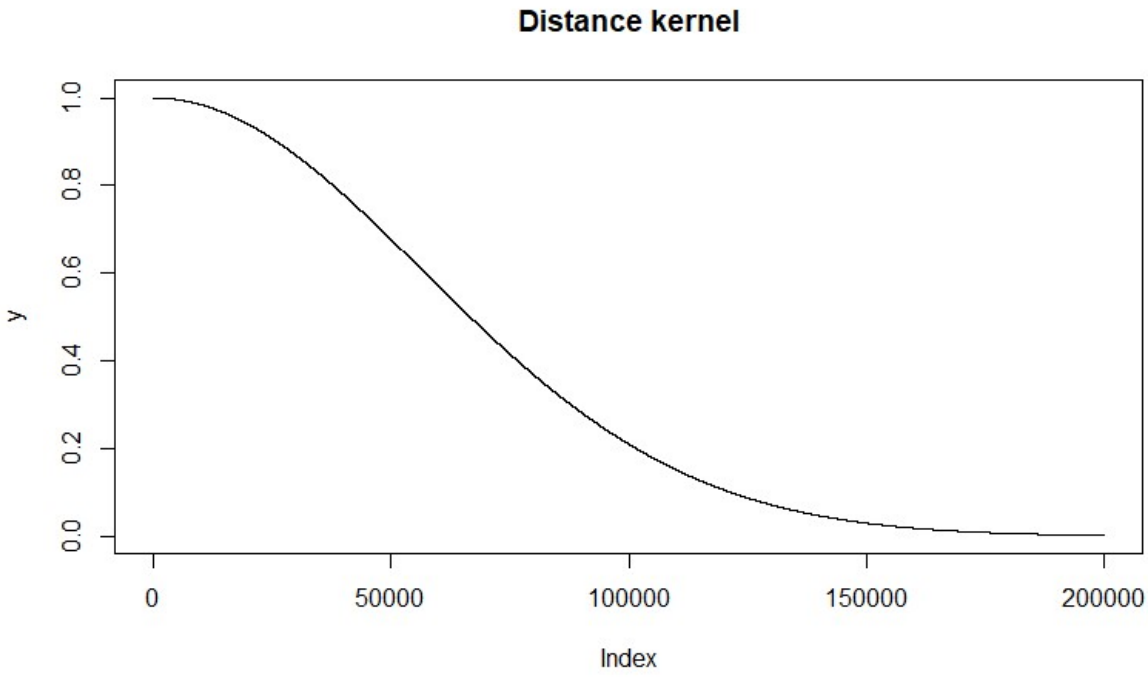
Assignment 3

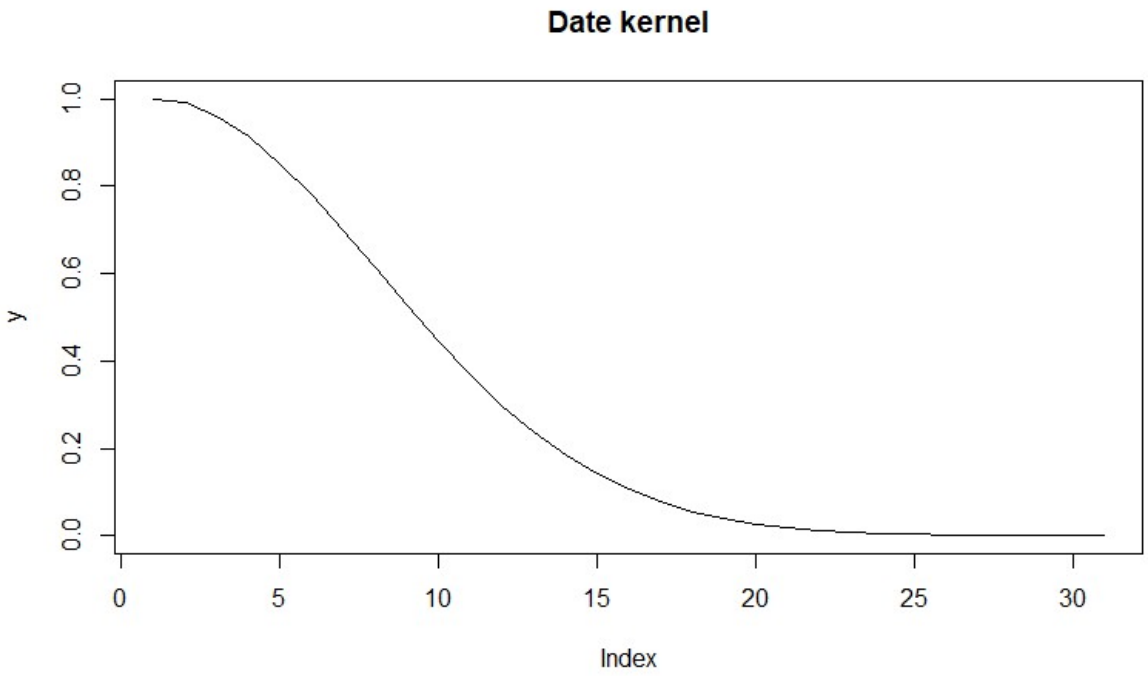
Assignment 1

Smoothing Coefficient

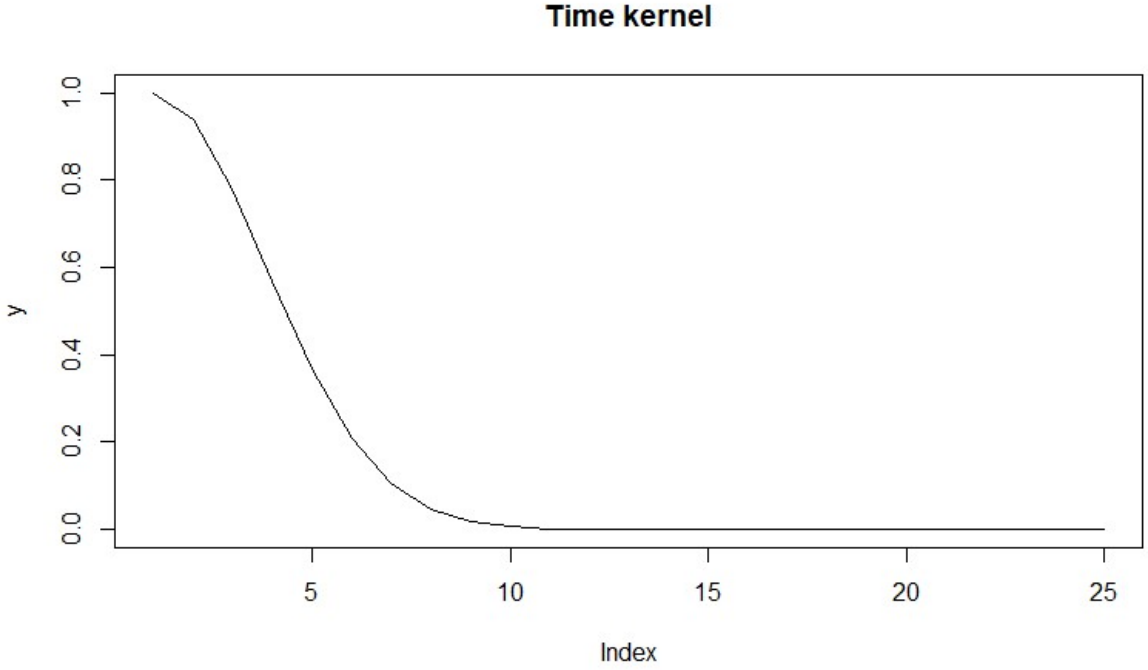
To choose an appropriate smoothing coefficient we look at the kernel for different values on difference in distance, date and time. By studying the graphs for different values on h the appropriate h was chosen for each kernel.



For the distance kernel a smoothing coefficient of 80000 was chosen to give closer points more weight. Where points further than 150km has a kernel weight close to zero.



