MA316 Assignment 3

18343040 Ziyang He

December 29, 2021

1 Exercise 3.21

Question In example 18.2, let $(x_1, x_2, x_3, x_4, x_5) = (82, 72, 45, 34, 17)$. Choose the appropriate warm-up period and simulation, and use R to calculate the posterior mean estimate of β. Starting from different initial values, do a few more times to investigate the size of the estimated error.

Solution Example 18.2 introduces a Bayesian financial problem. Assuming that there are 5 stocks with $n = 250$ trading days' yield records, let X_i denotes the number of times the i-th stock's yield has the highest return rate in n trading days. Suppose that $X_1, \ldots, X_5$) follows a multinomial distribution with probability

$$\mathbf{p} = (\frac{1}{3}, \frac{1-\beta}{3}, \frac{1-2\beta}{3}, \frac{2\beta}{3}, \frac{\beta}{3}),$$

where $\beta \in (0, 0.5)$ is unknown. Assuming that β has a prior distribution $p_0(\beta) \sim U(0, 0.5)$, the posterior distribution of β is

$$\begin{aligned} f(\beta|x_1, \ldots, x_5) &\propto p(x_1, \ldots, x_5|\beta)p_0(\beta) \\ &\propto (1-\beta)^{x_2}(1-2\beta)^{x_3}\beta^{x_4+x_5} I_{(0,0.5)}(\beta) = \widetilde{\pi}(\beta). \end{aligned}$$

Due to the difficulty of sampling directly from the posterior distribution of β, use Metropolis-Hasting Sampling (MH method) to generate a sequence of β. Suppose we have $\beta^{(t)}$, set trial sampling distribution $T(y|\beta^{(t)})$ as $U(0, 0.5)$,

$$r(\beta^{(t)}, y) = \min\left(1, \frac{\widetilde{\pi}(y)}{\widetilde{\pi}(\beta^{(t)})}\right),$$

sample y from $U(0, 0.5)$ and accept $\beta(t+1) = y$ with probability $r(\beta(t), y)$.

The following part shows the R codes for calculating the posterior mean estimate and the standard error of β using the default initial values. The warm-up period is set to be the first 20% of the iteration.

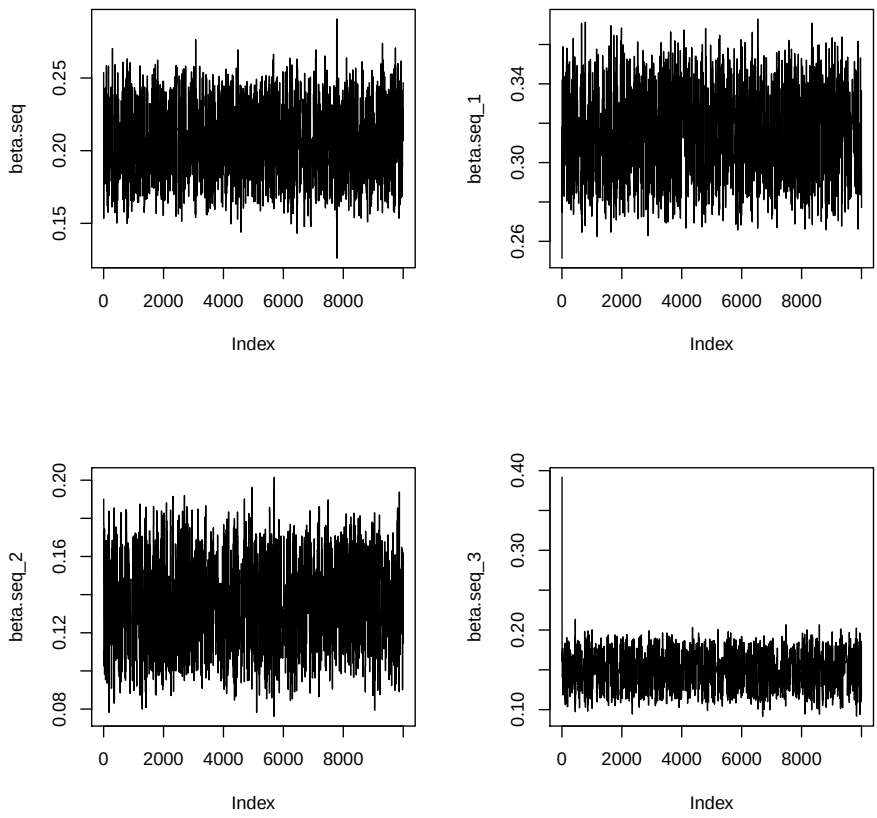
```
# set random seed
set.seed(10)
```

```r
## Sample
x <- c(82, 72, 45, 34, 17)

## Distribution Function
p <- function(beta, x) {
  out <-
    (1 - beta) ^ x[2] * (1 - 2 * beta) ^ x[3] *
    beta ^ (x[4] + x[5]) * (beta > 0) * (beta < 0.5) * 1
  out
}

## Probability Function
r <- function(beta_t, y, x) {
  out <- min(1, p(y, x) / p(beta_t, x))
  out
}

## MH Algorithm
rmh <- function(x, iter) {
  beta <- runif(n = 1, min = 0, max = 0.5)
  beta.seq <- c(beta)
  for (i in 1:iter) {
    y <- runif(n = 1, min = 0, max = 0.5)
    if (y <= r(beta, y, x)) {
      beta <- y
    }
    beta.seq <- c(beta.seq, beta)
  }
  beta.seq
}

# Calculate the beta sequence
beta.seq <- rmh(x, 1e4)

# Calculate the posterior mean estimate of beta
beta.bar <- mean(beta.seq[-1:-2e3])
beta.bar

## [1] 0.2080086

# Calculate the standard error of beta
beta.SE <- sqrt(var(beta.seq[-1:-2e3]) / 8e3)
beta.SE

## [1] 0.000287239
```

