# R Examples - An Introduction to Bayesian Statistics

Christina Knudson and An-Ting Jhuang

## Contents



This document demonstrates the hands-on examples using R for the short course - an introduction to Bayesian statistics.

## Install required packages

```
#Load packages
library(devtools)
library(ggplot2)
library(HDInterval)
library(MCMCpack)
library(mcmc)
library(mcmcse)
library(Rcpp)
library(RcppArmadillo)
library(stableGR)
```

## Coin flip example

We have a coin and wonder the probability of heads. We're willing to toss the coin a large number of times to find out more about this coin. Let $\theta$ be the probability of heads, $n$ be the number of tosses, and $y$ be the number of heads in $n$ tosses.

*Practical*: How likely is heads?
*Statistical*: What's the estimate of $\theta$?

First, let's set up the problem.

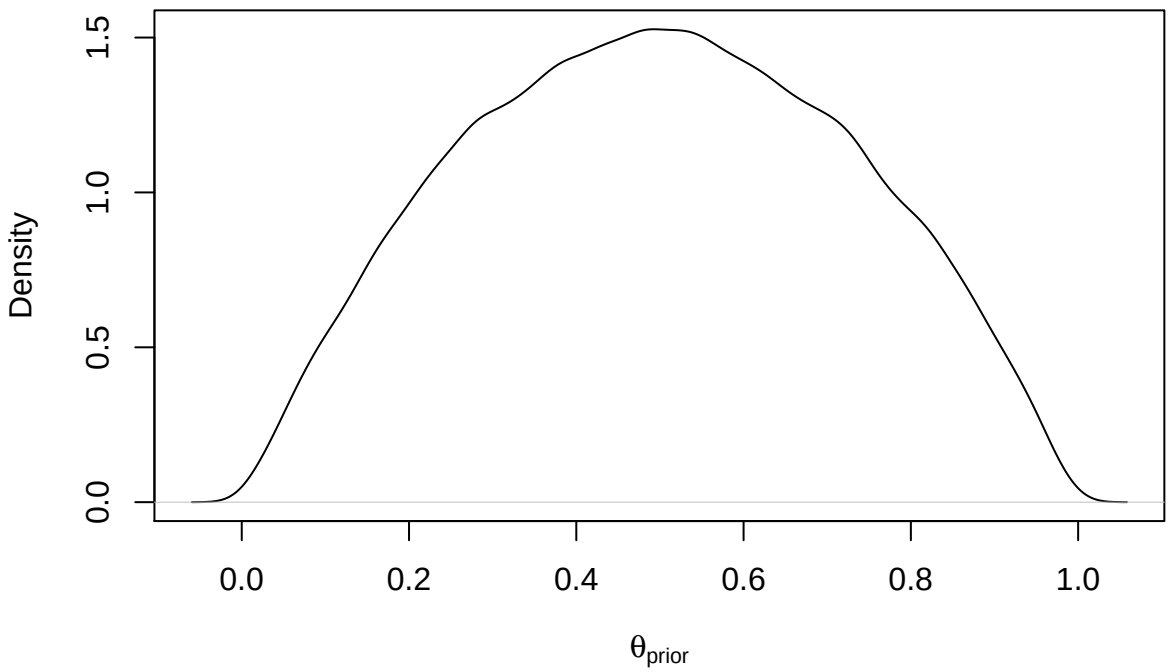
```
n       <- n_post <- 10^5
y       <- 4.8*10^4
alpha   <- 2
beta    <- 2
```

Now, let's create the posterior and likelihood.

```
set.seed(13878)
th_post  <- rbeta(n_post,alpha+y,beta+n-y)
th_pror  <- rbeta(n_post,alpha,beta)
y_like   <- rbinom(n_post,n,y/n)
```

