Instituto Tecnológico de Buenos Aires

Trabajo Práctico N$^{\underline{o}}$5

Optimización

93.54 Métodos Numéricos

Grupo N°4

Legajo N°	Nombre
61428	Kevin Amiel Wahle
61430	Francisco Basili
61431	Nicolás Bustelo

18/06/2022

Índice

1. Introducción

El objetivo de este informe es presentar y explicar el funcionamiento de un programa realizado en *Python*, el cual consiste en implementar la minimización de una función basada en el método de máximo descenso utilizando interpolación cuadrática para la búsqueda lineal. El código utilizado se encuentra en el Anexo, al final del informe, el cual incluye el testbench utilizado.

2. *minimi(f, Df, x0, tol, maxiter)*

Para cumplir con lo pedido se debía cumplir con lo siguiente:

1. $d_k = -\nabla f(x_k)$
2. Obtengo α_k (se explicará posteriormente)
3. $x_{k+1} = x_k + \alpha_k \cdot d_k$

Se calcularon las diferentes derivadas parciales para la obtención del gradiente en la función *grad(coef)*.

Para la obtención del α_k, se utilizó interpolación cuadrática. Para ello se debieron tener en cuenta algunos casos para modificar a, b y c de forma tal de "encontrar una sonrisa". En caso de no ser posible, porque el intervalo es demasiado pequeño o demasiado grande, se volvía a comenzar hasta una determinada cantidad de veces. Superado ese limite de pruebas se asignaba $X_n = x$. Finalmente se utilizó que:

$$hmin = (c-a)\frac{4f_c - f_b - 3f_a}{4f_c - 2f_b - 2f_a} = c \cdot \frac{4f_c - f_b - 3f_a}{4f_c - 2f_b - 2f_a}$$

para obtener el α_k que se utilizaría en el procedimiento ya mencionado previamente.

3. temperatura()

Se indicó una tolerancia de forma que fuera un orden mayor al épsilon de máquina, junto con un número máximo de iteraciones. Luego proponiendo valores iniciales experimentales, se llamó a la función principal explicada en la sección previa con los parámetros correspondientes, obteniéndose los coeficientes pedidos por la consigna, de forma tal de que la función brindada por la cátedra se ajustase lo mejor posible a los datos del archivo. Sus valores promedio, luego de ejecutar la con diferentes x_0, fueron $[a, b, c, T1, T2] = [35.999, -0.6, 0.9991, 23.999, 24]$.

4. *test()*

Con el objetivo de verificar el funcionamiento del código, decidimos calcular en una función aparte diferentes tipos de error entre los datos del enunciado y los generados con los parámetros obtenidos. Los resultados fueron:

1. ECM: 0.661
2. Error Máximo $1.69^o C$
3. Error Relativo medio: 1.96 %

Luego graficamos comparativamente los datos del enunciado con los obtenidos en figura 1. A pesar de no coincidir exactamente a causa de la no selección del mejor x_0, podemos ver que la función se asimila visualmente en forma a la de los datos.

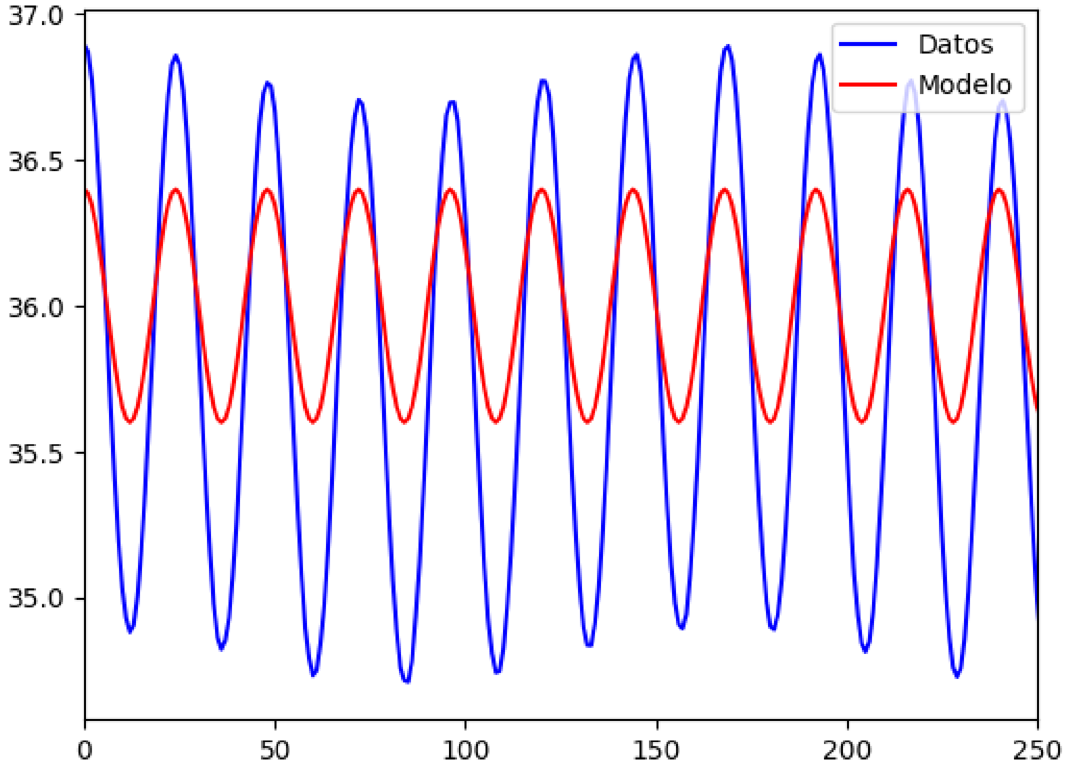

Figura 1: Análisis de los resultados

Adicionalmente, decidimos probar el código con la función de una esfera centrada en $(0, 0, 0)$ (véase ecuación (1)) y probar buscar su mínimo con la función desarrollada. Se puede ver que a pesar de elegir $x_0 = (-47.5, 20, -12.6)$, la función dió como resultado el valor real el cual es $x = (0, 0, 0)$.

