Lab3

Xuan Wang & Lepeng Zhang

2024-05-08

Question 1 Gibbs sampling for the logistic regression

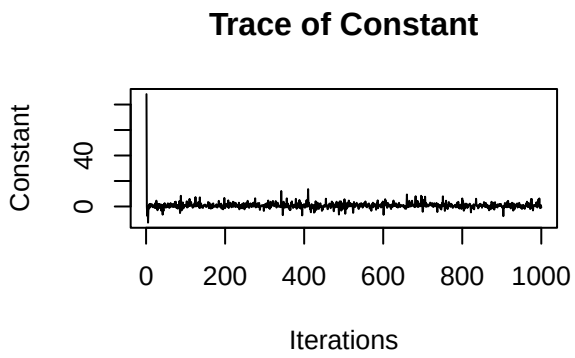
1a)

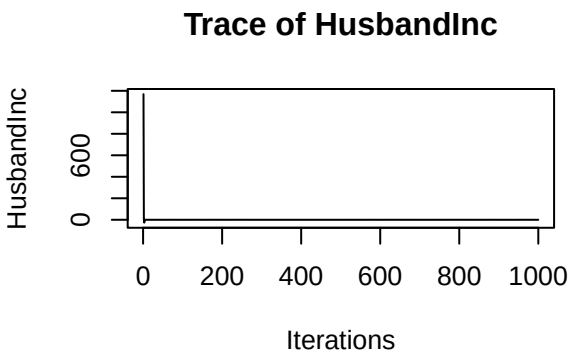
```r
# Load necessary libraries
library(MASS)
library(BayesLogit)
library(coda)
library(mvtnorm)
set.seed(12345)


# Load the data
data <- read.table("WomenAtWork.dat", header = TRUE)

# Define the logistic function
logistic <- function(x) {
  return(exp(x) / (1 + exp(x)))
}

# Define Response and Features
y <- data$Work
X <- as.matrix(data[,-1])
n <- nrow(X)
p <- ncol(X)


# Initialize parameters
tau <- 3
B <- diag(rep(tau^2, p))
b <- rep(0, p)
w <- rep(1, n)
k <- y - 1/2

# Gibbs sampler
n_iter <- 1000
chain_b <- matrix(0, nrow = n_iter, ncol = p)
for (iter in 1:n_iter) {
  for (i in 1:n) {
    w[i] <- rpg(1, abs(X[i,] %*% b) + 1e-10)
  }
  Vw <- solve(t(X) %*% diag(w) %*% X + solve(B))
```

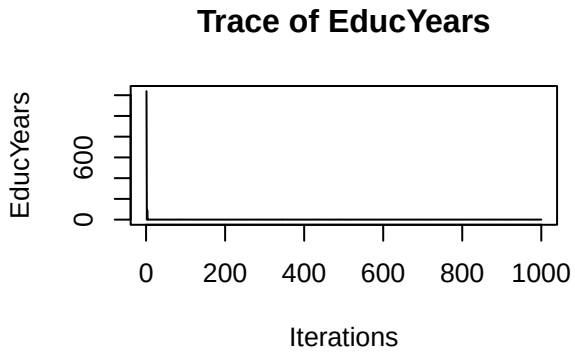
```
  mw <- Vw %*% (t(X) %*% k + solve(B) %*% b)
  b <- rnorm(p, mean = mw, sd = sqrt(diag(Vw)))
  chain_b[iter,] <- b
}

# Evaluate convergence
chain_b_mcmc <- mcmc(chain_b)
summary(chain_b_mcmc)
```

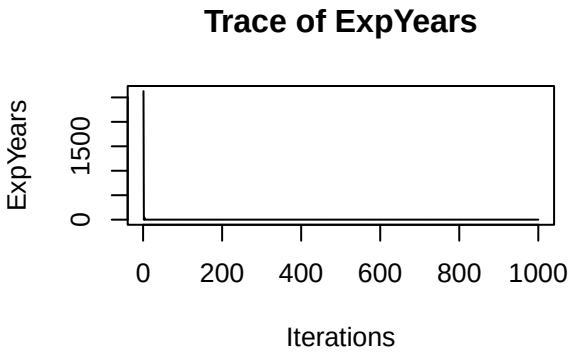
```
##
## Iterations = 1:1000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##          Mean      SD Naive SE Time-series SE
## [1,]  0.80233   3.436  0.10867        0.09089
## [2,]  1.11627  36.976  1.16929        1.16929
## [3,]  1.44646  39.285  1.24230        1.33741
## [4,]  2.77210  83.111  2.62819        2.62819
## [5,]  4.15996 133.895  4.23412        4.23412
## [6,] -1.52851   2.980  0.09423        0.13529
## [7,]  0.02921   2.142  0.06775        0.05131
##
## 2. Quantiles for each variable:
##
##             2.5%      25%      50%       75%     97.5%
## var1 -3.214e+00 -0.24121  0.59453  1.570860  5.699117
## var2 -1.229e-01 -0.03321 -0.01641 -0.004928  0.018884
## var3 -6.240e-02  0.02413  0.08062  0.162113  0.558556
## var4  7.226e-05  0.03073  0.05979  0.112041  0.396509
## var5 -1.907e-01 -0.06615 -0.03426 -0.016242  0.009439
## var6 -4.990e+00 -1.77893 -1.03997 -0.569574 -0.087503
## var7 -4.270e-01 -0.10792 -0.01462  0.080500  0.338294
```

```
var_names <- colnames(X)
par(mfrow = c(2, 2))
for (i in 1:p) {
  plot(chain_b[, i], main = paste("Trace of", var_names[i]), type = "l", xlab='Iterations', ylab= paste
}
```

