Optimización inteligente

Tarea 13. Método del Gradiente Conjugado (Fletcher-Reeves)

Ángel García Báez

2024-06-12

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 busqueda del gradiente conjugado.

A continuación se muestra el pseudocodigo para implementar dicho método:

1.1 Método de busqueda del Gradiente Conjugado

Algoritmo:

Paso 1: Elegir un punto inicial $x^{(0)}$ y tres parámetros de terminación ϵ_1, ϵ_2 y ϵ_3.

Paso 2: Encontrar $\nabla f(x^{(0)})$ y hacer $s^{(0)} = -\nabla f(x^{(0)})$.

Paso 3: Encontrar $\lambda^{(0)}$ tal que $f(x^{(0)} + \lambda^{(0)} s^{(0)})$ se minimice con una tolerancia de ϵ_1.

Hacer $x^{(1)} = x^{(0)} + \lambda^{(0)} s^{(0)}$ y $k = 1$. Calcular $\nabla f(x^{(1)})$.

Paso 4: Hacer $s^{(k)} = -\nabla f(x^{(k)}) + \frac{\|\nabla f(x^{(k)})\|^2}{\|\nabla f(x^{(k-1)})\|^2} s^{(k-1)}$.

Paso 5: Encontrar $\lambda^{(k)}$ tal que $f(x^{(k)} + \lambda^{(k)} s^{(k)})$ sea mínima con una tolerancia ϵ_1.

Hacer $x^{(k+1)} = x^{(k)} + \lambda^{(k)} s^{(k)}$.

Paso 6: ¿Es $\frac{\|x^{(k+1)} - x^{(k)}\|}{\|x^{(k)}\|} \leq \epsilon_2$ o $\|\nabla f(x^{(k+1)})\| \leq \epsilon_3$?

Si es así, TERMINAR.

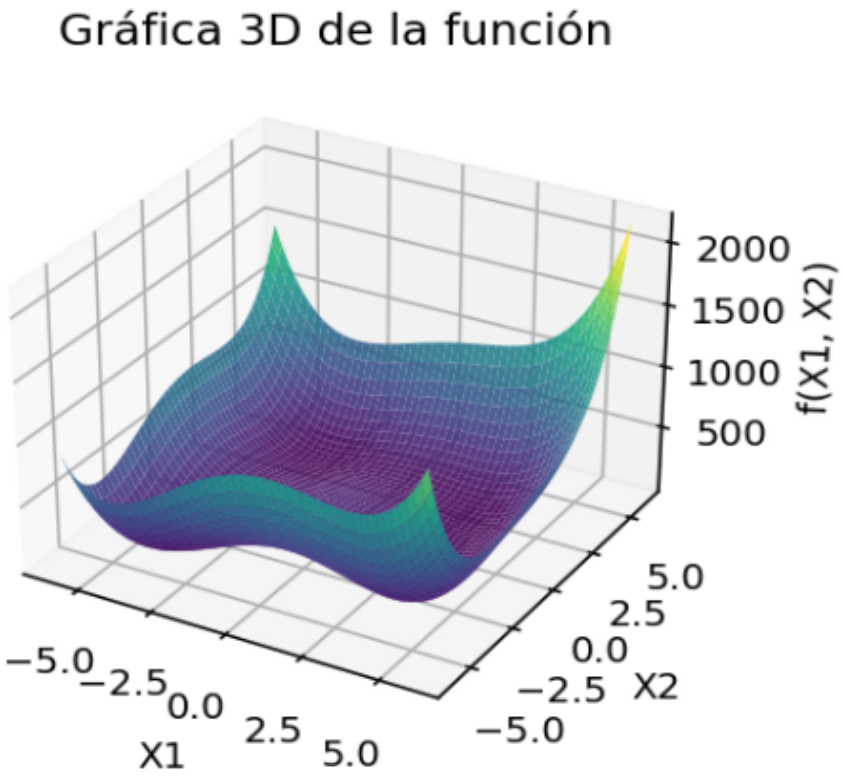
ELSE $k = k + 1$. GOTO Paso 4.

NOTA

Debido a la acumulación de errores de redondeo, para mejorar el desempeño del algoritmo se recomienda re-inicializar la dirección de búsqueda cada $m = n + 1$ (n variables) iteraciones, con la dirección del descenso empinado.

