BAYESIAN NETWORKS 2nd Exercise Block 2

Jeremy Williams

December 23rd, 2017

```
#install.packages("bnlearn")
#install.packages("gRain")
#source("http://bioconductor.org/biocLite.R")
#biocLite("Rgraphviz")
#biocLite("RBGL")
#install.packages("Rcpp")

suppressMessages(suppressWarnings(library("bnlearn")))
suppressMessages(suppressWarnings(library("gRain")))
suppressMessages(suppressWarnings(library("gRbase")))
suppressMessages(suppressWarnings(library("Rgraphviz")))
suppressMessages(suppressWarnings(library("RBGL")))
suppressMessages(suppressWarnings(library("Rcpp")))

# First: which are the possible values of the nodes (all nodes are boolean):

TF <- c("True","False")

# Specify the CPTs:
node.M <- cptable(~ M, values = c(2, 8), levels = TF)
node.S <- cptable(~ S|M, values = c(8, 2,2,8), levels = TF)
node.B <- cptable(~ B|M, values = c(2, 8, 5, 95 ), levels = TF)
node.C <- cptable(~ C|S+B , values = c(8, 2, 8, 2,8,2,5,95 ), levels = TF)
node.H <- cptable(~ H|B, values = c(8, 2, 6, 4 ), levels = TF)

# Create an intermediate representation of the CPTs:

jwnodes <- list(node.M,node.S,node.B,node.C, node.H)
plist <- compileCPT(jwnodes)
plist
```

```
## CPTspec with probabilities:
##  P( M )
##  P( S | M )
##  P( B | M )
##  P( C | S B )
##  P( H | B )
```

```
plist$M
```

```
## M
##  True False
##   0.2   0.8
## attr(,"class")
## [1] "parray" "array"
```

```
plist$S
```

```
##        M
## S       True False
##   True   0.8   0.2
##   False  0.2   0.8
## attr(,"class")
## [1] "parray" "array"
```

```
plist$B
```

```
##         M
## B       True False
##   True   0.2  0.05
##   False  0.8  0.95
## attr(,"class")
## [1] "parray" "array"
```

```
plist$C
```

```
## , , B = True
##
##         S
## C       True False
##   True   0.8   0.8
##   False  0.2   0.2
##
## , , B = False
##
##         S
## C       True False
##   True   0.8  0.05
##   False  0.2  0.95
##
## attr(,"class")
## [1] "parray" "array"
```

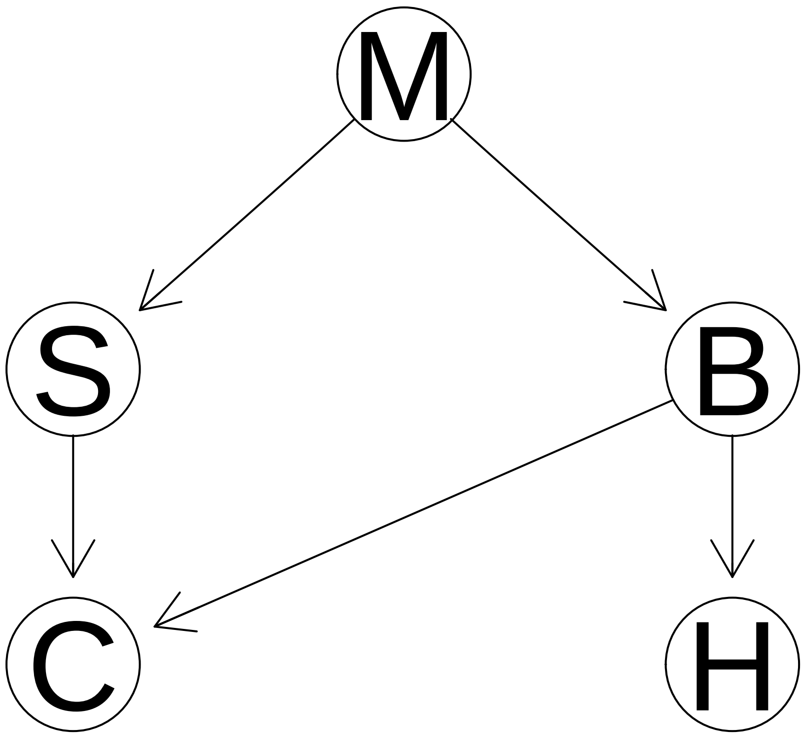
```
plist$H
```

```
##         B
## H       True False
##   True   0.8   0.6
##   False  0.2   0.4
## attr(,"class")
## [1] "parray" "array"
```

```
# Create a network of name "Norman.net", for instance:

jwBN<-grain(plist)
summary(jwBN)
```

```
## Independence network: Compiled: FALSE Propagated: FALSE
##  Nodes : chr [1:5] "M" "S" "B" "C" "H"
```

```
# The graph:

jwBNplot=plot(jwBN)
```


