Computer Lab3 - 732A73

Helena Llorens Lluís (hllor282), Yi Yang (yiyan338)

2025-05-15

1. Gibbs sampling for the logistic regression

We aim to implement a Gibbs sampler for a logistic regression model using Polya-Gamma latent variable augmentation.We augment the data with Polya-gamma latent $variables^2$,$i = 1, ...n$:

$$\omega_i = \frac{1}{2\pi^2} \sum_{k=1}^{\infty} \frac{g_k}{\left(k - \frac{1}{2}\right)^2 + \left(\frac{x_i'\beta}{4\pi^2}\right)^2}$$

where $g_k \sim Exp(1)$ are independent draws from the exponential distribution with mean.This representation allows the logistic likelihood to be expressed in a conditionally Gaussian form, enabling efficient Bayesian updates.

Given the augmented latent variables $\omega = (\omega_1, ..., \omega_n)$, we sample from the joint posterior $p(\omega, \beta \mid y)$ using the following steps:

1. Sample latent variables:$\omega_i \mid \beta \sim \text{PG}(1, x_i^\top \beta), \quad i = 1, \ldots, n$
2. Sample regression coefficients β from the multivariate normal posterior:$\beta \mid y, \omega \sim \mathcal{N}(m_\omega, V_\omega)$ where:$V_\omega = \left(X^\top \Omega X + B^{-1}\right)^{-1}$, $m_\omega = V_\omega \left(X^\top \kappa + B^{-1} b\right)$ With:

$$\kappa = \begin{pmatrix} y_1 - \frac{1}{2} \\ \vdots \\ y_n - \frac{1}{2} \end{pmatrix}$$

$$\Omega = \text{diag}(\omega_1, \ldots, \omega_n)$$

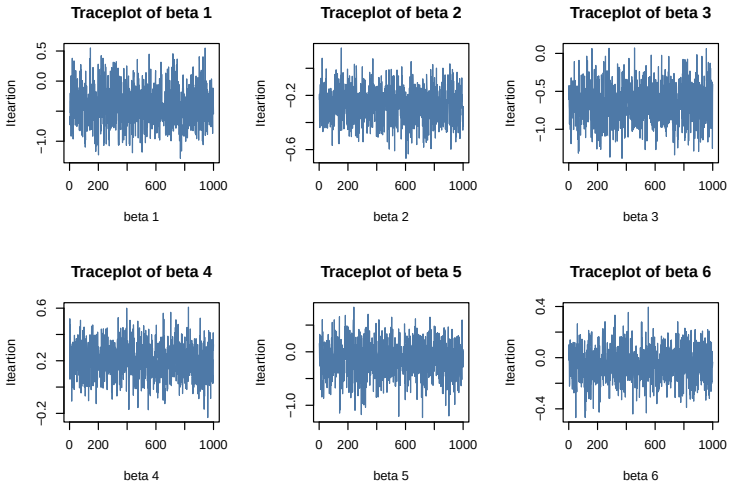
$$\beta \sim \mathcal{N}(b, B)$$

We assume a normal prior on the coefficients $\boldsymbol{\beta} \sim \mathcal{N}(0, \tau^2 I)$ with $\tau = 3$.And evaluate convergence using trace plots and Inefficiency Factors (IFs).The Inefficiency Factor (IF) measures how much autocorrelation is present in the Markov Chain. An IF close to 1 indicates nearly independent samples (ideal), while larger values suggest higher autocorrelation and less efficient sampling.

Table 1: Inefficiency Factors

	IFs_a
beta1	0.9386926
beta2	1.0301006

	IFs_a
beta3	0.9704138
beta4	1.3467480
beta5	1.1802178
beta6	0.9521449



The estimated Inefficiency Factors (IFs) for the six regression coefficients range from approximately 0.93 to 1.34. Most parameters exhibit IFs close to 1, indicating efficient sampling with low autocorrelation. Overall, the Gibbs sampler appears to perform well in terms of sampling efficiency.

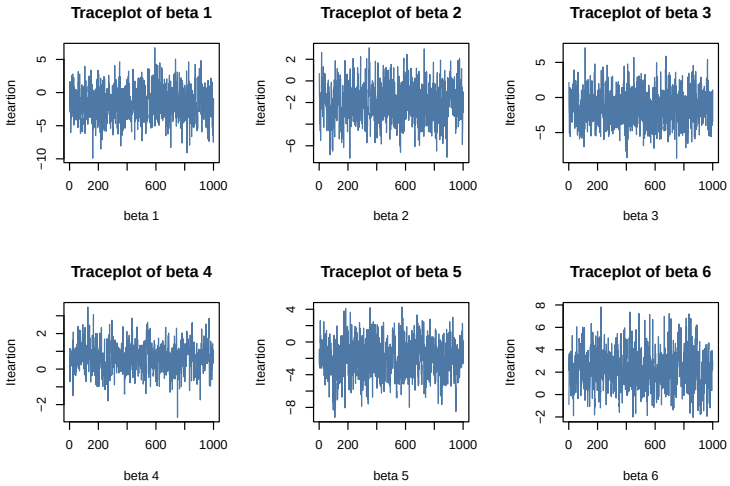
The trace plots for all six parameters exhibit good mixing and stationarity, with no apparent trends or drifts. The chains fluctuate steadily around stable means, indicating that the sampler has converged and is effectively exploring the posterior distribution.

