Optimización inteligente

Tarea 6. Estimaciones Cuadráticas Sucesivas.

Ángel García Báez

2024-10-18

Contents

1 Breve introducción

Se planteo un ejercicio en clase donde se pedía encontrar el valor de x que lograra encontrar el mínimo global de la función $f(x) = x^2 + \frac{54}{x}$ haciendo uso del método conocido como ***estimaciones cuadráticas sucesivas***.

A continuación se muestra el pseudocódigo resumido de lo que busca hacer el algoritmo:

1.1 Método de busqueda de estimaciones cuadráticas sucesivas

Algoritmo

Paso 1: Hacer que x_1 sea un punto inicial y Δ sea el tamaño del paso (o incremento). Pedir $TOL1$ y $TOL2$

Calcular $x_2 = x_2 + \Delta$

Paso 2: Evaluar $f(x_1)$ y $f(x_2)$

Paso 3: IF $f(x_1) > f(x_2)$ THEN $x_3 = x_1 + 2\Delta$

ELSE $x_3 = x_1 - \Delta$.
Evaluar $f(x_3)$.

Paso 4: Determinar $F_{min} = min(f_1, f_2, f_3)$ y X_{min} es el punto x_i que corresponde a F_{min}.

Paso 5: Calcular $\bar{x}$ usando x_1, x_2, x_3.

Paso 6: ¿Es $|F_{min} - f(\bar{x})| \leq TOL1$ AND $|X_{min} - \bar{x}| \leq TOL2$?

Si no se cumple, GOTO paso 7

ELSE el óptimo es el mejor de los 4 puntos.

TERMINAR

Paso 7: Almacenar el mejor punto (X_{min} o $\bar{x}$) y dos puntos que lo rodeen, si esto es posible.

Si no, almacena los 3 mejores puntos.
Re-etiquetarlos deacuerdo a: $x_1 < x_2 < x_3$.
GOTO paso 4

Nota: Para estimar $\bar{x}$ se hace la aproximación del valor a una función cuadrática con interpolación de la siguiente forma:

$$q(x) = a_0 + a_1(x - x_1) + a_2(x - x_1)(x - x_2)$$

donde si $(x_1, f_1), (x_2, f_2)$ y (x_3, f_3) son tres puntos de esta función, pueden estimarse los coeficientes de la siguiente forma:

$$a_0 = f_1, \quad a_1 = \frac{f_2 - f_1}{x_2 - x_1}, \quad a_2 = \frac{1}{x_3 - x_2}\left(\frac{f_3 - f_1}{x_3 - x_1} - a_1\right)$$

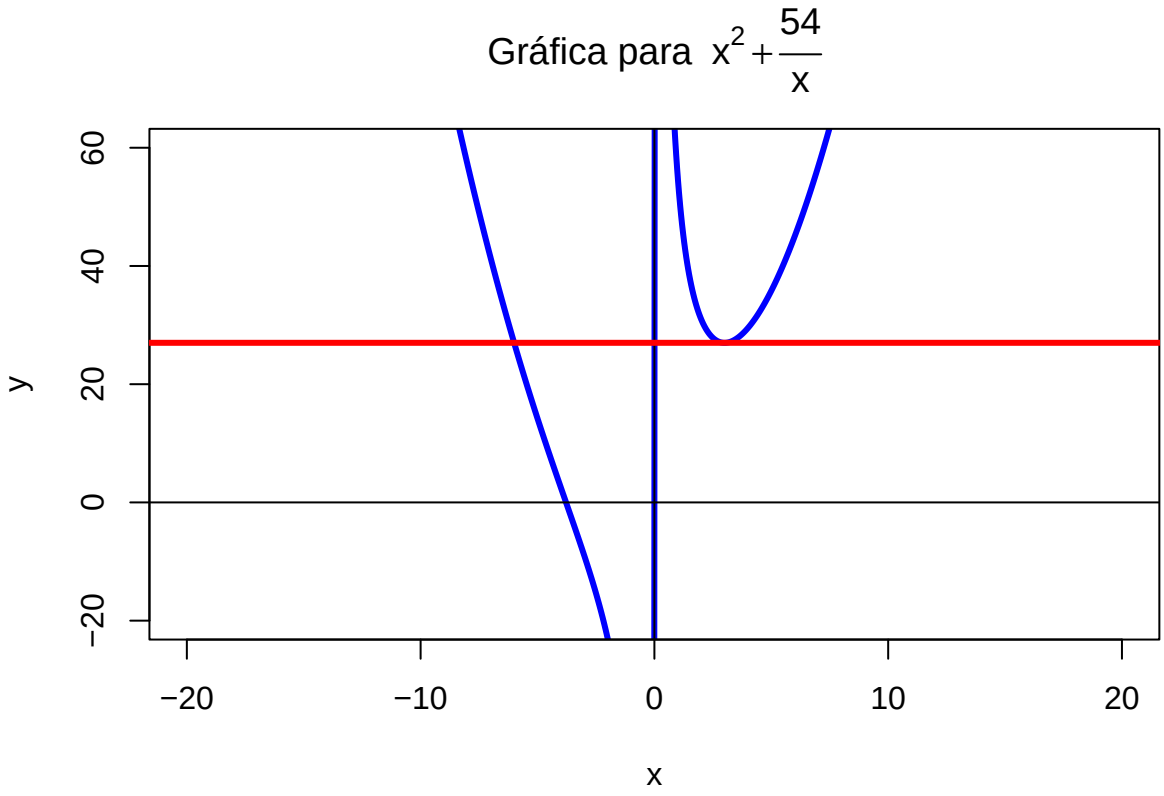
Puede demostrarse que el óptimo x^* de $q(x)$ se encuentra en:

$$x^* = \frac{x1 + x2}{2} - \frac{a_1}{2a_2}$$

Una vez explicado lo que se debe de hacer y visto la estructura del método, se realizo la implementación de dicho algoritmo (Anexo 1) en lenguaje R para ponerlo a prueba contra la función dada en un inicio.

2 Experimentación con los resultados.

Primero, debido a que la función se puede graficar en un espacio de 2 dimensiones, se recurrió a esto para tener un referente visual del comportamiento de la misma:



La función presenta ciertos comportamientos interesantes, por ejemplo, a medida de que se acerca al 0 por la izquierda, dicha función tiende al infinito negativo (si buscara ahí el mínimo, nunca lo encontraría) y si se acerca al 0 por la derecha, el valor de la función tiende a ir hacia el infinito positivo (ahí tampoco es factible buscar).

Dado que la mayoría de los problemas que tienen que ver con cosas del mundo real implican que los valores sean positivos, se optara por buscar valores mayores o iguales a 0 de X tal que permitan llegar al mínimo que se ubica cuando $x = 3$ que devuelve un $f(x) = 27$.

2.1 Resolución de clase

Durante la clase, se dieron los parámetros de inicio para la búsqueda: $x_1 = 1$, $\Delta = 1$, $Tol_1 = Tol_2 = 0.001$, los cuales al ser evaluados en el algoritmo implementado, devuelven los siguientes resultados:

iteración	**Xmin**	**Xestrella**	**Fmin**	**Festrella**	**Tol1**	**Tol2**
1	3	2.700000	27	27.29000	0.290000	0.300000
2	3	2.961538	27	27.00448	0.004476	0.038462
3	3	2.998668	27	27.00001	0.000005	0.001332
4	3	2.999994	27	27.00000	0.000000	0.000006

El algoritmo logra encontrar una buena solución para $x = 3$ (la cual coincide con el óptimo) pero la aproximación que encuentra para $x^* = 2.999994$ es una muy buena aproximación y lo consigue en apenas 4 iteraciones.

2.2 Solución de clase cambiando el valor de $\Delta = 0.1$

Con la finalidad de explorar el comportamiento del método al cambiar el valor de $\Delta = 0.1$ y manteniendo constante $x_1 = 1$ y las tolerancias de $Tol1 = Tol2 = 0.001$ se obtuvo el siguiente resultado:

iteración	**Xmin**	**Xestrella**	**Fmin**	**Festrella**	**Tol1**	**Tol2**
1	1.200000	1.610629	46.44000	36.12140	10.318601	0.410629
2	1.610629	1.881248	36.12140	32.24344	3.877956	0.270619
3	1.881248	2.197944	32.24344	29.39937	2.844070	0.316696
4	2.197944	2.532572	29.39937	27.73612	1.663253	0.334628
5	2.532572	2.768922	27.73612	27.16910	0.567017	0.236351
6	2.768922	2.917099	27.16910	27.02101	0.148095	0.148177
7	2.917099	2.980305	27.02101	27.00117	0.019839	0.063206
8	2.980305	2.997093	27.00117	27.00003	0.001143	0.016788
9	2.997093	2.999783	27.00003	27.00000	0.000025	0.002690
10	2.999783	2.999993	27.00000	27.00000	0.000000	0.000210

Al variar el valor de Δ, el algoritmo logra encontrar una buena solución para $x = 2.999783$ pero la aproximación que encuentra para $x^* = 2.999993$ es mejor aproximación y lo consigue en apenas 10 iteraciones.

