Laboration report in Bayesian Statistics

Bayesian Learning, Computer Lab 3

732A91

Duc Tran
William Wiik

Division of Statistics and Machine Learning
Department of Computer Science
Linköping University

10 May 2024

Question 1:

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

a)

Question:

Implement (code!) a Gibbs sampler that simulates from the joint posterior $p(w, \beta \mathbf{x})$ 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 Ine ciency Factors (IFs) and by plotting the trajectories of the sampled Markov chains.

Answer:

The gibbs sampling is implemented as follows.

```
data <- read.table("WomenAtWork.dat", header = TRUE)
x <- as.matrix(data[, 2:8])
y <- data$Work
kappa <- y-1/2

# Prior of Beta
tau_0 <- 3
mu_0 <- rep(0, 7)
sigma2_0 = tau_0^2 * diag(7)

# Gibbs sampling setup (Burn in period of 50)
n_sample <- 1050
gibbs_omega <- matrix(nrow = n_sample, ncol=132)
gibbs_beta <- matrix(nrow = n_sample+1, ncol=7)

# Initial sample of beta
set.seed(13)
gibbs_beta[1,] <- rmvnorm(1, mean = mu_0, sigma = sigma2_0)

# Sampling with Gibbs
for (iter in 1:n_sample){

  # Argument for Polya-gamma sampler
  z <- x %*% gibbs_beta[iter, ]

  # Sampling omega with Polya-gamma sampler
  for(obs_index in 1:length(z)){
    gibbs_omega[iter, obs_index] <- rpg(num = 1, h = 1, z = z[obs_index])
  }

  # Sampling beta
  omega <- diag(132) * gibbs_omega[iter, ]
```

```
  v_omega <- solve(t(x) %*% omega %*% x + solve(sigma2_0))
  m_omega <- v_omega %*% (t(x) %*% kappa + solve(sigma2_0) %*% mu_0)
  gibbs_beta[iter + 1, ] <- rmvnorm(n = 1, mean = m_omega, sigma = v_omega)

}

# Remove last beta value (since 1 value was initialized before algorithm)
gibbs_beta <- gibbs_beta[-1, ]

# Remove burn-in samples
gibbs_beta2 <- gibbs_beta[-c(1:50), ]
gibbs_omega2 <- gibbs_omega[-c(1:50), ]
```

Before calculating the inefficiency factors (IFs), the burn-in sampled draws were removed.

The inefficiency factors are calculated with the following formula:

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

The inefficiency factor for each β parameters are as follows.

```
# Removed burn in period observations to calculate IF
for(i in 1:7){
  IF <- sum(acf(gibbs_beta2[, i], plot=FALSE)$acf)*2 +  1
  cat("The inefficiency factor for beta", i, "is", IF, "\n")
}
```

```
## The inefficiency factor for beta 1 is 3.837507
## The inefficiency factor for beta 2 is 2.786703
## The inefficiency factor for beta 3 is 3.940719
## The inefficiency factor for beta 4 is 4.069517
## The inefficiency factor for beta 5 is 3.704083
## The inefficiency factor for beta 6 is 3.513666
## The inefficiency factor for beta 7 is 3.479237
```

In figure 1, the trajectories of the markov chain for each parameter is presented.

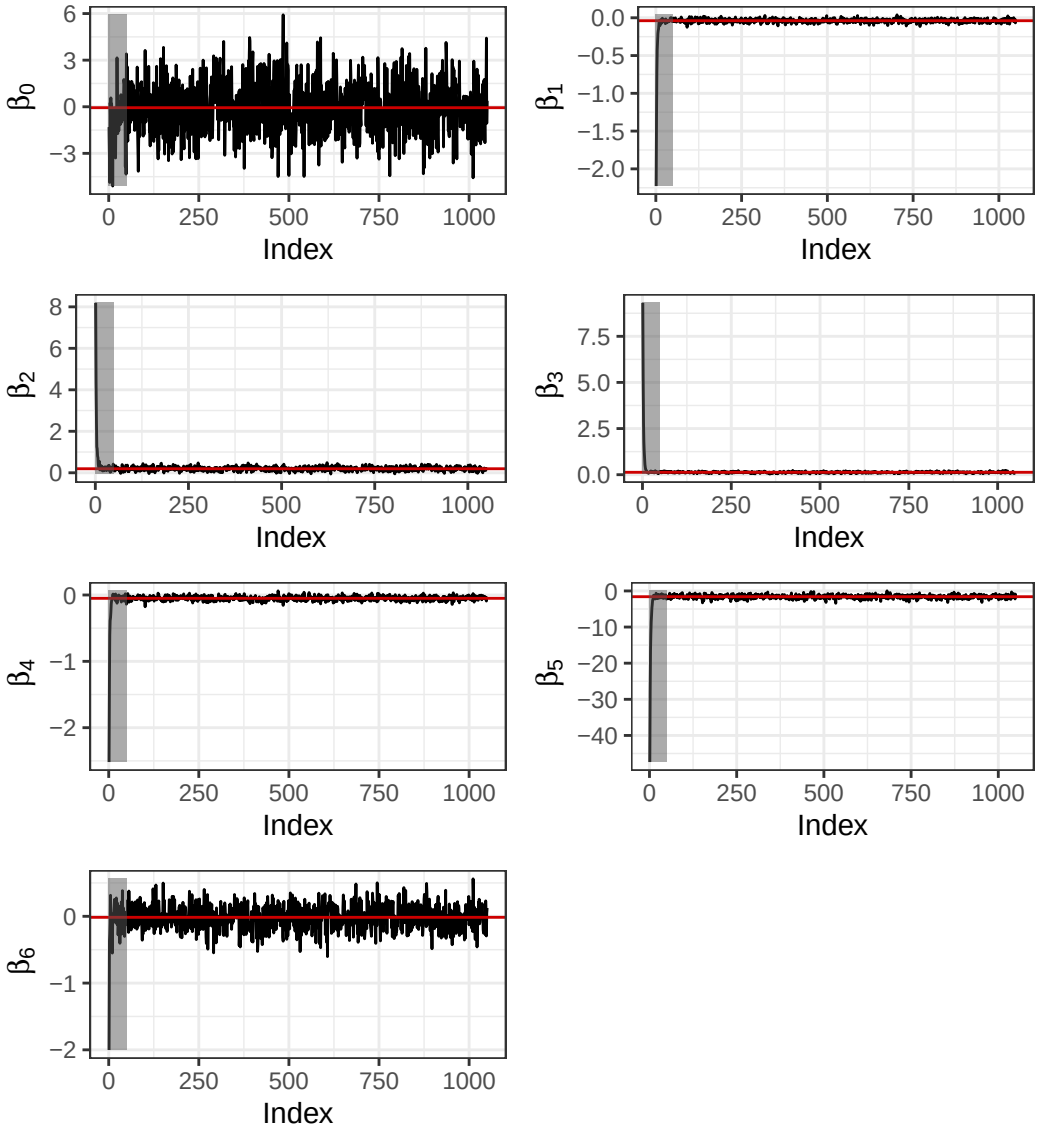

Figure 1: Trajectories of the sampled Markov chains.

In figure 1, the draws in the gray area are for the burn-in samples, where 50 samples were discarded. Examining the chains after the burn-in period, the Gibbs samples appears to have converged.

