LAB 3: STATE SPACE MODELS

Phillip Hölscher

3 10 2019

Contents

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 state space models (SSMs). The robot moves along the horizontal axis according to the following SSM:

$$
\begin{aligned}
p\left(z_t|z_{t-1}\right) &= \left(\mathcal{N}\left(z_t|z_{t-1},1\right)+\mathcal{N}\left(z_t|z_{t-1}+1,1\right)+\mathcal{N}\left(z_t|z_{t-1}+2,1\right)\right)/3 && \text{If Transition model}\\
p\left(x_i|z_t\right) &= \left(\mathcal{N}\left(x_i|z_t,1\right)+\mathcal{N}\left(x_i|z_t-1,1\right)+\mathcal{N}\left(x_i|z_t+1,1\right)\right)/3 && \text{If Emission model}\\
p\left(z_1\right) &= Unif_{\text{orm }(0,100)}
\end{aligned}
$$

1 Assignment

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.

Result assignemnt 1

- Simulation

This text shows the transition and emission model. Before we can simulate 100 time steps, we have to implement these models. Then I simulated the 100 time steps.

```
set.seed(12345)
############ implement SSM

#### simulate T =100
### x_1:100 & z_1:100

########## frist  step - before sampling
#### Transition model - z


transition_model = function(zt_zt1){
  # zt_zt1 = mu
  # sd = is always 1

  #sample from distribution 1,2 or 3

  sample_distribution = sample(1:3, size = 1)

  if(sample_distribution == 1){
    term = rnorm(n = 1, mean = zt_zt1, sd = 1)
  } else if(sample_distribution == 2){
    term = rnorm(n = 1, mean = zt_zt1 + 1, sd = 1)
  } else{
    term = rnorm(n = 1, mean = zt_zt1 +2, sd = 1)
  }

  return(term)
}
```

```
########## second step - before sampling
#### Emission model - x

emission_model = function(xt_zt, sd){
  # xt_zt = mu
  # sd = is always 1

  #sample from distribution 1,2 or 3

  sample_distribution = sample(1:3, size = 1)

  if(sample_distribution == 1){
    term = rnorm(n = 1, mean = xt_zt, sd = sd)
  } else if(sample_distribution == 2){
    term = rnorm(n = 1,mean = xt_zt - 1, sd = sd)
  } else{
    term = rnorm(n = 1,mean = xt_zt + 1, sd = sd)
  }

  return(term)
}



########## third step - sampling
#### Initial model - given above
# p(z1) = Uniform(0,100)

#### sample 100 time steps
sample_function = function(sample_nr = 100, sd = 1){

  # init vector for x & z
  BigT = sample_nr
  x_1_100 = c()
  z_1_100 = c()

  ### init values for timestep 1
  # z_1_100[1] = z1
  initial_model = runif(n = 1, min = 0, max = sample_nr)
  z1 = initial_model

  for (t in 1:BigT) {

      # sample z vector
      if (t == 1){
        z_1_100[t] = z1
      } else{
        z_1_100[t] = transition_model(z_1_100[t-1])
      }

      # sample x vector
      x_1_100[t] = emission_model(z_1_100[t], sd)
  }
```

```
  # return z and x vector
  result = data.frame("z" = z_1_100,
                      "x" = x_1_100)

}
```

