oct2022

Olof Swedberg

2024-10-20

# 1 Bayesian Networks

```
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
```

```
data("asia")
data = asia

dagT = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]") # True Asian Network
tenFit = bn.fit(dagT,data[1:10,], method = "bayes") # Fit on first ten
```

```
## 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.
```

```
# Fit as grain
grain_fit = as.grain(tenFit)
compiled_grain = compile(grain_fit)

# B and E distribution
for (i in 11:5000) {
  z = NULL
  for (j in c("A","S", "T", "L", "X", "D")) {
    if (data[i,j] == "no"){
      z = c(z,"no")
    } else {
      z = c(z, "yes")
    }
  }

  obs_evidence = setEvidence(compiled_grain, nodes = c("A","S", "T", "L", "X", "D"),
                             states = z) # Enter evidence
  distrB = querygrain(obs_evidence, nodes = c("B"))$B #posterior distr.
```

```r
  data[i,"B"] = sample(c("no", "yes"),size = 1, prob = distrB)

  obs_evidence = setEvidence(compiled_grain, nodes = c("A","S", "T", "L", "X", "D"),
                             states = z) # Enter evidence
  distrE = querygrain(obs_evidence, nodes = c("E"))$E #posterior distr.
  data[i,"E"] = sample(c("no", "yes"),size = 1, prob = distrE)
}


learnedFit = bn.fit(dagT,data, method = "bayes") # Learn with the learned network
trueFit = bn.fit(dagT,asia, method = "bayes") # Learn with the true network

tenFit$D
```

```
##
##   Parameters of node D (multinomial distribution)
##
## Conditional probability table:
##
## , , E = no
##
##      B
## D             no        yes
##   no  0.50000000 0.02380952
##   yes 0.50000000 0.97619048
##
## , , E = yes
##
##      B
## D             no        yes
##   no  0.10000000 0.50000000
##   yes 0.90000000 0.50000000
```

```r
learnedFit$D
```

```
##
##   Parameters of node D (multinomial distribution)
##
## Conditional probability table:
##
## , , E = no
##
##      B
## D             no        yes
##   no  0.77544239 0.08357771
##   yes 0.22455761 0.91642229
##
## , , E = yes
##
##      B
## D             no        yes
##   no  0.17435550 0.31039755
##   yes 0.82564450 0.68960245
```

```r
trueFit$D
```

```
##
##   Parameters of node D (multinomial distribution)
##
## Conditional probability table:
##
## , , E = no
##
##      B
## D             no        yes
##   no  0.90012963 0.21376147
##   yes 0.09987037 0.78623853
##
## , , E = yes
##
##      B
## D             no        yes
##   no  0.27777778 0.14630225
##   yes 0.72222222 0.85369775
```

As suspected, the distribution of D obtained from the 5000 cases with imputed values performs better than the one fitted by 10 cases.

# Hidden Markov Models

```r
set.seed(12345)
library(HMM)
states = c("1.a","1.b","2.a","2.b","2.c","3.a","3.b","4.a","5.a","5.b")
emissionSymbols = c("1","2","3","4","5")

transitionProb = matrix(c(.5,.5, 0, 0, 0, 0, 0, 0, 0, 0, #State 1
                          0,.5,.5, 0, 0, 0, 0, 0, 0, 0, #State 1
                          0, 0,.5,.5, 0, 0, 0, 0, 0, 0, #State 2
                          0, 0, 0,.5,.5, 0, 0, 0, 0, 0, #State 2
                          0, 0, 0, 0,.5,.5, 0, 0, 0, 0, #State 2
                          0, 0, 0, 0, 0,.5,.5, 0, 0, 0, #State 3
                          0, 0, 0, 0, 0, 0,.5,.5, 0, 0, #State 3
                          0, 0, 0, 0, 0, 0, 0,.5,.5, 0, #State 4
                          0, 0, 0, 0, 0, 0, 0, 0,.5,.5, #State 5
                          .5, 0, 0, 0, 0, 0, 0, 0, 0,.5),#State 5
                        nrow = 10, ncol = 10, byrow = TRUE)
p = 1/3
emissionProb = matrix(c(p, p, 0, 0, p, #State 1
                        p, p, 0, 0, p, #State 1
                        p, p, p, 0, 0, #State 2
                        p, p, p, 0, 0, #State 2
                        p, p, p, 0, 0, #State 2
                        0, p, p, p, 0, #State 3
                        0, p, p, p, 0, #State 3
                        0, 0, p, p, p, #State 4
                        p, 0, 0, p, p, #State 5
                        p, 0, 0, p, p),#State 5
```

```
                        nrow = 10, ncol = 5, byrow = TRUE)
startProb = c(0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1)
hmm = initHMM(States = states, Symbols = emissionSymbols, startProbs = startProb, transProbs = transiti
              emissionProbs = emissionProb)

nIter = 100
simulation = simHMM(hmm, nIter)
simulation
```

```
## $states
##   [1] "5.a" "5.a" "5.a" "5.a" "5.b" "1.a" "1.b" "1.b" "1.b" "1.b" "2.a" "2.a"
##  [13] "2.b" "2.b" "2.b" "2.b" "2.b" "2.b" "2.b" "2.c" "3.a" "3.a" "3.b" "4.a"
##  [25] "5.a" "5.b" "5.b" "5.b" "1.a" "1.b" "1.b" "2.a" "2.a" "2.b" "2.b" "2.b"
##  [37] "2.c" "2.c" "2.c" "3.a" "3.b" "3.b" "4.a" "5.a" "5.b" "1.a" "1.b" "2.a"
##  [49] "2.a" "2.b" "2.c" "2.c" "3.a" "3.a" "3.b" "3.b" "4.a" "4.a" "4.a" "4.a"
##  [61] "5.a" "5.b" "5.b" "5.b" "5.b" "1.a" "1.a" "1.b" "1.b" "1.b" "1.b" "1.b"
##  [73] "2.a" "2.a" "2.a" "2.b" "2.c" "2.c" "2.c" "3.a" "3.b" "4.a" "4.a" "4.a"
##  [85] "4.a" "4.a" "5.a" "5.a" "5.a" "5.b" "5.b" "5.b" "1.a" "1.a" "1.a" "1.a"
##  [97] "1.a" "1.a" "1.b" "2.a"
##
## $observation
##   [1] "1" "5" "4" "5" "5" "2" "5" "2" "1" "1" "3" "3" "2" "3" "2" "2" "2" "3"
##  [19] "1" "1" "3" "3" "2" "4" "4" "4" "4" "1" "1" "2" "1" "2" "2" "3" "2" "2"
##  [37] "2" "1" "1" "4" "3" "4" "5" "5" "1" "2" "2" "3" "2" "3" "2" "3" "4" "3"
##  [55] "2" "4" "4" "3" "4" "3" "4" "5" "4" "4" "5" "5" "2" "1" "2" "5" "2" "1"
##  [73] "2" "1" "3" "2" "2" "2" "2" "4" "2" "3" "5" "5" "4" "4" "5" "4" "1" "1"
##  [91] "4" "4" "1" "1" "2" "1" "1" "2" "1" "3"
```

```
# results = matrix(0, nrow = length(simulation$states), ncol = 2)
#
# for (i in 1:length(simulation$states)) {
#   results[i,1] = simulation$states[i]
#   results[i,2] = simulation$observation[i]
# }
# results
```

