Lab3

Christian von Koch

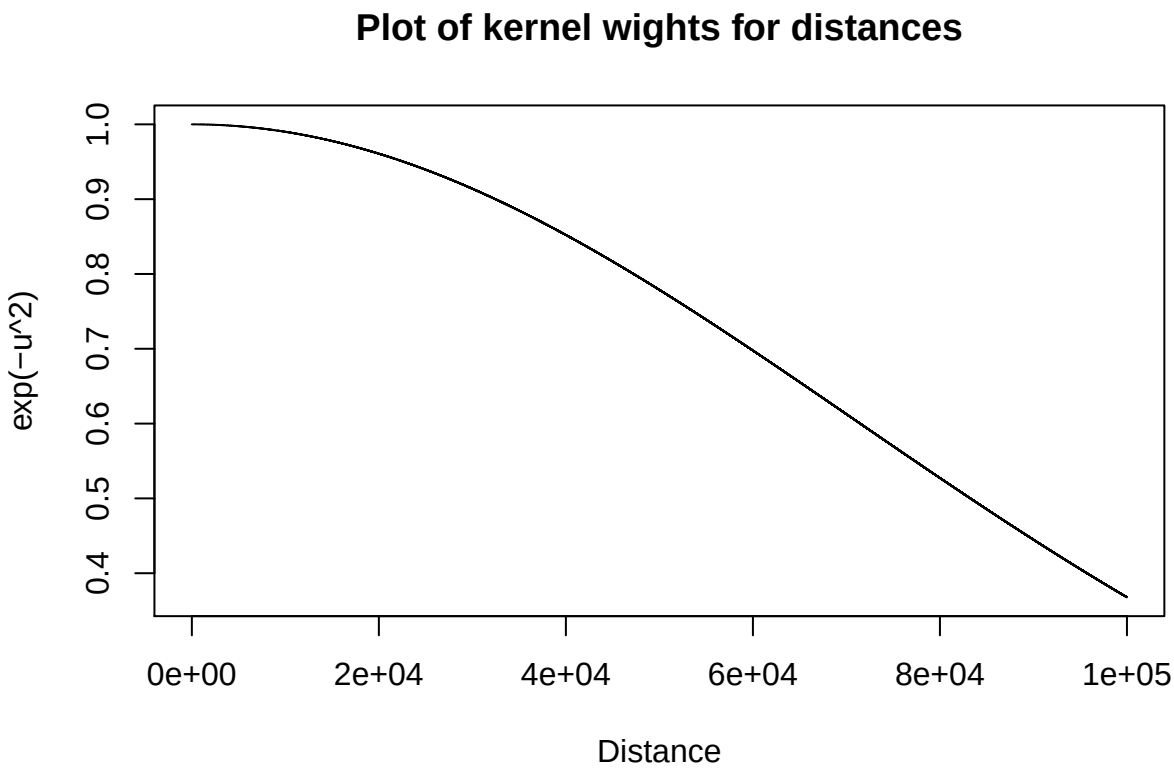
2019-12-17

Assignment 1

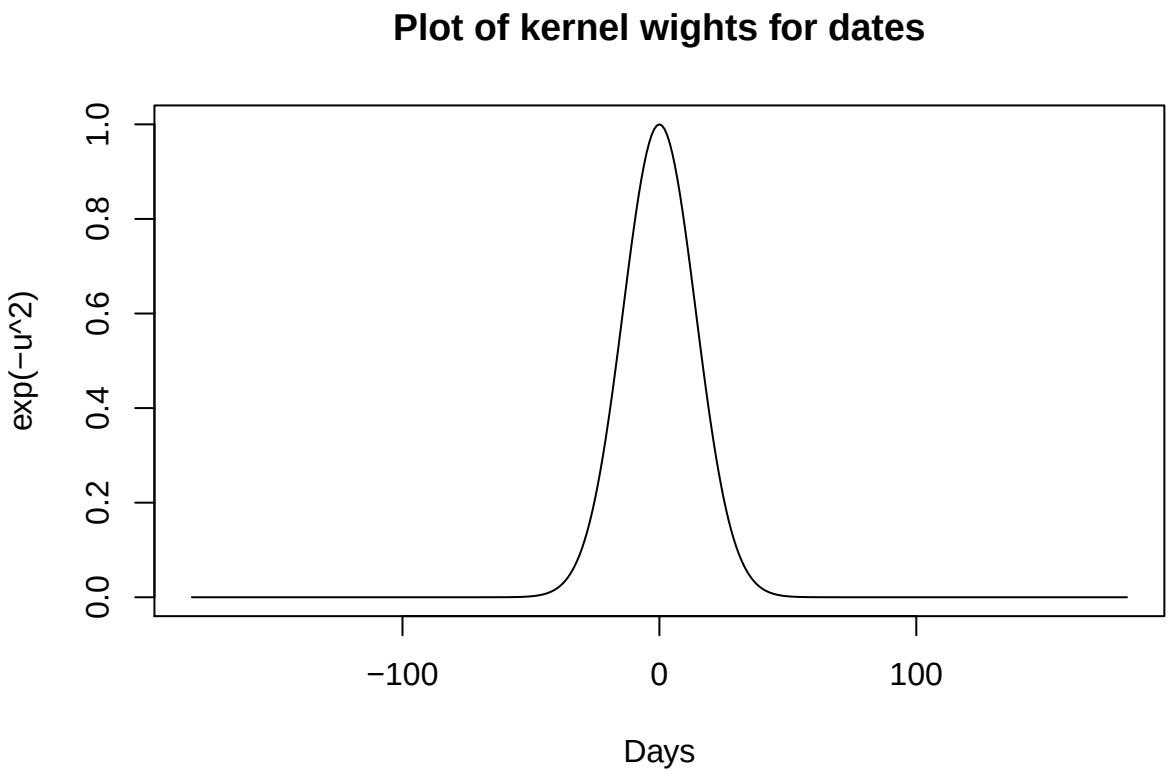
```
set.seed(1234567890)
#install.packages("geosphere")
library(geosphere)
stations <- read.csv("stations.csv")
temps <- read.csv("temps50k.csv")
#A join operation on "station_number"
st <- merge(stations,temps,by="station_number")
n = dim(st)[1]
#Kernel weighting factors
h_distance <- 100000
h_date <- 20
h_time <- 2
#Latitude of interest
a <- 59.4059
#Longitude of interest
b <- 18.0256
#Coordinates for Danderyd
#Create a vector of the point of interest
placeOI = c(a, b)
dateOI <- as.Date("1995-07-29") # The date to predict (up to the students), my birth date
timesOI = 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")
```

