STAT340 Project 1

Pratik Nag

Question 1.

We have first defined the the matrix with M=8 rows and N=15 columns.
Here at first we are defining three functions, Let is look at the following codes.

```
M <- 8
N <- 15
set.seed(22309988)
A <- matrix(sample(1:(N*M), N*M, replace = FALSE), M, N)

BB= matrix(NA,M,N)



plot_figure <- function() {
  BB <- A
  for(j in 1:N)
    BB[1:M, j] = A[M:1, j]
  image(x = seq(from = 0.5, to = N + 0.5), y = seq(from = 0.5, to = M +
                  0.5), z = t(BB), col = gray.colors(n = M * N,
                   start = 0, end = 1), xlab = "", ylab = "", axes = FALSE, asp = 1)
  return(BB)
}
plot_numbers <- function(cex = 1) {
  for (i in 1:M) {
  for (j in 1:N) {
  text(j, i, as.character(BB[i,j]),
       col = "brown", cex = cex)
  }
  }
}

plot_path <- function(path, lwd = 2, col = "blue4") {
  xc = c(1:N)
  yc = M - path + 1
  lines(xc, yc, lwd = lwd, col = col)
}

BB = plot_figure()
plot_numbers()
```

105	33	2	63	90	30	111	64	19	113	22	6	116	120	56
109	97	57	10	35	71	86	62	54	55	32	81	49	70	27
92	7	37	98	50	66	102	84	75	112	13	1	39	26	17
46	115	36	87	118	79	53	47	52	5	65	14	69	21	67
18	38	45	85	107	95	16	44	59	101	25	119	58	93	40
80	103	68	82	8	24	110	41	117	73	104	60	89	23	74
28	61	4	15	43	108	51	114	88	3	77	106	94	29	99
91	78	100	72	11	34	42	12	83	48	9	31	20	96	76

The above matrix named A is a MXN matrix
Now let us write the algorithm for finding the smoothing path using Dynamic programming approach.

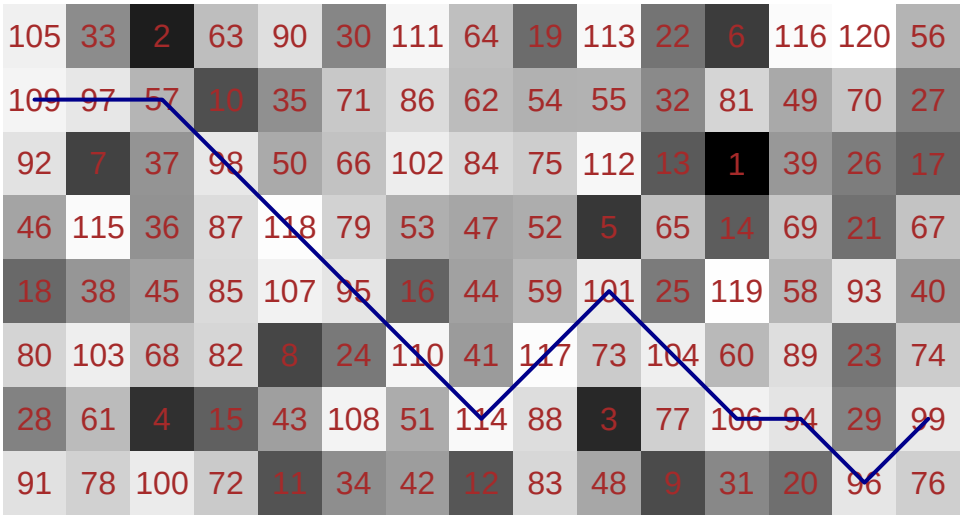
```
H <- matrix(NA, M, N)
D <- 1
H[, 1] <- A[, 1]

for(j in 2:N) {
    for(i in 1:M) {
        i.low <- max(1, i-D)
        i.high <- min(M, i+D)
        ii <- i.low -1 + which.max(H[i.low:i.high, j-1] + A[i, j])
        H[i, j] <- H[ii, j-1] + A[i, j]
    }
}

xs <- numeric(N)
xs[N] <- which.max(H[, N])
for(j in (N-1):1) {
        i.low <- max(1, xs[j+1]-D)
        i.high <- min(M, xs[j+1]+D)
        ii <- i.low -1 + which.max(H[i.low:i.high, j])
        xs[j] <- ii
}
```

Here the list xs is our most likely path.

Now let us plot the best possible path in the grayscale image generated above with the help of plot_path() function that I created.
below is the whole code in a single window.

```
BB= matrix(NA,M,N)
set.seed(22309988)
plot_figure <- function() {
  BB <- A
  for(j in 1:N)
    BB[1:M, j] = A[M:1, j]
  image(x = seq(from = 0.5, to = N + 0.5), y = seq(from = 0.5, to = M +
                  0.5), z = t(BB), col = gray.colors(n = M * N,
                   start = 0, end = 1), xlab = "", ylab = "", axes = FALSE, asp = 1)
  return(BB)
}
plot_numbers <- function(cex = 1) {
  for (i in 1:M) {
  for (j in 1:N) {
  text(j, i, as.character(BB[i,j]),
       col = "brown", cex = cex)
  }
  }
}

plot_path <- function(path, lwd = 2, col = "blue4") {
  xc = c(1:N)
  yc = M - path + 1
  lines(xc, yc, lwd = lwd, col = col)
}

M <- 8
N <- 15
A <- matrix(sample(1:(N*M), N*M, replace = FALSE), M, N)
H <- matrix(NA, M, N)
D <- 1
H[, 1] <- A[, 1]

BB = plot_figure()
plot_numbers()
for(j in 2:N) {
  for(i in 1:M) {
    i.low <- max(1, i-D)
    i.high <- min(M, i+D)
    ii <- i.low -1 + which.max(H[i.low:i.high, j-1] + A[i, j])
    H[i, j] <- H[ii, j-1] + A[i, j]
  }
}

xs <- numeric(N)
xs[N] <- which.max(H[, N])
for(j in (N-1):1) {
  i.low <- max(1, xs[j+1]-D)
  i.high <- min(M, xs[j+1]+D)
```

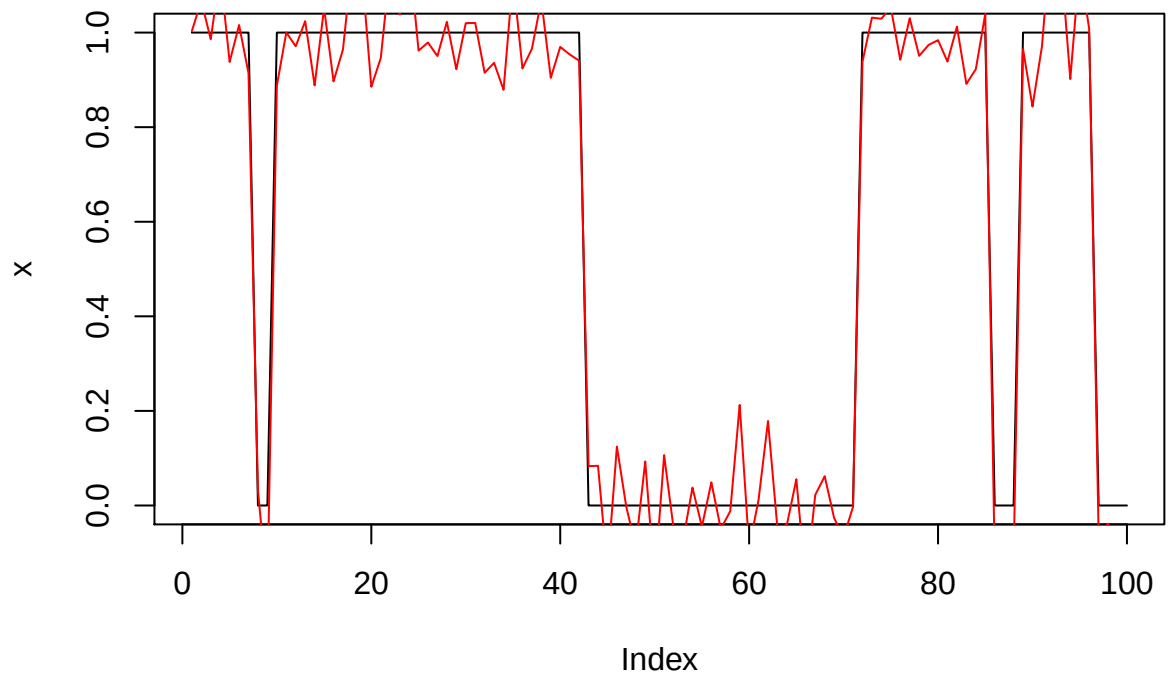
```
  ii <- i.low -1 + which.max(H[i.low:i.high, j])
  xs[j] <- ii
}
plot_path(path = xs)
```



Question 2.

In this problem I am first assigning some initial values to the parameters p, q, σ^2 and for the specified p will simulate a dataset X and consequently Y's for our validation purposes. Here I have taken $T = 100$ and $p = 0.90, \sigma^2 = 0.1$ for simulation. Code is given below.

```
p <- 0.90
T = 100
sigma_2 <- 0.1
set.seed(234)
x = rep(NA,T)
y = rep(NA,T)

x[1] = 1
for (i in 2:T){
  if (runif(1) < p){
    x[i] = x[i-1]
  }else{
    x[i] = 1-x[i-1]
  }
}
y = rnorm(T,x,sigma_2)
plot(x,type = "l")
lines(y,col = "red")
```



We will use this generated value of Y for our calculations.

Filtering

We will be using the following formula for filtering

$$\pi(X_t|Y_1, Y_2, ..., Y_t) \propto \pi(X_t|Y_1, Y_2, ..., Y_{t-1})\pi(Y_t|X_t) \ for\ all\ t = 1, 2, ..., T$$
$$where\ the\ normalizing\ constant\ is\ Z_t = \pi(Y_t|Y_1, Y_2, .., Y_{t-1}) = \sum_{X_t} \pi(X_t|Y_1, Y_2, ..., Y_t)$$

Here the first term in the expression is the prediction for time point t.

Prediction

Formula for Prediction :

$$\pi(X_t|Y_1, Y_2, ..., Y_{t-1}) = \sum_{X_{t-1}} \pi(X_t|X_{t-1})\pi(X_{t-1}|Y_1, Y_2, ..., Y_{t-1})$$

The second term is basically the filtering at stage $t-1$.

Code for filtering and prediction

Code for the above calculation.

```
# we have defined state 1 when x[t] = 1 and state 2 otherwise

filtering_state_1 = rep(NA,T)
filtering_state_2 = rep(NA,T)
prediction_state_1 = rep(NA,T)
prediction_state_2 = rep(NA,T)
Z = rep(NA,T)

prop_filtering_function <- function(prediction,dist_func){
  return(prediction*dist_func)
}

prediction_function <- function(p,filtering1,filtering2,state){
  if (state == 1){
    return(p*filtering1 + (1-p)*filtering2)
  }else{
    return((1-p)*filtering1 + p*filtering2)
  }
}

filtering_and_prediction <- function(p){
  for (t in 1:T){
    if (t == 1){
      prediction_state_1[t] = 0.5
      prediction_state_2[t] = 0.5
      prop_filtering_state_1 = prop_filtering_function(prediction_state_1[t],
                                                       dist_func_normal(y[t],1))
      prop_filtering_state_2 = prop_filtering_function(prediction_state_2[t],
```

```
                                                    dist_func_normal(y[t],0))
      norm_constant = prop_filtering_state_1 + prop_filtering_state_2
      filtering_state_1[t] = prop_filtering_state_1/norm_constant
      filtering_state_2[t] = prop_filtering_state_2/norm_constant
      Z[t] = norm_constant
    }else{
      prediction_state_1[t] = prediction_function(p,filtering_state_1[t-1],
                                                  filtering_state_2[t-1],1)
      prediction_state_2[t] = prediction_function(p,filtering_state_1[t-1],
                                                  filtering_state_2[t-1],0)
      prop_filtering_state_1 = prop_filtering_function(prediction_state_1[t],
                                                    dist_func_normal(y[t],1))
      prop_filtering_state_2 = prop_filtering_function(prediction_state_2[t],
                                                    dist_func_normal(y[t],0))
      norm_constant = prop_filtering_state_1 + prop_filtering_state_2
      filtering_state_1[t] = prop_filtering_state_1/norm_constant
      filtering_state_2[t] = prop_filtering_state_2/norm_constant
      Z[t] = norm_constant
    }
  }
  return(matrix(c(filtering_state_1,filtering_state_2,prediction_state_1,
                  prediction_state_2,Z),nrow = 5, byrow = TRUE))
}
```

Smoothing

Given $Y_1, Y_2, ..., Y_T$ the smoothing at time point $t = s$ where s = 1,2,. . . ,T is

$$\pi(X_s|Y_{1:T}) = \sum_{X_{s+1}} \frac{\pi(X_{s+1}|X_s)\pi(X_s|Y_{1:s})\pi(X_{s+1}|Y_{1:T})}{\pi(X_{s+1}|Y_{1:s})}$$

Which is basically a function of filtering at time point s, smoothing at time point $s + 1$ and prediction at time point $s + 1$.