$$f(\overline{x}) = \sum_{i=1}^{3} x_i^2 \qquad grad(\overline{x}) = (2x_1, 2x_2, 2x_3) \tag{1}$$

5. Anexo

5.1. Código completo en Python

```python
# ----------------------------------------------------------------------------
#  @file     +temperamental.py+
#  @brief    +Optimización+
#  @author   +Grupo 4+
# ----------------------------------------------------------------------------

# ----------------------------------------------------------------------------
# LIBRARIES
# ----------------------------------------------------------------------------
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math as mt

# ----------------------------------------------------------------------------
# FUNCTION DEF
# ----------------------------------------------------------------------------
data = np.loadtxt("temp.txt")
t=data[:,0]; y=data[:,1]

#Optimización basada el método de gradientes conjugados usando interpolación cuadrática
def minimi(f, Df, x0, tol, maxiter):
    x = x0
    g = -Df(x) # Gradiente

    # Evaluo primero si el gradiente es menor a la tolerancia
    if(np.linalg.norm(g)==0):
        print("X0 es un mínimo")
        return x

    # Determinamos un alfa para cada paso de interación usando interpolación cuadrática
    for i in range(maxiter):
        alfa_min = 1
        #H = alfa_min * g

        a = 0
        fa= f(x+a*g)
        b = 1
        fb= f(x+b*g)
        c = 1/2
        fc= f(x+c*g)
      #(a,fa) - (c,fc) - (b,fb)
        k=1
        while 1 :
            if fa > fc:
                #sigue "bajando"
                if fc > fb:
                    c, fc  = b, fb
                    b *= 2
                    fb = f(x+b*g)
                else:
                    break
            else:
                #está aumentando
                b, fb  = c, fc
                c  /= 2
                fc = f(x+c*g)

            #si el intervalo es demasiado chico o demasiado grande, comencemos nuevamente
            if (b < 1e-6) or (b > 1e6) :
                k = k + 1
                if k == 100 :
                    break
                b = np.random.rand(1)
                c /= 2

        # Si después de muchas iteraciones, no llegamos a nada, devuelvo algo al azar
    entre 0 y 1.
```

```
        if k == 100 :
            x_n = x
        else:
            alfa_min = c*((4*fc-fb-3*fa)/(4*fc-2*fb-2*fa))
            x_n = x + alfa_min*g

        if(np.linalg.norm(x_n-x)<tol):
            break
        x = x_n
    return x

f_aux= lambda x: y - (x[0]+x[1]*np.cos(2*np.pi*t/x[3])+x[2]*np.cos(2*np.pi*t/x[4]))
f = lambda x: np.sum(np.square(f_aux(x)))/len(f_aux(x))

def grad(coef):
    # coef = [a, b, c, T1, T2]
    df=np.zeros(len(coef))
    difflocal = f_aux(coef)

    df[0]=-2*np.sum(difflocal)  #d/da
    df[1]=-2*np.sum(difflocal*np.cos(2*np.pi*t/coef[3]))   #d/db
    df[2]=-2*np.sum(difflocal*np.cos(2*np.pi*t/coef[4]))   #d/dc
    df[3]=-2*np.sum(difflocal*coef[1]*np.pi*2*t*np.sin(2*np.pi*t/coef[3])/(coef[3]**2)) #
    d/dT1
    df[4]=-2*np.sum(difflocal*coef[2]*np.pi*2*t*np.sin(2*np.pi*t/coef[4])/(coef[4]**2)) #
    d/dT2
    return df

def temperatura():
    xo=np.array([36.00,-0.6,1.0,24.00,24.00])
    tol=1e-15; max_it=1000
    x = minimi(f, grad, xo, tol, max_it)
    error = f_aux(x) # Error local
    return x, error

# ------------------------------------------------------------------------------
# TEST
# ------------------------------------------------------------------------------
def temp_test():
    #Evaluamos la función temperatura
    x, error=temperatura()
    print("Coef obtenidos ([a,b,c,T1,T2]): \n", x)
    print("ECM: ", np.sum(np.power(error,2))/len(error))
    print("Error Máximo", np.max(np.abs(error)), " C ")
    print("Error Relativo medio: ", (np.sum(abs(error/y))/len(error))*100, "%")

    temp = lambda t: (x[0]+x[1]*np.cos(2*np.pi*t/x[3])+x[2]*np.cos(2*np.pi*t/x[4]))

    plt.plot(t, y, 'b', label='Datos')
    plt.plot(t, temp(t), 'r', label='Modelo')
    plt.xlim(0, 250)
    plt.legend()
    plt.show()
    return

esfera = lambda x: x[0]**2+x[1]**2+x[2]**2
gradesfera = lambda x: np.array([2*x[0], 2*x[1], 2*x[2]])

def test():
    tol=1e-15; max_it=1000

    print("Funciones de prueba para minimi:\n")
    print("Función esférica: \n")
    x0esf=np.array([-47.5,20,-12.6])
    x1=minimi(esfera, gradesfera, x0esf, tol, max_it)
    print("El mínimo real es: [0,0,0] mientras que el mínimo calculado por minimi con x0=
    ", x0esf, "es: ", x1)
    print("\nLa norma de la diferencia entre el mínimo real y el cálculado es: ", np.
    linalg.norm(np.zeros(3)-x1))

    print("\n\n")

```

```
    print("Evaluación de la funcion de temperatura: \n")
    temp_test()
    return
```