# 732A96: Advance Machine Learning

LAB 1: Graphical Models

*Arian Barakat/ariba405*

## Background

The purpose of the lab is to put in practice some of the concepts covered in the lectures. You can use any data set you like, e.g. you own data, data from public repositories, or data included in the R packages bnlearn and gRain. Check for instance https://www.bnlearn.com/documentation. The learning.test, asia or alarm data sets should suffice. Some questions may be easier to solve with one data set than with the others and, thus, you may need to try with different data sets.

### Question 1

**Question:**
Show that multiple runs of the hill-climbing algorithm can return non-equivalent DAGs. Explain why this happens. Hint: Check the function hc in the bnlearn package. Note that you can specify the initial structure, the number of random restarts, the score, and the equivalent sample size (a.k.a imaginary sample size) in the BDeu score. Use these options to answer the question. You may also want to use the functions plot, arcs, vstructs, cpdag and all.equal.

**Answer:**

For this question, we'll use the *alarm* dataset as it has the most number of nodes/variables among the three datasets, which increases the search space for the **hc()**-algorithm. The initial structure will be set to NULL (since we have no knowledge about the data as of this moment), the score criteria is set to the default (BIC) and the number of random restarts to 1.

From the result, we can observe that we obtain a non-equivalent graph already after 3 runs and that different arc(s) is: LVV->PCWP. According to the definition, two graphs are equivalent graphs if the represent the same dependencies (they have the same adjacencies and unshielded colliders), which isn't the case in the two constructed graphs as seen figure 1.

The reason why we obtain non-equivalent graphs is due to the fact the hill-climbing algorithm is a greedy algorithm. The algorithm adds, removes and reverses edges in order to maximize the score criteria. Due to the greedy property, the possibility of getting stuck in a local minimum is present. In our case, the large search space makes the algorithm more prone to get caught in local minima as the different combinations of nodes introduce relatively more complex landscape.

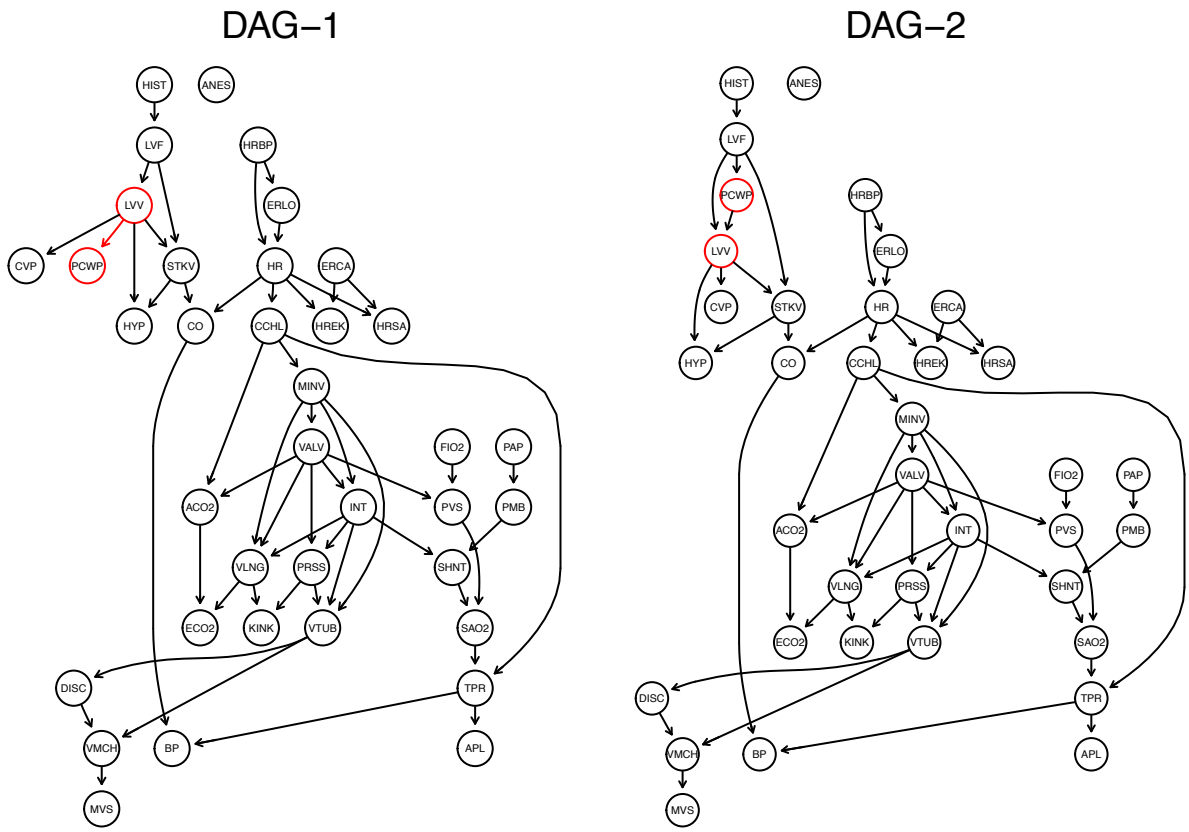



Figure 1: Bayesian Networks, non-equivalent graph DAG-1 and DAG-2

## Question 2

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

**Answer:**

From figure 2 we can see that as the user-defined imaginary sample size (*iss*) increases, the graph grows more densely but at the same time decreases in the score. The explanation for this behavior is that the BDeu score, which requires the prior parameter (equivalent/imaginary sample size), is sensitive to values of the parameter iss.

In a general setting the prior takes the form of a balanced/uniform Dirichlet distribution (for multinomial data), however in case of binary outcome, the prior can be collapsed to a beta distribution. Higher values of ISS implies that we are very certain about that the parameter distribution is centered around 0.5 (in the case of binary variable/nodes), which implies that the posterior distribution of the parameter will coincide with the prior as iss $\rightarrow \infty$. In other words, as iss increases:

$$p(A|B = 0) \sim 0.5$$
$$p(A|B = 1) \sim 0.5$$

Given this, we might argue that a model without an edge between nodes A and B is similar to a model with an edge between the same nodes. However, in such a case, Bayesian Networks has the tendency to favor the presence of an edge, which makes the graph grow more densely. As the graph grows more densely, the score will be penalized as a result of the larger dimensions.

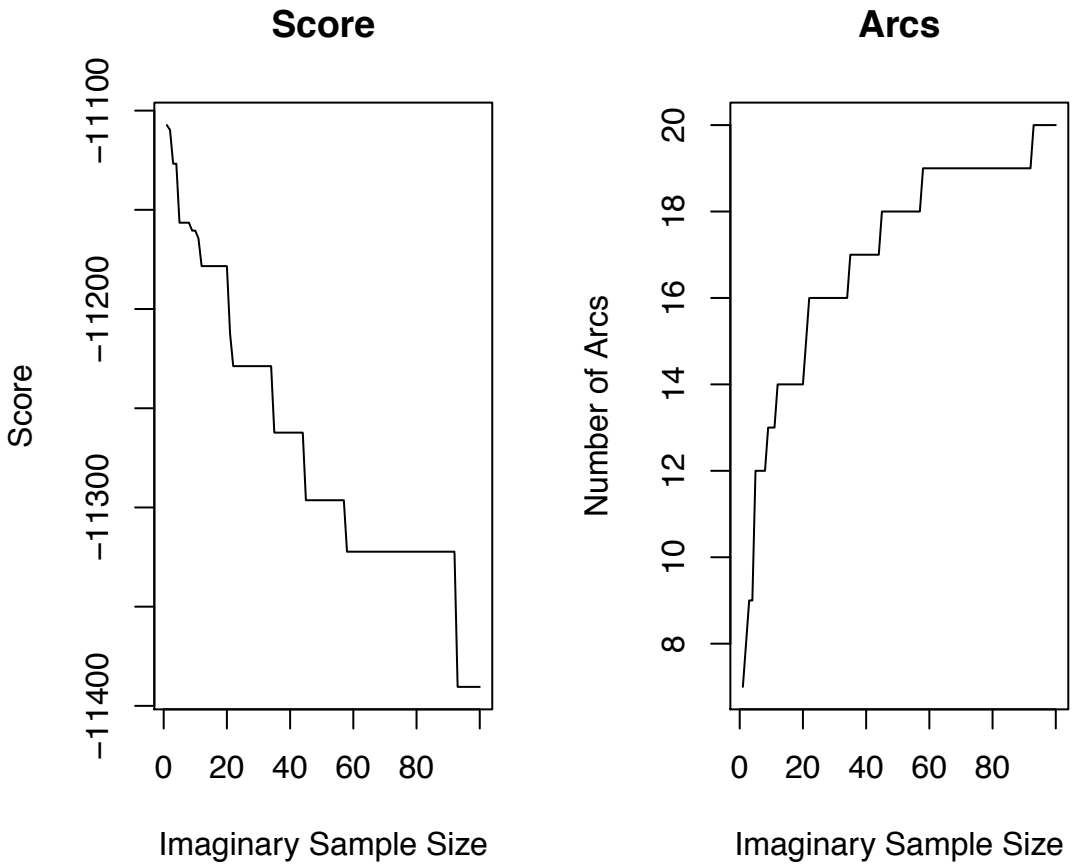



Figure 2: Graph score and arc density with respect to Imaginary Sample Size

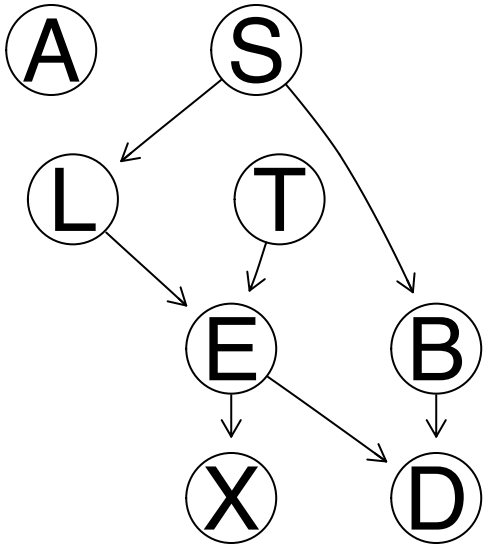



Figure 3: Bayesian Network using Hill-Climbing, Asia Data

## Question 3

**Question:**

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

**Answer:**

From the learned DAG in figure 3 we can construct some queries, which are shown in table 1.

Table 1: Queries

| Event | Evidence |
|---|---|
| (L == 'yes') | (S == 'yes') |
| (X == 'no') | (S == 'yes') & (T == 'yes') |
| (B == 'yes') | (NULL) |

Figure 4 present the result for the queries using the exact approach but also the approximate approach. For every query, the approximate approach has been run 1000 times. From the plot(s) we can see that the approximate algorithm gives us different values in every iteration. The key to understanding why the approximate approach gives us different answers is the fact the algorithm is using Monte-Carlo simulations to obtain the result of our query.

The plots also reveal that as we add more evidence (observed nodes), the approximate approach has the tendency to get less accurate, which especially can be seen in the second graph in figure 4. The reason behind this behavior is that the algorithm is based on observations that are compatible with the provided evidence.

# 732A96: Advance Machine Learning

LAB 2: HIDDEN MARKOV MODELS

*Arian Barakat/ariba405*

## Background

The purpose of the lab is to put in practice some of the concepts covered in the lectures. To do so, you are asked to model the behavior of a robot that walks around a ring. The ring is divided into 10 sectors. At any given time point, the robot is in one of the sectors and decides with equal probability to stay in that sector or move to the next sector. You do not have direct observation of the robot. However, the robot is equipped with a tracking device that you can access. The device is not very accurate though: If the robot is in the sector $i$, then the device will report that the robot is in the sectors $[i-2, i+2]$ with equal probability.

## Questions

### (1)

**Question:**
Build a HMM for the scenario described above

**Answer:**

See code and comments in appendix

**! Note**
In this particular HMM model, the robot is modeled to be able to move backward, forward to the next sector or stay in the same sector with equal probability.

## (2)

**Question:**
Simulate the HMM for 100 time steps.

**Answer:**

See code and comments in appendix

## (3)

**Question:**

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

**Answer:**

See code and comments in appendix

## (4)

**Question:**

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

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

**Answer:**

Table 1: Accuracy given Method

| | |
|---|---|
| Filtered | 0.55 |
| Smoothed | 0.58 |
| Viterbi | 0.36 |

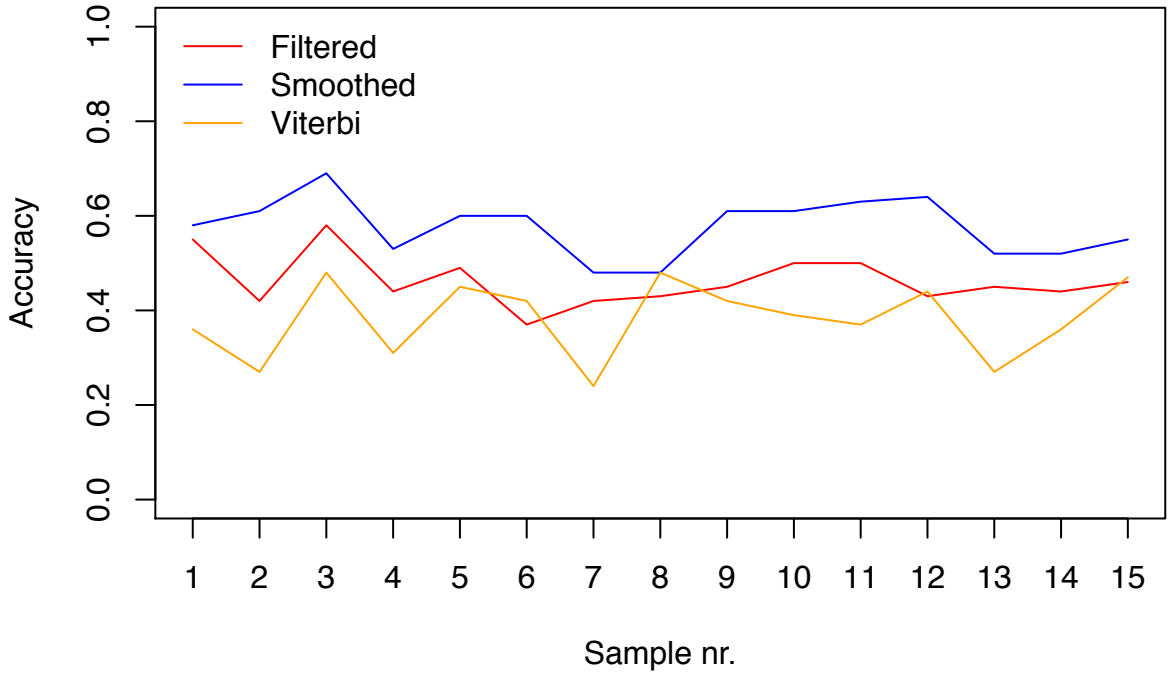



Figure 1: Accuracy with respect to Sample nr. and Method

## (5)

**Question:**

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

**Answer:**

Figure 1 displays the accuracy with respect to Sample nr. for the three different methods. As observed in the figure, the smoothed method seems to be generally more accurate than the other two methods. This observation can be linked to that the smoothed method uses all observed data when drawing inference about/predicting the hidden state (true location) of the robot while the other two use the observations sequentially. The former approach can be assumed to give better accuracy generally, however, it can be a non-pragmatic method in many cases.


Figure 2: Step plot of Entropy with respect to Observation number and Method

## (6)

**Question:**

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

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

**Answer:**

Figure 2 displays the step plot of the empirical entropies with respect to observation number and method. From the figure, we can observe that the step plots stay pretty much at the same level, which indicates that our knowledge/certainty about the location of the robot does not improve as the number of observations increase. The observed result can be derived to the fact that the transition matrix stays the same over time (steps) and the states $Z_{t+1}$ and $Z_{t-1}$ are conditionally independent given $Z_t$.

