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 in M07\_Lesson\_02.Rmd 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)

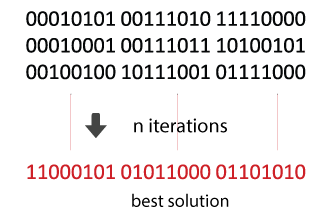
## Loading required package: GA  
## Loading required package: foreach  
## Loading required package: iterators  
## Package 'GA' version 2.2  
## Type 'citation("GA")' for citing this R package in publications.

# Data

We will be using the [UCI Machine Learning Repository: Wine Data Set](https://archive.ics.uci.edu/ml/datasets/Wine). These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

# 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

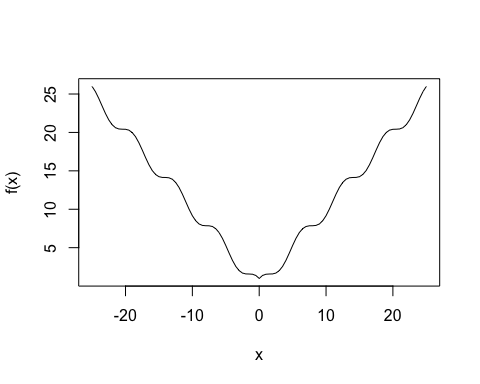
*Pros*  
\* Fast (and low memory)  
\* Finding a candidate representation and fitness function are the bulk of the work.

*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)



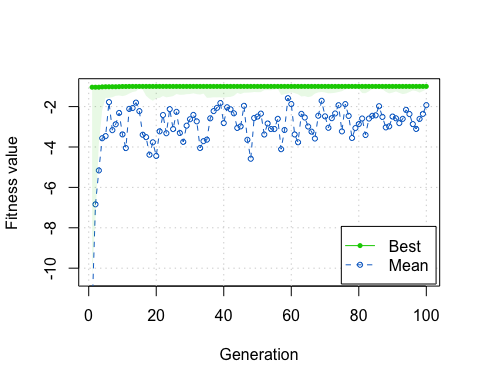
fit <- function(x) -f(x)  
Gene\_Alg <- ga(type = "real-valued", fitness = fit, min = -25, max = 25)

## Iter = 1 | Mean = -12.12321 | Best = -1.039825   
## Iter = 2 | Mean = -6.838973 | Best = -1.039825   
## Iter = 3 | Mean = -5.160673 | Best = -1.039825   
## Iter = 4 | Mean = -3.565659 | Best = -1.022745   
## Iter = 5 | Mean = -3.46081 | Best = -1.017069   
## Iter = 6 | Mean = -1.777199 | Best = -1.017069   
## Iter = 7 | Mean = -3.165609 | Best = -1.016085   
## Iter = 8 | Mean = -2.874249 | Best = -1.016085   
## Iter = 9 | Mean = -2.315605 | Best = -1.011572   
## Iter = 10 | Mean = -3.376115 | Best = -1.004297   
## Iter = 11 | Mean = -4.051436 | Best = -1.004297   
## Iter = 12 | Mean = -2.114815 | Best = -1.001512   
## Iter = 13 | Mean = -2.070309 | Best = -1.001002   
## Iter = 14 | Mean = -1.803986 | Best = -1.001002   
## Iter = 15 | Mean = -2.224279 | Best = -1.000718   
## Iter = 16 | Mean = -3.393505 | Best = -1.000718   
## Iter = 17 | Mean = -3.511629 | Best = -1.000484   
## Iter = 18 | Mean = -4.383342 | Best = -1.000484   
## Iter = 19 | Mean = -3.766733 | Best = -1.000484   
## Iter = 20 | Mean = -4.443003 | Best = -1.000484   
## Iter = 21 | Mean = -3.224346 | Best = -1.000484   
## Iter = 22 | Mean = -2.417031 | Best = -1.000484   
## Iter = 23 | Mean = -3.31864 | Best = -1.000484   
## Iter = 24 | Mean = -2.125768 | Best = -1.000484   
## Iter = 25 | Mean = -3.112061 | Best = -1.000484   
## Iter = 26 | Mean = -2.26029 | Best = -1.000484   
## Iter = 27 | Mean = -3.305455 | Best = -1.000484   
## Iter = 28 | Mean = -3.746163 | Best = -1.000484   
## Iter = 29 | Mean = -2.944758 | Best = -1.000484   
## Iter = 30 | Mean = -2.62486 | Best = -1.000484   
## Iter = 31 | Mean = -2.412045 | Best = -1.000484   
## Iter = 32 | Mean = -2.729808 | Best = -1.000484   
## Iter = 33 | Mean = -4.049549 | Best = -1.000484   
## Iter = 34 | Mean = -3.709327 | Best = -1.000484   
## Iter = 35 | Mean = -3.636332 | Best = -1.000484   
## Iter = 36 | Mean = -2.581342 | Best = -1.000484   
## Iter = 37 | Mean = -2.221494 | Best = -1.000484   
## Iter = 38 | Mean = -2.062017 | Best = -1.000484   
## Iter = 39 | Mean = -1.824269 | Best = -1.000484   
## Iter = 40 | Mean = -2.813758 | Best = -1.000484   
## Iter = 41 | Mean = -2.032412 | Best = -1.000484   
## Iter = 42 | Mean = -2.132742 | Best = -1.000484   
## Iter = 43 | Mean = -2.33188 | Best = -1.000484   
## Iter = 44 | Mean = -3.055355 | Best = -1.000484   
## Iter = 45 | Mean = -2.979158 | Best = -1.000484   
## Iter = 46 | Mean = -1.962257 | Best = -1.000484   
## Iter = 47 | Mean = -3.643373 | Best = -1.000484   
## Iter = 48 | Mean = -4.581349 | Best = -1.000484   
## Iter = 49 | Mean = -2.56516 | Best = -1.000484   
## Iter = 50 | Mean = -2.494174 | Best = -1.000484   
## Iter = 51 | Mean = -2.34725 | Best = -1.000484   
## Iter = 52 | Mean = -3.381756 | Best = -1.000484   
## Iter = 53 | Mean = -2.836881 | Best = -1.000484   
## Iter = 54 | Mean = -3.110748 | Best = -1.000484   
## Iter = 55 | Mean = -3.111402 | Best = -1.000484   
## Iter = 56 | Mean = -2.606618 | Best = -1.000484   
## Iter = 57 | Mean = -4.108581 | Best = -1.000484   
## Iter = 58 | Mean = -3.158498 | Best = -1.000484   
## Iter = 59 | Mean = -1.582485 | Best = -1.000484   
## Iter = 60 | Mean = -1.873915 | Best = -1.000484   
## Iter = 61 | Mean = -3.378522 | Best = -1.000484   
## Iter = 62 | Mean = -3.766059 | Best = -1.000484   
## Iter = 63 | Mean = -2.359693 | Best = -1.000484   
## Iter = 64 | Mean = -2.540086 | Best = -1.000484   
## Iter = 65 | Mean = -2.986461 | Best = -1.000484   
## Iter = 66 | Mean = -3.24108 | Best = -1.000484   
## Iter = 67 | Mean = -3.580883 | Best = -1.000484   
## Iter = 68 | Mean = -2.452672 | Best = -1.000484   
## Iter = 69 | Mean = -1.712896 | Best = -1.000484   
## Iter = 70 | Mean = -2.482325 | Best = -1.000484   
## Iter = 71 | Mean = -3.049455 | Best = -1.000484   
## Iter = 72 | Mean = -2.571847 | Best = -1.000484   
## Iter = 73 | Mean = -2.341205 | Best = -1.000484   
## Iter = 74 | Mean = -1.930581 | Best = -1.000484   
## Iter = 75 | Mean = -3.226502 | Best = -1.000484   
## Iter = 76 | Mean = -1.877294 | Best = -1.000093   
## Iter = 77 | Mean = -2.460513 | Best = -1.00004   
## Iter = 78 | Mean = -3.553152 | Best = -1.000027   
## Iter = 79 | Mean = -3.063268 | Best = -1.000027   
## Iter = 80 | Mean = -2.865488 | Best = -1.000027   
## Iter = 81 | Mean = -2.590597 | Best = -1.000027   
## Iter = 82 | Mean = -3.405599 | Best = -1.000027   
## Iter = 83 | Mean = -2.593021 | Best = -1.000027   
## Iter = 84 | Mean = -2.457114 | Best = -1.000027   
## Iter = 85 | Mean = -2.431432 | Best = -1.000027   
## Iter = 86 | Mean = -1.980054 | Best = -1.000027   
## Iter = 87 | Mean = -2.511141 | Best = -1.000027   
## Iter = 88 | Mean = -3.028609 | Best = -1.000027   
## Iter = 89 | Mean = -2.978239 | Best = -1.000027   
## Iter = 90 | Mean = -2.493498 | Best = -1.000027   
## Iter = 91 | Mean = -2.57805 | Best = -1.000027   
## Iter = 92 | Mean = -2.81799 | Best = -1.000027   
## Iter = 93 | Mean = -2.606608 | Best = -1.000027   
## Iter = 94 | Mean = -2.160958 | Best = -1.000027   
## Iter = 95 | Mean = -2.370005 | Best = -1.000027   
## Iter = 96 | Mean = -2.865957 | Best = -1.000027   
## Iter = 97 | Mean = -3.105834 | Best = -1.000027   
## Iter = 98 | Mean = -2.609608 | Best = -1.000027   
## Iter = 99 | Mean = -2.360192 | Best = -1.000027   
## Iter = 100 | Mean = -1.927947 | Best = -1.000027

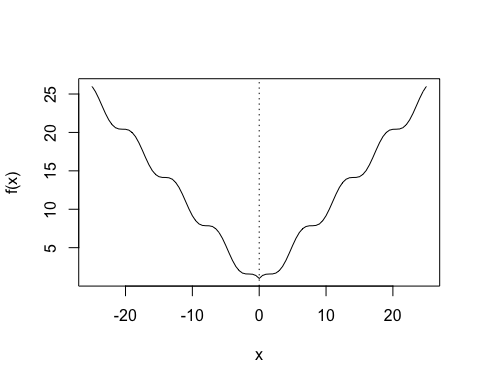
summary(Gene\_Alg)

## +-----------------------------------+  
## | 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 = -1.000027   
## Solution =   
## x1  
## [1,] -2.743304e-05

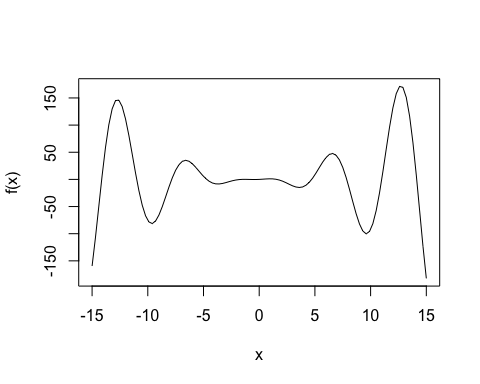
plot(Gene\_Alg)



