## 1. Graphical models

library(bnlearn)  
library(gRain)

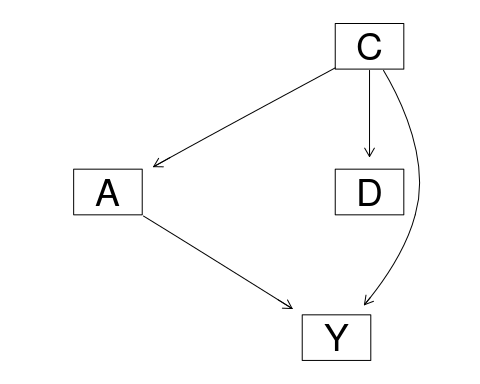
## Loading required package: gRbase

##   
## Attaching package: 'gRbase'

## The following objects are masked from 'package:bnlearn':  
##   
## ancestors, children, nodes, parents

dag = model2network("[C][D|C][A|C][Y|A:C]")  
  
graphviz.plot(dag)

## Loading required namespace: Rgraphviz



countC\_ND = 0  
countD\_NC = 0  
for (i in 1:1000) {  
 dag = model2network("[C][D|C][A|C][Y|A:C]")  
 # C  
 s = runif(1)  
 cptC = c(s, 1-s)  
 dim(cptC) = c(2)  
 dimnames(cptC) = list(C = c("C1", "C0"))  
  
 # D | C  
 s1 = runif(1)  
 s2 = runif(1)  
 cptD = matrix(c(s1, 1-s1,   
 s2, 1-s2),  
 nrow = 2)  
 dimnames(cptD) = list(D = c("D1", "D0"), C = c("C1", "C0"))  
  
 # A | C  
 s1 = runif(1)  
 s2 = runif(1)  
 cptA = matrix(c(s1, 1-s1,   
 s2, 1-s2),  
 nrow = 2)  
 dimnames(cptA) = list(A = c("A1", "A0"), C = c("C1", "C0"))  
   
 # Y | A,C  
 s1 = runif(1)  
 s2 = runif(1)  
 s3 = runif(1)  
 s4 = runif(1)  
 cptY = matrix(c(s1,1-s1,  
 s2,1-s2,  
 s3,1-s3,  
 s4,1-s4),  
 )  
 dim(cptY) = c(2,2,2)  
 dimnames(cptY) = list(Y = c("Y1", "Y0"), A = c("A1", "A0"), C = c("C1", "C0"))  
  
 fit = custom.fit(dag, list(C = cptC,D = cptD,A = cptA, Y = cptY))  
 model = compile(as.grain(fit))  
   
 # for p(y|a,c)  
 Y1\_A1C1 <- querygrain(setEvidence(object = model, nodes = c("A", "C"), states = c("A1", "C1")),nodes = "Y")$Y[1]  
 Y1\_A1C0 <- querygrain(setEvidence(object = model, nodes = c("A", "C"), states = c("A1", "C0")),nodes = "Y")$Y[1]  
 Y1\_A0C1 <- querygrain(setEvidence(object = model, nodes = c("A", "C"), states = c("A0", "C1")),nodes = "Y")$Y[1]  
 Y1\_A0C0 <- querygrain(setEvidence(object = model, nodes = c("A", "C"), states = c("A0", "C0")),nodes = "Y")$Y[1]  
   
 # for p(y|a,d)  
 Y1\_A1D1 <- querygrain(setEvidence(object = model, nodes = c("A", "D"), states = c("A1", "D1")),nodes = "Y")$Y[1]  
 Y1\_A1D0 <- querygrain(setEvidence(object = model, nodes = c("A", "D"), states = c("A1", "D0")),nodes = "Y")$Y[1]  
 Y1\_A0D1 <- querygrain(setEvidence(object = model, nodes = c("A", "D"), states = c("A0", "D1")),nodes = "Y")$Y[1]  
 Y1\_A0D0 <- querygrain(setEvidence(object = model, nodes = c("A", "D"), states = c("A0", "D0")),nodes = "Y")$Y[1]  
   