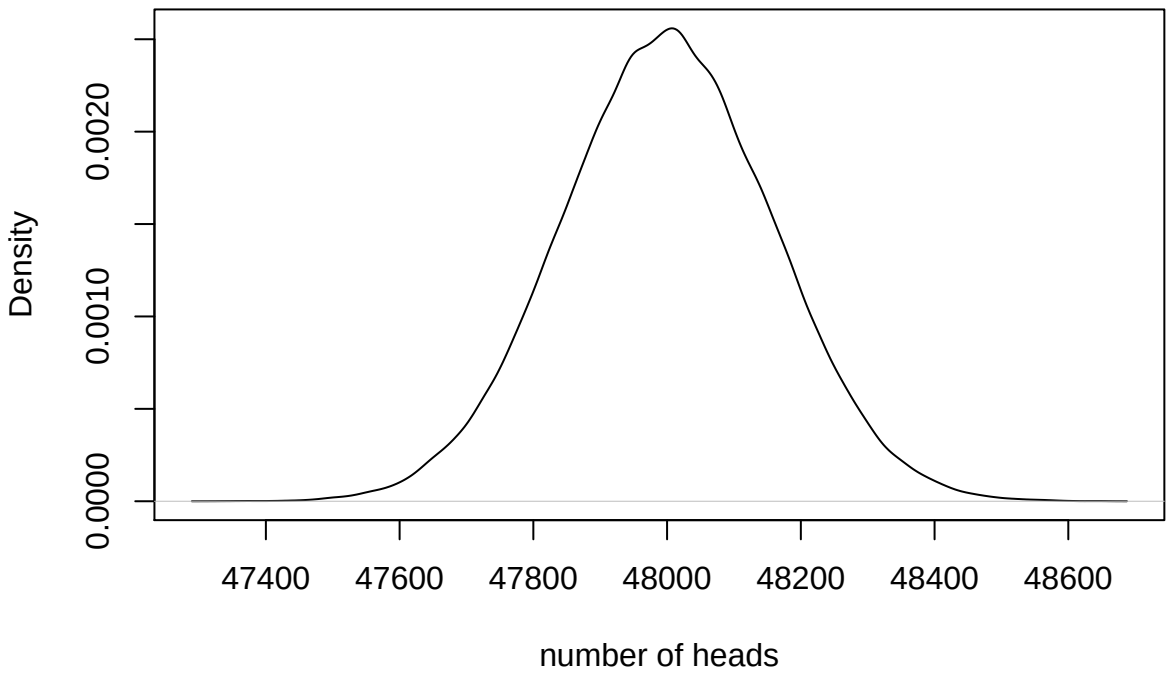
We can now plot the prior density for $\theta$.

```
plot(density(th_pror),xlab=expression(theta[prior]),main="")
```



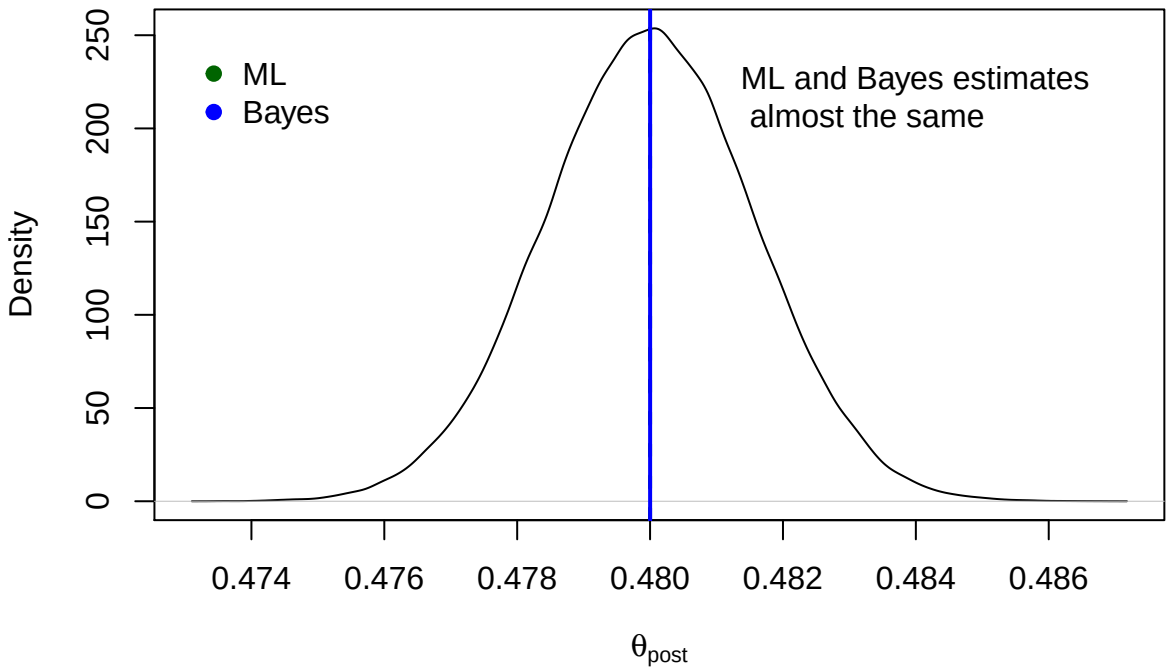

Below is the likelihood.

```
plot(density(y_like), xlab = "number of heads",main="")
```




