

Tentadokument

Gustav Sternelöv

18 oktober 2016

Contents

1	Lab 1	2
1.1	Group report	2
1.2	Individual report	6
2	Lab 2	12
2.1	Group report	12
2.2	Individual report	19
3	Lab 3	25
3.1	Group report	25
4	Lab 4	35
4.1	Individual/Group report	35
5	Example code (Lab 3 and Lab 4)	45
5.1	R code for simulating from a GP	45
5.2	Quick demo of the R package kernlab	46
5.3	UC model demo on Nile flow data	49
5.4	DLM package demo on Nile flow data	50
6	Lab instructions	53
6.1	Lab 1	53
6.2	Lab 2	53

1 Lab 1

1.1 Group report

1.1.1 Assignment 1

```
library(bnlearn)
# Assignment 1
data("learning.test")
data("alarm")
data("asia")

set.seed(1897)

init_graph <- random.graph(LETTERS[1:6])
res1 <- hc(learning.test, start = init_graph, restart = 100)
```

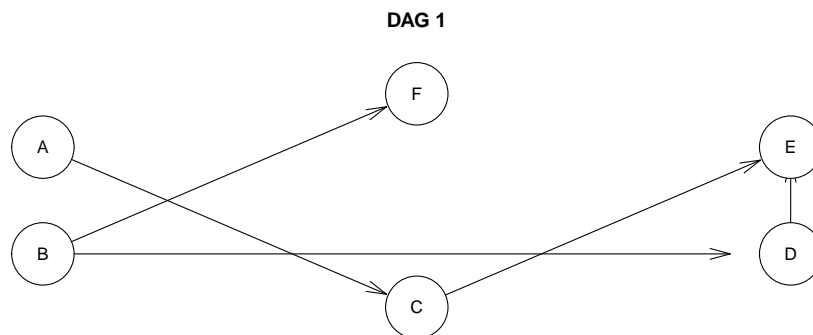
We begin by using the *Hill-climbing algorithm* to setup the structure of the network. We then use this structure in the *Hill-climbing algorithm* and set the number of restarts equal to 100.

```
all.equal(cpdag(init_graph), cpdag(res1))
```

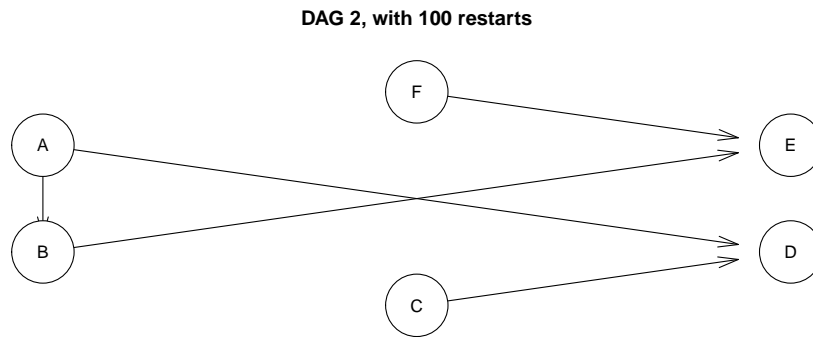
```
## [1] "Different number of directed/undirected arcs"
```

The function *all.equal* can be used to compare the different networks, we here get the result that they are not the same. This can also be seen in the two graphs below.

```
plot(init_graph, main="DAG 1")
```



```
plot(res1, main="DAG 2, with 100 restarts")
```



We notice that we receive two different structures which represent two different models.

```
score(init_graph, learning.test)
```

```
## [1] -28054.13
```

```
score(res1, learning.test)
```

```
## [1] -24006.73
```

The score of the two dags is seen above. We notice that they receive different scores which is logical since the DAGs are different. However, the score is not a sufficient value to use for comparing if two DAGs are equal since it is possible for two different DAGs to obtain the same score. This is a consequence of the independencies, DAGs with the same independencies will always be equivalent if the score is used for comparing them.

1.1.2 Assignment 2

```

# Assignment 2
library(ggplot2)
res_asia <- hc(asia)
res1_asia <- hc(asia, start = res_asia, restart = 100)

```

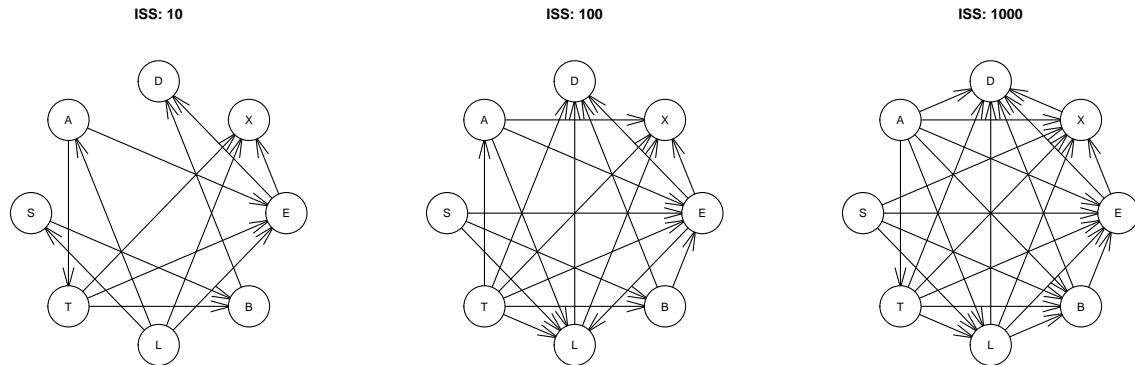
We examine the effect of decreasing the regularization by looking at the complexity of the model given different imaginary sample size. Below are the plots for ISS equal 10, 100 and 1000. It can be seen that as the ISS increases the effect is that more complex models are received.

```

p1 <- hc(asia, start=res_asia, restart = 100, score="bde", iss=10)
p2 <- hc(asia, start=res_asia, restart = 100, score="bde", iss=100)
p3 <- hc(asia, start=res_asia, restart = 100, score="bde", iss=1000)

par(mfrow=c(1,3))
plot(p1, main="ISS: 10")
plot(p2, main="ISS: 100")
plot(p3, main="ISS: 1000")

```



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

When the number of ISS increases we obtain more dense DAGs. However these changes becomes less evident when the ISS becomes large. This is due to that the regularization has decreased. The explanation of this occurrence lies in how the parameter for the penalty has changed. Since the model is less regularized the penalty must have been lowered, and as a result the models become more complex. This is not wanted as a complex model may be heavily overfitted to data and therefore the model normally is regularized more than in the examples here.

1.1.3 Assignment 3

```
library(gRain)
# Assignment 3
#true data
Init2 <- hc(asia,restart = 100)
fit <- bn.fit(x = Init2,data =asia)
#exact value
grainOBJ<-as.grain(fit)
compGrainOBJ<-compile(grainOBJ)
compGrainObj2<-setFinding(compGrainOBJ,nodes = c("A"),states = c("no"))
compGrainObj4<-setFinding(compGrainOBJ,nodes = c("A","B","E"),states = c("no","yes","no"))
compGrainObj6<-setFinding(compGrainOBJ,nodes = c("A","B","E","D","T"),states = c("yes","yes","no","no",
prob1 <- querygrain(compGrainObj2,type = "marginal")
prob2 <- querygrain(compGrainObj4,type = "marginal")
prob3 <- querygrain(compGrainObj6,type = "marginal")
exaxt1 <- c(prob1$B[2],prob2$D[2],prob3$S[2])

#approximate data
#approximate answer will be bad when we have many nodes since it will give us very few observation
set.seed(11827)
compAprox2<-cpdist(fit,nodes =c("A","B"),evidence = (A=="no"))
set.seed(11827)
compAprox4<-cpdist(fit,nodes =c("A","B","E","D"),evidence = (A=="no"&B=="yes"&E=="no"))
set.seed(11827)
compAprox6<-cpdist(fit,nodes =c("A","B","E","D","T","S"),evidence = (A=="yes"&B=="yes"&E=="no"&D=="no"&
```

```

approx <- c(prop.table(table(compAprox2$B))[2], prop.table(table(compAprox4$D))[2], prop.table(table(compAprox4$E))[2])

data.frame("Obs.Nodes" = c("A", "A,B,E", "A,B,E,D,T"),
           "Statement" = c("No", "No,Yes,No", "y,y,n,n,n"),
           "Next included" = c("B", "D", "S"), "Statement"=c(
             "Yes", "Yes", "Yes"), "True prob"=round(exact1,5),
           "Approx prob"=round(approx,5), "Diff"=round(abs(exact1-approx), 5))

```

```

##   Obs.Nodes Statement. Next.included Statement True.prob Approx.prob
## 1      A      No      B      Yes    0.50980    0.52119
## 2    A,B,E No,Yes,No      D      Yes    0.78627    0.79787
## 3 A,B,E,D,T y,y,n,n,n      S      Yes    0.68457    0.80000
##      Diff
## 1 0.01139
## 2 0.01160
## 3 0.11543

```

One can see that the approximated algorithm get less efficient when number of observed nodes increases, this can be seen in the “diff” column where the difference increases. This is an expected outcome, since more observed nodes leads to that there is lesser observations that satisfies the decided statement. So theoretically given zero observed nodes the approximated algorithm and LS should be equally good.

In the table below are the same query for the approximate method performed three times.

```

compAprox2<-cpdist(fit,nodes =c("A","B"),evidence = (A=="no"))
prop.table(table(compAprox2))

```

```

##      B
## A      no      yes
## no  0.4879081 0.5120919
## yes 0.0000000 0.0000000

```

```

compAprox2<-cpdist(fit,nodes =c("A","B"),evidence = (A=="no"))
prop.table(table(compAprox2))

```

```

##      B
## A      no      yes
## no  0.4876867 0.5123133
## yes 0.0000000 0.0000000

```

```

compAprox2<-cpdist(fit,nodes =c("A","B"),evidence = (A=="no"))
prop.table(table(compAprox2))

```

```

##      B
## A      no      yes
## no  0.5003019 0.4996981
## yes 0.0000000 0.0000000

```

The function *cpdist* are based on Monte Carlo particle filters, which uses a random simulation and therefore it is possible to receive different result for each run.

1.1.4 Assignment 4

```
# Assignment 4

fraction <- data.frame(Every1=NA, Every10=NA, Every20=NA)

Every <- c(1,10,20)
burn <- c(300, 1000, 5000)
countCol <- 0

for(i in Every){
  countCol <- countCol + 1
  countRow <- 0
  for(j in burn) {
    countRow <- countRow + 1

    DAGSGraph<-random.graph(LETTERS[1:5],method = "melancon",num=30000,every=i,burn.in=j)
    cpFraction<-list()
    for(k in 1:length(DAGSGraph)){
      cpFraction[[k]]<-cpdag(DAGSGraph[[k]])
    }
    fraction[countRow, countCol] <- length(unique(cpFraction)) / 30000
  }
}
row.names(fraction) <- c("Burn300", "Burn1000", "Burn5000")
```

Depending on the chosen settings for the *random.graph* function a different number of unique EGs are returned. In the table below are the results for different values of every and burn-in summarized. The burn-in parameter does not seem to affect the fraction of EGs returned. Every on the other hand do have some effect since a higher value increase the number of EGs. Especially, for a value higher than one larger fractions are returned.

fraction

##	Every1	Every10	Every20
## Burn300	0.2020000	0.2485333	0.2483667
## Burn1000	0.2062000	0.2489333	0.2472000
## Burn5000	0.2058333	0.2478667	0.2478333

Since around 25 % of the DAGs represent different independence models the structure learning becomes a bit problematic. This is because the same essential graph in many cases then represent several different DAGs and it is not possible to determine which of these DAGs that is the most appropriate one. The effect of this is that it is hard to find the true structure for a network, more precise the true casualities in the actual network.

1.2 Individual report

```
set.seed(311015)
TestData <- data.frame(ground=rep(c("H","A"),500,each=10),
                        surface=rep(c("Artif", "Grass"),1000, each=5),
                        goals=sample(c(0,2,3,4,5), 10000, replace=TRUE,prob=c(2/9,2/6,1/6,1/6,1/6)),
```

```

bookings=sample(c(0,1,2,3), 10000, TRUE, rep(1/4,4)))

TestData$Res = 0
TestData$Red = 0
set.seed(311015)
for (i in 1:nrow(TestData)){
  if(TestData$ground[i] == "H" & TestData$surface[i] == "Artif"){
    TestData$Res[i] = sample(c("1", "X", "2"), 1, prob=c(2/3, 1/6, 1/6))
  }
  if(TestData$ground[i] == "A" & TestData$surface[i] == "Artif"){
    TestData$Res[i] = sample(c("1", "X", "2"), 1, prob=c(1/6, 1/6, 2/3))
  }
  if(TestData$ground[i] == "H" & TestData$surface[i] == "Grass"){
    TestData$Res[i] = sample(c("1", "X", "2"), 1, prob=c(3/6, 2/6, 1/6))
  }
  if(TestData$ground[i] == "A" & TestData$surface[i] == "Grass"){
    TestData$Res[i] = sample(c("1", "X", "2"), 1, prob=c(1/6, 3/6, 2/6))
  }
  if(TestData$goals[i] == 0){
    TestData$Res[i] = "X"
  }else{
    TestData$Res[i] = TestData$Res[i]
  }
  if(TestData$bookings[i] > 2){
    TestData$Red[i] = 1
  }else{
    TestData$Red[i] = 0
  }
}

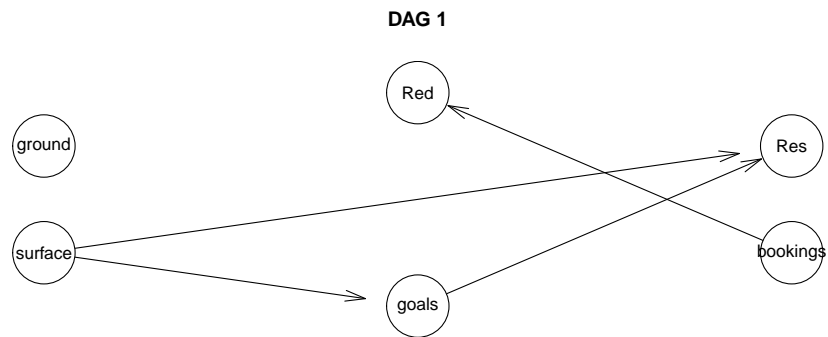
TestData$Res <- as.factor(TestData$Res)
TestData$ground <- seq_along(levels(TestData$ground))[TestData$ground]
TestData$surface <- seq_along(levels(TestData$surface))[TestData$surface]
TestData$Res <- seq_along(levels(TestData$Res))[TestData$Res]
TestData$ground <- as.factor(TestData$ground)
TestData$surface <- as.factor(TestData$surface)
TestData$Res <- as.factor(TestData$Res)
TestData$goals <- as.factor(TestData$goals)
TestData$bookings <- as.factor(TestData$bookings)
TestData$Red <- as.factor(TestData$Red)
# Lab 1

## Assignment 1
# Initialize a random graph with 5 nodes
set.seed(311015)
init_graph <- random.graph(colnames(TestData))
# Run the score based structure learning algorithm with 100 restarts
res1 <- hc(TestData, start = init_graph, restart = 100)
all.equal(cpdag(init_graph), cpdag(res1))

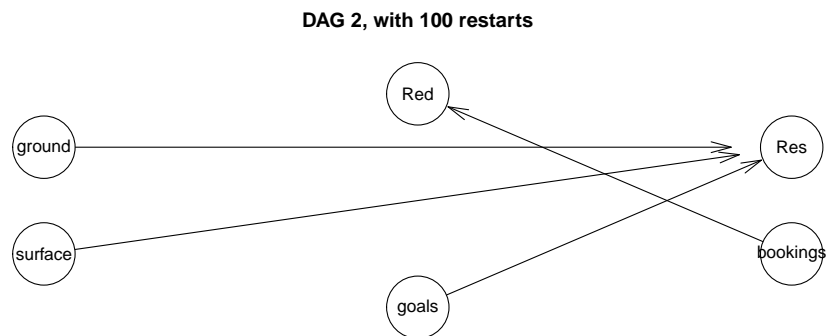
```

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

```
plot(init_graph, main = "DAG 1")
```



```
plot(res1, main = "DAG 2, with 100 restarts")
```



```
score(init_graph, TestData)
```

```
## [1] -49307.88
```

```
score(res1, TestData)
```

```
## [1] -44270.15
```

```
## Assignment 2
```

```
# Run the score based learning algorithm for 3 different imaginary sample sizes  
# initial graph is the graph returned in assignment 1
```

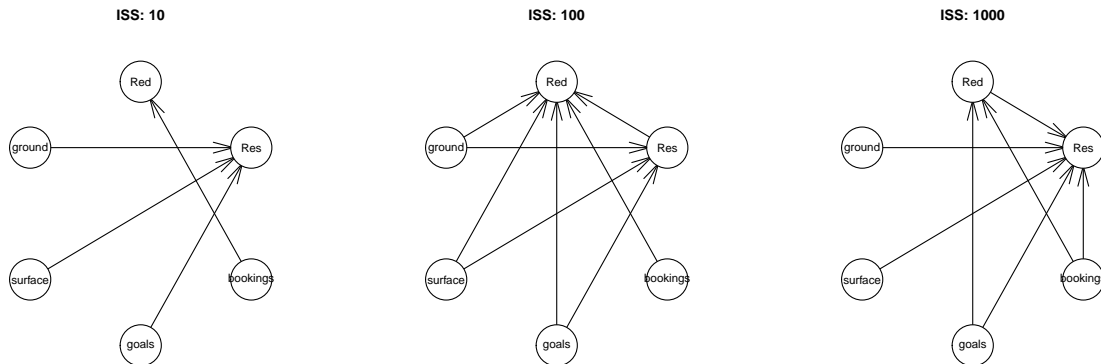
```
p1 <- hc(TestData, start = res1, restart = 100, score = "bde", iss = 10)  
p2 <- hc(TestData, start = res1, restart = 100, score = "bde", iss = 100)  
p3 <- hc(TestData, start = res1, restart = 100, score = "bde", iss = 1000)
```



```

par(mfrow = c(1, 3))
plot(p1, main = "ISS: 10")
plot(p2, main = "ISS: 100")
plot(p3, main = "ISS: 1000")

```



```

par(mfrow = c(1, 1))

# In general, number of connections increases with iss.
# Results in a model that finds non-existing connections.
# Overfitted to this data, this sample.

## Assignment 3
# creates object for approximate method from res1
res1Fit <- bn.fit(res1, TestData)

# approximate method, no evidence
evi_cb <- cpdist(res1Fit, nodes = c("surface"), evidence = TRUE)
prop.table(table(evi_cb))

```

```

## evi_cb
##      1      2
## 0.493 0.507

```

```

# Exact method, no evidence
# object for exact method
res1Fit_grain <- as.grain(res1Fit)

res1Fit_grain <- compile(res1Fit_grain)
res1grain <- setFinding(res1Fit_grain, nodes=c("surface"),
                        states = TRUE)

# Which node that is shown below is determined by [i]
querygrain(res1grain)[2]

```

```

## $surface
## surface

```

```
## 1 2
## 0.5 0.5
```

```
# Clearly see difference between methods when looking at ground or surface
# for which the proportions are known.
```

```
# Approximate method, with evidence
evi_cb <- cpdist(res1Fit, nodes = c("goals"), evidence = (surface == "2"))
prop.table(table(evi_cb))
```

```
## evi_cb
##      0      2      3      4      5
## 0.2103365 0.3233173 0.1506410 0.1488381 0.1668670
```

```
# Exact method, with evidence
ltFit_grain <- setEvidence(res1Fit_grain,nodes=c("surface"), states = c("2") )
querygrain(ltFit_grain)[3]
```

```
## $goals
## goals
##      0      2      3      4      5
## 0.2192 0.3126 0.1536 0.1524 0.1622
```

```
# Still similar results from the respective methods #
```

```
# Approximate method for several different seeds
set.seed(0814)
evi_cb <- cpdist(res1Fit, nodes = c("Red"), evidence = TRUE)
prop.table(table(evi_cb))
```

```
## evi_cb
##      0      1
## 0.7572 0.2428
```

```
set.seed(311015)
evi_cb <- cpdist(res1Fit, nodes = c("Red"), evidence = TRUE)
prop.table(table(evi_cb))
```

```
## evi_cb
##      0      1
## 0.7531 0.2469
```

```
set.seed(1991)
evi_cb <- cpdist(res1Fit, nodes = c("Red"), evidence = TRUE)
prop.table(table(evi_cb))
```

```
## evi_cb
##      0      1
## 0.7492 0.2508
```

```
set.seed(1897)
evi_cb <- cpdist(res1Fit, nodes = c("Red"), evidence = TRUE)
prop.table(table(evi_cb))
```

```
## evi_cb
##      0      1
## 0.751 0.249
```

As expected, different results are obtained for different seeds

Approximate versus Exact when many observed nodes

```
evi_cb <- cpdist(res1Fit, nodes = c("Res"),evidence=c(ground=="2" & surface=="1" &
                                                       goals=="3" & bookings=="2" & Red=="0"))
prop.table(table(evi_cb))
```

```
## evi_cb
##      1      2      3
## 0.95 0.05 0.00
```

Exact method, no evidence

```
ltFit_grain <- setFinding(res1Fit_grain,nodes=c("ground","surface","goals", "bookings", "red"),states =
querygrain(ltFit_grain)[4]
```

```
## $Res
## Res
##      1      2      3
## 0.880829 0.119171 0.000000
```

Results differs more when more observed nodes

*# Problem for approximate method: Fewer and fewer observations that matches the evidence
when the number of observed nodes increases*

2 Lab 2

2.1 Group report

```
library(HMM)
library(ggplot2)
library(scales)
library(gridExtra)
```

2.1.1 1

```
library(ggplot2)
# Assignment 1
# SET UP MY HMM OBJECT
library(HMM)

# Set up my transmission matrix
# P of leaving and P of staying in a given state is 0.5.
# Robot can only move forward.

# The bad GPS gives a prob of 0.2 to be between i-2:i+2.
trans<-matrix(rep(0,100),nrow = 10)
emission<-matrix(rep(0,100),nrow = 10)
for(i in 1:ncol(trans)){
  trans[i,i]<-0.5
  if((i+1)%10==0){
    trans[i,i+1]<-0.5
  }
  trans[i,(i+1)%10]<-0.5
}
for(i in 1:ncol(emission)){
  emission[i,c(i-2,i-1,i,i+1,i+2)%10]<-0.2
  if(sum(emission[i,])!=1){
    emission[i,ncol(emission)]<-0.2
  }
}
# Set up my Hidden markov model
# Symbols=hidden layers?
set.seed(11827)
hmm <- initHMM(States= 1:10, Symbols = 1:10,
               startProbs = rep(0.1,10), transProbs = trans,
               emissionProbs = emission)
```

According to the scenario given in the lab description we build the model by this code:

```
# The start is randomly choosen with 10% for each state.
hmm <- initHMM(States= 1:10, Symbols = 1:10,
               startProbs = rep(0.1,10), transProbs = trans,
               emissionProbs = emission)
```

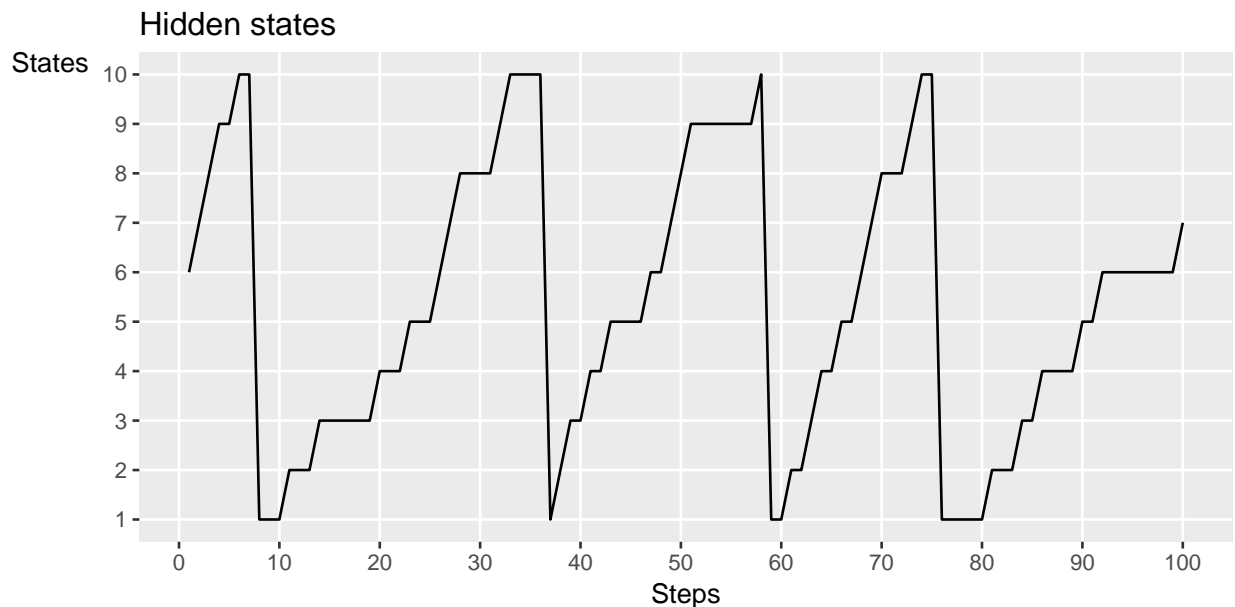
The robot has a chance of staying of 50 % and 50 % of moving one state up. The GPS of the robot is of poor quality therefore the transmission matrix will have 20 % probability for all states ± 2 states from the current state. We set the starting probability equal for all the states.

