Lab 1

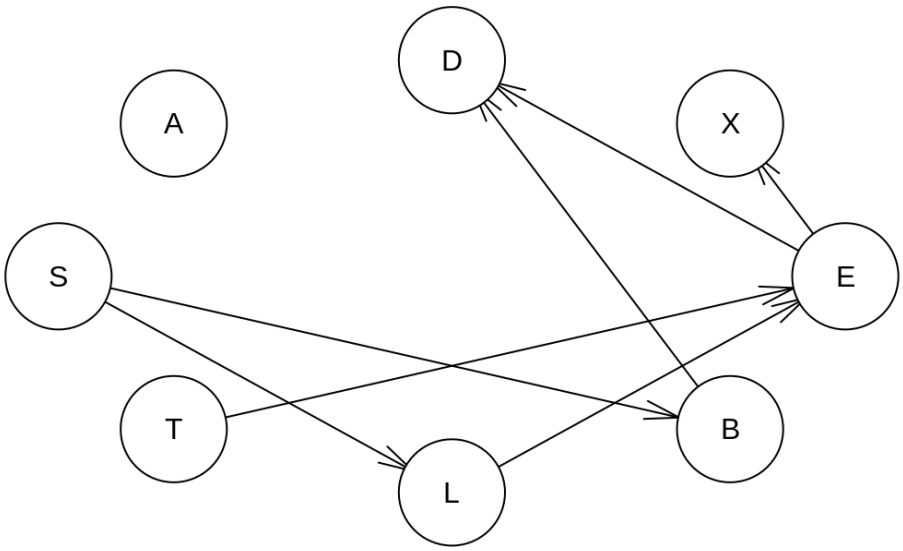
Emil Luusua

17 September 2019

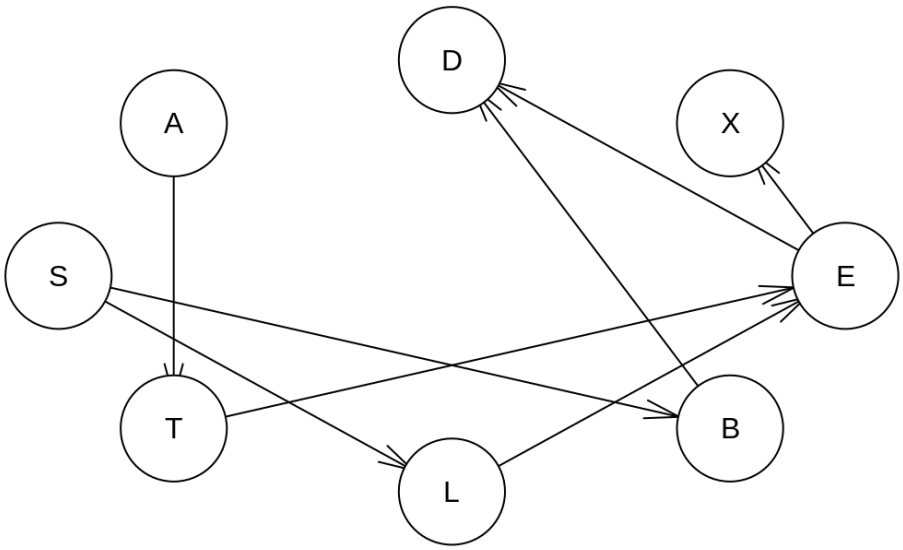
Question 1

Two runs of the hill-climbing algorithm can return non-equivalent BN structures. This can easily be shown by modifying the iss score parameter, which controls the imaginary sample size of the BDeu score.

```
dag <- hc(asia, score = 'bde', iss = 1)
plot(dag)
```




```
plot(hc(asia, score = 'bde', iss = 2))
```



Note the additional arc between from ‘A’ to ‘T’ in the second plot. Differences in the resulting graph such as this can occur when using different scorings since it will influence how different aspects are weighted, in the case of iss the weight given to the prior is altered.

Question 2

We start by splitting the data into train and test sets. Also extract labels S from the test set.

```
set.seed(1)
sample_size <- floor(0.8 * nrow(asia))
train_indices <- sample(seq_len(nrow(asia)), size = sample_size)

asia_train <- asia[train_indices, ]
asia_test <- asia[-train_indices, ]
labels <- asia_test['S']
asia_test$S <- NULL
```

We use the structure learnt with by Hill Climbing in question 1 and learn the parameters of the model with Maximum Likelihood parameter estimation since we are not missing any data.

```
fit <- bn.fit(dag, asia_train)
```

We perform classification of the test set using exact inference in gRain and display the results in a confusion matrix using caret.

```
net <- compile(as.grain(fit))
data <- list()

for (i in seq_len(nrow(asia_test))) {
  evidence <- list()

  for (col in colnames(asia_test)) {
    evidence <- append(evidence, as.character(asia_test[i, col]))
  }

  posterior <- querygrain(setFinding(net, nodes=colnames(asia_test), states=
evidence))$S

  if (posterior['yes'] > posterior['no']) {
    # Classify as positive
    data <- append(data, 'yes')

  } else {
    # Classify as negative
    data <- append(data, 'no')
  }
}
confusionMatrix(factor(unlist(data)), factor(unlist(labels)))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  334 131
##        yes 136 399
##
##                Accuracy : 0.733
##                  95% CI : (0.7044, 0.7602)
##     No Information Rate : 0.53
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4637
##
##  Mcnemar's Test P-Value : 0.8066
##
##             Sensitivity : 0.7106
##             Specificity : 0.7528
##          Pos Pred Value : 0.7183
##          Neg Pred Value : 0.7458
##              Prevalence : 0.4700
##          Detection Rate : 0.3340
##    Detection Prevalence : 0.4650
##       Balanced Accuracy : 0.7317
##
##        'Positive' Class : no
##
```

We then compare the results when using the true structure, with parameters still learned from the training data. This is shown to be exactly the same.

```
true_dag <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
true_fit <- bn.fit(true_dag, asia_train)

true_net <- compile(as.grain(true_fit))
data <- list()

for (i in seq_len(nrow(asia_test))) {
  evidence <- list()

  for (col in colnames(asia_test)) {
    evidence <- append(evidence, as.character(asia_test[i, col]))
  }

  posterior <- querygrain(setFinding(true_net, nodes=colnames(asia_test), st
ates=evidence))$S

  if (posterior['yes'] > posterior['no']) {
    # Classify as positive
    data <- append(data, 'yes')

  } else {
    # Classify as negative
    data <- append(data, 'no')
  }
}
confusionMatrix(factor(unlist(data)), factor(unlist(labels)))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  334 131
##        yes 136 399
##
##                Accuracy : 0.733
##                  95% CI : (0.7044, 0.7602)
##     No Information Rate : 0.53
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4637
##
##  Mcnemar's Test P-Value : 0.8066
##
##             Sensitivity : 0.7106
##             Specificity : 0.7528
##          Pos Pred Value : 0.7183
##          Neg Pred Value : 0.7458
##              Prevalence : 0.4700
##          Detection Rate : 0.3340
##    Detection Prevalence : 0.4650
##       Balanced Accuracy : 0.7317
##
##        'Positive' Class : no
##
```

Question 3

What is the Markov blanket of S?

```
mb(dag, 'S')
```

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

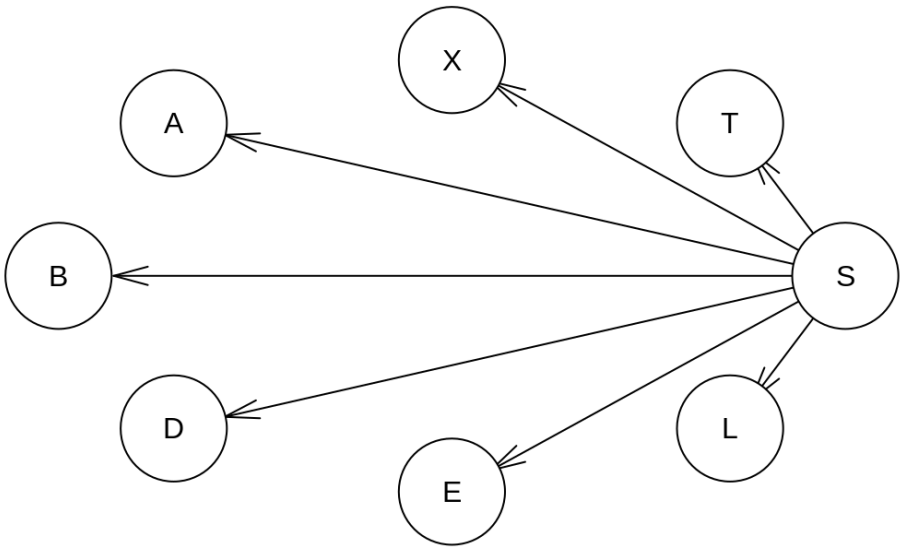
We thus perform the same procedure for classification as previously, only the evidence is altered such that it contains observations for L and B. Once again the results are identical.

```
observations <- c('L', 'B')
data <- list()

for (i in seq_len(nrow(asia_test))) {
  evidence <- list()

  for (col in observations) {
    evidence <- append(evidence, as.character(asia_test[i, col]))
  }

  posterior <- querygrain(setFinding(net, nodes=observations, states=evidenc
e))$S

  if (posterior['yes'] > posterior['no']) {
    # Classify as positive
    data <- append(data, 'yes')

  } else {
    # Classify as negative
    data <- append(data, 'no')
  }
}
confusionMatrix(factor(unlist(data)), factor(unlist(labels)))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  334 131
##        yes 136 399
##
##                Accuracy : 0.733
##                  95% CI : (0.7044, 0.7602)
##     No Information Rate : 0.53
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4637
##
##  Mcnemar's Test P-Value : 0.8066
##
##             Sensitivity : 0.7106
##             Specificity : 0.7528
##          Pos Pred Value : 0.7183
##          Neg Pred Value : 0.7458
##              Prevalence : 0.4700
##          Detection Rate : 0.3340
##    Detection Prevalence : 0.4650
##       Balanced Accuracy : 0.7317
##
##        'Positive' Class : no
##
```

A Naïve Bayes classifier can be represented by a Bayesian network where the prediction variable is the sole parent of all other variables. This will result in the graph representing the same conditional independence as is assumed for a Naïve Bayes model.

```
naive_dag <- model2network("[S][A|S][T|S][L|S][B|S][D|S][E|S][X|S]")
naive_fit <- bn.fit(naive_dag, asia_train)
plot(naive_dag)
```



Now once again we perform classification of S by observing all other variables, and once again the same results are achieved.

```
naive_net <- compile(as.grain(fit))
data <- list()

for (i in seq_len(nrow(asia_test))) {
  evidence <- list()

  for (col in colnames(asia_test)) {
    evidence <- append(evidence, as.character(asia_test[i, col]))
  }

  posterior <- querygrain(setFinding(naive_net, nodes=colnames(asia_test), s
tates=evidence))$S

  if (posterior['yes'] > posterior['no']) {
    # Classify as positive
    data <- append(data, 'yes')

  } else {
    # Classify as negative
    data <- append(data, 'no')
  }
}
confusionMatrix(factor(unlist(data)), factor(unlist(labels)))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  334 131
##        yes 136 399
##
##                Accuracy : 0.733
##                  95% CI : (0.7044, 0.7602)
##     No Information Rate : 0.53
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4637
##
##  Mcnemar's Test P-Value : 0.8066
##
##             Sensitivity : 0.7106
##             Specificity : 0.7528
##          Pos Pred Value : 0.7183
##          Neg Pred Value : 0.7458
##              Prevalence : 0.4700
##          Detection Rate : 0.3340
##    Detection Prevalence : 0.4650
##       Balanced Accuracy : 0.7317
##
##        'Positive' Class : no
##
```

Question 5

So throughout all experiments, the same results have been achieved using four different models of the data. How can this possibly make sense?

The first three cases are quite clear, since S will have the same Markov blanket in all these graphs (L and B). As these variables are observed in each case, the S node will be independent from all other nodes. As such observing the other nodes or modeling the independencies among them will not make a difference to the classification of S.

The Naïve Bayes network is quite different though, and could very well yield a different result since ALL variables will be taken into account when performing the classification in comparison to only L and B in earlier cases. The result could be better since even if the other nodes are deemed independent in the Bayes Networks, there might be a small correlation with S that could still benefit classification but was not considered worthy of the added model complexity. Or it could make the results worse by introducing false dependencies making the predictions “noisier” because of the relatively small dataset. In this case though, it seems that L and B dominate the decision made by the Naive Bayes model as well resulting in the exact same predictions.

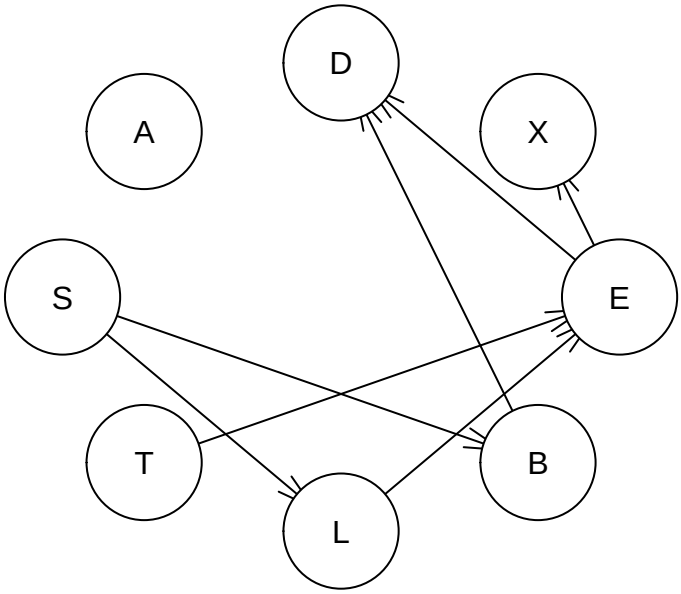
Group Report

krisi211, marpe902, erisv795, emilu795

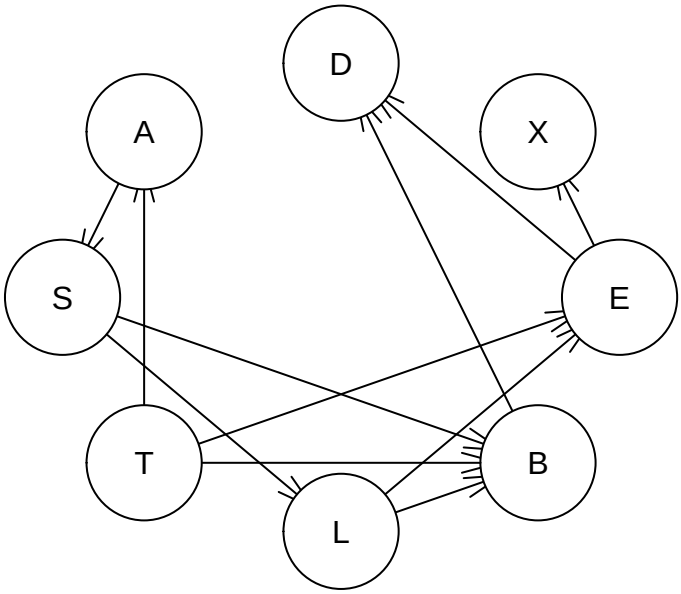
2019-09-19

Assigment 1

No random restarts, bic score

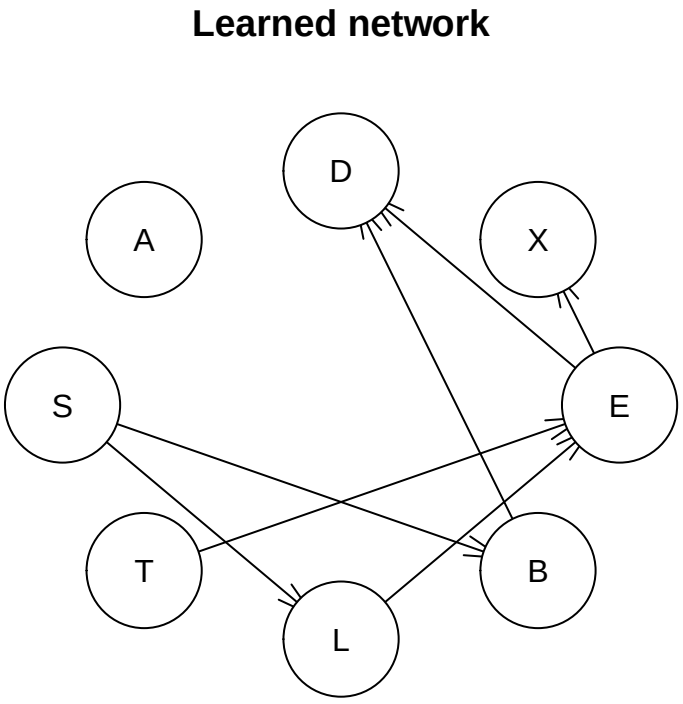


10 random restarts, aic score



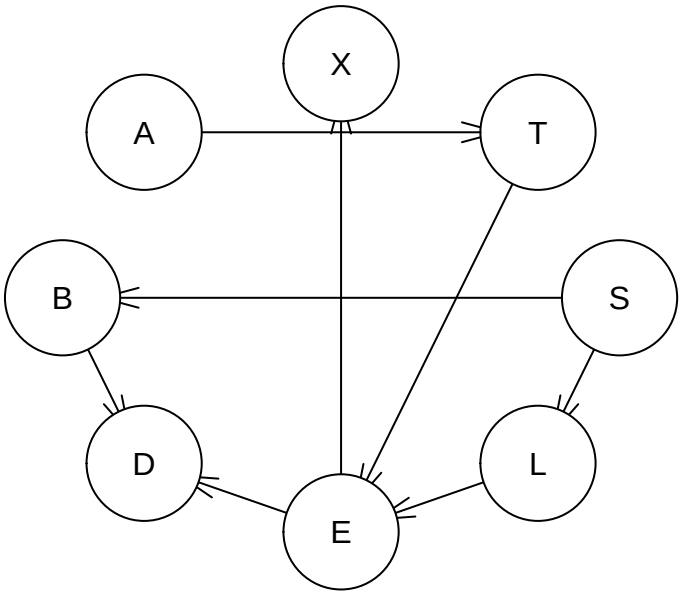
As we can see, the two networks are not equivalent since the have different number of arcs. This is because the hill climbing algorithm cannot guarantee optimality, it can get stuck in a local maxima, and this can yield different networks if it get stuck in a different local maxima after a restart for example. The different scoring options might give different local maxima as well. So a local maxima with the 'bic' score might is different from the local maxima with the 'aic' score, since the value different information.

Assignment 2




```
##            Actual
## Prediction  no yes
##         no  341 119
##         yes 157 383
```

```
## [1] "Classification rate:  0.724"
```




```
##           Actual
## Prediction  no yes
##        no  341 119
##        yes 157 383
```

```
## [1] "Classification rate:  0.724"
```

Assignment 3

Confusion matrix for the learned network when using the markov blanket:

```
##           Actual
## Prediction  no yes
##        no  341 119
##        yes 157 383
```

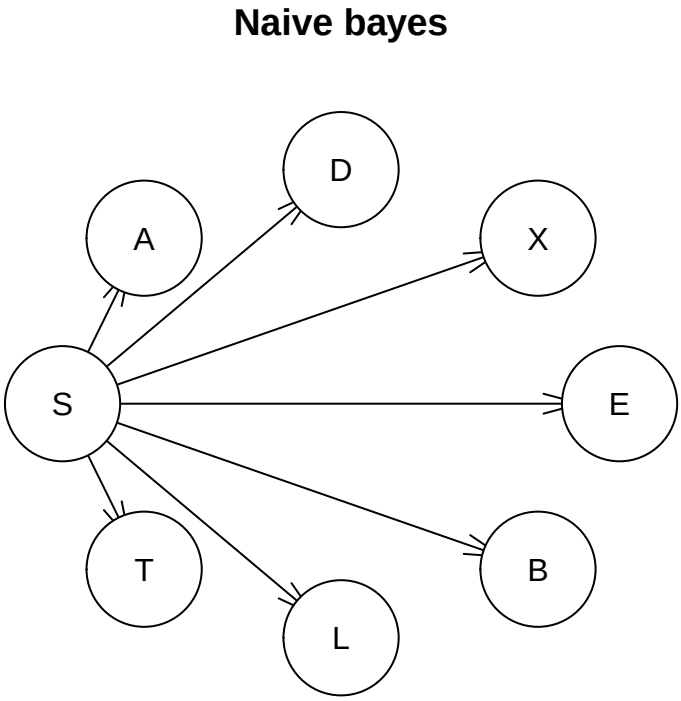
```
## [1] 0.724
```

Confusion matrix for the true network when using the markov blanket:

```
##           Actual
## Prediction  no yes
##        no  341 119
##        yes 157 383
```

```
## [1] 0.724
```

Assigment 4




```
##           Actual
## Prediction  no yes
##        no  374 181
##        yes 124 321
```

```
## [1] 0.695
```

Confusion matrix for the true network from assignment 2:

```
##           Actual
## Prediction  no yes
##        no  341 119
##        yes 157 383
```

```
## [1] 0.724
```

Assignment 5

In the true network and our learned network, when predicting on 'S', the nodes 'B' and 'L' blocks all other information when observed, meaning only these two nodes will have an impact on 'S'. As it turns out, these two are also the markov blanket for both networks. This is why we get the same results for the two networks in both assignment 2 and 3. In assignment 4, we used a naive bayes classifier which assumes independency between all other nodes. This assumption is wrong, as we can see in the true network, this is why we get a slightly worse classification rate.

One in out group got the same result in assignment 4 as in assignment 2 and 3. This could be explained by the fact that even though the markov blanket in the Na??ve Bayes network is different from the one in assignments 2 and 3, it still contains all the information necessary for predicting S (the nodes B and L). It does take "unnecessary" nodes into account as well, but since they are in fact independent this should not

have a large effect on results. Thus making it possible for the same results to be achieved with this model as well.

Appendix - Code

```
###### Init ######
library(bnlearn)
library(gRain)
data('asia')

###### Functions ######
predict.bn = function(junction.tree,data,target_nodes, evidence_nodes)
{
  predictions = rep(0,dim(data)[1])
  for (i in 1:dim(data)[1]) {
    states = NULL
    for (node in evidence_nodes) {
      states[node] = ifelse(data[i,node]=="yes","yes","no")
    }
    evidence = setEvidence(junction.tree,
                           nodes = evidence_nodes,
                           states = states)
    prob = querygrain(evidence,
                      nodes = target_nodes)
    predictions[i] = ifelse(prob$S["yes"] >= prob$S["no"],"yes","no")
  }
  return(predictions)
}

###### Exercise 1 ######
set.seed(1234)
bn = hc(x = asia)
bn_with_restart = hc(x = asia, restart=10, score = 'aic')
plot(bn)
plot(bn_with_restart)
all.equal(bn,bn_with_restart)


###### Exercise 2 ######
smp_size = dim(asia)[1]
set.seed(123)
id = sample(1:smp_size,floor(smp_size*0.8))
train = asia[id,]
test = asia[-id,]

bn = hc(x=train)
fitted.bn = bn.fit(bn,data=train)
junction.tree = compile(as.grain(fitted.bn)) #RIP

true.bn = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
true.fitted.bn = bn.fit(true.bn,data=train)
true.junction.tree=compile(as.grain(true.fitted.bn))
```

```
nodes = colnames(asia)
evidence_nodes = nodes[nodes != "S"]
target_nodes = nodes[nodes == "S"]

prediction = predict.bn(junction.tree = junction.tree,
                        data = test,
                        target_nodes = target_nodes,
                        evidence_nodes = evidence_nodes)
true.prediction = predict.bn(junction.tree =  true.junction.tree,
                             data = test,
                             target_nodes = target_nodes,
                             evidence_nodes = evidence_nodes)
print(table(prediction,test$S))
print(table(true.prediction,test$S))

###### Exercise 3 ######
markov.blanket = mb(fitted.bn,"S")
true.markov.blanket = mb(true.fitted.bn,"S")
prediction.mb = predict.bn(junction.tree = junction.tree,
                      data = test,
                      target_nodes = target_nodes,
                      evidence_nodes = markov.blanket)
true.prediction.mb = predict.bn(junction.tree = true.junction.tree,
                      data = test,
                      target_nodes = target_nodes,
                      evidence_nodes = true.markov.blanket)
print(table(prediction.mb,test$S))
print(table(true.prediction.mb,test$S))

###### Exercise 4 ######
naive.bayes.dag = empty.graph(nodes = nodes)
naive.bayes.dag = set.arc(naive.bayes.dag, from = "S", to = "A")
naive.bayes.dag = set.arc(naive.bayes.dag, from = "S", to = "D")
naive.bayes.dag = set.arc(naive.bayes.dag, from = "S", to = "X")
naive.bayes.dag = set.arc(naive.bayes.dag, from = "S", to = "E")
naive.bayes.dag = set.arc(naive.bayes.dag, from = "S", to = "B")
naive.bayes.dag = set.arc(naive.bayes.dag, from = "S", to = "L")
naive.bayes.dag = set.arc(naive.bayes.dag, from = "S", to = "T")
plot(naive.bayes.dag)
nbd.fit = bn.fit(naive.bayes.dag,data=train)
junc.tree = compile(as.grain(nbd.fit))

predict.nbd = predict.bn(junc.tree, test, target_nodes, evidence_nodes)
print(table(predict.nbd,test$S))
print(table(true.prediction,test$S))
```

Lab 2

Emil Luusua

30 September 2019

Question 1

We create a Hidden Markov Model (HMM) with a hidden state Z^t that takes values $z_0 - z_9$ which represent the sector where the robot is at time step t, and an observed random variable X^t that takes values $x_0 - x_9$ which represent the reading of the tracking device at time step t.

