Steven Maharaj 695281 Assignment 2, Question 3
MAST90125: Bayesian Statistical Learning

Due: Friday 20 September 2019

There are places in this assignment where R code will be required. Therefore set the random seed so assignment is reproducible.

```r
set.seed(695281)  #Please change random seed to your student id number.
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
```

```r
library(ggplot2)
library(mvtnorm)
library(coda)
```

Question Three (18 marks)

A group of 453 Bangladeshi women in 5 districts were asked about contraceptive use. The response variable *use* is an indicator for contraceptive use (coded N for no and Y for yes). Other covariates of interest are categorical variables for geographical location *district* (5 levels), and *urban* (2 levels), and number of living children *livch* (4 levels), and the continuous covariate for standardised age *age*.A random intercept for the district was suggested. This suggested the following model should be fitted,

$$\boldsymbol{\theta} = \mathbf{Z}\mathbf{u} + \mathbf{X}\beta,$$

where $\boldsymbol{\theta}$ is a link function, $\mathbf{Z}$ is an indicator variable for district, $\mathbf{u}$ is a random intercept with prior $p(\mathbf{u}) = \mathcal{N}(\mathbf{0}, \sigma_u^2\mathbf{I})$, and $\mathbf{X}$ is a design matrix for fixed effects $\boldsymbol{\beta}$, where $\boldsymbol{\beta}$ includes the coefficients for the intercept, urban status, living children, and age.

Data can be downloaded from LMS as `Contraceptionsubset.csv`.

a) Fit a generalised linear mixed model assuming a logistic link using `Stan`. The R and stan code below covers the following steps.

- Importing the data.
- Constructing design matrices.
- Provides code to go into the stan file.
- Running stan in R. This assumes your stan file is called *logitmm.stan*, and that you will run the sampler for 2000 iterations and 4 chains.

Note that provided code assumes everything required is located in your working directory in R.

```r
#Step one: Importing data,  constructing design matrices and calculating matrix dimensions.
dataX= read.csv("Contraceptionsubset.csv",header=TRUE)

n<-dim(dataX)[1]
Z    = table(1:n,dataX$district)     #incidence matrix for district
Q    = dim(Z)[2]
D1   = table(1:n,dataX$livch) #Dummy indicator for living children
D2   = table(1:n,dataX$urban) #Dummy indicator for urban status

#fixed effect design matrix
X    = cbind(rep(1,n),dataX$age,D1[,-1],D2[,-1])
P    = dim(X)[2]
y    = rep(0,n)
y[dataX$use %in% 'Y'] = 1
```

An example stan file.

```
//
// This Stan program defines a logistic mixed model
//
// Learn more about model development with Stan at:
//
//    http://mc-stan.org/users/interfaces/rstan.html
//    https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started
//

data {
  int<lower=0> n;   //number of observations
  int<lower=0> Q;   //number of random effect levels
  int<lower=0> P;   //number of fixed effect levels
  int y[n];      //response vector
  matrix[n,Q] Z;     //indicator matrix for random effect levels
  matrix[n,P] X;     //design matrix for fixed effects
}

// The parameters accepted by the model.
// accepts three sets of parameters 'beta', 'u' and 'sigma'.
parameters {
  vector[P] beta; //vector of fixed effects of length P.
  vector[Q] u; //vector of random effects of length Q.
  real<lower=0> sigma; //random effect standard deviation
}


// The model to be estimated. We model the output
// 'y' to be bernoulli with logit link function,
// and assume a i.i.d. normal prior for u.
model {
  u ~ normal(0,sigma);                //prior for random effects.
  y ~ bernoulli_logit(X*beta+ Z*u);  //likelihood

}
```

```r
library(rstan)
```

```
## Loading required package: StanHeaders
```

```
## rstan (Version 2.19.2, GitRev: 2e1f913d3ca3)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
```

```
## rstan_options(auto_write = TRUE)
##
## Attaching package: 'rstan'
## The following object is masked from 'package:coda':
##
##     traceplot
```

```r
logistic.mm <-stan(file="logitmm.stan",data=c('Z','X','y','n','P','Q'),iter=2000,chains=4)
```

```
##
## SAMPLING FOR MODEL 'logitmm' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 8.9e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.89 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 1.52893 seconds (Warm-up)
## Chain 1:                1.21177 seconds (Sampling)
## Chain 1:                2.7407 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'logitmm' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 4.3e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.43 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
```

```
## Chain 2:
## Chain 2:  Elapsed Time: 1.60601 seconds (Warm-up)
## Chain 2:                1.26134 seconds (Sampling)
## Chain 2:                2.86735 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'logitmm' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 4.7e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.47 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 1.52475 seconds (Warm-up)
## Chain 3:                1.18497 seconds (Sampling)
## Chain 3:                2.70972 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'logitmm' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 4.4e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.44 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 2.14194 seconds (Warm-up)
## Chain 4:                1.21487 seconds (Sampling)
## Chain 4:                3.35682 seconds (Total)
```

```
## Chain 4:
```

```r
print(logistic.mm)
```

```
## Inference for Stan model: logitmm.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##             mean se_mean   sd    2.5%     25%     50%     75%   97.5% n_eff
## beta[1]    -2.01    0.02 0.59   -3.21   -2.36   -2.00   -1.66   -0.81  1126
## beta[2]    -0.04    0.00 0.02   -0.08   -0.05   -0.04   -0.03   -0.01  2301
## beta[3]     1.23    0.01 0.34    0.57    1.01    1.23    1.46    1.89  2432
## beta[4]     1.45    0.01 0.36    0.74    1.20    1.44    1.69    2.15  2234
## beta[5]     1.79    0.01 0.38    1.07    1.54    1.78    2.04    2.54  1610
## beta[6]     1.22    0.00 0.26    0.72    1.04    1.22    1.39    1.74  2732
## u[1]       -1.06    0.02 0.56   -2.27   -1.37   -1.04   -0.72   -0.06  1082
## u[2]       -0.25    0.02 0.57   -1.41   -0.57   -0.26    0.08    0.83  1239
## u[3]        0.43    0.02 0.56   -0.76    0.12    0.43    0.74    1.50  1055
## u[4]        0.18    0.02 0.56   -0.95   -0.13    0.18    0.51    1.28  1078
## u[5]        0.69    0.02 0.56   -0.44    0.37    0.67    1.00    1.82  1197
## sigma       1.10    0.02 0.70    0.41    0.70    0.93    1.27    2.86  1480
## lp__     -274.34    0.07 2.51 -280.05 -275.89 -274.04 -272.49 -270.41  1246
##          Rhat
## beta[1]     1
## beta[2]     1
## beta[3]     1
## beta[4]     1
## beta[5]     1
## beta[6]     1
## u[1]        1
## u[2]        1
## u[3]        1
## u[4]        1
## u[5]        1
## sigma       1
## lp__        1
##
## Samples were drawn using NUTS(diag_e) at Thu Sep 26 18:48:59 2019.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

Note that in Stan, defaults for burn-in (warm-up) is one half of all iterations in stan, and no thinning. Note the code is written using the stan file and csv is in your working directory. Use the `print` function to report posterior means, standard deviations, 95 % central credible intervals and state from the output whether you believe the chains have converged. Also report the reference categories for *urban* and *livch*.

Reporting for PART A:

posterior means, standard deviations can be found in the above table. The lower limit of the 95 % central credible intervals given by the column labeled "2.5%" while the upperer limit of the 95 % central credible intervals given by the column labeled "97.5%".