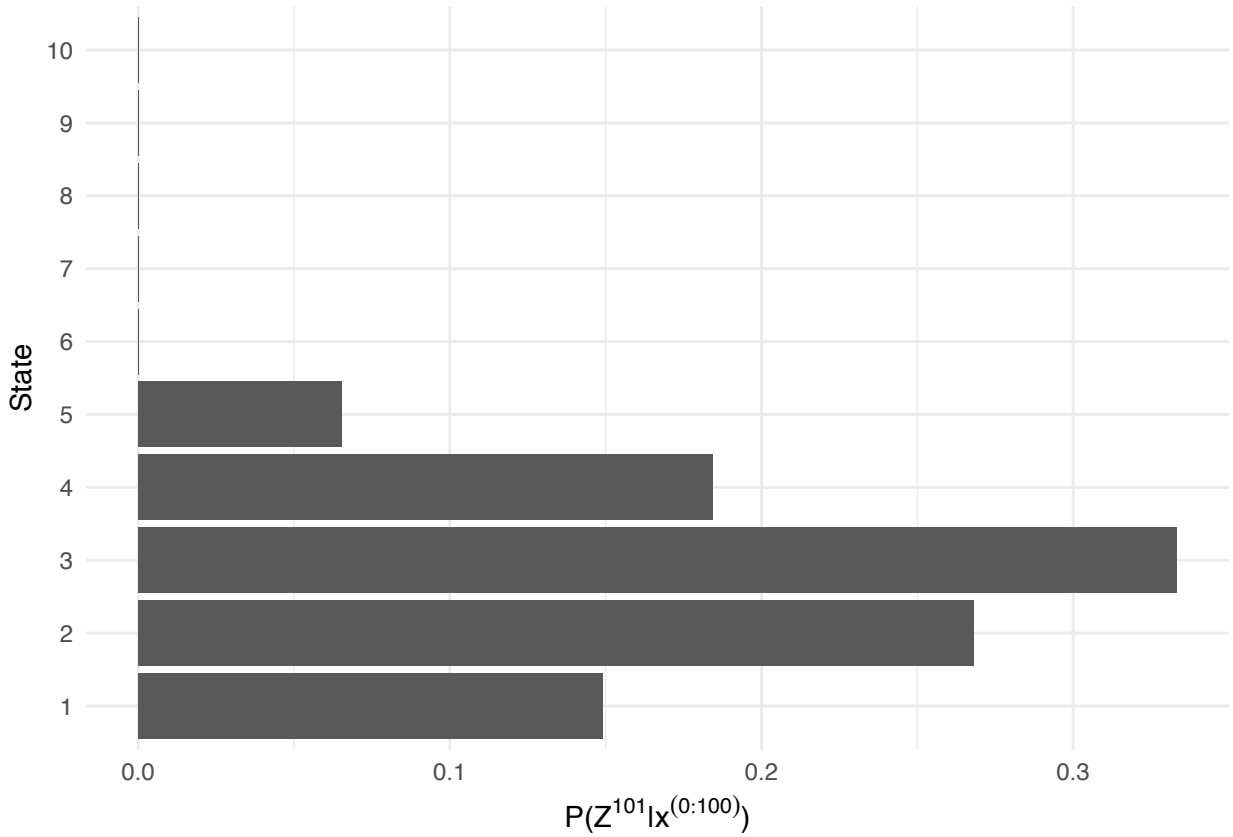



Figure 3: Probabilities of the hidden states for the time step 101

## (7)

**Question:**

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

**Answer:**

Table 2: Probabilities of the hidden states for the time step 101

| States | Pr |
|---|---|
| 1 | 0.1490070 |
| 2 | 0.2680546 |
| 3 | 0.3333333 |
| 4 | 0.1843263 |
| 5 | 0.0652787 |
| 6 | 0.0000000 |
| 7 | 0.0000000 |
| 8 | 0.0000000 |
| 9 | 0.0000000 |
| 10 | 0.0000000 |

# Appendix

```
knitr::opts_chunk$set(echo = FALSE,
                      warning = FALSE,
                      message = FALSE,
                      fig.pos = "HTBP")

library(HMM)
library(knitr)
library(entropy)
library(ggplot2)
library(gridExtra)


# Question 1

# The transition Matrix

nrSectors <- 10
intervalLength <- 5
prTransChange <- 1/3 # Uniform
prEmiss <- 1/5 # Uniform


# Creating Trans.Matrix and filling it w/ pr
# + Assuming that the robot can go backward as well
A <- matrix(nrow = nrSectors,
            ncol = nrSectors)

# Creating Emiss.Matrix and filling it w/ pr
E <- matrix(nrow = nrSectors,
            ncol = nrSectors)


for(rows in 1:nrSectors){
  for(cols in 1:nrSectors){
    A[rows, cols] <- ifelse(abs(rows - cols) <= 1 || abs(rows - cols) >= 9,
                            prTransChange,
                            0)
    E[rows, cols] <- ifelse(abs(rows - cols) <= 2 || abs(rows - cols) >= 8,
                            prEmiss,
                            0)
  }
}



HMM_model <- initHMM(States = as.character(1:nrSectors),
                     Symbols = as.character(1:nrSectors),
                     transProbs = A,
                     emissionProbs = E)
```

```r
set.seed(123456)

HMM_model_sim<- simHMM(HMM_model,
                       length = 100)



# Question 3
simulated_states <- HMM_model_sim$states
simulated_obs <- HMM_model_sim$observation


logFilteredPr <- forward(HMM_model,
                         observation =  simulated_obs)

SmoothPr <- posterior(HMM_model,
                      observation = simulated_obs)


mostProbPath <- viterbi(HMM_model,
                        observation = simulated_obs)

methodPr <- list("Filtered" = logFilteredPr,
                 "Smoothed" = SmoothPr,
                 "Viterbi" = mostProbPath)




# Question 4

predictState <- function(methodList){

  predicted <- list()

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

    if(names(methodList)[i] == "Filtered"){
      predicted[[i]] <- as.vector(apply(exp(methodList[[i]]), MARGIN = 2, FUN = function(x){
        which.max(prop.table(x))
      }))
    }

    if(names(methodList)[i] == "Smoothed"){
      predicted[[i]] <- as.vector(apply(methodList[[i]], 2, which.max))
    }

    if(names(methodList)[i] == "Viterbi"){
      predicted[[i]] <- as.numeric(methodList[[i]])
    }
```

```
  }

  predicted <- lapply(predicted, as.character)
  names(predicted) <- c("Filtered",
                        "Smoothed",
                        "Viterbi")

  return(predicted)

}

probableState <- predictState(methodPr)

calcAccur <- function(predictedState, trueStates){
  accurMethod <- lapply(predictedState, FUN = function(x){
    sum(x == trueStates)/length(x)
  })

  return(unlist(accurMethod))
}


estimatedAccur <- calcAccur(probableState, simulated_states)


kable(as.matrix(estimatedAccur), caption = "Accuracy given Method")




#itSamples <- seq(100, 1500, 100)
itSamples <- rep(100, 15)
accurMatrix <- matrix(nrow = length(itSamples),
                      ncol = 3)
set.seed(123456)
for(iter in 1:length(itSamples)){

  HMM_model_sim_temp <- simHMM(HMM_model,
                               length = itSamples[iter])

  simulated_states_temp <- HMM_model_sim_temp$states
  simulated_obs_temp <- HMM_model_sim_temp$observation


  logFilteredPr_temp <- forward(HMM_model,
                        observation =  simulated_obs_temp)

  SmoothPr_temp <- posterior(HMM_model,
                      observation = simulated_obs_temp)


  mostProbPath_temp <- viterbi(HMM_model,
                        observation = simulated_obs_temp)
```

```
  methodPr_temp <- list("Filtered" = logFilteredPr_temp,
                        "Smoothed" = SmoothPr_temp,
                        "Viterbi" = mostProbPath_temp)

  probableState_temp <- predictState(methodPr_temp)

  accurMatrix[iter,] <- calcAccur(probableState_temp, simulated_states_temp)
}

xGrid <- 1:length(itSamples)
plot(x = xGrid,
     y = accurMatrix[,1],
     ylab = "Accuracy",
     xlab = "Sample nr.",
     col = "red",
     type = "l",
     ylim = c(0,1),
     xaxt = "n")
axis(1, at = xGrid)
lines(x = xGrid,
     y = accurMatrix[,2],
     col = "blue",
     type = "l")
lines(x = xGrid,
     y = accurMatrix[,3],
     col = "orange",
     type = "l")
legend("topleft",
       legend = names(methodPr),
       col = c("red", "blue", "orange"),
       lty = rep(1,3),
       bty = "n")


# Question 6

calcEntropy <- function(methodList){

  entrop <- list()

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

    if(names(methodList)[i] == "Filtered"){
      entrop[[i]] <- as.vector(apply(exp(methodList[[i]]), MARGIN = 2, FUN = function(x){
        entropy.empirical(prop.table(x))
      }))
    }

    if(names(methodList)[i] == "Smoothed"){
      entrop[[i]] <- as.vector(apply(methodList[[i]], 2, entropy.empirical))
    }
```

```
  }

  names(entrop) <- c("Filtered","Smoothed")

  return(entrop)

}


itSamples <- 100
entropies <- list()

set.seed(123456)
for(iter in 1:length(itSamples)){

  HMM_model_sim_temp <- simHMM(HMM_model,
                               length = itSamples[iter])

  simulated_states_temp <- HMM_model_sim_temp$states
  simulated_obs_temp <- HMM_model_sim_temp$observation


  logFilteredPr_temp <- forward(HMM_model,
                                observation =  simulated_obs_temp)

  SmoothPr_temp <- posterior(HMM_model,
                             observation = simulated_obs_temp)


  methodPr_temp <- list("Filtered" = logFilteredPr_temp,
                        "Smoothed" = SmoothPr_temp)

  entropies[[iter]] <- calcEntropy(methodPr_temp)

}


dfList <- list()


for(iter in 1:length(itSamples)){

  dfList[[iter]] <- data.frame(Entropy = c(entropies[[iter]][[1]],entropies[[iter]][[2]]),
                               Method =  as.factor(c(rep("Filtered", itSamples[iter]),
                                                     rep("Smoothed", itSamples[iter]))),
                               Iteration = as.character(itSamples[iter]),
                               Observation = 1:itSamples[iter])

}


mergedDF <- Reduce(rbind, dfList)
```

```r
ggplot(data = mergedDF) +
  geom_step(aes(y = Entropy, col = Method, x = Observation)) +
  xlab("Observation #") +
  scale_fill_manual(values = c("red", "blue")) +
  theme_minimal() +
  facet_grid(~Method)



# Question 7


# Will be using the table from the posterior function at step 100

hiddenStateStep101 <- SmoothPr[,100] %*% A


ggplot(data = data.frame(pr = as.vector(hiddenStateStep101),
                         states = as.factor(1:length(hiddenStateStep101)))) +
  geom_bar(aes(y = pr, x = states), stat = "identity") +
  xlab("State") + ylab(expression(paste("P(",Z^101,"|",x^(0:100),")"))) +
  coord_flip() +
  theme_minimal()

kable((data.frame(States = as.factor(1:length(hiddenStateStep101)),
                  Pr = as.vector(hiddenStateStep101))),
      caption = "Probabilities of the hidden states for the time step 101")
```

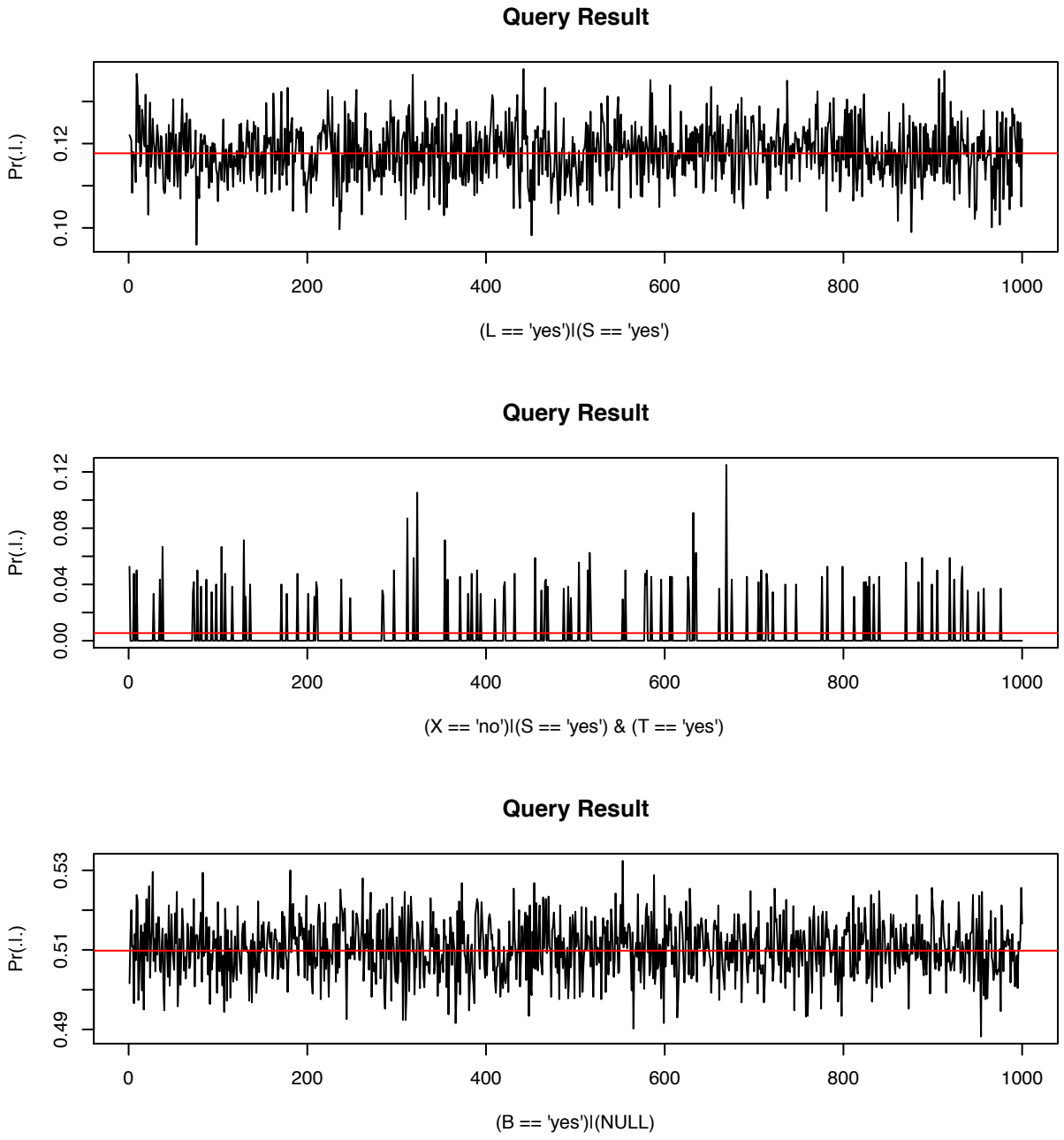



Figure 4: Approximate and Exant Answer with respect to iterations for different Queries, n = 1000

## Question 4

**Question:**

There are 29281 DAGs with five nodes. Compute approximately the fraction of the 29281 DAGs that represent different independence models. In the light of the result obtained, would you say that it is preferable to perform structure learning in the space of DAGs or in the space of essential graphs? Hint: You do not need to produce the 29281 DAGs. Instead you can sample DAGs uniformly, convert each DAG in the sample to its essential graph (a.k.a completed partial DAG or CPDAG), and then count how many different essential graphs you have left. You can do all this with the functions random.graph, cpdag, lapply, unique and length in the bnlearn and core packages. Try different values for the parameters every and burn.in in the function random.graph.

