ExamHelpFile

October 17, 2018

Lab 1

Question1

The dataset used for this lab is "Aisa" dataset, it isa Small synthetic data set from Lauritzen and Spiegelhalter (1988) about lung diseases (tuberculosis, lung cancer or bronchitis) and visits to Asia.We need to show multiple runs of the hill-climbing algorithm can return non-equivalent Bayesian network (BN) structures.

First loading the dataset

```
library(bnlearn)
library(gRain)
library(MASS)
data(asia)
```

It contains the following variables:

```
names(asia)
```

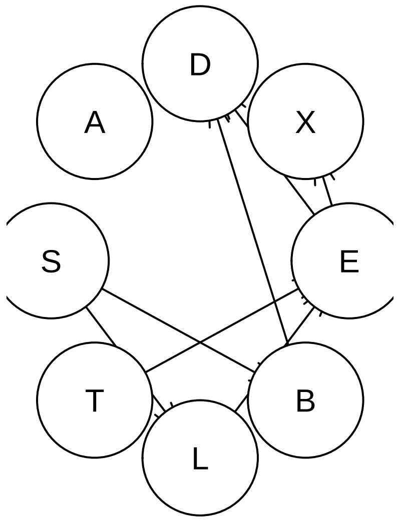
```
## [1] "A" "S" "T" "L" "B" "E" "X" "D"
```

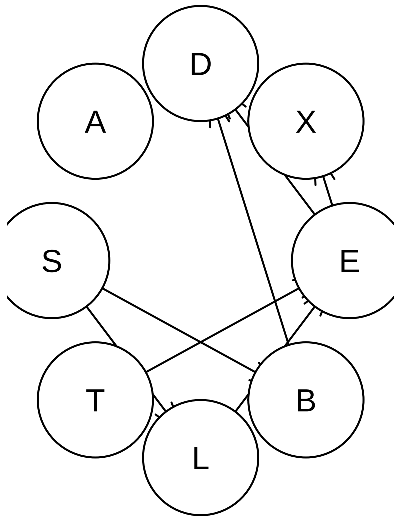
Firstly running the hill climbing algorithm with the different restart values and checking the results. A hill climbing greedy search on the space of the directed graphs which learn the structure of a Bayesian network. The optimized implementation uses score, restart etc.Hill climbing finds optimal solutions for convex problems for other problems it will find only local optima.

```
hcrun1<-hc(asia,restart = 0)
hcrun2<-hc(asia,restart=100)
```

plots of different settings of restarts are displayed below:

```
par(mfrow=c(1,2))
plot(hcrun1)
plot(hcrun2)
```





vstructs retuns v-structures of a Bayesian network according to the value of arc parameters.

```
vstructs(hcrun1)
```

```
##      X   Z   Y
## [1,] "T" "E" "L"
## [2,] "B" "D" "E"
```

```
vstructs(hcrun2)
```

```
##      X   Z   Y
## [1,] "T" "E" "L"
## [2,] "B" "D" "E"
```

The arc set of a directed graph is the set of all arcs (directed edges) of the graph which is returned by arcs() in bnlearn. From the results we can see the different arcs are returned.

```
arcs(hcrun1)
```

```
##      from to
## [1,] "B"  "D"
## [2,] "L"  "E"
## [3,] "E"  "X"
## [4,] "S"  "B"
## [5,] "T"  "E"
## [6,] "E"  "D"
## [7,] "S"  "L"
```

```
arcs(hcrun2)
```

```
##      from to
## [1,] "T"  "E"
## [2,] "B"  "D"
## [3,] "S"  "B"
## [4,] "S"  "L"
## [5,] "L"  "E"
## [6,] "E"  "D"
## [7,] "E"  "X"
```

The results of all.equals() is dispalyed below.The all.equals() checks the 'near equality' and return the difference. Equalence class can be visualized by a graph that has the same skeleton,all the DAG's in the equivalence class have the same directed edge. Where as completed partially directed acyclic graph (CPDAG) equivalence checked with the bidirected edges (sometimes, undirected edges are used instead), hence all.equals(cpdag) always returns TRUE.

```
result1<-NULL
result2<-NULL
for(i in 1:5)
{
hcrun1<-hc(asia,restart = 0)
hcrun2<-hc(asia,restart=100)
result1[i]<-all.equal(hcrun1,hcrun2)
result2[i]<-all.equal(cpdag(hcrun1),cpdag(hcrun2))
}
cat("Results with all.equals:")
```

```
## Results with all.equals:
```

```
result1
```

```
## [1] "TRUE"                "TRUE"                "TRUE"
## [4] "Different arc sets" "Different arc sets"
```

```
cat("Results with all.equals(cpdag):")
```

```
## Results with all.equals(cpdag):
```

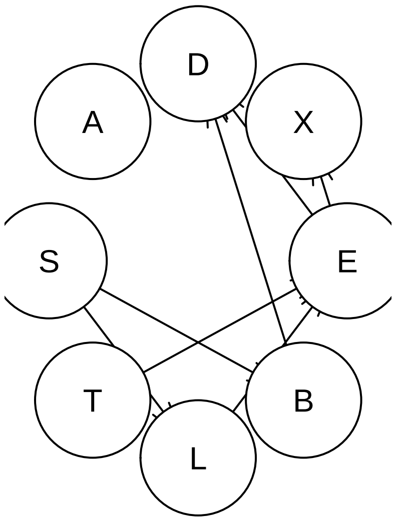
```
result2
```

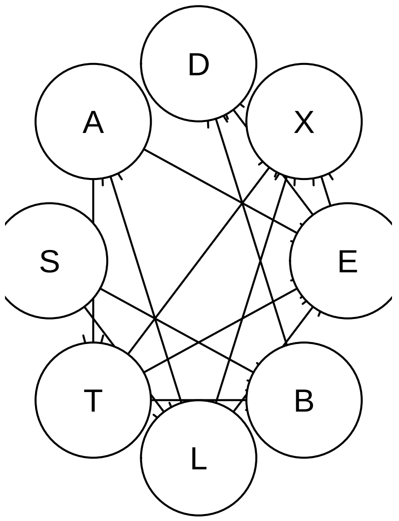
```
## [1] TRUE TRUE TRUE TRUE TRUE
```

case2

In case2 we are changing the settings with the equivalent/imaginary sample size in the Bayesian Dirichlet equivalent uniform (BDeu) score, it determines how much weight is assigned to the prior in terms of the size of an imaginary sample. Results are displayed below with different sample sizes. As the vstructure of the bayesian network is different **cpdag** returns different arc set.

```
hcrun3<-hc(asia,score="bde",iss=1,restart = 0)
hcrun4<-hc(asia,score="bde",iss=10,restart = 0)
par(mfrow=c(1,2))
plot(hcrun3)
plot(hcrun4)
```





```r
arcs(hcrun3)
```

```
##      from to
## [1,] "B"  "D"
## [2,] "L"  "E"
## [3,] "E"  "X"
## [4,] "S"  "B"
## [5,] "T"  "E"
## [6,] "E"  "D"
## [7,] "S"  "L"
```

```r
arcs(hcrun4)
```

```
##       from to
##  [1,] "B"  "D"
##  [2,] "L"  "E"
##  [3,] "E"  "X"
##  [4,] "S"  "B"
##  [5,] "T"  "E"
##  [6,] "E"  "D"
##  [7,] "S"  "L"
##  [8,] "A"  "T"
##  [9,] "A"  "E"
## [10,] "L"  "X"
## [11,] "T"  "X"
## [12,] "T"  "B"
## [13,] "L"  "A"
```

```
vstructs(hcrun3)
```

```
##      X   Z   Y  
## [1,] "T" "E" "L"
## [2,] "B" "D" "E"
```

```
vstructs(hcrun4)
```

```
##      X   Z   Y  
## [1,] "S" "B" "T"
## [2,] "A" "E" "T"
## [3,] "A" "E" "L"
## [4,] "T" "E" "L"
## [5,] "T" "X" "L"
## [6,] "T" "X" "E"
## [7,] "L" "X" "E"
## [8,] "B" "D" "E"
```

```
result3<-NULL
result4<-NULL
g<-NULL
for(i in 1:5)
{
  hcrun3<-hc(asia,score="bde",iss=1,restart = 0)
  hcrun4<-hc(asia,score="bde",iss=10,restart = 0)
  result3[i]<-all.equal(hcrun3,hcrun4)
  result4[i]<-all.equal(cpdag(hcrun3),cpdag(hcrun4))
}
cat("RESULTS of all.equal")
```

```
## RESULTS of all.equal
```

```
result3
```

```
## [1] "Different number of directed/undirected arcs"
## [2] "Different number of directed/undirected arcs"
## [3] "Different number of directed/undirected arcs"
## [4] "Different number of directed/undirected arcs"
## [5] "Different number of directed/undirected arcs"
```

```
cat("RESULTS of all.equal(cpdag)")
```

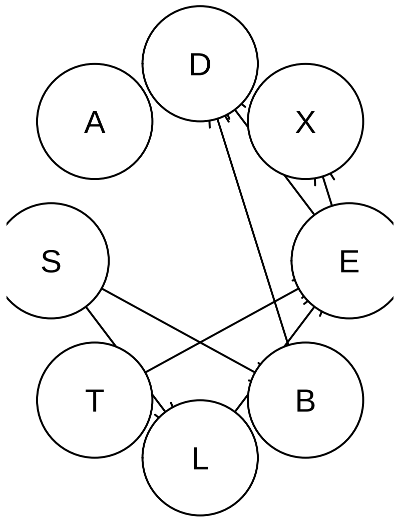
```
## RESULTS of all.equal(cpdag)
```

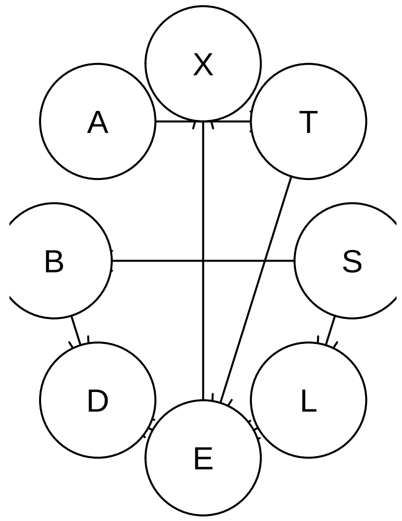
```
result4
```

```
## [1] "Different number of directed/undirected arcs"
## [2] "Different number of directed/undirected arcs"
## [3] "Different number of directed/undirected arcs"
## [4] "Different number of directed/undirected arcs"
## [5] "Different number of directed/undirected arcs"
```

Question 2

The asia dataset is divided into train(80%) and test(20%) set. computing the posterior probability distribution of S for each case and classify it in the most likely class.classifying the variable S given observations for all the rest of the variables.The confusion matrix is reported below:





```
## Confusion Matrix for asia data Learned from hc

##      pred
##        no yes
##   no  322 146
##   yes 120 412

## The Accuracy of model

## [1] 0.734

## Misclassfication rate

## [1] 0.266

## Confusion Matrix for TRUE DAG

##      dagpred
##        no yes
##   no  322 146
##   yes 120 412

## Accuracy for TRUE DAG

## [1] 0.734

## Misclassfication rate for TRUE DAG

## [1] 0.266
```

Question 3

classifying S given observations only for the Markov blanket of S, i.e. its parents plus its children plus the parents of its children minus S itself. The he confusion matrix is reported.

```
cat("Markov blanket of S")
```

```
## Markov blanket of S
```

```
mb(bnFit,node="S")
```

```
## [1] "L" "B"
```

```
## Confusion Matrix for Markov Blanket of S
```

```
##      dagpred_mb
##         no yes
##   no  322 146
##   yes 120 412
```

```
## Accuracy  rate
```

```
## [1] 0.734
```

```
## Misclassfication rate
```

```
## [1] 0.266
```

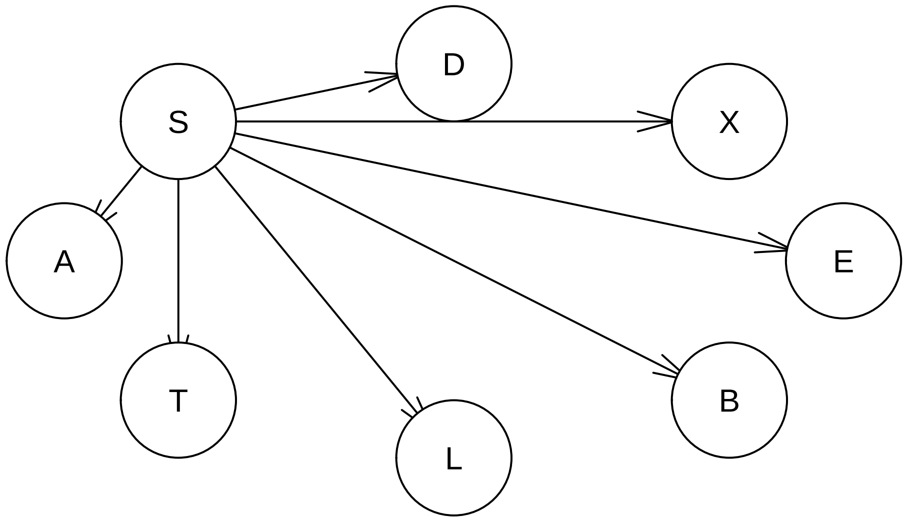
Question 4

Navie Bayes classifier is modelled as Bayesian Network. In Navie Bayes classifier assumption is the predictive variables are independent given the class variable.

```
n=dim(asia)[1]

id=sample(1:n, floor(n*0.8))
train=asia[id,]
test=asia[-id,]
xTrain <- train[,-2]
yTrain <- train[,2] #Response

xTest <- test[,-2]
yTest <- test[,2]
n<-naive.bayes(train,"S")
cat("The plot for the Navie Bayes")
```

```
## The plot for the Navie Bayes
```

```
plot(n)
```

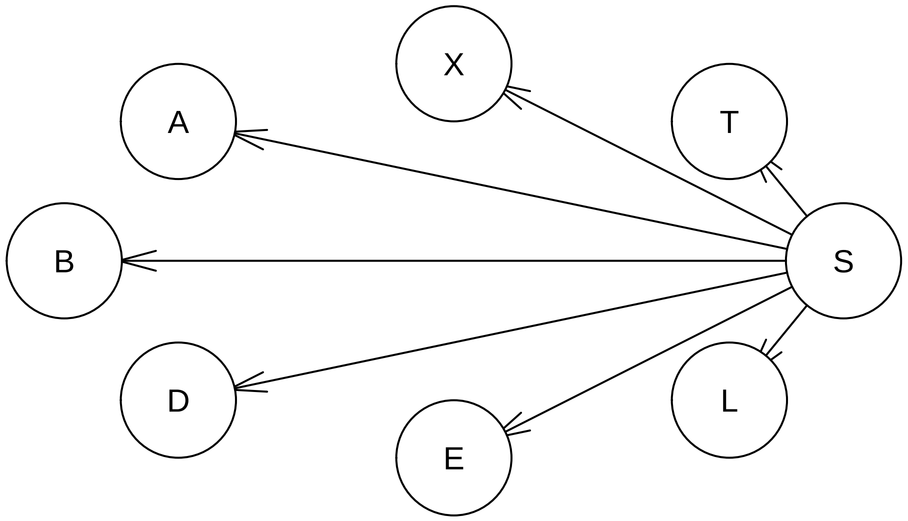


Constructing the Bayesian Network with hand.

```
dag_navie = model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")
cat("Bayesian Network constructed with hand")
```

```
## Bayesian Network constructed with hand
```

```
plot(dag_navie)
```



```r
dag_navie <- bn.fit(dag_navie, train, method = "mle")
dagGrain_navie <- as.grain(dag_navie)
dagCompile_navie <- compile(dagGrain_navie)
```

The results of Navie Bayes Classifier for multiple runs is displayed below:

```
## Confusion Matrix for Navie Bayes Model for FIRST RUN

##      dagpred_n
##        no yes
##   no  387 116
##   yes 169 328

## Accuracy of the model for FIRST RUN

## [1] 0.715

## Misclassification of the model for FIRST RUN

## [1] 0.285

## Confusion Matrix for Navie Bayes Model for SECOND RUN

##      dagpred_n
##        no yes
##   no  363 110
##   yes 188 339

## Accuracy of the model for SECOND RUN

## [1] 0.702
## Misclassification of the model for SECOND RUN

## [1] 0.298
```

Question 5

Comparing the results from step 2- step 4

Firstly comparing the results of setp2 and step 3. In the step2 we have classified the variable S given observations for all the rest of the variables and in step 3 classifying S given observations only for the Markov blanket of S. Given in a BN structure, a DAG represents the independence relationships.D-Separation in DAG implies independence.**In a Bayesian network, a variable S is conditionally independent of all other variables given its Markov blanket**. The Markov blanket of S d-separates S from all other nodes. Hence from the results of cofusion matrix we can observe that we have obtained **similar(same) confusion matrix**.

Next comparing the results from step4 where Navie Bayes classifier is modelled as bayesian network with the assumption that all predictive variables are independent given the class variable. The defining characteristic of the Naive Bayes model is that it makes the extremely aggressive and over-simplified assumption.The structure is given as a priori (and hence no structure learning procedure is required).Usually this independence assumption works well for most cases from the results of confusion matrix(running multiple times) we can see results are almost same(similar) and differs in some cases. How ever Bayesian Network classifier from step 2 and step3 doesnot outperform from step4.

Contribution

The report is prepared with the help of all the group members, each and every individual has contributed for the successful completion of Lab1.

- Question 1- Saman and Rab Nawaz
- Question 2- Tania and Priya
- Question 3- Farha and Priya
- Question 4- Rab Nawaz and Tania
- Question 5- Farha and Saman

Report is complied by all the group members.

Appendix

```
#Question1
library(bnlearn)
library(gRain)
library(MASS)
data(asia)
hcrun1<-hc(asia,restart = 0)
hcrun2<-hc(asia,restart=100)
par(mfrow=c(1,2))
plot(hcrun1)
plot(hcrun2)
vstructs(hcrun1)
vstructs(hcrun2)
arcs(hcrun1)
arcs(hcrun2)
result1<-NULL
result2<-NULL
```

```r
for(i in 1:5)
{
  hcrun1<-hc(asia,restart = 0)
  hcrun2<-hc(asia,restart=100)
  result1[i]<-all.equal(hcrun1,hcrun2)
  result2[i]<-all.equal(cpdag(hcrun1),cpdag(hcrun2))
}
cat("Results with all.equals:")
result1
cat("Results with all.equals(cpdag):")
result2

#CASE2

hcrun3<-hc(asia,score="bde",iss=1,restart = 0)
hcrun4<-hc(asia,score="bde",iss=10,restart = 0)
par(mfrow=c(1,2))
plot(hcrun3)
plot(hcrun4)
arcs(hcrun3)
arcs(hcrun4)
vstructs(hcrun3)
vstructs(hcrun4)
result3<-NULL
result4<-NULL
g<-NULL
for(i in 1:5)
{
  hcrun3<-hc(asia,score="bde",iss=1,restart = 0)
  hcrun4<-hc(asia,score="bde",iss=10,restart = 0)
  result3[i]<-all.equal(hcrun3,hcrun4)
  result4[i]<-all.equal(cpdag(hcrun3),cpdag(hcrun4))
}
cat("RESULTS of all.equal")
result3
cat("RESULTS of all.equal(cpdag)")
result4

#Question2

#split the data
n=dim(asia)[1]

id=sample(1:n, floor(n*0.8))
train=asia[id,]
test=asia[-id,]

xTrain <- train[,-2]
yTrain <- train[,2] #Response

xTest <- test[,-2]
yTest <- test[,2] #Response
```

```r
#true plot
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
plot(dag)
arcs(dag)
vstructs(dag)
cpdag(dag)

dagFit <- bn.fit(dag, train, method = "mle")

dagGrain <- as.grain(dagFit)
dagCompile <- compile(dagGrain)

#exact algorithm for inference in BNs

##BN for maxlikelihood parameter estimator
bnModel <- hc(train)
plot(bnModel)
bnFit <- bn.fit(bnModel, train, method = "mle")

#Convert bn.fit object to grain object
bnAsGrain <- as.grain(bnFit)
bnCompile <- compile(bnAsGrain)

sProb <- NULL
for(i in 1:1000){
  z <- NULL
  for(j in c("A","T","L","B","E","X","D")){
    if(xTest[i,j]=="no"){
      z <- c(z, "no")
    } else {
      z <- c(z, "yes")
    }
  }
  setEvid <- setEvidence(bnCompile, nodes = c("A","T","L","B","E","X","D"),
                         states = z)
  sProb[i] <- querygrain(setEvid, nodes = c("S"))
}

sProbList <- (matrix(0,1000,ncol = 2))
for(i in 1:1000){
  sProbList[i,1] <- sProb[[i]][[1]]
  sProbList[i,2] <- sProb[[i]][[2]]
}

pred <- c()
pred[which(sProbList[,1] > 0.5)] <- "no"
pred[which(sProbList[,2] >= 0.5)] <- "yes"
t_first<-table(test$S, pred)
accuracy1_first<- (sum(diag(t_first)))/sum(t_first)
accuracy1_first
misclassification_first<-1-accuracy1_first
misclassification_first
```

```
new_prediction=cpdist(bnFit,nodes=c("A","T","L","B","E","X","D"),evidence=TRUE,method="lw")

#true plot
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
dagFit <- bn.fit(dag, train, method = "mle")
dagGrain <- as.grain(dagFit)
dagCompile <- compile(dagGrain)

dagsProb <- NULL
for(i in 1:1000){
  dagz <- NULL
  for(j in c("A","T","L","B","E","X","D")){
    if(xTest[i,j]=="no"){
      dagz <- c(dagz, "no")
    } else {
      dagz <- c(dagz, "yes")
    }
  }
  dagsetEvid <- setEvidence(dagCompile, nodes = c("A","T","L","B","E","X","D"),
                            states = dagz)
  dagsProb[i] <- querygrain(dagsetEvid, nodes = c("S"))
}

dagsProbList <- (matrix(0,1000,ncol = 2))
for(i in 1:1000){
  dagsProbList[i,1] <- dagsProb[[i]][[1]]
  dagsProbList[i,2] <- dagsProb[[i]][[2]]
}

dagpred <- c()
dagpred[which(dagsProbList[,1] >= 0.5)] <- "no"
dagpred[which(dagsProbList[,2] >= 0.5)] <- "yes"
t2<-table(test$S, dagpred)
accuracy1_dag<-(sum(diag(t2)))/sum(t2)
accuracy1_dag
misclassification_dag<-1-accuracy1_first
misclassification_dag


#Question 3

mb(bnFit,node="S")

dagsProb_mb <- NULL
for(i in 1:1000){
  dagz_mb <- NULL
  for(j in c("L","B")){
    if(xTest[i,j]=="no"){
      dagz_mb <- c(dagz_mb, "no")
    } else {
      dagz_mb <- c(dagz_mb, "yes")
    }
  }
  dagsetEvid_mb <- setEvidence(dagCompile, nodes = c("L","B"),
                               states = dagz_mb)
  dagsProb_mb[i] <- querygrain(dagsetEvid_mb, nodes = c("S"))
}

dagsProbList_mb <- (matrix(0,1000,ncol = 2))
for(i in 1:1000){
  dagsProbList_mb[i,1] <- dagsProb_mb[[i]][[1]]
  dagsProbList_mb[i,2] <- dagsProb_mb[[i]][[2]]
}

dagpred_mb <- c()
dagpred_mb[which(dagsProbList_mb[,1] >= 0.5)] <- "no"
dagpred_mb[which(dagsProbList_mb[,2] >=0.5)] <- "yes"
t2<-table(test$S, dagpred_mb)

accuracy1_mb<-(sum(diag(t2)))/sum(t2)
accuracy1_mb
misclassification_mb<-1-accuracy1_first
misclassification_mb

##Question4

n=dim(asia)[1]

id=sample(1:n, floor(n*0.8))
train=asia[id,]
test=asia[-id,]
xTrain <- train[,-2]
yTrain <- train[,2] #Response

xTest <- test[,-2]
yTest <- test[,2] #Response
 g<-hc(train)

n<-naive.bayes(train,"A")


dag_navie = model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")
dag_navie <- bn.fit(dag_navie, train, method = "mle")
dagGrain_navie <- as.grain(dag_navie)
dagCompile_navie <- compile(dagGrain_navie)

dagsProb_n <- NULL
for(i in 1:1000){
  dagn <- NULL
  for(j in c("A","T","L","B","E","X","D")){
    if(xTest[i,j]=="no"){
      dagn <- c(dagn, "no")
    } else {
      dagn <- c(dagn, "yes")
    }
  }
  dagsetEvid_n <- setEvidence(dagCompile_navie, nodes = c("A","T","L","B","E","X","D"),
                              states = dagn)
  dagsProb_n[i] <- querygrain(dagsetEvid_n, nodes = c("S"))
}

dagsProbList_n <- (matrix(0,1000,ncol = 2))
for(i in 1:1000){
  dagsProbList_n[i,1] <- dagsProb_n[[i]][[1]]
  dagsProbList_n[i,2] <- dagsProb_n[[i]][[2]]
}

dagpred_n <- c()
dagpred_n[which(dagsProbList_n[,1] >= 0.5)] <- "no"
dagpred_n[which(dagsProbList_n[,2] >= 0.5)] <- "yes"
t1<-table(test$S, dagpred_n)
accuracy1_n<- (sum(diag(t1)))/sum(t1)
accuracy1_n
misclassification_n<-1-accuracy1_n
misclassification_n
```

LAB 2: HIDDEN MARKOV MODELS

The purpose of this lab is to model the behavior of a robot that walks around a ring, The ring is divided into 10 sectors. At any given time point, the robot is in one of the sect ors and decides with equal probability to stay in that sector or move to the next sector. We need to define tranision probability which is Stochastic matrix containing the transition probabilities between the states. So,the probability for the two adjacent columns is taken as 0.5 in a such a way that diagonal of a matrix should be 0.5 and the rows of the matrix must sum to 1.

