TDDE15-Exam Oct 2022

Fredrik Ramberg

Task 1

```
library(bnlearn)
library(gRain)
```

```
## Loading required package: gRbase
##
## Attaching package: 'gRbase'
## The following objects are masked from 'package:bnlearn':
##
##     ancestors, children, nodes, parents
```

```
data("asia")

set.seed(12345)

# n <- nrow(asia)
# train_indices <- sample(1:n, size = round(0.8 * n))
#
# train_data <- asia[train_indices, ]
# test_data <- asia[-train_indices, ]

tr <- asia[1:10,]
te <- asia[11:5000,]
te <- te[,-5]
te <- te[,-5]

true_dag <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

#Learning the parameters
bn <- bn.fit(true_dag, tr)
```

```
## Warning in check.data(data, allow.missing = TRUE): variable A in the data has
## levels that are not observed in the data.
## Warning in check.data(data, allow.missing = TRUE): variable L in the data has
## levels that are not observed in the data.
```

```
bn_comp <- compile(as.grain(bn))
```

```
## Warning in from.bn.fit.to.grain(x): NaN conditional probabilities in D,
## replaced with a uniform distribution.
## Warning in from.bn.fit.to.grain(x): NaN conditional probabilities in E,
## replaced with a uniform distribution.
```

```
## Warning in from.bn.fit.to.grain(x): NaN conditional probabilities in T,
## replaced with a uniform distribution.
```

```
# Imputing the values
nodes <- colnames(te)
predicted_B <- c()
predicted_E <- c()
for (i in 11:5000){
  states <- te[i,]

  # Posterior probability for "yes" and "no" for each datapoint
  posterior <- querygrain(setEvidence(bn_comp, nodes = nodes, states = states), nodes = c("B", "E"))

  # Classification based on the posterior probability
  pred_B <- ifelse(posterior$B["yes"] >= posterior$B["no"], "yes", "no")
  pred_E <- ifelse(posterior$E["yes"] >= posterior$E["no"], "yes", "no")

  # Store the predicted class
  predicted_B <- c(predicted_B, pred_B)
  predicted_E <- c(predicted_E, pred_E)
}



##all data
####################################
bn <- bn.fit(true_dag,asia)
bn <- compile(as.grain(bn))
####################################

#Learning the parameters with imputed data
te_imputed <- cbind(te,data.frame(B = predicted_B))
te_imputed <- cbind(te_imputed,data.frame(E = predicted_E))
asia_imputed <- rbind(tr, te_imputed)

bn_imputed <- bn.fit(true_dag, asia_imputed)
bn_imputed <- compile(as.grain(bn_imputed))
```

```
## Warning in from.bn.fit.to.grain(x): NaN conditional probabilities in D,
## replaced with a uniform distribution.
```

```
print("Estimated on 10 values:")
```

```
## [1] "Estimated on 10 values:"
```

```
print(bn_comp$cptlist$D)
```

```
## , , E = no
##
##      B
## D      no yes
##   no  0.5   0
##   yes 0.5   1
##
## , , E = yes
##
```

```
##      B
## D     no yes
##   no   0 0.5
##   yes  1 0.5
```

```r
print("Estimated on imputed values:")
```

```
## [1] "Estimated on imputed values:"
```

```r
print(bn_imputed$cptlist$D)
```

```
## , , E = no
##
##      B
## D      no       yes
##   no  0.5 0.5301301
##   yes 0.5 0.4698699
##
## , , E = yes
##
##      B
## D     no yes
##   no   0 0.5
##   yes  1 0.5
```

```r
####################################
print("All Asia dataset:")
```

```
## [1] "All Asia dataset:"
```

```r
print(bn$cptlist$D)
```

```
## , , E = no
##
##      B
## D             no       yes
##   no  0.90017286 0.2137306
##   yes 0.09982714 0.7862694
##
## , , E = yes
##
##      B
## D            no       yes
##   no  0.2773723 0.1459227
##   yes 0.7226277 0.8540773
```

```r
####################################
```

The conditional distribution from D obtained from the 10 first data points is closer to the true one, we can clearly see this by calculating it based on all the asia dataset (which is a much better estimate).