```r
print(summary(logistic.mm)$summary)
```

```
##                 mean      se_mean         sd        2.5%          25%
## beta[1]   -2.01030050 0.0177090209 0.59412704   -3.2126422   -2.35819525
```

```
## beta[2]   -0.04214017 0.0003647766 0.01749937   -0.0771671   -0.05338352
## beta[3]    1.23207517 0.0068231905 0.33646783    0.5659346    1.01390920
## beta[4]    1.44766009 0.0076461800 0.36137498    0.7375624    1.20459896
## beta[5]    1.78929396 0.0093728704 0.37613535    1.0743897    1.53649493
## beta[6]    1.21639026 0.0049852543 0.26059271    0.7185883    1.03977246
## u[1]      -1.06275576 0.0171555132 0.56436421   -2.2699659   -1.37142413
## u[2]      -0.25489581 0.0161806806 0.56952451   -1.4074344   -0.57137738
## u[3]       0.42541069 0.0173308629 0.56299322   -0.7568421    0.12006699
## u[4]       0.18475245 0.0169065649 0.55506518   -0.9452371   -0.12563814
## u[5]       0.69203489 0.0160977691 0.55701986   -0.4360102    0.36515485
## sigma      1.10279222 0.0181485968 0.69807840    0.4141299    0.69594181
## lp__    -274.33957141 0.0710899545 2.50946526 -280.0506863 -275.89450998
##                  50%          75%         97.5%    n_eff      Rhat
## beta[1]   -2.00071542   -1.66048095 -8.137643e-01 1125.562 1.0028517
## beta[2]   -0.04190229   -0.03039232 -9.684808e-03 2301.393 1.0008164
## beta[3]    1.22785036    1.45582991  1.893473e+00 2431.712 1.0003286
## beta[4]    1.44472312    1.69041346  2.151146e+00 2233.712 1.0022645
## beta[5]    1.77969771    2.03909874  2.538047e+00 1610.435 1.0021622
## beta[6]    1.21564004    1.38717360  1.744403e+00 2732.435 1.0020165
## u[1]      -1.03749524   -0.72160839 -6.331785e-02 1082.210 1.0035870
## u[2]      -0.25724419    0.07818563  8.251233e-01 1238.886 1.0028778
## u[3]       0.43126008    0.74223850  1.496745e+00 1055.276 1.0032488
## u[4]       0.17671158    0.50561014  1.278456e+00 1077.897 1.0029502
## u[5]       0.66847636    1.00332512  1.815226e+00 1197.319 1.0031387
## sigma      0.93149252    1.27266697  2.858592e+00 1479.525 1.0007791
## lp__    -274.04470478 -272.48530390 -2.704118e+02 1246.080 0.9999782
```

Using the Gelman-Rubin diagnostic (Rhat) only 6 out of the 12 parameters These were beta[2],beta[3],beta[4],beta[5],beta[6],sigm had an Rhat value close to 1. All u parameters and beta[1] do not appear to converge thus the chain did not converge.

For *urban* and *livch* the reference categories are "N" and "0" respectivley. This is infered from the way the DataFrame X was constructed. `X = cbind(rep(1,n),dataX$age,D1[,-1],D2[,-1])`

b) An alternative to the logit link when analysing binary data is the probit. The probit link is defined as,

$$
\begin{aligned}
y_i &= \begin{cases} 1 & \text{if } z_i \geq 0 \\ 0 & \text{if } z_i < 0 \end{cases} \\
z_i &= \mathbf{x}_i'\boldsymbol{\beta} + \epsilon_i, \quad \epsilon \sim \mathcal{N}(0,1).
\end{aligned}
$$

In lecture 14, we showed how by letting z_i be normal, probit regression can be fitted using a Gibbs sampler, but to do so, it requires the ability to sample from a truncated normal defined on either $(-\infty, 0)$ (if $y_i = 0$) or $(0, \infty)$ (if $y_i = 1$). Check by comparing the empirical and the true density that a modified version of the inverse cdf method can be used to produce draws from a truncated normal. Do this for the case where $x \in (0, \infty)$ and $x \in (-\infty, 0)$ with parameters $\mu = 0.5$ and $\sigma = 1$.

Hints: If y is drawn from a truncated normal with lower bound a, upper bound b and parameters μ, σ^2 then then $p(y|\mu, \sigma^2, a, b)$ is

$$
\frac{\frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y-\mu)^2/2}}{\int_{-\infty}^{b} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y-\mu)^2/2} dy - \int_{-\infty}^{a} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y-\mu)^2/2} dy},
$$

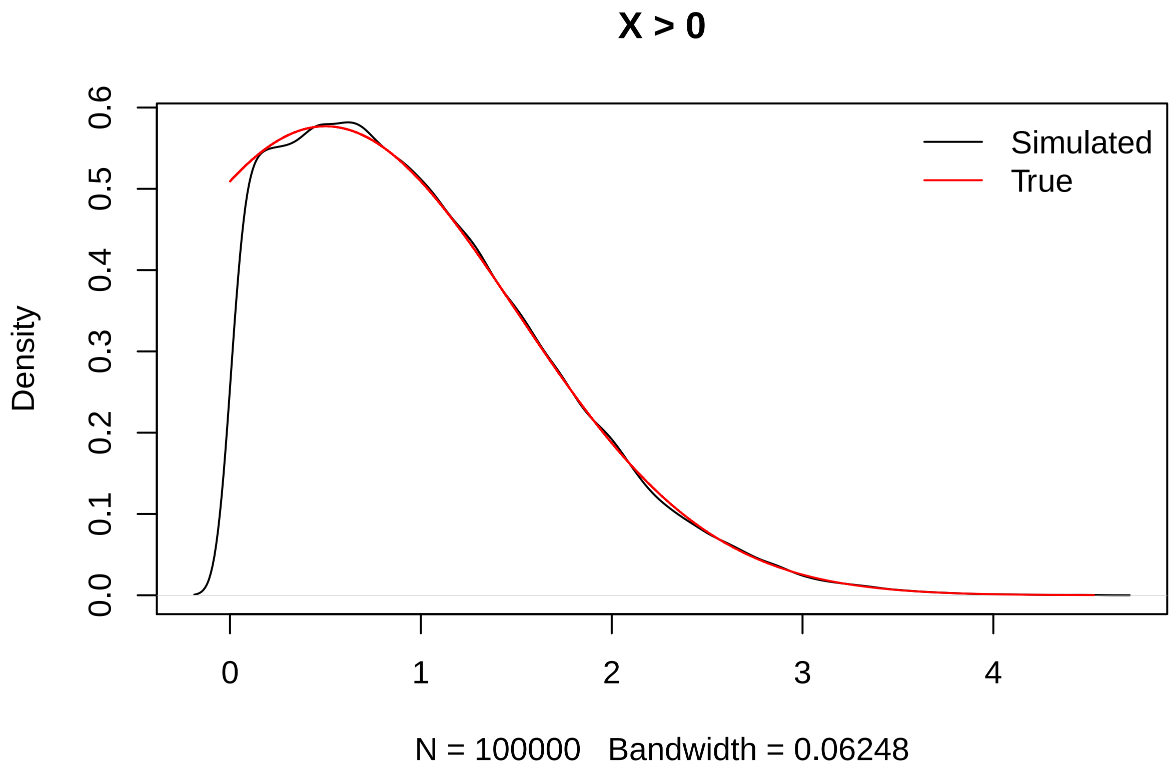
which in `R` means the truncated normal density can be written as `dnorm(x,mean=mu,sd=sigma)/(pnorm(b,mean=mu,sd=sigma)-pnorm(a,mean=mu,sd=sigma))`

The inverse cdf method involves drawing v from $U(0,1)$ so that $x \sim p(x)$ can be found solving $x = F^{-1}(x)$, where F is the cdf. If the only change compared to drawing from a normal distribution is truncation, think about what happens to the bounds of the uniform distribution.

Answer: Part B

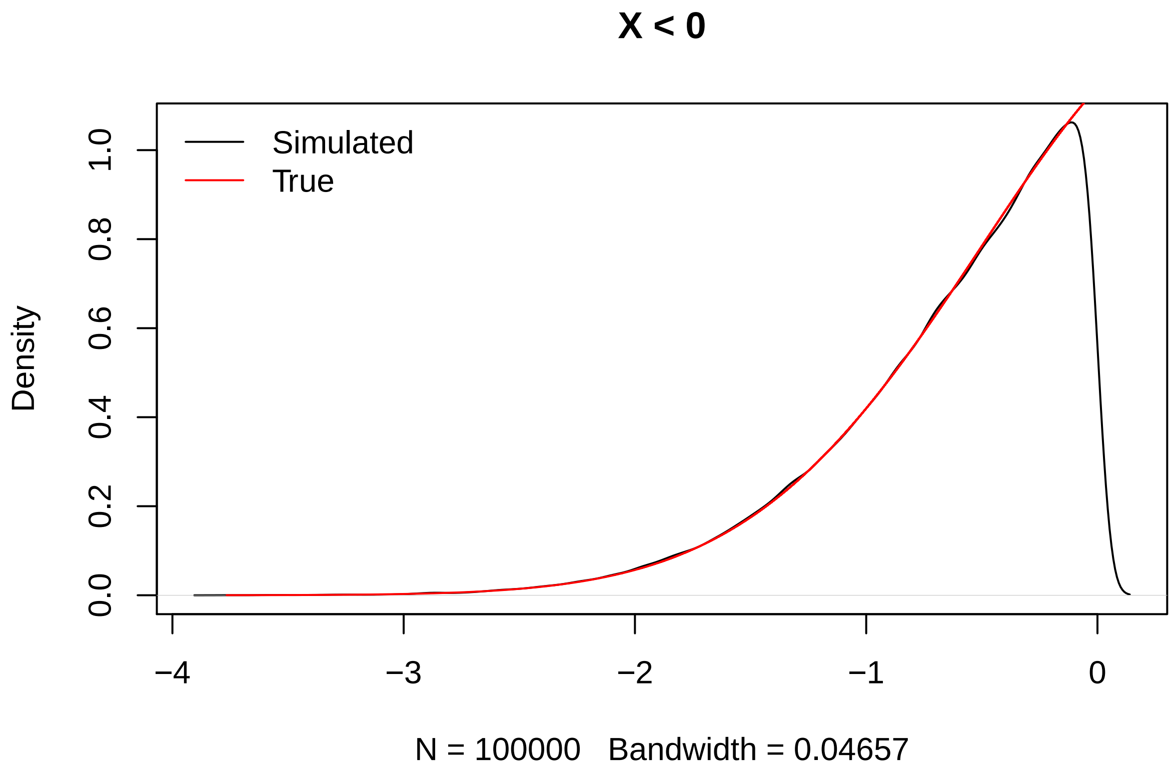
Case $x \in (0, \infty)$