$$\begin{tikzcd} & M \arrow[dl] \arrow[dr] & \\ S \arrow[d] & & B \arrow[dll] \arrow[d] \\ C & & H \end{tikzcd}$$

```r
par(mfrow = c(1,1))
jwBNplot
```

```
## [1] "A graph with 5 nodes."
```

```r
#############
#Question a)#
#############

#i) Compute P(M=T/H=F)

# If evidence is "H=False", in order to compute the probability of the
# other nodes, first we add the evidence to the network and name the
# new BN jwBN1:

jwBN1<-setEvidence(jwBN,nodes=c("H"),states=c("False"))
jwBN1
```

```
## Independence network: Compiled: TRUE Propagated: TRUE
##   Nodes: chr [1:5] "M" "S" "B" "C" "H"
##   Evidence:
##   nodes is.hard.evidence hard.state
## 1     H              TRUE      False
##   pEvidence: 0.384000
```

```r
# The marginal distributions given the evidence are:

jwmarg1=querygrain(jwBN1,nodes=c("M"), type="marginal")
```

```
jwmarg1
```

```
## $M
## M
##   True  False
## 0.1875 0.8125
```

```
# P(M=T/H=F) is:

jwmarg1$M["True"]
```

```
##   True
## 0.1875
```

```
#ii) Compute P(M=F/H=T)

# If evidence is "H=True", in order to compute the probability of the
# other nodes, first we add the evidence to the network and name the
# new BN jwBN2:

jwBN2<-setEvidence(jwBN,nodes=c("H"),states=c("True"))
jwBN2
```

```
## Independence network: Compiled: TRUE Propagated: TRUE
##   Nodes: chr [1:5] "M" "S" "B" "C" "H"
##   Evidence:
##   nodes is.hard.evidence hard.state
## 1     H              TRUE       True
##   pEvidence: 0.616000
```

```
# The marginal distributions given the evidence are:

jwmarg2=querygrain(jwBN2,nodes=c("M"), type="marginal")
jwmarg2
```

```
## $M
## M
##      True     False
## 0.2077922 0.7922078
```

```
# P(M=F/H=T) is:

jwmarg2$M["False"]
```

```
##     False
## 0.7922078
```

```
#############
#Question b)#
#############

#LS Algorithm for a single evidence and the particular BN jwBN.

binary_random.LS <- function(p.prior.True){
  seed <- runif(1)
  if(seed < p.prior.True){return("True");}
  else{return("False");}
```

```r
}

run_iterations1 = 10^4

LS_algo <- function(pred.node = "M", evid.node = "H",
                    pred.value = "True", evid.value = "False",
                    iterations = run_iterations1){

  count.pred_evid = 0; count.evid = 0;

  iter_counter = 0
  while(iter_counter < iterations){

    #Values of root node and children
    M.value <- binary_random.LS(plist$"M"[["True"]])
    S.value <- binary_random.LS(plist$"S"[ "True" , M.value])
    B.value <- binary_random.LS(plist$"B"[ "True" , M.value])
    C.value <- binary_random.LS(plist$"C"[,, B.value]["True", S.value])
    H.value <- binary_random.LS(plist$"H"["True", B.value])

    #Update counters
    node_value <- function(node = "M"){
      if(node == "M"){return(M.value)}
      else if(node == "S"){return(S.value)}
      else if(node == "B"){return(B.value)}
      else if(node == "C"){return(C.value)}
      else if(node == "H"){return(H.value)}
      else{print("Node not found");return(NA);}
    }

    e_observed <- node_value(evid.node)
    p_observed <- node_value(pred.node)

