STATS551FinalProject

Mingyu Zhong

2022-12-10

Data Preprocessing

```r
# Initialize Environment
# setwd("Z:/CurrQuarter/STATS 551/Final Project/pubg-finish-placement-prediction")
setwd("D:/University of Michigan/CurrQuarter/STATS 551/Final Project/pubg-finish-placement-prediction")
rm(list=ls())
```

```r
# load full training data
raw.data = read.csv("train_V2.csv")
oriSamSize = nrow(raw.data)
full.data = raw.data

# fpp_list = unique(full.data$matchType)[grep("fpp", unique(full.data$matchType))]
# rm_mode_list = c(fpp_list, "flaretpp", "crashtpp")
# mode_list = c("squad","solo")
# # "duo",
#
# full.data = full.data %>% filter(matchType == mode_list[1] |
#                                    matchType == mode_list[2] )
# |matchType == mode_list[3]

full.data = full.data %>% filter(matchType == "duo")
# squad.data = full.data %>% filter(matchType == "squad")




full.data = full.data %>% mutate(ngid = paste0(matchId, groupId))

# table(full.data$groupId) %>% sort(decreasing = T)
range(full.data$winPlacePerc)
full.data = full.data %>% select(-c(Id, matchType,matchId,groupId))

# res.d = full.data %>% select(ngid, winPlacePerc)
#
# fg = table(full.data$ngid) %>% sort(decreasing = T)
# fg = names(fg[1])
#
# full.data %>% filter(ngid == fg)
```

```r
full.data = full.data %>%
  group_by(ngid) %>%
  summarise(across(everything(), mean))


full.data = full.data %>% select(-c(ngid))
```

Checking distribution of Predictors

```r
outliers.index = numeric()
trans.data  =full.data
inverseTrans = c(17)
needLogTransform = c(1:6,9:11,16,18:23)
needSecondLog = c(1,4:6,9,10,16,23)
needNegCorrect = c(13,14)
# for(i in 1:24){
#   # boxplot(X[,i], main = i)
#   outliers <- boxplot.stats(full.data[,i], coef = 43)$out
#   outliers.index = c(outliers.index,which(full.data[,i] %in% outliers))
# }
# outliers.index = unique(outliers.index)
# length(outliers.index)
# X = X[-outliers.index,]
# nrow(X)

for(i in 1:24){
  if(i %in% needLogTransform){
    trans.data[,i] = log(full.data[,i] + 1 )
  }
  if(i %in% needSecondLog){
    trans.data[,i] = log(trans.data[,i]+1)
  }
  if(i %in% inverseTrans){
    trans.data[,i] = 1/(full.data[,i]+1)
  }
  if(i %in% needNegCorrect){
    trans.data[,i] = log(sqrt(max(full.data[,i]+1)-full.data[,i]))
  }
  boxplot(trans.data[,i], main = i)
  # }
}
# ncol(X)
# for(i in 1:24){
#   print(paste(i, colnames(trans.data)[i]))
# }
full.data = trans.data
```

```r
# Split Train and Test Data
set.seed(551)
train.index = sample(seq(1,nrow(full.data)), size = 6*nrow(full.data)/10)
train.data = full.data[train.index,]
```

```r
test.data = full.data[-train.index,]

# Select important variables
str(train.data)
d = data.frame(train.data)
# non_numeric_columns <- sapply(d, is.numeric)
# colnames(d)[!non_numeric_columns]
# d = d %>% select(-c(Id, groupId, matchId, matchType))
y = d$winPlacePerc

# Standardize Predictors
td.mean = vector()
td.std = vector()
for(i in 1:(ncol(d)-1)){
  td.mean[i] = mean(d[,i])
  td.std[i] = sd(d[,i])
  d[,i] =  (d[,i] - td.mean[i]) / td.std[i]
}

# test.data = test.data %>% select(-c(Id, groupId, matchId, matchType))
str(test.data)
for(i in 1:(ncol(d)-1)){
  test.data[,i] = (test.data[,i]- td.mean[i]) / td.std[i]
}

# Save to a file for easy import
saveRDS(d, file = "split_train_dataset.rds")
saveRDS(test.data, file = "split_test_dataset.rds")
```

Start Here

Load Training Data

```
rm(list=ls())
tdata = readRDS("split_train_dataset.rds")
# testdata = readRDS("split_test_dataset.rds")
# tdata = rbind(tdata, testdata)
# str(tdata)
```

Model Selection with Gibbs

```
set.seed(5511)
p = ncol(tdata)
# Model.Selection.Gibbs = function(nruns, S){
nruns = 23
S=13
  runs.Z = matrix(nrow=p, ncol=0)
  runs.beta = matrix(nrow=p, ncol=0)
  for(nrun in 1:nruns){
    # reformat y
    y= as.matrix(tdata$winPlacePerc)
    one.index = which(y==1)
    second.highest = sort(unique(y), decreasing = T)[2]
    remake.one = runif(length(one.index), min = second.highest+0.0001,max= 1-0.0001)
    y[one.index] = remake.one
    zero.index = which(y==0)
    second.lowest = sort(unique(y))[2]
    remake.zero = runif(length(zero.index), min = 0+0.0001,max= second.lowest-0.0001)
    y[zero.index] = remake.zero
    y=qnorm(y)


    z = as.matrix(rep(1,p+1))
    X = tdata %>% select(-c(winPlacePerc)) %>% as.matrix
    X = cbind(intercept = rep(1, nrow(X)), X) %>% as.matrix

    # downsample
    samples.index = sample(seq(1,nrow(X)), size = nrow(X)/200)
    X = X[samples.index,]
    y = y[samples.index]
    g=nrow(X)
    n=nrow(X)
    v0 = 2
    sigma0.2 = var(tdata$winPlacePerc)

    lpy.X = function(y,X,g=length(y),nu0=2,s20 = try(summary(lm(y~-1+X))$sigma^2,silent=TRUE)){
      n = dim(X)[1]
      p=dim(X)[2]
      if(p==0){
```

```
      Hg=0
      S20 = mean(y^2)
    }
    if(p>0){
      Hg = (g/(g+1)) * X %*% solve(t(X)%*%X)%*%t(X)
    }
    SSRg = t(y)%*%(diag(1,nrow=n) -Hg)%*% y
    -0.5 *(n*log(pi)+p*log(1+g)+(nu0+n)*log(nu0*s20+SSRg) -nu0*log(nu0*s20))+lgamma((nu0+n)/2) - lgamm
  }

  z = rep(1,dim(X)[2])
  lpy.c = lpy.X(y,X[,z==1,drop=FALSE])
  # S=40
  Z = matrix(NA,S,dim(X)[2])
  # sigma2 = vector()
  beta = matrix(0,nrow=p,ncol=S)

  #Gibbs sampler
  set.seed(551)
  for(s in 1:S){
    for(j in sample(1:dim(X)[2])){
      zp = z
      zp[j]=1-zp[j]
      lpy.p = lpy.X(y,X[,zp==1,drop=FALSE])
      r=(lpy.p-lpy.c)*(-1)^(zp[j]==0)
      z[j]=rbinom(1,1,1/(1+exp(-r)))
      if(z[j]==zp[j]){lpy.c = lpy.p}
    }
    Z[s,] = z
    currZ = Z[s,]

    Xz = X[,currZ==1,drop=FALSE]
    v0 = 2
    sigma0.2 = 1

    m = g/(g+1) * (solve((t(Xz)%*%Xz))%*%t(Xz) %*% y)
    SSRg = t(y) %*% (diag(nrow(X)) - g/(g+1) * Xz %*% solve((t(Xz)%*%Xz))%*%t(Xz)) %*% y

    invga.shape = (v0+n) /2
    invga.rate = (v0+sigma0.2 + SSRg)/2

    sim.sigma = rinvgamma(1, invga.shape, invga.rate)
    # sigma2[s] = sim.sigma
    V = g/(g+1) * (sim.sigma * solve((t(Xz)%*%Xz)))
    sim.beta = mvtnorm::rmvnorm(1, mean = m, sigma = V)
    beta[which(currZ ==1), s] = t(sim.beta)
  }

  runs.beta = cbind(runs.beta,beta)
  runs.Z =  cbind(runs.Z,t(Z))
  print(nrun)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
## [1] 18
## [1] 19
## [1] 20
## [1] 21
## [1] 22
## [1] 23
```

```
  # return(list(runs.beta,runs.Z))
# }


# cached_result = memoise::cache(Model.Selection.Gibbs())
# for(i in 1:p){
#   beta_j = beta[i,]
#   print(length(beta_j[beta_j != 0]) / S)
#   print(cat("The 95% CI for beta_",i,"is",quantile(beta_j, 0.05/150),quantile(beta_j, (1-0.05/150))))
# }

# 25:40 - 248.17
# 100:500 - 142
# 200:1000 - 89.98
# 200:1100 - 95.84
# 230:1200 - 87.77
```

```
#Print the Highest Prob model
Zcomb = vector()
for(i in 1:S){
  Zcomb[i]= paste(runs.Z[,i],collapse=' ')
}
highest_comb = vector()
Zcomb = as.data.frame(table(Zcomb))
Zcomb = Zcomb[order(Zcomb$Freq, decreasing = T),]
# Zcomb$Freq = round(Zcomb$Freq / sum(Zcomb$Freq), 3)
Zcomb[1,]$Zcomb

for(i in 1:S){
  if(paste(runs.Z[,i],collapse=' ') == Zcomb[1,]$Zcomb){
    highest_comb = runs.Z[,i]
  }
}
colnames(X)[which(highest_comb %% 2 == 1)]
```

```
# Rank of magnitude
gib.beta.mean = vector()
for(i in 1:p){
  gib.beta.mean[i] = mean(runs.beta[i,])
}
gibs.coeff = cbind(gib.beta.mean=abs(round(gib.beta.mean,3)), name =  c("intercept",colnames(tdata[1:24]
gibs.coeff = gibs.coeff[order(gibs.coeff$gib.beta.mean, decreasing = T),]
gibs.coeff
```

```
##    gib.beta.mean            name
## 8          0.702       killPlace
## 23         0.418    walkDistance
## 11         0.353     killStreaks
## 3          0.179          boosts
## 14         0.066        maxPlace
## 13         0.052   matchDuration
## 15          0.04       numGroups
## 2          0.029         assists
## 24         0.015 weaponsAcquired
## 6          0.014   headshotKills
## 5          0.012           DBNOs
## 21          0.01       teamKills
## 12         0.005     longestKill
## 18         0.005    rideDistance
## 16         0.004      rankPoints
## 1          0.003       intercept
## 10         0.002           kills
## 17         0.002         revives
## 20         0.001    swimDistance
## 25         0.001       winPoints
## 4              0     damageDealt
## 7              0           heals
## 9              0      killPoints
## 19             0       roadKills
## 22             0 vehicleDestroys
```

```
paste(gibs.coeff$name, collapse = ", ")
```

```
## [1] "killPlace, walkDistance, killStreaks, boosts, maxPlace, matchDuration, numGroups, assists, weap
```

Test Error of Gibbs

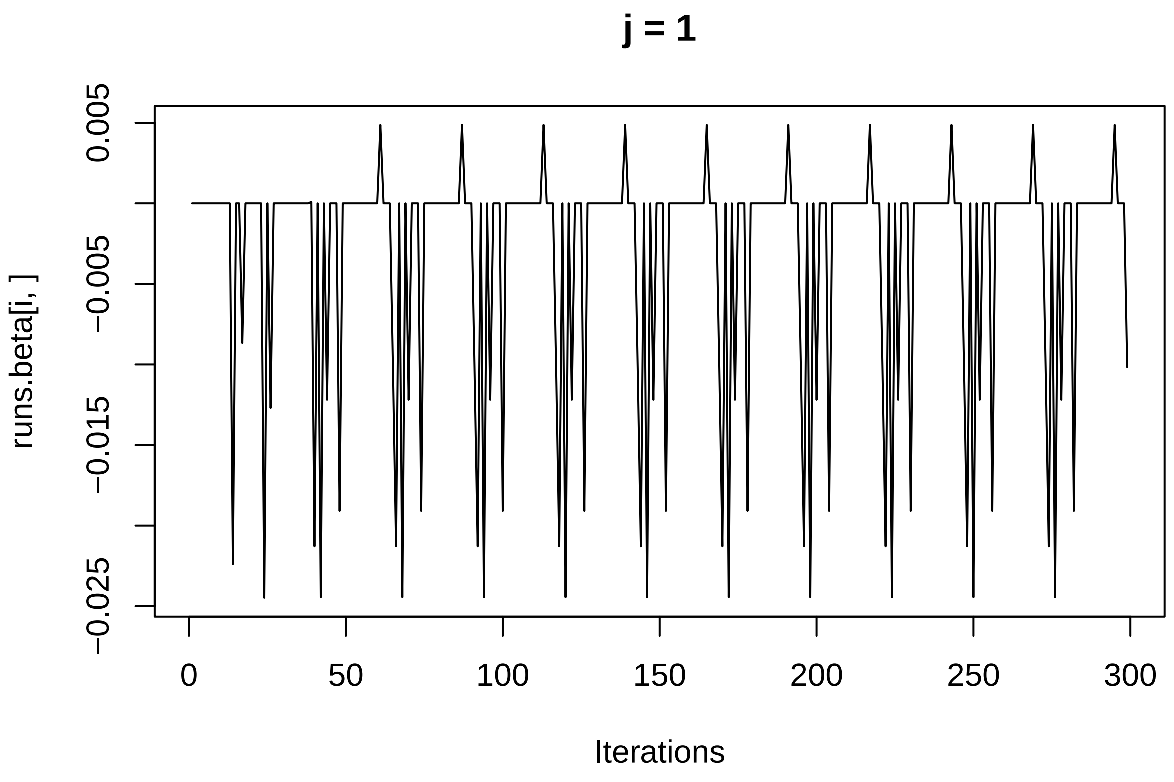
```
test.data = readRDS("split_test_dataset.rds")
X = test.data %>% select(-c(winPlacePerc)) %>% as.matrix()
X = cbind(rep(1,nrow(X)), X)
```

```r
y = test.data$winPlacePerc

pred.Gibbs= X %*%  gib.beta.mean
pred.Gibbs = pnorm(pred.Gibbs)
test.error.gibbs = sum((y-pred.Gibbs)^2);test.error.gibbs
```

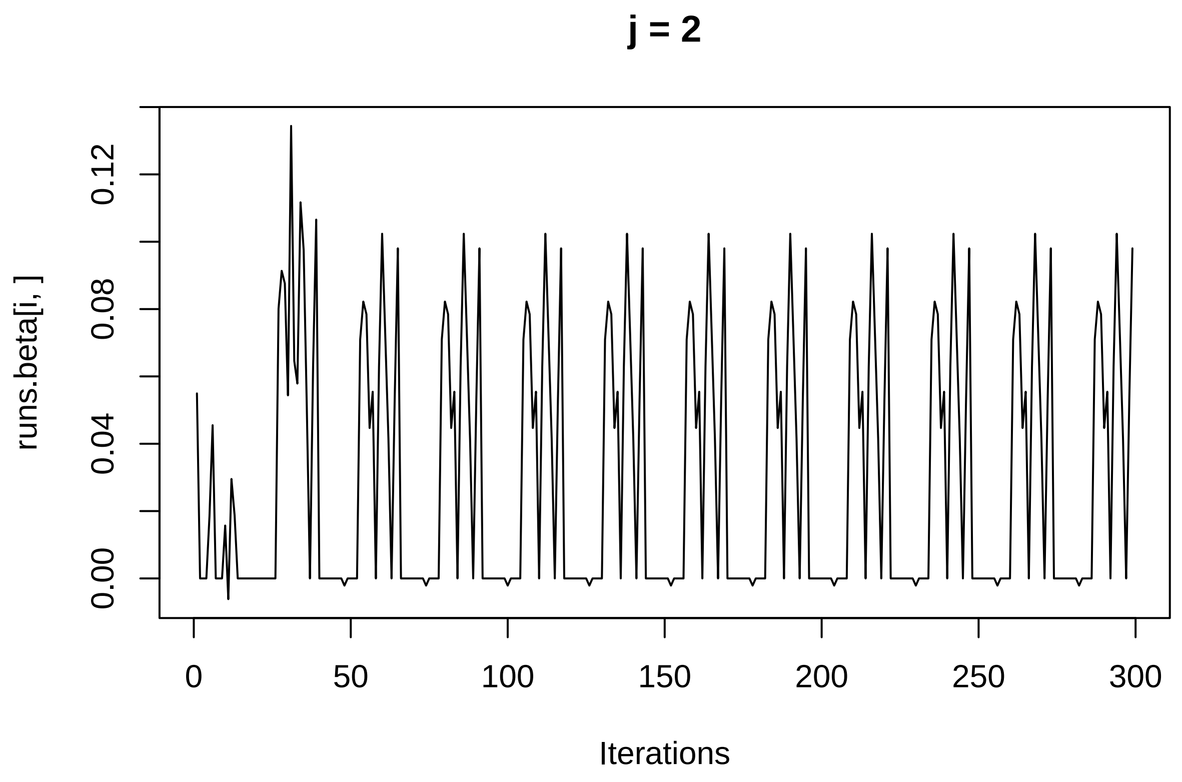
```
## [1] 501.2854
```

Examine Convergence of Gibbs Sequence

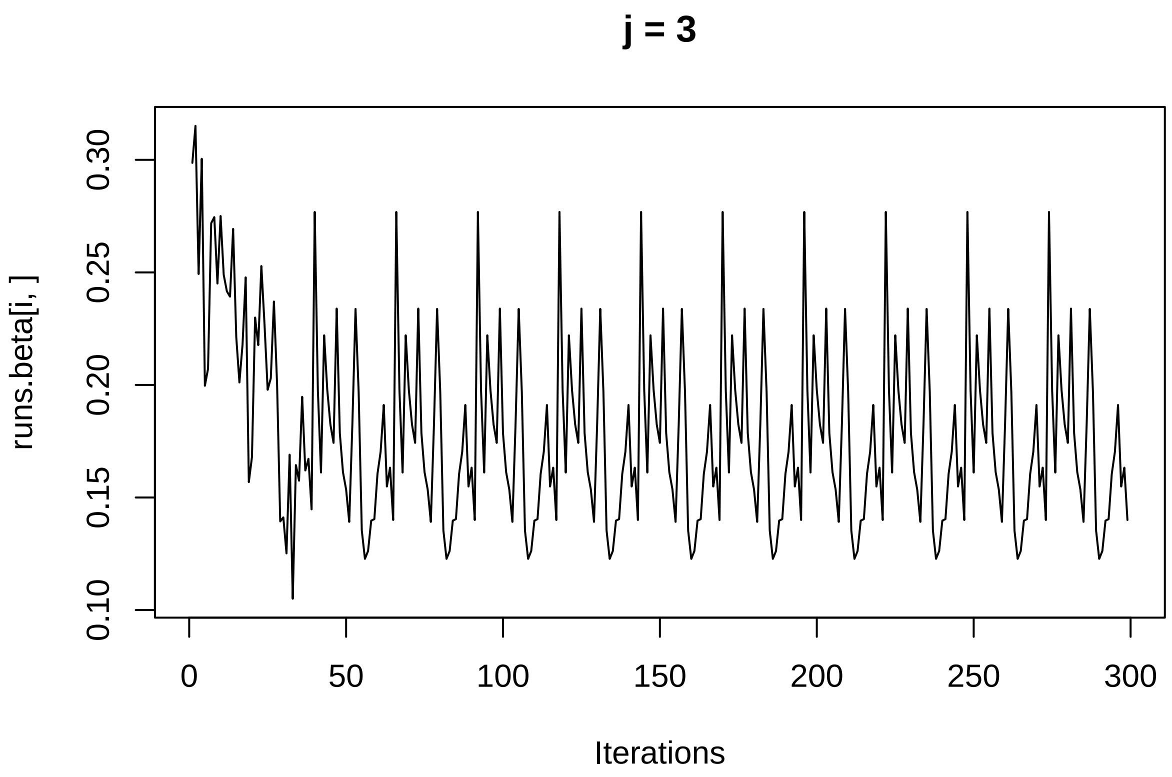
```r
for (i in 1:ncol(X)) {
  plot(1:ncol(runs.beta), runs.beta[i,], 'l', main = sprintf("j = %d", i), xlab = "Iterations")
  print(paste0(i, " ",coda::effectiveSize(runs.beta[i,])))
}
```



