part3

Omkar Bhutra (omkbh878)

9 January 2020

Lab 3 stuff

Q1. 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 Bishops book and/or the slides for the last lecture on state space models (SSMs). The robot moves along the horizontal axis according to the following SSM:

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

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

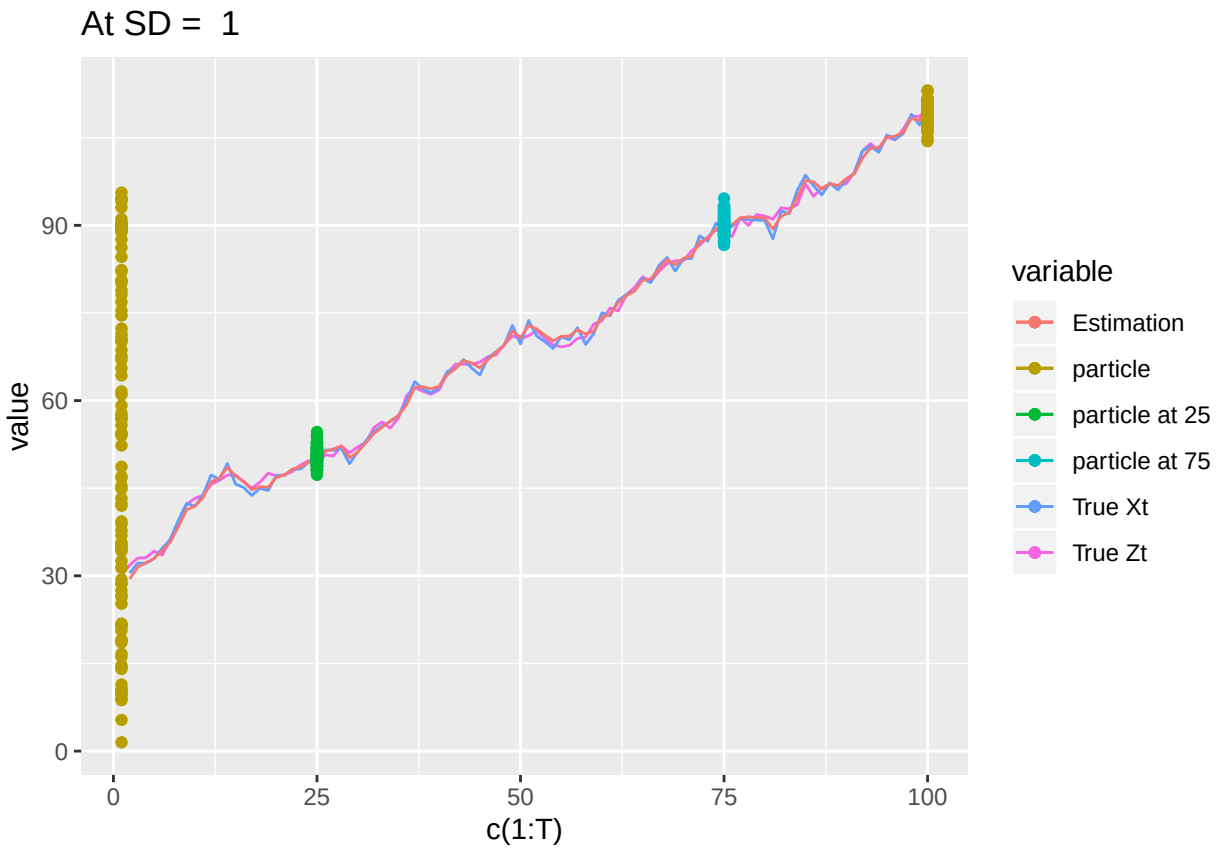
Initial Model: $p(z_1) = Uniform(0, 100)$

A) Implement the SSM above. Simulate 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. Show the particles, the expected location and the true location for the first and last time steps, as well as for two intermediate time steps of your choice.

```
initModel <- function(len){
  x <- runif(len,0,100)
  return(x)
}

transitionmodel <- function(zt){
  probs = rep(1/3,3)
  draw = sample(1:3,1,prob = probs)
  if(draw==1){
    normTrans <- rnorm(1,zt,transition_sd)
  }
  else if(draw==2){
    normTrans <- rnorm(1,zt+1,transition_sd)
  }
  else{
    normTrans <- rnorm(1,zt+2,transition_sd)
  }
  return(normTrans)
}
```

```r
emmisionmodel <- function(zt,emission_sd){
  probs = rep(1/3,3)
  draw = sample(1:3,1,prob = probs)

  if(draw==1){
    normEmis <- rnorm(1,zt,emission_sd)
  }
  else if(draw==2){
    normEmis <- rnorm(1,zt-1,emission_sd)
  }
  else{
    normEmis <- rnorm(1,zt+1,emission_sd)
  }
  return(normEmis)
}

emission_density <- function(xt,zt){
  x <- (dnorm(xt,zt,emission_sd)+dnorm(xt,zt-1,emission_sd)+dnorm(xt,zt+1,emission_sd))/3
  return(x)
}


emission_sd = 1
transition_sd = 1

ssm <- function(emission_sd,transition_sd){
  T = 100
  particles = 100
  zt = xt = error = rep(NA,T)

  # for zt and xt
  zt[1] = initModel(1)
  for(i in 2:T){
    zt[i] <- transitionmodel(zt[i-1])
    xt[i] <- emmisionmodel(zt[i],emission_sd)
  }

  initParticles <- initModel(particles)
  particleWeights<- rep(1/particles,particles)
  estimation <- c()
  Particles25 = Particles75 = NA

  for(i in 2:T){
    newParticles <- sample(1:particles,particles,prob=particleWeights,replace = T)
    initParticles <- sapply(initParticles[newParticles],transitionmodel)

    if(i == 25){
      Particles25 <- c(initParticles)
    }
    else if(i == 75){
      Particles75 <- c(initParticles)
    }
```

