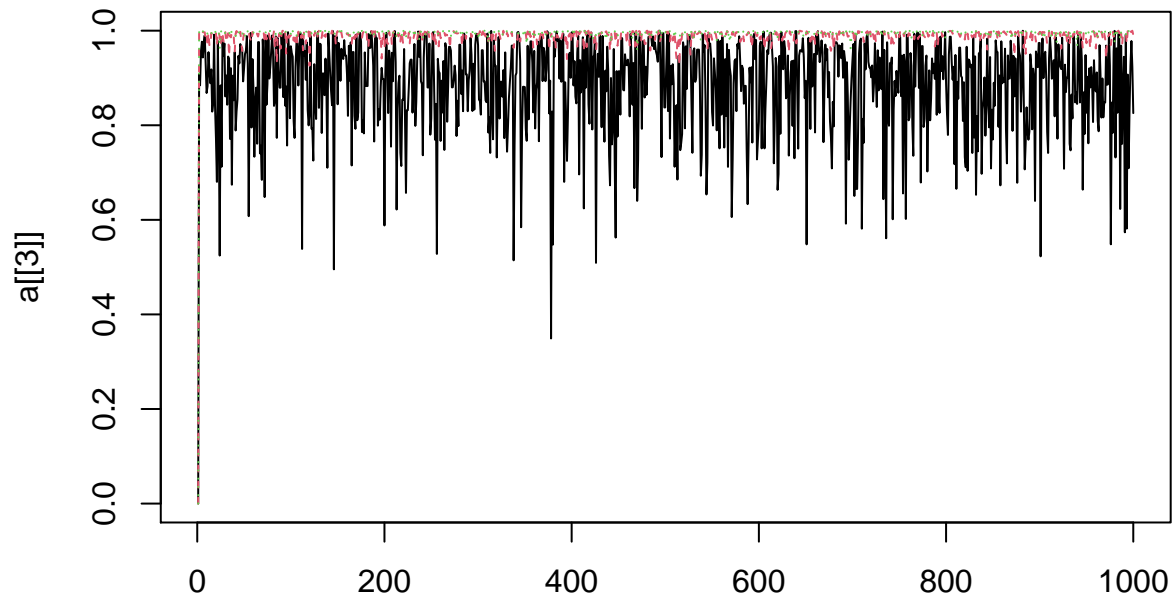




```
matplotlib((a[[2]]), type = "l")
```



```
matplotlib(a[[3]], type = "l")
```



```
a[[4]]/its
```

```
## [1] 0.827
```

```
a[[5]]/its
```

```
## [1] 2.927
```

```
sig <- 1:3
rho <- 4:6
lol <- 4:6
mat <- diag(sig)
mat[upper.tri(mat)] <- rho
mat[lower.tri(mat)] <- lol
mat
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    5
## [2,]    4    2    6
## [3,]    5    6    3
```

```
mat2 <- mat
o <- rnorm(3); o
```

```
## [1]  2.0553089 -0.1402064 -1.4958225
```

```
for(i in 1:nrow(mat)) {
  mat2[i,i] <- o[i]
}
mat2
```

```
##      [,1]      [,2]      [,3]
## [1,] 2.055309 4.0000000 5.0000000
## [2,] 4.000000 -0.1402064 6.0000000
## [3,] 5.000000 6.0000000 -1.4958223
```

```
mat2[upper.tri(mat)] * c(1,0, 10)
```

```
## [1]  4  0 60
```

```
mat2
```

```
##           [,1]      [,2]      [,3]
## [1,] 2.055309  4.000000  5.000000
## [2,] 4.000000 -0.1402064 6.000000
## [3,] 5.000000  6.000000 -1.495823
```

## Inverse-Wishart Method

We will largely use the precision matrix,  $\Lambda$  which is simply just  $\Sigma^{-1}$ .

In our case, we have that

$$\Lambda = \begin{pmatrix} 2.361 & -0.486 & 0.324 \\ -0.486 & 0.590 & -0.162 \\ -0.324 & -0.162 & 0.262 \end{pmatrix}$$

It then follows that

$$\mathbf{y}_i \mid \boldsymbol{\mu}, \Lambda \sim \mathcal{N}(\boldsymbol{\mu}, \Lambda^{-1})$$

$$\Lambda \sim W(\mathbf{S}, n_0)$$

From this intuition, and leveraging the fact that  $\Lambda = \Sigma^{-1}$ , it then follows the full conditional of our covariance matrix  $\Sigma$  follows an Inverse Wishart Distribution:

$$\Sigma \sim W^{-1}(\mathbf{S}^{-1} + \sum_{j=1}^{100} (\mathbf{y}_j - \boldsymbol{\mu})(\mathbf{y}_j - \boldsymbol{\mu})', n_0 + n)$$

where  $n_0$  represents the degrees of freedom such that  $n_0 = p - 1$  where  $p = \dim(S)$ , where  $S$  is a positive-definite matrix.

## The Algorithms

```
mu <- mu
S <- diag(3)
Sinv <- solve(S)
M <- c(0,0,0)
Lam <- solve(sig); Lam

## Error in solve.default(sig): 'a' (3 x 1) must be square
## Error in eval(expr, envir, enclos): object 'Lam' not found

p <- dim(sig)[1]

library(MCMCpack)

## Loading required package: coda
```

```

## Loading required package: MASS

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##      select

## ##
## ## Markov Chain Monte Carlo Package (MCMCpack)
## ## Copyright (C) 2003-2020 Andrew D. Martin, Kevin M. Quinn, and Jong Hee Park
## ##
## ## Support provided by the U.S. National Science Foundation
## ## (Grants SES-0350646 and SES-0350613)
## ##

##
## Attaching package: 'MCMCpack'

## The following objects are masked from 'package:invgamma':
##
##      dinvgamma, rinvgamma

p_muvec <- function(n = nrow(data)) {
  mean <- solve(Sinv + n * Lam) %*% (Sinv %*% M + n * Lam %*% colMeans(data))
  cov <- solve(Sinv + n * Lam)
  rmvnorm(1, mean, cov)
}

p_cov <- function(mu, df = p, n = nrow(data)) {
  summation <-
  pdmat <- Sinv + summation
  riwish(df + n, pdmat)
}

gibbs <- function(its = 10) {
  mat_sig <- array(NA, dim = c(3, 3, its))
  mat_mu <- matrix(NA, nrow = its, ncol = 3)
  mat_sig[1,1,1] <- diag(3)
  mat_mu[1,] <- rep(0, 3)
  for(i in 2:its) {
    # call p_sig with mu = mat_mu[i-1,]
    # call p_mu with updated value from p_sig
    mat_sig[i,] <- p_cov()
    mat_mu[i,] <- p_mu(ST = diag(mat_sig[i,]))
  }
  list(mat_mu, mat_sig)
}

a <- array(NA, c(2,2,5))
select_ar <- matrix(ncol = 2, byrow = TRUE, c(
  1,2,
  2,1))
a[1:2,1:2,1:3]

```

```
## , , 1
##
##      [,1] [,2]
## [1,]    NA    NA
## [2,]    NA    NA
##
## , , 2
##
##      [,1] [,2]
## [1,]    NA    NA
## [2,]    NA    NA
##
## , , 3
##
##      [,1] [,2]
## [1,]    NA    NA
## [2,]    NA    NA
```