OPTIMIZACIÓN: TAREA 8

EZAU FARIDH TORRES TORRES.

Maestría en Ciencias con Orientación en Matemáticas Aplicadas.

CENTRO DE INVESTIGACIÓN EN MATEMÁTICAS.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style = "dark")

def BACKTRAKING(alpha_i: float, p: float, c: float,
                xk: np.array, f, fxk: np.array,
                gradfxk: np.array, pk: np.array, Nb: int):
    alpha = alpha_i
    for i in range(Nb):
        if f(xk + alpha*pk) <= fxk + c*alpha*(gradfxk.T)@pk:
            return alpha, i
        alpha = p*alpha
    return alpha, Nb

def f_Himmelblau(x: np.array):
    return (x[0]**2 + x[1] - 11)**2 + (x[0] + x[1]**2 - 7)**2
def grad_Himmelblau(x: np.array):
    x1 = 4*x[0]*(x[0]**2 + x[1] - 11) + 2*(x[0] + x[1]**2 - 7)
    x2 = 2*(x[0]**2 + x[1] - 11) + 4*x[1]*(x[0] + x[1]**2 - 7)
    return np.array([x1,x2], dtype = float)
def Hess_Himmelblau(x: np.array):
    x11 = 12*x[0]**2 + 4*x[1] - 42
    x12 = 4*x[0] + 4*x[1]
    x22 = 4*x[0] + 12*x[1]**2 - 26
    return np.array([[x11, x12], [x12, x22]], dtype = float)

def f_Beale(x: np.array):
    return (1.5 - x[0] + x[0]*x[1])**2 + (2.25 - x[0] + x[0]*x[1]**2)**2 + (
def grad_Beale(x: np.array):
    x1 = 2*(x[1] - 1)*(1.5 - x[0] + x[0]*x[1]) + 2*(x[1]**2 - 1)*(2.25 - x[0
    x2 = 2*x[0]*(1.5 - x[0] + x[0]*x[1]) + 4*x[0]*x[1]*(2.25 - x[0] + x[0]*x
    return np.array([x1,x2], dtype = float)
def Hess_Beale(x: np.array):
    x11 = 2*(x[1]**3 - 1)**2 + 2*(x[1]**2 - 1)**2 + 2*(x[1] - 1)**2
    x12 = 4*x[0]*x[1]*(x[1]**2 - 1) + 4*x[1]*(x[0]*x[1]**2 - x[0]+2.25) + 6*
    x22 = 18*x[0]**2*x[1]**4 + 8*x[0]**2*x[1]**2 + 2*x[0]**2 + 12*x[0]*x[1]*
    return np.array([[x11, x12], [x12, x22]], dtype = float)
```

```python
def f_Rosenbrock(x: np.array):
    n = len(x)
    s = 0
    for i in range(n-1):
        s = s + 100*(x[i+1] - x[i]**2)**2 + (1 - x[i])**2
    return s
def grad_Rosenbrock(x: np.array):
    n = len(x)
    grad = np.zeros(n)
    grad[0] = -400*x[0]*(x[1] - x[0]**2) - 2*(1-x[0])
    grad[n-1] = 200*(x[n-1] - x[n-2]**2)
    for j in range(1,n-1):
        grad[j] = 200*(x[j]-x[j-1]**2) - 400*x[j]*(x[j+1] - x[j]**2) - 2*(1-
    return np.array(grad, dtype = float)
def Hess_Rosenbrock(x: np.array):
    n = len(x)
    Hess = np.zeros((n,n))
    Hess[0,0] = -400*(x[1]-x[0]**2) + 800*x[0]**2 + 2
    Hess[1,0] = -400*x[0]
    Hess[n-2,n-1] = -400*x[n-2]
    Hess[n-1,n-1] = 200
    for j in range(1,n-1):
        Hess[j-1,j] = -400*x[j-1]
        Hess[j,j] = -400*(x[j+1]-x[j]**2) +800*x[j]**2 + 202
        Hess[j+1,j] = -400*x[j]
    return np.array(Hess, dtype = float)
```

1.- EJERCICIO 1:

Programar el método de BFGS modificado descrito en el Algoritmo 2 de la Clase 23.

1.1.

Programe la función que implementa el algoritmo. Debe recibir como parámetros

- El punto inicial $\mathbf{x}_0$
- La matriz $\mathbf{H}_0$
- La función f
- El gradiente $\nabla f(\mathbf{x})$
- La tolerancia τ
- El número máximo de iteraciones N
- Los paramétros ρ, c_1, N_b del algoritmo de backtracking

In [ ]:

```python
def BFGS_MOD(f, gradf, xk: np.array, tol: float, Hk: np.array,
             N: int, alpha_i, p: float, c: float, Nb: int):
    """
    BFGS METHOD WITH MODIFICATION FOR THE HESSIAN MATRIX.

