Bayesian Learning:Computer Lab-3

Mowniesh Asokan(mowas455),Shahrukh Iqbal(shaiq681)

18/05/2021

1.Gibbs Sampler for normal model.

a.

We are provided with text file rainfall.dat dataset that consist of daily records from 1948 to 1983. It is given,

$$\ln y_1 ... lny_n|\sigma^2, \mu \overset{iid}{\sim} \mathcal{N}(\mu, \sigma^2) \tag{1}$$

From(1) we can restate that,

$$[y_1 ....., y_n, mu, \sigma^2] \overset{iid}{\sim} log\mathcal{N}(\mu, \sigma^2 \tag{2}$$

Also,

$$\mu \sim \mathcal{N}(\mu_0, \tau_0^2) \tag{3}$$

$$\sigma^2 \sim Inv - \chi^2(\mu_0, \sigma_0^2) \tag{4}$$

From Lecture-7, we know the full conditional posterior of μ and σ^2 to be,

$$\mu|\sigma^2, y \sim \mathcal{N}(\mu_n, \tau_n^2) \tag{5}$$

$$\sigma^2|\mu, y \sim Inv - \chi^2(\nu_n, \frac{\nu_0\sigma^2 + \sum(logy_i - \mu)^2}{n + \nu_0}) \tag{6}$$

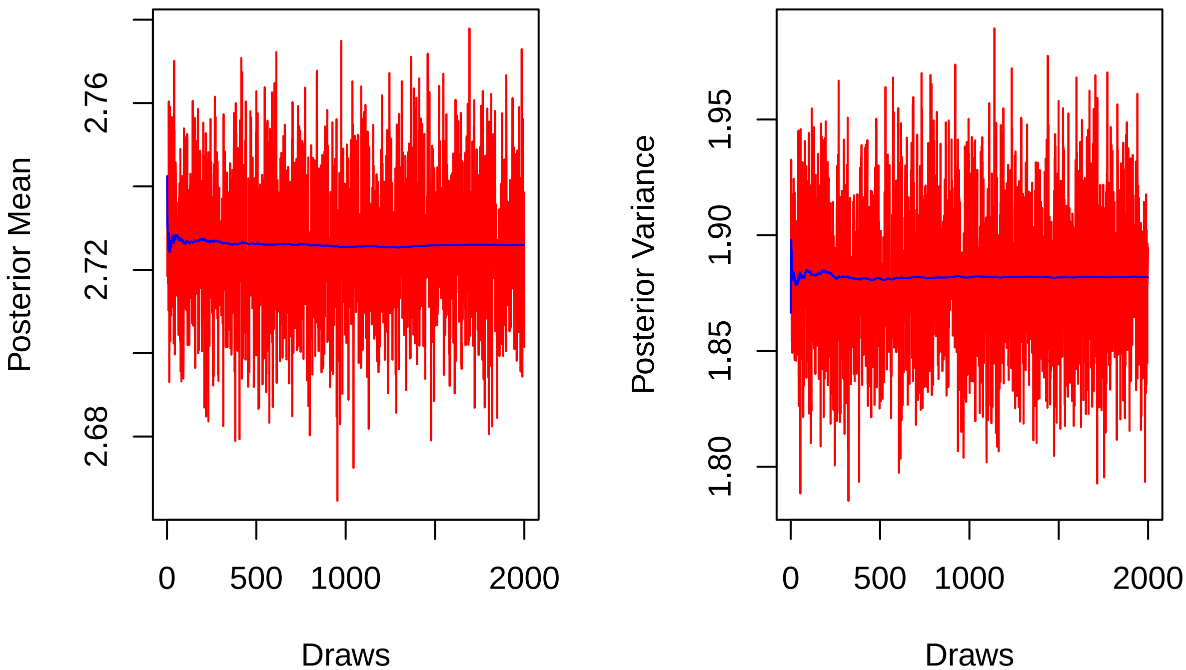
where,

$$\nu_n = \nu_0 + n$$
$$\mu_n = (1 - w_0).\mu_0 + w.log(\bar{y})$$
$$w = \frac{(\frac{n}{\sigma^2})}{(\frac{n}{\sigma^2} + \frac{n}{\tau^2})}$$
$$\tau_n^2 = \frac{1}{(\frac{n}{\sigma^2} + \frac{n}{\tau^2})} \tag{7}$$

We initialize the parameter as,

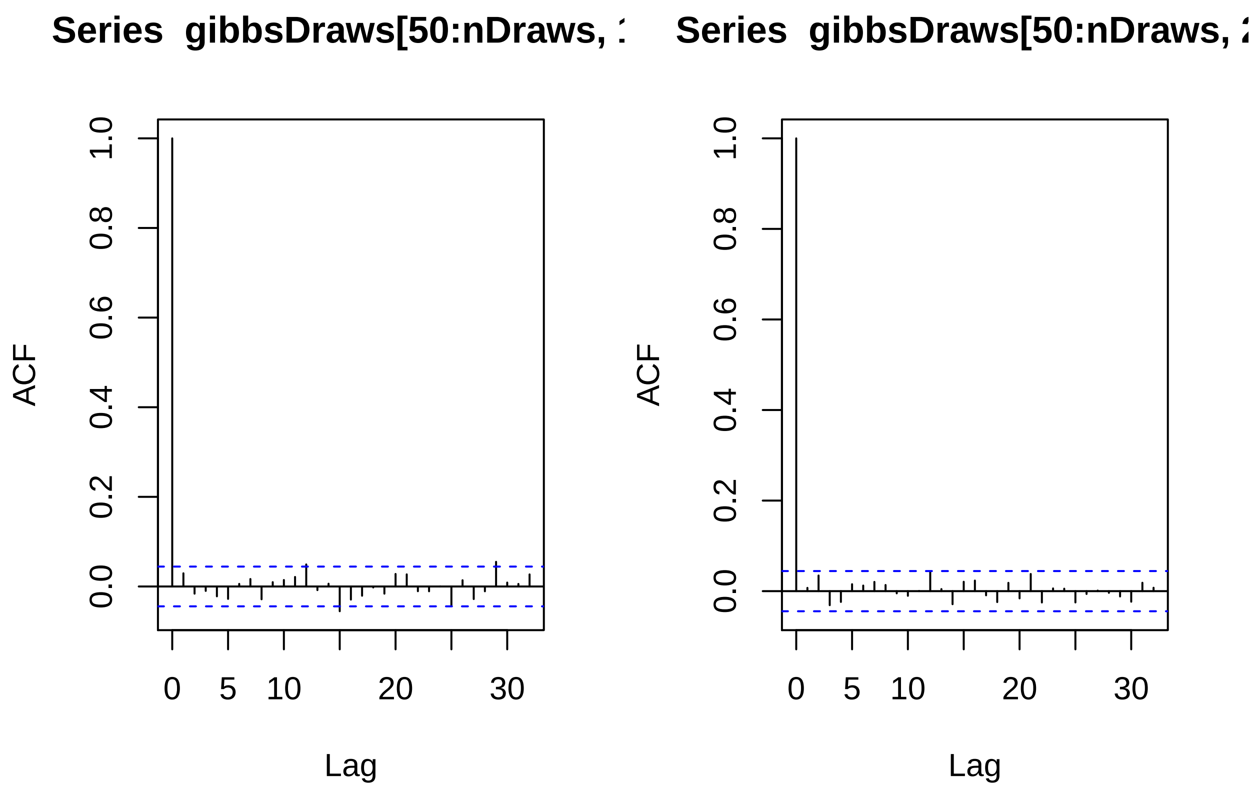
$$\mu_0 = \frac{1}{N}\sum^n logy_i$$
$$\nu_0 = variance(logy_1....logy_n)$$
$$\sigma_0^2 = 1$$

$nDraws = 2000$ Using the initialized parameters we generate a Gibbs Sampling for the posterior of μ and σ^2 respectively.



