stat 601 A2

2023-03-22

Question 1: Bayesian Linear Regression

i)

```
data = read.csv("VaccineUptake.csv", header = TRUE)
head(data,3)
```

```
##   Coverage  Incidence
## 1 69.37620  0.7803454
## 2 56.59420 32.5109646
## 3 61.00795 16.9616828
```

```
model1 = lm(Incidence~Coverage, data = data)
summary(model1)
```

```
##
## Call:
## lm(formula = Incidence ~ Coverage, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -32.633  -7.217  -1.169   5.719  30.747
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 134.4630    22.6518   5.936 9.37e-07 ***
## Coverage     -1.9025     0.3741  -5.085 1.24e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.07 on 35 degrees of freedom
## Multiple R-squared:  0.4249, Adjusted R-squared:  0.4085
## F-statistic: 25.86 on 1 and 35 DF,  p-value: 1.239e-05
```

NIMBLE model syntax

```
library(nimble)
```

```
## nimble version 0.13.1 is loaded.
## For more information on NIMBLE and a User Manual,
```

```
## please visit https://R-nimble.org.
##
## Note for advanced users who have written their own MCMC samplers:
##   As of version 0.13.0, NIMBLE's protocol for handling posterior
##   predictive nodes has changed in a way that could affect user-defined
##   samplers in some situations. Please see Section 15.5.1 of the User Manual.

##
## Attaching package: 'nimble'

## The following object is masked from 'package:stats':
##
##     simulate
```

```
n = nrow(data)
b0_true = 134.4630
b1_true = -1.9025
sigma_true = 12.07
x = data$Coverage
y = data$Incidence

lmCode = nimbleCode({
  # likelihood
  for(i in 1:n){
    mu[i] <- b0 + b1*x[i]
    y[i]~dnorm(mu[i], sd=sigma)
  }

  # priors
  b0~dunif(-800,800)
  b1~dunif(-800,800)
  sigma~dunif(0,500)

})
```

Sampling from the posterior using the default MCMC algorithm

```
dataList = list(y = y)
constantsList = list(n=n, x=x)
initsList = list(b0=130, b1=-3, sigma =12)
# create nimble model object
output = nimbleMCMC(lmCode, data = dataList,
                            constants = constantsList,
                            inits = initsList,
                            niter = 30000, summary = TRUE, nburnin = 2000, thin = 3)
```

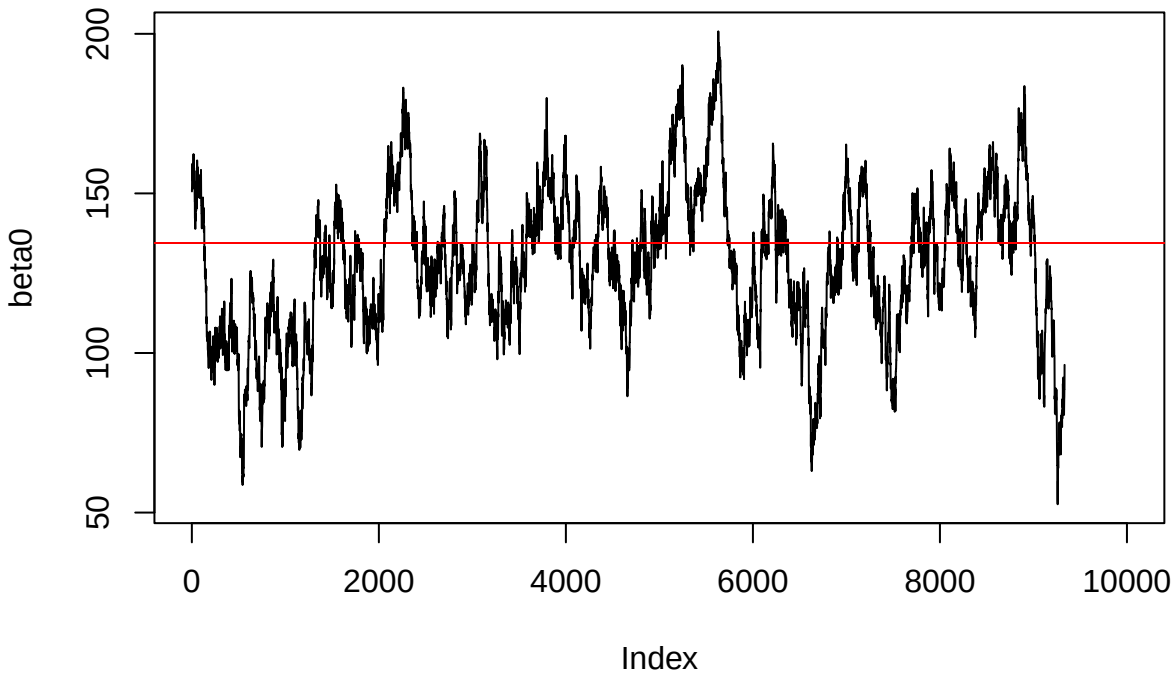
```
## Defining model

## Building model

## Setting data and initial values
```

```
## Running calculate on model
##   [Note] Any error reports that follow may simply reflect missing values in model variables.

## Checking model sizes and dimensions

## Checking model calculations

## Compiling
##   [Note] This may take a minute.
##   [Note] Use 'showCompilerOutput = TRUE' to see C++ compilation details.

## running chain 1...

## |-------------|-------------|-------------|-------------|
## |-----------------------------------------------------|
```

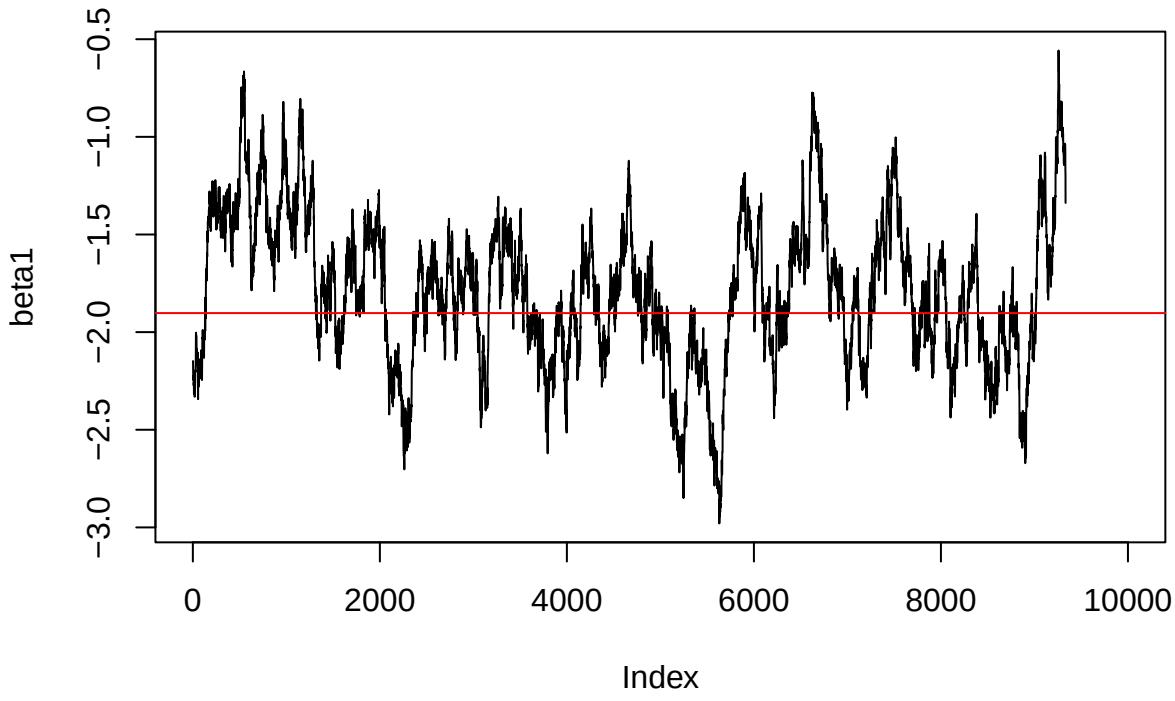
```
output$summary
```

```
##             Mean     Median    St.Dev. 95%CI_low  95%CI_upp
## b0    127.810310 127.950586 23.0202979 80.291906 174.624455
## b1     -1.792675  -1.792352  0.3798344 -2.561066  -1.003026
## sigma  12.515817  12.344455  1.5727673  9.943551  16.012588
```

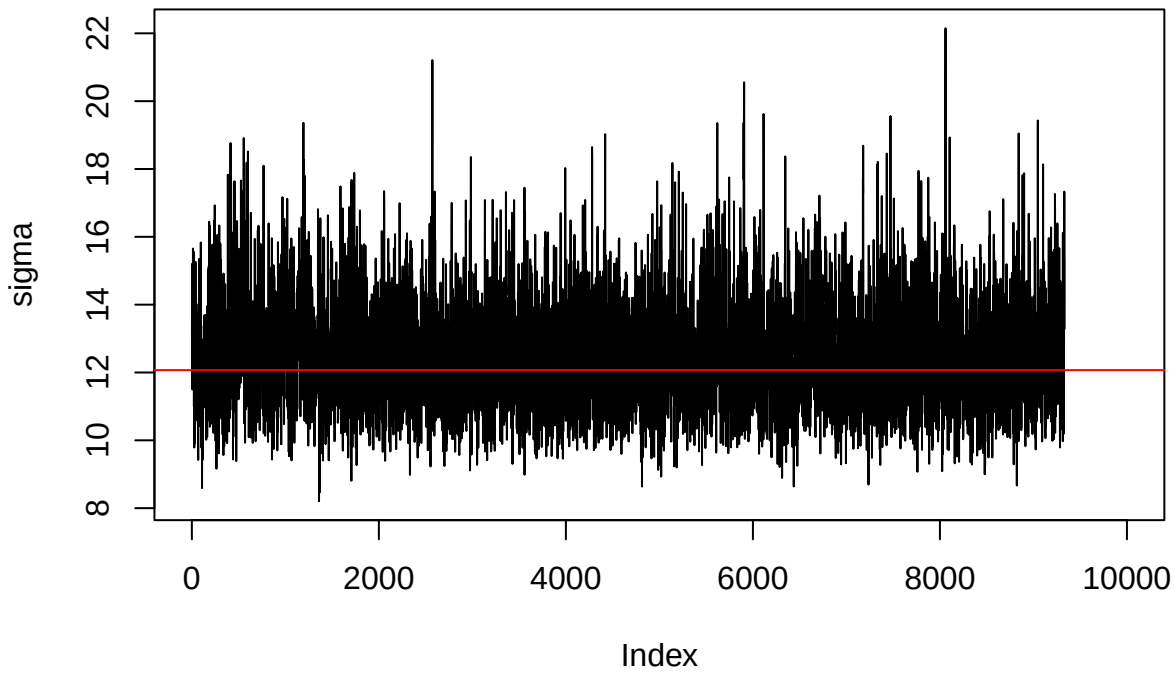
```
plot(output$samples[,'b0'], type = "l", xlim = c(0,10000), ylab = "beta0")
abline(h=b0_true, col = "red")
```