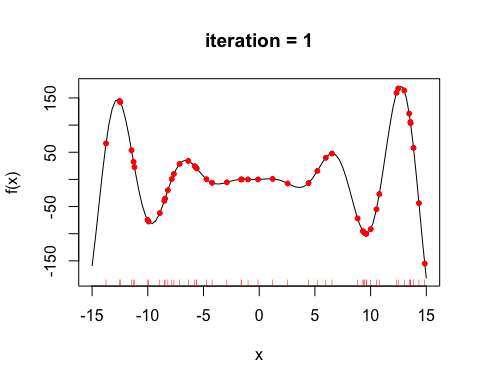
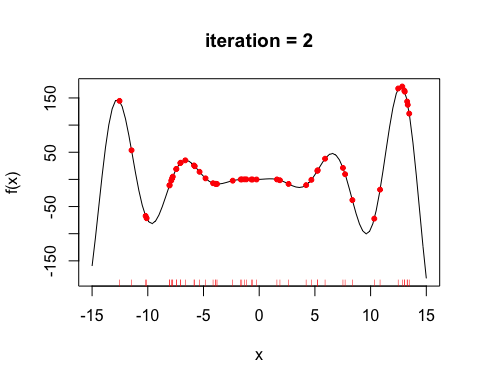
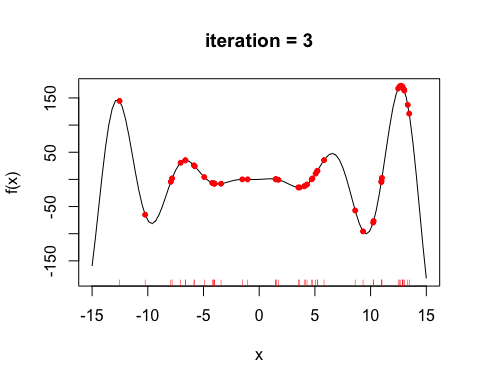
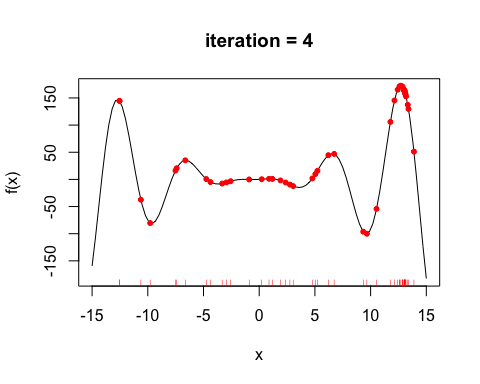
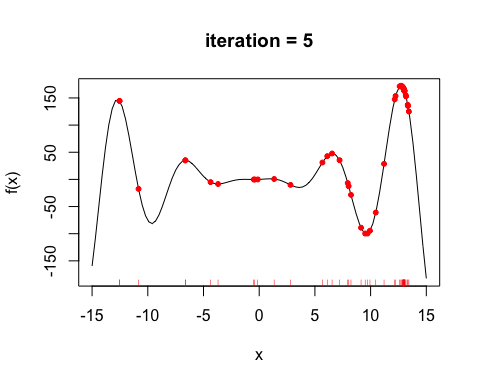
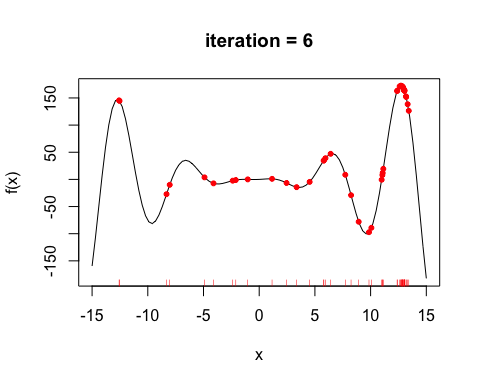
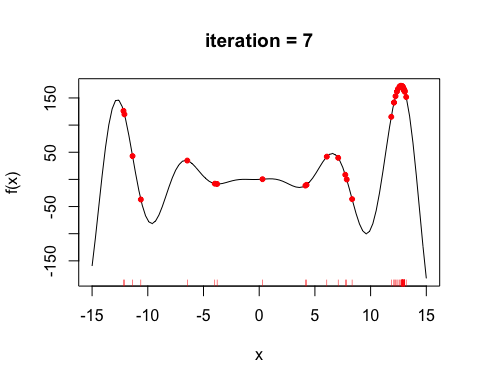
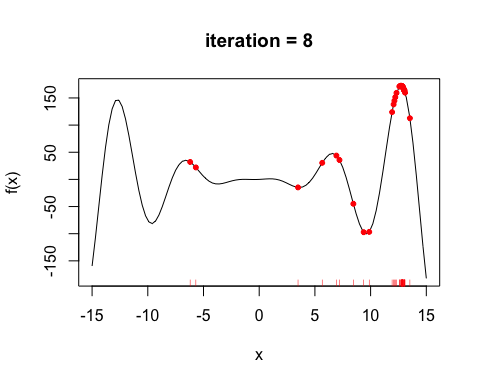
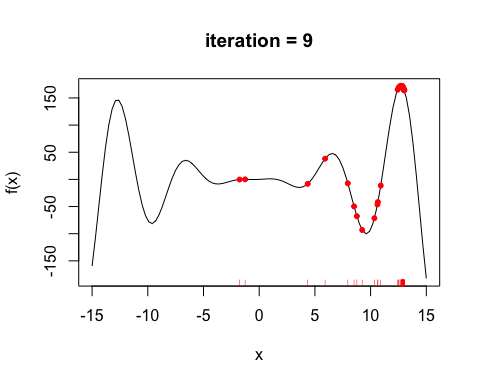
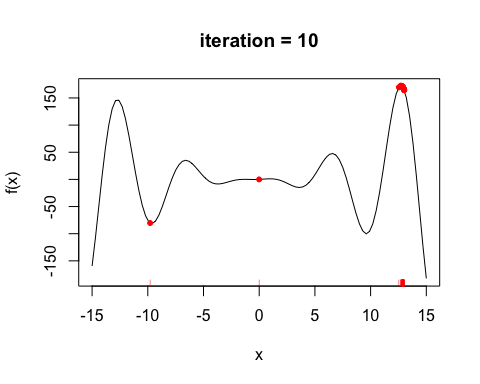
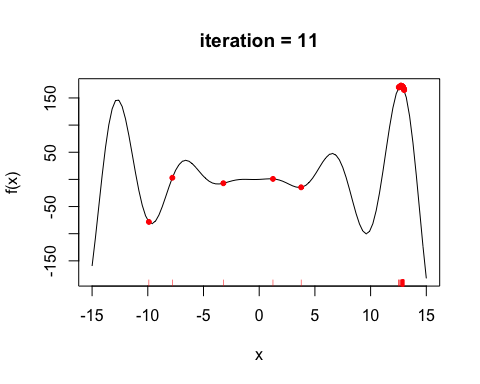
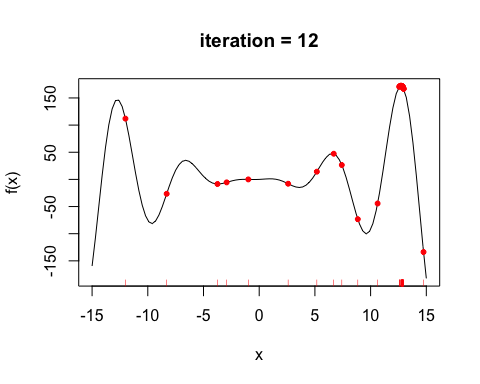
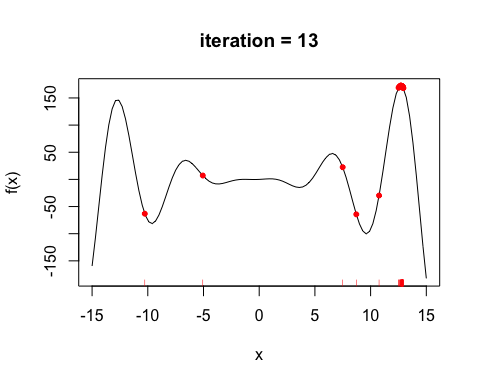
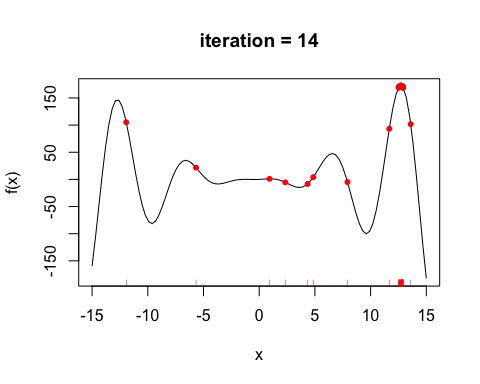
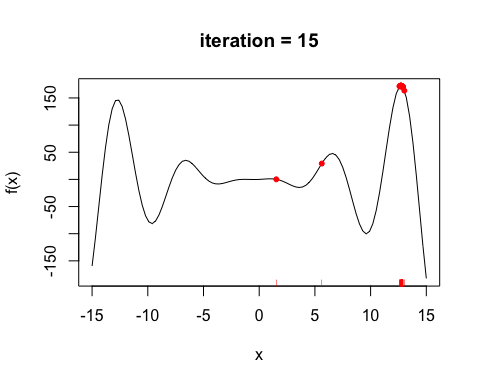
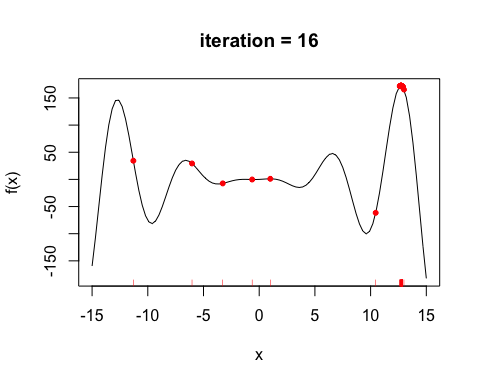
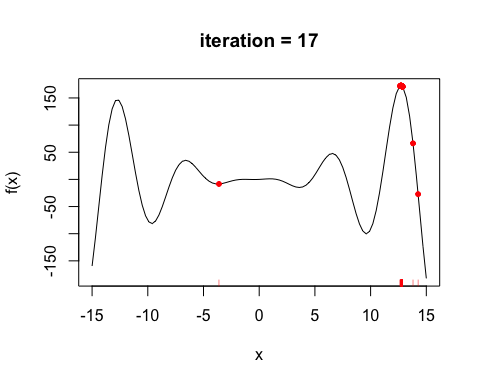
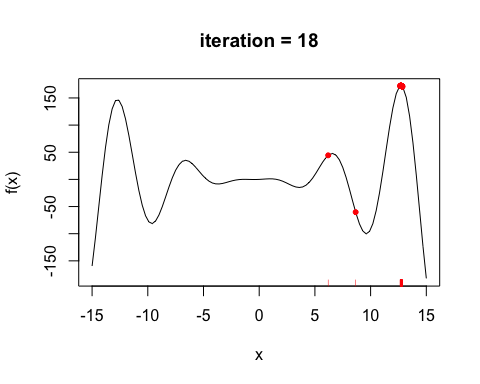
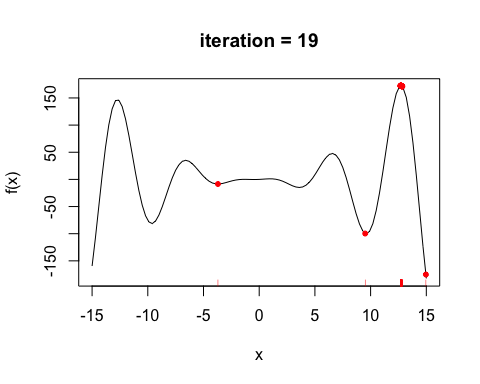
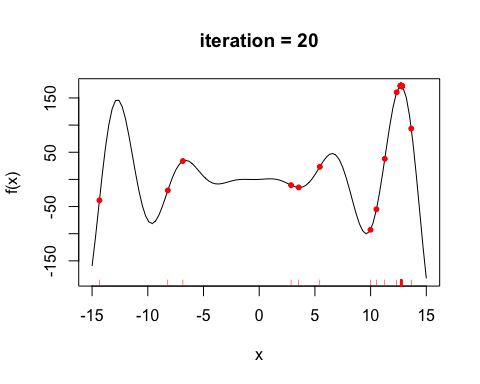
