Stat 413 Assignment 3

Milagros N. Cortez

2023-04-13

1 Optimizing the Ackley function

for $x \in \mathbb{R}^2$, the Ackley function is

$$f(x) = -20exp(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}) - exp(\frac{1}{d}\sum_{i=1}^{d} cos(2\pi x_i)) + 20 + e$$

Let the search domain be the hypercube $\mathcal{D} = [-32, 32]^d$. Compare the performance of three basic search methods - SRS, LRS, ELRS- on this function in dimensions $d = 2, 4, 6$. Don’t forget to tune the parameters in LRS and ELRS to try to improve their performance.

We set the Ackley function in R and plot it in 2D:

```
set.seed(423)
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.1.3
```

```
library(plotly)
```

```
## Warning: package 'plotly' was built under R version 4.1.3
```

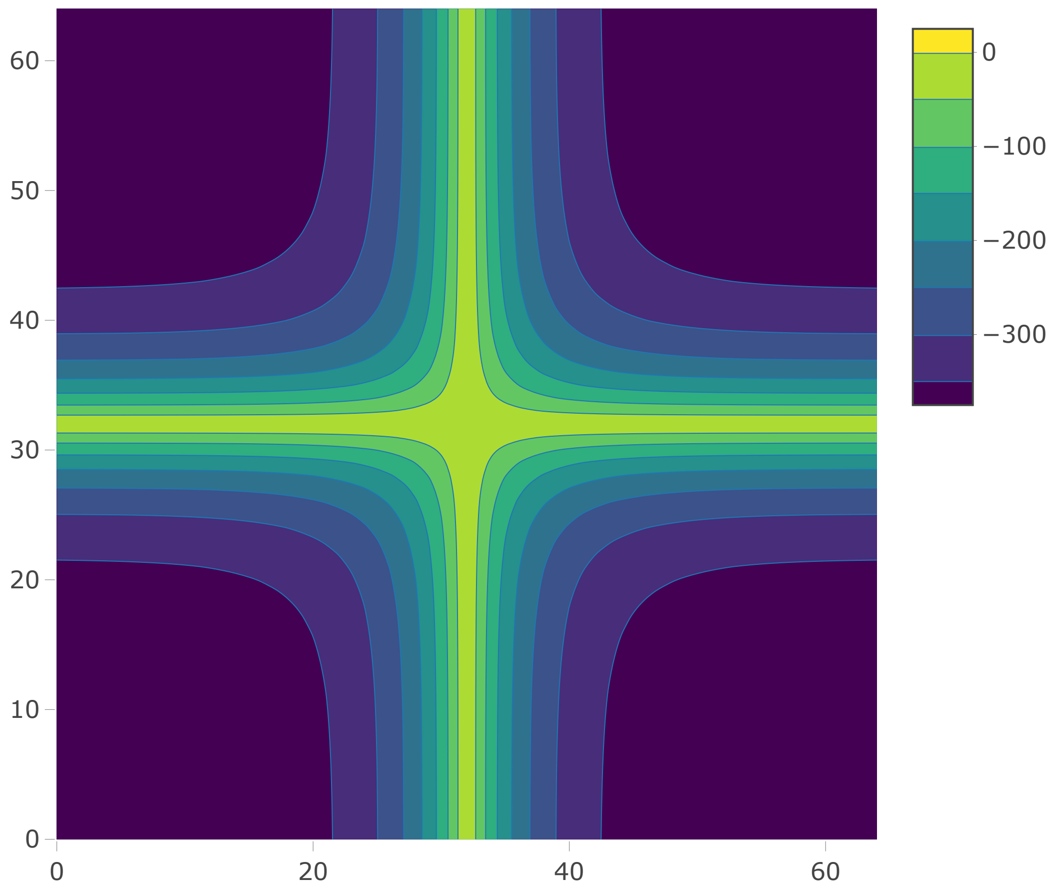
```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```r
# Ackley Function
ackley = function(th){
  len = length(th)
  val1 = sum(th^2)
  val2 = sum(cos(2*pi*th))
  totf = -20*exp(-0.2*sqrt((1/len)*val1))-exp((1/len)*val2)+20+exp(1)
  return(totf)
}

# plotting the function in 2D
tt = seq(-32, 32, 1)
x = matrix(-20*exp(-0.2*sqrt(tt^2))-exp(cos(2*pi*tt))+20+exp(1))
y = matrix(-20*exp(-0.2*sqrt(tt^2))-exp(cos(2*pi*tt))+20+exp(1))
mm = -(x%*%t(y))
fig = plot_ly()%>%add_trace(z = mm, type = "contour")
fig
```



For this assignment we work with stochastic optimization and look at three methods of basic stochastic search: Simple Random Search (SRS), Localized Random Search (LRS), and Enhanced Localized Random Search (ELRS).

SRS

SRS is the easiest method of stochastic search and optimization. This method consists on picking points (θ's) at random. The θ's can be sampled uniformly or using any other distribution of choice.

Algorithm

The algorithm for a simple random search consists on inputting a loss function $L(\theta)$ to minimize, a domain Θ to search over, and a Maximum number of samples n. Using these inputs, the algorithm then picks the best θ inside the domain Θ at random denoted as θ_{best}. This value is then applied to the loss function to get $L_{best} = L(\theta_{best})$. Then for i from 2 to n we similarly repeat the process of picking a $\theta \in \Theta$ at random which we use to compute $L = L(\theta)$. If $L < L_{best}$ we set $L_{best} = L$, and $\theta_{best} = \theta$. Once this process is iterated until n, the algoritm returns as output $\{\theta_{best}, L_{best}\}$

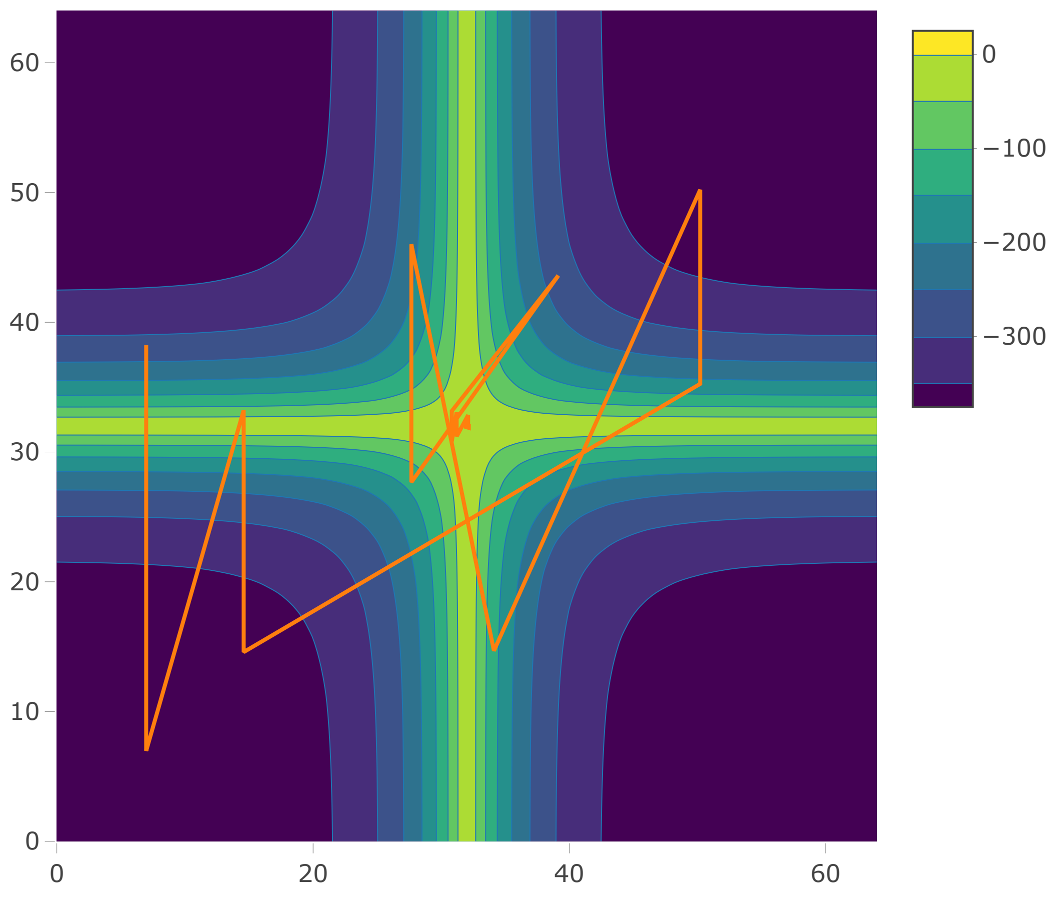
Using the information above as well as the algoritm, we apply SRS to the Ackley function:

```r
minimize.srs = function(f.loss, d, n = 1e5){
  theta = matrix(0, nrow = n, ncol = d)
  val.loss = rep(0, n)
  # simulating the first data point
  theta[1,] = runif(d, -32, 32)
  val.loss[1] = f.loss(theta[1,])
  for(i in 2:n){
    theta[i,] = runif(d, -32, 32)
    val.loss[i] = f.loss(theta[i,])
    if(val.loss[i]>val.loss[i-1]){
      theta[i,] = theta[i-1]
      val.loss[i] = val.loss[i-1]
    }
  }
  return(list(
    theta = theta,
    loss = val.loss
  ))
}
```

At $d = 2$

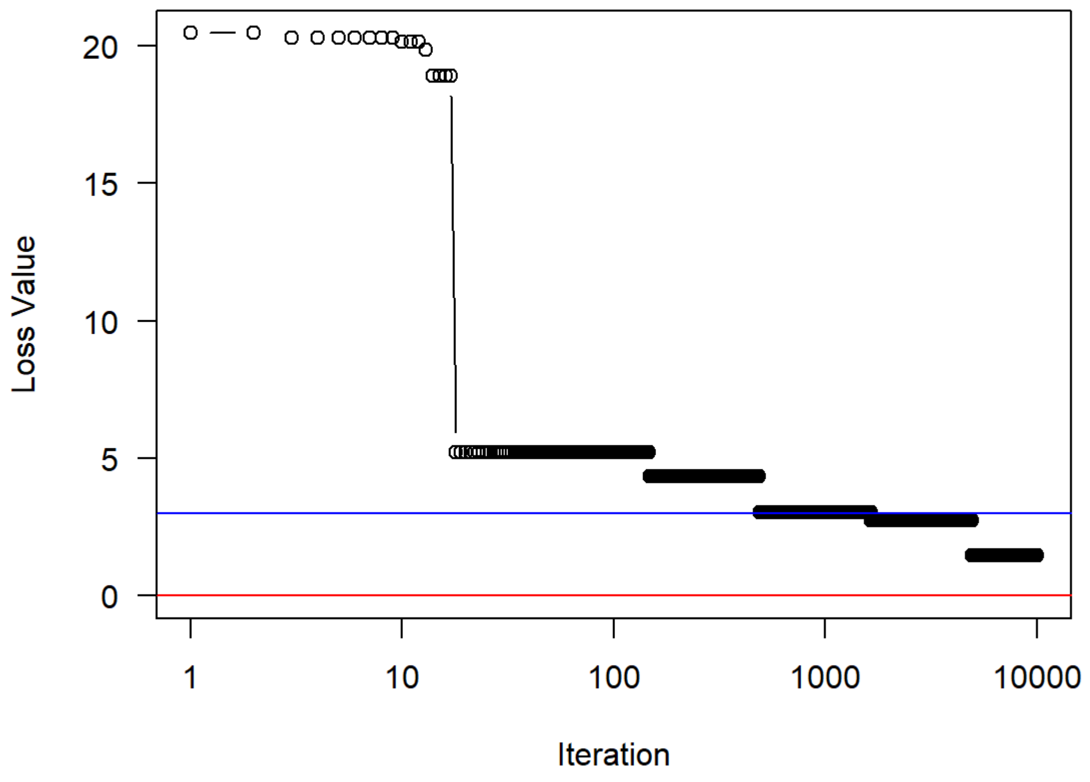
```r
# for d = 2
nn = 1e4
out = minimize.srs(f.loss = ackley, d = 2, n = nn)
```

```
# contour plot
fig%>%add_trace(x = out$theta[,1]+32, y = out$theta[,2]+32,  type = "scatter", mode = "line")
```



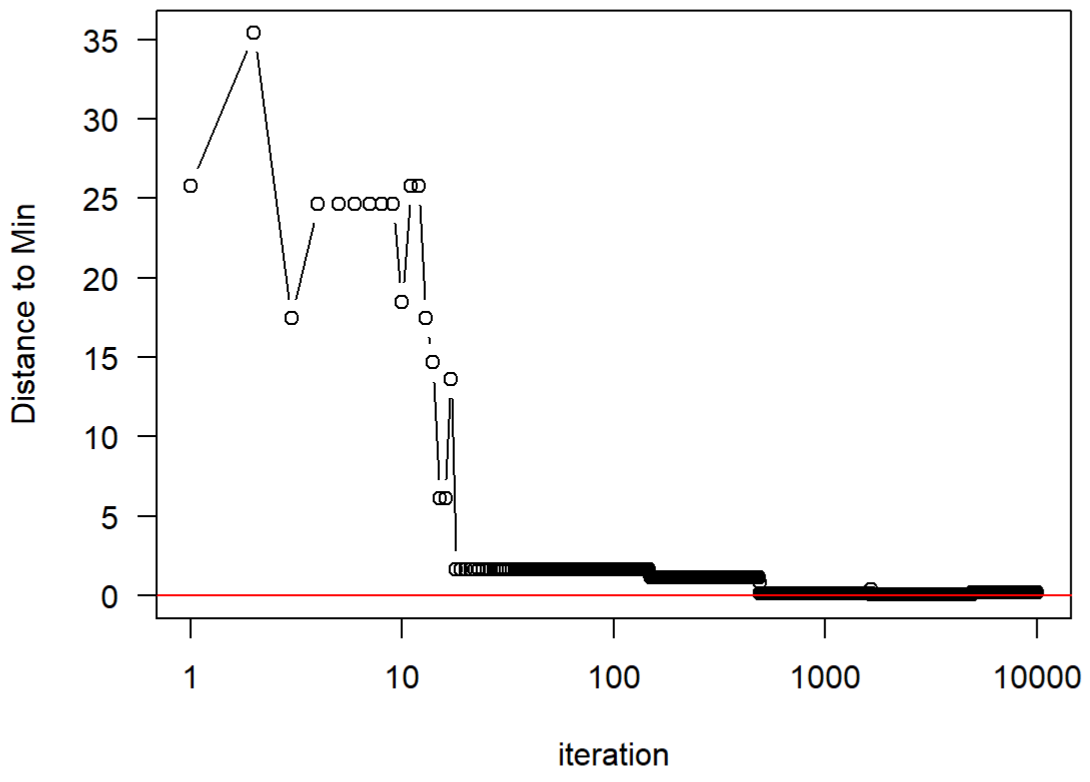
We can see from the plot above that when using SRS, this method uniformly selects a random point in $\mathcal{D} = [-32, 32]^2$ which we can see in this plot is the furthest point from the center, and at every iteration as the algorithm finds a new set of $\{\theta_{best}, L_{best}\}$ these points will approach to the center until the center of the plot is reached indicating that $\theta^* = argmin_\theta\{L(\theta)\}$ was found. A note of this plot is that the points do not show a pattern or direction towards the center for which the algorithm has a more random approach on how it finds the $\theta^* = argmin_\theta\{L(\theta)\}$ which minimizes the loss function.

```
plot(1:nn, out$loss, type = 'b', log = 'x', las = 1, ylab = "Loss Value",
     xlab = "Iteration", ylim = c(0, max(out$loss)))
abline(h = 3, col = "blue")
abline(h = 0, col = "red")
```



Looking at the loss value over each iteration plot we can see that we start at a loss value of around 20, and there is a slow decrease in the loss value until 10 iterations are reached. After 10 iterations, the loss value decreases in a step like pattern until we reach a small value that is less than 3 (blue horizontal line) and more than 0 (red horizontal line).

```
th.dist = sqrt(rowSums((out$theta)^2))
plot(1:nn, th.dist, type = 'b', log = 'x', las = 1, ylab = "Distance to Min", xlab = "iteratio
n", ylim = c(0, max(th.dist)))
abline(h = 0, col = "red")
```



From the plot above we can see that the distance of the points to $\theta^* = argmin_\theta\{L(\theta)\}$ decreases as we go over each iteration. We can see that from the first iteration until a bit over 10 iterations the distance of the θ_{best} to θ^* varies more than the distance of the θ_{best} to θ^* of the rest of the iterations until 10000. The θ_{best} of the iterations after 10 until 10000 have a smaller distance which indicates they are close to θ^* until the distance is close to 0 between iteration 1000 and 10000, which matches to the penultimate step at the right of the loss value per iteration plot.