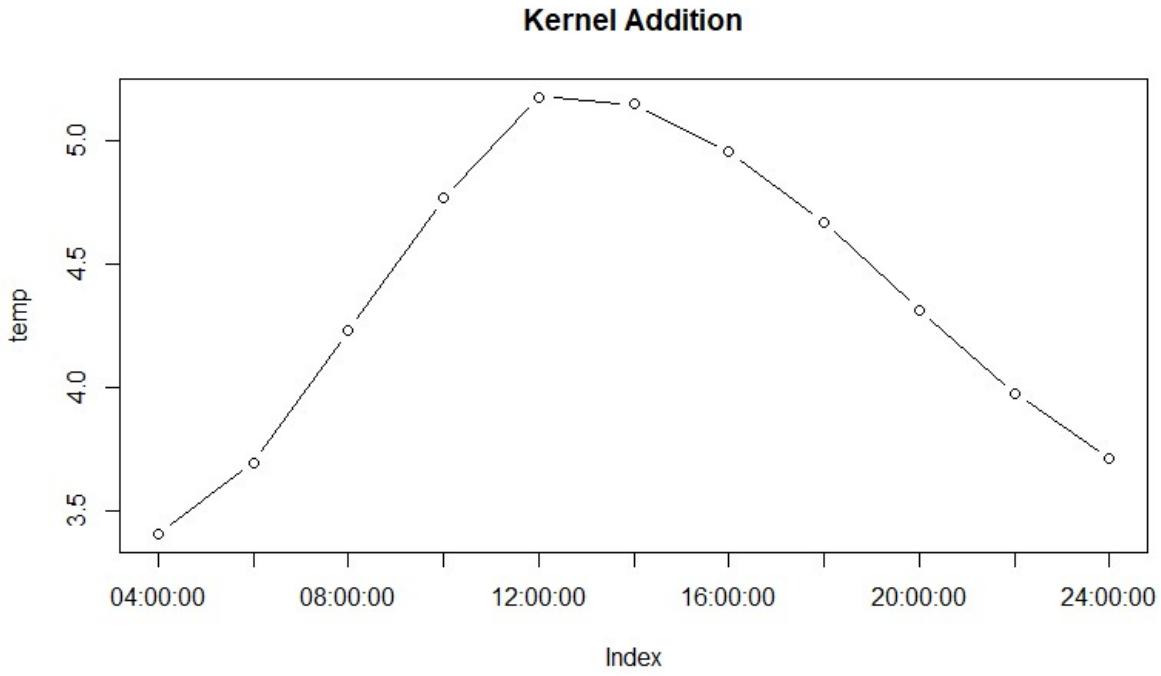
For the date kernel a smoothing coefficient of 10 was chosen to give closer points more weight. Where dates with more than 20 days in difference has a kernel weight close to zero. When we account for the number of days between our date of interest, we only look at the date, not the year. Therefore, we use mod 365.25 to get the number of days. If the number of days is larger than 180, we also adjust our number of days to the smallest distance.



