统计计算第三次作业

吕坤升, 15338140, 统计学

May 6, 2018

Contents

1 Please write down the form of Monte Carlo average of log-likelihood (which you are going to evaluate).

Complete-data likelihood function

$$
\begin{aligned}
L(\Omega|Y_{ij},U_i,Z_{1,i},Z_{2,i}) &= \prod_{i=1}^{n}\prod_{c=1}^{2}\left\{\pi_c f_c(Z_{c,i})[\prod_{j=1}^{T} f_c(Y_{ij}|Z_{c,i})]\right\}^{\omega_{ic}} \\
&= exp\left\{\sum_{i=1}^{n}\sum_{c=1}^{2}\omega_{ic}\Big[ln\pi_c - ln(\sqrt{2\pi}\sigma_c) - \frac{Z_{c,i}^2}{2\sigma_c^2} + \sum_{j=1}^{T}[Y_{ij}lnP_{ij}^{(c)} + (1-Y_{ij})ln(1-P_{ij}^{(c)})]\Big]\right\}
\end{aligned}
$$

Complete-data loglikelihood function

$$l=\sum_{i=1}^{n}\sum_{c=1}^{2}\omega_{ic}\Big[ln\pi_c-ln(\sqrt{2\pi}\sigma_c)-\frac{Z_{c,i}^2}{2\sigma_c^2}+\sum_{j=1}^{T}[Y_{ij}lnP_{ij}^{(c)}+(1-Y_{ij})ln(1-P_{ij}^{(c)})]\Big]$$

l 中的 Z_1,Z_2,U 为 missing data，在后面的代码中不再单独分开 $Z1$ 和 $Z2$，可以看成 $Z=I_{U=1}Z_1+I_{U=2}Z2$。设置 $\Omega=(\beta_1,\beta_2,\sigma_1,\sigma_2,\pi_1)$

这里用 MCEM 方法要优化的蒙特卡洛 log 似然函数均值为:

$$Q(\Omega|\Omega^m)=E(l|Y_{ij},\Omega^{(m)})=\frac{1}{N}\sum_{k=1}^{N}\sum_{i=1}^{n}\sum_{c=1}^{2}\omega_{ic}\Big[ln\pi_c-ln(\sqrt{2\pi}\sigma_c)-\frac{Z_{i,k}^2}{2\sigma_c^2}+\sum_{j=1}^{T}[Y_{ij}lnP_{ij}^{(c)}+(1-Y_{ij})ln(1-P_{ij}^{(c)})]\Big].$$

其中的 $Z_{i,k}$ 和对应的 U_i 是从条件分布 $p(U,Z|Y_{ij},\Omega^{(m)})$ 中抽取的。

2 Please write down details of the EM algorithm you use for this simulation, especially the Metropolis-Hastings steps.

为了从条件分布 $p(U,Z|Y_{ij},\Omega^{(m)})$ 中抽取 missing data U 和 Z，注意到这里的 U 和 Z 都是 100 维 (n 维) 的，且每一维对应的 $X_{c,ij},Y_{ij}$ 都不一样，因此对于 U 和 Z 的每一维都需要单独的抽样。对于每一维，我们先给定初值 $U^{(0)}$ 和 $Z^{(0)}$，利用 $P(U^{(k+1)}|Y_{ij},Z^{(k)},\Omega^{(m)})$ 确定 U 的第 k+1 维，再利用 $P(Z^{(k+1)}|Y_{ij},U^{(k+1)},\Omega)$ 来确定 Z 的第 k+1 维。在这里注意，U 的条件分布可以写出显性表达式且容易得到取 1 或 2 的概率，可以直接算出概率利用伯努利分布来直接抽样，但 Z 的条件分布十分难求，因此对 Z 的每一维都需要用到 Metropolis-Hasting method。

具体的 EM 算法如下:

E-Step

第 m 步时

Gibbs Sampling

在这里我们需要从条件分布 $p(U,Z|Y_{ij},\Omega^{(m)})$ 中抽取 N_{Gibbs} 份 Z 和 U。

对于每一份的抽取:

给定第 k-1 维的 Z($Z^{(k-1)}$)