    Args:
    - f:       function to minimize.
    - gradf:   gradient of the function.
    - xk:      initial point.
    - tol:     tolerance.
    - Hk:      initial Hessian matrix.
    - N:       maximum number of iterations.
    - alpha_i: initial step size.
    - p:       reduction factor for the step size.
    - c:       constant for the Armijo condition.
    - Nb:      maximum number of iterations for the backtracking line search

    Returns:
    - xk:  optimal point.
    - gk:  gradient at the optimal point.
    - k:   number of iterations.
    - T/F: if the method converged.
    """
    n = len(xk)
    for k in range(N-1):
        gk = gradf(xk)
        if np.linalg.norm(gk) < tol:
            return xk, gk, k, True
        pk = - Hk @ gk
        if pk.T @ gk > 0:
            lb1 = 10**(-5) + (pk.T @ gk)/(gk.T @ gk)
            Hk = Hk + lb1*np.eye(n)
            pk = pk - lb1*gk
        ak = BACKTRAKING(alpha_i = alpha_i, p = p, c = c, xk = xk, f = f,
                         fxk = f(xk), gradfxk = gk, pk = pk, Nb = Nb)[0]
        xk_n = xk + ak * pk
        gk_n = gradf(xk_n)
        sk = xk_n - xk
        yk = gk_n - gk
        if yk.T @ sk <= 0:
            lb2 = 10**(-5) - (yk.T @ sk)/(yk.T @ yk)
            Hk = Hk + lb2*np.eye(n)
        else:
            rhok = 1/(yk.T @ sk)
            Hk = (np.eye(n) - rhok*np.outer(sk,yk)) @ Hk @ (np.eye(n) - rhok
        xk = xk_n
    return xk, gk, N, False
```

1.2.

Pruebe el algoritmo para minimizar las siguientes funciones usando los parámetros

$N = 5000$, $\tau = \sqrt{n}\epsilon_m^{1/3}$, donde n es la dimensión de la variable $\mathbf{x}$, $\mathbf{H}_0$ como la matriz identidad y ϵ_m es el épsilon máquina. Para backtracking use $\rho = 0.5$, $c_1 = 0.001$ y el número máximo de iteraciones $N_b = 500$.

En cada caso imprima los siguientes datos:

- la dimensión n,
- $f(\mathbf{x}_0)$,
- el número k de iteraciones realizadas,
- $f(\mathbf{x}_k)$,
- las primeras y últimas 4 entradas del punto $\mathbf{x}_k$ que devuelve el algoritmo,
- la norma del vector gradiente $\mathbf{g}_k$,
- la variable res que indica si el algoritmo terminó porque se cumplió el criterio de la tolerancia o terminó por iteraciones.

In [ ]:
```
eps = np.finfo(float).eps
N = 5000
p = 0.5
c = 0.001
Nb = 500
alpha_i = 1
```

Función de cuadrática 1: Para $\mathbf{x} = (x_1, x_2, \ldots, x_n)$

- $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}_1 \mathbf{x} - \mathbf{b}_1^\top \mathbf{x}$, donde $\mathbf{A}_1$ y $\mathbf{b}_1$ están definidas por

$$\mathbf{A}_1 = n\mathbf{I} + \mathbf{1} = \begin{bmatrix} n & 0 & \cdots & 0 \\ 0 & n & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & n \end{bmatrix} + \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}, \qquad \mathbf{b}_1 = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix},$$

donde $\mathbf{I}$ es la matriz identidad y $\mathbf{1}$ es la matriz llena de 1's, ambas de tamaño n, usando los puntos iniciales

- $\mathbf{x}_0 = (0, \ldots, 0) \in \mathbb{R}^{10}$
- $\mathbf{x}_0 = (0, \ldots, 0) \in \mathbb{R}^{100}$
- $\mathbf{x}_0 = (0, \ldots, 0) \in \mathbb{R}^{1000}$

In [ ]:
```
n = 10
A1 = n*np.eye(n, dtype = float) + np.ones([n,n], dtype = float)
b1 = np.ones(n, dtype = float)
f_cuad = lambda x: 0.5 * x.T @ A1 @ x - b1.T @ x
gradf_cuad = lambda x: A1 @ x - b1
x0 = np.zeros(n, dtype = float)
```