The defining characteristics of the HMM is its initial model, transition model as well as emission model. In this case we use a uniform distribution for the initial model, the transition probabilities are 0.5 to stay in the current state, and 0.5 to move to the next one, and the emission probabilities are 0.2 to observe any of the states within 2 steps of the actual one. These matrices are manually defined as follows:

```
states <- c('z0', 'z1', 'z2', 'z3', 'z4', 'z5', 'z6', 'z7', 'z8', 'z9')
symbols <- c('x0', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9')
transProbs <- matrix(c(
  c(0.5, 0, 0, 0, 0, 0, 0, 0, 0, 0.5),
  c(0.5, 0.5, 0, 0, 0, 0, 0, 0, 0, 0),
  c(0, 0.5, 0.5, 0, 0, 0, 0, 0, 0, 0),
  c(0, 0, 0.5, 0.5, 0, 0, 0, 0, 0, 0),
  c(0, 0, 0, 0.5, 0.5, 0, 0, 0, 0, 0),
  c(0, 0, 0, 0, 0.5, 0.5, 0, 0, 0, 0),
  c(0, 0, 0, 0, 0, 0.5, 0.5, 0, 0, 0),
  c(0, 0, 0, 0, 0, 0, 0.5, 0.5, 0, 0),
  c(0, 0, 0, 0, 0, 0, 0, 0.5, 0.5, 0),
  c(0, 0, 0, 0, 0, 0, 0, 0, 0.5, 0.5)), 10)
emissionProbs <- matrix(c(
  c(0.2, 0.2, 0.2, 0, 0, 0, 0, 0, 0.2, 0.2),
  c(0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0, 0, 0.2),
  c(0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0, 0),
  c(0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0),
  c(0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0),
  c(0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0),
  c(0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0),
  c(0, 0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2),
  c(0.2, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2),
  c(0.2, 0.2, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2)), 10)

hmm <- initHMM(states, symbols, transProbs = transProbs, emissionProbs = emissionProbs)
```

Question 2

We run a simulation of the HMM lasting for 100 time steps, producing the following robot path and observed symbols:

```
simulation <- simHMM(hmm, 100)
print(simulation)
```

```
## $states
##   [1] "z1" "z1" "z1" "z2" "z2" "z2" "z2" "z3" "z3" "z3" "z3" "z4" "z5" "z5"
##  [15] "z5" "z5" "z5" "z6" "z7" "z7" "z7" "z8" "z9" "z9" "z0" "z1" "z1" "z2"
```

```
##  [29] "z2" "z3" "z3" "z4" "z4" "z4" "z4" "z4" "z4" "z4" "z5" "z5" "z6" "z7"
##  [43] "z8" "z9" "z9" "z0" "z1" "z2" "z2" "z3" "z3" "z3" "z3" "z3" "z4" "z4"
##  [57] "z4" "z4" "z4" "z4" "z4" "z5" "z6" "z7" "z7" "z7" "z7" "z8" "z9" "z9"
##  [71] "z0" "z0" "z0" "z1" "z2" "z3" "z4" "z4" "z5" "z6" "z7" "z8" "z9" "z0"
##  [85] "z0" "z1" "z2" "z3" "z3" "z3" "z4" "z5" "z5" "z5" "z6" "z6" "z7" "z7"
##  [99] "z7" "z7"
##
## $observation
##   [1] "x0" "x2" "x0" "x2" "x4" "x4" "x2" "x3" "x3" "x4" "x1" "x2" "x3" "x7"
##  [15] "x7" "x4" "x4" "x8" "x8" "x5" "x8" "x7" "x8" "x8" "x2" "x9" "x0" "x4"
##  [29] "x4" "x4" "x5" "x2" "x5" "x6" "x6" "x5" "x5" "x3" "x5" "x4" "x6" "x5"
##  [43] "x0" "x0" "x7" "x0" "x0" "x2" "x0" "x4" "x2" "x1" "x3" "x3" "x2" "x3"
##  [57] "x3" "x6" "x4" "x2" "x4" "x4" "x7" "x6" "x9" "x9" "x5" "x7" "x1" "x0"
##  [71] "x2" "x1" "x0" "x9" "x2" "x3" "x2" "x4" "x7" "x8" "x8" "x6" "x9" "x1"
##  [85] "x2" "x0" "x0" "x2" "x5" "x2" "x2" "x5" "x3" "x7" "x6" "x6" "x9" "x5"
##  [99] "x7" "x6"
```

Question 3

Only using the observations x^t, the filtered and smoothed distributions can be calculated as

$$p(Z^t|x^{0:t}) = \frac{\alpha(Z^t)}{\sum_{z^t}(\alpha(z^t))}$$

$$p(Z^t|x^{0:T}) = \frac{\alpha(Z^t)\beta(Z^t)}{\sum_{z^t}(\alpha(z^t)\beta(z^t))}$$

where α is computed with the forward function and β is computed with the backward function of the R package hmm. The package also provides the Viterbi algorithm, which is used to compute the most probable path.

```
observations <- simulation$observation

alpha <- exp(forward(hmm, observations))
beta <- exp(backward(hmm, observations))

filtered <- prop.table(alpha, 2)
smoothed <- prop.table(alpha*beta, 2)
most_probable_path <- viterbi(hmm, observations)
```

Question 4

The most probable state in each time step is computed with the function which.max, the accuracy of each method can then be calculated by comparing the most probable states and the simulated ones.

```
most_probable_state <- function(x) {
  idx <- which.max(x)
  return(states[idx])
}

filtered_predictions <- apply(filtered, 2, most_probable_state)
smoothed_predictions <- apply(smoothed, 2, most_probable_state)
```

Accuracy of Viterbi algorithm

```r
viterbi_results <- table(most_probable_path == simulation$states)
print(viterbi_results['TRUE'] / sum(viterbi_results))
```

```
## TRUE
## 0.53
```

Accuracy of filtered distribution

```r
filtered_results <- table(filtered_predictions == simulation$states)
print(filtered_results['TRUE'] / sum(filtered_results))
```

```
## TRUE
##  0.5
```

Accuracy of smoothed distribution

```r
smoothed_results <- table(smoothed_predictions == simulation$states)
print(smoothed_results['TRUE'] / sum(smoothed_results))
```

```
## TRUE
## 0.73
```

Question 5

Repeat the exact same approach 10 times and print out the accuracies.

```r
viterbi_accuracies <- list()
filtered_accuracies <- list()
smoothed_accuracies <- list()

for(i in 1:10) {
  simulation <- simHMM(hmm, 100)
  observations <- simulation$observation

  alpha <- exp(forward(hmm, observations))
  beta <- exp(backward(hmm, observations))

  filtered <- prop.table(alpha, 2)
  smoothed <- prop.table(alpha*beta, 2)
  most_probable_path <- viterbi(hmm, observations)

  filtered_predictions <- apply(filtered, 2, most_probable_state)
  smoothed_predictions <- apply(smoothed, 2, most_probable_state)

  viterbi_results <- table(most_probable_path == simulation$states)
  viterbi_accuracies <- append(viterbi_accuracies, viterbi_results['TRUE']
                               / sum(viterbi_results))

  filtered_results <- table(filtered_predictions == simulation$states)
  filtered_accuracies <- append(filtered_accuracies, filtered_results['TRUE']
                                / sum(filtered_results))

  smoothed_results <- table(smoothed_predictions == simulation$states)
  smoothed_accuracies <- append(smoothed_accuracies, smoothed_results['TRUE']
                                / sum(smoothed_results))
}
```

```r
accuracies <- matrix(
  c(viterbi_accuracies, filtered_accuracies, smoothed_accuracies), 3, byrow=TRUE)
dimnames(accuracies) <- list(c('Viterbi', 'Filtered', 'Smoothed'))
print(accuracies)
```

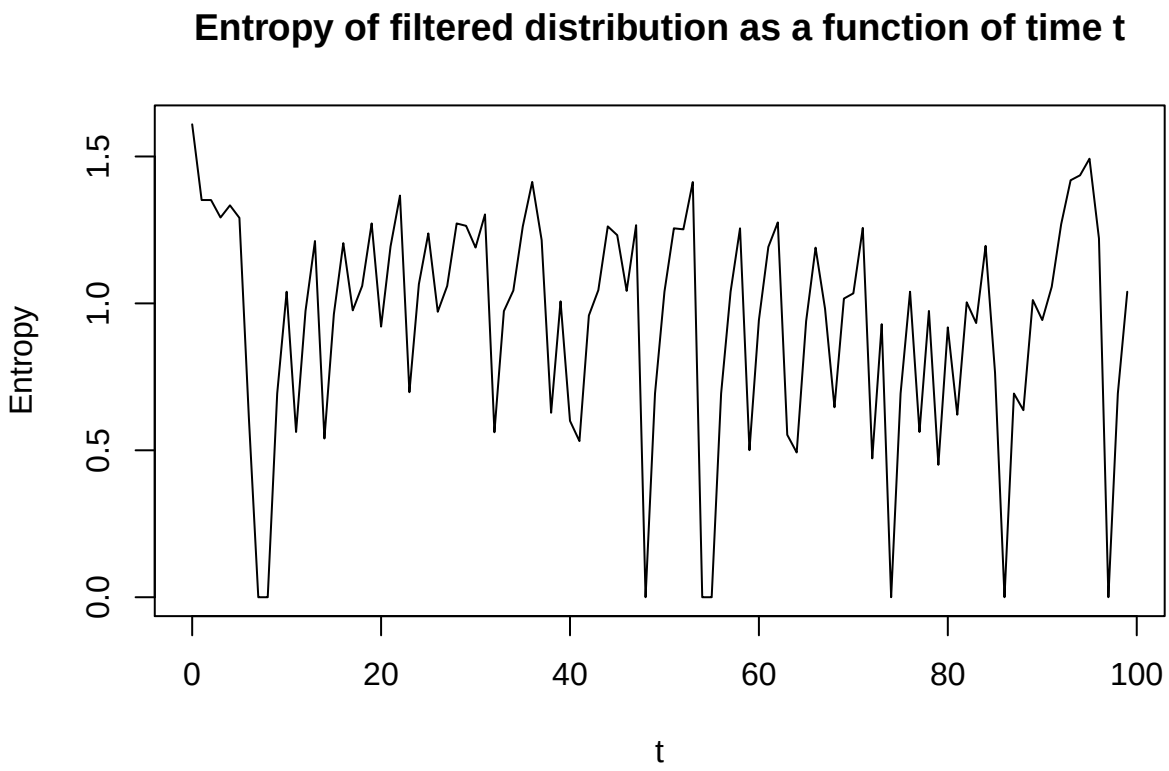
```
##          [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## Viterbi  0.5  0.51 0.46 0.52 0.43 0.47 0.56 0.56 0.45 0.64
## Filtered 0.5  0.43 0.61 0.5  0.55 0.49 0.52 0.53 0.57 0.44
## Smoothed 0.74 0.61 0.69 0.64 0.64 0.62 0.78 0.66 0.66 0.6
```

It can be seen that the accuracy of the smoothed distributions is superior to both the filtered distributions and the most probable path, and this is easily explainable. Since the smoothed distribution takes the observations of *all* timesteps into account, it will perform better than the filtered distribution which only considers the *previous* ones. And though the most probable path is also calculated from all timesteps, it has the additional constraint of having to generate a valid path, meaning that it cannot maximize the probability of correct predictions in each time step like the smoothed distribution does.

Question 6

We will examine how the number of observations affect the confidence in predictions by looking at the entropy of each time step in the filtered distribution. With exception for the global maximum at t=0, it seems like an increased number of observations does not increase confidence in where the robot is.

```r
plot(0:99, apply(filtered, 2, entropy.empirical), type = 'l', xlab = 't', ylab = 'Entropy',
     main = 'Entropy of filtered distribution as a function of time t')
```



Question 7

We seek to compute

$$p(Z^{101}|x^{0:100}) = \sum_{z^t} p(Z^{101}|z^{100})p(z^{100}|x^{0:100})$$

where $p(Z^{101}|z^{100})$ is the transition probability and $p(z^{100}|x^{0:100})$ is the filtered probability distribution.

```
probs <- list()

for (Z in states) {
  prob <- 0

  for (z in states) {
    prob <- prob + hmm$transProbs[z, Z] * filtered[z, 100]
  }

  probs <- append(probs, prob)
}
probs <- array(unlist(probs))
dimnames(probs) <- list(states)
print(probs)
```

```
##    z0    z1    z2    z3    z4    z5    z6    z7    z8    z9
## 0.000 0.000 0.125 0.375 0.375 0.125 0.000 0.000 0.000 0.000
```

Lab 2 Group Report

krisi211, marpe902, erisv795, emilu795

9/30/2019

Assignment 1

We create a Hidden Markov Model (HMM) with a hidden state Z^t that takes values $z_0 - z_9$ which represent the sector where the robot is at time step t, and an observed random variable X^t that takes values $x_0 - x_9$ which represent the reading of the tracking device at time step t.

The defining characteristics of the HMM is its initial model, transition model as well as emission model. In this case we use a uniform distribution for the initial model, the transition probabilities are 0.5 to stay in the current state, and 0.5 to move to the next one, and the emission probabilities are 0.2 to observe any of the states within 2 steps of the actual one. These matrices are manually defined as follows:

```
states <- c('z0', 'z1', 'z2', 'z3', 'z4', 'z5', 'z6', 'z7', 'z8', 'z9')
symbols <- c('x0', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9')
transProbs <- matrix(c(
  c(0.5, 0, 0, 0, 0, 0, 0, 0, 0, 0.5),
  c(0.5, 0.5, 0, 0, 0, 0, 0, 0, 0, 0),
  c(0, 0.5, 0.5, 0, 0, 0, 0, 0, 0, 0),
  c(0, 0, 0.5, 0.5, 0, 0, 0, 0, 0, 0),
  c(0, 0, 0, 0.5, 0.5, 0, 0, 0, 0, 0),
  c(0, 0, 0, 0, 0.5, 0.5, 0, 0, 0, 0),
  c(0, 0, 0, 0, 0, 0.5, 0.5, 0, 0, 0),
  c(0, 0, 0, 0, 0, 0, 0.5, 0.5, 0, 0),
  c(0, 0, 0, 0, 0, 0, 0, 0.5, 0.5, 0),
  c(0, 0, 0, 0, 0, 0, 0, 0, 0.5, 0.5)), 10)
emissionProbs <- matrix(c(
  c(0.2, 0.2, 0.2, 0, 0, 0, 0, 0, 0.2, 0.2),
  c(0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0, 0, 0.2),
  c(0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0, 0),
  c(0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0),
  c(0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0),
  c(0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0),
  c(0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0),
  c(0, 0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2),
  c(0.2, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2),
  c(0.2, 0.2, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2)), 10)

hmm <- initHMM(states, symbols, transProbs = transProbs, emissionProbs = emissionProbs)
```

Assignment 2

We run a simulation of the HMM lasting for 100 time steps, producing the following robot path and observed symbols:

```
simulation <- simHMM(hmm, 100)
print(simulation)
```

```
## $states
##   [1] "z1" "z1" "z1" "z2" "z2" "z2" "z2" "z3" "z3" "z3" "z3" "z4" "z5" "z5"
##  [15] "z5" "z5" "z5" "z6" "z7" "z7" "z7" "z8" "z9" "z9" "z0" "z1" "z1" "z2"
```

```
##  [29] "z2" "z3" "z3" "z4" "z4" "z4" "z4" "z4" "z4" "z4" "z5" "z5" "z6" "z7"
##  [43] "z8" "z9" "z9" "z0" "z1" "z2" "z2" "z3" "z3" "z3" "z3" "z3" "z4" "z4"
##  [57] "z4" "z4" "z4" "z4" "z4" "z5" "z6" "z7" "z7" "z7" "z7" "z8" "z9" "z9"
##  [71] "z0" "z0" "z0" "z1" "z2" "z3" "z4" "z4" "z5" "z6" "z7" "z8" "z9" "z0"
##  [85] "z0" "z1" "z2" "z3" "z3" "z3" "z4" "z5" "z5" "z5" "z6" "z6" "z7" "z7"
##  [99] "z7" "z7"
##
## $observation
##   [1] "x0" "x2" "x0" "x2" "x4" "x4" "x2" "x3" "x3" "x4" "x1" "x2" "x3" "x7"
##  [15] "x7" "x4" "x4" "x8" "x8" "x5" "x8" "x7" "x8" "x8" "x2" "x9" "x0" "x4"
##  [29] "x4" "x4" "x5" "x2" "x5" "x6" "x6" "x5" "x5" "x3" "x5" "x4" "x6" "x5"
##  [43] "x0" "x0" "x7" "x0" "x0" "x2" "x0" "x4" "x2" "x1" "x3" "x3" "x2" "x3"
##  [57] "x3" "x6" "x4" "x2" "x4" "x4" "x7" "x6" "x9" "x9" "x5" "x7" "x1" "x0"
##  [71] "x2" "x1" "x0" "x9" "x2" "x3" "x2" "x4" "x7" "x8" "x8" "x6" "x9" "x1"
##  [85] "x2" "x0" "x0" "x2" "x5" "x2" "x2" "x5" "x3" "x7" "x6" "x6" "x9" "x5"
##  [99] "x7" "x6"
```

Assignment 3

Only using the observations x^t, the filtered and smoothed distributions can be calculated as

$$p(Z^t|x^{0:t}) = \frac{\alpha(Z^t)}{\sum_{z^t}(\alpha(z^t))}$$

$$p(Z^t|x^{0:T}) = \frac{\alpha(Z^t)\beta(Z^t)}{\sum_{z^t}(\alpha(z^t)\beta(z^t))}$$

where α is computed with the forward function and β is computed with the backward function of the R package hmm. The package also provides the Viterbi algorithm, which is used to compute the most probable path.

```
observations <- simulation$observation

alpha <- exp(forward(hmm, observations))
beta <- exp(backward(hmm, observations))

filtered <- prop.table(alpha, 2)
smoothed <- prop.table(alpha*beta, 2)
most_probable_path <- viterbi(hmm, observations)
```

Assignment 4

The most probable state in each time step is computed with the function which.max, the accuracy of each method can then be calculated by comparing the most probable states and the simulated ones.

```
most_probable_state <- function(x) {
  idx <- which.max(x)
  return(states[idx])
}

filtered_predictions <- apply(filtered, 2, most_probable_state)
smoothed_predictions <- apply(smoothed, 2, most_probable_state)
```

Accuracy of Viterbi algorithm

```r
viterbi_results <- table(most_probable_path == simulation$states)
print(viterbi_results['TRUE'] / sum(viterbi_results))
```

```
## TRUE
## 0.53
```

Accuracy of filtered distribution

```r
filtered_results <- table(filtered_predictions == simulation$states)
print(filtered_results['TRUE'] / sum(filtered_results))
```

```
## TRUE
##  0.5
```

Accuracy of smoothed distribution

```r
smoothed_results <- table(smoothed_predictions == simulation$states)
print(smoothed_results['TRUE'] / sum(smoothed_results))
```

```
## TRUE
## 0.73
```

Assignment 5

Repeat the exact same approach 10 times and print out the accuracies.

```r
viterbi_accuracies <- list()
filtered_accuracies <- list()
smoothed_accuracies <- list()

for(i in 1:10) {
  simulation <- simHMM(hmm, 100)
  observations <- simulation$observation

  alpha <- exp(forward(hmm, observations))
  beta <- exp(backward(hmm, observations))

  filtered <- prop.table(alpha, 2)
  smoothed <- prop.table(alpha*beta, 2)
  most_probable_path <- viterbi(hmm, observations)

  filtered_predictions <- apply(filtered, 2, most_probable_state)
  smoothed_predictions <- apply(smoothed, 2, most_probable_state)

  viterbi_results <- table(most_probable_path == simulation$states)
  viterbi_accuracies <- append(viterbi_accuracies, viterbi_results['TRUE']
                               / sum(viterbi_results))

  filtered_results <- table(filtered_predictions == simulation$states)
  filtered_accuracies <- append(filtered_accuracies, filtered_results['TRUE']
                                / sum(filtered_results))

  smoothed_results <- table(smoothed_predictions == simulation$states)
  smoothed_accuracies <- append(smoothed_accuracies, smoothed_results['TRUE']
                                / sum(smoothed_results))
}
```

```r
accuracies <- matrix(
  c(viterbi_accuracies, filtered_accuracies, smoothed_accuracies), 3, byrow=TRUE)
dimnames(accuracies) <- list(c('Viterbi', 'Filtered', 'Smoothed'))
print(accuracies)
```

```
##          [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## Viterbi  0.5  0.51 0.46 0.52 0.43 0.47 0.56 0.56 0.45 0.64
## Filtered 0.5  0.43 0.61 0.5  0.55 0.49 0.52 0.53 0.57 0.44
## Smoothed 0.74 0.61 0.69 0.64 0.64 0.62 0.78 0.66 0.66 0.6
```

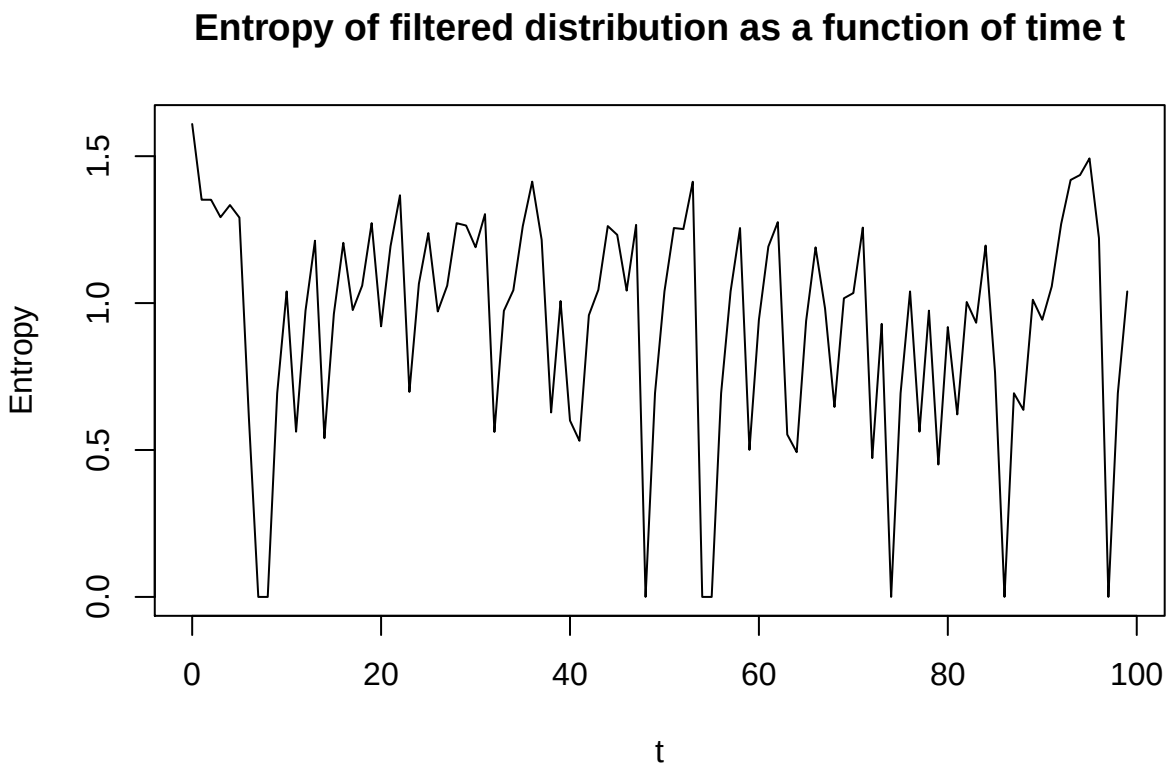
It can be seen that the accuracy of the smoothed distributions is superior to both the filtered distributions and the most probable path, and this is easily explainable. Since the smoothed distribution takes the observations of *all* timesteps into account, it will perform better than the filtered distribution which only considers the *previous* ones. And though the most probable path is also calculated from all timesteps, it has the additional constraint of having to generate a valid path, meaning that it cannot maximize the probability of correct predictions in each time step like the smoothed distribution does.

Assignment 6

The members of the group used two different approaches to assignment 6.

One approach is to calculate the entropy at each time step when using the filtered distribution, thus seeing how confident the predictions are at the beginning when only a few observations have been made in contrast to at the end when t nears 100. Plotting the entropy this way gives the following graph, suggesting that more observations does not imply lower entropy (higher confidence).

```r
plot(0:99, apply(filtered, 2, entropy.empirical), type = 'l', xlab = 't', ylab = 'Entropy',
     main = 'Entropy of filtered distribution as a function of time t')
```



Another approach is to perform another simulation, this time using more than 100 time steps and seeing if this affects the entropy in any significant way. This approach gives the following results:

```
sim.200 = simHMM(hmm,200)
alpha.200 = exp(forward(hmm,sim.200$observation))
filtered.200 = prop.table(alpha.200,2)
```

The entropy for 100 simulated time steps:

```
entropy.empirical(filtered)
```

```
## [1] 5.535763
```

The entropy for 200 simulated time steps:

```
entropy.empirical(filtered.200)
```

```
## [1] 6.207583
```

Since the entropy score is higher for 200 simulated time steps than for 100 simulated time steps, meaning that more simulated time steps gives more uncertainty. Therefor more observations does not mean that we better know were the robot is.

Assignment 7

We seek to compute

$$p(Z^{101}|x^{0:100}) = \sum_{z^t} p(Z^{101}|z^{100})p(z^{100}|x^{0:100})$$

where $p(Z^{101}|z^{100})$ is the transition probability and $p(z^{100}|x^{0:100})$ is the filtered probability distribution.

```
probs <- list()

for (Z in states) {
  prob <- 0

  for (z in states) {
    prob <- prob + hmm$transProbs[z, Z] * filtered[z, 100]
  }

  probs <- append(probs, prob)
}
probs <- array(unlist(probs))
dimnames(probs) <- list(states)
print(probs)
```

```
##    z0    z1    z2    z3    z4    z5    z6    z7    z8    z9
## 0.000 0.000 0.125 0.375 0.375 0.125 0.000 0.000 0.000 0.000
```

Lab 3

Emil Luusua

4 October 2019

Question 1

We start by defining functions that sample the transition, emission and initial models. The functions rnorm and runif are used to sample from normal/uniform distributions. For the mixture models each component is sampled, and then one of these samples are chosen at random. The standard deviation of the emission model (sd) is taken as an argument to ease in question 2.

```
transition_model <- function(prev) {
  components <- c(rnorm(1, prev, 1), rnorm(1, prev + 1, 1), rnorm(1, prev + 2, 1))
  return(sample(components, 1))
}

emission_model <- function(pos, sd) {
  components <- c(rnorm(1, pos, sd), rnorm(1, pos - 1, sd), rnorm(1, pos + 1, sd))
  return(sample(components, 1))
}

initial_model <- function() {
  return(runif(1, 0, 100))
}
```

We also implement a function that evaluates the probability density function of the emission model which can be used to compute the weights of the particle filter. This is done by evaluating the densities of each component and then taking the average.

```
emission_density <- function(obs, pos, sd) {
  components <- c(dnorm(obs, pos, sd), dnorm(obs, pos - 1, sd), dnorm(obs, pos + 1, sd))
  return(mean(components))
}
```

We implement a function which performs a simulation of the state space model for 100 time steps and run it with standard deviation set to 1.

```
simulate_SSM <- function(sd) {
  T <- 100

  initial_pos <- initial_model()
  initial_obs <- emission_model(initial_pos, sd)

  res <- list()
  res$states <- rep(initial_pos, 100)
  res$observation <- rep(initial_obs, 100)

  for (t in 2:T) {
    res$states[t] <- transition_model(res$states[t-1])
    res$observation[t] <- emission_model(res$states[t], sd)
  }
  return(res)
}
```

```
simulation <- simulate_SSM(1)
print(simulation)
```

```
## $states
##   [1]   6.936092   7.663796   8.044751   8.768376  10.914176  10.541719
##   [7]  13.705987  14.663302  14.349030  17.195820  18.190756  17.688374
##  [13]  18.066706  21.100409  23.151840  24.665540  25.883802  24.501565
##  [19]  22.668095  24.035609  23.322178  25.278869  27.640815  31.276464
##  [25]  30.756430  32.213728  34.103179  35.250463  36.236799  38.328719
##  [31]  39.969492  40.342416  41.211341  42.215793  42.160526  43.200188
##  [37]  41.925372  43.032388  44.636134  46.131723  45.286144  47.608645
##  [43]  48.681337  51.606974  51.866902  53.938699  55.938736  57.458991
##  [49]  57.256813  58.822515  60.037076  61.504379  60.449408  63.750069
##  [55]  64.351917  66.713789  67.936669  67.977009  68.712783  68.408103
##  [61]  68.691349  71.853369  71.604300  75.302710  76.896840  79.968063
##  [67]  81.819606  82.413968  85.081642  86.156303  86.459776  87.716805
##  [73]  88.056485  90.109029  92.969746  93.917472  95.942964  98.099243
##  [79] 100.308829 100.040190 100.919362 100.376073 101.076424 103.023248
##  [85] 101.894107 102.060801 103.745453 104.122598 106.877694 106.872345
##  [91] 107.520329 108.761196 109.780867 109.960500 112.410534 112.258361
##  [97] 112.272334 115.066480 117.456556 118.416481
##
## $observation
##   [1]   7.843011   7.511138   7.341287   9.780355  11.588157  11.339825
##   [7]  11.728934  15.658592  14.544647  19.340881  17.919702  16.414314
##  [13]  17.845001  21.325467  22.600382  25.141014  25.794321  25.282215
##  [19]  19.484256  24.336131  23.414762  25.057522  29.356199  31.931012
##  [25]  33.449978  32.254474  35.601541  35.643043  37.481094  38.297308
##  [31]  41.968774  39.090663  39.563526  43.885890  44.444616  41.663492
##  [37]  41.305225  42.977498  45.487453  46.993496  42.348581  46.654170
##  [43]  49.500169  51.095714  50.273275  53.112570  56.467121  59.725722
##  [49]  58.066946  56.323091  59.881801  61.295470  60.189195  62.204026
##  [55]  61.837063  66.345605  70.676691  69.290940  68.758788  67.396118
##  [61]  69.541993  72.784732  71.572754  75.907780  74.900185  82.943593
##  [67]  82.544844  83.082042  82.532340  85.345718  85.139391  88.447691
##  [73]  88.134805  89.307835  93.750478  92.358504  94.995538  97.915975
##  [79]  99.975175  99.775012  99.563152 103.164857 102.809928 102.427494
##  [85] 102.308416 102.425571 104.576209 102.790109 107.927225 111.289437
##  [91] 106.972454 107.821229 108.261961 109.235527 111.720347 111.449040
##  [97] 110.448339 114.931943 117.096644 116.603993
```