We evaluate the convergence of the Gibbs Sampling by calculating the Inefficiency factor(IF) and Effective Sample Size. Assuming an asymptotic Relation for the number of Draws i.e.$nDraws = 2000$,we calculate the inefficiency factor and effective sample size by using the equation 8 and 9.

$$InefficiencyFactor(IF) = 1 + 2\sum_{k=1}^{\infty} \rho_k \tag{8}$$



Where "ρ_k" is the autocorelation at $log'k'$ and $nDraws \to \infty$

$$EffectiveSampleSize = \frac{1}{N} * IF \tag{9}$$

For the fixed number of iteration ,an MCMC sample achieves faster convergence ,if the effective sample size is large.

Since,for the both μ and σ^2, we get an effective sample size close to nDraws. We can state analytically that convergence is achieved.

We also plot the trajectories of the realized MCMC samples for μ and σ^2 in figure-1. ν_n usual inspection of figure-1, we can conclude some inferences as stated above i.e the posterior statistic converge to the expected μ and σ^2.

b.Histogram or Kernel Density Estimation of daily Precipitation.

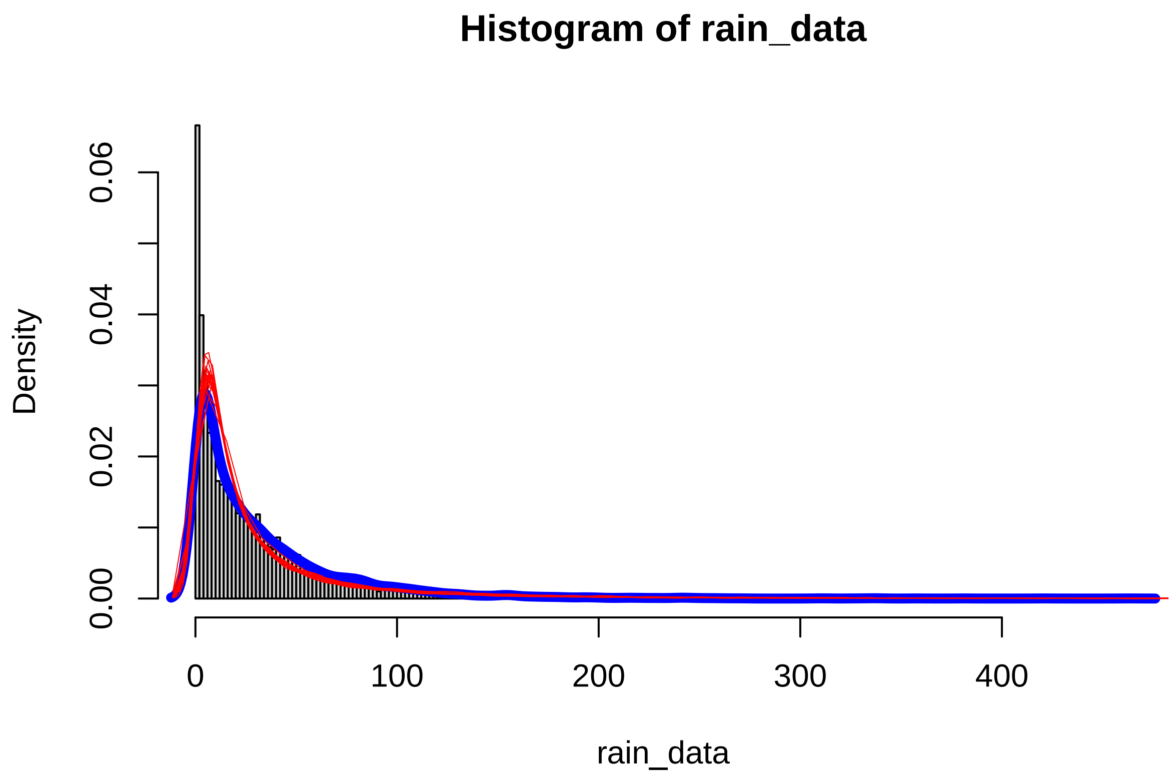
In this Figure-2, we plot the histogram for the Daily precipitaion $y_1..., y_n$ along with its density.

In order to draw $p(\widetilde{y}|y)$ we use the following Monte Carlo approximation for the integral.

$$p(\widetilde{y}|y) = \int_\theta p(\widetilde{y}|\theta)p(\theta|y)d\theta$$

$$= \int_{\mu,\sigma} p(\widetilde{y}|\mu, \sigma^2)p(\mu, \sigma^2|y)d\mu d\sigma^2$$

$$\approx \sum_{i=1}^{M} p(\widetilde{y}|\mu_i, \sigma_i^2)p(\mu_i, \sigma_i^2|y) \tag{10}$$



Using result from equation(2) and (10), we draw $\widetilde{y}$ and plot the same in figure-2 along with y. we observe that predicted $\widetilde{y}$ is able to capture trend, peak and fail of the observed y

2.Metropolis Random Walk for Poisson regression

a.

We implement a maximum likelihood approach to calculate point estimates for Beta Coefficients.From a frequentist perspective,we reject the null hypothesis$J\beta_i = 0$

Thus the Intercept,VerifyID,Scaled,MajBlem,LargBook,and MinBidshare are significant Covariates.

```
## (Intercept) PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##  1.07244206 -0.02054076 -0.39451647  0.44384257 -0.05219829 -0.22087119
##     LargNeg     LogBook MinBidShare
##  0.07067246 -0.12067761 -1.89409664
```

b.

We perform Bayesian analysis of the Poisson Regression,

$$y_i|\beta \sim Pois[exp(X_i^T\beta)], i = 1, 2...n \tag{11}$$

We propose a Zellner's g-prior for β coefficients

$$\beta \sim \mathcal{N}[0, 100(X^TX)^{-1}] \tag{12}$$

where X is $n * p$ covariate matrix.

we assume the posterior density of $J\beta$ to be approximately multivariate normal

$$\beta|y \sim \mathcal{N}(\hat{\beta}, J_y^{-1}(\hat{\beta})) \tag{13}$$

where, $\widetilde{\beta} = PosteriorMode$ and $J_y(\hat{\beta}) - \frac{\partial^2 lnp(\beta|y)}{\partial^2\beta}$

```
##         Const PowerSeller  VerifyID    Sealed     Minblem    MajBlem    LargNeg
## [1,] 1.069841 -0.02051246 -0.393006 0.4435555 -0.05246627 -0.2212384 0.07069683
##         LogBook MinBidShare
## [1,] -0.1202177   -1.891985
```

c.Metropolis function to sample from the posterior of β in the Poisson regression for the eBay dataset.

From Bayes Theorem,

$$p(\beta|y)\alpha p(y|\beta).p(\beta)$$

Thus, we need to estimate the complete data likelihood in order to estimate the posterior $J\beta$ so that we can obtain results of equation.(13) using optimization Techniques.