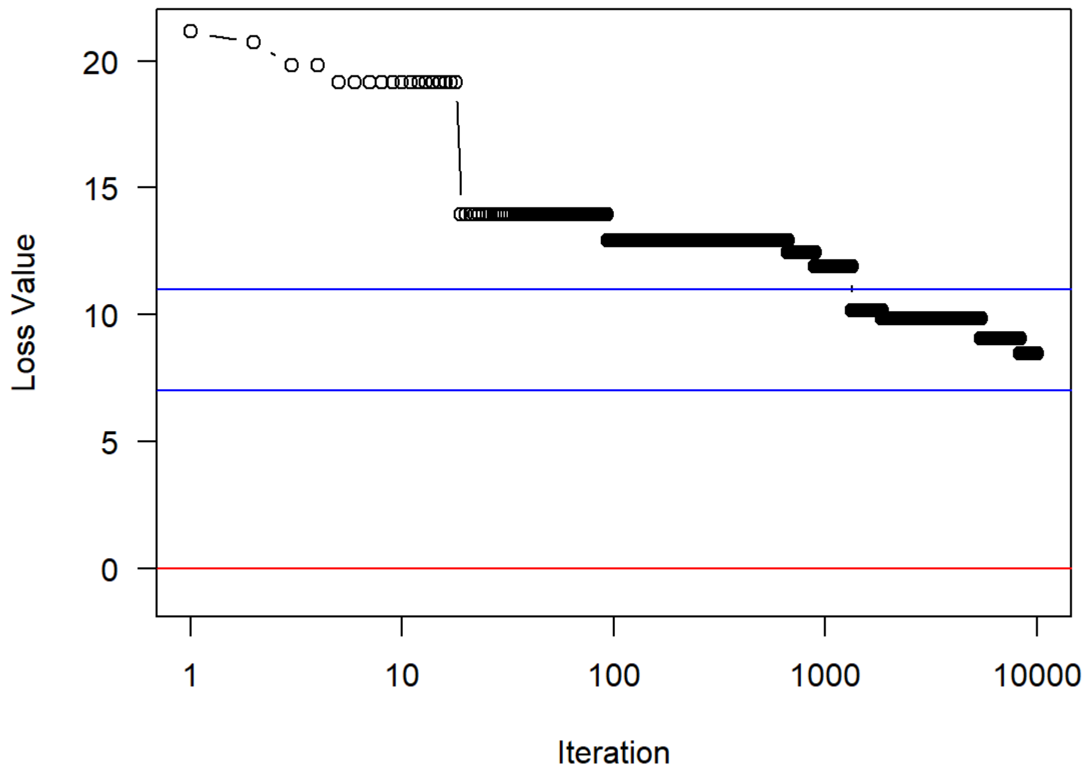
Based on the plots above, we can take a value in between 1000 to 10000 to find the value of the loss function that has a distance to the minimum close to 0 indicating that $\theta_{best} \approx \theta^* = argmin_\theta\{L(\theta)\}$

```
x = 10000-1000
xf = 1000+x
cat("Theta Value:    ", out$theta[xf,1], "\nLoss Value:     ", out$loss[xf], "\nDistance to mi
n:", sqrt(rowSums((out$theta)^2))[xf])
```

```
## Theta Value:     0.1525298
## Loss Value:      1.448566
## Distance to min: 0.2157097
```

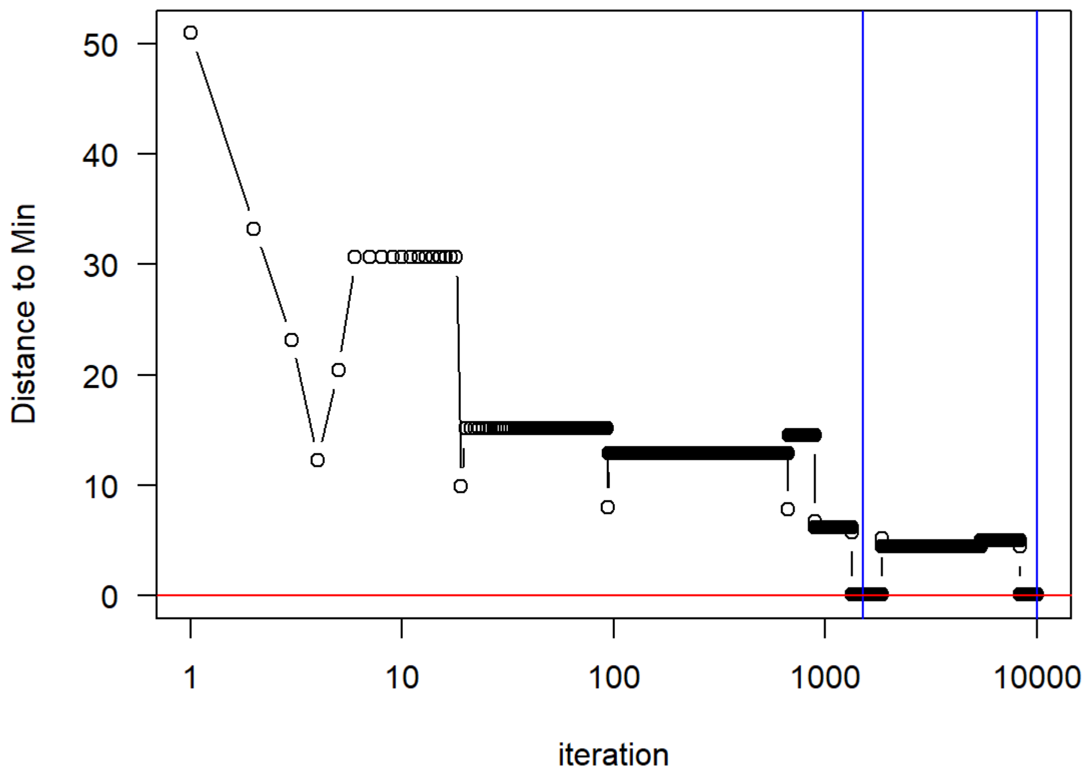
At $d = 4$

```
# for d = 4
nn = 1e4
out2 = minimize.srs(f.loss = ackley, d = 4, n = nn)

plot(1:nn, out2$loss, type = 'b', log = 'x', las = 1, ylab = "Loss Value",
     xlab = "Iteration", ylim = c(-1, max(out2$loss)))
abline(h = 7, col = "blue")
abline(h = 11, col = "blue")
abline(h = 0, col = "red")
```



Looking at the loss value over each iteration plot we can see that we start at a loss value of around 20, and there is a gradual decrease in the loss value until about 5 or 6 iterations are reached. After these iterations, the loss value decreases in a gradual step like pattern until we reach a value that is less than 11 and more than 7 which would be $L(\theta_{best}) \approx L(\theta^*)$.

```
th.dist = sqrt(rowSums((out2$theta)^2))
plot(1:nn, th.dist, type = 'b', log = 'x', las = 1, ylab = "Distance to Min", xlab = "iteratio
n", ylim = c(0, max(th.dist)))
abline(v = 1500, col = "blue")
abline(v = 10000, col = "blue")
abline(h = 0, col = "red")
```



From the plot above we can see that the distance of the points to $\theta^* = argmin_\theta\{L(\theta)\}$ decreases as we go over each iteration. We can see that even though there is a decrease in distance overall, the distance of the θ_{best} to θ^* varies throughout the plot compared to the plot for $d = 2$ where we had a gradual decrease after a bit over 10 iterations. The θ_{best} of some of the iterations between 1000 to 10000 have a smaller distance close to to 0 which indicates they are close to θ^*. These iterations are split in two parts, the first part of the iterations are before 4500 iterations, and the second part is after 4500 iterations until 10000 which match to two of the steps at the right side of the plot of the loss value over the iterations.

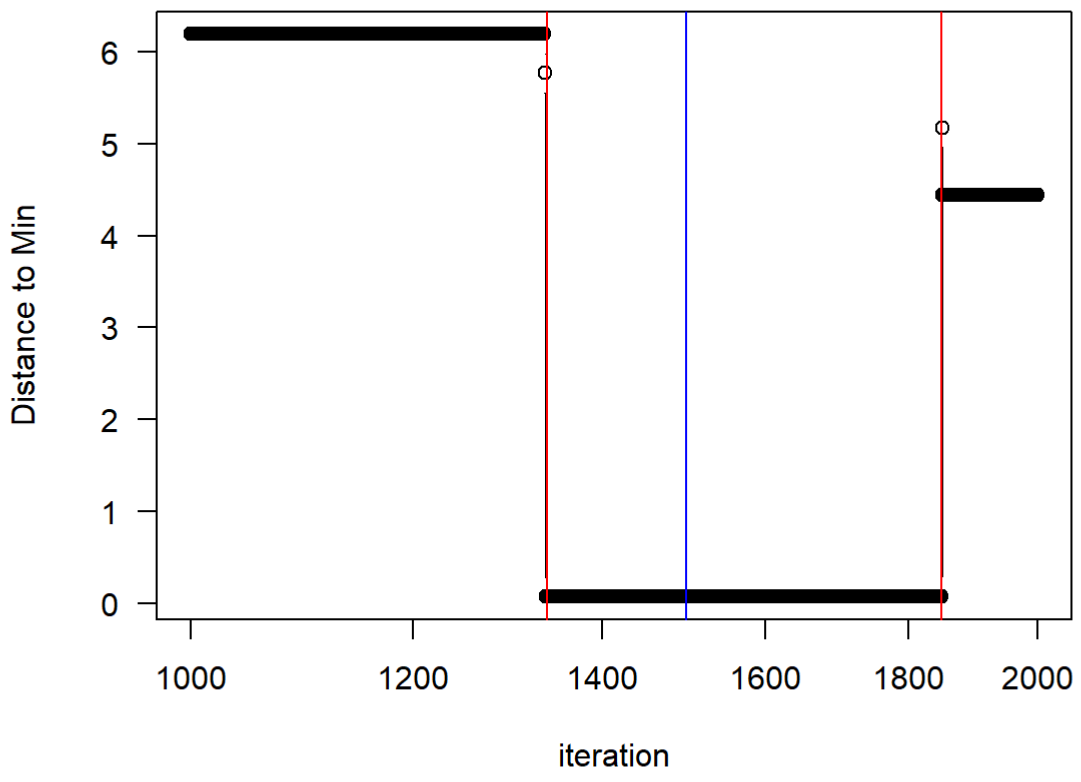
Based on the plots above, we can take two values from these two identified parts between 1000 to 10000 iterations to find the value of the loss function that has a distance to the minimum close to 0 indicating that $\theta_{best} \approx \theta^* = argmin_\theta\{L(\theta)\}$. The location of both values are also indicated by the blue vertical lines in the plot above.

```
cat("First part \n Theta Value:    ", out2$theta[1500,1], "\n Loss Value:     ", out2$loss[150
0], "\n Distance to min:", sqrt(rowSums((out2$theta)^2))[1500], "\nSecond part \n Theta Value:
", out2$theta[10000,1], "\n Loss Value:     ", out2$loss[10000], "\n Distance to min:", sqrt(row
Sums((out2$theta)^2))[10000])
```

```
## First part
##  Theta Value:     -0.03784607
##  Loss Value:      10.17128
##  Distance to min: 0.07569215
## Second part
##  Theta Value:     0.05891857
##  Loss Value:      8.453338
##  Distance to min: 0.1178371
```

From the following output we can see that we get a smaller distance between θ_{best} to θ^* from the values of the first part before 4500 iterations. We can plot this part to see in finer detail the distance to min over this smaller range of iterations:

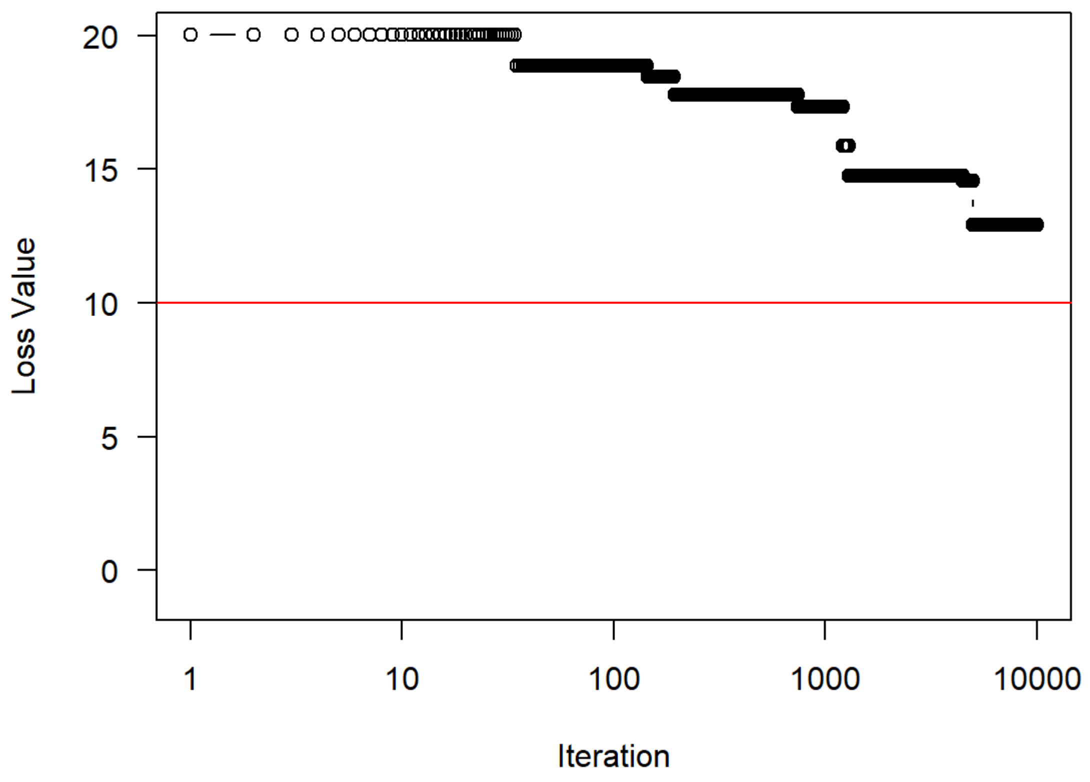
```
th.dist = sqrt(rowSums((out2$theta)^2))[1000:2000]
plot(1000:2000, th.dist, type = 'b', log = 'x', las = 1, ylab = "Distance to Min", xlab = "itera
tion")
abline(v = 1500, col = "blue")
abline(v = 1339, col = "red")
abline(v = 1849, col = "red")
```



We can see that between 1338 to 1849 iterations we get the smallest distance to min indicating that $\theta_{best} \approx \theta^* = argmin_\theta\{L(\theta)\}$, when we have a loss value of around 10.17128.

