Optimización inteligente

Tarea 2. Eliminación de regiones

Ángel García Báez

2024-09-22

Breve introducción

Se planteo un ejercicio en clase donde se pedía encontrar el intervalo (espacio de búsqueda) donde sería factible encontrar el mínimo global de la función $f(x) = x^2 + \frac{54}{x}$ haciendo uso del método conocido como ***eliminación de regiones***.

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

Método de eliminación de regiones

Algoritmo

Paso 1: Elegir un límite inferior a y un limite superior b

Definir la tolerancia ε
$x_m = (a + b)/2$
$Lo = L = b - a$
Calcular $f(x_m)$

Paso 2: $x_1 = a + (L/4); x_2 = b - (L/4)$

Calcular $f(x_1)$ y $f(x_2)$

Paso 3: IF $f(x_1) < f(x_m)$ THEN

$b = x_m$
$x_m = x_1$
GO TO PASO 5

Paso 4: IF $f(x_1) < F(x_2)$ THEN

$a = x_m$
$x_m = x_2$
GO TO PASO 5

ELSE

$a = x_1$
$b = x_2$
GO TO PASO 5

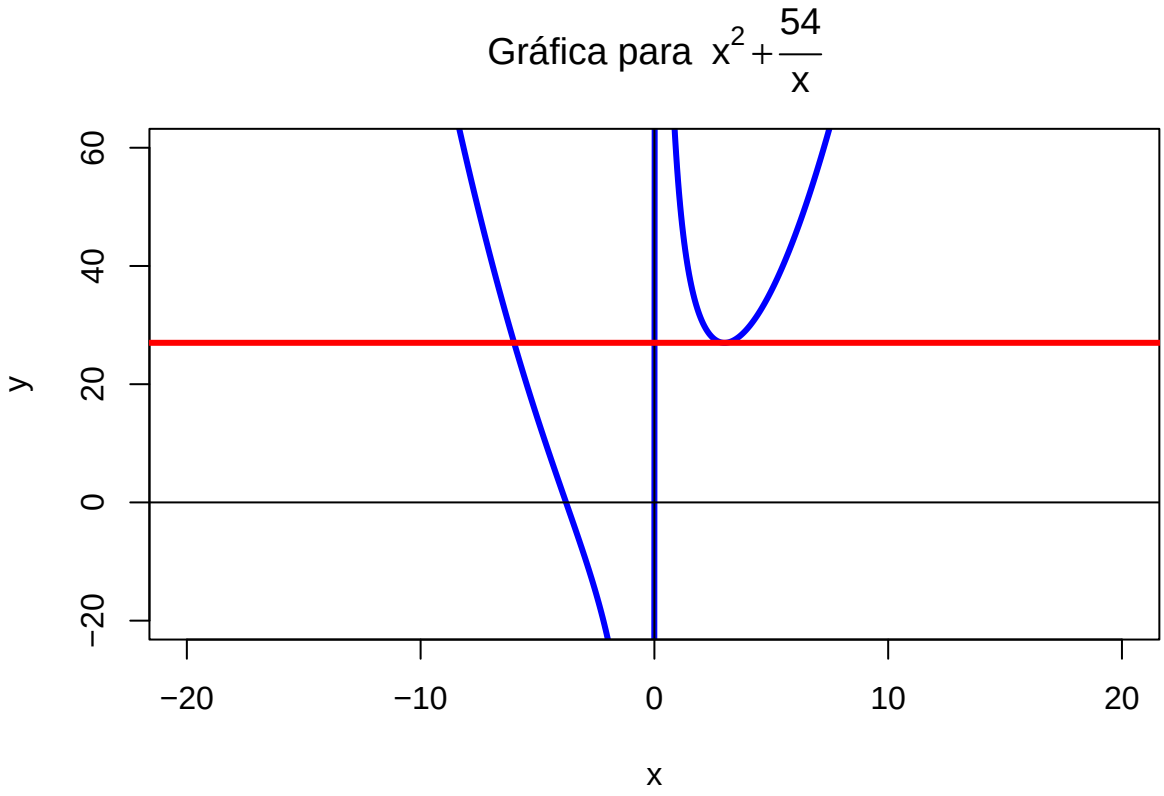
PASO 5: $L = b - a$

IF $|L| < \varepsilon$ TERMINAR
ELSE GO TO PASO 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.

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 $f(x) = 27$.

