TDDE07 Bayesian Learning Lab 3

Erik Jareman - erija971
Ludvig Knast - ludkn080

1a

Gibbs sampler:

```
# --- Lab 3 / Assignment 1 ---
nDraws = 1000

# --- a ---
data = readRDS("Precipitation.rds")
n = length(data)
logData = log(data)
logDataMean = mean(logData)

my0 = 1
sigma0 = 1
sigma = 1 # initial value updated later
v0 = 1
tao0 = 1
gibbsSamples = matrix(0, nDraws, 2)

for (i in 1:nDraws) {
  # draw next my
  w = (n/sigma) / ((n/sigma) + (1/tao0))
  tao = 1 / ((n/sigma0) + (1/sigma))
  myN = w * logDataMean + (1 - w)*my0
  my = rnorm(1, myN, tao)
  gibbsSamples[i, 1] = my

  # draw next sigma
  chiDraw = rchisq(1, df=(v0+n)) # v=v0+n
  sigmaVar = (v0*sigma0 + sum((logData-my)^2)) / (n + v0)
  sigma = n * sigmaVar / chiDraw
  gibbsSamples[i, 2] = sigma
}
```

Inefficiency factors:

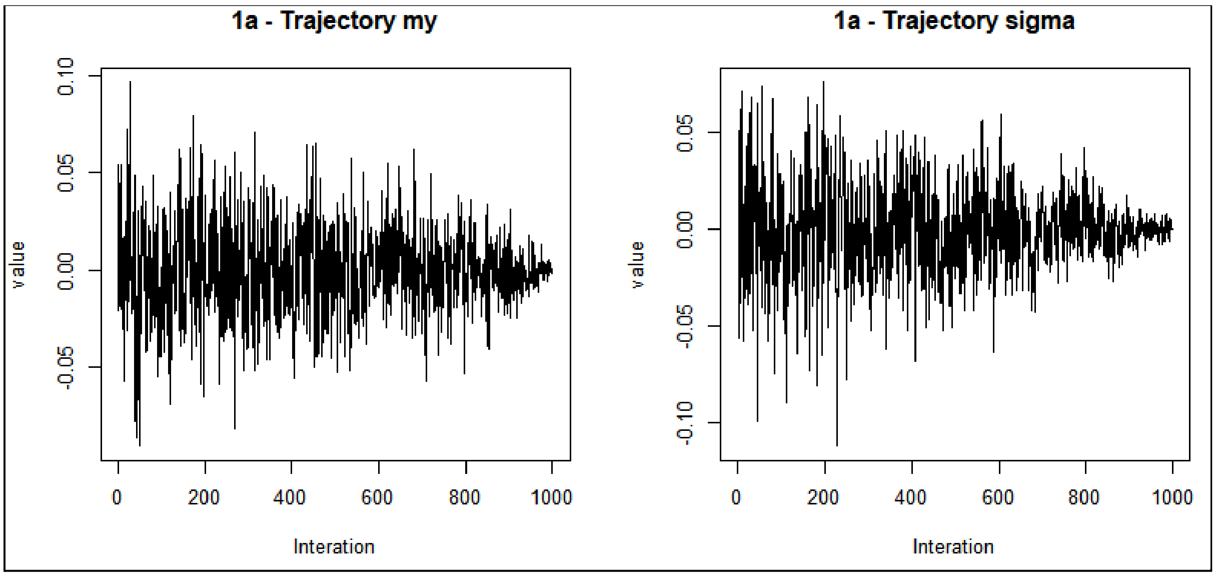
IFmy = 1.2 (depends on run)

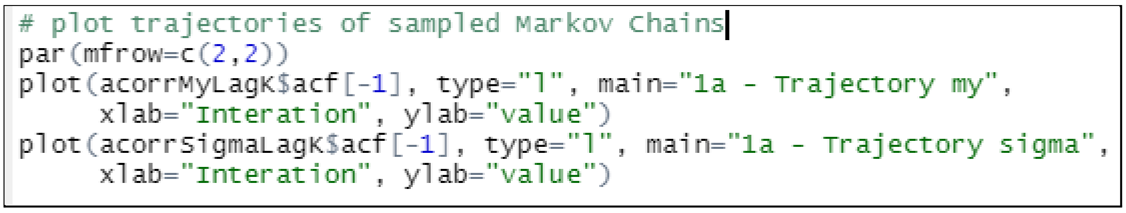
IFsigma = 0.9 (depends on run)

