lab3

William Bergekrans & Shahin Salehi

2021-05-11

1. Gibbs Sampling for a Normal Model

We assume that $ln(y_i), ..., ln(y_n) \sim^{iid} N(\mu, \sigma^2)$ and both μ and σ^2 are unknown. $\mu \sim N(\mu_0, \tau_0^2)$ and $\sigma^2 \sim Inv - \chi^2(v_o, \sigma_0^2)$.

a) We shall implement a Gibbs sampler that simulates from the posterior $p(\mu, \sigma^2 | ln\ y_1, ..., ln\ y_n)$. A conditionally conjugate prior is:

$$\mu \sim N(\mu_0, \tau_0^2)$$

$$\sigma^2 \sim \text{Inv-}\chi^2(v_0, \sigma_0^2)$$

The full conditional posteriors are:

$$\mu | \sigma^2, x \sim N(\mu_n, \tau_n^2)$$

$$\sigma^2 | \mu, x \sim \text{Inv-}\chi^2\left(v_n, \frac{v_0\sigma_0^2 + \sum_{i=1}^n (x_i - \mu)^2}{n + v_0}\right)$$

Where

$$\frac{1}{\tau_n^2} = \frac{n}{\sigma^2} + \frac{1}{\tau_0^2}$$

$$v_n = v_0 + n$$

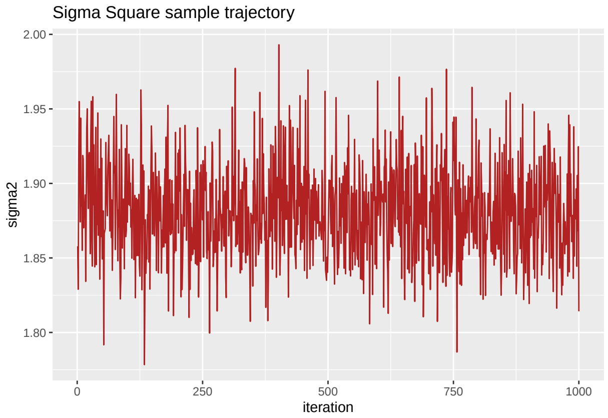
and

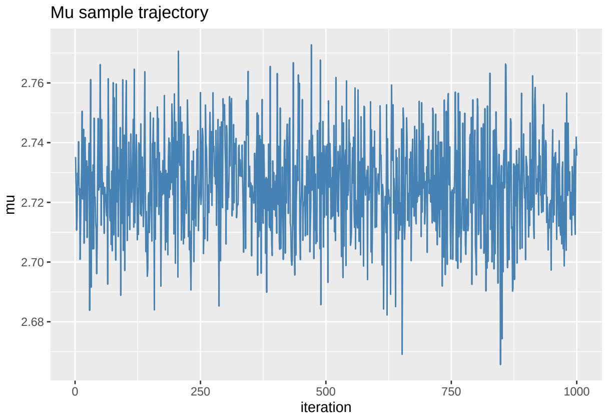
$$\mu_n = W\bar{x} + (1 - W)\mu_0$$

$$W = \frac{\frac{n}{\sigma^2}}{\frac{n}{\sigma^2} + \frac{1}{\tau_0^2}}$$

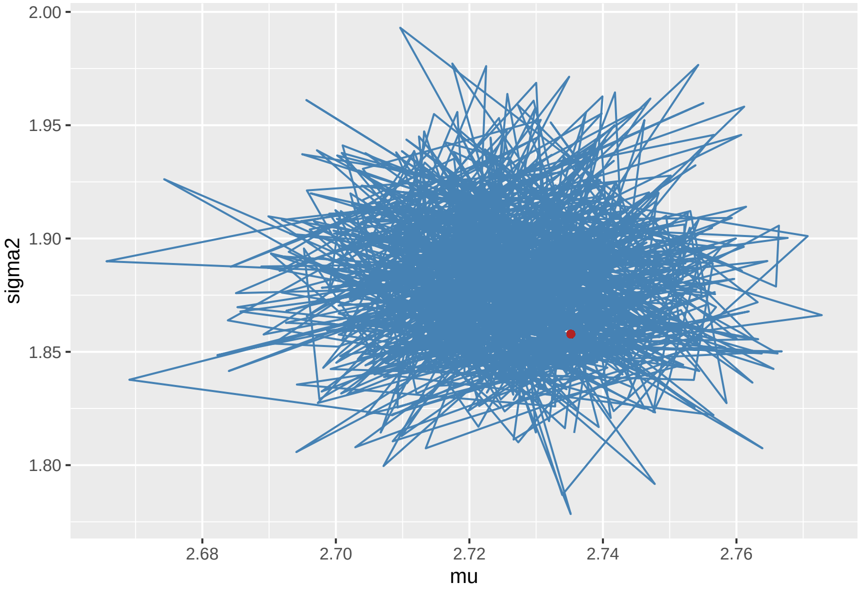
k is the number of imaginary observations determined in our prior and n is the number of observations. v is the degrees of freedom. μ_n, the posterior mean, is a weighted average of prior and data information.