```r
rtn <- function(n,b,a,mu,Sigma){
  u <- runif(n)
  g <- pnorm((b-mu)/Sigma) - pnorm((a-mu)/Sigma)
  x <-qnorm((g) * u + pnorm((a-mu)/Sigma))*Sigma + mu
}



x<- rtn(n=100000,b=100,a=0,mu = 0.5,Sigma = 1)

mu = 0.5
Sigma = 1
b=100
a=0


x<-sort(x)
true_den <- dnorm((x-mu)/Sigma)/(pnorm((b-mu)/Sigma) - pnorm((a-mu)/Sigma))

plot(density(x),col = 1,main="X > 0")
lines(x,true_den,col = 2,type = "l")
legend('topright',legend=c('Simulated','True'),col=1:2,lty=1,bty='n')
```



Case $x \in (-\infty, 0)$

```
x<- rtn(n=100000,b=0,a=-100,mu = 0.5,Sigma = 1)

mu = 0.5
Sigma = 1
b=0
a=-100


x<-sort(x)
true_den <- dnorm((x-mu)/Sigma)/(pnorm((b-mu)/Sigma) - pnorm((a-mu)/Sigma))

plot(density(x),col = 1,main="X < 0")
lines(x,true_den,col = 2,type = "l")
legend('topleft',legend=c('Simulated','True'),col=1:2,lty=1,bty='n')
```



c) Implement a Gibbs sampler to fit the same mixed model as fitted in Stan in a), but now with a probit link. As before, fit 4 chains, each running for 2000 iterations, with the first 1000 iterations discarded as burn-in. Perform graphical convergence checks and Gelman-Rubin diagnostics. Report posterior means, standard deviations and 95 % central credible intervals for $\sigma, \boldsymbol{\beta}, \mathbf{u}$ by combining chains.

We implement a Gibbs sampler to fit the same mixed model, but now with a probit link.

Assumsing,

- $p(\boldsymbol{\beta}) \propto 1$
- $p(\boldsymbol{u}) = \mathcal{N}(\mathbf{0}, \sigma_u^2 \boldsymbol{I})$
- $p(\tau_u) = Ga(\alpha_u, \gamma_u)$

It can be shown that we have the folling conditional posteriors

$$p\left(\tau_u \mid \cdot\right) = \mathrm{Ga}\left(\alpha_u + q/2, \gamma_u + \mathbf{u}'\mathbf{u}/2\right)$$

$$p\left(\left(\begin{array}{c} \beta \\ u \end{array}\right) \mid \cdot\right) = \mathcal{N}\left(\left(\begin{array}{cc} X'X & X'Z \\ Z'X & Z'Z + \tau_u \boldsymbol{I}^{-1} \end{array}\right)^{-1} \left(\begin{array}{c} X'z \\ Z'z \end{array}\right), \left(\begin{array}{cc} X'X & X'Z \\ Z'X & Z'Z + \tau_u \boldsymbol{I}^{-1} \end{array}\right)^{-1}\right)$$

We define our inputs for the Gibbs Sampler

```
#Step one: Importing data,  constructing design matrices and calculating matrix dimensions.
dataX= read.csv("Contraceptionsubset.csv",header=TRUE)
n<-dim(dataX)[1]
Z    = table(1:n,dataX$district)      #incidence matrix for district
Q    = dim(Z)[2]
D1   = table(1:n,dataX$livch) #Dummy indicator for living children
D2   = table(1:n,dataX$urban) #Dummy indicator for urban status
```

```
#fixed effect design matrix
X    = cbind(rep(1,n),dataX$age,D1[,-1],D2[,-1])
P    = dim(X)[2]
y    = rep(0,n)
y[dataX$use %in% 'Y'] = 1
a <- 0.01
g <- 0.01
# iter =2000
```

Construct a Gibbs sampler

```
Gibbsq3 <- function(iter,Z,X,y,burnin,tauu_0,a,g){
  q    = dim(Z)[2]
  p    = dim(X)[2]

  W<-cbind(X,Z)            #for the joint conditional posterior for b,u
  WTW <-crossprod(W)
  I0 <- diag(p+q)
  diag(I0)[1:p] <- 0

  #starting values.
  t_u <- tauu_0
  u   <-rnorm(q,0,sd=1/sqrt(t_u))
  b   <-rnorm(p,0,sd=1/sqrt(t_u))

  #storing results.
  par <-matrix(0,iter,p+q+1)

  for (i in 1:iter) {
    Prec <-WTW + t_u*I0
    t_u <- rgamma(1,a + q*0.5,g + crossprod(u)*0.5)
    z <- rtn(n=length(y),b=100,a=0,mu = crossprod(t(X),b)+crossprod(t(Z),u),Sigma = 1)*(y==1) + rtn(n=le
    z[is.nan(z)] = 0
    z[is.infinite(z)] = 0
    P.mean <- solve(Prec)%*%crossprod(W,z)
    P.var  <-solve(Prec)
    res <- rmvnorm(1,P.mean,P.var)
    b <- res[1:p]
    u <- res[p+1:q]
    par[i,]<-c(b,u,1/t_u)
  }
par <-par[-c(1:burnin),] #removing initial iterations
colnames(par)<-c(paste('beta',1:p,sep=''),paste('u',1:q,sep=''),"sigma2_u")
 return(par)
}
```

```
chain1 <- Gibbsq3(iter=2000,Z=Z,X=X,y=y,burnin=1000,tauu_0 = 1,a=a,g=g)
chain2 <- Gibbsq3(iter=2000,Z=Z,X=X,y=y,burnin=1000,tauu_0 = 0.5,a=a,g=g)
chain3 <- Gibbsq3(iter=2000,Z=Z,X=X,y=y,burnin=1000,tauu_0 = 2,a=a,g=g)
chain4 <- Gibbsq3(iter=2000,Z=Z,X=X,y=y,burnin=1000,tauu_0 = 5,a=a,g=g)



ml1<-as.mcmc.list(as.mcmc((chain1[1:500,])))
ml2<-as.mcmc.list(as.mcmc((chain2[1:500,])))
```

```r
ml3<-as.mcmc.list(as.mcmc((chain3[1:500,])))
ml4<-as.mcmc.list(as.mcmc((chain4[1:500,])))
ml5<-as.mcmc.list(as.mcmc((chain1[500+1:500,])))
ml6<-as.mcmc.list(as.mcmc((chain2[500+1:500,])))
ml7<-as.mcmc.list(as.mcmc((chain4[500+1:500,])))
ml8<-as.mcmc.list(as.mcmc((chain4[500+1:500,])))
estml<-c(ml1,ml2,ml3,ml4,ml5,ml6,ml7,ml8)
```

```r
#Gelman-Rubin diagnostic.
gelman.diag(estml)[[1]]
```

```
##          Point est. Upper C.I.
## beta1      1.006765   1.015685
## beta2      1.004688   1.012092
## beta3      1.007661   1.018578
## beta4      1.012278   1.028304
## beta5      1.008874   1.021522
## beta6      1.003811   1.010400
## u1         1.006061   1.009664
## u2         1.003382   1.005225
## u3         1.003820   1.005600
## u4         1.004594   1.010112
## u5         1.004628   1.010125
## sigma2_u   1.229076   1.278956
```

```r
#effective sample size.
effectiveSize(estml)
```

```
##    beta1    beta2    beta3    beta4    beta5    beta6       u1       u2
## 2634.014 1510.871 1389.718 1358.778 1356.406 1487.991 2687.404 3047.083
##       u3       u4       u5 sigma2_u
## 3576.350 2849.980 3214.235 2010.176
```

The Gelman-Rubin diagnostic showns most parameters have converged except sigma2_u as there Rhat value are sufficiently close to one.

