Genetic Algorithms

Nik Bear Brown

In this lesson we'll learn the theory behind using genetic algorithms as an optimization and search technique.

Additional packages needed

To run the code you may need additional packages.

- If necessary install the followings packages.

```
install.packages("ggplot2");
install.packages("genalg");
install.packages("GA");
```

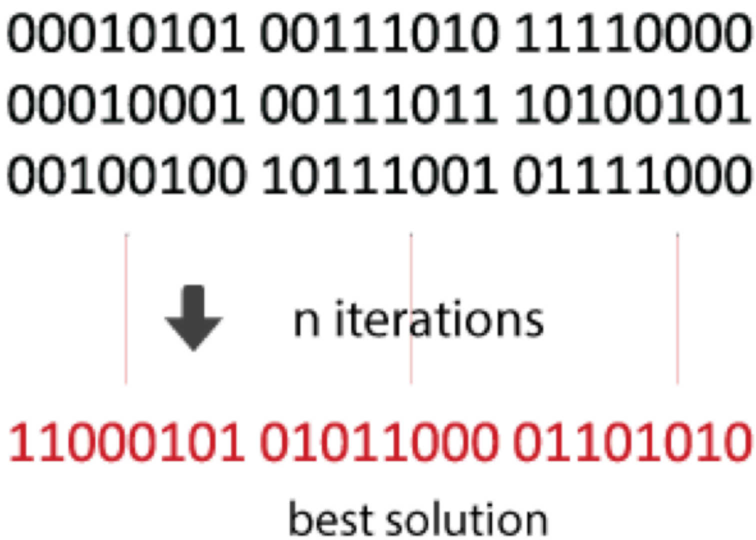
```
require(ggplot2)

## Loading required package: ggplot2

require(genalg)

## Loading required package: genalg

require(GA)

## Loading required package: GA

## Loading required package: foreach

## Loading required package: iterators

## Package 'GA' version 3.0.2

## Type 'citation("GA")' for citing this R package in publications.
```

Data

We will be using the UCI Machine Learning Repository: Wine Data Set. These data are the results of a chemical analysis of wines grown in

Genetic Algorithms

A genetic algorithm (GA) is a search heuristic that mimics the process of natural selection. This heuristic is often used to generate useful solutions to optimization and search problems.



Genetic algorithm (GA)

Genetic Algorithm Pseudocode

Create an initial population, typically random

While the best candidate so far is not a solution:
Create new population using crossover and mutation.
Evaluate the fitness of each candidate in the population.
Replace/delete least-fit population

Return the best candidate found

Genetic Algorithm Basic components

- Candidate representation
  - Important to choose this well the form can effect the solution.
  - The typical candidate representation is a binary string.
- successor functions.
  - Mutation, crossover
  - Mutation - Given a candidate, return a slightly different candidate.

    - Crossover - Given two candidates, produce one that has elements of each.

- Fitness function
    - The fitness function quantitates estimates how close a candidate is to being a solution.

- Solution test
    - Check whether the candidate is a solution.
- Some parameters
    - Population size

    - Generation limit

Pros and Cons

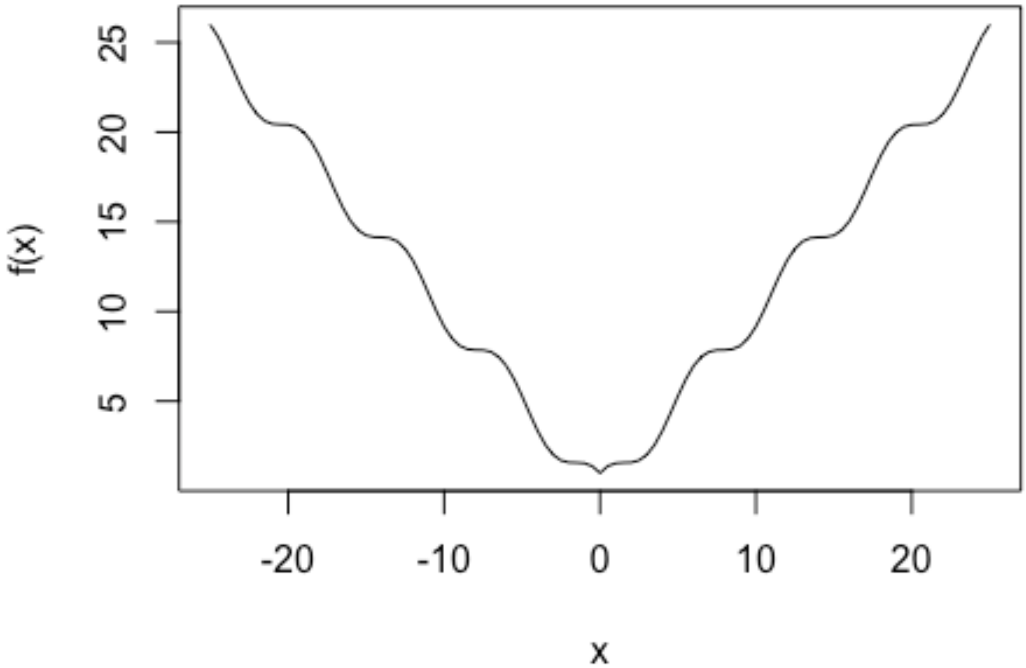
Pros
* Fast (and low memory)
* Finding a candidate representation and fitness function are the bulk of the work.

Cons
* Randomized (not guaranteed optimal or complete).
* Can get stuck on local maxima (crossover is intended to help get out of local maxima)

Genetic Algorithms by hand in R

Genetic Algorithms by hand in R

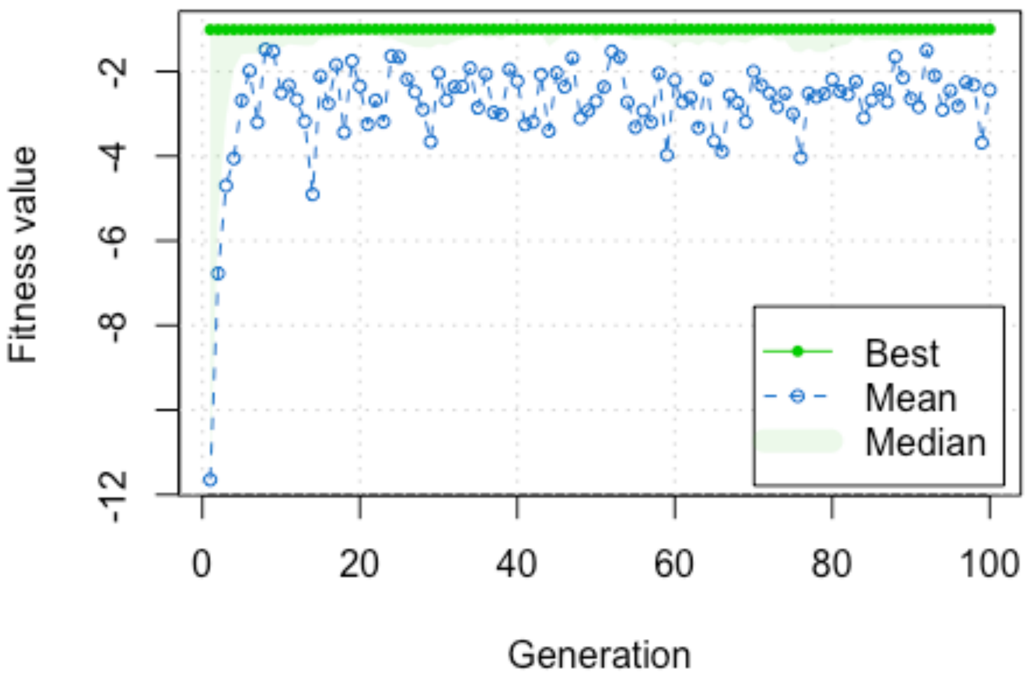
```
# Function: f(x) = |x|+ cos(x)
f <- function(x) abs(x)+cos(x)
min=-25
max=25
curve(f, min, max)
```



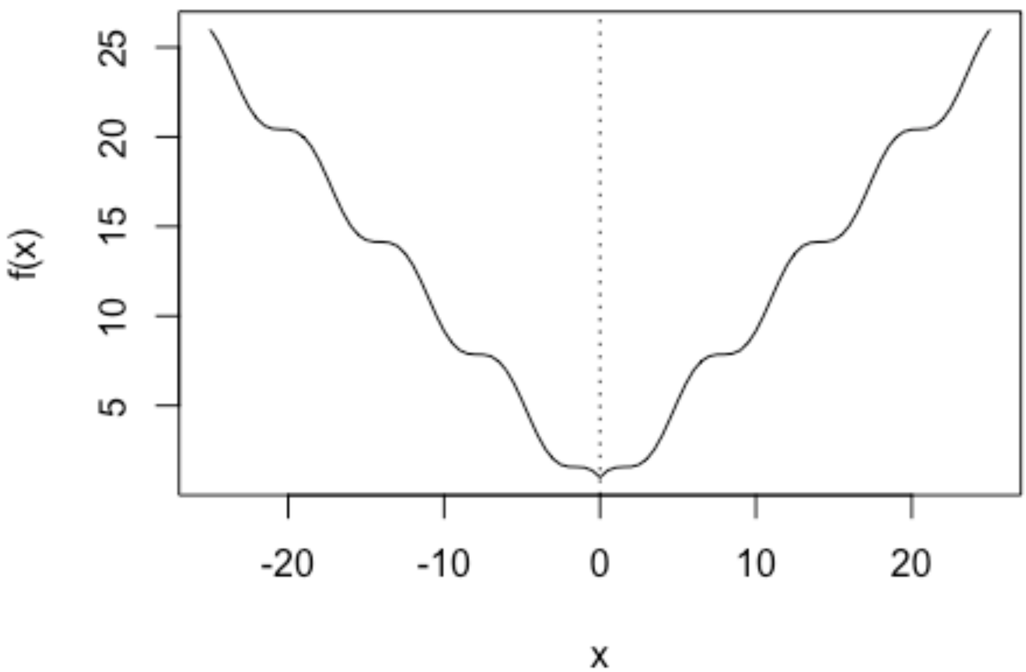

```
fit <- function(x) -f(x)
Gene_Alg <- ga(type = "real-valued", fitness = fit, min = -25, max =
25)
summary(Gene_Alg)

## +-----------------------------------+
## |          Genetic Algorithm        |
## +-----------------------------------+
##
## GA settings:
## Type                  =  real-valued
## Population size       =  50
## Number of generations =  100
## Elitism               =  2
## Crossover probability =  0.8
## Mutation probability  =  0.1
## Search domain =
##      x1
## Min -25
## Max  25
##
## GA results:
## Iterations             = 100
```

```
## Fitness function value = -1.000038
## Solution =
##                  x1
## [1,] 3.751608e-05
```

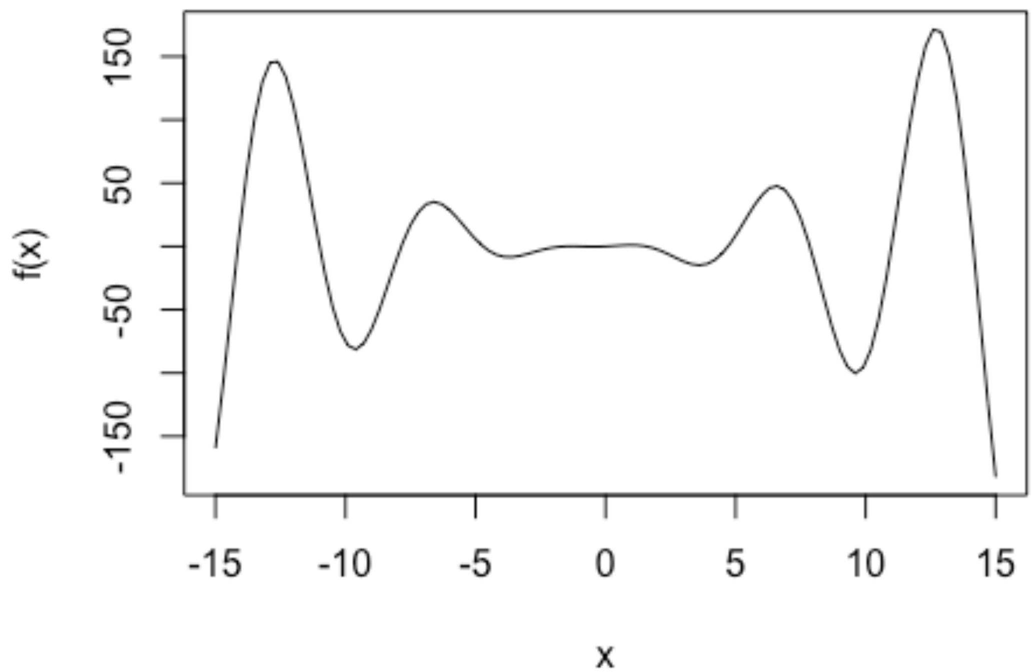
```r
plot(Gene_Alg)
```



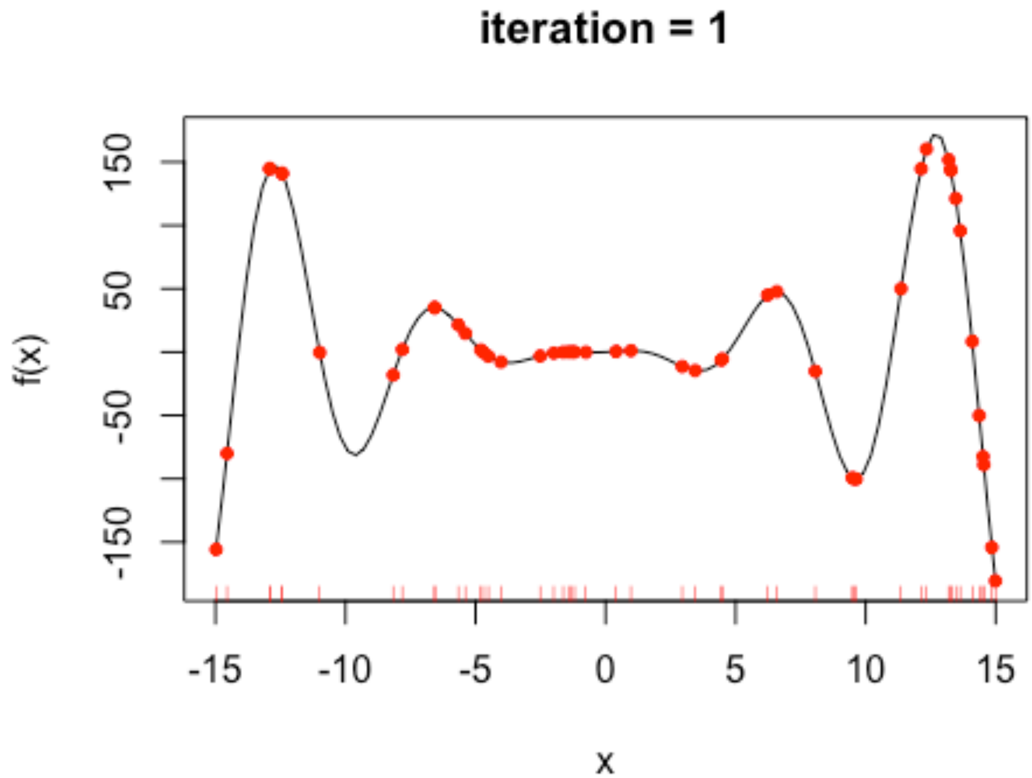

```r
curve(f, -25, 25)
abline(v = Gene_Alg@solution, lty = 3)
```




```
#-----------------------          one-dimensional function      -------
-----------------

# Function : f(x) = (x^2 + x)* cos(x)
f <- function(x) (x^2+x)*cos(x)     # -15 < x < 15
curve(f, -15, 15)
```

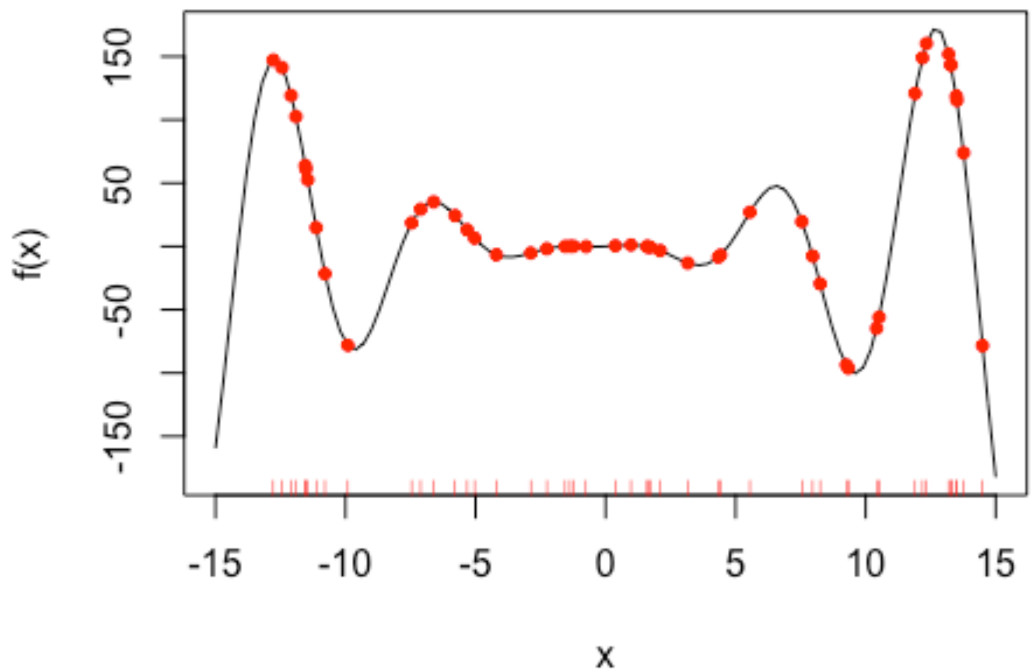


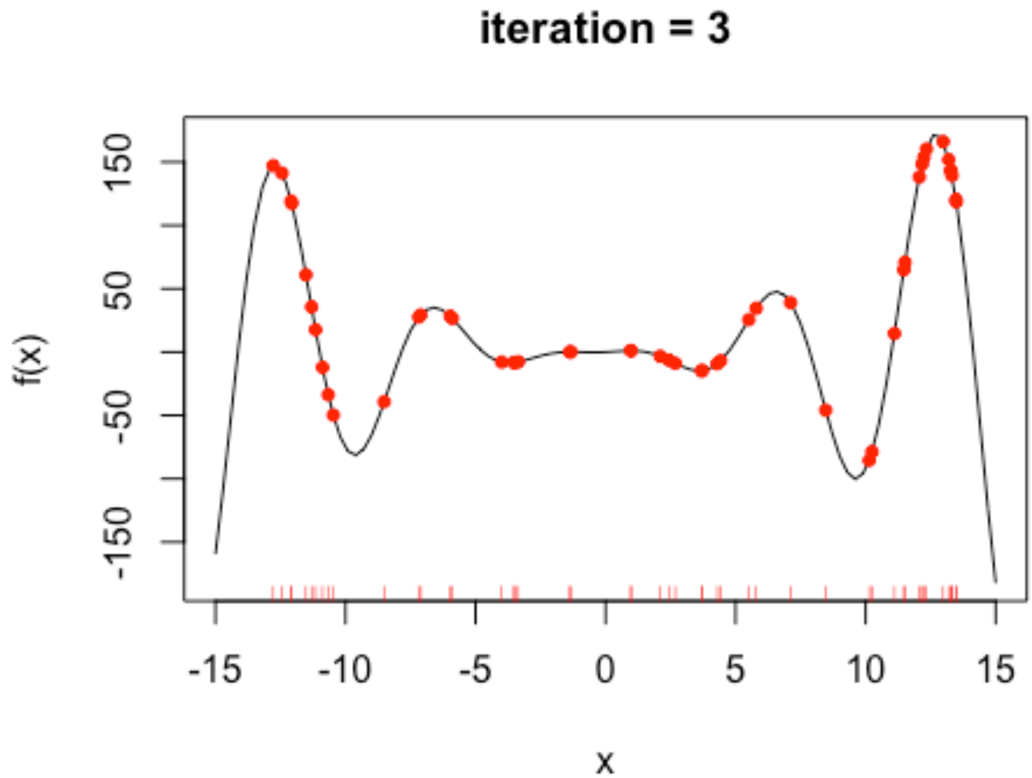

```r
# tracing function
monitor <- function(obj)
{
  curve(f, -15, 15, main = paste("iteration =", obj@iter))
  points(obj@population, obj@fitness, pch = 20, col = 2)
  rug(obj@population, col = 2)
  Sys.sleep(0.2)
}
## For the maximization of this function we may use f directly as the
fitness function:
Gene_Alg <- ga(type = "real-valued", fitness = f, min = -15, max = 15,
monitor = monitor)
```

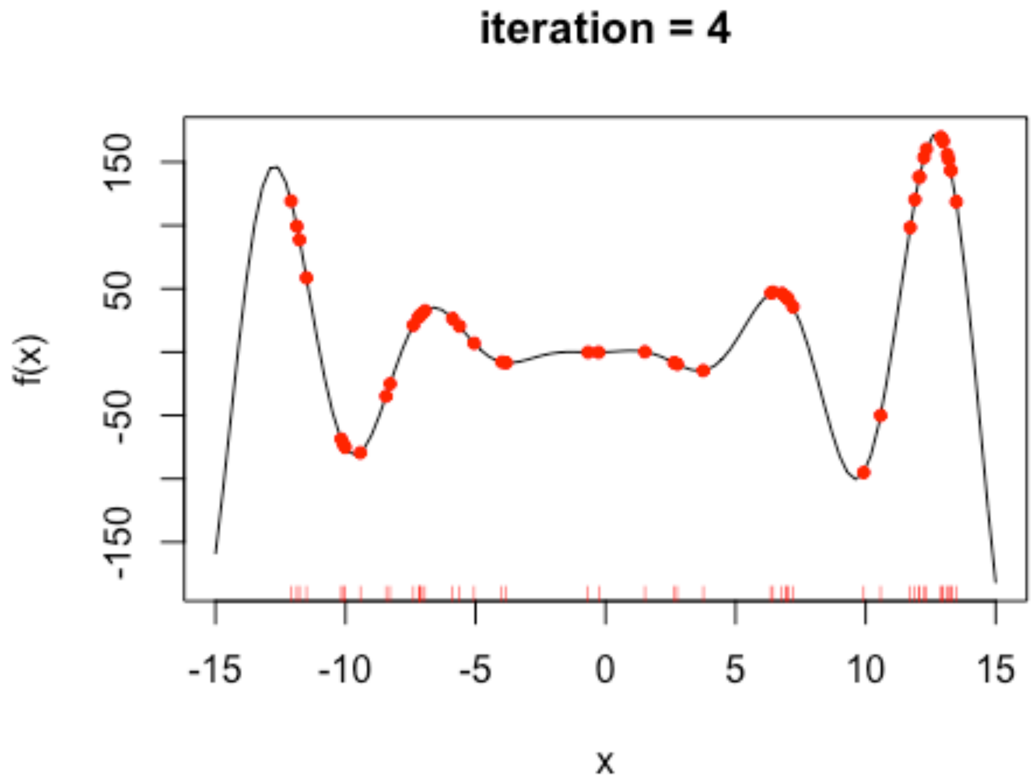


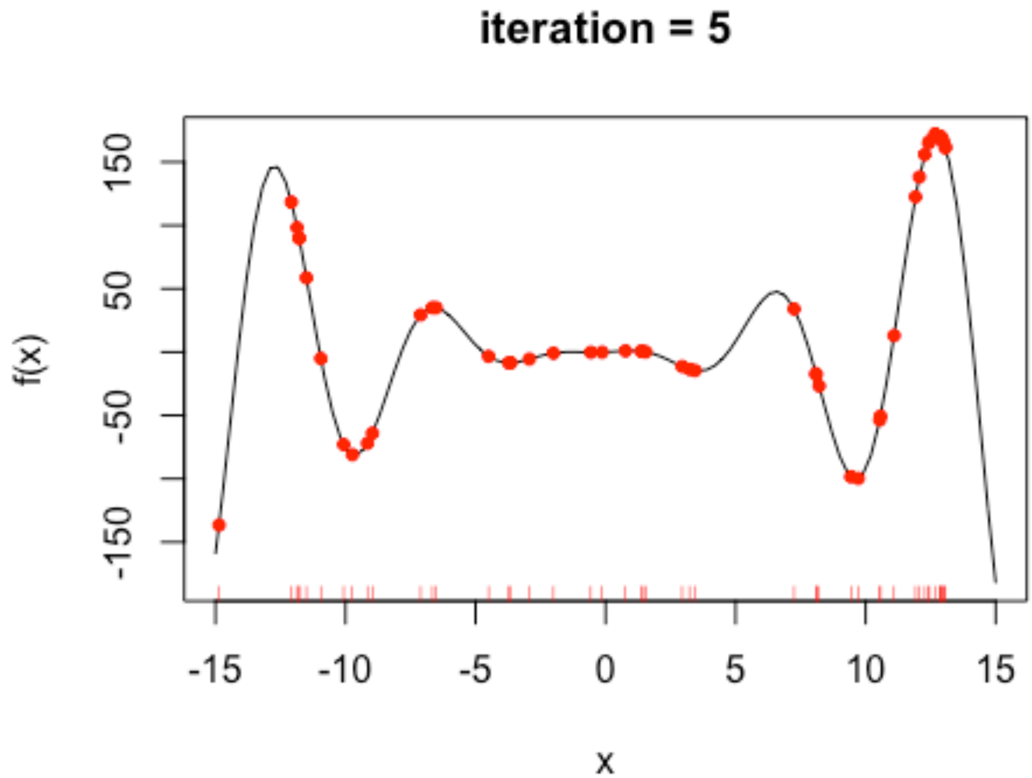
iteration = 1
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

iteration = 2

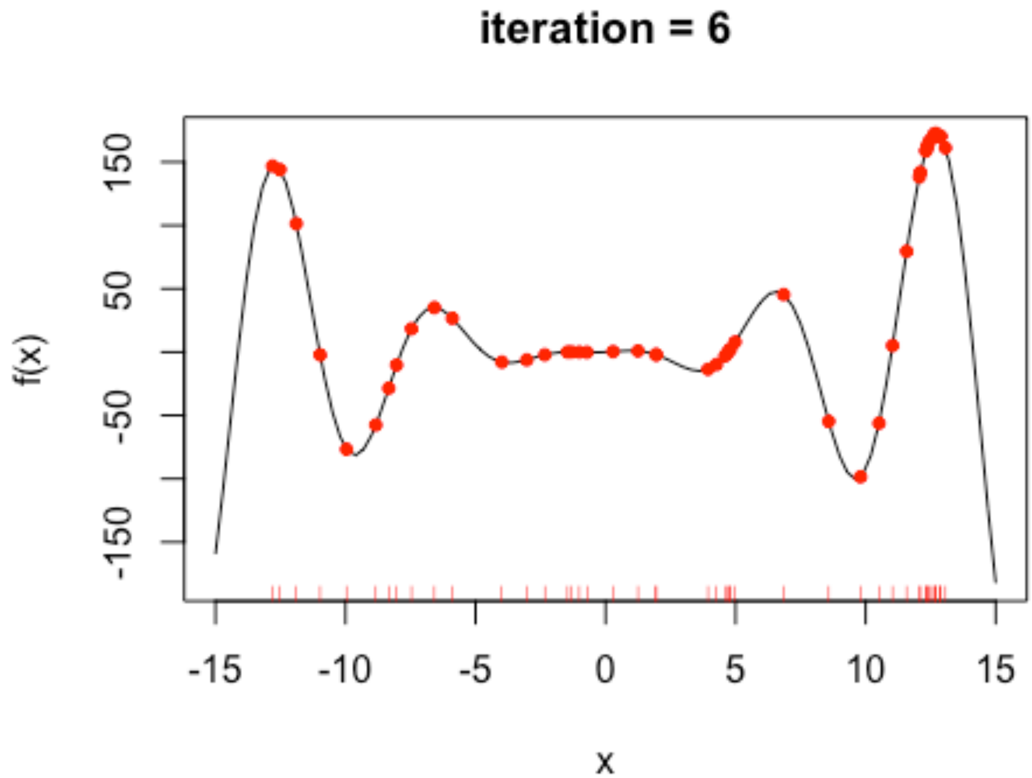



iteration = 3
150
50
-50
-150
f(x)
-15
-10
-5
0
5
10
15
x

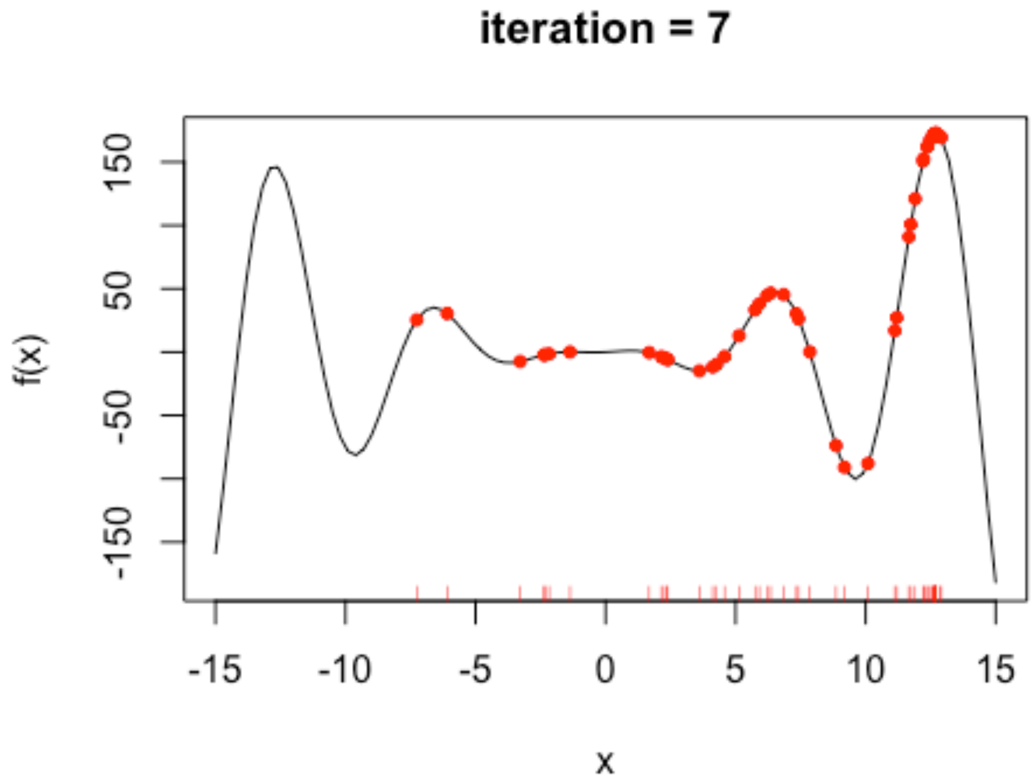

iteration = 4
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x



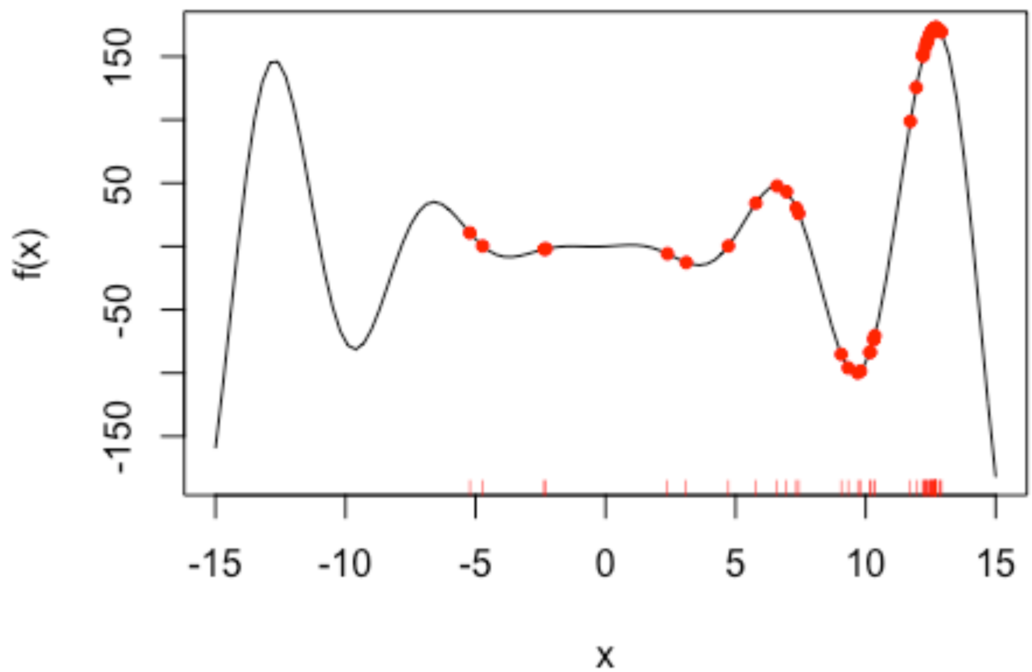
iteration = 5
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