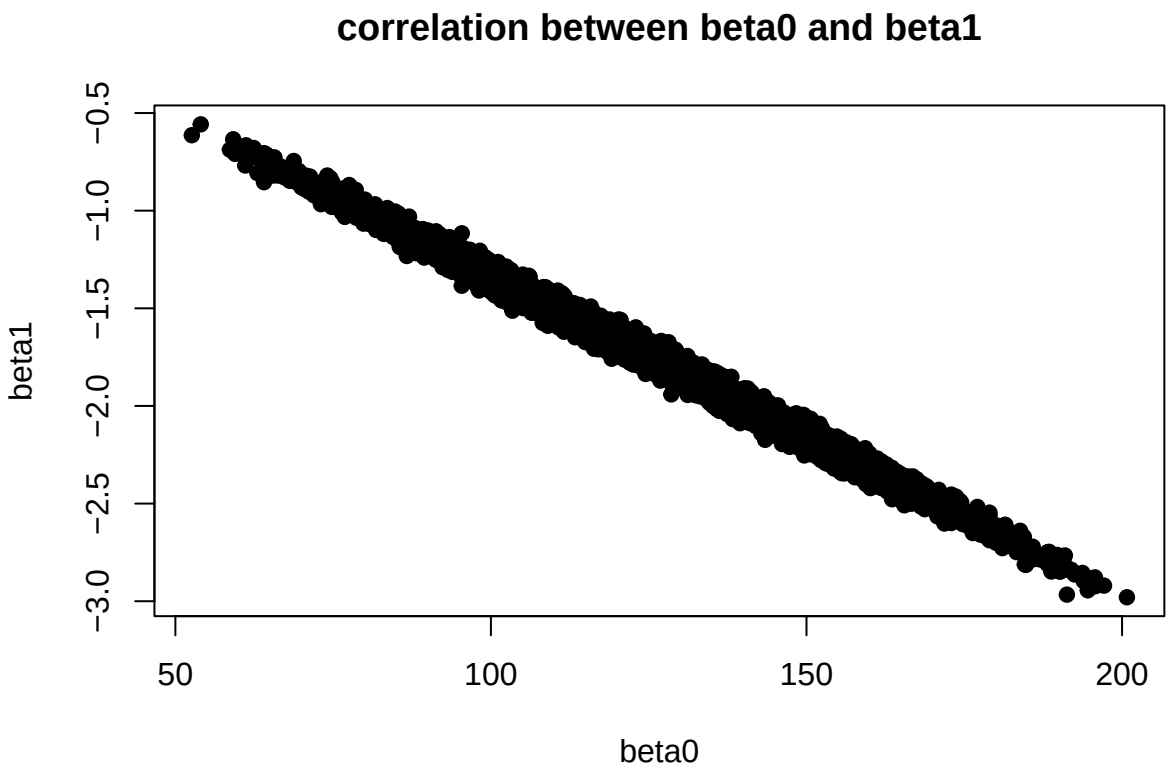
```
plot(output$samples[,'b1'], type = "l", xlim = c(0,10000), ylab = "beta1")
abline(h=b1_true, col = "red")
```




```
plot(output$samples[,'sigma'], type = "l", xlim = c(0,10000), ylab = "sigma")
abline(h=sigma_true, col = "red")
```




```
plot(output$samples[,'b0'], output$samples[,'b1'], main = "correlation between beta0 and beta1",
     xlab = "beta0", ylab = "beta1", pch = 19)
```



ii)

```
lmModel <- nimbleModel(lmCode, data = dataList,
                            constants = constantsList,
                            inits = initsList)
```

```
## Defining model

## Building model

## Setting data and initial values

## Running calculate on model
##   [Note] Any error reports that follow may simply reflect missing values in model variables.

## Checking model sizes and dimensions
```

```
model1MCMCconfiguration <- configureMCMC(lmModel)
```

```
## ===== Monitors =====
## thin = 1: b0, b1, sigma
```

```
## ===== Samplers =====
## RW sampler (3)
##   - b0
##   - b1
##   - sigma
```

```
# model1MCMCconfiguration$printSamplers()
model1MCMCconfiguration$removeSamplers('b0')
model1MCMCconfiguration$addSampler(target = 'b0', type = 'RW_block')
```

```
##   [Note] Assigning an RW_block sampler to nodes with very different scales can result in low MCMC ef
```

```
model1MCMCconfiguration$removeSamplers('b1')
model1MCMCconfiguration$addSampler(target = 'b1', type = 'RW_block')
```

```
##   [Note] Assigning an RW_block sampler to nodes with very different scales can result in low MCMC ef
```

```
# configureMCMC(model1)
lmMCMC = buildMCMC(model1MCMCconfiguration)
lmCompiled = compileNimble(lmModel, lmMCMC)
```

```
## Compiling
##   [Note] This may take a minute.
##   [Note] Use 'showCompilerOutput = TRUE' to see C++ compilation details.
```

```
samples = runMCMC(lmCompiled$lmMCMC, niter = 30000, nburnin = 2000, summary = TRUE, thin = 3)
```

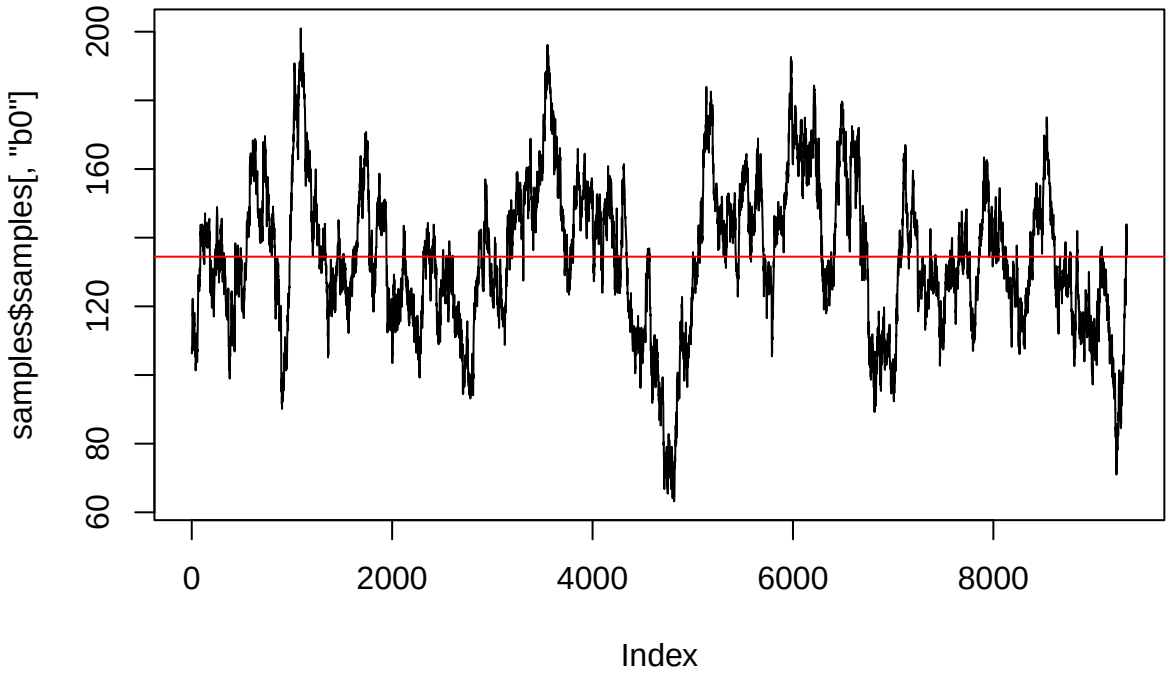
```
## running chain 1...
```

```
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
```

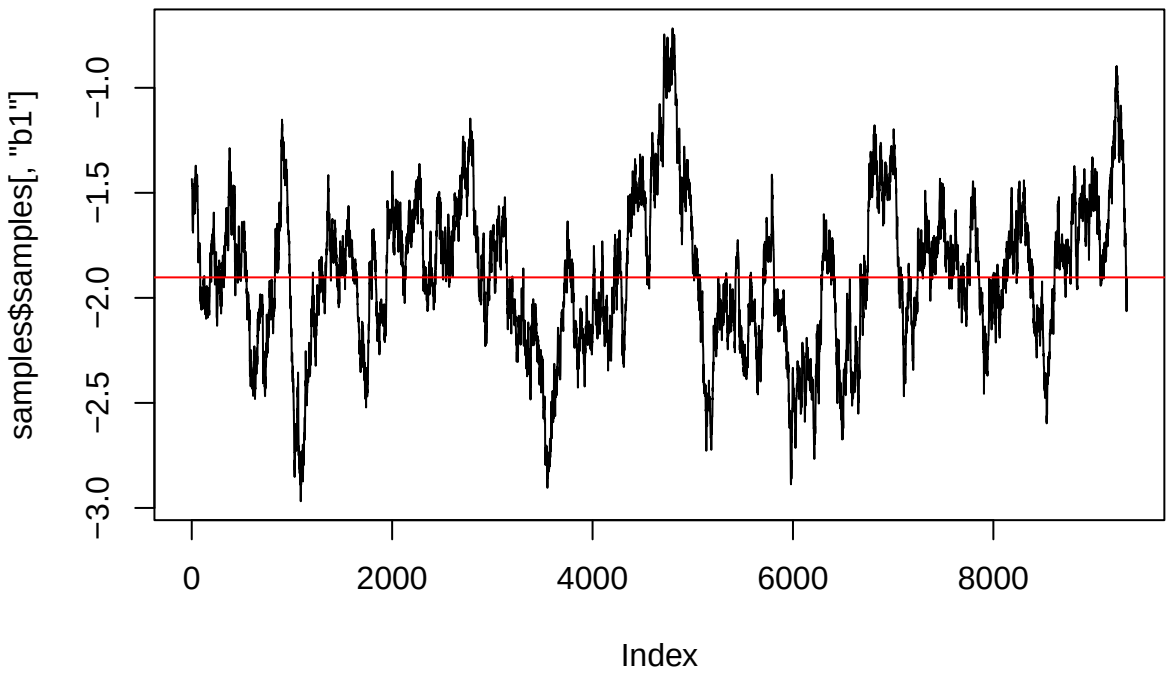
```
samples$summary
```

```
##             Mean     Median    St.Dev. 95%CI_low  95%CI_upp
## b0    134.104177 133.427144 21.4537307 93.354788 176.862190
## b1     -1.896409  -1.885196  0.3545956 -2.595972  -1.225303
## sigma  12.475577  12.315503  1.5711557  9.853881  15.981260
```