Considering that the imputed data (row 11:5000) only contains B = "yes" and E="No" this will bias the results, . In comparison with the 10 row data we achieve a posterior based on varied (but limited) data. Essentially the extra data will only update the params for E = "no" and B="yes" with heavily biased data.

```
E = no
     B
D      yes
  no     0
```

```
  yes     1

E = no
     B
D        yes
  no  0.5301301
  yes 0.4698699
```

Task 2

- 5 sectors
- emission = [i-1, i+1]
- ** Must spend at least
    - 2 timesteps in sector 1
    - 3 timesteps in sector 2
    - 2 timesteps in sector 3
    - 1 timesteps in sector 4
    - 2 timesteps in sector 5
    -

```
library(HMM)

#states = hidden states (nr = 10)
states <- c("1a", "1b", "2a", "2b", "2c","3a", "3b", "4a", "5a", "5b")

#symbols = observations
symbols <- c(1:5)


transProbs <- c(0.5,  0,  0,  0,  0,  0,  0,  0,  0,0.5,
                0.5,0.5,  0,  0,  0,  0,  0,  0,  0,  0,
                  0,0.5,0.5,  0,  0,  0,  0,  0,  0,  0,
                  0,  0,0.5,0.5,  0,  0,  0,  0,  0,  0,
                  0,  0,  0,0.5,0.5,  0,  0,  0,  0,  0,
                  0,  0,  0,  0,0.5,0.5,  0,  0,  0,  0,
                  0,  0,  0,  0,  0,0.5,0.5,  0,  0,  0,
                  0,  0,  0,  0,  0,  0,0.5,0.5,  0,  0,
                  0,  0,  0,  0,  0,  0,  0,0.5,0.5,  0,
                  0,  0,  0,  0,  0,  0,  0,  0,0.5,0.5)
##### Old
# New
transProbs <- matrix(transProbs,nrow = 10, ncol = 10)

emissionProbs <- c(1/3, 1/3,   0,   0, 1/3,
                   1/3, 1/3,   0,   0, 1/3,
                   1/3, 1/3, 1/3,   0,   0,
                   1/3, 1/3, 1/3,   0,   0,
                   1/3, 1/3, 1/3,   0,   0,
                     0, 1/3, 1/3, 1/3,   0,
                     0, 1/3, 1/3, 1/3,   0,
```

```r
                    0,    0, 1/3, 1/3, 1/3,
                  1/3,    0,    0, 1/3, 1/3,
                  1/3,    0,    0, 1/3, 1/3)

startProbs <- rep(0.1, 10)

######## Symbols
# States
emissionProbs <- matrix(emissionProbs,ncol = 5, nrow = 10, byrow = TRUE)


hmm <- initHMM(states,symbols, transProbs = transProbs, emissionProbs = emissionProbs, startProbs = sta

transProbs
