Lab 1 - AML Group Report

Alessia De Biase Alejandro Garcia, Anna-Katharina Fürgut, Erika Anderskär

September 20th 2018

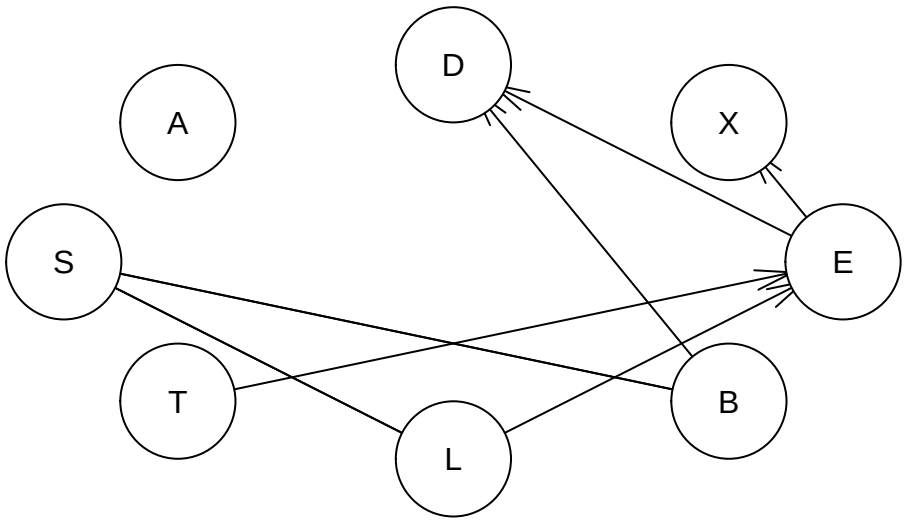
Questions and Answers

1. *Show that multiple runs of the hill-climbing algorithm can return non-equivalent Bayesian network (BN) structures. Explain why this happens.*

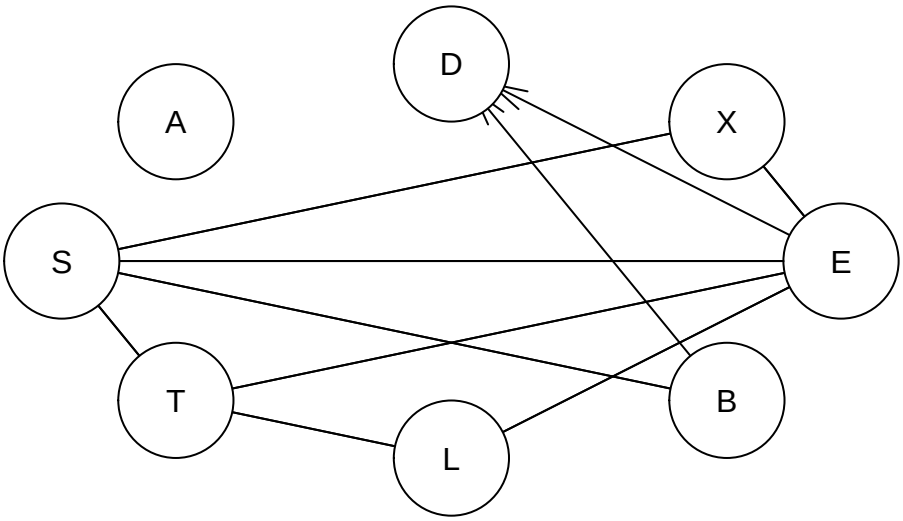
In this assignment we will use the dataset `asia` from the `bnlearn` package and we will run the hill-climbing algorithm on this dataset to obtain many Bayesian network structures. To do this we will use the `hc` function from the same package specifying different values of parameters:

- initial structure (`start`), if not specified it's an empty DAG;
- number of random restarts (`restart`) which reduces the probability of getting stuck in local maximum;
- the score (`score`);
- the equivalent sample size in the BDeu score (`iss`)).

```
data("asia")

starting=random.graph(names(asia)) #random DAG as starting one

trial_1=hc(asia, restart = 11)
plot(cpdag(trial_1))
```



```
trial_3=hc(asia, start=starting)
plot(cpdag(trial_3))
```




```
trial_2=hc(asia, score="bic")
trial_4=hc(asia, score="aic")
trial_5=hc(asia, score="loglik")

#true Asia BN
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
```

The DAGs trial1 and trial3 for example are non equivalent, because they represent different independence models (see their cpdags).

The *loglik* score makes many more connections between nodes because it's satisfied with weak condition relations; *BIC* score is the strongest score. The last plot shows the true DAG.

The first and the third have:

```
all.equal(trial_1,trial_2)
```

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

The first and the second have:

```
all.equal(trial_1,trial_3)
```

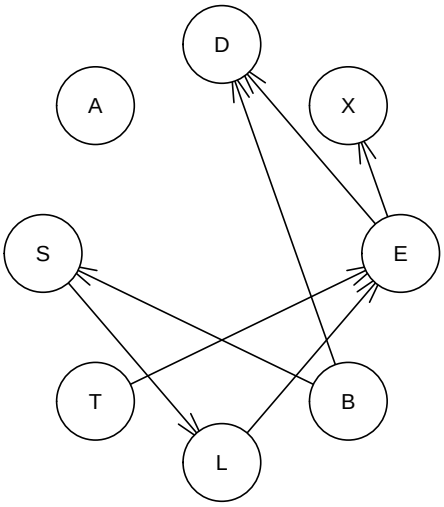
```
## [1] "Different number of directed/undirected arcs"
```

The second and the third have:

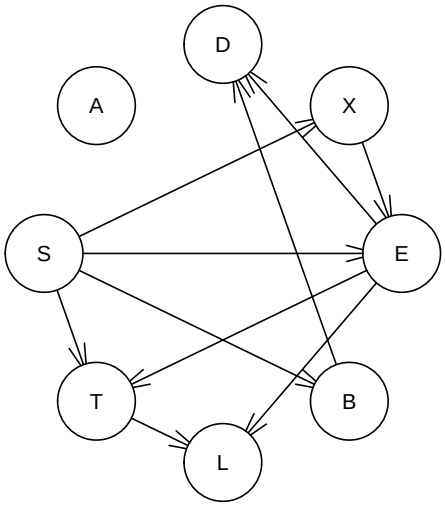
```
all.equal(trial_2,trial_3)
```

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

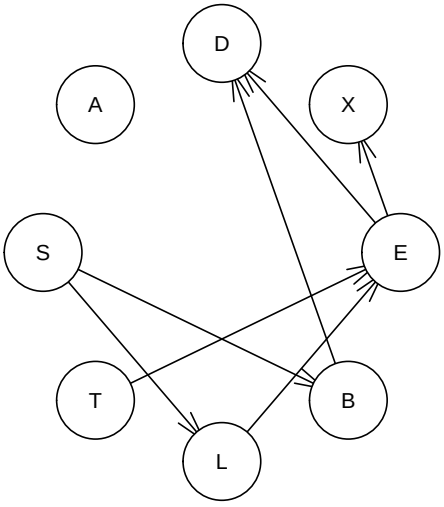
DAG with restart=11



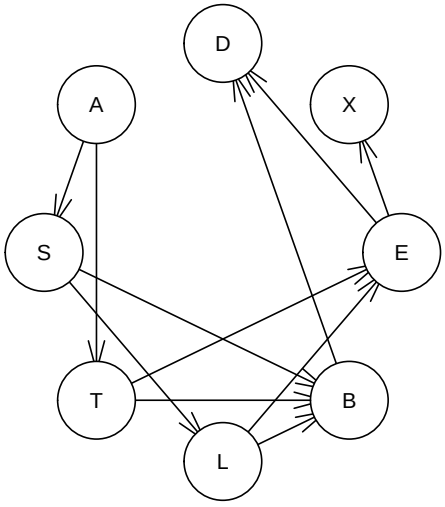
DAG with random initial structure



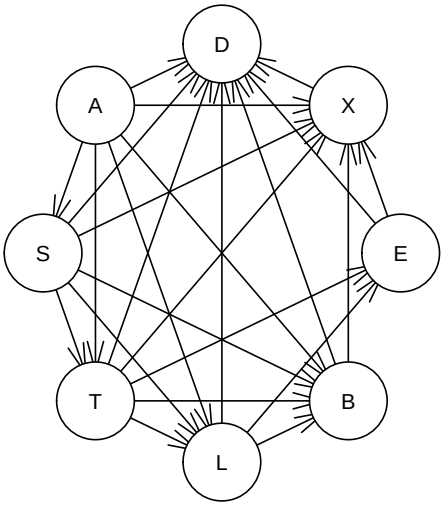
DAG with score=BIC



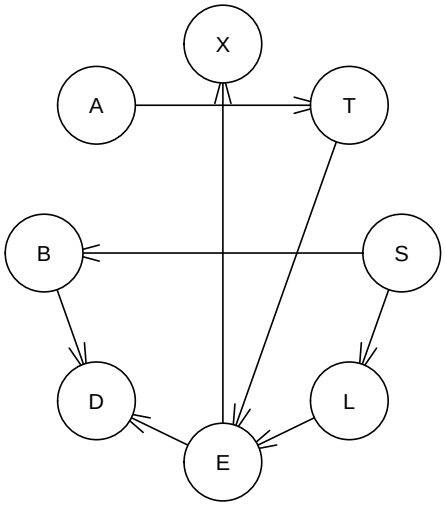
DAG with score=AIC



DAG with score=loglik



True DAG

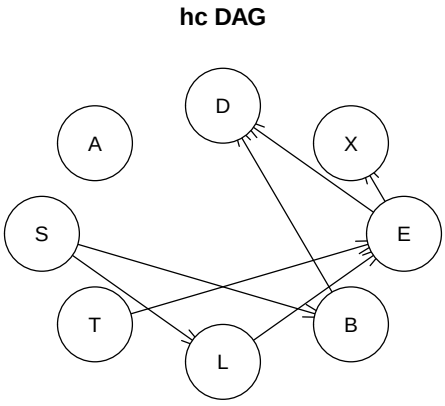


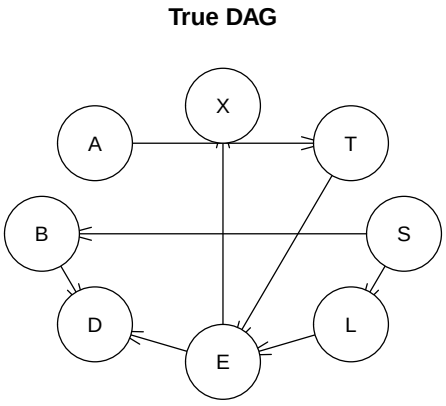
2. *Learn a BN from 80 % of the Asia dataset. Use the BN learned to classify the remaining 20 % of the Asia dataset in two classes: S=yes and S=no. Report the confusion matrix, i.e. true/false positives/negatives.*

In this step we will learn a Bayesian Network from 80% of the Asian dataset using a score-based algorithm which assigns a score to each candidate Bayesian network and try to maximize it with a Greedy search algorithm.

The function `bn.fit` will fit the parameters. We will start from two different DAGs: `learn`, output of the function `hc`, and `dag`, which is the true Asian BN.

```
#2.

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

#determine the structure of a DAG
learn=hc(train)

#determine the type of data and fit parameters
fit=bn.fit(learn,train)
fit_true=bn.fit(dag,train)
```





After we learnt the parameters we would like to classify the remaining 20% of the Asian dataset in two classes (S=yes,S=no).

We will apply both methods: exact Inference and approximate Inference.

Exact Inference

Here we will use Junction Trees, an exact Inference Algorithm which creates a tree in which each clique is a node and adjacent cliques are linked by arcs.

To find the predicted values for the variable S we first will use `setFinding` function on each row of our test dataset and then we will find the posterior probabilities using the function `querygrain` on the previous output specifying which node we want to make inferences on.

```
#Exact inference - junction trees (sequenctial and iterative in nature) -
junc=compile(as.grain(fit))
junc_true=compile(as.grain(fit_true))

pred<-c()
pred_true<-c()

for(i in 1:nrow(test)){

  a=setFinding(junc,nodes=colnames(test[,-2]),states=t(test[i,-2]))
  b=setFinding(junc_true,nodes=colnames(test[,-2]),states=t(test[i,-2]))

  probS=querygrain(a,nodes=c("S"),evidence=a$evidence)
  probS_true=querygrain(b,nodes=c("S"),evidence=b$evidence)

  answer=c("no","yes")
  pred[i]=answer[ifelse(probS$S[1]>=0.5,1,2)]
  pred_true[i]=answer[ifelse(probS_true$S[1]>=0.5,1,2)]

}
```

The following tables show two confusion matrix, the first one refers to our Bayesian Network and the second one refers to the true Asian Bayesian Network.

	no	yes
no	353	127
yes	127	393

	no	yes
no	353	127
yes	127	393

As we can see both the predictions give a misclassification error of 25.4 %. Every node is dependent of S because it is connected to S through a path. But B and L are the Markov blanket of S, which means that the rest of the nodes are independent of S given B and L, and thus B and L are all that is needed to classify S.

Approximate Inference

In approximate inference we loop through all rows in the testdata and a sample of random observations conditional on the evidence is generated for each row. Then the prediction for that row is made with majority vote.

```
#Approximate inference
prediction <- c()
for (i in 1:nrow(test)){
  ran_obs <- cpdist(fit, nodes = "S", evidence = ((A == test[i,"A"]) & (T == test[i,"T"]) &
                      (L == test[i,"L"]) & (B == test[i,"B"]) & (E == test[i,"E"]) &
                      (X == test[i,"X"]) & (D == test[i,"D"])))
  prediction[i] <- names(which.max(table(ran_obs)))
}

#true DAG
prediction_dag <- c()
for (i in 1:nrow(test)){
  ran_obs <- cpdist(fit_true, nodes = "S", evidence = ((A == test[i,"A"]) & (T == test[i,"T"]) &
                      (L == test[i,"L"]) & (B == test[i,"B"]) & (E == test[i,"E"]) &
```

```
                        (X == test[i,"X"]) & (D == test[i,"D"])))
  prediction_dag[i] <- names(which.max(table(ran_obs)))
}
```

The following tables show two confusion matrix, the first one refers to our Bayesian Network and the second one refers to the true Asian Bayesian Network.

	no	yes
no	354	126
yes	131	389

	no	yes
no	354	126
yes	129	391

Comparing the prediction results with the ones of the true Asia Bayesian Network using Approximate inference we obtain:

Var1	Freq
FALSE	4
TRUE	996

In 4 cases the predictions are different, this is because the conditional probabilities are different in the two DAGs.

3. *Now, you are asked to classify S given observations only for the so-called Markov blanket of S, i.e. its parents plus its children plus the parents of its children minus S itself. Report again the confusion matrix.*

We repeat the same process of the previous step but this time we only use observations for the Markov blanket of S which are L, B.

```
#3.

pred_MB<-c()

for(i in 1:nrow(test)){

  a=setFinding(junc,nodes=mb(learn,"S"),states=t(test[i,mb(learn,"S")]))

  probS=querygrain(a,nodes=c("S"),evidence=a$evidence)

  answer=c("no","yes")
  pred_MB[i]=answer[ifelse(probS$S[1]>=0.5,1,2)]

}
```

The following confusion matrix is obtained:

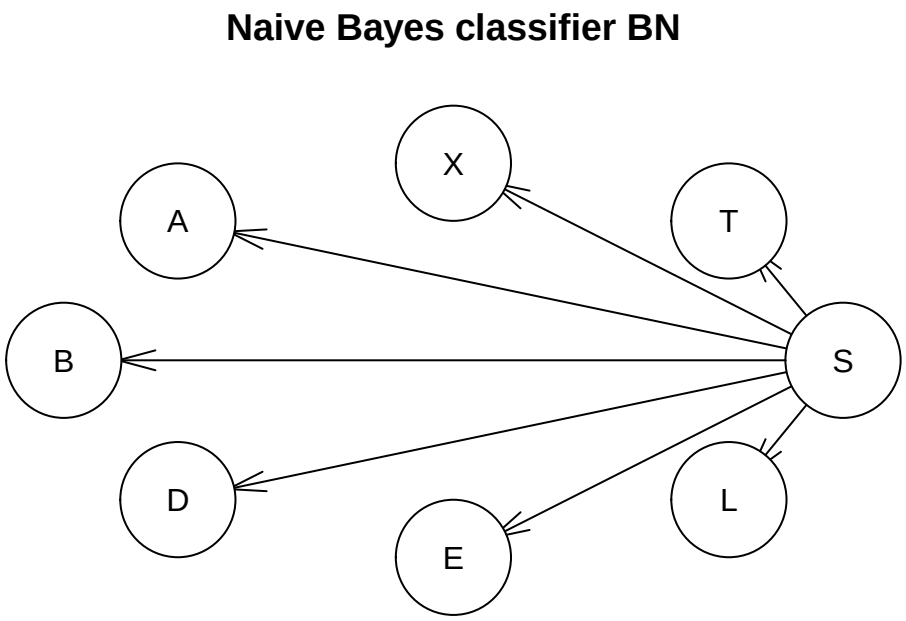
	no	yes
no	353	127
yes	127	393

Also in this case we obtain the same confusion matrix for the same reason we stated before, S in the DAG is only "affected" from its the Markov blanket. Using the approximate inference we also obtain approximately the same results (1-2 different observations).