The code above reads the relevant data set and sets the initial values. The h values have been determined from the graphs shown below. The location of interest chosen is Danderyds kommun in Sweden, and the date of interest is the 29th of July 1995.

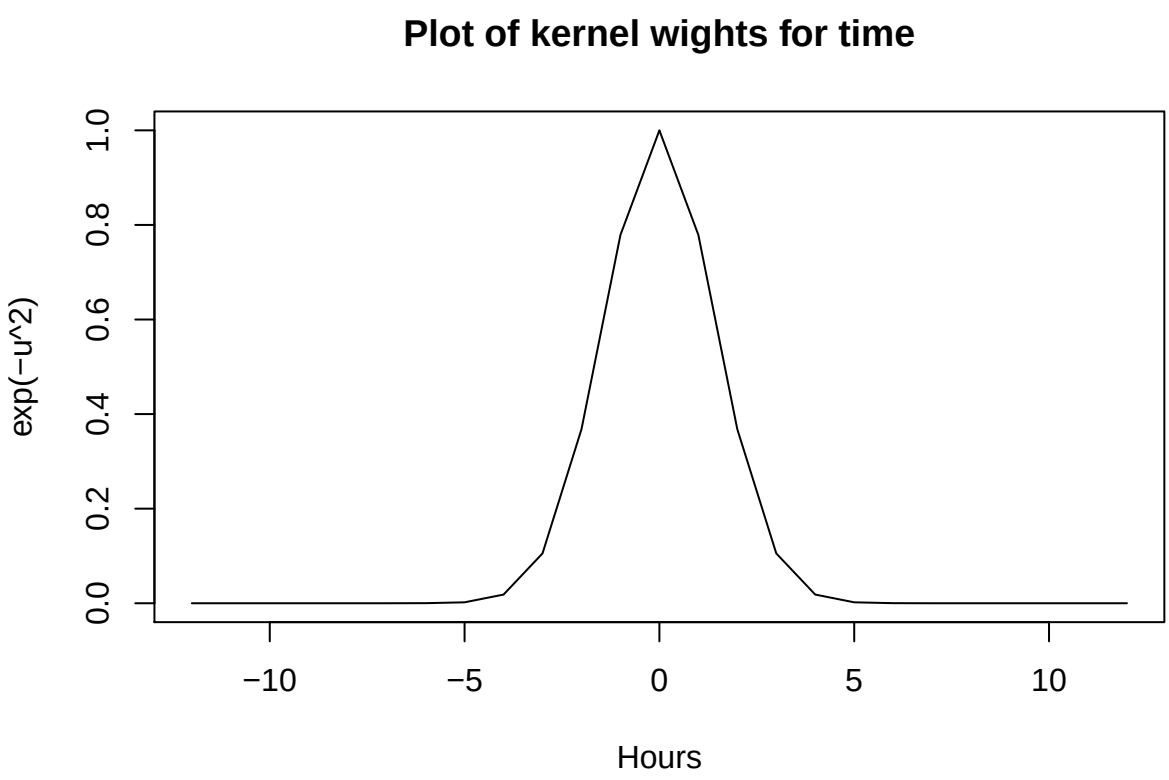
```
plotDist = function(dist, h){
  u = dist/h
  plot(dist, exp(-u^2), type="l", main="Plot of kernel wights for distances",
       xlab="Distance")
}

dist = seq(0, 100000, 1)
plotDist(dist, h_distance)
```




```
plotDate = function(date, h){
  u = date/h
  plot(date, exp(-u^2), type="l", main="Plot of kernel wights for dates", xlab="Days")
}

date = seq(-182,182,1)
plotDate(date, h_date)
```




```
plotTime = function(time, h){
  u = time/h
  plot(time, exp(-u^2), type="l", main="Plot of kernel wights for time", xlab="Hours")
}

time = seq(-12,12,1)
plotTime(time, h_time)
```



