```r
library("mvtnorm")
library(mvtnorm)

###################### 1 A ############################
# Gibbs sampler for a normal model
# The dataset Precipitation.rds consists of daily records of weather with
rain or snow (in units of mm) from the beginning of
# 1962 to the end of 2008 in a certain area.
# Assume the natural log of the daily precipitation {y1, ..., yn} to be
independent normally distributed,
# ln y1, ...,ln yn|mu, sigma^2 ~ iid N(mu, sigma^2) where both mu and
sigma^2 are unknown.
# Let mu ~ N(mu0, tau0squared) independently of sigma^2 ~ Inv-chi^2(v0,
sigma0^2)

# Implement (code!) a Gibbs sampler that simulates from the joint posterior
p(mu, sigma^2 | ln y1,...,ln yn).
# The full conditional posteriors are given on the slides from Lecture 7.
# Evaluate the convergence of the Gibbs sampler by calculating the
Ineffciency Factors (IFs)
# and by plotting the trajectories of the sampled Markov chains.

data=readRDS("Precipitation.rds")
n = length(data)
log_data = log(data)

# Initial parameters
mu0 = mean(log_data)
sigma0 = 10
v0 = 250
tau0 = 100
nDraws = 10000

# Priors independent --> use Gibbs sampling to sample from the posterior
# posterior distribution of mu and sigma

# Want to sample from our two full conditional posteriors (mu and sigma
respectively)

# Gibbs sampling
gibbsDraws <- matrix(0,nDraws,2)
colnames(gibbsDraws) = c("mu", "sigma")
sigma <- 1 # Initial value for sigma
v_n = v0+n # Degrees of freedom

set.seed(12345)
for (i in 1:nDraws){

  # Calculate posterior parameters
  w = (n/sigma) / ((n/sigma)+1/tau0)
  mu_n = w*mean(log_data) + (1-w)*mu0
  tau_n = 1/((n/sigma)+(1/tau0))

  # Update mu given sigma
  mu <- rnorm(1, mean = mu_n, sd = tau_n)
  gibbsDraws[i,1] <- mu

  # Update sigma given mu
  parameter_squared = (v0*sigma0+sum(log_data-mu)^2)/(n+v0)
  sigma = v_n*parameter_squared/rchisq(1,v_n)
  gibbsDraws[i,2] <- sigma
```

```
}

# Calculate Inefficiency Factors (IFs)
a_Gibbs = acf(gibbsDraws[,1])
IF_Gibbs = 1+2*sum(a_Gibbs$acf[-1])

#Plot Gibbs sampling
plot(1:nDraws, gibbsDraws[,1], type="l", col="orange")
hist(gibbsDraws[,1], col="orange")

a_Gibbs = acf(gibbsDraws[,2])
IF_Gibbs = 1+2*sum(a_Gibbs$acf[-1])

plot(1:nDraws, gibbsDraws[,2], type="l", col="orange")
hist(gibbsDraws[,2], col="orange")


###################### 1 B ##########################
# Plot the following in one figure:
# 1) a histogram or kernel density estimate of the daily precipitation {y1,
..., yn}.
# 2) The resulting posterior predictive density p(ytilde|y1,...,yn) using
the simulated posterior draws from (a)
# How well does the posterior predictive density agree with this data?

# Want to use our sigma and mu from the Gibbs sampling to make draws of y

postY = c(1:nDraws)

set.seed(12345)
postY = rnorm(n=nDraws, mean=exp(gibbsDraws[,1]), sd=exp(gibbsDraws[,2]))

plot(density(data), lwd=2, axes=FALSE)
axis(1)
axis(2)
lines(density(postY), col="orange", lwd=2)



############################## ASSIGNMENT 2 ############################
# Metropolis Random Walk for Poisson regression
# Consider the following Poisson regression model: yi|Beta iid ~
Poisson[exp(t(xi) * Beta)], i = 1,...,n,
# where yi is the count for the ith observation in the sample and xi is the
p-dimensional vector with
# covariate observations for the ith observation. Use the data set
eBayNumberOfBidderData.dat.
# This dataset contains observations from 1000 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), some binary 1/0 vars,
# 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).


############################## 2a ###################################
# Obtain the maximum likelihood estimator of Beta in the Poisson regression
model for the eBay data
```

```r
# [Hint: glm.R, don't forget that glm() adds its own intercept so don't
input the covariate Const].
# Which covariates are significant?

library("mvtnorm")
library("MASS")

data = read.table("eBayNumberOfBidderData.dat", header=TRUE)
n = nrow(data)


# fitting a generalized linear model using our data, nBids depends on all
parameter except Const (the intercept).