    if(e_observed == evid.value){
      count.evid = count.evid +1;
      if(p_observed == pred.value){
        count.pred_evid = count.pred_evid +1}
    }
    iter_counter = iter_counter +1;
  }#main simulation while

  return(count.pred_evid/count.evid)
}#end function

# Estimated P(M=T/H=F)
LsExt_prob1 <- LS_algo("M","H","True", "False")
LsExt_prob1
```

```
## [1] 0.185718
```

```r
# Estimated P(M=F/H=T)

LsExt_prob2 <- LS_algo("M","H","False", "True")
LsExt_prob2
```

```
## [1] 0.7949466
```

```
#############
#Question c)#
#############

#LW Algorithm for a single evidence and the particular BN jwBN.

binary_random.LW <- function(p.prior.True){
  seed <- runif(1)
  if(seed < p.prior.True){return("True");}
  else{return("False");}
}

run_iterations2 = 10^4

LW_algo <- function(pred.node = "M",evid.node = "H",
                   pred.value = "True", evid.value = "False",
                   iterations = run_iterations2){

  count.pred_evid = 0; count.evid = 0;
  likelihood.e <-0

  iter_counter = 0
  while(iter_counter < iterations){

    #Value of root nodes (M)
    if("M" == evid.node){
      M.value <- evid.value
      likelihood.e <- plist$"M"[[evid.value]]
    }
    else{
      M.value <- binary_random.LW(plist$"M"[["True"]])
    }
    #Values for each child
    #S
    if("S" == evid.node){
      S.value <- evid.value
      likelihood.e <- plist$"S"[ "True" , M.value]
    }
    else{
      S.value <- binary_random.LW(plist$"S"[ "True" , M.value])
    }

    #B
    if("B" == evid.node){
      B.value <- evid.value
      likelihood.e <- plist$"B"[ "True" , M.value]
    }

    else{
      B.value <- binary_random.LW(plist$"B"[ "True" , M.value])
    }
    #C
    if("C" == evid.node){
      C.value <- evid.value
      likelihood.e <- plist$"C"[,, B.value]["True", S.value]
    }

    else{
      C.value <- binary_random.LW(plist$"C"[,, B.value]["True", S.value])
    }

    #H
    if("H" == evid.node){
      H.value <- evid.value
      likelihood.e <- plist$"H"["True", B.value]
    }

    else{
      H.value <- binary_random.LW(plist$"H"["True", B.value])
    }

    #Update counters
    node_value <- function(node = "M"){
      if(node == "M"){return(M.value)}
      else if(node == "S"){return(S.value)}
      else if(node == "B"){return(B.value)}
      else if(node == "C"){return(C.value)}
      else if(node == "H"){return(H.value)}
      else{print("Node not found");return(NA);}
    }

    e_observed <- node_value(evid.node)
    p_observed <- node_value(pred.node)

    if(e_observed == evid.value){
      count.evid = count.evid + likelihood.e;
      if(p_observed == pred.value){
        count.pred_evid = count.pred_evid +likelihood.e}
    }
    iter_counter = iter_counter +1;
  }#main simulation while

  return(count.pred_evid/count.evid)
}#end function


# Estimated P(M=T/H=F)
LwExt_prob1 <- LW_algo("M","H","True", "False")
LwExt_prob1
```

```
## [1] 0.2135875
```

```
# Estimated P(M=F/H=T)

LwExt_prob2 <- LW_algo("M","H","False", "True")
```

```
LwExt_prob2
```

```
## [1] 0.7959376
```