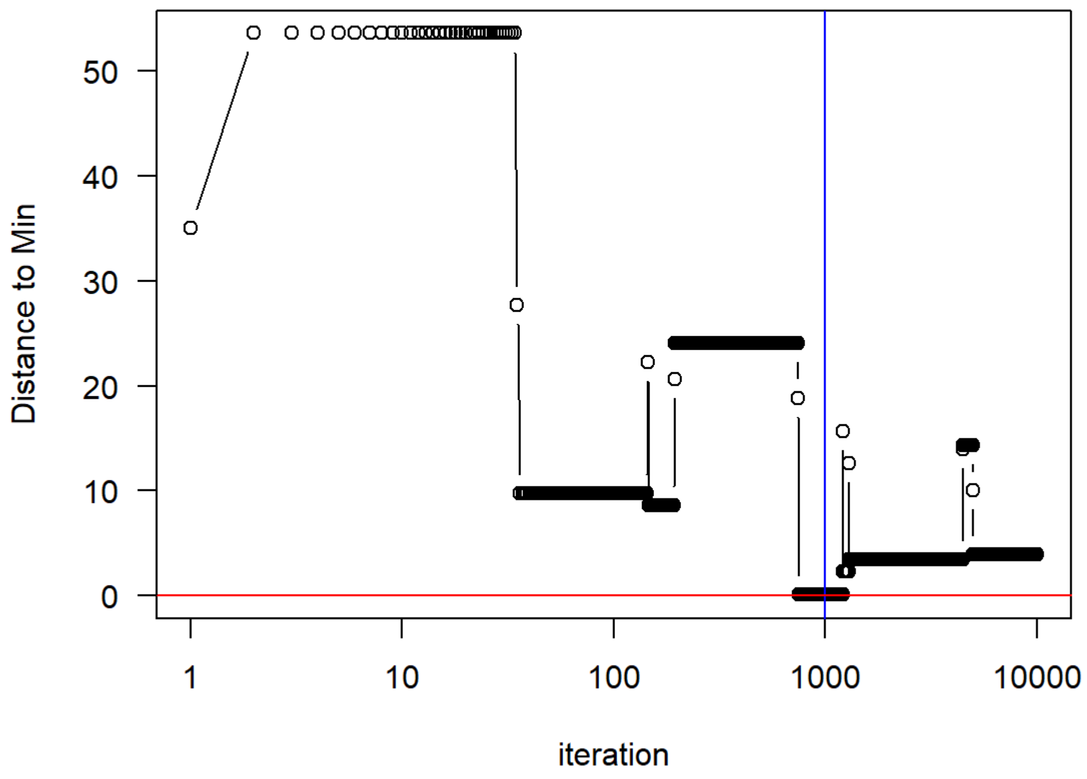
At $d = 6$

```
# for d = 6
nn = 1e4
out3 = minimize.srs(f.loss = ackley, d = 6, n = nn)

plot(1:nn, out3$loss, type = 'b', log = 'x', las = 1, ylab = "Loss Value",
     xlab = "Iteration", ylim = c(-1, max(out3$loss)))
abline(h = 10, col = "red")
```



Looking at the loss value over each iteration plot we can see that the loss value decreases in a gradual step like pattern until we reach a value over 10.

```
th.dist = sqrt(rowSums((out3$theta)^2))
plot(1:nn, th.dist, type = 'b', log = 'x', las = 1, ylab = "Distance to Min", xlab = "iteratio
n", ylim = c(0, max(th.dist)))
abline(v = 1000, col = "blue")
abline(h = 0, col = "red")
```



From the plot above we can see that even though there is a decrease in distance overall, the distance of the θ_{best} to θ^* varies throughout the plot compared to the plot for $d = 2$ where we had a gradual decrease after a bit over 10 iterations, and $d = 4$ which had less variation and a more definite gradual decrease. The θ_{best} of some of the iterations around 1000 have a smaller distance close to to 0 which indicates they are close to θ^*.

Based on the plots above, we can take the 1000th iteration to find the value of the loss function that has a distance to the minimum close to 0 indicating that $\theta_{best} \approx \theta^* = argmin_\theta\{L(\theta)\}$. The location of the value is also indicated by the blue vertical line in the plot above.

```
cat(" Theta Value:    ", out3$theta[1000,1], "\n Loss Value:     ", out3$loss[1000], "\n Distanc
e to min:", sqrt(rowSums((out3$theta)^2))[1000])
```

```
##  Theta Value:     -0.05618913 
##  Loss Value:      17.33105 
##  Distance to min: 0.1376347
```

LRS

LRS consists on a more localized search method where we pick a θ value within an established domain uniformly at random and we adjust this value randomly through the use of a random vector v. In this way, we get values of theta that move through the domain in a certain direction towards $\theta^* = argmin_\theta\{L(\theta)\}$ as opposed to SRS which had points scattered throughout the domain approaching $\theta^* = argmin_\theta\{L(\theta)\}$.

Algorithm

The algorithm for a localized random search method consists of imputing a loss function $L(\theta)$ to minimize, a domain Θ, a maximum number of samples n, and a distribution of f to sample from. With these inputs we first pick a $\theta_1 \in \Theta$ uniformly at random, and using the loss function we calculate $L_1 = L(\theta_1)$. We iterate after this from 2 to n samples, with each iteration consisting on picking a random vector $v$from f, setting $\theta_i = \theta_{i-1} + v$. Given that $\theta_i \notin \Theta$, we set $\theta_i = \theta_{i-1}$ and $L_1 = L_{i-1}$ and move on to the next iteration. Else we calculate $L_i = L(\theta_i)$, and if $L_{i-1} \leq L_i$ we set $L_i = L_{i-1}$ and $\theta_i = \theta_{i-1}$. After all the iterations the output of the algorithm is the set $\{\theta_n, L_n\}$

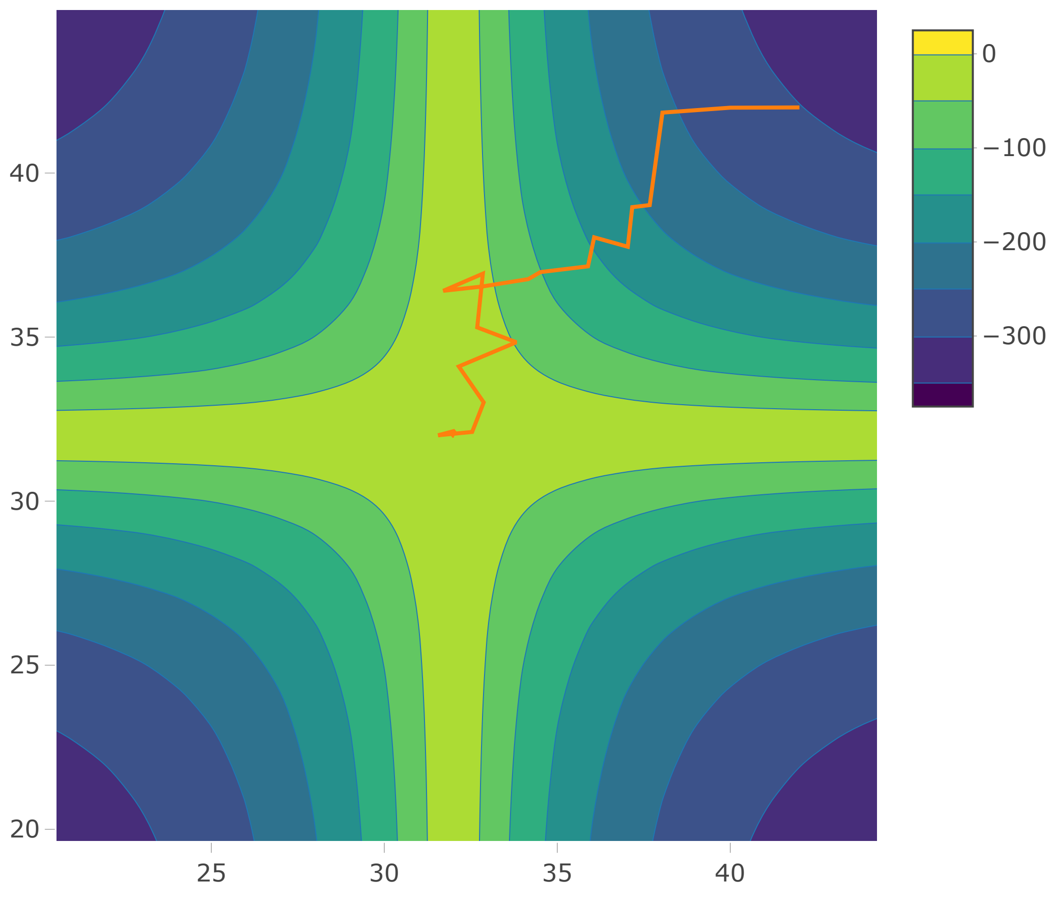
Using the information above as well as the algoritm, we apply LRS to the Ackley function:

```
minimize.lrs <- function(
  f.loss, d, n = 1e5, sig=1, startingTheta = NULL, thresh = 0.25){
  theta = matrix( 0, nrow = n, ncol = d)
  val.loss = rep(0,n)
  # Simulating the first datapoint
  if( is.null(startingTheta) ){
    theta[1,] = runif( d, -32, 32)
  }
  else {
    theta[1,] = startingTheta
  }

  val.loss[1] = f.loss( theta[1,] )

  for( i in 2:n ){
    v = rnorm(d,0,sig)
    theta[i,]   = theta[i-1,]+v
    if( prod((theta[i,]<=32)*(theta[i,]>=-32)) == 0 ){
      theta[i,] = theta[i-1,]
      val.loss[i] =  val.loss[i-1]
      next
    }

    val.loss[i] = f.loss( theta[i,] )

    if( val.loss[i]>val.loss[i-1]-thresh ){
      theta[i,] = theta[i-1,]
      val.loss[i] = val.loss[i-1]
    }
  }
  return(
    list(
      theta = theta,
      loss  = val.loss
    )
  )
}
```

At $d = 2$

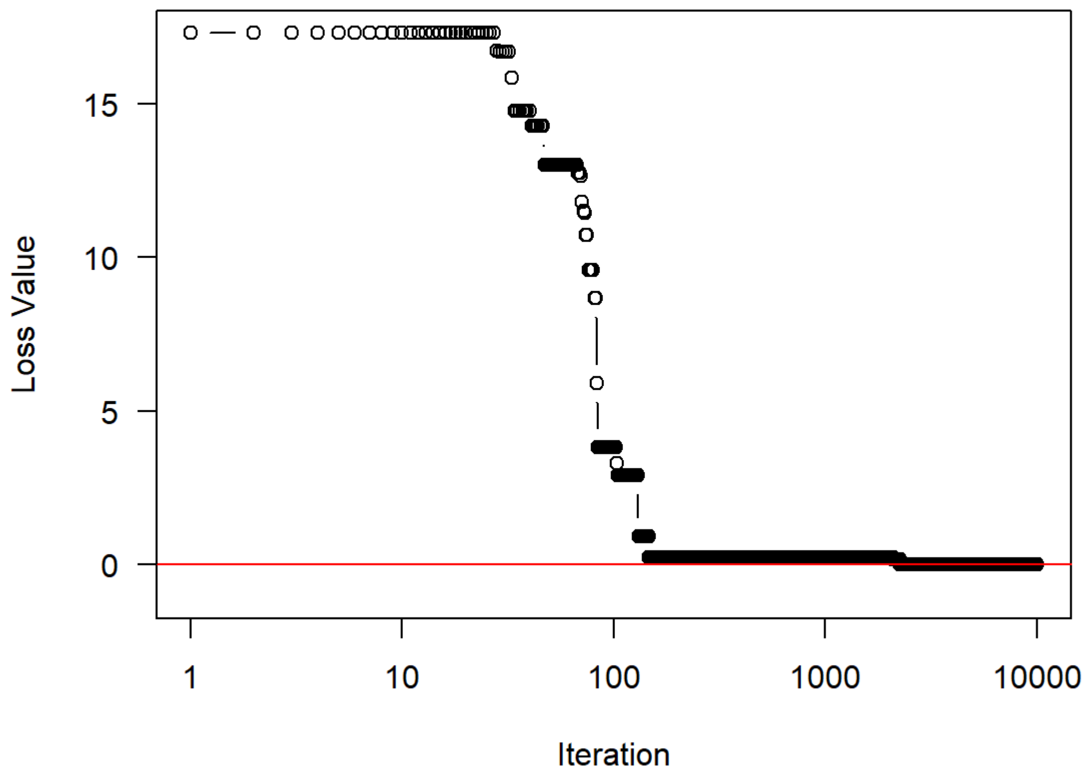
Based on the SRS results, we pick a starting theta value of 10 for each d as it is relatively close to 0 for the rest of the problem.

```
# for d = 2
nn = 1e4
outl1 = minimize.lrs(f.loss = ackley, d=2, n = nn, sig=1, startingTheta = c(10, 10), thresh = 0)

# contour plot
fig%>%add_trace(x = outl1$theta[,1]+32, y = outl1$theta[,2]+32,  type = "scatter", mode = "lin
e")
```



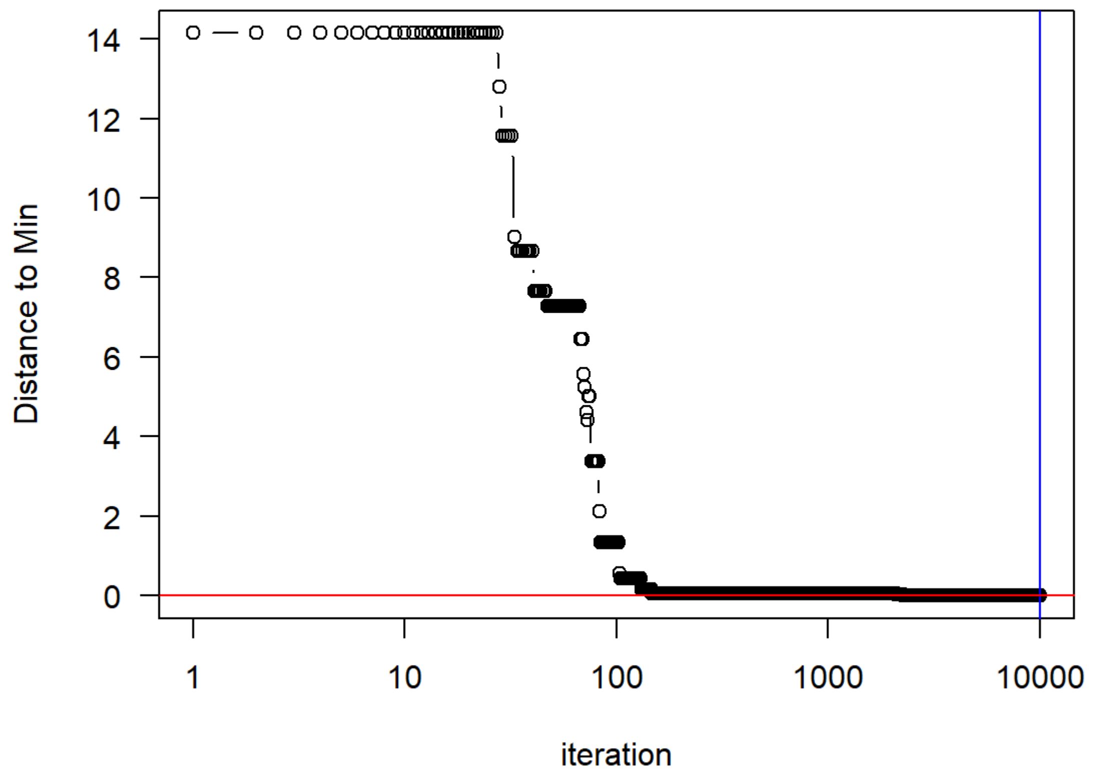
We can see from the contour plot that the points follow a more definite path compared to the contour plot from SRS.

```
plot(1:nn, outl1$loss, type = 'b', log = 'x', las = 1, ylab = "Loss Value",
     xlab = "Iteration", ylim = c(-1, max(outl1$loss)))
abline(h = 0, col = "red")
```



Looking at the loss value over each iteration plot we can see that the loss value decreases drastically after 10 iterations and after 100 iterations it slowly decreases to a value close to zero.

```
th.dist = sqrt(rowSums((outl1$theta)^2))
plot(1:nn, th.dist, type = 'b', log = 'x', las = 1, ylab = "Distance to Min", xlab = "iteratio
n", ylim = c(0, max(th.dist)))
abline(v = 10000, col = "blue")
abline(h = 0, col = "red")
```



From the plot above we can see that there is also a drastic decrease in the distance of the θ_{best} to θ^* after 10 iterations and after 100 iterations the distance slowly decreases until it approaches 0. The θ_{best} of some of the iterations after 100 have a smaller distance close to to 0 which indicates they are close to θ^*.

