Advanced ML exam

1

In order to learn 80 percent, I will dicard two nodes of the data set. The dataset has 5000 observations, so I will get 80 percent randomly from sampling

```
library(bnlearn)
```

```
##
## Attaching package: 'bnlearn'

## The following object is masked from 'package:stats':
##
##     sigma
```

```
library(gRain)
```

```
## Loading required package: gRbase

##
## Attaching package: 'gRbase'

## The following objects are masked from 'package:bnlearn':
##
##     ancestors, children, parents
```

```
data(asia)
set.seed(12345)
index<- sample(1:5000,4000)
myasiaTrain<- asia[index,]
myasiaTest<- asia[-index,]
```

Now we learn the parameters by using naive.bayes on our training data to learn the model and predicting on it. The result is the confusion matrix below

```
bn = naive.bayes(myasiaTrain, "S")
fitted = bn.fit(bn, myasiaTrain)
pred = predict(fitted, myasiaTest)
table(pred, myasiaTest[, "S"])
```

```
##
## pred   no yes
##   no  349 188
##   yes 119 344
```

Now, you are asked to classify S given observations only for the so-called Markov blanket of S, i.e. its parents plus its children plus the parents of its children minus S itself. Report again the confusion matrix.(1 p)

```
# plot(bn)
#
# ncols<-colnames(myasiaTest)
# ncols<- ncols[-2]
# plot(bn)
# fitted = bn.fit(bn, myasiaTrain)
# pred = predict(fitted, myasiaTest)
# table(pred, myasiaTest[, -2])
```

Explain why the results of the previous exercises coincide (as long as you use the same BN learned)

The result coincide beacuse the graph is a DAG learnt by all the other nodes and since all other outcomes are learnt but the S itself, is predicting just on one variable

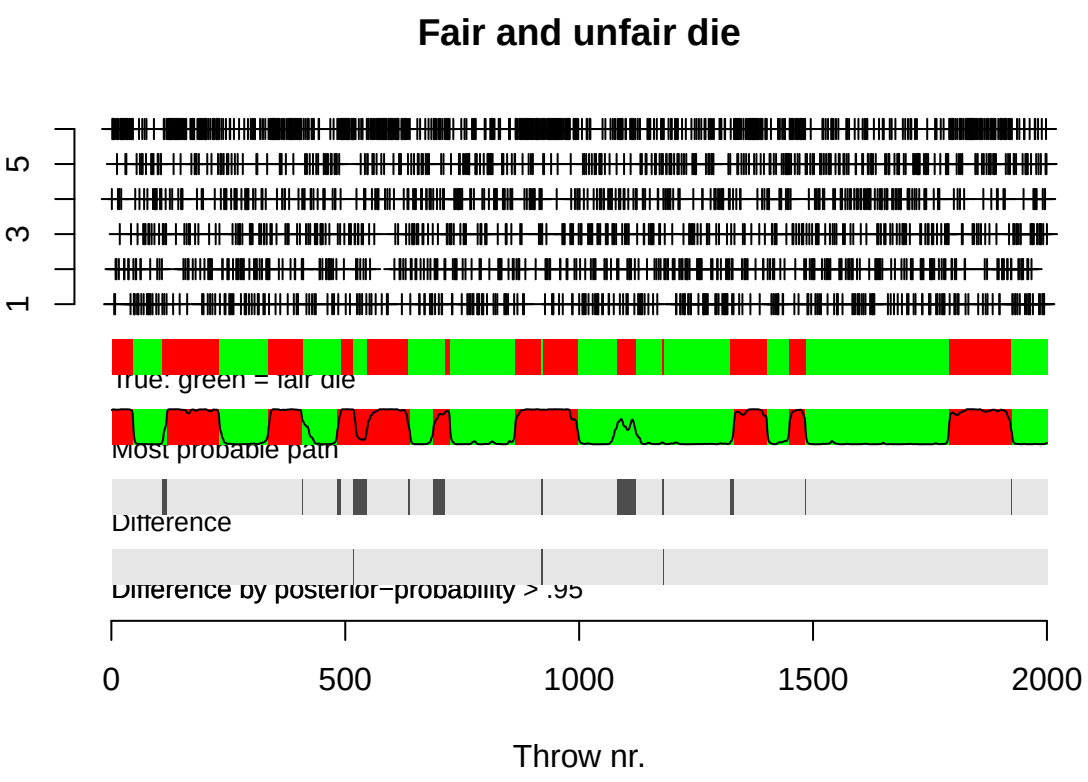
2

a)