In order to investigate the estimated error of different initial values, randomly generate three sets for simulation. The R codes are shown as below.

```r
# Generate Different Initial Values
ini.val <- function(m = 250, n = 5) {
  out <-
    diff(c(0, sort(
      sample(
        x = 1:m,
        size = 4,
        replace = FALSE,
        prob = rep(1 / m, m)
      )
    ), m))
  return(out)
}

# Calculate the beta sequence
beta.seq_1 <- rmh(x_1 <- ini.val(), 1e4)
beta.seq_2 <- rmh(x_2 <- ini.val(), 1e4)
beta.seq_3 <- rmh(x_3 <- ini.val(), 1e4)

# Calculate the posterior mean estimate of beta
beta.bar_1 <- mean(beta.seq_1[-1:-2e3])
beta.bar_2 <- mean(beta.seq_2[-1:-2e3])
beta.bar_3 <- mean(beta.seq_3[-1:-2e3])
beta.bar.all <-
  data.frame(
    Default = beta.bar,
    Rand_1 = beta.bar_1,
    Rand_2 = beta.bar_2,
    Rand_3 = beta.bar_3
  )
beta.bar.all

##     Default    Rand_1    Rand_2    Rand_3
## 1 0.2080086 0.3159344 0.1367417 0.1497569

# Calculate the standard error of beta
beta.SE_1 <- sqrt(var(beta.seq_1[-1:-2e3]) / 8e3)
beta.SE_2 <- sqrt(var(beta.seq_2[-1:-2e3]) / 8e3)
beta.SE_3 <- sqrt(var(beta.seq_3[-1:-2e3]) / 8e3)
beta.SE.all <- data.frame(
  Default = beta.SE,
  Rand_1 = beta.SE_1,
  Rand_2 = beta.SE_2,
```

```
  Rand_3 = beta.SE_3
)
beta.SE.all

##       Default       Rand_1       Rand_2       Rand_3
## 1 0.000287239 0.0002581311 0.0002601667 0.0002606555

# Plot the beta sequence
par(mfrow = c(2, 2))
plot(beta.seq, type = 'l')
plot(beta.seq_1, type = 'l')
plot(beta.seq_2, type = 'l')
plot(beta.seq_3, type = 'l')
```



From the results above we can see that the posterior mean estimates of β are related to the initial values, while the standard error of the estimates are

basically the same.

2 Exercise 3.25

Question In example 18.5, let $n = 20, \alpha = \beta = 0.5$. Generate a Gibbs sampler of (X, Y), and compare the histogram of Y and the probability density function of $Beta(\alpha, \beta)$.