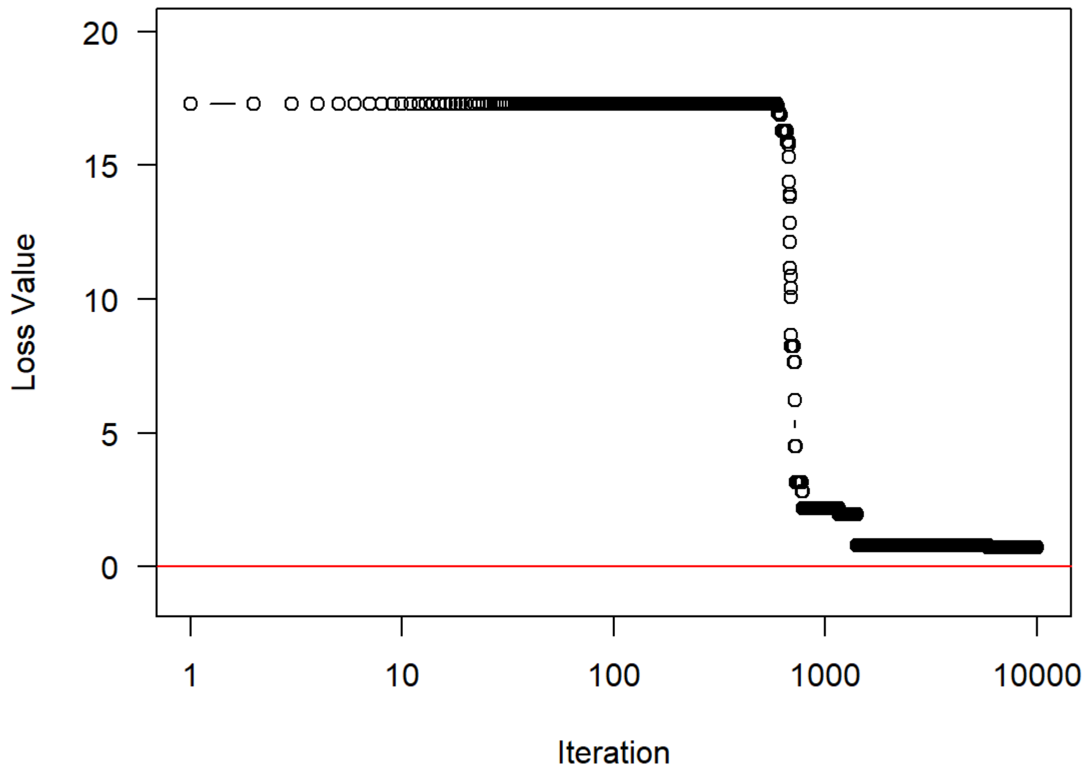
Based on the plots above, we can take the 10000th iteration to find the value of the loss function that has a distance to the minimum close to 0 indicating that $\theta_{best} \approx \theta^* = argmin_{\theta}\{L(\theta)\}$. The location of the value is also indicated by the blue vertical line in the plot above.

```
cat(" Theta Value:    ", outl1$theta[10000,1], "\n Loss Value:     ", outl1$loss[10000], "\n Dis
tance to min:", sqrt(rowSums((outl1$theta)^2))[10000])
```

```
##  Theta Value:     0.002254553
##  Loss Value:      0.007016738
##  Distance to min: 0.002425412
```

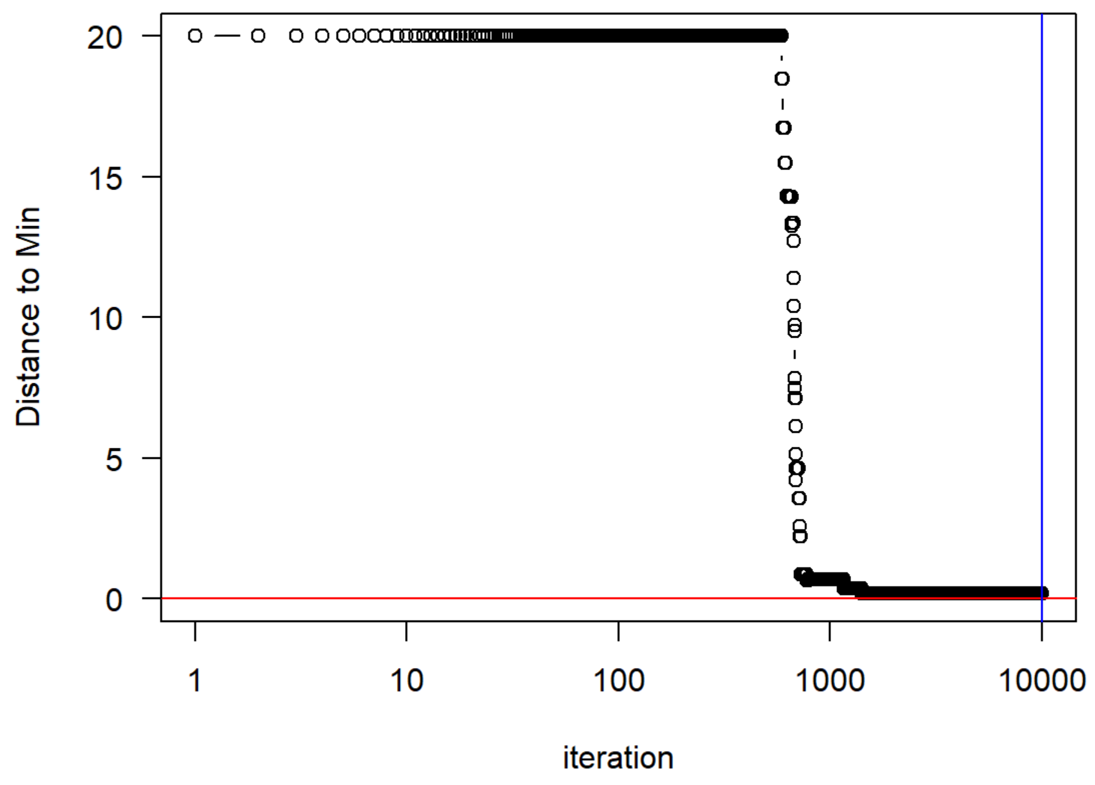
At $d = 4$

```
# for d = 4
nn = 1e4
outl2 = minimize.lrs(f.loss = ackley, d=4, n = nn, sig=1, startingTheta = c(10, 10, 10, 10), thr
esh = 0)

plot(1:nn, outl2$loss, type = 'b', log = 'x', las = 1, ylab = "Loss Value",
     xlab = "Iteration", ylim = c(-1, max(out3$loss)))
abline(h = 0, col = "red")
```



Looking at the loss value over each iteration plot we can see that the loss value decreases drastically a bit before 1000 iterations, and after 1000 iterations it slowly decreases to a value slightly above zero. This is similar to the plots at $d = 2$.

```
th.dist = sqrt(rowSums((outl2$theta)^2))
plot(1:nn, th.dist, type = 'b', log = 'x', las = 1, ylab = "Distance to Min", xlab = "iteratio
n", ylim = c(0, max(th.dist)))
abline(h = 0, col = "red")
abline(v = 10000, col = "blue")
```



From the plot above we can see that there is also a drastic decrease in the distance of the θ_{best} to θ^* a bit before 1000 iterations and after 1000 iterations the distance slowly decreases until it approaches 0. The θ_{best} of some of the iterations after 1000 have a smaller distance close to to 0 which indicates they are close to θ^*.

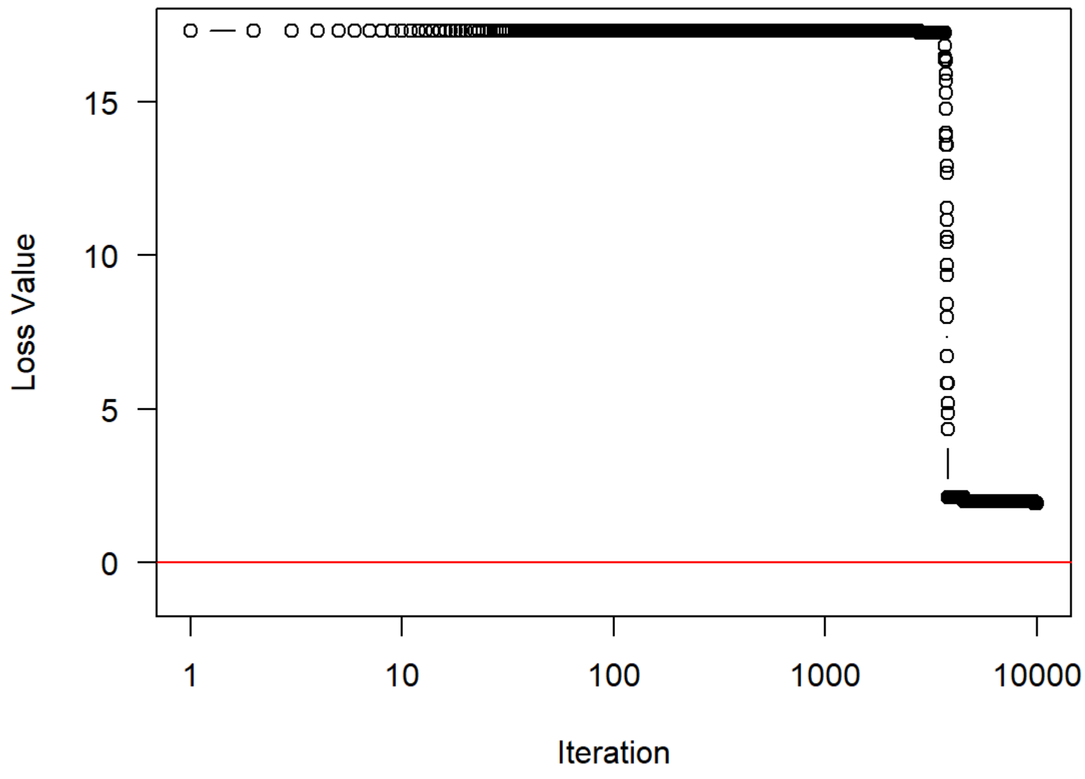
Based on the plots above, we can take the 10000th iteration to find the value of the loss function that has a distance to the minimum close to 0 indicating that $\theta_{best} \approx \theta^* = argmin_{\theta}\{L(\theta)\}$. The location of the value is also indicated by the blue vertical line in the plot above.

```
cat(" Theta Value:    ", outl2$theta[10000,1], "\n Loss Value:     ", outl2$loss[10000], "\n Dis
tance to min:", sqrt(rowSums((outl2$theta)^2))[10000])
```

```
##  Theta Value:     -0.07021837
##  Loss Value:      0.738826
##  Distance to min: 0.1796002
```

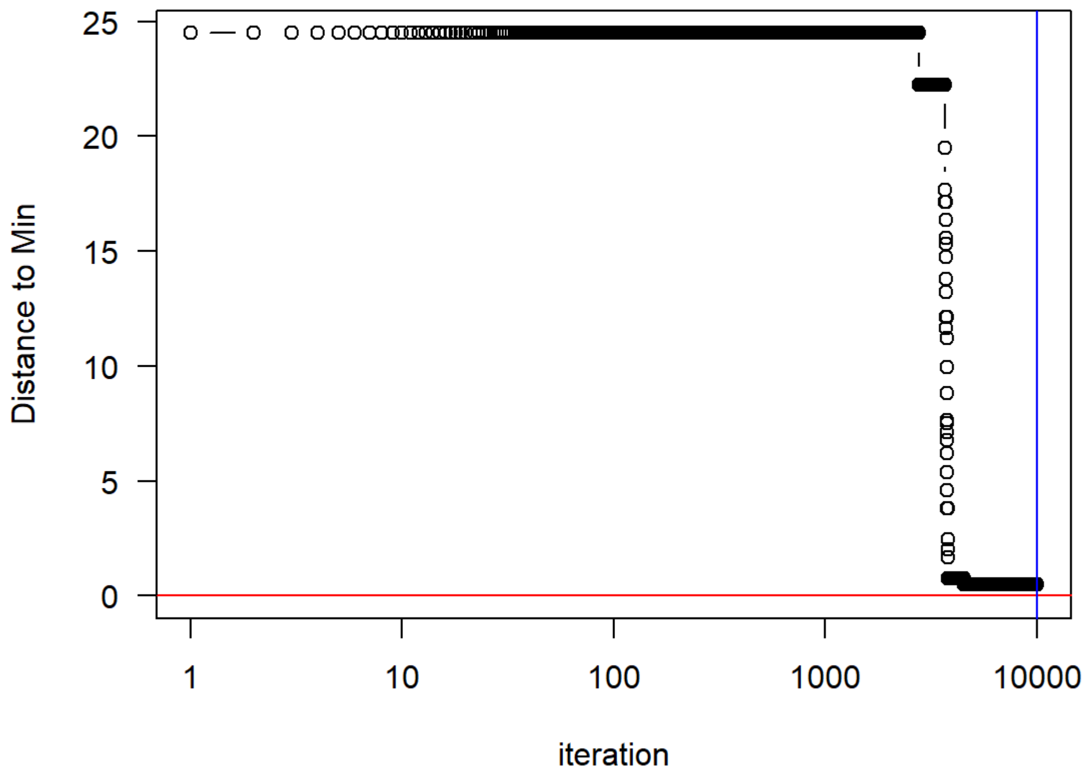
At $d = 6$

```
# for d = 6
nn = 1e4
outl3 = minimize.lrs(f.loss = ackley, d=6, n = nn, sig=1, startingTheta = c(10, 10, 10, 10, 10,
10), thresh = 0)

plot(1:nn, outl3$loss, type = 'b', log = 'x', las = 1, ylab = "Loss Value",
     xlab = "Iteration", ylim = c(-1, max(outl3$loss)))
abline(h = 0, col = "red")
```



Looking at the loss value over each iteration plot we can see that the loss value decreases drastically between 1000 and 10000 iterations, and then it slowly decreases to a value between 0 and 5. This is similar to the plots at $d = 2$ and $d = 4$, but we note that as d increases, the loss value around 10000 iterations increases.

```
th.dist = sqrt(rowSums((outl3$theta)^2))
plot(1:nn, th.dist, type = 'b', log = 'x', las = 1, ylab = "Distance to Min", xlab = "iteratio
n", ylim = c(0, max(th.dist)))
abline(h = 0, col = "red")
abline(v = 10000, col = "blue")
```



From the plot above we can see that there is also a drastic decrease in the distance of the θ_{best} to θ^* between 1000 and 10000 iterations and then the distance slowly decreases until it approaches 0. The θ_{best} of some of the iterations after 1000 have a smaller distance close to to 0 which indicates they are close to θ^*.

Based on the plots above, we can take the 10000th iteration to find the value of the loss function that has a distance to the minimum close to 0 indicating that $\theta_{best} \approx \theta^* = argmin_\theta\{L(\theta)\}$. The location of the value is also indicated by the blue vertical line in the plot above.

```
cat(" Theta Value:    ", outl3$theta[10000,1], "\n Loss Value:     ", outl3$loss[10000], "\n Dis
tance to min:", sqrt(rowSums((outl3$theta)^2))[10000])
```

```
##  Theta Value:     0.1879413
##  Loss Value:      1.949463
##  Distance to min: 0.4782237
```

ELRS

This method is similar to LRS with an additional bias term that directs the random path of points to a direction that looks to minimize the loss function.