**Answer:**

See code in appendix.

By running the code we obtain a number of unique essential graphs equal to 51.997 percent of all the created graphs with 5 nodes. Given this, one could argue that it's preferred to perform structure learning in the space of essential graphs instead of DAGs, which would in a sense make score based algorithms less prone to get caught in local minima.

# Appendix

```
# Setup

library(bnlearn)
source("..//Source/biocLite.R")
#source("http://bioconductor.org/biocLite.R")
#biocLite(c("graph", "Rgraphviz", "RBGL"))
library(gRain)
library(stringr)


# Question 1

data("learning.test")
data("asia")
data("alarm")



count <- 0
is_equivalent <- TRUE
DAG <- list()
set.seed(12345)

while(is_equivalent){
  count <- count + 1
  DAG[[1]] <- hc(alarm, start = NULL, restart = 1)
  DAG[[2]] <- hc(alarm, start = NULL, restart = 1)
  test <- all.equal(cpdag(DAG[[1]]),cpdag(DAG[[2]]))
  is_equivalent <- ifelse(any(test == "TRUE"), TRUE, FALSE)
}

DAG_arcs <- lapply(DAG, function(x){
  x_arcs <- arcs(x)
  out <- apply(x_arcs, MARGIN = 1, FUN = function(x){paste(x, collapse = "->")})
  return(out)
})

differentArcs <- setdiff(DAG_arcs[[1]], DAG_arcs[[2]])


par(mfrow = c(1,2))
graphviz.plot(DAG[[1]],
              main = "DAG-1",
              highlight = list(arcs =
                               matrix(c("LVV", "PCWP"),
                                      ncol = 2,
                                      dimnames =
                                        list(NULL,
                                             c("from","to"))),
                               nodes = c("LVV", "PCWP")))
graphviz.plot(DAG[[2]],
```

```
             main = "DAG-2",
             highlight = list(nodes = c("LVV", "PCWP")))
par(mfrow = c(1,1))

# Question 2

totSampleSize <- 100
scores <- vector("numeric", length = totSampleSize)
nrArcs <- vector("numeric", length = totSampleSize)

for(i in 1:totSampleSize){
  learned_DAG <- hc(asia, score = "bde", iss = i)
  scores[i] <- score(learned_DAG, asia)
  nrArcs[i] <- nrow(arcs(learned_DAG))
}

par(mfrow = c(1,2))
plot(y = scores, x = 1:totSampleSize,
     type = "l", main = "Score",
     xlab = "Imaginary Sample Size",
     ylab = "Score")
plot(y = nrArcs, x = 1:totSampleSize,
     type = "l", main = "Arcs",
     xlab = "Imaginary Sample Size",
     ylab = "Number of Arcs")
par(mfrow = c(1,1))


#Question 3


DAG_asia <- hc(asia)
DAG_asia_fit <- bn.fit(DAG_asia, data = asia, method = "mle")

graphviz.plot(DAG_asia_fit)

# These queries have been decided from the learned structure
queries <- matrix(data = c("(L == 'yes')", "(S == 'yes')",
                           "(X == 'no')", "(S == 'yes') & (T == 'yes')",
                           "(B == 'yes')", "(NULL)"),
                  ncol = 2,
                  byrow = TRUE,
                  dimnames = list(NULL, c("Event", "Evidence")))

knitr::kable(queries, caption = "Queries")



# Exact
junctionTree_asia <- compile(as.grain(DAG_asia_fit))

queryExact <- vector("numeric", length = nrow(queries))
for(i in 1:nrow(queries)){
```

```
  if(str_detect(queries[i,2], "&")){
    evid <- str_split(queries[i,2], pattern = "&", simplify = TRUE)
    evid <- str_extract_all(evid, pattern = "[A-z+]", simplify = TRUE)
  } else{
    evid <- str_extract_all(queries[i,2], pattern = "[A-z+]", simplify = TRUE)
  }


  if(str_c(evid, collapse = "") == "NULL"){
    # No evidence to set

    event <- str_extract_all(queries[i,1], pattern = "[A-z+]", simplify = TRUE)
    queryRes <- querygrain(junctionTree_asia, nodes = event[1])
    queryExact[i] <- queryRes[[1]][str_c(event[-1], collapse = "")]

  } else{
    currentEvidence <- setEvidence(junctionTree_asia,
                                   nodes = evid[,1],
                                   states = apply(evid[,-1, drop = FALSE],
                                                  1,
                                                  function(x){str_c(x, collapse = "")}))

      event <- str_extract_all(queries[i,1], pattern = "[A-z+]", simplify = TRUE)
      queryRes <- querygrain(currentEvidence, nodes = event[1])
      queryExact[i] <- queryRes[[1]][str_c(event[-1], collapse = "")]
  }
}

# Approximate

queryApprox <- list()
for(i in 1:nrow(queries)){

  evid <- str_extract_all(queries[i,2], pattern = "[A-z+]", simplify = TRUE)
  if(str_c(evid, collapse = "") == "NULL"){
    queryApprox[[i]] <- sapply(1:1000, function(i){cpquery(DAG_asia_fit,
                            event = eval(parse(text = queries[i, 1])),
                            evidence = TRUE)})

  } else {
      queryApprox[[i]] <- sapply(1:1000,FUN = function(i){cpquery(DAG_asia_fit,
                            event = eval(parse(text = queries[i, 1])),
                            evidence = eval(parse(text = queries[i, 2])))})

  }

}
```

```r
queries_name <-  apply(queries,MARGIN = 1,
                       FUN = function(x){str_c(x, collapse = "|")})

par(mfrow = c(3,1))
for(i in 1:nrow(queries)){
  plot(y = queryApprox[[i]],
       x = 1:1000,
       type = "l",
       main = "Query Result",
       xlab = queries_name[i],
       ylab = "Pr(.|.)")
  abline(h = queryExact[i], col = "red")
}
par(mfrow = c(1,1))

# Question 4

numberOfGraphs <- 25000
sampledDAGs <- random.graph(nodes = LETTERS[1:5],
                            num = numberOfGraphs,
                            method = "ic-dag",
                            burn.in = 12 * length(LETTERS[1:5])^2,
                            every = 1)
uniqueDAGs <- unique(sampledDAGs)
equivDAGs <- lapply(uniqueDAGs, cpdag)
percentUnique <- round((length(unique(equivDAGs))/length(uniqueDAGs))*100, digits = 3)
```

# 732A96: Advance Machine Learning

LAB 3: Gaussian Processes

*Arian Barakat/ariba405*

## Question 1: Implementing Gaussian process regression from scratch

This first exercise will have you writing your own code for the Gaussian process regression model:

$$y = f(x) + \epsilon, \quad \varepsilon \sim N(0, \sigma_n^2)$$

$$f \sim GP\left[0, k\left(x, x'\right)\right]$$

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

### (a)

Write your own code for simulating from the posterior distribution of $f(x)$ using the squared exponential kernel. The function (name it **posteriorGP**) should return vectors with the posterior mean and variance of $f$, both evaluated at a set of $x$-values ($x^\star$). You can assume that the prior mean of $f$ is zero for all $x$. The function should have the following inputs:

- $x$ (vector of training inputs)
- $y$ (vector of training targets/outputs)
- $x_*$ (vector of inputs where the posterior distribution is evaluated)
- hyperParam (vector with two elements $\sigma_f$ and $\ell$)
- sigmaNoise ($\sigma_n$)

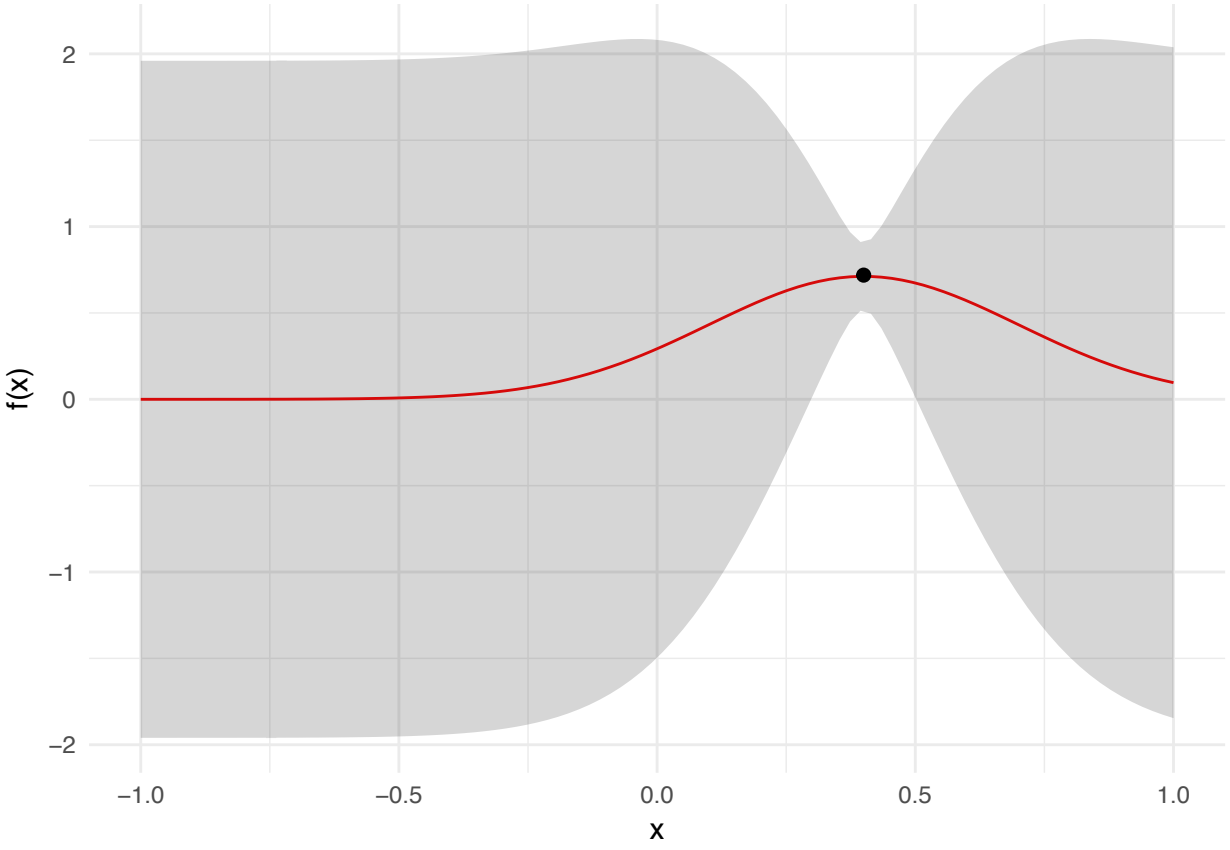
**Answer:**

See code in appendix

## (b)

Now let the prior hyperparameters be $\sigma_f = 1, \ell = 0.3$. Update this prior with a single observation: $(x, y) = (0.4, 0.719)$. Assume that the noise standard deviation is known to be $\sigma_n = 0.1$. Plot the posterior mean of $f$ over the interval $x \in [-1, 1]$. Plot also 95% probability (pointwise) bands for $f$.

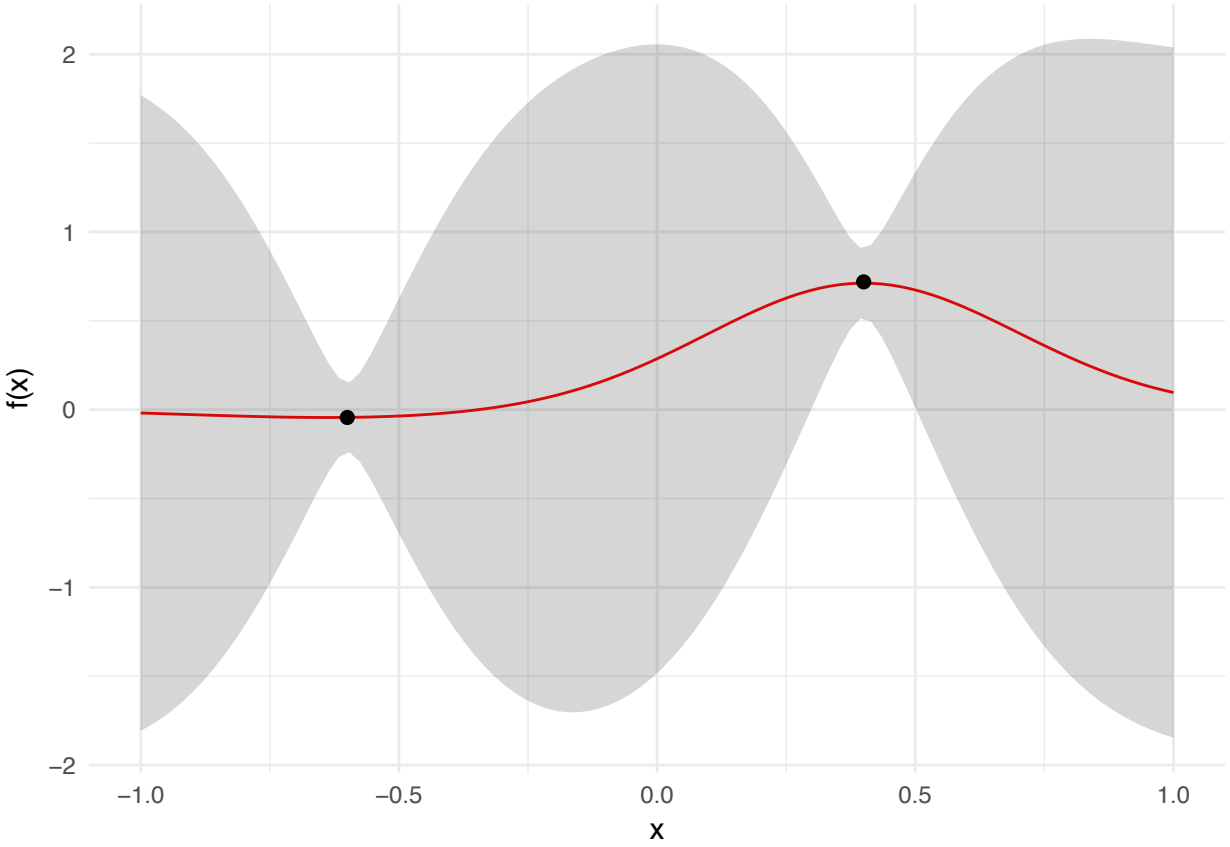
**Answer:**



## (c)

Update your posterior from 1b) with another observation: $(x, y) = (-0.6, -0.044)$. Plot the posterior mean of $f$ over the interval $x \in [-1, 1]$. Plot also 95% probability bands for $f$.[Hint: updating the posterior after one observation with a new observation gives the same result as updating the prior directly with the two observations. Bayes is beautiful!]

