examhelpaml

Omkar Bhutra (omkbh878)

9 January 2020

yaml setup for lab 1

LAB 1 stuff

Task 1:

```
data("asia")
asia <- as.data.frame(asia)
#default parameter run of HC
hillClimbingResults = hc(asia)
print(hillClimbingResults)
```

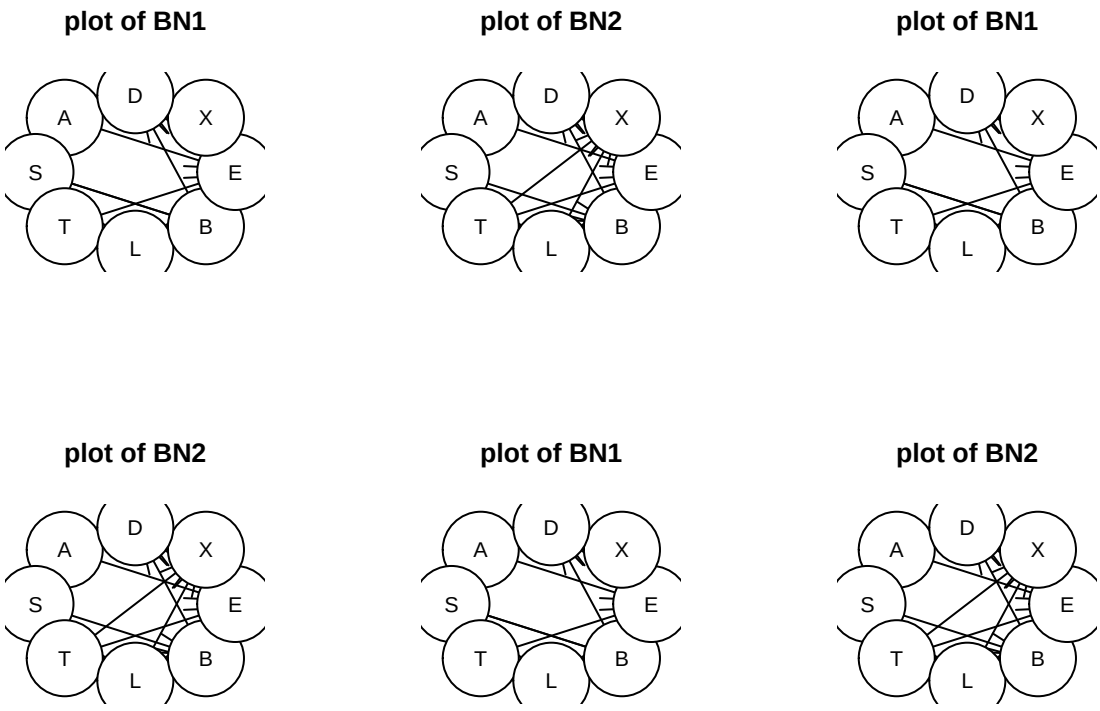
```
##
##   Bayesian network learned via Score-based methods
##
##   model:
##    [A][S][T][L|S][B|S][E|T:L][X|E][D|B:E]
##   nodes:                                 8
##   arcs:                                  7
##     undirected arcs:                     0
##     directed arcs:                       7
##   average markov blanket size:           2.25
##   average neighbourhood size:            1.75
##   average branching factor:              0.88
##
##   learning algorithm:                    Hill-Climbing
##   score:                                 BIC (disc.)
##   penalization coefficient:              4.258597
##   tests used in the learning procedure:  77
##   optimized:                             TRUE
```

```
par(mfrow = c(2,3))

hillclimb <- function(x){
  hc1 <- hc(x,restart = 15,score = "bde", iss = 3)
  hc2 <- hc(x,restart = 10,score = "bde", iss = 5)
  hc1dag <- cpdag(hc1)
  plot(hc1dag, main = "plot of BN1")
  hc1arcs <- vstructs(hc1dag, arcs = TRUE)
  # print(hc1arcs)
  arcs(hc1)
  hc2dag <- cpdag(hc2)
  plot(hc2dag, main = "plot of BN2")
  hc2arcs <- vstructs(hc2dag, arcs = TRUE)
```

```
  # print(hc2arcs)
  arcs(hc2)
  print(all.equal(hc1dag,hc2dag))
}
for(i in 1:3){
  hillclimb(asia)
}
```

```
## [1] "Different number of directed/undirected arcs"
```

```
## [1] "Different number of directed/undirected arcs"
```




```
## [1] "Different number of directed/undirected arcs"
```

Non-equivalant solutions are produced using different starting parameters such as number of restarts, scoring algorithm, imaginary sample size etc. Hill climbing algorithm is a deterministic one, and hence generates different results on different input parameters. Hill climbing leads to a local maxima of the objective function which makes two different results possible with the same code. Adding imaginary sample size affects the relationships between the nodes such that we get more edges. Number of restarts increases the possibility of getting the same result.

Task 2:

Network structure is trained by the hill climbing algorithm using the BDE score. bn.fit is used to learn the paramets using maximum liklihood estimation. To predict S in the test data , evidence in the network is set to the values of the test data excluding S, using setEvidence. querygrain is used to find the probabilty, maximum of the values is taken to find the misclassification rate.

```
#setting up training and testing datasets
id <- sample(x = seq(1, dim(asia)[1], 1),
             size = dim(asia)[1]*0.8,
             replace = FALSE)
asia.train <- asia[id,]
asia.test <- asia[-id,]

bnprediction = as.numeric()
#fitting a model using Hill Climbing
bnmodel <- hc(asia.train,score = "bde",restart = 50)
bnmodelfit <- bn.fit(bnmodel,asia.train,method = 'mle') #max liklihood estimation
bngrain <- as.grain(bnmodelfit)
comp <- compile(bngrain)

bnmodel_true <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
bnmodelfit_true <- bn.fit(bnmodel_true, asia.train)
bngrain_true <- as.grain(bnmodelfit_true)
comp_true <- compile(bngrain_true)

bnpredict <- function(bntree, data, obs_variables, pred_variable) {
  for (i in 1:dim(data)[1]) {
    x <- NULL
    for (j in obs_variables) {
      x[j] <- if(data[i, j] == "yes") "yes" else "no"
    }
    evidence <- setEvidence(object = bntree,nodes = obs_variables,states=x)
    prob_dist_fit  <- querygrain(object = evidence,nodes = pred_variable)$S
    bnprediction[i] <- if (prob_dist_fit["yes"] >= 0.5) "yes" else "no"
  }
  return(bnprediction)
}

# Predict S from Bayesian Network and test data observations
bnprediction <- bnpredict(bntree = comp,
                          data = asia.test,
                          obs_variables = c("A", "T", "L", "B", "E", "X", "D"),
                          pred_variable = c("S"))
bnprediction_true <- bnpredict(bntree = comp_true,
                               data = asia.test,
                               obs_variables = c("A", "T", "L", "B", "E", "X", "D"),
                               pred_variable = c("S"))

# Calculate confusion matricies
confusion_matrix_fit <- table(bnprediction, asia.test$S)
print(confusion_matrix_fit)
```

```
##
```

```
## bnprediction  no yes
##          no  349 112
##          yes 158 381
```

```
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit))/sum(confusion_matrix_fit)))
```

```
## [1] "Misclassification rate: 0.27"
```

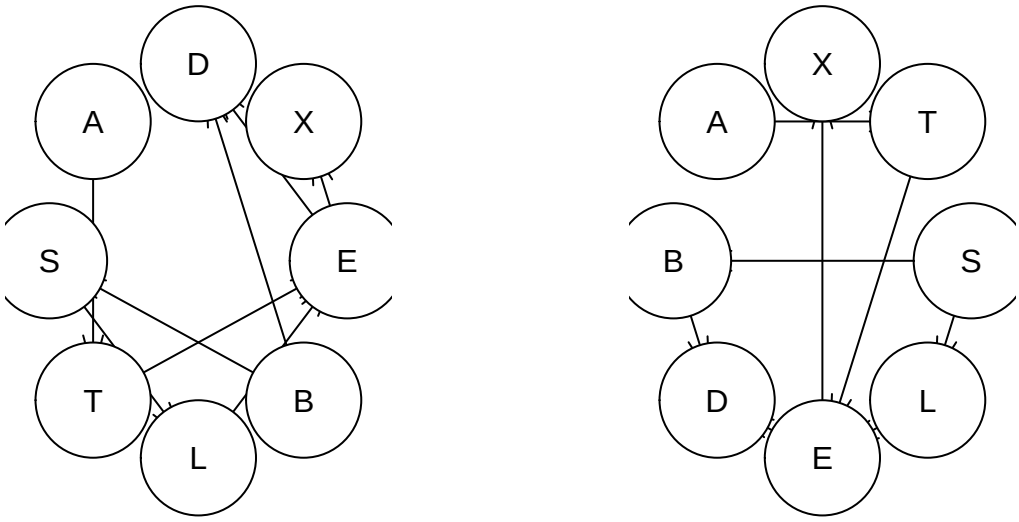
```
confusion_matrix_true <- table(bnprediction_true, asia.test$S)
print(confusion_matrix_true)
```

```
##
## bnprediction_true  no yes
##               no  349 112
##               yes 158 381
```

```
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_true))/sum(confusion_matrix_true)))
```

```
## [1] "Misclassification rate: 0.27"
```

```
par(mfrow=c(1,2))
plot(bnmodel)
title("hill climb network")
plot(bnmodel_true)
title("True network")
```



```r
par(mfrow=c(1,1))
```

The Misclassification rate remain the same for both the hill climb network and the true network. However, The state of "Different number of directed/undirected arcs" persists.

Task 3:

Using the markov blanket, we predict the results again, it is expected to output the same result.

```r
markov_blanket = mb(bnmodel, c("S"))
markov_blanket_true = mb(bnmodel_true, c("S"))
bnpredict_mb <- bnpredict(comp, asia.test, markov_blanket, c("S"))
bnpredict_mb_true <- bnpredict(comp_true, asia.test, markov_blanket_true, c("S"))
confusion_matrix_fit <- table(bnpredict_mb, asia.test$S)
confusion_matrix_fit_true <- table(bnpredict_mb_true, asia.test$S)
print(confusion_matrix_fit)
```

```
##
## bnpredict_mb  no yes
##          no  349 112
##          yes 158 381
```

```r
print(confusion_matrix_fit_true)
```

```
##
## bnpredict_mb_true  no yes
##               no  349 112
##               yes 158 381
```

```r
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit))/sum(confusion_matrix_fit)))
```

```
## [1] "Misclassification rate: 0.27"
```

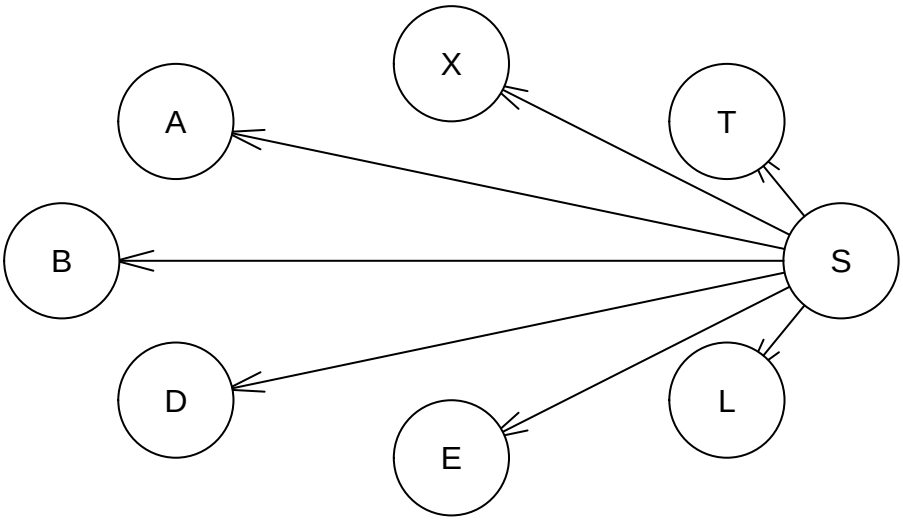
```r
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit_true))/sum(confusion_matrix_fit_
```

```
## [1] "Misclassification rate: 0.27"
```

Task 4:

The naiveBayes structure implies that all variables are independant of S, this implies that there is a diffrent true structure used.

```r
naiveBayes = model2network("[S][A|S][B|S][D|S][E|S][X|S][L|S][T|S]")
plot(naiveBayes, main = "Naive Bayes")
```




```
naiveBayes.fit <- bn.fit(naiveBayes, asia.train)
result_naive <- bnpredict(compile(as.grain(naiveBayes.fit)), asia.test, c("A", "T", "L", "B", "E", "X",
# Calculate confusion matricies
confusion_matrix_naive_bayes <- table(result_naive, asia.test$S)
print(confusion_matrix_naive_bayes)
```

```
##
## result_naive  no yes
##          no  369 181
##          yes 138 312
```

```
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_naive_bayes))/sum(confusion_matrix_na
```

```
## [1] "Misclassification rate: 0.319"
```

Task 5:

In Task 2, Same results are observed for the trained structure and the true structure. Same results are also obtained in Task 3 as before, this is expected due to the reduction of the node from the rest of the structure. The reduced number of predictors still provides the same misclassification rate although some information loss is expected. In Task 4, Naive bayes is not a predictor for the data as the structure used is different as the variables are inferred to be independant of S, but all are assumed to have an effect on S.

```r
knitr::opts_chunk$set(echo = TRUE)
library("dplyr")
library("ggplot2")
library("gRain")
library("bnlearn")
data("asia")
asia <- as.data.frame(asia)
#default parameter run of HC
hillClimbingResults = hc(asia)
print(hillClimbingResults)
par(mfrow = c(2,3))

hillclimb <- function(x){
  hc1 <- hc(x,restart = 15,score = "bde", iss = 3)
  hc2 <- hc(x,restart = 10,score = "bde", iss = 5)
  hc1dag <- cpdag(hc1)
  plot(hc1dag, main = "plot of BN1")
  hc1arcs <- vstructs(hc1dag, arcs = TRUE)
  # print(hc1arcs)
  arcs(hc1)
  hc2dag <- cpdag(hc2)
  plot(hc2dag, main = "plot of BN2")
  hc2arcs <- vstructs(hc2dag, arcs = TRUE)
  # print(hc2arcs)
  arcs(hc2)
  print(all.equal(hc1dag,hc2dag))
}
for(i in 1:3){
  hillclimb(asia)
}
#setting up training and testing datasets
id <- sample(x = seq(1, dim(asia)[1], 1),
             size = dim(asia)[1]*0.8,
             replace = FALSE)
asia.train <- asia[id,]
asia.test <- asia[-id,]

bnprediction = as.numeric()
#fitting a model using Hill Climbing
bnmodel <- hc(asia.train,score = "bde",restart = 50)
bnmodelfit <- bn.fit(bnmodel,asia.train,method = 'mle') #max liklihood estimation
bngrain <- as.grain(bnmodelfit)
comp <- compile(bngrain)

bnmodel_true <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
bnmodelfit_true <- bn.fit(bnmodel_true, asia.train)
bngrain_true <- as.grain(bnmodelfit_true)
comp_true <- compile(bngrain_true)

bnpredict <- function(bntree, data, obs_variables, pred_variable) {
  for (i in 1:dim(data)[1]) {
    x <- NULL
    for (j in obs_variables) {
```

```
      x[j] <- if(data[i, j] == "yes") "yes" else "no"
    }
    evidence <- setEvidence(object = bntree,nodes = obs_variables,states=x)
    prob_dist_fit  <- querygrain(object = evidence,nodes = pred_variable)$S
    bnprediction[i] <- if (prob_dist_fit["yes"] >= 0.5) "yes" else "no"
  }
  return(bnprediction)
}

# Predict S from Bayesian Network and test data observations
bnprediction <- bnpredict(bntree = comp,
                          data = asia.test,
                          obs_variables = c("A", "T", "L", "B", "E", "X", "D"),
                          pred_variable = c("S"))
bnprediction_true <- bnpredict(bntree = comp_true,
                               data = asia.test,
                               obs_variables = c("A", "T", "L", "B", "E", "X", "D"),
                               pred_variable = c("S"))

# Calculate confusion matricies
confusion_matrix_fit <- table(bnprediction, asia.test$S)
print(confusion_matrix_fit)
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit))/sum(confusion_matrix_fit)))

confusion_matrix_true <- table(bnprediction_true, asia.test$S)
print(confusion_matrix_true)
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_true))/sum(confusion_matrix_true)))

par(mfrow=c(1,2))
plot(bnmodel)
title("hill climb network")
plot(bnmodel_true)
title("True network")
par(mfrow=c(1,1))
markov_blanket = mb(bnmodel, c("S"))
markov_blanket_true = mb(bnmodel_true, c("S"))
bnpredict_mb <- bnpredict(comp, asia.test, markov_blanket, c("S"))
bnpredict_mb_true <- bnpredict(comp_true, asia.test, markov_blanket_true, c("S"))
confusion_matrix_fit <- table(bnpredict_mb, asia.test$S)
confusion_matrix_fit_true <- table(bnpredict_mb_true, asia.test$S)
print(confusion_matrix_fit)
print(confusion_matrix_fit_true)
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit))/sum(confusion_matrix_fit)))
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit_true))/sum(confusion_matrix_fit_t
naiveBayes = model2network("[S][A|S][B|S][D|S][E|S][X|S][L|S][T|S]")
plot(naiveBayes, main = "Naive Bayes")

naiveBayes.fit <- bn.fit(naiveBayes, asia.train)
result_naive <- bnpredict(compile(as.grain(naiveBayes.fit)), asia.test, c("A", "T", "L", "B", "E", "X",
# Calculate confusion matricies
confusion_matrix_naive_bayes <- table(result_naive, asia.test$S)
print(confusion_matrix_naive_bayes)
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_naive_bayes))/sum(confusion_matrix_na
```

```r
# Question 1: PGMs

# Learn a BN from the Asia dataset, find a separation statement (e.g. B _|_ E | S, T) and, then, check
# it corresponds to a statistical independence.
library(bnlearn)
library(gRain)
set.seed(123)
data("asia")
hc3<-hc(asia,restart=10,score="bde",iss=10)
plot(hc3)
hc4<-bn.fit(hc3,asia,method="bayes")
hc5<-as.grain(hc4)
hc6<-compile(hc5)
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("yes","yes","yes"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("yes","yes","no"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("yes","no","yes"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("yes","no","no"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("no","yes","yes"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("no","yes","no"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("no","no","yes"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("no","no","no"))
querygrain(hc7,c("B"))

# Sample DAGs at random and, then, check which of them coincide with their CPDAGs, i.e. the CPDAGs have
# undirected edge (recall that a CPDAG has an undirected edge if and only if there are two DAGs in the
# equivalence class that differ in the direction of that edge).

#  The exact ratio according to the literature is 11.2
library(bnlearn)
set.seed(123)
ss<-50000
x<-random.graph(c("A","B","C","D","E"),num=ss,method="melancon",every=50,burn.in=30000)

y<-unique(x)
z<-lapply(y,cpdag)

r=0
for(i in 1:length(y)) {
  if(all.equal(y[[i]],z[[i]])==TRUE)
    r<-r+1
}
length(y)/r
# PGMs

#source("https://bioconductor.org/biocLite.R")
#biocLite("RBGL")
```

```r
library(bnlearn)
library(gRain)
set.seed(567)
data("asia")
ind <- sample(1:5000, 4000)
tr <- asia[ind,]
te <- asia[-ind,]

nb0<-empty.graph(c("S","A","T","L","B","E","X","D"))
arc.set<-matrix(c("S", "A", "S", "T", "S", "L", "S", "B", "S", "E", "S", "X", "S", "D"),
                ncol = 2, byrow = TRUE, dimnames = list(NULL, c("from", "to")))
arcs(nb0)<-arc.set
#nb<-naive.bayes(tr,"S")
plot(nb0)

totalError<-array(dim=6)
k<-1
for(j in c(10,20,50,100,1000,2000)){
  nb<-bn.fit(nb0,tr[1:j,],method="bayes")
  nb2<-as.grain(nb)
  nb2<-compile(nb2)

  error<-matrix(c(0,0,0,0),nrow=2,ncol=2)
  for(i in 1:1000){
    z<-NULL
    for(j in c("A","T","L","B","E","X","D")){
      if(te[i,j]=="no"){
        z<-c(z,"no")
      }
      else{
        z<-c(z,"yes")
      }
    }

    nb3<-setFinding(nb2,nodes=c("A","T","L","B","E","X","D"),states=z)
    x<-querygrain(nb3,c("S"))

    if(x$S[1]>x$S[2]){
      y<-1
    }
    else{
      y<-2
    }

    if(te[i,2]=="no"){
      error[y,1]<-error[y,1]+1
    }
    else{
      error[y,2]<-error[y,2]+1
    }
  }
  totalError[k]<-(error[1,2]+error[2,1])/1000
  k<-k+1
```

```
}
totalError

nb0<-empty.graph(c("S","A","T","L","B","E","X","D"))
arc.set<-matrix(c("A", "S", "T", "S", "L", "S", "B", "S", "E", "S", "X", "S", "D", "S"),
                ncol = 2, byrow = TRUE, dimnames = list(NULL, c("from", "to")))
arcs(nb0)<-arc.set
#nb<-naive.bayes(tr,"S")
plot(nb0)

totalError<-array(dim=6)
k<-1
for(j in c(10,20,50,100,1000,2000)){
  nb<-bn.fit(nb0,tr[1:j,],method="bayes")
  nb2<-as.grain(nb)
  nb2<-compile(nb2)

  error<-matrix(c(0,0,0,0),nrow=2,ncol=2)
  for(i in 1:1000){
    z<-NULL
    for(j in c("A","T","L","B","E","X","D")){
      if(te[i,j]=="no"){
        z<-c(z,"no")
      }
      else{
        z<-c(z,"yes")
      }
    }

    nb3<-setFinding(nb2,nodes=c("A","T","L","B","E","X","D"),states=z)
    x<-querygrain(nb3,c("S"))

    if(x$S[1]>x$S[2]){
      y<-1
    }
    else{
      y<-2
    }

    if(te[i,2]=="no"){
      error[y,1]<-error[y,1]+1
    }
    else{
      error[y,2]<-error[y,2]+1
    }
  }
  totalError[k]<-(error[1,2]+error[2,1])/1000
  k<-k+1
}
totalError

# Discussion
# The NB classifier only needs to estimate the parameters for distributions of the form
```

```
# p(C) and P(A_i|C) where C is the class variable and A_i is a predictive attribute. The
# alternative model needs to estimate p(C) and P(C|A_1,...,A_n). Therefore, it requires
# more data to obtain reliable estimates (e.g. many of the parental combinations may not
# appear in the training data and this is actually why you should use method="bayes", to
# avoid zero relative frequency counters).This may hurt performance when little learning
# data is available. This is actually observed in the experiments above. However, when
# the size of the learning data increases, the alternative model should outperform NB, because
# the latter assumes that the attributes are independent given the class whereas the former
# does not. In other words, note that p(C|A_1,...,A_n) is proportional to P(A_1,...,A_n|C) p(C)
# by Bayes theorem. NB assumes that P(A_1,...,A_n|C) factorizes into a product of factors
# p(A_i|C) whereas the alternative model assumes nothing. The NB's assumption may hurt
# performance. This can be observed in the experiments.
# Question 1.1 (Monty Hall)

library(bnlearn)
library(gRain)

MH<-model2network("[D][P][M|D:P]") # Structure
cptD = matrix(c(1/3, 1/3, 1/3), ncol = 3, dimnames = list(NULL, c("D1", "D2", "D3"))) # Parameters
cptP = matrix(c(1/3, 1/3, 1/3), ncol = 3, dimnames = list(NULL, c("P1", "P2", "P3")))
cptM = c(
  0,.5,.5,
  0,0,1,
  0,1,0,
  0,0,1,
  .5,0,.5,
  1,0,0,
  0,1,0,
  1,0,0,
  .5,.5,0)
dim(cptM) = c(3, 3, 3)
dimnames(cptM) = list("M" = c("M1", "M2", "M3"), "D" =  c("D1", "D2", "D3"), "P" = c("P1", "P2", "P3"))
MHfit<-custom.fit(MH,list(D=cptD,P=cptP,M=cptM))
MHcom<-compile(as.grain(MHfit))

MHfitEv<-setFinding(MHcom,nodes=c(""),states=c("")) # Exact inference
querygrain(MHfitEv,c("P"))
MHfitEv<-setFinding(MHcom,nodes=c("D","M"),states=c("D1","M2"))
querygrain(MHfitEv,c("P"))
MHfitEv<-setFinding(MHcom,nodes=c("D","M"),states=c("D1","M3"))
querygrain(MHfitEv,c("P"))

table(cpdist(MHfit,"P",evidence=TRUE)) # Alternatively, one can use approximate inference
table(cpdist(MHfit,"P",(D=="D1" & M=="M2")))
table(cpdist(MHfit,"P",(D=="D1" & M=="M3")))

# Question 1.2 (XOR)

library(bnlearn)
library(gRain)

xor<-model2network("[A][B][C|A:B]") # Structure
cptA = matrix(c(0.5, 0.5), ncol = 2, dimnames = list(NULL, c("0", "1"))) # Parameters
```

```r
cptB = matrix(c(0.5, 0.5), ncol = 2, dimnames = list(NULL, c("0", "1")))
cptC = c(1,0,0,1,0,1,1,0)
dim(cptC) = c(2, 2, 2)
dimnames(cptC) = list("C" = c("0", "1"), "A" =  c("0", "1"), "B" = c("0", "1"))
xorfit<-custom.fit(xor,list(A=cptA,B=cptB,C=cptC))

for(i in 1:10){
  xordata<-rbn(xorfit,1000) # Sample
  xorhc<-hc(xordata,score="bic") # Learn
  plot(xorhc)
}

# The HC fails because the data comes from a distribution that is not faithful to the graph.
# In other words, the graph has an edge A -> C but A is marginally independent of C in the
# distribution. The same for B and C. And, of course, the same for A and B since there is no
# edge between them. So, the HC cannot find two dependent variables to link with an edge and,
# thus, it adds no edge to the initial empty graph.
```