Question d) #############

Compute exactly by "hand"

Now. . . . applying the Bayes' Theorem we have

P(M=T | H=F) = [P(H=F | M=T)P(M=T)] / P(H=F)

Now,

We need to find P(H=F | M=T), P(M=T), P(H=F)

1) P(H=F | M=T),
   = P(H=F | B=T,M=T)P(B=T | M=T) + P(H=F | B=F ,M=T)P(B=F | M=T) = P(H=F | B=T)P(B=T | M=T) + P(H=F | B=F)P(B=F | M=T) = $(0.2 \times 0.2) + (0.4 \times 0.8) = 0.36$
2) P(M=T) = 0.2,
3) P(B=T) = P(B=T | M=T)P(M=T) + P(B=T | M=F)P(M=F) = $(0.2 \times 0.2)+(0.05 \times 0.08) = 0.08$,
4) P(B=F) = 1 - P(M=T) = 1 - 0.2 = 0.92
5) P(H=F) = P(H=F | B=T)P(B=T) + P(H=F | B=F)P(B=F) = $(0.2\times0.08) + (0.4\times0.92) = 0.384$.

Now, we substitute all of the necessary values:

P(M=T | H=F) = [P(H=F | M=T)P (M=T)] / P(H=F) = (0.36 x 0.2) / 0.384 = 0.1875

Compare with (Question a), (Question b) and (Question c)

We have calculated P(M=T | H=F) = 0.1875 above.

Looking at the values by the R code (gRain), we can say that the calculation (by hand) is correct and corresponds with gRain for both P(M=T | H=F) and P(M=F | H=T).

```
#############
#Question e)#
#############

#install.packages("bnlearn")
#install.packages("gRain")
#source("http://bioconductor.org/biocLite.R")
#biocLite("Rgraphviz")
#biocLite("RBGL")
#install.packages("Rcpp")

suppressMessages(suppressWarnings(library("bnlearn")))
suppressMessages(suppressWarnings(library("gRain")))
suppressMessages(suppressWarnings(library("gRbase")))
suppressMessages(suppressWarnings(library("Rgraphviz")))
suppressMessages(suppressWarnings(library("RBGL")))
suppressMessages(suppressWarnings(library("Rcpp")))

# First: which are the possible values of the nodes (all nodes are boolean):

TF <- c("True","False")

# Specify the CPTs:
```

```r
node.M <- cptable(~ M, values = c(2, 8), levels = TF)
node.S <- cptable(~ S|M, values = c(8, 2,2,8), levels = TF)
node.B <- cptable(~ B|M, values = c(2, 8, 5, 95 ), levels = TF)
node.C <- cptable(~ C|S+B , values = c(8, 2, 8, 2,8,2,5,95 ), levels = TF)
node.H <- cptable(~ H|B, values = c(8, 2, 6, 4 ), levels = TF)

# Create an intermediate representation of the CPTs:

jwnodes <- list(node.M,node.S,node.B,node.C, node.H)
plist <- compileCPT(jwnodes)

# Create a network of name "Norman.net", for instance:

jwBN<-grain(plist)

# new BN jwBN1:

jwBN1<-setEvidence(jwBN,nodes=c("H"),states=c("False"))

# The marginal distributions given the evidence are:

jwmarg1=querygrain(jwBN1,nodes=c("M"), type="marginal")

# new BN jwBN2:

jwBN2<-setEvidence(jwBN,nodes=c("H"),states=c("True"))


# The marginal distributions given the evidence are:

jwmarg2=querygrain(jwBN2,nodes=c("M"), type="marginal")

##############################
#Kullback-Leibler divergence#
##############################

# For evidence H=F and M,
# Compute the Kullback-Leibler divergence for the LS Algorithm and LW algorithm.

prob_exact <- jwmarg1$M[["True"]]

#KL divergence for the LS algorithm

KL_LS <- function(prob_exact = jwmarg1$M[["True"]],
                  iter = 10^4){
  prob_ls <- LS_algo("M","H","True", "False",iter)
  kl <- prob_exact * log(prob_exact/prob_ls) +   (1-prob_exact)*log((1-prob_exact)/(1-prob_ls))
  if(kl <0){print("Negative KL_LS!!!")}
  return(kl)
}

#KL divergence for the LW algorithm
```

```r
KL_LW <- function(prob_exact = jwmarg1$M[["True"]],
                  iter = 10^4){
  prob_lw <- LW_algo("M","H","True", "False", iter)
  kl <- prob_exact * log(prob_exact/prob_lw) +   (1-prob_exact)*log((1-prob_exact)/(1-prob_lw))
  if(kl <0){print("Negative KL_LW!!!")}
  return(kl)
}

#Consider the following array of integers for LS and LW algorithms.

iter.vector <- c(10^3, 2*10^3, 3*10^3, 4*10^3, 5*10^3, 6*10^3, 7*10^3)


#Sampling the estimated probabilities using Rep 10 and 20:

########
#Rep 10#
########

reps = 10;

ls.vector1 <- numeric(length(iter.vector))
ls.vector2 <- numeric(length(iter.vector))

lw.vector1 <- numeric(length(iter.vector))
lw.vector2 <- numeric(length(iter.vector))

ls.matrix1 <- matrix(data = 0, nrow = length(iter.vector)
                     , ncol = reps)
ls.matrix2 <- matrix(data = 0, nrow = length(iter.vector)
                     , ncol = reps)

lw.matrix1 <- matrix(data = 0, nrow = length(iter.vector)
                     , ncol = reps)
lw.matrix2 <- matrix(data = 0, nrow = length(iter.vector)
                     , ncol = reps)

for(i in 1:length(iter.vector)){
  for(j in 1:reps){
    iter = iter.vector[i]
    ls.matrix1[i,j] <- LS_algo("M","H","True", "False", iter)
    ls.matrix2[i,j] <- LS_algo("M","H","False", "True", iter)
    lw.matrix1[i,j] <- LW_algo("M","H","True", "False", iter)
    lw.matrix2[i,j] <- LW_algo("M","H","False", "True", iter)
  }
  ls.vector1[i] <- mean(ls.matrix1[i,])
  ls.vector2[i] <- mean(ls.matrix2[i,])
  lw.vector1[i] <- mean(lw.matrix1[i,])
  lw.vector2[i] <- mean(lw.matrix2[i,])
}

# Exact prob - P(M=T/H=F)
```