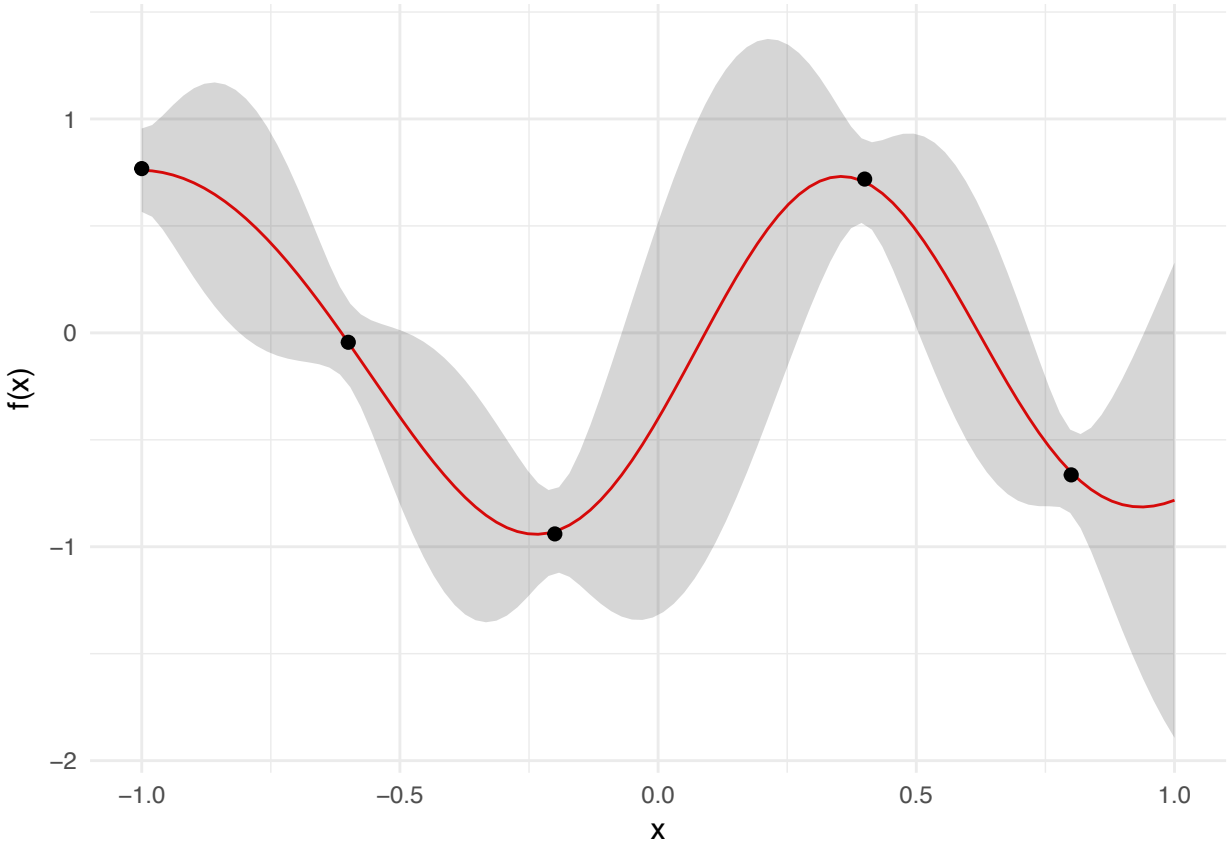
**Answer:**



## (d)

Compute the posterior distribution of $f$ using all 5 data points in Table 1 below (note that the two previous observations are included in the table).Plot the posterior mean of $f$ over the interval $x \in [-1, 1]$. Plot also 95% probability intervals for $f$.

**Answer:**

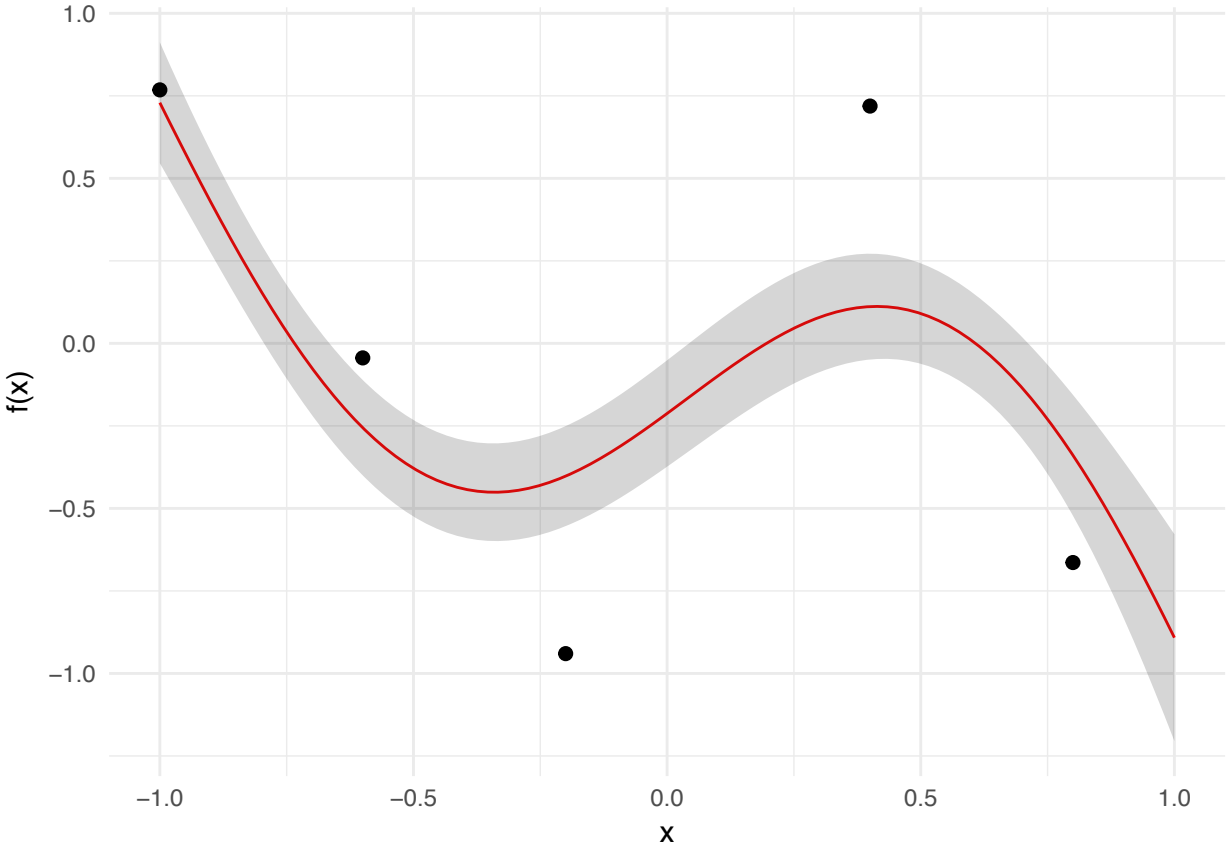


## (e)

Repeat 1d), this time with the hyperparameters $\sigma_f = 1, \ell = 1$. Compare the results.

**Answer:**

If we compare the figure for question (e) with the one provided in question (d) we can observe that the former one has a more smoothed or "generalized" posterior GP. A greater value of the $\ell$ hyperparameter implies that we place relatively more weight on distant training points compared to when we use a kernel with lower length scale. Another observation is that the variance in this model seems to have decreased, which intuitively can be linked to the fact we have relatively more information as we allow a greater influence from distant points.



# Question 2: Gaussian process regression on real data using the kernlab package

This exercise lets you explore the kernlab package on a data set of daily mean temperature in Stockholm (Tullinge) during the period January 1, 2010 - December 31, 2015. I have removed the leap year day February 29, 2012 to make your life simpler

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

## (a)

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

**Answer:**

```
##          [,1]
## [1,] 2.426123
```

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

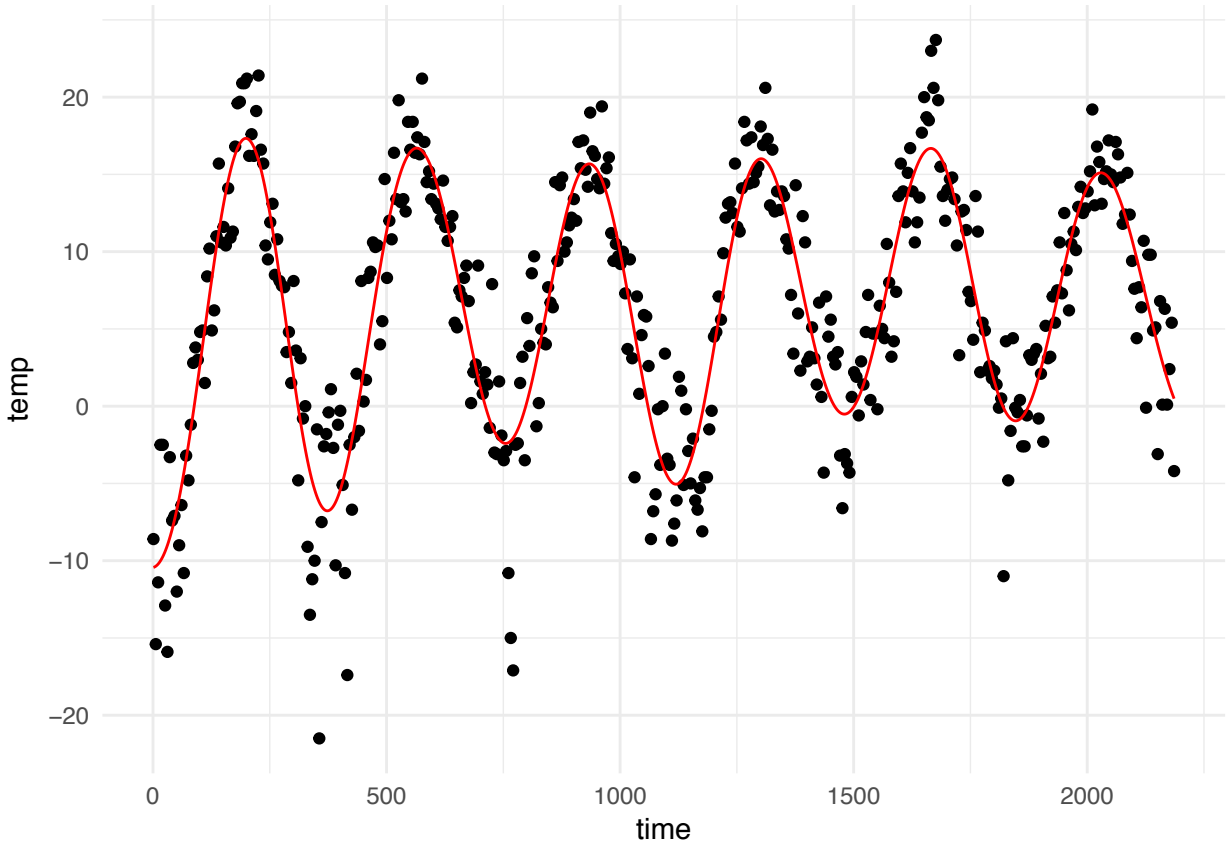
## (b)

Consider first the model:

$$temp = f(\text{time}) + \varepsilon \quad \varepsilon \sim N(0, \sigma_n^2)$$
$$f \sim GP\left[0, k(\text{time}, \text{time}')\right]$$

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

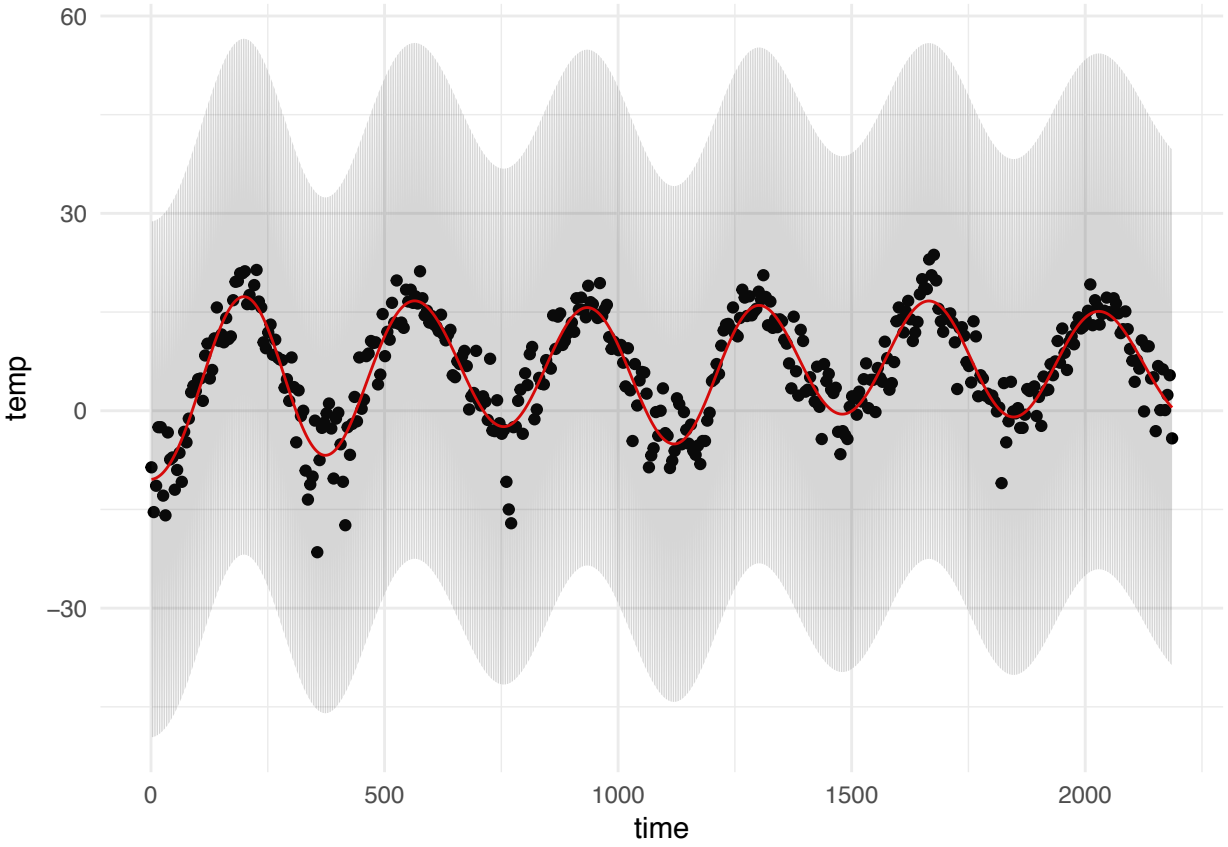
**Answer:**



## (c)

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

**Answer:**



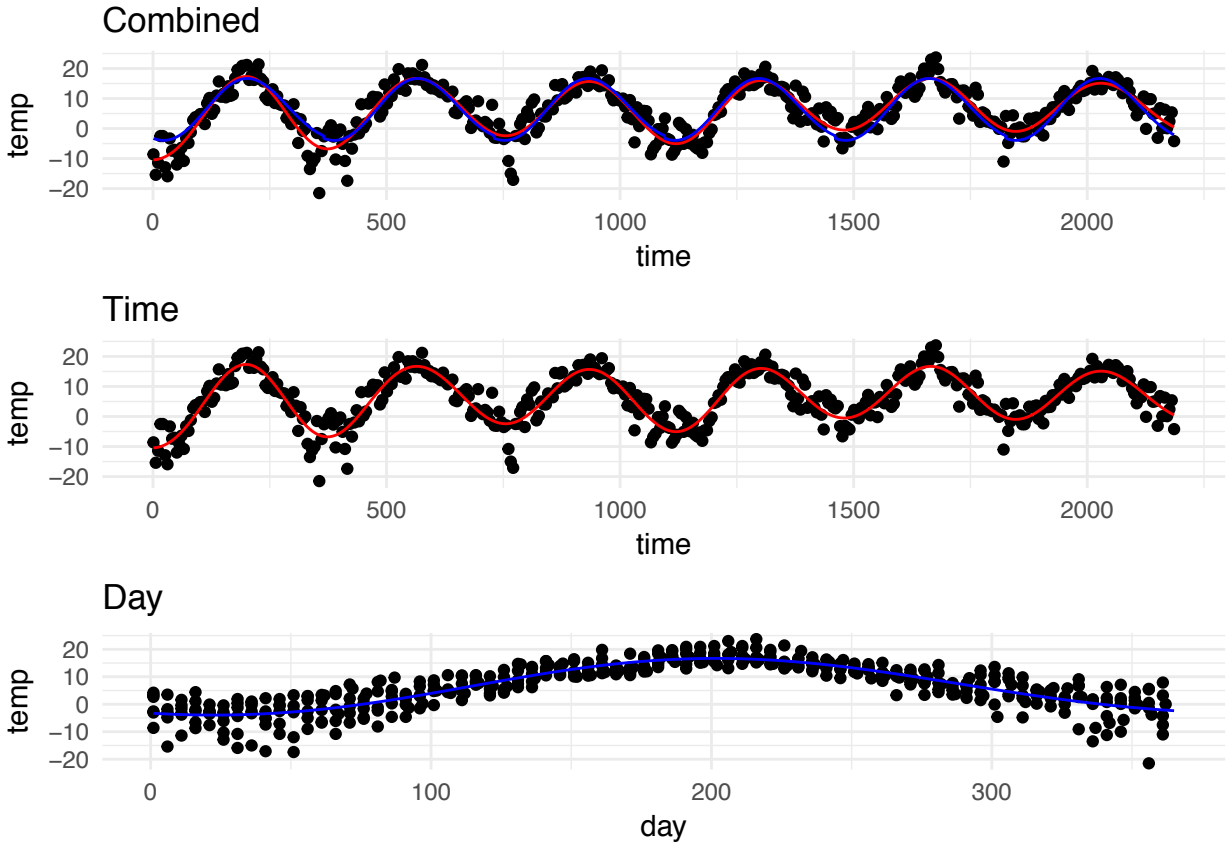
## (d)

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

**Answer:**

Before a conclusion is made it's worth mentioning that R recycles the prediction from the "Day" model when we impose the predictions over the time-axis. In other words, the predictions will have the same pattern from year to year.