Smoothing code.

```
# we have defined state 1 when x[t] = 1 and state 2 otherwise

smoothing_state_1 = rep(NA,T)
smoothing_state_2 = rep(NA,T)

smoothing_function <- function(p,filtering,smoothing1,prediction1,smoothing2,
                               prediction2,state){
  if (state == 1){
    return((p*filtering*smoothing1/prediction1) +
             ((1-p)*filtering*smoothing2/prediction2))
  }else{
    return(((1-p)*filtering*smoothing1/prediction1) +
             (p*filtering*smoothing2/prediction2))
  }
```

```
}

smoothing <- function(p){
  for (t in T:1){
    if (t == T){
      smoothing_state_1[t] = filtering_state_1[t]
      smoothing_state_2[t] = filtering_state_2[t]
    }else{
      smoothing_state_1[t] = smoothing_function(p,filtering_state_1[t],
                                                smoothing_state_1[t+1],
                                                prediction_state_1[t+1],
                                                smoothing_state_2[t+1],
                                                prediction_state_2[t+1],1)
      smoothing_state_2[t] = smoothing_function(p,filtering_state_2[t],
                                                smoothing_state_1[t+1],
                                                prediction_state_1[t+1],
                                                smoothing_state_2[t+1],
                                                prediction_state_2[t+1],0)
    }
  }
  return(return(matrix(c(smoothing_state_1,smoothing_state_2),nrow = 2, byrow = TRUE)))
}
```

Calculation when Y follows normal conditioned on X

Now for the first case when,

$$\pi(Y_t|X_t) \sim N(X_t, \sigma^2)$$

Code:

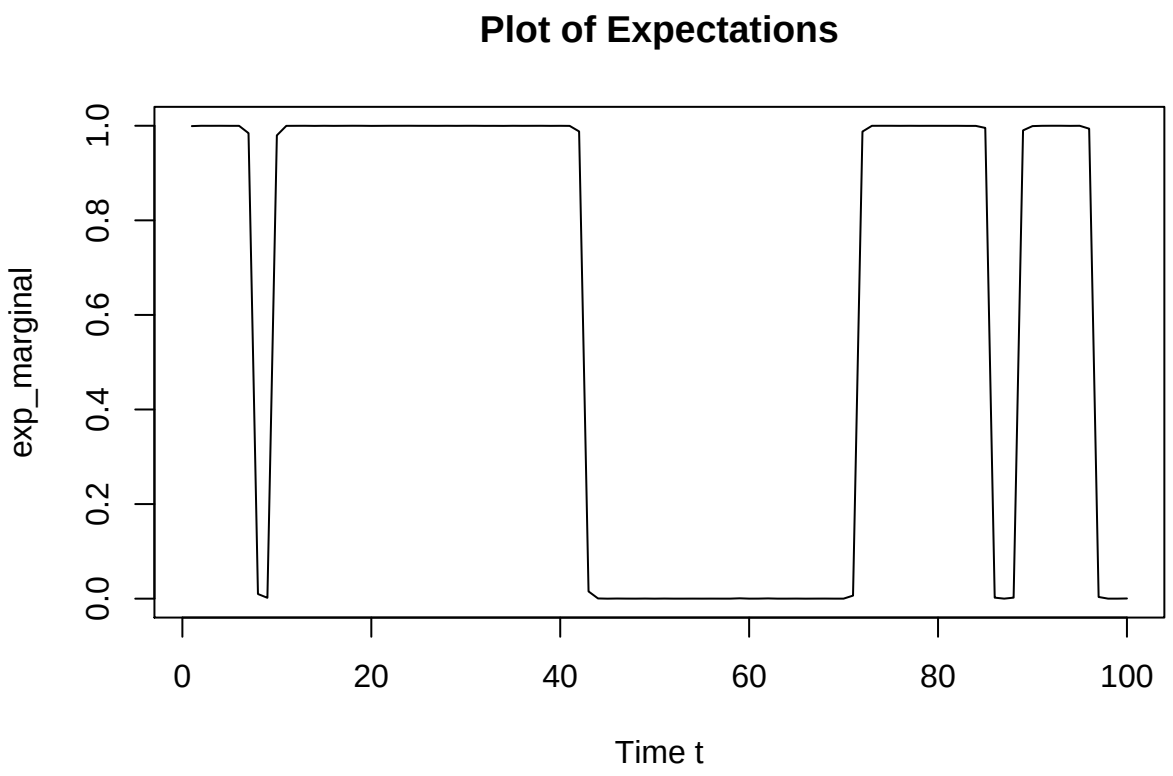
```
dist_func_normal <- function(y,x){
  return(dnorm(y,x,sqrt(sigma_2)))
}

maximizer <- function(a,b){
  if (a > b){
    return(1)
  }else{
    return(0)
  }
}

# Calculation of filtering and prediction for p = 0.9 as stated earlier
k = filtering_and_prediction(p)
filtering_state_1 = k[1,]
filtering_state_2 = k[2,]
prediction_state_1 = k[3,]
prediction_state_2 = k[4,]

l = smoothing(p)
smoothing_state_1 = l[1,]
smoothing_state_2 = l[2,]
```

1. For pre-defined values of θ we then can find the marginal mean and standard deviation, which is given by

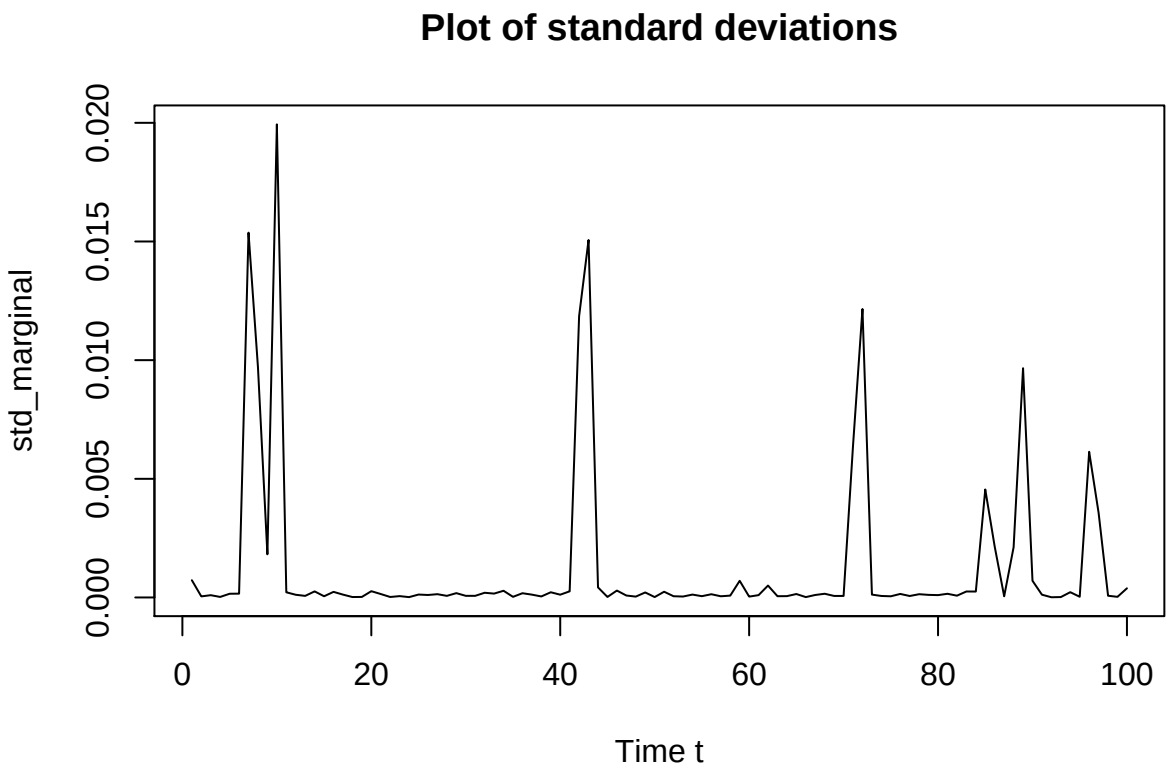
$$E(X_t|Y_{1:T}) = \ P(X_t = 1|Y_{1:T}) = \text{Probability of smoothing at time point t}$$
$$V(X_t|Y_{1:T}) = \ P(X_t = 1|Y_{1:T})(1 - \ P(X_t = 1|Y_{1:T}))$$

Code : Here we are plotting the expectation and standard deviation over the time points.

```
exp_marginal = smoothing_state_1
std_marginal = smoothing_state_1*smoothing_state_2
plot(exp_marginal,type = "l",xlab = "Time t")
title(main = "Plot of Expectations" )
```




```
plot(std_marginal,type="l",xlab = "Time t")
title(main = "Plot of standard deviations")
```

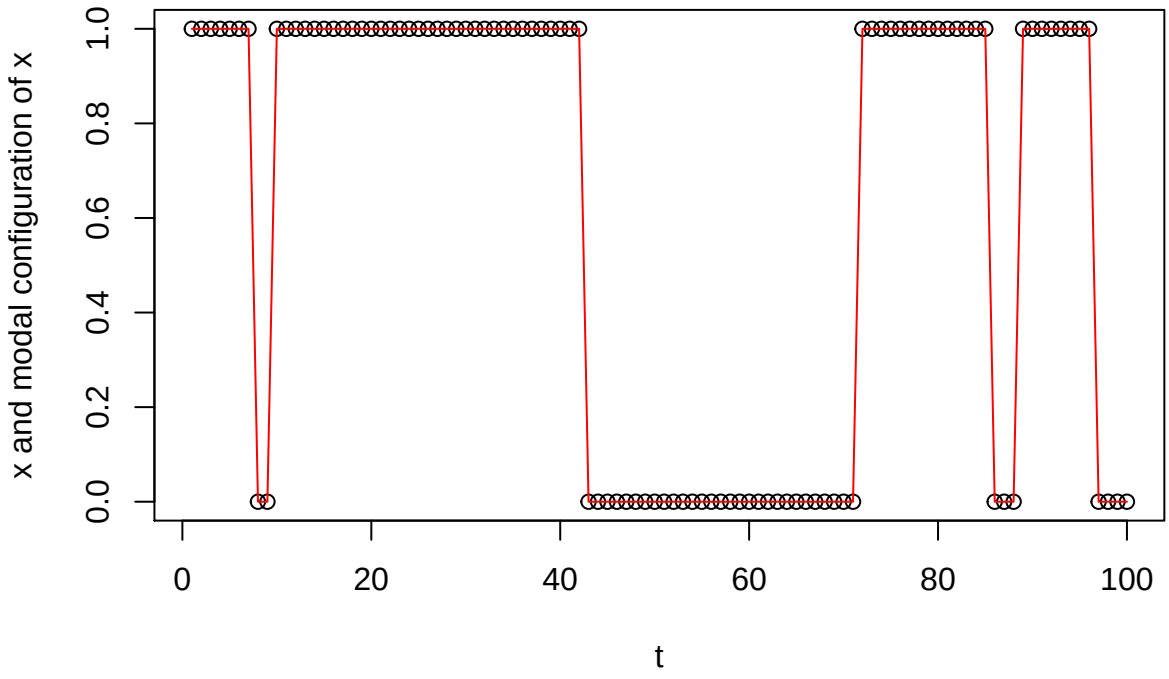


Now to find the modal configuration of X, we will have to do sampling over $\pi(X_{1:T}|Y_{1:T})$ and find the mode from there.

Let us look at the following code:

```
maximizer <- function(a,b){
  if (a > b){
    return(1)
  }else{
    return(0)
  }
}
mod_x = rep(NA,T)
Modal_config <- function(p){
  mod_x[T] = maximizer(filtering_state_1[T],filtering_state_2[T])

  for (t in (T-1):1){
    if (mod_x[t+1] == 1){
      sampling_state_1 = p*filtering_state_1[t]
      sampling_state_2 = (1-p)*filtering_state_2[t]
    }else{
      sampling_state_1 = (1-p)*filtering_state_1[t]
      sampling_state_2 = p*filtering_state_2[t]
    }
    mod_x[t] = maximizer(sampling_state_1,sampling_state_2)
```

```
  }
  return(mod_x)
}
mod_x = Modal_config(p)

# plotting X and modal configuration of X
plot(x, xlab = "t",ylab = "x and modal configuration of x")
lines(mod_x, col = "red")
```



Hence from the above plot it is quite evident that the xodal configuration of X matches with X very well.

2. Here we have to find posterior mode and distribution of p.
Let us first assume that the prior distribution of p is Uniform(0,1) i,e; p $\sim U(0,1)$
Now we have to find

$$\begin{aligned} p^* &= \ argmax\ \pi(p|Y_{1:T}) \\ &= \ argmax\ \pi(p)\pi(Y_{1:T}|p) \end{aligned}$$

$$Here\ \pi(Y_{1:T}|p)\ is\ proportional\ to \prod_{i=1}^{T} Z_i$$

$$and\ \pi(p) \sim U(0,1)$$

$$Hence\ p^* = argmax\ \prod_{i=1}^{T} Z_i$$

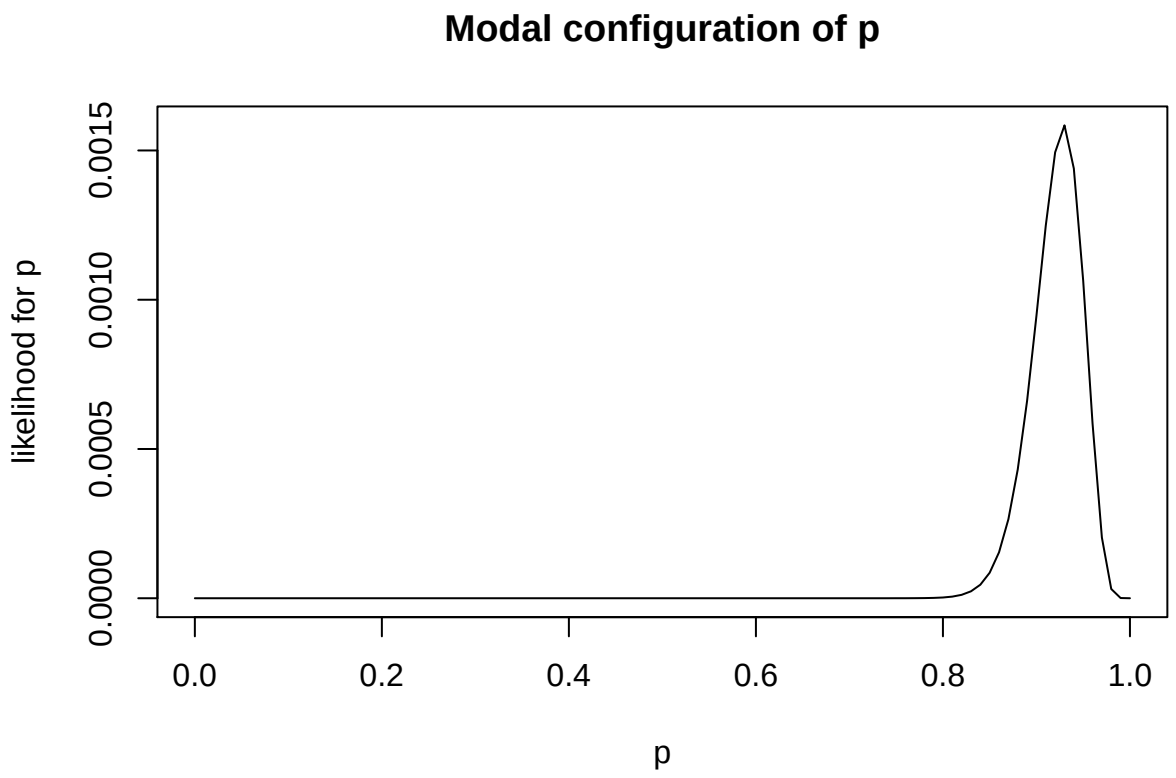
So, if we plot this function with respect to p we will find the section where p maximizes.