2.1.2 2

```
##### Assignment 2
# Simulate given hmm and n.
set.seed(311015)
simulateHMM <- simHMM(hmm, 100) # n is sample size

sim_2 <- data.frame(Time=1:100, Path=simulateHMM$states) # DF for ggplot

# Plotting simulated states with time.
ggplot(sim_2) + geom_line(aes(x=Time, Path)) +
  scale_x_continuous(breaks=seq(0,100,10)) +
  scale_y_continuous(breaks=1:10) +
  theme(panel.grid.minor = element_blank(), axis.title.y = element_text(angle=0)) +
  labs(y="States", x="Steps", title="Hidden states")
```



We simulate a HMM for 100 time steps. For this specific simulation we started at state six, and ended at state seven after 100 steps. As expected, the robot does not take any steps backwards, either it stays or it moves to a higher state.

2.1.3 3-4

```
##### Assignment 3-4
# Function to calculate probable path given probabilities matrix(states x time).
countProb <- function(x, states, n) {
  mostProb <- c()
```

```

for(i in 1:n) {
  mostProb[i] <- which.max(x[states,i] )
}
return(mostProb)
}
# START FUNCTION
# Require a hmm object.
# This function returns accuracy of smoothing and filtering method.
# hmm is initHMM object, n =samplem and states=Vector of states.
MYFUNC <- function(hmm, n=100, states) {

  # options(scipen = 999) # Used to keep numbers as numbers.
  #####
  # This is assignment 2
  #####
  # Simulate given hmm and n.
  set.seed(311015)
  simulateHMM <- simHMM(hmm, n) # n is sample size

  #####
  # This is assignment 3
  #####
  # Calculate Alpha and Beta matrix. Use exp to get real prob insted of logged.
  set.seed(173113)
  alpha <- prop.table(exp(forward(hmm, simulateHMM$observation)))

  for(i in 1:n) {
    alpha[, i] <- alpha[,i]/colSums(alpha)[i]
  }

  # Calculate smooth probabilities.
  smooth<- posterior(hmm, simulateHMM$observation) # Smooth

  # Most probable path using viterbi algorithm
  probPathViterbi <- viterbi(hmm, simulateHMM$observation)
  #####
  # This is assignment 4
  #####
  # Set all information in big data.frame
  bigDF <- as.data.frame(cbind(probPathSmoothing = countProb(smooth, states = states, n=n),
                              probPathFiltering = countProb(alpha, states = states, n=n),
                              probPathViterbi = probPathViterbi,
                              Simulated = simulateHMM$states,
                              accSmooth = c(0),
                              accFiltering = c(0),
                              accViterbi= c(0)))

  # Set all equal simulated path to 1. Then I can use mean() to calculate Accuracy.
  bigDF[bigDF$probPathSmoothing==bigDF$Simulated, "accSmooth"] <- 1
  bigDF[bigDF$probPathFiltering==bigDF$Simulated, "accFiltering"] <- 1
  bigDF[bigDF$probPathViterbi==bigDF$Simulated, "accViterbi"] <- 1

  # Return a list consisting of three accuracy, paths and a simulated vector.

```

```

resList <- list(AccuarySmooth= mean(bigDF$accSmooth),
  AccuaryFiltering= mean(bigDF$accFiltering),
  AccuaryViterbi= mean(bigDF$accViterbi),
  probPathSmoothing= bigDF$probPathSmoothing,
  probPathFiltering= bigDF$probPathFiltering,
  probPathViterbi= bigDF$probPathViterbi,
  Simulates= bigDF$Simulated,
  alpha=alpha)

return(resList)
}

```

To solve this assignment we choose to make a function to be able to calculate the filtering algorithm. This is done by using the following formula:

$$Filtering : \frac{\alpha(z^t)}{\sum_{z^t} \alpha(z^t)}$$

For the smoothing probabilities we use the function *posterior()*. Which calculates accordingly to this following formula:

$$Smoothing : \frac{\alpha(z^t)\beta(z^t)}{\sum_{z^t} \alpha(z^t)\beta(z^t)}$$

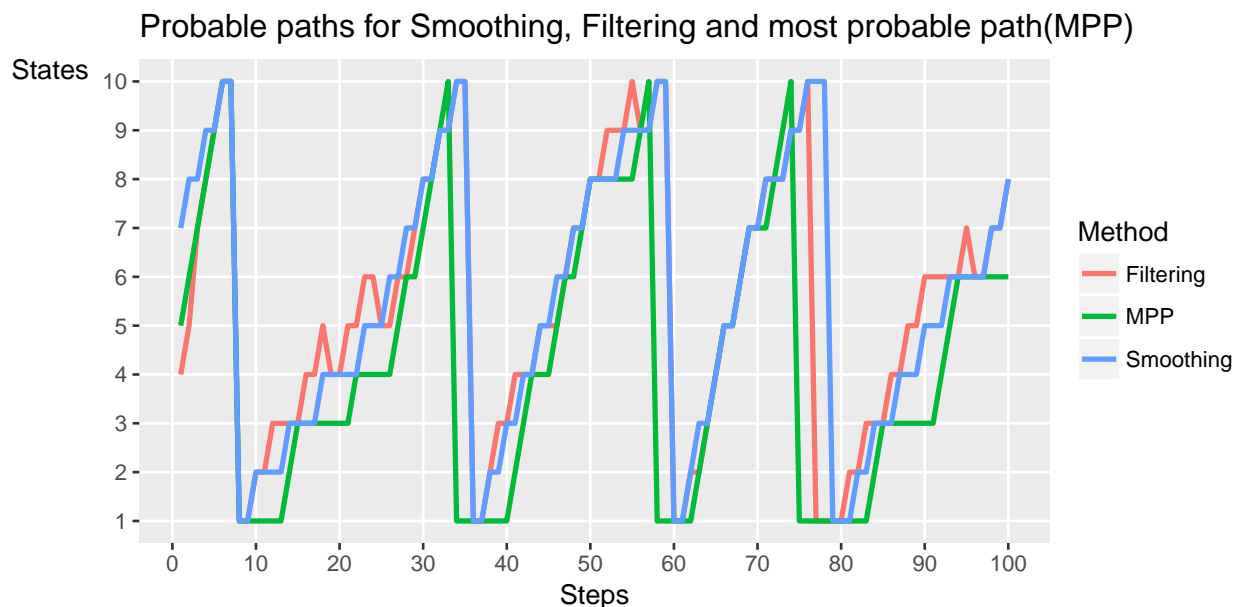
And finally we compute the most probable path using the *Viterbi* algorithm.

The accuracy is shown below.

##	AccuarySmooth	AccuaryFiltering	AccuaryViterbi
##	0.66	0.57	0.44

Here we notice that the accuracy is highest for the smoothed distribution and is the lowest for the most probable path.

We plot the predicted paths and investigate them further.



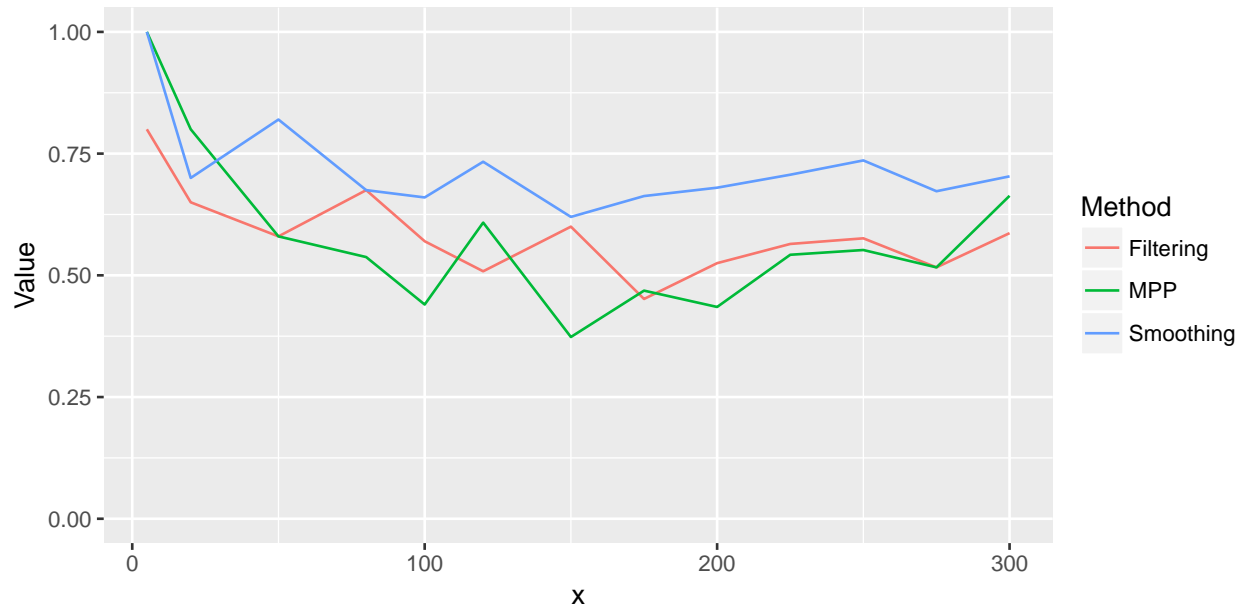
We notice that the different paths start and end in different states. One interesting finding is that the Filtered path at least moves backwards four times, this is not allowed and therefore we suspect that this distribution will not give the most optimal result. It can also be seen that the Viterbi algorithm returns a path that get stuck for long sequences, especially for state one.

2.1.4 5

```
##### Assignment 5
# Investigate how accuracy changes for larger sample sizes
n <- c(5,20,50,80,100,120,150,175,200,225,250,275,300) # These are the sample sizes
# DF for plotting
df <- data.frame(Value=c(NA),
                  Method=rep(c("MPP", "Filtering", "Smoothing"), each=length(n)),
                  x=rep(n, 3))

# These values are needed as a consequence of ggplot plotting
# k, j, l is the
k <- 0
j <- length(n)
l <- length(n) * 2
for(i in n) {
  k <- k+1
  j <- j+1
  l <- l+1
  temp <- MYFUNC(hmm = hmm, n = i, states=1:10)
  df[k, "Value"] <- temp[3] # Smooth acc
  df[j, "Value"] <- temp[2] # Filtering acc
  df[l, "Value"] <- temp[1] # MPP acc
}

ggplot() +
  geom_line(data=df, aes(x=x, y=Value, color=Method)) +
  ylim(c(0,1))
```

The three different accuracies are visualized in the above plot. This is done for different sample sizes. We noticed that the smoothed accuracies are superior to the two others for all sample sizes. At a glance of the plot one can say that both the filtering and MPP obtain similar accuracy.

The explanation of the smoothed superior behavior compared to the filtered accuracy is that the smoothed uses more data. This can be seen in the formulas in assignment 3, where the smoothing algorithm uses both the backward and forward part of the algorithm, compared to the filtered algorithm which only uses the forward part.

The smoothing distribution is obtained by the *Forward-Backwards algorithm* while the most probable path is obtained by the *Viterbi algorithm*. The Forward-backwards algorithm gives marginal probability for each individual state while the Viterbi gives probability for the most likely sequence of states.

2.1.5 6

```
##### Assignment 6
library(entropy)

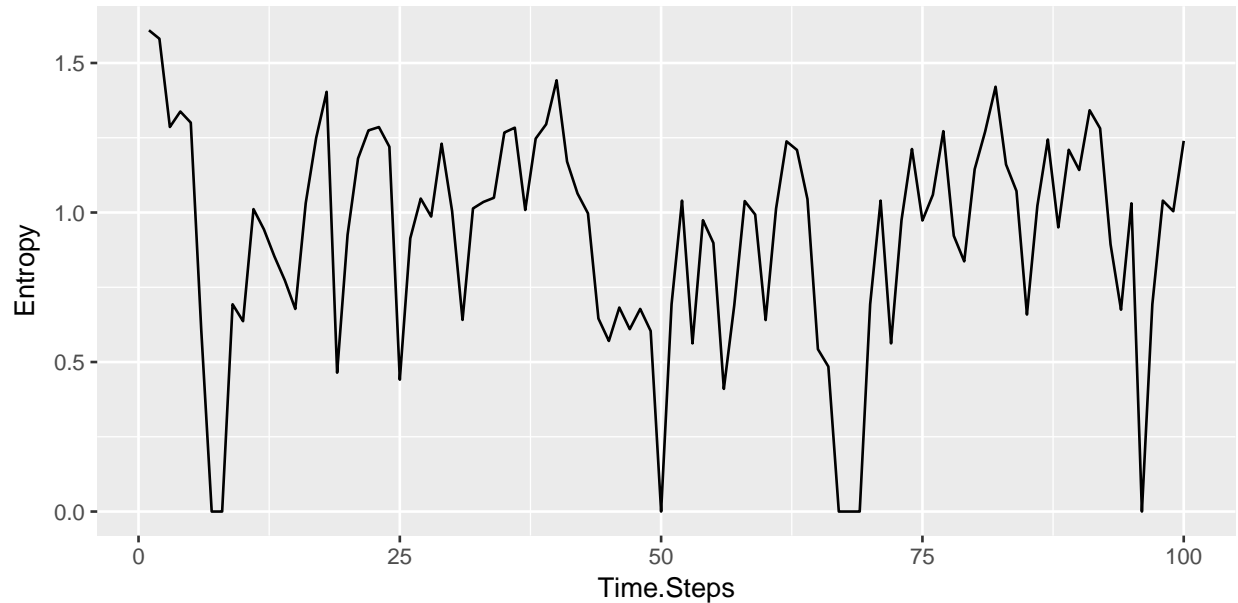
##
## Attaching package: 'entropy'

## The following object is masked from 'package:bnlearn':
##
##      discretize

current.model <- MYFUNC(hmm = hmm, n = 100, states=1:10)
Estimated_Entropy <- c()

for(i in 1:100) {
  Estimated_Entropy[i] <- entropy.empirical(current.model$alpha[,i])
}
gg_entropy <- data.frame(Entropy= Estimated_Entropy,
```

```
Time.Steps=1:100)
ggplot() + geom_line(data=gg_entropy, aes(x=Time.Steps, y=Entropy))
```



Based on the appearance of the graph we can see that increasing time steps does not affect the trend or evidence of the robots position. For low values of the empirical entropy we are more sure about the position of the robot.

2.1.6 7

```
##### Assignment 7
set.seed(2131)
sim <- simHMM(hmm, 100)
alpha <- prop.table(exp(forward(hmm, sim$observation)))
filtering <- alpha/colSums(alpha)
filtering[,100] %*% trans / sum(filtering[,100] %*% trans) # We know state in t100=3
```

```
##      [,1] [,2]      [,3]      [,4]      [,5]      [,6] [,7] [,8] [,9]
## [1,]    0    0 0.01969317 0.1797954 0.4803068 0.3202046    0    0    0
##      [,10]
## [1,]      0
```

The most probable state for $t=101$ for this simulation is that the robot will be in state 5. The vector above is calculated by the following formulas.

$$p(Z^{101}|X^{0:100}) = \sum_{Z^{100}} p(Z^{101}, Z^{100}|X^{0:100})$$

$$p(Z^{101}|Z^{100}, X^{0:100})p(Z^{100}|X^{0:100}) = p(Z^{101}|Z^{100})p(Z^{100}|X^{0:100})$$

Basically this is just the transition matrix times our previous estimated filtering distribution.

2.2 Individual report

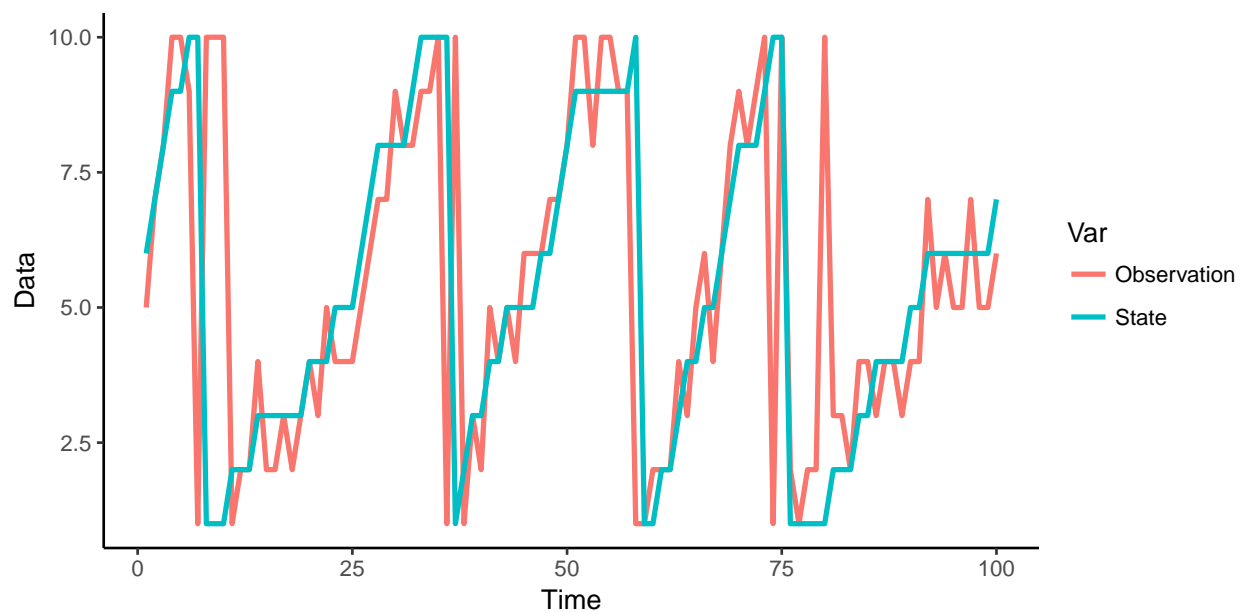
```
# 1. Initialize the model
# Transition matrix is unchanged
# Emission matrix is modified, see below

seq_eP <- rep(1:10,10)
transP <- matrix(data=0, nrow=10, ncol=10)
for(i in 1:10){
  transP[i, c(seq_eP[i], seq_eP[i+1])] <- 0.5
}
# Emission matrix only give probabilities for states +-1
emissionP <- matrix(data=0, nrow=10, ncol=10)
for(i in 1:10){
  j <- i+10
  emissionP[c(seq_eP[j-1],seq_eP[j], seq_eP[j+1]),i] <- 0.2
}

HMM1 <- initHMM(States = c(1:10), Symbols = c(1:10), startProbs = rep(0.1, 10),
               transProbs = transP, emissionProbs = emissionP)

# 100 observations are simulated from the model
set.seed(311015)
Sim1 <- simHMM(HMM1, 100)

# A plot over the simulated states and the observations
Sims1 <- data.frame(Time=rep(1:100, 2), Data=c(Sim1$states, Sim1$observation),
                  Var=rep(c("State", "Observation"), each=100))
ggplot(Sims1, aes(x=Time, y=Data, col=Var)) + geom_line(size=1) + theme_classic()
```



```

# State sequence never moves backwards but observation sometimes does

# The filtered distribution is computed
options(scipen=99)
alpha <- exp(forward(HMM1, Sim1$observation))
alpha <- prop.table(alpha)
filtFunc <- function(Data){
  Data <- Data / sum(Data)
}
filtering <- apply(alpha,2,filtFunc)
probState <- function(Data){
  Data <- which.max(Data)
}
Filter <- data.frame(state=apply(filtering, 2, probState), Time=1:100)

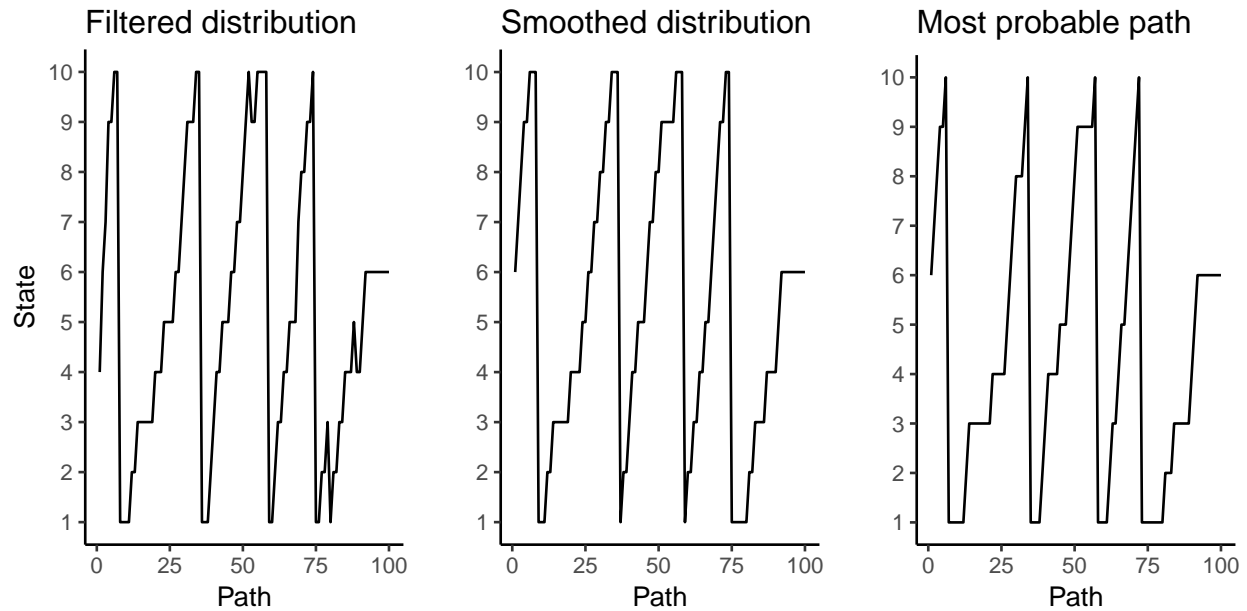
filter_p <- ggplot(Filter, aes(x=Time, y=state)) + geom_path() +
  labs(x="Path", y="State",title="Filtered distribution") +
  scale_y_continuous(breaks=c(1:10)) + theme_classic()

# Smoothed distribution
smoothing <- posterior(HMM1, Sim1[[2]])
smooth <- data.frame(state=apply(smoothing, 2, probState), Time=1:100)
smooth_p <- ggplot(smooth, aes(x=Time, y=state)) + geom_path() +
  labs(x="Path", y="", title= "Smoothed distribution") +
  scale_y_continuous(breaks=c(1:10)) + theme_classic()

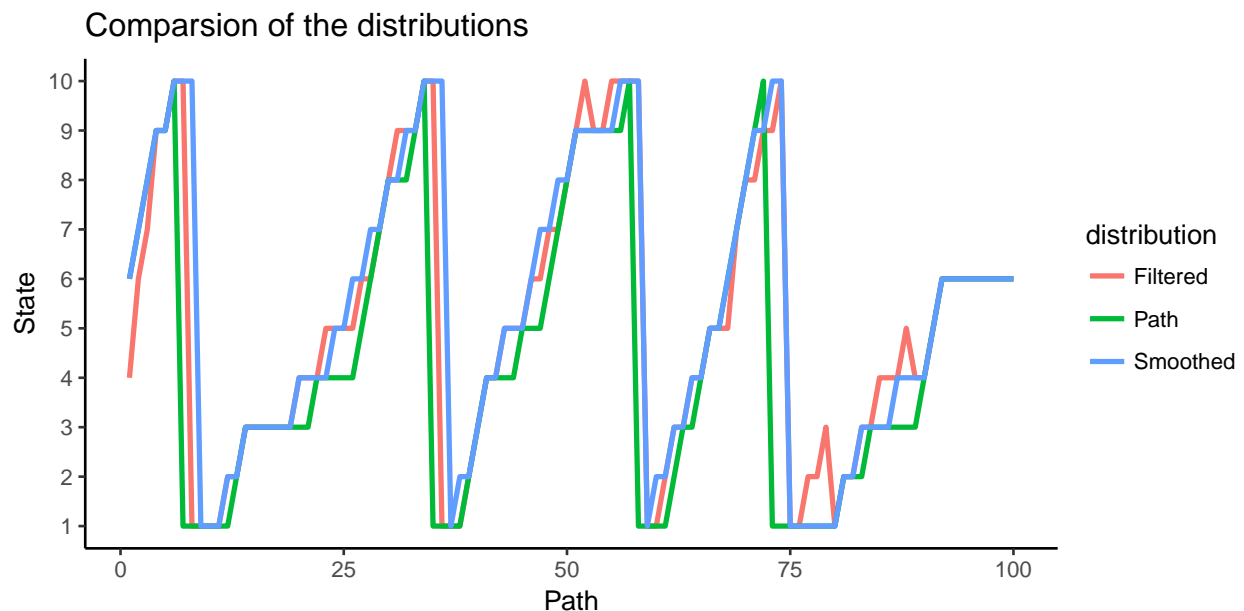
# The most probable path
path <- data.frame(state=as.numeric(viterbi(HMM1, Sim1[[2]])), Time=1:100)
path_p <- ggplot(path, aes(x=Time, y=state)) + geom_path() +
  labs(x="Path", y="", title="Most probable path") +
  scale_y_continuous(breaks=c(1:10)) + theme_classic()

# All distributions next to each other
plots <- list(filter_p, smooth_p, path_p)
plots1 <- arrangeGrob(grobs = plots, nrow=1)
plot(plots1)