Algorithm

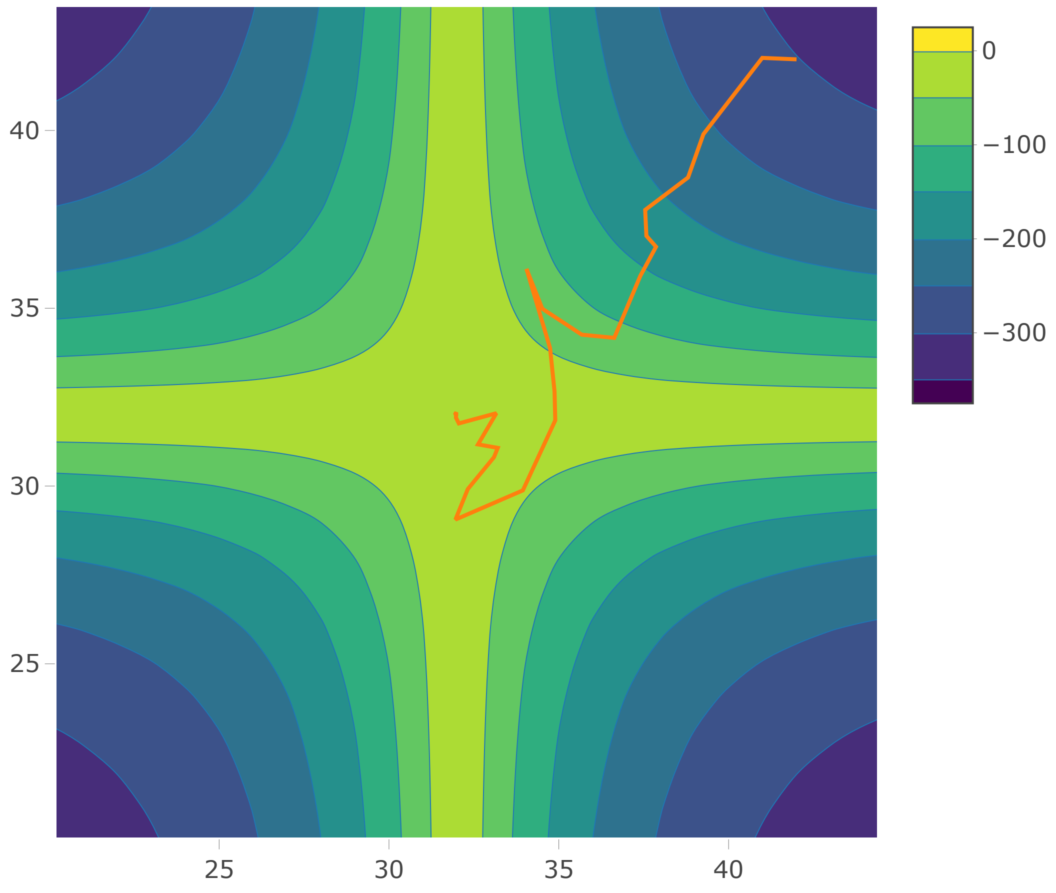
The algorithm of ELRS is similar to LRS except that instead of calculating $\theta_i = \theta_{i-1} + v$ we calculate $\theta_i = \theta_{i-1} + b_{i-1} + v$, where b_{i-1} is the additional bias term. Given that $L(\theta_i) < L(\theta_{i-1})$ the new bias term would be $b_i = 0.2b_{i-1} + 0.4v$, else we define the bias term to be $b_i = 0.5b_{i-1}$

Using the information above as well as the algoritm, we apply ELRS to the Ackley function:

```
minimize.els <- function(
  f.loss, d, n = 1e5, sig=1, startingTheta=NULL, thresh = 0.25){
  theta    = matrix( 0, nrow = n, ncol = d)
  val.loss = rep(0,n)
  b = matrix( 0, nrow = n, ncol = d)
  if( is.null(startingTheta) ){
    theta[1,] = runif( d, -32, 32)
  } else {
    theta[1,] = startingTheta
  }
  val.loss[1] = f.loss( theta[1,] )

  for( i in 2:n ){
    v = rnorm(d,0,sig);
    theta[i,] = theta[i-1,]+b[i-1,]+v
    if( prod((theta[i,]<=32)*(theta[i,]>=-32)) == 1 ){
      val.loss[i] = f.loss( theta[i,] )
      if( val.loss[i]<val.loss[i-1]-thresh ){
        b[i,] = 0.2*b[i-1,] + 0.4*v
        next
      }
    }
    theta[i,] = theta[i-1,]+b[i-1,]-v
    if( prod((theta[i,]<=32)*(theta[i,]>=-32)) == 1 ){
      val.loss[i] = f.loss( theta[i,] )
      if( val.loss[i]<val.loss[i-1] -thresh){
        b[i,] = 0.2*b[i-1,] - 0.4*v
        next
      }
    }
    b[i,] = 0.5*b[i-1,];
    theta[i,] = theta[i-1,]
    val.loss[i] = val.loss[i-1]
  }
  return(
    list(
      theta = theta,
      loss = val.loss,
      bias = b
    )
  )
}
```

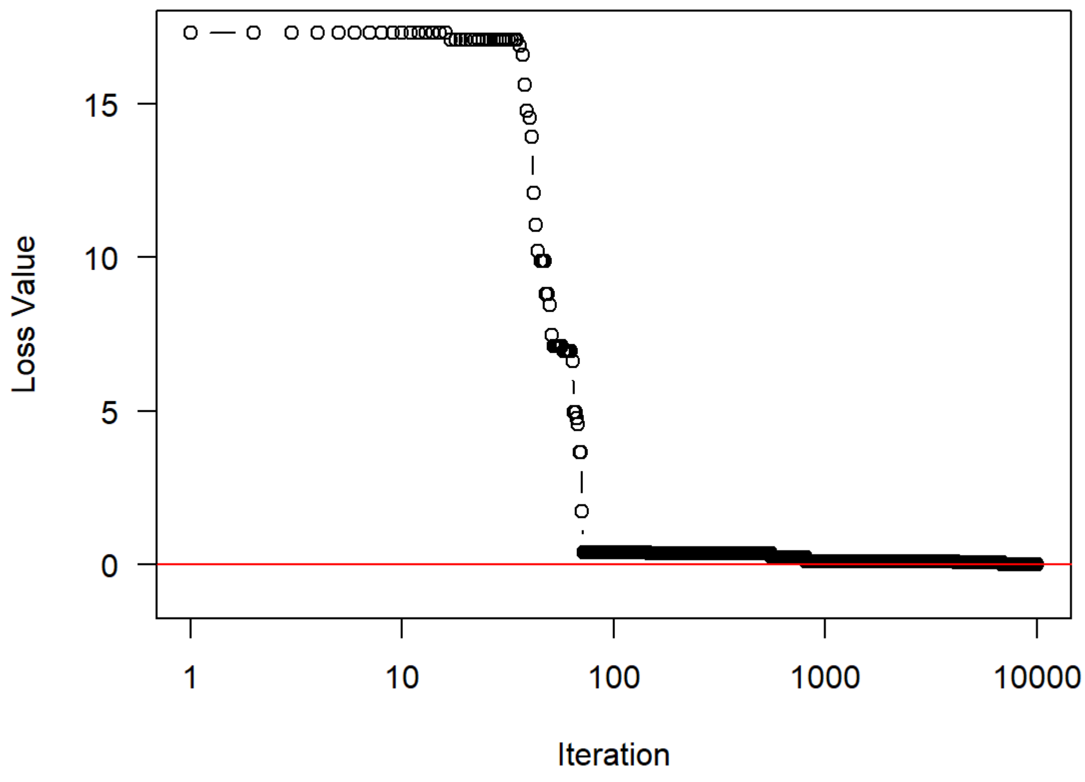
At $d = 2$

```
# for d = 2
nn = 1e4
oute1 = minimize.els(f.loss = ackley, d=2, n = nn, sig=1, startingTheta = c(10, 10), thresh = 0)

# contour plot
fig%>%add_trace(x = oute1$theta[,1]+32, y = oute1$theta[,2]+32,  type = "scatter", mode = "lin
e")
```



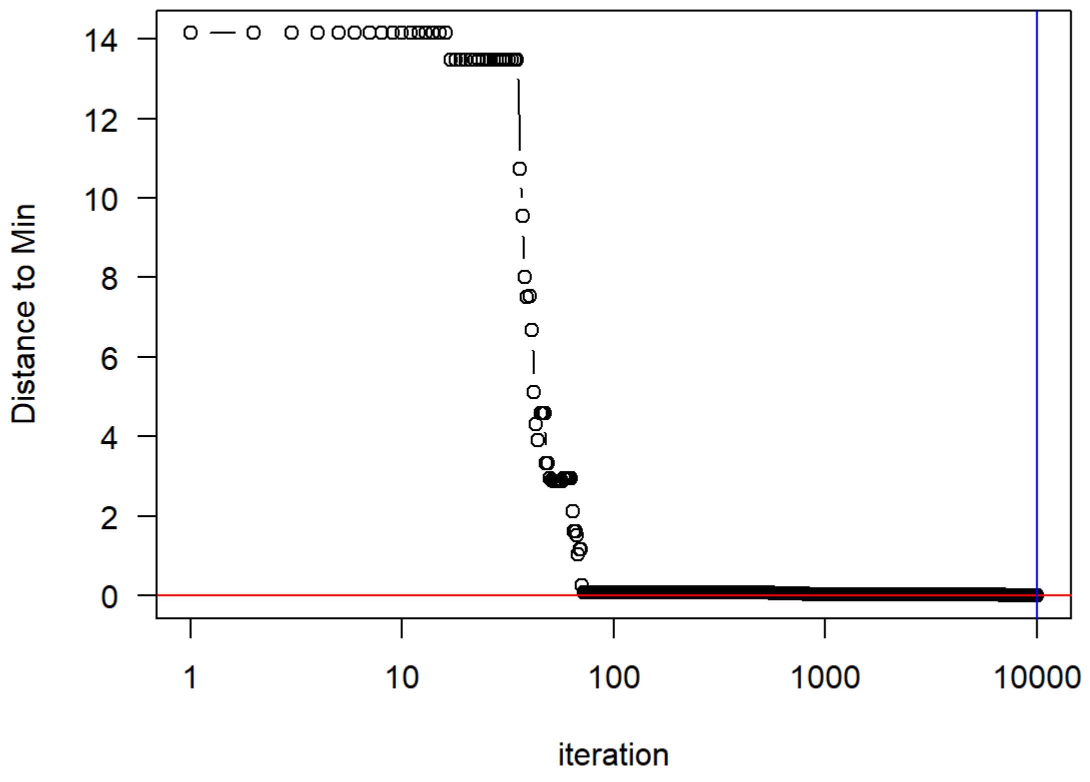
Looking at the contour plot above, we can see that it is similar to the contour plot of LRS at $d = 2$. However, we can see that the points follow a more definite direction that curves to the center of the plot.

```
plot(1:nn, oute1$loss, type = 'b', log = 'x', las = 1, ylab = "Loss Value",
     xlab = "Iteration", ylim = c(-1, max(oute1$loss)))
abline(h = 0, col = "red")
```



Looking at the loss value over each iteration plot we can see that the loss value decreases drastically after 10 iterations up to 100 iterations, and then it slowly decreases until it approaches to a value close to 0.

```
th.dist = sqrt(rowSums((oute1$theta)^2))
plot(1:nn, th.dist, type = 'b', log = 'x', las = 1, ylab = "Distance to Min", xlab = "iteratio
n", ylim = c(0, max(th.dist)))
abline(h = 0, col = "red")
abline(v = 10000, col = "blue")
```



From the plot above we can see that there is also a drastic decrease in the distance of the θ_{best} to θ^* between 10 and 100 iterations and then the distance slowly decreases until it approaches 0 where it stagnates up to 10000 iterations. The θ_{best} of some of the iterations after 100 have a smaller distance close to to 0 which indicates they are close to θ^*.

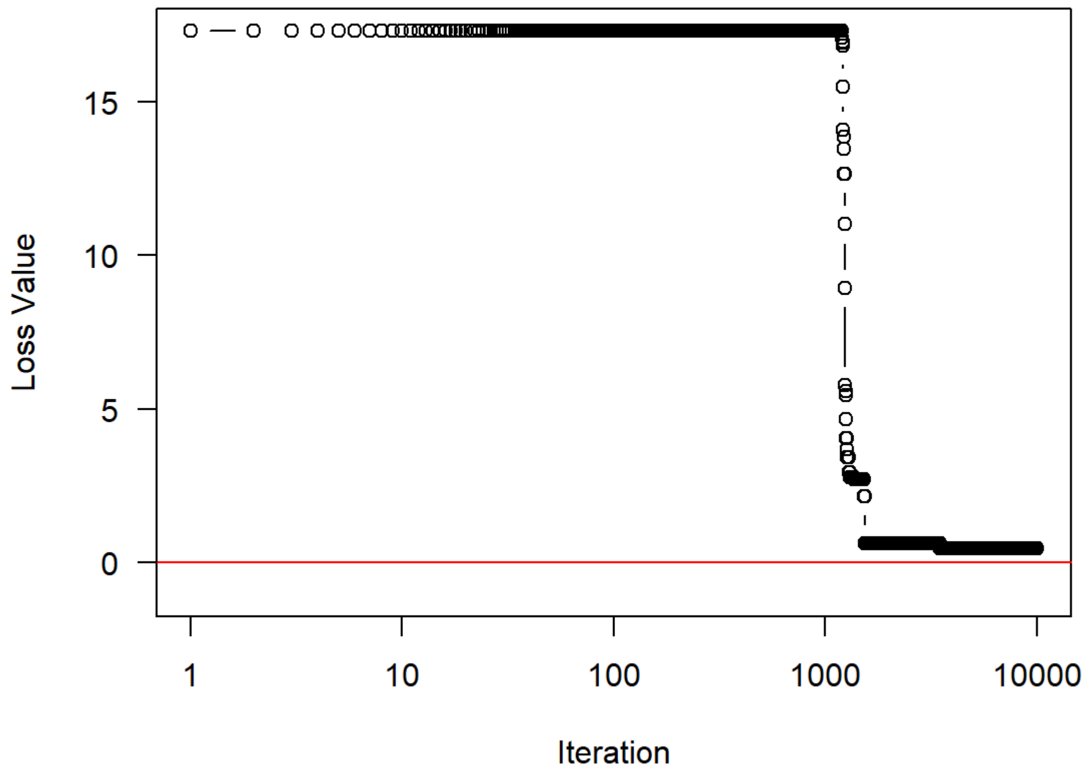
Based on the plots above, we can take the 10000th iteration to find the value of the loss function that has a distance to the minimum close to 0 indicating that $\theta_{best} \approx \theta^* = argmin_{\theta}\{L(\theta)\}$. The location of the value is also indicated by the blue vertical line in the plot above.

```
cat(" Theta Value:    ", oute1$theta[10000,1], "\n Loss Value:     ", oute1$loss[10000], "\n Dis
tance to min:", sqrt(rowSums((oute1$theta)^2))[10000])
```

```
##  Theta Value:     0.009527589
##  Loss Value:      0.03133647
##  Distance to min: 0.01011637
```

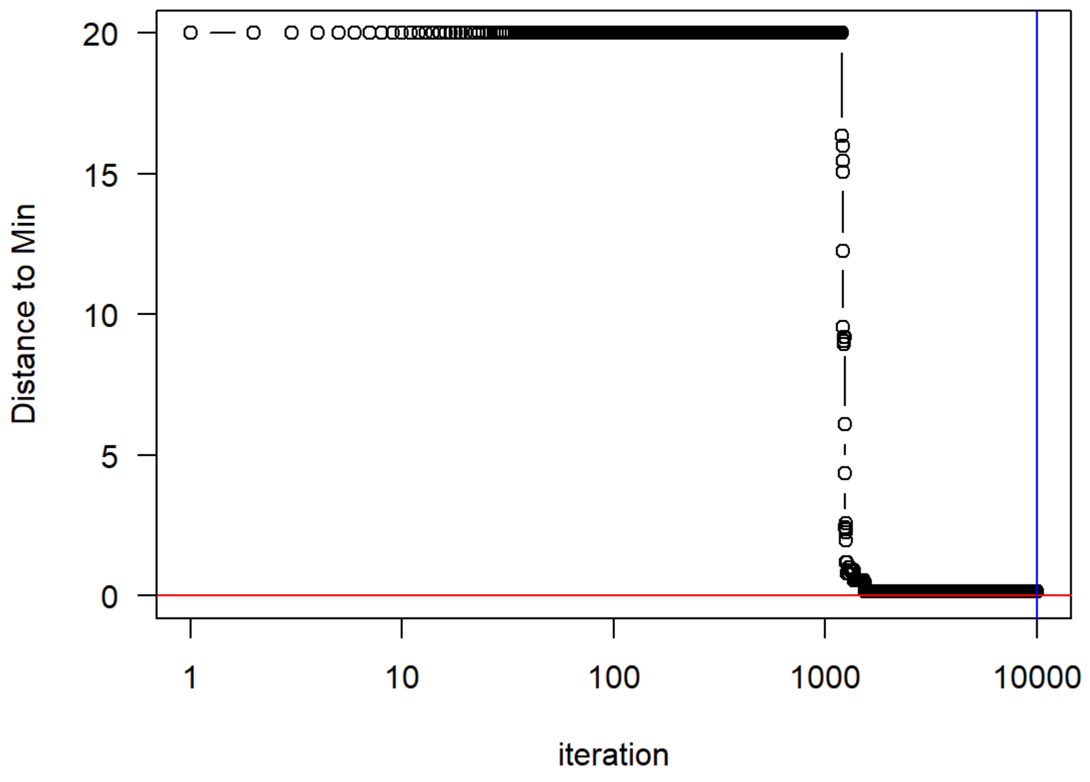
At $d = 4$

```
# for d = 4
nn = 1e4
oute2 = minimize.els(f.loss = ackley, d=4, n = nn, sig=1, startingTheta = c(10, 10, 10, 10), thr
esh = 0)

plot(1:nn, oute2$loss, type = 'b', log = 'x', las = 1, ylab = "Loss Value",
     xlab = "Iteration", ylim = c(-1, max(oute2$loss)))
abline(h = 0, col = "red")
```



Looking at the loss value over each iteration plot we can see that the loss value decreases drastically after 1000 iterations, and then it slowly decreases until it approaches to a value close but above 0. In this plot we can also see that the drastic decrease is much steeper than at $d = 2$.

```
th.dist = sqrt(rowSums((oute2$theta)^2))
plot(1:nn, th.dist, type = 'b', log = 'x', las = 1, ylab = "Distance to Min", xlab = "iteratio
n", ylim = c(0, max(th.dist)))
abline(h = 0, col = "red")
abline(v = 10000, col = "blue")
```



From the plot above we can see that there is also a drastic decrease in the distance of the θ_{best} to θ^* after 1000 iterations and then the distance slowly decreases until it approaches 0 where it stagnates up to 10000 iterations. The θ_{best} of some of the iterations after 1000 have a smaller distance close to to 0 which indicates they are close to θ^*.