Finally, we can plot the posterior density along with the maximum likelihood estimate.

```
#Density plot of posterior likelihood and ML/Bayes estimates
plot(density(th_post),xlab=expression(theta[post]),main="")
abline(v=.48,col="darkgreen",lty=2,lwd=2)
abline(v=(alpha+y)/(alpha+beta+n),col="blue",lwd=2)
legend(.473,250,c("ML","Bayes"),col=c("darkgreen","blue"),pch=rep(19,2),bty="n")
legend(.4805,250,"ML and Bayes estimates \n almost the same",bty="n")
```




# Linear regression

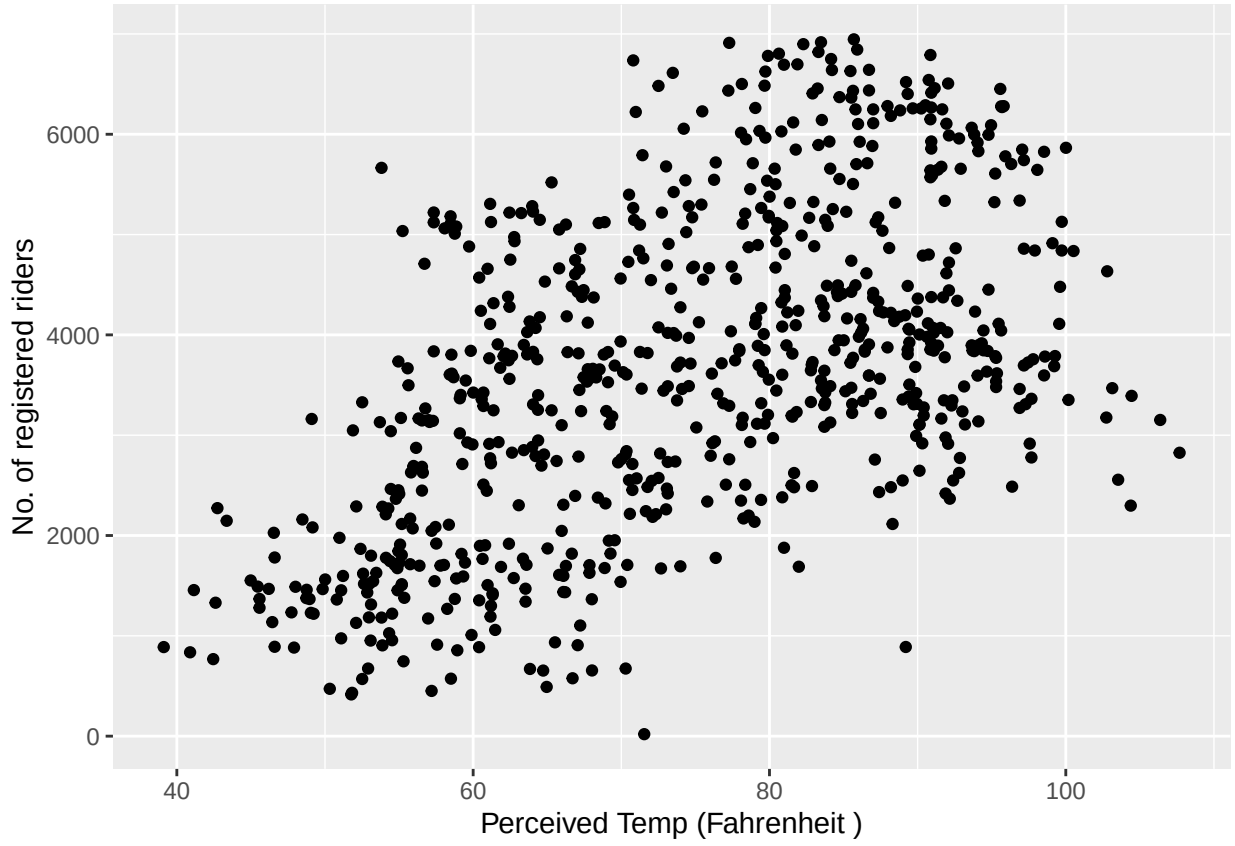
## Demo

We'll run a linear regression using frequentist and Bayesian methods. We use the *bikeshare* dataset as a sample dataset. The data contains the hourly and daily count of rentalbikes between years 2011 and 2012 in Capital bikeshare system with the corresponding weather and seasonal information. The objective is to investigate if perceived temperature is related to the number of registered riders.