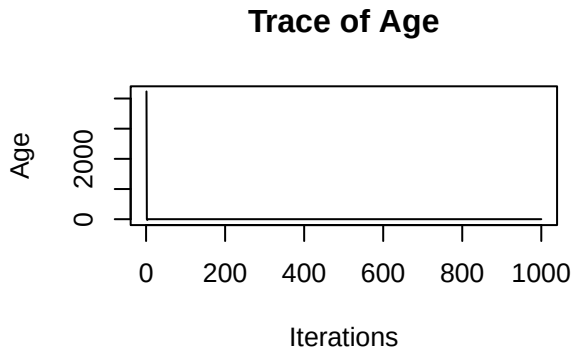


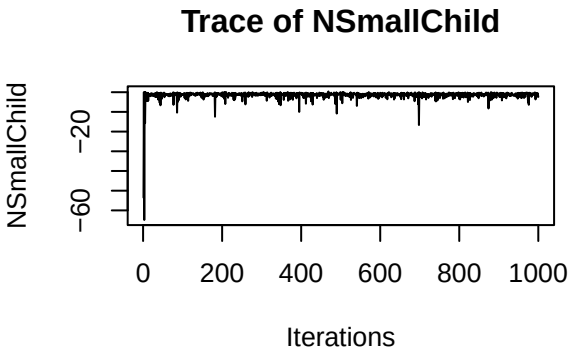


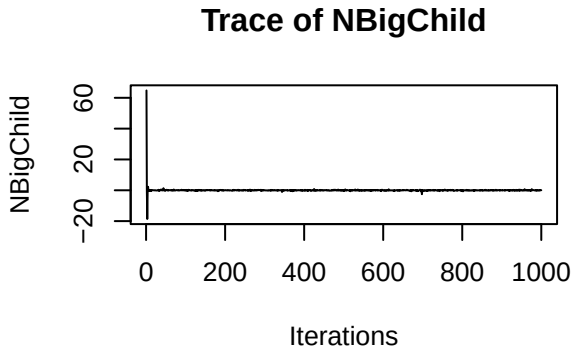





```
par(mfrow = c(1, 1))  # Reset the plot layout
```








```
# Calculate Inefficiency Factors (IFs)
IFs <- apply(chain_b, 2, function(x) {
```

```
  acf_x <- acf(x, plot = FALSE)$acf[,,1]
  1 + 2 * sum(acf_x[2:length(acf_x)])
})

cat("The Ineffciency Factors (IFs) is: \n")
```

```
## The Ineffciency Factors (IFs) is:
```

```
cat(IFs)
```

```
## 0.6583263 0.9520363 1.145592 1.037095 0.9873157 1.982357 0.5107157
```

```
# Compute 90% credible interval
x_new <- c(1, 22, 12, 7, 38, 1, 0)
pr_y <- logistic(x_new %*% t(chain_b))
CI_90 <- quantile(pr_y, probs = c(0.05, 0.95), na.rm = TRUE)
cat("90% equal tail credible interval for Pr(y = 1|x) is: ", CI_90)
```

```
## 90% equal tail credible interval for Pr(y = 1|x) is:  0.008488858 0.9016289
```

Question 2 Metropolis Random Walk for Poisson regression

2a)

```
# Load necessary libraries
library(MASS)
set.seed(12345)
# Load the data
data <- read.table("eBayNumberOfBidderData_2024.dat", header = TRUE)

# Obtain the maximum likelihood estimator of  in the Poisson regression model
model <- glm(nBids ~ PowerSeller + VerifyID + Sealed + Minblem + MajBlem + LargNeg + LogBook + MinBidSha
             family = poisson(link = "log"), data = data)
summary(model)
```

```
##
## Call:
## glm(formula = nBids ~ PowerSeller + VerifyID + Sealed + Minblem +
##     MajBlem + LargNeg + LogBook + MinBidShare, family = poisson(link = "log"),
##     data = data)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.07981    0.03393  31.828  < 2e-16 ***
## PowerSeller -0.03566    0.04167  -0.856 0.392109
## VerifyID    -0.45564    0.12748  -3.574 0.000351 ***
## Sealed       0.45515    0.06226   7.311 2.65e-13 ***
## Minblem     -0.06837    0.07198  -0.950 0.342228
## MajBlem     -0.22554    0.09525  -2.368 0.017894 *
## LargNeg      0.05382    0.06406   0.840 0.400787
## LogBook     -0.08499    0.03234  -2.628 0.008599 **
## MinBidShare -1.82490    0.07843 -23.269  < 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: 1699.6  on 799  degrees of freedom
## Residual deviance:  691.8  on 791  degrees of freedom
## AIC: 2879.1
##
## Number of Fisher Scoring iterations: 5
```

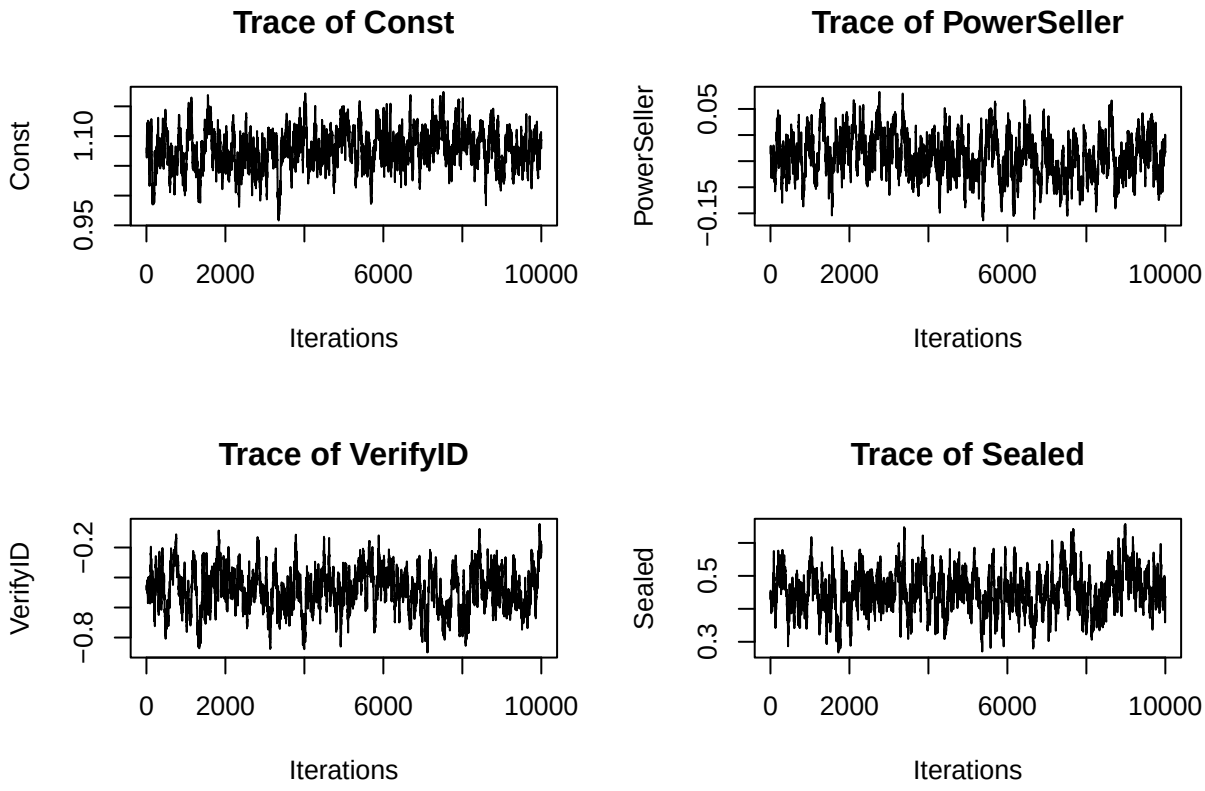
According to above results, the significant covariates are: Intercept, VerifyID, Sealed, and MinBidShare.

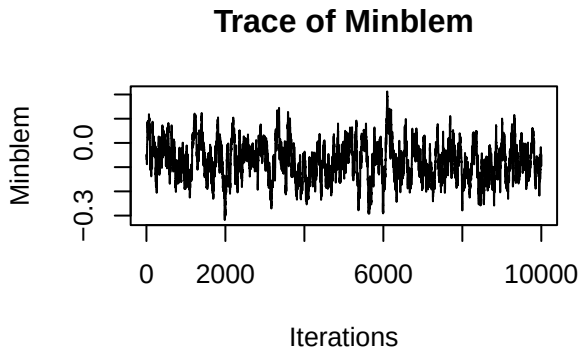
```
# Bayesian analysis of the Poisson regression
prior <- function(beta, X, y) {
  n <- dim(X)[1]
  p <- dim(X)[2]
  beta_prior <- rep(0, p)
  Sigma_prior <- 100 * solve(t(X) %*% X)
  dmvnorm(beta, mean = beta_prior, sigma = Sigma_prior)
}

posterior <- function(beta, X, y) {
  likelihood <- sum(dpois(y, lambda = exp(X %*% beta), log = TRUE))
  prior <- prior(beta, X, y)
  return(likelihood + prior)
}

optim_result <- optim(par = coef(model), fn = posterior, X = model.matrix(model), y = data$nBids,
                      control = list(fnscale = -1), hessian = TRUE)
beta_hat <- optim_result$par
Sigma_hat <- solve(-optim_result$hessian)

print(beta_hat)
```