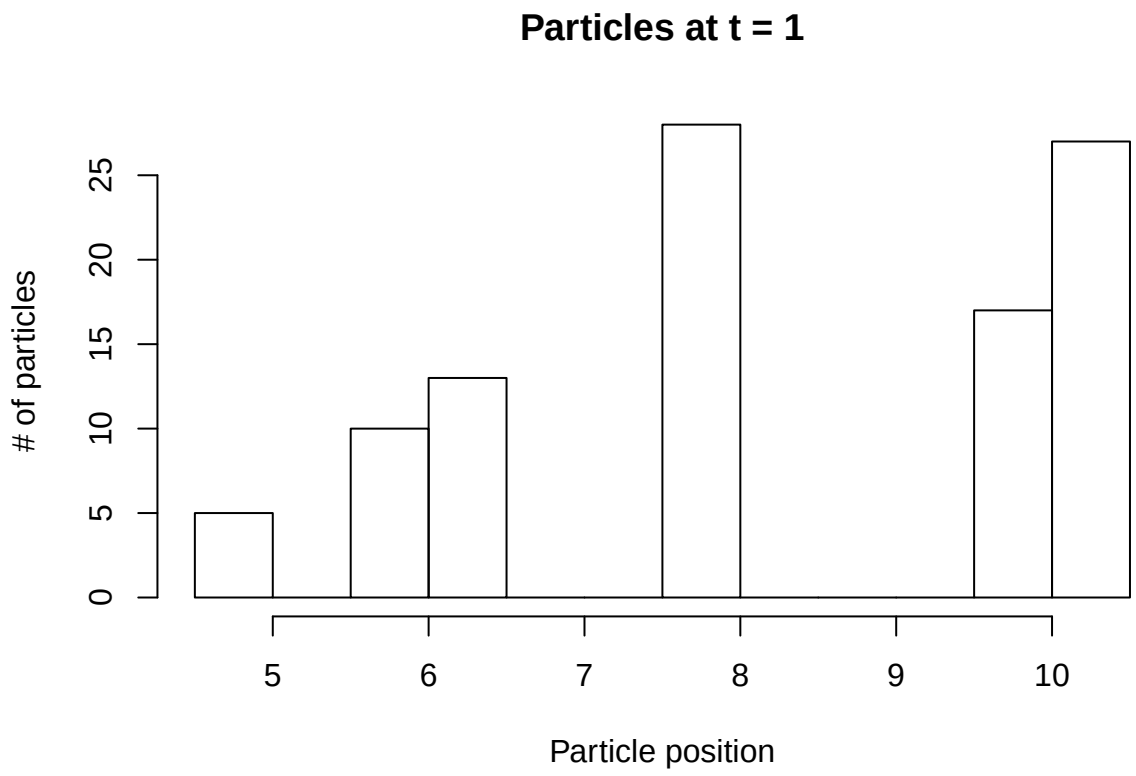
Next we implement the particle filter using 100 particles. The version presented by Thrun et al. in *Probabilistic Robotics* is utilized. The option of using weights for correction and the standard deviation is taken as arguments to ease in questions 2 and 3. The functions returns the belief particles at each time step.

```
particle_filter <- function(observations, sd, use_weights) {
  # Constants
  T <- 100
  M <- 100

  # Initialize variables
  particles <- matrix(rep(0, T * M), ncol = M)
  bel <- rep(0, M)
  bel_bar <- rep(0, M)
```

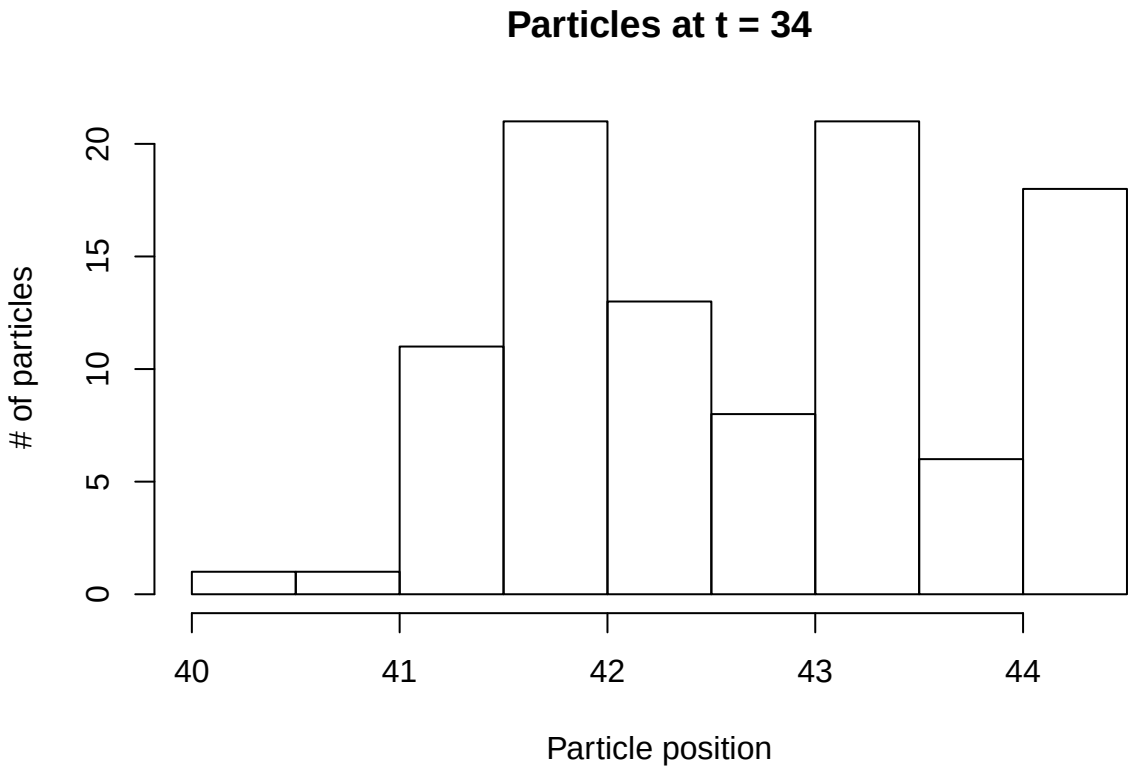
```
  w <- rep(1, M)

  # Generate initial particles
  for(m in 1:M) {
    bel[m] <- initial_model()
  }

  for(t in 1:T) {
    for(m in 1:M) {
      bel_bar[m] <- transition_model(bel[m])

      if(use_weights) {
        w[m] <- emission_density(observations[t], bel_bar[m], sd)
      }
    }

    bel <- sample(bel_bar, M, replace = TRUE, prob = w)
    particles[t, ] <- bel
  }

  return(particles)
}
```

We apply the particle filter to our simulated observations and show the particles, their mean (the expected location) and the true location at time steps $t = 1, 34, 67, 100$.

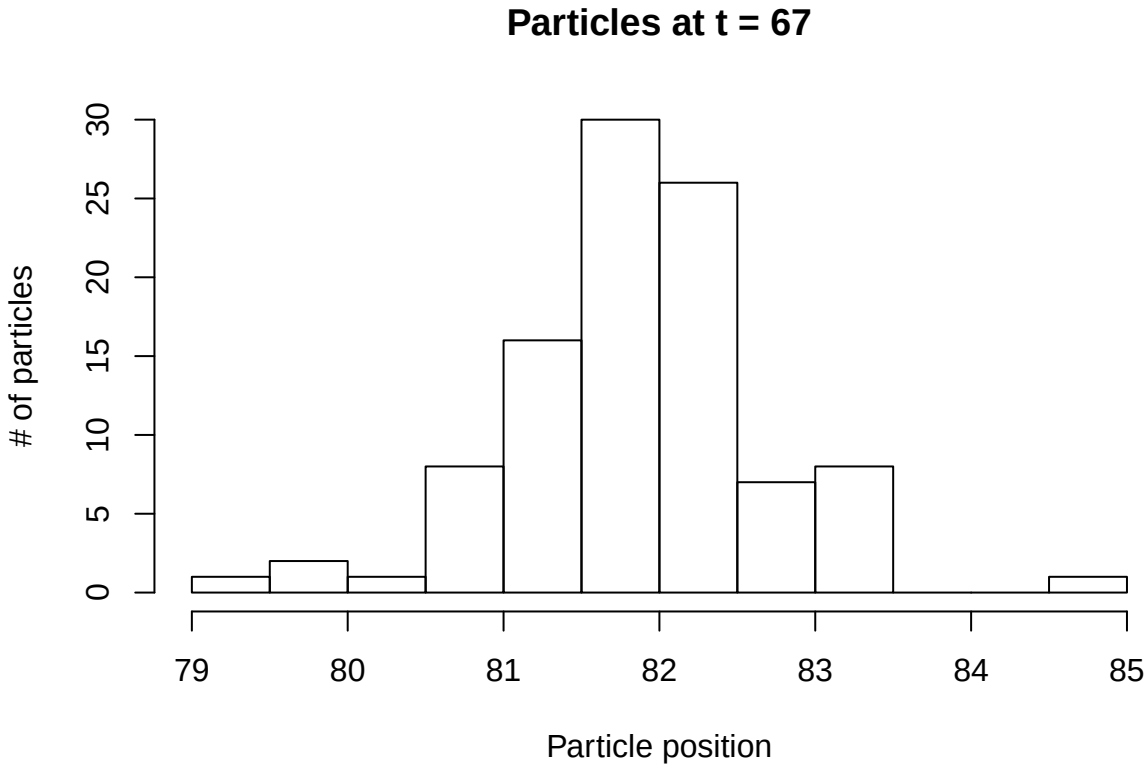
```
particles <- particle_filter(simulation$observation, 1, TRUE)
```



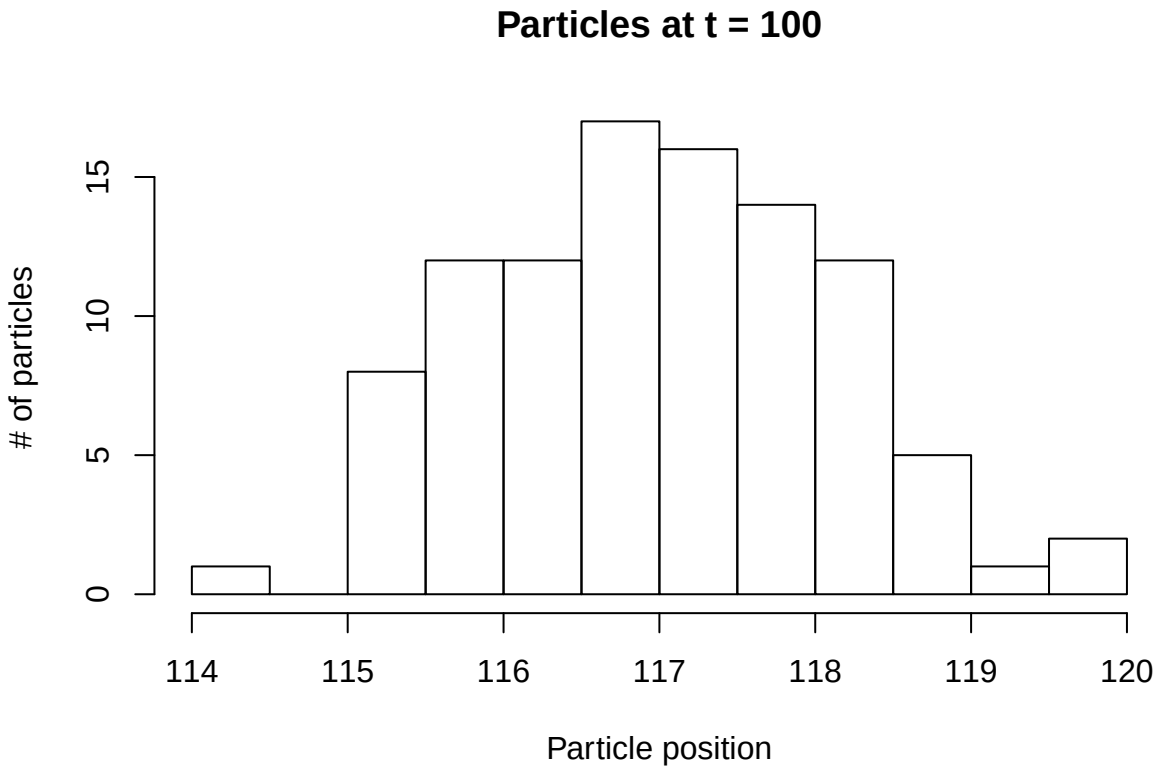
Expected location at $t = 1$: 8.1120261
True location at $t = 1$: 6.9360916



Expected location at $t = 34$: 42.6762199
True location at $t = 34$: 42.2157926



Expected location at $t = 67$: 81.8148068
True location at $t = 67$: 81.8196063



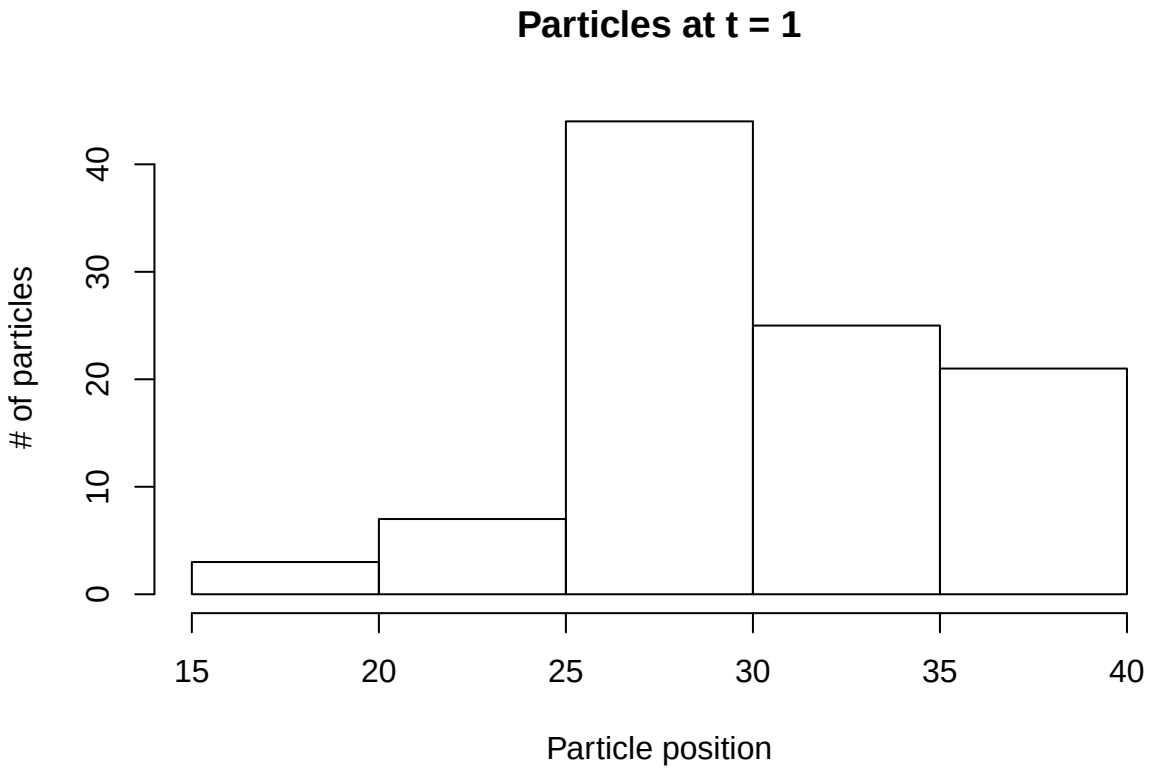
Expected location at $t = 100$: 117.0005333
True location at $t = 100$: 118.4164807

Question 2

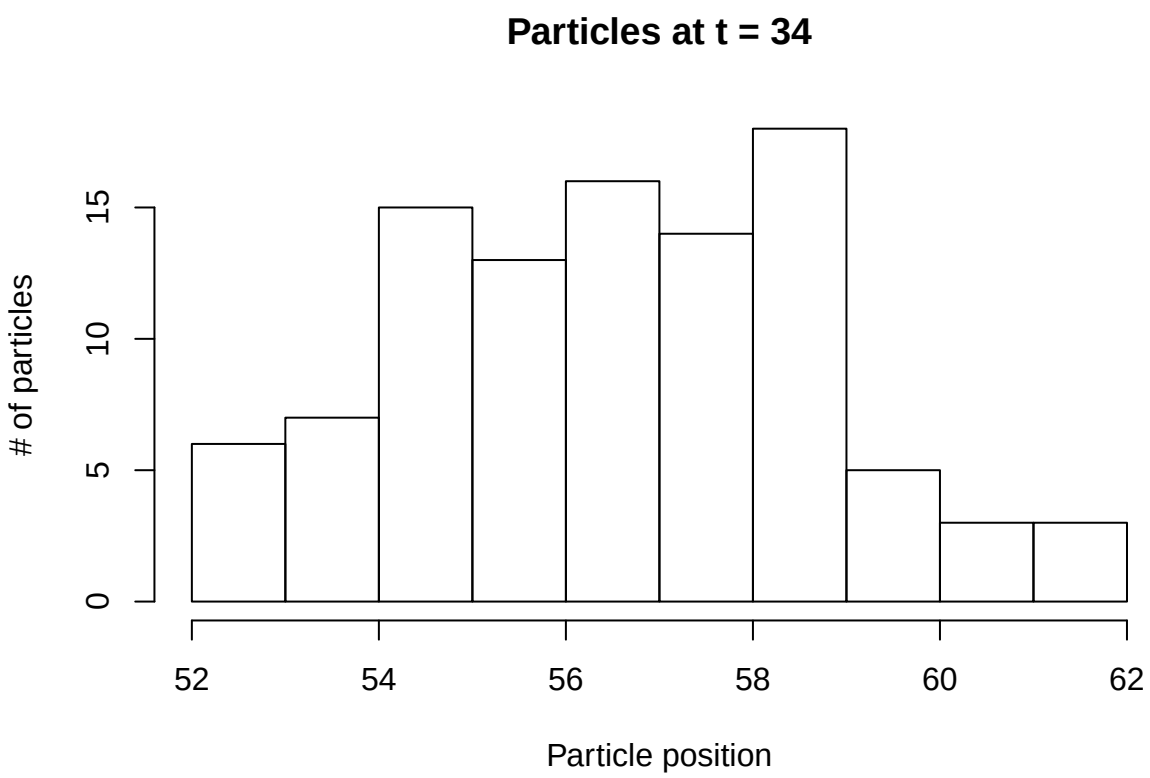
We now apply the same procedure, using standard deviations 5 and 50. The results are presented below, and unsurprisingly the conclusion is that the higher degree of variance in our observations cause the model to make more inaccurate predictions.

Standard deviation of 5

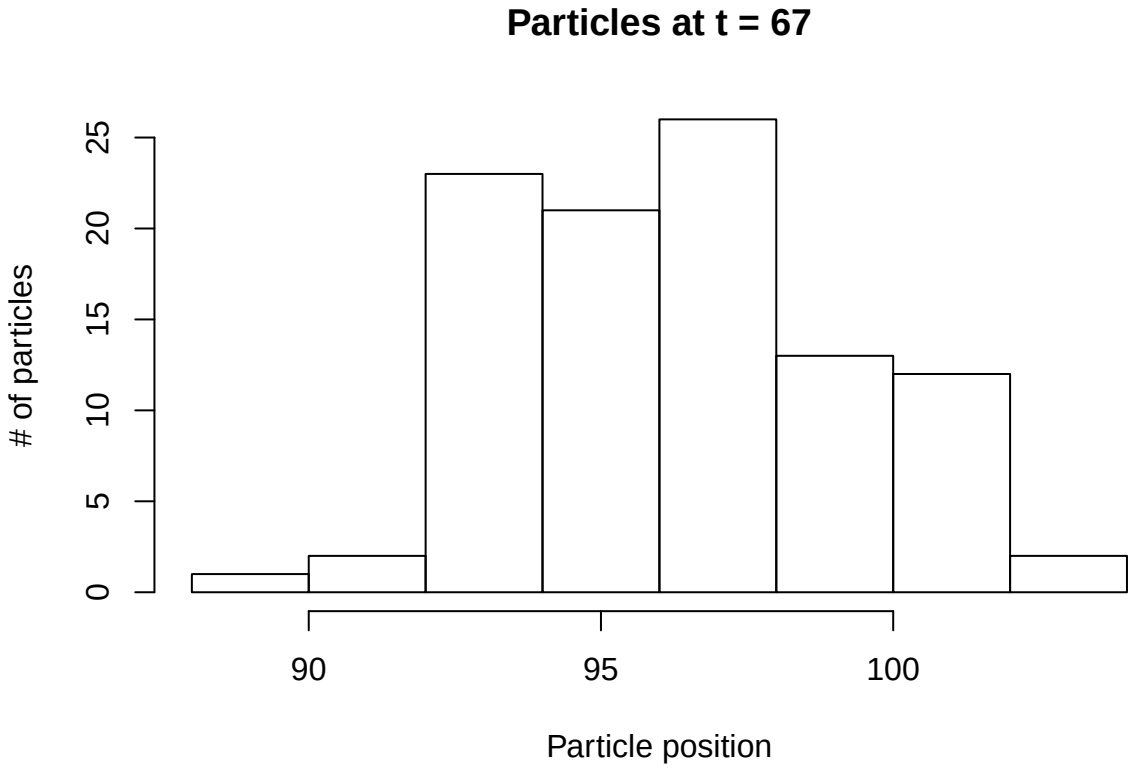
```
sd <- 5
simulation <- simulate_SSM(sd)
particles <- particle_filter(simulation$observation, sd, TRUE)
```



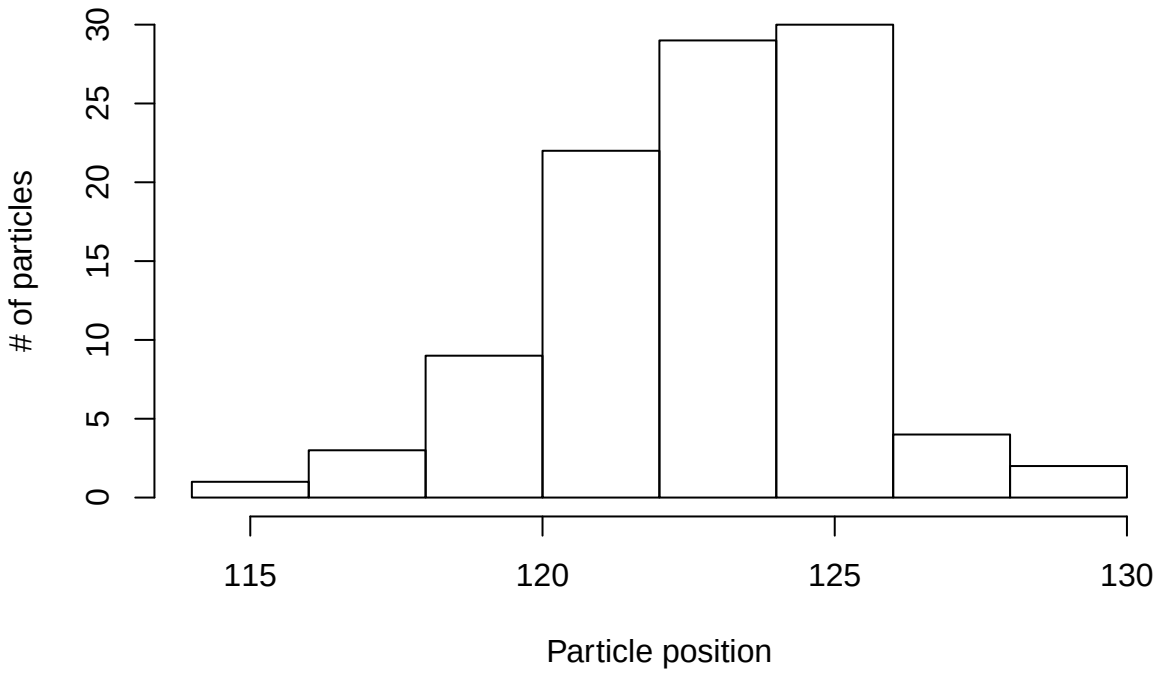
Expected location at $t = 1$: 29.5683074
True location at $t = 1$: 24.4206734



Expected location at $t = 34$: 56.5266417
True location at $t = 34$: 53.2940375



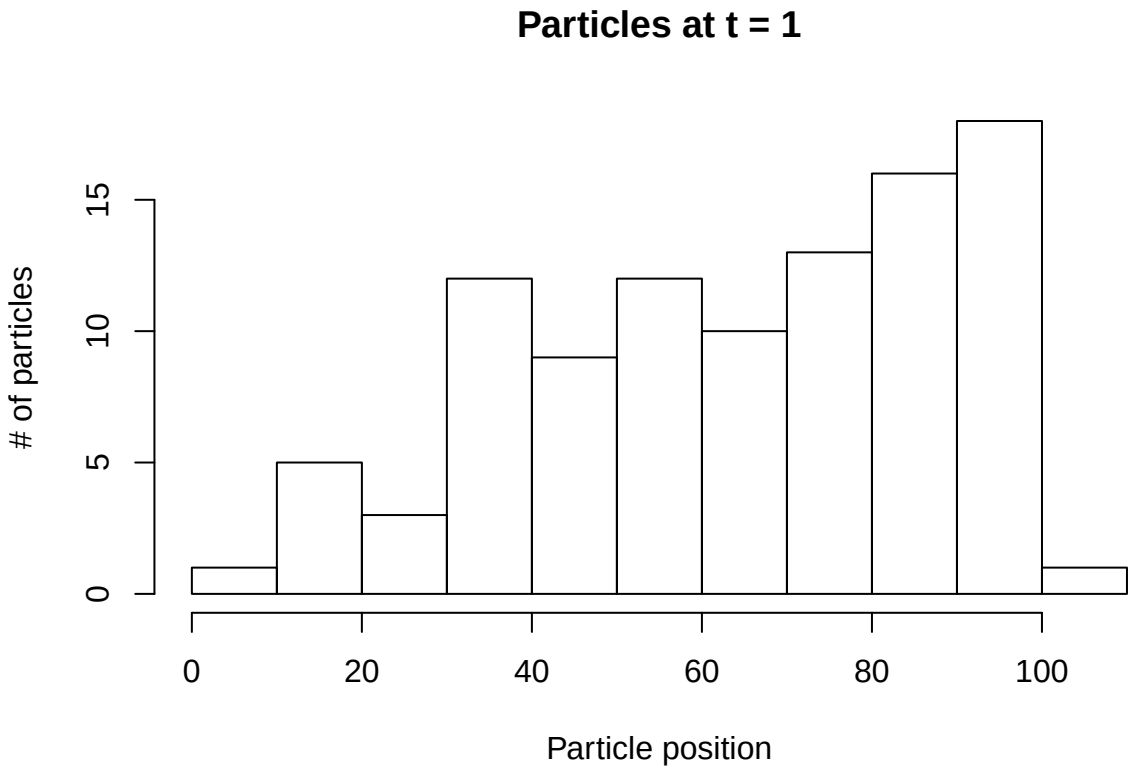
Expected location at $t = 67$: 96.376995
True location at $t = 67$: 96.0020445