From the combined figure, one could draw the conclusion that models are quite similar to each other with some minor differences at the yearly minimum and maximum temperatures. The "Day" model has the benefit of being able to capture the seasonal variation within the years, while the "Time" model is beneficial for capturing the longterm variation and trend. It can be assumed that none of these models will be able to capture the interacted variation between the variables independently, at least very poorly. One could theoretically be better off by constructing a kernel that is able to combine the effect of the two variables.



## (e)

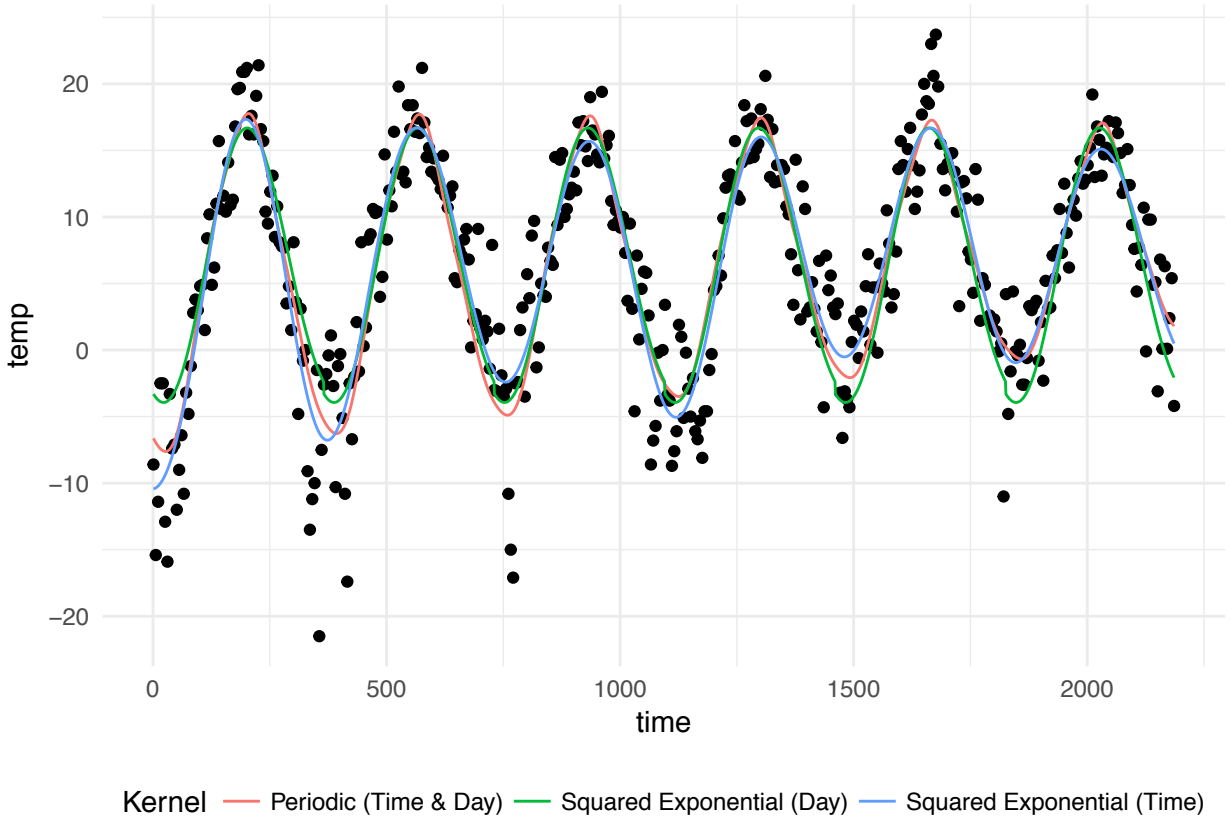
Now implement a generalization of the periodic kernel given in my slides from Lecture 2 of the GP topic (Slide 6)

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

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

**Answer:**

From the figure, we can observe that the periodic model seems relatively more smooth compared to the other models yet being able to capture more extreme observation (e.g. time $\approx 400$ and time $\approx 1500$). This kind of behavior indicates that the model is rather a good model for capturing the seasonal and longterm variation.



# Question 3: Gaussian process classification using the kernlab package

## (a)

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

**Answer:**

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

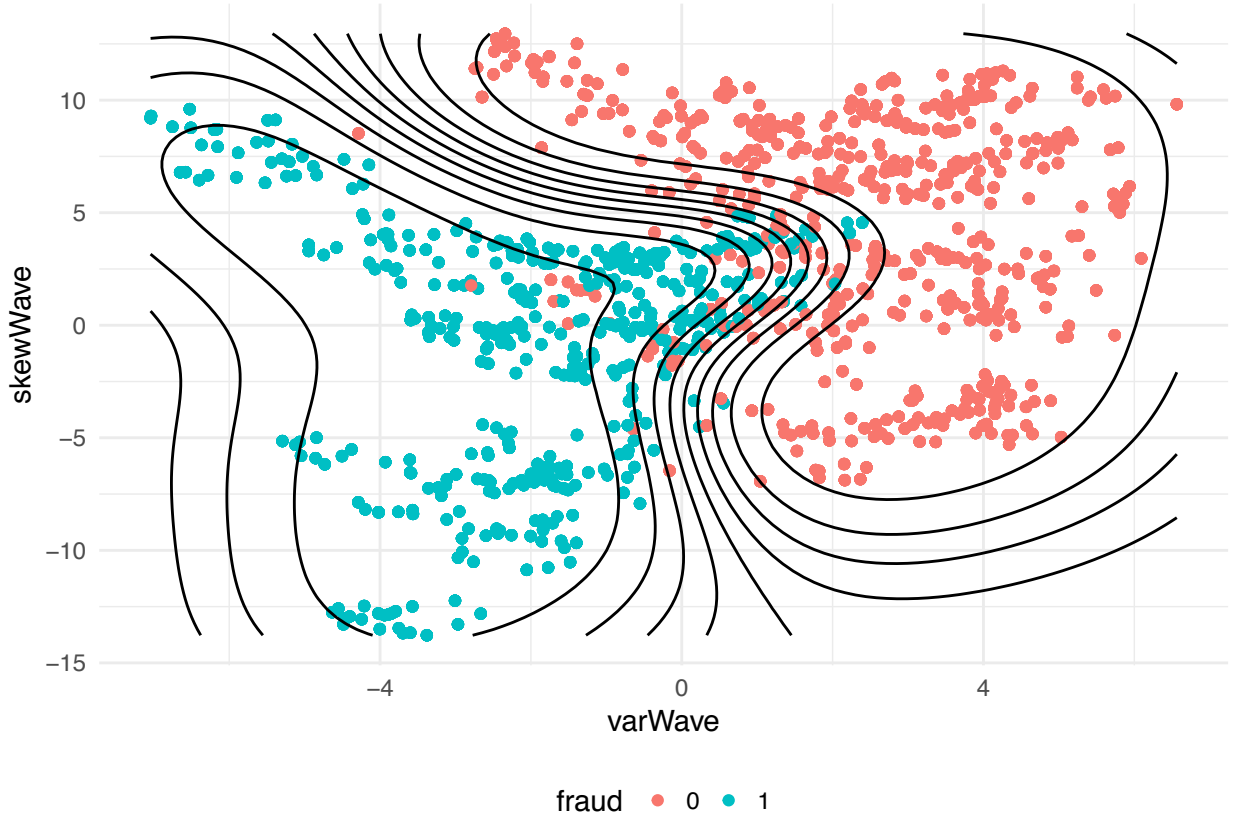



Table 1: Confusion Matrix, Training Data

| | 0 | 1 |
|---|---|---|
| 0 | 512 | 44 |
| 1 | 24 | 420 |

```
## Accuracy, Train data: 0.932000
```

## (b)

Using the estimated model from 3a), make predictions for the testset. Compute the accuracy.

**Answer:**

Table 2: Confusion Matrix, Test Data

| | 0 | 1 |
|---|---|---|
| 0 | 191 | 15 |
| 1 | 9 | 157 |

```
## Accuracy, Test data: 0.935484
```

## (c)

Train a model using all four covariates. Make predictions on the test and compare the accuracy to the model with only two covariates.

**Answer:**

From the result below we can observe that the prediction accuracy improves as we increase the number of features in the model.

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

Table 3: Confusion Matrix, Test Data (All four Covariates)

| | 0 | 1 |
|---|---|---|
| 0 | 205 | 1 |
| 1 | 0 | 166 |

```
## Accuracy, Test (All four Covariates) data: 0.997312
```