The trajectories of the sampled markov chains are as follows:





As the plot shows both samples seem to converge well, as the samples follow a straight line and seem to have a similar variance for all iterations.



The steps seem to be centered around a distribution and does not seem to care much for the starting point, which is good.

IF_mu	IF_sigma
1.386632	1.075427

Both the inefficiency factors are very low which indicates that the sampling is efficient and that there is low correlation between the samples made. Inefficiency factors bellow 10 are generally accepted which makes these values very good.

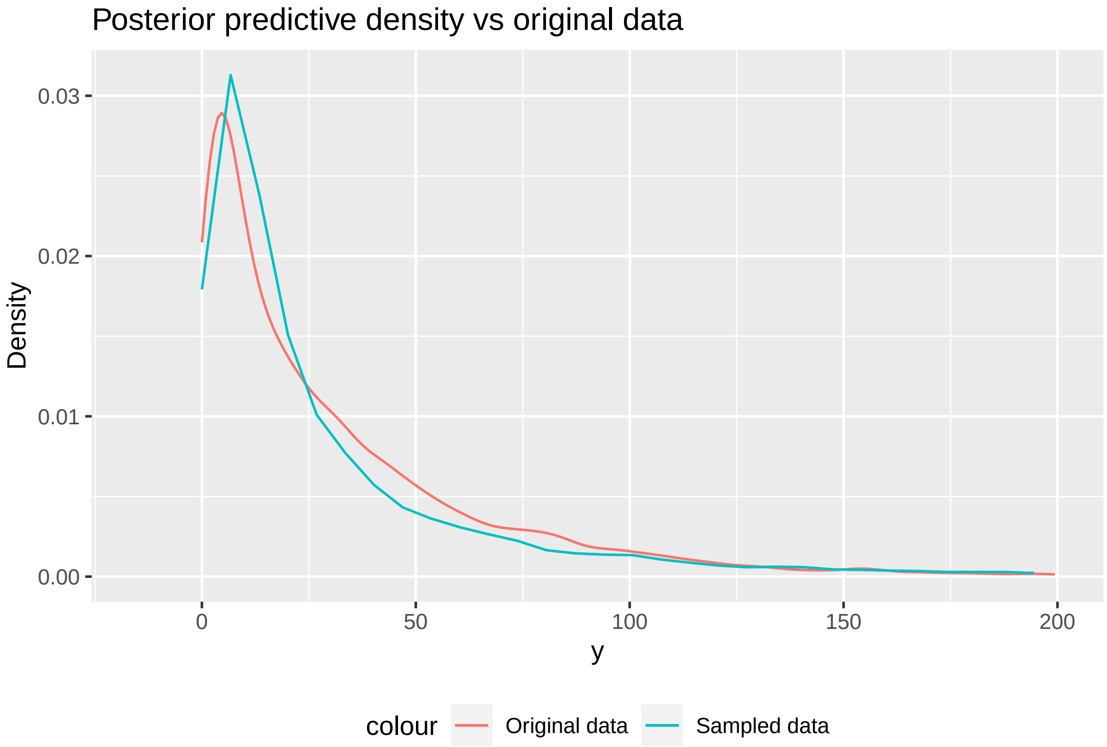
Code for 1a:

```r
# Import rainfall data
data <- read.table("rainfall.dat")
names(data) <- "y"
data$ln_y <- log(data$y) # Natural logarithm of observations

# Code for 1a
# Set prior parameters
u0 <- 0
tao2_0 <- 1
v0 <- 5
sigma2_0 <- 1

n <- length(data$y)
vn <- v0 + n

# Initialize Gibbs parameter, sigma2
sigma2 <- 1
nIter <- 1000 # Set the number of Gibbs iterations
gibbsDraws <- matrix(0, nIter, 2) # To store draws in

# Carry out the Gibbs sampling
for (i in 1:nIter) {
  # Variables that depends on sigma2:
  w <- (n/sigma2) / ((n/sigma2)+(1/tao2_0)) # Weight parameter W
  un <- w*mean(data$ln_y) + (1-w)*u0 # Weighted mean
  tao2_n <- 1/((n/sigma2)+(1/tao2_0))
  mu <- rnorm(1, un, sqrt(tao2_n)) # Draw mu from a normal dist.
  # Next draw sigma, which depends on mu
  scalingFactor <- (v0*sigma2_0 + sum((data$ln_y-mu)^2))/(n+v0)
  sigma2 <- rinvchisq(1, vn, scalingFactor) # Draw sigma square.
  #Store the draws
  gibbsDraws[i, 1] <- mu
  gibbsDraws[i, 2] <- sigma2
}

df1 <- data.frame(
  mu = gibbsDraws[,1],
  sigma2 = gibbsDraws[,2],
  iteration = seq(1,nIter, 1)
)

# Plot the trajectories
ggplot(df1, aes(x=iteration)) +
  geom_line(aes(y=sigma2), color="firebrick") +
  ggtitle("Sigma Square sample trajectory") + theme_set(theme_grey())

ggplot(df1, aes(x=iteration)) +
  geom_line(aes(y=mu), color="steelblue") +
  ggtitle("Mu sample trajectory")

# Calculate the inefficiency factor
rho_mu <- acf(gibbsDraws[,1], plot = FALSE)
rho_sigma <- acf(gibbsDraws[,2], plot = FALSE)

IF_mu <- 1 + 2*sum(rho_mu$acf[-1])
IF_sigma <- 1+ 2*sum(rho_sigma$acf[-1])

# Plot the steps
ggplot(df1, aes(x=mu,y=sigma2)) +
  geom_path(color="steelblue", direction="vh", stat = "identity") +
  geom_point(aes(x=mu[1], y = sigma2[1]), color="firebrick")
```