Solution The target distribution is

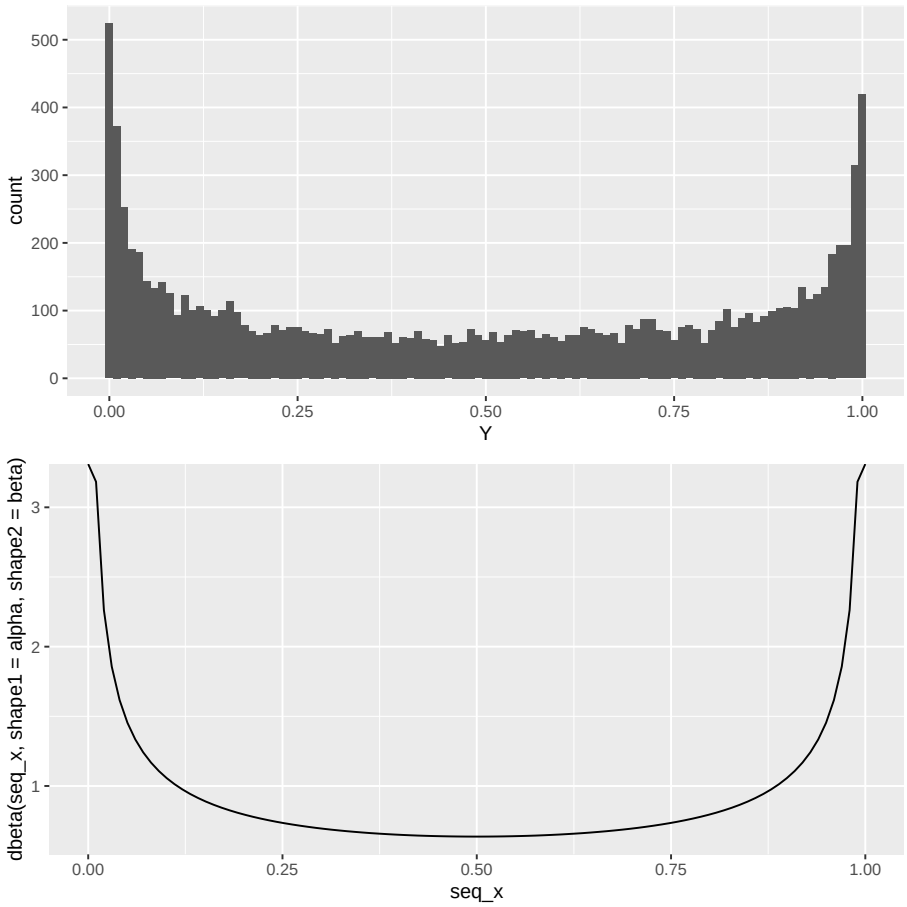
$$\pi(x, y) \propto C_n^x y^{x+\alpha-1}(1-y)^{n-x+\beta-1}, x = 0, 1, \ldots, n, 0 \leq y \leq 1,$$

therefore $X|Y \sim B(n, y), Y|X \sim Beta(x+\alpha, n-x+\beta)$. Notice that the marginal distribution of Y is $Beta(\alpha, \beta)$, hence use Gibbs Sampling to generate a sample sequence of (X, Y).

The following part shows the R codes for using Gibbs Sampling to generate the sample sequence.

```r
## Packages
library(ggplot2)
library(gridExtra)

## Parameters
n <- 20
alpha <- 0.5
beta <- 0.5
N <- 1e4

## Target Distribution
ptd <- function(n, x, y, alpha, beta) {
  out <-
    choose(n, x) * y ^ (x + alpha - 1) *
    (1 - y) ^ (n - x + beta - 1)
  out
}

## Gibbs Sampling
x <- 0
y <- rbeta(n = 1, shape1 = alpha, shape2 = beta)
X <- c(x)
Y <- c(y)
avg_Y <- c(mean(Y))
for (i in 1:N) {
  x <- rbinom(n = 1, size = n, prob = y)
  X <- c(X, x)
```

```
  y <- rbeta(n = 1,
             shape1 = x + alpha,
             shape2 = n - x + beta)
  Y <- c(Y, y)
  avg_Y <- c(avg_Y, mean(Y))
}

## Histogram
data <- data.frame(X, Y)
p1 <-
  ggplot(data = data, mapping = aes(x = Y)) +
  geom_histogram(binwidth = 0.01)

## Density of Beta(alpha,beta)
seq_x <- seq(0, 1, length = 100)
p2 <- qplot(
  x = seq_x,
  y = dbeta(seq_x, shape1 = alpha, shape2 = beta),
  geom = "line"
)

grid.arrange(p1, p2, nrow = 2)
```



As can be seen, the shapes of both histogram and probability density function are basically the same.