# family = poisson to specify Poisson regression model
set.seed(12345)
glmModel <- glm(nBids ~ . - Const, data = data, family = poisson)
glmModel



################################ 2b ####################################
# Let's do a Bayesian analysis of the Poisson regression. Let the prior be
Beta ~ N[0, 100 * (t(X)X)^-1],
# where X is the n × 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 ~ N(Beta-tilde, J_y^-1(Beta-tilde)) , where Beta-tilde is the
posterior mode and
# Jy^-1(Beta-tilde) is the negative Hessian at the posterior mode.
# These 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.).


y = data[,1]
X = as.matrix(data[,2:ncol(data)])

mu0 = as.matrix(c(rep(0,9)))
sigma0 = 100*solve(t(X)%*%X)

# function for calculating the log posterior for the Poisson mode.
LogPostPoisson <- function(betas,y,X,mu,sigma){
  #linPred <- X%*%betas;
  # lambda = exp(x*Beta) is our theta, i.e. the parameter we are interested
in estimating (see L2 Poisson LH)
  # could also call this variable theta to be consistent with course
notations
  lambda = as.matrix(exp(X%*%betas))
  logLikelihood = sum(y*log(lambda) - lambda - log(factorial(y)))
  #if (abs(logLik) == Inf) logLik = -20000; # Likelihood is not finite,
steer the optimizer away from here!
  logPrior <- dmvnorm(t(betas), t(mu), sigma, log=TRUE); #dmvnorm -
Multivariate Normal Density and Random Deviates
  return(logLikelihood + logPrior)
}

# initialize Beta to be 9 zeros, since 9 covariates/variables
initVal <- matrix(0,9,1)
```

```r
# numeric optimization for Beta using above function
set.seed(12345)
OptimRes <-
optim(initVal,LogPostPoisson,gr=NULL,y,X,mu0,sigma0,method=c("BFGS"),contro
l=list(fnscale=-1),hessian=TRUE)

# posterior mode, essentially what is optimized by the optimization.
# posterior mode is same as posterior mean and posterior variance for
multivariate normal distribution
betaTilde = OptimRes$par

# J_y_inv(betaTilde)
Jy_inv_betaTilde = -solve(OptimRes$hessian)
sigma = -solve(OptimRes$hessian)

betaTilde
glmModel$coefficients
Jy_inv_betaTilde



################################ 2c ####################################
# Let's simulate from the actual posterior of Beta 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 theta.
# Let the proposal density be the multivariate normal density mentioned in
Lecture 8 (random walk Metropolis):
# theta_p|theta^(i-1) ~ N(theta^(i-1), c * sigma)
# i.e. above is: theta_proposal | theta previous ~ N()...
#, where sigma = Jy^-1(beta-Tilde) wasobtained 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.
# The note HowToCodeRWM.pdf in Lisam describes how yo can do this in R.
# Now, use your new Metropolis function to sample from the posterior of
Beta in the Poisson regression for the eBay dataset.
# Assess MCMC convergence by graphical methods.


# RWM sampler function
# prevBeta could also be called prevTheta to be more consistent with
lecture notation
RWMSampler = function(c, sigma, prevBeta, logPostFunc, ...) {

  # sample proposal beta
  proposalBeta = rmvnorm(n=1, mean = prevBeta, sigma = c*sigma)
  proposalBeta = t(proposalBeta)

  # compute acceptance probability, using the user selected function
  alpha = min(1, exp(logPostFunc(proposalBeta, ...) - logPostFunc(prevBeta,
...)) )
```

```
  # with probability alpha, set theta(i) = sample proposal, otherwise set
theta(i) = theta(i-1) (i.e. prev beta)
  prob = runif(1)
  if (alpha >= prob) {
    return (proposalBeta)
  }
  return (prevBeta)
}


# initial beta0 (i.e. theta0 in the general case general)
initVal <- matrix(0,9,1)
c = 3
# sigma was generated in b)

#test = RWMSampler(c, sigma, initVal, LogPostPoisson, y, X, mu0, sigma0)

nDraws = 10000
posteriorDraws = matrix(nrow=9, ncol=nDraws)
# For the first draw, use initVal
set.seed(12345)
posteriorDraws[,1] = RWMSampler(c, sigma, initVal, LogPostPoisson, y, X,
mu0, sigma0)
# for draw 2 - nDraws
set.seed(12345)
for (i in 2:(nDraws)) {
  posteriorDraws[,i] = RWMSampler(c, sigma, (posteriorDraws[,i-1]),
LogPostPoisson, y, X, mu0, sigma0)
}

# compare with approximation from b) and plot deviance
glmEstimation = glmModel$coefficients
par(mfrow=c(3,3))
for (betaIndex in 1:9) {
  plot(c(1:nDraws), abs(posteriorDraws[betaIndex,]-
glmEstimation[betaIndex]), type="l", col="red",
  xlab = "draw", ylab="deviance true and approximated", main =
paste("Absolute deviance between actual and approximated Beta", betaIndex))
}