4. *Repeat the exercise (2) using a naive Bayes classifier, i.e. the predictive variables are independent given the class variable. Model the naive Bayes classifier as a BN.*

We first create a DAG in which the predictive variables are independent given the class variable as it follow:

```
#4.

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



Now we repeat what we did in step (2) but using the DAG from `e`.

```
fit_NB=bn.fit(e,train)

junc_NB=compile(as.grain(fit_NB))

pred_NB<-c()

for(i in 1:nrow(test)){

  a=setFinding(junc_NB,nodes=colnames(test[,-2]),states=t(test[i,-2]))

  probS=querygrain(a,nodes=c("S"),evidence=a$evidence)

  answer=c("no","yes")
  pred_NB[i]=answer[ifelse(probS$S[1]>=0.5,1,2)]

}
```

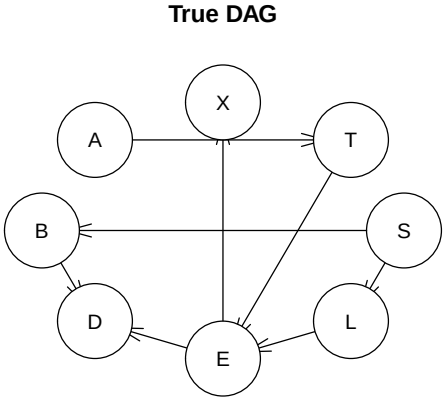
We obtain the following confusion matrix:

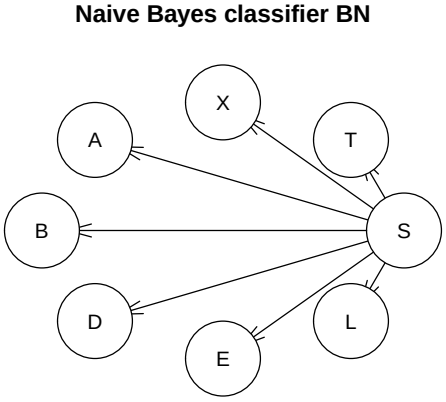
	no	yes
no	372	108
yes	196	324

The misclassification error in this case is 30.4 % , a bit more.

5. *Explain why you obtain the same or different results in the exercises (2-4).*

Step 2. and 4. give different results, this happens because the two DAGs we used don't have the same dependencies. The Naive Bayes model considers nodes that are outside the true Markov blanket when classifying S. This just hurts performance. For this reason learning from the two DAGs doesn't give the same values of parameters and then the same conditional distributions.





Appendix

```
knitr::opts_chunk$set(echo = TRUE)
#1.

# source("http://bioconductor.org/biocLite.R")
# biocLite("RBGL")
library(bnlearn)
library(gRain)
library(knitr)
set.seed(12345)

data("asia")

starting=random.graph(names(asia)) #random DAG as starting one

trial_1=hc(asia, restart = 11)
plot(cpdag(trial_1))
trial_3=hc(asia, start=starting)
plot(cpdag(trial_3))

trial_2=hc(asia, score="bic")
trial_4=hc(asia, score="aic")
trial_5=hc(asia, score="loglik")

#true Asia BN
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

all.equal(trial_1,trial_2)
all.equal(trial_1,trial_3)
all.equal(trial_2,trial_3)

par(mfrow=c(2,2))

plot(trial_1,main="DAG with restart=11")
plot(trial_3,main="DAG with random initial structure")
plot(trial_2,main="DAG with score=BIC")
plot(trial_4,main="DAG with score=AIC")
plot(trial_5,main="DAG with score=loglik")
plot(dag,main="True DAG")

#2.

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

#determine the structure of a DAG
learn=hc(train)

#determine the type of data and fit parameters
fit=bn.fit(learn,train)
```

```r
fit_true=bn.fit(dag,train)
par(mfrow=c(1,2))
plot(learn,main="hc DAG")
plot(dag, main="True DAG")


#Exact inference - junction trees (sequenctial and iterative in nature) -
junc=compile(as.grain(fit))
junc_true=compile(as.grain(fit_true))

pred<-c()
pred_true<-c()

for(i in 1:nrow(test)){

  a=setFinding(junc,nodes=colnames(test[,-2]),states=t(test[i,-2]))
  b=setFinding(junc_true,nodes=colnames(test[,-2]),states=t(test[i,-2]))

  probS=querygrain(a,nodes=c("S"),evidence=a$evidence)
  probS_true=querygrain(b,nodes=c("S"),evidence=b$evidence)

  answer=c("no","yes")
  pred[i]=answer[ifelse(probS$S[1]>=0.5,1,2)]
  pred_true[i]=answer[ifelse(probS_true$S[1]>=0.5,1,2)]

}
library(knitr)
library(kableExtra)

t=table(test$S,pred)
t_true=table(test$S,pred_true)
kable(t)
kable(t_true)


#Approximate inference
prediction <- c()
for (i in 1:nrow(test)){
  ran_obs <- cpdist(fit, nodes = "S", evidence = ((A == test[i,"A"]) & (T == test[i,"T"]) &
                      (L == test[i,"L"]) & (B == test[i,"B"]) & (E == test[i,"E"]) &
                      (X == test[i,"X"]) & (D == test[i,"D"])))
  prediction[i] <- names(which.max(table(ran_obs)))
}

#true DAG
prediction_dag <- c()
for (i in 1:nrow(test)){
  ran_obs <- cpdist(fit_true, nodes = "S", evidence = ((A == test[i,"A"]) & (T == test[i,"T"]) &
                      (L == test[i,"L"]) & (B == test[i,"B"]) & (E == test[i,"E"]) &
                      (X == test[i,"X"]) & (D == test[i,"D"])))
  prediction_dag[i] <- names(which.max(table(ran_obs)))
}
```

```r
# Report the confusion matrix
confusion_matrix <- table(test$S,prediction)
confusion_matrix_true<-table(test$S,prediction_dag)

kable(confusion_matrix)
kable(confusion_matrix_true)


t_app=table(prediction == prediction_dag)

kable(t_app)

#3.

pred_MB<-c()

for(i in 1:nrow(test)){

  a=setFinding(junc,nodes=mb(learn,"S"),states=t(test[i,mb(learn,"S")]))

  probS=querygrain(a,nodes=c("S"),evidence=a$evidence)

  answer=c("no","yes")
  pred_MB[i]=answer[ifelse(probS$S[1]>=0.5,1,2)]

}
t_MB=table(test$S,pred_MB)

kable(t_MB)

#4.

e=model2network("[S][T|S][L|S][B|S][D|S][E|S][X|S][A|S]")
plot(e,main="Naive Bayes classifier BN")

fit_NB=bn.fit(e,train)

junc_NB=compile(as.grain(fit_NB))

pred_NB<-c()

for(i in 1:nrow(test)){

  a=setFinding(junc_NB,nodes=colnames(test[,-2]),states=t(test[i,-2]))

  probS=querygrain(a,nodes=c("S"),evidence=a$evidence)

  answer=c("no","yes")
  pred_NB[i]=answer[ifelse(probS$S[1]>=0.5,1,2)]

}
```

```r
t_NB=table(test$S,pred_NB)

kable(t_NB)


#5.
par(mfrow=c(1,2))
plot(dag, main="True DAG")
plot(e,main="Naive Bayes classifier BN")
```

Lab 2 - AML Group 1 Report

Alessia De Biase, Alejandro Garcia, Anna-Katharina Fürgut, Erika Anderskär

In this lab we are asked to model the behavior of a robot that walks around a ring. The ring is divided into 10 sectors. At any given time point, the robot is in one of the sectors and decides with equal probability to stay in that sector or move to the next sector. We don't have direct observation of the robot but it is equipped with a tracking device that we can access. The device is not very accurate so if the robot is in the sector i then the device will report that the robot is in the sectors [*i-2,i+2*] with equal probability.

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

To build an HMM we will use the function initHMM which takes as parameters:

- `States`: a vector of 10 states (enumerate from 1 to 10) indicating the 10 sectors our robot can be into;
- `Symbol`: a vector of possible observations, so again a vector of number from 1 to 10;
- `startProb`: a vector with the starting probabilities of the states, in this case the probability of starting in one position or in another one is the same so we have a 10% probability of being in each of the states;
- `transProbs`: the transition probability matrix between the states, in this case we have a probability of 50% of remaining in the actual state (i) and a probability of 50% of moving to the next state (i+1);
- `emissionProb`: matrix of emission probabilities of the states, knowing that we observe from the sensor that the robot is in position i, how likely may it truly be there or in the other positions? Because of the low accuracy of the sensor we have a probability of 20% of being in each of the following: i-2, i-1, i, i+1 and i+2.

```
library(HMM)
# (1)
States <- 1:10
sectors <- 10
intervalLength <- 5
transition_change_prob <- 1/2
emission_prob <- 1/intervalLength
# Creating Transition Matrix
trans_mat <- matrix(0,nrow = sectors, ncol = sectors)

# Creating Emission Matrix
emission_mat <- matrix(0,nrow = sectors,ncol = sectors)

for (j in 1:sectors) {
  trans_mat[j, j %% 10 + 1] <- 1/2
  trans_mat[j,j] <- 1/2
  for (i in -2:2) {
    emission_mat[j, (j + i - 1) %% 10 + 1] <- 1/5
  }
}

trans_mat[10,1] <- transition_change_prob
model <- initHMM(States = as.character(1:sectors),
                     Symbols = as.character(1:sectors),
                     transProbs = trans_mat,
                     emissionProbs = emission_mat)
```

(2)

We now simulate the HMM for 100 timesteps with the function `simHMM`.

```
# (2)
set.seed(1234)
simulation_model<- simHMM(model, length = 100)
```

(3)

We discard the hidden states from the sample obtained above. For that we use the remaining observations to compute the filtered and smoothed probability distributions for each of the 100 time points. Additionally, we compute the most probable path.

```
# (3)
sim_states <- simulation_model$states
sim_obs <- simulation_model$observation
filtered_logProbs <- forward(model, observation = sim_obs)
backwards_probs <- backward(model, observation = sim_obs)
#smoothed_probs <- posterior(model, observation = sim_obs)
most_probable_path <- viterbi(model, observation = sim_obs)
```

(4)

We compute the accuracy of the filtered and smoothed probability distributions, and of the most probable path. We are doing that by computing the percentage of the true hidden states that are guessed by each method.

```
# (4)
filter_probtable <- prop.table(exp(filtered_logProbs),2)
smoothed_probtable <- prop.table(exp(backwards_probs+filtered_logProbs),2)

filtered_schatzer <- apply(filter_probtable, 2, which.max)
smoothed_schatzer <- apply(smoothed_probtable, 2, which.max)

filtered_acc <- mean(simulation_model$states == filtered_schatzer)
smoothed_acc <- mean(simulation_model$states == smoothed_schatzer)
most_probable_path_acc <- mean(simulation_model$states == most_probable_path)
```

Accuracies:

```
# Filtered:
filtered_acc
```

```
## [1] 0.63
```

```
# Smoothed:
smoothed_acc
```

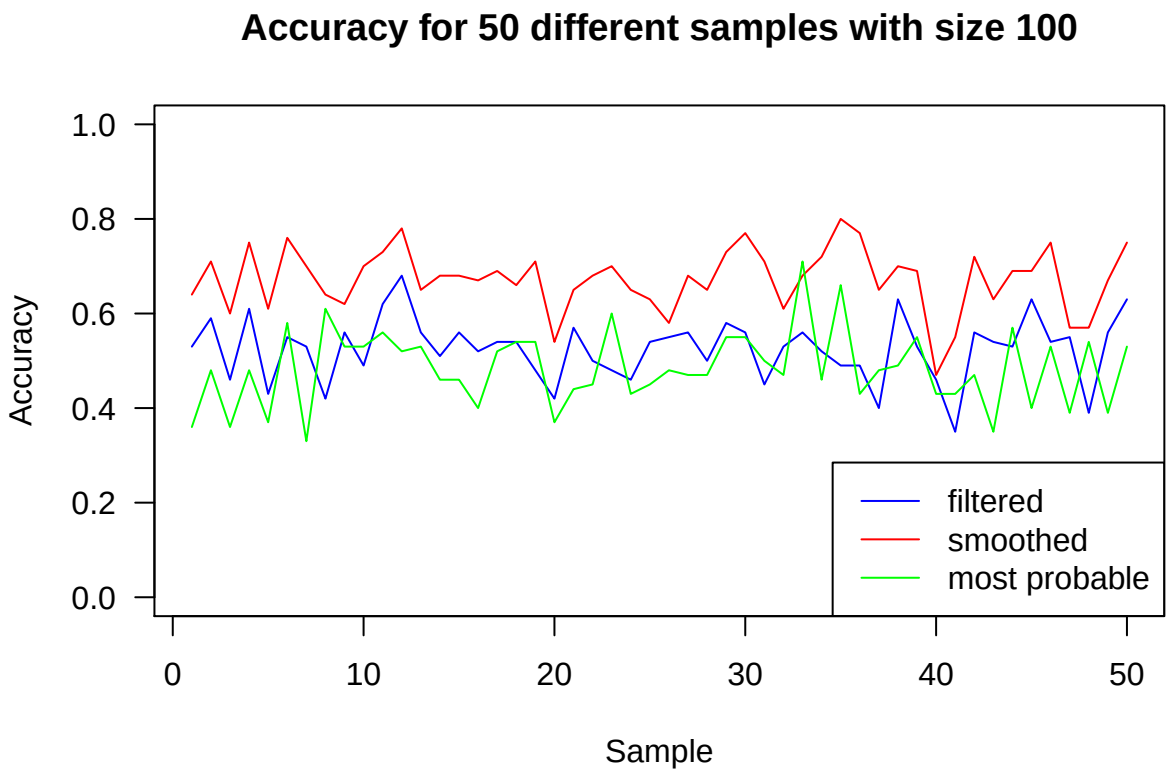
```
## [1] 0.75
```

```r
# Most probable path:
most_probable_path_acc
```

```
## [1] 0.49
```

(5)

```r
# (5)
set.seed(123)
acc_samp <- matrix(ncol = 3, nrow = 50)
for(i in 1:50){
  simulation_model<- simHMM(model, length = 100)
  sim_states <- simulation_model$states
  sim_obs <- simulation_model$observation
  filtered_logProbs <- forward(model, observation = sim_obs)
  backwards_probs <- backward(model, observation = sim_obs)
  most_probable_path <- viterbi(model, observation = sim_obs)

  filter_probtable <- prop.table(exp(filtered_logProbs),2)
  smoothed_probtable <- prop.table(exp(backwards_probs+filtered_logProbs),2)

  filtered_schatzer <- apply(filter_probtable, 2, which.max)
  smoothed_schatzer <- apply(smoothed_probtable, 2, which.max)

  acc_samp[i,1] <- mean(simulation_model$states == filtered_schatzer)
  acc_samp[i,2] <- mean(simulation_model$states == smoothed_schatzer)
  acc_samp[i,3] <- mean(simulation_model$states == most_probable_path)