The device is not very accurate though, If the robot is in the sector i, then the device will report that the robot is in the sectors $[i-2; i+2]$ with equal probability. We need to define emission probability which is a Stochastic matrix containing the emission probabilities of the states. So, the probability for the five adjacent columns is taken as 0.2 and the rows of the matrix must sum to 1.

Building a hidden Markov model

For building a hidden Markov model with the following settings **initHMM()** function is used. This function is designed to make inference on the states through the observation of emissions. The stochastics(randomly defined) of the model is defined by the **starting probabilities** of the states, **transition probabilities** between states and the **emission probabilities** of the states.

The Hidden markov mode is initialised with the following settings.

Part 1

```
library(HMM)
library(ggplot2)
library(gridExtra)
library(entropy)




state_value =c(1:10)
```

```
symbol_value = c(1:10)
startP_value = rep(0.1, 10)
transProbs_value=matrix(0,nrow = 10,ncol = 10)
diag(transProbs_value) = 0.5 #setting on diagnal
diag(transProbs_value[,-1]) = 0.5 #setting valve next entry to the diagnal value
transProbs_value[10,1] = 0.5

emission_prob_value <- matrix(c(0.2,0.2,0.2,0,0,0,0,0,0.2,0.2,
                          0.2,0.2,0.2,0.2,0,0,0,0,0,0.2,
                          0.2,0.2,0.2,0.2,0.2,0,0,0,0,0,
                          0,0.2,0.2,0.2,0.2,0.2,0,0,0,0,
                          0,0,0.2,0.2,0.2,0.2,0.2,0,0,0,
                          0,0,0,0.2,0.2,0.2,0.2,0.2,0,0,
                          0,0,0,0,0.2,0.2,0.2,0.2,0.2,0,
                          0,0,0,0,0,0.2,0.2,0.2,0.2,0.2,
                          0.2,0,0,0,0,0,0.2,0.2,0.2,0.2,
                          0.2,0.2,0,0,0,0,0,0.2,0.2,0.2),ncol = 10,byrow = TRUE)


initilize_hmm <- initHMM(States = state_value,
                   Symbols = symbol_value,
                   startProbs = startP_value,
                   transProbs = transProbs_value,
                   emissionProbs = emission_prob_value)
initilize_hmm
```

```
## $States
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $Symbols
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $startProbs
##   1   2   3   4   5   6   7   8   9  10
## 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
##
## $transProbs
##     to
## from   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
##
## $emissionProbs
##       symbols
## states   1   2   3   4   5   6   7   8   9  10
##     1  0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2
##     2  0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2
##     3  0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0
##     4  0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0
##     5  0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0
##     6  0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0
##     7  0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0
##     8  0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2
##     9  0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2
##     10 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2
```

The init HMM function is used to initialize a Hidden markov model.In HMM the states are hidden and hence cannot be seen.A person can only see the emissions from the observations.THe function takes five variables States,Symbols,startsProbs,transProbs,emissionProbs.It is given in the question that the robot has equal probability to stay in the sector or move to the next. So we take a probability of 0.5 for the two near by columns of transProb matrix and go diagonally. Also it is mentioned that the device locates the robot with probabilities [i+2,i-2].Hence the emission probabiliy matrix is given a value of 0.2.

Part 2

Simulate the HMM for 100 time steps.

Simulating the path of states and observations for a given Hidden Markov Model using **simHMM** with the 100 time steps as length of the simulated sequence. The reult returned by simHMM is :

```
simulated_hmm <- simHMM(initilize_hmm, 100)
simulated_hmm
```

```
## $states
##   [1] 10  1  1  2  3  4  5  6  7  7  8  8  8  8  8  9 10  1  1  2  2  2  3
##  [24]  4  5  6  6  7  8  8  9  9  9 10  1  2  2  2  3  3  3  4  5  5  6  6
##  [47]  7  8  8  8  9  9  9  9  9  9 10 10  1  2  3  3  3  4  4  5  6  7  8
##  [70]  9  9 10 10  1  2  3  3  3  4  5  6  7  7  8  8  8  9  9 10 10 10 10
##  [93] 10 10 10  1  2  2  3  4
##
## $observation
##   [1]  8  9  2 10  1  5  5  4  9  6 10  8  9 10  7  1  9  1  3  1 10  4  2
##  [24]  4  7  6  8  8 10 10  9  8 10  1  3 10  4  4  5  2  4  2  5  3  6  6
##  [47]  7  9  9  8 10  7  8  1  9  8  1  8  9  3  4  3  4  4  4  4  5  9  8
##  [70]  7  9 10  2  9  1  3  4  1  6  6  5  6  7  8  8  7  7  7  8  2  2  1
##  [93]  2  8  2 10 10  3  1  4
```

part 3,4

We need to discard the hidden states from the sample obtained in the step2. Only observations are used to compute the filtered and smoothed probability distributions for each of the 100 time points.

First computing the filtered probability distributions using **forward** function. Which returns a matrix containing the forward probabilities.

Next finding the smoothed probability distribution. Smoothing is used to deal with the problem of zero probabilities.Smoothing is used to improve the probability estimates. smoothing techniques used for adjusting the maximum likelihood estimate of probabilities to produce more accurate probabilities. **posterior** function is used compute a matrix of the posterior probabilities.

The **Viterbi-algorithm** computes the most probable path of states for a sequence of observations for a given Hidden Markov Model.

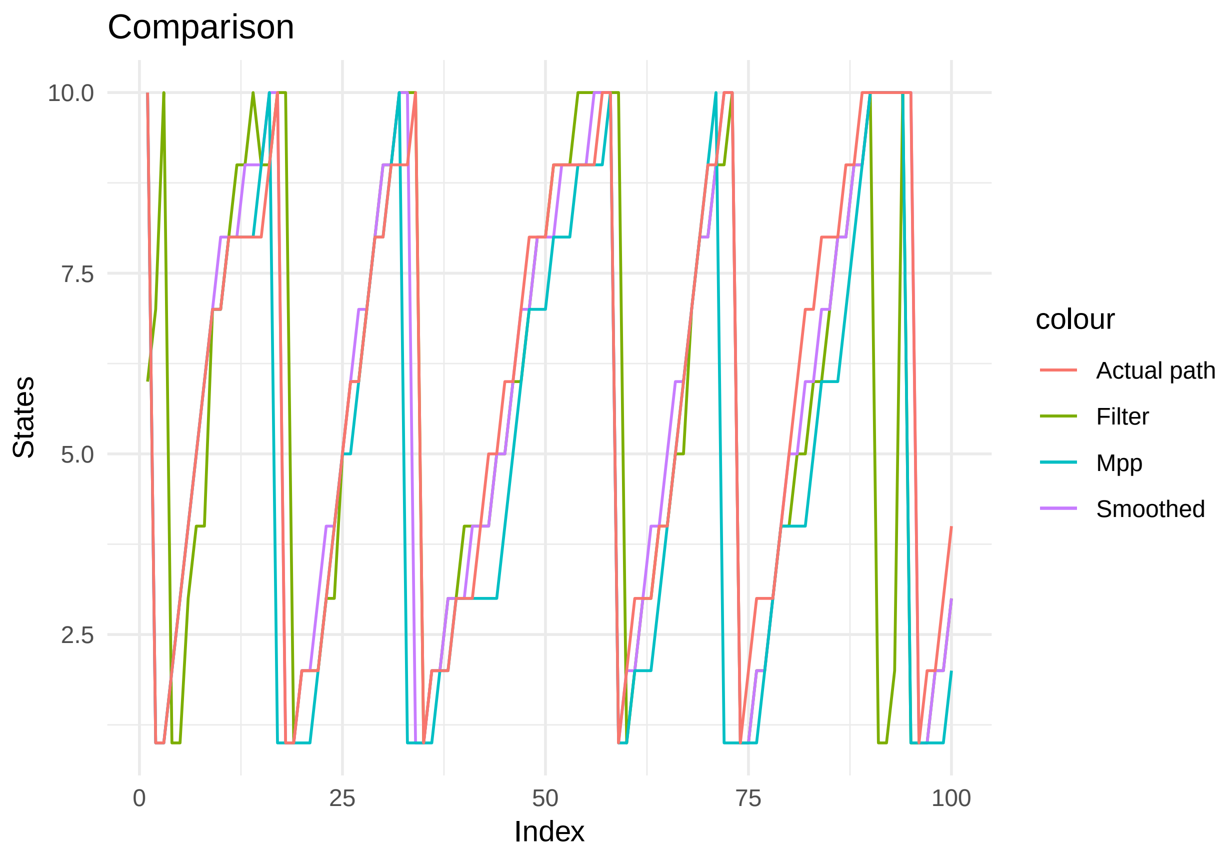
```
##smooth the observations for the 100 points

smooth_value <- posterior(hmm = initilize_hmm, observation = observations)

###finding the most probable path


probable_path <- viterbi(initilize_hmm, observation = observations)

##actual path
actual_Path <- simulated_hmm$states
```

Now we need to find the accuracy of the filtered,smoothed probability distribution and the accuacy of the most probable path we found

```
## The accuracy of filtered distribution is  0.49
```

```
## The accuracy of smoothed distribution is  0.63
```

```
## The accuracy of most probable path is  0.51
```



Part5

Repeating the previous exercise with different simulated samples. The results are:

filter	smooth	proable path
0.57	0.61	0.57

filter	smooth	proable path
0.54	0.70	0.68
0.47	0.53	0.46
0.53	0.64	0.51
0.65	0.73	0.45
0.50	0.77	0.49
0.64	0.65	0.47
0.50	0.71	0.57
0.51	0.64	0.43
0.54	0.74	0.57

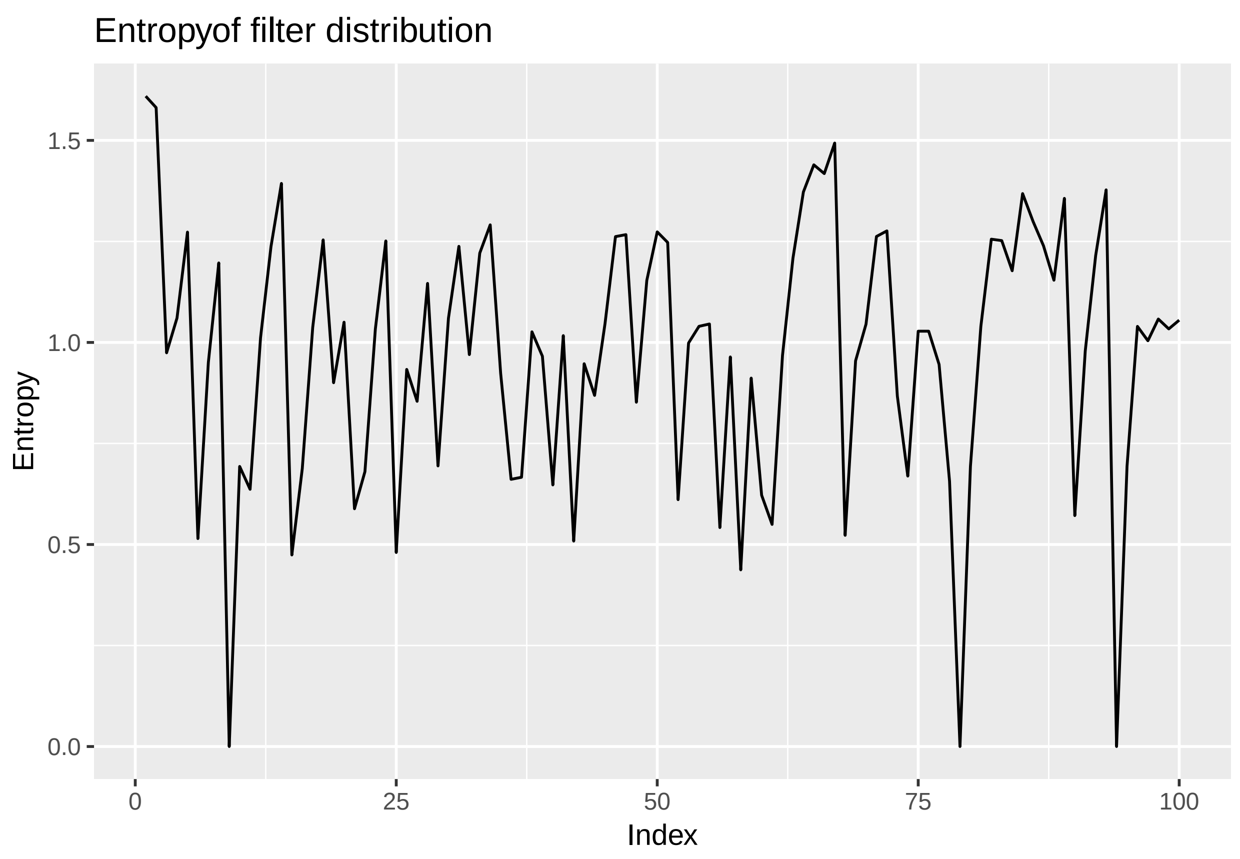
From the results we can see that smoothed distribution is more accurate than the most probable paths. Viterbi's algorithm is similar to the forward algorithm, with the difference that the summation over the states at time step k becomes a maximization. The reason smoothing techniques performs well it adjusts the maximum likelihood estimate of probabilities to produce more accurate probabilities.

Posterior uses forward and backward variables while filter only uses forward variables to determine the result. Moreover, posterior chooses the most probable state at every position while viterbi determines the most probable path on average thus especially in case when there are many paths, it outperforms viterbi algorithm.

Part 6

computing the entropy of the filtered distributions with the function entropy.empirical. The result is shown in a plot.

```
entropy_value<- data.frame(Index = 1:100 ,Entropy = apply(normalized_filter,
                                                          MARGIN = 2,
                                                          FUN =entropy.empirical))

ggplot(data = entropy_value, aes(x = Index, y = Entropy)) + geom_line()+
  ggtitle("Entropyof filter distribution")
```



Entropy is the measure of uncertainity. The empirical entropy estimator is the maximum likelihood estimator. If there are many zero counts and the sample size is small it is very inefficient and also strongly biased. Lower the entropy defines the better prediction.

From the graph above we can state that it is not true that the more observations we have the better you know where the robot is because the assumption of Markovian model is that it depends only on the previous observation thus increasing the number of observations does not effect the result significantly.

Part 7

```
Normalized_smoother[,100] %*% initilize_hmm$transProbs
```

```
##      to
##       1           2           3           4           5 6 7 8 9 10
##   [1,] 0 0.001666667 0.003928571 0.003333333 0.001071429 0 0 0 0  0
```

Since we have markovian assumption that future(t+1) does not depend on past(t-1) given present(t) because present contains the information about past. Probability of a state can be determined by product rule. Thus we have used the information of state 100 which contains the information about the previous states and multiplied it with the transition probability to get the state probability at time step 101

Appendix

```r
library(HMM)
library(ggplot2)
library(gridExtra)
library(entropy)

1.###hidden Markov model (HMM)
'
The init HMM function is used to initialize a Hidden markov model.In HMM the states are hidden and henc
A person can only see the emissions from the observations.THe function takes five variables States,Symb
transProbs,emissionProbs.It is given in the question that the robot has equal probability to stay in th
So we take a probability of 0.5 for the two near by columns of transProb matrix and go diagonally.
Also it is mentioned that the device locates the robot with probabilities [i+2,i-2].Hence the emission 
'


state_value =c(1:10)
symbol_value = c(1:10)
startP_value = rep(0.1, 10)
transProbs_value=matrix(0,nrow = 10,ncol = 10)
diag(transProbs_value) = 0.5 #setting on diagnal
diag(transProbs_value[,-1]) = 0.5 #setting valve next entry to the diagnal value
transProbs_value[10,1] = 0.5

emission_prob_value <- matrix(c(0.2,0.2,0.2,0,0,0,0,0,0.2,0.2,
                          0.2,0.2,0.2,0.2,0,0,0,0,0,0.2,
                          0.2,0.2,0.2,0.2,0.2,0,0,0,0,0,
                          0,0.2,0.2,0.2,0.2,0.2,0,0,0,0,
                          0,0,0.2,0.2,0.2,0.2,0.2,0,0,0,
                          0,0,0,0.2,0.2,0.2,0.2,0.2,0,0,
                          0,0,0,0,0.2,0.2,0.2,0.2,0.2,0,
                          0,0,0,0,0,0.2,0.2,0.2,0.2,0.2,
                          0.2,0,0,0,0,0,0.2,0.2,0.2,0.2,
                          0.2,0.2,0,0,0,0,0,0.2,0.2,0.2),ncol = 10,byrow = TRUE)


initilize_hmm <- initHMM(States = state_value,
                   Symbols = symbol_value,
                   startProbs = startP_value,
                   transProbs = transProbs_value,
                   emissionProbs = emission_prob_value)


##############################2############################


simulated_hmm <- simHMM(initilize_hmm, 100)


###############3#########################

#Discard hidden states from the sample
```

```
observations = simulated_hmm$observation

##filter the observations for the 100 points

filtered_value = forward(initilize_hmm,observations)

##smooth the observations for the 100 points

smooth_value <- posterior(hmm = initilize_hmm, observation = observations)

###finding the most probable path


probable_path <- viterbi(initilize_hmm, observation = observations)
##actual path
actual_Path <- simulated_hmm$states


##A plot is the best way to compare the how much the path deviates from the actual path

data_frame <- data.frame(obs = 1:100, probable_path, actual_Path)

probable_path_plot <- ggplot(data_frame, aes(x= obs, y=probable_path)) + geom_path() +
  labs(y="Most probable path")

actual_Path_plot <- ggplot(data_frame, aes(x= obs, y=actual_Path)) + geom_path() +
  labs(y="Actual path ")

grid.arrange( probable_path_plot, actual_Path_plot)


##################################4####################################
'
Now we need to find the accurcy of the filtered,smoothed probability distribution and the accuacy of the

'
##filter accuracy

##fisrt we do the normalization using the exp and prop.table


Normalized_filter <- exp(filtered_value)
normalized_final <- prop.table(Normalized_filter, margin = 2)

##most probable state
best_value<- data.frame(state=0, observation=1:100)
for(i in 1:100){
  best_value[i,1] <- which.max(normalized_final[,i])
}

##compare for accuracy filter

table_comparison<- table(best_value[,1] ==actual_Path)
```

```
accuracy_filter <- table_comparison[[2]]/sum(table_comparison[[1]], table_comparison[[2]])

accuracy_filter

######smoother accuracy

Normalized_smoother <- exp(smooth_value)
normalized_final_smoother <- prop.table(Normalized_smoother, margin = 2)

##most probable state
best_value_smoother<- data.frame(state=0, observation=1:100)
for(i in 1:100){
  best_value_smoother[i,1] <- which.max(normalized_final_smoother[,i])
}


table_comparison<- table(best_value_smoother[,1] ==actual_Path)
accuracy_smoother <- table_comparison[[2]]/sum(table_comparison[[1]], table_comparison[[2]])

accuracy_smoother


###probable_state accuracy


table_comparison<- table(probable_path==actual_Path)
accuracy_path <- table_comparison[[2]]/sum(table_comparison[[1]], table_comparison[[2]])

accuracy_path

final_result = data.frame(index=c(1:100),
                   "filter_result"=most_probable_state_filter,
                   "smooth_result"=most_probable_state_smooth,
                   "probable_path"=probable_path)


ggplot(data = final_result) +
  geom_line(aes(x=index,y=filter_result , col = "Filter")) +
  geom_line(aes(x=index,y=smooth_result , col = "Smoothed")) +
  geom_line(aes(x=index,y=probable_path , col = "Mpp")) +
  geom_line(aes(x=index,y=actual_path , col = "Actual path")) +
  theme_minimal() +xlab("Index") +
  ylab("States") + ggtitle("Comparison")

#############################5#######################################

###writing a functon for finding the filter,smmother and probable path accuracy so that it can be used

Robot_state <- function(Markov,simobs ){

  simulated_hmm <- simHMM(Markov, simobs)

  observations = simulated_hmm$observation
```

```
filtered_value = forward(Markov,observations)

##smooth the observations for the 100 points

smooth_value <- posterior(hmm = Markov, observation = observations)

###finding the most probable path


probable_path <- viterbi(Markov, observation = observations)
library(HMM)
##actual path
actual_Path <- simulated_hmm$states




Normalized_filter <- exp(filtered_value)
normalized_final <- prop.table(Normalized_filter, margin = 2)

##most probable state
best_value<- data.frame(state=0, simobs)
for(i in simobs){
  best_value[i,1] <- which.max(normalized_final[,i])
}

##compare for accuracy filter

table_comparison<- table(best_value[,1] ==actual_Path)
accuracy_filter <- table_comparison[[2]]/sum(table_comparison[[1]], table_comparison[[2]])

accuracy_filter

######smoother accuracy



Normalized_smoother <- exp(smooth_value)
normalized_final_smoother <- prop.table(Normalized_smoother, margin = 2)

##most probable state
best_value_smoother<- data.frame(state=0, simobs)
for(i in simobs){
  best_value_smoother[i,1] <- which.max(normalized_final_smoother[,i])
}


table_comparison<- table(best_value_smoother[,1] ==actual_Path)
accuracy_smoother <- table_comparison[[2]]/sum(table_comparison[[1]], table_comparison[[2]])

accuracy_smoother
```

```
  ###probable_state accuracy

  table_comparison<- table(probable_path==actual_Path)
  accuracy_path <- table_comparison[[2]]/sum(table_comparison[[1]], table_comparison[[2]])



  return(data.frame("filter accuracy"= accuracy_filter,"smoothing accuracy" = accuracy_smoother,
                    "probable path accuracy"= accuracy_path))
}


simulations = data.frame()
for (i in 1:10) {
 simulations <- rbind(simulations,as.vector(as.numeric(unlist(Robot_state(initilize_hmm,100)))))
}
colnames(simulations) = c("filter","smooth","proable path")


#########################6#################################################

 entropy_value<- data.frame(Index = 1:100 ,
                            Entropy =
                              apply(normalized_final, MARGIN = 2,
                                    FUN=entropy.empirical))

ggplot(data = entropy_value, aes(x = Index, y = Entropy)) + geom_line() + ggtitle("Entropy of filter di


#########################7#################################################

Normalized_smoother[,100] %*% initilize_hmm$transProbs
```

LAB 3: STATE SPACE MODELS

The purpose of the lab is to put in practice some of the concepts State Space Model by implementing the particle filter for robot localization.

