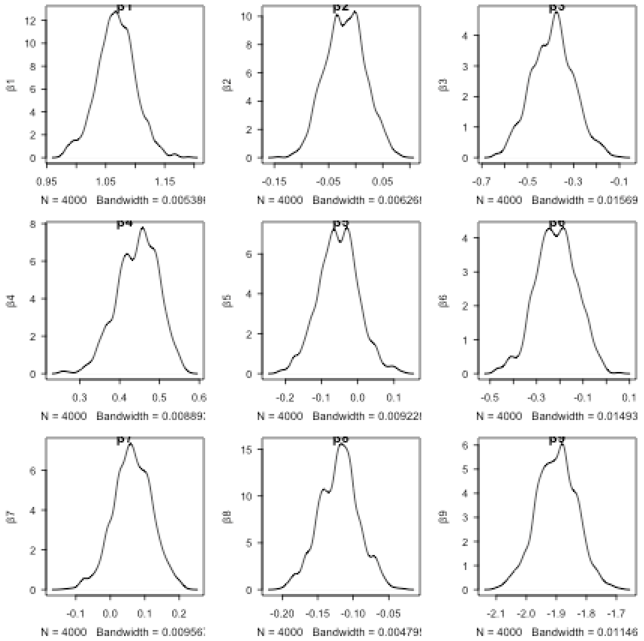
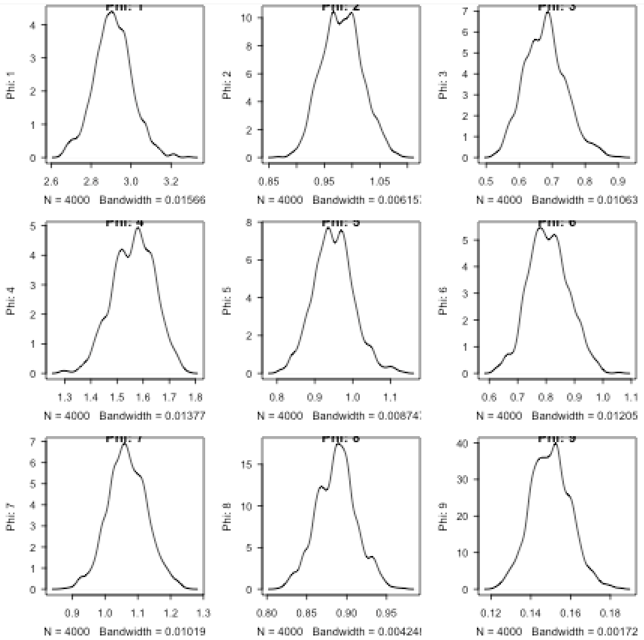
```
########################## Lab 4 ##########################
## MCMC
## Metropolis algorithm
## Poisson regression
## Hessian
## Mode
## Maximum likelihood estimator
## Program general functions

# Task 1: Poisson regression - the MCMC way
# a) Obtain the maximum likelihood estimator of β in the Poisson regression model
# Find significant covariates
# b) Bayesian analysis of the Poisson regression
# Find mode and hessian of Beta
# c) Simulate from the actual posterior of β using the Metropolis algorithm and
# compare with the approximate results in b)
# Program general function
# d) Use MCMC draws from c) to simulate from the predictive distribution of
# the number of bidders in a new auction

####################################################################################################

library(mvtnorm)

############# Task 1 #############
# Consider the following Poisson regression model:
#   y_i | β ~ Poisson[ exp(t(x_i) * β ) ], i = 1, ..., n

dataset <- read.table("eBayNumberOfBidderData.dat", header = TRUE)
X <- dataset[, -1]
Y <- dataset[, 1]

# a)
# Obtain the maximum likelihood estimatior of β

# Fit Poisson model
nBids.fitted <- glm(formula = nBids ~.-Const,
               data = dataset,
               family = poisson)

# Find significant coefficients
# 99.9%: (Intercept), VerifyID, Sealed, LogBook, MinBidShare
# 95%: MajBlen
nBids.summary <- summary(nBids.fitted)

# b)
# Bayesian Analysis of the Poisson regression
# Zellner's prior:
#     β ~ N(0, 100 * (t(X)*X)^(-1)), X: n x p covariate matrix
#
# Approximate posterior density:
#     β | y ~ N[ B_hat, Jy(β_hat)^(-1) ]

# Functions
logPostPois <- function(β, mu, sigma, y, X) {
  n <- length(Y)

  # log likelihood
  log.likelihood <- sum(y * X %*% β - exp(X %*% β))

  # If log.likelihood -Inf or Inf
  if (abs(log.likelihood) == Inf) log.likelihood = -20000;

