Lab Report:Lab3-Group8

Udaya Shanker Mohanan Nair(udamo524), Uday Jain(udaja983)

2025-05-19

Assignment 1 Gibbs sampling for the logistic regression

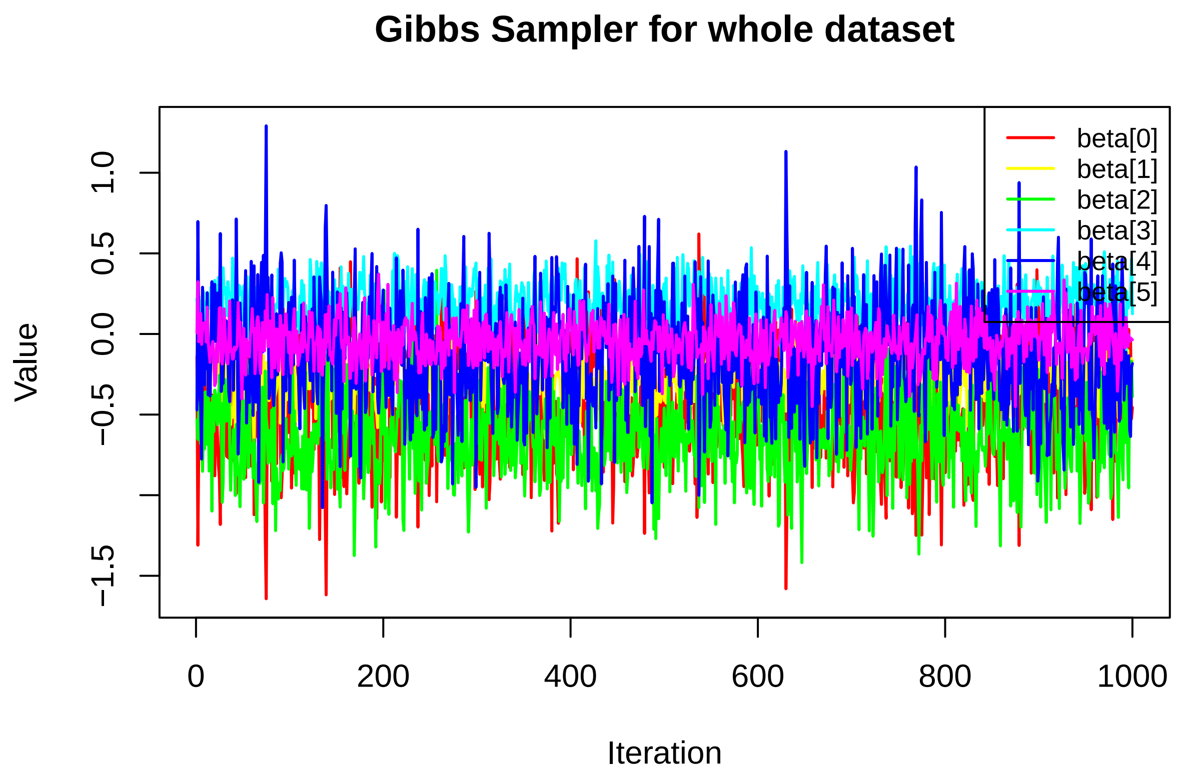
```r
##ASSIGNMENT 1

library(mvtnorm)
library(BayesLogit)

disease_data <- read.csv("Disease.csv")
x <- cbind(
  Intercept = 1,
  age_new = (disease_data$age -
               mean(disease_data$age))/sd(disease_data$age),
  gender = disease_data$gender,
  duration_new = (disease_data$duration_of_symptoms -
                    mean(disease_data$duration_of_symptoms))/sd(disease_data$duration_of_symptoms),
  dyspnoea = disease_data$dyspnoea,
  white_Blood_new = (disease_data$white_blood -
                       mean(disease_data$white_blood))/sd(disease_data$white_blood)
)

y <- disease_data$class_of_diagnosis
size <- nrow(x)
col_size <- ncol(x)

tau <- 3


gibbs_implementation <- function(x, y, n_iter=1000) {
  beta_samples <- matrix(0, n_iter, ncol(x))

  beta_value <- rep(0, ncol(x))
  size <- nrow(x)
  col_size <- ncol(x)

  prior_precision <- diag(1/tau^2, col_size)

  for (iter in 1:n_iter) {
    z <- x %*% beta_value
    poly_gamma <- rpg(size, 1, z)
    v <- solve(t(x) %*% diag(poly_gamma) %*% x + prior_precision)
    m <- v %*% t(x) %*% (y - 0.5)
    beta_value <- as.numeric(rmvnorm(1, m, v))
    beta_samples[iter,] <- beta_value
  }
  return(beta_samples)
}

set.seed(123)
results <- gibbs_implementation(x, y)

if_evaluation <- function(beta_values) {
  acf_value <- acf(beta_values, plot=FALSE, lag.max=50)$acf[,,1]
  if_value <- 1 + 2 * sum(acf_value[-1])
  return(if_value)
}

res_if <- apply(results, 2, if_evaluation)
cat("Inefficiency Factors for whole dataset:\n")
```

```
## Inefficiency Factors for whole dataset:
```

```r
print(res_if)
```

```
## [1] 1.7322117 0.9684822 0.9952840 0.6659212 2.2940092 0.4340299
```

```r
plot(1, type = "n",
     xlim = c(1, nrow(results)),
     ylim = range(results),
     xlab = "Iteration",
     ylab = "Value",
     main = "Gibbs Sampler for whole dataset")

cols <- rainbow(col_size)

for (i in 1:col_size) {
  lines(results[, i], col = cols[i], lwd = 1.5)
}
legend("topright",
       legend = paste0("beta[", 0:(col_size-1), "]"),
       col = cols,
       lty = 1,
       lwd = 1.5,
       cex = 0.8)
```



```
results_for_10_observations <- gibbs_implementation(x[1:10,], y[1:10])
results_for_40_observations <- gibbs_implementation(x[1:40,], y[1:40])
results_for_80_observations <- gibbs_implementation(x[1:80,], y[1:80])

res_if_10_observations <- apply(results_for_10_observations, 2, if_evaluation)
cat("Inefficiency Factors for 10 Observations:\n")
```

```
## Inefficiency Factors for 10 Observations:
```

```
print(res_if_10_observations)
```

```
## [1] 0.827587 1.420033 1.373011 2.115056 0.165961 2.412803
```

```
res_if_40_observations <- apply(results_for_40_observations, 2, if_evaluation)
cat("Inefficiency Factors for 40 Observations:\n")
```

```
## Inefficiency Factors for 40 Observations:
```

```
print(res_if_40_observations)
```

```
## [1] 1.8745666 1.0827795 1.9680445 0.2510922 1.7632683 2.3757253
```

```r
res_if_80_observations <- apply(results_for_80_observations, 2, if_evaluation)
cat("Inefficiency Factors for 80 Observations:\n")
```