To double check if we have implemented the algorithm correctly, we compared the result with the function glm since for Gibbs draws we have

$$\bar{\theta} = \frac{1}{N}\sum_{t=1}^{N}\theta^{(t)} \rightarrow E(\theta)$$

```
# 7 different expected values of beta
exp_beta <- c()
for(i in 1:7){
  exp_beta[i] <- sum(gibbs_beta2[, i]) / length(gibbs_beta2[, i])
}

# GLM
model <- glm(Work ~ ., family="binomial", data = data[,-2])
model <- model$coefficients
table_data <- data.frame(Glm = model, Gibbs = exp_beta)
kable(table_data, digits = 3, caption ="Sanity check of parameter estimation from Gibbs compared to glm.
```

Table 1: Sanity check of parameter estimation from Gibbs compared to glm.

	Glm	Gibbs
(Intercept)	0.023	-0.060
HusbandInc	-0.038	-0.039
EducYears	0.184	0.193
ExpYears	0.121	0.129
Age	-0.049	-0.050
NSmallChild	-1.565	-1.598
NBigChild	-0.025	-0.015

From table 1, the estimated values from Gibbs are similar to the glm estimation except from a minor difference in the intercept which can be due to randomness.

b)

Question:

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.

Answer:

The predictive posterior distribution that the corresponding woman is working is presented in figure 2.

```
# Used data where burn in period samples are removed
x_obs <- c(1, 22, 12, 7, 38, 1, 0)
probs_obs <- exp(gibbs_beta2 %*% x_obs) / (1 + exp(gibbs_beta2 %*% x_obs))
equal_tail_interval <- quantile(probs_obs, probs=c(0.05, 0.95))

plot_data <- data.frame(prob = probs_obs)
ggplot(plot_data, aes(prob)) +
  geom_histogram(fill="darkorange2", colour="black", bins =50) +
  geom_vline(xintercept = equal_tail_interval, colour="blue2", linewidth=1, linetype="dashed") +
  labs(x="Probability", y="Count") +
  theme_bw()
```

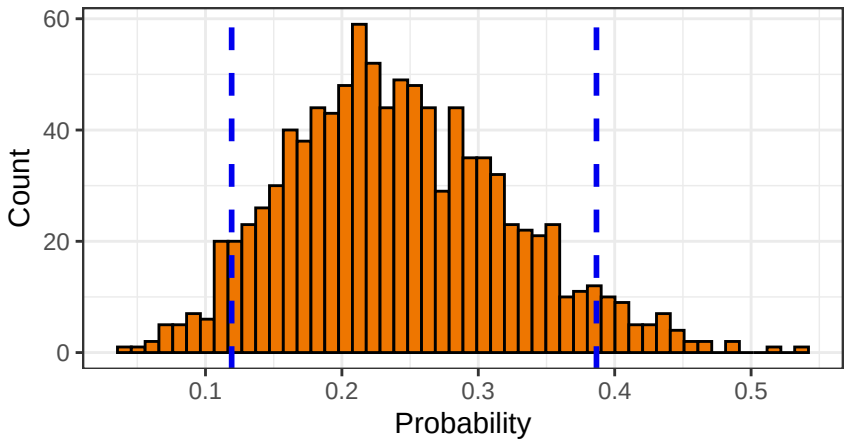

Figure 2: Predictive posterior distribution for a 38-year-old woman, with one child (3 years old), 12 years of education, 7 years of experience, and a husband with an income of 22 with a 90% equal tail credible interval.

```
equal_tail_interval
```

```
##        5%       95%
## 0.1191719 0.3866231
```

In figure 2, the 90% equal tail credible interval is [0.1192, 0.3866].

Question 2:

Metropolis Random Walk for Poisson regression

Consider the following Poisson regression model

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

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 verified 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)

Question:

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 significant?

Answer:

The maximum likelihood estimator of β in the Poission regression model are as follows.

```
# Can skip making factor variables since all factor variables are 0/1
data <- read.table("eBayNumberOfBidderData_2024.dat", header=TRUE)
x <- as.matrix(data[,-1])
y <- data$nBids
names_cov <- names(data[, -1])
model <- glm(nBids ~ ., family="poisson", data = data[, -2])
#model$coefficients
summary.glm(model)
```

```
##
## Call:
## glm(formula = nBids ~ ., family = "poisson", data = data[, -2])
##
## 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
```

From the output, the estimates are found under "Estimate". To evaluate which covariates that are significant, a significance level of 5% with frequentist inference is used. From the frequentist inference the parameters as follows are significant:

- Intercept
- VerifyID
- Sealed
- MajBlem
- LogBook
- MinBidShare

b)

Question:

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

$$\beta|y \sim \mathcal{N}\big(\tilde{\beta}, J_Y^{-1}(\tilde{\beta})\big),$$

where $\tilde{\beta}$ is the posterior mode and J_Y^{-1} is the negative Hessian at the posterior mode. $\tilde{\beta}$ and J_Y^{-1} 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.).

Answer:

The Bayesian analysis was implemented as follows.

```
# Prior
mu_0 <- rep(0, 9)
sigma2_0 <- 100*solve(t(x) %*% x)
beta_0 <- rmvnorm(1, mean = mu_0, sigma = sigma2_0)

LogPostPoisson <- function(betas,y,x,mu,Sigma){
  SmallVal <- .Machine$double.xmin
  logLik <- sum( (y * betas %*% t(x) + SmallVal) -
                   exp(betas %*% t(x) + SmallVal) -
                   (log(factorial(y))))
  # Prior value
  logPrior <- dmvnorm(betas, mu, Sigma, log=TRUE);
  return(logLik + logPrior)
}


Npar <- dim(x)[2]
initVal <- matrix(0,Npar,1)
logPost = LogPostPoisson

# y = response
# x = covariates
# beta_0 = prior values of beta parameters
# sigma2_0 = prior values of covariance matrix
OptimRes <- optim(initVal,logPost,gr=NULL,y,x,beta_0,sigma2_0,method=c("BFGS"),
                  control=list(fnscale=-1),hessian=TRUE)
```

The estimated modes found by the method is as follows.

```
optim_modes <- t(OptimRes$par)
colnames(optim_modes) <- names_cov
optim_modes
```

```
##         Const PowerSeller   VerifyID   Sealed     Minblem    MajBlem    LargNeg
## [1,] 1.073483 -0.03520322 -0.4617468 0.461176 -0.06708607 -0.2285222 0.05662916
##         LogBook MinBidShare
## [1,] -0.0842466   -1.829378
```

```r
sigma <- -solve(OptimRes$hessian)
```

Comparing this output with the output from glm model in Question 2a, the estimates are similar.

c)

Question:

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

Answer:

The implementation of the Metropolis algorithm is as follows.

```r
RWMSampler <- function(n_sample, theta, logPostFun, y, x, beta_0, sigma2_0, c, sigma){

  # Setup result
  theta_sample <- matrix(nrow = n_sample, ncol = length(theta))
  acceptance <- 0

  # Metropolis Algorithm
  for(iter in 1:n_sample){

    # Sample from proposal distribution
    proposal <- rmvnorm(n = 1, mean = theta, sigma = c*sigma)

    # if value is 0.99% we should accept 99% of time. So > u is correct?
    prop_value <- logPostFun(proposal, y, x, beta_0, sigma2_0)
    old_value <- logPostFun(theta, y, x, beta_0, sigma2_0)
    u <- runif(1, 0, 1)
    # Check if we accept the new proposed value
    if(exp(prop_value - old_value) > u){
      theta_sample[iter, ] <- proposal
      theta <- proposal
      acceptance <- acceptance + 1
    } else {
```

```
      theta_sample[iter, ] <- theta
    }
  }

  cat("Acceptance rate:", round(acceptance/n_sample, 4))
  return(theta_sample)
}

n_draws <- 4500
samples <- matrix(nrow = n_draws, ncol = 9)
theta <- rep(0, 9)
c <- 0.5
sigma <- -solve(OptimRes$hessian)
proposal <- rmvnorm(n = 1, mean = theta, sigma = c*sigma)
# mu is beta prior and sigma is covariance prior to get prior probability in Bayes.
MH_sample <- RWMSampler(n_sample = n_draws, theta, logPostFun=LogPostPoisson, y, x, beta_0, sigma2_0, c,
```

```
## Acceptance rate: 0.3356
```