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]  0.5  0.5  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0
##  [2,]  0.0  0.5  0.5  0.0  0.0  0.0  0.0  0.0  0.0   0.0
##  [3,]  0.0  0.0  0.5  0.5  0.0  0.0  0.0  0.0  0.0   0.0
##  [4,]  0.0  0.0  0.0  0.5  0.5  0.0  0.0  0.0  0.0   0.0
##  [5,]  0.0  0.0  0.0  0.0  0.5  0.5  0.0  0.0  0.0   0.0
##  [6,]  0.0  0.0  0.0  0.0  0.0  0.5  0.5  0.0  0.0   0.0
##  [7,]  0.0  0.0  0.0  0.0  0.0  0.0  0.5  0.5  0.0   0.0
##  [8,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.5  0.5   0.0
##  [9,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.5   0.5
## [10,]  0.5  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.5
```

```r
emissionProbs
```

```
##            [,1]      [,2]      [,3]      [,4]      [,5]
##  [1,] 0.3333333 0.3333333 0.0000000 0.0000000 0.3333333
##  [2,] 0.3333333 0.3333333 0.0000000 0.0000000 0.3333333
##  [3,] 0.3333333 0.3333333 0.3333333 0.0000000 0.0000000
##  [4,] 0.3333333 0.3333333 0.3333333 0.0000000 0.0000000
##  [5,] 0.3333333 0.3333333 0.3333333 0.0000000 0.0000000
##  [6,] 0.0000000 0.3333333 0.3333333 0.3333333 0.0000000
##  [7,] 0.0000000 0.3333333 0.3333333 0.3333333 0.0000000
##  [8,] 0.0000000 0.0000000 0.3333333 0.3333333 0.3333333
##  [9,] 0.3333333 0.0000000 0.0000000 0.3333333 0.3333333
## [10,] 0.3333333 0.0000000 0.0000000 0.3333333 0.3333333
```

```r
set.seed(12345)
simHMM(hmm, 100)
```

```
## $states
##   [1] "5a" "5a" "5a" "5a" "5b" "1a" "1b" "1b" "1b" "1b" "2a" "2a" "2b" "2b" "2b"
##  [16] "2b" "2b" "2b" "2b" "2c" "3a" "3a" "3b" "4a" "5a" "5b" "5b" "5b" "1a" "1b"
##  [31] "1b" "2a" "2a" "2b" "2b" "2b" "2c" "2c" "2c" "3a" "3b" "3b" "4a" "5a" "5b"
##  [46] "1a" "1b" "2a" "2a" "2b" "2c" "2c" "3a" "3a" "3b" "3b" "4a" "4a" "4a" "4a"
##  [61] "5a" "5b" "5b" "5b" "5b" "1a" "1a" "1b" "1b" "1b" "1b" "1b" "2a" "2a" "2a"
##  [76] "2b" "2c" "2c" "2c" "3a" "3b" "4a" "4a" "4a" "4a" "4a" "5a" "5a" "5a" "5b"
##  [91] "5b" "5b" "1a" "1a" "1a" "1a" "1a" "1a" "1b" "2a"
##
## $observation
##   [1] 1 5 4 5 5 2 5 2 1 1 3 3 2 3 2 2 2 3 1 1 3 3 2 4 4 4 4 1 1 2 1 2 2 3 2 2 2
##  [38] 1 1 4 3 4 5 5 1 2 2 3 2 3 2 3 4 3 2 4 4 3 4 3 4 5 4 4 5 5 2 1 2 5 2 1 2 1
```

```
##  [75] 3 2 2 2 2 4 2 3 5 5 4 4 5 4 1 1 4 4 1 1 2 1 1 2 1 3
```

