732A73 Bayesian Learning

Computer lab 3

Johannes Hedström & Mikael Montén

STIMA
Department of Computer and Information Science
Linköpings universitet

2024-05-10

Contents

```r
library(coda)
library(readxl)
library(mvtnorm)
library(tidyr)
library(BayesLogit)
library(tinytex)
library(rstan)
```

1 Gibbs sampling for the logistic regression

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

1.1 a

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

```r
women <- read.table('WomenAtWork.dat', header=TRUE)

# picking out the variables from the data
y <- women$Work
X <- as.matrix(women[,-1])
set.seed(123456789)
# creating variables that are used in the sampling
tau <- 3
betas <- rep(1,7)
n_samp <- 3000
kappa <- as.matrix(y-0.5,ncol=1)
beta_samples <- matrix(NA, nrow = n_samp, ncol = 7)
beta_samples[1,] <- betas

wi <- matrix(0,nrow = nrow(X),ncol=n_samp)
i=2
j=1
for (i in 2:n_samp){

  for (j in 1:nrow(X)){

    wi[j,i-1] <- rpg(1,h=1,X[j,]%*%beta_samples[i-1,]) # polya gamma draws for data
  }
  omega <- diag(wi[,i-1])
  #B_prior <- dmvnorm(betas,rep(1,7),diag(tau ^ 2,7), log=TRUE)
```

```
  Vw <- solve(t(X)%*%omega%*%X + solve(diag(tau ^ 2,7))) # equation for Vm
  mw <- Vw %*%(t(X)%*%kappa) # equatino for mw
  beta_samples[i,] <- rmvnorm(1,mw,Vw)
  #loglik <- sum( linpred*y - log(1 + exp(linpred)))

  #B_prior+loglik

}
```

Inefficiency factor of MCMC

$$IF = 1 + 2 \cdot \sum_{k=1}^{\infty} \rho_k$$

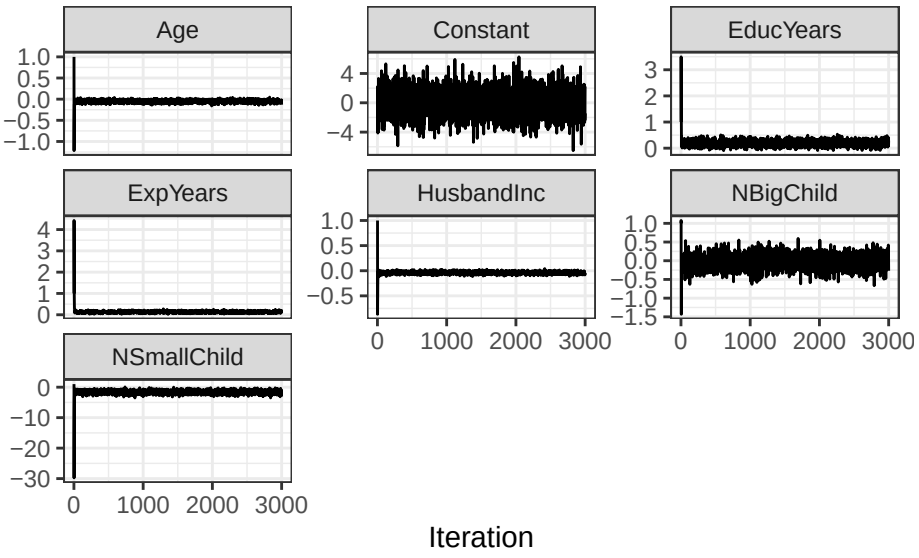
where $\rho_k = Corr(\theta^i, \theta^{(i+k)})$

```
# sum of the autocorrelations for the samples, minus the first row
sum_cor <- sum(acf(beta_samples[-1,],plot=FALSE)$acf)

IF <- 1 + 2*sum_cor # calculating the IF

# print
cat('The inefficiency factor is ',IF)
```

```
## The inefficiency factor is  5.96238
```

There is autocorrelation within the draws as we're not getting a value close to 1, we need to take around 6 times more draws then a independent draw to get the same information.

```
library(ggplot2)
# Combine the beta_samples data into a data frame suitable for ggplot
df <- data.frame(iteration = rep(seq_len(nrow(beta_samples)), ncol(beta_samples)),
                 value = c(beta_samples),
                 parameter = rep(names(women)[-1], each= nrow(beta_samples)))

 # creating several line plots
plots <- ggplot(df, aes(x = iteration, y = value)) +
  geom_line() +
  facet_wrap(~ parameter, scales = "free_y") +
  labs(x = "Iteration",y='') +
  theme_bw()

# Print the plots
print(plots)
```

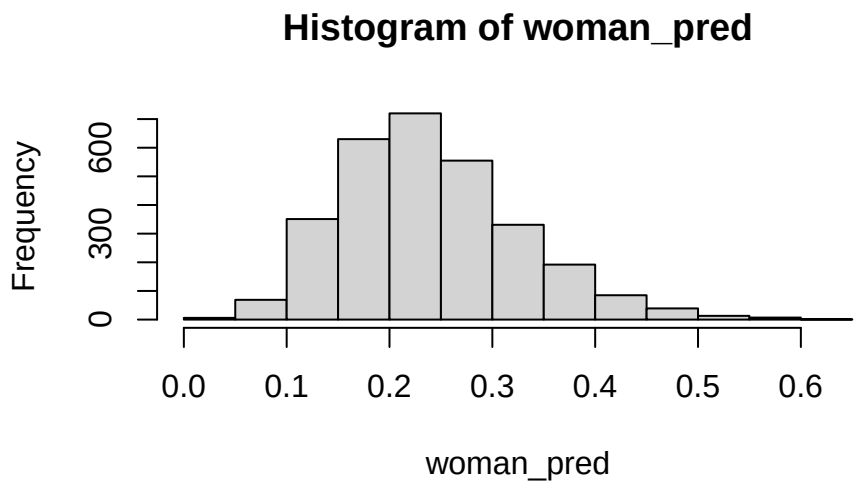


Looks like the parameters converge rather quickly.