We sampled 4000 values for each β parameter with a burn-in period of 500. The markov chains are presented in figure 3.

```
## Acceptance rate: 0.182
```

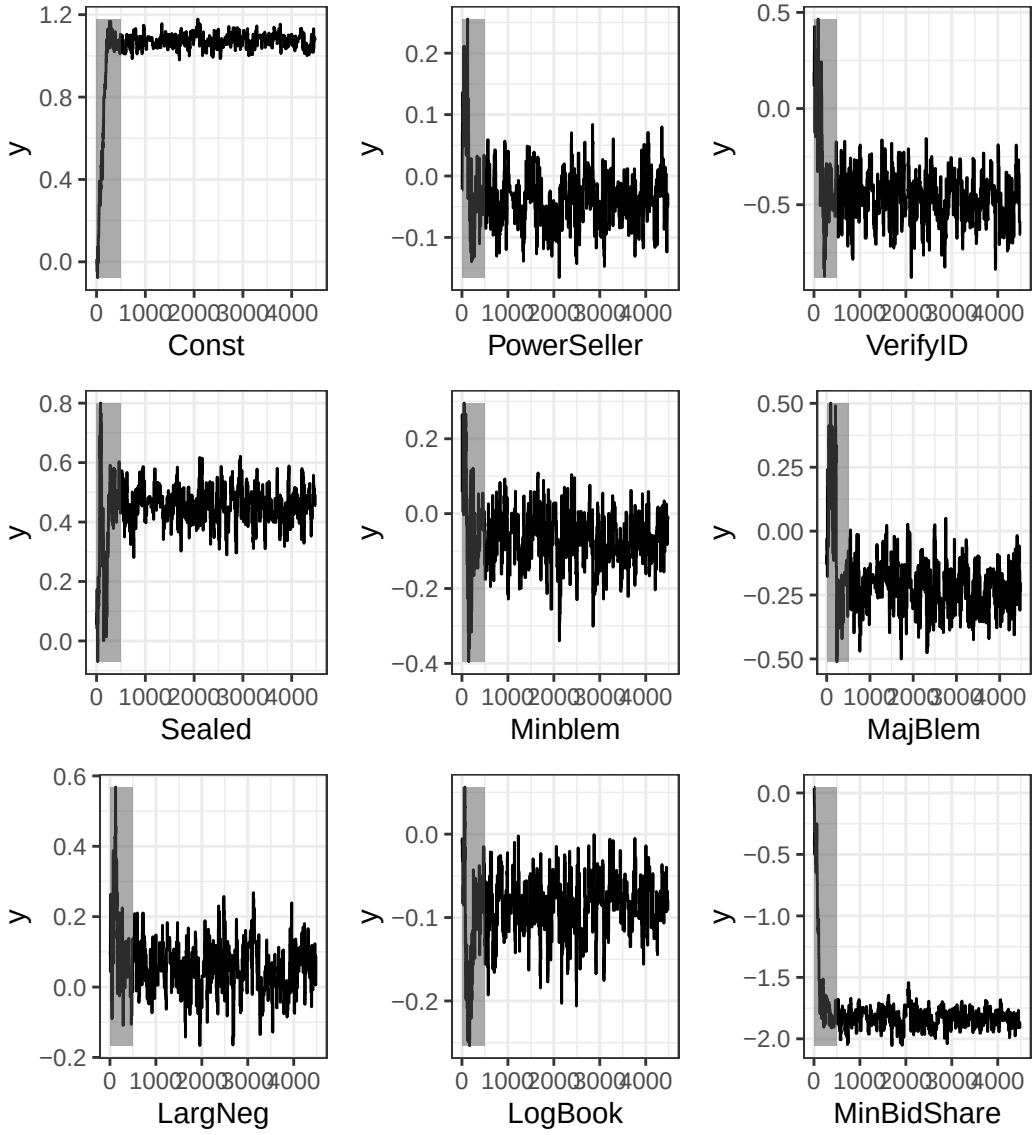

Figure 3: Trajectories of the sampled Markov chains.

In figure 3, the draws in the gray area are for the burn-in samples, where 500 samples were discarded. Examining the chains after the burn-in period, we have convergence for the MCMC.

d)

Question:

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

Answer:

The predictive distribution is presented in figure 4.

```
# Remove burn-in period samples
MH_sample <- MH_sample[-c(1:500), ]
x_obs <- c(1, 1, 0, 1, 0, 1, 0, 1.2, 0.8)

nrBidders <- c()
for(i in 1:nrow(MH_sample)) {
  nrBidders[i] <- rpois(1, exp(MH_sample[i, ] %*% c(1, 1, 0, 1, 0, 1 , 0, 1.2, 0.8)) )
}
```

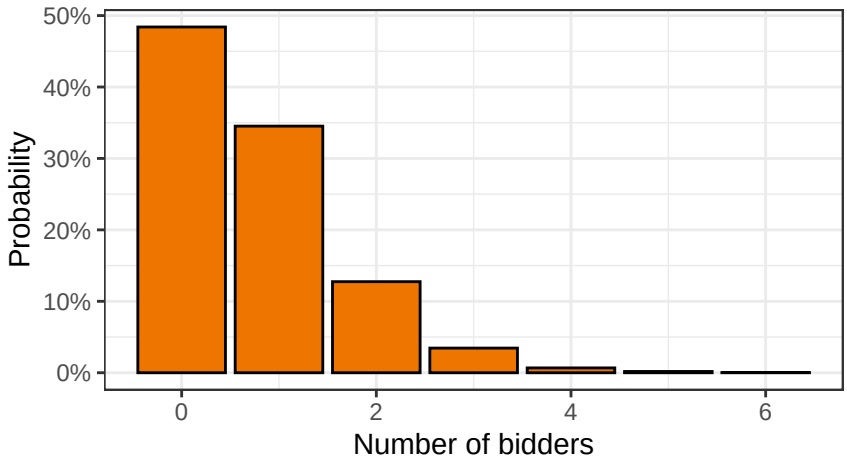

Figure 4: Predictive posterior distribution of the probability if no bidders for the new auction.

In figure 4, the probability of no bidders is 48.4%.

Question 3:

a)

Question:

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

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

for given values of μ, ϕ and σ^2. Start the process at $x_1 = \mu$ 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 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 $x_{1:T}$?

Answer:

The function was implemented with the code as follows.

```
# 3a                                                                    ####
sim_ar1 <- function(N, mu, sd, phi ){
  x <- rep(mu, N)
  for(n in 2:N){
    x[n] <- rnorm(1, mu + phi * (x[n-1] - mu), sd)
  }
  return(x)
}
```

We tried different values of ϕ between -1 and 1, where ϕ changes how the mean of the new draw is dependent on the last draw.

With positive values, the mean for the next value is dependent on where the previous draw was, this is presented in figure 5.