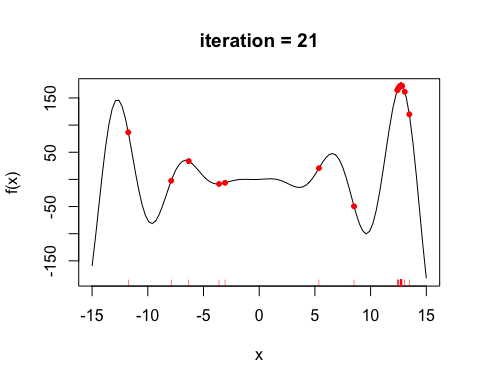
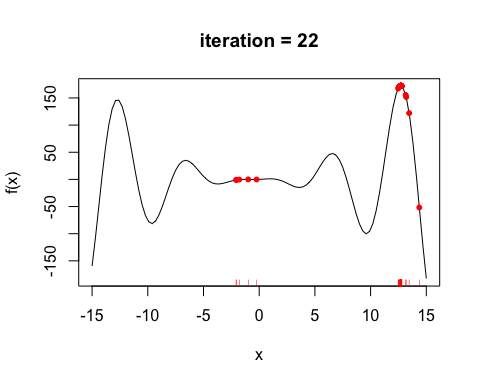
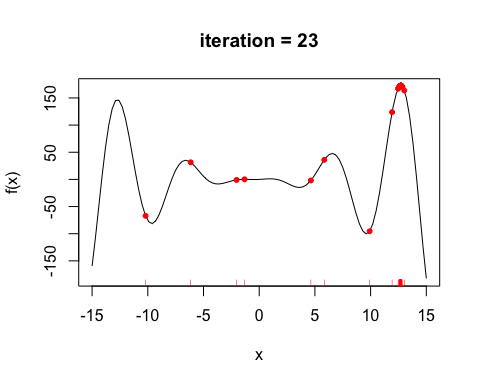
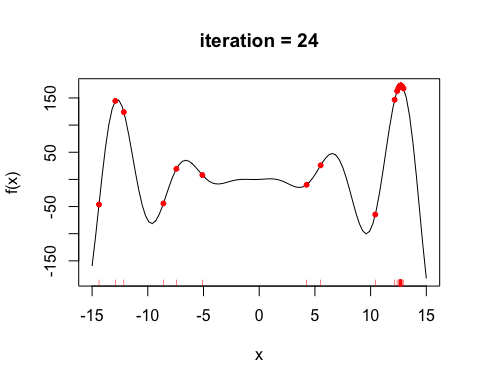
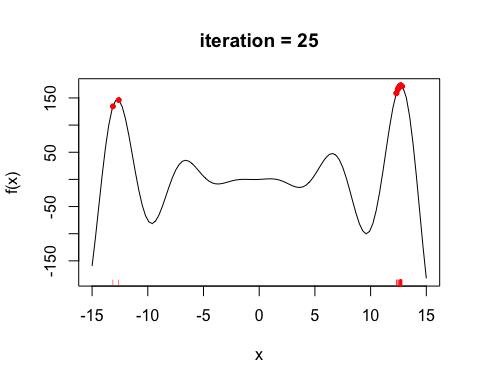
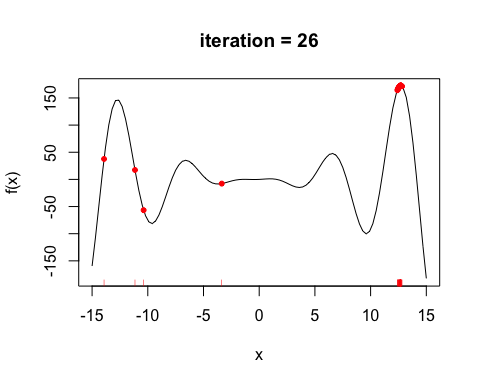
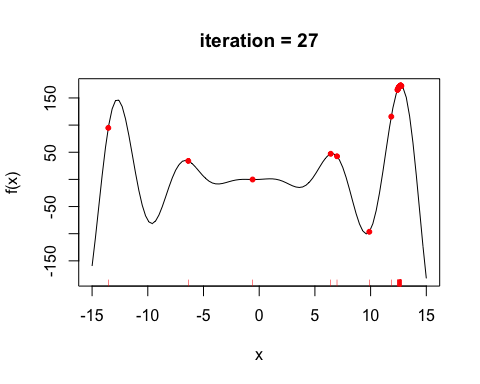
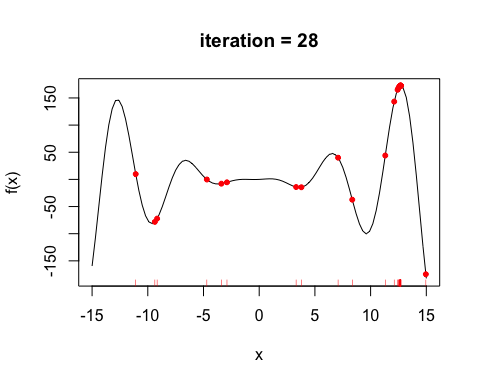
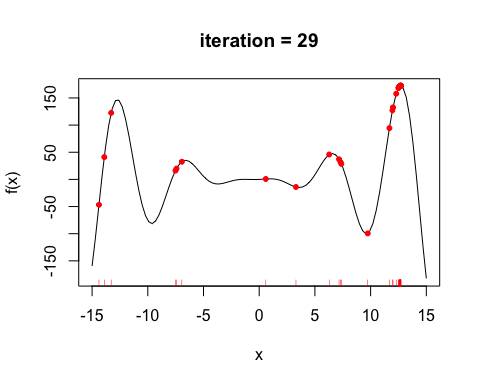
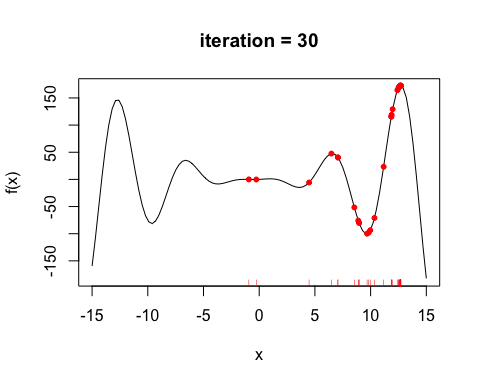
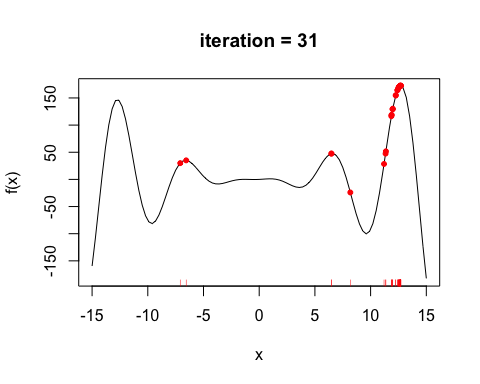
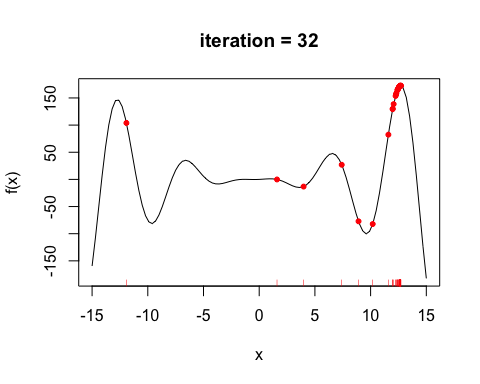
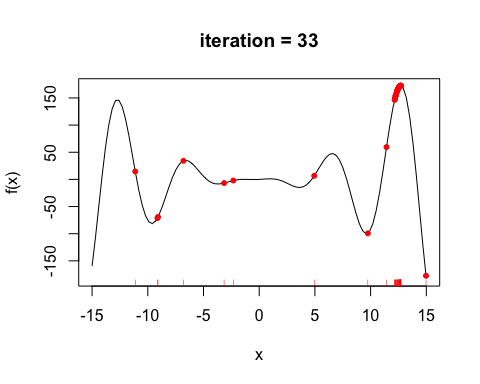
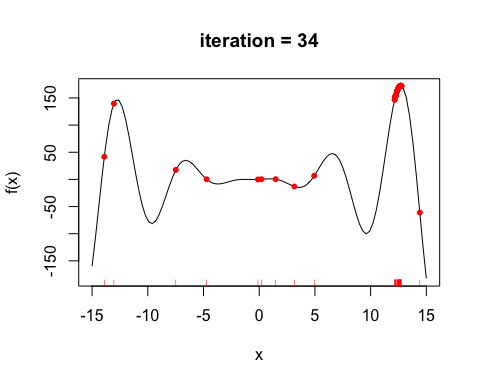
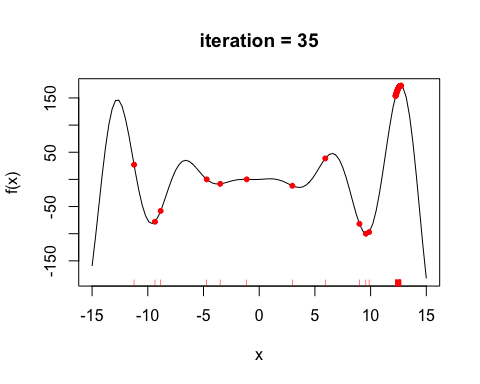
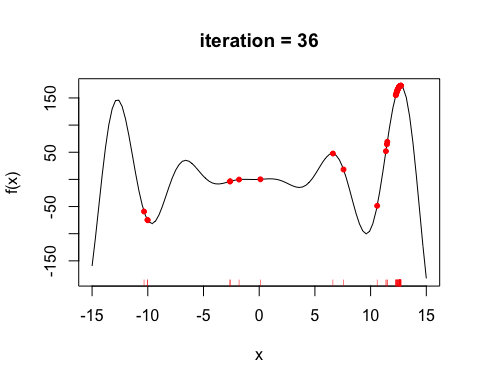
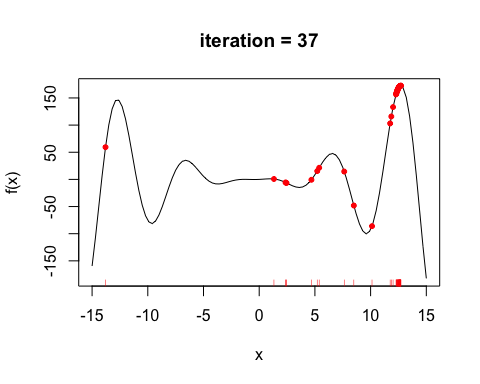
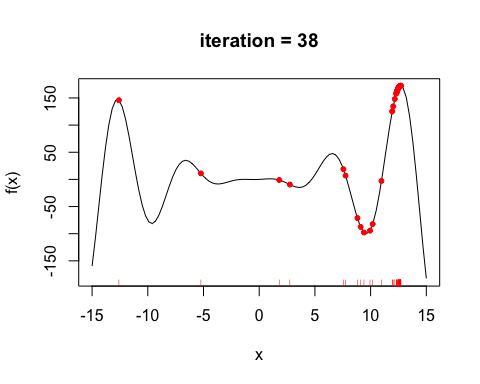
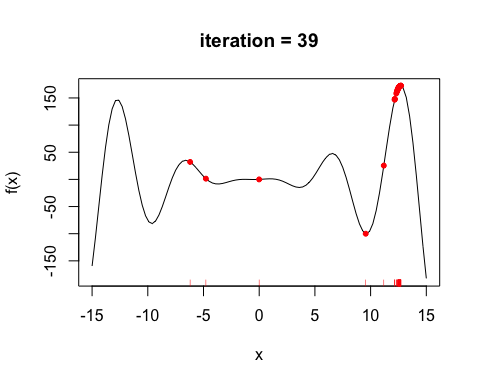
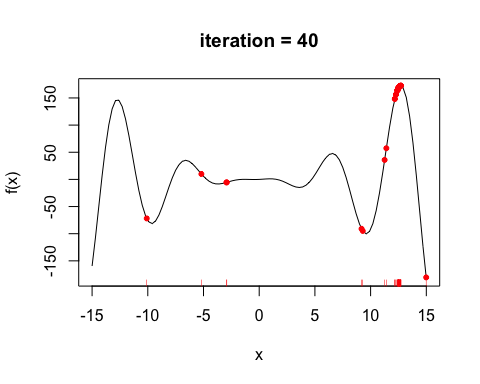
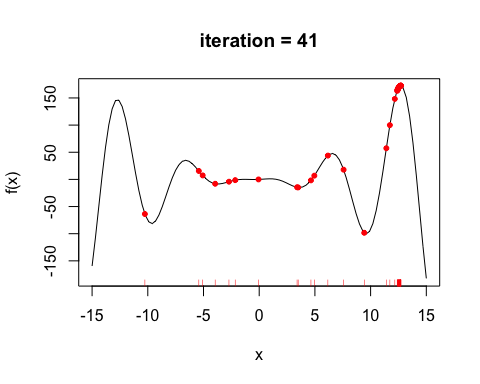