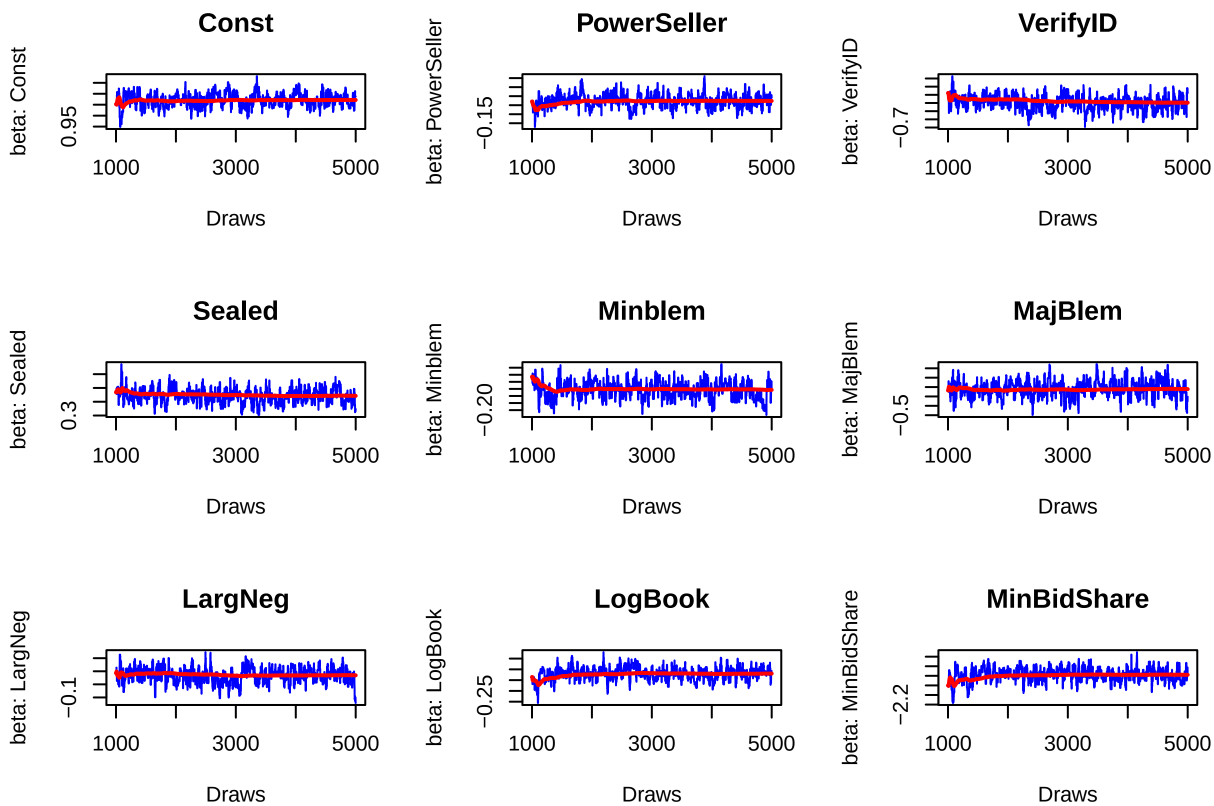
Since $y_i|\beta \sim Pois[exp(X_i^T\beta)]$

$$L(y|\beta) \overset{iid}{\sim} \prod_{i=1}^{n} \frac{(X^T\beta)^{y_i}.e^{-X^T\beta}}{y_i!}$$

$$logL(y|\beta) = \sum^{n}(y_i logX_i^T\beta - X_i^T\beta - logy_i!) \tag{14}$$

```
f.MCMC.MH = function(logPostFunc,nDraws,c,beta_init,X,y,mu,sigma,hessian, warmup=0){
  post_var_mat = - solve(hessian)*c
  beta_matrix = matrix(nrow = nDraws,ncol = length(beta_init))
  beta_matrix[1,] = beta_init
  accept = 0
  for(i in 2:nDraws){
    beta_now = beta_matrix[i-1,] # {beta_{i-1}}
    # Draw candidate
    beta_cand = as.vector(rmvnorm(1,mean = beta_now, sigma = post_var_mat))
    # Calculate Metropolis-Hasting ratio
    # q(.|.)~ Normal. hence it is excluded in ratio calculation due to symmetry
    R = exp(logPostFunc(beta_init=beta_cand,X,y,mu,sigma) - logPostFunc(beta_init=beta_now,X,y,mu,sigma
    #
    alpha = min(1,R)
    # Draw from Uniform distribution
    u = runif(1)
```

```
    if(u <= alpha){
      beta_now = beta_cand
      accept = accept+1
    }
    beta_matrix[i,] = beta_now
  }
  return(list(beta = beta_matrix[warmup:nDraws,], acceptRate = accept/nDraws*100))
}
```



we substitute result of equation.14 to estimate the log posterior as calculating log avoids the overflow problem.

we list the result of $J\hat{\beta}$ in table-1

Figure -2 shows the MCMC convergence $j\beta$ for warm-up period. The converged value approximate towards Beta values listed in Table -1

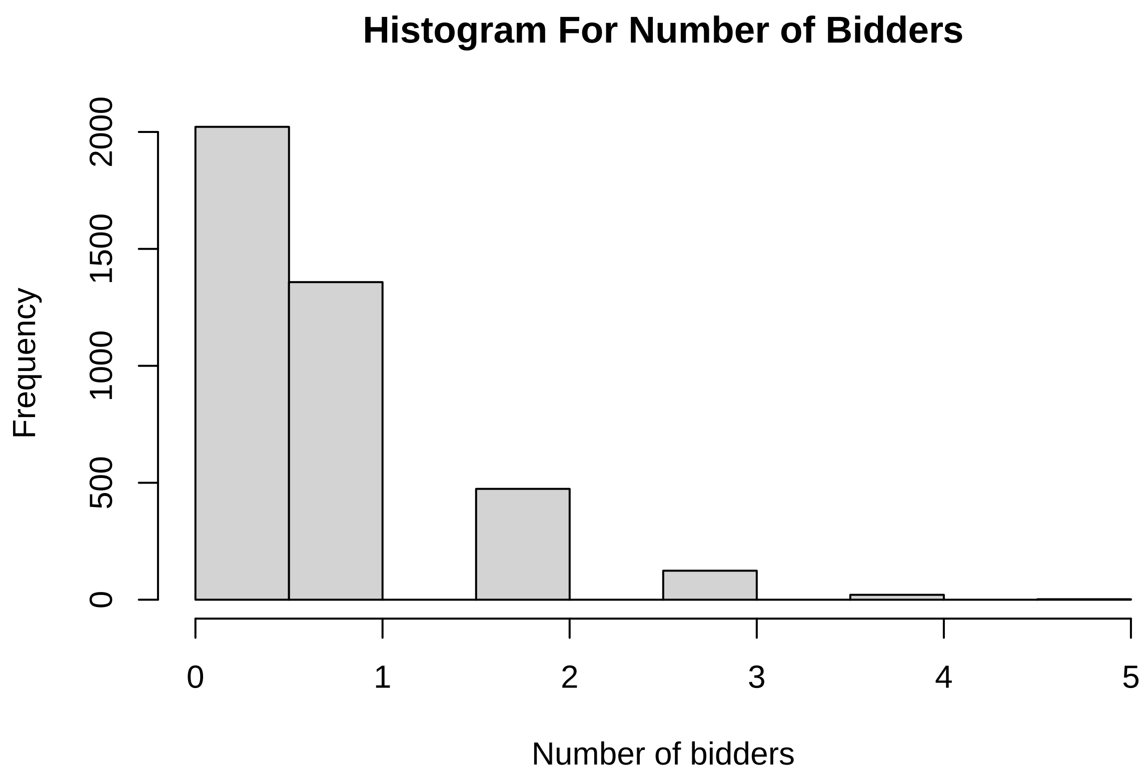
d.

We use samples MCMC draws of β from (c) to simulate from the predictive distribution of nBids, by substituting in Equation.11

We present a histogram of Predictive Distribution in figure-2. It is evident that event $Pr(nBids = 0)$ has the higher probability of occurance.

Using the MCMC results,

$$Pr(nBids = 0) = \frac{1}{N}\sum_{i=1}^{N} I(y_i = 0) \tag{15}$$



We calculate the probability of event $Pr(nBids = 0)$ to be equal to 0.468133

3.Time series models in Stan

a.

