Assignment 3 - Nonlinear optimization

Andrea Piancone

Contents

1 Problem 1 **1**
1.1 Ricerca dell'intervallo . . . 1
1.2 Bisezione in R: do-it-yourself . . . 2
1.2.1 Ricerca della prima radice . . . 3
1.2.2 Ricerca della seconda radice . . . 3
1.2.3 Ricerca della terza radice . . . 3
1.3 Bisezione in R: NLRoot package . . . 3
1.3.1 Ricerca della prima radice . . . 3
1.3.2 Ricerca della seconda radice . . . 4
1.3.3 Ricerca della terza radice . . . 4

2 Problem 2 **4**
2.1 Funzione obiettivo . . . 4
2.2 Gradiente e matrice Hessiana della funzione . . . 6
2.3 Metodo del gradiente discendente . . . 7
2.4 Metodo di Newton . . . 7

3 Problem 3 **8**
3.1 Funzione obiettivo . . . 8
3.2 Soluzione iniziale . . . 8
3.3 Generare una soluzione candidata . . . 8
3.4 Criterio di accettazione . . . 9
3.5 Simulated Annealing . . . 9

```
library(ggplot2)
library(NLRoot)
library(magrittr)
library(Hmisc)
library(ggforce)
library(optimx)
```

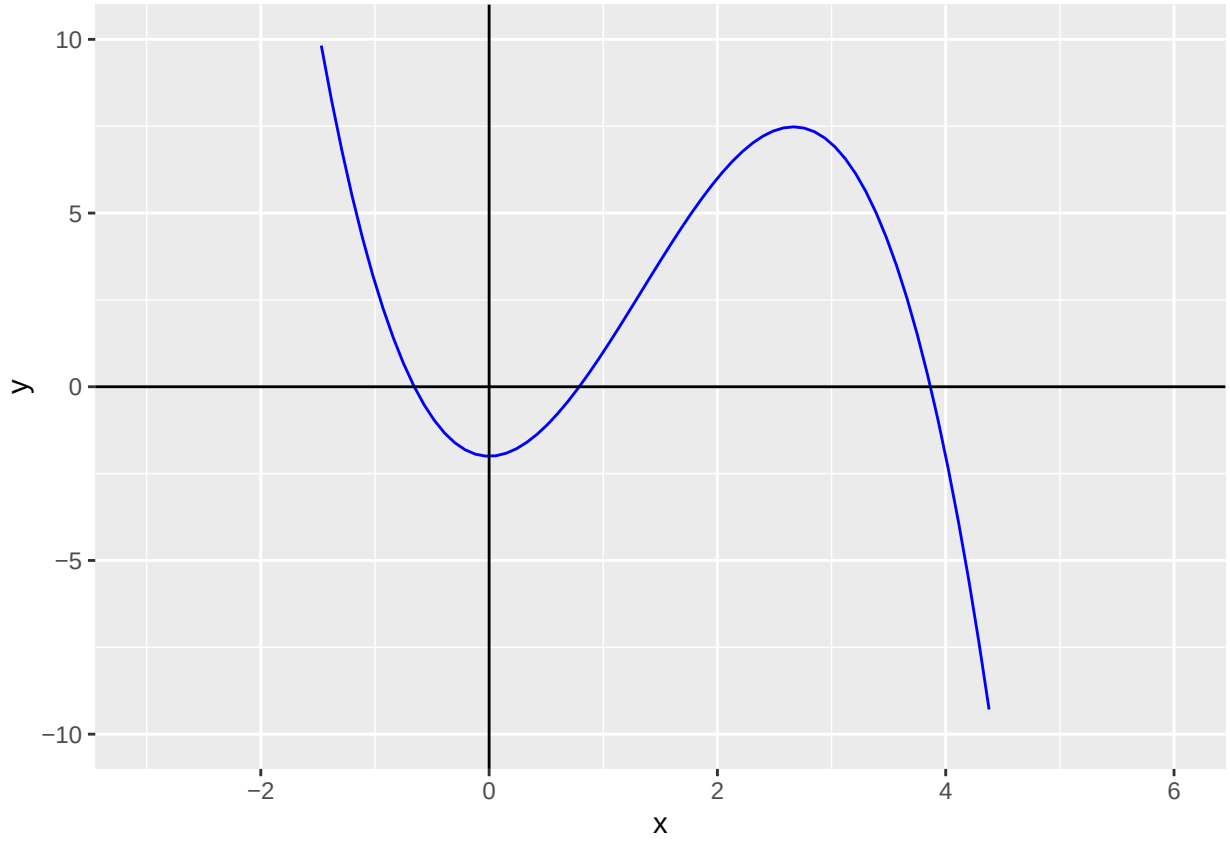
1 Problem 1

Using the bisection method calculate at least one zero for $f(x) = -x^3 + 4x^2 - 2$ starting for a suitable initial guess.