1.2 b

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

```
x_woman <- c(1,22,12,7,38,1,0) # creating the woman

woman_pred <- c() # vector for predictions



# the logistic regression function conjugate to get prob of not working
woman_pred <- (exp(x_woman%*%t(beta_samples[-1,]))/((1+exp(x_woman%*%t(beta_samples[-1,])))))



hist(woman_pred)
```




```
colMeans(beta_samples)
```

```
## [1] -0.02846908 -0.04015258  0.19363776  0.13427990 -0.05039106 -1.63520413
## [7] -0.02039350
```

The highest density is around 25% that the woman would be working, which most likely is because she has a small child as that parameter has the largest absolute posterior mean value.

```
df_q <- data.frame(quantile(woman_pred, c(0.05, 0.95)))

colnames(df_q) <-' '

knitr::kable(df_q,digits=3, caption="10% Equal tail interval")
```

Table 1: 10% Equal tail interval

5%	0.116
95%	0.397

5 percent of the posterior probability mass is below 11.6% and above 39.7%.

2 Metropolis Random Walk for Poisson regression

Consider the following Poisson regression model

$$y_i|\beta \overset{iid}{\sim} Poisson[exp(x_i^T\beta)]$$

where yi is the count for the ith observation in the sample and xi 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 verifed 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).

2.1 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 signifcant?

```
coins <- read.table('eBayNumberOfBidderData_2024.dat', header=TRUE)


mle <- glm(nBids ~ .-Const, data = coins, family = poisson)
summary(mle)
```

```
##
## Call:
## glm(formula = nBids ~ . - Const, family = poisson, data = coins)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.3793  -0.7218  -0.0452   0.5242   2.5719
##
## 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
```

VerfyID, Sealed, LogBook and MinBidShare and MajBlem are significant on a 99% significance level.

2.2 b)

Let's do a Bayesian analysis of the Poisson regression. Let the prior be $\beta \sim N[0, 100 \cdot (X^T X)^{-1}]$, where X is the n x 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:

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

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

```
library(mvtnorm)

y <- as.matrix(coins[,1])
X <- as.matrix(coins[,-1])

n <- nrow(X)
p <- ncol(X)
Xnames <- colnames(X)

# prior
mu <- as.matrix(rep(0,p))
Sigma <- as.matrix(100*(solve(t(X) %*% X)))

poissonPost <- function(betas,y,X,mu,Sigma){

  linPred <- X%*%betas;
  logLik <- sum(linPred*y - exp(linPred)) # poisson likelihood
```

```
  logPrior <- dmvnorm(betas, mu, Sigma, log=TRUE);

  if (abs(logLik) == Inf) logLik = -20000;

  return(logLik + logPrior)
}

# Select the initial values for beta
initVal <- matrix(0,p,1)

# The argument control is a list of options to the optimizer optim, where fnscale=-1 means
# that we minimize the negative log posterior. Hence, we maximize the log posterior.
OptimRes <- optim(initVal,poissonPost,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
```

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
```

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
```

MLE coefficients:

```
coef(mle)
```

```
## (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
```

Compared to the MLE coefficients, the resulting approximate posterior mode coefficients are very similar.

2.3 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):

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

$\Sigma = J_y^{-1}(\hat{\beta})$ 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.

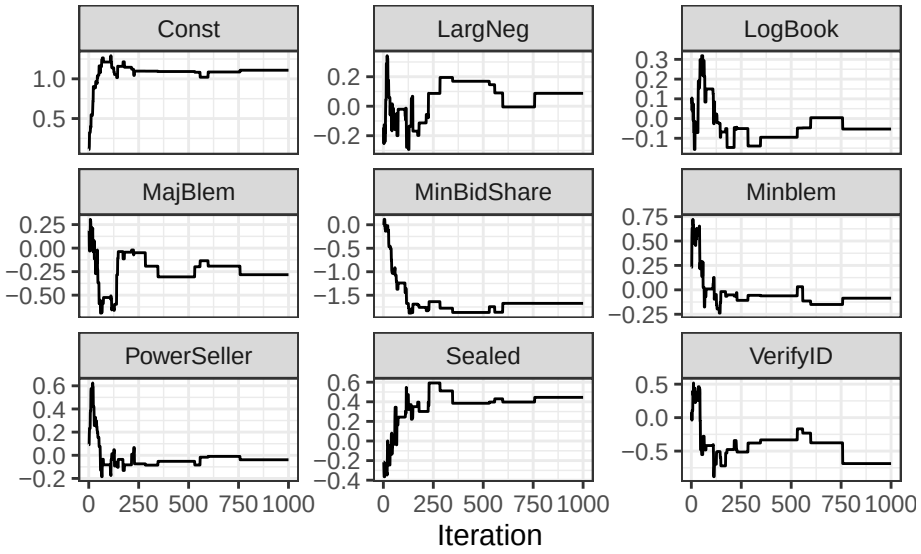
Now, use your new Metropolis function to sample from the posterior of β in the Poisson regression for the eBay dataset. Assess MCMC convergence by graphical methods