When studying the graphs above further, the h values can be motivated. For distance, it is noteable that the kernel does not give weight to measurements with a distance of 100 000 metres from the location of interest. It gives quite a little weight to distances which are around 50 000 metres from the location of interest. This seems reasonable since not a lot of data points are at the exact place of the location of interest and therefore the function needs to account for measurements which is further away to be able to draw a conclusion regarding the temperature. When the difference in distance is further than 100 000 metres it also seems reasonable to not account for that measurement when predicting the temperature. Elaborating further regarding the date, it again seems reasonable to apply an h value which does not give weight to a date difference which is more than around 30 days. Since the time of year (especially in Sweden) is changing quite fast the function should not account for temperature measurements at dates more than around a month from the date of interest. Similarily with time, the function should not account for time differences of more than 5 hours since the temperature can change significantly during this time.

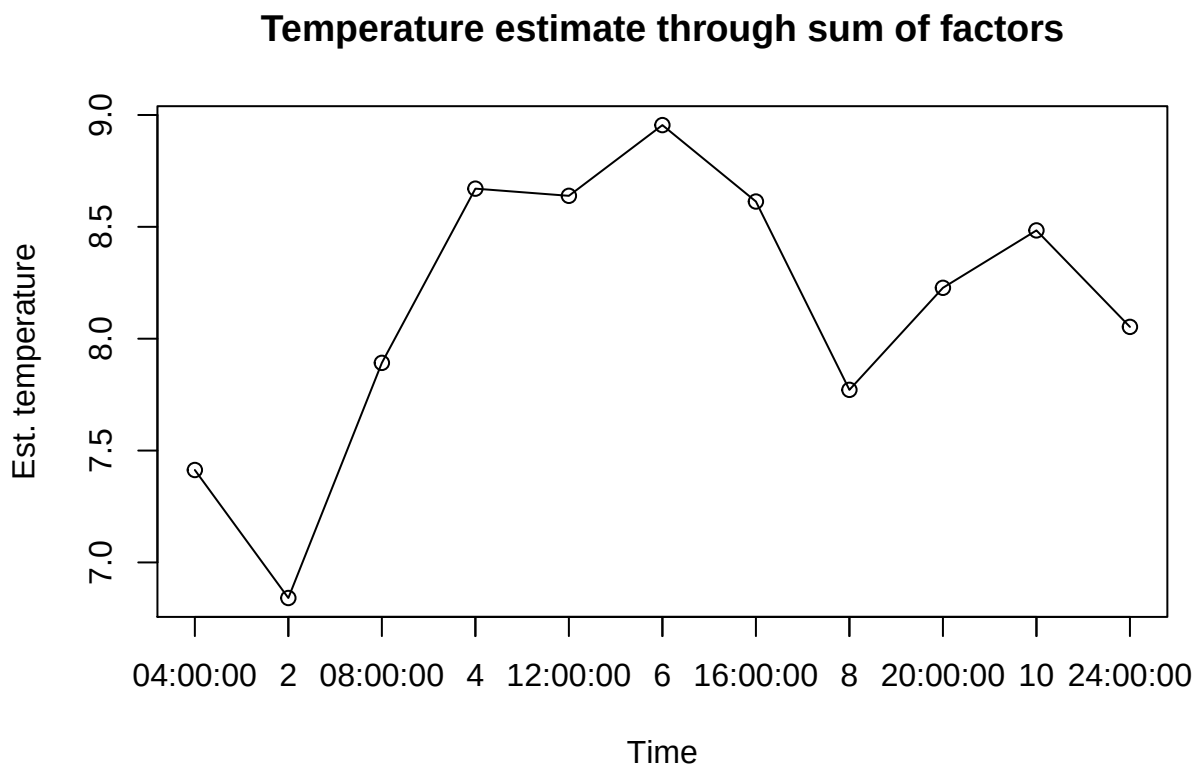
```
#Remove posterior data
filter_posterior = function(date, time, data){
  return(data[which(as.numeric(difftime(strptime(paste(date, time, sep=" "),
                                                 format="%Y-%m-%d %H:%M:%S"),
                                        strptime(paste(data$date, data$time, sep=" "),
                                                 format="%Y-%m-%d %H:%M:%S")))>0), ])
}

#A gaussian function for the difference in distance
gaussian_dist = function(place, data, h) {
  lat = data$latitude
  long = data$longitude
  points = data.frame(lat,long)
  u = distHaversine(points, place)/h
```

```
  return (exp(-u^2))
}

xy = gaussian_dist(placeOI, st, h_distance)

#A gaussian function for difference in days
gaussian_day = function(date, data, h){
  compare_date = as.Date(data$date)
  diff = as.numeric(date-compare_date)
  for (i in 1:length(diff)) {
    if (diff[i] > 365) {
      diff[i] = diff[i] %% 365
      if(diff[i]>182){
        diff[i]=365-diff[i]
      }
    }
  }
  u = diff/h
  return (exp(-u^2))
}

#A gaussian function for difference in hours
gaussian_hour = function(hour, data, h){
  compare_hour = strptime(data$time, format="%H:%M:%S")
  compare_hour = as.numeric(format(compare_hour, format="%H"))
  hour = strptime(hour, format="%H:%M:%S")
  hour = as.numeric(format(hour, format="%H"))
  diff = abs(hour-compare_hour)
  for (i in 1:length(diff)){
    if(diff[i]>12){
      diff[i] = 24-diff[i]
    }
  }
  u=diff/h
  return(exp(-u^2))
}
```