2.3 Solución de clase cambiando el valor de $\Delta = 0.5$

Con la finalidad de explorar el comportamiento del método al cambiar el valor de $\Delta = 0.5$ y manteniendo constante $x_1 = 1$ y las tolerancias de $Tol1 = Tol2 = 0.001$ se obtuvo el siguiente resultado:

iteración	**Xmin**	**Xestrella**	**Fmin**	**Festrella**	**Tol1**	**Tol2**
1	2.000000	2.131579	31.00000	29.87696	1.123038	0.131579
2	2.131579	2.517647	29.87696	27.78714	2.089817	0.386068
3	2.517647	2.773373	27.78714	27.16247	0.624672	0.255726
4	2.773373	2.909408	27.16247	27.02513	0.137342	0.136035
5	2.909408	2.979383	27.02513	27.00128	0.023851	0.069974
6	2.979383	2.996894	27.00128	27.00003	0.001252	0.017512
7	2.996894	2.999751	27.00003	27.00000	0.000029	0.002857
8	2.999751	2.999992	27.00000	27.00000	0.000000	0.000241

Al variar el valor de Δ, el algoritmo logra encontrar una buena solución para $x = 2.999751$ pero la aproximación que encuentra para $x^* = 2.999992$ es mejor aproximación y lo consigue en apenas 8 iteraciones.

2.4 Solución de clase cambiando el valor de inicio a $x_1 = 5$

Con la finalidad de explorar el comportamiento del método al cambiar el valor de $x_1 = 5$ y manteniendo constante $\Delta = 1$ y las tolerancias de $Tol1 = Tol2 = 0.001$ se obtuvo el siguiente resultado:

iteración	Xmin	Xestrella	Fmin	Festrella	Tol1	Tol2
1	4.000000	2.327586	29.50000	28.61766	0.882342	1.672414
2	2.327586	3.041667	28.61766	27.00516	1.612497	0.714080
3	3.041667	3.073044	27.00516	27.01575	0.010592	0.031378
4	3.041667	3.008883	27.00516	27.00024	0.004925	0.032784
5	3.008883	2.999554	27.00024	27.00000	0.000236	0.009329
6	2.999554	2.999962	27.00000	27.00000	0.000001	0.000408

Al variar el valor de x_1, el algoritmo logra encontrar una buena solución para $x = 2.999554$ pero la aproximación que encuentra para $x^* = 2.999962$ es mejor aproximación y lo consigue en apenas 6 iteraciones.

2.5 Solución de clase cambiando el valor de inicio a $x_1 = 0.1$

Con la finalidad de explorar el comportamiento del método al cambiar el valor de $x_1 = 0.1$ y manteniendo constante $\Delta = 1$ y las tolerancias de $Tol1 = Tol2 = 0.001$ se obtuvo el siguiente resultado:

iteración	**Xmin**	**Xestrella**	**Fmin**	**Festrella**	**Tol1**	**Tol2**
1	2.100000	1.642972	30.12429	35.56663	5.442343	0.457028
2	2.100000	2.262473	30.12429	28.98647	1.137815	0.162473
3	2.262473	2.623480	28.98647	27.46599	1.520477	0.361007
4	2.623480	2.837910	27.46599	27.08182	0.384174	0.214430
5	2.837910	2.943688	27.08182	27.00963	0.072186	0.105778
6	2.943688	2.989305	27.00963	27.00034	0.009290	0.045617
7	2.989305	2.998697	27.00034	27.00001	0.000339	0.009392
8	2.998697	2.999923	27.00001	27.00000	0.000005	0.001226
9	2.999923	2.999998	27.00000	27.00000	0.000000	0.000076

Al variar el valor de x_1, el algoritmo logra encontrar una buena solución para $x = 2.999923$ pero la aproximación que encuentra para $x^* = 2.999998$ es mejor aproximación y lo consigue en apenas 9 iteraciones.