1b) Now we use our sampled mean and variation to predict rainfall. We know that:

$$lny_1, ..., lny_n|\mu, \sigma^2 \overset{iid}{\sim} N(\mu, \sigma^2)$$

We want to plot the density for $p(\tilde{y}|y_1, ..., y_n)$ which we get by taking the exponential of a draw from the above distribution for lny_i. The results are as follows:



The density is cut at 0 because the rainfall cannot be negative. We see that the two graph are quite similar which indicates that our model has a good fit. The sampled data density has a longer tail, which is not visible in the graph for visibility reasons. All in all the posterior density agrees quite well with the sampled data.

Code for 1b:

```
# Code for 1b
yDraws <- rnorm(data$y, gibbsDraws[,1], sqrt(gibbsDraws[,2]))

y_dens_original <- density(data$y, from = 0)
y_dens_draws <- density(exp(yDraws), from = 0)

df2 <- data.frame(
  org_den_x <- y_dens_original$x,
  org_den_y <- y_dens_original$y,
  samp_den_x <- y_dens_draws$x,
  samp_den_y <- y_dens_draws$y
)

ggplot(df2) +
  geom_line(aes(org_den_x, org_den_y, color="Original data")) +
  geom_line(aes(samp_den_x, samp_den_y, color="Sampled data")) +
  xlim(-15,200) +
  xlab("y") +
  ylab("Density") +
  ggtitle("Posterior predictive density vs original data") +
  theme(legend.position = "bottom")
```

2. Metropolis Random Walk for Poisson Regression

We use the given Poisson Regression model:

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

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. We have a dataset with 1000 observations from ebay coin auctions.

a) We first obtain a maximum likelihood estimator of β for a poisson model using the function glm(). The optimal β coefficients are as follows:

	beta_labels	
(Intercept)	B0	1.0724
PowerSeller	B1	-0.0205
VerifyID	B2	-0.3945
Sealed	B3	0.4438
Minblem	B4	-0.0522
MajBlem	B5	-0.2209
LargNeg	B6	0.0707
LogBook	B7	-0.1207
MinBidShare	B8	-1.8941

As we can see in the table the β closest to 0 are B1, B4 and B6. Indicating that the covariates with the smallest impact are if the seller is a powerseller, if there are minimal faults with the coin, or if there is a lot of negative feedback on the seller. The most significant covariates are the minBidShare followed by if the coin is in a sealed letter and if the seller is verified by eBay.

Code for 2a:

```
# Code for 2a
ebayData <- read.table("eBayNumberOfBidderData.dat", header = TRUE)
mle <- glm(nBids~., ebayData[-2], family = "poisson") # Train poisson model

beta_labels <- c("B0", "B1", "B2", "B3", "B4", "B5", "B6", "B7", "B8")
beta_table <- cbind(beta_labels, round(mle$coefficients,4)) # For printing
```

2b) Now we want to do a Bayesian analysis of the Poisson regression. We shall use Zellner's g-prior for β which is $\beta \sim N\left(0, (X^TX)^{-1}\right)$. The posterior equals the sum of the prior and likelihood which we shall use to calculate $\tilde{\beta}$ and $J_y^{-1}(\tilde{\beta})$.

In Poisson Regression $\lambda(X) = \beta X$ and the log likelihood function is:

$$l(\beta) = \sum_{i=1}^{n} Y_i(\beta X) - exp(\beta X) - ln(Y_i!)$$

We creaded a function to calculate the log posterior, by adding the log prior and log likelihood for β. When we run the optim() function we recieve the following $\tilde{\beta}$ and $J_y^{-1}(\tilde{\beta})$:

Beta values
1.0724
-0.0205
-0.3945
0.4438
-0.0522
-0.2209
0.0707
-0.1207
-1.8941

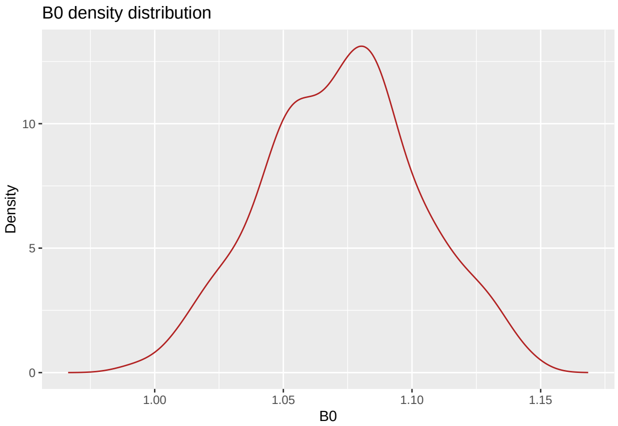
B0	B1	B2	B3	B4	B5	B6	B7	B8
0.0009	-0.0007	-0.0003	-0.0003	-0.0004	-0.0003	-0.0005	1e-04	0.0011
-0.0007	0.0014	0.0000	-0.0003	0.0001	-0.0002	0.0003	1e-04	-0.0006
-0.0003	0.0000	0.0085	-0.0008	-0.0001	0.0002	0.0003	-3e-04	-0.0004
-0.0003	-0.0003	-0.0008	0.0026	0.0004	0.0005	0.0003	-1e-04	-0.0001
-0.0004	0.0001	-0.0001	0.0004	0.0036	0.0003	0.0001	1e-04	-0.0001
-0.0003	-0.0002	0.0002	0.0005	0.0003	0.0084	0.0004	-1e-04	0.0003
-0.0005	0.0003	0.0003	0.0003	0.0001	0.0004	0.0032	-3e-04	-0.0001
0.0001	0.0001	-0.0003	-0.0001	0.0001	-0.0001	-0.0003	8e-04	0.0010
0.0011	-0.0006	-0.0004	-0.0001	-0.0001	0.0003	-0.0001	1e-03	0.0051

If we compare our $\tilde{\beta}$ with the results from 2a we see that they are the same, which they should be as the model in 2a also simulates from the posterior for β. In our approximate posterior these values are our mean values for the β draws.

The approximate posterior density is:

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

We can now draw approximate β values from this distribution. For example, the density function for β_0 looks as follows:



As expected β_0 is quite well centered around the mean value which is 1.0724 both from using the glm() function and optimizing our own log posterior function for β.

Code for part 2b:

```r
# Code for 2b
# Prior values
x <- as.matrix(ebayData[,2:10]) # Covariates, including constant
y <- ebayData$nBids # Target variable
u0 <- rep(0, dim(x)[2]) # Mean values for Zellner prior
sigmaZellner <- 100 * solve(t(x) %*% x) # Sigma value for Zellner prior
betas <- mle$coefficients

# Log posterior function
logPost <- function(beta, x, y, mean, sigma) {
  # Draw prior density
  prior <- dmvnorm(x = beta, mean = mean, sigma = sigma)
  # Calculate the log likelihood
  logLike <- sum(y * (beta%*%t(x)) - exp(beta %*% t(x)) - log(factorial(y)))
  if (is.infinite(logLike)){logLike <- -20000} # Avoid infinity
  return (prior + logLike)
}

init_values <- rep(0, dim(x)[2])
# Optimize the beta and J parameters
opt_res <- optim(init_values, fn=logPost, gr = NULL, x, y, u0, sigmaZellner, method=c("BFGS"), control=

kable(round(opt_res$par,4), col.names = c("Beta values"))

kable(round(-solve(opt_res$hessian), 4), col.names = c("B0", "B1", "B2", "B3", "B4", "B5", "B6", "B7",

beta_draws <- rmvnorm(600, opt_res$par, -solve(opt_res$hessian))
b0_dens <- density(beta_draws[,1])
df2b <- data.frame(x = b0_dens$x, y = b0_dens$y)

ggplot(df2b, aes(x, y)) +
  geom_line(color="firebrick") +
  ggtitle("B0 density distribution") + xlab("B0") + ylab("Density")
```

2c) The proposal density for the Metropolis algorithm is:

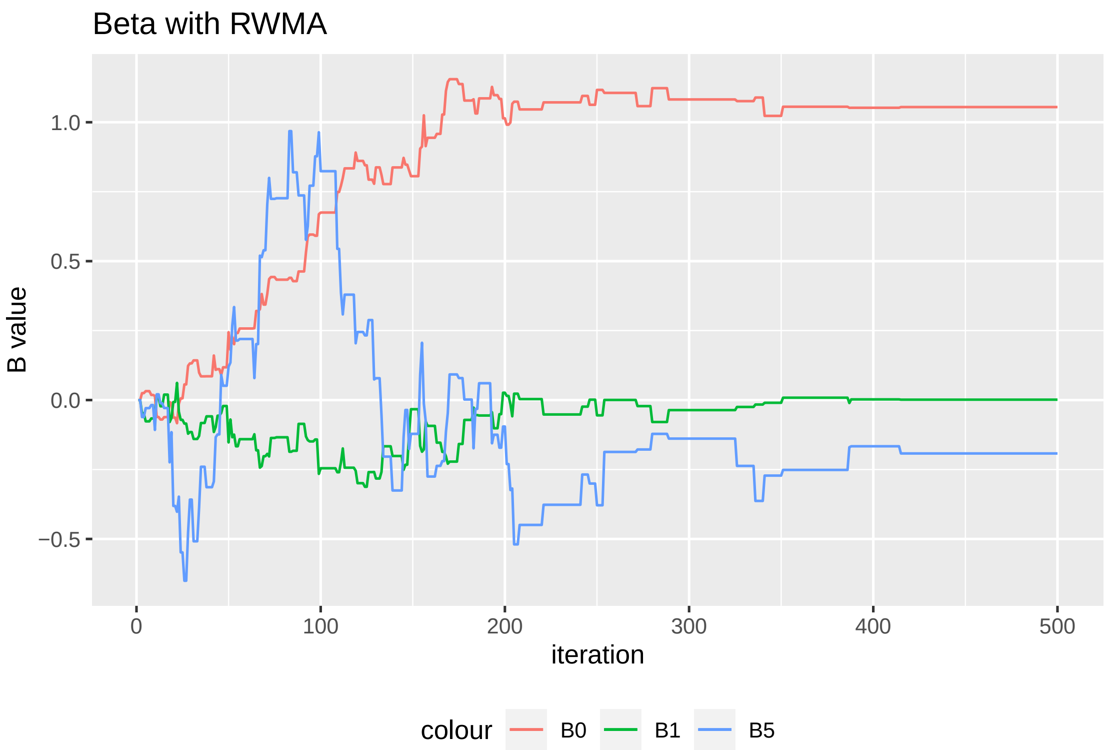
$$\theta_p|\theta^{(i-1)} \sim N(\theta^{(i-1)}, c \cdot \Sigma)$$

Using our random walk metropolis function with a c=2 and 500 iterations, we get:

```
## [1] "Acceptance ratio: 0.21"
```

The acceptance ratio should generally be between 20-25% which it is, indicating that the algorithm is effective.

In the following graph we see the beta values for B0, B1 and B5 for the different iterations:



As we can see the three plotted variables all seem to stabilize after around 250 iterations and after that there are a lot more rejected draws. This means that the algorithm's started to converge and the values are around the beta values obtained earlier using glm(), which they should be!

Code for 2c:

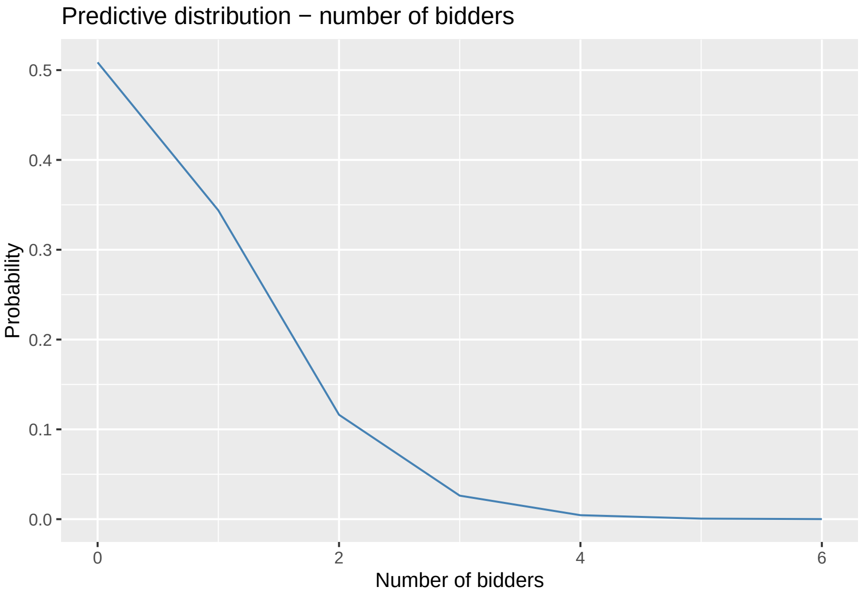
```r
# Code for 2c
RWMASampler <- function(logPostFunc, c, Sigma, nIter, ...) {
  accepted <- 0 # Variable to keep track of accepted proposals
  thetas <- matrix(rep(0, dim(Sigma)[1]), nIter, dim(Sigma)[1]) # Parameter to sample
  for (i in 2:nIter) {
    thetas[i,] <- rmvnorm(1, thetas[i-1,], c * Sigma) # Draw proposal
    alpha <- min(1, exp(logPostFunc(thetas[i,], ...) - logPostFunc(thetas[i-1,], ...)))
    rNum <- runif(1, 0, 1) # Draw a number from a uniform dist (0,1)
    if (alpha < rNum) {
      thetas[i,] = thetas[i-1,] # Reject the draw
    } else {
      accepted <- accepted +1 # Accept the draw, draw already stored in thetas
    }
  }
  print(paste0("Acceptance ratio: ", accepted/nIter))
  return(thetas) # Return the result
}

# Set parameters
c <- 2
nIter <- 500
coVar <- -solve(opt_res$hessian) # Use results from 2b for sigma

betas <- RWMASampler(logPost, c, coVar, nIter, x, y, u0, sigmaZellner)
```

```r
df2c <- data.frame(
  B0 = betas[,1],
  B1 = betas[,2],
  B5 = betas[,6],
  iteration = seq(1, nIter, 1)
)

ggplot(df2c, aes(iteration)) +
  geom_line(aes(y=B0, color="B0")) +
  geom_line(aes(y=B1, color="B1")) +
  geom_line(aes(y=B5, color="B5")) +
  ggtitle("Beta with RWMA") + ylab("B value") +
  theme(legend.position = "bottom")
```