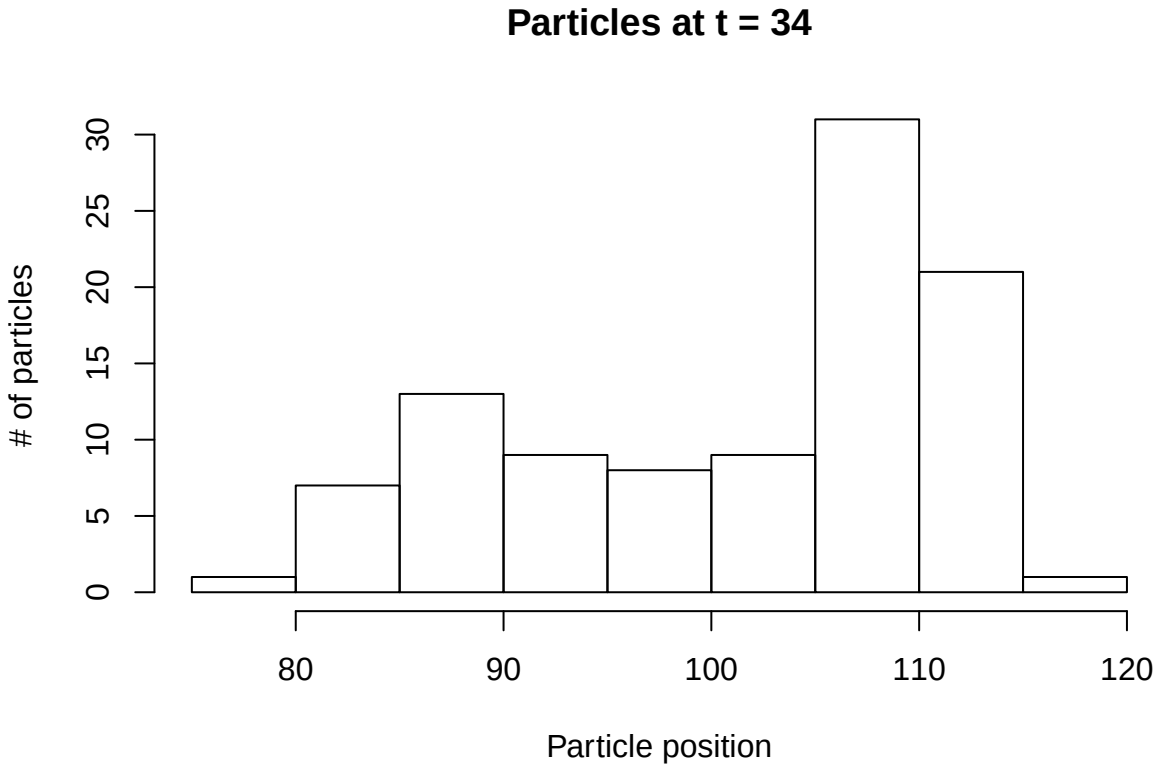
Expected location at $t = 100$: 122.7330781
True location at $t = 100$: 122.1667248

Standard deviation of 50

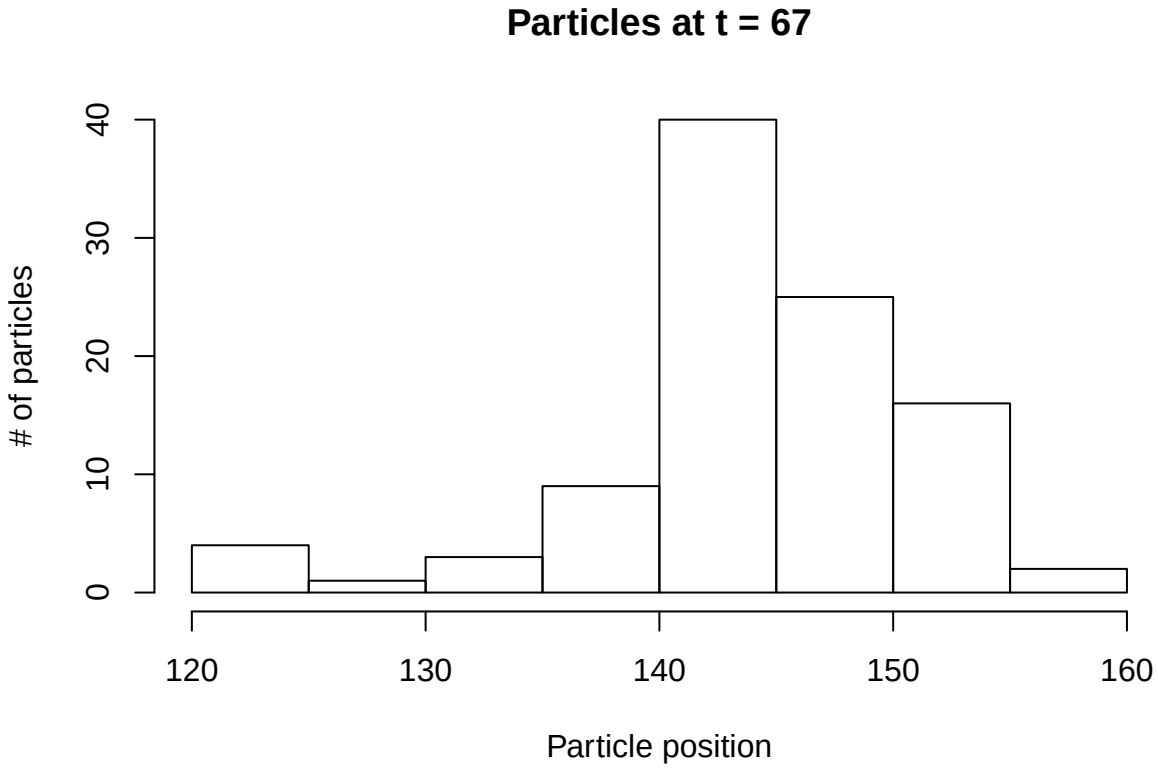
```
sd <- 50
simulation <- simulate_SSM(sd)
particles <- particle_filter(simulation$observation, sd, TRUE)
```



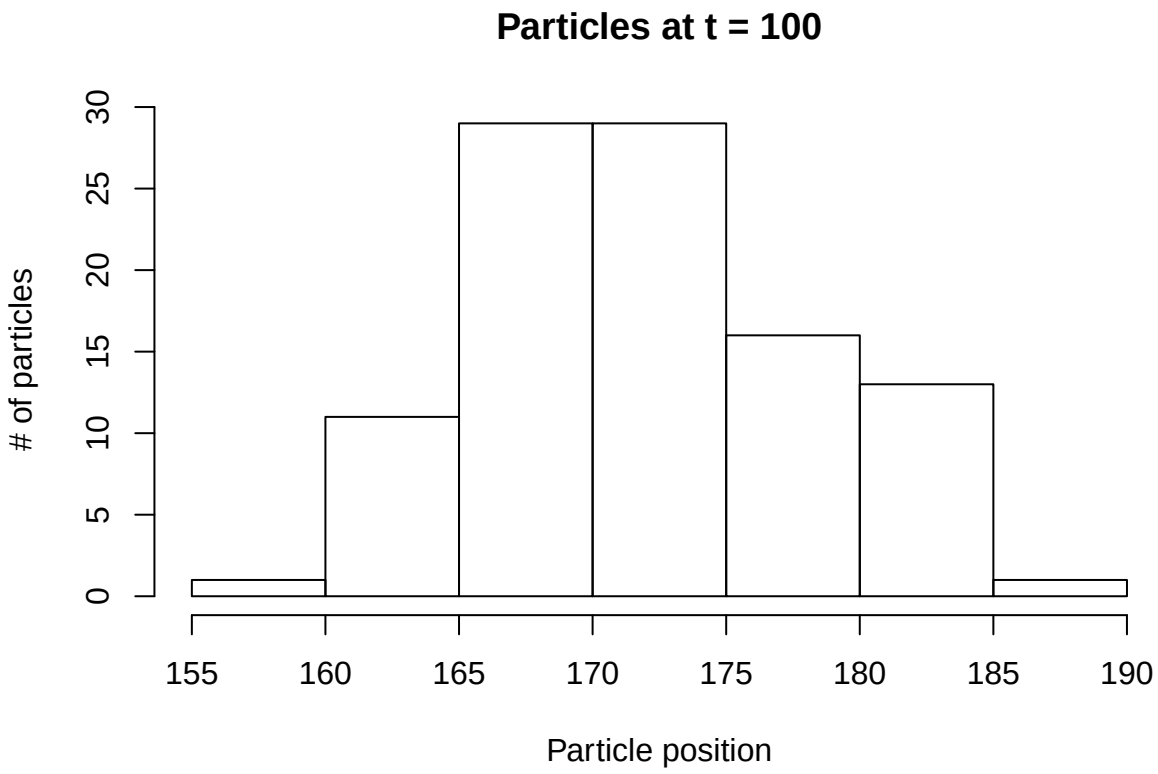
Expected location at $t = 1$: 64.6291816
True location at $t = 1$: 74.8752381



Expected location at $t = 34$: 101.5647865
True location at $t = 34$: 112.925117



Expected location at $t = 67$: 143.9426178
True location at $t = 67$: 149.8718779

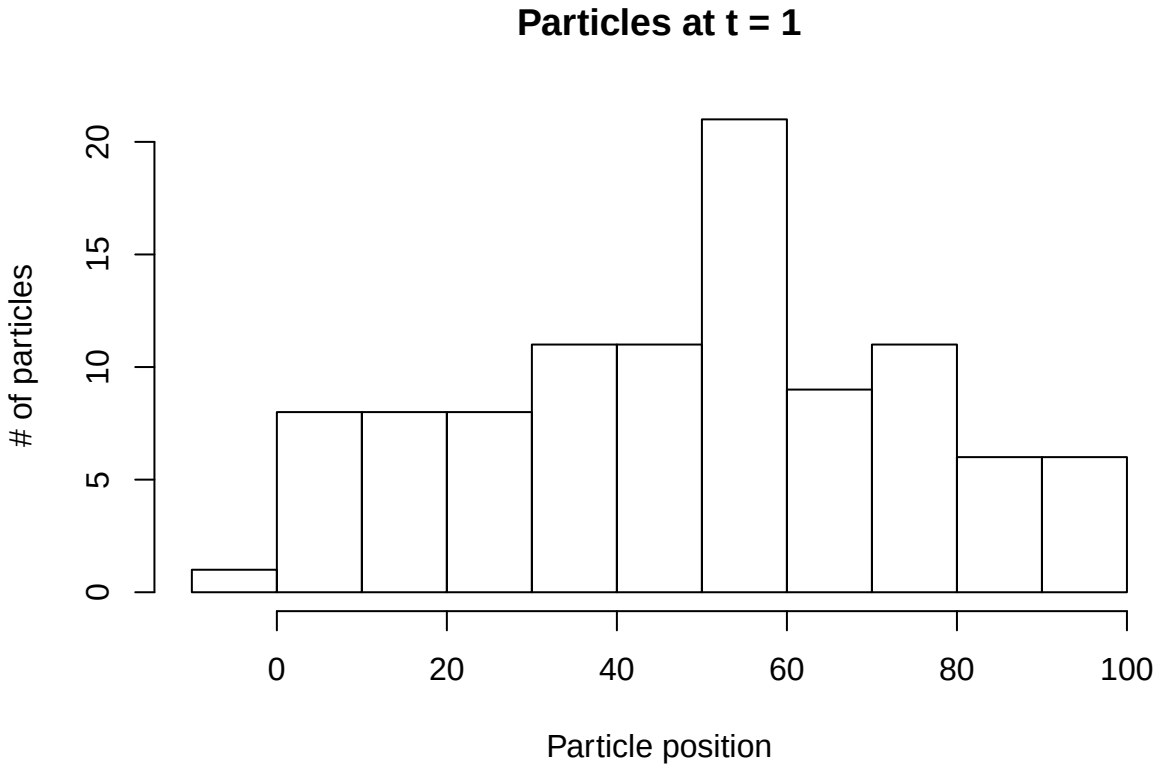


Expected location at $t = 100$: 172.1861847
True location at $t = 100$: 178.3200596

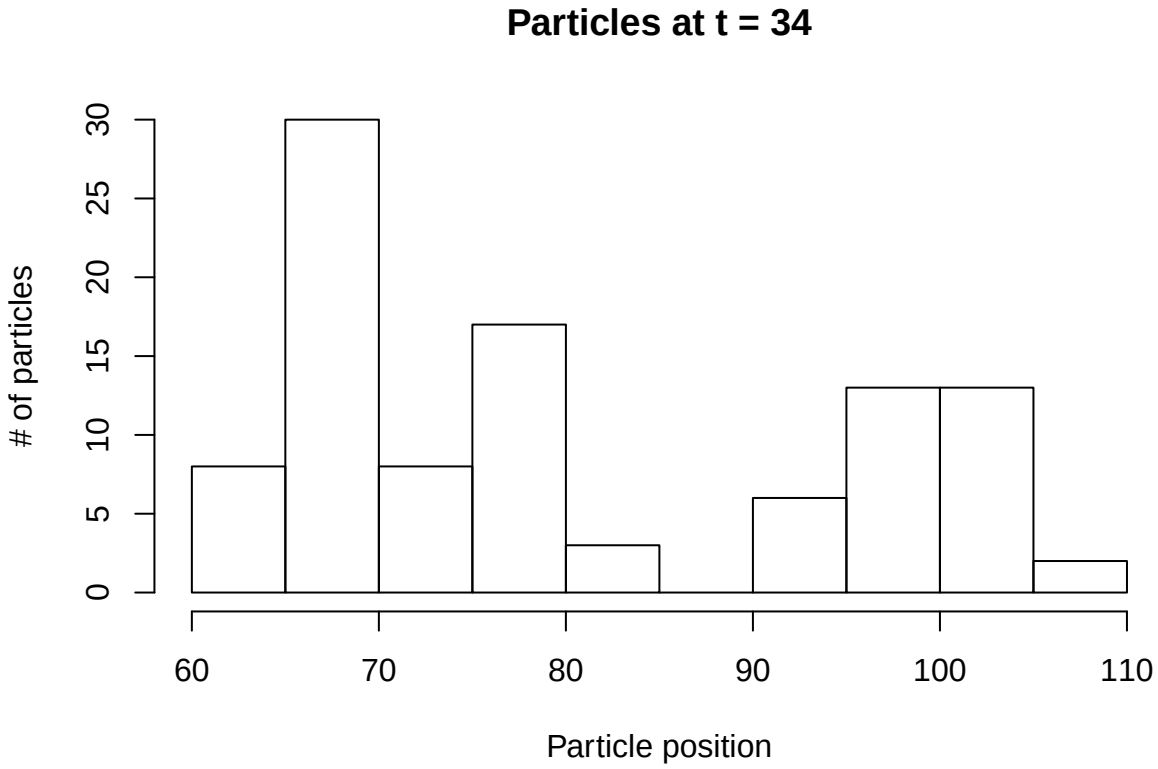
Question 3

We repeat the procedure one final time, with correction weights disabled. This performs significantly worse, since it means that we do not take our observations into account at all. So basically the particle filter just performs another simulation, independent of our observations.

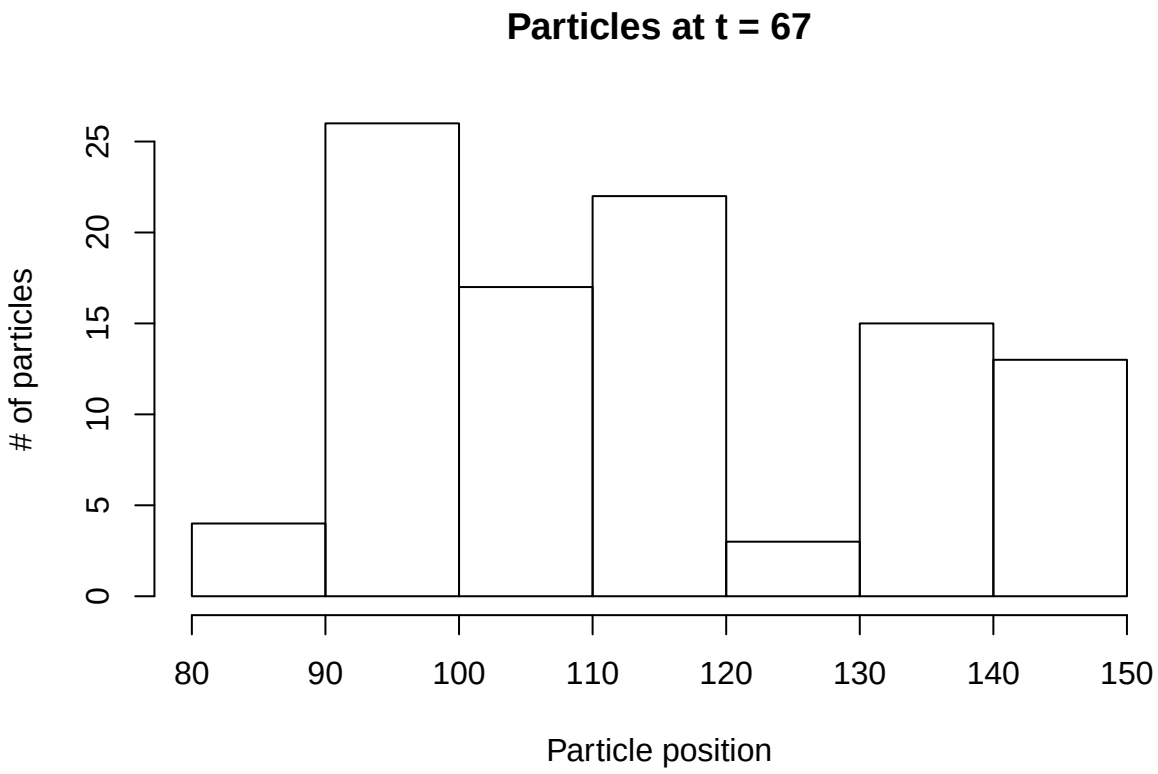
```
sd <- 1
simulation <- simulate_SSM(sd)
particles <- particle_filter(simulation$observation, sd, FALSE)
```



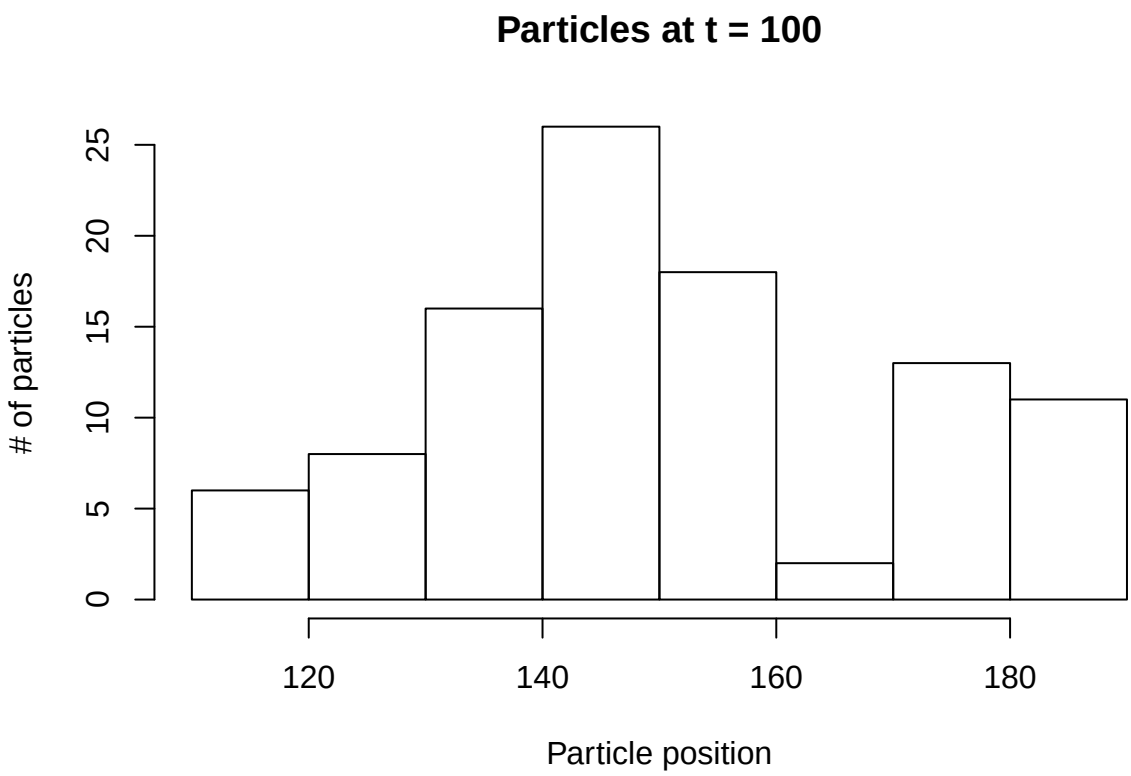
Expected location at $t = 1$: 48.2250285
True location at $t = 1$: 10.0488621



Expected location at $t = 34$: 80.4806726
True location at $t = 34$: 44.611242



Expected location at $t = 67$: 114.0093972
True location at $t = 67$: 73.3918424



Expected location at $t = 100$: 149.6663561
True location at $t = 100$: 117.2489298

Lab 3

Kristian Sikiric (krisi211), Erik Svensson (erisv796), Martin Persson (marpe902), Emil Luusua (emilu795)

The lab

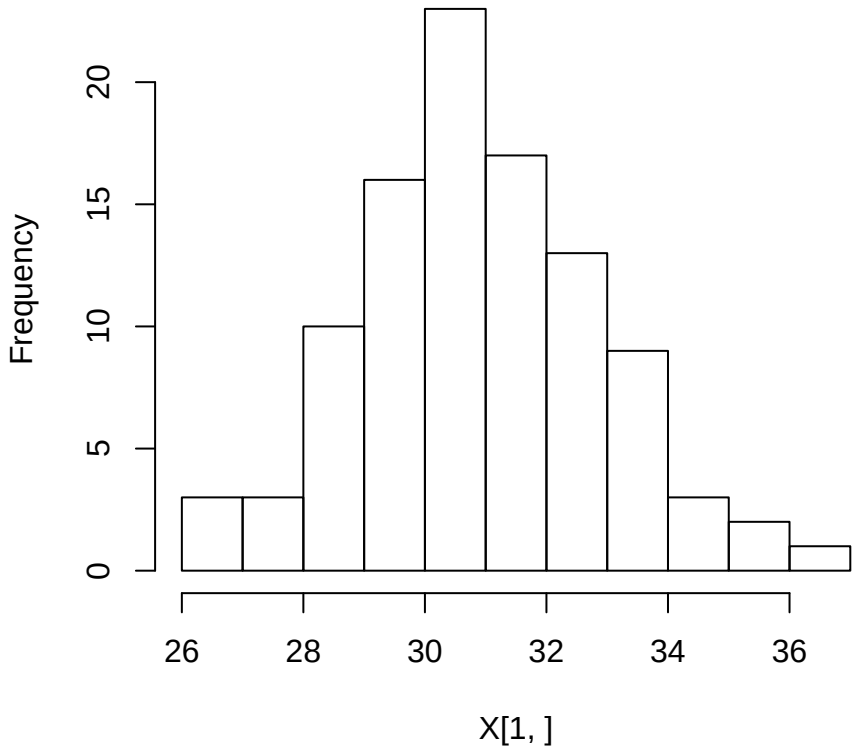
The purpose of the lab is to put in practice some of the concepts covered in the lectures. To do so, you are asked to implement the particle filter for robot localization. The robot moves along the horizontal axis according to the following SSM:

Transition model: $p(z_t|z_{t1}) = (N(z_t|z_{t-1}, 1) + N(z_t|z_{t-1} + 1, 1) + N(z_t|z_{t-1} + 2, 1))/3$

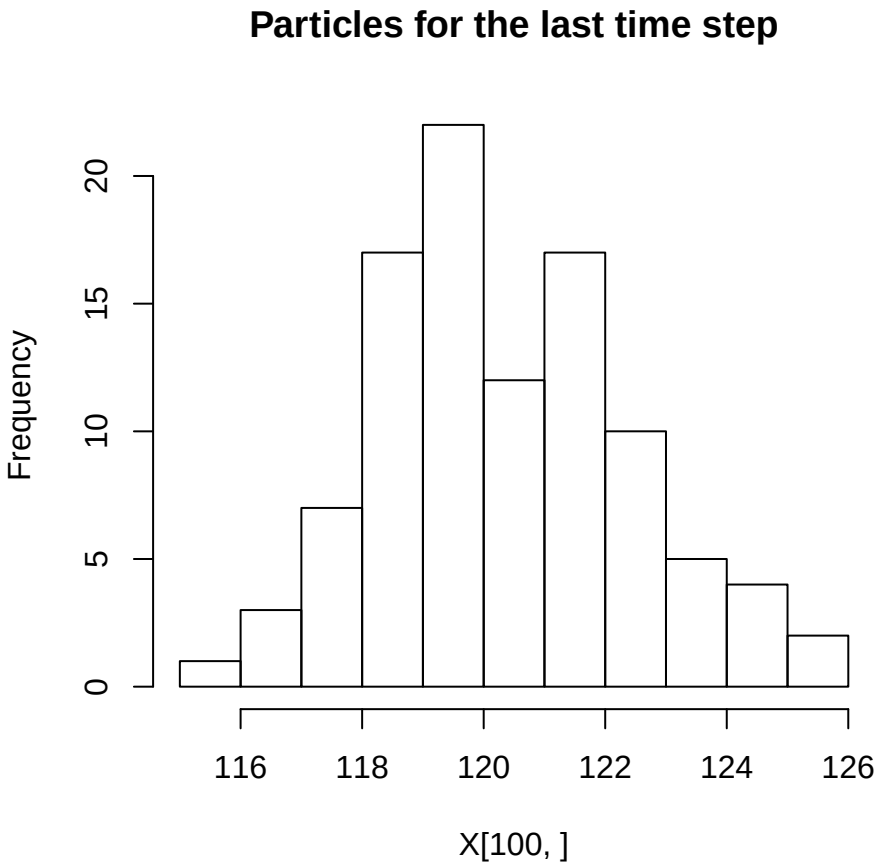
Emission model: $p(x_t|z_t) = (N(x_t|z_t, 1) + N(x_t|z_t - 1, 1) + N(x_t|z_t + 1, 1))/3$ Initial model: $p(z_1) = Uniform(0, 100)$

Assignment 1

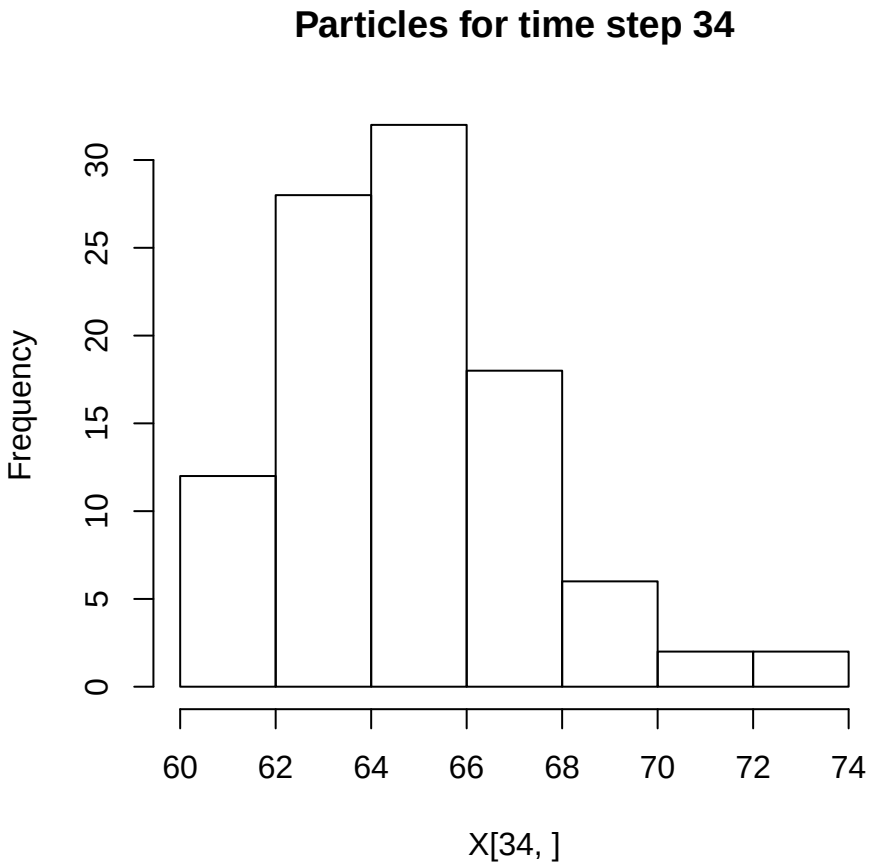
The first assignment was to implement the SSM described above and simulate it for 100 time steps to obtain states and observations. Then the observations were to be used to identify the states via particle filtering. Below the particles for time steps 1, 100, 34 and 35 are shown, as well as the expected value for these states and the true states.