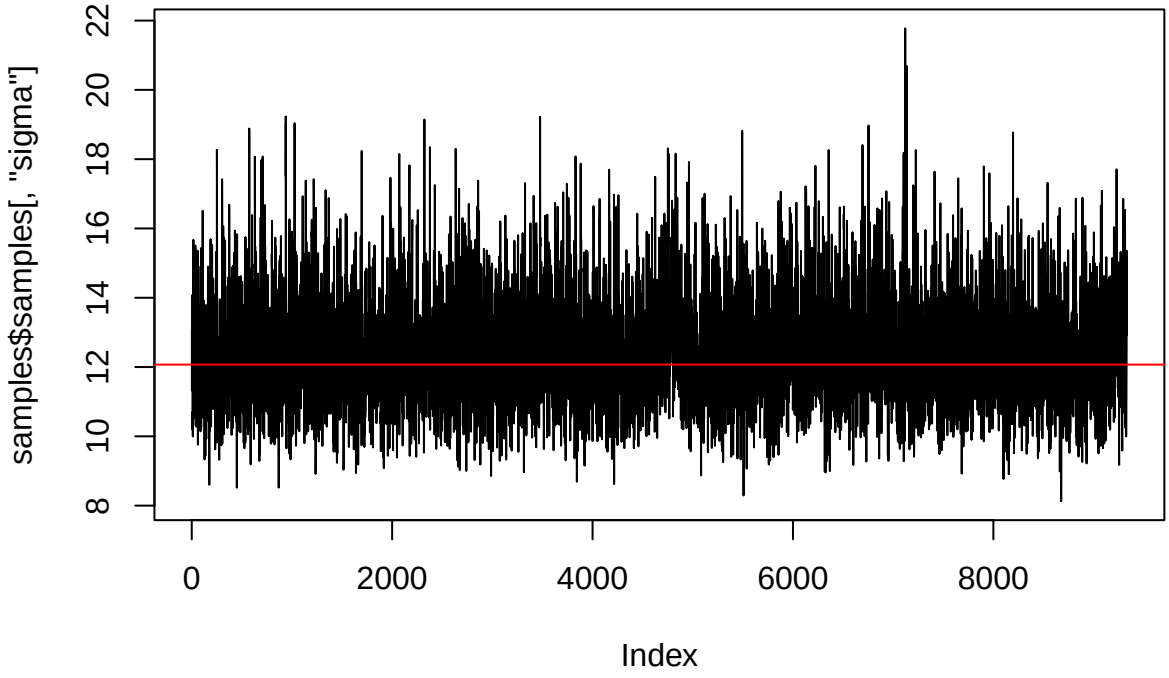
```
plot(samples$samples[,'b0'], type = "l")
abline(h=b0_true, col = "red")
```



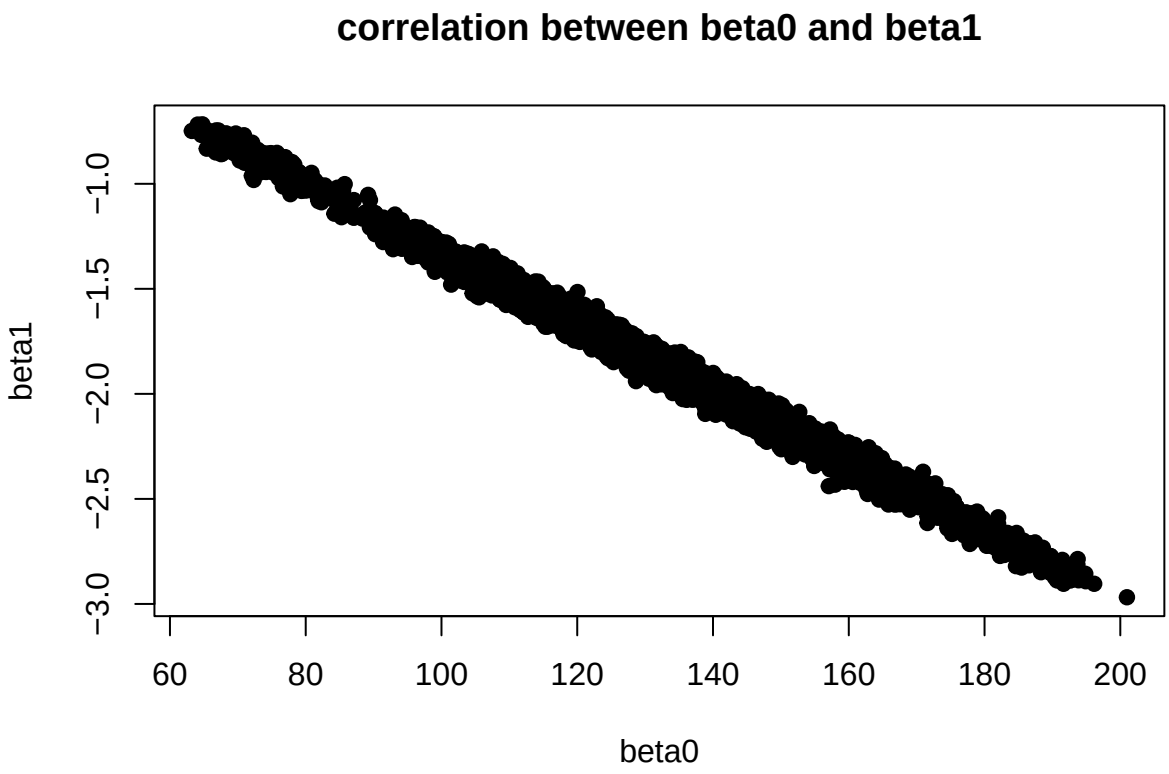

```
plot(samples$samples[,'b1'], type = "l")
abline(h=b1_true, col = "red")
```




```
plot(samples$samples[,'sigma'], type = "l")
abline(h=sigma_true, col = "red")
```




```
plot(samples$samples[,'b0'], samples$samples[,'b1'], main = "correlation between beta0 and beta1",
     xlab = "beta0", ylab = "beta1", pch = 19)
```



The default MH proposals (samplers) being used in part i) updates one parameter at a time and it moves orthogonality; on the other hand, the RW block sampler being used in part ii) updates both parameters in one step, so it can move in any direction in the parameter space, which is more efficient for correlated variables as illustrated on the scatter plot between β_0 and β_1

iii)

```
# create nimble model object
output = nimbleMCMC(lmCode, data = dataList,
                            constants = constantsList,
                            inits = initsList, nchains = 20,
                            niter = 30000, summary = TRUE)
```

```
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
```

```
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
```

```
output$summary$chain20
```

```
##             Mean     Median   St.Dev. 95%CI_low  95%CI_upp
## b0    130.970317 130.712522 19.510453 91.840202 167.918775
## b1     -1.845691  -1.842532  0.322244 -2.456435  -1.203038
## sigma  12.496924  12.324685  1.584592  9.901282  16.001219
```

```
output$summary$all.chains
```

```
##             Mean     Median    St.Dev. 95%CI_low  95%CI_upp
## b0    134.845653 134.348303 24.5332867 88.909482 184.378952
## b1     -1.908928  -1.900645  0.4052861 -2.724947  -1.152909
## sigma  12.544651  12.379484  1.5919393  9.911806  16.137030
```

Question 2: Bayesian SIR Model

part i)

```
library(nimble)
data = read.csv("A2Epidemic1.csv", header = TRUE)
head(data)
```

```
##   NewCases NewRemovals
## 1        1           2
## 2        1           2
## 3        2           0
## 4        3           1
## 5        1           4
## 6        1           0
```

```
# create Istar and Rstar
new_cases <- data$NewCases
new_removals <- data$NewRemovals
```

```
SIR_code <- nimbleCode({
  S[1] <- N - I0 - R0
  I[1] <- I0
  R[1] <- R0
  probIR <- 1 - exp(-gamma)
  ### loop over time
  for(t in 1:tau) {
    probSI[t] <- 1 - exp(- beta * I[t] / N)
    Istar[t] ~ dbin(probSI[t], S[t])
    Rstar[t] ~ dbin(probIR, I[t])
    # update S, I, R
    S[t + 1] <- S[t] - Istar[t]
    I[t + 1] <- I[t] + Istar[t] - Rstar[t]
    R[t + 1] <- R[t] + Rstar[t]
}
# priors
beta ~ dunif(0,10)
gamma ~ dunif(0,10)
})
```

```
constantsList <- list(N = 1000, I0 = 5, R0 = 0, tau = 17)
dataList <- list(Istar = new_cases, Rstar = new_removals)
sirModel <- nimbleModel(SIR_code, constants = constantsList, data = dataList)
```

```
## Defining model

## Building model

## Setting data and initial values
```

```
## Running calculate on model
##   [Note] Any error reports that follow may simply reflect missing values in model variables.

## Checking model sizes and dimensions

##   [Note] This model is not fully initialized. This is not an error.
##          To see which variables are not initialized, use model$initializeInfo().
##          For more information on model initialization, see help(modelInitialization).
```

```
initisList = list(beta = runif(1,0,1), gamma = runif(1,0,10))
sirModel$setInits(initisList)
# exclude data from parent nodes
dataNodes <- c('Istar', 'Rstar')
dataNodes <- sirModel$expandNodeNames(dataNodes, returnScalarComponents = TRUE)
parentNodes <- sirModel$getParents(dataNodes, stochOnly = TRUE)
parentNodes <- parentNodes[-which(parentNodes %in% dataNodes)]
# parentNodes # beta gamma
parentNodes <- sirModel$expandNodeNames(parentNodes, returnScalarComponents = TRUE)
nodesToSim <- sirModel$getDependencies(parentNodes, self = FALSE, downstream = T)

## for part iii)
part_i_S = sirModel$S
part_i_I = sirModel$I
part_i_R = sirModel$R
```