Inefficiency factors (code):

```
# Inefficiency factor, IF=1+2*sum(rho(k)), where rho(k) is the sample
# autocorrelation at lag k calculated from sample values
acorrMyLagK = acf(gibbsSamples[,1], lag=nDraws, pl=FALSE)
acorrSigmaLagK = acf(gibbsSamples[,2], lag=nDraws, pl=FALSE)
MyIF = 1 + 2*sum(acorrMyLagK$acf[-1][1:30])
SigmaIF = 1 + 2*sum(acorrSigmaLagK$acf[-1][1:30])
```

Trajectories of sampled markov chains (with code):

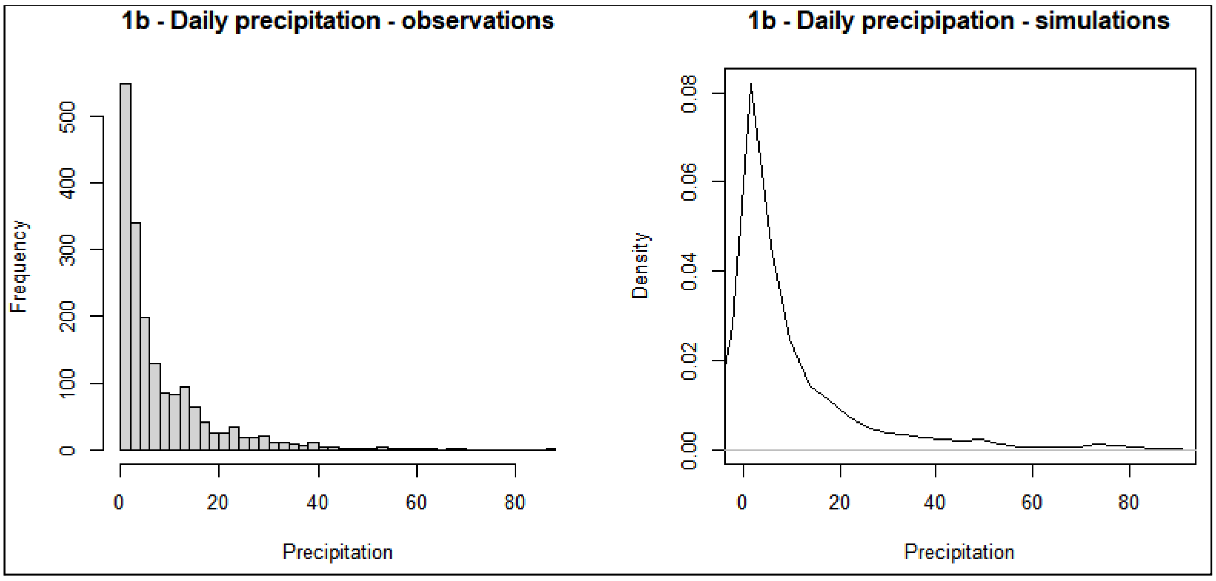


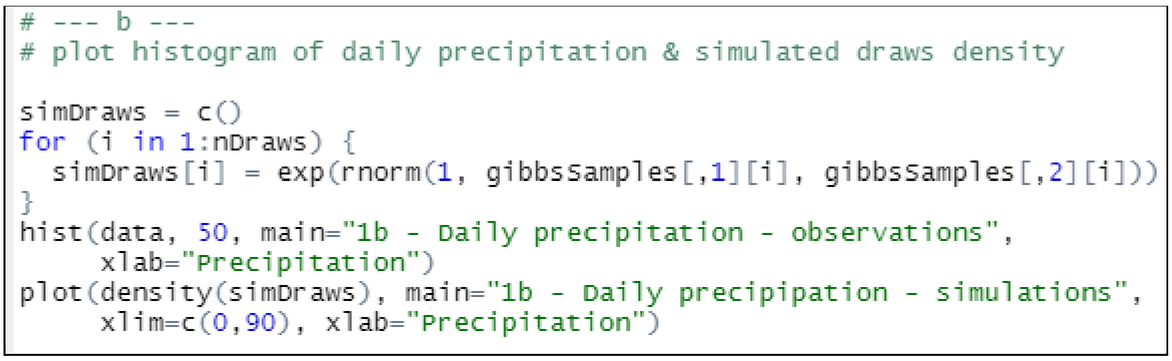


```
# plot trajectories of sampled Markov Chains
par(mfrow=c(2,2))
plot(acorrMyLagK$acf[-1], type="l", main="1a - Trajectory my",
     xlab="Interation", ylab="value")
plot(acorrSigmaLagK$acf[-1], type="l", main="1a - Trajectory sigma",
     xlab="Interation", ylab="value")
```