Then, we repeat the Gibbs sampling procedure described in (a), but only using the first **m** observations of the dataset.We implement the sampling for $m = 10, 40, 80$, and then analyze the posterior draws for the regression coefficients β. The Gibbs sampler is run for **1000** iterations, using the same prior variance$\sigma^2 = 3^2$.

$m = 10$:

Table 2: Inefficiency Factors when m=10

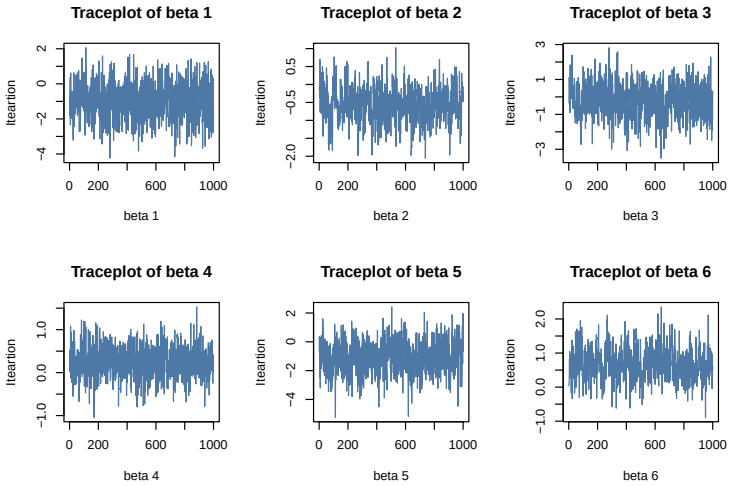
	IFs_10
beta1	1.073536
beta2	2.657061
beta3	1.691435
beta4	3.232157
beta5	2.878239
beta6	2.135046



$m = 40$:

Table 3: Inefficiency Factors when m=40

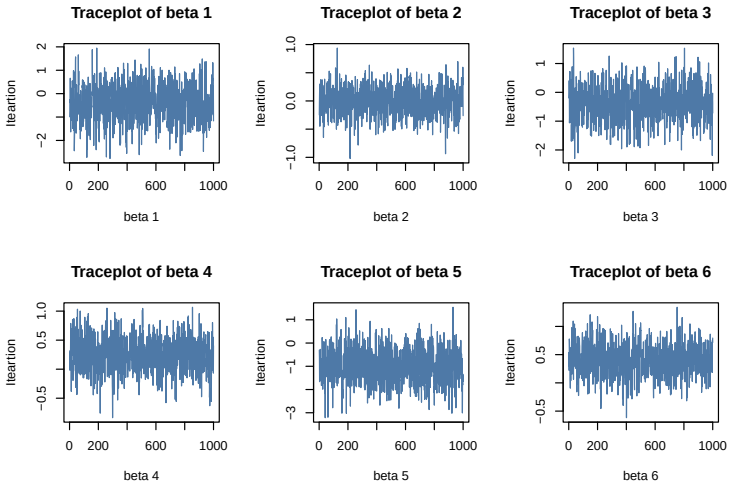
	IFs_40
beta1	1.512644
beta2	2.599845
beta3	2.107945
beta4	1.625064
beta5	2.888913
beta6	3.853052



$m = 80$:

Table 4: Inefficiency Factors when m=80

	IFs_80
beta1	1.094688
beta2	1.303400
beta3	1.599420
beta4	1.267893
beta5	1.627268
beta6	1.271844