```r
#Reporting posterior means and credible intervals.
#Means
colMeans(rbind(chain1,chain2,chain3,chain4))
```

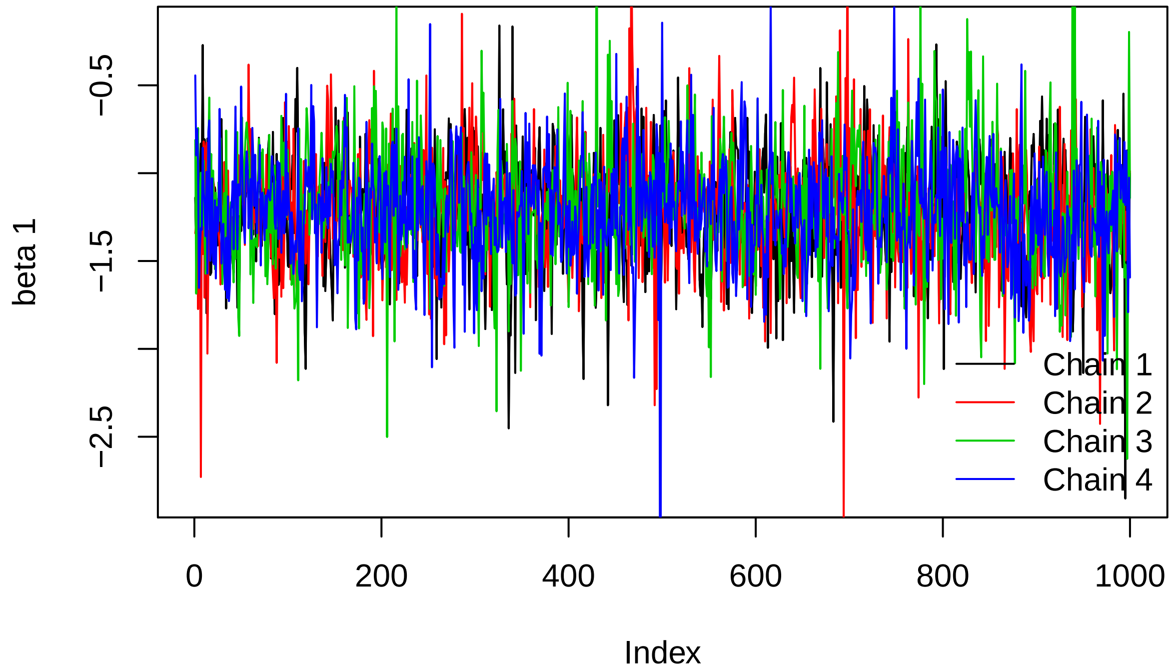
```
##       beta1       beta2       beta3       beta4       beta5       beta6
## -1.20492128 -0.02461357  0.73893378  0.86302252  1.06957571  0.73649399
##          u1          u2          u3          u4          u5    sigma2_u
## -0.61837533 -0.15145391  0.24689652  0.10747145  0.40255668  0.36395473
```

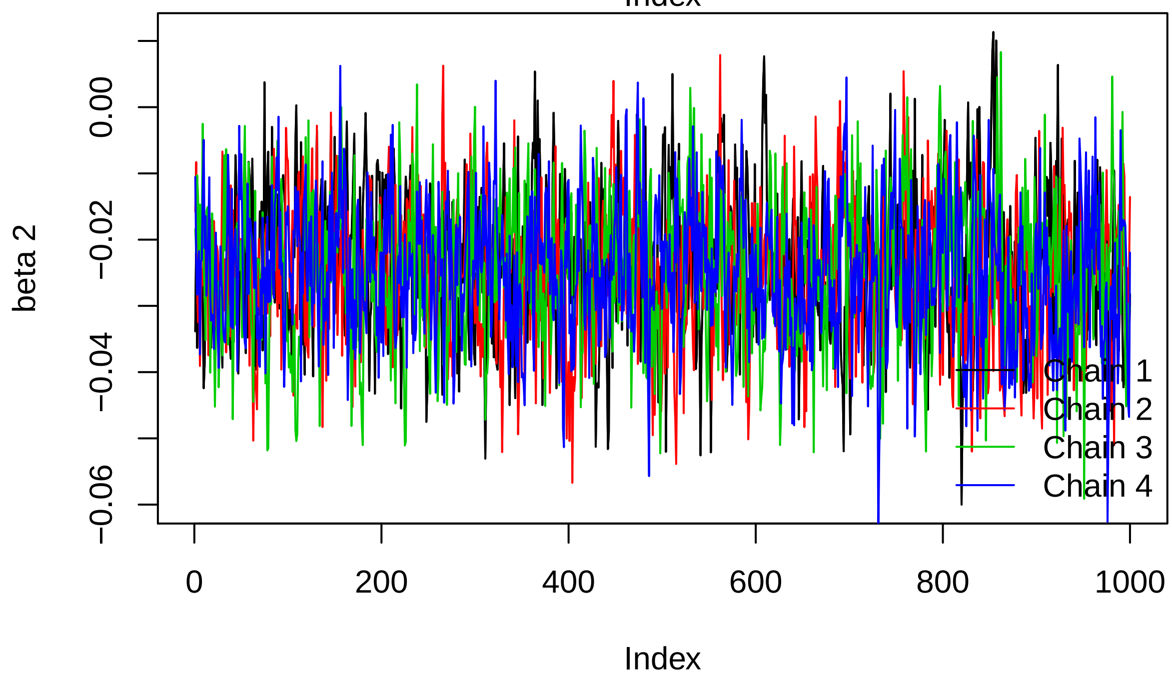
```r
#95 % central Credible interval
apply(rbind(chain1,chain2,chain3,chain4) ,2, FUN =function(x) quantile(x,c(0.025,0.975) ))
```

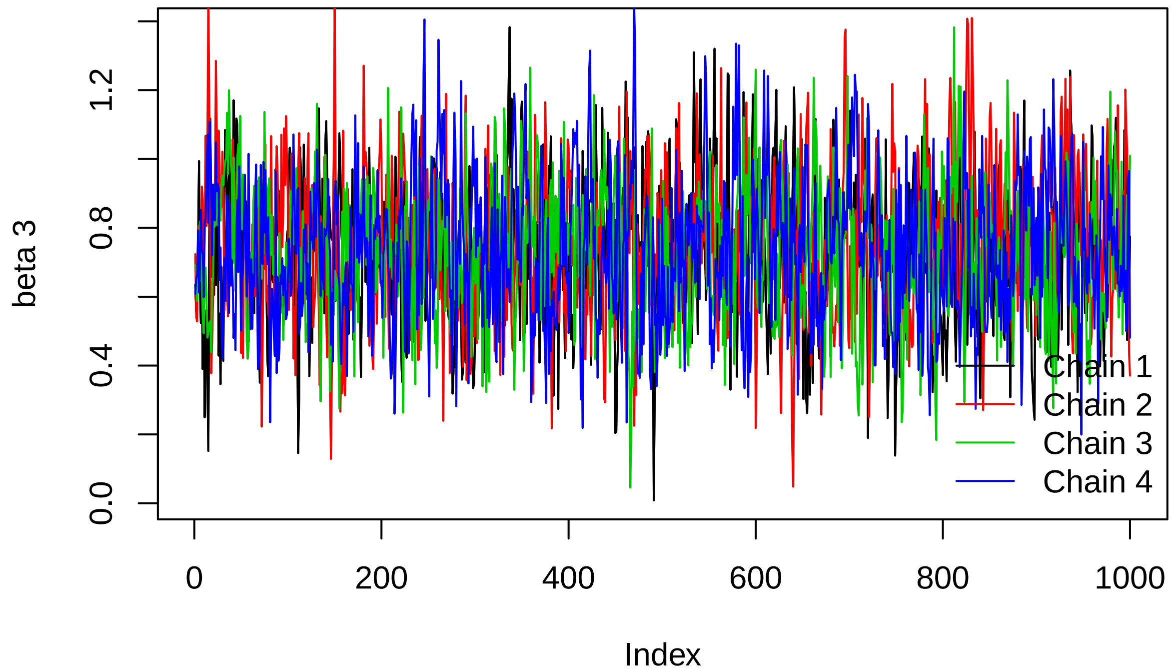
