Computer Lab 3

Hugo Morvan

2024-05-27

1. Gibbs sampling for logistic regression

Consider again the logistic regression model in problem 2 from the previous computer lab 2. Use the prior $\beta \sim N(0, \tau^2 I)$, where $\tau = 3$.

a)

Implement (code!) a Gibbs sampler that simulates from the joint posterior $p(\omega, \beta|x)$ by augmenting the data with Polya-gamma latent variables ω_i , i = 1,. . . , n. The full conditional posteriors are given on the slides from Lecture 7. Evaluate the convergence of the Gibbs sampler by calculating the Inefficiency Factors (IFs) and by plotting the trajectories of the sampled Markov chains.

From the lecture: Gibbs sampling for the Logistic regression:

Given $\omega = (\omega_1, \omega_2, ..., \omega_n)$, the conditional posterior of β with prior $\beta \sim N(b, B)$ follows a multivariate normal distribution. ω is not observed –> Gibbs sampling to the rescue. Simulate from the joint posterior $p(\omega, \beta|y)$ by iterating between :

- $\omega_i|\beta \sim PG(1, x_i^T\beta)$, i = 1. . . n
- $$\beta|y, \omega \sim N(m_\omega, V_\omega)$$
  ,
  $$V_\omega = (X^T \Omega X + B^{-1})^{-1}$$
  ,
  $$m_\omega = V_\omega(X^T \kappa + B^{-1}b)$$
  ,

where $\kappa = (y_i - 0.5, ..., y_n - 0.5)$, Ω is the diagonal matrix of $\omega_i's$ and $PG(1, x_i^T\beta)$ is the poly-gamma sampler from the BayesLogit R package.

From Lecture 7 slide 23:

Logistic regression:

$$Pr(y_i = 1|x_i, \beta) = \frac{exp(x_i^T\beta)}{1 + exp(x_i^T\beta)}$$

The posterior distribution is not known. Augment the data with Polya-gamma latent variables ω_i , i=1,. . . ,n.

$$\omega_i = \frac{1}{2 * \pi^2} * \sum_{k=1}^{\infty} \frac{g_k}{(k - 0.5)^2 + \frac{(x_i^T\beta)^2}{4\pi^2}}$$

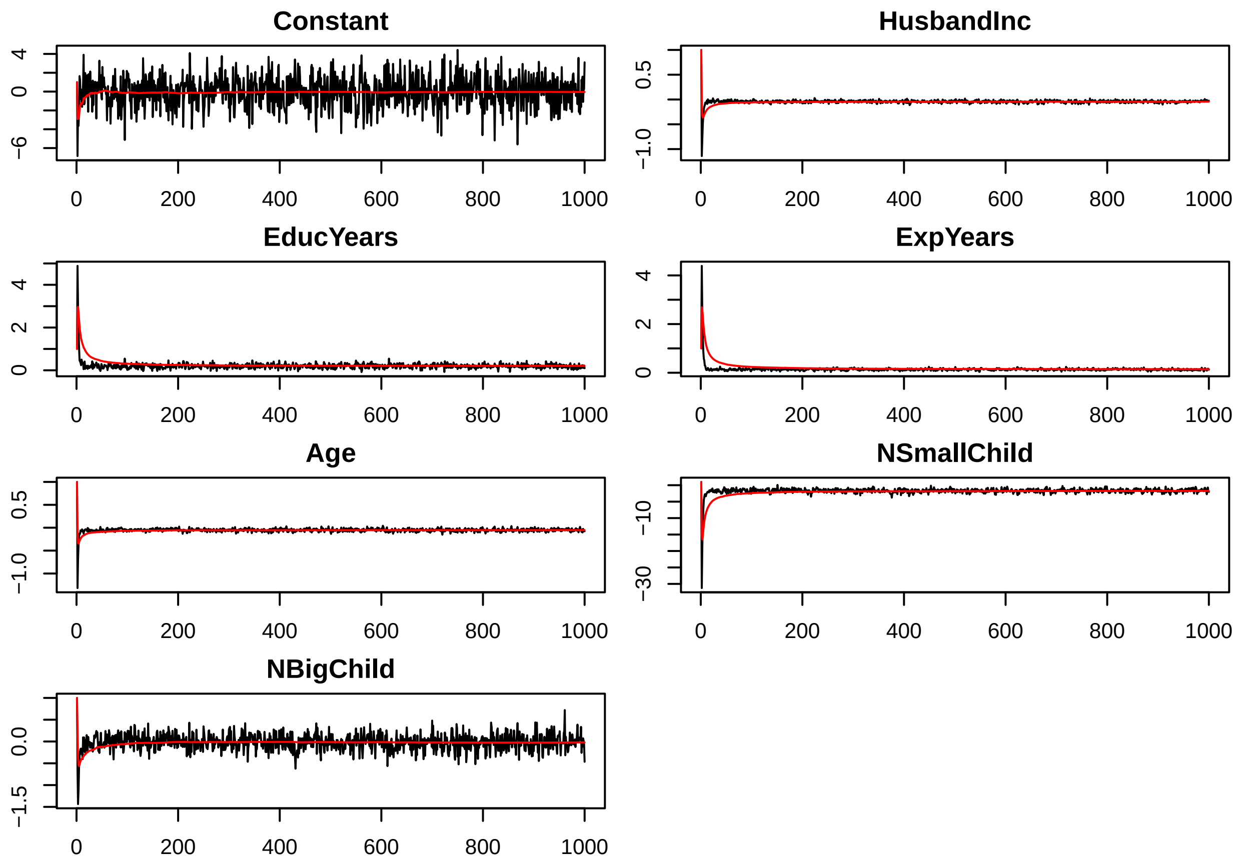
, where g_k are independent draws from the exponential distribution with mean 1.

```r
library(BayesLogit)
library(mvtnorm)

WomenData <- read.table("WomenAtWork.dat", header = T) # read data from .dat file
Nobs <- dim(WomenData)[1] # number of observations
y <- WomenData$Work

Covs <- c(2:8) # Selects which covariates/features to include
X <- as.matrix(WomenData[,Covs]);
Xnames <- colnames(X)
nPar <- dim(X)[2] #7

# Setting up the prior

nDraws = 1000

tau <- 3
b = as.matrix(rep(0,nPar)) # Prior mean vector, needed to calculate m_omega
B = tau^2*diag(nPar) #Needed to calculate V_omega and m_omega, prior covariance matrix
kappa <- as.matrix(y-0.5) #Needed to calculate m_omega

omega_i_given_beta = matrix(0, nrow = nrow(X), ncol=nDraws)

betas = matrix(nrow=nDraws, ncol=nPar)
betas[1,] = rep(1,nPar) # Initial betas values, should it be from N(b,B) ?

for(draw in 2:nDraws){ #Start at 2 because we have set 1 above

  #First, augment the data with Poly-gamma latent variables
  for(row in 1:nrow(X)){
    omega_i_given_beta[row, draw-1] = rpg(1, h=1, X[row,]%*%betas[draw-1,])
  }
  #Omega is the diagonal matrix of the omega_i's
  Omega = diag(omega_i_given_beta[,draw-1])
  # Calculate V_omega and m_omega,
  V_omega = solve(t(X) %*% Omega %*% X + solve(B))
  m_omega = V_omega%*%(t(X) %*% kappa + solve(B) %*% b)

  #Finally, calculate beta given y and omega
  betas[draw,] = rmvnorm(1, m_omega, V_omega)
}

#Ineffiecency Factor
IF_Gibbs = c(rep(0,7))
for(beta in 1:7){
  IF_Gibbs[beta] = 1+2*colSums(colSums(acf(betas[-1,beta], plot=FALSE)$acf))
}
for(i in 1:7){
  print(paste("IF for ", Xnames[i], " : ", IF_Gibbs[i]))
}
```

```
## [1] "IF for  Constant  :  2.86742299853657"
## [1] "IF for  HusbandInc  :  5.45766809835261"
## [1] "IF for  EducYears  :  5.26276103244083"
## [1] "IF for  ExpYears  :  5.52280755910341"
## [1] "IF for  Age  :  4.79558074940109"
## [1] "IF for  NSmallChild  :  5.80485151231444"
## [1] "IF for  NBigChild  :  4.41101120033789"
```