- 1) 计算

$$p_1=\pi_1^{(m)}\frac{1}{\sqrt{2\pi}\sigma_1^{(m)}}e^{-\frac{(Z^{(k-1)})^2}{2(\sigma_1^{(m)})^2}}\prod_{j=1}^{T}(P_{ij}^{(1)})^{Y_{ij}}(1-P_{ij}^{(1)})^{1-Y_{ij}}.$$

- 2) 计算

$$p_2=\pi_2^{(m)}\frac{1}{\sqrt{2\pi}\sigma_2^{(m)}}e^{-\frac{(Z^{(k-1)})^2}{2(\sigma_2^{(m)})^2}}\prod_{j=1}^{T}(P_{ij}^{(2)})^{Y_{ij}}(1-P_{ij}^{(2)})^{1-Y_{ij}}.$$

  其中 $P_{ij}^{(c)}=g^{-1}(\beta_c X_{c,ij}+Z^{(k-1)})$.

- 3) 因为 $P(U^{(k+1)}=1|Y_{ij},Z^{(k)},\Omega^{(m)})$ 和 $P(U^{(k+1)}=2|Y_{ij},Z^{(k)},\Omega^{(m)})$ 的分母相同，因此只需计算分子 (p_1&p_2), 找到归一化参数，则可以得到 $p=P(U^{(k+1)}=1|Y_{ij},Z^{(k)},\Omega^{(m)})=\frac{p_1}{p_1+p_2}$.

- 4) 用伯努利分布进行抽样，第 k 维的 U($U^{(k)}$) 以 p 为概率取 1，以 1-p 的概率取 2.

给定第 k 维的 U($U^{(k)}$)

Metropolis-Hasting Method

- 1) 给定马氏链的第 t 个元素，记为 Z_t，以 $N(Z_t, 1)$ 的转移概率抽取下一个元素 y
- 2)

  若 $U^{(k)}$ 为 1，则计算

$$p_{up} = \frac{1}{\sqrt{2\pi}\sigma_1^{(m)}} e^{-\frac{y^2}{2(\sigma_1^{(m)})^2}} \prod_{j=1}^{T} (P_{ij}^{(1)})^{Y_{ij}} (1 - P_{ij}^{(1)})^{1-Y_{ij}},$$

$$p_{under} = \frac{1}{\sqrt{2\pi}\sigma_1^{(m)}} e^{-\frac{Z_t^2}{2(\sigma_1^{(m)})^2}} \prod_{j=1}^{T} (P_{ij}^{(1)})^{Y_{ij}} (1 - P_{ij}^{(1)})^{1-Y_{ij}},$$

  若 $U^{(k)}$ 为 2，则计算

$$p_{up} = \frac{1}{\sqrt{2\pi}\sigma_2^{(m)}} e^{-\frac{y^2}{2(\sigma_2^{(m)})^2}} \prod_{j=1}^{T} (P_{ij}^{(2)})^{Y_{ij}} (1 - P_{ij}^{(2)})^{1-Y_{ij}},$$

$$p_{under} = \frac{1}{\sqrt{2\pi}\sigma_2^{(m)}} e^{-\frac{Z_t^2}{2(\sigma_2^{(m)})^2}} \prod_{j=1}^{T} (P_{ij}^{(2)})^{Y_{ij}} (1 - P_{ij}^{(2)})^{1-Y_{ij}}.$$

- 3) 计算

$$p = \frac{p_{up}}{p_{under}}.$$

- 4) 抽取 $rand \sim U(0, 1)$
- 5)

  若 $rand <= p$, 马氏链的第 t+1 个元素 $Z_{t+1} = y$

  若 $rand > p$, 马氏链的第 t+1 个元素 $Z_{t+1} = Z_t$

- 6) 重复步骤 1-5，1000 次，得到一条 1000 个元素的马氏链，丢弃前 900 个元素，在最后 100 个元素中随机抽取 1 个元素作为第 k 维的 Z($Z^{(k)}$)