1.1 Ricerca dell'intervallo

In primo luogo, si proietta la funzione sul piano Cartesiano per trovare l'intervallo di valori all'interno del quale si trova la radice della funzione o le radici della funzione nel caso in cui ne ha più di una.

```r
fun1 <- function(x) {-x^3 + 4*x^2 -2}
p <- ggplot(data = data.frame(x = 0), mapping = aes(x = x))
p + stat_function(fun = fun1, col = 'blue') + xlim(-3,6) + geom_hline(yintercept = 0) + geom_vline(xinte
  ylim(-10,10)
```



La funzione possiede 3 radici, e tutte sono contenute nell'intervallo [-3, 6]. Inoltre, si può dimostrare tramite il teroema di Ruffini che un polinomio di grado n ha al più n radici, si ha che tutte le radici della funzione si concentrano nell'intervallo selezionato. Poichè ci sono più radici, la scelta dell'intervallo in cui cercare la radice influenzerà il risultato del metodo di Bisezione. Partendo da destra si ha che:

- La prima radice si trova nell'intervallo (3, 4)
- La seconda radice si trova nell'intervallo (0,1)
- La terza radice si trova nell'intervallo (-1, 0)

1.2 Bisezione in R: do-it-yourself

Dapprima si impiega una funzione "fai da te" per trovare la radice di una funzione in un intervallo con il metodo di Bisezione. Il metodo di Bisezione richiede come "parametri in ingresso" la funzione, gli estremi dell'intervallo in cui cercare lo zero, una soglia di tolleranza di errore e il numero massimo di iterazioni consentite. Si definisce dunque la funzione con cui eseguire il metodo di Bisezione.

```r
bisection <- function(f, a, b, n = 1000, tol = 1e-7) {
  if (sign(f(a) == sign(f(b)))) {
    stop('signs of f(a) and f(b) must differ')
  }
  for (i in 1:n) {
    c <- (a + b)/2
```

```
    if ((f(c) == 0) || ((b - a) / 2) < tol) {
      return(c)
    }
  ifelse(sign(f(c)) == sign(f(a)),
         a <- c,
         b <- c )
  }
  print('Too many iterations')
}
```

1.2.1 Ricerca della prima radice

Come anticipato, la prima radice si trova nell'intervallo (3,4), quindi tramite la funzione *bisetion* cerchiamo lo zero della funzione in tale intervallo.

```
x1 <- bisection(fun1, 3, 4)
cat('La radice x1 della funzione è: ', x1)
```

```
## La radice x1 della funzione è:  3.866198
```

1.2.2 Ricerca della seconda radice

La seconda radice della funzione si trova nell'intervallo (0, 1). Tramite la funzione *bisection* si trova la radice della funzione in tale intervallo.

```
x2 <- bisection(fun1, 0, 1)
cat('La radice x2 della funzione è: ', x2)
```

```
## La radice x2 della funzione è:  0.7892441
```

1.2.3 Ricerca della terza radice

La terza radice della funzione si trova nell'intervallo (-1, 0). Tramite la funzione *bisection* si trova la radice della funzione in tale intervallo.

```
x3 <- bisection(fun1, -1, 0)
cat('La radice x3 della funzione è: ', x3)
```

```
## La radice x3 della funzione è:  -0.6554424
```

1.3 Bisezione in R: NLRoot package

Ora si cercano nuovamente gli zeri della funzione, tramite il comando *BFfzero*, della libreria *NLRoot*.

1.3.1 Ricerca della prima radice

```
x1 <- BFfzero(fun1, 3, 4)
```

```
## [1] 1
## [1] 3.866196
## [1] 3.594802e-05
## [1] "finding root is successful"
```

```
print(x1)
```

```
## [1] "finding root is successful"
```

1.3.2 Ricerca della seconda radice

```
x2 <- BFfzero(fun1, 0, 1)
```

```
## [1] 1
## [1] 0.7892426
## [1] -6.958254e-06
## [1] "finding root is successful"
```

```
print(x2)
```

```
## [1] "finding root is successful"
```

1.3.3 Ricerca della terza radice

```
x3 <- BFfzero(fun1, -1, 0)
```

```
## [1] 1
## [1] -0.6554413
## [1] -7.168402e-06
## [1] "finding root is successful"
```

```
print(x3)
```

```
## [1] "finding root is successful"
```

Come si osserva, le due funzioni impiegate portano a valori molto simili degli zeri della funzione. In sintesi, arrotondati alla quarta cifra decimale, le radici della funzione sono:

Radice	Valore
x_1	3.8662
x_2	0.7892
x_3	-0.6554

2 Problem 2

Consider the following minimization problem: $min\ f(x_1, x_2) = 2x_1^2 + x_1x_2 + 2(x_2 - 3)^2$.

1. Apply an iteration of the gradient method by performing the line search in an exact way, starting from the point $A = (-1, 4)^T$ Report all the steps of the method, not just the result.
2. Apply an iteration of Newton's method from point A. Verify that the point found is the minimum of function f. Report all the steps of the method, not just the result.
3. How many iterations of Newton's method are required to optimize a quadratic function?