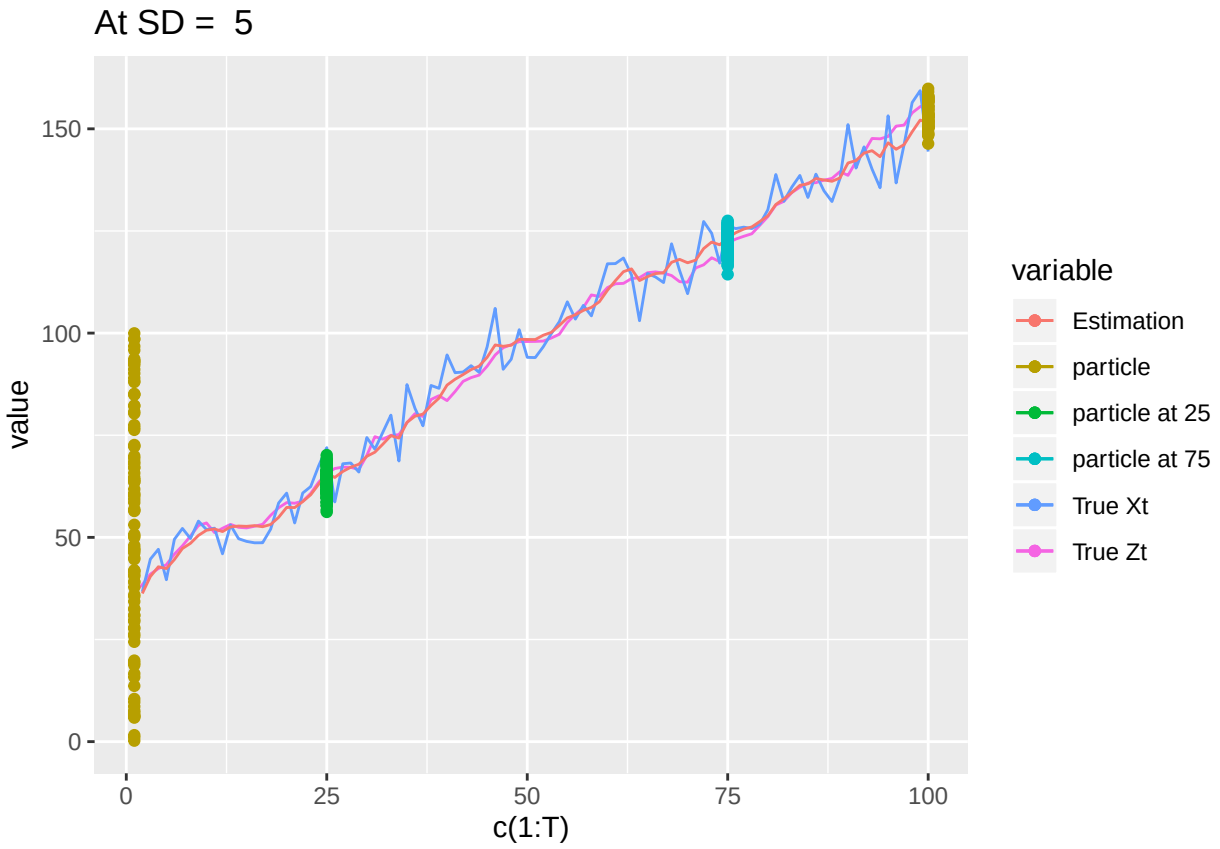
```
    for(j in 1:particles){
      particleWeights[j] <- emission_density(xt[i],initParticles[j])
    }
    #Normalise
    particleWeights <- particleWeights/sum(particleWeights)

    Ezt <- sum(particleWeights*initParticles)
    error[i] <- abs(zt[i]-Ezt)
    estimation[i] <- sum(particleWeights * initParticles)
  }

  data = data.frame(zt,xt,estimation,particles = initModel(particles),Particles25,Particles75,initParti

  ggplot(data,aes(x=c(1:T),y=value,color=variable,xlab="timestep"))+
    geom_line(aes(y=data$zt,col='True Zt'))+
    geom_line(aes(y=data$xt,col='True Xt'))+
    geom_line(aes(y=data$estimation,col='Estimation'))+
    geom_point(aes(x=rep(1,T),y=data$particles,col='particle'))+
    geom_point(aes(x=rep(25,T),y=data$Particles25,col='particle at 25'))+
    geom_point(aes(x=rep(75,T),y=data$Particles75,col='particle at 75'))+
    geom_point(aes(x=rep(T,T),y=data$initParticles,col='particle'))+
    ggtitle(paste('At SD = ',emission_sd))
}
ssm(emission_sd ,transition_sd )
```