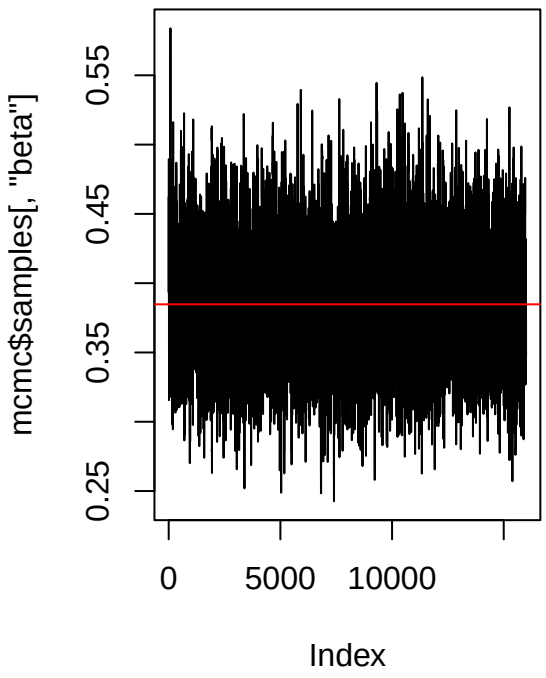
```
mcmc <- nimbleMCMC(sirModel, nodesToSim, summary = TRUE,
                   nchains = 1, thin = 1, niter = 20000, nburnin = 4000)
```

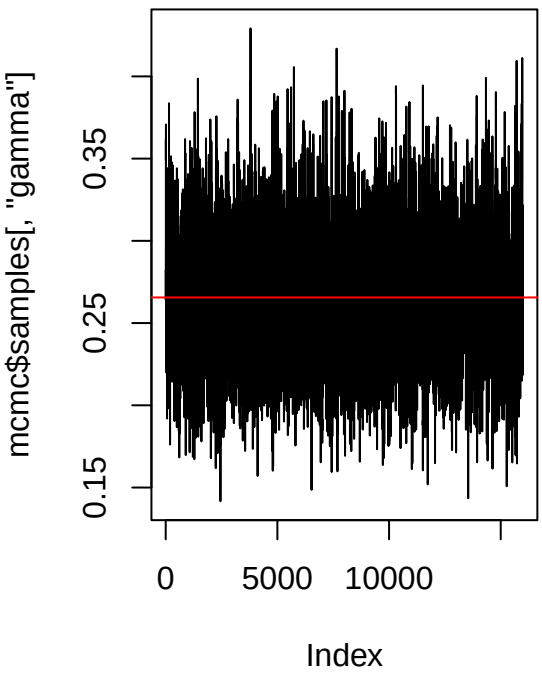
```
## Compiling
##   [Note] This may take a minute.
##   [Note] Use 'showCompilerOutput = TRUE' to see C++ compilation details.

## running chain 1...

## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
```

```
# summarize results
mcmc$summary
```

```
##            Mean    Median    St.Dev. 95%CI_low 95%CI_upp
## beta  0.3847440 0.3827027 0.04431530 0.3042194 0.4759302
## gamma 0.2655722 0.2644907 0.03840752 0.1932829 0.3458471
```

```
par(mfrow = c(1,2))
plot(mcmc$samples[, 'beta'], type = "l")
abline(h=mean(mcmc$samples[,'beta']), col = "red")
plot(mcmc$samples[, 'gamma'], type = "l")
abline(h=mean(mcmc$samples[,'gamma']), col = "red")
```

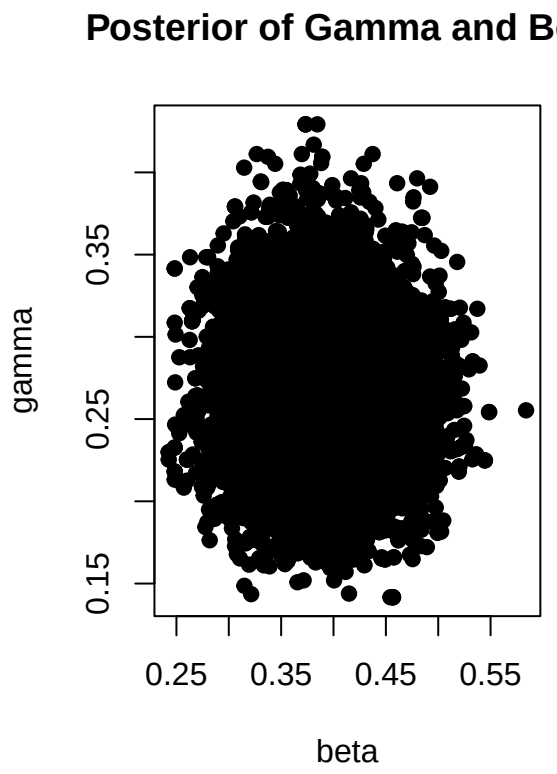





```
# Compute posterior correlation
df = data.frame(beta = mcmc$samples[,1], gamma = mcmc$samples[,2])
cor(df)
```

```
##              beta       gamma
## beta  1.000000000 0.008835179
## gamma 0.008835179 1.000000000
```

```
# Plot
plot(mcmc$samples[, 'beta'], mcmc$samples[, 'gamma'], xlab = 'beta', ylab = 'gamma',
     pch = 19, main = "Posterior of Gamma and Beta")
```



part ii)

```
SIR_code <- nimbleCode({
  S[1] <- N - I0 - R0
  I[1] <- I0
  R[1] <- R0
  probIR <- 1 - exp(-gamma)
  ### loop over time
  for(t in 1:tau) {
    probSI[t] <- 1 - exp(- beta * I[t] / N)
    Istar[t] ~ dbin(probSI[t], S[t])
    Rstar[t] ~ dbin(probIR, I[t])
    # update S, I, R
    S[t + 1] <- S[t] - Istar[t]
    I[t + 1] <- I[t] + Istar[t] - Rstar[t]
    R[t + 1] <- R[t] + Rstar[t]
}
# priors
beta ~ dexp(100)
gamma ~ dunif(0,10)
})
```

```
constantsList <- list(N = 1000, I0 = 5, R0 = 0, tau = 17)
dataList <- list(Istar = new_cases, Rstar = new_removals)
sirModel <- nimbleModel(SIR_code, constants = constantsList, data = dataList)
```

```
## Defining model

## Building model

## Setting data and initial values

## Running calculate on model
##   [Note] Any error reports that follow may simply reflect missing values in model variables.

## Checking model sizes and dimensions

##   [Note] This model is not fully initialized. This is not an error.
##          To see which variables are not initialized, use model$initializeInfo().
##          For more information on model initialization, see help(modelInitialization).
```

```
initisList = list(beta = runif(1,0,1), gamma = runif(1,0,10))
sirModel$setInits(initisList)
# exclude data from parent nodes
dataNodes <- c('Istar', 'Rstar')
dataNodes <- sirModel$expandNodeNames(dataNodes, returnScalarComponents = TRUE)
dataNodes
```

```
##  [1] "Istar[1]"  "Istar[2]"  "Istar[3]"  "Istar[4]"  "Istar[5]"  "Istar[6]"
##  [7] "Istar[7]"  "Istar[8]"  "Istar[9]"  "Istar[10]" "Istar[11]" "Istar[12]"
## [13] "Istar[13]" "Istar[14]" "Istar[15]" "Istar[16]" "Istar[17]" "Rstar[1]"
## [19] "Rstar[2]"  "Rstar[3]"  "Rstar[4]"  "Rstar[5]"  "Rstar[6]"  "Rstar[7]"
## [25] "Rstar[8]"  "Rstar[9]"  "Rstar[10]" "Rstar[11]" "Rstar[12]" "Rstar[13]"
## [31] "Rstar[14]" "Rstar[15]" "Rstar[16]" "Rstar[17]"
```

```
parentNodes <- sirModel$getParents(dataNodes, stochOnly = TRUE)
parentNodes
```

```
##  [1] "beta"      "gamma"     "Istar[1]"  "Rstar[1]"  "Rstar[2]"  "Istar[2]"
##  [7] "Rstar[3]"  "Istar[3]"  "Rstar[4]"  "Istar[4]"  "Rstar[5]"  "Istar[5]"
## [13] "Rstar[6]"  "Istar[6]"  "Rstar[7]"  "Istar[7]"  "Rstar[8]"  "Istar[8]"
## [19] "Rstar[9]"  "Istar[9]"  "Rstar[10]" "Istar[10]" "Rstar[11]" "Istar[11]"
## [25] "Rstar[12]" "Istar[12]" "Rstar[13]" "Istar[13]" "Rstar[14]" "Istar[14]"
## [31] "Rstar[15]" "Istar[15]" "Rstar[16]" "Istar[16]"
```

```
parentNodes <- parentNodes[-which(parentNodes %in% dataNodes)]
parentNodes
```

```
## [1] "beta"  "gamma"
```

```
# parentNodes # beta gamma
parentNodes <- sirModel$expandNodeNames(parentNodes, returnScalarComponents = TRUE)
nodesToSim <- sirModel$getDependencies(parentNodes, self = FALSE, downstream = T)
nodesToSim
```

```
##   [1] "probSI[1]"  "probIR"     "Istar[1]"   "Rstar[1]"   "S[2]"
##   [6] "I[2]"       "R[2]"       "probSI[2]"  "Rstar[2]"   "Istar[2]"
##  [11] "R[3]"       "S[3]"       "I[3]"       "probSI[3]"  "Rstar[3]"
##  [16] "Istar[3]"   "R[4]"       "S[4]"       "I[4]"       "probSI[4]"
##  [21] "Rstar[4]"   "Istar[4]"   "R[5]"       "S[5]"       "I[5]"
##  [26] "probSI[5]"  "Rstar[5]"   "Istar[5]"   "R[6]"       "S[6]"
##  [31] "I[6]"       "probSI[6]"  "Rstar[6]"   "Istar[6]"   "R[7]"
##  [36] "S[7]"       "I[7]"       "probSI[7]"  "Rstar[7]"   "Istar[7]"
##  [41] "R[8]"       "S[8]"       "I[8]"       "probSI[8]"  "Rstar[8]"
##  [46] "Istar[8]"   "R[9]"       "S[9]"       "I[9]"       "probSI[9]"
##  [51] "Rstar[9]"   "Istar[9]"   "R[10]"      "S[10]"      "I[10]"
##  [56] "probSI[10]" "Rstar[10]"  "Istar[10]"  "R[11]"      "S[11]"
##  [61] "I[11]"      "probSI[11]" "Rstar[11]"  "Istar[11]"  "R[12]"
##  [66] "S[12]"      "I[12]"      "probSI[12]" "Rstar[12]"  "Istar[12]"
##  [71] "R[13]"      "S[13]"      "I[13]"      "probSI[13]" "Rstar[13]"
##  [76] "Istar[13]"  "R[14]"      "S[14]"      "I[14]"      "probSI[14]"
##  [81] "Rstar[14]"  "Istar[14]"  "R[15]"      "S[15]"      "I[15]"
##  [86] "probSI[15]" "Rstar[15]"  "Istar[15]"  "R[16]"      "S[16]"
##  [91] "I[16]"      "probSI[16]" "Rstar[16]"  "Istar[16]"  "R[17]"
##  [96] "S[17]"      "I[17]"      "probSI[17]" "Rstar[17]"  "Istar[17]"
## [101] "R[18]"      "S[18]"      "I[18]"
```