2.1 Funzione obiettivo

La funzione obiettivo da minimizzare è la seguente: $min\ f(x_1, x_2) = 2x_1^2 + x_1x_2 + 2(x_2 - 3)^2$. Si defisce la funzione obiettivo tramite codice R.

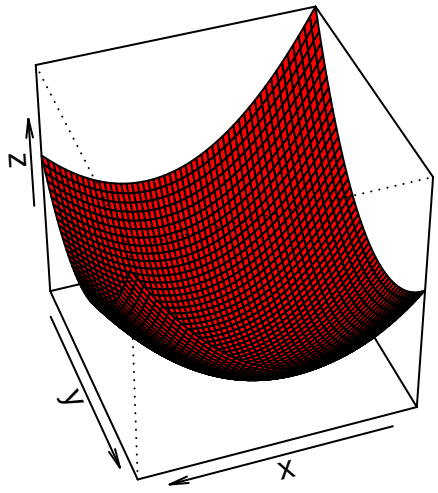
```
fun2 <- function (x) {
   return (2*x[1]^2 + x[1]*x[2] + 2*(x[2] - 3)^2)
}
```

In primo luogo si decide di visualizzare la funzione in piano tridimensionale per verificare l'eventuale presenza di ottimi locali e globali.

```r
fun2_bis <- function (x,y) {
   return (2*x^2 + x*y + 2*(y - 3)^2)
}
```

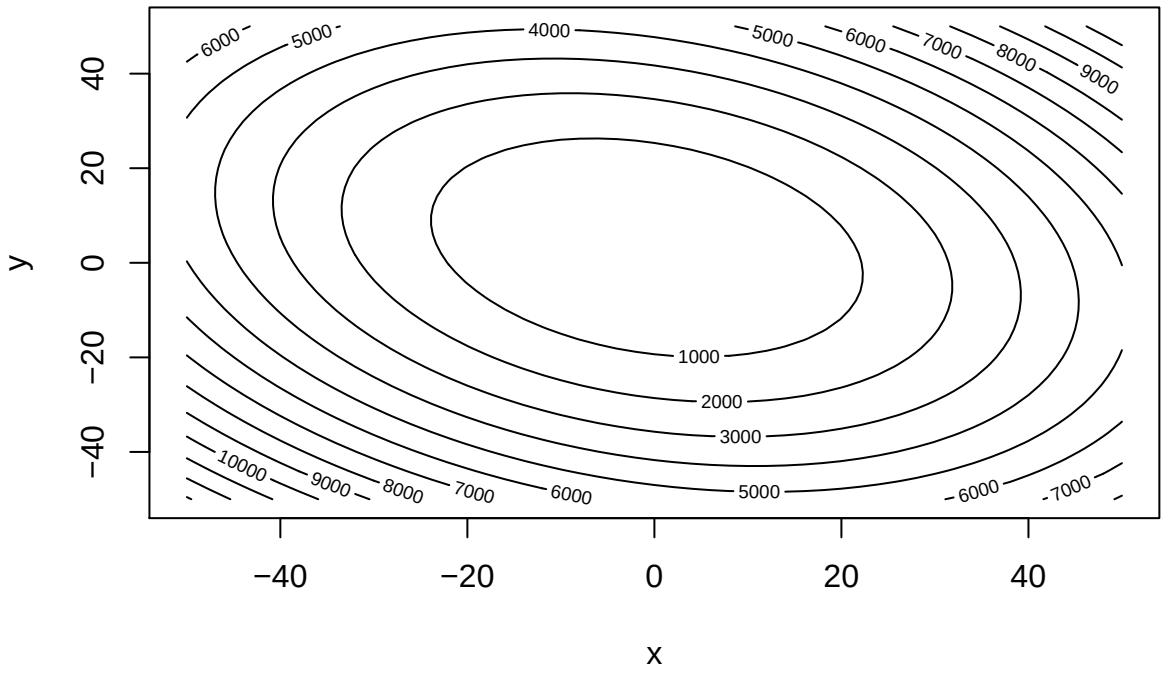
```r
x <- seq(-50, 50, 2)
y <- seq(-50, 50, 2)
z <- outer(x, y, fun2_bis)
z[is.na(z)] <- 1

persp(x, y, z, col = "red", theta = 160, phi = 40, r = 10, d = 3)
```



La funzione oggetto di studio è convessa con un unico punto di minimo globale. Si procede a visualizzare le *contours lines* della funzione da minimizzare.

```r
contour(x, y, z, xlab = "x", ylab = "y")
```



Il contour plot conferma la convessità della funzione e l'esistenza di un unico punto di minimo globale. Date queste premesse il metodo del gradiente discendente ed il metodo di Newton portano ad individuare il punto di minimo globale, senza incorrere nel rischio di trovare un punto di minimo locale, non essendo presenti tali punti all'interno del dominio della funzione.