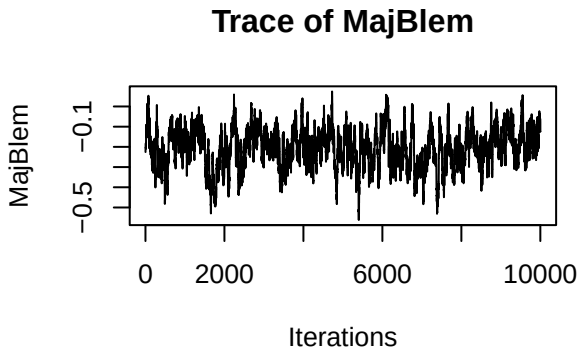
```
## (Intercept) PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##  1.07980512 -0.03566493 -0.45563760  0.45515199 -0.06836819 -0.22554138
##     LargNeg     LogBook MinBidShare
##  0.05382386 -0.08498844 -1.82490142
```

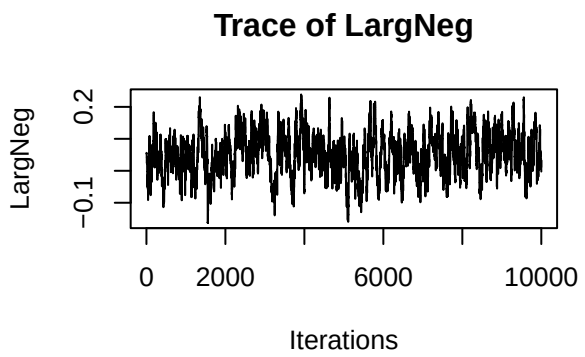
```
# Metropolis algorithm
metropolis <- function(posterior, init, iter, Sigma, c, X, y) {
  chain <- matrix(NA, nrow = iter, ncol = length(init))
  chain[1, ] <- init
  for (i in 2:iter) {
    proposal <- mvrnorm(n = 1, mu = chain[(i - 1), ], Sigma = c * Sigma)
    log_prob <- exp(posterior(proposal, X, y) - posterior(chain[(i - 1), ], X, y))
    accept_prob <- min(1,log_prob)
    if (runif(1) <= accept_prob) {
      chain[i, ] <- proposal
    } else {
      chain[i, ] <- chain[(i - 1), ]
    }
  }
  return(chain)
}

chain <- metropolis(posterior, init = beta_hat, iter = 10000, Sigma = Sigma_hat, c = 0.1,
                    X = model.matrix(model), y = data$nBids)
```

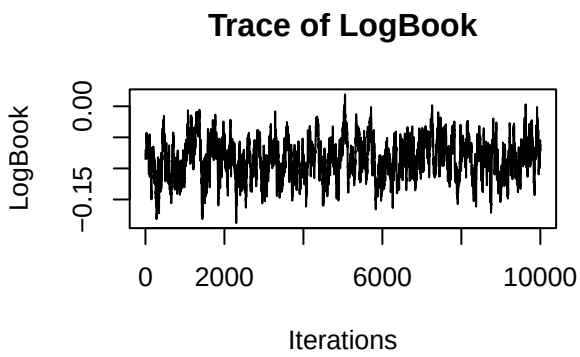
```r
names <- colnames(data[,-1])
par(mfrow = c(2, 2))
for (i in 1:9) {
  plot(chain[, i], main = paste("Trace of", names[i]), type = "l", xlab='Iterations', ylab= paste(names
}
```



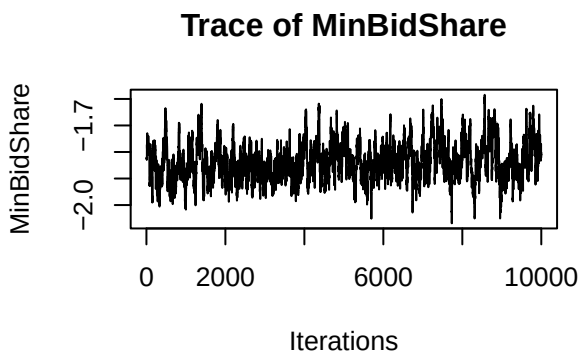




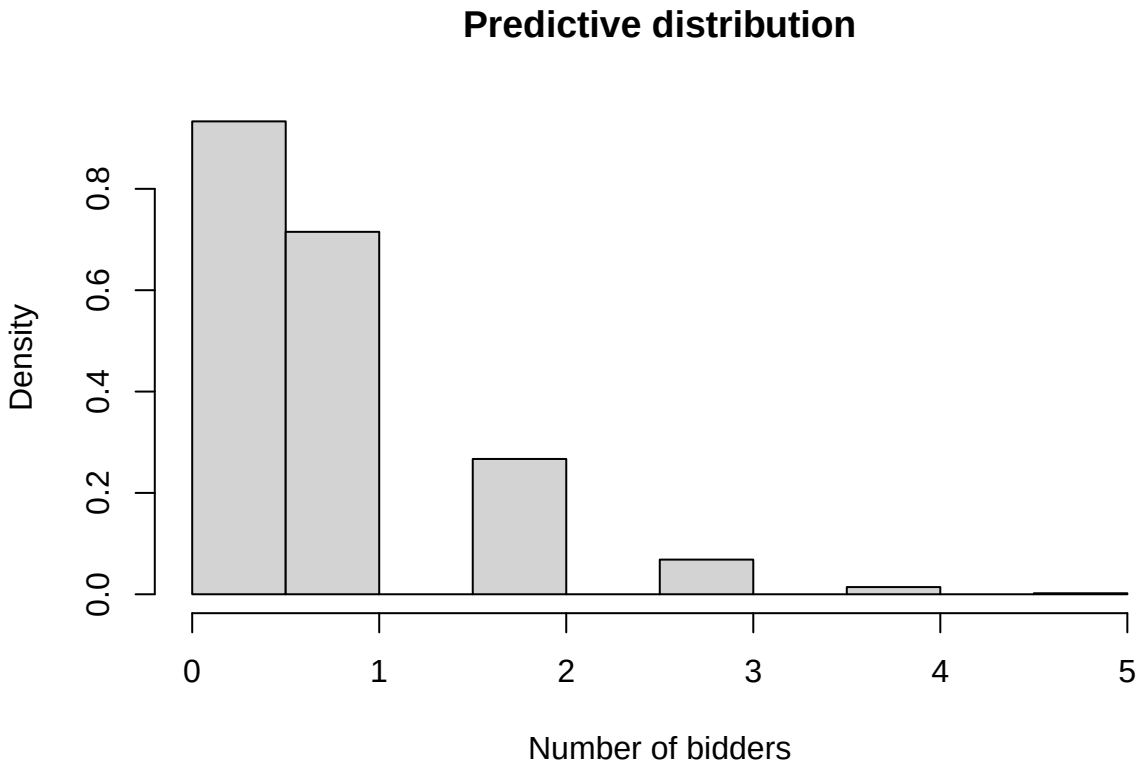





```r
par(mfrow = c(1, 1))  # Reset the plot layout
```




```r
# Predictive distribution
new_auction <- c(1, 1, 0, 1, 0, 1, 0, 1.2, 0.8)
pred_draws <- rpois(10000, lambda = exp(new_auction %*% t(chain)))
hist(pred_draws, freq = FALSE, main = "Predictive distribution", xlab = "Number of bidders")
```




```
prob_no_bidders <- mean(pred_draws == 0)
cat("The probability of no bidders in this new auction:" , prob_no_bidders)
```