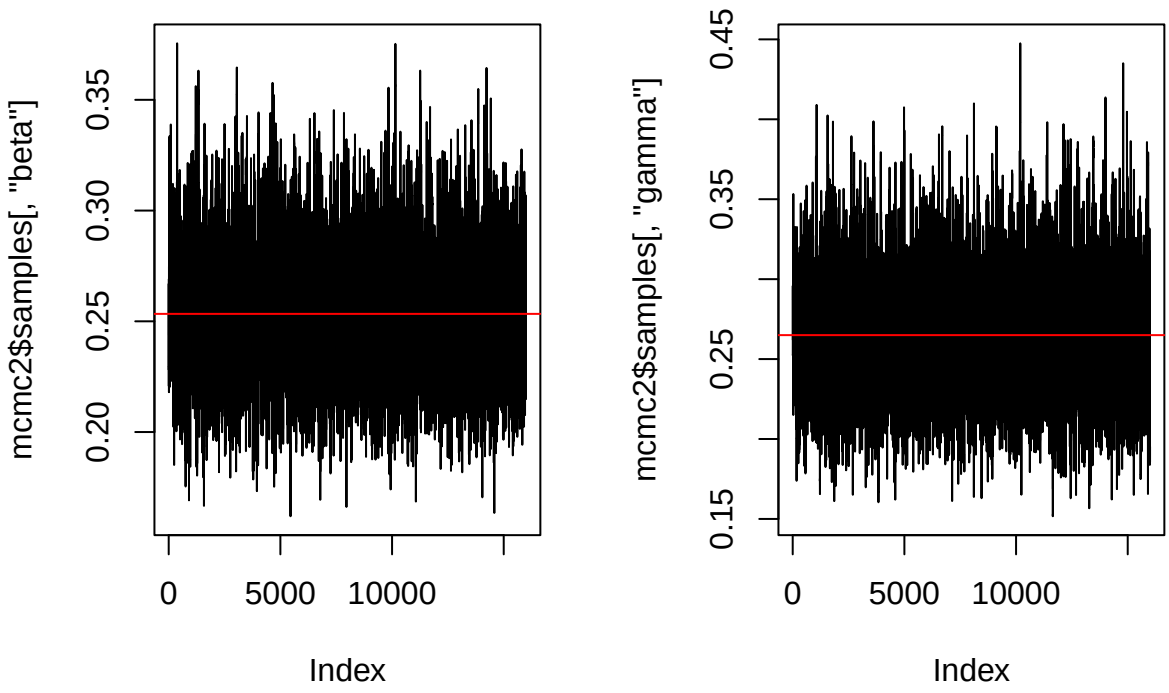
```
mcmc2 <- nimbleMCMC(sirModel, nodesToSim, summary = TRUE,
                    nchains = 1, thin = 1, niter = 20000, nburnin = 4000)
```

```
## Compiling
##   [Note] This may take a minute.
##   [Note] Use 'showCompilerOutput = TRUE' to see C++ compilation details.

## running chain 1...

## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
```

```
# summarize results
mcmc2$summary
```

```
##            Mean    Median    St.Dev. 95%CI_low 95%CI_upp
## beta  0.2533586 0.2525870 0.02971439 0.1993145 0.3150011
## gamma 0.2649806 0.2626608 0.03894670 0.1940661 0.3455451
```

```
par(mfrow = c(1,2))
plot(mcmc2$samples[, 'beta'], type = "l")
abline(h=mean(mcmc2$samples[,'beta']), col = "red")
plot(mcmc2$samples[, 'gamma'], type = "l")
abline(h=mean(mcmc2$samples[,'gamma']), col = "red")
```



As we can observe, the γ posterior estimate has no noticeable change due to the change of prior of β, I think this makes sense because γ and β have no correlation as their scatter plot is a circle-like shape, not a line.

part iii)

Under the posterior mean

```
library(nimble)
data = read.csv("A2Epidemic1.csv", header = TRUE)
# head(data)
# create Istar and Rstar
new_cases <- data$NewCases
new_removals <- data$NewRemovals

SIR_code <- nimbleCode({
S[1] <- N - I17 - R17
I[1] <- I17
R[1] <- R17

probIR <- 1 - exp(-gamma)
### loop over time
for(t in 1:tau) {
  probSI[t] <- 1 - exp(- beta * I[t] / N)
```

```
  Istar[t] ~ dbin(probSI[t], S[t])
  Rstar[t] ~ dbin(probIR, I[t])
  # update S, I, R
  S[t + 1] <- S[t] - Istar[t]
  I[t + 1] <- I[t] + Istar[t] - Rstar[t]
  R[t + 1] <- R[t] + Rstar[t]
}
# priors
## beta_posterior_mean 0.3850350
beta ~ dnorm(0.3850350, 1e-3)
## gamma_posteror_mean 0.2654375
gamma ~ dnorm(0.2654375, 1e-3)
})
```

```
constantsList <- list(N=1000, I17=32, R17=46, tau=4)
initialList <- list(beta = 0.3850350, gamma = 0.2654375)

sirModel <- nimbleModel(SIR_code, constants = constantsList, inits = initialList)
```

```
## Defining model

## Building model

## Setting data and initial values

## Running calculate on model
##   [Note] Any error reports that follow may simply reflect missing values in model variables.

## Checking model sizes and dimensions

##   [Note] This model is not fully initialized. This is not an error.
##          To see which variables are not initialized, use model$initializeInfo().
##          For more information on model initialization, see help(modelInitialization).
```

```
dataNodes <- c('Istar', 'Rstar')

# exclude data from parent nodes
dataNodes <- c('Istar', 'Rstar')
dataNodes <- sirModel$expandNodeNames(dataNodes, returnScalarComponents = TRUE)
parentNodes <- sirModel$getParents(dataNodes, stochOnly = TRUE)
parentNodes
```

```
## [1] "beta"     "gamma"    "Istar[1]" "Rstar[1]" "Rstar[2]" "Istar[2]" "Rstar[3]"
## [8] "Istar[3]"
```

```
parentNodes <- parentNodes[-which(parentNodes %in% dataNodes)]
parentNodes
```

```
## [1] "beta"  "gamma"
```

```
# parentNodes # beta gamma
parentNodes <- sirModel$expandNodeNames(parentNodes, returnScalarComponents = TRUE)
parentNodes
```

```
## [1] "beta"  "gamma"
```