part2

Omkar Bhutra (omkbh878)

9 January 2020

LAB 2 stuff

The purpose of the lab is to put in practice some of the concepts covered in the lectures. To do so, you are asked 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 sectors and decides with equal probability to stay in that sector or move to the next sector. You do not have direct observation of the robot. However, the robot is equipped with a tracking device that you can access. 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.

Question 1: Build a hidden Markov model (HMM) for the scenario described above

A robot moves around a ring which divided into 10 sectors. The robot is in one sector at any given time step and its equally probable for the robot to stay in the state as it is to move to the next state. The robot has a tracking device. If the robot is in sector i, the tracking device will report that the robot is in the sectors [i - 2, i + 2] with equal probability i.e P = 0.2 for being in each position of that range.

Create transition matrix where each row consists of: $P(Z^t|Z^t - 1), t = 1, ..., 10$

```
states <- paste("state",1:10,sep="")
symbols <- paste("symbol",1:10,sep="")
transition_vector <- c(0.5, 0.5, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0.5, 0.5, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0.5, 0.5, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0.5, 0.5, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0.5, 0.5, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0.5, 0.5, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0.5, 0.5, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.5, 0.5, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0.5, 0.5,
0.5, 0, 0, 0, 0, 0, 0, 0, 0, 0.5)

emission_vector <- 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)


transition_matrix <- matrix(data = transition_vector,nrow = 10,ncol = 10)

emission_matrix <- matrix(data = emission_vector,nrow = 10,ncol = 10)
# States are the hidden variables
```

```r
# Symbols are the observable variables
hmm <- initHMM(States = states,
Symbols = symbols,
startProbs = rep(0.1,10),
transProbs = transition_matrix,
emissionProbs = emission_matrix)
hmm
```

```
## $States
##  [1] "state1"  "state2"  "state3"  "state4"  "state5"  "state6"  "state7"
##  [8] "state8"  "state9"  "state10"
##
## $Symbols
##  [1] "symbol1"  "symbol2"  "symbol3"  "symbol4"  "symbol5"  "symbol6"
##  [7] "symbol7"  "symbol8"  "symbol9"  "symbol10"
##
## $startProbs
##  state1  state2  state3  state4  state5  state6  state7  state8  state9
##     0.1     0.1     0.1     0.1     0.1     0.1     0.1     0.1     0.1
## state10
##     0.1
##
## $transProbs
##          to
## from      state1 state2 state3 state4 state5 state6 state7 state8 state9
##   state1     0.5    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
##   state2     0.5    0.5    0.0    0.0    0.0    0.0    0.0    0.0    0.0
##   state3     0.0    0.5    0.5    0.0    0.0    0.0    0.0    0.0    0.0
##   state4     0.0    0.0    0.5    0.5    0.0    0.0    0.0    0.0    0.0
##   state5     0.0    0.0    0.0    0.5    0.5    0.0    0.0    0.0    0.0
##   state6     0.0    0.0    0.0    0.0    0.5    0.5    0.0    0.0    0.0
##   state7     0.0    0.0    0.0    0.0    0.0    0.5    0.5    0.0    0.0
##   state8     0.0    0.0    0.0    0.0    0.0    0.0    0.5    0.5    0.0
##   state9     0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.5    0.5
##   state10    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.5
##          to
## from      state10
##   state1      0.5
##   state2      0.0
##   state3      0.0
##   state4      0.0
##   state5      0.0
##   state6      0.0
##   state7      0.0
##   state8      0.0
##   state9      0.0
##   state10     0.5
##
## $emissionProbs
##          symbols
## states    symbol1 symbol2 symbol3 symbol4 symbol5 symbol6 symbol7 symbol8
##   state1      0.2     0.2     0.2     0.0     0.0     0.0     0.0     0.0
##   state2      0.2     0.2     0.2     0.2     0.0     0.0     0.0     0.0
##   state3      0.2     0.2     0.2     0.2     0.2     0.0     0.0     0.0
```

```
##   state4      0.0     0.2     0.2     0.2     0.2     0.2     0.0     0.0
##   state5      0.0     0.0     0.2     0.2     0.2     0.2     0.2     0.0
##   state6      0.0     0.0     0.0     0.2     0.2     0.2     0.2     0.2
##   state7      0.0     0.0     0.0     0.0     0.2     0.2     0.2     0.2
##   state8      0.0     0.0     0.0     0.0     0.0     0.2     0.2     0.2
##   state9      0.2     0.0     0.0     0.0     0.0     0.0     0.2     0.2
##   state10     0.2     0.2     0.0     0.0     0.0     0.0     0.0     0.2
##          symbols
## states    symbol9 symbol10
##   state1      0.2      0.2
##   state2      0.0      0.2
##   state3      0.0      0.0
##   state4      0.0      0.0
##   state5      0.0      0.0
##   state6      0.0      0.0
##   state7      0.2      0.0
##   state8      0.2      0.2
##   state9      0.2      0.2
##   state10     0.2      0.2
```

Question 2: Simulate the HMM for 100 time steps.

```
hmm_sim <- simHMM(hmm = hmm,length = 100)
hmm_sim
```

```
## $states
##   [1] "state7"  "state7"  "state7"  "state6"  "state6"  "state6"  "state6"
##   [8] "state6"  "state5"  "state5"  "state5"  "state4"  "state3"  "state2"
##  [15] "state2"  "state1"  "state10" "state10" "state9"  "state9"  "state9"
##  [22] "state9"  "state8"  "state8"  "state8"  "state7"  "state6"  "state6"
##  [29] "state5"  "state4"  "state4"  "state3"  "state3"  "state3"  "state3"
##  [36] "state3"  "state3"  "state3"  "state2"  "state2"  "state2"  "state1"
##  [43] "state1"  "state1"  "state1"  "state1"  "state1"  "state1"  "state1"
##  [50] "state1"  "state1"  "state1"  "state1"  "state1"  "state1"  "state1"
##  [57] "state10" "state10" "state10" "state9"  "state9"  "state9"  "state9"
##  [64] "state9"  "state9"  "state8"  "state7"  "state6"  "state6"  "state5"
##  [71] "state5"  "state4"  "state4"  "state4"  "state3"  "state3"  "state2"
##  [78] "state2"  "state2"  "state2"  "state2"  "state1"  "state1"  "state1"
##  [85] "state1"  "state1"  "state1"  "state10" "state10" "state9"  "state8"
##  [92] "state8"  "state8"  "state8"  "state8"  "state8"  "state7"  "state6"
##  [99] "state5"  "state5"
##
## $observation
##   [1] "symbol7"  "symbol7"  "symbol6"  "symbol6"  "symbol8"  "symbol8"
##   [7] "symbol7"  "symbol7"  "symbol6"  "symbol3"  "symbol5"  "symbol4"
##  [13] "symbol5"  "symbol3"  "symbol10" "symbol10" "symbol2"  "symbol8"
##  [19] "symbol1"  "symbol7"  "symbol1"  "symbol8"  "symbol8"  "symbol7"
##  [25] "symbol10" "symbol8"  "symbol5"  "symbol4"  "symbol6"  "symbol3"
##  [31] "symbol2"  "symbol3"  "symbol1"  "symbol5"  "symbol1"  "symbol2"
##  [37] "symbol1"  "symbol1"  "symbol2"  "symbol4"  "symbol4"  "symbol1"
##  [43] "symbol2"  "symbol2"  "symbol1"  "symbol2"  "symbol9"  "symbol10"
##  [49] "symbol3"  "symbol2"  "symbol1"  "symbol3"  "symbol10" "symbol9"
```

```
##  [55] "symbol10" "symbol9"  "symbol10" "symbol1"  "symbol9"  "symbol10"
##  [61] "symbol7"  "symbol10" "symbol10" "symbol8"  "symbol8"  "symbol9"
##  [67] "symbol5"  "symbol6"  "symbol7"  "symbol7"  "symbol7"  "symbol2"
##  [73] "symbol6"  "symbol3"  "symbol2"  "symbol4"  "symbol3"  "symbol4"
##  [79] "symbol4"  "symbol2"  "symbol2"  "symbol10" "symbol1"  "symbol3"
##  [85] "symbol10" "symbol9"  "symbol1"  "symbol2"  "symbol8"  "symbol9"
##  [91] "symbol9"  "symbol10" "symbol7"  "symbol8"  "symbol7"  "symbol7"
##  [97] "symbol5"  "symbol8"  "symbol6"  "symbol5"
```

Question 3: Discard the hidden states from the sample obtained above. Use the remaining observationsto compute the filtered and smoothed probability distributions for each of the 100 time points. Compute also the most probable path. Question 4: Compute the accuracy of the filtered and smoothed probability distributions, and of themost probable path. That is, compute the percentage of the true hidden states that are guessed by each method

Hint: Note that the function forward in the HMM package returns probabilities in log scale. You may need to use the functions exp and prop.table in order to obtain a normalized probability distribution. You may also want to use the functions apply and which.max to find out the most probable states. Finally, recall that you can compare two vectors A and B elementwise as A==B, and that the function table will count the number of times that the different elements in a vector occur in the vector.

Forward probabilities can be found in a hidden markov model with hidden states X upto observation at time t defined as the probability of observing the sequence of observations $e_1, ..., e_k$ and that the state at time t is X. That is: $f[X,t] := P(X|E_1 = e_1, ..., E_k = e_k)$

```
robot <- function(hiddenmarkovmodel,pars){
hmm_sim <- simHMM(hmm = hmm,length = 100)
hmm_obs <- hmm_sim$observation
hmm_states <- hmm_sim$states
#filter: forward function does the filtering and returns log probabilities
log_filter = forward(hmm = hmm,observation = hmm_obs)
filter = exp(log_filter)
#normalised probability distribution
filternormalised <- prop.table(filter,margin = 2)
# find out the most probable states
filternormalised_probable <- apply(filternormalised,MARGIN = 2, FUN = which.max)
accuracy_filtering <- sum(paste("state", filternormalised_probable, sep ="")
==hmm_states)/length(hmm_states)
#filternormalised
#accuracy_filtering
#smoothing using function posterior
smooth <- posterior(hmm,hmm_obs)
smoothnormalised <- prop.table(smooth, margin = 2)
smoothnormalised_probable <- apply(smoothnormalised,MARGIN = 2, FUN = which.max)
accuracy_smooth <- sum(paste("state", smoothnormalised_probable, sep ="")
==hmm_states)/length(hmm_states)
#smoothnormalised
#accuracy_smooth
#Finding the most probable path using viterbi algorithm
probable_path <- viterbi(hmm = hmm, observation = hmm_obs)
accuracy_probable_path <- sum(probable_path == hmm_states)/ length(hmm_states)
probable_path #most probable path
#accuracy_probable_path
if(pars == "filter"){
```

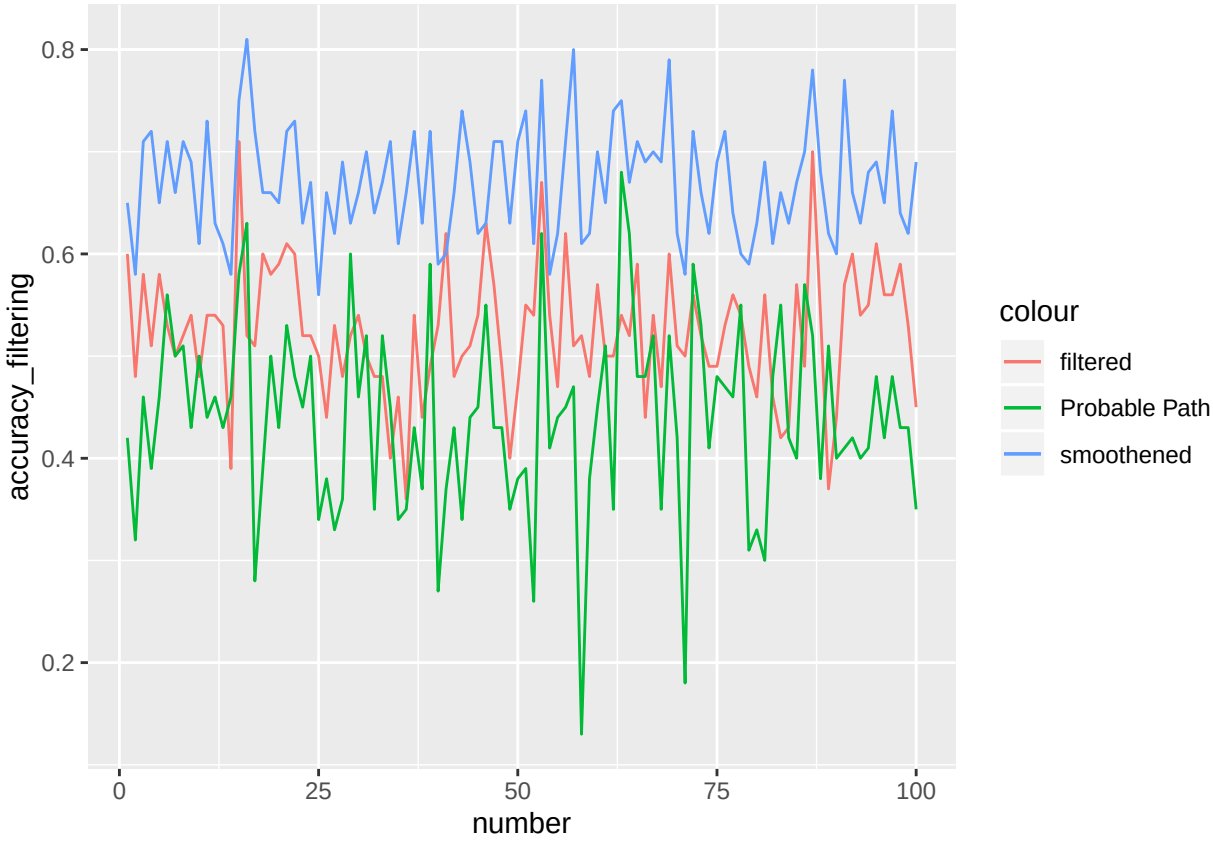
```
return(filternormalised)
}
if(pars == "smooth"){
return(smoothnormalised)
}
if(pars == "ProbablePath"){
return(probable_path)
}
if(pars == "accuracy"){
return(c(accuracy_filtering = accuracy_filtering,
accuracy_smooth = accuracy_smooth,
accuracy_probable_path = accuracy_probable_path))
}
}
```

Question 5: Repeat the previous exercise with different simulated samples. In general, the smoothed distributions should be more accurate than the filtered distributions. Why ? In general,the smoothed distributions should be more accurate than the most probable paths, too. Why ?

```
robot(hmm,"ProbablePath")
```

```
##   [1] "state8"  "state8"  "state7"  "state6"  "state6"  "state6"  "state5"
##   [8] "state4"  "state3"  "state2"  "state1"  "state1"  "state10" "state9"
##  [15] "state9"  "state9"  "state9"  "state8"  "state7"  "state6"  "state5"
##  [22] "state5"  "state4"  "state4"  "state3"  "state2"  "state1"  "state1"
##  [29] "state1"  "state1"  "state1"  "state1"  "state1"  "state1"  "state10"
##  [36] "state9"  "state8"  "state8"  "state8"  "state8"  "state8"  "state8"
##  [43] "state7"  "state6"  "state6"  "state6"  "state6"  "state6"  "state6"
##  [50] "state5"  "state5"  "state5"  "state5"  "state5"  "state4"  "state3"
##  [57] "state3"  "state3"  "state2"  "state2"  "state2"  "state1"  "state1"
##  [64] "state1"  "state10" "state9"  "state8"  "state7"  "state7"  "state6"
##  [71] "state6"  "state6"  "state5"  "state4"  "state3"  "state2"  "state2"
##  [78] "state2"  "state1"  "state1"  "state1"  "state10" "state9"  "state8"
##  [85] "state7"  "state6"  "state5"  "state5"  "state5"  "state5"  "state5"
##  [92] "state5"  "state5"  "state4"  "state3"  "state3"  "state3"  "state3"
##  [99] "state3"  "state3"
```

```
acc <- sapply(1:100,FUN = function(x){robot(hmm,"accuracy")})
acc <- data.frame(t(acc))
acc$number <- 1:100

ggplot(data = acc) + geom_line(aes(x=number,y=accuracy_filtering,col="filtered"))+
geom_line(aes(x=number,y=accuracy_smooth,col="smoothened"))+
geom_line(aes(x=number,y=accuracy_probable_path,col="Probable Path"))
```




```
colMeans(acc[,-4])
```

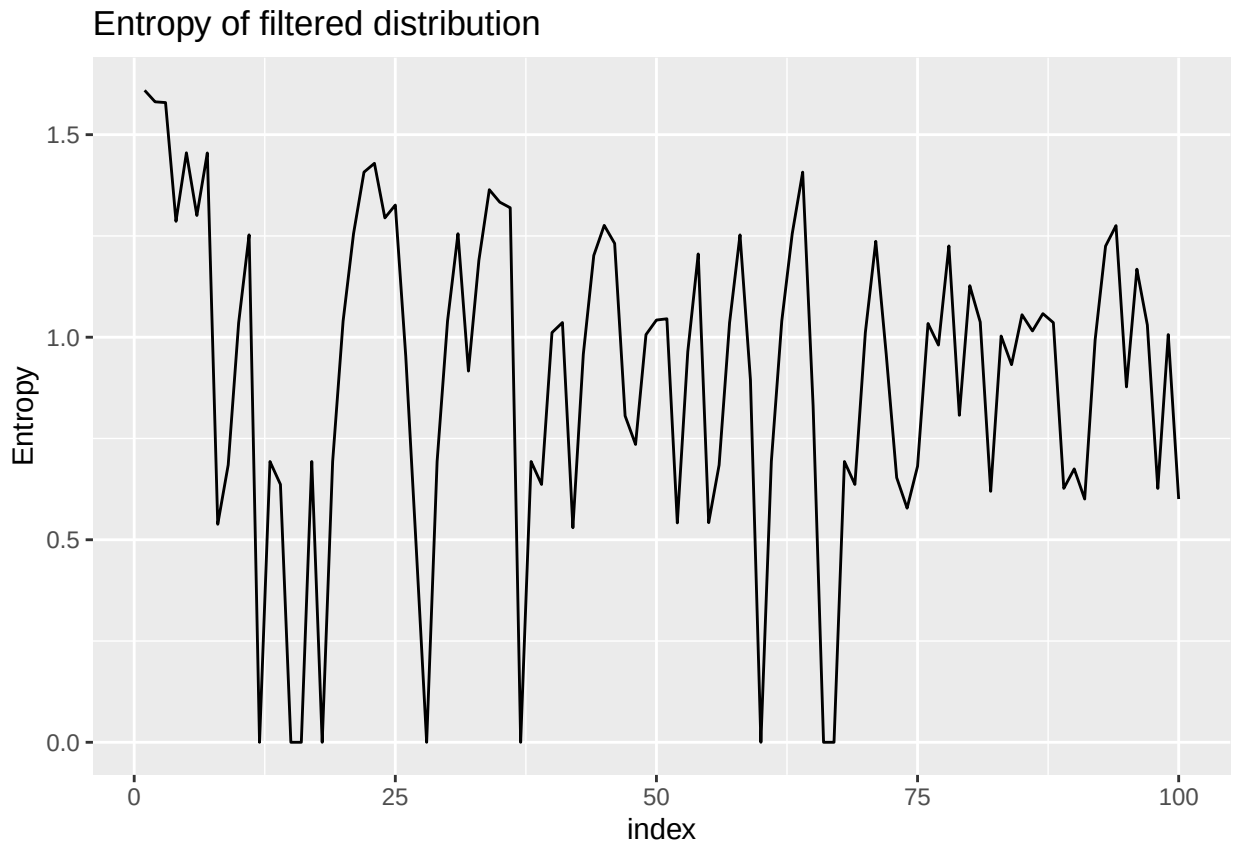
```
##     accuracy_filtering        accuracy_smooth accuracy_probable_path
##                 0.5234                 0.6704                 0.4394
```

The smoothed distribution is using more information i.e the whole set of observed emission x[0:T] whereas filtered only uses data emitted up to that point x[0:t] The most probable path generated by the viterbi algorithm is constrained by the transitions between states in the hidden variables. The smooth distribution approximates the most probable state and hence can make jumps from one state to another when predicting current state.