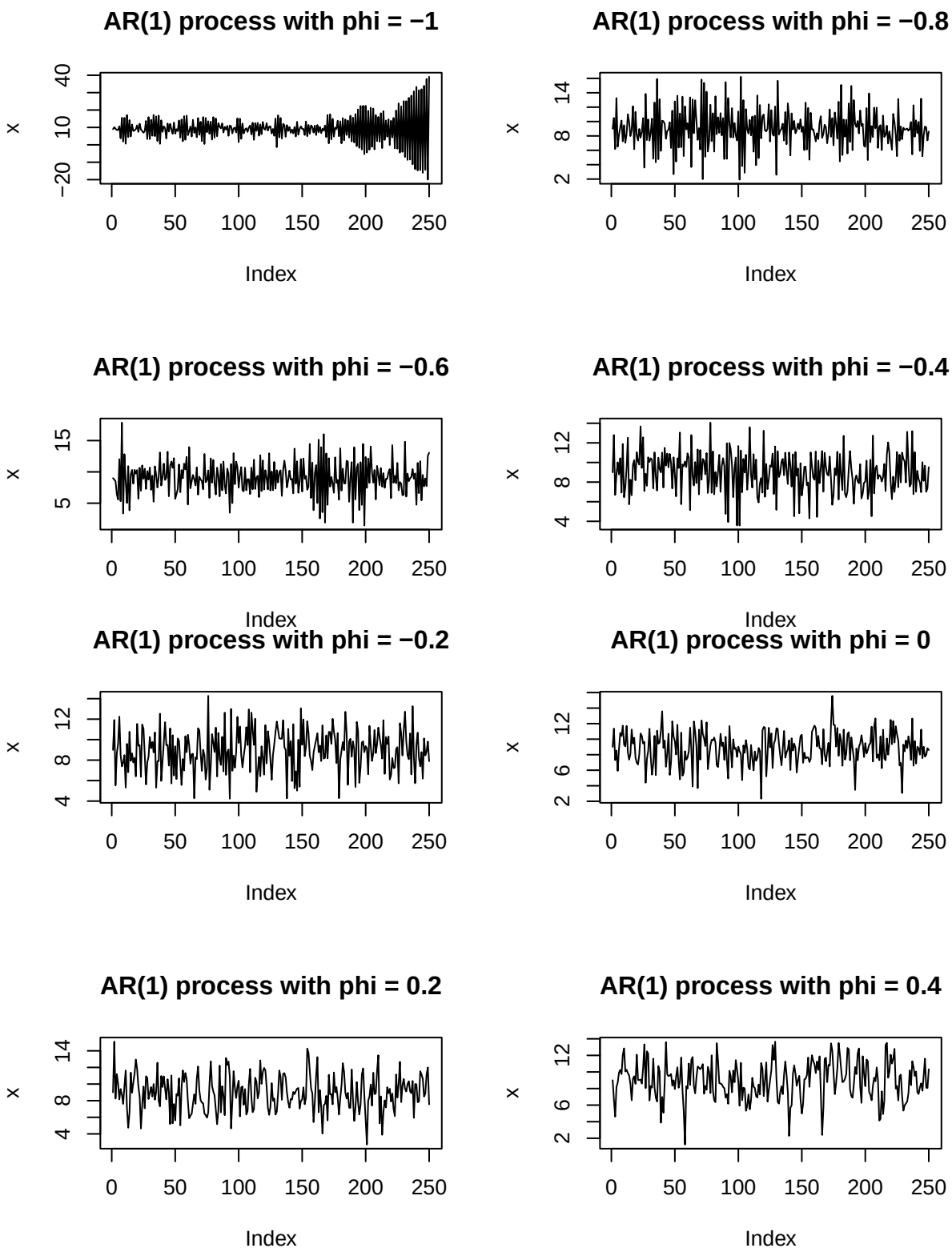
```
## The probability of no bidders in this new auction: 0.4666
```

Question 3 Time series models in Stan

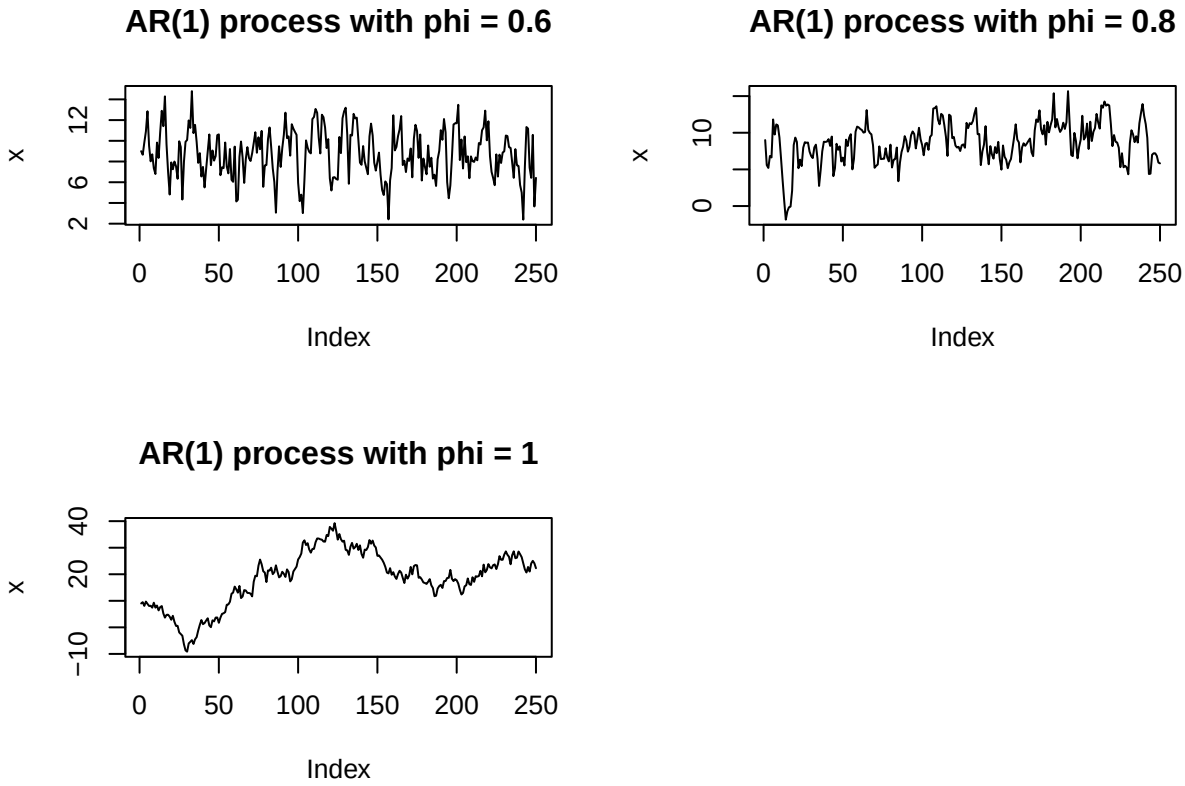
3a)

```
# Load required library
library(rstan)
set.seed(12345)
# Function to simulate AR(1) process
simulate_ar1 <- function(mu, phi, sigma_sq, T) {
  x <- numeric(T)
  x[1] <- mu
  for (t in 2:T) {
    e_t <- rnorm(1, mean = 0, sd = sqrt(sigma_sq))
    x[t] <- mu + phi * (x[t-1] - mu) + e_t
  }
  return(x)
}

# Simulate AR(1) process
mu <- 9
sigma_sq <- 4
T <- 250
phi_values <- seq(-1, 1, by = 0.2)
par(mfrow = c(2, 2))
```

```r
for (phi in phi_values) {
  x <- simulate_ar1(mu, phi, sigma_sq, T)
  plot(x, main = paste("AR(1) process with phi =", phi), type = "l")
}
```



```r
par(mfrow = c(1, 1))  # Reset the plot layout
```



ϕ close to -1: The plot shows a strong negative autocorrelation. That is, if the process was high (or low) at one point, it is likely to be low (or high) at the next point. This aligns with the first plot that there is a wider up and down range as the process frequently changes between high and low values.