1b

Observed daily precipitation and simulated draws density (with code):

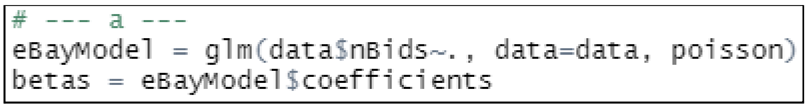




```r
# --- b ---
# plot histogram of daily precipitation & simulated draws density

simDraws = c()
for (i in 1:nDraws) {
  simDraws[i] = exp(rnorm(1, gibbsSamples[,1][i], gibbsSamples[,2][i]))
}
hist(data, 50, main="1b - Daily precipitation - observations",
     xlab="Precipitation")
plot(density(simDraws), main="1b - Daily precipipation - simulations",
     xlim=c(0,90), xlab="Precipitation")
```

2a

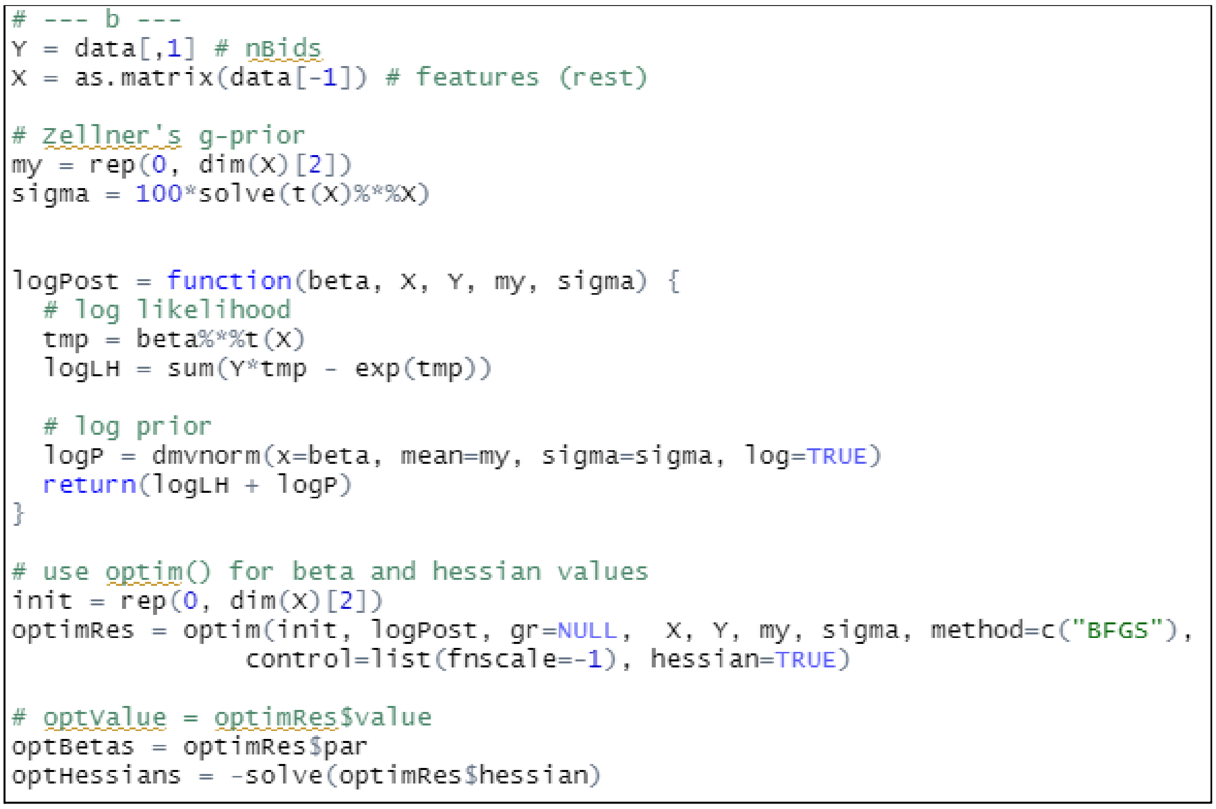
Maximum likelihood estimator of beta in the poisson regression model for the ebay data:



```
# --- a ---
eBayModel = glm(data$nBids~., data=data, poisson)
betas = eBayModel$coefficients
```