Task 3

From labs:

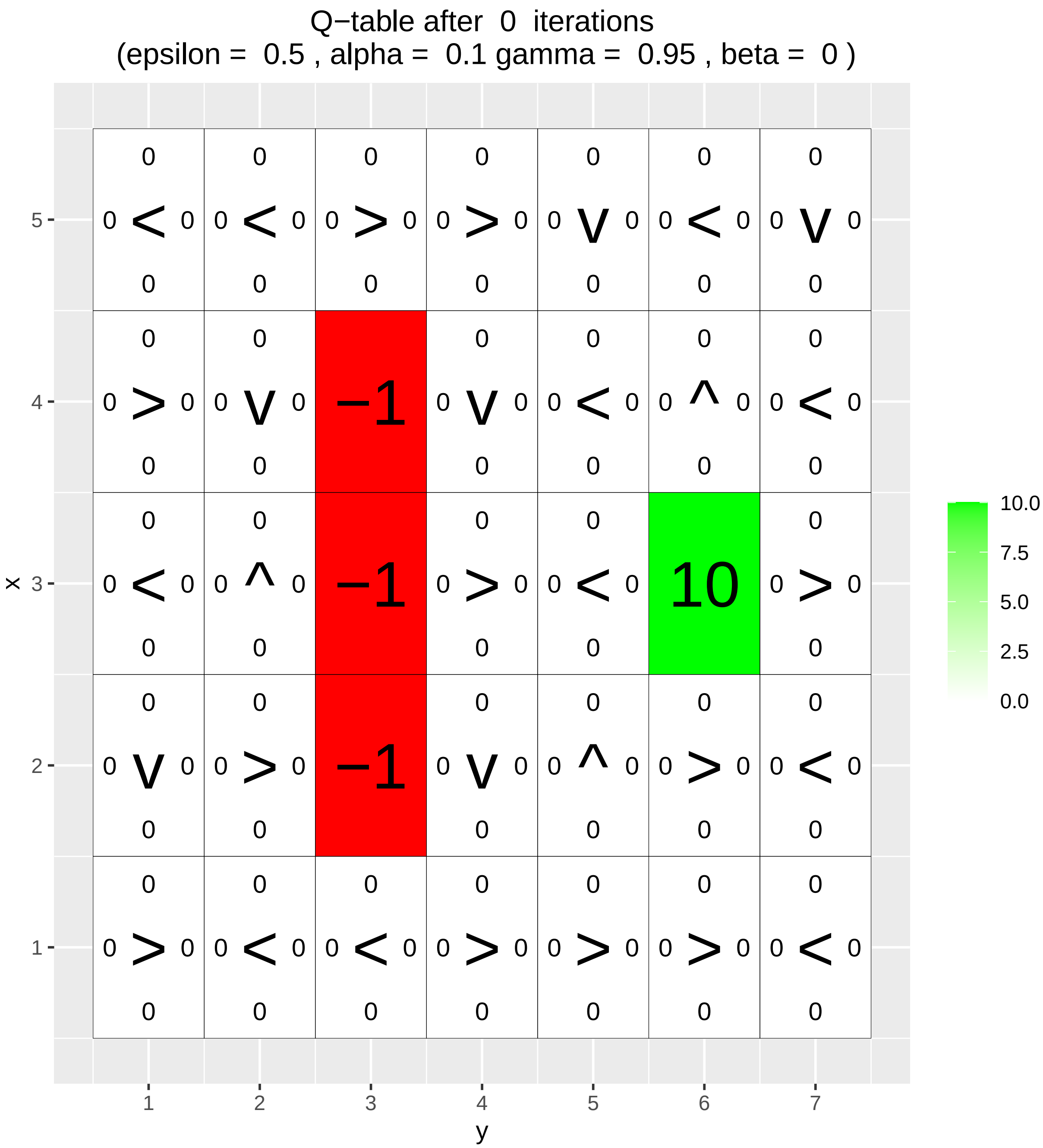
```r
library(ggplot2)

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){

  # Visualize an environment with rewards.
  # Q-values for all actions are displayed on the edges of each tile.
  # The (greedy) policy for each state is also displayed.
  #
  # Args:
  #   iterations, epsilon, alpha, gamma, beta (optional): for the figure title.
  #   reward_map (global variable): a HxW array containing the reward given at each state.
  #   q_table (global variable): a HxWx4 array containing Q-values for each state-action pair.
  #   H, W (global variables): environment dimensions.

  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,"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 a greedy action for state (x,y) from q_table.
  #
  # Args:
  #   x, y: state coordinates.
  #   q_table (global variable): a HxWx4 array containing Q-values for each state-action pair.
  #
  # Returns:
  #   An action, i.e. integer in {1,2,3,4}.

  # Your code here.
  max <- c(which(q_table[x,y,] == max(q_table[x,y,])))
  if(length(max) > 1) {
    max <- sample(max, size = 1)
  }

  return (max)

  # max <- which.max(q_table[x,y,])#which(q_table[x,y,] == max(q_table[x,y,]))
  #
  # return (max)#sample(max, size = 1))

}

EpsilonGreedyPolicy <- function(x, y, epsilon){

  # Get an epsilon-greedy action for state (x,y) from q_table.
  #
  # Args:
  #   x, y: state coordinates.
  #   epsilon: probability of acting randomly.
  #
  # Returns:
  #   An action, i.e. integer in {1,2,3,4}.

  # Your code here.

  max <- GreedyPolicy(x,y)

  epsilon_prob <- runif(1)
  if (epsilon_prob <= epsilon){
    max <-sample(1:4,1)
  }

  return(max)
```