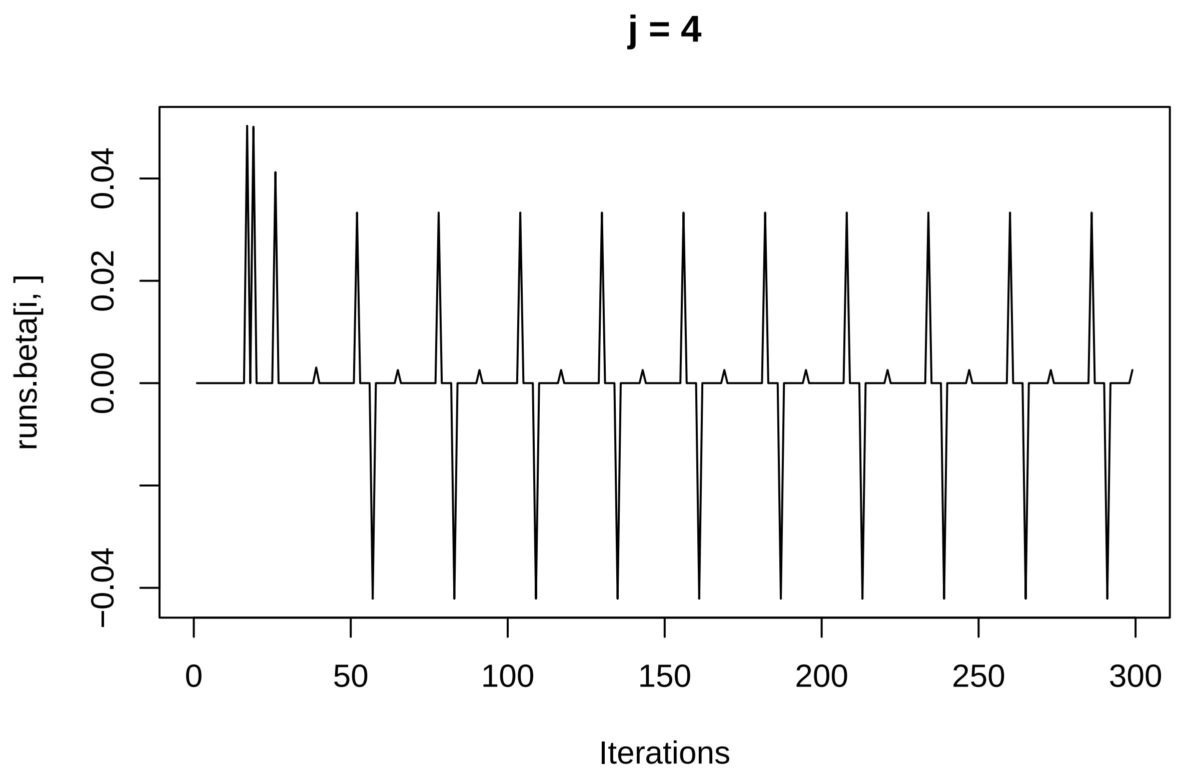
```
## [1] "1 8927.69247015049"
```



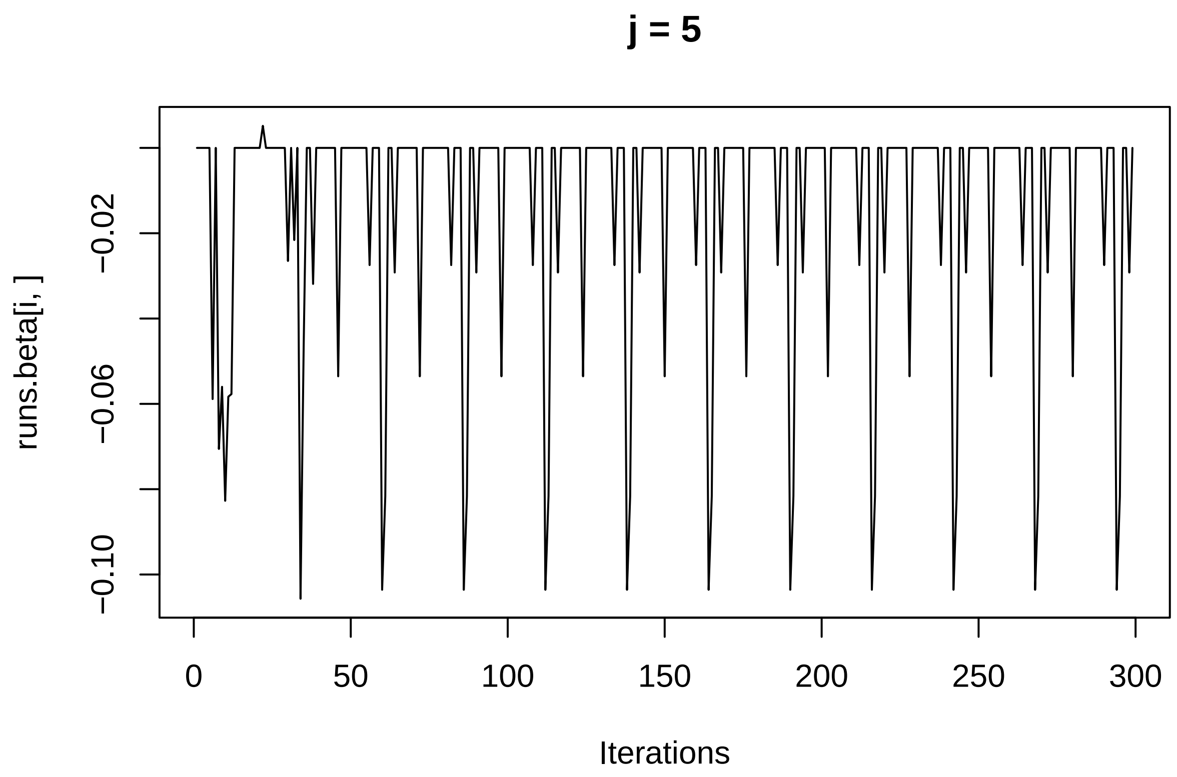
```
## [1] "2 1097.48268831267"
```



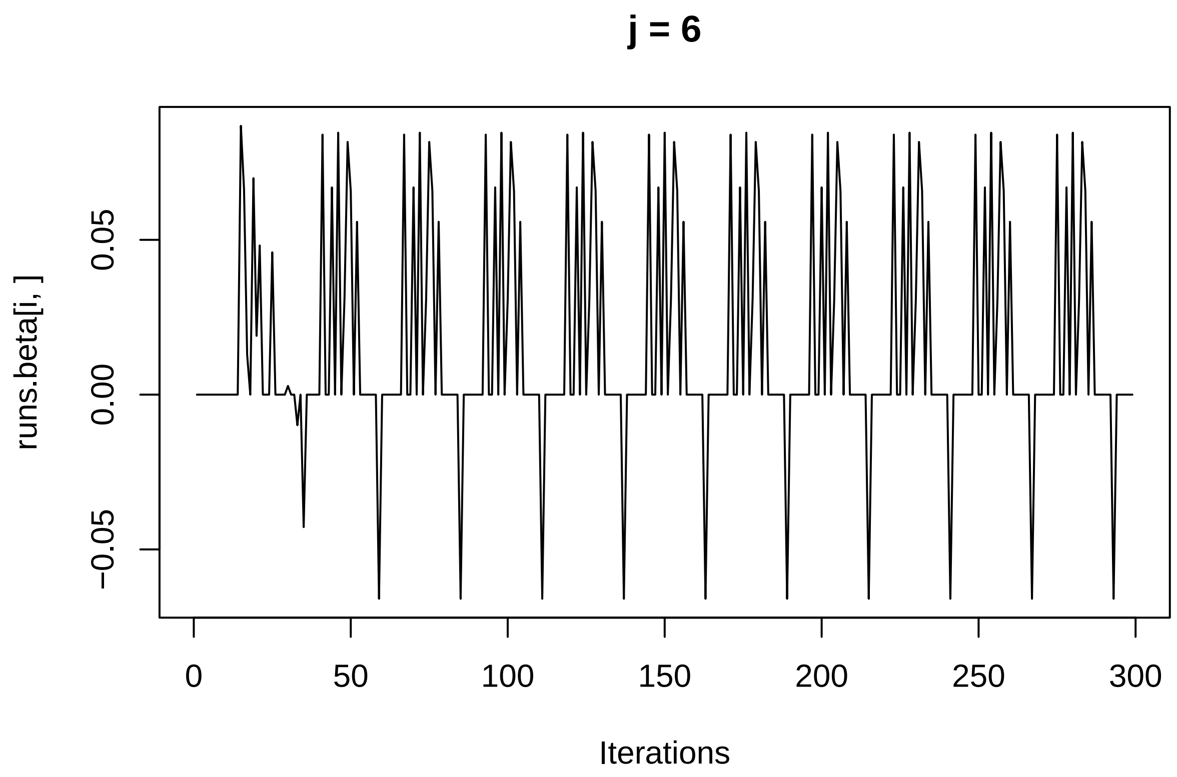
```
## [1] "3 64.640940404304"
```



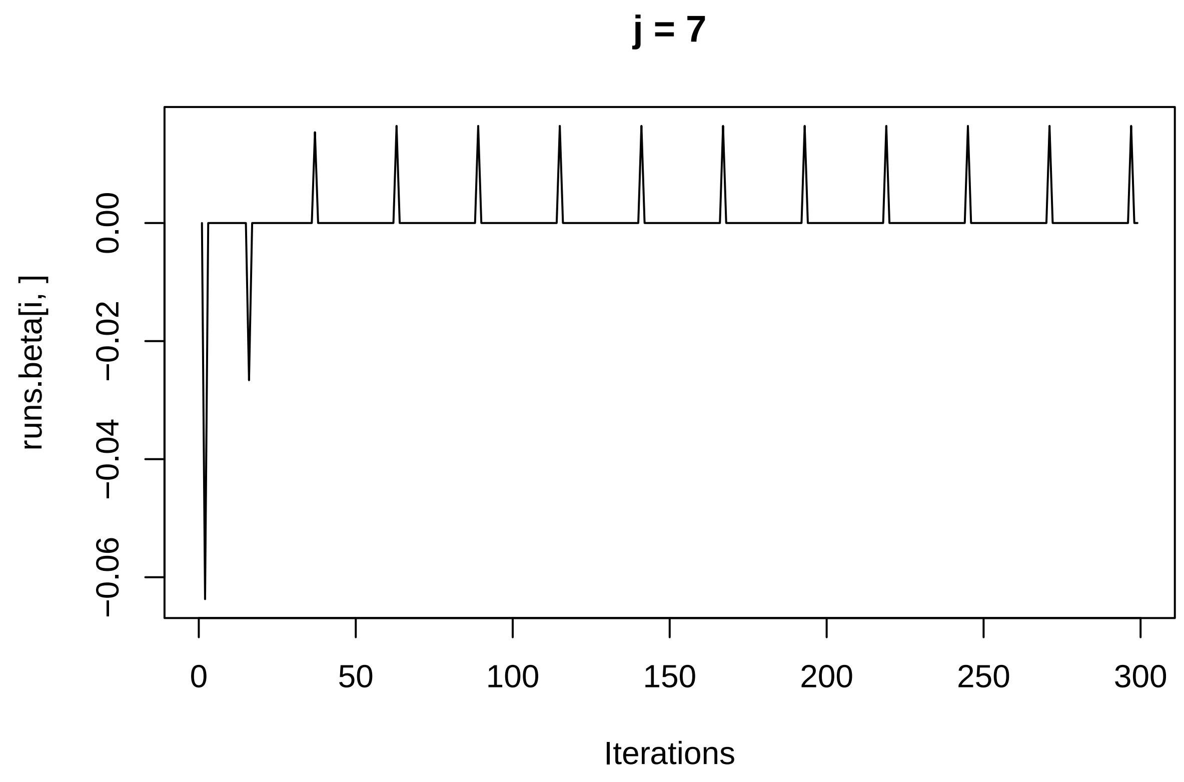
```
## [1] "4 715.919476272831"
```



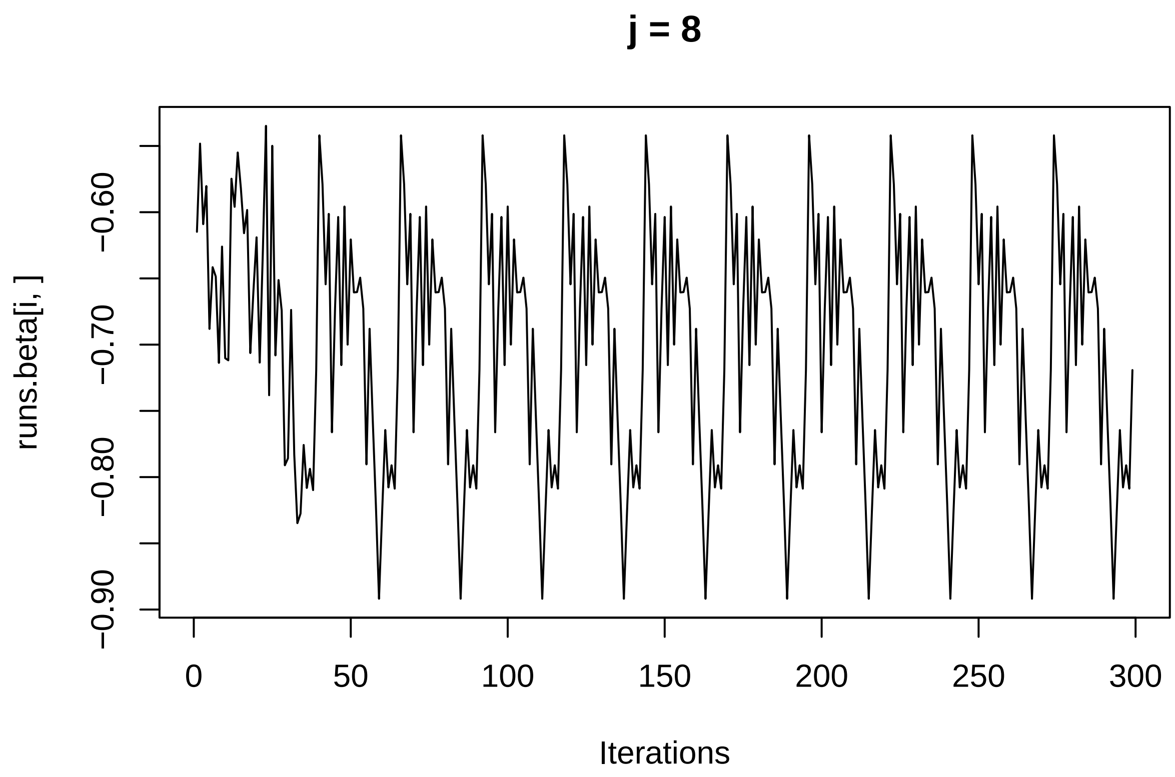
```
## [1] "5 11810.7888405435"
```



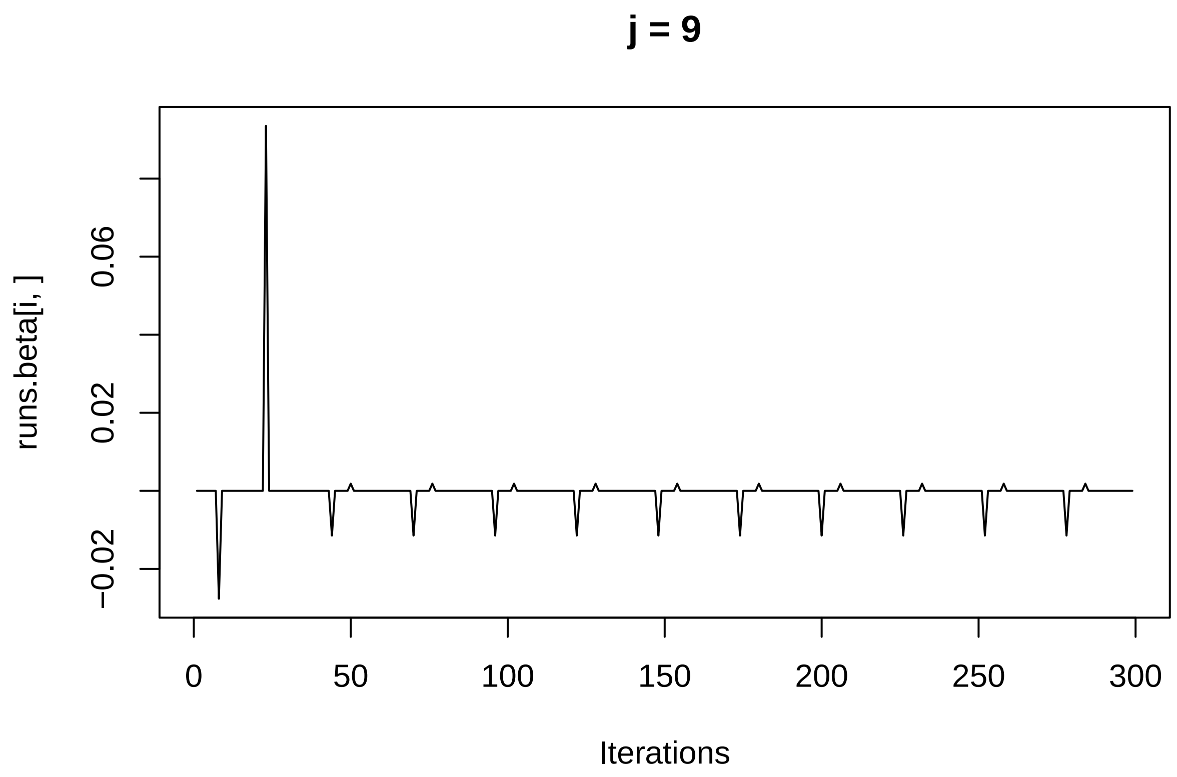
```
## [1] "6 5514.16517050316"
```



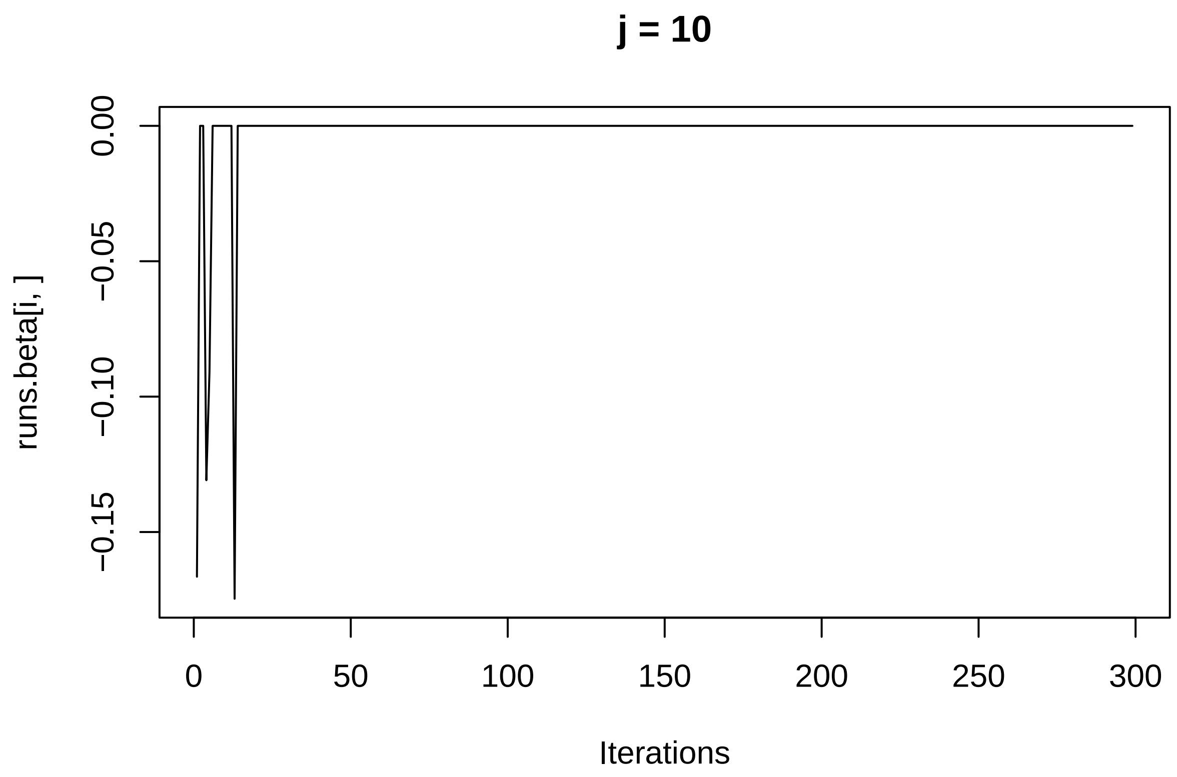
```
## [1] "7 298.999999999998"
```



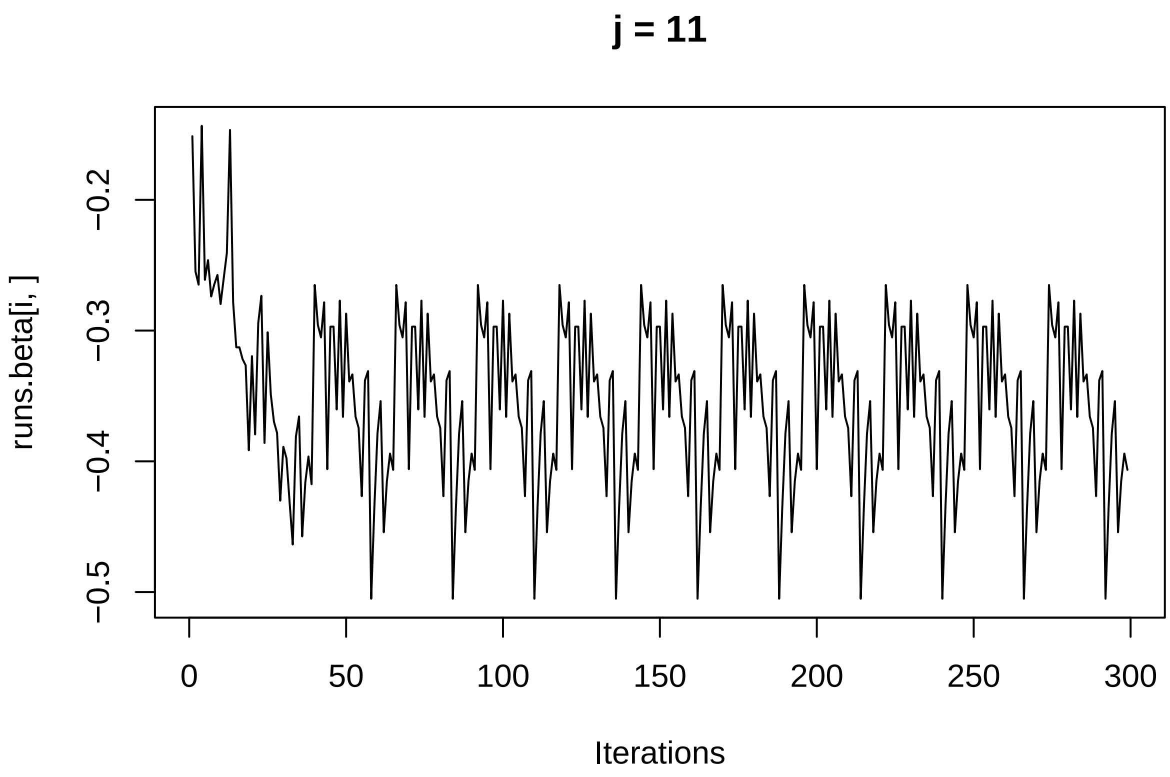
```
## [1] "8 319.872037948086"
```