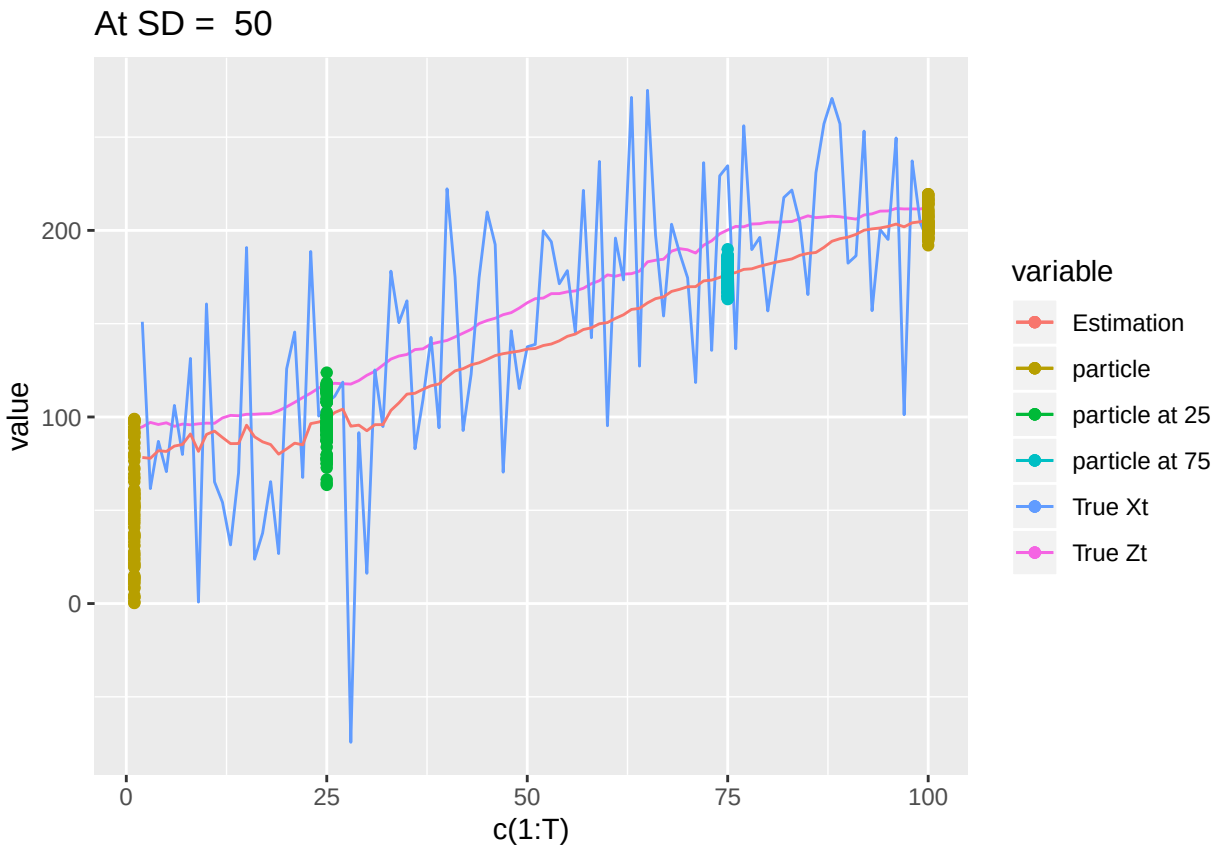
Intermediate particles number 25 and 75 are chosen and their locations are mapped in the figure.

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

```
emission_sd = 5
ssm(emission_sd ,transition_sd )
```




```
emission_sd = 50
ssm(emission_sd ,transition_sd )
```