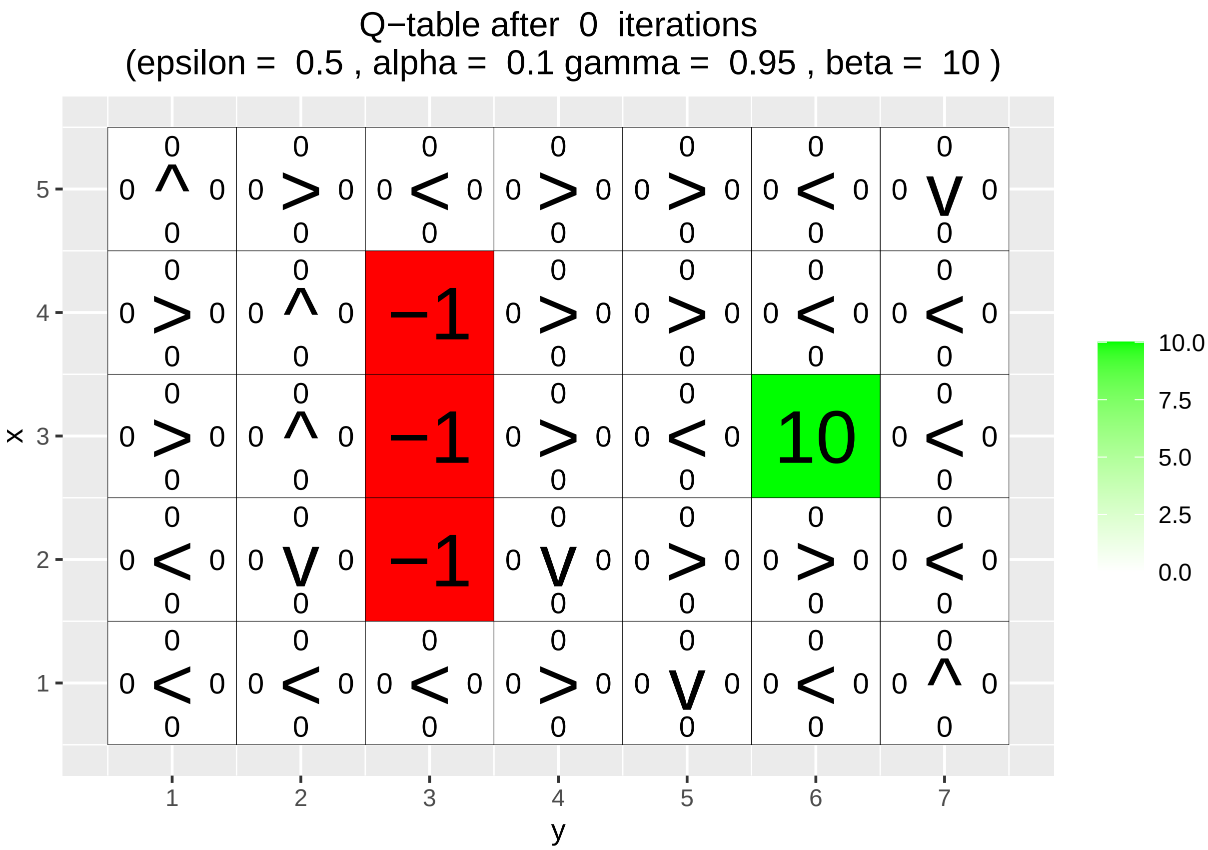
## Reinforcement Learning

```
#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 = 10) {
  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)))
}
```

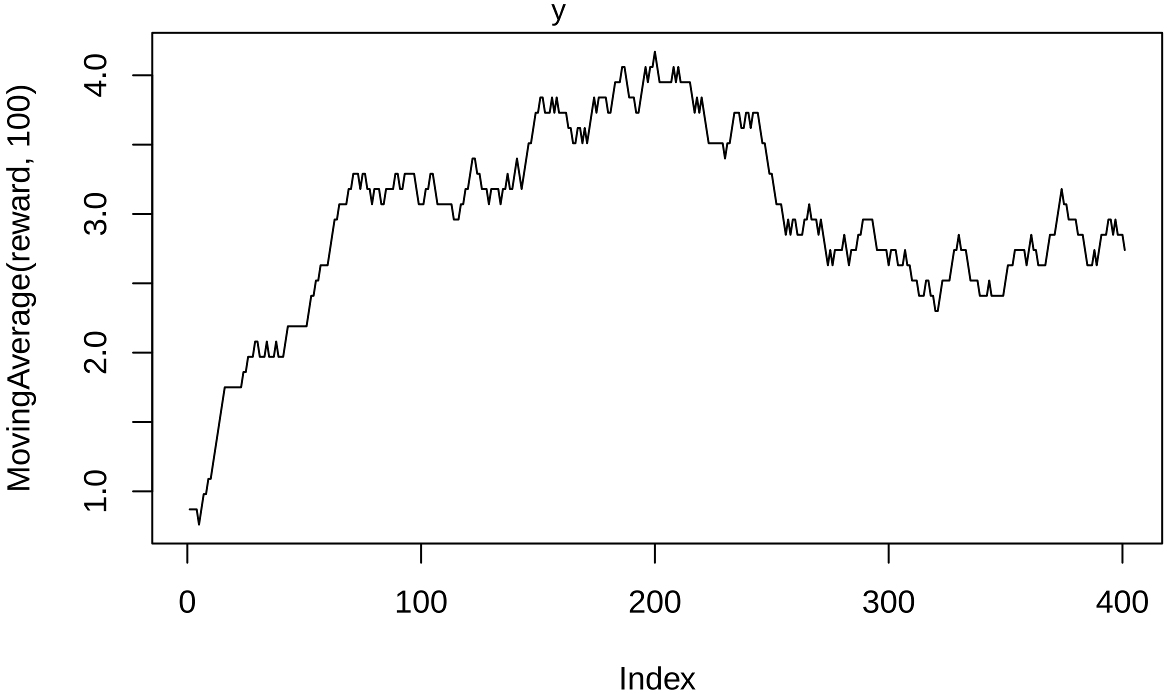
```
GreedyPolicy <- function(x, y){
  q_values = q_table[x, y, ]
  max_actions = which(q_values == max(q_values))
  if (length(max_actions) == 1) {
    return(max_actions)
  } else {
    return(sample(max_actions, 1)) }
}

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 = 1, gamma = 0.95, beta = 0){
```