Part 1

The robot moves along horizontally along the X-axis according to the following models:

Transition Model: $p(z_t|z_{t-1}) = (N(z_t|z_{t-1}, 1), N(z_t|z_{t-1} + 1, 1), N(z_t|z_{t-1} + 2, 1))/3$

Emission Model: $p(x_t|z_t) = (N(x_t|z_t, 1), N(x_t|z_t - 1, 1), N(x_t|z_t + 1, 1))/3$

Initial Model : $p(z_1) = Uniform(0, 100)$

Implementing the State Space Models

We will now implement the given state space models i-e Emission and Transition models uing the initial model

Emission Model

```r
#emission model
emission_model<- function(z,standard_deviation){
  emission_value = (rnorm(1,mean = z,sd= standard_deviation) +
                      rnorm(1,mean = z - 1,sd= standard_deviation) +
                      rnorm(1,mean = z +1,sd= standard_deviation)
                      )/3
  return(emission_value)
}
```

Transition Model

```r
transition_model<- function(z,standard_deviation)
{
  transition_value <- (rnorm(1,mean = z,sd = standard_deviation) +
                         rnorm(1,mean = z+1,sd = standard_deviation) +
                         rnorm(1,mean = z+2,sd = standard_deviation)
                         )/3
  return(transition_value)
}
```

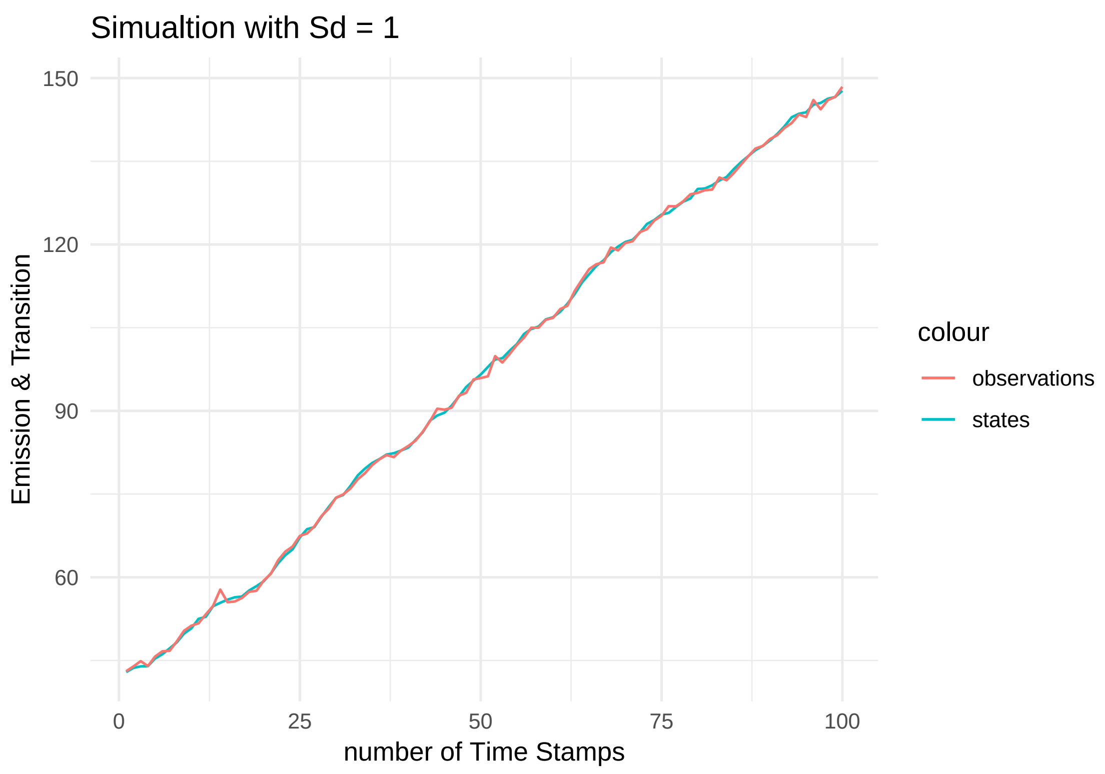
Simulating for T = 100 time steps with (Sd_Emission = 1 & Sd_Transition = 1)

We have to perform simulation in order to generate true states and observations from transition and emission model respectively.

```r
true_result_sd1 = generate_data(sd_emission = 1,sd_transition = 1,no_of_obs = 100)
```

When we simulate model for $T = 100$ time steps for given model, we obtain the $z_{1:100}$ which are states and $x_{1:100}$ which are observations.

```r
#plot comparision of emission model and transition model
ggplot(data = true_result_sd1) +
  geom_line(aes(x=index,y=states , col = "states")) +
  geom_line(aes(x=index,y=observations , col = "observations")) +
  theme_minimal() +xlab("number of Time Stamps") +
  ylab("Emission & Transition") + ggtitle("Simualtion with Sd = 1")
```



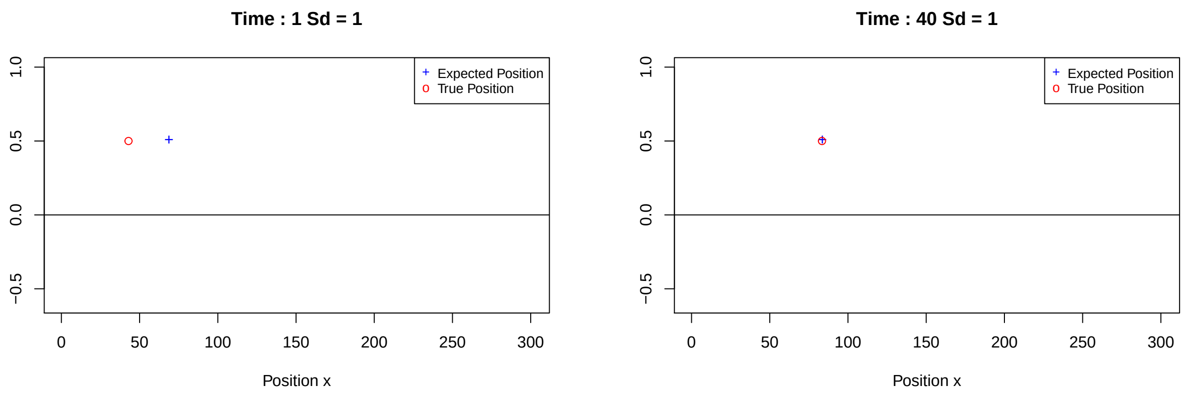
In the above graph X-Axis represents Time steps and Y-Axis represents values of states and observations. It can be observed that states and observation follow each other keeping Standard deviation of Emission and Transition = 1.

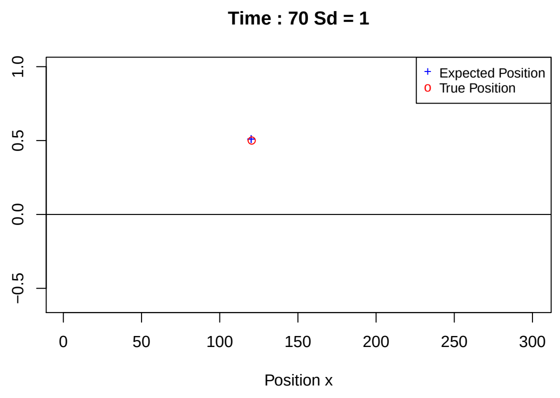
Localization of Robot

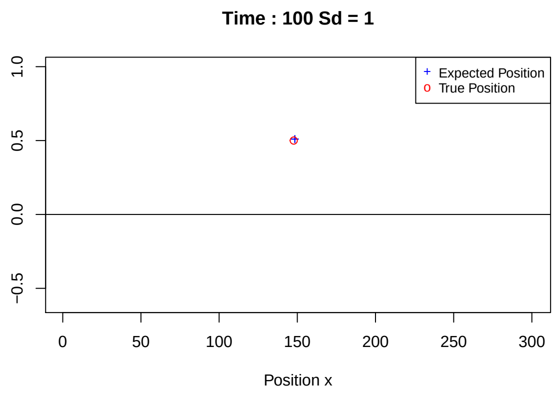
Implementation of particle Filter and comparision at different time steps

Now we have implemented the particle filter to determine the position of robot

```r
result_sd1 = particleFilter(X = true_result_sd1$observations, M = 100, Tsteps=100,
                            sd_transition = 1,sd_emission = 1)
#plot results with sd_emission = 1 and sd_transition = 1

plot_result(result = result_sd1, true_values = true_result_sd1,heading = "Sd = 1")
```





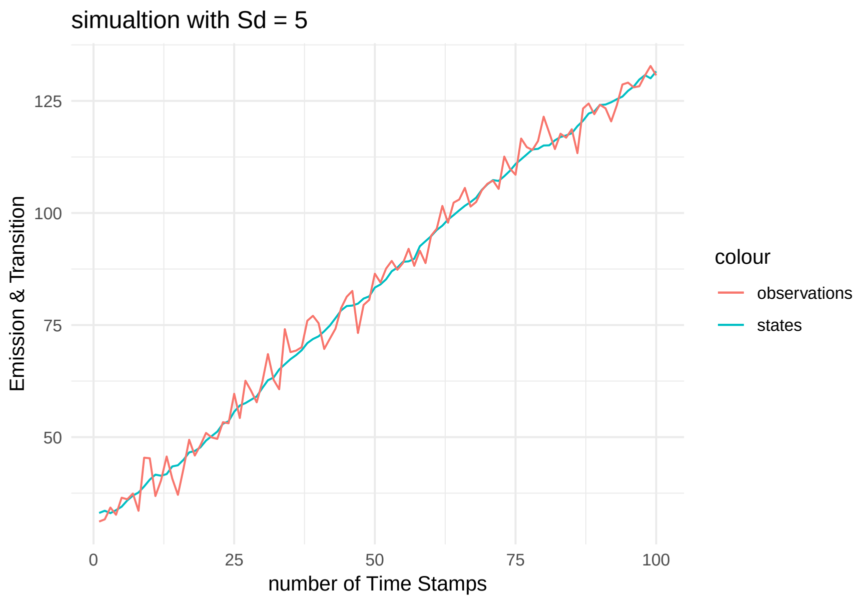


Using the particle filter, we have determined the localization of robot. The graphs represent the true states and estimated states of the robot. For sd = 1 for emission and transition, it can be seen that the expected position of robot is very close (almost same) to the true position of robot for all time steps except for the beginning i-e at time step 1.

Part 2(a)

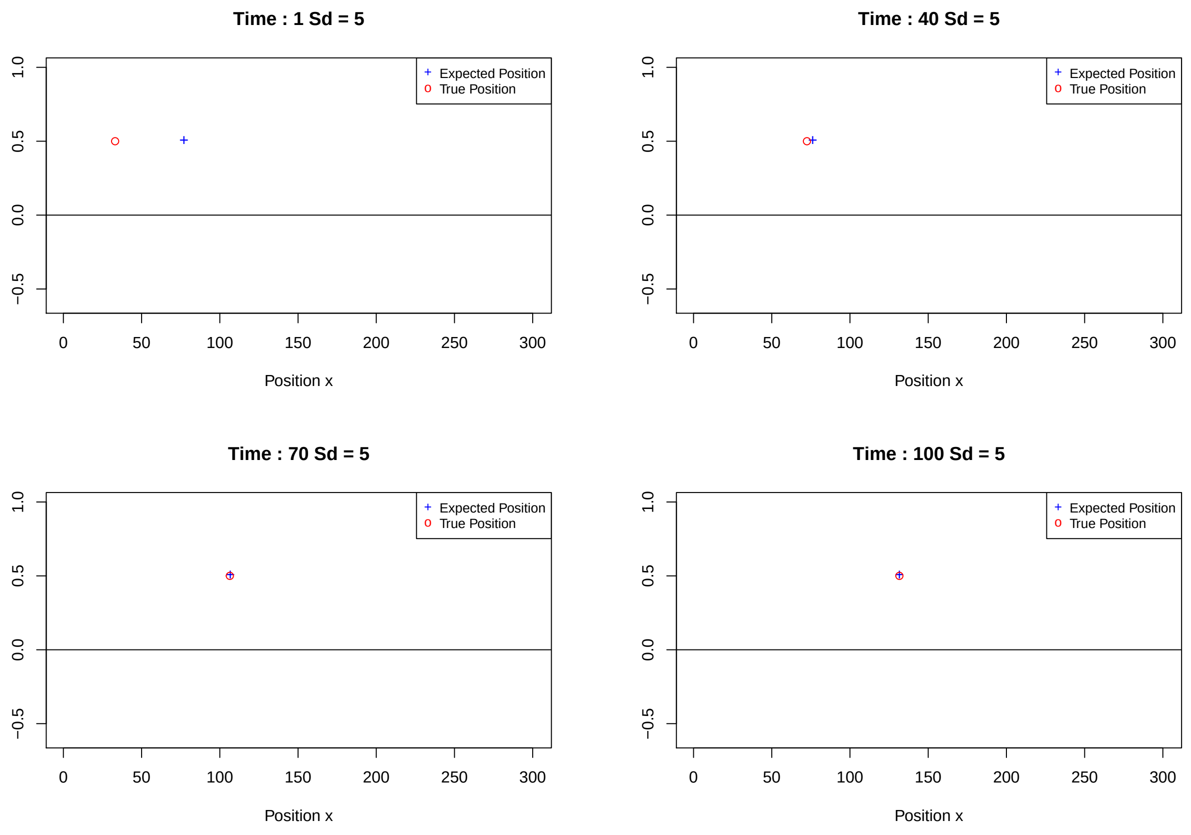
Simulating for T = 100 time with (Sd_Emission = 5 & Sd_Transition = 1)

```
true_result_sd5 = generate_data(sd_emission = 5,sd_transition = 1,no_of_obs = 100)
#plot comparision of emission model and transition model
ggplot(data = true_result_sd5) +
  geom_line(aes(x=index,y=states , col = "states")) +
  geom_line(aes(x=index,y=observations , col = "observations")) +
  theme_minimal() +xlab("number of Time Stamps") +
  ylab("Emission & Transition") + ggtitle("simualtion with Sd = 5")
```



After setting standard deviation of emission model = 5, it can be seen that there is some variation between observations and states.

```r
result_sd5 = particleFilter(X = true_result_sd5$observations, M = 100, Tsteps=100,
                              sd_transition = 1,sd_emission = 5)
#plot results with sd_emission = 5 and sd_transition = 1
plot_result(result = result_sd5, true_values = true_result_sd5,heading = "Sd = 5")
```

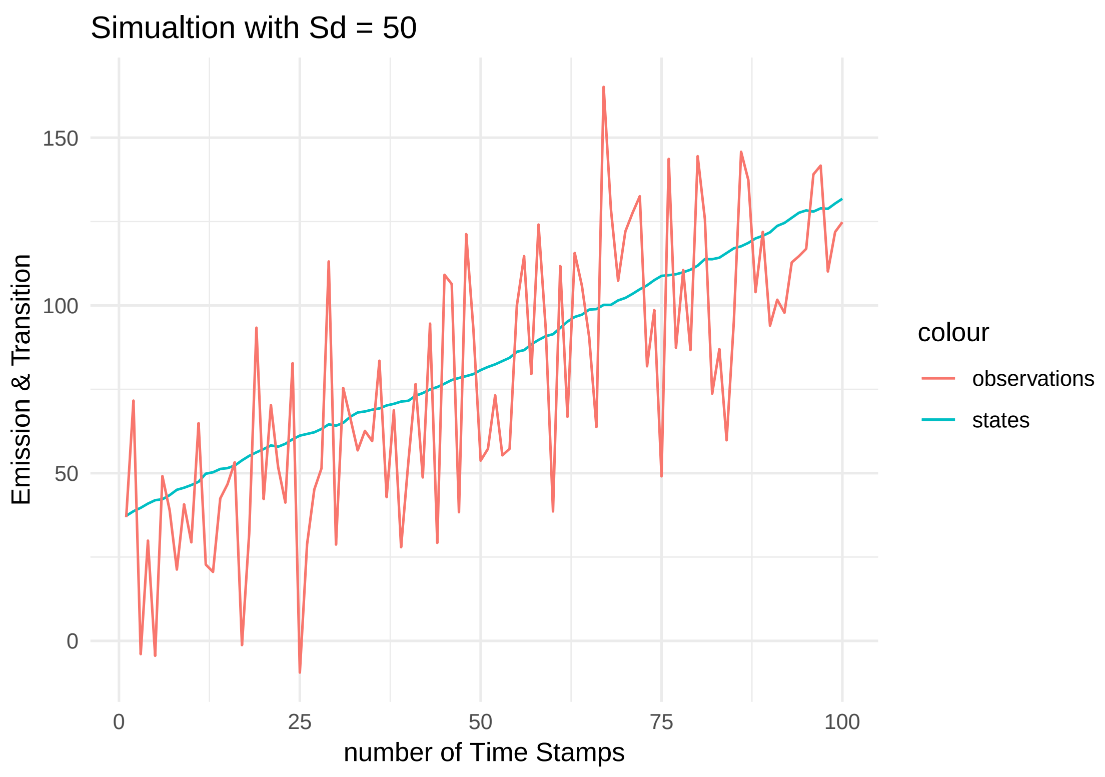


The expected position of robot does not exactly matches the true position in the till time step = 40 but with increasing time steps, the uncertainty is decreased.

Part 2(b)

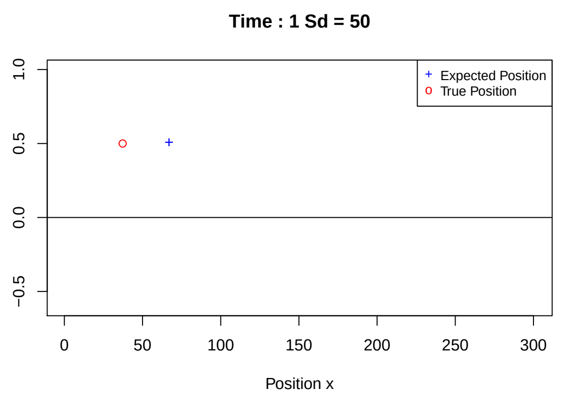
Simulating for T = 100 time with (Sd_Emission = 50 & Sd_Transition = 1)

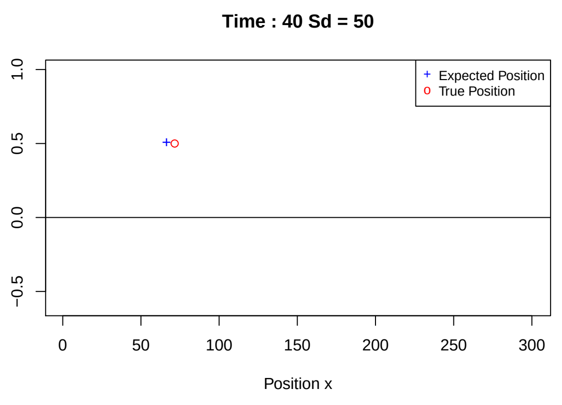
```r
true_result_sd50 = generate_data(sd_emission = 50,sd_transition = 1,no_of_obs = 100)

#plot comparision of emission model and transition model
ggplot(data = true_result_sd50) +
  geom_line(aes(x=index,y=states , col = "states")) +
  geom_line(aes(x=index,y=observations , col = "observations")) +
  theme_minimal() +xlab("number of Time Stamps") +
  ylab("Emission & Transition") + ggtitle("Simualtion with Sd = 50")
```

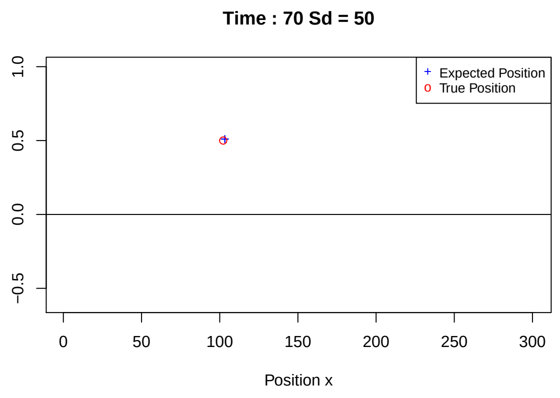


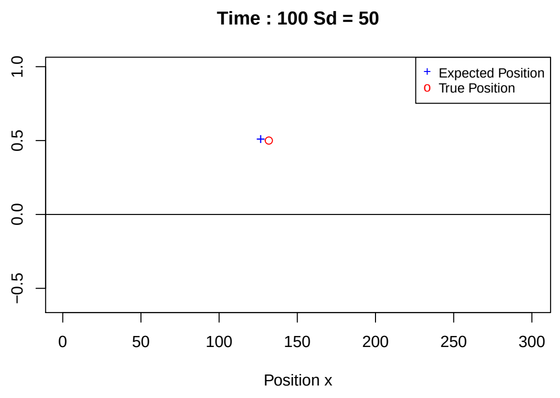
At standard deviation = 50, observations are fluctuating a lot and differs greatly from that of states which reflects the increase in uncertainty.

```r
result_sd50 = particleFilter(X = true_result_sd50$observations, M = 100, Tsteps=100,
                              sd_transition = 1,sd_emission = 50)
#plot results with sd_emission = 50 and sd_transition = 1
plot_result(result = result_sd50, true_values = true_result_sd50,heading = "Sd = 50")
```









The expected position of robot in this case is not as accurate as in case of sd = 1 or 5 but the overall result is still not too bad. There is uncertainty in the beginning as well as towards the end.