2.2 Gradiente e matrice Hessiana della funzione

Per poter svolgere una singola iterazione del metodo del gradiente discendente e del metodo di Newton, è necessario ricavare il gradiente della funzione e la sua matrice Hessiana.

Si iniza calcolando il gradiente della funzione $f(x_1, x_2)$, che è pari a:

$$\nabla_f = [4x_1 + x_2, x_1 + 4x_2 - 12]$$

. Il gradiente verrà usato nel metodo numerico per cercare il minimo locale.

Si calcola anche la matrice Hessiana, al fine di stabilire se il punto stazionario individuato è di minimo e per poter svolgere il metodo di Newton.

$$H = \begin{pmatrix} 4 & 1 \\ 1 & 4 \end{pmatrix}$$

La sua inversa è data da:

$$H^{-1} = \begin{pmatrix} \frac{4}{15} & \frac{-1}{15} \\ \frac{-1}{15} & \frac{4}{15} \end{pmatrix}$$

2.3 Metodo del gradiente discendente

Partendo dal punto iniziale $x_0 = (-1, 4)$, il primo passo consiste nel valutare il gradiente della funzione nel punto x_0. Si ottiene quindi: $\nabla f(x_0) = [0, 3]$. Ora si deve verificare se il punto iniziale approssima in maniera soddisfacente il punto di minimo della funzione. A tal fine si calcola la norma del vettore gradiente valutato nel punto x_0, e se la norma risulta essere più piccola di un valore positivo piccolo a piacere che si fissa pari a 0.01, allora il punto x_0 sarà una buona approssimazione del punto di minimo della funzione, e quindi l'algoritmo si interrompe. In caso contario si passa dal punto x_0 al punto x_1 La norma del vettore $\nabla f(x_0)$ è pari a:

$$||\nabla f(x_0)|| = \sqrt{0^2 + 3^2} = \sqrt{9} = 3$$

Il criterio di convergenza non è rispettato. Quindi è necessario passare dal punto x_0 al punto x_1. Il punto x_1 è dato dalla seguente formula: $x_1 = x_0 - \alpha_0 \nabla f(x_0)$. Quindi operando le opportune sostituzioni:

$$x_1 = [-1, 4] - \alpha_0[0, 3] = [-1, 4] - [0, 3\alpha_0] = [-1, 4 - 3\alpha_0]$$

Si sceglie come valore di α_0 (che rappresenta lo stepsize), quello che minimizza la funzione in una variabile

$$f(x_0 - \alpha_0 \nabla f(x_0)) = -2 + 3\alpha_0 + 2(1 - 3\alpha_0)^2 = 18\alpha_0^2 - 9\alpha_0$$

Si tratta di una parabola con concavità rivolta verso l'alto. Quindi il suo vertice è il punto di minimo. L'ascissa di tale vertice si trova in corrispondeza di $\alpha_0 = -\frac{-9}{36} = \frac{1}{4}$. Quindi lo step size a questa iterazione è pari a 0.25. Si procede a calcolare le coordinate del punto x_1, che sono date da:

$$x_1 = [-1, 4 - 3(0.25)] = [-1, 3.25]$$

Si procede a verificare se questo punto soddisfa il criterio di convergenza:

$$\nabla f(x_1) = [-0.75, 0]$$

la cui norma è 0.75. Il punto x_1 non è il punto di minimo della funzione, in quanto il criterio di convergenza non è soddisfatto.

2.4 Metodo di Newton

Il metodo a partitre dal punto iniziale $x_0 = (-1, 4)$, poichè questo punto si sa che non è ottimale, il metodo di Newton deve proseguire. Si deve passare dal punto x_0, al punto x_1, definito come:

$$x_1 = x_0 - H^{-1}(x_0)\nabla f(x_0)$$

Quindi si deve valutare sia il gradiente della funzione che l'inversa della matrice Hessiana nel punto x_0. La matrice Hessiana, è già stata ricavata, ed ha tutti elementi che sono delle costanti. Il gradiente della funzione nel punto x_0 è stato ricavato nel punto precedente ed è dato dal seguente vettore: $[0, 3]$. Si dispongono di tutte le informazioni necessarie per trovare le coordinate del punto x_1.

$$x_1 = \begin{bmatrix} -1 \\ 4 \end{bmatrix} - \begin{pmatrix} \frac{4}{15} & \frac{-1}{15} \\ \frac{-1}{15} & \frac{4}{15} \end{pmatrix} \begin{bmatrix} 0 \\ 3 \end{bmatrix} = \begin{bmatrix} -1 \\ 4 \end{bmatrix} - \begin{bmatrix} -\frac{1}{5} \\ \frac{4}{5} \end{bmatrix} = [-0.8, 3.2]^T$$