重复上述算法 n 次，可得到一份满足要求的 U 和 Z。再重复 N_{Gibbs} 次，则可以得到进行 MCEM 所需的 N_{Gibbs} 份随机样本。丢弃掉前 100 份随机样本，作为 burn-in，将剩余的随机样本代入式子，计算期望值:

$$Q = E(l|\Omega^{(m)}) = \frac{1}{N} \sum_{k=1}^{N} \sum_{i=1}^{n} \sum_{c=1}^{2} \omega_{ic} \Big[ln\pi_c - ln(\sqrt{2\pi}\sigma_c) - \frac{Z_{i,k}^2}{2\sigma_c^2} + \sum_{j=1}^{T} [Y_{ij} ln P_{ij}^{(c)} + (1 - Y_{ij}) ln(1 - P_{ij}^{(c)})] \Big]$$

此时，这里的 $Z_{i,k}$ 和对应的 U_i，是指经历上述算法后得到的。这里的 $N = N_{Gibbs} - N_{burn-in}$

M-Step

运用拟牛顿法 (Quasi-Newton Methods) 最大化上面好不容易得到的 Q，更新新一轮的参数。验证是否收敛，不收敛则返回到上一步。

Trick

- 在这里，$N = N_Gibbs - N_{burn-in}$ 随迭代的次数逐渐增加，在代码中设置为第 m 次迭代的时候，$N = 5 * m$。
- 设置 T=30，可以更快的达到收敛。

3 What are your initial values? What is your convergence rule?

初值设置为

$$\beta_1 = 0.5, \beta_2 = 1.5, \sigma_1 = 2.5, \sigma_2 = 11, \pi_1 = 0.8,$$

在 100 次 simulation 中，设置当第 m 次迭代更新产生的参数与上一次产生的参数差距小于 0.1 定为收敛，在一次 simulation 中，设置差距小于 0.01 才收敛，但在跑最大次数前仍然无法收敛，但是从 Figure 1 中可看出来已经是收敛了。

4 How to accelerate your EM algorithm? Any improvement you can observe?

- 拟牛顿法因为不需要二阶导数的信息，比普通的牛顿法更加有效。在这里使用了拟牛顿法。
- 在 R 中使用 sapply 而不是 for 作为循环，然后可使用 snowfall 等 Rpackage 进行并行。
- 由于 R 中反复调用函数会降低速度，且算法中包含了大量的循环，因此将函数改成内联的形式可以大大加快代码运行的速度。使用这个方法几乎比原来耗的时间少了一半。
- 在 M 步中，一些参数可以直接写出表达式直接求导最优化，可以减少利用拟牛顿法优化的参数个数，提高速度。

5 Try different numbers of simulations: 200,300,...,1000. And plot the corresponding MSE.

取了一次 simulation 的结果，Figure 1 画出每个参数的 update path，可以看到除了 σ_2 的估计较为不准外，其他参数与参数差距较小。

由于 simulation 跑的十分的慢，即使放松了收敛的条件，在作业截止时间前最多跑了 17 次，将每一次的 MSE 依次数画成散点图，如 Figure 2，可以观察到每一次的 MSE 保持大致的稳定。