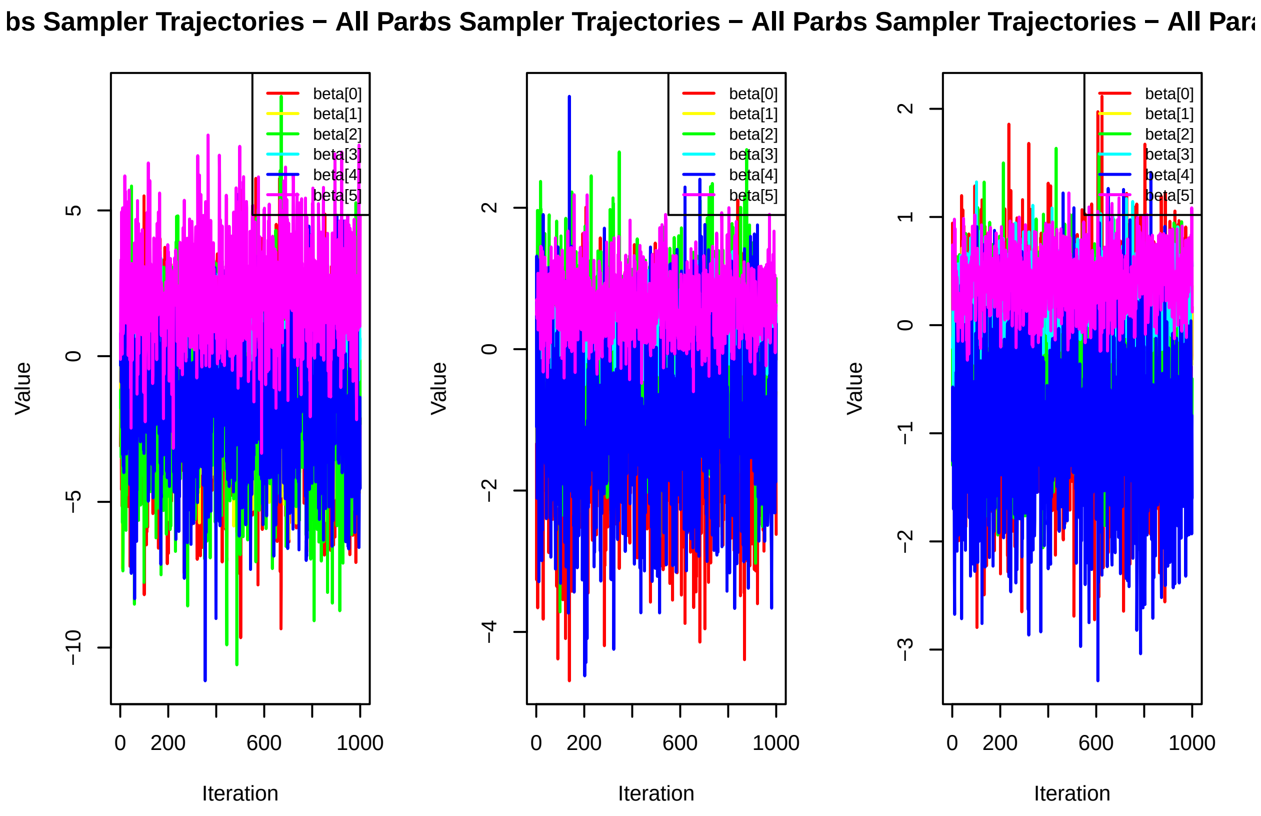
```
## Inefficiency Factors for 80 Observations:
```

```r
print(res_if_80_observations)
```

```
## [1] 1.3885638 1.0347889 1.3911514 0.9197756 1.6944566 3.0431779
```

```r
par(mfrow = c(1,3))
#10 Observations
plot(1, type = "n",
     xlim = c(1, nrow(results_for_10_observations)),
     ylim = range(results_for_10_observations),
     xlab = "Iteration",
     ylab = "Value",
     main = "Gibbs Sampler Trajectories - All Parameters")

cols <- rainbow(col_size)

for (i in 1:col_size) {
  lines(results_for_10_observations[, i], col = cols[i], lwd = 1.5)
}
legend("topright",
       legend = paste0("beta[", 0:(col_size-1), "]"),
       col = cols,
       lty = 1,
       lwd = 1.5,
       cex = 0.8)

#40 Observations

plot(1, type = "n",
     xlim = c(1, nrow(results_for_40_observations)),
     ylim = range(results_for_40_observations),
     xlab = "Iteration",
     ylab = "Value",
     main = "Gibbs Sampler Trajectories - All Parameters")

cols <- rainbow(col_size)

for (i in 1:col_size) {
  lines(results_for_40_observations[, i], col = cols[i], lwd = 1.5)
}
legend("topright",
       legend = paste0("beta[", 0:(col_size-1), "]"),
       col = cols,
       lty = 1,
       lwd = 1.5,
       cex = 0.8)

#80 Observations
```

```r
plot(1, type = "n",
     xlim = c(1, nrow(results_for_80_observations)),
     ylim = range(results_for_80_observations),
     xlab = "Iteration",
     ylab = "Value",
     main = "Gibbs Sampler Trajectories - All Parameters")

cols <- rainbow(col_size)

for (i in 1:col_size) {
  lines(results_for_80_observations[, i], col = cols[i], lwd = 1.5)
}
legend("topright",
       legend = paste0("beta[", 0:(col_size-1), "]"),
       col = cols,
       lty = 1,
       lwd = 1.5,
       cex = 0.8)
```



```r
par(mfrow = c(1,1))
```