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 iniciaizar el método con los siguientes parámetros:

$$x_0 = (0, 0) \quad f(x_0) = 170 \quad \varepsilon = 0.001$$

adicionalmente se plantearon los limites $a = 0, b = 1$ para inicializar el método de eliminación de regiones dentro del método principal

Una vez inicializado el algoritmo con dichos valores de x_0, ε, a y b se hizo la prueba de correr el algoritmo.

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

X1	**X2**	**Y**
0.000000	0.000000	170.000000
1.780454	2.797856	32.126336
2.266582	2.992828	26.074439
2.701270	2.978864	21.454301
2.590075	-1.477784	38.240181
3.290016	-2.369533	10.106584
3.603310	-2.394679	5.634075
3.631445	-1.870423	0.117350
3.583558	-1.856888	0.001223
3.586240	-1.851443	0.000292
3.584272	-1.850442	0.000082

Para esta evaluación de clase, con los parámetros iniciales dados, el método logra encontrar un punto que aproxima bastante bien uno de los mínimos de la función. Dicho punto es $[3.584272, -1.850442]$ el cual satisface el criterio de tolerancia en apenas 11 iteraciones.

3.2 Propuesta con interalo de busqueda [-10,10]

Debido a la naturaleza tridimensional de la función, no es tan facil visualizar si es que dicha función es unimodal o multimodal. Se sospecha que la función tiene más de un minimo, por lo que se decide probar que sucede al ampliar el intervalo de busqueda del método de acotamiento para probar sí es que encuentra otro punto que se pueda considerar un minimo.

Se mantienen todos los parametros iguales a excepción del intervalo de busqueda, el cual cambia por $a = -10$ y $b = 10$

A continuación se pone a prueba el algoritmo con dichos cambios:

X1	**X2**	**Y**
0.000000	0.000000	170.000000
1.783504	2.802648	32.125785
2.263534	2.993953	26.178561
-2.785003	3.174484	0.090264
-2.801690	3.130518	0.000403
-2.805139	3.131228	0.000000

El algoritmo logra encontrar otro mínimo de la función en apenas 6 iteraciones. Dicho punto que también minimiza a la función es $[-2.805139, 3.131228]$.

4 Conclusiones

Tras la implementación del método y de las corridas de clase, se observa que el método logra aproximar bastante bien optimos de la función mediante el gradiente conjugado pero llama mucho la atención que se esperaba llegar al punto $[3, 2]$ pero se termino por llegar a otros puntos minimos que tambien satisfacen la restricción de tolerancia, ademas que se lleva su tiempo en converger, tal y como se pudo observar en la primera corrida de clase. Por lo que si bien, el método da buenos resultados, acarrea implicito el problema del redondeo decimal conforme va avanzando, lo que termina en situaciones como esta donde puede que no se encuentre el minimo que se busca y termine por llegar a otras soluciones locales.

