oct2021

2024-10-25

Graphical Models

```r
# install.packages("bnlearn")
set.seed(12345)
library(bnlearn)
library(gRain)
```

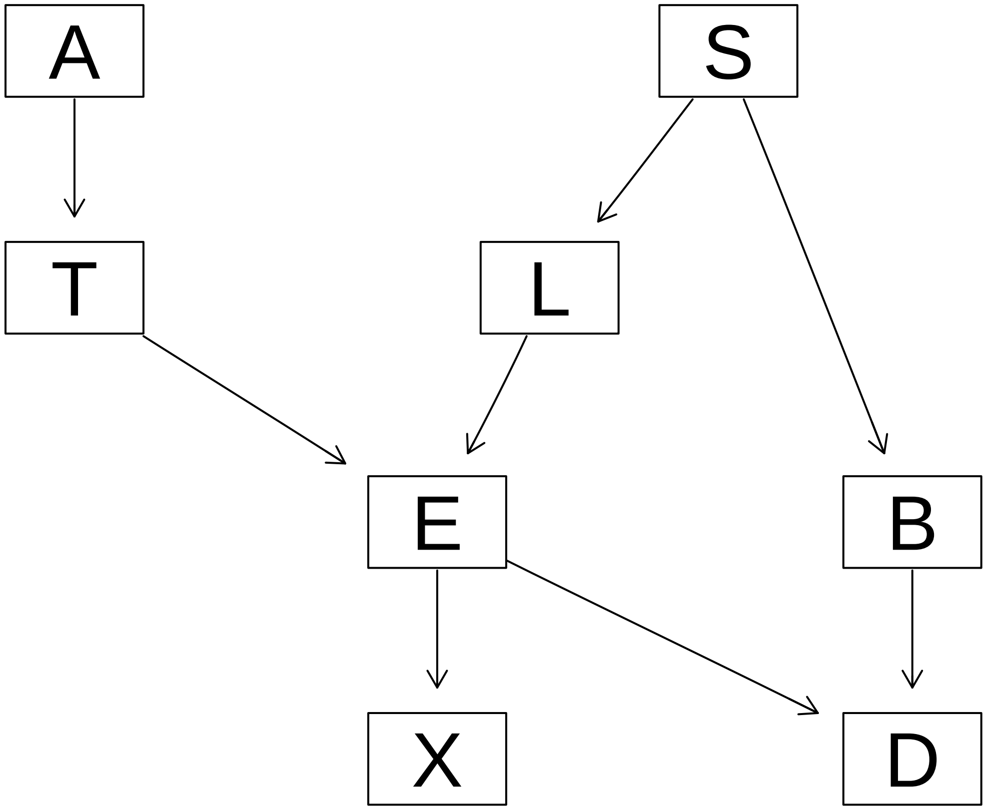
```
## Loading required package: gRbase
```

```
##
## Attaching package: 'gRbase'
```

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

```r
data("asia")
# The true Aisian Network
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
graphviz.plot(dag)
```

```
## Loading required namespace: Rgraphviz
```




```r
#Fit the BN structure using maximum likelihood estimators
fitted = bn.fit(dag, asia, method = "bayes")

# Fit as grain
grain_fit = as.grain(fitted)
compiled_grain = compile(grain_fit)

sampledData = matrix(NA, nrow = 1000, ncol = 8)
colNames = c("A","S","T","L","E","B","X","D")

for (i in 1:1000) {
  A = sample(c("yes", "no"), 1, prob = fitted$A$prob)
  S = sample(c("yes", "no"), 1, prob = fitted$S$prob)
  T = sample(c("yes", "no"), 1, prob = fitted$T$prob[,A])
  L = sample(c("yes", "no"), 1, prob = fitted$L$prob[,S])
  E = sample(c("yes", "no"), 1, prob = fitted$E$prob[,L,T])
  B = sample(c("yes", "no"), 1, prob = fitted$B$prob[,S])
  X = sample(c("yes", "no"), 1, prob = fitted$X$prob[,E])
  D = sample(c("yes", "no"), 1, prob = fitted$D$prob[,B,E])
  sampledData[i,] = c(A,S,T,L,E,B,X,D)
}

# Approximate P(S|D=yes)
foo = sampledData[which(sampledData[,8]=="yes"),2]
table(foo)/length(foo)
```

```
## foo
##        no       yes
```

```
## 0.3623188 0.6376812
```

```r
# Exact P(S|D=yes)
obs_evidence = setEvidence(compiled_grain, nodes = c("D"), states = c("yes"))
querygrain(obs_evidence, nodes = "S")$S
```

```
## S
##        no       yes
## 0.3344944 0.6655056
```

Hidden Markov Models (5 p)

```r
set.seed(12345)
library(HMM)

states = c("H", "S1", "S2")
emissionSymbols = c("H", "S")

transitionProb = matrix(c(0.9,0.1,0, # healthy
                          0,  0, 1,   # sick day 1
                          0.2,0,0.8 ),# sick day 2
                        byrow = TRUE, nrow = 3, ncol = 3)

#  print(transitionProb)

emissionProb = matrix(c(0.7, 0.3,
                        0.4, 0.6, # Assuming true positive = sick
                        0.4, 0.6),
                      byrow = TRUE, nrow = 3, ncol = 2)

# print(emissionProb)

startProb = c(0.5,0.5,0)

hmm = initHMM(States = states, Symbols = emissionSymbols,
              startProbs = startProb, transProbs = transitionProb,
              emissionProbs = emissionProb)
nIter = 100
simulation = simHMM(hmm, nIter)
print(simulation)
```

```
## $states
##   [1] "H"  "H"  "H"  "H"  "H"  "H"  "H"  "H"  "H"  "S1" "S2" "S2" "S2" "S2" "S2"
##  [16] "S2" "S2" "S2" "S2" "H"  "H"  "H"  "S1" "S2" "S2" "S2" "S2" "S2" "S2" "S2"
##  [31] "S2" "S2" "S2" "S2" "S2" "S2" "H"  "S1" "S2" "S2" "S2" "S2" "H"  "H"  "H"
##  [46] "H"  "H"  "H"  "H"  "H"  "S1" "S2" "S2" "S2" "S2" "S2" "S2" "S2" "S2" "S2"
##  [61] "S2" "S2" "H"  "H"  "S1" "S2" "H"  "H"  "H"  "S1" "S2" "S2" "S2" "S2" "S2"
##  [76] "S2" "H"  "H"  "H"  "H"  "H"  "H"  "H"  "H"  "H"  "H"  "H"  "H"  "H"  "H"
##  [91] "S1" "S2" "S2" "S2" "S2" "S2" "S2" "S2" "S2" "S2"
##
## $observation
##   [1] "H" "H" "S" "H" "H" "S" "H" "S" "H" "S" "H" "S" "H" "S" "H" "H" "H" "S"
##  [19] "S" "H" "H" "H" "S" "H" "H" "H" "H" "S" "S" "H" "S" "H" "H" "H" "H" "H"
##  [37] "H" "S" "S" "H" "S" "H" "H" "H" "H" "S" "S" "H" "S" "H" "H" "S" "H" "S"
```

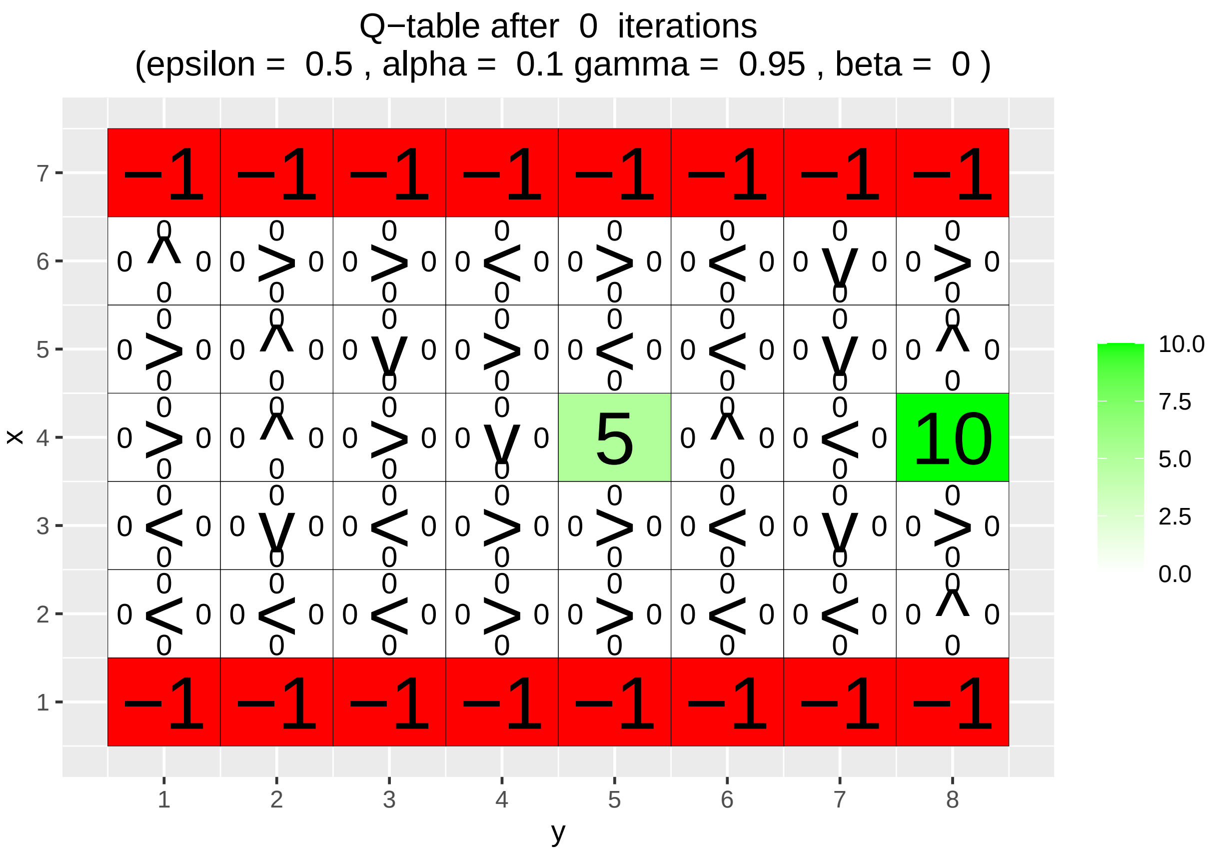
```
## [55] "S" "H" "H" "S" "H" "H" "H" "S" "S" "S" "S" "S" "S" "H" "S" "S" "H" "S"
## [73] "H" "S" "S" "H" "S" "S" "S" "S" "H" "H" "H" "H" "S" "S" "H" "S" "H" "H"
## [91] "H" "H" "S" "S" "H" "S" "S" "H" "S" "S"
```

Reinforcement Learning (5 p)

```r
# install.packages("ggplot2")
# install.packages("vctrs")
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){
  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){
  q_values = q_table[x, y, ]
```

