Optimización I. Tarea 8

Y. Sarahi García Gozález

Librerías

```
In [ ]: import numpy as np
        import functions as fn #libreria con las funciones Himmelblau,Beale,Rosenbrock,Hartman
        from math import exp
```

```
In [ ]: #imprimimos el epsilon de la máquina
        epsilon = np.finfo(float).eps
        print("Epsilon de la máquina:", epsilon)
```

```
Epsilon de la máquina: 2.220446049250313e-16
```

```
In [ ]: def imprime(f,n,x0,metodo,metodo_args):

            '''
            Esta función imprime:
            - la dimensión $n$,
            - f(x_0)
            - el  número $k$ de iteraciones realizadas
            - f(x_k)
            - las primeras y últimas 4 entradas del punto $\mathbf{x}_k$ que devuelve el algoritmo,
            - la norma del gradiente g_k,
            - promedio de iteraciones del algoritmo 1
            - la variable $bres$ para saber si el algoritmo puedo converger.

            '''

            xk,fk,gk,k,indicador= metodo(*metodo_args)



            print('Dimensión n = ', n )
            print('f(x_0) = ', f(x0) )
            print('Número de iteraciones = ', k )
            print('f(x_k) = ', fk )
            print('Primeras cuatro entradas de x_k= ', xk[:4])
            print('Últimas cuatro entradas de x_k= ', xk[-4:])
            print('Norma del gradiente ||gk||   = ', np.linalg.norm(gk))



            if(indicador):
                print("Sí se cumplio el criterio de convergencia")
```

```
In [ ]: #definimos la funcion que genera la matriz A de acuerdo a la instrucción anterior
        def genera_A1(n):
            return np.ones((n,n))+n*np.eye(n)
        #definimos la funcion que genera la matriz A de acuerdo a la instrucción anterior
        def genera_A2(n):
            A = np.empty([n,n], dtype=float)
            for i in range(n):
                for j in range(n):
                    u=0.25*((i-j)**2)
                    A[i][j]=exp(-1*u)
            return A
```

Ejercicio 1

```
In [ ]: def backtracking(alpha_ini,x_k,f,f_k,df,p_k,rho=0.5,c=0.001,iter_max=500):

            '''
                Esta funcion parte de un tamaño de paso inicial alpha_ini y lo va recortando hasta que
                cumple la cond de descenso suficiente

                parametros:
                    valores (float): alpha_ini, rho entre (0,1), f(x_k), Df(x_k) (gradiente en el punto x_k), c_1,
                    direccion de descenso (np.rray): p_k

                returns:
                    el tamaño de paso a_k
                    numero de iteraciones realizadas i_k
            '''

            alpha=alpha_ini #fijamos alpha como el alpha inicial

            for i in range(iter_max):
                x_kp=x_k+alpha*p_k
                gp=c*np.dot(df(x_k),p_k) #hacemos el producto gradiente por direccion de descenso p


                #si la condicion de descenso se cumple, terminamos
                if f(x_kp)<=(f_k + alpha*gp):
                    return alpha

                alpha=alpha*rho #si no se cumple la cond, hacemos alpha*rho

            return  alpha
```

```
In [ ]: #def Newton_truncado(f,Df,Hf,x0,tol,max_iter=5000):

        def BFGS_modificado(f, Df, H0, x0, tol, max_iter=5000):
            xk = x0
            gk = Df(xk)
            Hk = H0
            n = len(xk)
            I = np.identity(n)
            for k in range(max_iter):

                if np.linalg.norm(gk) <= tol:
                    return xk, f(xk), gk,k, True

                pk = np.dot(-Hk, gk)
                if np.dot(pk,gk) > 0:
                    l1 = 10e-5 + (np.dot(pk,gk)/np.dot(gk,gk))
                    Hk = Hk + l1*I
                    pk = pk - l1*gk

                alpha = backtracking(1, xk, f,f(xk), Df, pk)
                xk_new = xk + alpha*pk
                gk_new = Df(xk_new)
                y = gk_new - gk
                s = xk_new - xk

                if np.dot(y,y) < tol:
                    return xk, f(xk), gk,k, True

                if np.dot(y,s) <= 0:
                    l2 = 10e-5 - (np.dot(y,s)/np.dot(y,y))
                    Hk = Hk + l2*I
                else:
                    rho_k = 1/np.dot(y,s)
                    Hk = (np.identity(n) - rho_k * (s@y.T)) @ Hk @ (np.identity(n) - rho_k * (y@s.T)) + rho_k * (s@s.T)

                xk = xk_new
                gk = gk_new


            return xk, f(xk), gk,k, False
```