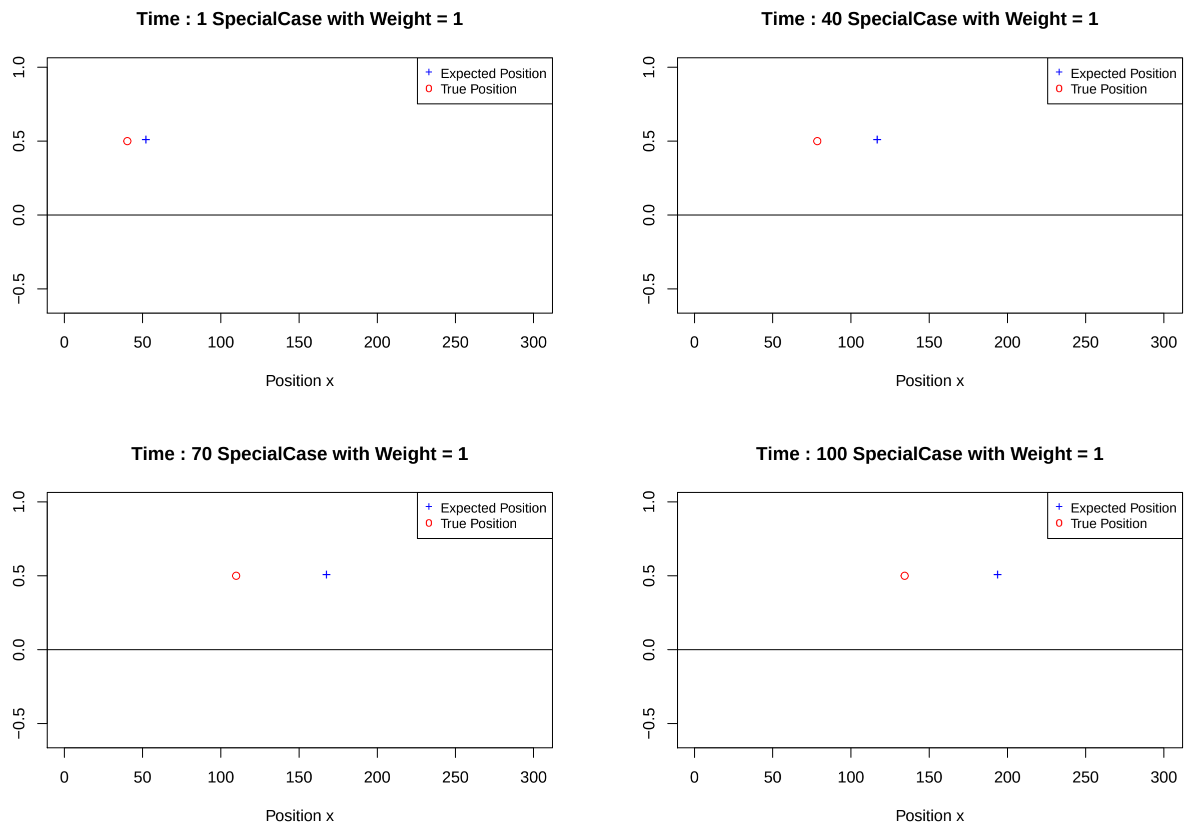
Part 3

Special Case

Here we have to keep weights = 1 for every time step

```
sepcial_particleFilter <- function(X, M, Tsteps, sd_transition,sd_emission)
{

  #Initialization
  ztBar <- matrix(0, nrow = 100, ncol = 100) #store transition model values
  ztBar[1,] = runif(100, 0, 100)
  wt <- matrix(1, 100, 100) #particle weights

  wt[1,] = rep((1/Tsteps),Tsteps)
  for(t in 2: (Tsteps))
  {
    for(m in 1:M) #prediction
    {
      #transition Model
      sampling <- sample(1:3,1,replace = FALSE)
      if(sampling == 1) {
        ztBar[t,m] <- (rnorm(1, mean = ztBar[t-1,m],
                             sd = sqrt(sd_transition)))
      }else if(sampling == 2){
        ztBar[t,m] <- (rnorm(1, mean = ztBar[t-1,m] + 1,
                             sd = sqrt(sd_transition)))
      }else{
        ztBar[t,m] <- (rnorm(1, mean = ztBar[t-1,m] + 2,
                             sd = sqrt(sd_transition)))
      }

    }
    #Correction
    wt[t,] = rep((1/Tsteps),Tsteps)
    for(m in 1:M){
      ztBar[t,m] <- sample(ztBar[t,],1, prob = wt[t,], replace = TRUE)
    }
  }
  return(list(Z_est=ztBar, wt=wt , expected_location = rowSums(wt * ztBar)))
}
```

```
special_result = sepcial_particleFilter(X = special_true_result$observations,
                                         M = 100, Tsteps=100,
                                        sd_transition = 1,
                                        sd_emission = 1)
#plot results with sd_emission = 1 and sd_transition = 1

plot_result(result = special_result,
            true_values = special_true_result,
            heading = "SpecialCase with Weight = 1")
```



For the Special case i-e weights =1 which reflects no correction, it is obvious that the expected position of robot is completely different from the true position of robot, it makes sense as by keeping weights = 1, all particles have equal probability of being selected and thus there is no prediction based on former information.

Appendix

Question 1

```r
set.seed(54321)
library(ggplot2)

#emission model
emission_model<- function(z,standard_deviation){
  emission_value = (rnorm(1,mean = z,sd= standard_deviation) +
                      rnorm(1,mean = z - 1,sd= standard_deviation) +
                      rnorm(1,mean = z +1,sd= standard_deviation)
                      )/3
  return(emission_value)
}

transition_model<- function(z,standard_deviation)
{
  transition_value <- (rnorm(1,mean = z,sd = standard_deviation) +
                         rnorm(1,mean = z+1,sd = standard_deviation) +
                         rnorm(1,mean = z+2,sd = standard_deviation)

                         )/3
  return(transition_value)
}

#genrate a data from simuation
generate_data <- function(sd_emission,sd_transition,no_of_obs)
{
  #vector Initialization
  Z = rep(0,no_of_obs)
  X = rep(0,no_of_obs)

  #init first State uniformly
  Z[1] = runif(1,0,100)
  #intial X[1]
  X[1] = emission_model(Z[1],sd_emission)

  #generating obervations and states
  for (i in 2:no_of_obs) {
    Z[i] = transition_model(Z[i-1],sd_transition) #transition model
    X[i] = emission_model(Z[i],sd_emission) #emission model
  }

  return(data.frame(index=c(1:no_of_obs),"states" = Z,"observations"=X))
}



true_result_sd1 = generate_data(sd_emission = 1,sd_transition = 1,no_of_obs = 100)


#plot comparision of emission model and transition model
ggplot(data = true_result_sd1) +
  geom_line(aes(x=index,y=states , col = "states")) +
  geom_line(aes(x=index,y=observations , col = "observations")) +
  theme_minimal() +xlab("number of Time Stamps") +
  ylab("Emission & Transition") + ggtitle("simualtion with Sd = 1")


#particle Filter

particleFilter <- function(X, M, Tsteps, sd_transition,sd_emission)
{

  #Initialization
  ztBar <- matrix(0, nrow = 100, ncol = 100) #store transition model values
  ztBar[1,] = runif(100, 0, 100) #particles
  wt <- matrix(0, 100, 100) #particle weights
  normalize_weight <- matrix(0, 100, 100) #normalize weights

  #weights for first particle
  for (i in 1:Tsteps) {
    wt[1,i] <- (dnorm(X[i],ztBar[1,i] , sqrt(sd_emission)) +
                  dnorm(X[i],ztBar[1,i] - 1, sqrt(sd_emission)) +
                  dnorm(X[i],ztBar[1,i] + 1, sqrt(sd_emission))) / 3
  }
  #normalize weight for first particle
  normalize_weight[1,] = wt[1,] / sum(wt[1,])

  for(t in 2: (Tsteps))
  {
    for(m in 1:M) #prediction
    {
      #transition Model
      sampling <- sample(1:3,1,replace = FALSE)
      if(sampling == 1) {
        ztBar[t,m] <- (rnorm(1, mean = ztBar[t-1,m],
                             sd = sqrt(sd_transition)))
      }else if(sampling == 2){
        ztBar[t,m] <- (rnorm(1, mean = ztBar[t-1,m] + 1,
                             sd = sqrt(sd_transition)))
      }else{
        ztBar[t,m] <- (rnorm(1, mean = ztBar[t-1,m] + 2,
                             sd = sqrt(sd_transition)))
      }

      #Importance weight
      wt[t,m] <- (dnorm(X[t] , mean = ztBar[t,m],   sd = sqrt(sd_emission))+
                    dnorm(X[t] , mean = ztBar[t,m]-1, sd = sqrt(sd_emission))+
                   dnorm(X[t] , mean = ztBar[t,m]+1, sd = sqrt(sd_emission)))/3

    }
    #Correction
    for(m in 1:M){
      ztBar[t,m] <- sample(ztBar[t,],1, prob = wt[t,], replace = TRUE)
    }
    normalize_weight[t,] = wt[t,] / sum(wt[t,])

  }

  return(list(Z_est=ztBar, wt=wt , expected_location = rowSums(normalize_weight * ztBar)))
}
#plot function
plot_result <- function(result,true_values,heading)
{

  #for time = 1
  par(mfrow=c(1,1))
  plot(x = 1:300, y = rep(0,300),
       type = "l",
       col = "white",
       ylim = c(-0.6,1),
       xlab = "Position x",
       ylab = "",
       main = paste("Time : 1", heading))
  #points(true_result_sd1$states[1],col="green",cex = 1,pch = 19)
  abline(h = 0, col = "black")
  points(x = result$expected_location[1], y = 0.5, pch = "+",col = "blue")
  points(x = true_values$states[1], y = 0.5, col = "red")
  legend("topright", legend=c("Expected Position", "True Position"),
         col=c("blue", "red"), cex=0.8,pch = c("+","o"))


  #for time = 40

  plot(x = 1:300, y = rep(0,300),
       type = "l",
       col = "white",
       ylim = c(-0.6,1),
       xlab = "Position x",
       ylab = "",
       main = paste("Time : 40", heading))
  #points(true_result_sd1$states[1],col="green",cex = 1,pch = 19)
  abline(h = 0, col = "black")
  points(x = result$expected_location[40], y = 0.5, pch = "+",col = "blue")
  points(x = true_values$states[40], y = 0.5, col = "red")
  legend("topright", legend=c("Expected Position", "True Position"),
         col=c("blue", "red"), cex=0.8,pch = c("+","o"))


  #for time = 70

  plot(x = 1:300, y = rep(0,300),
       type = "l",
       col = "white",
       ylim = c(-0.6,1),
       xlab = "Position x",
       ylab = "",
       main = paste("Time : 70", heading))
  #points(true_result_sd1$states[1],col="green",cex = 1,pch = 19)
  abline(h = 0, col = "black")
  points(x = result$expected_location[70], y = 0.5, pch = "+",col = "blue")
  points(x = true_values$states[70], y = 0.5, col = "red")
  legend("topright", legend=c("Expected Position", "True Position"),
         col=c("blue", "red"), cex=0.8,pch = c("+","o"))


  #for time = 100

  plot(x = 1:300, y = rep(0,300),
       type = "l",
       col = "white",
       ylim = c(-0.6,1),
       xlab = "Position x",
       ylab = "",
       main = paste("Time : 100", heading))
  #points(true_result_sd1$states[1],col="green",cex = 1,pch = 19)
  abline(h = 0, col = "black")
  points(x = result$expected_location[100], y = 0.5, pch = "+",col = "blue")
  points(x = true_values$states[100], y = 0.5, col = "red")
  legend("topright", legend=c("Expected Position", "True Position"),
         col=c("blue", "red"), cex=0.8,pch = c("+","o"))

}

result_sd1 = particleFilter(X = true_result_sd1$observations, M = 100, Tsteps=100,
                            sd_transition = 1,sd_emission = 1)
#plot results with sd_emission = 1 and sd_transition = 1
plot_result(result = result_sd1, true_values = true_result_sd1,heading = "Sd = 1")
```

Question 2

```r
# question 2
true_result_sd5 = generate_data(sd_emission = 5,sd_transition = 1,no_of_obs = 100)


#plot comparision of emission model and transition model
ggplot(data = true_result_sd5) +
  geom_line(aes(x=index,y=states , col = "states")) +
  geom_line(aes(x=index,y=observations , col = "observations")) +
  theme_minimal() +xlab("number of Time Stamps") +
  ylab("Emission & Transition") + ggtitle("simualtion with Sd = 5")


result_sd5 = particleFilter(X = true_result_sd5$observations, M = 100, Tsteps=100,
                            sd_transition = 1,sd_emission = 5)
#plot results with sd_emission = 5 and sd_transition = 1
plot_result(result = result_sd5, true_values = true_result_sd5)
```

```r
true_result_sd50 = generate_data(sd_emission = 50,sd_transition = 1,no_of_obs = 100)


#plot comparision of emission model and transition model
ggplot(data = true_result_sd50) +
  geom_line(aes(x=index,y=states , col = "states")) +
  geom_line(aes(x=index,y=observations , col = "observations")) +
  theme_minimal() +xlab("number of Time Stamps") +
  ylab("Emission & Transition") + ggtitle("simualtion with Sd = 50")



result_sd50 = particleFilter(X = true_result_sd50$observations, M = 100, Tsteps=100,
                             sd_transition = 1,sd_emission = 50)
#plot results with sd_emission = 50 and sd_transition = 1
plot_result(result = result_sd50, true_values = true_result_sd50)
```

```r
#question 3 Sepcial case
sepcial_particleFilter <- function(X, M, Tsteps, sd_transition,sd_emission)
{

  #Initialization
  ztBar <- matrix(0, nrow = 100, ncol = 100) #store transition model values
  ztBar[1,] = runif(100, 0, 100)
  wt <- matrix(1, 100, 100) #particle weights

  #weights for first row
  # for (i in 1:Tsteps) {
  #   wt[1,i] <- (dnorm(X[i],ztBar[1,i] , sqrt(sd_emission)) +
  #                  dnorm(X[i],ztBar[1,i] - 1, sqrt(sd_emission)) +
  #                  dnorm(X[i],ztBar[1,i] + 1, sqrt(sd_emission))) / 3
  # }
  #
  #normalized Weight
  wt[1,] = rep((1/Tsteps),Tsteps)
  for(t in 2: (Tsteps))
  {
    for(m in 1:M) #prediction
    {
      #transition Model
      sampling <- sample(1:3,1,replace = FALSE)
      if(sampling == 1) {
        ztBar[t,m] <- (rnorm(1, mean = ztBar[t-1,m],
                             sd = sqrt(sd_transition)))
      }else if(sampling == 2){
        ztBar[t,m] <- (rnorm(1, mean = ztBar[t-1,m] + 1,
                             sd = sqrt(sd_transition)))
      }else{
        ztBar[t,m] <- (rnorm(1, mean = ztBar[t-1,m] + 2,
                             sd = sqrt(sd_transition)))
      }

      #Importance weight
      # wt[t,m] <- (dnorm(X[t] , mean = ztBar[t,m],   sd = sqrt(sd_emission))+
      #                dnorm(X[t] , mean = ztBar[t,m]-1, sd = sqrt(sd_emission))+
      #                dnorm(X[t] , mean = ztBar[t,m]+1, sd = sqrt(sd_emission)))/3
      #
    }
    #Correction
    #ztBar[t,]<- sample(ztBar[t,], M, replace = TRUE, prob = wt[t,])
    wt[t,] = rep((1/Tsteps),Tsteps)
    for(m in 1:M){
      ztBar[t,m] <- sample(ztBar[t,],1, prob = wt[t,], replace = TRUE)
    }



  }
  return(list(Z_est=ztBar, wt=wt , expected_location = rowSums(wt * ztBar)))
}


special_true_result_sd50 = generate_data(sd_emission = 50,sd_transition = 1,no_of_obs = 100)


# #plot comparision of emission model and transition model
# ggplot(data = special_true_result_sd50) +
#   geom_line(aes(x=index,y=states , col = "states")) +
#   geom_line(aes(x=index,y=observations , col = "observations")) +
#   theme_minimal() +xlab("number of Time Stamps") +
#   ylab("Emission & Transition") + ggtitle("simualtion with Sd = 1")



special_result = sepcial_particleFilter(X = special_true_result$observations,
                                         M = 100, Tsteps=100,
                                        sd_transition = 1,
                                        sd_emission = 50)
#plot results with sd_emission = 50 and sd_transition = 1
plot_result(result = special_result,
            true_values = special_true_result,
            heading = "SpecialCase with Weight = 1")
```

LAB 3: STATE SPACE MODELS

The purpose of the lab is to put in practice some of the concepts State Space Model by implementing the particle filter for robot localization.

Part 1

The robot moves along horizontally along the X-axis according to the following models:

Transition Model: $p(z_t|z_{t-1}) = (N(z_t|z_{t-1}, 1), N(z_t|z_{t-1} + 1, 1), N(z_t|z_{t-1} + 2, 1))/3$

Emission Model: $p(x_t|z_t) = (N(x_t|z_t, 1), N(x_t|z_t - 1, 1), N(x_t|z_t + 1, 1))/3$

Initial Model : $p(z_1) = Uniform(0, 100)$

Implementing the State Space Models

We will now implement the given state space models i-e Emission and Transition models uing the initial model

Emission Model

```r
#emission model
emission_model<- function(mean_v,sd_emission){
  sampling <- sample(1:3,1,replace = FALSE)
  if(sampling == 1) {
    emission_value <- (rnorm(1, mean = mean_v,
                             sd = sqrt(sd_emission)))
  }else if(sampling == 2){
    emission_value <- (rnorm(1, mean = mean_v - 1,
                             sd = sqrt(sd_emission)))
  }else{
    emission_value <- (rnorm(1, mean = mean_v + 1 ,
                             sd = sqrt(sd_emission)))
  }

  return(emission_value)
}
```

Transition Model

```r
transition_model<- function(mean_v,sd_transition)
{
  sampling <- sample(1:3,1,replace = FALSE)
  if(sampling == 1) {
    transition_value <- (rnorm(1, mean = mean_v,
                               sd = sqrt(sd_transition)))
  }else if(sampling == 2){
    transition_value <- (rnorm(1, mean = mean_v + 1,
                               sd = sqrt(sd_transition)))
  }else{
    transition_value <- (rnorm(1, mean = mean_v + 2,
                               sd = sqrt(sd_transition)))
  }

  return(transition_value)
}
```

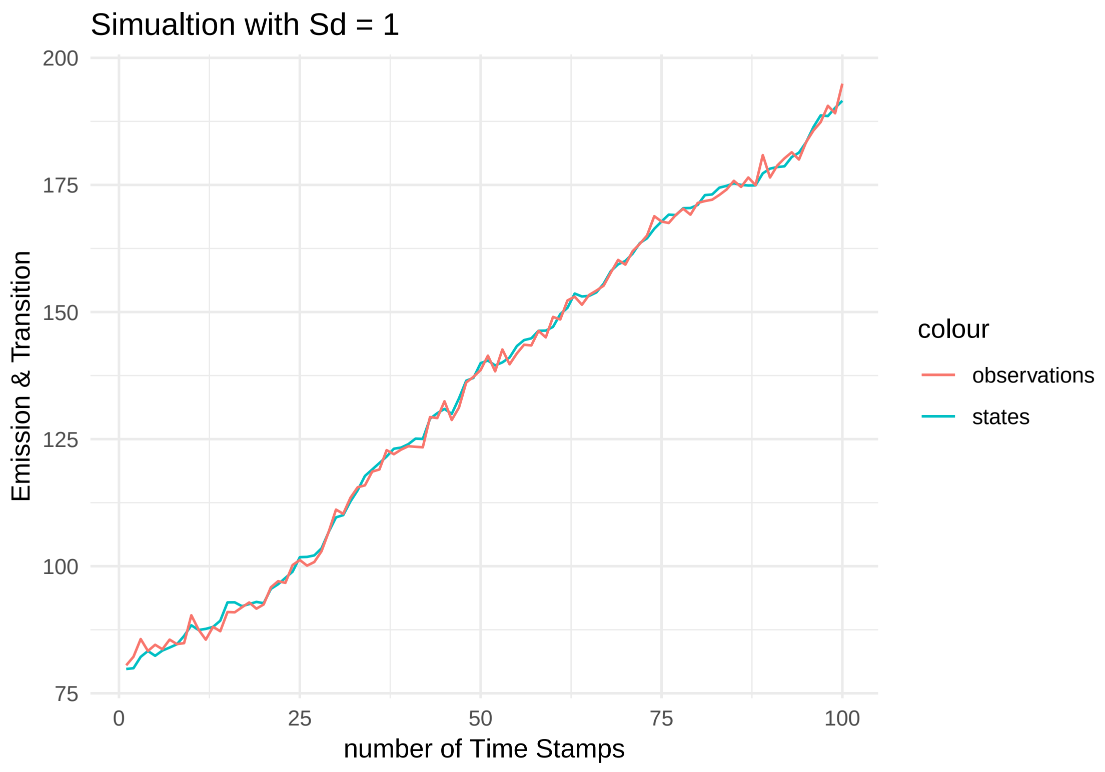
Simulating for T = 100 time steps with (Sd_Emission = 1 & Sd_Transition = 1)

We have to perform simulation in order to generate true states and observations from transition and emission model respectively.

```r
true_result_sd1 = generate_data(sd_emission = 1,sd_transition = 1,no_of_obs = 100)
```

When we simulate model for $T = 100$ time steps for given model, we obtain the $z_{1:100}$ which are states and $x_{1:100}$ which are observations.

```r
#plot comparision of emission model and transition model
ggplot(data = true_result_sd1) +
  geom_line(aes(x=index,y=states , col = "states")) +
  geom_line(aes(x=index,y=observations , col = "observations")) +
  theme_minimal() +xlab("number of Time Stamps") +
  ylab("Emission & Transition") + ggtitle("Simualtion with Sd = 1")
```



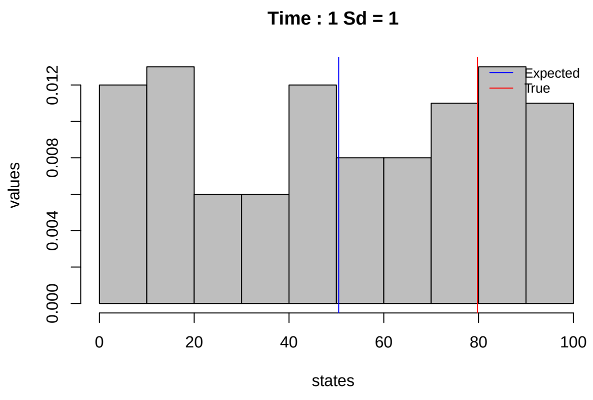
In the above graph X-Axis represents Time steps and Y-Axis represents values of states and observations. It can be observed that states and observation follow each other keeping Standard deviation of Emission and Transition = 1.

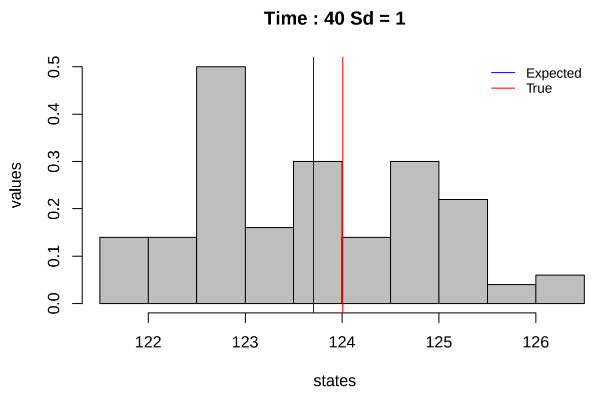
Localization of Robot

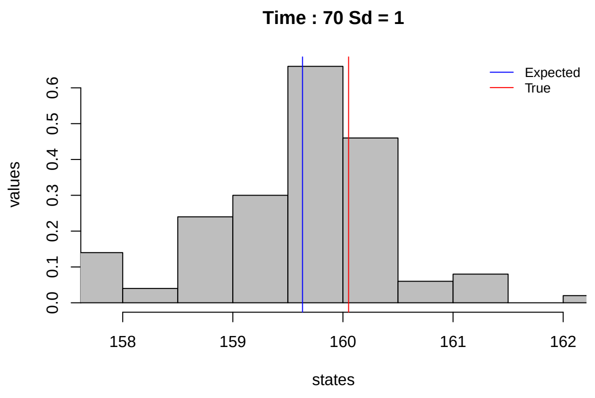
Implementation of particle Filter and comparision at different time steps

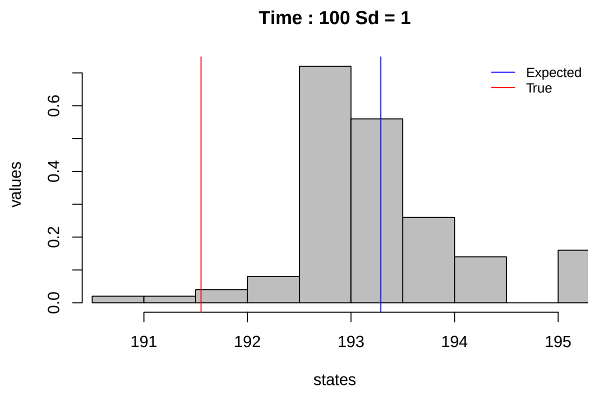
Now we have implemented the particle filter to determine the position of robot

```
result_sd1 = particleFilter(X = true_result_sd1$observations, M = 100, Tsteps=100,
                            sd_transition = 1,sd_emission = 1)
#plot results with sd_emission = 1 and sd_transition = 1

plot_histogram(result = result_sd1, true_values = true_result_sd1,heading = "Sd = 1")
```









Using the particle filter, we have determined the localization of robot. The graphs represent the true states (with red) and estimated states(in blue) of the robot and the histograms represent each particles.