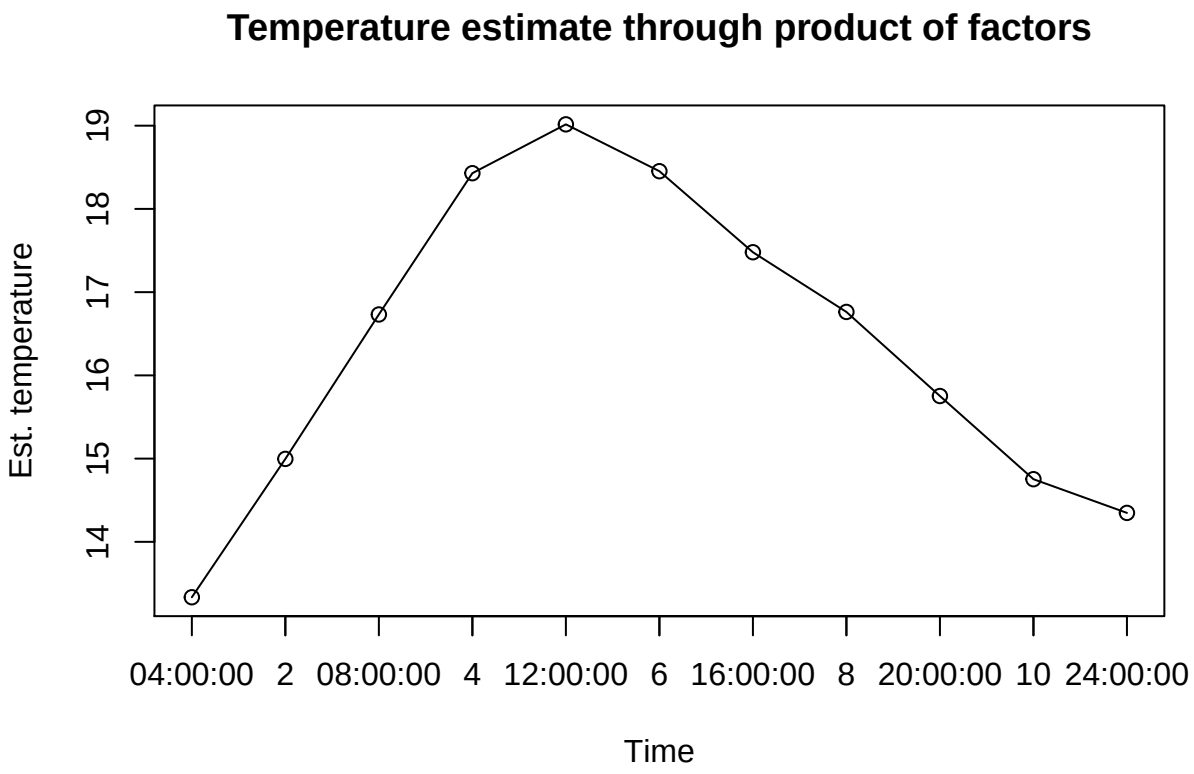
The functions above are used to filter out relevant data points that will be used for the temperature estimation of a specific date and time as well as computing the different kernel functions for differences of distance, date and time between the values in the data and the location and time of interest.

```
#Defining values that will be used in loop below
kernel_sum = c()
kernel_mult = c()

#Looping through time array and data points in nested loop to calculate the 11 kernel
#values
for (time in timesOI) {
  filtered_data = filter_posterior(dateOI, time, st)
  kernel_dist = gaussian_dist(placeOI, filtered_data, h_distance)
  kernel_day = gaussian_day(dateOI, filtered_data, h_date)
  kernel_time = gaussian_hour(time, filtered_data, h_time)
  sum_kernel = kernel_dist+kernel_day+kernel_time
  temp_sum = sum(sum_kernel * filtered_data$air_temperature)/sum(sum_kernel)
```

```
  mult_kernel = kernel_dist*kernel_day*kernel_time
  temp_mult = sum(mult_kernel * filtered_data$air_temperature)/sum(mult_kernel)
  kernel_sum = c(kernel_sum, temp_sum)
  kernel_mult = c(kernel_mult, temp_mult)
}


plot(kernel_sum, type="o", main ="Temperature estimate through sum of factors",
     xlab="Time", ylab="Est. temperature")
axis(1, at=1:length(timesOI), labels=timesOI)
```




```
plot(kernel_mult, type="o", main="Temperature estimate through product of factors",
     xlab="Time", ylab="Est. temperature")
axis(1, at=1:length(timesOI), labels=(timesOI))
```