ϕ close to 0: The plot shows there are not too much autocorrelation. The values in the process are not influenced by their preceding values. As a result, it can be seen that the fluctuation becomes smooth and within a regular range, meaning the process appears more random and less predictable.

ϕ close to 1: The plot shows a strong positive autocorrelation. If the process was high (or low) at one point, it is likely to be high (or low) at the next point. This can lead to irregular up and down fluctuations as the process tends to maintain its current state for longer periods, as the last plot shown.

```r
set.seed(12345)
# Stan code
stan_code <- "
data {
  int<lower=0> T;
  vector[T] y;
}
parameters {
  real mu;
  real<lower=-1, upper=1> phi;
  real<lower=0> sigma;
}
model {
  vector[T] nu;
  nu[1] <- mu;
  for (t in 2:T) {
```

```
    nu[t] <- mu + phi * (y[t-1] - mu);
  }
  y ~ normal(nu, sigma);
}
"

# Compile Stan model
stan_model <- stan_model(model_code = stan_code)

# Simulate two AR(1) processes
x1 <- simulate_ar1(mu, 0.3, sigma_sq, T)
y1 <- simulate_ar1(mu, 0.97, sigma_sq, T)

# Fit Stan model to the simulated data
fit_x1 <- sampling(stan_model, data = list(T = T, y = x1))
fit_y1 <- sampling(stan_model, data = list(T = T, y = y1))
```

```
# Print the results
print(summary(fit_x1)$summary)
```

```
##               mean     se_mean         sd        2.5%          25%          50%
## mu       9.1905497 0.002709162 0.15520180    8.895450    9.0862328    9.1918514
## phi      0.2424291 0.001016187 0.06148230    0.125924    0.2006003    0.2421251
## sigma    1.8490849 0.001317939 0.08381263    1.689257    1.7903087    1.8469382
## lp__  -278.1752514 0.026466853 1.21964442 -281.273451 -278.7220575 -277.8755240
##                75%        97.5%    n_eff      Rhat
## mu       9.2957065    9.4892556 3281.886 1.0011961
## phi      0.2838876    0.3694193 3660.603 0.9995990
## sigma    1.9031192    2.0221761 4044.160 0.9993457
## lp__  -277.2639605 -276.7786754 2123.547 1.0018915
```

```
print(summary(fit_y1)$summary)
```

```
##               mean     se_mean         sd         2.5%          25%
## mu       8.0753019 0.036446688 1.82916093    4.5986100    6.8420866
## phi      0.9695688 0.000362634 0.01573012    0.9355659    0.9595571
## sigma    1.9141773 0.001706622 0.08598444    1.7495130    1.8578694
## lp__  -289.4189068 0.043535591 1.39998956 -293.1693479 -290.0527891
##                50%          75%        97.5%    n_eff      Rhat
## mu       7.9923555    9.2803634   11.9913829 2518.765 1.0011927
## phi      0.9706627    0.9811544    0.9964299 1881.598 0.9999003
## sigma    1.9097000    1.9676696    2.0933737 2538.430 0.9997161
## lp__  -289.0323030 -288.3924958 -287.8628621 1034.096 1.0020259
```

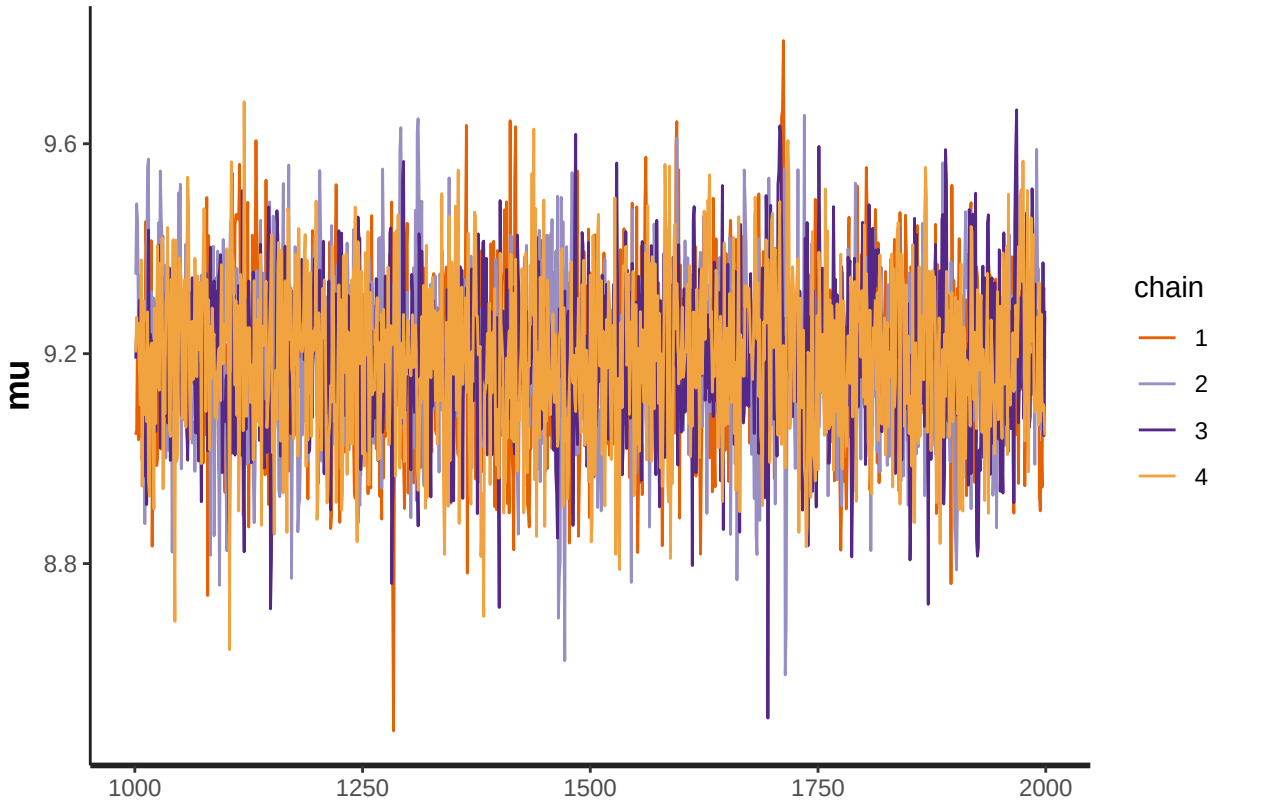
From above results, for the first dataset from ϕ=0.3,

parameters	mean	95% credible intervals	the number of effective samples
mu	9.191	8.895-9.489	3281
phi	0.242	0.201-0.369	3660
sigma	1.849	1.689-2.022	4044