  # for number (7)
  prediction_101_filter <- filter_probtable[,100] %*% trans_mat
  prediction_101_smoothed <- smoothed_probtable[,100] %*% trans_mat
}

plot(x = 1:50,y = acc_samp[,1], col = "blue", type = "l",
     ylim = c(0,1),ylab = "Accuracy",xlab = "Sample", las = 1,
     main = "Accuracy for 50 different samples with size 100")
lines(x = 1:50,y = acc_samp[,2], col = "red", type = "l")
lines(x = 1:50,y = acc_samp[,3], col = "green",type = "l")
legend("bottomright", col = c("blue", "red", "green"), lty = rep(1,3),
       legend = c("filtered", "smoothed", "most probable"))
```



The smoothed distribution is more accurate than the filtered one, since it uses all observations and not only the observations preceding the current timestep. In contrast to the most probable path, the smoothed distribution is a point estimate without the restriction that the sequence has to be a valid path.

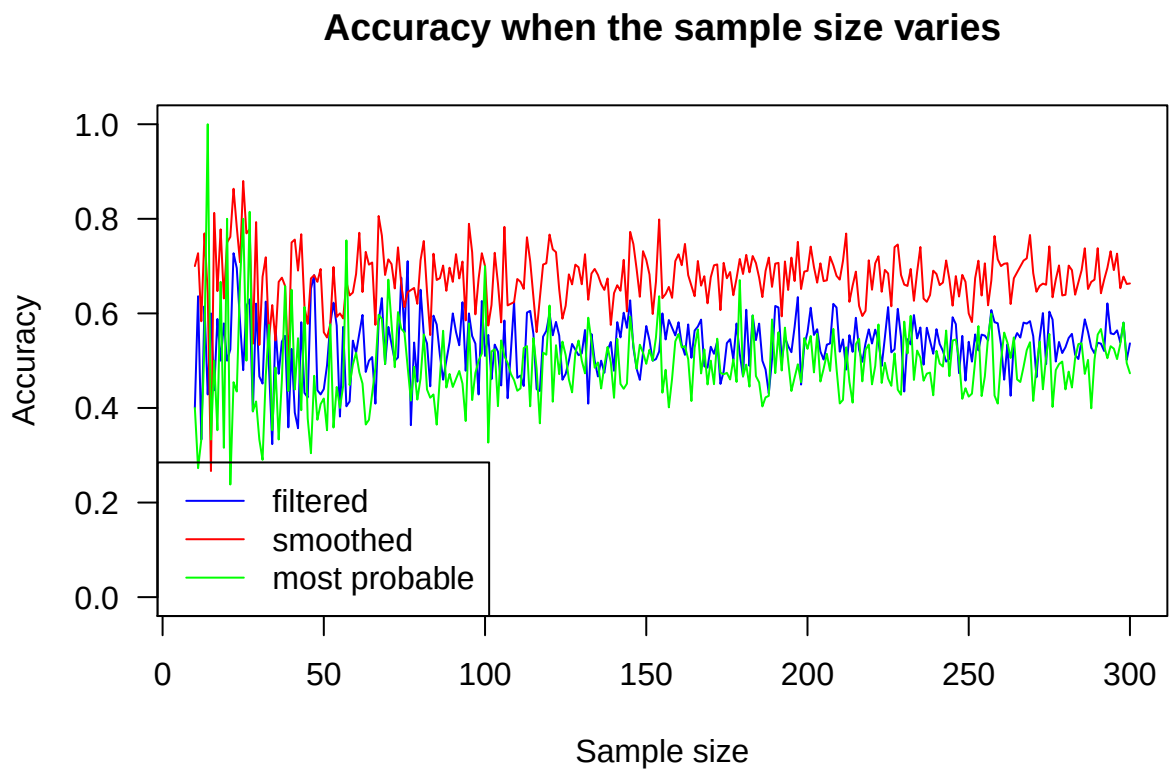
Generally for the smoothed distribution more observations are used. While the Viterbi computes the most probable path (which is in contrast to the other two always consistent/valid and can not accept a longer step than allowed), the smoother computes the state that maximizes the marginal distribution for each time step. Since the accuracy is computed by comparing the most probable estimate for each time step individually, the smoother will yield better results.

Alltough smoothed distribution provides the best accuracy, it shouldn't be forgotten that knowledge about future values and non valid paths is not always reliable.

(6)

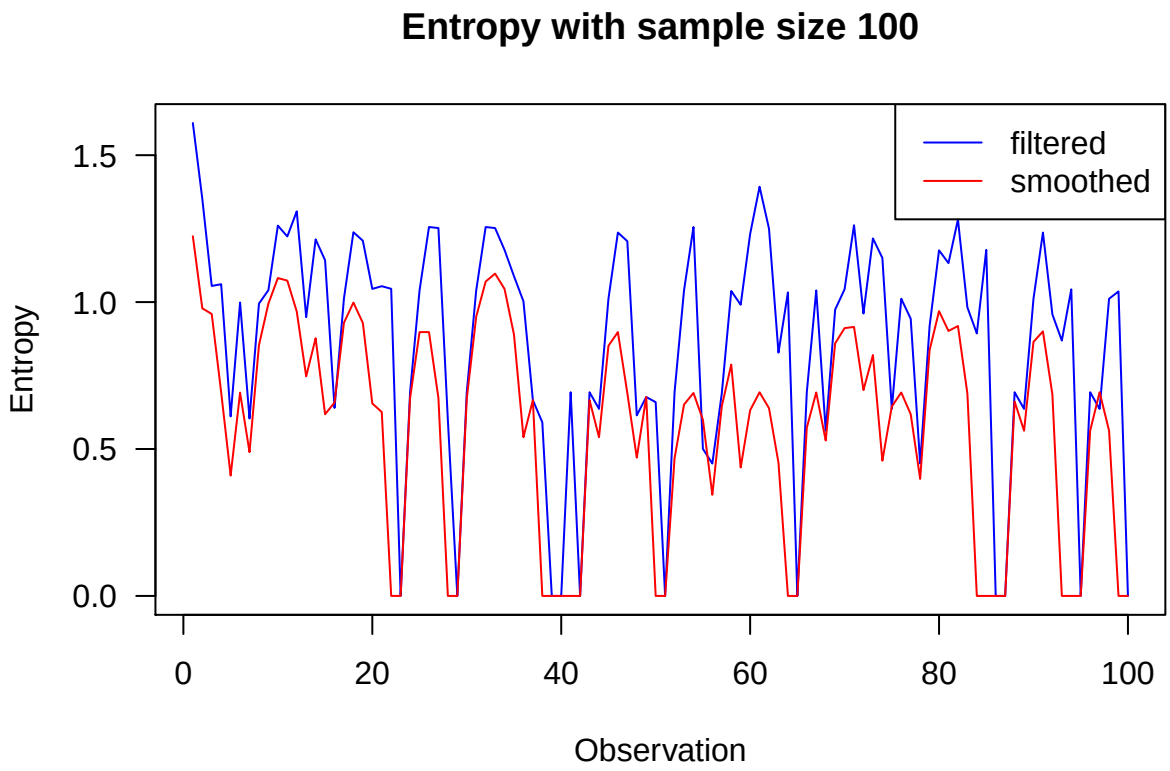
```
# (6)
library(entropy)
sample_size <- seq(from = 10, to = 300, by = 1)
set.seed(777)
acc_samp_6 <- matrix(ncol = 3, nrow = length(sample_size))
for(i in 1:length(sample_size)){
  simulation_model_6 <- simHMM(model, length = sample_size[i])
  sim_states_6 <- simulation_model_6$states
  sim_obs_6 <- simulation_model_6$observation
  filtered_logProbs_6 <- forward(model, observation = sim_obs_6)
```

```
  backwards_probs <- backward(model, observation = sim_obs_6)
  most_probable_path <- viterbi(model, observation = sim_obs_6)

  filter_probtable_6 <- prop.table(exp(filtered_logProbs_6),2)
  smoothed_probtable_6 <- prop.table(exp(backwards_probs+filtered_logProbs_6),2)

  filtered_schatzer <- apply(filter_probtable_6, 2, which.max)
  smoothed_schatzer_6 <- apply(smoothed_probtable_6, 2, which.max)

  acc_samp_6[i,1] <- mean(simulation_model_6$states == filtered_schatzer)
  acc_samp_6[i,2] <- mean(simulation_model_6$states == smoothed_schatzer_6)
  acc_samp_6[i,3] <- mean(simulation_model_6$states == most_probable_path)

}

# Accuracy Plot
plot(x = sample_size,y = acc_samp_6[,1], col = "blue", type = "l",
     ylim = c(0,1), ylab = "Accuracy",xlab = "Sample size", las = 1,
     main = "Accuracy when the sample size varies")
lines(x = sample_size, y = acc_samp_6[,2], col = "red", type = "l")
lines(x = sample_size, y = acc_samp_6[,3], col = "green",type = "l")
legend("bottomleft", col = c("blue", "red", "green"), lty = rep(1,3),
       legend = c("filtered", "smoothed", "most probable"))
```




```
# Entropy Plot
filter_entropy <- apply(filter_probtable, 2, entropy.empirical)
```

```r
smoothed_entropy <- apply(smoothed_probtable, 2, entropy.empirical)

plot(x = 1:length(filter_entropy),y = filter_entropy, col = "blue", type = "l",
      ylab = "Entropy",xlab = "Observation", las = 1,
     main = "Entropy with sample size 100")
lines(x = 1:length(smoothed_entropy), y = smoothed_entropy, col = "red", type = "l")
legend("topright", col = c("blue", "red"), lty = rep(1,2),
       legend = c("filtered", "smoothed"))
```



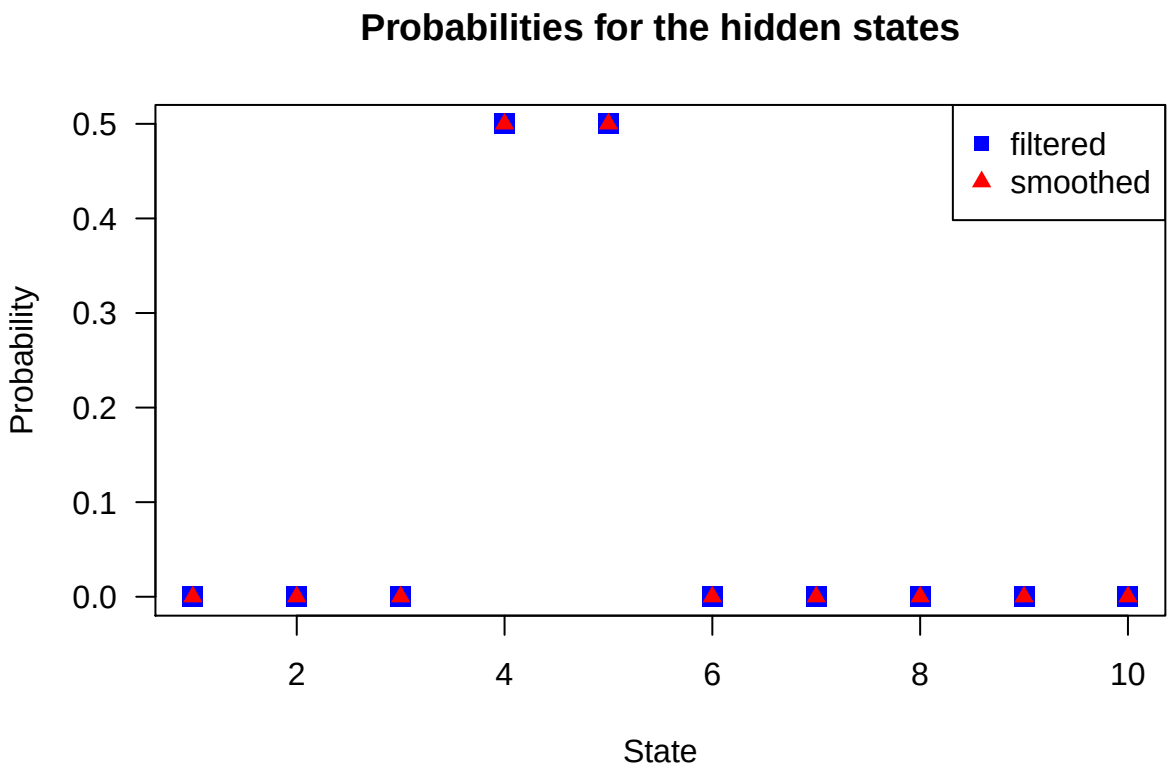
As can be seen in the plot, the entropy does not decrease for higher sample sizes. A possible reason is that the hidden markov model introduces uncertainty in each time step. This is an inherent property for HMM in general.

The entropy varies largely due to noise and inaccuracy in the robot's tracking device. Even though a sequence can build up to a certainty, the next step might confuse and decrease the knowledge and therefore increase the entropy once again.

(7)

```r
# (7)
prediction_101_filter <- filter_probtable[,100] %*% trans_mat
prediction_101_smoothed <- smoothed_probtable[,100] %*% trans_mat

plot(x = 1:10, y = prediction_101_filter, col = "blue", pch = 15,cex=1.4,
```

```
     ylab = "Probability",xlab = "State", las = 1,
     main = "Probabilities for the hidden states")
points(x = 1:10, y = prediction_101_smoothed, col = "red", pch = 17)#,type = "l")
legend("topright", col = c("blue", "red"), pch = c(15,17),
       legend = c("filtered", "smoothed"))
```



AML_Lab3

Alejandro Garcia, Alessia De Biase, Anna-Katharina Fürgut, Erika Anderskär

2 October 2018

State Space Models

In this assignment we are asked to implement the particle filter for robot localization. The robot moves along the horizontal axis according to the following **SSM**:

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

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

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

1. *Implement the SSM above.Simulate the SSM for T=100 time steps to obtain $z_{1:100}$ and $x_{1:100}$. Use the observations to identify the state via particle filtering. Use 100 particles. Show the particles, the expected location and the true location for the first and last time steps, as well as for two intermediate time steps of your choice.*

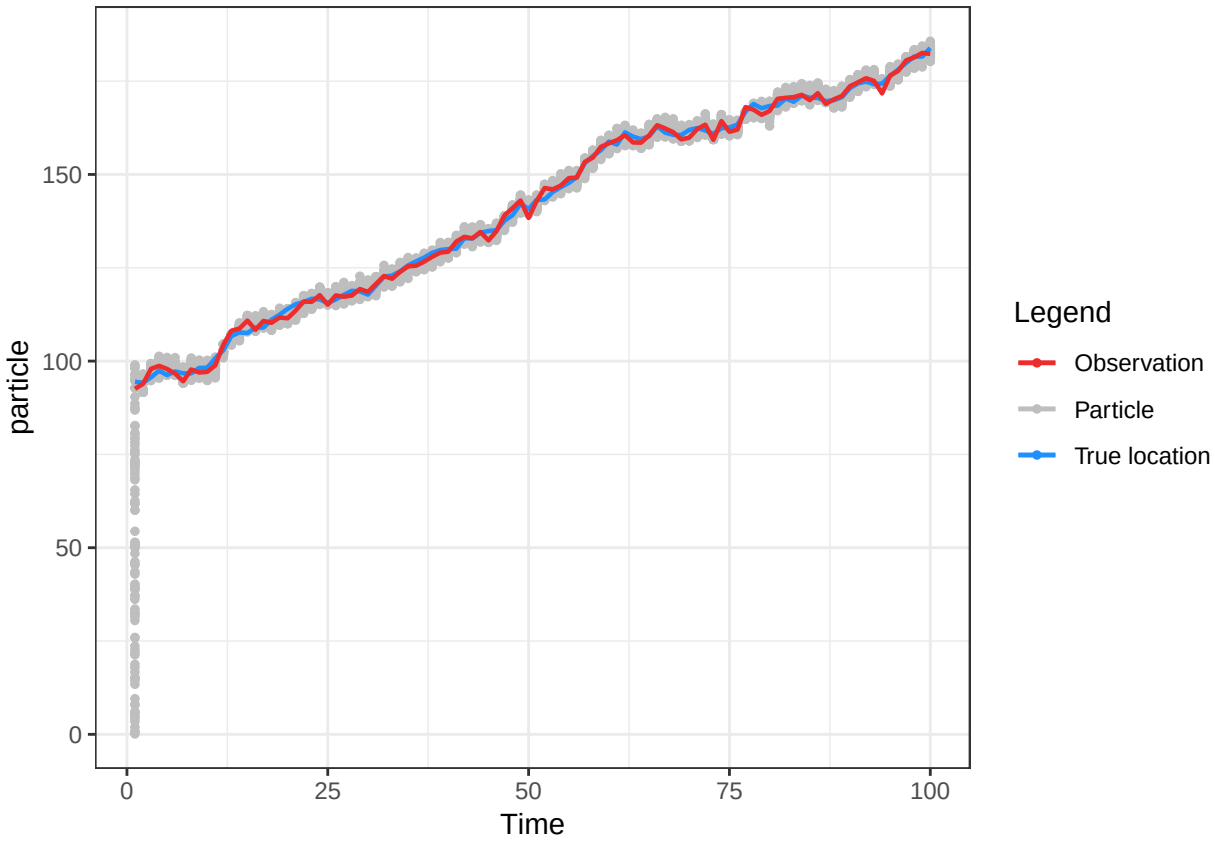
```
generate_data <- function(variance){

  zt <- runif(100, 0, 100)

  #Transition model
  for(i in 1:99){
    model <- c()
    model[1] <- rnorm(1,zt[i], 1)
    model[2] <- rnorm(1,zt[i]+1, 1)
    model[3] <- rnorm(1,zt[i]+2, 1)
    zt[i+1] <- model[sample(1:3,1)]
  }


  xt <- vector(length = 100)