2.6 Solución de clase cambiando el valor de $\Delta = -0.5$

Con la finalidad de explorar el comportamiento del método al cambiar el valor de $\Delta = -0.5$ y manteniendo constante $x_1 = 1$ y las tolerancias de $Tol1 = Tol2 = 0.001$ se obtuvo el siguiente resultado:

iteración	**Xmin**	**Xestrella**	**Fmin**	**Festrella**	**Tol1**	**Tol2**
1	1.500000	1.479452	38.25000	38.68878	0.438778	0.020548
2	1.500000	1.911184	38.25000	31.90736	6.342642	0.411184
3	1.911184	2.267245	31.90736	28.95785	2.949504	0.356060
4	2.267245	2.534188	28.95785	27.73071	1.227146	0.266944
5	2.534188	2.789135	27.73071	27.14012	0.590594	0.254947
6	2.789135	2.926732	27.14012	27.01637	0.123741	0.137596
7	2.926732	2.982478	27.01637	27.00092	0.015449	0.055747
8	2.982478	2.997664	27.00092	27.00002	0.000908	0.015185
9	2.997664	2.999832	27.00002	27.00000	0.000016	0.002168
10	2.999832	2.999995	27.00000	27.00000	0.000000	0.000163

Al variar el valor de Δ, el algoritmo logra encontrar una buena solución para $x = 2.999832$ pero la aproximación que encuentra para $x^* = 2.999995$ es mejor aproximación y lo consigue en apenas 10 iteraciones.

2.7 Solución de clase cambiando el valor inicial de $x_1 = -1$ y el valor $\Delta = 0.5$

Con la finalidad de explorar el comportamiento del método al cambiar el valor de $\Delta = 0.5$ con un punto de inicio en los valores negativos para $x_1 = -1$ y las tolerancias de $Tol1 = Tol2 = 0.001$ se obtuvo el siguiente resultado:

```
## [1] "No se logro encontrar un buen optimo, este fue el recorrido en 50 iters:"
```

iteración	Xmin	Xestrella	Fmin	Festrella	Tol1	Tol2
1	-0.5	-0.750000	-107.75	-71.43750	36.31250	0.250000
2	-0.5	-1.132867	-107.75	-46.38328	61.36672	0.632867
3	-0.5	-1.132867	-107.75	-46.38328	61.36672	0.632867
4	-0.5	-1.132867	-107.75	-46.38328	61.36672	0.632867
5	-0.5	-1.132867	-107.75	-46.38328	61.36672	0.632867
6	-0.5	-1.132867	-107.75	-46.38328	61.36672	0.632867
7	-0.5	-1.132867	-107.75	-46.38328	61.36672	0.632867
8	-0.5	-1.132867	-107.75	-46.38328	61.36672	0.632867
9	-0.5	-1.132867	-107.75	-46.38328	61.36672	0.632867
10	-0.5	-1.132867	-107.75	-46.38328	61.36672	0.632867

Al iniciar el algoritmo en valores negativos, puesto que la función tiene al infinito negativo cuando se acerca a 0, esta de alguna forma se atora y ya no avanza más de forma que no logra encontrar el mínimo que si encuentran las otras corridas en 50 iteraciones. Los resultados solo muestran las primeras 10 iteraciones, pero el algoritmo converge a un resultado donde ya no se mueven los valores prácticamente para nada.

3 Conclusiones y comentarios finales

Tras implementar el método y hacer varias pruebas, el algoritmo resulta que encuentra buenos resultados en los positivos, donde se alberga el mínimo óptimo, no importa si se cambia el tamaño del Δ, o la inicialización de x_1 siempre y cuando se encuentre en los positivos. Por otro lado, al inicializar el x_1 en valores negativos, donde se encuentra la asintota al infinito negativo, empieza a comportarse extraño, porque los valores de x no se mueven por más que se hagan iteraciones.