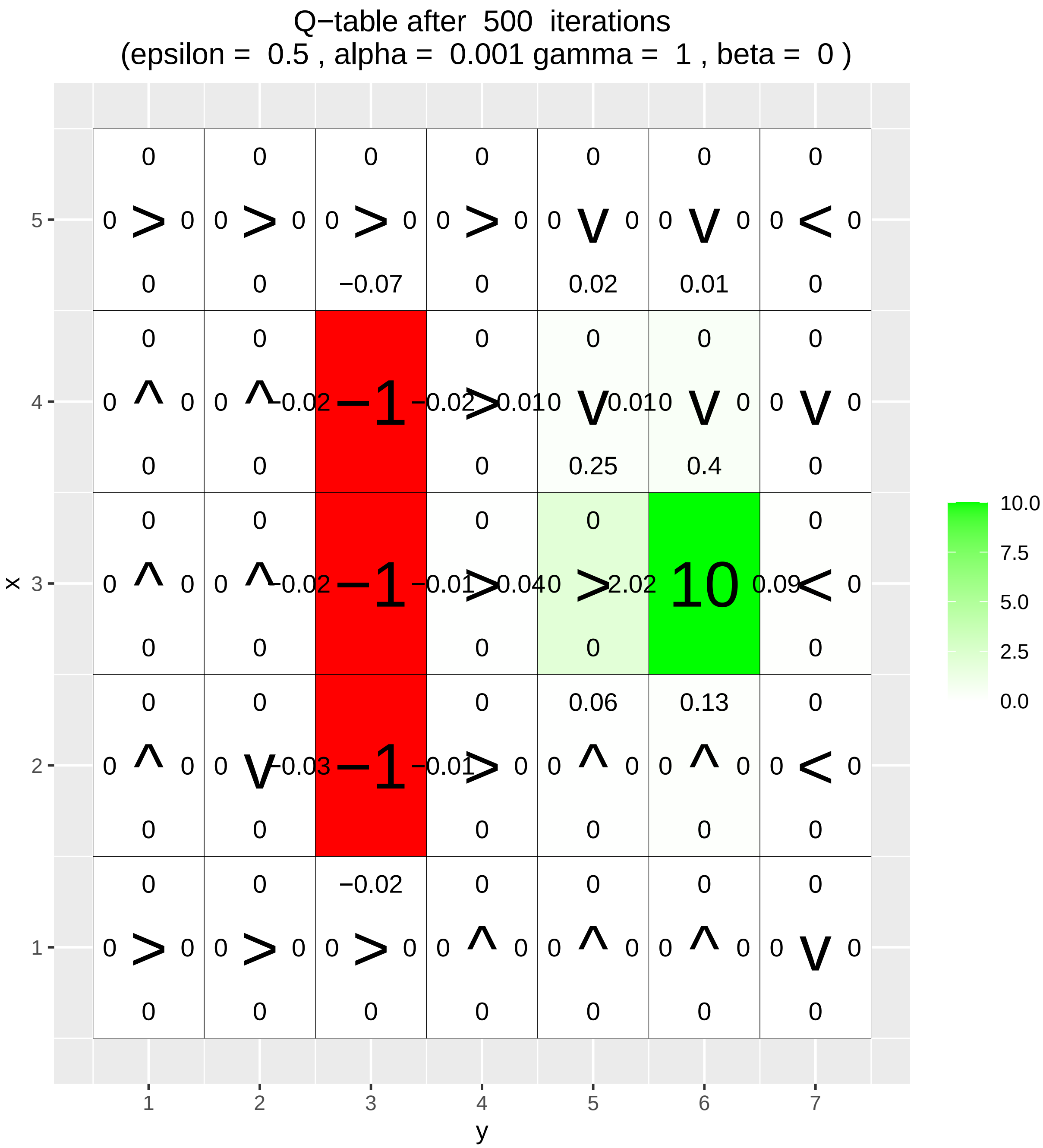
```r
  # If i want to put in a validity constraint
  # repeat{
  #   new_state <- c(x,y) + unlist(action_deltas[max])
  #   xn <- new_state[1]
  #   yn <- new_state[2]
  #   if(!(yn < 1 || xn < 1 || yn > W || xn > H)){
  #     return(max)
  #   }
  # }

}

transition_model <- function(x, y, action, beta){

  # Computes the new state after given action is taken. The agent will follow the action
  # with probability (1-beta) and slip to the right or left with probability beta/2 each.
  #
  # Args:
  #   x, y: state coordinates.
  #   action: which action the agent takes (in {1,2,3,4}).
  #   beta: probability of the agent slipping to the side when trying to move.
  #   H, W (global variables): environment dimensions.
  #
  # Returns:
  #   The new state after the action has been taken.

  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)
}

q_learning <- function(start_state, epsilon = 0.5, alpha = 0.1, gamma = 0.95,
                       beta = 0){

  # Perform one episode of Q-learning. The agent should move around in the
  # environment using the given transition model and update the Q-table.
  # The episode ends when the agent reaches a terminal state.
  #
  # Args:
  #   start_state: array with two entries, describing the starting position of the agent.
  #   epsilon (optional): probability of acting randomly.
  #   alpha (optional): learning rate.
  #   gamma (optional): discount factor.
  #   beta (optional): slipping factor.
  #   reward_map (global variable): a HxW array containing the reward given at each state.
  #   q_table (global variable): a HxWx4 array containing Q-values for each state-action pair.
  #
  # Returns:
  #   reward: reward received in the episode.
  #   correction: sum of the temporal difference correction terms over the episode.
```