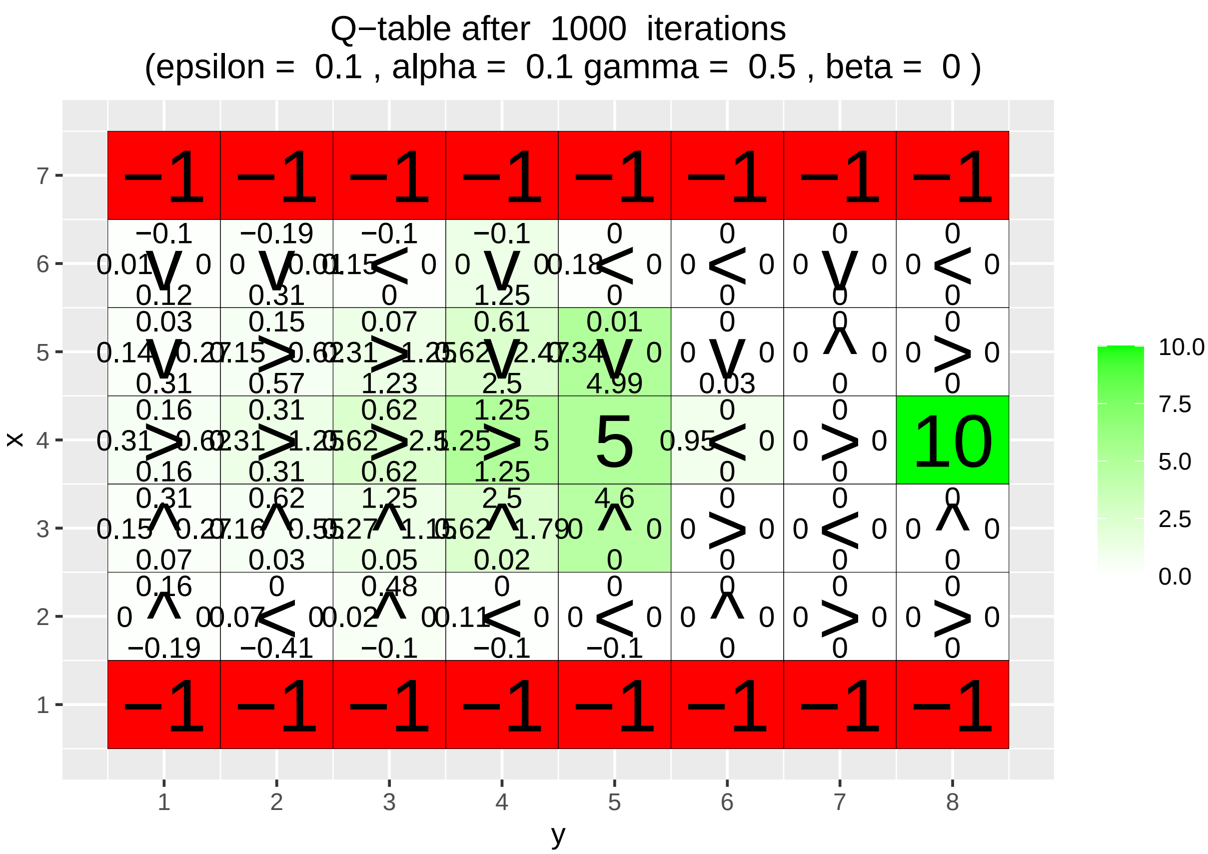
```r
  # Find all actions with the maximum Q-value
  max_actions = which(q_values == max(q_values))
  if (length(max_actions) == 1) {
    return(max_actions)
  } else {
    return(sample(max_actions, 1))
  }

}

EpsilonGreedyPolicy <- function(x, y, epsilon){
  if (runif(1) < epsilon) {
    return (sample(1:4,1))
  } else {
    return (GreedyPolicy(x,y))
  }
}

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)
}

q_learning <- function(start_state, epsilon = 0.5, alpha = 0.1, gamma = 0.95,
                       beta = 0, state = TRUE){
  Q = start_state
  x = Q[1]
  y = Q[2]
  episode_correction = 0
  repeat{

    # Follow policy, execute action, get reward.
    action = EpsilonGreedyPolicy(x,y,epsilon) # follow policy
    next_state = transition_model(x,y,action,beta) # excecute action
    reward = reward_map[next_state[1],next_state[2]] # get reward

    # Q-table update.
    correction = reward + gamma * max(q_table[next_state[1],next_state[2],])-q_table[x,y,action]
    if (state) {
      q_table[x,y,action] <<- q_table[x,y,action] + alpha * (correction)
      episode_correction = episode_correction + correction
    }

    x = next_state[1]
    y = next_state[2]

    if(reward!=0)
      # End episode.
      return (c(reward,episode_correction))
```

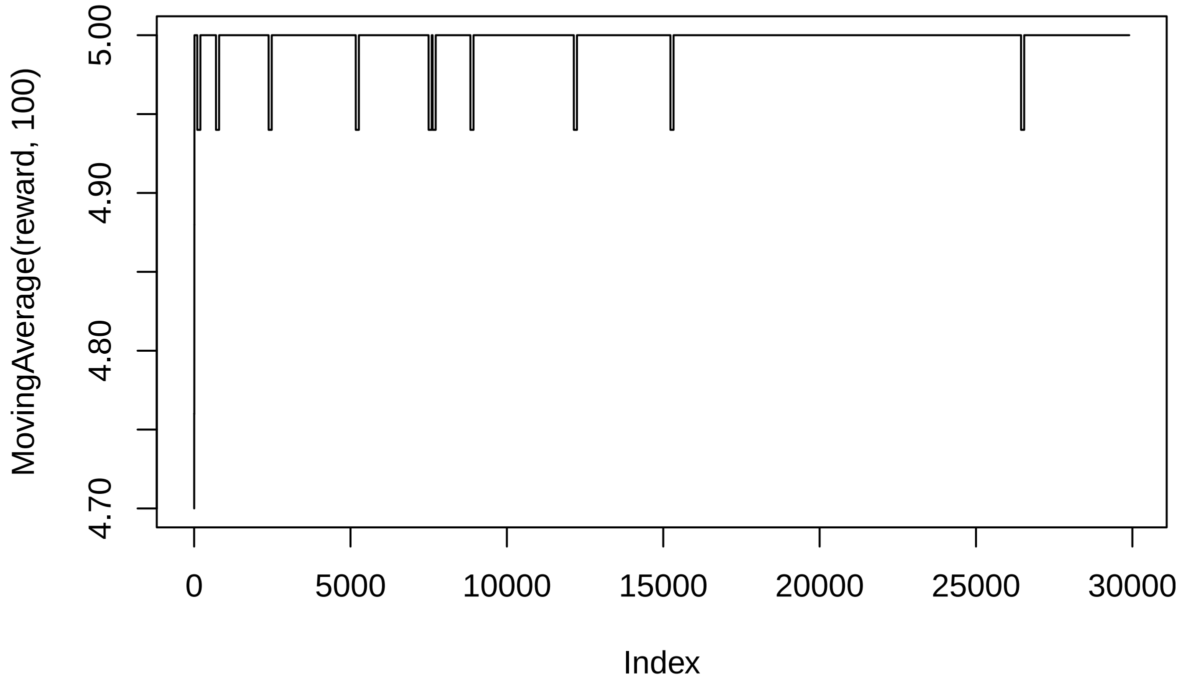
```
  }

}

# Environment B (the effect of epsilon and gamma)

H <- 7
W <- 8

reward_map <- matrix(0, nrow = H, ncol = W)
reward_map[1,] <- -1
reward_map[7,] <- -1
reward_map[4,5] <- 5
reward_map[4,8] <- 10

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

vis_environment()
```



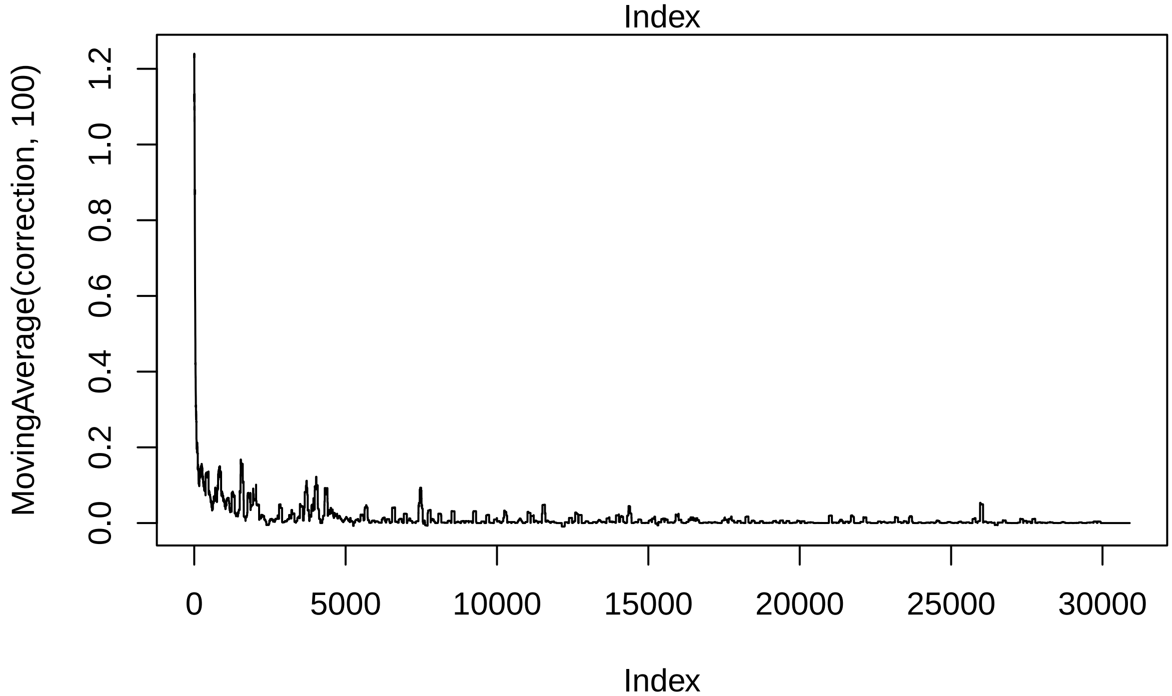

```
MovingAverage <- function(x, n){

  cx <- c(0,cumsum(x))
  rsum <- (cx[(n+1):length(cx)] - cx[1:(length(cx) - n)]) / n

  return (rsum)
}
```

```r
mrew= NULL
for(k in c(0.1, 0.25, 0.5)){ # epsilon
  for(j in c(0.5,0.75,0.95)){ # gamma
    q_table <- array(0,dim = c(H,W,4))
    reward <- NULL
    correction <- NULL

    for(i in 1:30000){
      foo <- q_learning(epsilon = k, gamma = j, start_state = c(4,1))
      reward <- c(reward,foo[1])
      correction <- c(correction,foo[2])
    }

    for(i in 1:1000){
      foo <- q_learning(epsilon = 0, gamma = j, start_state = c(4,1), state = FALSE)
      reward2 <- c(reward,foo[1])
      correction <- c(correction,foo[2])
    }

    vis_environment(i, epsilon = k, gamma = j)
    plot(MovingAverage(reward,100),type = "l")
    plot(MovingAverage(correction,100),type = "l")
    mrew = c(mrew, mean(reward2))
  }
}
```