Also by normalizing the product of Z_i's we can find the posterior distribution of p.

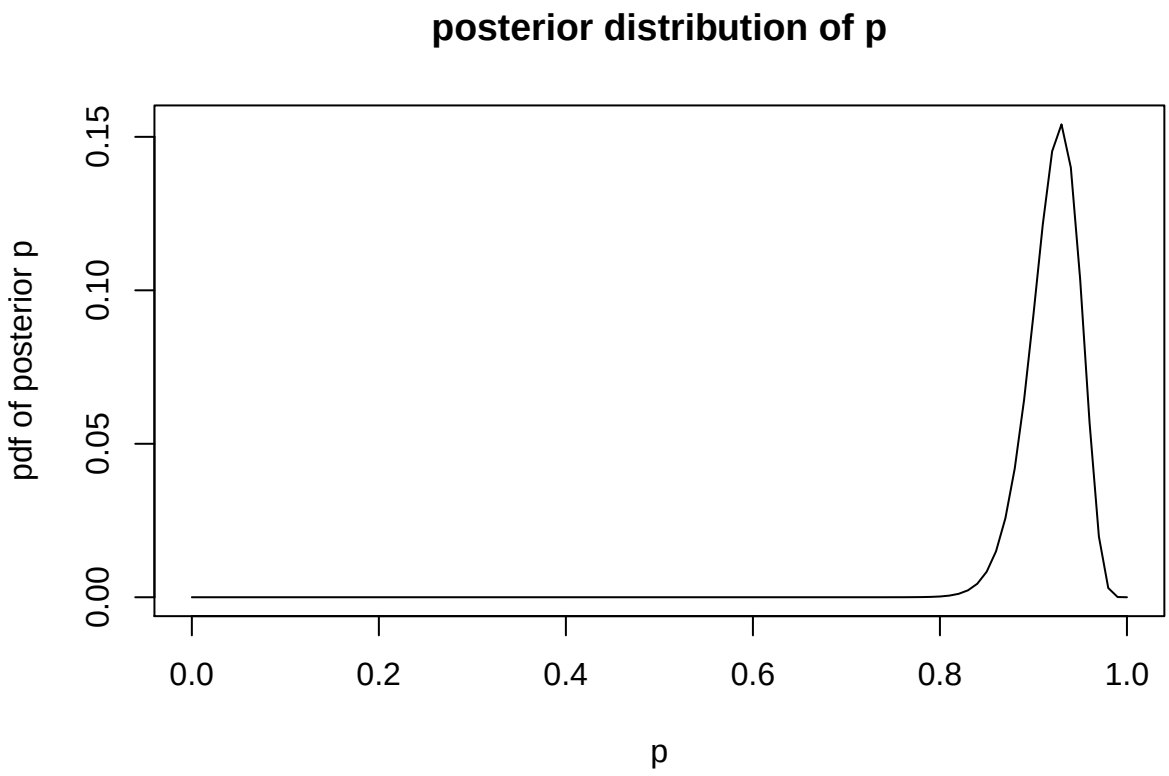
In the following code we have plotted the for both the cases.

code:

```
# modal value of p

sequence = seq(from = 0, to = 1 , by = 0.01)
likelihood = rep(NA,length(sequence))
for (i in 1:length(sequence)){
  k = filtering_and_prediction(sequence[i])
  Z = k[5,]
  likelihood[i] = prod(Z)
}
plot(sequence,likelihood, type = "l", xlab = "p", ylab = "likelihood for p")
title(main = "Modal configuration of p")
```




```
# posterior distribution of P
normalizing_p = sum(likelihood)

posterior_p = likelihood/normalizing_p
plot(sequence,posterior_p, type = "l", xlab = "p", ylab = "pdf of posterior p" )
title(main = "posterior distribution of p")
```



So we see the modal value of p matches with our initially defined value of p.

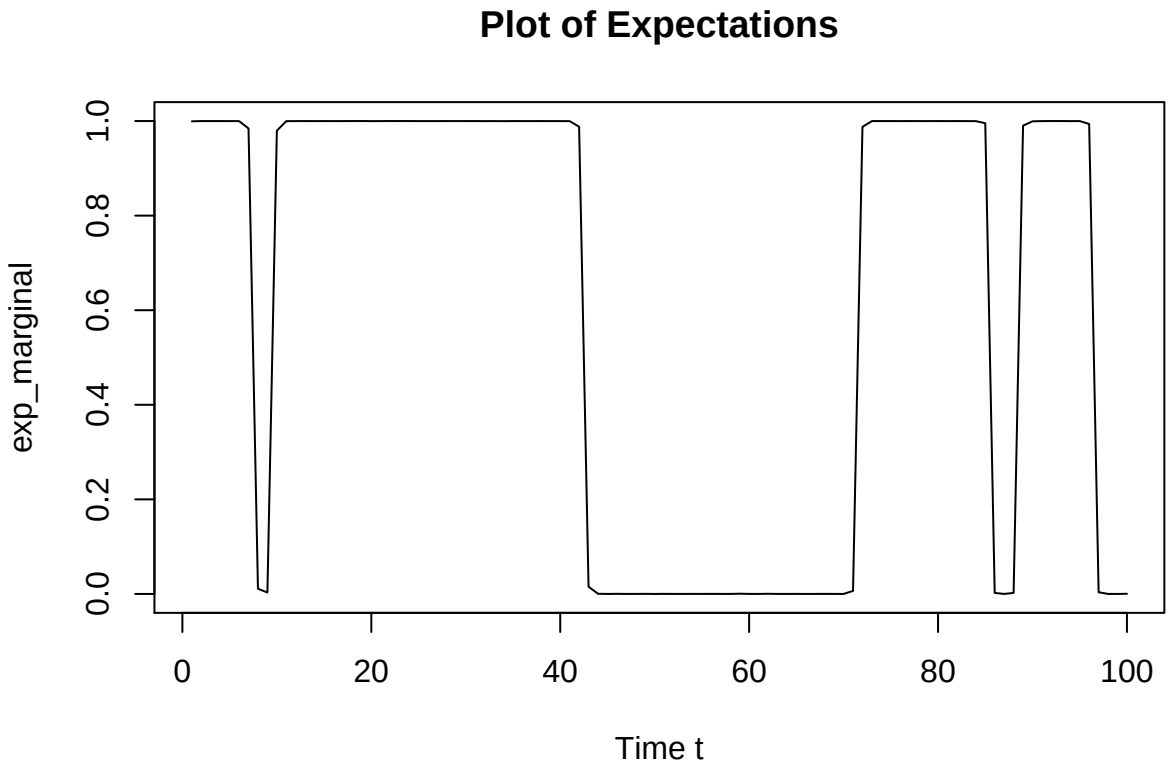
3. Here we have to do the same as 1. but after taking the uncertainty of p into consideration. Let us look at the following.

$$Here\ E(X_t|Y_{1:T}, p) = \ smoothing\ state\ 1\ probability\ conditioned\ on\ p.$$
$$E(X_t|Y_{1:T}, p) \approx \sum_p E(X_t|Y_{1:T}, p_i)\ \pi(p_i|Y_{1:T})$$

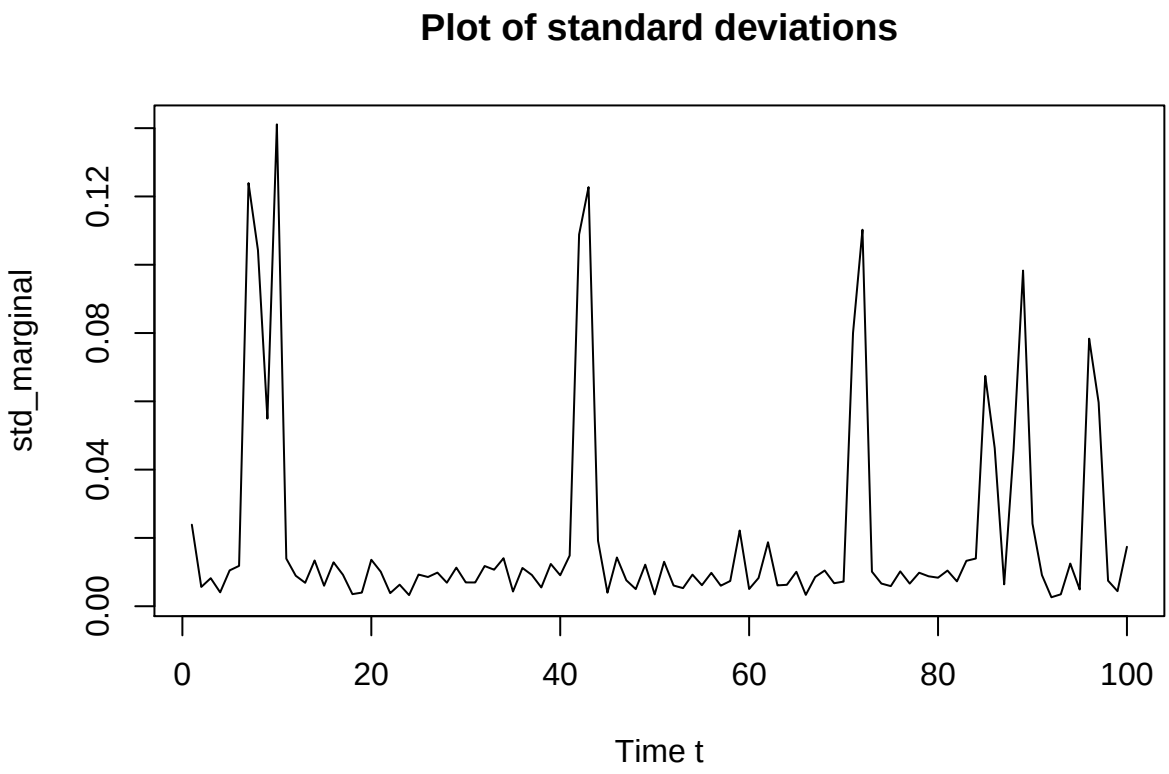
Writing this in code we get.

```
# marginal expectation conditioned on p
exp_marginal = rep(NA,T)
for (t in 1:T){
  exp_part = rep(NA,length(sequence))
  for (i in 1:length(sequence)){
    k = filtering_and_prediction(sequence[i])
    filtering_state_1 = k[1,]
    filtering_state_2 = k[2,]
    prediction_state_1 = k[3,]
    prediction_state_2 = k[4,]
    Z = k[5,]
    l = smoothing(sequence[i])
    smoothing_state_1 = l[1,]
    exp_part[i] = smoothing_state_1[t]*(prod(Z)/normalizing_p)
  }
```

```
  exp_marginal[t] = sum(exp_part)
}

exp_marginal_2 = exp_marginal

var_marginal = exp_marginal_2 - exp_marginal^2
std_marginal = sqrt(var_marginal)

plot(exp_marginal,type = "l",xlab = "Time t")
title(main = "Plot of Expectations" )
```




```
plot(std_marginal,type="l",xlab = "Time t")
title(main = "Plot of standard deviations")
```



Here exp_marginal is the marginal expectation and var_marginal is the variance of the marginal both conditioned on p.