Assignment 2 : Metropolis Random Walk for Poisson regression

```r
##ASSIGNMENT 2
library(mvtnorm)
library(MASS)
data<-read.table("eBayNumberOfBidderData_2025.dat",header=TRUE)
X <- model.matrix(~ PowerSeller + VerifyID + Sealed +
                    Minblem + MajBlem + LargNeg + LogBook +
                    MinBidShare, data = data)
y <- data$nBids
################################### Part A ###########################
poisson_glm<-glm(y ~ . -1,data=as.data.frame(X),family=poisson)
cat("Summary\n")
```

```
## Summary
```

```r
print(summary(poisson_glm))
```

```
##
## Call:
## glm(formula = y ~ . - 1, family = poisson, data = as.data.frame(X))
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## `(Intercept)`  1.08534    0.03592  30.213  < 2e-16 ***
## PowerSeller   -0.02833    0.04433  -0.639  0.52280
## VerifyID      -0.34902    0.13008  -2.683  0.00729 **
## Sealed         0.50101    0.06676   7.504 6.18e-14 ***
## Minblem       -0.12189    0.08388  -1.453  0.14619
## MajBlem       -0.24884    0.09865  -2.523  0.01165 *
## LargNeg        0.03610    0.07075   0.510  0.60987
## LogBook       -0.07280    0.03505  -2.077  0.03783 *
## MinBidShare   -1.76665    0.08292 -21.306  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 4131.37  on 700  degrees of freedom
## Residual deviance:  593.76  on 691  degrees of freedom
## AIC: 2529.5
##
## Number of Fisher Scoring iterations: 5
```

```r
cat("\nAs observed from summary, Intercept along with VerifyID, Sealed, \n
MajBlem, LogBook and MinBidShare are significant covariates.")
```

```
##
## As observed from summary, Intercept along with VerifyID, Sealed,
##
## MajBlem, LogBook and MinBidShare are significant covariates.
```

```r
#################################### Part B ############################
# Log-posterior function (with Zellner's g-prior)
log_posterior <- function(beta, X, y) {
  lambda <- exp(X %*% beta)       # exp(X ) = predicted Poisson rates
  log_lik <- sum(dpois(y,lambda,log=TRUE))     # Sum of log-likelihoods
  log_prior <- dmvnorm(beta,mean=rep(0,dim(X)[2]),
                      sigma=100*solve(t(X)%*%X),log=TRUE)
  return(log_lik + log_prior)
}
# Find posterior mode
optim_result<-optim(rep(0,dim(X)[2]),log_posterior,X=X,y=y,
                     control=list(fnscale=-1),hessian=TRUE)
beta_tilde<-optim_result$par
J_inv<-solve(-optim_result$hessian)  # Posterior covariance
names(beta_tilde)<-colnames(X)
colnames(J_inv)<-rownames(J_inv)<-colnames(X)
cat("\nBeta_tilde:\n")
```

```
##
## Beta_tilde:
```

```r
print(beta_tilde)
```

```
##  (Intercept)  PowerSeller     VerifyID       Sealed      Minblem      MajBlem
##  1.083356669 -0.007280161 -0.024583323  0.503016907 -0.043645091 -0.105898988
##      LargNeg      LogBook  MinBidShare
##  0.214272038 -0.066582591 -1.652945142
```

```r
cat("\nCovariance Matrix (Jy(-1) beta_tilde)\n")
```

```
##
## Covariance Matrix (Jy(-1) beta_tilde)
```

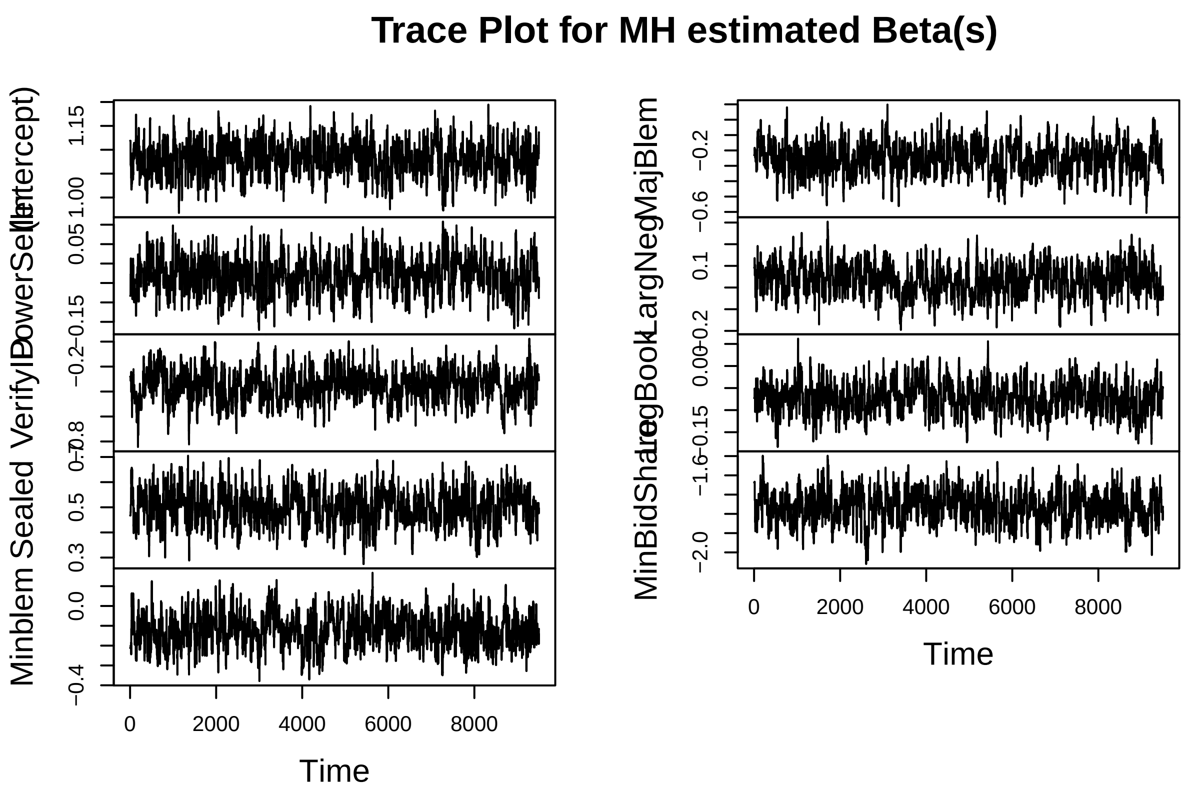
```r
print(J_inv)
```

```
##               (Intercept)   PowerSeller      VerifyID        Sealed
## (Intercept)  1.260971e-03 -0.0010014926 -0.0002880175 -0.0005185488
## PowerSeller -1.001493e-03  0.0019142227 -0.0001124431 -0.0001786808
## VerifyID    -2.880175e-04 -0.0001124431  0.0124865323 -0.0013641746
## Sealed      -5.185488e-04 -0.0001786808 -0.0013641746  0.0043005309
## Minblem     -6.064828e-04  0.0000841006  0.0001045665  0.0005164962
## MajBlem     -4.138960e-04 -0.0002287781  0.0003588459  0.0005360809
## LargNeg     -6.857255e-04  0.0003631365  0.0003997369  0.0004405196
## LogBook      3.154009e-05  0.0002124258 -0.0002575469 -0.0000768578
## MinBidShare  1.325391e-03 -0.0007370076 -0.0001983139 -0.0003699649
##                   Minblem       MajBlem       LargNeg       LogBook
## (Intercept) -6.064828e-04 -4.138960e-04 -6.857255e-04  3.154009e-05
## PowerSeller  8.410060e-05 -2.287781e-04  3.631365e-04  2.124258e-04
## VerifyID     1.045665e-04  3.588459e-04  3.997369e-04 -2.575469e-04
## Sealed       5.164962e-04  5.360809e-04  4.405196e-04 -7.685780e-05
## Minblem      6.430545e-03  4.962350e-04  8.833409e-05 -1.109472e-04
## MajBlem      4.962350e-04  8.650177e-03  5.783662e-04 -9.383533e-05
## LargNeg      8.833409e-05  5.783662e-04  4.405579e-03 -4.005484e-04
## LogBook     -1.109472e-04 -9.383533e-05 -4.005484e-04  1.186365e-03
## MinBidShare -3.698609e-04  3.474073e-04 -6.705869e-05  1.330474e-03
##               MinBidShare
## (Intercept)  1.325391e-03
## PowerSeller -7.370076e-04
## VerifyID    -1.983139e-04
## Sealed      -3.699649e-04
## Minblem     -3.698609e-04
## MajBlem      3.474073e-04
## LargNeg     -6.705869e-05
## LogBook      1.330474e-03
## MinBidShare  6.373720e-03
```

```
################################### Part C ############################
RWMSampler <- function(logPostFunc,theta_init,cov_prop,n_iter,burn_in,c,...) {
  # logPostFunc: Function to compute the log-posterior density. First argument
  # must be `theta`.
  # theta_init: Initial parameter vector.
  # cov_prop Proposal covariance matrix.
  # n_iter Total number of iterations.
  # burn_in Burn-in samples.
  # c Step size scaling factor.  # Tune for 25-30% acceptance
  p<-length(theta_init)
  theta<-matrix(NA, n_iter, p)
  theta[1, ]<-theta_init
  log_post_current<-logPostFunc(theta[1, ], ...)
  n_accept<-0
  for (i in 2:n_iter) {
    # Propose new theta
    theta_prop<-MASS::mvrnorm(1,mu=theta[i-1, ],Sigma=c*cov_prop)
    # Compute log-posterior at proposal
    # Log acceptance probability (avoid numerical overflow)
    log_alpha<-logPostFunc(theta_prop,...)-logPostFunc(theta[i-1, ], ...)
    if(log(runif(1)) < log_alpha){
      theta[i, ]<-theta_prop
      n_accept<-n_accept+1
    } else{
      theta[i, ]<-theta[i-1, ]
    }
  }
  theta<-theta[(burn_in+1):n_iter, ]
  acceptance_rate<-n_accept/n_iter
  cat("Acceptance rate:", acceptance_rate,"\n")
  return(theta)
}

LogPostPoisson <- function(theta, X, y) {
  lambda<-exp(X%*%theta)
  log_lik<-sum(dpois(y,lambda,log=TRUE))
  log_prior<-mvtnorm::dmvnorm(theta,mean=rep(0,ncol(X)),
                             sigma=100*solve(t(X)%*%X),log=TRUE)
  return(log_lik+log_prior)
}
samplesMH<-RWMSampler(logPostFunc=LogPostPoisson,theta_init=beta_tilde,
                      cov_prop=J_inv,n_iter=10000,burn_in=500,c=0.7,X=X,y=y)
```

```
## Acceptance rate: 0.2633
```

```
colnames(samplesMH)<-colnames(X)
plot.ts(samplesMH, main = "Trace Plot for MH estimated Beta(s)")
```



```
cat("\nSummary Results\n")
```

```
##
## Summary Results
```