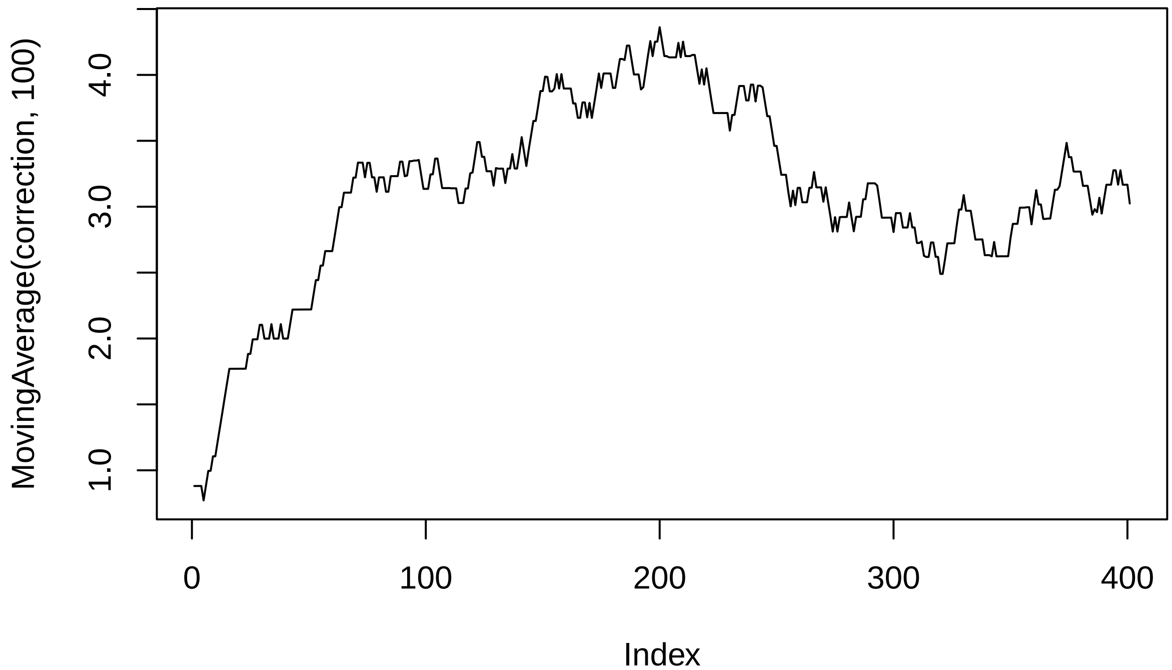
```r
  Q = start_state
  x = Q[1]
  y = Q[2]
  episode_correction = 0
  repeat{
    action = EpsilonGreedyPolicy(x,y,epsilon) # follow policy
    next_state = transition_model(x,y,action,beta) # excecute action
    reward = reward_map[next_state[1],next_state[2]] # get reward
    # Q-table update.
    correction = reward + gamma * max(q_table[next_state[1],next_state[2],])-q_table[x,y,action]
    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 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

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

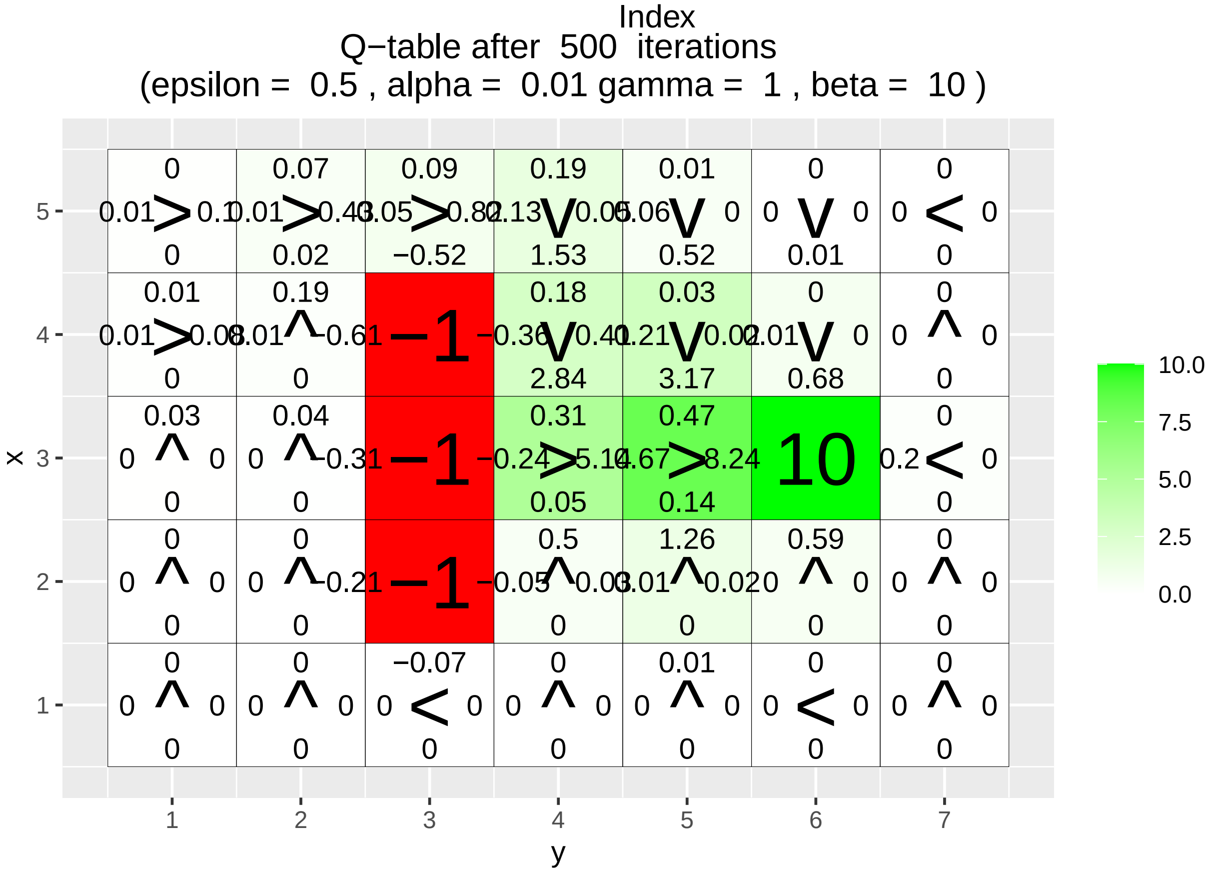


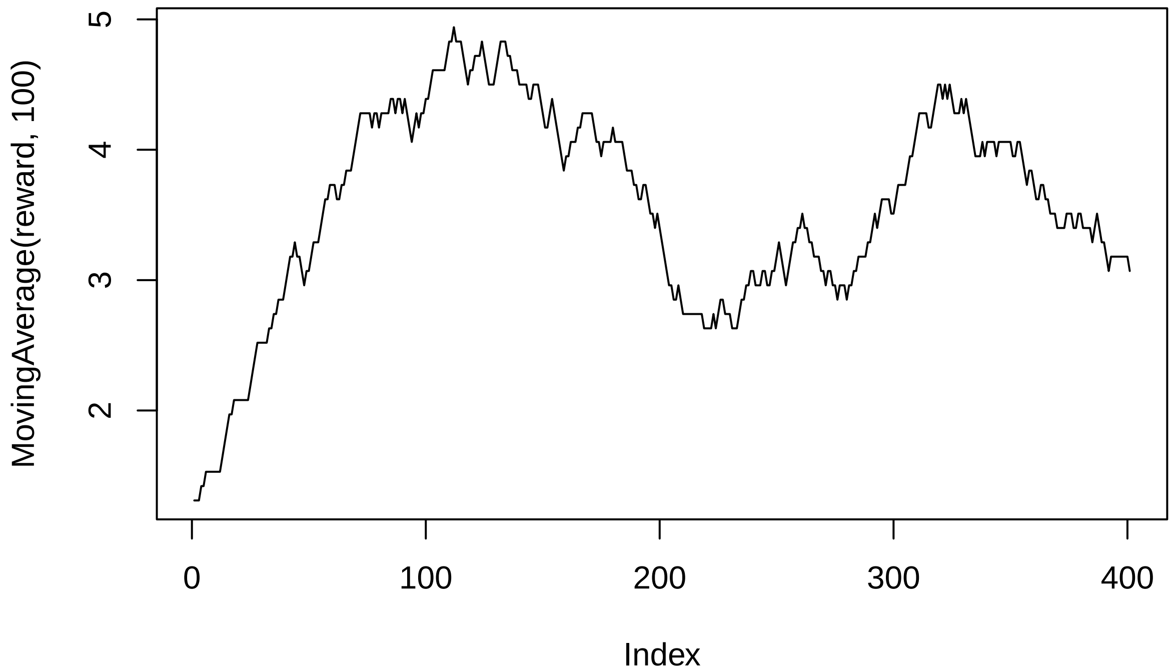