# Appendix

```
knitr::opts_chunk$set(echo = FALSE,
                      warning = FALSE,
                      message = FALSE,
                      error = FALSE)

library(ggplot2)
library(kernlab)
library(timeDate)
library(dplyr)
library(grid)
library(gridExtra)
library(knitr)
pathData <- "../Data"


# Question 1 (a)

# Kernal - Squared Exponential

# Setting up the kernel
sqExpKernel <- 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)
}


posteriorGP <- function(x,y,xStar,K,hyperParam,sigmaNoise){

  if(is.data.frame(x) || is.matrix(x)){
    nParam <- nrow(x)
    nDim <- ncol(x)
  } else {
    nParam <- length(x)
    nDim <- 1
  }

  if(!is.null(hyperParam)){
      Ktrain <- do.call(K, args = list(x, x,hyperParam[1], hyperParam[2]))
      Kstar <- do.call(K, args = list(x, xStar,hyperParam[1], hyperParam[2]))
      Ktest <- do.call(K, args = list(xStar, xStar,hyperParam[1], hyperParam[2]))
  } else {
    # If a kernel with pre-specified hyperparameters is provided
      Ktrain <- do.call(K, args = list(x, x))
      Kstar <- do.call(K, args = list(x, xStar))
      Ktest <- do.call(K, args = list(xStar, xStar))
    }
```

```r
  # Based on Algorithm 2.1 on page 19 of Rasmussen and Willams (RW)
  L <- chol(Ktrain + (sigmaNoise^2)*diag(nParam))
  alpha <- solve(L, solve(t(L), y))
  fBarStar <- t(Kstar)%*%alpha
  v <- solve(t(L), Kstar)
  vVar <- Ktest - t(v)%*%v


  return(list("Fstar" = as.vector(fBarStar),
              "Fcovar" = vVar))
}




# Question 1 (b)

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

xGrid <- seq(-1,1, length.out = 100)
hypPar <- c(1,0.3)

posteriorGPQ1b <- posteriorGP(x = x, y = y,
                              xStar = xGrid,
                              K = sqExpKernel,
                              hyperParam = hypPar,
                              sigmaNoise = 0.1)

ggplot(data.frame(f = posteriorGPQ1b$Fstar,
                  x = xGrid,
                  var = diag(posteriorGPQ1b$Fcovar))) +
  geom_line(aes(y = f, x = x), col = "red") +
  geom_ribbon(aes(x = x,
                  ymin = (f - 1.96*sqrt(var)),
                  ymax = (f + 1.96*sqrt(var))),
              alpha = 0.2) +
  ylab("f(x)") +
  theme_minimal() +
  geom_point(data = data.frame(xTrain = x,
                               yTrain = y),
             aes(x = xTrain, y = yTrain), col = "black", size = 2)


# Removing since it's not going
# to be used anymore (consumes memory)
rm(posteriorGPQ1b)


# Question 1 (c)
```

```r
x <- c(x, -0.6)
y <- c(y, -0.044)

posteriorGPQ1c <- posteriorGP(x = x, y = y,
                              xStar = xGrid,
                              K = sqExpKernel,
                              hyperParam = hypPar,
                              sigmaNoise = 0.1)

ggplot(data.frame(f = posteriorGPQ1c$Fstar,
                  x = xGrid,
                  var = diag(posteriorGPQ1c$Fcovar))) +
  geom_line(aes(y = f, x = x), col = "red") +
  geom_ribbon(aes(x = x,
                  ymin = (f - 1.96*sqrt(var)),
                  ymax = (f + 1.96*sqrt(var))),
              alpha = 0.2) +
  ylab("f(x)") +
  theme_minimal() +
  geom_point(data = data.frame(xTrain = x,
                               yTrain = y),
             aes(x = xTrain, y = yTrain), col = "black", size = 2)

# Removing since it's not going
# to be used anymore (consumes memory)
rm(posteriorGPQ1c)



# Question 1 (d)

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


posteriorGPQ1d <- posteriorGP(x = x, y = y,
                              xStar = xGrid,
                              K = sqExpKernel,
                              hyperParam = hypPar,
                              sigmaNoise = 0.1)

ggplot(data.frame(f = posteriorGPQ1d$Fstar,
                  x = xGrid,
                  var = diag(posteriorGPQ1d$Fcovar))) +
  geom_line(aes(y = f, x = x), col = "red") +
  geom_ribbon(aes(x = x,
                  ymin = (f - 1.96*sqrt(var)),
                  ymax = (f + 1.96*sqrt(var))),
              alpha = 0.2) +
  ylab("f(x)") +
  theme_minimal() +
  geom_point(data = data.frame(xTrain = x,
```

```
                                yTrain = y),
             aes(x = xTrain, y = yTrain), col = "black", size = 2)


# Removing since it's not going
# to be used anymore (consumes memory)
rm(posteriorGPQ1d)




# Question 1 (e)

hypPar <- c(1,1)

posteriorGPQ1e <- posteriorGP(x = x, y = y,
                              xStar = xGrid,
                              K = sqExpKernel,
                              hyperParam = hypPar,
                              sigmaNoise = 0.1)

ggplot(data.frame(f = posteriorGPQ1e$Fstar,
                  x = xGrid,
                  var = diag(posteriorGPQ1e$Fcovar))) +
  geom_line(aes(y = f, x = x), col = "red") +
  geom_ribbon(aes(x = x,
                  ymin = (f - 1.96*sqrt(var)),
                  ymax = (f + 1.96*sqrt(var))),
              alpha = 0.2) +
  ylab("f(x)") +
  theme_minimal() +
  geom_point(data = data.frame(xTrain = x,
                               yTrain = y),
             aes(x = xTrain, y = yTrain), col = "black", size = 2)


# Removing since it's not going
# to be used anymore (consumes memory)
rm(posteriorGPQ1e)


# Question 2

# Reading the Data
dataTullinge <- read.csv(paste(pathData, "TempTullinge.csv", sep = "/"),
                         header = TRUE, sep = ";")

dataTullinge$date <- strptime(dataTullinge$date, format = "%d/%m/%y")
dataTullinge$time <- 1:nrow(dataTullinge)
dataTullinge$day <- dayOfYear(as.timeDate(dataTullinge$date))

# Thinning, keeping every 5th obs. starting with the first
toKeep <- ((1:nrow(dataTullinge)) %% 5) - 1) == 0
```

```r
dataTullinge <- dataTullinge[toKeep,]



# Question 2 (a)


# Kernel using closjure

sqExpKer <- function(sigma, ell){

  rval <- function(x,y, sigmaF = sigma, l = ell){
    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])/l)^2 )
    }
    return(K)
  }

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


sigmaF <- 2
ell <- 1
kernelSqExp <- sqExpKer(sigma = sigmaF, ell = ell)

x <- 1
xPrime <- 2
kernelSqExp(x, xPrime)

x <- c(1,3,4)
xStar <- c(2,3,4)


kernelMatrix(kernel = kernelSqExp, x = x, y = xStar)


# Question 2 (b)

sigmaF <- 20
ell <- 0.2


formula <- as.formula(temp ~ time + I(time^2))

sigmaSq_n <- dataTullinge %>%
              lm(formula = formula) %>%
              residuals() %>%
              var()
```

```r
GPfit <- gausspr(temp ~ time, data = dataTullinge,
                 kernel = sqExpKer, kpar = list(sigma = sigmaF, ell = ell),
                 var = sigmaSq_n)

timeRange <- range(dataTullinge$time)
xGrid <- seq(timeRange[1],timeRange[2], by = 1)
predictedQ2b <- predict(GPfit, data.frame(time = xGrid))

plotGPQ2b <- dataTullinge %>%
  ggplot() +
  geom_point(aes(y = temp, x = time)) +
  theme_minimal() +
  geom_line(data = data.frame(fbar = as.vector(predictedQ2b),
                              x = xGrid),
            aes(x = x, y = fbar), col = "red")


print(plotGPQ2b)



# Question 2 (c)

posteriorGPQ2c <- posteriorGP(x = dataTullinge$time,
                              y = dataTullinge$temp,
                              xStar = xGrid,
                              K = sqExpKernel,
                              hyperParam = c(sigmaF, ell),
                              sigmaNoise = sqrt(sigmaSq_n))


plotGPQ2c <- plotGPQ2b +
  geom_ribbon(data = data.frame(f = as.vector(predictedQ2b),
                                x = xGrid,
                                var = diag(posteriorGPQ2c$Fcovar)),
              aes(x = x,
                  ymin = (f - 1.96*sqrt(var)),
                  ymax = (f + 1.96*sqrt(var))),
              alpha = 0.2)



# Removing since it's not going
# to be used anymore (consumes memory)
rm(posteriorGPQ2c)
print(plotGPQ2c)


# Question 2 (d)

GPfitDay <- gausspr(temp ~ day, data = dataTullinge,
                    kernel = sqExpKer, kpar = list(sigma = sigmaF, ell = ell*6),
                    var = sigmaSq_n)
```

```r
xGridDay <- seq(1,365, 1)

predictedQ2Day <- predict(GPfitDay, data.frame(day = xGridDay))

plotGPQ2Day <- dataTullinge %>%
  ggplot() +
  geom_point(aes(y = temp, x = day)) +
  theme_minimal() +
  geom_line(data = data.frame(fbarDay = as.vector(predictedQ2Day),
                              x = xGridDay),
            aes(x = x, y = fbarDay), col = "blue")


predictedQ2DayTime <- predict(GPfitDay, data.frame(day = xGrid))
plotGPQ2Joined <- plotGPQ2b +
  geom_line(data = data.frame(fbarDay = rep(as.vector(predictedQ2Day), 6)[1:length(xGrid)],
                              x = xGrid),
            aes(x = x, y = fbarDay), col = "blue")

grid.arrange(grobs = list(plotGPQ2Joined + ggtitle("Combined"),
                          plotGPQ2b + ggtitle("Time"),
                          plotGPQ2Day + ggtitle("Day")))


# Question 2 (e)

periodicKernel <- function(ell, period, sigma){

  rval <- function(x, y, ell1 = ell[1] , ell2 = ell[2], d = period, sigmaF = sigma){
    n1 <- length(x)
    n2 <- length(y)
    K <- matrix(NA, n1, n2)
    for (i in 1:n2) {
      K[, i] <-  sigmaF^2*exp(-(2*(sin(pi*abs(x - y[i])/d)^2)) / ell1^2)*
        exp(-0.5*((x - y[i])^2/ell2^2))
    }
    return(K)
  }
  class(rval) <- "kernel"
  return(rval)
}

sigmaF <- 20
ell <- c(1,10)
d <- 365/sd(dataTullinge$time)


GPfitPeriodic <- gausspr(temp ~ time, data = dataTullinge,
                         kernel = periodicKernel,
                         kpar = list(sigma = sigmaF,
                                     ell = ell,
                                     period = d),
                         var = sigmaSq_n
```

```
                          )


posteriorGPeriodic <- posteriorGP(x = dataTullinge$time,
                                  y = dataTullinge$temp,
                                  xStar = xGrid,
                                  K = periodicKernel(sigma = sigmaF,
                                                     ell = ell,
                                                     period = d),
                                  hyperParam = NULL,
                                  sigmaNoise = sqrt(sigmaSq_n))

predictedPeriodic <- predict(GPfitPeriodic,
                             data.frame(time = xGrid))




plotGPperiodic <- dataTullinge %>%
  ggplot() +
  geom_point(aes(y = temp, x = time)) +
  theme_minimal() +
  geom_line(data = data.frame(fbar = as.vector(predictedPeriodic),
                              x = xGrid),
            aes(x = x, y = fbar), col = "red") +
  geom_ribbon(data = data.frame(f = as.vector(predictedPeriodic),
                                x = xGrid,
                                var = diag(posteriorGPeriodic$Fcovar)),
              aes(x = x,
                  ymin = (f - 1.96*sqrt(var)),
                  ymax = (f + 1.96*sqrt(var))),
              alpha = 0.2)

plotGPperiodicVSsqExp <- dataTullinge %>%
  ggplot() +
  geom_point(aes(y = temp, x = time)) +
  theme_minimal() +
  geom_line(data = data.frame(fbar = c(as.vector(predictedQ2b),
                                       as.vector(predictedPeriodic),
                                       rep(as.vector(predictedQ2Day), 6)[1:length(xGrid)]),
                              Kernel = c(rep("Squared Exponential (Time)", length(as.vector(predictedQ2
                                         rep("Periodic (Time & Day)", length(as.vector(predictedPeriodi
                                         rep("Squared Exponential (Day)", length(as.vector(predictedPer
                              x = xGrid),
            aes(x = x, y = fbar, col = Kernel))


plotGPperiodicVSsqExp + theme(legend.position="bottom")


# Question 3

dataBank <- read.csv(paste(pathData,"banknoteFraud.csv", sep = "/"),
```

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



# Question 3 (a)

fraudGPclass <- gausspr(fraud ~ varWave + skewWave,
                        data = dataBank[trainOBS,])
trainPredProbabilities <- predict(fraudGPclass,
                              newdata = dataBank[trainOBS,],
                              type = "probabilities")
trainPredLabel <- predict(fraudGPclass,
                      newdata = dataBank[trainOBS,],
                      type = "response")

dataGrid <- expand.grid(seq(min(dataBank[trainOBS,]$varWave),
                            max(dataBank[trainOBS,]$varWave),
                            length.out = 100),
                        seq(min(dataBank[trainOBS,]$skewWave),
                            max(dataBank[trainOBS,]$skewWave),
                            length.out = 100))
colnames(dataGrid) <- c("varWave", "skewWave")
predPrGrid <- predict(fraudGPclass,
                      newdata = dataGrid,
                      type = "probabilities")

dataGrid <- cbind(dataGrid, predPrGrid)
colnames(dataGrid) <- c(colnames(dataGrid)[1:2], "Pr0", "Pr1")

#  Plot
cbind(dataBank[trainOBS,],
      data.frame(Pr = ifelse(dataBank[trainOBS,]$fraud == 1,
                             trainPredProbabilities[,2],
                             trainPredProbabilities[,1]),
                 Pr2 = predPrGrid[,2])) %>%
  ggplot() +
  geom_point(aes(x = varWave,
                 y = skewWave,
                 col = fraud)) +
  geom_contour(data = dataGrid,
               aes(x = varWave,
                   y = skewWave,
                   z =  Pr1),
               col = "black") +
  theme_minimal() +
  theme(legend.position = "bottom")
```

```r
confTableTrainPred <- table("True" = dataBank[trainOBS,]$fraud,
                            "Predicted" = trainPredLabel)


kable(confTableTrainPred, caption = "Confusion Matrix, Training Data")

accurTrain <- sum(diag(confTableTrainPred))/sum(confTableTrainPred)

printString <- sprintf("Accuracy, %s data: %f", "Train", accurTrain)
cat(printString)


# Question 3 (b)

testPredLabel <- predict(fraudGPclass,
                         newdata = dataBank[-trainOBS,],
                         type = "response")

confTableTestPred <- table("True" = dataBank[-trainOBS,]$fraud,
                           "Predicted" = testPredLabel)


kable(confTableTestPred, caption = "Confusion Matrix, Test Data")

accurTest <- sum(diag(confTableTestPred))/sum(confTableTestPred)


printString <- sprintf("Accuracy, %s data: %f", "Test", accurTest)
cat(printString)


# Question 3 (c)

fraudGPclassAllcov <- gausspr(fraud ~ varWave + skewWave + kurtWave + entropyWave,
                              data = dataBank[trainOBS,])


testPredLabelAllcov <- predict(fraudGPclassAllcov,
                               newdata = dataBank[-trainOBS,],
                               type = "response")


confTableTestPredAllcov <- table("True" = dataBank[-trainOBS,]$fraud,
                           "Predicted" = testPredLabelAllcov)


kable(confTableTestPredAllcov, caption = "Confusion Matrix, Test Data (All four Covariates)")

accurTestAllcov <- sum(diag(confTableTestPredAllcov))/sum(confTableTestPredAllcov)