We write a function in R to simulate data from an AR(1) process using equation (16)

$$x_t = \mu + \phi(x_{t-1} - \mu) + \epsilon_t, \epsilon_t \overset{iid}{\sim} \mathcal{N}(0, \sigma^2) \tag{16}$$

Where μ,ϕ and σ^2 are parameters.We start the process at $x_1 = \mu$ and simulate for x_t for $t = 2, 3, ...T$ and return the vector $x_1 : T$ containing all time points.

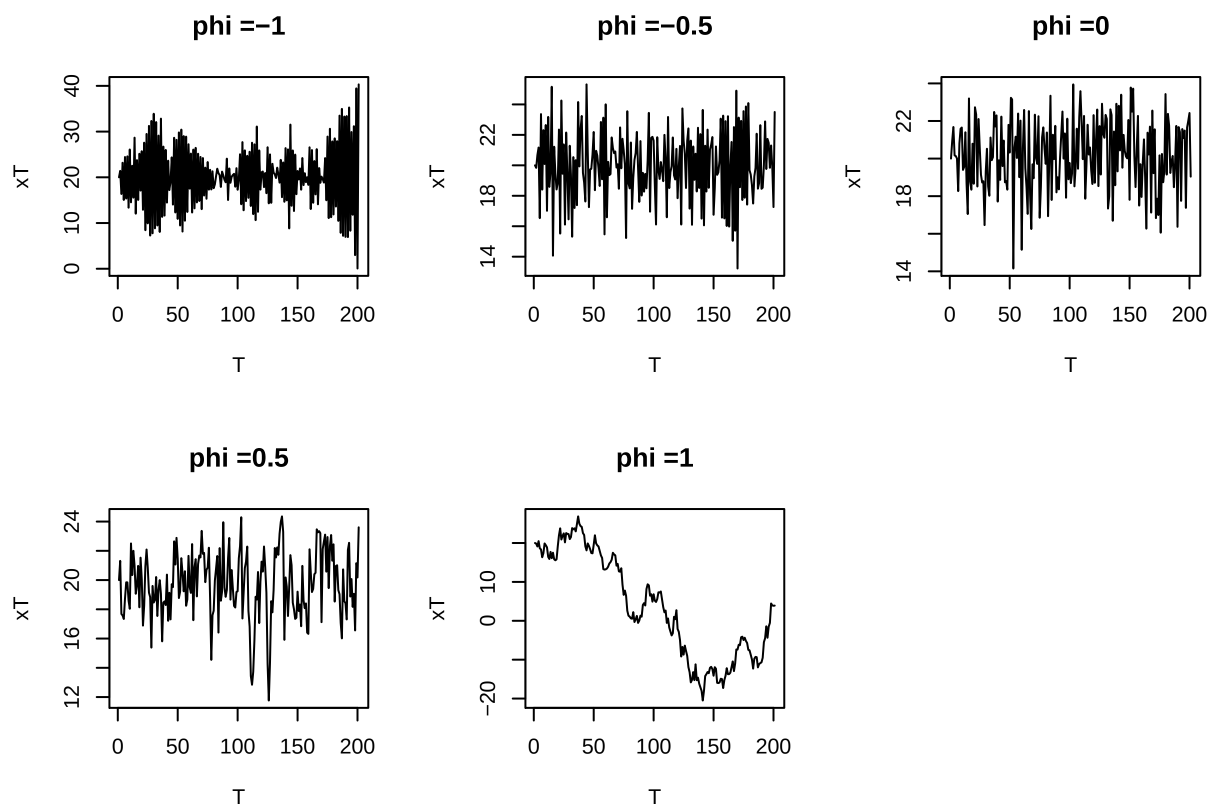
We parametize $\mu = 20$,$\sigma^2 = 4$ and T=200.Then, we experiment with various values of ϕ between -1 to 1 i.e.,$\phi = -1, -0.5, 0, 0.5, 1$

The same has been plotted in figure shown below.We see that on increasing ϕ from -1 to 1 these are changes in mean and variance of the parameter,So when moving from -1 to 1 there is a reduction in variance .Also,on moving to positive side the series starts respecting seasonality and for $\phi = 1$ there is a clean upward trend.

Therefore,ϕ effects the scale and size of generated $x_{1:T}$ based on its values.

```
## Loading required package: StanHeaders
```

```
## Loading required package: ggplot2

## rstan (Version 2.21.2, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)

## Do not specify '-march=native' in 'LOCAL_CPPFLAGS' or a Makevars file

##
## Attaching package: 'rstan'

## The following object is masked from 'package:coda':
##
##     traceplot
```



The variance among the x_t will be decrease when the ϕ value closes to zero.From the above shown plot the x_t value are ranging between 16 to 25 and that varies certain limit for the value $\phi = 0.5 - 0$

b.

We implement an AR(1) process for $\phi = 0.3$ and $\phi = 0.9$ and generate $y_{1:T}$ as synthetic data.Next we implement a Stan code that sample from the posterior of $y_{1:T}$.

For that we have assumed parameters μ, ϕ,σ^2 to be non-informative i.e.,

$$\mu \propto k_1$$

$$\phi \propto k_2$$

$$\sigma^2 \propto k_3$$

Where k_1,k_2 and k_3 are constant.Finally, we generate the samples of the posterior draws using the distribution

$$y_t \sim \mathcal{N}(\mu + \phi(y_{t-1} - \mu), \sigma^2) \tag{17}$$

1.)

```
##
## SAMPLING FOR MODEL 'ee2e00a7acb3f7d71bd22ef9230a5901' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2500 [  0%]  (Warmup)
## Chain 1: Iteration:  250 / 2500 [ 10%]  (Warmup)
## Chain 1: Iteration:  500 / 2500 [ 20%]  (Warmup)
## Chain 1: Iteration:  750 / 2500 [ 30%]  (Warmup)
## Chain 1: Iteration: 1000 / 2500 [ 40%]  (Warmup)
## Chain 1: Iteration: 1001 / 2500 [ 40%]  (Sampling)
## Chain 1: Iteration: 1250 / 2500 [ 50%]  (Sampling)
## Chain 1: Iteration: 1500 / 2500 [ 60%]  (Sampling)
## Chain 1: Iteration: 1750 / 2500 [ 70%]  (Sampling)
## Chain 1: Iteration: 2000 / 2500 [ 80%]  (Sampling)
## Chain 1: Iteration: 2250 / 2500 [ 90%]  (Sampling)
## Chain 1: Iteration: 2500 / 2500 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.194 seconds (Warm-up)
## Chain 1:                0.243 seconds (Sampling)
## Chain 1:                0.437 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'ee2e00a7acb3f7d71bd22ef9230a5901' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2500 [  0%]  (Warmup)
## Chain 2: Iteration:  250 / 2500 [ 10%]  (Warmup)
## Chain 2: Iteration:  500 / 2500 [ 20%]  (Warmup)
## Chain 2: Iteration:  750 / 2500 [ 30%]  (Warmup)
## Chain 2: Iteration: 1000 / 2500 [ 40%]  (Warmup)
## Chain 2: Iteration: 1001 / 2500 [ 40%]  (Sampling)
## Chain 2: Iteration: 1250 / 2500 [ 50%]  (Sampling)
## Chain 2: Iteration: 1500 / 2500 [ 60%]  (Sampling)
```

```
## Chain 2: Iteration: 1750 / 2500 [ 70%]  (Sampling)
## Chain 2: Iteration: 2000 / 2500 [ 80%]  (Sampling)
## Chain 2: Iteration: 2250 / 2500 [ 90%]  (Sampling)
## Chain 2: Iteration: 2500 / 2500 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.237 seconds (Warm-up)
## Chain 2:                0.266 seconds (Sampling)
## Chain 2:                0.503 seconds (Total)
## Chain 2:
```

```
##
## SAMPLING FOR MODEL 'ee2e00a7acb3f7d71bd22ef9230a5901' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2500 [  0%]  (Warmup)
## Chain 1: Iteration:  250 / 2500 [ 10%]  (Warmup)
## Chain 1: Iteration:  500 / 2500 [ 20%]  (Warmup)
## Chain 1: Iteration:  750 / 2500 [ 30%]  (Warmup)
## Chain 1: Iteration: 1000 / 2500 [ 40%]  (Warmup)
## Chain 1: Iteration: 1001 / 2500 [ 40%]  (Sampling)
## Chain 1: Iteration: 1250 / 2500 [ 50%]  (Sampling)
## Chain 1: Iteration: 1500 / 2500 [ 60%]  (Sampling)
## Chain 1: Iteration: 1750 / 2500 [ 70%]  (Sampling)
## Chain 1: Iteration: 2000 / 2500 [ 80%]  (Sampling)
## Chain 1: Iteration: 2250 / 2500 [ 90%]  (Sampling)
## Chain 1: Iteration: 2500 / 2500 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.191 seconds (Warm-up)
## Chain 1:                0.202 seconds (Sampling)
## Chain 1:                0.393 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'ee2e00a7acb3f7d71bd22ef9230a5901' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2500 [  0%]  (Warmup)
## Chain 2: Iteration:  250 / 2500 [ 10%]  (Warmup)
## Chain 2: Iteration:  500 / 2500 [ 20%]  (Warmup)
## Chain 2: Iteration:  750 / 2500 [ 30%]  (Warmup)
## Chain 2: Iteration: 1000 / 2500 [ 40%]  (Warmup)
## Chain 2: Iteration: 1001 / 2500 [ 40%]  (Sampling)
## Chain 2: Iteration: 1250 / 2500 [ 50%]  (Sampling)
## Chain 2: Iteration: 1500 / 2500 [ 60%]  (Sampling)
## Chain 2: Iteration: 1750 / 2500 [ 70%]  (Sampling)
## Chain 2: Iteration: 2000 / 2500 [ 80%]  (Sampling)
## Chain 2: Iteration: 2250 / 2500 [ 90%]  (Sampling)
```

```
## Chain 2: Iteration: 2500 / 2500 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.194 seconds (Warm-up)
## Chain 2:                0.176 seconds (Sampling)
## Chain 2:                0.37 seconds (Total)
## Chain 2:
```