4 Anexo 1: Código fuente del algotimo de estimaciones cuadráticas sucesivas

```
## #### Método de las Estimaciones Cuadráticas Sucesivas ####
## #fx = función que se desea minímizar
## #a = x1 = Punto de inicio
## #Dx = Tamaño del paso
## #T1 y T2 = Tolerancia para la convergencia
##
## MSC = function(fx,a,Dx,T1,T2){
##   #### Paso 1 ####
##   # Contador para llevar el control de esto
##   contador = 0
##   # Calcula el punto x2
##   x1 = a
##   x2 = x1 + Dx
##
##   #### Paso 2 ####
##   # Calcula las evaluaciones de fx1 y fx2
##   fx1 = fx(x1)
##   fx2 = fx(x2)
##
##   #### Paso 3 ####
##   # Revisar hacia donde se esta moviendo para calcular el punto x3
##   if(fx1>fx2){
##     #### Si se cumple encontes:
##     x3 = x1 + 2*Dx
##   }else{
##     # SI no se cumple
##     x3 = x1 - Dx
##   }
##
##   # Objeto para guardar los resultados
##   resultados = data.frame(iteración = 0, Xmin = 0,Xestrella = 0 ,Fmin = 0,
##                           Festrella = 0, Tol1 = 0, Tol2 = 0)
##   ### Condición de paro para iterar del punto 4 al 7##
##   while(contador<=50){
##     #### Paso 4 ####
##     contador = contador + 1
##     fx1 = fx(x1)
##     fx2 = fx(x2)
##     fx3 = fx(x3)
##     # Determinar Fmin = min(f1,f2,f3) y Xmin
##     X = c(x1,x2,x3)
##     Fi = c(fx1,fx2,fx3)
##     ### Determiniar Fmin
##     Fmin = min(Fi)
##     ### DETERMINAR Xmin
##     Xmin = X[which.min(Fi)]
##
##     #### Paso 5 ####
##     # Calcular X*
##     a0 = fx1
##     a1 = (fx2-fx1)/(x2-x1)
```

```
##     a2 = (1/(x3-x2)) * ((fx3-fx1)/(x3-x1) - a1 )
##     ## Calculo del X*
##     xe = (x1+x2)/2 - (a1/(2*a2))
##     fxe = fx(xe)
##
##     ### Agregarlo al selector del optimo
##     resultados[contador,]  = c(contador,Xmin,Xestrella = xe ,Fmin,fxe,
##                                Tol1 = abs(Fmin-(fxe)), Tol2 = abs(Xmin-(xe)))
##
##     #### Paso 6 ####
##     if((abs(Fmin-(fxe))<= T1) & (abs(Xmin-(xe))<= T2)){
##       # Devolver el optimo
##       return(resultados)
##     }
##
##     #### Paso 7 ####
##     # Asignar los nuevos puntos de x
##     X = c(x1,x2,x3,xe)
##     Fi = c(fx1,fx2,fx3,fxe)
##     ### Determiniar Fmin
##     Fmin = min(Fi)
##     ### DETERMINAR Xmin
##     Xmin = X[which.min(Fi)]
##     ### Re ordenarlos
##     XS = sort(X[-which.max(Fi)])
##
##     x1 = XS[1]
##     x2 = XS[2]
##     x3 = XS[3]
##   }
##   #  Devolver en caso de no lograr la convergencia
##   print("No se logro encontrar un buen optimo, este fue el recorrido en 50 iters:")
##   return(resultados)
## }
```

5 Anexo 2: Código para hacer las evaluaciones del método.

```
## ##Cargar el script con el método de Estimaciones Cuadráticas Sucesivas ####
## source("Tarea 6. Estimaciones cuadráticas sucesivas MAIN.R")
## # Función a evaluar #
## fx = function(x){(x^2)+(54/x)}
## #### Evaluación para el ejercicio de clase ####
## # x1 = 1
## # Delta = 1
## # Tol1=Tol2= 0.001
## res1 = round(MSC(fx,1,1,0.001,0.001),6)
## res1
## #### Evaluación para el ejercicio de clase disminuyendo cambiando el delta= 0.1 ####
## # x1 = 1
## # Delta = 0.1
## # Tol1=Tol2= 0.00001
## res2 = round(MSC(fx,1,0.1,0.001,0.001),6)
## res2
## #### Evaluación para el ejercicio de clase cambiando el delta= 0.5 ####
## # x1 = 1
## # Delta = 0.5
## # Tol1=Tol2= 0.001
## res3 = round(MSC(fx,1,0.5,0.001,0.001),6)
## res3
## #### Evaluación para el ejercicio de clase cambiando el punto de inicio en 5 ####
## # x1 = 5
## # Delta = 1
## # Tol1=Tol2= 0.001
## res4 = round(MSC(fx,5,1,0.001,0.001),6)
## res4
## #### Evaluación para el ejercicio de clase cambiando el punto de inicio en 0.1 ####
## # x1 = 0.1
## # Delta = 1
## # Tol1=Tol2= 0.001
## res5 = round(MSC(fx,0.1,1,0.001,0.001),6)
## res5
## #### Evaluación para el ejercicio de clase cambiando el delta a uno negativo ####
## # x1 = 1
## # Delta = -0.5
## # Tol1=Tol2= 0.001
## res6 = round(MSC(fx,1,-0.5,0.001,0.001),6)
## res6
## #### Evaluación para el ejercicio de clase cambiando el punto de inicio en los negativos ####
## # x1 = -1
## # Delta = 0.5
## # Tol1=Tol2= 0.001
## res7 = round(MSC(fx,-1,0.5,0.001,0.001),6)
## res7
```