2b

Bayesian analysis of the Poisson regression.



```
# --- b ---
Y = data[,1] # nBids
X = as.matrix(data[-1]) # features (rest)

# Zellner's g-prior
my = rep(0, dim(X)[2])
sigma = 100*solve(t(X)%*%X)


logPost = function(beta, X, Y, my, sigma) {
  # log likelihood
  tmp = beta%*%t(X)
  logLH = sum(Y*tmp - exp(tmp))

  # log prior
  logP = dmvnorm(x=beta, mean=my, sigma=sigma, log=TRUE)
  return(logLH + logP)
}

# use optim() for beta and hessian values
init = rep(0, dim(X)[2])
optimRes = optim(init, logPost, gr=NULL,  X, Y, my, sigma, method=c("BFGS"),
               control=list(fnscale=-1), hessian=TRUE)

# optValue = optimRes$value
optBetas = optimRes$par
optHessians = -solve(optimRes$hessian)
```

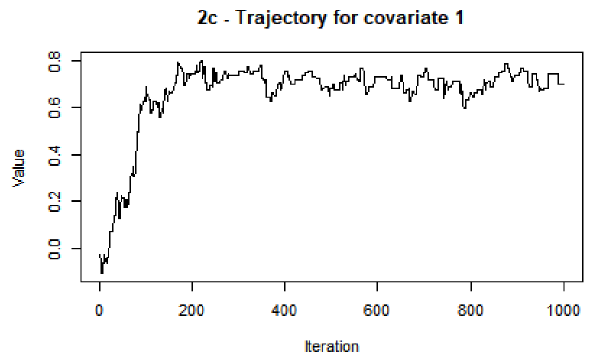
2c
General metropolis function:

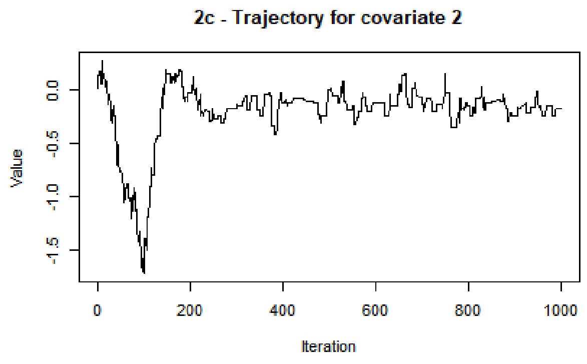
```
runMetropolis = function(nRuns, c, beta, hessian, postFunc, ...) {

  acceptedDraws = matrix(0, nrow=nRuns, ncol=length(beta))
  prevProposal = beta

  for (i in 1:nRuns) {
    # draw from proposal density
    proposal = rmvnorm(1, prevProposal, hessian*c)

    # draw from supplied posterior density
    drawRatio = min(1, exp(postFunc(proposal, ...) - postFunc(prevProposal, ...)))

    # draw a random ratio
    randomRatio = runif(1, 0, 1)

    # save proposal if OK, else stay on same
    if (drawRatio >= randomRatio) {
      prevProposal = proposal
    }
    acceptedDraws[i,] = prevProposal
  }
  return(acceptedDraws)
}
```

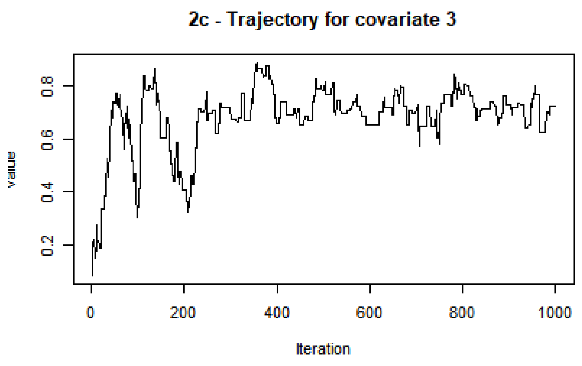
Use the general function with earlier data, and code for convergence plots:

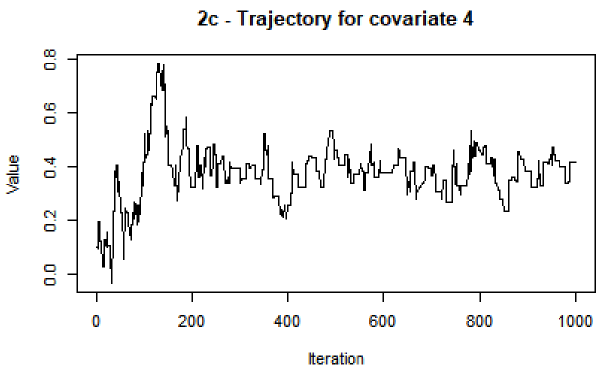
```
# use function to sample from earlier poisson regression for eBay data
nRuns = 1000
metroSample = runMetropolis(nRuns=nRuns, c=1, beta=my, hessian=optHessians,
                            postFunc=logPost, X, Y, my, sigma)

# Assess MCMC convergence by graphical methods
par(mfrow=c(3,3))
for(i in 1:8){
  plot(1:nRuns, type='l', metroSample[,i],
       main=paste(c("2c - Trajectory for covariate", i), collapse= " "),
       xlab="Iteration", ylab="Value")
}
```