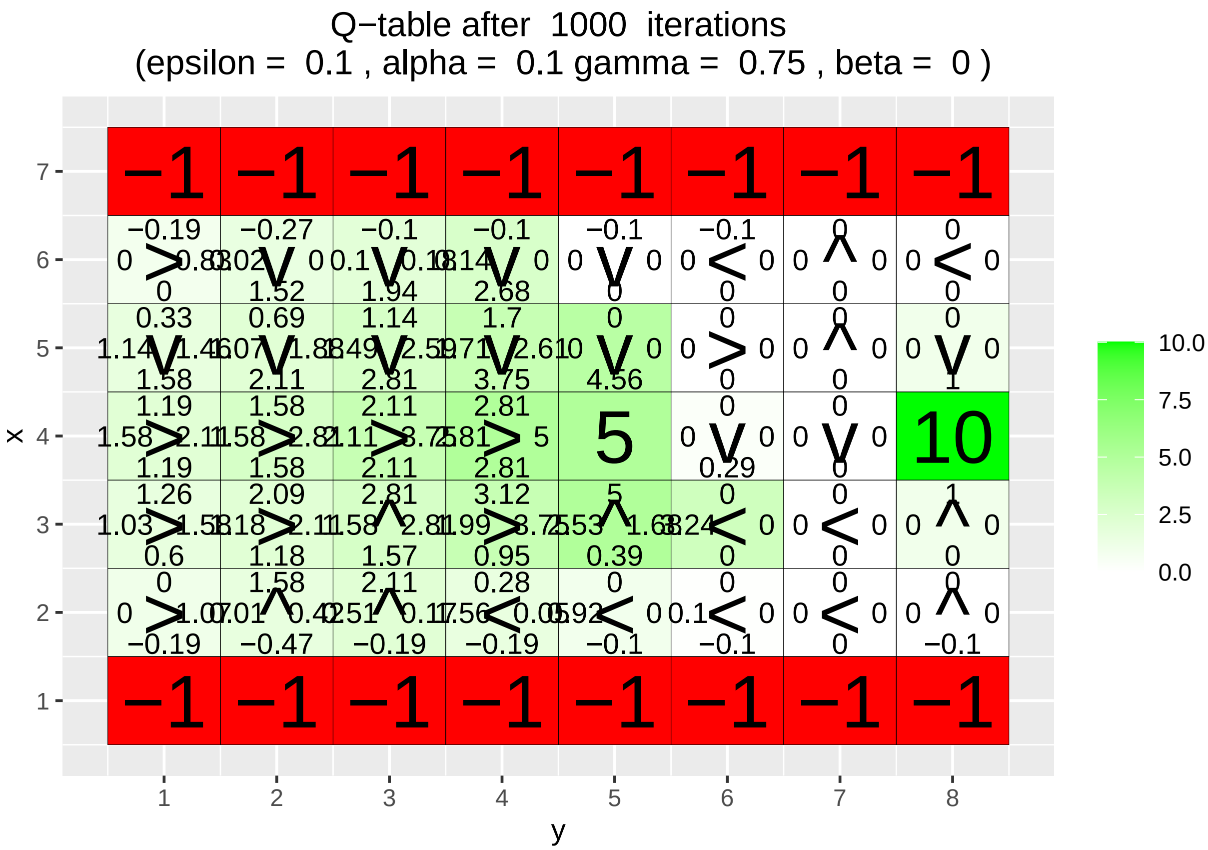


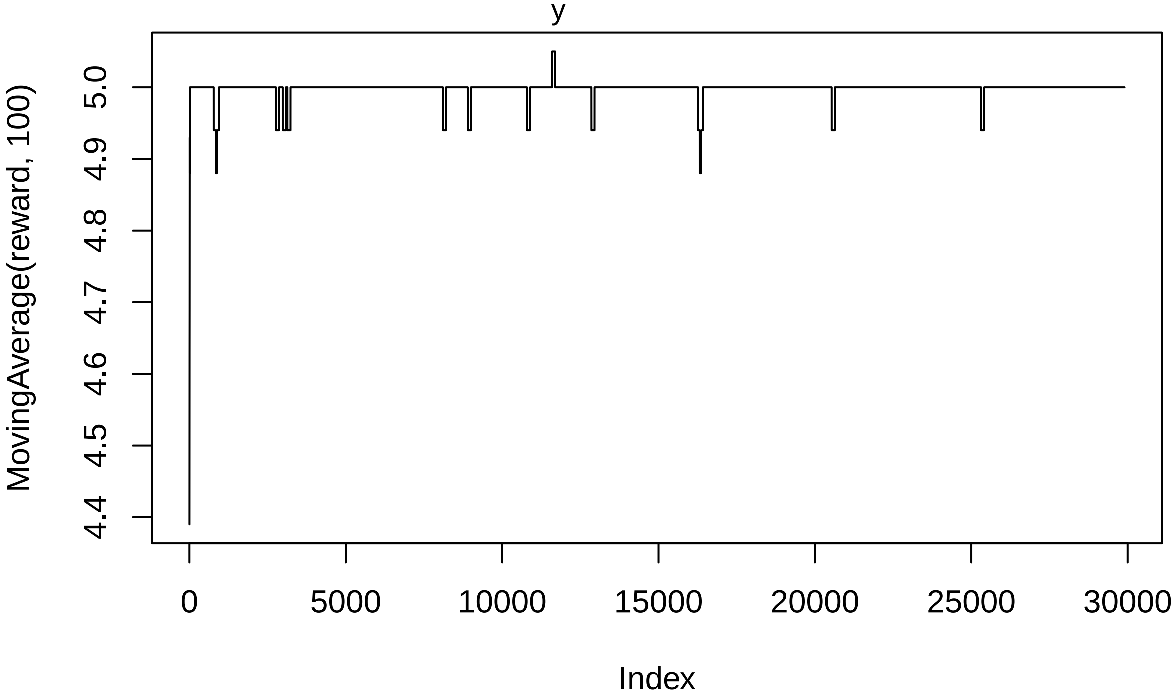


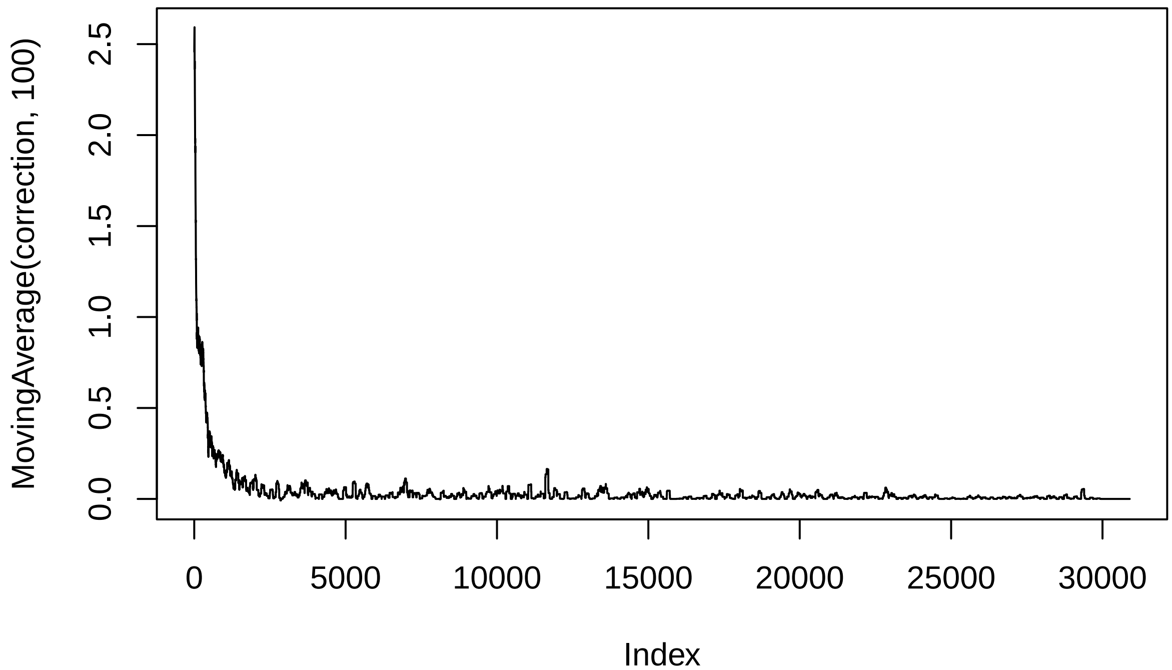
MovingAverage(reward, 100)
4.70
4.80
4.90
5.00
0
5000
10000
15000
20000
25000
30000
Index

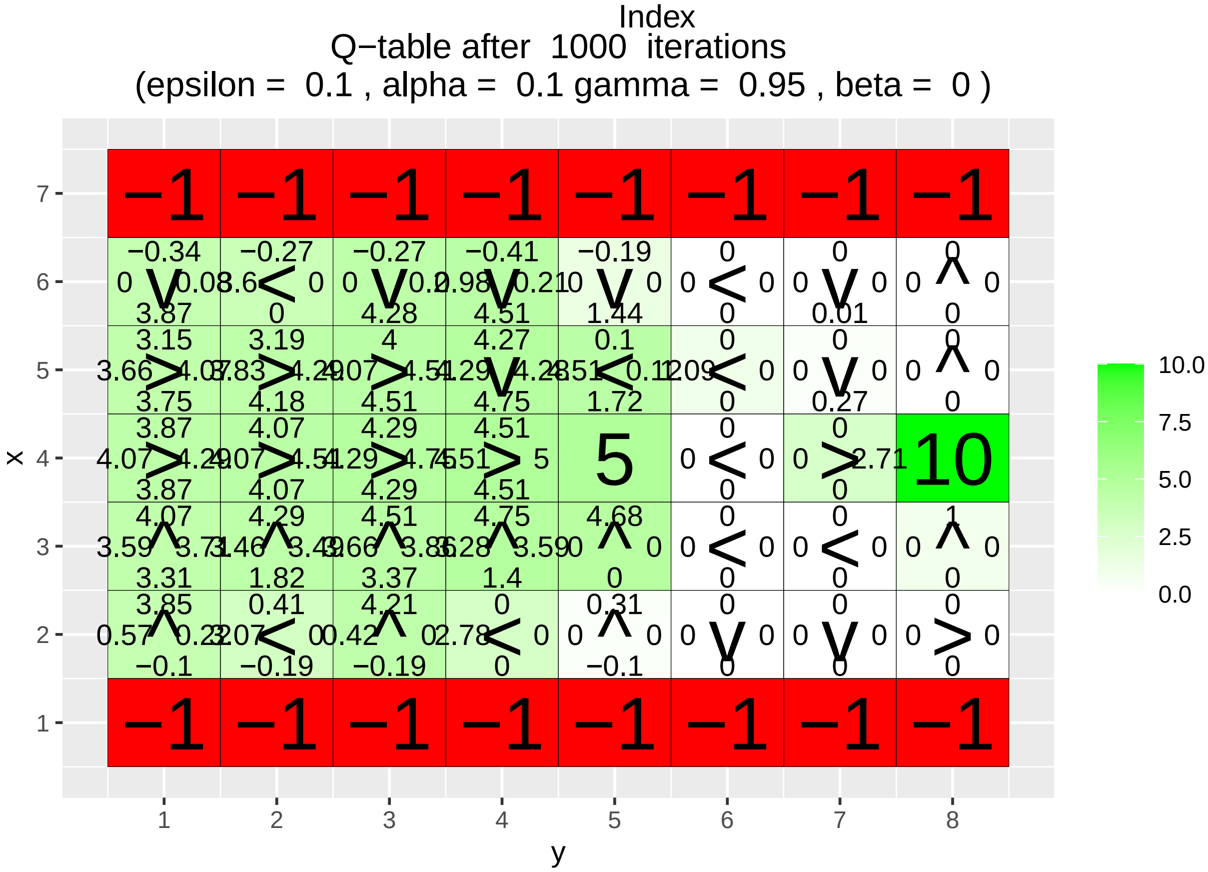


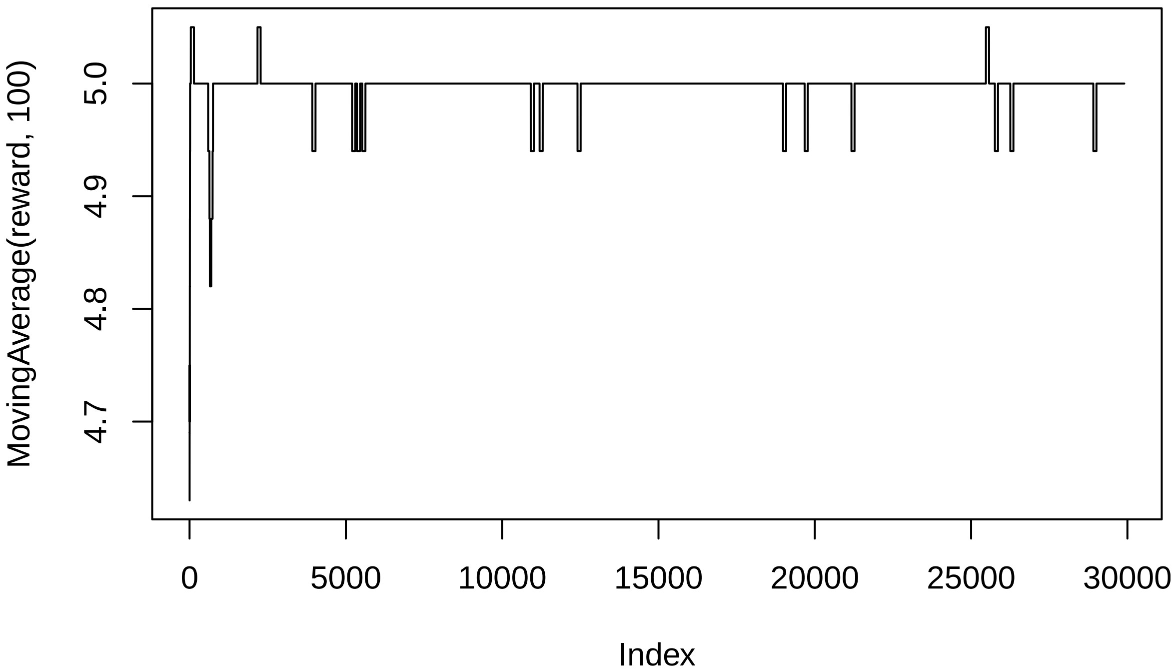
MovingAverage(correction, 100)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
5000
10000
15000
20000
25000
30000
Index

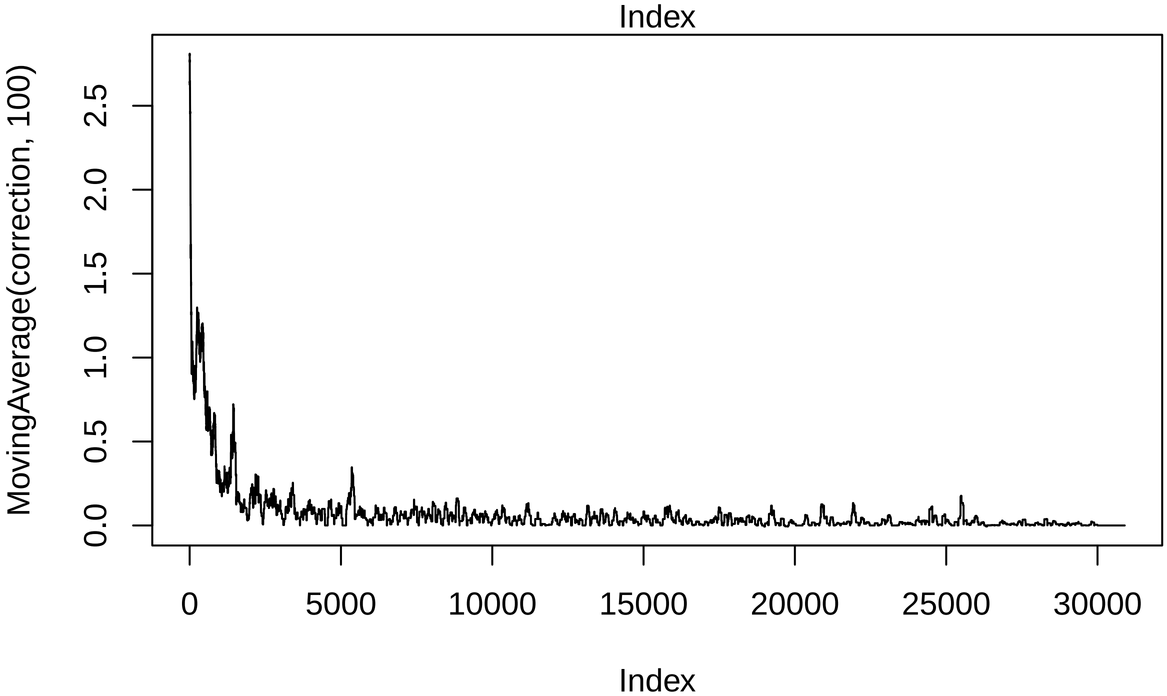

Q−table after 1000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
1
2
3
4
5
6
7
8
10.0
7.5
5.0
2.5
0.0