The hmm I am doing will have 6 states (3 for dice 1 and 3 for dice 2), having each dice the possibility of being repeated 0.5, and 0.5 to move to the next dice. Doing this, it is probable that it will stay always more than three turns, but at least it will be 3 in each dice. Finally, it will change of dice as well for die 1 to 2 with probability 0.5. Dice1 will be the fair one and dice two the not fair one.

```
library(HMM)
dishonestCasino()
```

```
## Plot simulated throws:
```



```
## Simulated, which die was used:
##
## Most probable path (viterbi):
##
## Differences:
##
## Posterior-probability:
##
## Difference with classification by posterior-probability:
##
## Difference with classification by posterior-probability > .95:
```

```
States<- c("die1_1","die1_2","die1_3","die2_1","die2_2","die2_3")
Symbols<-1:6
Trans<-matrix(c(0.5,0.5,0,0,0,0,
                0,0.5,0.5,0,0,0,
                0,0,0.5,0.5,0,0,
                0,0,0,0.5,0.5,0,
                0,0,0,0,0.5,0.5,
                0.5,0,0,0,0,0.5),6,6, byrow =TRUE)

Emission<-matrix(NA,6,6)
Emission[1:3,]<- rep(1/6,6) # Fair dice
Emission[4,]<- c(1/10,1/10,1/10,1/10,1/10,1/2)
Emission[5,]<- c(1/10,1/10,1/10,1/10,1/10,1/2)
Emission[6,]<- c(1/10,1/10,1/10,1/10,1/10,1/2)# Fair dice
startprobs<- c(0.5,0,0,0.5,0,0) # It starts in position 1 or 4, which means first dice first throw or 2

myhmm<- initHMM(States, Symbols, startprobs, Trans, Emission)
##Sampling
simHMM(myhmm, 50)
```

```
## $states
##  [1] "die1_1" "die1_2" "die1_3" "die1_3" "die2_1" "die2_2" "die2_2"
##  [8] "die2_3" "die2_3" "die1_1" "die1_2" "die1_2" "die1_2" "die1_2"
## [15] "die1_3" "die2_1" "die2_1" "die2_1" "die2_1" "die2_1" "die2_2"
## [22] "die2_3" "die2_3" "die2_3" "die2_3" "die1_1" "die1_1" "die1_2"
## [29] "die1_3" "die1_3" "die2_1" "die2_2" "die2_2" "die2_2" "die2_2"
## [36] "die2_3" "die2_3" "die1_1" "die1_1" "die1_2" "die1_3" "die2_1"
## [43] "die2_1" "die2_2" "die2_2" "die2_3" "die1_1" "die1_1" "die1_2"
## [50] "die1_3"
##
## $observation
##  [1] 1 6 5 5 6 6 6 6 2 4 6 1 2 4 4 1 3 6 3 5 6 6 6 6 6 1 2 3 4 6 6 5 5 4 6
## [36] 6 1 2 1 5 4 6 1 6 6 6 1 5 3 3
```

It can be seen that each die is used 3 times at least consecutively.

b)

Explain how to learn the parameters of a HMM given a sample of observations: If our model is a Bayesian network, and we have observed a full data set $X = x_1, .., x_N$, we can determine the parameters

of an HMM using maximum likelihood. The likelihood function is obtained from the joint distribution by marginalizing over the latent obeserved variables such that:

$$p(X|\theta) = \sum_z p(X, Z|\theta)$$

If it is an incomplete sample, it is necessary to use the EM algorithm (the expectation maximization). The EM algorithm begins with some initial selection for the model parameters (prior theta) . In the E step, this parameter is taken to find the posterior distribution of the latent variables

$$p(Z|X, priortheta)$$

. We then use this posterior distribution to evaluate the expectation of the logarithm of the complete-data likelihood function, as a function of the parameters thera to give the function $Q(theta, priortheta)$.

If our data is Markov, we will be using the Iterative Proportional Fitting Procedure, wehere we using also the log maximum likelihood estimation with derivatives.