The pattern is stronger for larger positive values of ϕ, the sampled values are also larger when ϕ increases and is presented in figure 6.

For large negative values, the mean will instead be close to the negative difference between the previous draw (y_{t-1}) and the mean (μ), which gives us an oscillating pattern around the mean and is presented in figure 7.

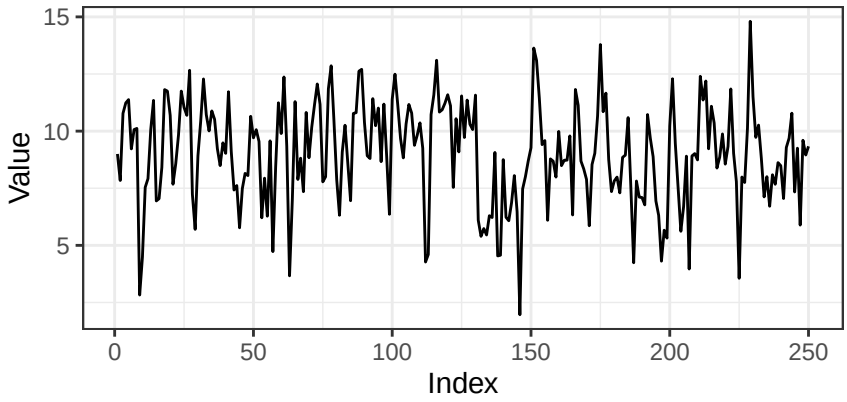

Figure 5: Simulated data with $\phi = 0.5$

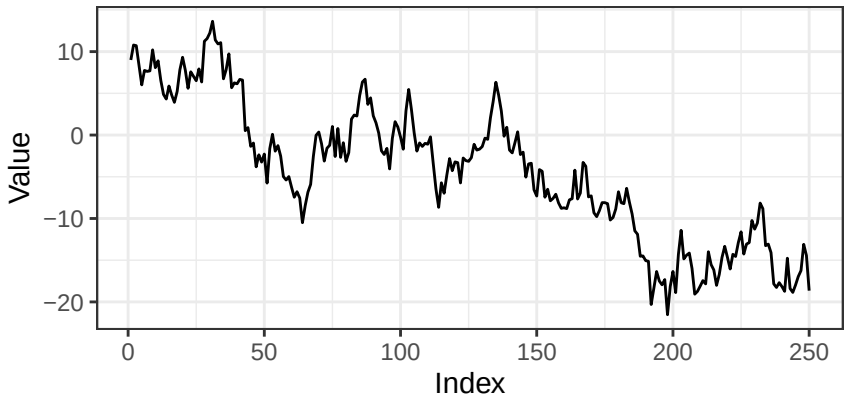

Figure 6: Simulated data with $\phi = 0.99$

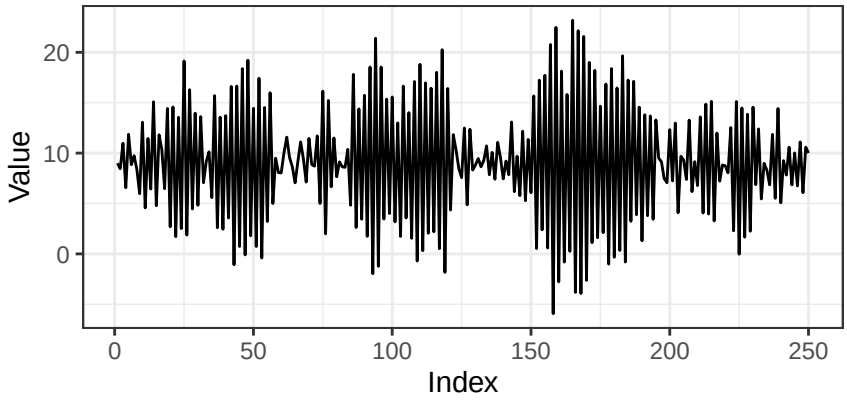

Figure 7: Simulated data with $\phi = -0.99$

b)

Question:

Use your function from a) to simulate two AR(1)-processes, $x_{1:T}$ with $= 0.3$ 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.

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?

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

Answer:

The two different AR(1)-processes were implemented as follows.

```
StanModel = '
data {
  int<lower=0> N;
  vector[N] x;
}
parameters {
  real mu;
  real<lower=-1, upper=1> phi;
  real<lower=0> sigma2;
}
model {
  mu ~ normal(0, 100);
  phi ~ uniform(-1, 1);
  sigma2 ~ scaled_inv_chi_square(1,2);
  for (n in 2:N)
    x[n] ~ normal(mu + phi * (x[n-1]-mu), sqrt(sigma2));
}'
N <- 250
mu <- 9
sd <- 2
set.seed(13)
x <- sim_ar1(N, mu, sd, phi=0.3)
y <- sim_ar1(N, mu, sd, phi=0.97)
data <- list(N = N, mu = mu, x = x, sigma2 = 4)
warmup <- 1000
niter <- 2000
fit_x <- stan(model_code=StanModel,data=data, warmup=warmup,iter=niter,chains=4, refresh=0)
data <- list(N = N, mu = mu, x = y, sigma2 = 4)
fit_y <- stan(model_code=StanModel,data=data, warmup=warmup,iter=niter,chains=4, refresh=0)
```

For AR(1) process with $\phi = 0.3$ we have

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

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##            mean se_mean    sd     2.5%      25%      50%      75%    97.5%
## mu        8.792   0.003 0.205    8.373    8.655    8.794    8.930    9.190
## phi       0.372   0.001 0.060    0.255    0.332    0.372    0.413    0.489
## sigma2    4.194   0.006 0.378    3.513    3.935    4.169    4.445    4.996
## lp__   -304.203   0.029 1.214 -307.353 -304.757 -303.881 -303.323 -302.813
##        n_eff  Rhat
## mu      3710 0.999
## phi     3035 1.002
## sigma2  3849 1.001
## lp__    1798 1.002
##
## Samples were drawn using NUTS(diag_e) at Fri May 10 12:23:59 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).
```

From the output, the 95% credible interval is found by using the value from 2.5% and 97.5% from the output. Number of effective draws are found under n_eff.

For the parameter μ we have 3710 effective draws and the 95% credible interval [8.373, 9.190], which covers the true value 9. Similarly for ϕ and σ the effective draws are over 3000 for each parameters, and 95% credible interval covers the true values of the parameters.

The MCMC draws are presented in figure 8.

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

# Do automatic traceplots of all chains
traceplot(fit_x, nrow=3)
```

