Optimización inteligente

Tarea 7. Método de Newthon-Rhapson con derivadas aproximadas

Ángel García Báez

2024-10-21

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 ***Newtho-Rhapson***.

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: Proporcionar el punto inicial x_1 y elegir una tolerancia ϵ.

k = 1 Calcular $f'(x_1)$

Paso 2: Calcular $f''(x_1)$

Paso 3: Calcular $x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$

Calcular $f'(x_1)$

Paso 4: IF $|f'(x_{k+1})| \leq \epsilon$ THEN TERMINAR

ELSE $k = k + 1$. GOTO Paso 2

Nota:

Para estimar las derivadas cuando por alguna razón no se conoce su forma analítica, se pueden hacer de la siguiente forma:

$$f'(x_k) = \frac{f(x_k + \Delta x) - f(x_k - \Delta x)}{2\Delta x}$$

$$f''(x_k) = \frac{f(x_k + \Delta x) - 2f(x_k) + f(x_k - \Delta x)}{(\Delta x)^2}$$

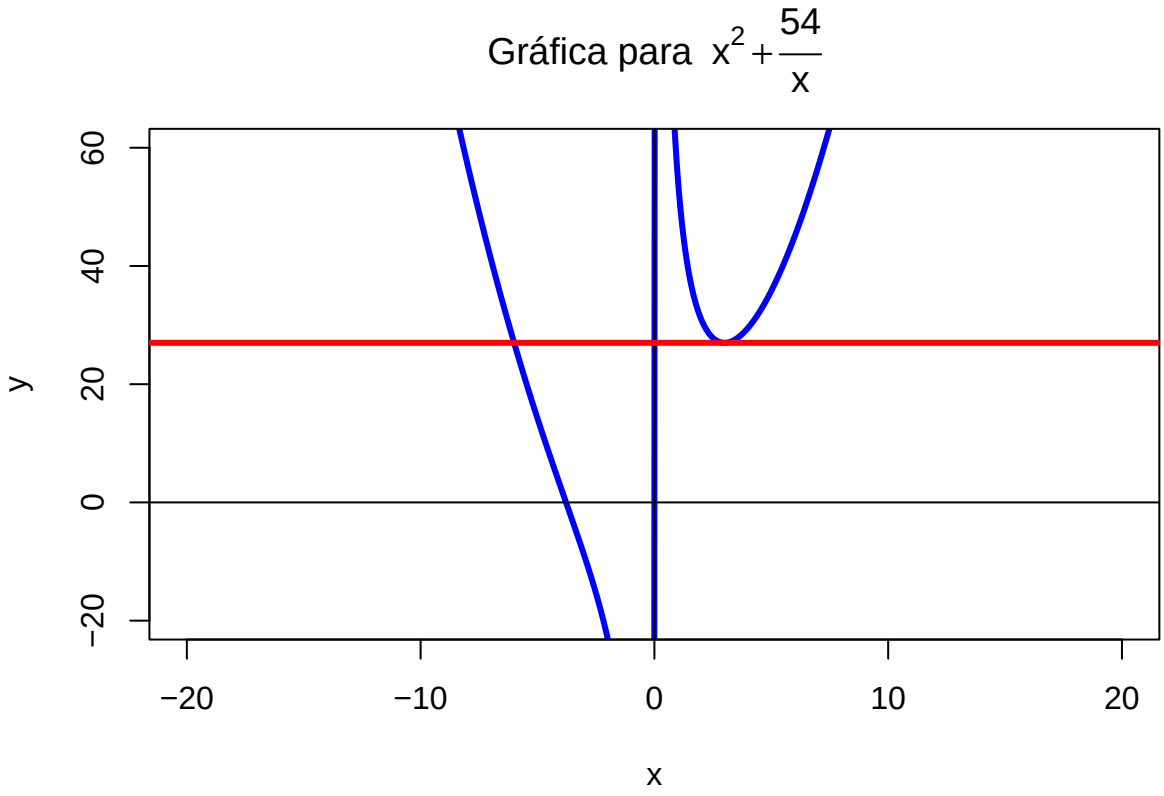
donde Δx es un valor pequeño.

Nótese que se requiere evaluar la función en 3 puntos: x_k, $x_k + \Delta x$ y $x_k - \Delta x$.