  #Emission model
  for(i in 1:100){
    model <- c()
    model[1] <- rnorm(1,zt[i], variance)
    model[2] <- rnorm(1,zt[i]+1, variance)
    model[3] <- rnorm(1,zt[i]-1, variance)
    xt[i] <- model[sample(1:3,1)]
  }
  return(data.frame(real = zt, obs = xt))
}

dataVar1 <- generate_data(1)
```

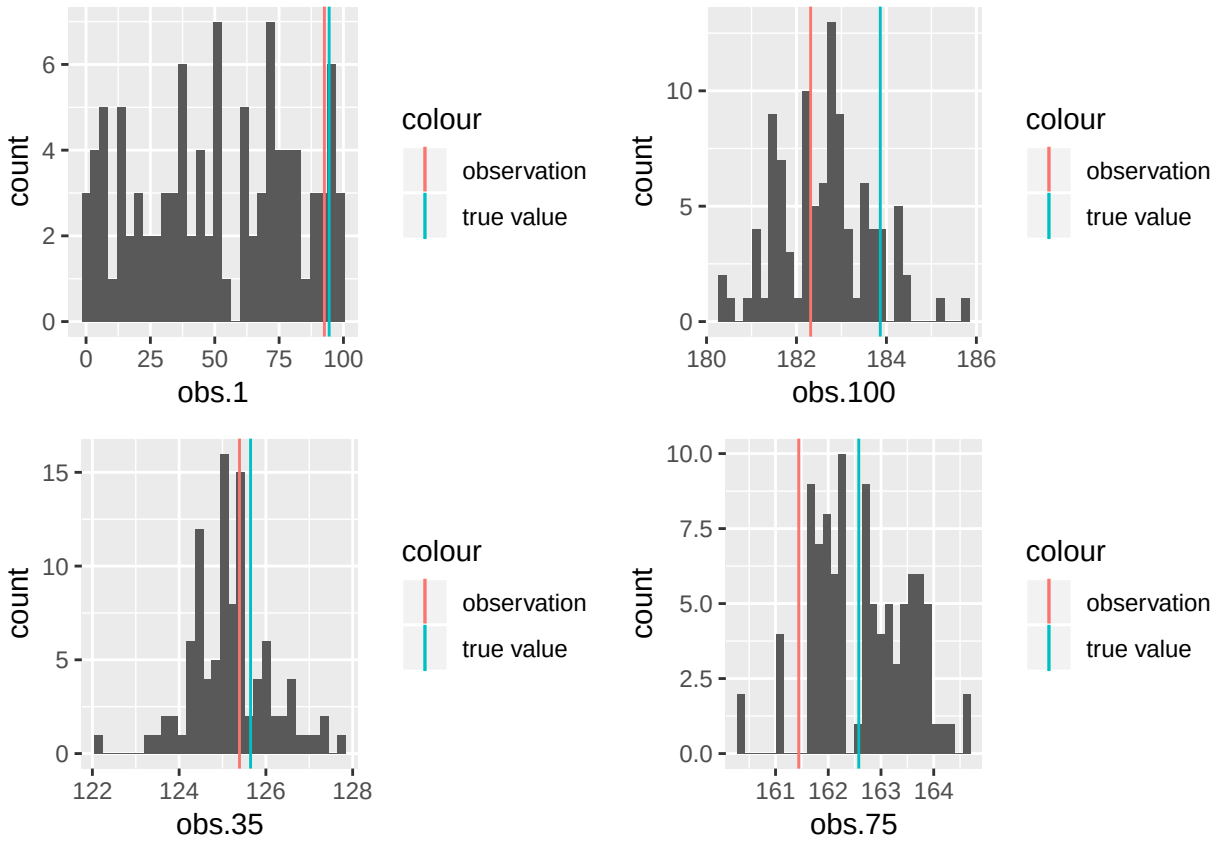
We now implement *Particle Filtering* which approximates the the state space of our robot by a random sample of $bel(x_t)$. The particles (in our case 100) are the samples of the posterior distribution at each time step t.

This algorithm is made of the following steps:

1. **Initialization**: we set the initial weights to 1/n Particles and the initial believe ($Z[1,]$, first set of particles) sampling from a uniform distribution from 0 to 100 ($p(x_0)$);
2. **Prediction**: we calculate each particle at time t using the state transition probability and the same particle at time $t-1$.
3. **Importance Weight**: we calculate the *importance factor*, used to incorporate the observations from simulations into the particle set and they represent the probability of the measurement z_t under the particle m at time t.
4. **Correction**: after building the temporary set of particles and the respective weights, we want to resample. The probability of drawing each particle is given by its importance weight, now the particles are distributed according to the posterior

```
partic_filter <- function(xt, variance){
  M<-100
  chi <- c()

  #Initialization
  chi <- runif(M, 0, 100)
  pred <- matrix(vector(length=100*M), nrow=M)
  pred[,1] <- chi

  for(t in 2:100){
    chihat <- c()
    w <- c()
    for(m in 1:M){
      model <- c()
      model[1] <- rnorm(1,chi[m], 1)
      model[2] <- rnorm(1,chi[m]+1, 1)
      model[3] <- rnorm(1,chi[m]+2, 1)
      xtm <- model[sample(1:3,1)]
      w[m] <- (dnorm(xtm, xt[t], variance)+
                 dnorm(xtm, xt[t]+1, variance)+
                 dnorm(xtm, xt[t]-1, variance))/3
      chihat[m] <- xtm
    }
    probs <- cumsum(w/sum(w))
    for(m in 1:M){
        chi[m] <- chihat[findInterval(runif(1), probs)+1]
    }
    pred[,t] <- chi

  }
  return(pred)
}
```

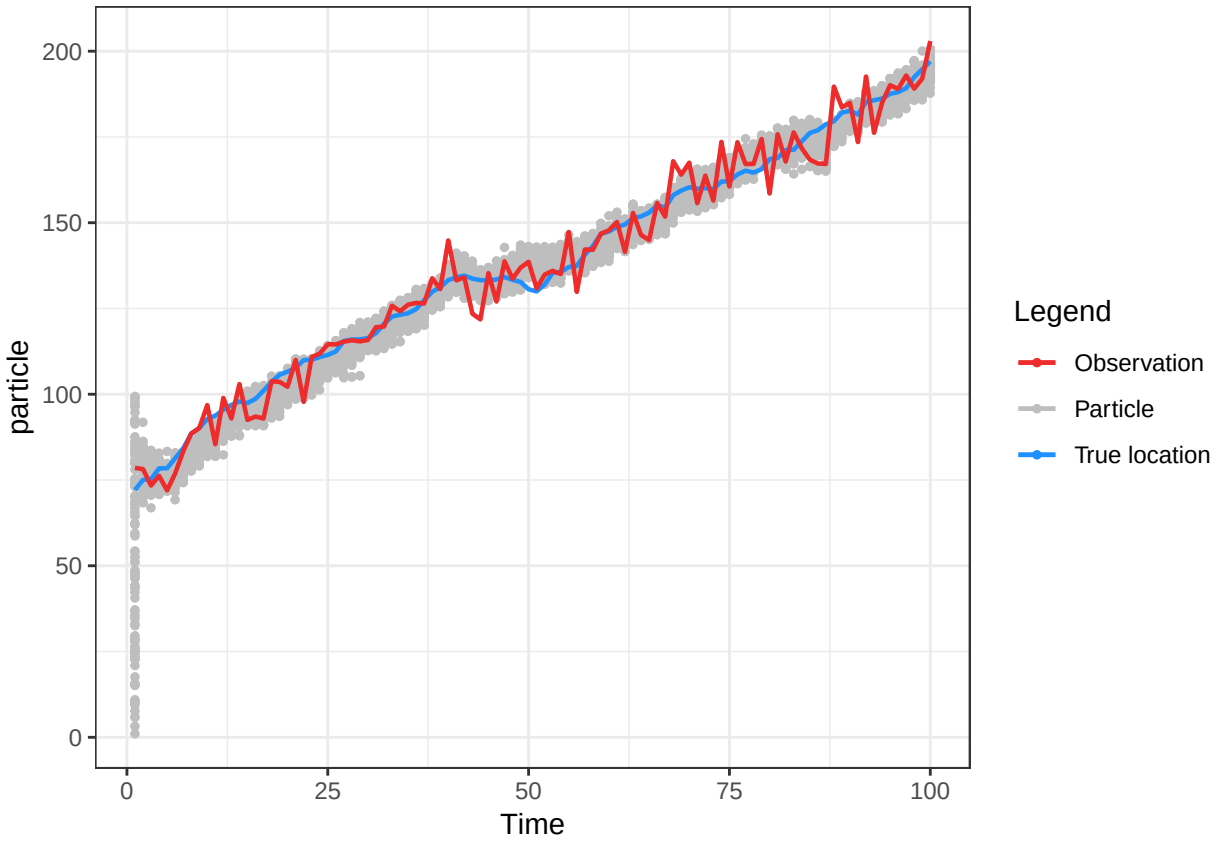

particle
150
100
50
0
0
25
50
75
100
Time
Legend
Observation
Particle
True location

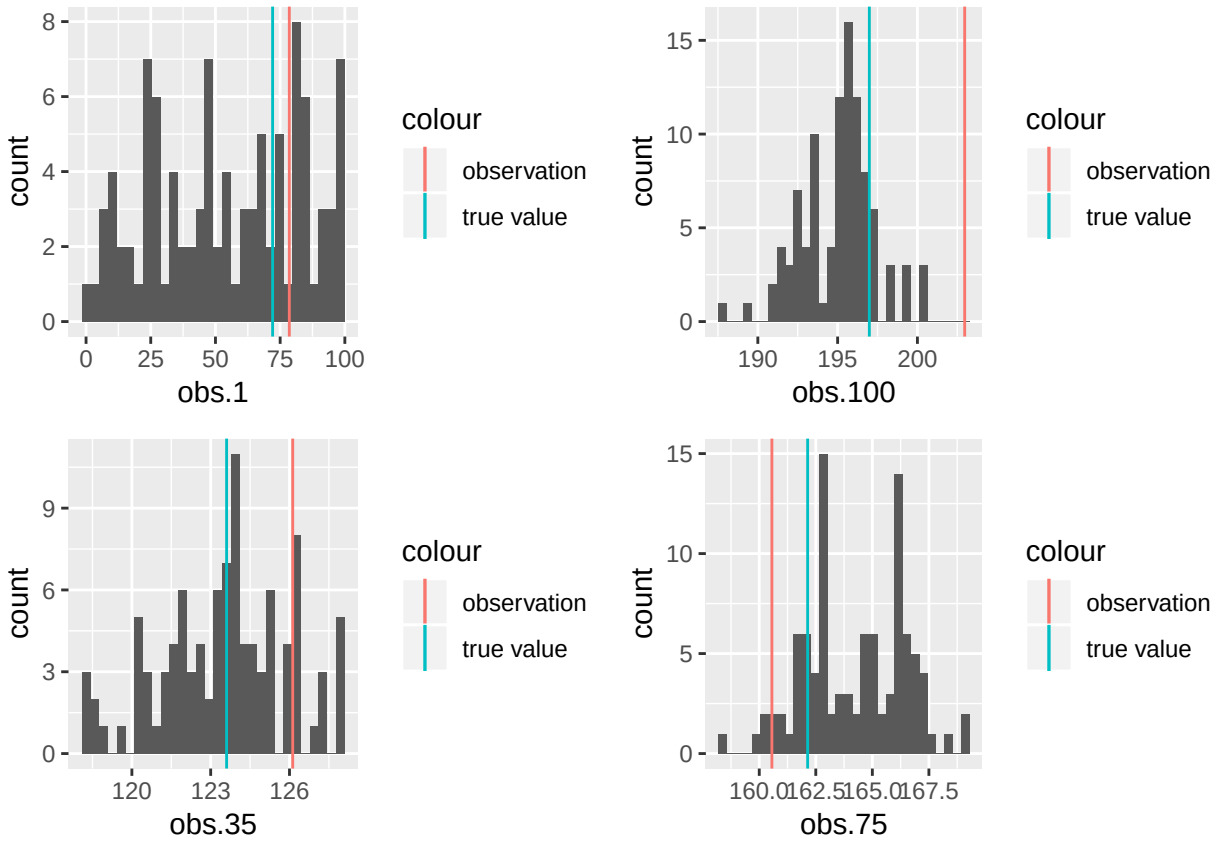


We can see that the distribution of the particles are quite accurate in this case.

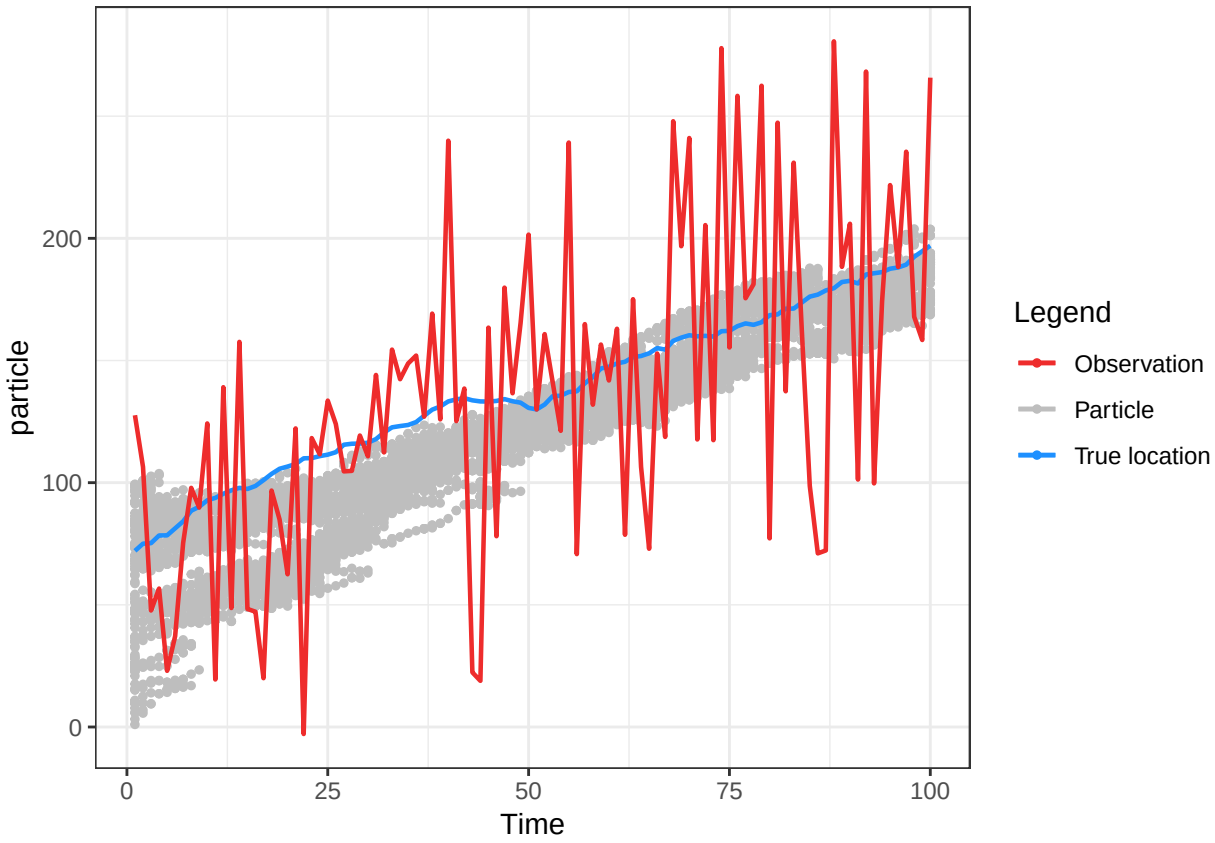
2. *Repeat the exercise above replacing the standard deviation of the emission model with 5 and then with 50. Comment on how this affects the results.*

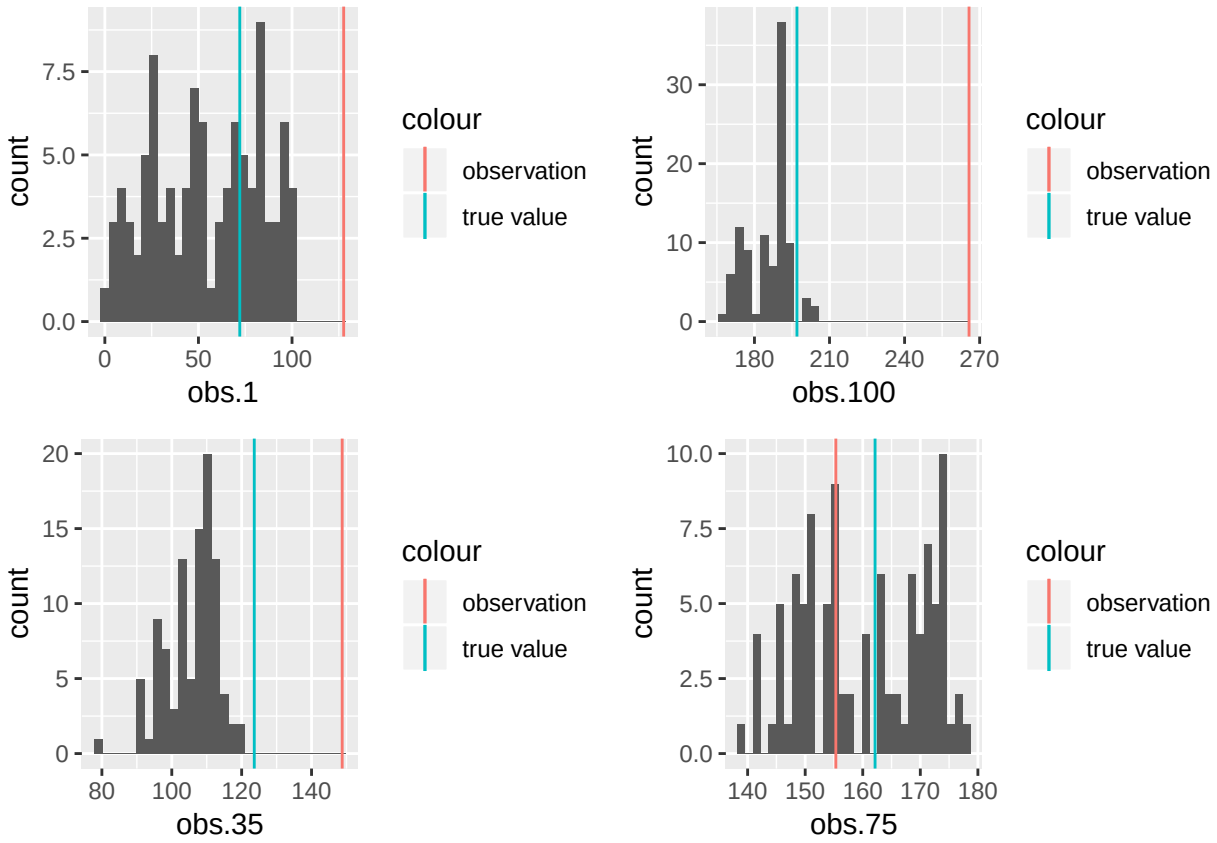
Changing the standard deviation from 1 to 5 we notice that the observations are a bit more instable and a bit far from the real states. Also the particles have different distributions, at each time step they have a bigger variance.




count
obs.1
obs.100
obs.35
obs.75
colour
observation
true value

Changing the standard deviation to 50 adds too much noise to the observations, they are very far apart from our actual states. In contrast to the observations, the particle filtering seems like getting closer to the actual state, without being too influenced by the noise.



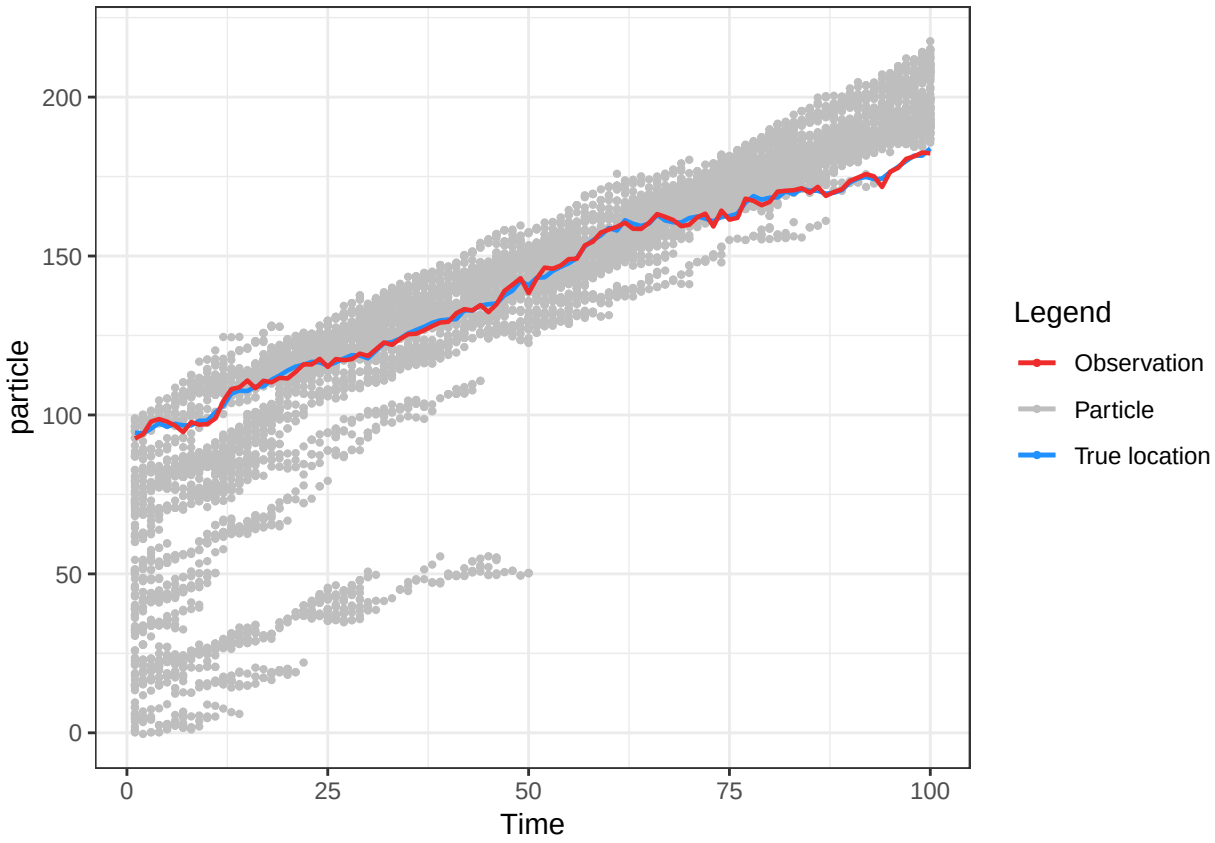


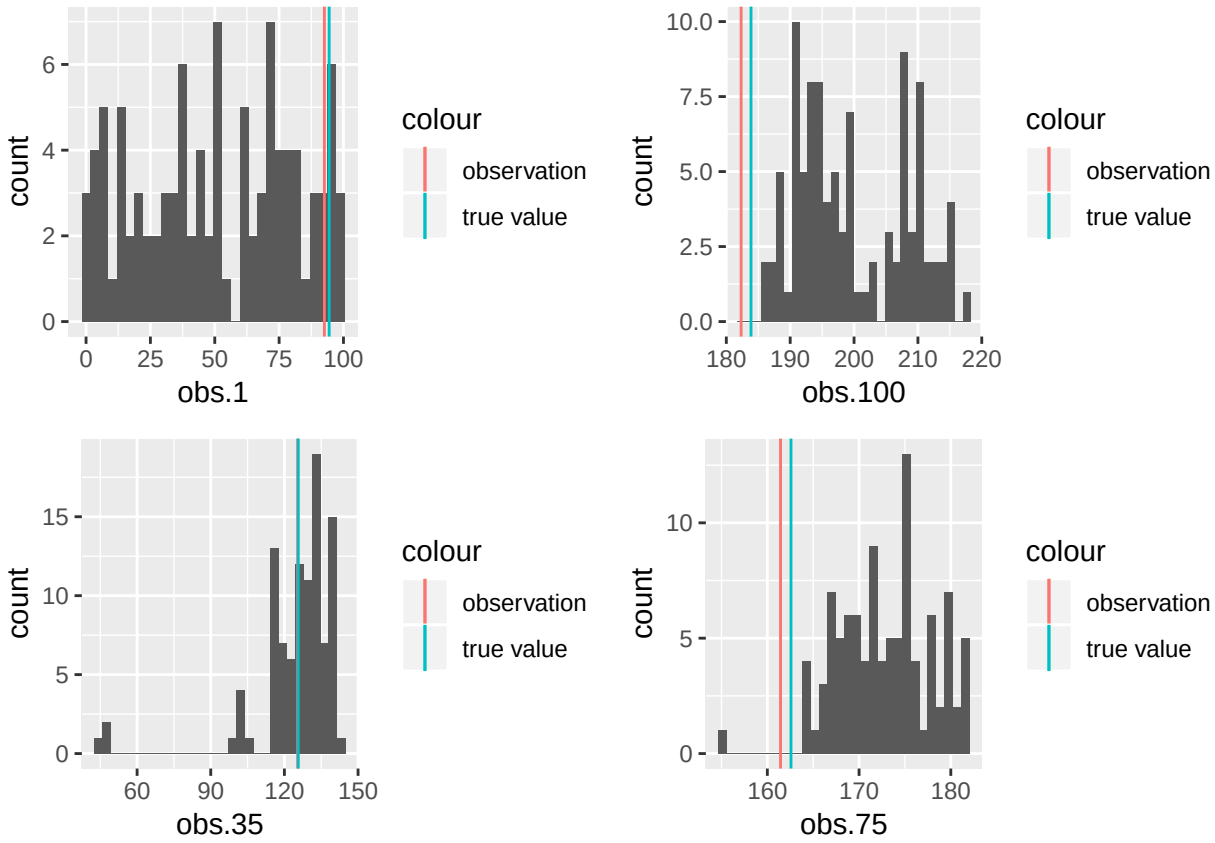
In this case the distribution of the particles is not very informative as the standard deviation is too big to give a proper distribution.

3. *Finally, show and explain what happens with the weights in the particle filter are always equal to 1 (there is not correction).*

The following graph shows what happens when the weights are not updated but they are always equal to 1. As we can see after the time step 1 the particles don't improve their knowledge and the particles are not distributed according to the posterior belief anymore.