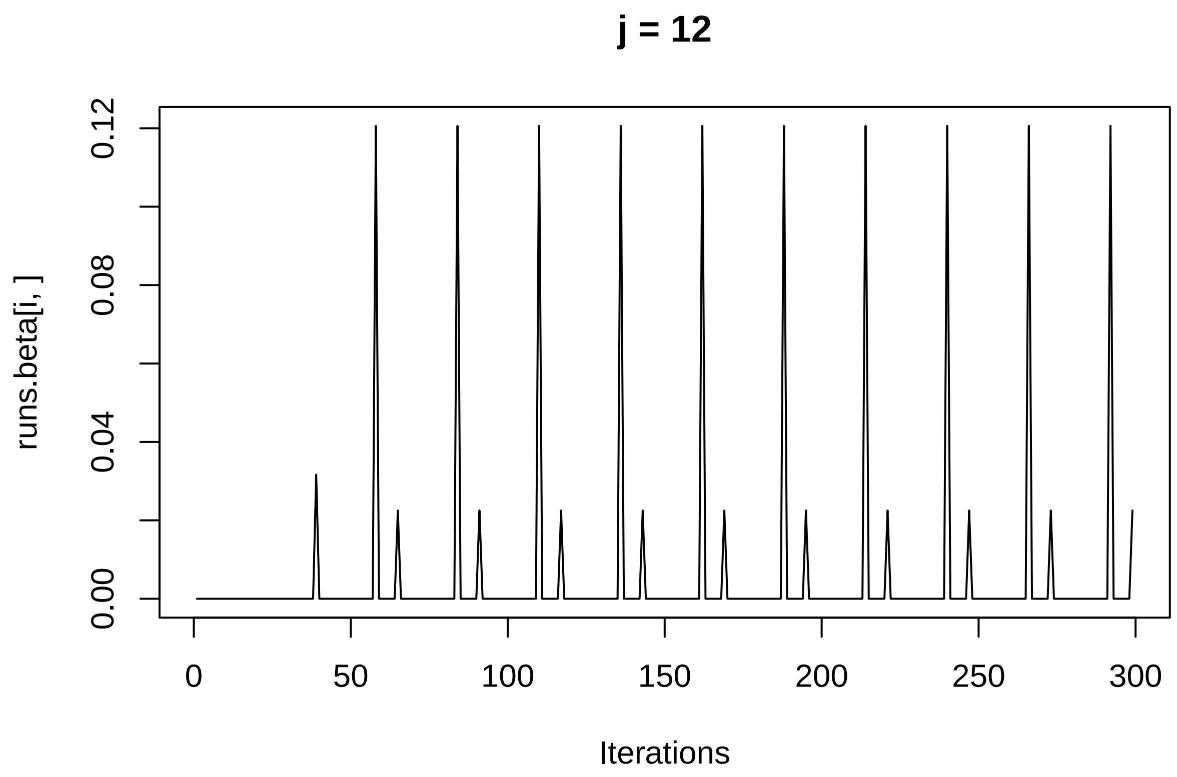
```
## [1] "9 298.999999999997"
```



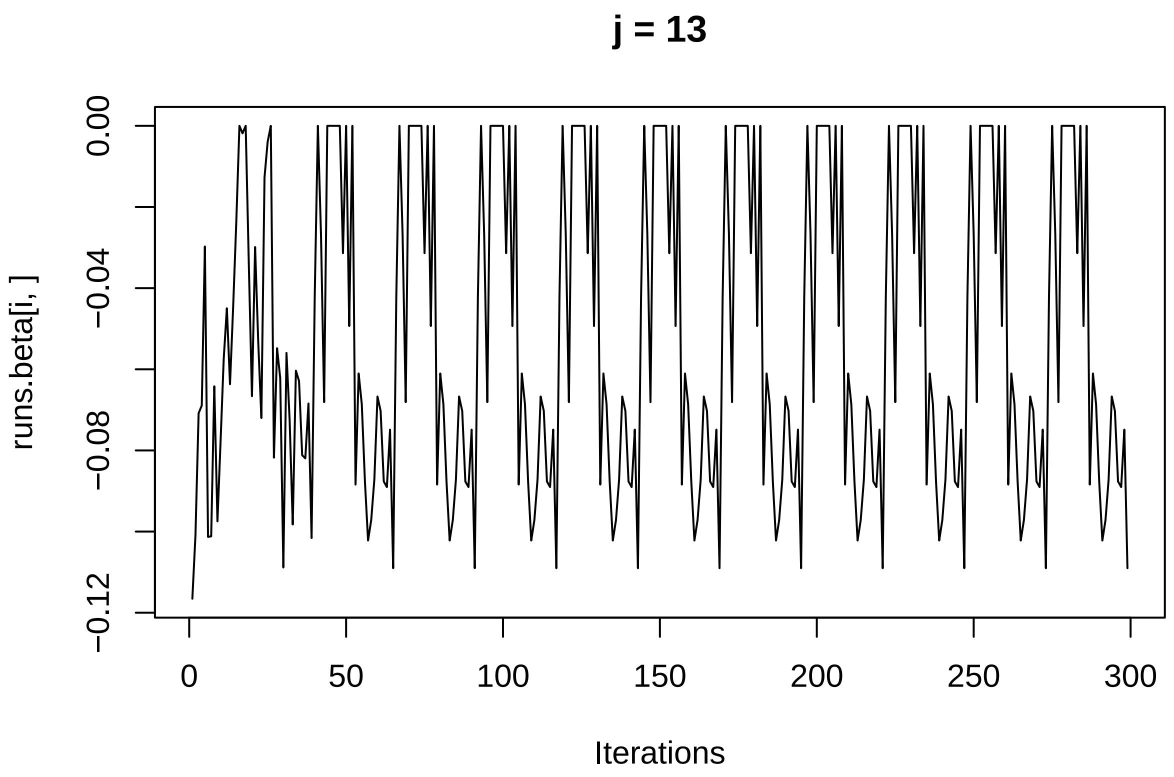
```
## [1] "10 101.822135908116"
```



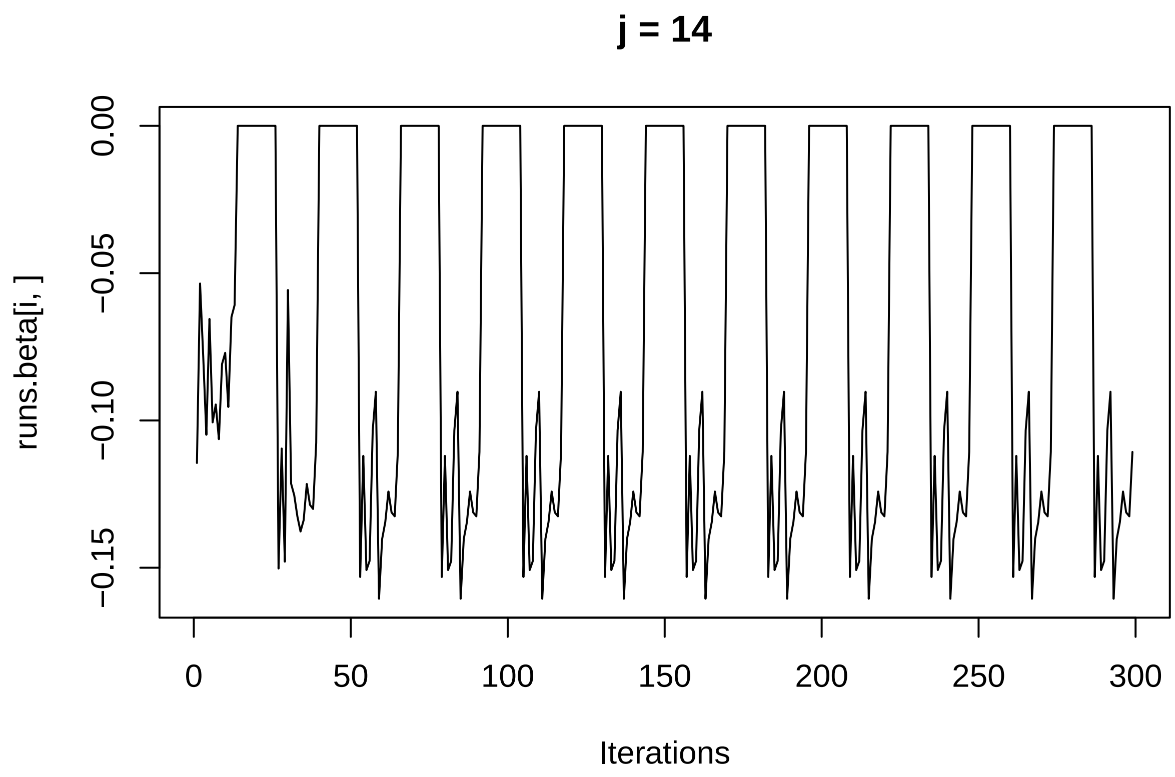
```
## [1] "11 134.180287793539"
```



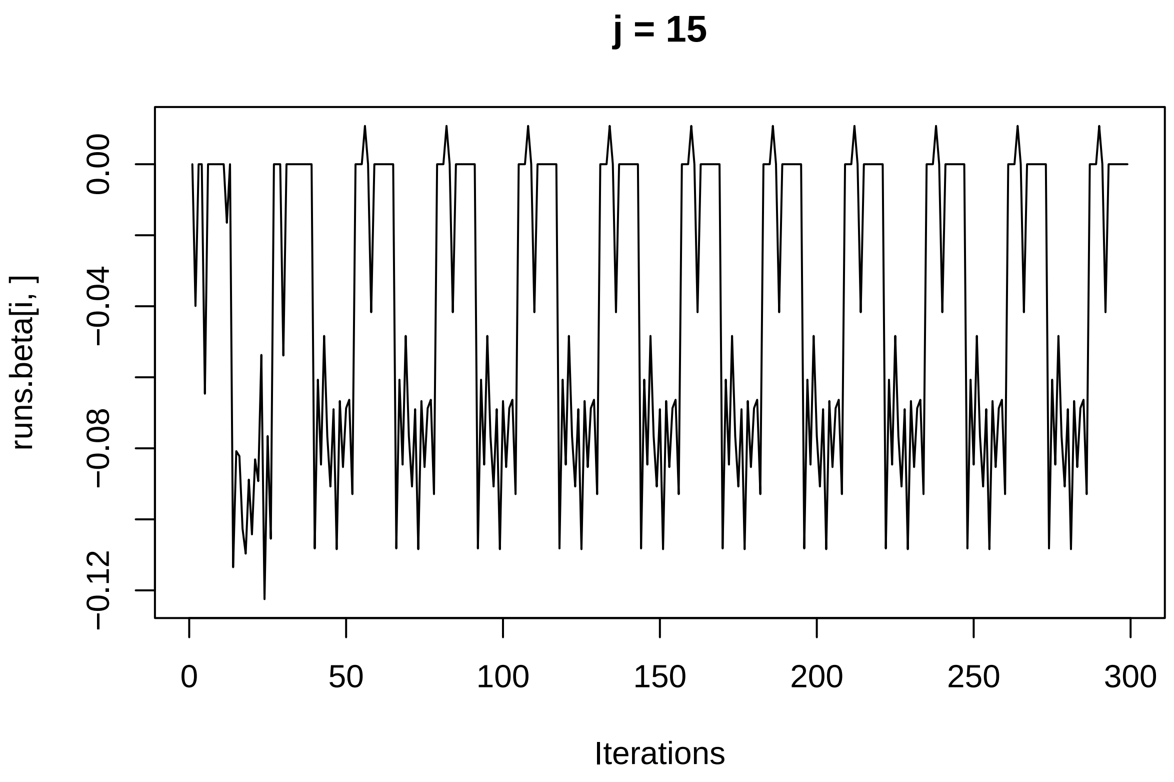
```
## [1] "12 5243.32885497493"
```



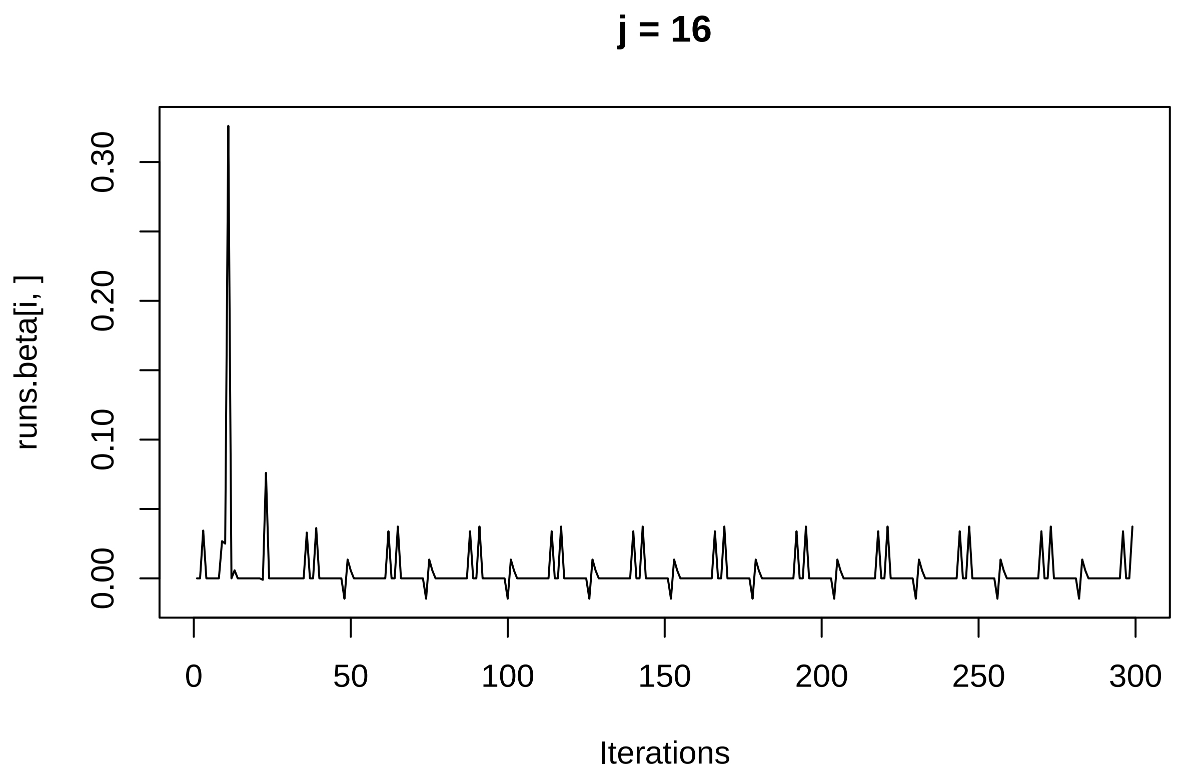
```
## [1] "13 1679.10826885113"
```



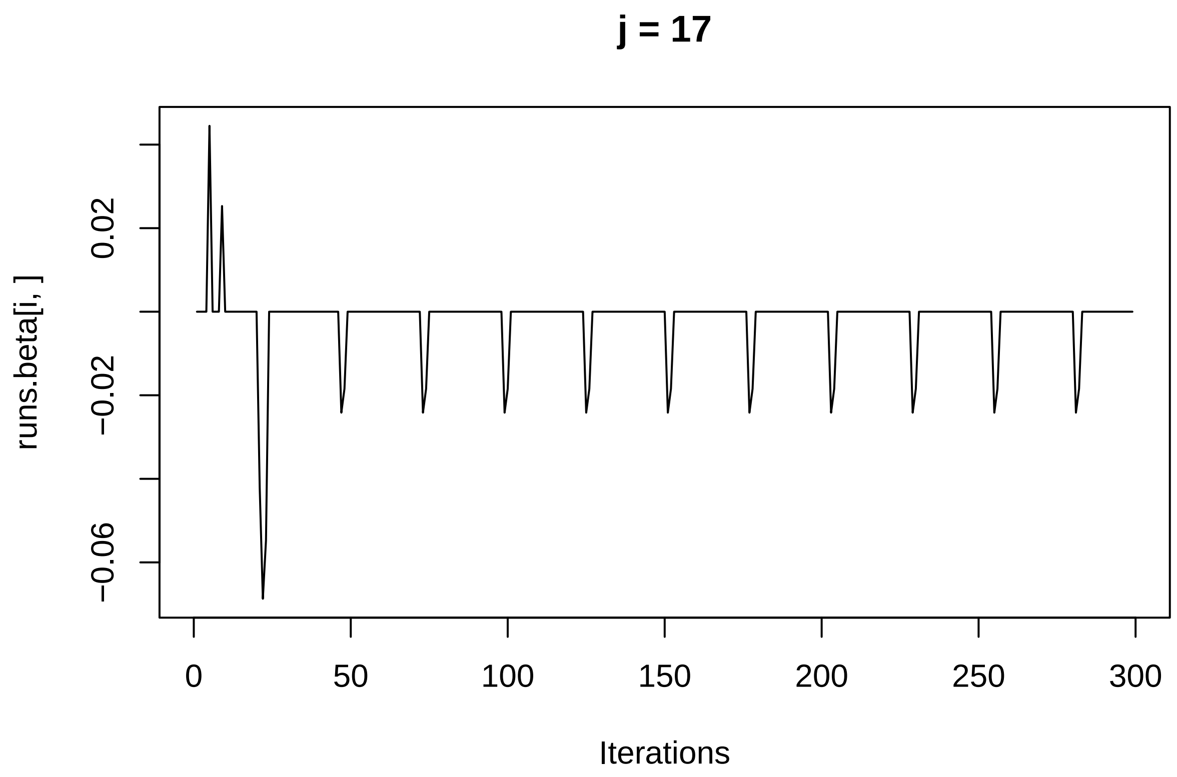
```
## [1] "14 2250.68523353558"
```



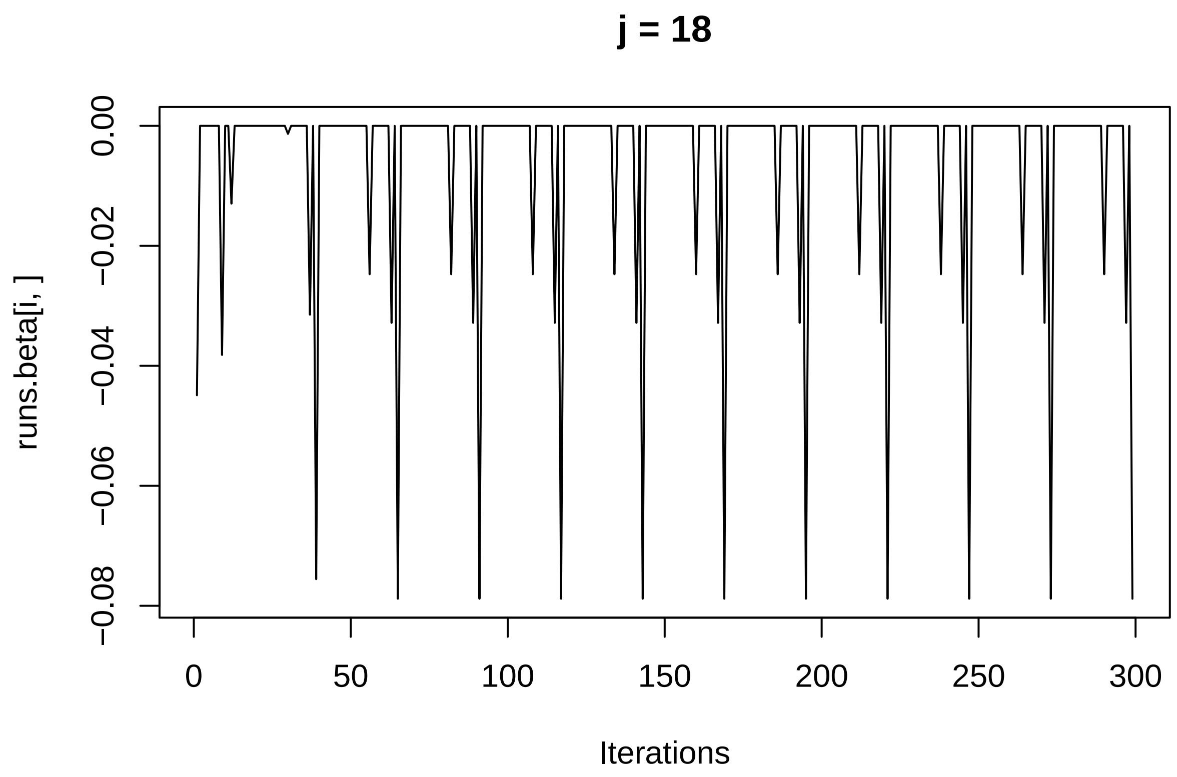
```
## [1] "15 1532.07513209017"
```


j = 16
runs.beta[i,]
0.00
0.10
0.20
0.30
0
50
100
150
200
250
300
Iterations

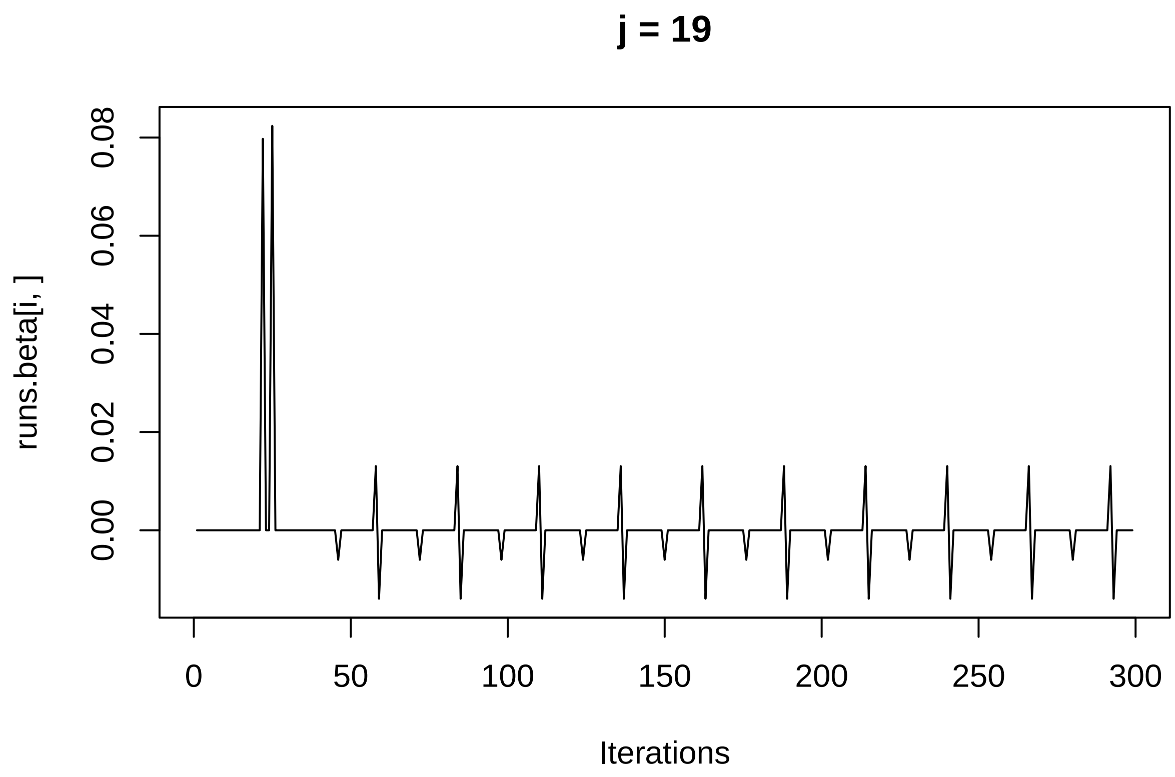