Plotting the trajectories of the sampled Markov chains in black with the cumulative mean in red for each of the 7 parameters:

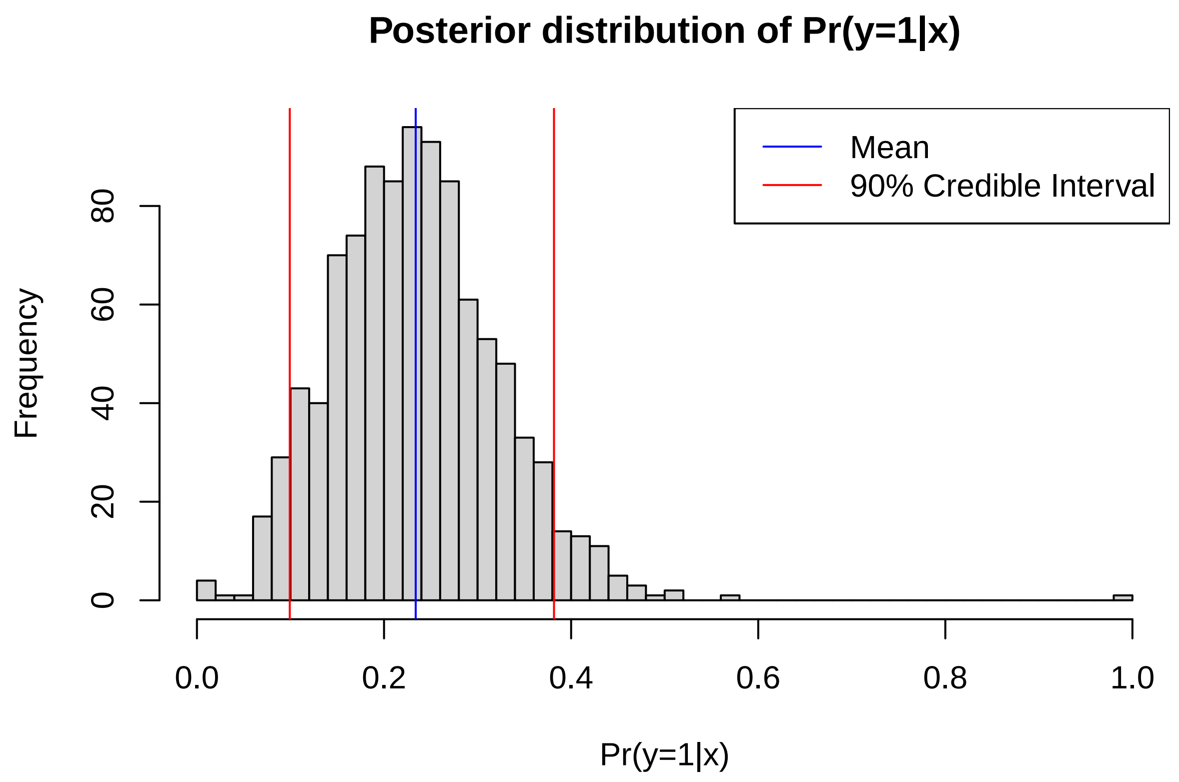
```
#Plotting the trajectories + cummulative sums of the sampled Markov chains
cumsumData = matrix(nrow=nDraws, ncol=nPar)
for(i in 1:nPar){
  cumsumData[,i] = cumsum(betas[,i])/seq(1,nDraws)
}
par(mar=c(2,2,2,1))
par(mfrow=c(4,2))
for (i in 1:7){
  plot(1:nDraws, betas[,i], type = "l", main=Xnames[i])
  lines(1:nDraws, cumsumData[,i],type="l", col="red")
}
```



b)

Use the posterior draws from a) to compute a 90% equal tail credible interval for Pr(y = 1|x), where the values of x corresponds to a 38-year-old woman, with one child (3 years old), 12 years of education, 7 years of experience, and a husband with an income of 22. A 90% equal tail credible interval (a, b) cuts off 5% percent of the posterior probability mass to the left of a, and 5% to the right of b.

```r
#New data
#Constant,  HusbandIncome, EducYears, ExpYears, Age, NSmallCHild, NBigChild
x = c(1, 22, 12, 7, 38, 1, 0)
Pr_y1 = matrix(nrow=nDraws, ncol=1)
for(draw in 1:nDraws){
  Pr_y1[draw] = exp(t(x)%*%betas[draw,])/(1+exp(t(x)%*%betas[draw,]))
}

hist(Pr_y1, breaks=50, main="Posterior distribution of Pr(y=1|x)", xlab="Pr(y=1|x)")
abline(v=quantile(Pr_y1, c(0.05, 0.95)), col="red")
abline(v=mean(Pr_y1), col="blue")
legend("topright", legend=c("Mean", "90% Credible Interval"), col=c("blue", "red"), lty=1:1)
```



```r
print(quantile(Pr_y1, c(0.05, 0.95)))
```

```
##        5%       95%
## 0.0992591 0.3816702
```

```r
print(mean(Pr_y1))
```

```
## [1] 0.2338593
```

2. Metropolis Random Walk for Poisson regression

Consider the following Poisson regression model

$$y_i|\beta \sim^{iid} Poisson[exp(x_i^T\beta)], i = 1, ..., n,$$

where y_i is the count for the ith observation in the sample and x_i is the p-dimensional vector with covariate observations for the ith observation. Use the data set eBayNumberOfBidderData_2024.dat. This dataset contains observations from 800 eBay auctions of coins. The response variable is **nBids** and records the number of bids in each auction. The remaining variables are features/covariates (**x**):

- **Const** (for the intercept)
- **PowerSeller** (equal to 1 if the seller is selling large volumes on eBay)
- **VerifyID** (equal to 1 if the seller is a veriffed seller by eBay)
- **Sealed** (equal to 1 if the coin was sold in an unopened envelope)
- **MinBlem** (equal to 1 if the coin has a minor defect)
- **MajBlem** (equal to 1 if the coin has a major defect)
- **LargNeg** (equal to 1 if the seller received a lot of negative feedback from customers)
- **LogBook** (logarithm of the book value of the auctioned coin according to expert sellers. Standardized)
- **MinBidShare** (ratio of the minimum selling price (starting price) to the book value. Standardized).

a)

Obtain the maximum likelihood estimator of β in the Poisson regression model for the eBay data [Hint: glm.R, don't forget that glm() adds its own intercept so don't input the covariate Const]. Which covariates are signifficant?

```r
ebayData <- read.table("eBayNumberOfBidderData_2024.dat", header = T)
y <- ebayData$nBids
X <- as.matrix(ebayData[,-1])
Xnames <- colnames(X)

glmFit <- glm(y ~ X - 1, family = poisson)
summary(glmFit)
```

```
##
## Call:
## glm(formula = y ~ X - 1, family = poisson)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## XConst        1.07981    0.03393  31.828  < 2e-16 ***
## XPowerSeller -0.03566    0.04167  -0.856 0.392109
## XVerifyID    -0.45564    0.12748  -3.574 0.000351 ***
## XSealed       0.45515    0.06226   7.311 2.65e-13 ***
## XMinblem     -0.06837    0.07198  -0.950 0.342228
## XMajBlem     -0.22554    0.09525  -2.368 0.017894 *
## XLargNeg      0.05382    0.06406   0.840 0.400787
## XLogBook     -0.08499    0.03234  -2.628 0.008599 **
## XMinBidShare -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: 4833.6  on 800  degrees of freedom
## Residual deviance:  691.8  on 791  degrees of freedom
## AIC: 2879.1
##
## Number of Fisher Scoring iterations: 5
```

The covariates XConst, XVerifyID, XSealed, XMajBlem, XLogBook and XMinBidShare are significant at the 95% level.

b)