- Program general function object that uses the Metropolis algorithm to generate random draws from an arbitrary posterior density

```r
# theta is a vector of model parameters for which posterior density is evaluated, must be first argument
# logPostFunc is function object that computes log posterior density at any value of parameter vector
# C is tuning parameter
# Sigma is approximate posterior std deviation, J^(-1)_y(Beta)

RWMSampler <- function(theta, logPostFunc, C, Sigma, its, y, X, mu){

  n <- its # iterations

  theta0 <- matrix(theta, nrow = ncol(diag(Sigma)), ncol = 1) # initial matrix of theta

  theta1 <- matrix(nrow = n, ncol = ncol(diag(Sigma))) # accepted thetas
  colnames(theta1) <- colnames(X)

  # propsal density is multivariate normal (random walk metropolis)
  theta1[1,] <- rmvnorm(1, mean = theta0, sigma = C * diag(Sigma))

  for(i in 2:n){

     # theta_(i-1) is set to the previous value of proposed theta
     theta0 <- c(as.numeric(theta1[i-1,]))

     # theta_p | theta_(i-1) is calculated
     theta_p <- rmvnorm(1, mean = theta0, sigma = C * diag(Sigma))

     theta_p <- c(theta_p)

     # ratio of Metropolis posterior acceptance ratio = exp[log p(theta_p | y) - log p(theta^(i-1) | y)]
     ratio <- exp(logPostFunc(theta_p, y, X, mu, diag(Sigma)) - logPostFunc(theta0, y, X, mu, diag(Sigma

     # acceptance probability
     alpha <- min(1, ratio)


     if(alpha > runif(1,0,1)){
       theta1[i,] <- theta_p

     } else {
       theta1[i,] <- theta0
     }
  }

  return(theta1)
}
```

- Use Metropolis function to sample from posterior of β in the Poisson regression. Assess MCMC convergence graphically.

```
set.seed(1234567890)
metropolisRW <- RWMSampler(theta = 0, logPostFunc = poissonPost, C = 0.2, Sigma = approxPostStd,
                                   its = 1000, y = y, X = X, mu = mu)

# Combine the beta_samples data into a data frame suitable for ggplot
mp <- data.frame(iteration = rep(seq_len(nrow(metropolisRW)), ncol(metropolisRW)),
                 value = c(metropolisRW),
                 parameter = rep(names(coins)[-1], each= nrow(metropolisRW)))

 # creating several line plots
plots_mp<- ggplot(mp, aes(x = iteration, y = value)) +
  geom_line() +
  facet_wrap(~ parameter, scales = "free_y") +
  labs(x = "Iteration",y='') +
  theme_bw()

# Print the plots
print(plots_mp)
```



The MCMC draws seems to converge after a short burn in phase for all variables. You can see in the graph how the sampling works, where the line can be flat due to the random uniform draw is low so the value is not updated.

```
metropolisRW[1000,]
```

```
##       Const PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##  1.10842835 -0.03871684 -0.68708449  0.44500643 -0.08559367 -0.28262184
##     LargNeg     LogBook MinBidShare
##  0.08736893 -0.05321337 -1.67291240
```

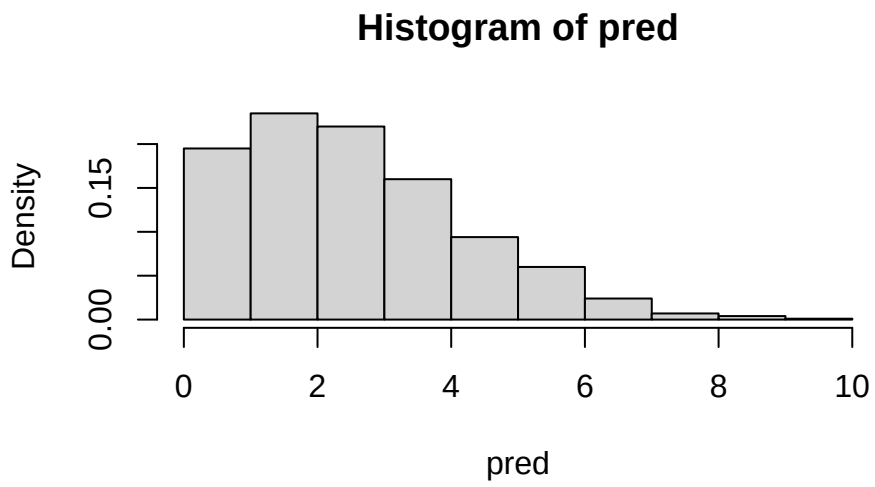
The coefficient values for the last draw is similar to the previous GLM and approximate posterior modes, albeit slightly different.

2.4 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

```
set.seed(1234567890)

# constant + new bidder values
newBidder <- as.matrix(c(1,1,0,1,0,1,0,1.2,0.8))

pred <- c()
for(i in 1:nrow(metropolisRW)){
  # calculate lambda for poisson regression model
  # and create vector of predictive values from random walk function
  lambda <- exp(newBidder %*% metropolisRW[i,])
  pred[i] <- rpois(1,lambda)

}

hist(pred, freq = FALSE)
```




```
table(pred)[1]/1000
```

```
##     0
## 0.057
```

The resulting distribution of predicted bidders follows a Poisson distribution which is expected. A total of 57 predictions for no bidders, which equals a probability of $57/1000 = 0.057 = 5.7\%$ of no one bidding.