3

This is the kernel provided by Mattias:

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

a

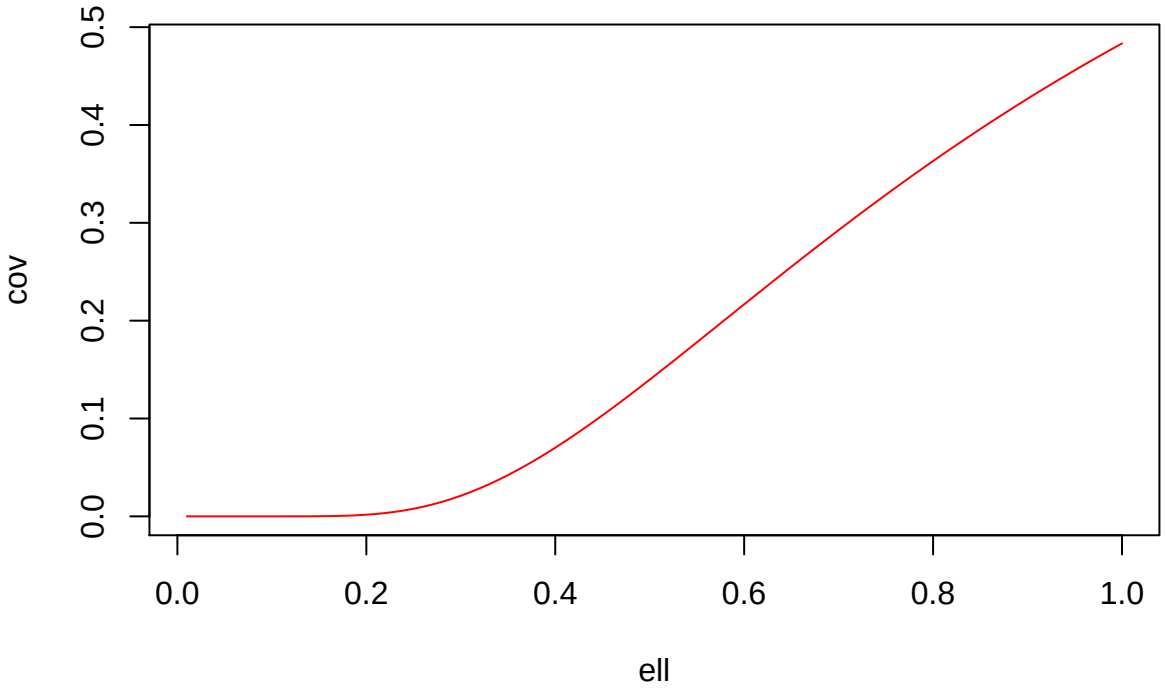
```
##Initial values given
library("ggplot2")
library("kernlab")
```

```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
library("mvtnorm")
sigmaf = 1
x1<-0
x2<-1
ell = seq(0.01,1,by=0.01)
```