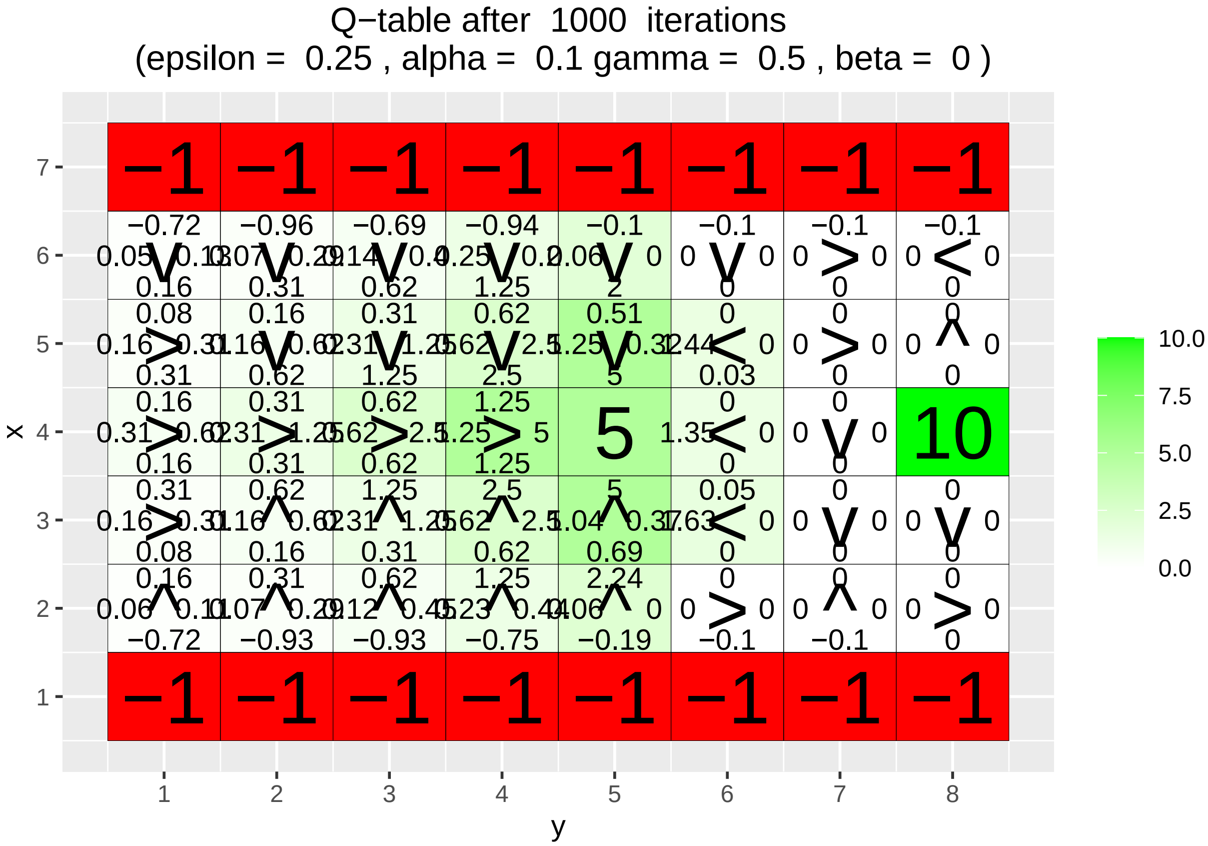

MovingAverage(reward, 100)
Index
4.4
4.5
4.6
4.7
4.8
4.9
5.0
0
5000
10000
15000
20000
25000
30000

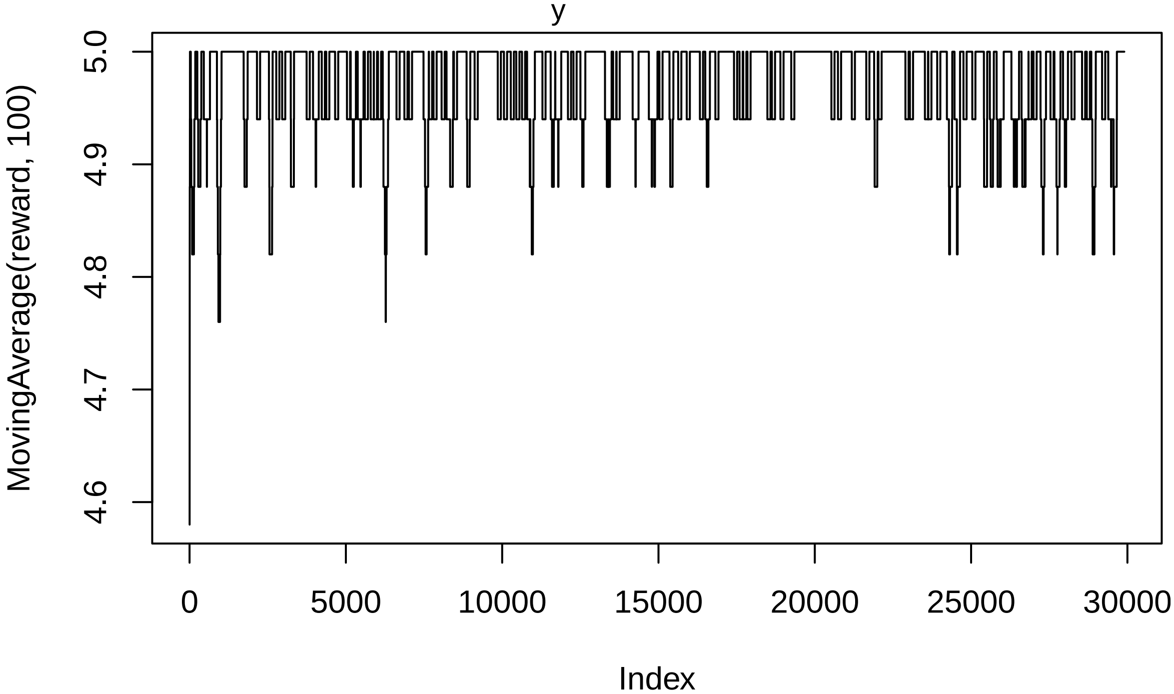

MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
2.0
2.5
0
5000
10000
15000
20000
25000
30000
Index

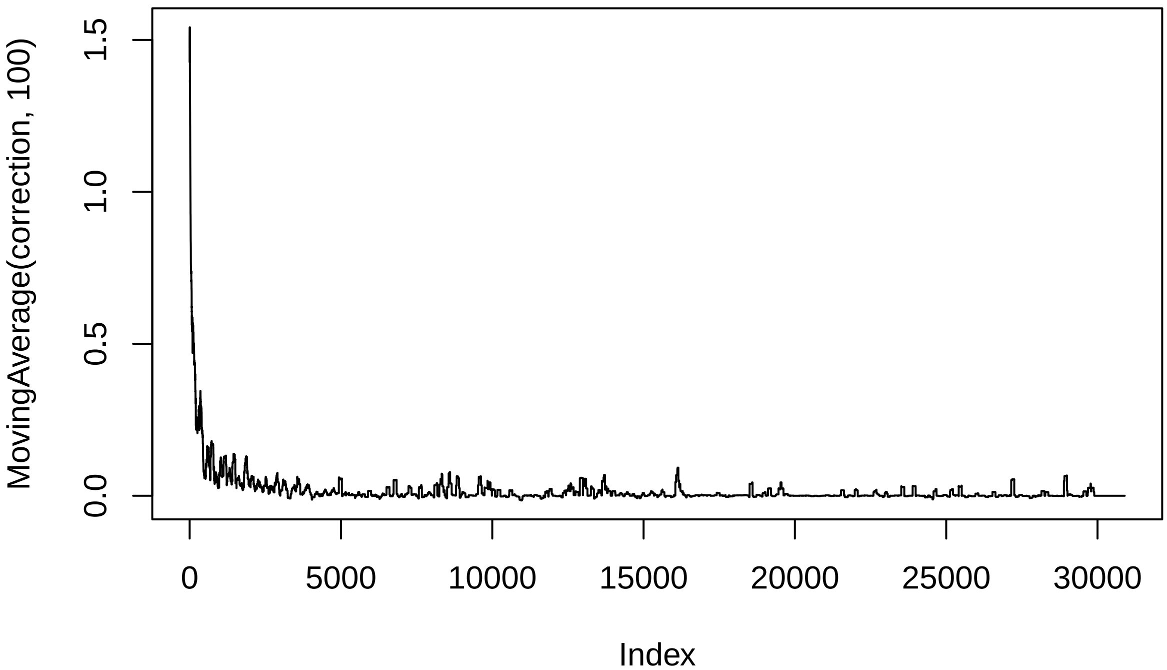

Q−table after 1000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0

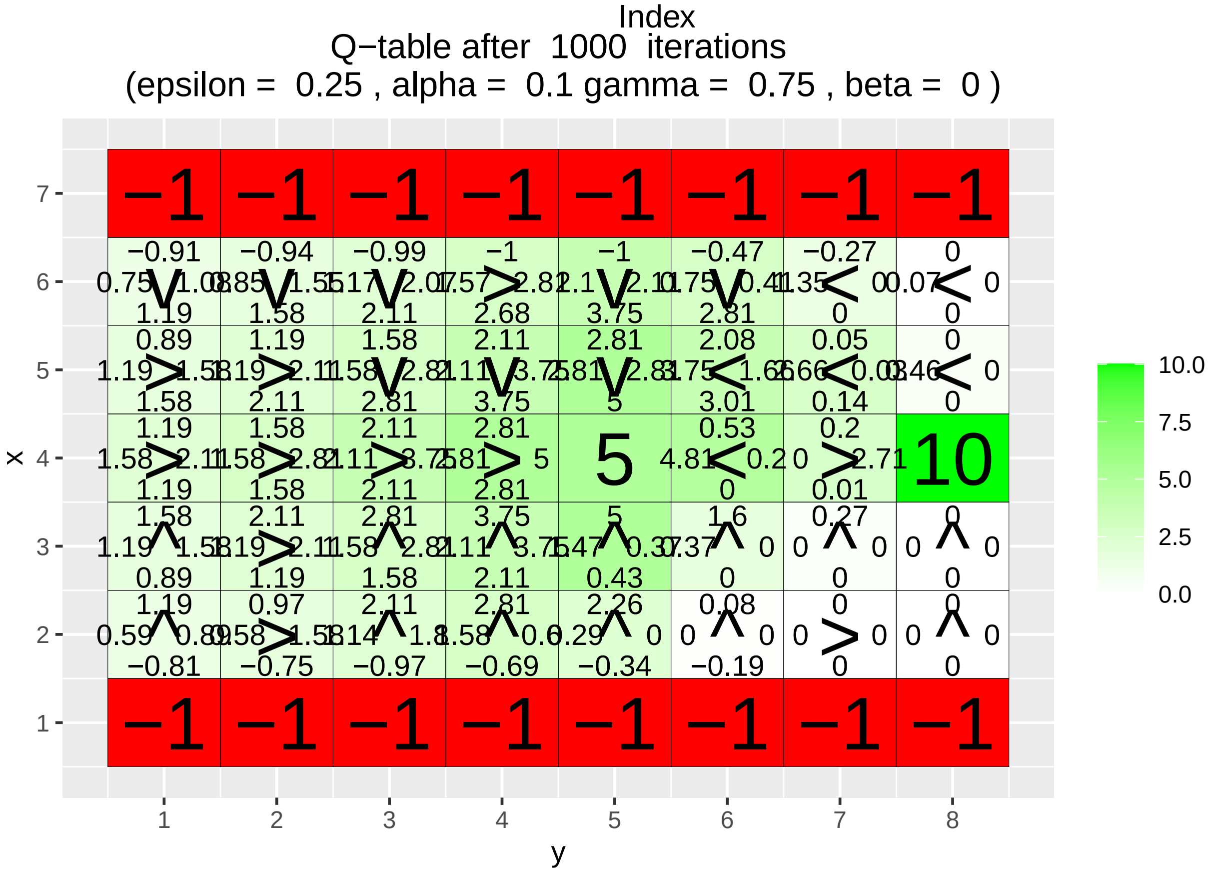

MovingAverage(reward, 100)
4.7
4.8
4.9
5.0
0
5000
10000
15000
20000
25000
30000
Index

