STA 360: Assignment 6

Michael Lin

March 11, 2015

1. (a) Given the p.d.f.

$$p(x, y) \propto \mathbb{1}(|x - y| < c)\mathbb{1}(x, y \in (0, 1))$$

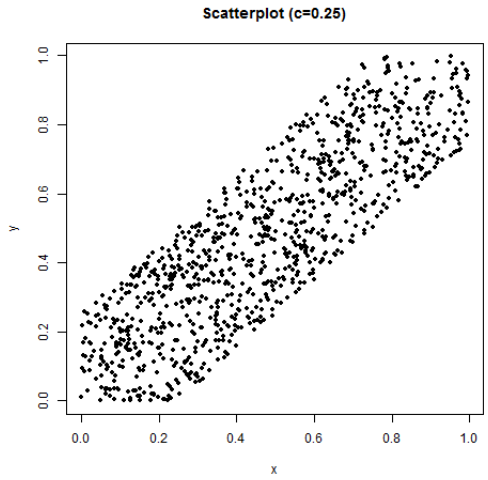
the Gibbs sampler for this distribution is:

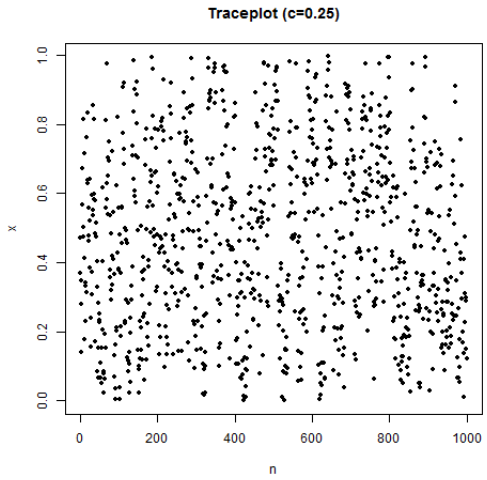
$$p(x|y) \propto \mathbb{1}(|x - y| < c)\mathbb{1}(x \in (0, 1))$$

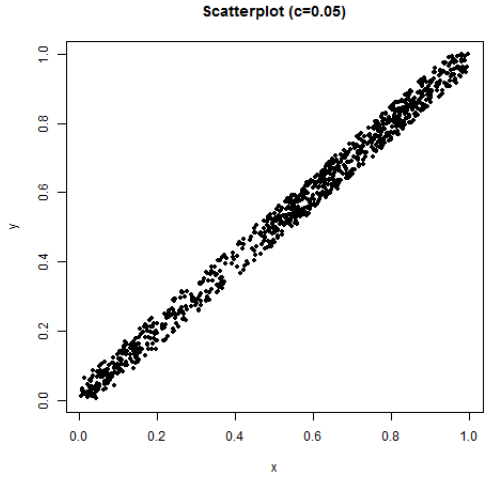
$$p(y|x) \propto \mathbb{1}(|x - y| < c)\mathbb{1}(y \in (0, 1))$$

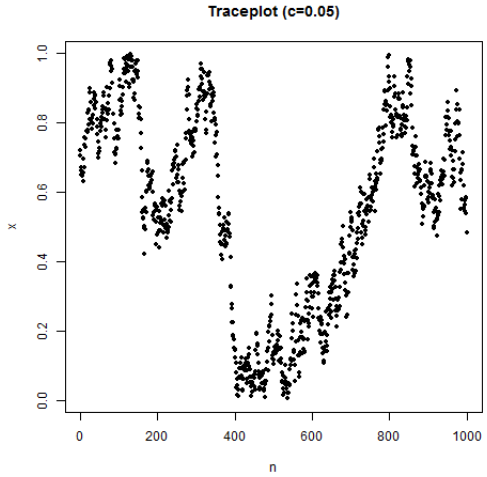
(b) See R code.