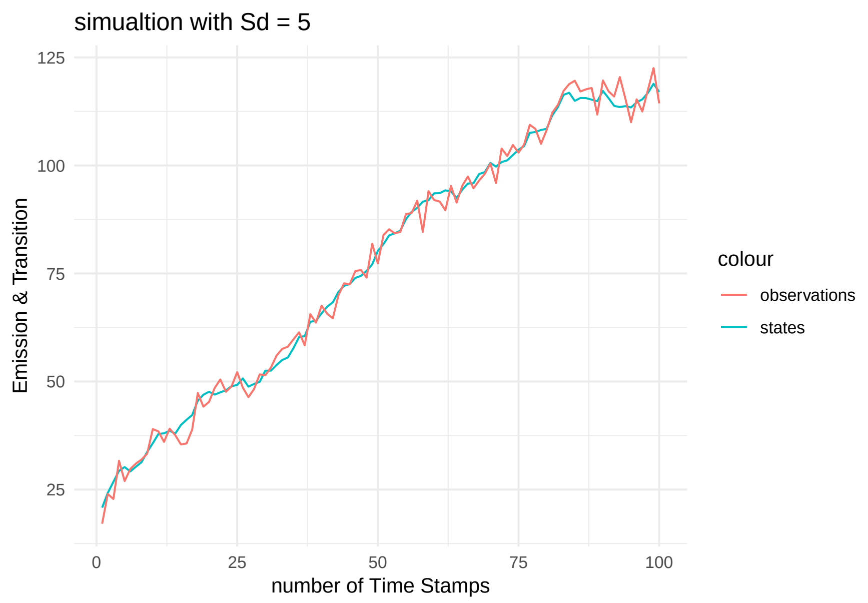
For sd = 1 for emission and transition, it can be seen that the expected position of robot is very close (almost same) to the true position of robot for all time steps except for the beginning i-e at time step 1.

Note: scale of X-axis is changing respective to points.

Part 2(a)

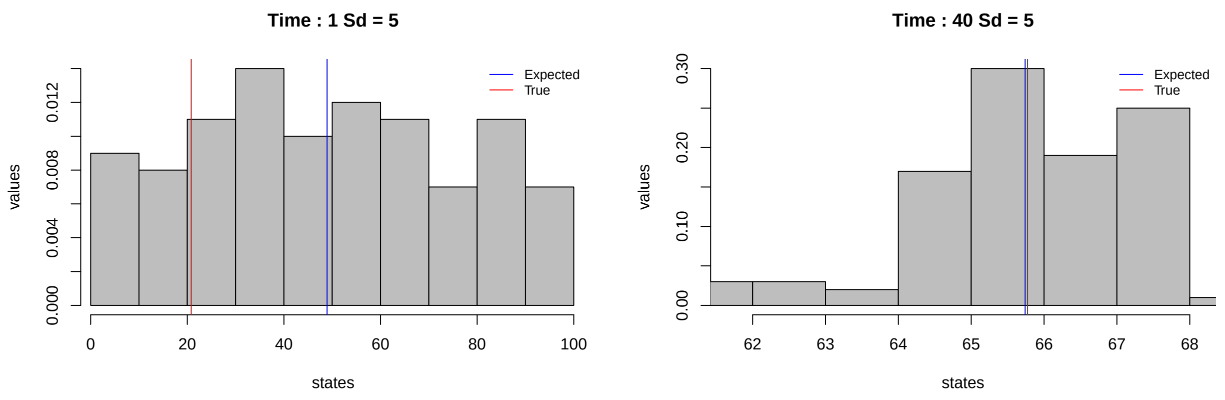
Simulating for T = 100 time with (Sd_Emission = 5 & Sd_Transition = 1)

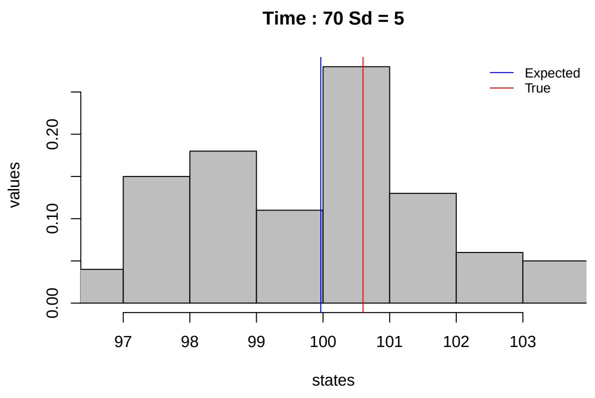
```r
true_result_sd5 = generate_data(sd_emission = 5,sd_transition = 1,no_of_obs = 100)
#plot comparision of emission model and transition model
ggplot(data = true_result_sd5) +
  geom_line(aes(x=index,y=states , col = "states")) +
  geom_line(aes(x=index,y=observations , col = "observations")) +
  theme_minimal() +xlab("number of Time Stamps") +
  ylab("Emission & Transition") + ggtitle("simualtion with Sd = 5")
```

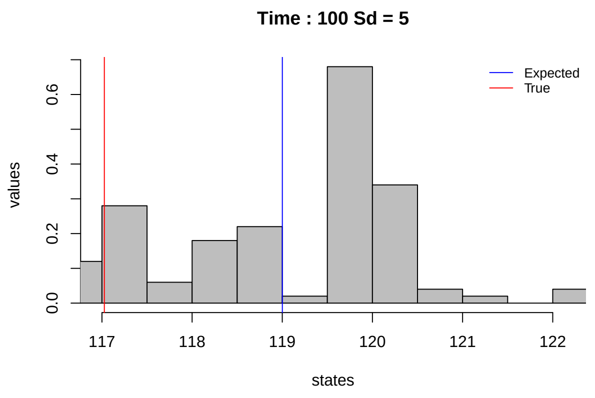


After setting standard deviation of emission model = 5, it can be seen that there is some variation between observations and states.

```
result_sd5 = particleFilter(X = true_result_sd5$observations, M = 100, Tsteps=100,
                            sd_transition = 1,sd_emission = 5)
#plot results with sd_emission = 5 and sd_transition = 1

plot_histogram(result = result_sd5, true_values = true_result_sd5,heading = "Sd = 5")
```





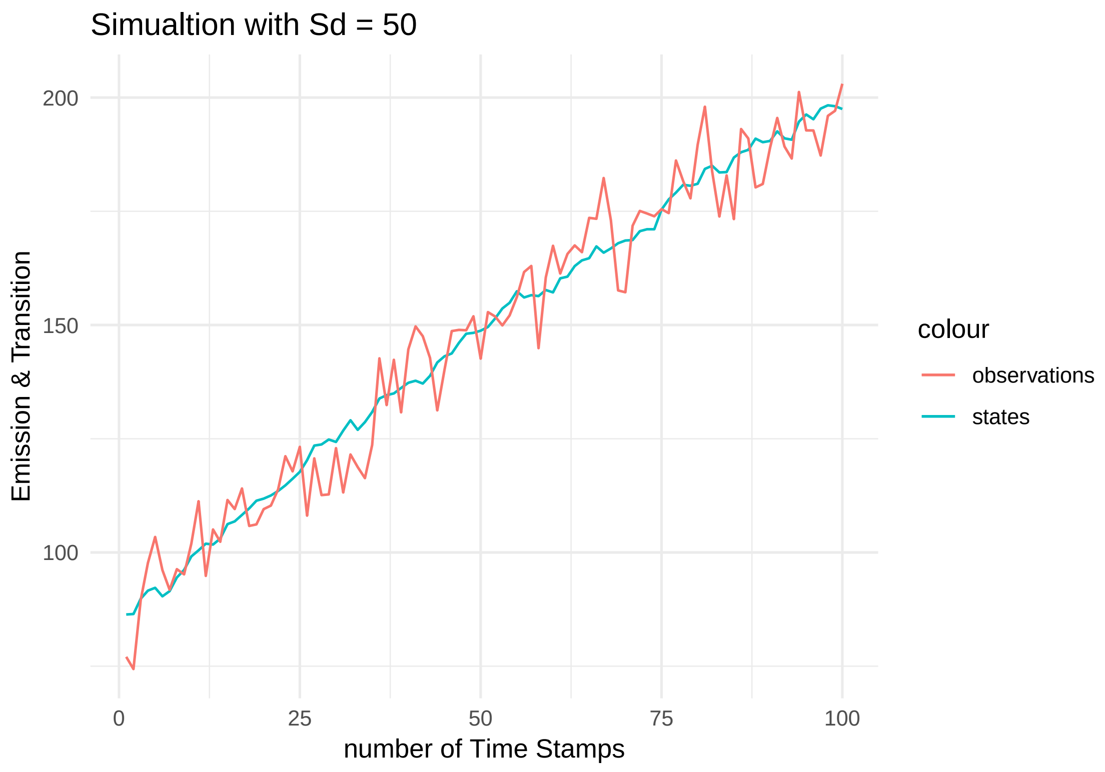


The expected position of robot is still very close to the true position but the uncertainty has increased for time step 100 as compared to previous time steps (i-e time step 440 and 70)

Part 2(b)

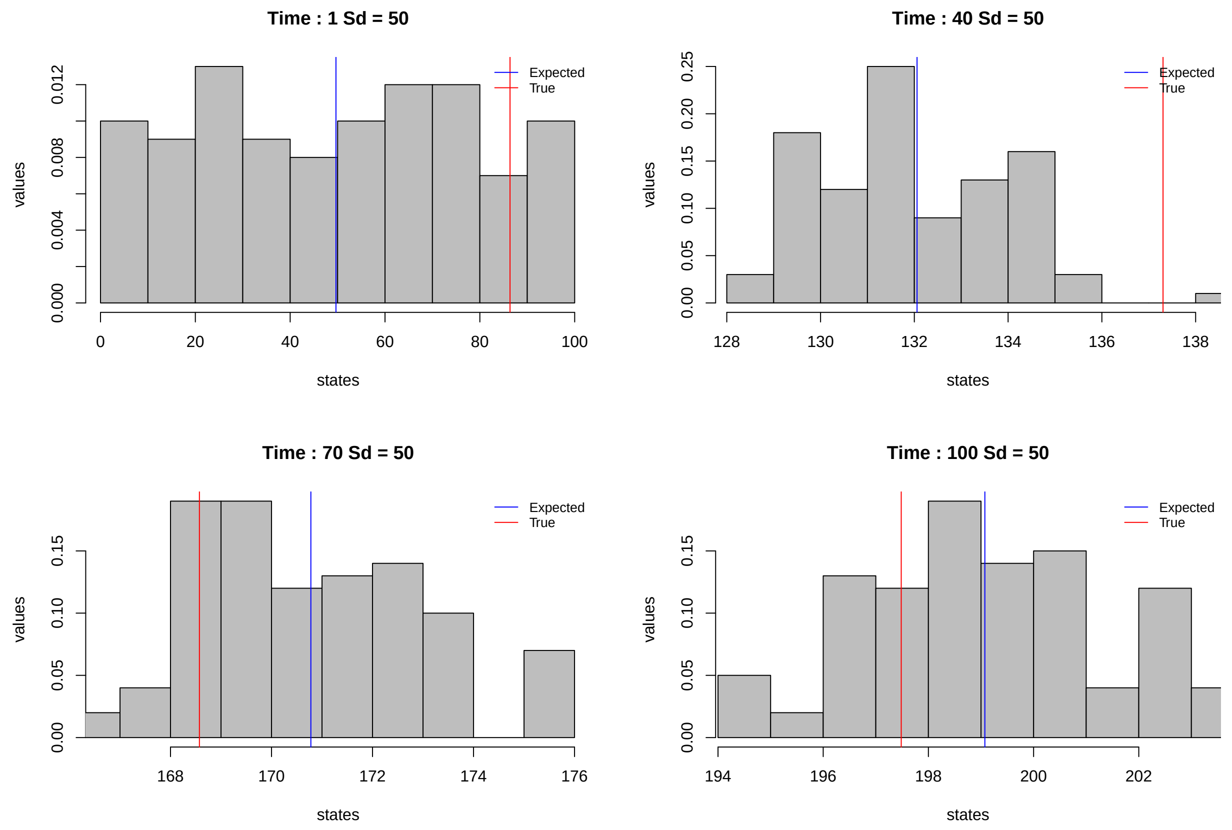
Simulating for T = 100 time with (Sd_Emission = 50 & Sd_Transition = 1)

```
true_result_sd50 = generate_data(sd_emission = 50,sd_transition = 1,no_of_obs = 100)

#plot comparision of emission model and transition model
ggplot(data = true_result_sd50) +
  geom_line(aes(x=index,y=states , col = "states")) +
  geom_line(aes(x=index,y=observations , col = "observations")) +
  theme_minimal() +xlab("number of Time Stamps") +
  ylab("Emission & Transition") + ggtitle("Simualtion with Sd = 50")
```



At standard deviation = 50, observations are fluctuating a lot and differs greatly from that of states which reflects the increase in uncertainty.

```r
result_sd50 = particleFilter(X = true_result_sd50$observations, M = 100,

#plot results with sd_emission = 50 and sd_transition = 1

plot_histogram(result = result_sd50, true_values = true_result_sd50,heading = "Sd = 50")
```



The expected position of robot in this case is not as accurate as in case of sd = 1 or 5, it can be seen that the distance between true and expected position has increased but the overall result is still not too bad. There is large uncertainty in the beginning.

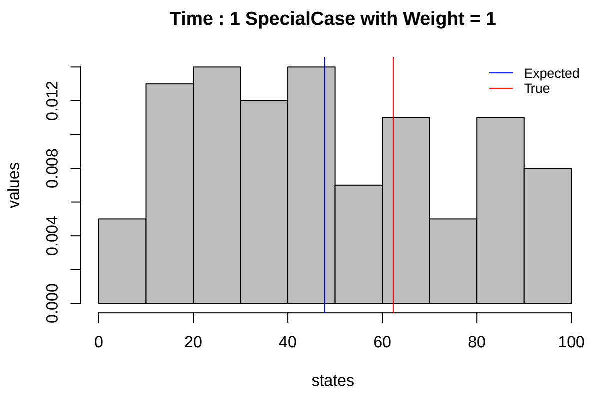
Part 3

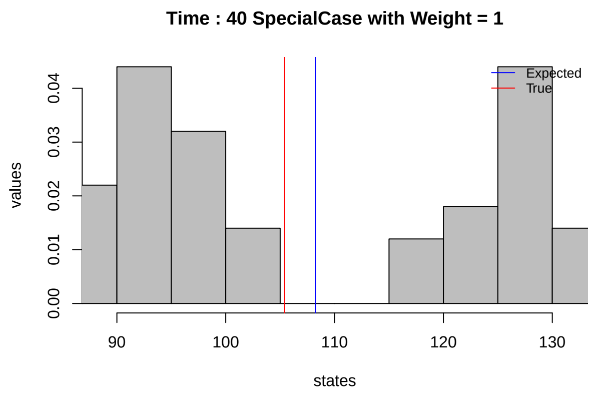
Special Case

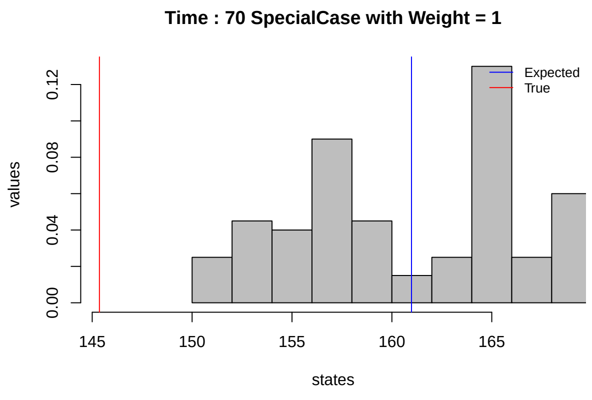
Here we have to keep weights = 1 for every time step

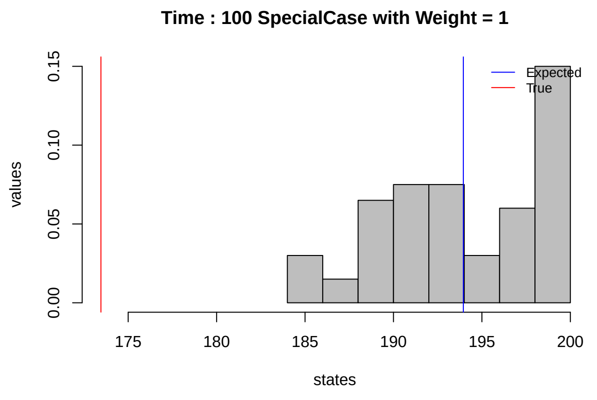
```r
sepcial_particleFilter <- function(X, M, Tsteps, sd_transition,sd_emission)
{

  #Initialization
  ztBar <- matrix(0, nrow = 100, ncol = 100) #store transition model values
  ztBar[1,] = runif(100, 0, 100)
  wt <- matrix(1, 100, 100) #particle weights

  wt[1,] = rep((1/Tsteps),Tsteps)
  for(t in 2: (Tsteps))
  {
    for(m in 1:M) #prediction
    {
      #transition Model
      sampling <- sample(1:3,1,replace = FALSE)
      if(sampling == 1) {
        ztBar[t,m] <- (rnorm(1, mean = ztBar[t-1,m],
                            sd = sqrt(sd_transition)))
      }else if(sampling == 2){
        ztBar[t,m] <- (rnorm(1, mean = ztBar[t-1,m] + 1,
                            sd = sqrt(sd_transition)))
      }else{
        ztBar[t,m] <- (rnorm(1, mean = ztBar[t-1,m] + 2,
                            sd = sqrt(sd_transition)))
      }

    }
    #Correction
    wt[t,] = rep((1/Tsteps),Tsteps)
    for(m in 1:M){
      ztBar[t,m] <- sample(ztBar[t,],1, prob = wt[t,], replace = TRUE)
    }



  }
  return(list(Z_est=ztBar, wt=wt , expected_location = rowSums(wt * ztBar)))
}
```

```
special_result = sepcial_particleFilter(X = special_true_result$observations,
                                        M = 100, Tsteps=100,
                                       sd_transition = 1,
                                       sd_emission = 1)

#plot results with sd_emission = 1 and sd_transition = 1

plot_histogram(result = special_result,
            true_values = special_true_result,
            heading = "SpecialCase with Weight = 1")
```









For the Special case i-e weights =1 which reflects no correction, it is obvious that the expected position of robot is completely different from the true position of robot, in most cases there is no particle corresponding to the true state. It makes sense as by keeping weights = 1, all particles have equal probability of being selected and thus there is no prediction based on former information.