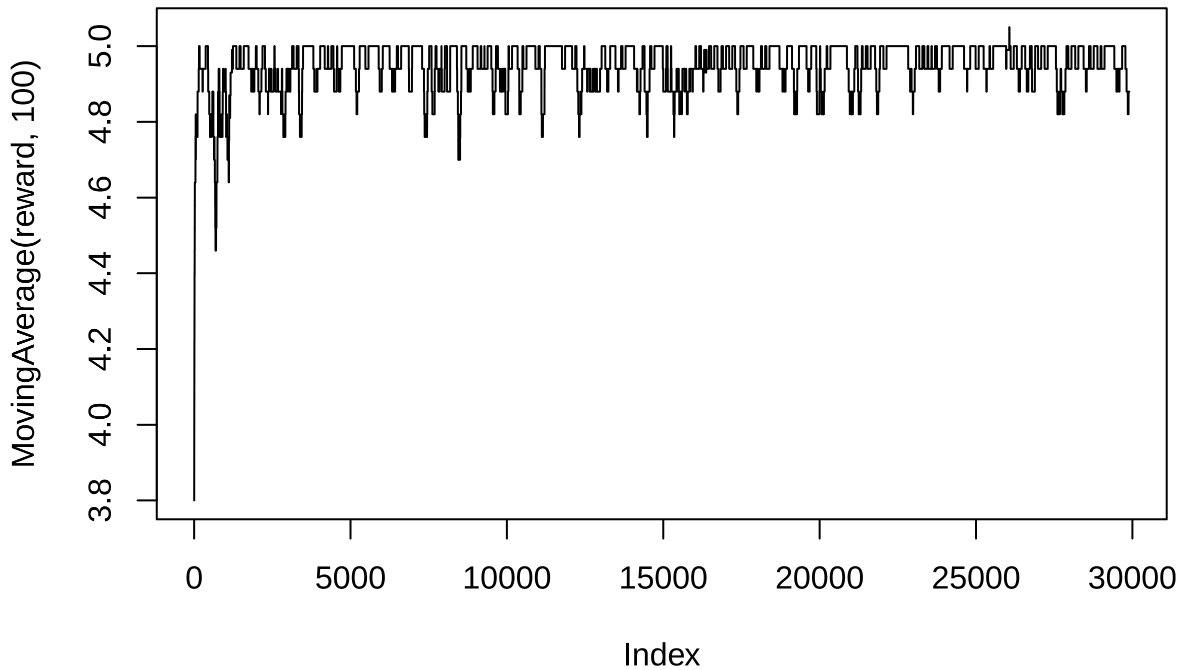

MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
2.0
2.5
0
5000
10000
15000
20000
25000
30000
Index

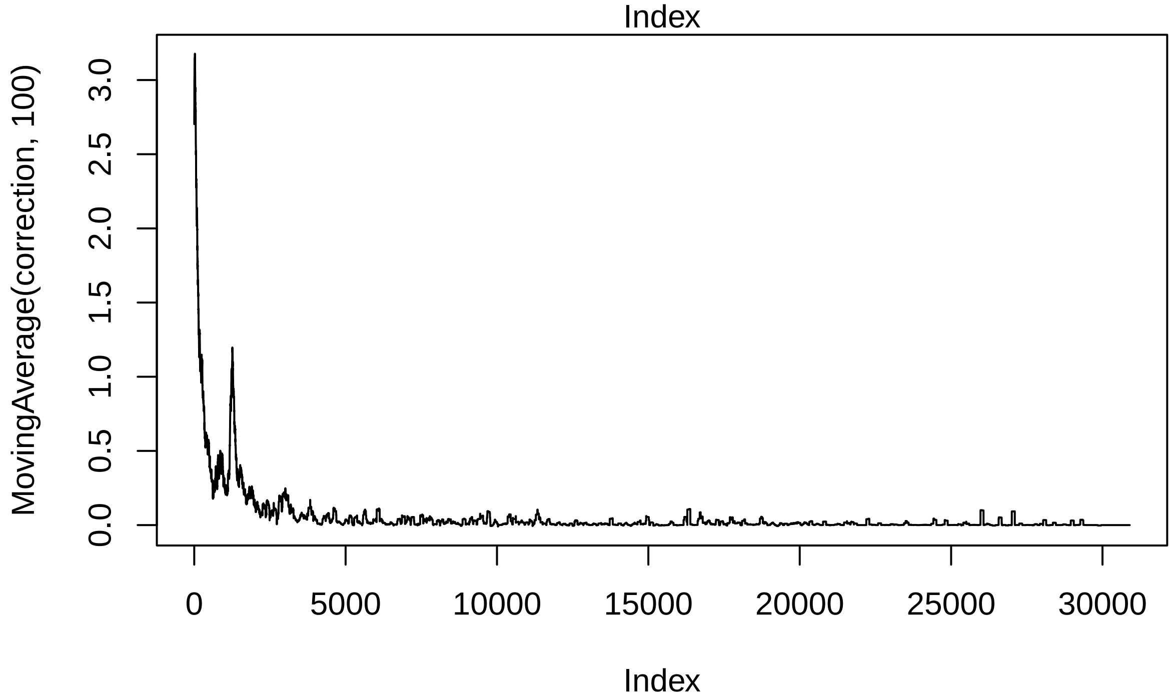

Q−table after 1000 iterations
(epsilon = 0.25 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
10
5

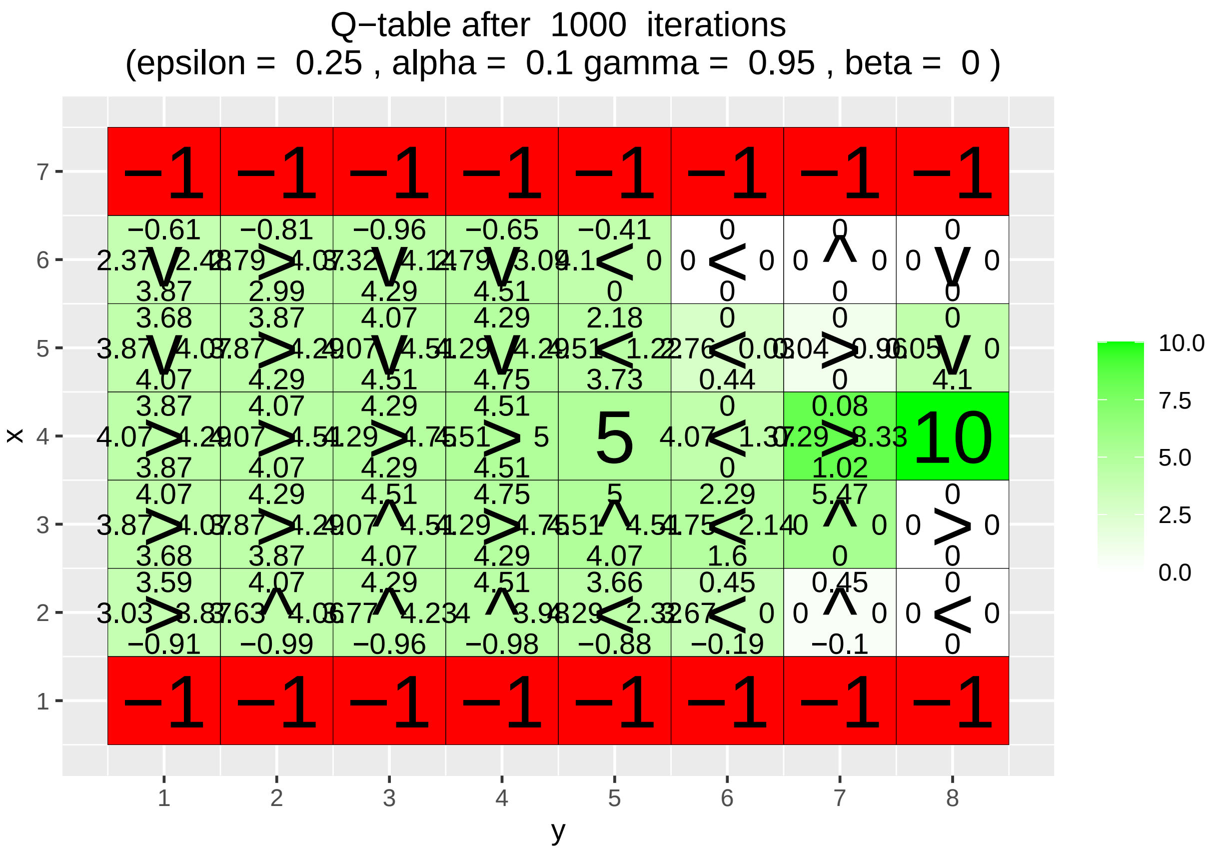

MovingAverage(reward, 100)
Index

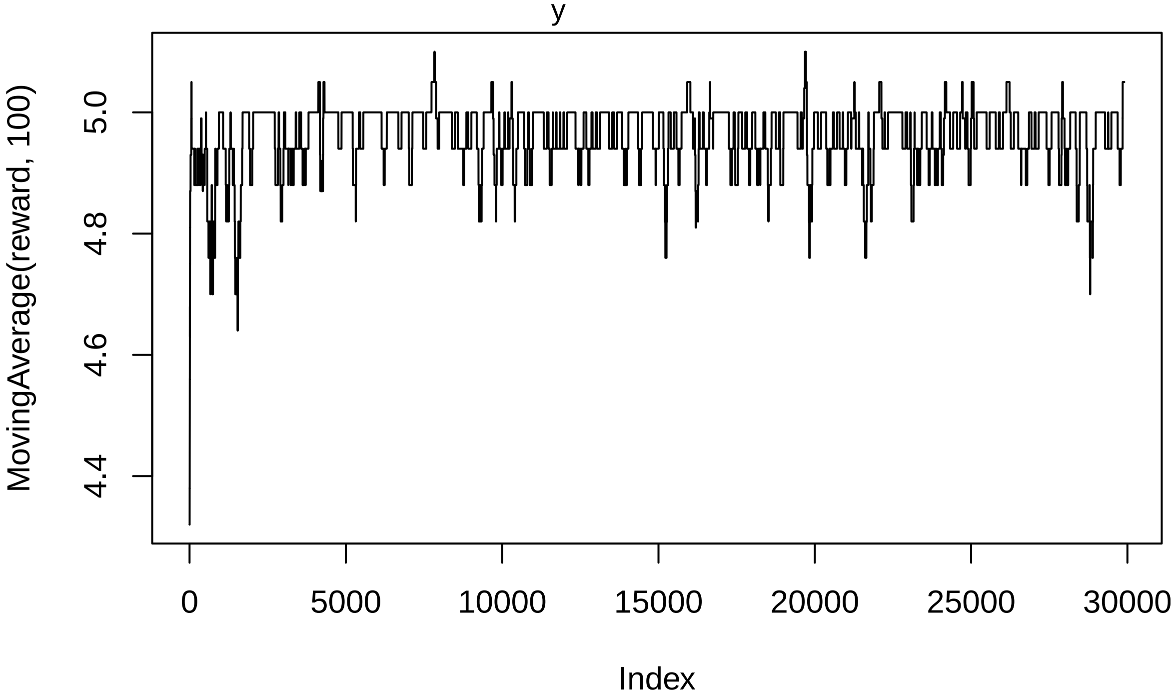

MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
0
5000
10000
15000
20000
25000
30000
Index

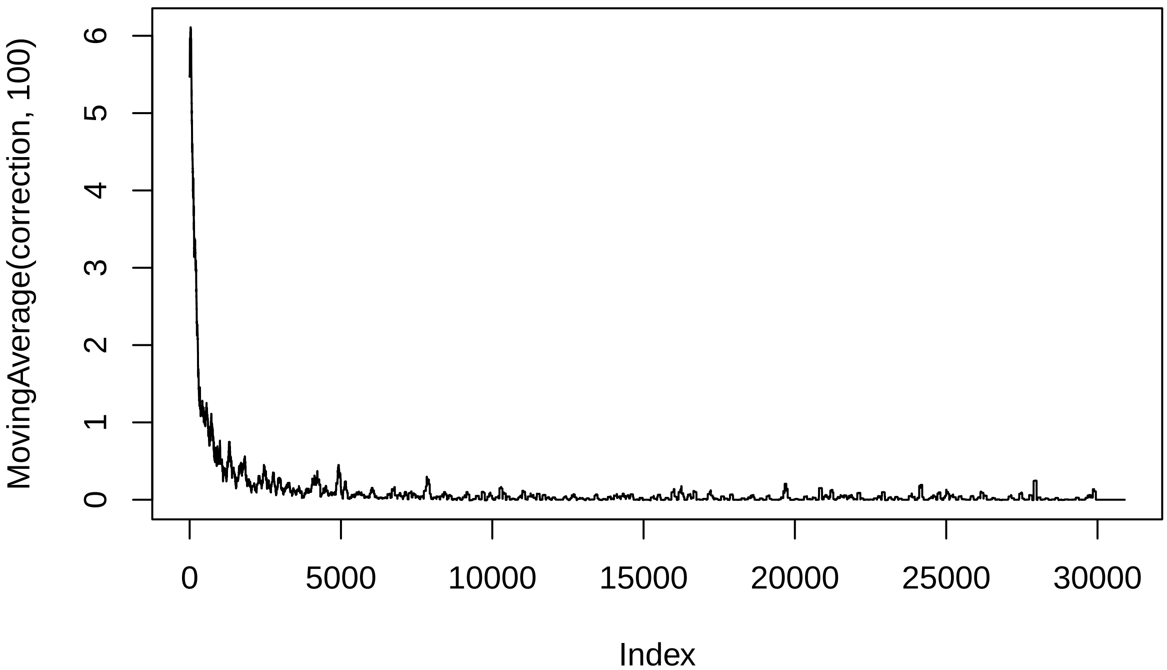

Q−table after 1000 iterations
(epsilon = 0.25 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0