4. Here we have to find the modal configuration of (x,p) jointly.
Let us write the problem in the following way.

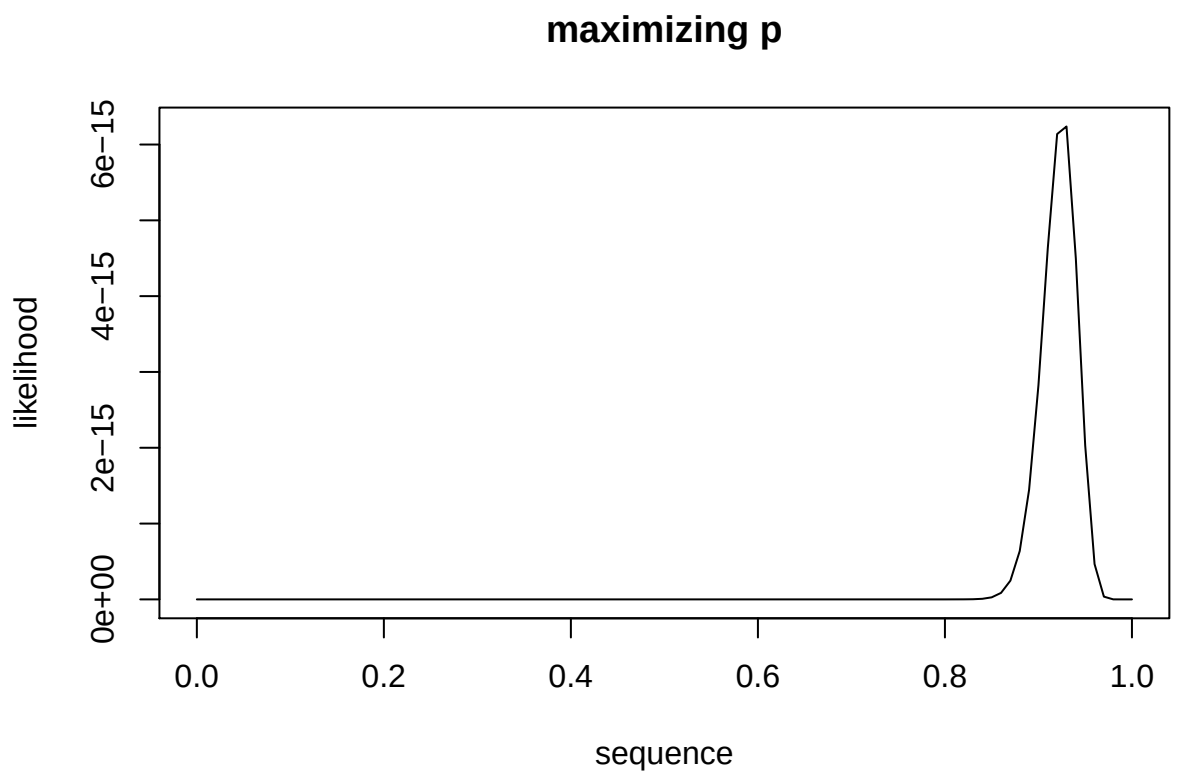
$$\begin{aligned} Here\ (p^*, X^*) = &\ argmax\ \pi(p, X_{1:T}|Y_{1:T}) \\ \propto &\ argmax\ \pi(X_{1:T}|Y_{1:T}, p)\ \pi(p|Y_{1:T}) \end{aligned}$$

So here we will first maximize X as a function of p then put that value above and maximize over p.

code:

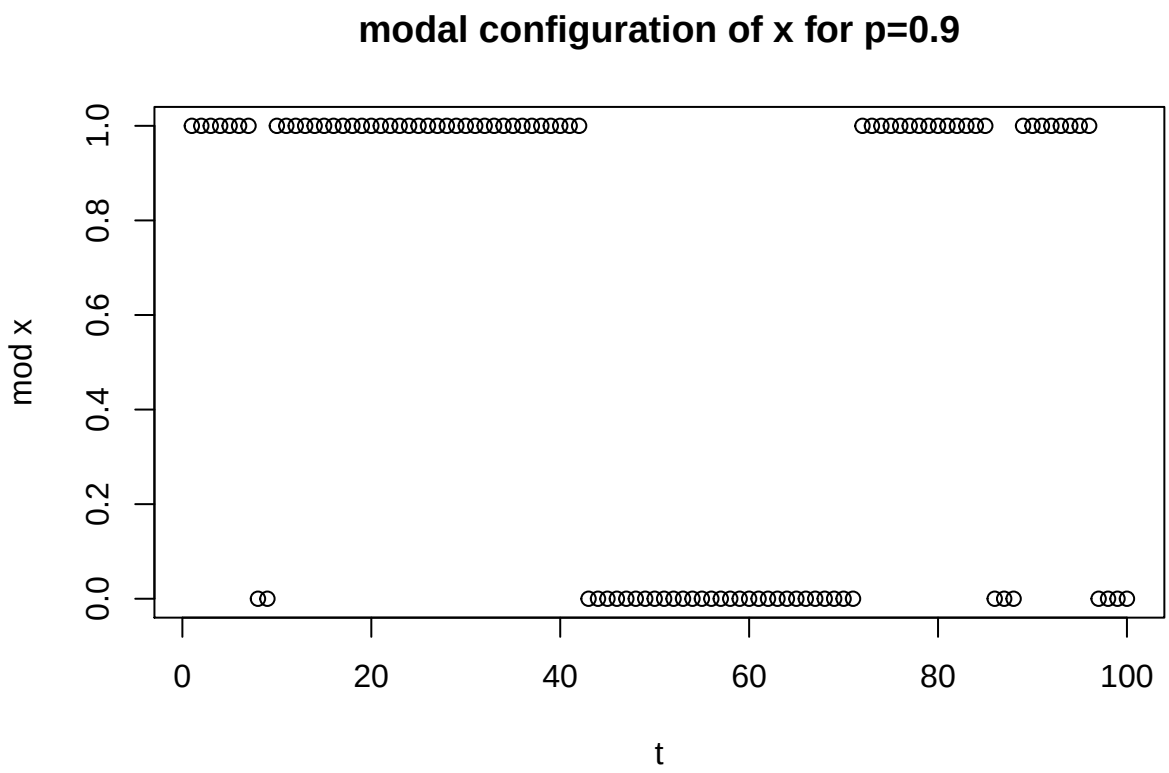
```
# Mod (p,x)

# Modal configuration of X conditioned on p
mod_x = rep(NA,T)
mod_x_dist = rep(NA,T)
filtering_state_1= rep(NA,T)
filtering_state_2= rep(NA,T)
Modal_config <- function(p,filtering_state_1,filtering_state_2){
  mod_x_dist[T] = max(filtering_state_1[T],filtering_state_2[T])
  mod_x[T] = maximizer(filtering_state_1[T],filtering_state_2[T])
  for (t in (T-1):1){
    if (mod_x[t+1] == 1){
      sampling_state_1 = p*filtering_state_1[t]
```

```
      sampling_state_2 = (1-p)*filtering_state_2[t]
    }else{
      sampling_state_1 = (1-p)*filtering_state_1[t]
      sampling_state_2 = p*filtering_state_2[t]
    }
    mod_x_dist[t] = max(sampling_state_1,sampling_state_2)
    mod_x[t] = maximizer(sampling_state_1,sampling_state_2)
  }
  return(prod(mod_x_dist))
}
sequence = seq(from = 0, to = 1 , by = 0.01)
likelihood = rep(NA,length(sequence))
for (i in 1:length(sequence)){
  k = filtering_and_prediction(sequence[i])
  filtering_state_1 = k[1,]
  filtering_state_2 = k[2,]
  Z = k[5,]
  mod_x_dist = Modal_config(sequence[i],filtering_state_1,filtering_state_2)
  likelihood[i] = prod(Z) * mod_x_dist
}

plot(sequence,likelihood,type="l")
title(main = "maximizing p")
```



Hence, From the plot it is evident that the mode for p is nearing 0.9 and putting this value in the modal configuration we will get the modal configuration of X. Code:

```r
p = 0.9
mod_x = rep(NA,T)
k = filtering_and_prediction(p)
filtering_state_1 = k[1,]
filtering_state_2 = k[2,]
Modal_config <- function(p){
  mod_x[T] = maximizer(filtering_state_1[T],filtering_state_2[T])

  for (t in (T-1):1){
    if (mod_x[t+1] == 1){
      sampling_state_1 = p*filtering_state_1[t]
      sampling_state_2 = (1-p)*filtering_state_2[t]
    }else{
      sampling_state_1 = (1-p)*filtering_state_1[t]
      sampling_state_2 = p*filtering_state_2[t]
    }
    mod_x[t] = maximizer(sampling_state_1,sampling_state_2)
  }
  return(mod_x)
}
mod_x = Modal_config(p)

# plotting X and modal configuration of X
plot(mod_x, xlab = "t",ylab = "mod x",main= "modal configuration of x for p=0.9")
```



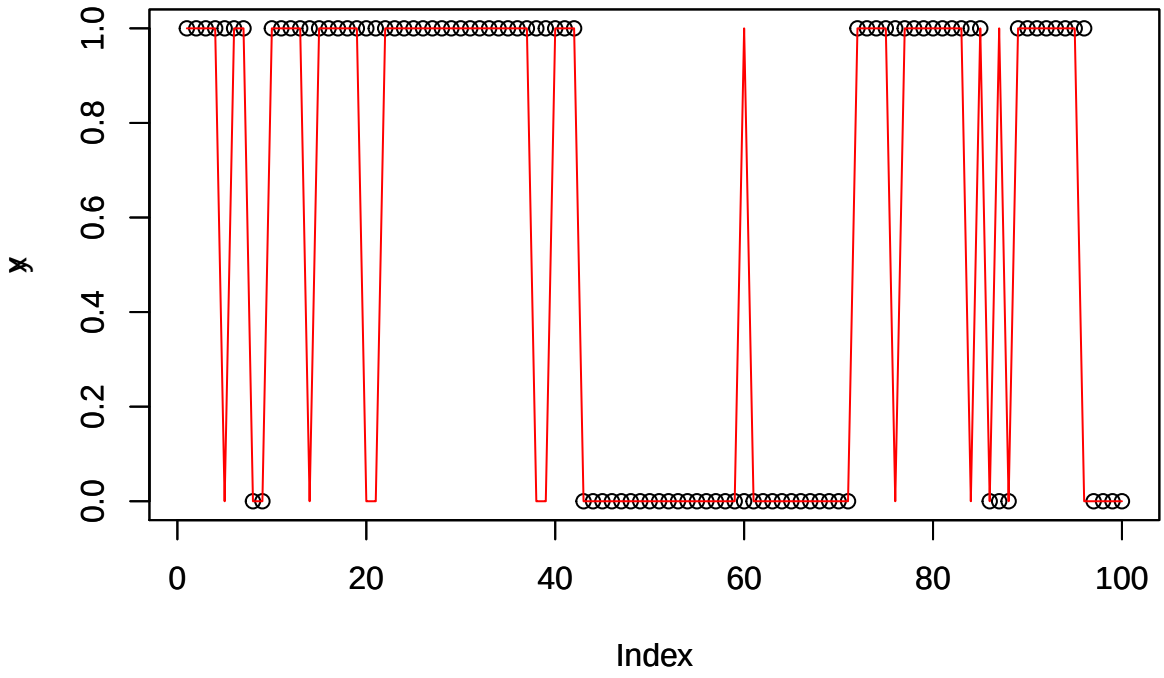
Calculation when Y follows the Discrete distribution.

Here the conditional distribution of $y_t|x_t$ is a discrete distribution as follows.

$$y_t|x_t = \begin{cases} x_t & \text{with probability } q \\ 1 - x_t & \text{with probability } 1 - q \end{cases}$$

Let us do the same process as above under this second distribution also. code:

```
# second distribution of Y
q = 0.9
y = rep(NA,T)
gen_y <- function(q){
  for(i in 1:T){
    if(runif(1) < q){
      y[i] = x[i]
    }else{
      y[i] = 1-x[i]
    }
  }
  return(y)
}
y = gen_y(q)
plot(x)
par(new = T)
plot(y,type="l",col="red")
```




```r
dist_func_bernoulli <- function(y,x){
  if ( y == x){
    return(q)
  }else{
    return(1-q)
  }
}

# we have defined state 1 when x[t] = 1 and state 2 otherwise

filtering_state_1 = rep(NA,T)
filtering_state_2 = rep(NA,T)
prediction_state_1 = rep(NA,T)
prediction_state_2 = rep(NA,T)
Z = rep(NA,T)

prop_filtering_function <- function(prediction,dist_func){
  return(prediction*dist_func)
}

prediction_function <- function(p,filtering1,filtering2,state){
  if (state == 1){
    return(p*filtering1 + (1-p)*filtering2)
  }else{
    return((1-p)*filtering1 + p*filtering2)
  }
```