Appendix

Question 1

```r
set.seed(54321)
library(ggplot2)

emission_model<- function(mean_v,sd_emission){
  sampling <- sample(1:3,1,replace = FALSE)
  if(sampling == 1) {
    emission_value <- (rnorm(1, mean = mean_v,
                          sd = sqrt(sd_emission)))
  }else if(sampling == 2){
    emission_value <- (rnorm(1, mean = mean_v - 1,
                          sd = sqrt(sd_emission)))
  }else{
    emission_value <- (rnorm(1, mean = mean_v + 1 ,
                          sd = sqrt(sd_emission)))
  }

  return(emission_value)
}

transition_model<- function(mean_v,sd_transition)
{
  sampling <- sample(1:3,1,replace = FALSE)
  if(sampling == 1) {
    transition_value <- (rnorm(1, mean = mean_v,
                            sd = sqrt(sd_transition)))
  }else if(sampling == 2){
    transition_value <- (rnorm(1, mean = mean_v + 1,
                            sd = sqrt(sd_transition)))
  }else{
    transition_value <- (rnorm(1, mean = mean_v + 2,
                            sd = sqrt(sd_transition)))
  }

  return(transition_value)
}

#genrate a data from simuation
generate_data <- function(sd_emission,sd_transition,no_of_obs)
{
  #vector Initialization
  Z = rep(0,no_of_obs)
  X = rep(0,no_of_obs)

  #init first State uniformly
  Z[1] = runif(1,0,100)
  #intial X[1]
  X[1] = emission_model(Z[1],sd_emission)

  #generating obervations and states
  for (i in 2:no_of_obs) {
    Z[i] = transition_model(Z[i-1],sd_transition) #transition model
    X[i] = emission_model(Z[i],sd_emission) #emission model
  }

  return(data.frame(index=c(1:no_of_obs),"states" = Z,"observations"=X))
}



true_result_sd1 = generate_data(sd_emission = 1,sd_transition = 1,no_of_obs = 100)


#plot comparision of emission model and transition model
ggplot(data = true_result_sd1) +
  geom_line(aes(x=index,y=states , col = "states")) +
  geom_line(aes(x=index,y=observations , col = "observations")) +
  theme_minimal() +xlab("number of Time Stamps") +
  ylab("Emission & Transition") + ggtitle("simualtion with Sd = 1")


#particle Filter

particleFilter <- function(X, M, Tsteps, sd_transition,sd_emission)
{

  #Initialization
  ztBar <- matrix(0, nrow = 100, ncol = 100) #store transition model values
  ztBar[1,] = runif(100, 0, 100) #particles
  wt <- matrix(0, 100, 100) #particle weights
  normalize_weight <- matrix(0, 100, 100) #normalize weights

  #weights for first particle
  for (i in 1:Tsteps) {
    wt[1,i] <- (dnorm(X[i],ztBar[1,i] , sqrt(sd_emission)) +
                 dnorm(X[i],ztBar[1,i] - 1, sqrt(sd_emission)) +
                 dnorm(X[i],ztBar[1,i] + 1, sqrt(sd_emission))) / 3
  }
  #normalize weight for first particle
  normalize_weight[1,] = wt[1,] / sum(wt[1,])

  for(t in 2: (Tsteps))
  {
    for(m in 1:M) #prediction
    {
      #transition Model
      sampling <- sample(1:3,1,replace = FALSE)
      if(sampling == 1) {
        ztBar[t,m] <- (rnorm(1, mean = ztBar[t-1,m],
                             sd = sqrt(sd_transition)))
      }else if(sampling == 2){
        ztBar[t,m] <- (rnorm(1, mean = ztBar[t-1,m] + 1,
                             sd = sqrt(sd_transition)))
      }else{
        ztBar[t,m] <- (rnorm(1, mean = ztBar[t-1,m] + 2,
                             sd = sqrt(sd_transition)))
      }

      #Importance weight
      wt[t,m] <- (dnorm(X[t] , mean = ztBar[t,m],   sd = sqrt(sd_emission))+
                     dnorm(X[t] , mean = ztBar[t,m]-1, sd = sqrt(sd_emission))+
                    dnorm(X[t] , mean = ztBar[t,m]+1, sd = sqrt(sd_emission)))/3

    }
    #Correction
    for(m in 1:M){
      ztBar[t,m] <- sample(ztBar[t,],1, prob = wt[t,], replace = TRUE)
    }
    normalize_weight[t,] = wt[t,] / sum(wt[t,])

  }

  return(list(Z_est=ztBar, wt=wt , expected_location = rowSums(normalize_weight * ztBar)))
}

plot_histogram <- function(result,true_values,heading)
{
  hist(result$Z_est[1,], col="grey",
       border="black",prob = TRUE,
       xlab = "states",main = paste("Time : 1",heading),
       ylab = "values",
       xlim = c(min(min(true_values$states[1],result$Z_est[1,])),
       max(max(true_values$states[1],result$Z_est[1,]))))
  abline(v=true_values$states[1], col="red")
  abline(v=mean(result$Z_est[1,]), col="blue")
#  abline(v=result$expected_location[1],col= "green")
  legend("topright", legend=c("Expected", "True"),
         col=c("blue", "red"), cex=0.8,lty = 1,box.lty=0,bty = "n")



  hist(result$Z_est[40,], col="grey",
       border="black",prob = TRUE,
       xlab = "states",main = paste("Time : 40",heading),
       ylab = "values",
       xlim = c(min(min(true_values$states[40],result$Z_est[40,])),
       max(max(true_values$states[40],result$Z_est[40,]))))
  abline(v=true_values$states[40], col="red")
  abline(v=mean(result$Z_est[40,]), col="blue")
#  abline(v=result$expected_location[40],col= "green")
  legend("topright", legend=c("Expected", "True"),
         col=c("blue", "red"), cex=0.8,lty = 1,box.lty=0,bty = "n")





  hist(result$Z_est[70,], col="grey",
       border="black",prob = TRUE,
       xlab = "states",main = paste("Time : 70",heading),
       ylab = "values",
       xlim = c(min(min(true_values$states[70],result$Z_est[70,])),
       max(max(true_values$states[70],result$Z_est[70,]))))
  abline(v=true_values$states[70], col="red")
  abline(v=mean(result$Z_est[70,]), col="blue")
 # abline(v=result$expected_location[70],col= "green")
  legend("topright", legend=c("Expected", "True"),
         col=c("blue", "red"), cex=0.8,lty = 1,box.lty=0,bty = "n")


  hist(result$Z_est[100,], col="grey",
       border="black",prob = TRUE,
       xlab = "states",main = paste("Time : 100",heading),
       ylab = "values",
       xlim = c(min(min(true_values$states[100],result$Z_est[100,])),
       max(max(true_values$states[100],result$Z_est[100,]))))
  abline(v=true_values$states[100], col="red")
  abline(v=mean(result$Z_est[100,]), col="blue")
#  abline(v=result$expected_location[100],col= "green")
  legend("topright", legend=c("Expected", "True"),
         col=c("blue", "red"), cex=0.8,lty = 1,box.lty=0,bty = "n")
}
#plot function
# plot_result <- function(result,true_values,heading)
# {
#
#   #for time = 1
#   par(mfrow=c(1,1))
#   plot(x = 1:300, y = rep(0,300),
#        type = "l",
#        col = "white",
#        ylim = c(-0.6,1),
#        xlab = "Position x",
#        ylab = "",
#        main = paste("Time : 1", heading))
#   #points(true_result_sd1$states[1],col="green",cex = 1,pch = 19)
#   abline(h = 0, col = "black")
#   points(x = result$expected_location[1], y = 0.5, pch = "+",col = "blue")
#   points(x = true_values$states[1], y = 0.5, col = "red")
#   legend("topright", legend=c("Expected Position", "True Position"),
#          col=c("blue", "red"), cex=0.8,pch = c("+","o"))
#
#
#   #for time = 40
#
#   plot(x = 1:300, y = rep(0,300),
#        type = "l",
#        col = "white",
#        ylim = c(-0.6,1),
#        xlab = "Position x",
#        ylab = "",
```

```r
#        main = paste("Time : 40", heading))
#    #points(true_result_sd1$states[1],col="green",cex = 1,pch = 19)
#    abline(h = 0, col = "black")
#    points(x = result$expected_location[40], y = 0.5, pch = "+",col = "blue")
#    points(x = true_values$states[40], y = 0.5, col = "red")
#    legend("topright", legend=c("Expected Position", "True Position"),
#           col=c("blue", "red"), cex=0.8,pch = c("+","o"))
#
#
#    #for time = 70
#
#    plot(x = 1:300, y = rep(0,300),
#        type = "l",
#        col = "white",
#        ylim = c(-0.6,1),
#        xlab = "Position x",
#        ylab = "",
#        main = paste("Time : 70", heading))
#    #points(true_result_sd1$states[1],col="green",cex = 1,pch = 19)
#    abline(h = 0, col = "black")
#    points(x = result$expected_location[70], y = 0.5, pch = "+",col = "blue")
#    points(x = true_values$states[70], y = 0.5, col = "red")
#    legend("topright", legend=c("Expected Position", "True Position"),
#           col=c("blue", "red"), cex=0.8,pch = c("+","o"))
#
#
#    #for time = 100
#
#    plot(x = 1:300, y = rep(0,300),
#        type = "l",
#        col = "white",
#        ylim = c(-0.6,1),
#        xlab = "Position x",
#        ylab = "",
#        main = paste("Time : 100", heading))
#    #points(true_result_sd1$states[1],col="green",cex = 1,pch = 19)
#    abline(h = 0, col = "black")
#    points(x = result$expected_location[100], y = 0.5, pch = "+",col = "blue")
#    points(x = true_values$states[100], y = 0.5, col = "red")
#    legend("topright", legend=c("Expected Position", "True Position"),
#           col=c("blue", "red"), cex=0.8,pch = c("+","o"))
#
# }
#
# result_sd1 = particleFilter(X = true_result_sd1$observations, M = 100, Tsteps=100, sd_transition = 1,
# #plot results with sd_emission = 1 and sd_transition = 1
plot_histogram(result = result_sd1, true_values = true_result_sd1,heading = "Sd = 1")
```

Question 2

```r
# question 2
true_result_sd5 = generate_data(sd_emission = 5,sd_transition = 1,no_of_obs = 100)
```

```r
#plot comparision of emission model and transition model
ggplot(data = true_result_sd5) +
  geom_line(aes(x=index,y=states , col = "states")) +
  geom_line(aes(x=index,y=observations , col = "observations")) +
  theme_minimal() +xlab("number of Time Stamps") +
  ylab("Emission & Transition") + ggtitle("simualtion with Sd = 5")


result_sd5 = particleFilter(X = true_result_sd5$observations, M = 100, Tsteps=100,
                            sd_transition = 1,sd_emission = 5)
#plot results with sd_emission = 5 and sd_transition = 1
plot_result(result = result_sd5, true_values = true_result_sd5)




true_result_sd50 = generate_data(sd_emission = 50,sd_transition = 1,no_of_obs = 100)


#plot comparision of emission model and transition model
ggplot(data = true_result_sd50) +
  geom_line(aes(x=index,y=states , col = "states")) +
  geom_line(aes(x=index,y=observations , col = "observations")) +
  theme_minimal() +xlab("number of Time Stamps") +
  ylab("Emission & Transition") + ggtitle("simualtion with Sd = 50")



result_sd50 = particleFilter(X = true_result_sd50$observations, M = 100, Tsteps=100,
                             sd_transition = 1,sd_emission = 50)
#plot results with sd_emission = 50 and sd_transition = 1
plot_histogram(result = result_sd50, true_values = true_result_sd50)
```

Question 3

```r
#question 3 Sepcial case
sepcial_particleFilter <- function(X, M, Tsteps, sd_transition,sd_emission)
{

  #Initialization
  ztBar <- matrix(0, nrow = 100, ncol = 100) #store transition model values
  ztBar[1,] = runif(100, 0, 100)
  wt <- matrix(1, 100, 100) #particle weights

  #normalized Weight
  wt[1,] = rep((1/Tsteps),Tsteps)
  for(t in 2: (Tsteps))
  {
    for(m in 1:M) #prediction
    {
      #transition Model
      sampling <- sample(1:3,1,replace = FALSE)
      if(sampling == 1) {
        ztBar[t,m] <- (rnorm(1, mean = ztBar[t-1,m],
                             sd = sqrt(sd_transition)))
      }else if(sampling == 2){
        ztBar[t,m] <- (rnorm(1, mean = ztBar[t-1,m] + 1,
                             sd = sqrt(sd_transition)))
      }else{
        ztBar[t,m] <- (rnorm(1, mean = ztBar[t-1,m] + 2,
                             sd = sqrt(sd_transition)))
      }

   }
    #Correction
    #ztBar[t,]<- sample(ztBar[t,], M, replace = TRUE, prob = wt[t,])
    #wt[t,] = rep((1/Tsteps),Tsteps)
    for(m in 1:M){
      ztBar[t,m] <- sample(ztBar[t,],1, replace = TRUE)
    }



  }
  return(list(Z_est=ztBar, wt=wt , expected_location = rowSums(wt * ztBar)))
}


special_true_result_sd50 = generate_data(sd_emission = 50,sd_transition = 1,no_of_obs = 100)


special_result = sepcial_particleFilter(X = special_true_result$observations,
                                         M = 100, Tsteps=100,
                                        sd_transition = 1,
                                        sd_emission = 50)
#plot results with sd_emission = 50 and sd_transition = 1
plot_histogram(result = special_result,
            true_values = special_true_result,
            heading = "SpecialCase with Weight = 1")
```

LAb 4 Gp Regression

Question 1: Implementing GP Regression

This task is to implement GP Regression algorithm in Rasmussen and Williams's book (page 19). Given Gaussian process regression model:

$$y = f(x) = \epsilon$$

Where,

$$\epsilon \sim N(0, \sigma_n^2)$$

$$f \sim GP(0, k(x, x'))$$

$k(x, x')$ is squared exponential kernel

1.1

Codes for function *posteriorGP*, which returns a vector with posterior mean and variance of f is below:

```r
##1
sqr_exp_kernel <- function(sigmaf, ell)
{
  rval <- function(x, y) {
    n1 <- length(x)
    n2 <- length(y)
    K <- matrix(NA, n1, n2)
    for (i in 1:n2){
      r = abs(x-y[i]);
      K[,i] <- sigmaf^2*exp(-0.5*( (r)/ell)^2 )
    }

    return(K)
  }
  class(rval) <- "kernel"
  return(rval)
}

#algorithm 2.1 on page 19
# x = inputs
# y = Target
# k = covariance function
# sigmaSquareN = noise level
# xStar = test input
posteriorGP <- function(x,y,k,sigmaNoise,xStar)
{
  n = length(x)
  #k(x,x)
  k_ = k(x , x)
  #k(x,x*)
  kStar = k(x,xStar)
  #k(x*,x*)
  kStarStar = k(xStar,xStar)

  I =  diag(dim(k_)[2] )
  L <- t(chol(k_ + (sigmaNoise^2) * I))
  alpha <- solve(t(L), solve(L, y))

  fStarMean = t(kStar) %*% alpha
  v <- solve(L, kStar)
  v_fstar = kStarStar - t(v) %*% v

  #temprary
  log_prob = -0.5 %*% t(y) %*% alpha - sum( log(diag(L)) - n/2 * log(2*pi) )

  return(list(mean=  fStarMean , var = v_fstar, marginal_likehood = log_prob ))
}
```

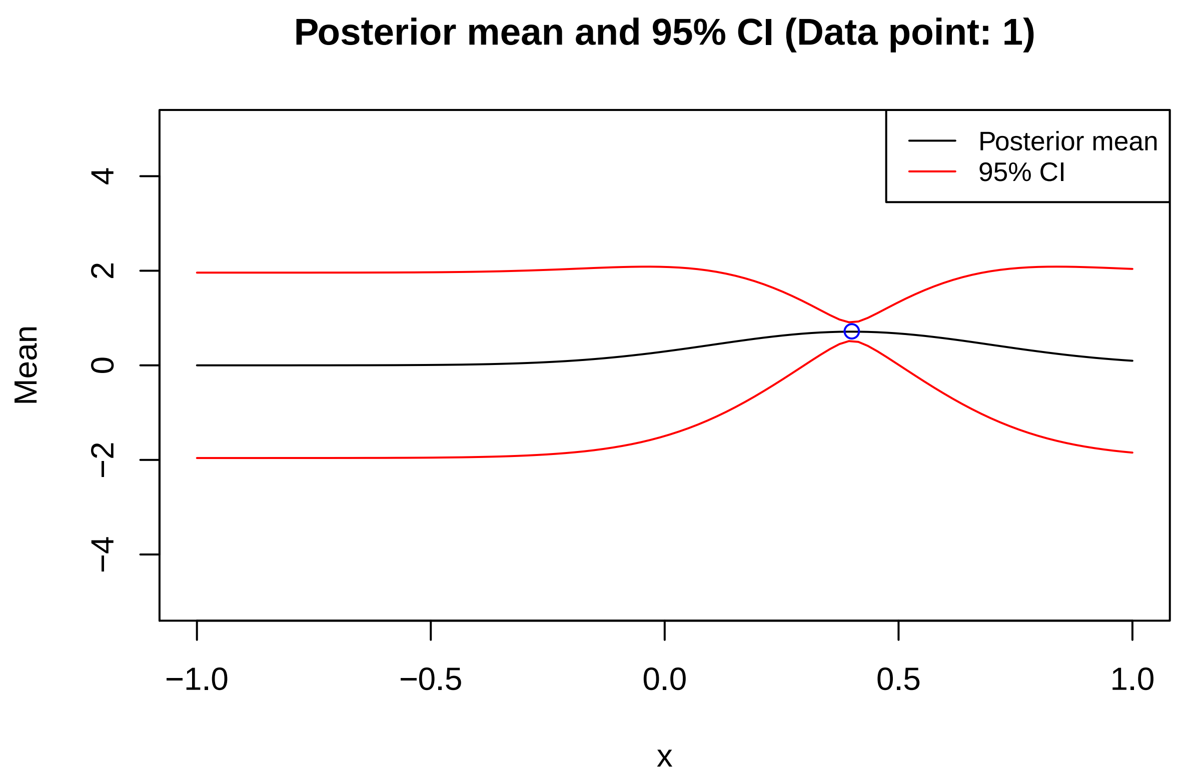
1.2

The function *posteriorGP* is implemented with the following setting:

$x = 0.4$; $y = 0.719$; $\sigma_f = 1$; $l = 0.3$; $\sigma_n = 0.1$; and $x \in [-1, 1]$

```r
##2
sigmaF = 1
ell = 0.3
x <- 0.4
y = 0.719
sigmaNoise = 0.1
xStar <- seq(-1, 1, length = 100)
SE_Func <- sqr_exp_kernel(sigmaf =sigmaF, ell = ell ) #par = sigmaf, ell

GP1 <- posteriorGP(x=x, y=y,
                   k=SE_Func,
                   sigmaNoise=sigmaNoise,
                   xStar=xStar)
#95%CI
lowerBound1 <- GP1$mean - 1.96*sqrt(diag(GP1$var))
upperBound1 <- GP1$mean + 1.96*sqrt(diag(GP1$var))
```

Posterior mean is plotted over the x interval, with 95% probability (pointwise) bands.

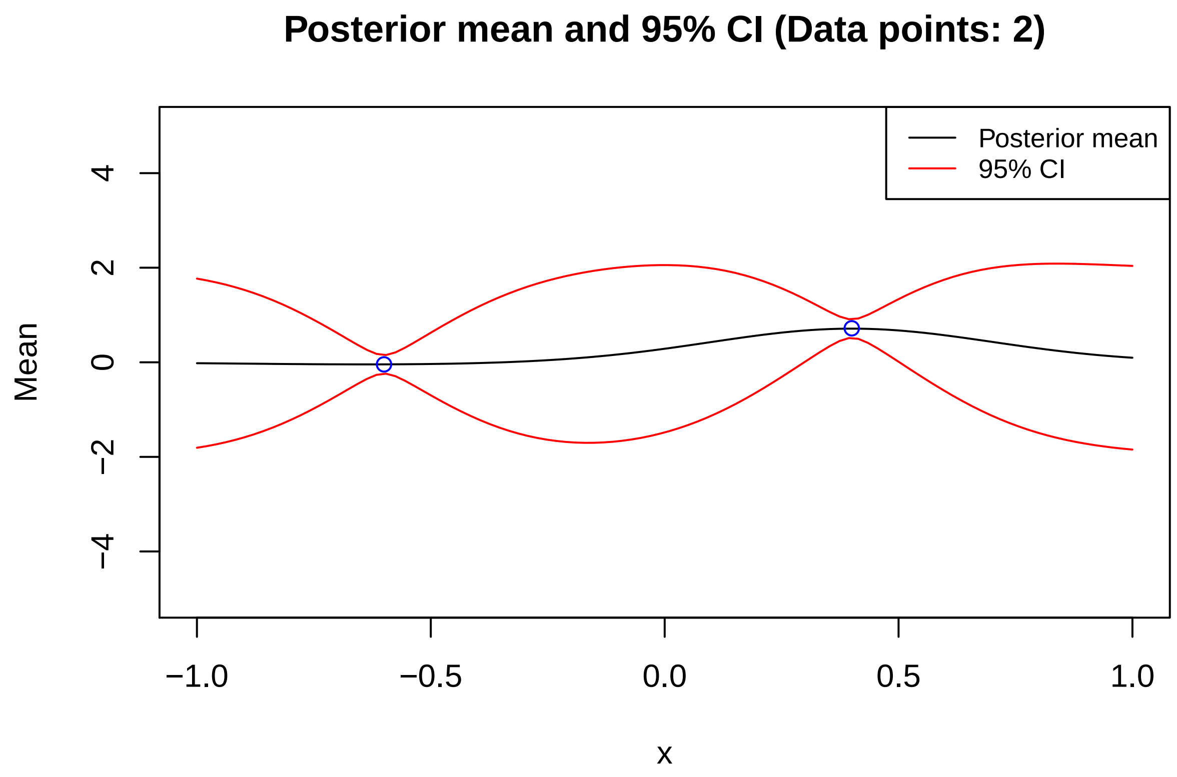


1.3

The function *posteriorGP* is implemented with the following updated setting:

$x = (-0.6, 0.4)$; $y = (-0.044, 0.719)$; $\sigma_f = 1$; $l = 0.3$; $\sigma_n = 0.1$; and $x \in [-1, 1]$

Similarly, posterior mean is plotted over the x interval, with 95% probability (pointwise) bands.

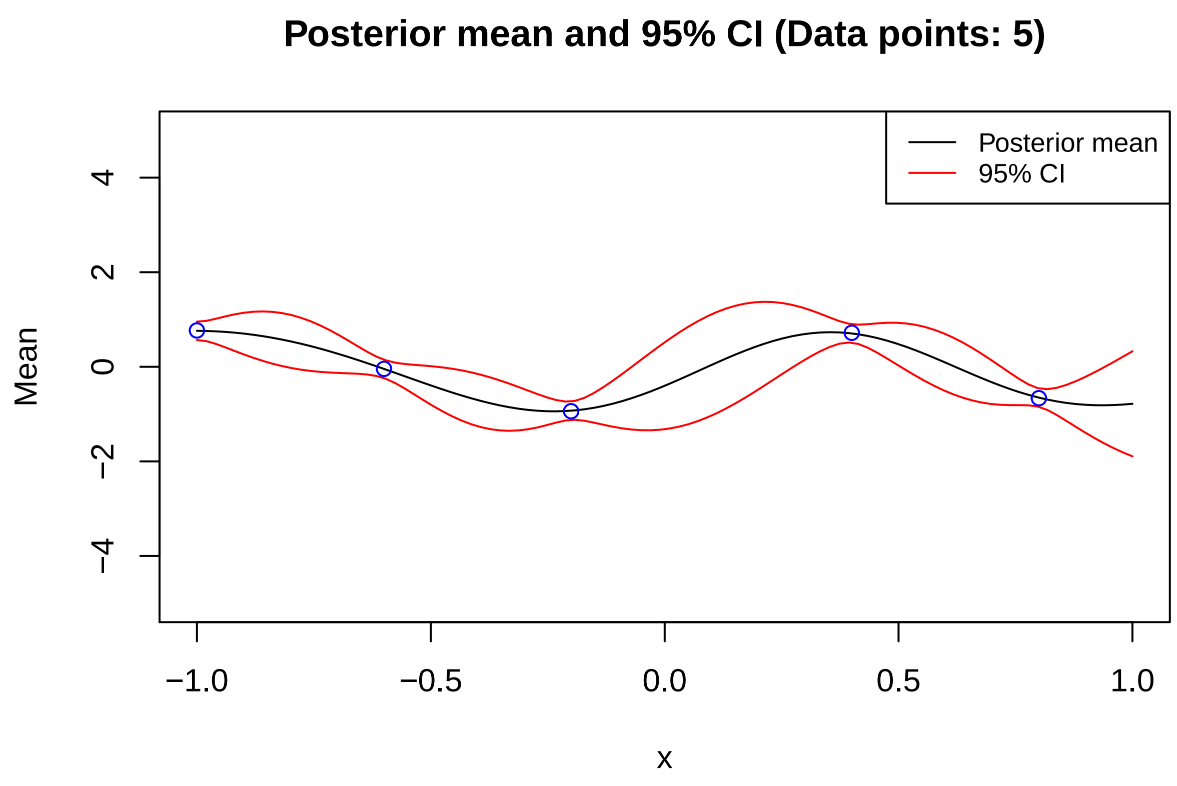


1.4

The function *posteriorGP* is again implemented with the following updated setting:

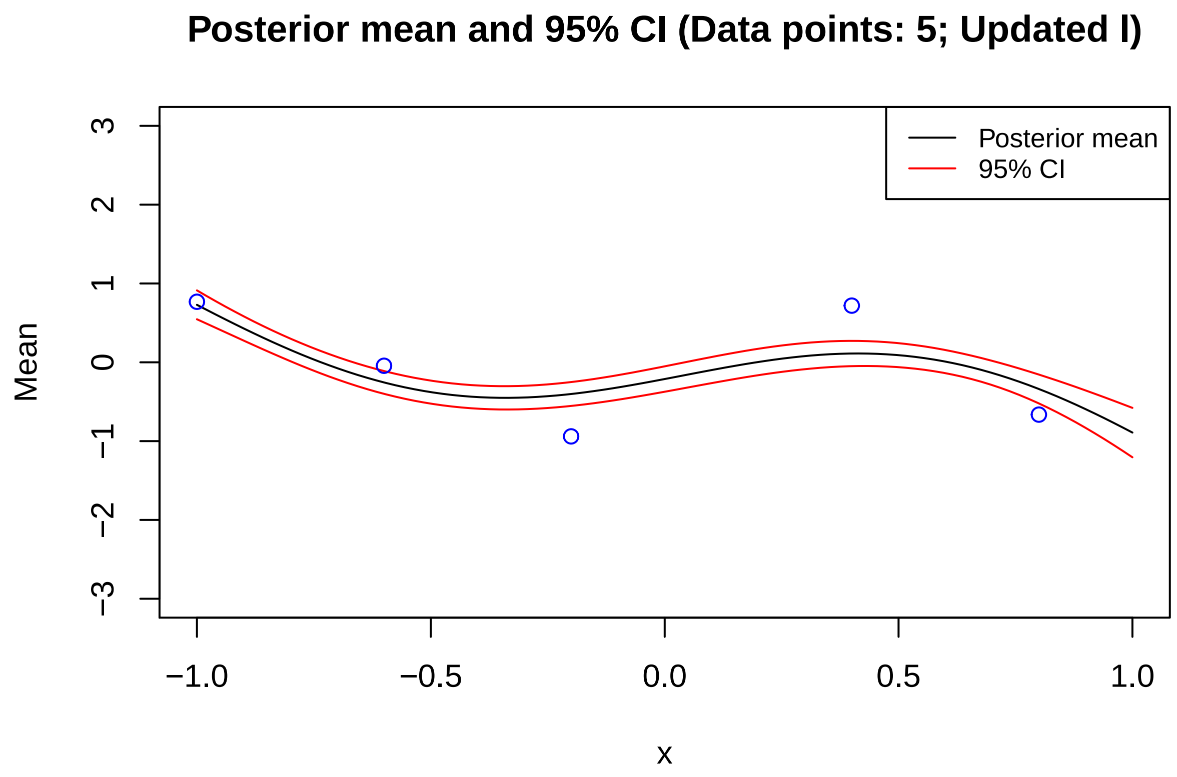
$x = (-1.0, -0.6, -0.2, 0.4, 0.8)$; $y = (0.768, -0.044, 0.940, 0.719, -0.664)$; $\sigma_f = 1$; $l = 0.3$; $\sigma_n = 0.1$; and $x \in [-1, 1]$

In the same way, plot of posterior mean over the x interval, with 95% probability (pointwise) bands shown below.