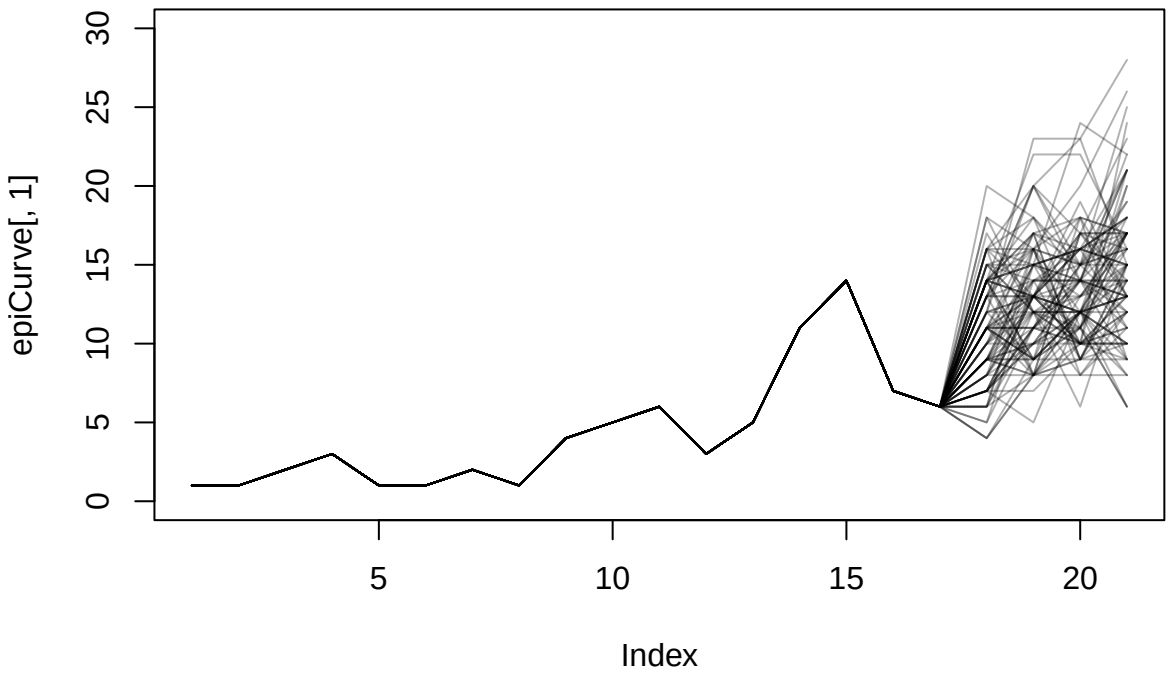
```
nodesToSim <- sirModel$getDependencies(parentNodes, self = FALSE, downstream = T)
nodesToSim
```

```
##  [1] "probSI[1]" "probIR"    "Istar[1]"  "Rstar[1]"  "S[2]"      "I[2]"
##  [7] "R[2]"      "probSI[2]" "Rstar[2]"  "Istar[2]"  "R[3]"      "S[3]"
## [13] "I[3]"      "probSI[3]" "Rstar[3]"  "Istar[3]"  "R[4]"      "S[4]"
## [19] "I[4]"      "probSI[4]" "Rstar[4]"  "Istar[4]"  "R[5]"      "S[5]"
## [25] "I[5]"
```

```
# mcmc <- nimbleMCMC(sirModel, parentNodes, summary = TRUE, nchains = 1, thin = 1,
#                    niter = 1000, nburnin = 100)
```

```
# mcmc$summary
```

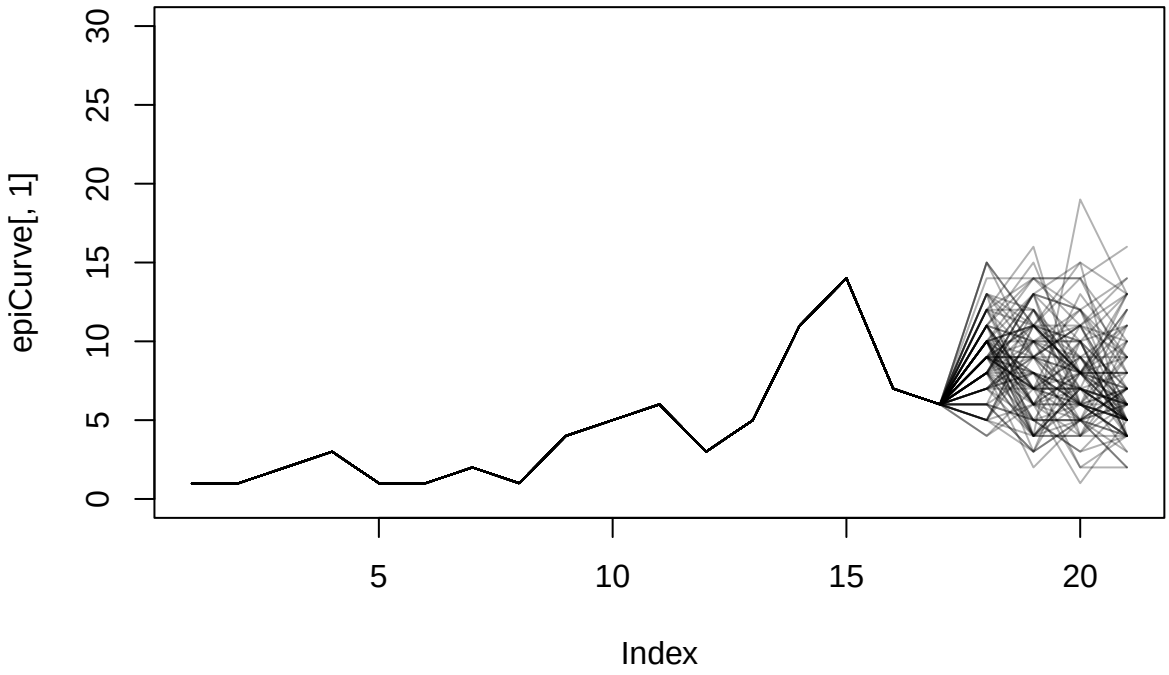
```
nSim <- 100
# Create storage for epidemic curve simulations
epiCurve <- matrix(NA, nrow = 21, ncol = nSim)

# Loop over simulations
for (i in 1:nSim) {

  sirModel$simulate(nodesToSim, includeData = TRUE)
  epiCurve[,i] <- c(new_cases, sirModel$Istar[1:4])
}

plot(epiCurve[,1], type = 'l', col = adjustcolor('black',alpha = 0.3), ylim = c(0, 30))
for (i in 2:nSim) {
  lines(epiCurve[,i], col = adjustcolor('black', alpha = 0.3))
}
```



From the posterior predictive distribution

```
set.seed(120)
# sample() is a versatile function in R that can be used to randomly sample elements
# from a vector or set of values. replace = FALSE indicates sample without replacement

my_vec <- c(1:10000)
# sample 100 values randomly from my_vec
Random <- as.array(sample(my_vec, 100, replace = FALSE))

nSim <- 100
# Create storage for epidemic curve simulations
epiCurve <- matrix(NA, nrow = 21, ncol = nSim)

# Loop over simulations
for (i in 1:nSim) {
  initialList <- list(beta=mcmc$samples[Random[i], 'beta'],
                      gamma=mcmc$samples[Random[i], 'gamma'])
  sirModel$setInits(initisList)

  sirModel$simulate(nodesToSim, includeData = TRUE)
  epiCurve[,i] <- c(new_cases, sirModel$Istar[1:4])
}
```

```
plot(epiCurve[,1], type = 'l', col = adjustcolor('black',alpha = 0.3), ylim = c(0, 30))
for (i in 2:nSim) {
  lines(epiCurve[,i], col = adjustcolor('black', alpha = 0.3))
}
```



From these two plots, I find that the discrepancy/difference between forecasts under the full posterior distribution seems to be larger than the difference between forecasts under the posterior mean. This is because we set a fixed value for posterior mean forecasting, i.e., the prior is fixed; however, for full posterior distribution, we have pooled 100 samples from the 10000 samples of beta and gamma, which add more randomness to the forecast. So we can observe a larger discrepancy under the full posterior distribution.

part iv)

```
my_vec <- c(1:10000)
# sample 100 values randomly from my_vec
Random <- as.array(sample(my_vec, 100, replace = FALSE))

nSim <- 100
# Create storage for epidemic curve simulations
infectious <- matrix(NA, nrow = 90, ncol = nSim)
```

```
SIR_code <- nimbleCode({
  S[1] <- N - I17 - R17
  I[1] <- I17
  R[1] <- R17
  probIR <- 1 - exp(-gamma)
  ### loop over time
  for(t in 1:tau) {
    probSI[t] <- 1 - exp(- beta * I[t] / N)
    Istar[t] ~ dbin(probSI[t], S[t])
    Rstar[t] ~ dbin(probIR, I[t])
    # update S, I, R
    S[t + 1] <- S[t] - Istar[t]
    I[t + 1] <- I[t] + Istar[t] - Rstar[t]
    R[t + 1] <- R[t] + Rstar[t]
}
# priors
beta ~ dunif(0,10)
gamma ~ dunif(0,10)
})
```

```
constantsList <- list(N=1000, I17=32, R17=46, tau=75)

sirModel <- nimbleModel(SIR_code, constants = constantsList)
```

```
## Defining model

## Building model

## Running calculate on model
##   [Note] Any error reports that follow may simply reflect missing values in model variables.

## Checking model sizes and dimensions

##   [Note] This model is not fully initialized. This is not an error.
##          To see which variables are not initialized, use model$initializeInfo().
##          For more information on model initialization, see help(modelInitialization).
```

```
initisList = list(beta = runif(1,0,1), gamma = runif(1,0,10))
sirModel$setInits(initisList)
```

```
dataNodes <- c('Istar', 'Rstar')

# exclude data from parent nodes
dataNodes <- c('Istar', 'Rstar')
dataNodes <- sirModel$expandNodeNames(dataNodes, returnScalarComponents = TRUE)
parentNodes <- sirModel$getParents(dataNodes, stochOnly = TRUE)
parentNodes <- parentNodes[-which(parentNodes %in% dataNodes)]
# parentNodes # beta gamma
parentNodes <- sirModel$expandNodeNames(parentNodes, returnScalarComponents = TRUE)
nodesToSim <- sirModel$getDependencies(parentNodes, self = FALSE, downstream = T)

# Loop over simulations
for (i in 1:nSim) {
  initialList <- list(beta=mcmc$samples[Random[i], 'beta'],
                      gamma=mcmc$samples[Random[i], 'gamma'])
  sirModel$setInits(initialList)

  sirModel$simulate(nodesToSim, includeData = TRUE)
  infectious[,i] <- c(part_i_I, sirModel$I[1:72])
}
```

```
dataList <- list(Istar = sirModel$Istar, Rstar = sirModel$Rstar)
constantsList <- list(N=1000, I17=32, R17=46, tau=75)
sirModel <- nimbleModel(SIR_code, constants = constantsList, data = dataList)
```

```
## Defining model

## Building model

## Setting data and initial values

## Running calculate on model
##   [Note] Any error reports that follow may simply reflect missing values in model variables.

## Checking model sizes and dimensions

##   [Note] This model is not fully initialized. This is not an error.
##          To see which variables are not initialized, use model$initializeInfo().
##          For more information on model initialization, see help(modelInitialization).
```