Finally, the estimations for the temperatures are made through the summation of the different kernel functions as well as multiplication of the differnt kernel functions. It is noteable that the curves for the summation of kernel functions differs from the multiplication of kernel functions. The summation of kernel functions provides estimatet closer to the mean of all temperatures (approx. 4,62) than what the multiplication of kernel functions has provided. This can be due to that data points which have received a high weight through the kernel functions will have more impact in the multiplication of kernel functions than with the summation of kernel functions, and similarily data points which have received a low weight through the kernel functions will have more impact in the multiplication of kernel functions than with the summation of kernel functions. To conclude, the three different weights in the multiplication of kernel all has to be quite high in order for the total weight to be high. On the other hand if one weight is low the whole weight is going to be low even though the other two are high. The result of this is that the data points with high weight is more significant and perhaps more similar to the point of interest and time of interest for the multiplication of kernels than for the summation of kernels. This can also be seen in the graphs where the temperatures for the multiplication of kernels seem more reasonable intuitively than for the summation of kernels and seem like a more accurate model.

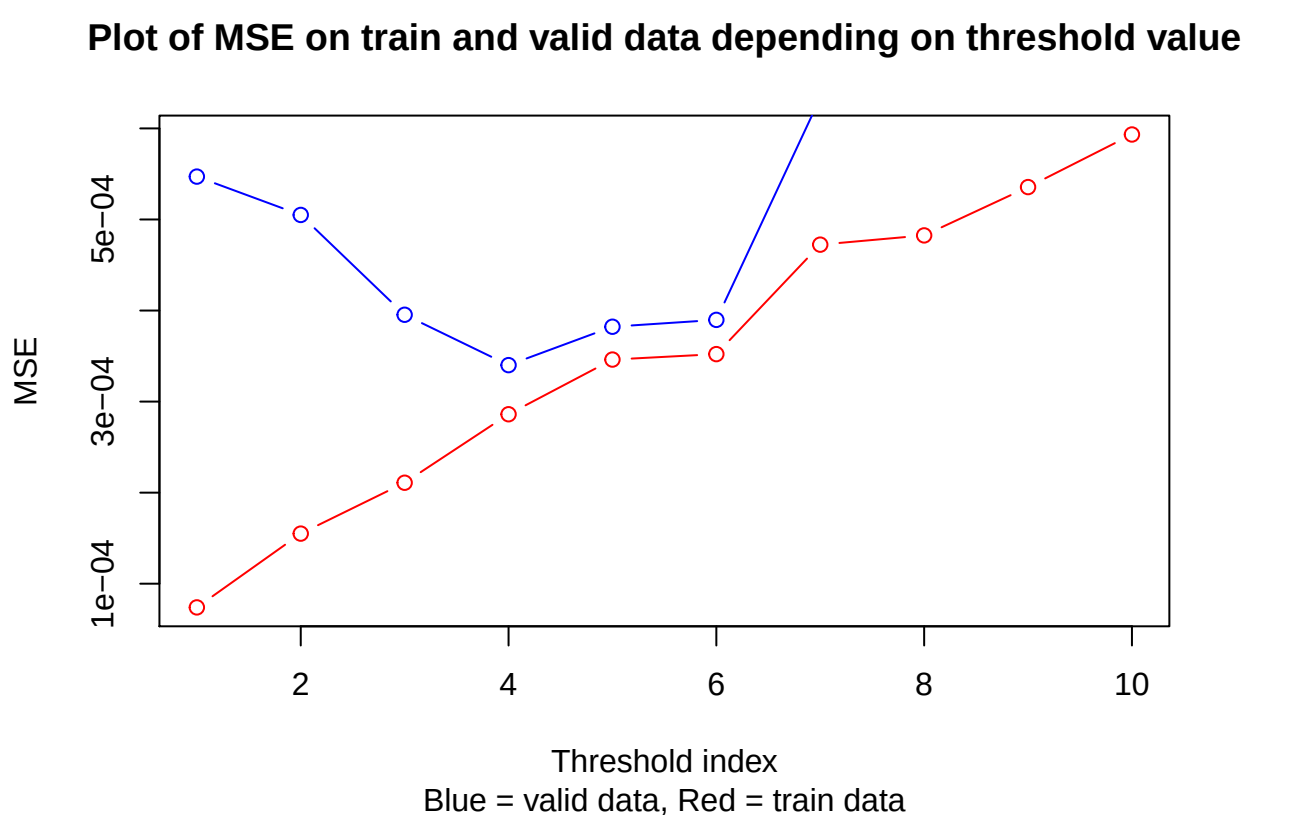
Assignment 3

```
#install.packages("neuralnet")
library(neuralnet)
set.seed(1234567890)
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
train <- trva[1:25,] # Training
valid <- trva[26:50,] # Validation
n = dim(valid)[1]
```

```
# Random initialization of the weights in the interval [-1, 1]
winit <- runif(31, -1, 1)
trainScore = rep(0,10)
validScore = rep(0,10)
```