Si valuta il criterio di convergenza calcolando la norma del vettore gradiente, valutato nel punto x_1. Si ha che:

$$\nabla f(x_1) = [0, 0]$$

Il punto x_1, annulla il gradiente, quindi esso è un punto stazionario. Per stabilire analiticamente se si tratta di un punto di minimo, si deve stabilire il segno della matrice Hessiana. A tal fine si ricavano i suoi autovalori:

$$H - \lambda I = \begin{pmatrix} 4 & 1 \\ 1 & 4 \end{pmatrix} - \begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix} = \begin{pmatrix} 4 - \lambda & 1 \\ 1 & 4 - \lambda \end{pmatrix}$$

Si ricava il suo determinante:

$$Det(H - \lambda I) = \lambda^2 - 8\lambda + 15$$

I valori di λ che annullano il determinante sono gli autovalori della matrice e sono: $\lambda_1 = 3$ e $\lambda_2 = 5$. Poichè gli autovalori della matrice Hessiana sono strettamente positivi, significa che essa è definita positiva, quindi il punto stazionario x_1 è un punto di minimo, che in questo caso è globale.

Non sorprende che il metodo di Newton è giunto a convergenza con una sola iterazione. Infatti, per le funzioni quadratiche (la funzione oggetto di ottimizzazione è di questa natura) è sufficiente una sola iterazione per il metodo di Newton per giungere a convergenza. In quanto si sta approssimando una funzione quadratica con un altra funzione quadratica.

3 Problem 3

Use the Simulated Annealing algorithm to find the global minimum of the following function:

$$f(x) = 34e^{-\frac{1}{2}(\frac{x-88}{2})^2} + (\frac{x}{10} - 2sin(\frac{x}{10}))^2$$

Notice that $f(x)$ may have several local optima, thus restarting and a careful selection of the algorithm parameters may be necessary.

3.1 Funzione obiettivo

La funzione obiettivo di cui bisogna trovre il minimo globale è:

$$f(x) = 34e^{-\frac{1}{2}(\frac{x-88}{2})^2} + (\frac{x}{10} - 2sin(\frac{x}{10}))^2$$

```
fun3 <- function(x) {
  return(34*exp(-1/2 * ((x - 88)/2)^2) + (x/10 - 2*sin(x/10))^2)
}
```

3.2 Soluzione iniziale

La scelta del punto iniziale condiziona fortemente il risultato finale dell'algoritmo di Simulated Annealing ed influenza fortemente le sue performace. Dunque è una scelta fondamentale. In assenza di informazione si genera un punto iniziale ricorrendo ad una distribuzione uniforme, definita tra -1000 e 1000.

3.3 Generare una soluzione candidata

Nel seguente codide, si definisce una funzione per generare una soluzione candidata partendo da quella corrente, aggiunendo a quella corrente una quantità proveniente da una distribuzione Normale, con media $\mu = 0$ e varianza $\sigma = 20$. Quindi indicando con x_k la soluzione attuale, la soluzione candidata è data da: $x_{k+1} = x_k + N(0, 20)$

```
CreateNewSolution <- function(x) {
  x <- x + rnorm(1, 0, 20)
  return(as.numeric(x))
}
```

3.4 Criterio di accettazione

Nel caso in cui la soluzione candidata porti ad un peggioramento della funzione obiettivo, l'algoritmo si simulated annealing non scarta a priori tale soluzione, ma l'accetta con un grado di probabilità che deve essere minore di una quantità ϵ. Per determinare tale quantità ϵ si ricorre al criterio di accettazione di Metropolis, il quale si basa sulla distribuzione di Boltzman.

$$\epsilon = e^{\frac{f(x_{k+1})-f(x_k)}{T_{k+1}}}$$

Dove T_{k+1} è la temperatura.

```
acceptanceCriterion <- function(fcandidate, fcurrent,fbest, temperature) {
  if (fcandidate < fbest) {
    return(TRUE)
  }
  else {
    if(exp((fcurrent-fcandidate)/temperature)>runif(1, 0, 1)) {
      return(TRUE)
    }
    return(FALSE)
  }
}
```

3.5 Simulated Annealing

Si crea il codice per per costruire l'algoritmo di simulated annealing.

```
SA <- function(start.point, obj.fun, maxIterNoChange=200) {
  solutiuon <- start.point #soluzione iniziale
  tmin <- 0.0001           #temperatura minima
  T.init <- 1000           #temperatura iniziale
  coolingRate <- 0.999
  Value <- obj.fun(solutiuon)
  BestValue <- Value
  TraceBest <- c(Value)
  TraceCurrentBest <- c(Value)
  IterNoChange <- 0

  while (T >= tmin) {
    IterNoChange = IterNoChange + 1
    NewSolution <- CreateNewSolution(solutiuon) #Soluzione candidata
    new_value <- obj.fun(NewSolution)

    if(acceptanceCriterion(new_value, Value, BestValue, T.init)) {
      if(new_value <= BestValue) {
        BestValue <- new_value
      }
      solutiuon <- NewSolution
      Value <- new_value
      IterNoChange <- 0
    }
    T.init <- coolingRate * T.init
    TraceBest <- append(TraceBest, BestValue)
    TraceCurrentBest <- append(TraceCurrentBest, Value)
    if(IterNoChange >= maxIterNoChange){ break}
```

```
  }
  res <- list(Value <- solutiuon,
              ObJ.Value <- obj.fun(solutiuon),
              TraceBest <- TraceBest,
              Trace <- TraceCurrentBest)
  class(res) <- "SAObj"
  names(res) <- c("xValue", "yValue", "TraceBest", "Trace")
  return(res)
}
```