Let's do a Bayesian analysis of the Poisson regression. Let the prior be $\beta \sim N[0, 100 * (X^T X)^{-1}$, where X is the n * p covariate matrix. This is a commonly used prior, which is called Zellner's g-prior. Assume first that the posterior density is approximately multivariate normal:

$$\beta|y \sim N[\tilde{\beta}, J_y^{-1}(\tilde{\beta})]$$

, where $\tilde{\beta}$ is the posterior mode and $J_y(\tilde{\beta})$ is the negative Hessian at the posterior mode. $\tilde{\beta}$ and $J_y(\tilde{\beta})$ can be obtained by numerical optimization (optim.R) exactly like you already did for the logistic regression in Lab 2 (but with the log posterior function replaced by the corresponding one for the Poisson model, which you have to code up.).

```r
# Prior
n = nrow(X)
p = ncol(X)

mu <- as.matrix(rep(0,p))
Sigma <- as.matrix(100 * solve(t(X) %*% X) )
betas = as.matrix(rep(0,p)) #Initialized to zero

posterior_function <- function(betas, y, X,  mu, Sigma){
  #Note to self: parameter to be optim is the first argument
  linearPredictor <- X %*% betas;
  logLik <- sum( linearPredictor*y - exp(linearPredictor) )
  logPrior <- dmvnorm(betas, mu, Sigma, log=TRUE);
  return(logLik + logPrior)
}

OptimRes <- optim(par = betas, fn = posterior_function, gr=NULL, y, X, mu, Sigma,
                  method=c("BFGS"),control=list(fnscale=-1),hessian=TRUE)

# Printing the results to the screen
names(OptimRes$par) <- Xnames # Naming the coefficient by covariates
approxPostStd <- sqrt(diag(solve(-OptimRes$hessian))) # Computing approximate standard deviations.
names(approxPostStd) <- Xnames # Naming the coefficient by covariates
cat('---- The posterior mode is: ----\n')
```

```
## ---- The posterior mode is: ----
```

```
print(OptimRes$par[1:9])
```

```
##       Const PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##  1.07721720 -0.03567963 -0.45353183  0.45484863 -0.06863401 -0.22583912
##     LargNeg     LogBook MinBidShare
##  0.05387677 -0.08454639 -1.82275698
```

```
cat('\n---- The approximate posterior standard deviation is: ----\n')
```

```
##
## ---- The approximate posterior standard deviation is: ----
```

```
print(approxPostStd)
```

```
##       Const PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##  0.03389556  0.04167562  0.12715595  0.06227165  0.07198300  0.09527403
##     LargNeg     LogBook MinBidShare
##  0.06408047  0.03233568  0.07826924
```

```
cat("\n---- comparing with GLM: ----\n")
```

```
##
## ---- comparing with GLM: ----
```

```
coef(glmFit)
```

```
##       XConst XPowerSeller    XVerifyID      XSealed     XMinblem     XMajBlem
##   1.07980512  -0.03566493  -0.45563760   0.45515199  -0.06836819  -0.22554138
##     XLargNeg     XLogBook XMinBidShare
##   0.05382386  -0.08498844  -1.82490142
```

c)

Let's simulate from the actual posterior of β using the Metropolis algorithm and compare the results with the approximate results in b). Program a general function that uses the Metropolis algorithm to generate random draws from an arbitrary posterior density. In order to show that it is a general function for any model, we denote the vector of model parameters by θ. Let the proposal density be the multivariate normal density mentioned in Lecture 8 (random walk Metropolis):

$$\theta_p|\theta^{(i-1)} \sim N[\theta^{(i-1)}, c * \Sigma]$$

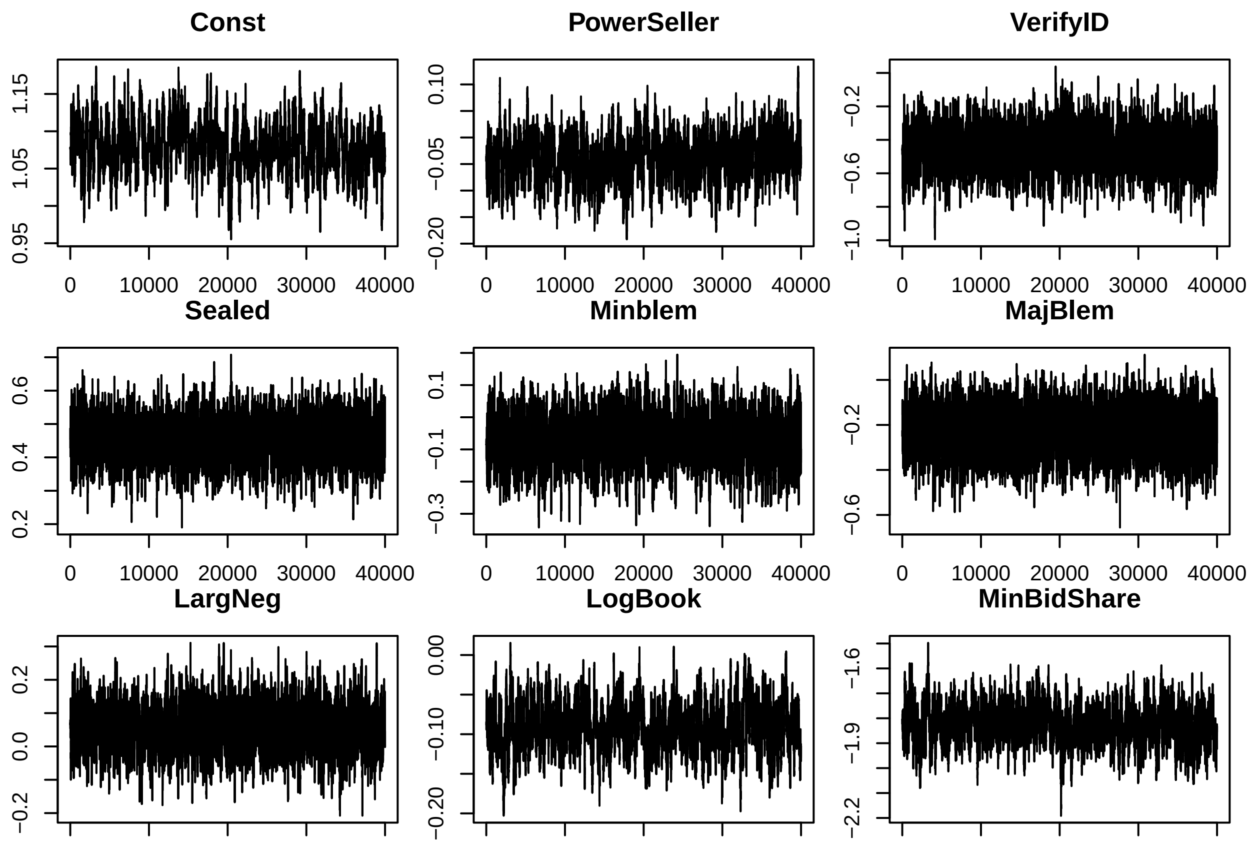
, where $\Sigma = J_y^{-1}(\tilde{\beta})$ was obtained in b). The value c is a tuning parameter and should be an input to your Metropolis function. The user of your Metropolis function should be able to supply her own posterior density function, not necessarily for the Poisson regression, and still be able to use your Metropolis function. This is not so straightforward, unless you have come across function objects in R. The note HowToCodeRWM.pdf in Lisam describes how you can do this in R.

Now, use your new Metropolis function to sample from the posterior of β in the Poisson regression for the eBay dataset. Assess MCMC convergence by graphical methods.