Question 6: Is it true that the more observations you have the better you know where the robot is ?

Hint: You may want to compute the entropy of the filtered distributions with the function entropy.empirical of the package entropy.

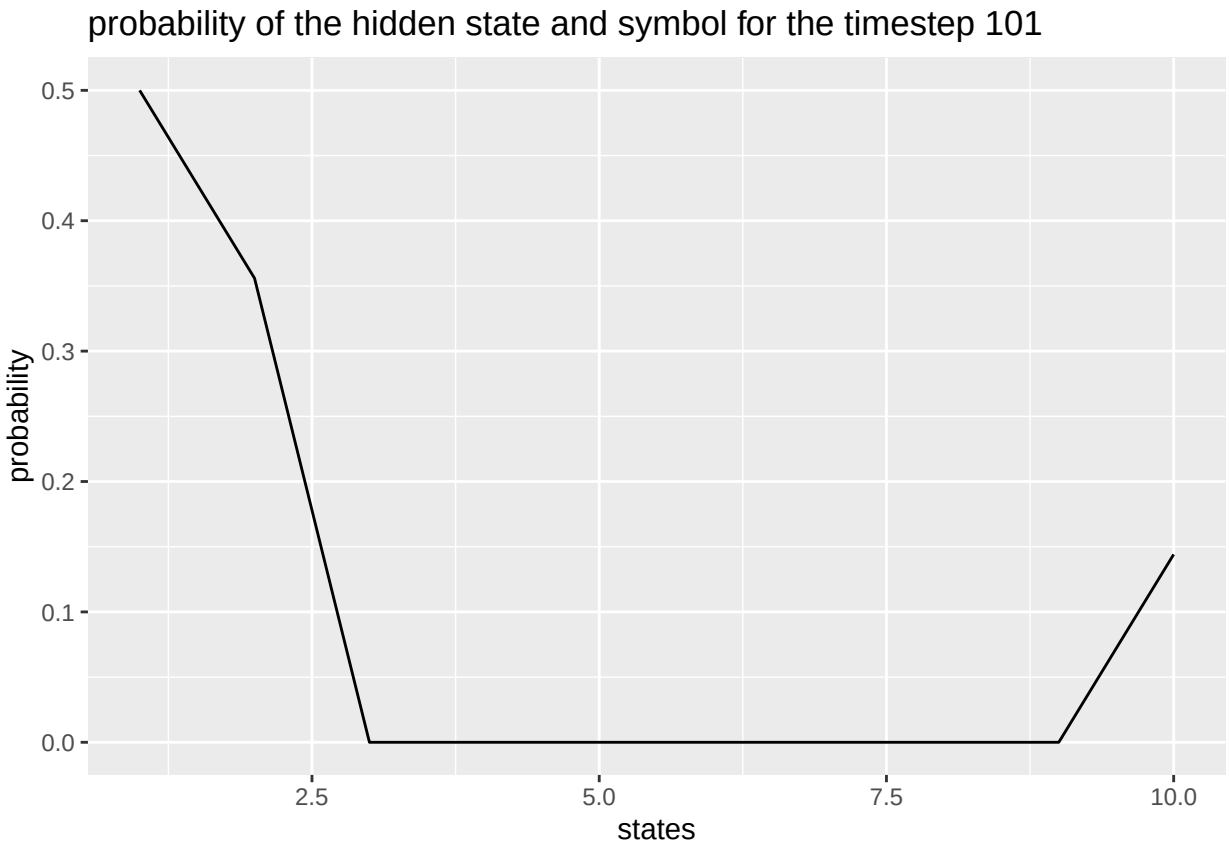
```
hmm_filter <- robot(hmm,"filter")
hmm_filter_entropy <- data.frame(index = 1:100, Entropy = apply(hmm_filter, MARGIN = 2,
FUN = entropy::entropy.empirical))
ggplot(hmm_filter_entropy,aes(x = index, y = Entropy))+
geom_line()+ggtitle("Entropy of filtered distribution")
```



The entropy is random even when we increase the number of observations added to the hidden markov model. This is because the HMM is Markovian and only depends on the previous observation.

Question 7: Consider any of the samples above of length 100. Compute the probabilities of the hidden states for the time step 101.

```
posterior <- hmm_filter[,100]
# matrix multiplication
probability <- as.data.frame(hmm$transProbs %*% posterior)
ggplot(probability)+geom_line(aes(x=1:10,y=V1))+ggtitle("probability of the hidden state and symbol for
```



The Markovian assumption is that only relavant state is the current state. All previous states feeds though the 100 states and provides us information about the states and observations. On multiplication with transition probabilities to get the probability of the hidden state and symbol for the timestep 101. That the entropy of the filtered distributions does not decrease monotonically. It has to do with the fact that you get noisy observations and, thus, you will more often than not be uncertain as to where the robot is.

```
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(echo = TRUE)
library("dplyr")
library("ggplot2")
library("gRain")
library("bnlearn")
library("HMM")
library("entropy")
states <- paste("state",1:10,sep="")
symbols <- paste("symbol",1:10,sep="")
transition_vector <- c(0.5, 0.5, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0.5, 0.5, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0.5, 0.5, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0.5, 0.5, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0.5, 0.5, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0.5, 0.5, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0.5, 0.5, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.5, 0.5, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0.5, 0.5,
0.5, 0, 0, 0, 0, 0, 0, 0, 0, 0.5)
```

```
emission_vector <- 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)


transition_matrix <- matrix(data = transition_vector,nrow = 10,ncol = 10)

emission_matrix <- matrix(data = emission_vector,nrow = 10,ncol = 10)
# States are the hidden variables
# Symbols are the observable variables
hmm <- initHMM(States = states,
Symbols = symbols,
startProbs = rep(0.1,10),
transProbs = transition_matrix,
emissionProbs = emission_matrix)
hmm
hmm_sim <- simHMM(hmm = hmm,length = 100)
hmm_sim
robot <- function(hiddenmarkovmodel,pars){
hmm_sim <- simHMM(hmm = hmm,length = 100)
hmm_obs <- hmm_sim$observation
hmm_states <- hmm_sim$states
#filter: forward function does the filtering and returns log probabilities
log_filter = forward(hmm = hmm,observation = hmm_obs)
filter = exp(log_filter)
#normalised probability distribution
filternormalised <- prop.table(filter,margin = 2)
# find out the most probable states
filternormalised_probable <- apply(filternormalised,MARGIN = 2, FUN = which.max)
accuracy_filtering <- sum(paste("state", filternormalised_probable, sep ="")
==hmm_states)/length(hmm_states)
#filternormalised
#accuracy_filtering
#smoothing using function posterior
smooth <- posterior(hmm,hmm_obs)
smoothnormalised <- prop.table(smooth, margin = 2)
smoothnormalised_probable <- apply(smoothnormalised,MARGIN = 2, FUN = which.max)
accuracy_smooth <- sum(paste("state", smoothnormalised_probable, sep ="")
==hmm_states)/length(hmm_states)
#smoothnormalised
#accuracy_smooth
#Finding the most probable path using viterbi algorithm
probable_path <- viterbi(hmm = hmm, observation = hmm_obs)
accuracy_probable_path <- sum(probable_path == hmm_states)/ length(hmm_states)
probable_path #most probable path
#accuracy_probable_path
```

```
if(pars == "filter"){
return(filternormalised)
}
if(pars == "smooth"){
return(smoothnormalised)
}
if(pars == "ProbablePath"){
return(probable_path)
}
if(pars == "accuracy"){
return(c(accuracy_filtering = accuracy_filtering,
accuracy_smooth = accuracy_smooth,
accuracy_probable_path = accuracy_probable_path))
}
}
robot(hmm,"filter")
robot(hmm,"smooth")
robot(hmm,"ProbablePath")

acc <- sapply(1:100,FUN = function(x){robot(hmm,"accuracy")})
acc <- data.frame(t(acc))
acc$number <- 1:100

ggplot(data = acc) + geom_line(aes(x=number,y=accuracy_filtering,col="filtered"))+
geom_line(aes(x=number,y=accuracy_smooth,col="smoothened"))+
geom_line(aes(x=number,y=accuracy_probable_path,col="Probable Path"))

colMeans(acc[,-4])
hmm_filter <- robot(hmm,"filter")
hmm_filter_entropy <- data.frame(index = 1:100, Entropy = apply(hmm_filter, MARGIN = 2,
FUN = entropy::entropy.empirical))
ggplot(hmm_filter_entropy,aes(x = index, y = Entropy))+
geom_line()+ggtitle("Entropy of filtered distribution")
posterior <- hmm_filter[,100]
# matrix multiplication
probability <- as.data.frame(hmm$transProbs %*% posterior)
ggplot(probability)+geom_line(aes(x=1:10,y=V1))+ggtitle("probability of the hidden state and symbol for
# Question 2: HMMs

# Build the HMM.

library(HMM)
#set.seed(123)

States<-1:100
Symbols<-1:2 # 1=door

transProbs<-matrix(rep(0,length(States)*length(States)), nrow=length(States), ncol=length(States), byrou
for(i in 1:99){
  transProbs[i,i]<-.1
  transProbs[i,i+1]<-.9
}
```

```
emissionProbs<-matrix(rep(0,length(States)*length(Symbols)), nrow=length(States), ncol=length(Symbols),
for(i in States){
  if(i %in% c(10,11,12,20,21,22,30,31,32)){
    emissionProbs[i,1]<-.9
    emissionProbs[i,2]<-.1
  }
  else{
    emissionProbs[i,1]<-.1
    emissionProbs[i,2]<-.9
  }
}

startProbs<-rep(1/100,100)
hmm<-initHMM(States,Symbols,startProbs,transProbs,emissionProbs)

# If the robot observes a door, it can be in front of any of the three doors. If it then observes a lon
# sequence of non-doors, then it know that it was in front of the third door.

obs<-c(1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2)
pt<-prop.table(exp(forward(hmm,obs)),2)

which.maxima<-function(x){ # This function is needed since which.max only returns the first maximum.
  return(which(x==max(x)))
}

apply(pt,2,which.maxima)
# HMMs

library(HMM)
library(entropy)
set.seed(567)
States=1:10
Symbols=1:10
transProbs=matrix(c(.5,.5,0,0,0,0,0,0,0,0,
                    0,.5,.5,0,0,0,0,0,0,0,
                    0,0,.5,.5,0,0,0,0,0,0,
                    0,0,0,.5,.5,0,0,0,0,0,
                    0,0,0,0,.5,.5,0,0,0,0,
                    0,0,0,0,0,.5,.5,0,0,0,
                    0,0,0,0,0,0,.5,.5,0,0,
                    0,0,0,0,0,0,0,.5,.5,0,
                    0,0,0,0,0,0,0,0,.5,.5,
                    .5,0,0,0,0,0,0,0,0,.5), nrow=length(States), ncol=length(States), byrow = TRUE)
emissionProbs=matrix(c(.2,.2,.2,0,0,0,0,0,.2,.2,
                       .2,.2,.2,.2,0,0,0,0,0,.2,
                       .2,.2,.2,.2,.2,0,0,0,0,0,
                       0,.2,.2,.2,.2,.2,0,0,0,0,
                       0,0,.2,.2,.2,.2,.2,0,0,0,
                       0,0,0,.2,.2,.2,.2,.2,0,0,
                       0,0,0,0,.2,.2,.2,.2,.2,0,
                       0,0,0,0,0,.2,.2,.2,.2,.2,
                       .2,0,0,0,0,0,.2,.2,.2,.2,
                       .2,.2,0,0,0,0,0,.2,.2,.2), nrow=length(States), ncol=length(States), byrow = TRU
```

```
startProbs=c(.1,.1,.1,.1,.1,.1,.1,.1,.1,.1)
hmm=initHMM(States,Symbols,startProbs,transProbs,emissionProbs)
sim=simHMM(hmm,100)
logf=forward(hmm,sim$observation[1:100])
ef=exp(logf)
pt=prop.table(ef,2)
maxpt=apply(pt,2,which.max)
table(maxpt==sim$states)
post=posterior(hmm,sim$observation[1:100])
maxpost=apply(post,2,which.max)
table(maxpost==sim$states)

# Forward phase

a<-matrix(NA,nrow=100, ncol=length(States))
for(i in States){
  a[1,i]<-emissionProbs[sim$observation[1],i]*startProbs[i]
}

for(t in 2:100){
  for(i in States){
    a[t,i]<-emissionProbs[i,sim$observation[t]]*sum(a[t-1,]*transProbs[,i])
  }
}

for(t in 1:100){
  a[t,]<-a[t,]/sum(a[t,])
}

maxa=apply(a,1,which.max)
table(maxa==sim$states)

# Backward phase

b<-matrix(NA,nrow=100, ncol=length(States))
for(i in States){
  b[100,i]<-1
}

for(t in 99:1){
  for(i in States){
    b[t,i]<-sum(b[t+1,]*emissionProbs[,sim$observation[t+1]]*transProbs[i,])
  }
}

for(t in 1:100){
  for(i in States){
    b[t,i]<-b[t,i]*a[t,i]
  }
  b[t,]<-b[t,]/sum(b[t,])
}

maxb=apply(b,1,which.max)
```

```r
table(maxb==sim$states)

# Question 2 (HMMs)

library(HMM)

States=1:10 # Sectors
Symbols=1:11 # Sectors + malfunctioning
transProbs=matrix(c(.5,.5,0,0,0,0,0,0,0,0,
                    0,.5,.5,0,0,0,0,0,0,0,
                    0,0,.5,.5,0,0,0,0,0,0,
                    0,0,0,.5,.5,0,0,0,0,0,
                    0,0,0,0,.5,.5,0,0,0,0,
                    0,0,0,0,0,.5,.5,0,0,0,
                    0,0,0,0,0,0,.5,.5,0,0,
                    0,0,0,0,0,0,0,.5,.5,0,
                    0,0,0,0,0,0,0,0,.5,.5,
                    .5,0,0,0,0,0,0,0,0,.5), nrow=length(States), ncol=length(States), byrow = TRUE)
emissionProbs=matrix(c(.1,.1,.1,0,0,0,0,0,.1,.1,.5,
                       .1,.1,.1,.1,0,0,0,0,0,.1,.5,
                       .1,.1,.1,.1,.1,0,0,0,0,0,.5,
                       0,.1,.1,.1,.1,.1,0,0,0,0,.5,
                       0,0,.1,.1,.1,.1,.1,0,0,0,.5,
                       0,0,0,.1,.1,.1,.1,.1,0,0,.5,
                       0,0,0,0,.1,.1,.1,.1,.1,0,.5,
                       0,0,0,0,0,.1,.1,.1,.1,.1,.5,
                       .1,0,0,0,0,0,.1,.1,.1,.1,.5,
                       .1,.1,0,0,0,0,0,.1,.1,.1,.5), nrow=length(States), ncol=length(Symbols), byrow =
startProbs=c(.1,.1,.1,.1,.1,.1,.1,.1,.1,.1)
hmm=initHMM(States,Symbols,startProbs,transProbs,emissionProbs)

obs=c(1,11,11,11)
posterior(hmm,obs)
viterbi(hmm,obs)
```

part3

Omkar Bhutra (omkbh878)

9 January 2020

Lab 3 stuff

Q1. The purpose of the lab is to put in practice some of the concepts covered in the lectures. To do so, you are asked to implement the particle filter for robot localization. For the particle filter algorithm, please check Section 13.3.4 of Bishops book and/or the slides for the last lecture on state space models (SSMs). The robot moves along the horizontal axis according to the following SSM:

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

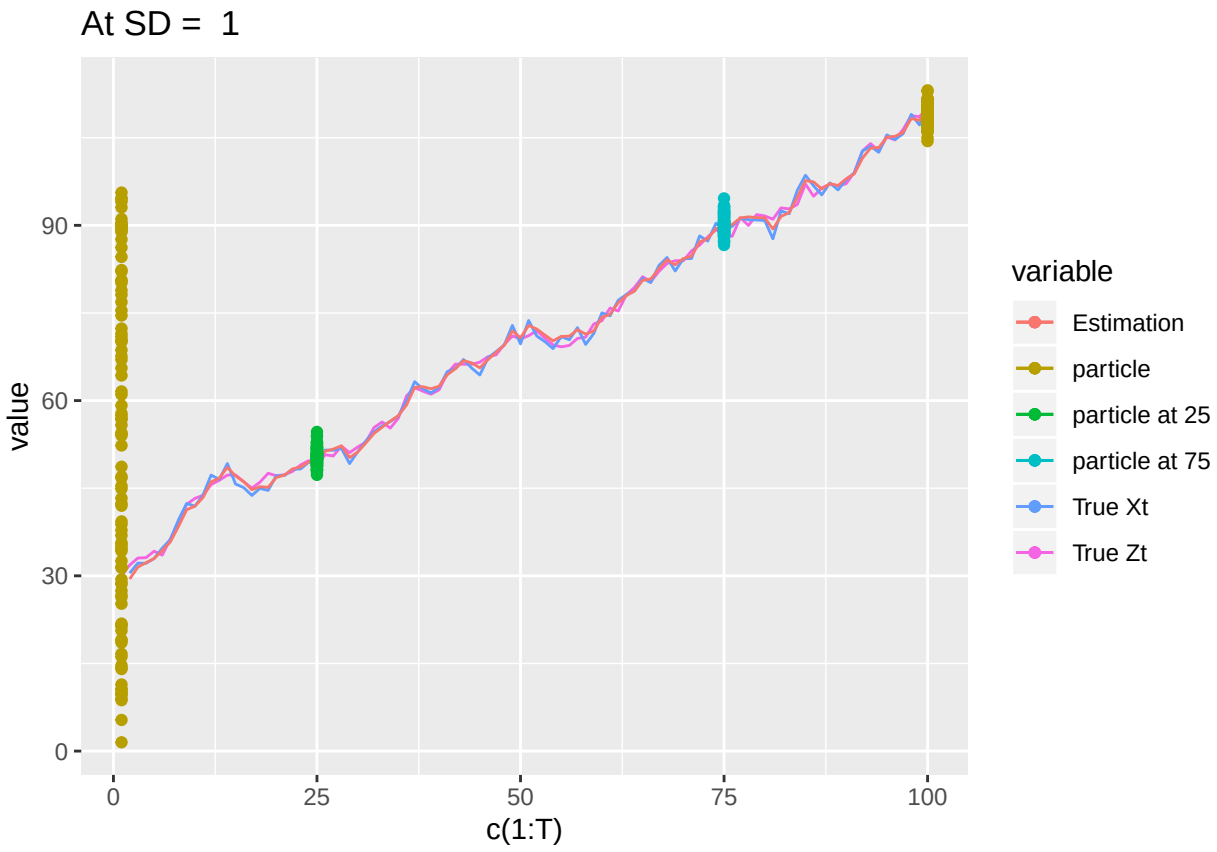
Initial Model: $p(z_1) = Uniform(0, 100)$