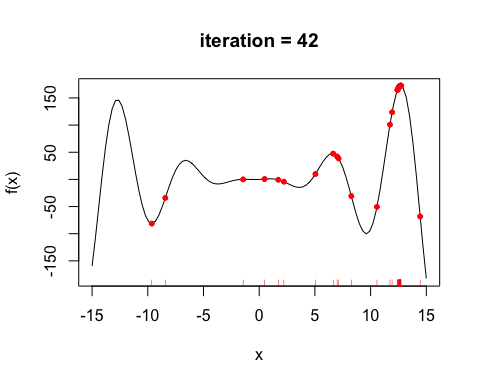
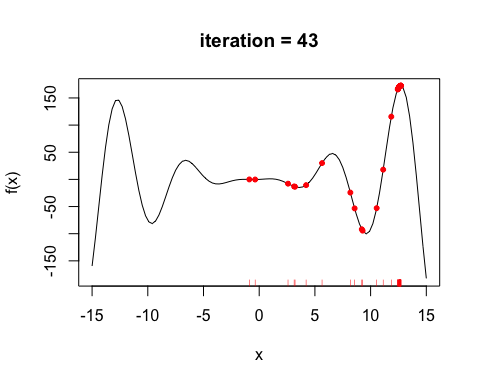
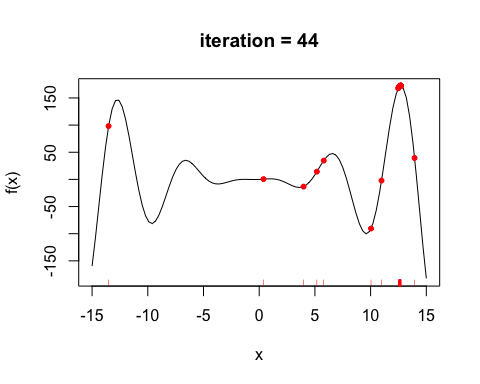
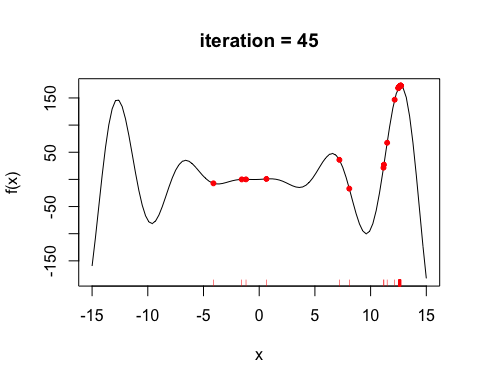
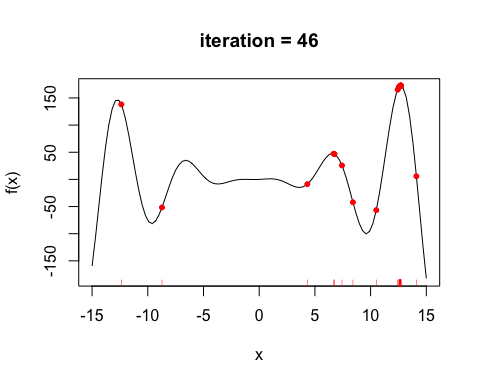
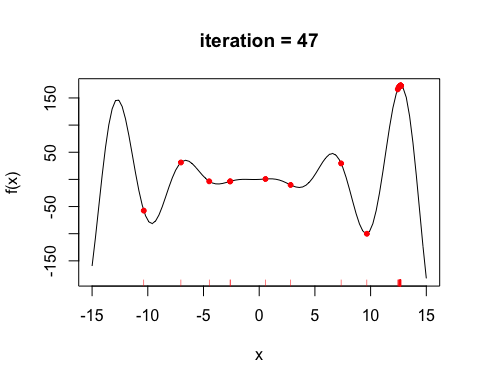
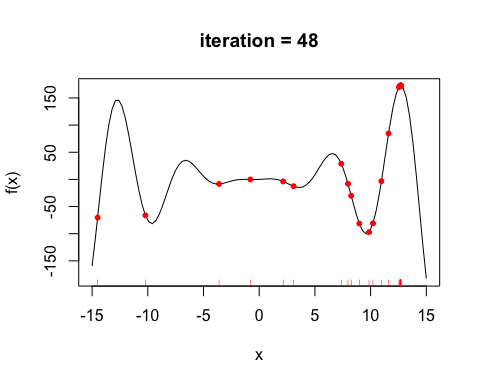
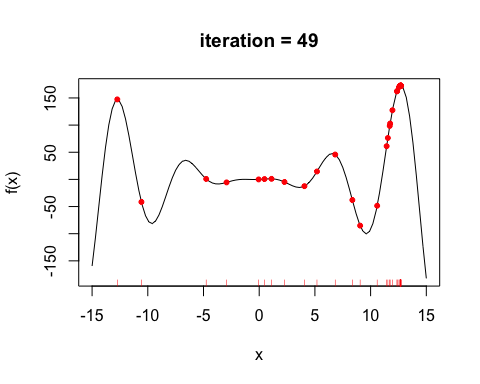
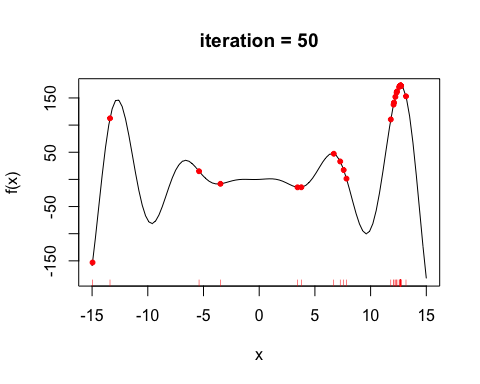
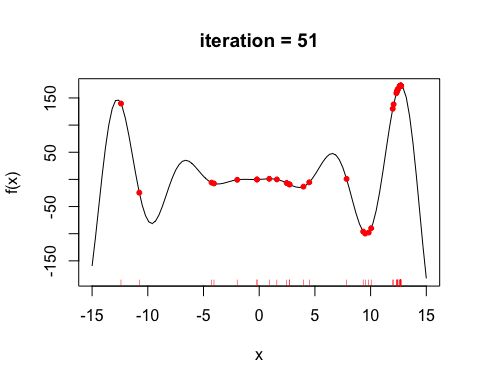
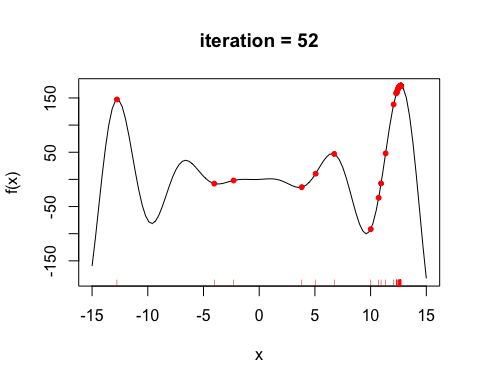
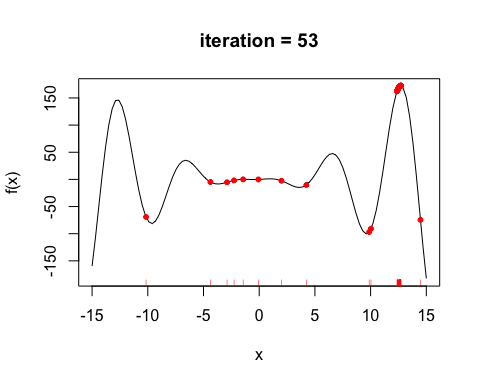
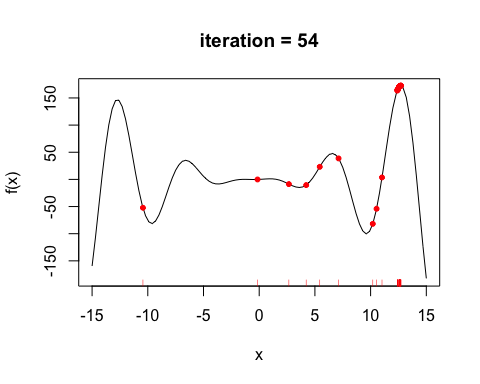
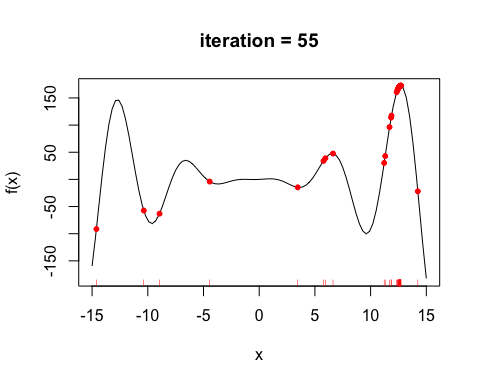
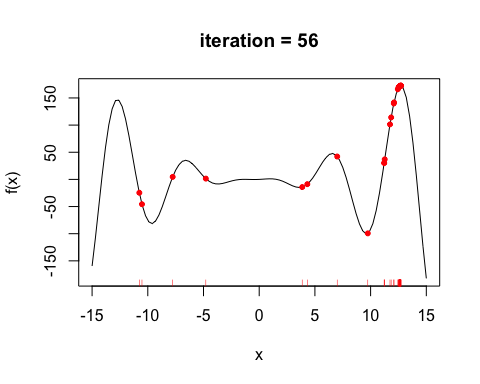
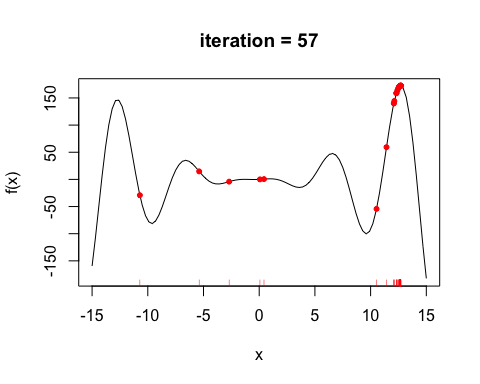
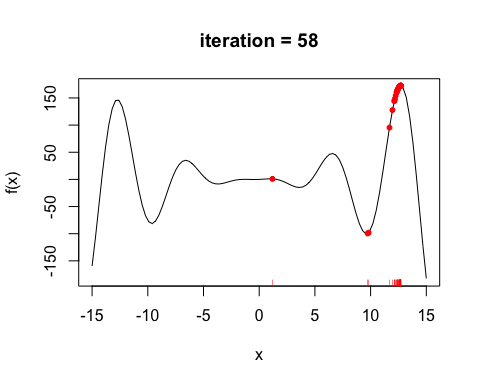
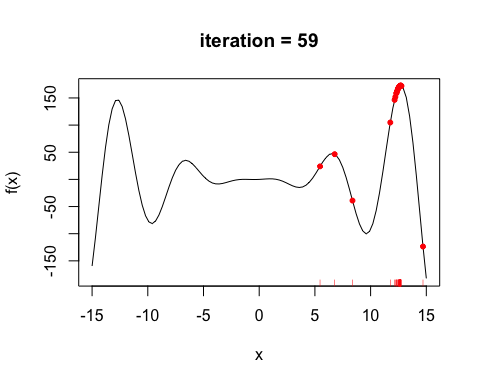
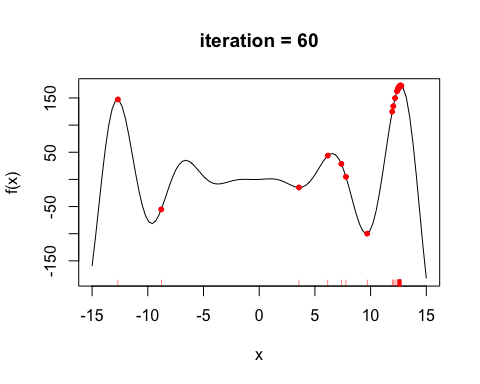
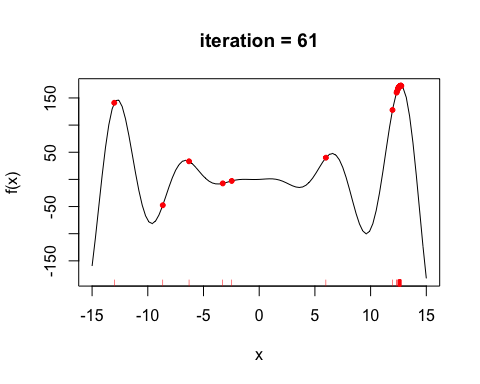