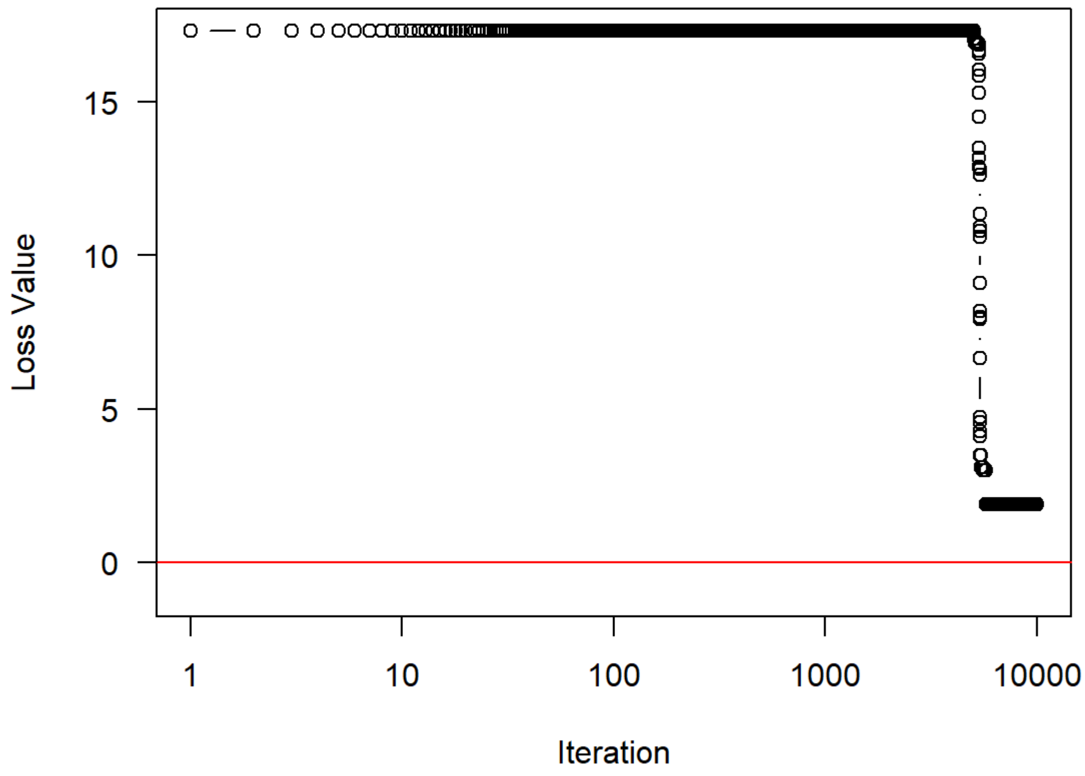
Based on the plots above, we can take the 10000th iteration to find the value of the loss function that has a distance to the minimum close to 0 indicating that $\theta_{best} \approx \theta^* = argmin_{\theta}\{L(\theta)\}$. The location of the value is also indicated by the blue vertical line in the plot above.

```
cat(" Theta Value:    ", oute2$theta[10000,1], "\n Loss Value:     ", oute2$loss[10000], "\n Dis
tance to min:", sqrt(rowSums((oute2$theta)^2))[10000])
```

```
##  Theta Value:     -0.01187215
##  Loss Value:      0.4842939
##  Distance to min: 0.1325773
```

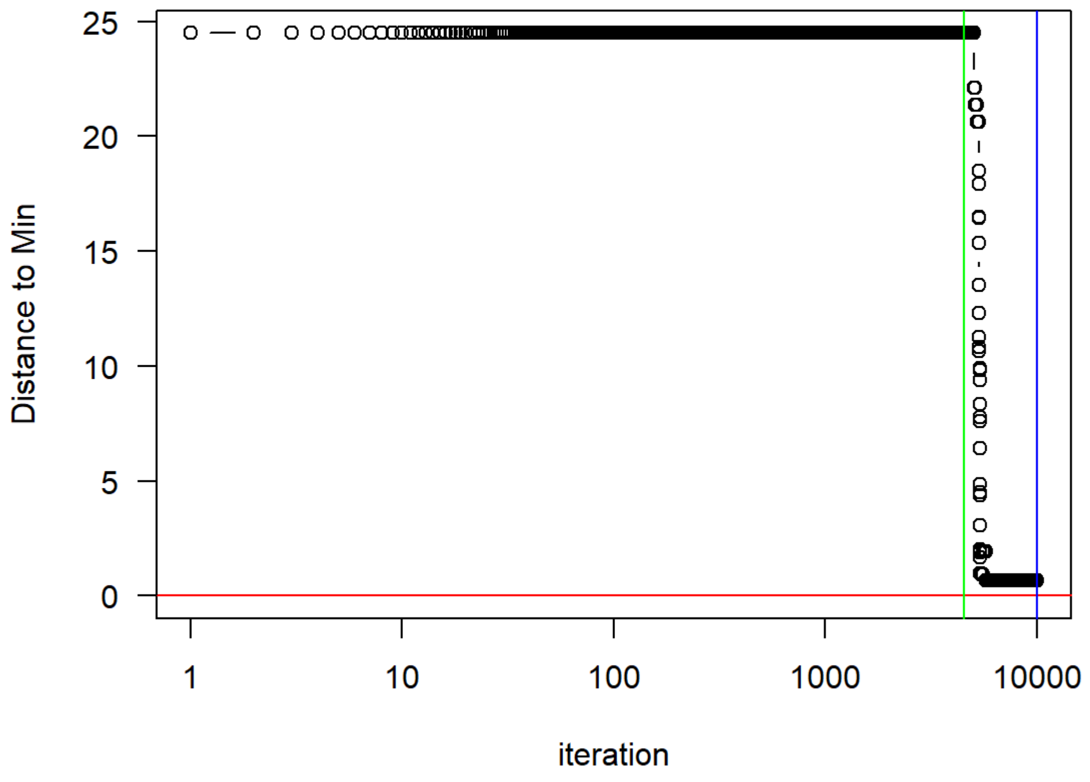
At $d = 6$

```
# for d = 6
nn = 1e4
oute3 = minimize.els(f.loss = ackley, d=6, n = nn, sig=1, startingTheta = c(10, 10, 10, 10, 10,
10), thresh = 0)

plot(1:nn, oute3$loss, type = 'b', log = 'x', las = 1, ylab = "Loss Value",
     xlab = "Iteration", ylim = c(-1, max(oute3$loss)))
abline(h = 0, col = "red")
```



Looking at the loss value over each iteration plot we can see that the loss value decreases drastically after 4500 iterations, and then it stagnates to a value between 0 and 5. In this plot we can also see that the drastic decrease is similar to $d = 4$.

```
th.dist = sqrt(rowSums((oute3$theta)^2))
plot(1:nn, th.dist, type = 'b', log = 'x', las = 1, ylab = "Distance to Min", xlab = "iteratio
n", ylim = c(0, max(th.dist)))
abline(h = 0, col = "red")
abline(v = 10000, col = "blue")
abline(v = 4500, col = "green")
```



From the plot above we can see that there is also a drastic decrease in the distance of the θ_{best} to θ^* after 4500 iterations and then the distance slowly decreases until it approaches 0 where it stagnates up to 10000 iterations (green line). The θ_{best} of some of the iterations close to 10000 have a smaller distance close to to 0 which indicates they are close to θ^*.

Based on the plots above, we can take the 1000th iteration to find the value of the loss function that has a distance to the minimum close to 0 indicating that $\theta_{best} \approx \theta^* = argmin_\theta\{L(\theta)\}$. The location of the value is also indicated by the blue vertical line in the plot above.

```
cat(" Theta Value:    ", oute3$theta[10000,1], "\n Loss Value:     ", oute3$loss[10000], "\n Dis
tance to min:", sqrt(rowSums((oute3$theta)^2))[10000])
```

```
##  Theta Value:     -0.07589501
##  Loss Value:      1.896833
##  Distance to min: 0.6511634
```

Conclusion

Comparing all the methods used for the Akley function at $d = 2, 4, 6$ we note: At $d = 2$ using LRS and ELRS we got similar contour plots, but we got a more refined contour plot for ELRS which is expected and explained by the additional bias term. Comparing the following plots from these three methods at $d = 2$ we also found that all the plots comparing loss values or distance values over the iterations had a drastic decrease around the area of iteration 10 to 100. In terms of the values of the loss function and theta at the smallest distance to the min found by each algorithm at $d = 2$, LRS had the smallest value of θ_{best} at 0.002254553 resulting in a loss value of

$L(\theta_{best}) = 0.007016738$ and a distance to minimum of 0.002425412, all which are close to zero. ELRS at this dimension resulted in slightly higher values for theta, the loss function at the found theta, and the distance to minimum.

Comparing the plots from these three methods at $d = 4$ we also found that all the plots comparing loss values or distance values over the iterations had a drastic decrease around the area of iteration 100 to 1000 for LRS and ELRS. For SRS we have a decreasing step pattern that is less drastic compared to the othe plots of the other methods at $d = 4$. In terms of the values of the loss function and theta at the smallest distance to the min found by each algorithm at $d = 4$, LRS also had the smallest value of θ_{best} at -0.07021837 resulting in a loss value of $L(\theta_{best}) = 0.738826$ and a distance to minimum of 0.1796002, all which are close to zero. ELRS at this dimension resulted in slightly higher values for theta, while SRS resulted in a higher value for the loss function at the found theta, with the smallest distance to minimum found at $d = 4$ of 0.075669215. We also note that all theta values found by these three methods at this d were all negative.

Comparing the plots from these three methods at $d = 6$ we also found that all the plots comparing loss values or distance values over the iterations had a drastic decrease around the area of iteration 1000 to 10000 for LRS and ELRS. For SRS we have a decreasing step pattern that varies more in contrast to the SRS plots at $d = 4$. In terms of the values of the loss function and theta at the smallest distance to the min found by each algorithm at $d = 6$, ELRS had the smallest value of θ_{best} at -0.07589501 resulting in a loss value of $L(\theta_{best}) = 1.896833$ and a distance to minimum of 0.6511634, all which are close to zero. LRS at this dimension resulted in slightly higher values for theta, while SRS resulted in a higher value for the loss function at the found theta, with the smallest distance to minimum found at $d = 6$.

We note that within the methods, as d increased for SRS the plots had more variation, the value of θ_{best} that would result in the smallest distance to min value would decrease, the value of the loss function would increase, and the distance to minimum would not have an evident increasing or decreasing pattern as d increased. For LRS, as d increased the plots would show a drastic decrease in loss value and distance to minimum in later iterations, the distance to minimum would increase, and the loss value would also increase. There was no evident pattern for the value of theta for increasing d in LRS. Lastly, for ELRS, similar to the other methods, as d increased the distance to minimum value would increase along with the loss value. The value of theta also decreased to negative values after $d = 4$.

2 Modeling the Nile River

In the R dataset package, there is the Nile dataset, which tracks the annual flow of the Nile river over 100 years. Use the Robbins-Monro Algorithm to model the flow using a linear combination of radical basis functions. That is, for $x \in [1, 100]$, let

$$f_{\theta,\sigma}(x) = \sum_{i=1}^{k} \theta_i exp(-\sigma_i|x - \frac{100_i}{k+1}|)$$

The parameters to optimize over are $_i,…, _k $ and $\sigma_1, \ldots, \sigma_k$. Try optimizing with two different loss functions: the squared error and the absolute error,

$L_2(\theta, \sigma) = \frac{1}{2n} \sum_{j=1}^{100} (y_j - f_{\theta,\sigma}(x_j))^2$ and $L_1(\theta, \sigma) = \frac{1}{n} \sum_{j=1}^{100} |y_j - f_{\theta,\sigma}(x_j)|$,

Where $x_j = j$ is the jth year and y_j is the jth flow measurement. You can pick the number of basis functions k as you like, but note that there will be $2k$ parameters to optimize over.

Robins-Monro Algorithm

The Robbins-Monro algorithm is a gradient method similar to Newton-Raphson algorithm. This method particularly requires the use of step-sizes ($gain$ sequences) such that $\sum_{k=1}^{\infty} a_k$ and $\sum_{k=1}^{\infty} a_k^2 < \infty$, allowing us to explore a space Θ.

Being a gradient method, we assume we have a loss function $L(\theta) = E[Q(\theta, \sigma)]$ where Q is the loss function with noise, and $y(\theta)$ is a measurement of the gradient of Q with noise.

Gradient Functions

As the Robbins-Monro Algoritm requires the selection of a step size a_k and to update out parameter vector by $\theta_k \leftarrow \theta_{k-1} - a_k y(\theta_k - 1)$ with $y(\theta_k - 1)$ being a noisy estimation of the gradient, we first find the gradient functions for the loss functions. This can be done by taking the derivative of each loss function:

Gradient of loss function 1, given $n = 100$ and a fixed σ:

$$\nabla L_1(\theta, \sigma) = \frac{\partial}{\partial \theta_i} \frac{1}{n} \sum_{j=1}^{100} |y_j - f_{\theta,\sigma}(x_j)|$$

$$= \frac{1}{n} \sum_{j=1}^{n} \frac{\partial}{\partial \theta_i} |y_j - f_{\theta,\sigma}(x_j)|$$

$$= \frac{1}{n} \sum_{j=1}^{n} (\frac{y_j - f_{\theta,\sigma}(x_j)}{|y_j - f_{\theta,\sigma}(x_j)|} \frac{\partial}{\partial \theta_i} (y_j - f_{\theta,\sigma}(x_j))$$

$$= \frac{1}{n} \sum_{j=1}^{n} (\frac{y_j - \sum_{i=1}^{k} \theta_i exp(-\sigma_i|x_j - \frac{100_i}{k+1}|)}{|y_j - \sum_{i=1}^{k} \theta_i exp(-\sigma_i|x_j - \frac{100_i}{k+1}|)|} \frac{\partial}{\partial \theta_i} (y_j - \sum_{i=1}^{k} \theta_i exp(-\sigma_i|x_j - \frac{100_i}{k+1}|))$$

$$= \frac{1}{n} \sum_{j=1}^{n} (\frac{y_j - \sum_{i=1}^{k} \theta_i exp(-\sigma_i|x_j - \frac{100_i}{k+1}|)}{|y_j - \sum_{i=1}^{k} \theta_i exp(-\sigma_i|x_j - \frac{100_i}{k+1}|)|} (-\sum_{i=1}^{k} exp(-\sigma_i|x_j - \frac{100_i}{k+1}|))$$

$$= -\frac{1}{n} \sum_{j=1}^{n} \frac{y_j - \sum_{i=1}^{k} \theta_i exp(-\sigma_i|x_j - \frac{100_i}{k+1}|)}{|y_j - \sum_{i=1}^{k} \theta_i exp(-\sigma_i|x_j - \frac{100_i}{k+1}|)|} \sum_{i=1}^{k} exp(-\sigma_i|x_j - \frac{100_i}{k+1}|)$$

Thus,

$$gradient\ 1 = \nabla L_1(\theta,\sigma) = -\frac{1}{n}\sum_{j=1}^{n}\frac{y_j - \sum_{i=1}^{k}\theta_i exp(-\sigma_i|x_j - \frac{100_i}{k+1}|)}{|y_j - \sum_{i=1}^{k}\theta_i exp(-\sigma_i|x_j - \frac{100_i}{k+1}|)|}\sum_{i=1}^{k} exp(-\sigma_i|x_j - \frac{100_i}{k+1}|) + \epsilon$$

Gradient of loss function 2, given $n = 100$ and a fixed σ:

$$\nabla L_2(\theta,\sigma) = \frac{\partial}{\partial\theta_i}\frac{1}{2n}\sum_{j=1}^{100}(y_j - f_{\theta,\sigma}(x_j))^2$$

$$= \frac{1}{2n}\sum_{j=1}^{n}\frac{\partial}{\partial\theta_i}(y_j - f_{\theta,\sigma}(x_j))^2$$

$$= \frac{1}{2n}\sum_{j=1}^{n} 2(y_j - f_{\theta,\sigma}(x_j))\frac{\partial}{\partial\theta_i}(y_j - f_{\theta,\sigma}(x_j))$$

$$= \frac{1}{2n}\sum_{j=1}^{n} 2(y_j - \sum_{i=1}^{k}\theta_i exp(-\sigma_i|x_j - \frac{100_i}{k+1}|))\frac{\partial}{\partial\theta_i}(y_j - \sum_{i=1}^{k}\theta_i exp(-\sigma_i|x_j - \frac{100_i}{k+1}|))$$

$$= \frac{1}{2n}\sum_{j=1}^{n} 2(y_j - \sum_{i=1}^{k}\theta_i exp(-\sigma_i|x_j - \frac{100_i}{k+1}|))(-\sum_{i=1}^{k} exp(-\sigma_i|x_j - \frac{100_i}{k+1}|))$$

$$= -\frac{2}{2n}\sum_{j=1}^{n}(y_j - \sum_{i=1}^{k}\theta_i exp(-\sigma_i|x_j - \frac{100_i}{k+1}|))\sum_{i=1}^{k} exp(\sigma_i|x_j - \frac{100_i}{k+1}|)$$

$$= -\frac{1}{n}\sum_{j=1}^{n}(y_j - \sum_{i=1}^{k}\theta_i exp(-\sigma_i|x_j - \frac{100_i}{k+1}|))\sum_{i=1}^{k} exp(\sigma_i|x_j - \frac{100_i}{k+1}|)$$

Thus,

$$gradient\ 2 = \nabla L_1(\theta,\sigma) = -\frac{1}{n}\sum_{j=1}^{n}(y_j - \sum_{i=1}^{k}\theta_i exp(-\sigma_i|x_j - \frac{100_i}{k+1}|))\sum_{i=1}^{k} exp(\sigma_i|x_j - \frac{100_i}{k+1}|) + \epsilon$$

We note above that the gradients resemble means as the equation for the mean is defined as $mean = \frac{1}{n}\sum_{j=1}^{n}\frac{\partial Q(\theta,\sigma)}{\partial\theta}$ with $\frac{\partial Q(\theta,\sigma)}{\partial\theta}$ being $-\frac{y_j - \sum_{i=1}^{k}\theta_i exp(-\sigma_i|x_j - \frac{100_i}{k+1}|)}{|y_j - \sum_{i=1}^{k}\theta_i exp(-\sigma_i|x_j - \frac{100_i}{k+1}|)|}\sum_{i=1}^{k} exp(-\sigma_i|x_j - \frac{100_i}{k+1}|)$ for gradient 1 and $-(y_j - \sum_{i=1}^{k}\theta_i exp(-\sigma_i|x_j - \frac{100_i}{k+1}|))\sum_{i=1}^{k} exp(\sigma_i|x_j - \frac{100_i}{k+1}|)$ for gradient 2.