- Particle filter

Here I implemented the particle filter.

```
##############################
### implement particle filter

########## frist step
# compute the densitiy function of the emission model
# we need this for the weights
weights_density = function(x, z, sd) {
  # input: sd - in question 2 we have to change the sd, so it needs to be flexible

  # init vector to save results
  weight = c()
  M = length(x)

  for(m in 1:M) {
    # density for given emission model
    weight1 = dnorm(x, mean = z, sd = sd, log = FALSE)
    weight2 = dnorm(x, mean = z - 1, sd = sd, log = FALSE)
    weight3 = dnorm(x, mean = z + 1, sd = sd, log = FALSE)
    weight[m] = (weight1 + weight2 + weight3) / 3
  }
  return(weight)
}

########## second step
## Draws particles of given particles z with corresponding weights w
## Utilizes multinomial distrubtion draw
particle_draw = function(z_particles, weight) {

  # init vector to save results
  res = c()
  M = length(weight)

  draws = rmultinom(1, M, weight)
  for (i in 1:M) {
    if (draws[i]) {
      for (j in 1:draws[i]) {
        res = append(res, z_particles[i])
      }
    }
  }
  return(res)
}

########## third
###### particle filter algorithm
```

```
particle_filtering = function(sim, sd, equalWeight) {
  x = sim$x # extract observations

  T = 100
  M = 100
  # init bel(x_t) - Z & bel_bar(x_t) - Z_bar
  Z = matrix(0, nrow = M, ncol = T)
  Z_bar = matrix(0, nrow = M, ncol = T)

  for (t in 1:T) {
    if (t == 1) Z[,1] = runif(M, 0, 100)
    else {
      z_particles = c()
      weights = c()
      for (m in 1:M) {
        z_particles[m] = transition_model(Z[m,t-1])

        if (equalWeight) weights[m] = 1
        else weights[m] = weights_density(x[t], z_particles[m], sd)
      }
      Z_bar[,t] = z_particles
      Z[,t] = particle_draw(Z_bar[,t], weights)
    }
  }
  return(Z)
}
```

- Plot the results

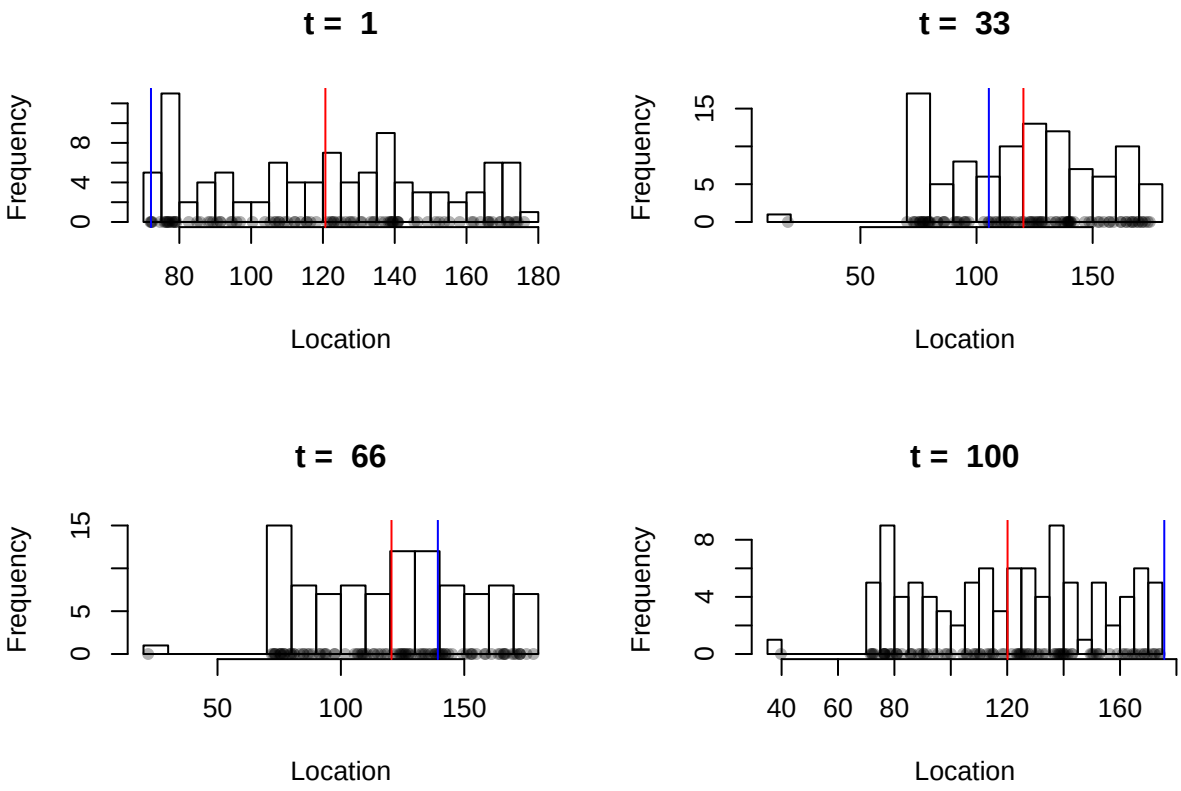
To visual the results I did juse a histogram, which represents the particles as well as the points of the x-axis. The red line represents the mean of the particles (expected location) and the blue line does represent the true location.