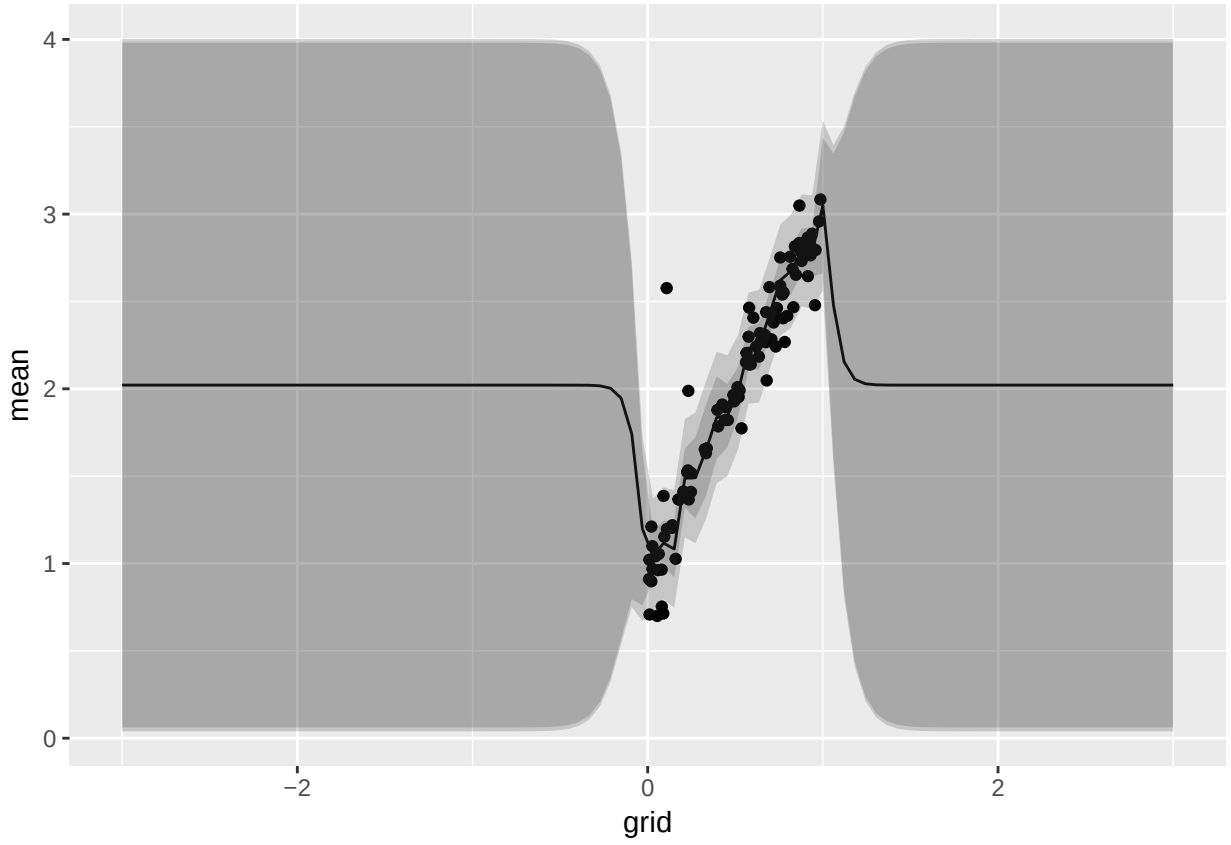
```r
cov<-integer(length(ell))
for(i in 1:length(ell)){
  k2<-k(sigmaf = sigmaf,ell = ell[i])
  cov[i]<-kernelMatrix(k2,x1,x2) #only one observation

}

plot(x = ell, y = cov, type = "l", col = "red")
```



The graphic shows how much correlated or how much is the covariance k(0, 0.1) is affected by a change of the hyperparameter ell. The hyperparameter ell is an smoother such that the larger ell is, the kernel gets smoother and so the correlation of k(0,0.1). ###b For l equal to 0.2

```r
##Initial values given
load("~/given_files/Exam/GPdata.RData")
sigmaN<-0.15**2
ell<-0.2
sigmaf<-1
grid<-seq(-3,3,length.out = 100)
model<-k(sigmaf = sigmaf, ell = ell)
cov<-kernelMatrix(model, x,y)
##mean
fit<-gausspr(x=x, y = y,kernel =model, kpar= list(sigma = sigmaf, ell= ell), var = sigmaN)
mean<-predict(fit, grid)
##postCov
postCov<-kernelMatrix(model,grid,grid)-kernelMatrix(model, grid,x)%*%solve(kernelMatrix(model,x,x)+diag

mysd<-diag(postCov)
lower95<-mean-1.96*mysd
high95<-mean+1.96*mysd
lowerpred<-mean-1.96*sqrt(mysd**2+sigmaN)
```