Función cuadrática con A1

$n = 10$

```
In [ ]: n=10
        H0 = np.identity(n)
        tau=(np.sqrt(n))*((epsilon)**(1/3))
        A=genera_A1(n)
        b=np.ones(n)
        x0=np.zeros(n)
        #definimos la funcion cuadratica, su gradiente y su Hessiana como inline functions
        cuadratica = lambda x: 0.5 * np.dot(x.T, np.dot(A, x)) - np.dot(b.T, x)
        D_cuadratica = lambda x: 0.5 * (np.dot(A.T, x) + np.dot(A, x)) - b


        argumentos_NT=[cuadratica,D_cuadratica,H0,x0,tau]
        imprime(cuadratica,n,x0,BFGS_modificado,argumentos_NT)
```

```
Dimensión n =  10
f(x_0) =  0.0
Número de iteraciones =  4
f(x_k) =  -0.24999997743284488
Primeras cuatro entradas de x_k=  [0.04998498 0.04998498 0.04998498 0.04998498]
Últimas cuatro entradas de x_k=  [0.04998498 0.04998498 0.04998498 0.04998498]
Norma del gradiente ||gk||   =  0.0009500979969103435
Sí se cumplio el criterio de convergencia
```

$n = 100$

```
In [ ]: n=100
        H0 = np.identity(n)
        tau=(np.sqrt(n))*((epsilon)**(1/3))
        A=genera_A1(n)
        b=np.ones(n)
        x0=np.zeros(n)
        #definimos la funcion cuadratica, su gradiente y su Hessiana como inline functions
        cuadratica = lambda x: 0.5 * np.dot(x.T, np.dot(A, x)) - np.dot(b.T, x)
        D_cuadratica = lambda x: 0.5 * (np.dot(A.T, x) + np.dot(A, x)) - b

        argumentos_NT=[cuadratica,D_cuadratica,H0,x0,tau]
        imprime(cuadratica,n,x0,BFGS_modificado,argumentos_NT)
```

```
Dimensión n =  100
f(x_0) =  0.0
Número de iteraciones =  9
f(x_k) =  -0.24999996545321657
Primeras cuatro entradas de x_k=  [0.00500186 0.00500186 0.00500186 0.00500186]
Últimas cuatro entradas de x_k=  [0.00500186 0.00500186 0.00500186 0.00500186]
Norma del gradiente ||gk||   =  0.00371735302578493
Sí se cumplio el criterio de convergencia
```

$n = 1000$

```
In [ ]: n=1000
        H0 = np.identity(n)
        tau=(np.sqrt(n))*((epsilon)**(1/3))
        A=genera_A1(n)
        b=np.ones(n)
        x0=np.zeros(n)
        #definimos la funcion cuadratica, su gradiente y su Hessiana como inline functions
        cuadratica = lambda x: 0.5 * np.dot(x.T, np.dot(A, x)) - np.dot(b.T, x)
        D_cuadratica = lambda x: 0.5 * (np.dot(A.T, x) + np.dot(A, x)) - b


        argumentos_NT=[cuadratica,D_cuadratica,H0,x0,tau]
        imprime(cuadratica,n,x0,BFGS_modificado,argumentos_NT)
```

```
Dimensión n =  1000
f(x_0) =  0.0
Número de iteraciones =  11
f(x_k) =  -0.24999996810151942
Primeras cuatro entradas de x_k=  [0.00050018 0.00050018 0.00050018 0.00050018]
Últimas cuatro entradas de x_k=  [0.00050018 0.00050018 0.00050018 0.00050018]
Norma del gradiente ||gk||   =  0.011295747369060454
Sí se cumplio el criterio de convergencia
```