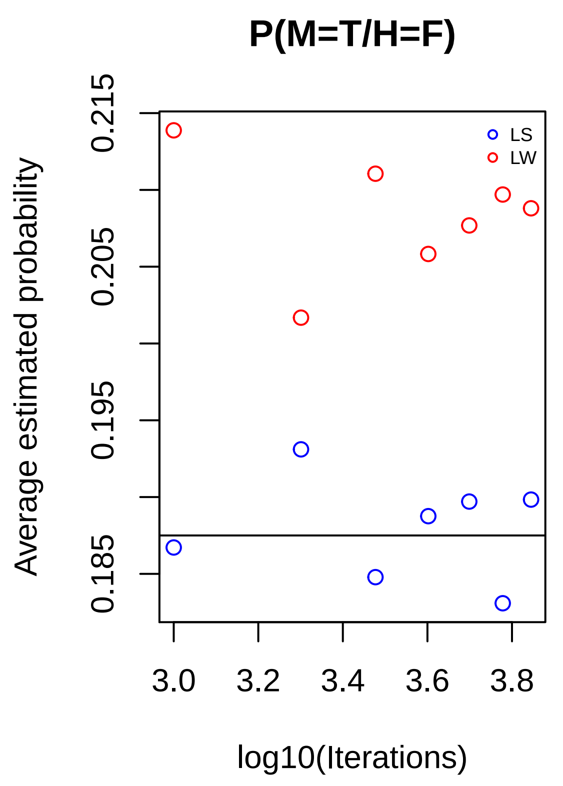
```
par(mfrow = c(1,2))

Ext_probT <- jwmarg1$M["True"]
Ext_probT
```