```



```
# All distributions in the same plot
Filter$distribution <- "Filtered"
smooth$distribution <- "Smoothed"
path$distribution <- "Path"
allDist <- rbind(Filter, smooth, path)
ggplot(allDist, aes(x=Time, y=state, col=distribution)) + geom_path(size=1) +
  labs(x="Path", y="State", title="Comparison of the distributions") +
  scale_y_continuous(breaks=c(1:10)) + theme_classic()
```



```
# The accuracy for the respective distribution
data.frame(AccuracyFiltered = sum(Sim1[[1]] == Filter$state) / 100,
           AccuracySmoothed = sum(Sim1[[1]] == smooth$state) / 100,
           AccuracyPath = sum(Sim1[[1]] == path$state) / 100)
```

```
## AccuracyFiltered AccuracySmoothed AccuracyPath
## 1 0.69 0.76 0.64
```

```
# The frequency for each state for all distributions
# Not sure if this says something of importance
data.frame(Filtered=prop.table(table(Filter$state))[1:10],
            Smoothed=prop.table(table(smooth$state))[1:10],
            Path=prop.table(table(path$state))[1:10])
```

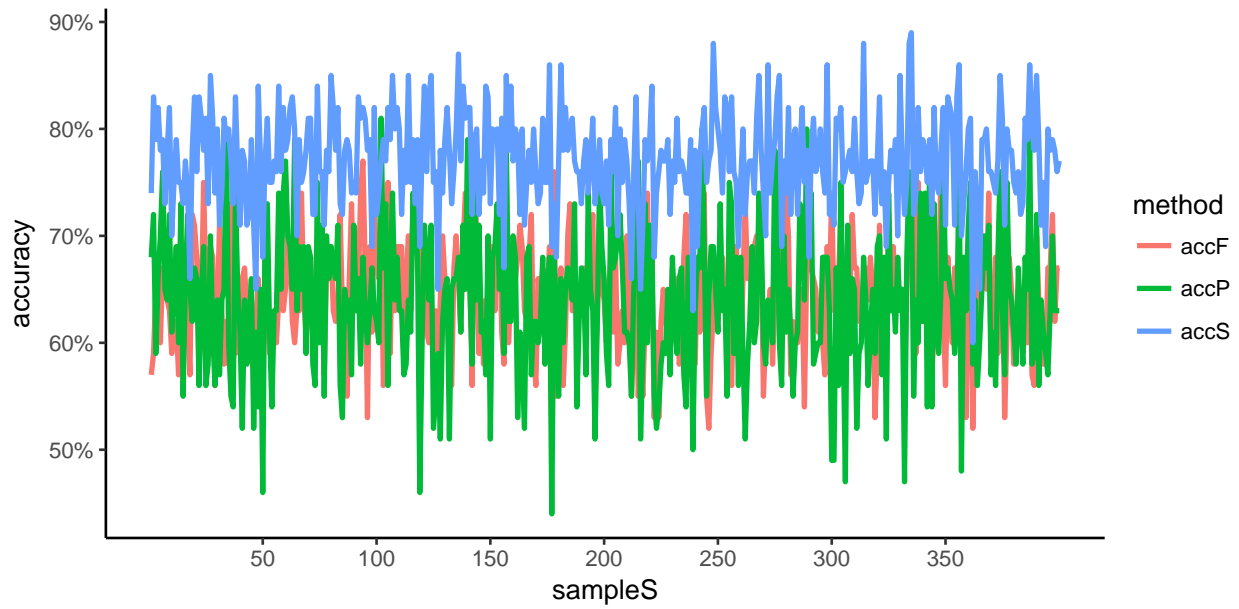
```
## Filtered.Var1 Filtered.Freq Smoothed.Var1 Smoothed.Freq Path.Var1
## 1 1 0.12 1 0.11 1
## 2 2 0.08 2 0.08 2
## 3 3 0.12 3 0.13 3
## 4 4 0.13 4 0.12 4
## 5 5 0.12 5 0.08 5
## 6 6 0.14 6 0.14 6
## 7 7 0.05 7 0.06 7
## 8 8 0.04 8 0.06 8
## 9 9 0.10 9 0.11 9
## 10 10 0.10 10 0.11 10
## Path.Freq
## 1 0.22
## 2 0.06
## 3 0.17
## 4 0.11
## 5 0.07
## 6 0.13
## 7 0.04
## 8 0.06
## 9 0.10
## 10 0.04
```

```
# The accuracy with different samples
# Samples 100 observations 1000 times and calculate the accuracy for each sample
accuracy <- data.frame(sampleS=1:400, accF=0, accS=0, accP=0)
set.seed(0814)
for(t in 1:nrow(accuracy)){
  Sim2 <- simHMM(HMM1, 100)
  #Filtering
  alpha <- exp(forward(HMM1, Sim2$observation))
  alpha <- prop.table(alpha)
  filtering <- apply(alpha,2,filtFunc)
  Filter <- data.frame(state=apply(filtering, 2, probState), Time=1:100)
  # Smoothing
  smoothing <- posterior(HMM1, Sim2[[2]])
  smooth <- data.frame(state=apply(smoothing, 2, probState), Time=1:100)
  # MPP
  path <- data.frame(state=as.numeric(viterbi(HMM1, Sim2[[2]])), Time=1:100)
  accuracy[t,2] <- sum(Sim2[[1]] == Filter$state) / 100
  accuracy[t,3] <- sum(Sim2[[1]] == smooth$state) /100
  accuracy[t,4] <- sum(Sim2[[1]] == path$state) / 100
}
```

```

}
# tidyr to transform data to ggplot format
accuracy2 <- tidyr::gather(accuracy, "method", "accuracy", 2:4)
ggplot(accuracy2, aes(x=sampleS, y=accuracy, col=method)) + geom_line( size=1)+
  theme_classic()+scale_y_continuous(labels=percent) +
  scale_x_continuous(breaks=seq(50,350,50))

```



```

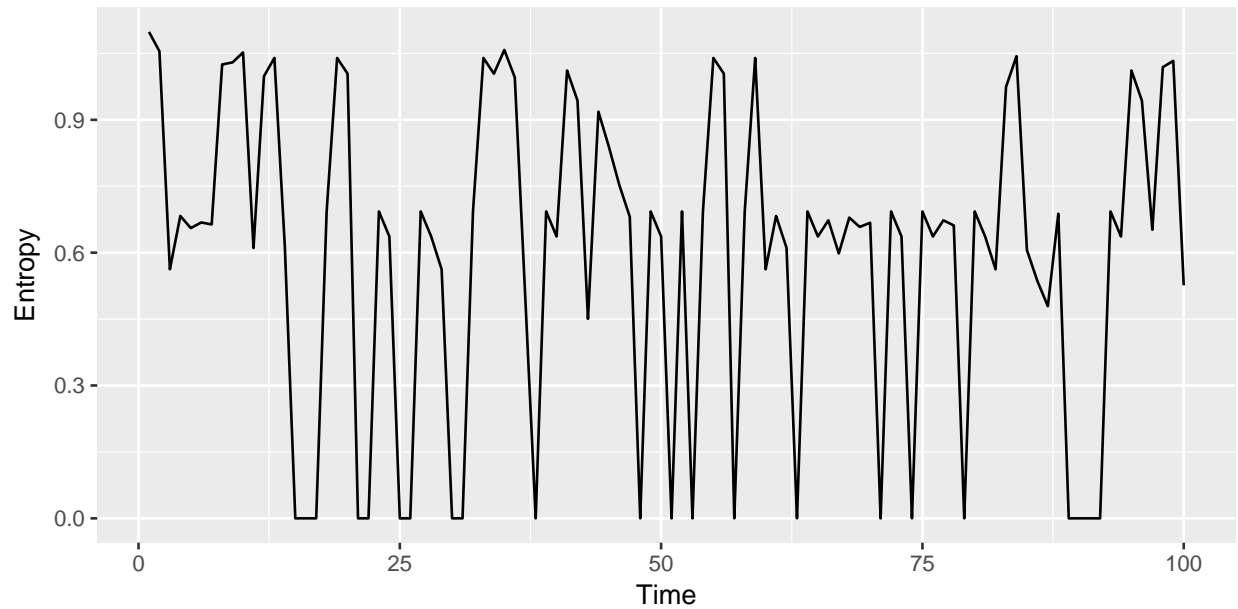
# Smoothed the most accurate. Very even between filtered and MPP

```

```

# The entropy for each time step for the filtered distribution
set.seed(1897)
Sim3 <- simHMM(HMM1, 100)
alpha <- prop.table(exp(forward(HMM1, Sim3$observation)))
filtering <- apply(alpha,2,filtFunc)
Estimated_Entropy <- c()
for (i in 1:100) {
  Estimated_Entropy[i] <- entropy.empirical(filtering[, i])
}
gg_entropy <- data.frame(Entropy = Estimated_Entropy, Time = 1:100)
ggplot(gg_entropy, aes(x = Time, y = Entropy)) + geom_line()

```



```
# Higher value indicates higher uncertainty over where the robot is.
# More sure for lower values and knows exactly where it is if entropy = 0
# Can see that the certainty for the robot's position not increases with time.
# That we knew the position in the previous time step does not necessarily helps us
# in the next time step.
```

```
# For the simulation in the previous assignment are the probabilities for state 101
# calculated. This is done by using the probabilities from the filtering for the
# 100th simulation and the whole transition matrix
```

```
filtering[, 100] %*% transP/sum(filtering[, 100] %*% transP)
```

```
##      [,1] [,2] [,3] [,4]      [,5] [,6]      [,7] [,8] [,9] [,10]
## [1,]  0    0    0    0 0.3902439 0.5 0.1097561    0    0    0
```

```
# Highest probability for Z101 = 6. That is the individually most probable state
# for time step 101.
```


3 Lab 3

3.1 Group report

3.1.1 Assignment 1

3.1.1.1 a)

The implementation of a function that simulates from the posterior distribution $f(x)$ by using the squared exponential kernel is done in two steps. In the first step a function that computes the squared exponential kernel is created. The formula for the squared exponential kernel can be seen below:

$$K(x, x') = \sigma_f^2 \times \exp(-0.5 \times (\frac{x - x'}{\iota})^2)$$

The second step is to build the function *PosteriorGP*. The aim with this function is to calculate the posterior mean and variance of f over a grid of x -values. This is done by implementing algorithm 2.1 from the book *Gaussian Processes for Machine Learning* (Rasmussen and Williams).

The code that has been used to implement the functions can be seen in the appendix *R-code*.

The prior mean of f is assumed to be zero for all x , which gives the following prior distribution for $f(x)$:

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

Then, the posterior gaussian distribution looks as following:

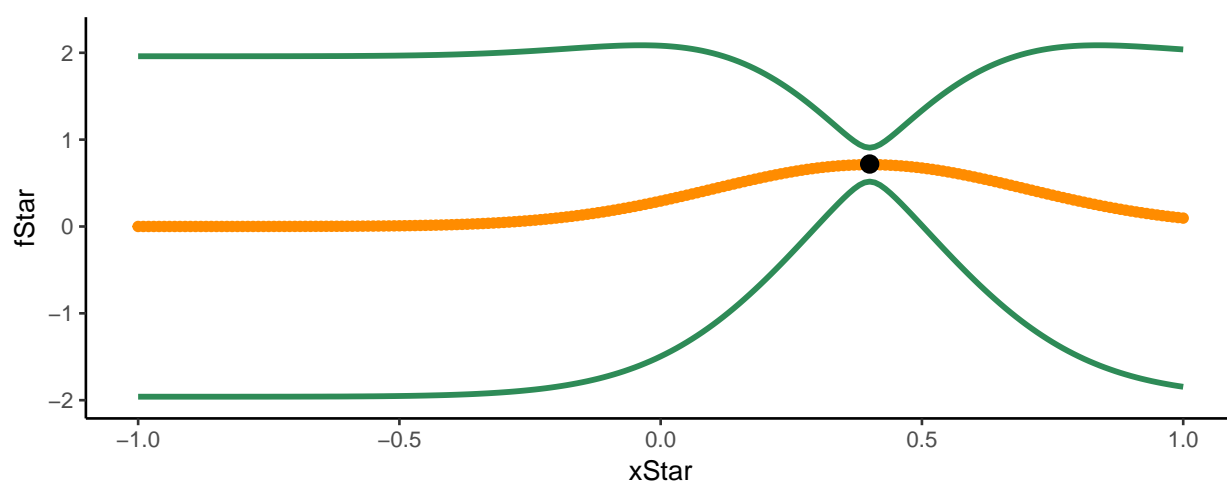
$$f_* \mid x, y, x_* \sim N(\bar{f}_*, \text{cov}(\bar{f}_*))$$

```
library(ggplot2)
# Lab 3
# Assignment 1
SqExpKernel <- function(x1, x2, hyperParam){
  K <- matrix(nrow=length(x1), ncol=length(x2))
  for (i in 1:length(x2)){
    K[, i] <- hyperParam[1]^2 * exp(-0.5 * ((x1-x2[i])/hyperParam[2])^2)
  }
  return(K)
}
PosteriorGP <- function(x, y, xStar, hyperParam, sigmaNoise){
  # Calculates f star bar
  K <- SqExpKernel(x, x, hyperParam)
  L <- t(chol(K + sigmaNoise*diag(dim(K)[1])))
  alpha <- solve(t(L), solve(L, y))
  # Posterior mean
  fStar <- (SqExpKernel(xStar, x, hyperParam)) %*% alpha
  # Posterior variance
  v_f <- solve(L, t(SqExpKernel(xStar, x, hyperParam)))
  cov_fStar <- SqExpKernel(xStar, xStar, hyperParam) - (t(v_f) %*% v_f)
  # Store all values in a list
  val_list <- list(fStar=fStar, xStar=xStar, cov_fStar=cov_fStar, xStar=xStar)
  return(val_list)
}
```

3.1.1.2 b)

Since the noise standard deviation is assumed to be known the parameter σ_n is set to 0.1. The prior hyperparameter σ_f is set to 1 and the second prior hyperparameter ι is set to 0.3. Furthermore the prior is updated with one observation, $(x,y)=(0.4, 0.719)$. A plot over the posterior mean of f over the interval $x \in [-1,1]$ with 95 % probability bands for f can be seen below.

```
assign1B <- PosteriorGP(x=0.4, y=0.719, xStar=seq(-1,1, 0.01), hyperParam=c(1, 0.3),
  sigmaNoise=0.1^2)
Upper1B <- assign1B$fStar + 1.96 * sqrt(diag(assign1B$cov_fStar))
Lower1B <- assign1B$fStar - 1.96 * sqrt(diag(assign1B$cov_fStar))
plotD <- data.frame(fStar=assign1B$fStar, xStar=assign1B$xStar, Lower=Lower1B, Upper=Upper1B)
xy <- data.frame(x=0.4, y=0.719)
ggplot(plotD, aes(y=fStar, x=xStar)) + geom_point(col="darkorange") + ylim(-2,2.2) +
  geom_line(data=plotD, aes(xStar, Lower), col="seagreen", size=1.1) +
  geom_line(data=plotD, aes(xStar, Upper), col="seagreen", size=1.1) + geom_point(data=xy, aes(x,y), size=100)
```



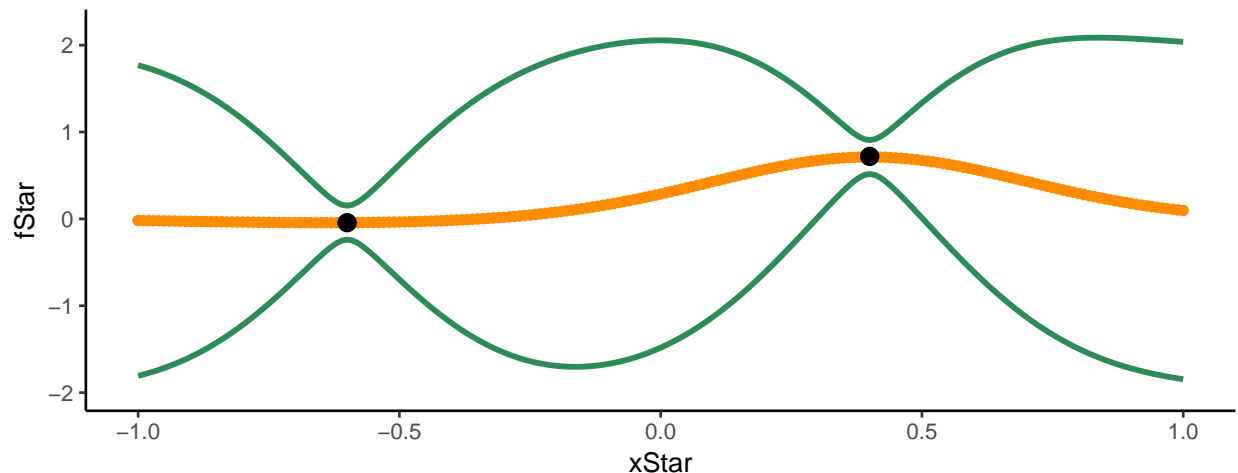
The black dot is the observed value and it can be seen that the probability bands are more narrow around this value.