```r
MovingAverage <- function(x, n){
  cx <- c(0,cumsum(x))
  rsum <- (cx[(n+1):length(cx)] - cx[1:(length(cx) - n)]) / n
  return (rsum)
}

for (i in c(0.001, 0.01, 0.1)) {
  q_table <- array(0,dim = c(H,W,4))
  reward <- NULL
  correction <- NULL
  for (j in 1:500) {
    foo <- q_learning(alpha = i, gamma = 1, start_state = c(3,1))
    reward <- c(reward,foo[1])
    correction <- c(correction,foo[2])
  }
  vis_environment(500, alpha = i, gamma = 1)
  plot(MovingAverage(reward,100),type = "l")
  plot(MovingAverage(correction,100),type = "l")
}
```

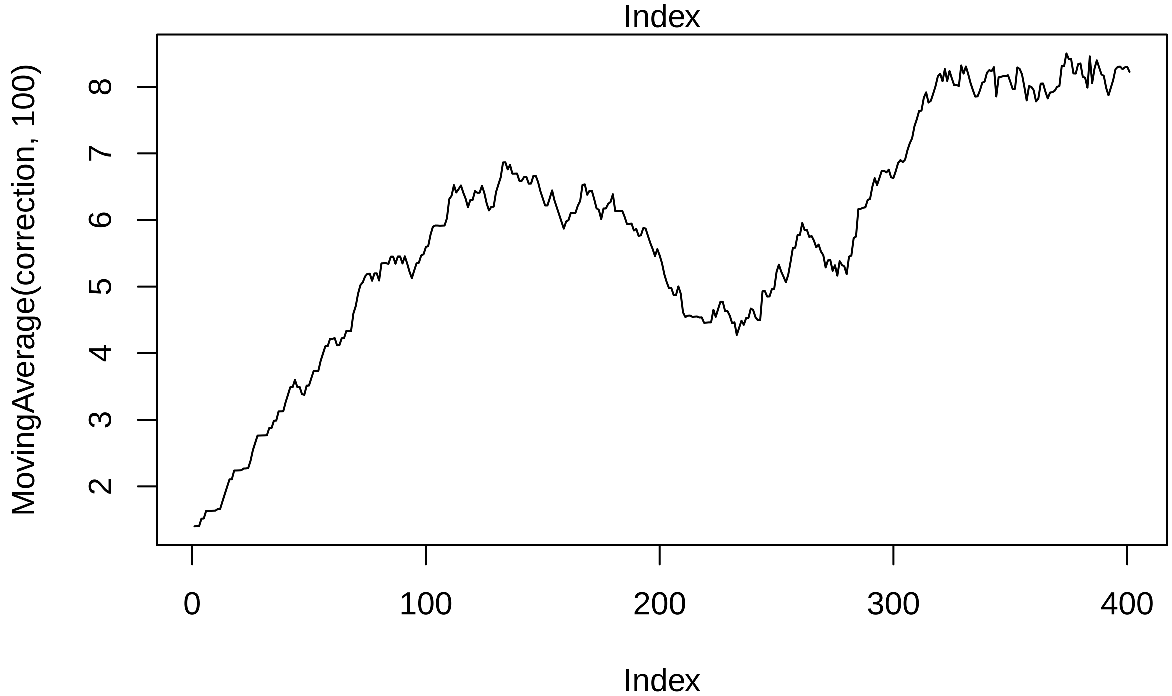

Q−table after 500 iterations
(epsilon = 0.5 , alpha = 0.001 gamma = 1 , beta = 10 )
x
y
10.0
7.5
5.0
2.5
0.0

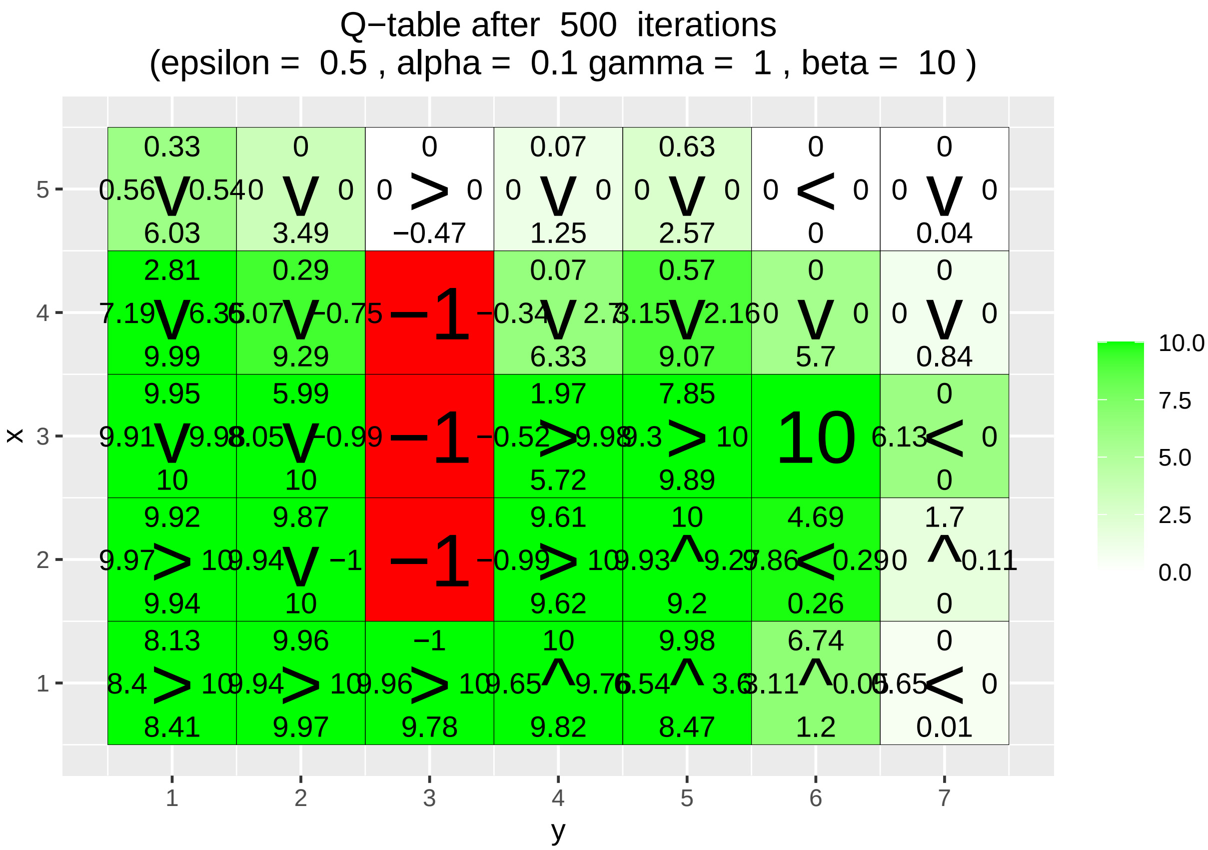

MovingAverage(reward, 100)
Index


MovingAverage(correction, 100)
1.0
2.0
3.0
4.0
0
100
200
300
400
Index

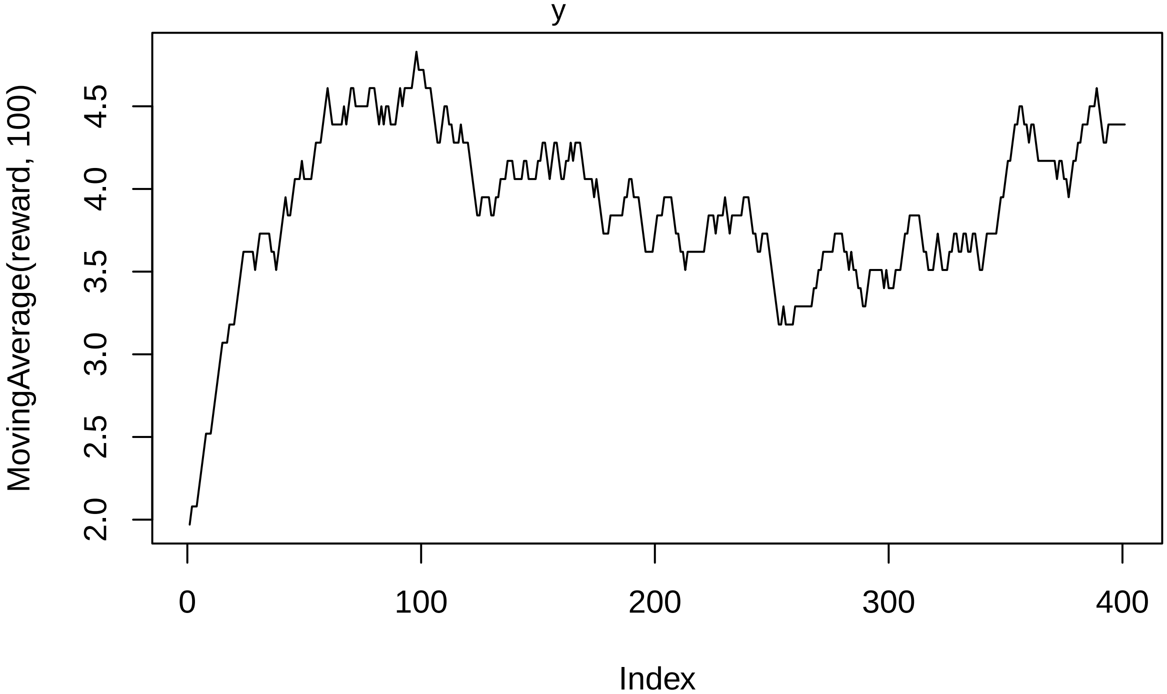

Q−table after 500 iterations
(epsilon = 0.5 , alpha = 0.01 gamma = 1 , beta = 10 )
x
y
10.0
7.5
5.0
2.5
0.0


MovingAverage(reward, 100)
5
4
3
2
0
100
200
300
400
Index


MovingAverage(correction, 100)
8
7
6
5
4
3
2
0
100
200
300
400
Index