2d) The Poisson model for y that we use is:

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

We use the β draws from 2c to predict on y. The predictive distribution is calculated using the ppois() function and the results is as follows:



As we can see the probability for 0 bidders in this particular auction is quite high, at 0.51.

Code for 2d:

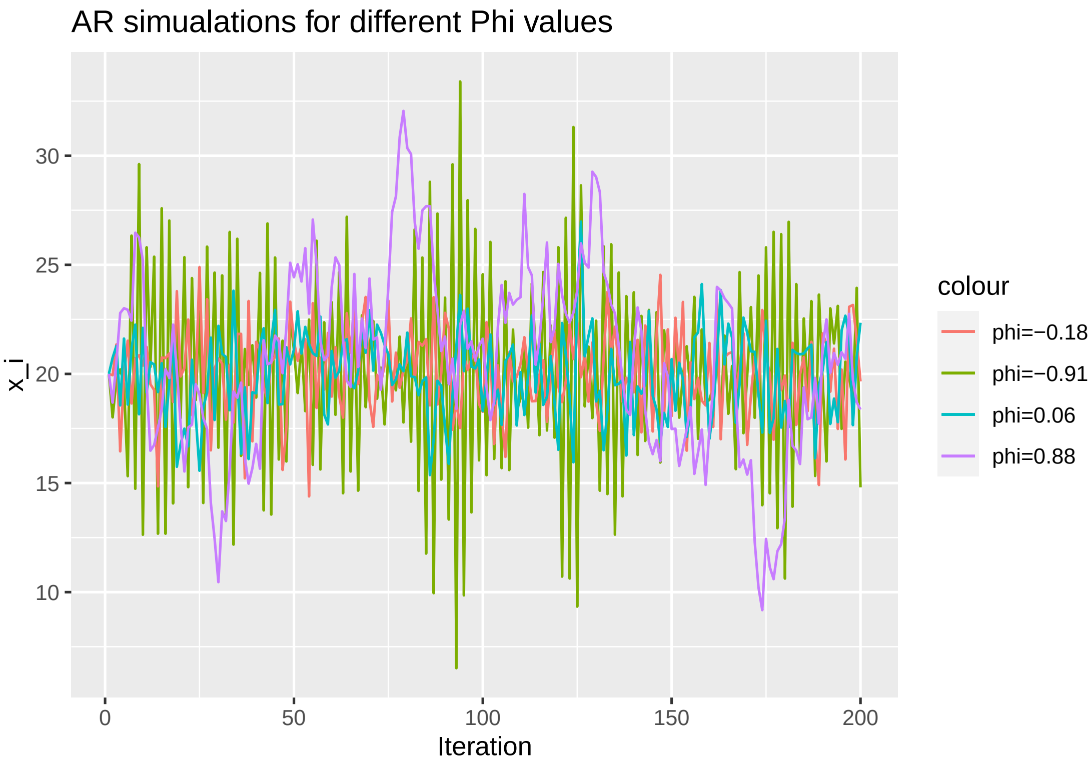
```r
# Code for 2d
xNew <- c(1, 1, 1, 1, 0, 1, 0, 1, 0.7) # Set x, with intercept constant
lastBetas <- betas[dim(betas)[1], ] # Use the last beta draws from RWMA.
lambda <- exp(t(xNew) %*% lastBetas)
yDen <- dpois(0:6, lambda = lambda) # Generate predictive density
df2d <- data.frame(x=0:6, y=yDen)

ggplot(df2d, aes(x, y)) +
  geom_line(color="steelblue") +
  ggtitle("Predictive distribution - number of bidders") +
  labs(x="Number of bidders", y="Probability")
```

3. Time Series Models in Stan

The AR(1)-process we shall simulate data from is:

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

We initialize the model with the given parameters and $x_1 = \mu$. We run the AR process simulation for sampled $\phi \sim U(-1, 1)$ and in the following plot we see four of the results.