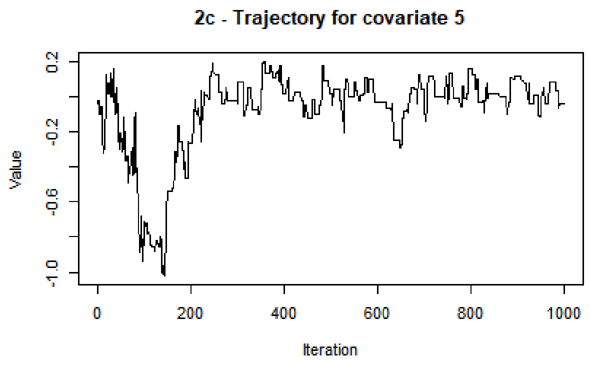
Convergence plots for covariates:

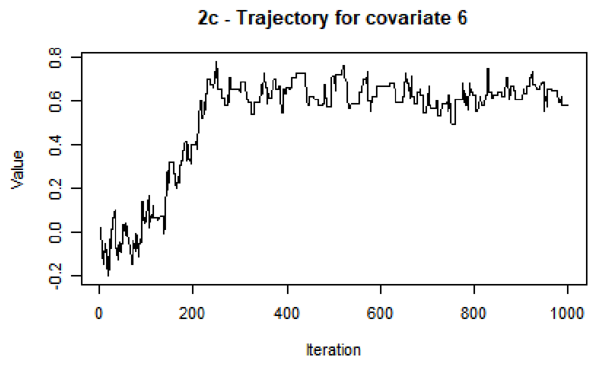


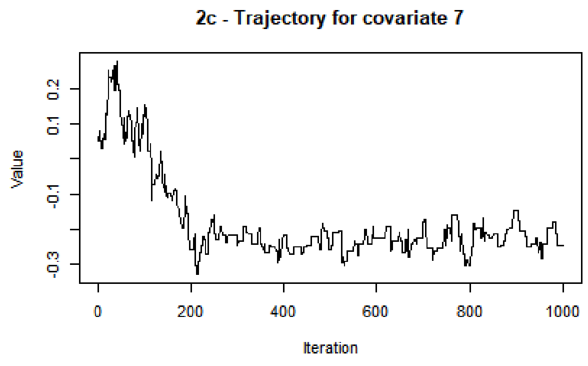


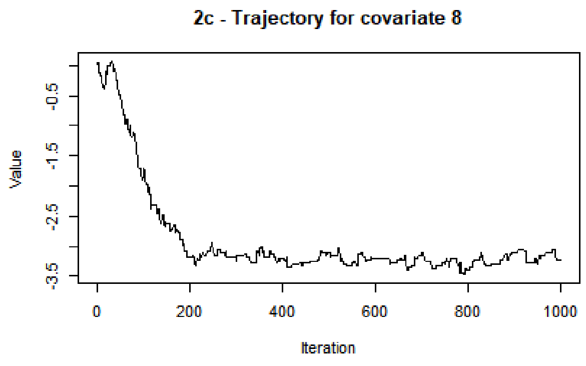






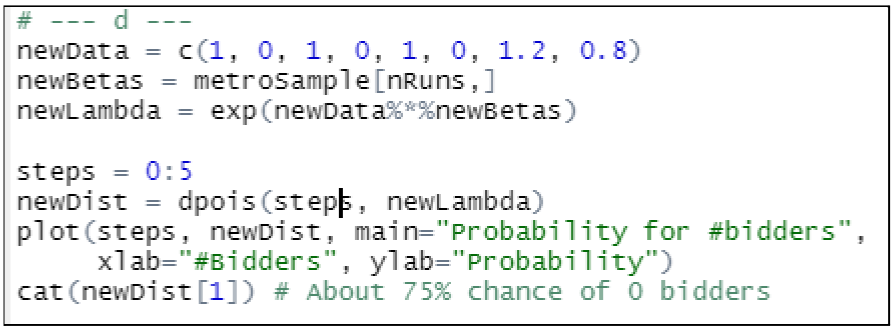






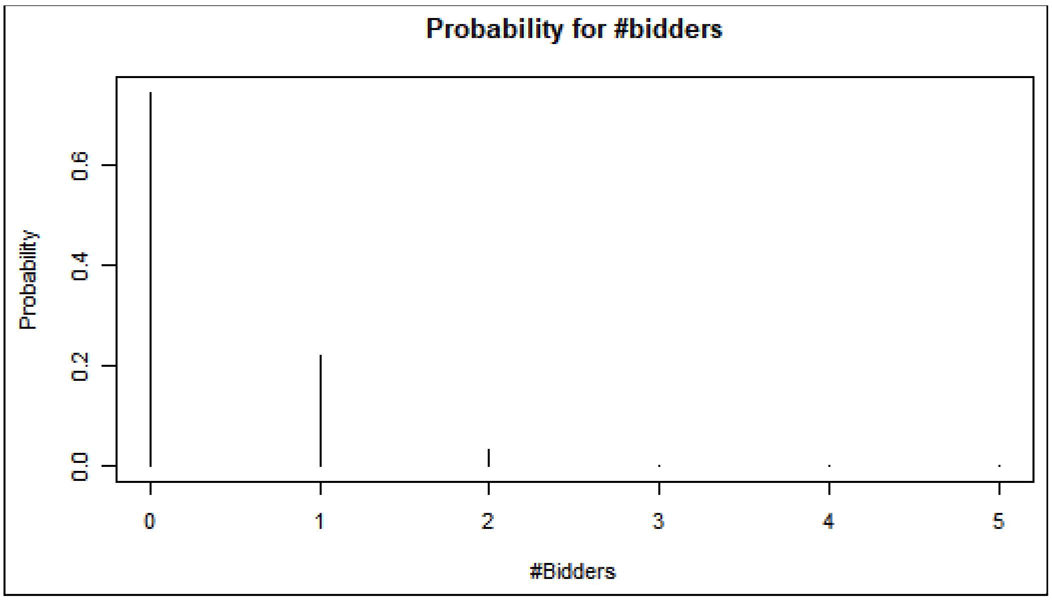
2d

Use MCMC draws to simulate from the predictive distribution, and plot the distribution. The probability of no bidders in the new auction was about 75%.

