Universidad Nacional Autónoma de México

Escuela Nacional de Estudios Superiores

Unidad Juriquilla

Computación III

Tarea 2

Método de Gradiente para Optimización en varias variables sin restricciones

Bruno Arturo López Pacheco

Dr. Ulises Olivares Pinto

No. De Cuenta 317347140

Licenciatura en Tecnología

Objetivo

Utilizar el método del gradiente para optimización en varias variables sin restricciones mostrando su desempeño en diferentes funciones de varias variables.

Marco teórico

Método de gradiente para optimización en varias variables sin restricciones

El ascenso de gradiente es un algoritmo que estima numéricamente dónde una función genera sus valores críticos. Esto significa que encuentra máximos o mínimos locales.

Se sabe que el gradiente evaluado en cualquier punto representa la dirección del ascenso o descenso más pronunciado por alguna función. De aquí nace la idea de maximizar la función, comenzando con una entrada aleatoria, y tantas veces como se pueda, dar un pequeño paso en dirección del gradiente para moverse "ascendiendo" la función.

En caso de que se quiera minimizar la función, se usa el negativo del gradiente para ir en el descenso más pronunciado. Teniendo un punto inicial x_o y nos movemos una distancia positiva α en la dirección del gradiente negativo, el nuevo x_1 queda como:

$$x_1 = x_0 + \alpha \nabla f(x_0)$$

O:

$$x_{n+1} = x_n + \alpha \nabla f(x_n)$$

Una de sus limitaciones es que solo encuentra mínimos locales (en lugar del mínimo global). Tan pronto como el algoritmo encuentra algún punto que sea un mínimo local, nunca escapará mientras el tamañi de paso no exceda el tamaño del foso.

Metodología

Implementación

Se implementaron varias funciones genéricas para facilitar la implementación de este método. Entre ellas están:

Función que crea el vector gradiente a partir de la función original:

```
def gradient(self,f,xs):
        grad=[]
        for i in range(len(xs)):
            grad.append(diff(f,xs[i]))
        return grad
```

Función que crea matriz hessiana a partir de la función original:

```
def hessian(self,f,xs):
        hess=[]
        for i in range(len(xs)):
            aux=[]
            for j in range(len(xs)):
                aux.append(diff(diff(f,xs[i]),xs[j]))
            hess.append(aux)
        return hess
```

Función que sustituye valores de puntos en los vectores y matrices:

```
def substitution(self,array,xs,point,type):
        if type==2:
            #print("es hessiana")
            for i in range(len(array)):
                for j in range(len(array)):
                    for k in range(len(xs)):
                        array[i][j]=array[i][j].subs(xs[k],point[k])
            return array
        elif type==1:
            #print("es vector")
            for i in range(len(array)):
                for j in range(len(xs)):
                    array[i]=array[i].subs(xs[j],point[j])
            return array
```

Función que calcula el producto punto entre un vector y una constante:

```
def mult(self,array,var):
        res=[]
        for i in range(len(array)):
            res.append(array[i]*var)
        return res
```

Función que realiza el producto punto entre dos vectores:

```
def producto(self,array1,array2):
      res=0
      for i in range(len(array1)):
        res+=array1[i]*array2[i]
      return res
```

Función que realiza la suma entre dos vectores:

```
def suma(self,array1,array2):
        res=[]
        for i in range(len(array1)):
            res.append(array1[i]+array2[i])
        return res
```

Función que realiza el cálculo del error usando la definición de la norma vectorial:

```
def norma(self,vector0,vector1):
        # Cálculo de error por definición de norma
        suma=0
        for i in range(len(vector0)):
            aux=(vector0[i]-vector1[i])**2
            suma=suma+aux
        return m.sqrt(suma)
```

Función que realiza el método central del gradiente, basado en la siguiente ecuación:

$$x_{n+1} = x_n + \alpha \nabla f(x_n)$$

Pruebas

Se realizaron 4 pruebas, tomando 4 funciones arbitrarias con 1,2,3 y 4 variables. Se tomó el tiempo total de cómputo para encontrar sus puntos máximos.

También se realizó una comparación gráfica entre los tiempos de ejecución de dichas funciones arbitrarias.