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: $x = 1$, $\Delta = 0.001$, $Tol_1 = 0.001$, los cuales al ser evaluados en el algoritmo implementado, devuelven los siguientes resultados:

Iter	**X**	**derivada1**	**derivada2**	**delta**	**Evals**
1	1.000000	-52.0000540000538	110.000108	0.00100000	3
2	1.472727	-21.9541537994128	35.814282	0.01472727	6
3	2.085727	-8.2428431706144	13.904047	0.02085727	9
4	2.678565	-2.1700591727884	7.620315	0.02678565	12
5	2.963338	-0.2233205332178	6.150722	0.02963338	15
6	2.999646	-0.0027251739639	6.001817	0.02999646	18
7	3.000100	-0.0000004124904	6.000000	0.03000100	21

El algoritmo encuentra en 7 iteraciones y 21 evaluaciones 1 valor de X que cumplen con la restricción de tolerancia, dicho valor es $x = 3.0001$, dicho valor es una solución muy buena considerando el hecho de que el mínimo es 3 cerrado.

2.2 Solución de clase cambiando el Δ para que sea fijo.

Con la finalidad de explorar el comportamiento del método al cambiar el Δ para que sea constante y mantiene los demás parámetros de inicio constantes, se obtuvo el siguiente resultado:

Iter	**X**	**derivada1**	**derivada2**	**delta**	**Evals**
1	1.000000	-52.0000540000538	110.000108	0.001	3
2	1.472727	-21.9516753175846	35.810917	0.001	6
3	2.085716	-8.2417657989335	13.903058	0.001	9
4	2.678518	-2.1696649378189	7.620049	0.001	12
5	2.963249	-0.2232518082224	6.150680	0.001	15
6	2.999546	-0.0027235833890	6.001816	0.001	18
7	3.000000	-0.0000004118235	6.000000	0.001	21

Al mantener constante el valor de Δ, el algoritmo nuevamente encuentra una buena solución en apenas 7 iteraciones, pero el valor de Delta al no variar, logra encontrar una solución aun más precisa respecto a la primera solución encontrada en la evaluacion anterior.

2.3 Solución de clase cambiando el valor de inicio en 50$

Para explorar como se comporta el algoritmo si se le inicia en un valor por mucho muy lejano de donde se encuentra el optimo, se decidio poner el punto de inicio en 50, manteniendo todos los demas valores constantes y haciendo uso del delta dinamico:

Iter	X	derivada1	derivada2	delta	Evals
1	50.00000000	99.97839999983	2.000864	0.0010000000	3
2	0.03239206	-51,470.63268489735	3,177,984.343092	0.0003239206	6
3	0.04858806	-22,875.79715861481	941,628.175732	0.0004858806	9
4	0.07288193	-10,166.96157433577	279,004.126147	0.0007288193	12
5	0.10932212	-4,518.56051247896	82,671.074427	0.0010932212	15
6	0.16397921	-2,008.11550984639	24,498.318392	0.0016397921	18
7	0.24594873	-892.29540923101	7,261.946335	0.0024594873	21
8	0.36882150	-396.27536674948	2,154.873478	0.0036882150	24
9	0.55271878	-175.67278261450	641.667868	0.0055271878	27
10	0.82649403	-77.40718343686	193.314562	0.0082649403	30
11	1.22691489	-33.42252564696	60.482224	0.0122691489	33
12	1.77951570	-13.49525472116	21.167334	0.0177951570	36
13	2.41706668	-4.40986648359	9.648941	0.0241706668	39
14	2.87409783	-0.78964148034	6.549488	0.0287409783	42
15	2.99466320	-0.03268009350	6.021826	0.0299466320	45
16	3.00009014	-0.00005918729	6.000039	0.0300009014	48

Al variar el valor de de inicio en $x = 50$ y manteniendo constantes los demás parámetros, el algoritmo se tarda 16 iteraciones pero consigue encontrar una solución factible que cumple con el criterio de tolerancia. La solución a la que llega es de $x = 3.00009$.

2.4 Solución de clase cambiando el valor de inicio a $x_1 = -10$

Para explorar como se comporta el algoritmo si se le inicia en un valor por mucho muy lejano de donde se encuentra el optimo, se decidió poner el punto de inicio en -10, manteniendo todos los demas valores constantes y haciendo uso del delta dinamico:

Iter	**X**	**derivada1**	**derivada2**	**delta**	**Evals**
1	-10.0000000	-20.5400000054	1.892000	0.001000000	3
2	0.8562367	-71.9505445101	174.062275	0.008562367	6
3	1.2695976	-30.9654464867	54.779937	0.012695976	9
4	1.8348675	-12.3711299862	19.484494	0.018348675	12
5	2.4697893	-3.9139703505	9.169477	0.024697893	15
6	2.8966370	-0.6432152605	6.444112	0.028966370	18
7	2.9964514	-0.0219182729	6.014629	0.029964514	21
8	3.0000956	-0.0000266453	6.000018	0.030000956	24

Al iniciar el algoritmo en -10, resulta sorprende como logra llegar en apenas 8 iteraciones a una solución muy bien aproximada de $x = 3.00009$ considerando el hecho de que presenta una asintota al infinito negativo alrededor del 0, es muy impresionante como logra brincarse esta limitante.