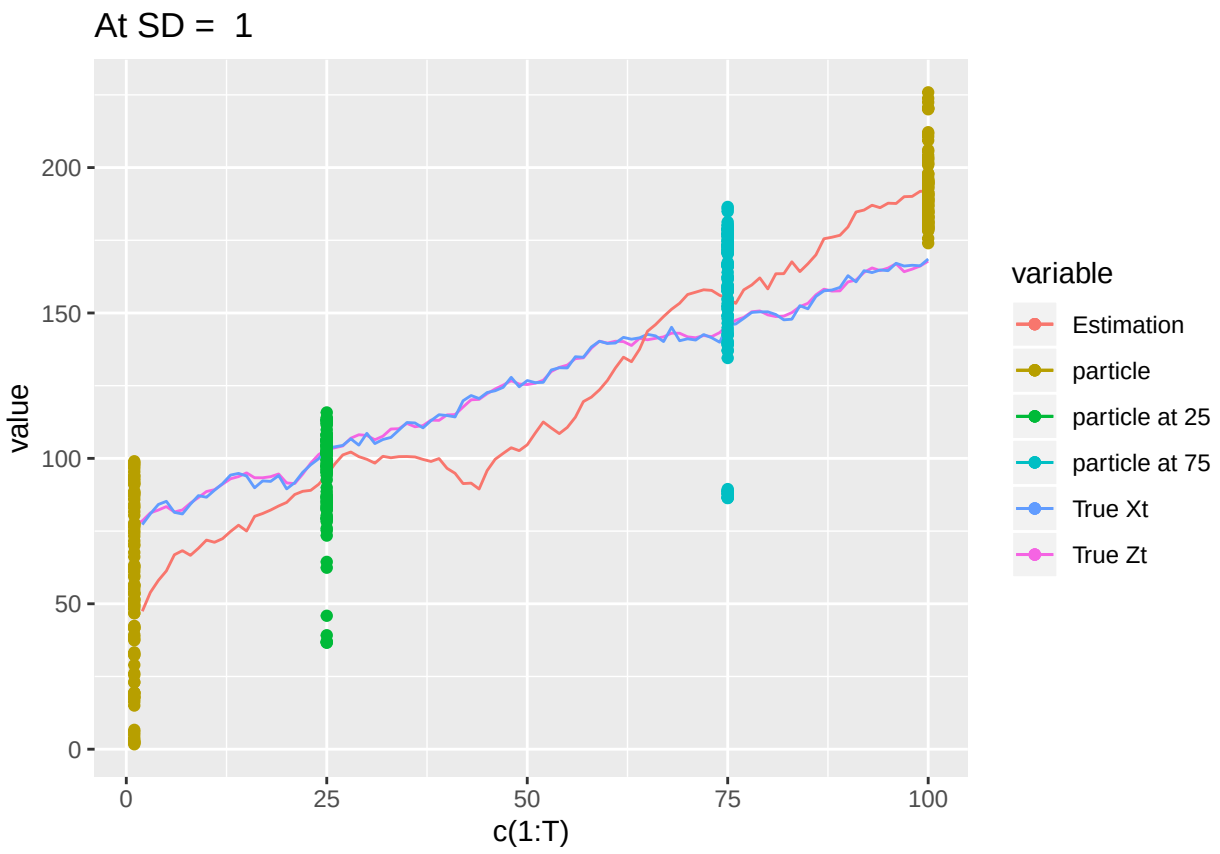


Increasing the standard deviation not only makes the true x_t scattered but also scatters the particles both intermediate and endstep.

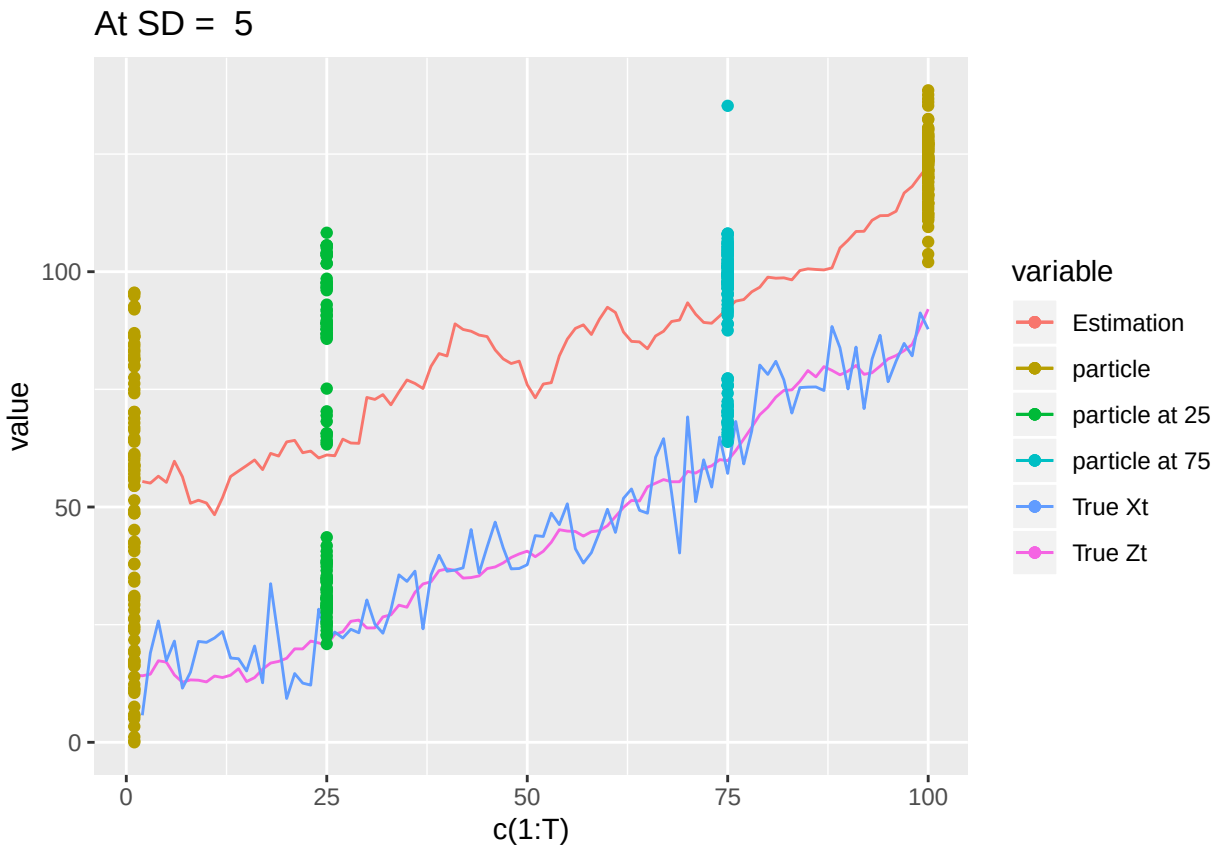
C) Finally, show and explain what happens when the weights in the particle filter are always equal to 1, i.e. there is no correction.

```
ssm <- function(emission_sd,transition_sd){
  T = 100
  particles = 100
  zt = xt = error = rep(NA,T)

  # for zt and xt
  zt[1] = initModel(1)
  for(i in 2:T){
    zt[i] <- transitionmodel(zt[i-1])
    xt[i] <- emmisionmodel(zt[i],emission_sd)
  }

  initParticles <- initModel(particles)
  particleWeights<- rep(1/particles,particles)
  estimation <- c()
  Particles25 = Particles75 = NA

  for(i in 2:T){
```