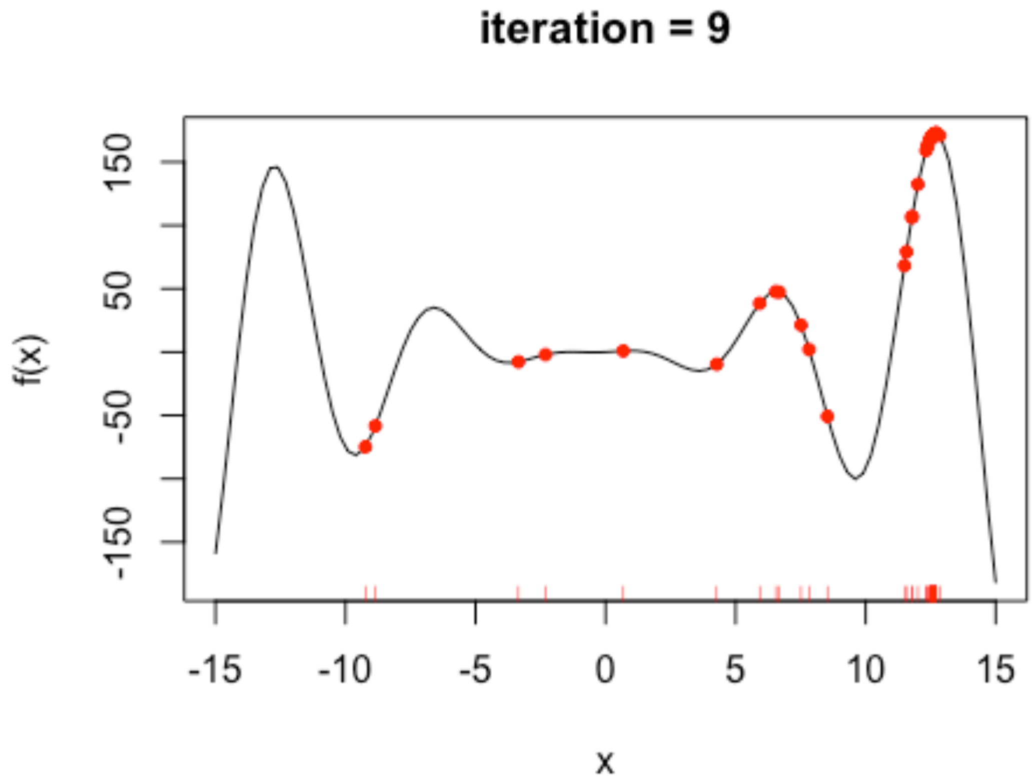


iteration = 6
150
50
-50
-150
f(x)
-15
-10
-5
0
5
10
15
x

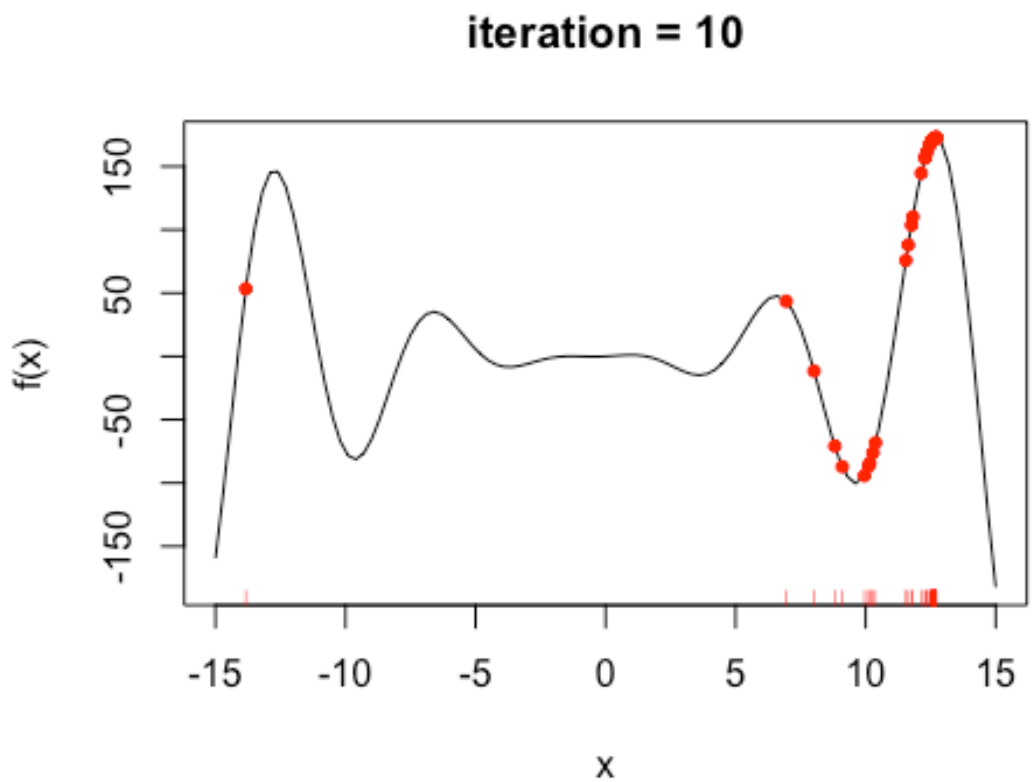

iteration = 7
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

iteration = 8

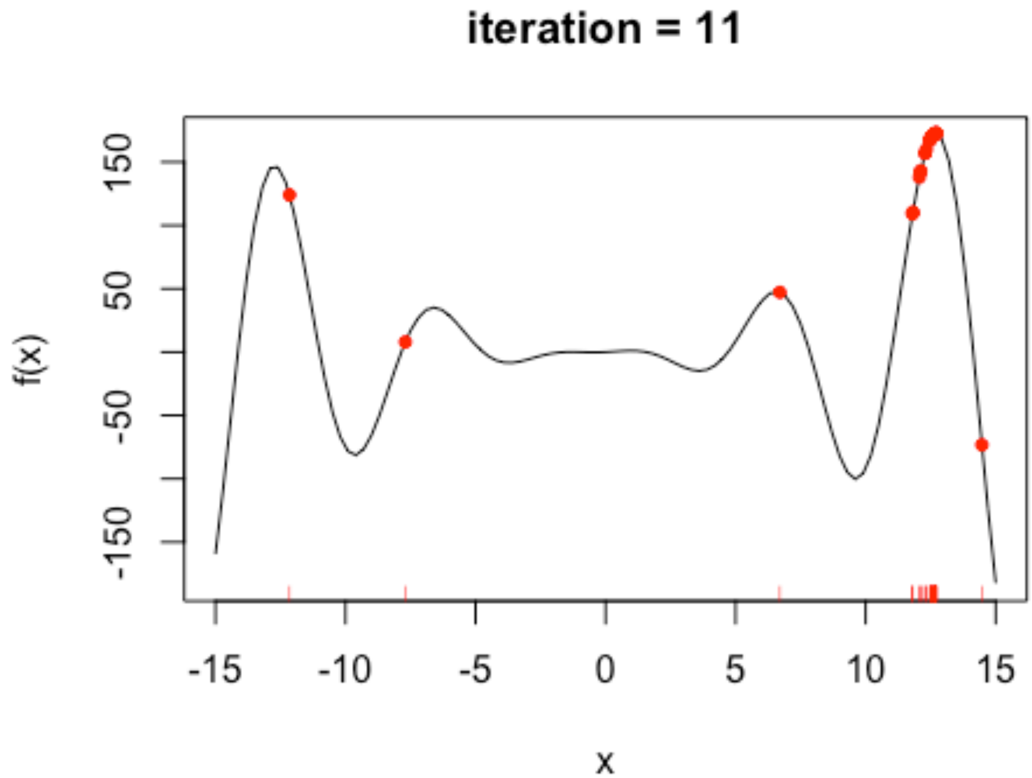


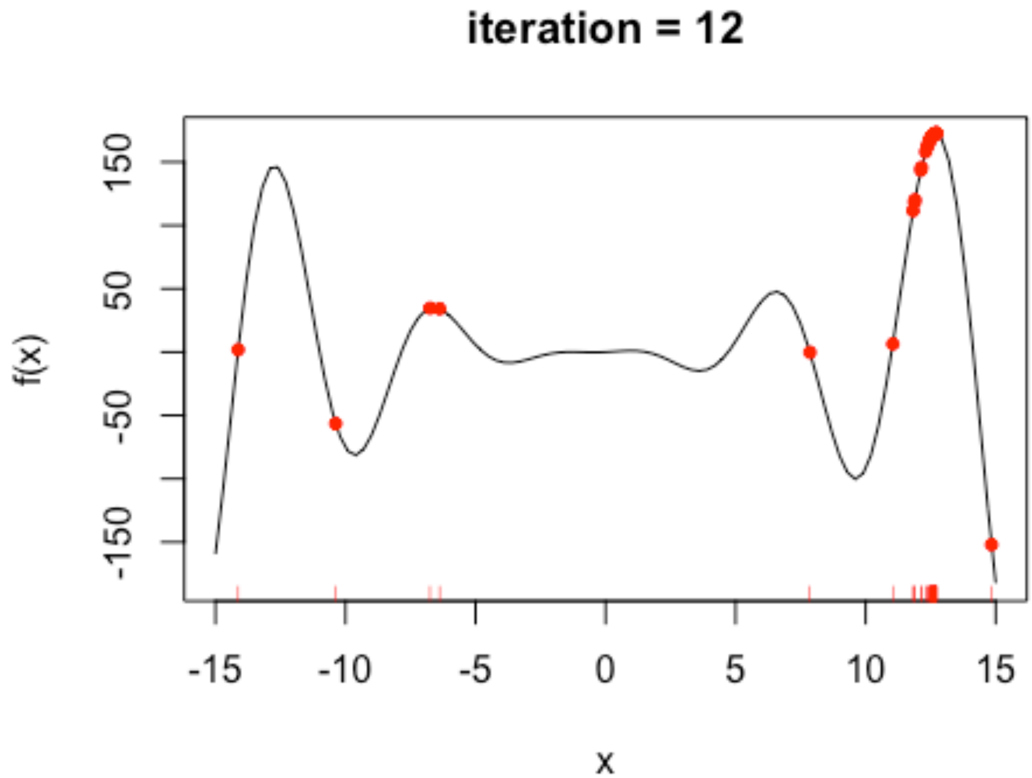


iteration = 9
150
50
-50
-150
f(x)
-15
-10
-5
0
5
10
15
x

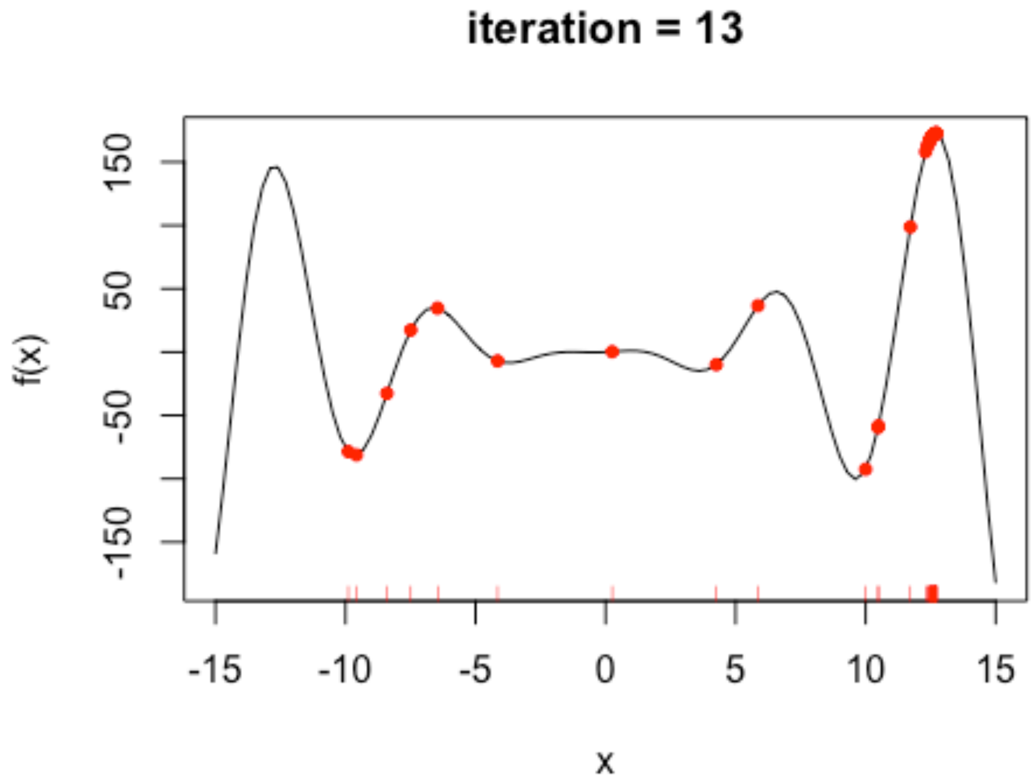


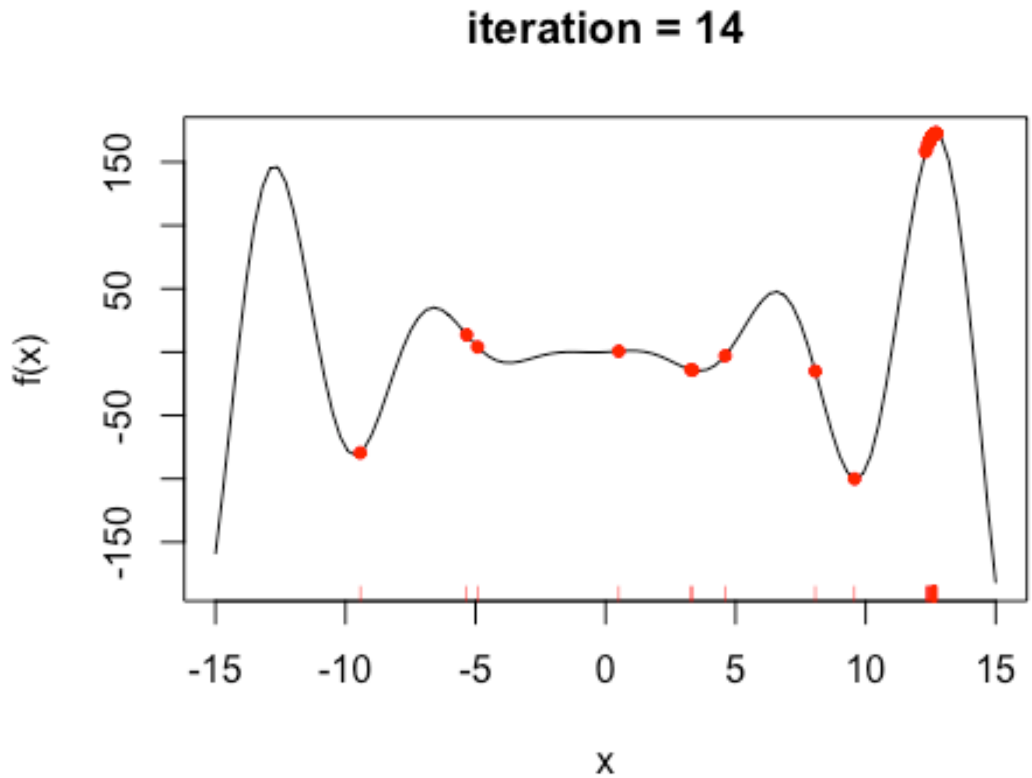
iteration = 10
f(x)
x
150
50
-50
-150
-15
-10
-5
0
5
10
15


iteration = 11
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

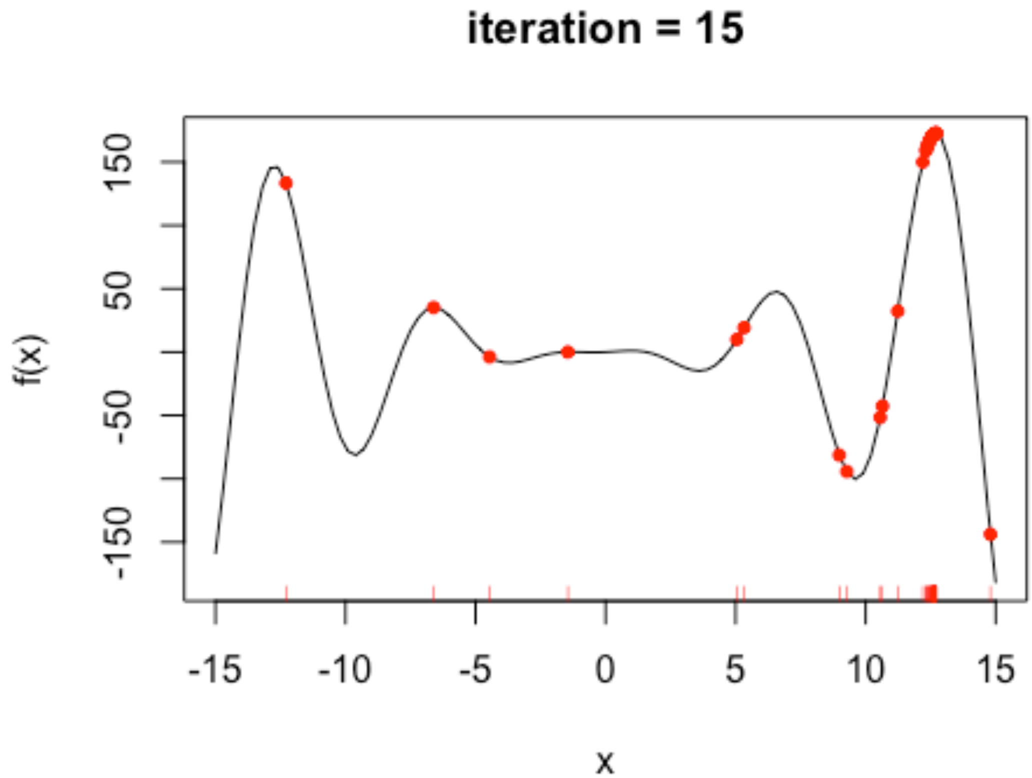


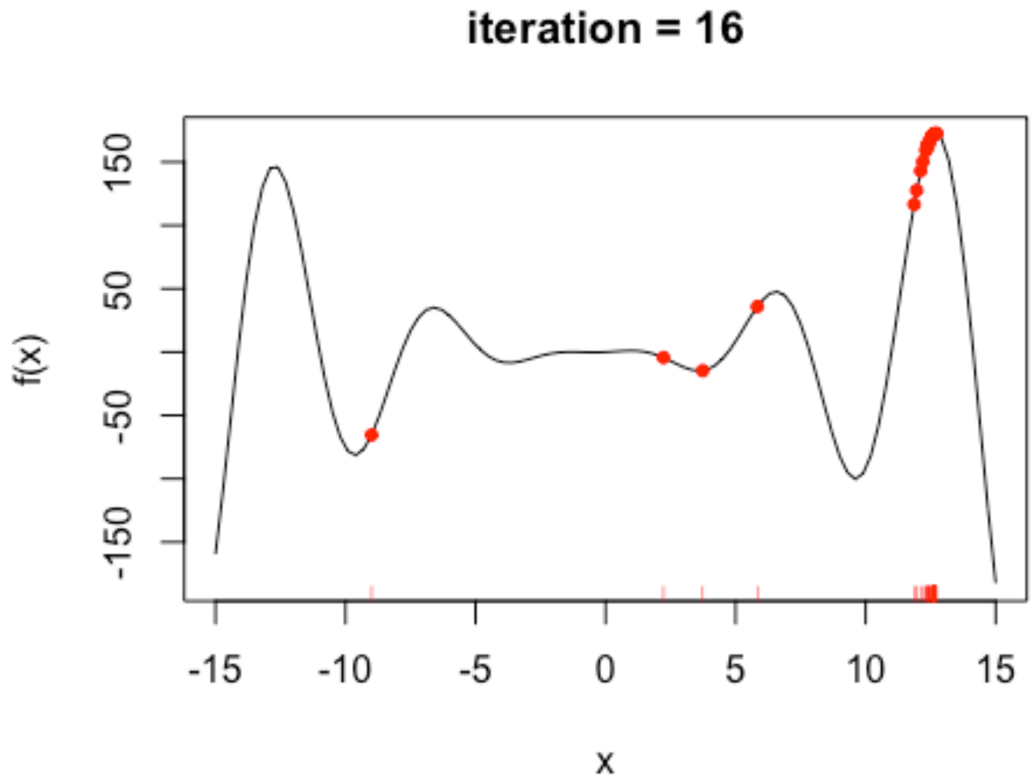
iteration = 12
150
50
-50
-150
f(x)
-15
-10
-5
0
5
10
15
x

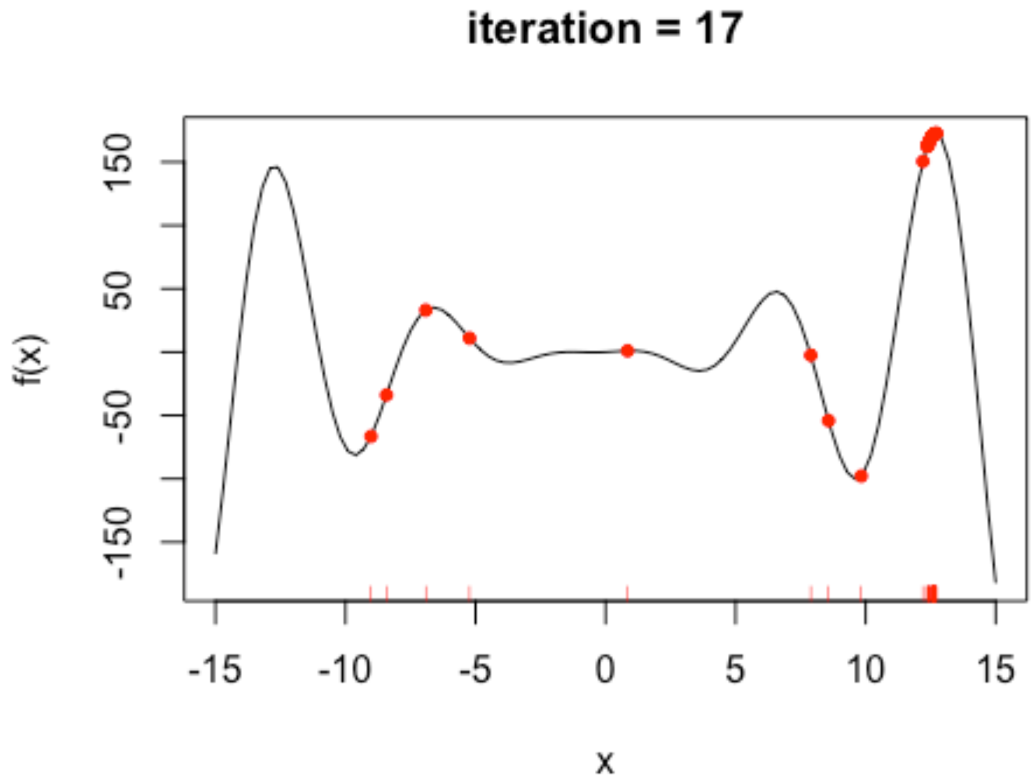

iteration = 13
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x



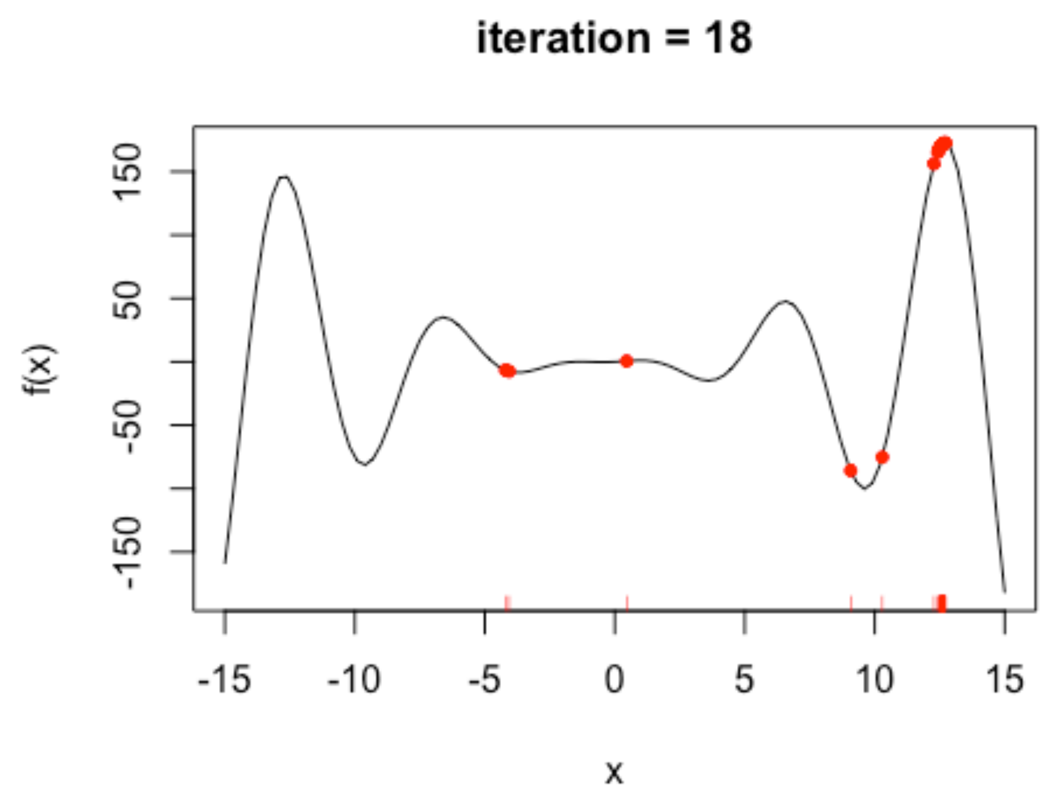
iteration = 14
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

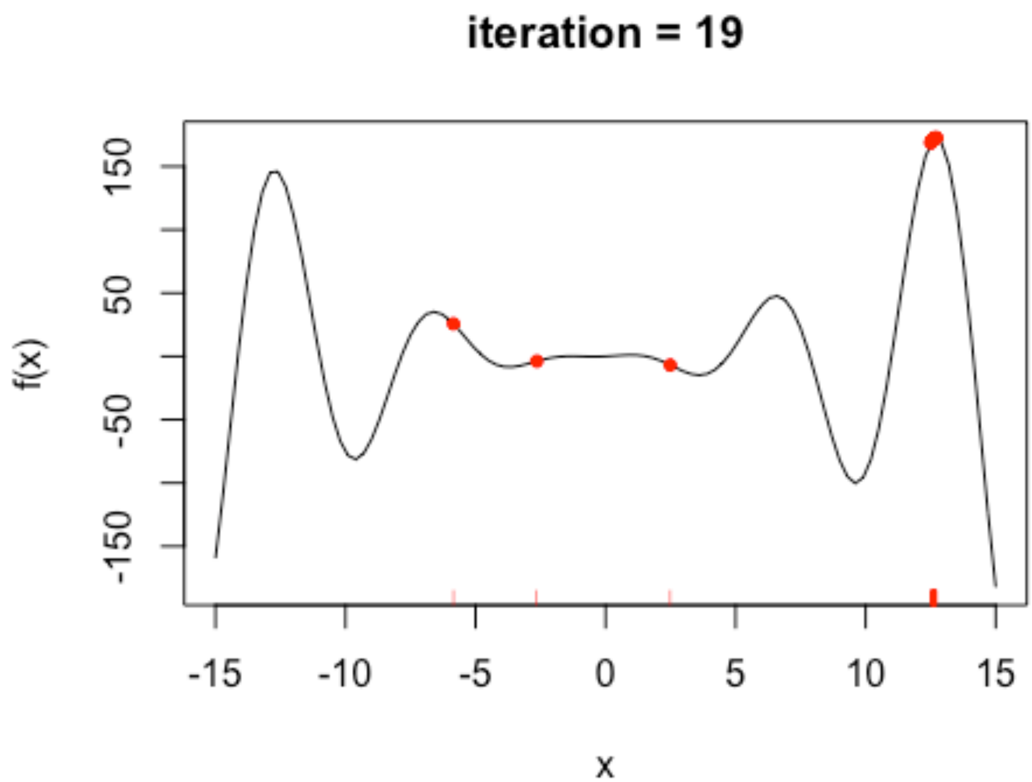

iteration = 15
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

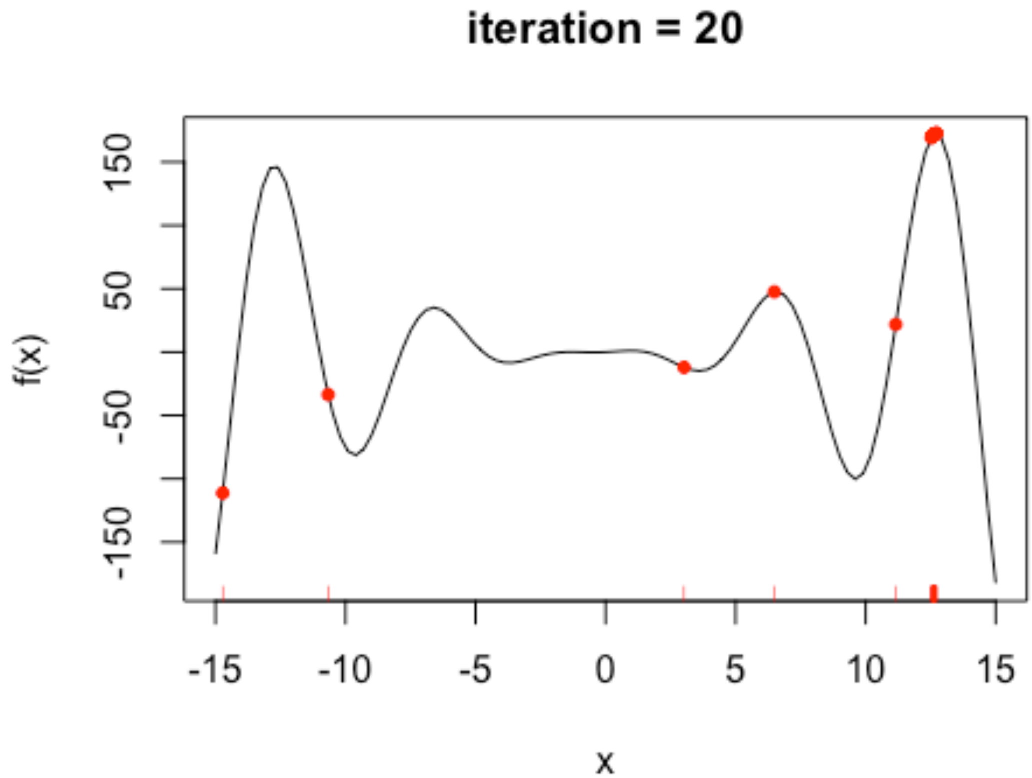

iteration = 16
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x



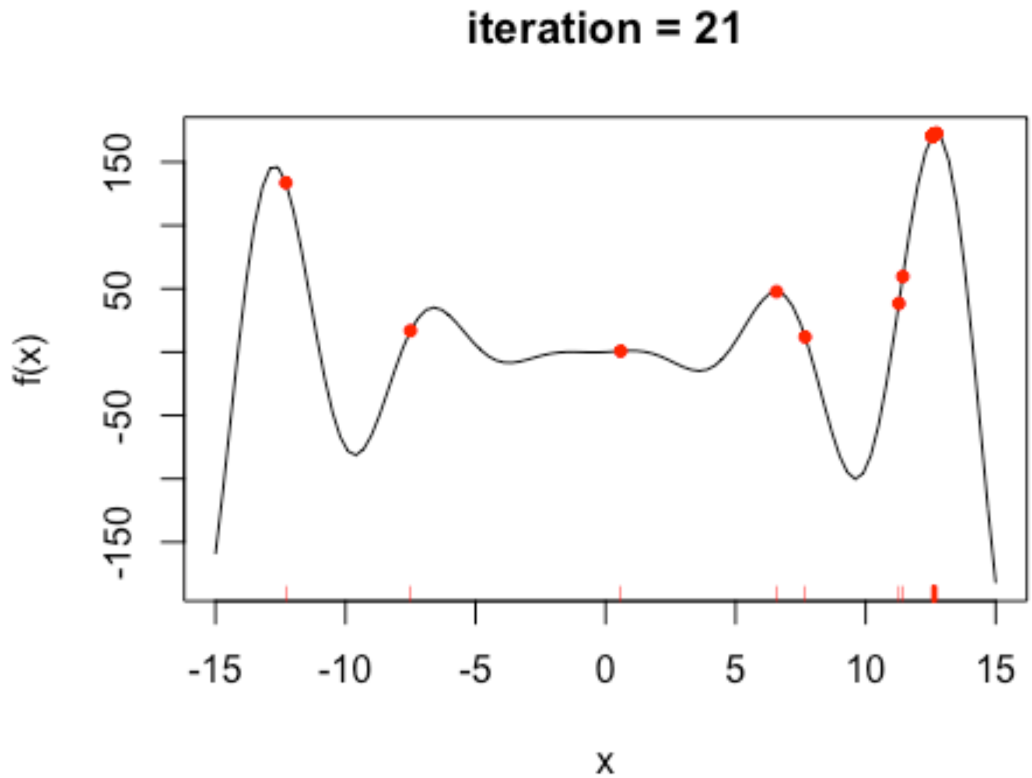
iteration = 17
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