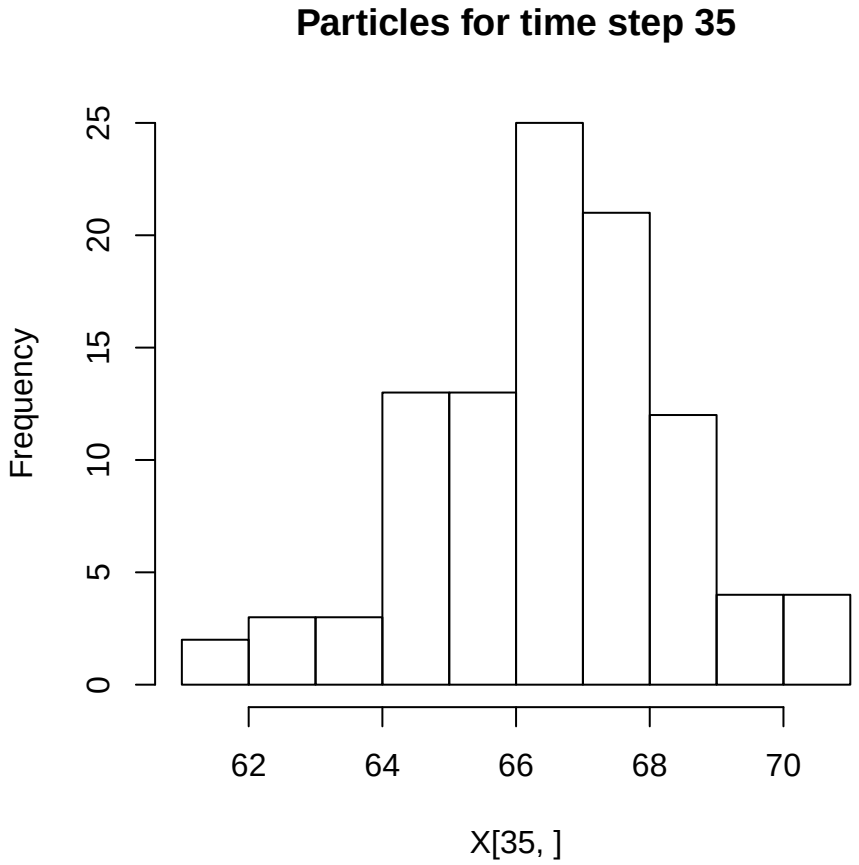

```
## [1] "Expected state for time step 1: 31.5319012461235"
## [1] "True state for time step 1: 28.7577520124614"
```




```
## [1] "Expected state for time step 100: 120.226234033726"
## [1] "True state for time step 100: 117.38417628889"
```



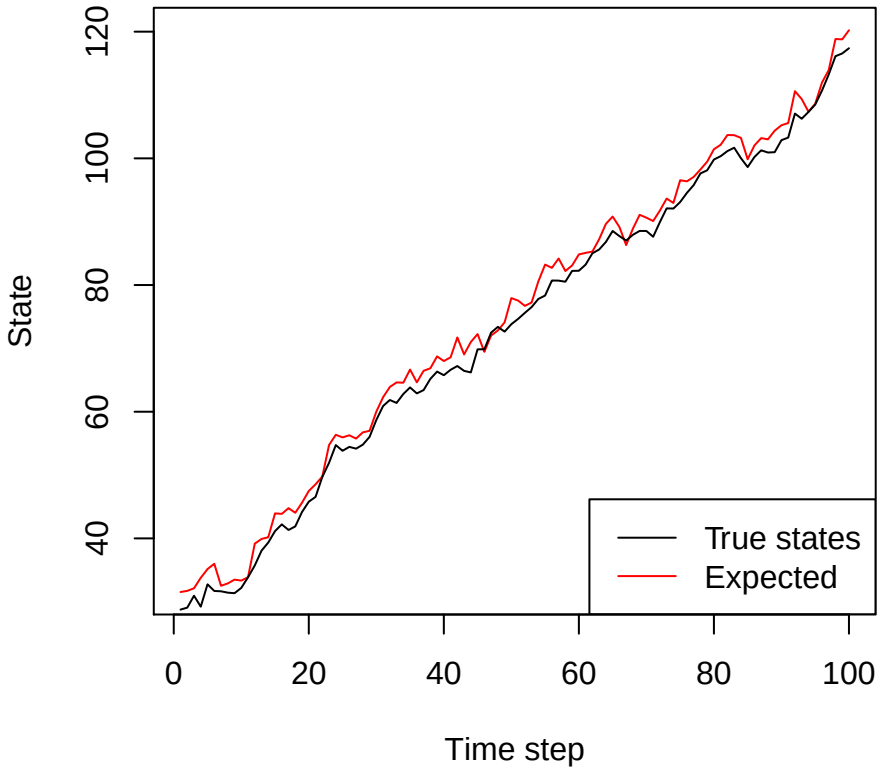

```
## [1] "Expected state for time step 34: 64.5748055970024"
## [1] "True state for time step 34: 62.7955988671768"
```




```
## [1] "Expected state for time step 35: 66.6594099422637"
```

```
## [1] "True state for time step 35: 63.8332817145742"
```

Below we can see the expected state for each time step in comparison with the true state and the MSE between the two.

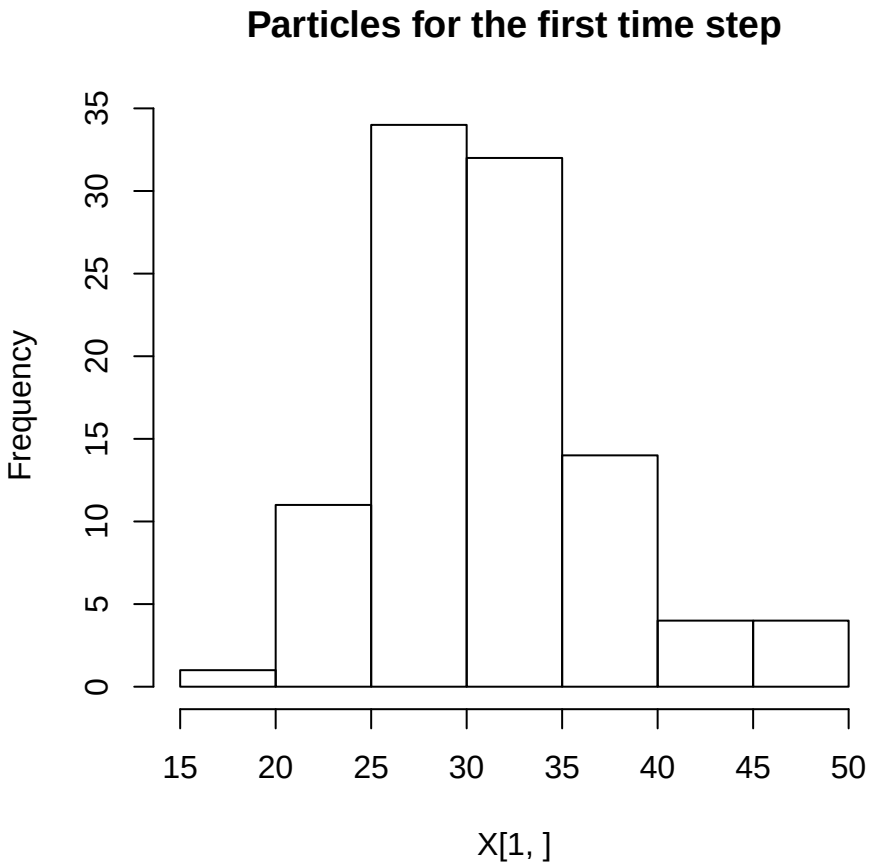



```
## [1] "MSE:  5.22430332845788"
```

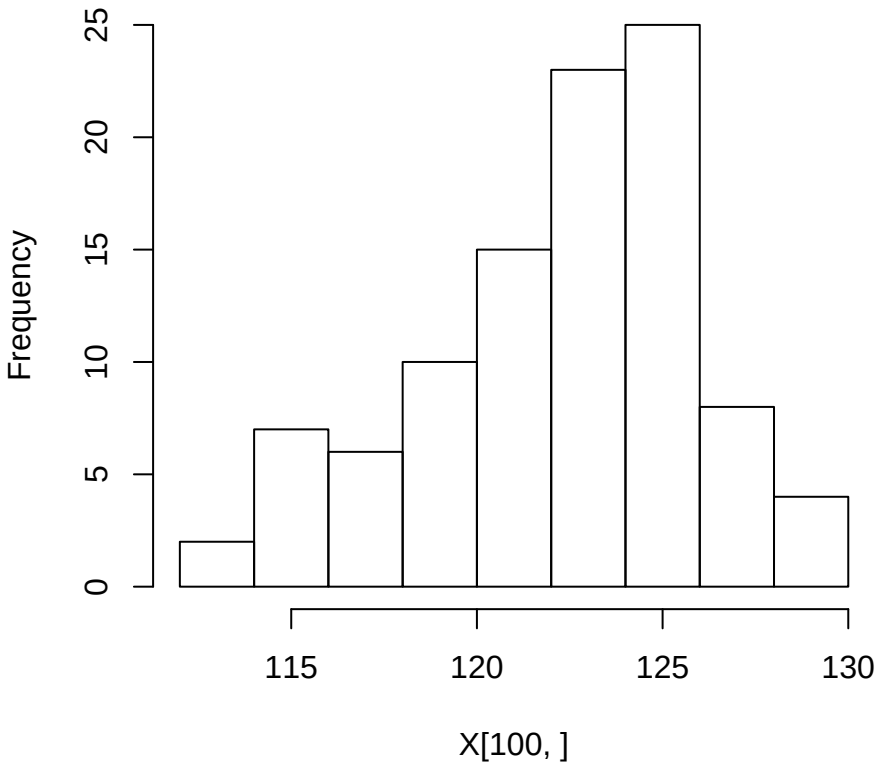
Assignment 2

Now the above assignment was to be redone with different standard deviations for the emission model. First the standard deviation was increased to 5, then to 50. By doing this, we introduce more uncertenty, which makes it harder to identify the correct states. This is shown below.

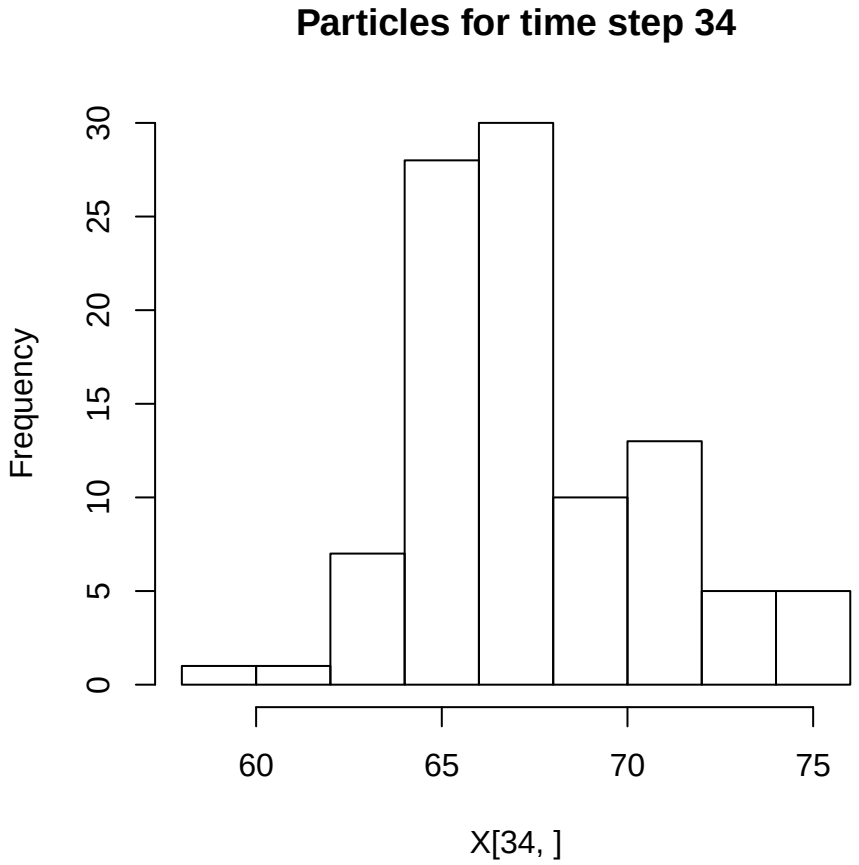
The particles when standard deviation is 5.



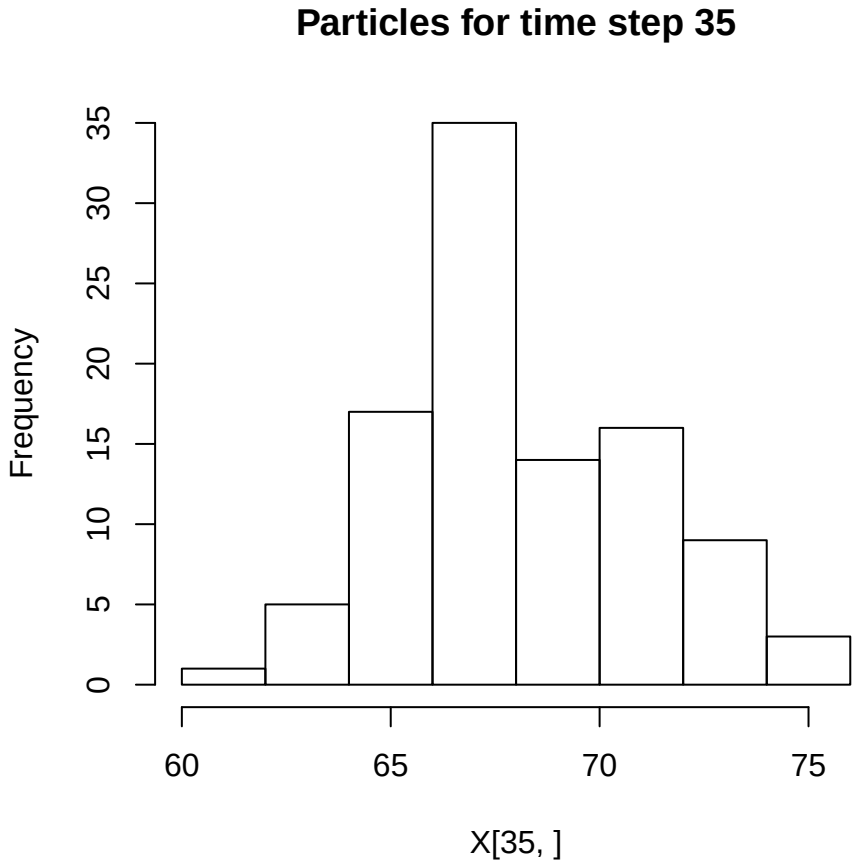

```
## [1] "Expected state for time step 1: 31.6962012232346"
## [1] "True state for time step 1: 28.7577520124614"
```




```
## [1] "Expected state for time step 100: 122.177833504923"
## [1] "True state for time step 100: 117.38417628889"
```

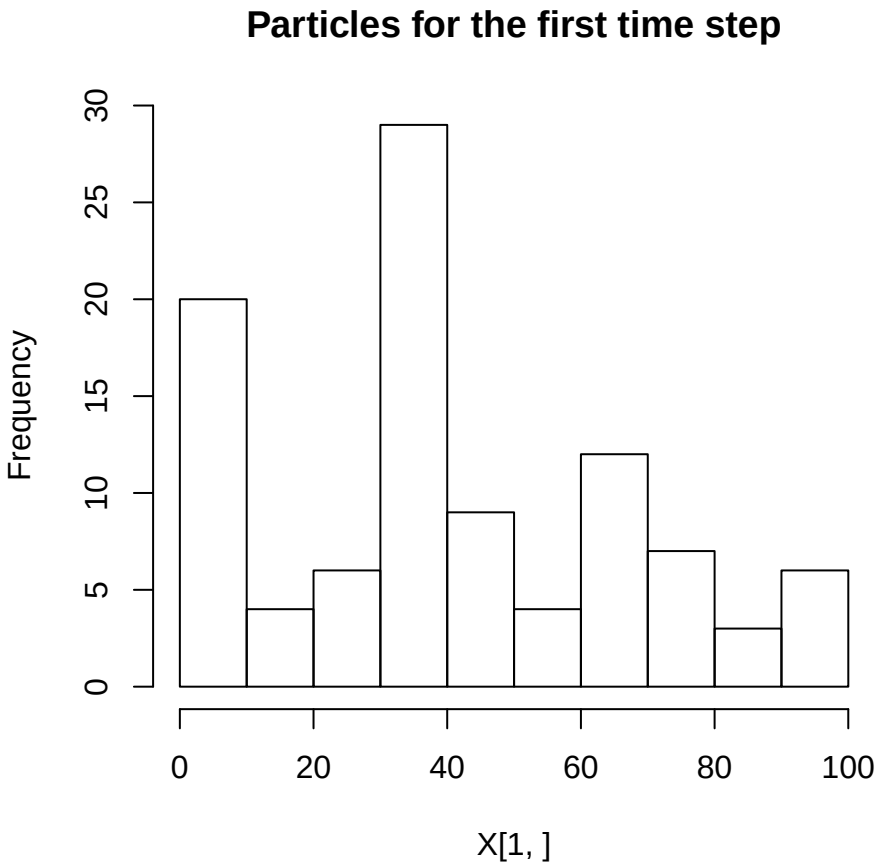



```
## [1] "Expected state for time step 34: 67.4970433247885"
## [1] "True state for time step 34: 62.7955988671768"
```

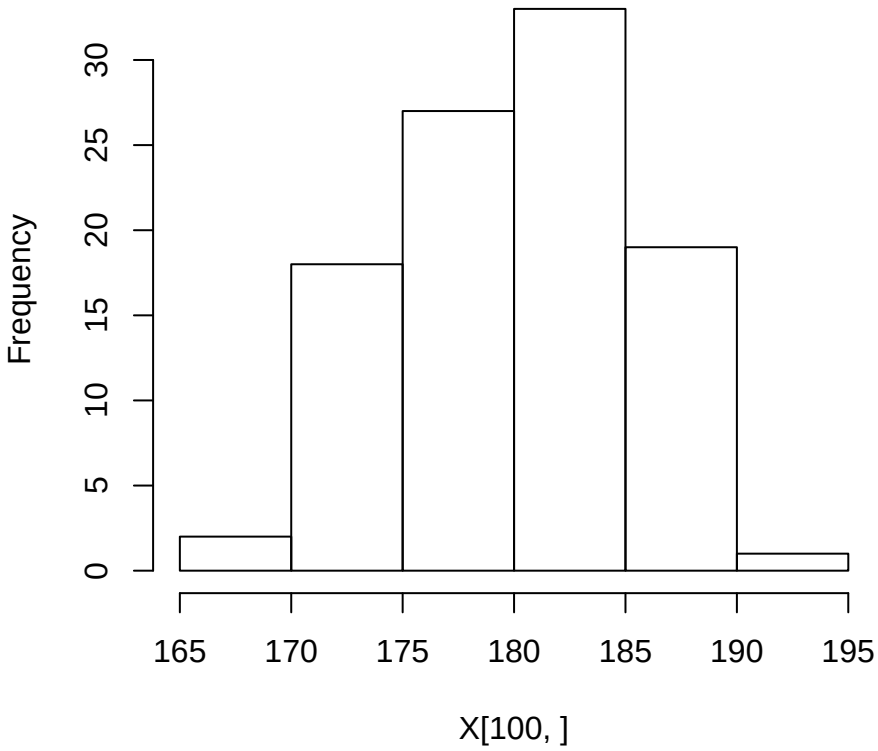



```
## [1] "Expected state for time step 35: 68.3244233728647"
## [1] "True state for time step 35: 63.8332817145742"
```

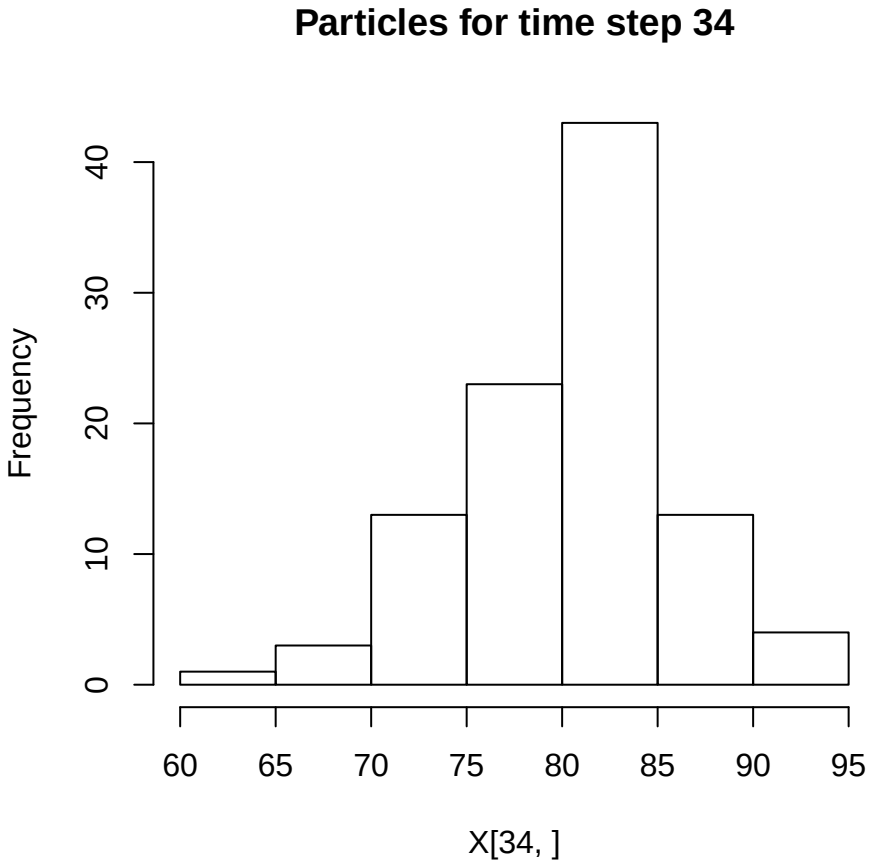
The particles for when the standrd deviation is 50.



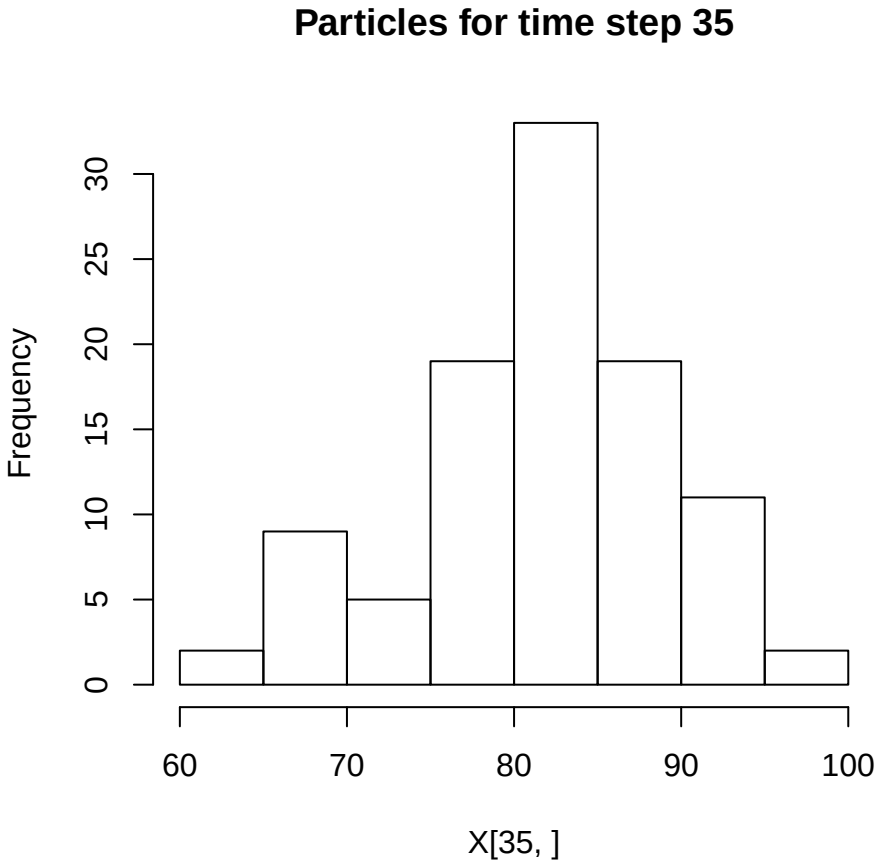

```
## [1] "Expected state for time step 1: 39.5742491482157"
## [1] "True state for time step 1: 28.7577520124614"
```




```
## [1] "Expected state for time step 100: 180.123538237553"
## [1] "True state for time step 100: 117.38417628889"
```



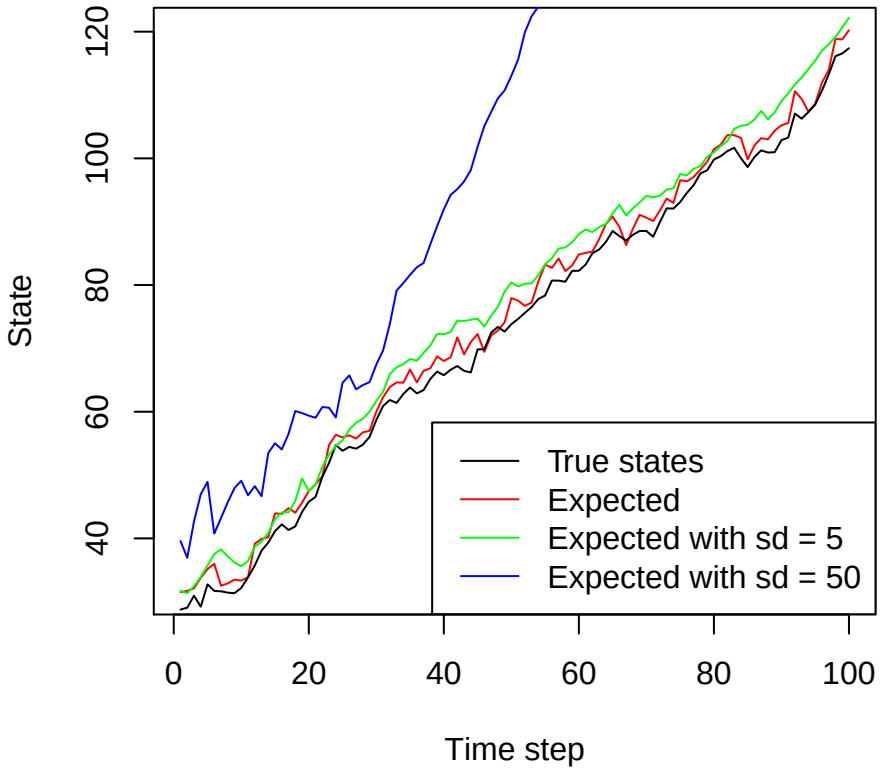

```
## [1] "Expected state for time step 34: 80.307289732634"
## [1] "True state for time step 34: 62.7955988671768"
```




```
## [1] "Expected state for time step 35: 81.6073204595691"
```

```
## [1] "True state for time step 35: 63.8332817145742"
```

Below is a graph showing the expected particles for all time steps and the different MSE for when tha standard deviation is 5 and 50.