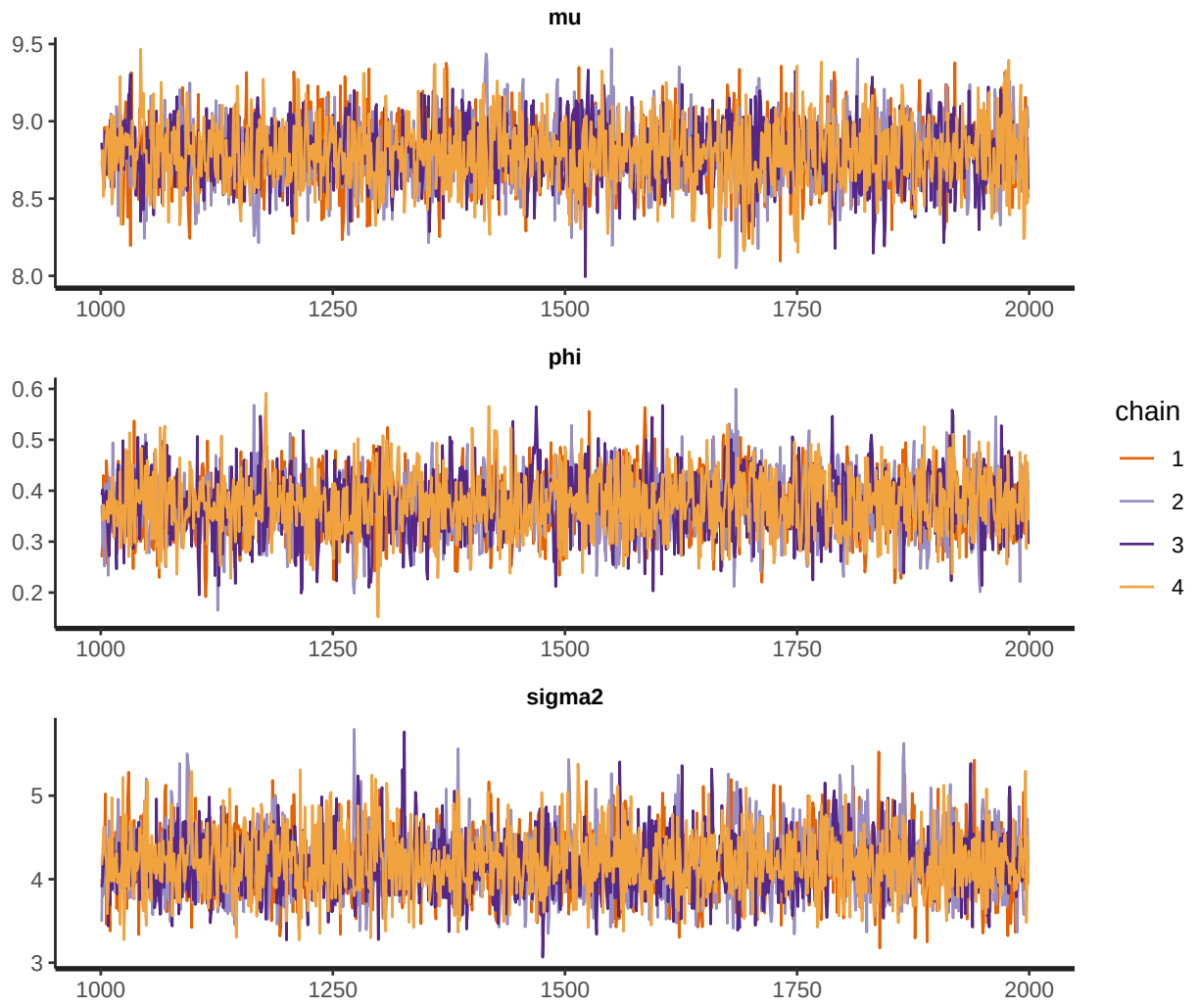

Figure 8: Traceplots for model with $\phi = 0.3$

From figure 8, the four different chain converge and evaluate the posterior at the same area for each parameter.

The bivariate plot between the parameters μ and ϕ are presented in figure 9.

```
# Bivariate posterior plots
pairs(fit_x, pars = c("mu", "phi"))
```

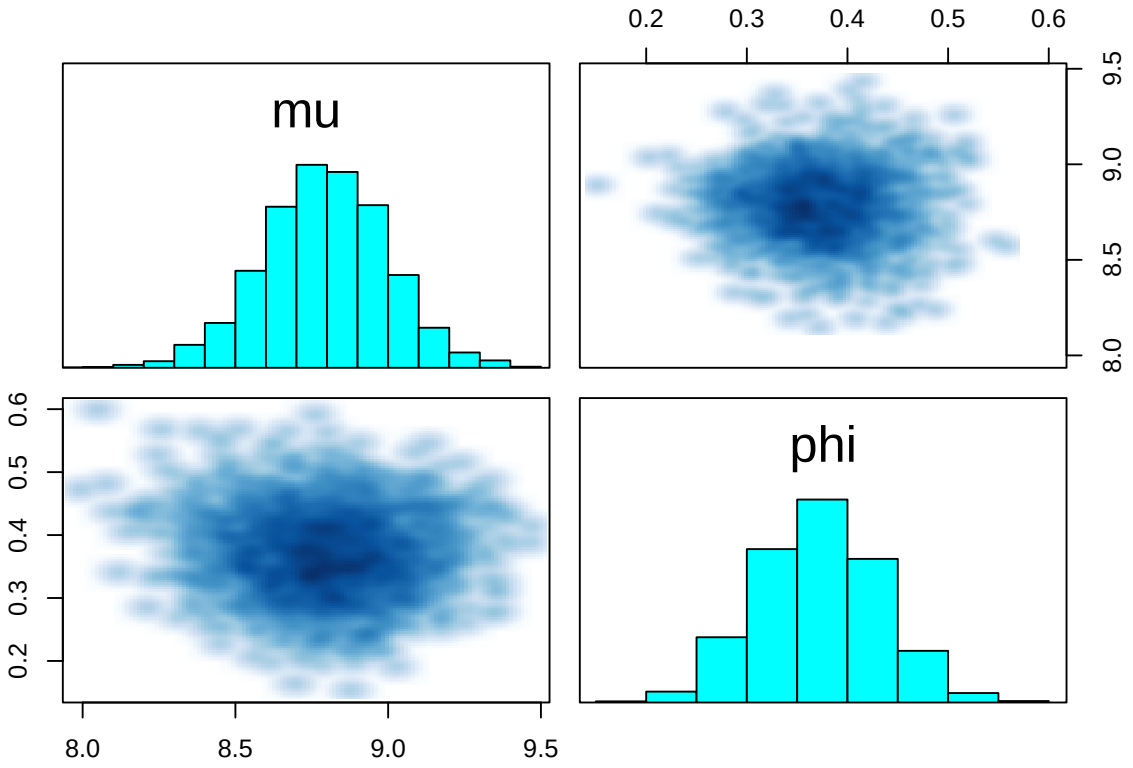

Figure 9: Bivariate posterior plots for model with $\phi = 0.3$

In figure 9, the distribution of each parameter looks normal distributed. From the bivariate plot, the parameters appears to be multivariate normal distributed with no correlation between the parameters.

For AR(1) process with $\phi = 0.97$ we have

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

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##            mean se_mean     sd     2.5%      25%      50%      75%    97.5%
## mu       13.541   1.987 34.246  -53.325    9.623   14.126   18.084   76.004
## phi       0.976   0.001  0.017    0.940    0.963    0.977    0.991    1.000
## sigma2    4.600   0.017  0.416    3.832    4.313    4.570    4.867    5.452
## lp__   -319.073   0.144  2.256 -324.797 -320.333 -318.409 -317.332 -316.486
##        n_eff  Rhat
## mu       297 1.007
## phi      334 1.014
## sigma2   614 1.003
## lp__     244 1.023
##
## Samples were drawn using NUTS(diag_e) at Fri May 10 12:24:03 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).
```

From the output, we can see that number of effective draws for the parameters μ and ϕ are 297 and 334 respectively, meaning we did not get a good sampling for these parameters. Both of the credible interval covers the true value of the parameters but for μ the credible interval is really large (compared to previous model). For the parameter σ the 95% credible interval covers the true value with 614 effective draws.

The trace plots for the parameters are presented in figure 10.

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

# Do automatic traceplots of all chains
traceplot(fit_y, nrow=3)
```