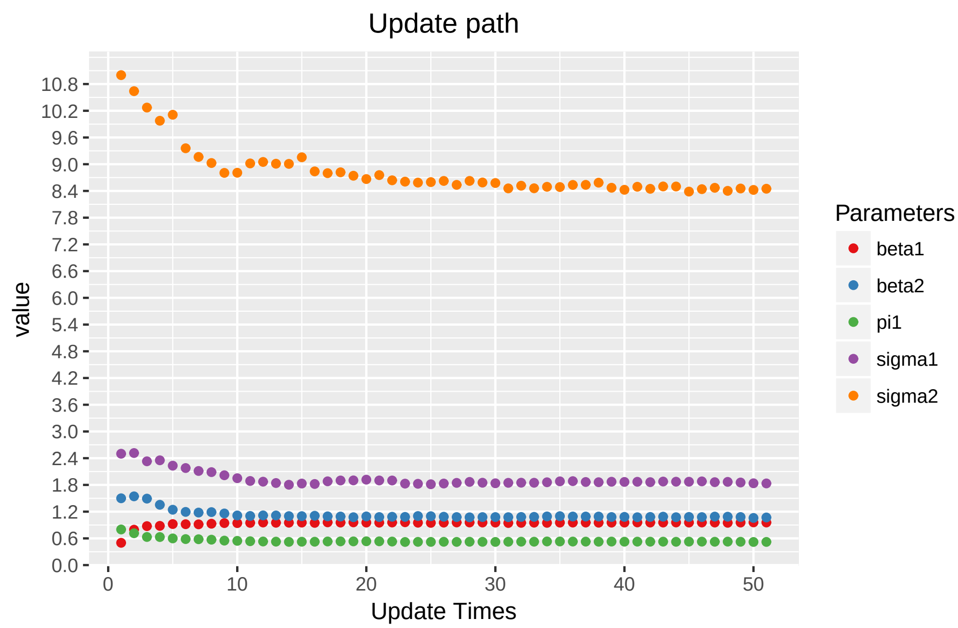

Figure 1: Update path of five parameters for single simulation.

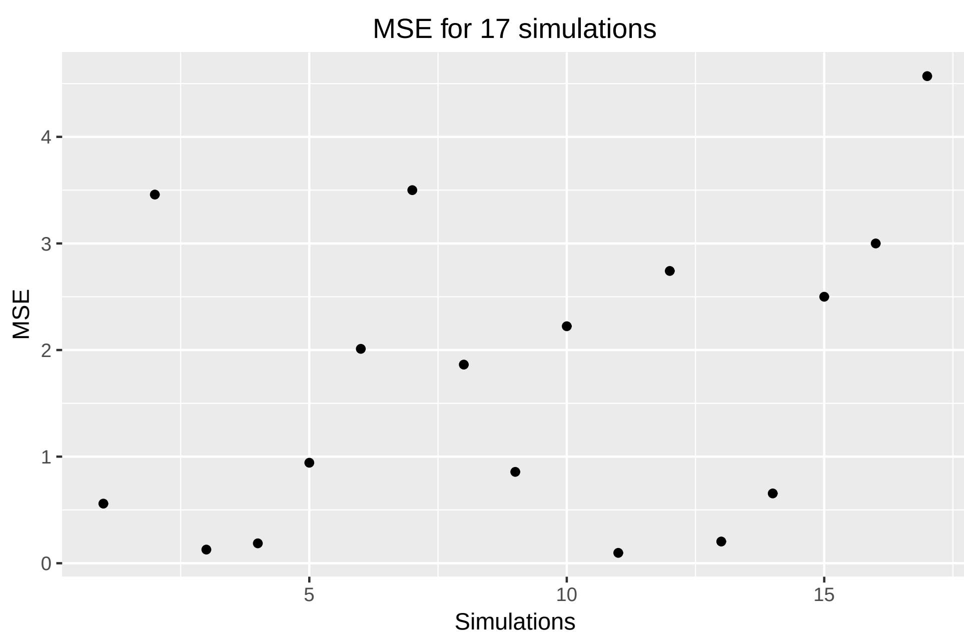

Figure 2: MSE for 17 simulations.

6 Code

6.1 functions

```
rm(list = ls())
setwd("C:/Users/Administrator/Desktop/lvksh/统计计算")
g <- function(x){
  return(exp(x)/(1+exp(x)))
}
logLikelihood <- function(Z,
                          U,
                          Omega,
                          m = 1) {
  # 该函数传入100维的Z和U，计算loglikelihood
  wi1 <- which(U == 1)
  wi2 <- which(U == 2)

  l1 <- sapply(wi1,function(i){
    TT <- sapply(1:Time,function(j){
      eta <- Omega[1] * X1[i,j] + Z[i]
      if(eta >= 600) eta <- 600 # 限制可能出现的超大数
      TTT <- Y_ij[i,j] * (eta - log(1 + exp(eta))) + (1 - Y_ij[i,j]) *
          (- log(1 + exp(eta)))
      return(TTT)
    })
    lll <- (log(Omega[5]) - log(sqrt(2*pi) * Omega[3]) - Z[i]^2/(2*
        Omega[3]^2) + sum(TT))
    return(lll)
  })
  l1 <- sum(l1)

  l2 <- sapply(wi2,function(i){
    TT <- sapply(1:Time,function(j){
      eta <- Omega[2] * X2[i,j] + Z[i]

      if(eta >= 600) eta <- 600
      TTT <- Y_ij[i,j] * (eta - log(1 + exp(eta))) + (1 - Y_ij[i,j]) *
          (- log(1 + exp(eta)))
      return(TTT)
    })
    lll <- (log(1-Omega[5]) - log(sqrt(2*pi) * Omega[4]) - Z[i]^2/(2*
        Omega[4]^2) + sum(TT))
    return(lll)
  })
  l2 <- sum(l2)

  L <- l1 + l2
  return(L)
}