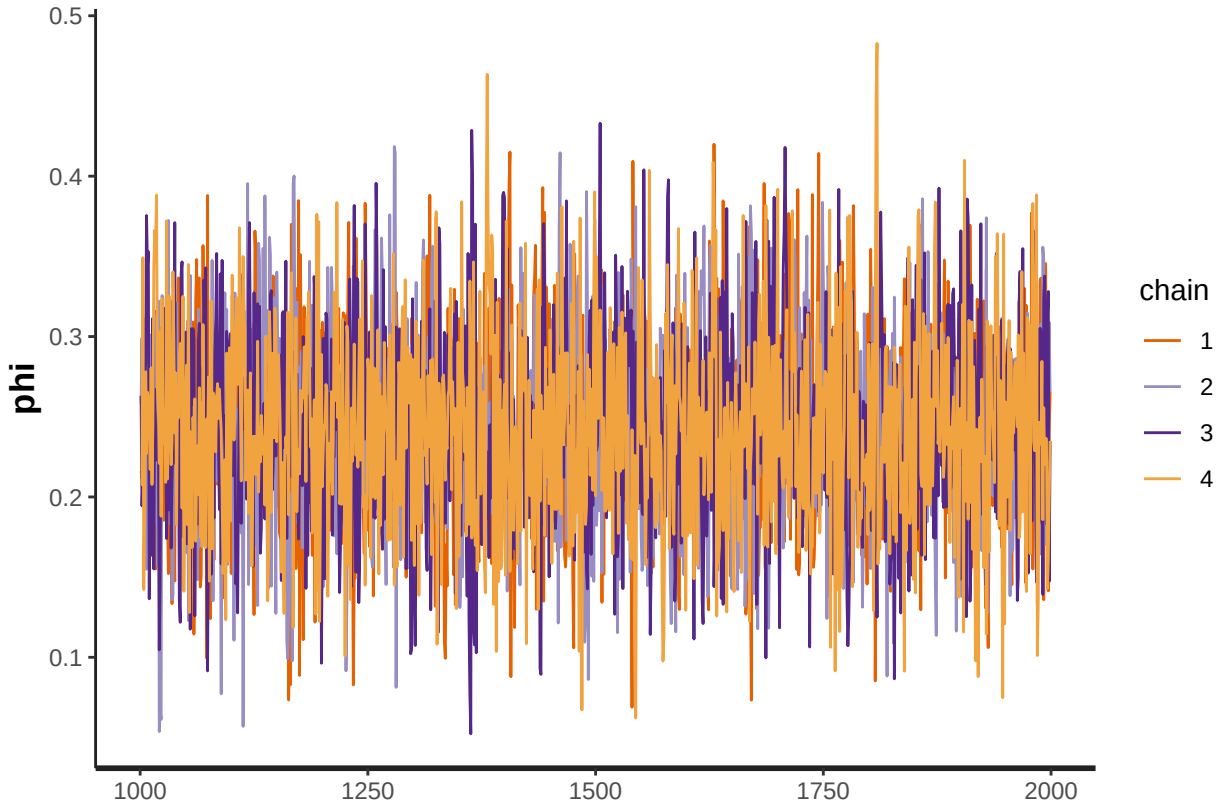
for the second dataset from ϕ=0.97,

parameters	mean	95% credible intervals	the number of effective samples
mu	8.075	4.598-11.991	2518
phi	0.969	0.935-0.996	1881
sigma	1.914	1.749-2.093	2538

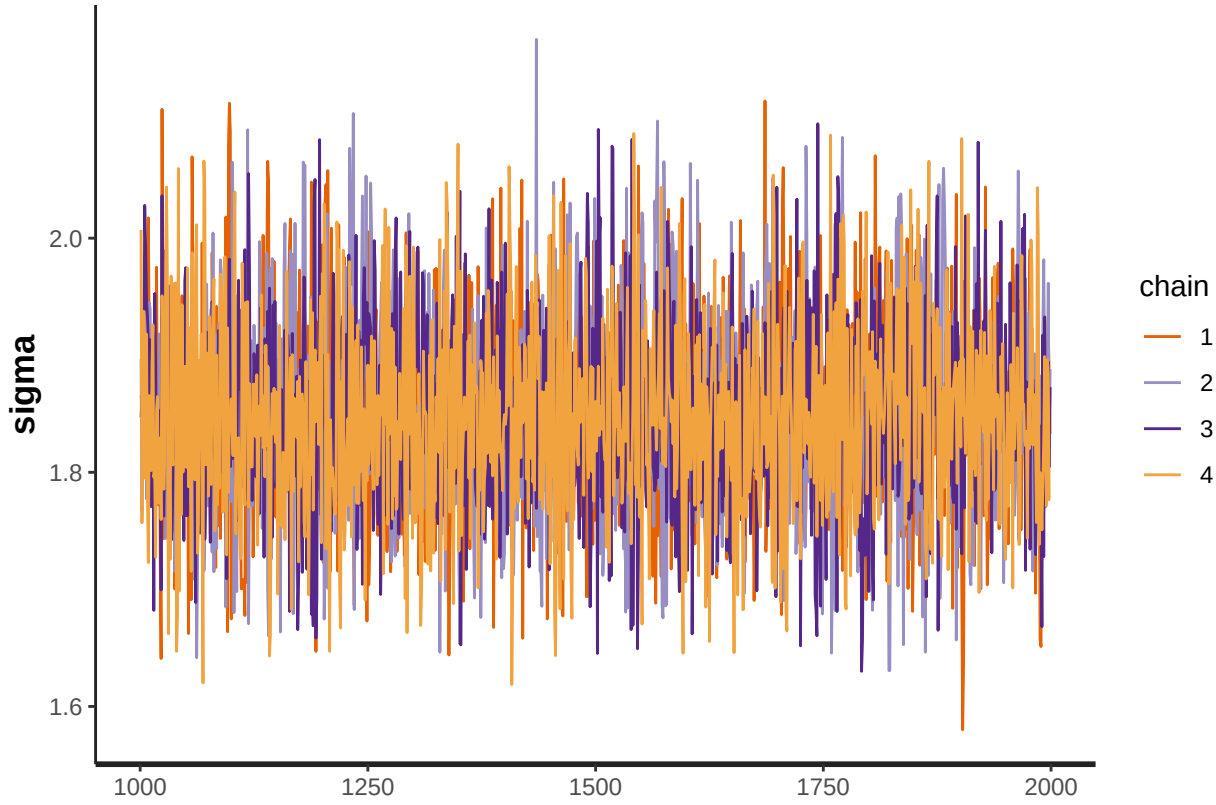
```
stan_trace(fit_x1, pars=c("mu"))
```




```
stan_trace(fit_x1, pars=c("phi"))
```