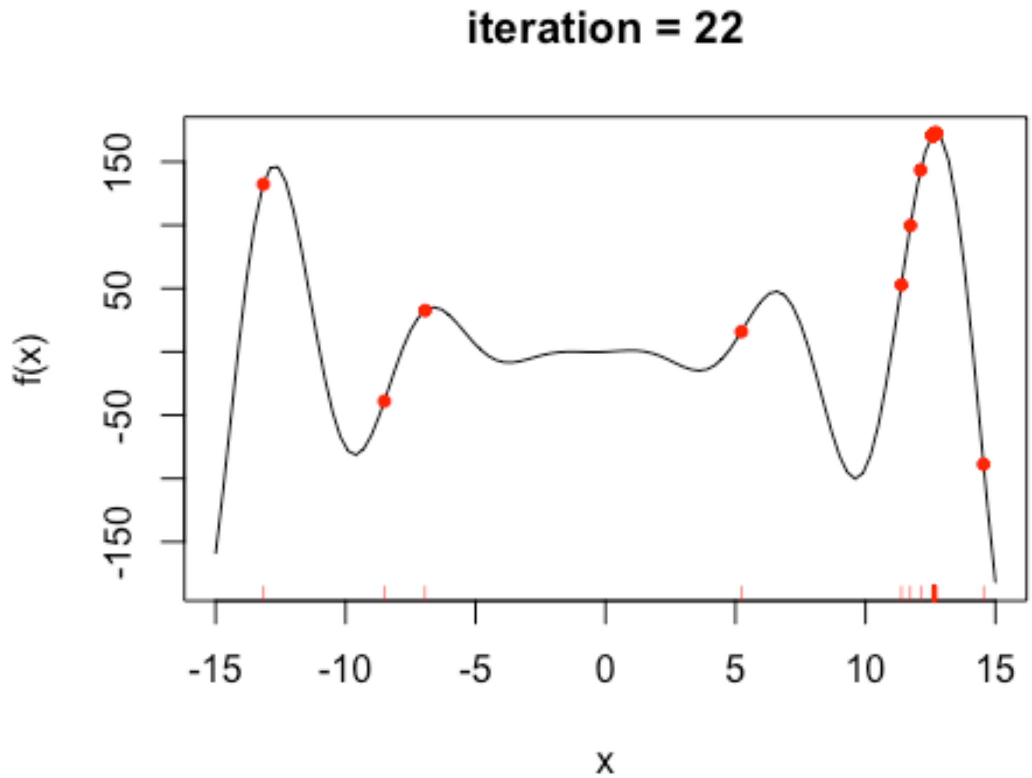

iteration = 18
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

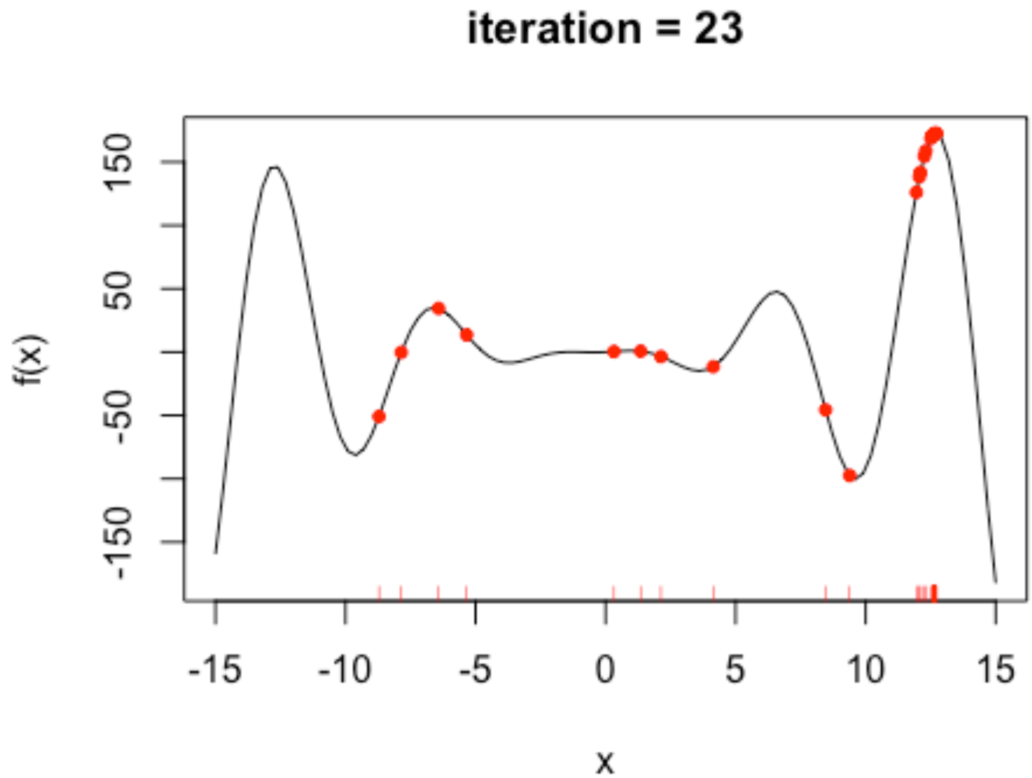

iteration = 19
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

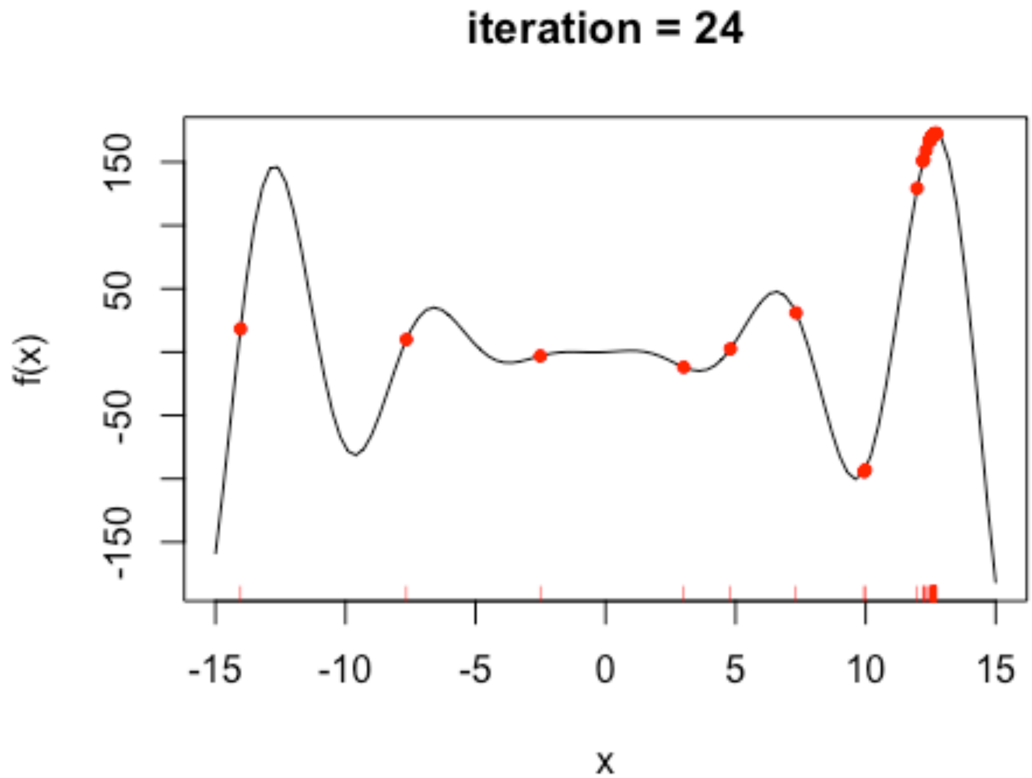


iteration = 20
f(x)
x
150
50
-50
-150
-15
-10
-5
0
5
10
15

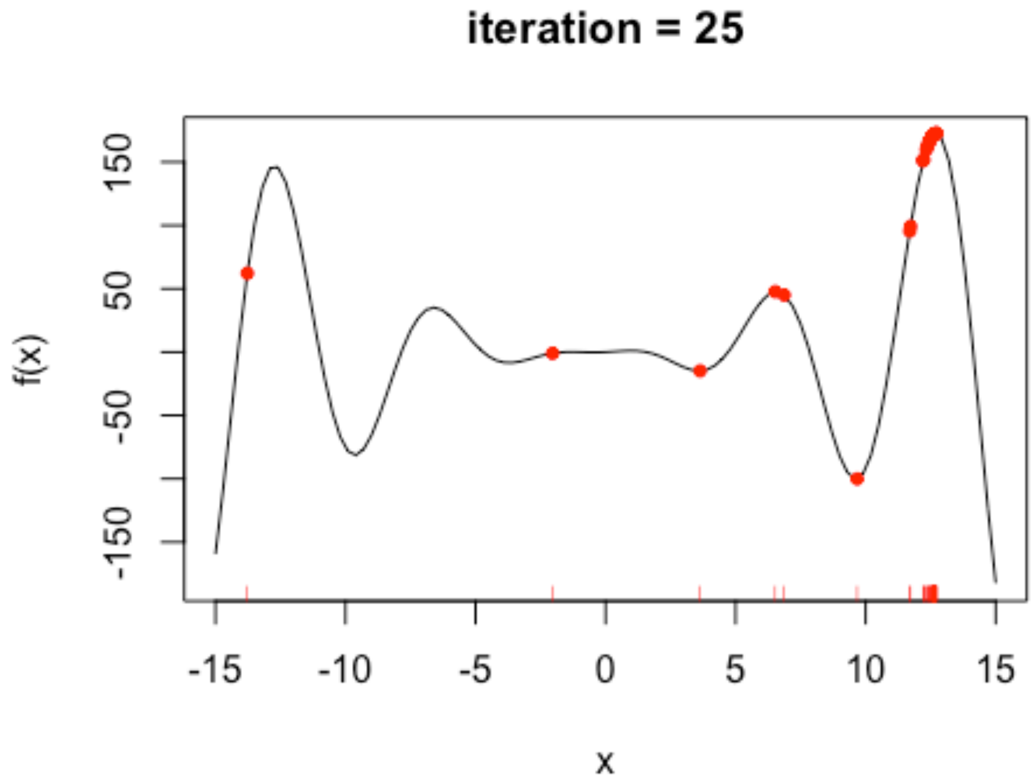

iteration = 21
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

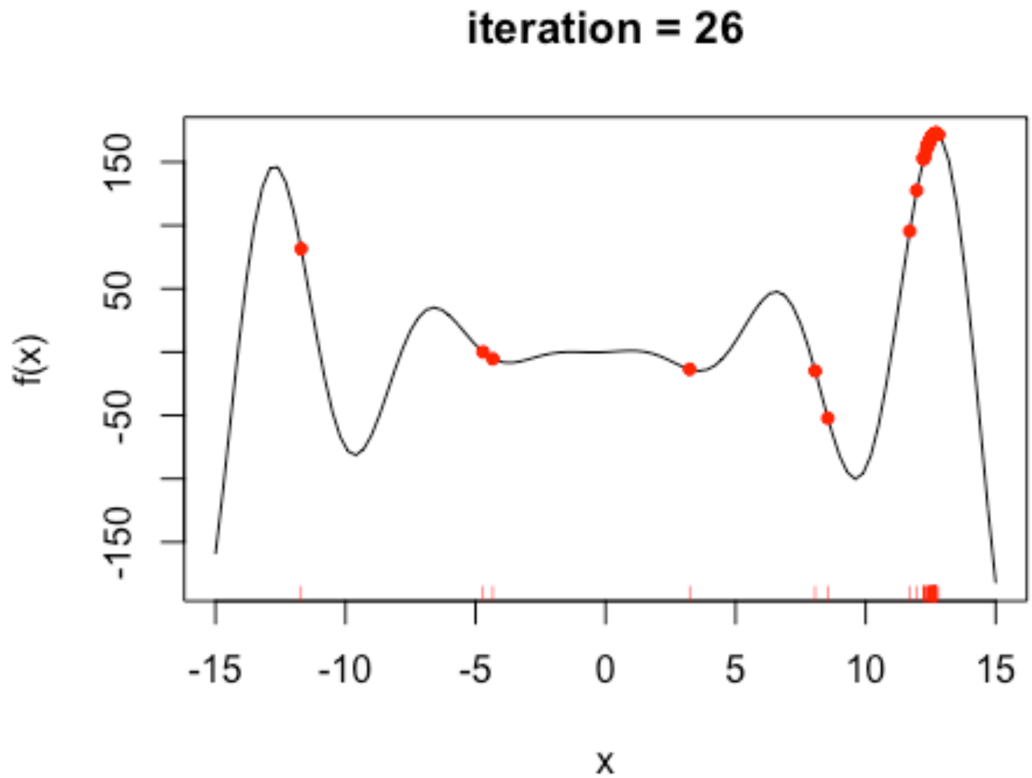

iteration = 22
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

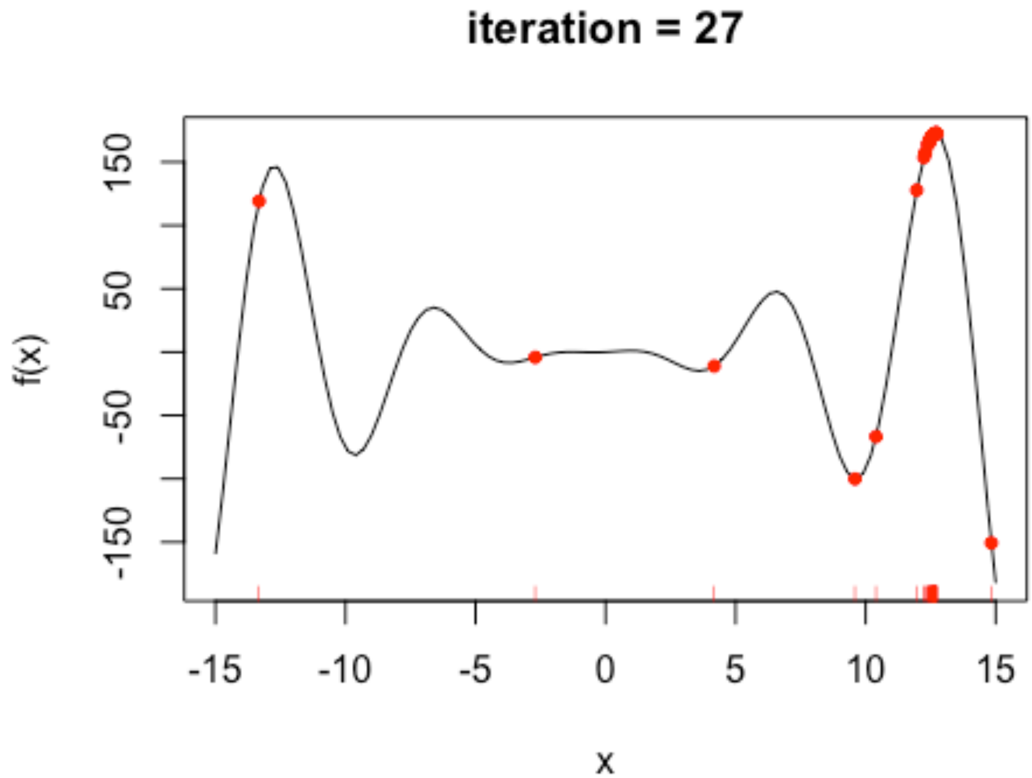

iteration = 23
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

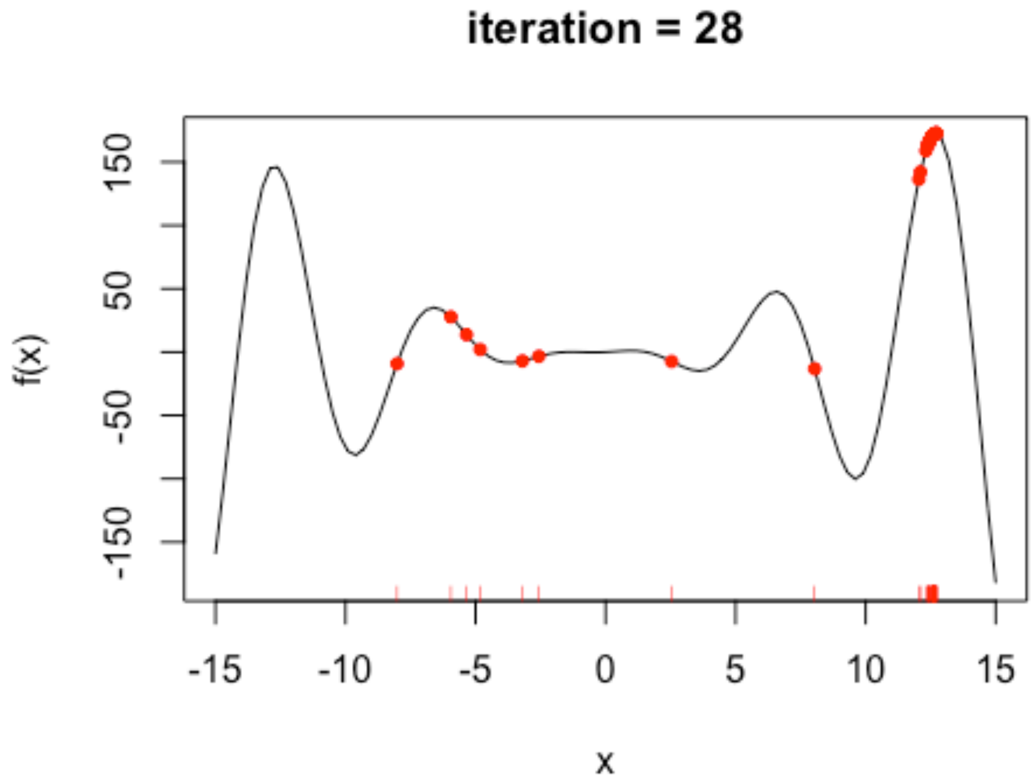


iteration = 24
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

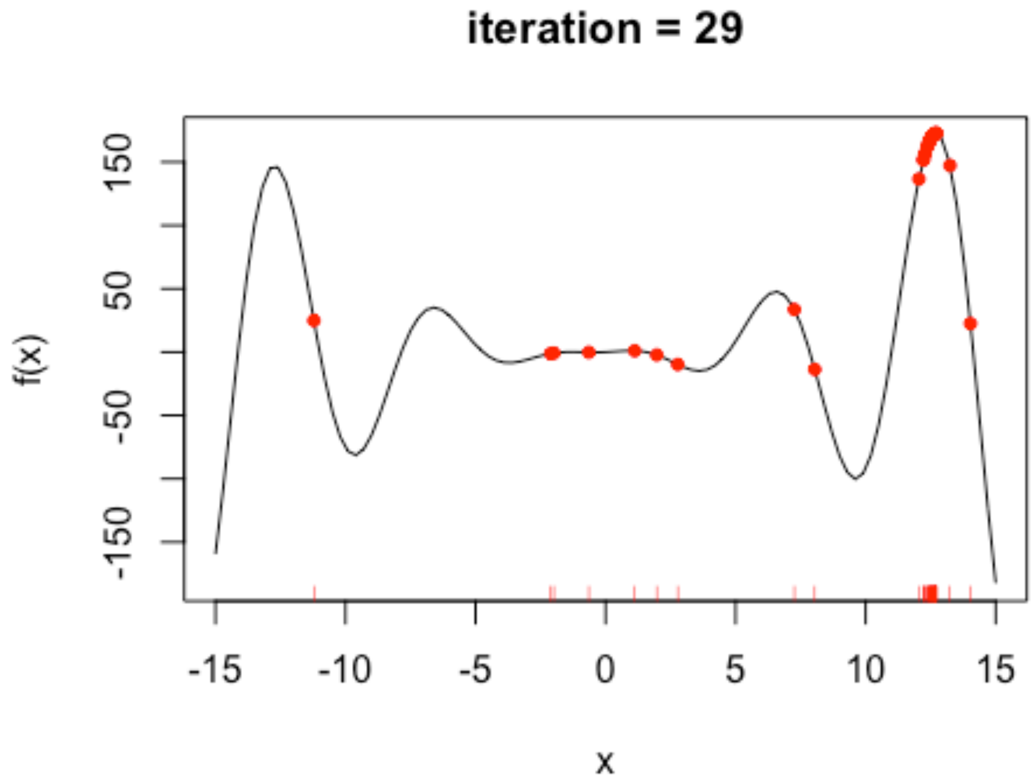

iteration = 25
150
50
-50
-150
f(x)
-15
-10
-5
0
5
10
15
x


iteration = 26
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

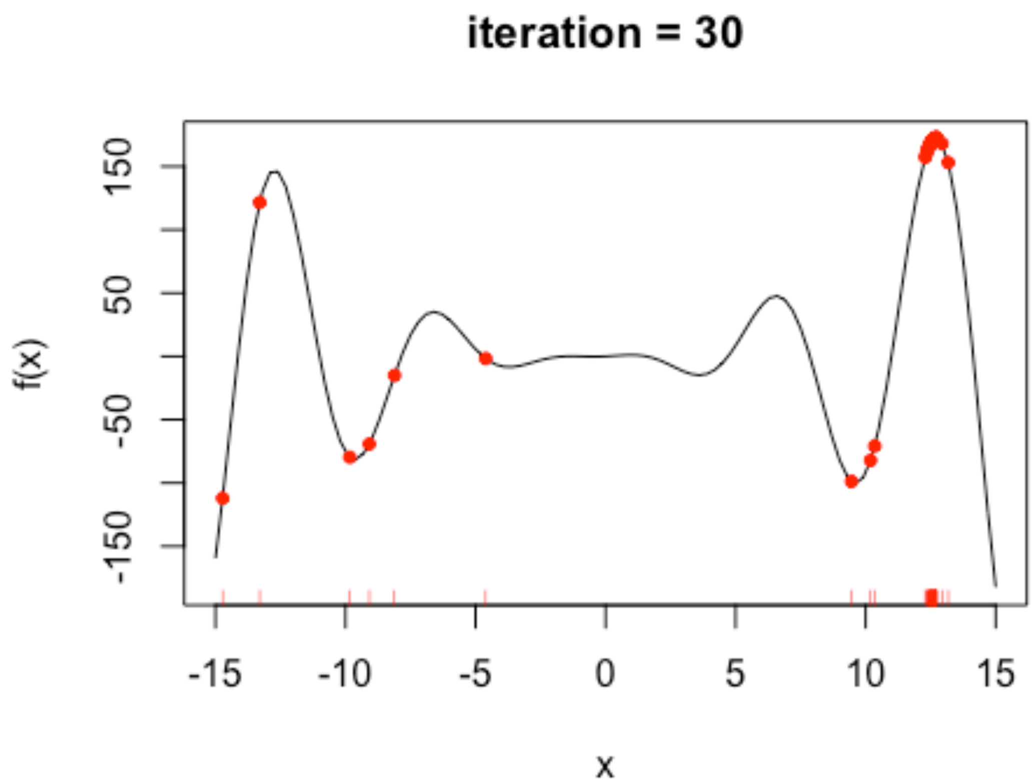

iteration = 27
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

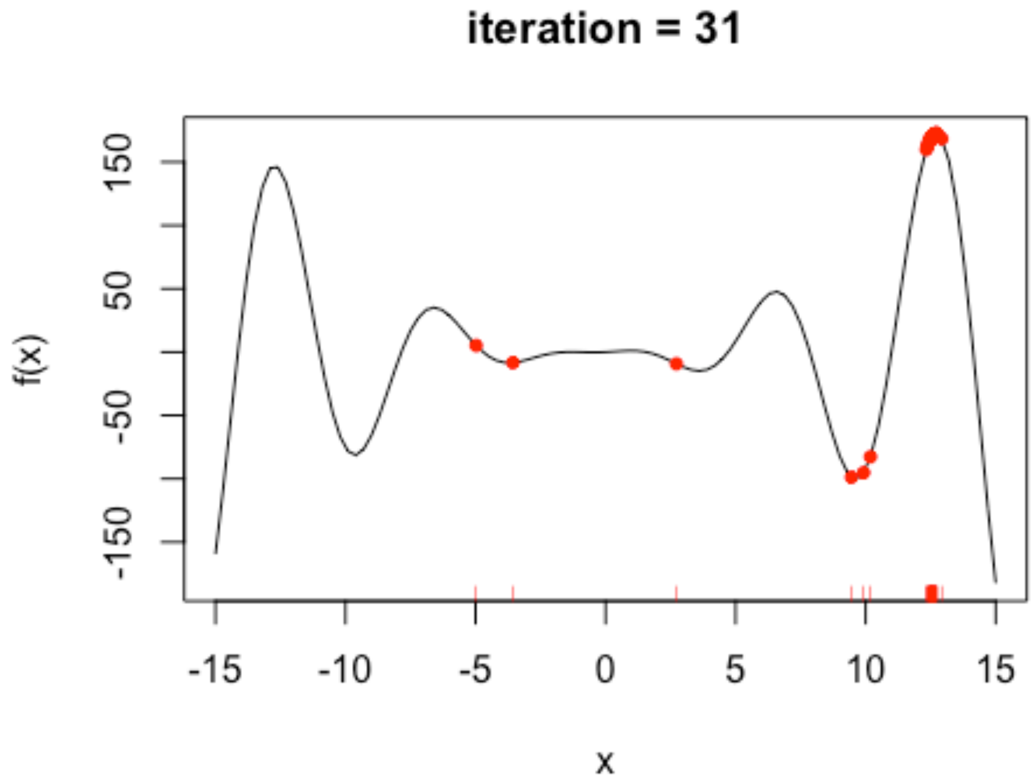


iteration = 28
150
50
-50
-150
f(x)
-15
-10
-5
0
5
10
15
x

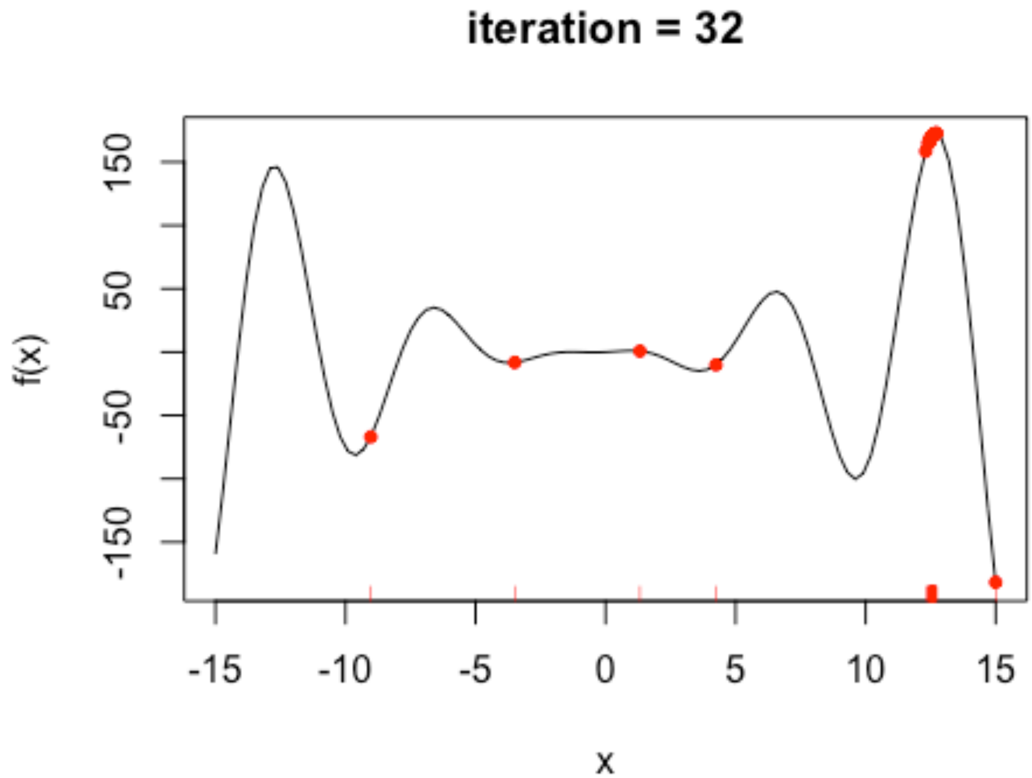


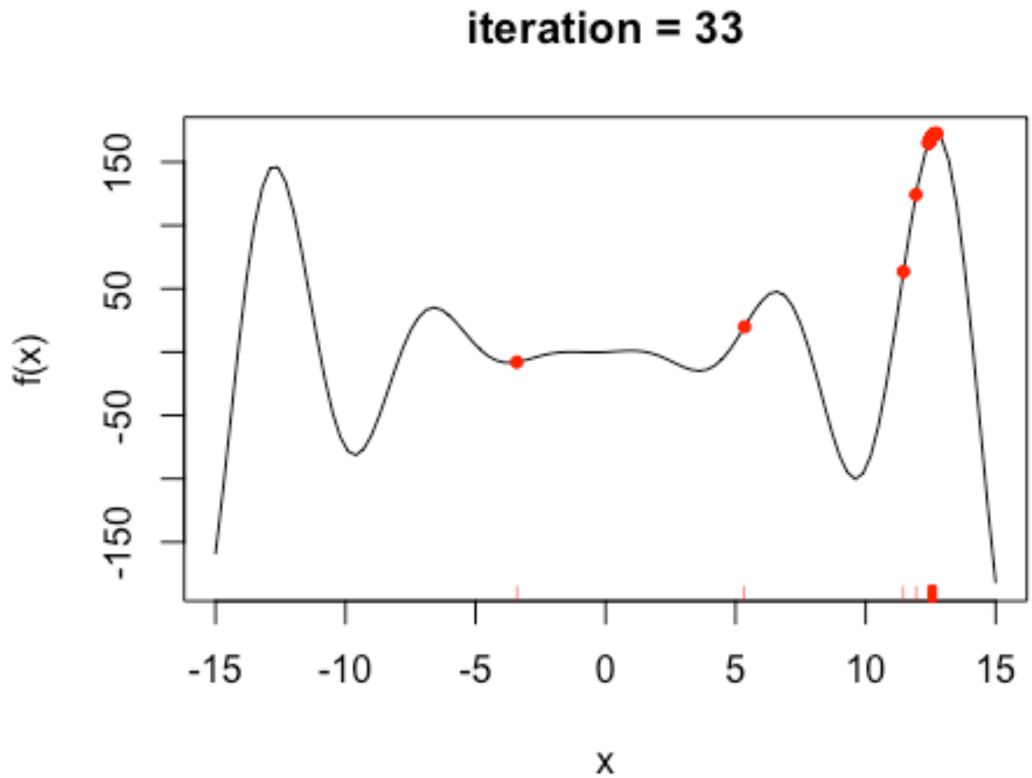
iteration = 29
f(x)
x
150
50
-50
-150
-15
-10
-5
0
5
10
15

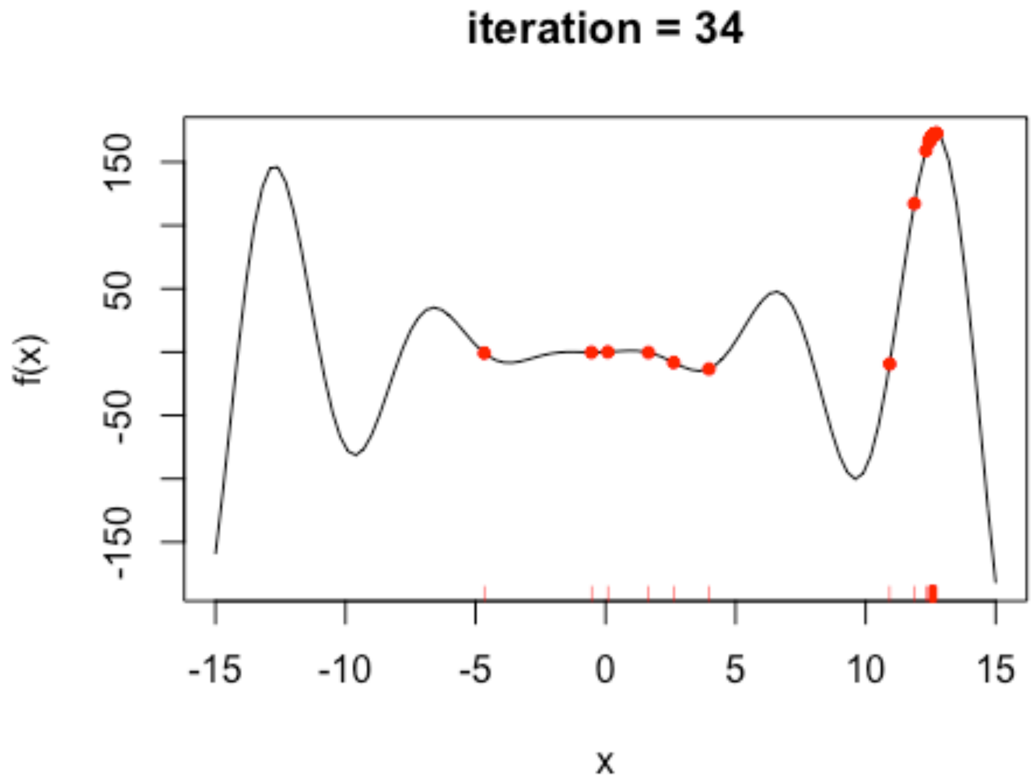

iteration = 30
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

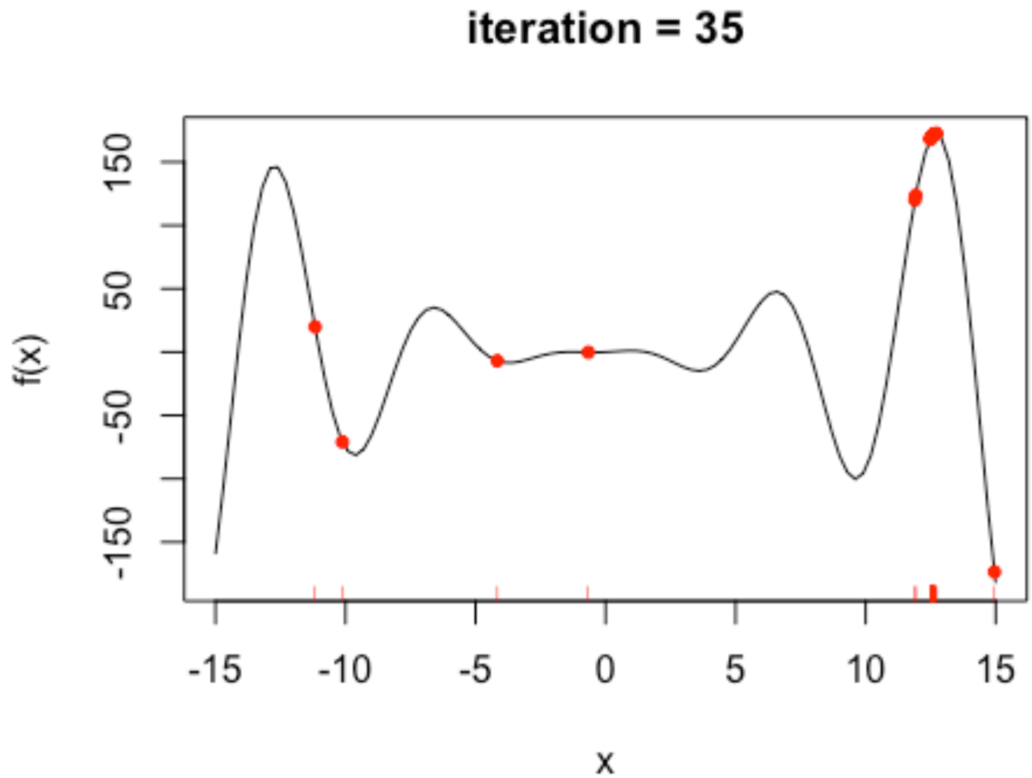


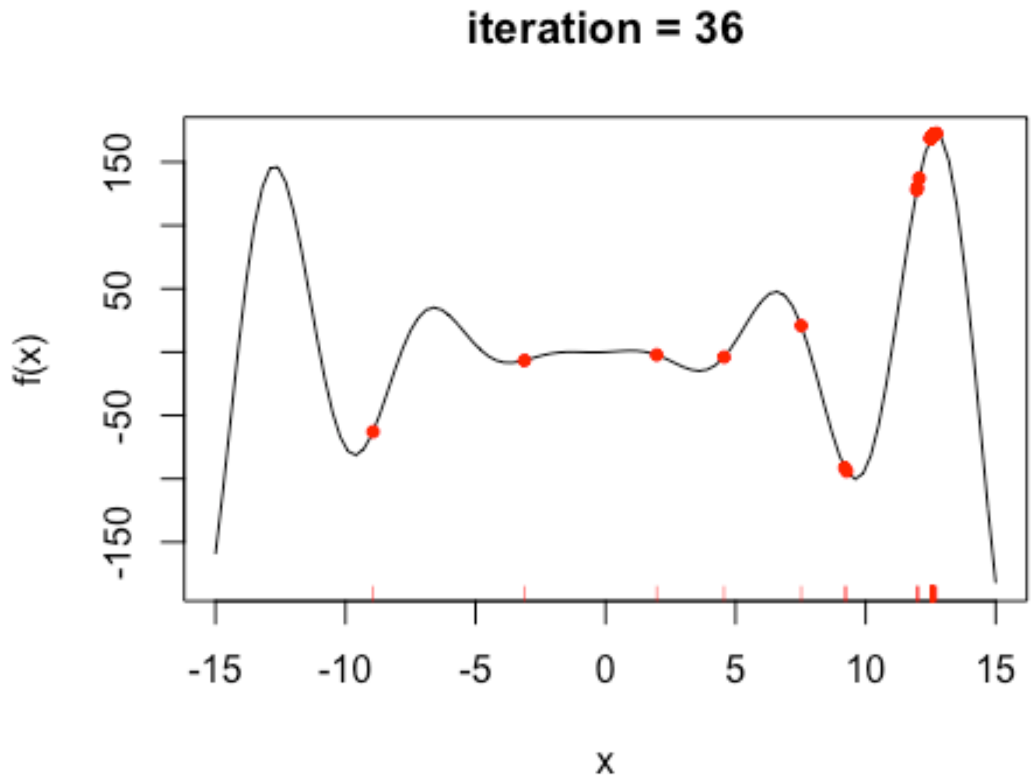
iteration = 31
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x