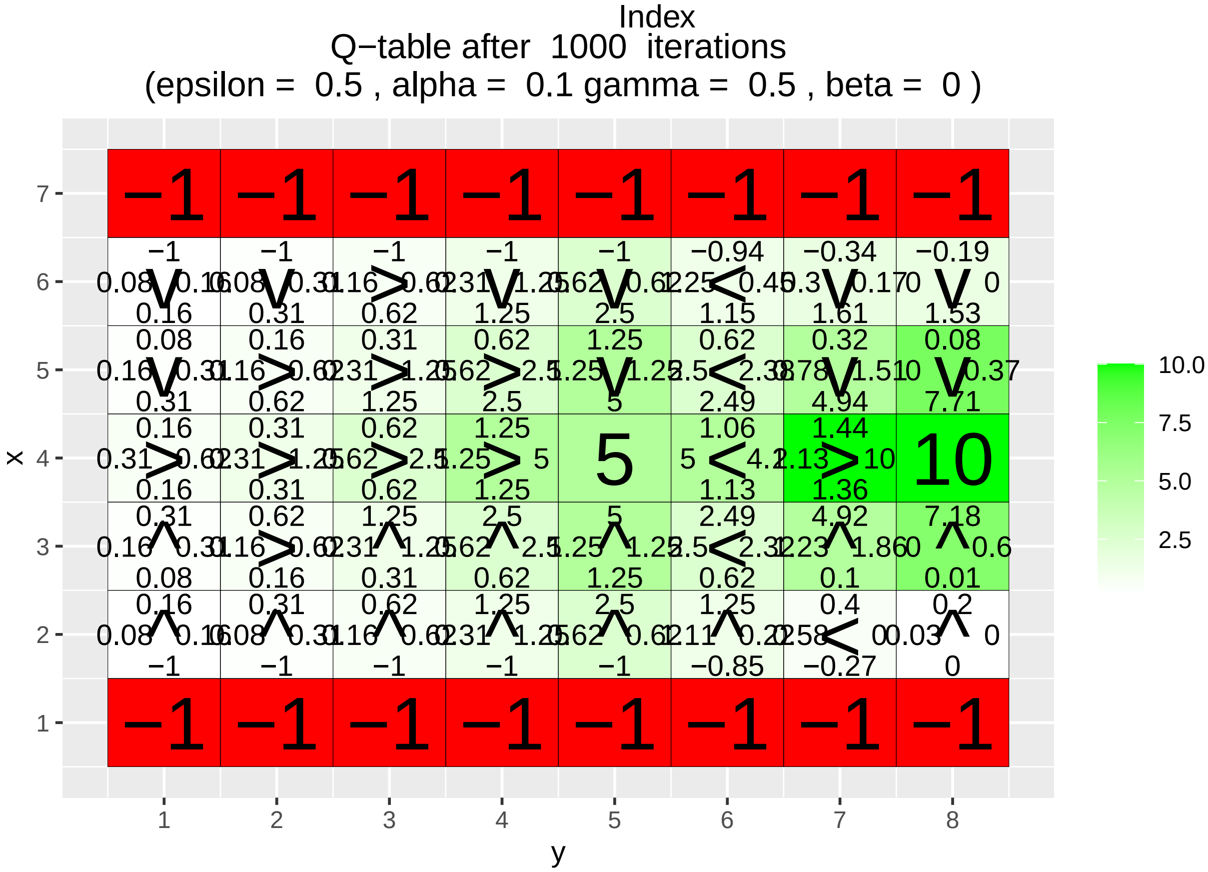

MovingAverage(reward, 100)
3.8
4.0
4.2
4.4
4.6
4.8
5.0
0
5000
10000
15000
20000
25000
30000
Index

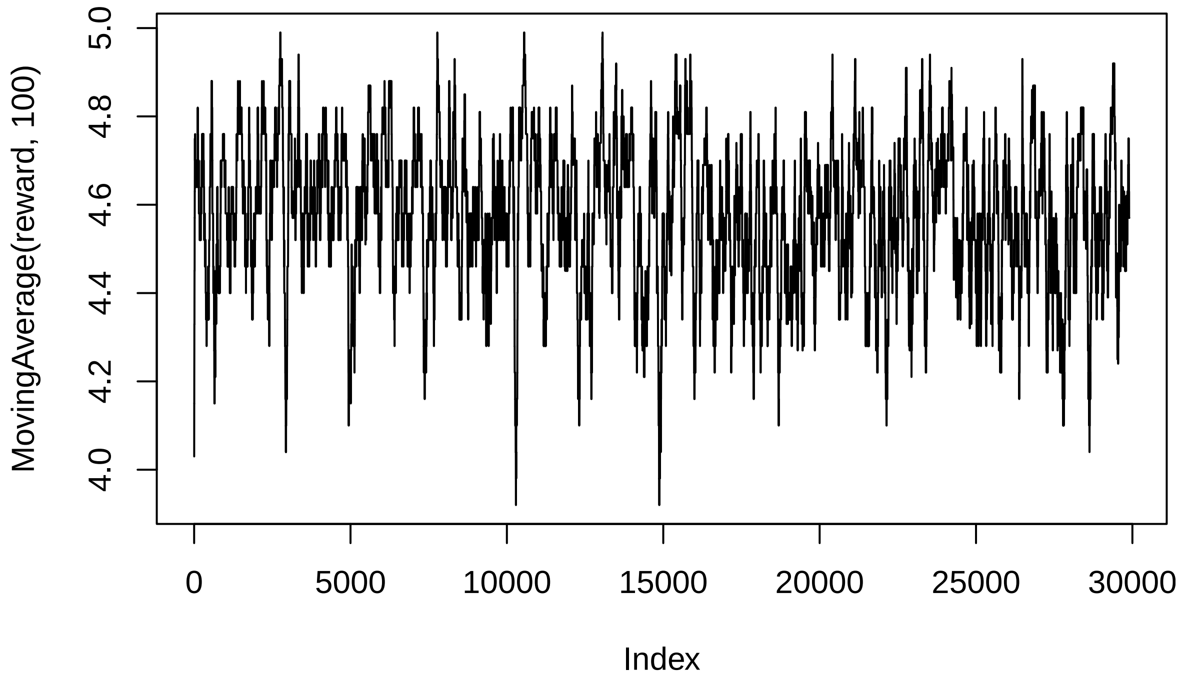

MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
5000
10000
15000
20000
25000
30000
Index


Q−table after 1000 iterations
(epsilon = 0.25 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
1
2
3
4
5
6
7
8
10.0
7.5
5.0
2.5
0.0

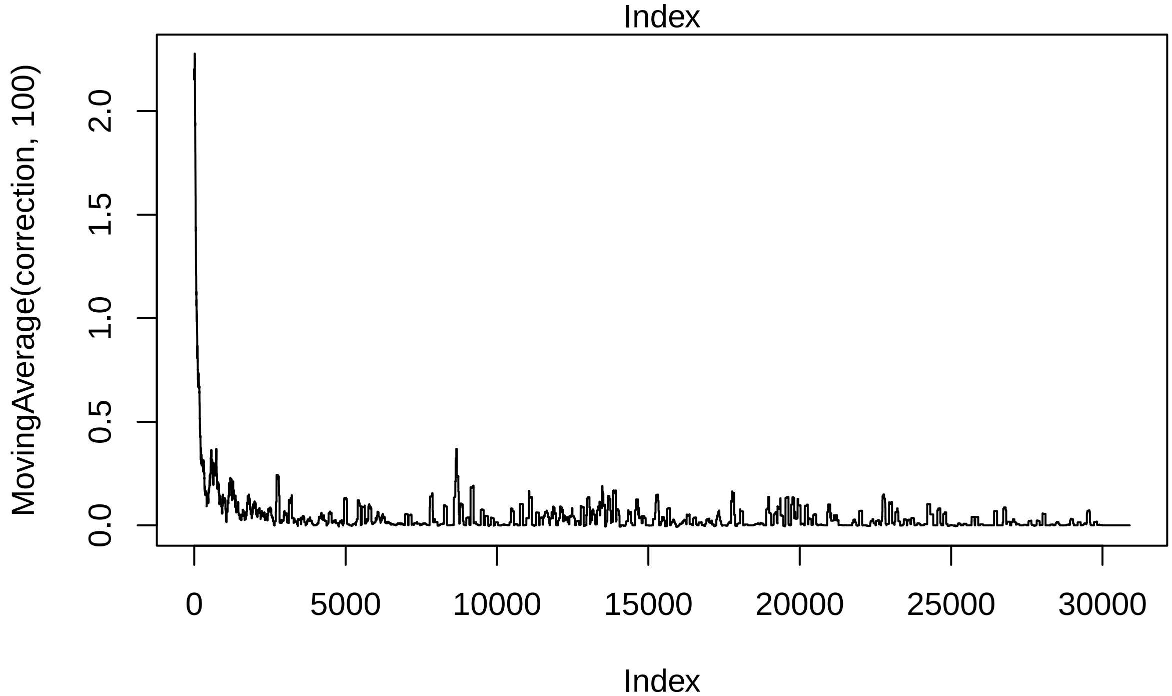

MovingAverage(reward, 100)
Index
4.4
4.6
4.8
5.0
0
5000
10000
15000
20000
25000
30000


MovingAverage(correction, 100)
Index
0
1
2
3
4
5
6
0
5000
10000
15000
20000
25000
30000

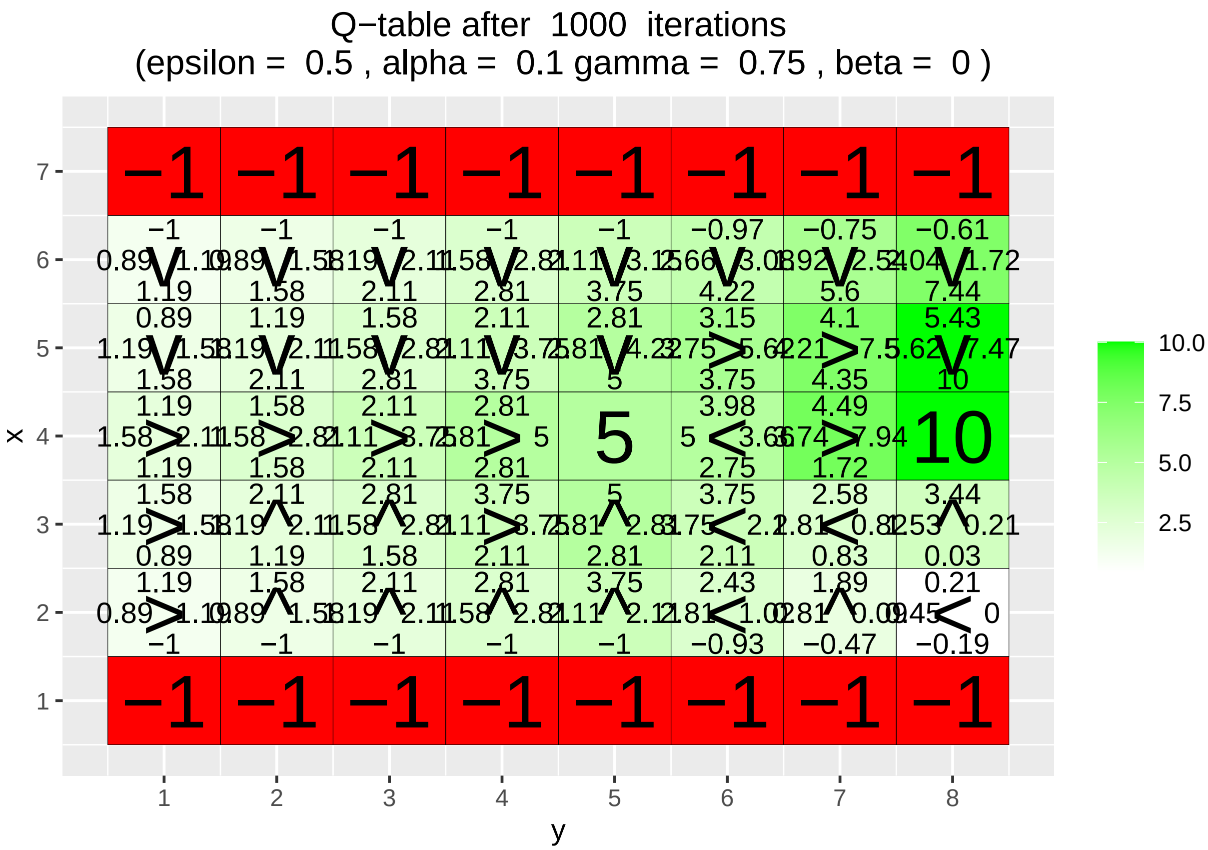

Q−table after 1000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
10.0
7.5
5.0
2.5