Plot for the time steps 1, 33, 66, 100

```
### sample data
set.seed(12345)
sample_data = sample_function(sd = 1)

### particle filter
set.seed(12345)
pf = particle_filtering(sim = sample_data, sd = 1, equalWeight = FALSE)

### plot
par(mfrow = c(2,2))
plot_function(pf, sample_data$z, 1)
plot_function(pf, sample_data$z, 33)
plot_function(pf, sample_data$z, 66)
plot_function(pf, sample_data$z, 100)
```



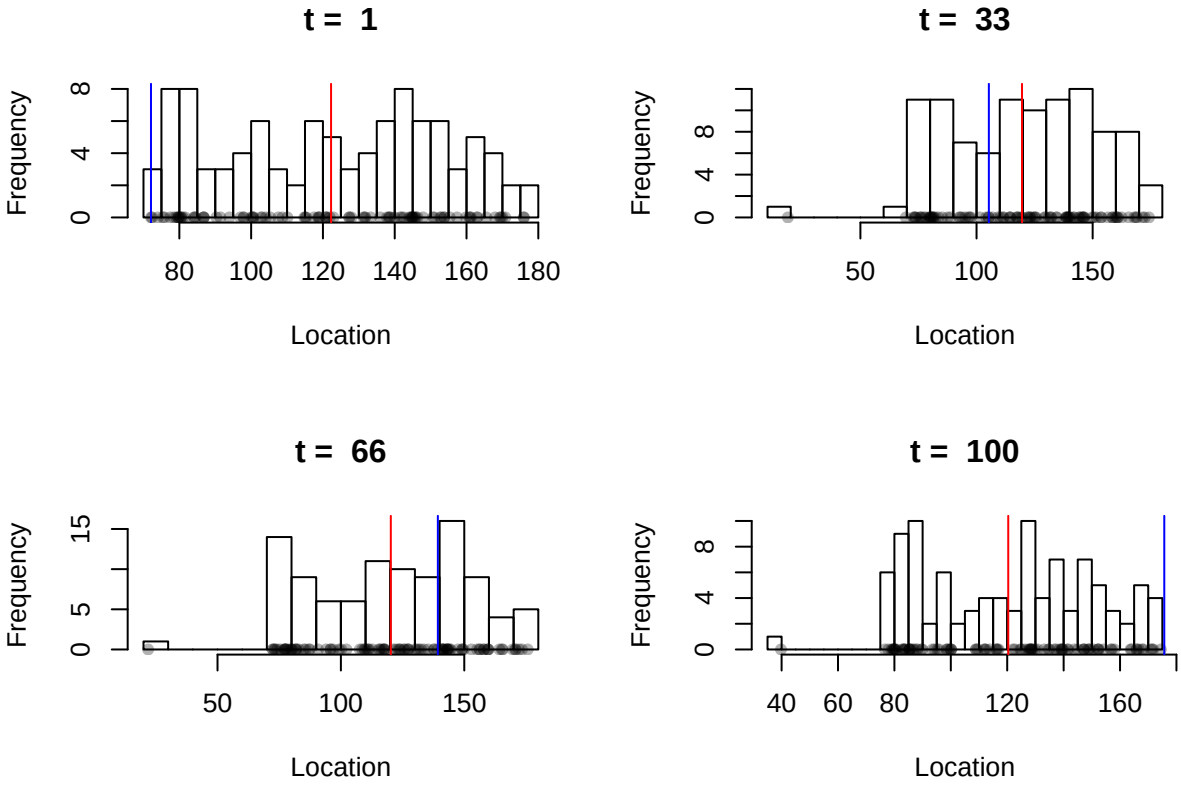
Here we can see that the result at t = 1 is very bad. It has a very large range and also the mean and true value are far apart. This improves with larger t.