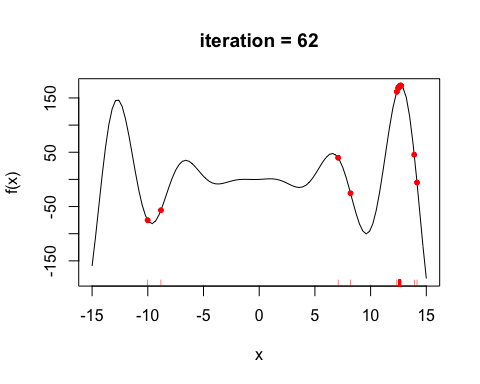
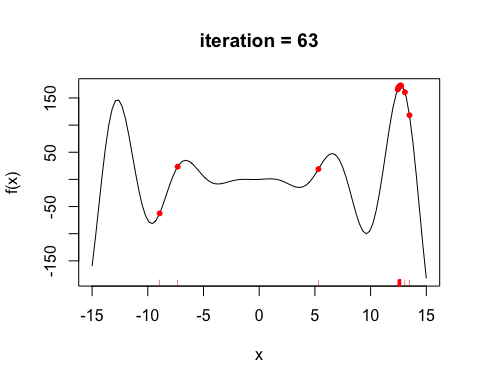
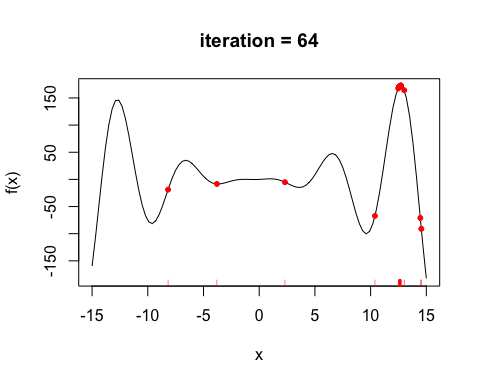
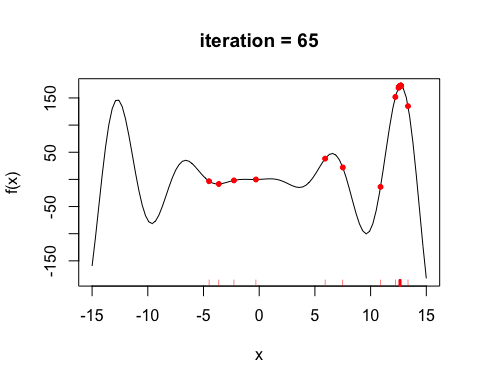
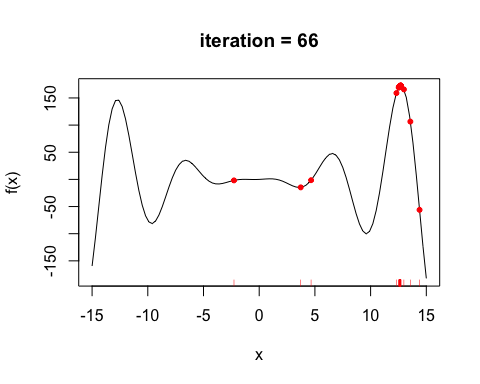
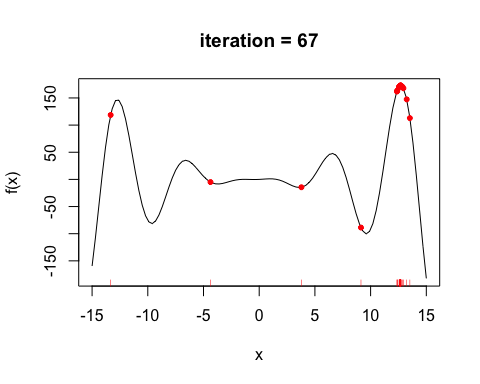
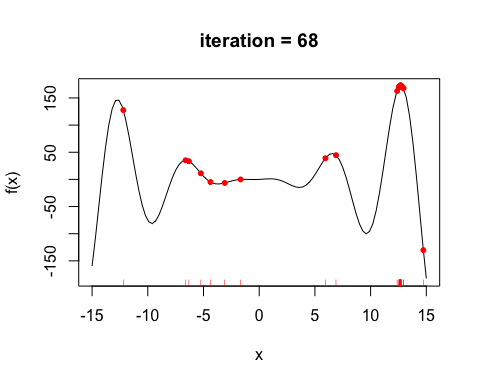
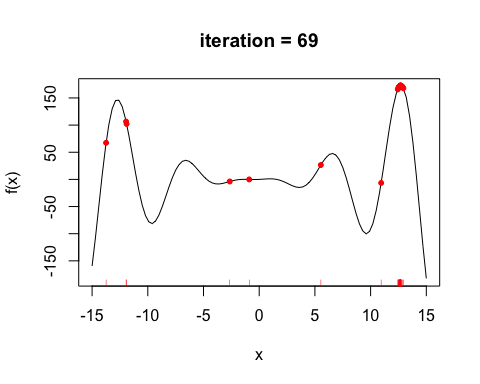
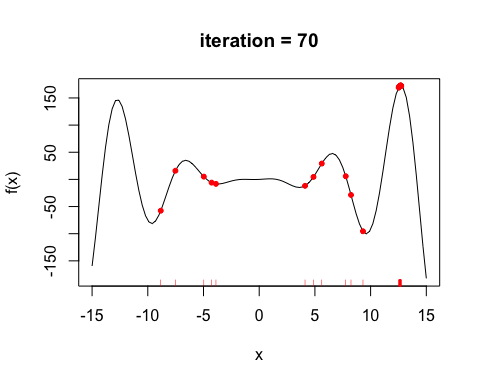
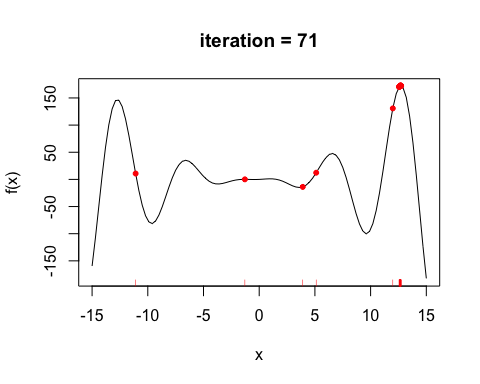
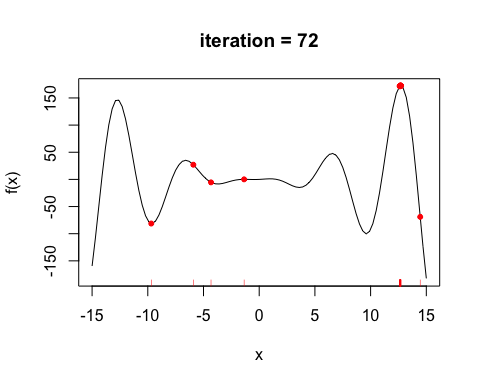
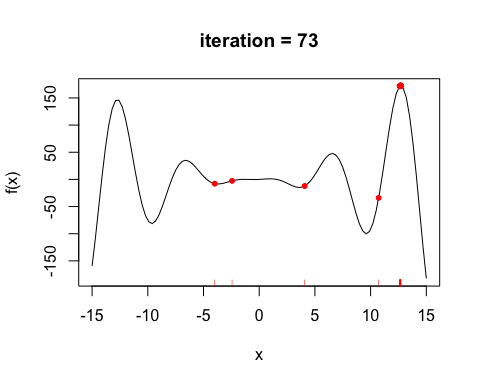
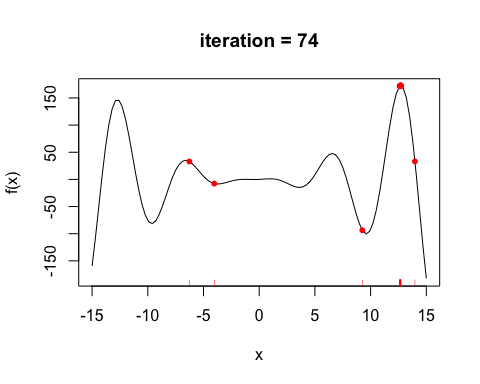
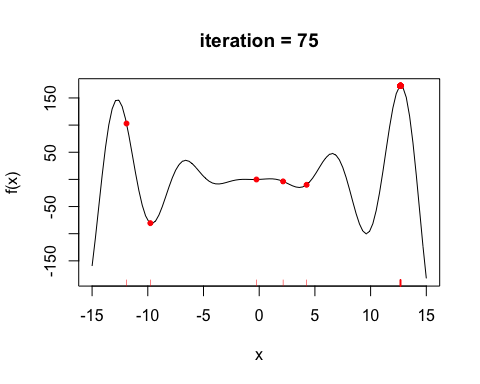
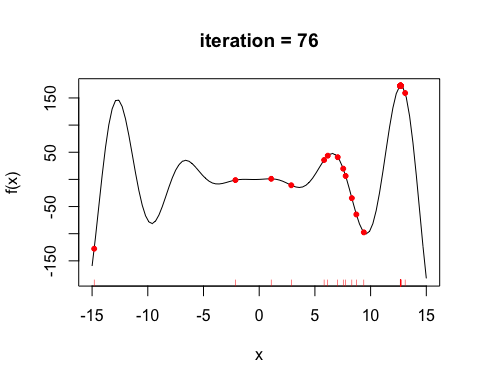
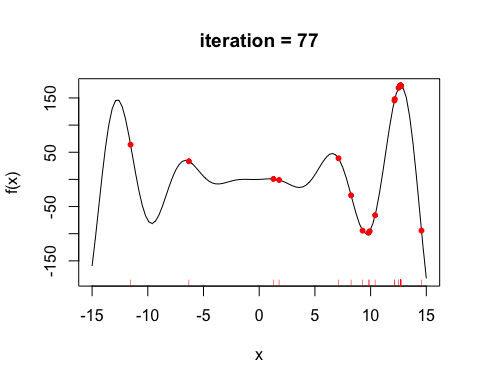
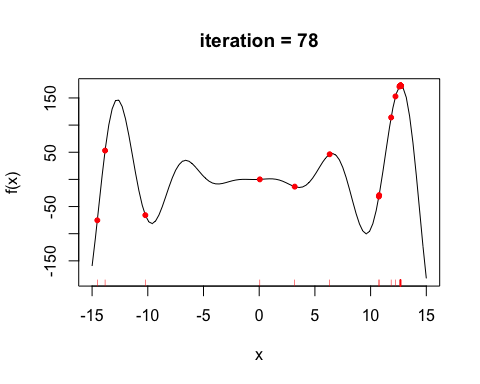
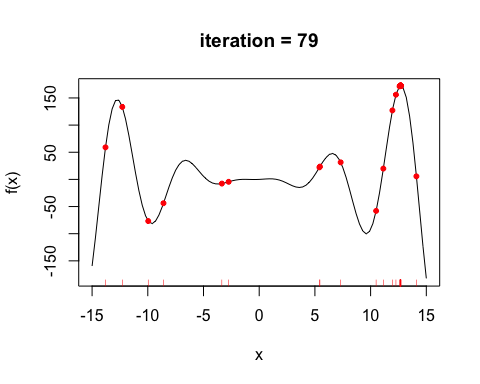
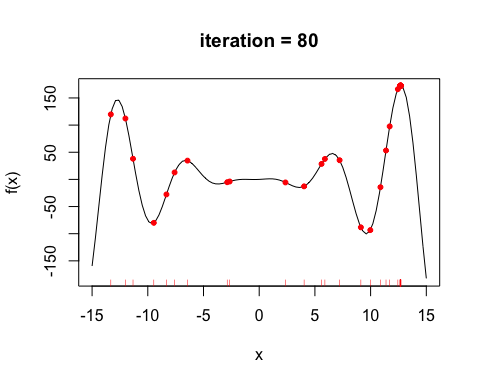
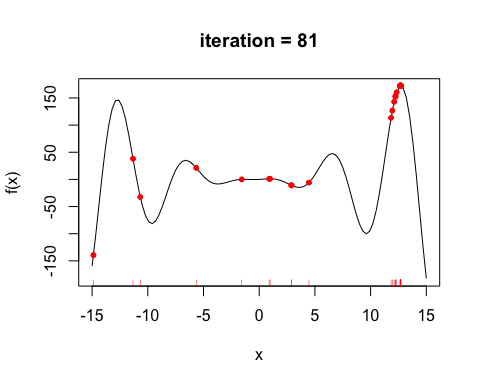