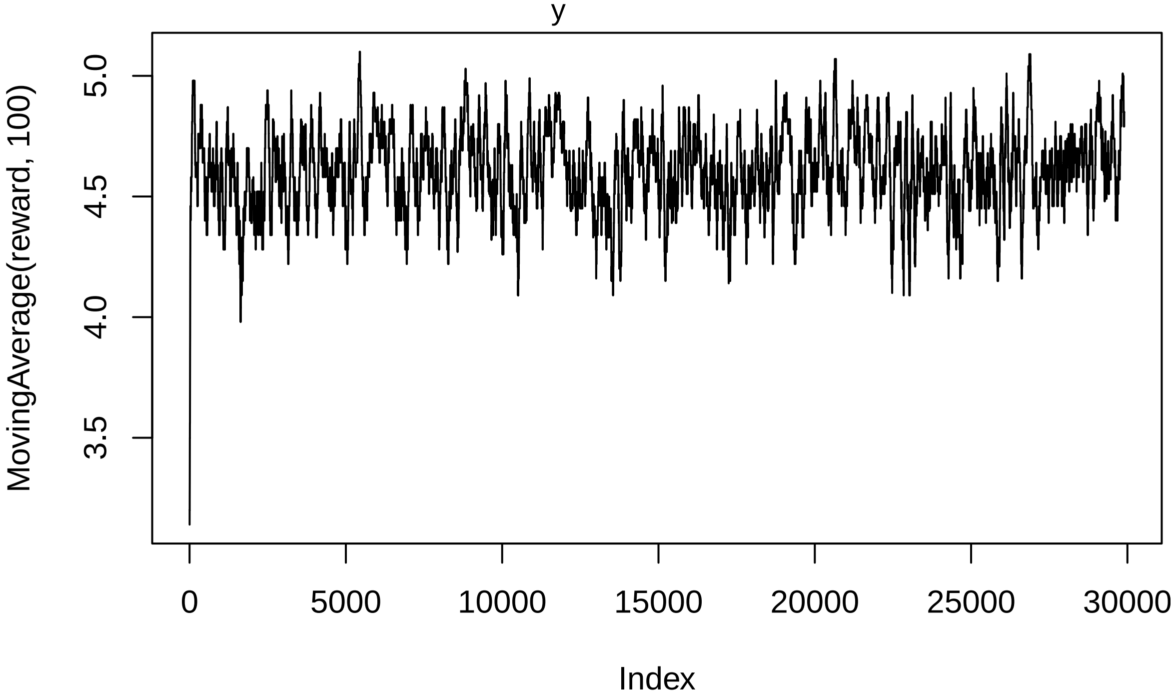


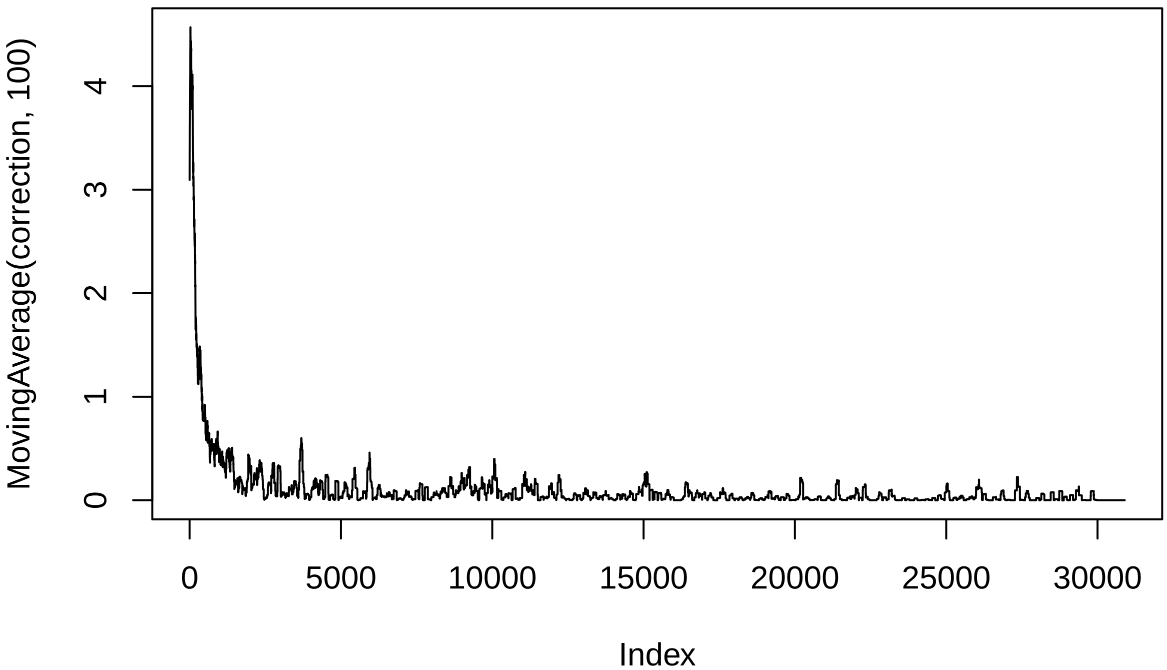
MovingAverage(reward, 100)
4.0
4.2
4.4
4.6
4.8
5.0
0
5000
10000
15000
20000
25000
30000
Index

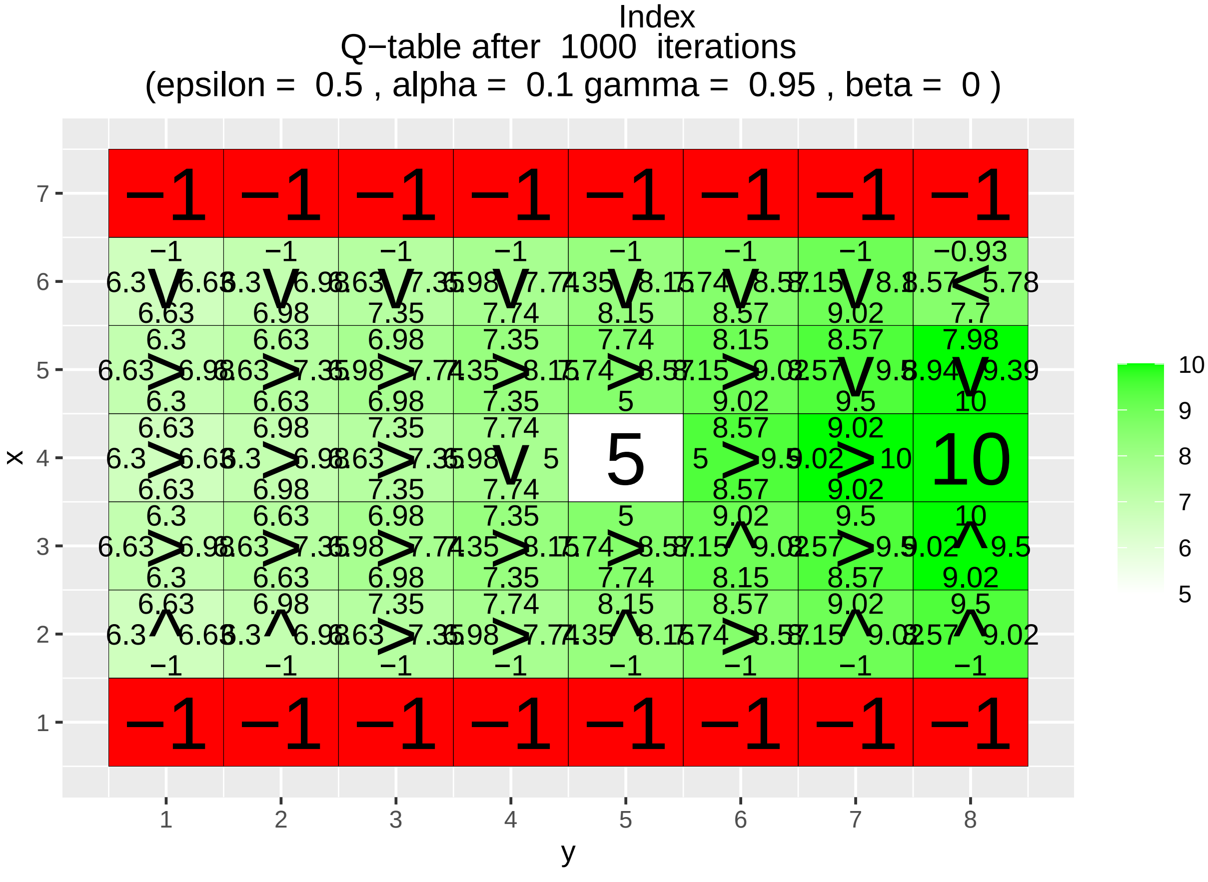


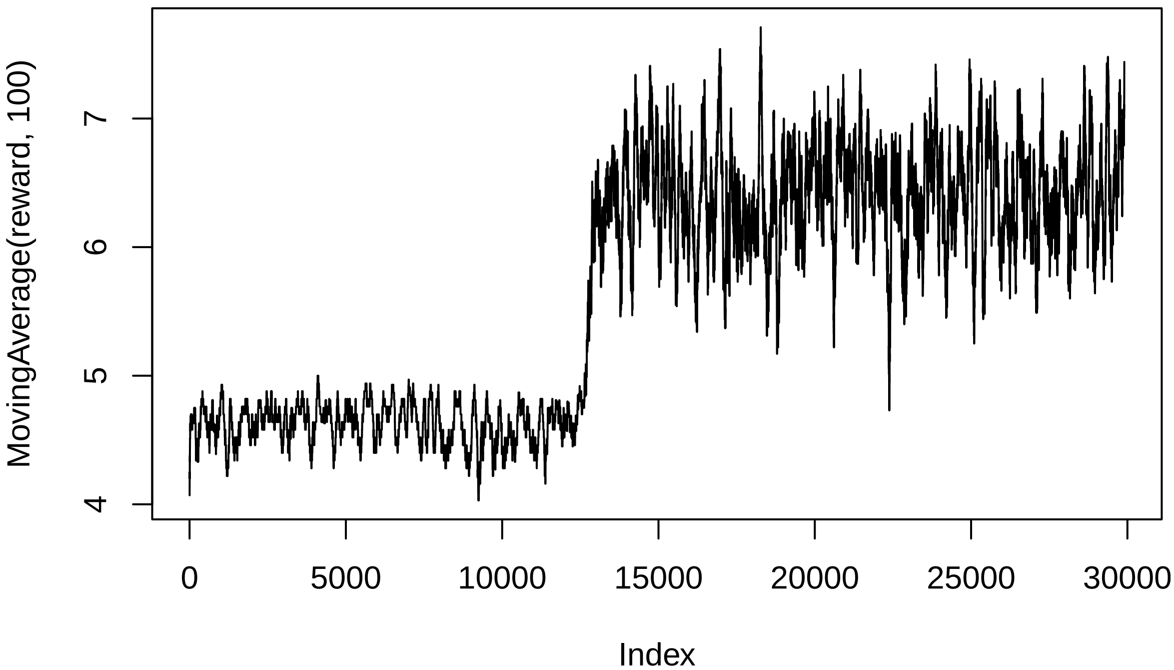
MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
2.0
0
5000
10000
15000
20000
25000
30000
Index

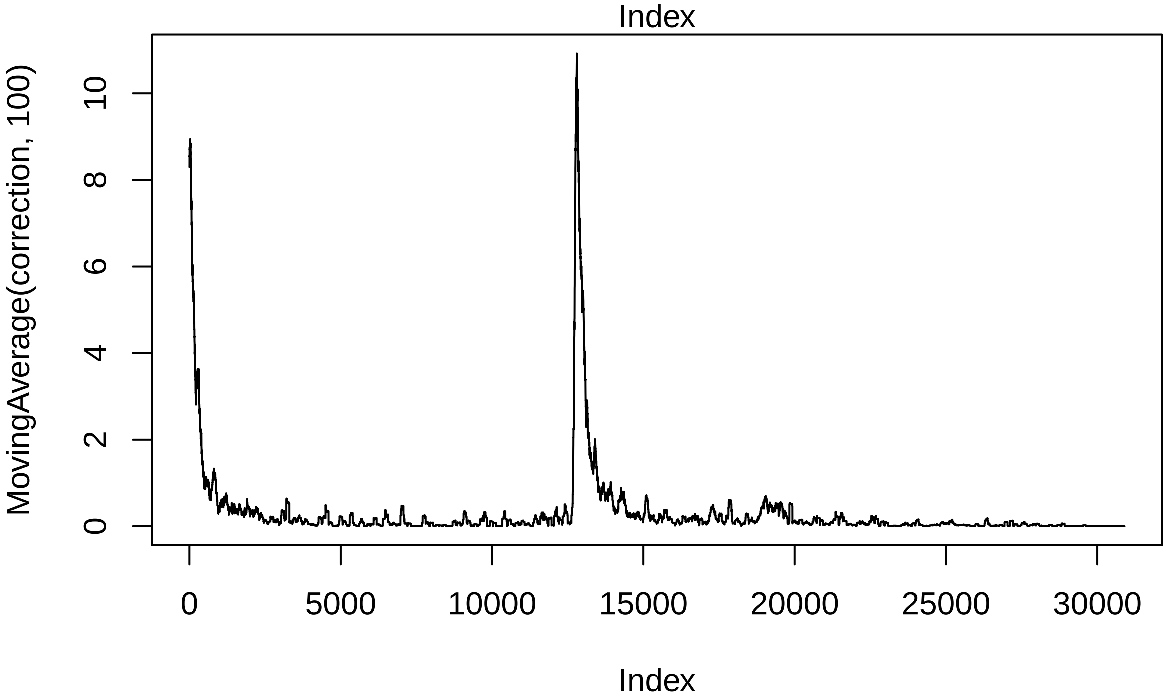

Q−table after 1000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
10.0
7.5
5.0
2.5


MovingAverage(reward, 100)
Index


MovingAverage(correction, 100)
Index
0
5000
10000
15000
20000
25000
30000


Q−table after 1000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
5
10






```
mrew
```

```
## [1] 4.997000 4.995533 4.996067 4.967401 4.942702 4.957068 4.573181 4.600580
## [9] 5.644512
```