A) Implement the SSM above. Simulate it for T = 100 time steps to obtain z 1:100 (i.e., states) and x 1:100 (i.e., observations). Use the observations (i.e., sensor readings) to identify the state (i.e., robot location) via particle filtering. Use 100 particles. Show the particles, the expected location and the true location for the first and last time steps, as well as for two intermediate time steps of your choice.

```
initModel <- function(len){
  x <- runif(len,0,100)
  return(x)
}

transitionmodel <- function(zt){
  probs = rep(1/3,3)
  draw = sample(1:3,1,prob = probs)
  if(draw==1){
    normTrans <- rnorm(1,zt,transition_sd)
  }
  else if(draw==2){
    normTrans <- rnorm(1,zt+1,transition_sd)
  }
  else{
    normTrans <- rnorm(1,zt+2,transition_sd)
  }
  return(normTrans)
}
```

```r
emmisionmodel <- function(zt,emission_sd){
  probs = rep(1/3,3)
  draw = sample(1:3,1,prob = probs)

  if(draw==1){
    normEmis <- rnorm(1,zt,emission_sd)
  }
  else if(draw==2){
    normEmis <- rnorm(1,zt-1,emission_sd)
  }
  else{
    normEmis <- rnorm(1,zt+1,emission_sd)
  }
  return(normEmis)
}

emission_density <- function(xt,zt){
  x <- (dnorm(xt,zt,emission_sd)+dnorm(xt,zt-1,emission_sd)+dnorm(xt,zt+1,emission_sd))/3
  return(x)
}


emission_sd = 1
transition_sd = 1

ssm <- function(emission_sd,transition_sd){
  T = 100
  particles = 100
  zt = xt = error = rep(NA,T)

  # for zt and xt
  zt[1] = initModel(1)
  for(i in 2:T){
    zt[i] <- transitionmodel(zt[i-1])
    xt[i] <- emmisionmodel(zt[i],emission_sd)
  }

  initParticles <- initModel(particles)
  particleWeights<- rep(1/particles,particles)
  estimation <- c()
  Particles25 = Particles75 = NA

  for(i in 2:T){
    newParticles <- sample(1:particles,particles,prob=particleWeights,replace = T)
    initParticles <- sapply(initParticles[newParticles],transitionmodel)

    if(i == 25){
      Particles25 <- c(initParticles)
    }
    else if(i == 75){
      Particles75 <- c(initParticles)
    }
```

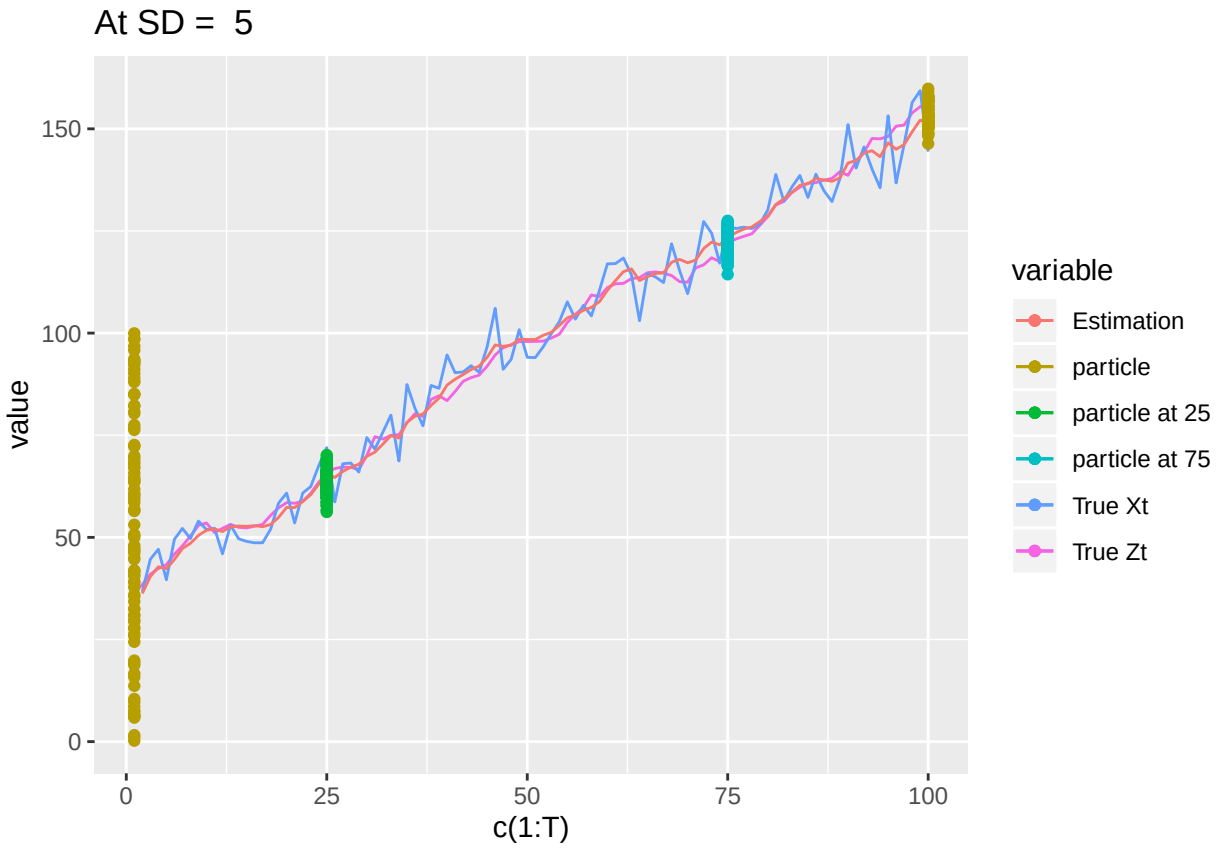
```
    for(j in 1:particles){
      particleWeights[j] <- emission_density(xt[i],initParticles[j])
    }
    #Normalise
    particleWeights <- particleWeights/sum(particleWeights)

    Ezt <- sum(particleWeights*initParticles)
    error[i] <- abs(zt[i]-Ezt)
    estimation[i] <- sum(particleWeights * initParticles)
  }

  data = data.frame(zt,xt,estimation,particles = initModel(particles),Particles25,Particles75,initParti

  ggplot(data,aes(x=c(1:T),y=value,color=variable,xlab="timestep"))+
    geom_line(aes(y=data$zt,col='True Zt'))+
    geom_line(aes(y=data$xt,col='True Xt'))+
    geom_line(aes(y=data$estimation,col='Estimation'))+
    geom_point(aes(x=rep(1,T),y=data$particles,col='particle'))+
    geom_point(aes(x=rep(25,T),y=data$Particles25,col='particle at 25'))+
    geom_point(aes(x=rep(75,T),y=data$Particles75,col='particle at 75'))+
    geom_point(aes(x=rep(T,T),y=data$initParticles,col='particle'))+
    ggtitle(paste('At SD = ',emission_sd))
}
ssm(emission_sd ,transition_sd )
```



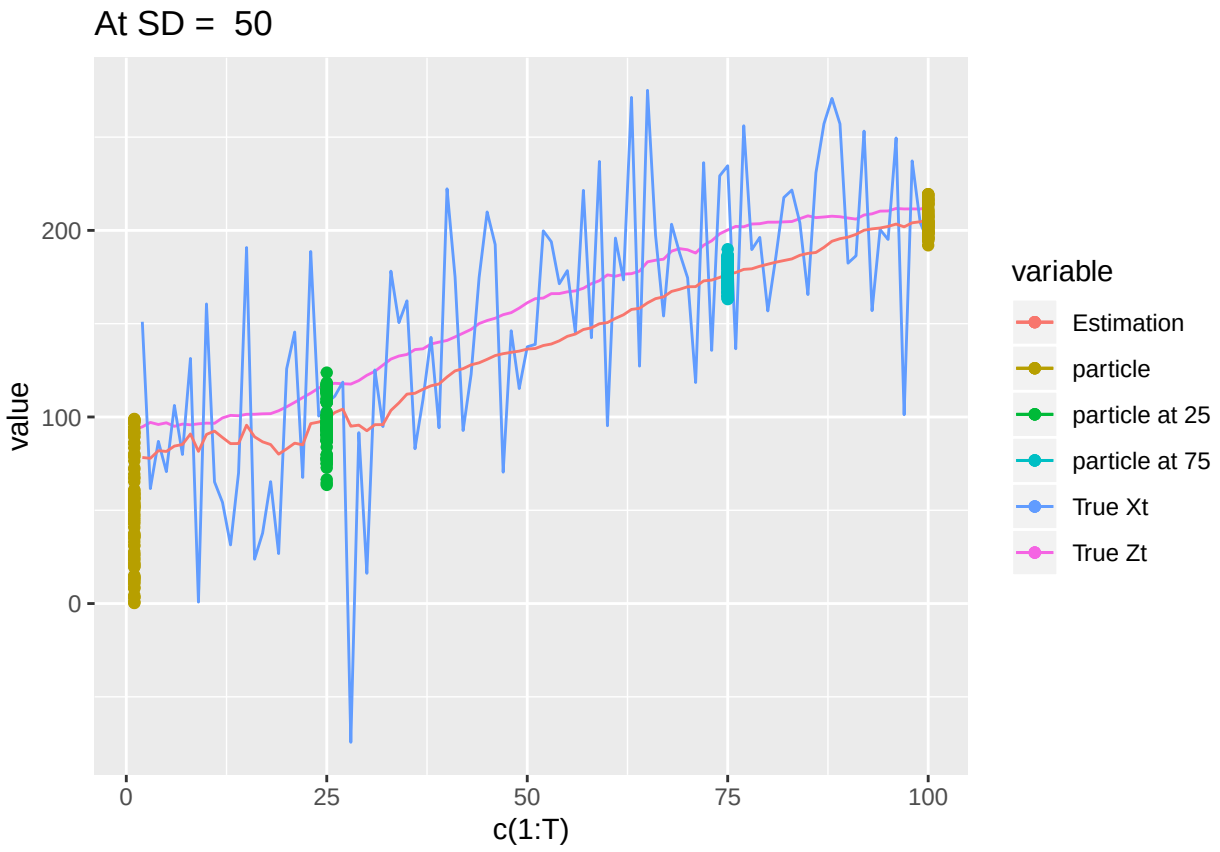
Intermediate particles number 25 and 75 are chosen and their locations are mapped in the figure.

B)Repeat the exercise above replacing the standard deviation of the emission model with 5 and then with 50. Comment on how this affects the results.

```
emission_sd = 5
ssm(emission_sd ,transition_sd )
```




```
emission_sd = 50
ssm(emission_sd ,transition_sd )
```

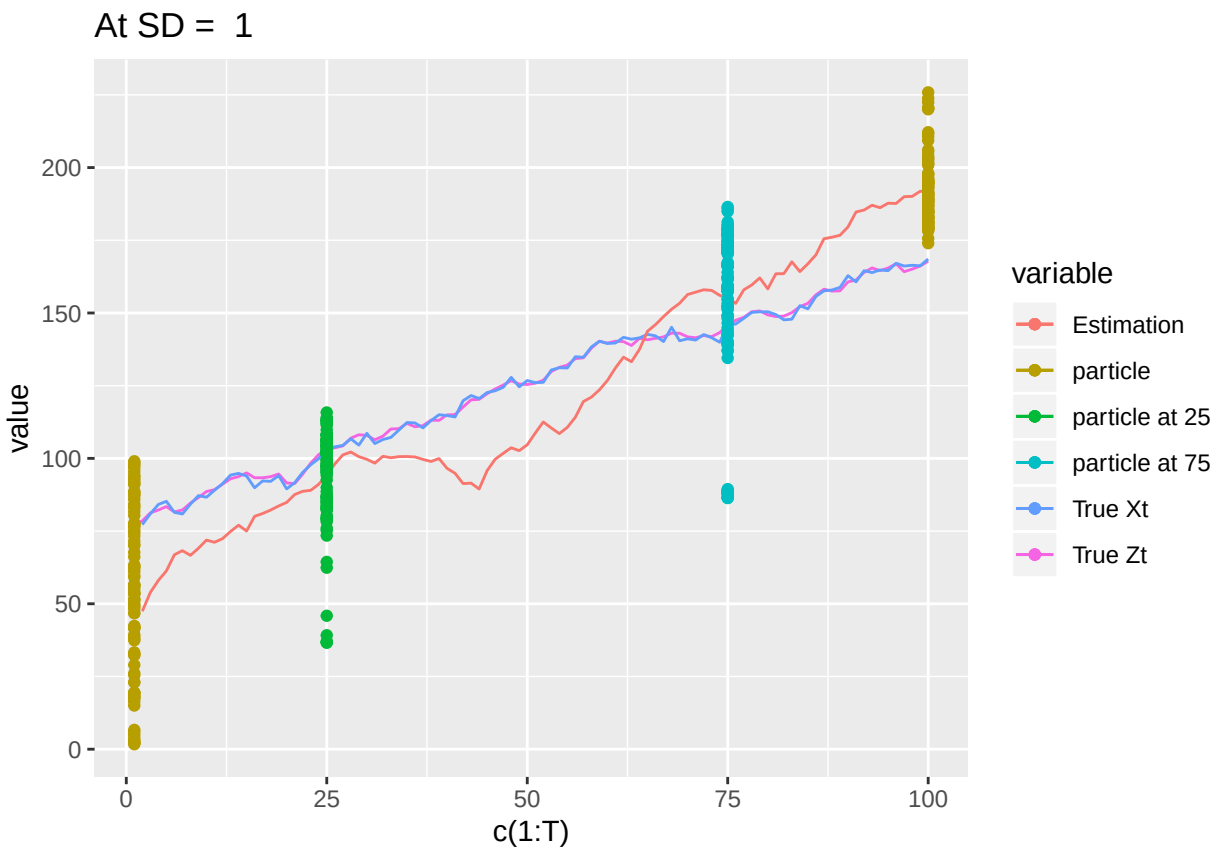


Increasing the standard deviation not only makes the true x_t scattered but also scatters the particles both intermediate and endstep.

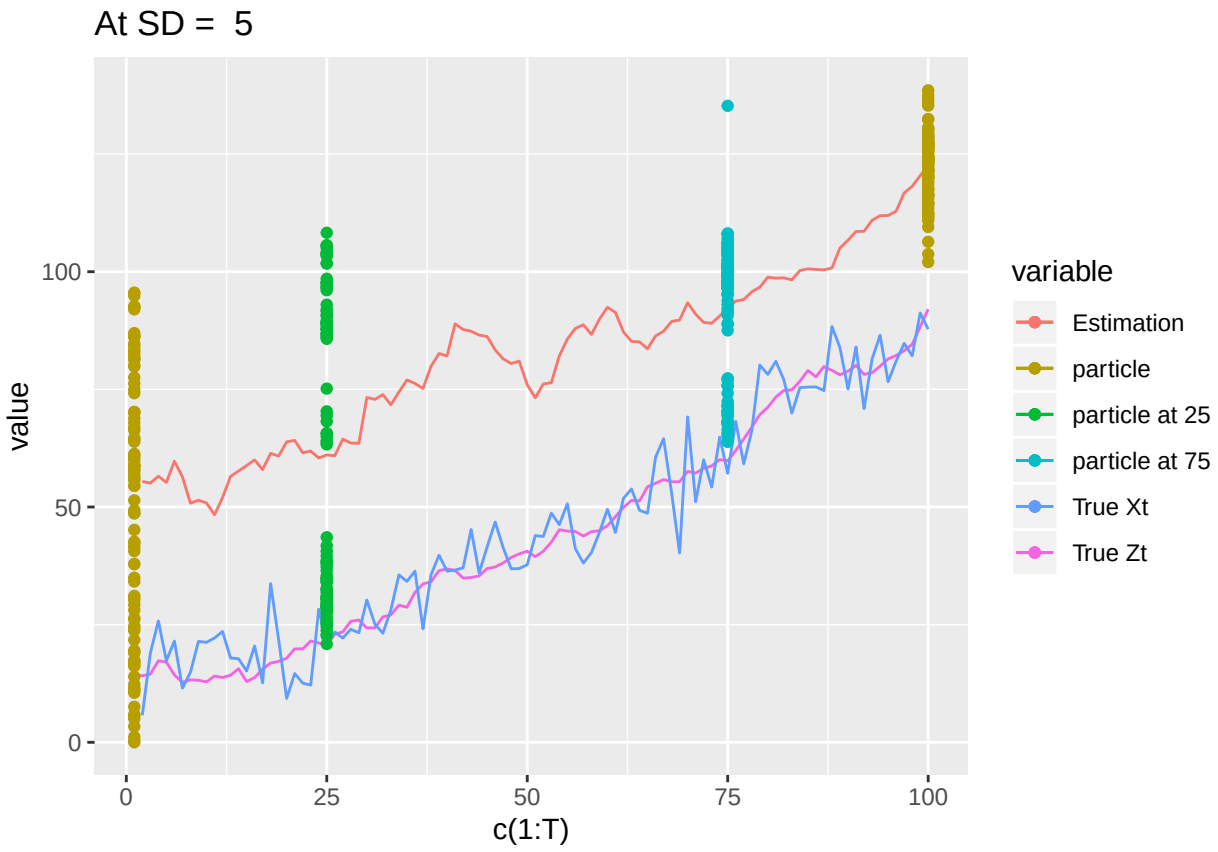
C) Finally, show and explain what happens when the weights in the particle filter are always equal to 1, i.e. there is no correction.

```
ssm <- function(emission_sd,transition_sd){
  T = 100
  particles = 100
  zt = xt = error = rep(NA,T)

  # for zt and xt
  zt[1] = initModel(1)
  for(i in 2:T){
    zt[i] <- transitionmodel(zt[i-1])
    xt[i] <- emmisionmodel(zt[i],emission_sd)
  }

  initParticles <- initModel(particles)
  particleWeights<- rep(1/particles,particles)
  estimation <- c()
  Particles25 = Particles75 = NA

  for(i in 2:T){
```

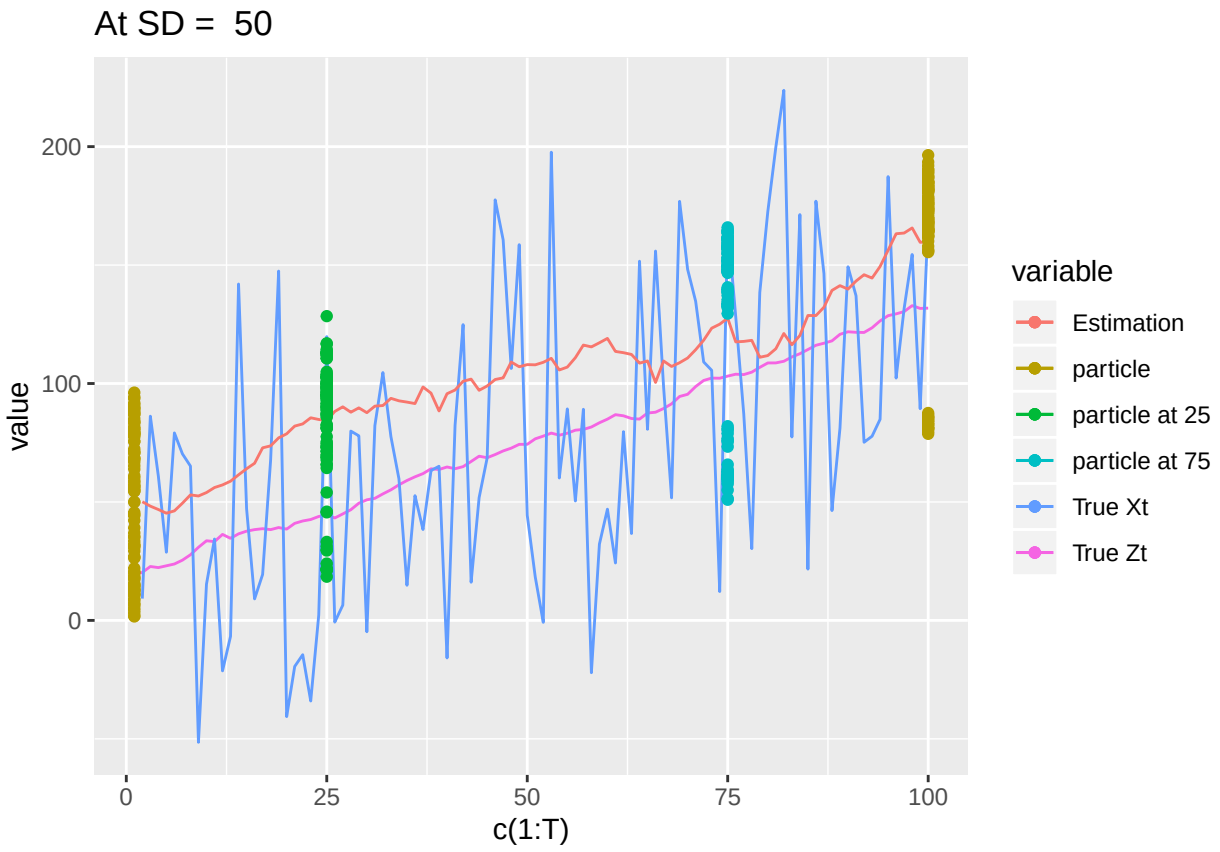
```
    newParticles <- sample(1:particles,particles,prob=particleWeights,replace = T)
    initParticles <- sapply(initParticles[newParticles],transitionmodel)

    if(i == 25){
      Particles25 <- c(initParticles)
    }
    else if(i == 75){
      Particles75 <- c(initParticles)
    }


    for(j in 1:particles){
      particleWeights[j] <- 1
    }
    #Normalise
    particleWeights <- particleWeights/sum(particleWeights)

    Ezt <- sum(particleWeights*initParticles)
    error[i] <- abs(zt[i]-Ezt)
    estimation[i] <- sum(particleWeights * initParticles)
  }

  data = data.frame(zt,xt,estimation,particles = initModel(particles),Particles25,Particles75,initParti

  ggplot(data,aes(x=c(1:T),y=value,color=variable,xlab="timestep"))+
    geom_line(aes(y=data$zt,col='True Zt'))+
    geom_line(aes(y=data$xt,col='True Xt'))+
    geom_line(aes(y=data$estimation,col='Estimation'))+
    geom_point(aes(x=rep(1,T),y=data$particles,col='particle'))+
    geom_point(aes(x=rep(25,T),y=data$Particles25,col='particle at 25'))+
    geom_point(aes(x=rep(75,T),y=data$Particles75,col='particle at 75'))+
    geom_point(aes(x=rep(T,T),y=data$initParticles,col='particle'))+
    ggtitle(paste('At SD = ',emission_sd))
}

emission_sd = 1
ssm(emission_sd ,transition_sd )
```




```
emission_sd = 5
ssm(emission_sd ,transition_sd )
```




```
emission_sd = 50
ssm(emission_sd ,transition_sd )
```