Let's start with some exploratory data analysis.

```
#Import the data
bikes <- read.csv("http://www.cknudson.com/data/bikes.csv")

#EDA
ggplot(bikes, aes(y = riders_registered, x = temp_feel)) +
  geom_point() +
  xlab("Perceived Temp (Fahrenheit )") +
  ylab("No. of registered riders")
```




```
summary(bikes)
```

```
##          date        season         year          month     day_of_week
##  2011-01-01:  1   fall  :178   Min.   :2011   Aug    : 62   Fri:104
##  2011-01-02:  1   spring:184   1st Qu.:2011   Dec    : 62   Mon:105
##  2011-01-03:  1   summer:188   Median :2012   Jan    : 62   Sat:105
##  2011-01-04:  1   winter:181   Mean   :2012   Jul    : 62   Sun:105
##  2011-01-05:  1                3rd Qu.:2012   Mar    : 62   Thu:104
##  2011-01-06:  1                Max.   :2012   May    : 62   Tue:104
##  (Other)   :725                               (Other):359   Wed:104
##   weekend        holiday    temp_actual      temp_feel         humidity
##  Mode :logical   no :710   Min.   :36.36   Min.   : 39.12   Min.   :0.0000
##  FALSE:521       yes: 21   1st Qu.:56.88   1st Qu.: 62.41   1st Qu.:0.5200
##  TRUE :210                 Median :68.78   Median : 75.81   Median :0.6267
##                            Mean   :68.56   Mean   : 74.69   Mean   :0.6279
##                            3rd Qu.:80.37   3rd Qu.: 86.77   3rd Qu.:0.7302
##                            Max.   :95.59   Max.   :107.68   Max.   :0.9725
##
##    windspeed      weather_cat  riders_casual    riders_registered
##  Min.   : 1.500   categ1:463   Min.   :   2.0   Min.   :  20
##  1st Qu.: 9.042   categ2:247   1st Qu.: 315.5   1st Qu.:2497
##  Median :12.125   categ3: 21   Median : 713.0   Median :3662
##  Mean   :12.763                Mean   : 848.2   Mean   :3656
##  3rd Qu.:15.625                3rd Qu.:1096.0   3rd Qu.:4776
##  Max.   :34.000                Max.   :3410.0   Max.   :6946
##
```

```
##   riders_total
##  Min.   :  22
##  1st Qu.:3152
##  Median :4548
##  Mean   :4504
##  3rd Qu.:5956
##  Max.   :8714
##
```

Next, let's fit two linear regression models: one frequentist and one Bayesian.

```
#Frequentist
freq.fit   <- lm(riders_registered ~ temp_feel, data=bikes)

#Bayesian
bayes.fit  <- MCMCregress(riders_registered~temp_feel,
                          b0=freq.fit$coefficients,B0=.01,
                          c0=.1,d0=.1,
                          beta.start=freq.fit$coefficients,
                          data=bikes,burnin=10^3,mcmc=10^5)

#Output
summary(freq.fit)
```

```
##
## Call:
## lm(formula = riders_registered ~ temp_feel, data = bikes)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3607.1  -959.2  -153.8   998.2  3304.8
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -667.916    251.608  -2.655  0.00811 **
## temp_feel     57.892      3.306  17.514  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1310 on 729 degrees of freedom
## Multiple R-squared:  0.2961, Adjusted R-squared:  0.2952
## F-statistic: 306.7 on 1 and 729 DF,  p-value: < 2.2e-16
```

```
summary(bayes.fit)
```

```
##
## Iterations = 1001:101000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 1e+05
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                   Mean        SD  Naive SE Time-series SE
## (Intercept)    -667.90 9.975e+00 3.154e-02      3.154e-02
```

```
## temp_feel        57.89 6.474e-01 2.047e-03      2.047e-03
## sigma2      1718089.06 9.051e+04 2.862e+02      2.862e+02
##
## 2. Quantiles for each variable:
##
##                   2.5%        25%        50%        75%      97.5%
## (Intercept)    -687.45    -674.65    -667.88    -661.21    -648.29
## temp_feel        56.62      57.45      57.89      58.33      59.17
## sigma2      1550133.89 1655338.50 1714929.37 1777374.65 1904058.94
```

### You try

Everyone: Tweak the code in the demo above to fit a linear regression with the number of registered riders as the response and the humidity as the predictor. Use the frequentist linear model coefficients for the initial values.