```

```
Gibbs_Sampling <- function(Z, # 给定初值
                            Omega,
                            m ){ # EM算法的第m步
  # 返回Gibbs抽样得到的Z，U
  # 建立存放Gibbs抽样的结果矩阵
  Z_ <- list(Z = array(dim = c(n,Gibbs_N+burn_in+1)), U = array(dim = c(n,Gibbs_N+burn_in+1)))
  # 给定初值
  Z_$Z[,1] <- Z$Z
  Z_$U[,1] <- Z$U
  # 开始交错更新
  for (k in 2:(Gibbs_N+burn_in+1))  {
      for (i in 1:n){
        # 在第k份的第i维 先生成第k份的U的第i维，再用这个用MH去抽第k份的Z的第i维
        U_Z <- Z_$Z[i,k-1]
        U_U <- Z_$U[i,k-1]
        p1 <- prod(g(Omega[1] * X1[i,] + U_Z)^Y_ij[i,] * (1 - g(Omega[1] * X1[i,] + U_Z))^(1 - Y_ij[i,])) * Omega[5] * dnorm(x = U_Z,mean = 0,sd = Omega[3])
        p2 <- prod( g(Omega[2] * X2[i,] + U_Z)^Y_ij[i,]  * (1 - g(Omega[2] * X2[i,] + U_Z))^(1 - Y_ij[i,])) * (1 - Omega[5]) * dnorm(x = U_Z,mean = 0,sd = Omega[4])
        prod <- p1 / (p1 + p2)
        if(is.nan(prod)) prod = 1
        Z_$U[i,k] <- rbinom(n = 1,size = 1,prob = 1 - prod) + 1

        # 用上面抽到的U的第i维，来抽Z的第i维，这里需要用到MH
        Z_U <- Z_$U[i,k]
        #Z_U <- Z_$Z[i,k-1]
        # 建立MH算法的马氏链
        MH_Z <- array(dim = c(1,Metro_N+1))
        MH_Z_initial <- rnorm(n = 1,mean = 0,sd = 1)
        MH_Z[1] <- MH_Z_initial # 链的第一个元素为初值
        for(t in 2:(Metro_N+1)){
          temp <- rnorm(n = 1,mean = MH_Z[t-1],sd = 1) # 以转移概率得到一个数
          if (Z_U == 1){
            p_up <- prod( (g(Omega[1] * X1[i,] + temp))^Y_ij[i,]  * (1 - g(Omega[1] * X1[i,] + temp))^(1 - Y_ij[i,])) * dnorm(x = temp,mean = 0,sd = Omega[3])
            p_under <- prod( (g(Omega[1] * X1[i,] + MH_Z[t-1]))^Y_ij[i,] * (1 - g(Omega[1] * X1[i,] + MH_Z[t-1]))^(1 - Y_ij[i,])) * dnorm(x = MH_Z[t-1],mean = 0,sd = Omega[3])
            Reject_p <- p_up / p_under
          } else if(Z_U == 2) {
            p_up <- prod( g(Omega[2] * X2[i,] + temp)^Y_ij[i,]  * (1 - g(Omega[2] * X2[i,] + temp))^(1 - Y_ij[i,])) * dnorm(x =
```

```
                    temp,mean = 0,sd = Omega[4])
            p_under <- prod( g(Omega[2] * X2[i,] + MH_Z[t-1])^Y_ij[i,]
                * (1 - g(Omega[2] * X2[i,] + MH_Z[t-1]))^(1 - Y_ij[i,])
                ) * dnorm(x = MH_Z[t-1],mean = 0,sd = Omega[4])
            Reject_p <- p_up / p_under
          }
          rand <- runif(1)
          if(Reject_p == Inf) Reject_p <- 0.5
          if(rand <= Reject_p) MH_Z[t] <- temp
          else MH_Z[t] <- MH_Z[t-1]
        }
        Z_$Z[i,k] <- sample(x = MH_Z[(Metro_N-100):(Metro_N + 1)],size
            = 1)
        # 在马氏链里的最后100个数里抽一个作为最终的第k份的Z的第i维
      }
  }
  return(Z_)
}

fn <- function(Omega){
  ff <- sapply(2:(Gibbs_N + 1),function(k){
    temp <- logLikelihood(Z = Gibbs_result$Z[,burn_in +k],U = Gibbs_
        result$U[,burn_in +k],Omega,m)
    return(temp)
  })
  return(-mean(ff)) # 这里是负的 为了optim函数
}
```

