```
# Appendix 1

################
# Assignment 1 #
################

# Libraries
library(geosphere)

# Setup
set.seed(12345)
stations.dataframe = read.csv("stations.csv", fileEncoding="latin1")
temps.dataframe = read.csv("temps50k.csv", fileEncoding="latin1")
dataframe = merge(stations.dataframe, temps.dataframe, by="station_number")
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 = numeric(length(times))

# Functions
# Returns the data excluding those with dates after given date
get_filtered_data_date = function(date){
  dates.valid = as.Date(dataframe$date)<=as.Date(date)
  return(dataframe[dates.valid,])
}

# Returns the data excluding those with time after the time of a given date
get_filtered_data_time = function(data, date, time){
  diff = as.numeric(difftime(strptime(time, format="%H"), strptime(data$time, format="%H"), units="hours"))
  times.valid = as.Date(data$date)<as.Date(date) | (diff < 0)
  return(data[times.valid,])
}

# Returns distances between one point and other points
get_distances = function(point, points){
  return(distHaversine(point, points))
}

# Returns the time difference between one time and other times
get_time_differences = function(time, times){
  time = strptime(time, format="%H:")
  times = strptime(times, format="%H:")
  times.diff = as.numeric(difftime(time,times, units="hours"))
  times.diff = times.diff %% 24
  return(times.diff)
}

# Returns the date difference between one date and other dates
get_date_differences = function(date, dates){
    date = as.Date(date)
    dates = as.Date(dates)
    date.diff = as.numeric(difftime(date,dates))
    date.diff = date.diff
    return(date.diff)
}

# Returns the gaussian kernel value of given u
get_gaussian_kernel = function(u){
  return(exp(-u^2))
}

# Returns the kernel values for all the distances of given data to a point
get_gaussian_kernel_distance = function(point, data, h.distance){
  points = matrix(c(data$latitude, data$longitude), nrow=dim(data[1]), ncol=2)
  distances = get_distances(point, points)
  u.distance = distances/h.distance
  gaussian.kernel.distance = get_gaussian_kernel(u.distance)
  return(gaussian.kernel.distance)
}

# Returns the kernel values for all the dates of given data to a date
get_gaussian_kernel_date = function(date, data, h.days){
  days = get_date_differences(date, data$date)
  u.days = days/h.days
  gaussian.kernel.date = get_gaussian_kernel(u.days)
  return(gaussian.kernel.date)
}

# Returns the kernel values for all the times of given data to a time
get_gaussian_kernel_time = function(time, data, h.time){
  times = get_time_differences(time, data$time)
```

```
  u.times = times/h.time
  gaussian.kernel.time = get_gaussian_kernel(u.times)
  return(gaussian.kernel.time)
}

# Returns a prediction using the sum of kernels
y_sum = function(kernel.distance, kernel.date, kernel.time, data){
  kernel.sum = kernel.distance + kernel.date + kernel.time
  y.sum = (kernel.sum %*% data$air_temperature) / sum(kernel.sum)
  return(y.sum)
}

# Returns a prediction using the product of kernels
y_prod = function(kernel.distance, kernel.date, kernel.time, data){
  kernel.prod = kernel.distance * kernel.date * kernel.time
  print(kernel.distance[1])
  print(kernel.date[1])
  print(kernel.time[1])
  print(kernel.prod[1])
  y.prod = (kernel.prod %*% data$air_temperature) / sum(kernel.prod)
  return(y.prod)
}

# Find reasonable kernal value
# values to test
xgrid.distance = seq(0,1000000)
xgrid.date = seq(0,31)
xgrid.time = seq(0:24)

# h_values to test
h_distance = 100000
h_date = 7
h_time = 4

# Plot the gaussian value for given sequence
plot(xgrid.distance, get_gaussian_kernel(xgrid.distance/h_distance), type="l",
     main="Gassian Kernal value for different location distances", xlab="Distance (m)", ylab="Kernal Value")
plot(xgrid.date, get_gaussian_kernel(xgrid.date/h_date), type="l",
     main="Gassian Kernal value for different date distances", xlab="Date Difference (d)", ylab="Kernal Value")
plot(xgrid.time, get_gaussian_kernel(xgrid.time/h_time), type="l",
     main="Gassian Kernal value for different time distances", xlab="Time Difference (h)", ylab="Kernal Value")

# Implementation
# Coordinates
# Stockholm
stockholm_lat = 59.3293235
stockholm_long = 18.0685808

# Kiruna
kiruna_lat = 67.85
kiruna_long = 20.2166667

# Malmö
malmo_lat = 55.60587
malmo_long = 13.00073

# Implementation
point = c(kiruna_lat, kiruna_long)
point = c(stockholm_lat, stockholm_long)
point = c(malmo_lat, malmo_long)
date = "2010-12-30"
data.filtered.date = get_filtered_data_date(date) # Filter data based on date
y.prod = numeric(length(times))
y.sum = numeric(length(times))

# Calculate the predictions
for(i in 1:length(times)){
  data.filtered.time = get_filtered_data_time(data.filtered.date, date, times[i]) # Remove times after current time
  gaussian.kernel.distance = get_gaussian_kernel_distance(point, data.filtered.time, h_distance)
  gaussian.kernel.date = get_gaussian_kernel_date(date, data.filtered.time, h_date)
  gaussian.kernel.time = get_gaussian_kernel_time(times[i], data.filtered.time, h_time)
  y.sum[i] = y_sum(gaussian.kernel.distance, gaussian.kernel.date, gaussian.kernel.time, data.filtered.time)
  y.prod[i] = y_prod(gaussian.kernel.distance, gaussian.kernel.date, gaussian.kernel.time, data.filtered.time)
}
# Plot the predicted temperatures
# Sum of kernels
plot(y.sum, xaxt='n', main="Predicted Temperature (Sum of kernels)", ylab="Temperature", xlab="Time", ylim=c(-25,3))
axis(1, at=1:length(times), labels=times)

# Product of kernels
```

```
plot(y.prod, xaxt='n', main="Predicted Temperature (Product of kernels)", ylab="Temperature", xlab="Time",
ylim=c(-25,3))
axis(1, at=1:length(times), labels=times)
```