1.5

Likewise, the function *posteriorGP* is implemented with similar setting as 1.4, but updated $l = 1$.



Comparision

It can been seen from the above graph that, the larger the value of $l \;\; i.e \;\; ell$ the smoother the line we get as shown in the graph. Since the parameter $l \;\; i.e \;\; ell$ controls the smoothnes of the curves as it is smoother hyperparamter.

Question 2: GP Regression with kernlab

In this question, dataset of daily mean temperature in Stockholm (Tullinge) is used

```
library(kernlab)
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.c
knitr::kable(head(data))
```

date	temp
01/01/10	-8.6
02/01/10	-11.3
03/01/10	-11.5
04/01/10	-11.2
05/01/10	-16.4
06/01/10	-15.4

To make the algorithm works faster, we are ask to subsample the data and use only every fifth observation.

```r
#generate a time variable
time = seq(1,2190,1)

#for day
day = rep(seq(1,365,1),6)

#generate a every fifth value
every_fifth_time = time[seq(1, length(time), 5)]
every_fifth_day = day[seq(1, length(day), 5)]


#adding a time and day to original dataFrame
data$day = day
data$time = time

#every fifth data exract from dataFrame on basis of day
every_fifth_df = data[data$day %in% every_fifth_day,]
```

2.1

First, instead of define a *square exponential kernel function* again, we use the *sqrtExpKernel* in Question 1. Then use the *kernelMatrix* function to compute the covariance matrix for the input vectors $X = (1, 3, 4)^T$ and $X_* = (2, 3, 4)\ T$.

```r
ell = 1 #x
sigmaF = 2 #x'

X <- matrix(c(1,3,4),ncol = 1)
Xstar <- matrix(c(2,3,4),ncol = 1)

SE_Func = sqr_exp_kernel(sigmaf = sigmaF, ell = ell) # MaternFunc is a kernel FUNCTION
#SE_Func(1,2) # Evaluating the kernel in x=1, x'=2

# Computing the whole covariance matrix K from the kernel.
K <- kernelMatrix(kernel = SE_Func, x = X, y = Xstar) # So this is K(X,Xstar)
K
```

```
## An object of class "kernelMatrix"
##           [,1]      [,2]       [,3]
## [1,] 2.4261226 0.5413411 0.04443599
## [2,] 2.4261226 4.0000000 2.42612264
## [3,] 0.5413411 2.4261226 4.00000000
```

2.2

Given the model: $temp = f(time) + \epsilon$ with $\epsilon \sim N(0, \sigma_n^2$ and $f \sim GP(0, k(time, time'))$.

First, we calculated the residual variance from a simple qadratic regression fit.

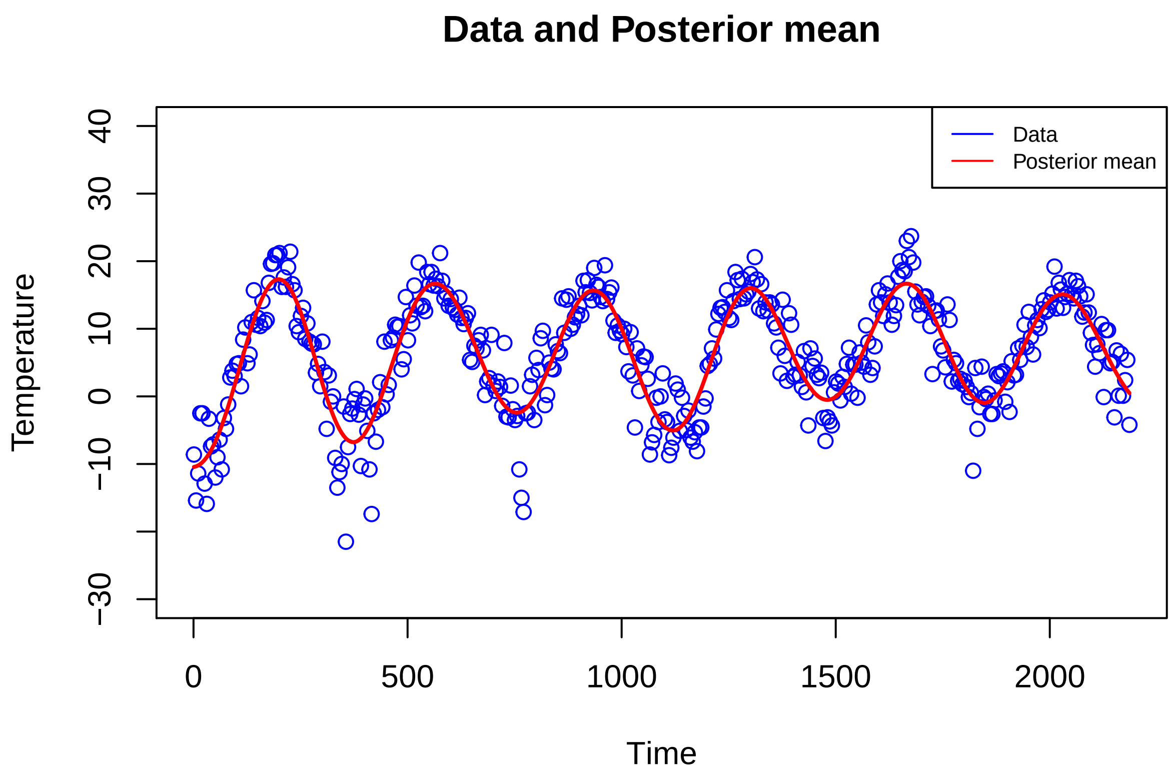
```r
Time <- every_fifth_df$time
Temp <- every_fifth_df$temp

quadratic_fit <- lm(Temp ~ Time + I(Time^2))
sigma_noise <- sd(residuals(quadratic_fit))
sigma_noise
```

```
## [1] 8.176288
```

Then, we computed the posterior mean as follow.

```
# Fit the GP with home made SE_Func
sigmaF <- 20
ell <- 0.2

SE_Func2 = sqr_exp_kernel(sigmaf = sigmaF, ell = ell) # sqr_exp_kernel is a kernel FUNCTION

GPfit <- gausspr(Time, Temp, kernel = SE_Func2,
                 var = sigma_noise^2)
meanPred <- predict(GPfit, Time)
```

The scatter plot of the data shown below with superimpose the posterior mean of f as a curve.

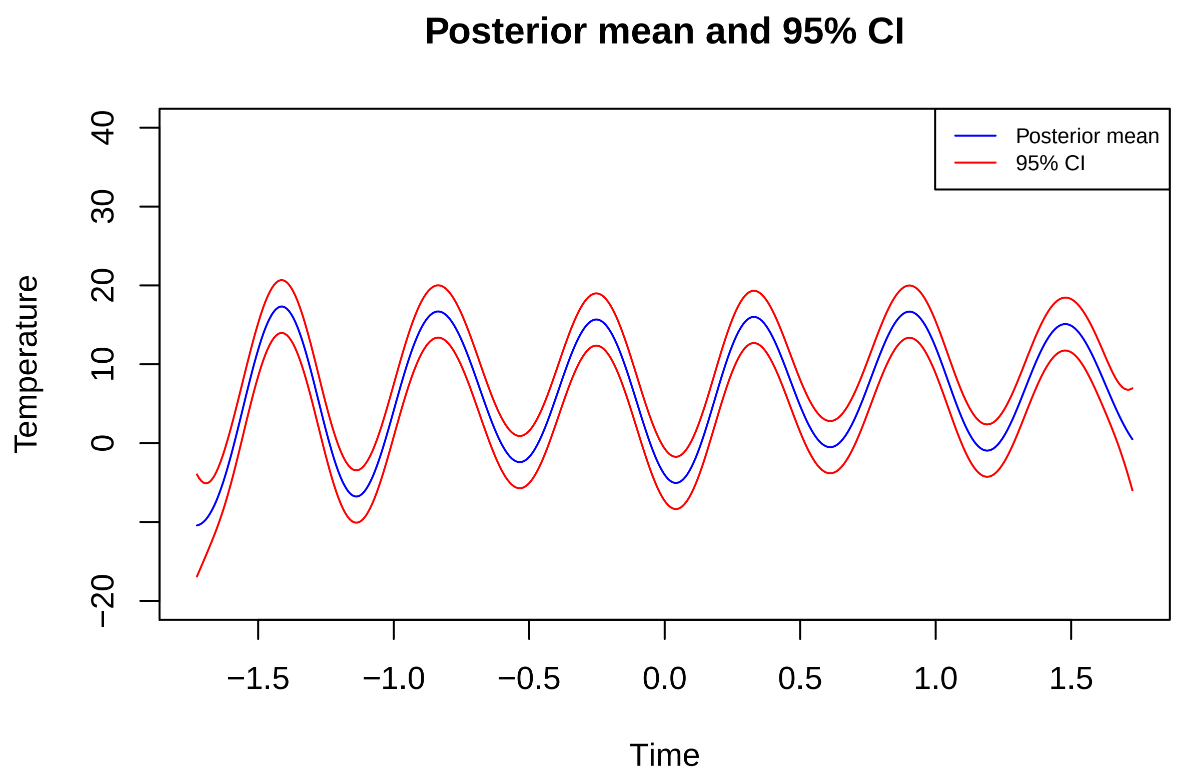


2.3

In this part, we need to compute the posterior variance using *posteriorGP* function in Question 1. We used *scale* function for out input. Then compute the 95% porbability bands.

```
# which is X
scaledTime <- scale(every_fifth_df$time)
#which is y
scaledTemp <- scale(every_fifth_df$temp)

posteriorVariance = posteriorGP(x = scaledTime,
                                y = scaledTemp,
                                                     k= SE_Func2,
                                                   sigmaNoise = sigma_noise,
                                                   xStar = scaledTime)

#95%CI
lowerBound1 <- meanPred - 1.96*sqrt(diag(posteriorVariance$var))
upperBound1 <- meanPred + 1.96*sqrt(diag(posteriorVariance$var))
```

Posterior mean in part 2 is used with 95%CI is plotted below.



2.4

Now, the model given is the function of day.

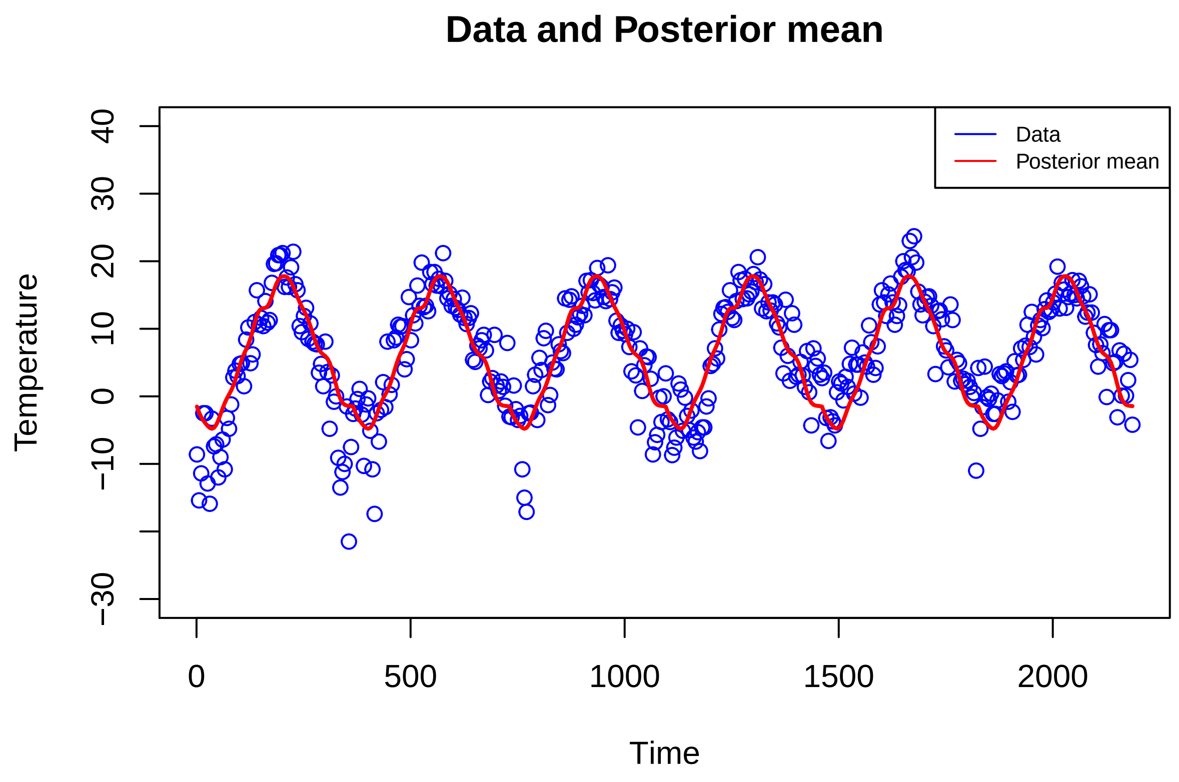
$temp = f(day) + \epsilon$ with $\epsilon \sim N(0, \sigma_n^2$ and $f \sim GP(0, k(day, day'))$.

The model is estimated using the squared exponential function.

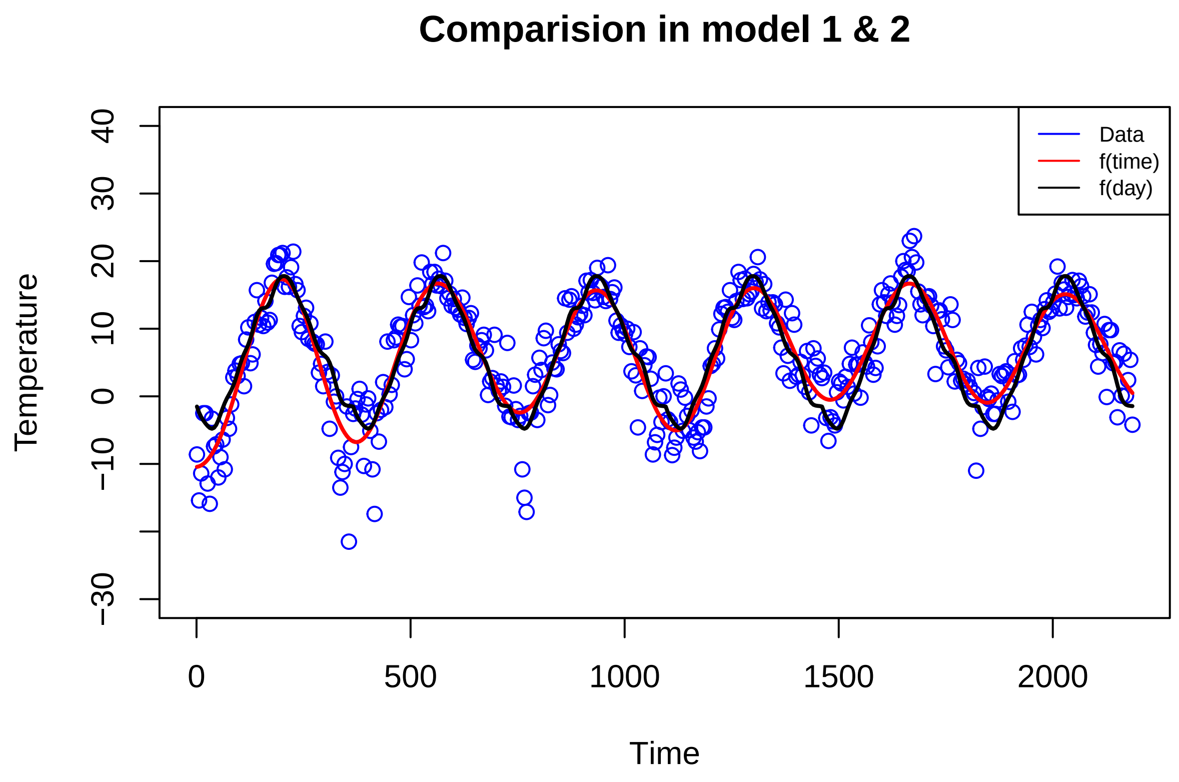
```
Day = every_fifth_df$day

sigmaF <- 20
ell <- 0.2

SE_Func2 = sqr_exp_kernel(sigmaf = sigmaF, ell = ell) # sqr_exp_kernel is a kernel FUNCTION

GPfit2 <- gausspr(Day, Temp,
                  kernel = SE_Func2,
                  var = sigma_noise^2)
```

```r
meanPred2 <- predict(GPfit2, Day)
```

Plot shown below the time variable on the horizontal axis.



Compare the results of both models. What are the pros and cons of each model?



The advantage of having prediction with respect to time is that we get the result at different time span on the same day while with respect to day we get the mean prediction of temperature in a day. Moreover, we get more data points to use while doing prediction w.r.t time.

One thing we must notice is that we are not using the periodic kernel, thus time would give better results as the next temperature point is related to the previous point in observation while when using day, the closest data point is the temperature point on the same day previous year.

2.5

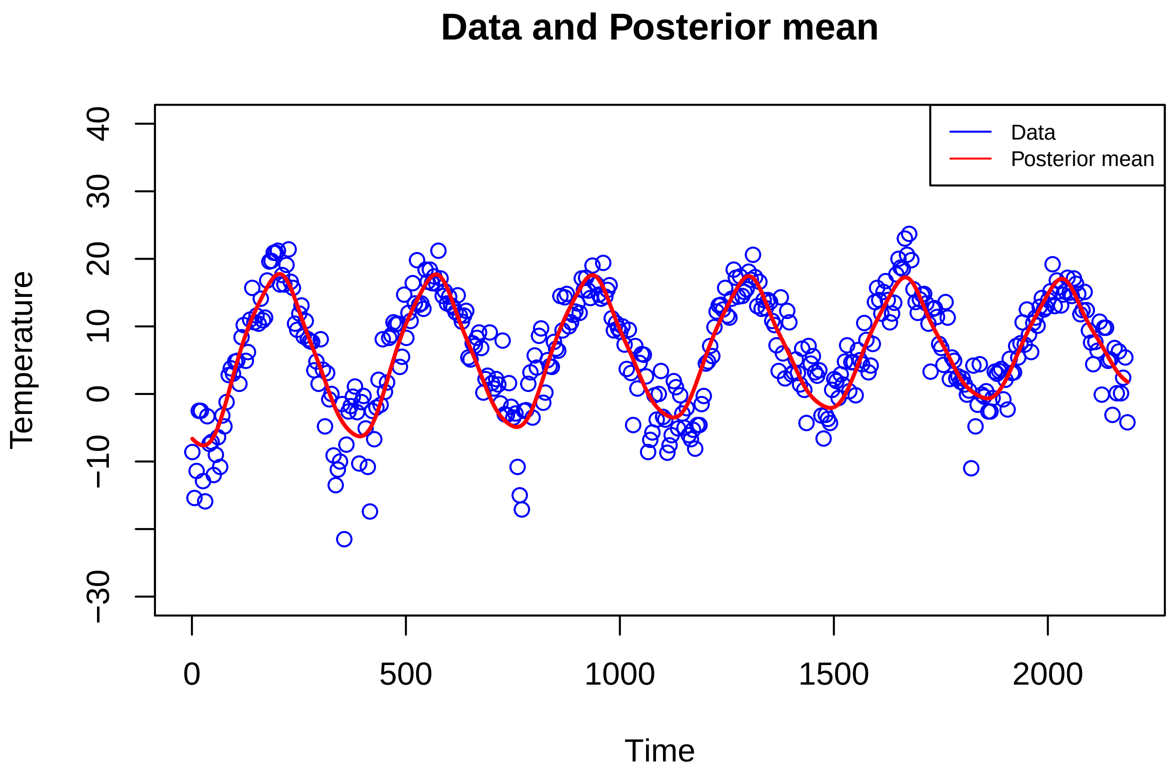
GP model then now estimated using periodic kernel. So, first we implemented the periodic kernel as follow.

```
periodicKernel <- function(sigmaf, l1, l2, d){
  pVal <- function(x1, x2){

    term1 <- (-2*sin(pi*abs(x1-x2)/d)^2)/(l1^2)
    term2 <- (-abs(x1-x2)^2)/(2*l2^2)
    k <- sigmaf^2 * exp(term1) * exp(term2)
    return(k)
  }
  class(pVal) <- "kernel"
  return(pVal)
}
```

The GP model is estimated and plotted below.

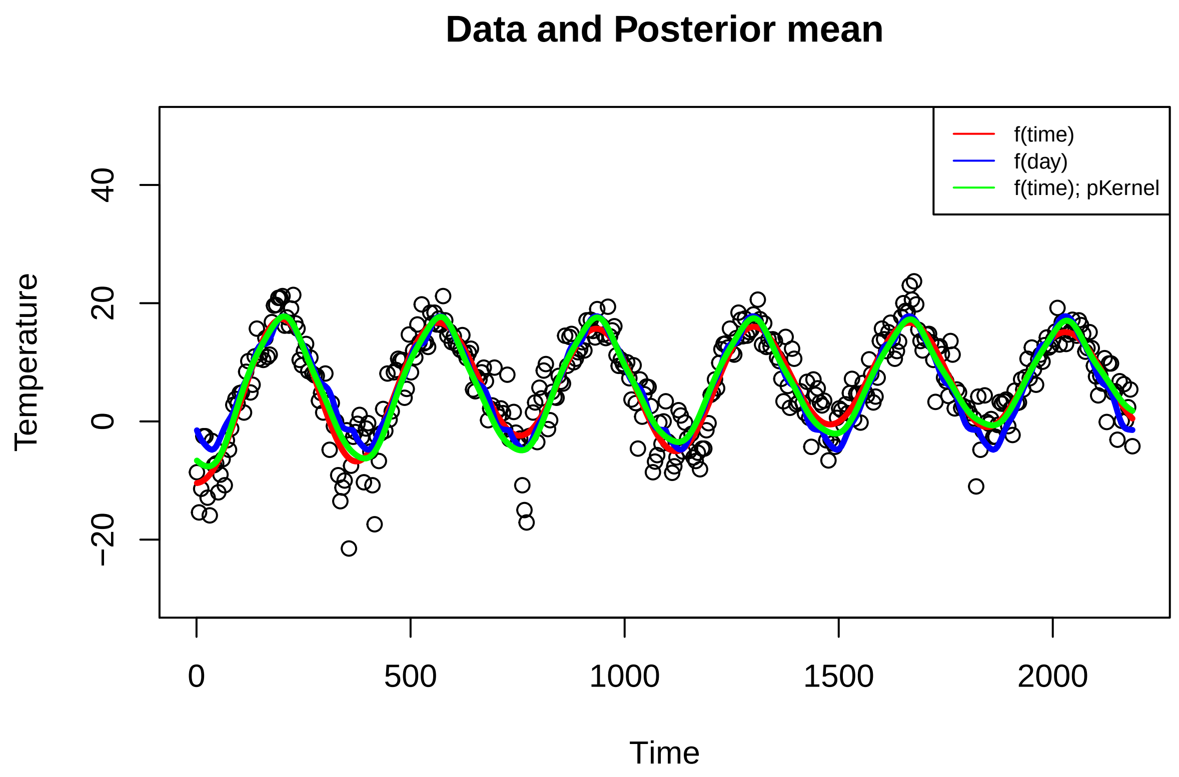
```
Time <- every_fifth_df$time
Temp <- every_fifth_df$temp
```

```
sigmaf=20
l1=1
l2=10
d=365/sd(Time)

pKernel <- periodicKernel(sigmaf, l1, l2, d)

GPFit3 <- gausspr(Time ,Temp, kernel = pKernel, var=sigma_noise^2)
meanPred3 <- predict(GPFit3, Time)
```



Comparison

To compare easily, we plotted all three models on the same plot.