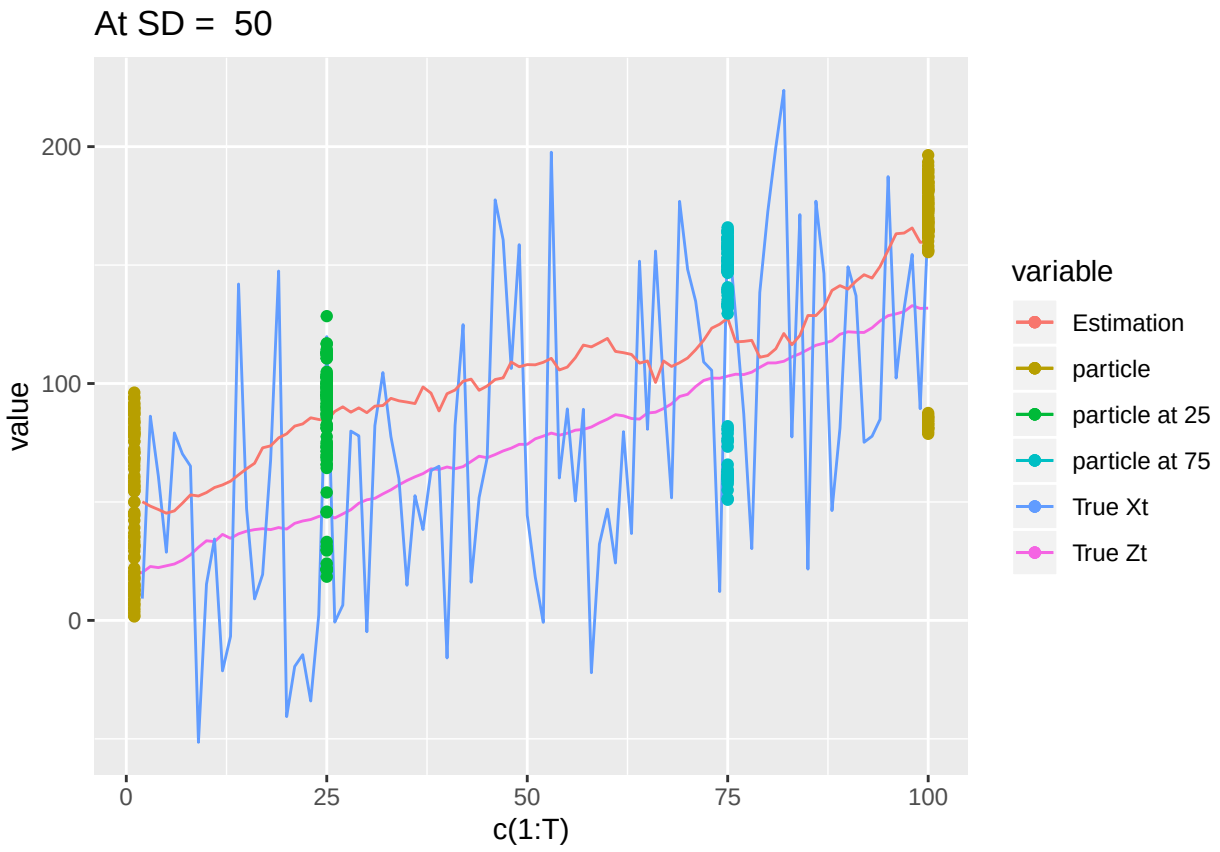
```
    newParticles <- sample(1:particles,particles,prob=particleWeights,replace = T)
    initParticles <- sapply(initParticles[newParticles],transitionmodel)

    if(i == 25){
      Particles25 <- c(initParticles)
    }
    else if(i == 75){
      Particles75 <- c(initParticles)
    }


    for(j in 1:particles){
      particleWeights[j] <- 1
    }
    #Normalise
    particleWeights <- particleWeights/sum(particleWeights)

    Ezt <- sum(particleWeights*initParticles)
    error[i] <- abs(zt[i]-Ezt)
    estimation[i] <- sum(particleWeights * initParticles)
  }

  data = data.frame(zt,xt,estimation,particles = initModel(particles),Particles25,Particles75,initParti

  ggplot(data,aes(x=c(1:T),y=value,color=variable,xlab="timestep"))+
    geom_line(aes(y=data$zt,col='True Zt'))+
    geom_line(aes(y=data$xt,col='True Xt'))+
    geom_line(aes(y=data$estimation,col='Estimation'))+
    geom_point(aes(x=rep(1,T),y=data$particles,col='particle'))+
    geom_point(aes(x=rep(25,T),y=data$Particles25,col='particle at 25'))+
    geom_point(aes(x=rep(75,T),y=data$Particles75,col='particle at 75'))+
    geom_point(aes(x=rep(T,T),y=data$initParticles,col='particle'))+
    ggtitle(paste('At SD = ',emission_sd))
}

emission_sd = 1
ssm(emission_sd ,transition_sd )
```




```
emission_sd = 5
ssm(emission_sd ,transition_sd )
```




```
emission_sd = 50
ssm(emission_sd ,transition_sd )
```