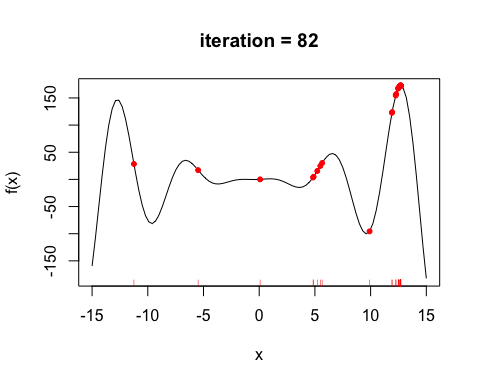
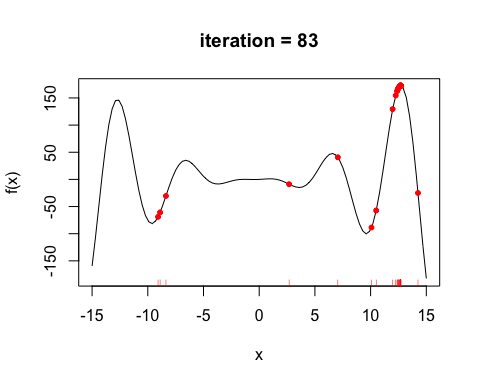
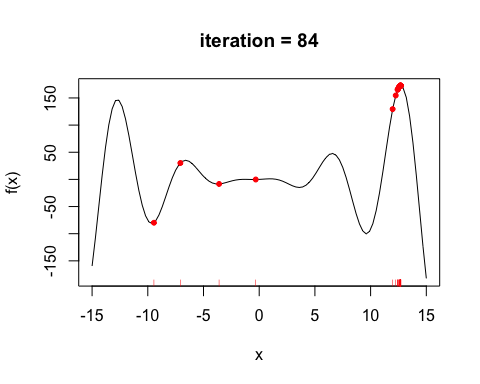
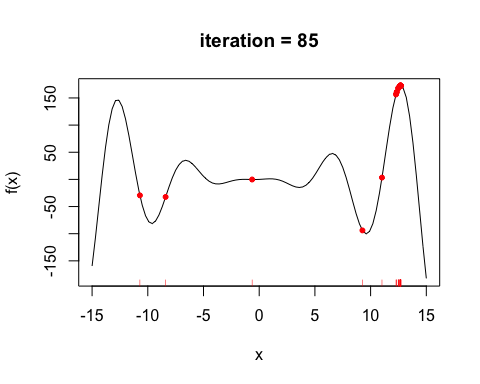
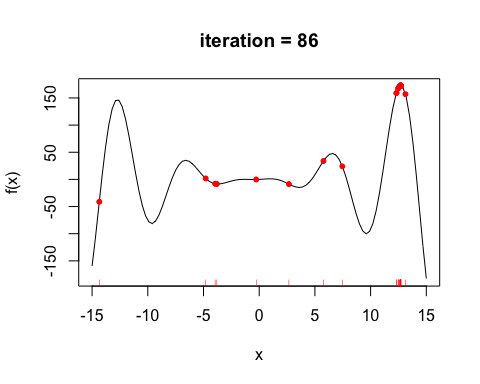
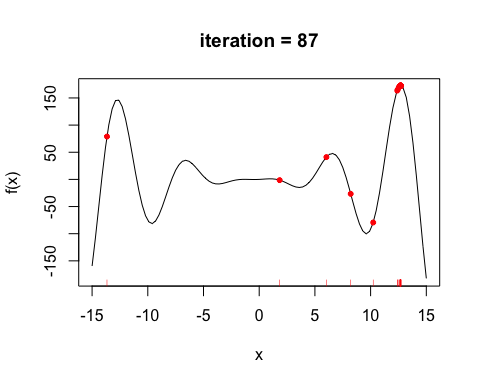
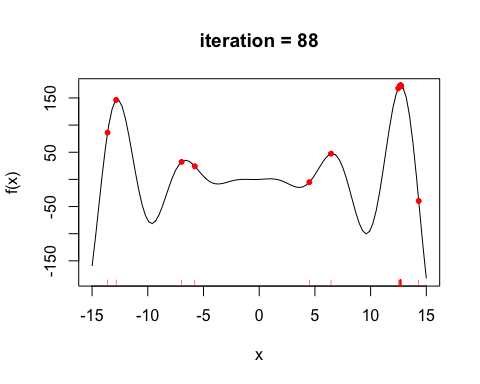
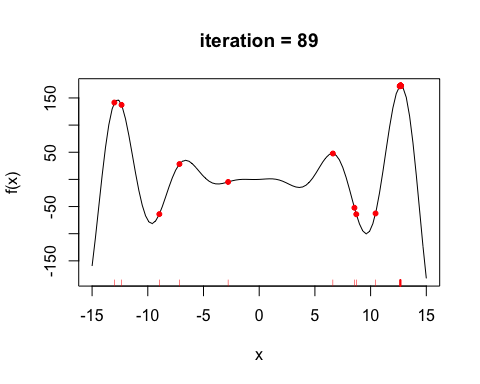
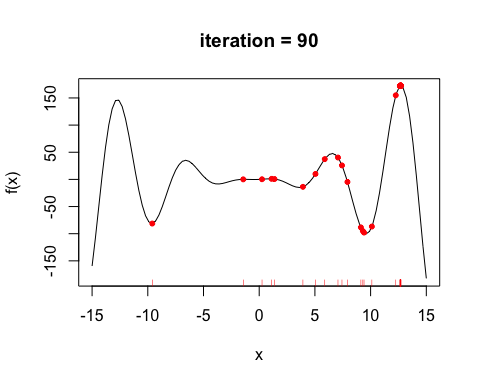
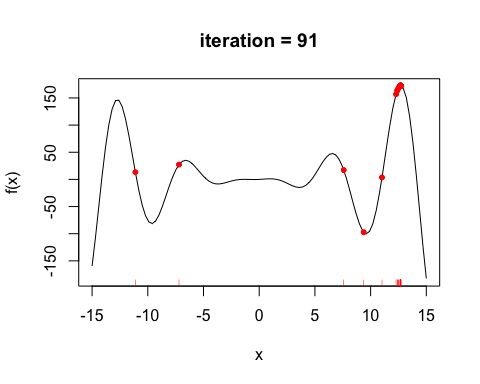
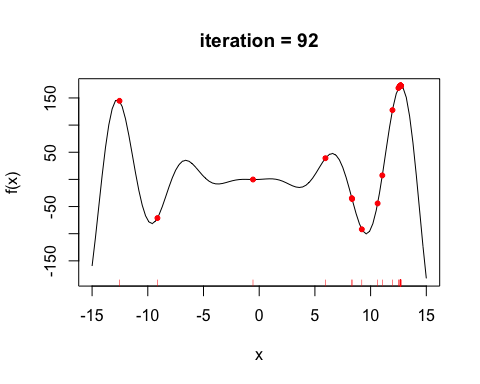
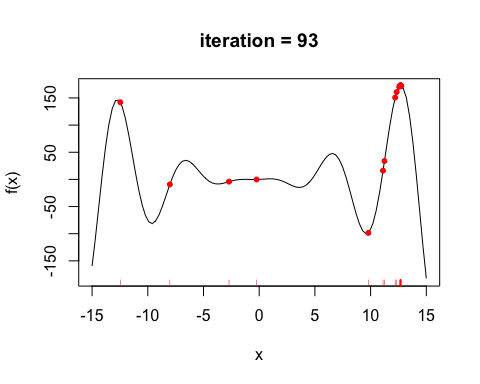
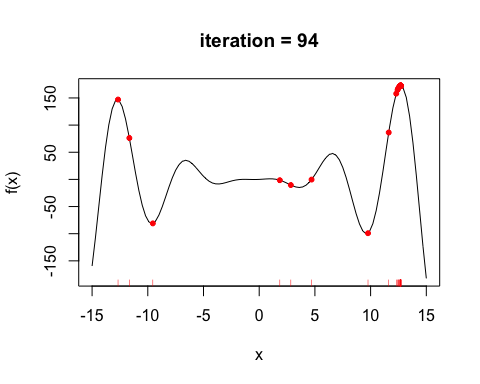
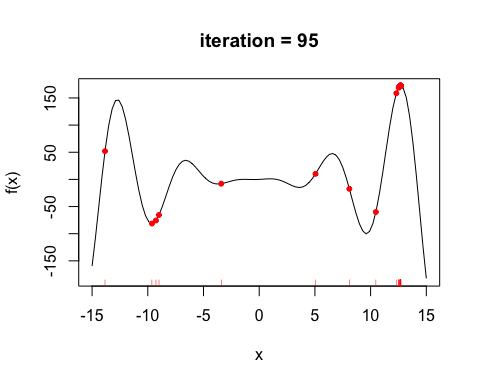
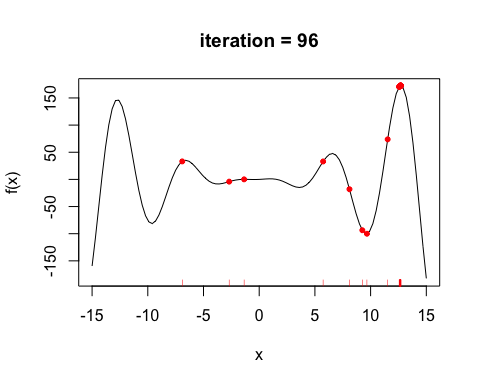
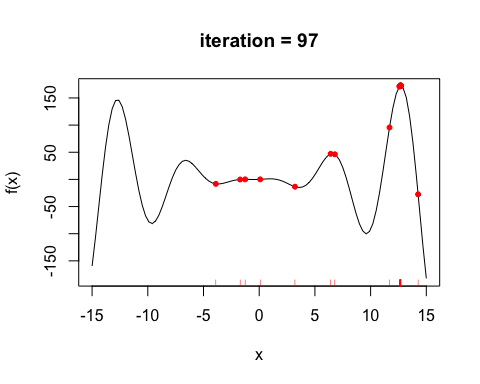
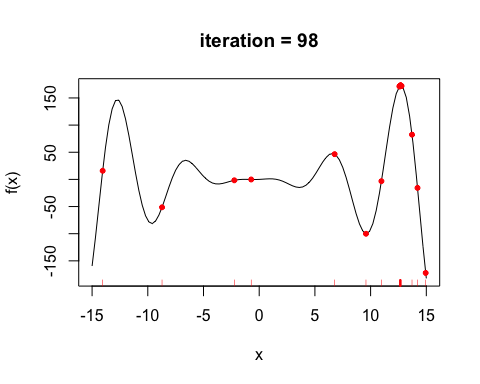
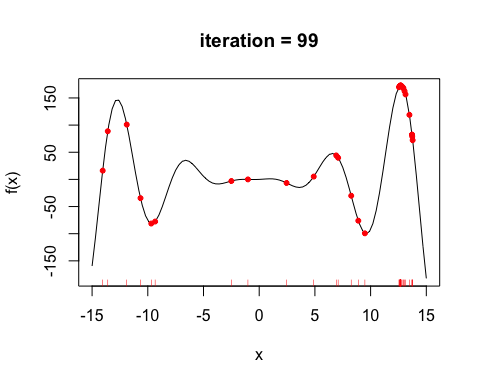
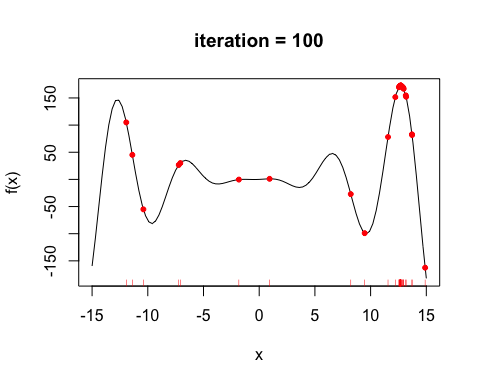
curve(f, -25, 25)  
abline(v = Gene\_Alg@solution, lty = 3)