The periodic kernel seems to capture both day and time from part 2.4 and 2.5 because now not only the previous time point is observed rather, analysis is made using the temperature on the given time the previous years.

Question 3: GP Classification with kernlab

Data is downloaded in to R.

```
##   varWave skewWave kurtWave entropyWave fraud
## 1 3.62160   8.6661  -2.8073    -0.44699     0
## 2 4.54590   8.1674  -2.4586    -1.46210     0
## 3 3.86600  -2.6383   1.9242     0.10645     0
## 4 3.45660   9.5228  -4.0112    -3.59440     0
## 5 0.32924  -4.4552   4.5718    -0.98880     0
## 6 4.36840   9.6718  -3.9606    -3.16250     0
```

Data is divided into training and testing sets.

```r
library(kernlab)
library(AtmRay)

set.seed(111);
SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
train <- data[SelectTraining,]
test <- data[-SelectTraining,]
```

3.1

The package *kernlab* is used to fit a Gaussian process classification model for fraud on the training data. *Fraud* is used as response variable, and covariates *varWave* and *skewWave* are used as predictors.

```
GPfitTrain <- gausspr(fraud ~ varWave + skewWave , data = train)
```

```
## Using automatic sigma estimation (sigest) for RBF or laplace kernel
```

```
predTrain <- predict(GPfitTrain, train)
```

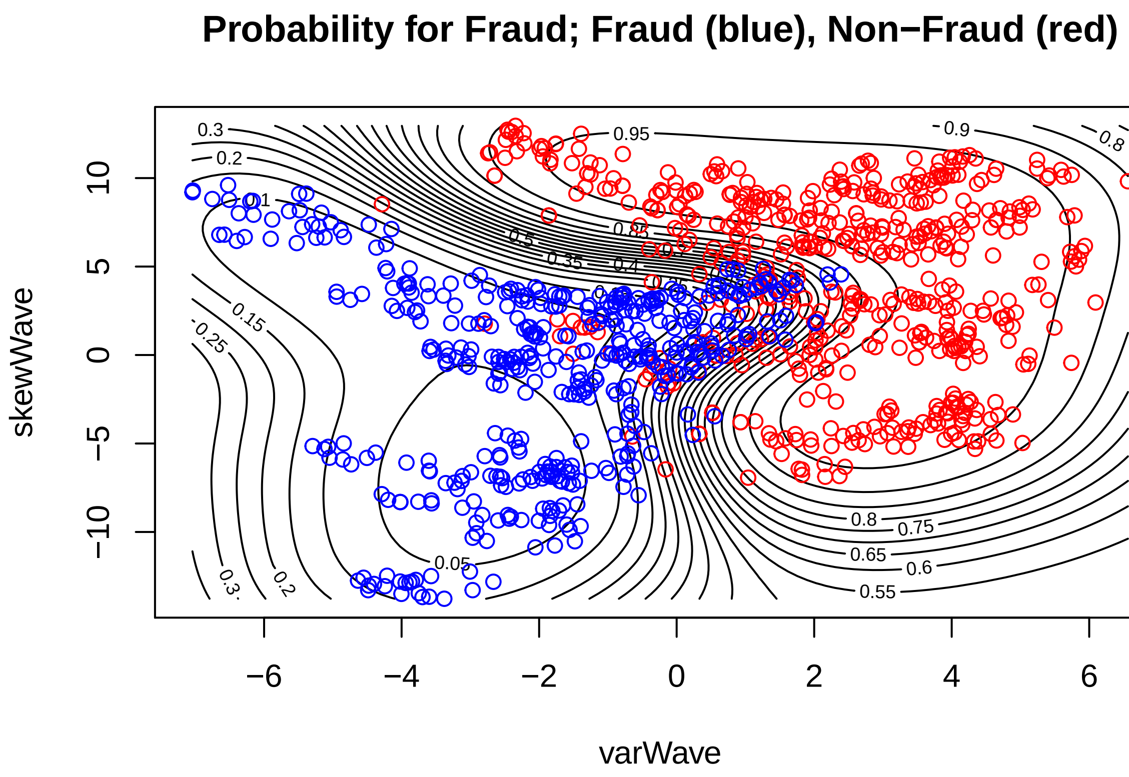
Confusion matrix and its accuracy are computed below.

```
##          Actual
## Predict   0   1
##         0 512  24
##         1  44 420
```

```
## Accuracy:  0.932
```

Plot contours of the prediction probabilities over a suitable grid of values for varWave and skewWave. Overlay the training data for fraud = 1 (as blue points) and fraud = 0 (as red points).

```
#Contour plot
x1 <- seq(min(train$varWave), max(train$varWave),length=100) #Grid for varWave
x2 <- seq(min(train$skewWave),max(train$skewWave),length=100) #Grid for SkewWave

gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(train)[1:2]

probPreds <- predict(GPfitTrain, gridPoints, type="probabilities")

contour(x1, x2,
        matrix(probPreds[,1],100, byrow = TRUE), 20,
        xlab = "varWave",
        ylab = "skewWave",
        main = "Probability for Fraud; Fraud (blue), Non-Fraud (red)")

points(train$varWave[train$fraud==0],train$skewWave[train$fraud==0],col="red") #Non-fraud
points(train$varWave[train$fraud==1],train$skewWave[train$fraud==1],col="blue") #Fraud
```



3.2

The estimate model from 3.1 is re-used to make predictions for the test set.

```
predTest <- predict(GPfitTrain, test)
```

Again, confusion matrix and its accuracy is computed and show below.

```
##         Actual
## Predict   0   1
##       0 191   9
##       1  15 157
```

```
## Accuracy:  0.9354839
```

3.3

The four covariates are used in this part, which are *varWave*, *skewWave*, *kurtWave*, and *entropyWave*. The predictions based on testing data.

```
GPfitAllTrain <- gausspr(fraud ~ . , data = train)
```

```
## Using automatic sigma estimation (sigest) for RBF or laplace kernel
```

```
predTestAll <- predict(GPfitAllTrain, test)
```

```
##             testActural
## trainPredict   0   1
##             0 205   0
##             1   1 166
## Accuracy 0.9973118
```

Comparison

The accuracy of two models, two covariates and all four covariates with predictions made on testing data shown below.

	Accuracy
Two covariates	0.9354839
Four covariates	0.9973118

Appendix

Question 1

```r
#part 1
sqr_exp_kernel <- function(sigmaf, ell)
{
  rval <- function(x, y) {
    n1 <- length(x)
    n2 <- length(y)
    K <- matrix(NA, n1, n2)
    for (i in 1:n2){
      r = abs(x-y[i]);
      K[,i] <- sigmaf^2*exp(-0.5*( (r)/ell)^2 )
    }

    return(K)
  }
  class(rval) <- "kernel"
  return(rval)
}


posteriorGP <- function(x,y,k,sigmaNoise,xStar)
{
  n = length(x)
  #k(x,x)
  k_ = k(x , x)
  #k(x,x*)
  kStar = k(x,xStar)
  #k(x*,x*)
  kStarStar = k(xStar,xStar)

  I =  diag(dim(k_)[2] )
  L <- t(chol(k_ + (sigmaNoise^2) * I))
  alpha <- solve(t(L), solve(L, y))

  fStarMean = t(kStar) %*% alpha
  v <- solve(L, kStar)
  v_fstar = kStarStar - t(v) %*% v

  #temprary
  log_prob = -0.5 %*% t(y) %*% alpha - sum( log(diag(L)) - n/2 * log(2*pi) )

  return(list(mean=  fStarMean , var = v_fstar, marginal_likehood = log_prob ))
}



#part 2
sigmaF = 1
ell = 0.3
x <- 0.4
y = 0.719
sigmaNoise = 0.1
xStar <- seq(-1, 1, length = 100)
SE_Func <- sqr_exp_kernel(sigmaf =sigmaF, ell = ell ) #par = sigmaf, ell

GP1 <- posteriorGP(x=x, y=y,
                   k=SE_Func,
                   sigmaNoise=sigmaNoise,
                   xStar=xStar)
#95%CI
lowerBound1 <- GP1$mean - 1.96*sqrt(diag(GP1$var))
upperBound1 <- GP1$mean + 1.96*sqrt(diag(GP1$var))

#Plot the posterior mean
plot(x=xStar, y=GP1$mean, type = "l", ylim = c(-5,5),
     xlab = "x",
     ylab = "Mean",
     main = "Posterior mean and 95% CI (Data point: 1)")

#Plot 95%CI pointwise?????
lines(x=xStar, y=lowerBound1, col="red")
lines(x=xStar, y=upperBound1, col="red")
points(x=x, y=y, col="blue")
legend("topright", legend=c("Posterior mean", "95% CI" ),
       col=c("black", "red"), lty=1:1, cex=0.8)


#part 3
sigmaF = 1
ell = 0.3
x2 <- c(-0.6, 0.4)  #updated
y2 = c(-0.044, 0.719) #updated
sigmaNoise = 0.1
xStar <- seq(-1, 1, length = 100)
SE_Func <- sqr_exp_kernel(sigmaf =sigmaF, ell = ell )
GP2 <- posteriorGP(x=x2, y=y2,
                   k=SE_Func,
                   sigmaNoise=sigmaNoise,
                   xStar=xStar)
#95%CI
lowerBound2 <- GP2$mean - 1.96*sqrt(diag(GP2$var))
upperBound2 <- GP2$mean + 1.96*sqrt(diag(GP2$var))
```

```r
#Plot the posterior mean
plot(x=xStar, y=GP2$mean, type = "l", ylim = c(-5,5),
     xlab = "x",
     ylab = "Mean",
     main = "Posterior mean and 95% CI (Data points: 2)")

#Plot 95%CI
lines(x=xStar, y=lowerBound2, col="red")
lines(x=xStar, y=upperBound2, col="red")
points(x=x2, y=y2, col="blue")
legend("topright", legend=c("Posterior mean", "95% CI" ),
       col=c("black", "red"), lty=1:1, cex=0.8)



#part 4
#Initial setting
sigmaF = 1
ell = 0.3
x3 <- c(-1, -0.6, -0.2, 0.4, 0.8 )  #updated
y3 = c(0.768, -0.044, -0.940, 0.719, -0.664) #updated
sigmaNoise = 0.1
xStar <- seq(-1, 1, length = 100)
SE_Func <- sqr_exp_kernel(sigmaf =sigmaF, ell = ell )

GP3 <- posteriorGP(x=x3, y=y3,
                   k=SE_Func,
                   sigmaNoise=sigmaNoise,
                   xStar=xStar)
#95%CI
lowerBound3 <- GP3$mean - 1.96*sqrt(diag(GP3$var))
upperBound3 <- GP3$mean + 1.96*sqrt(diag(GP3$var))

#Plot the posterior mean
plot(x=xStar, y=GP3$mean, type = "l", ylim = c(-5,5),
     xlab = "x",
     ylab = "Mean",
     main = "Posterior mean and 95% CI (Data points: 5)")

#Plot 95%CI
lines(x=xStar, y=lowerBound3, col="red")
lines(x=xStar, y=upperBound3, col="red")
points(x=x3, y=y3, col="blue")
legend("topright", legend=c("Posterior mean", "95% CI" ),
       col=c("black", "red"), lty=1:1, cex=0.8)


#part 5
sigmaF = 1
ell2 = 1 #Updated
x3 <- c(-1, -0.6, -0.2, 0.4, 0.8 )
y3 = c(0.768, -0.044, -0.94, 0.719, -0.664)
sigmaNoise = 0.1
```

```r
xStar <- seq(-1, 1, length = 100)
SE_Func <- sqr_exp_kernel(sigmaf =sigmaF, ell = ell2 )

GP4 <- posteriorGP(x=x3, y=y3,
                   k=SE_Func,
                   sigmaNoise=sigmaNoise,
                   xStar=xStar)
#95%CI
lowerBound4 <- GP4$mean - 1.96*sqrt(diag(GP4$var))
upperBound4 <- GP4$mean + 1.96*sqrt(diag(GP4$var))

#Plot the posterior mean
plot(x=xStar, y=GP4$mean, type = "l", ylim = c(-3,3),
     xlab = "x",
     ylab = "Mean",
     main = "Posterior mean and 95% CI (Data points: 5; Updated l)")

#Plot 95%CI
lines(x=xStar, y=lowerBound4, col="red")
lines(x=xStar, y=upperBound4, col="red")
points(x=x3, y=y3, col="blue")
legend("topright", legend=c("Posterior mean", "95% CI" ),
       col=c("black", "red"), lty=1:1, cex=0.8)
```

Question 2

```r
library(kernlab)
library(AtmRay)
library(mvtnorm)

data = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.c

#generate a time variable
time = seq(1,2190,1)

#for day
day = rep(seq(1,365,1),6)

#generate a every fifth value
every_fifth_time = time[seq(1, length(time), 5)]
every_fifth_day = day[seq(1, length(day), 5)]


#adding a time and day to original dataFrame
data$day = day
data$time = time

#every fifth data exract from dataFrame on basis of day
every_fifth_df = data[data$day %in% every_fifth_day,]


###Part_1
```

```r
sqr_exp_kernel <- function(sigmaf, ell)
{
  rval <- function(x, y) {
    n1 <- length(x)
    n2 <- length(y)
    K <- matrix(NA, n1, n2)
    for (i in 1:n2){
      r = abs(x-y[i]);
      K[,i] <- sigmaf^2*exp(-0.5*( (r)/ell)^2 )
    }

    return(K)
  }
  class(rval) <- "kernel"
  return(rval)
}

ell = 1 #x
sigmaF = 2 #x'

X <- matrix(c(1,3,4),ncol = 1)
Xstar <- matrix(c(2,3,4),ncol = 1)

SE_Func = sqr_exp_kernel(sigmaf = sigmaF, ell = ell) # MaternFunc is a kernel FUNCTION
#SE_Func(1,2) # Evaluating the kernel in x=1, x'=2

# Computing the whole covariance matrix K from the kernel.
K <- kernelMatrix(kernel = SE_Func, x = X, y = Xstar) # So this is K(X,Xstar)


## Part 2
Time <- every_fifth_df$time
Temp <- every_fifth_df$temp

quadratic_fit <- lm(Temp ~ Time + I(Time^2))
sigma_noise <- sd(residuals(quadratic_fit))

# Fit the GP with home made SE_Func
sigmaF <- 20
ell <- 0.2

SE_Func2 = sqr_exp_kernel(sigmaf = sigmaF, ell = ell) # sqr_exp_kernel is a kernel FUNCTION

GPfit <- gausspr(Time, Temp, kernel = SE_Func2, kpar = list(sigmaf = sigmaf, ell=ell), var = sigma_nois
meanPred <- predict(GPfit, Time)

#plot
plot(x=every_fifth_df$time, y=every_fifth_df$temp, col="blue", ylim=c(-30, 40),
     xlab="Time",
     ylab="Temperature",
     main="Data and Posterior mean")
lines(x=every_fifth_df$time, y=meanPred, col="red", lwd = 2)
legend("topright", legend=c("Data", "Posterior mean" ),
       col=c("blue", "red"), lty=1:1, cex=0.7)


#part 3
#algorithm 2.1 on page 19
# x = inputs
# y = Target
# k = covariance function
# sigmaSquareN = noise level
# xStar = test input
posterior_variance <- function(x,y,k,sigmaNoise,xStar)
{
  n = length(x)
  #k(x,x)
  k_ = k(x , x)
  #k(x,x*)
  kStar = k(x,xStar)
  #k(x*,x*)
  kStarStar = k(xStar,xStar)

  I =  diag(dim(k_)[2] )
  L <- t(chol(k_ + (sigmaNoise^2) * I))
  alpha <- solve(t(L), solve(L, y))

  fStarMean = t(kStar) %*% alpha
  v <- solve(L, kStar)
  v_fstar = kStarStar - t(v) %*% v

  #temprary
  log_prob = -0.5 %*% t(y) %*% alpha - sum( log(diag(L)) - n/2 * log(2*pi) )

  return(list(mean=  fStarMean , var = v_fstar, marginal_likehood = log_prob ))
}
# which is X
scaledTime <- scale(every_fifth_df$time)
#which is y
scaledTemp <- scale(every_fifth_df$temp)




posteriorVariance = posterior_variance(x = scaledTime, y = scaledTemp,k= SE_Func2,
                   sigmaNoise = sigma_noise,xStar = scaledTime)



#95%CI
lowerBound1 <- meanPred - 1.96*sqrt(diag(posteriorVariance$var))
upperBound1 <- meanPred + 1.96*sqrt(diag(posteriorVariance$var))


#plot
plot(x=scaledTime, y=meanPred, ylim=c(-20, 40), type="l",
```

```r
     xlab="Time",
     ylab="Temperature",
     main="Posterior mean and 95% CI")
lines(x=scaledTime, y=lowerBound1, col="red")
lines(x=scaledTime, y=upperBound1, col="red")
legend("topright", legend=c("Posterior mean", "95% CI" ),
       col=c("blue", "red"), lty=1:1, cex=0.7)


#part 4
Day = every_fifth_df$day

sigmaF <- 20
ell <- 0.2

SE_Func2 = sqr_exp_kernel(sigmaf = sigmaF, ell = ell) # sqr_exp_kernel is a kernel FUNCTION

GPfit2 <- gausspr(Day, Temp, kernel = SE_Func2, kpar = list(sigmaf = sigmaf, ell=ell), var = sigma_nois
meanPred2 <- predict(GPfit2, Day)

#plot
plot(x=every_fifth_df$time, y=every_fifth_df$temp, col="blue", ylim=c(-30, 40),
     xlab="Time",
     ylab="Temperature",
     main="Data and Posterior mean")
lines(x=every_fifth_df$time, y=meanPred2, col="red", lwd = 2)
legend("topright", legend=c("Data", "Posterior mean" ),
       col=c("blue", "red"), lty=1:1, cex=0.7)


#compare models
plot(x=every_fifth_df$time, y=every_fifth_df$temp, col="blue", ylim=c(-30, 40),
     xlab="Time",
     ylab="Temperature",
     main="Comparision in model 1 & 2")

lines(x=every_fifth_df$time, y=meanPred, col="red", lwd = 2)
lines(x=every_fifth_df$time, y=meanPred2, col="black", lwd = 2)
legend("topright", legend=c("Data", "f(time)", "f(day)" ),
       col=c("blue", "red","black"), lty=1:1, cex=0.7)

# part 5
periodicKernel <- function(sigmaf, l1, l2, d){
  pVal <- function(x1, x2){

    term1 <- (-2*sin(pi*abs(x1-x2)/d)^2)/(l1^2)
    term2 <- (-abs(x1-x2)^2)/(2*l2^2)
    k <- sigmaf^2 * exp(term1) * exp(term2)
    return(k)
  }
  class(pVal) <- "kernel"
  return(pVal)
}
```

```r
##setting
Time <- every_fifth_df$time
Temp <- every_fifth_df$temp
sigmaf=20
l1=1
l2=10
d=365/sd(Time)

pKernel <- periodicKernel(sigmaf, l1, l2, d)

GPFit3 <- gausspr(Time ,Temp, kernel = pKernel, var=sigma_noise^2)
meanPred3 <- predict(GPFit3, Time)

##Plot
plot(x=every_fifth_df$time, y=every_fifth_df$temp, col="blue", ylim=c(-30, 40),
     xlab="Time",
     ylab="Temperature",
     main="Data and Posterior mean")
lines(x=every_fifth_df$time, y=meanPred3, col="red", lwd=2)
legend("topright", legend=c("Data", "Posterior mean" ),
       col=c("blue", "red"), lty=1:1, cex=0.7)

##Comparison
plot(x=every_fifth_df$time, y=every_fifth_df$temp, col="black", ylim=c(-30, 50),
     xlab="Time",
     ylab="Temperature",
     main="Data and Posterior mean")

lines(x=every_fifth_df$time, y=meanPred, col="red", lwd=3)
lines(x=every_fifth_df$time, y=meanPred2, col="blue", lwd=3)
lines(x=every_fifth_df$time, y=meanPred3, col="green", lwd=3)
legend("topright", legend=c("f(time)", "f(day)", "f(time); pKernel" ),
       col=c("red", "blue","green"), lty=1:1, cex=0.7)
```

Question 3

```r
library(kernlab)
library(AtmRay)
#Reading and setting the data
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/
GaussianProcess/Code/banknoteFraud.csv", header=FALSE, sep=",")

names(data) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")
data[,5] <- as.factor(data[,5])
head(data)


set.seed(111)
sampledData <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
train <- data[sampledData,]
test <- data[-sampledData,]
```

```
#3.1
#gaussion fiting
GPfitTrain <- gausspr(fraud ~ (varWave + skewWave) , data = train)
#predition
predTrain <- predict(GPfitTrain, train)

### confusion matrix
predTrainMat <- table(Predict=predTrain, Actual=train$fraud)
predTrainMat

predTrainAccuracy <- (sum(diag(predTrainMat)))/sum(predTrainMat)

cat("Accuracy : ", predTrainAccuracy )


#Contour plot
x1 <- seq(min(train$varWave), max(train$varWave),length=100) #Grid for varWave
x2 <- seq(min(train$skewWave),max(train$skewWave),length=100) #Grid for SkewWave

gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(train)[1:2]

probPreds <- predict(GPfitTrain, gridPoints, type="probabilities")

contour(x1, x2,
        matrix(probPreds[,1],100, byrow = TRUE), 20,
        xlab = "varWave",
        ylab = "skewWave",
        main = "Probability for Fraud; Fraud (blue), Non-Fraud (red)")

points(train$varWave[train$fraud==0],train$skewWave[train$fraud==0],col="red") #Non-fraud
points(train$varWave[train$fraud==1],train$skewWave[train$fraud==1],col="blue") #Fraud



#### 3.2
predTest <- predict(GPfitTrain, test)

predTestMat <- table(Predict=predTest, Actual=test$fraud)
predTestMat

predTestAccuracy <- (sum(diag(predTestMat)))/sum(predTestMat)

cat("Accuracy: ", predTestAccuracy )

#### 3.3

GPfitAllTrain <- gausspr(fraud ~ . , data = train)
predTestAll <- predict(GPfitAllTrain, test)
```

```r
### accuracy

predTestAllMat <- table(trainPredict = predTestAll, testActural = test$fraud)
predTestAllMat

predTestAllAccuracy <- (sum(diag(predTestAllMat)))/sum(predTestAllMat)
cat("Accuracy", predTestAllAccuracy)


### compairion
tableDF <- data.frame(Accuracy=c(predTestAccuracy, predTestAllAccuracy))
rownames(tableDF) <- c("Two covariates", "Four covariates")
```