In the plot we can see that when ϕ get closer to 0, the simulations are more stable and centered around the mean value (20). As we get values further away from 0 there are a lot more oscillations and the process explores more values, from looking at the formula for the AR(1) process, a larger ϕ give more weight to the previous observation, resulting in a more spread out graph with higher variation. This aligns well with how our graph looks for higher $|\phi|$ values.

Code for 3a:

```r
# Code for 3a
u <- 20
sigma2 <- 4
t <- 200

ARSimulation <- function(u, sigma2, phi, t) {
  x = rep(0, t)
  x[1] = u
  for (i in 2:t) {
    eps = rnorm(1, 0, sqrt(sigma2)) # Draw error
    x[i] = u + phi*(x[i-1]-u) + eps
  }
  return(x)
}

set.seed(123)
phis <- runif(10, -1,1) # Draw 10 samples for phi
AR_phi <- matrix(0, nrow=t, ncol=length(phis))
for (i in 1:length(phis)) {
  AR_phi[,i] <- ARSimulation(u, sigma2, phis[i], t)
}

df3a <- data.frame(
  x = 1:t,
  phi6 = AR_phi[,6],
  phi3 = AR_phi[,3],
  phi7 = AR_phi[,7],
  phi5 = AR_phi[,5])

ggplot(df3a, aes(x=x)) +
  geom_line(aes(y=phi6, color="phi=-0.91")) +
  geom_line(aes(y=phi3, color="phi=-0.18")) +
  geom_line(aes(y=phi7, color="phi=0.06")) +
  geom_line(aes(y=phi5, color="phi=0.88")) +
  ggtitle("AR simualations for different Phi values") +
  labs(x="Iteration", y="x_i")
```

3b) Now we write a Stan model for estimating the μ, ϕ and σ^2 values. We do this on two data sets that are sampled using the AR function written in 3a, using two different values for ϕ.

For the first data set with $\phi = 0.3$ the mean for the simulations using the function in 3a is 19.633. When $\phi = 0.9$ the same mean is 19.454. When we sample from the posterior of μ, σ^2, ϕ using Stan code we get:

Table 5: x data set

	x
u	19.629
phi	0.380
sigma	1.751

Table 6: y data set

	x
u	19.588
phi	0.918
sigma	1.945

The 95% credible intervals for the parameters are:

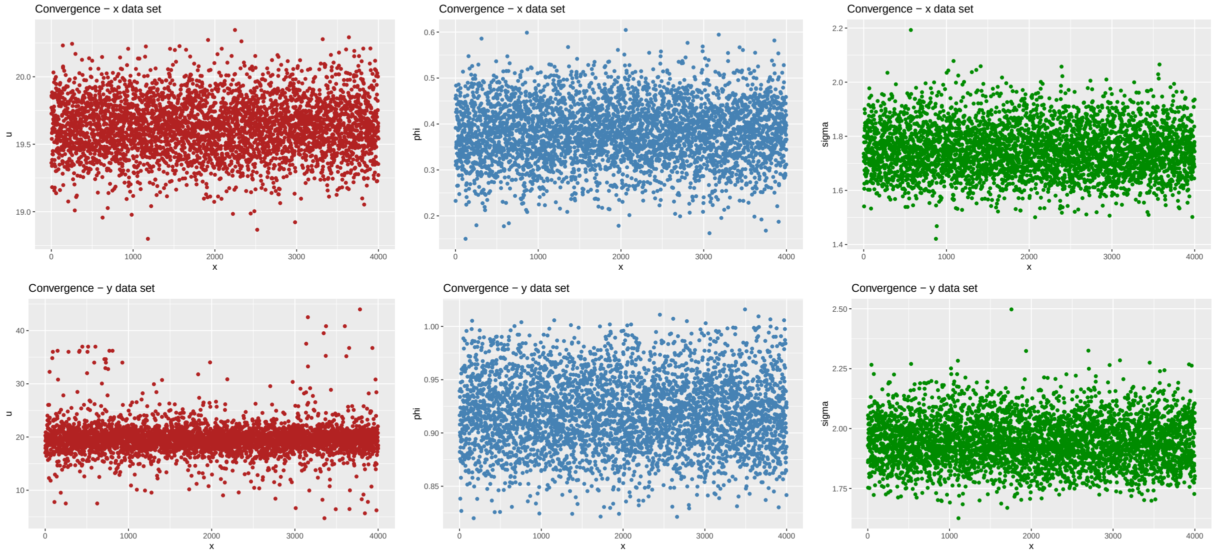
Table 7: x data set

	2.5%	97.5%
u	19.235	20.032
phi	0.251	0.509
sigma	1.587	1.944

Table 8: y data set

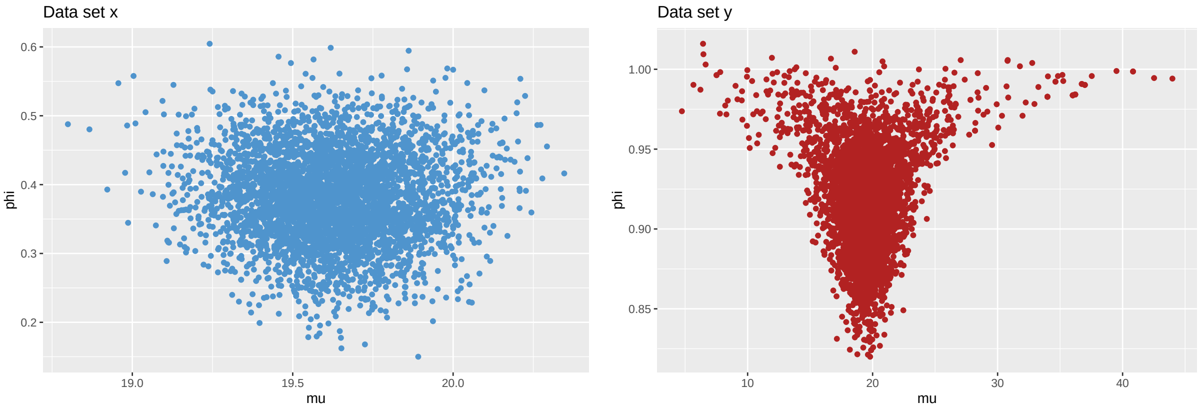
	2.5%	97.5%
u	14.544	25.093
phi	0.855	0.986
sigma	1.768	2.148

As we can see the interval for data set x is a lot narrower, due to the ϕ being closer to 0. However, Compared to the original values used for generating our x and y data sets for the parameters the model seem to estimate the true values quite well.



The sampler seem to converge very well for both data set x and y. The variation is even for all iterations and centered around the same mean.

The joint distribution for μ and ϕ are:



We see that the two joint posterior distributions for the two data sets look very different. We see that for data set x the distribution is very even for all directions. In comparison data set y has much higher variation for μ for ϕ values closer to 1. It is clear that when ϕ approaches 0.95 and above the variation for μ increase exponentially.

Code for 3b:

```
# Code for 3b
xt <- ARSimulation(u, sigma2, 0.3, t)
yt <- ARSimulation(u, sigma2, 0.9, t)

stanModel <- '
  data {
    int<lower=0> T;
    vector[T] x;
  }
  parameters {
    real u;
    real phi;
    real<lower=0> sigma;
  }
  model {
  for (t in 2:T)
    x[t] ~ normal(u + phi * (x[t-1] - u), sigma);
  }
'

xData <- list(T = t, x = xt)
yData <- list(T = t, x = yt)
# Execute models with stan
xStan <- stan(model_code = stanModel, data = xData, verbose = FALSE)
yStan <- stan(model_code = stanModel, data = yData, verbose = FALSE)

x_summary <- summary(xStan, pars = c("u", "phi", "sigma"), probs=c(0.025,0.975))$summary
y_summary <- summary(yStan, pars = c("u", "phi", "sigma"), probs=c(0.025,0.975))$summary

# Evaluate convergence
xStanDraws <- extract(xStan)
yStanDraws <- extract(yStan)

xFrame <- data.frame(
  x = 1:length(xStanDraws$u),
  u = xStanDraws$u,
  sigma = xStanDraws$sigma,
  phi = xStanDraws$phi
)

yFrame <- data.frame(
  x = 1:length(yStanDraws$u),
  u = yStanDraws$u,
  sigma = yStanDraws$sigma,
  phi = yStanDraws$phi
)

# X data set
ggplot(xFrame, aes(x=x)) +
  geom_point(aes(y=u), color="firebrick") + ggtitle("Convergence - x data set")

ggplot(xFrame, aes(x=x)) +
  geom_point(aes(y=phi), color="steelblue") + ggtitle("Convergence - x data set")

ggplot(xFrame, aes(x=x)) +
  geom_point(aes(y=sigma), color="green4") + ggtitle("Convergence - x data set")

# y data set
ggplot(yFrame, aes(x=x)) +
  geom_point(aes(y=u), color="firebrick") + ggtitle("Convergence - y data set")

ggplot(yFrame, aes(x=x)) +
  geom_point(aes(y=phi), color="steelblue") + ggtitle("Convergence - y data set")

ggplot(yFrame, aes(x=x)) +
  geom_point(aes(y=sigma), color="green4") + ggtitle("Convergence - y data set")

dfx <- data.frame(
  x=xStanDraws$u,
  y=xStanDraws$phi
)

dfy <- data.frame(
  x=yStanDraws$u,
  y=yStanDraws$phi
)

ggplot(dfx, aes(x=x, y=y)) +
  geom_point(color="steelblue3") + ggtitle("Data set x") +
  labs(x="mu", y="phi")

ggplot(dfy, aes(x=x, y=y)) +
  geom_point(color="firebrick") + ggtitle("Data set y") +
  labs(x="mu", y="phi")
```