```
## [1] "16 299"
```



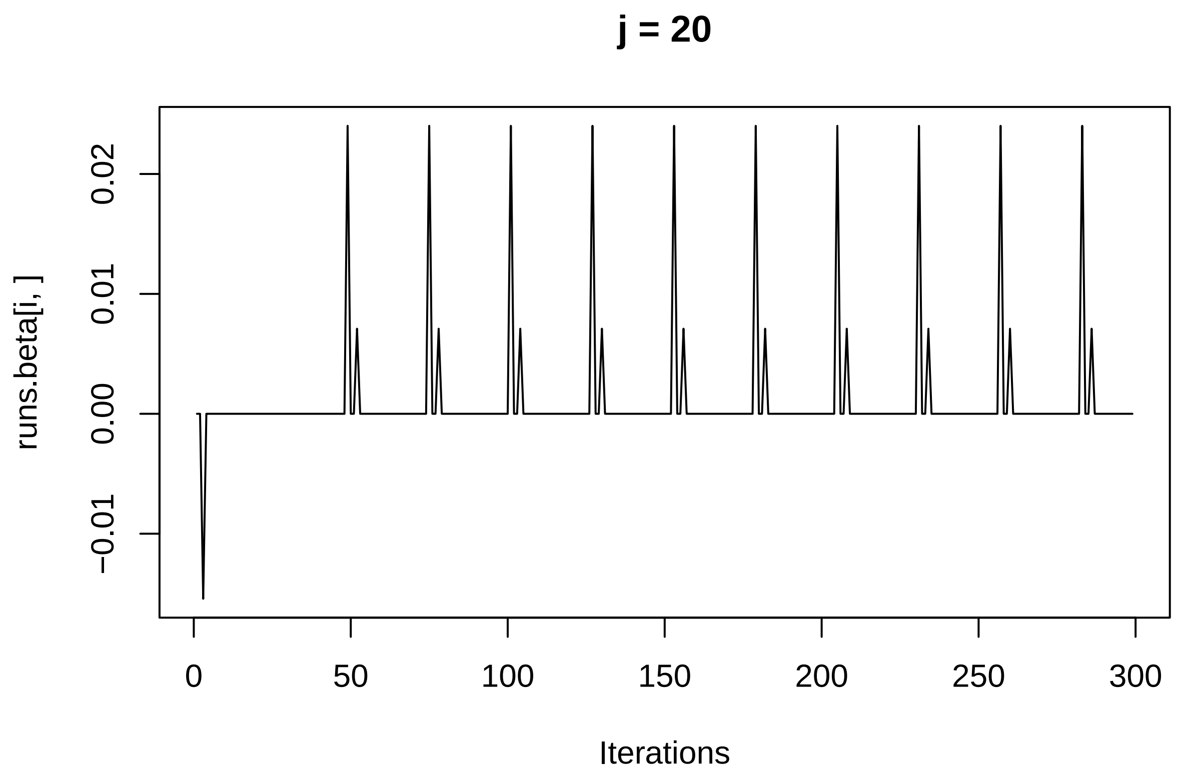
```
## [1] "17 172.913323767868"
```



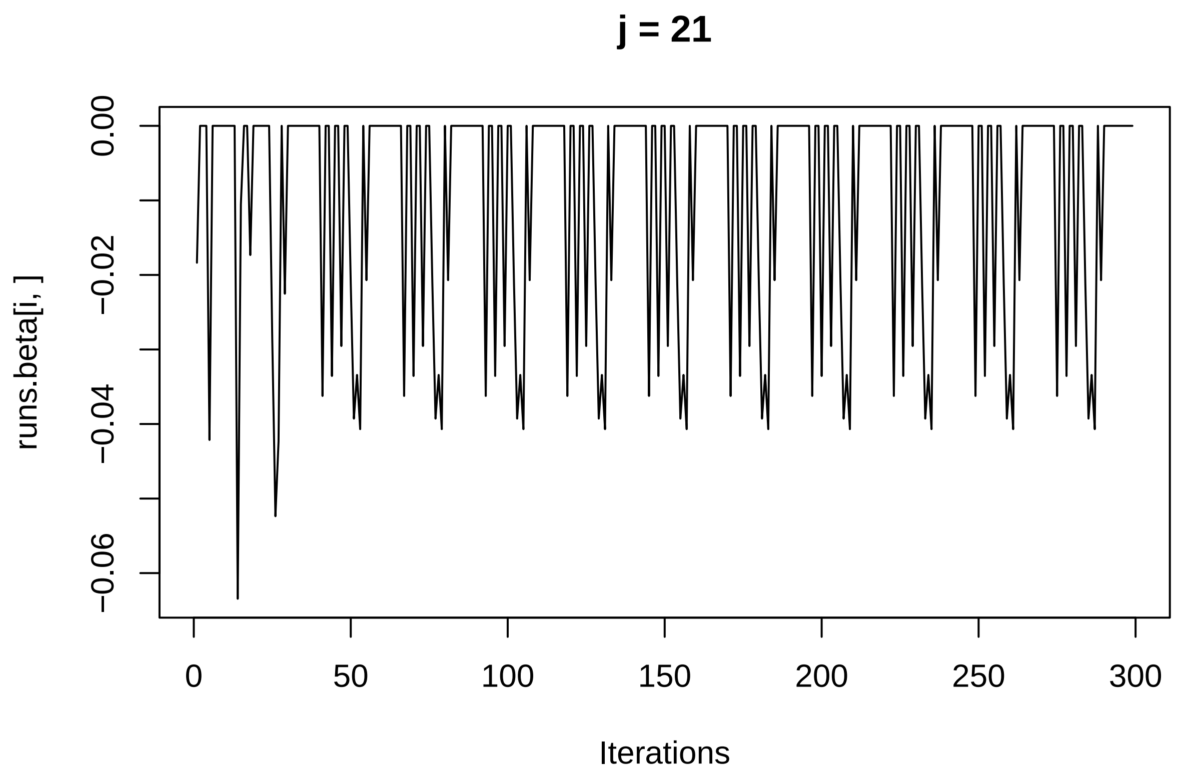
```
## [1] "18 9292.57072677952"
```



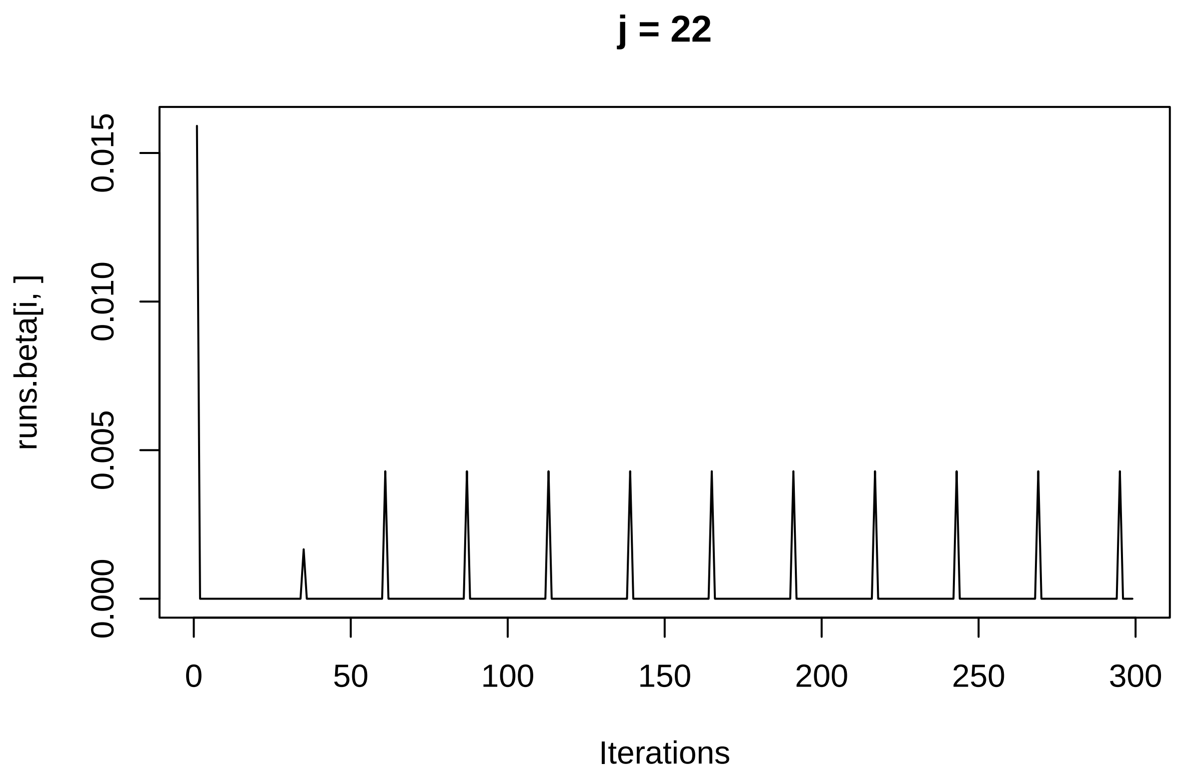
```
## [1] "19 199.380311640869"
```



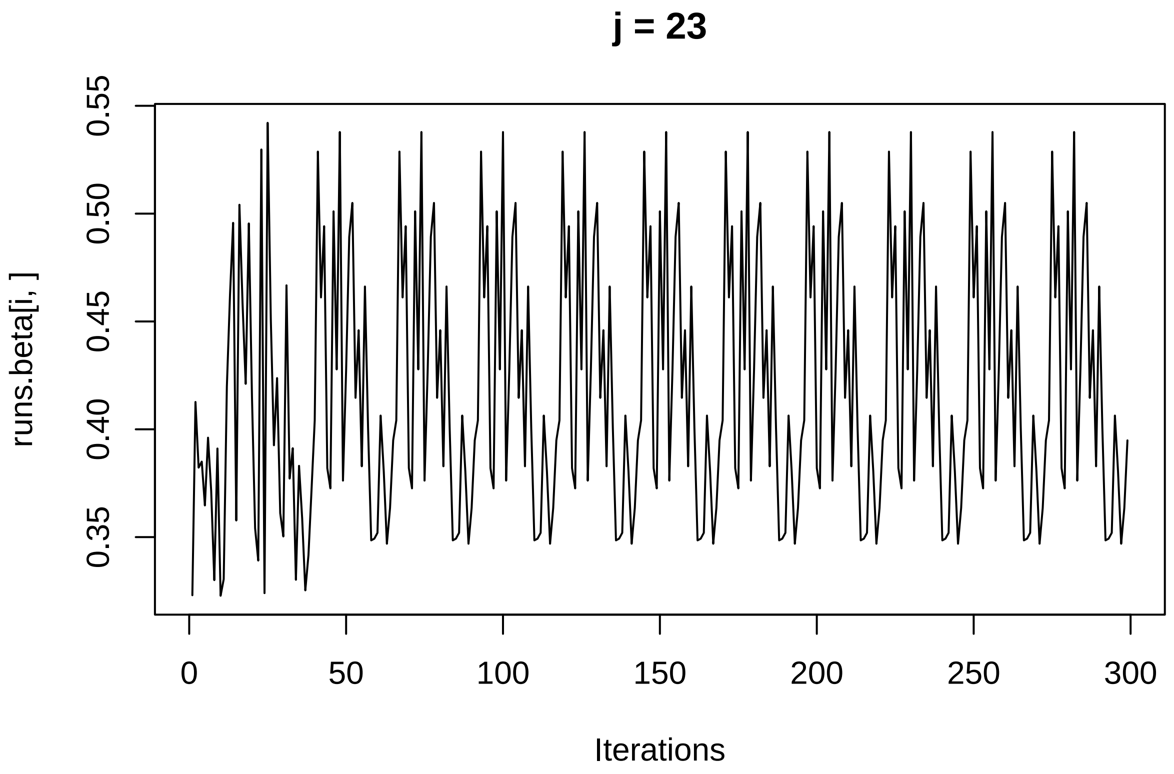
```
## [1] "20 229.482308408579"
```



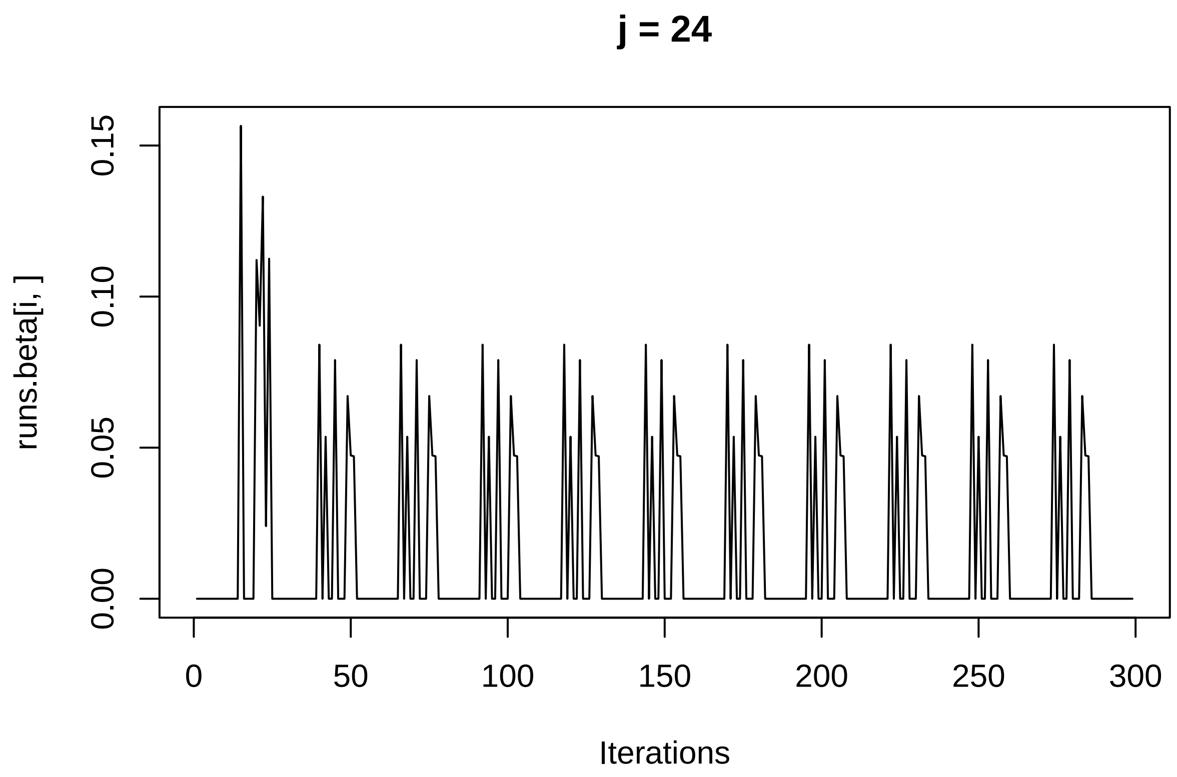
```
## [1] "21 23075.4605192301"
```



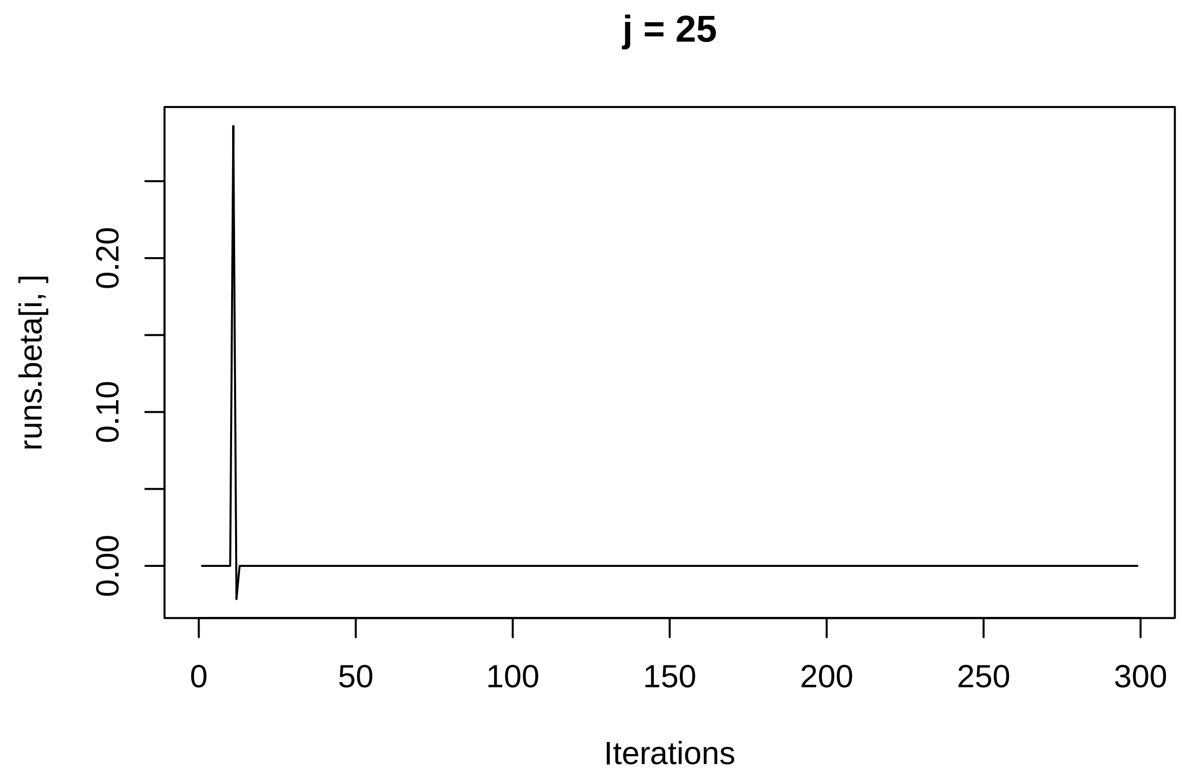
```
## [1] "22 298.999999999996"
```



```
## [1] "23 1495.47138428653"
```



```
## [1] "24 2023.31375237052"
```



```
## [1] "25 299.000000000001"
```