#----------------------- 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)



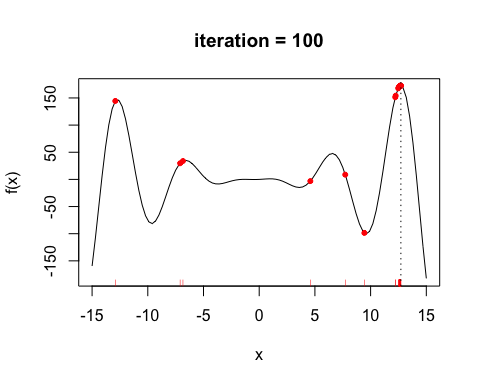
# tracing function  
monitor <- function(obj)  
{  
 curve(f, -15, 15, main = paste("iteration =", obj@iter))  
 points(obj@population, obj@fitness, pch = 20, col = 2)  
 rug(obj@population, col = 2)  
 Sys.sleep(0.2)  
}  
## For the maximization of this function we may use f directly as the fitness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

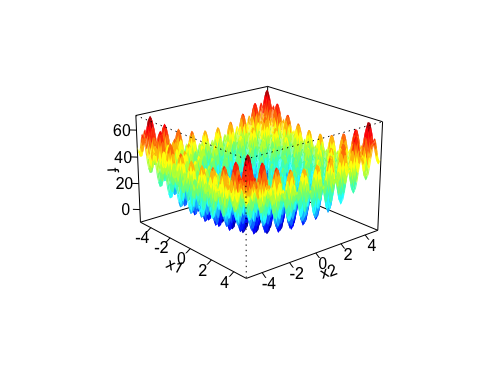
## End(Not run)  
# or if you want to suppress the tracing  
Gene\_Alg <- ga(type = "real-valued", fitness = f, min = -15, max = 15, monitor = NULL)  
summary(Gene\_Alg)