3.1.1.3 c)

The posterior from *b*) is updated with another observation, $(x,y)=(-0.6, -0.044)$.

```
assign1C <- PosteriorGP(x=c(0.4, -0.6), y=c(0.719, -0.044), xStar=seq(-1,1, 0.01), hyperParam=c(1, 0.3))
Upper1C <- assign1C$fStar + 1.96 * sqrt(diag(assign1C$cov_fStar))
Lower1C <- assign1C$fStar - 1.96 * sqrt(diag(assign1C$cov_fStar))

plotD <- data.frame(fStar=assign1C$fStar, xStar=assign1C$xStar, Lower=Lower1C, Upper=Upper1C)
xy <- data.frame(x=c(0.4, -0.6), y=c(0.719, -0.044))
ggplot(plotD, aes(y=fStar, x=xStar)) + geom_point(col="darkorange") + ylim(-2,2.2) +
  geom_line(data=plotD, aes(xStar, Lower), col="seagreen", size=1.1) +
  geom_line(data=plotD, aes(xStar, Upper), col="seagreen", size=1.1) + geom_point(data=xy, aes(x,y), size=100)
```



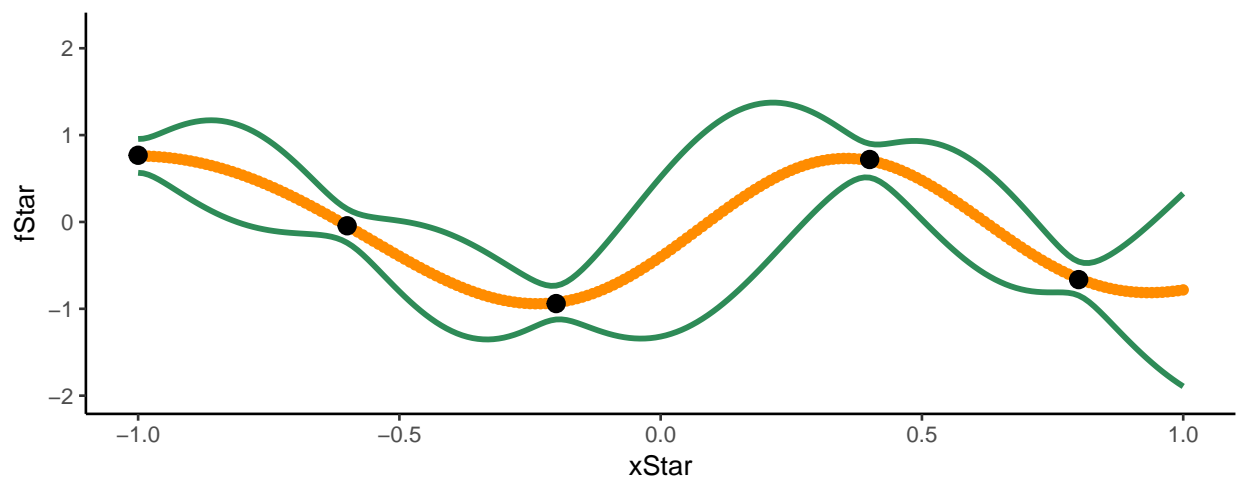
Again it can be seen that the probability bands are more narrow around the observed values, and that they are quite wide for the other values.

3.1.1.4 d)

In *d*) the number of observations rises to five, resulting in the following plot over the posterior mean of f and its 95 % probability intervals.

```
assign1D <- PosteriorGP(x=c(0.8, 0.4, -0.2, -0.6, -1), y=c(-0.664, 0.719, -0.94, -0.044, 0.768), xStar=
Upper1D <- assign1D$fStar + 1.96 * sqrt(diag(assign1D$cov_fStar))
Lower1D <- assign1D$fStar - 1.96 * sqrt(diag(assign1D$cov_fStar))

plotD <- data.frame(fStar=assign1D$fStar, xStar=assign1D$xStar, Lower=Lower1D, Upper=Upper1D)
xy <- data.frame(x=c(0.8, 0.4, -0.2, -0.6, -1), y=c(-0.664, 0.719, -0.94, -0.044, 0.768))
ggplot(plotD, aes(y=fStar, x=xStar)) + geom_point(col="darkorange") + ylim(-2,2.2) +
  geom_line(data=plotD, aes(xStar, Lower), col="seagreen", size=1.1) +
  geom_line(data=plotD, aes(xStar, Upper), col="seagreen", size=1.1) + geom_point(data=xy, aes(x,y), size=1.1)
```



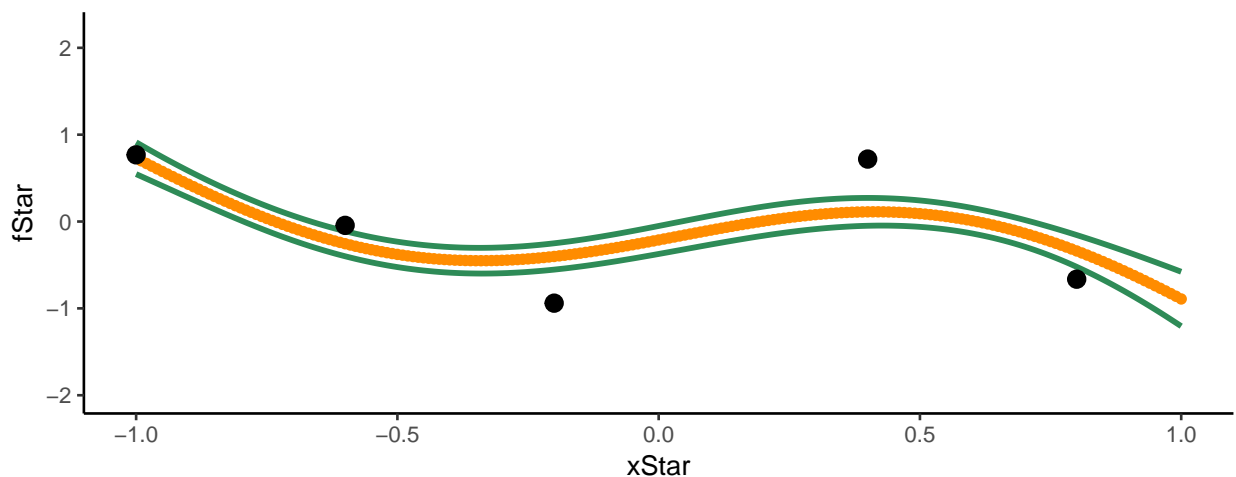
Compared to the plots in *b*) and *c*), the curve for the posterior mean of f is less straight/ more curvaceous than before. The probability bands has also changed and are thanks to the rise from two to five observed values more narrow, but also quite wiggly.

3.1.1.5 e)

The hyperparameter ι is now set to 1. The other parameters are unchanged and the same observations as in *d)* are used.

```
assign1E <- PosteriorGP(x=c(0.8, 0.4, -0.2, -0.6, -1), y=c(-0.664, 0.719, -0.94, -0.044, 0.768), xStar=1)
Upper1E <- assign1E$fStar + 1.96 * sqrt(diag(assign1E$cov_fStar))
Lower1E <- assign1E$fStar - 1.96 * sqrt(diag(assign1E$cov_fStar))

plotD <- data.frame(fStar=assign1E$fStar, xStar=assign1E$xStar, Lower=Lower1E, Upper=Upper1E)
xy <- data.frame(x=c(0.8, 0.4, -0.2, -0.6, -1), y=c(-0.664, 0.719, -0.94, -0.044, 0.768))
ggplot(plotD, aes(y=fStar, x=xStar)) + geom_point(col="darkorange") + ylim(-2,2.2) +
  geom_line(data=plotD, aes(xStar, Lower), col="seagreen", size=1.1) +
  geom_line(data=plotD, aes(xStar, Upper), col="seagreen", size=1.1) + geom_point(data=xy, aes(x,y), size=100)
```



Compared to the plot in *d)*, the probability bands obtained with ι set to 1 looks much smoother and lies much closer to the curve for the posterior mean of f . Another change is that curve for the posterior mean of f no longer goes through all of the observed values. Instead the fitted curve appears to be an average between the observed values that lies closest to each other. With ι set to 1 the curve becomes all too smooth.

3.1.1.6 Summarize - Assignment 1

As the number of observations increases we become more sure about the values of the posterior mean. How sure we become depends on the prior hyperparameters. σ_f controls the uncertainty for the prior and length scale controls how many observations that affects the estimated posterior mean.

3.1.2 Assignment 2

```
# Assignment 2
library(kernlab)
temps <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge")
temps$time <- 1:2190
temps$day <- rep(1:365, 6)
temps$index <- rep(1:5, 438)
thinTemps <- subset(temps, temps$index == 1)[,1:4]
```

3.1.2.1 a)

The code where we define our own square exponential kernel function can be seen below.

```
sqExpK <- function(sigmaf = 1, ell = 1) {  
  rval <- function(x, xStar = NULL) {  
    return(sigmaf * exp (-1/2 * ((x-xStar)/ell)^2 ))  
  }  
  class(rval) <- "kernel"  
  return(rval)  
}
```

When evaluated in the point $x = 1$ and $x' = 2$ the following value is given.

```
sqExpKFunc = sqExpK(sigmaf = 1, ell = 2) # sqExpKFunc is a kernel FUNCTION  
sqExpKFunc(1,2) # Evaluating the kernel in x=1, x'=2
```

```
## [1] 0.8824969
```

The *kernelMatrix* is tested for the vectors $\mathbf{x} = (1, 3, 4)^T$ and $\mathbf{x}_* = (2, 3, 4)^T$. Returned by this function is the covariance matrix $\mathbf{K}(x, x_*)$. This matrix can be interpreted as following:
The covariance between ($x=1, x'=4$) is 0.32.

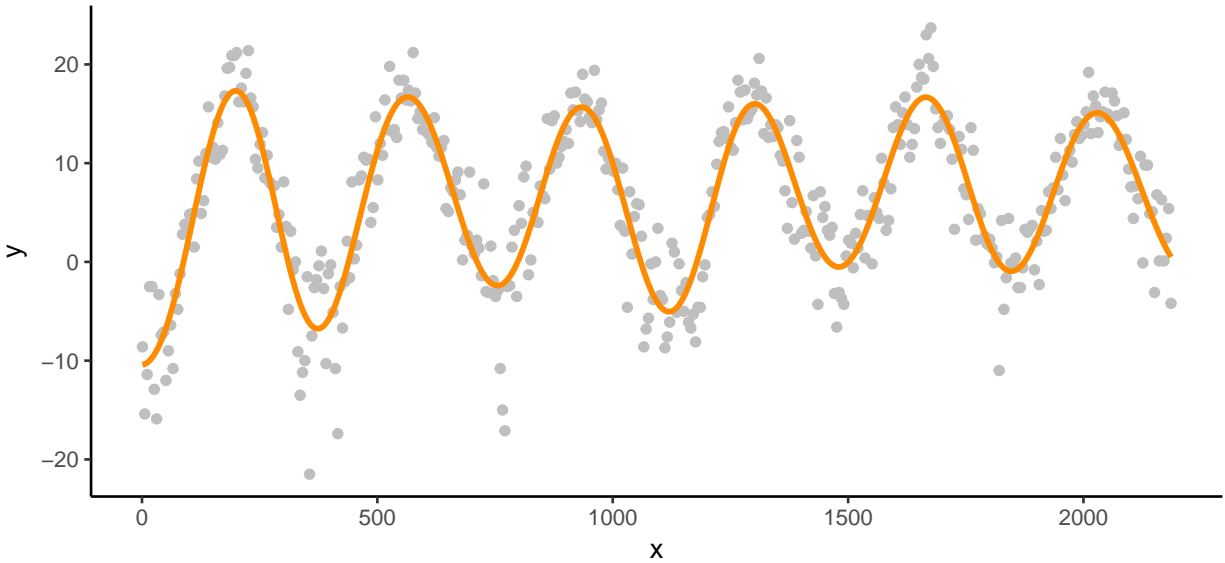
```
# Computing the whole covariance matrix K from the kernel.  
x <- as.matrix(c(1,3,4))  
xStar <- as.matrix(c(2,3,4))  
K <- kernelMatrix(kernel = sqExpKFunc, x = x, y = xStar) # So this is K(X,Xstar)  
row.names(K) <- c(1,3,4)  
colnames(K) <- c(2,3,4)  
K
```

```
## An object of class "kernelMatrix"  
##           2           3           4  
## 1 0.8824969 0.6065307 0.3246525  
## 3 0.8824969 1.0000000 0.8824969  
## 4 0.6065307 0.8824969 1.0000000
```

3.1.2.2 b)

The gaussian process regression model where the temperature is a function of time is estimated. At every data point in the training set is the posterior mean computed. In the graph below are the points in the data plotted together with the posterior mean. The prior for σ_n^2 is 66.85 which is the variance obtained when a simple quadratic regression is estimated. The prior Hyperparameters σ_f and ι are set to 20 and 0.2.

```
quadLM <- lm(temp ~ time + time^2, data=temps)  
sigma_n <- var(quadLM$residuals)  
  
sqExpKFunc = sqExpK(sigmaf = 20^2, ell = 0.2)  
reg <- gausspr(thinTemps$time, thinTemps$temp, kernel = sqExpKFunc, var=sigma_n)  
postMean <- data.frame(pred=predict(reg), y=thinTemps$temp, x=thinTemps$time)  
plotB <- ggplot(postMean, aes(x=x, y=y)) + geom_point(col="grey75") + geom_line(data=postMean, aes(x=x,  
plotB
```

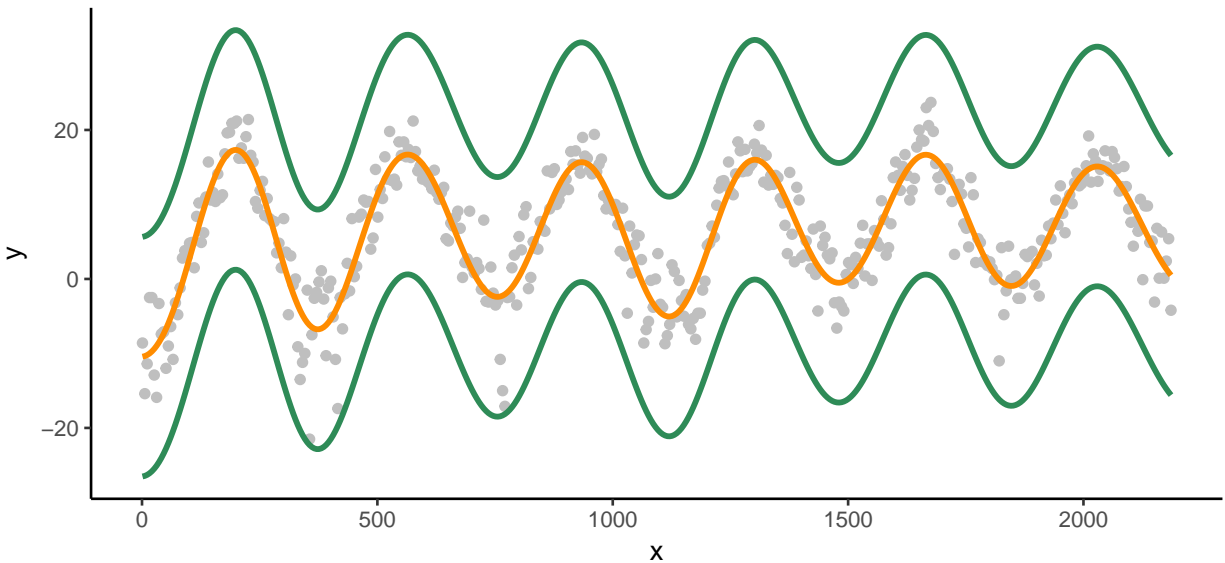


The orange curve are the predicted values and the grey dots the observed temperature. Overall, the fitted model seem to be rather good. There is some yearly variation which is modeled quite well.

3.1.2.3 c)

The algorithm implemented in 1.a) that was used for computing the posterior distribution of f is now used for computing the posterior variance of the results presented in 2.b). The result is presented by adding 95 % posterior probability bands to the plot of the posterior means presented in the previous exercise.

```
covar_f <- PosteriorGP(x=thinTemps$time, y = thinTemps$temp, xStar = thinTemps$time,
  hyperParam=c(20^2, 0.2),sigmaNoise =sigma_n )
probBands <- data.frame(Upper=postMean[,1] + 1.96 * sqrt(diag(covar_f$cov_fStar)) , Lower = postMean[,1]
plotB + geom_line(data=probBands, aes(x=x, y=Upper), size=1.1, col="seagreen") + geom_line(data=probBands, aes(x=x, y=Lower), size=1.1, col="seagreen")
```

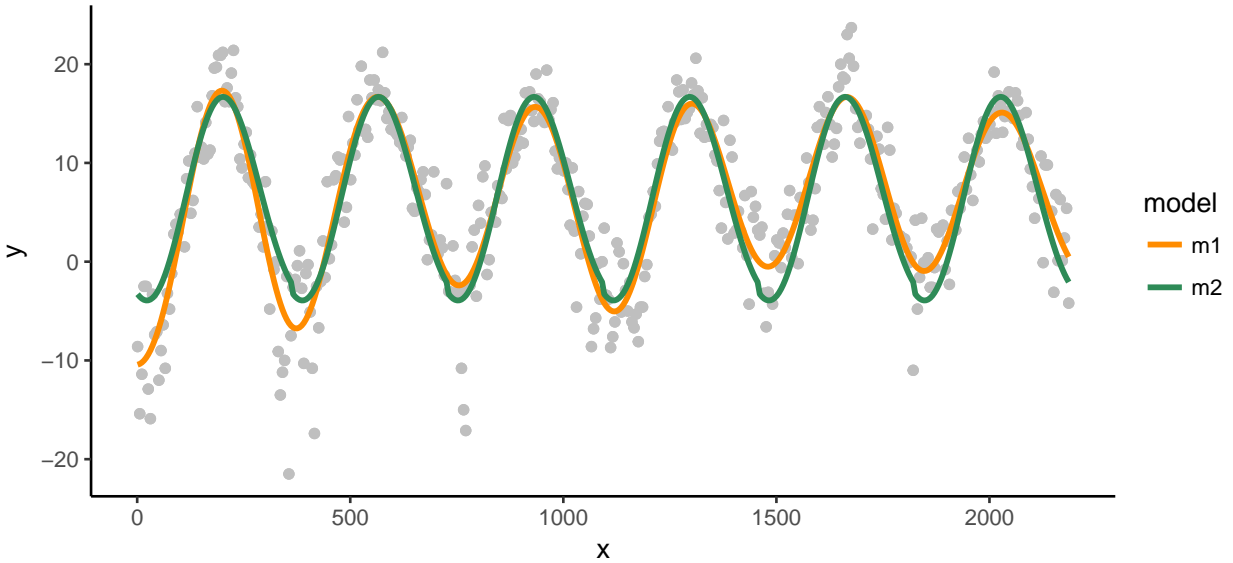


The 95 % posterior probability bands looks reasonable as all the observed temperatures lies inside the bands. However, the bands are perhaps a bit too wide. For example, the 95 % interval for the temperature during summer goes from around 0 to 35.

3.1.2.4 d)

A model with temperature as a function of day is estimated. The posterior mean of this model is compared to the posterior mean of the model which had temperature as a function of time.

```
quadLM2 <- lm(temp ~ day + day^2, data=temps)
sigma_n2 <- var(quadLM2$residuals)
# Small difference, 67.3641 vs 64.10528
sqExpKFuncD = sqExpK(sigmaf = 20^2, ell = 1.2)
regD <- gausspr(thinTemps$day, thinTemps$temp, kernel = sqExpKFuncD, var=sigma_n2)
postMeanD <- data.frame(pred=predict(regD), y=thinTemps$temp, x=thinTemps$time)
postMT <- data.frame(rbind(postMean, postMeanD), model=c(rep("m1", 438), rep("m2", 438)))
ggplot(postMT, aes(y=y, x=x)) + geom_point(col="grey75") + geom_line(aes(x=x, y=pred, col=model), size=
```



A comparison of the curves gives that the model with day as input is constant and shows the exactly same pattern every year. Positive with this model is that it takes historical data into account when estimating the posterior mean. A negative aspect with this model is that it not allows any variation for the estimates given for specific day.

The model that has time as input is more flexible. It allows the posterior mean for a day to differ between years as the estimate mainly is affected by the temperatures observed the weeks or days before during that year. The observed values the years before does not have any larger impact on the estimate.

You could say that the pro for one model is a con for the second, and vice versa.

3.1.2.5 e)

A periodic kernel is implemented and the formula for the kernel is given below.

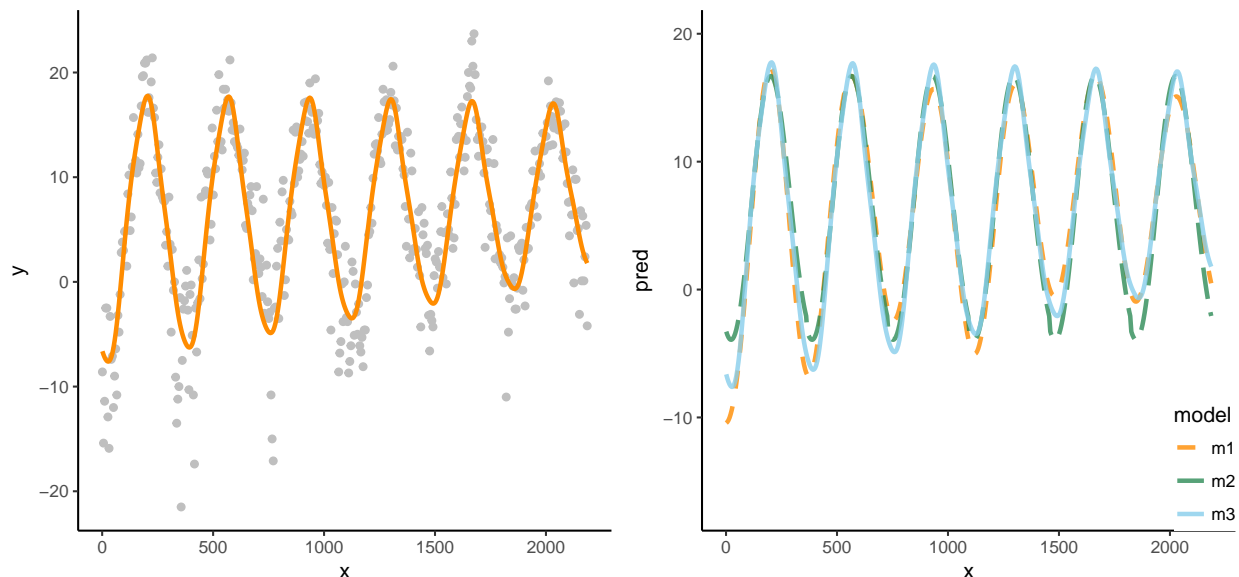
$$K(x, x') = \sigma_f^2 \times \exp\left(\frac{2\sin^2(\pi|x - x'|/d)}{\iota_1^2}\right) \times \exp\left(-0.5 \times \left(\frac{x - x'}{\iota_1^2}\right)^2\right)$$

Two new parameters are then introduced. The correlation between the same day in different years is controlled by the parameter ι_2^2 and the period is specified with the parameter d . A Gaussian process model is estimated with time as explanatory variable.

```

PeriodicK <- function(sigmaf = 1, ell_one = 1, ell_two=1, d=1){
  rval <- function(x, xStar = NULL) {
    return(sigmaf * exp(-(2*sin(pi*abs(x-xStar)/d)^2)/ ell_one^2) *
           exp(-0.5 * (abs(x-xStar)^2 / ell_two^2)))
  }
  class(rval) <- "kernel"
  return(rval)
}
PeriodicKFunc = PeriodicK(sigmaf = 20^2, ell_one = 1, ell_two=10, d=365/sd(thinTemps$time))
#PeriodicKFunc is a kernel FUNCTION
regE <- gausspr(thinTemps$time, thinTemps$temp, kernel = PeriodicKFunc, var=sigma_n)
postMeanE <- data.frame(pred=predict(regE), y=thinTemps$temp, x=thinTemps$time)
plotE <- ggplot(postMeanE, aes(x=x, y=y)) + geom_point(col="grey75") + geom_line(data=postMeanE, aes(x=x, y=y))
postMT <- rbind(postMT, data.frame(postMeanE, model="m3"))
plotAll <- ggplot(postMT, aes(x=x, y=pred)) + geom_line(aes(col=model, linetype=model),size=1.1, alpha=0.5)
library(gridExtra)
grid.arrange(plotE, plotAll, ncol=2)

```



The periodic kernel combines both data from earlier years and data from days closer to the actual day. In practice it works as a combination of the models discussed in 2.d). The fitted curve in the plot to the left is similar to the earlier ones, with some exceptions. The posterior mean for the temperature during the winter increases for every year and the temperature for the summer looks to be constant.