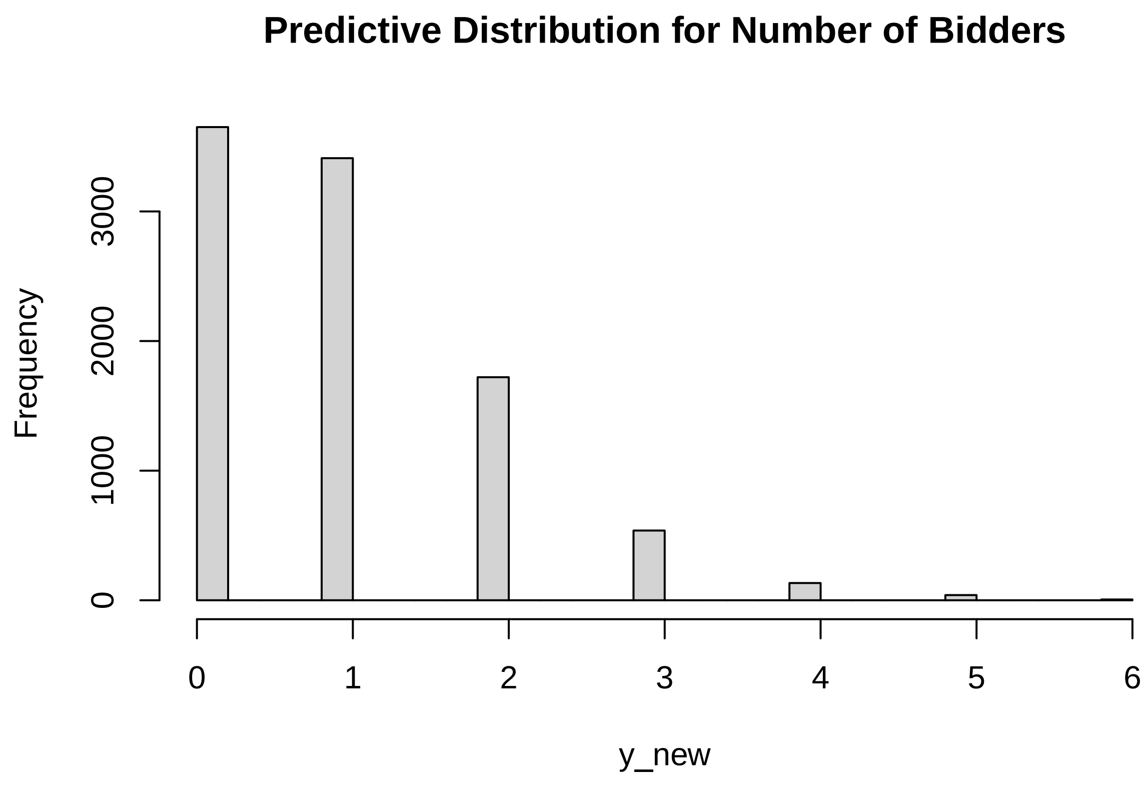
```
print(data.frame(Covariate=colnames(samplesMH),
           Mean=round(colMeans(samplesMH),4),
           Std=round(apply(samplesMH,2,sd),4)),row.names=FALSE)
```

```
##    Covariate    Mean    Std
##  (Intercept)  1.0804 0.0345
##  PowerSeller -0.0289 0.0454
##     VerifyID -0.3561 0.1259
##       Sealed  0.5009 0.0676
##      Minblem -0.1226 0.0834
##      MajBlem -0.2524 0.1013
##      LargNeg  0.0344 0.0709
##      LogBook -0.0719 0.0342
##  MinBidShare -1.7640 0.0815
```

```
cat("\nComparing the values with those in part B, we find while values for
    Intercept, Sealed , LogBook and MinBidShare are similar, other covariates
    show significant differences in values\n")
```

```
##
## Comparing the values with those in part B, we find while values for
##     Intercept, Sealed , LogBook and MinBidShare are similar, other covariates
##     show significant differences in values
```

```
#################################### Part D ############################
x_new<-c(1,1,0,1,0,1,0,1.3,0.7)
lambda_new<-exp(samplesMH %*% x_new)
y_new<-rpois(nrow(samplesMH),lambda_new)
hist(y_new,breaks=30,main="Predictive Distribution for Number of Bidders")
```



```
cat("\nProbability of no bidders (y_new = 0):",mean(y_new==0),"\n")
```