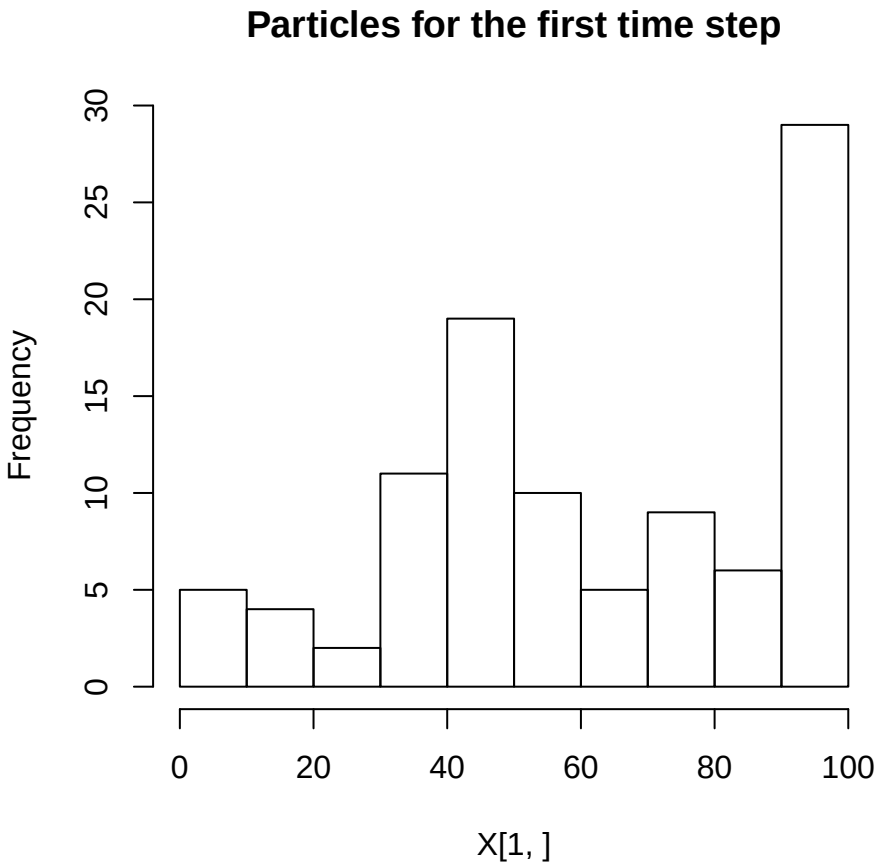

```
## [1] "MSE for sd = 5: 20.6584713266299"
```

```
## [1] "MSE for sd = 50: 1693.35438081811"
```

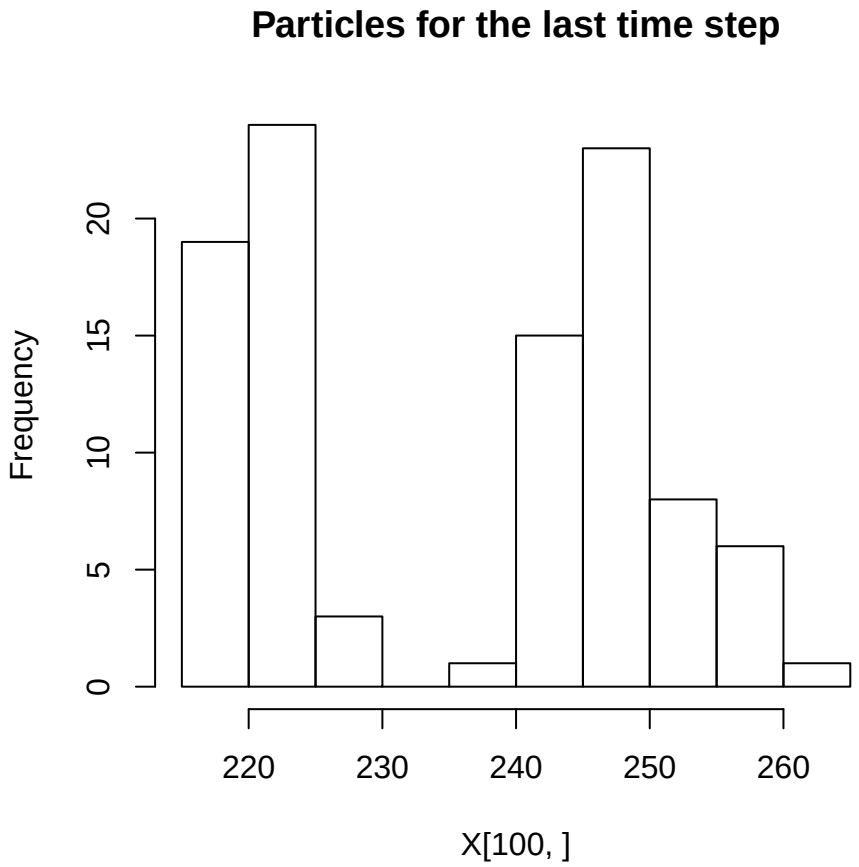
Compared to the first assignment, we see in the histograms that the mass is much more spread out across different states. This is really prominent when the standard deviation is equal to 50, meaning that the filter is really uncertain in this case. The graph above also shows that when the standard deviation is 50, the filter diverges, this can be explained by the fact that the filter becomes more and more uncertain and after a while, it has no idea what the true state is.

Assignment 3

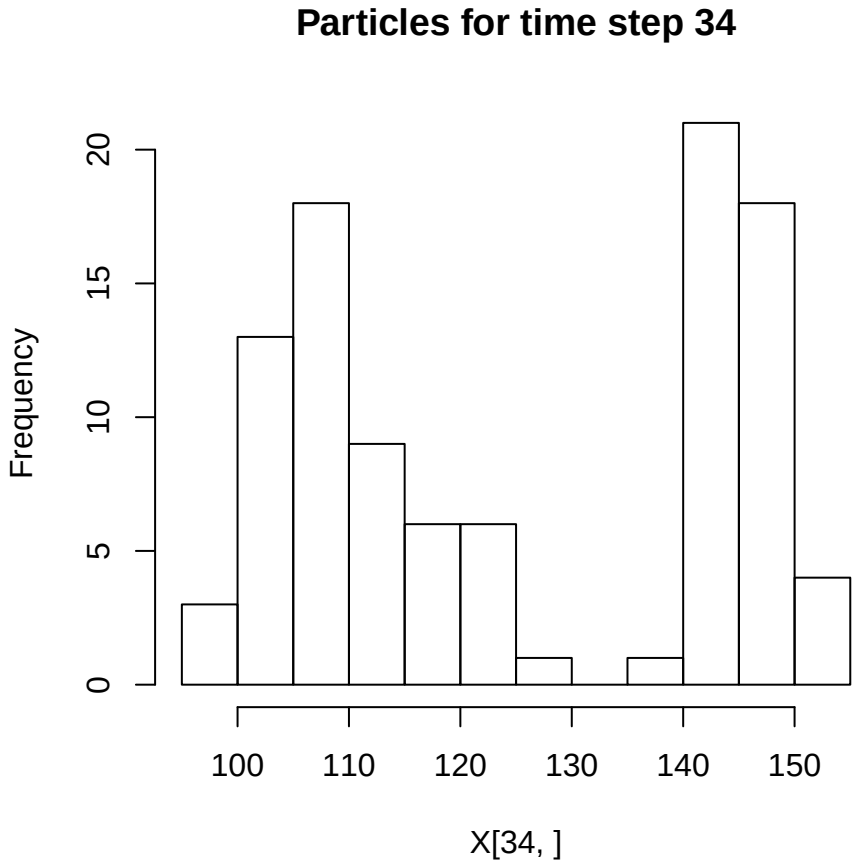
In this assignment, we were to show and explain what happens when the weights are constant and set to 1. This means that we see all particles as equally probable. This means that even bad particles will be taken into account even if they are not part of the true distribution. This is shown in the histograms as well as in the graph below.



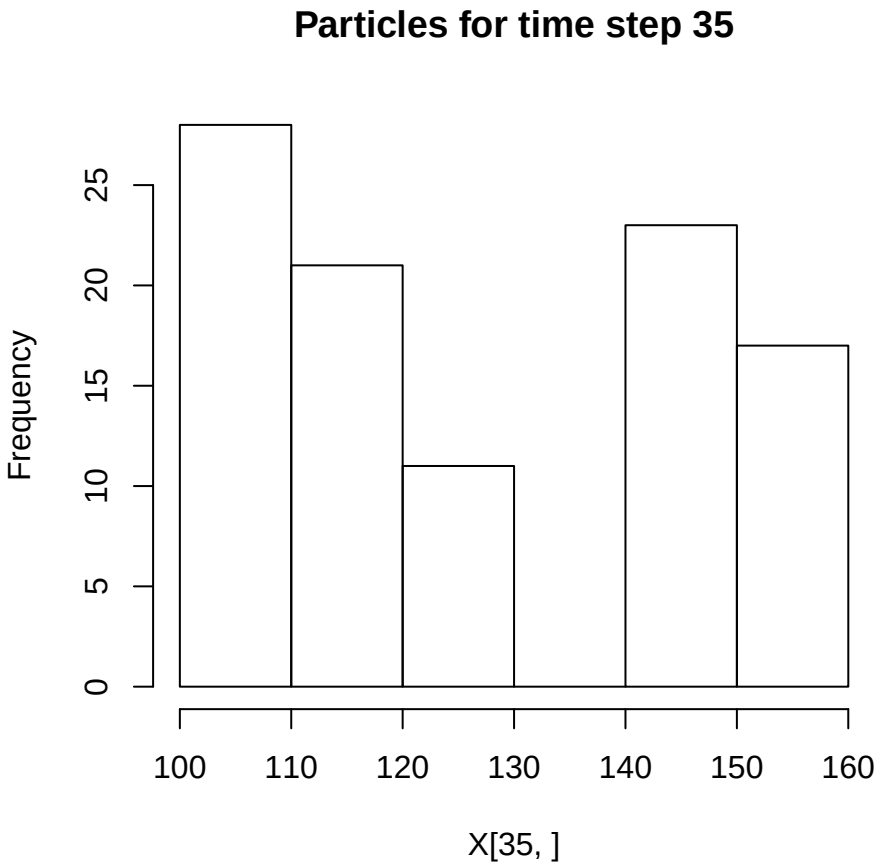

```
## [1] "Expected state for time step 1: 61.0496911560912"
## [1] "True state for time step 1: 28.7577520124614"
```



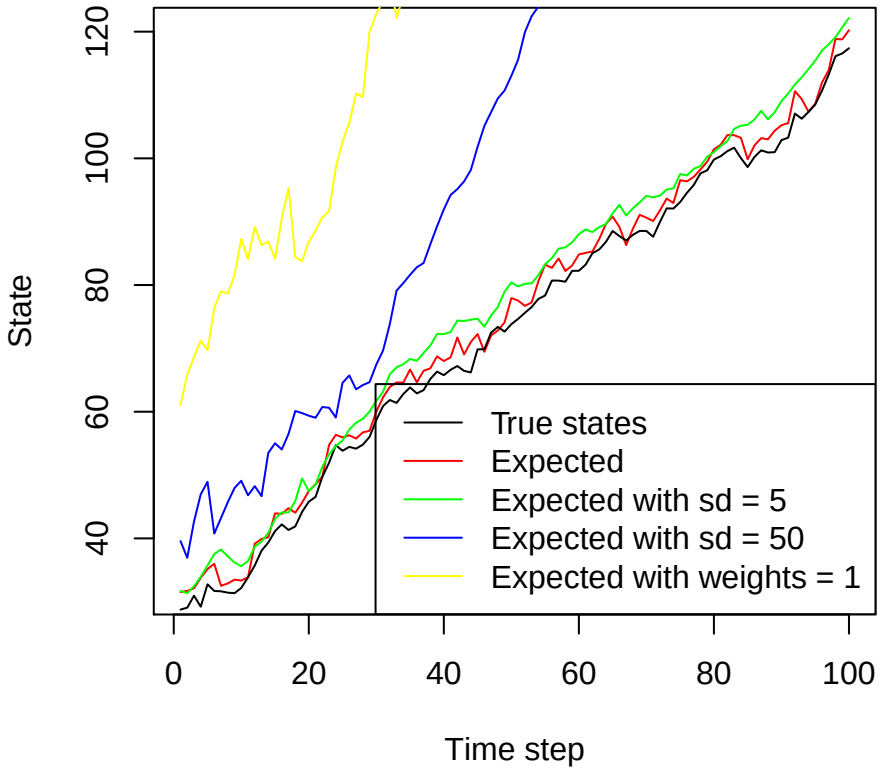

```
## [1] "Expected state for time step 100: 235.516908205982"
## [1] "True state for time step 100: 117.38417628889"
```




```
## [1] "Expected state for time step 34: 125.72204453717"
## [1] "True state for time step 34: 62.7955988671768"
```




```
## [1] "Expected state for time step 35: 126.956424501068"
## [1] "True state for time step 35: 63.8332817145742"
```




```
## [1] "MSE for wheights = 1:  6368.78526945243"
```

We can see that this is really bad and that then expected states for this cases diverges rapidly and is not even close to the true states.

Appendix - Code

```
####### Innit ######
set.seed(123)
z1 = runif(1,0,100)
T = 100
M = 100
sd_emission = 1
####### Functions ######
transition = function(z,sd = 1){
  comp = sample(0:2,1)
  return (rnorm(1,z+comp,sd))
}

emission = function(z,sd = sd_emission){
  comp = sample(-1:1,1)
  return (rnorm(1,z+comp,sd))
}

density.emission = function(z,x,sd = sd_emission){
  p = dnorm(x,z,sd) + dnorm(x,z-1,sd) + dnorm(x,z+1,sd)
  p = sum(p)
  return(p/3)
```

```
}

particle_filter = function(weight = TRUE){
  w = rep(1,100)
  particles = runif(M,0,100)
  expected = rep(0,100)
  X = matrix(NA,nrow = T, ncol = M)
  for (t in 1:T){
    if(weight){
      for (m in 1:M){
        w[m] = density.emission(z = particles[m],x = observations[t])
      }
    }
    w = w / sum(w)
    for(m in 1:M){
      particle = sample(particles,size = 1, replace = TRUE, prob = w)
      particles[m] = transition(particle)
    }
    X[t,] = particles
    tmp = NULL
    for(m in 1:M){
      tmp[m] = w[m]*particles[m]
    }
    expected[t] = sum(tmp)
  }
  return (list("Expected" = expected,"Particles" = X))
}

####### Assignment 1 ######
#SSM
states = rep(0,T)
observations = rep(0,T)
states[1] = z1

observations[1] = emission(states[1])
for (t in 2:T){
  states[t] = transition(states[t-1])
  observations[t] = emission(states[t])
}

kalman = particle_filter()
expected = kalman$Expected
X = kalman$Particles

hist(X[1,])
hist(X[100,])
hist(X[34,])
hist(X[35,])

plot(1:T,expected,type = "l", col = "red")
lines(1:T,states, col = "black")
mean((states-expected)^2)
```

```
###### Assignment 2 ######
sd_emission = 5
expected.5 = particle_filter()$Expected
lines(1:T,expected.5,col = "green")
mean((states-expected.5)^2)

sd_emission = 50
expected.50 = particle_filter()$Expected
lines(1:T,expected.50,col = "blue")
mean((states-expected.50)^2)

###### Assignment 3 ######
sd_emission = 1
expected.w = particle_filter(weight = FALSE)$Expected
lines(1:T,expected.w,col = "yellow")
mean((states-expected.w)^2)

legend("bottomright", col = c("black", "red", "green", "blue", "yellow"),
       legend = c("True states", "Expected", "Expected with sd = 5",
                  "Expected with sd = 50", "Expected with weights = 1"),
       lty = 1)
```

Lab 4

Emil Luusua

10/16/2019

2.1. Implementing GP Regression

(1)

The function posteriorGP is implemented as it is stated in (Rasmussen and Willams, 2005).

```
posteriorGP <- function(X, y, XStar, sigmaNoise, k) {
  n <- length(X)

  K <- matrix(NA, n, n)
  for(row in 1:n) {
    for(col in 1:n) {
      K[row, col] <- k(X[row], X[col])
    }
  }

  L <- t(chol(K + sigmaNoise^2 * diag(n)))
  alpha <- solve(t(L), solve(L, y))

  m <- length(XStar)
  mean <- rep(0, m)
  variance <- rep(0, m)
  for(i in 1:m) {
    kStar <- matrix(NA, n, 1)
    for(row in 1:n) {
      kStar[row] <- k(X[row], XStar[i])
    }

    mean[i] <- t(kStar) %*% alpha
    v <- solve(L, kStar)
    variance[i] <- k(XStar[i], XStar[i]) - t(v) %*% v
  }

  return(list(mean=mean, variance=variance))
}
```

The kernel function that computes the covariance between two x values is implemented, as well as a function for modelling given observations with a Gaussian Process and plotting the resulting mean with pointwise 95% confidence intervals. The observations are plotted in red.

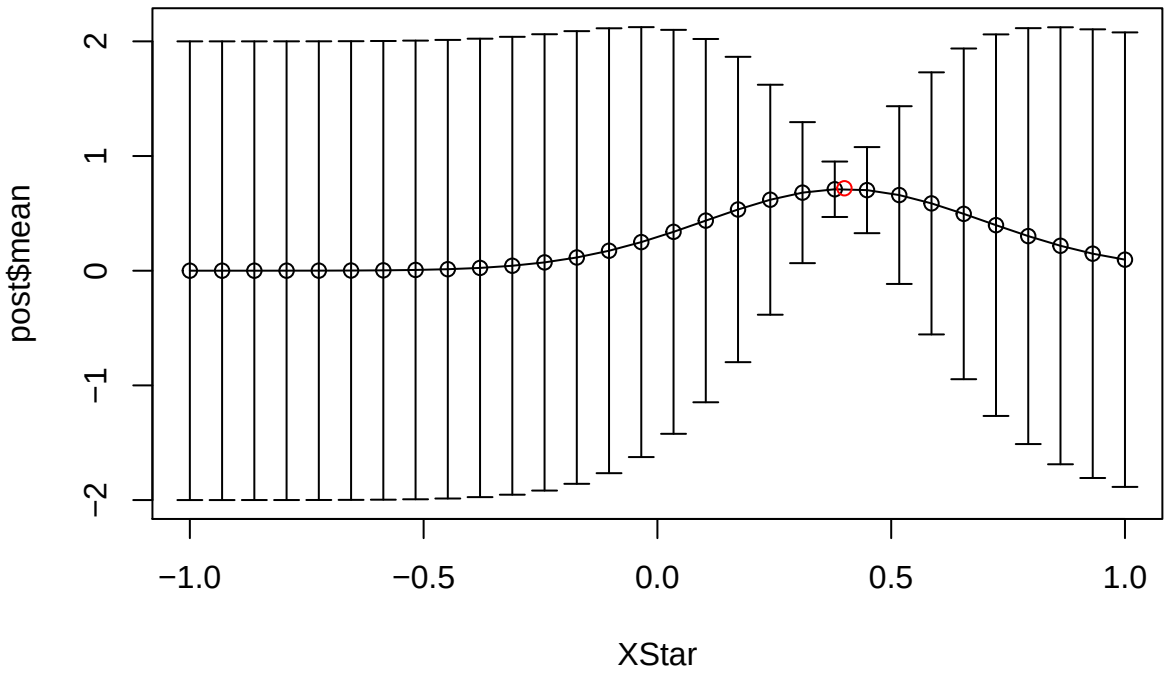
```
SquaredExpKernel <- function(sigmaF, l) {
  k <- function(x1, x2) {
    return(sigmaF^2 * exp(-0.5 * (((x1 - x2) / l)^2)))
  }
  class(k) <- 'kernel'

  return(k)
}
```

```
plotGP <- function(X, y, k) {
  XStar <- seq(-1,1,length=30)
  sigmaNoise <- 0.1

  post <- posteriorGP(X, y, XStar, sigmaNoise, k)
  plotCI(XStar, post$mean, ui=post$mean + 2 * sqrt(post$variance), li=post$mean - 2 * sqrt(post$varianc
  lines(XStar, post$mean)
  points(X, y, col="red")
}
```

(2)

Run the model for a single observation with $\sigma_f = 1$ and $l = 0.3$. It can be seen that for x values far away from the observation, the model does not know the correct value for f_*, so the variance is high and it converges to the prior mean of 0. This is because the kernel is set to result in low covariance between observations far from eachother.

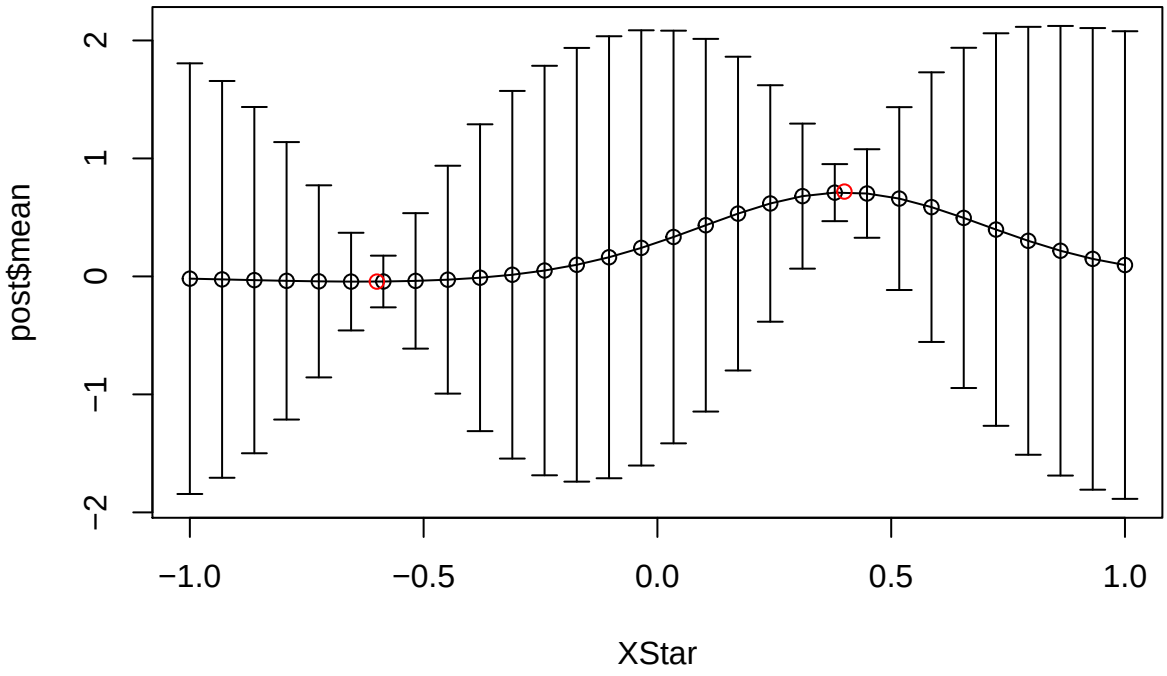
```
sigmaF <- 1
l <- 0.3
k <- SquaredExpKernel(sigmaF, l)

X <- c(0.4)
y <- c(0.719)

plotGP(X, y, k)
```



(3)

Updating the posterior model with a new observation is equivalent to directly updating the prior with both observations. This is done with the following result, again following the same pattern of converging on the prior mean far away from the observations. Since the new observation coincides with the prior mean, f_* does not change but the variance around the new observation is significantly lower.

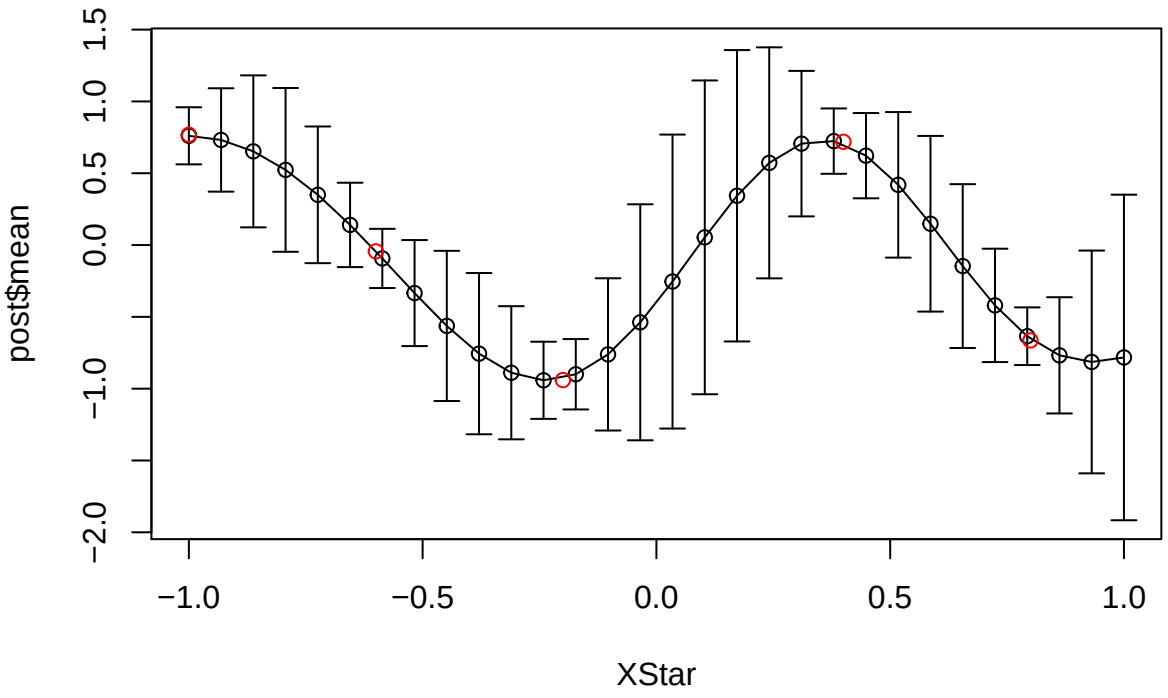
```
X <- c(0.4, -0.6)
y <- c(0.719, -0.044)
plotGP(X, y, k)
```



(4)

The model using all 5 observations follows the same pattern as earlier ones, with the confidence intervals being greater inbetween observations. But since we now have a good spread of observations it is generally much smaller, and the function still manages to intersect all of them.