## +-----------------------------------+  
## | 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,] 12.71675

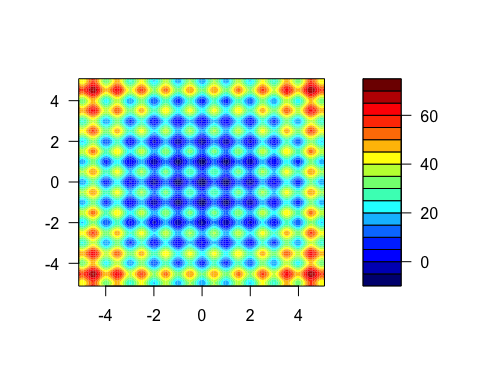
monitor(Gene\_Alg)  
abline(v = Gene\_Alg@solution, lty = 3)



# ----------------- 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)



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



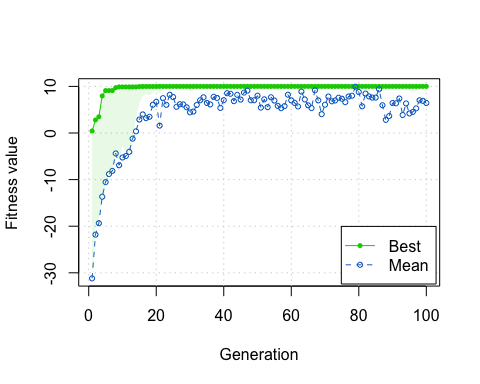
Gene\_Alg <- ga(type = "real-valued", fitness = function(x) -Rastrigin\_Fun(x[1], x[2]),  
 min = c(-5.12, -5.12), max = c(5.12, 5.12),  
 popSize = 50, maxiter = 100)

## Iter = 1 | Mean = -31.19921 | Best = 0.4348065   
## Iter = 2 | Mean = -21.80908 | Best = 2.794041   
## Iter = 3 | Mean = -19.35144 | Best = 3.477235   
## Iter = 4 | Mean = -13.67543 | Best = 7.954995   
## Iter = 5 | Mean = -10.54185 | Best = 9.10502   
## Iter = 6 | Mean = -8.812362 | Best = 9.10502   
## Iter = 7 | Mean = -8.131187 | Best = 9.10502   
## Iter = 8 | Mean = -4.387105 | Best = 9.725084   
## Iter = 9 | Mean = -6.911992 | Best = 9.89092   
## Iter = 10 | Mean = -5.261357 | Best = 9.89092   
## Iter = 11 | Mean = -4.93912 | Best = 9.89092   
## Iter = 12 | Mean = -4.036814 | Best = 9.89092   
## Iter = 13 | Mean = -1.21995 | Best = 9.89092   
## Iter = 14 | Mean = 0.3641831 | Best = 9.89092   
## Iter = 15 | Mean = 2.888007 | Best = 9.974197   
## Iter = 16 | Mean = 3.986708 | Best = 9.978242   
## Iter = 17 | Mean = 3.193413 | Best = 9.984541   
## Iter = 18 | Mean = 3.482475 | Best = 9.984541   
## Iter = 19 | Mean = 6.063661 | Best = 9.984541   
## Iter = 20 | Mean = 6.689973 | Best = 9.984541   
## Iter = 21 | Mean = 1.587819 | Best = 9.999695   
## Iter = 22 | Mean = 7.475563 | Best = 9.999695   
## Iter = 23 | Mean = 6.049682 | Best = 9.999695   
## Iter = 24 | Mean = 8.204117 | Best = 9.999695   
## Iter = 25 | Mean = 7.719887 | Best = 9.999695   
## Iter = 26 | Mean = 5.623631 | Best = 9.999695   
## Iter = 27 | Mean = 6.176074 | Best = 9.999695   
## Iter = 28 | Mean = 6.126327 | Best = 9.999695   
## Iter = 29 | Mean = 5.5162 | Best = 9.999695   
## Iter = 30 | Mean = 4.432498 | Best = 9.999695   
## Iter = 31 | Mean = 4.595195 | Best = 9.999695   
## Iter = 32 | Mean = 6.03143 | Best = 9.999695   
## Iter = 33 | Mean = 7.024147 | Best = 9.999695   
## Iter = 34 | Mean = 7.666912 | Best = 9.999695   
## Iter = 35 | Mean = 6.45237 | Best = 9.999695   
## Iter = 36 | Mean = 6.133739 | Best = 9.999695   
## Iter = 37 | Mean = 7.762466 | Best = 9.999695   
## Iter = 38 | Mean = 7.528125 | Best = 9.999695   
## Iter = 39 | Mean = 5.382778 | Best = 9.999695   
## Iter = 40 | Mean = 7.046204 | Best = 9.999695   
## Iter = 41 | Mean = 8.538338 | Best = 9.999695   
## Iter = 42 | Mean = 8.418167 | Best = 9.999695   
## Iter = 43 | Mean = 6.83574 | Best = 9.999695   
## Iter = 44 | Mean = 8.195128 | Best = 9.999695   
## Iter = 45 | Mean = 7.176094 | Best = 9.999695   
## Iter = 46 | Mean = 8.665425 | Best = 9.999695   
## Iter = 47 | Mean = 9.102639 | Best = 9.999695   
## Iter = 48 | Mean = 7.025588 | Best = 9.999695   
## Iter = 49 | Mean = 7.026516 | Best = 9.999695   
## Iter = 50 | Mean = 8.004545 | Best = 9.999695   
## Iter = 51 | Mean = 5.4521 | Best = 9.999695   
## Iter = 52 | Mean = 7.228564 | Best = 9.999699   
## Iter = 53 | Mean = 5.556304 | Best = 9.999699   
## Iter = 54 | Mean = 7.668071 | Best = 9.999699   
## Iter = 55 | Mean = 6.956952 | Best = 9.999699   
## Iter = 56 | Mean = 5.825354 | Best = 9.999699   
## Iter = 57 | Mean = 5.330261 | Best = 9.999699   
## Iter = 58 | Mean = 5.765951 | Best = 9.9997   
## Iter = 59 | Mean = 8.218666 | Best = 9.9997   
## Iter = 60 | Mean = 7.005841 | Best = 9.9997   
## Iter = 61 | Mean = 6.469658 | Best = 9.9997   
## Iter = 62 | Mean = 5.692056 | Best = 9.999701   
## Iter = 63 | Mean = 8.885755 | Best = 9.999701   
## Iter = 64 | Mean = 7.210121 | Best = 9.999701   
## Iter = 65 | Mean = 5.941806 | Best = 9.999701   
## Iter = 66 | Mean = 5.348057 | Best = 9.999701   
## Iter = 67 | Mean = 9.169632 | Best = 9.999701   
## Iter = 68 | Mean = 6.999929 | Best = 9.999701   
## Iter = 69 | Mean = 4.042517 | Best = 9.999701   
## Iter = 70 | Mean = 6.073743 | Best = 9.999701   
## Iter = 71 | Mean = 7.821705 | Best = 9.999701   
## Iter = 72 | Mean = 6.800491 | Best = 9.999701   
## Iter = 73 | Mean = 6.978645 | Best = 9.999991   
## Iter = 74 | Mean = 7.569177 | Best = 9.999991   
## Iter = 75 | Mean = 7.351291 | Best = 9.999991   
## Iter = 76 | Mean = 6.62159 | Best = 9.999991   
## Iter = 77 | Mean = 7.841418 | Best = 9.999991   
## Iter = 78 | Mean = 7.998132 | Best = 9.999991   
## Iter = 79 | Mean = 9.918223 | Best = 9.999991   
## Iter = 80 | Mean = 8.80369 | Best = 9.999991   
## Iter = 81 | Mean = 5.730466 | Best = 9.999991   
## Iter = 82 | Mean = 8.44972 | Best = 9.999991   
## Iter = 83 | Mean = 7.860493 | Best = 9.999991   
## Iter = 84 | Mean = 7.592363 | Best = 9.999995   
## Iter = 85 | Mean = 7.597684 | Best = 9.999995   
## Iter = 86 | Mean = 9.397382 | Best = 9.999995   
## Iter = 87 | Mean = 5.959629 | Best = 9.999995   
## Iter = 88 | Mean = 2.826527 | Best = 9.999995   
## Iter = 89 | Mean = 3.644326 | Best = 9.999995   
## Iter = 90 | Mean = 6.440913 | Best = 9.999995   
## Iter = 91 | Mean = 6.372425 | Best = 9.999995   
## Iter = 92 | Mean = 7.40006 | Best = 9.999995   
## Iter = 93 | Mean = 3.868545 | Best = 9.999995   
## Iter = 94 | Mean = 6.341909 | Best = 9.999995   
## Iter = 95 | Mean = 4.199308 | Best = 9.999995   
## Iter = 96 | Mean = 4.528206 | Best = 9.999995   
## Iter = 97 | Mean = 5.281694 | Best = 9.999995   
## Iter = 98 | Mean = 7.027513 | Best = 9.999995   
## Iter = 99 | Mean = 6.892478 | Best = 9.999995   
## Iter = 100 | Mean = 6.433351 | Best = 9.999995

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 = 9.999995   
## Solution =   
## x1 x2  
## [1,] 0.0001663998 -3.750458e-06

plot(Gene\_Alg)



# Genetic Algorithms in R

We will use Genetic Algorithms to solve the [knapsack problem](https://en.wikipedia.org/wiki/Knapsack_problem). The version of the knapsack problem problem being solved is the 0-1 knapsack problem, which restricts the number 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 and a value , along with a maximum weight capacity ,

Here 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,   
 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

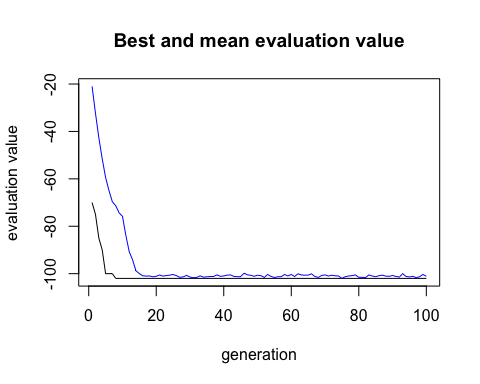
solution = c(0, 1, 1, 1, 1, 1, 1)  
knap[solution == 1, ]

## 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

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

## 102 / 117

plot(my\_GA\_Model)



# Assingment

* Use a genetic algorithm to search for a solution to the famous Travelling Salesman Problem in R. The same problem that we used in the simulated annealing lesson.
* Answer the following questions:
  + How does the genetic algorithm solution compare to the simulated annealing solution?
  + How does the crossover probability effect the genetic algorithm solution?
  + How does the mutation probability effect the genetic algorithm solution?

# Resources

* [Genetic algorithms: a simple R example](http://www.r-bloggers.com/genetic-algorithms-a-simple-r-example/)
* [Genetic algorithms](http://www.obitko.com/tutorials/genetic-algorithms/index.php)
* [Using Genetic Algorithms in Quantitative Trading](http://www.thertrader.com/2014/03/14/using-genetic-algorithms-in-quantitative-trading/)

```