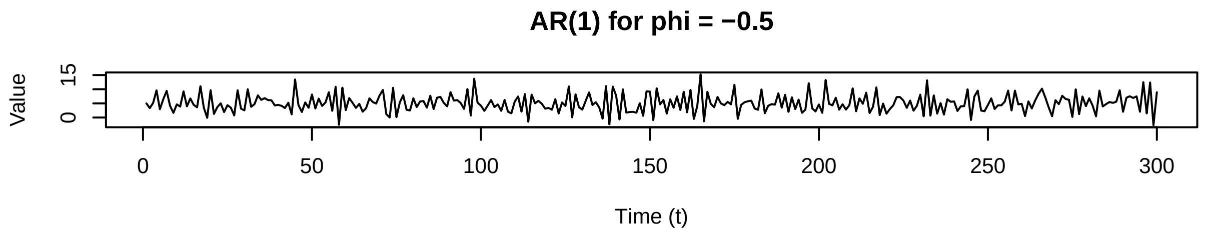
```
##
## Probability of no bidders (y_new = 0): 0.3843158
```

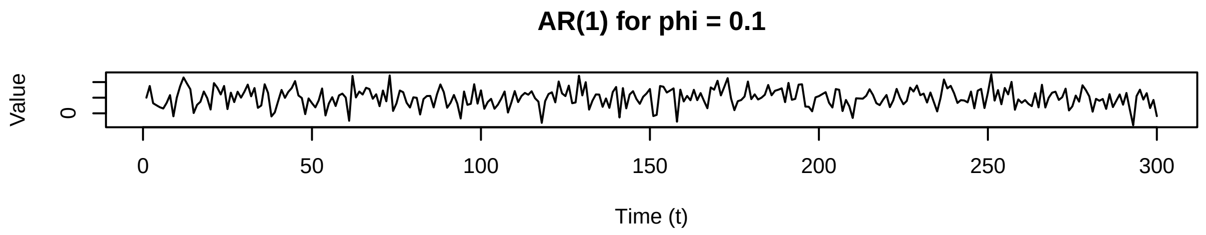
Assignment 3 : Time series models in Stan

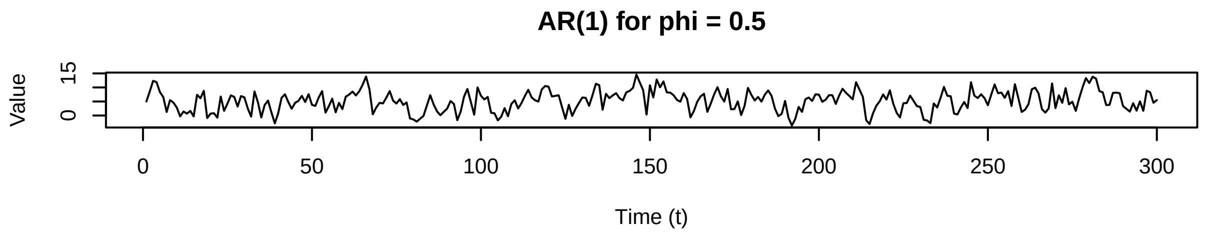
```r
library(rstan)
```

```
## Loading required package: StanHeaders

##
## rstan version 2.32.7 (Stan version 2.32.2)

## 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)
## For within-chain threading using `reduce_sum()` or `map_rect()` Stan functions,
## change `threads_per_chain` option:
## rstan_options(threads_per_chain = 1)
```

```r
library(ggplot2)
#Part A
ar1_simulation <- function(mu, phi, sigma_square, T) {
  results <- numeric(T)
  results[1] <- mu
  for (value in 2:T) {
    results[value] <- mu + phi * (results[value - 1] - mu) +
      rnorm(1, mean = 0, sd = sqrt(sigma_square))
  }
  return(results)
}

set.seed(123)
mu <- 5
sigma_square <- 9
T <- 300
phi_values <- c(-0.5, 0.1, 0.5)

par(mfrow = c(3,1))
for (phi_value in phi_values) {
  result <- ar1_simulation(mu, phi_value, sigma_square, T)
  plot(result, type = 'l', main = paste("AR(1) for phi =", phi_value),
       xlab = "Time (t)", ylab = "Value")
}
```







```r
par(mfrow = c(1,1))
```

for phi = -0.5, it shows an alternative behaviour this is because of negative autocorrelation.Tendency of the values to move in opposite direction from the previous value.

Now for phi = 0.1, when you compare nearby values, in most of the case current value doesn't much depend on the previous value. So therefore it is having weak autocorrelation.