2 Assignment

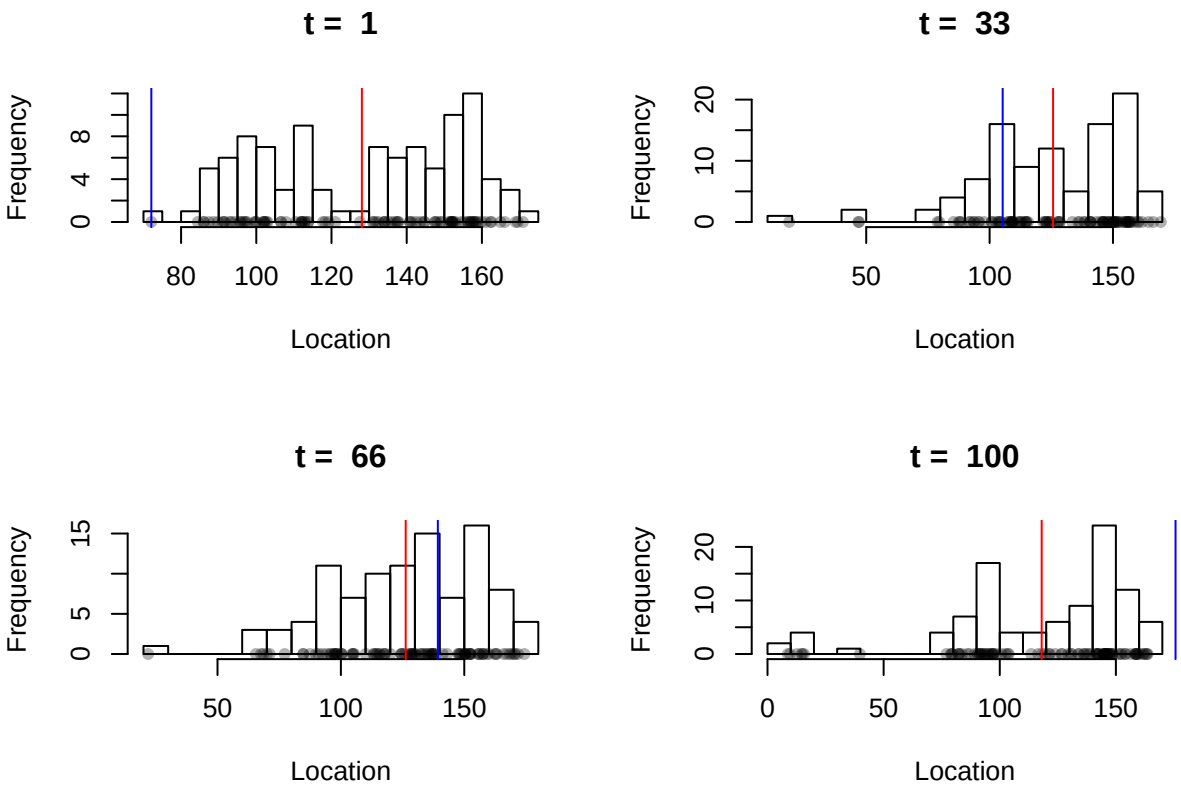
Repeat the exercise above replacing the standard deviation of the emission model with 5 and then with 50. Comment on how this affects the results.