It is seen that the estimation and true values do not match and accuracy is reduced. This change is consistent from all the standard deviation runs of 1,5 and 50. value of the particle at all timesteps are scattered as compared to earlier runs. It reduces the quality of the filtering.

```
knitr::opts_chunk$set(echo = TRUE)
library("dplyr")
library("ggplot2")
library("KFAS")
initModel <- function(len){
  x <- runif(len,0,100)
  return(x)
}

transitionmodel <- function(zt){
  probs = rep(1/3,3)
  draw = sample(1:3,1,prob = probs)
  if(draw==1){
    normTrans <- rnorm(1,zt,transition_sd)
  }
  else if(draw==2){
    normTrans <- rnorm(1,zt+1,transition_sd)
  }
  else{
    normTrans <- rnorm(1,zt+2,transition_sd)
  }
  return(normTrans)
```

```
}

emmisionmodel <- function(zt,emission_sd){
  probs = rep(1/3,3)
  draw = sample(1:3,1,prob = probs)

  if(draw==1){
    normEmis <- rnorm(1,zt,emission_sd)
  }
  else if(draw==2){
    normEmis <- rnorm(1,zt-1,emission_sd)
  }
  else{
    normEmis <- rnorm(1,zt+1,emission_sd)
  }
  return(normEmis)
}

emission_density <- function(xt,zt){
  x <- (dnorm(xt,zt,emission_sd)+dnorm(xt,zt-1,emission_sd)+dnorm(xt,zt+1,emission_sd))/3
  return(x)
}


emission_sd = 1
transition_sd = 1

ssm <- function(emission_sd,transition_sd){
  T = 100
  particles = 100
  zt = xt = error = rep(NA,T)

  # for zt and xt
  zt[1] = initModel(1)
  for(i in 2:T){
    zt[i] <- transitionmodel(zt[i-1])
    xt[i] <- emmisionmodel(zt[i],emission_sd)
  }

  initParticles <- initModel(particles)
  particleWeights<- rep(1/particles,particles)
  estimation <- c()
  Particles25 = Particles75 = NA

  for(i in 2:T){
    newParticles <- sample(1:particles,particles,prob=particleWeights,replace = T)
    initParticles <- sapply(initParticles[newParticles],transitionmodel)

    if(i == 25){
      Particles25 <- c(initParticles)
    }
    else if(i == 75){
      Particles75 <- c(initParticles)
```

```
    }


    for(j in 1:particles){
      particleWeights[j] <- emission_density(xt[i],initParticles[j])
    }
    #Normalise
    particleWeights <- particleWeights/sum(particleWeights)

    Ezt <- sum(particleWeights*initParticles)
    error[i] <- abs(zt[i]-Ezt)
    estimation[i] <- sum(particleWeights * initParticles)
  }

  data = data.frame(zt,xt,estimation,particles = initModel(particles),Particles25,Particles75,initParti

  ggplot(data,aes(x=c(1:T),y=value,color=variable,xlab="timestep"))+
    geom_line(aes(y=data$zt,col='True Zt'))+
    geom_line(aes(y=data$xt,col='True Xt'))+
    geom_line(aes(y=data$estimation,col='Estimation'))+
    geom_point(aes(x=rep(1,T),y=data$particles,col='particle'))+
    geom_point(aes(x=rep(25,T),y=data$Particles25,col='particle at 25'))+
    geom_point(aes(x=rep(75,T),y=data$Particles75,col='particle at 75'))+
    geom_point(aes(x=rep(T,T),y=data$initParticles,col='particle'))+
    ggtitle(paste('At SD = ',emission_sd))
}
ssm(emission_sd ,transition_sd )
emission_sd = 5
ssm(emission_sd ,transition_sd )

emission_sd = 50
ssm(emission_sd ,transition_sd )
ssm <- function(emission_sd,transition_sd){
  T = 100
  particles = 100
  zt = xt = error = rep(NA,T)

  # for zt and xt
  zt[1] = initModel(1)
  for(i in 2:T){
    zt[i] <- transitionmodel(zt[i-1])
    xt[i] <- emmisionmodel(zt[i],emission_sd)
  }

  initParticles <- initModel(particles)
  particleWeights<- rep(1/particles,particles)
  estimation <- c()
  Particles25 = Particles75 = NA

  for(i in 2:T){
    newParticles <- sample(1:particles,particles,prob=particleWeights,replace = T)
    initParticles <- sapply(initParticles[newParticles],transitionmodel)
```

```
    if(i == 25){
      Particles25 <- c(initParticles)
    }
    else if(i == 75){
      Particles75 <- c(initParticles)
    }


    for(j in 1:particles){
      particleWeights[j] <- 1
    }
    #Normalise
    particleWeights <- particleWeights/sum(particleWeights)

    Ezt <- sum(particleWeights*initParticles)
    error[i] <- abs(zt[i]-Ezt)
    estimation[i] <- sum(particleWeights * initParticles)
  }

  data = data.frame(zt,xt,estimation,particles = initModel(particles),Particles25,Particles75,initPartic

  ggplot(data,aes(x=c(1:T),y=value,color=variable,xlab="timestep"))+
    geom_line(aes(y=data$zt,col='True Zt'))+
    geom_line(aes(y=data$xt,col='True Xt'))+
    geom_line(aes(y=data$estimation,col='Estimation'))+
    geom_point(aes(x=rep(1,T),y=data$particles,col='particle'))+
    geom_point(aes(x=rep(25,T),y=data$Particles25,col='particle at 25'))+
    geom_point(aes(x=rep(75,T),y=data$Particles75,col='particle at 75'))+
    geom_point(aes(x=rep(T,T),y=data$initParticles,col='particle'))+
    ggtitle(paste('At SD = ',emission_sd))
}

emission_sd = 1
ssm(emission_sd ,transition_sd )