Now for phi = 0.5,values are showing some tendency to change smoothly as the time move forward. so in a way it can be considered as a moderate positive autocorrelation.

```r
#Part B
x_results_phi4 <- ar1_simulation(mu, 0.4, sigma_square, T)
x_results_phi98 <- ar1_simulation(mu, 0.98, sigma_square, T)

stan_code <- "
  data {
    int<lower=1> T;
    vector[T] results;
  }

  parameters {
    real mu;
    real<lower=-1, upper=1> phi;
    real<lower=0> sigma;
  }
  model {
    results[1] ~ normal(mu, sigma);
    for (t in 2:T) {
      results[t] ~ normal(mu + phi * (results[t - 1] - mu), sigma);
    }
  }
"

stan_model <- stan_model(model_code = stan_code)
```

```
## Trying to compile a simple C file

## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## using C compiler: ‘Apple clang version 16.0.0 (clang-1600.0.26.6)’
## using SDK: ‘MacOSX15.2.sdk’
## clang -arch arm64 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Library/Framew
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/library/StanHeade
## In file included from /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/library/RcppEigen/
## In file included from /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/library/RcppEigen/
## /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/library/RcppEigen/include/Eigen/src/Core
##   679 | #include <cmath>
##       |          ^~~~~~~
## 1 error generated.
## make: *** [foo.o] Error 1
```

```
stan_code_data4 <- list(T = T, results = x_results_phi4)
stan_code_data98 <- list(T = T, results = x_results_phi98)

fit_phi4 <- sampling(stan_model, data = stan_code_data4, chains = 5, iter = 5000)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 4.4e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.44 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 1: Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1: Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1: Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 1: Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 1: Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 1: Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 1: Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 1: Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 1: Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 1: Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 1: Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 1:
```

```
## Chain 1:  Elapsed Time: 0.174 seconds (Warm-up)
## Chain 1:                0.194 seconds (Sampling)
## Chain 1:                0.368 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.4e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.14 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 2: Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 2: Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 2: Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 2: Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 2: Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 2: Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 2: Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 2: Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 2: Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 2: Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 2: Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.177 seconds (Warm-up)
## Chain 2:                0.197 seconds (Sampling)
## Chain 2:                0.374 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1.3e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.13 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 3: Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 3: Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 3: Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 3: Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 3: Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 3: Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 3: Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 3: Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 3: Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 3: Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 3: Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.182 seconds (Warm-up)
## Chain 3:                0.222 seconds (Sampling)
## Chain 3:                0.404 seconds (Total)
## Chain 3:
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1.4e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.14 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 4: Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 4: Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 4: Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 4: Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 4: Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 4: Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 4: Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 4: Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 4: Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 4: Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 4: Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.175 seconds (Warm-up)
## Chain 4:                0.208 seconds (Sampling)
## Chain 4:                0.383 seconds (Total)
## Chain 4:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 5).
## Chain 5:
## Chain 5: Gradient evaluation took 1.4e-05 seconds
## Chain 5: 1000 transitions using 10 leapfrog steps per transition would take 0.14 seconds.
## Chain 5: Adjust your expectations accordingly!
## Chain 5:
## Chain 5:
## Chain 5: Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 5: Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 5: Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 5: Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 5: Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 5: Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 5: Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 5: Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 5: Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 5: Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 5: Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 5: Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 5:
## Chain 5:  Elapsed Time: 0.173 seconds (Warm-up)
## Chain 5:                0.191 seconds (Sampling)
## Chain 5:                0.364 seconds (Total)
## Chain 5:
```

```r
fit_phi98 <- sampling(stan_model, data = stan_code_data98, chains = 5, iter = 5000)
```

```
##
```

```
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 1.6e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.16 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 1: Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1: Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1: Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 1: Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 1: Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 1: Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 1: Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 1: Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 1: Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 1: Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 1: Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.299 seconds (Warm-up)
## Chain 1:                0.234 seconds (Sampling)
## Chain 1:                0.533 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.4e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.14 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 2: Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 2: Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 2: Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 2: Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 2: Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 2: Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 2: Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 2: Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 2: Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 2: Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 2: Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.234 seconds (Warm-up)
## Chain 2:                0.247 seconds (Sampling)
## Chain 2:                0.481 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1.3e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.13 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 3: Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 3: Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 3: Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 3: Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 3: Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 3: Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 3: Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 3: Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 3: Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 3: Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 3: Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.295 seconds (Warm-up)
## Chain 3:                0.235 seconds (Sampling)
## Chain 3:                0.53 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1.4e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.14 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 4: Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 4: Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 4: Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 4: Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 4: Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 4: Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 4: Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 4: Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 4: Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 4: Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 4: Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.251 seconds (Warm-up)
## Chain 4:                0.228 seconds (Sampling)
## Chain 4:                0.479 seconds (Total)
## Chain 4:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 5).
## Chain 5:
## Chain 5: Gradient evaluation took 1.4e-05 seconds
## Chain 5: 1000 transitions using 10 leapfrog steps per transition would take 0.14 seconds.
## Chain 5: Adjust your expectations accordingly!
## Chain 5:
## Chain 5:
## Chain 5: Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 5: Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 5: Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 5: Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 5: Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 5: Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 5: Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 5: Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 5: Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 5: Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 5: Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 5: Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 5:
## Chain 5:  Elapsed Time: 0.26 seconds (Warm-up)
## Chain 5:                0.264 seconds (Sampling)
## Chain 5:                0.524 seconds (Total)
## Chain 5:
```

```
# (i)

cat("\nResults when phi is 0.4:\n")
```

```
##
## Results when phi is 0.4:
```

```
summary(fit_phi4, pars = c("mu", "phi", "sigma"), probs = c(0.025, 0.975))
```

```
## $summary
##            mean      se_mean         sd      2.5%    97.5%    n_eff      Rhat
## mu    5.1340432 0.0025486448 0.27423506 4.5907821 5.666901 11577.83 0.9998909
## phi   0.3445944 0.0005120374 0.05562645 0.2358653 0.453538 11802.10 0.9999425
## sigma 3.0546286 0.0011449725 0.12663684 2.8212190 3.314707 12232.92 0.9997858
##
## $c_summary
## , , chains = chain:1
##
##          stats
## parameter      mean         sd      2.5%     97.5%
##     mu    5.1389353 0.28126517 4.5780136 5.6719021
##     phi   0.3443423 0.05454264 0.2417256 0.4515201
##     sigma 3.0523646 0.12853954 2.8140168 3.3276212
##
## , , chains = chain:2
##
##          stats
## parameter      mean         sd      2.5%    97.5%
##     mu    5.1351944 0.27410043 4.5946128 5.651836
##     phi   0.3426123 0.05656884 0.2325967 0.450022
##     sigma 3.0561410 0.12779740 2.8204814 3.330175
##
## , , chains = chain:3
##
##          stats
## parameter      mean         sd     2.5%     97.5%
##     mu    5.1312679 0.27795155 4.579051 5.6976396
##     phi   0.3453526 0.05567862 0.235404 0.4518877
##     sigma 3.0545158 0.12898244 2.813916 3.3208158
##
## , , chains = chain:4
##
##          stats
## parameter      mean         sd      2.5%     97.5%
##     mu    5.1371386 0.27221281 4.6023174 5.6648891
##     phi   0.3451877 0.05478377 0.2356945 0.4533443
##     sigma 3.0554572 0.12150972 2.8272143 3.2976894
##
## , , chains = chain:5
##
##          stats
## parameter      mean         sd      2.5%     97.5%
##     mu    5.1276796 0.26544987 4.6134637 5.6573348
##     phi   0.3454772 0.05651952 0.2358665 0.4583193
##     sigma 3.0546641 0.12627875 2.8247586 3.2993513
```

```r
cat("\nResults when phi is 0.98:\n")
```

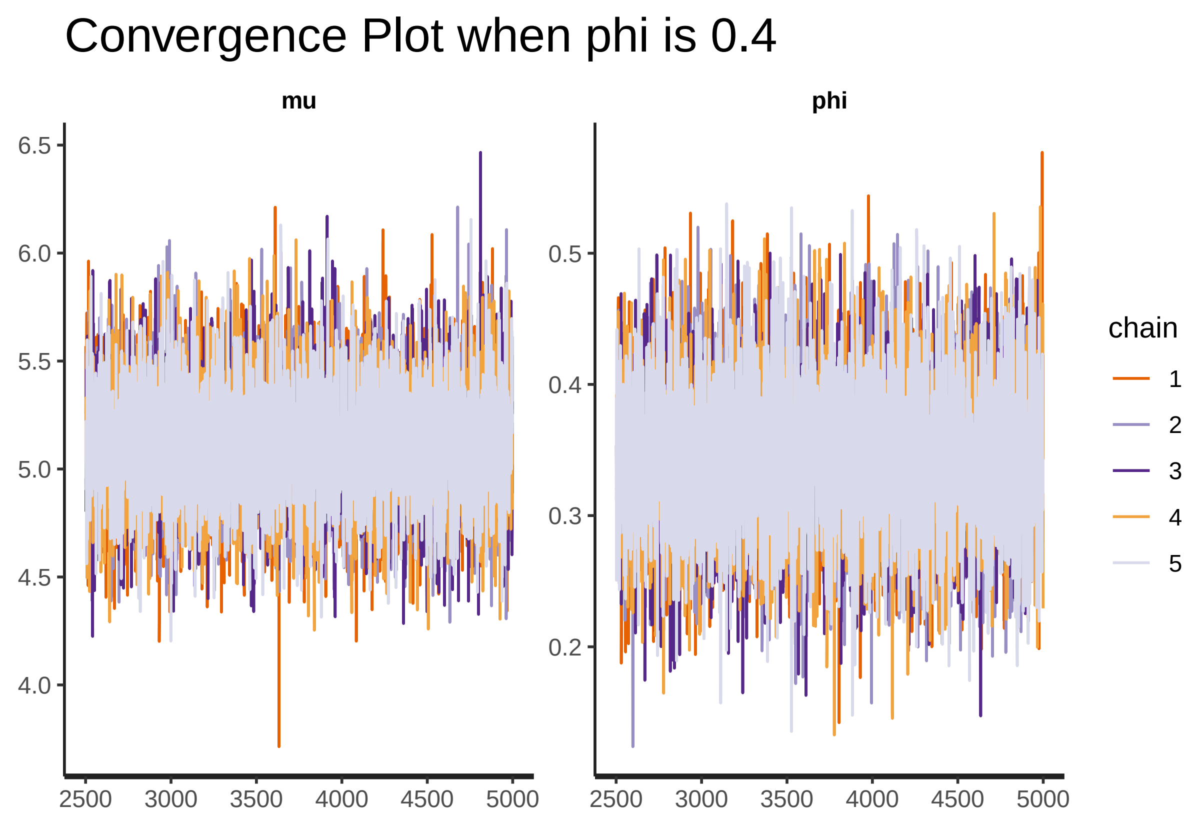
```
##
## Results when phi is 0.98:
```

```r
summary(fit_phi98, pars = c("mu", "phi", "sigma"), probs = c(0.025, 0.975))
```