#MH algorithm

```
set.seed(5511)
y= as.matrix(tdata$winPlacePerc)
one.index = which(y==1)
second.highest = sort(unique(y), decreasing = T)[2]
remake.one = runif(length(one.index), min = second.highest+0.0001,max= 1-0.0001)
y[one.index] = remake.one
zero.index = which(y==0)
second.lowest = sort(unique(y))[2]
remake.zero = runif(length(zero.index), min = 0+0.0001,max= second.lowest-0.0001)
y[zero.index] = remake.zero
y=qnorm(y)

p = ncol(tdata)
z = as.matrix(rep(1,p+1))
X = tdata %>% select(-c(winPlacePerc)) %>% as.matrix
X = cbind(intercept = rep(1, nrow(X)), X) %>% as.matrix


ptm <- proc.time()
sigma2 = var(y)
beta.now = c(rnorm(p,0,2))
```

```r
gamma.now = rep(1,p)
sigma2.now = 1/rgamma(1, shape = 1, rate = 1*var(y))
sigma2.sim = vector()
count.sigma2 = 0

S = 20
gammas.sim = rep(" ", S)
gamma.sim = matrix(0, ncol = S, nrow=p)
beta.sim = matrix(0, ncol = S, nrow=p)
beta.gamma.sim = matrix(0, ncol = S, nrow=p)
count.beta = 0
count = 0
for(s in 1:S){
  if(s%%(S/10) == 1){print(s)}

  #update gamma
  theta.now = X[, gamma.now==1, drop=F] %*% beta.now[gamma.now==1]
  ln.likelihood.now = sum(dnorm(y, theta.now , sqrt(sigma2.now), log = T))
    # - sum((y-theta.now)^2) / (2*sigma2.now)
  gamma.new = gamma.now
  index = seq(1, p)
  sam.index = sample(index, size=1)
  gamma.new[sam.index] = 1-gamma.new[sam.index]

  theta.new = X[, gamma.new==1, drop=F] %*% beta.now[gamma.new==1]
  ln.likelihood.new = sum(dnorm(y, theta.new , sqrt(sigma2.now), log = T))
  # log(2*pi*sigma2.now) - sum((y-theta.new)^2) / (2*sigma2.now)
  denom = ln.likelihood.now
  num = ln.likelihood.new

  if(runif(1)<exp(num - denom) ){
    count = count +1
    gamma.now = gamma.new
  }

  gammas.sim[s] = paste(gamma.now,collapse=' ')
  gamma.sim[,s] = gamma.now

  #update beta
  prior.prob.now = sum(log(dnorm(beta.now,0,2)))
  theta.now = X[, gamma.now==1, drop=F] %*% beta.now[gamma.now==1]
  ln.likelihood.now = sum(dnorm(y, theta.now , sqrt(sigma2.now), log = T))
    # - sum((y-theta.now)^2) / (2*sigma2.now)

  beta.new = mvtnorm::rmvnorm(1, mean = beta.now, sigma = sigma2.now*solve(t(X)%*%X))
  theta.new = X[, gamma.now==1, drop=F] %*% beta.new[gamma.now==1]
  ln.likelihood.new = sum(dnorm(y, theta.new , sqrt(sigma2.now), log = T))
    # - sum((y-theta.new)^2) / (2*sigma2.now)
  prior.prob.new = sum(log(dnorm(beta.new,0,2)))

  denom = ln.likelihood.now + prior.prob.now
  num = ln.likelihood.new + prior.prob.new
  if(runif(1)<exp(num - denom) ){
    count.beta = count.beta +1
    beta.now = beta.new
    theta.now = theta.new
  }
  beta.sim[,s] = beta.now

  betaWIthGamma = rep(0,p)
  betaWIthGamma[which(gamma.now==1)] = beta.now[which(gamma.now==1)]
  beta.gamma.sim[,s] = betaWIthGamma

  #update sigma
  SSRg = sum((y-theta.now)^2)
  prior.prob.sigma2.now = log(dgamma(1/sigma2.now ,shape = 1, rate = var(y)))
  ln.likelihood.now = sum(dnorm(y, theta.now , sqrt(sigma2.now), log = T))
    # - sum((y-theta.now)^2) / (2*sigma2.now)

  sigma2.new = 1/rgamma(1, shape = 1, rate = sigma2.now )
  prior.prob.sigma2.new = log(dgamma(1/sigma2.new ,shape = 1, rate = var(y)))
  ln.likelihood.new = sum(dnorm(y, theta.now , sqrt(sigma2.new), log = T))
    # - sum((y-theta.now)^2) / (2*sigma2.new)
  denom = ln.likelihood.now + prior.prob.sigma2.now + log(dgamma(1/sigma2.now, shape = 1, rate = sigma2
  num = ln.likelihood.new + prior.prob.sigma2.new + log(dgamma(1/sigma2.new, shape = 1, rate = sigma2.n
  if(runif(1)<exp(num - denom) ){
    count.sigma2 = count.sigma2 +1
    sigma2.now = sigma2.new
  }
  sigma2.sim[s] = sigma2.now
}
```

```
## [1] 1
## [1] 3
## [1] 5
## [1] 7
## [1] 9
## [1] 11
## [1] 13
## [1] 15
## [1] 17
## [1] 19
```

```
proc.time() - ptm
```

```
##
## 0.63 0.08 1.91
```

```
# 0.052928 sec / iteration
print(count / S)
```

```
## [1] 0.4
```

```
print(count.beta / S)
```

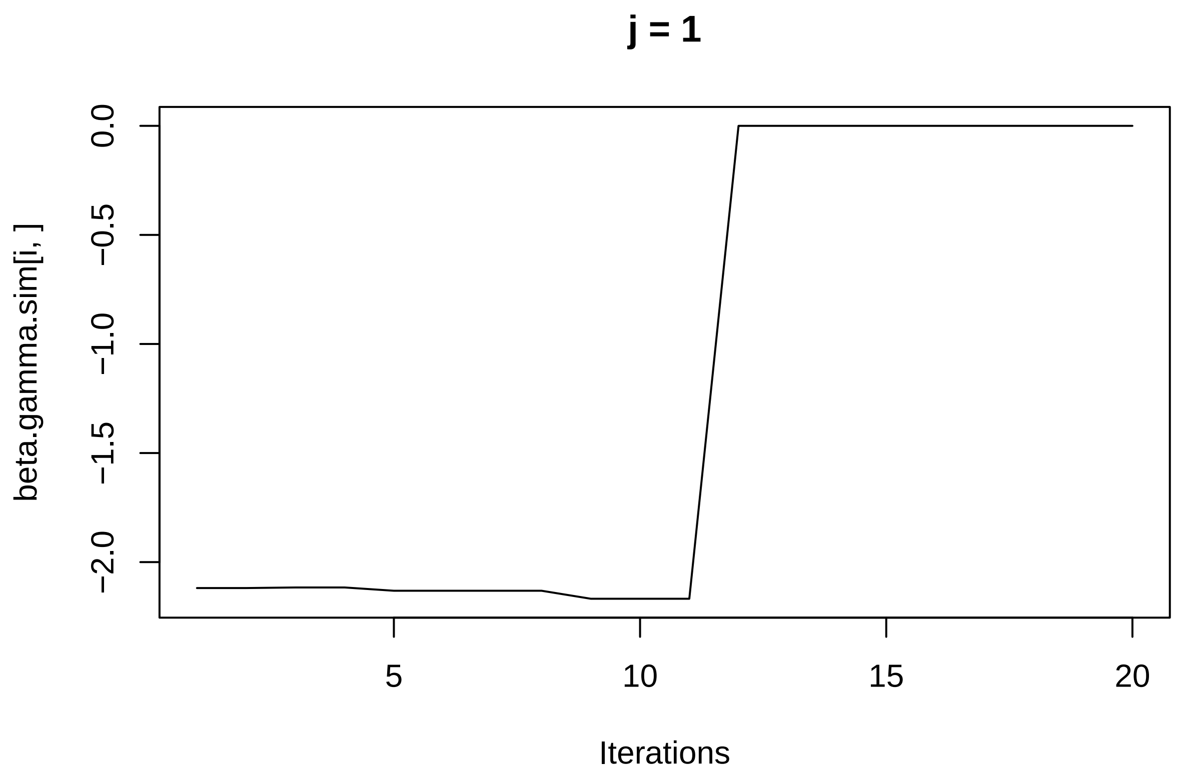
```
## [1] 0.35
```

```
print(count.sigma2 / S)
```

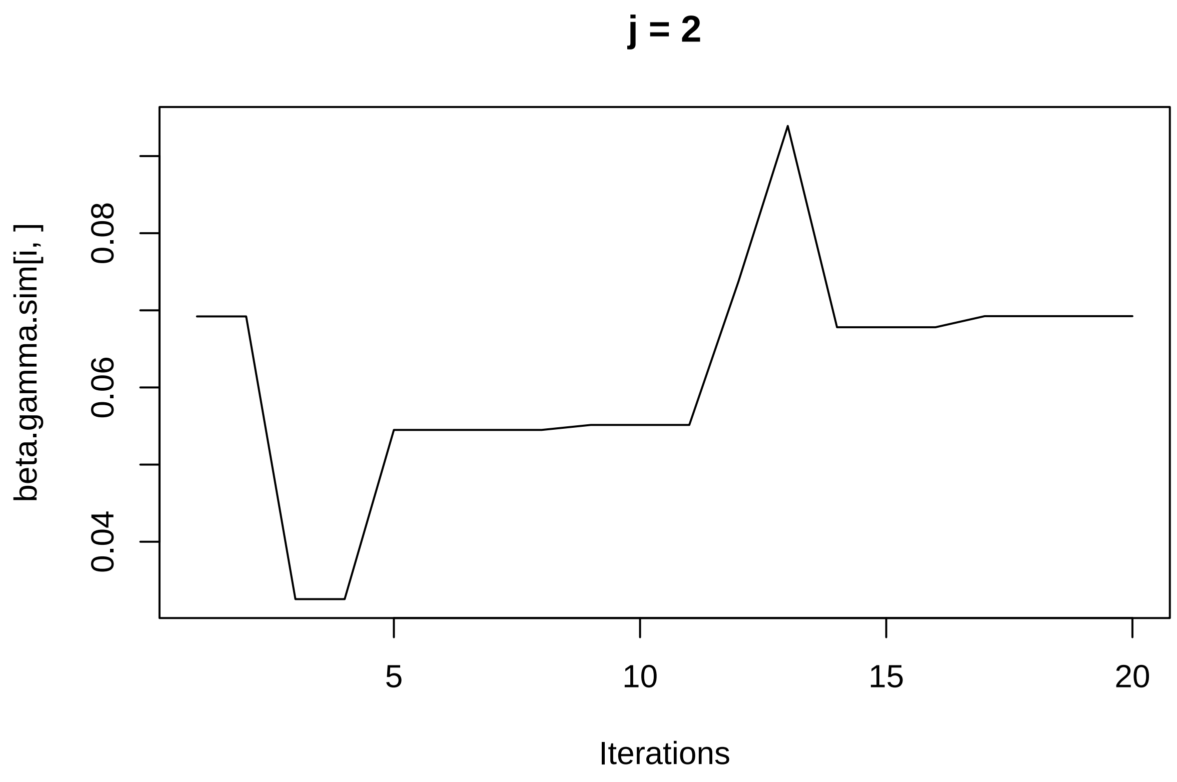
```
## [1] 0.35
```

```
#Second chain
set.seed(5512)
p = ncol(tdata)
X = tdata %>% select(-c(winPlacePerc)) %>% as.matrix
X = cbind(rep(1,nrow(X)), X) %>% as.matrix
y= as.matrix(tdata$winPlacePerc)

ptm <- proc.time()
sigma2 = var(y)
beta.now = c(rnorm(p,0,2)+3)
gamma.now = rep(1,p)
S = 33000
gammas.sim2 = rep(" ", S)
gamma.sim2 = matrix(0, ncol = S, nrow=p)
# beta.sim2 = matrix(0, ncol = S, nrow=p)
beta.gamma.sim2 = matrix(0, ncol = S, nrow=p)
count.beta = 0
count = 0
for(s in 1:S){
  if(s%%(S/10) == 0){print(s)}
  #update gamma
  theta.now = X[, gamma.now==1, drop=F] %*% beta.now[gamma.now==1]

  ln.likelihood.now = - sum((y-theta.now)^2) / (2*sigma2)

  # now.prob = 0.045
  # gamma.new = c(rbinom(n=p,size=1,prob = now.prob))

  gamma.new = gamma.now
  index = seq(1, p)
  sam.index = sample(index, size=1)
  gamma.new[sam.index] = 1-gamma.new[sam.index]

  theta.new = X[, gamma.new==1, drop=F] %*% beta.now[gamma.new==1]
  ln.likelihood.new = - sum((y-theta.new)^2) / (2*sigma2)

  denom = ln.likelihood.now
  num = ln.likelihood.new

  if(runif(1)<exp(num - denom) ){
    count = count +1
    gamma.now = gamma.new
  }
  gammas.sim2[s] = paste(gamma.now,collapse=' ')
  gamma.sim2[,s] = gamma.now

  #update beta
  # X.now = X[, gamma.now==1, drop=F]

  prior.prob.now = sum(log(dnorm(beta.now,0,2)))
  theta.now = X[, gamma.now==1, drop=F] %*% beta.now[gamma.now==1]
  ln.likelihood.now = - sum((y-theta.now)^2) / (2*sigma2)

  beta.new = mvtnorm::rmvnorm(1, mean = beta.now, sigma = as.numeric(var(y))*solve(t(X)%*%X))
  theta.new = X[, gamma.now==1, drop=F] %*% beta.new[gamma.now==1]
  ln.likelihood.new = - sum((y-theta.new)^2) / (2*sigma2)
  prior.prob.new = sum(log(dnorm(beta.new,0,2)))

  denom = ln.likelihood.now + prior.prob.now
  num = ln.likelihood.new + prior.prob.new

  if(runif(1)<exp(num - denom) ){
    count.beta = count.beta +1
    beta.now = beta.new
  }
  # beta.sim[,s] = beta.now

  betaWIthGamma = rep(0,p)
  betaWIthGamma[which(gamma.now==1)] = beta.now[which(gamma.now==1)]
  beta.gamma.sim2[,s] = betaWIthGamma
}
proc.time() - ptm
# 0.052928 sec / iteration
print(count / S)
print(count.beta / S)
```

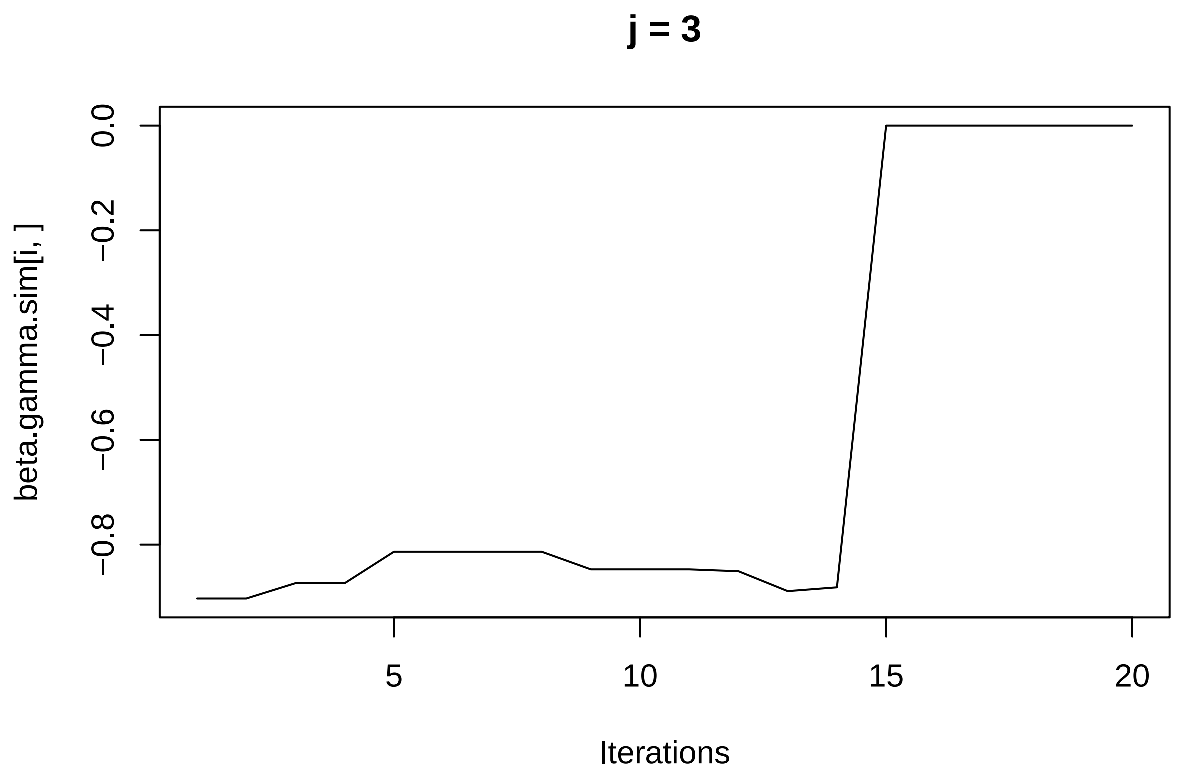
```
for (i in 1:ncol(X)) {
  plot(1:S, beta.gamma.sim[i,], 'l', main = sprintf("j = %d", i), xlab = "Iterations")
  print(paste0(i, " ",coda::effectiveSize(beta.gamma.sim[i,])))
  # print(mean(beta.gamma.sim[i,]))
}
```