iteration = 32
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

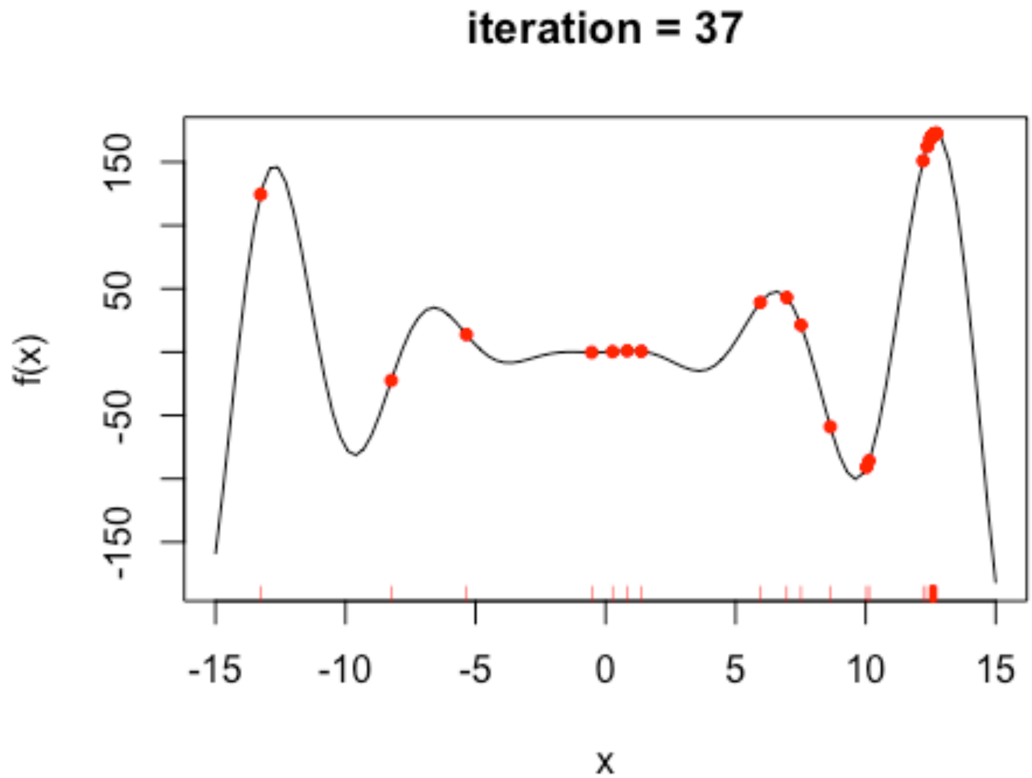

iteration = 33
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

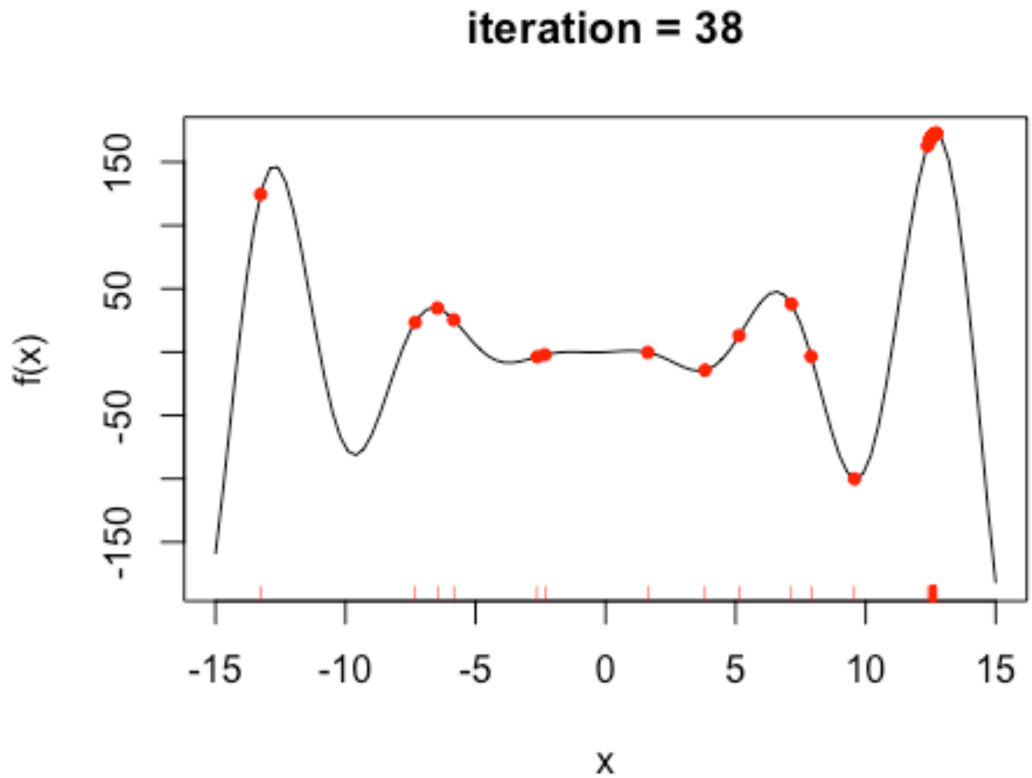

iteration = 34
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x


iteration = 35
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

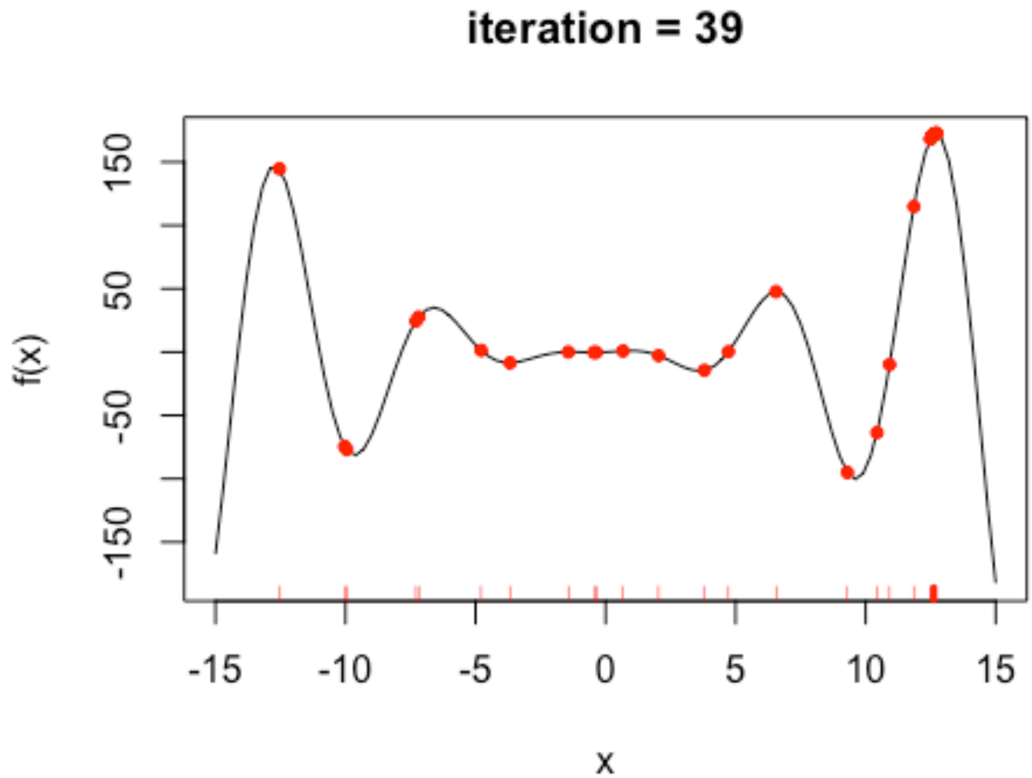


iteration = 36
f(x)
x
150
50
-50
-150
-15
-10
-5
0
5
10
15

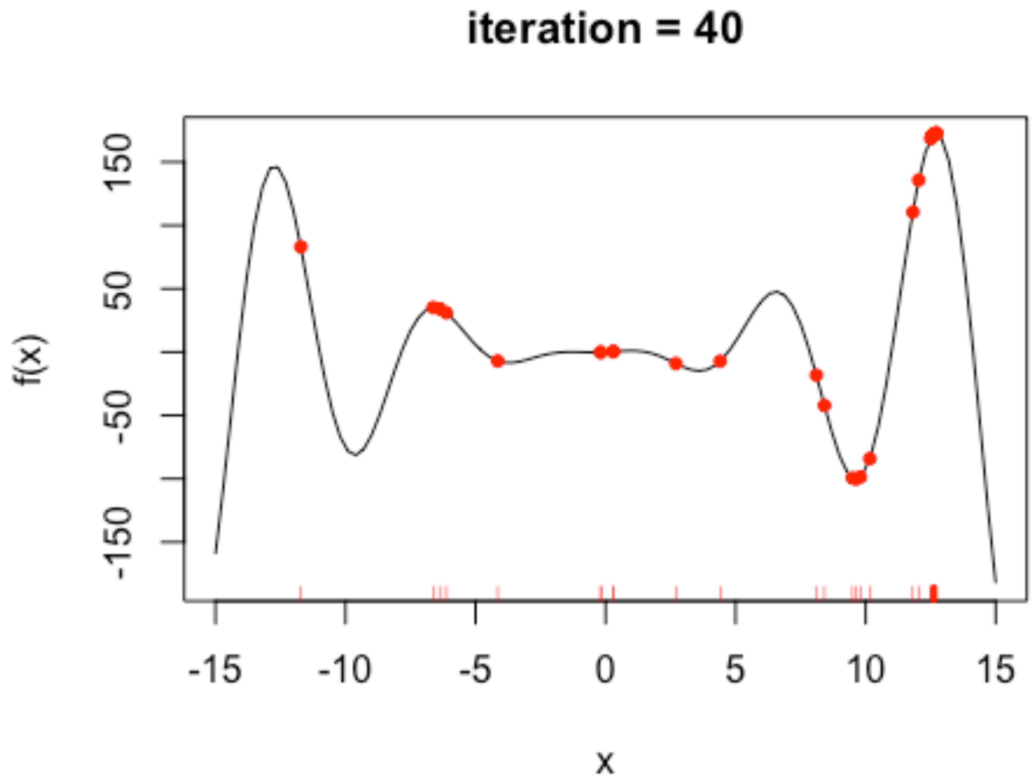

iteration = 37
f(x)
x
150
50
-50
-150
-15
-10
-5
0
5
10
15



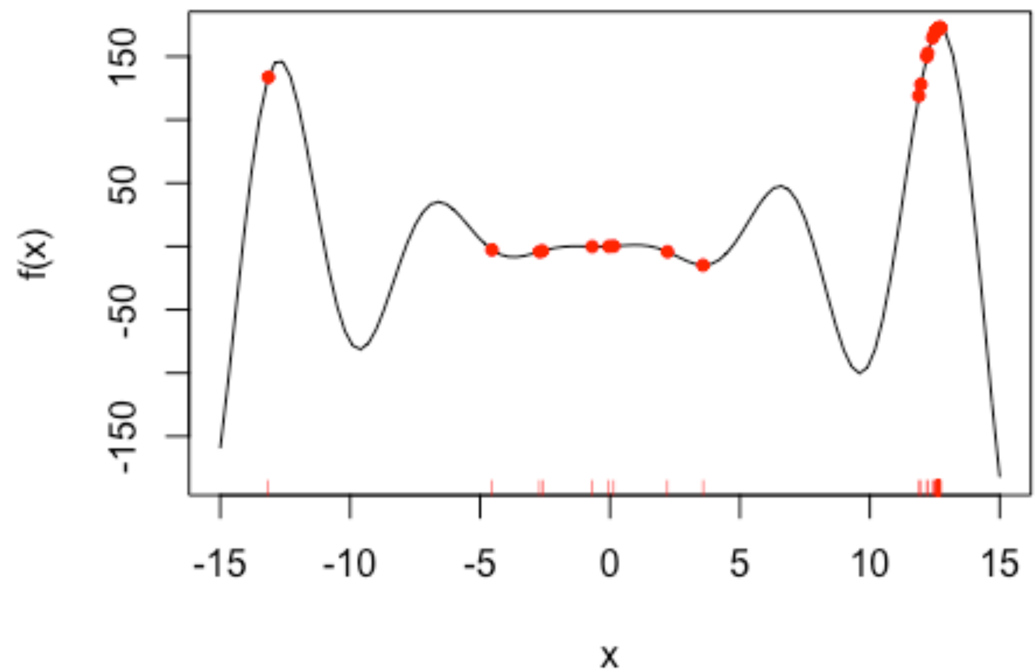
iteration = 38
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

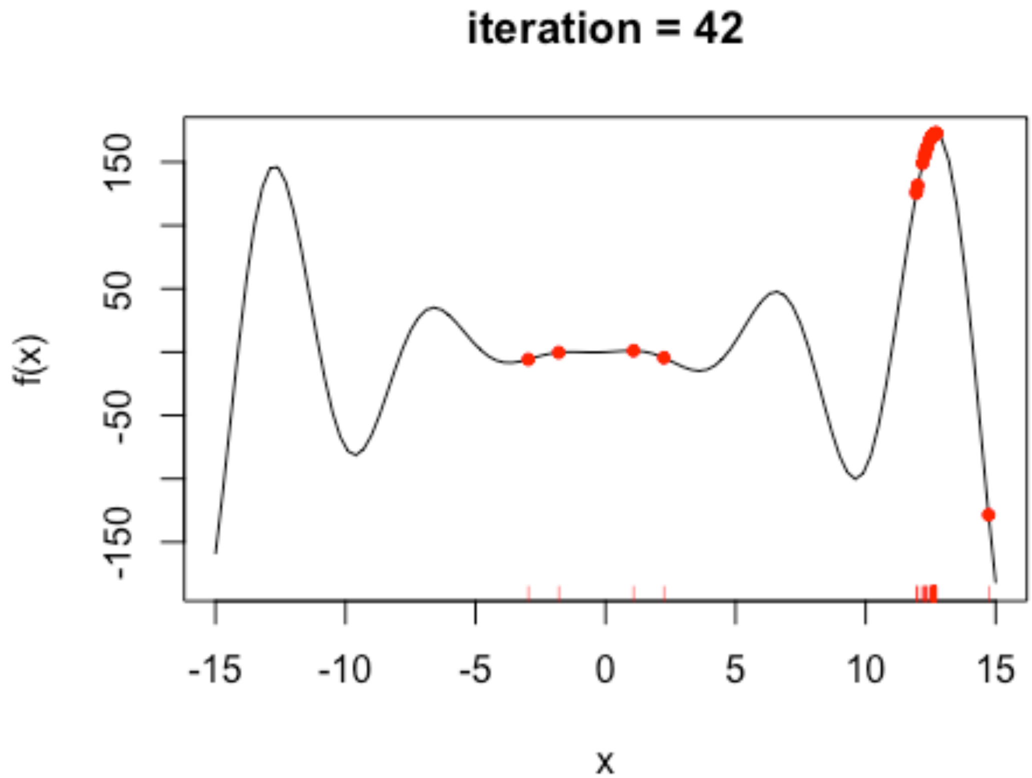


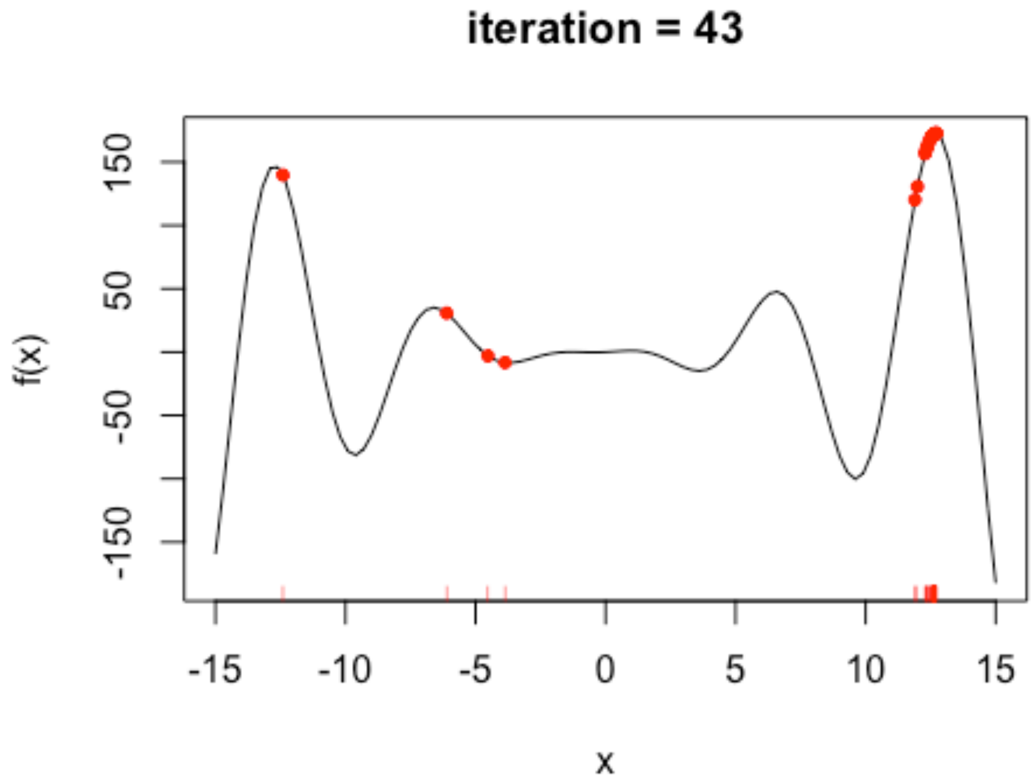
iteration = 39
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

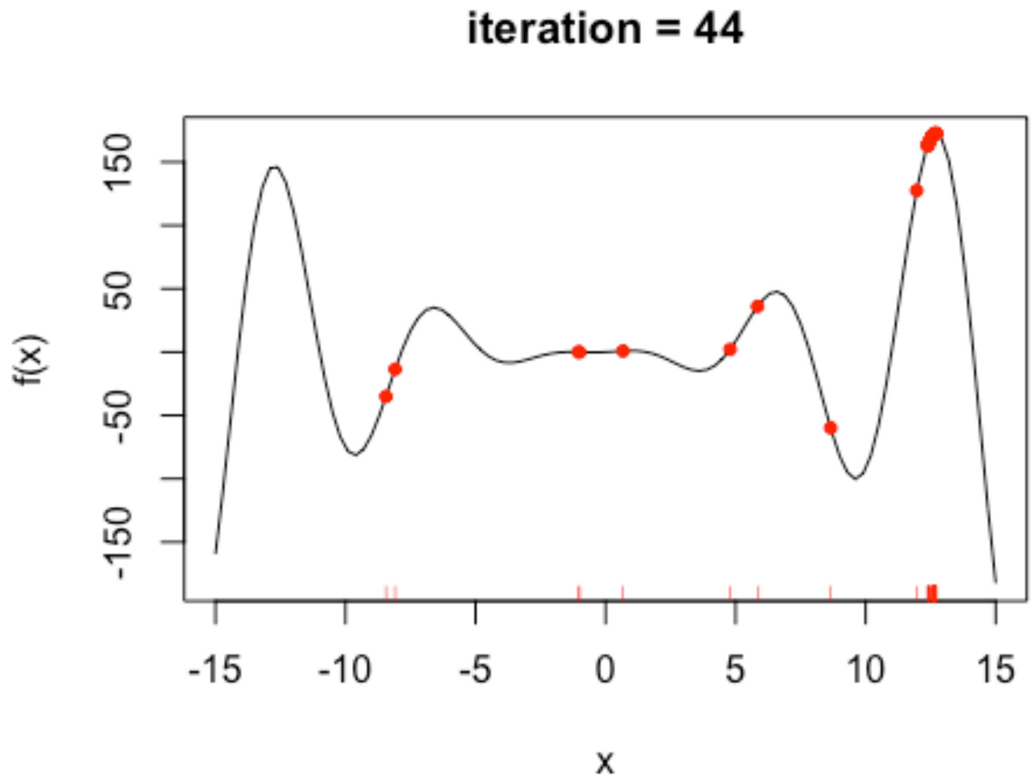

iteration = 40
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

iteration = 41

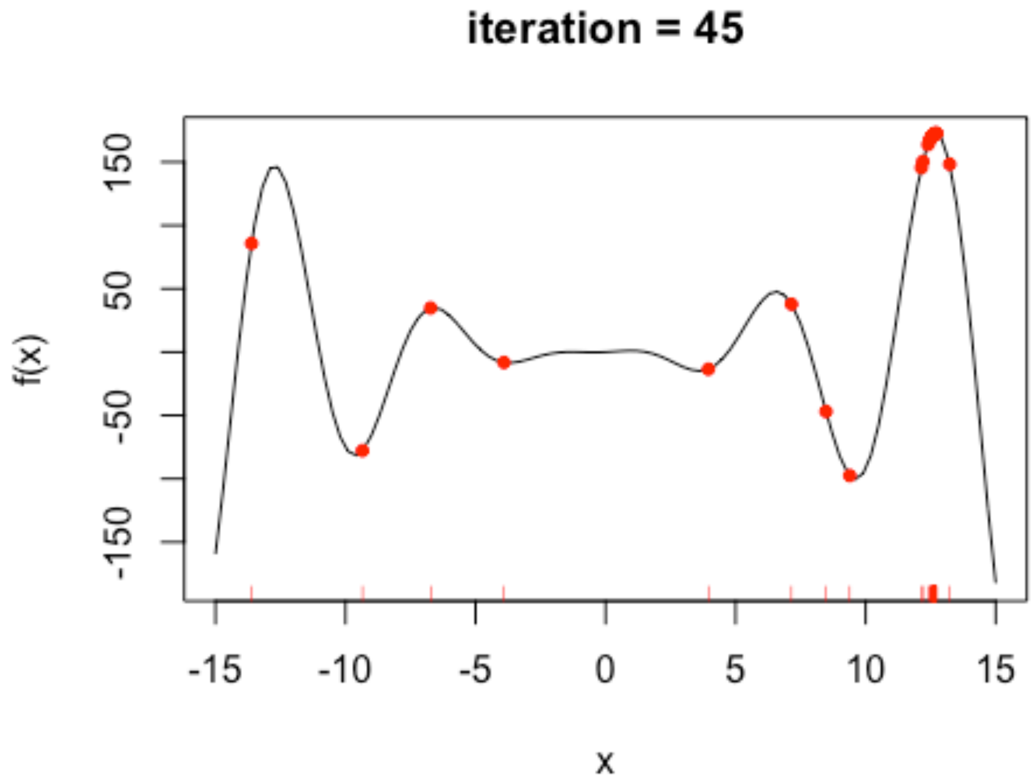



iteration = 42
150
50
-50
-150
f(x)
-15
-10
-5
0
5
10
15
x

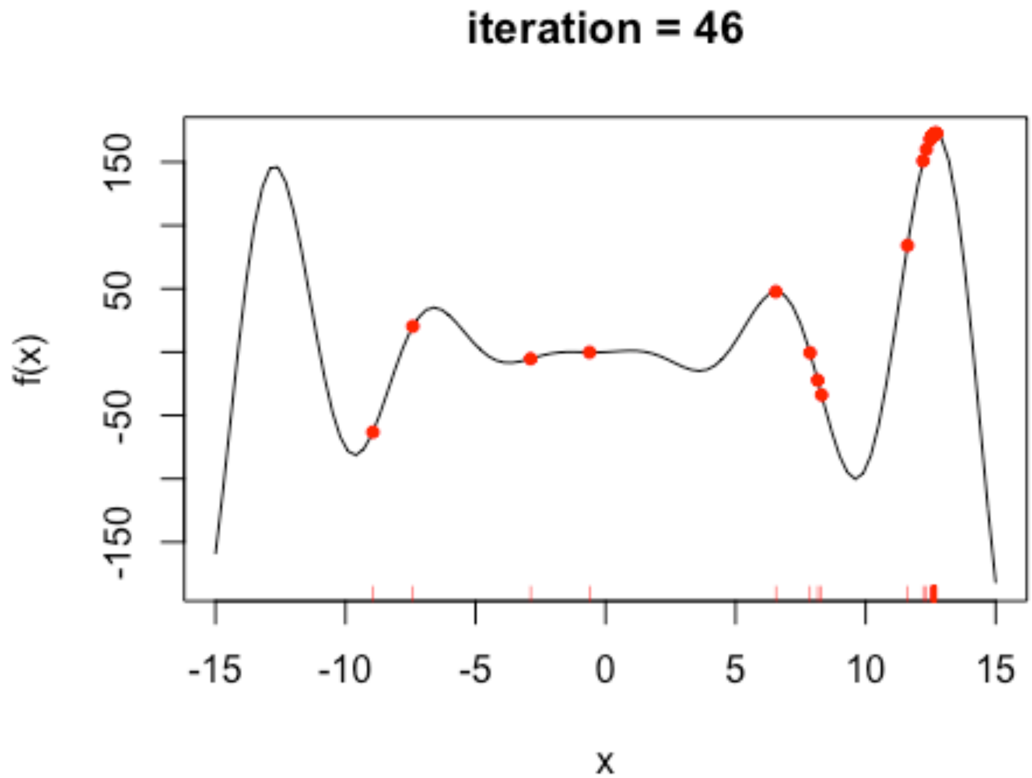

iteration = 43
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

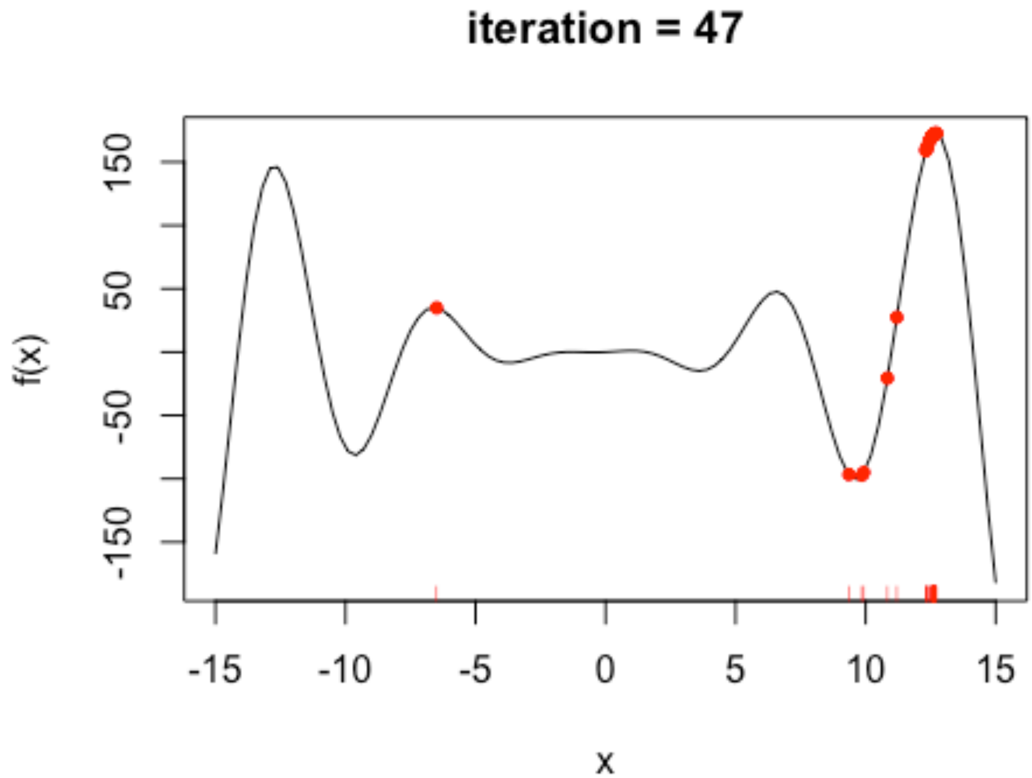


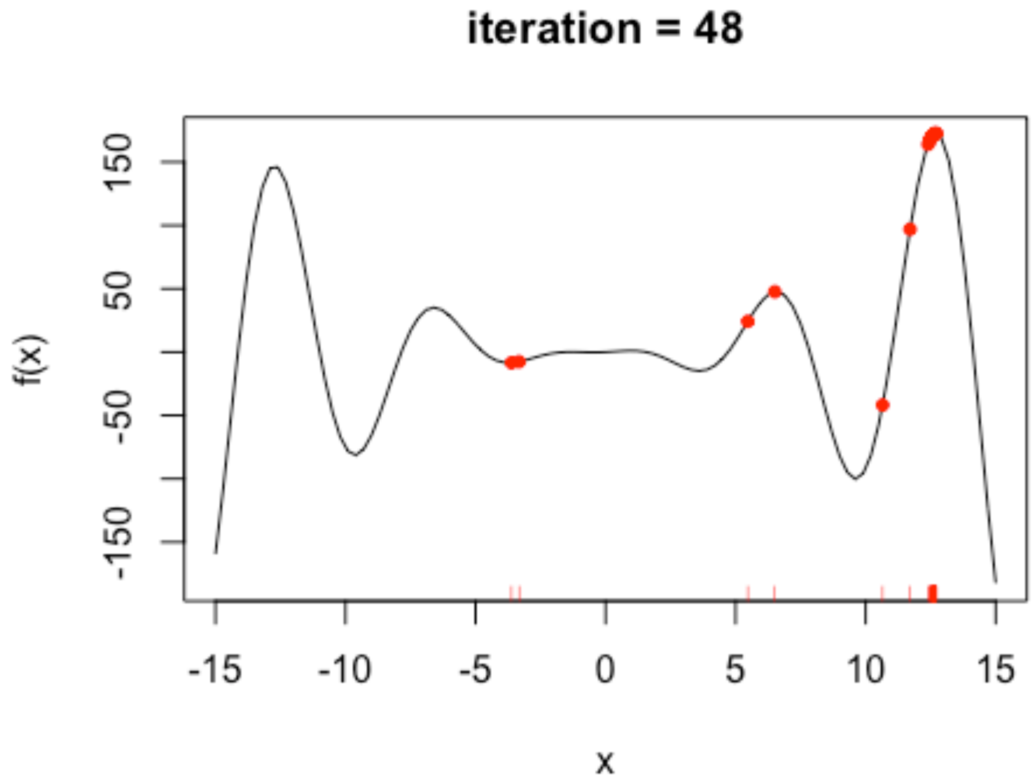
iteration = 44
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