If you finish the above problem early, try one or both of these problems:

- If you are familiar with multiple linear regression, try creating a linear model with the number of registered riders as the response and a few variables (such as the windspeed, the perceived temperature, and the humidty) as the predictors. (The syntax for this is the same as that of the lm function.) Use the frequentist linear model coefficients for the initial values.
- If you want to play with initial conditions, try changing the starting values from the frequentist linear model coefficients to something else. Notice how the starting point can effect the resulting model. Vary your Monte Carlo sample size (your chain length) to see how it interacts with starting values to impact the resulting model.

## Logistic regression

### Demo

Next, we consider a logistic regression example. For this, we create a function that calculates the log unnormalized posterior density.

```
data(logit)
out <- glm(y~x1,data=logit,family=binomial,x=TRUE)
```

Create our function that calculates the log unnormalized posterior density.

```
lupost_factory <- function(x,y)function(beta){
   eta  <- as.numeric(x%*%beta)
   logp <- ifelse(eta<0,eta-log1p(exp(eta)),-log1p(exp(-eta)))
   logq <- ifelse(eta<0,-log1p(exp(eta)),-eta-log1p(exp(-eta)))
   logl <- sum(logp[y==1])+sum(logq[y==0])
   return(logl-sum(beta^2)/8)
   }
lupost <- lupost_factory(out$x,out$y)
```

Now, we run a Metropolis random walk using the log unnormalized posterior function.

```
set.seed(317)
beta.init <- as.numeric(coefficients(out))
out <- metrop(lupost,beta.init,1e3)
names(out)
```

```
##  [1] "accept"       "batch"        "initial"      "final"        "accept.batch"
##  [6] "initial.seed" "final.seed"   "time"         "lud"          "nbatch"
```

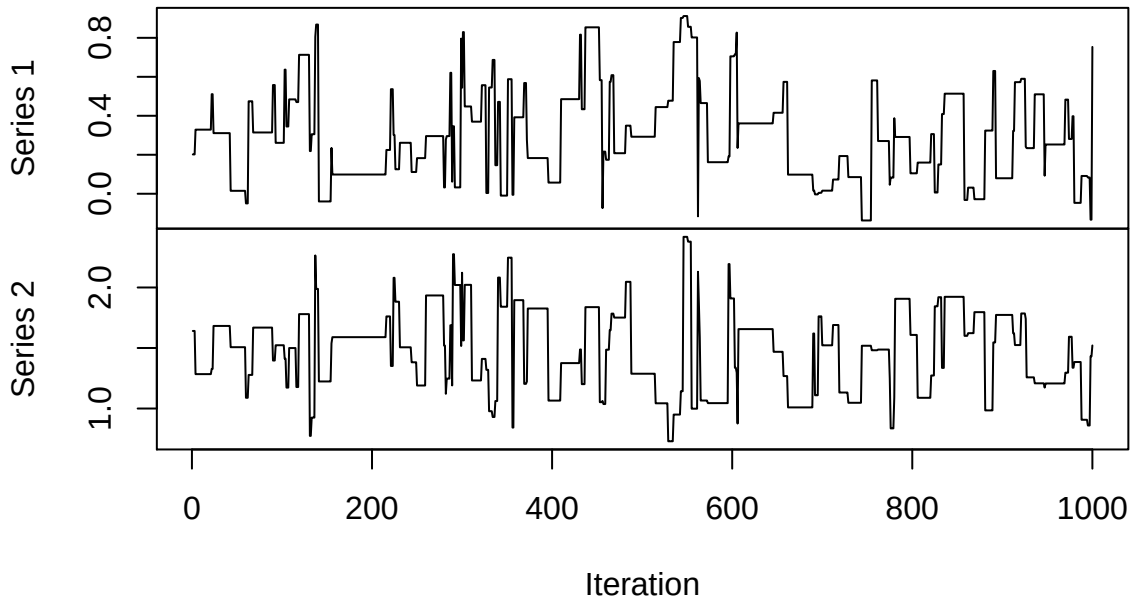
```
## [11] "blen"          "nspac"         "scale"         "debug"
```

```
out$accept
```