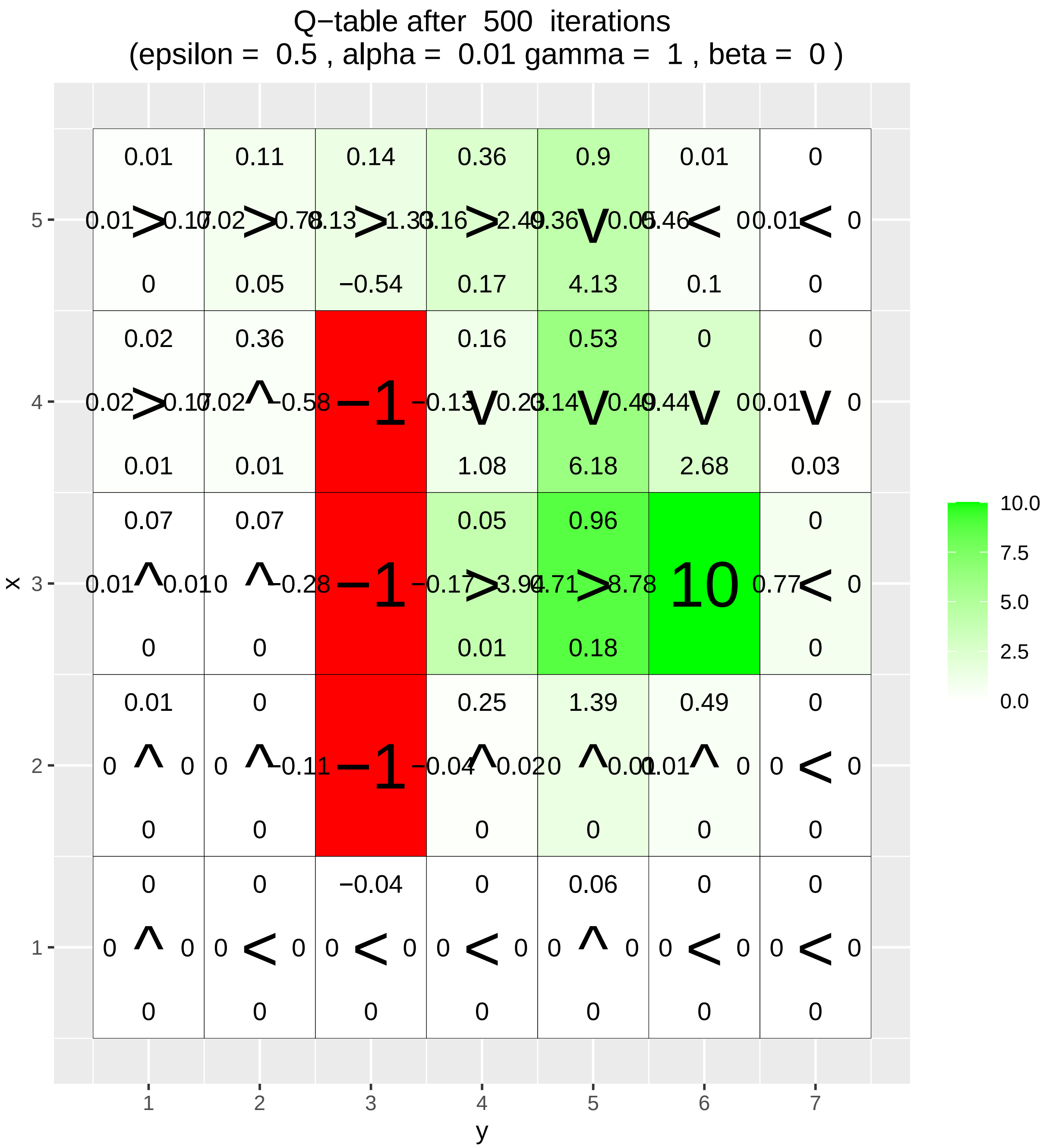
```
  #   q_table (global variable): Recall that R passes arguments by value. So, q_table being
  #   a global variable can be modified with the superassigment operator <<-.

  # Your code here.

  # h <- sample(1:nrow(reward_map), size=1)
  # w <- sample(1:ncol(reward_map), size=1)
  # state <- c(h,w)
  state <- start_state
  episode_correction <- 0
  episode_reward <- 0
  repeat{
    # Follow policy, execute action, get reward.
    action <- EpsilonGreedyPolicy(x = state[1], y = state[2], epsilon = epsilon)
    next_state <- transition_model(x = state[1], y = state[2], action, beta)
    reward <- reward_map[next_state[1], next_state[2]]

    # Q-table update.
    temp_diff <- alpha*(reward + gamma * max(q_table[next_state[1], next_state[2],]) -
                        q_table[state[1], state[2], action])
    q_table[state[1], state[2], action] <<- q_table[state[1], state[2], action] + temp_diff

    episode_correction <- episode_correction + temp_diff
    episode_reward <- episode_reward + reward

    if(reward!=0)
      # End episode.
      return (c(episode_reward,episode_correction))
    state <- next_state
  }

}
```