For almost all coefficients, the inefficiency factors decrease significantly as m increases from 10 to 80.This shows that the posterior draws become more efficient (less autocorrelated) with more data and the posterior becomes more concentrated.However, we observe an interesting phenomenon at $m = 40$, where some coefficients (notably β_6) exhibit noticeably higher inefficiency factors compared to both $m = 10$ and $m = 80$. This suggests that with a moderate sample size, the Markov chains may experience temporary mixing issues or higher posterior uncertainty for certain parameters. Such non-monotonic behavior could be due to increased complexity in the posterior landscape that temporarily affect sampling efficiency. Therefore, it is important to not assume that increasing the sample size always leads to smoother convergence—performance can fluctuate depending on the structure of the data and the parameter space.

```r
library("BayesLogit")
library("mvtnorm")
#(a)
#read data
data <- read.csv("Disease.csv")
data <- as.data.frame(data)

#normalize age, symtoms and white blood counts
age_mean <- mean(data$age)
age_sd <- sd(data$age)
duration_mean <- mean(data$duration_of_symptoms)
duration_sd <- sd(data$duration_of_symptoms)
white_mean <- mean(data$white_blood)
white_sd <- sd(data$white_blood)
```

```r
data[,2] <- (data[,2]-age_mean)/age_sd
data[,4] <- (data[,4]-duration_mean)/duration_sd
data[,6] <- (data[,6]-white_mean)/white_sd

y <- data[,7] # response
X <- as.matrix(data[,1:6]) # Covariates


# prior sigma2
sigma2 <- 3^2

#number of iterations
num_iterations <- 1000


#Gibbs sampling
gibbs <- function(X,y,num_iterations,sigma2){
  if(nrow(X)!=length(y))
    stop("X and y should have the same length.")

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

  #initialize beta
  beta_samples <- matrix(NA,nrow = num_iterations,ncol = p)
  beta <- rep(0,p)


for (i in 1:num_iterations) {
  #latent variable omega
  #scale parameter for PG(1,x*beta)
  scalePar <- X%*%beta
  #construct latent variables for each observation thus num=n
  omega_variable <- rpg(num=n,h=rep(1,n),z=scalePar) #poly-Gamma sampling

  #update beta
  Omega_diagmatrix <- diag(omega_variable)

  V_inverse <- t(X) %*% Omega_diagmatrix %*%X +diag(1/sigma2,p)
  V <- solve(V_inverse)

  k <- y-0.5
  b <- rep(0,p) #prior mean
  m <- V%*%(t(X)%*%k+solve(diag(1/sigma2,p))%*%b)
  beta <- as.vector(rmvnorm(1,mean = m,sigma = V ))

  beta_samples[i,] <- beta
}
  return(beta_samples)
}

result <- gibbs(X=X,y=y,num_iterations=1000,sigma2 = 3^2)
```

```r
#compute IFs ,lag.max = max_lag ,max_lag=100

Ifs <- function(chain){
  acf <- acf(chain,plot=F)$acf[-1]
  return(1+2*sum(acf))
}

IFs_a <- apply(result, 2,Ifs)

#compute Ifs using coda package
#library("coda")
#mcmc_result <- as.mcmc(result)
#ess <- effectiveSize(mcmc_result)
#ifactor <- nrow(result)/ess

#plot the trajectories of the sampled Markov chains

par(mfrow=c(2,3))
for (i in 1:ncol(result)) {
  plot(result[,i],type="l",col="#4E79A7",
       main=paste("Traceplot of beta",i),
       xlab=paste("beta",i),
       ylab="Iteartion",
       lwd=1)
}


#(b)
#m=10
result_10 <- gibbs(X=X[1:10,],y=y[1:10],num_iterations=1000,sigma2 = 3^2)
IFs_10 <- apply(result_10, 2, Ifs)

par(mfrow=c(2,3))
for (i in 1:ncol(result_10)) {
  plot(result_10[,i],type="l",col="#4E79A7",
       main=paste("Traceplot of beta",i),
       xlab=paste("beta",i),
       ylab="Iteartion",
       lwd=1)
}

#m=40

result_40 <- gibbs(X=X[1:40,],y=y[1:40],num_iterations=1000,sigma2 = 3^2)
IFs_40 <- apply(result_40, 2, Ifs)

par(mfrow=c(2,3))
for (i in 1:ncol(result_40)) {
  plot(result_40[,i],type="l",col="#4E79A7",
       main=paste("Traceplot of beta",i),
       xlab=paste("beta",i),
       ylab="Iteartion",
       lwd=1)
```

```
}

#m=80
result_80 <- gibbs(X=X[1:80,],y=y[1:80],num_iterations=1000,sigma2 = 3^2)
IFs_80 <- apply(result_80, 2, Ifs)

par(mfrow=c(2,3))
for (i in 1:ncol(result_80)) {
  plot(result_80[,i],type="l",col="#4E79A7",
       main=paste("Traceplot of beta",i),
       xlab=paste("beta",i),
       ylab="Iteartion",
       lwd=1)
}
```

2. Metropolis Random Walk for Poisson regression

We consider the Poisson regression model:

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

To obtain the MLE of β, we fit the model using the `glm()` function with a Poisson likelihood.

```
# read data
data <- read.table("eBayNumberOfBidderData_2025.dat", header = T)

# (a) Poisson MLE using glm
mod <- glm(nBids ~ PowerSeller + VerifyID + Sealed + Minblem + MajBlem +
             LargNeg + LogBook + MinBidShare, data = data, family = "poisson")
coefs <- summary(mod)$coefficients
kable(coefs, caption = "Summary of the Poisson regression model")
```

Table 5: Summary of the Poisson regression model

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	1.0853366	0.0359229	30.2129332	0.0000000
PowerSeller	-0.0283250	0.0443252	-0.6390282	0.5228046
VerifyID	-0.3490151	0.1300773	-2.6831361	0.0072935
Sealed	0.5010095	0.0667643	7.5041528	0.0000000
Minblem	-0.1218935	0.0838830	-1.4531365	0.1461859
MajBlem	-0.2488434	0.0986451	-2.5226133	0.0116486
LargNeg	0.0361005	0.0707495	0.5102585	0.6098704
LogBook	-0.0727990	0.0350548	-2.0767205	0.0378274
MinBidShare	-1.7666504	0.0829198	-21.3055341	0.0000000

From this table, we observe that the covariates `VerifyID`, `Sealed`, `MajBlem`, `LogBook` and `MinBidShare` are statistically significant at the 5% level. This suggests that these variables have a meaningful effect on the expected number of bids in an eBay auction.