It is seen that the estimation and true values do not match and accuracy is reduced. This change is consistent from all the standard deviation runs of 1,5 and 50. value of the particle at all timesteps are scattered as compared to earlier runs. It reduces the quality of the filtering.

```
knitr::opts_chunk$set(echo = TRUE)
library("dplyr")
library("ggplot2")
library("KFAS")
initModel <- function(len){
  x <- runif(len,0,100)
  return(x)
}

transitionmodel <- function(zt){
  probs = rep(1/3,3)
  draw = sample(1:3,1,prob = probs)
  if(draw==1){
    normTrans <- rnorm(1,zt,transition_sd)
  }
  else if(draw==2){
    normTrans <- rnorm(1,zt+1,transition_sd)
  }
  else{
    normTrans <- rnorm(1,zt+2,transition_sd)
  }
  return(normTrans)
```

```
}

emmisionmodel <- function(zt,emission_sd){
  probs = rep(1/3,3)
  draw = sample(1:3,1,prob = probs)

  if(draw==1){
    normEmis <- rnorm(1,zt,emission_sd)
  }
  else if(draw==2){
    normEmis <- rnorm(1,zt-1,emission_sd)
  }
  else{
    normEmis <- rnorm(1,zt+1,emission_sd)
  }
  return(normEmis)
}

emission_density <- function(xt,zt){
  x <- (dnorm(xt,zt,emission_sd)+dnorm(xt,zt-1,emission_sd)+dnorm(xt,zt+1,emission_sd))/3
  return(x)
}


emission_sd = 1
transition_sd = 1

ssm <- function(emission_sd,transition_sd){
  T = 100
  particles = 100
  zt = xt = error = rep(NA,T)

  # for zt and xt
  zt[1] = initModel(1)
  for(i in 2:T){
    zt[i] <- transitionmodel(zt[i-1])
    xt[i] <- emmisionmodel(zt[i],emission_sd)
  }

  initParticles <- initModel(particles)
  particleWeights<- rep(1/particles,particles)
  estimation <- c()
  Particles25 = Particles75 = NA

  for(i in 2:T){
    newParticles <- sample(1:particles,particles,prob=particleWeights,replace = T)
    initParticles <- sapply(initParticles[newParticles],transitionmodel)

    if(i == 25){
      Particles25 <- c(initParticles)
    }
    else if(i == 75){
      Particles75 <- c(initParticles)
```

```
    }


    for(j in 1:particles){
      particleWeights[j] <- emission_density(xt[i],initParticles[j])
    }
    #Normalise
    particleWeights <- particleWeights/sum(particleWeights)

    Ezt <- sum(particleWeights*initParticles)
    error[i] <- abs(zt[i]-Ezt)
    estimation[i] <- sum(particleWeights * initParticles)
  }

  data = data.frame(zt,xt,estimation,particles = initModel(particles),Particles25,Particles75,initParti

  ggplot(data,aes(x=c(1:T),y=value,color=variable,xlab="timestep"))+
    geom_line(aes(y=data$zt,col='True Zt'))+
    geom_line(aes(y=data$xt,col='True Xt'))+
    geom_line(aes(y=data$estimation,col='Estimation'))+
    geom_point(aes(x=rep(1,T),y=data$particles,col='particle'))+
    geom_point(aes(x=rep(25,T),y=data$Particles25,col='particle at 25'))+
    geom_point(aes(x=rep(75,T),y=data$Particles75,col='particle at 75'))+
    geom_point(aes(x=rep(T,T),y=data$initParticles,col='particle'))+
    ggtitle(paste('At SD = ',emission_sd))
}
ssm(emission_sd ,transition_sd )
emission_sd = 5
ssm(emission_sd ,transition_sd )

emission_sd = 50
ssm(emission_sd ,transition_sd )
ssm <- function(emission_sd,transition_sd){
  T = 100
  particles = 100
  zt = xt = error = rep(NA,T)

  # for zt and xt
  zt[1] = initModel(1)
  for(i in 2:T){
    zt[i] <- transitionmodel(zt[i-1])
    xt[i] <- emmisionmodel(zt[i],emission_sd)
  }

  initParticles <- initModel(particles)
  particleWeights<- rep(1/particles,particles)
  estimation <- c()
  Particles25 = Particles75 = NA

  for(i in 2:T){
    newParticles <- sample(1:particles,particles,prob=particleWeights,replace = T)
    initParticles <- sapply(initParticles[newParticles],transitionmodel)
```

```
    if(i == 25){
      Particles25 <- c(initParticles)
    }
    else if(i == 75){
      Particles75 <- c(initParticles)
    }


    for(j in 1:particles){
      particleWeights[j] <- 1
    }
    #Normalise
    particleWeights <- particleWeights/sum(particleWeights)

    Ezt <- sum(particleWeights*initParticles)
    error[i] <- abs(zt[i]-Ezt)
    estimation[i] <- sum(particleWeights * initParticles)
  }

  data = data.frame(zt,xt,estimation,particles = initModel(particles),Particles25,Particles75,initPartic

  ggplot(data,aes(x=c(1:T),y=value,color=variable,xlab="timestep"))+
    geom_line(aes(y=data$zt,col='True Zt'))+
    geom_line(aes(y=data$xt,col='True Xt'))+
    geom_line(aes(y=data$estimation,col='Estimation'))+
    geom_point(aes(x=rep(1,T),y=data$particles,col='particle'))+
    geom_point(aes(x=rep(25,T),y=data$Particles25,col='particle at 25'))+
    geom_point(aes(x=rep(75,T),y=data$Particles75,col='particle at 75'))+
    geom_point(aes(x=rep(T,T),y=data$initParticles,col='particle'))+
    ggtitle(paste('At SD = ',emission_sd))
}

emission_sd = 1
ssm(emission_sd ,transition_sd )

emission_sd = 5
ssm(emission_sd ,transition_sd )

emission_sd = 50
ssm(emission_sd ,transition_sd )
# Question 3: GP
### Question 3(a)
# Change to your path
library(kernlab)
library(mvtnorm)

# From KernelCode.R:
# Squared exponential, k
k <- function(sigmaf = 1, ell = 1)
{
  rval <- function(x, y = NULL)
  {
    r = sqrt(crossprod(x-y))
```

```
    return(sigmaf^2*exp(-r^2/(2*ell^2)))
  }
  class(rval) <- "kernel"
  return(rval)
}


# Simulating from the prior for ell = 0.2
kernel02 <- k(sigmaf = 1, ell = 0.2) # This constructs the covariance function
xGrid = seq(-1,1,by=0.1)
K = kernelMatrix(kernel = kernel02, xGrid, xGrid)

colors = list("black","red","blue","green","purple")
f = rmvnorm(n = 1, mean = rep(0,length(xGrid)), sigma = K)
plot(xGrid,f, type = "l", ylim = c(-3,3), col = colors[[1]])
for (i in 1:4){
  f = rmvnorm(n = 1, mean = rep(0,length(xGrid)), sigma = K)
  lines(xGrid,f, col = colors[[i+1]])
}

# Simulating from the prior for ell = 1
kernel1 <- k(sigmaf = 1, ell = 1) # This constructs the covariance function
xGrid = seq(-1,1,by=0.1)
K = kernelMatrix(kernel = kernel1, xGrid, xGrid)

colors = list("black","red","blue","green","purple")
f = rmvnorm(n = 1, mean = rep(0,length(xGrid)), sigma = K)
plot(xGrid,f, type = "l", ylim = c(-3,3), col = colors[[1]])
for (i in 1:4){
  f = rmvnorm(n = 1, mean = rep(0,length(xGrid)), sigma = K)
  lines(xGrid,f, col = colors[[i+1]])
}


# Computing the correlation functions

# ell = 0.2
kernel02(0,0.1) # Note: here correlation=covariance since sigmaf = 1
kernel02(0,0.5)

# ell = 1
kernel1(0,0.1) # Note: here correlation=covariance since sigmaf = 1
kernel1(0,0.5)

# The correlation between the function at x = 0 and x=0.1 is much higher than
# between x = 0 and x=0.5, since the latter points are more distant.
# Thus, the correlation decays with distance in x-space. This decay is much more
# rapid when ell = 0.2 than when ell = 1. This is also visible in the simulations
# where realized functions are much more smooth when ell = 1.


### Question 3(b)
```

```r
load("GPdata.RData")

### ell = 0.2

sigmaNoise = 0.2

# Set up the kernel function
kernelFunc <- k(sigmaf = 1, ell = 0.2)

# Plot the data and the true
plot(x, y, main = "", cex = 0.5)

GPfit <- gausspr(x, y, kernel = kernelFunc, var = sigmaNoise^2)
# Alternative: GPfit <- gausspr(y ~ x, kernel = k, kpar = list(sigmaf = 1, ell = 0.2), var = sigmaNoise
xs = seq(min(x),max(x), length.out = 100)
meanPred <- predict(GPfit, data.frame(x = xs)) # Predicting the training data. To plot the fit.
lines(xs, meanPred, col="blue", lwd = 2)

# Compute the covariance matrix Cov(f)
n <- length(x)
Kss <- kernelMatrix(kernel = kernelFunc, x = xs, y = xs)
Kxx <- kernelMatrix(kernel = kernelFunc, x = x, y = x)
Kxs <- kernelMatrix(kernel = kernelFunc, x = x, y = xs)
Covf = Kss-t(Kxs)%*%solve(Kxx + sigmaNoise^2*diag(n), Kxs)

# Probability intervals for f
lines(xs, meanPred - 1.96*sqrt(diag(Covf)), col = "red")
lines(xs, meanPred + 1.96*sqrt(diag(Covf)), col = "red")

# Prediction intervals for y
lines(xs, meanPred - 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")
lines(xs, meanPred + 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")


legend("topright", inset = 0.02, legend = c("data","post mean","95% intervals for f", "95% predictive i
       col = c("black", "blue", "red", "purple"),
       pch = c('o',NA,NA,NA), lty = c(NA,1,1,1), lwd = 2, cex = 0.55)



### ell = 1

sigmaNoise = 0.2

# Set up the kernel function
kernelFunc <- k(sigmaf = 1, ell = 1)

# Plot the data and the true
plot(x,y, main = "", cex = 0.5)
#lines(xGrid,fVals, type = "l", col = "black", lwd = 3) # true mean

GPfit <- gausspr(x, y, kernel = kernelFunc, var = sigmaNoise^2)
xs = seq(min(x), max(x), length.out = 100)
```

```r
meanPred <- predict(GPfit, data.frame(x = xs)) # Predicting the training data. To plot the fit.
lines(xs, meanPred, col="blue", lwd = 2)

# Compute the covariance matrix Cov(f)
n <- length(x)
Kss <- kernelMatrix(kernel = kernelFunc, x = xs, y = xs)
Kxx <- kernelMatrix(kernel = kernelFunc, x = x, y = x)
Kxs <- kernelMatrix(kernel = kernelFunc, x = x, y = xs)
Covf = Kss-t(Kxs)%*%solve(Kxx + sigmaNoise^2*diag(n), Kxs)

# Probability intervals for f
lines(xs, meanPred - 1.96*sqrt(diag(Covf)), col = "red")
lines(xs, meanPred + 1.96*sqrt(diag(Covf)), col = "red")

# Prediction intervals for y
lines(xs, meanPred - 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")
lines(xs, meanPred + 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")


legend("topright", inset = 0.02, legend = c("data","post mean","95% intervals for f", "95% predictive i
       col = c("black", "blue", "red", "purple"),
       pch = c('o',NA,NA,NA), lty = c(NA,1,1,1), lwd = 2, cex = 0.55)


# Question: Explain the difference between the results from ii) and iii).
# Answer: ii) is about the uncertainty of the function f, which is the MEAN of y
#         iii) is about the uncertainty of individual y values. They are uncertain for
#              two reasons: you don't know f at the test point, and you don't know
#              the error (epsilon) that will hit this individual observation

# Question: Discuss the differences in results from using the two length scales.
#           Answer: shorter length scale gives less smooth f. We are overfitting the data.
#           Answer: longer length scale gives more smoothness.

# Question: Do you think a GP with a squared exponential kernel is a good model for this data? If not,
#           Answer: One would have to experiment with other length scales, or estimate
#           the length scales (see question 3c), but this is not likely to help here.
#           The issue is that the data seems to be have different smoothness for small x
#           than it has for large x (where the function seems much more flat)
#           The solution is probably to have different length scales for different x


### Question 3(c)

# For full points here you should mention EITHER of the following two approaches:
# 1. The marginal likelihood can be used to select optimal hyperparameters,
# and also the noise variance. We can optimize the log marginal likelihood with
# respect to the hyperparameters. In Gaussian Process Regression the marginal likelihood
# is availble in closed form (a formula).
# 2. We can use sampling methods (e.g. MCMC) to sample from the marginal posterior of the hyperparamete
# We need a prior p(theta) for the hyperparameter and then Bayes rule gives the marginal posterior
# p(theta | data) propto p(data | theta)*p(theta)
# where p(data | theta) is the marginal likelihood (f has been integrated out).
```

```r
# If the noise variance is unknown, we can treat like any of the kernel hyperparameters and infer the n
# jointly with the length scale and the prior variance sigma_f

# SSMs

set.seed(12345)
start_time <- Sys.time()

T<-100
mu_0<-50
Sigma_0<-10
R<-1
Q<-5

x<-vector(length=T)
z<-vector(length=T)
err<-vector(length=T)

for(t in 1:T){
  x[t]<-ifelse(t==1,rnorm(1,mu_0,Sigma_0),x[t-1]+1+rnorm(1,0,R))
  z[t]<-x[t]+rnorm(1,0,Q)
}

mu<-mu_0
Sigma<-Sigma_0*Sigma_0 # KF uses covariances
for(t in 2:T){
  pre_mu<-mu+1
  pre_Sigma<-Sigma+R*R # KF uses covariances
  K<-pre_Sigma/(pre_Sigma+Q*Q) # KF uses covariances
  mu<-pre_mu+K*(z[t]-pre_mu)
  Sigma<-(1-K)*pre_Sigma

  err[t]<-abs(x[t]-mu)

  cat("t: ",t,", x_t: ",x[t],", E[x_t]: ",mu," , error: ",err[t],"\n")
  flush.console()
}

mean(err[2:T])
sd(err[2:T])

end_time <- Sys.time()
end_time - start_time
# Question 3 (SSMs)

T<-100
mu_0<-50
Sigma_0<-sqrt(100^2/12)
n_par<-100
tra_sd<-.1
emi_sd<-1

ini_dis<-function(n){ # Sampling the initial model
```

```
  return (runif(n,min=0,max=100))
}

tra_dis<-function(zt){ # Sampling the transition model
  pi=sample(c(0,1,2),1)
  return (rnorm(1,mean=zt+pi,sd=tra_sd))
}

emi_dis<-function(zt){ # Sampling the emision model
  pi=sample(c(-1,0,1),1)
  return (rnorm(1,mean=zt+pi,sd=emi_sd))
}

den_emi_dis<-function(xt,zt){ # Density of the emission model
  return (((dnorm(xt,mean=zt,sd=emi_sd)
            +dnorm(xt,mean=zt-1,sd=emi_sd)
            +dnorm(xt,mean=zt+1,sd=emi_sd))/3))
}

z<-vector(length=T) # Hidden states
x<-vector(length=T) # Observations

for(t in 1:T){ # Sampling the SSM
  z[t]<-ifelse(t==1,ini_dis(1),tra_dis(z[t-1]))
  x[t]<-emi_dis(z[t])
}

plot(z,col="red",main="True location (red) and observed location (black)")
points(x)

# Particle filter starts

bel<-ini_dis(n_par) # Initial particles
err<-vector(length=T) # Error between expected and true locations
w<-vector(length=n_par) # Importance weights

cat("Initial error: ",abs(z[1]-mean(bel)),"\n")

for(t in 1:T){
  for(i in 1:n_par){
    w[i]<-den_emi_dis(x[t],bel[i])
  }
  w<-w/sum(w)

  Ezt<-sum(w * bel) # Expected location
  err[t]<-abs(z[t]-Ezt) # Error between expected and true locations

  com<-sample(1:n_par,n_par,replace=TRUE,prob=w) # Sampling new particles according to the importance w
  bel<-sapply(bel[com],tra_dis) # Project
}

mean(err[(T-10):T])
sd(err[(T-10):T])
```

```r
# Kalman filter starts

R<-tra_sd
Q<-emi_sd

err<-vector(length=T)

mu<-mu_0
Sigma<-Sigma_0*Sigma_0 # KF uses covariances
for(t in 2:T){
  pre_mu<-mu+1
  pre_Sigma<-Sigma+R*R # KF uses covariances
  K<-pre_Sigma/(pre_Sigma+Q*Q) # KF uses covariances
  mu<-pre_mu+K*(x[t]-pre_mu)
  Sigma<-(1-K)*pre_Sigma

  err[t]<-abs(z[t]-mu)
}

mean(err[(T-10):T])
sd(err[(T-10):T])
```

part4

Omkar Bhutra (omkbh878)

9 January 2020

LAB 4 stuff

install.packages("mvtnorm")

library("mvtnorm")

Covariance function

SquaredExpKernel <- function(x1,x2,sigmaF=1,l=3){ n1 <- length(x1) n2 <- length(x2) K <- matrix(NA,n1,n2) for (i in 1:n2){ K[,i] <- sigmaF^2*exp(-0.5*((x1-x2[i])/l)^2) } return(K) }

Mean function

MeanFunc <- function(x){ m <- sin(x) return(m) }

Simulates nSim realizations (function) from a GP with mean m(x) and covariance K(x,x')

over a grid of inputs (x)

SimGP <- function(m = 0,K,x,nSim,...){ n <- length(x) if (is.numeric(m)) meanVector <- rep(0,n) else meanVector <- m(x) covMat <- K(x,x,...) f <- rmvnorm(nSim, mean = meanVector, sigma = covMat) return(f) }

xGrid <- seq(-5,5,length=20)

Plotting one draw

sigmaF <- 1 l <- 1 nSim <- 1 fSim <- SimGP(m=MeanFunc, K=SquaredExpKernel, x=xGrid, nSim, sigmaF, l) plot(xGrid, fSim[1,], type="p", ylim = c(-3,3)) if(nSim>1){ for (i in 2:nSim) { lines(xGrid, fSim[i,], type="p") } } lines(xGrid,MeanFunc(xGrid), col = "red", lwd = 3) lines(xGrid, MeanFunc(xGrid) - 1.96*sqrt(diag(SquaredExpKernel(xGrid,xGrid,sigmaF,l))), col = "blue", lwd = 2) lines(xGrid, MeanFunc(xGrid) + 1.96*sqrt(diag(SquaredExpKernel(xGrid,xGrid,sigmaF,l))), col = "blue", lwd = 2)

Plotting using manipulate package

library(manipulate)

plotGPPrior <- function(sigmaF, l, nSim){ fSim <- SimGP(m=MeanFunc, K=SquaredExpKernel, x=xGrid, nSim, sigmaF, l) plot(xGrid, fSim[1,], type="l", ylim = c(-3,3), ylab="f(x)", xlab="x") if(nSim>1){ for (i in 2:nSim) { lines(xGrid, fSim[i,], type="l") } } lines(xGrid,MeanFunc(xGrid), col = "red", lwd = 3) lines(xGrid, MeanFunc(xGrid) - 1.96*sqrt(diag(SquaredExpKernel(xGrid,xGrid,sigmaF,l))), col = "blue", lwd = 2) lines(xGrid, MeanFunc(xGrid) + 1.96*sqrt(diag(SquaredExpKernel(xGrid,xGrid,sigmaF,l))), col = "blue", lwd = 2) title(paste('length scale =',l,', sigmaf =',sigmaF)) }

manipulate(plotGPPrior(sigmaF, l, nSim = 10), sigmaF = slider(0, 2, step=0.1, initial = 1, label = "SigmaF"), l = slider(0, 2, step=0.1, initial = 1, label = "Length scale, l"))

Lab 4 setup

Assignment 1

a)

```
tullinge  <- read.csv('https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTulli
data <- read.csv('https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud
                 header=FALSE, sep=',')
# The kernel function
exp_kern <- function(x,xi,l, sigmaf ){

  return((sigmaf^2)*exp(-0.5*( (x - xi) / l )^2))
}
# The implementation, can take a custom kernel of any class.
linear_gp <- function(x,y,xStar,hyperParam,sigmaNoise,kernel){
  n <- length(x)
  kernel_f <- kernel
  # K = Covariance matrix calculation
  K <- function(X, XI,...){

    kov <- matrix(0,nrow = length(X), ncol = length (XI))

    for(i in 1:length(XI)){

      kov[,i]<- kernel_f(X,XI[i],...)

    }
    return(kov)
  }
  l <-hyperParam[1]
  sigmaf <- hyperParam[2]
  #K(X,X)
  K_xx   <- K(x,x, l = l, sigmaf = sigmaf) #, kernel = exp_kern
  #K(X*,X*)
  K_xsxs <- K(xStar,xStar, l = l, sigmaf = sigmaf) # kernel = exp_kern,
  #K(X,X*)
  K_xxs  <- K(x,xStar, l = l, sigmaf = sigmaf)  #kernel = exp_kern,
```

```
  # Algorithm in page 19 of the Rasmus/Williams book
  sI <- sigmaNoise^2 * diag(dim(as.matrix(K_xx))[1])
  # L is transposed according to a definition in the R & W book
  L_transposed <- chol(K_xx + sI)
  L <- t(L_transposed)

  alpha <- solve(t(L), solve(L,y))
  f_bar_star <- t(K_xxs) %*% alpha
  v <- solve(L,K_xxs)
  V_fs <- K_xsxs - t(v) %*% v
  log_mlike <- -0.5 %*% t(y) %*% alpha - sum( diag(L) - n/2 * log(2*pi) )
  return(list(fbar = f_bar_star, vf = V_fs, log_post= log_mlike))
}
```