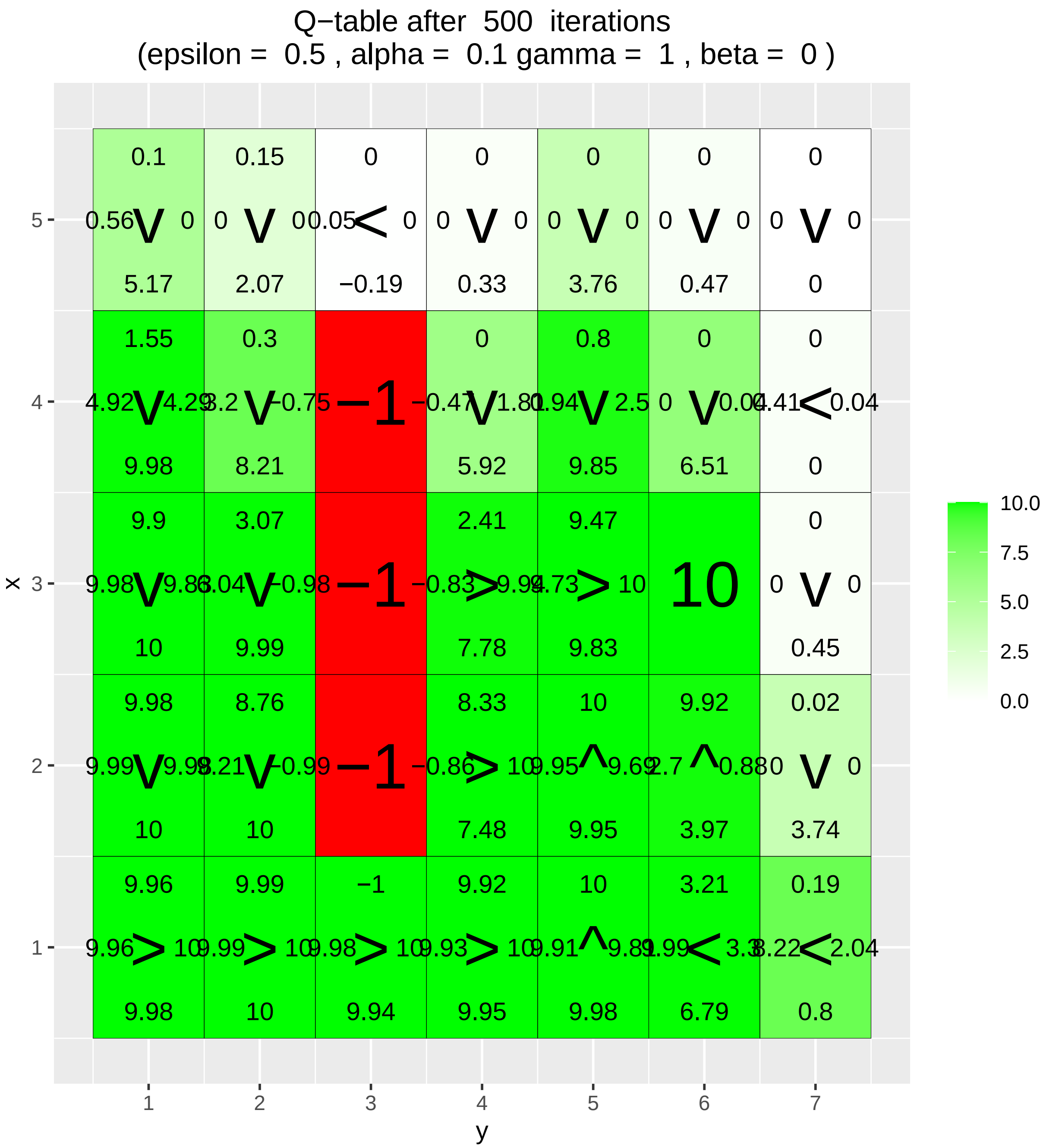
```
#####################################################################################################
# Q-Learning Environments
#####################################################################################################
set.seed(12345)
# Environment A (learning)
H <- 5
W <- 7

reward_map <- matrix(0, nrow = H, ncol = W)
reward_map[3,6] <- 10
reward_map[2:4,3] <- -1

alphas <- c(0.001, 0.01, 0.1)

q_table <- array(0,dim = c(H,W,4))

vis_environment()
```




```r
for (a in alphas){
  q_table <- array(0,dim = c(H,W,4))
  for(i in 1:500){
    foo <- q_learning(start_state = c(3,1), gamma = 1, alpha = a)

    # if(any(i==c(10,100,1000,10000)))
    #   vis_environment(i)
  }
  vis_environment(500, gamma=1, alpha = a)
}
```