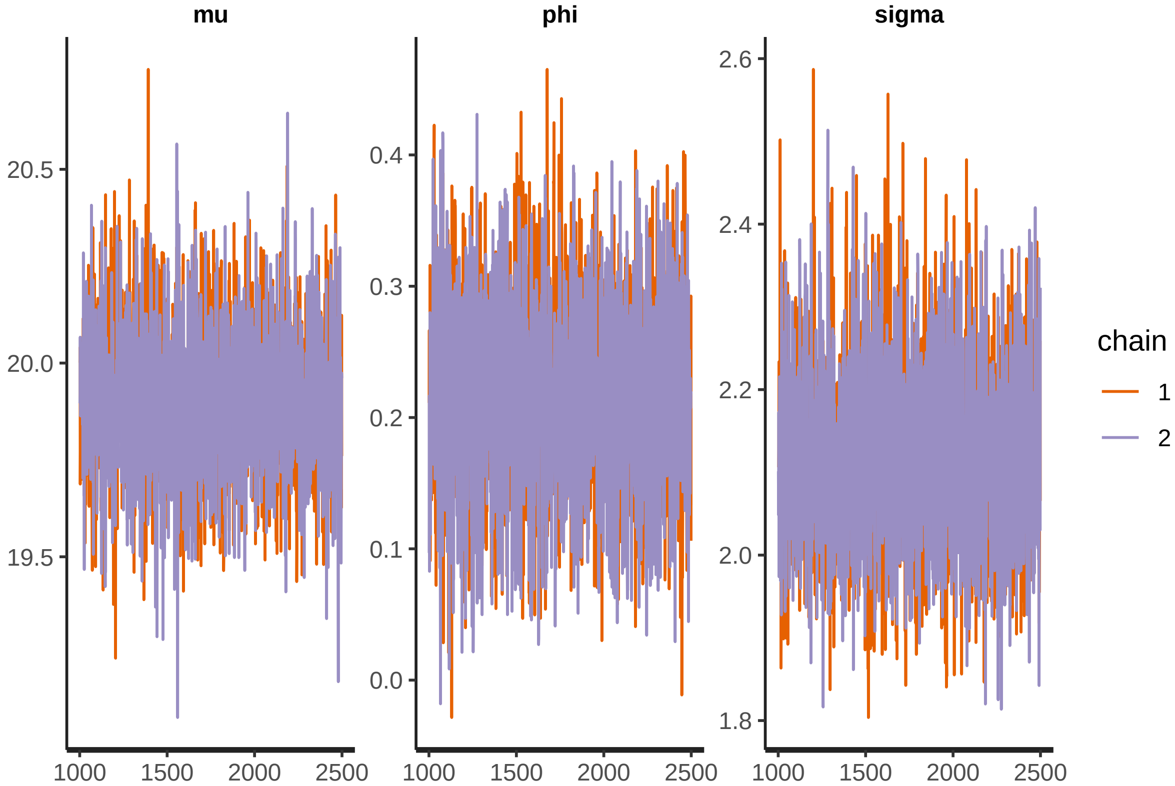
	mean	se_mean	sd	2.5%	97.5%	n_eff	Rhat
mu	19.9095586	0.0036045	0.1871976	19.5361953	20.2845149	2697.201	0.9996457
phi	0.2156293	0.0014509	0.0719059	0.0732143	0.3601646	2456.108	0.9997986
sigma	2.1183032	0.0019254	0.1079441	1.9243582	2.3470191	3143.210	0.9995011
lp__	-249.5645984	0.0314038	1.2324264	-252.7750138	-248.1663486	1540.135	1.0018481

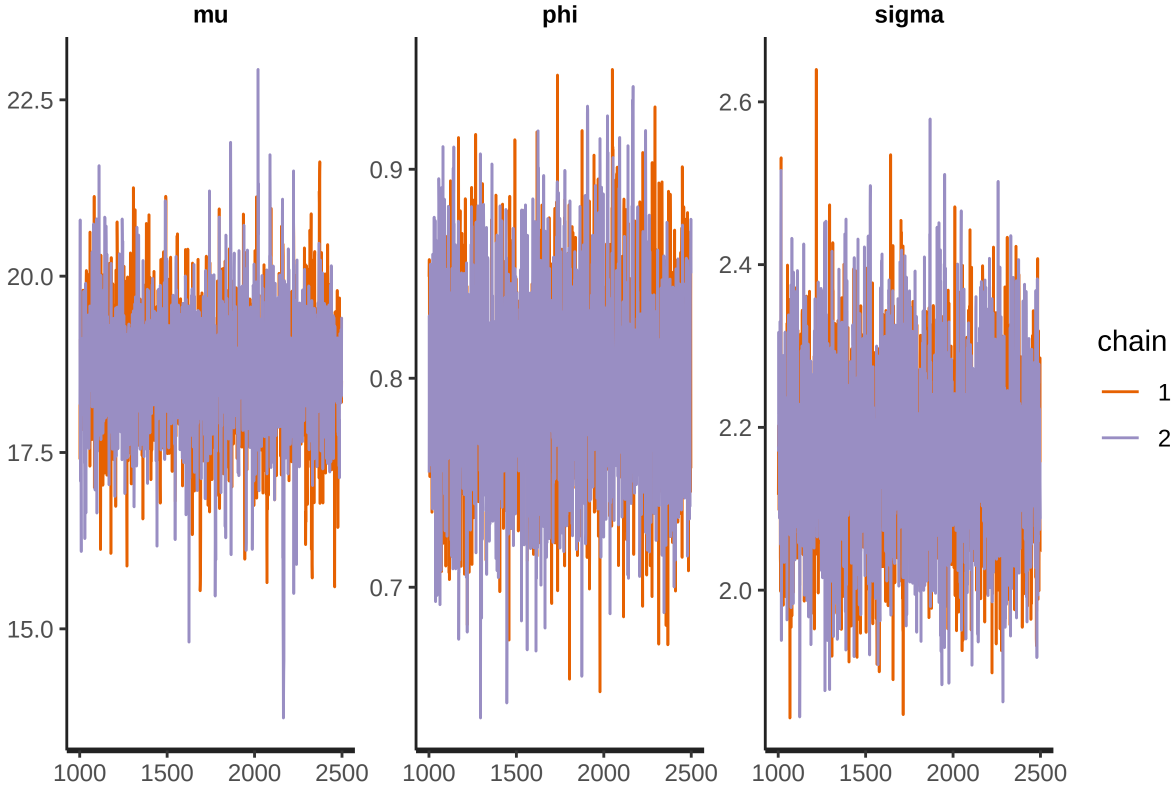
	mean	se_mean	sd	2.5%	97.5%	n_eff	Rhat
mu	18.679289	0.0230528	0.8898536	16.8279807	20.3873740	1490.0170	1.000202
phi	0.801028	0.0010974	0.0449543	0.7133427	0.8870708	1677.9486	1.000927
sigma	2.166033	0.0023975	0.1086412	1.9649320	2.3825308	2053.3952	1.000330
lp__	-254.655303	0.0442114	1.3953294	-258.4141683	-253.0978349	996.0566	1.000088