The code above, initializes relevant values and creates the data frame for random numbers between 0 and 10 which in turn are applied to the sinus function. The data is divided into training data and validation data in an 1:1 relation. The weights are initialized randomely. 31 weights are initialized due to a hidden layer of 10 neural nodes, 1 input node and 1 output node which give 10*1 (from input to hidden layer) + 10*1 (from hidden layer to output) + 1*10 (from input bias to hidden layer) + 1*1 (from hidden layer bias to output) = 31 arrows, i.e. weights.

```
for(i in 1:10) {
  nn_temp <- neuralnet(Sin~Var, data=train, hidden=10, threshold=i/1000,
                       startweights=winit)
  nn = as.data.frame(nn_temp$net.result)
  pred=predict(nn_temp, newdata=valid)
  trainScore[i] = 1/n*sum((nn[,1]-train$Sin)^2)
  validScore[i] = 1/n*sum((pred-valid$Sin)^2)
}
plot(1:10, trainScore[1:10],
     main="Plot of MSE on train and valid data depending on threshold value", type="b",
     col="red", xlab="Threshold index", ylab="MSE", sub="Blue = valid data, Red = train data")
points(1:10, validScore[1:10], type="b", col="blue")
```




```
min_error=min(validScore[1:10])
print(min_error)
```

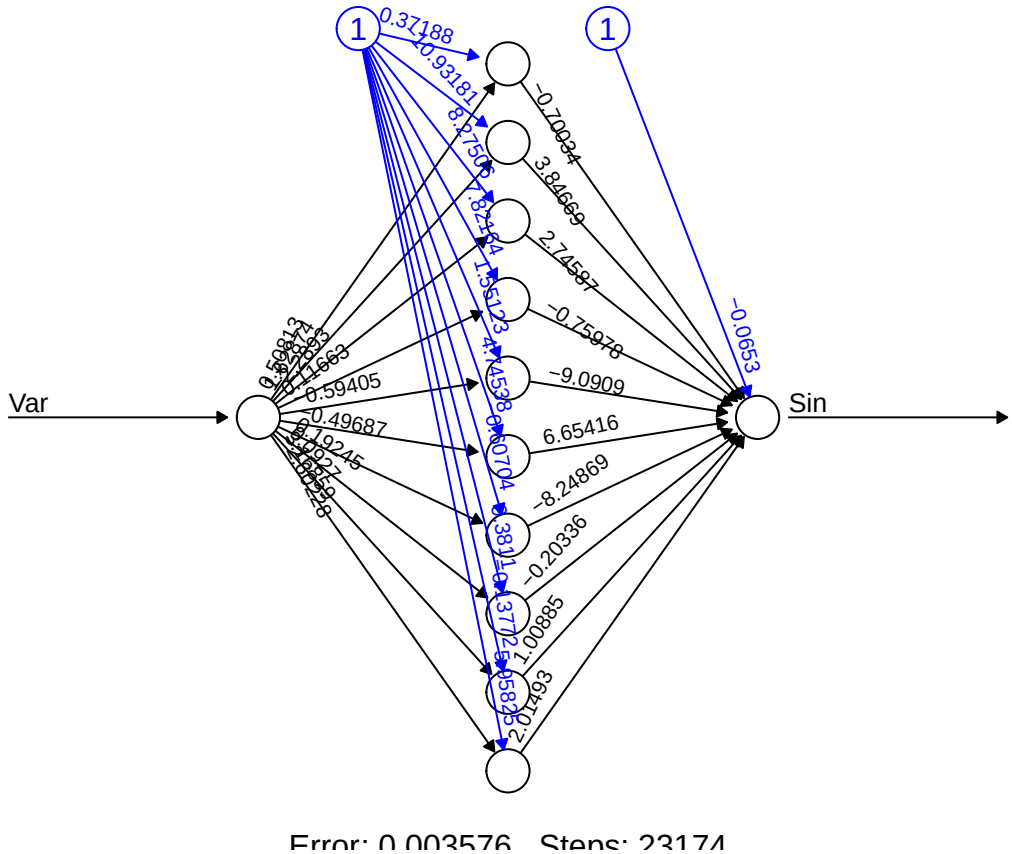
```
## [1] 0.0003400358
```

```
optimal_i=which(validScore[1:10] == min_error)
print(optimal_i)
```

```
## [1] 4
```

As seen in the graph above, naturally the train data performs the best when the threshold value is as small as possible, i.e. 1/1000, and performance decreases as this threshold increases for the train data. From the graph we can see that the threshold value 4/1000 performs the best on the validation data (since it results in the lowest MSE) and therefore this threshold will be used moving forward in the assignment.