```
partic_filter_noweight <- function(xt, variance){
  M<-100
  chi <- c()

  #Initialization
  chi <- runif(100, 0, 100)
  pred <- matrix(vector(length=10000), nrow=100)
  pred[,1] <- chi
  for(t in 2:100){
    chihat <- c()
    w <- c()
    for(m in 1:M){
      model <- c()
      model[1] <- rnorm(1,chi[m], 1)
      model[2] <- rnorm(1,chi[m]+1, 1)
      model[3] <- rnorm(1,chi[m]+2, 1)
      xtm <- model[sample(1:3,1)]
      w <- rep(1,100)
      chihat[m] <- xtm
    }
    probs <- cumsum(w/sum(w))
    for(m in 1:M){
        chi[m] <- chihat[findInterval(runif(1), probs)+1]
    }
    pred[,t] <- chi

  }
  return(pred)
}
```


200
150
100
50
0
particle
0
25
50
75
100
Time
Legend
Observation
Particle
True location


count
obs.1
obs.100
obs.35
obs.75
colour
observation
true value

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
require(ggplot2)
require(gridExtra)
generate_data <- function(variance){

  zt <- runif(100, 0, 100)

  #Transition model
  for(i in 1:99){
    model <- c()
    model[1] <- rnorm(1,zt[i], 1)
    model[2] <- rnorm(1,zt[i]+1, 1)
    model[3] <- rnorm(1,zt[i]+2, 1)
    zt[i+1] <- model[sample(1:3,1)]
  }


  xt <- vector(length = 100)

  #Emission model
  for(i in 1:100){
    model <- c()
    model[1] <- rnorm(1,zt[i], variance)
    model[2] <- rnorm(1,zt[i]+1, variance)
    model[3] <- rnorm(1,zt[i]-1, variance)
    xt[i] <- model[sample(1:3,1)]
  }
  return(data.frame(real = zt, obs = xt))
}

dataVar1 <- generate_data(1)

partic_filter <- function(xt, variance){
  M<-100
  chi <- c()

  #Initialization
  chi <- runif(M, 0, 100)
  pred <- matrix(vector(length=100*M), nrow=M)
  pred[,1] <- chi

  for(t in 2:100){
    chihat <- c()
    w <- c()
    for(m in 1:M){
      model <- c()
      model[1] <- rnorm(1,chi[m], 1)
      model[2] <- rnorm(1,chi[m]+1, 1)
      model[3] <- rnorm(1,chi[m]+2, 1)
      xtm <- model[sample(1:3,1)]
      w[m] <- (dnorm(xtm, xt[t], variance)+
```

```
                dnorm(xtm, xt[t]+1, variance)+
                dnorm(xtm, xt[t]-1, variance))/3
      chihat[m] <- xtm
    }
    probs <- cumsum(w/sum(w))
    for(m in 1:M){
        chi[m] <- chihat[findInterval(runif(1), probs)+1]
    }
    pred[,t] <- chi

  }
  return(pred)
}
set.seed(12345)

var <- 1
pred <- partic_filter(dataVar1$obs, var)
data <- data.frame(obs =pred , real = dataVar1$real,realObs = dataVar1$obs, step = 1:100)

hist1 <- ggplot(data)+
  geom_histogram(aes(obs.1,))+
  geom_vline(aes(xintercept=data$realObs[1],colour="observation"))+
  geom_vline( aes(xintercept = data$real[1],colour="true value"))
hist2 <- ggplot(data)+
  geom_histogram(aes(obs.100))+
  geom_vline(aes(xintercept=data$realObs[100],colour="observation"))+
  geom_vline( aes(xintercept = data$real[100],colour="true value"))
hist3 <- ggplot(data)+
  geom_histogram(aes(obs.35))+
  geom_vline(aes(xintercept=data$realObs[35],colour="observation"))+
  geom_vline( aes(xintercept = data$real[35],colour="true value"))
hist4 <- ggplot(data)+
  geom_histogram(aes(obs.75))+
  geom_vline(aes(xintercept=data$realObs[75],colour="observation"))+
  geom_vline( aes(xintercept = data$real[75],colour="true value"))

N <- 100
M <- 100
locations <- data.frame(index = 1:N,
                        true_location = data$real,
                        exp_location = data$realObs)

partikel <- data.frame(index = rep(1:N, M),
                       particle = as.vector(t(pred)))


ggplot(partikel, aes(x = index), colour = "grey") +
    xlab("Time") +
    theme_bw() +
    geom_point(aes(y = particle, color = "Particle"), size=1) +
    geom_line(data = locations, aes(y = true_location, color = "True location"), size=0.8) +
    geom_line(data = locations, aes(y = exp_location, color = "Observation"), size=0.8) +
    scale_color_manual(values=c("firebrick2", "grey", "dodgerblue1"), name = "Legend")
```

```r
suppressMessages(grid.arrange(hist1, hist2,hist3, hist4, nrow=2))



set.seed(12345)
var <- 5
dataVar5 <- generate_data(var)
pred <- partic_filter(dataVar5$obs, var)
data <- data.frame(obs =pred , real = dataVar5$real,realObs = dataVar5$obs, step = 1:100)

hist1 <- ggplot(data)+
  geom_histogram(aes(obs.1,))+
  geom_vline(aes(xintercept=data$realObs[1],colour="observation"))+
  geom_vline( aes(xintercept = data$real[1],colour="true value"))
hist2 <- ggplot(data)+
  geom_histogram(aes(obs.100))+
  geom_vline(aes(xintercept=data$realObs[100],colour="observation"))+
  geom_vline( aes(xintercept = data$real[100],colour="true value"))
hist3 <- ggplot(data)+
  geom_histogram(aes(obs.35))+
  geom_vline(aes(xintercept=data$realObs[35],colour="observation"))+
  geom_vline( aes(xintercept = data$real[35],colour="true value"))
hist4 <- ggplot(data)+
  geom_histogram(aes(obs.75))+
  geom_vline(aes(xintercept=data$realObs[75],colour="observation"))+
  geom_vline( aes(xintercept = data$real[75],colour="true value"))

N <- 100
M <- 100
locations <- data.frame(index = 1:N,
                        true_location = data$real,
                        exp_location = data$realObs)

partikel <- data.frame(index = rep(1:N, M),
                       particle = as.vector(t(pred)))

ggplot(partikel, aes(x = index), colour = "grey") +
    xlab("Time") +
    theme_bw() +
    geom_point(aes(y = particle, color = "Particle"), size=1) +
    geom_line(data = locations, aes(y = true_location, color = "True location"), size=0.8) +
    geom_line(data = locations, aes(y = exp_location, color = "Observation"), size=0.8) +
    scale_color_manual(values=c("firebrick2", "grey", "dodgerblue1"), name = "Legend")

suppressMessages(grid.arrange(hist1, hist2,hist3, hist4, nrow=2))
set.seed(12345)
var <- 50
dataVar50 <- generate_data(var)
pred <- partic_filter(dataVar50$obs, var)
data <- data.frame(obs =pred , real = dataVar50$real,realObs = dataVar50$obs, step = 1:100)

hist1 <- ggplot(data)+
  geom_histogram(aes(obs.1,))+
```

```
  geom_vline(aes(xintercept=data$realObs[1],colour="observation"))+
  geom_vline( aes(xintercept = data$real[1],colour="true value"))
hist2 <- ggplot(data)+
  geom_histogram(aes(obs.100))+
  geom_vline(aes(xintercept=data$realObs[100],colour="observation"))+
  geom_vline( aes(xintercept = data$real[100],colour="true value"))
hist3 <- ggplot(data)+
  geom_histogram(aes(obs.35))+
  geom_vline(aes(xintercept=data$realObs[35],colour="observation"))+
  geom_vline( aes(xintercept = data$real[35],colour="true value"))
hist4 <- ggplot(data)+
  geom_histogram(aes(obs.75))+
  geom_vline(aes(xintercept=data$realObs[75],colour="observation"))+
  geom_vline( aes(xintercept = data$real[75],colour="true value"))

N <- 100
M <- 100
locations <- data.frame(index = 1:N,
                        true_location = data$real,
                        exp_location = data$realObs)

partikel <- data.frame(index = rep(1:N, M),
                       particle = as.vector(t(pred)))

ggplot(partikel, aes(x = index), colour = "grey") +
    xlab("Time") +
    theme_bw() +
    geom_point(aes(y = particle, color = "Particle"), size=1) +
    geom_line(data = locations, aes(y = true_location, color = "True location"), size=0.8) +
    geom_line(data = locations, aes(y = exp_location, color = "Observation"), size=0.8) +
    scale_color_manual(values=c("firebrick2", "grey", "dodgerblue1"), name = "Legend")

suppressMessages(grid.arrange(hist1, hist2,hist3, hist4, nrow=2))


partic_filter_noweight <- function(xt, variance){
  M<-100
  chi <- c()

  #Initialization
  chi <- runif(100, 0, 100)
  pred <- matrix(vector(length=10000), nrow=100)
  pred[,1] <- chi
  for(t in 2:100){
    chihat <- c()
    w <- c()
    for(m in 1:M){
      model <- c()
      model[1] <- rnorm(1,chi[m], 1)
      model[2] <- rnorm(1,chi[m]+1, 1)
      model[3] <- rnorm(1,chi[m]+2, 1)
      xtm <- model[sample(1:3,1)]
      w <- rep(1,100)
```

```
      chihat[m] <- xtm
    }
    probs <- cumsum(w/sum(w))
    for(m in 1:M){
        chi[m] <- chihat[findInterval(runif(1), probs)+1]
    }
    pred[,t] <- chi

  }
  return(pred)
}

set.seed(12345)

var <- 1

pred <- partic_filter_noweight(dataVar1$obs, var)
data <- data.frame(obs =pred , real = dataVar1$real,realObs = dataVar1$obs, step = 1:100)

hist1 <- ggplot(data)+
  geom_histogram(aes(obs.1,))+
  geom_vline(aes(xintercept=data$realObs[1],colour="observation"))+
  geom_vline( aes(xintercept = data$real[1],colour="true value"))
hist2 <- ggplot(data)+
  geom_histogram(aes(obs.100))+
  geom_vline(aes(xintercept=data$realObs[100],colour="observation"))+
  geom_vline( aes(xintercept = data$real[100],colour="true value"))
hist3 <- ggplot(data)+
  geom_histogram(aes(obs.35))+
  geom_vline(aes(xintercept=data$realObs[35],colour="observation"))+
  geom_vline( aes(xintercept = data$real[35],colour="true value"))
hist4 <- ggplot(data)+
  geom_histogram(aes(obs.75))+
  geom_vline(aes(xintercept=data$realObs[75],colour="observation"))+
  geom_vline( aes(xintercept = data$real[75],colour="true value"))

N <- 100
M <- 100
locations <- data.frame(index = 1:N,
                        true_location = data$real,
                        exp_location = data$realObs)

partikel <- data.frame(index = rep(1:N, M),
                       particle = as.vector(t(pred)))

ggplot(partikel, aes(x = index), colour = "grey") +
    xlab("Time") +
    theme_bw() +
    geom_point(aes(y = particle, color = "Particle"), size=0.8) +
    geom_line(data = locations, aes(y = true_location, color = "True location"), size=0.8) +
    geom_line(data = locations, aes(y = exp_location, color = "Observation"), size=0.8) +
    scale_color_manual(values=c("firebrick2", "grey", "dodgerblue1"), name = "Legend")
```

```
suppressMessages(grid.arrange(hist1, hist2,hist3, hist4, nrow=2))
```