 if(Y1\_A1C1 >= Y1\_A1C0 && Y1\_A0C1 >= Y1\_A0C0){  
 if(Y1\_A1C1 <= Y1\_A1C0 && Y1\_A0C1 <= Y1\_A0C0){  
 monotoneC = TRUE  
 }  
 } else {  
 monotoneC = FALSE  
 }  
   
 monotoneD = FALSE  
 if(Y1\_A1D1 >= Y1\_A1D0 && Y1\_A0D1 >= Y1\_A0D0){  
 if(Y1\_A1D1 <= Y1\_A1D0 && Y1\_A0D1 <= Y1\_A0D0){  
 monotoneD = TRUE  
 }  
 } else {  
 monotoneD = FALSE  
 }  
   
   
 if(monotoneC && !monotoneD){  
 countC\_ND = countC\_ND +1  
 }else if(!monotoneC && monotoneD){  
 countD\_NC = countD\_NC +1   
 }  
}  
  
countC\_ND

## [1] 0

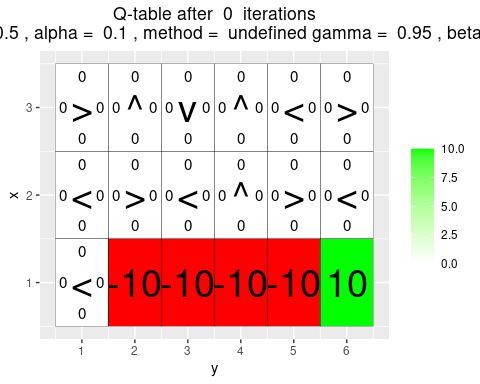
countD\_NC

## [1] 0

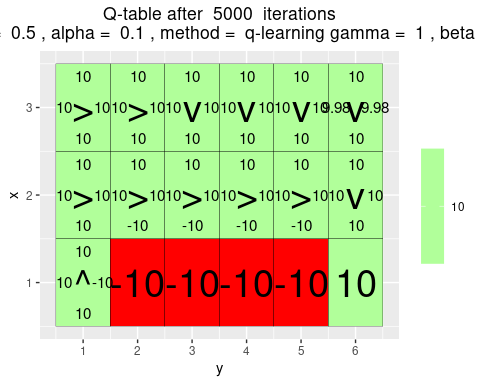
1. how many of parametrisation result in p(|a,c) is monotone in C but p(y|a,d) is not monotone in d
2. p(y∣a, d) is monotone in D but p(y∣a, c) is not monotone in C

set.seed(1234)  
library(ggplot2)  
  
arrows <- c("^", ">", "v", "<")  
action\_deltas <- list(c(1,0), # up  
 c(0,1), # right  
 c(-1,0), # down  
 c(0,-1)) # left  
  
vis\_environment <- function(iterations=0, epsilon = 0.5, alpha = 0.1, gamma = 0.95, beta = 0, method ="undefined"){  
  
 df <- expand.grid(x=1:H,y=1:W)  
 foo <- mapply(function(x,y) ifelse(reward\_map[x,y] == 0,q\_table[x,y,1],NA),df$x,df$y)  
 df$val1 <- as.vector(round(foo, 2))  
 foo <- mapply(function(x,y) ifelse(reward\_map[x,y] == 0,q\_table[x,y,2],NA),df$x,df$y)  
 df$val2 <- as.vector(round(foo, 2))  
 foo <- mapply(function(x,y) ifelse(reward\_map[x,y] == 0,q\_table[x,y,3],NA),df$x,df$y)  
 df$val3 <- as.vector(round(foo, 2))  
 foo <- mapply(function(x,y) ifelse(reward\_map[x,y] == 0,q\_table[x,y,4],NA),df$x,df$y)  
 df$val4 <- as.vector(round(foo, 2))  
 foo <- mapply(function(x,y)   
 ifelse(reward\_map[x,y] == 0,arrows[GreedyPolicy(x,y)],reward\_map[x,y]),df$x,df$y)  
 df$val5 <- as.vector(foo)  
 foo <- mapply(function(x,y) ifelse(reward\_map[x,y] == 0,max(q\_table[x,y,]),  
 ifelse(reward\_map[x,y]<0,NA,reward\_map[x,y])),df$x,df$y)  
 df$val6 <- as.vector(foo)  
   