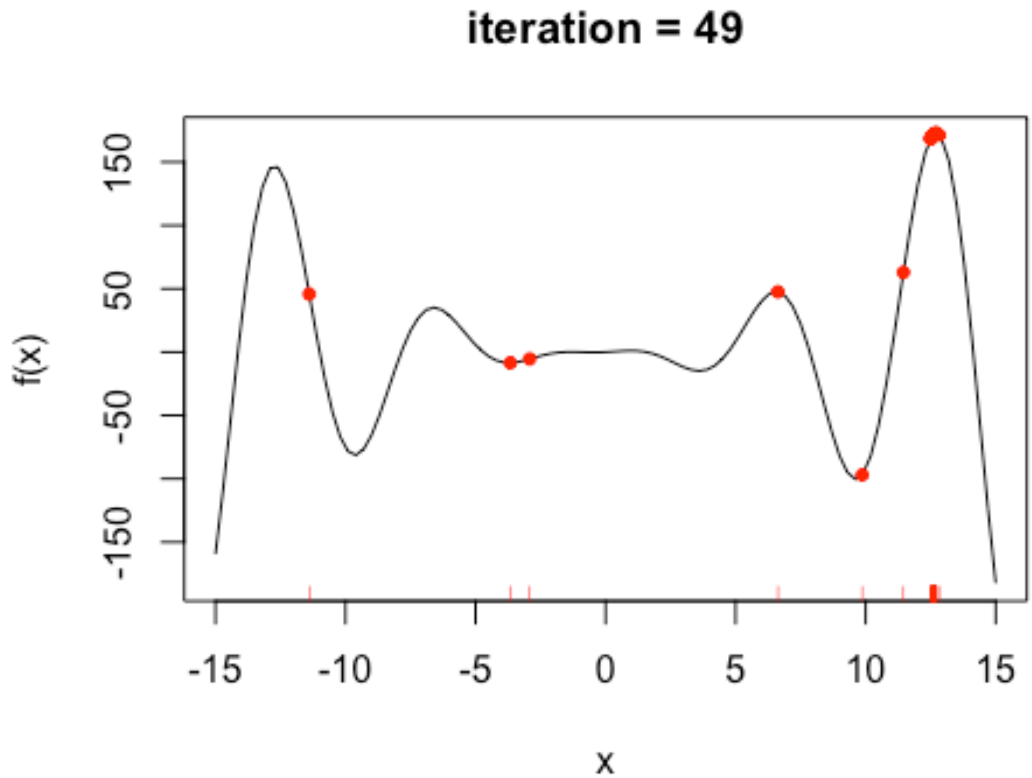


iteration = 45
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

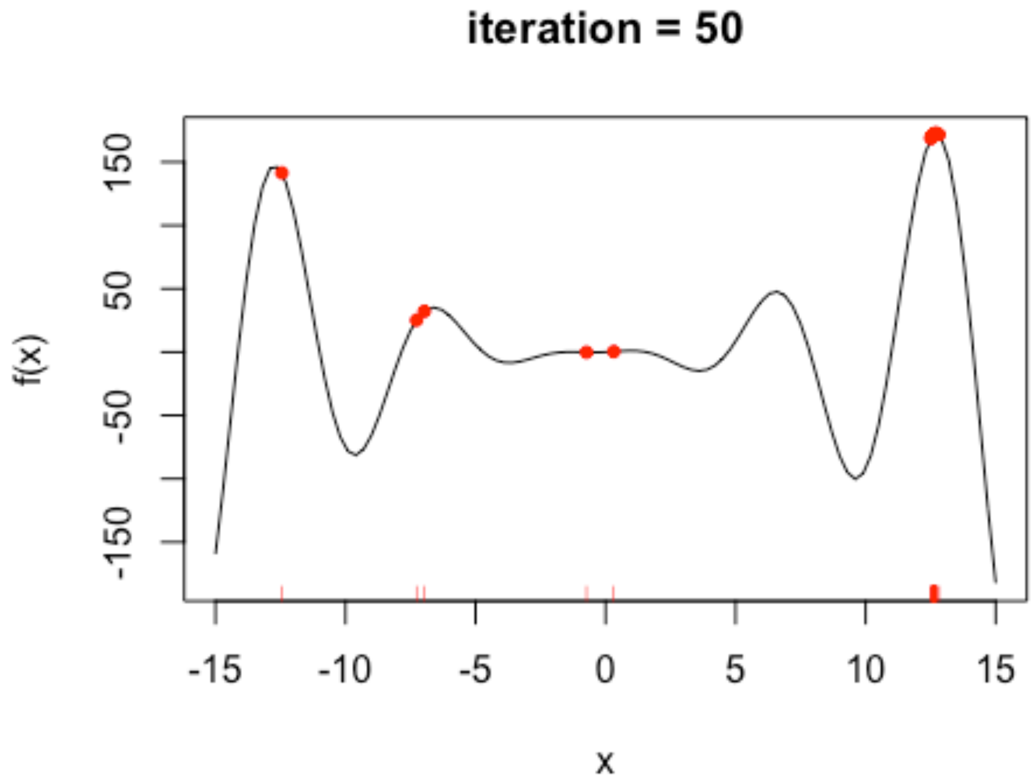

iteration = 46
f(x)
x
150
50
-50
-150
-15
-10
-5
0
5
10
15


iteration = 47
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

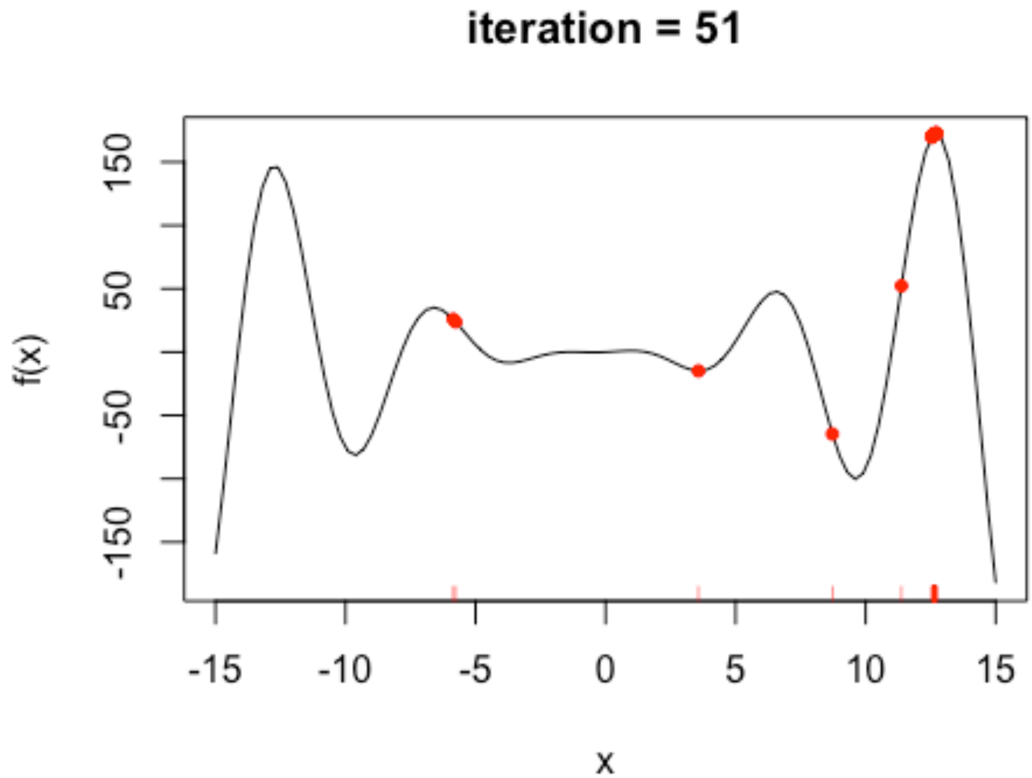

iteration = 48
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

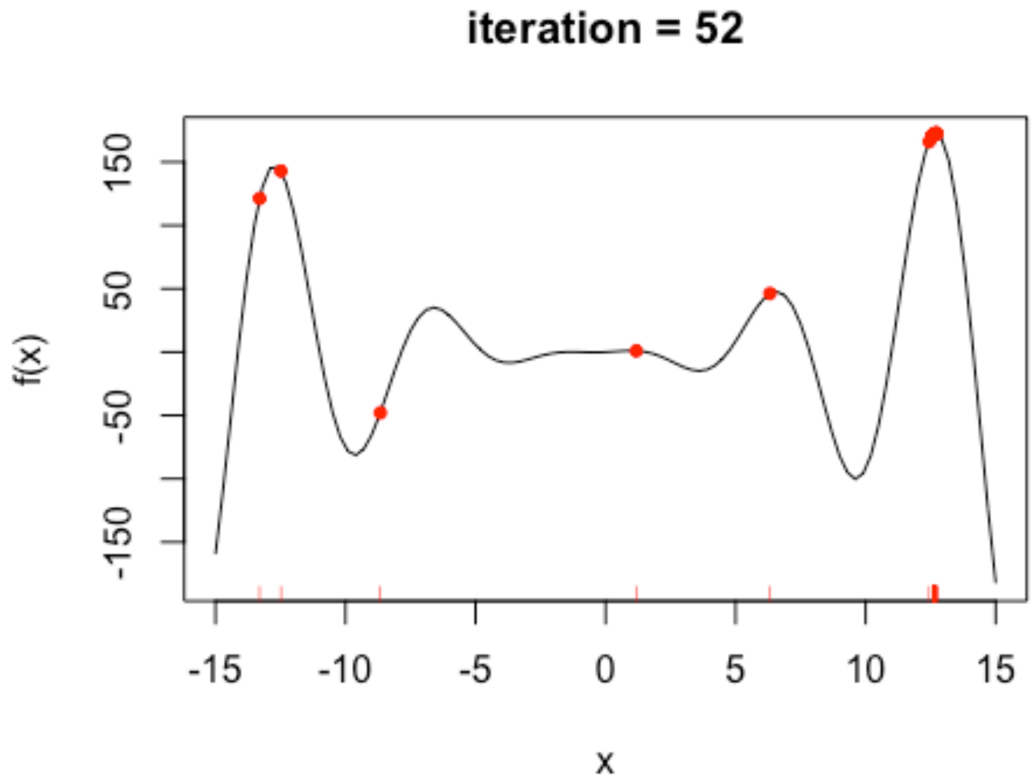


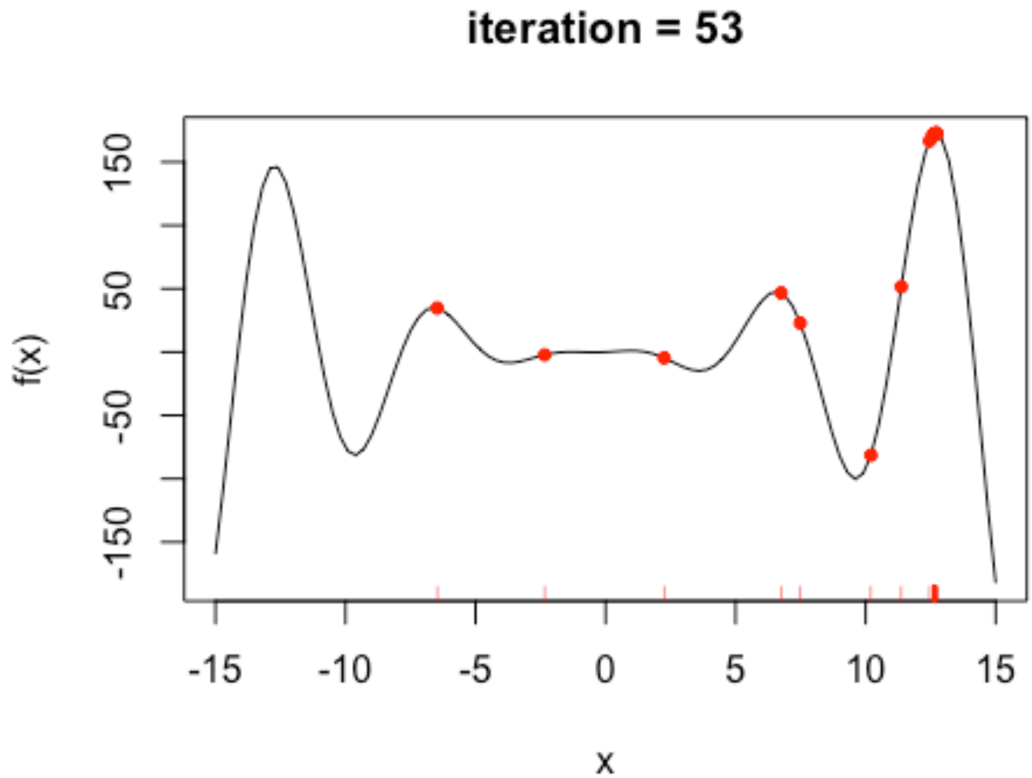
iteration = 49
150
50
-50
-150
f(x)
-15
-10
-5
0
5
10
15
x



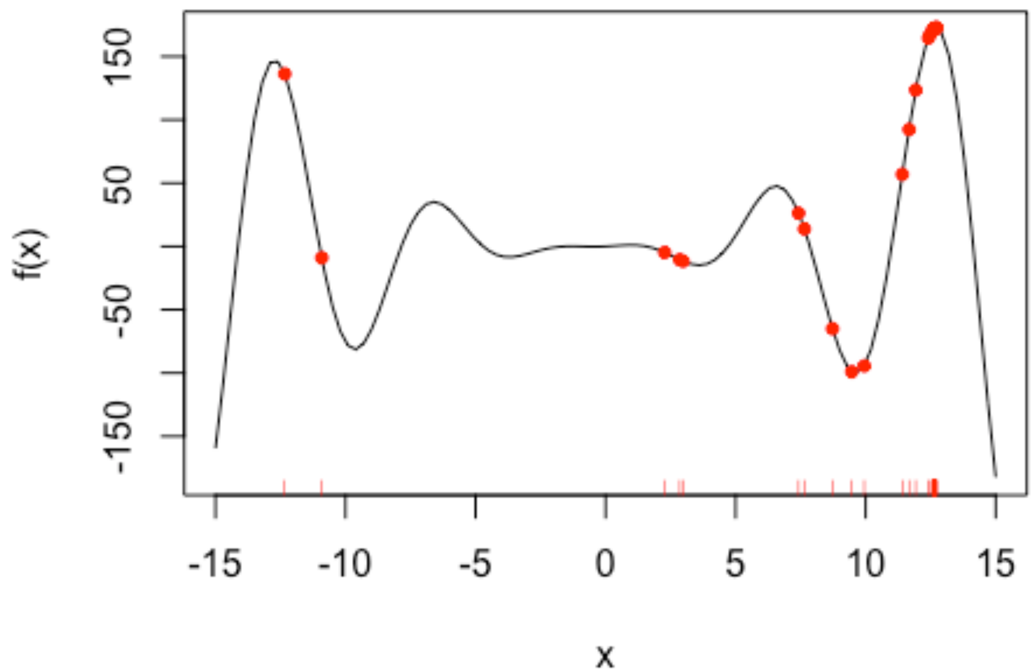
iteration = 50
150
50
-50
-150
f(x)
-15
-10
-5
0
5
10
15
x

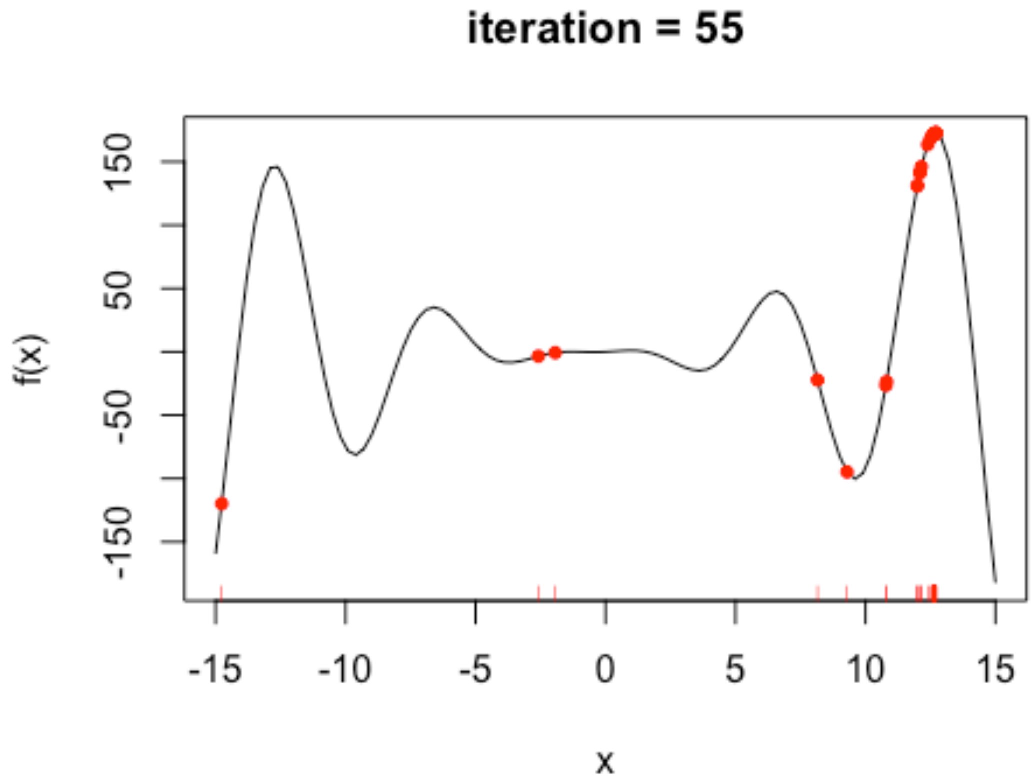

iteration = 51
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

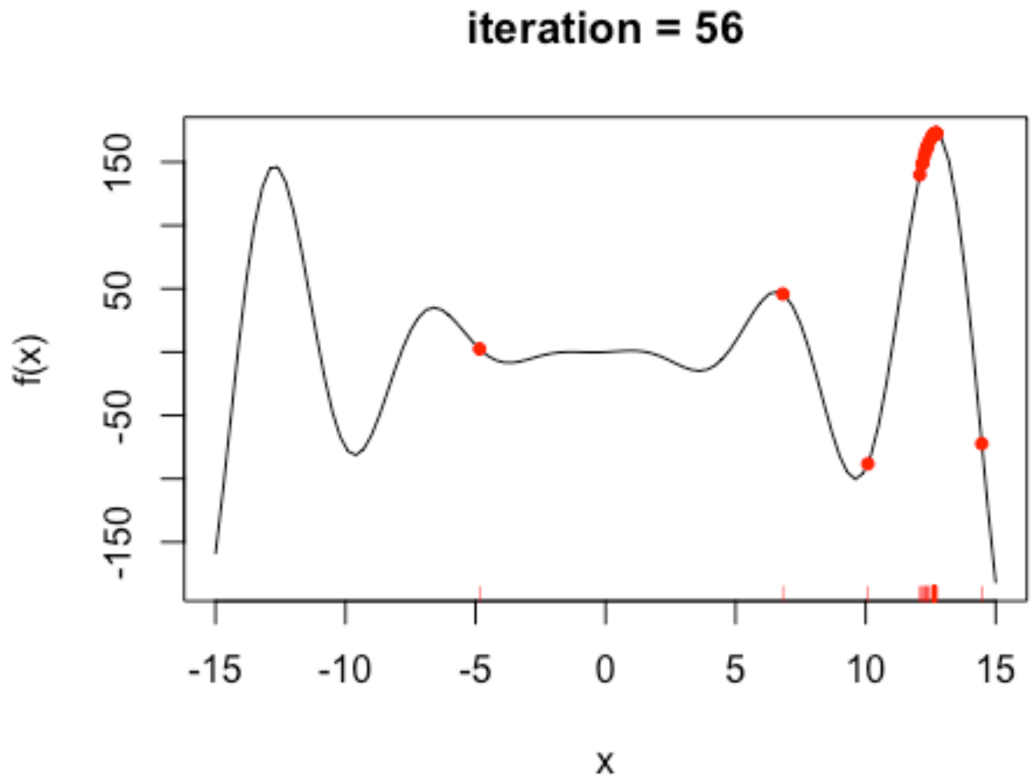

iteration = 52
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x


iteration = 53
f(x)
x
150
50
-50
-150
-15
-10
-5
0
5
10
15

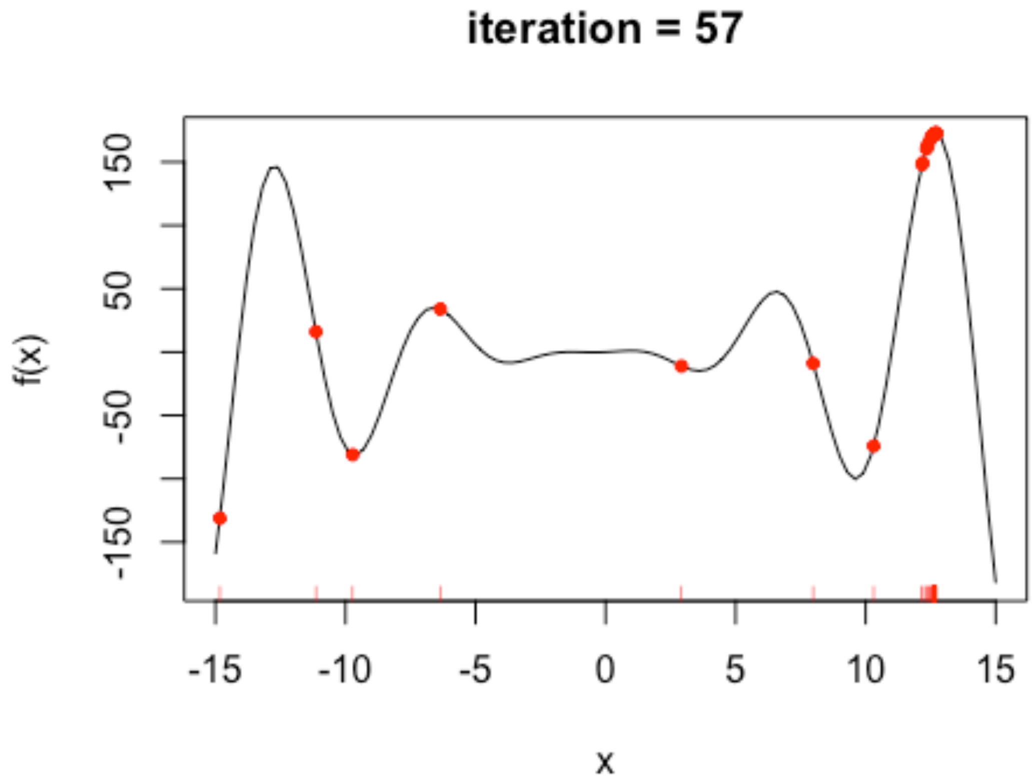
iteration = 54

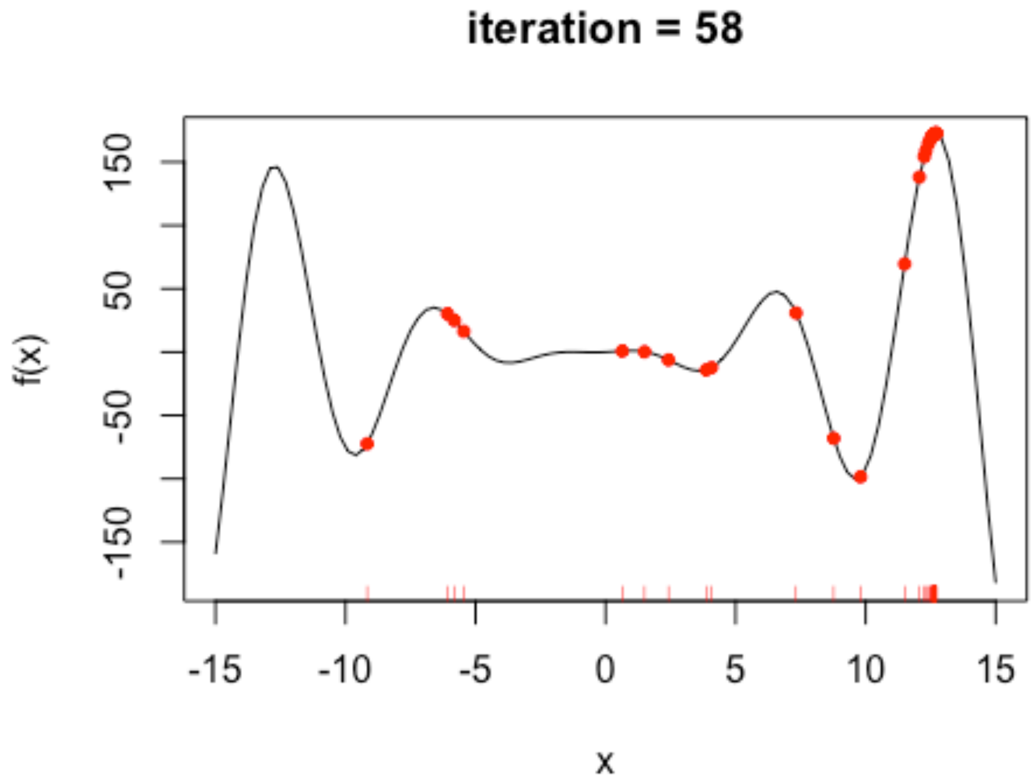


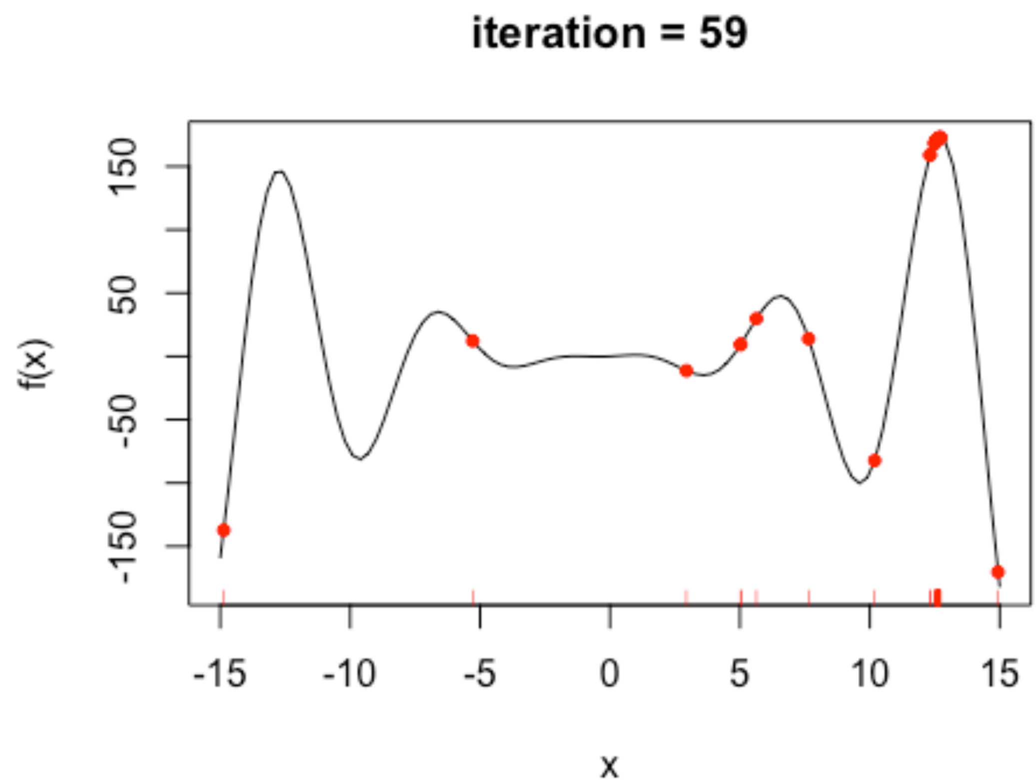

iteration = 55
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

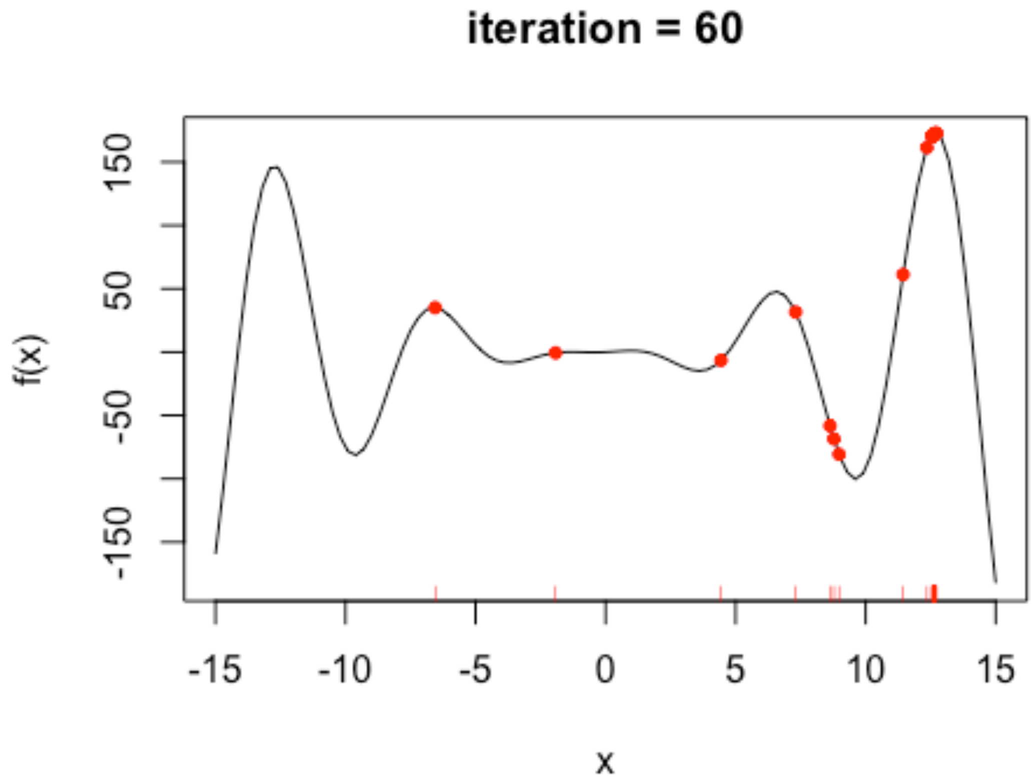


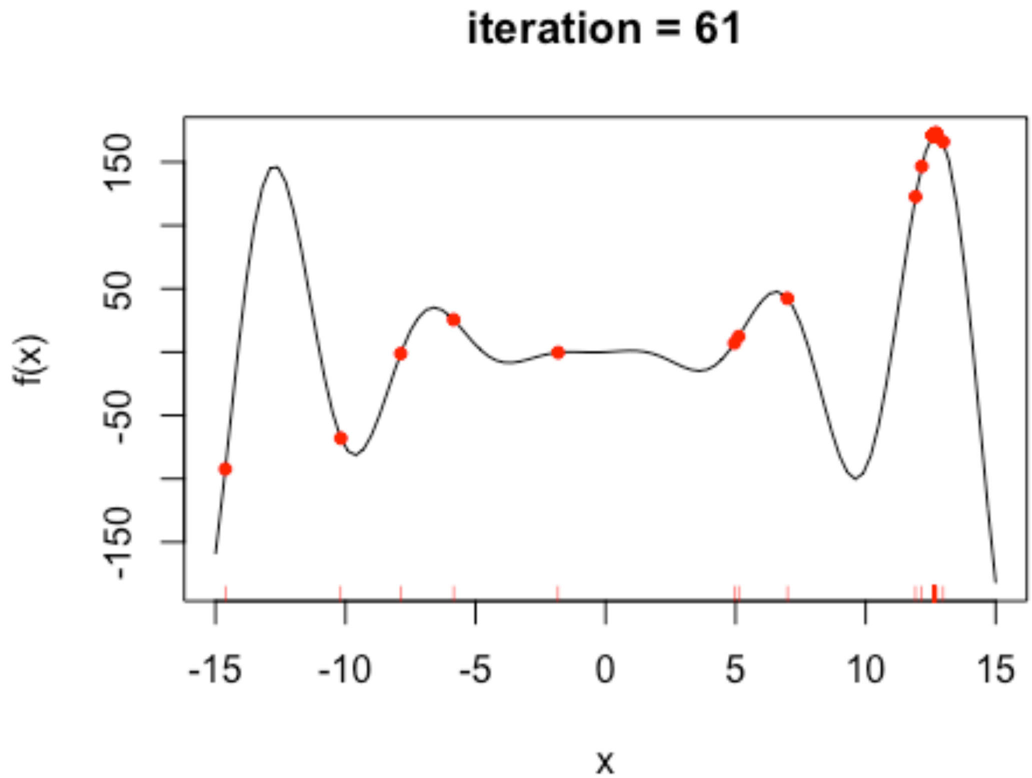
iteration = 56
f(x)
x
150
50
-50
-150
-15
-10
-5
0
5
10
15