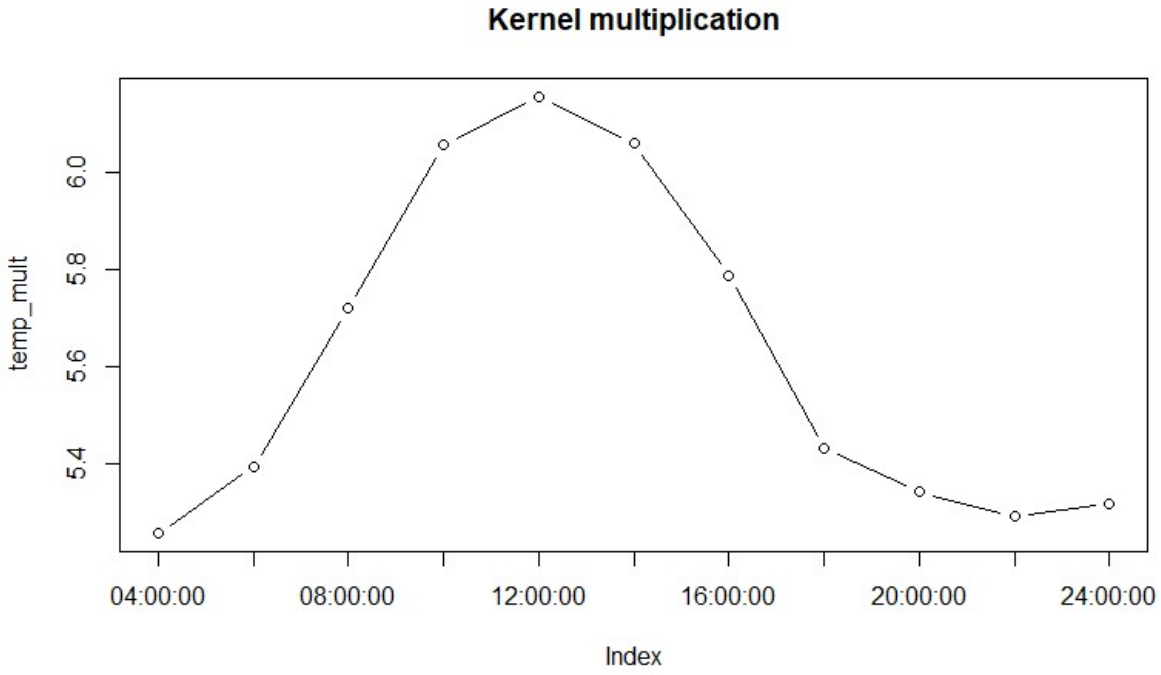
For the time kernel a smoothing coefficient of 4 was chosen to give closer points more weight. Where times with more than 7 hours in difference has a kernel weight close to zero. If the number of hours in difference is larger than 12, we adjust the number of hours to the smallest distance.

Temperature

Plot of temperature using addition between kernels:



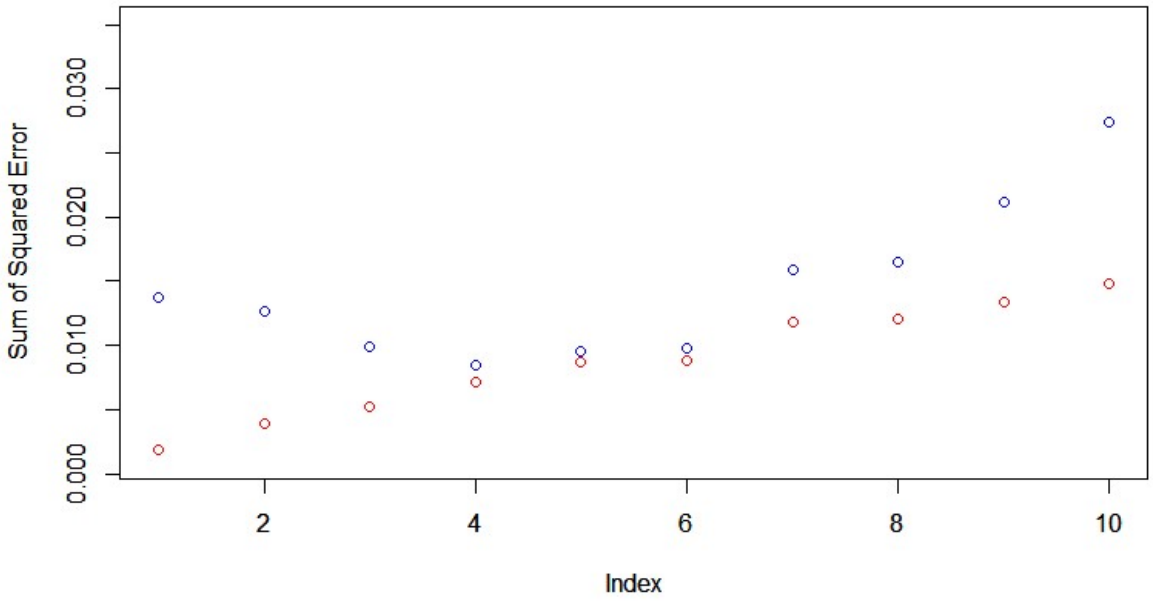
Plot of temperature using multiplication between kernels.