Lab 4

Alejandro Garcia, Alessia De Biase, Anna-Katharina Fürgut, Erika Anderskär

17 oktober 2018

Assignment 1 - Implementing GP Regression

In this task we implement our own code for the Gaussian process regression model:

$$y = f(x) + \epsilon,\ \epsilon \sim N(0, \sigma_n^2),\ f \sim GP(0, \kappa(x, x'))$$

where κ is the Kernel funciton.

The algorithm used for the Gaussian Process is from Rasmussen and William's book.

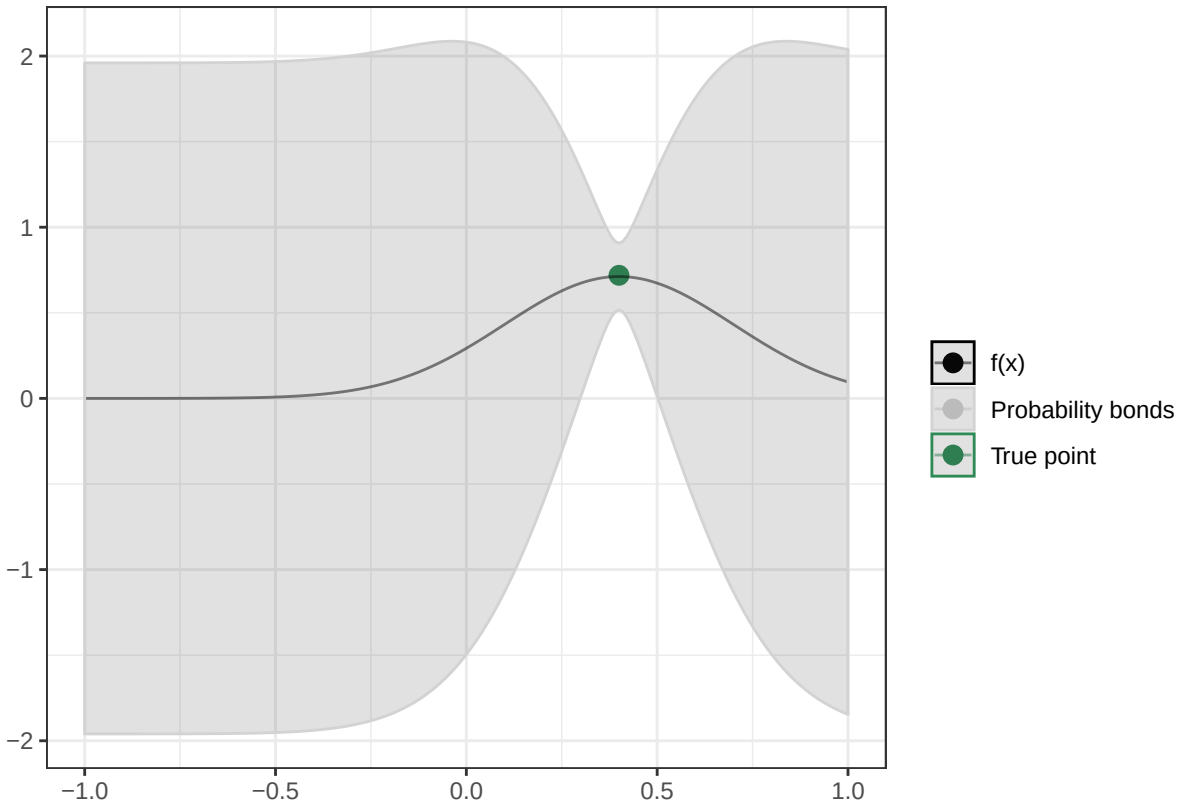
```
library("mvtnorm")
require(ggplot2)
require(kernlab)
require(AtmRay)
require(knitr)

posteriorGP <- function(X, y, Xstar, hyperParam, sigmaNoise){

  # X is input training inputs, vector
  # y is training outputs/targets, vector
  # Xstar is inputs where the posterior is evaluated, vector
  # hyperparam is  the two elements sigmaF and l, vector
  # sigmaNoise is Noise standard deviation

  n = length(X)

  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)
  }


  # Calculate the different covariance matrices
  K_x_x = SquaredExpKernel(X, X, sigmaF = hyperParam[1], l = hyperParam[2])
  K_x_xstar = SquaredExpKernel(X, Xstar, sigmaF = hyperParam[1], l = hyperParam[2])
  K_xstar_xstar = SquaredExpKernel(Xstar, Xstar, sigmaF = hyperParam[1], l = hyperParam[2])


  sI <- sigmaNoise^2 * diag(dim(as.matrix(K_x_x))[1])

  L = t(chol(K_x_x + sI))
  L_trans = (t(L))
```

```
  alpha = solve(L_trans, solve(L, y))
  f_bar_star = t(K_x_xstar) %*% alpha
  v = solve(L, K_x_xstar)
  Vf =  K_xstar_xstar - (t(v)  %*% v)
  marg_lik = (- 0.5 %*% t(y) %*% alpha - sum(diag(log(L)) - n/2 %*% log(2*pi)))

  return(list(f_bar_star = as.vector(f_bar_star), Vf = Vf, marg_lik = marg_lik))

}
```

1.2

In this we task we set the hyperparameters $\sigma_f = 1$ and $l = 0.3$ and upate this prior with a single observation $(x, y) = (0.4, 0.719)$. The noise standard deviation is known to be $\sigma_n = 0.1$.

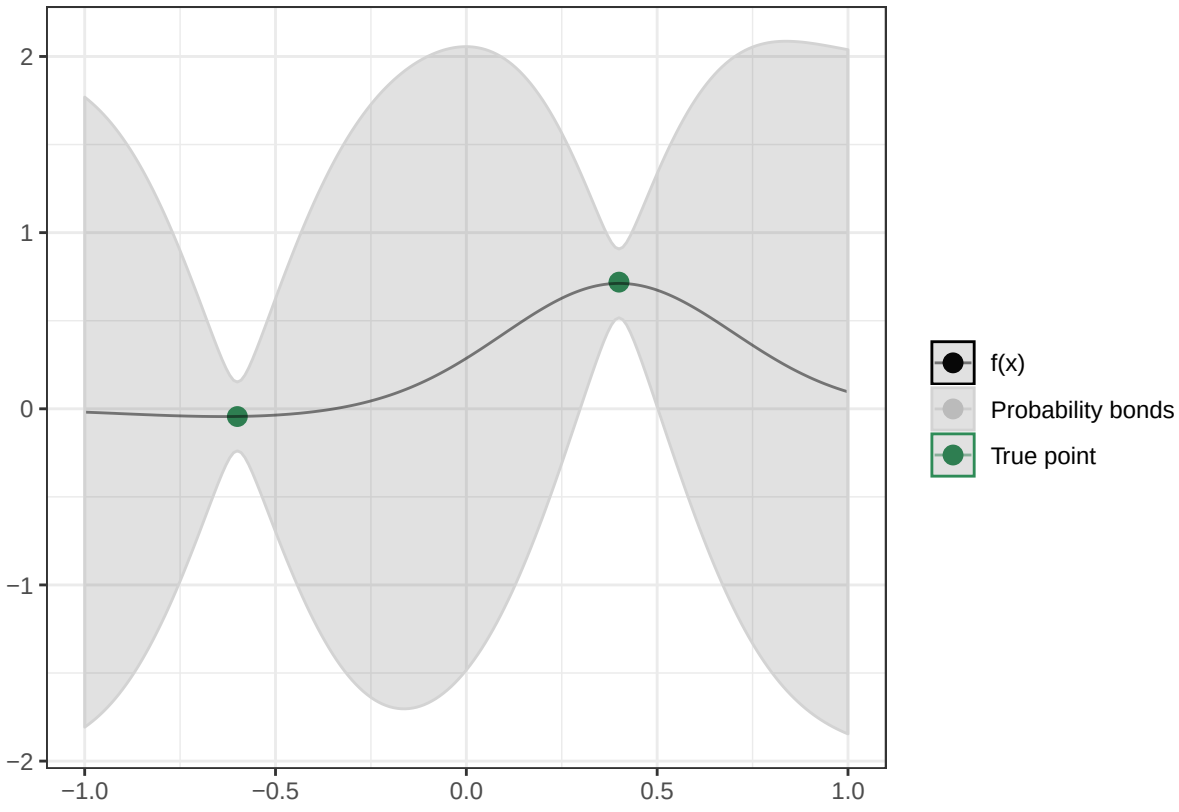
```
hyperParam = c(1, 0.3)
sigmaNoise = 0.1
xGrid = seq(-1, 1, 0.01)

X <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719 ,-0.664)


res = posteriorGP(X[4],y[4],Xstar = xGrid, hyperParam, sigmaNoise)

upp <- res$f_bar_star + 1.96*sqrt(diag(res$Vf))
low <- res$f_bar_star - 1.96*sqrt(diag(res$Vf))

ggplot()+
  geom_point(aes(x= X[4], y = y[4], colour = "True point"), size = 3)+
    geom_line(aes(x=xGrid, y =  res$f_bar_star, colour = "f(x)"), alpha = 0.5)+
    geom_ribbon(aes(ymin = low, ymax = upp, x = xGrid, colour = "Probability bonds"), alpha= 0.15)+
  theme_bw()+
  labs(x = "", y = "")+
  scale_colour_manual("", values = c("black", "lightgrey", "seagreen"))
```



1.3

Here we update our posterior with another point $(x, y) = (-0.6, -0.044)$. Since this is the same as updateing our posterior directly with the two points, we just plug those points in our function.

```
res2 = posteriorGP(X[c(4,2)],y[c(4,2)],Xstar = xGrid, hyperParam, sigmaNoise)

upp2 <- res2$f_bar_star + 1.96*sqrt(diag(res2$Vf))
low2 <- res2$f_bar_star - 1.96*sqrt(diag(res2$Vf))

ggplot()+
  geom_point(aes(x= X[c(4,2)], y = y[c(4,2)], colour = "True point"), size = 3)+
    geom_line(aes(x=xGrid, y =  res2$f_bar_star, colour = "f(x)"), alpha = 0.5)+
    geom_ribbon(aes(ymin = low2, ymax = upp2, x = xGrid, colour = "Probability bonds"), alpha= 0.15)+
  theme_bw()+
  labs(x = "", y = "")+
  scale_colour_manual("", values = c("black", "lightgrey", "seagreen"))
```

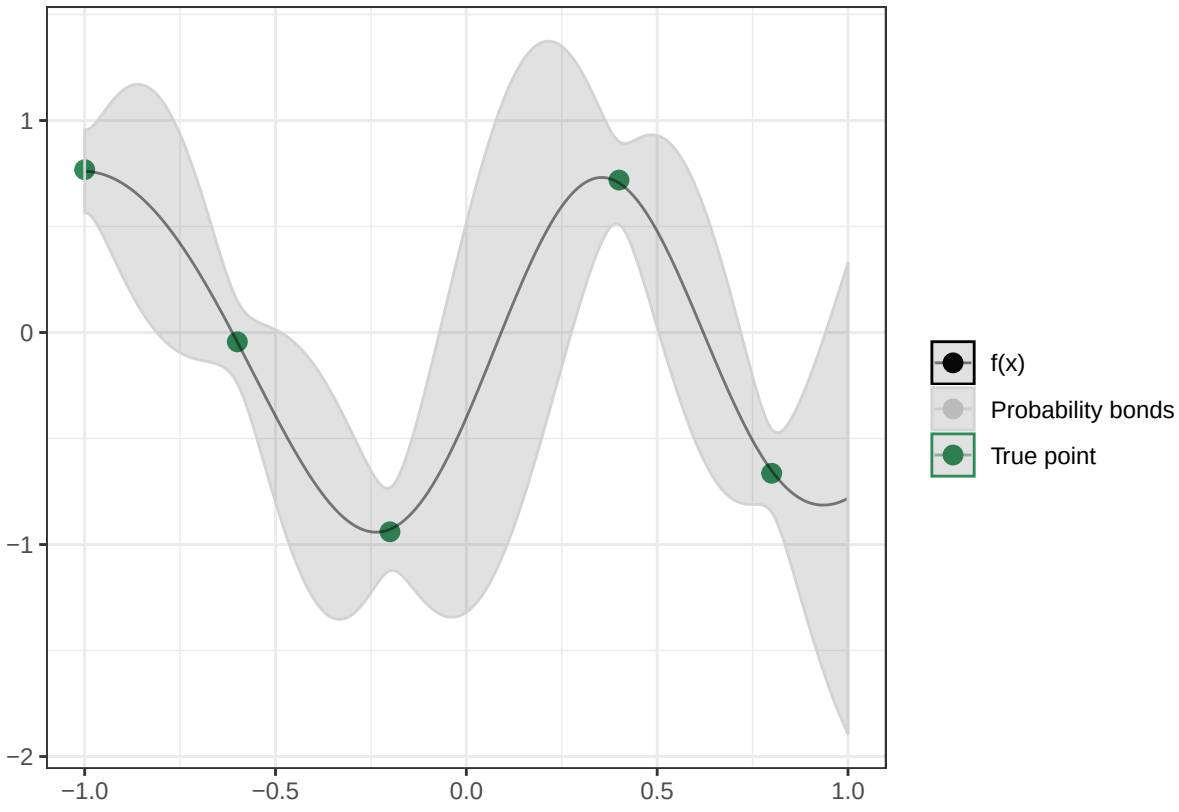


The plot shows the posterior mean based on two observation. The posterior mean seems not to have changed compared to the previous plot, maybe due to that the new observation is equal with the posterior mean in that point. But the probability band around the 2 observations is more narrow than for the rest.