Poichè la funzione di cui si vuole trovare il minimo globale, può avere diversi ounti di minimo locale, una sola iterazione potrebbe non essere sufficiente per trovare l'ottimo globale, in quanto l'iterazione potrebbe partire da un punto iniziale che blocca l'algoritmo in un punto di ottimo locale. Per superare questo problema si decide di ricorrere a 100 iterazioni, dove ad ogni iterazione il punto iniziale viene generato da una distribuzione uniforme che assume valori compresi tra -1000 e 1000. Al fine della procedura si sceglie di considerare come ottimo globale, il valore della x che consente di raggiungere il valore più basso della funzione obiettivo.

```
set.seed(123456789)
lista_soluzioni_x <- c()
lista_soluzioni_y <- c()
for (i in seq(1,100)) {
  start <- runif(1, -1000, 1000)
  global_minimun <- SA(start, fun3, 1000)
  lista_soluzioni_x <- append(lista_soluzioni_x, global_minimun$xValue)
  lista_soluzioni_y <- append(lista_soluzioni_y, global_minimun$yValue)
}

tmp_df <- data.frame(cbind(lista_soluzioni_x, lista_soluzioni_y))
  colnames(tmp_df) <- c("x", "y")
  optimal_solution <- tmp_df[which.min(tmp_df$y),]
  cat("Il minimo globale si trova in x = ", optimal_solution$x)
```

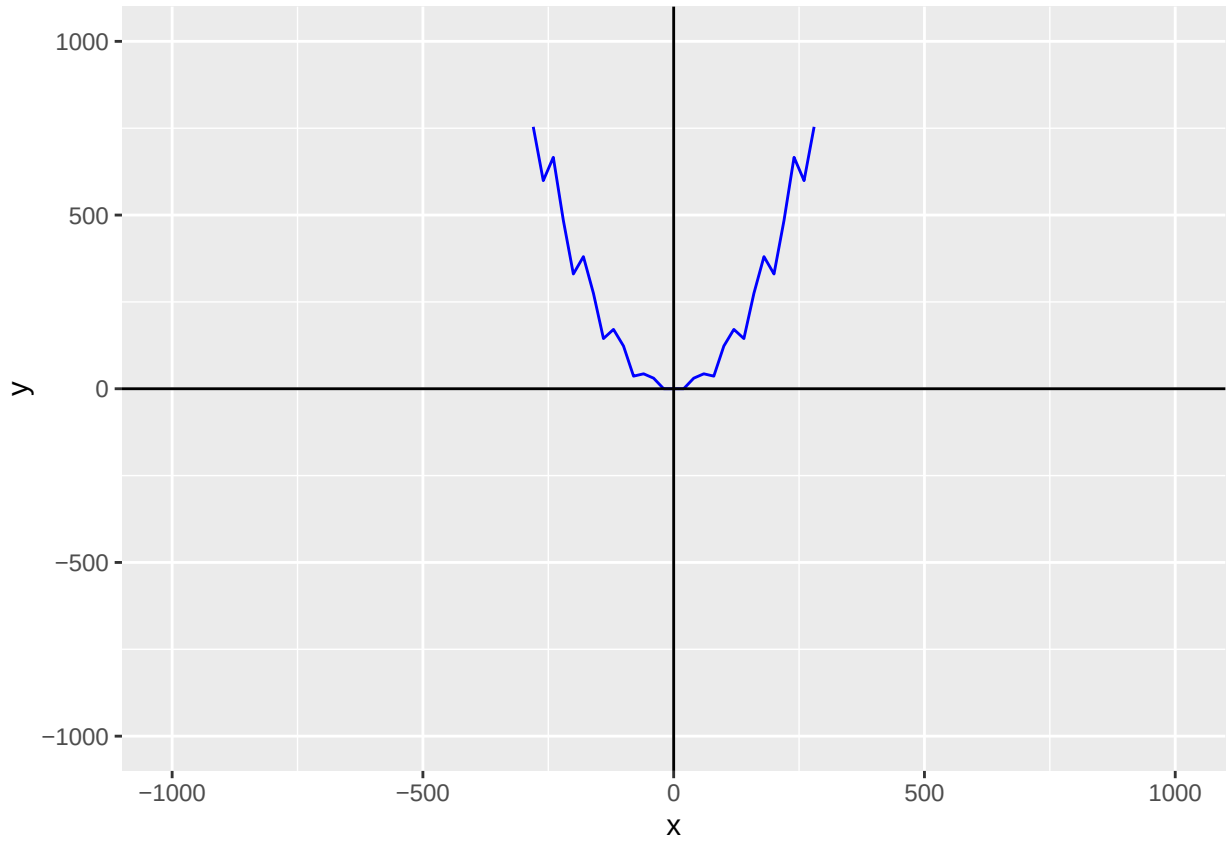
```
## Il minimo globale si trova in x =  -0.0001943005
```

```
  cat("Il valore della funzione obiettivo in corrispondenza del minimo globale è: ",
      optimal_solution$y)
```

```
## Il valore della funzione obiettivo in corrispondenza del minimo globale è:  3.775268e-10
```