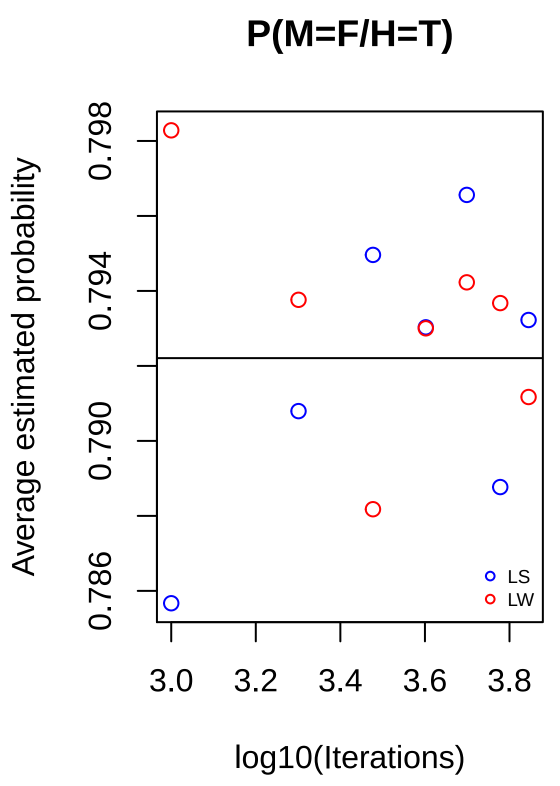
```
##   True
## 0.1875
```

```
plot(log10(iter.vector), ls.vector1,
     ylim = c(min(ls.vector1, lw.vector1), max(ls.vector1, lw.vector1)),
     col = "blue",
     xlab = "log10(Iterations)",
     ylab = "Average estimated probability",
     main = "P(M=T/H=F)")
points(log10(iter.vector), lw.vector1, col = "red")
abline(h=Ext_probT, col = "black")
legend("topright",
       c("LS", "LW"), col = c( "blue", "red"),
       bty = 'n',
       pch = 21,
       cex = 0.6)

# Exact prob - P(M=F/H=T)

Ext_probF <- jwmarg2$M["False"]
Ext_probF
```

```
##     False
## 0.7922078
```

```
plot(log10(iter.vector), ls.vector2,
     ylim = c(min(ls.vector2, lw.vector2), max(ls.vector2, lw.vector2)),
     col = "blue",
     xlab = "log10(Iterations)",
     ylab = "Average estimated probability",
     main = "P(M=F/H=T)")
points(log10(iter.vector), lw.vector2, col = "red")
abline(h=Ext_probF, col = "black")
legend("bottomright",
       c("LS", "LW"), col = c( "blue", "red"),
       bty = 'n',
       pch = 21,
       cex = 0.6)
```





```
########
#Rep 20#
########

reps1 = 20;

ls.vector1 <- numeric(length(iter.vector))
ls.vector2 <- numeric(length(iter.vector))

lw.vector1 <- numeric(length(iter.vector))
lw.vector2 <- numeric(length(iter.vector))

ls.matrix1 <- matrix(data = 0, nrow = length(iter.vector)
                      , ncol = reps1)
ls.matrix2 <- matrix(data = 0, nrow = length(iter.vector)
                      , ncol = reps1)

lw.matrix1 <- matrix(data = 0, nrow = length(iter.vector)
                      , ncol = reps1)
lw.matrix2 <- matrix(data = 0, nrow = length(iter.vector)
                      , ncol = reps1)

for(i in 1:length(iter.vector)){
  for(j in 1:reps1){
    iter = iter.vector[i]
    ls.matrix1[i,j] <- LS_algo("M","H","True", "False", iter)
```