printString <- sprintf("Accuracy, %s data: %f", "Test (All four Covariates)", accurTestAllcov)
cat(printString)
```

# 732A96: Advance Machine Learning

LAB 4: STATE SPACE MODELS

*Arian Barakat/ariba405*

**Background:**

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

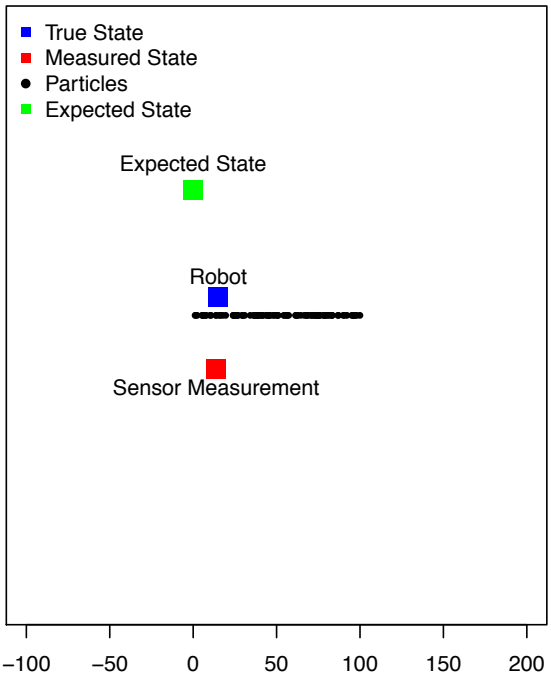
See instruction sheet for model specification

Implement the SSM above. Run it for $T = 100$ time steps to obtain $z_{1:100}$ (i.e. states) and $x_{1:100}$ (i.e. observations). Use the observations (i.e. sensor readings) to identify the state (i.e. robot location) via particle filtering. Use 100 particles. For each time step, show the particles, the expected location and the true location. Repeat the exercise after replacing the standard deviation of the emission model with 5 and then with 50. Comment on how this affects the results. Finally, show and explain what happens when the weights in the particle filter are always equal to 1, i.e. there is no correction.

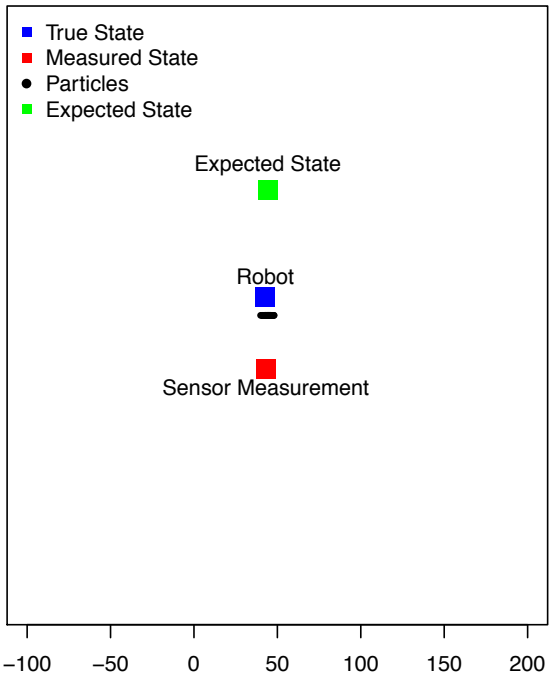
## Standard Deviation: 1

### Link To Video

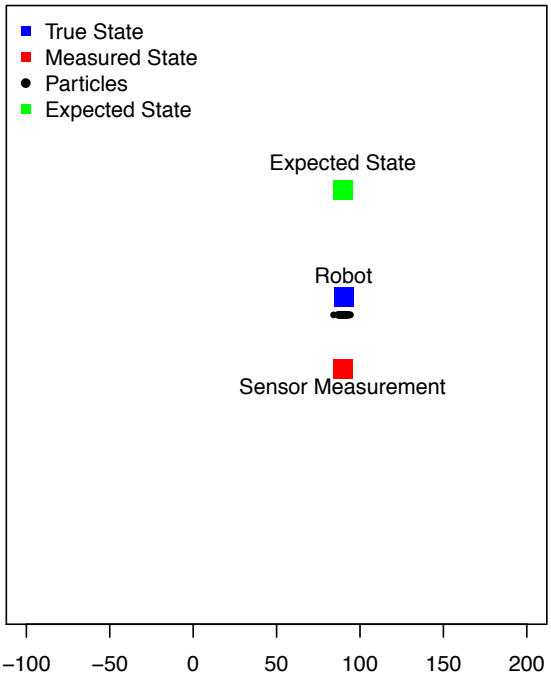
**Time Step: 1**



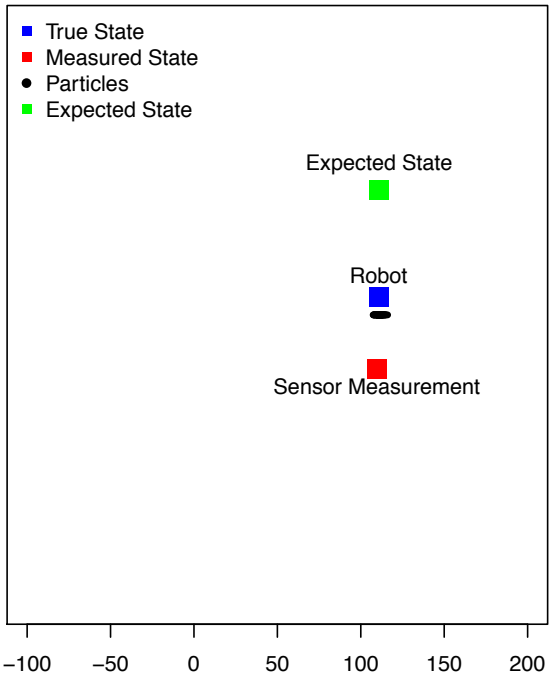

**Time Step: 40**




**Time Step: 80**



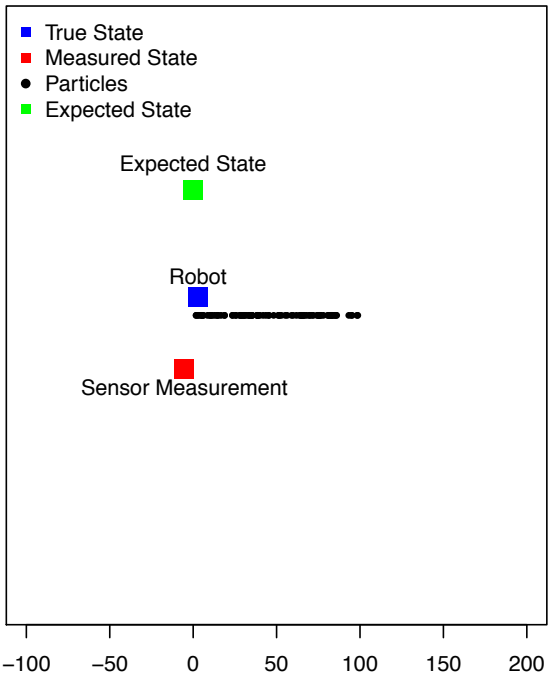

**Time Step: 100**



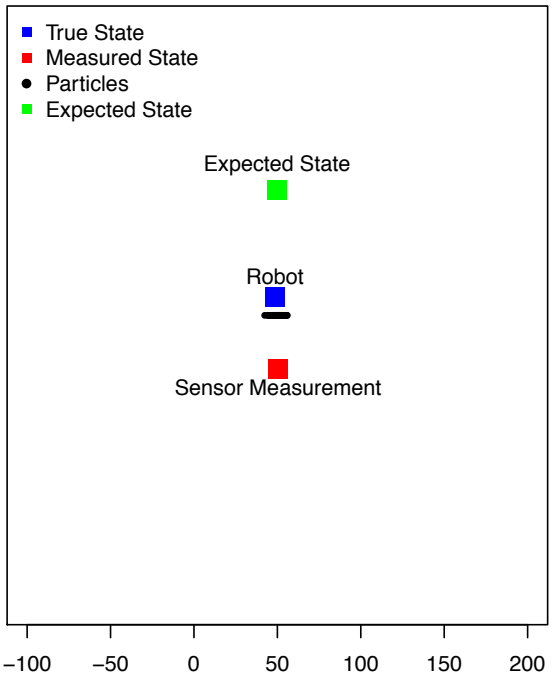
## Standard Deviation: 5

### Link To Video

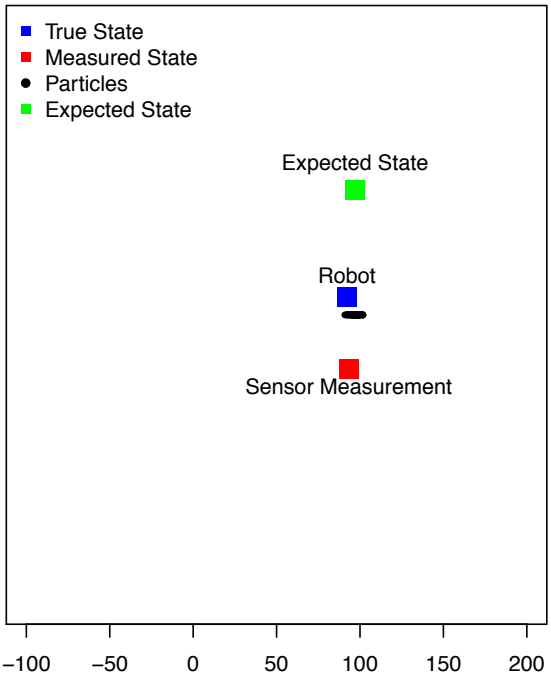
**Time Step: 1**



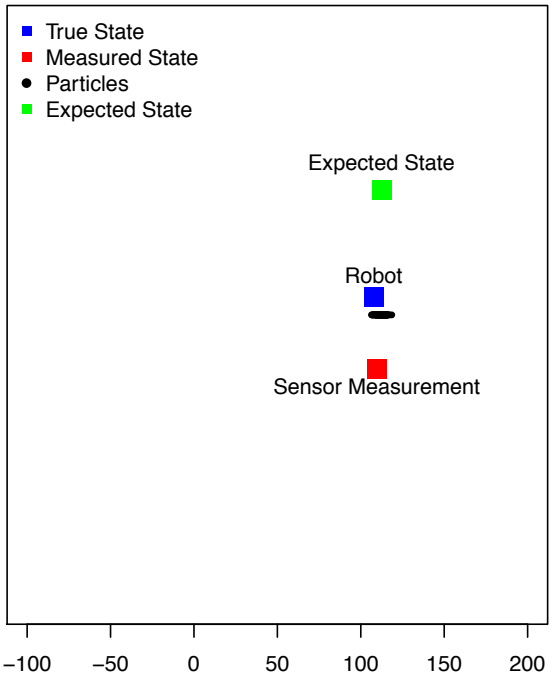

**Time Step: 40**




**Time Step: 80**



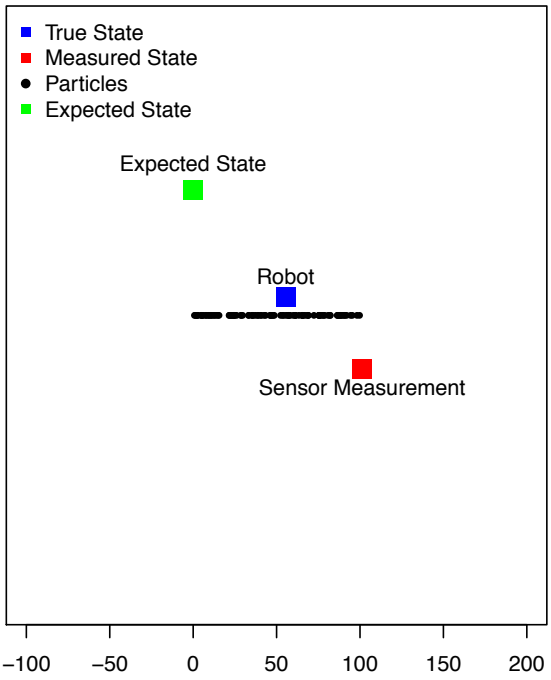

**Time Step: 100**



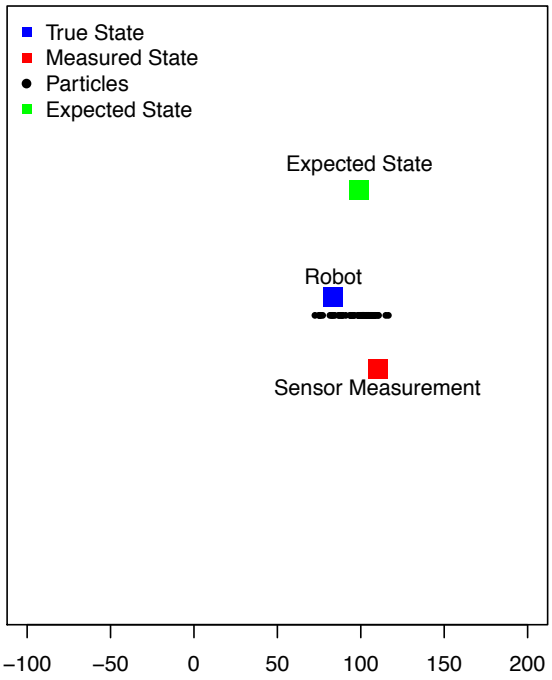
## Standard Deviation: 50

### Link To Video

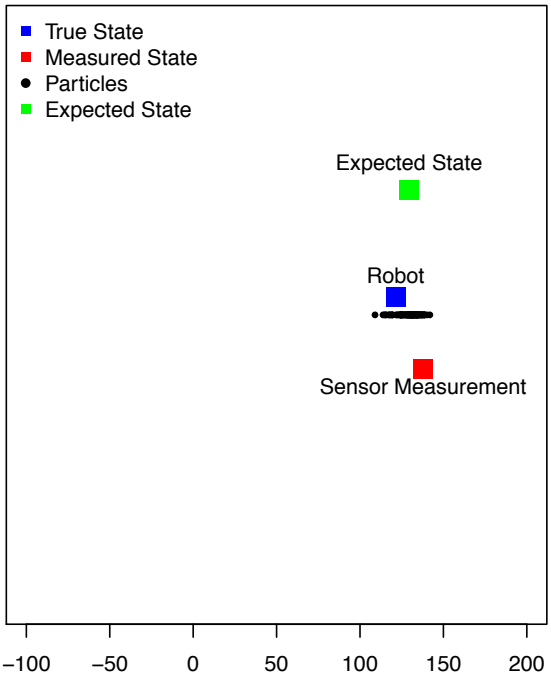
**Time Step: 1**



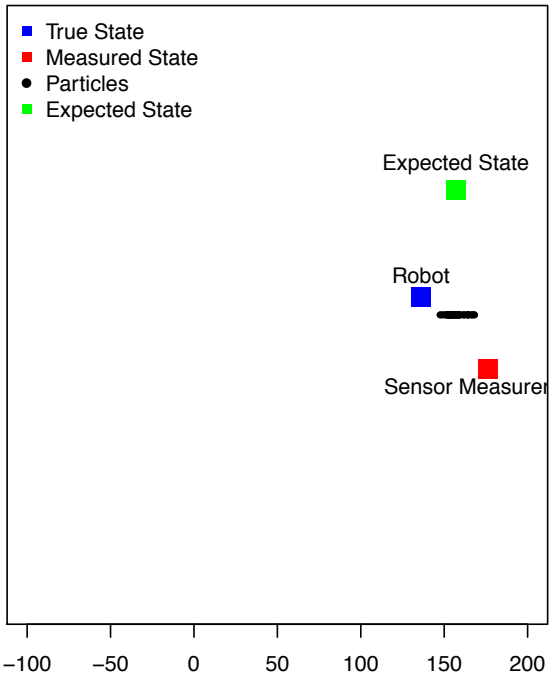

**Time Step: 40**




**Time Step: 80**



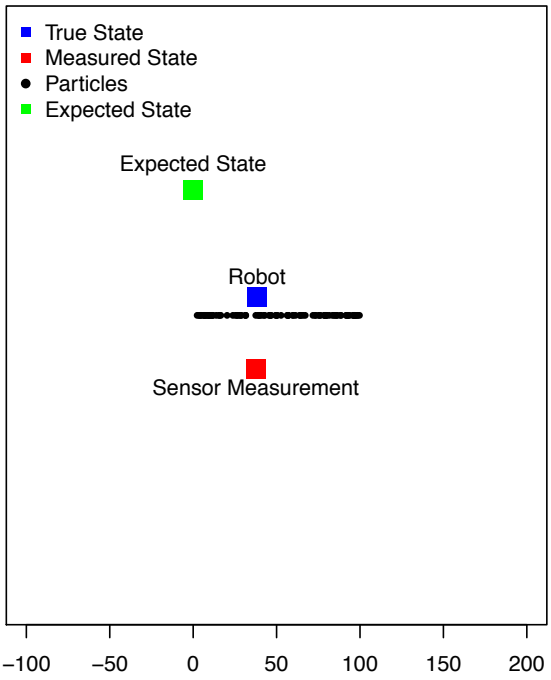

**Time Step: 100**



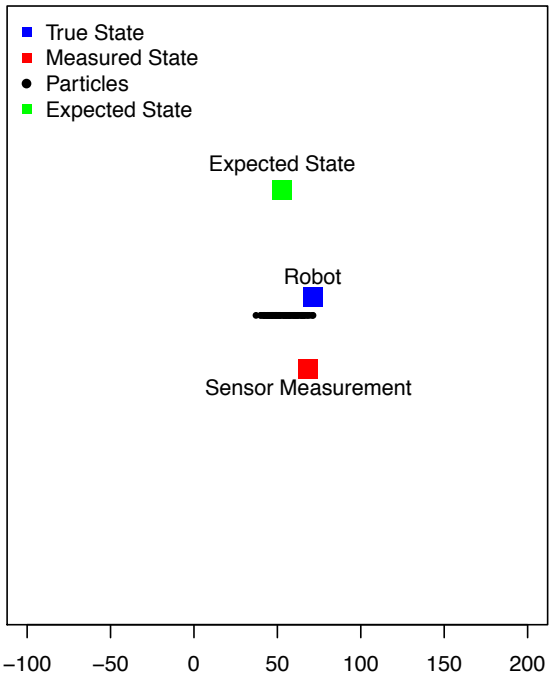
## Equal Weight, Standard Deviation: 1

### Link To Video

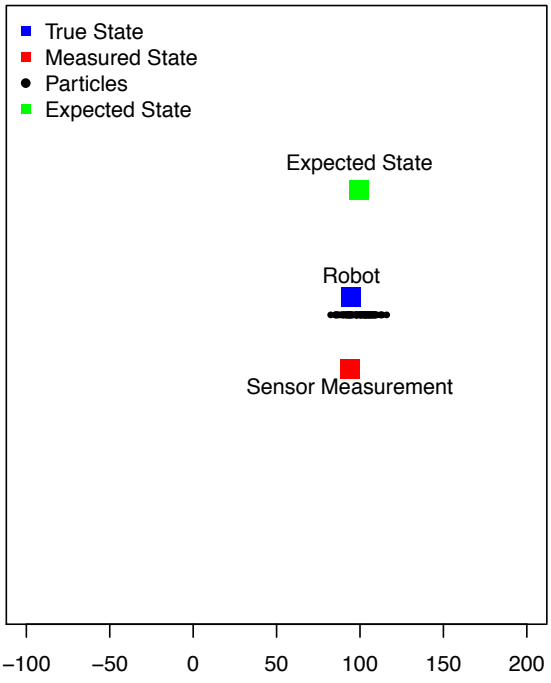
**Time Step: 1**



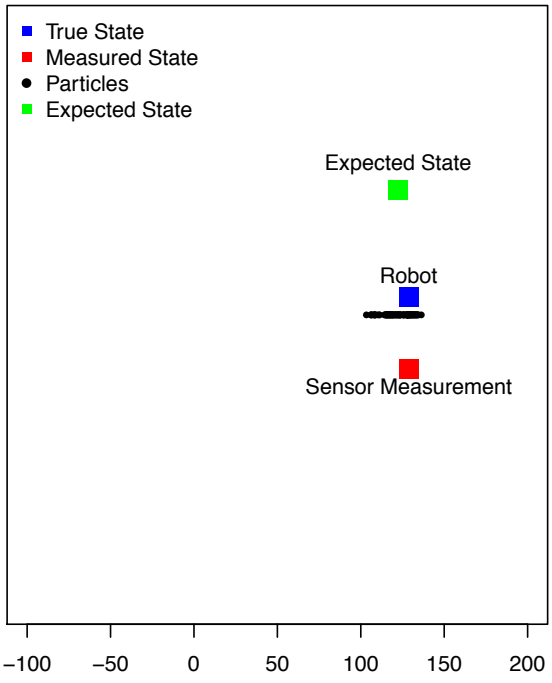

**Time Step: 40**



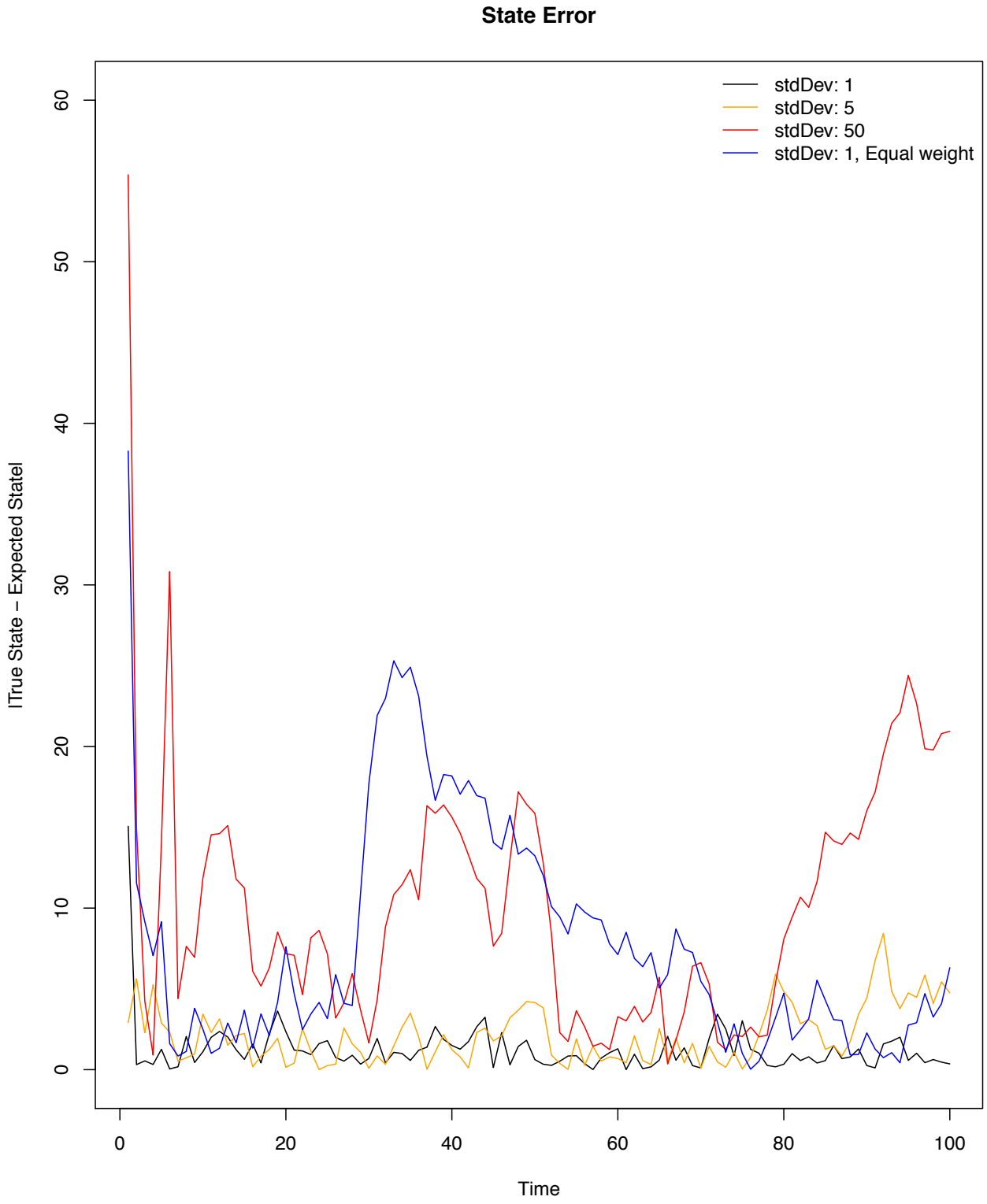

**Time Step: 80**




**Time Step: 100**



## State Error


State Error
stdDev: 1
stdDev: 5
stdDev: 50
stdDev: 1, Equal weight
|True State – Expected State|
Time
0
10
20
30
40
50
60
0
20
40
60
80
100

## Discussion

By increasing the standard deviation for the emission model we're implying that we're less certain about the sensor measurements. This, in turn, will affect the weight distribution of our particles (distant particles from the true state will be given a higher weight compared a to a simulation with a lower standard deviation), which affect the position of the expected state. The consequence of this can be viewed in the *state error* figure, where we can observe that the absolute difference between the true and expected state increases as we increase the standard deviation.

In the case when we're giving all particles equal weight the problem of absolute difference is similarly ill-behaved as in the case when the standard deviation is set to 50. Once again, the behavior can be linked to the fact that distant particles (from the true state) have a relatively high weight.

# Appendix