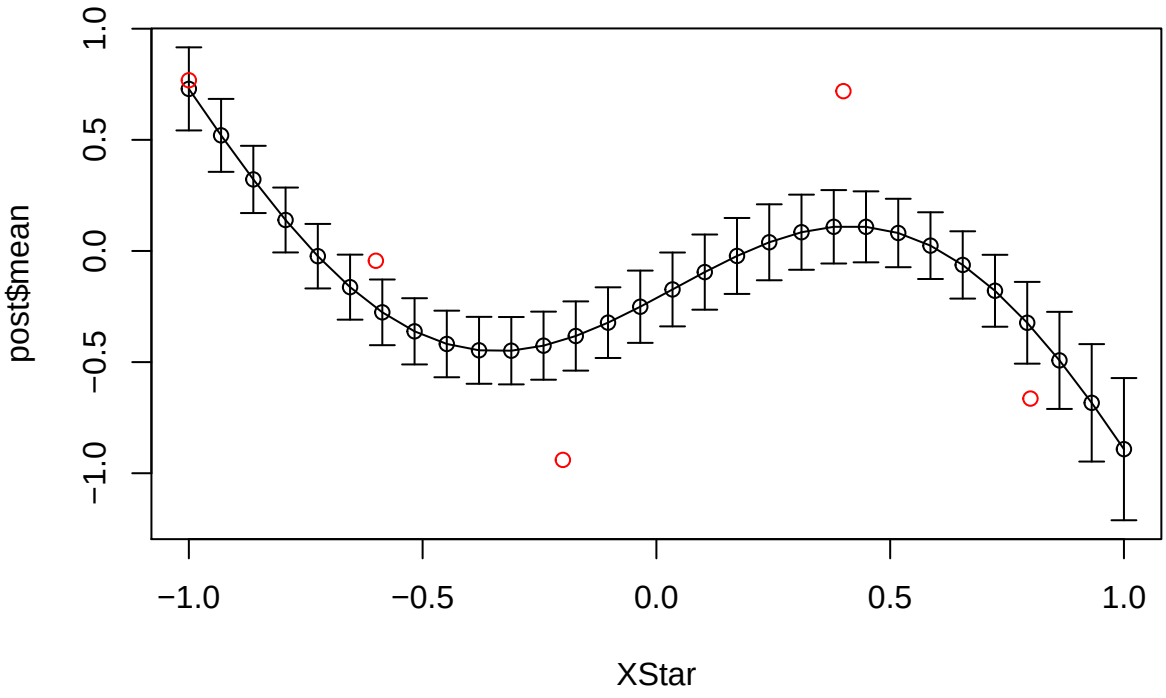
```
X <- c(-1, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
plotGP(X, y, k)
```



(5)

Running the model with a higher value for the l hyperparameter should result in a smoother posterior function. For $l = 1$, the smoothness constraint now prohibits the function from intersecting the observations points. It also results in a much more consistent confidence interval since there is not as much room for adjustment that still fit with the observations.

```
sigmaF <- 1
l <- 1
k <- SquaredExpKernel(sigmaF, l)
plotGP(X, y, k)
```



2.2. GP Regression with kernlab

Load the data set and create variables temp, time and day that contains every fifth observation. Also introduce standardized versions of these variables to account for the fact that the function gausspr standardizes both x and y. Store the mean and standard deviation of temp so that we can unscale the result in the end.

```
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.
temp <- data$temp
time <- 1:2190
day <- 1:2190
for(year in 0:5) {
  day[(year * 365 + 1): ((year + 1) * 365)] <- 1:365
}

temp <- temp[c(TRUE, FALSE, FALSE, FALSE, FALSE)]
time <- time[c(TRUE, FALSE, FALSE, FALSE, FALSE)]
day <- day[c(TRUE, FALSE, FALSE, FALSE, FALSE)]

timeScaled <- scale(time)
dayScaled <- scale(day)
tempScaled <- scale(temp)
tempMean <- attr(tempScaled, 'scaled:center')
tempStd <- attr(tempScaled, 'scaled:scale')
```

(1)

The squared exponential kernel function defined earlier is reused with kernlab. Run it with some dummy inputs to validate that it works correctly.

```
SEkernel <- SquaredExpKernel(sigmaF, l)
SEkernel(1, 2)
```

```
## [1] 0.6065307
```

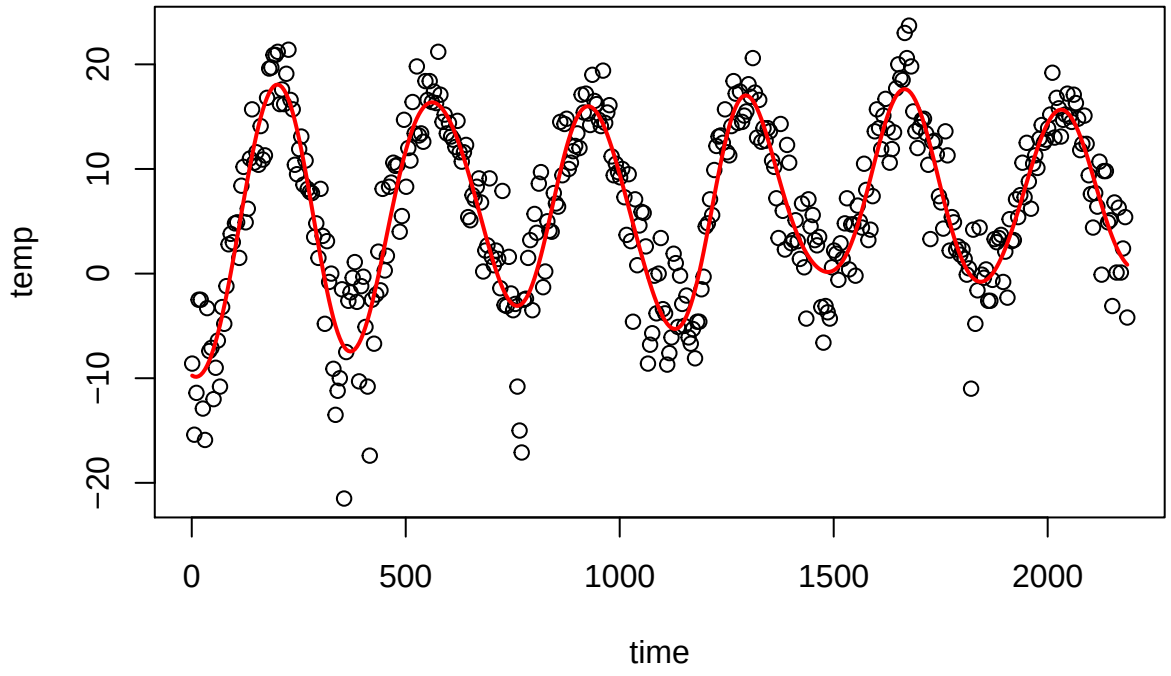
```
X <- c(1, 3, 4)
XStar <- c(2, 3, 4)
kernelMatrix(SEkernel, x = X, y = XStar)
```

```
## An object of class "kernelMatrix"
##           [,1]      [,2]      [,3]
## [1,] 0.6065307 0.1353353 0.0111090
## [2,] 0.6065307 1.0000000 0.6065307
## [3,] 0.1353353 0.6065307 1.0000000
```

(2)

The temperature is modeled as a Gaussian Process of the absolute time. Here the posterior mean is plotted, showing that it does a good job of following the observations while maintaining good smoothness.

```
sigmaf <- 20
ell <- 0.2

polyFit <- lm(tempScaled ~  timeScaled + I(timeScaled^2))
sigmaNoise <- sd(polyFit$residuals)

GPfit <- gausspr(timeScaled, tempScaled, kernel = SquaredExpKernel, kpar = list(sigmaF <- sigmaf, l <- 
meanPred1 <- predict(GPfit, timeScaled)
plot(time, temp)
lines(time, meanPred1 * tempStd + tempMean, col='red', lwd = 2)
```

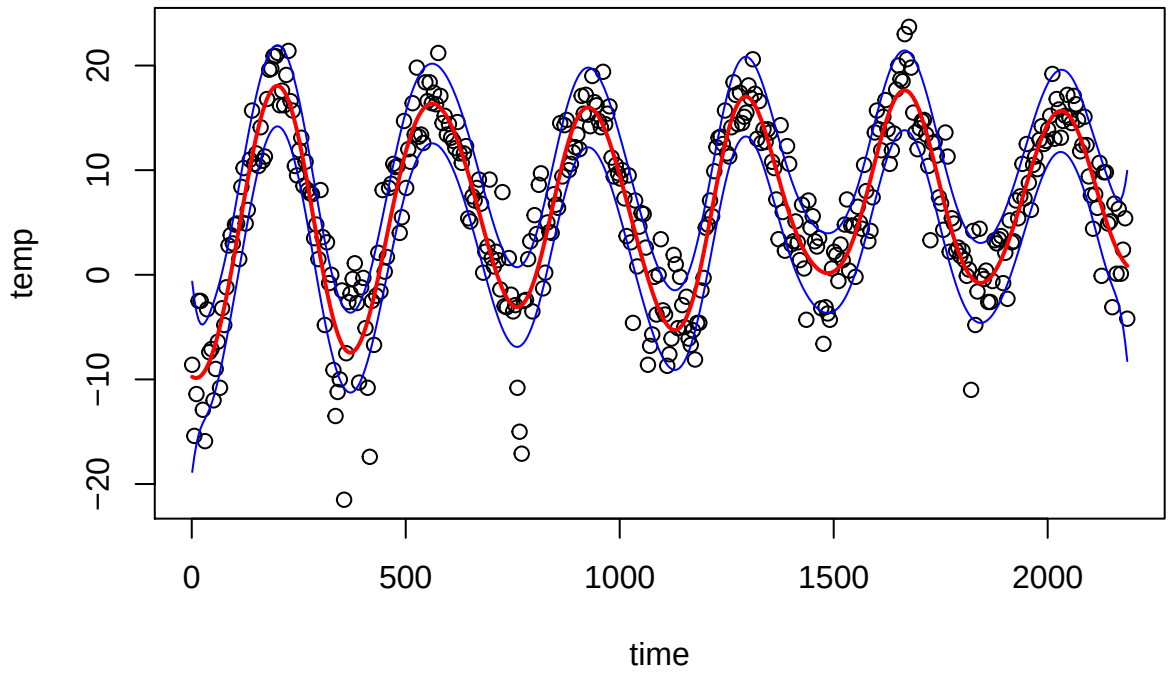


(3)

The 95% probability bands for f are plotted, calculated from the posterior variance which can be computed by the function posteriorGP implemented earlier. The mean is plotted in red and the bands in blue.

```
SEkernel <- SquaredExpKernel(sigmaf, ell)
var1 <- posteriorGP(X = timeScaled,
                    y = tempScaled,
```

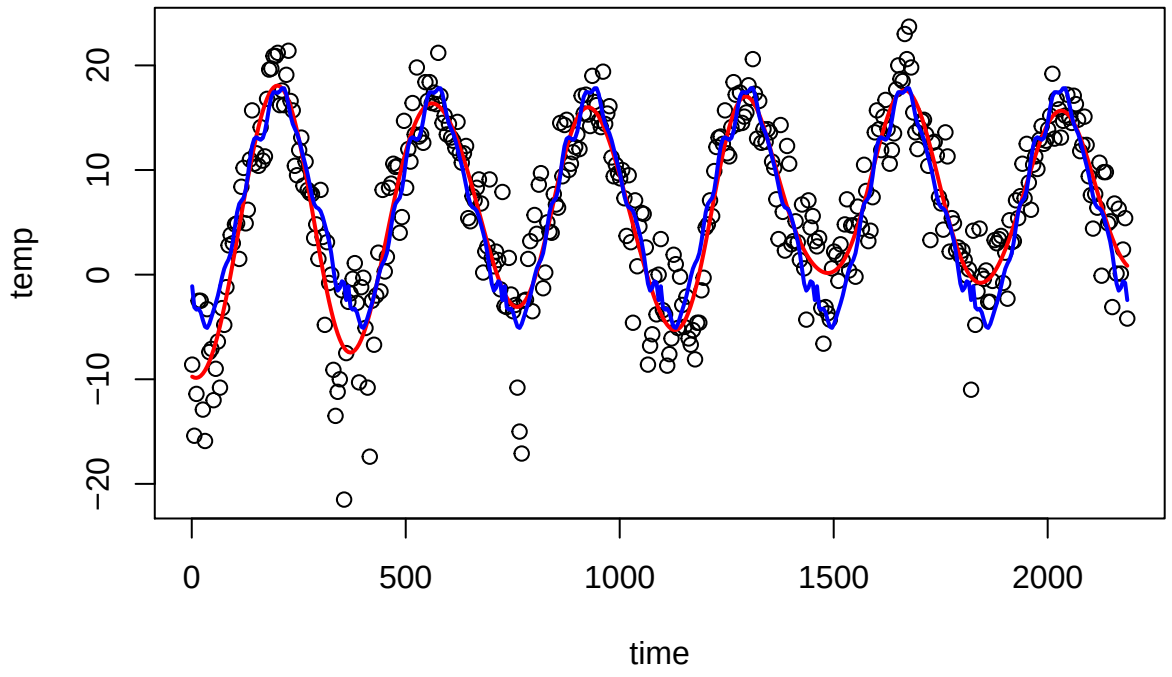
```
                  XStar = timeScaled,
                  sigmaNoise = sigmaNoise,
                  k = SEkernel)$variance

plot(time, temp)
lines(time, meanPred1 * tempStd + tempMean, col='red', lwd = 2)
lines(time, (meanPred1 - 1.96*sqrt(var1)) * tempStd + tempMean, col = "blue")
lines(time, (meanPred1 + 1.96*sqrt(var1)) * tempStd + tempMean, col = "blue")
```



(4)

The temperature is modelled as a Gaussian Process depending on the day of the year. This results in the predictions becoming periodic, i.e. the same for each year and no global trend or individual deviations within a single year is taken into account. It obviously has an easier time picking up on the cyclical nature of the data, but since the previous model was already capable of doing this, the day model seems to be strictly worse.

```
polyFit <- lm(tempScaled ~  dayScaled + I(dayScaled^2))
sigmaNoise <- sd(polyFit$residuals)

GPfit <- gausspr(dayScaled, tempScaled, kernel = SquaredExpKernel, kpar = list(sigmaF <- sigmaf, l <- el
meanPred2 <- predict(GPfit, dayScaled)

plot(time, temp)
lines(time, meanPred1 * tempStd + tempMean, col='red', lwd = 2)
lines(time, meanPred2 * tempStd + tempMean, col='blue', lwd = 2)
```



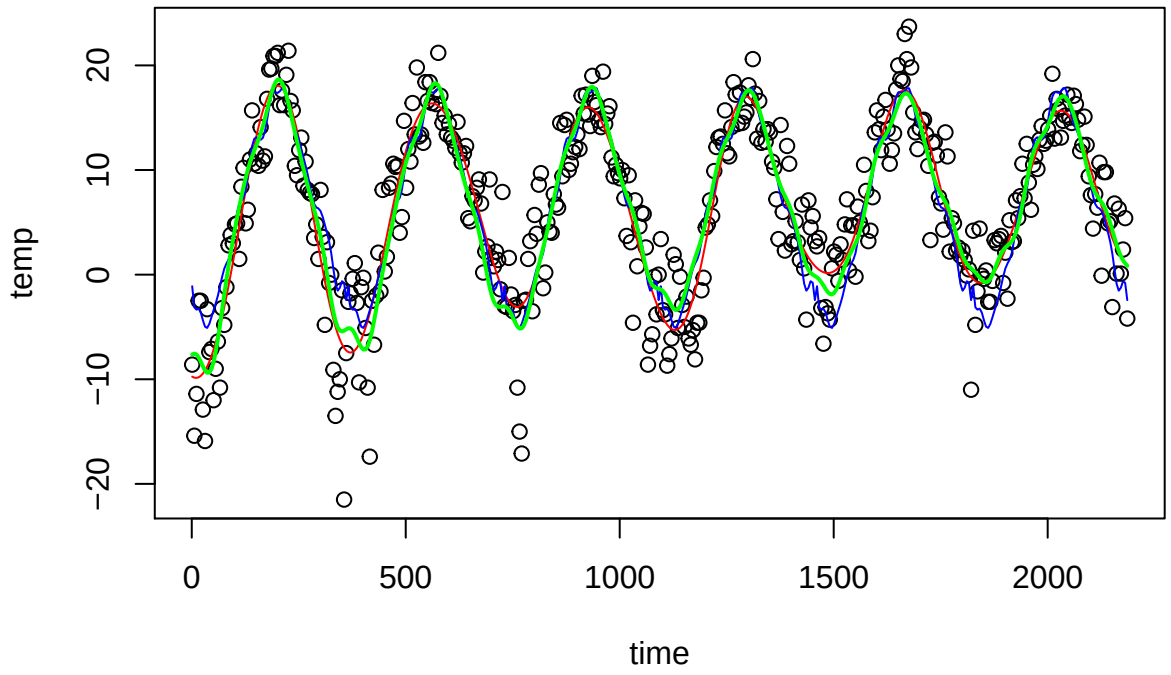
(5)

The periodic kernel is implemented.

```
PeriodicKernel <- function(sigmaF, l1, l2, d) {
  k <- function(x1, x2) {
    return(sigmaF^2 * exp(-0.5 * (((x1 - x2)/l2)^2)) * exp(-2 * (sin(pi / d * abs(x1 - x2))/l1)^2))
  }
  class(k) <- 'kernel'

  return(k)
}
```

A model utilizing the periodic kernel (plotted in green) is compared to the previous models. Theoretically, it should be able to pick up both on the cyclical pattern in the data and the global trend, but with this particular hyperparameter configuration is seems similar to the first model, except it is less smooth.

```
ell1 <- 1
ell2 <- 10
d <- 365 / sd(time)

polyFit <- lm(tempScaled ~  timeScaled + I(timeScaled^2))
sigmaNoise <- sd(polyFit$residuals)

GPfit <- gausspr(timeScaled, tempScaled, kernel = PeriodicKernel, kpar = list(sigmaF <- sigmaf, l1 <- el
meanPred3 <- predict(GPfit, timeScaled)

plot(time, temp)
lines(time, meanPred1 * tempStd + tempMean, col='red')
lines(time, meanPred2 * tempStd + tempMean, col='blue')
lines(time, meanPred3 * tempStd + tempMean, col='green', lwd = 2)
```



2.3. GP Classification with kernlab

Data is loaded and indices for training data are sampled.

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

SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
```

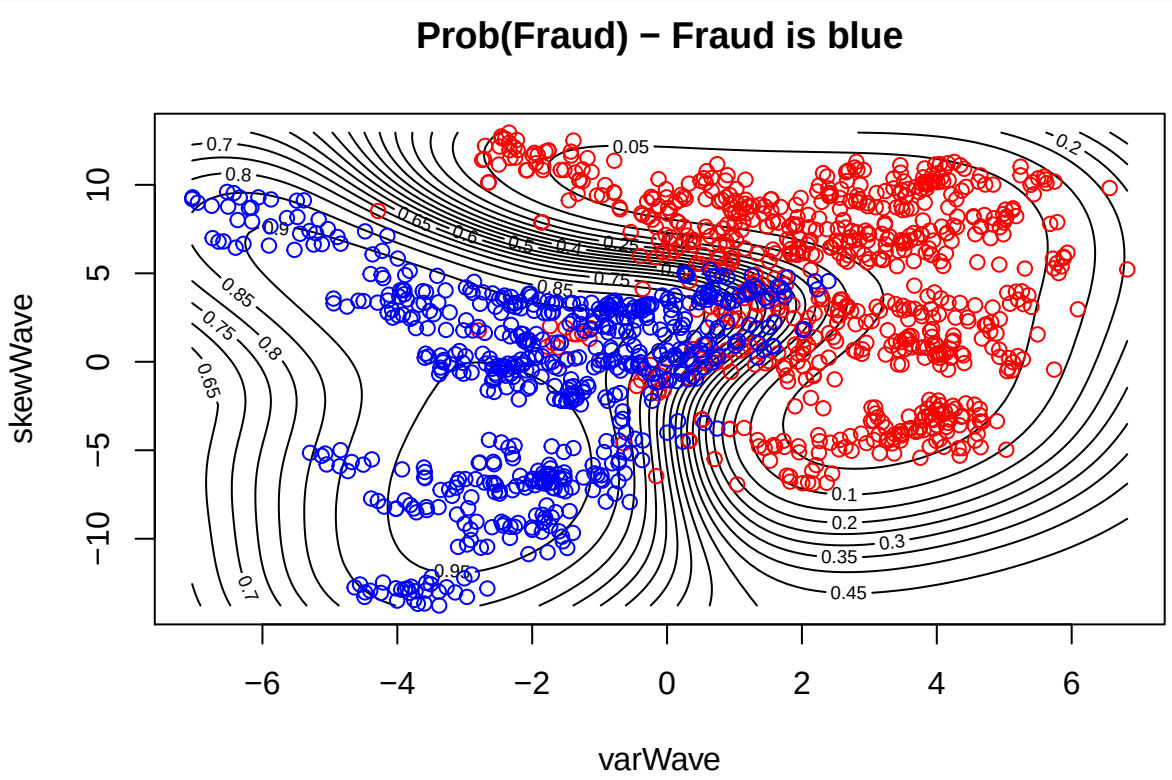
(1)

A Gaussian Process model is fitted for fraud classification using covariants varWave and skewWave. The confusion matrix with accuracy is printed, and the contour plot shows the probablity distribution for frauds over the covariant input space. The actual data points overlay the contour plots, and a clear correspondece between these and the contours can be seen.

```
GPfit <- gausspr(fraud ~ varWave + skewWave, data=data[SelectTraining, ])

## Using automatic sigma estimation (sigest) for RBF or laplace kernel
confusionMatrix(table(predict(GPfit, data[SelectTraining, 1:2]), data[SelectTraining, 5]))

## Confusion Matrix and Statistics
##
##
##       0   1
##   0 503  18
##   1  41 438
##
##                Accuracy : 0.941
##                  95% CI : (0.9246, 0.9548)
##     No Information Rate : 0.544
##     P-Value [Acc > NIR] : < 2.2e-16
```

```
##
##                   Kappa : 0.8816
##
##  Mcnemar's Test P-Value : 0.004181
##
##             Sensitivity : 0.9246
##             Specificity : 0.9605
##          Pos Pred Value : 0.9655
##          Neg Pred Value : 0.9144
##              Prevalence : 0.5440
##          Detection Rate : 0.5030
##    Detection Prevalence : 0.5210
##       Balanced Accuracy : 0.9426
##
##        'Positive' Class : 0
##
```

```
x1 <- seq(min(data[,1]),max(data[,1]),length=100)
x2 <- seq(min(data[,2]),max(data[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(data)[1:2]
probPreds <- predict(GPfit, gridPoints, type="probabilities")
contour(x1,x2,matrix(probPreds[,2],100,byrow = TRUE), 20, xlab = "varWave", ylab = "skewWave", main = '
points(data[data[,5]==0,1],data[data[,5]==0,2],col="red")
points(data[data[,5]==1,1],data[data[,5]==1,2],col="blue")
```



(2)

Predictions are made on our test data which the model has not seen previously, and the confusion matrix with accuracy is printed.

```
confusionMatrix(table(predict(GPfit, data[-SelectTraining, 1:2]), data[-SelectTraining, 5]))
```

```
## Confusion Matrix and Statistics
##
##
##       0   1
##   0 199   9
##   1  19 145
##
##                Accuracy : 0.9247
##                  95% CI : (0.8931, 0.9494)
##     No Information Rate : 0.586
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8463
##
##  Mcnemar's Test P-Value : 0.08897
##
##             Sensitivity : 0.9128
##             Specificity : 0.9416
##          Pos Pred Value : 0.9567
##          Neg Pred Value : 0.8841
##              Prevalence : 0.5860
##          Detection Rate : 0.5349
##    Detection Prevalence : 0.5591
##       Balanced Accuracy : 0.9272
##
##        'Positive' Class : 0
##
```

(3)

A new model is trained using all of the 4 covariates, the resulting test accuracy is substantially better than the model only using 2 covariates. This suggests that these features hold valuable information that can be used to accurately predict frauds.

```
GPfit <- gausspr(fraud ~ varWave + skewWave + kurtWave + entropyWave, data=data[SelectTraining, ])
```

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

```
confusionMatrix(table(predict(GPfit, data[-SelectTraining, 1:4]), data[-SelectTraining, 5]))
```

```
## Confusion Matrix and Statistics
##
##
##       0   1
##   0 216   0
##   1   2 154
##
##                Accuracy : 0.9946
##                  95% CI : (0.9807, 0.9993)
##     No Information Rate : 0.586
```

```
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9889
##
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 0.9908
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9872
##              Prevalence : 0.5860
##          Detection Rate : 0.5806
##    Detection Prevalence : 0.5806
##       Balanced Accuracy : 0.9954
##
##        'Positive' Class : 0
##
```

Lab 4

Kristian Sikiric (krisi211)

Assignment 1

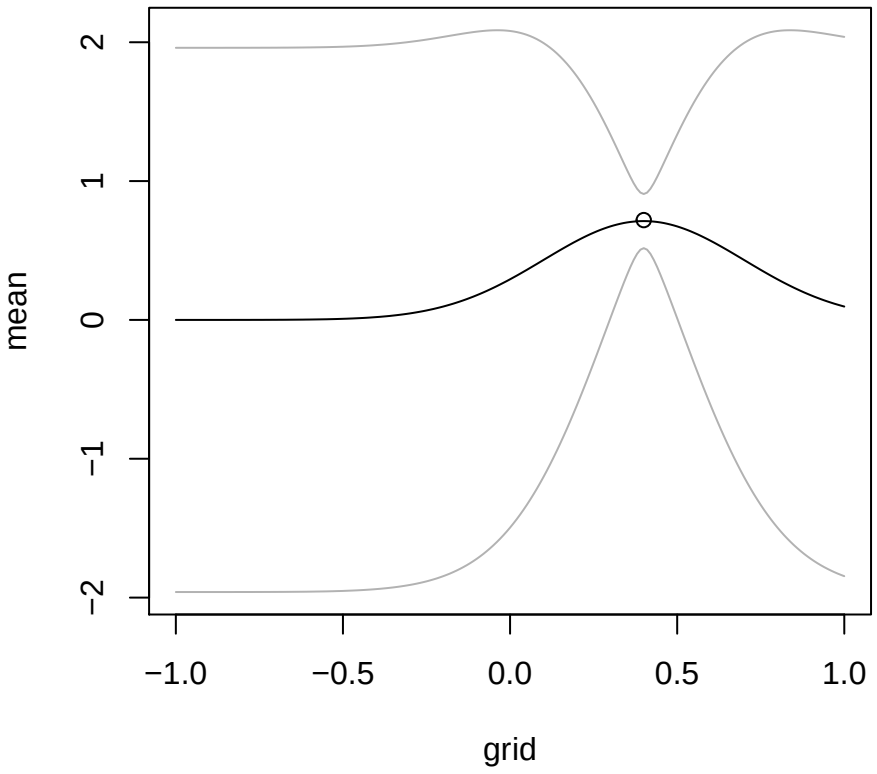
1.1

First the algorithm from Rasmussen and Willams' book to be able to do gausian regression was implemented. Below the implementation in r can be seen.