Code of the Robbins-Monro, loss, and gradient functions.

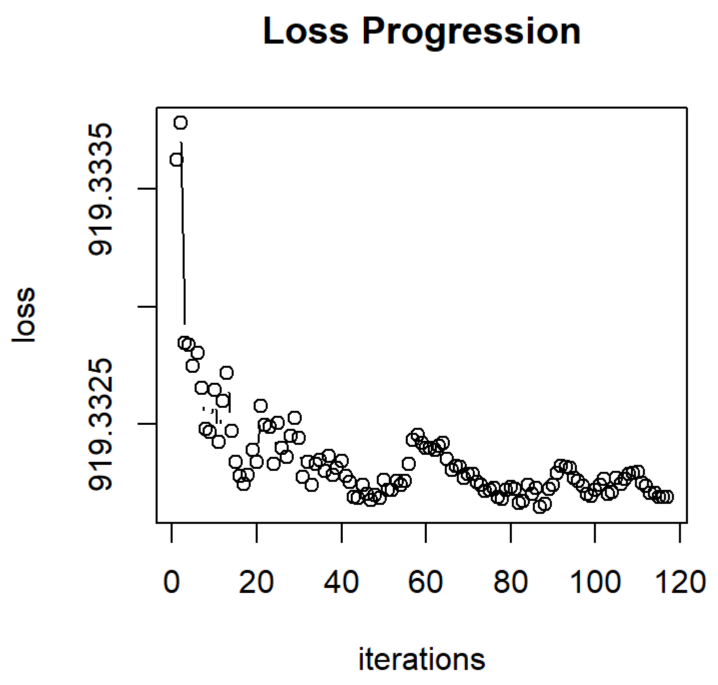
```r
library(datasets)
# Global variable for noise
sig = 1

# back function for gradients
g.exp = function(theta, valid = F){
  k = length(theta)
  x = 1:100
  sigma = 1 # sigma is fixed
  if (valid == T){
    thetav = theta[1]*exp(-sigma*abs(x-(100/(k+1))))
    for (i in 2:k){
      thetav = thetav + theta[i]*exp(-sigma*abs(x-(100*i/(k+1))))
    }
    return(thetav)
  }
  else{
    expv = exp(-sigma*abs(x-(100/(k+1))))
    for (i in 2:k){
      expv = expv + exp(-sigma*abs(x-(100*i/(k+1))))
    }
    return(expv)
  }
}
# Loss function 1
loss1 = function(th){
  al1 = mean(abs(Nile-g.exp(th, valid = T)))
  return(al1)
}

# Loss function 2
loss2 = function(th){
  sq = (Nile-g.exp(th, valid = T))^2
  al2 = (1/2)*mean(sq)
  return(al2)
}


# gradient of loss function 1
grad.rad1 = function(th){
  dd = length(th)
  main1 = (Nile - g.exp(th ,valid = T))*(g.exp(th, valid = F))/abs((Nile - g.exp(th ,valid = 
T)))
  gv1 = -mean(main1)+ rnorm(dd,0,sig)
  return(gv1)
}

# gradient of loss function 2
grad.rad2 = function(th){
  dd = length(th)
  main2 = (Nile - g.exp(th ,valid = T))*(g.exp(th, valid = F))
  gv2 = -mean(main2)+ rnorm(dd,0,1000*sig)
  return(gv2)
}

# Robbins-Monro main function
robbinsMonro <- function(grad.f, d=2, max.steps=100, step.size = 1/(1:max.steps),
                         loss.f = NULL, startTheta = NULL, thresh=1e-4){
  th = matrix(0,max.steps,d); #save theta values
  gr = matrix(0,max.steps,d); #save gradients
  ls = rep(0,max.steps); # save loss values
  if( is.null(startTheta) ){
    th[1,] <- rnorm( d,0,1 );
  } else {
    th[1,] <- startTheta;
  }
  gr[1,] <- grad.f( th[1,] );
  if( !is.null(loss.f) )  # if loss function is given
    ls[1] <- loss.f( th[1,] );
  for( i in 2:max.steps ){
    th[i,] <- th[i-1,] - step.size[i]*gr[i-1,];
    gr[i,] <- grad.f( th[i,] )
    if( !is.null(loss.f) )
      ls[i] <- loss.f(th[i,])
    if( sum( (th[i,]-th[i-1,])^2 ) < thresh^2 )
      break;
  }
  return(list(
    theta = th[1:i,],
    grad  = gr[1:i,],
    loss  = ls[1:i]
  ))
}
```

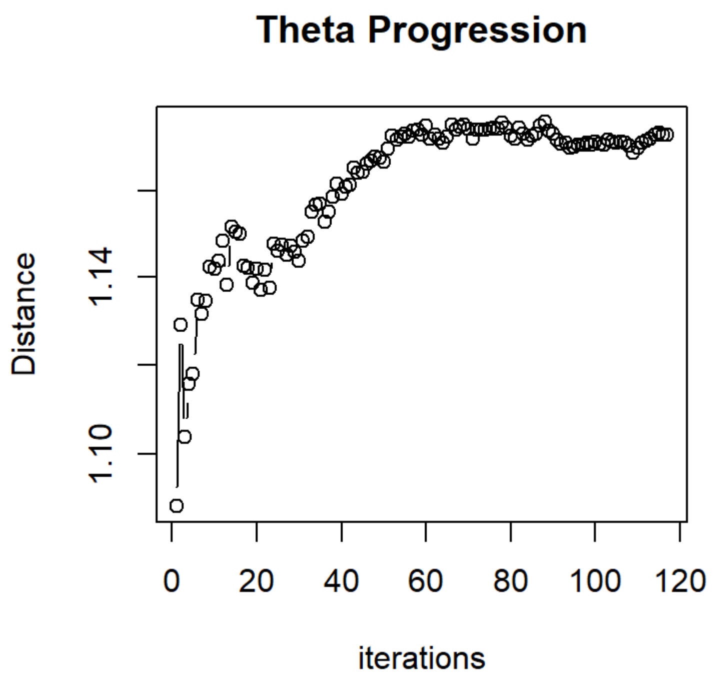
We try $k = 2, 4, 6$ basis functions

At d = 2

Loss and Gradient function 1

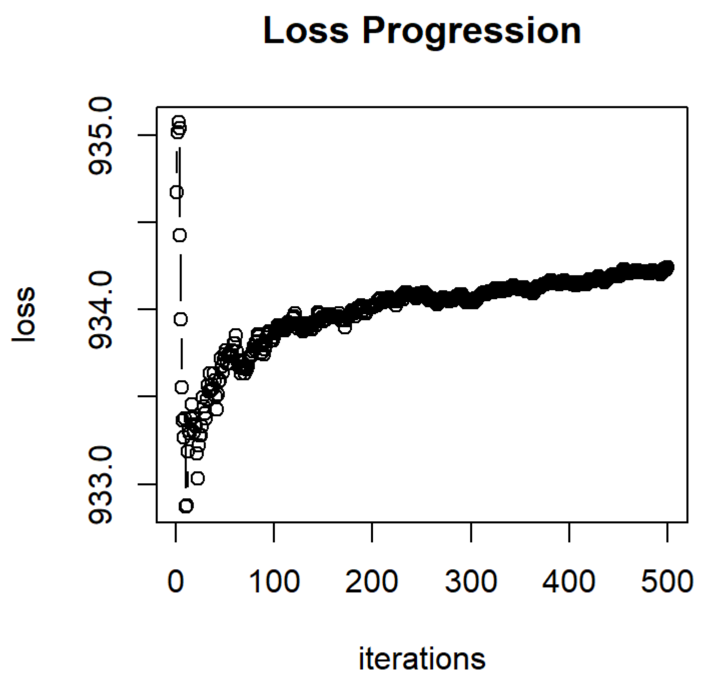
```r
# Loss and gradient function 1 at k=2
l1k2 = robbinsMonro(grad.f = grad.rad1, d=2, max.steps = 500, step.size = 0.1/(1:1000),
                         loss.f = loss1, startTheta = NULL, thresh=1e-4)

plot(
  l1k2$loss,type='b', log='y',
  main = "Loss Progression",xlab="iterations",ylab="loss"
)

plot(
  sqrt(rowSums(l1k2$theta^2)),type='b', log='y',
  main = "Theta Progression",xlab="iterations",ylab="Distance"
)
```

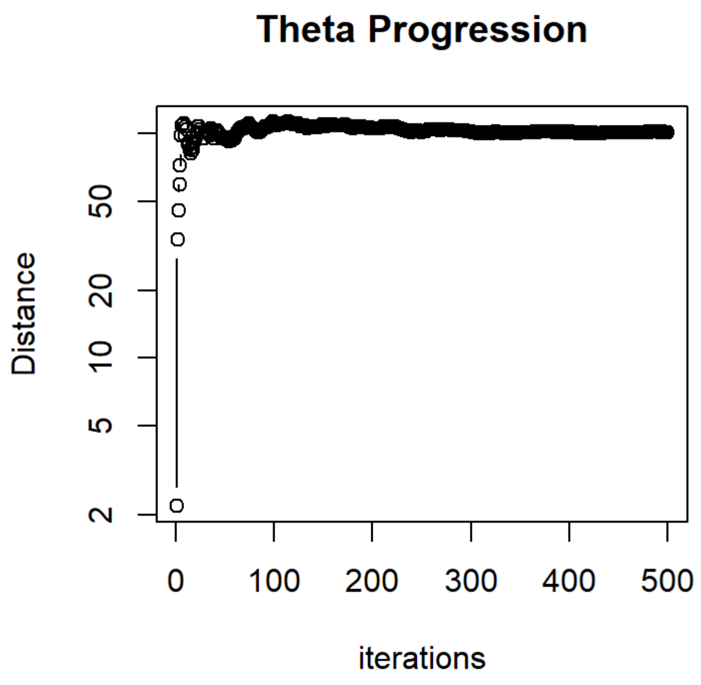




We can see that using the Robins-Monro algorithm on the first loss and gradient function at $d = 2$ to model the flow of the Nile over 100 years results in the plots above. From the loss progression plot we can see that as the number of iterations increases, the value of the loss decreases, and the loss function has an inverse relationship with the progression of theta as the theta progression increases with the number of iterations. We can also see that the number of iterations is around 120.

```
# Loss and gradient function 2 at k=2
l2k2 = robbinsMonro(grad.f = grad.rad2, d=2, max.steps = 500, step.size = 0.1/(1:1000),
                         loss.f = loss2, startTheta = NULL, thresh=1e-4)

plot(sqrt(l2k2$loss*2),type='b', log='y', main = "Loss Progression",xlab="iterations",ylab="los
s")

plot(sqrt(rowSums(l2k2$theta^2)),type='b', log='y',
     main = "Theta Progression", xlab="iterations", ylab="Distance")
```



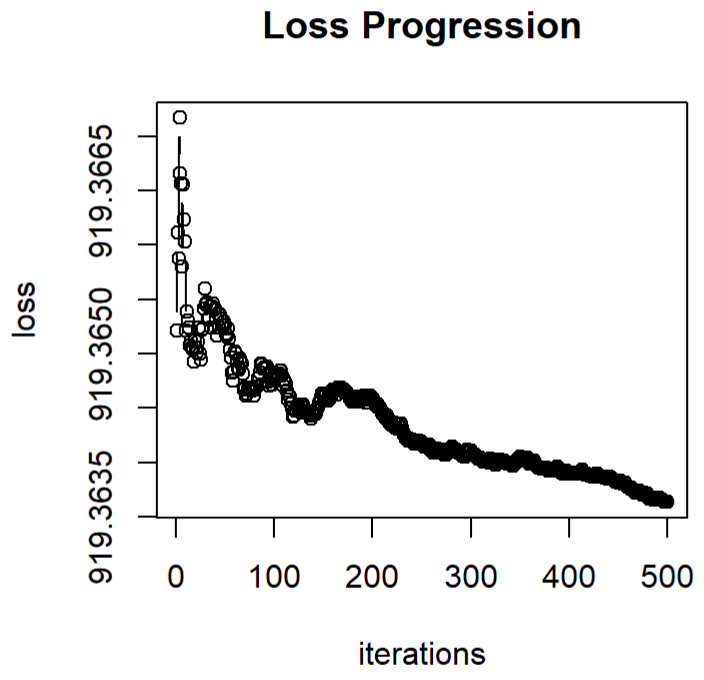


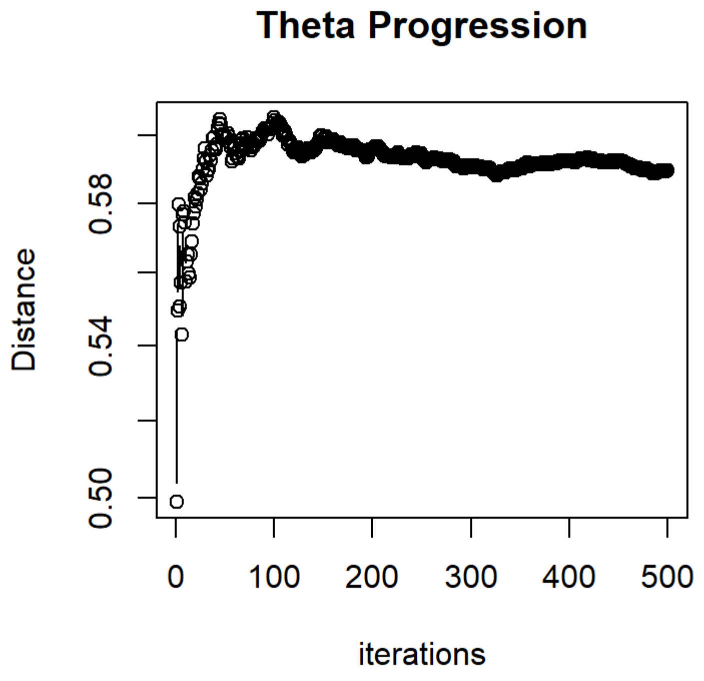
We can see that using the Robins-Monro algorithm on the second loss and gradient function at $d = 2$ to model the flow of the Nile over 100 years results in the plots above. Initially, from the loss progression plot we could see that the value of the loss approached to a value around 436500 and the scale ranged from 400000 to 500000. Furthermore, we proceeded to to a transformation on the loss values to match the scale of the previous loss progression plot. The loss projection plot shows initially a deastic decrease, and then after 100 iterations we can see a slowly decreasing trend around 934. We can see that the loss progression shows an inverse relationship with the theta progression since the theta progression plot starts with a steep increase in diantance on the first iteration and then the values gradually slowly decrease around a distance of 60. Note that there was no scaling performed on the plot of the theta progression.