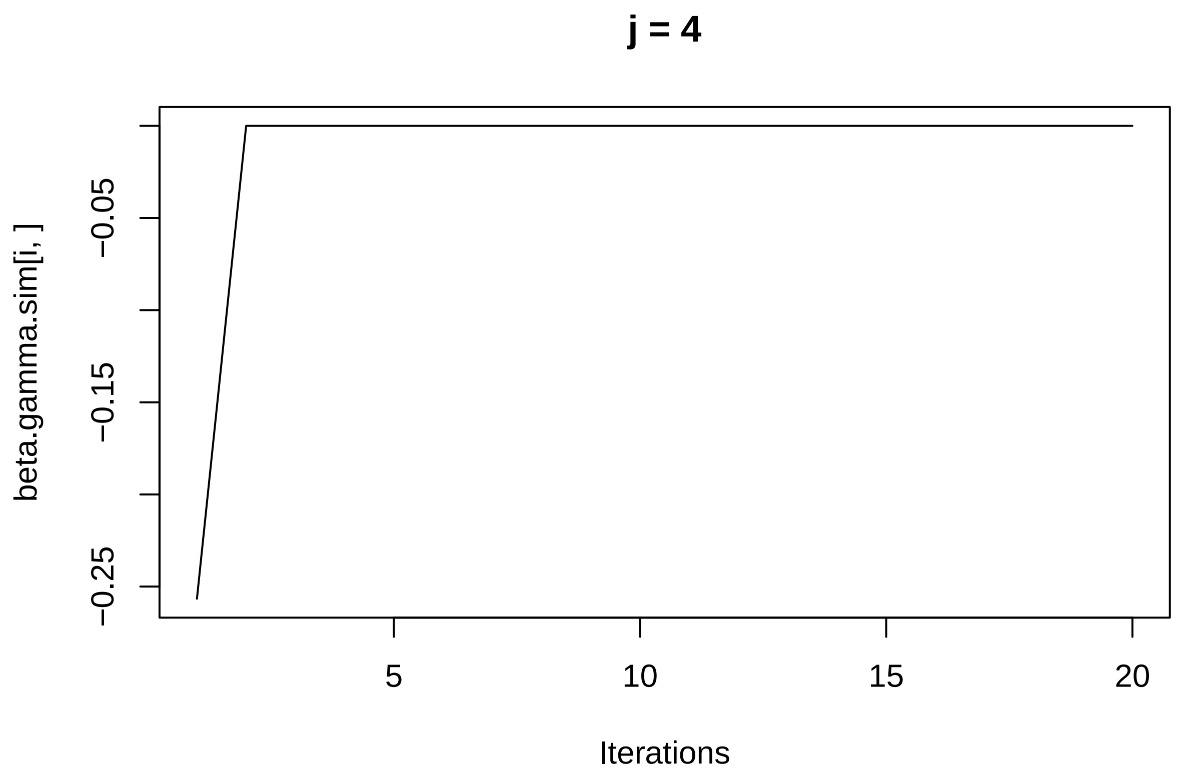
```
## [1] "1 1.58405262641913"
```



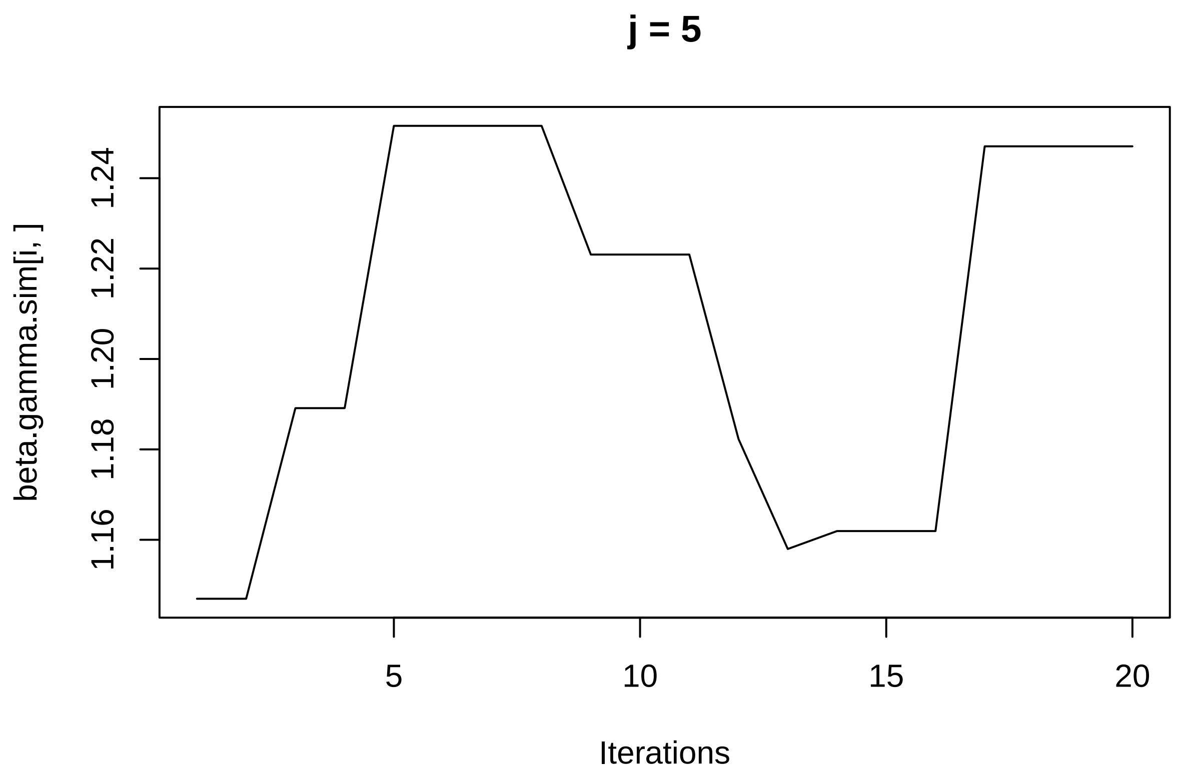
```
## [1] "2 5.65111987846014"
```



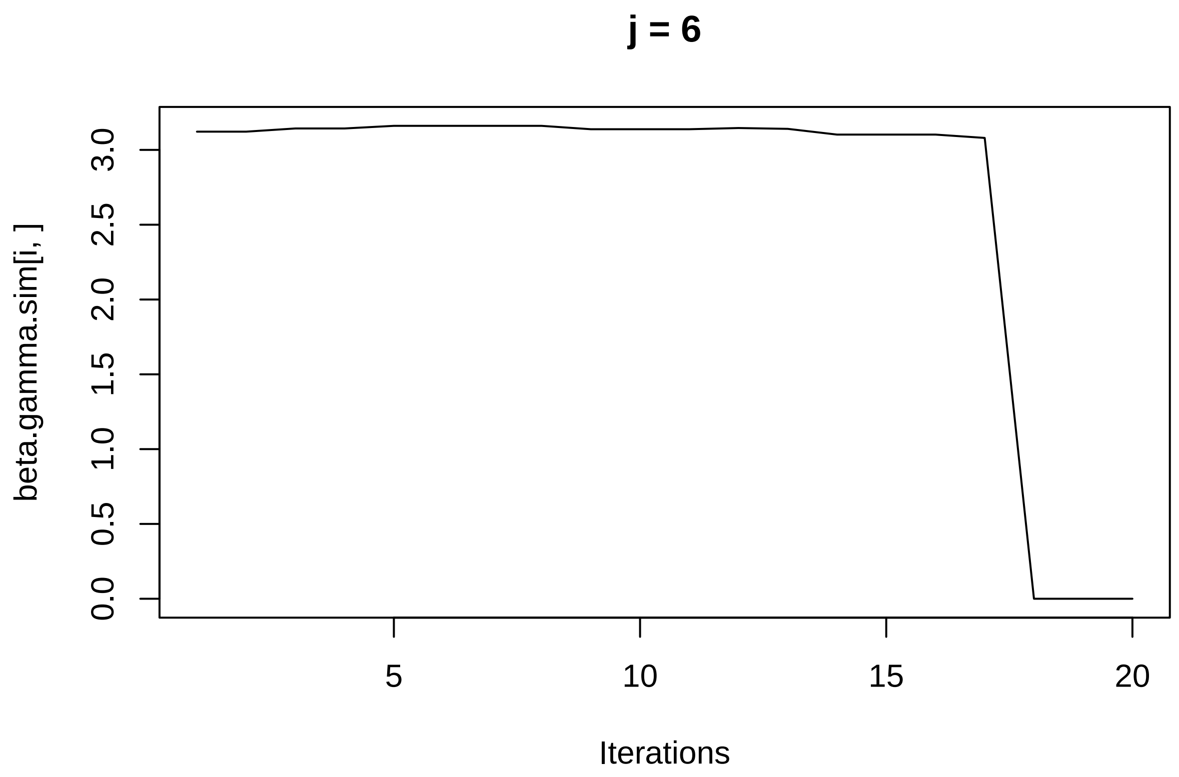
```
## [1] "3 2.10772227704122"
```



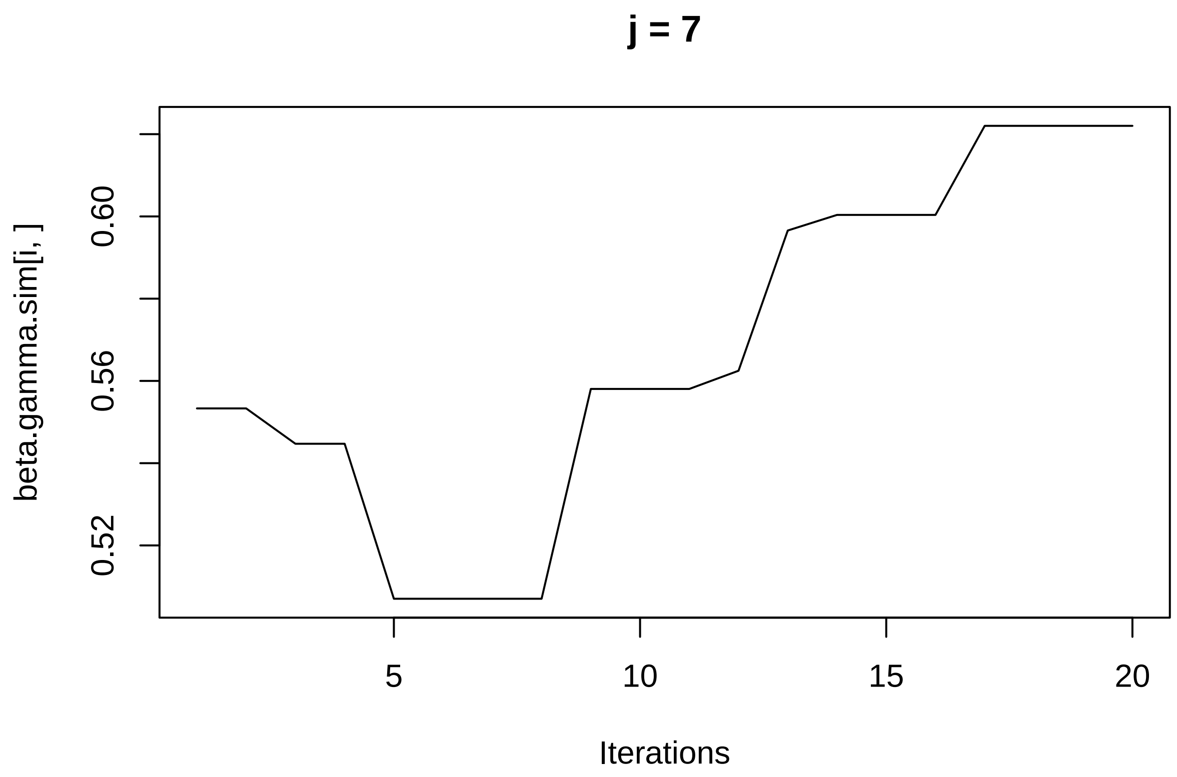
```
## [1] "4 20"
```



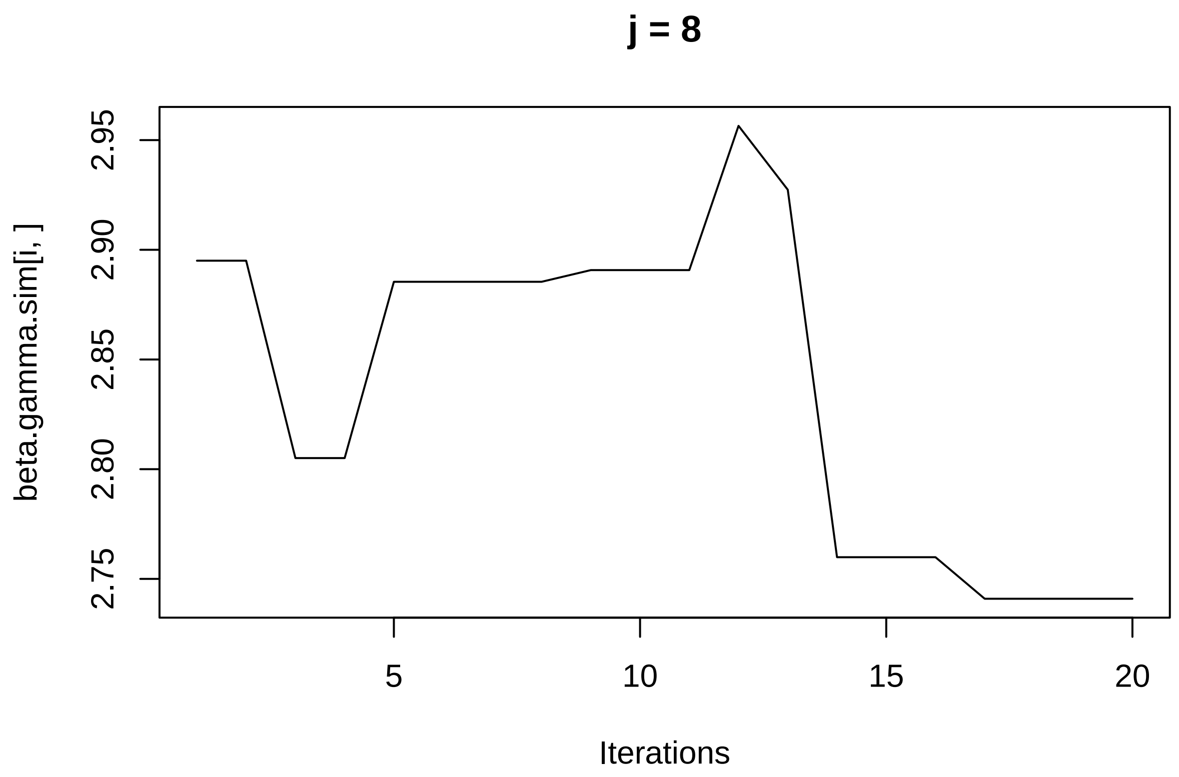
```
## [1] "5 25.1068949659177"
```



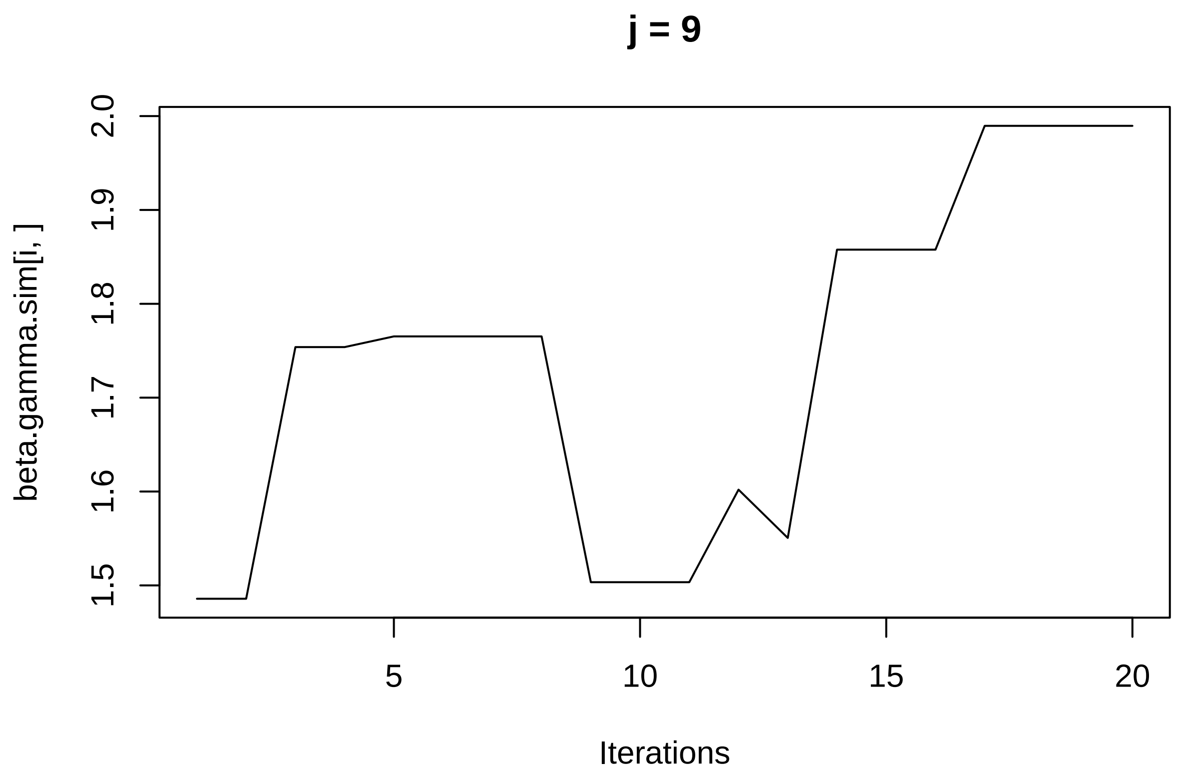
```
## [1] "6 3.81643470167932"
```



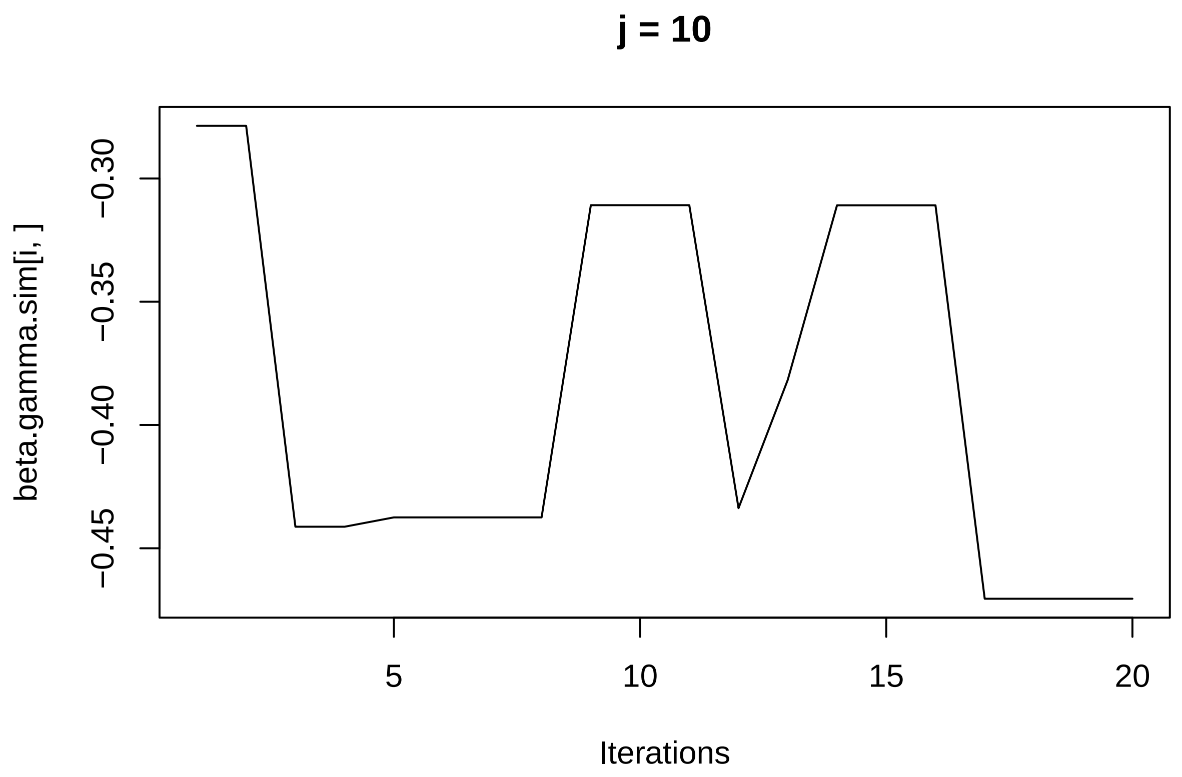
```
## [1] "7 1.40865274820465"
```



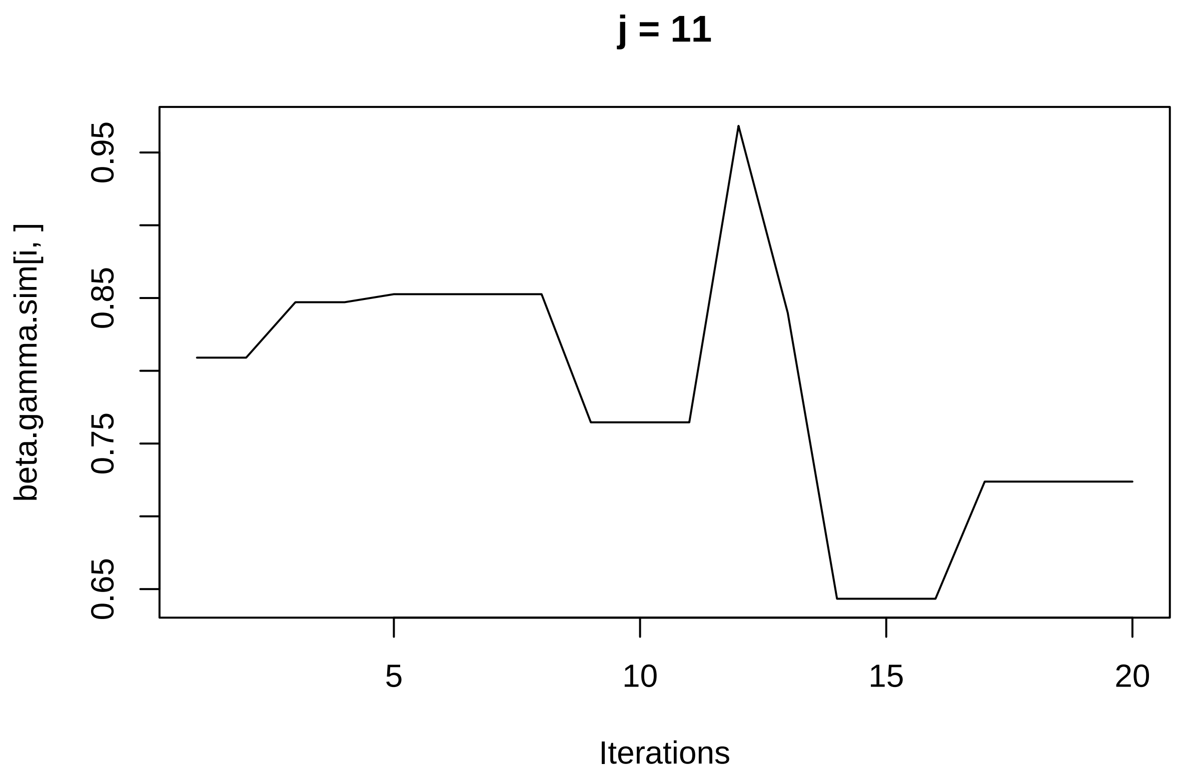
```
## [1] "8 3.24059845867508"
```



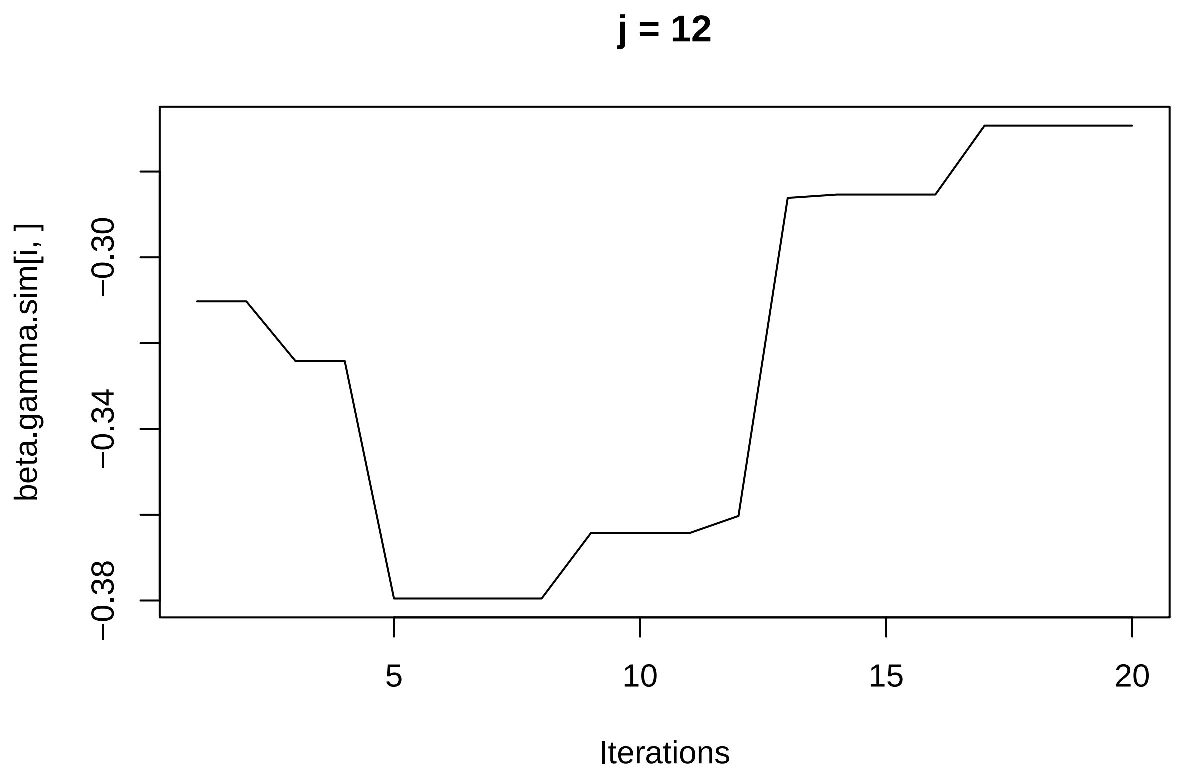
```
## [1] "9 3.38662402500952"
```



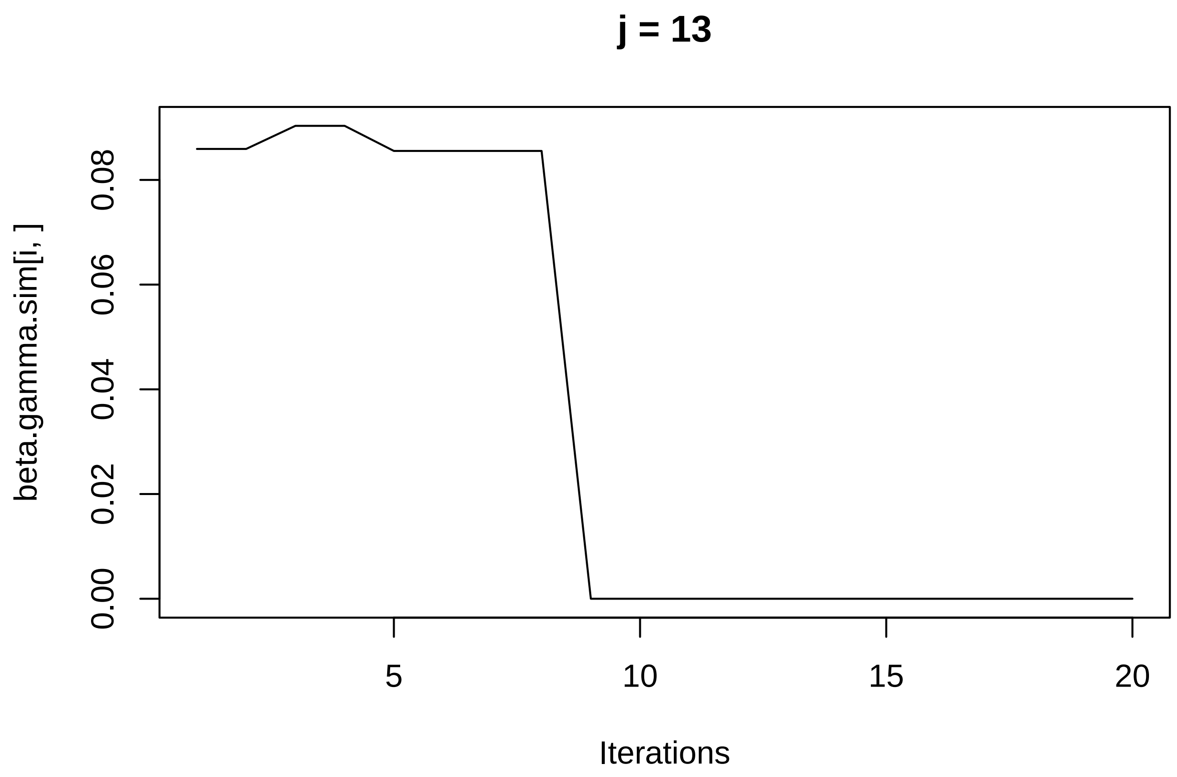
```
## [1] "10 12.3984181106994"
```



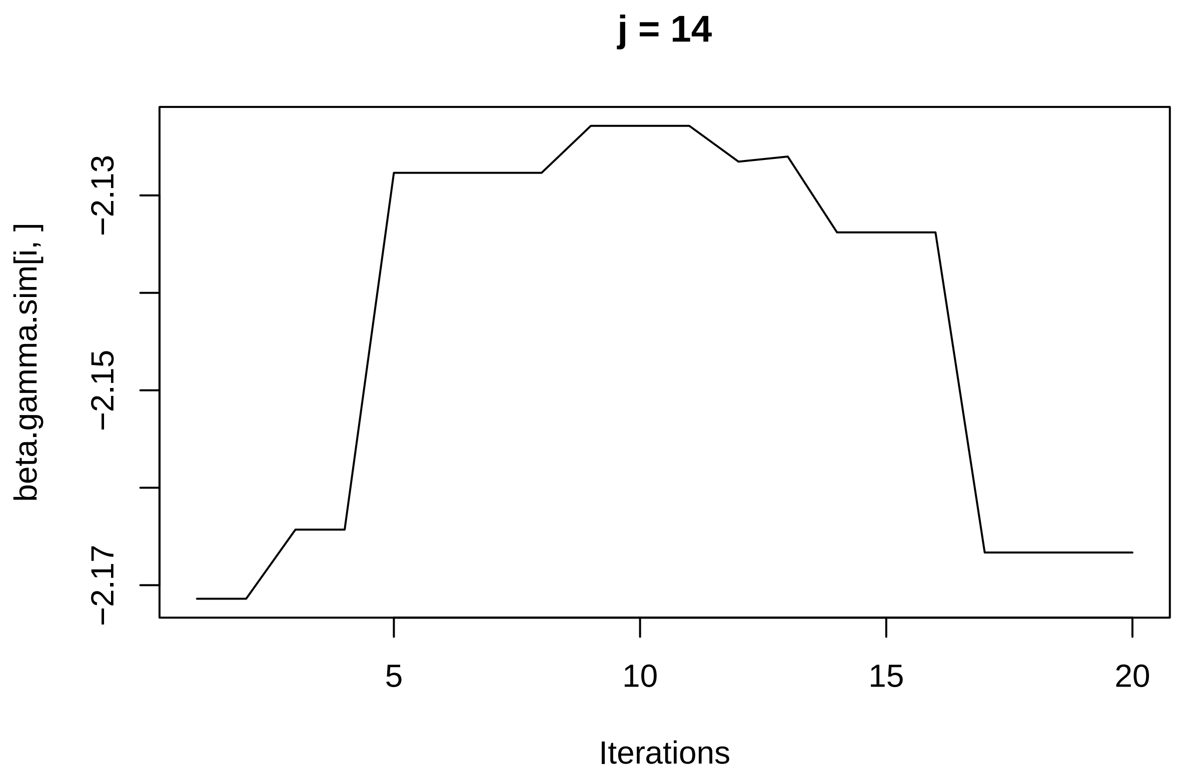
```
## [1] "11 4.97296194589513"
```



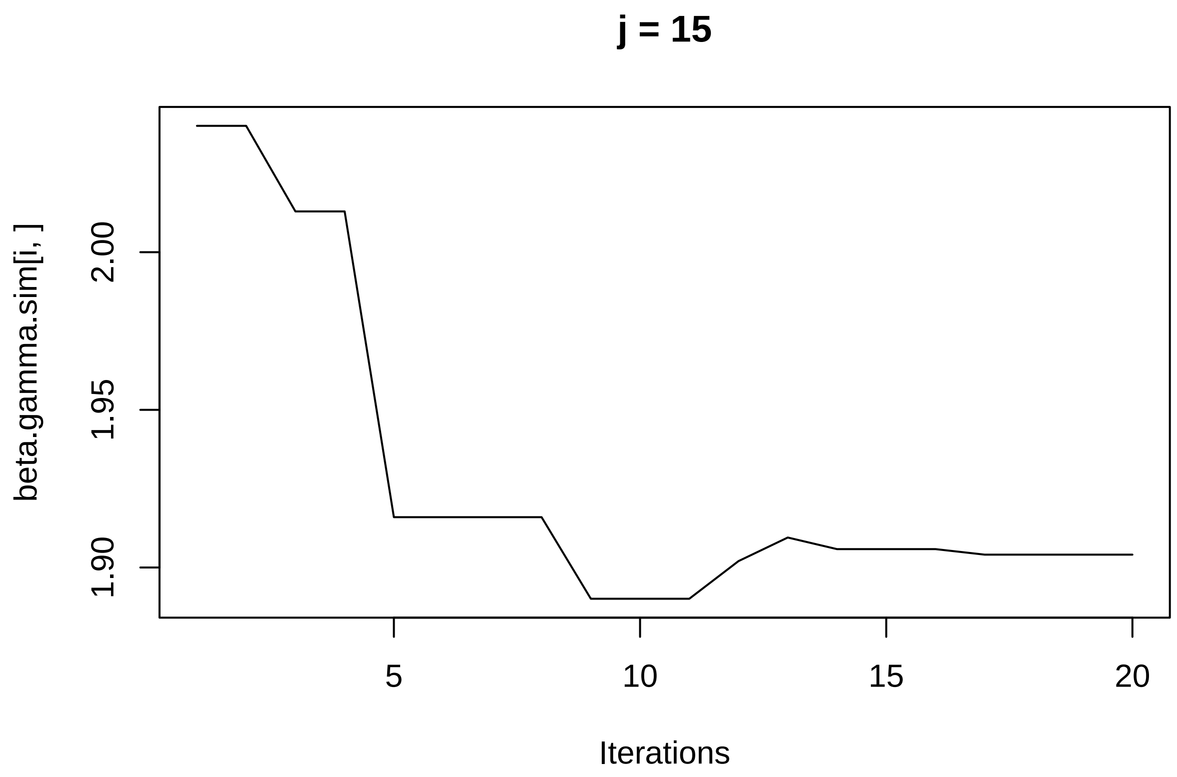
```
## [1] "12 1.73023642471374"
```



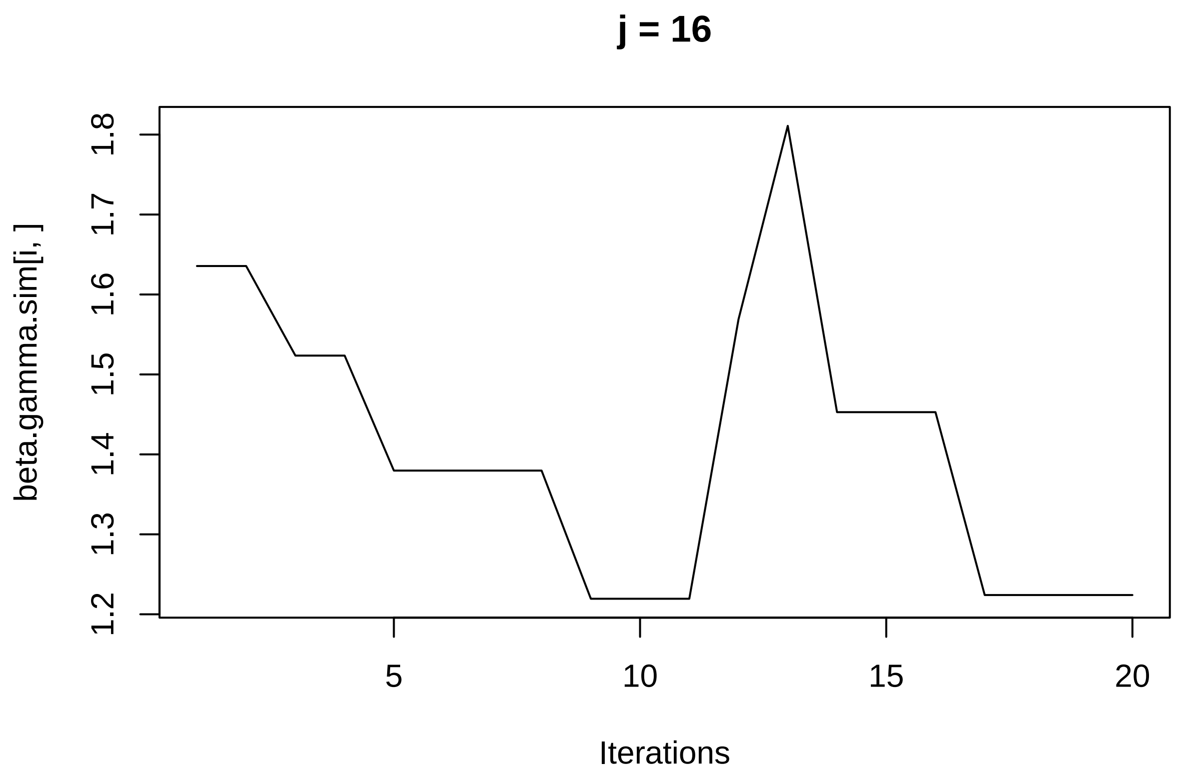
```
## [1] "13 1.58507204022262"
```



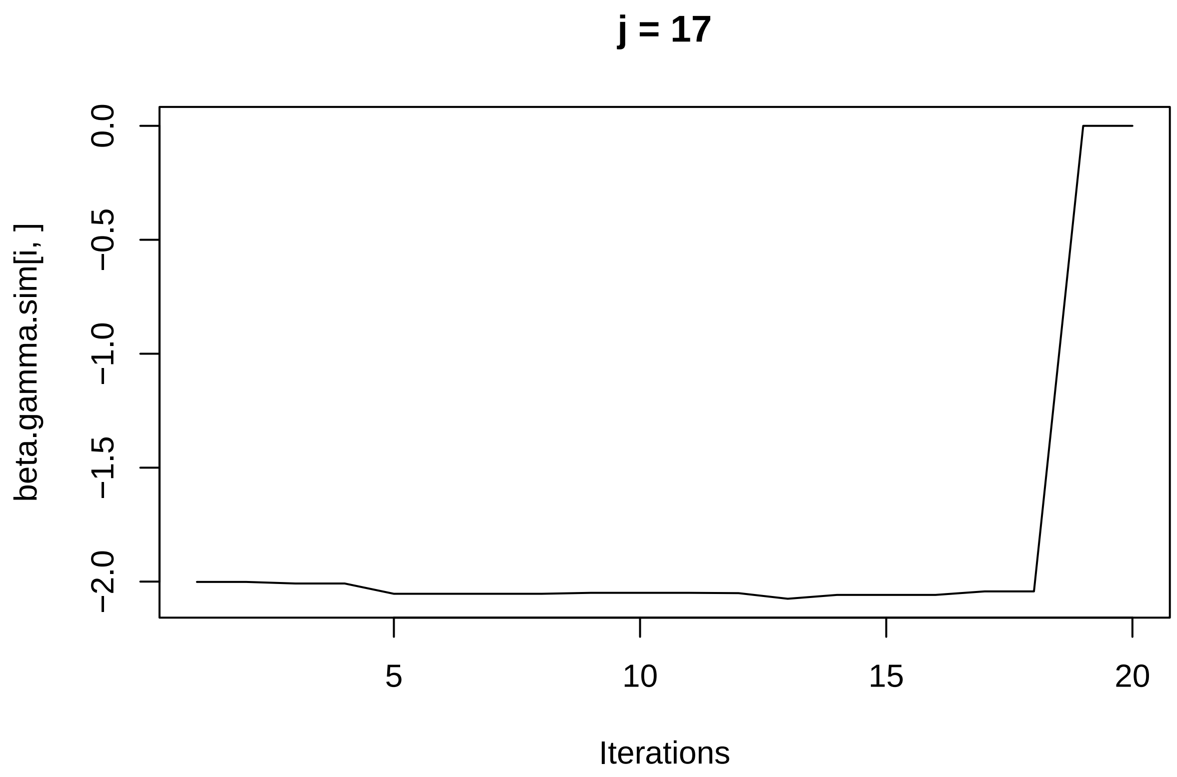
```
## [1] "14 2.71957749404512"
```



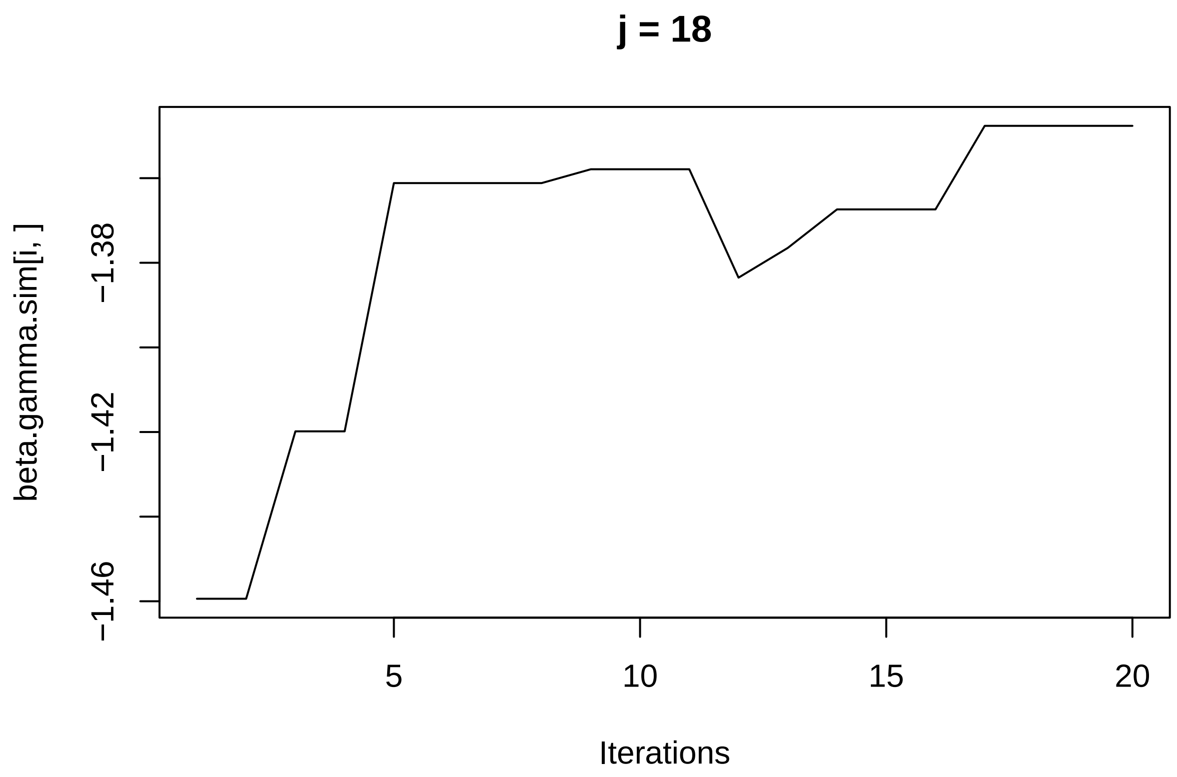
```
## [1] "15 2.62431026445946"
```



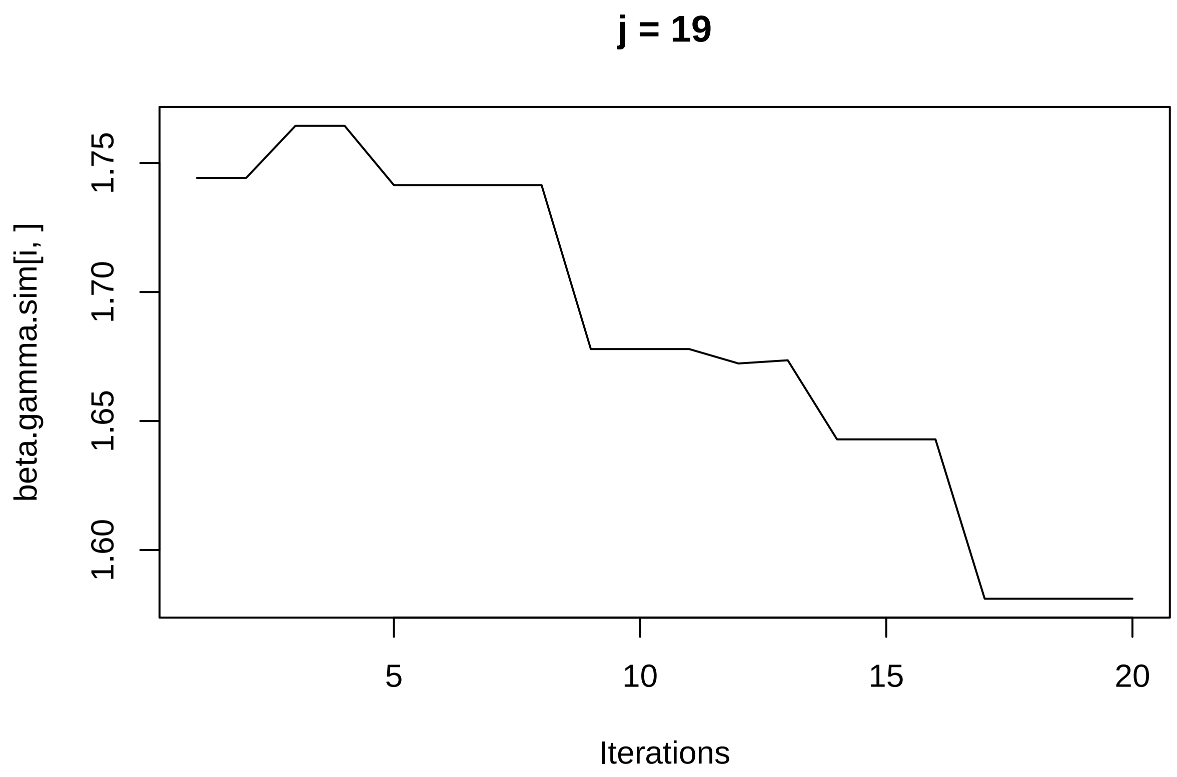
```
## [1] "16 5.40723371822474"
```



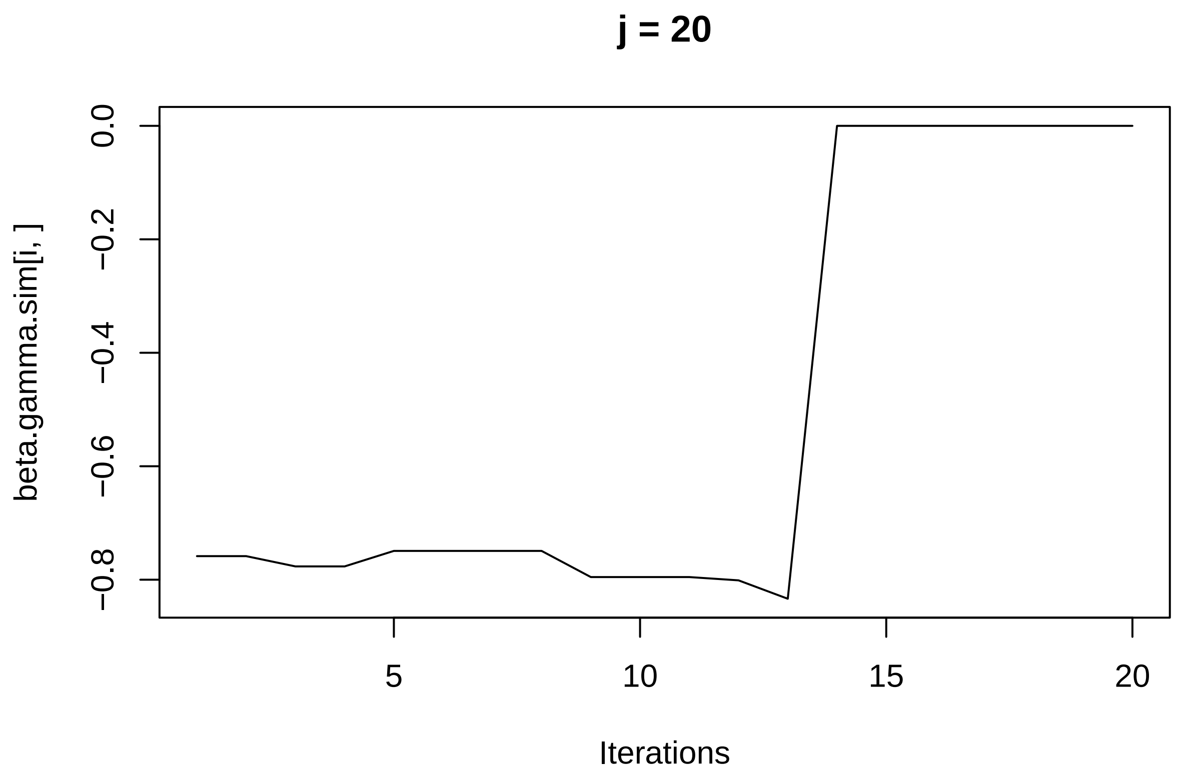
```
## [1] "17 12.1687186244389"
```



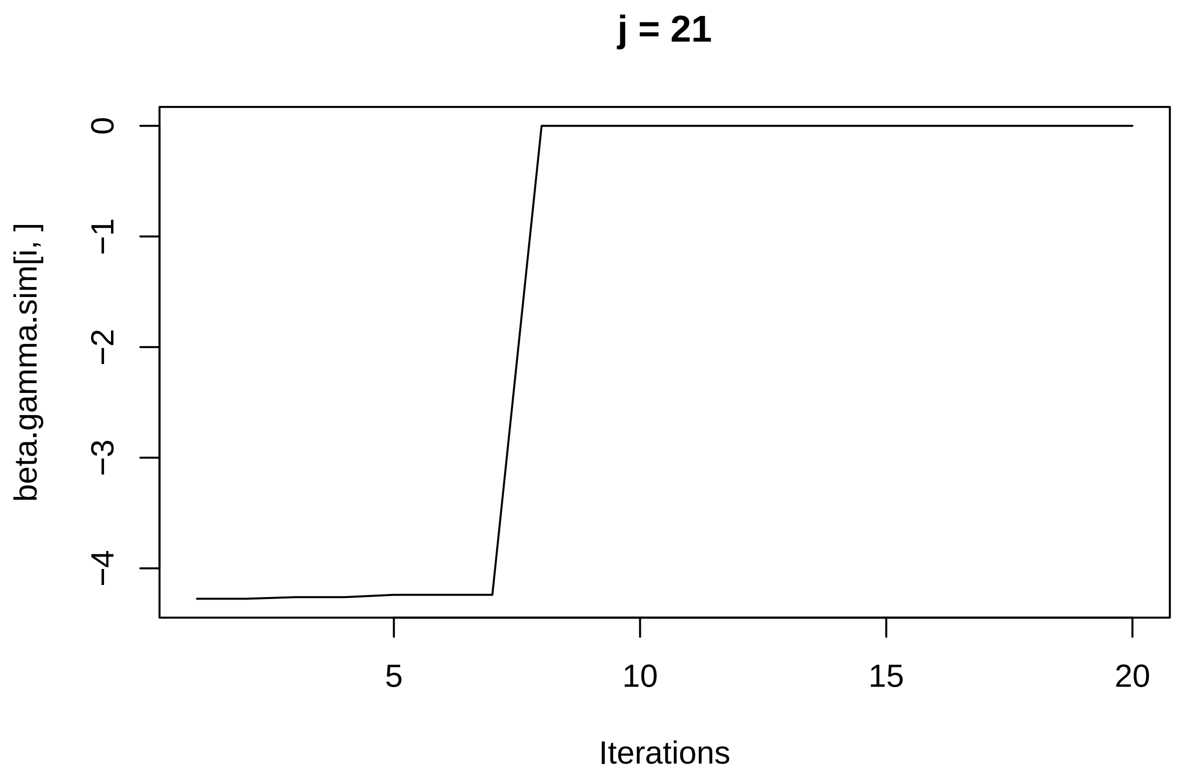
```
## [1] "18 3.35450948856924"
```



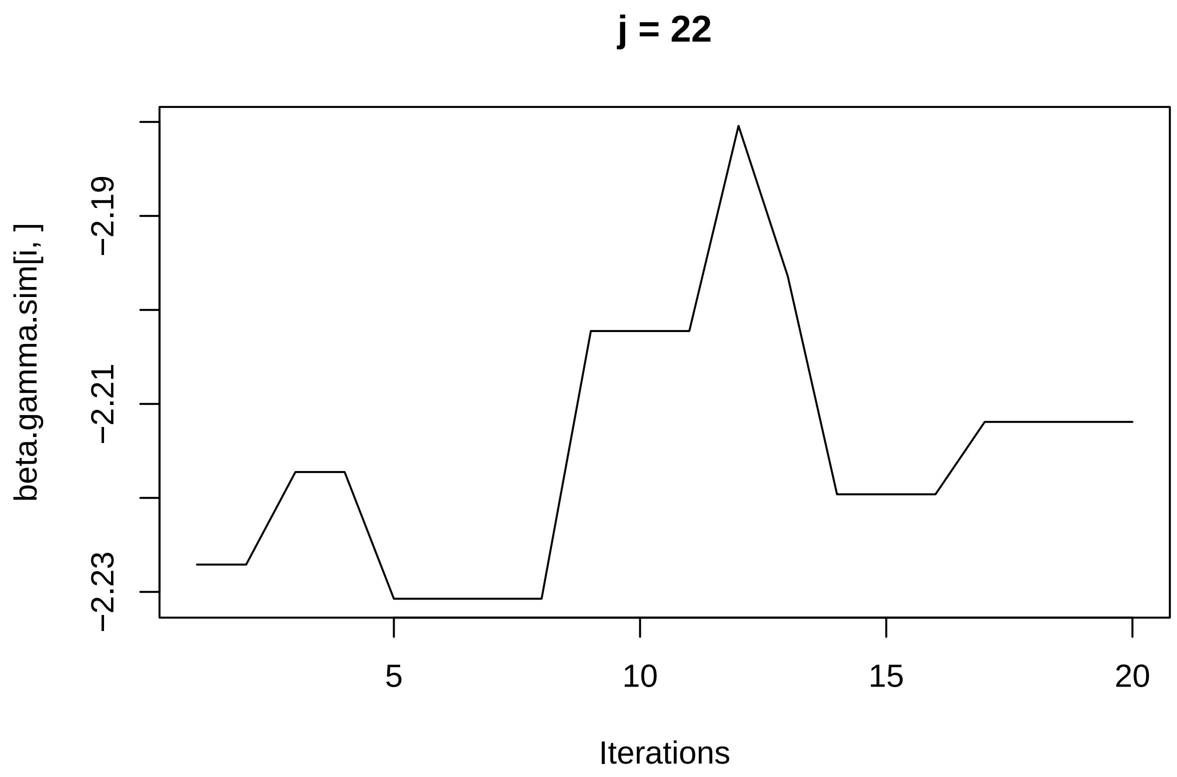
```
## [1] "19 1.49792479103866"
```



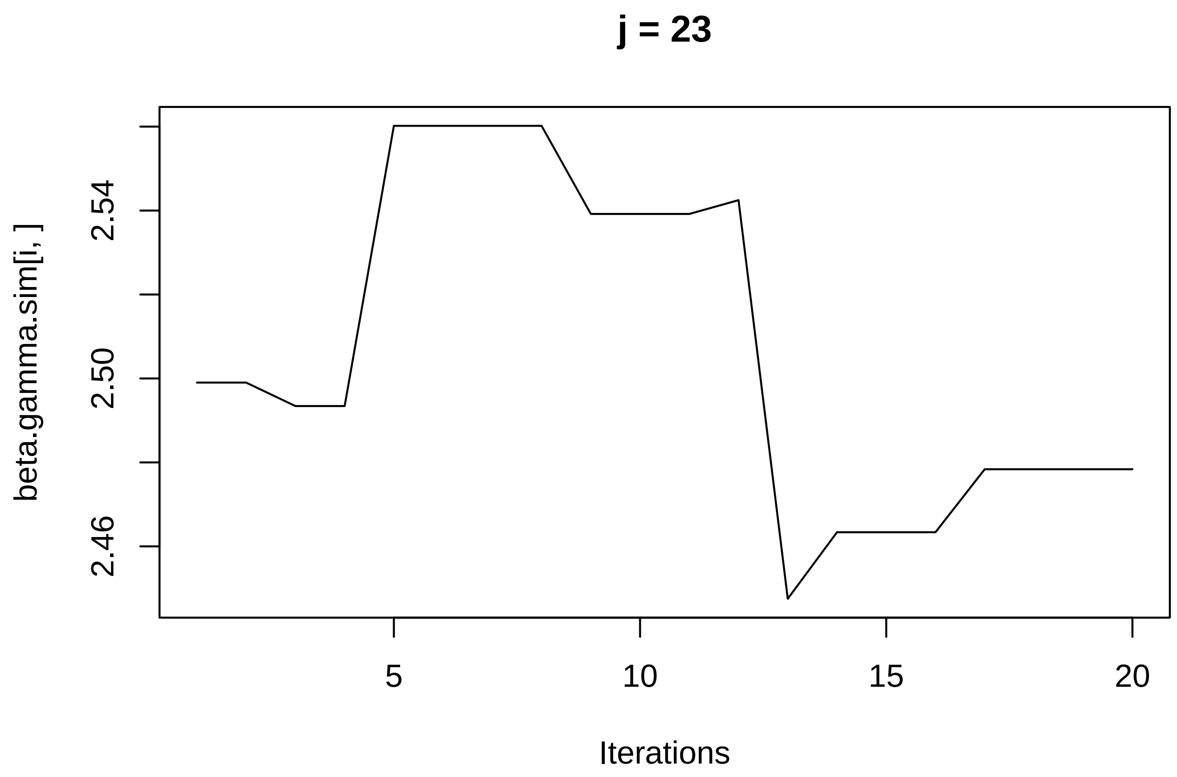
```
## [1] "20 1.93376053782194"
```



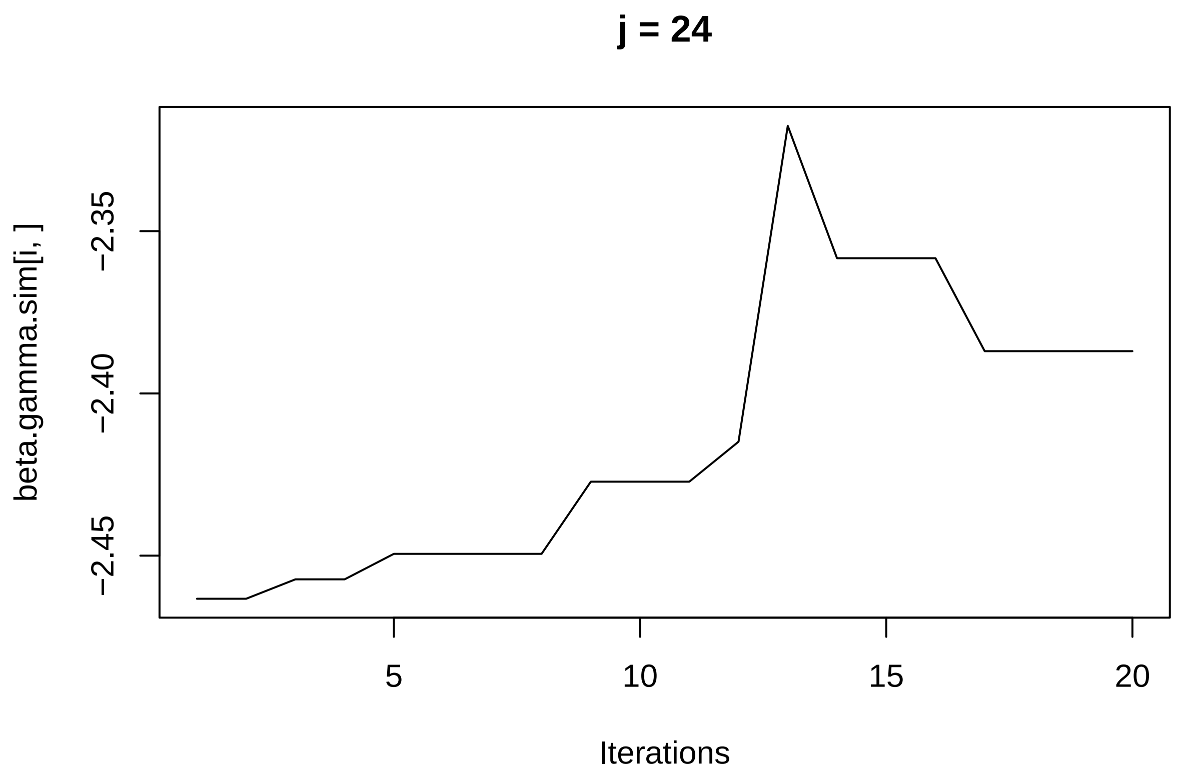
```
## [1] "21 1.75560934369473"
```