We now adopt a Bayesian approach to inference in the Poisson regression model. Specifically, we place a Zellner's g-prior on the regression coefficients β, defined as:

$$\beta \sim N[0, 100 \cdot (X^T X)^{-1}]$$

To approximate the posterior, we assume it is approximately multivariate normal:

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

where $\tilde{\beta}$ is the posterior mode and $J_Y^{-1}(\tilde{\beta})$ is the negative Hessian and the posterior mode. We look for these values by numerical optimization. We obtain both via numerical optimization of the log-posterior using the BFGS algorithm:

```
# (b)
y <- data$nBids
X <- model.matrix(mod)

# log-posterior function
log_posterior <- function(beta, X, y){
  XtX <- t(X) %*% X
  sigma0_inv <- (1/100) * XtX
  # llik
  eta <- X %*% beta
  llik <- sum(y * eta - exp(eta))

  # log-prior (gaussian with mean 0)
  logprior <- -0.5 * t(beta) %*% sigma0_inv %*% beta

  # Return scalar value
  log_post <- llik + logprior

  return(as.numeric(log_post))
}

# find posterior mode
# Initialize beta
init_beta <- rep(0, ncol(X))

# Optimization
opt_result <- optim(
  par = init_beta,
  fn = function(beta) -log_posterior(beta, X, y), # negative because optim minimizes
  method = "BFGS",
  hessian = TRUE
)

# Posterior mode and Hessian
beta_tilde <- opt_result$par
Hessian <- opt_result$hessian
```

Below are the estimated values of the posterior mode $\tilde{\beta}$

```
kable(round(beta_tilde, 2), caption = "Posterior mode estimates of beta")
```

Table 6: Posterior mode estimates of beta

x
1.08
-0.03
-0.35
0.50
-0.12
-0.25
0.04
-0.07
-1.76

We also extract the observed information matrix $J_Y(\tilde{\beta})$

```
kable(round(Hessian, 2), caption = "Negative Hessian (observed information) at the posterior mode")
```

Table 7: Negative Hessian (observed information) at the posterior mode

2483.52	1050.55	63.15	271.74	154.05	109.95	241.82	408.67	-473.48
1050.55	1050.55	32.86	159.59	58.61	56.62	31.59	-98.47	-77.05
63.15	32.86	63.15	29.05	1.84	0.00	0.00	18.57	-10.46
271.74	159.59	29.05	271.74	0.00	0.00	0.00	19.20	-25.41
154.05	58.61	1.84	0.00	154.05	0.00	16.20	34.36	-24.07
109.95	56.62	0.00	0.00	0.00	109.95	0.00	27.41	-28.23
241.82	31.59	0.00	0.00	16.20	0.00	241.82	158.42	-77.24
408.67	-98.47	18.57	19.20	34.36	27.41	158.42	1301.18	-362.91
-473.48	-77.05	-10.46	-25.41	-24.07	-28.23	-77.24	-362.91	310.73

To obtain samples from the true posterior distribution of the regression coefficients β, we implement the Random Walk Metropolis-Hastings algorithm. The proposal distribution is defined as a multivariate normal random walk:

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

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

The following code defines and runs the Metropolis sampler:

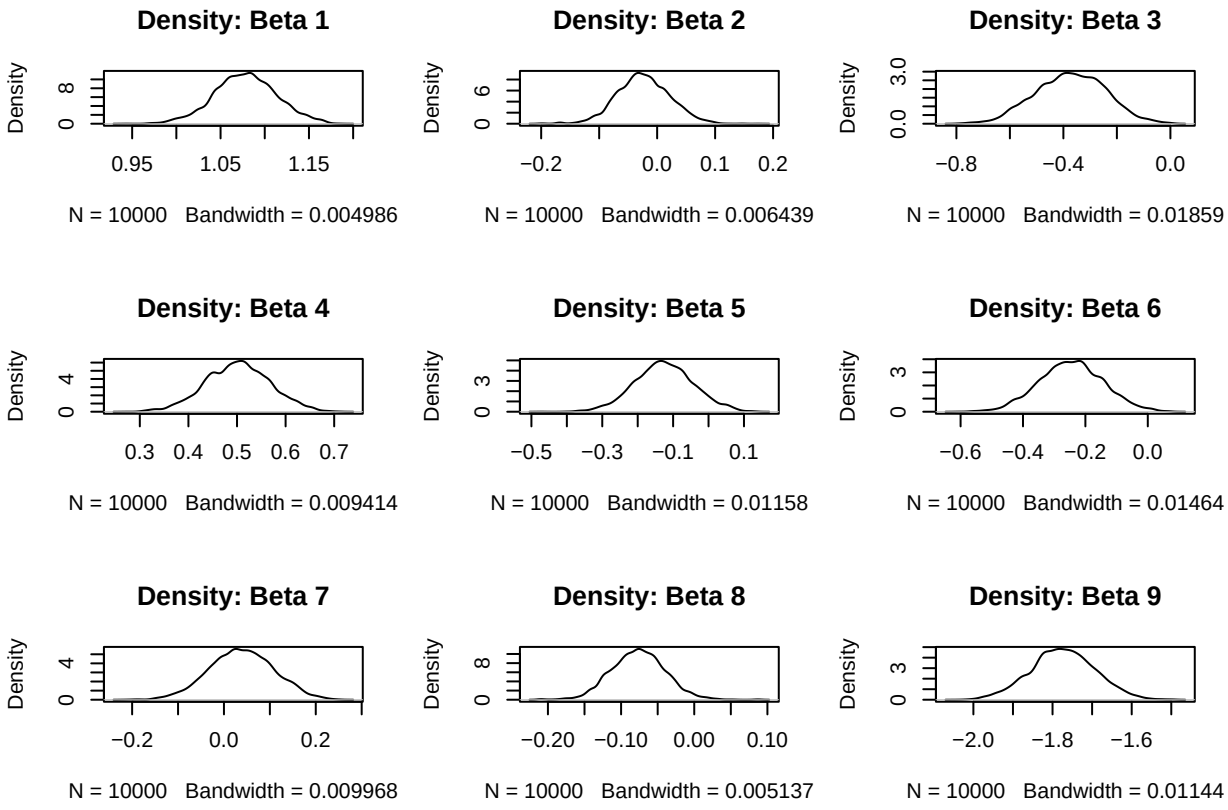
```
# (c)
MetropolisHastings <- function(x_init, n_iter, prop_dist_rand, target, X, y, Sigma, c){
  set.seed(1234)

  # initial values
  p <- length(x_init)
  x <- matrix(NA, nrow = n_iter, ncol = p)
  x[1, ] <- x_init
  acc <- 0

  for (t in 2:n_iter) {
```

```
    x_candidate <- prop_rand(x[t - 1, ], Sigma, c)
    x_candidate <- as.vector(x_candidate)

    log_r <- target(x_candidate, X, y) - target(x[t - 1, ], X, y)
    if (log(runif(1)) < log_r) {
      x[t, ] <- x_candidate
      acc <- acc + 1
    } else {
      x[t, ] <- x[t - 1, ]
    }
  }
  return(list(chain = x, acceptance_rate = sum(acc)/(n_iter-1)))
}

prop_rand <- function(theta, Sigma, c){
  mvtnorm::rmvnorm(1, mean = theta, sigma = c * Sigma)
}

#init_beta <- rep(0, ncol(X))  # or use beta_tilde from earlier optimization
Sigma <- solve(Hessian)

result <- MetropolisHastings(
  x_init = beta_tilde, # init_beta,
  n_iter = 10000,
  prop_dist = prop_rand,
  target = log_posterior,
  X = X,
  y = y,
  Sigma = Sigma,
  c = 0.5  # tuning parameter
)

posterior_samples <- result$chain
acceptance_rate <- result$acceptance_rate
```

We chose a tuning parameter `c` that yields an acceptance rate around 25–30% because this range is widely recommended for Random Walk Metropolis algorithms, balancing the trade-off between proposal acceptance and efficient exploration of the posterior distribution. We got an acceptance rate of 0.320332, which is acceptable.

To further explore the posterior distributions of the parameters β_j we plot the estimated marginal densities

```
# Density plots
par(mfrow = c(3, 3))
for (j in 1:ncol(posterior_samples)) {
  plot(density(posterior_samples[, j]), main = paste("Density: Beta", j))
}
```



Using the MCMC samples, we now simulate from the posterior predictive distribution of the number of bidders in a new auction with the following characteristics:

- PowerSeller = 1
- VerifyID = 0
- Sealed = 1
- MinBlem = 0
- MajBlem = 1
- LargNeg = 0
- LogBook = 1.3
- MinBidShare = 0.7

This involves two steps:

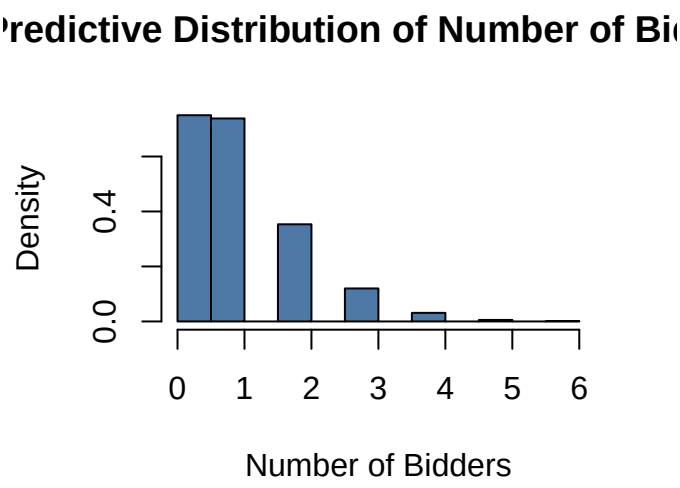
1. Compute the rate parameter λ for each posterior sample of β:

$$\lambda^{(s)} = exp(x_{new}^T \beta^{(s)}), \quad s = 1, ..., S$$

2. Simulate the number of bidders from a Poisson distribution using each $\lambda^{(s)}$

$$\tilde{y}^{(s)} \sim Poisson(\lambda^{(s)})$$

```r
# (d)
x_new <- c(1,    # Intercept
           1,    # PowerSeller
           0,    # VerifyID
           1,    # Sealed
           0,    # MinBlem
           1,    # MajBlem
           0,    # LargNeg
           1.3,  # LogBook
           0.7)  # MinBidShare

# Compute lambda for each posterior draw
lambda_draws <- apply(posterior_samples, 1, function(beta) {
  exp(sum(beta * x_new))
})

# Simulate nBids for new auction
set.seed(1234)
pred_nBids <- rpois(length(lambda_draws), lambda = lambda_draws)

# Plot predictive distribution
par(mfrow = c(1,1))
hist(pred_nBids, col = "#4E79A7", probability = TRUE, breaks = 20,
     main = "Predictive Distribution of Number of Bidders",
     xlab = "Number of Bidders")
```




```r
# Estimate P(nBids = 0)
prob_zero_bidders <- mean(pred_nBids == 0)
```

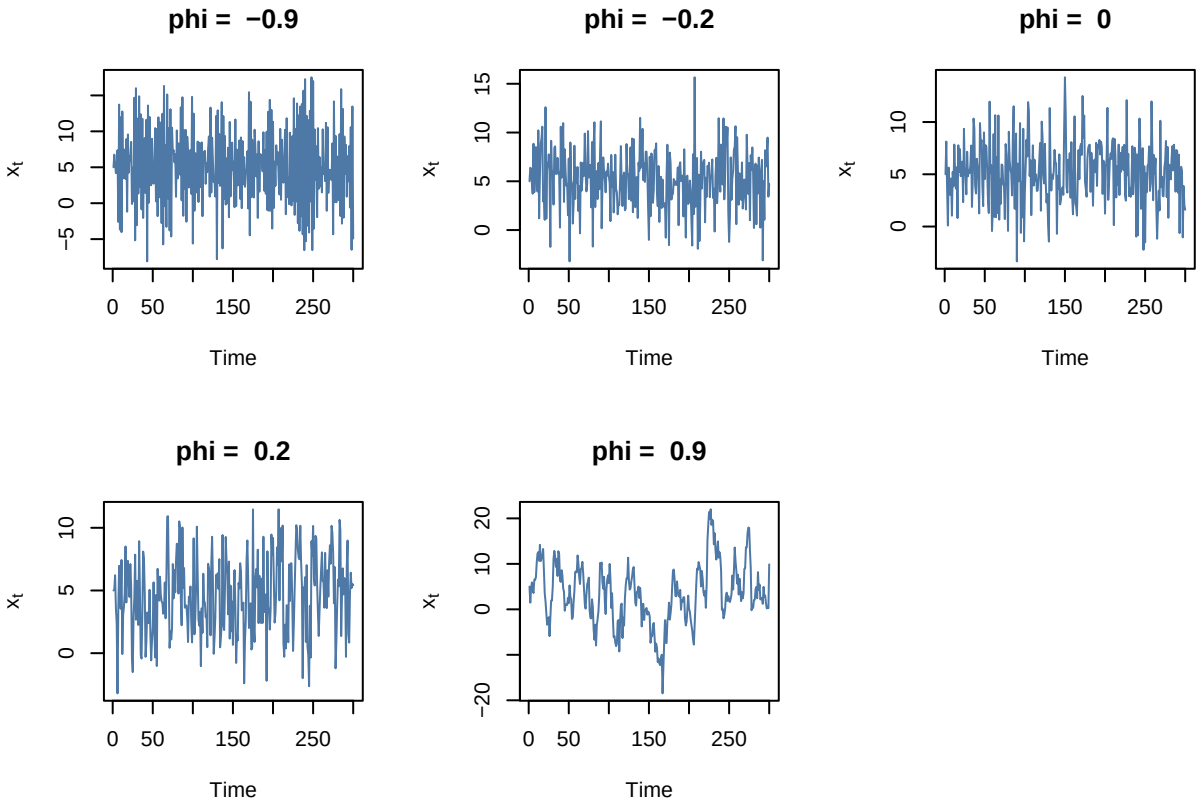
The probability of no bidders in this new auction is 0.3751.

3. Time series models in Stan

We aim to simulate and fit an AR(1) (Auto-Regressive of order 1) process to study how the current value of a time series depends on its previous value. The AR(1) model is defined as:

$$x_t = \mu + \phi(x_{t-1} - \mu) + \varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0, \sigma^2)$$

where:
μ is the long-term (stationary) mean of the process.
ϕ is the autoregressive coefficient, which determines how strongly the past value influences the current one.
$\sigma^2$is the variance of the Gaussian noise.
$|\phi| < 1$ensures stationarity of the process.
We first write a function simulates data from the AR(1) model, then use $\mu = 5$, $\sigma^2 = 9$, and $t = 300$ with values of ϕ between `-1 and 1`.The plots show as follows:



From the plots, we observe that the value of $\phi = -0.9, -0.2, 0, 0.2, 0.9$ has a strong influence on the behavior of the time series. - When $\phi = 0$, the time series behaves like white noise: each value is independent and identically distributed with no dependence on past values. - When $\phi > 0$, especially near 1 (e.g., $\phi = 0.9$), the series shows strong positive autocorrelation: values change slowly, resulting in smoother and more persistent trends. - When $\phi < 0$, especially near -1 (e.g., $\phi = -0.9$), the series exhibits negative autocorrelation: values tend to alternate direction rapidly, creating high-frequency oscillations. - As $\phi \to 1$, the process changes more slowly and stays in the same direction longer, meaning that past values have a stronger influence on current values.