Primer acercamiento

Durante la clase, se dieron los parámetros de inicio para la búsqueda: $a = 0$, $b = 5$ y una tolerancia $\varepsilon = 0.001$, los cuales al ser evaluados en el algoritmo implementado, devuelven los siguientes resultados:

Iter	A	B	f(x1)	f(x2)	tolerancia	evaluaciones
1	1.250000	3.750000	44.76250	28.46250	2.500000	3
2	2.500000	3.750000	32.31562	27.04563	1.250000	6
3	2.812500	3.437500	27.11016	27.52550	0.625000	9
4	2.812500	3.125000	27.00295	27.22374	0.312500	12
5	2.890625	3.046875	27.03679	27.00652	0.156250	15
6	2.968750	3.046875	27.01507	27.00018	0.078125	18
7	2.988281	3.027344	27.00041	27.00223	0.039062	21
8	2.988281	3.007812	27.00001	27.00092	0.019531	24
9	2.993164	3.002930	27.00014	27.00003	0.009766	27
10	2.998047	3.002930	27.00006	27.00000	0.004883	30
11	2.999268	3.001709	27.00000	27.00001	0.002441	33
12	2.999268	3.000488	27.00000	27.00000	0.001221	36
13	2.999573	3.000183	27.00000	27.00000	0.000610	39

El algoritmo logra encontrar una convergencia que satisface el criterio de tolerancia impuesto ($\varepsilon = 0.001$) al cabo de 13 iteraciones y después de realizar unas 39 evaluaciones de la función. El intervalo de búsqueda donde es factible encontrar el valor que minimiza la función es entre:

$$[2.999573, 3.000183]$$

Este intervalo encontrado bajo los criterios dados en clase, logra resolver perfectamente el problema sin mayor complicación.

Exploración y uso del método

Con la finalidad de explorar el comportamiento del método, se propone volver a correr el algoritmo para el mismo problema pero esta vez en un intervalo determinado por $a = 0$ y $b = 10$ para comprobar a que resultado llega si se dobla el intervalo de búsqueda.

A continuación se muestra el resultado de dicha evaluación:

Iter	A	B	f(x1)	f(x2)	tolerancia	evaluaciones
1	0.000000	5.000000	27.85000	63.45000	5.000000	3
2	1.250000	3.750000	44.76250	28.46250	2.500000	6
3	2.500000	3.750000	32.31562	27.04563	1.250000	9
4	2.812500	3.437500	27.11016	27.52550	0.625000	12
5	2.812500	3.125000	27.00295	27.22374	0.312500	15
6	2.890625	3.046875	27.03679	27.00652	0.156250	18
7	2.968750	3.046875	27.01507	27.00018	0.078125	21
8	2.988281	3.027344	27.00041	27.00223	0.039062	24
9	2.988281	3.007812	27.00001	27.00092	0.019531	27
10	2.993164	3.002930	27.00014	27.00003	0.009766	30
11	2.998047	3.002930	27.00006	27.00000	0.004883	33
12	2.999268	3.001709	27.00000	27.00001	0.002441	36
13	2.999268	3.000488	27.00000	27.00000	0.001221	39
14	2.999573	3.000183	27.00000	27.00000	0.000610	42

Los resultados para cuando se aumenta el rango de $[0, 5]$ a un rango de $[0, 10]$ muestran que apenas y se necesitaría 1 iteración más para poder encontrar el mismo intervalo que en la corrida pasada.

Como lo indica su nombre, el método lo que hizo fue cortar a la mitad (misma magnitud en la que se aumento el intervalo de 5 a 10).

Propuesta de un intervalo de 2 a 3

A continuación se explora que pasaría al acortar el intervalo de $[0, 5]$ a uno más pequeño como puede ser el intervalo $[2, 3]$ que sin que el algoritmo lo sepa, contiene el valor que minimiza la función en la parte superior del intervalo. Los resultados de evaluar el algoritmo así son:

Iter	**A**	**B**	**f(x1)**	**f(x2)**	**tolerancia**	**evaluaciones**
1	2.500000	3	29.06250	27.19886	0.500000	3
2	2.750000	3	27.46205	27.04823	0.250000	6
3	2.875000	3	27.11016	27.01188	0.125000	9
4	2.937500	3	27.02693	27.00295	0.062500	12
5	2.968750	3	27.00666	27.00073	0.031250	15
6	2.984375	3	27.00166	27.00018	0.015625	18
7	2.992188	3	27.00041	27.00005	0.007812	21
8	2.996094	3	27.00010	27.00001	0.003906	24
9	2.998047	3	27.00003	27.00000	0.001953	27
10	2.999023	3	27.00001	27.00000	0.000977	30

Contrario a lo que pensaba en un primero momento, que el algoritmo llegaría rápidamente en menos de 3 iteraciones a darse cuenta de que su intervalo ya contiene al valor que minimiza la función, el algoritmo se toma hasta 10 iteraciones para llegar a dicha conclusión, resaltando el hecho de que en todas ellas el valor superior del intervalo $B = 3$ se mantiene constante, lo que se va reduciendo por mitad es la parte inferior del intervalo hasta lograr satisfacer el criterio de tolerancia ($\epsilon < 0.001$)