Q−table after 500 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 1 , beta = 10 )
x
y
10.0
7.5
5.0
2.5
0.0

MovingAverage(reward, 100)
Index

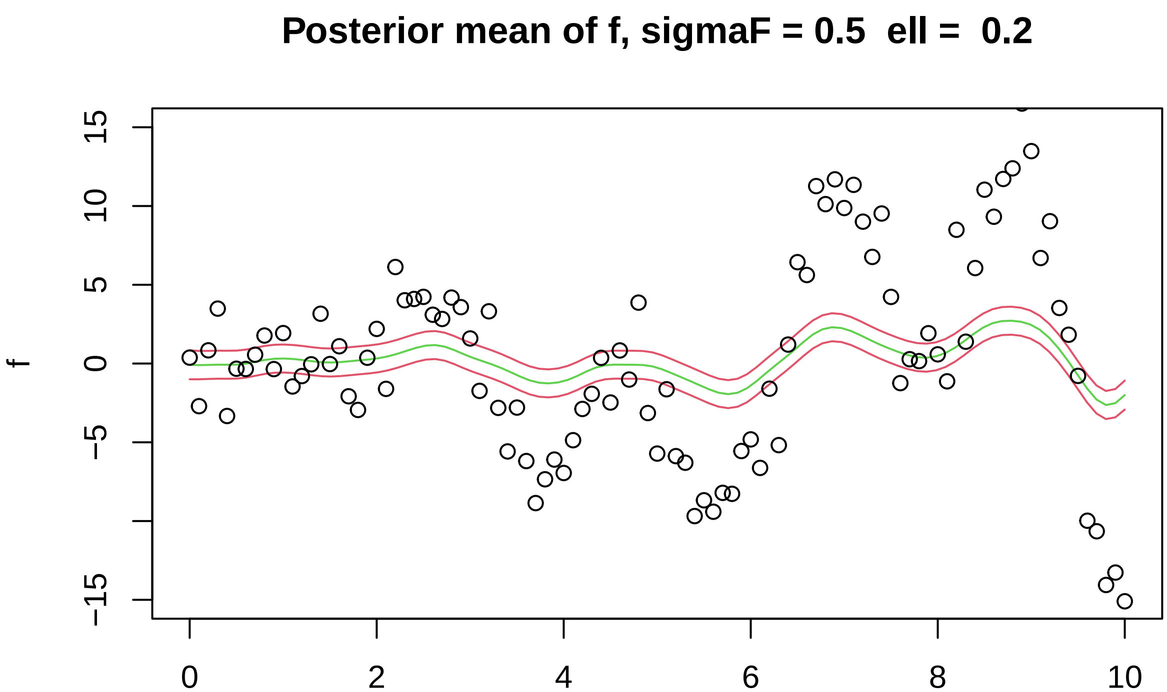

With a $\gamma = 1$, there is no discounting of future rewards, meaning that the agent will always strive for the +10 (which is the only reward in this case). This is why the q-values for the shortest path to ten is almost always around 10. The learning rate $\alpha$ has an impact in how much the q-table is updated after taking an action. A high learning rate will therefore have a larger impact on the q-table after a fixed amount of iterations compared to a lower learning rate. Even though the correction graph on $\alpha = 0.1$ does not go down to 0, we can see that the q-table is less and less updated since it is reaching convergence. Similarly, the reward average over episodes is somewhat converged, meaning that the agent performs similarly between episodes. The agents with lower values on $\alpha$ does not achieve the same results over 500 iterations.

# GP

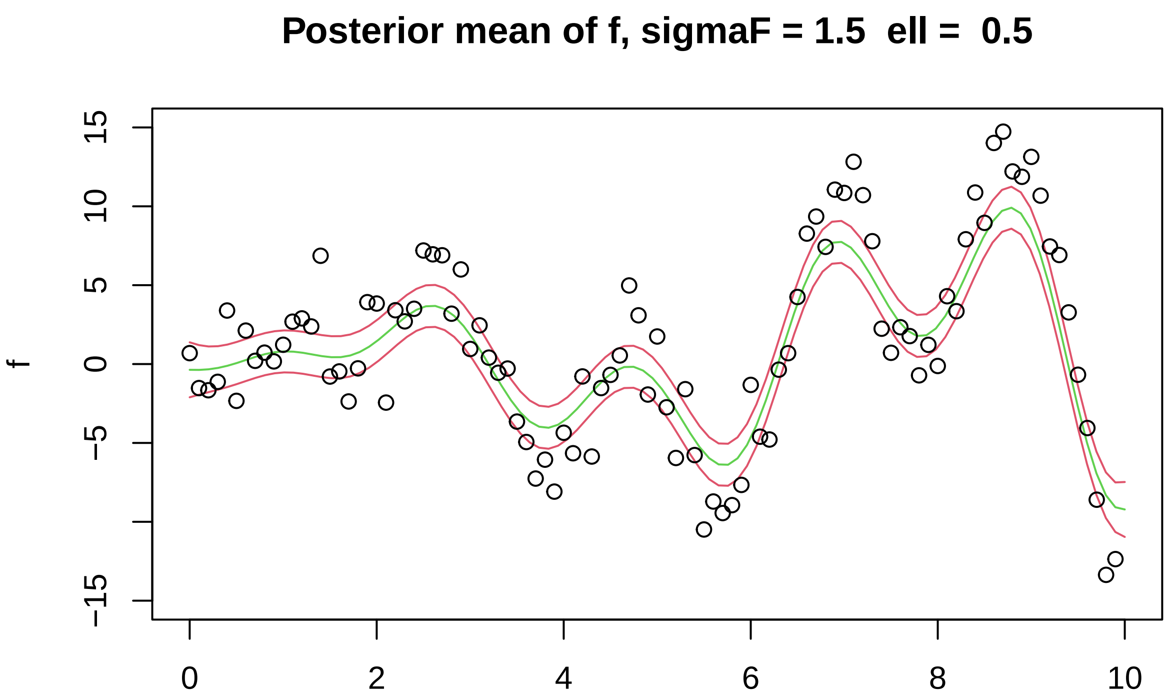
### Part 1

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