The two plots of temperature using the two different kernels is a bit different. In the first plot, where addition between kernels is used, a date and time kernel can have high weight while the distance between the measure and our point of interest is large, which gives us a low distance kernel. That combination will still give the measure a quite high kernel due to the characteristics of addition. While in the multiplication kernel that measure will have a smaller weight.

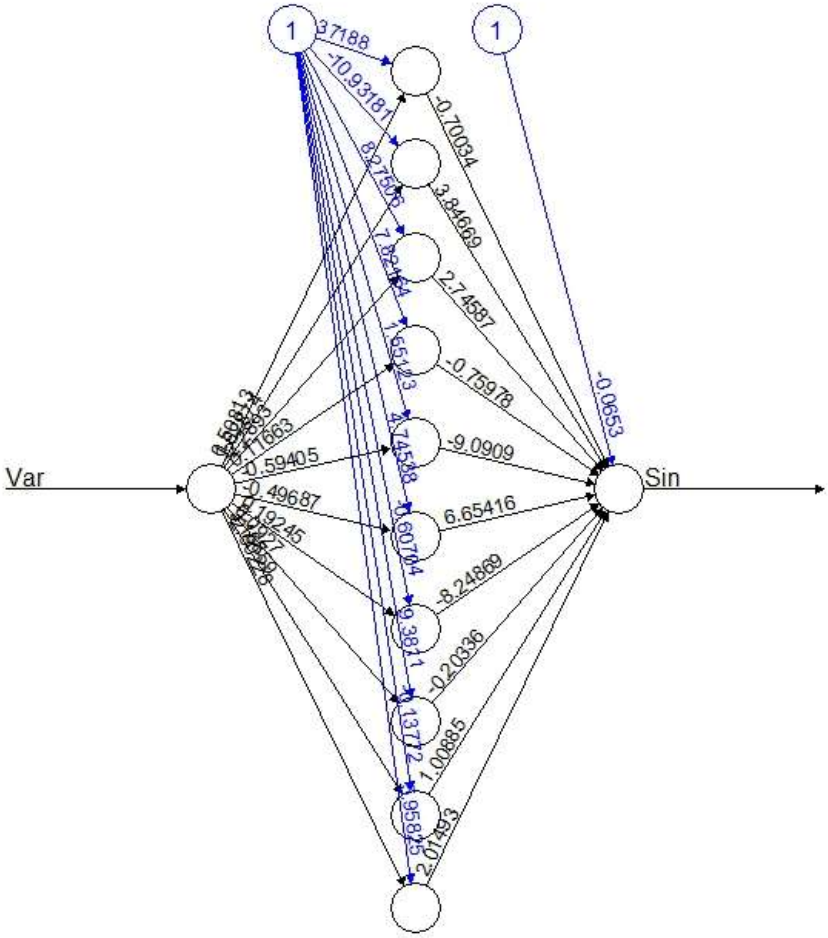
Assignment 3

To choose an appropriate threshold, the different neural networks were compared by using the sum of squared errors between predictions and real values for validation data. In the following graph blue dots are errors for validation data and red dots are errors for training data.