 print(ggplot(df,aes(x = y,y = x)) +  
 scale\_fill\_gradient(low = "white", high = "green", na.value = "red", name = "") +  
 geom\_tile(aes(fill=val6)) +  
 geom\_text(aes(label = val1),size = 4,nudge\_y = .35,na.rm = TRUE) +  
 geom\_text(aes(label = val2),size = 4,nudge\_x = .35,na.rm = TRUE) +  
 geom\_text(aes(label = val3),size = 4,nudge\_y = -.35,na.rm = TRUE) +  
 geom\_text(aes(label = val4),size = 4,nudge\_x = -.35,na.rm = TRUE) +  
 geom\_text(aes(label = val5),size = 10) +  
 geom\_tile(fill = 'transparent', colour = 'black') +   
 ggtitle(paste("Q-table after ",iterations," iterations\n",  
 "(epsilon = ",epsilon,", alpha = ",alpha,", method = ",method,"gamma = ",gamma,", beta = ",beta,")")) +  
 theme(plot.title = element\_text(hjust = 0.5)) +  
 scale\_x\_continuous(breaks = c(1:W),labels = c(1:W)) +  
 scale\_y\_continuous(breaks = c(1:H),labels = c(1:H)))  
}  
  
GreedyPolicy <- function(x, y){  
 # Get the Q-values for all actions at state (x, y)  
 q\_values <- q\_table[x, y, ]  
   
 # Find the max Q-value  
 max\_q <- max(q\_values)  
   
 # Identify all actions with maximum Q-value  
 max\_actions <- which(q\_values == max\_q)  
   
 # Check and resolve ties  
 if (length(max\_actions) > 1) {  
 action <- sample(max\_actions, 1)  
 } else {  
 action <- max\_actions  
 }  
 return(action)  
}  
  
EpsilonGreedyPolicy <- function(x, y, epsilon){  
 # Generate a random numb  
 rand\_num <- runif(1)  
 if (rand\_num < epsilon){  
 #select a random action  
 action <- sample(1:4, 1)  
 } else {  
 # use the greedy policy  
 action <- GreedyPolicy(x, y)  
 }  
 return(action)  
}  
  
transition\_model <- function(x, y, action, beta){  
 delta <- sample(-1:1, size = 1, prob = c(0.5\*beta,1-beta,0.5\*beta))  
 final\_action <- ((action + delta + 3) %% 4) + 1  
 foo <- c(x,y) + unlist(action\_deltas[final\_action])  
 foo <- pmax(c(1,1),pmin(foo,c(H,W)))  
 return (foo)  
}  
  
  
SARSA <- function(start\_state, epsilon = 0.5, alpha = 0.1, gamma = 0.95, beta = 0){  
 x <- start\_state[1]  
 y <- start\_state[2]  
 episode\_reward <- 0  
 episode\_correction <- 0  
 iter = 0  
 repeat{  
 action <- EpsilonGreedyPolicy(x, y, epsilon)  
 next\_state <- transition\_model(x, y, action, beta)  
 x\_new <- next\_state[1]  
 y\_new <- next\_state[2]  
 R <- reward\_map[x\_new, y\_new]  
 Q\_SA <- q\_table[x, y, action]  
 next\_action = EpsilonGreedyPolicy(x\_new, y\_new, epsilon)  
 Q\_SA\_prime <- q\_table[x\_new, y\_new, next\_action]  
 TD\_correction <- ifelse(R==0,-1,R) + gamma \* Q\_SA\_prime - Q\_SA  
 episode\_correction <- episode\_correction + TD\_correction  
 q\_table[x, y, action] <<- Q\_SA + alpha \* TD\_correction  
 episode\_reward <- episode\_reward + ifelse(R==0,-1,R)  
 x <- x\_new  
 y <- y\_new  
 if (R != 0){  
 return (c(episode\_reward-iter, episode\_correction))  
 } else {  
 iter = iter +1  
 }  
 }  
}  
  