Result assignemnt 2

- Plots sd = 5



- Plots sd = 50



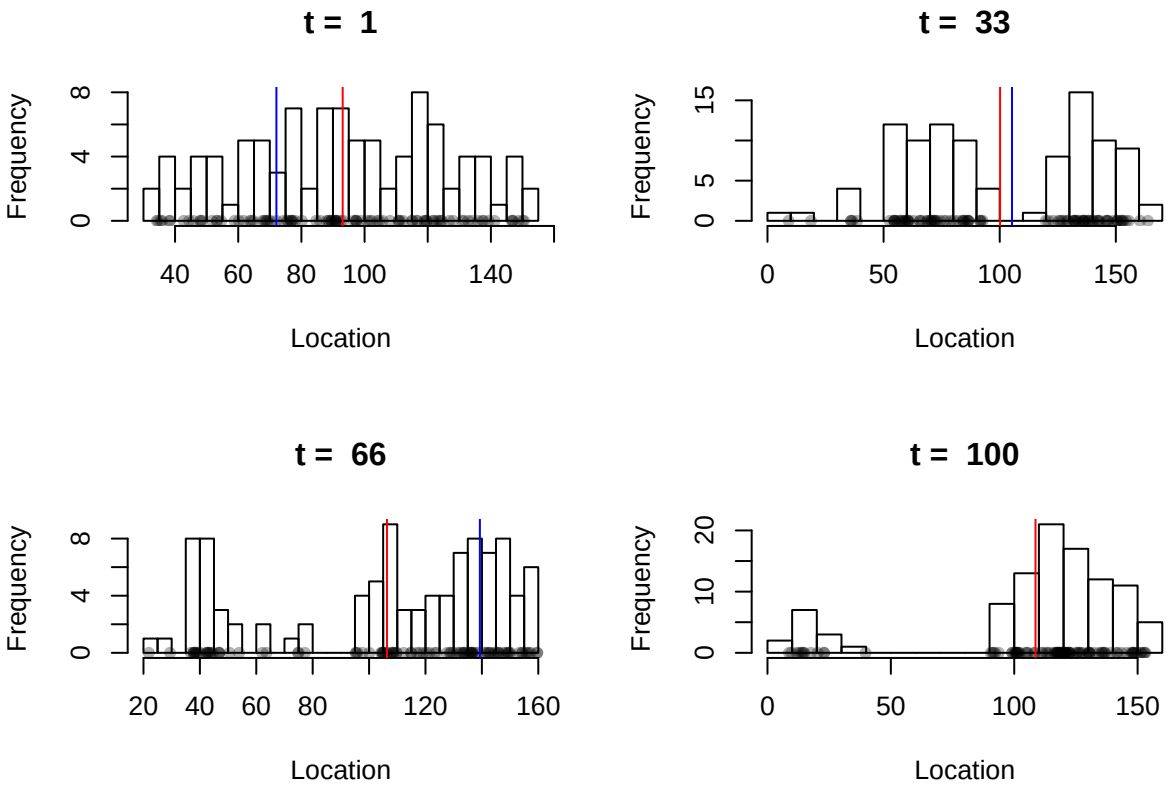
In general, the result should be worse than in the previous task because the variation of the observations will greatly increase.

In the first plot, however, we can see that changes have hardly taken place. However, if we increase the variance to 50, we see the negative effects. At t = 100 there is a large range and the distribution in the 0 range is high.

3 Assignment

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

Result assignemnt 3



If the weights for all particles are 1, which means all particles have the same affects to the estimations of states.We can see the result very well in the plots. In no task part is the prediction as differently distributed as in this one. The range of the location is very large.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
set.seed(12345)
############ implement SSM

#### simulate T =100
### x_1:100 & z_1:100

########## frist  step - before sampling
#### Transition model - z


transition_model = function(zt_zt1){
  # zt_zt1 = mu
  # sd = is always 1

  #sample from distribution 1,2 or 3

  sample_distribution = sample(1:3, size = 1)

  if(sample_distribution == 1){
    term = rnorm(n = 1, mean = zt_zt1, sd = 1)
  } else if(sample_distribution == 2){
    term = rnorm(n = 1, mean = zt_zt1 + 1, sd = 1)
  } else{
    term = rnorm(n = 1, mean = zt_zt1 +2, sd = 1)
  }

  return(term)
}

########## second step - before sampling
#### Emission model - x

emission_model = function(xt_zt, sd){
  # xt_zt = mu
  # sd = is always 1

  #sample from distribution 1,2 or 3

  sample_distribution = sample(1:3, size = 1)

  if(sample_distribution == 1){
    term = rnorm(n = 1, mean = xt_zt, sd = sd)
  } else if(sample_distribution == 2){
    term = rnorm(n = 1,mean = xt_zt - 1, sd = sd)
  } else{
    term = rnorm(n = 1,mean = xt_zt + 1, sd = sd)
  }

  return(term)
}
```

```
########## third step - sampling
#### Initial model - given above
# p(z1) = Uniform(0,100)

#### sample 100 time steps
sample_function = function(sample_nr = 100, sd = 1){

  # init vector for x & z
  BigT = sample_nr
  x_1_100 = c()
  z_1_100 = c()

  ### init values for timestep 1
  # z_1_100[1] = z1
  initial_model = runif(n = 1, min = 0, max = sample_nr)
  z1 = initial_model

  for (t in 1:BigT) {

      # sample z vector
      if (t == 1){
        z_1_100[t] = z1
      } else{
        z_1_100[t] = transition_model(z_1_100[t-1])
      }

      # sample x vector
      x_1_100[t] = emission_model(z_1_100[t], sd)
  }
  # return z and x vector
  result = data.frame("z" = z_1_100,
                      "x" = x_1_100)