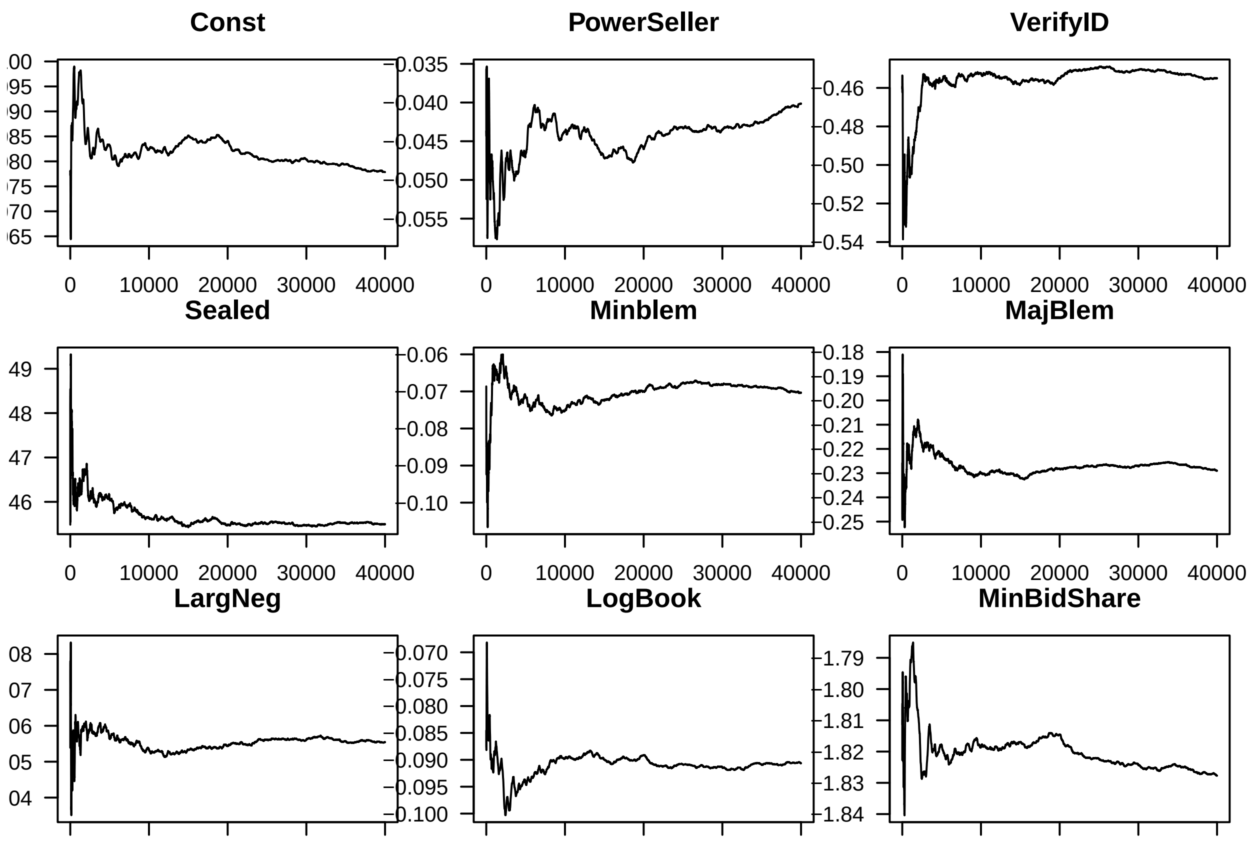
```r
# Metropolis function
Metropolis <- function(n, theta0, posterior_function, c, y, X, mu, Sigma){
  # n: number of iterations
  # theta0: initial value of theta
  # posterior_function: function that returns the log posterior
  # c: tuning parameter
  # y: response variable
  # X: covariate matrix
  # mu: prior mean
  # Sigma: prior covariance matrix

  p = length(theta0)
  theta = matrix(NA, nrow = n, ncol = p)
  theta[1,] = theta0
  accepted = 0
  for(i in 2:n){
    theta_prop = rnorm(p, theta[i-1,], diag(c*Sigma))
    logR = posterior_function(theta_prop, y, X, mu, Sigma) - posterior_function(theta[i-1,], y, X, mu, S
    if(log(runif(1)) < logR){
      theta[i,] = theta_prop
      accepted = accepted + 1
    } else {
      theta[i,] = theta[i-1,]
    }
  }
  print(paste("Acceptance rate: ", accepted/n))
  return(theta)
}

#Simulating from the posterior
n = 40000
c = 0.03
theta0 = OptimRes$par
posterior_function <- function(betas, y, X,  mu, Sigma){
  #Note to self: parameter to be optim is the first argument
  linearPredictor <- X %*% betas;
  logLik <- sum( linearPredictor*y - exp(linearPredictor) )
  logPrior <- dmvnorm(betas, mu, Sigma, log=TRUE);
  return(logLik + logPrior)
}

thetas = Metropolis(n, theta0, posterior_function, c, y, X, mu, Sigma)
```

```
## [1] "Acceptance rate:  0.290725"
```

```r
#Assessing convergence
par(mar=c(1,2,3,1))
par(mfrow=c(3,3))
for(i in 1:p){
  plot(thetas[,i], type = "l", main = Xnames[i])
}
```



```r
cummean <- function(x){
  cumsum(x) / seq_along(x)
}

#Plot the cumulative mean
for (i in 1:p){
  plot(cummean(thetas[,i]), type="l", main = Xnames[i], las=1)
}
```



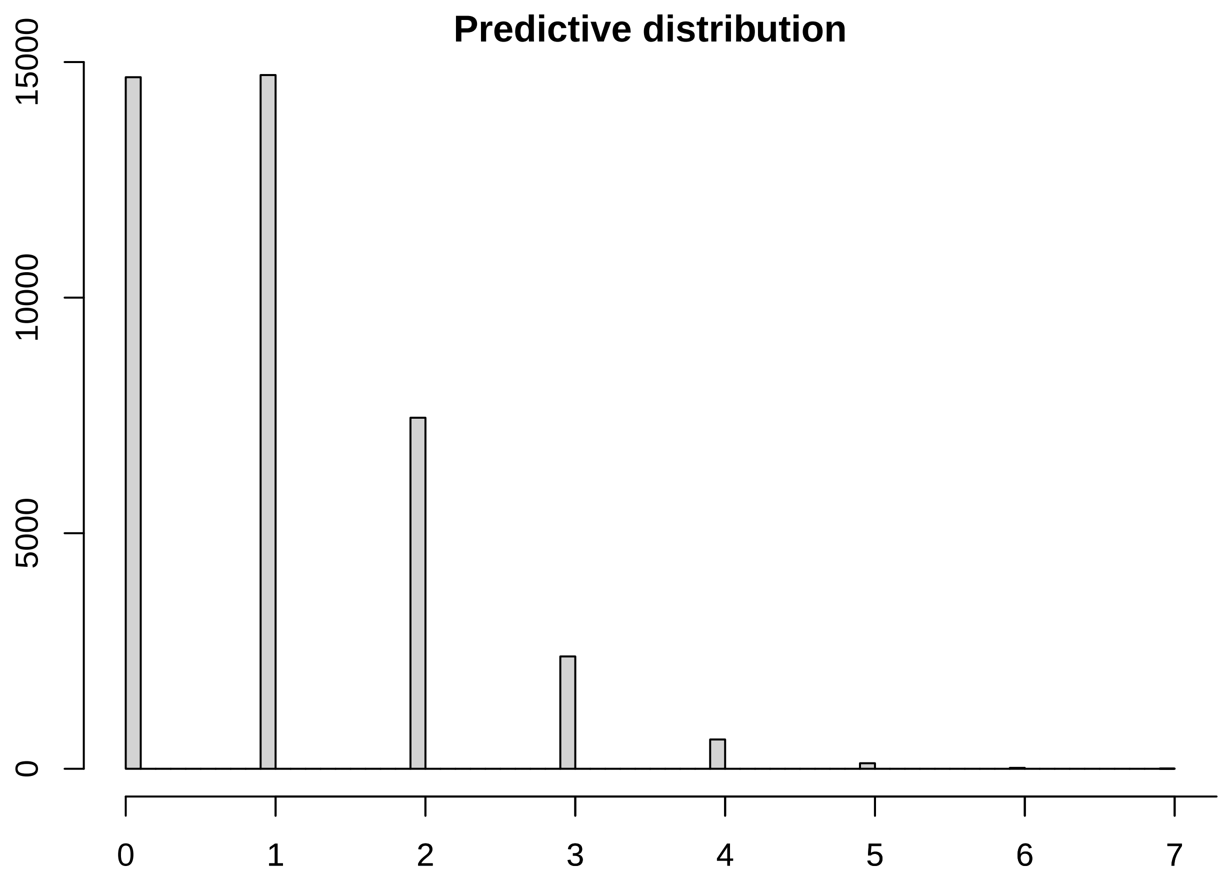
d)

Use the MCMC draws from c) to simulate from the predictive distribution of the number of bidders in a new auction with the characteristics below. Plot the predictive distribution. What is the probability of no bidders in this new auction?