  # log prior
  # OBS: Use dmvnorm!! You want the likelihood of the betas!
  log.prior <- dmvnorm(x = β, mean = mu, sigma = sigma, log = TRUE)

  return (log.likelihood + log.prior)
}

# Setup
X.matrix <- as.matrix(X)
beta_init <- as.matrix(10 * rep(1, dim(X.matrix)[2]))
sigma <- 100 * solve(t(X.matrix)%*%(X.matrix))
mu <- rep(0, dim(sigma)[2])
optim.res <- optim(par = beta_init,
                   fn = logPostPois,
                   gr = NULL,
                   mu = mu,
                   sigma = sigma,
                   y = Y,
                   X = X.matrix,
                   method = "BFGS",
                   control = list(fnscale = -1),
                   hessian = TRUE
                   )

β.mode <- optim.res$par # Mode of betas
β.neg.inv.hessian <- -solve(optim.res$hessian) # Negative inverse hessian, when betas = mode

# c) Metropolis
```

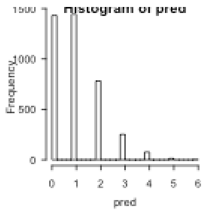
```r
# Functions
Metropolis <- function(theta, logPostFunc, c, Σ, nDraws, warmUp, ...) {
  theta_c <- theta # theta_c: Current thetas
  thetas <- matrix(nrow = nDraws,
                   ncol = length(theta))
  sigma_c <- c * Σ
  for (i in -warmUp:nDraws) {

    # Generate new thetas in the surounding area of theta_c
    theta_p <- as.vector(rmvnorm(n = 1, mean = theta_c, sigma = sigma_c))

    log.post.p <- logPostFunc(theta_p, ...)
    log.post.c <- logPostFunc(theta_c, ...)

    # Calculate the acceptance probability.
    # If there's a high probability of theta_p, accept_prob will be equal to 1 and the bern trial will
    # draw in its favour.
    accept_prob <- min(1, exp(log.post.p - log.post.c))

    # Bern trial to decide if theta_c should continue as before or if a "jump" to newly generated theta_p should be made
    bern_trial <- rbinom(1, size = 1, prob = accept_prob)

    # Set theta_c to new value
    if (bern_trial == 1) {
      theta_c <- theta_p
    }

    # When burn-in is passed, start to save the theta-values
    if(i > 0) thetas[i, ] <- theta_c;
  }

  return(thetas)
}

# Setup
X.matrix <- as.matrix(X)
beta_init <- rep(0, dim(X.matrix)[2])
sigma <- 100 * solve(t(X.matrix)%*%(X.matrix))
mu <- rep(0, dim(sigma)[2])
c = 0.5

nDraws = 4000 # No. of draws
warmUp = round(nDraws/5) # No. of Burn-in

metropolis.res <- Metropolis(theta = beta_init,
                             logPostFunc = logPostPois,
                             c = c,
                             Σ = β.neg.inv.hessian,
                             nDraws = nDraws,
                             warmUp = warmUp,
                             mu = mu,
                             sigma = sigma,
                             y = Y,
                             X = X.matrix)

postPhi <- exp(metropolis.res) # Phi = exp(β) is often more interprentable

# Plot β
par(mfrow = c(3, 3))
for (i in 1:dim(metropolis.res)[2]) {
  main_legend <- paste(c("β", i), collapse = "")
  plot(density(metropolis.res[, i]),
       main = main_legend,
       type = 'l',
       ylab = main_legend)
}
```



```
# Plot Phis
par(mfrow = c(3, 3))
for (i in 1:dim(postPhi)[2]) {
  main_legend <- paste(c("Phi:", i), collapse = " ")
  plot(density(postPhi[, i]),
       main = main_legend,
       type = 'l',
       ylab = main_legend)
}
```



```
# d) Use draws in c) to simulate the predictive distribution of a new bid

new_bid <- c(1, 1, 1, 1, 0, 0, 0, 1, 0.5)

# Simulation from predictive distribution by using Metropolis betas
# Simulate from Poisson distribution with lambda = exp(x_i * β)
lambda <- metropolis.res %*% new_bid
pred <- rpois(n = nDraws,
              exp(lambda))

# Hist draws
hist(pred, breaks = 30)

# Calculate probability of no bidders
prob_no_bidder <- sum(pred == 0)/length(pred)

stitch(script="lab3.R", system.file("misc", "knitr-template.Rhtml", package="knitr"))
```


Histogram of pred
Frequency
pred
0
500
1000
1500
0 1 2 3 4 5 6