5 Anexo 1: Codigo principal del método de Gradiente Conjugado

```
##
## #### Método de las conjugadas del gradiente
## conjugadas = function(fx,X0,tol,a,b){
##   #### Paso 1 ####
##   # Funciones auxiliares
##   # Función que calcula la norma
##   norma <- function(VEC){sqrt(sum(VEC^2))}
##   # Función que saca el gradiente numerico
##   gradienten <- function(fx, X, h) {
##     # Parametros de dimensionalidad
##     D <- length(X)  # Dimensiones de la entrada
##     resu <- numeric(D)  # Inicializamos el vector del gradiente
##     for (i in 1:D) {
##       # Crear copias de X para aplicar las perturbaciones
##       Xpos <- X
##       Xneg <- X
##       # Aplicar la perturbación en la dimensión i
##       Xpos[i] <- X[i] + h
##       Xneg[i] <- X[i] - h
##       # Calcular la diferencia de las evaluaciones
##       resu[i] <- (fx(Xpos) - fx(Xneg)) / (2 * h)
##     }
##     return(resu)
##   }
##   # Función para minimizar el lambda
##   optilambda <- function(X, S, a, b, tol1) {
##     fa <- function(lambda) fx(X + lambda * S)
##     res <- optimize(fa, interval = c(a, b), tol = tol1)
##     return(res$minimum)
##   }
##   # Definir las 3 tolerancias
##   tol1 = tol2 = tol3 = tol
##   # Definir el objeto que contenga los cambios en X
##   D = length(X0)
##   X = matrix(0,nrow = 1,ncol = D)
##   # Definir el objeto que va a contener las direcciones
##   S = matrix(0,nrow = 1,ncol = D)
##   # Tambien defino una matriz de gradientes
##   grads = matrix(0,nrow = 1,ncol = D)
##   #### Paso 2 ####
##   # Definir el X0 como el primer elemento de la matriz X
##   X[1,] = X0
##   # Encontrar el gradiente para f(X0)
##   grads[1,] = gradienten(fx,X[1,],tol1)
##   # Encontrar la dirección del gradiente S0
##   #Actualizar la dirección en la matriz de direcciones
##   S[1,] = -grads[1,]
##   # Paso 3 Encontrar un lambda0 tal que f(x0 + lambda0*S1)
##   # Sea minimo con tolerancia tol1
##   lmin = optilambda(X[1,],S[1,],a,b,tol1)
##   # Definir X1 como X0 + lambda0*S0
##   X = rbind(X,X[1,] + lmin*S[1,])
```

```
##   # Calcular el gradiente de X1
##   grads = rbind(grads,gradienten(fx,X[2,],tol1))
##   # Iniciar K en K + 1
##   k = 1
##
##   # Paso 6 implicito que devuelve al paso 4 de forma iterativa
##   norma2 = 5 # norma auxiliar para la condición de paro
##   norma3 = 5 # norma auxiliar para la condición de paro
##   while(norma2>tol2 & norma3 >tol3){
##     #### Paso 4 ####
##     # Crear una nueva dirección
##     # Considera reinicializar la cosa cada n + 1 iteraciones
##     #### Paso 4: Calcular nueva dirección ####
##     if(k %% (D+1) == 0){
##       # Re-inicializar dirección al descenso empinado
##       ns = -grads[k+1,]
##     } else {
##       # Actualizar dirección conjugada
##       ns = -grads[k+1,] + S[k,] * (norma(grads[k+1,])/norma(grads[k,]))^2
##     }
##     # Añadir la nueva dirección K
##     S = rbind(S,ns)
##     #### Paso 5 ####
##     #Encontrar un lambdaK tal que:
##     # f(Xk + lambdaK*Sk) sea minima para la tol1
##     lmin = optilambda(X[k+1,], S[k+1,], a, b, tol1);lmin
##     # Hacer que el nuevo Xk sea Xk + lambdaK*sK
##     X = rbind(X,X[k+1,] + lmin*S[k+1,] )
##     #### Paso 6 ####
##     # Verifica mediante las normas si el nuevo punto es
##     # parecido al punto anterior
##     norma2 = norma(X[k+2,]-X[k+1,])/norma(X[k+1,])
##     # Actualiza la matriz de gradientes con el nuevo punto
##     grads = rbind(grads, gradienten(fx,X[k+2,],tol1))
##     # Verifica si el gradiente ya cumple el criterio
##     norma3 = norma(grads[k+2,])
##     # Actualiza el K por K+1
##     k = k+1
##   }
##   # Revuelve la matriz de X
##   resu = as.data.frame(X)
##   # Calcular los valores de Y y la tolerancia
##   Y = c()
##   for(i in 1:nrow(X)){
##     Y[i] = fx(X[i,])
##   }
##   # Juntarlos
##   resu = round(cbind.data.frame(resu,Y),6)
##   names(resu)[-(D+1)] = paste0("X",1:D)
##   return(resu)
## }
```

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

```
## rm(list=ls())
## # Busqueda del gradiente conjugado
## # Librería para hacer las tablas bonitas
## # Llamo el código
## source("Tarea 13. Conjugadas  MAIN.R")
## # Declaro la función que se desea minimizar
## #### Evaluación para el ejercicio de clase ####
## # fx = función multivariable
## # X = punto inicial
## # tol = tolerancia
## # a = Limite inferior de busqueda
## # b = Limite superior de busqueda
## # tol = Tolerancia permitida
##
## # Parametros de inicio
## # Función objetivo
## fx = function(X){(X[1]^2+X[2]-11)^2 + (X[1] + X[2]^2-7)^2}
## # Parametros de la función
## X0 = c(0,0)
## h = 0.001
## a = 0.001
## b = 1
## tol = 0.001
## # Resultado de la corrida de clase forzandolo un poco
## conjugadas(fx,X0,tol,a,b)
```