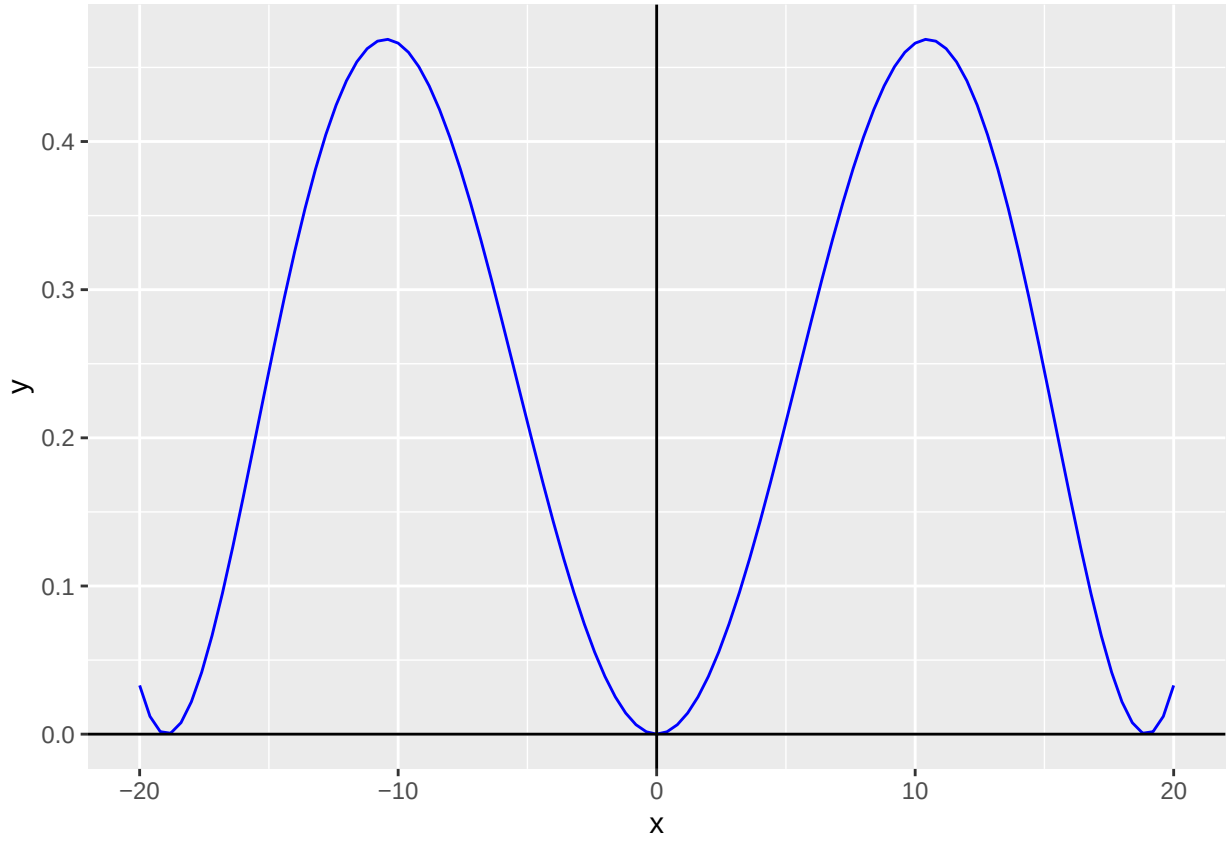
Il minimo globale si trova in un intorno di 0 ed in corrispondenza di tale valore di ascissa si osserva che la funzione obiettivo assume un valore molto prossimo a 0. Dunque è ragionevole ritenere che il punto di minimo globale, si trova in corrispondenza del punto di coordinate (0, 0), quindi nell'origine degli assi. Per avere la conferma della soluzione trovata dall'algoritmo di Simulated Annealing, si può visualizzare la funzione sul piano cartesiano.

```
p <- ggplot(data = data.frame(x = 0), mapping = aes(x = x))
p + stat_function(fun = fun3, col = 'blue') + geom_hline(yintercept = 0) + geom_vline(xintercept = 0) +
```



Il grafico sembra confermare la presenza di un minimo globale in corrispondenza dell'origine degli assi. Tuttavia, per avere maggiore chiarezza si decide di zoomare su tale grafico focalizzando l'attenzione nell'interavllo [-20, 20].