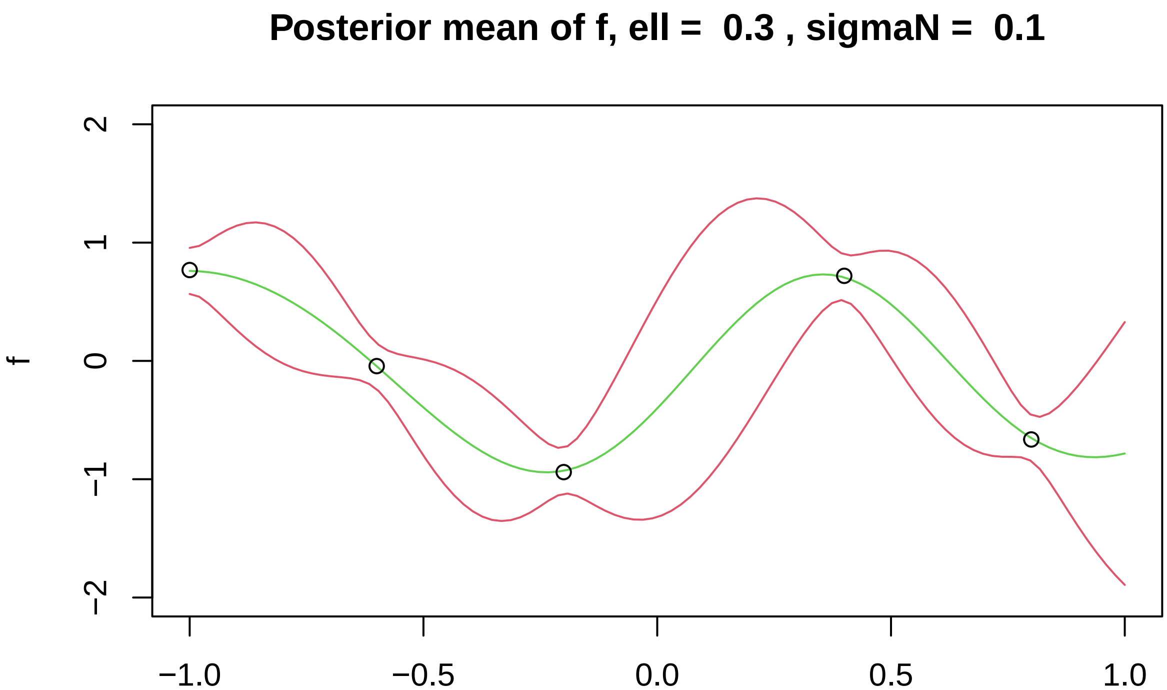
Highest average reward was having $\epsilon = 0.5$ and $\gamma = 0.95$, with 10 in the validation. This combination also recieved the highest total average reward.

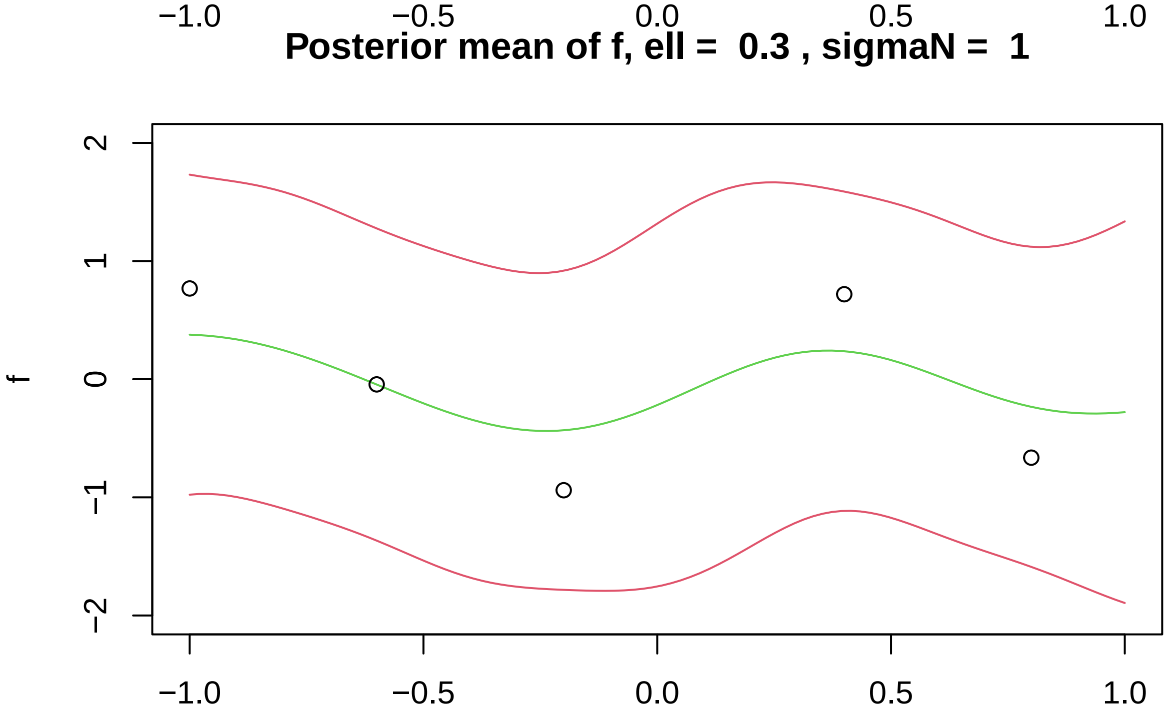
4. Gaussian Processes (5 p)

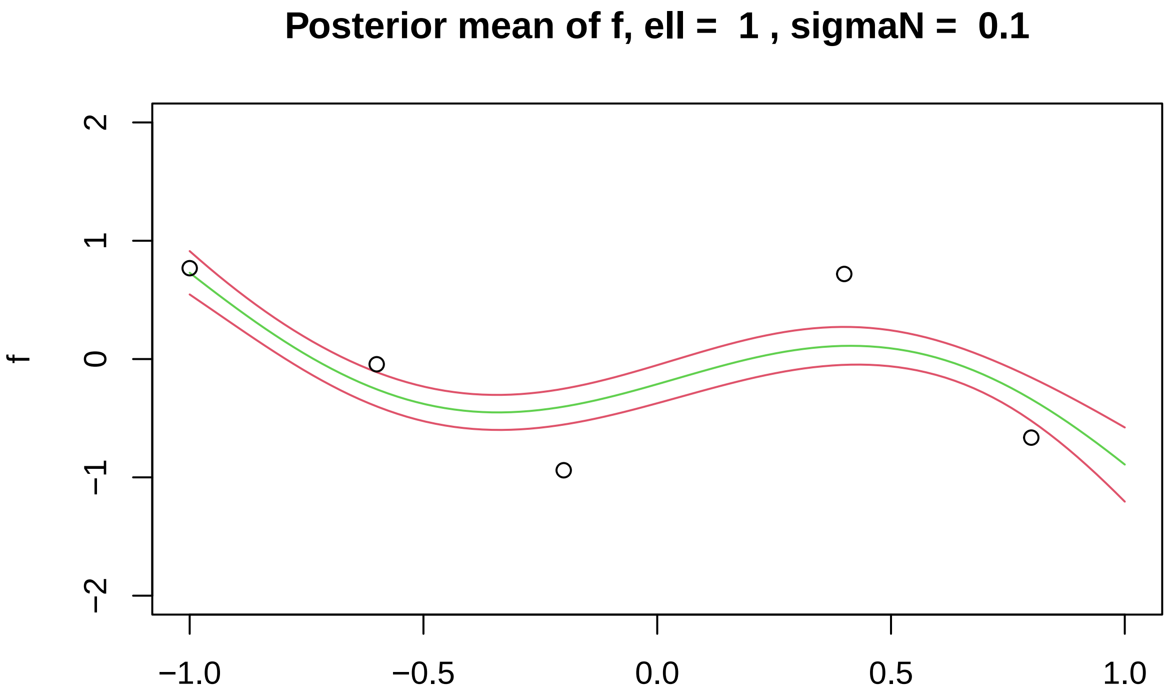
```
posteriorGP = function(X, y, sigmaNoise, XStar, k, ...) {
```

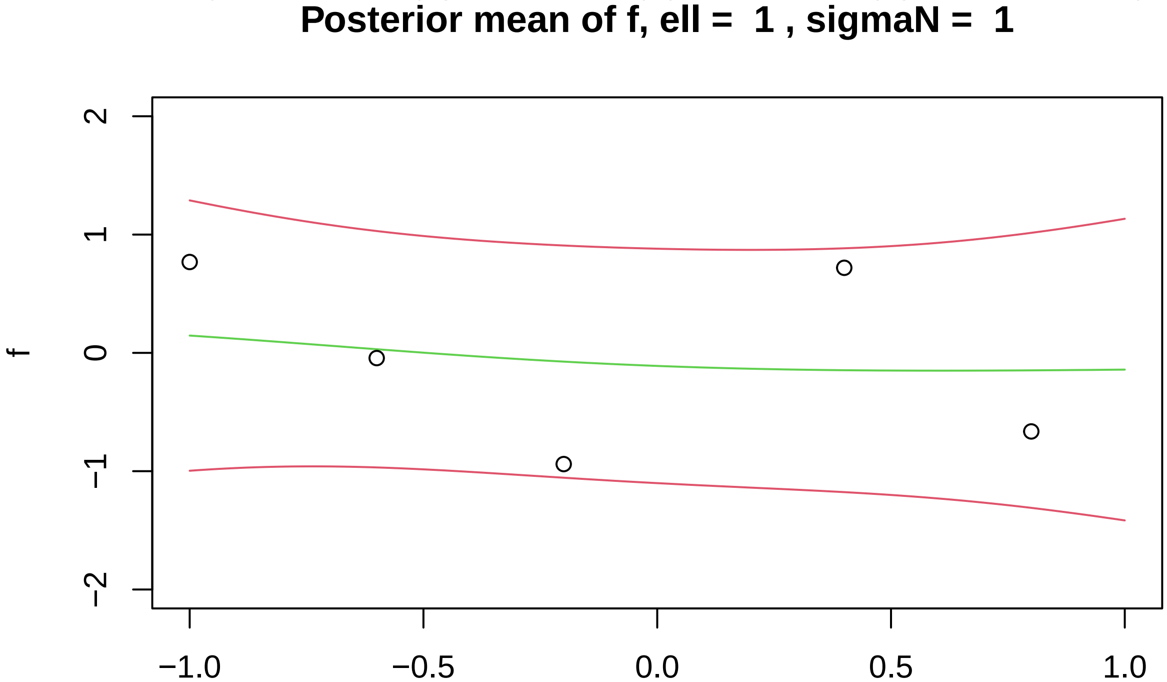
```r
  # Line 2
  n = length(X) # No of training points
  K = k(X,X,...)    # Covariance for training points
  kStar = k(X,XStar,...) # Covariance for training and test points
  # Cholesky decomposition, Lower triangular matrix
  L = t(chol(K + sigmaNoise**2 * diag(n)))
  alpha = solve(t(L), solve(L, y))

  # Line 4
  fStar = t(kStar)%*%alpha #posterior mean
  v = solve(L, kStar)

  # Line 6 :  Posterior variance
  V_fStar = k(XStar, XStar,...) - t(v)%*%v
  log_marg_likelihood = -(1/2)*t(y)%*%alpha - sum(log(diag(L))) - (n/2)*log(2*pi)

  return(list(mean = fStar, variance = V_fStar, log_likelihood = log_marg_likelihood))
}

library("mvtnorm")

# Covariance function
SquaredExpKernel <- function(x1,x2,sigmaF=1,ell=3){
  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])/ell)^2 )
  }
  return(K)
}


sigmaF = 1
ell = 0.3
sigmaN = 0.1
xGrid = seq(-1,1,length = 100)


# 2.1.4
x = c(-1, -0.6, -0.2, 0.4, 0.8)
y = c(0.768, -0.044, -0.94, 0.719, -0.664)

ell = c(0.3, 1)
sigmaN = c(0.1, 1)

for (i in 1:2) {
  for (j in 1:2) {
    posterior = posteriorGP(X=x, y=y, sigmaNoise=sigmaN[j], XStar=xGrid,
                          k = SquaredExpKernel, sigmaF, ell[i])
    plot(x = xGrid, y = posterior$mean, type = "l", col = 3, ylim = c(-2,2), ylab = "f",
         xlab = "", main = paste("Posterior mean of f, ell = ",ell[i],", sigmaN = ",sigmaN[j]))
    lines(x = xGrid, y =posterior$mean +1.96*sqrt(diag(posterior$variance)), type = "l", col = 2)
```

```
    lines(x = xGrid, y =posterior$mean -1.96*sqrt(diag(posterior$variance)), type = "l", col = 2)
    points(x,y)

  }
}
```






Posterior mean of f, ell = 1 , sigmaN = 0.1
f
−2
−1
0
1
2
−1.0
−0.5
0.0
0.5
1.0


Posterior mean of f, ell = 1 , sigmaN = 1
f
−2
−1
0
1
2
−1.0
−0.5
0.0
0.5
1.0