############################### 2d ####################################
# 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?

# dont forget to add 1 for the intercept
x= c(1, 1, 0, 1, 0, 1, 0, 1.2, 0.8)

# for each of our posterior draws, perform a draw from the poisson
distribution with lambda = exp(x*Beta)
# this draw represents the estimated number of bids
set.seed(12345)
nrBids = rpois(nDraws, lambda=exp(x%*%posteriorDraws))

#nrBids = c(1:nDraws)
#for (i in 1:nDraws) {
#  nrBids[i] = rpois(1, lambda=exp(x%*%posteriorDraws[,i]))
```

```r
#}

par(mfrow=c(1,1))
hist(nrBids)

#count the probability the the nr of bids is 0
probNoBids = sum(nrBids==0)/nDraws
probNoBids



############################### ASSIGNMENT 3 ############################
# Time series models in Stan

################################ 3a ###################################
# Write a function in R that simulates data from the AR(1)-process xt = mu
+ phi(xt-1 - mu) + epslion_t, epsilon_t ~N(0, sigma^2),
# for given values of mu, phi and sigma^2. Start the process at x1 = mu and
then simulate values for xt for t = 2, 3,...,T
# and return the vector x1:T containing all time points. Use my = 13,
sigma^2 = 3 and T = 300 and look at some different realizations
# (simulations) of x1:T for values of phi between -1 and 1 (this is the
interval of phi where the AR(1)-process is stationary).
# Include a plot of at least one realization in the report. What effect
does the value of phi have on x1:T ?

library(rstan)

mu = 13
sigma2 = 3
T = 300

# function that simulate values for x2, x3, ..., xT given a phi
ARSim = function(phi) {
  x = c(1:T)
  x[1] = mu
  for (i in 2:T) {
    # the rnorm() term corresponds to epsilon.
    x[i] = mu + phi * (x[i-1] - mu) + rnorm(n=1, mean = 0, sd =
sqrt(sigma2))
  }
  return(x)
}


# the different phi values to be used to get different simulations
phiVals = seq(-1, 1, 0.1)

# perform the simulation for the different phis using the ARSim function
xSimulations = matrix(ncol = T, nrow = length(phiVals))
set.seed(12345)
for (i in 1:length(phiVals)) {
  x = ARSim(phiVals[i])
  xSimulations[i,] = x
}

par(mfrow=c(3,3))
plot(xSimulations[1,], type="l", ylab = "x", main = "Traceplot, phi = -1")
plot(xSimulations[2,], type="l", ylab = "x", main = "Traceplot, phi = -
0.9")
```

```r
plot(xSimulations[5,], type="l", ylab = "x", main = "Traceplot, phi = -
0.6")
plot(xSimulations[8,], type="l", ylab = "x", main = "Traceplot, phi = -
0.3")
plot(xSimulations[11,], type="l", ylab = "x", main = "Traceplot, phi = 0")
plot(xSimulations[14,], type="l", ylab = "x", main = "Traceplot, phi =
0.3")
plot(xSimulations[17,], type="l", ylab = "x", main = "Traceplot, phi =
0.6")
plot(xSimulations[20,], type="l", ylab = "x", main = "Traceplot, phi =
0.9")
plot(xSimulations[21,], type="l", ylab = "x", main = "Traceplot, phi = 1")


# alternative for more efficient plotting
par(mfrow=c(3,3))
plot(xSimulations[1,], type="l", ylab = "x", main = "Traceplot, phi = -1")
for (i in 1:(length(phiVals)/3)) {
  print(3*i)
  plot(xSimulations[3*i,], type="l", ylab = "x", main = paste("Traceplot,
phi = ", phiVals[3*i]))

}

par(mfrow = c(1,1))

############################### 3b ####################################
# Use your function from a) to simulate two AR(1)-processes, x1:T with phi
= 0.2 and y1:T with phi = 0.95.
# Now, treat your simulated vectors as synthetic data, and treat the values
of mu, phi and sigma^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 mu and phi. Comments?


# Stan model which has uninformative priors, taken from Stan documentation
for AR(1)-process
StanModel = '
data {
  int<lower=0> N;
  vector[N] y;
}
parameters {
  real mu;
  real phi;
  real<lower=0> sigma;
}
model {
```

```
  for (n in 2:N)
    y[n] ~ normal(mu + phi * (y[n-1] - mu) , sigma);
}
'

# x: phi = 0.2
x = ARSim(0.2)
data_x = list(N=T, y=x)
set.seed(12345)
#default warmup (1000), default iterations (2000), default chains = 4
fit_x = stan(model_code = StanModel, data=data_x)

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

# Extract posterior samples
postDraws_x <- extract(fit_x)

# Do automatic traceplots of all chains
traceplot(fit_x)

# plot(postDraws_x$mu, type="l",ylab="mu",main="Traceplot") # Traceplots of
the first chain
# pairs(fit_x) # Bivariate posterior plots


# y: phi = 0.95
y = ARSim(0.95)
data_y = list(N=T, y=y)
set.seed(12345)
fit_y =stan(model_code = StanModel, data=data_y)

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

# Extract posterior samples
postDraws_y <- extract(fit_y)

# Do automatic traceplots of all chains
traceplot(fit_y)

# plot(postDraws_y$mu, type="l",ylab="mu",main="Traceplot") # Do traceplots
of the first chain
#pairs(fit_y) # Bivariate posterior plots

# plot joint posterior of mu and phi
plot(postDraws_x$mu, postDraws_x$phi,ylab="phi", xlab="mu")
plot(postDraws_y$mu, postDraws_y$phi,ylab="phi", xlab="mu")
```