# Optimización inteligente

## Tarea 14. Método Marquart

**Ángel García Báez**

**2024-09-12**

## 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 Marquart.

A continuación se muestra el pseudocodigo para implementar dicho método:

## 1.1 Método de Marquart

Algoritmo:

Paso 1: Elegir un punto inicial $X^{(0)}$, el número máximo de iteraciones, $M$, y un parámetro de terminación, $\epsilon$.
Hacer $k = 0$ y $\lambda^{(0)} = 1 \times 10^4$ (un valor grande).

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

Paso 3: IF $\|\nabla f(X^{(k)})\| \leq \epsilon$ o $k \geq M$ THEN Terminar

ELSE GOTO Paso 4.

Paso 4: Calcular $s(X^{(k)}) = -\left[H^{(k)} + \lambda^{(k)} I\right]^{-1} \nabla f(X^{(k)})$
Hacer $X^{(k+1)} = X^{(k)} + s(X^{(k)})$.

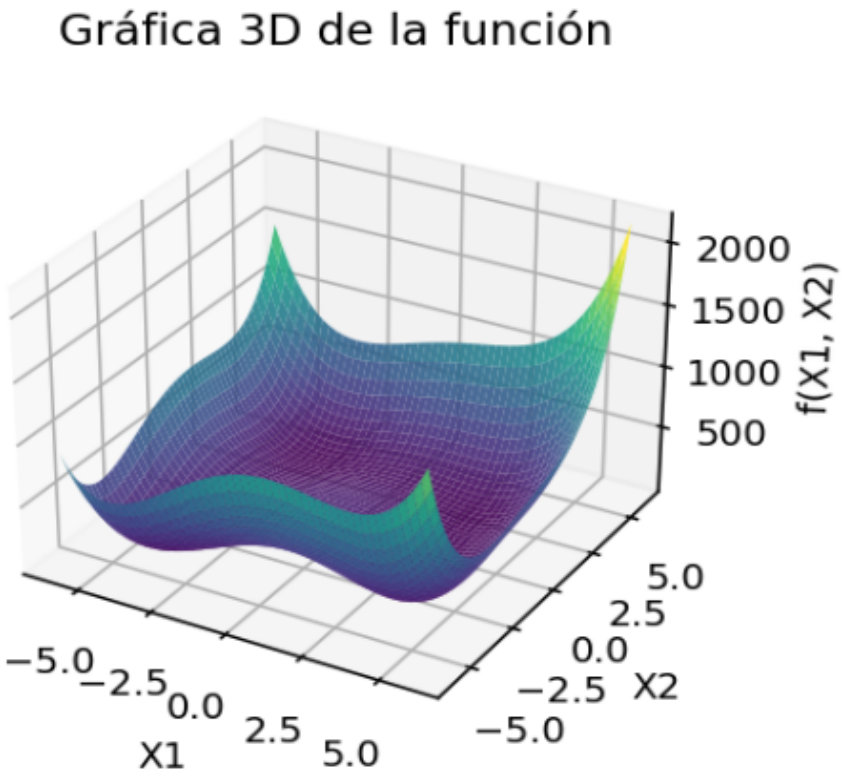
Paso 5: ¿Es $f(X^{(k+1)}) < f(X^{(k)})$?
Si es así, GOTO Paso 6.
ELSE GOTO Paso 7.

Paso 6: Hacer $\lambda^{(k+1)} = \frac{1}{2}\lambda^{(k)}$
$k = k + 1$. GOTO Paso 2.

Paso 7: Hacer $\lambda^{(k+1)} = 2\lambda^{(k)}$
GOTO Paso 4.

## 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:

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

Una vez inicializado el algoritmo con dichos valores de $x_0$, $\varepsilon$ se hizo la prueba de correr el algoritmo.

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

| **X1** | **X2** | **f(X)** | **NormaGrad** | **DiffX** |
|---:|---:|---:|---:|---:|
| 0.000000 | 0.000000 | 170.000000 | 5.000000 | 5.000000 |
| -1.098164 | -2.787647 | 158.405798 | 26.076810 | 0.068201 |
| 3.411509 | -0.340989 | 12.144742 | 58.697830 | 0.923331 |
| 3.275671 | 1.861530 | 2.600123 | 6.061872 | 0.785905 |
| 2.999708 | 2.009866 | 0.001608 | 20.374124 | 0.999382 |
| 2.999983 | 2.000009 | 0.000000 | 0.375652 | 0.999994 |
| 3.000000 | 2.000000 | 0.000000 | 0.001060 | 0.999899 |
| 3.000000 | 2.000000 | 0.000000 | 0.000008 | 0.000000 |

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 $[3, 2]$ el cual satisface el criterio de tolerancia en apenas 8 iteraciones.