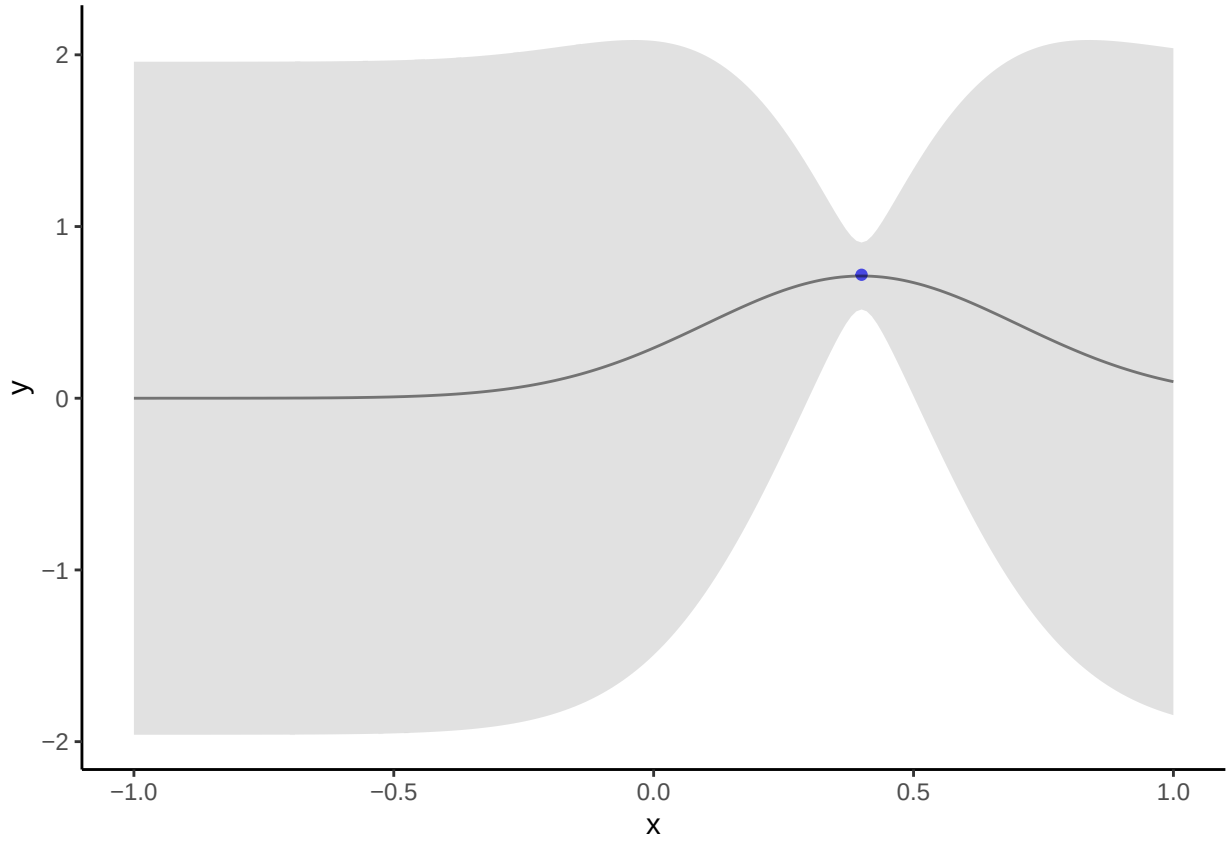
```
# Utility function for the tasks
plot_gp<- function(plot_it,band_it){
  ggplot() +

    geom_point(
      aes(x = x, y = y),
      data = plot_it,
      col = "blue",
      alpha = 0.7) +

    geom_line(
      aes(x = xs, y = fbar),
      data = band_it,
      alpha = 0.50) +

    geom_ribbon(
      aes(ymin = low, ymax = upp, xs),
      data = band_it,
      alpha = 0.15) +

    theme_classic()

}
```

```
# The data given
x <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719 ,-0.664)
# The noise
sn <- 0.1
# The training grid
xs <- seq(-1,1,0.01)
# Hyperparameters l an sigma
hyperParam <- c(0.3, 1)
# Another utility function
repeter <- function(x,y,xs,sn,hyperParam,kernel){
  res <- linear_gp(x,y,xs,hyperParam,sn,kernel)
  # If you want the prediction band just add the noise variance (ie the sigma_n)
  upp <- res$fbar + 1.96*sqrt(diag(res$vf))
  low <- res$fbar - 1.96*sqrt(diag(res$vf))
```

```
  plot_it <- data.frame(x = x, y = y)
  band_it <- data.frame(xGrid = xs, fbar = res$fbar,  upp = upp, low = low)
  plot_gp(plot_it,band_it)
}
```

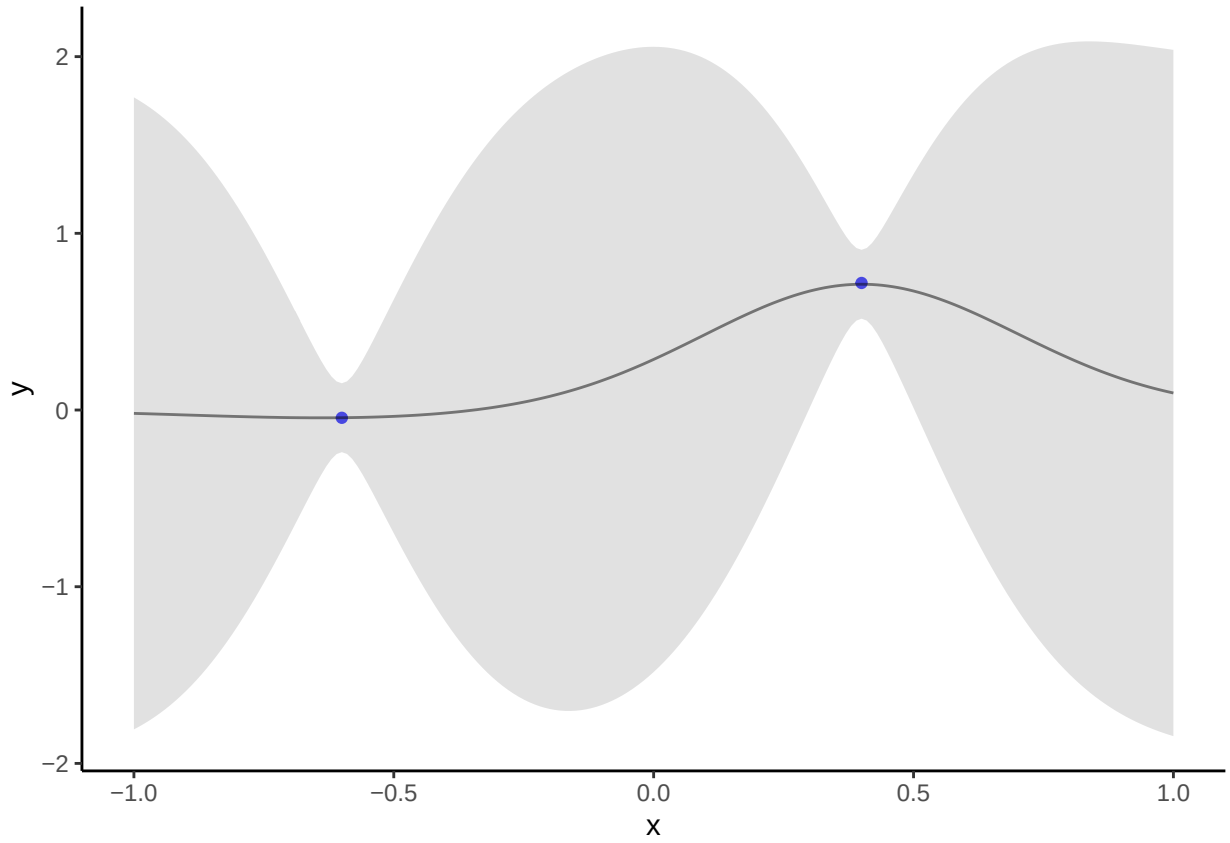
b)

```
repeter(x = x[4], y = y[4],xs,sn,hyperParam, kernel = exp_kern)
```



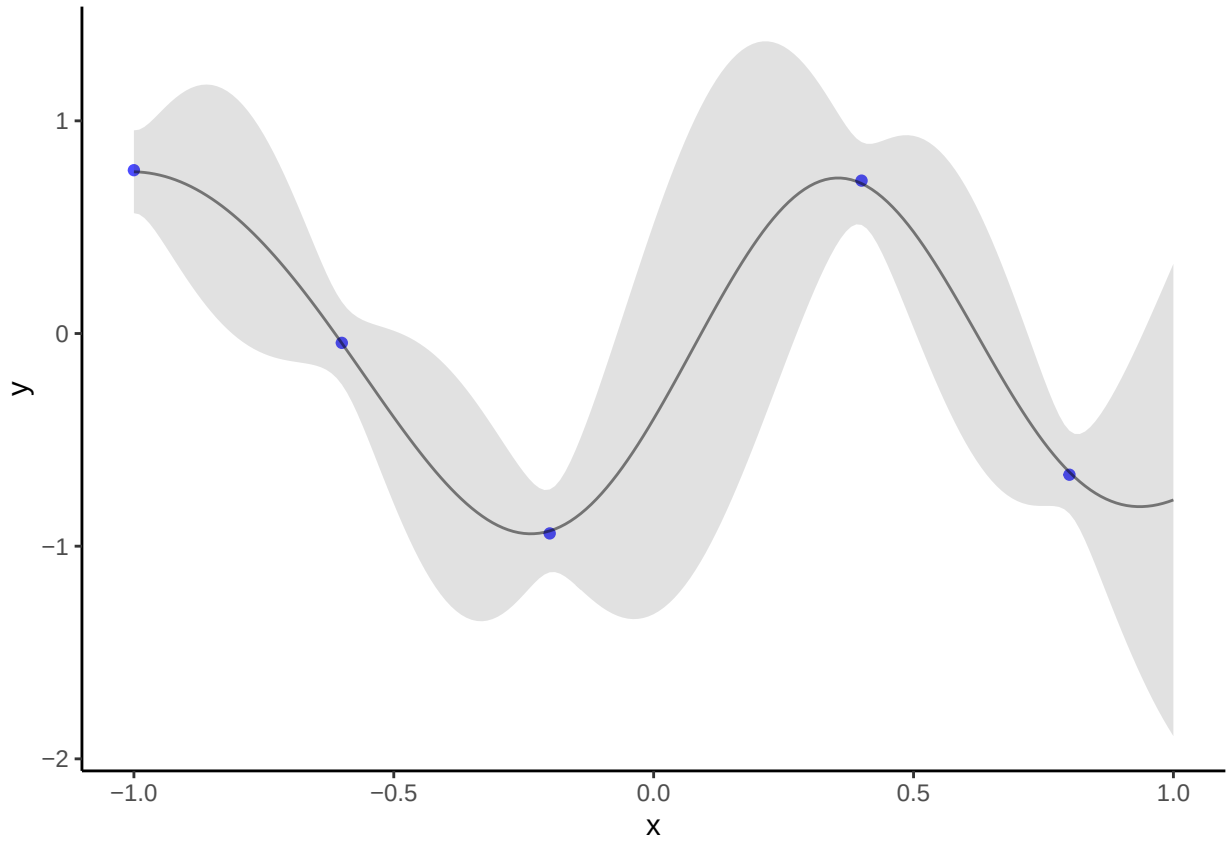
c)

```
repeter(x = x[c(2,4)], y = y[c(2,4)],xs,sn,hyperParam, kernel = exp_kern)
```



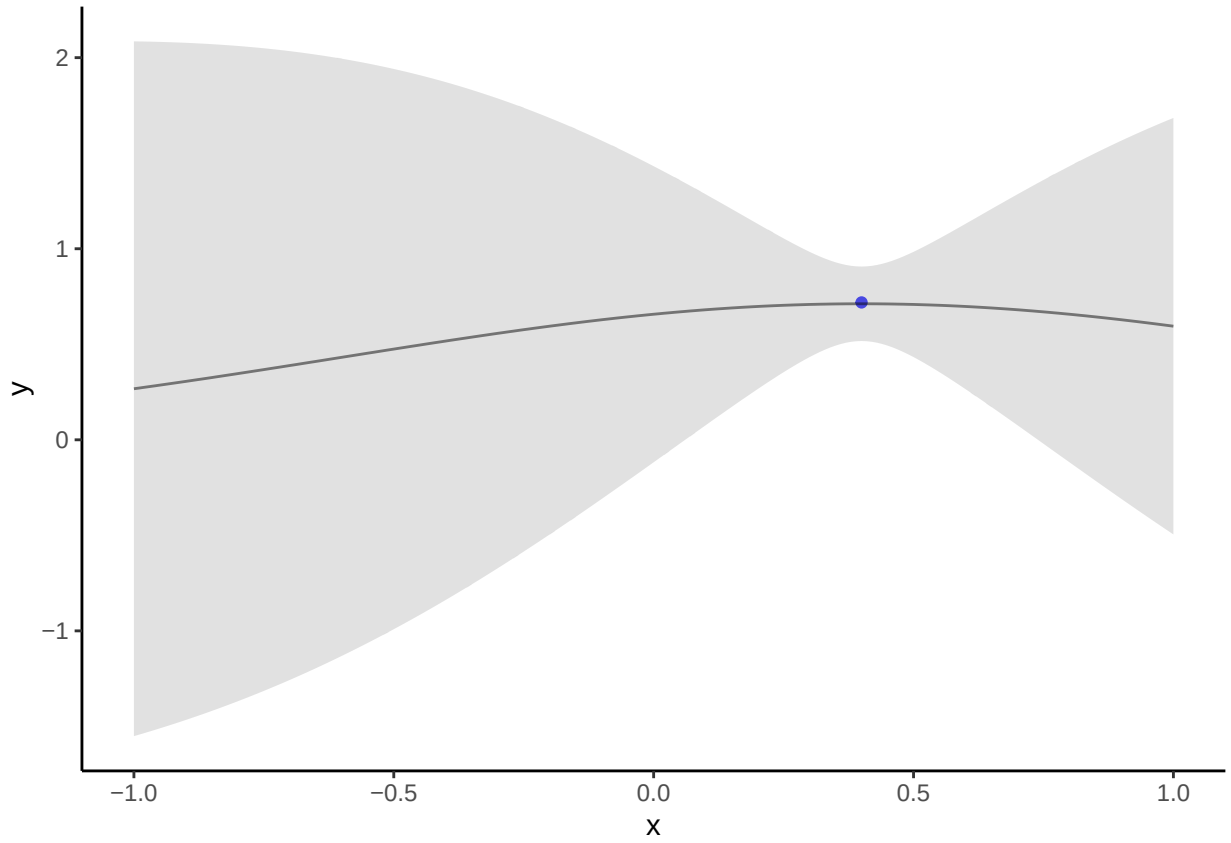
d)

```
repeter(x = x, y = y,xs,sn,hyperParam, kernel = exp_kern)
```

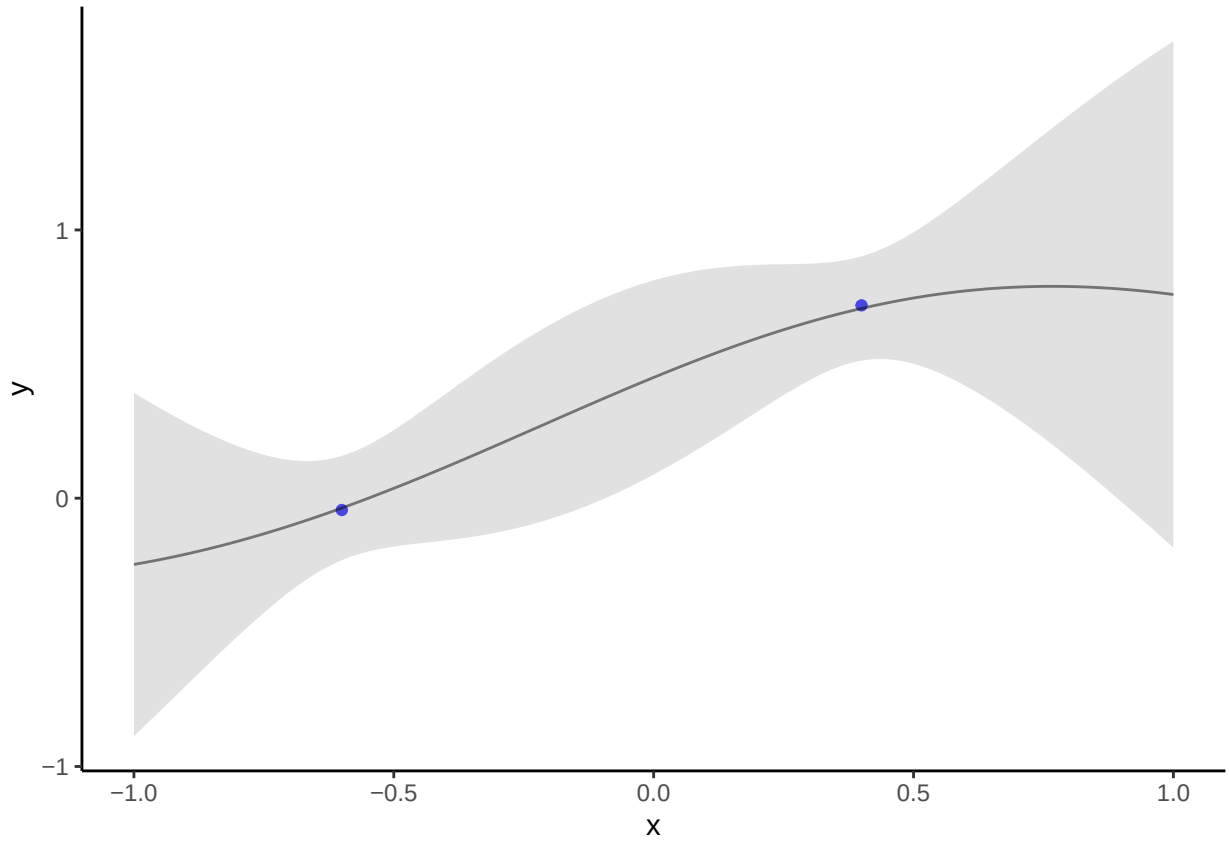


e)

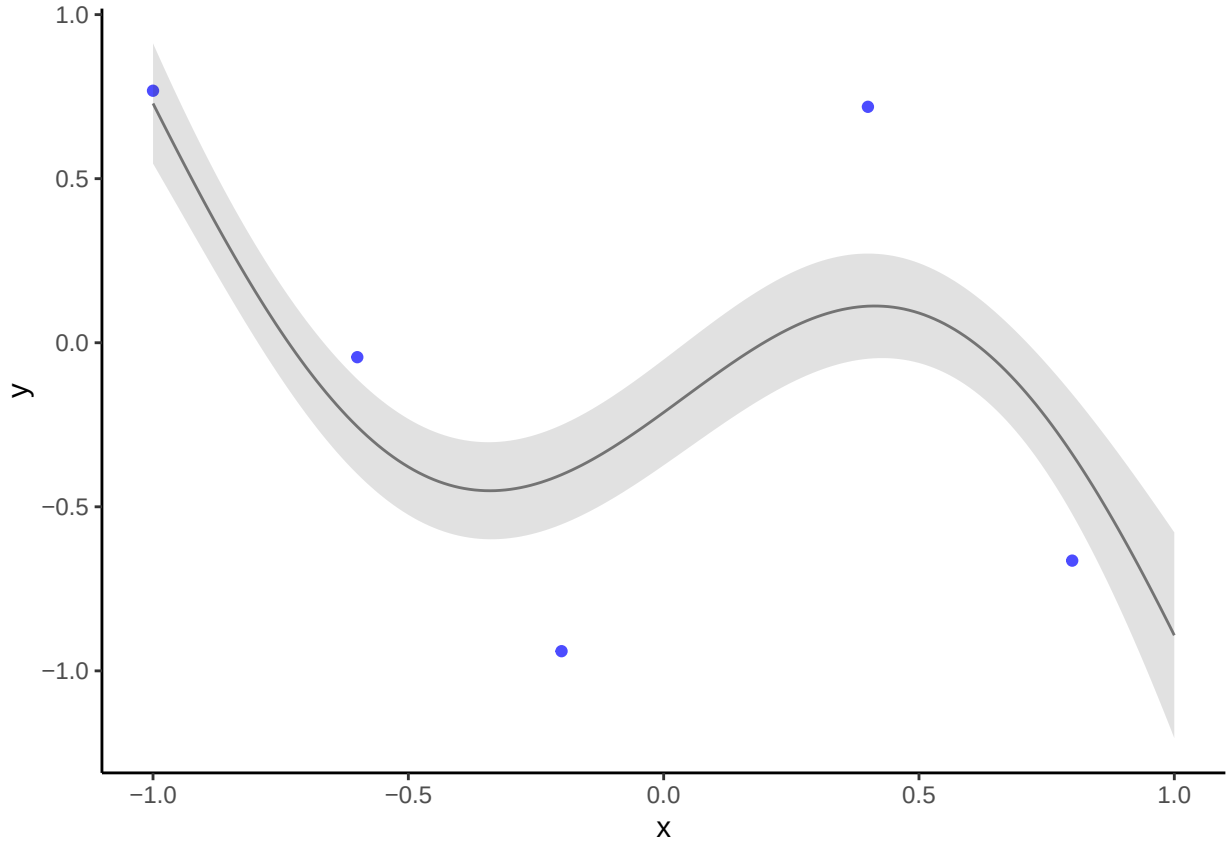
```
x <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719 ,-0.664)
sn <- 0.1
xs <- seq(-1,1,0.01)
hyperParam <- c(1, 1)
repeter(x = x[4], y = y[4],xs,sn,hyperParam, kernel = exp_kern)
```




```
repeter(x = x[c(2,4)], y = y[c(2,4)],xs,sn,hyperParam, kernel = exp_kern)
```




```
repeter(x = x, y = y,xs,sn,hyperParam, kernel = exp_kern)
```