Las funciones utilizadas son:

$$f_1(x_1) = -x_1^2 + 2$$

$$f_2(x_1, x_2) = 2x_1x_2 + 2x_1 - x_1^2 - 2x_2^2$$

$$f_3(x_1, x_2, x_3) = -x_1^2 - x_2^2 - x_3^2$$

$$f_4(x_1, x_2, x_3, x_4) = -x_1^2 - x_2^2 - x_3^2 - x_4^2$$

Tomando los puntos iniciales:

$$f_1(-2)$$

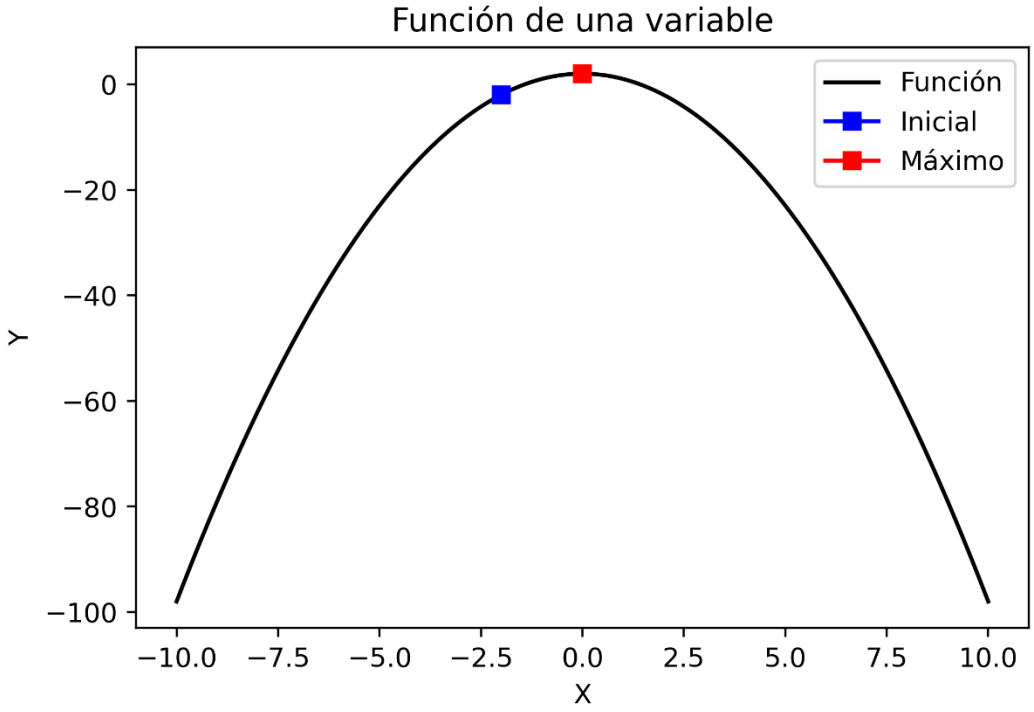
$$f_2(2{,}0)$$

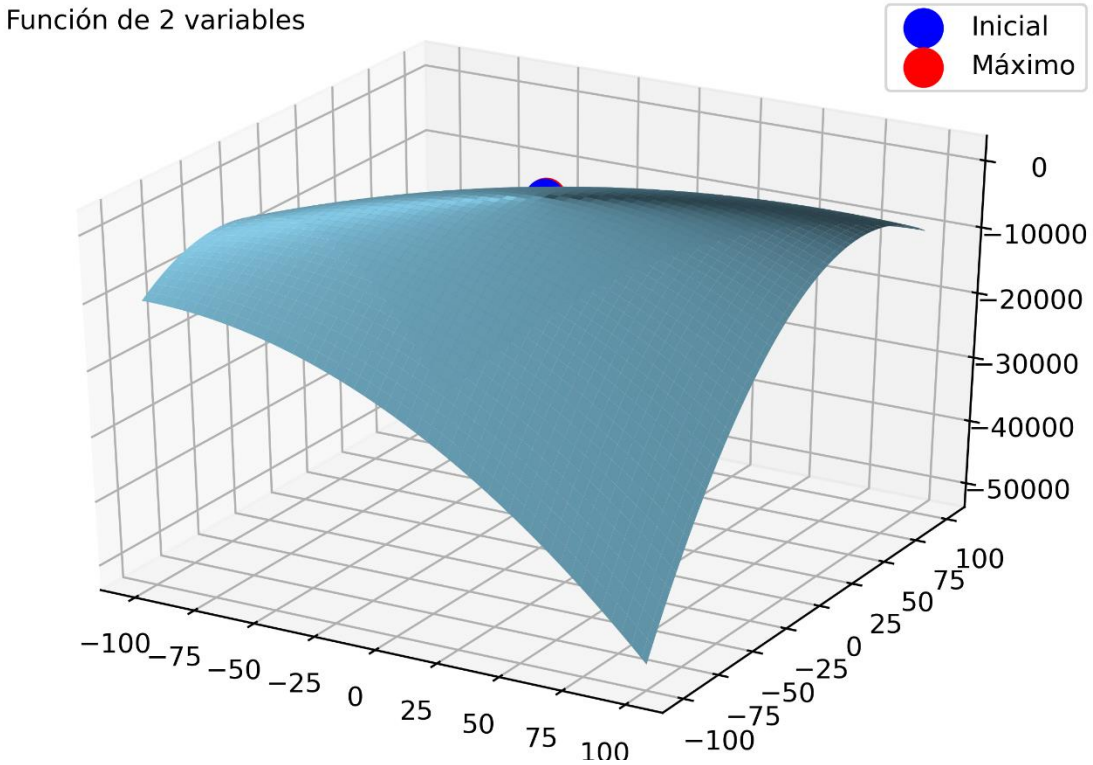
$$f_3(-10{,}10{,}5)$$

$$f_4(-10{,}10{,}5{,}-5)$$

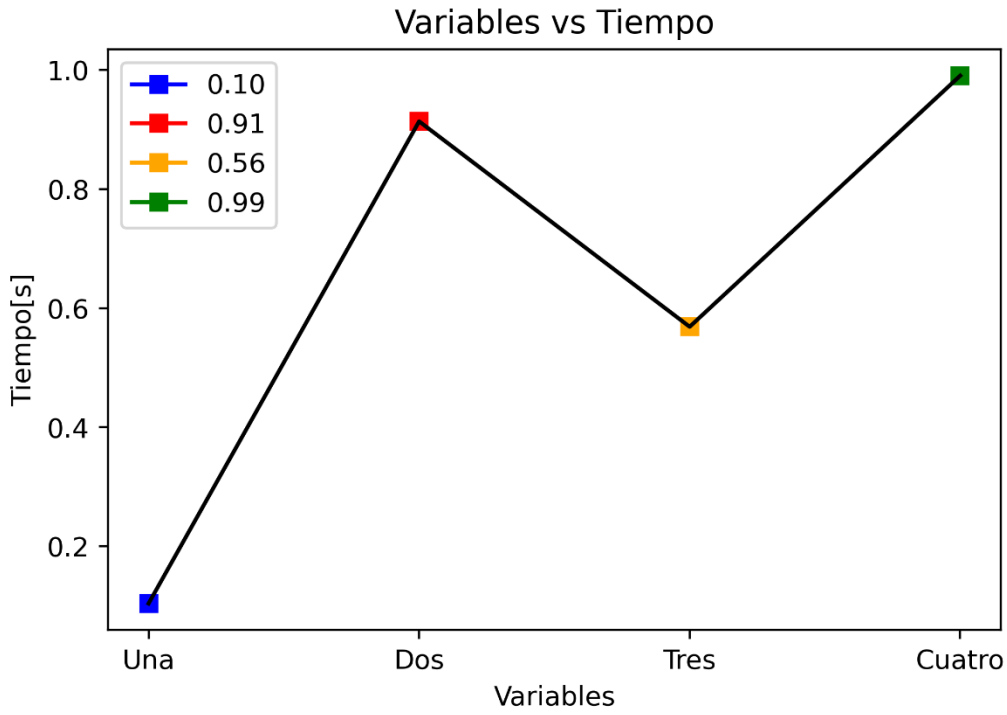
Resultados

Las gráficas de las primeras dos funciones junto con sus puntos iniciales de búsqueda y puntos máximos locales se encuentran a continuación:





Para estas 4 funciones se pudieron encontrar sus mínimos locales. Y sus tiempos de ejecución se muestran en la siguiente gráfica:



Conclusión

A pesar de que los tiempos de ejecución fueron relativamente bajos, el número de iteraciones es bastante alto, esto debido al tamaño de paso en cada iteración tomada. Hubo un caso en el que se tuvieron 165 iteraciones, pero se cree que esto es debido a la naturaleza del método básico de gradiente. Por lo cual se concluye que la velocidad de convergencia del método de ascenso del gradiente es muy baja con respecto a otros métodos (por ejemplo el de Newton).

Referencias

Chapra, S., & Canale, R. (2011). METODOS NUMERICOS PARA INGENIEROS (Spanish Edition) (6th ed.). McGraw-Hill Interamericana de España S.L.