6.2 Main function for 100 simulations

```
burn_in <- 100
# initial value
    ____________________________________________________________
# Gibbs_N <- 100 #要求Gibbs的次数
reps <- 50 # 最大循环次数
Metro_N <- 1000 # 马氏链的长度
# 第一轮的参数初值
b1 <- 0.5
b2 <- 1.5
s1 <- 2.5
s2 <- 11
PI <- 0.8
Omega <- c(b1=b1,b2=b2,s1=s1,s2=s2,PI=PI)


OMEGA <- array(dim = c(5,100))
for(simulations in 1:100){
  set.seed(simulations)
  n <- 100
  Time <- 30
```

```
  beta1 <- 1
  beta2 <- 1
  pi1 <- 0.6
  sigma1 <- 2
  sigma2 <- 10
  U_i <- rbinom(n = n,size = 1,prob = 1 - pi1) + 1
  X1 <- matrix(rnorm(n = n*Time),nrow = n,ncol = Time) # fixed effect
  X2 <- matrix(rnorm(n = n*Time),nrow = n,ncol = Time)
  z1 <- rnorm(n = n,mean = 0,sd = sigma1) # random effect
  z2 <- rnorm(n = n,mean = 0,sd = sigma2)
  Y_ij <- array(dim = c(n,Time))
  for(i in 1:n) {
    for(j in 1:Time) {
      # generate the Y_ij AKA observed data
      # the group item indicates the cluster for each subject
      if(U_i[i] == 1) {
        Y_ij[i,j] <- g(beta1 * X1[i,j] + z1[i])
      }
      else {
        Y_ij[i,j] <- g(beta2 * X2[i,j] + z2[i])
      }
      Y_ij[i,j] <- rbinom(n = 1,size = 1,prob = Y_ij[i,j])
    }
  }

  for (m in 1:reps)  {
    Gibbs_N <- 10 * m
    U_initial <- rbinom(n = n,size = 1,prob = Omega[5]) + 1
    Z_initial <- rnorm(n = n,mean = 0,sd = (Omega[3] + Omega[4])/2)
    Z_list <- list(Z=Z_initial,U=U_initial)
    Gibbs_result <- Gibbs_Sampling(Z_list,Omega,m)
    update_par <- optim(par = c(0.9,1.1,2.5,11,0.7),fn =  fn,method = "BFGS",control = list(trace = 2))
    update_par <- update_par$par
    if(all(abs(update_par - Omega) <= 1e-01)) {
      break
    }
    Omega <- update_par
  }
    OMEGA[,simulations] <- update_par
}
save(OMEGA,file = "OMEGA_100.rdata")
```