The format of these plots will be performed for the other cases below for the second set of functions.

At $d = 4$

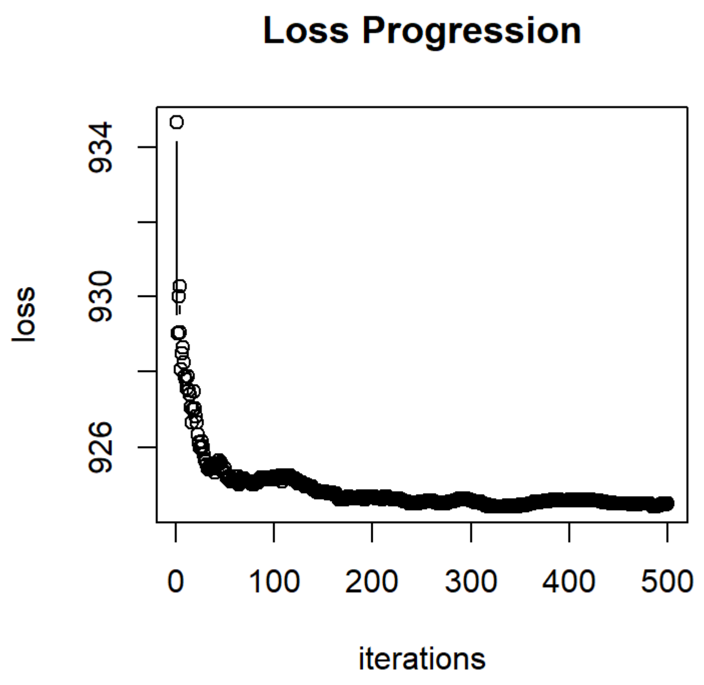
```
# Loss and gradient function 1 at k=4
l1k4 = robbinsMonro(grad.f = grad.rad1, d=4, max.steps = 500, step.size = 0.1/(1:1000),
                         loss.f = loss1, startTheta = NULL, thresh=1e-4)

plot(l1k4$loss,type='b', log='y', main = "Loss Progression",xlab="iterations",ylab="loss")

plot(sqrt(rowSums(l1k4$theta^2)),type='b', log='y', main = "Theta Progression", xlab="iteration
s", ylab="Distance")
```

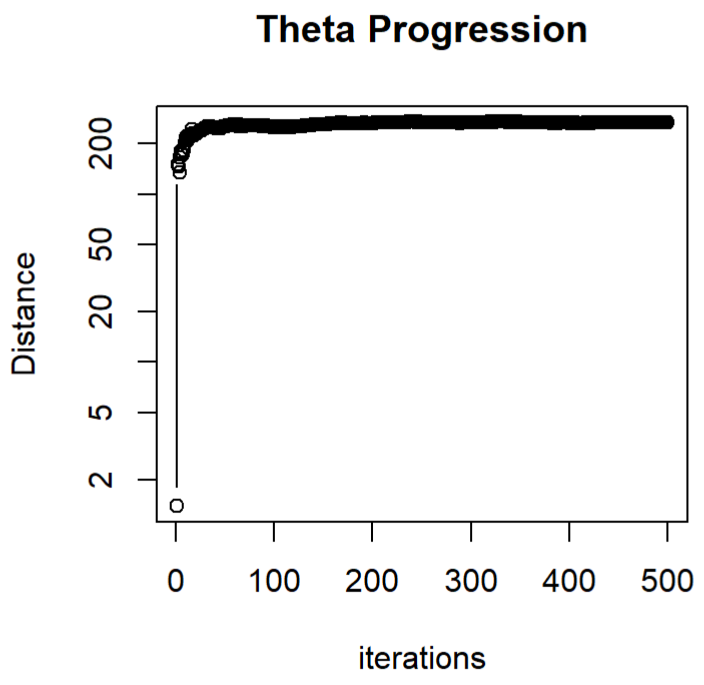




We can see that using the Robins-Monro algorithm on the first loss and gradient function at $d = 4$ to model the flow of the Nile over 100 years results in the plots above. From the loss progression plot we can see that as the number of iterations increases, the value of the loss decreases up to a value of 919.3635. The plot of the theta progression shows that distance increases in the first iteration, but then between 100 and 200 iterations it starts to gradually decrease. Between 100 and 200 iterations we can also see that the loss progression also shows a more gradual decrease until the last iteration approaches value close to 919.3635

```
# Loss and gradient function 2 at k=4
l2k4 = robbinsMonro(grad.f = grad.rad2, d=4, max.steps = 500, step.size = 0.1/(1:1000),
                         loss.f = loss2, startTheta = NULL, thresh=1e-4)

plot(sqrt(l2k4$loss*2),type='b', log='y', main = "Loss Progression",xlab="iterations",ylab="los
s")

plot(sqrt(rowSums(l2k4$theta^2)),type='b', log='y', main = "Theta Progression", xlab="iteration
s", ylab="Distance")
```

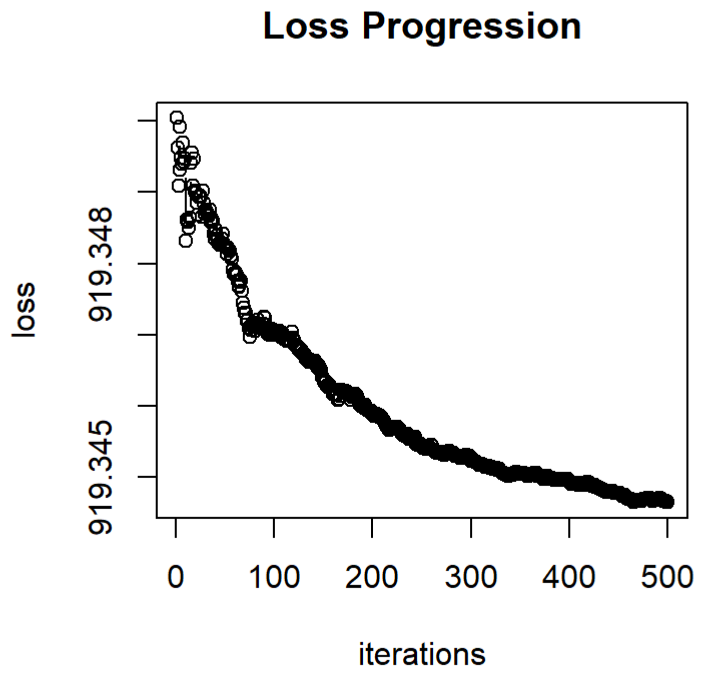


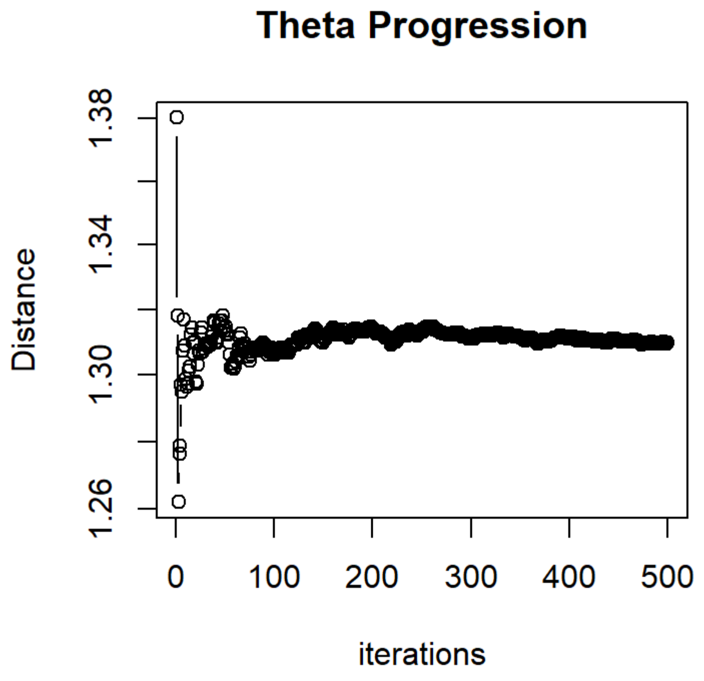


We can see that using the Robins-Monro algorithm on the second loss and gradient function at $d = 4$ to model the flow of the Nile over 100 years results in the plots above. From the loss progression plot we can see that once again as the number of iterations increases, the value of the loss decreases until it reaches a value below 926, with a steep decrease in the first iterations between itheration 1 and 100, and a slower gradual decrease after 100 iterations. Looking at the progression of theta plot, we can also see that similar to the progression of theta plot at $d = 2$ where we have a rapid increase in distance on the first iterations until we reach a distance where most of the iterations stagnate which we can observe here is around 200.

At $d = 6$

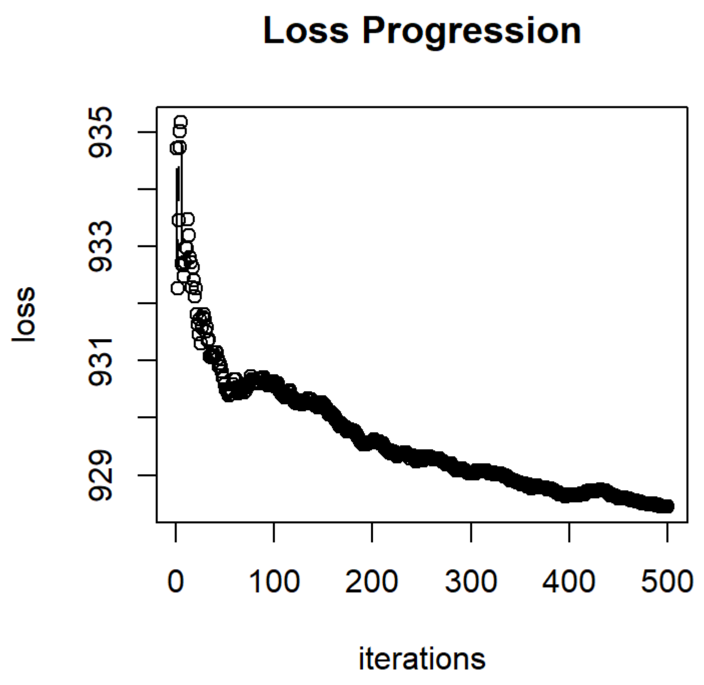
```r
# Loss and gradient function 1 at k=6
l1k6 = robbinsMonro(grad.f = grad.rad1, d=6, max.steps = 500, step.size = 0.1/(1:1000),
                         loss.f = loss1, startTheta = NULL, thresh=1e-4)

plot(l1k6$loss,type='b', log='y', main = "Loss Progression",xlab="iterations",ylab="loss")

plot(sqrt(rowSums(l1k6$theta^2)),type='b', log='y', main = "Theta Progression", xlab="iteration
s", ylab="Distance")
```

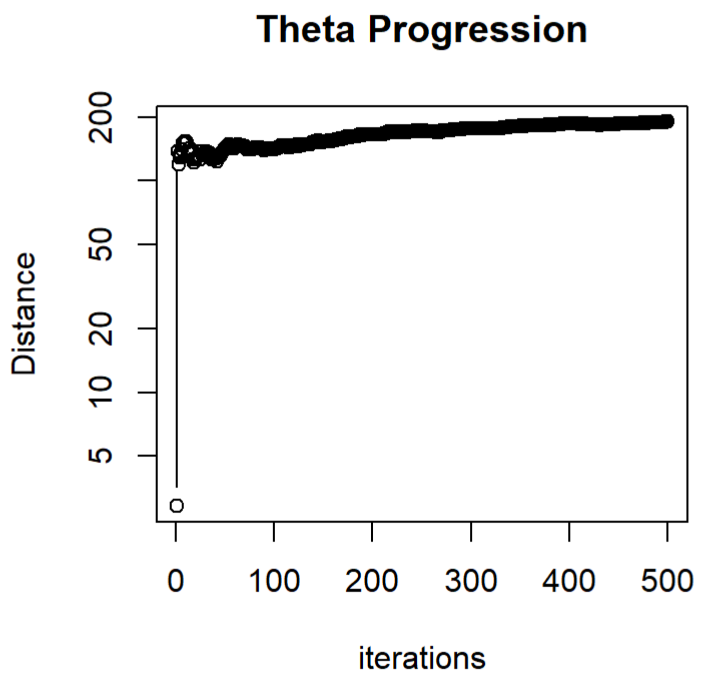




We can see that using the Robins-Monro algorithm on the first loss and gradient function at $d = 6$ to model the flow of the Nile over 100 years results in the plots above. From the loss progression plot we can see that once again as the number of iterations increases, the value of the loss decreases, with a steep decrease in the first iterations, and a more gradual decrease after 100 iterations. Looking at the progression of theta plot, we can also see that during the first iterations there is a great variation in distance, but after 100 iterations the distance of each iteration stagnates around a value between 1.3 and 1.32.

```
# Loss and gradient function 2 at k=6
l2k6 = robbinsMonro(grad.f = grad.rad2, d=6, max.steps = 500, step.size = 0.1/(1:1000),
                         loss.f = loss2, startTheta = NULL, thresh=1e-4)

plot(sqrt(l2k6$loss*2),type='b', log='y', main = "Loss Progression",xlab="iterations",ylab="los
s")

plot(sqrt(rowSums(l2k6$theta^2)),type='b', log='y', main = "Theta Progression", xlab="iteration
s", ylab="Distance")
```





We can see that using the Robins-Monro algorithm on the second loss and gradient function at $d = 6$ to model the flow of the Nile over 100 years results in the plots above. From the loss progression plot we can see that once again as the number of iterations increases, the value of the loss decreases, with a steep decrease in the first iterations, and a more gradual decrease after 100 iterations. We can see that after 500 iterations the loss value less than 929 which is consistent with the findings above. Looking at the progression of theta plot, we can also see that during the first iterations there is a steep increase in the first iterations, but then the following iterations stagnate to a value around 100 and 200.

Conclusion

From the plots above we first noted that the scale of the plots for the loss progression and theta progression using the second loss and gradient functions was significantly larger than the scale of the plots for the loss progression and theta progression using the first loss and gradient functions. Using the second set of equations, the loss progression plot had values around 4000000 to 5000000 which we decided to transform to match the scale of the loss progression plot for the first set of functions, and we decided to keep the scale of the theta progression plot as it's transformation was more complex. The scale of the theta progression plot for the second set of functions has distance values that range from 0 to 200. Using the first set of equations on the other hand, the loss progression plot has values around 919, and the theta progression plot has distance values that range around 1 to 1.2. This scale pattern is consistent through all values of d. The scale of the loss progression plot for the second set of functions after scaling did show a slightly larger range of values from 915 to 935 which captures values around 919 found by the first set of functions.

We can also see that as we increase d, the theta progression plot shows that as the number of iterations increase the distance starts to stagnate around a particular value which is more evident at $d = 6$ where we can see the points form a horizontal line between 1.3 to 1.32 for the theta progression plot using the first loss and gradient functions. We also see a similar pattern for the theta progression plots for the second set of functions as d increases, at the start the distance abruptly increases and then it varies slightly until the iterations stagnate around a value between 100 and 200.

Comparing both sets of loss and gradient functions, we can see that the absolute error loss function and its gradient show a theta progression with smaller distances and loss values that are around the flow measurements. The squared error and its gradient needed a transformation to match the range of values of the loss function to those found by the first set of functions that re around the flow measurements.The parrern of the distance values on the theta progression plots obtained from both sets of equations at each d were consistent and showed a similar pattern.We do note that when using the first set of functions \$ L_1(,)=*{j=1}^{100}|y_j - f*{, }(x_j)|\$ and $gradient\ 1 = -\frac{1}{n}\sum_{j=1}^{n} \frac{y_j - \sum_{i=1}^{k} \theta_i exp(-\sigma_i|x_j - \frac{100_i}{k+1}|)}{|y_j - \sum_{i=1}^{k} \theta_i exp(-\sigma_i|x_j - \frac{100_i}{k+1}|)|} \sum_{i=1}^{k} exp(-\sigma_i|x_j - \frac{100_i}{k+1}|) + \epsilon$ we get a smaller range of values for the loss and distance.