3 Exercise 3.28

Question In phase modulation communication, consider the following state space model

$$X_t = \phi_1 X_{t-1} + \eta_t, \eta_t \sim N(0, \sigma_\eta^2), t = 1, 2, \ldots, n,$$
$$y_t = A\cos(ft + X_t) + \varepsilon_t, \varepsilon_t \sim N(0, \sigma_\varepsilon^2), t = 1, 2, \ldots, n,$$

where $\phi_1 = 0.6, \sigma_\eta^2 = 1/6, A = 320, f = 1.072 \times 10^7, \sigma_\varepsilon^2 = 1$. $(y_1, \ldots, y_n)$ are the observed values, $X_1, \ldots, X_n)$ are unobservable random variables.

(1) Let $X_0 = 0, n = 128$, generate $(X_t, y_t), t = 1, 2, \ldots, n$.

(2) Design the SIS algorithm to generate appropriate weighted samples of $X_1, \ldots, X_n$

under the conditions of known $y_1, \ldots, y_n$. Independently generate $N = 10000$ sets and use the forward step of the above state space model.

(3) Consider the distribution of the obtained weights $\{W_i\}$.

(4) Conduct Residual Resampling on every step of the SIS.

(5) According to the posterior mean method, use the above improved sampling to estimate $(X_1, \ldots, X_n)$.

(6) For each X_t, calculate the standard error of the posterior estimate.

(7) Repeat the estimation process for $M = 400$ times independently, calculate the new estimated standard error from M different posterior estimates, and compare with the result obtained in (6).

Solution

(1) Simply generate the sequence of (X, Y) by the equations of the state space model. The following R codes accomplish the task.

```r
## Parameters
phi_1 <- 0.6
sigma2_n <- 1 / 6
A <- 320
f <- 1.072 * 1e7
sigma2_eps <- 1

## Generating Function
rpair <- function(X_old, t, N) {
  eta_new <- rnorm(n = N,
                   mean = 0,
                   sd = sqrt(sigma2_n))
  eps_new <- rnorm(n = N,
                   mean = 0,
                   sd = sqrt(sigma2_eps))
  X_new <- phi_1 * X_old + eta_new
  y_new <- A * cos(f * t + X_new) + eps_new
  out <- data.frame(X_t = X_new, y_t = y_new)
  return(out)
}


X_0 <- 0
n <- 128
t <- 1
X <- c()
y <- c()
```

```
while (t <= n) {
  pair_t <- rpair(X_0, t, 1)
  X <- c(X, pair_t$X_t)
  y <- c(y, pair_t$y_t)
  t <- t + 1
}
```

(2) This is a filter smoothing problem. Given $y_1, \ldots, y_n$, the posterior sample can be produced using SIS algorithm. Let the trial sampling density be $g_t(x_t|\mathbf{x}_{t-1}) = q_t(x_t|x_{t-1})$ (Markov hypothesis). For $t = 1, \pi_1(x) \propto f_1(y_1|x_1)p(x_1)$, where $p(x_1)$ is the prior probability density function. Sample $X_1 \sim g_1(x_1)$, and let $g_1(x_1) = \pi_1(x_1)$ if possible.

In this state space model, suppose the prior distribution of X_1 to be $U(-1, 1)$. Due to the difficulty of sampling X_1 from $\pi_1(x_1)$, we hereby just simplify the situation and choose $g_1(x_1) = p(x_1)$.

The following R codes show the procedure to independently generate $N = 10000$ weighted samples using the sequential importance sampling.

```
N <- 1e4
X_SIS <- runif(n = 1, min = -1, max = 1)
X_t <- X_SIS
W_t <- rep(1, N)
W_SIS <- rbind(W_1 = W_t)

t <- 2
while (t <= n) {
  X_t <- rpair(X_t, t, N)$X_t
  U_t <-
    pnorm(
      q = y[t],
      mean = A * cos(f * t + X_t),
      sd = sqrt(sigma2_eps)
    )
  W_t <- W_t * U_t
  X_SIS <- rbind(X_SIS, X_t)
  W_SIS <- rbind(W_SIS, W_t)
  t <- t + 1
}
```

(3) The following part shows the first ten weights of the first five streams.

```
head(W_SIS[, 1:5],10)
```

```
##            [,1] [,2] [,3]         [,4]         [,5]
## W_1 1.000000000    1    1 1.000000e+00  1.00000e+00
## W_t 1.000000000    1    0 1.000000e+00  1.00000e+00
## W_t 0.006279899    1    0 1.000000e+00 3.50387e-100
## W_t 0.006279899    1    0 1.000000e+00 3.50387e-100
## W_t 0.006279899    1    0 6.476871e-41 3.50387e-100
## W_t 0.006279899    1    0 6.476871e-41 3.50387e-100
## W_t 0.006279899    1    0 6.476871e-41 3.50387e-100
## W_t 0.000000000    0    0 0.000000e+00  0.00000e+00
## W_t 0.000000000    0    0 0.000000e+00  0.00000e+00
## W_t 0.000000000    0    0 0.000000e+00  0.00000e+00
```

Notice that the weights quickly converge to 0 in the first five streams. Actually, all the weights converge to 0, which inspires us to improve the SIS algorithm by adjusting the weights in each step, leading to residual resampling.