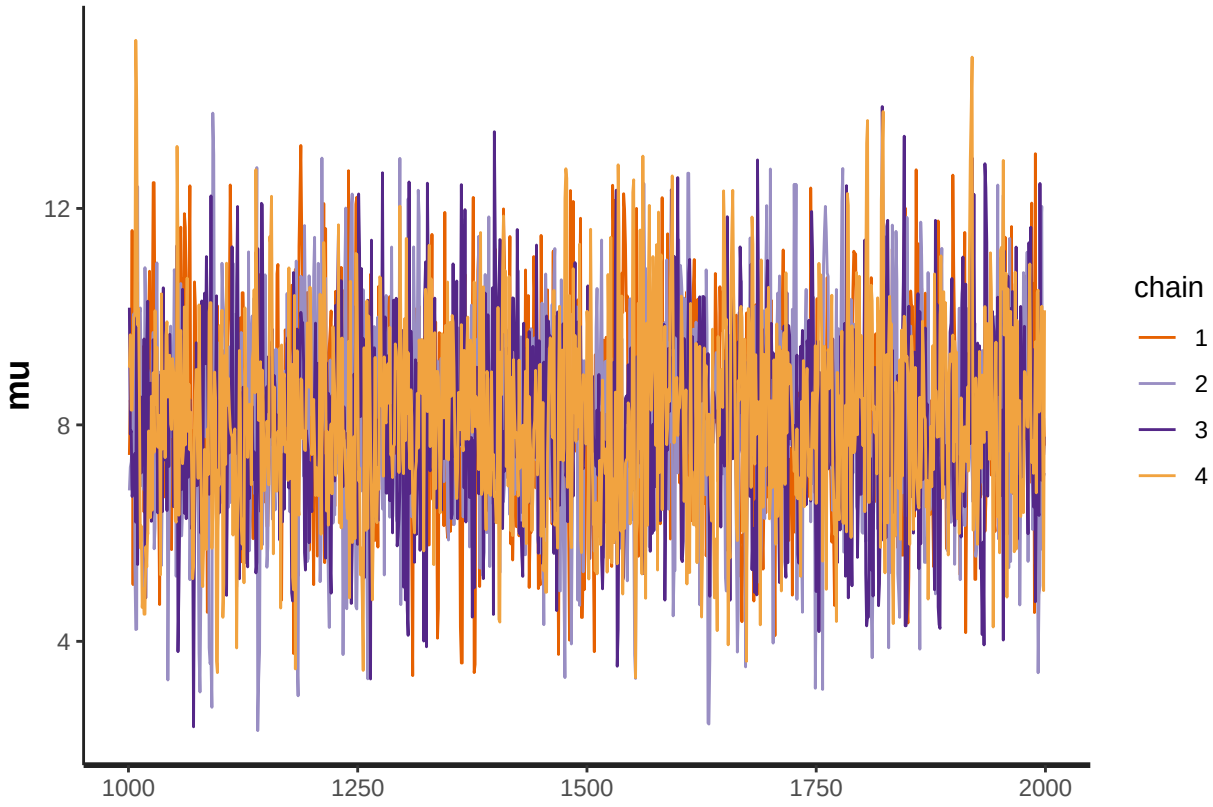

```
stan_trace(fit_x1, pars=c("sigma"))
```



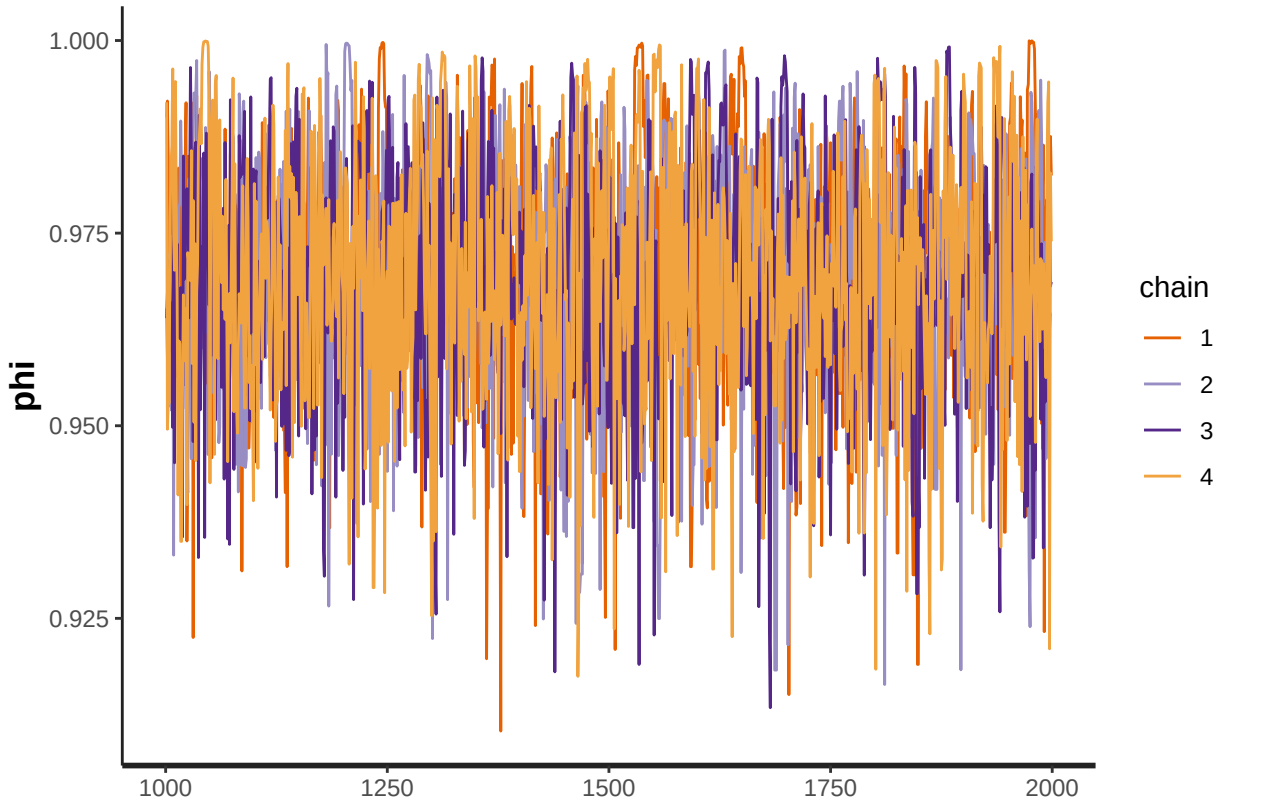
Above three trace plots are for the first dataset from ϕ=0.3, it can be seen that all three parameters converge

quite well in the stationery distribution, the plots are also symmetric around the mean value.

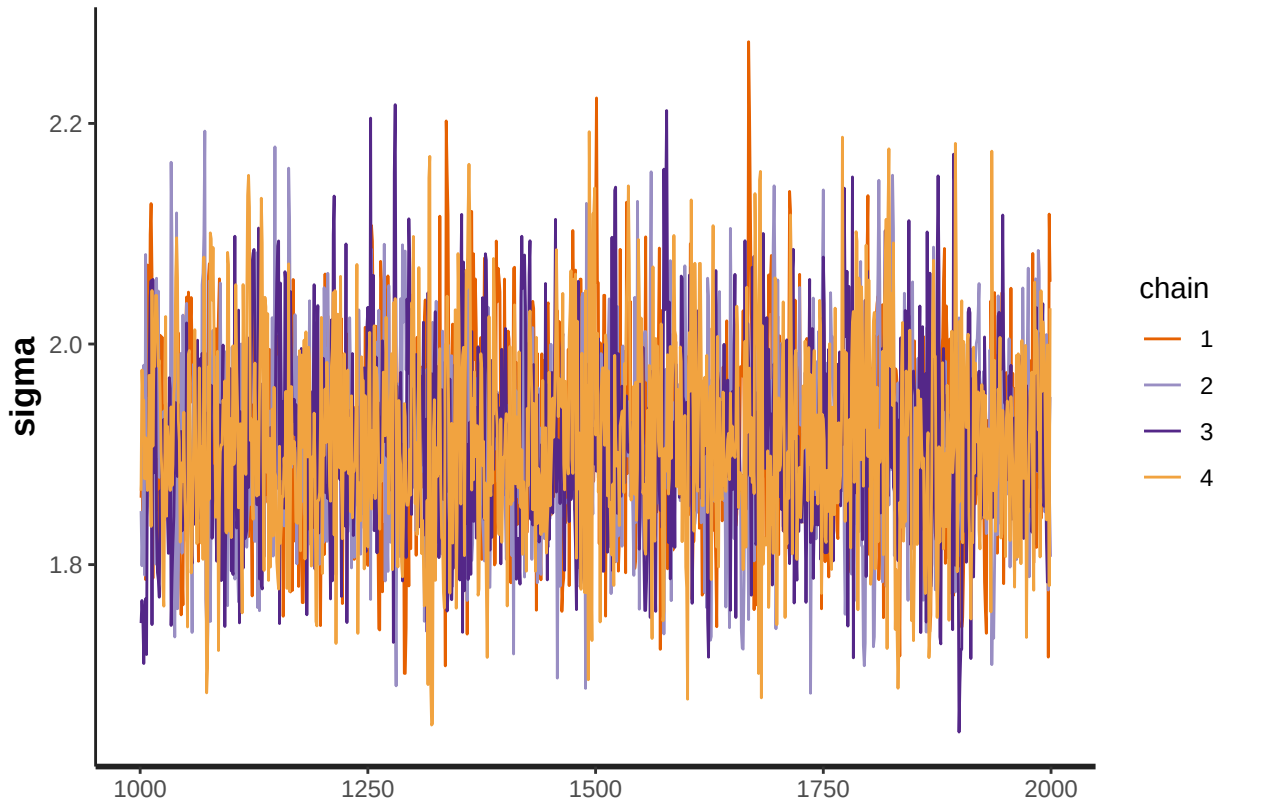
```
stan_trace(fit_y1, pars=c("mu"))
```




```
stan_trace(fit_y1, pars=c("phi"))
```