Assignment 2

Data preparations

```
tullinge$time <- 1:nrow(tullinge)
tullinge$day <- rep(1:365,6)
time_sub <- tullinge$time %in% seq(1,2190,5)
tullinge <- tullinge[time_sub,]
```

a)

```
kern_maker <- function(l,sigmaf){

  exp_k <- function(x,y = NULL){

    return((sigmaf^2)*exp(-0.5*( (x - y) / l )^2))
  }

  class(exp_k) <- "kernel"
  return(exp_k)
}
```

```
# gausspr()
# kernelMatrix()
ell <- 1
# SEkernel <- rbfdot(sigma = 1/(2*ell^2)) # Note how I reparametrize the rbfdo (which is the SE kernel)
# SEkernel(1,2)
my_exp <- kern_maker(l = 10, sigmaf =20)
x <- c(1,3,4)
x_star <- c(2,3,4)
#my_exp(x,x_star)
kernelMatrix(my_exp,x,x_star)
```

```
## An object of class "kernelMatrix"
##          [,1]     [,2]    [,3]
## [1,] 398.0050 392.0795 382.399
## [2,] 398.0050 400.0000 398.005
## [3,] 392.0795 398.0050 400.000
```

b)

```
lm_tull <- lm(temp ~ time + I(time^2), data = tullinge)
sigma_2n <- var(resid(lm_tull))
a2b_kern <- kern_maker(l = 0.2, sigmaf = 20 )
gp_tullinge <- gausspr(x = tullinge$time,
                       y = tullinge$temp,
                       kernel = a2b_kern,
                       var = sigma_2n)
```

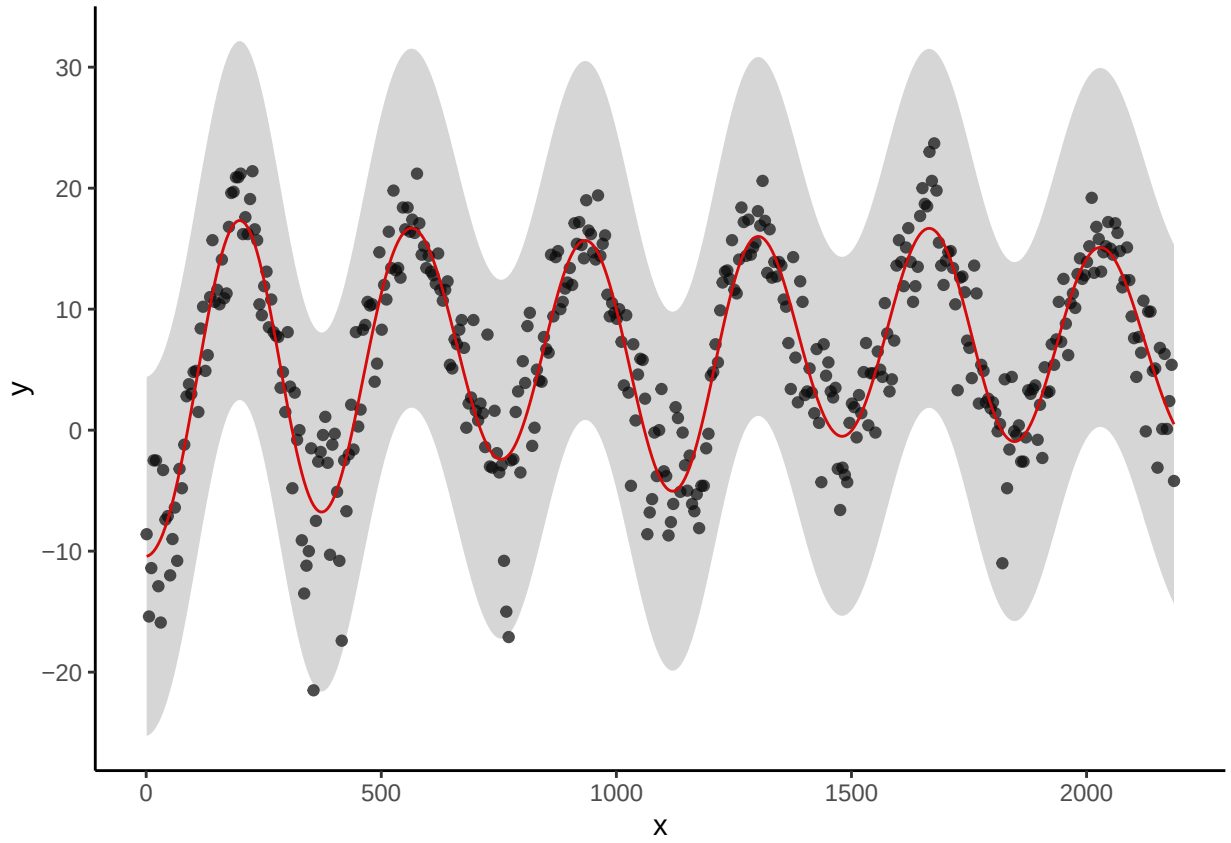
See task c) for the plot.

c)

```
sn_2c <- sqrt(sigma_2n)
xs_2c <- tullinge$time
hyperParam_2c <- c(0.2, 20)
res_2c<- linear_gp(x = tullinge$time,
                   y = tullinge$temp,
                   xStar = xs_2c,
                   sigmaNoise  = sn_2c,
                   hyperParam = hyperParam_2c,
                   kernel = exp_kern)
upp2c <- predict(gp_tullinge) + 1.96*sqrt(diag(res_2c$vf))
low2c <- predict(gp_tullinge) - 1.96*sqrt(diag(res_2c$vf))
plot_it1 <- data.frame(x = tullinge$time, y = tullinge$temp)
band_it1 <- data.frame(xGrid = xs_2c, fbar = res_2c$fbar,  upp = upp2c, low = low2c)
```

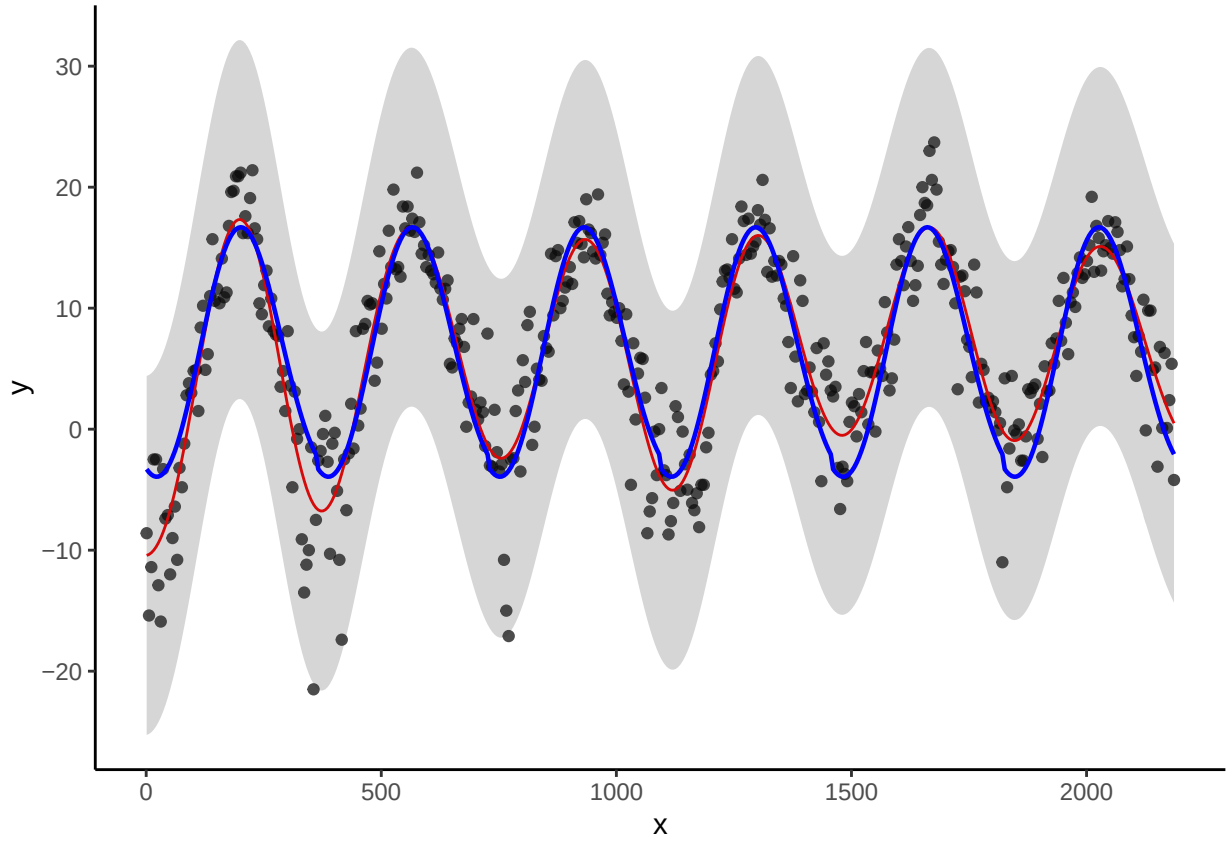
```
C2 <- ggplot() +

  geom_point(
```

```
    aes(x = x, y = y),
    data = plot_it1,
    col = "black",
    alpha = 0.7) +

  geom_line(
    aes(x = xGrid, y = predict(gp_tullinge)),
    data = band_it1,
    alpha = 1,
    col = "red") +

  geom_ribbon(
    aes(ymin = low2c, ymax = upp2c, x = xGrid),
    data = band_it1,
    alpha = 0.2) +

  theme_classic()
#plot(x=band_it1$xGrid, y=band_it1$fbar, type = "l")
C2
```



d)

```
a2d_kern <- kern_maker(l = 1.2, sigmaf = 20 )
gp_tullinge_d <- gausspr(x = tullinge$day,
                         y = tullinge$temp,
                         kernel = a2d_kern,
                         var = sigma_2n)
C23 <- C2 + geom_line(aes(x= tullinge$time,y = predict(gp_tullinge_d)), col = "blue", size = 0.8)
#geom_point(data = tullinge, aes(x= time, y = temp)) +


C23
```




```
# plot(y = tullinge$temp, x = tullinge$day)
# #lines(x = tullinge$time, y = fitted(lm_tull), col = "red")
# lines(x = tullinge$day, y = predict(gp_tullinge_d), col = "red" , lwd = 1)
```

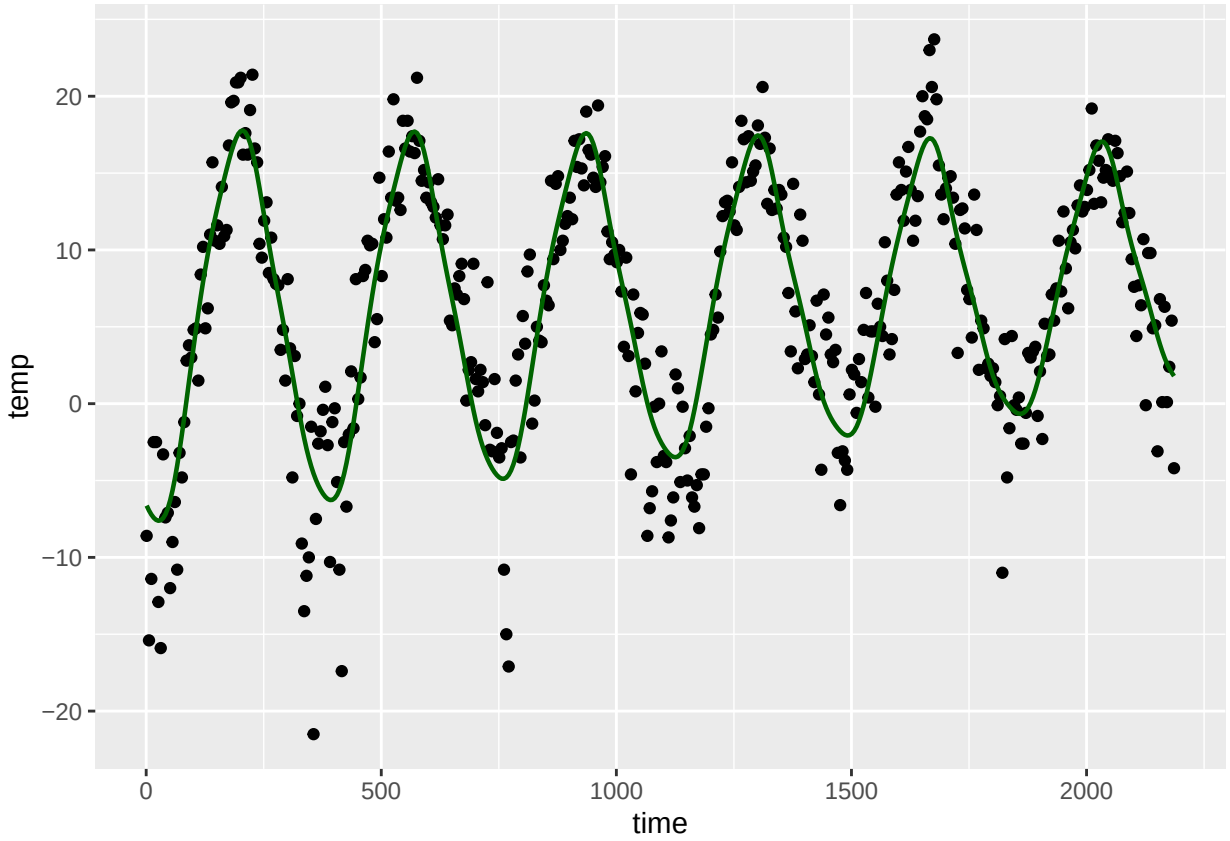
The process model after time has an advantage in that sense that you can capture a trend isolated to a specific time since your modeling using the closest observations in time rather than the day model that assumes that the clostes related temperature point is the one on the same day previous years.

e)

```r
# periodic_kernel <- function(x,xi,sigmaf,d, l_1, l_2){
#
#   part1 <- exp(2 * sin(pi * abs(x - xi) / d)^2 / l_1^2 )
#   part2 <- exp(-0.5 * abs(x - xi)^2 / l_2)
#
#   sigmaf^2 * part1 * part2
#
# }
kern_maker2 <- function(sigmaf,d, l_1, l_2){

  periodic_kernel <- function(x,y = NULL){

    part1 <- exp(-2 * sin(pi * abs(x - y) / d)^2 / l_1^2 )
    part2 <- exp(-0.5 * abs(x - y)^2 / l_2^2)

    sigmaf^2 * part1 * part2

  }

  class(periodic_kernel) <- "kernel"
  return(periodic_kernel)
}
```

```r
sigmaff <- 20
l1 <- 1
l2 <- 10
d_est <- 365 / sd(tullinge$time)

periodic_kernel <- kern_maker2(sigmaf = sigmaff,
                               d = d_est ,
                               l_1 = l1,
                               l_2 = l2)
gp_tullinge_et <- gausspr(x = tullinge$time,
                          y = tullinge$temp,
                          kernel = periodic_kernel,
                          var = sigma_2n)
gp_tullinge_ed <- gausspr(x = tullinge$day,
                          y = tullinge$temp,
                          kernel = periodic_kernel,
                          var = sigma_2n)
```

```r
ggplot(data = tullinge, aes(x= time, y = temp)) +
  geom_point() +
  geom_line(aes(y = predict(gp_tullinge_et)), col = "darkgreen", size = 0.8)
```



This kernel seems to catch both variation in day and over time.

Assignment 3

```
names(data) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")
data[,5] <- as.factor(data[,5])
set.seed(111)
SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
train <- data[SelectTraining,]
test <- data[-SelectTraining,]
```

a)

```
colnames(data)
```

```
## [1] "varWave"     "skewWave"    "kurtWave"    "entropyWave" "fraud"
```

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

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

```
GPfitFraud
```

```
## Gaussian Processes object of class "gausspr"
## Problem type: classification
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  1.2043047635594
##
## Number of training instances learned : 1000
## Train error : 0.068
```

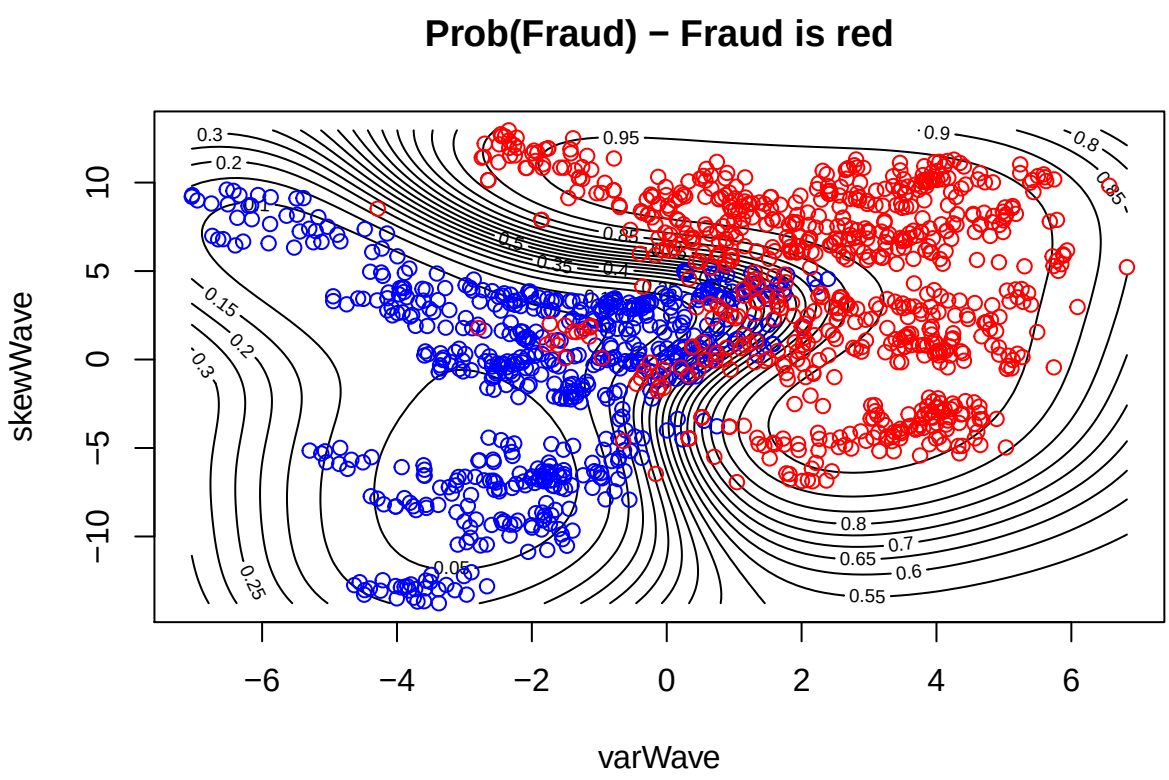
```
# predict on the test set
fit_train<- predict(GPfitFraud,train[,c("varWave","skewWave")])
table(fit_train, train$fraud) # confusion matrix
```

```
##
## fit_train   0   1
##         0 512  24
##         1  44 420
```

```
mean(fit_train == train$fraud)
```

```
## [1] 0.932
```

```
# probPreds <- predict(GPfitIris, iris[,3:4], type="probabilities")
x1 <- seq(min(data[,"varWave"]),max(data[,"varWave"]),length=100)
x2 <- seq(min(data[,"skewWave"]),max(data[,"skewWave"]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))
gridPoints <- data.frame(gridPoints)
names(gridPoints) <- c("varWave","skewWave")
probPreds <- predict(GPfitFraud, gridPoints, type="probabilities")
contour(x1,x2,t(matrix(probPreds[,1],100)), 20,
        xlab = "varWave", ylab = "skewWave",
        main = 'Prob(Fraud) - Fraud is red')
points(data[data[,5]== 1,"varWave"],data[data[,5]== 1,"skewWave"],col="blue")
points(data[data[,5]== 0,"varWave"],data[data[,5]== 0,"skewWave"],col="red")
```



b)

```
# predict on the test set
fit_test<- predict(GPfitFraud,test[,c("varWave","skewWave")])
table(fit_test, test$fraud) # confusion matrix
```

```
##
## fit_test   0   1
##        0 191   9
##        1  15 157
```

```
mean(fit_test == test$fraud)
```

```
## [1] 0.9354839
```

c)