```r
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)
}

# Initialize paramters
sigmaF = 0.5
ell = 0.2 # ell = 0.2 seems to fit the data visually best
sigmaN = 2
xGrid = seq(0,10,length = 100)

X<-seq(0,10,.1)

Yfun<-function(x){
  return (x*(sin(x)+sin(3*x))+rnorm(length(x),0,2))
}
x = X
y = Yfun(x)

posterior = posteriorGP(X=x, y=y, sigmaNoise=sigmaN,
                        XStar=xGrid, k = SquaredExpKernel, sigmaF, ell)
plot(x = xGrid, y = posterior$mean, type = "l", col = 3,
     ylim = c(-15,15), ylab = "f", xlab = "", main = paste("Posterior mean of f, sigmaF =",
                                                    sigmaF," ell = ",ell))
lines(x = xGrid, y =posterior$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,Yfun(X),xlim=c(0,10),ylim=c(-15,15))
```



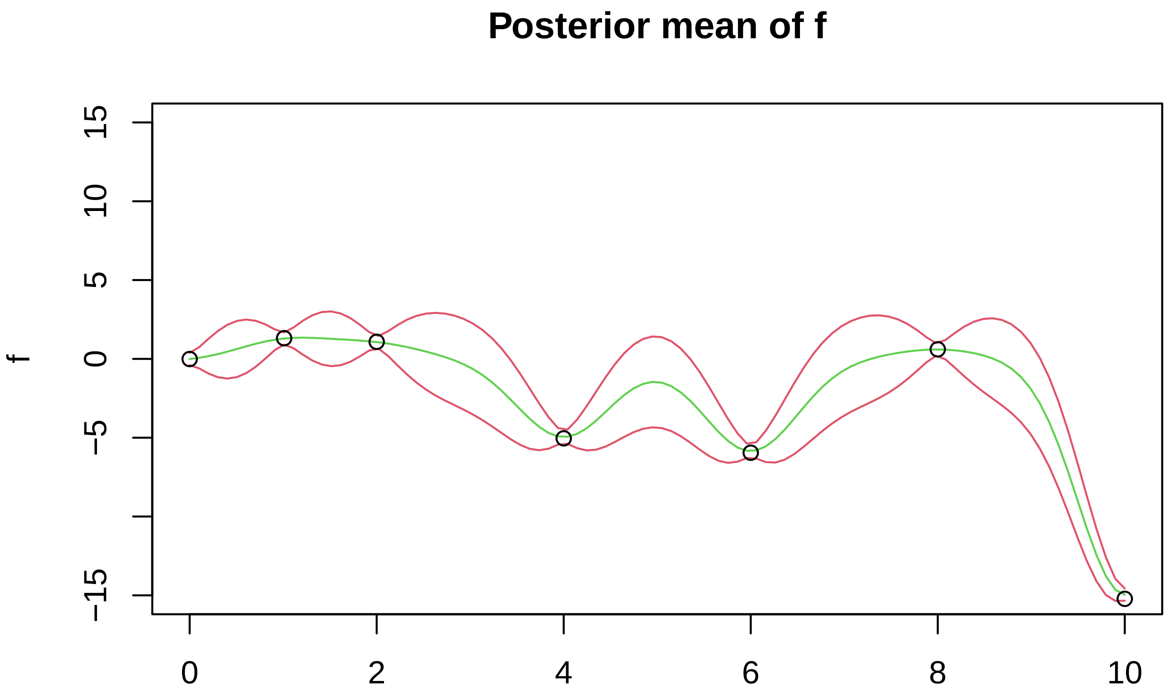