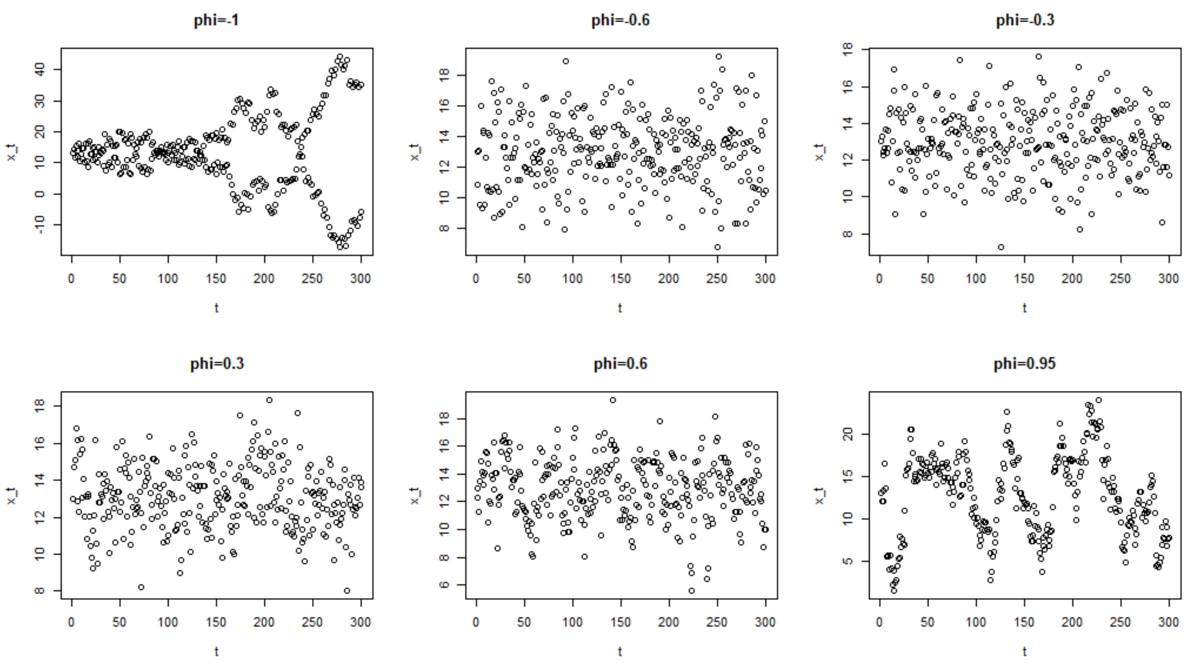


```
# --- d ---
newData = c(1, 0, 1, 0, 1, 0, 1.2, 0.8)
newBetas = metroSample[nRuns,]
newLambda = exp(newData%*%newBetas)

steps = 0:5
newDist = dpois(steps, newLambda)
plot(steps, newDist, main="Probability for #bidders",
     xlab="#Bidders", ylab="Probability")
cat(newDist[1]) # About 75% chance of 0 bidders
```