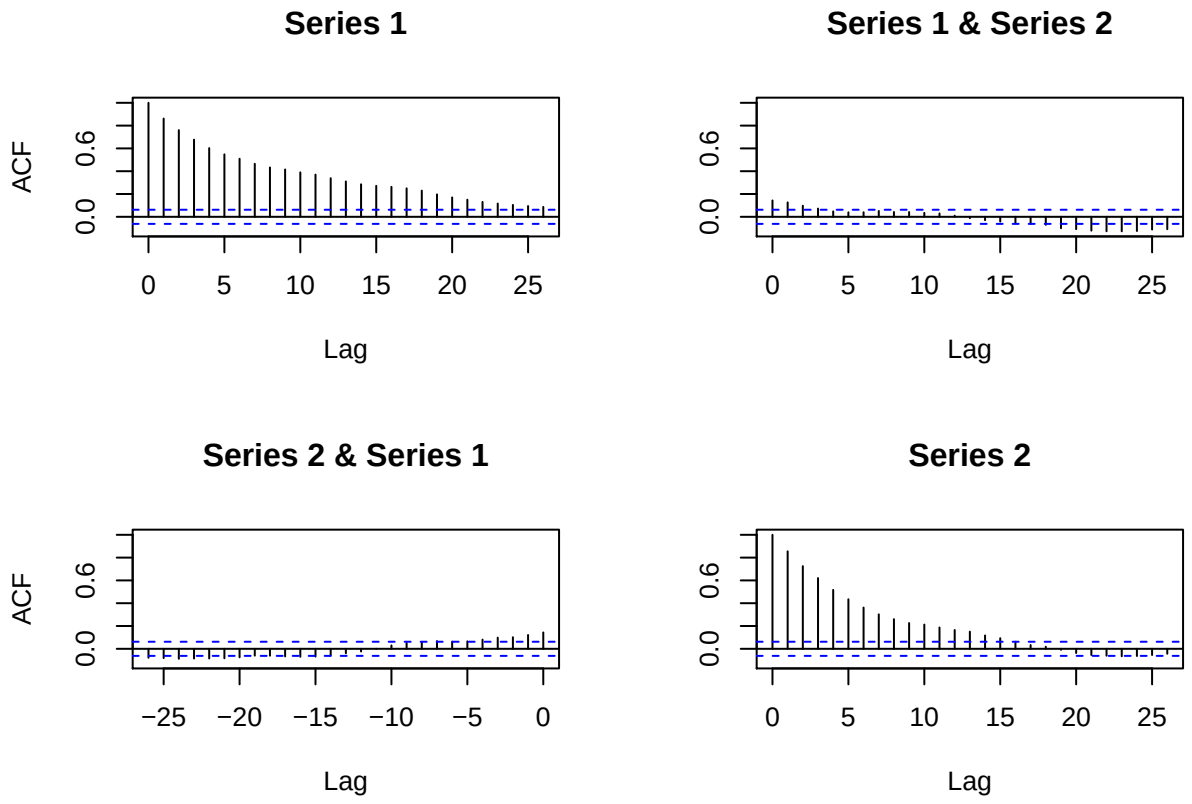
```
## [1] 0.133
```

After running our sampler, we can look at some plots. The MCMC samples are contained in out$batch.

```
plot(ts(out$batch),main="",xlab="Iteration")
```




```
acf(out$batch)
```




Now, we continue to use the lupost factory function, but let's move on to a more interesting data set: the Titanic survival data.

```
data("titanic.complete")
names(titanic.complete)
```

```
## [1] "Survived" "Pclass"   "Sex"      "Age"      "SibSp"    "Parch"    "Fare"
## [8] "Embarked"
```

For the response, use whether the passenger survived. For the predictor, use the fare the passenger paid.

```
set.seed(317)
titanic.mod <- glm(Survived ~ Fare, data = titanic.complete, family = "binomial")
beta.init <- as.numeric(coefficients(titanic.mod))
titanic.mod <- MCMClogit(Survived ~ Fare, data = titanic.complete)
summary(titanic.mod)
```

```
##
## Iterations = 1001:11000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                 Mean       SD  Naive SE Time-series SE
## (Intercept) -0.90241 0.107855 1.079e-03      3.289e-03
```

```
## Fare         0.01608 0.002502 2.502e-05      7.565e-05
##
## 2. Quantiles for each variable:
##
##                  2.5%     25%      50%      75%    97.5%
## (Intercept) -1.11859 -0.9759 -0.90001 -0.82927 -0.69723
## Fare         0.01149  0.0143  0.01599  0.01776  0.02106
```

### You try

Continue to use the MCMClogit function and the Titanic survival data, but change the the predictor.