Función cuadrática con A2

$n = 10$

```
In [ ]: n=10
        H0 = np.identity(n)
        tau=(np.sqrt(n))*((epsilon)**(1/3))
        A=genera_A2(n)
        b=np.ones(n)
        x0=np.zeros(n)

        argumentos_NT=[cuadratica,D_cuadratica,H0,x0,tau]
        imprime(cuadratica,n,x0,BFGS_modificado,argumentos_NT)
```

```
Dimensión n =  10
f(x_0) =  0.0
Número de iteraciones =  5
f(x_k) =  -1.582691875500642
Primeras cuatro entradas de x_k=  [0.31823556 0.31669959 0.31597412 0.31576632]
Últimas cuatro entradas de x_k=  [0.31576632 0.31597412 0.31669959 0.31823556]
Norma del gradiente ||gk||   =  0.474850390221194
Sí se cumplio el criterio de convergencia
```

$n = 100$

```
In [ ]: n=100
        H0 = np.identity(n)
        tau=(np.sqrt(n))*((epsilon)**(1/3))
        A=genera_A2(n)
        b=np.ones(n)
        x0=np.zeros(n)

        #definimos la funcion cuadratica, su gradiente y su Hessiana como inline functions
        cuadratica = lambda x: 0.5 * np.dot(x.T, np.dot(A, x)) - np.dot(b.T, x)
        D_cuadratica = lambda x: 0.5 * (np.dot(A.T, x) + np.dot(A, x)) - b


        argumentos_NT=[cuadratica,D_cuadratica,H0,x0,tau]
        imprime(cuadratica,n,x0,BFGS_modificado,argumentos_NT)
```

```
Dimensión n =  100
f(x_0) =  0.0
Número de iteraciones =  6
f(x_k) =  -14.258764201243128
Primeras cuatro entradas de x_k=  [0.28510335 0.28509147 0.28508586 0.28508425]
Últimas cuatro entradas de x_k=  [0.28508425 0.28508586 0.28509147 0.28510335]
Norma del gradiente ||gk||   =  0.5417374850841334
Sí se cumplio el criterio de convergencia
```

$n = 1000$

```
In [ ]: n=1000
        H0 = np.identity(n)
        tau=(np.sqrt(n))*((epsilon)**(1/3))
        A=genera_A2(n)
        b=np.ones(n)
        x0=np.zeros(n)

        #definimos la funcion cuadratica, su gradiente y su Hessiana como inline functions
        cuadratica = lambda x: 0.5 * np.dot(x.T, np.dot(A, x)) - np.dot(b.T, x)
        D_cuadratica = lambda x: 0.5 * (np.dot(A.T, x) + np.dot(A, x)) - b


        argumentos_NT=[cuadratica,D_cuadratica,H0,x0,tau]
        imprime(cuadratica,n,x0,BFGS_modificado,argumentos_NT)
```

```
Dimensión n =  1000
f(x_0) =  0.0
Número de iteraciones =  10
f(x_k) =  -141.1999072003726
Primeras cuatro entradas de x_k=  [0.28238294 0.28238275 0.28238267 0.28238264]
Últimas cuatro entradas de x_k=  [0.28238264 0.28238267 0.28238275 0.28238294]
Norma del gradiente ||gk||   =  0.5463587122650305
Sí se cumplio el criterio de convergencia
```