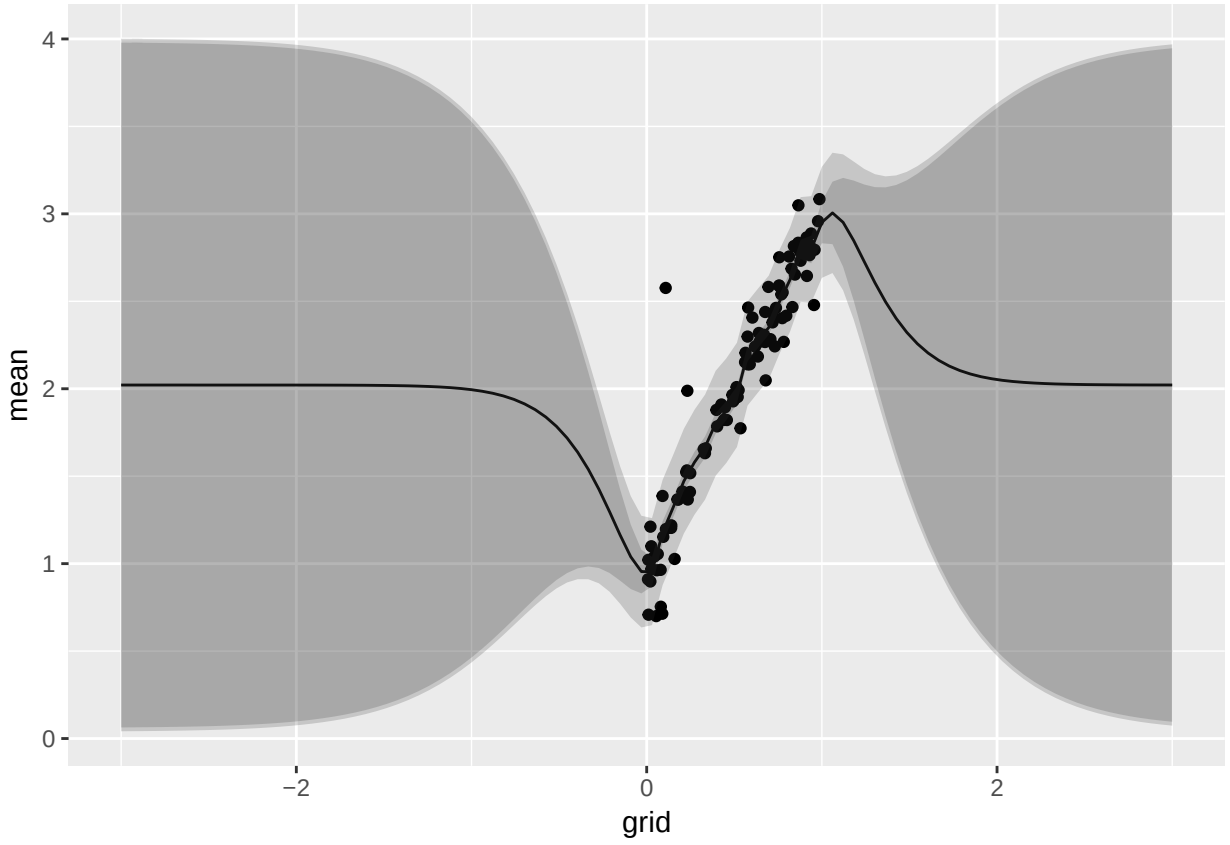
```
highpred<-mean+1.96*sqrt(mysd**2+sigmaN)

myframe<- data.frame(x=x, y = y, grid = grid,
                     mean = mean,
                     lowerpred=lowerpred,
                     highpred=highpred,
                     lower95 = lower95,
                     high95=high95)

###Plotting

ggplot2::ggplot(myframe, aes(x=grid))+
  geom_line(aes(x = grid,y = mean))+
  geom_point(aes(x=x, y =y))+
  geom_ribbon(aes(ymin=lower95, ymax=high95 ), alpha = 0.2)+
  geom_ribbon(aes(ymin=lowerpred, ymax=highpred ), alpha = 0.2)
```



Now, for l equal to 1

```
##Initial values given

ell<-1
grid<-seq(-3,3,length.out = 100)
model<-k(sigmaf = sigmaf, ell = ell)
cov<-kernelMatrix(model, x,y)
##mean
```

```
fit<-gausspr(x=x, y = y,kernel =model, kpar= list(sigma = sigmaf, ell= ell), var = sigmaN)
mean<-predict(fit, grid)
##postCov
postCov<-kernelMatrix(model,grid,grid)-kernelMatrix(model, grid,x)%*%solve(kernelMatrix(model,x,x)+diag

mysd<-diag(postCov)
lower95<-mean-1.96*mysd
high95<-mean+1.96*mysd
lowerpred<-mean-1.96*sqrt(mysd**2+sigmaN)
highpred<-mean+1.96*sqrt(mysd**2+sigmaN)

myframe<- data.frame(x=x, y = y, grid = grid,
                     mean = mean,
                     lowerpred=lowerpred,
                     highpred=highpred,
                     lower95 = lower95,
                     high95=high95)

###Plotting

ggplot2::ggplot(myframe, aes(x=grid))+
  geom_line(aes(x = grid,y = mean))+
  geom_point(aes(x=x, y =y))+
  geom_ribbon(aes(ymin=lower95, ymax=high95 ), alpha = 0.2)+
  geom_ribbon(aes(ymin=lowerpred, ymax=highpred ), alpha = 0.2)
```



In the graphs we can see that the larger the ell the smoother our mean and confidence bands prediction are. This is of course correlated with what i commented in part a. Also, it can be seen that when there are less or no points in an range, there width of the interval increases a lot, making the mean go in our case to 2.