Then,we focus on performing Bayesian inference for the parameters of an AR(1) process using Stan, based on simulated data from Question 1.

We first use the `AR1()` function (from Question 1) to simulate two datasets of length $t = 300$ with known parameters:
-Dataset1: $\mu = 5$,$\phi = 0.4$ and $\sigma^2 = 9$
-Dataset2: $\mu = 5$,$\phi = 0.98$ and $\sigma^2 = 9$

Bayesian Model Specification

We use a Bayesian approach to estimate the three parameters:μ =,ϕ and σ,where $\sigma = \sqrt{\sigma^2}$.The model was defined in Stan as:

$$x_1 \sim \mathcal{N}(\mu, \sigma)$$
$$x_t \sim \mathcal{N}(\mu + \phi(x_{t-1} - \mu), \sigma), \quad t = 2, \ldots, T$$

Priors
We use weakly informative (non-informative) priors to reflect minimal prior knowledge:
$\mu \sim \mathcal{N}(0, 100)$:wide prior covering possible values of the mean.
$\phi \sim$ Uniform$(-1, 1)$:ensures stationarity of the AR(1) process.
$\sigma \sim$ Cauchy$(0, 5)$:a common heavy-tailed prior for scale parameters, weakly informative.

We run the Stan model with 4 chains and 2000 iterations for each dataset, and summarized the posterior results:

Table 8: Posterior mean, 95% CIs and the number of ef fective posterior samples(phi=0.4)

	mean	2.5%	97.5%	n_eff
mu	4.9056314	4.3155363	5.4839831	3855.854
phi	0.4010326	0.3013691	0.5047824	3634.926
sigma	3.0283043	2.7948251	3.2761219	3949.068

Table 9: Posterior mean, 95% CIs and the number of ef fective posterior samples(phi=0.98)

	mean	2.5%	97.5%	n_eff
mu	3.7166188	-0.9894248	8.9243786	2656.724
phi	0.9595402	0.9241078	0.9926098	2154.466
sigma	3.0309008	2.8039001	3.2825061	2250.335

For $\phi = 0.4$, all 95% credible intervals contain the true values,the effective sample sizes (`n_eff` > 3700 for all parameters) are large, indicating good mixing and reliable inference.

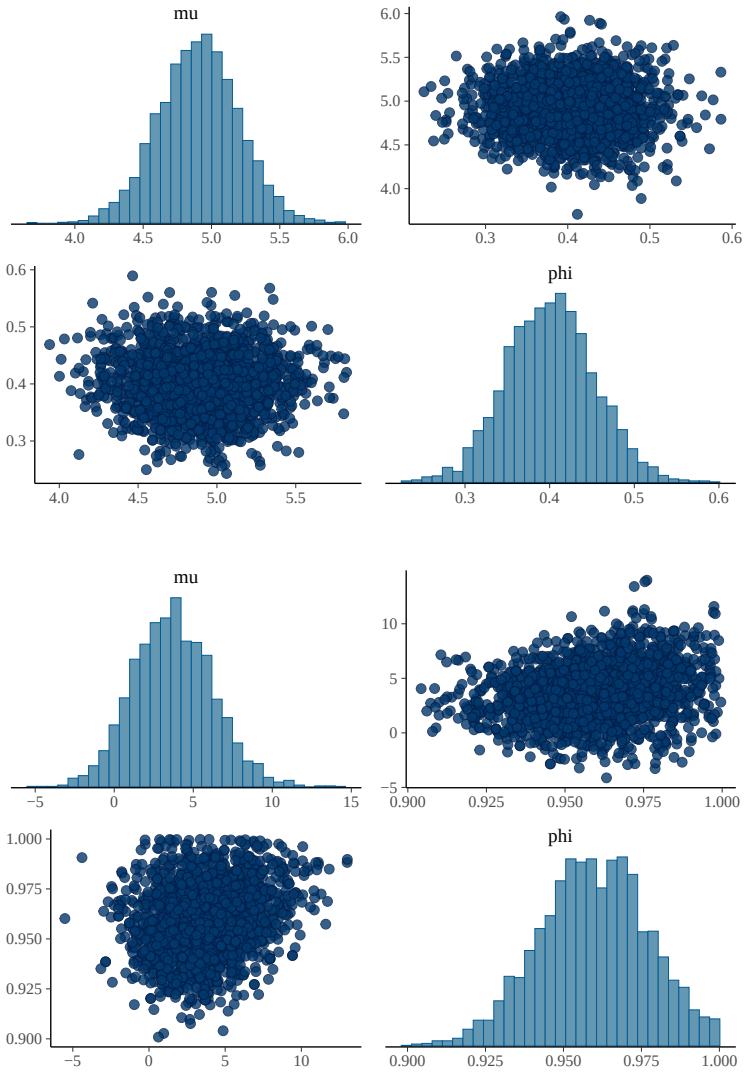
For $\phi = 0.98$, the posterior mean for ϕ is close to the true value, and CI is tight. For μ,although the posterior mean is 5.84, the 95% CI is very wide: [0.13, 11.39], showing high uncertainty.For σ,its posterior mean (2.95) is again close to the true value, but with wider uncertainty compared to the $\phi = 0.4$ case.The effective sample sizes (`n_eff`: 2200–2400) are lower than the first case but still acceptable.