# 4 Anexo 1: Codigo principal del método de Marquart

```
##
## # Función del método de Newthon HIBRIDO #
## NH = function(fx,X0,tol,a,b){
##   #### Paso 1, elegir un punto inicial
##   # Definir internamente las funciones que necesito
##   ## Función que calcula la norma
##   norma <- function(VEC){sqrt(sum(VEC^2))}
##   ## Función del gradiente
##   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 calcular la Hessiana
##   hess =  function(fx,X0){
##     # Definir las dimensiones
##     P <- length(X0) # Dimensión
##     tol <- 0.001 # Perturbación numérica
##     # Inicializar la matriz Hessiana
##     H <- matrix(0, ncol = P, nrow = P)
##     # Calcular la Hessiana
##     for (i in 1:P) {
##       for (j in 1:P) {
##         if (i == j) {
##           # Diagonal principal: segunda derivada respecto a la misma variable
##           # Generar el vector de perturbaciones
##           vec_perturb <- numeric(P)
##           # Dar el valor del error de pertubación en la variable i-esima
##           vec_perturb[i] <- tol
##           H[i, j] <- (fx(X0 + vec_perturb) - 2 * fx(X0) + fx(X0 - vec_perturb)) / (tol^2)
##         } else if (i > j) {
##           # Derivadas cruzadas (fuera de la diagonal)
##           vec_perturb_i <- numeric(P) # Un vector para perturbar la variable i
##           vec_perturb_j <- numeric(P) # Un vector para perturbar la variable j
##           vec_perturb_i[i] <- tol #Dar el valor de la pertubación para i
##           vec_perturb_j[j] <- tol #Dar el valor de la pertubación para j
##           # Fórmula para derivadas cruzadas
##           H[i, j] <- (fx(X0 + vec_perturb_i + vec_perturb_j) -
##                         fx(X0 + vec_perturb_i - vec_perturb_j) -
##                         fx(X0 - vec_perturb_i + vec_perturb_j) +
##                         fx(X0 - vec_perturb_i - vec_perturb_j)) / (4 * tol^2)
```

```
##           # Simetría: H[j, i] = H[i, j]
##           H[j, i] <- H[i, j]
##         }
##       }
##     }
##     # Devolver la matriz Hessiana como salida
##     return(H)
##   }
##   # Función para minimizar el lambda
##   optilambda <- function(X, a, b, tol) {
##     # Calcular el gradiente la Hessiana
##     GH = solve(hess(fx,X))%*%gradienten(fx,X,0.001)
##     # Actualizar la función lambda temporal
##     fa <- function(lambda) fx(X-lambda*GH)
##     # Optimizar el lambda
##     res <- optimize(fa, interval = c(a, b), tol = tol)
##     # EXtraer el Alfa
##     lambd = res$minimum
##     # Calcular el nuevo valor de Xk
##     Xk = X-lambd*GH
##     # Reportar el resultado
##     return(Xk)
##   }
##   # Definir la matriz de valores de X
##   # Definir el objeto que contenga los cambios en X
##   D = length(X0)
##   X = matrix(X0,nrow = 1,ncol = D)
##   # Definir el K inicial
##   k = 0
##   # Calcular f(X) en el punto y guardarlo
##   FX = c(fx(X0))
##   # Condición auxiliar para el método
##   cond1 = cond2 = 5
##   # Guargar en un vector
##   cond1l = cond1
##   cond2l = cond2
##   #### Paso 3 implicito como condición de paro ####
##   while(cond1>tol&cond2>tol ){
##     #### Paso 2 Calcular el gradiente K-esimo
##     gradiente = gradienten(fx,X[k+1,],0.001)
##     # Paso 4, efectuar la busqueda unidireccional para encontrar lambda K
##     X = rbind(X, t(optilambda(X[k+1,],a,b,0.001)))
##     #Paso 5. es (f(xk+1)-f(xk) / f(xk))<0.001
##     # Actualizar fx para xk+1
##     FX = c(FX,fx(X[k+2,]))
##     # Verificar los criterios de paro
##     cond1 = norma(gradiente)
##     cond2 = abs((FX[k+2]-FX[k+1])/FX[k+1])
##     # Guardar los criterios
##     cond1l = c(cond1l,cond1)
##     cond2l = c(cond2l,cond2)
##     # Verificar la condición
##     # Actualizar el K = K+1
##     k = k+1
```

```
##     # Regresa al paso 2
##   }
##   # Acomodar la salida
##   resu = round(cbind.data.frame(X,FX,cond1l,cond2l),6)
##   names(resu) = c(paste0("X",1:D),"f(X)","NormaGrad","DiffX")
##   # Reportar los resultados
##   return(resu)
## }
```

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

```
## rm(list=ls())
##
## # Cargar la función principal
## source("Tarea 14 Marquart MAIN.R")
## #### Evaluación de clase ####
## # 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)
## a = 0
## b = 5
## tol = 0.001
## NH(fx,X0,tol,a,b)
## #### Evaluación del libro ####
## X1 = c(2,1)
## a = 0
## b = 5
## tol = 0.001
## NH(fx,X1,tol,a,b)
```