- **PowerSeller** = 1
- **VerifyID** = 0
- **Sealed** = 1
- **MinBlem** = 0
- **MajBlem** = 1
- **LargNeg** = 0
- **LogBook** = 1.2
- **MinBidShare** = 0.8

```r
# New auction
Xnew = as.matrix(c(1, 0, 1, 0, 1, 0, 1.2, 0.8))

pred = c()
# Predictive distribution
for (i in 1:n){
  lambda = c(pred, exp(Xnew %*% thetas[i,]))
  pred = append(pred, rpois(1, lambda))
}
```

```r
par(mfrow=c(1,1))
par(mar=c(2,2,1,1))
hist(pred, main = "Predictive distribution", xlab = "Number of bidders", ylab = "Frequency", breaks = "S
axis(1, at = c(1:length(table(pred))), labels = c(1:length(table(pred))))
```



```r
#find the unique values in pred
pred_dist = table(pred)
print(pred_dist)
```

```
## pred
##     0     1     2     3     4     5     6     7
## 14678 14724  7451  2384   622   118    19     4
```

```r
# Probability of no bidders
prob_no_bidders = pred_dist[1]/sum(pred_dist)
cat("The probability of no bidders in the new auction is: ", prob_no_bidders)
```

```
## The probability of no bidders in the new auction is:  0.36695
```

3. Time series models in Stan

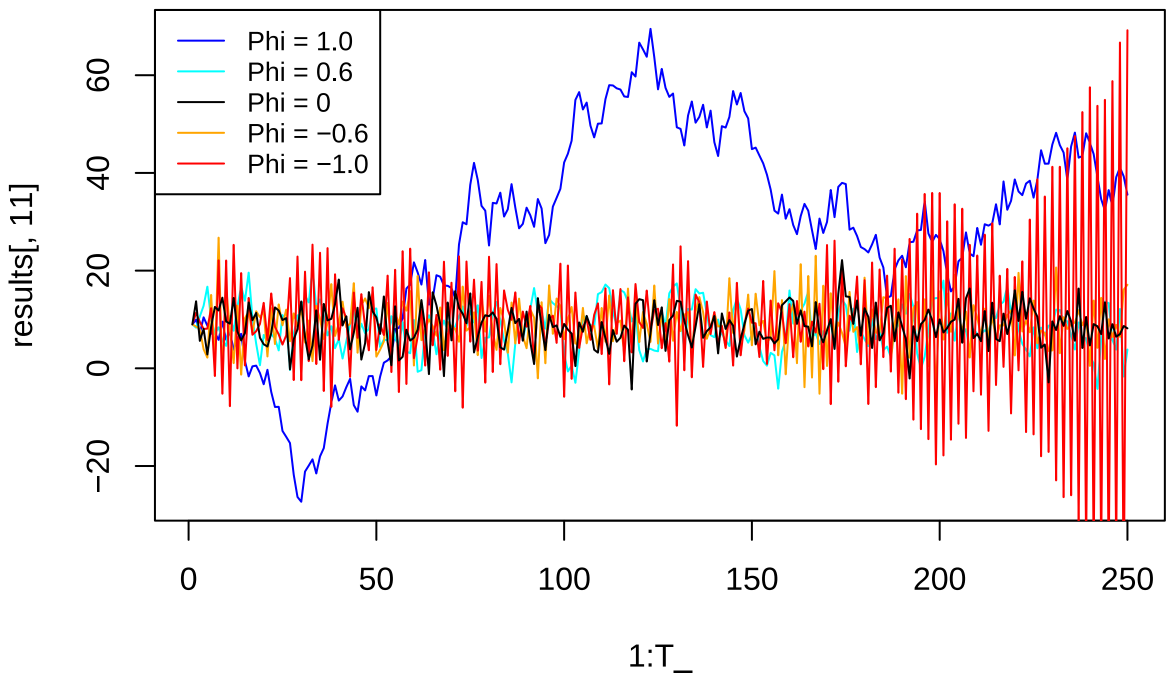
a)

Write a function in R that simulates data from the AR(1)-process

$$x_t = \mu + \phi(x_{t-1} - \mu) + \epsilon_t, \epsilon_t \sim N[0, \sigma^2]$$

, for given values of μ, ϕ and σ^2 . Start the process at $x1 = \mu$ and then simulate values for x_t for t = 2, 3. . . , T and return the vector $x_{1:T}$ containing all time points. Use $\mu = 9, \sigma^2 = 4$ and T = 250 and look at some different realizations (simulations) of $x_{1:T}$ for values of ϕ between -1 and 1 (this is the interval of ϕ where the AR(1)-process is stationary). Include a plot of at least one realization in the report. What effect does the value of ϕ have on $x_{1:T}$?

```r
set.seed(12345)
mu = 9
sig_sq = 4
T_ = 250
phis = seq(-1,1,0.2) # -1.0, -0.8, ... 0 ... 0.8, 1.0
results = matrix(nrow = T_, ncol = length(phis))
results[1,] = mu #initialise the matrix to mu for t=1

for (phi in 1:length(phis)){
  for (t in 2:T_){
    #Equation : x_t = mu + phi*      (x_t-1 - mu)             + N(0,sig_sq)
    results[t, phi] = mu + phis[phi]*(results[t-1, phi] - mu) + rnorm(1, 0, sig_sq)
  }
}
#dim(results)

#print(phis[3])
plot(x = 1:T_, y = results[,11],type='l', col = "blue") #phi = 1.0
lines(x = 1:T_, y = results[,9], col = 'cyan') #phi = 0.6

lines(x = 1:T_, y = results[,3], col='orange') #phi = -0.6
lines(x = 1:T_, y = results[,1], col='red') #phi = -1.0

lines(x = 1:T_, y = results[,6], col = 'black') #phi = 0
legend("topleft", legend=c("Phi = 1.0", "Phi = 0.6","Phi = 0","Phi = -0.6","Phi = -1.0"),
       col=c("blue","cyan","black", "orange", "red"),lty=c(1,1,1,1,1), cex=c(0.8,0.8,0.8,0.8,0.8))
```



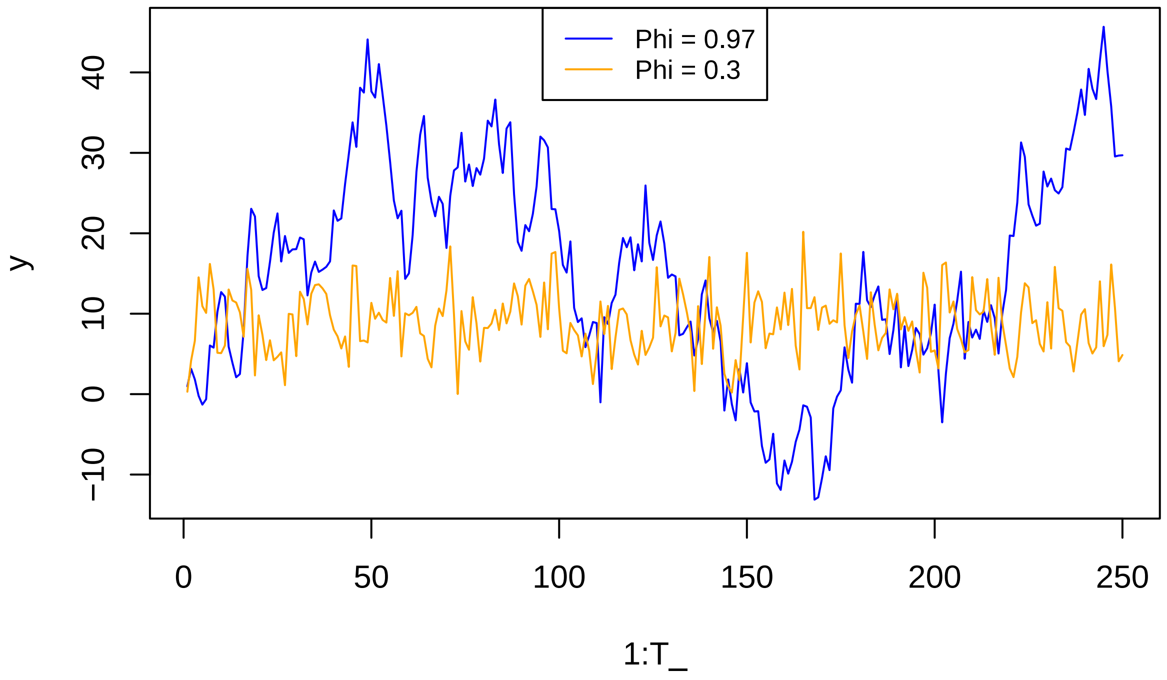
Phi values closer to 0 seems to concentrate the evolution of $x_{1:T}$ around the starting value of mu = 9. Values of +/- 1.0 does not leads to a convergence.