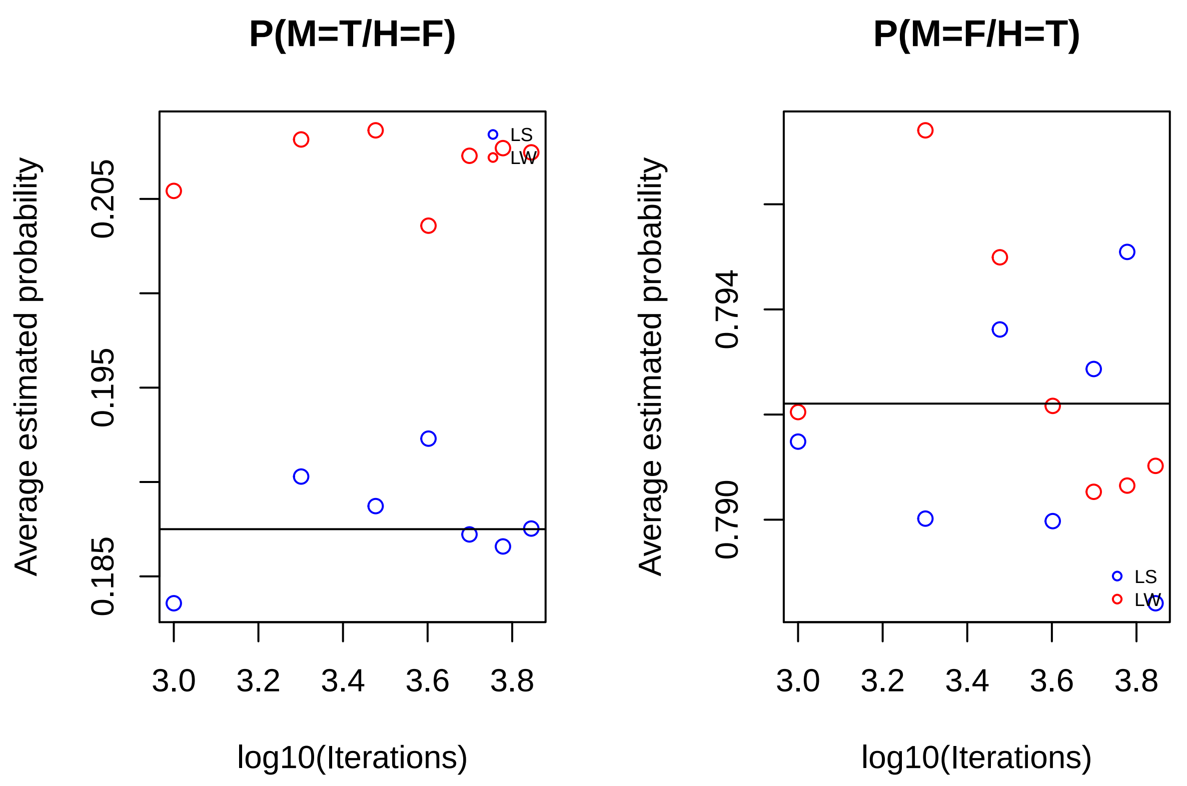
```r
    ls.matrix2[i,j] <- LS_algo("M","H","False", "True", iter)
    lw.matrix1[i,j] <- LW_algo("M","H","True", "False", iter)
    lw.matrix2[i,j] <- LW_algo("M","H","False", "True", iter)
  }
  ls.vector1[i] <- mean(ls.matrix1[i,])
  ls.vector2[i] <- mean(ls.matrix2[i,])
  lw.vector1[i] <- mean(lw.matrix1[i,])
  lw.vector2[i] <- mean(lw.matrix2[i,])
}

# Exact prob - P(M=T/H=F)

par(mfrow = c(1,2))

Ext_probT <- jwmarg1$M["True"]
Ext_probT
```

```
##   True
## 0.1875
```

```r
plot(log10(iter.vector), ls.vector1,
     ylim = c(min(ls.vector1, lw.vector1), max(ls.vector1, lw.vector1)),
     col = "blue",
     xlab = "log10(Iterations)",
     ylab = "Average estimated probability",
     main = "P(M=T/H=F)")
points(log10(iter.vector), lw.vector1, col = "red")
abline(h=Ext_probT, col = "black")
legend("topright",
       c("LS", "LW"), col = c( "blue", "red"),
       bty = 'n',
       pch = 21,
       cex = 0.6)

# Exact prob - P(M=F/H=T)

Ext_probF <- jwmarg2$M["False"]
Ext_probF
```