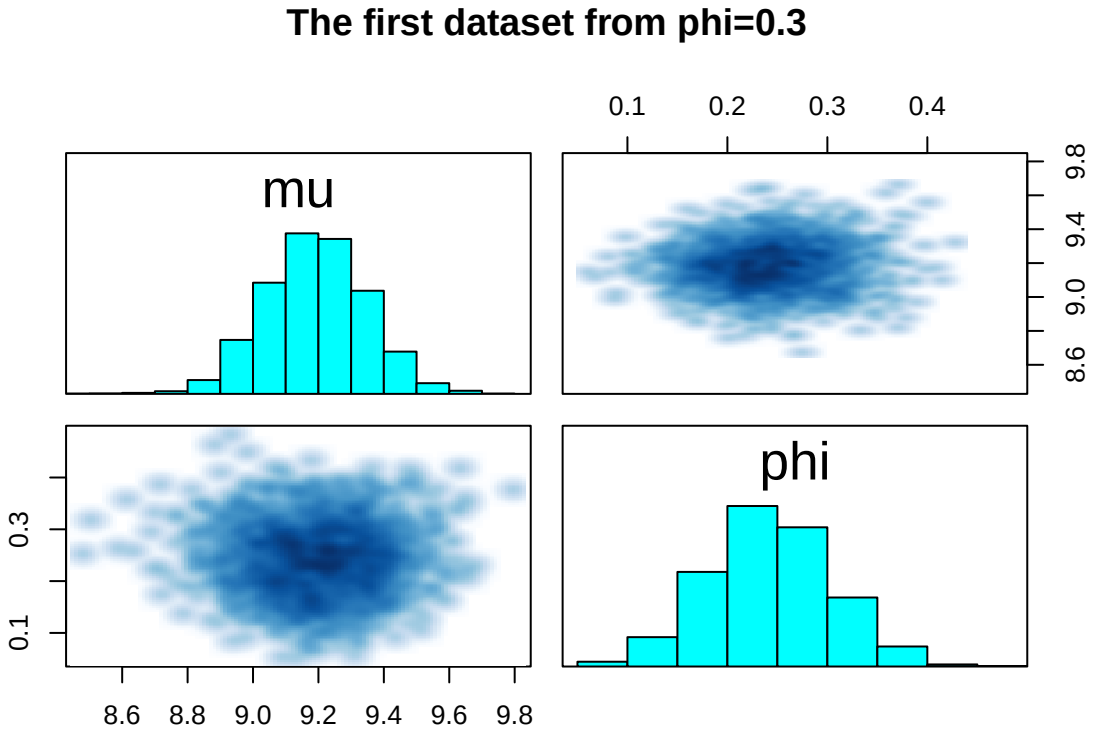

```
stan_trace(fit_y1, pars=c("sigma"))
```



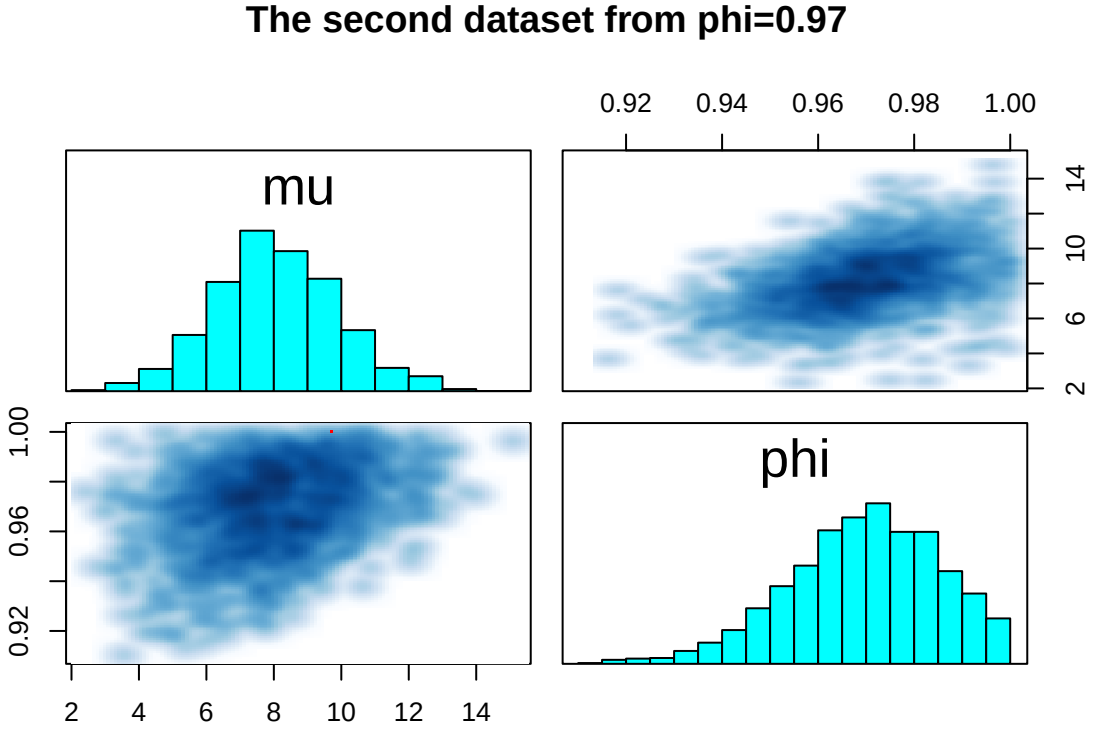
Above three trace plots are for the second dataset from ϕ=0.97, it can be seen that the parameter mu

doesn't converge well, the variance of the points is bigger. This is because high ϕ leads to a strong positive autocorrelation.

```
pairs(fit_x1, pars = c("mu", "phi"), main = "The first dataset from phi=0.3")
```




```
pairs(fit_y1, pars = c("mu", "phi"), main = "The second dataset from phi=0.97")
```



From above joint posterior plots of μ and ϕ, the first dataset generates symmetric ellipsoid shape, which could be possibly a multivariate normal distribution. While the second dataset generates a deformed ellipsoid,

which are hard to guess the distribution.