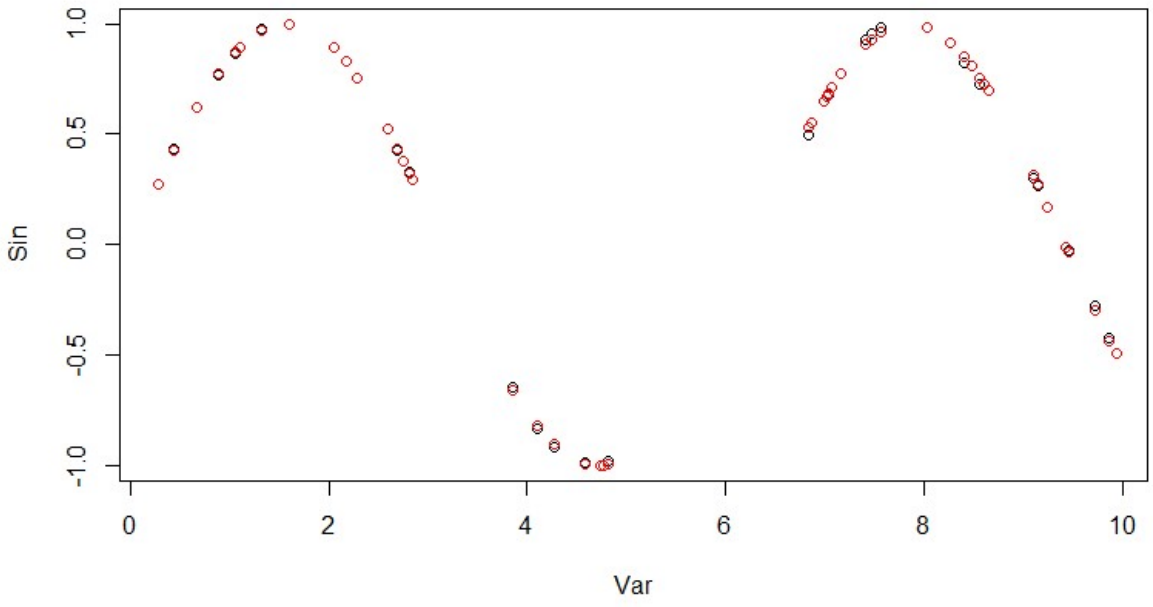


The smallest sum of square errors for validation data was found at a threshold = 4/1000.

Using threshold = 4/1000 the following Neural Network was provided:



In the following plot black dots are predictions form our Neural Network and red dots are real data, generated from sin funciton. Both datasets are following a sin curve.



