Optimización inteligente

Tarea 12. Método del descenso empinado (Cauchy)

Ángel García Báez

2024-02-12

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 $f(x_1, x_2) = (x_1^2 + x_2 - 11) + (x_1 + x_2^2 - 7)$ haciendo uso de el método de busqued del descenso empinado.

A continuación se muestra el pseudocodigo para implementar dicho método:

1.1 Método del descenso empinado.

Algoritmo

Paso 1: Elegir un punto inicial $X^{(0)}$ y dos parámetros de terminación ε_1 y ε_2. Hacer $k = 0$

Paso 2: Calcular $\nabla f(X^{(k)})$

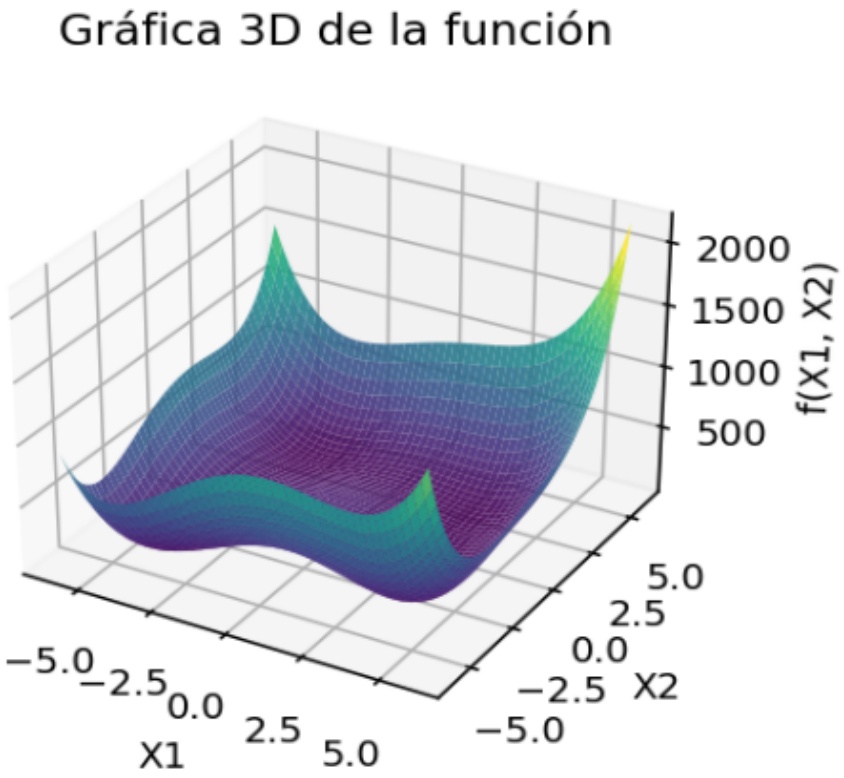
Paso 3: **IF** $\|\nabla f(X^{(k)})\| \leq \varepsilon_1$ **THEN** Terminar **ELSE** ir al Paso 4.

Paso 4: Efectuar una búsqueda unidireccional para encontrar $\lambda^{(k)}$ y calcular: $X^{(k+1)} = X^{(k)} - \lambda^{(k)} \nabla f(X^{(k)})$ tal que $f(X^{(k+1)})$ sea mínima. Para efectuar la búsqueda unidireccional puede usarse la tolerancia ε_2 directamente en el método de minimización adoptado. Alternativamente, puede checarse si $\|\nabla f(X^{(k+1)}) \cdot \nabla f(X^{(k)})\| \leq \varepsilon_2$.

Paso 5: ¿Es $\left|\frac{f(X^{(k+1)}) - f(X^{(k)})}{f(X^{(k)})}\right| \leq \varepsilon_1$? Si es así, Terminar. **ELSE** $k = k + 1$. Ir al Paso 2.

2 Experimentación con los resultados.

Primero, debido a que la función se puede graficar en un espacio de 3 dimensiones, se recurrió a esto para tener un referente visual del comportamiento de la misma:



La función ya no es tan fácil de visualizar o seguir, debido al aumento de una dimensión extra. Dicha función se comporta como una sabana arrugada, la cual parece tener un mínimo muy en el centro de la función con posibles mínimos locales a las orillas.

3 Experimentación

3.1 Resolución de clase

Durante la clase se abordo la forma de inicializar el método con los siguientes parámetros:

$$x_0 = (0, 0) \quad f(x_0) = 170 \quad tol1 = tol2 = 0.001 \quad a = 0 \quad b = 1$$

con el objetivo de encontrar el mínimo de la función planteada.