1.4

Here we use all the 5 points in our data.

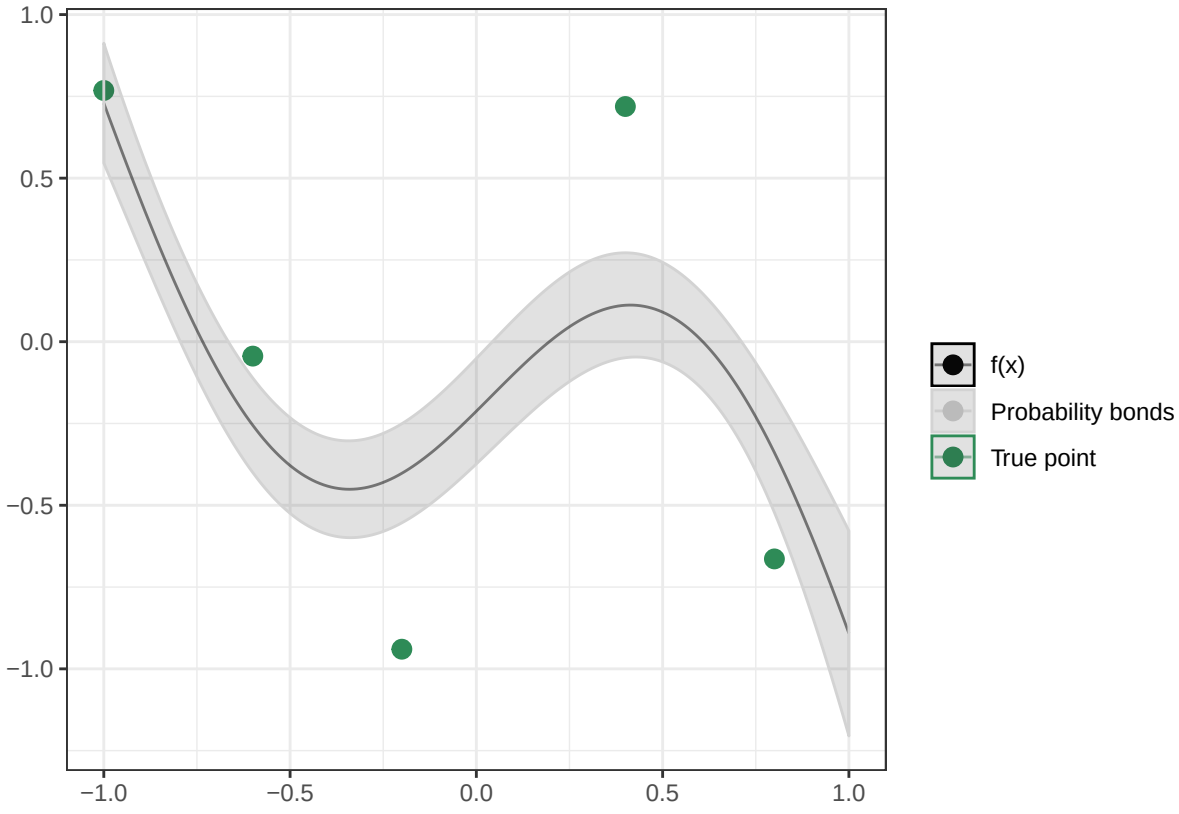
```
res3 = posteriorGP(X,y,Xstar = xGrid, hyperParam, sigmaNoise)

upp3 <- res3$f_bar_star + 1.96*sqrt(diag(res3$Vf))
low3 <- res3$f_bar_star - 1.96*sqrt(diag(res3$Vf))

ggplot()+
  geom_point(aes(x= X, y = y, colour = "True point"), size = 3)+
    geom_line(aes(x=xGrid, y =  res3$f_bar_star, colour = "f(x)"), alpha = 0.5)+
    geom_ribbon(aes(ymin = low3, ymax = upp3, x = xGrid, colour = "Probability bonds"), alpha= 0.15)+
  theme_bw()+
  labs(x = "", y = "")+
  scale_colour_manual("", values = c("black", "lightgrey", "seagreen"))
```



1.5

In this task we use all the data once again, but now we change the length scale to $l = 1$.

```
hyperparam = c(1,1)

res4 = posteriorGP(X,y,Xstar = xGrid, hyperparam, sigmaNoise)

upp4 <- res4$f_bar_star + 1.96*sqrt(diag(res4$Vf))
low4 <- res4$f_bar_star - 1.96*sqrt(diag(res4$Vf))

ggplot()+
  geom_point(aes(x= X, y = y, colour = "True point"), size = 3)+
    geom_line(aes(x=xGrid, y =  res4$f_bar_star, colour = "f(x)"), alpha = 0.5)+
    geom_ribbon(aes(ymin = low4, ymax = upp4, x = xGrid, colour = "Probability bonds"), alpha= 0.15)+
  theme_bw()+
  labs(x = "", y = "")+
  scale_colour_manual("", values = c("black", "lightgrey", "seagreen"))
```



The length scale determines the smoothness. When the length scale is higher we put more weight on data points that are further away. This leads to a smoother GP and it does't capture the behaviour of the data. We can also see that the variance is smaller indicating that we are more certain about the location of the posterior even though the posterior mean doesn't capure the true points.

Assignment 2 GP regression with kernlab

In this task we use data with daily mean temperature in Tullinge, Stockholm.

2.1

Data preprocessing:

```
TempTullinge <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempT

TempTullinge$Day = rep(1:365,6)
TempTullinge$Time = 1:nrow(TempTullinge)

TullingeSubset <- TempTullinge$Time %in% seq(1,2190,5)
TullingeSubset <- TempTullinge[TullingeSubset,]

Time = TullingeSubset$Time
temp = TullingeSubset$temp
```

Here we write a function that generates a squared exponential kernel with the desired hyperparameters as input.

```
# Make own kernel

kernel_maker <- function(sigmaF = 1, ell = 1){

  rval <- function(x1, x2) {
    return(sigmaF^2*exp(-0.5*( (x1-x2)/ell)^2 ))
  }

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


kernel1 <- kernel_maker()

# Evaluating the kernel in (1,2)
```

We evaluate the kernel in the point $x = 1, x' = 2$ and calculate the covariance matrix $K(X, X_*)$.

```
kable(kernel1(1,2))
```

x
0.6065307

```
x = c(1,3,4)
x_star = c(2,3,4)

kable(kernelMatrix(kernel = kernel1, x = x, y = x_star)) #K(x,xstar)
```

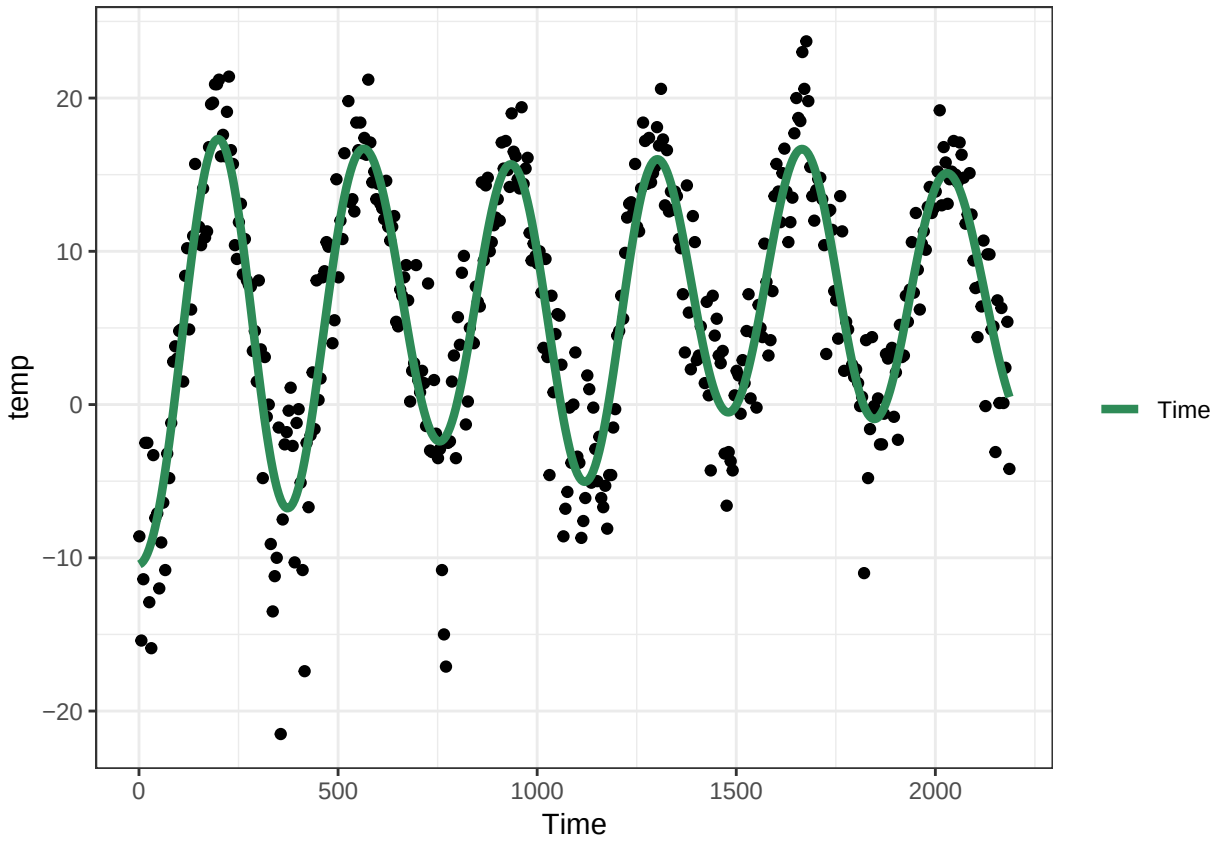
0.6065307	0.1353353	0.0111090
0.6065307	1.0000000	0.6065307
0.1353353	0.6065307	1.0000000

2.2

First we consider a Gaussian Process were Temperature is modeled by Time.

```
fit1 = lm(temp ~ Time + I(Time^2), data = TullingeSubset)

sigmaNoise = var(resid(fit1))

kernel2 = kernel_maker(sigmaF = 20, ell = 0.2)

GPfit <- gausspr(TullingeSubset$Time,
                 TullingeSubset$temp,
                 kernel = kernel2,
                 var = sigmaNoise)

meanPred <- predict(GPfit, TullingeSubset$Time) # Predicting the training data.
```

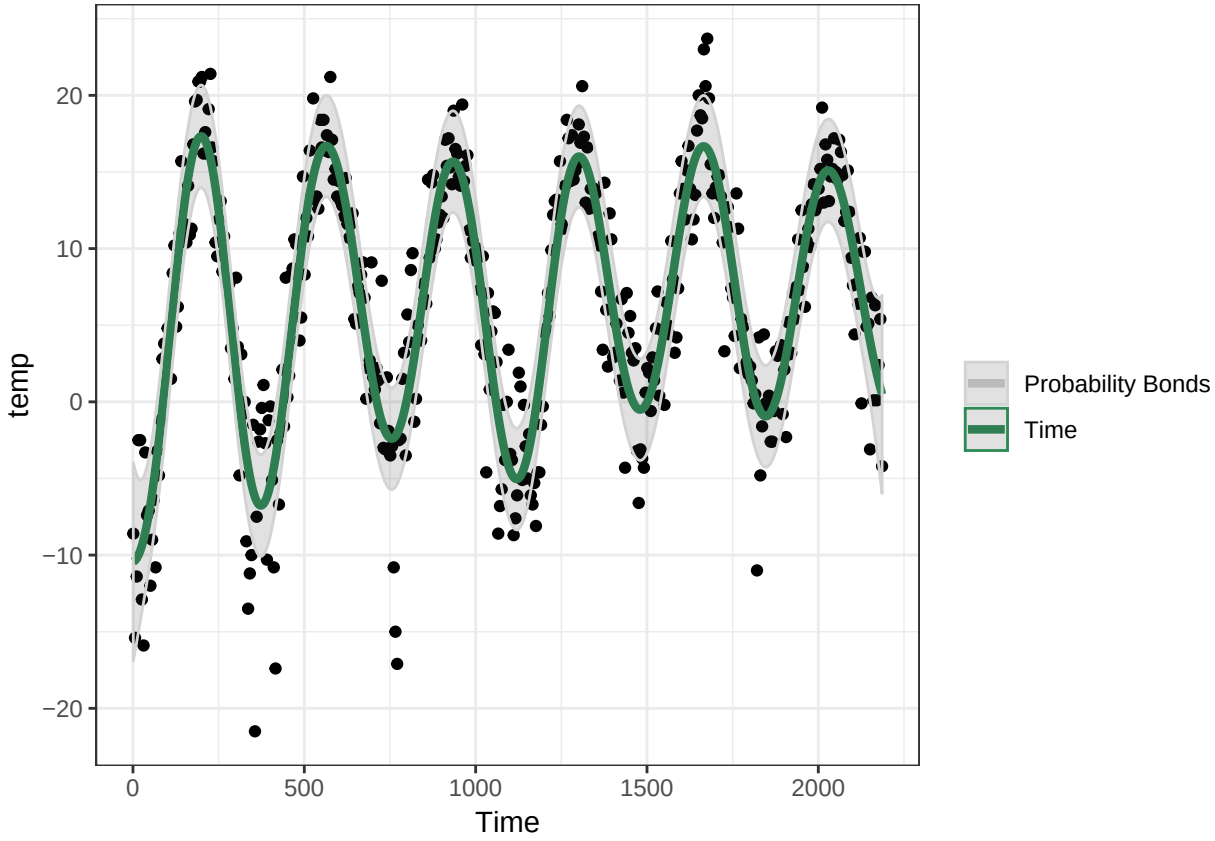
```
TullingeSubset$meanPred = meanPred

p1 = ggplot(data = TullingeSubset)+
  geom_point( aes(x=Time, y=temp))+
  geom_line(aes(x=Time, y=meanPred, colour = "Time"), size = 1.5)+theme_bw()

p1 + scale_colour_manual("", values = "seagreen")
```



2.3

```
temp = scale(TullingeSubset$temp)
time = scale(TullingeSubset$Time)

res23 = posteriorGP(X = time,
            y = temp,
            Xstar = time,
            hyperParam = c(20, 0.2),
            sigmaNoise = sd(resid(fit1)))

upp5 <- predict(GPfit) + 1.96*sqrt(diag(res23$Vf))
low5 <- predict(GPfit) - 1.96*sqrt(diag(res23$Vf))
```

```
p1 + geom_ribbon(aes(ymin = low5, ymax = upp5, x = TullingeSubset$Time, colour = "Probability Bonds"), a
scale_colour_manual("", values = c("lightgrey", "seagreen"))
```