Appendix

Assignment 1

```
RNGversion('3.5.1')

library(readr)

library(geosphere)


#---Assignment 1 ----

stations = read.csv("C:/Users/oskar/OneDrive/Universitet/Linköping Universitet/År4/Machine learning/Lab 3/stations.csv")

temps = read.csv("C:/Users/oskar/OneDrive/Universitet/Linköping Universitet/År4/Machine learning/Lab 3/temps50k.csv")


set.seed(1234567890)

st <- merge(stations,temps,by="station_number")



h_distance <- 80000 # These three values are up to the students

h_date <- 10

h_time <- 4

a <- 58.4274 # The point to predict (up to the students)

b <- 14.826

date <- "2013-11-04" # The date to predict (up to the students)

times <- c("04:00:00", "06:00:00", "08:00:00", "10:00:00", "12:00:00", "14:00:00", "16:00:00", "18:00:00", "20:00:00", "22:00:00", "24:00:00")

temp <- vector(length=length(times))

temp_mult <- vector(length=length(times))


# Students' code here

#test h values

max(distHaversine(data.frame(st$latitude, st$longitude), c(a,b)))

dist = seq(0,200000, 1)

y = exp(-(dist/h_distance)^2)
```

```
plot(y, type="l", main = "Distance kernel")
#satisfied with h_distance = 80000 beacause then we stop to care if distance>than200km
dat = seq(0,30, 1)
y = exp(-(dat/h_date)^2)
plot(y, type="l", main = "Date kernel")
#satisfied with h_date = 10 because then we stop to care if the date is older than 25 days.
tim = seq(0, 24, 1)
y = exp(-(tim/h_time)^2)
plot(y, type="l", main = "Time kernel")
#satisfied with h_time = 4 because then only times within 7 hours is used for estimate.

# remove all posterior dates
str(st)
st$date = as.Date(st$date, format = "%Y-%m-%d")
#remove earlier times than 04:00:00
filtered_data = st[st$date <= date, ]
filtered_data = filtered_data[!(filtered_data$date==date && substr(filtered_data$time, 1,
2)<substr(times[1], 1, 2))]

#Gussian kernel is used: k(u) = exp(-||u||^2),
#||.|| is the Euclidean norm.
euclidean = function(X){
  return (sqrt(sum(X^2)))
}

#gussian kernel
kernel_dist = function(X, X_n, h){
  distance = distHaversine(X, X_n)
  u = (distance)/h  # calculate u = X-Xn/h
  return(exp(-euclidean(u))) #calculate k
}
```

```
kernel_date = function(X, X_n, h){
  distance = as.numeric((X - X_n))%%365.25
  if (distance > 365/2) {
    distance = 365 - distance
  }
  u = distance / h
  return(exp(-euclidean(as.numeric(u))))
}

kernel_time = function(X, X_n, h){
  distance = as.numeric(X) - as.numeric(X_n)
  if (distance > 12){
    distance = 24-distance
  }
  u = distance/h
  return(exp(-euclidean(u)))
}



#calculate y with dist
#filtered_data = filtered_data[order(filtered_data$latitude, filtered_data$longitude),]

#kernel_weight = 0
n = nrow(filtered_data)
k=vector("numeric", length = n)
k_mult = vector("numeric", length = n)
k_loop=vector("numeric", length = n)
k_mult_loop = vector("numeric", length = n)

for(i in 1:n){
```

```
  k_dist = kernel_dist(c(filtered_data$longitude[i], filtered_data$latitude[i]), c(a,b), h_distance )
  k_date = kernel_date(filtered_data$date[i], as.Date(date), h_date)

  k[i]=k_date + k_dist
  k_mult[i] = k_date * k_dist
}
for(j in 1:(length(times))){
  for (i in 1:n) {
    k_time = kernel_time(substr(filtered_data$time[i], 1, 2), substr(times[j], 1, 2), h_time)
    k_loop[i] = k[i] + k_time
    k_mult_loop[i] = k_mult[i] * k_time

    temp[j] = temp[j] + k_loop[i]*filtered_data$air_temperature[i]
    temp_mult[j] = temp_mult[j] + k_mult_loop[i] * filtered_data$air_temperature[i]
  }
  temp[j] = temp[j] / sum(k_loop)
  temp_mult[j] = temp_mult[j] /sum(k_mult_loop)
}

#test_temp = test_temp/kernel_weight #kernel weighted temp.
#test_temp
plot(temp, xaxt = "n", type = "b", main = "Kernel Addition")
axis(1, at=1:length(df.times), labels=df.times)

plot(temp_mult, xaxt = "n", type = "b", main ="Kernel multiplication")
axis(1, at=1:length(df.times), labels=df.times)
```

Assignment 3

```
library(neuralnet)
set.seed(1234567890)
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation
# Random initialization of the weights in the interval [-1, 1]
#set.seed(12345). Did not use this becaus it was not used in the code skeleton.
winit <- runif(31, -1, 1)
n = 10
SE_tr = vector("numeric", length = n)
SE_va = vector("numeric", length = n)
for(i in 1:n) {
  nn <- neuralnet(Sin ~ Var, data=tr, hidden = c(10), startweights = winit, threshold = i/1000 )

  p_tr = predict(nn, newdata = tr)
  SE_tr[i] = sum((tr$Sin - p_tr)^2)
  p_va = predict(nn, newdata = va)
  SE_va[i] = sum((va$Sin - p_va)^2)
}

which.min(SE_va) # 4/1000 has the lowest error.

plot(SE_tr, col = "red", ylim = c(0.001, 0.035), ylab = "Sum of Squared Error")
par(new=TRUE)
plot(SE_va, col = "blue", ylim = c(0.001, 0.035), ylab = "Sum of Squared Error")

#4/1000has the lowest Squared Error. Therefore 4/1000 is shosen as threshold.
plot(nn <- neuralnet(Sin ~ Var, data=tr, hidden = c(10), startweights = winit, threshold = 4/1000))

# Plot of the predictions (black dots) and the data (red dots)
```

```
plot(prediction(nn)$rep1)

points(trva, col = "red")
```