This result very well shows the ability of the periodic kernel. For some parts of the year the posterior mean is constant and for some it is more flexible.

3.1.3 Assignment 3

```

# Assignment 3
library(AtmRay)
data <- read.csv("https://github.com/STIMaLiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud.csv",
                 header=FALSE, sep=",")
names(data) <- c("varWave", "skewWave", "kurtWave", "entropyWave", "fraud")

```



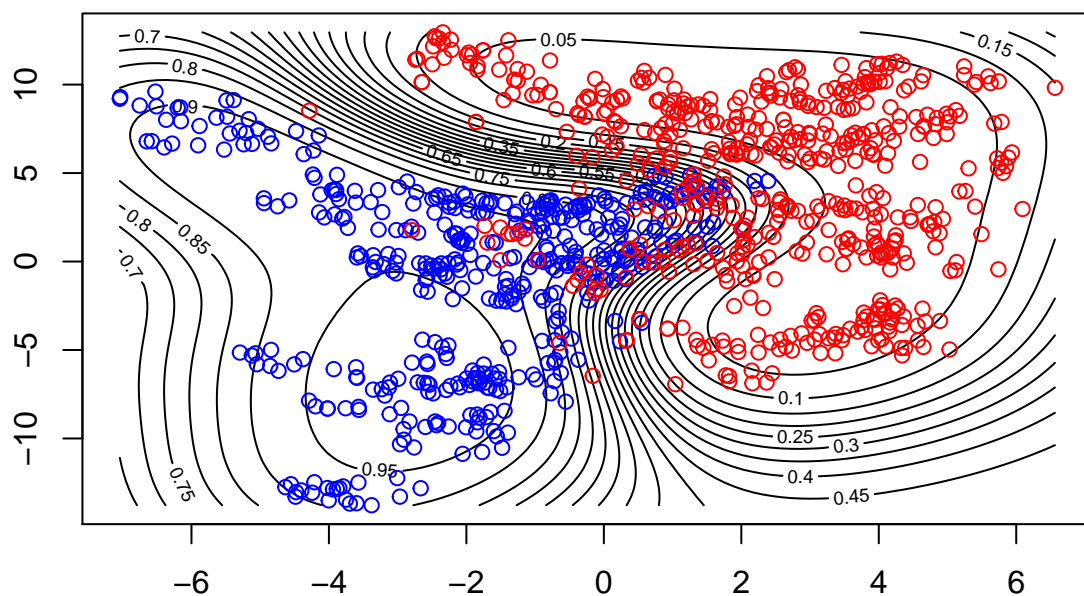
```
data[,5] <- as.factor(data[,5])
set.seed(111); SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
Train <- data[SelectTraining, ]
Test <- data[-SelectTraining, ]
```

3.1.3.1 a)

```
# a)
cfA <- gausspr(fraud ~ varWave + skewWave, data=Train)
gridX <- seq(min(Train$varWave), max(Train$varWave), length=100)
gridY <- seq(min(Train$skewWave), max(Train$skewWave), length=100)
gridP <- meshgrid(gridX, gridY)
gridP <- cbind(c(gridP$x), c(gridP$y))
gridP <- data.frame(gridP)
names(gridP) <- names(Train)[1:2]
```

A model with the variables *varWave* and *skewWave* as input variables and *fraud* as target variable is estimated. The contours of the prediction probability for an observation to get classified as fraud or not is plotted below. In the contour plot is also training data added where *fraud*=1 are coloured blue and observations with *fraud*=0 coloured red.

```
probPredsA <- predict(cfA, gridP, type="probabilities")
contour(x=gridX, y= gridY,z=t(matrix(probPredsA[,2],100)),20)
points(Train[Train[,5]== 1,1],Train[Train[,5]== 1,2],col="blue")
points(Train[Train[,5]== 0,1],Train[Train[,5]== 0,2],col="red")
```



It can be seen that the model relatively well separates the training data correct with the prediction probabilities obtained with the model. This conclusion is also confirmed by the confusion matrix for the classifier on the training data and the accuracy.

```
# predict on the training set
confMatA <- table(predict(cfA,Train[,1:2]), Train[,5]) # confusion matrix
confMatA
```

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

```
cat("Accuracy:", sum(diag(confMatA)) / sum(confMatA))
```

```
## Accuracy: 0.932
```

3.1.3.2 b)

The estimated model is then used for making predictions on the test set and the results are presented in a confusion matrix. The accuracy for the test data is almost the same as for the training data. In other words does the model seem to be a rather good fit.

```
confMatB <- table(predict(cfA,Test[,1:2]), Test[,5]) # confusion matrix
confMatB
```

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

```
cat("Accuracy:", sum(diag(confMatB)) / sum(confMatB))
```

```
## Accuracy: 0.9354839
```

3.1.3.3 c)

A new model with two more input variables, *kurtWave* and *entropyWave*, is fitted. The confusion matrix and the accuracy of this model when the observations in the test data are predicted is presented below. A higher accuracy is received as all but one of the observations are classified correctly with this model.

```
cfC <- gausspr(fraud ~ varWave + skewWave + kurtWave + entropyWave, data=Train)
```

```
## Accuracy: 0.9973118
```

4 Lab 4

4.1 Individual/Group report

4.1.1 Assignment 1

```
library(ggplot2)
library(dlm)
library(gridExtra)
library(mvtnorm)
```

4.1.1.1 a) - Simulate the model

$$\begin{aligned} y_t &= Cx_t + \epsilon, & \epsilon_t &\sim \mathcal{N}(0, \Omega_\epsilon) \\ x_t &= Ax_{t-1} + Bu_t + v_t, & v_t &\sim \mathcal{N}(0, \Omega_v) & x_0 &\sim \mathcal{N}(\mu_0, \Sigma_0) \end{aligned}$$

Set $x_0 = 0$ and simulate $T=100$ observations from the following model.

$$\begin{aligned} y_t &= 0.93x_t + \epsilon, & \epsilon &\sim \mathcal{N}(0, \Omega_\epsilon = 0.5) \\ x_t &= x_{t-1} + v_t, & v_t &\sim \mathcal{N}(0, \Omega_v = 0.1) & x_0 &\sim \mathcal{N}(\mu_0 = 0, \Sigma_0 = 10) \end{aligned}$$

A linear Gaussian state space model is implemented in line with the given instructions. 100 observations are simulated from this model and the obtained states(x) and observations (y) are plotted in figure 1.

```
# Assignment 1

# a) - Simulate the model
ssm1 <- function(C=matrix(0), A=matrix(1), B=matrix(0), u=matrix(0), x0=0,
                 omegaE=0, omegaV=0, TimeS=0) {
  xt <- 0
  yt <- 0
  vt <- rnorm(1, 0, omegaV)
  xt[1] <- A%*%x0 + vt
  et <- rnorm(1, 0, omegaE)
  yt[1] <- C%*%xt + B%*%u + et
  for(i in 2:TimeS){
    vt <- rnorm(1, 0, omegaV)
    xt[i] <- A%*%xt[i-1] + vt
    et <- rnorm(1, 0, omegaE)
    yt[i] <- C%*%xt[i] + B%*%u + et
  }
  res <- data.frame(Val=c(xt,yt), t= rep(1:TimeS,2),
                   Value=rep(c("State","Observation"), each=TimeS))
  return(res)
}
set.seed(1234)
resA <- ssm1(C = matrix(0.93), x0 = 0, omegaE = sqrt(0.5), omegaV = sqrt(0.1), TimeS = 100)
plotA <- ggplot(resA, aes(x=t, y=Val, col=Value)) + geom_line(size=1) + theme_classic() +
  ylim(-4,4) + scale_color_manual(values=c("skyblue", "royalblue"))
plotA + labs(caption = "Figure 1")
```

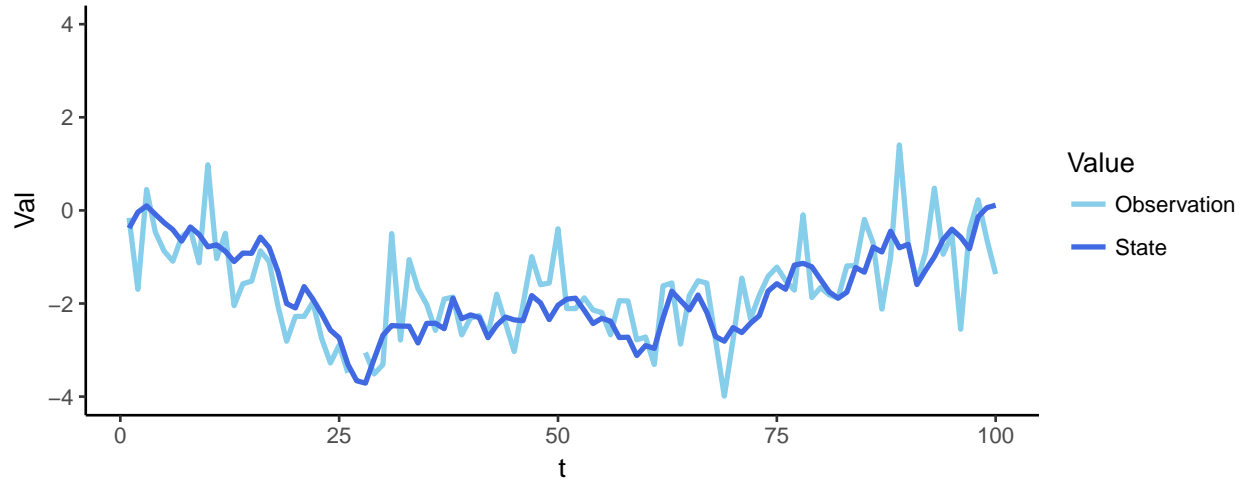


Figure 1

4.1.1.2 b) - Filtering: Sequential state inference

The point estimates for the Kalman filter and the 0.95 probability bands for the point estimates are plotted together with the states and observations simulated in *a*).

```
KalmanF <- function(yt, C=matrix(0), A=matrix(1),
                    B=matrix(0), u=matrix(0), omegaE=0, omegaV=0, TimeS=0){
  KalmanMat <- matrix(c(0,0), ncol=2, nrow=101)
  for(i in 1:TimeS){
    # Prediction update
    muBar_t <- A %*% KalmanMat[i,1] + B%*%u
    sigmaBar_t <- A%*%KalmanMat[i,2]%*%t(A) + omegaV
    # Measurement update
    K_t <- sigmaBar_t%*%t(C) %*% solve(C%*%sigmaBar_t%*%t(C) + omegaE)
    KalmanMat[i+1,1] <- muBar_t + K_t %*% (yt[i] - C%*%muBar_t)
    KalmanMat[i+1,2] <- (diag(ncol(K_t))-K_t%*%C) %*% sigmaBar_t
  }
  KalmanFrame <- data.frame(KalmanMat[-1, ], Time=1:TimeS)
  return(KalmanFrame)
}

resB <- KalmanF(yt = resA[101:200,1], C = matrix(0.93), omegaE = 0.5, omegaV = 0.1, TimeS=100)
resB_1 <- data.frame(Val=c(resB[,1],resB[,1]+1.96*sqrt(resB[,2]),resB[,1]-1.96*sqrt(resB[,2])), Time=resB$Time)

plotA + geom_line(data=resB_1, aes(x=Time, y=Val, col=Value), size=1) +ylim(-5,4)+
  scale_colour_manual(values=c("red3", "darkorange", "skyblue", "royalblue", "darkorange"))+
  labs(x="Time", title="Kalman filter estimates with 0.95 probability intervals", caption="Figure 2")
```

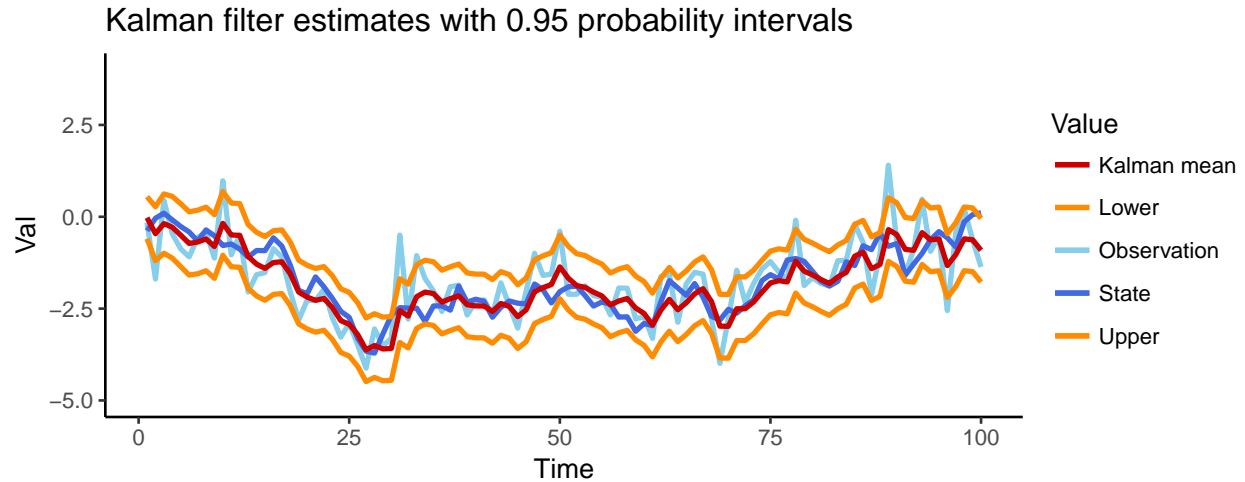


Figure 2

In figure 3 are the Kalman filter estimates and probability intervals obtained by the implemented Kalman algorithm compared to the estimates returned by the *dmlFilter* function from the *dml* package.

```
filteringB <- dlmFilter(y=resA[101:200,1], dlm(m0=0, V=0.5, W=0.1, C0=10, FF=0.93, GG=1) )

filterB <- data.frame(y=(filteringB$m)[-1])
filterVar <- unlist(dlmSvd2var(u = filteringB$U.C, d = filteringB$D.C))
filterB[101:300,1] <- c(filterB$y + 1.96*sqrt(filterVar[-1]),
                        filterB$y - 1.96*sqrt(filterVar[-1]))
filterB$Time <- rep(1:100, 3)
filterB$Value <- rep(c("dml Kalman mean", "dml Upper", "dml Lower"), each=100)

ggplot(resB_1, aes(x=Time, y=Val, col=Value, linetype=Value)) + geom_line(size=1) + theme_classic() + ylin
```

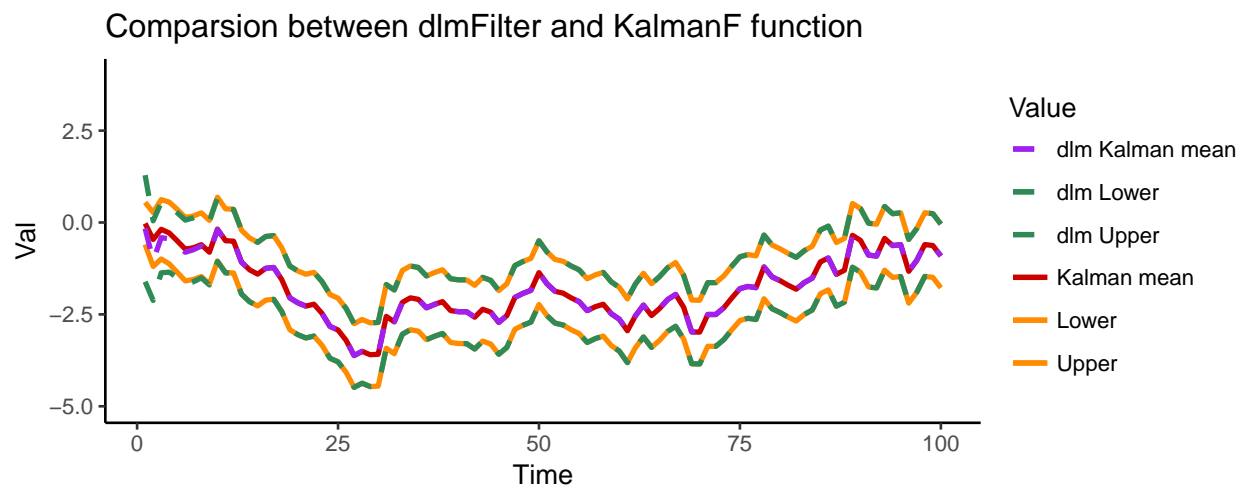


Figure 3

Except for the first values are the point estimates and the probability intervals identical.

4.1.1.3 c) - Prediction of state and data by simulation

When simulating we use k equals 5 and i goes from 1:1000.

$$x_{T+k}^{(i)} \sim p(x_{T+k} \mid Y_{1:T})$$

$$y_{T+k}^{(i)} \sim p(y_{T+k} \mid x_{T+k}^{(i)})$$

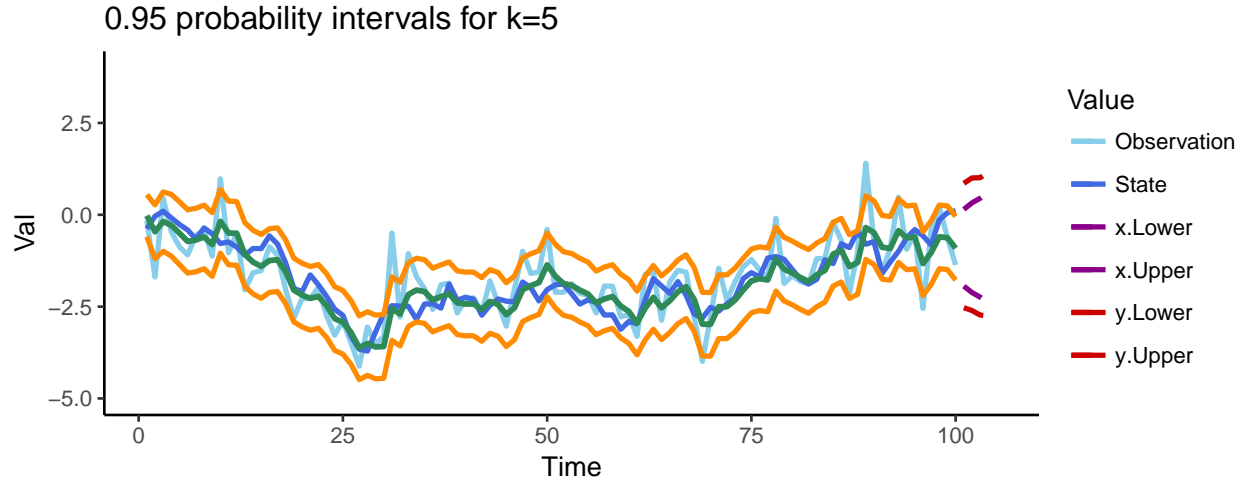
The probability intervals for 5 future states and observations are plotted in figure 2 together with the data plotted in figure 2. The 5 future time steps are simulated 10 000 times and a 0.95 probability interval is given by the quantiles from the sample.

```
PredFunc <- function(C=matrix(0), A=matrix(1), B=matrix(0), u=matrix(0), x0=0,
                    omegaE=0, omegaV=0, TimeS=0){
  xt <- 0
  yt <- 0
  vt <- rnorm(1, 0, omegaV)
  xt[1] <- x0 + vt
  et <- rnorm(1, 0, omegaE)
  yt[1] <- C%%xt + B%%u + et
  for(i in 2:TimeS){
    vt <- rnorm(1, 0, omegaV)
    xt[i] <- A%%xt[i-1] + vt
    et <- rnorm(1, 0, omegaE)
    yt[i] <- C%%xt[i] + B%%u + et
  }
  res <- data.frame(x=c(xt, y=yt))
  return(res)
}

PredC <- as.data.frame(matrix(nrow=10, ncol=10000))
for(i in 1:10000){
  PredC[,i] <- PredFunc(C = matrix(0.93), x0 = rnorm(1, mean=resB$X1[100],
                    sd=sqrt(resB$X2[100])), omegaE = sqrt(0.5), omegaV = sqrt(0.1),
                    TimeS = 5)
}

quanTs <- t(apply(PredC, 1, quantile, probs=c(0.025, 0.975)))
quanTsFrame <- data.frame(x=c(quanTs[,1], quanTs[,2]), Time= 100+rep(1:5, 4),
                    Var=rep(c("x", "y"), 2, each=5), InterV=rep(c("Lower", "Upper"), each=10))
quanTsFrame$Interval <- interaction(quanTsFrame$Var, quanTsFrame$InterV)
quanTsFrame$Interval <- factor(quanTsFrame$Interval, levels=rev(levels(quanTsFrame$Interval)))

ggplot(resA, aes(x=t, y=Val, col=Value)) + geom_line(size=1) + theme_classic() + ylim(-5,4) +
  geom_line(data=resB_1[1:100,], aes(x=Time, y=Val), col="seagreen", size=1.1) +
  geom_line(data=resB_1[101:200,], aes(x=Time, y=Val), col="darkorange", size=1) +
  geom_line(data=resB_1[201:300,], aes(x=Time, y=Val), col="darkorange", size=1) +
  geom_line(data=quanTsFrame, aes(x=Time, y=x, col=Interval), size=1, linetype="dashed") +
  scale_color_manual(values=c("skyblue", "royalblue", "darkmagenta", "darkmagenta", "red3", "red3")) + labs
```



We notice how the width of these bands increase with time. This implies that we become more unsure for more distant future time steps.