b)

Use your function from a) to simulate two AR(1)-processes, $x_{1:T}$ with $\phi = 0.3$ and $y_{1:T}$ with $\phi = 0.97$. Now, treat your simulated vectors as synthetic data, and treat the values of μ, ϕ and σ^2 as unknown parameters. Implement Stan-code that samples from the posterior of the three parameters, using suitable non-informative priors of your choice. [Hint: Look at the time-series models examples in the Stan user's guide/reference manual, and note the different parameterization used here.]

```r
library(rstan)
set.seed(123)
x = rep(0, 250)
mu = 9
phi = 0.3
x[1] = phi
for (t in 2:T_){
  #Equation : x_t = mu + phi*(x_t-1 - mu)+ N(0,sig_sq)
  x[t] = mu + phi*(x[t-1] - mu) + rnorm(1, 0, sig_sq)
}

y = rep(0, 250)
phi = 0.97
y[1] = phi
```

```
for (t in 2:T_){
  #Equation : x_t = mu + phi*(x_t-1 - mu)+ N(0,sig_sq)
  y[t] = mu + phi*(y[t-1] - mu) + rnorm(1, 0, sig_sq)
}

plot(x = 1:T_, y = y ,type='l', col = "blue") #phi = 0.97
lines(x = 1:T_, y = x , col = 'orange') #phi = 0.3
legend("top", legend=c("Phi = 0.97", "Phi = 0.3"), col=c("blue","orange"), lty=c(1,1), cex=c(0.8,0.8))
```



```
StanModel = '
data {
  int<lower=0> T;
  real x[T];
}
parameters {
  real mu;
  real phi;
  real<lower=0> sigma;
}
model {
  mu ~ normal(9, 30); //very flat bell curve, with mean 9, and large variance
  phi ~ uniform(-1, 1);
  sigma ~ scaled_inv_chi_square(1,2); // Scaled-inv-chi2 with nu 1,sigma 2
  for (t in 2:T){
    x[t] ~ normal(mu + phi*(x[t-1] - mu), sigma);
  }
}'

#phi = 0.3
data = list(T = T_, x = x)
fit = stan(model_code = StanModel, data = data, warmup = 1000, iter = 3000, chains = 4)
```

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

```
## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## using C compiler: ‘gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0’
## gcc -I"/usr/share/R/include" -DNDEBUG   -I"/usr/lib/R/site-library/Rcpp/include/"  -I"/home/h/R/x86_
## In file included from /home/h/R/x86_64-pc-linux-gnu-library/4.4/RcppEigen/include/Eigen/Core:19,
##                  from /home/h/R/x86_64-pc-linux-gnu-library/4.4/RcppEigen/include/Eigen/Dense:1,
##                  from /usr/lib/R/site-library/StanHeaders/include/stan/math/prim/fun/Eigen.hpp:22,
##                  from <command-line>:
## /home/h/R/x86_64-pc-linux-gnu-library/4.4/RcppEigen/include/Eigen/src/Core/util/Macros.h:679:10: fata
##   679 | #include <cmath>
##       |          ^~~~~~~
## compilation terminated.
## make: *** [/usr/lib/R/etc/Makeconf:195: foo.o] Error 1
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000161 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 1.61 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 1: Iteration:  300 / 3000 [ 10%]  (Warmup)
## Chain 1: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 1: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 1: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 1: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 1: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 1: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 1: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 1: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 1: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 1: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.546 seconds (Warm-up)
## Chain 1:                1.147 seconds (Sampling)
## Chain 1:                1.693 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2: Rejecting initial value:
## Chain 2:   Log probability evaluates to log(0), i.e. negative infinity.
## Chain 2:   Stan can't start sampling from this initial value.
## Chain 2:
## Chain 2: Gradient evaluation took 9.6e-05 seconds
```

```
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.96 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 2: Iteration:  300 / 3000 [ 10%]  (Warmup)
## Chain 2: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 2: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 2: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 2: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 2: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 2: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 2: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 2: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 2: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 2: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.59 seconds (Warm-up)
## Chain 2:                1.154 seconds (Sampling)
## Chain 2:                1.744 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.0001 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 1 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 3: Iteration:  300 / 3000 [ 10%]  (Warmup)
## Chain 3: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 3: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 3: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 3: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 3: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 3: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 3: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 3: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 3: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 3: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.56 seconds (Warm-up)
## Chain 3:                0.929 seconds (Sampling)
## Chain 3:                1.489 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 9.4e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.94 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
```

```
## Chain 4: Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 4: Iteration:  300 / 3000 [ 10%]  (Warmup)
## Chain 4: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 4: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 4: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 4: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 4: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 4: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 4: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 4: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 4: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 4: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.574 seconds (Warm-up)
## Chain 4:                1.164 seconds (Sampling)
## Chain 4:                1.738 seconds (Total)
## Chain 4:
```

```
# Print the fitted model
print(fit,digits_summary=3)
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=3000; warmup=1000; thin=1;
## post-warmup draws per chain=2000, total post-warmup draws=8000.
##
##           mean se_mean    sd     2.5%      25%      50%      75%    97.5% n_eff
## mu       8.940   0.004 0.314    8.327    8.730    8.937    9.148    9.563  7331
## phi      0.229   0.001 0.062    0.108    0.188    0.229    0.270    0.353  7375
## sigma    3.783   0.002 0.170    3.468    3.664    3.777    3.894    4.136  7962
## lp__  -456.643   0.019 1.223 -459.832 -457.214 -456.317 -455.740 -455.249  4258
##        Rhat
## mu    1.001
## phi   1.000
## sigma 1.000
## lp__  1.000
##
## Samples were drawn using NUTS(diag_e) at Mon May 27 16:31:53 2024.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
# Extract posterior samples
postDraws <- extract(fit)

#phi = 0.97
data2 = list(T = T_, x = y)
fit2 = stan(model_code = StanModel, data = data2, warmup = 1000, iter = 3000, chains = 4)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 9.7e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.97 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 1: Iteration:  300 / 3000 [ 10%]  (Warmup)
## Chain 1: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 1: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 1: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 1: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 1: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 1: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 1: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 1: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 1: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 1: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 1.777 seconds (Warm-up)
## Chain 1:                1.332 seconds (Sampling)
## Chain 1:                3.109 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 9.9e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.99 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 2: Iteration:  300 / 3000 [ 10%]  (Warmup)
## Chain 2: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 2: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 2: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 2: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 2: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 2: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 2: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 2: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 2: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 2: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 1.76 seconds (Warm-up)
## Chain 2:                1.681 seconds (Sampling)
## Chain 2:                3.441 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3: Rejecting initial value:
## Chain 3:   Log probability evaluates to log(0), i.e. negative infinity.
## Chain 3:   Stan can't start sampling from this initial value.
## Chain 3:
## Chain 3: Gradient evaluation took 9.4e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.94 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 3: Iteration:  300 / 3000 [ 10%]  (Warmup)
## Chain 3: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 3: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 3: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 3: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 3: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 3: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 3: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 3: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 3: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 3: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 2.743 seconds (Warm-up)
## Chain 3:                1.182 seconds (Sampling)
## Chain 3:                3.925 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4: Rejecting initial value:
## Chain 4:   Log probability evaluates to log(0), i.e. negative infinity.
## Chain 4:   Stan can't start sampling from this initial value.
## Chain 4:
## Chain 4: Gradient evaluation took 7.7e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.77 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 4: Iteration:  300 / 3000 [ 10%]  (Warmup)
## Chain 4: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 4: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 4: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 4: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 4: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 4: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 4: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 4: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 4: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 4: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 1.366 seconds (Warm-up)
## Chain 4:                0.717 seconds (Sampling)
## Chain 4:                2.083 seconds (Total)
## Chain 4:
```

```
## Warning: There were 689 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
# Print the fitted model
print(fit2,digits_summary=3)
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=3000; warmup=1000; thin=1;
## post-warmup draws per chain=2000, total post-warmup draws=8000.
##
##            mean se_mean    sd     2.5%      25%      50%      75%    97.5% n_eff
## mu       17.999   0.335 9.443    0.627   13.356   17.355   21.888   38.829   796
## phi       0.959   0.000 0.020    0.917    0.945    0.959    0.973    0.995  1887
## sigma     4.028   0.003 0.183    3.694    3.901    4.019    4.147    4.406  4116
## lp__   -472.589   0.033 1.239 -475.621 -473.261 -472.303 -471.646 -471.086  1397
##         Rhat
## mu     1.002
## phi    1.001
## sigma 1.000
## lp__  1.002
##
## Samples were drawn using NUTS(diag_e) at Mon May 27 16:32:06 2024.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
# Extract posterior samples
postDraws2 <- extract(fit2)
```

i)