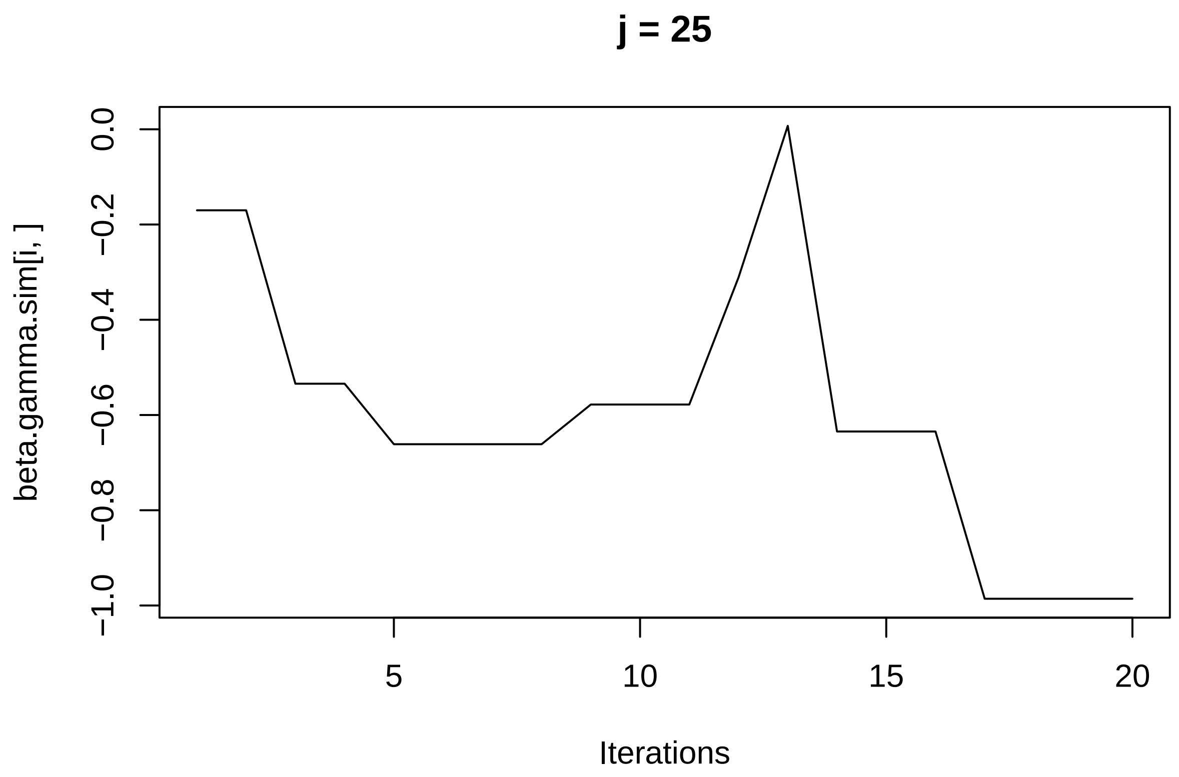
```
## [1] "22 4.38667385644732"
```



```
## [1] "23 2.91390087839954"
```



```
## [1] "24 2.37147083178321"
```



```
## [1] "25 4.94539619509848"
```

```
# gammas = as.data.frame(table(c(gammas.sim,gammas.sim2)))
gammas = as.data.frame(table(gammas.sim))

gammas = gammas[order(gammas$Freq, decreasing = T),]
gammas$Freq = format(gammas$Freq / (2*S), scientific = F)

hp.combo = vector()
for(i in 1:S){
  if(gammas$gammas.sim[1] == paste(gamma.sim[,i],collapse=' ')){
    hp.combo = gamma.sim[,i]
  }
}
colnames(X)[which(hp.combo %% 2 == 1)]
```

```
##  [1] "intercept"        "assists"          "boosts"           "DBNOs"
##  [5] "headshotKills"    "heals"            "killPlace"        "killPoints"
##  [9] "kills"            "killStreaks"      "longestKill"      "matchDuration"
## [13] "maxPlace"         "numGroups"        "rankPoints"       "revives"
## [17] "rideDistance"     "roadKills"        "swimDistance"     "teamKills"
## [21] "vehicleDestroys"  "walkDistance"     "weaponsAcquired"  "winPoints"
```

```
for(i in 1:p){
  plotrix::multhist(list(beta.gamma.sim2[i,], beta.gamma.sim[i,]), freq = F)
}
```

Compare Posterior mean of beta Gibbs and MH

```
gib.beta.mean = vector()
mh.beta.mean = vector()
for(i in 1:p){
  gib.beta.mean[i] = mean(beta[i,]) %>% round(3)
  mh.beta.mean[i] = mean(c(beta.gamma.sim[i,],beta.gamma.sim2[i,]))%>% round(3)
}
cbind(gib.beta.mean,mh.beta.mean, abs(mh.beta.mean-gib.beta.mean))
```

Compare Posterior mean of beta for two MH chain

```
mh.beta.mean1 = vector()
mh.beta.mean2 = vector()
for(i in 1:p){
  mh.beta.mean1[i] = mean(beta.gamma.sim[i,]) %>% round(3)
  mh.beta.mean2[i] =  mean(beta.gamma.sim2[i,])%>% round(3)