4.1.1.4 d) - State and model parameter inference

Given the information from the lab instructions we create our FFBS function in order to be able to update the x values. We implement a bayesian linear regression to be able to update our θ values. This will be used in our gibbs function and the update process will continue iteratively for 1000 iterations. These iterations can be seen below. It seems to converged rather quickly.

```
KalmanF_2 <- function(yt, C=matrix(0), A=matrix(1),
                     B=matrix(0), u=matrix(0), omegaE=0, omegaV=0, TimeS=0){
  KalmanMat <- matrix(c(0,0,0,0), ncol=4, nrow=101)
  for(i in 1:TimeS){
    # Prediction update
    KalmanMat[i+1,3] <- A %*% KalmanMat[i,1] + B%*%u #muBar_t
    KalmanMat[i+1,4] <- A%*%KalmanMat[i,2]%*%t(A) + omegaV #sigmaBar_t
    # Measurement update
    K_t <- KalmanMat[i+1,4] %*%t(C) %*% solve(C%*%KalmanMat[i+1,4] %*%t(C) + omegaE)
    KalmanMat[i+1,1] <- KalmanMat[i+1,3] + K_t %*% (yt[i] - C%*%KalmanMat[i+1,3]) #mu_t
    KalmanMat[i+1,2] <- (diag(ncol(K_t))-K_t%*%C) %*% KalmanMat[i+1,4] # sigma_t
  }
  KalmanFrame <- data.frame(KalmanMat[-1,], Time=1:TimeS)
  return(KalmanFrame)
}

BS <- function(A, Kalman, x){
  x_t <- data.frame(Val=0)
  for(k in (nrow(Kalman)-1):1){
    ht <- Kalman[k, 1] + Kalman[k, 2]%*%t(A)%*%solve(Kalman[k+1, 4]) %*% (x[k+1] - Kalman[k+1, 3])
    Ht <- Kalman[k,2]-Kalman[k,2]%*%t(A)%*%solve(Kalman[k+1,4])%*%A%*%Kalman[k,2]
    x_t[k,1] <- rnorm(1, mean=ht, sd=sqrt(Ht))
  }
  return(x_t)
}
```

```

bayesReg <- function(beta0, Xt, y, omega0, cov0){
  betaHat <- solve(t(Xt)%*%Xt)%*% t(Xt) %*% as.matrix(y)
  sigma0 <- diag(omega0/cov0,2)
  betaNew <- solve(t(Xt)%*%Xt + sigma0) %*% (t(Xt)%*%Xt%*%betaHat + sigma0%*%beta0)
  sigmaNew <- t(Xt)%*%Xt + sigma0
  Vari <- omega0 * solve(sigmaNew)
  coef <- rmvnorm(n=length(y), mean=betaNew, sigma=sqrt(diag(sqrt(diag(Vari)),length(betaNew))))
  return(coef)
}

# Gibbs sampling
y = resA[101:200, 1]
x = resA[1:100, 1]
set.seed(311015)
theta <- data.frame(matrix(ncol=2, nrow=1000)) #nrow sets number of sims
theta[1,2] <- 1 #start value for theta
Xt= matrix(data= c(rep(1,100),resA[1:100,1]), ncol=2)
for(j in 1:nrow(theta)){
  KalmanRes <- KalmanF_2(yt = y, C = matrix(theta[j,2]), omegaE = 0.5, omegaV = 0.1,
                        Times = 100)
  NewX <- (BS(A = matrix(1), Kalman = KalmanRes, x=x))
  Xt[1:99,2] <- as.numeric(NewX[,1])
  Xt[100,2] <- KalmanRes[100,1]
  coefs <- bayesReg(beta0 = matrix(data=c(0,0), ncol=1),Xt=Xt, y=y, omega0=0.5,cov0=100)
  theta[j+1,] <- colMeans(coefs)
}
ggplot(theta, aes(y=X2, x=1:nrow(theta))) + geom_line() + theme_classic() + labs(caption="Figure 5", title="")

```

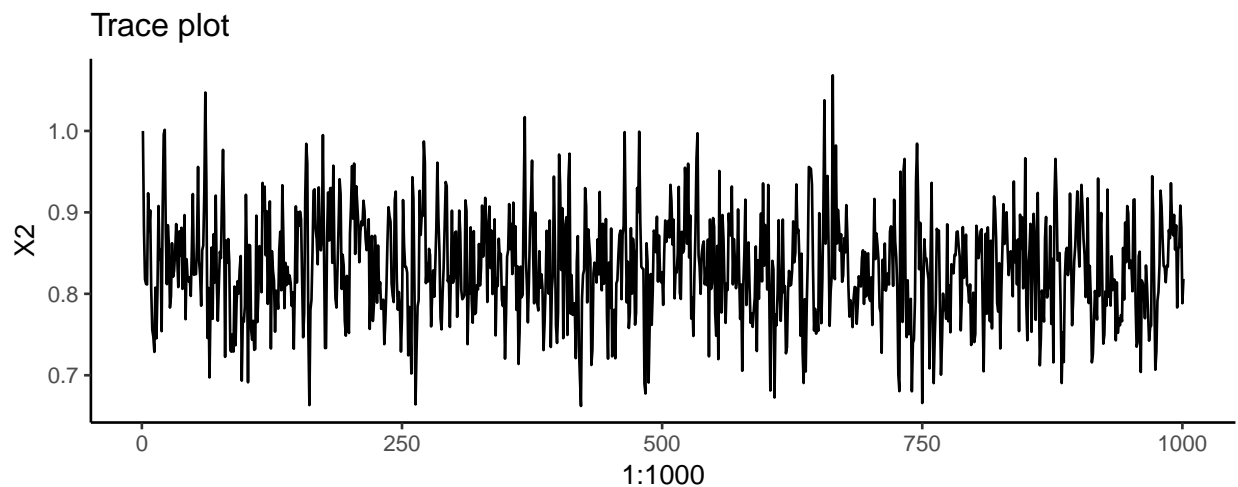


Figure 5

```

ggplot(theta, aes(x=X2)) + geom_density() + theme_classic() +
  geom_vline(xintercept = c(mean(theta$X2),0.93), col=c("red","blue")) + labs(caption="Figure 6", title="")

```

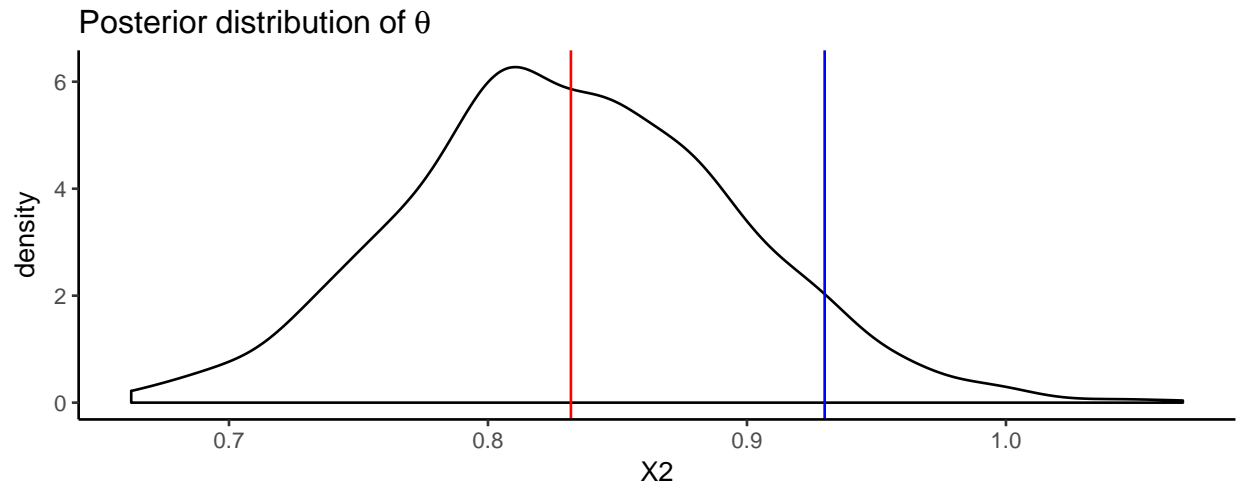



Figure 6

In the density plot is the probability for different theta values plotted. The mean of our theta equals to 0.8320197 which is close to our true value. The true value, 0.93, could be obtained but the probability for this to happen is low.

4.1.2 Assignment 2

4.1.2.1 a) - Estimating the variance components by MLE.

```
# Assignment 2
load("C:\\Users\\Gustav\\Documents\\AdvML\\Lab4\\CAPM.Rda")

# a) - Estimating the variance components by MLE

CAPMv <- CAPM_data[,c(10,17,18)]
Zt <- matrix(CAPM_data$MARKET - CAPM_data$RKFREE)
Yt <- matrix(CAPMv$IBM - CAPMv$RKFREE)

buildLocalTrend <- function(x,data){
  V = exp(x[1])
  W = diag(exp(x[2:3]),2,2)
  return(dlm(
    m0 = c(0,1),
    C0 = diag(c(100,0.5),2),
    FF = matrix(c(1,1),1,2),
    GG = diag(2),
    JFF = matrix(c(0,1),1,2),
    V = V,
    W = W,
    X=data))
}

initVal <- c(1,1,1) # Initial values for optim on the estimated parameters
MLEs <- dlmMLE(Yt, parm = initVal, build = buildLocalTrend, data = Zt)
dlmWithMLEs <- buildLocalTrend(MLEs$par, data = Zt)
```

By MLE the following estimates are given for the variance components.

$$\sigma_{\epsilon}^2 = 0.0023275,$$

$$\sigma_{v^{(1)}}^2 = 0.0000112,$$

$$\sigma_{v^{(2)}}^2 = 0.0006338$$

4.1.2.2 b) - Filtering and smoothing

The filtering and smoothing distribution is computed for α and β . In the figures below are these distributions plotted together with 0.95 probability intervals.

The respective distributions for the alpha parameter are shown in the figures below. As expected, the same pattern is seen in both figures and a more smooth curve is returned for the smoothig distribution.

```
## Filtering ##
Filter_2b <- dlmFilter(y = Yt, dlmWithMLEs)

# Variances
u1 <- matrix((t(data.frame(Filter_2b$U.C)))[c(seq(1,242,2)),1])
u2 <- matrix((t(data.frame(Filter_2b$U.C)))[c(seq(2,242,2)),2])
u_1 <-list()
u_2 <- list()
for(i in 1:120){
  u_1[i]<-c(u1[i+1])
  u_2[i]<-c(u2[i+1])
}
Filter_y <- data.frame(alpha=c(Filter_2b$m)[2:121],beta=c(Filter_2b$m)[123:242])
Filter_y$alphaVar <- unlist(dlmSvd2var(u = u_1, d = matrix(Filter_2b$D.C[-1,1])))
Filter_y$betaVar <- unlist(dlmSvd2var(u = u_2, d = matrix(Filter_2b$D.C[-1,2])))
# Intervals
Filter_y$alphaLower <- Filter_y$alpha - 1.96*sqrt(Filter_y$alphaVar)
Filter_y$alphaUpper <- Filter_y$alpha + 1.96*sqrt(Filter_y$alphaVar)
Filter_y$betaLower <- Filter_y$beta - 1.96*sqrt(Filter_y$betaVar)
Filter_y$betaUpper <- Filter_y$beta + 1.96*sqrt(Filter_y$betaVar)

alphaFilter <- ggplot(Filter_y, aes(x=1:120, y=alpha)) + geom_line(col="seagreen", size=1) + theme_classic()
  geom_line(data=Filter_y, aes(x=1:120, y=alphaLower), col="darkorange", size=1) +
  geom_line(data=Filter_y, aes(x=1:120, y=alphaUpper), col="darkorange", size=1)

betaFilter <- ggplot(Filter_y, aes(x=1:120, y=beta)) + geom_line( col="seagreen", size=1) + theme_classic()
  geom_line(data=Filter_y, aes(x=1:120, y=betaLower), col="darkorange", size=1) +
  geom_line(data=Filter_y, aes(x=1:120, y=betaUpper), col="darkorange", size=1)

#### Smoothed ####
Smooth_2b <- dlmSmooth(y = Yt, dlmWithMLEs)
Smooth_frame <- data.frame(alpha=Smooth_2b$s[-1,1], beta=Smooth_2b$s[-1,2])

covList = dlmSvd2var(Smooth_2b$U.S, Smooth_2b$D.S)
varMat = t(sapply(covList, FUN=function(x) sqrt(diag(x))))

Smooth_frame$alphaSd <- varMat[-1,1]
Smooth_frame$betaSd <- varMat[-1,2]
# Intervals
Smooth_frame$alphaLower <- Smooth_frame$alpha - 1.96*Smooth_frame$alphaSd
Smooth_frame$alphaUpper <- Smooth_frame$alpha + 1.96*Smooth_frame$alphaSd
```

```

Smooth_frame$betaLower <- Smooth_frame$beta - 1.96*Smooth_frame$betaSd
Smooth_frame$betaUpper <- Smooth_frame$beta + 1.96*Smooth_frame$betaSd

alphaSmooth <- ggplot(Smooth_frame, aes(x=1:120, y=alpha)) + geom_line(size=1, col="seagreen") + theme_
  geom_line(data=Smooth_frame, aes(x=1:120, y=alphaLower), size=1, col="darkorange") +
  geom_line(data=Smooth_frame, aes(x=1:120, y=alphaUpper), size=1, col="darkorange")

betaSmooth <- ggplot(Smooth_frame, aes(x=1:120, y=beta)) + geom_line(size=1, col="seagreen")+theme_classic
  geom_line(data=Smooth_frame, aes(x=1:120, y=betaLower), size=1, col="darkorange") +
  geom_line(data=Smooth_frame, aes(x=1:120, y=betaUpper), size=1, col="darkorange")

grid.arrange(alphaFilter, alphaSmooth, ncol=2)

```

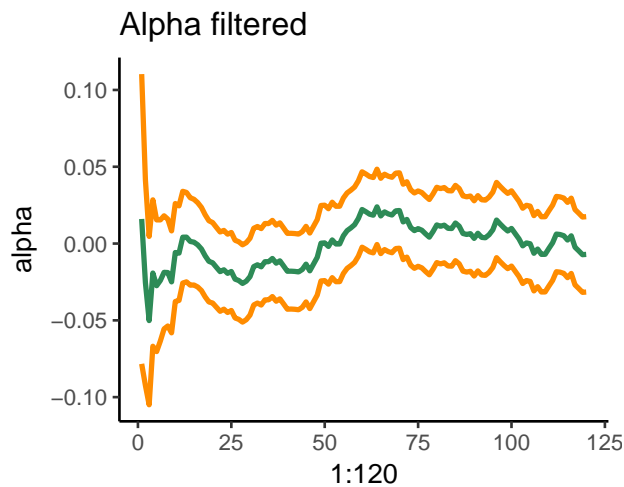


Figure 7

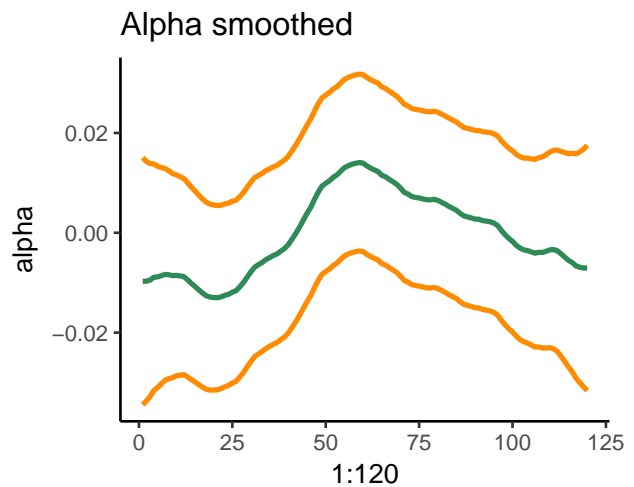


Figure 8

For the beta parameter is the pattern a little bit different for the respective distribution. In figure 10, for the smooth distribution, the value is increasing from around time point 50 and onwards. This pattern was not seen in the figure for the filter distribution.

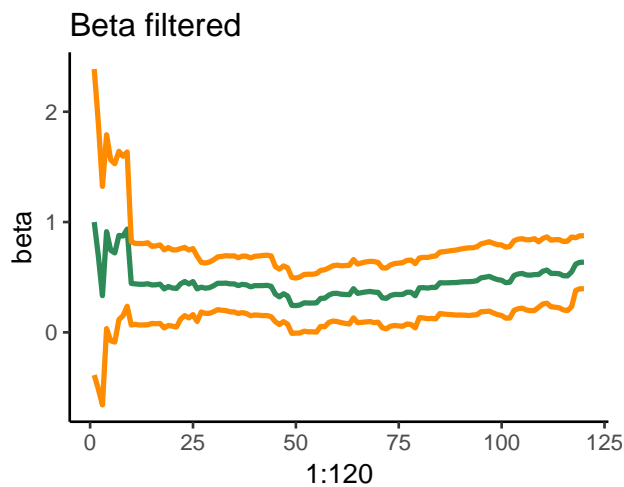


Figure 9

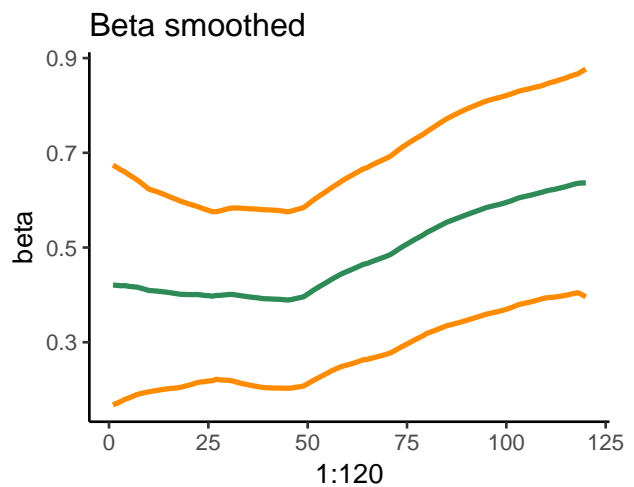


Figure 10

4.1.2.3 c) - Average sensitivity pre and post 1982 - a retrospective analysis

The retrospective analysis is performed by first computing the filtering distribution for the time points pre 1982. This is done by only using data for the first 48 observed time steps. Then, the same thing is done once again but for the data points from time point 49 to 120.

The `dmlmBSample` function is used for making the *BS* step in the *FFBS* algorithm on the respective filtering distribution. With this approach are the mean values presented below obtained for the respective time period. Hence, it is concluded that the average sensitivity was higher after 82 than before 82.

```
filter_1 <- dlmFilter(Yt[1:48], buildLocalTrend(MLEs$par, Zt[1:48]))
filter_2 <- dlmFilter(Yt[49:120], buildLocalTrend(MLEs$par, Zt[49:120]))

set.seed(311015)
mean_1 <- mean(dlmBSample(filter_1)[,2])
mean_2 <- mean(dlmBSample(filter_2)[,2])

cat("Mean before 82:", mean_1)
```

```
## Mean before 82: 0.4105294
```

```
cat("Mean after 82:", mean_2)
```

```
## Mean after 82: 0.6313772
```

4.1.2.4 d) - Bayesian inference for the variance components

The first way is to use a likelihood that integrates out the states (i.e. will only be a function of the three sigma you have). This likelihood is then multiplied by a prior for the unknown's sigmas. This posterior can be sampled from using, for example, Metropolis-Hastings.

See slide 6, Lecture 2 of how a likelihood a LGSS looks when it integrated the states.

4.1.3 Comments from Matias

När ni kör Gibbs sampling för att skatta theta så verkar det som att ni simulerar nya data $y_{1:T}$ hela tiden. Det ska ni inte göra, om ni kollar på posteriorn så är ni intresserade av $p(\theta, x_{1:T} | y_{1:T})$ - dvs BETINGAT på det data ni har (glöm inte att Bayesianer ALLTID betraktar data som känt. Det som är okänt i en bayesiansk analys är det vi försöker skatta). Resten av uppgift 1 d) har ni tänkt rätt på men åtgärda detta. Kör minst 5000 dragningar när ni samplar posteriorn om det går (koden blir snabbare när ni slipper simulera y hela tiden).

Det ni är intresserade av är att beräkna sannolikheten $\Pr(\text{mean}(\beta_{1:48}) < \text{mean}(\beta_{49:100}) | y_{1:100})$. NOTERA att smoothing ger marginal fördelning för EN tidpunkt. Här är ni intresserade av simultan (joint) fördelning $\beta_{1:100} | y_{1:100}$. Drag från denna fördelning fås med FFBS. Det är alltså viktigt att ni kan skilja på vad det är FFBS ger (drag från en simultan fördelning givet $y_{1:T}$) och smoothing (drag från en marginal fördelning givet $y_{1:T}$)

2.d). Det finns i princip två vägar att gå.

Första vägen är att använda en likelihood som integrerar ut states (dvs kommer bara vara en funktion av de tre sigma ni har). See slide 6, Lecture 2 för hur en likelihood in en LGSS ser ut när man integrerat ut states. Denna likelihood multipliceras sedan med en prior för de okända sigma's ni har, och denna posterior kan man sampla från med t.ex Metropolis-Hastings.

Den andra vägen är att använda Gibbs sampling på ett liknande sätt som i uppgift 1 d). I 1 d) så hade ni en lutning som var okänd och en varians som var känd - här är det tvärtom: ni kommer ha en okänd varians. Vi vet från Bayesian learning kursen att en normal-model med okänd varians har en conjugate prior som är Inv-scale-Chi2.

5 Example code (Lab 3 and Lab 4)

5.1 R code for simulating from a GP

```
# User input
nSim <- 10
sigmaF <- 0.1
l <- 2

#install.packages("mvtnorm")
library("mvtnorm")

# Setting up the kernel
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)
}

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

SimGP <- function(m = 0,K,x,nSim,...){
  # Simulates nSim realizations (function) form a Gaussian process with mean m(x) and covariance K(x,x')
  # over a grid of inputs (x)
  n <- length(x)
  if (is.numeric(m)) meanVector <- rep(0,n) else meanVector <- m(x)
  covMat <- K(x,x,...)
  f <- rmvnorm(n, mean = meanVector, sigma = covMat)
  return(f)
}

xGrid <- seq(-5,5,length=20)
fSim <- SimGP(m=MeanFunc, K=SquaredExpKernel, x=xGrid, nSim, sigmaF, l)

plot(xGrid, fSim[1,], type="l", ylim = c(-3,3))
for (i in 2:nSim) {
  lines(xGrid, fSim[i,], type="l")
}
lines(xGrid,MeanFunc(xGrid), col = "red", lwd = 3)

# 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")
  for (i in 2:nSim) {
    lines(xGrid, fSim[i,], type="l")
  }
  lines(xGrid, MeanFunc(xGrid), col = "red", lwd = 3)
  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")
)

```

5.2 Quick demo of the R package kernlab

```

#####
## Kernlab demo for Gaussian processes regression and classification
## Author: Mattias Villani, Linköping University. http://mattiasvillani.com
#####

#####
### Prelims: Setting path and installing/loading packages #####
#####
setwd('/Users/mv/Dropbox/Teaching/AdvML/GaussianProcess/Code')
#install.packages('kernlab')
#install.packages("AtmRay") # To make 2D grid like in Matlab's meshgrid
library(kernlab)
library(AtmRay)

#####
### Messin' around with kernels #####
#####
# This is just to test how one evaluates a kernel function
# and how one computes the covariance matrix from a kernel function
X <- matrix(rnorm(12), 4, 3)
Xstar <- matrix(rnorm(15), 5, 3)
ell <- 1
SEkernel <- rbfdot(sigma = 1/(2*ell^2)) # Note how I reparametrize the rbfdo (which is the SE kernel) i
SEkernel(1,2) # Just a test - evaluating the kernel in the points x=1 and x'=2
# Computing the whole covariance matrix K from the kernel. Just a test.
K <- kernelMatrix(kernel = SEkernel, x = X, y = Xstar) # So this is K(X,Xstar)

# Own implementation of Matern with nu = 3/2 (See RW book equation 4.17)
# Note that a call of the form kernelFunc <- Matern32(sigmaF = 1, ell = 0.1) returns a kernel FUNCTION.
# You can now evaluate the kernel at inputs: kernelFunc(x = 3, y = 4).
# Note also that class(kernelFunc) is of class "kernel", which is a class defined by kernlab.