```
tol = np.sqrt(n)*eps**(1/3)
H0 = np.eye(n)
xk, gk, k, RES = BFGS_MOD(f = f_cuad, gradf = gradf_cuad, xk = x0, tol = tol
                          N = N, alpha_i = alpha_i, p = p, c = c, Nb = Nb)
print("DIMENSION:   ", n)
print("f(x0):       ", f_cuad(x0))
print("ITERACIONES: ", k)
print("f(xk):       ", f_cuad(xk))
print("xk:          ", xk[:4], "...", xk[-4:])
print("||gk||:      ", np.linalg.norm(gk))
print("CONVERGENCIA:", RES)
```

```
DIMENSION:    10
f(x0):        0.0
ITERACIONES:  2
f(xk):        -0.25
xk:           [0.05 0.05 0.05 0.05] ... [0.05 0.05 0.05 0.05]
||gk||:       1.041481514324134e-15
CONVERGENCIA: True
```

In [ ]:
```
n = 100
A1 = n*np.eye(n, dtype = float) + np.ones([n,n], dtype = float)
b1 = np.ones(n, dtype = float)
f_cuad = lambda x: 0.5 * x.T @ A1 @ x - b1.T @ x
gradf_cuad = lambda x: A1 @ x - b1
x0 = np.zeros(n, dtype = float)
tol = np.sqrt(n)*eps**(1/3)
H0 = np.eye(n)
xk, gk, k, RES = BFGS_MOD(f = f_cuad, gradf = gradf_cuad, xk = x0, tol = tol
                          N = N, alpha_i = alpha_i, p = p, c = c, Nb = Nb)
print("DIMENSION:   ", n)
print("f(x0):       ", f_cuad(x0))
print("ITERACIONES: ", k)
print("f(xk):       ", f_cuad(xk))
print("xk:          ", xk[:4], "...", xk[-4:])
print("||gk||:      ", np.linalg.norm(gk))
print("CONVERGENCIA:", RES)
```

```
DIMENSION:    100
f(x0):        0.0
ITERACIONES:  2
f(xk):        -0.2499999999999999
xk:           [0.005 0.005 0.005 0.005] ... [0.005 0.005 0.005 0.005]
||gk||:       3.225767447537171e-12
CONVERGENCIA: True
```

In [ ]:
```
n = 1000
A1 = n*np.eye(n, dtype = float) + np.ones([n,n], dtype = float)
b1 = np.ones(n, dtype = float)
f_cuad = lambda x: 0.5 * x.T @ A1 @ x - b1.T @ x
gradf_cuad = lambda x: A1 @ x - b1
x0 = np.zeros(n, dtype = float)
tol = np.sqrt(n)*eps**(1/3)
H0 = np.eye(n)
```

```
xk, gk, k, RES = BFGS_MOD(f = f_cuad, gradf = gradf_cuad, xk = x0, tol = tol
                          N = N, alpha_i = alpha_i, p = p, c = c, Nb = Nb)
print("DIMENSION:   ", n)
print("f(x0):       ", f_cuad(x0))
print("ITERACIONES: ", k)
print("f(xk):       ", f_cuad(xk))
print("xk:          ", xk[:4], "...", xk[-4:])
print("||gk||:      ", np.linalg.norm(gk))
print("CONVERGENCIA:", RES)
```