Report the posterior mean, 95% credible intervals and the number of effective posterior samples for the three inferred parameters for each of the simulated AR(1)-process. Are you able to estimate the true values?

```
#For phi = 0.3
mu_mean = round(summary(fit)$summary[1,1],3)
mu_95ci = round(summary(fit)$summary[1,c(4,8)],3)
mu_eff = round(summary(fit)$summary[1,9])

phi_mean = round(summary(fit)$summary[2,1],3)
phi_95ci = round(summary(fit)$summary[2,c(4,8)],3)
phi_eff = round(summary(fit)$summary[2,9])

sigma_mean = round(summary(fit)$summary[3,1],3)
sigma_95ci = round(summary(fit)$summary[3,c(4,8)],3)
sigma_eff = round(summary(fit)$summary[3,9])

#For phi = 0.97
mu_mean2 = round(summary(fit2)$summary[1,1],3)
mu_95ci2 = round(summary(fit2)$summary[1,c(4,8)],3)
mu_eff2 = round(summary(fit2)$summary[1,9])

phi_mean2 = round(summary(fit2)$summary[2,1],3)
phi_95ci2 = round(summary(fit2)$summary[2,c(4,8)],3)
phi_eff2 = round(summary(fit2)$summary[2,9])
```

```r
sigma_mean2 = round(summary(fit2)$summary[3,1],3)
sigma_95ci2 = round(summary(fit2)$summary[3,c(4,8)],3)
sigma_eff2 = round(summary(fit2)$summary[3,9])
```

Using phi = 0.3:

- For μ, the mean is 8.936, with 95% credible interval of [8.337 , 9.549]. The number of effective posterior samples is 6925.
- For ϕ, the mean is 0.23, with 95% credible interval of [0.107 , 0.353]. The number of effective posterior samples is 7750.
- For σ, the mean is 3.782, with 95% credible interval of [3.463 , 4.126]. The number of effective posterior samples is 7929.

Using phi = 0.97:

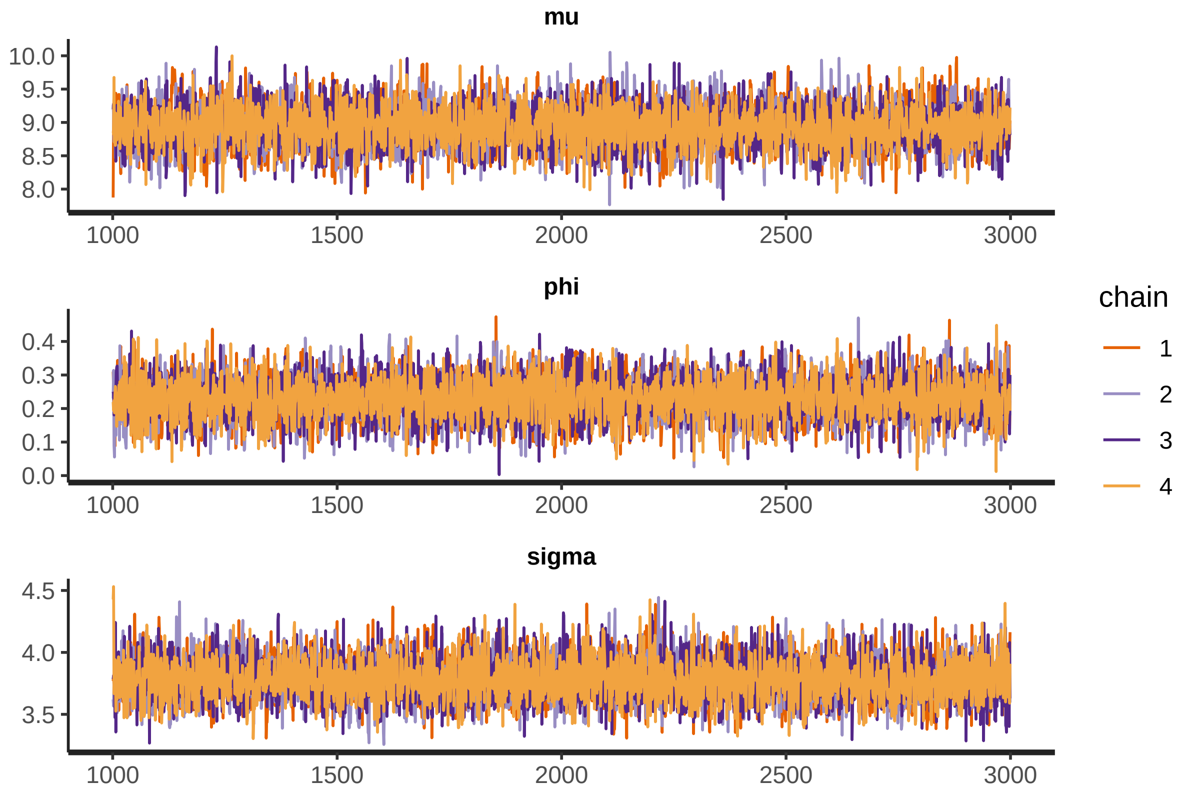
- For μ, the mean is 19.224, with 95% credible interval of [-0.54 , 49.089]. The number of effective posterior samples is 365.
- For ϕ, the mean is 0.959, with 95% credible interval of [0.917 , 0.995]. The number of effective posterior samples is 1780.
- For σ, the mean is 4.032, with 95% credible interval of [3.697 , 4.402]. The number of effective posterior samples is 3761.

ii)

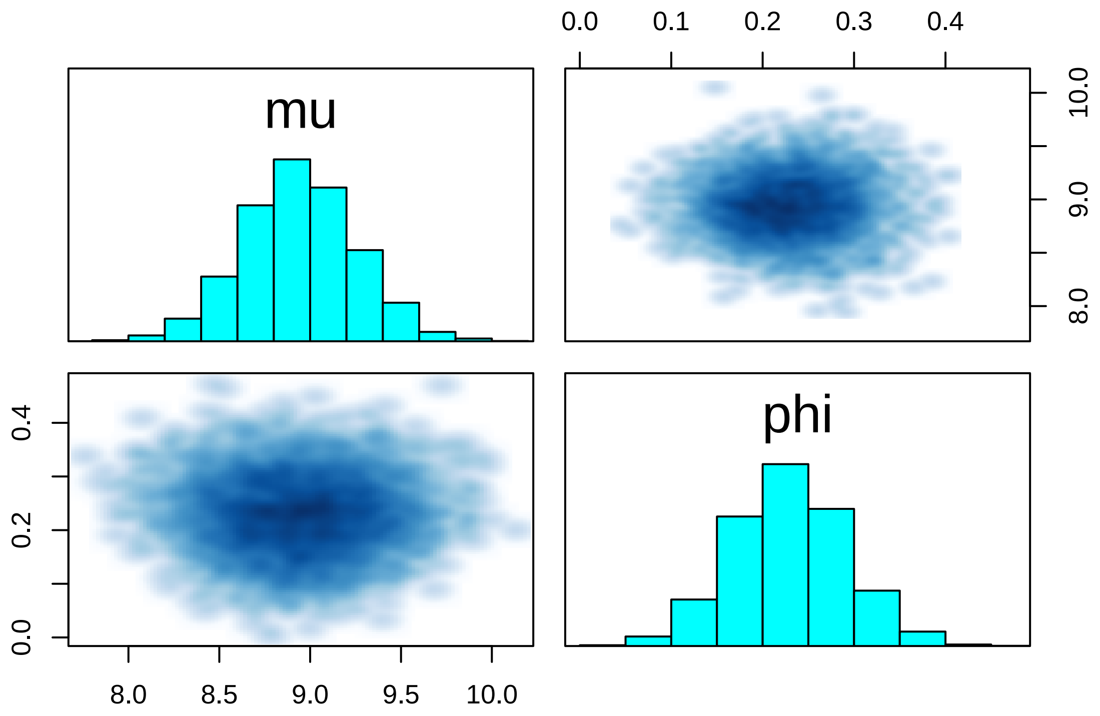
For each of the two data sets, evaluate the convergence of the samplers and plot the joint posterior of μ and ϕ. Comments?