The problem with this model is that we are trying to predict points given the distance between our observed points. In our dataset, data is really compressed in between one range so that the matern model will give a more stable confidence interval and mean, but the problem is that it is not possible to make realistic predictions outside this range of points beacuse we dont have points outside this range. Also, outside the range, the prediction goes to 2, which is a value that does not explain well the data.

For these reasons, I would say that GP model is too complex for this dataset given that we dont have middle points that we will be interested in predicting. It looks that a normal regression could work in this model for the same dataset.

4

a

To make an state Spacemodel, the first thing I need is to have a latent variable for my real observation, and those are going to be given by the following code from my lab:

```
set.seed(12345)
#######Generating model
#initial model
p1<-runif(1,0,100)
# Transition
sd_T<-1
sd_E<-5

Transition<- function(n,p0, sd_2){
  z<- integer(n)
  z[1]<-p0
  for(i in 2:n){
    p<- sample(c(z[i-1]+2,z[i-1],z[i-1]+1),1)
    z[i]<-rnorm(1,p,sqrt(sd_2))
  }
  return(z)
}

##Emission model
Emission<- function(vectorz, sd_2){
  n<- length(vectorz)
  x<- integer(n)
  for(i in 1:n){
    p<- sample(c(vectorz[i]-1,vectorz[i],vectorz[i]+1),1)
    x[i]<-rnorm(1,p,sqrt(sd_2))
}
return(x)
}
myZ<-Transition(100, p0= p1, sd_2= sd_T**2)

x_t<- Emission(myZ, sd_2 = sd_E**2)
```

Now, I am going to try to make a model that will predict my next observation based on the observed plus

the mean difference between the next observation and the predicted observation.

$$\mu_t = \mu_{t-1} + (mu[t-1] - z[t-1])$$

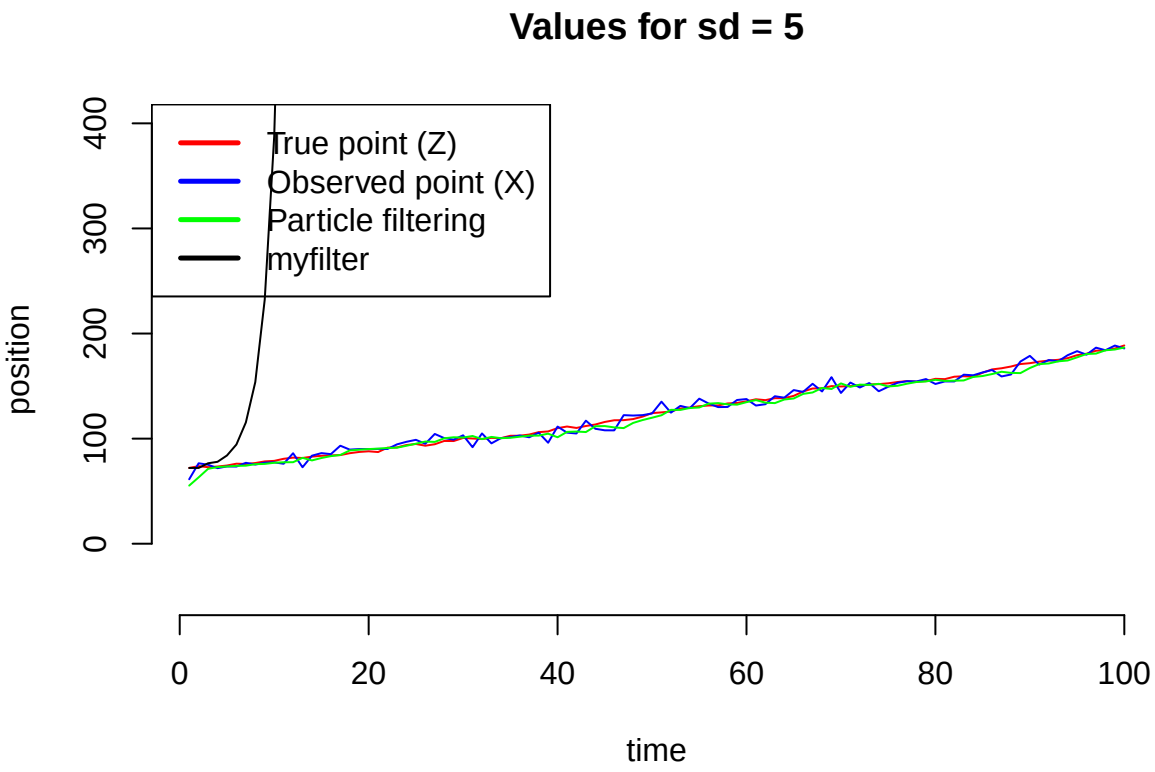
$$\sigma_t = \sigma_{t-1} + mean(differencemean)$$

```
######Kalman filter
#Specifications of variables
t_total <- 100
Sigma_0 <- sd_E
mu_0 <- p1


my_filter <- function(t_total, Sigma_0, mu_0, z) {
  # Arguments follow notation on the slides, Lecture SSM 1, slide 11, except for
  # all them are scalars and not matrices/vectors.
  #
  # Returns:
  # List:
  # $mu
  # $sigma
  t_total <- t_total
  mu <- 0
  sigma <- 0
  mu[1] <- mu_0
  sigma[1] <- Sigma_0
  differencemean<-0
  differencesigma<-0
  for (t in 2:t_total) {

    mu[t] <- mu[t-1] + differencemean[t-1]
    differencemean[t]<-abs(mu[t]-z[t])
    sigma[t] <- sigma[t-1] + mean(differencemean)
    differencesigma[t]<-sd(mu-z[1:t])
  }
  return(list(mu = mu, sigma = sigma))
}



myfilter<-my_filter(t_total, Sigma_0, mu_0, x_t)
```