(4) The residual resampling is more efficient, since the weight of each stream is adjusted to be the same, and has a smaller simulation error. Shown as below are the R codes for residual resampling.

```
residual_resampling <- function(y) {
  X_SIS_RR <- runif(N, min = -1, max = 1)
  pi_1 <- function(X_1) {
    out <- exp(-(y[1] - A * cos(f * 1 + X_1)) ^ 2 /
                 (2 * sigma2_eps))
    return(out)
  }
  X_t <- X_SIS_RR
  W_t <- pi_1(X_1 = X_t) / 2
  W_SIS_RR <- mean(W_t)

  t <- 2
  while (t <= n) {
    X_t <- rpair(X_t, t, N)$X_t
    U_t <-
      pnorm(
        q = y[t],
        mean = A * cos(f * t + X_t),
        sd = sqrt(sigma2_eps)
      )
    W_t <- W_t * U_t
    W_t.bar <- mean(W_t)
    k_t <- floor(W_t / W_t.bar)
    N_r_t <- N - sum(k_t)
```

```
    # The resampling probability
    p.resample_t <- (W_t / W_t.bar - k_t) / N_r_t

    # The index of sample corresponding to its
    # resample number for each X_t
    index.all_t <- c()
    for (k in 1:max(k_t)) {
      index.all_t <- c(index.all_t, which(k_t >= k))
      k <- k + 1
    }
    # Ensure the exception of N_r_t equals to 0 being considered
    if (N_r_t > 0) {
      index.resample_t <-
        sample(
          x = 1:N,
          size = N_r_t,
          replace = TRUE,
          prob = p.resample_t
        )
      index.all_t <- c(index.all_t, index.resample_t)
    }

    # Record X and W for each t
    X_SIS_RR <- rbind(X_SIS_RR, X_t)[, index.all_t]
    W_SIS_RR <- c(W_SIS_RR, W_t.bar)
    W_t <- rep(W_t.bar, N)
    t <- t + 1
  }
  out <- list(X = X_SIS_RR, W = W_SIS_RR)
  return(out)
}
SIS_RR <- residual_resampling(y)
```

(5) Since residual resampling has adjusted the weight of each stream to be the same, the posterior mean estimate of X_i is

$$X_i^* = \frac{1}{N}\sum_{j=1}^{N} X_i^{(j)}.$$

```
X.posterior <- rowMeans(SIS_RR$X)
```

(6) The standard error of each X_t is defined as

$$SE(X_t) = \sqrt{\frac{Var(X_t)}{N}}$$

```
X.SE <- sqrt((rowMeans(SIS_RR$X ^ 2) - rowMeans(SIS_RR$X) ^ 2) / N)
```

(7) To repeat the process for $M = 400$ times independently, it's not efficient to use the normal for loop. Hereby uses the paralleled computing package FOREACH and DOPARALLEL to improve the computing performance.

```
library(foreach)
library(doParallel)
M <- 400
clnum <- detectCores(logical = FALSE)
cl <- makeCluster(mc <- getOption("cl.cores", clnum))
registerDoParallel(cl)

SE <- function(y) {
  SIS_RR <- residual_resampling(y)
  out <- sqrt((rowMeans(SIS_RR$X ^ 2) - rowMeans(SIS_RR$X) ^ 2) / N)
  return(out)
}

SE.M <-
  foreach(exponent = 1:M, .combine = rbind)
  %dopar% SE(y)

stopImplicitCluster()

## Error:  <text>:16:3:  unexpected SPECIAL
## 15:    foreach(exponent = 1:M, .combine = rbind)
## 16:    %dopar%
##        ^
```

Regretfully the overleaf has some issues with the compile process and couldn't compile the the pipe operator '%' after words. However the paralleled experiment is a successful try and the standard error are likely the same. The original codes are given separately for reproduction.