iteration = 57
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

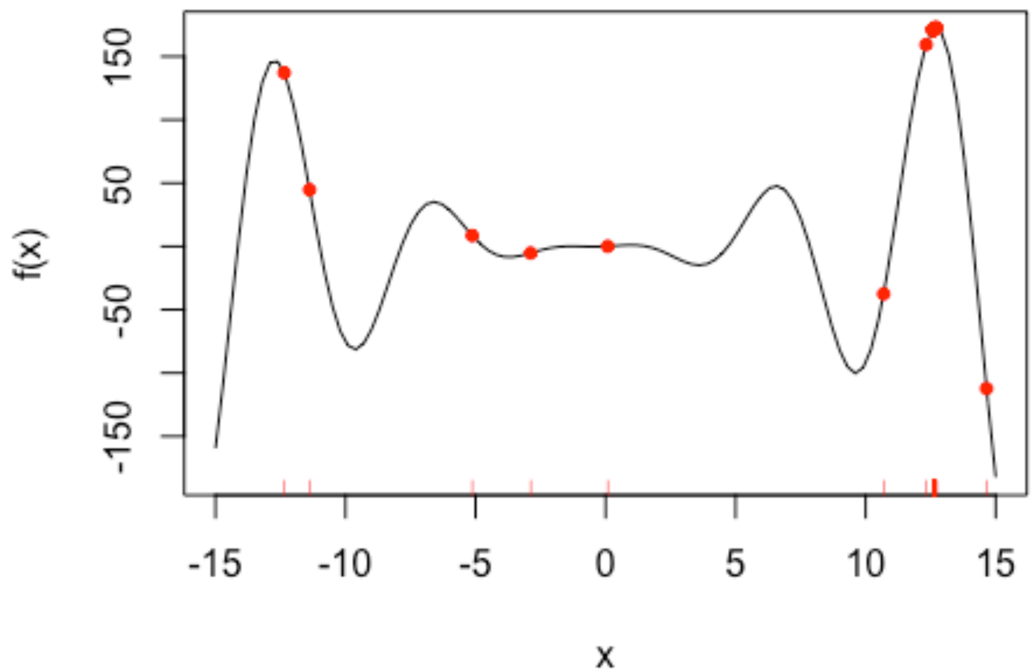

iteration = 58
f(x)
x
150
50
-50
-150
-15
-10
-5
0
5
10
15

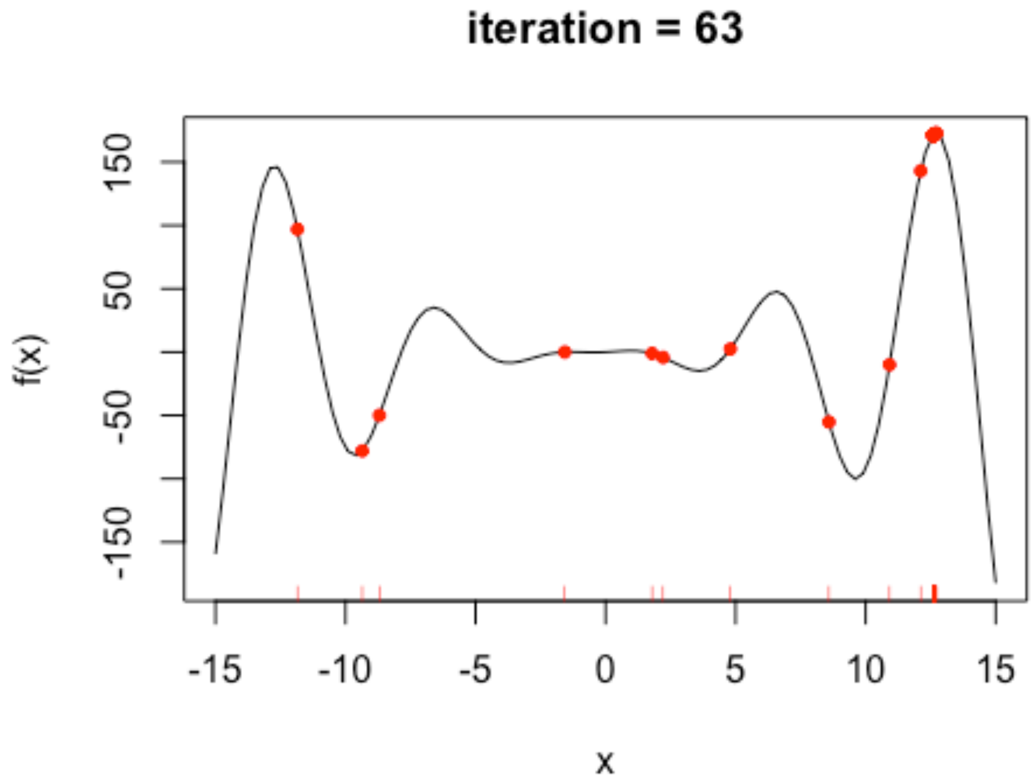

iteration = 59
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

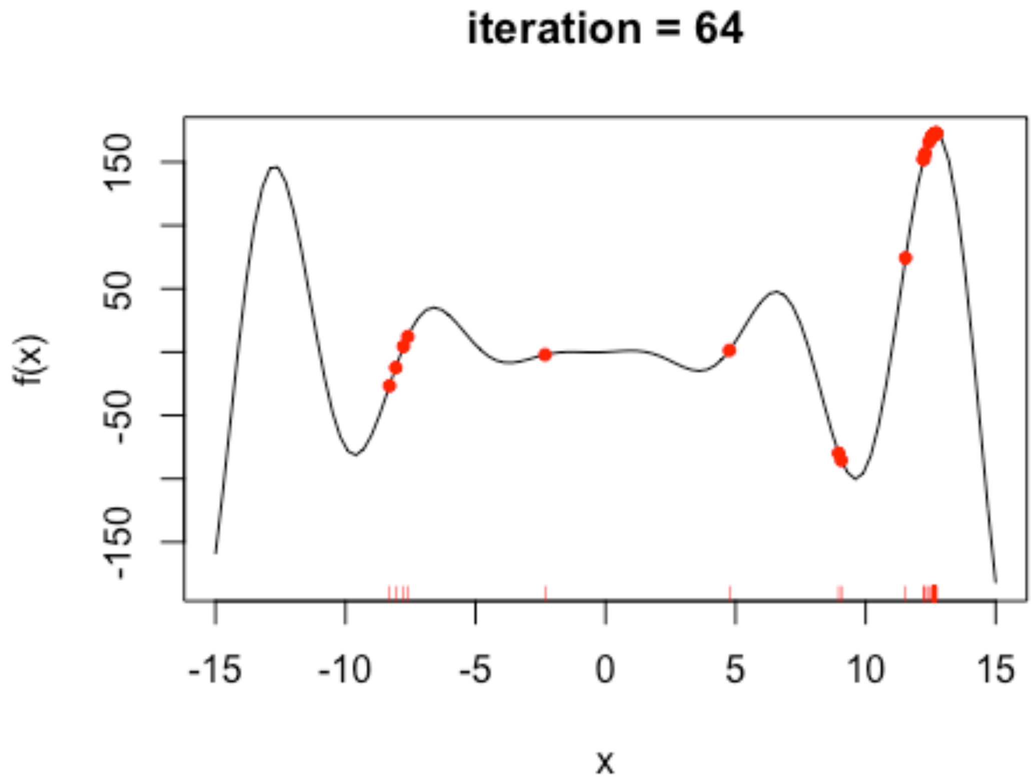

iteration = 60
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

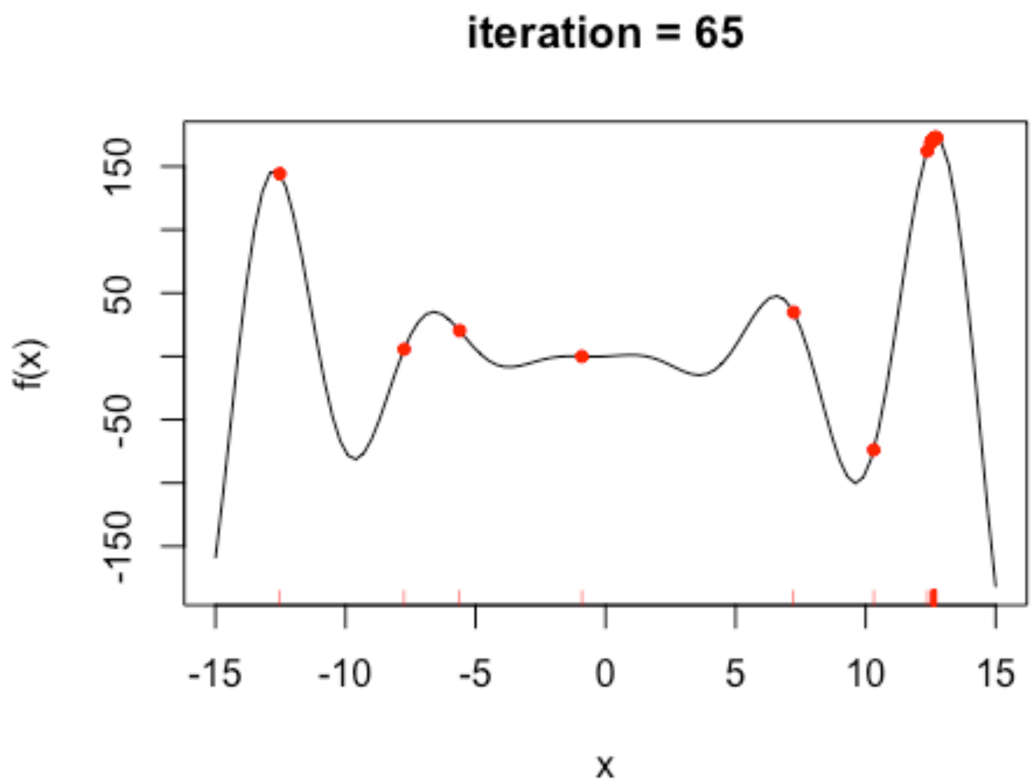

iteration = 61
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

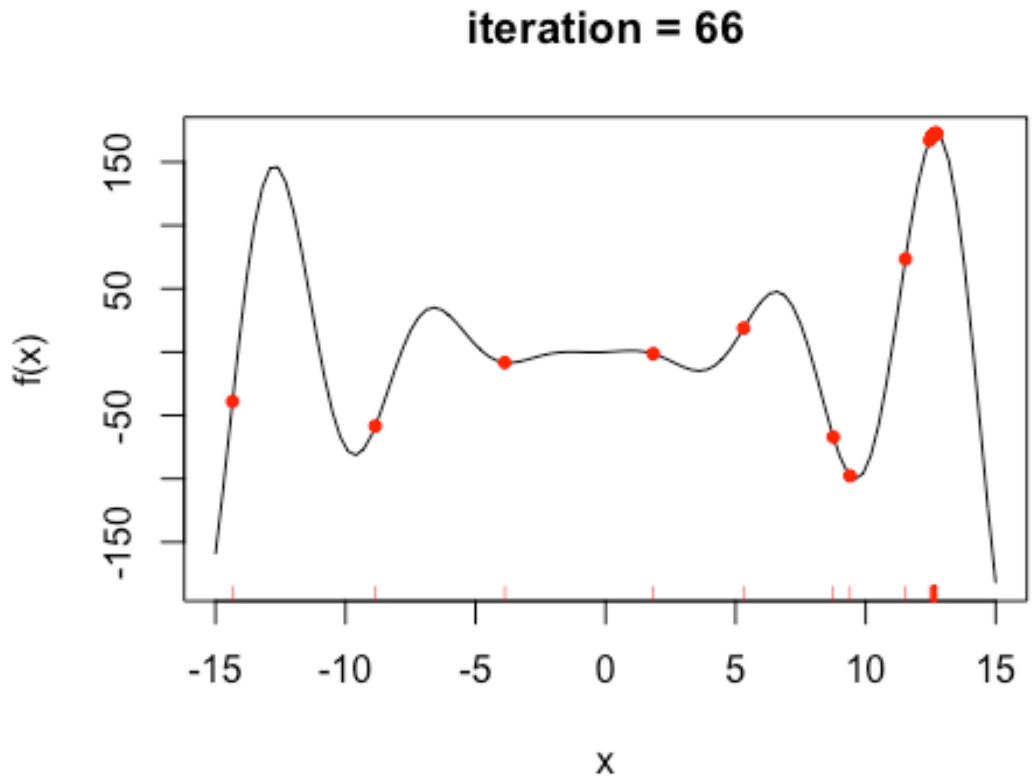
iteration = 62



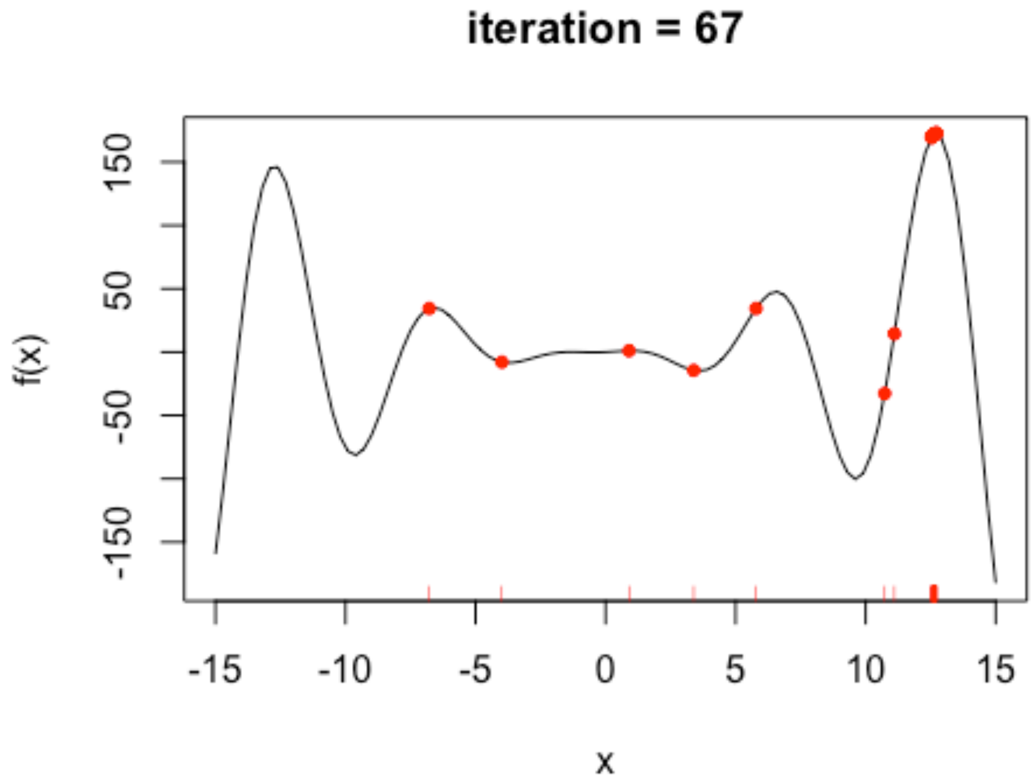

iteration = 63
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

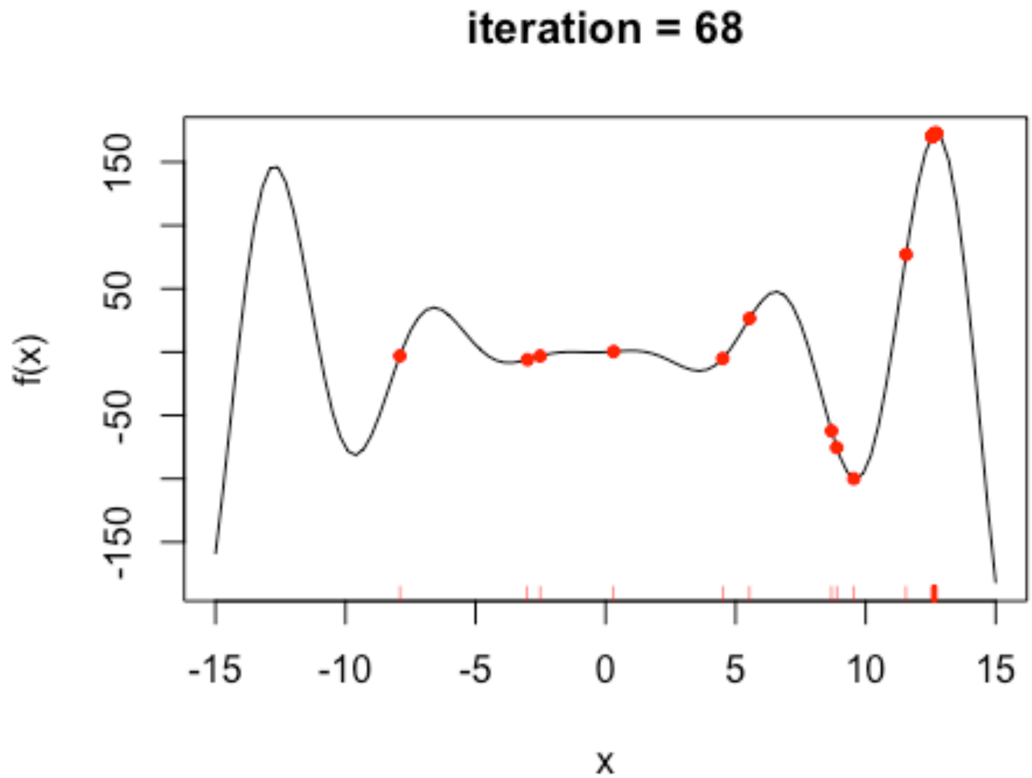

iteration = 64
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x


iteration = 65
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x



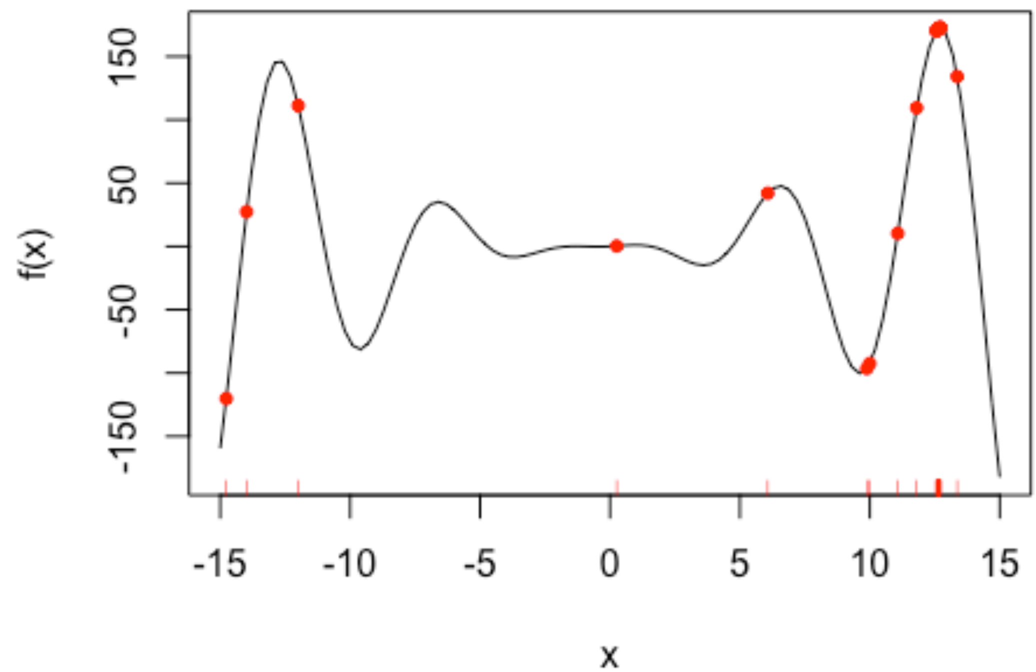
iteration = 66
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

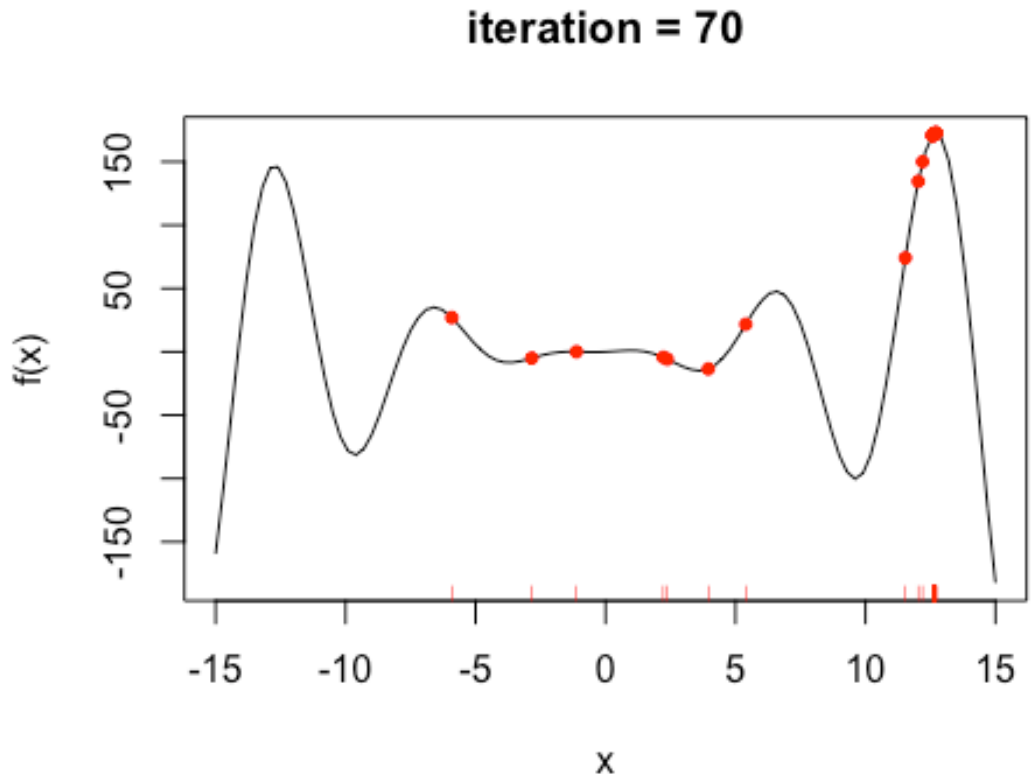

iteration = 67
f(x)
x
150
50
-50
-150
-15
-10
-5
0
5
10
15

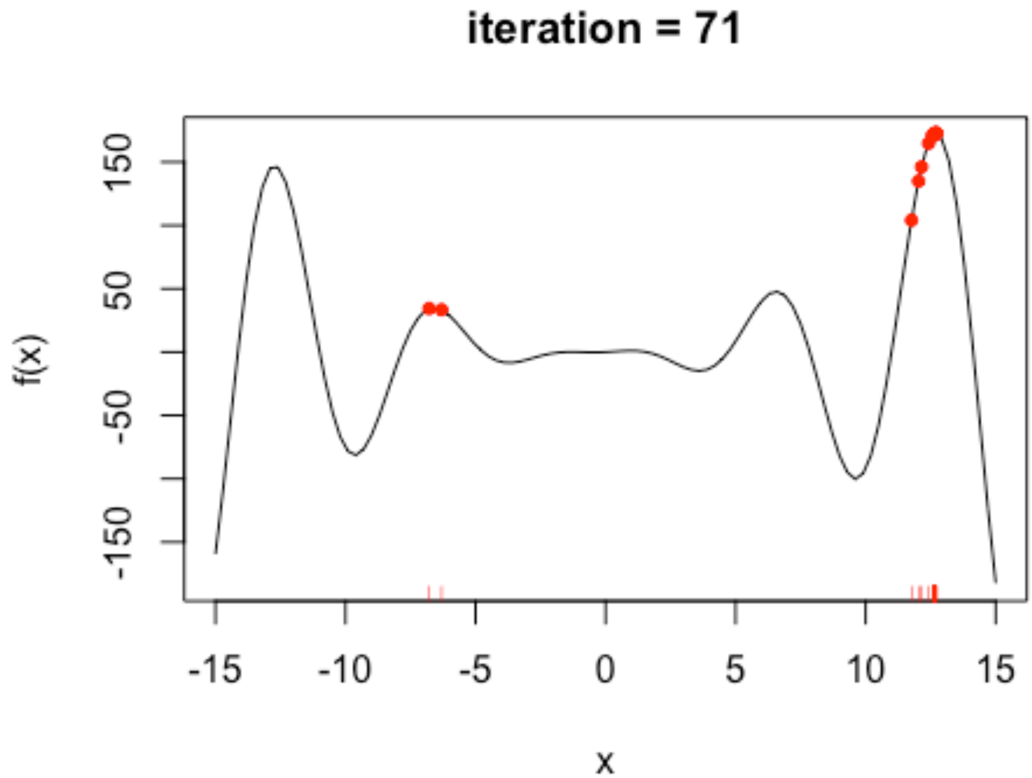


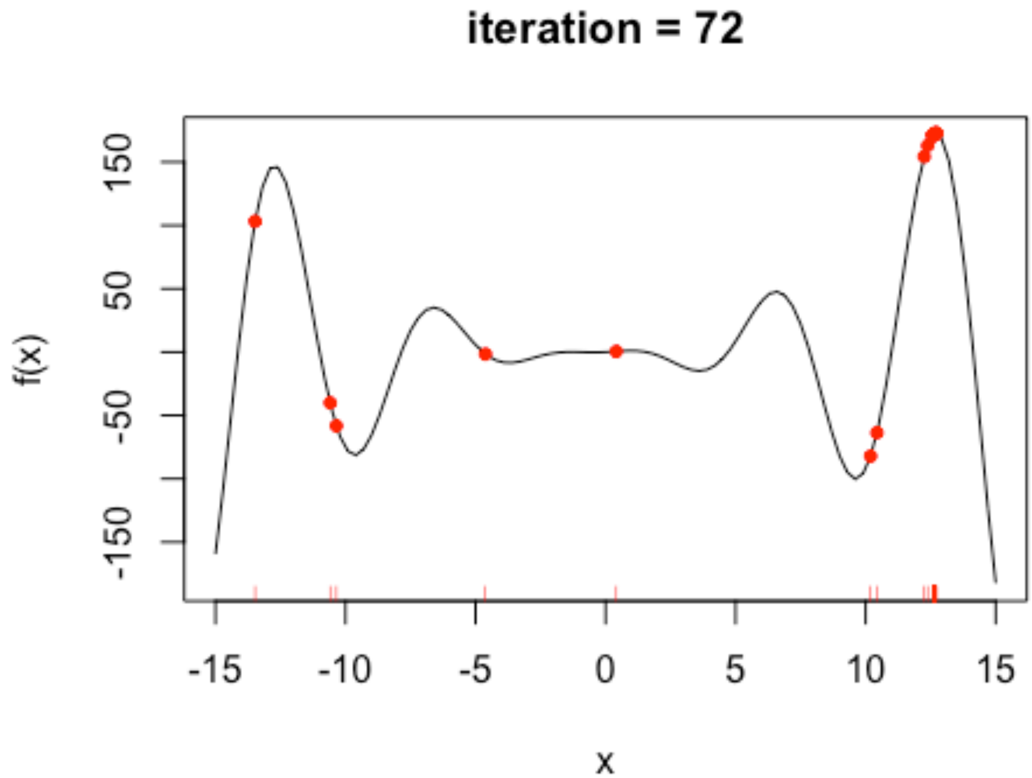
iteration = 68
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