- Use the age of the passenger to predict the log odds of survival.
- If you are comfortable with multiple regression, incorporate multiple predictors. (The syntax is the same as that of the lm function.)

## MCMC diagnostics

### Demo

After running the samplers, you can look at the MCMC diagnostics. One of the most popular diagnostics is the Gelman-Rubin diagnostic. We use the stable version of it from the package stableGR.

Input the MCMC samples.

```
stable.GR(titanic.mod)
```

```
## $psrf
## [1] 1.000414 1.000401
##
## $mpsrf
## [1] 1.00044
##
## $means
## (Intercept)        Fare
## -0.90241027  0.01607569
##
## $n.eff
## [1] 1020.501
##
## $blather
## [1] FALSE
```

If you want to isolate just the multivariate Gelman-Rubin statistic (mpsrf):

```
stable.GR(titanic.mod)$mpsrf
```

```
## [1] 1.00044
```

We say the markov chain has converged once the Gelman Rubin statistic (psrf) is below a threshold. We can calculate the threshold (i.e. the target psrf) as long as we know the number of parameters (p) and the number of chains (m). We can check the former with

```
ncol(titanic.mod)
```

```
## [1] 2
```

```
target.psrf(p=2, m=1)
```

```
## $psrf
## [1] 1.000066
##
## $epsilon
## [1] 0.05
```

Since our psrf is larger than the target, we need to sample more samples. If we don't want to do the comparison ourself, we can use the following command.

```
n.eff(titanic.mod)
```

```
## $n.eff
## [1] 1020.501
##
## $converged
## [1] FALSE
##
## $n.target
## [1] 73778
```

It tells us whether we have enough samples with the converged argument. It also tells us the effective sample size (n.eff), meaning the number of uncorrelated samples that produce the same variance as our correlated (MCMC) samples.

In this case, we need to have about 7 or 8 times the Monte Carlo sample size. We can update this in a few ways. An easy (but less intelligent way) is to rerun the code with the larger Monte Carlo sample size. The default (which we used before) was 10000, so let's multiply that by 8 to be safe.

```
newmod <- MCMClogit(Survived ~ Fare, data = titanic.complete, mcmc=80000)
n.eff(newmod)
```

```
## $n.eff
## [1] 8825.746
##
## $converged
## [1] TRUE
##
## $n.target
## NULL
```

Our new, bigger sample is now sufficiently large (it has converged). There are other (smarter) ways to increase the sample size without having to redo it, but we don't have time to get into that today. (Essentially, you want to take the last value of your Markov chain and use that as your new initial value. Run the model from there, then append your new samples to your old samples.)

Other potentially useful model information includes the Monte Carlo standard errors. Standard error (as you know it in the frequentist setting) represents the variability from sample to sample, assuming identical sample sizes and sampling schemes. Monte Carlo standard error represents the variability from MCMC sample to MCMC sample, assuming the chains have equal lengths and are constructed with the same sampler. We want small Monte Carlo standard errors.

The following command treats each variable (or column of our MCMC sample) separately.

```
mcse.mat(titanic.mod)
```

```
##                      est           se
## (Intercept) -0.90241027 3.484705e-03
## Fare         0.01607569 7.539261e-05
```

This next command estimates the covariance matrix.