emission_sd = 5
ssm(emission_sd ,transition_sd )

emission_sd = 50
ssm(emission_sd ,transition_sd )
# Question 3: GP
### Question 3(a)
# Change to your path
library(kernlab)
library(mvtnorm)

# From KernelCode.R:
# Squared exponential, k
k <- function(sigmaf = 1, ell = 1)
{
  rval <- function(x, y = NULL)
  {
    r = sqrt(crossprod(x-y))
```

```
    return(sigmaf^2*exp(-r^2/(2*ell^2)))
  }
  class(rval) <- "kernel"
  return(rval)
}


# Simulating from the prior for ell = 0.2
kernel02 <- k(sigmaf = 1, ell = 0.2) # This constructs the covariance function
xGrid = seq(-1,1,by=0.1)
K = kernelMatrix(kernel = kernel02, xGrid, xGrid)

colors = list("black","red","blue","green","purple")
f = rmvnorm(n = 1, mean = rep(0,length(xGrid)), sigma = K)
plot(xGrid,f, type = "l", ylim = c(-3,3), col = colors[[1]])
for (i in 1:4){
  f = rmvnorm(n = 1, mean = rep(0,length(xGrid)), sigma = K)
  lines(xGrid,f, col = colors[[i+1]])
}

# Simulating from the prior for ell = 1
kernel1 <- k(sigmaf = 1, ell = 1) # This constructs the covariance function
xGrid = seq(-1,1,by=0.1)
K = kernelMatrix(kernel = kernel1, xGrid, xGrid)

colors = list("black","red","blue","green","purple")
f = rmvnorm(n = 1, mean = rep(0,length(xGrid)), sigma = K)
plot(xGrid,f, type = "l", ylim = c(-3,3), col = colors[[1]])
for (i in 1:4){
  f = rmvnorm(n = 1, mean = rep(0,length(xGrid)), sigma = K)
  lines(xGrid,f, col = colors[[i+1]])
}


# Computing the correlation functions

# ell = 0.2
kernel02(0,0.1) # Note: here correlation=covariance since sigmaf = 1
kernel02(0,0.5)

# ell = 1
kernel1(0,0.1) # Note: here correlation=covariance since sigmaf = 1
kernel1(0,0.5)

# The correlation between the function at x = 0 and x=0.1 is much higher than
# between x = 0 and x=0.5, since the latter points are more distant.
# Thus, the correlation decays with distance in x-space. This decay is much more
# rapid when ell = 0.2 than when ell = 1. This is also visible in the simulations
# where realized functions are much more smooth when ell = 1.


### Question 3(b)
```

```r
load("GPdata.RData")

### ell = 0.2

sigmaNoise = 0.2

# Set up the kernel function
kernelFunc <- k(sigmaf = 1, ell = 0.2)

# Plot the data and the true
plot(x, y, main = "", cex = 0.5)

GPfit <- gausspr(x, y, kernel = kernelFunc, var = sigmaNoise^2)
# Alternative: GPfit <- gausspr(y ~ x, kernel = k, kpar = list(sigmaf = 1, ell = 0.2), var = sigmaNoise
xs = seq(min(x),max(x), length.out = 100)
meanPred <- predict(GPfit, data.frame(x = xs)) # Predicting the training data. To plot the fit.
lines(xs, meanPred, col="blue", lwd = 2)

# Compute the covariance matrix Cov(f)
n <- length(x)
Kss <- kernelMatrix(kernel = kernelFunc, x = xs, y = xs)
Kxx <- kernelMatrix(kernel = kernelFunc, x = x, y = x)
Kxs <- kernelMatrix(kernel = kernelFunc, x = x, y = xs)
Covf = Kss-t(Kxs)%*%solve(Kxx + sigmaNoise^2*diag(n), Kxs)

# Probability intervals for f
lines(xs, meanPred - 1.96*sqrt(diag(Covf)), col = "red")
lines(xs, meanPred + 1.96*sqrt(diag(Covf)), col = "red")

# Prediction intervals for y
lines(xs, meanPred - 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")
lines(xs, meanPred + 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")


legend("topright", inset = 0.02, legend = c("data","post mean","95% intervals for f", "95% predictive i
       col = c("black", "blue", "red", "purple"),
       pch = c('o',NA,NA,NA), lty = c(NA,1,1,1), lwd = 2, cex = 0.55)



### ell = 1

sigmaNoise = 0.2

# Set up the kernel function
kernelFunc <- k(sigmaf = 1, ell = 1)

# Plot the data and the true
plot(x,y, main = "", cex = 0.5)
#lines(xGrid,fVals, type = "l", col = "black", lwd = 3) # true mean

GPfit <- gausspr(x, y, kernel = kernelFunc, var = sigmaNoise^2)
xs = seq(min(x), max(x), length.out = 100)
```

```r
meanPred <- predict(GPfit, data.frame(x = xs)) # Predicting the training data. To plot the fit.
lines(xs, meanPred, col="blue", lwd = 2)

# Compute the covariance matrix Cov(f)
n <- length(x)
Kss <- kernelMatrix(kernel = kernelFunc, x = xs, y = xs)
Kxx <- kernelMatrix(kernel = kernelFunc, x = x, y = x)
Kxs <- kernelMatrix(kernel = kernelFunc, x = x, y = xs)
Covf = Kss-t(Kxs)%*%solve(Kxx + sigmaNoise^2*diag(n), Kxs)

# Probability intervals for f
lines(xs, meanPred - 1.96*sqrt(diag(Covf)), col = "red")
lines(xs, meanPred + 1.96*sqrt(diag(Covf)), col = "red")

# Prediction intervals for y
lines(xs, meanPred - 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")
lines(xs, meanPred + 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")


legend("topright", inset = 0.02, legend = c("data","post mean","95% intervals for f", "95% predictive i
       col = c("black", "blue", "red", "purple"),