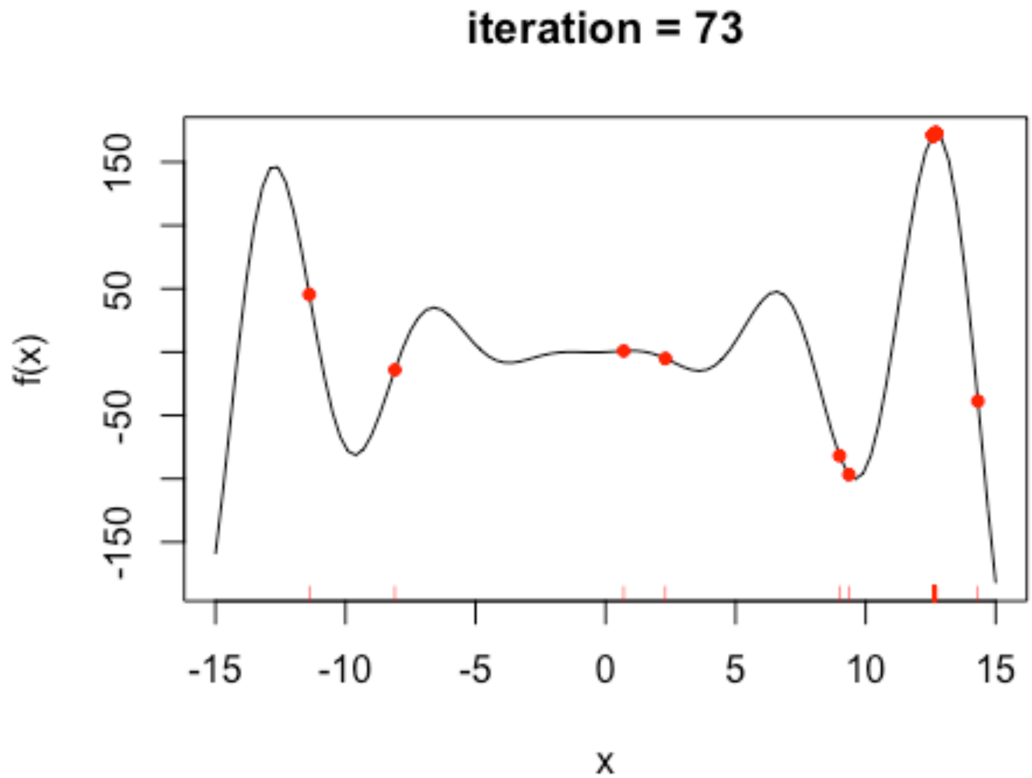
iteration = 69

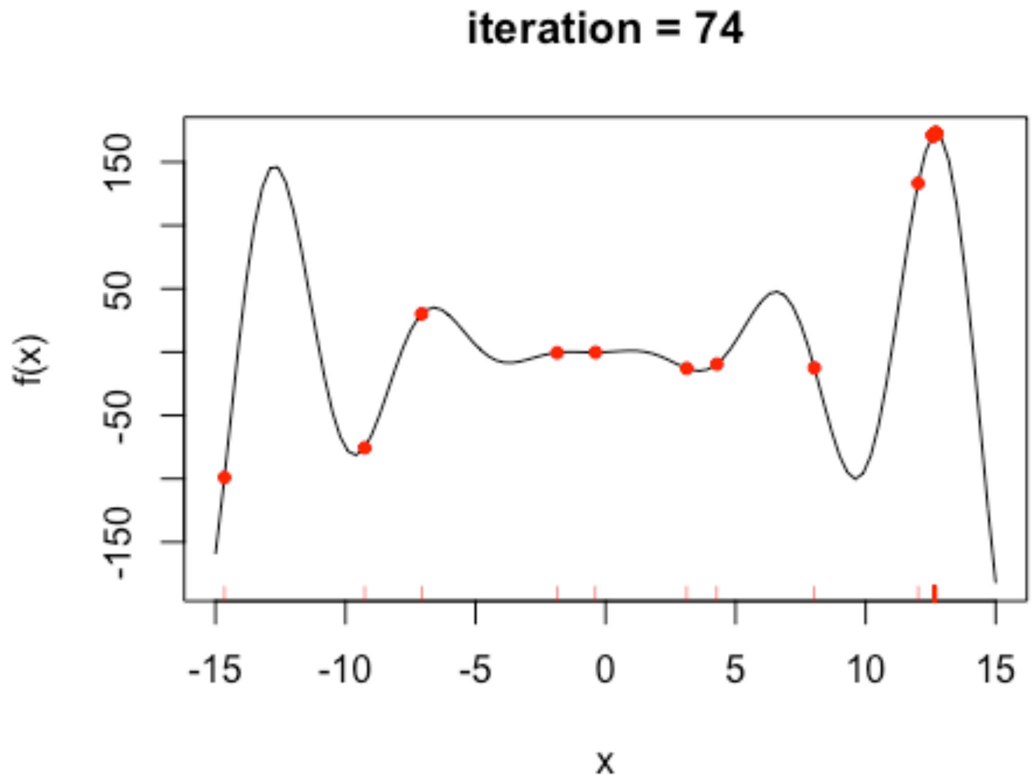


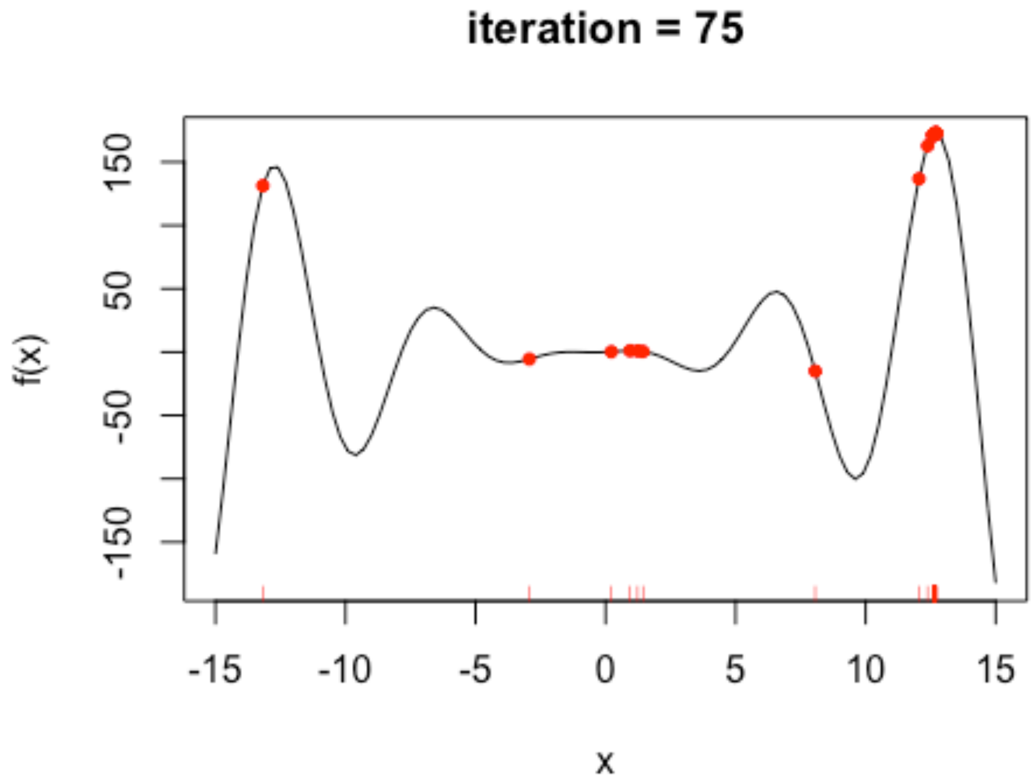

iteration = 70
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x


iteration = 71
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

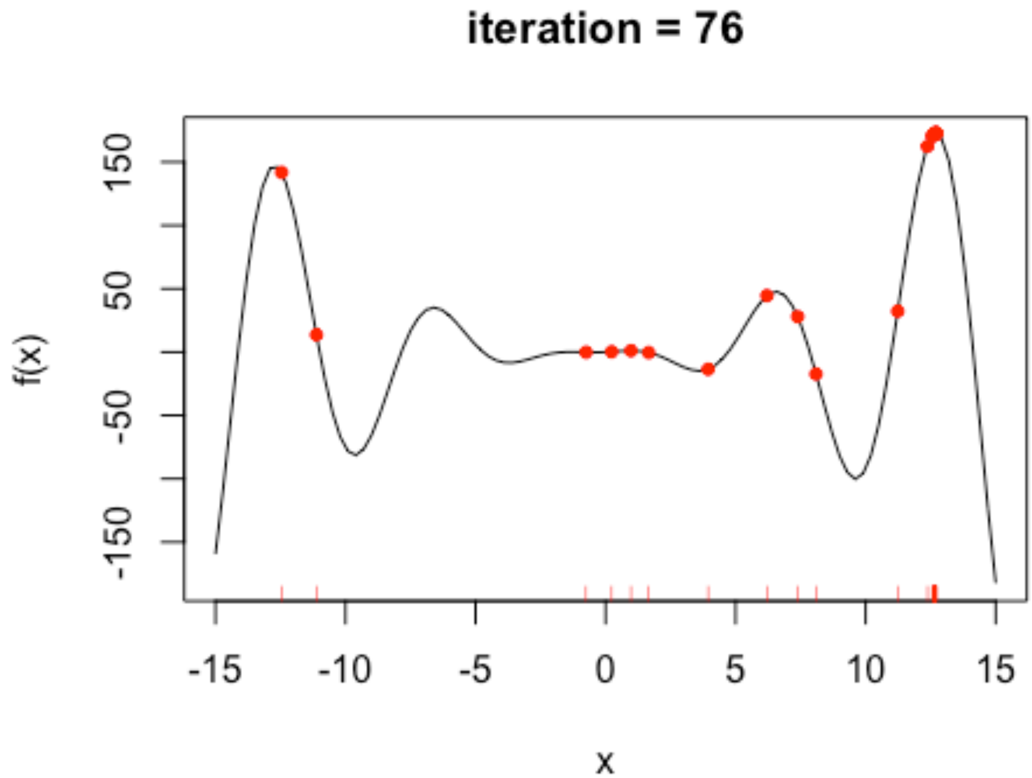

iteration = 72
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x


iteration = 73
150
50
-50
-150
f(x)
-15
-10
-5
0
5
10
15
x

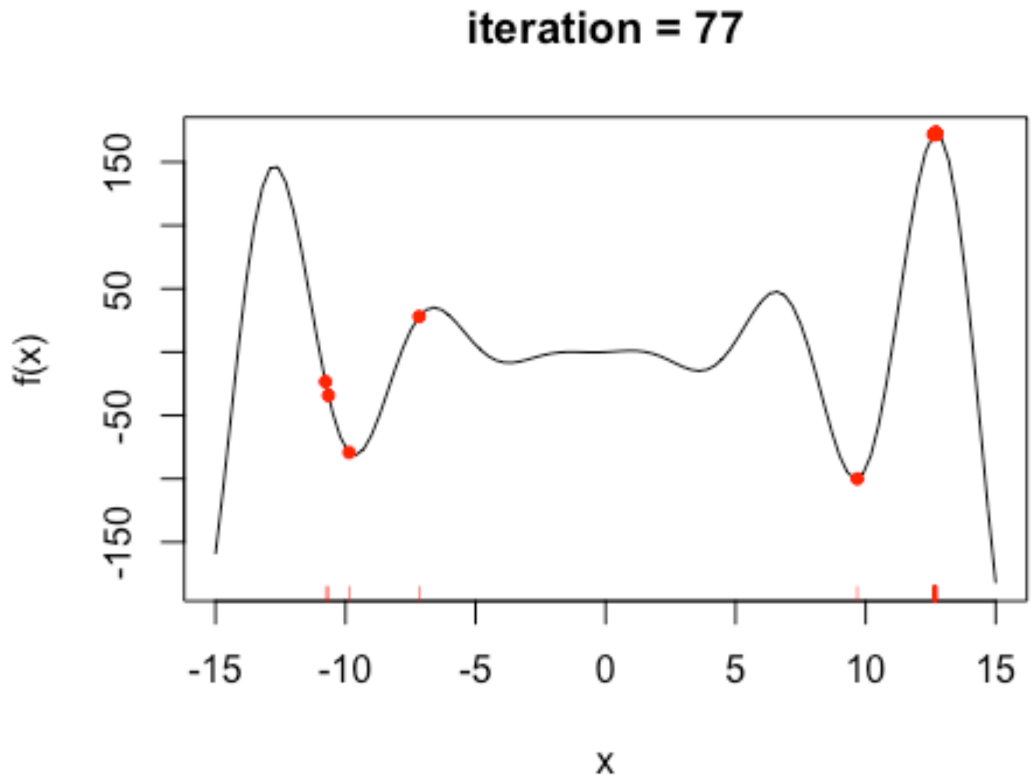

iteration = 74
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

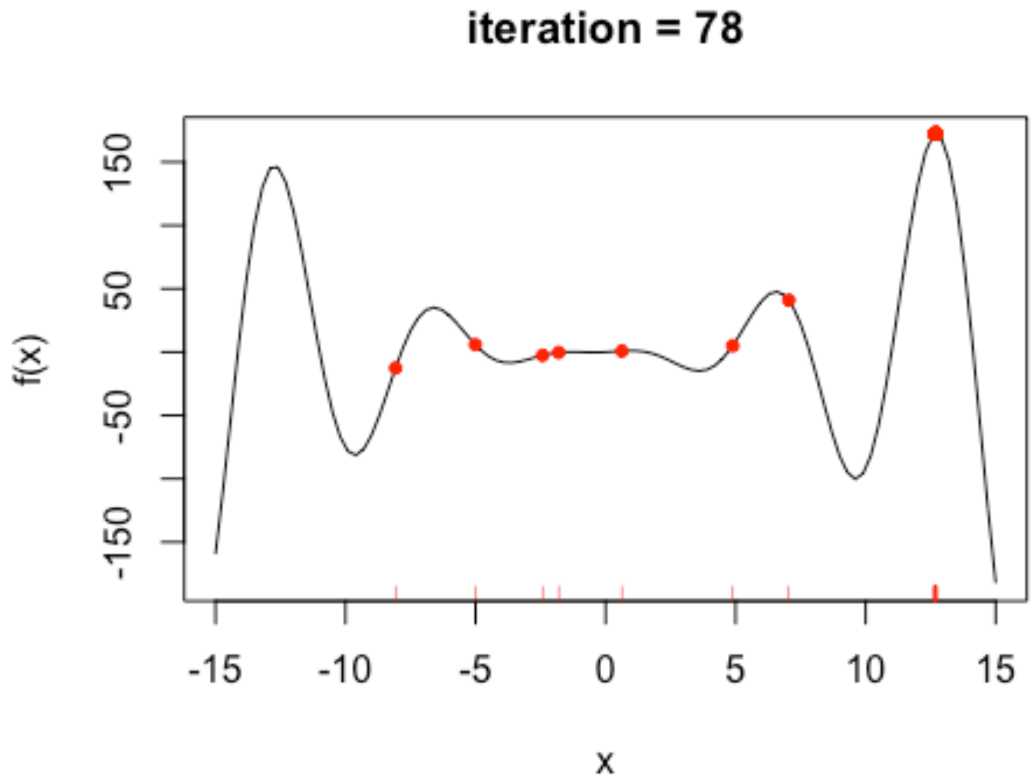


iteration = 75
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

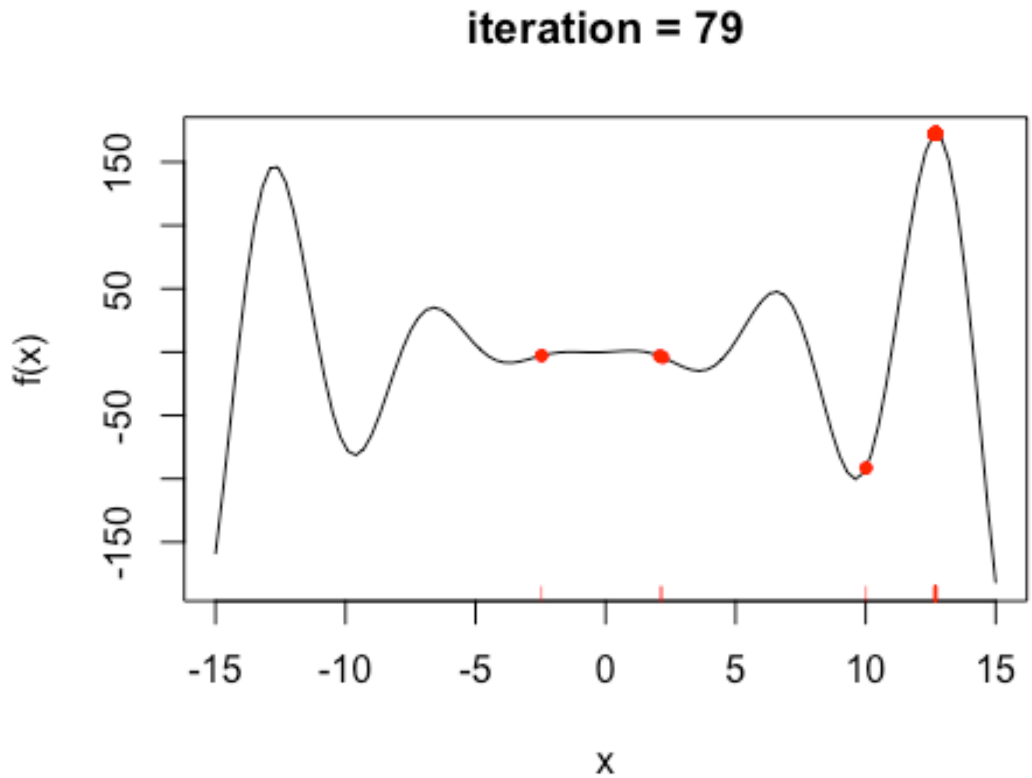


iteration = 76
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

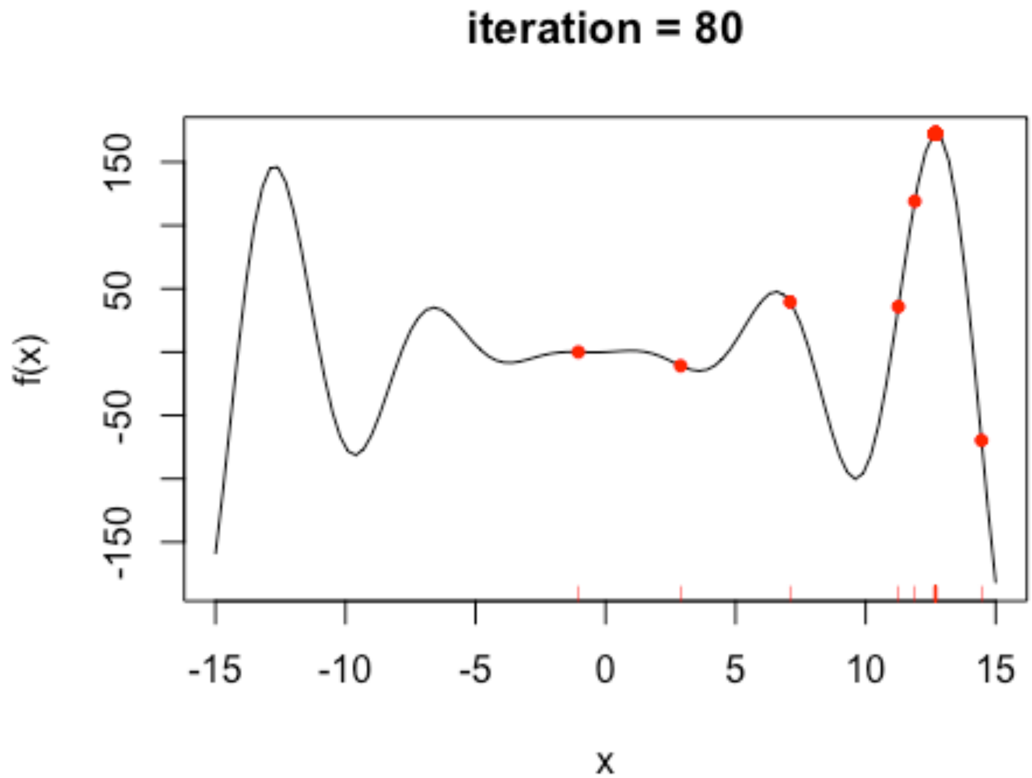

iteration = 77
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

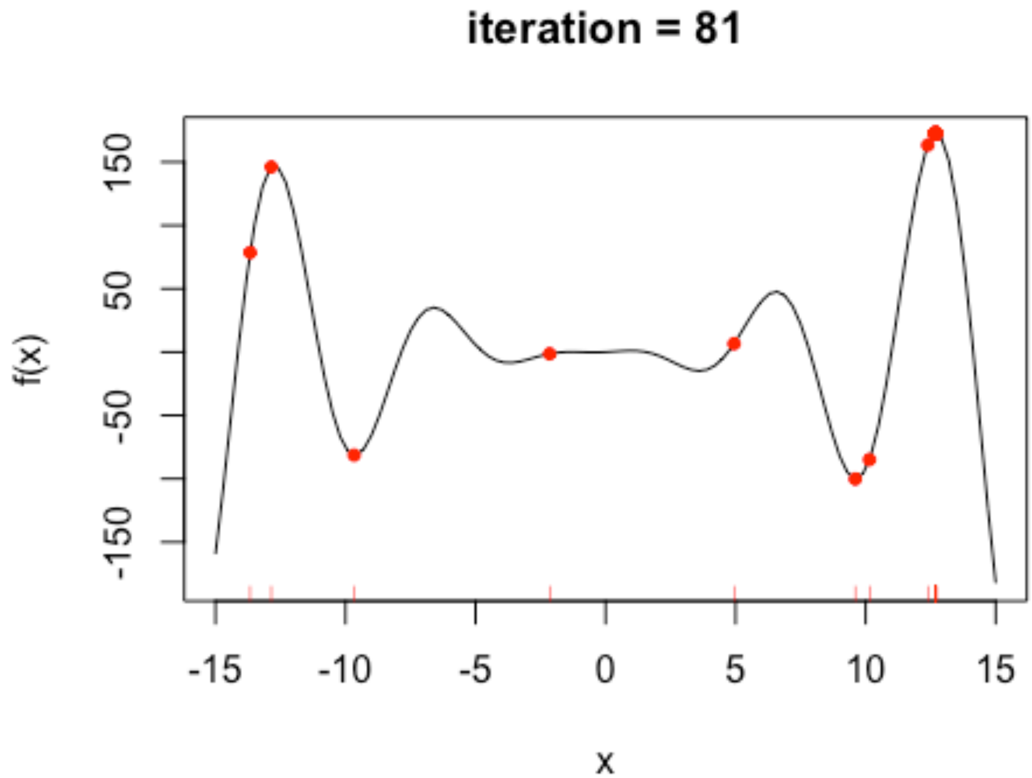


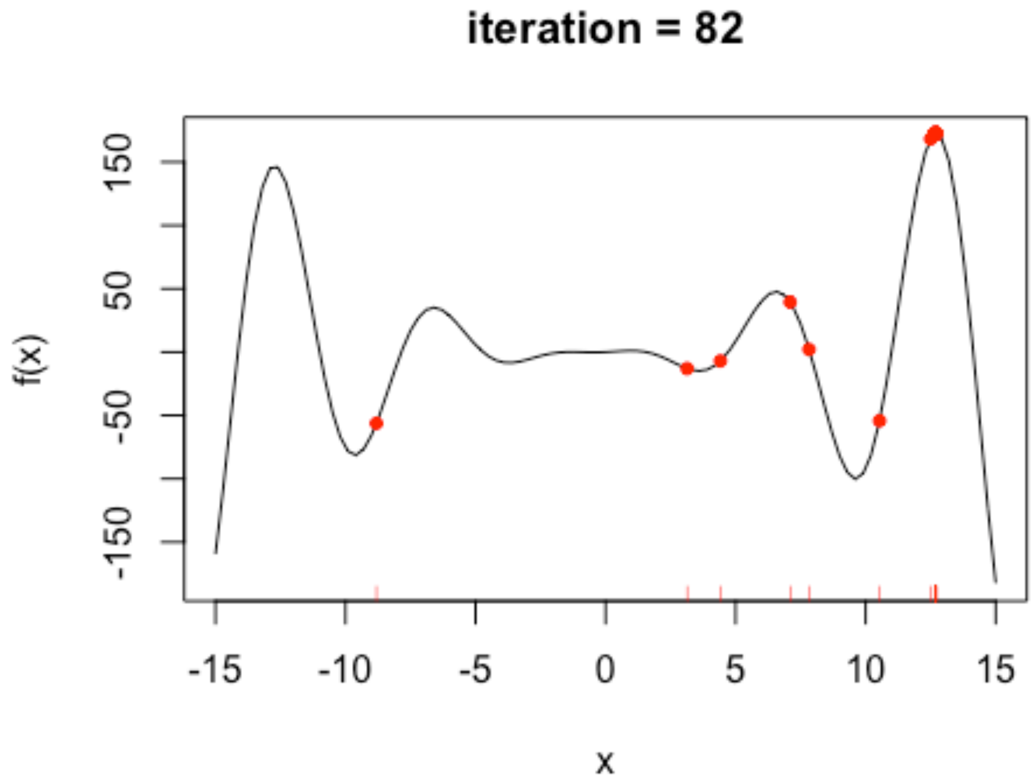
iteration = 78
150
50
-50
-150
f(x)
-15
-10
-5
0
5
10
15
x



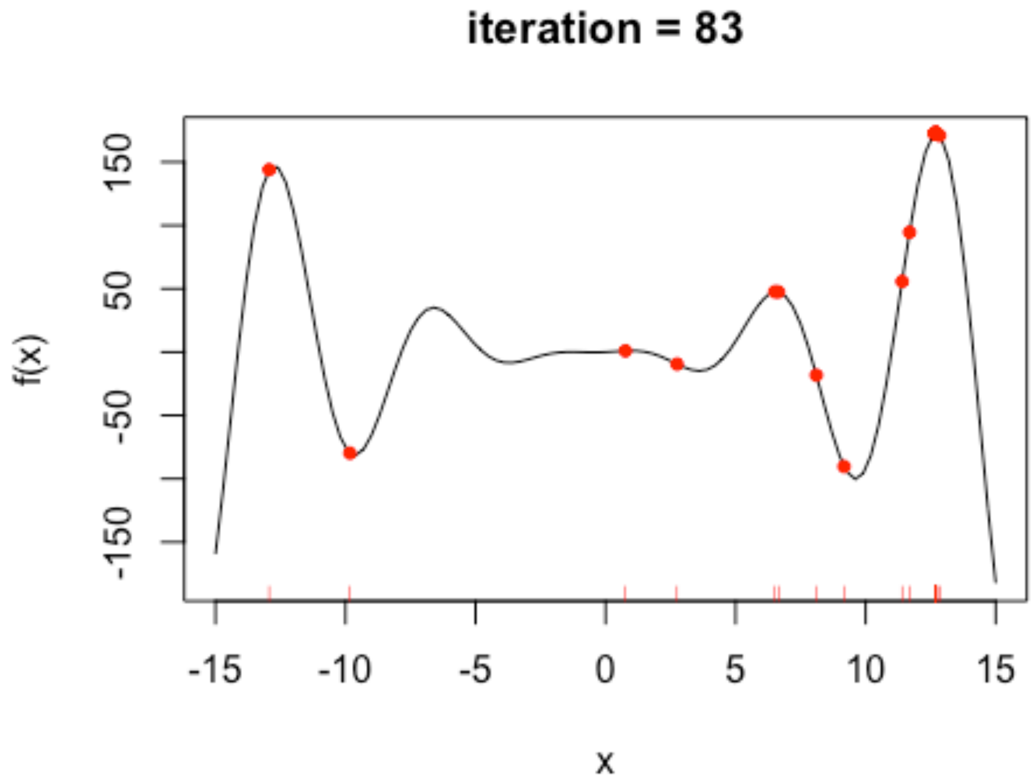
iteration = 79
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