```
##            beta1        beta2     beta3     beta4     beta5     beta6
## 2.5%  -1.8362102 -0.044954070 0.3528615 0.4275205 0.6258834 0.4163952
## 97.5% -0.5813139 -0.004112405 1.1421664 1.3040343 1.5221386 1.0505475
##                u1         u2         u3         u4        u5   sigma2_u
## 2.5%  -1.20235126 -0.7576729 -0.3399557 -0.4781558 -0.1458978 0.05185139
## 97.5% -0.06249074  0.4502347  0.8265559  0.6688757  0.9920050 1.45070931
```

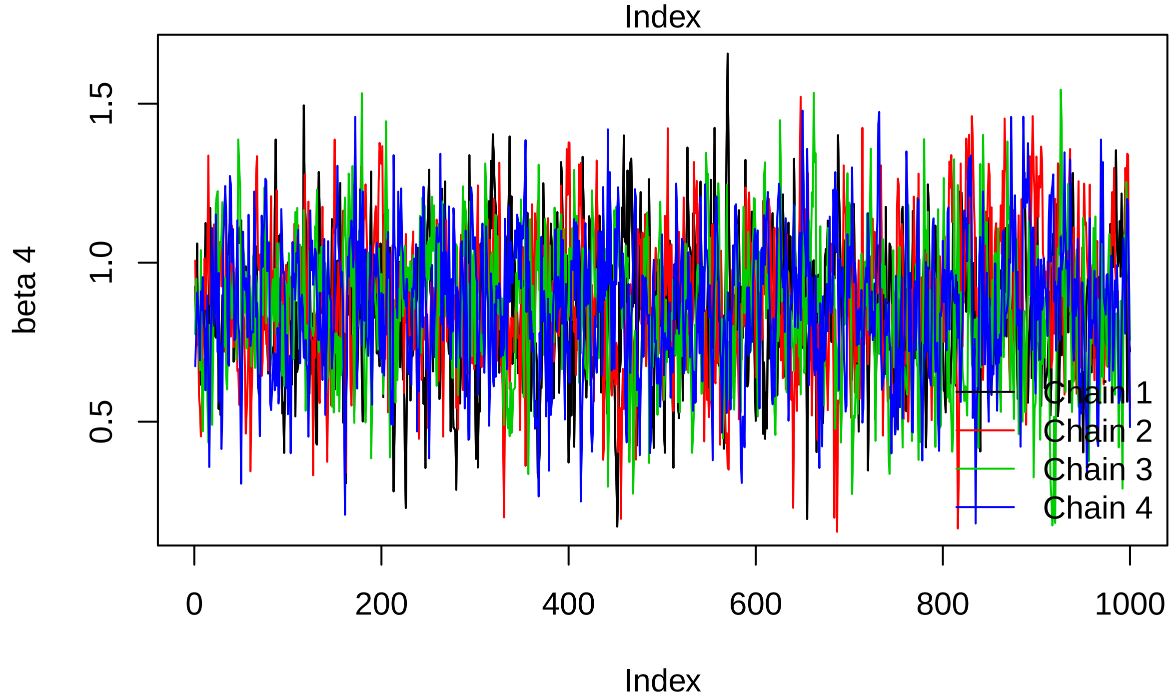
```r
# beta
for(i in 1:6){
```

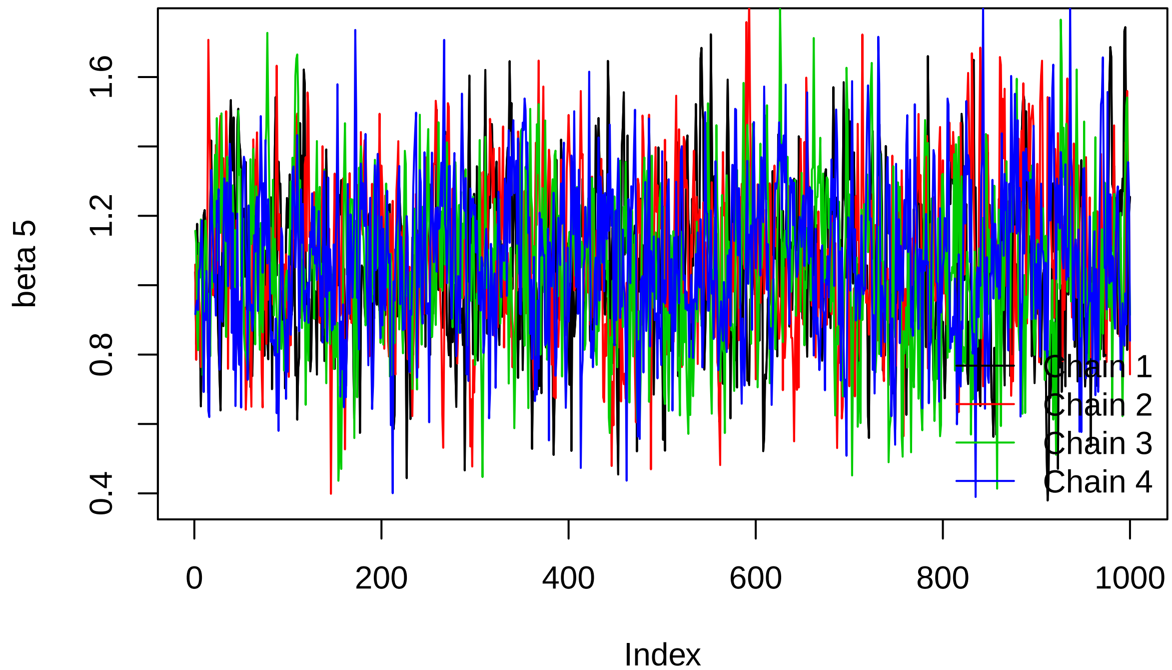
```
plot(chain1[,i],type='l',ylab=paste("beta",as.character(i)),col=1)