```
DIMENSION:    1000
f(x0):        0.0
ITERACIONES:  2
f(xk):        -0.24999999999999845
xk:           [0.0005 0.0005 0.0005 0.0005] ... [0.0005 0.0005 0.0005 0.000
5]
||gk||:       2.902695606392848e-10
CONVERGENCIA: True
```

Función de cuadrática 2: Para $\mathbf{x} = (x_1, x_2, \ldots, x_n)$

- $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}_2 \mathbf{x} - \mathbf{b}_2^\top \mathbf{x}$, donde $\mathbf{A}_2 = [a_{ij}]$ y $\mathbf{b}_2$ están definidas por

$$a_{ij} = exp\left(-0.25(i-j)^2\right), \qquad \mathbf{b}_2 = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

usando los puntos iniciales:

- $\mathbf{x}_0 = (0, \ldots, 0) \in \mathbb{R}^{10}$
- $\mathbf{x}_0 = (0, \ldots, 0) \in \mathbb{R}^{100}$
- $\mathbf{x}_0 = (0, \ldots, 0) \in \mathbb{R}^{1000}$

In [ ]:
```
n = 10
A2 = np.zeros((n,n), dtype = float)
for i in range(n):
    for j in range(n):
        A2[i,j] = np.exp(-0.25*(i-j)**2)
b2 = np.ones(n, dtype = float)
f_cuad = lambda x: 0.5 * x.T @ A2 @ x - b2.T @ x
gradf_cuad = lambda x: A2 @ x - b2
x0 = np.zeros(n, dtype = float)
tol = np.sqrt(n)*eps**(1/3)
H0 = np.eye(n)
xk, gk, k, RES = BFGS_MOD(f = f_cuad, gradf = gradf_cuad, xk = x0, tol = tol
                          N = N, alpha_i = alpha_i, p = p, c = c, Nb = Nb)
print("DIMENSION:   ", n)
print("f(x0):       ", f_cuad(x0))
```

```
print("ITERACIONES: ", k)
print("f(xk):       ", f_cuad(xk))
print("xk:          ", xk[:4], "...", xk[-4:])
print("||gk||:      ", np.linalg.norm(gk))
print("CONVERGENCIA:", RES)
```

```
DIMENSION:    10
f(x0):        0.0
ITERACIONES:  18
f(xk):        -1.7934208025210774
xk:           [ 1.36910165 -1.16637731  1.60908339 -0.61339229] ... [-0.6133
9229  1.60908339 -1.16637731  1.36910165]
||gk||:       3.6453605641428233e-06
CONVERGENCIA: True
```

In [ ]:
```
n = 100
A2 = np.zeros((n,n), dtype = float)
for i in range(n):
    for j in range(n):
        A2[i,j] = np.exp(-0.25*(i-j)**2)
b2 = np.ones(n, dtype = float)
f_cuad = lambda x: 0.5 * x.T @ A2 @ x - b2.T @ x
gradf_cuad = lambda x: A2 @ x - b2
x0 = np.zeros(n, dtype = float)
tol = np.sqrt(n)*eps**(1/3)
H0 = np.eye(n)
xk, gk, k, RES = BFGS_MOD(f = f_cuad, gradf = gradf_cuad, xk = x0, tol = tol
                          N = N, alpha_i = alpha_i, p = p, c = c, Nb = Nb)
print("DIMENSION:   ", n)
print("f(x0):       ", f_cuad(x0))
print("ITERACIONES: ", k)
print("f(xk):       ", f_cuad(xk))
print("xk:          ", xk[:4], "...", xk[-4:])
print("||gk||:      ", np.linalg.norm(gk))
print("CONVERGENCIA:", RES)
```

```
DIMENSION:    100
f(x0):        0.0
ITERACIONES:  139
f(xk):        -14.494330069283096
xk:           [ 1.44628123 -1.41633442  2.11047122 -1.42499584] ... [-1.4249
928   2.11047032 -1.41633792  1.44628086]
||gk||:       5.05688281168634e-05
CONVERGENCIA: True
```

In [ ]:
```
n = 1000
A2 = np.zeros((n,n), dtype = float)
for i in range(n):
    for j in range(n):
        A2[i,j] = np.exp(-0.25*(i-j)**2)
b2 = np.ones(n, dtype = float)
f_cuad = lambda x: 0.5 * x.T @ A2 @ x - b2.T @ x
gradf_cuad = lambda x: A2 @ x - b2
x0 = np.zeros(n, dtype = float)
```