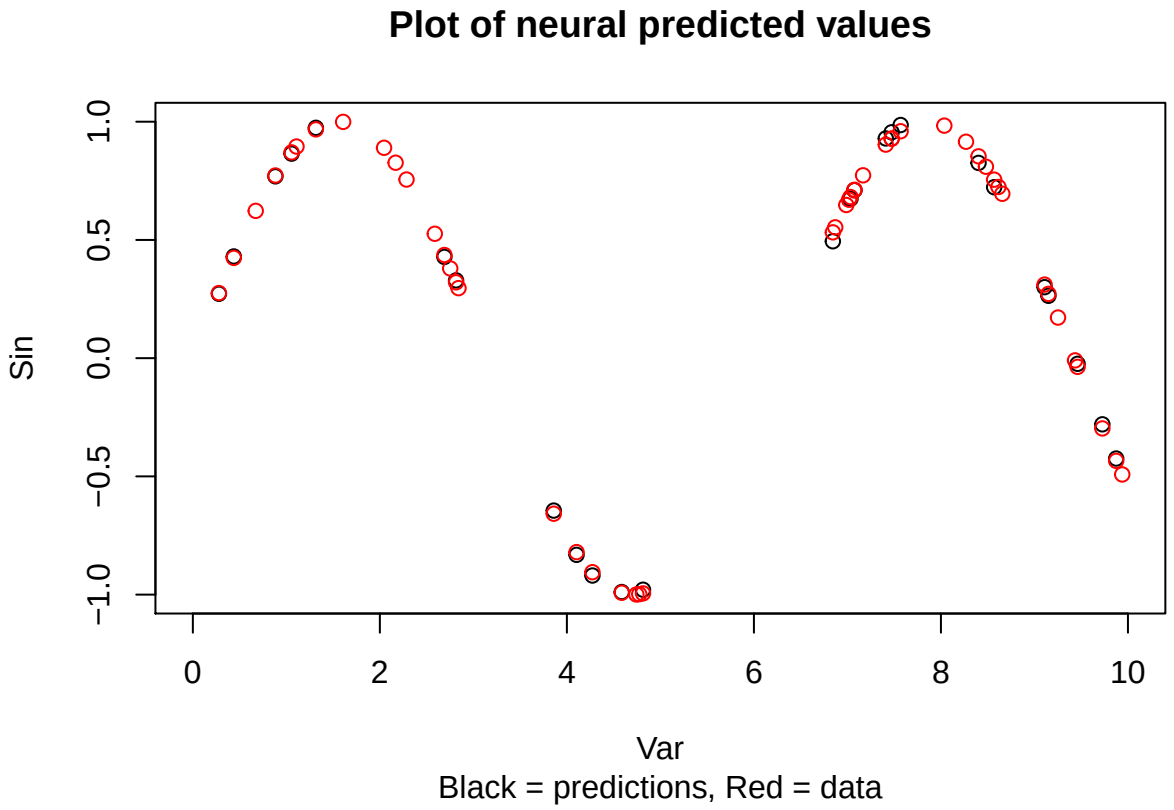
```
optimal_nn = neuralnet(Sin~Var, data=train, hidden=10, threshold=optimal_i/1000,
                       startweights=winit)
plot(optimal_nn, fontsize = 10, rep="best")
```




```
# Plot of the predictions (black dots) and the data (red dots)
plot(prediction(optimal_nn)$rep1, xlim=c(0,10), ylim=c(-1,1),
     main = "Plot of neural predicted values", sub="Black = predictions, Red = data")
```

```
## Data Error:  0;
```

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



The optimal neural network with threshold 4/1000 is chosen which results in the neural network shown above. The last two graphs briefly show how similar the predicted values from the model are in comparison to the real sinus curve. One can conclude that the neural network created resembles the shape of the sinus curve with quite a precision.