lines(chain1[,i],type='l',col=1,ylab=expression(beta[i]))
lines(chain2[,i],type='l',col=2,ylab=expression(beta[i]))
lines(chain3[,i],type='l',col=3,ylab=expression(beta[i]))
lines(chain4[,i],type='l',col=4,ylab=expression(beta[i]))
legend('bottomright',legend=c('Chain 1','Chain 2','Chain 3','Chain 4'),col=1:4,lty=1,bty='n')}
```

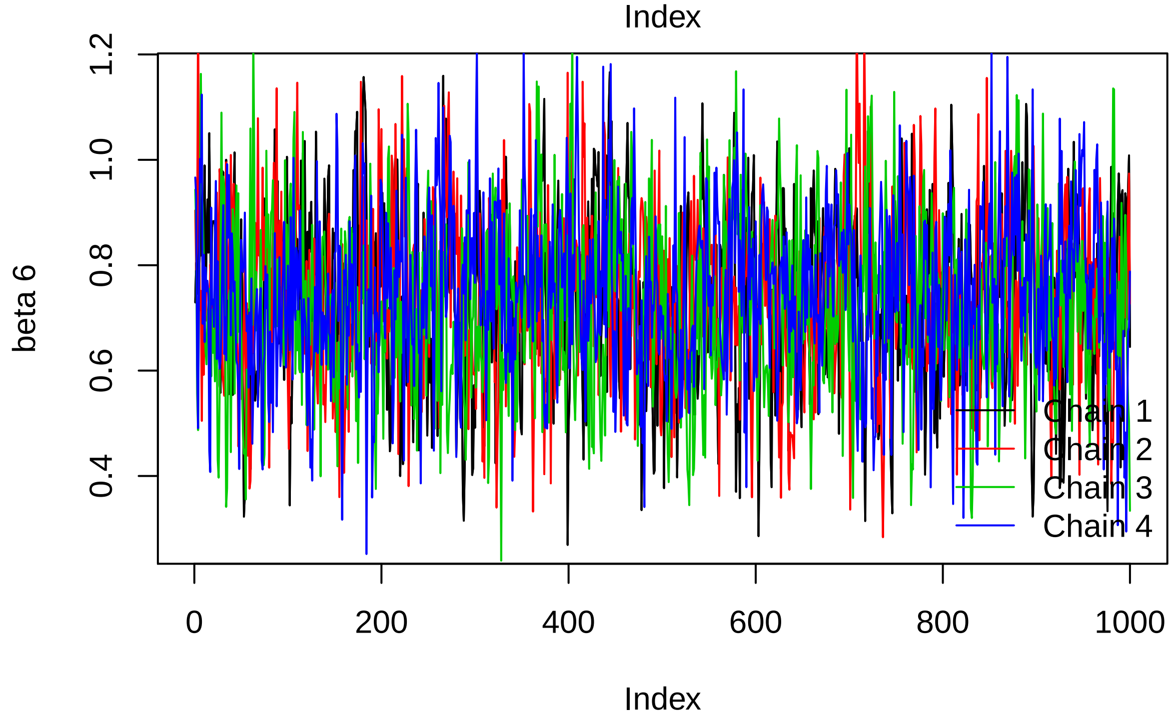




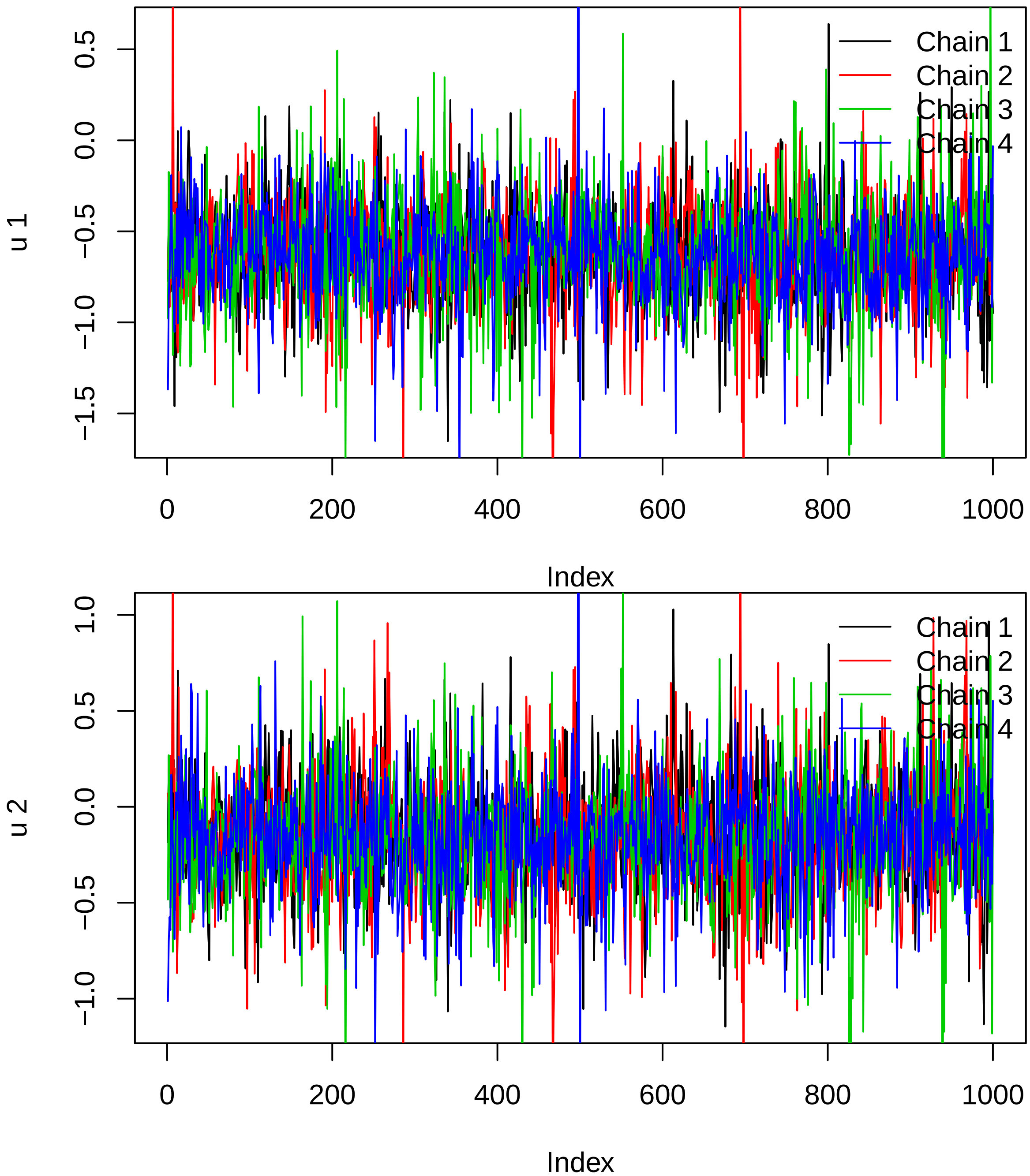

beta 3
0.0
0.4
0.8
1.2
0
200
400
600
800
1000
Index
Chain 1
Chain 2
Chain 3
Chain 4


beta 4
0.5
1.0
1.5
0
200
400
600
800
1000
Index
Chain 1
Chain 2
Chain 3
Chain 4

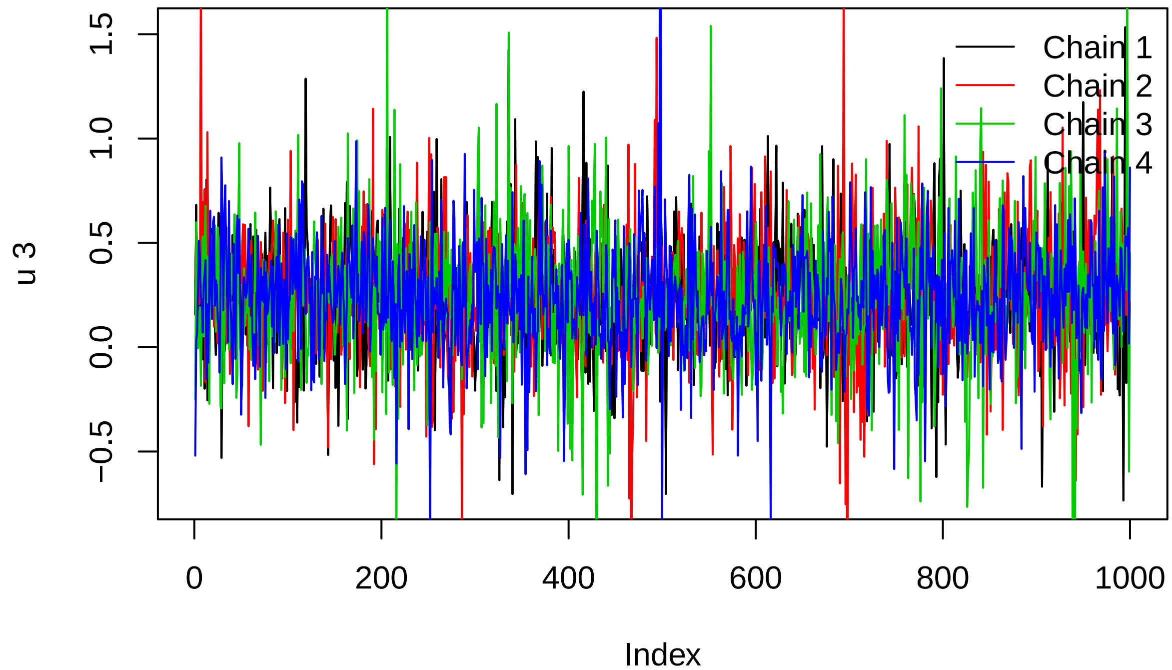




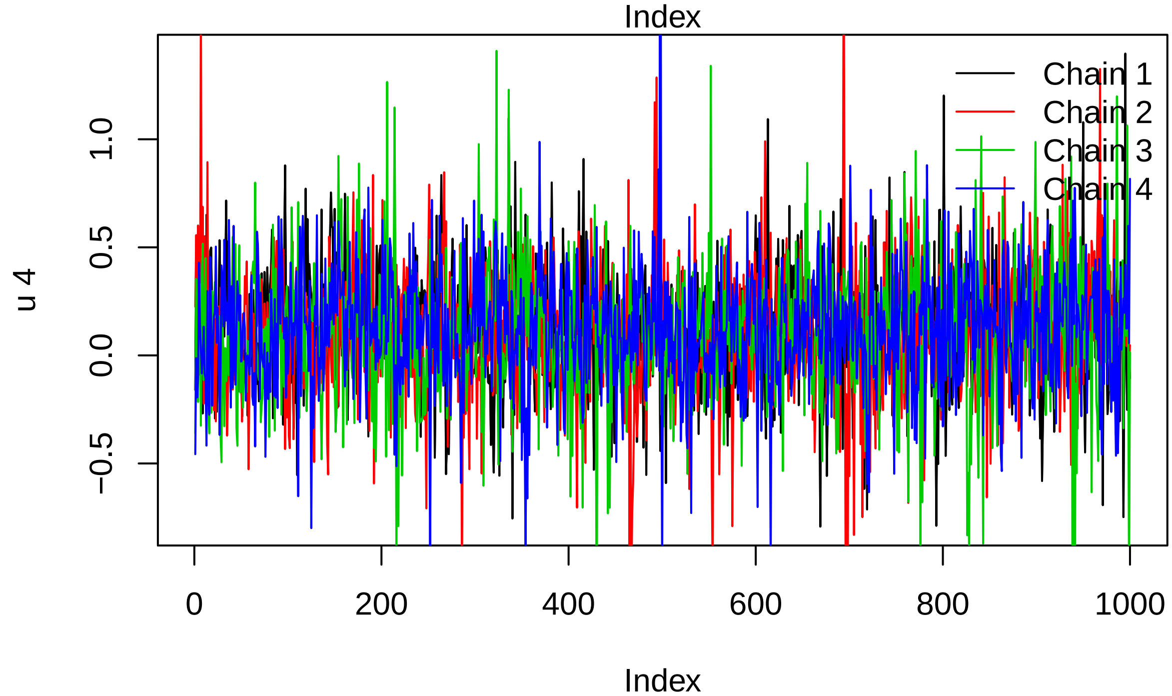