Evaluando en valores negativos

Como experimento adicional, se plantea la idea de ver que ocurre si se explora un intervalo en los números negativos, como lo puede ser para $a = -5$ y $b = 0$:

Iter	A	B	f(x1)	f(x2)	tolerancia	evaluaciones
1	-2.500000	0	-0.33750	-41.63750	2.500000	3
2	-1.250000	0	-25.28438	-86.00938	1.250000	6
3	-0.625000	0	-56.72109	-172.70234	0.625000	9
4	-0.312500	0	-114.98027	-345.57559	0.312500	12
5	-0.156250	0	-230.34507	-691.19390	0.156250	15
6	-0.078125	0	-460.78627	-1,382.39847	0.078125	18
7	-0.039062	0	-921.59657	-2,764.79962	0.039062	21
8	-0.019531	0	-1,843.19914	-5,529.59991	0.019531	24
9	-0.009766	0	-3,686.39979	-11,059.19998	0.009766	27
10	-0.004883	0	-7,372.79995	-22,118.39999	0.004883	30
11	-0.002441	0	-14,745.59999	-44,236.80000	0.002441	33
12	-0.001221	0	-29,491.20000	-88,473.60000	0.001221	36
13	-0.000610	0	-58,982.40000	-176,947.20000	0.000610	39

Se sabia de antemano que la función presenta una asintota en 0, cuando se aproxima por la izquierda tiende a valer $-\infty$, para este caso, con unos parámetros de entrada igual que para el caso visto en clase pero negativos (-5 y 0), se logra una convergencia que satisface el criterio de tolerancia ($\epsilon = 0.001$) en apenas 13 iteraciones, con un intervalo:

$$[-0.000610, 0]$$

El algoritmo encontró que ese intervalo es suficiente espacio de búsqueda para satisfacer la tolerancia sin la necesidad de irse a buscar más profundo hacia el $-\infty$.

Conclusiones y comentarios finales

El implementar y observar el funcionamiento de este método fue enriquecedor para ampliar el panorama sobre los métodos para buscar intervalos que garanticen la existencia de un optimo.

Sobre el método, resulta impresionante como ir reduciendo por mitad en cada paso que da resulta una estrategia efectiva que cumple su trabajo en pocas iteraciones, debido a su naturaleza implícita en el nombre.

Demostró ser capaz de encontrar un intervalo lo suficientemente fino para buscar en pocas iteraciones tanto para los números positivos como para los números negativos.

Su criterio de tolerancia que de forma interna va disminuyendo por mitad en cada iteración es lo que le permite converger a un intervalo fino sin la necesidad de re-evaluar los parámetros de forma manual hasta dar con una región factible de búsqueda.

Anexo 1: Codigo fuente del algotimo de eliminación por regiones.

```
## #### Función del método de Eliminación de Regiones ####
##
## ### Parámetros de entrada ###
## ## a = LImite inferior del intervalo
## ## b = Limite superior del intervalo
## ## epsilon = Error permitido, fijado a 0.001
## ## f(x) = Función
## ElimReg = function(a,b,epsilon,FX){
##   ### Paso 1 ###
##   xm = (a+b)/2 # Iniciar el valor medio x_m
##   L = b-a # Definir la amplitud del intervalo que servira como tolerancia
##   contador = 0 # Contador de iteraciones
##   evaluador = 0 # Contador de evaluaciones de la función
##   # Objeto donde se guardan los datos
##   resultados = data.frame(Iter = 0, A = 0,B = 0,fx1 = 0, fx2 = 0,tolerancia = 0,
##                           evaluaciones = 0)
##   # Ciclo WHILE para los pasos del 2 al 5 #
##   while(L>epsilon){
##     #### Paso 2 ####
##     x1 = a + (L/4); x2 = b - (L/4)
##     fxm = FX(xm)
##     evaluador = evaluador +1 # Se hizo 1 evaluación de la función
##     # Calcular f(x1) y f(x2)
##     fx1 = FX(x1);fx2 = FX(x2)
##     evaluador = evaluador +2 # Se hicieron 2 evaluaciones de la función
##     #### Paso 3 ####
##     if(fx1<fxm){
##       # Cambiar los limites
##       b = xm; xm = x1
##     }else{
##       #### Paso 4 ####
##       if(fx2<fxm){
##         # Actualización de parametros
##         a = xm; xm = x2
##       }else{
##         #### Paso 4.1 ####
##         a = x1; b = x2
##       }}
##     #### PASO 5 ####
##     L = abs(b-a) # Actualizar si ya se cumple la tolerancia
##     contador = contador +1
##     # Agregar los resultados
##     resultados[contador,] = c(contador,a,b,fx1,fx2,L,evaluador)
##   }
##   # Reportar la tabla con los resultados
##   names(resultados)[4:5] = c("f(x1)","f(x2)")
##   return(resultados)
## }
```