The plot:



3.
a)



The closer phi is to 0, the more normally distributed the draws are (with mean mu and variance of sigma squared) as well as getting less influenced by previous draws.

```
library(rstan)

mu = 13
sigma2 = 3
T = 300

# AR1 sim function
AR1.sim = function(mu, sigma2, T, phi) {
  x = numeric()
  x[1] = mu
  for(i in 2:T) {
    x[i] = mu + phi * (x[i-1] - mu) + rnorm(1, mean = 0, sd = sqrt(sigma2))
  }
  return(x)
}

# a)
# Simulating draws of x-values with different phis
phis = seq(-1,1,0.05)
nPhis = length(phis)
xVals = matrix(nrow=nPhis, ncol=T, 0)
for(i in 1:nPhis) {
  xVals[i,] = AR1.sim(mu, sigma2, T, phis[i])
}
# results
par(mfrow=c(2,3))
plot(xVals[1,], xlab="t", ylab="x_t", main="phi=-1")
plot(xVals[9,], xlab="t", ylab="x_t", main="phi=-0.6")
plot(xVals[15,], xlab="t", ylab="x_t", main="phi=-0.3")
plot(xVals[27,], xlab="t", ylab="x_t", main="phi=0.3")
plot(xVals[33,], xlab="t", ylab="x_t", main="phi=0.6")
plot(xVals[40,], xlab="t", ylab="x_t", main="phi=0.95")
```

b)
i)
Posterior mean for x (phi=0.3)
x.postMu = 13.21
x.postPhi = 0.20
x.postSigma = 1.75

95% CI for x (phi=0.3)
x.CI.mu = (12.9, 13.5)
x.CI.phi = (0.0896, 0.3148)
x.CI.sigma = (1.62, 1.9)

Number of effective posterior samples x (phi=0.3)
x.nEff.mu = 4125
x.nEff.phi = 3741
x.nEff.sigma = 3894

Posterior mean for y (phi=0.95)
y.postMu = 10.86
y.postPhi = 0.93
y.postSigma = 1.67

95% CI for y (phi=0.95)
y.CI.mu = (6.29, 14.91)
y.CI.phi = (0.89, 0.983)
y.CI.sigma = (1.54, 1.82)

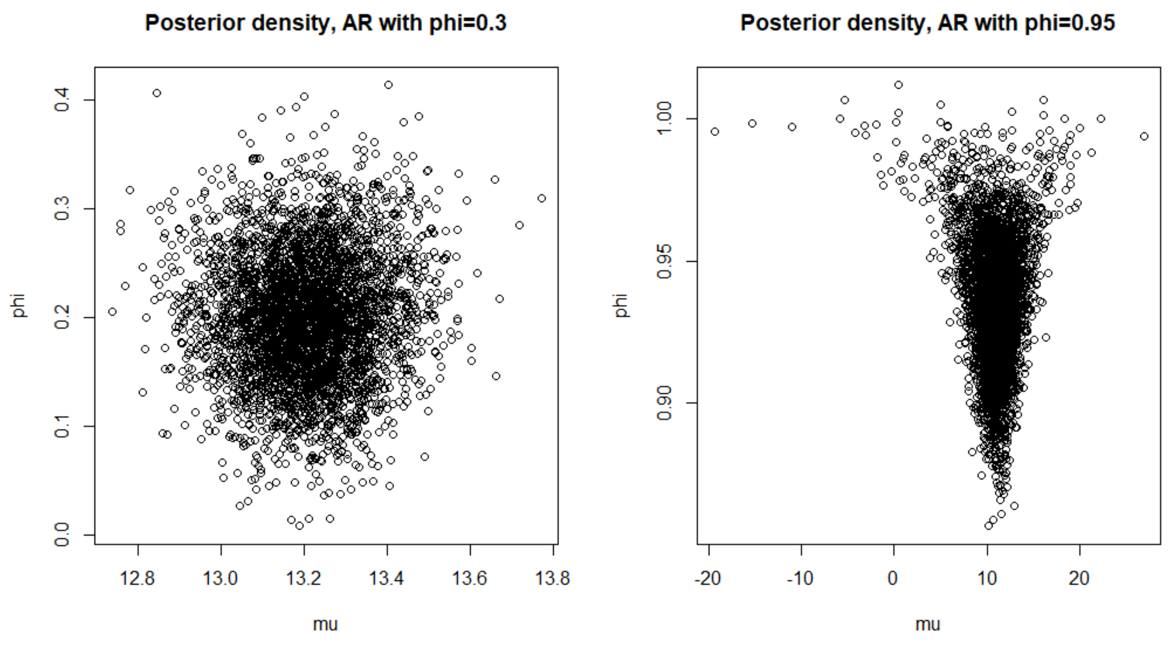
Number of effective posterior samples y (phi=0.95)
y.nEff.mu = 1305
y.nEff.phi = 1322
y.nEff.sigma = 1600