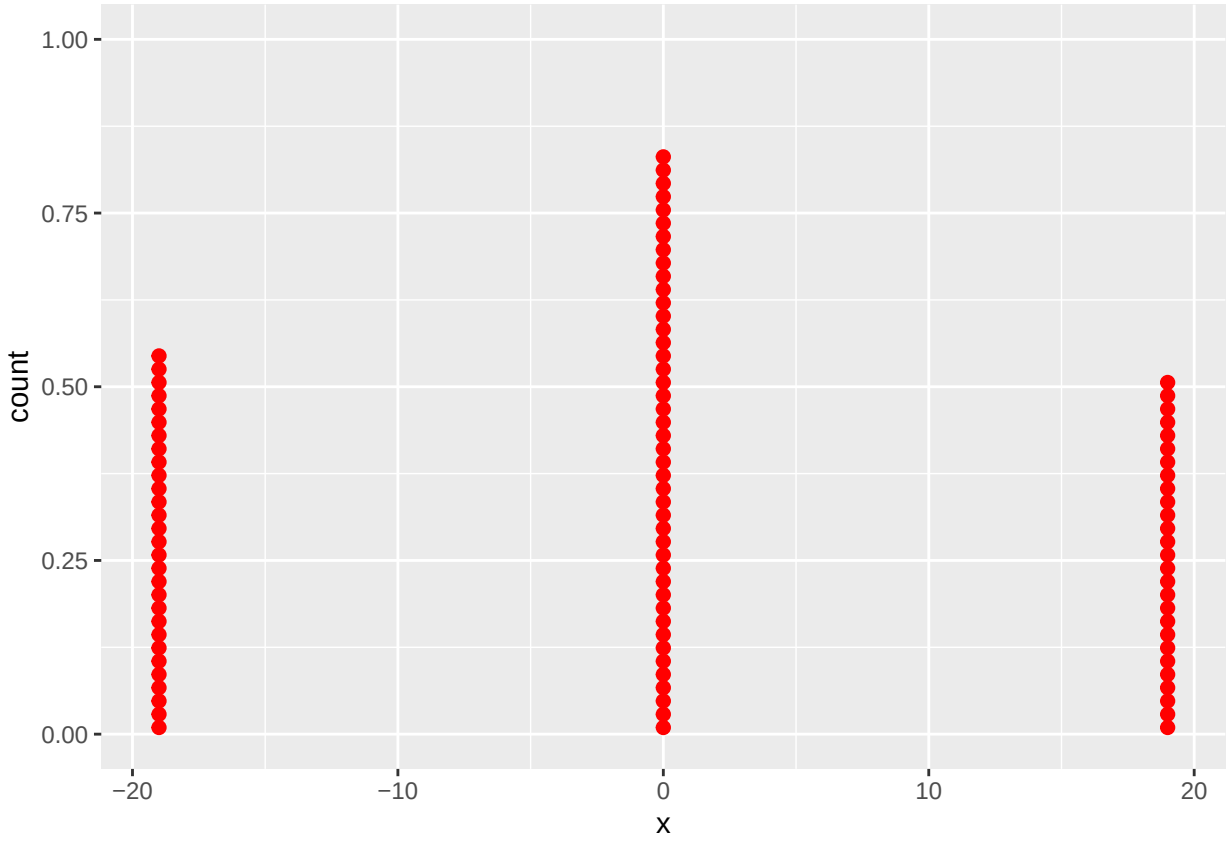
```
p <- ggplot(data = data.frame(x = 0), mapping = aes(x = x))
p + stat_function(fun = fun3, col = 'blue') + geom_hline(yintercept = 0) + geom_vline(xintercept = 0) +
```



Zoomando su tale intervallo, sembrano esserci altri due punti di minimo globale vicini al punto -20 e 20. In corrispondenza di tali punti la funzione obiettivo assume il vaore zero, come nell'origine degli assi.

Si decide di ricorrere ad un dotplot per verificare se l'algoritmo ha individuato questi punti.

```
ggplot(tmp_df, aes(x = x)) + geom_dotplot(method="histodot",
                                          binwidth = 0.5,
                                          col = "red", fill = "red")
```



Dal dotplot emerge che la funzione ha trovato tre "classi" di punti di ascisssa:

1. Punti che si concentrano in un intorno di -19;
2. Punti che si concentrano in un intorno di 0;
3. Punti che si concentrano un un intorno di 19.

Si può concludere che la funzione possiede tre punti di minimo globale, in corrispondenza dei quali assume un valore pari a zero e l'algoritmo del simulated annealing è stato in grado di trovare tali punti di minimo globale.

In sentesi i punti di minimo globale si trovano in prossimità di -19, nell'origine, e in prossimità di +19.