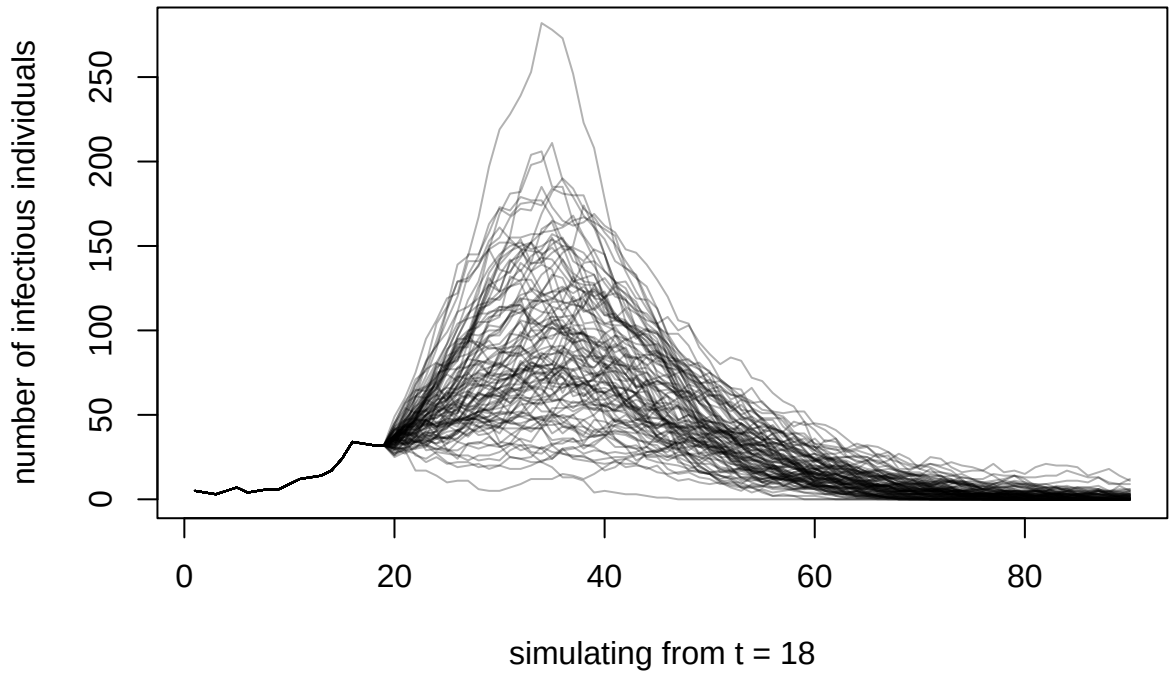
```
mcmc.iv <- nimbleMCMC(sirModel, nodesToSim, summary = TRUE, nchains = 1, thin = 1,
                  niter = 2000, nburnin = 100)
```

```
## Compiling
##   [Note] This may take a minute.
##   [Note] Use 'showCompilerOutput = TRUE' to see C++ compilation details.

## running chain 1...

## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
```

```
mcmc.iv$summary
```

```
##            Mean    Median    St.Dev. 95%CI_low 95%CI_upp
## beta  0.3588465 0.3594596 0.01498329 0.3292226 0.3884636
## gamma 0.2644835 0.2649036 0.01343466 0.2403580 0.2877906
```

```
plot(infectious[,1], type = 'l', col = adjustcolor('black',alpha = 0.3), ylim = c(0, 280),
     xlab = "simulating from t = 18", ylab = "number of infectious individuals")
for (i in 2:nSim) {
  lines(infectious[,i], col = adjustcolor('black', alpha = 0.3))
}
```




```
time.of.peak = apply(infectious, 1, max)
```

```
# Calculate the posterior mean of time.of.peak
posterior_mean <- mean(time.of.peak)

# Calculate the 95% credible interval of peak.sim
credible_interval <- quantile(time.of.peak, c(0.025, 0.975))

# Print the results
cat("Posterior mean of time of the epidemic peak:", posterior_mean, "\n")
```

```
## Posterior mean of time of the epidemic peak: 73.62222
```

```r
cat("95% credible interval of time of the epidemic peak:",
    "(", credible_interval[1], ",",credible_interval[2], ")", "\n")
```

```
## 95% credible interval of time of the epidemic peak: ( 4.225 , 268.5 )
```

Question 3: Bayesian SIR Model with Behavioural-change

part a)

```
library(nimble)
dataset = read.csv("A2Epidemic2.csv")
head(dataset)
```

```
##   NewCases NewRemovals
## 1        2           2
## 2        2           3
## 3        4           0
## 4        5           2
## 5        2           5
## 6        4           1
```

```
# create Istar and Rstar
new_cases <- dataset$NewCases
new_removals <- dataset$NewRemovals
```

```
SIR_code <- nimbleCode({
  S[1] <- N - I0 - R0
  I[1] <- I0
  R[1] <- R0
  probIR <- 1 - exp(-gamma)
  ### loop over time
  for(t in 1:tau) {
    probSI[t] <- 1 - exp(- beta * I[t] / N)
    Istar[t] ~ dbin(probSI[t], S[t])
    Rstar[t] ~ dbin(probIR, I[t])
    # update S, I, R
    S[t + 1] <- S[t] - Istar[t]
    I[t + 1] <- I[t] + Istar[t] - Rstar[t]
    R[t + 1] <- R[t] + Rstar[t]
}
# priors
beta ~ dunif(0,10)
gamma ~ dunif(0,10)
})
```

```
constantsList <- list(N = 1000, I0 = 5, R0 = 0, tau = 100)
dataList <- list(Istar = new_cases, Rstar = new_removals)
sirModel <- nimbleModel(SIR_code, constants = constantsList, data = dataList)
```

```
## Defining model

## Building model

## Setting data and initial values
```

```
## Running calculate on model
##   [Note] Any error reports that follow may simply reflect missing values in model variables.

## Checking model sizes and dimensions

##   [Note] This model is not fully initialized. This is not an error.
##          To see which variables are not initialized, use model$initializeInfo().
##          For more information on model initialization, see help(modelInitialization).
```

```
initisList = list(beta = runif(1,0,1), gamma = runif(1,0,10))
sirModel$setInits(initisList)
# exclude data from parent nodes
dataNodes <- c('Istar', 'Rstar')
dataNodes <- sirModel$expandNodeNames(dataNodes, returnScalarComponents = TRUE)
parentNodes <- sirModel$getParents(dataNodes, stochOnly = TRUE)
parentNodes <- parentNodes[-which(parentNodes %in% dataNodes)]
# parentNodes # beta gamma
parentNodes <- sirModel$expandNodeNames(parentNodes, returnScalarComponents = TRUE)
nodesToSim <- sirModel$getDependencies(parentNodes, self = FALSE, downstream = T)
```

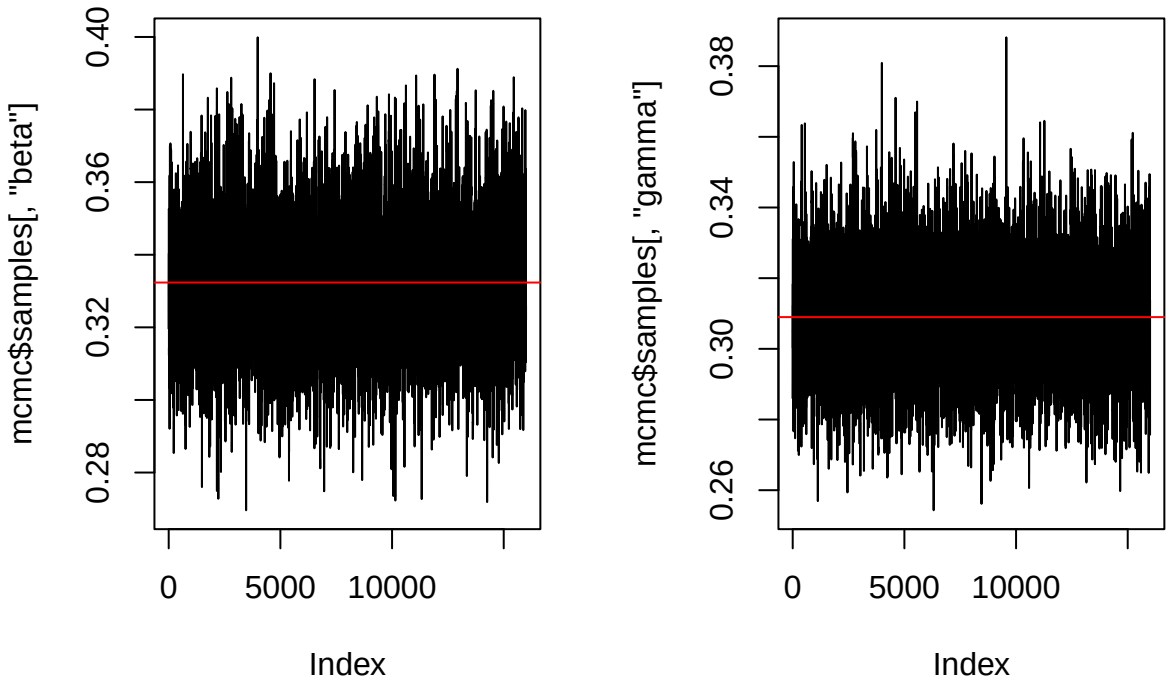
```
mcmc <- nimbleMCMC(sirModel, nodesToSim, summary = TRUE,
                   nchains = 1, thin = 1, niter = 20000, nburnin = 4000)
```

```
## Compiling
##   [Note] This may take a minute.
##   [Note] Use 'showCompilerOutput = TRUE' to see C++ compilation details.

## running chain 1...

## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
```

```
# summarize results
mcmc$summary
```

```
##            Mean    Median    St.Dev. 95%CI_low 95%CI_upp
## beta  0.3323437 0.3323271 0.01761787 0.2985412 0.3680631
## gamma 0.3089845 0.3084219 0.01621064 0.2780413 0.3413864
```

```
par(mfrow = c(1,2))
plot(mcmc$samples[, 'beta'], type = "l")
abline(h=mean(mcmc$samples[,'beta']), col = "red")
plot(mcmc$samples[, 'gamma'], type = "l")
abline(h=mean(mcmc$samples[,'gamma']), col = "red")
```

