```
x_prior = function(x,i,j) {
  nrows = dim(x)[1]
  ncolumns = dim(x)[2]
  sum_1 = indicator_x(x,i,j)
  return(2*sum_1)
}

indicator_x = function(x,i,j) {
  nrows = dim(x)[1]
  ncolumns = dim(x)[2]
  indicator_diff = 0
  if (i > 1) {
    left_0 = x[i,j] == x[i-1,j]
    left_1 = (1-x[i,j]) == x[i-1,j]
    indicator_diff = indicator_diff + (left_1 - left_0)
  }
  if (i < nrows) {
    right_0 = x[i,j] == x[i+1,j]
    right_1 = (1-x[i,j]) == x[i+1,j]
    indicator_diff = indicator_diff + (right_1 - right_0)
  }
  if (j > 1) {
    up_0 = x[i,j] == x[i,j-1]
    up_1 = (1-x[i,j]) == x[i,j-1]
    indicator_diff = indicator_diff + (up_1 - up_0)
  }
  if (j < ncolumns) {
    down_0 = x[i,j] == x[i,j+1]
    down_1 = (1-x[i,j]) == x[i,j+1]
    indicator_diff = indicator_diff + (down_1 - down_0)
  }
  return(indicator_diff)
}
```

```
x_update <- function(times, x, mu_0, mu_1, sigma_0, sigma_1, beta){
  for (it in 1:times) {
    i = ceiling(nrows*runif(1))
    j = ceiling(ncolumns*runif(1))
    x_prop = 1 - x[i,j]
    x_new = x
    x_new[i,j] = x_prop
    I = x_prior(x,i,j)
    ising = beta*I
    if (x[i,j] == 0) {
      #normal = sigma_0/sigma_1 * exp(-1/(2*sigma_1^2)*(y[i,j]-mu_1)^2 + 1/(2*sigma_0)*(y[i,j]-mu_0)^2)
      normal = log(sigma_0) - log(sigma_1) - 1/(2*sigma_1^2)*(y[i,j]-mu_1)^2 + 1/(2*sigma_0)*(y[i,j]-mu
    }
    if(x[i,j] == 1){
      #normal = sigma_1/sigma_0 * exp(-1/(2*sigma_0^2)*(y[i,j]-mu_0)^2 + 1/(2*sigma_1)*(y[i,j]-mu_1)^2)
      normal = log(sigma_1) - log(sigma_0) -1/(2*sigma_0^2)*(y[i,j]-mu_0)^2 + 1/(2*sigma_1)*(y[i,j]-mu_
    }
    fratio = normal + ising
    #print(fratio)
    alpha = min(0,fratio)
```

```
    u = log(runif(1))
    if (u < alpha) {
      x[i,j] = x_prop
    }
  }
  return(x)
}

## Acceptance probability

# test1 <- muone(N_1, ysum1, sigma_1, mu1_old)
# test1

muzero = function(N_0, ysum0, sigma_0, mu_0) {
  mu_0_prop <- rnorm(1, ysum0/N_0, sqrt((sigma_0^2)/N_0))
  if (mu_0_prop )
  qfratio = 3*(log(mu_1-mu_0_prop)-log(mu_1-mu_0_prop)) - (mu_0 - mu_0_prop)/sqrt(sigma_0*sigma_1)
  alpha = min(0,qfratio)
  u = log(runif(1))
  if (u < alpha) {
    mu_0 = mu_0_prop
  }
  return(mu_0)
}

muone = function(N_1, ysum1, sigma_1, mu_1) {
  mu_1_prop <- rnorm(1, ysum1/N_1, sqrt((sigma_1^2)/N_1))
  if (mu_1_prop < mu_0) {
    return(mu_1)
  }
  else {
    qfratio = 3*(log(mu_1_prop-mu_0) - log(mu_1-mu_0)) - (mu_1_prop - mu_1)/sqrt(sigma_0*sigma_1)
    alpha = min(0,qfratio)
    u = log(runif(1))
    if (u < alpha) {
      mu_1 = mu_1_prop
    }
    return(mu_1)
  }
}

# test0 <- muzero(N_0, ysum0, sigma_0, mu0_old)
# test0
```

```
#Acceptance probability for sigma_0
# a_0 = 2.5
# b_0 = 6

sigmazero <- function(a_0, b_0, N_0, mu_0, mu_1, sigma_0, sigma_1){
  sigma_0squared_prop = rinvgamma(1, shape = a_0 + N_0/2, scale = b_0 + sum((y[x==0]-mu_0)^2)/2)
  sigma_0_prop = sqrt(sigma_0squared_prop)
  logomega_0 = (2*a_0 + 1)*(log(sigma_0_prop) - log(sigma_0)) + b_0 * (1/sigma_0squared_prop - 1/(sigma