```
# u
for(i in 1:5){

  plot(chain1[,i+6],type='l',ylab=paste("u",as.character(i)),col=1)
  lines(chain1[,i+6],type='l',col=1)
  lines(chain2[,i+6],type='l',col=2)
  lines(chain3[,i+6],type='l',col=3)
  lines(chain4[,i+6],type='l',col=4)
  legend('topright',legend=c('Chain 1','Chain 2','Chain 3','Chain 4'),col=1:4,lty=1,bty='n')}
```



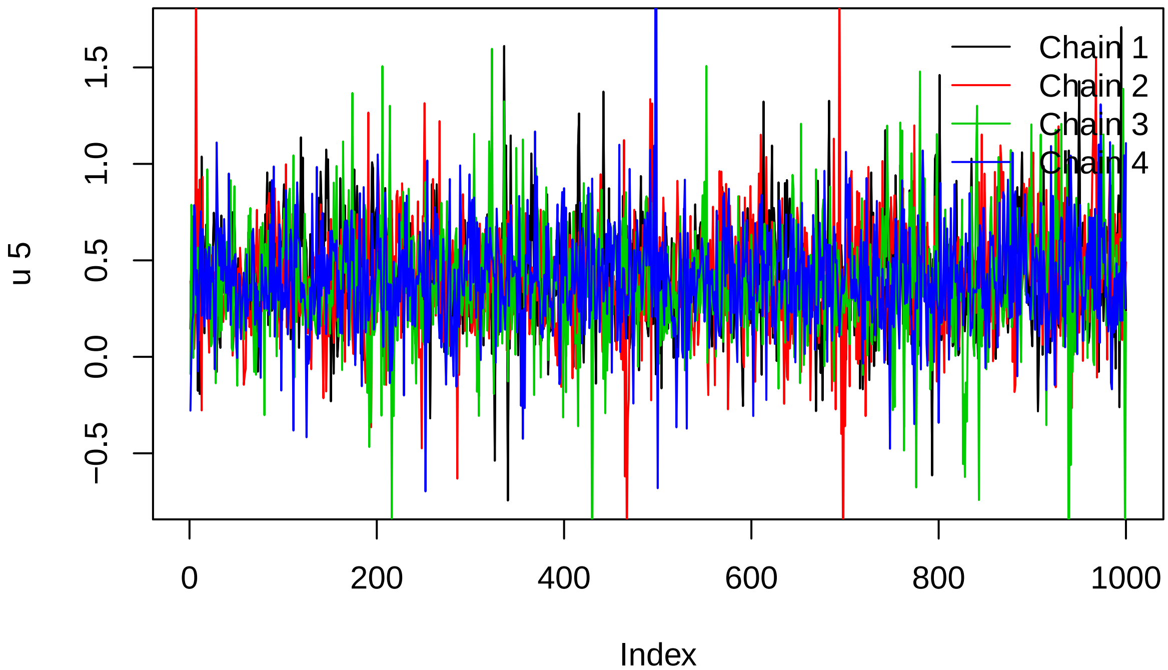
u 1
0.5
0.0
−0.5
−1.0
−1.5
0
200
400
600
800
1000
Index
Chain 1
Chain 2
Chain 3
Chain 4
u 2
1.0
0.5
0.0
−0.5
−1.0
0
200
400
600
800
1000
Index
Chain 1
Chain 2
Chain 3
Chain 4



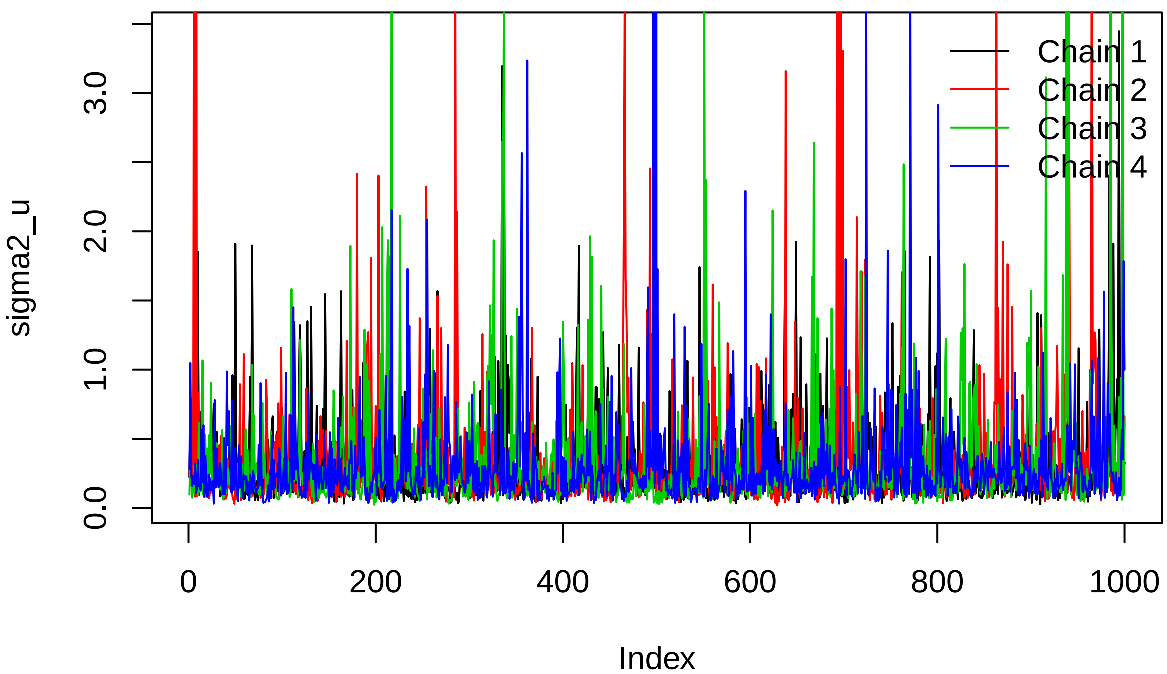
u 3
Chain 1
Chain 2
Chain 3
Chain 4
−0.5
0.0
0.5
1.0
1.5
0
200
400
600
800
1000
Index



u 4
Chain 1
Chain 2
Chain 3
Chain 4
−0.5
0.0
0.5
1.0
0
200
400
600
800
1000
Index