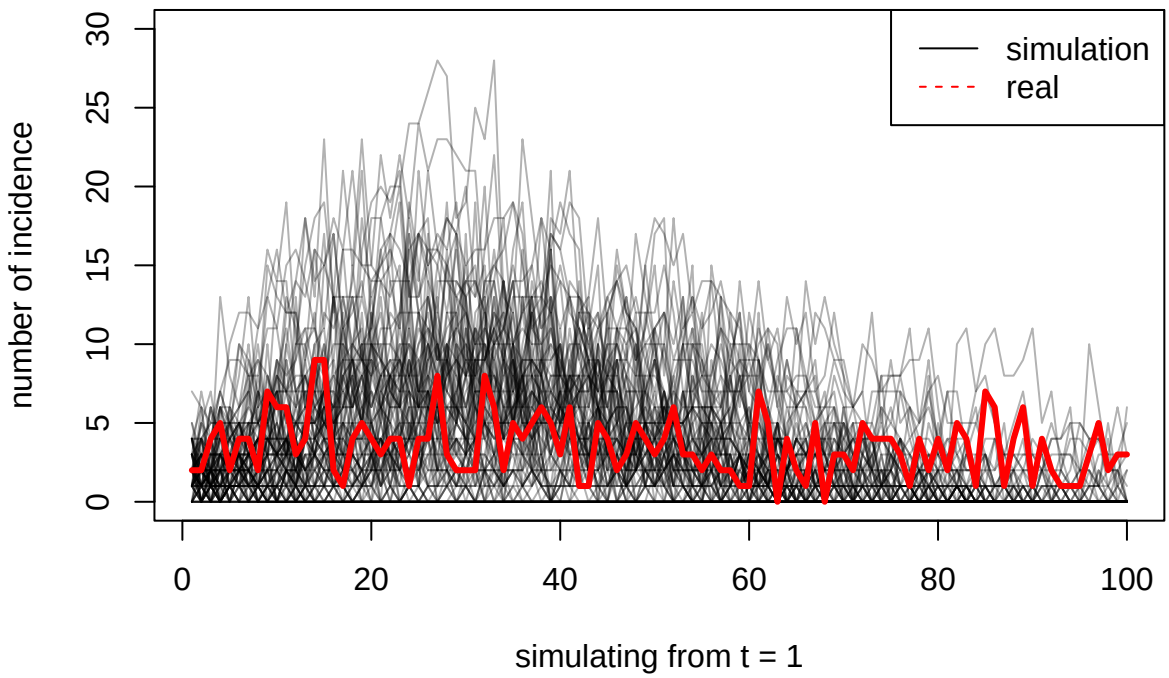


part ii)

```
my_vec <- c(1:10000)
# sample 100 values randomly from my_vec
Random <- as.array(sample(my_vec, 100, replace = FALSE))

nSim <- 100
# Create storage for epidemic curve simulations
incidence <- matrix(NA, nrow = 100, ncol = nSim)

# Loop over simulations
for (i in 1:nSim) {
  initialList <- list(beta=mcmc$samples[Random[i], 'beta'],
                      gamma=mcmc$samples[Random[i], 'gamma'])
  sirModel$setInits(initialList)

  sirModel$simulate(nodesToSim, includeData = TRUE)
  incidence[,i] <- sirModel$Istar
}
```

```
plot(incidence[,1], type = 'l', col = adjustcolor('black',alpha = 0.3), ylim = c(0, 30),
     xlab = "simulating from t = 1", ylab = "number of incidence")
for (i in 2:nSim) {
  lines(incidence[,i], col = adjustcolor('black', alpha = 0.3))
```

```
}
lines(new_cases, col = "red", lwd = 3)
legend('topright', legend = c("simulation", "real"), col = c("black", "red"), lty = 1:2)
```



part iii)

of course, we need to change our model / nimbleCode()

```
SIR_code <- nimbleCode({
S[1] <- N - I0 - R0
I[1] <- I0
R[1] <- R0
probIR <- 1 - exp(-gamma)
### loop over time
for(t in 1:tau) {
A[t] <- delta1*(1-exp(-delta2*I[t]))
probSI[t] <- 1 - exp(- beta * (1-A[t]) * I[t] / N)
Istar[t] ~ dbin(probSI[t], S[t])
Rstar[t] ~ dbin(probIR, I[t])
# update S, I, R
S[t + 1] <- S[t] - Istar[t]
I[t + 1] <- I[t] + Istar[t] - Rstar[t]
R[t + 1] <- R[t] + Rstar[t]
}
# priors
beta ~ dunif(0,10)
gamma ~ dunif(0,10)
delta1 ~ dunif(0,1)
delta2 ~ dunif(0,1)
})
```

part iv)

```
constantsList <- list(N = 1000, I0 = 5, R0 = 0, tau = 100)
dataList <- list(Istar = new_cases, Rstar = new_removals)
sirModel <- nimbleModel(SIR_code, constants = constantsList, data = dataList)
```

```
## Defining model

## Building model

## Setting data and initial values

## Running calculate on model
##   [Note] Any error reports that follow may simply reflect missing values in model variables.

## Checking model sizes and dimensions

##   [Note] This model is not fully initialized. This is not an error.
##          To see which variables are not initialized, use model$initializeInfo().
##          For more information on model initialization, see help(modelInitialization).
```

```
initisList = list(beta = runif(1,0,10), gamma = runif(1,0,10),
                  delta1 = runif(1,0,1), delta2 = runif(1,0,1))
sirModel$setInits(initisList)
```

```
dataNodes <- c('Istar', 'Rstar', 'A')
dataNodes <- sirModel$expandNodeNames(dataNodes, returnScalarComponents = TRUE)
parentNodes <- sirModel$getParents(dataNodes, stochOnly = TRUE)
parentNodes <- parentNodes[-which(parentNodes %in% dataNodes)]
parentNodes <- sirModel$expandNodeNames(parentNodes, returnScalarComponents = TRUE)
parentNodes # # beta gamma delta1 delta2
```

```
## [1] "beta"   "gamma"  "delta1" "delta2"
```

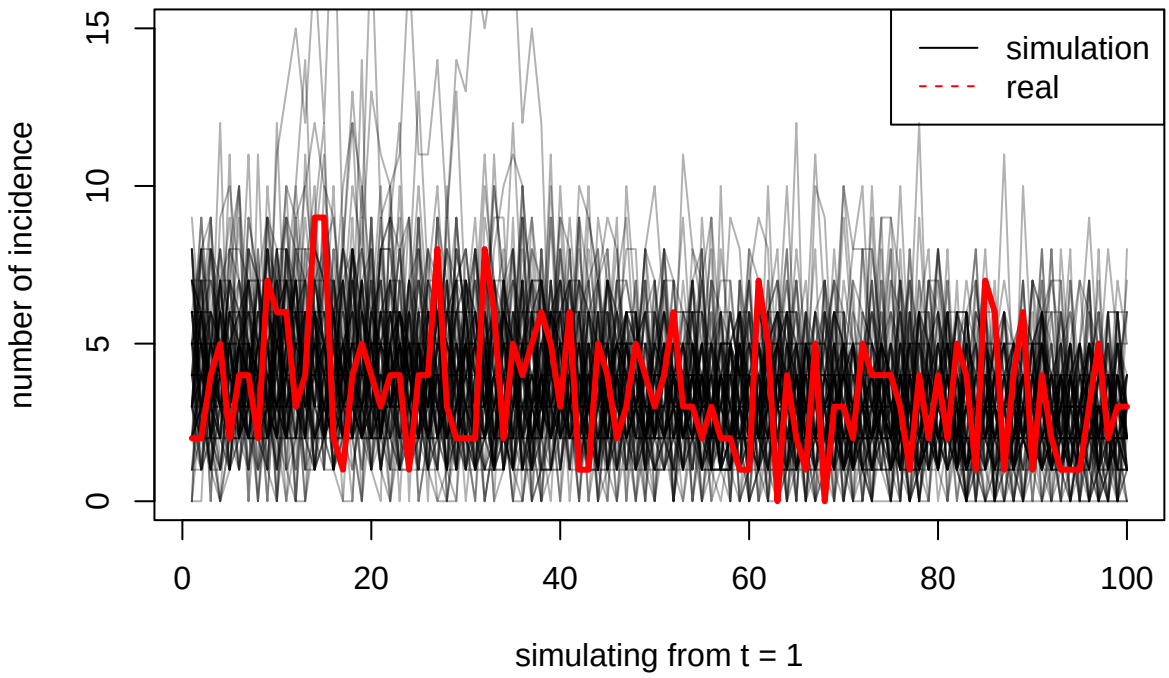
```
nodesToSim <- sirModel$getDependencies(parentNodes, self = FALSE, downstream = T)
```

```
mcmc <- nimbleMCMC(sirModel, nodesToSim, summary = TRUE,
                   nchains = 1, thin = 1, niter = 20000, nburnin = 4000)
```

```
## Compiling
##   [Note] This may take a minute.
##   [Note] Use 'showCompilerOutput = TRUE' to see C++ compilation details.

## running chain 1...

## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
```

```r
my_vec <- c(1:10000)
# sample 100 values randomly from my_vec
Random <- as.array(sample(my_vec, 100, replace = FALSE))

nSim <- 100
# Create storage for epidemic curve simulations
incidence <- matrix(NA, nrow = 100, ncol = nSim)

# Loop over simulations
for (i in 1:nSim) {
  initialList <- list(beta=mcmc$samples[Random[i], 'beta'],
                      gamma=mcmc$samples[Random[i], 'gamma'],
                      delta1=mcmc$samples[Random[i], 'delta1'],
                      delta2=mcmc$samples[Random[i], 'delta2'])
  sirModel$setInits(initialList)

  sirModel$simulate(nodesToSim, includeData = TRUE)
  incidence[,i] <- sirModel$Istar
}
```

```r
plot(incidence[,1], type = 'l', col = adjustcolor('black',alpha = 0.3), ylim = c(0, 15),
     xlab = "simulating from t = 1", ylab = "number of incidence")
for (i in 2:nSim) {
  lines(incidence[,i], col = adjustcolor('black', alpha = 0.3))
}
lines(new_cases, col = "red", lwd = 3)
legend('topright', legend = c("simulation", "real"), col = c("black", "red"), lty = 1:2)
```



What I found is that without the implementation of behavioral change, the number of incidence is going to surge. So behavior change is important to be considered to control the spread of disease