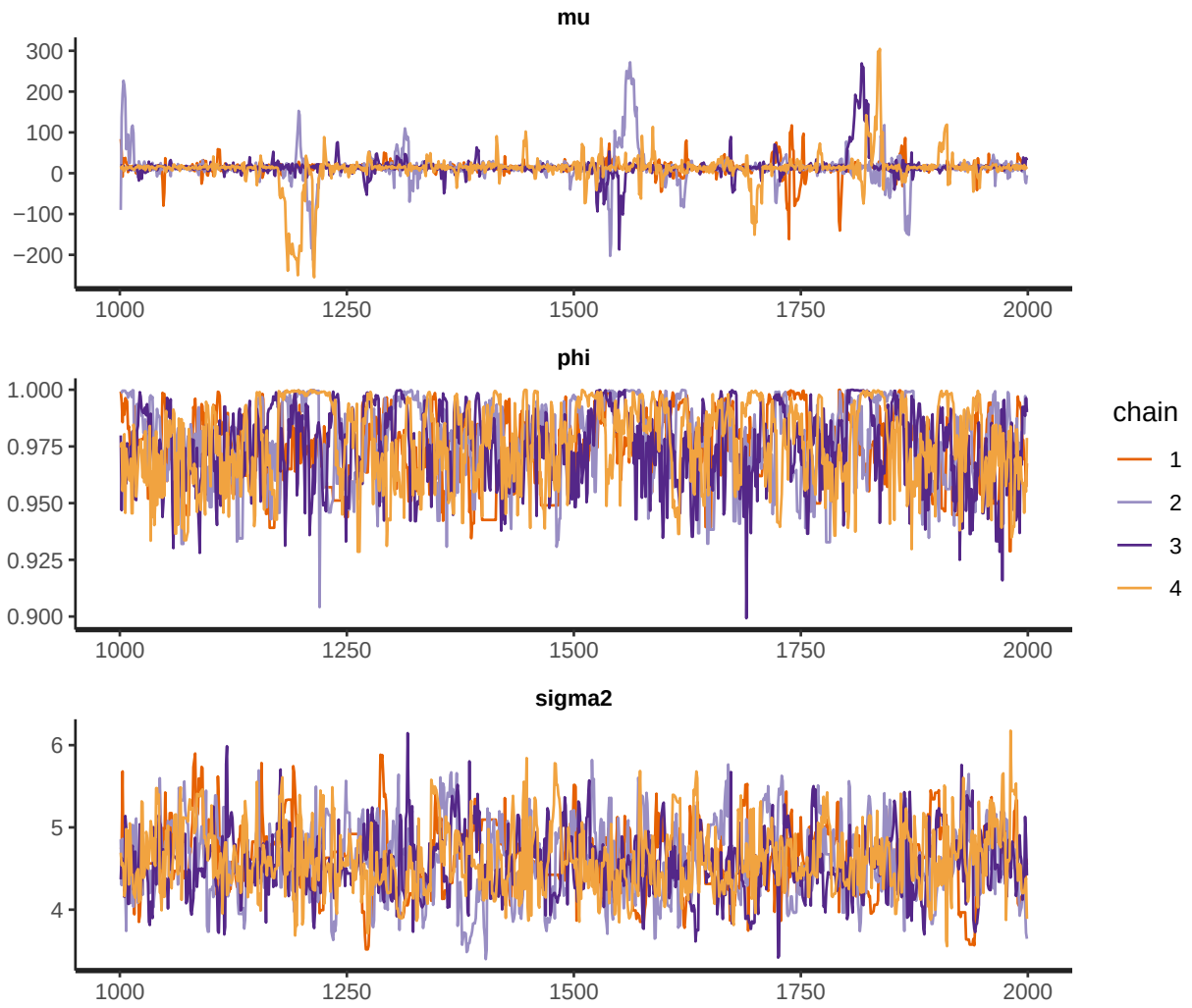

Figure 10: Traceplots for model with $\phi = 0.97$

In figure 10, we can see that the chains for the parameters μ, ϕ, and σ evaluate at different values between the chains.

The bivariate plot for the parameter μ and ϕ are presented in figure 11.

```
# Do automatic traceplots of all chains
pairs(fit_y, pars = c("mu", "phi"))
```

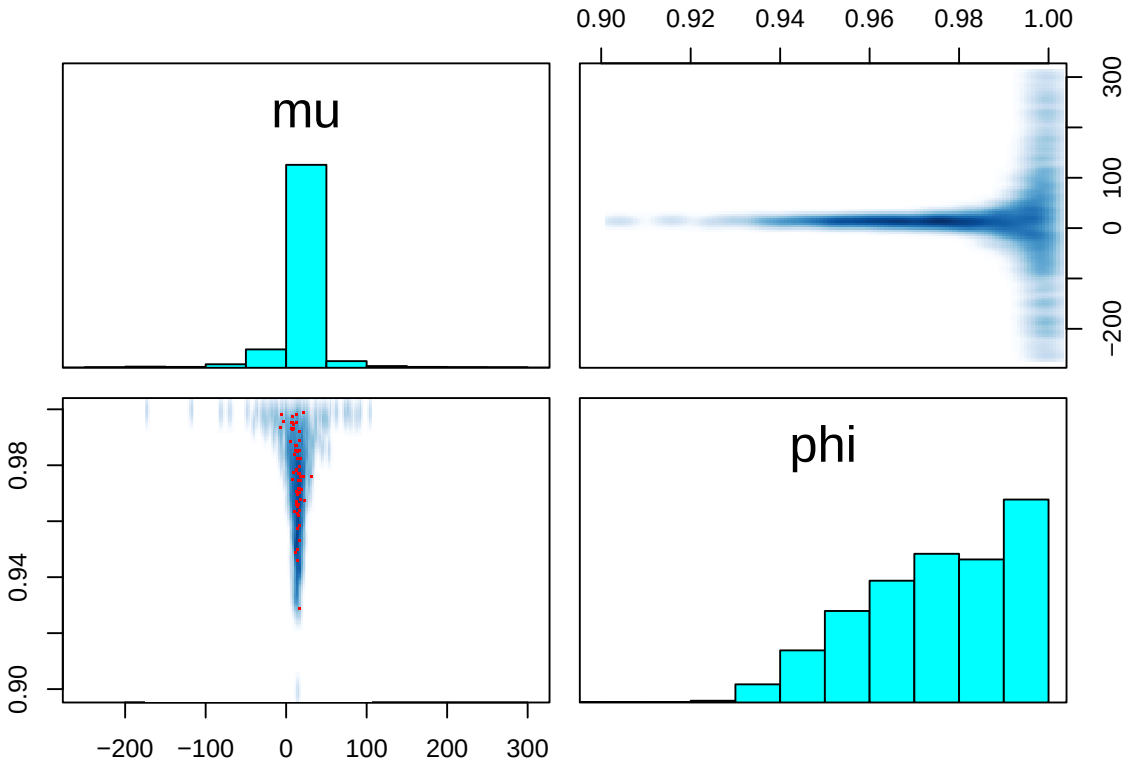

Figure 11: Traceplots for model with $\phi = 0.3$

In figure 11, there appears to be some dependencies between the parameters. The values of ϕ was between 0.98 and 1 for μ lower than -20 and above 40.