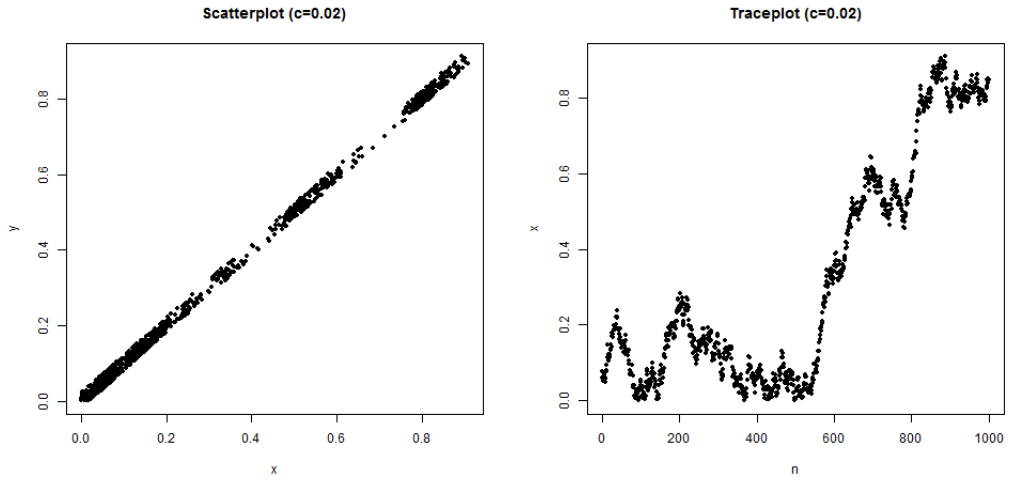
(c) See below for graphs.











(d) The sampler will perform worse and worse as c gets smaller because larger samples are needed to fully cover the probability space. Since c is small, each sweep in the Gibbs sampling algorithm takes only a very small step (maximum of c). As a result, the distance covered by the sampler for 10^3 steps is not sufficient to sample the entire probability space because c is small. For instance, the sampler was only able to sample in a subset of the interval (0,1) for both x and y. One easy way to ensure full coverage of probability space is taking larger number of samples, but this is not the most efficient method.

2. (a) Given the p.d.f.

$$p(u, v) = \mathbb{1}(|v| < c/2)\mathbb{1}(|v| < u < 1 - |v|)$$

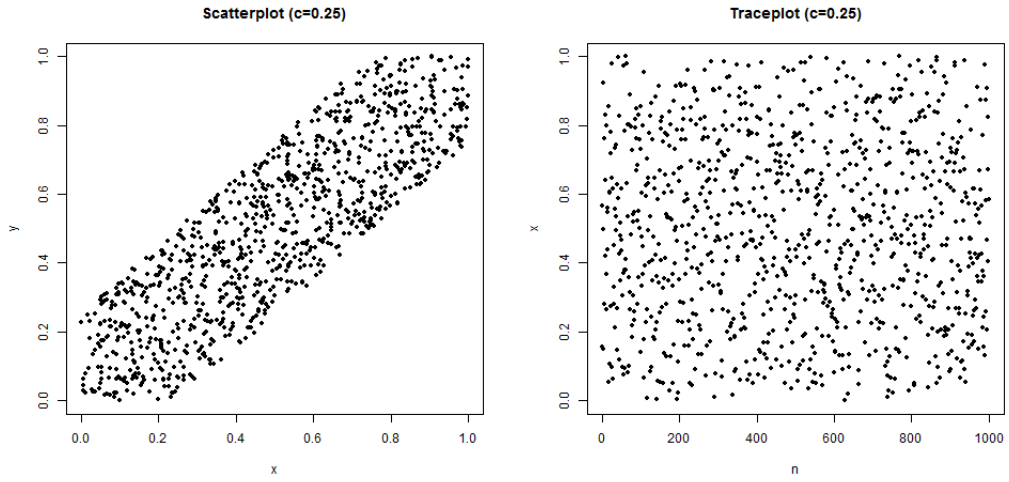
the Gibbs sampler for this distribution is the following:

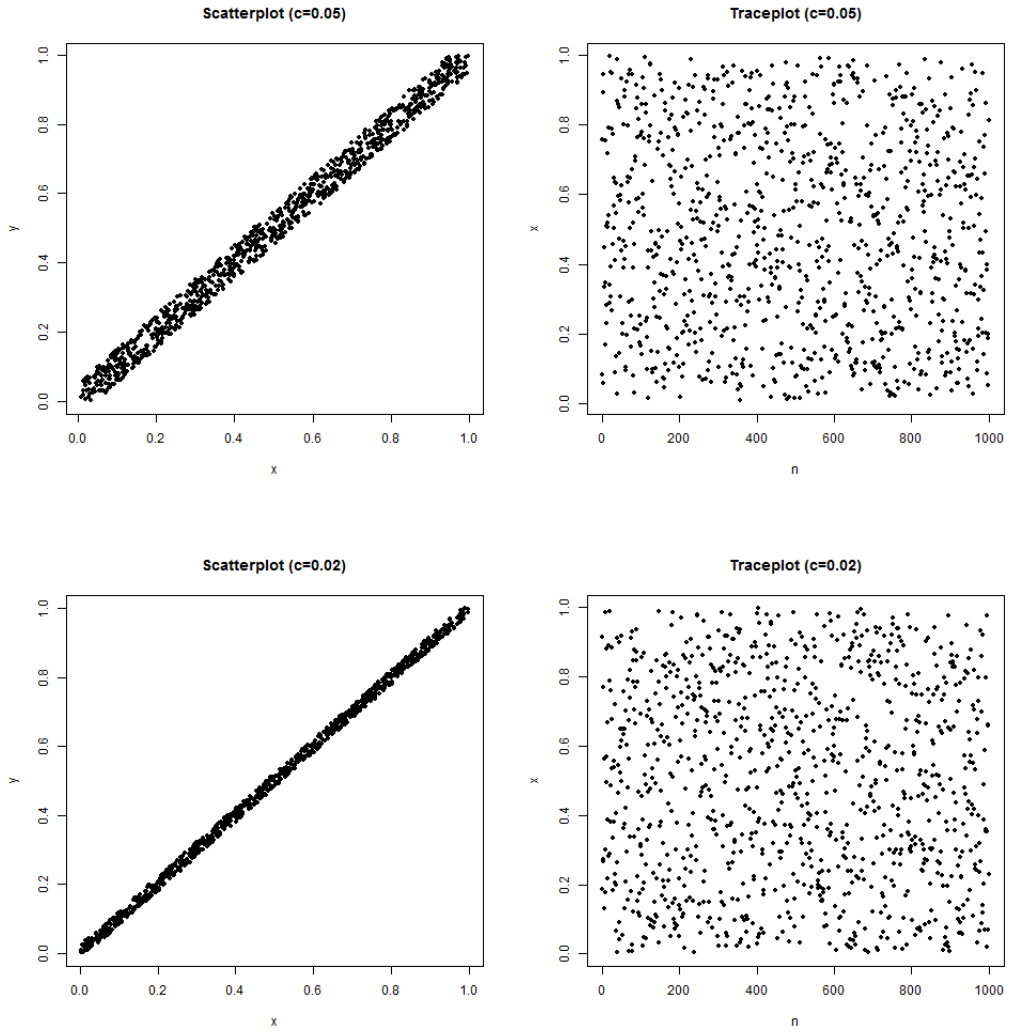
$$p(u|v) = \mathbb{1}(|v| < u < 1 - |v|)$$

$$p(v) = \mathbb{1}(|v| < c/2)$$

(b) See R code.

(c) See below for graphs.





(d) This sampler does not suffer from the issue as the previous one. Since the v's don't have autocorrelation, each step is independent of c, which means the range of each step is not restrained by c. Therefore, the sampler is able to cover the entire probability space.

3. (a) The full conditional $p(\beta|y,x,z,c)$:

$$\begin{aligned}
p(\beta|y,x,z,c) = p(\beta|x,z) &\propto \Big[\prod_{i=1}^{n} p(z_i|\beta,x_i)\Big]p(\beta|x_i) \\
&= \Big[\prod_{i=1}^{n} \text{Normal}(z_i|\beta x_i,1)\Big]\text{Normal}(\beta|0,\tau_\beta^2) \\
&\propto \exp(-\frac{1}{2}\sum(z_i-\beta x_i)^2)\exp(-\frac{\beta^2}{2\tau_\beta^2}) \\
&= \exp(-\frac{1}{2}\sum(z_i^2-2\beta x_i z_i+\beta^2 x_i^2))\exp(-\frac{\beta^2}{2\tau_\beta^2}) \\
&\propto \exp(-\frac{1}{2}(\frac{\beta^2}{\tau_\beta^2}+\beta^2\sum x_i^2-2\beta\sum x_i z_i)) \\
&= \exp(-\frac{1}{2}(\beta^2(\frac{1}{\tau_\beta^2}+\sum x_i^2)-2\beta\sum x_i z_i)) \\
&\propto \exp(\frac{1}{2(\frac{1}{\tau_\beta^2}+\sum x_i^2)^{-1}}(\beta^2-2\beta\frac{\sum x_i z_i}{\frac{1}{\tau_\beta^2}+\sum x_i^2})) \\
&\propto \text{Normal}\Big(\frac{\sum x_i z_i}{\frac{1}{\tau_\beta^2}+\sum x_i^2},(\frac{1}{\tau_\beta^2}+\sum x_i^2)^{-1}\Big)
\end{aligned}$$

(b) The full conditional $p(c|y,x,z,\beta)$:

$$\begin{aligned}
p(c|y,x,z,\beta) &\propto \Big[\prod_{i=1}^{n} p(y_i|c,z_i)\Big]p(c) \\
&= \Big[\prod_{i=1}^{n} \mathbb{1}(z_i>c)^{y_i}\mathbb{1}(z_i<c)^{1-y_i}\Big]\text{Normal}(0,\tau_c^2) \\
&= \mathbb{1}(\max(z_i : z_i<c) < c < \min(z_i : z_i>c))\text{Normal}(0,\tau_c^2)
\end{aligned}$$

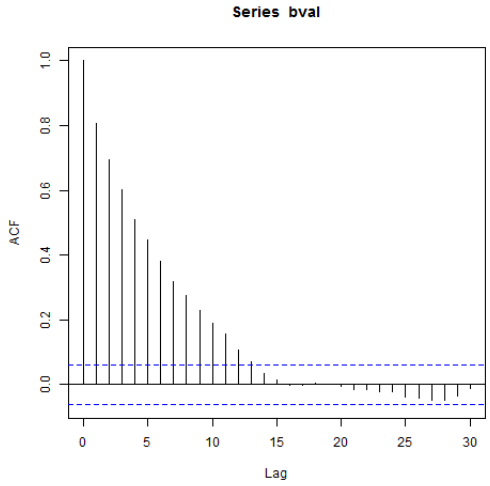
Note this is simply the normal distribution $\text{Normal}(0,\tau_c^2)$ bounded below by $\max(z_i : z_i<c)$ and bounded above by $\min(z_i : z_i>c)$.

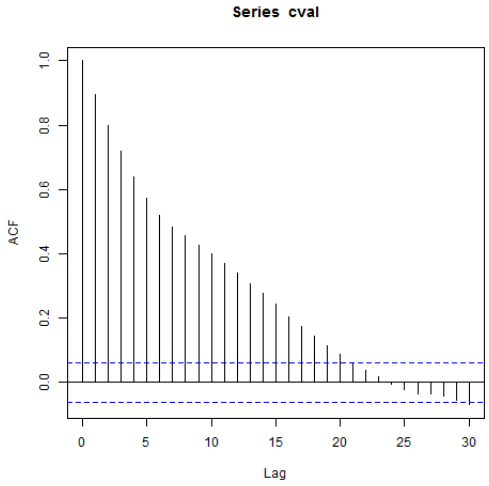
The full conditional $p(z_i|y,x,z_{-i},\beta,c)$:

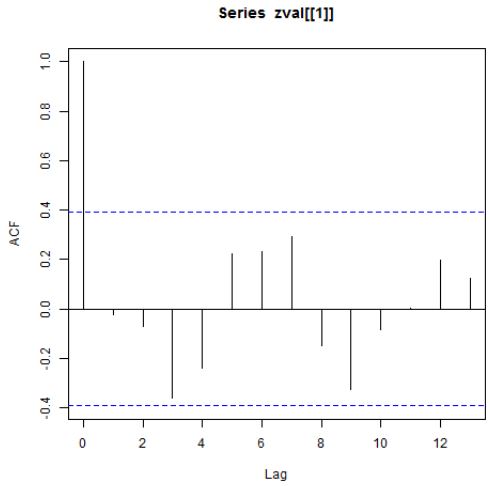
$$\begin{aligned}
p(z_i|y,x,z_{-i},\beta,c) &\propto p(y|c,z_i)p(z_i|\beta,x_i) \\
&= \mathbb{1}(z_i<c)^{1-y_i}\mathbb{1}(z_i>c)^{y_i}\text{Normal}(\beta x_i,1)
\end{aligned}$$

If $y_i=1$, this is simply the normal distribution $\text{Normal}(\beta x_i,1)$ bounded below by c. On the other hand, if $y_i=0$, then this is the same normal distribution bounded above by c.

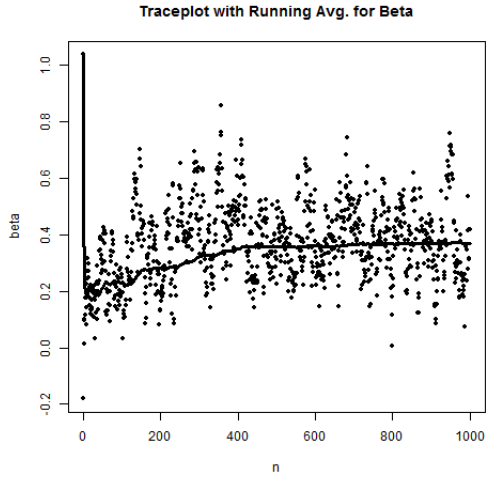
(c) Below are the plots of the autocorrelation function for one run:

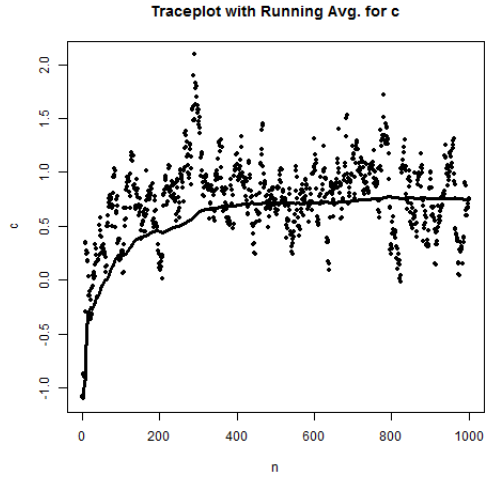


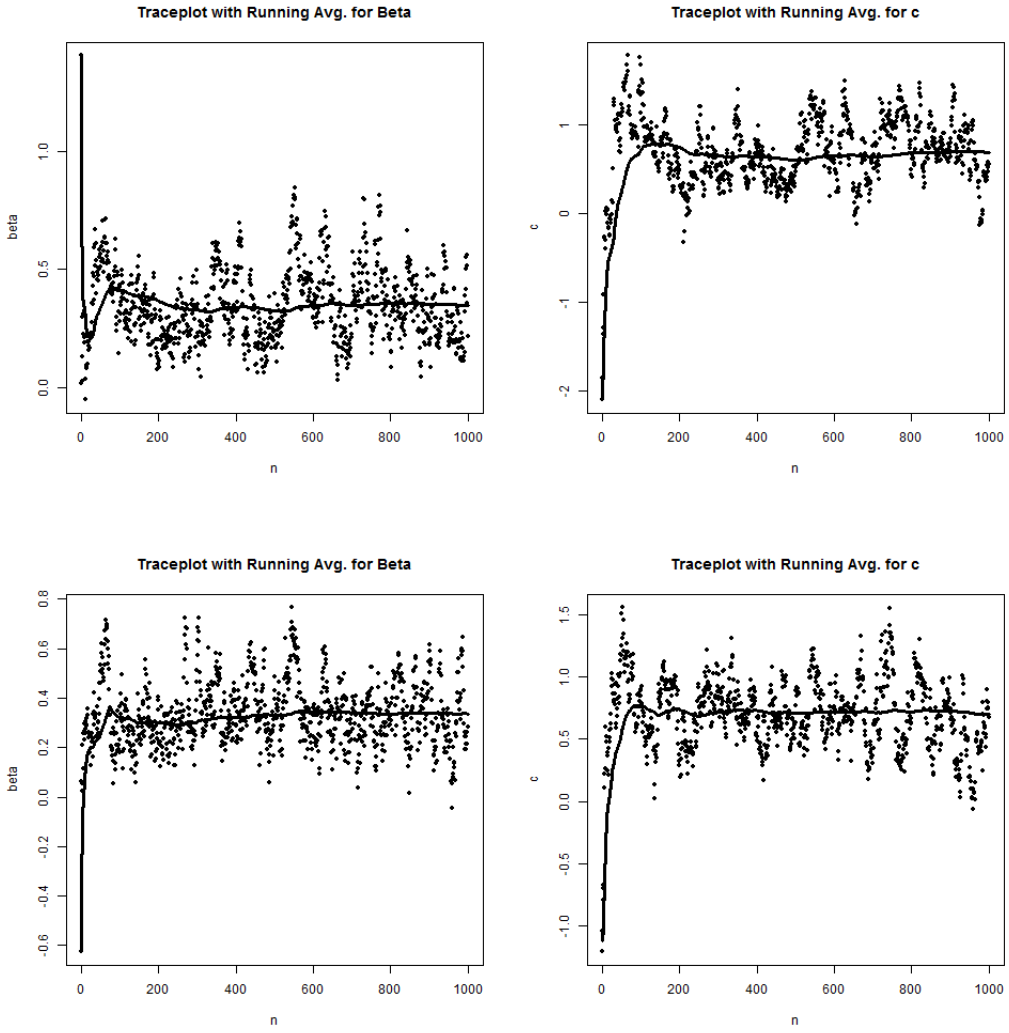




Also included here are the traceplots with running averages for β and c for 3 different initial values:







Based on these trace plots, the Markov chain mixes relatively well because the sampler seems to traverse through regions of low probability and the running average converges. Therefore, the sampler seems to have reached the right distribution. In addition, autocorrelations for β and c decrease over successive sample, which also indicates that the sampler mixes well.

(d) A 95% posterior confidence interval for β is:

$$[0.114, 0.630]$$

Furthermore, $\Pr(\beta > 0|y, x) = 0.998$.

R code for number 1:

```
## Initialize parameters ##
c = 0.02
n = 10^3
y = runif(1,0,1)

## Initialize vectors ##
xval = rep(NA,n)
yval = rep(NA,n)

yval[1] = y

x = runif(1,0,1)
while(abs(x-y)>=c){
  x = runif(1,0,1)
}
xval[1] = x;

## Gibbs Sampler ##
for(i in 2:n){
  y = runif(1,0,1)
  while(abs(xval[i-1]-y)>=c){
    y = runif(1,0,1)
  }
  yval[i] = y

  x = runif(1,0,1)
  while(abs(yval[i]-x)>=c){
    x = runif(1,0,1)
  }
  xval[i] = x
}

## Plot ##
##png("scat_002.png")
plot(xval, yval, pch=20, main = "Scatterplot (c=0.02)", xlab = "x", ylab = "y")
##dev.off()

##png("trace_002.png")
plot(1:n,xval,pch=20, main = "Traceplot (c=0.02)", xlab = "n", ylab = "x")
##dev.off()
```

R code for number 2:

```
## Initialize parameters ##
c = 0.25
n = 10^3
v = runif(1,-c/2,c/2)

## Initialize vectors ##
uval = rep(NA,n)
vval = rep(NA,n)

vval[1] = v
unif.min = max(0,abs(v))
unif.max = min(1,1-abs(v))

u = runif(1,unif.min,unif.max)

uval[1] = u;

## Gibbs Sampler ##
for(i in 2:n){
  v = runif(1,-c/2,c/2)
  vval[i] = v

  unif.min = max(0,abs(vval[i]))
  unif.max = min(1,1-abs(vval[i]))

  u = runif(1, unif.min, unif.max)

  uval[i] = u;
}

xval = uval + vval
yval = uval - vval

## Plot ##
png("scat_025x.png")
plot(xval, yval, pch=20, main = "Scatterplot (c=0.25)", xlab = "x", ylab = "y")
dev.off()

png("trace_025x.png")
plot(1:n,xval,pch=20, main = "Traceplot (c=0.25)", xlab = "n", ylab = "x")
dev.off()
```