```
knitr::opts_chunk$set(echo = FALSE,
                      warning = FALSE,
                      message = FALSE,
                      fig.height = 11,
                      fig.width = 9)

library(ggplot2)
library(dplyr)
library(grid)
library(gridExtra)
library(knitr)
library(png)
pathImg <- "../img"
pathData <- "../Data"
pathVid <- "/Users/arianbarakat/Dropbox/Public"



transModGeneric <- function(standDev){
  # Description:
  #     Function using closjure
  # Input:
  #     standDev: The Standard Deviation for the Normals
  # Returns:
  #     Model: Returns a function that calculates p(z_t+1 | z_t)
  #             linear combination of normals

  model <- function(z_t, z_tprev, sdDev = standDev){
    (dnorm(x =z_t , mean = z_tprev, sd = sdDev) +
      dnorm(x =z_t , mean = z_tprev + 1, sd = sdDev) +
      dnorm(x =z_t , mean = z_tprev + 2, sd = sdDev))/3
  }
  return(model)
}

emissModGeneric <- function(standDev){
  # Description:
  #     Function using closjure
  # Input:
  #     standDev: The Standard Deviation for the Normals
  # Returns:
  #     Model: Returns a function that calculates p(x_t | z_t),
  #             linear combination of normals

    model <- function(x_t, z_t, sdDev = standDev){
    (dnorm(x = x_t, mean = z_t, sd = sdDev) +
      dnorm(x = x_t, mean = z_t + 1, sd = sdDev) +
      dnorm(x = x_t, mean = z_t - 1, sd = sdDev))/3
  }
  return(model)
}
```

```r
calcWeight <- function(sample, state_t,  emissModel, case = c("normal", "equalWeight")){
  # Description:
  #     Function that calculates the weights of the particles
  #     in each time step
  # Input:
  #     sample:     x_t
  #     state_t:    z_t()
  #     emissModel: function that calculates p(x_t | z_t)
  #     case:       Normal particle filter algorithm or
  #                 special case when all weight are equal (w = 1)
  # Returns:
  #     w: Weights of the particles in each time step

  if(!is.character(case) && (case %in% c("normal", "equalWeight"))){
    stop("Supply case: 'normal' or 'equalWeight' in character form")
  }

  if(case == "normal"){
    emissVec <- do.call(emissModel, args = list(x_t = sample,
                                                z_t = state_t))

    w <- emissVec/sum(emissVec)
  }

  if(case == "equalWeight"){
    w <- rep(1, times = length(state_t))
    w <- w/sum(w)
  }

  return(w)
}

sampleObsStateGeneric <- function(standDev){
  # Description:
  #     Return a function (using closujure) that create a sample of x_t+1
  #     given z_t
  # Input:
  #     standDev: The Standard Deviation for the Normals
  # Returns:
  #     Model:

  model <- function(z_t, stDev = standDev, ...){
    mu <- c(z_t, z_t + 1, z_t - 1)

    comp <- sample(1:length(mu), size = 1, prob = rep(1/length(mu), times = 3))
    rnorm(n = 1, mean = mu[comp], sd = stDev)
  }

  return(model)
}

sampleHiddenStateGeneric <- function(standDev){
  # Description:
```

```
  #     Return a function (using closujure) that create a sample of z_t+1
  #     given z_t and weights. Function 13.119 in Bishop (p. 646)
  # Input:
  #     standDev: The Standard Deviation for the Normals
  # Returns:
  #     Model:

  model <- function(weights, statesPrev, stDev = standDev){

    if(length(weights) == 1){
      sampleMeans <- statesPrev
    } else {
      sampleMeans <- sample(statesPrev,
                            size = length(weights),
                            replace = TRUE,
                            prob = weights)

    }


    newStates <- vapply(sampleMeans, FUN = function(z_t, stdDev = stDev){

      mu <- c(z_t, z_t + 1, z_t + 2)
      comp <- sample(1:length(mu),
                     size = 1,
                     prob = rep(1/length(mu),
                                times = length(mu)))

      rnorm(n = 1, mean = mu[comp], sd = stDev)

    }, FUN.VALUE = numeric(1))

   return(newStates)
  }
  return(model)
}


initialModel <- function(minimum, maximum){
  model <- function(nSample, min = minimum, max = maximum){
    runif(n = nSample, min = min, max = max)
  }
  return(model)
}


particleFilter <- function(nStep, nParticles, standardDev, initMod, weightCase = "normal"){

  # Setup
  prEmiss <- emissModGeneric(standDev = standardDev)
  prTrans <- transModGeneric(standDev = 1)
  sampleXt <- sampleObsStateGeneric(standDev = standardDev)
  sampleZt <- sampleHiddenStateGeneric(standDev = 1)
```

```
  zParticles_tm <- matrix(nrow = nStep,
                ncol = nParticles)

  trueState <- vector("numeric", length = nStep)
  sensorRobot <- vector("numeric", length = nStep)
  expectedState <- vector("numeric", length = nStep)

  # Generate Data
  trueState[1] <- do.call(initMod, args = list(nSample = 1))
  sensorRobot[1] <- sampleXt(trueState[1])

  for(step in 2:nStep){
    trueState[step] <- sampleZt(weights = c(1),
                                statesPrev = trueState[step -1])
    sensorRobot[step] <- sampleXt(trueState[step])

  }



  # Init
  zParticles_tm[1,] <- z_t <- do.call(initMod, args = list(n = nParticles))

  for(step in 2:nStep){

    x_t <- sensorRobot[step]

    w_t <- calcWeight(sample = x_t,
                      state_t = z_t,
                      emissModel = prEmiss,
                      case = weightCase)

    zParticles_tm[step,] <- z_t <- sampleZt(weights = w_t,
                                            statesPrev = zParticles_tm[step -1,])
    expectedState[step] <- sum(w_t*z_t)
  }

  return(list("trueStates" = trueState,
              "sensorMeasurment" = sensorRobot,
              "statesParticles" = zParticles_tm,
              "exptectedState" = expectedState))
}



plotSSMsim <- function(object, plotToReport = FALSE,
                       plotToVideo = FALSE,
                       videoArgs = list("path" = NULL, "name" = NULL) ){
  suppressMessages(suppressWarnings(require(animation)))

  nIter <- length(object[["trueStates"]])

  if(plotToReport){
```

```
    stepLength <- c(1, round(c(nIter*0.4, nIter*0.8)),nIter)
  } else {
    stepLength <- seq(1, nIter, by = 1)
  }

  plotOut <- function(plotObject, plotSequence){
    for(step in plotSequence){

      plot(plotObject$statesParticles[step,],
           y = rep(2, 100),
           xlim = c(-100, 200), xlab ="", ylab = "",
           pch = 16, cex = .7, yaxt = "n",  main = paste("Time Step:", step))
      points(x = plotObject$trueStates[step], y = 2.05, pch = 15, col = "blue", cex = 2)
      text(x = plotObject$trueStates[step], y = 2.11, labels = "Robot", cex = 1)
      points(x = plotObject$sensorMeasurment[step], y = 1.85, pch = 15, col = "red", cex = 2)
      text(x = plotObject$sensorMeasurment[step], y = 1.80, labels = "Sensor Measurement", cex = 1)
      points(x = plotObject$exptectedState[step], y = 2.35, pch = 15, col = "green", cex = 2)
      text(x = plotObject$exptectedState[step], y = 2.42, labels = "Expected State", cex = 1)
      legend("topleft",
             legend = c("True State",
                        "Measured State",
                        "Particles",
                        "Expected State"),
             pch = c(15,15,16, 15),
             col = c("blue", "red", "black", "green"),
             bty = "n")
      Sys.sleep(0.1)
    }
  }


  if(plotToReport){
    par(mfrow = c(2,2))
    plotOut(plotObject = object, plotSequence = stepLength)
    par(mfrow = c(1,1))
  }

  if(plotToVideo){
    saveVideo({
      plotOut(plotObject = object, plotSequence = stepLength)
    },
    interval = 0.1,
    video.name = paste(videoArgs[[1]], videoArgs[[2]], sep = "/"),
    ani.width=800,
    ani.height=800)
  }

  if(!plotToReport){
    plotOut(plotObject = object, plotSequence = stepLength)
  }

}
```

```r
plotError <- function(object, add = FALSE, lineCol = NULL){
  steps <- length(object[["trueStates"]])
  if(!add){
    plot(x = 1:steps,
         y = abs(object$trueStates - object$exptectedState),
         type = "l",
         xlab = "Time",
         ylab = "|True State - Expected State|",
         main = "State Error",
         ylim = c(0,60))
  }

  if(add){
    if(is.null(lineCol)){
      stop("Provide line colour")
    }
    lines(x = 1:steps,
         y = abs(object$trueStates - object$exptectedState),
         col = lineCol)
  }

}



######## Running the Code


set.seed(1991)
initModd <- initialModel(minimum = 0, maximum = 100)
simSSMsd1 <- particleFilter(nStep = 100,
                            nParticles = 100,
                            standardDev = 1,
                            initMod = initModd)

simSSMsd5 <- particleFilter(nStep = 100,
                            nParticles = 100,
                            standardDev = 5,
                            initMod = initModd)

simSSMsd50 <- particleFilter(nStep = 100,
                            nParticles = 100,
                            standardDev = 50,
                            initMod = initModd)

simSSMsd1_equalWeight <- particleFilter(nStep = 100,
                                        nParticles = 100,
                                        standardDev = 1,
                                        initMod = initModd,
                                        weightCase =  "equalWeight")


# plotSSMsim(simSSMsd1, plotToVideo = TRUE, videoArgs = list(pathVid, "simSSMsd1.mp4"))
```

```r
plotSSMsim(simSSMsd1, plotToReport = TRUE)
```

```r
# plotSSMsim(simSSMsd5, plotToVideo = TRUE, videoArgs = list(pathVid, "simSSMsd5.mp4"))
plotSSMsim(simSSMsd5, plotToReport = TRUE)
```

```r
# plotSSMsim(simSSMsd50, plotToVideo = TRUE, videoArgs = list(pathVid, "simSSMsd50.mp4"))
plotSSMsim(simSSMsd50, plotToReport = TRUE)
```

```r
# plotSSMsim(simSSMsd1_equalWeight, plotToVideo = TRUE, videoArgs = list(pathVid, "simSSMsd1_equalWeigh
plotSSMsim(simSSMsd1_equalWeight, plotToReport = TRUE)
```

```r
plotError(simSSMsd1)
plotError(simSSMsd5, add = TRUE, lineCol = "orange")
plotError(simSSMsd50, add = TRUE, lineCol = "red")
plotError(simSSMsd1_equalWeight, lineCol = "blue", add = TRUE)
legend("topright",
       legend = c("stdDev: 1",
                  "stdDev: 5",
                  "stdDev: 50",
                  "stdDev: 1, Equal weight"),
       lty = rep(1, times = 4),
       col = c("black", "orange", "red", "blue"),
       bty = "n")
```