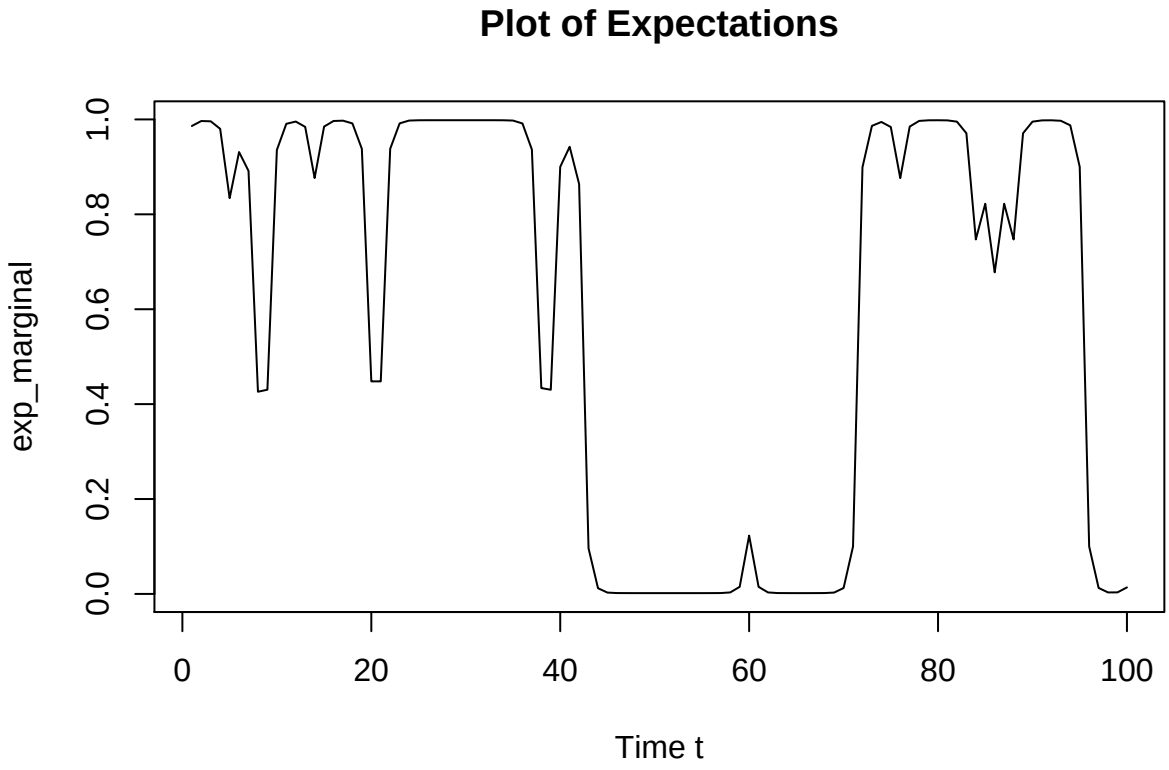
```
}

filtering_and_prediction <- function(p){
  for (t in 1:T){
    if (t == 1){
      prediction_state_1[t] = 0.5
      prediction_state_2[t] = 0.5
      prop_filtering_state_1 = prop_filtering_function(prediction_state_1[t],
                                                       dist_func_bernoulli(y[t],1))
      prop_filtering_state_2 = prop_filtering_function(prediction_state_2[t],
                                                       dist_func_bernoulli(y[t],0))
      norm_constant = prop_filtering_state_1 + prop_filtering_state_2
      filtering_state_1[t] = prop_filtering_state_1/norm_constant
      filtering_state_2[t] = prop_filtering_state_2/norm_constant
      Z[t] = norm_constant
    }else{
      prediction_state_1[t] = prediction_function(p,filtering_state_1[t-1],
                                                  filtering_state_2[t-1],1)
      prediction_state_2[t] = prediction_function(p,filtering_state_1[t-1],
                                                  filtering_state_2[t-1],0)
      prop_filtering_state_1 = prop_filtering_function(prediction_state_1[t],
                                                       dist_func_bernoulli(y[t],1))
      prop_filtering_state_2 = prop_filtering_function(prediction_state_2[t],
                                                       dist_func_bernoulli(y[t],0))
      norm_constant = prop_filtering_state_1 + prop_filtering_state_2
      filtering_state_1[t] = prop_filtering_state_1/norm_constant
      filtering_state_2[t] = prop_filtering_state_2/norm_constant
      Z[t] = norm_constant
    }
  }
  return(matrix(c(filtering_state_1,filtering_state_2,prediction_state_1,
                  prediction_state_2,Z),nrow = 5, byrow = TRUE))
}

# Calculation of filtering and prediction for p = 0.9 as stated earlier
k = filtering_and_prediction(p)
filtering_state_1 = k[1,]
filtering_state_2 = k[2,]
prediction_state_1 = k[3,]
prediction_state_2 = k[4,]

l = smoothing(p)
smoothing_state_1 = l[1,]
smoothing_state_2 = l[2,]
```

1. For pre-defined values of θ we then can find the marginal mean and standard deviation, which is given by

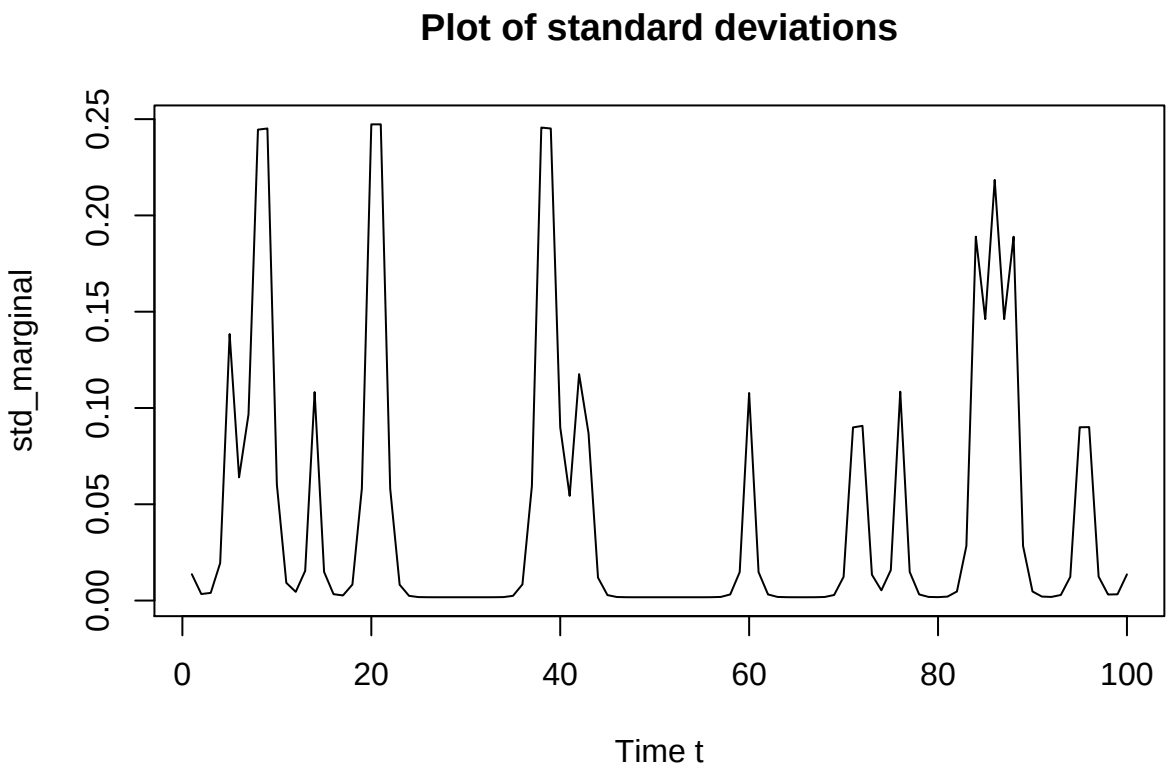
$$E(X_t|Y_{1:T}) = \ P(X_t = 1|Y_{1:T}) = \text{Probability of smoothing at time point t}$$
$$V(X_t|Y_{1:T}) = \ P(X_t = 1|Y_{1:T})(1 - \ P(X_t = 1|Y_{1:T}))$$

Code : Here we are plotting the expectation and standard deviation over the time points.

```
exp_marginal = smoothing_state_1
std_marginal = smoothing_state_1*smoothing_state_2
plot(exp_marginal,type = "l",xlab = "Time t")
title(main = "Plot of Expectations" )
```




```
plot(std_marginal,type="l",xlab = "Time t")
title(main = "Plot of standard deviations")
```

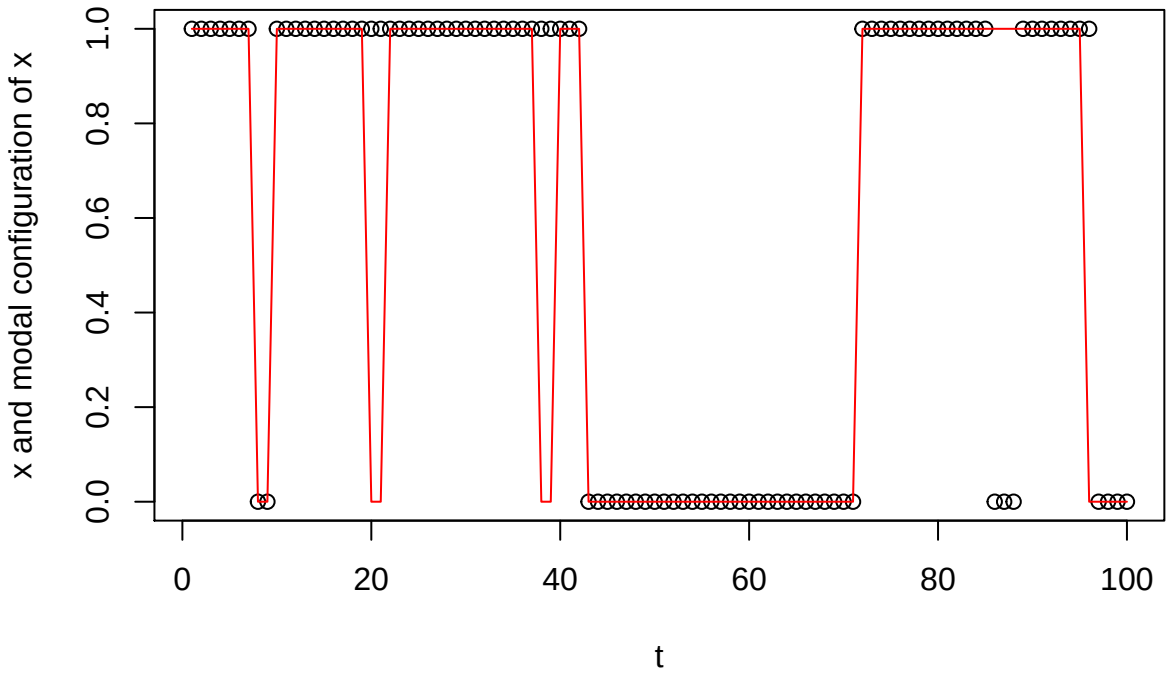


Now to find the modal configuration of X, we will have to do sampling over $\pi(X_{1:T}|Y_{1:T})$ and find the mode from there.

Let us look at the following code:

```
maximizer <- function(a,b){
  if (a > b){
    return(1)
  }else{
    return(0)
  }
}
mod_x = rep(NA,T)
Modal_config <- function(p){
  mod_x[T] = maximizer(filtering_state_1[T],filtering_state_2[T])

  for (t in (T-1):1){
    if (mod_x[t+1] == 1){
      sampling_state_1 = p*filtering_state_1[t]
      sampling_state_2 = (1-p)*filtering_state_2[t]
    }else{
      sampling_state_1 = (1-p)*filtering_state_1[t]
      sampling_state_2 = p*filtering_state_2[t]
    }
    mod_x[t] = maximizer(sampling_state_1,sampling_state_2)
```

```
  }
  return(mod_x)
}
mod_x = Modal_config(p)

# plotting X and modal configuration of X
plot(x, xlab = "t",ylab = "x and modal configuration of x")
lines(mod_x, col = "red")
```



Hence from the above plot it is quite evident that the modal configuration of X matches with X very well.

2. Here we have to find posterior mode and distribution of p.
Let us first assume that the prior distribution of p is Uniform(0,1) i,e; p $\sim U(0,1)$
Now we have to find

$$\begin{aligned} p^* &= \ argmax\ \pi(p|Y_{1:T}) \\ &= \ argmax\ \pi(p)\pi(Y_{1:T}|p) \end{aligned}$$

$$Here\ \pi(Y_{1:T}|p)\ is\ proportional\ to \prod_{i=1}^{T} Z_i$$

$$and\ \pi(p) \sim U(0,1)$$

$$Hence\ p^* = argmax\ \prod_{i=1}^{T} Z_i$$

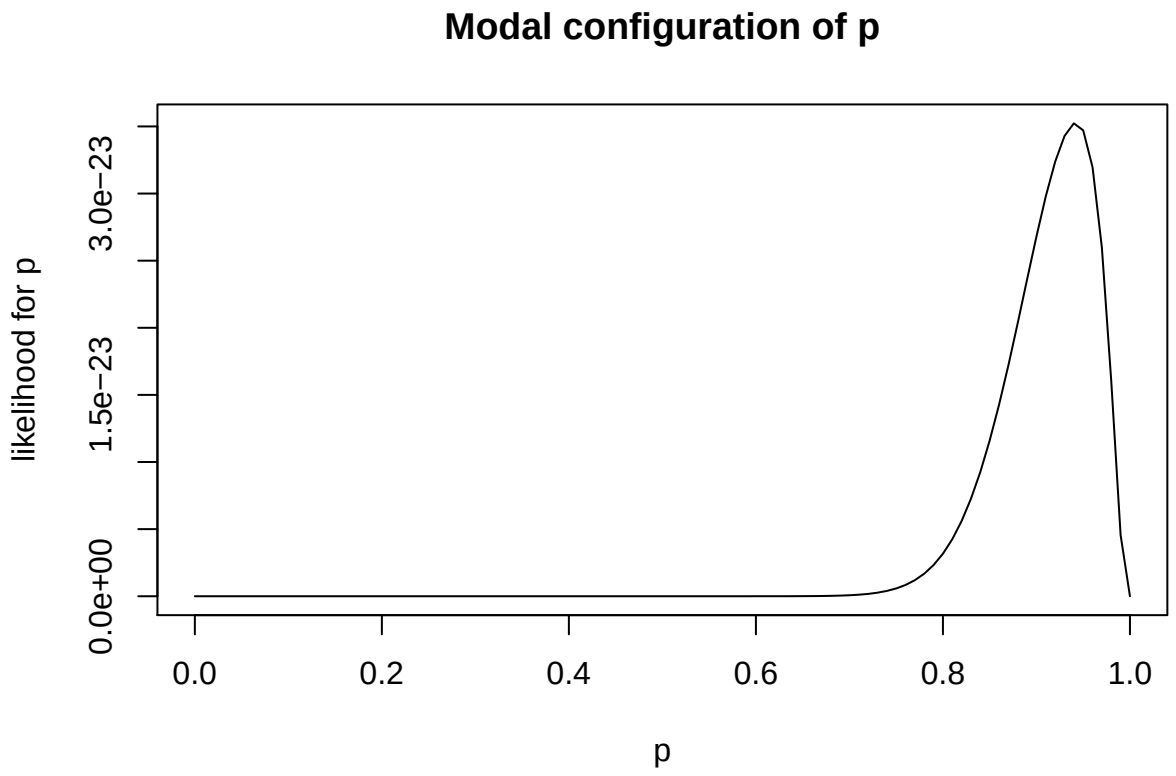
So, if we plot this function with respect to p we will find the section where p maximizes.