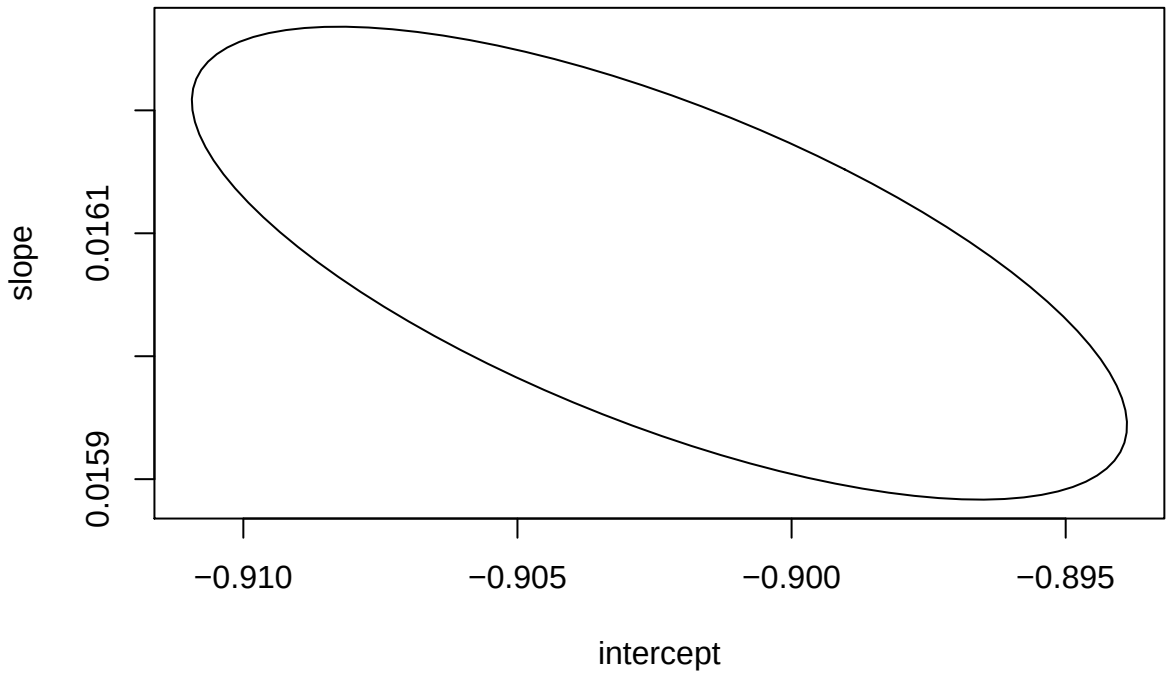
```
mcse.multi(titanic.mod)
```

```
## $cov
##              [,1]          [,2]
## [1,]  0.12143170 -1.878170e-03
## [2,] -0.00187817  6.176029e-05
##
## $est
## (Intercept)        Fare
## -0.90241027  0.01607569
```

Visualizations help make this make sense. The code below plots the credible region (jointly constructed for the two components). We can see an ellipse. A perfect circle would indicate independence between the components. The elliptical shape tells us the components are dependent.

```
mcerror <- mcse.multi(titanic.mod, blather = TRUE)

plot(confRegion(mcerror,  level = .95), xlab = "intercept", ylab = "slope", type = "l")
```




## You try

Choose one of the models you have created. Implement the methods shown above.

# Credible Sets

## Demo

Once we are certain we have a sufficently large sample, we can construct credible sets (or credible intervals), the Bayesian analog of the frequentist confidence interval. An easy way to calculate a credible interval is by using quantiles. The code below calculates a 95 percent credible interval for our second parameter.

```
quantile(titanic.mod[,2], c(.025, .975))
```

```
##       2.5%      97.5%
## 0.01149369 0.02106026
```

This is useful for hypothesis testing! Suppose you want to know whether the fare has any relationship to the log odds of a passenger's survival. If there's no relationship, then the coefficient on fare is zero. This credible interval does not contain 0. Therefore, we can say that there is indeed a linear relationship between the fare a passenger paid and the passenger's log odds of survival. In fact, because the coefficient is positive, we can say that passengers who paid more had higher chances of survival.

Monte Carlo standard errors are always a good accompaniment to any estimate. Let's find the Monte Carlo standard errors for the lower endpoint of our .025 quantile.

```
mcse.q.mat(titanic.mod, method = "bm", q=.025)
```

```
##                      est           se
## (Intercept) -1.11859163 0.0081269853
## Fare         0.01149369 0.0001210395
```

The standard error is quite small compared to the coefficient for fare. Even after taking the standard error into account, we can be fairly certain that the .025 quantile is not zero.

The credible intervals based on the highest posterior density can be calculated as follows.

```
hdi(titanic.mod)
```

```
##       (Intercept)       Fare
## lower  -1.1204212 0.01137472
## upper  -0.7028561 0.02087360
## attr(,"credMass")
## [1] 0.95
```

This credible interval tells a very similar story: it does not contain zero, so we can safely say that the fare a passenger paid has a linear relationship with their log odds of survival.

## You try

Choose one of the models you have created. Implement the methods shown above.