Q−table after 500 iterations
(epsilon = 0.5 , alpha = 0.001 gamma = 1 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0


Q−table after 500 iterations
(epsilon = 0.5 , alpha = 0.01 gamma = 1 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0



We can see clearly that the model learns better paths for higher alpha. Since alpha effects directly how much the q_table is effected we can see that the values of the q_table is much lower for lower alpha values. Although the model is not trained on only 500 episodes it would discover a closer to optimal policy for alpha = 0.1 for higher iterations such as in the labs. The lower value alphas might also lead to an optimal policy but it would take longer to converge.

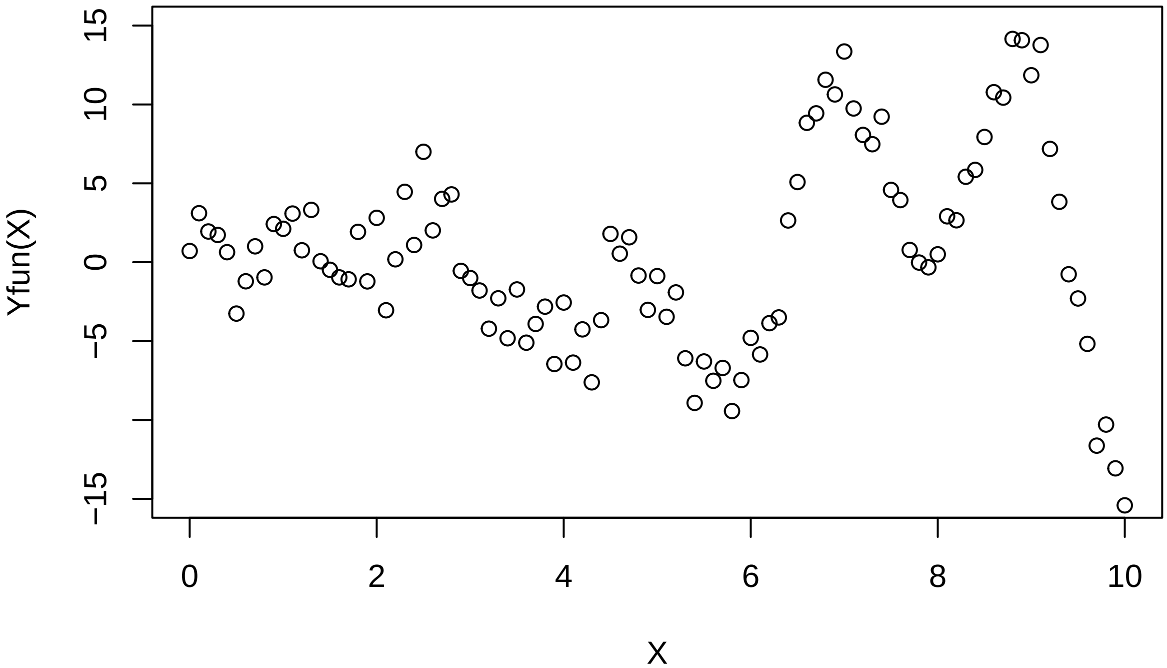
Task 4

4.1

```
library(kernlab)
```

```
##
## Attaching package: 'kernlab'
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
X<-seq(0,10,.1)
Yfun<-function(x){
  return (x*(sin(x)+sin(3*x))+rnorm(length(x),0,2))
  }
plot(X,Yfun(X),xlim=c(0,10),ylim=c(-15,15))
```




```
#nested Square Exponetial Kernel
nestedSEK <- function(sigmaF=1,l=3) {
  fixedSEK <- function(x1,x2){
    n1 <- length(x1)
    n2 <- length(x2)
    K <- matrix(NA,n1,n2)
    for (i in 1:n2){
      K[,i] <- sigmaF^2*exp(-0.5*( (x1-x2[i])/l)^2 )
    }
    return(K)
  }
  class(fixedSEK) <- 'kernel'
  return(fixedSEK)
}

sigmaNoise <- 2

#Chosing an ell
ell <- 0.01

#setting hyperparameters in kernel function
```

```
SEK <- nestedSEK(sigmaF = 0.5, l = ell)

modelGP <- gausspr(X, Yfun(X), scaled = FALSE,kernel = SEK, var = sigmaNoise^2, variance.model = TRUE)


posteriorMean <- predict(modelGP, X)
sd <- predict(modelGP, X, type="sdeviation")

upper <- posteriorMean + 1.96 * sd
lower <- posteriorMean - 1.96 * sd

plot(x= X, y = Yfun(X),
     xlab = "time", ylab = "temp", main = "Temperature predictions", lwd = 1.5)
lines(x=X, y = posteriorMean, col = "red", lwd = 3)
lines(X, upper, col = "blue", lwd = 1)
lines(X, lower, col = "blue", lwd = 1)
legend("bottomright", legend=c("Data", "Predictions", "Confidence Interval"),
       pch=c(1, NA, NA), lty=c(NA, 1, 1), lwd=c(NA, 2, 2), col=c("black", "red", "blue"))
```