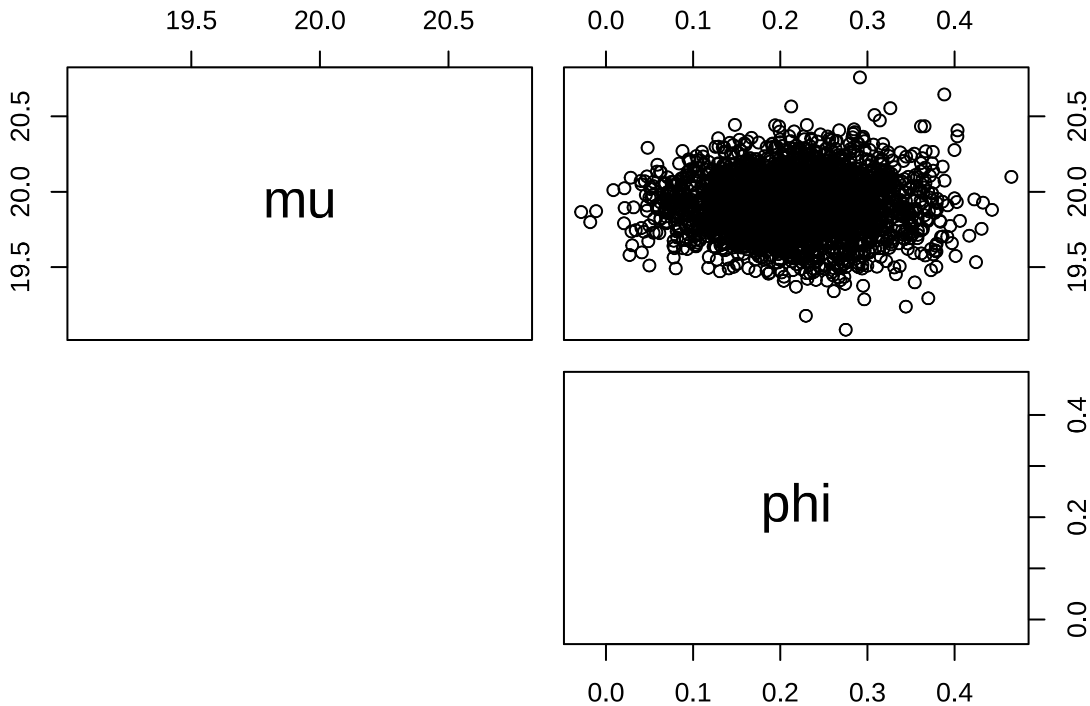
The estimated $\mu_1 = 20.117$ and $\sigma_1^2 = 2.0814$ values are equal to the mean and sigma value where we obtained from the Rstan function for the model-1.But the mean and sigma value of the model-2 is not equal to the estimated value of $\mu = 22.18$ and $\sigma = 5.299$.where the value of the simulated mean and sigma are shown above.

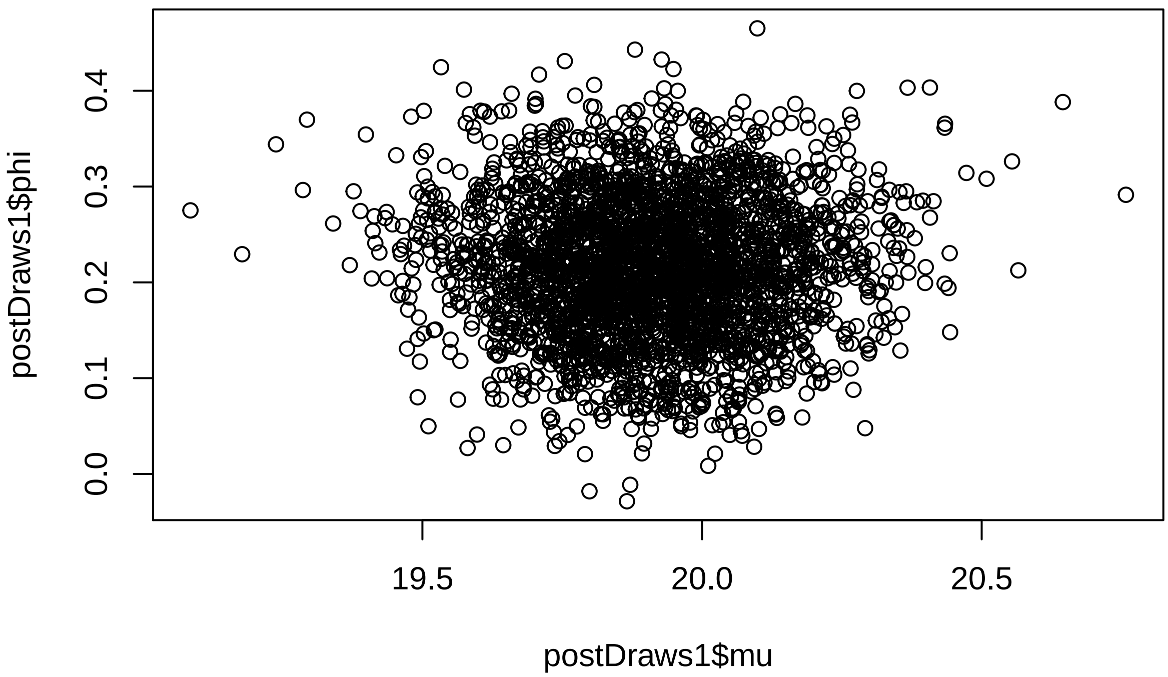
2.)