3 Time series models in Stan

3.1 a)

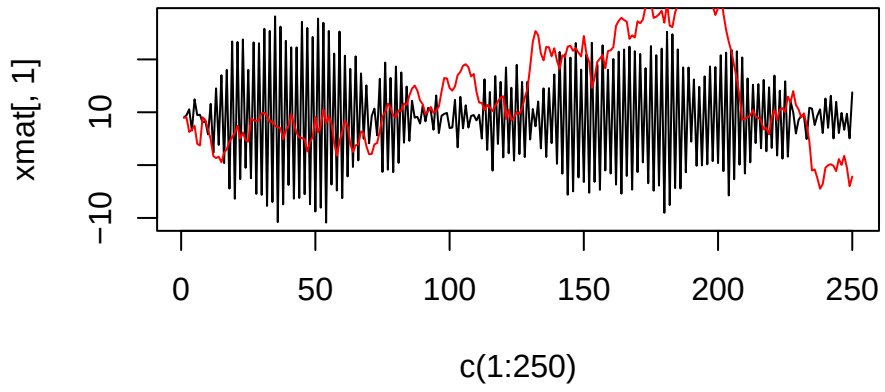
Write a function in R that simulates data from the AR(1)-process

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

for given values of μ, ϕ and σ^2. Start the process at x1 = μ and then simulate values for xt for t = 2,3…T and return the vector x1:T containing all time points. Use $\mu = 9$, $\sigma^2 = 4$ and T = 250 and look at some different realizations (simulations) of x1: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 $x_{1:T}$

```
mu <- 9
sigma <- 4
phi <- seq(-1,1,by=0.1)
xmat <- matrix(mu,ncol=length(phi), nrow=250)
```

```r
for (j in 1:length(phi)){# looping over different phi values
  for (i in 2:250){ # looping over all time stamps
  xmat[i,j] <- mu + phi[j]*(xmat[(i-1),j] - mu) + rnorm(1,0,sqrt(sigma))
}
}

plot(x=c(1:250), y=xmat[,1],type='l')
lines(xmat[,21], col='red')
```



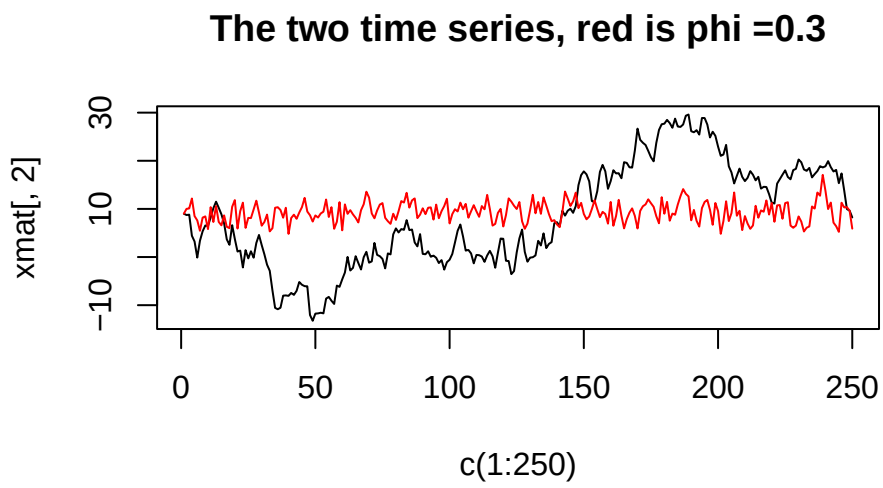
The value of phi makes how xt should depend on the previous value, positive/negative autocorrelation, we can see that -1(black line) creates the time serie that goes from positive to negative every other value which indicates negative autocorrelation and that phi = 1 makes the red line where the values follow each other closly and have a high autocorrelation.

3.2 b)

Use your function from a) to simulate two AR(1)-processes, x1:T with $\phi = 0.3$ and y1: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 Stancode 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.]

- Report the posterior mean, 95% credible intervals and the number of effective posterior samples for the three inferred parameters for each of thesimulated AR(1)-process. Are you able to estimate the true values?
- For each of the two data sets, evaluate the convergence of the samplers and plot the joint posterior of μ and ϕ. Comments?

```r
set.seed(123456789)

mu <- 9
sigma <- 4
phi2 <- c(0.3,0.97)
xmat <- matrix(mu,ncol=2, nrow=250)

for (j in 1:length(phi2)){# looping over different phi values
  for (i in 2:250){ # looping over all time stamps
  xmat[i,j] <- mu + phi2[j]*(xmat[(i-1),j] - mu) + rnorm(1,0,sqrt(sigma))
}
}

plot(x=c(1:250), y=xmat[,2],type='l', main='The two time series, red is phi =0.3')
lines(xmat[,1], col='red')
```




```r
# stan model phi = 0.3, code from slides (changed)

y=xmat[,1]
N=length(y)


StanModel = '
data {
  int<lower=0> N; // Number of observations
  real y[N];
}
parameters {
  real mu;
```

```
  real<lower=-1,upper=1> phi; // creating phi with the interval -1,1
  real<lower=0> sigma2;
}
model {
  mu ~ normal(9,20); // Normal with mean 9, st.dev. 20
  phi ~ uniform(-1,1); //  uniform in the interal
  sigma2 ~ scaled_inv_chi_square(1,2); // Scaled-inv-chi2 with nu 1,sigma 2
  for(i in 2:N){
    y[i] ~ normal(mu + phi * (y[i-1] - mu),sqrt(sigma2));
  }
}'
```

```
data <- list(N=N, y=y)
warmup <- 1000
niter <- 2000
fit <- stan(model_code=StanModel,data=data, warmup=warmup,iter=niter,chains=4)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000107 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 1.07 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.194 seconds (Warm-up)
## Chain 1:                0.197 seconds (Sampling)
## Chain 1:                0.391 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 3.3e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.33 seconds.
## Chain 2: Adjust your expectations accordingly!
```

```
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.174 seconds (Warm-up)
## Chain 2:                0.186 seconds (Sampling)
## Chain 2:                0.36 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 3.5e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.35 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.201 seconds (Warm-up)
## Chain 3:                0.159 seconds (Sampling)
## Chain 3:                0.36 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 3.1e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.31 seconds.
```

```
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.175 seconds (Warm-up)
## Chain 4:                0.172 seconds (Sampling)
## Chain 4:                0.347 seconds (Total)
## Chain 4:
```

```
# Print the fitted model
#print(fit,digits_summary=3)
fitsum <- summary(fit)$summary[c(1,2,3),c(1,4,8,9)]
# Extract posterior samples

# Do traceplots of the first chain
#par(mfrow = c(1,1))
#plot(postDraws$mu[1:(niter-warmup)],type="l",ylab="mu",main="Traceplot")
# Do automatic traceplots of all chains
#traceplot(fit)
# Bivariate posterior plots
#pairs(fit)
knitr::kable(fitsum,caption = 'phi = 0.3')
```