```r
# Changing sigmaF and ell
sigmaF = 1.5
ell = 0.5

posterior = posteriorGP(X=x, y=y, sigmaNoise=sigmaN,
                        XStar=xGrid, k = SquaredExpKernel, sigmaF, ell)
plot(x = xGrid, y = posterior$mean, type = "l", col = 3,
     ylim = c(-15,15), ylab = "f", xlab = "", main = paste("Posterior mean of f, sigmaF =",
                                                        sigmaF," ell = ",ell))
lines(x = xGrid, y =posterior$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,Yfun(X),xlim=c(0,10),ylim=c(-15,15))
```

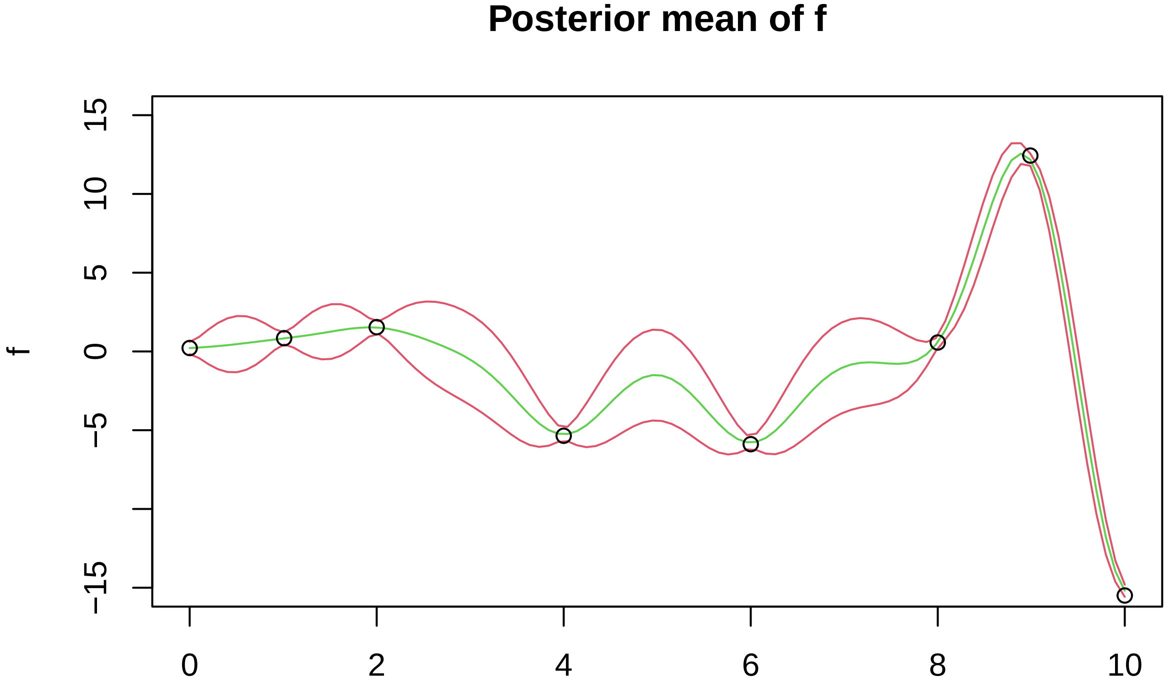


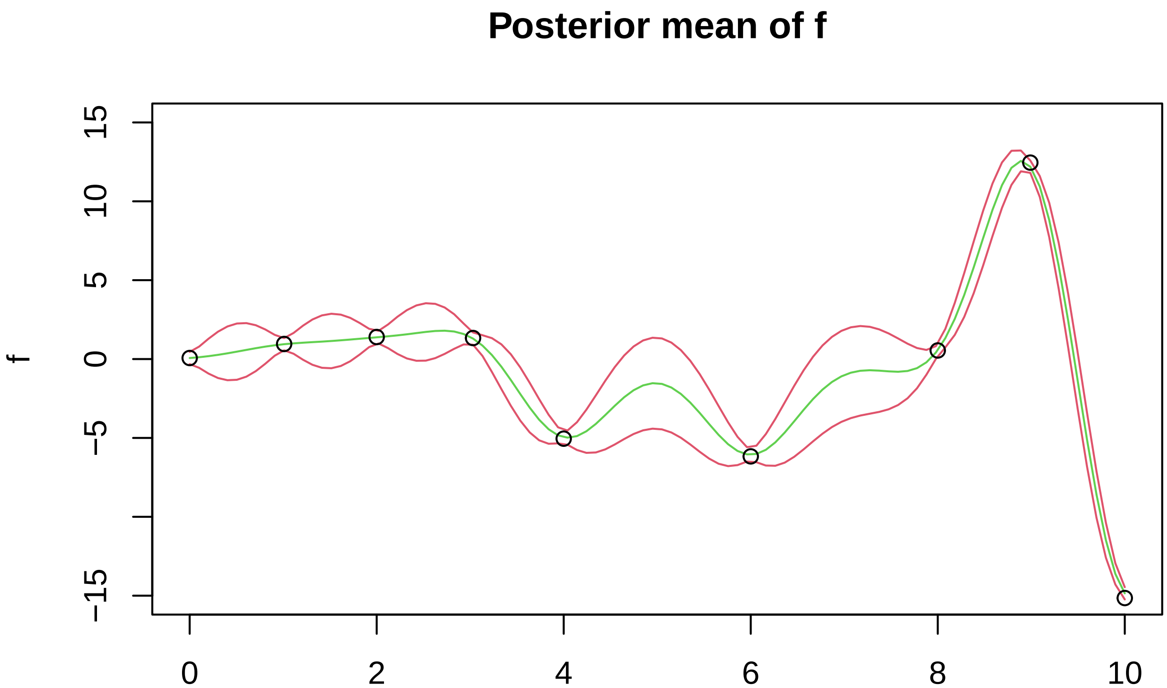

**Part 2**

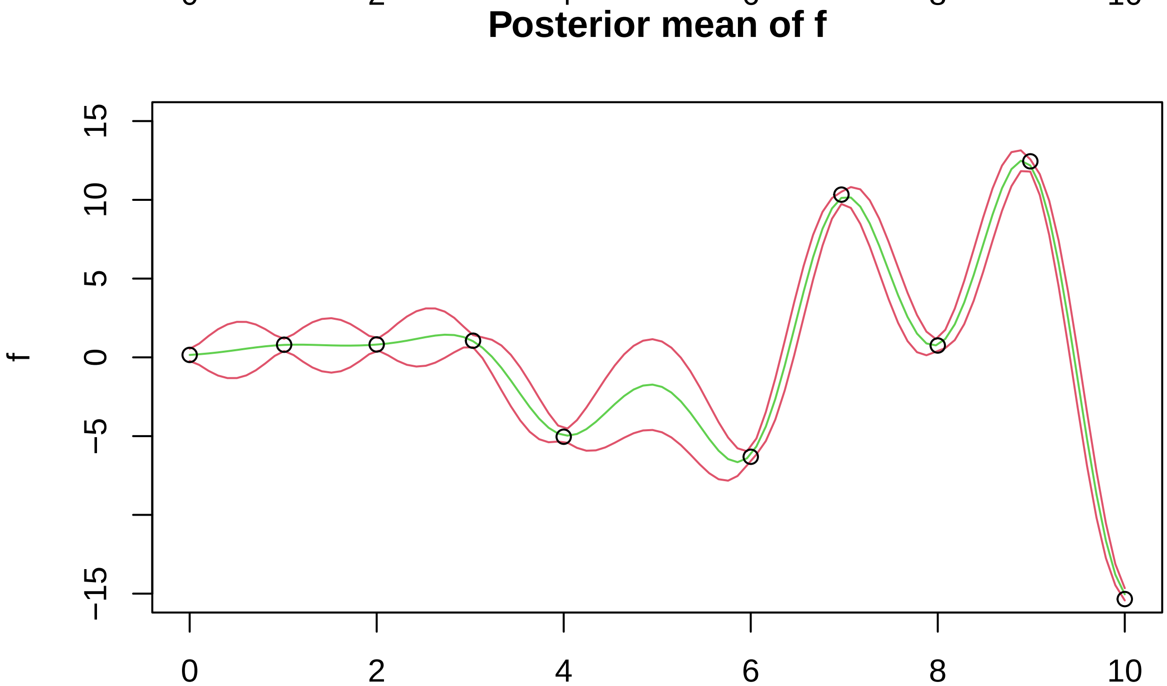
```
X<-seq(0,10,2)

Yfun<-function(x){
  return (x*(sin(x)+sin(3*x))+rnorm(length(x),0,.2))
}

sigmaF = 1.5
ell = 0.5
sigmaN = .2
xGrid = seq(0,10,length = 100)
x = X
y = Yfun(x)

posterior = posteriorGP(X=x, y=y, sigmaNoise=sigmaN,
                        XStar=xGrid, k = SquaredExpKernel, sigmaF, ell)

xvals = X


for (i in 1:4) {
  nextX = which.max(2*1.96*sqrt(diag(posterior$variance)))
  xvals = c(xvals, xGrid[nextX])
  y = Yfun(xvals)
  posterior = posteriorGP(X=xvals, y=y, sigmaNoise=sigmaN,
                          XStar=xGrid, k = SquaredExpKernel, sigmaF, ell)
  plot(x = xGrid, y = posterior$mean, type = "l", col = 3, xlim = c(0,10),
     ylim = c(-15,15), ylab = "f", xlab = "", main = paste("Posterior mean of f"))
  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(xvals,y,xlim=c(0,10),ylim=c(-15,15))

}
```




Posterior mean of f
f
15
10
5
0
−5
−15
0
2
4
6
8
10


Posterior mean of f
f
15
10
5
0
−5
−15
0
2
4
6
8
10