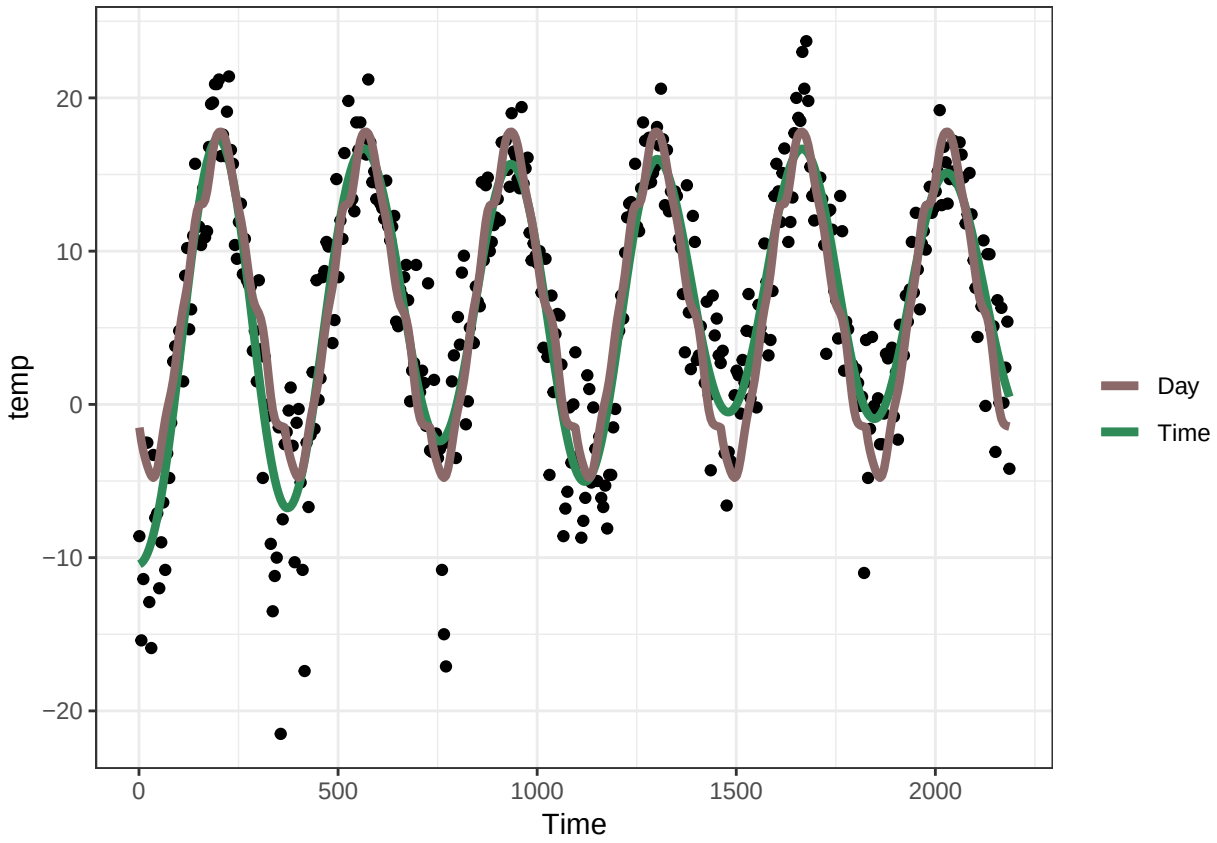


2.4

Now, we consider a model were Temperatur is modeled by Day.

```
fit2 = lm(temp ~ Day + I(Day^2), data = TullingeSubset)

sigmaNoise24 = var(resid(fit2))

kernel3 = kernel_maker(sigmaF = 20, ell = 0.2)

GPfit2 <- gausspr(TullingeSubset$Day,
                  TullingeSubset$temp,
                  kernel = kernel2,
                  var = sigmaNoise)

meanPred2 <- predict(GPfit2, TullingeSubset$Day) # Predicting the training data.


TullingeSubset$meanPred2 = meanPred2

p2 = p1 + geom_line(aes(x=Time, y=meanPred2, colour = "Day"), size = 1.5)+scale_colour_manual("", values
```

```
p2
```

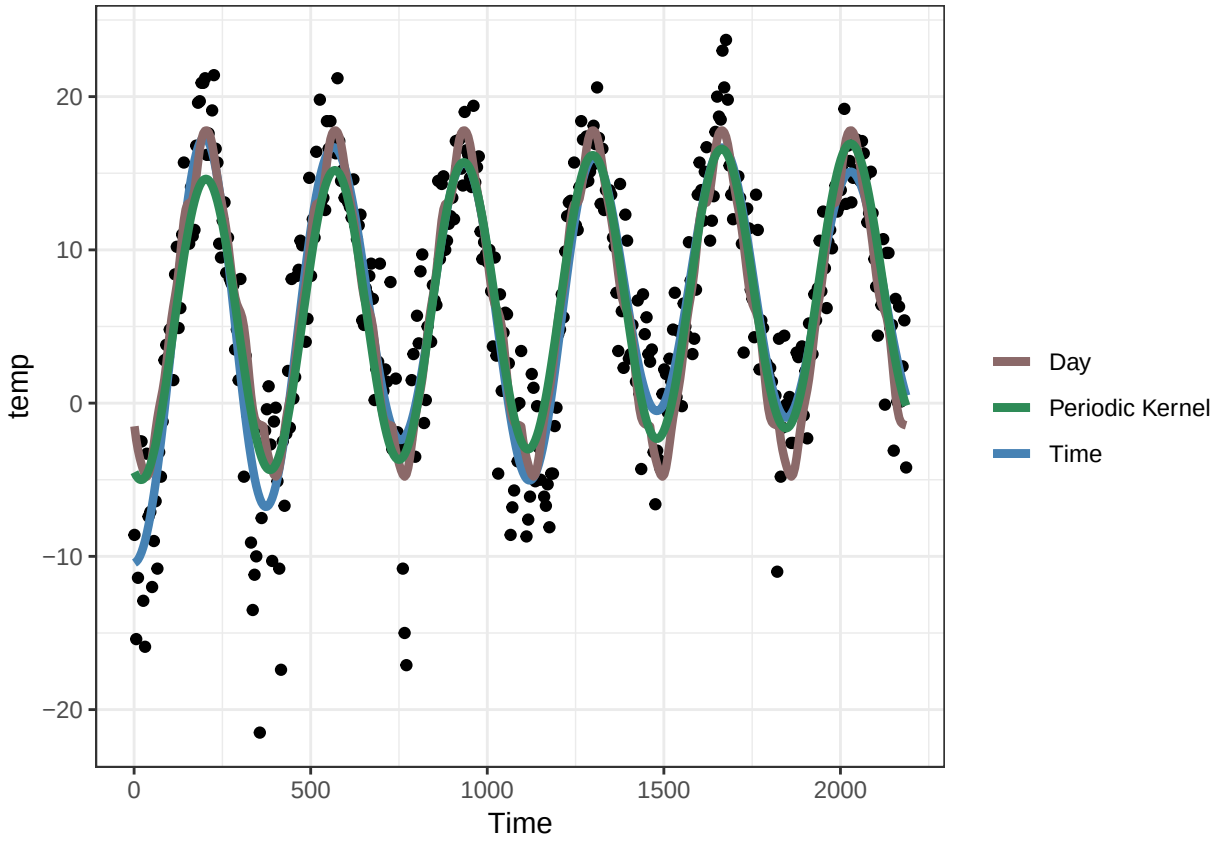


The two models follow a similar pattern with som minor differences in the maximum and minumum from year to year. The Day model captures the seasonal pattern and the Time model captures the longterm trend over time.

2.4

```
periodic_kernel_maker <- function(sigmaF = 1, d, l1 = 1, l2 = 1){

  rval <- function(x1, x2) {

    return(sigmaF^2*exp(-2*sin(pi*abs(x1 - x2)/d)^2/l2^2) * exp(-0.5*abs(x1 - x2)^2/l2^2))
  }

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

kernel3 = periodic_kernel_maker(sigmaF = 20, l1 = 1, l2 = 10, d = 365/sd(Time))

GPfit <- gausspr(TullingeSubset$Time,
                 TullingeSubset$temp,
                 kernel = kernel3,
```

```
                  var = sigmaNoise)

meanPred3 <- predict(GPfit, TullingeSubset$Time) # Predicting the training data.


TullingeSubset$meanPred3 = meanPred3

p2 + geom_line(aes(x=Time, y=meanPred3, colour = "Periodic Kernel"), size = 1.5)+ scale_colour_manual("
```



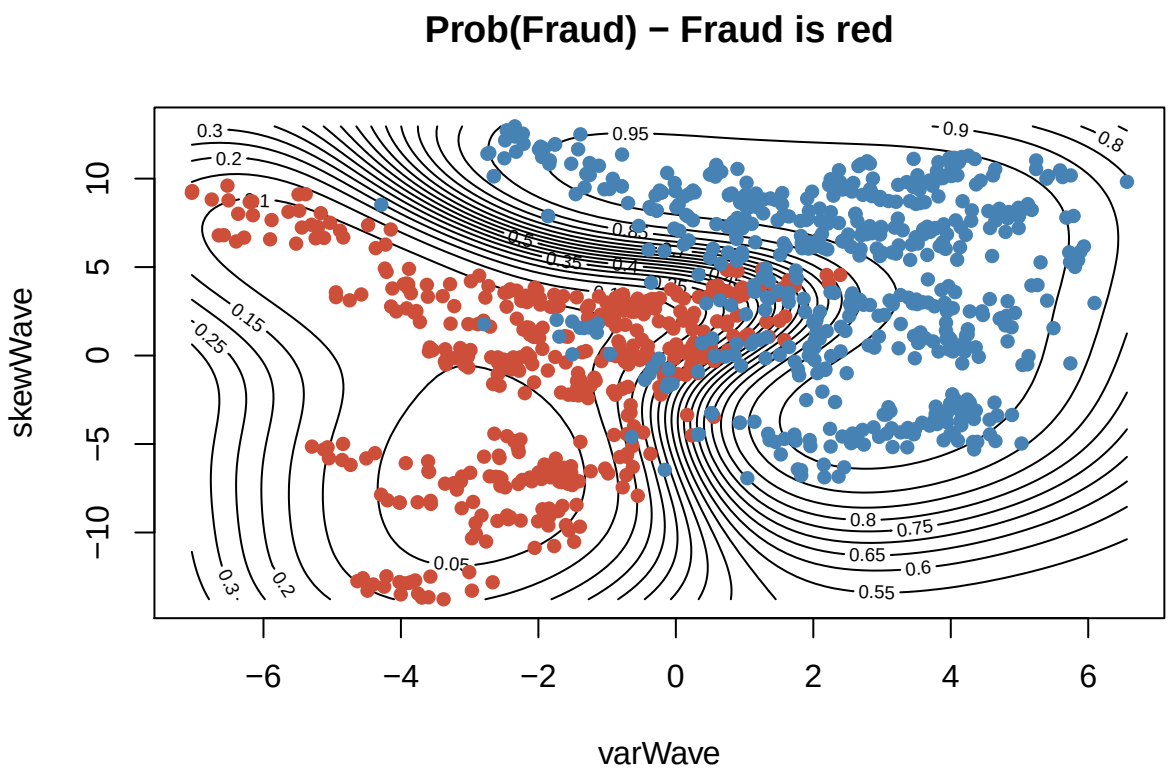
Here we use the general periodic kernel, which has the advantages that it captures both the seasonal pattern and the long term trend over time. As seen in the plot, the periodic kernel lies between the Day and Time model.

3 GP Classification with kernlab

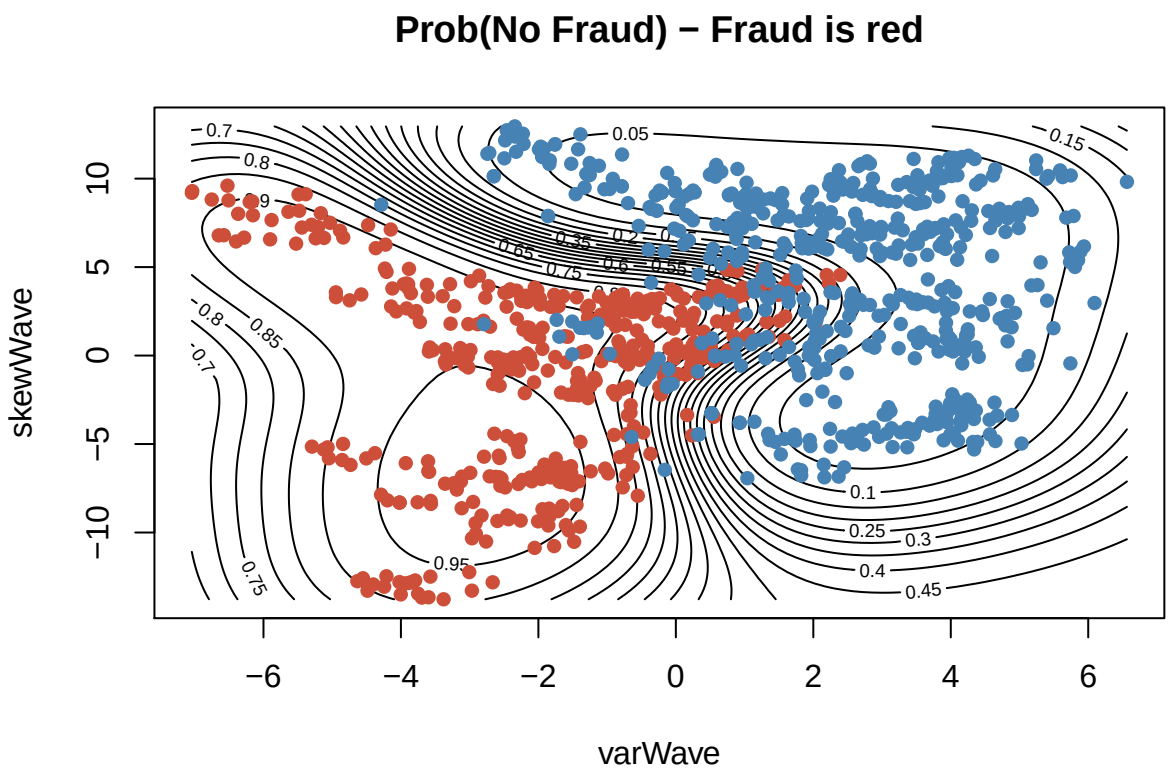
In this task we perform Classification on banknote fraud data. We use 1000 observations as training data and the rest as test.

```
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,]


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

## Using automatic sigma estimation (sigest) for RBF or laplace kernel
probPreds <- predict(GPfitFraud, train[,1:2], type="probabilities")
x1 <- seq(min(train[,1]),max(train[,1]),length=100)
x2 <- seq(min(train[,2]),max(train[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(train)[1:2]
probPreds <- predict(GPfitFraud, gridPoints, type="probabilities")

contour(x1,x2,t(matrix(probPreds[,1],100)), 20,
        xlab = "varWave", ylab = "skewWave",
        main = 'Prob(Fraud) - Fraud is red')
points(train[train[,5]== 1,"varWave"],train[train[,5]== 1,"skewWave"],col="tomato3", pch = 16)
points(train[train[,5]== 0,"varWave"],train[train[,5]== 0,"skewWave"],col="steelblue", pch = 16)
```



```
contour(x1,x2,t(matrix(probPreds[,2],100)), 20,
        xlab = "varWave", ylab = "skewWave",
        main = 'Prob(No Fraud) - Fraud is red')
points(train[train[,5]== 1,"varWave"],train[train[,5]== 1,"skewWave"],col="tomato3", pch = 16)
points(train[train[,5]== 0,"varWave"],train[train[,5]== 0,"skewWave"],col="steelblue", pch = 16)
```




```
# Confusion matrix
kable(table(predict(GPfitFraud,train[,1:2]), train[,5]), caption = "Confusion matrix for train data") #
```

Table 3: Confusion matrix for train data

	0	1
0	512	24
1	44	420

```
kable(sum(predict(GPfitFraud,train[,1:2]) == train[,5])/nrow(train), caption = "Accuracy for train data
```

Table 4: Accuracy for train data

x
0.932

3.2

```
#table(predict(GPfitIris,train[,1:2]), train[,5]) # confusion matrix

kable(sum(predict(GPfitFraud,test[,1:2]) == test[,5])/nrow(test), caption = "Accuracy for model with tw
```

Table 5: Accuracy for model with two covariates

x
0.9354839

3.3

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

## Using automatic sigma estimation (sigest) for RBF or laplace kernel
kable(sum(predict(GPfitFraud_Full_Model,test[,1:4]) == test[,5])/nrow(test), caption ="Accuracy for mod
```

Table 6: Accuracy for model with all covariates

x
0.9973118

When all the covariates are used in the model the accuacy for the test data is 99.7 % and when only two covariates are used the accuracy for the test data is 94.5 %