Table 2: phi = 0.3

	mean	2.5%	97.5%	n_eff
mu	9.303049	8.9304339	9.6797036	3648.160
phi	0.310892	0.1874938	0.4311182	3959.470
sigma2	4.177366	3.4856605	4.9711657	3764.764

The posterior mean for mu, phi and sigma are close to the true values for the time serie, and all parameters have over 3500 efficient draws.

```
# stan model phi = 0.97
```

```
y=xmat[,2]
N=length(y)


StanModel = '
data {
  int<lower=0> N; // Number of observations
  real y[N];
}
parameters {
  real mu;
  real<lower=-1,upper=1> phi; // creating phi with the interval -1,1
  real<lower=0> sigma2;
}
model {
  mu ~ normal(9,20); // Normal with mean 9, st.dev. 20
  phi ~ uniform(-1,1); //  uniform in the interal
  sigma2 ~  scaled_inv_chi_square(1,2); // Scaled-inv-chi2 with nu 1,sigma 2
  for(i in 2:N){
    y[i] ~ normal(mu + phi * (y[i-1] - mu),sqrt(sigma2));
  }
}'
```

```
data <- list(N=N, y=y)
warmup <- 1000
niter <- 2000
fit2 <- stan(model_code=StanModel,data=data, warmup=warmup,iter=niter,chains=4)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 8.9e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.89 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
```

```
## Chain 1:
## Chain 1:  Elapsed Time: 0.432 seconds (Warm-up)
## Chain 1:                0.278 seconds (Sampling)
## Chain 1:                0.71 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 3.4e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.34 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.514 seconds (Warm-up)
## Chain 2:                0.26 seconds (Sampling)
## Chain 2:                0.774 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 3.3e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.33 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
```

```
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.394 seconds (Warm-up)
## Chain 3:                0.296 seconds (Sampling)
## Chain 3:                0.69 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 4.9e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.49 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.382 seconds (Warm-up)
## Chain 4:                0.338 seconds (Sampling)
## Chain 4:                0.72 seconds (Total)
## Chain 4:
```

```
# Print the fitted model
fit2sum <- summary(fit2)$summary[c(1,2,3),c(1,4,8,9)]

knitr::kable(fit2sum,caption = 'phi = 0.97')
```

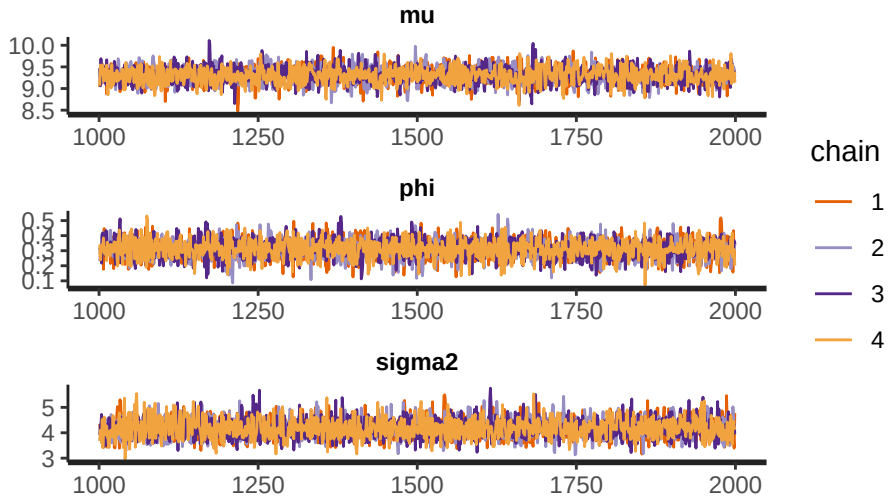
Table 3: phi = 0.97

	mean	2.5%	97.5%	n_eff
mu	8.5236416	-15.2849269	33.7546042	1432.573
phi	0.9856238	0.9629817	0.9991961	1210.263
sigma2	4.0157767	3.3595703	4.7798564	1707.218