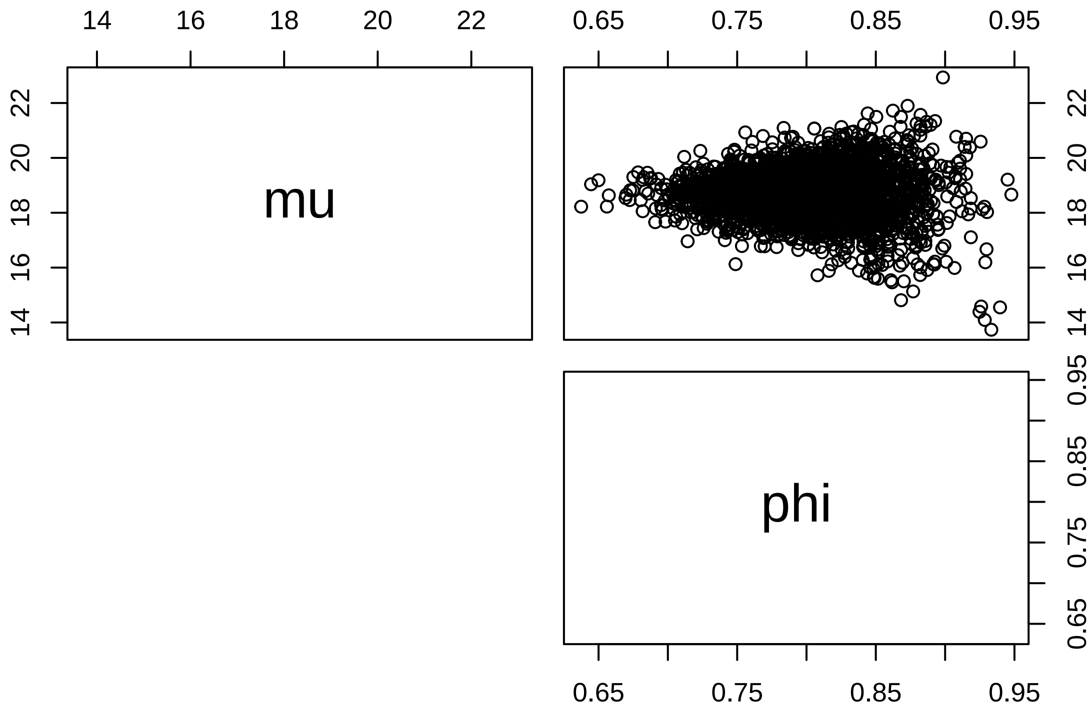
Plots of the Joint Posterior μ and ϕ

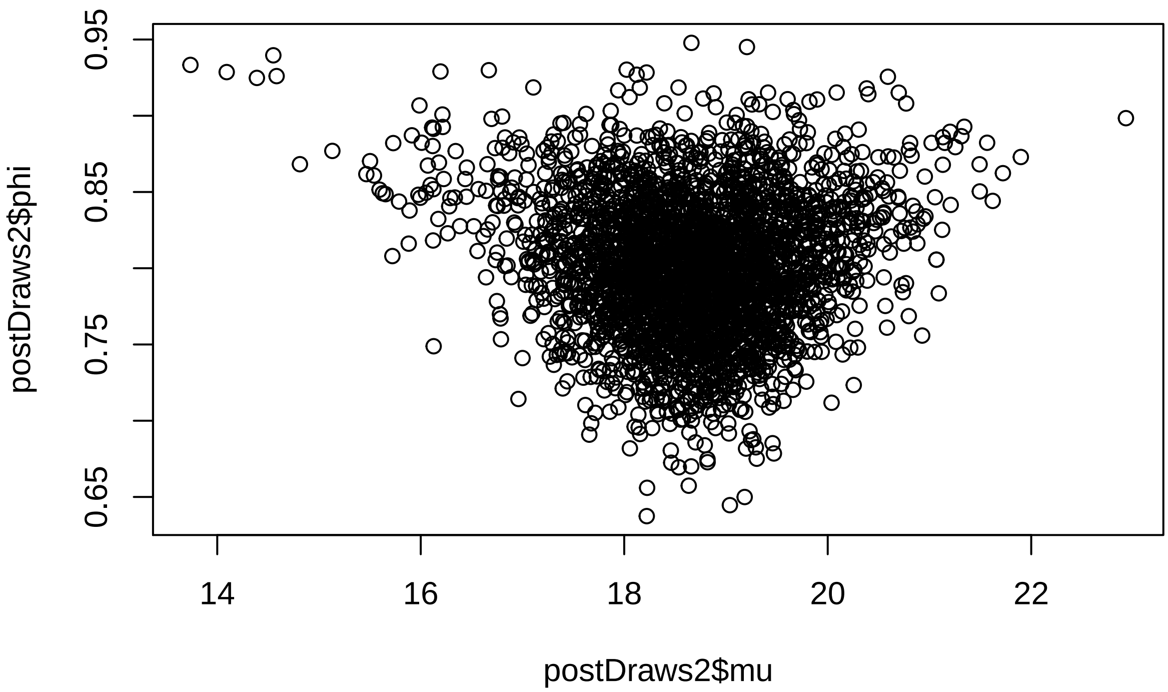

mu
phi
sigma
20.5
20.0
19.5
0.4
0.3
0.2
0.1
0.0
2.6
2.4
2.2
2.0
1.8
1000
1500
2000
2500
chain
1
2


mu
phi
sigma
22.5
20.0
17.5
15.0
0.9
0.8
0.7
2.6
2.4
2.2
2.0
1000
1500
2000
2500
chain
1
2