Appendix

The code used in this laboration report are summarised in the code as follows:

```
library(ggplot2)
library(BayesLogit)
library(mvtnorm)
library(gridExtra)
library(rstan)
library(knitr)

knitr::opts_chunk$set(
  echo = TRUE,
  fig.width = 4.5,
  fig.height = 2.4)
data <- read.table("WomenAtWork.dat", header = TRUE)
x <- as.matrix(data[, 2:8])
y <- data$Work
kappa <- y-1/2

# Prior of Beta
tau_0 <- 3
mu_0 <- rep(0, 7)
sigma2_0 = tau_0^2 * diag(7)

# Gibbs sampling setup (Burn in period of 50)
n_sample <- 1050
gibbs_omega <- matrix(nrow = n_sample, ncol=132)
gibbs_beta <- matrix(nrow = n_sample+1, ncol=7)

# Initial sample of beta
set.seed(13)
gibbs_beta[1,] <- rmvnorm(1, mean = mu_0, sigma = sigma2_0)

# Sampling with Gibbs
for (iter in 1:n_sample){

  # Argument for Polya-gamma sampler
  z <- x %*% gibbs_beta[iter, ]

  # Sampling omega with Polya-gamma sampler
  for(obs_index in 1:length(z)){
    gibbs_omega[iter, obs_index] <- rpg(num = 1, h = 1, z = z[obs_index])
  }

  # Sampling beta
  omega <- diag(132) * gibbs_omega[iter, ]
  v_omega <- solve(t(x) %*% omega %*% x + solve(sigma2_0))
  m_omega <- v_omega %*% (t(x) %*% kappa + solve(sigma2_0) %*% mu_0)
```

```
  gibbs_beta[iter + 1, ] <- rmvnorm(n = 1, mean = m_omega, sigma = v_omega)

}

# Remove last beta value (since 1 value was initialized before algorithm)
gibbs_beta <- gibbs_beta[-1, ]

# Remove burn-in samples
gibbs_beta2 <- gibbs_beta[-c(1:50), ]
gibbs_omega2 <- gibbs_omega[-c(1:50), ]
# Removed burn in period observations to calculate IF
for(i in 1:7){
  IF <- sum(acf(gibbs_beta2[, i], plot=FALSE)$acf)*2 +  1
  cat("The inefficiency factor for beta", i, "is", IF, "\n")
}
# 7 different expected values of beta
exp_beta <- c()
for(i in 1:7){
  exp_beta[i] <- sum(gibbs_beta2[, i]) / length(gibbs_beta2[, i])
}

# Plot trajectories?
plot_data <- data.frame(x=1:1050, y = gibbs_beta)

p1 <-
  ggplot(plot_data, aes(x=x)) +
  geom_line(aes(y=y.1)) +
  geom_hline(yintercept = exp_beta[1], col="red3") +
  theme_bw() +
  annotate("rect",alpha = .5,
           xmin = 0, xmax = 50,
           ymin = min(plot_data$y.1), ymax = max(plot_data$y.1)) +
  labs(x = "Index", y = expression(~beta[0]))

p2 <-
ggplot(plot_data, aes(x=x)) +
  geom_line(aes(y=y.2)) +
  geom_hline(yintercept = exp_beta[2], col="red3") +
  theme_bw() +
  annotate("rect",alpha = .5,
           xmin = 0, xmax = 50,
           ymin = min(plot_data$y.2), ymax = max(plot_data$y.2)) +
  labs(x = "Index", y = expression(~beta[1]))

p3 <-
ggplot(plot_data, aes(x=x)) +
  geom_line(aes(y=y.3)) +
  geom_hline(yintercept = exp_beta[3], col="red3") +
  theme_bw() +
```

```
  annotate("rect",alpha = .5,
           xmin = 0, xmax = 50,
           ymin = min(plot_data$y.3), ymax = max(plot_data$y.3)) +
  labs(x = "Index", y = expression(~beta[2]))


p4 <-
ggplot(plot_data, aes(x=x)) +
  geom_line(aes(y=y.4)) +
  geom_hline(yintercept = exp_beta[4], col="red3") +
  theme_bw() +
  annotate("rect",alpha = .5,
           xmin = 0, xmax = 50,
           ymin = min(plot_data$y.4), ymax = max(plot_data$y.4)) +
  labs(x = "Index", y = expression(~beta[3]))

p5 <-
ggplot(plot_data, aes(x=x)) +
  geom_line(aes(y=y.5)) +
  geom_hline(yintercept = exp_beta[5], col="red3") +
  theme_bw() +
  annotate("rect",alpha = .5,
           xmin = 0, xmax = 50,
           ymin = min(plot_data$y.5), ymax = max(plot_data$y.5)) +
  labs(x = "Index", y = expression(~beta[4]))

p6 <-
ggplot(plot_data, aes(x=x)) +
  geom_line(aes(y=y.6)) +
  geom_hline(yintercept = exp_beta[6], col="red3") +
  theme_bw() +
  annotate("rect",alpha = .5,
           xmin = 0, xmax = 50,
           ymin = min(plot_data$y.6), ymax = max(plot_data$y.6)) +
  labs(x = "Index", y = expression(~beta[5]))

p7 <-
ggplot(plot_data, aes(x=x)) +
  geom_line(aes(y=y.7)) +
  geom_hline(yintercept = exp_beta[7], col="red3") +
  theme_bw() +
  annotate("rect",alpha = .5,
           xmin = 0, xmax = 50,
           ymin = min(plot_data$y.7), ymax = max(plot_data$y.7)) +
  labs(x = "Index", y = expression(~beta[6]))

grid.arrange(p1, p2, p3,
             p4, p5, p6,
             p7, nrow = 4)
```

```r
# 7 different expected values of beta
exp_beta <- c()
for(i in 1:7){
  exp_beta[i] <- sum(gibbs_beta2[, i]) / length(gibbs_beta2[, i])
}

# GLM
model <- glm(Work ~ ., family="binomial", data = data[,-2])
model <- model$coefficients
table_data <- data.frame(Glm = model, Gibbs = exp_beta)
kable(table_data, digits = 3, caption ="Sanity check of parameter estimation from Gibbs compared to glm.
# Used data where burn in period samples are removed
x_obs <- c(1, 22, 12, 7, 38, 1, 0)
probs_obs <- exp(gibbs_beta2 %*% x_obs) / (1 + exp(gibbs_beta2 %*% x_obs))
equal_tail_interval <- quantile(probs_obs, probs=c(0.05, 0.95))

plot_data <- data.frame(prob = probs_obs)
ggplot(plot_data, aes(prob)) +
  geom_histogram(fill="darkorange2", colour="black", bins =50) +
  geom_vline(xintercept = equal_tail_interval, colour="blue2", linewidth=1, linetype="dashed") +
  labs(x="Probability", y="Count") +
  theme_bw()

equal_tail_interval

# Can skip making factor variables since all factor variables are 0/1
data <- read.table("eBayNumberOfBidderData_2024.dat", header=TRUE)
x <- as.matrix(data[,-1])
y <- data$nBids
names_cov <- names(data[, -1])
model <- glm(nBids ~ ., family="poisson", data = data[, -2])
#model$coefficients
summary.glm(model)
# Prior
mu_0 <- rep(0, 9)
sigma2_0 <- 100*solve(t(x) %*% x)
beta_0 <- rmvnorm(1, mean = mu_0, sigma = sigma2_0)

LogPostPoisson <- function(betas,y,x,mu,Sigma){
  SmallVal <- .Machine$double.xmin
  logLik <- sum( (y * betas %*% t(x) + SmallVal) -