```
posteriorGP = function(X,y,XStar,hyperParam,sigmaNoise){
  n = length((X))
  K = squeredExpKernel(X,X,hyperParam[1],hyperParam[2])
  L = t(chol(K+(sigmaNoise^2)*diag(n)))
  alpha = solve(t(L),solve(L,y))
  kstar = squeredExpKernel(X,XStar,hyperParam[1], hyperParam[2])
  fbarstar = t(kstar)%*%alpha

  v = solve(L, kstar)
  V.f = squeredExpKernel(XStar,XStar,hyperParam[1], hyperParam[2])-t(v)%*%v


  return(list("Mean" = fbarstar, "Variance" = diag(V.f)))
}
```

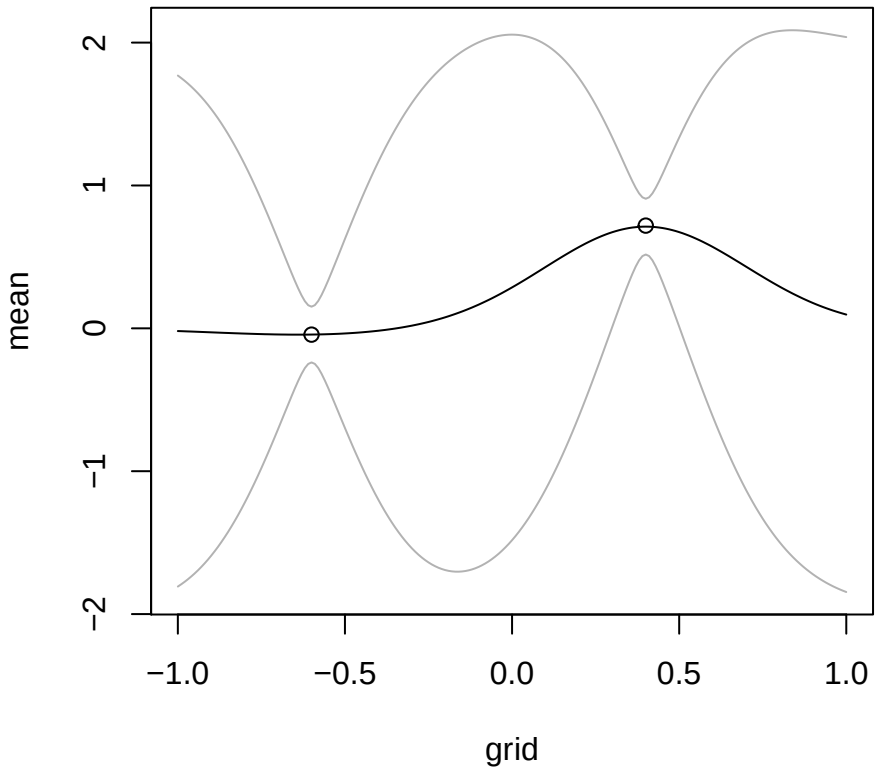
1.2

Now we were to update our prior with a single observation: $x = 0.4$ and $y = 0.719$. The hyperparameters were set to $\sigma_f = 1$ and $l = 0.3$ and we assumed $\sigma_n = 0.1$ The 95% probability bands were also calculated, see plot below.



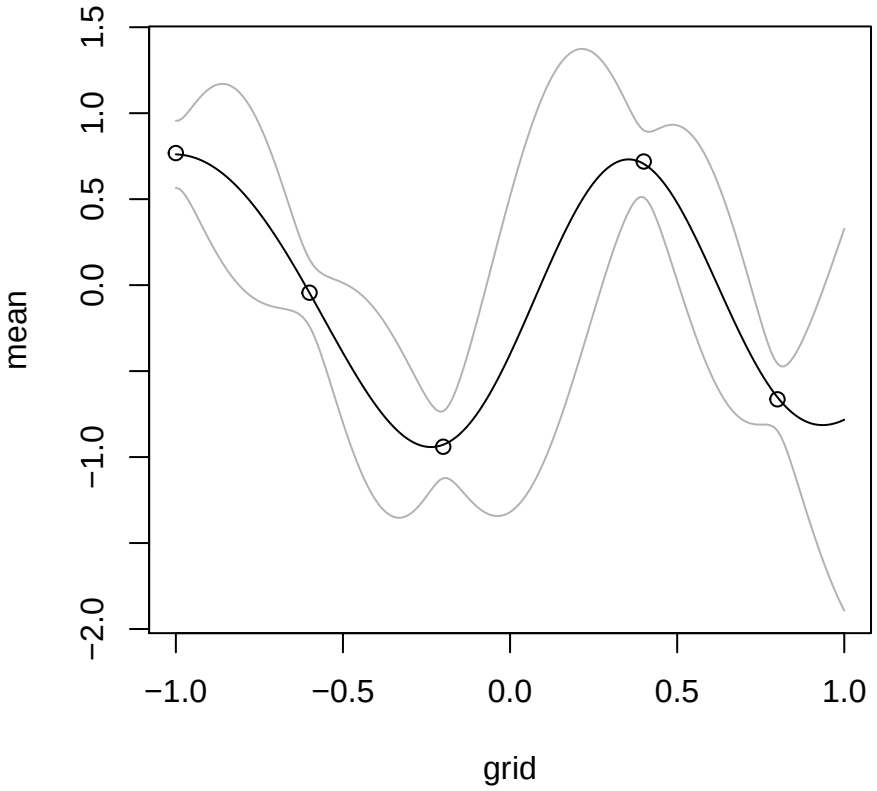
1.3

Now we were to update the postierior from 1.2 with a new observation $x = -0.6$ and $y = -0.044$. Since this is the same as updating the prior with two values at once, the values $x = (0.4, -0.6)$ and $y = (0.719, -0.044)$ was sent to the funciton. The following plot shows the mean and the 95% probability bands.



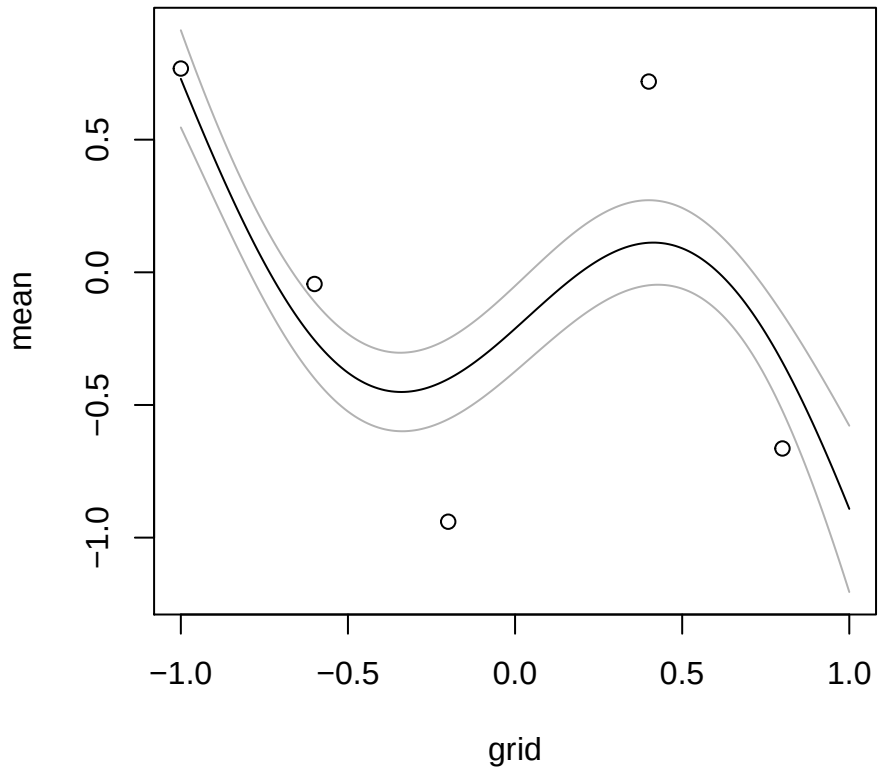
1.4

Now we were to compute the posterior with all the five data points below. $x = (-1.0, -0.6, -0.2, 0.4, 0.8)$ and $y = (0.768, -0.044, -0.940, 0.719, -0.664)$ Below is the plot of the mean and the 95% probability bands.



1.5

Now we were to repeat the last assignment but with $l = 1$. The following plot shows the resulting mean and 95% probability bands.



We see that the mean no longer goes through the five points sent to the function, as compared to assignment 1.4 were they do. The 95% probability bands are much more even than in assignment 1.4 as well. This is due to the l paramater specifying the smoothing of the curve.

Assignment 2

In this assignment we were to do GP regression with the library kernlab. The regression was to be done on temperature data for each day in a 6 year period. First the kernel was to be implemented, this is the same kernel as in assignment 1, but it is nested in a class.

```
GPkernel = function(sigmaf,ell){
  squeredExpKernel= function(x,y){
    n1 = length(x)
    n2 = length(y)
    k = matrix(NA,n1,n2)
    for (i in 1:n2){
      k[,i] = sigmaf^2*exp(-0.5*( (x-y[i])/ell)^2 )
    }
    return(k)
  }
  class(squeredExpKernel) <- "kernel"
  return(squeredExpKernel)
}
```

2.1

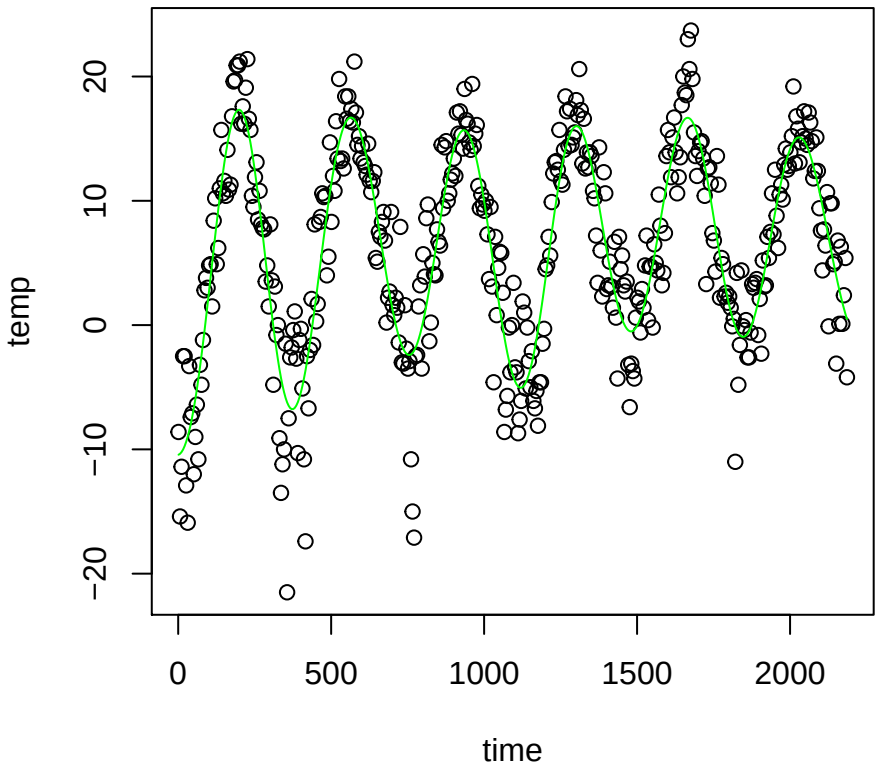
The kernel was evaluated for $sigma_f = 20$ and $l = 0.2$ giving the following result:

```
##              [,1]
## [1,] 0.001490661
```

The kernel was evaluated on the points $x = 1$ and $x' = 2$. This can be interpreted that the kernel does not affect nearby values much. This makes sense, since we are prediciting temperatures, and maybee the temperatures in one day does not affect the temperatures the next day, so it is reasonable that the kernel do not care to much about the temperatures for the next day.

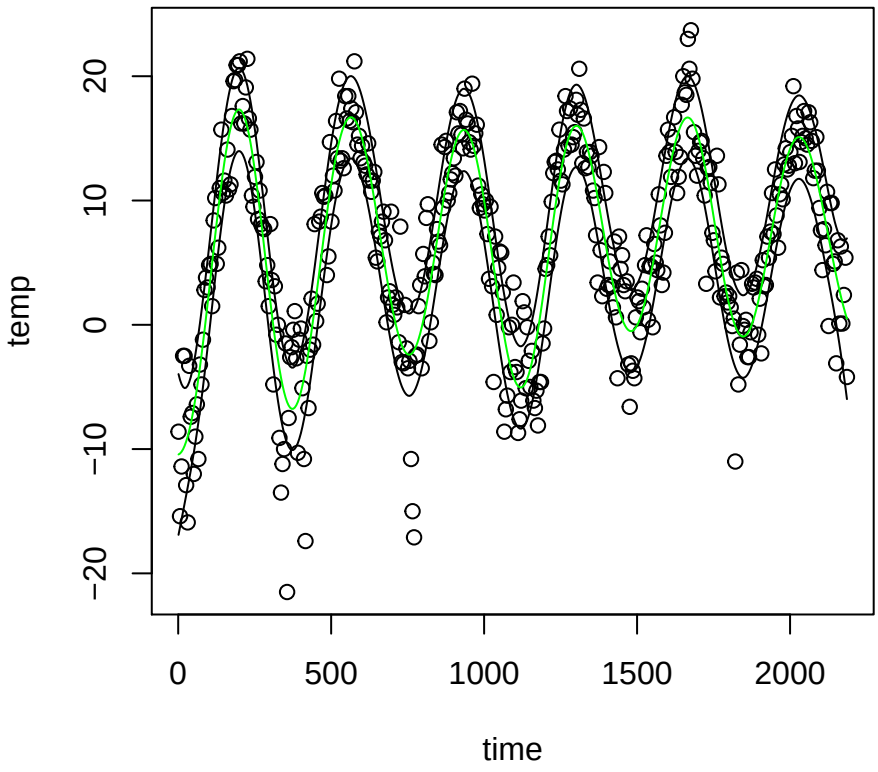
2.2

Now we were to predict the temperature given the time, were time is the number of days from the start of the dataset. $sigma_f = 20$ and $l = 0.2$ was used in the kernel, and the σ_n was calculated by first fitting a simple qudratic linear regression. The kernel from 2.1 was used in the GP regression, the following plot shows a scatter plot of the temperatures given the time, and the predicitve mean is plotted as the green line.



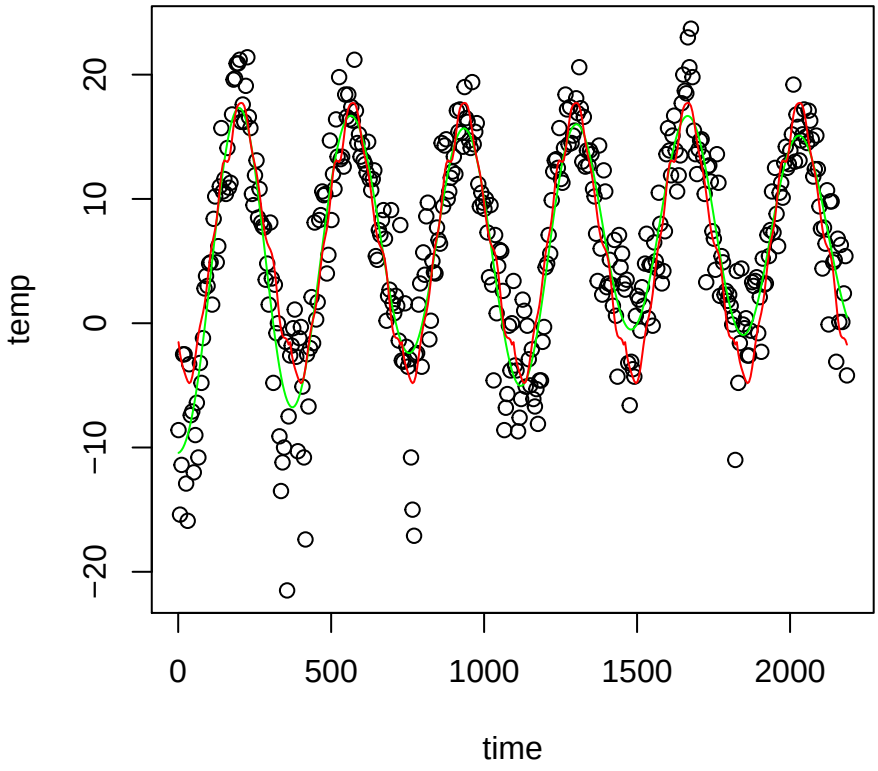
2.3

The variance was calculated using the algorithm in assignment 1. From the variance the 95% probability bands was computed and ploted, they are the black lines in the plot below.



2.4

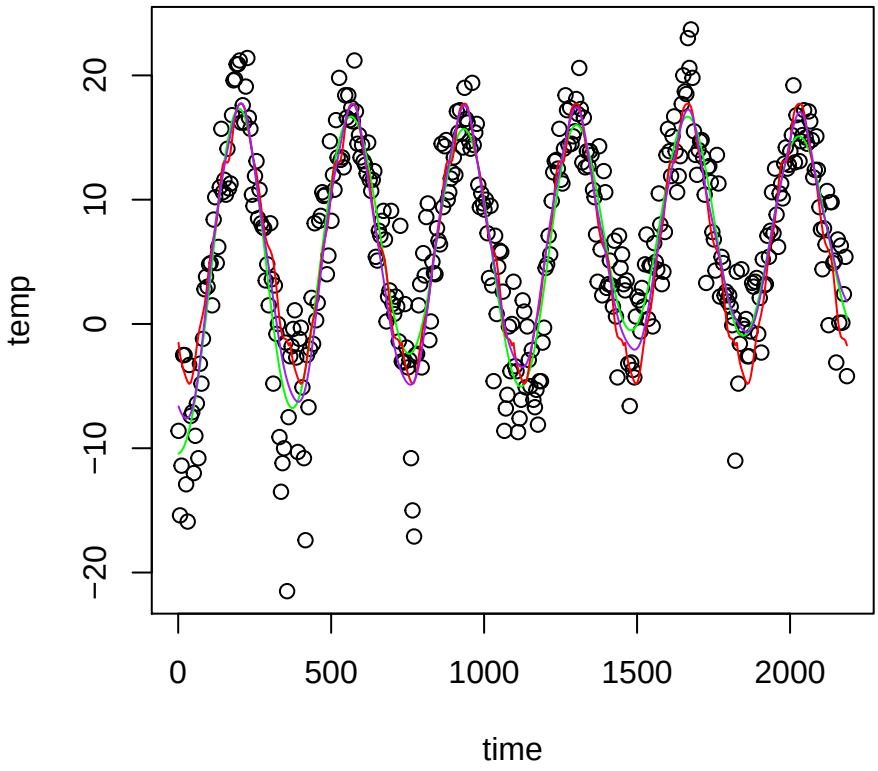
Now we were to redo assignment 2.2 but with days instead of time, were days are the days since the start of the year. The resulting predicitive mean is plotted below in red together with the mean from 2.2 (in green).



We see that there is not much difference, both models fit the data good.

2.5

Now we were to redo assignment 2.2 but with a different kernel. The kernel was given by the following function: $k(x,x') = \sigma_f^2 exp(-\frac{2*sin^2(pi|x-x'|/d)}{l_1^2})exp(-0.5\frac{|x-x'|^2}{l_2^2})$ were $\sigma_f = 20, l_1 = 1, l_2 = 10, d = 365/sd(time)$. Now a GP regression model was fitted with this kernel and plotted below in purple together with the two models from the previous assignments.



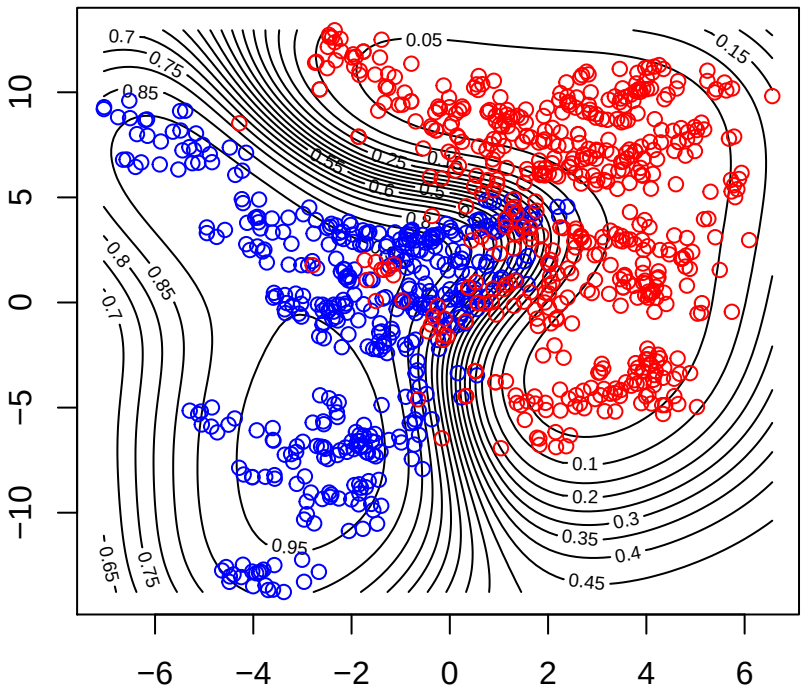
All three models looks similar and fits the data very i good in my opinion.

Assignment 3

3.1

Now we were to do classification with the kernlab library. First we were to train a model with only 2 out of 4 covariates. Below are the contour of the prediction probabilities overlayed with the target variable, fraud. The dots are colored blue when fraud is equal to 1 and red when fraud is equal to 0.

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



Below is the confusion matrix and accuracy when predicting on the train data.

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

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

3.2

Now we used the model from 3.1 and computed the confusion matrix and accuracy on the test data set.

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

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

We ee that the test accuracy is similar to the train data, showing that we did not overfit when training.

3.3

Now we were to train the model on all covariates and make predictions on the test set, below is the confusion matrix and accuracy.

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

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

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

Appendix - Code

```
###### Functions #######
posteriorGP = function(X,y,XStar,hyperParam,sigmaNoise){
  n = length((X))
  K = squeredExpKernel(X,X,hyperParam[1],hyperParam[2])
  L = t(chol(K+(sigmaNoise^2)*diag(n)))
  alpha = solve(t(L),solve(L,y))
  kstar = squeredExpKernel(X,XStar,hyperParam[1], hyperParam[2])
  fbarstar = t(kstar)%*%alpha

  v = solve(L, kstar)
  V.f = squeredExpKernel(XStar,XStar,hyperParam[1], hyperParam[2])-t(v)%*%v


  return(list("Mean" = fbarstar, "Variance" = diag(V.f)))
}

squeredExpKernel= function(x1,x2,sigmaF,l){
  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)
}

GPkernel = function(sigmaf,ell){
  squeredExpKernel= function(x,y){
    n1 = length(x)
    n2 = length(y)
    k = matrix(NA,n1,n2)
    for (i in 1:n2){
      k[,i] = sigmaf^2*exp(-0.5*( (x-y[i])/ell)^2 )
    }
    return(k)
  }
  class(squeredExpKernel) <- "kernel"
  return(squeredExpKernel)
}

plotGP = function(mean,variance,grid,x,y){
  plot(grid,mean,ylim = c(min(mean-1.96*sqrt(variance))
                          ,max(mean+1.96*sqrt(variance))),
       type = "l")
  lines(grid,
        mean+1.96*sqrt(variance),
        col = rgb(0, 0, 0, 0.3))
  lines(grid,
        mean-1.96*sqrt(variance),
        col = rgb(0, 0, 0, 0.3))
  points(x,y)
```

```
}

GP.periodic.kernel = function(sigmaf, l1,l2,d){
  periodicKernel = function(x1,x2){
    return(  (sigmaF^2)*exp(-(2*sin(pi*abs(x1-x2)/d)^2)/l1^2)*
               exp(-0.5*(abs(x1-x2)^2)/l2^2))
  }
  class(periodicKernel) <- "kernel"
  return(periodicKernel)
}




############################
###### Assignment 1 #######
############################
###### 1.2 ######
sigmaF = 1
l = 0.3
hyperParam = c(sigmaF,l)
sigmaNoise = 0.1
X = 0.4
y = 0.719

XStar = seq(-1,1,by=0.01)
GP = posteriorGP(X,y,XStar,hyperParam,sigmaNoise)
plotGP(GP$Mean,GP$Variance,XStar,X,y)

##### 1.3 ######
sigmaF = 1
l = 0.3
hyperParam = c(sigmaF,l)
sigmaNoise = 0.1
X = c(0.4,-0.6)
y = c(0.719,-0.044)

XStar = seq(-1,1,by=0.01)
GP = posteriorGP(X,y,XStar,hyperParam,sigmaNoise)
plotGP(GP$Mean,GP$Variance,XStar,X,y)

##### 1.4 ######
sigmaF = 1
l = 0.3
hyperParam = c(sigmaF,l)
sigmaNoise = 0.1
X = c(-1.0, -0.6, -0.2, 0.4, 0.8)
y = c(0.768, -0.044, -0.940, 0.719, -0.664)

XStar = seq(-1,1,by=0.01)
GP = posteriorGP(X,y,XStar,hyperParam,sigmaNoise)
plotGP(GP$Mean,GP$Variance,XStar,X,y)
```

```
##### 1.5 ######
sigmaF = 1
l = 1
hyperParam = c(sigmaF,l)
sigmaNoise = 0.1
X = c(-1.0, -0.6, -0.2, 0.4, 0.8)
y = c(0.768, -0.044, -0.940, 0.719, -0.664)

XStar = seq(-1,1,by=0.01)
GP = posteriorGP(X,y,XStar,hyperParam,sigmaNoise)
plotGP(GP$Mean,GP$Variance,XStar,X,y)

##########################
###### Assignment 2 ######
##########################

###### Innit ######
library(kernlab)
data = read.csv(
  "https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.csv",
  header=TRUE,
  sep=";")

time = seq(1,2190,by=5)
day = rep(seq(1,365,by=5),6)
###### 2.1 ######
X = c(1,3,4)
XStar = c(2,3,4)
x = 1
xprim = 2

gpK = GPkernel(sigmaf = 20,ell = 0.2)
gpK(x = 1, y = 2)
K = kernelMatrix(gpK,x=X,y=XStar)

###### 2.2 ######
sigmaF = 20
l = 0.2
temp = data[time,]$temp
lm.fit = lm(temp ~ time + I(time^2))
sigmaNoise = sd(lm.fit$residuals)

GPfit = gausspr(x = time,
                y = temp,
                kernel = GPkernel,
                kpar = list(sigmaf = sigmaF, ell = l),
                var = sigmaNoise^2)
meanPred = predict(GPfit,time) #Predict on train data
plot(time,temp)
lines(time,meanPred, col = "green")

###### 2.3 ######
var = posteriorGP(X = scale(time),
```

```
                  y = scale(temp),
                  XStar = scale(time),
                  hyperParam = c(sigmaF,l),
                  sigmaNoise = sigmaNoise)$Variance
lines(time,meanPred+1.96*sqrt(var), col = "black")
lines(time,meanPred-1.96*sqrt(var), col = "black")

###### 2.4 ######
sigmaF = 20
l = 0.2
temp = data[time,]$temp
lm.fit = lm(temp ~ day + I(day^2))
sigmaNoise = sd(lm.fit$residuals)

GPfit = gausspr(x = day,
                y = temp,
                kernel = GPkernel,
                kpar = list(sigmaf = sigmaF, ell = l),
                var = sigmaNoise^2)
meanPred = predict(GPfit,day) #Predict on train data
lines(time,meanPred, col = "red")

###### 2.5 ######
sigmaf = 10
l1 = 1
l2 = 10
d = 365/sd(time)
temp = data[time,]$temp
lm.fit = lm(temp ~ time + I(time^2))
sigmaNoise = sd(lm.fit$residuals)

GPfit = gausspr(x = time,
                y = temp,
                kernel = GP.periodic.kernel,
                kpar = list(sigmaf = sigmaf, l1 = l1, l2 = l2, d=d),
                var = sigmaNoise^2)
meanPred = predict(GPfit,time) #Predict on train data
#plot(time,temp)
lines(time,meanPred, col = "purple")


##########################
###### Assignment 3 ######
##########################
###### Innit ######
library(AtmRay)
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud
                 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 ######
GP.fit = gausspr(fraud ~ varWave+skewWave, data = train)

x1 <- seq(min(train$varWave),max(train$varWave),length=100)
x2 <- seq(min(train$skewWave),max(train$skewWave),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))
gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(data)[1:2]
probPreds <- predict(GP.fit, gridPoints, type="probabilities")

contour(x1,x2,matrix(probPreds[,2],100,byrow = TRUE), 20)

points(train[train$fraud == 1,"varWave"],
       train[train$fraud == 1,"skewWave"],
       col = "blue")

points(train[train$fraud == 0,"varWave"],
       train[train$fraud == 0,"skewWave"],
       col = "red")

con.mat.train = table(predict(GP.fit,train),train$fraud)
acc.train = sum(diag(con.mat.train)/sum(con.mat.train))

###### 3.2 ######
con.mat.test = table(predict(GP.fit,test),test$fraud)
acc.test = sum(diag(con.mat.test)/sum(con.mat.test))

###### 3.3 ######
GP.fit.all = gausspr(fraud ~ ., data = train)
con.mat.test.all = table(predict(GP.fit.all,test),test$fraud)
acc.test.all = sum(diag(con.mat.test.all)/sum(con.mat.test.all))
```