  
q\_learning <- function(start\_state, epsilon = 0.5, alpha = 0.1, gamma = 0.95, beta = 0){  
 x <- start\_state[1]  
 y <- start\_state[2]  
 episode\_reward <- 0  
 episode\_correction <- 0  
 repeat{  
 action <- EpsilonGreedyPolicy(x, y, epsilon)  
 next\_state <- transition\_model(x, y, action, beta)  
 x\_new <- next\_state[1]  
 y\_new <- next\_state[2]  
 R <- reward\_map[x\_new, y\_new]  
 Q\_SA <- q\_table[x, y, action]  
 max\_QSAprime <- max(q\_table[x\_new, y\_new, ])  
 TD\_correction <- R + gamma \* max\_QSAprime - Q\_SA  
 episode\_correction <- episode\_correction + TD\_correction  
 q\_table[x, y, action] <<- Q\_SA + alpha \* TD\_correction  
 episode\_reward <- episode\_reward + R  
 x <- x\_new  
 y <- y\_new  
 if (R != 0){  
 return (c(episode\_reward, episode\_correction))  
 }  
 }  
}

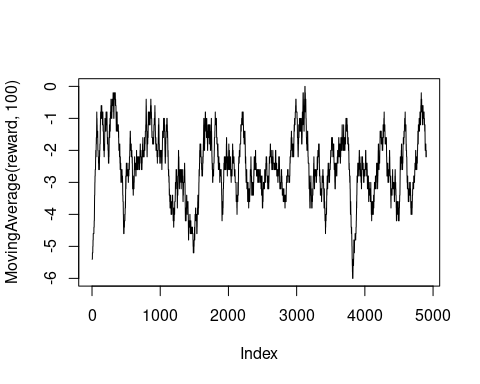
H <- 3  
W <- 6  
  
MovingAverage <- function(x, n){  
 cx <- c(0,cumsum(x))  
 rsum <- (cx[(n+1):length(cx)] - cx[1:(length(cx) - n)]) / n  
 return (rsum)  
}  
  
reward\_map <- matrix(0, nrow = H, ncol = W)  
reward\_map[1,2:5] <- -10  
reward\_map[1,6] <- 10  
  
q\_table <- array(0,dim = c(H,W,4))  
vis\_environment()



reward <- NULL  
   
   
#----Q-learning-----#   
 for(i in 1:5000) {  
 foo <- q\_learning(epsilon = 0.5, gamma = 1, start\_state = c(1,1))  
 reward <- c(reward,foo[1])  
 }  
  
vis\_environment(i, epsilon = 0.5, gamma = 1, alpha = 0.1, beta = 0, method = "q-learning")



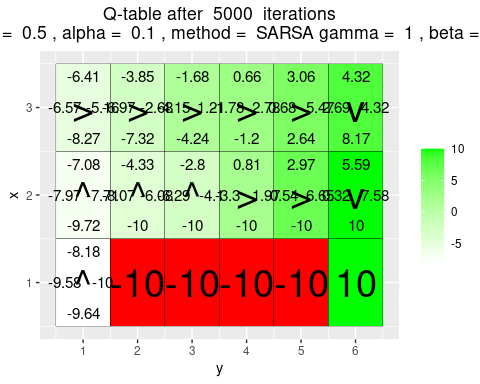
plot(MovingAverage(reward,100),type = "l")