While we can estimate ϕ and σ reasonably well, estimating μ becomes much harder when ϕ is close to 1, due to the process becoming more persistent and the data offering less information about the long-run mean.

Convergence Diagnostics and Joint Posterior Plots
We evaluate sampler convergence using two standard diagnostics:
- $\hat{R}$ (Gelman-Rubin statistic): all parameters had $\hat{R} \approx 1$, indicating convergence.
- Effective Sample Size (n_eff):
- For $\phi = 0.4$: n_eff > 3700 for all parameters.

- For $\phi = 0.98$: n_eff between 2200–2400.
The chains mix well in both cases.Slightly lower `n_eff` when $\phi = 0.98$, reflecting slower mixing due to high autocorrelation in the data.

We use `mcmc_pairs()` to plot the joint posterior samples of μ and ϕ:



For the dataset with low autocorrelation ($\phi = 0.4$), the posterior samples are well-mixed, showing no significant correlation between parameters. This indicates efficient sampling and good convergence.

In contrast, the high-autocorrelation dataset ($\phi = 0.98$) yields a highly correlated posterior between μ and ϕ,with long, stretched shapes in the joint distribution plots. This is expected as the process approaches non-stationarity, making posterior inference more challenging and potentially slowing convergence.

```
#(a)

#AR(1)-process
set.seed(12345)
AR1 <- function(mu,phi,sigma2,t){
  points <- numeric(t)
```

```
  points[1] <- mu
  for (i in 2:t){
    xt <- mu+phi*(points[i-1]-mu)+rnorm(1,mean = 0,sd=sqrt(sigma2))
    points[i] <- xt
  }
return(points)
}

#result1 <- AR1(mu=5,phi=0.8,sigma2 = 9,t=300)

phi_vals <- c(-0.9,-0.2,0,0.2,0.9)
par(mfrow=c(2,3))
set.seed(12345)
for (phi in phi_vals) {
  x <- AR1(mu=5,phi=phi,sigma2 = 9,t=300)
  plot(x,type="l",main=paste("phi = ",phi),
       xlab=" Time ",
       ylab=expression(x[t]),
       col="#4E79A7",
       lwd=1)

}

#(b)
library("rstan")

#phi=0.4
data1 <- AR1(mu=5,phi=0.4,sigma2=9,t=300)

stan_data1 <- list(
  T=length(data1),
  x=data1
)

stan_model_code <- "
data{
  int<lower=1> T; #time points
  vector[T] x;    #sequence
}

parameters {
  real mu;
  real<lower=-1, upper=1> phi;
  real<lower=0> sigma;  #! using sigma instead of sigma2, more stable
}


model{
  mu~normal(0,100);
  phi ~ uniform(-1, 1);
  sigma~cauchy(0,5);  #heavy tail
  x[1]~normal(mu,sigma);
  for (t in 2:T) {
```

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

fit1 <- stan(
  model_code = stan_model_code,
  data=stan_data1,
  iter=2000,
  chains=4,
  seed=12345
)



#phi=0.98
data2 <- AR1(mu=5,phi=0.98,sigma2=9,t=300)

stan_data2 <- list(
  T=length(data1),
  x=data2
)

fit2 <- stan(
  model_code = stan_model_code,
  data=stan_data2,
  iter=2000,
  chains=4,
  seed=12345
)

# mean of posterior, 95%ci,num of effective samples
sum_fit1 <-summary(fit1)$summary
fit1_df <- sum_fit1[1:3,c("mean","2.5%","97.5%","n_eff")]
print(fit1_df)

sum_fit2 <-summary(fit2)$summary
fit2_df <- sum_fit2[1:3,c("mean","2.5%","97.5%","n_eff")]
print(fit2_df)

#evaluate the convergence of tha samplers
#print(fit1, pars=c("mu", "phi", "sigma"))
#print(fit2, pars=c("mu", "phi", "sigma"))



#plot the joint posteriors of mu and phi
par(mfrow=c(1,1))
pairs(fit1, pars=c("mu", "phi"))
pairs(fit2, pars=c("mu", "phi"))

library(bayesplot)
```

```r
posterior1 <- as.array(fit1)
mcmc_pairs(posterior1, pars = c("mu", "phi"))


posterior2 <- as.array(fit2)
mcmc_pairs(posterior2, pars = c("mu", "phi"))
```