}

##############################
### implement particle filter

########## frist step
# compute the densitiy function of the emission model
# we need this for the weights
weights_density = function(x, z, sd) {
  # input: sd - in question 2 we have to change the sd, so it needs to be flexible

  # init vector to save results
  weight = c()
  M = length(x)

  for(m in 1:M) {
    # density for given emission model
    weight1 = dnorm(x, mean = z, sd = sd, log = FALSE)
```

```
    weight2 = dnorm(x, mean = z - 1, sd = sd, log = FALSE)
    weight3 = dnorm(x, mean = z + 1, sd = sd, log = FALSE)
    weight[m] = (weight1 + weight2 + weight3) / 3
  }
  return(weight)
}

########## second step
## Draws particles of given particles z with corresponding weights w
## Utilizes multinomial distrubtion draw
particle_draw = function(z_particles, weight) {

  # init vector to save results
  res = c()
  M = length(weight)

  draws = rmultinom(1, M, weight)
  for (i in 1:M) {
    if (draws[i]) {
      for (j in 1:draws[i]) {
        res = append(res, z_particles[i])
      }
    }
  }
  return(res)
}

########## third
###### particle filter algorithm
particle_filtering = function(sim, sd, equalWeight) {
  x = sim$x # extract observations

  T = 100
  M = 100
  # init bel(x_t) - Z & bel_bar(x_t) - Z_bar
  Z = matrix(0, nrow = M, ncol = T)
  Z_bar = matrix(0, nrow = M, ncol = T)

  for (t in 1:T) {
    if (t == 1) Z[,1] = runif(M, 0, 100)
    else {
      z_particles = c()
      weights = c()
      for (m in 1:M) {
        z_particles[m] = transition_model(Z[m,t-1])

        if (equalWeight) weights[m] = 1
        else weights[m] = weights_density(x[t], z_particles[m], sd)
      }
      Z_bar[,t] = z_particles
      Z[,t] = particle_draw(Z_bar[,t], weights)
    }
  }
```

```
  return(Z)
}

##### implement a plot function

# Show the particles, the expected location and the true location

plot_function = function(particles, true_location, t) {
  M = dim(particles)[2]

  # histogram of the particles
  hist(x = particles[t, ],
       breaks = 20,
       main = paste('t = ', t),
       xlab = 'Location')
  # represent particles also via points
  points(x = particles[t, ],
         y = rep(0, M),
         pch = 16,
         col = rgb(0, 0, 0, 0.3))

  # the expected (mean) and true location
  abline(v = mean(particles[t, ]), col = 'red')
  abline(v = true_location[t], col = 'blue')

}


### sample data
set.seed(12345)
sample_data = sample_function(sd = 1)

### particle filter
set.seed(12345)
pf = particle_filtering(sim = sample_data, sd = 1, equalWeight = FALSE)

### plot
par(mfrow = c(2,2))
plot_function(pf, sample_data$z, 1)
plot_function(pf, sample_data$z, 33)
plot_function(pf, sample_data$z, 66)
plot_function(pf, sample_data$z, 100)


### sample data
set.seed(12345)
sample_data = sample_function(sd = 5)

### particle filter
set.seed(12345)
pf = particle_filtering(sim = sample_data, sd = 5, equalWeight = FALSE)

### plot
```

```r
par(mfrow = c(2,2))
plot_function(pf, sample_data$z, 1)
plot_function(pf, sample_data$z, 33)
plot_function(pf, sample_data$z, 66)
plot_function(pf, sample_data$z, 100)


### sample data
set.seed(12345)
sample_data = sample_function(sd = 50)

### particle filter
set.seed(12345)
pf = particle_filtering(sim = sample_data, sd = 50, equalWeight = FALSE)

### plot
par(mfrow = c(2,2))
plot_function(pf, sample_data$z, 1)
plot_function(pf, sample_data$z, 33)
plot_function(pf, sample_data$z, 66)
plot_function(pf, sample_data$z, 100)


### sample data
set.seed(12345)
sample_data = sample_function(sd = 1)

### particle filter
set.seed(12345)
pf = particle_filtering(sim = sample_data, sd = 1, equalWeight = TRUE)

### plot
par(mfrow = c(2,2))
plot_function(pf, sample_data$z, 1)
plot_function(pf, sample_data$z, 33)
plot_function(pf, sample_data$z, 66)
plot_function(pf, sample_data$z, 100)
```