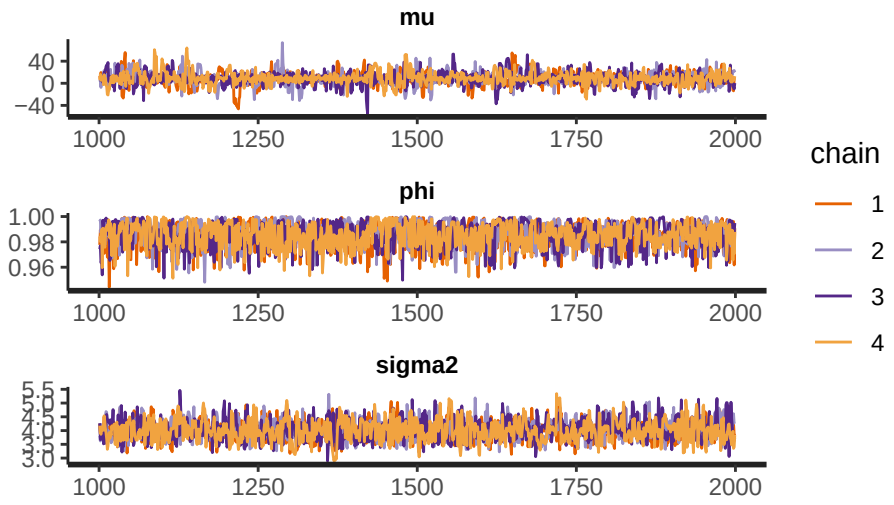
The time serie with phi = 0.97 also have posterior means close to the true values but the credible interval for mu is much wider and even cover 0, but the other credible intervals are tighter. The number of efficient draws is also much lower for this value of phi, this has to do with the correlation between phi and mu when phi has

a high value. As phi = 0.97 every draw will change more and will also be dependent on the drawn mu, this might explain the high variance for mu.

```
# Do traceplots

traceplot(fit, warmup=TRUE, nrow=3)
```

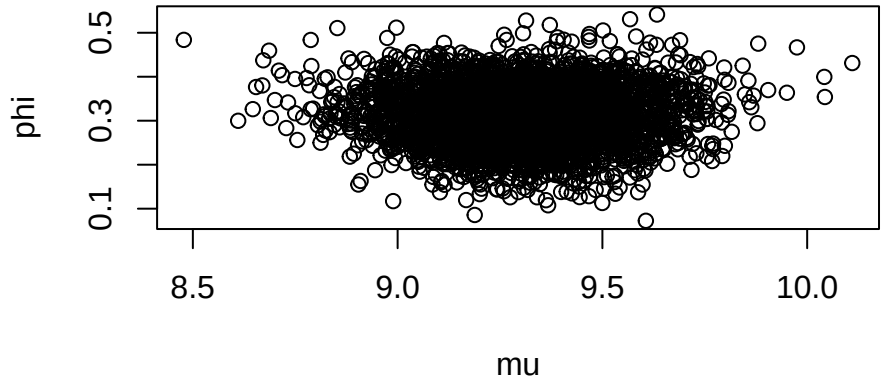



```
# eval convergence
postDraws2 <- extract(fit2)
# Do traceplots of the  chain

traceplot(fit2, warmup=TRUE, nrow=3)
```



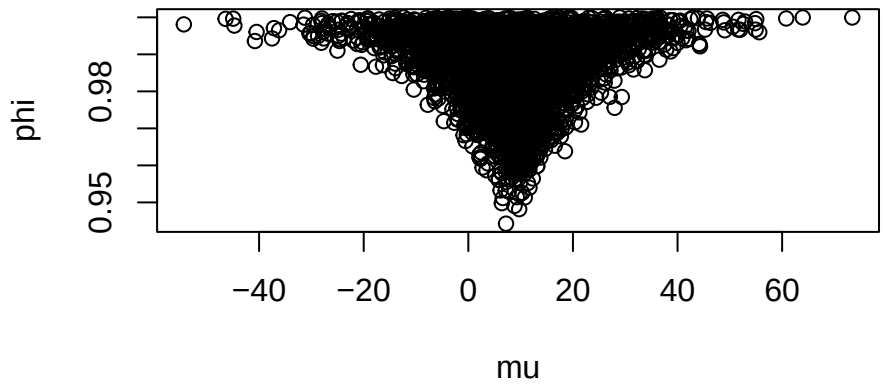
Looking at the traceplots for the parameters for the two time series we can see that all parameters seems to converge quickly as they go close and vary around the true value after very few iterations. Whats notable is that mu for the timeserie with phi = 0.97 is that this vary much more than all other parameters and it also seems to have some outliers here and there.

```
# joint posterior of phi = 0.3
postDraws <- extract(fit)
plot(postDraws$mu,y=postDraws$phi, type='p', ylab='phi', xlab='mu')
```



The values for phi and mu are close to the center which has the highest density and are where the true values also are. There doesn't look like there is any correlation between phi and mu, as there isnt any clear pattern.

```r
# joint posterior of phi = 0.97
postDraws2 <- extract(fit2)
plot(postDraws2$mu,y=postDraws2$phi, type='p', ylab='phi', xlab='mu')
```



For phi = 0.97 we have a upper bound where phi cant go over 1 so we get a border for phi there, but we can see that the values for phi centered around its posterior mean of 0.985. The variance for mu is large and the pattern of we can see with the downward and upward trend is the high autocorrelation, a negative value will be followed by a negative and a positive with a positive etc.