```

```

Matern32 <- function(sigmaf = 1, ell = 1)
{
  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 <- kernelMatrix(kernel = MaternFunc, x = X, y = Xstar) # So this is K(X,Xstar)

#####
###      Regression on the LIDAR data      ###
#####
lidarData <- read.table('https://raw.githubusercontent.com/STIMALiU/AdvMLCourse/master/GaussianProcess/
                        header = T)
LogRatio <- lidarData$LogRatio
Distance <- lidarData$Distance

# Estimating the noise variance from a third degree polynomial fit
polyFit <- lm(LogRatio ~ Distance + I(Distance^2) + I(Distance^3) )
sigmaNoise = sd(polyFit$residuals)

plot(Distance,LogRatio)

# Fit the GP with built in Square exponential kernel (called rbfdot in kernlab)
ell <- 2
GPfit <- gausspr(Distance, LogRatio, kernel = rbfdot, kpar = list(sigma = 1/(2*ell^2)), var = sigmaNoise)
meanPred <- predict(GPfit, Distance) # Predicting the training data. To plot the fit.
lines(Distance, meanPred, col="blue", lwd = 2)

# Fit the GP with home made Matern
sigmaf <- 1
ell <- 2
# GPfit <- gausspr(Distance, LogRatio, kernel = Matern32(ell=1)) # NOTE: this also works and is the same
GPfit <- gausspr(Distance, LogRatio, kernel = Matern32, kpar = list(sigmaf = sigmaf, ell=ell), var = sigmaNoise)
meanPred <- predict(GPfit, Distance)
lines(Distance, meanPred, col="purple", lwd = 2)

# Trying another length scale
sigmaf <- 1
ell <- 1
GPfit <- gausspr(Distance, LogRatio, kernel = Matern32, kpar = list(sigmaf = sigmaf, ell=ell), var = sigmaNoise)
meanPred <- predict(GPfit, Distance)

```

```

lines(Distance, meanPred, col="green", lwd = 2)

# And now with a different sigmaf
sigmaf <- 0.1
ell <- 2
GPfit <- gausspr(Distance, LogRatio, kernel = Matern32, kpar = list(sigmaf = sigmaf, ell=ell), var = si
meanPred <- predict(GPfit, Distance)
lines(Distance, meanPred, col="black", lwd = 2)

#####
####      Classification on Iris data      ####
#####

data(iris)
GPfitIris <- gausspr(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width, data=iris)
GPfitIris

# predict on the training set
predict(GPfitIris, iris[,1:4])
table(predict(GPfitIris, iris[,1:4]), iris[,5]) # confusion matrix

# Now using only Sepal.Length and Sepal.Width to classify
GPfitIris <- gausspr(Species ~ Sepal.Length + Sepal.Width, data=iris)
GPfitIris
# predict on the training set
predict(GPfitIris, iris[,1:2])
table(predict(GPfitIris, iris[,1:2]), iris[,5]) # confusion matrix

# Now using only Petal.Length + Petal.Width to classify
GPfitIris <- gausspr(Species ~ Petal.Length + Petal.Width, data=iris)
GPfitIris
# predict on the training set
predict(GPfitIris, iris[,3:4])
table(predict(GPfitIris, iris[,3:4]), iris[,5]) # confusion matrix

# class probabilities
probPreds <- predict(GPfitIris, iris[,3:4], type="probabilities")
x1 <- seq(min(iris[,3]), max(iris[,3]), length=100)
x2 <- seq(min(iris[,4]), max(iris[,4]), length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(iris)[3:4]
probPreds <- predict(GPfitIris, gridPoints, type="probabilities")

# Plotting for Prob(setosa)
contour(x1,x2,matrix(probPreds[,1],100), 20, xlab = "Petal.Length", ylab = "Petal.Width", main = 'Prob(
points(iris[iris[,5]=='setosa',3],iris[iris[,5]=='setosa',4],col="red")
points(iris[iris[,5]=='virginica',3],iris[iris[,5]=='virginica',4],col="blue")
points(iris[iris[,5]=='versicolor',3],iris[iris[,5]=='versicolor',4],col="green")

```



```

# Plotting for Prob(Versicolor)
contour(x1,x2,matrix(probPreds[,2],100), 20, xlab = "Petal.Length", ylab = "Petal.Width", main = 'Prob(
points(iris[iris[,5]=='setosa',3],iris[iris[,5]=='setosa',4],col="red")
points(iris[iris[,5]=='virginica',3],iris[iris[,5]=='virginica',4],col="blue")
points(iris[iris[,5]=='versicolor',3],iris[iris[,5]=='versicolor',4],col="green")

# Plotting for Prob(virginica)
contour(x1,x2,matrix(probPreds[,3],100), 20, xlab = "Petal.Length", ylab = "Petal.Width", main = 'Prob(
points(iris[iris[,5]=='setosa',3],iris[iris[,5]=='setosa',4],col="red")
points(iris[iris[,5]=='virginica',3],iris[iris[,5]=='virginica',4],col="blue")
points(iris[iris[,5]=='versicolor',3],iris[iris[,5]=='versicolor',4],col="green")

# Plotting the decision boundaries
meanPred <- matrix(max.col(probPreds),100)
plot(gridPoints, pch=".", cex=3, col=ifelse(meanPred==1, "red", ifelse(meanPred==2, "green", "blue")))
points(iris[iris[,5]=='setosa',3],iris[iris[,5]=='setosa',4],col="red", cex=10, pch=".")
points(iris[iris[,5]=='virginica',3],iris[iris[,5]=='virginica',4],col="blue", cex=10, pch=".")
points(iris[iris[,5]=='versicolor',3],iris[iris[,5]=='versicolor',4],col="green", cex=10, pch=".")

```

5.3 UC model demo on Nile flow data

```

# Unobserved components model
# My own spin of the analysis in the vignette to the rucm package in R
# Mattias Villani, http://mattiasvillani.com

#install.packages('rucm')
library(rucm)

# Plotting the data
# The Nile dataset in is the datasets package
timeNile <- seq(1871, 1970)
nTraining <- length(timeNile) # Total number of data points
nPreds <- 20
par(cex.axis = 0.8, cex.main = 0.8, cex.lab = 0.8)
plot(Nile, ylab = "Flow of Nile", lwd = 2, xlab = 'year',
     xlim = c(1,nTraining+nPreds), main = "Modeling the Nile flow")
lines(seq(1,nTraining), Nile, lwd = 2)

#### Fitting a local level model ###
#  $y(t) = \mu(t) + \epsilon(t)$  [sigma2_eps is the Irregular_Variance in rucm]
#  $\mu(t) = \mu(t-1) + \nu(t)$  [sigma2_nu is the Level_Variance in rucm]
localLevelModel <- ucm(formula = Nile~0, data = Nile, level = TRUE)
localLevelModel
lines(seq(1,nTraining), localLevelModel$s.level, col = "red", lwd = 2)

#### Fitting a local trend model ###
#  $y(t) = \mu(t) + \epsilon(t)$  [sigma2_eps is the Irregular_Variance in rucm]
#  $\mu(t) = \mu(t-1) + \beta_t + \nu(t)$  [sigma2_nu is the Level_Variance in rucm]
#  $\beta(t) = \beta(t-1) + \eta(t)$  [sigma2_eta is the Slope_Variance in rucm]

```

```

localTrendModel <- ucm(formula = Nile~0, data = Nile, level = TRUE, slope = TRUE)

# Adding the fit from the local trend model
#plot(Nile, ylab = "Flow of Nile", lwd = 2, xlab = 'year')
lines(seq(1,nTraining), localTrendModel$s.level + localTrendModel$s.slope, col = "blue", lwd = 2, xlab = "year")

# Make predictions from local level model
predsLocalLevel <- predict(localLevelModel$model, n.ahead = nPreds)
predsLocalTrend <- predict(localTrendModel$model, n.ahead = nPreds)

lines(seq(nTraining + 1,nTraining + nPreds), predsLocalLevel,
      type = "l", lwd = 2, col = "red", lty = 2)
lines(seq(nTraining + 1,nTraining + nPreds), predsLocalTrend,
      type = "l", lwd = 2, col = "blue", lty = 2)
legend("topright", legend = c("Data","Fit local level", "Fit local trend",
                             "Predictions local level","Predictions local trend"),
      col = c("black","red","blue","red","blue"), lty = c(1,1,1,2,2), lwd = 2, cex = 0.45)

# Plotting the level and trend components separately.
plot(timeNile, localTrendModel$s.level, type = "l", col = "red", lwd = 2,
     main = "Inferred level", xlab = 'year', ylab = "")
plot(timeNile, localTrendModel$s.slope, type = "l", col = "red", lwd = 2,
     main = "Inferred slope", xlab = 'year', ylab = "")

```

5.4 DLM package demo on Nile flow data

```

# Demo of the dlm package using the nile data
# Author: Mattias Villani, http://mattiasvillani.com

#install.packages("dlm")
library(dlm)

# Plotting the data
# The Nile dataset in is the datasets package
timeNile <- seq(1871, 1970)
nTraining <- length(timeNile) # Total number of data points
nPreds <- 20
par(cex.axis = 0.8, cex.main = 0.8, cex.lab = 0.8)
plot(Nile, ylab = "Flow of Nile", lwd = 2, xlab = 'year',
     xlim = c(1,nTraining+nPreds), main = "Modeling the Nile flow")
lines(seq(1,nTraining), Nile, lwd = 2)

# Set up the local trend model
# the state is  $\theta_t = (\mu_t, \beta_t)'$ 
m0 = c(0,0) # Prior mean for the first state  $\theta_0$ 
C0 = 100*diag(2) # Prior covariance for the first state  $\theta_0$ 
FF = matrix(c(1,0),1,2) # The mapping from the states to the measurements.
# Note FF must be a matrix object even if it is a vector ...
GG = matrix(c(1,1,0,1), 2, 2, byrow = T)
V = 10

```

```

W = 0.01*diag(2)

dlmLocalTrend <- dlm(m0 = m0, CO = CO, FF = FF, GG = GG, V = V, W = W)

# Kalman filtering
dlmKFfilt <- dlmFilter(y = Nile, dlmLocalTrend)

# Plotting the sum of the filtered states
lines(seq(1,nTraining), dlmKFfilt$a[,1] + dlmKFfilt$a[,2], col = "red", lwd = 2)

# Plotting the predicted response, y
lines(seq(1,nTraining), dlmKFfilt$f, col = "green", lwd = 2)

# State smoothing
dlmSmoothed <- dlmSmooth(y = Nile, dlmLocalTrend)

# Plotting the data the sum of the filtered and smoothed series for the two states.
par(cex.axis = 0.8, cex.main = 0.8, cex.lab = 0.8)
plot(Nile, ylab = "Flow of Nile", lwd = 2, xlab = 'year',
      xlim = c(1,nTraining+nPreds), main = "Modeling the Nile flow")
lines(seq(1,nTraining), Nile, lwd = 2)

# Plotting the sum of the filtered states
lines(seq(1,nTraining), dlmKFfilt$a[,1] + dlmKFfilt$a[,2], col = "red", lwd = 2)

# Plotting the sum of the smoothed states. Note that dlmSmoothed$s[1,] is theta at time t=0
lines(seq(1,nTraining), dlmSmoothed$s[2:(nTraining+1),1] + dlmSmoothed$s[2:(nTraining+1),2], col = "blue", lwd = 2)

# Parameter estimation

# First create so called build function that takes the parameters that you want to estimate
# as input and returns a list (or a dlm object) with the state space matrices as output
buildLocalTrend <- function(x){

  V = exp(x[1]) # NOTE THE EXPONENTIAL. x must be unrestricted parameters, otherwise optim will go bad
  W = diag(exp(x[2:3]),2,2)

  return(dlm(
    m0 = rep(0,2),
    CO = 100*diag(2),
    FF = matrix(c(1,0),1,2),
    GG = matrix(c(1,1,0,1), 2, 2, byrow = T),
    V = V,
    W = W))
}

initVal <- c(1,1,1) # Initial values for optim on the estimated parameters
dlmLocalTrend <- buildLocalTrend(initVal) # Just to see that the build function works
MLEs <- dlmMLE(Nile, parm = initVal, build = buildLocalTrend)
dlmWithMLEs <- buildLocalTrend(MLEs$par)

```

```

# Let's plot the filtered and smoothed states using the MLEs

# First plot the Nile data
par(cex.axis = 0.8, cex.main = 0.8, cex.lab = 0.8)
plot(Nile, ylab = "Flow of Nile", lwd = 2, xlab = 'year',
      xlim = c(1,nTraining+nPreds), main = "Modeling the Nile flow")
lines(seq(1,nTraining), Nile, lwd = 2)

# Filtering
dlmKFfilt <- dlmFilter(y = Nile, dlmWithMLEs)

# Plotting the sum of the filtered states
lines(seq(1,nTraining), dlmKFfilt$a[,1] + dlmKFfilt$a[,2], col = "red", lwd = 2)

# State smoothing
dlmSmoothed <- dlmSmooth(y = Nile, dlmWithMLEs)

# Plotting the sum of the smoothed states. Note that dlmSmoothed$s[1,] is theta at time t=0
lines(seq(1,nTraining), dlmSmoothed$s[2:(nTraining+1),1]
      + dlmSmoothed$s[2:(nTraining+1),2], col = "blue", lwd = 2)

# Bayesian analysis
nDraws <- 1000
nStates <- 2
postDraws <- array(NA, c(nTraining, nDraws, nStates) )
for (i in 1:nDraws){
  postDraws[,i,] <- dlmBSample(dlmKFfilt)[-1,]
}

# Plotting the smoothed series (posterior mean)
# and the first 3 draws from posterior
par(cex.axis = 0.8, cex.main = 0.8, cex.lab = 0.8)
plot(Nile, ylab = "Flow of Nile", lwd = 2, xlab = 'year',
      xlim = c(1,nTraining+nPreds), main = "Modeling the Nile flow")
lines(seq(1,nTraining), Nile, lwd = 2)
lines(seq(1,nTraining), dlmSmoothed$s[2:(nTraining+1),1]
      + dlmSmoothed$s[2:(nTraining+1),2], col = "red", lwd = 2)
for (i in 1:3){
  lines(seq(1,nTraining), postDraws[,i,1] + postDraws[,i,2], col = "blue", lwd = 1)
}

# Plotting the marginal posterior of the state at time t
hist(postDraws[50,,1], 40)

```

6 Lab instructions

6.1 Lab 1

6.1.1 (1)

Show that multiple runs hill-climbing algorithm can return non-equivalent DAGs. Can we check if two DAGs are equivalent by checking if they receive the same score ? Hint: Check the function `hc` in the `bnlearn` package. Note that you can specify the initial structure as well as the number of random restarts. Use these options to answer the question. You may also want to use the functions `plot`, `arcs`, `vstructs`, `cpdag` and `all.equal`.

6.1.2 (2)

Show that increasing the equivalent sample size (a.k.a imaginary sample size) in the BDeu score decreases regularization. Explain why this happens. Hint: Run some structure learning algorithm (e.g. check the function `hc` in the `bnlearn` package) multiple times and show that it tends to end in more densely connected DAGs. Or produce a histogram of the scores of a random sample of DAGs with different imaginary sample sizes and see if they come closer or not one to another (e.g. check the functions `hist`, `random.graph`, `sapply` and `score` in the `bnlearn` and `core` packages).

6.1.3 (3)

You already know the LS algorithm for inference in BNs, which is an exact algorithm. There are also approximate algorithms for when the exact ones are too demanding computationally. Compare the answers given to some queries by exact and approximate inference algorithms. When running the approximate algorithm several times, why may we obtain different answers ? Is the approximate algorithm equally accurate when the query includes no observed nodes and when it includes many observed nodes ? Hint: For exact inference, you may need the functions `bn.fit` and `as.grain` from the `bnlearn` package, and the functions `compile`, `setFinding` and `querygrain` from the package `gRain`. For approximate inference, you may need the functions `prop.table`, `table` and `cpdist` from the `bnlearn` package.

6.1.4 (4)

There are 29281 DAGs with five nodes. Compute approximately the fraction of the 29281 DAGs that represent different independence models. How does this affect structure learning ? Hint: You do not need to produce the 29281 DAGs. Instead you can sample DAGs uniformly, convert each DAG in the sample to its essential graph (a.k.a completed partial DAG or CPDAG), and then count how many different essential graphs you have left. You can do all this with the functions `random.graph`, `cpdag`, `lapply`, `unique` and `length` in the `bnlearn` and `core` packages.

6.2 Lab 2

6.2.1 (1)

Build a HMM for the scenario described above.

6.2.2 (2)

Simulate the HMM for 100 time steps.

6.2.3 (3)

Discard the hidden states from the sample obtained above. Use the remaining observations to compute the filtered and smoothed probability distributions for each of the 100 time points. Compute also the most probable path.

6.2.4 (4)

Compute the accuracy of the filtered and smoothed probability distributions, and of the most 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.

6.2.5 (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 path, too. Why ?

6.2.6 (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`.

6.2.7 (7)

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

Gaussian Processes - Computer Lab

Deadline: See LISAM
Teacher: Mattias Villani
Grades: Pass/Fail
Submission: Via LISAM

You should use R to solve the lab since the computer exam will be in R.
Output: an individual report and a group report to be presented on the seminar.
Attach your code in LISAM as separate files.

1. **Implementing Gaussian process regression from scratch.** This first exercise will have you writing your own code for the Gaussian process regression model:

$$y = f(x) + \varepsilon \quad \varepsilon \sim N(0, \sigma_n^2) \\ f \sim GP[0, k(x, x')].$$

When it comes to the posterior distribution for f , I **strongly** suggest that you implement Algorithm 2.1 on page 19 of Rasmussen and Williams (RW) book. That algorithm uses the Cholesky decomposition (`chol()` in R) to attain numerical stability. Here is what you need to do:

- (a) Write your own code for simulating from the posterior distribution of $f(x)$ using the squared exponential kernel. The function (name it `posteriorGP`) should return vectors with the posterior mean and variance of f , both evaluated at a set of x -values (x^*). You can assume that the prior mean of f is zero for all x . The function should have the following inputs:
- i. `x` (vector of training inputs)
 - ii. `y` (vector of training targets/outputs)
 - iii. `xStar` (vector of inputs where the posterior distribution is evaluated, i.e. x^* . As in my slides).
 - iv. `hyperParam` (vector with two elements σ_f and ℓ)
 - v. `sigmaNoise` (σ_n).
- [Hint: I would write a separate function for the Kernel (see my `GaussianProcess.R` function on the course web page) which is then called from the `posteriorGP` function.]
- (b) Now let the prior hyperparameters be $\sigma_f = 1, \ell = 0.3$. Update this prior with a single observation: $(x, y) = (0.4, 0.719)$. Assume that the noise standard deviation is known to be $\sigma_n = 0.1$. Plot the posterior mean of f over the interval $x \in [-1, 1]$. Plot also 95% probability (pointwise) bands for f .

x	-1.0	-0.6	-0.2	0.4	0.8
y	0.768	-0.044	-0.940	0.719	-0.664

Table 1: Simple data set for GP regression.

- (c) Update your posterior from 1b) with another observation: $(x, y) = (-0.6, -0.044)$. Plot the posterior mean of f over the interval $x \in [-1, 1]$. Plot also 95% probability bands for f . [Hint: updating the posterior after one observation with a new observation gives the same result as updating the prior directly with the two observations. Bayes is beautiful!]
- (d) Compute the posterior distribution of f using all 5 data points in Table 1 below (note that the two previous observations are included in the table). Plot the posterior mean of f over the interval $x \in [-1, 1]$. Plot also 95% probability intervals for f .
- (e) Repeat 1d), this time with the hyperparameters $\sigma_f = 1, \ell = 1$. Compare the results.

2. **Gaussian process regression on real data using the kernlab package.** This exercise lets you explore the kernlab package on a data set of daily mean temperature in Stockholm (Tullinge) during the period January 1, 2010 - December 31, 2015. I have removed the leap year day February 29, 2012 to make your life simpler. You can read the dataset with the command:

```
read.csv('https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.csv', header=TRUE, sep=',')
```

Create the variable `time` which records the day number since the start of the dataset (i.e. `time = 1, 2, ..., 365 \cdot 6 = 2190`). Also, create the variable `day` that records the day number since the start of each year (i.e. `day = 1, 2, ..., 365, 1, 2, ..., 365`). Estimating a GP on 2190 observations can take some time on slower computers, so let's thin out the data by only using every fifth observation. This means that your time variable is now `time = 1, 6, 11, ..., 2186` and `day = 1, 6, 11, ..., 361, 1, 6, ..., 361`.

- (a) Familiarize yourself with the following functions in `kernlab`, in particular the `gausspr` and `kernelMatrix` function. Do `?gausspr` and read the input arguments and the output. Also, go through my `KernLabDemo.R` carefully; you will need to understand it. Now, define your own square exponential kernel function (with parameters ℓ (`ell`) and σ_f (`sigmaf`)), evaluate it in the point $x = 1$, $x' = 2$, and use the `kernelMatrix` function to compute the covariance matrix $K(\mathbf{x}, \mathbf{x}_*)$ for the input vectors $\mathbf{x} = (1, 3, 4)^T$ and $\mathbf{x}_* = (2, 3, 4)^T$.
- (b) Consider first the model:

$$\begin{aligned} temp &= f(time) + \varepsilon \quad \varepsilon \sim N(0, \sigma_n^2) \\ f &\sim GP[0, k(time, time')] \end{aligned}$$

Let σ_n^2 be the residual variance from a simple quadratic regression fit (using the `lm()` function in R). Estimate the above Gaussian process regression model using the squared exponential function from 2a) with $\sigma_f = 20$ and $\ell = 0.2$. Use the `predict` function to compute the posterior mean at every data point in the training datasets. Make a scatterplot of the data and superimpose the posterior mean of f as a curve (use `type="l"` in the plot function). Play around with different values on σ_f and ℓ (no need to write this in the report though).

- (c) **kernlab** can compute the posterior variance of f , but I suspect a bug in the code (I get weird results). Do your own computations for the posterior variance of f (hint: Algorithm 2.1 in RW), and plot 95% (pointwise) posterior probability bands for f . Use $\sigma_f = 20$ and $\ell = 0.2$. Superimpose those bands on the figure with the posterior mean in 2b).
- (d) Consider now the model

$$\begin{aligned} \text{temp} &= f(\text{day}) + \varepsilon \quad \varepsilon \sim N(0, \sigma_n^2) \\ f &\sim GP[0, k(\text{day}, \text{day}')] \end{aligned}$$

Estimate the model using the squared exponential function from 2a) with $\sigma_f = 20$ and $\ell = 6 \cdot 0.2 = 1.2$. (I multiplied ℓ by 6 compared to when you used **time** as input variable since **kernlab** automatically standardizes the data which makes the distance between points larger for **day** compared to **time**). Superimpose the posterior mean from this model on the fit (posterior mean) from the model with **time** using $\sigma_f = 20, \ell = 0.2$. Note that this plot should also have the **time** variable on the horizontal axis. Compare the results from using **time** to the ones with **day**. What are the pros and cons of each model?