       pch = c('o',NA,NA,NA), lty = c(NA,1,1,1), lwd = 2, cex = 0.55)


# Question: Explain the difference between the results from ii) and iii).
# Answer: ii) is about the uncertainty of the function f, which is the MEAN of y
#         iii) is about the uncertainty of individual y values. They are uncertain for
#              two reasons: you don't know f at the test point, and you don't know
#              the error (epsilon) that will hit this individual observation

# Question: Discuss the differences in results from using the two length scales.
#           Answer: shorter length scale gives less smooth f. We are overfitting the data.
#           Answer: longer length scale gives more smoothness.

# Question: Do you think a GP with a squared exponential kernel is a good model for this data? If not,
#           Answer: One would have to experiment with other length scales, or estimate
#           the length scales (see question 3c), but this is not likely to help here.
#           The issue is that the data seems to be have different smoothness for small x
#           than it has for large x (where the function seems much more flat)
#           The solution is probably to have different length scales for different x


### Question 3(c)

# For full points here you should mention EITHER of the following two approaches:
# 1. The marginal likelihood can be used to select optimal hyperparameters,
# and also the noise variance. We can optimize the log marginal likelihood with
# respect to the hyperparameters. In Gaussian Process Regression the marginal likelihood
# is availble in closed form (a formula).
# 2. We can use sampling methods (e.g. MCMC) to sample from the marginal posterior of the hyperparamete
# We need a prior p(theta) for the hyperparameter and then Bayes rule gives the marginal posterior
# p(theta | data) propto p(data | theta)*p(theta)
# where p(data | theta) is the marginal likelihood (f has been integrated out).
```

```r
# If the noise variance is unknown, we can treat like any of the kernel hyperparameters and infer the n
# jointly with the length scale and the prior variance sigma_f

# SSMs

set.seed(12345)
start_time <- Sys.time()

T<-100
mu_0<-50
Sigma_0<-10
R<-1
Q<-5

x<-vector(length=T)
z<-vector(length=T)
err<-vector(length=T)

for(t in 1:T){
  x[t]<-ifelse(t==1,rnorm(1,mu_0,Sigma_0),x[t-1]+1+rnorm(1,0,R))
  z[t]<-x[t]+rnorm(1,0,Q)
}

mu<-mu_0
Sigma<-Sigma_0*Sigma_0 # KF uses covariances
for(t in 2:T){
  pre_mu<-mu+1
  pre_Sigma<-Sigma+R*R # KF uses covariances
  K<-pre_Sigma/(pre_Sigma+Q*Q) # KF uses covariances
  mu<-pre_mu+K*(z[t]-pre_mu)
  Sigma<-(1-K)*pre_Sigma

  err[t]<-abs(x[t]-mu)

  cat("t: ",t,", x_t: ",x[t],", E[x_t]: ",mu," , error: ",err[t],"\n")
  flush.console()
}

mean(err[2:T])
sd(err[2:T])

end_time <- Sys.time()
end_time - start_time
# Question 3 (SSMs)

T<-100
mu_0<-50
Sigma_0<-sqrt(100^2/12)
n_par<-100
tra_sd<-.1
emi_sd<-1

ini_dis<-function(n){ # Sampling the initial model
```

```
  return (runif(n,min=0,max=100))
}

tra_dis<-function(zt){ # Sampling the transition model
  pi=sample(c(0,1,2),1)
  return (rnorm(1,mean=zt+pi,sd=tra_sd))
}

emi_dis<-function(zt){ # Sampling the emision model
  pi=sample(c(-1,0,1),1)
  return (rnorm(1,mean=zt+pi,sd=emi_sd))
}

den_emi_dis<-function(xt,zt){ # Density of the emission model
  return (((dnorm(xt,mean=zt,sd=emi_sd)
            +dnorm(xt,mean=zt-1,sd=emi_sd)
            +dnorm(xt,mean=zt+1,sd=emi_sd))/3))
}

z<-vector(length=T) # Hidden states
x<-vector(length=T) # Observations

for(t in 1:T){ # Sampling the SSM
  z[t]<-ifelse(t==1,ini_dis(1),tra_dis(z[t-1]))
  x[t]<-emi_dis(z[t])
}

plot(z,col="red",main="True location (red) and observed location (black)")
points(x)

# Particle filter starts

bel<-ini_dis(n_par) # Initial particles
err<-vector(length=T) # Error between expected and true locations
w<-vector(length=n_par) # Importance weights

cat("Initial error: ",abs(z[1]-mean(bel)),"\n")

for(t in 1:T){
  for(i in 1:n_par){
    w[i]<-den_emi_dis(x[t],bel[i])
  }
  w<-w/sum(w)

  Ezt<-sum(w * bel) # Expected location
  err[t]<-abs(z[t]-Ezt) # Error between expected and true locations

  com<-sample(1:n_par,n_par,replace=TRUE,prob=w) # Sampling new particles according to the importance w
  bel<-sapply(bel[com],tra_dis) # Project
}

mean(err[(T-10):T])
sd(err[(T-10):T])
```

```r
# Kalman filter starts

R<-tra_sd
Q<-emi_sd

err<-vector(length=T)

mu<-mu_0
Sigma<-Sigma_0*Sigma_0 # KF uses covariances
for(t in 2:T){
  pre_mu<-mu+1
  pre_Sigma<-Sigma+R*R # KF uses covariances
  K<-pre_Sigma/(pre_Sigma+Q*Q) # KF uses covariances
  mu<-pre_mu+K*(x[t]-pre_mu)
  Sigma<-(1-K)*pre_Sigma

  err[t]<-abs(z[t]-mu)
}

mean(err[(T-10):T])
sd(err[(T-10):T])
```