Función de Beale

```
In [ ]: x0=(2,3)
        n=2
        H0 = np.identity(n)
        tau=(np.sqrt(n))*((epsilon)**(1/3))
        argumentos_NT=[fn.Beale,fn.D_Beale, H0,x0,tau]
        imprime(fn.Beale,n,x0,BFGS_modificado,argumentos_NT)
```

```
Dimensión n =  2
f(x_0) =  3347.203125
Número de iteraciones =  345
f(x_k) =  9.804507494945847e-06
Primeras cuatro entradas de x_k=  [2.99231847 0.49819972]
Últimas cuatro entradas de x_k=  [2.99231847 0.49819972]
Norma del gradiente ||gk||   =  0.0062877758164408
Sí se cumplio el criterio de convergencia
```

Función de Himmelblau

```
In [ ]: x0=(2,4)
        n=len(x0)
        tau=(np.sqrt(n))*((epsilon)**(1/3))
        argumentos_NT=[fn.Himmelblau,fn.D_Himmelblau, H0,x0,tau]
        imprime(fn.Himmelblau,n,x0,BFGS_modificado,argumentos_NT)
```

```
Dimensión n =  2
f(x_0) =  130
Número de iteraciones =  11
f(x_k) =  1.0129025496381528e-09
Primeras cuatro entradas de x_k=  [-3.7793138  -3.28318989]
Últimas cuatro entradas de x_k=  [-3.7793138  -3.28318989]
Norma del gradiente ||gk||   =  0.0003886149322926008
Sí se cumplio el criterio de convergencia
```

Función de Rosenbrock

```
In [ ]: x0=(-1.2,1.0)
        n=len(x0)
        H0 = np.identity(n)
        tau=(np.sqrt(n))*((epsilon)**(1/3))
        argumentos_NT=[fn.Rosenbrock,fn.D_Rosenbrock, H0,x0,tau]
        imprime(fn.Rosenbrock,n,x0,BFGS_modificado,argumentos_NT)
```

```
Dimensión n =  2
f(x_0) =  24.199999999999996
Número de iteraciones =  1187
f(x_k) =  8.716636762728045e-07
Primeras cuatro entradas de x_k=  [1.00092874 1.00186788]
Últimas cuatro entradas de x_k=  [1.00092874 1.00186788]
Norma del gradiente ||gk||   =  0.002739083597746638
Sí se cumplio el criterio de convergencia
```

```
In [ ]: x0=(-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0)
        n=len(x0)
        H0 = np.identity(n)
        tau=(np.sqrt(n))*((epsilon)**(1/3))
        argumentos_NT=[fn.Rosenbrock,fn.D_Rosenbrock, H0,x0,tau]
        imprime(fn.Rosenbrock,n,x0,BFGS_modificado,argumentos_NT)
```

```
Dimensión n =  20
f(x_0) =  4597.999999999999
Número de iteraciones =  17
f(x_k) =  403.3903607365128
Primeras cuatro entradas de x_k=  [-0.57248447 -0.34107472  0.28432646 -0.34199132]
Últimas cuatro entradas de x_k=  [ 0.28432647 -0.34199132  0.28475586  1.37373821]
Norma del gradiente ||gk||   =  449.3534243328402
Sí se cumplio el criterio de convergencia
```

```
In [ ]: x0=(-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0,-1.2,1.0,-
        n=len(x0)
        H0 = np.identity(n)
        tau=(np.sqrt(n))*((epsilon)**(1/3))
        argumentos_NT=[fn.Rosenbrock,fn.D_Rosenbrock, H0,x0,tau]
        imprime(fn.Rosenbrock,n,x0,BFGS_modificado,argumentos_NT)
```

```
Dimensión n =  40
f(x_0) =  9680.0
Número de iteraciones =  13
f(x_k) =  634.2938734073682
Primeras cuatro entradas de x_k=  [-0.5850408  -0.35316475  0.27369441 -0.35339376]
Últimas cuatro entradas de x_k=  [ 0.27369441 -0.35339376  0.2738004   1.36460289]
Norma del gradiente ||gk||   =  552.0669040583429
Sí se cumplio el criterio de convergencia
```

No encontró el mínimo para Rosenbrock n=20,40, llegó a un punto que no es el mínimo.