Appendix

Code for Assignment 1

```
RNGversion('3.5.1')
set.seed(1234567890)
#install.packages("geosphere")
library(geosphere)
stations <- read.csv("stations.csv")
temps <- read.csv("temps50k.csv")
#A join operation on "station_number"
st <- merge(stations,temps,by="station_number")
n = dim(st)[1]
#Kernel weighting factors
h_distance <- 100000
h_date <- 20
h_time <- 2
#Latitude of interest
a <- 59.4059
#Longitude of interest
b <- 18.0256
#Coordinates for Danderyd
#Create a vector of the point of interest
placeOI = c(a, b)
dateOI <- as.Date("1995-07-29") # The date to predict (up to the students), my birth date
timesOI = c("04:00:00", "06:00:00", "08:00:00", "10:00:00", "12:00:00", "14:00:00", "16:00:00", "18:00:0
          "22:00:00", "24:00:00")

plotDist = function(dist, h){
  u = dist/h
  plot(exp(-u^2), type="l", main="Plot of kernel wights for distances", xlab="Distance")
}

dist = seq(0, 100000, 1)
plotDist(dist, h_distance)

plotDate = function(date, h){
  u = date/h
  plot(exp(-u^2), type="l", main="Plot of kernel wights for dates", xlab="Days")
}

date = seq(-182,182,1)
plotDate(date, h_date)

plotTime = function(time, h){
  u = time/h
  plot(exp(-u^2), type="l", main="Plot of kernel wights for time", xlab="Hours")
}

time = seq(-12,12,1)
plotTime(time, h_time)

#Remove posterior data
filter_posterior = function(date, time, data){
```

```
  return(data[which(as.numeric(difftime(strptime(paste(date, time, sep=" "), format="%Y-%m-%d %H:%M:%S"
                       strptime(paste(data$date, data$time, sep=" "),format="%Y-%m-%d %H:%M:%S")))>0),
}

#A gaussian function for the difference in distance
gaussian_dist = function(place, data, h) {
  lat = data$latitude
  long = data$longitude
  points = data.frame(lat,long)
  u = distHaversine(points, place)/h
  return (exp(-u^2))
}

xy = gaussian_dist(placeOI, st, h_distance)

#A gaussian function for difference in days
gaussian_day = function(date, data, h){
  compare_date = as.Date(data$date)
  diff = as.numeric(date-compare_date)
  for (i in 1:length(diff)) {
    if (diff[i] > 365) {
      diff[i] = diff[i] %% 365
      if(diff[i]>182){
        diff[i]=365-diff[i]
      }
    }
  }
  u = diff/h
  return (exp(-u^2))
}

#A gaussian function for difference in hours
gaussian_hour = function(hour, data, h){
  compare_hour = strptime(data$time, format="%H:%M:%S")
  compare_hour = as.numeric(format(compare_hour, format="%H"))
  hour = strptime(hour, format="%H:%M:%S")
  hour = as.numeric(format(hour, format="%H"))
  diff = abs(hour-compare_hour)
  for (i in 1:length(diff)){
    if(diff[i]>12){
      diff[i] = 24-diff[i]
    }
  }
  u=diff/h
  return(exp(-u^2))
}

#Defining values that will be used in loop below
kernel_sum = c()
kernel_mult = c()

#Looping through time array and data points in nested loop to calculate the 11 kernel values
for (time in timesOI) {
```

```
  filtered_data = filter_posterior(dateOI, time, st)
  kernel_dist = gaussian_dist(placeOI, filtered_data, h_distance)
  kernel_day = gaussian_day(dateOI, filtered_data, h_date)
  kernel_time = gaussian_hour(time, filtered_data, h_time)
  sum_kernel = kernel_dist+kernel_day+kernel_time
  temp_sum = sum(sum_kernel * filtered_data$air_temperature)/sum(sum_kernel)
  mult_kernel = kernel_dist*kernel_day*kernel_time
  temp_mult = sum(mult_kernel * filtered_data$air_temperature)/sum(mult_kernel)
  kernel_sum = c(kernel_sum, temp_sum)
  kernel_mult = c(kernel_mult, temp_mult)
}


plot(kernel_sum, type="o", main ="Temperature estimate through sum of factors", xlab="Time",
     ylab="Est. temperature")
axis(1, at=1:length(timesOI), labels=timesOI)
plot(kernel_mult, type="o", main="Temperature estimate through product of factors", xlab="Time",
     ylab="Est. temperature")
axis(1, at=1:length(timesOI), labels=(timesOI))
```

Code for Assignment 3

```
RNGversion('3.5.1')
#install.packages("neuralnet")
library(neuralnet)
set.seed(1234567890)
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
train <- trva[1:25,] # Training
valid <- trva[26:50,] # Validation
n = dim(valid)[1]
# Random initialization of the weights in the interval [-1, 1]
winit <- runif(31, -1, 1)
trainScore = rep(0,10)
validScore = rep(0,10)
for(i in 1:10) {
  nn_temp <- neuralnet(Sin~Var, data=train, hidden=10, threshold=i/1000, startweights=winit)
  nn = as.data.frame(nn_temp$net.result)
  pred=predict(nn_temp, newdata=valid)
  trainScore[i] = 1/n*sum((nn[,1]-train$Sin)^2)
  validScore[i] = 1/n*sum((pred-valid$Sin)^2)
}
plot(1:10, trainScore[1:10], type="b", col="red", xlab="Threshold index", ylab="MSE")
points(1:10, validScore[1:10], type="b", col="blue")
min_error=min(validScore[1:10])
print(min_error)
optimal_i=which(validScore[1:10] == min_error)
print(optimal_i)
optimal_nn = neuralnet(Sin~Var, data=train, hidden=10, threshold=optimal_i/1000, startweights=winit)
plot(optimal_nn)
# Plot of the predictions (black dots) and the data (red dots)
par(new=FALSE)
plot(prediction(optimal_nn)$rep1)
```

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