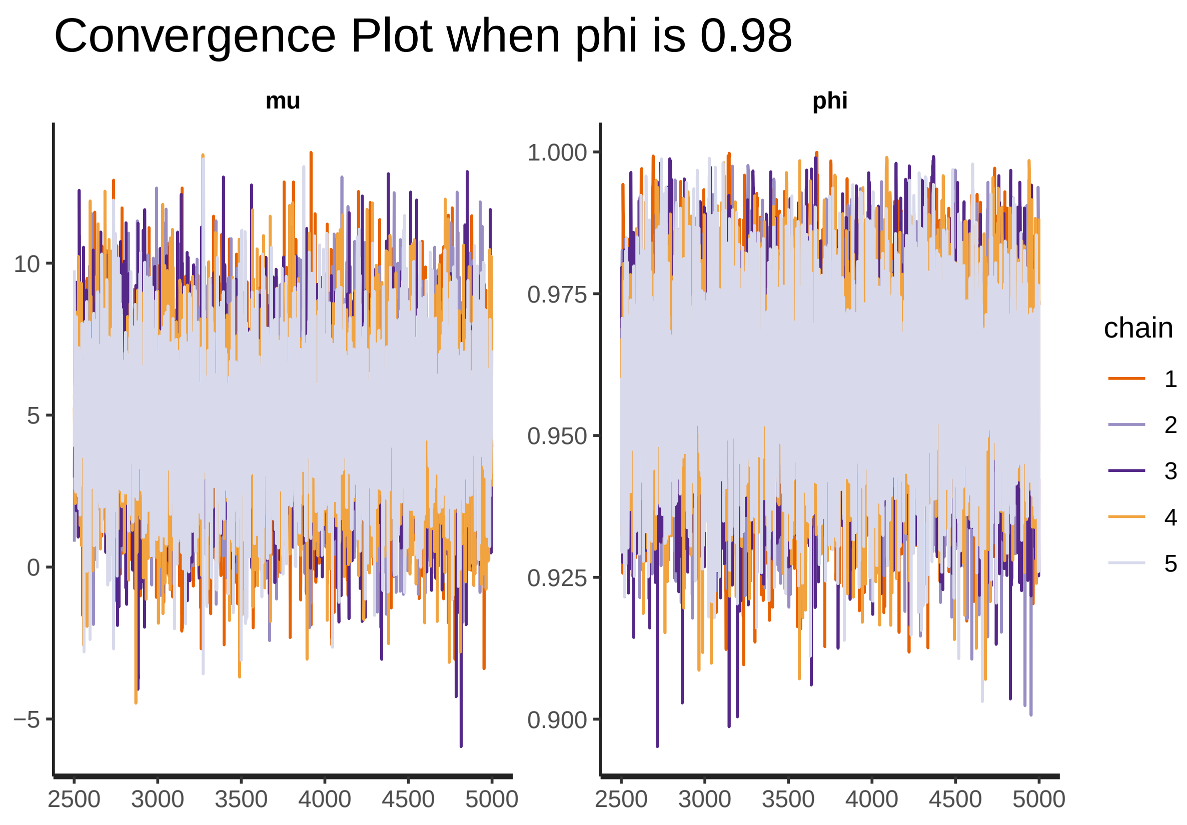
```
## $summary
##            mean      se_mean         sd       2.5%     97.5%    n_eff     Rhat
## mu    4.8109449 0.0256516234 2.46970780 0.06748737 9.7936832 9269.609 1.000326
## phi   0.9608244 0.0001952226 0.01617008 0.92880113 0.9908799 6860.629 0.999955
## sigma 2.9365173 0.0012614379 0.11992973 2.71027718 3.1825296 9039.032 1.000347
##
## $c_summary
## , , chains = chain:1
##
##          stats
## parameter      mean         sd       2.5%     97.5%
##     mu    4.7227598 2.51863874 -0.3165128 9.7845676
##     phi   0.9607914 0.01621682  0.9295531 0.9910221
##     sigma 2.9337475 0.12236479  2.7037871 3.1825296
##
## , , chains = chain:2
##
##          stats
## parameter     mean         sd      2.5%     97.5%
##     mu    4.842663 2.43700401 0.3331076 9.8639910
##     phi   0.960813 0.01567896 0.9296158 0.9890662
##     sigma 2.941064 0.12042606 2.7141548 3.1909348
##
## , , chains = chain:3
##
##          stats
## parameter      mean          sd      2.5%     97.5%
##     mu    4.8211801 2.52994434 0.1052078 9.7568865
##     phi   0.9613052 0.01716002 0.9274012 0.9937774
##     sigma 2.9383980 0.11933632 2.7137973 3.1811265
##
## , , chains = chain:4
##
##          stats
## parameter      mean          sd        2.5%     97.5%
##     mu    4.8387559 2.49444135 -0.08616949 9.9469548
##     phi   0.9608574 0.01581842  0.92902368 0.9901313
##     sigma 2.9353365 0.11859296  2.71054031 3.1798156
##
## , , chains = chain:5
##
##          stats
## parameter      mean          sd      2.5%    97.5%
##     mu    4.8293662 2.36466833 0.3338275 9.559214
##     phi   0.9603549 0.01593166 0.9291364 0.990525
##     sigma 2.9340408 0.11882032 2.7096044 3.174155
```

Results shows that the posterior summaries from stan were able to successfully estimate the true values of ar1 stimulation with some minor changes of aroung 0.25 difference for each paramter value.