}

cbind(format(mh.beta.mean1, digits = 5,scientific=F),format(mh.beta.mean2, digits = 5), format(mh.beta.
      , c("intercept",colnames(tdata)[1:24]))
```

Test Error of MH

```
mh.beta.mean = vector()
for(i in 1:p){
  # mh.beta.mean[i] = mean(c(beta.gamma.sim[i,],beta.gamma.sim2[i,]))
  mh.beta.mean[i] = mean(beta.gamma.sim[i,])
}

test.data = readRDS("split_test_dataset.rds")
X = test.data %>% select(-c(winPlacePerc)) %>% as.matrix()
X = cbind(rep(1,nrow(X)), X)
y = test.data$winPlacePerc

pred.MH = X %*%  mh.beta.mean
pred.MH = pnorm(pred.MH)
test.error.MH = sum((y-pred.MH)^2);test.error.MH
```

```
## [1] 18939.59
```

Lasso Model

```r
#reformat y
y= as.matrix(tdata$winPlacePerc)
one.index = which(y==1)
second.highest = sort(unique(y), decreasing = T)[2]
remake.one = runif(length(one.index), min = second.highest+0.0001,max= 1-0.0001)
y[one.index] = remake.one
zero.index = which(y==0)
second.lowest = sort(unique(y))[2]
remake.zero = runif(length(zero.index), min = 0+0.0001,max= second.lowest-0.0001)
y[zero.index] = remake.zero
y=qnorm(y)

X = tdata %>% select(-c(winPlacePerc)) %>% as.matrix
X = cbind(intercept = rep(1, nrow(X)), X) %>% as.matrix


cv.out <- cv.glmnet(X,y, nfolds = 5, alpha = 1)
test.data = readRDS("split_test_dataset.rds")
X.test = test.data %>% select(-c(winPlacePerc)) %>% as.matrix()
X.test = cbind(rep(1,nrow(X.test)), X.test)
y.test = test.data$winPlacePerc

model <- glmnet(x = X, y = y, alpha = 1, lambda = cv.out$lambda.min)
prediction.lasso <- predict(model, newx = X.test)
prediction.lasso = pnorm(prediction.lasso)
test.error.lasso = sum((y.test-prediction.lasso)^2);test.error.lasso
```

```
## [1] 494.7647
```