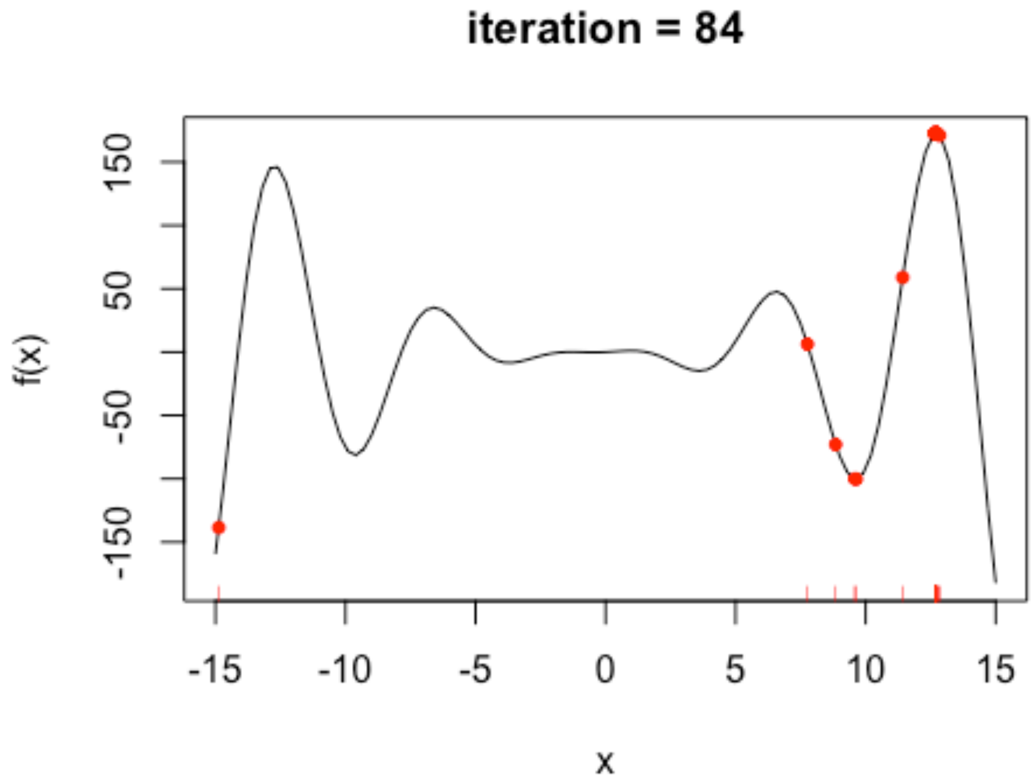

iteration = 80
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

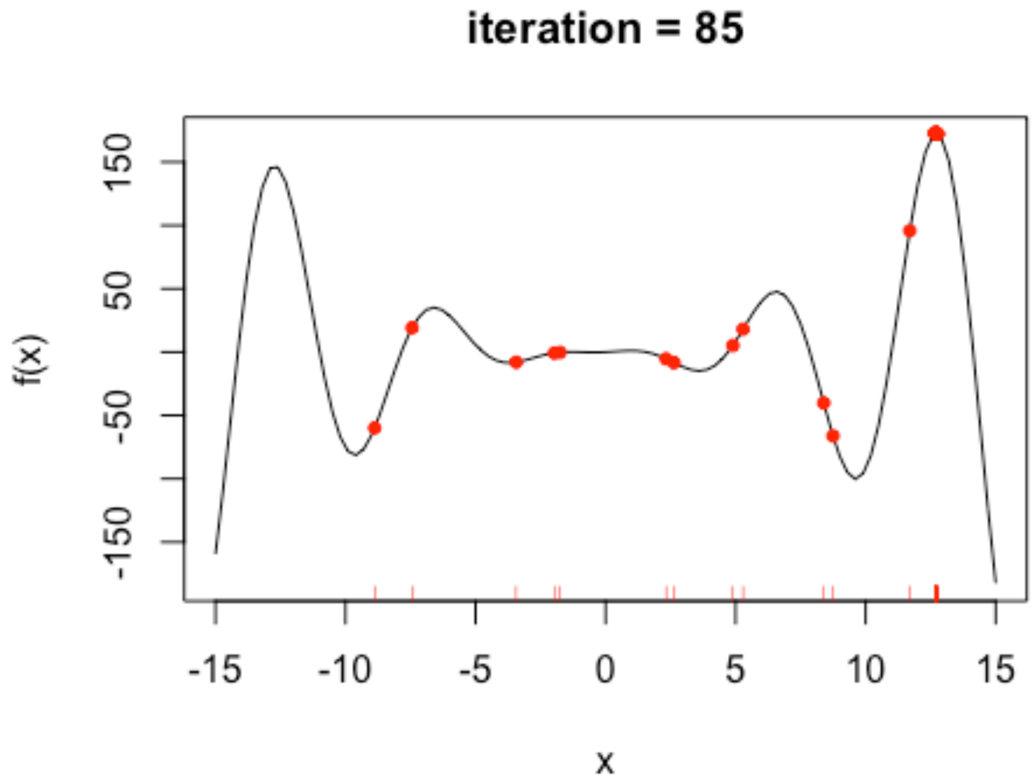

iteration = 81
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

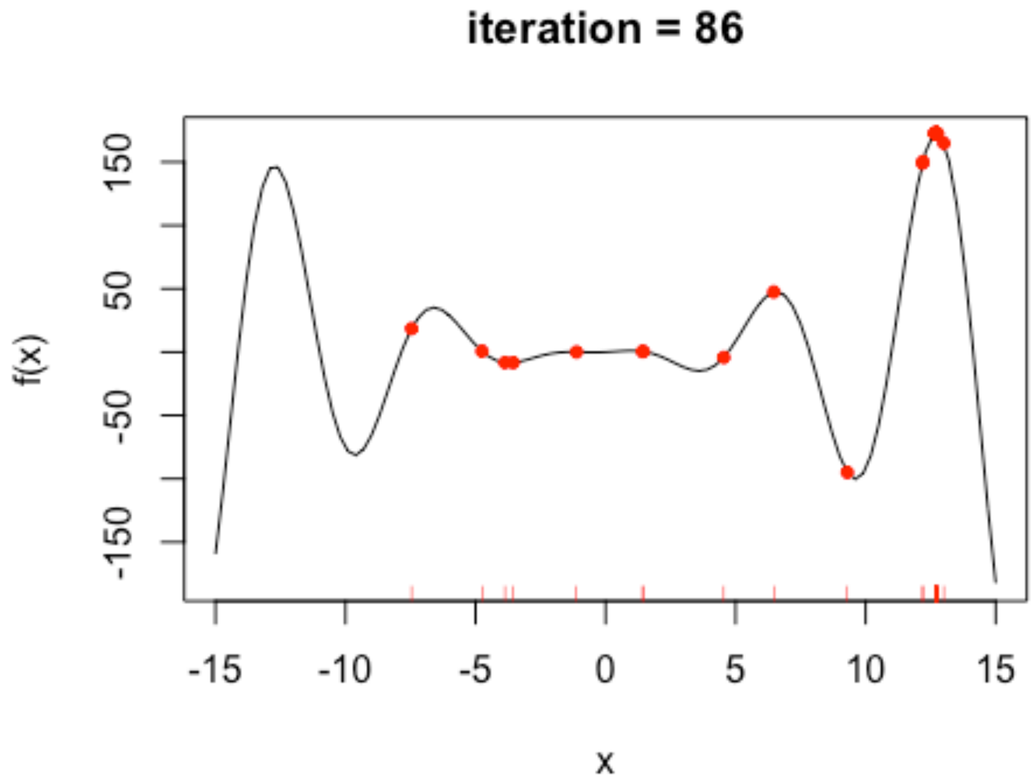


iteration = 82
150
50
-50
-150
f(x)
-15
-10
-5
0
5
10
15
x

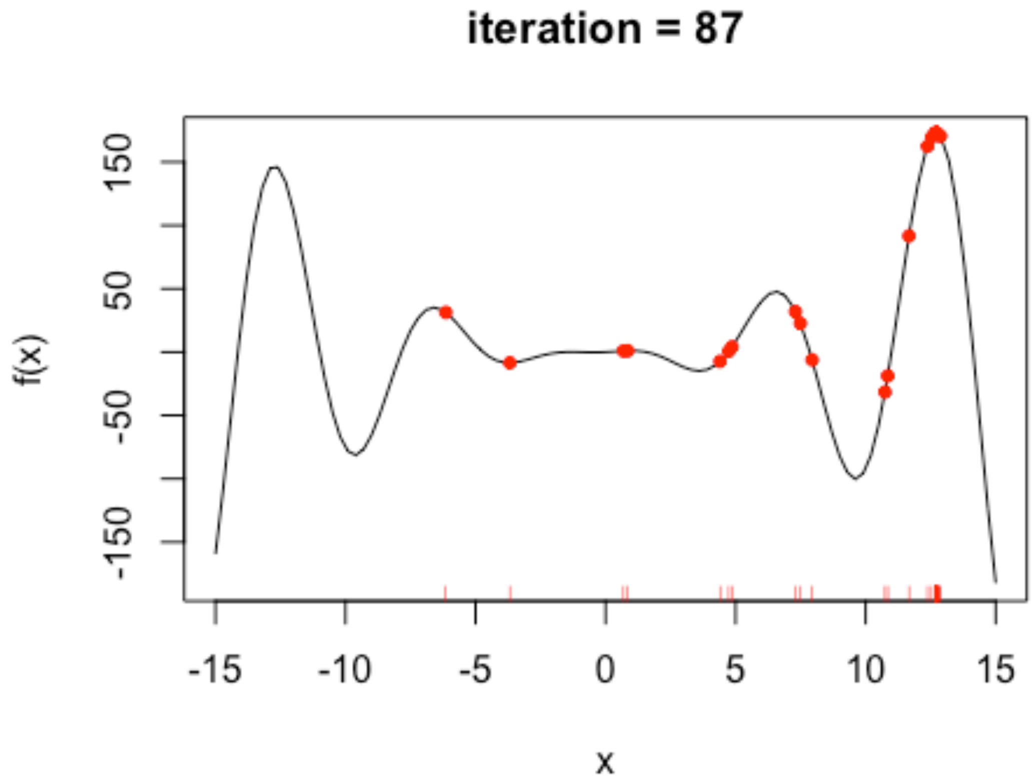

iteration = 83
150
50
-50
-150
f(x)
-15
-10
-5
0
5
10
15
x

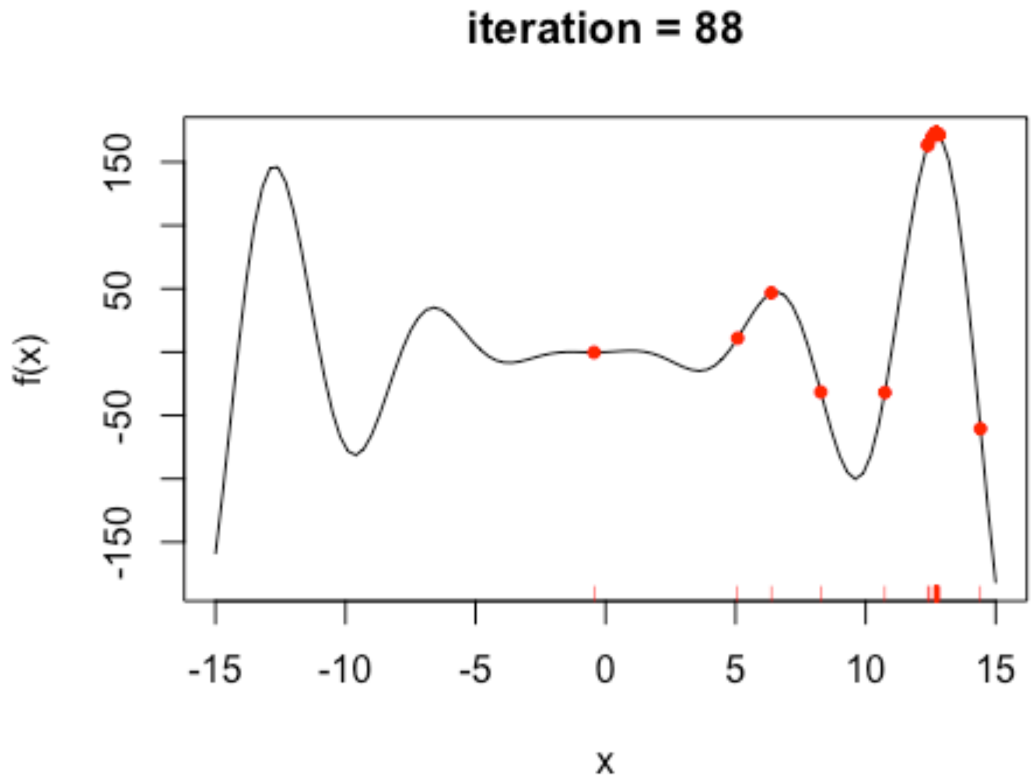

iteration = 84
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

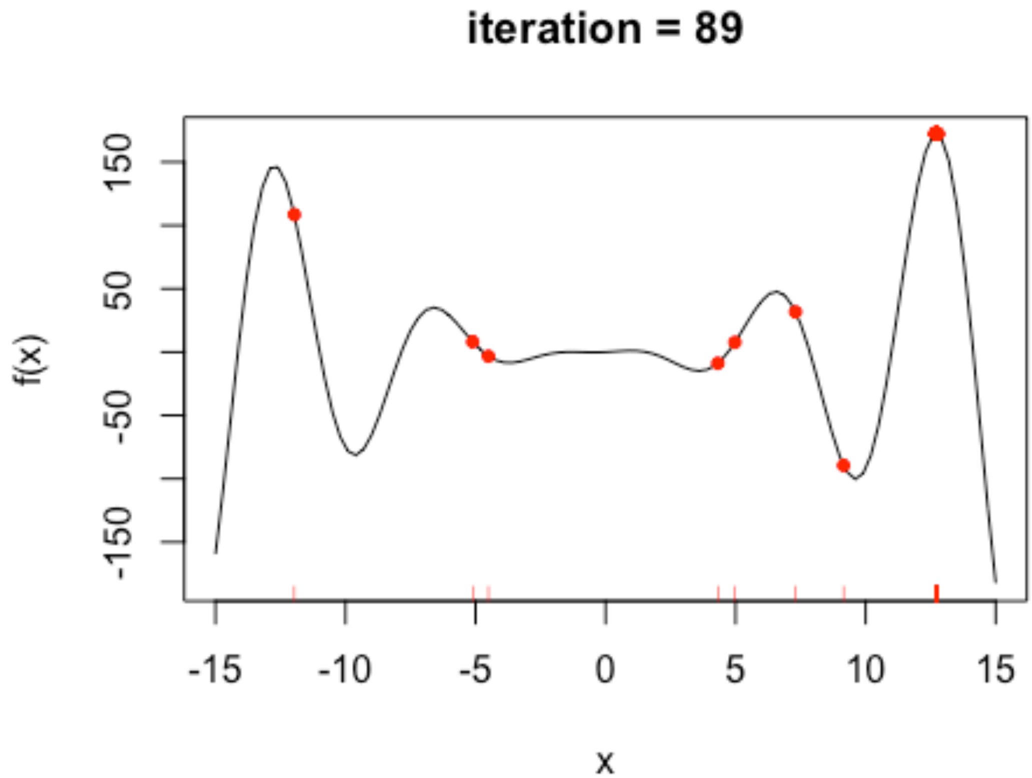

iteration = 85
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

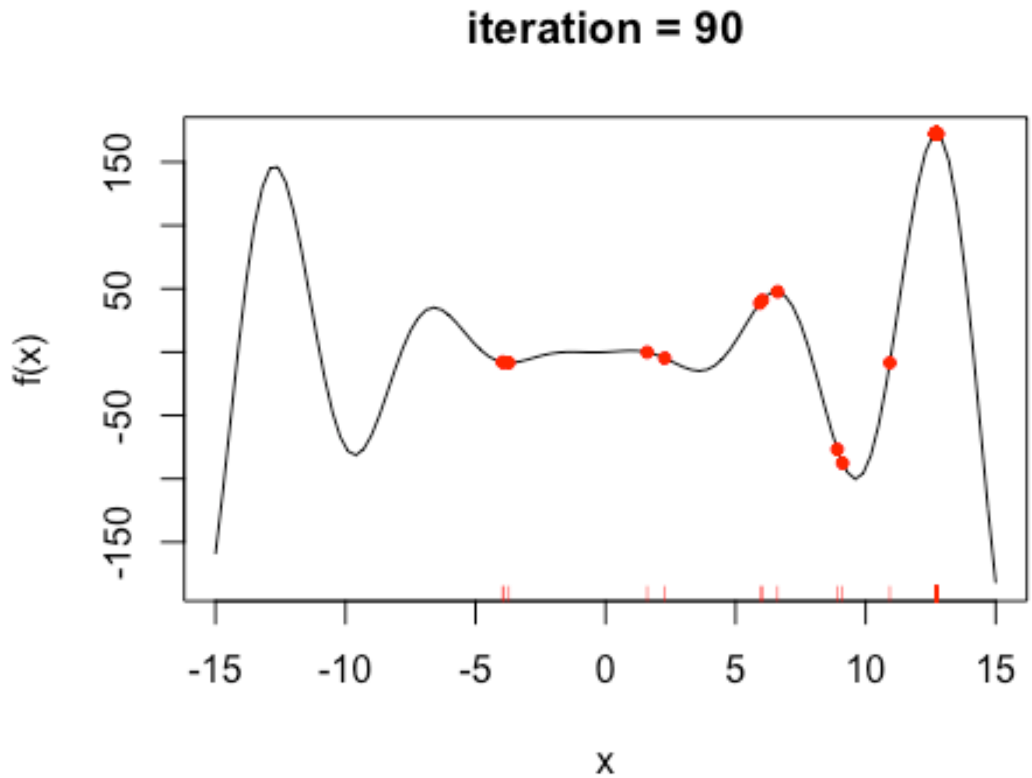


iteration = 86
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

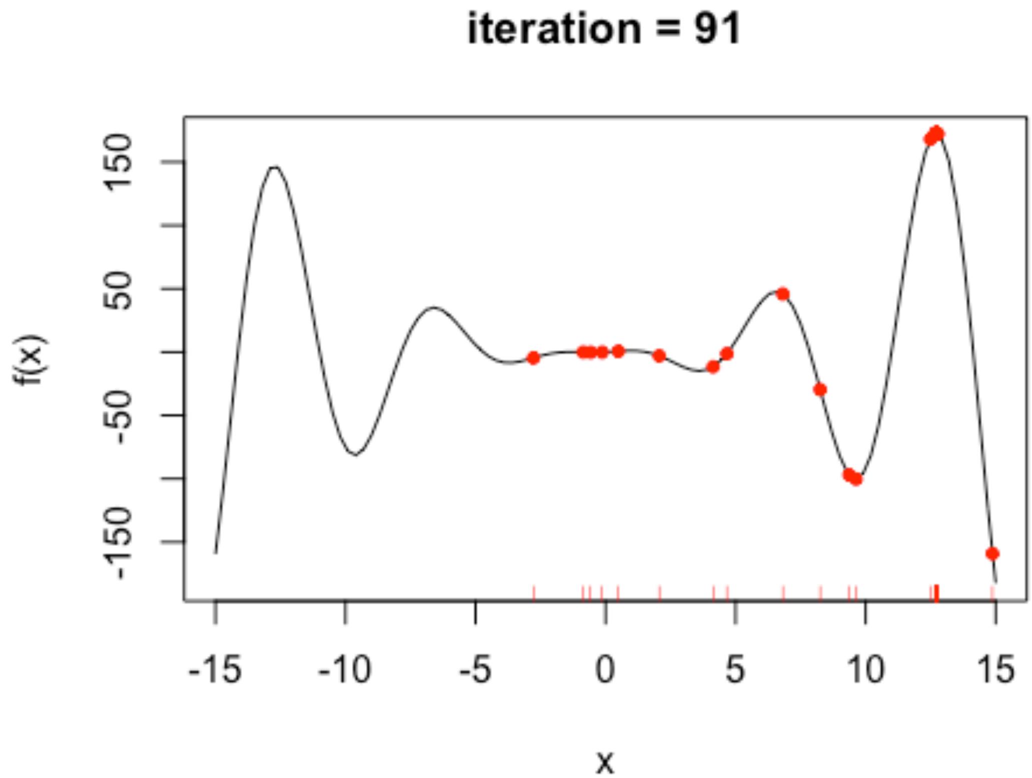

iteration = 87
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

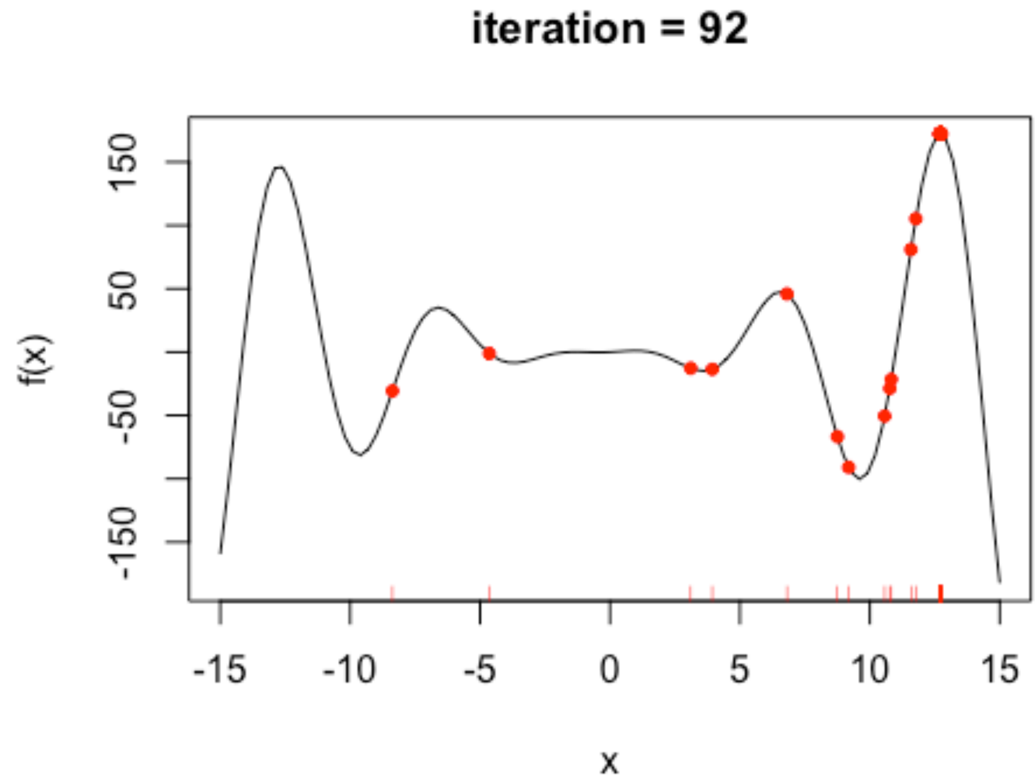

iteration = 88
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x


iteration = 89
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

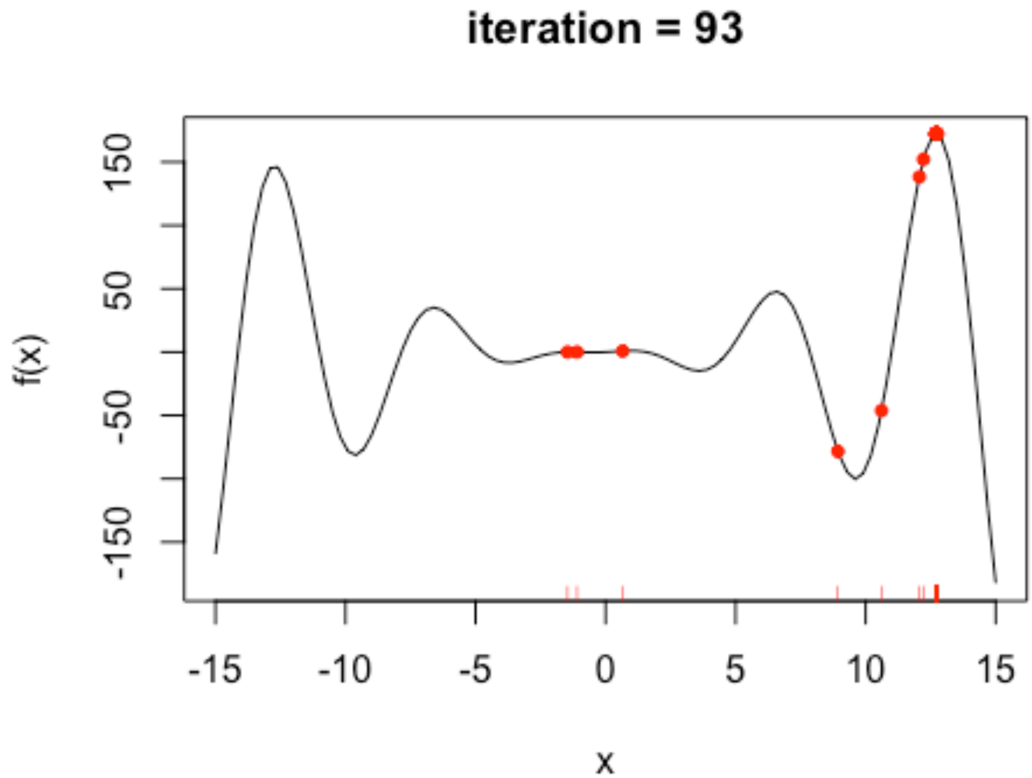


iteration = 90
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

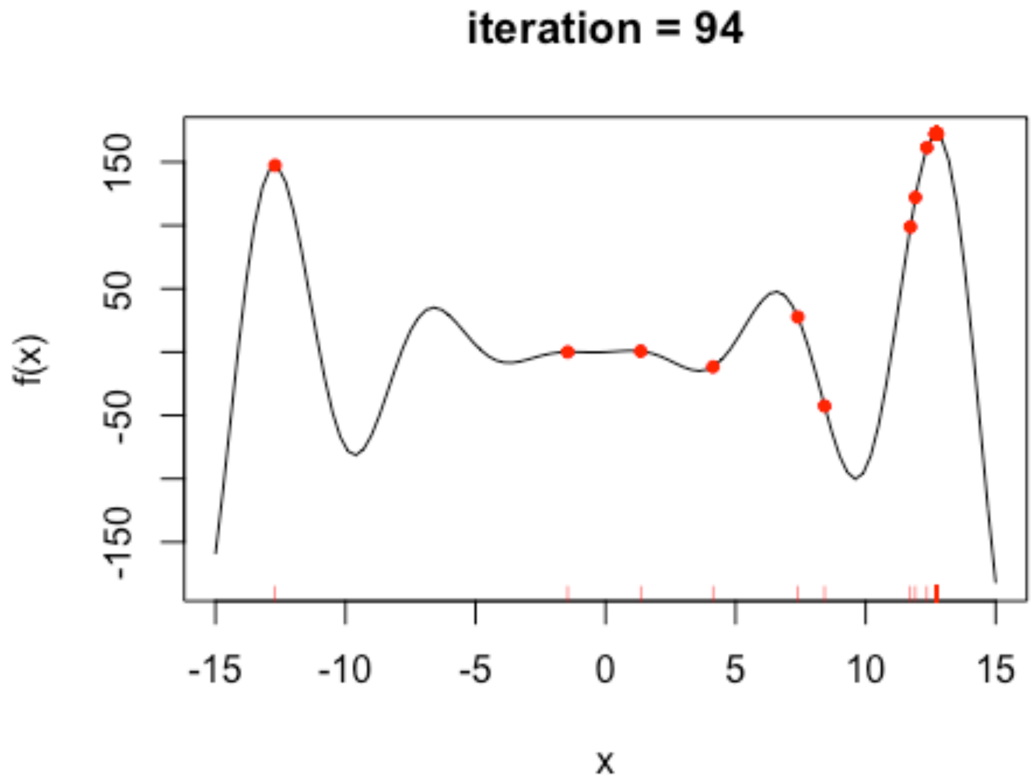

iteration = 91
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

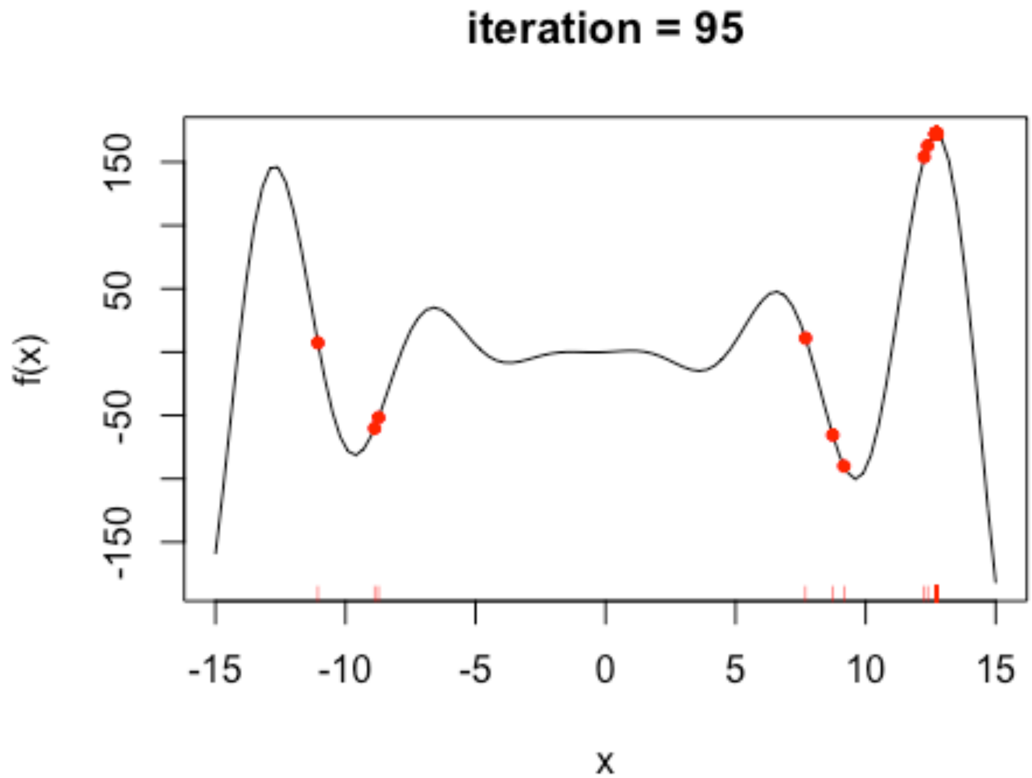


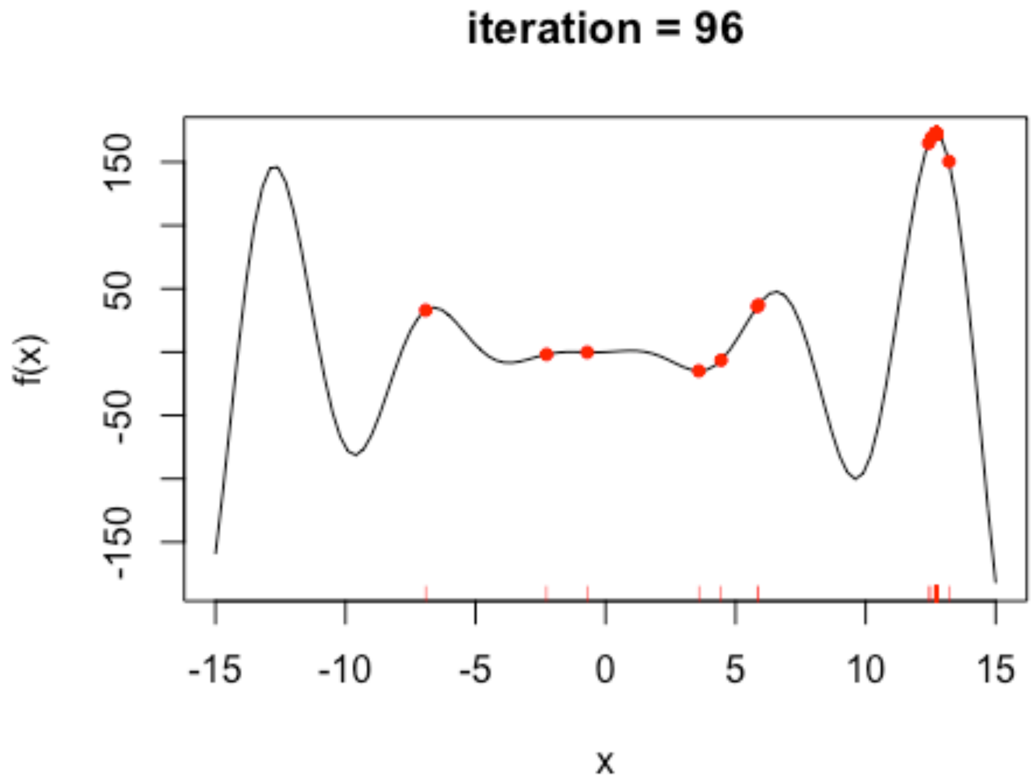
iteration = 92
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

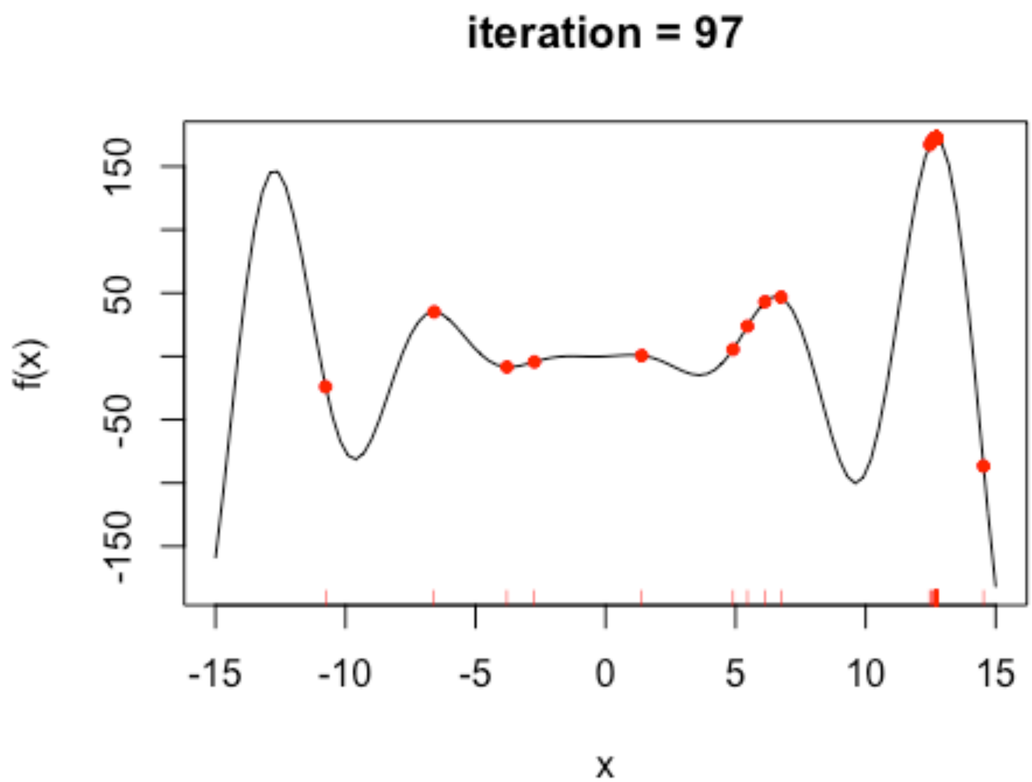


iteration = 93
150
50
-50
-150
f(x)
-15
-10
-5
0
5
10
15
x

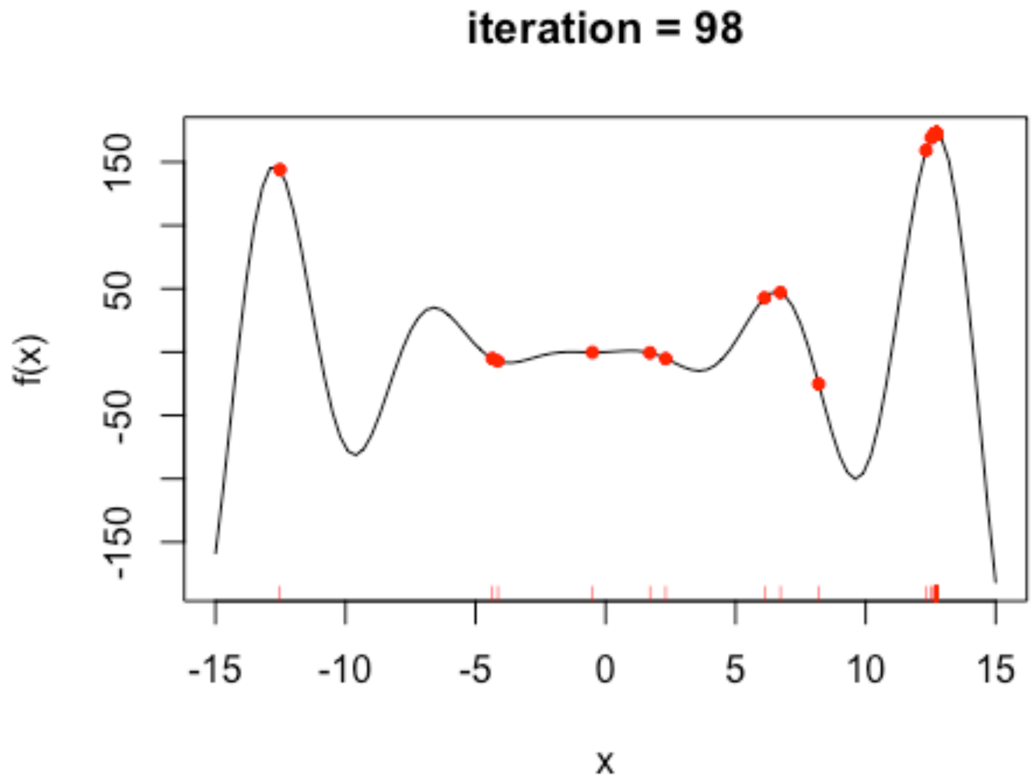

iteration = 94
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

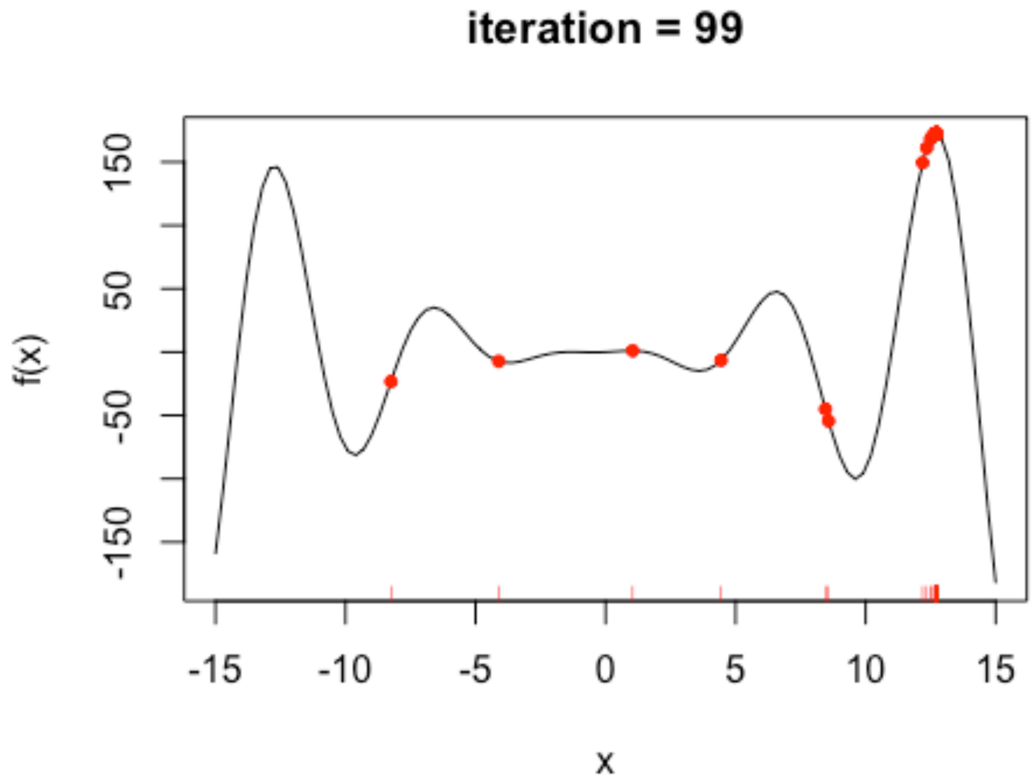

iteration = 95
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

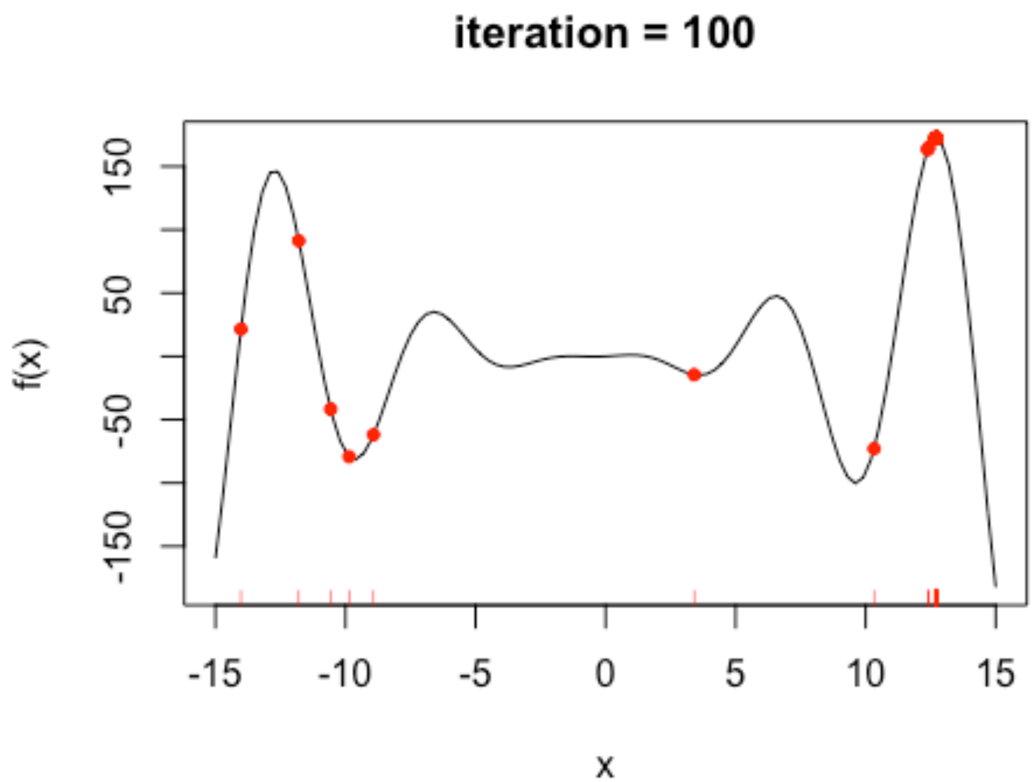

iteration = 96
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x



iteration = 97
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

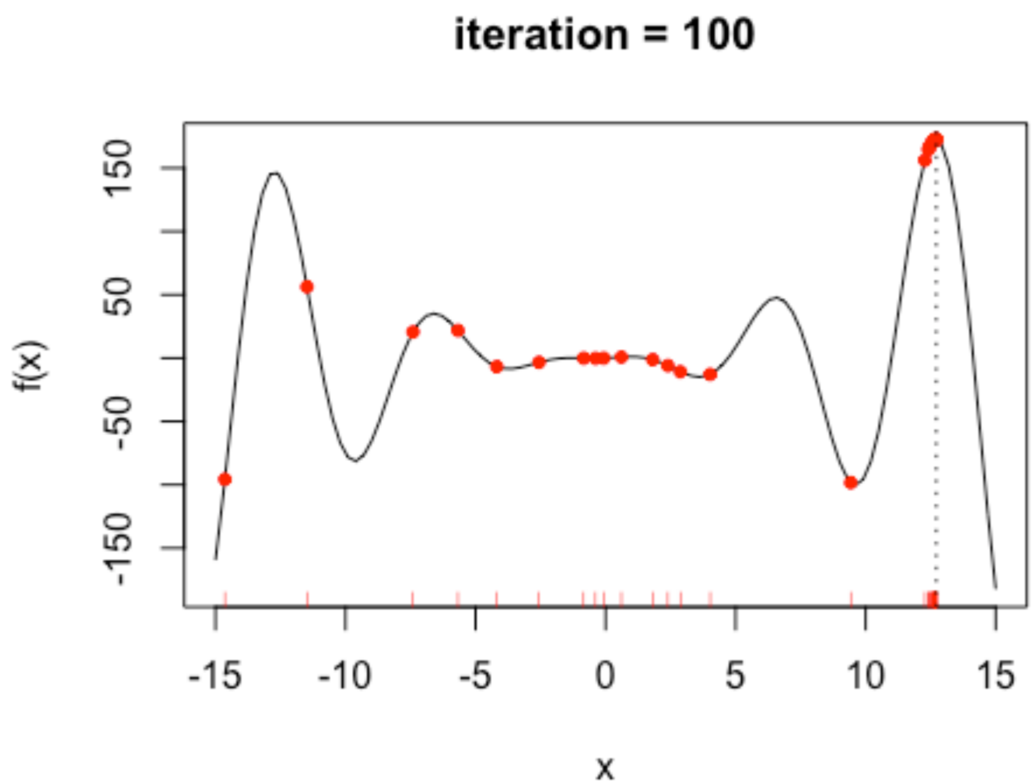

iteration = 98
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x


iteration = 99
f(x)
150
50
-50
-150
-15
-10
-5
0
5
10
15
x

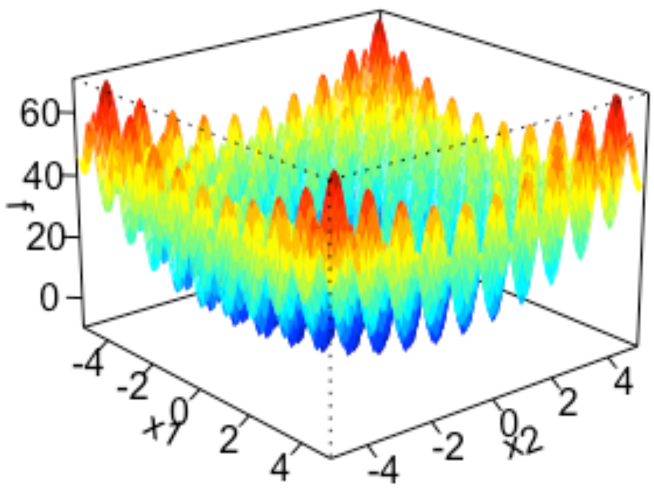



```
## End(Not run)
# or if you want to suppress the tracing
Gene_Alg <- ga(type = "real-valued", fitness = f, min = -15, max = 15,
monitor = NULL)
summary(Gene_Alg)

## +-----------------------------------+
## |         Genetic Algorithm         |
## +-----------------------------------+
##
## GA settings:
## Type                  =  real-valued
## Population size       =  50
## Number of generations =  100
## Elitism               =  2
## Crossover probability =  0.8
## Mutation probability  =  0.1
## Search domain =
##      x1
## Min -15
## Max  15
##
## GA results:
```

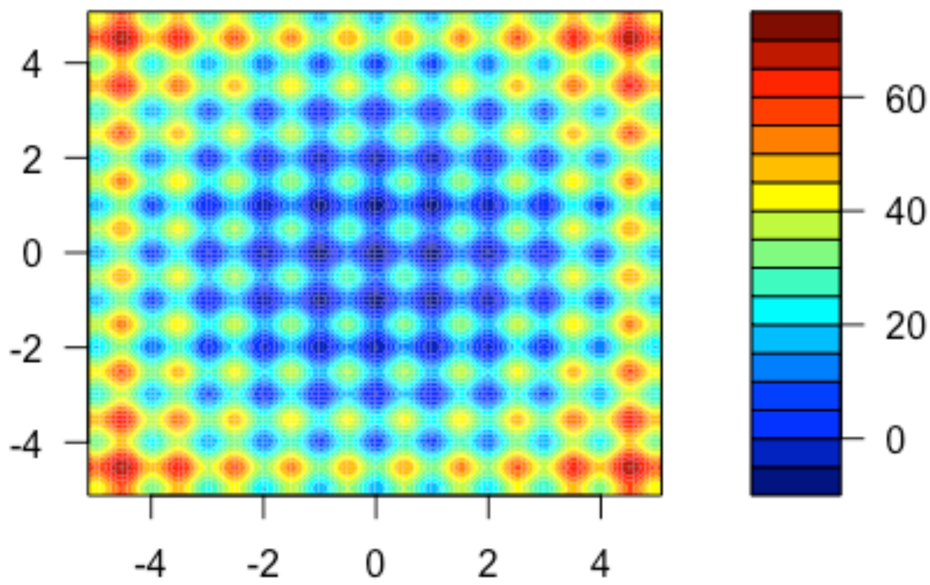
```
## Iterations             = 100
## Fitness function value = 172.4639
## Solution =
##            x1
## [1,] 12.71678

monitor(Gene_Alg)
abline(v = Gene_Alg@solution, lty = 3)
```




```
# -----------------          two-dimensional Rastrigin function   ------
-----------
Rastrigin_Fun <- function(x1, x2)
{
   10 +x1^2 + x2^2 - 10*(cos(2*pi*x1) + cos(2*pi*x2))
}
x1 <- x2 <- seq(-5.12, 5.12, by = 0.1)
f <- outer(x1, x2, Rastrigin_Fun)
persp3D(x1, x2, f, theta = 50, phi = 20)
```



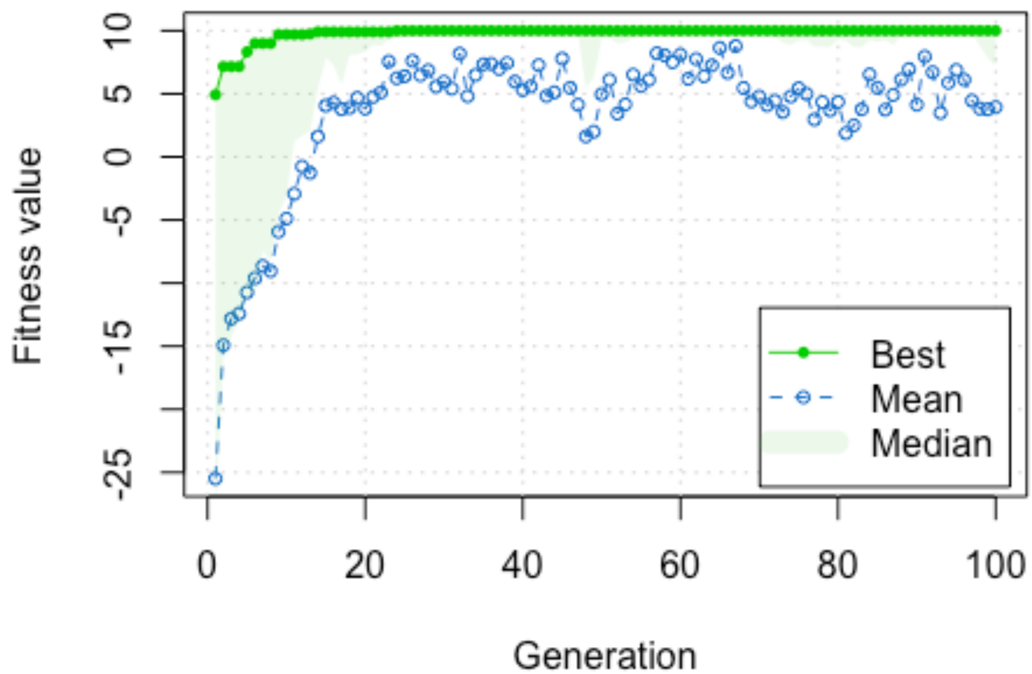

```r
filled.contour(x1, x2, f, color.palette = jet.colors)
```




```
Gene_Alg <- ga(type = "real-valued", fitness = function(x) -
Rastrigin_Fun(x[1], x[2]),
         min = c(-5.12, -5.12), max = c(5.12, 5.12),
         popSize = 50, maxiter = 100)

summary(Gene_Alg)

## +-----------------------------------+
## |          Genetic Algorithm        |
## +-----------------------------------+
##
## GA settings:
## Type                  =  real-valued
## Population size       =  50
## Number of generations =  100
## Elitism               =  2
## Crossover probability =  0.8
## Mutation probability  =  0.1
## Search domain =
##        x1    x2
## Min -5.12 -5.12
## Max  5.12  5.12
##
```

```
## GA results:
## Iterations             = 100
## Fitness function value = 9.999547
## Solution =
##                x1           x2
## [1,] 0.001381059 0.0006149782
```

```
plot(Gene_Alg)
```



Genetic Algorithms in R

We will use Genetic Algorithms to solve the knapsack problem. The version of the knapsack problem problem being solved is the 0-1 knapsack problem, which restricts the number x_i of copies of each kind of item to zero or one. Given a set of n items numbered from 1 up to n, each with a weight w_i and a value v_i, along with a maximum weight capacity W,

$$maximize \sum_{i=1}^{n} v_i\, x_i subjectto \sum_{i=1}^{n} w_i\, x_i \leq W and x_i \in \{0,1\}$$

Here x_i represents the number of instances of item i to include in the knapsack. Informally, the problem is to maximize the sum of the values of the items in the knapsack so that the sum of the weights is less than or equal to the knapsack's capacity.

```
knap <- data.frame(item = c( "oranges", "onions", "pocketknife",
"beans","sleeping bag", "rope", "compass"), value = c(15, 2, 10, 20,
30, 10, 30), weight = c( 10, 1,1, 5, 7, 5, 1))
head(knap)

##           item value weight
## 1      oranges    15     10
## 2       onions     2      1
## 3  pocketknife    10      1
## 4        beans    20      5
## 5 sleeping bag    30      7
## 6         rope    10      5

weight_limit <- 20
#Each number in this binary string represents whether or not to take an
item with you.
#A value of 1 refers to putting the specific item in the knapsack while
a 0 refers to leave the item at home.
chromosome = c(1, 0, 0, 1, 1, 0, 0)
knap[chromosome == 1, ]

##           item value weight
## 1      oranges    15     10
## 4        beans    20      5
## 5 sleeping bag    30      7

#check what amount of surivival points this configuration sums up
cat(chromosome %*% knap$value)

## 65

#Define evaluation function
evalFunc <- function(x) {
  current_solution_value <- x %*% knap$value
  current_solution_weight <- x %*% knap$weight

  if (current_solution_weight > weight_limit)
    return(0)
  else return(-current_solution_value)
}

# choose the number of iterations, design and run the model
iter = 100
my_GA_Model <- rbga.bin(size = 7,
                        popSize = 200,
                        iters = iter,
```

```r
                             mutationChance = 0.01,
                             elitism = T,
                             evalFunc = evalFunc)
cat(summary(my_GA_Model))
```

```
## GA Settings
##   Type                      = binary chromosome
##   Population size           = 200
##   Number of Generations     = 100
##   Elitism                   = TRUE
##   Mutation Chance           = 0.01
##
## Search Domain
##   Var 1 = [,]
##   Var 0 = [,]
##
## GA Results
##   Best Solution : 0 1 1 1 1 1 1
```

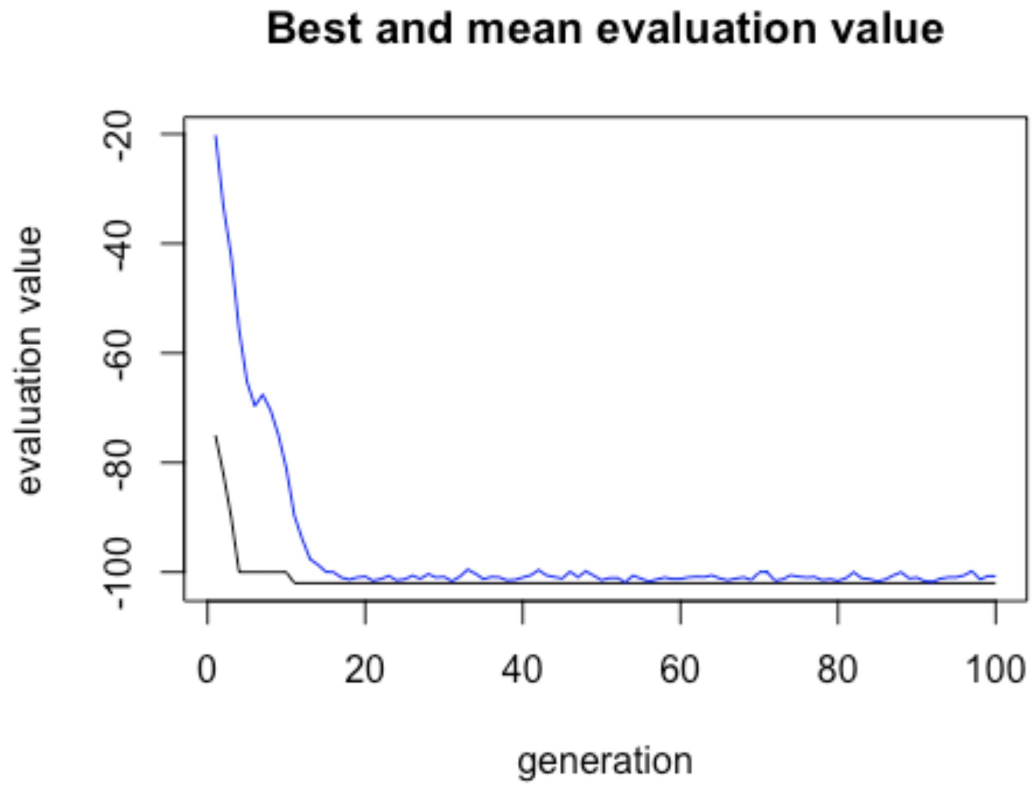
```r
solution = c(0, 1, 1, 1, 1, 1, 1)
knap[solution == 1, ]
```

```
##            item value weight
## 2        onions     2      1
## 3   pocketknife    10      1
## 4         beans    20      5
## 5  sleeping bag    30      7
## 6          rope    10      5
## 7       compass    30      1
```

```r
# solution vs available
cat(paste(solution %*% knap$value, "/", sum(knap$value)))
```

```
## 102 / 117
```

```r
plot(my_GA_Model)
```



Resources

- Genetic algorithms: a simple R example
- Genetic algorithms
- Using Genetic Algorithms in Quantitative Trading

```
```