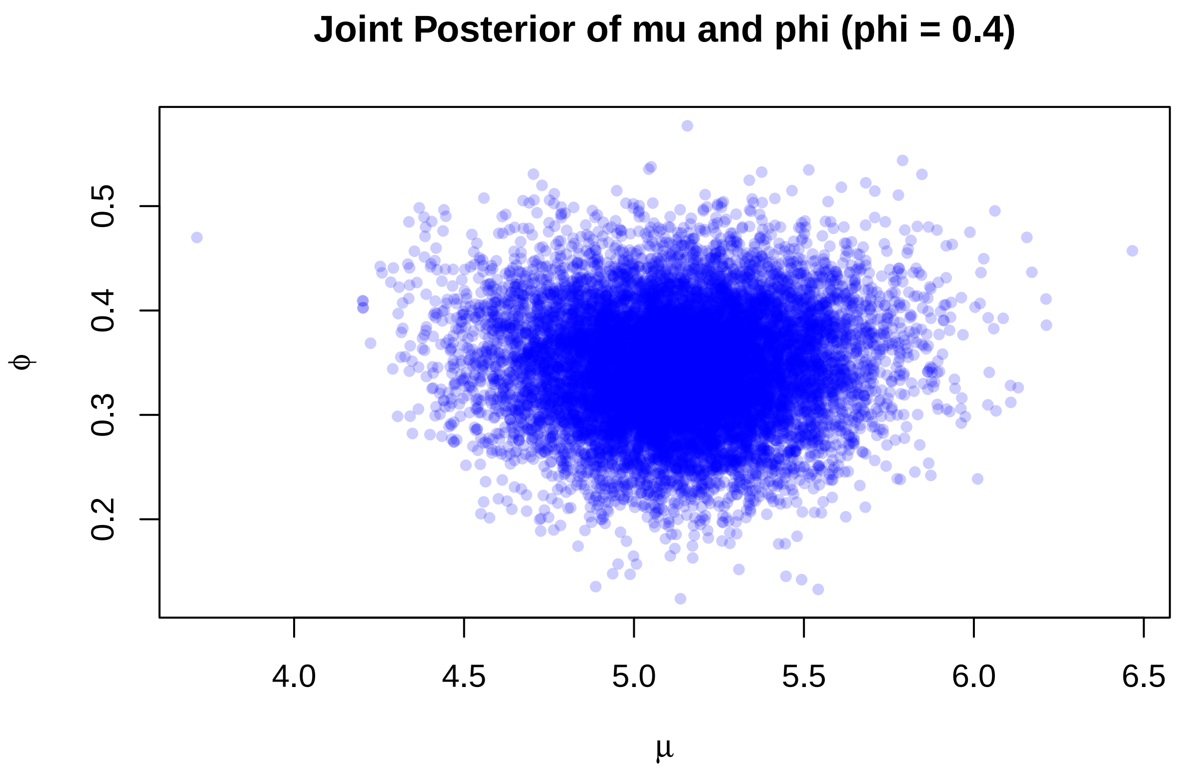
```r
#(ii)

#Convergence of the plots

traceplot(fit_phi4, pars = c("mu", "phi")) +
  ggtitle("Convergence Plot when phi is 0.4")
```



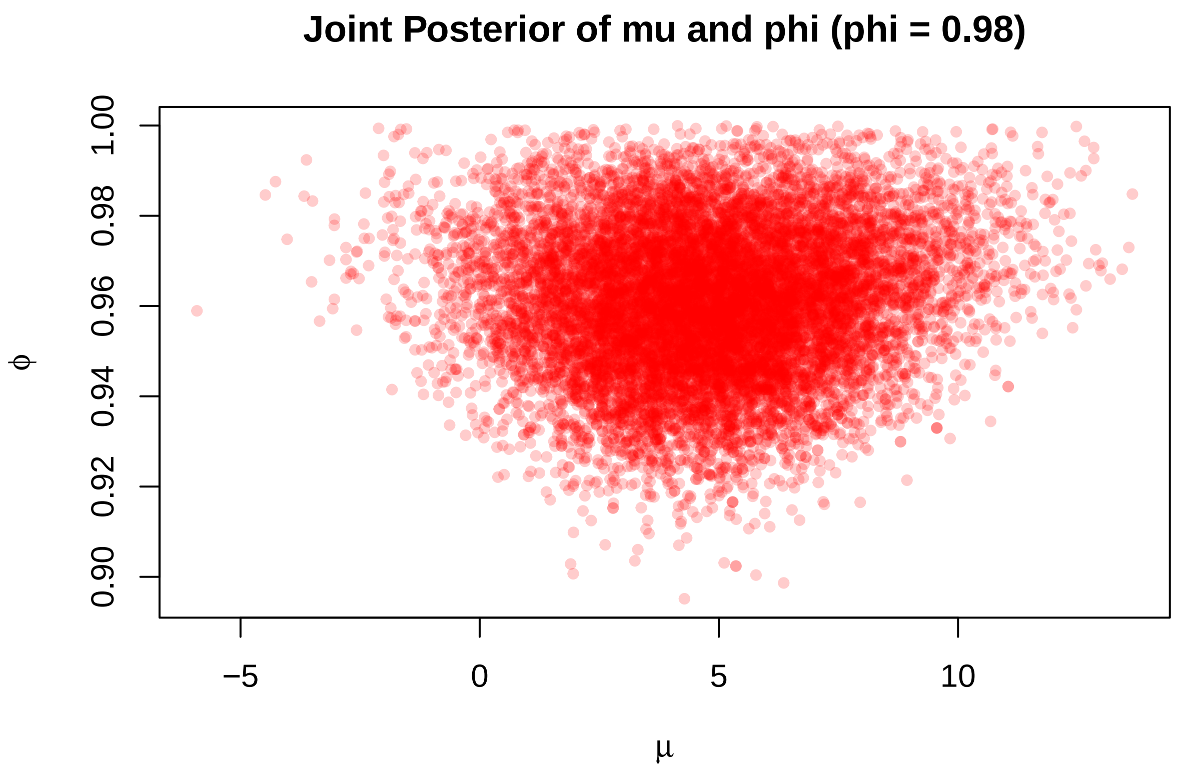
```r
traceplot(fit_phi98, pars = c("mu", "phi"))+
  ggtitle("Convergence Plot when phi is 0.98")
```



```r
posterior_phi4 <- as.matrix(fit_phi4)

plot(
  posterior_phi4[, "mu"], posterior_phi4[, "phi"],
  main = "Joint Posterior of mu and phi (phi = 0.4)",
  xlab = expression(mu),
  ylab = expression(phi),
  pch = 20, col = rgb(0, 0, 1, 0.2)
)
```



```r
posterior_phi98 <- as.matrix(fit_phi98)

plot(
  posterior_phi98[, "mu"], posterior_phi98[, "phi"],
  main = "Joint Posterior of mu and phi (phi = 0.98)",
  xlab = expression(mu),
  ylab = expression(phi),
  pch = 20, col = rgb(1, 0, 0, 0.2)
)
```


Joint Posterior of mu and phi (phi = 0.98)
φ
μ
0.90
0.92
0.94
0.96
0.98
1.00
−5
0
5
10