6.3 Main function for one simulation

```
set.seed(1)
n <- 100
Time <- 10
beta1 <- 1
beta2 <- 1
```

```
pi1 <- 0.6
sigma1 <- 2
sigma2 <- 10
U_i <- rbinom(n = n,size = 1,prob = 1 - pi1) + 1
X1 <- matrix(rnorm(n = n*Time),nrow = n,ncol = Time) # fixed effect
X2 <- matrix(rnorm(n = n*Time),nrow = n,ncol = Time)
z1 <- rnorm(n = n,mean = 0,sd = sigma1) # random effect
z2 <- rnorm(n = n,mean = 0,sd = sigma2)
Y_ij <- array(dim = c(n,Time))
for(i in 1:n) {
  for(j in 1:Time) {
    # generate the Y_ij AKA observed data
    # the group item indicates the cluster for each subject
    if(U_i[i] == 1) {
      Y_ij[i,j] <- g(beta1 * X1[i,j] + z1[i])
    }
    else {
      Y_ij[i,j] <- g(beta2 * X2[i,j] + z2[i])
    }
    Y_ij[i,j] <- rbinom(n = 1,size = 1,prob = Y_ij[i,j])
  }
}
burn_in <- 100
# initial value
# Gibbs_N <- 100 #要求Gibbs的次数
reps <- 50 # 最大循环次数
Metro_N <- 1000 # 马氏链的长度
# 第一轮的参数初值
b1 <- 0.5
b2 <- 1.5
s1 <- 2.5
s2 <- 11
PI <- 0.8
Omega <- c(b1=b1,b2=b2,s1=s1,s2=s2,PI=PI)
OMEGA <- array(dim = c(5,reps))
for (m in 1:reps)  {
  time.start <- Sys.time()
  Gibbs_N <- 4 * m
  U_initial <- rbinom(n = n,size = 1,prob = Omega[5]) + 1
  Z_initial <- rnorm(n = n,mean = 0,sd = (Omega[3] + Omega[4])/2)
  Z_list <- list(Z=Z_initial,U=U_initial)
  Gibbs_result <- Gibbs_Sampling(Z_list,Omega,m)
  update_par <- optim(par = c(0.9,1.1,2.5,9,0.7),fn =  fn,method = "BFGS",control = list(trace = 2))
  update_par <- update_par$par
  if(all(abs(update_par - Omega) <= 1e-02)) {
    cat("终于收敛啦！！！参数依次是:␣",Omega)
    break
  }
  OMEGA[,m] <- update_par
```

```r
  Omega <- update_par
  print(Omega)
  time.end <- Sys.time()
  print(time.end - time.start)
}
save(Omega,file = "Omega_once.Rdata")
```

6.4 Code for plot

```r
library(ggplot2)
library(reshape2)
Omega_once <- Omega_once[,-((which(is.na(Omega_once[1,]))[1]):101)]
OMEGA_100 <- OMEGA_100[,-((which(is.na(OMEGA_100[1,]))[1]):100)]
n1 = ncol(Omega_once)
n2 = ncol(OMEGA_100)
O_once <- melt(t(Omega_once))
O_once$Var2 <- rep(c("beta1","beta2","sigma1","sigma2",'pi1'),each = n1
  )
ggplot(data = O_once,aes(x=rep(1:n1,5),y=value,color=Var2)) +
  scale_colour_brewer(palette = 'Set1') +
  geom_point() +
  theme(plot.title=element_text(hjust=0.5)) +
  labs(color = "Parameters") +
  labs(title = "Update path") +
  xlab(label = "Update Times")

MSE <- apply(OMEGA_100,MARGIN = 2,FUN = function(e){
  sum((e - c(1,1,2,10,0.6))^2)
})
ggplot(data = data.frame(MSE),mapping = aes(x = 1:17,y = MSE)) +
  geom_point() +
  labs(title = "MSE for 17 simulations") +
  xlab(label = "Simulations") +
  theme(plot.title=element_text(hjust=0.5))
```

6.5 Worth noting in code

- 在计算 loglikelihood 的时候，由于 g^{-1} 函数中有指数函数，且 Z_2 的方差较大，很容易出现数字过大而被强行设为 Inf，而报错的结果。解决方案是设一个上界，例如在这里我设的上界是 e^{600}，若超过这个上界则令其等于该上界。可能也是因为这个原因导致了 $sigma_2$ 估计不准。
- 而同样的问题也出现在 Gibbs 抽样时计算接受概率的联乘的地方，而且即使每一项都相对比较小，例如说 e^{20}，而联乘之后也会变成 Inf，解决方案是若出现这种情况，则直接取接受概率为 1。这里也尝试过取 0 或者取 0.5，但是效果都不好。
- 在抽 U 时，计算 p_1, p_2 时也是同样的问题。
- 在 rbinom 函数中，设置 size=1 后，prob 参数指的是在 0 和 1 中取 1 的概率，但是题目中的 U 是取值 1 或 2 的，一开始我直接使用 rbinom + 1 的方式取 U，但是要注意这里的 prob 参数应该设置为 $1-\pi_1$ 才是 U 取 1 的概率。