4 Appendix

```
knitr::opts_chunk$set(echo = TRUE, message = FALSE, warning=FALSE, fig.width = 5, fig.height = 3, fig.al
set.seed(12345)
library(coda)
library(readxl)
library(mvtnorm)
library(tidyr)
library(BayesLogit)
library(tinytex)
library(rstan)
women <- read.table('WomenAtWork.dat', header=TRUE)

# picking out the variables from the data
y <- women$Work
X <- as.matrix(women[,-1])
set.seed(123456789)
# creating variables that are used in the sampling
tau <- 3
betas <- rep(1,7)
n_samp <- 3000
kappa <- as.matrix(y-0.5,ncol=1)
beta_samples <- matrix(NA, nrow = n_samp, ncol = 7)
beta_samples[1,] <- betas

wi <- matrix(0,nrow = nrow(X),ncol=n_samp)
i=2
j=1
for (i in 2:n_samp){

  for (j in 1:nrow(X)){

    wi[j,i-1] <- rpg(1,h=1,X[j,]%*%beta_samples[i-1,]) # polya gamma draws for data
  }
  omega <- diag(wi[,i-1])
  #B_prior <- dmvnorm(betas,rep(1,7),diag(tau ^ 2,7), log=TRUE)

  Vw <- solve(t(X)%*%omega%*%X + solve(diag(tau ^ 2,7))) # equation for Vm
  mw <- Vw %*%(t(X)%*%kappa) # equatino for mw
  beta_samples[i,] <- rmvnorm(1,mw,Vw)
  #loglik <- sum( linpred*y - log(1 + exp(linpred)))

  #B_prior+loglik

}
```

```r
# sum of the autocorrelations for the samples, minus the first row
sum_cor <- sum(acf(beta_samples[-1,],plot=FALSE)$acf)

IF <- 1 + 2*sum_cor # calculating the IF

# print
cat('The inefficiency factor is ',IF)

library(ggplot2)
# Combine the beta_samples data into a data frame suitable for ggplot
df <- data.frame(iteration = rep(seq_len(nrow(beta_samples)), ncol(beta_samples)),
                 value = c(beta_samples),
                 parameter = rep(names(women)[-1], each= nrow(beta_samples)))

 # creating several line plots
plots <- ggplot(df, aes(x = iteration, y = value)) +
  geom_line() +
  facet_wrap(~ parameter, scales = "free_y") +
  labs(x = "Iteration",y='') +
  theme_bw()

# Print the plots
print(plots)
x_woman <- c(1,22,12,7,38,1,0) # creating the woman

woman_pred <- c() # vector for predictions



# the logistic regression function conjugate to get prob of not working
woman_pred <- (exp(x_woman%*%t(beta_samples[-1,]))/((1+exp(x_woman%*%t(beta_samples[-1,])))))



hist(woman_pred)
colMeans(beta_samples)
df_q <- data.frame(quantile(woman_pred, c(0.05, 0.95)))

colnames(df_q) <-' '

knitr::kable(df_q,digits=3, caption="10% Equal tail interval")
coins <- read.table('eBayNumberOfBidderData_2024.dat', header=TRUE)


mle <- glm(nBids ~ .-Const, data = coins, family = poisson)
summary(mle)
library(mvtnorm)
```

```r
y <- as.matrix(coins[,1])
X <- as.matrix(coins[,-1])

n <- nrow(X)
p <- ncol(X)
Xnames <- colnames(X)

# prior
mu <- as.matrix(rep(0,p))
Sigma <- as.matrix(100*(solve(t(X) %*% X)))

poissonPost <- function(betas,y,X,mu,Sigma){

  linPred <- X%*%betas;
  logLik <- sum(linPred*y - exp(linPred)) # poisson likelihood
  logPrior <- dmvnorm(betas, mu, Sigma, log=TRUE);

  if (abs(logLik) == Inf) logLik = -20000;

  return(logLik + logPrior)
}

# Select the initial values for beta
initVal <- matrix(0,p,1)

# The argument control is a list of options to the optimizer optim, where fnscale=-1 means
# that we minimize the negative log posterior. Hence, we maximize the log posterior.
OptimRes <- optim(initVal,poissonPost,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
print(OptimRes$par[1:9])
print(approxPostStd)
coef(mle)
# theta is a vector of model parameters for which posterior density is evaluated, must be first argument
# logPostFunc is function object that computes log posterior density at any value of parameter vector
# C is tuning parameter
# Sigma is approximate posterior std deviation, J^(-1)_y(Beta)

RWMSampler <- function(theta, logPostFunc, C, Sigma, its, y, X, mu){

  n <- its # iterations

  theta0 <- matrix(theta, nrow = ncol(diag(Sigma)), ncol = 1) # initial matrix of theta
```

```
  theta1 <- matrix(nrow = n, ncol = ncol(diag(Sigma))) # accepted thetas
  colnames(theta1) <- colnames(X)

  # propsal density is multivariate normal (random walk metropolis)
  theta1[1,] <- rmvnorm(1, mean = theta0, sigma = C * diag(Sigma))

  for(i in 2:n){

     # theta_(i-1) is set to the previous value of proposed theta
     theta0 <- c(as.numeric(theta1[i-1,]))

     # theta_p | theta_(i-1) is calculated
     theta_p <- rmvnorm(1, mean = theta0, sigma = C * diag(Sigma))

     theta_p <- c(theta_p)

     # ratio of Metropolis posterior acceptance ratio = exp[log p(theta_p | y) - log p(theta^(i-1) | y)]