R code for number 3:

```
data = read.table("divorce.dat")

x = data$V1
y = data$V2
l = length(x)

n = 1000
bval = rep(0,n)
cval = rep(0,n)
zval = list()

tb2=16
tc2=16

bval[1] = rnorm(1)
cval[1] = rnorm(1)
zval[[1]] = rnorm(l)

for(i in 2:n){
  # Sample for beta
  bval[i] = rnorm(1, mean = sum(x*zval[[i-1]])/(1/tb2+sum(x^2)), sd = sqrt((1/tb2 + sum(x^2))^-1))

  # Sample for c
  temp = sort(zval[[i-1]])
  maxList = temp[temp<cval[i-1]]
  maxList = sort(maxList, decreasing = T)
  if(is.na(maxList[1])){
    maxZ = -Inf  # maxZ is largest Z_i that is less than c
  }
  else{
  maxZ = maxList[1]
  }

  minList = temp[temp>cval[i-1]]
  minList = sort(minList)
  if(is.na(minList[1])){
    minZ = Inf   # minZ is smallest Z_i that is greater than c
  }
  else{
    minZ = minList[1]
  }

  umin = pnorm(maxZ, mean = 0, sd = sqrt(tc2))
  umax = pnorm(minZ, mean = 0, sd = sqrt(tc2))
  U = runif(1, umin, umax)
  cval[i] = qnorm(U, mean = 0, sd = sqrt(tc2))

  # Sample for z
  ztemp = rep(0,25)
  for(j in 1:25){
    if(y[j]==1){
      lbound = pnorm(cval[i], mean = bval[i]*x[j], sd = 1)
      U = runif(1, lbound, 1)
```

```
54        ztemp[j] = qnorm(U, mean = bval[i]*x[j], sd = 1)
55
56      }
57
58      if(y[j]==0){
59        ubound = pnorm(cval[i], mean = bval[i]*x[j], sd = 1)
60        U = runif(1, 0, ubound)
61        ztemp[j] = qnorm(U, mean = bval[i]*x[j], sd = 1)
62      }
63    }
64    zval[[i]] = ztemp
65
66  }
67  nval = 1:n
68
69  ## running averages
70  bavg = rep(0,n)
71  bavg[1] = bval[1]
72  for(k in 2:n){
73    bavg[k] = (bavg[k-1]*(k-1) + bval[k])/k
74  }
75
76  cavg = rep(0,n)
77  cavg[1] = cval[1]
78  for(k in 2:n){
79    cavg[k] = (cavg[k-1]*(k-1) + cval[k])/k
80  }
81
82  ## plot
83  #png("bacf.png")
84  print(acf(bval))
85  #dev.off()
86
87  #png("acfc.png")
88  print(acf(cval))
89  #dev.off()
90
91  #png("acfz.png")
92  print(acf(zval[[1]]))
93  #dev.off()
94
95  #png("mixb3.png")
96  plot(1:n, bval, pch=20, main = "Traceplot with Running Avg. for Beta", xlab = "n", ylab = "beta")
97  for(i in 1:n-1){
98    lines(c(nval[i],nval[i+1]),c(bavg[i],bavg[i+1]), lwd=3)
99  }
100 #dev.off()
101
102 #png("mixc3.png")
103 plot(1:n, cval, pch=20, main = "Traceplot with Running Avg. for c", xlab = "n", ylab = "c")
104 for(i in 1:n-1){
105   lines(c(nval[i],nval[i+1]),c(cavg[i],cavg[i+1]),lwd=3)
106 }
107 #dev.off()
```

```
108
109 ## Calculate other things
110 quantile(bval, probs=0.025)
111 quantile(bval, probs=0.975)
112
113 sum=0
114 for(i in 1:n){
115   if(bval[i]>0)
116     sum=sum+1
117 }
118 print(sum/length(bval))
```