                   exp(betas %*% t(x) + SmallVal) -
                   (log(factorial(y))))
  # Prior value
  logPrior <- dmvnorm(betas, mu, Sigma, log=TRUE);
  return(logLik + logPrior)
}
```

```r
Npar <- dim(x)[2]
initVal <- matrix(0,Npar,1)
logPost = LogPostPoisson

# y = response
# x = covariates
# beta_0 = prior values of beta parameters
# sigma2_0 = prior values of covariance matrix
OptimRes <- optim(initVal,logPost,gr=NULL,y,x,beta_0,sigma2_0,method=c("BFGS"),
                  control=list(fnscale=-1),hessian=TRUE)
optim_modes <- t(OptimRes$par)
colnames(optim_modes) <- names_cov
optim_modes
sigma <- -solve(OptimRes$hessian)

RWMSampler <- function(n_sample, theta, logPostFun, y, x, beta_0, sigma2_0, c, sigma){

  # Setup result
  theta_sample <- matrix(nrow = n_sample, ncol = length(theta))
  acceptance <- 0

  # Metropolis Algorithm
  for(iter in 1:n_sample){

    # Sample from proposal distribution
    proposal <- rmvnorm(n = 1, mean = theta, sigma = c*sigma)

    # if value is 0.99% we should accept 99% of time. So > u is correct?
    prop_value <- logPostFun(proposal, y, x, beta_0, sigma2_0)
    old_value <- logPostFun(theta, y, x, beta_0, sigma2_0)
    u <- runif(1, 0, 1)
    # Check if we accept the new proposed value
    if(exp(prop_value - old_value) > u){
      theta_sample[iter, ] <- proposal
      theta <- proposal
      acceptance <- acceptance + 1
    } else {
      theta_sample[iter, ] <- theta
    }
  }

  cat("Acceptance rate:", round(acceptance/n_sample, 4))
  return(theta_sample)
}

n_draws <- 4500
samples <- matrix(nrow = n_draws, ncol = 9)
theta <- rep(0, 9)
```

```r
c <- 0.5
sigma <- -solve(OptimRes$hessian)
proposal <- rmvnorm(n = 1, mean = theta, sigma = c*sigma)
# mu is beta prior and sigma is covariance prior to get prior probability in Bayes.
MH_sample <- RWMSampler(n_sample = n_draws, theta, logPostFun=LogPostPoisson, y, x, beta_0, sigma2_0, c,
n_draws <- 4500
samples <- matrix(nrow = n_draws, ncol = 9)
theta <- rep(0, 9)
c <- 1
sigma <- -solve(OptimRes$hessian)
proposal <- rmvnorm(n = 1, mean = theta, sigma = c*sigma)
# mu is beta prior and sigma is covariance prior to get prior probability in Bayes.
MH_sample <- RWMSampler(n_sample = n_draws, theta, logPostFun=LogPostPoisson, y, x, beta_0, sigma2_0, c,

plots <- list()
for(i in 1:9){
  plot_data <- data.frame(y=MH_sample[, i])
  plots[[i]] <-
    ggplot(plot_data, aes(x=1:dim(plot_data)[1], y=y)) +
    geom_line() +
    theme_bw() +
    labs(x=names_cov[i]) +
    annotate("rect",alpha = .5,
           xmin = 0, xmax = 500,
           ymin = min(plot_data$y), ymax = max(plot_data$y))
}
# We have long burn in period (around 300)
grid.arrange(plots[[1]], plots[[2]], plots[[3]],
             plots[[4]], plots[[5]], plots[[6]],
             plots[[7]], plots[[8]], plots[[9]],
             nrow = 3)
# Remove burn-in period samples
MH_sample <- MH_sample[-c(1:500), ]
x_obs <- c(1, 1, 0, 1, 0, 1, 0, 1.2, 0.8)

nrBidders <- c()
for(i in 1:nrow(MH_sample)) {
  nrBidders[i] <- rpois(1, exp(MH_sample[i, ] %*% c(1, 1, 0, 1, 0, 1 , 0, 1.2, 0.8)) )
}
plot_data <- data.frame(x=nrBidders)
ggplot(plot_data, aes(x)) +
   geom_bar(aes(y = after_stat(count)/sum(after_stat(count))), fill="darkorange2", colour="black") +
         scale_y_continuous(labels=scales::percent) +
  labs(x = "Number of bidders",
       y = "Probability") +
  theme_bw()
# 3a                                                                    ####
sim_ar1 <- function(N, mu, sd, phi ){
  x <- rep(mu, N)
```

```r
  for(n in 2:N){
    x[n] <- rnorm(1, mu + phi * (x[n-1] - mu), sd)
  }
  return(x)
}
N <- 250
mu <- 9
sd <- 2

plot_data <- data.frame(data = sim_ar1(N, mu, sd, phi=0.5))

ggplot(plot_data, aes(x=1:N, y=data)) +
  geom_line() +
  theme_bw() +
  labs(x = "Index", y = "Value")
N <- 250
mu <- 9
sd <- 2

plot_data <- data.frame(data = sim_ar1(N, mu, sd, phi=0.99))

ggplot(plot_data, aes(x=1:N, y=data)) +
  geom_line() +
  theme_bw() +
  labs(x = "Index", y = "Value")
N <- 250
mu <- 9
sd <- 2

plot_data <- data.frame(data = sim_ar1(N, mu, sd, phi=-0.99))

ggplot(plot_data, aes(x=1:N, y=data)) +
  geom_line() +
  theme_bw() +
  labs(x = "Index", y = "Value")
StanModel = '
data {
  int<lower=0> N;
  vector[N] x;
}
parameters {
  real mu;
  real<lower=-1, upper=1> phi;
  real<lower=0> sigma2;
}
model {
  mu ~ normal(0, 100);
  phi ~ uniform(-1, 1);
  sigma2 ~ scaled_inv_chi_square(1,2);
```

```
  for (n in 2:N)
    x[n] ~ normal(mu + phi * (x[n-1]-mu), sqrt(sigma2));
}'
N <- 250
mu <- 9
sd <- 2
set.seed(13)
x <- sim_ar1(N, mu, sd, phi=0.3)
y <- sim_ar1(N, mu, sd, phi=0.97)
data <- list(N = N, mu = mu, x = x, sigma2 = 4)
warmup <- 1000
niter <- 2000
fit_x <- stan(model_code=StanModel,data=data, warmup=warmup,iter=niter,chains=4, refresh=0)
data <- list(N = N, mu = mu, x = y, sigma2 = 4)
fit_y <- stan(model_code=StanModel,data=data, warmup=warmup,iter=niter,chains=4, refresh=0)
# Print the fitted model
print(fit_x,digits_summary=3)
# Extract posterior samples
postDraws <- extract(fit_x)

# Do automatic traceplots of all chains
traceplot(fit_x, nrow=3)
# Extract posterior samples
postDraws <- extract(fit_x)

# Do automatic traceplots of all chains
traceplot(fit_x, nrow=3)
# Bivariate posterior plots
pairs(fit_x, pars = c("mu", "phi"))
# Print the fitted model
print(fit_y,digits_summary=3)
# Extract posterior samples
postDraws <- extract(fit_y)

# Do automatic traceplots of all chains
traceplot(fit_y, nrow=3)
# Extract posterior samples
postDraws <- extract(fit_y)

# Do automatic traceplots of all chains
traceplot(fit_y, nrow=3)
# Do automatic traceplots of all chains
pairs(fit_y, pars = c("mu", "phi"))
```