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

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

```
GPfitFraudFull
```

```
## Gaussian Processes object of class "gausspr"
## Problem type: classification
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.399933221120042
##
## Number of training instances learned : 1000
## Train error : 0.004
```

```
# predict on the test set
fit_Full<- predict(GPfitFraudFull,test[,-ncol(test)])
table(fit_Full, test$fraud) # confusion matrix
```

```
##
## fit_Full   0   1
##        0 205   0
##        1   1 166
```

```
mean(fit_Full == test$fraud)
```

```
## [1] 0.9973118
```

```
knitr::opts_chunk$set(echo = TRUE)
library(kernlab)
library(AtmRay)
library(ggplot2)
tullinge  <- read.csv('https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTulli
data <- read.csv('https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud
                 header=FALSE, sep=',')
# The kernel function
exp_kern <- function(x,xi,l, sigmaf ){

  return((sigmaf^2)*exp(-0.5*( (x - xi) / l )^2))
}
# The implementation, can take a custom kernel of any class.
linear_gp <- function(x,y,xStar,hyperParam,sigmaNoise,kernel){
  n <- length(x)
  kernel_f <- kernel
  # K = Covariance matrix calculation
  K <- function(X, XI,...){

    kov <- matrix(0,nrow = length(X), ncol = length (XI))

    for(i in 1:length(XI)){

      kov[,i]<- kernel_f(X,XI[i],...)

    }
    return(kov)
```

```
  }
  l <-hyperParam[1]
  sigmaf <- hyperParam[2]
  #K(X,X)
  K_xx   <- K(x,x, l = l, sigmaf = sigmaf) #, kernel = exp_kern
  #K(X*,X*)
  K_xsxs <- K(xStar,xStar, l = l, sigmaf = sigmaf) # kernel = exp_kern,
  #K(X,X*)
  K_xxs  <- K(x,xStar, l = l, sigmaf = sigmaf)  #kernel = exp_kern,
  # Algorithm in page 19 of the Rasmus/Williams book
  sI <- sigmaNoise^2 * diag(dim(as.matrix(K_xx))[1])
  # L is transposed according to a definition in the R & W book
  L_transposed <- chol(K_xx + sI)
  L <- t(L_transposed)

  alpha <- solve(t(L), solve(L,y))
  f_bar_star <- t(K_xxs) %*% alpha
  v <- solve(L,K_xxs)
  V_fs <- K_xsxs - t(v) %*% v
  log_mlike <- -0.5 %*% t(y) %*% alpha - sum( diag(L) - n/2 * log(2*pi) )
  return(list(fbar = f_bar_star, vf = V_fs, log_post= log_mlike))
}
# Utility function for the tasks
plot_gp<- function(plot_it,band_it){
  ggplot() +

    geom_point(
      aes(x = x, y = y),
      data = plot_it,
      col = "blue",
      alpha = 0.7) +

    geom_line(
      aes(x = xs, y = fbar),
      data = band_it,
      alpha = 0.50) +

    geom_ribbon(
      aes(ymin = low, ymax = upp, xs),
      data = band_it,
      alpha = 0.15) +

    theme_classic()

}
# The data given
x <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719 ,-0.664)
# The noise
sn <- 0.1
# The training grid
xs <- seq(-1,1,0.01)
# Hyperparameters l an sigma
```

```r
hyperParam <- c(0.3, 1)
# Another utility function
repeter <- function(x,y,xs,sn,hyperParam,kernel){
  res <- linear_gp(x,y,xs,hyperParam,sn,kernel)
  # If you want the prediction band just add the noise variance (ie the sigma_n)
  upp <- res$fbar + 1.96*sqrt(diag(res$vf))
  low <- res$fbar - 1.96*sqrt(diag(res$vf))
  plot_it <- data.frame(x = x, y = y)
  band_it <- data.frame(xGrid = xs, fbar = res$fbar,  upp = upp, low = low)
  plot_gp(plot_it,band_it)
}
repeter(x = x[4], y = y[4],xs,sn,hyperParam, kernel = exp_kern)
repeter(x = x[c(2,4)], y = y[c(2,4)],xs,sn,hyperParam, kernel = exp_kern)
repeter(x = x, y = y,xs,sn,hyperParam, kernel = exp_kern)
x <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719 ,-0.664)
sn <- 0.1
xs <- seq(-1,1,0.01)
hyperParam <- c(1, 1)
repeter(x = x[4], y = y[4],xs,sn,hyperParam, kernel = exp_kern)
repeter(x = x[c(2,4)], y = y[c(2,4)],xs,sn,hyperParam, kernel = exp_kern)
repeter(x = x, y = y,xs,sn,hyperParam, kernel = exp_kern)
tullinge$time <- 1:nrow(tullinge)
tullinge$day <- rep(1:365,6)
time_sub <- tullinge$time %in% seq(1,2190,5)
tullinge <- tullinge[time_sub,]
kern_maker <- function(l,sigmaf){

  exp_k <- function(x,y = NULL){

    return((sigmaf^2)*exp(-0.5*( (x - y) / l )^2))
  }

  class(exp_k) <- "kernel"
  return(exp_k)
}
# gausspr()
# kernelMatrix()
ell <- 1
# SEkernel <- rbfdot(sigma = 1/(2*ell^2)) # Note how I reparametrize the rbfdo (which is the SE kernel)
# SEkernel(1,2)
my_exp <- kern_maker(l = 10, sigmaf =20)
x <- c(1,3,4)
x_star <- c(2,3,4)
#my_exp(x,x_star)
kernelMatrix(my_exp,x,x_star)
lm_tull <- lm(temp ~ time + I(time^2), data = tullinge)
sigma_2n <- var(resid(lm_tull))
a2b_kern <- kern_maker(l = 0.2, sigmaf = 20 )
gp_tullinge <- gausspr(x = tullinge$time,
                       y = tullinge$temp,
                       kernel = a2b_kern,
                       var = sigma_2n)
```

```r
sn_2c <- sqrt(sigma_2n)
xs_2c <- tullinge$time
hyperParam_2c <- c(0.2, 20)
res_2c<- linear_gp(x = tullinge$time,
                   y = tullinge$temp,
                   xStar = xs_2c,
                   sigmaNoise  = sn_2c,
                   hyperParam = hyperParam_2c,
                   kernel = exp_kern)
upp2c <- predict(gp_tullinge) + 1.96*sqrt(diag(res_2c$vf))
low2c <- predict(gp_tullinge) - 1.96*sqrt(diag(res_2c$vf))
plot_it1 <- data.frame(x = tullinge$time, y = tullinge$temp)
band_it1 <- data.frame(xGrid = xs_2c, fbar = res_2c$fbar,  upp = upp2c, low = low2c)
C2 <- ggplot() +

  geom_point(
    aes(x = x, y = y),
    data = plot_it1,
    col = "black",
    alpha = 0.7) +

  geom_line(
    aes(x = xGrid, y = predict(gp_tullinge)),
    data = band_it1,
    alpha = 1,
    col = "red") +

  geom_ribbon(
    aes(ymin = low2c, ymax = upp2c, x = xGrid),
    data = band_it1,
    alpha = 0.2) +

  theme_classic()
#plot(x=band_it1$xGrid, y=band_it1$fbar, type = "l")
C2
a2d_kern <- kern_maker(l = 1.2, sigmaf = 20 )
gp_tullinge_d <- gausspr(x = tullinge$day,
                         y = tullinge$temp,
                         kernel = a2d_kern,
                         var = sigma_2n)
C23 <- C2 + geom_line(aes(x= tullinge$time,y = predict(gp_tullinge_d)), col = "blue", size = 0.8)
#geom_point(data = tullinge, aes(x= time, y = temp)) +


C23
# plot(y = tullinge$temp, x = tullinge$day)
# #lines(x = tullinge$time, y = fitted(lm_tull), col = "red")
# lines(x = tullinge$day, y = predict(gp_tullinge_d), col = "red" , lwd = 1)
# periodic_kernel <- function(x,xi,sigmaf,d, l_1, l_2){
#
#   part1 <- exp(2 * sin(pi * abs(x - xi) / d)^2 / l_1^2 )
#   part2 <- exp(-0.5 * abs(x - xi)^2 / l_2)
#
```

```
#   sigmaf^2 * part1 * part2
#
# }
kern_maker2 <- function(sigmaf,d, l_1, l_2){

  periodic_kernel <- function(x,y = NULL){

    part1 <- exp(-2 * sin(pi * abs(x - y) / d)^2 / l_1^2 )
    part2 <- exp(-0.5 * abs(x - y)^2 / l_2^2)

    sigmaf^2 * part1 * part2

  }

  class(periodic_kernel) <- "kernel"
  return(periodic_kernel)
}
sigmaff <- 20
l1 <- 1
l2 <- 10
d_est <- 365 / sd(tullinge$time)

periodic_kernel <- kern_maker2(sigmaf = sigmaff,
                               d = d_est ,
                               l_1 = l1,
                               l_2 = l2)
gp_tullinge_et <- gausspr(x = tullinge$time,
                          y = tullinge$temp,
                          kernel = periodic_kernel,
                          var = sigma_2n)
gp_tullinge_ed <- gausspr(x = tullinge$day,
                          y = tullinge$temp,
                          kernel = periodic_kernel,
                          var = sigma_2n)
ggplot(data = tullinge, aes(x= time, y = temp)) +
  geom_point() +
  geom_line(aes(y = predict(gp_tullinge_et)), col = "darkgreen", size = 0.8)
names(data) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")
data[,5] <- as.factor(data[,5])
set.seed(111)
SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
train <- data[SelectTraining,]
test <- data[-SelectTraining,]
colnames(data)
GPfitFraud <- gausspr(fraud ~  varWave + skewWave, data = train)
GPfitFraud
# predict on the test set
fit_train<- predict(GPfitFraud,train[,c("varWave","skewWave")])
table(fit_train, train$fraud) # confusion matrix
mean(fit_train == train$fraud)
# probPreds <- predict(GPfitIris, iris[,3:4], type="probabilities")
x1 <- seq(min(data[,"varWave"]),max(data[,"varWave"]),length=100)
x2 <- seq(min(data[,"skewWave"]),max(data[,"skewWave"]),length=100)
```

```
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))
gridPoints <- data.frame(gridPoints)
names(gridPoints) <- c("varWave","skewWave")
probPreds <- predict(GPfitFraud, gridPoints, type="probabilities")
contour(x1,x2,t(matrix(probPreds[,1],100)), 20,
        xlab = "varWave", ylab = "skewWave",
        main = 'Prob(Fraud) - Fraud is red')
points(data[data[,5]== 1,"varWave"],data[data[,5]== 1,"skewWave"],col="blue")
points(data[data[,5]== 0,"varWave"],data[data[,5]== 0,"skewWave"],col="red")
# predict on the test set
fit_test<- predict(GPfitFraud,test[,c("varWave","skewWave")])
table(fit_test, test$fraud) # confusion matrix
mean(fit_test == test$fraud)
GPfitFraudFull <- gausspr(fraud ~ ., data = train)
GPfitFraudFull
# predict on the test set
fit_Full<- predict(GPfitFraudFull,test[,-ncol(test)])
table(fit_Full, test$fraud) # confusion matrix
mean(fit_Full == test$fraud)
# Question 4: SSMs

# Kalman filter implementation.

set.seed(12345)
start_time <- Sys.time()

T<-10000
mu_0<-50
Sigma_0<-10
R<-1
Q<-5

x<-vector(length=T)
z<-vector(length=T)
err<-vector(length=T)

for(t in 1:T){
  x[t]<-ifelse(t==1,rnorm(1,mu_0,Sigma_0),x[t-1]+1+rnorm(1,0,R))
  z[t]<-x[t]+rnorm(1,0,Q)
}

mu<-mu_0
Sigma<-Sigma_0*Sigma_0 # KF uses covariances
for(t in 2:T){
  pre_mu<-mu+1
  pre_Sigma<-Sigma+R*R # KF uses covariances
  K<-pre_Sigma/(pre_Sigma+Q*Q) # KF uses covariances
  mu<-pre_mu+K*(z[t]-pre_mu)
  Sigma<-(1-K)*pre_Sigma

  err[t]<-abs(x[t]-mu)
```

```
  cat("t: ",t,", x_t: ",x[t],", E[x_t]: ",mu," , error: ",err[t],"\n")
  flush.console()
}

mean(err[2:T])
sd(err[2:T])

end_time <- Sys.time()
end_time - start_time

# Repetition with the particle filter.

set.seed(12345)
start_time <- Sys.time()

T<-10000
n_par<-100
tra_sd<-1
emi_sd<-5
mu_0<-50
Sigma_0<-10

ini_dis<-function(n){
  return (rnorm(n,mu_0,Sigma_0))
}

tra_dis<-function(zt){
  return (rnorm(1,mean=zt+1,sd=tra_sd))
}

emi_dis<-function(zt){
  return (rnorm(1,mean=zt,sd=emi_sd))
}

den_emi_dis<-function(xt,zt){
  return (dnorm(xt,mean=zt,sd=emi_sd))
}

z<-vector(length=T)
x<-vector(length=T)

for(t in 1:T){
  z[t]<-ifelse(t==1,ini_dis(1),tra_dis(z[t-1]))
  x[t]<-emi_dis(z[t])
}

err<-vector(length=T)

bel<-ini_dis(n_par)
w<-rep(1/n_par,n_par)
for(t in 2:T){
  com<-sample(1:n_par,n_par,replace=TRUE,prob=w)
  bel<-sapply(bel[com],tra_dis)
```

```
  for(i in 1:n_par){
    w[i]<-den_emi_dis(x[t],bel[i])
  }
  w<-w/sum(w)

  Ezt<-sum(w * bel)
  err[t]<-abs(z[t]-Ezt)

  cat("t: ",t,", z_t: ",z[t],", E[z_t]: ",Ezt," , error: ",err[t],"\n")
  flush.console()
}

mean(err[2:T])
sd(err[2:T])

end_time <- Sys.time()
end_time - start_time

# KF works optimally (i.e. it computes the exact belief function in closed-form) since the SSM sampled
# linear-Gaussian. The particle filter on the other hand is approximate. The more particles the closer
# performance to the KF's but at the cost of increasing the running time.

# GPs
#source('KernelCode.R') # Reading the Matern32 kernel from file

Matern32 <- function(sigmaf = 0.5, ell= 0.5)
{
  rval <- function(x, y = NULL) {
      r = sqrt(crossprod(x-y));
      return(sigmaf^2*(1+sqrt(3)*r/ell)*exp(-sqrt(3)*r/ell))
    }
  class(rval) <- "kernel"
  return(rval)
}


# Testing our own defined kernel function.
X <- matrix(rnorm(12), 4, 3) # Simulating some data
Xstar <- matrix(rnorm(15), 5, 3)
MaternFunc = Matern32(sigmaf = 1, ell = 2) # MaternFunc is a kernel FUNCTION
MaternFunc(c(1,1),c(2,2)) # Evaluating the kernel in x=c(1,1), x'=c(2,2)
# Computing the whole covariance matrix K from the kernel.
K <- kernlab::kernelMatrix(kernel = MaternFunc, x = X, y = Xstar) # So this is K(X,Xstar)

sigma2f = 1
ell = 0.5
zGrid <- seq(0.01, 1, by = 0.01)
count = 0
covs = rep(0,length(zGrid))
for (z in zGrid){
  count = count + 1
  covs[count] <- sigma2f*MaternFunc(0,z)
}
```

```r
plot(zGrid, covs, type = "l", xlab = "ell")

# The graph plots Cov(f(0),f(z)), the correlation between two FUNCTION VALUES, as a function of the dis
# two inputs (0 and z)
# As expected the correlation between two points on f decreases as the distance increases.
# The fact that points of f are dependent, as given by the covariance in the plot, makes the curves smo
# have nearby outputs when the correlation is large.

sigma2f = 0.5
ell = 0.5
zGrid <- seq(0.01, 1, by = 0.01)
count = 0
covs = rep(0,length(zGrid))
for (z in zGrid){
  count = count + 1
  covs[count] <- sigma2f*MaternFunc(0,z)
}
plot(zGrid, covs, type = "l", xlab = "ell")

# Changing sigma2f will have not effect on the relative covariance between points on the curve, i.e. wi
# smoothness. But lowering sigma2f makes the whole covariance curve lower. This means that the variance
# distribution over curves which is tighter (lower variance) around the mean function of the GP. This m
# from the GP will be less variable.

#####################################
### GP inference with the ell = 1
#####################################

library(kernlab)
#load("LidarData.txt")
# loading the data
LidarData <- read.delim2("C:/Users/Omkar/Downloads/Adv ML/LidarData.txt",header = TRUE, sep = "")
sigmaNoise = 0.05
x = LidarData$Distance
y = LidarData$LogRatio

# Set up the kernel function
kernelFunc <- Matern32(sigmaf = 1, ell = 1)

# Plot the data and the true
plot(x, y, main = "", cex = 0.5)

GPfit <- gausspr(x, y, kernel = kernelFunc, var = sigmaNoise^2)
xs = seq(min(x),max(x), length.out = 100)
meanPred <- predict(GPfit, xs) # Predicting the training data. To plot the fit.
lines(xs, meanPred, col="blue", lwd = 2)

# Compute the covariance matrix Cov(f)
n <- length(x)
Kss <- kernlab::kernelMatrix(kernel = kernelFunc, x = xs, y = xs)
Kxx <- kernlab::kernelMatrix(kernel = kernelFunc, x = x, y = x)
Kxs <- kernlab::kernelMatrix(kernel = kernelFunc, x = x, y = xs)
Covf = Kss-t(Kxs)%*%solve(Kxx + sigmaNoise^2*diag(n), Kxs)
```

```r
# Probability intervals for f
lines(xs, meanPred - 1.96*sqrt(diag(Covf)), col = "red")
lines(xs, meanPred + 1.96*sqrt(diag(Covf)), col = "red")

# Prediction intervals for y
lines(xs, meanPred - 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")
lines(xs, meanPred + 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")


legend("topright", inset = 0.02, legend = c("data","post mean","95% intervals for f", "95% predictive i
       col = c("black", "blue", "red", "purple"),
       pch = c('o',NA,NA,NA), lty = c(NA,1,1,1), lwd = 2, cex = 0.55)

####################################
### GP inference with the ell = 5
####################################

load("lidar.RData") # loading the data
sigmaNoise = 0.05
x = distance
y = logratio

# Set up the kernel function
kernelFunc <- k(sigmaf = 1, ell = 5)

# Plot the data and the true
plot(x, y, main = "", cex = 0.5)

GPfit <- gausspr(x, y, kernel = kernelFunc, var = sigmaNoise^2)
xs = seq(min(x),max(x), length.out = 100)
meanPred <- predict(GPfit, xs) # Predicting the training data. To plot the fit.
lines(xs, meanPred, col="blue", lwd = 2)

# Compute the covariance matrix Cov(f)
n <- length(x)
Kss <- kernelMatrix(kernel = kernelFunc, x = xs, y = xs)
Kxx <- kernelMatrix(kernel = kernelFunc, x = x, y = x)
Kxs <- kernelMatrix(kernel = kernelFunc, x = x, y = xs)
Covf = Kss-t(Kxs)%*%solve(Kxx + sigmaNoise^2*diag(n), Kxs)

# Probability intervals for f
lines(xs, meanPred - 1.96*sqrt(diag(Covf)), col = "red")
lines(xs, meanPred + 1.96*sqrt(diag(Covf)), col = "red")

# Prediction intervals for y
lines(xs, meanPred - 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")
lines(xs, meanPred + 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")


legend("topright", inset = 0.02, legend = c("data","post mean","95% intervals for f", "95% predictive i
       col = c("black", "blue", "red", "purple"),
       pch = c('o',NA,NA,NA), lty = c(NA,1,1,1), lwd = 2, cex = 0.55)
```

```r
# Discussion
# The larger length scale gives smoother fits. The smaller length scale seems to generate too jagged fi
# PF should outperform KF because the model is not liner-Gaussian, i.e. it is a mixture model.

# Question 4.1 (Hyperparameter selection via log marginal maximization)

tempData <- read.csv('https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullin
temp <- tempData$temp
time = 1:length(temp)

# Extract every 5:th observation
subset <- seq(1, length(temp), by = 5)
temp <- temp[subset]
time = time[subset]

polyFit <- lm(scale(temp) ~  scale(time) + I(scale(time)^2))
sigmaNoiseFit = sd(polyFit$residuals)

SEKernel2 <- function(par=c(20,0.2),x1,x2){
  n1 <- length(x1)
  n2 <- length(x2)
  K <- matrix(NA,n1,n2)
  for (i in 1:n2){
    K[,i] <- (par[1]^2)*exp(-0.5*( (x1-x2[i])/par[2])^2 )
  }
  return(K)
}

LM <- function(par=c(20,0.2),X,y,k,sigmaNoise){
  n <- length(y)
  L <- t(chol(k(par,X,X)+((sigmaNoise^2)*diag(n))))
  a <- solve(t(L),solve(L,y))
  logmar <- -0.5*(t(y)%*%a)-sum(diag(L))-(n/2)*log(2*pi)
  return(logmar)
}

bestLM<-LM(par=c(20,0.2),X=scale(time),y=scale(temp),k=SEKernel2,sigmaNoise=sigmaNoiseFit) # Grid searc
bestLM
besti<-20
bestj<-0.2
for(i in seq(1,50,1)){
  for(j in seq(0.1,10,0.1)){
    aux<-LM(par=c(i,j),X=scale(time),y=scale(temp),k=SEKernel2,sigmaNoise=sigmaNoiseFit)
    if(bestLM<aux){
      bestLM<-aux
      besti<-i
      bestj<-j
    }
  }
}
bestLM
besti
bestj
```

```r
foo<-optim(par = c(1,0.1), fn = LM, X=scale(time),y=scale(temp),k=SEKernel2,sigmaNoise=sigmaNoiseFit, m
           lower = c(.Machine$double.eps, .Machine$double.eps),control=list(fnscale=-1)) # Alternativel

foo$value
foo$par[1]
foo$par[2]

# Question 4.2 (Hyperparameter selection via validation set)

library(kernlab)

data <- read.csv('https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud
names(data) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")
data[,5] <- as.factor(data[,5])
set.seed(111); SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
y <- data[,5]
X <- as.matrix(data[,1:4])
yTrain <- y[SelectTraining]
yTest <- y[-SelectTraining]
XTrain <- X[SelectTraining,]
XTest <- X[-SelectTraining,]
SelectVal <- sample(1:1000, size = 200, replace = FALSE) # 800 samples for training, 200 for validation
yVal <- yTrain[SelectVal]
XVal <- XTrain[SelectVal,]
yTrain <- yTrain[-SelectVal]
XTrain <- XTrain[-SelectVal,]

acVal <- function(par=c(0.1)){ # Accuracy on the validation set
  gausspr(x = XTrain[,selVars], y = yTrain, kernel = "rbfdot", kpar = list(sigma=par[1]))
  predVal <- predict(GPfitFraud,XVal[,selVars])
  table(predVal, yVal)
  accuracyVal <-sum(predVal==yVal)/length(yVal)
  return(accuracyVal)
}

selVars <- c(1,2,3,4)
GPfitFraud <- gausspr(x = XTrain[,selVars], y = yTrain, kernel = "rbfdot", kpar = 'automatic')
GPfitFraud
predVal <- predict(GPfitFraud,XVal[,selVars])
table(predVal, yVal)
accuracyVal <-sum(predVal==yVal)/length(yVal)
accuracyVal

bestVal<-accuracyVal # Grid search
for(j in seq(0.1,10,0.1)){
  aux <- acVal(j)
  if(bestVal<aux){
    bestVal<-aux
    bestj<-j
  }
}
bestVal
bestj
```

```r
gausspr(x = XTrain[,selVars], y = yTrain, kernel = "rbfdot", kpar = list(sigma=bestj)) # Accuracy on th
predTest <- predict(GPfitFraud,XTest[,selVars])
table(predTest, yTest)
sum(predTest==yTest)/length(yTest)

foo<-optim(par = c(0.1), fn = acVal, method="L-BFGS-B",
           lower = c(.Machine$double.eps),control=list(fnscale=-1)) # Alternatively, one can use optim

acVal(foo$par)
foo$par

gausspr(x = XTrain[,selVars], y = yTrain, kernel = "rbfdot", kpar = list(sigma=foo$par)) # Accuracy on
predTest <- predict(GPfitFraud,XTest[,selVars])
table(predTest, yTest)
sum(predTest==yTest)/length(yTest)
```