The estimations are quite similar to the true values. The y data (phi 0.95) is estimating phi better whilst the x data (phi 0.3) is estimating mu better!

```
# b)
x.phi_3 = AR1.sim(mu, sigma2, T, phi=0.3)
y.phi_95 = AR1.sim(mu, sigma2, T, phi=0.95)

ARStanModel = 'data {
  int<lower=0> N;
  vector[N] x;
}
parameters {
  real mu;
  real phi;
  real<lower=0> sigma;
}
model {
  for (n in 2:N)
    x[n] ~ normal(mu + phi * (x[n-1] - mu), sigma);
}'

# MCMC
x.fit = stan(model_code=ARStanModel, data=list(x=x.phi_3, N=T))
y.fit = stan(model_code=ARStanModel, data=list(x=y.phi_95, N=T))

# Summaries
x.summary = summary(x.fit)
y.summary = summary(y.fit)

# Get n_eff
x.nEff = x.summary$summary[,"n_eff"]
y.nEff = y.summary$summary[,"n_eff"]

# N effective samples
x.nEff.mu = x.nEff["mu"]
# 4125

x.nEff.phi = x.nEff["phi"]
# 3741

x.nEff.sigma = x.nEff["sigma"]
# 3894

y.nEff.mu = y.nEff["mu"]
# 1305

y.nEff.phi = y.nEff["phi"]
# 1322

y.nEff.sigma = y.nEff["sigma"]
# 1600
# Get params
x.params = extract(x.fit)
y.params = extract(y.fit)

# CI:s
probs = c(0.025,0.975)

x.CI.mu <- apply(as.matrix(x.params$mu), 2, quantile, probs=probs)
# (12.9, 13.5)

x.CI.phi <- apply(as.matrix(x.params$phi), 2, quantile, probs=probs)
# (0.0896, 0.3148)

x.CI.sigma <- apply(as.matrix(x.params$sigma), 2, quantile, probs=probs)
# (1.62, 1.9)

y.CI.mu <- apply(as.matrix(y.params$mu), 2, quantile, probs=probs)
# (6.29, 14.91)

y.CI.phi <- apply(as.matrix(y.params$phi), 2, quantile, probs=probs)
# (0.89, 0.983)

y.CI.sigma <- apply(as.matrix(y.params$sigma), 2, quantile, probs=probs)
# (1.54, 1.82)


# Post means
x.postMean = get_posterior_mean(x.fit)
y.postMean = get_posterior_mean(y.fit)


# x post parameters
x.postMu = x.postMean[1,5]
# 13.21

x.postPhi = x.postMean[2,5]
# 0.20

x.postSigma = x.postMean[3,5]
# 1.75

# y post parameters
y.postMu = y.postMean[1,5]
# 10.86

y.postPhi = y.postMean[2,5]
# 0.93

y.postSigma = y.postMean[3,5]
# 1.67
```

ii)

The data from x (phi = 0.3) displays what looks to be a very normalized data (in respect to mu and phi) and also looks to have a middle point where the data is centered. The data from y (phi = 0.95) however indicates a much smaller spread of phi values but at the same time mu has a much higher spread and the data don't really seem to have a "center".



```r
#ii)
# plot density
par(mfrow=c(1,2))
plot(x=x.params$mu, y=x.params$phi,
     xlab="mu", ylab="phi", main="Posterior density, AR with phi=0.3")
plot(x=y.params$mu, y=y.params$phi,
     xlab="mu", ylab="phi", main="Posterior density, AR with phi=0.95")
```