Here I write the particle filter

```
######################
weights<- rep(0.01,100)
X<-runif(100,0,100)
ParticleFilter<- function(simulations, init_weights, grid, sd_E, sd_T, x_t= x_t){
  n<- length(grid)
  parameters<- matrix( ncol = n,nrow = simulations)
  newweigths<- matrix(ncol = n,nrow = simulations)
  newweigths[1,]<-init_weights/sum(init_weights)
  parameters[1,]<-sample(x =grid,size = n, replace=TRUE, prob = newweigths[1,])
  ### sample(dnorm(parameters[1,],grid) Left that
```

```
  xt<- integer(100)
  for(i in 2:simulations){
    for(j in 1:simulations){
      xt[j]<-Transition(2,parameters[i-1,j],sd_T)[2]
    }
    newweigths[i,]<-dnorm(x_t[i-1], xt, sqrt(sd_E))/sum(dnorm(x_t[i-1], xt,sqrt(sd_E)))
    parameters[i,]<-sample(xt, size =n, replace=TRUE, prob = newweigths[i,])
    }
  result<- list(parameters = parameters, weights=newweigths)
  return(result)
}


sd_2T<-1
sd_2E<-1
Myparticle<-ParticleFilter(simulations =100, init_weights = weights, grid= X,
sd_T = sd_T, sd_E = sd_E,x_t= x_t)
parameter_est<-Myparticle$parameters

par(mfrow=c(1,1))
plot(0, xlim= c(1,100), ylim = c(-50,400),
bty='n',pch='',ylab='position',xlab='time', main = paste0("Values for sd = ", sd_E))
lines(x = 1:100, y = myZ, col ="red")
lines(x = 1:100, y = x_t, col ="blue")
lines(x =1:100, y = apply(Myparticle$parameters,1,mean), col =
        "green")
lines(x = 1:100, y = myfilter$mu, col ="black")

legend("topleft", # places a legend at the appropriate place
c("True point (Z) ", "Observed point (X)", "Particle filtering","myfilter"), # puts text in the legend
lty=c(1,1), # gives the legend appropriate symbols (lines)
lwd=c(2.5,2.5),col=c("Red"," blue", "Green", "black")) # gives the legend lines the correct color and w
```



They both work, but my filter goes up crazily so I will need time to change staff, but I have not had time. ###b

The differences between the Kalman filter and the particle filter is that the kalman filter is a Gaussian Process model which has a prediction component and a correction component over the prediction, but is simpler in the sense that it does not use probabilistic inference and it is always unimodal on its prediction. On the other hand, the particle filter uses probabilistic inference on the prediction and it weigths these predictions over all the possible ones upgrading and giving more weights to the more likely ones based on prior observations as well. For that, it can also be multimodal.