```
# u
plot(chain1[,12],type='l',ylab=paste("sigma2_u"),col=1)
lines(chain1[,12],type='l',col=1)
lines(chain2[,12],type='l',col=2)
lines(chain3[,12],type='l',col=3)
lines(chain4[,12],type='l',col=4)
legend('topright',legend=c('Chain 1','Chain 2','Chain 3','Chain 4'),col=1:4,lty=1,bty='n')
```



d) For the co-efficients $\boldsymbol{\beta}$, $\mathbf{u}$, calculate the mean of the ratio of the posterior means $\boldsymbol{\beta}_{i,\text{logit}}/\boldsymbol{\beta}_{i,\text{probit}}$, $\mathbf{u}_{i,\text{logit}}/\mathbf{u}_{i,\text{probit}}$ obtained when fitting the logistic mixed model and the probit mixed model. To do this, you will need to apply the `extract` function to the stan model object. Once calculated, multiply the iterations obtained assuming a probit link by this constant and compare to the iterations obtained assuming a logit link.

Answer: PART D

```
means_prob <- colMeans(rbind(chain1,chain2,chain3,chain4))
log_variables <- extract(logistic.mm)
means_log <- c(colMeans(log_variables$beta),colMeans(log_variables$u),mean(log_variables$sigma))
ratio <- means_log/means_prob
ratio
```

```
##    beta1    beta2    beta3    beta4    beta5    beta6       u1       u2
## 1.668408 1.712070 1.667369 1.677430 1.672901 1.651596 1.718626 1.682993
##       u3       u4       u5 sigma2_u
## 1.723032 1.719084 1.719099 3.030026
```

```
# chains multiplied by the ratio

chain1s <- chain1*ratio
chain2s <- chain2*ratio
chain3s <- chain3*ratio
chain4s <- chain4*ratio


ml1s<-as.mcmc.list(as.mcmc((chain1[1:500,])))
ml2s<-as.mcmc.list(as.mcmc((chain2[1:500,])))
ml3s<-as.mcmc.list(as.mcmc((chain3[1:500,])))
ml4s<-as.mcmc.list(as.mcmc((chain4[1:500,])))
ml5s<-as.mcmc.list(as.mcmc((chain1[500+1:500,])))
ml6s<-as.mcmc.list(as.mcmc((chain2[500+1:500,])))
ml7s<-as.mcmc.list(as.mcmc((chain4[500+1:500,])))
ml8s<-as.mcmc.list(as.mcmc((chain4[500+1:500,])))
estmls<-c(ml1s,ml2s,ml3s,ml4s,ml5s,ml6s,ml7s,ml8s)
```

```
gelman.diag(estml)[[1]]
```

```
##          Point est. Upper C.I.
## beta1      1.006765   1.015685
## beta2      1.004688   1.012092
## beta3      1.007661   1.018578
## beta4      1.012278   1.028304
## beta5      1.008874   1.021522
## beta6      1.003811   1.010400
## u1         1.006061   1.009664
## u2         1.003382   1.005225
## u3         1.003820   1.005600
## u4         1.004594   1.010112
## u5         1.004628   1.010125
## sigma2_u   1.229076   1.278956
```

```
#effective sample size.
effectiveSize(estmls)
```

```
##    beta1    beta2    beta3    beta4    beta5    beta6       u1       u2
## 2634.014 1510.871 1389.718 1358.778 1356.406 1487.991 2687.404 3047.083
##       u3       u4       u5 sigma2_u
## 3576.350 2849.980 3214.235 2010.176
```

```r
#Reporting posterior means and credible intervals.
#Means
colMeans(rbind(chain1s,chain2s,chain3s,chain4s))
```

```
##       beta1       beta2       beta3       beta4       beta5       beta6
## -2.17157563 -0.04441606  1.33348690  1.55678291  1.92745112  1.32798677
##          u1          u2          u3          u4          u5    sigma2_u
## -1.11500364 -0.27368429  0.44620330  0.19509498  0.72529798  0.65743607
```

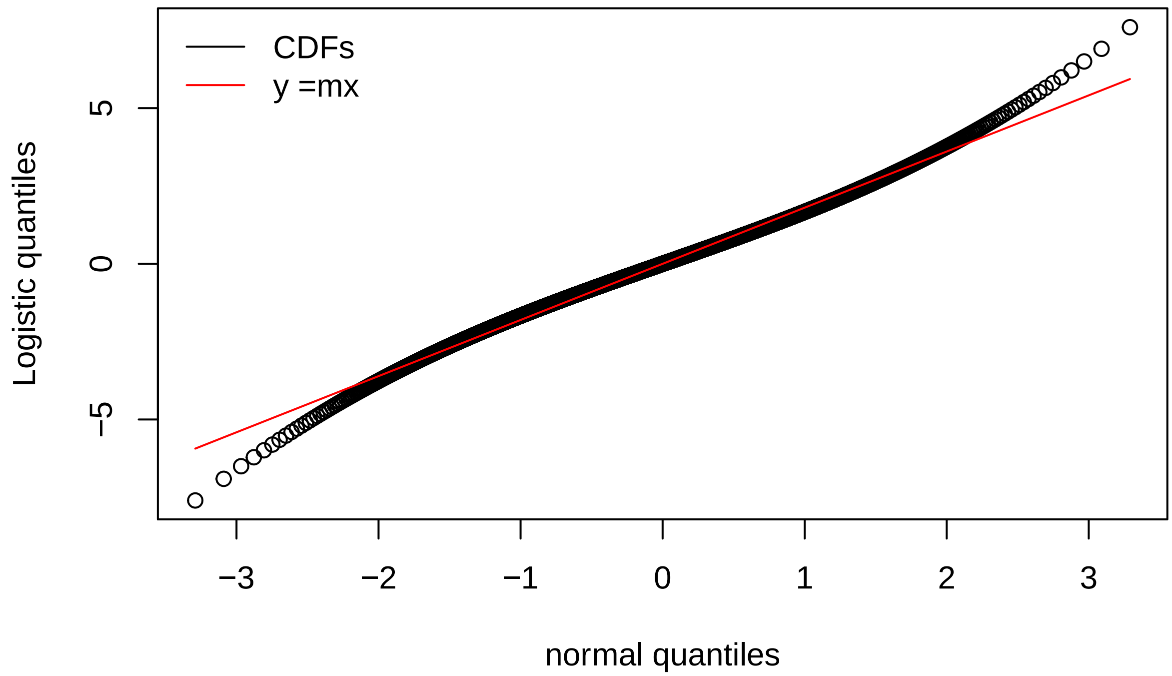
```r
#95 % central Credible interval
apply(rbind(chain1s,chain2s,chain3s,chain4s) ,2, FUN =function(x) quantile(x,c(0.025,0.975) ))
```

```
##           beta1        beta2     beta3     beta4    beta5     beta6
## 2.5%  -4.138895 -0.091750629 0.6074114 0.7300277 1.073574 0.7184235
## 97.5% -1.009394 -0.007043983 2.5855059 2.9857478 3.568723 2.4282676
##               u1         u2         u3         u4        u5   sigma2_u
## 2.5%  -2.4618543 -1.4196952 -0.6017555 -0.8622052 -0.250886 0.09074866
## 97.5% -0.1079262  0.8277801  1.5960446  1.2717346  1.948877 2.68763779
```

From the above report we can see that the posterior means of the probit link model multiplied by the ratio of posterior means are much closer to the log link model than the probit model alone. Also we see that all parameters converged except for sigma2_u.

e) The logistic link can be written in the same way as the probit link, but instead of $e_i \sim \mathcal{N}(0,1)$, the error term is $e_i \sim \text{Logistic}(0,1)$. By evaluating the standard normal and logistic inverse cdfs and superimposing the line $y = mx$ where m is the posterior ratio, do you think the results in d) were surprising.

```r
p <- seq(from = 0,to = 1,length.out = 2000)
xn <- qnorm(p)
xl <- qlogis(p)

plot(xn,xl,col=1,xlab="normal quantiles",ylab="Logistic quantiles")
lines(xn,mean(ratio)*xn,type='l',col=2)
legend('topleft',legend=c('CDFs','y =mx'),col=1:2,lty=1,bty='n')
```



From the above q-q plot we see that the logistic link is the normal link scaled by the posterior ratio (however the logistic link has slightly heavier tails). Hence once the iterations were scaled it was not surprising to see the posterior means for both model get closer to each other.