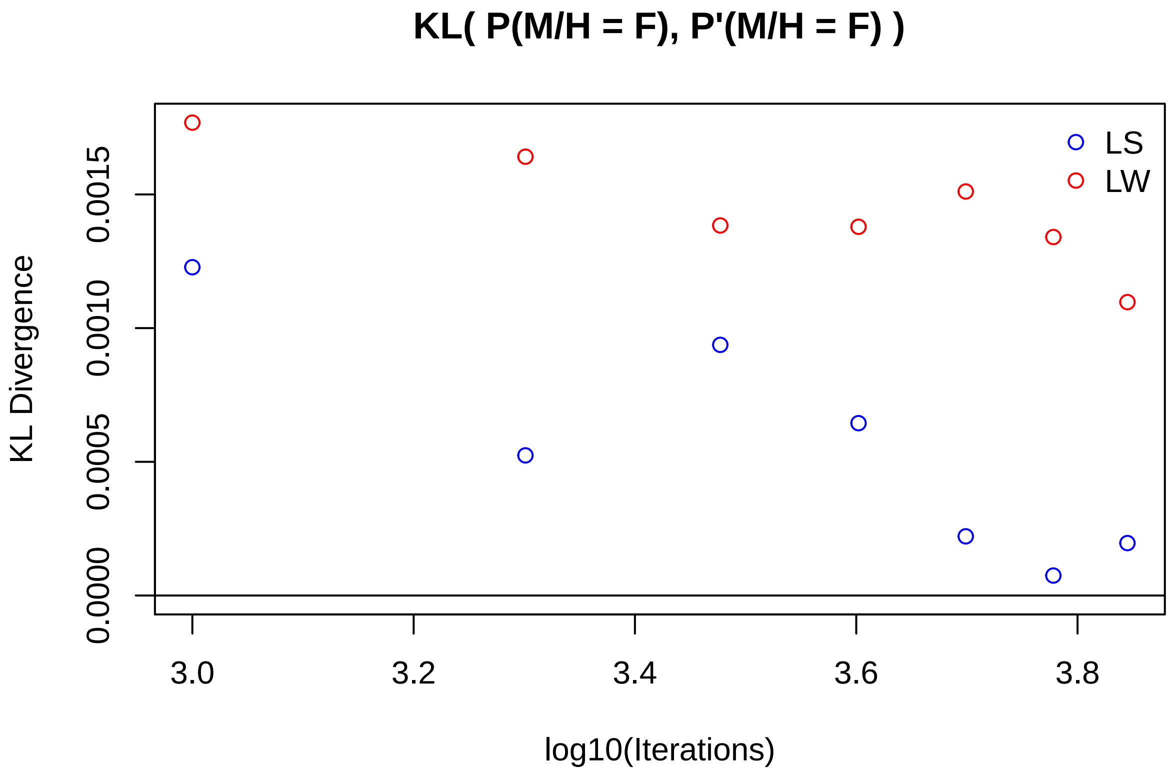
```
##     False
## 0.7922078
```

```r
plot(log10(iter.vector), ls.vector2,
     ylim = c(min(ls.vector2, lw.vector2), max(ls.vector2, lw.vector2)),
     col = "blue",
     xlab = "log10(Iterations)",
     ylab = "Average estimated probability",
     main = "P(M=F/H=T)")
points(log10(iter.vector), lw.vector2, col = "red")
abline(h=Ext_probF, col = "black")
legend("bottomright",
       c("LS", "LW"), col = c( "blue", "red"),
       bty = 'n',
       pch = 21,
       cex = 0.6)
```



```
##################################################################
#Kullback-Leibler divergence for LS Algorithm and LW algorithm #
##################################################################

kls.vector <- numeric(length(iter.vector))
klw.vector <- numeric(length(iter.vector))

reps2 = 10
kls.matrix <- matrix(data = 0, nrow = length(iter.vector)
                     , ncol = reps2)
klw.matrix <- matrix(data = 0, nrow = length(iter.vector)
                     , ncol = reps2)

for(i in 1:length(iter.vector)){
  for(j in 1:reps2){
    kls.matrix[i,j] <- KL_LS(iter = iter.vector[i])
    klw.matrix[i,j] <- KL_LW(iter = iter.vector[i])
  }
  kls.vector[i] <- mean(kls.matrix[i,])
  klw.vector[i] <- mean(klw.matrix[i,])
}

# Plot shows comparsion:

par(mfrow = c(1,1))
```

```r
plot(log10(iter.vector), kls.vector,
     ylim  = c(0, max( c(kls.vector, klw.vector)) ) ,
     xlab = "log10(Iterations)",
     ylab = "KL Divergence",
     main = "KL( P(M/H = F), P'(M/H = F) )",
     col = "blue")
points(log10(iter.vector), klw.vector, col = "red")

abline(h=0, col = "black")
legend("topright",
       c("LS", "LW"), col = c( "blue", "red"),
       bty = 'n',
       pch = 21,
       cex = 1)
```



Which algorithm seems to be better?

After looking at the results obtained above, based on the sample replications and the comparsion graph of both algorithms, it is easy to see that the LS algorithm converges to zero “0” better then the LW algorithm.

Therefore, it can be said that the LS algorithm is better than the LW algorithm; from all calcuated methods used.