     ratio <- exp(logPostFunc(theta_p, y, X, mu, diag(Sigma)) - logPostFunc(theta0, y, X, mu, diag(Sigma

     # acceptance probability
     alpha <- min(1, ratio)


     if(alpha > runif(1,0,1)){
       theta1[i,] <- theta_p

     } else {
       theta1[i,] <- theta0
     }
  }

  return(theta1)
}

set.seed(1234567890)
metropolisRW <- RWMSampler(theta = 0, logPostFunc = poissonPost, C = 0.2, Sigma = approxPostStd,
                                    its = 1000, y = y, X = X, mu = mu)

# Combine the beta_samples data into a data frame suitable for ggplot
mp <- data.frame(iteration = rep(seq_len(nrow(metropolisRW)), ncol(metropolisRW)),
                 value = c(metropolisRW),
                 parameter = rep(names(coins)[-1], each= nrow(metropolisRW)))

 # creating several line plots
plots_mp<- ggplot(mp, aes(x = iteration, y = value)) +
  geom_line() +
  facet_wrap(~ parameter, scales = "free_y") +
  labs(x = "Iteration",y='') +
  theme_bw()
```

```r
# Print the plots
print(plots_mp)
metropolisRW[1000,]
set.seed(1234567890)

# constant + new bidder values
newBidder <- as.matrix(c(1,1,0,1,0,1,0,1.2,0.8))

pred <- c()
for(i in 1:nrow(metropolisRW)){
  # calculate lambda for poisson regression model
  # and create vector of predictive values from random walk function
  lambda <- exp(newBidder %*% metropolisRW[i,])
  pred[i] <- rpois(1,lambda)

}

hist(pred, freq = FALSE)

table(pred)[1]/1000

mu <- 9
sigma <- 4
phi <- seq(-1,1,by=0.1)
xmat <- matrix(mu,ncol=length(phi), nrow=250)

for (j in 1:length(phi)){# looping over different phi values
  for (i in 2:250){ # looping over all time stamps
  xmat[i,j] <- mu + phi[j]*(xmat[(i-1),j] - mu) + rnorm(1,0,sqrt(sigma))
}
}

plot(x=c(1:250), y=xmat[,1],type='l')
lines(xmat[,21], col='red')
set.seed(123456789)

mu <- 9
sigma <- 4
phi2 <- c(0.3,0.97)
xmat <- matrix(mu,ncol=2, nrow=250)

for (j in 1:length(phi2)){# looping over different phi values
  for (i in 2:250){ # looping over all time stamps
  xmat[i,j] <- mu + phi2[j]*(xmat[(i-1),j] - mu) + rnorm(1,0,sqrt(sigma))
}
}

plot(x=c(1:250), y=xmat[,2],type='l', main='The two time series, red is phi =0.3')
```

```r
lines(xmat[,1], col='red')
# stan model phi = 0.3, code from slides (changed)

y=xmat[,1]
N=length(y)


StanModel = '
data {
  int<lower=0> N; // Number of observations
  real y[N];
}
parameters {
  real mu;
  real<lower=-1,upper=1> phi; // creating phi with the interval -1,1
  real<lower=0> sigma2;
}
model {
  mu ~ normal(9,20); // Normal with mean 9, st.dev. 20
  phi ~ uniform(-1,1); //  uniform in the interal
  sigma2 ~ scaled_inv_chi_square(1,2); // Scaled-inv-chi2 with nu 1,sigma 2
  for(i in 2:N){
    y[i] ~ normal(mu + phi * (y[i-1] - mu),sqrt(sigma2));
  }
}'
data <- list(N=N, y=y)
warmup <- 1000
niter <- 2000
fit <- stan(model_code=StanModel,data=data, warmup=warmup,iter=niter,chains=4)
# Print the fitted model
#print(fit,digits_summary=3)
fitsum <- summary(fit)$summary[c(1,2,3),c(1,4,8,9)]
# Extract posterior samples

# Do traceplots of the first chain
#par(mfrow = c(1,1))
#plot(postDraws$mu[1:(niter-warmup)],type="l",ylab="mu",main="Traceplot")
# Do automatic traceplots of all chains
#traceplot(fit)
# Bivariate posterior plots
#pairs(fit)
knitr::kable(fitsum,caption = 'phi = 0.3')

# stan model phi = 0.97

y=xmat[,2]
N=length(y)
```

```
StanModel = '
data {
  int<lower=0> N; // Number of observations
  real y[N];
}
parameters {
  real mu;
  real<lower=-1,upper=1> phi; // creating phi with the interval -1,1
  real<lower=0> sigma2;
}
model {
  mu ~ normal(9,20); // Normal with mean 9, st.dev. 20
  phi ~ uniform(-1,1); //  uniform in the interal
  sigma2 ~  scaled_inv_chi_square(1,2); // Scaled-inv-chi2 with nu 1,sigma 2
  for(i in 2:N){
    y[i] ~ normal(mu + phi * (y[i-1] - mu),sqrt(sigma2));
  }
}'
data <- list(N=N, y=y)
warmup <- 1000
niter <- 2000
fit2 <- stan(model_code=StanModel,data=data, warmup=warmup,iter=niter,chains=4)
# Print the fitted model
fit2sum <- summary(fit2)$summary[c(1,2,3),c(1,4,8,9)]

knitr::kable(fit2sum,caption = 'phi = 0.97')
# Do traceplots

traceplot(fit, warmup=TRUE, nrow=3)
# eval convergence
postDraws2 <- extract(fit2)
# Do traceplots of the  chain

traceplot(fit2, warmup=TRUE, nrow=3)
# joint posterior of phi = 0.3
postDraws <- extract(fit)
plot(postDraws$mu,y=postDraws$phi, type='p', ylab='phi', xlab='mu')
# joint posterior of phi = 0.97
postDraws2 <- extract(fit2)
plot(postDraws2$mu,y=postDraws2$phi, type='p', ylab='phi', xlab='mu')
```