Also by normalizing the product of Z_i's we can find the posterior distribution of p.

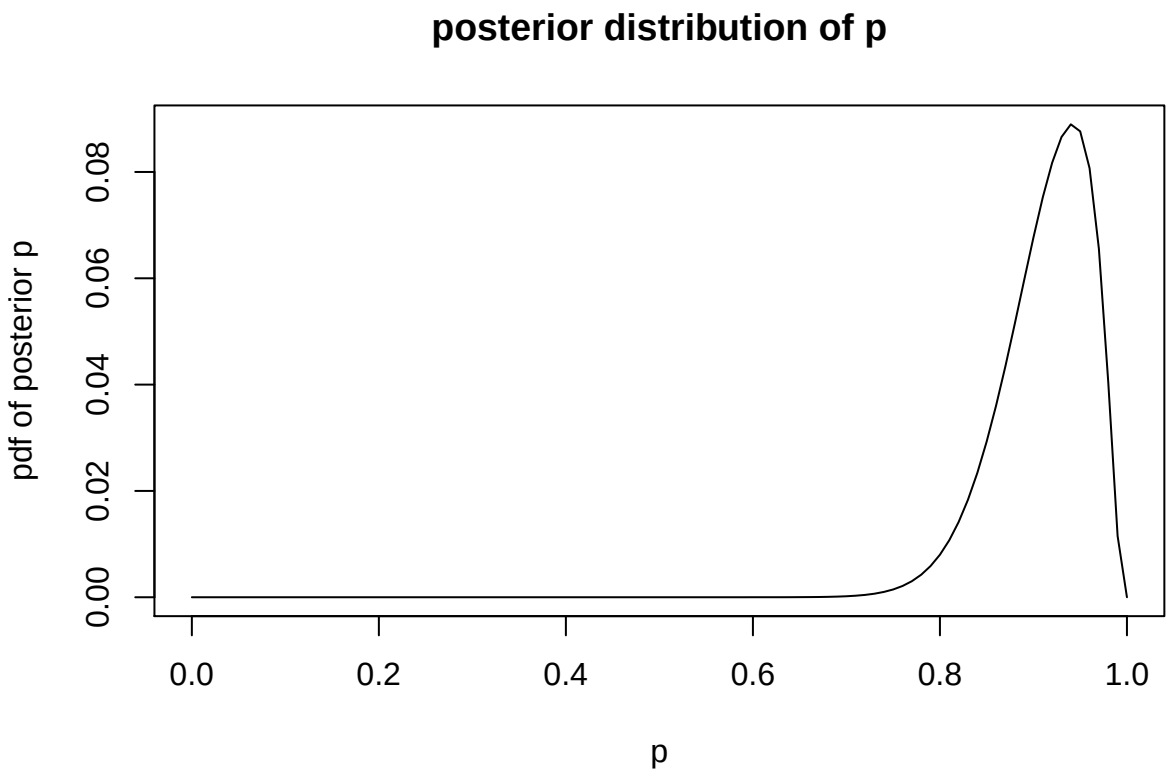
In the following code I have plotted for both the cases.

code:

```
# modal value of p

sequence = seq(from = 0, to = 1 , by = 0.01)
likelihood = rep(NA,length(sequence))
for (i in 1:length(sequence)){
  k = filtering_and_prediction(sequence[i])
  Z = k[5,]
  likelihood[i] = prod(Z)
}
plot(sequence,likelihood, type = "l", xlab = "p", ylab = "likelihood for p")
title(main = "Modal configuration of p")
```




```
# posterior distribution of P
normalizing_p = sum(likelihood)

posterior_p = likelihood/normalizing_p
plot(sequence,posterior_p, type = "l", xlab = "p", ylab = "pdf of posterior p" )
title(main = "posterior distribution of p")
```



So we see the modal value of p matches with our initially defined value of p.

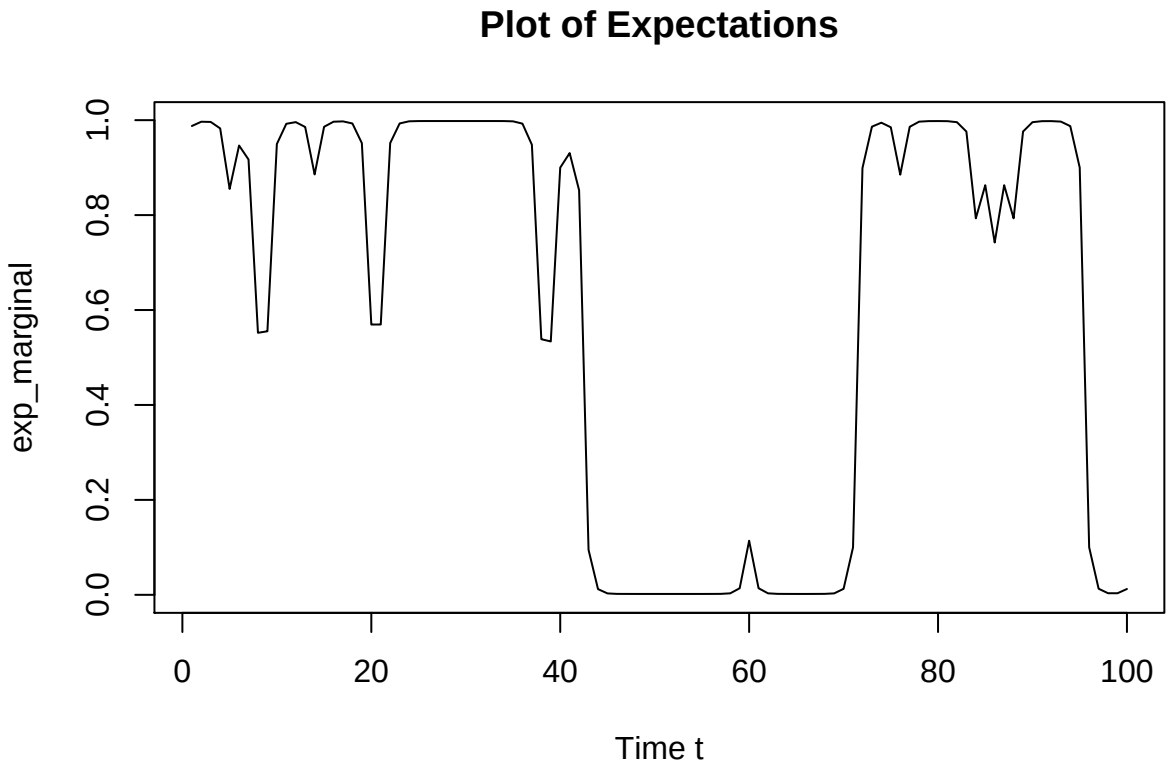
3. Here we have to do the same as 1. but after taking the uncertainty of p into consideration. Let us look at the following.

$$Here\ E(X_t|Y_{1:T}, p) = \ smoothing\ state\ 1\ probability\ conditioned\ on\ p.$$
$$E(X_t|Y_{1:T}, p) \approx \sum_p E(X_t|Y_{1:T}, p_i)\ \pi(p_i|Y_{1:T})$$

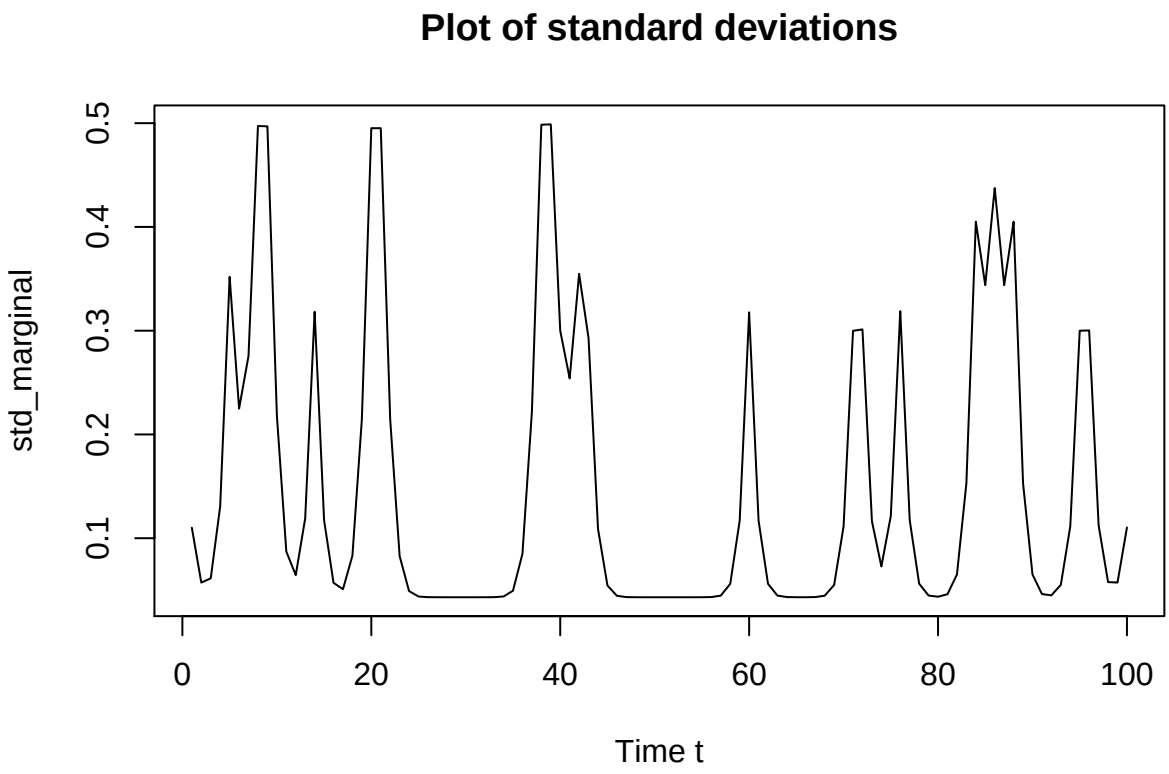
Writing this in code we get.

```
# marginal expectation conditioned on p
exp_marginal = rep(NA,T)
for (t in 1:T){
  exp_part = rep(NA,length(sequence))
  for (i in 1:length(sequence)){
    k = filtering_and_prediction(sequence[i])
    filtering_state_1 = k[1,]
    filtering_state_2 = k[2,]
    prediction_state_1 = k[3,]
    prediction_state_2 = k[4,]
    Z = k[5,]
    l = smoothing(sequence[i])
    smoothing_state_1 = l[1,]
    exp_part[i] = smoothing_state_1[t]*(prod(Z)/normalizing_p)
  }
```

```
  exp_marginal[t] = sum(exp_part)
}

exp_marginal_2 = exp_marginal

var_marginal = exp_marginal_2 - exp_marginal^2
std_marginal = sqrt(var_marginal)

plot(exp_marginal,type = "l",xlab = "Time t")
title(main = "Plot of Expectations" )
```




```
plot(std_marginal,type="l",xlab = "Time t")
title(main = "Plot of standard deviations")
```



Here exp_marginal is the marginal expectation and var_marginal is the variance of the marginal both conditioned on p.

4. Here we have to find the modal configuration of (x,p) jointly.
Let us write the problem in the following way.

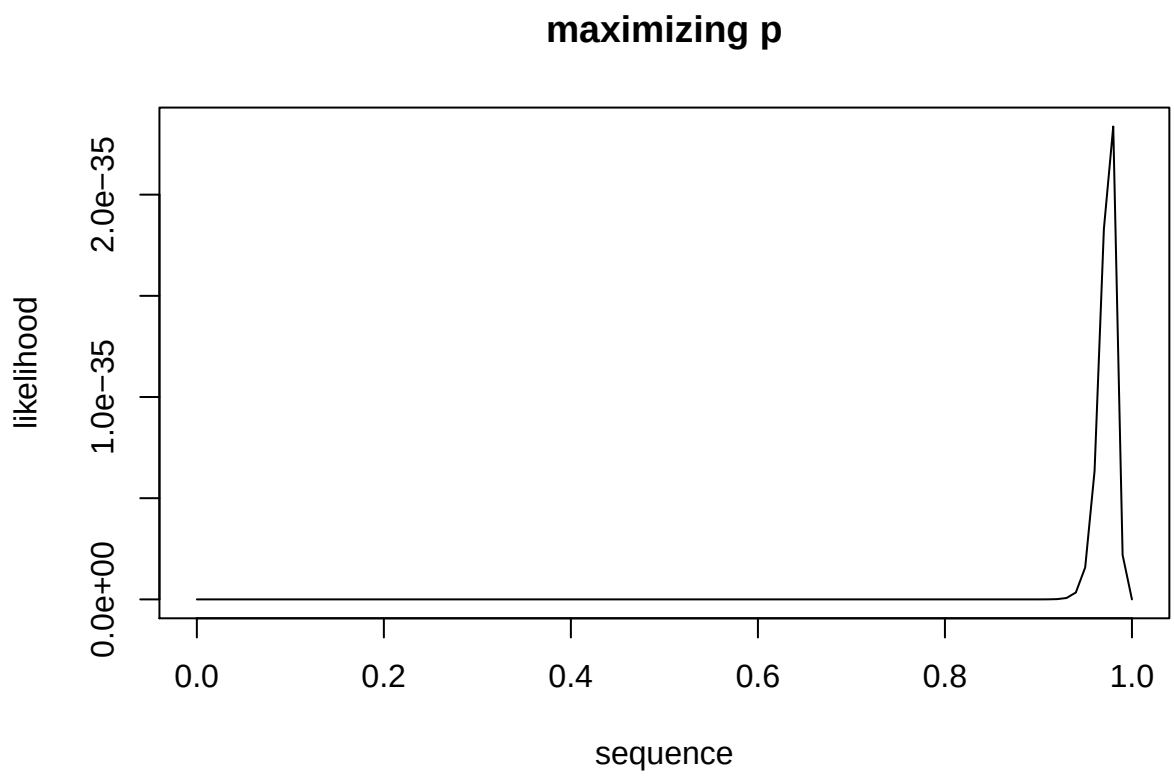
$$\begin{aligned} Here\ (p^*, X^*) = &\ argmax\ \pi(p, X_{1:T}|Y_{1:T}) \\ \propto &\ argmax\ \pi(X_{1:T}|Y_{1:T}, p)\ \pi(p|Y_{1:T}) \end{aligned}$$