- (e) Now implement a generalization of the periodic kernel given in my slides from Lecture 2 of the GP topic (Slide 6)

$$k(x, x') = \sigma_f^2 \exp\left(-\frac{2 \sin^2(\pi |x - x'| / d)}{\ell_1^2}\right) \times \exp\left(-\frac{1}{2} \frac{|x - x'|^2}{\ell_2^2}\right).$$

Note that we have two different length scales here, and ℓ_2 controls the correlation between the same day in different years (ℓ_2). So this kernel has four hyperparameters σ_f, ℓ_1, ℓ_2 and the period d . Estimate the GP model using the **time** variable with this kernel with hyperparameters $\sigma_f = 20, \ell_1 = 1, \ell_2 = 10$ and $d = 365/\text{sd}(\text{time})$. The reason for the rather strange period here is that **kernlab** standardized inputs to have standard deviation of 1. Compare the fit to the previous two models (with $\sigma_f = 20, \ell = 0.2$). Discuss the results.

3. Gaussian process classification using the kernlab package. Download the banknote fraud data:

```
data <- read.csv('https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud.csv',
header=FALSE, sep=',')
names(data) <- c("varWave", "skewWave", "kurtWave", "entropyWave", "fraud")
data[,5] <- as.factor(data[,5])
```

You can read about this dataset here. Choose 1000 observations as training data using the following command (i.e. use the vector **SelectTraining** to subset the training observations)

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

- (a) Use **kernlab** to fit a Gaussian process classification model for **fraud** on the training data, using **kernlab**. Use **kernlab**'s the default kernel and hyperparameters. Start with using only the first two covariates **varWave** and **skewWave** in the model. Plot contours of the prediction probabilities over a suitable grid of values for **varWave** and **skewWave**. Overlay the training data for **fraud** = 1 (as blue points) and **fraud** = 0 (as red points). You can take a lot of code for this from my **KernLabDemo.R**. Compute the confusion matrix for the classifier and its accuracy.
- (b) Using the estimated model from 3a), make predictions for the testset. Compute the accuracy.
- (c) Train a model using all four covariates. Make predictions on the test and compare the accuracy to the model with only two covariates.

Good luck!

- Mattias

State Space Models - Computer Lab

Deadline: October 17 at midnight

Teacher: Matias Quiroz

Grades: Pass/Fail

Submission: Via LISAM

You should use R to solve the lab since the computer exam will be in R.

Output: an individual report and a group report to be presented on the seminar.

Attach your code in LISAM as separate files.

1. **Implementation of the linear Gaussian state space model.** Consider the linear Gaussian state space model

$$\begin{aligned} y_t &= Cx_t + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, \Omega_\epsilon) \\ x_t &= Ax_{t-1} + Bu_t + v_t, \quad v_t \sim \mathcal{N}(0, \Omega_v) \quad (v_t \perp \epsilon_t), \quad x_0 \sim \mathcal{N}(\mu_0, \Sigma_0) \end{aligned}$$

where $y_t \in \mathbb{R}^{n_y \times 1}$ is the vector of observations at time t , $x_t \in \mathbb{R}^{n_x \times 1}$ is the unobserved state vector and $u_t \in \mathbb{R}^{n_u \times 1}$ is the control vector for x_t . The dimensions of the matrices of the system are $C \in \mathbb{R}^{n_y \times n_x}$, $\Omega_\epsilon \in \mathbb{R}^{n_y \times n_y}$, $A, \Omega_v, \mu_0, \Sigma_0 \in \mathbb{R}^{n_x \times n_x}$ and $B \in \mathbb{R}^{n_x \times n_u}$.

Your implementation will use simulated data to remind you that this is a good way to check that your code works. Moreover, real data can often give rise to numerical instabilities that have to be taken into account and this is outside the scope of this course. In this problem you will work with both a scalar measurement and state equation (i.e. $n_y = n_x = 1$), but you should write the code so that it can handle matrices. This is achieved by using the data type `matrix` in R when you define the variables (even if they are scalars!).

Check your results against those obtained using the R package `DLM`. This ensures that your code works properly, but more important it gives you a good practice for the computer exam.

- (a) **Simulate the model.** Use `set.seed(1234)` when you simulate data in this exercise. This is because (i) reproducibility facilitates the correction of the assignment and (ii) numerical instabilities do not appear (except the ones caused by bugs!) when implementing the filtering and smoothing recursions.

Set $x_0 = 0$ and simulate $T = 100$ observations from the following model

$$\begin{aligned} y_t &= 0.93x_t + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, \Omega_\epsilon = 0.5) \\ x_t &= x_{t-1} + v_t, \quad v_t \sim \mathcal{N}(0, \Omega_v = 0.1) \quad (v_t \perp \epsilon_t), \quad x_0 \sim \mathcal{N}(\mu_0 = 0, \Sigma_0 = 10). \end{aligned} \tag{1}$$

Note that this model does not have any control variables, but you can still implement the code in the general matrix form and define B and u_t to be a matrix/vector of only zeros. With `set.seed(1234)` the simulated state and observations should look as in Figure 1. The data is also found in `simulated_data.Rda`.

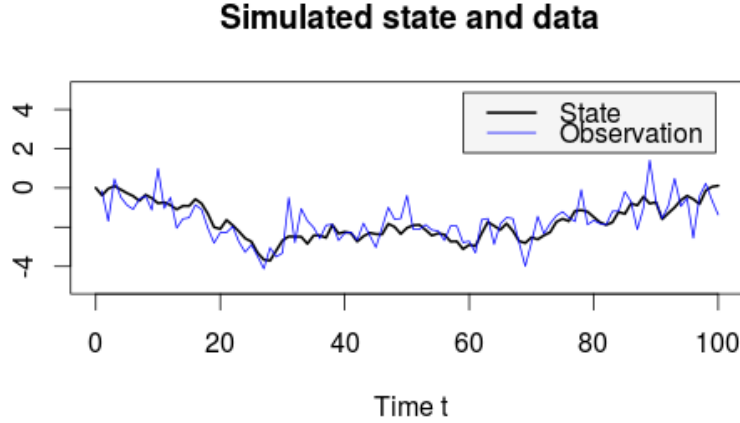


Figure 1: Simulated state and data with seed '1234'.

- (b) **Filtering: Sequential state inference.** We will learn x_t sequentially for the model in Equation (1), using only data up to t , i.e. $y_{1:t}$. In other words, we will compute the filtering distribution $p(x_t|y_{1:t})$ for $t = 1, \dots, T$. Because of the linear relationship between y_t and x_t and the normality of ϵ_t and v_t , $p(x_t|y_{1:t})$ is Gaussian, where the mean and variance are obtained from the Kalman filter recursions.

Implement the Kalman filter. For each $t = 1, \dots, T$, plot the point estimate $E[x_t|y_{1:t}]$ and a 0.95 probability interval for $x_t|y_{1:t}$. In the same graph, plot the true state x_t (that you simulated in (a)) and y_t . Compare your filtering distribution to that obtained by the DLM package using `dlmFilter`. Notice that `dlmFilter` does not return $\Sigma_t = V[x_t|y_{1:t}]$ but its Singular Value Decomposition (SVD) instead. You can obtain Σ_t using the function `dlmSvd2var`.

- (c) **Prediction of state and data by simulation.** We first note that both the predictive distribution for the state and the data are analytically available for the linear Gaussian state space model. We will here approach it by simulation to get a good intuition about the uncertainties involved when simulating the data and the states.

Suppose we want to predict future values of the process y_{T+k} conditional on $y_{1:T}$. It is easy to show that (convince yourself, but no need to include in the report)

$$\begin{aligned}
 p(y_{T+k}|y_{1:T}) &= \int p(y_{T+k}, x_{T+k}|y_{1:T}) dx_{T+k} \\
 &= \int p(y_{T+k}|x_{T+k}, y_{1:T}) p(x_{T+k}|y_{1:T}) dx_{T+k} \\
 &= \int p(y_{T+k}|x_{T+k}) p(x_{T+k}|y_{1:T}) dx_{T+k}, \tag{2}
 \end{aligned}$$

where the last equality follows from conditional independence in a general state space model (inspect the graphical model!). A fact that was often stressed during the course in Bayesian learning is that Bayes is all about averaging functions (here $p(y_{T+k}|x_{T+k})$) with respect to posterior uncertainty (here represented by $p(x_{T+k}|y_{1:T})$): notice that Equation (2) is nothing but

$$E_{x_{T+k}|y_{1:T}}[p(y_{T+k}|x_{T+k})].$$

To simulate from $p(y_{T+k}|y_{1:T})$ we can sample

$$\begin{aligned} x_{T+k}^{(i)} &\sim p(x_{T+k}|y_{1:T}) \\ y_{T+k}^{(i)} &\sim p(y_{T+k}|x_{T+k}^{(i)}). \end{aligned}$$

We note that the first step requires to simulate the state at $t = T + k$ conditional on $y_{1:T}$. It is easy to show that

$$p(x_{T+k}|y_{1:T}) = \int p(x_{T+k}|x_{T+k-1})p(x_{T+k-1}|y_{1:T})dx_{T+k-1}.$$

Your task is to use simulation and compute 0.95 probability intervals for $x_{T+k}|y_{1:T}$ and $y_{T+k}|y_{1:T}$ for $k = 1, 2, 3, 4, 5$. **Hint:** Start with $k = 1$ and note that $p(x_{T+k-1}|y_{1:T}) = p(x_T|y_{1:T})$ which you can obtain from the Kalman filter. You can then easily simulate $x_{T+k}|y_{1:T}$ recursively, starting from $k = 1$ and, for each k , you can easily obtain a sample from $y_{T+k}|y_{1:T}$ as explained above. This gives you one realization of $y_{T+1}|y_{1:T}, \dots, y_{T+5}|y_{1:T}$ (5 future observations) which you then repeat for a desired number of samples.

Plot the estimated probability intervals in a time-extended plot that also includes the true state, the filtered state (with probability intervals) and the observations, see Figure 2 for an example. The function `d1mForecast` can be used to check your results.

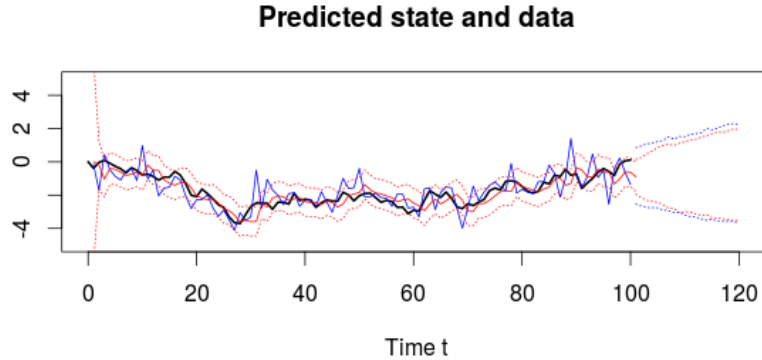


Figure 2: Example of the requested plot (with $k = 20$). True state (black), estimated state (red, filtered for $1 \leq t \leq 100$ and predicted for $t \geq 101$, dotted denotes probability intervals), observations (blue) and probability intervals for predicted distribution $p(y_{T+k}|y_{1:T})$ (dotted blue).

- (d) **State and model parameter inference.** Up to now we have assumed that all model parameters are known and the task has been to estimate the state vector. We now also let the model parameters be unknown and estimate them together with the states from data. For simplicity we assume known variances in both the observation and state equation, but we remark that it is in principle straightforward to estimate them. Thus our model is

$$\begin{aligned} y_t &= \theta x_t + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, \Omega_\epsilon = 0.5) \\ x_t &= x_{t-1} + v_t, \quad v_t \sim \mathcal{N}(0, \Omega_v = 0.1) \quad (v_t \perp \epsilon_t), \quad x_0 \sim \mathcal{N}(\mu_0 = 0, \Sigma_0 = 10), \end{aligned} \quad (3)$$

and since we are Bayesians we also need a prior for θ which we turn to in a short while. The posterior is now $p(\theta, x_{1:T}|y_{1:T})$ and we can use Gibbs sampling to explore it by simulation as follows. Initialize $\theta^{(0)}$ and, for $i = 1, \dots, N$, repeat

- (i) $x_{1:T}^{(i)} \sim p(x_{1:T}|\theta^{(i-1)}, y_{1:T})$
- (ii) $\theta^{(i)} \sim p(\theta|x_{1:T}^{(i)}, y_{1:T})$.

Step (i) can be solved by Forward Filtering and Backward Sampling (FFBS), which in turn requires the Kalman Filter that you implemented earlier. The backward sampling use the mean and variance of $p(x_{t+1}|y_{1:t})$ ($\bar{\mu}_{t+1}$ and $\bar{\Sigma}_{t+1}$) that you already computed with the Kalman Filter and can thus be reused for computational efficiency. Note that you have to run the Kalman Filter (and the backward sampling) for every new θ ! Step (ii) is easy to sample by noting that, once we condition on the simulated states from Step (i), the model for y_t in Equation (3) is an ordinary regression model with unknown slope but known variance. A normal prior for θ is conjugate for this model, so that the full conditional $p(\theta|x_{1:T}, y_{1:T})$ is also normal (with updated mean and variance).

Implement a Gibbs sampler that samples from $p(\theta, x_{1:T}|y_{1:T})$ following the instructions above. Take the vague prior $p(\theta) = \mathcal{N}(0, 100)$. Implement your own FFBS and, to check that it is correct, you can compare to `d1mBSample`. **Warning:** For some starting values of θ you can run into numerical problems as the Kalman filter can become numerically which our implementation, as already noticed, does not take into account. Change starting values and try again. I found that this problem can arise with a negative starting value or a very small one. Plot the posterior $p(\theta|y_{1:T})$ using e.g. a histogram or a kernel density estimate (don't forget to discard burn-in). Can you estimate θ accurately (recall that the true value is 0.93)?

2. **Regression with time-varying parameters.** The Capital Asset Pricing Model (CAPM) is useful for determining the risk of an asset. Let $y_t = r_t - r_t^f$, where r_t is the return of the asset of interest at time t and r_t^f is the return of a risk-free asset (e.g. a bond). Likewise, $z_t = r_t^M - r_t^f$ is the excess return at time t but for the market portfolio with return r_t^M . The CAPM model assumes that

$$y_t = \alpha + \beta z_t + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, \sigma_\epsilon^2).$$

The parameter α captures the excess returns not explained by z_t . The main parameter β measures the sensitivity of the asset w.r.t changes in the market: $\beta > 1$ (< 1) amplifies (dampens) changes in markets return and is considered an aggressive (conservative) investment.

The dynamic CAPM models both the "base"-return and sensitivity as time-varying. We here consider the following dynamic model

$$\begin{aligned} y_t &= \alpha_t + z_t \beta_t + \epsilon_t, & \epsilon_t &\sim \mathcal{N}(0, \sigma_\epsilon^2) \\ \alpha_t &= \alpha_{t-1} + v_t^{(1)}, & v_t^{(1)} &\sim \mathcal{N}(0, \sigma_{v^{(1)}}^2) & p(\alpha_0) &= \mathcal{N}(0, 100) \\ \beta_t &= \beta_{t-1} + v_t^{(2)}, & v_t^{(2)} &\sim \mathcal{N}(0, \sigma_{v^{(2)}}^2) & p(\beta_0) &= \mathcal{N}(1, 0.5) \text{ and } \epsilon_t \perp\!\!\!\perp v_t^{(1)} \perp\!\!\!\perp v_t^{(2)}. \end{aligned}$$

Note that we have a vector of states $x_t = (\alpha_t, \beta_t)$ and, in contrast to the previous problem, the system matrix C of the observation equation is time-dependent (here $C = z_t$).

You will use the data set `CAPM_data.Rda` to estimate the model. The variables of interest in the data set are $r_t^f = \text{RKFREE}$, $r_t^M = \text{MARKET}$ and $r_t = \text{IBM}$. The frequency of the data is monthly, from January 1978 to December 1987, a total of $T = 120$ data points. Use the `DLM` package in *R* to solve the filtering, smoothing and backward sampling problems in this exercise (your code from above cannot have a time-varying system matrix). In the `DLM` package, a time-varying system matrix z_t can be incorporated using the `JFF` input to the `d1m` function, see the

help. As an alternative, the function `d1mModReg` allows you to specify a regression model with time-varying parameters directly. However, **do not** use `d1mModReg`. Learning how to use the `d1m` function with time-varying system matrices is a good exercise as it allows you to treat far more general models.

- (a) **Estimating the variance components by MLE.** Find the Maximum Likelihood Estimate (MLE) of $\sigma_\epsilon^2, \sigma_{v(1)}^2, \sigma_{v(2)}^2$ (using the DLM package). For problems (b)-(c) you will use the MLE values obtain here.
- (b) **Filtering and smoothing.** Compute the filtering and smoothing distribution, respectively. In both cases, plot the result for both α and β (on separate graphs), together with 0.95 probability intervals.
- (c) **Average sensitivity pre and post 1982 - a retrospective analysis.** Do a retrospective analysis of the average sensitivity before and after year 1982 (year 1982 starts at $t = 49$). Did the average sensitivity increase after 1982? Motivate your answer. **Hints:** (i) recall that retrospective conditions on all available data and (ii) think like a Bayesian!
- (d) **Bayesian inference for the variance components.** Describe in detail how you would do Bayesian inference on the variance components. No need to implement it, but clearly motivate your answer.

Good luck!

Graphical Models (PRML, Ch 8)

- Simple way to visualize the structure of a probabilistic model → Design and motivate new models
- Understanding of a model's properties, conditional independencies for example
- Complex computations expressed in terms of graphical manipulation.

- Each node represents a random variable
- Lines express probabilistic relationships between the variables
- Bayesian networks, also called DAG's, are graphs where all links have a particular directionality indicated by arrows.
- DAGs are useful for expressing causal relationships
- $P(a, b, c) = P(c|a, b) P(b|a) P(a)$ gives the following DAG



$p(c|a, b)$ = Two incoming lines for c

- Since there is a line from a to b , a is the parent of b and b is the child of a .
- A node can represent a single variable as well as a set of variables.
- There must be no directed cycles (follow lines and end up at start node not allowed to be possible).
- A graphical model captures the causal process and is therefore often called generative model
- If there are K states and M variables, a fully connected graph (the completely general distribution) will have $K^M - 1$ parameters
- No links in the graph gives the product of the marginals and $M(K-1)$ parameters
- A graph over discrete variables turns into a bayesian model by setting dirichlet priors for the parameters.

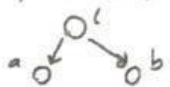
- Notation for conditional independence: $a \perp\!\!\!\perp b \mid c$

a is conditionally independent of b given c

- Is equivalent to $p(a|b, c) = p(a|c)$

- General framework for achieving a graphical model that shows conditional independencies is called d-separation

- $p(a, b, c) = p(a|b, c) p(b|c) = p(a|c) p(b|c)$

gives the graph  The path makes the nodes a and b dependent

given the empty set. Given an observed value of c are they independent (conditionally)

- The conditioned node then "blocks" the path.

- Another example: $p(a, b, c) = p(a) p(b) p(c|a, b)$

In a case where none of the variables are observed $\Rightarrow p(a) p(b)$

$\Rightarrow a \perp\!\!\!\perp b \mid \emptyset$. However, if conditioned on the observed value of c they are dependent: $a \not\perp\!\!\!\perp b \mid c$.

- If all paths are blocked, then A is said to be d-separated from B by C .

- A Markov network, or an undirected graphical model, is equivalent to a DAG except that all links are undirected.

- If path from A to B always passes through C , A and B are conditionally independent

- Same general idea as d-separation, but simpler to check for undirected graphs.

- The Markov blanket of a node consists of the set of neighbouring nodes

- A clique is a set of nodes that are fully connected.

- A maximal clique is a clique such that it is not possible to include any other nodes from the graph.

- The Moral graph is the conversion from an directed graph to an undirected.

- If one parent - just remove arrow.

- If more than one parents, the node and its parents must all belong to a single clique.
- The process of moralization adds the fewest needed extra lines and retains the maximum number of independence properties.

KOSKI & Noble

- Fork Connection



$$X \perp Y \mid Z$$

- Chain Connection



$$X \perp Y \mid Z$$

- Collider Connection



$$X \perp Y$$

$$X \not\perp Y \mid Z$$

- All connections between nodes in a DAG are of one of these types.
- The Markov blanket of a node in a DAG is the parents, the children and the parents of the children.
- Aalborg algorithm (by L.S)

1. DAG is moralized

- The Moral graph is the graph where each variable-parent set in the original DAG is a clique.

2. Moral graph is triangulated

3. Cliques are organized into a junction tree

- Can only be done if decomposable, which it is if triangulated
- Junction tree is a tree with all the cliques
- The nodes present in two adjacent cliques are called separators.

- Evidence is the information that certain nodes take specific values.

- Parameter learning: Using proportions from observed data, use dirichlet prior and update with bayes rule.

Hidden Markov Models (PRML, Ch. 13)

• Sequential data and stationary distributions. Stationarity means that underlying distribution for the sequence remains the same over the whole time period.

• HMM useful when there is a need of relaxing the iid assumption.

• In a simple HMM is the next observation independent of all earlier observations except the preceding one.

• Higher-order Markov Chains allow more than the preceding observation to have impact on the prediction.

• In a second-order Markov chain is information from the two preceding observations used.

• Higher-order models increases the flexibility but also the complexity in terms of number of parameters.

• If the variables are discrete, the number of parameters in the model will be $K^{M+1}(K-1)$ (M =order, K =states)

• To get past this problem is a latent variable, z_n , introduced for each observation.

• There is then always a path open between two observed values, so all ^{previous} observations can be used for making predictions (unless z is observed, which it isn't)

• If latent variables are discrete is a HMM obtained.

• The observed values can be discrete or continuous.

• If latent variables are discrete \rightarrow state space model

• The value of z_n depends on z_{n-1} and the probability for each state is specified in the transition matrix.

• The probabilities for x_n is given by the emission matrix

• When sampling: init value for $z_1 \Rightarrow$ Sample $x_1 \Rightarrow$ sample z_2 using transition matrix and z_1 .

• Forward-Backward algorithm

- α uses data up to time n
- β uses data from $n+1$ to N
- α works forward and β backwards

- If latent variables do have some meaningful interpretation is often the most probable sequence of hidden states for a given observation sequence of interest to find.

- The problem of finding the most probable sequence of latent states is not the same as that of finding the set of states that individually are the most probable.

- The latter might give sequences with very low, or zero, probability.

- The Viterbi algorithm is used for finding the most probable sequence/path.

- The algorithm only keeps the path that at each time step is the most probable. Then, at next step, selects the state which gives the, so far, most probable path and keeps only that sequence.

Ghahramani

- Markov property: Given the state S_{t-1} is S_t independent of all states prior to $t-1$

- Observations are generated by some process whose state is hidden from the observer.

- The properties above defines the HMM.

- HMM is a dynamic Bayesian network, which is a Bayesian network that works for modelling time series data.