mu
phi
19.5
20.0
20.5
0.0
0.1
0.2
0.3
0.4


postDraws1$phi
0.0
0.1
0.2
0.3
0.4
19.5
20.0
20.5
postDraws1$mu


mu
phi
14
16
18
20
22
0.65
0.75
0.85
0.95



Appendix: All code for this report

```
knitr::opts_chunk$set(echo = TRUE)
library(knitr)


library(coda)

rain_data = as.matrix(read.table("C:/Users/Mowniesh/OneDrive/Desktop/STIMA/4-Bayesian Learning/Lab-3/ra

log_data = sapply(rain_data, FUN = log)

n = dim(rain_data)[1] # No of Observations

mean_data = mean(log_data) # mean of (log y_1,...,log Y_n)

# Initialize hyper-parameters
# mu0 = 1
mu0 = mean_data
tau02 = 1
# v0 = 1
v0 = var(log_data)
sigma02 = 1
```

```r
nDraws = 2000
gibbsDraws = matrix(0,nDraws,2)




sigma2 = (v0*sigma02)/rchisq(1,df = v0)
for(i in 1:nDraws){
  taun2 = 1/(n/sigma2 + 1/tau02)
  taun = sqrt(taun2)
  w = (n/sigma2)/( n/sigma2 + 1/tau02 )
  mun = w*mean_data +  (1-w)*mu0

  # Update mu | sigma^2, data
  mu0 = rnorm(1,mun,taun)
  gibbsDraws[i,1] = mu0
  # update sigma^2 given mu
  vn = v0+n

  var_sigma = (v0*(sigma02^2) + sum((log_data - mu0)^2))/vn
  sigma2 = (vn*var_sigma)/rchisq(1,df = vn)
  gibbsDraws[i,2] = sigma2
}


# GIBBS SAMPLING
# Mu diagnostic
par(mfrow=c(1,2))
cusumMu =  cumsum(gibbsDraws[,1])/seq(1,nDraws) # Cumulative mean value of theta1, Gibbs draws
plot(x = 1:nDraws, y = gibbsDraws[,1], type = 'l', col="red", xlab= "Draws",
     ylab = "Posterior Mean")
lines(1:nDraws, cusumMu, type = "l", col="blue")
# Sigma Diagnostic
cusuVar = cumsum(gibbsDraws[,2])/seq(1,nDraws)
plot(x = 1:nDraws, y = gibbsDraws[,2], type = 'l', col="red",
     xlab="Draws", ylab = "Posterior Variance")
lines(1:nDraws, cusuVar, type = "l", col="blue")


par(mfrow=c(1,2))
## Effective Sample SIze
# Hoetings Pg. 224
warmup = 50 # Also called 'Burn-in'
# mu
mu_Gibbs <- acf(gibbsDraws[50:nDraws,1])

IF_mu_Gibbs <- 1+2*sum(mu_Gibbs$acf[-1])

ESS_mu = nDraws/IF_mu_Gibbs
# Sigma
sigma_Gibbs = acf(gibbsDraws[50:nDraws,2])

IF_sigma_Gibbs = 1 + 2*sum(sigma_Gibbs$acf[-1])
```

```
ESS_sigma = nDraws/IF_sigma_Gibbs


## b).
y_hat = matrix(data = NA, nrow = nDraws, ncol = n)
for(i in 1:nDraws){
  y_hat[i,] = rlnorm(n, mean = gibbsDraws[i,1], sd = sqrt(gibbsDraws[i,2]))
}


data_density = density(rain_data)

hist(rain_data, freq = F, breaks = 200) # Histogram
lines(x = data_density$x, y=data_density$y, col='blue', type='l',
     lwd=5, ylim = c(0,0.04))
for(i in seq(1,nDraws, 100)){
  y_d = density(y_hat[i,])
  lines(x = y_d$x, y = y_d$y, col='red', type = 'l', lwd = 0.5)
}


library(mvtnorm)
ebayData = as.data.frame(read.table("C:/Users/Mowniesh/OneDrive/Desktop/STIMA/4-Bayesian Learning/Lab-3/

ebayData2 = ebayData[,-2]
target = ebayData[,1]

# a)
model = glm(formula = 'nBids~.', family = poisson(), data = ebayData2)

#summary(model)

#model$coefficients
# VerifyID,Sealed,MajBlem,LogBook,MinBidShare are significant
model$coefficients




# b).

X = as.matrix(ebayData[,-1]) # Remove nBids
Xnames = colnames(X) # covariate Names
N = dim(X)[1] # No. of Data points
Npar = dim(X)[2] # No. of covariates
y = ebayData$nBids # target

# Zellner's G-prior statistics:
mu = as.matrix(rep(0,Npar)) # beta prior mu
sigma  = 100*solve(t(X)%*%X) # beta prior sigma

beta_init = as.vector(rep(0,Npar)) # Initializing betas
```

```
logPost = function(beta_init,X,y,mu,sigma){
  p1 = y%*%X%*%beta_init
  p2 = sum(exp(X%*%beta_init))
  logLik = p1-p2
  logPrior = dmvnorm(beta_init, mean = mu, sigma = sigma, log = TRUE)
  return(logLik + logPrior)
}

#logPost(beta_init=beta_init,X,y,mu,sigma)
OptimRes = optim(beta_init, logPost, gr=NULL,X,y,mu,sigma, method = c('BFGS'),
      control = list(fnscale=-1), hessian = TRUE)

beta_mode = t(OptimRes$par) # Beta @ mode
colnames(beta_mode) = Xnames
hessian = OptimRes$hessian
post_var_mat = - solve(hessian)
beta_mode
#knitr::kable((beta_mode))
f.MCMC.MH = function(logPostFunc,nDraws,c,beta_init,X,y,mu,sigma,hessian, warmup=0){
  post_var_mat = - solve(hessian)*c
  beta_matrix = matrix(nrow = nDraws,ncol = length(beta_init))
  beta_matrix[1,] = beta_init
  accept = 0
  for(i in 2:nDraws){
    beta_now = beta_matrix[i-1,] # {beta_{i-1}}
    # Draw candidate
    beta_cand = as.vector(rmvnorm(1,mean = beta_now, sigma = post_var_mat))
    # Calculate Metropolis-Hasting ratio
    # q(.|.)~ Normal. hence it is excluded in ratio calculation due to symmetry
    R = exp(logPostFunc(beta_init=beta_cand,X,y,mu,sigma) - logPostFunc(beta_init=beta_now,X,y,mu,sigma
    #
    alpha = min(1,R)
    # Draw from Uniform distribution
    u = runif(1)
    if(u <= alpha){
      beta_now = beta_cand
      accept = accept+1
    }
    beta_matrix[i,] = beta_now
  }
  return(list(beta = beta_matrix[warmup:nDraws,], acceptRate = accept/nDraws*100))
}




### c)

result = f.MCMC.MH(logPost, nDraws = 5000, c=0.6,beta_init=as.vector(rep(0,Npar)),
                   X,y,mu,sigma, hessian, warmup = 1000)
#result$acceptRate
par(mfrow = c(3,3))
```

```r
for (i in 1:Npar){
  plot(y = result$beta[,i], x = 1000:5000, type = 'l',
       main = Xnames[i], xlab="Draws", ylab = paste0('beta: ', Xnames[i]),
       col="blue")
  cusuM = cumsum(result$beta[,i])/seq(1,4001)
  lines(x = 1000:5000, cusuM, type = "l", col="red",
        lwd=2)
}


# d).
# New Auction data
x = as.vector(c(1,1,1,1,0,0,0,1,0.7))
betas = result$beta
N = length(exp(betas%*%x))
par(mfrow=c(1,1))
y_hat = rpois(n=N, lambda = exp(betas%*%x))
hist(y_hat, freq = T,xlab="Number of bidders",main="Histogram For Number of Bidders")

n0_bids<-mean(y_hat==0) #Pr(nBids = 0)


library(rstan)
## a)

AR1 = function(stamps,mu,error_var,phi){
  sigma = sqrt(error_var)
  X = vector(length = stamps+1)
  X[1] = mu
  for(i in 1:stamps){
    X[i+1] = mu + phi*(X[i]-mu)+ rnorm(1,mean = 0,sd = sigma)
  }
  return(X)
}

r = matrix(nrow = 5, ncol = 201)
phi = c(-1,-0.5,0,0.5,1)
for(i in 1:5){
r[i,] = AR1(stamps = 200,mu=20,error_var = 4, phi = phi[i])
}
#@Mowniesh: Use this for figure in 3 (a)
# Refer https://otexts.com/fpp2/stationarity.html for interpretation
par(mfrow = c(2,3))
for(i in 1:5){
plot(r[i,], type = 'l', xlab='T', ylab='xT',
     main = paste0("phi =",phi[i]))
}
simData1 = AR1(stamps = 200,mu=20,error_var = 4,phi = 0.3)
mu_1 = mean(simData1) # True mean
sigma_1 = sqrt(var(simData1)) # True Var
# phi = 0.9
simData2 = AR1(stamps = 200,mu=20,error_var = 4,phi = 0.9)
mu_2 = mean(simData2)
```

```
sigma_2 = sqrt(var(simData2))
###
# https://mc-stan.org/docs/2_22/stan-users-guide/autoregressive-section.html
StanModel = '
data {
  int<lower=0> N;
  vector[N] y;
}
parameters {
  real mu;
  real<lower=-1, upper=1> phi;
  real<lower=0> sigma;
}
model {
  for (n in 2:N)
    y[n] ~ normal(mu + phi*(y[n-1]-mu), sigma);
}

'

N = length(simData1)
data1 = list(N=N,y=simData1)
data2 = list(N=N, y=simData2)
warmup=1000
nDraws = 2500





# i).
fit1 = stan(model_code = StanModel, data = data1,
           warmup = warmup,iter = nDraws, chains = 2)
### Fitted Model
#print(fit1, digits_summary=3)

fit2 = stan(model_code = StanModel, data = data2,
            warmup = warmup,iter = nDraws, chains = 2)
### Fitted Model
#print(fit2, digits_summary=3)



kable(summary(fit1, probs = c(0.025, 0.975))$summary)
kable(summary(fit2, probs = c(0.025, 0.975))$summary)
## Extract Posterior Draws
postDraws1 = extract(fit1)
postDraws2 = extract(fit2)

## ii).
# Automatic Traceplot
traceplot(fit1)
traceplot(fit2)
```

```r
# Bivariate posterior Plots
# For 1st Draw
pairs(~ mu + phi, data = fit1,lower.panel=NULL )

plot(x = postDraws1$mu, y = postDraws1$phi)

# For 2nd Draw
pairs(~ mu + phi, data = fit2,lower.panel=NULL )

plot(x = postDraws2$mu, y = postDraws2$phi)
```