3 Conclusiones y comentarios finales

Después de programar el algoritmo con sus variantes para que sea dinámico y tenga forma de calcular las derivadas de forma numérica, se puso a prueba bajo distintos escenarios al variar su punto de inicio, desde valores muy cercanos al optimo conocido, así como valores que se alejan por mucho de este.

Los resultados obtenidos muestran que es muy importante la correcta inicialización del algoritmo para encontrar resultados adecuados y en una pequeña cantidad de iteraciones, pues como se pudo observar, al iniciarse en un valor muy alejado de donde se encuentra el optimo, llega a tardarse casi 20 iteraciones, mientras que al iniciarlo cerca de dicho valor, tarda apenas 7 u 8 iteraciones en lograrlo. Lo más sorprendente fue observar que dicho algoritmo podia sortear el problema de la asintota al ser iniciado en valores negativos.

4 Anexo 1: Código fuente del algotimo de Newthon-Rhapson

```
## #### Método de Newthon-Rhapson ####
## # fx = función que se desea minímizar
## # x = Punto de inicio
## # delta = Tamaño del paso
## # tolerancia = Tolerancia para la convergencia
## # fijo = Valor para decir si el delta es dinamico o no
## NS = function(fx,x,delta,tolerancia,fijo){
##   # Definir algunas funciones auxiliares como la de las derivadas
##   fp = function(fx,x,delta){
##     # Crear los valores necesarios que se repiten
##     a1 = fx(x + delta)
##     a2 = fx(x - delta)
##     ## Evaluar la primera y segunda derivada
##     fp1 = (a1-a2)/(2*delta)
##     fp2 = (a1-2*fx(x)+ a2)/(delta^2)
##     ## Regresar los valores
##     resu = c(fp1,fp2)
##     return(resu)
##   }
##   # Defino un valaor provicional que se actualizara solo en el while
##   fp1 = 10
##   k = 1
##   evalu = 0
##   resultados = data.frame(Iter = 0,X = 0,derivada1 = 0,
##                           derivada2 = 0,delta = 0,Evals =0)
##   # Paso 4 que queda implicito como criterio de paro
##   while(abs(fp1)>tolerancia){
##     # Paso 2
##     # Calcular las primeras y segundas derivadas numericamente
##     # Calcular las derivadas numericas
##     fpe = fp(fx,x,delta)
##     evalu = evalu + 3
##     # Asignar la primera derivada
##     fp1 = fpe[1]
##     # Asignar la segunda derivada
##     fp2 = fpe[2]
##     # Asignar los resultados al objeto
##     resultados[k,] = c(Iter = k,X = x,derivada1 = fp1,
##                        derivada2 = fp2,delta,evalu)
##     # Paso 3, actualizar el valor de X
##     # Calcular la nueva X
##     x = x - (fp1/fp2)
##     # Paso 4, Actualizar el valor de K y volver al paso 2
##     if(fijo==F){
##       # Actualizar el valor de delta dinamico
##       delta = ifelse(abs(x)>0.01,0.01*abs(x),0.01)
##     }
##     k = k+1
##   }
##   return(resultados)
## }
```

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

```
## # Librería para hacer las tablas bonitas
## library(flextable)
## # Llamo el código
## source("Tarea 7. Newthon Rhapson MAIN.R")
##
## # Declaro la función que se desea minimizar
## fx = function(x){(x^2) +  (54/x)}
##
## #### Evaluación para el ejercicio de clase ####
## # x = 1
## # Delta = 0.001
## # Tol= 0.001
## # Fijo = FALSE
## res1 = NS(fx,1,0.001,0.001,F)
##
## # Reportar los resultados bonitos
## autofit(align(theme_box(flextable(res1)), align = "center", part = "all"))
##
## #### Evaluación para el ejercicio de clase ####
## # x = 1
## # Delta = 0.001
## # Tol= 0.001
## # Fijo = TRUE
## res2 = NS(fx,1,0.001,0.001,T)
## # Reportar los resultados bonitos
## autofit(align(theme_box(flextable(res2)), align = "center", part = "all"))
##
## #### Evaluación para el ejercicio de clase ####
## # x = 50
## # Delta = 0.001
## # Tol= 0.001
## # Fijo = FALSE
## res3 = NS(fx,50,0.001,0.001,F)
## # Reportar los resultados bonitos
## autofit(align(theme_box(flextable(res3)), align = "center", part = "all"))
##
## #### Evaluación para el ejercicio de clase ####
## # x = -10
## # Delta = 0.001
## # Tol= 0.001
## # Fijo = FALSE
## res4 = NS(fx,-10,0.001,0.001,F)
## # Reportar los resultados bonitos
## autofit(align(theme_box(flextable(res4)), align = "center", part = "all"))
```