Una vez inicializado el algoritmo con dichos valores se hizo la prueba de correr el algoritmo.

A continuación se muestran los resultados de las evaluaciones que hizo el algoritmo:

Iter	**X1**	**X2**	**Y**	**Tol**
1	0.0000	0.0000	170.0000	5.0000
2	1.7828	2.8014	32.1208	Inf
3	3.0009	2.0261	0.0122	0.4348
4	2.9917	2.0118	0.0030	0.0047
5	3.0002	2.0063	0.0007	0.0028
6	2.9980	2.0029	0.0002	0.0011
7	3.0001	2.0014	0.0000	0.0007

Para esta evaluación de clase, con los parámetros iniciales dados, ocurre que el método logra converger en 7 iteraciones, de forma que el punto que encuentra y que logra satisfacer el criterio de tolerancia es $(3.0001, 2.0014)$.

4 Conclusiones y comentarios finales

El método del descenso del gradiente logra una solución factible bastante muy bien aproximada pero tiene la leve desventaja de que hay que estar calibrando el método de búsqueda interno que minimiza el tamaño del paso (λ), lo cual puede llevar bastante prueba y error.

5 Anexo 1: Codigo principal del método de descenso del gradiente.

```
## #### Función de Cauchy ####
##
## # fx = función multivariante
## # x = punto inicial
## # tol1 = tolerancia permitida
## # a = limite inferior del intervalo de busqueda
## # b = limite superior del intervalo de busqueda
##
## cauchy = function(fx,x,tol1,a,b){
##   # Parametros de inicio
##   tol2 = tol1
##   h = 0.0001
##   k = 1 # Punto de inicio par K
##   D = length(x) # Dimensiones de entrada
##   # Crear el objeto que guarda los valores de X
##   M = 100
##   X <- matrix(0, nrow = 1, ncol = D)
##   ### Funciones necesarias ###
##   #Función para calcular el gradiente
##   gradienten <- function(fx, X, h) {
##     D <- length(X)  # Dimensiones de la entrada
##     resu <- numeric(D)
##     f0 <- fx(X)     # Valor de la función en X actual
##     for (i in 1:D) {
##       perturbado <- X
##       perturbado[i] <- perturbado[i] + h
##       resu[i] <- (fx(perturbado) - f0) / h
##     }
##     return(resu)
##   }
##   # Función en terminos de lambda que minimiza el valor de fx(x-lambda*grad)
##   fa <- function(lambda) { fx(X[k,] - lambda * gradiente) }
##   # Función par extraer la norma
##   norma <- function(VEC){sqrt(sum(VEC^2))}
##   #### Paso 3 implicito como criterio de parada
##   # Norma auxiliar
##   normaG = 5
##   normaX = 5
##   tc = c(normaX)
##   # Paso 3 que implica el bucle hasta la convergencia
##   while(normaG >= tol1 & normaX >= tol1  & k < M){
##     # PASO 2: Calcular gradiente numérico
##     gradiente <- gradienten(fx, X[k, ], h)
##     # PASO 3: Calcula la norma del gradiente
##     normaG <- norma(gradiente)
##     # Paso 4: Efectuar la busqueda unidireccional par encontrar lambda
##     opt_result <- optimize(fa, interval = c(a, b), tol = 1e-5)
##     lmin <- opt_result$minimum
##     # Actualizar X
##     X =  rbind(X,X[k, ] - lmin * gradiente)
##     k <- k + 1
##     # Calcular las normas de X y el uevo X
```

```
##     normaX = norma(X[k,]-X[k-1,])/norma(X[k-1,])
##     tc = c(tc,normaX)
##   }
##
##   # Añadir al final la columna de resultados
##   X = cbind.data.frame(1:k,X,apply(X, 1, fx),tc)
##   # Ponerlo bonito para los resultados:
##   resu = as.data.frame(X)
##   names(resu) = c("Iter","X1","X2","Y","Tol")
##   resu = resu |> round(digits = 4)
##   # Devolver los resultados bonitos
##   return(resu)
## }
```

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

```
## rm(list=ls())
## # Cargar la función y dar los parametros iniciales
## source("Tarea 12. Busqueda Cauchy MAIN.R")
## library(flextable)
## fx = function(X){((X[1]^2+X[2]-11)^2 + (X[1] + X[2]^2-7)^2) |> as.numeric()}
## x = c(0,0) # Punto inicial
## tol1 = 0.001
## tol2 = 0.001
## # Paramertros para la derivada y el método de busqueda
## a = 0
## b = 1
## # Reportar los resultados
## resu = cauchy(fx,x,tol1,0,1)
## autofit(theme_box(flextable(resu)))
```