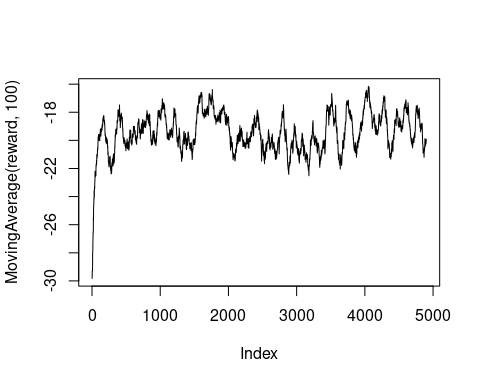
q\_table

## , , 1  
##   
## [,1] [,2] [,3] [,4] [,5] [,6]  
## [1,] 10 0 0 0.00000 0.000000 0.000000  
## [2,] 10 10 10 10.00000 10.000000 10.000000  
## [3,] 10 10 10 9.99999 9.998872 9.997823  
##   
## , , 2  
##   
## [,1] [,2] [,3] [,4] [,5] [,6]  
## [1,] -10 0 0 0.000000 0.000000 0.000000  
## [2,] 10 10 10 10.000000 10.000000 10.000000  
## [3,] 10 10 10 9.999974 9.999522 9.979197  
##   
## , , 3  
##   
## [,1] [,2] [,3] [,4] [,5] [,6]  
## [1,] 10 0 0 0 0 0  
## [2,] 10 -10 -10 -10 -10 10  
## [3,] 10 10 10 10 10 10  
##   
## , , 4  
##   
## [,1] [,2] [,3] [,4] [,5] [,6]  
## [1,] 10 0 0 0.000000 0.000000 0.000000  
## [2,] 10 10 10 10.000000 10.000000 10.000000  
## [3,] 10 10 10 9.999951 9.995896 9.982877

q\_table <- array(0,dim = c(H,W,4))  
reward <- NULL  
  
#---SARSA-----#   
 for(i in 1:5000) {  
 foo <- SARSA(epsilon = 0.5, gamma = 1, start\_state = c(1,1))  
 reward <- c(reward,foo[1])  
 }  
  
vis\_environment(i, epsilon = 0.5, gamma = 1, alpha = 0.1, beta = 0, method = "SARSA")



plot(MovingAverage(reward,100),type = "l")



q\_table

## , , 1  
##   
## [,1] [,2] [,3] [,4] [,5] [,6]  
## [1,] -8.178293 0.000000 0.000000 0.0000000 0.000000 0.000000  
## [2,] -7.083651 -4.326065 -2.802639 0.8137225 2.971831 5.591378  
## [3,] -6.408131 -3.848953 -1.679205 0.6631034 3.061932 4.322406  
##   
## , , 2  
##   
## [,1] [,2] [,3] [,4] [,5] [,6]  
## [1,] -10.000000 0.000000 0.000000 0.000000 0.000000 0.000000  
## [2,] -7.709918 -6.081099 -4.098580 1.965215 6.654653 7.580558  
## [3,] -5.160786 -2.682358 1.206189 2.776293 5.474450 4.319636  
##   
## , , 3  
##   
## [,1] [,2] [,3] [,4] [,5] [,6]  
## [1,] -9.638279 0.000000 0.000000 0.000000 0.000000 0.00000  
## [2,] -9.719517 -10.000000 -10.000000 -10.000000 -10.000000 10.00000  
## [3,] -8.265435 -7.318613 -4.238676 -1.196285 2.644619 8.17455  
##   
## , , 4  
##   
## [,1] [,2] [,3] [,4] [,5] [,6]  
## [1,] -9.576866 0.000000 0.000000 0.000000 0.0000000 0.0000000  
## [2,] -7.972233 -8.073171 -6.293719 -3.296149 -0.5415629 0.3244004  
## [3,] -6.570807 -6.971410 -4.154077 -1.781769 0.6785314 2.6850013