So here we will first maximize X as a function of p then put that value above and maximize over p.

code:

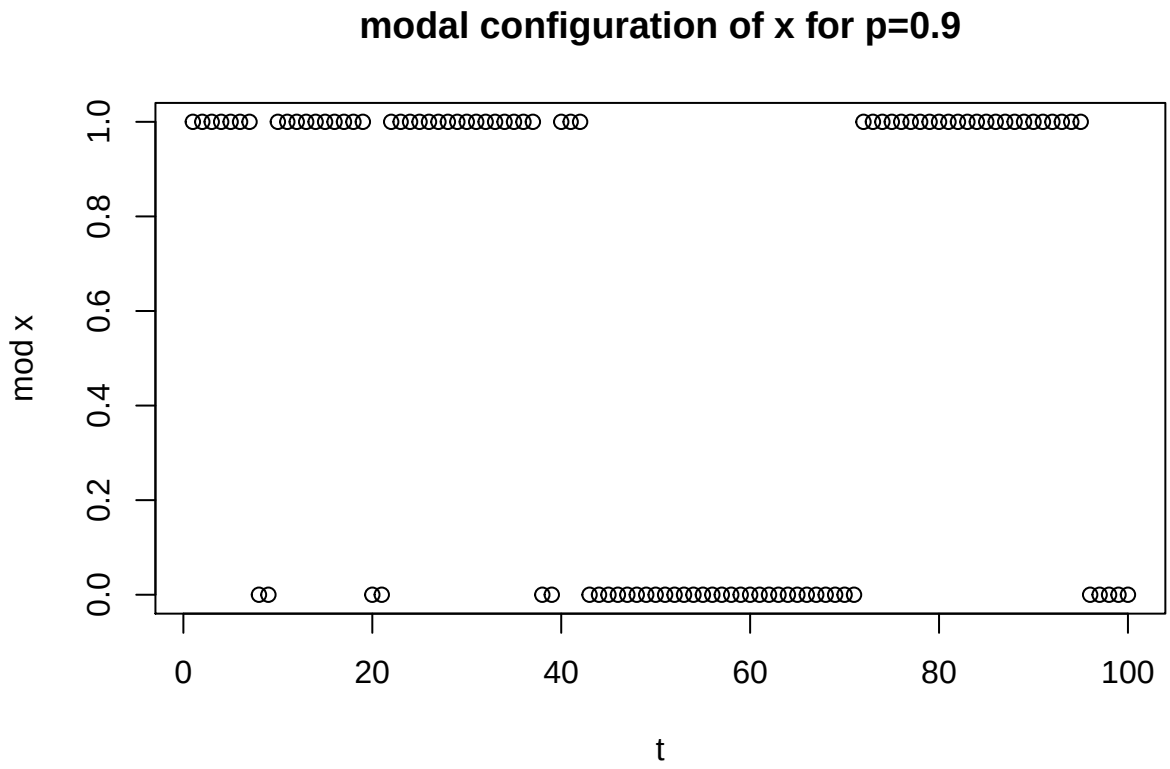
```
# Mod (p,x)

# Modal configuration of X conditioned on p
mod_x = rep(NA,T)
mod_x_dist = rep(NA,T)
filtering_state_1= rep(NA,T)
filtering_state_2= rep(NA,T)
Modal_config <- function(p,filtering_state_1,filtering_state_2){
  mod_x_dist[T] = max(filtering_state_1[T],filtering_state_2[T])
  mod_x[T] = maximizer(filtering_state_1[T],filtering_state_2[T])
  for (t in (T-1):1){
    if (mod_x[t+1] == 1){
      sampling_state_1 = p*filtering_state_1[t]
```

```
      sampling_state_2 = (1-p)*filtering_state_2[t]
    }else{
      sampling_state_1 = (1-p)*filtering_state_1[t]
      sampling_state_2 = p*filtering_state_2[t]
    }
    mod_x_dist[t] = max(sampling_state_1,sampling_state_2)
    mod_x[t] = maximizer(sampling_state_1,sampling_state_2)
  }
  return(prod(mod_x_dist))
}
sequence = seq(from = 0, to = 1 , by = 0.01)
likelihood = rep(NA,length(sequence))
for (i in 1:length(sequence)){
  k = filtering_and_prediction(sequence[i])
  filtering_state_1 = k[1,]
  filtering_state_2 = k[2,]
  Z = k[5,]
  mod_x_dist = Modal_config(sequence[i],filtering_state_1,filtering_state_2)
  likelihood[i] = prod(Z) * mod_x_dist
}

plot(sequence,likelihood,type="l")
title(main = "maximizing p")
```



Hence, From the plot it is evident that the mode for p is nearing 0.9 and putting this value in the modal configuration we will get the modal configuration of X. Code:

```r
p = 0.9
mod_x = rep(NA,T)
k = filtering_and_prediction(p)
filtering_state_1 = k[1,]
filtering_state_2 = k[2,]
Modal_config <- function(p){
  mod_x[T] = maximizer(filtering_state_1[T],filtering_state_2[T])

  for (t in (T-1):1){
    if (mod_x[t+1] == 1){
      sampling_state_1 = p*filtering_state_1[t]
      sampling_state_2 = (1-p)*filtering_state_2[t]
    }else{
      sampling_state_1 = (1-p)*filtering_state_1[t]
      sampling_state_2 = p*filtering_state_2[t]
    }
    mod_x[t] = maximizer(sampling_state_1,sampling_state_2)
  }
  return(mod_x)
}
mod_x = Modal_config(p)

# plotting X and modal configuration of X
plot(mod_x, xlab = "t",ylab = "mod x",main= "modal configuration of x for p=0.9")
```



The above plot shows the modal configuration of X for p=0.9.

Question 3.

I will be using the same code written in problem 2. to solve this problem.

I first have read the **sequence.txt** file then created the discrete probability function for $Y_t|X_t$
Code:

```
y = strsplit(readLines("C:/Users/Monalisha/Desktop/sequence.txt"),"")[[1]]
```

```
## Warning in readLines("C:/Users/Monalisha/Desktop/sequence.txt"): incomplete
## final line found on 'C:/Users/Monalisha/Desktop/sequence.txt'
```

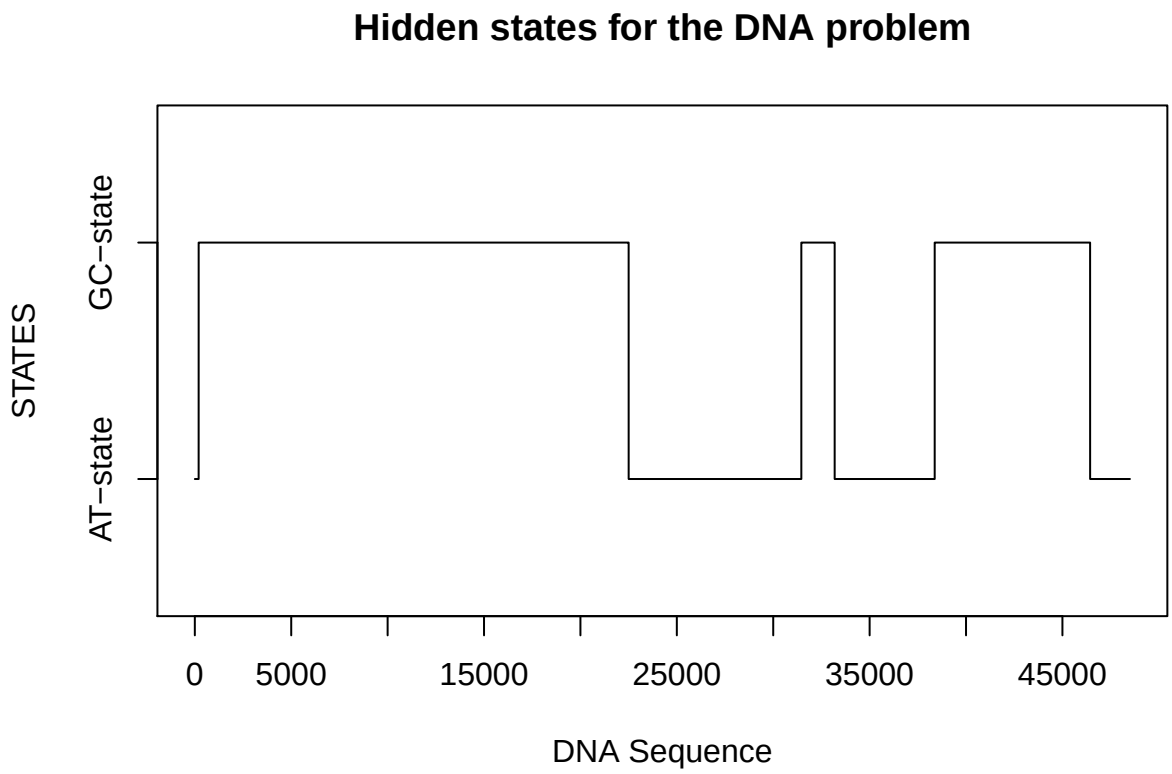
```
T = length(y)
p = 0.9998



y_cond_distbn <- function(Y,x){
  emission_state_2 = c(0.27,0.2084,0.198,0.3236)
  emission_state_1 = c(0.2462,0.2476,0.2985,0.2077)
  val_y = c("A","C","G","T")
  if(x == 0){
    for(i in 1:4){
      if(Y == val_y[i]){
        return(emission_state_2[i])
      }
    }

  }else{
    for(i in 1:4){
      if(Y == val_y[i]){
        return(emission_state_1[i])
      }
    }
  }
}
```

Here our final aim is to find the most likely state at each time point. In order to do so I have generated the smoothing distribution for the y values and found the mode of this distribution at each time point. In order to authenticate the way I am finding the mode I have done simulation on the same set of emission and transition matrics and found my modal distribution to be fairly correct. code for the modal distribution is given below :

```
filtering_state_1 = rep(NA,T)
filtering_state_2 = rep(NA,T)
prediction_state_1 = rep(NA,T)
prediction_state_2 = rep(NA,T)
Z = rep(NA,T)
prop_filtering_function <- function(prediction,dist_func){
  return(prediction*dist_func)
}
```

```
prediction_function <- function(p,filtering1,filtering2,state){
  if (state == 1){
    return(p*filtering1 + (1-p)*filtering2)
  }else{
    return((1-p)*filtering1 + p*filtering2)
  }
}

maximizer <- function(a,b){
  if (a > b){
    return(1)
  }else{
    return(0)
  }
}

filtering_and_prediction <- function(p){
  for (t in 1:T){
    if (t == 1){
      prediction_state_1[t] = 0.5
      prediction_state_2[t] = 0.5
      prop_filtering_state_1 = prop_filtering_function(prediction_state_1[t],
                                                       y_cond_distbn(y[t],1))
      prop_filtering_state_2 = prop_filtering_function(prediction_state_2[t],
                                                       y_cond_distbn(y[t],0))
      norm_constant = prop_filtering_state_1 + prop_filtering_state_2
      filtering_state_1[t] = prop_filtering_state_1/norm_constant
      filtering_state_2[t] = prop_filtering_state_2/norm_constant
      Z[t] = norm_constant
    }else{
      prediction_state_1[t] = prediction_function(p,filtering_state_1[t-1],
                                                  filtering_state_2[t-1],1)
      prediction_state_2[t] = prediction_function(p,filtering_state_1[t-1],
                                                  filtering_state_2[t-1],0)
      prop_filtering_state_1 = prop_filtering_function(prediction_state_1[t],
                                                       y_cond_distbn(y[t],1))
      prop_filtering_state_2 = prop_filtering_function(prediction_state_2[t],
                                                       y_cond_distbn(y[t],0))
      norm_constant = prop_filtering_state_1 + prop_filtering_state_2
      filtering_state_1[t] = prop_filtering_state_1/norm_constant
      filtering_state_2[t] = prop_filtering_state_2/norm_constant
      Z[t] = norm_constant
    }
  }
  return(matrix(c(filtering_state_1,filtering_state_2,
                  prediction_state_1,prediction_state_2,Z),nrow = 5, byrow = TRUE))
}

k = filtering_and_prediction(p)
filtering_state_1 = k[1,]
filtering_state_2 = k[2,]
prediction_state_1 = k[3,]
prediction_state_2 = k[4,]
```

```
Z = k[5,]

smoothing_function <- function(p,filtering,smoothing1,prediction1,
                               smoothing2,prediction2,state){
  if (state == 1){
    return((p*filtering*smoothing1/prediction1) +
             ((1-p)*filtering*smoothing2/prediction2))
  }else{
    return(((1-p)*filtering*smoothing1/prediction1) +
             (p*filtering*smoothing2/prediction2))
  }
}
smoothing <- function(p){
  for (t in T:1){
    if (t == T){
      smoothing_state_1[t] = filtering_state_1[t]
      smoothing_state_2[t] = filtering_state_2[t]
    }else{
      smoothing_state_1[t] = smoothing_function(p,filtering_state_1[t],
                                                smoothing_state_1[t+1],
                                                prediction_state_1[t+1],
                                                smoothing_state_2[t+1],
                                                prediction_state_2[t+1],1)
      smoothing_state_2[t] = smoothing_function(p,filtering_state_2[t],
                                                smoothing_state_1[t+1],
                                                prediction_state_1[t+1],
                                                smoothing_state_2[t+1],
                                                prediction_state_2[t+1],0)
    }
  }
  return(return(matrix(c(smoothing_state_1,smoothing_state_2),nrow = 2, byrow = TRUE)))
}

l = smoothing(p)
smoothing_state_1 = l[1,]
smoothing_state_2 = l[2,]

for (i in 1:T){
  mod_x[i] = maximizer(smoothing_state_1[i],smoothing_state_2[i])
}

# Plot the sequence

plot(mod_x, main = "Hidden states for the DNA problem",
     xlab = "DNA Sequence", ylab = "STATES", xaxt = "n", yaxt = "n", ylim = c(-0.5,
                                                                              1.5), type = "l")
axis(side = 1, at = seq(from = 0, to = T, by = 5000))
axis(side = 2, at = c(0, 1), labels = c("AT-state", "GC-state"))
```



Now we have to find the blocks of each states.