For $\phi = 0.3$:

```r
par(mfrow = c(1,3))
#phi = 0.3
traceplot(fit, nrow = 3)
```

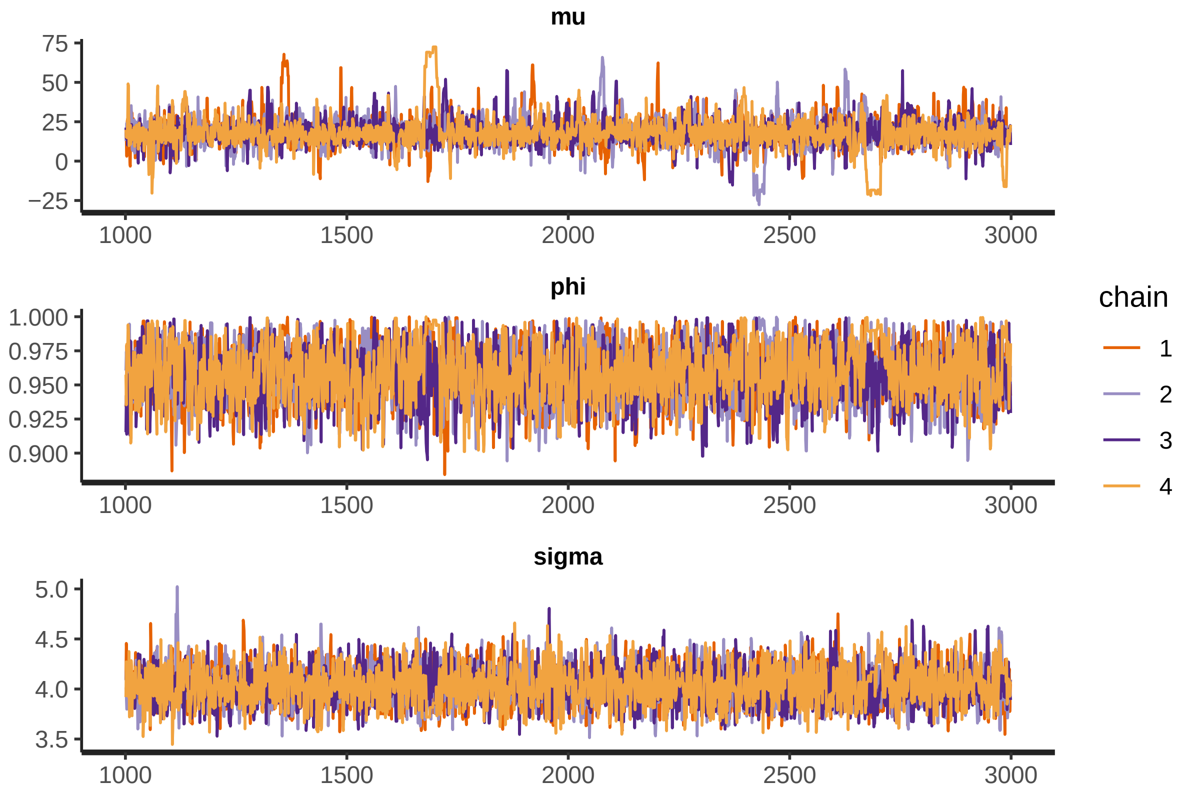


```r
#joint posterior of mu and phi
par(mfrow = c(1,1))
pairs(fit, pars = c("mu", "phi"))
```

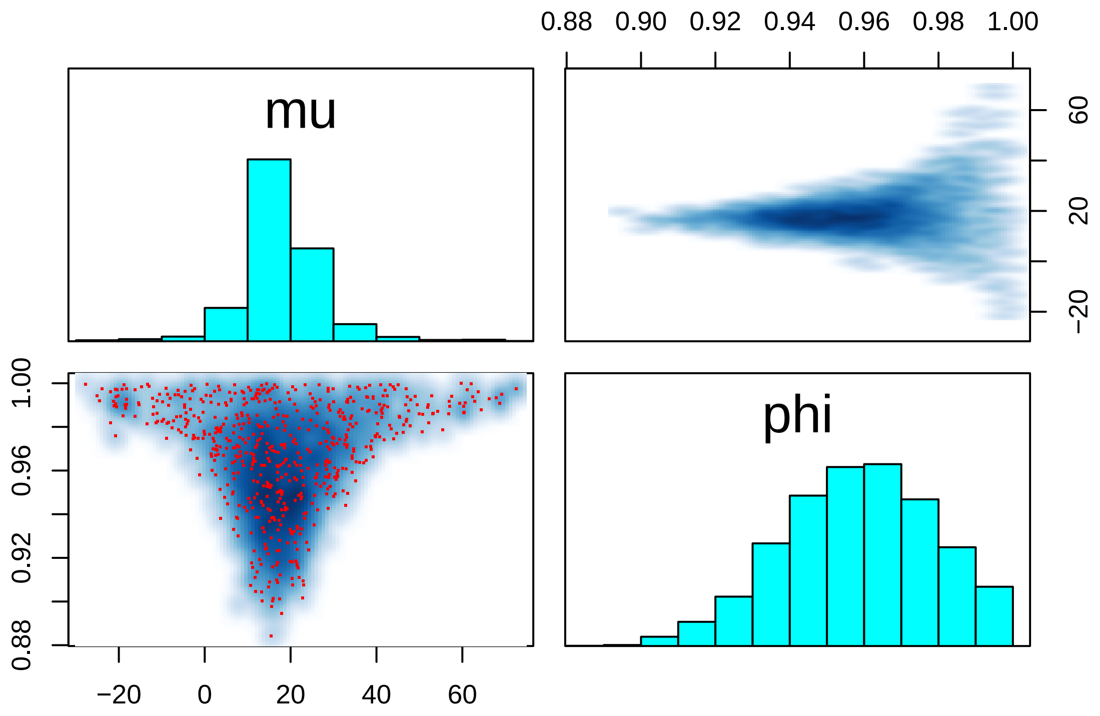


For $\phi = 0.97$:

```r
par(mfrow = c(1,3))
#phi = 0.97
traceplot(fit2, nrow = 3)
```



```r
#joint posterior of mu and phi
par(mfrow = c(1,1))
pairs(fit2, pars = c("mu", "phi"))
```



For both datasets, the samplers seem to have converged, although the tracesplot for phi=0.97 contains much more variation.

For $\phi = 0.3$, the posterior distribution of μ is centered around 9, which is the true value. The posterior distribution of ϕ is centered around 0.25, which is close to 0.3, which is the true value.

For $\phi = 0.97$, the joint posterior of μ and ϕ shows a strong correlation between the two parameters. The posterior distribution of μ is centered around 19, which quite far of the actual value of 9. The posterior distribution of ϕ seems to be centered around 0.97, or close to it, which is the true value. The big difference in the posterior distribution of μ is due to the fact that the data is generated from a process with a high ϕ value, which makes the process to be more dependent on the previous value of the process. It can be seen in the traceplot of the synthetic data that the blue curve (phi=0.97) is more often "above" the mean than "below", which gives an intuition as why the posterior mean of μ is higher than the true value. In the traceplot of of the chains for mu that there are some chains with high outliers, which may be the reason for the high posterior mean of μ. The number of effective posterior samples is also lower than for the other dataset.