  if (sigma_0 <= sigma_1 && sigma_0_prop < sigma_1) {
    case_0 = 3*(log(sigma_0)-log(sigma_0_prop)) - sqrt(sigma_1/sigma_0_prop) + sqrt(sigma_1/sigma_0)
```

```
  } else if (sigma_0 <= sigma_1 && sigma_0_prop > sigma_1) {
    case_0 = 3*log(sigma_0) - 2*log(sigma_0_prop) - log(sigma_1) - sqrt(sigma_0_prop/sigma_1) + sqrt(sig
  } else if (sigma_0 > sigma_1 && sigma_0_prop > sigma_1) {
    case_0 = 2*(log(sigma_0)-sigma_0_prop) -sqrt(sigma_0_prop/sigma_1) + sqrt(sigma_0/sigma_1)
  } else if (sigma_0 > sigma_1 && sigma_0_prop < sigma_1) {
    case_0 = 2*log(sigma_0) + log(sigma_1) - 3*log(sigma_0_prop) - sqrt(sigma_1/sigma_0_prop) + sqrt(sig
  }
  qfratio_0 = logomega_0+case_0
  alpha = min(0,qfratio_0)
  u = log(runif(1))
  if (u < alpha) {
    sigma_0 = sigma_0_prop
  }
  return(sigma_0)
}
```

```
#Acceptance probability for sigma_1
# a_1 = 2.5
# b_1 = 6

sigmaone <- function(a_1, b_1, N_1, mu_0, mu_1, sigma_0, sigma_1){
  sigma_1squared_prop = rinvgamma(1, shape = a_1 + N_1/2, scale = b_1 + sum((y[x==1]-mu_1)^2)/2)
  sigma_1_prop = sqrt(sigma_1squared_prop)
  logomega_1 = (2*a_1+1)*(log(sigma_1_prop)-log(sigma_1)) + b_1*(1/sigma_1_prop^2 - 1/sigma_1^2) - (mu_1
  if (sigma_0 <= sigma_1_prop && sigma_0 <= sigma_1) {
    case_1 = 2*(log(sigma_1)-log(sigma_1_prop)) - sqrt(sigma_1_prop/sigma_0) + sqrt(sigma_1/sigma_0)
  } else if (sigma_0 <= sigma_1_prop && sigma_0 > sigma_1) {
    case_1 = 3*log(sigma_1) - log(sigma_0) - 2*log(sigma_1_prop) - sqrt(sigma_1_prop/sigma_0) + sqrt(sig
  } else if (sigma_0 > sigma_1_prop && sigma_0 > sigma_1) {
    case_1 = 3*(log(sigma_1)-log(sigma_1_prop)) - sqrt(sigma_0/sigma_1_prop) + sqrt(sigma_0/sigma_1)
  } else if (sigma_0 > sigma_1_prop && sigma_0 <= sigma_1) {
    case_1 = log(sigma_0) + 2*log(sigma_1) - 3*log(sigma_1_prop) - sqrt(sigma_0/sigma_1_prop) + sqrt(sig
  }
  qfratio_1 = logomega_1+case_1
  alpha = min(0,qfratio_1)
  u = log(runif(1))
  if (u < alpha) {
    sigma_1 = sigma_1_prop
  }
  return(sigma_1)
}
```

```
# Updating x and parameters
numruns <- 100

#Initialize
y = read.table("./image.txt", header = FALSE, sep = " ")
nrows = dim(y)[1]
ncolumns = dim(y)[2]
x = matrix(rbinom(nrows * ncolumns, 1, 0.5), ncol = ncolumns, nrow = nrows)


mu_0 <- -1
sigma_0 <- 0.5
```

```
a_0 <- 2.5
b_0 <- 6

mu_1 <- 2
sigma_1 <- 0.5
a_1 <- 2.5
b_1 <- 6

beta <- 1

# Run algorithm
for (runs in 1:numruns){
  xupdate <- x_update(89*85, x, mu_0, mu_1, sigma_0, sigma_1, beta)

  #res0 <- fcount(xupdate,y,0)
  #N_0 <- res0$count
  N_0 = sum(x==0)
  #ysum0 <- res0$ysum
  ysum0 = sum(y[x==0])
  # Update mu_0
  mu_0 <- muzero(N_0, ysum0, sigma_0, mu_0)

  #res1 <- fcount(xupdate,y,1)
  #N_1 <- res1$count
  N_1 = sum(x==1)
  #ysum1 <- res1$ysum
  ysum1 = sum(y[x==1])
  #Update mu_1
  mu_1 <- muone(N_1, ysum1, sigma_1, mu_1)

  # Update sigma_0, sigma_1
  sigma_0 <- sigmazero(a_0, b_0, N_0, mu_0, mu_1, sigma_0, sigma_1)
  sigma_1 <- sigmaone(a_1, b_1, N_1, mu_0, mu_1, sigma_0, sigma_1)

  # Updates for next iteration
  x <- xupdate
}
```