Code:

```
change_index = c(1)
for (i in 2:length(mod_x)){
  if (mod_x[i] != mod_x[i-1]){
    change_index = c(change_index,i)
  }
}

block_DNAsequence = list()
for (i in 1:length(change_index)){
  if (i != length(change_index)){
    block_DNAsequence[[i]] = y[change_index[i]:change_index[i+1]]

  }else{
    block_DNAsequence[[i]] = y[change_index[i]:length(change_index)]
  }
}

state_names = c("AT-state", "GC-state","AT-state", "GC-state",
                "AT-state", "GC-state","AT-state")

for (i in 1:7){
  print(paste0("block number ",i," state name ",state_names[i],
```

```
              " contains ",length(block_DNAsequence[[i]])," elements."))
}
```

```
## [1] "block number 1 state name AT-state contains 199 elements."
## [1] "block number 2 state name GC-state contains 22304 elements."
## [1] "block number 3 state name AT-state contains 8957 elements."
## [1] "block number 4 state name GC-state contains 1730 elements."
## [1] "block number 5 state name AT-state contains 5189 elements."
## [1] "block number 6 state name GC-state contains 8062 elements."
## [1] "block number 7 state name AT-state contains 46430 elements."
```

```
print(paste0("total number of elements in the sequence is ",
             length(y)))
```

```
## [1] "total number of elements in the sequence is 48502"
```

Hence the above is the segregation w.r.t the two states.

Question 4.

Here in this problem we first check the simplest case i,e; $L(z,x) = \sum 1_{x_t \neq z_t}$.
The posterior expectation of this expression would be as following

$$E_{x|y}(L(z,x)) = \sum E_{x|y}(1_{x_t \neq z_t}) = \sum (1 - P(x_t = z_t|y_{1:T}))$$

Hence the *argmin* of the above expression will basically be the maximizer of the probability $P(x_t = z_t|y_{1:T})$, which is the smoothing distribution at time t i,e;

$$z^* = argmax\ \pi(x_t|y_{1:T})$$

The above is called the Optimal Bayes Estimator(OBE).
Now lets dive into the more complicated case case when $L(z,x)$ can be written as

$$L(z,x) = \sum 1_{x_t \neq z_t} + \lambda \sum 1_{[x_t \neq z_t]} 1_{[x_{t+1} \neq z_{t+1}]}$$

Then we can also rewrite the above expression in the following way.

$$L(z,x) = \sum (x_t - z_t)^2 + \lambda \sum (x_t - z_t)^2 (x_{t+1} - z_{t+1})^2$$

Now if we take the posterior expectation over the above expression we will get th following.

$$E_{x|y} L(z,x) = E_{x|y} \sum (x_t - z_t)^2 + \lambda\ E_{x|y} \sum (x_t - z_t)^2 (x_{t+1} - z_{t+1})^2$$

Now the first term can be written as

$$E_{x|y} \sum (x_t - z_t)^2 = \sum_{t=1}^{T} E_{x|y}(x_t + z_t - 2 * x_t * z_t)\ note\ that\ x_t^2 = x_t\ and\ z_t^2 = z_t$$

$$Hence\ E_{x|y} \sum (x_t - z_t)^2 = \sum_{t=1}^{T} (Z_t - 2 * Z_t * P(x_t = 1|y_{1:T}))\ +\ Constant$$

And that of the second term would be

$$\lambda\ E_{x|y} \sum (x_t - z_t)^2 (x_{t+1} - z_{t+1})^2$$

$$= \lambda \sum_{t=1}^{T-1} \quad z_t z_{t+1} - 2 z_t z_{t+1} \mathbb{E}_{x|y}(x_{t+1}) + z_t \mathbb{E}_{x|y}(x_{t+1})$$

$$- 2 z_t z_{t+1} \mathbb{E}_{x|y}(x_t) + 4 z_t z_{t+1} \mathbb{E}_{x|y}(x_t x_{t+1}) - 2 z_t \mathbb{E}_{x|y}(x_t x_{t+1})$$

$$+ z_{t+1} \mathbb{E}_{x|y}(x_t) - 2 z_{t+1} \mathbb{E}_{x|y}(x_t x_{t+1}) + \mathbb{E}_{x|y}(x_t x_{t+1}),$$

Where $E_{x|y}(x_t) = P(x_t = 1|y_{1:T})$, $E_{x|y}(x_{t+1}) = P(x_{t+1} = 1|y_{1:T})$ and $E_{x|y}(x_t x_{t+1}) = P(x_t = 1, x_{t+1} = 1|y_{1:T})$

It can be easily seen that the expressions for all the above expressions has already been coded in question.2 except for the 3rd expectation Now $E_{x|y}(x_t x_{t+1})$ is basically the inner function which we earlier summing out over x_{t+1}.

Let us now code the above process.
Here for T=500, the hidden state has been simulated with transition probability p = 0.8 and then based on that a discrete y distribution has been generated same as in question 2. I have created all the stages same as that of question 2 along with that the vector named *joint_smoothing_state_1* has been created which is additionally storing the values of $P(x_t = 1, x_{t+1} = 1|y_{1:T})$. here q for the discrete distribution of y given x has been taken 0.95. code:

```
# question 4
p <- 0.80
T = 500
set.seed(1234567)
x = rep(NA,T)
y = rep(NA,T)

x[1] = as.numeric(runif(1) > 0.5)
for (i in 2:T){
  if (runif(1) < p){
    x[i] = x[i-1]
  }else{
    x[i] = 1-x[i-1]
  }
}

q = 0.95
y = rep(NA,T)
gen_y <- function(q){
  for(i in 1:T){
    if(runif(1) < q){
      y[i] = x[i]
    }else{
      y[i] = 1-x[i]
    }
  }
  return(y)
}

y = gen_y(q)


dist_func_bernoulli <- function(y,x){
  if ( y == x){
    return(q)
  }else{
    return(1-q)
  }
}

filtering_state_1 = rep(NA,T)
filtering_state_2 = rep(NA,T)
prediction_state_1 = rep(NA,T)
prediction_state_2 = rep(NA,T)
Z = rep(NA,T)

prop_filtering_function <- function(prediction,dist_func){
```

```
  return(prediction*dist_func)
}

prediction_function <- function(p,filtering1,filtering2,state){
  if (state == 1){
    return(p*filtering1 + (1-p)*filtering2)
  }else{
    return((1-p)*filtering1 + p*filtering2)
  }
}


filtering_and_prediction <- function(p){
  for (t in 1:T){
    if (t == 1){
      prediction_state_1[t] = 0.5
      prediction_state_2[t] = 0.5
      prop_filtering_state_1 = prop_filtering_function(prediction_state_1[t],
                                                       dist_func_bernoulli(y[t],1))
      prop_filtering_state_2 = prop_filtering_function(prediction_state_2[t],
                                                       dist_func_bernoulli(y[t],0))
      norm_constant = prop_filtering_state_1 + prop_filtering_state_2
      filtering_state_1[t] = prop_filtering_state_1/norm_constant
      filtering_state_2[t] = prop_filtering_state_2/norm_constant
      Z[t] = norm_constant
    }else{
      prediction_state_1[t] = prediction_function(p,filtering_state_1[t-1],
                                                  filtering_state_2[t-1],1)
      prediction_state_2[t] = prediction_function(p,filtering_state_1[t-1],
                                                  filtering_state_2[t-1],0)
      prop_filtering_state_1 = prop_filtering_function(prediction_state_1[t],
                                                       dist_func_bernoulli(y[t],1))
      prop_filtering_state_2 = prop_filtering_function(prediction_state_2[t],
                                                       dist_func_bernoulli(y[t],0))
      norm_constant = prop_filtering_state_1 + prop_filtering_state_2
      filtering_state_1[t] = prop_filtering_state_1/norm_constant
      filtering_state_2[t] = prop_filtering_state_2/norm_constant
      Z[t] = norm_constant
    }
  }
  return(matrix(c(filtering_state_1,filtering_state_2,prediction_state_1,
                  prediction_state_2,Z),nrow = 5, byrow = TRUE))
}

k = filtering_and_prediction(p)
filtering_state_1 = k[1,]
filtering_state_2 = k[2,]
prediction_state_1 = k[3,]
prediction_state_2 = k[4,]

# smoothing

smoothing_state_1 = rep(NA,T)
```

```
smoothing_state_2 = rep(NA,T)
joint_smoothing = rep(NA,T)
smoothing_joint_function <- function(p,filtering,smoothing1,prediction1){
  return(p*filtering*smoothing1/prediction1)
}
smoothing_function <- function(p,filtering,smoothing1,prediction1,smoothing2,prediction2,state){
  if (state == 1){
    return((p*filtering*smoothing1/prediction1) +
             ((1-p)*filtering*smoothing2/prediction2))
  }else{
    return(((1-p)*filtering*smoothing1/prediction1) +
             (p*filtering*smoothing2/prediction2))
  }
}

smoothing <- function(p){
  for (t in T:1){
    if (t == T){
      smoothing_state_1[t] = filtering_state_1[t]
      smoothing_state_2[t] = filtering_state_2[t]

    }else{
      smoothing_state_1[t] = smoothing_function(p,filtering_state_1[t],
                                                smoothing_state_1[t+1],
                                                prediction_state_1[t+1],
                                                smoothing_state_2[t+1],
                                                prediction_state_2[t+1],1)
      smoothing_state_2[t] = smoothing_function(p,filtering_state_2[t],
                                                smoothing_state_1[t+1],
                                                prediction_state_1[t+1],
                                                smoothing_state_2[t+1],
                                                prediction_state_2[t+1],0)
      joint_smoothing[t] = smoothing_joint_function(p,filtering_state_1[t],
                                                    smoothing_state_1[t+1],
                                                    prediction_state_1[t+1])
    }
  }
  return(return(matrix(c(smoothing_state_1,joint_smoothing),nrow = 2, byrow = TRUE)))
}

l = smoothing(p)
smoothing_state_1 = l[1,]
joint_smoothing_state_1 = l[2,]
```

Now in the next phase I have computed the expected loss at each time point and stored the values in the matrix A.
Now to minimize this loss function I have applied the procedure of question 1. but here instead of finding the argmax I have done argmin.

Code:

```
exp_loss_at_t = function(t, z, lambda) {
# first part

first_part = z[1] - (2 * z[1] * smoothing_state_1[t]) +
  (lambda*((z[1] * smoothing_state_1[t+1]) - (2 * z[1] * joint_smoothing_state_1[t])))
# second part
second_part1 = (z[1] * z[2]) - (2 * z[1] * z[2] * smoothing_state_1[t+1]) -
  (2 *z[1] * z[2] * smoothing_state_1[t]) + (4 * z[1] * z[2] * joint_smoothing_state_1[t]) +
  (z[2] *smoothing_state_1[t]) - (2 * z[2] * joint_smoothing_state_1[t])

second_part = lambda*second_part1

  return(first_part+second_part)
}

A = matrix(NA, nrow = 2 , ncol = T)
compute_loss_matrix = function(lambda){
  for (t in 1:T){
    if (t!= T) {
      z_0 = exp_loss_at_t(t,c(1,2),lambda) + exp_loss_at_t(t,c(1,1),lambda)
      z_1 = exp_loss_at_t(t,c(2,1),lambda) + exp_loss_at_t(t,c(2,2),lambda)
      A[1,t] = z_0
      A[2,t] = z_1

    }else{
      A[1,t] = 1 - (2 * 1 * smoothing_state_1[t])
      A[2,t] = 2 - (2 * 2 * smoothing_state_1[t])
    }
  }
  return(A)
}

A = compute_loss_matrix(lambda = 1)
M <- 2
N <- T
H <- matrix(NA, M, N)
D <- 1
H[, 1] <- A[, 1]

for(j in 2:N) {
  for(i in 1:M) {
    ii <- which.min(H[, j-1] + A[i, j])
    H[i, j] <- H[ii, j-1] + A[i, j]
  }
}

xs <- numeric(N)
xs[N] <- which.min(H[, N])
for(j in (N-1):1) {
  i.low <- max(1, xs[j+1]-D)
  i.high <- min(M, xs[j+1]+D)
  ii <- i.low -1 + which.min(H[i.low:i.high, j])
  xs[j] <- ii
```

```
}
xs = xs -1

g = (x-xs)^2

print(paste("Misclassification = ",sum(g)))
```

```
## [1] "Misclassification =  22"
```

Hence from above we see that there are 22 misclassifications which is fairly good for T=500. Now plotting the predicted vs the actual will give us the following.

```
plot(x, type="l",main = "hidden state in black dots and the prediction in red line",
          lwd = 2,ylab = "actual vs. predicted",xlab = "time")
lines(xs, type="l",col = "red",lwd = 2,
     ylab = NULL,xlab = NULL)
```