```
tol = np.sqrt(n)*eps**(1/3)
H0 = np.eye(n)
xk, gk, k, RES = BFGS_MOD(f = f_cuad, gradf = gradf_cuad, xk = x0, tol = tol
                          N = N, alpha_i = alpha_i, p = p, c = c, Nb = Nb)
print("DIMENSION:   ", n)
print("f(x0):       ", f_cuad(x0))
print("ITERACIONES: ", k)
print("f(xk):       ", f_cuad(xk))
print("xk:          ", xk[:4], "...", xk[-4:])
print("||gk||:      ", np.linalg.norm(gk))
print("CONVERGENCIA:", RES)
```

```
DIMENSION:    1000
f(x0):        0.0
ITERACIONES:  257
f(xk):        -141.43698680561425
xk:           [ 1.44628094 -1.41635864  2.11049968 -1.42504235] ... [-1.4250
4235  2.11049968 -1.41635864  1.44628094]
||gk||:       0.00018789741226501192
CONVERGENCIA: True
```

Función de Beale : Para $\mathbf{x} = (x_1, x_2)$

$$f(\mathbf{x}) = (1.5 - x_1 + x_1 x_2)^2 + (2.25 - x_1 + x_1 x_2^2)^2 + (2.625 - x_1 + x_1 x_2^3)^2.$$

- $\mathbf{x}_0 = (2, 3)$

In [ ]:
```
x0 = np.array([2,3], dtype = float)
n = len(x0)
H0 = np.eye(n)
tol = np.sqrt(n)*eps**(1/3)
xk, gk, k, RES = BFGS_MOD(f = f_Beale, gradf = grad_Beale, xk = x0, tol = to
                          N = N, alpha_i = alpha_i, p = p, c = c, Nb = Nb)
print("DIMENSION:   ", n)
print("f(x0):       ", f_Beale(x0))
print("ITERACIONES: ", k)
print("f(xk):       ", f_Beale(xk))
print("xk:          ", xk)
print("||gk||:      ", np.linalg.norm(gk))
print("CONVERGENCIA:", RES)
```

```
/var/folders/_h/2wf1t3v96t99m5n9pzmlm9pm0000gn/T/ipykernel_1216/2848653580.p
y:41: RuntimeWarning: invalid value encountered in scalar divide
  lb2 = 10**(-5) - (yk.T @ sk)/(yk.T @ yk)
DIMENSION:    2
f(x0):        3347.203125
ITERACIONES:  5000
f(xk):        nan
xk:           [nan nan]
||gk||:       nan
CONVERGENCIA: False
```

Función de Himmelblau: Para $\mathbf{x} = (x_1, x_2)$

$$f(\mathbf{x}) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2.$$

- $\mathbf{x}_0 = (2, 4)$

In [ ]:
```
x0 = np.array([2,4], dtype = float)
n = len(x0)
H0 = np.eye(n)
tol = np.sqrt(n)*eps**(1/3)
xk, gk, k, RES = BFGS_MOD(f = f_Himmelblau, gradf = grad_Himmelblau, xk = x0
                          Hk = H0, N = N, alpha_i = alpha_i, p = p, c = c, N
print("DIMENSION:   ", n)
print("f(x0):       ", f_Himmelblau(x0))
print("ITERACIONES: ", k)
print("f(xk):       ", f_Himmelblau(xk))
print("xk:          ", xk)
print("||gk||:      ", np.linalg.norm(gk))
print("CONVERGENCIA:", RES)
```

```
DIMENSION:    2
f(x0):        130.0
ITERACIONES:  10
f(xk):        9.834452856641356e-16
xk:           [ 3.58442834 -1.84812653]
||gk||:       3.8589289437020564e-07
CONVERGENCIA: True
```

Función de Rosenbrock: Para $\mathbf{x} = (x_1, x_2, \ldots, x_n)$

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2\right] \quad n \geq 2.$$

- $\mathbf{x}_0 = (-1.2, 1.0) \in \mathbb{R}^2$
- $\mathbf{x}_0 = (-1.2, 1.0, \ldots, -1.2, 1.0) \in \mathbb{R}^{20}$
- $\mathbf{x}_0 = (-1.2, 1.0, \ldots, -1.2, 1.0) \in \mathbb{R}^{40}$

In [ ]:
```
x0 = np.array([-1.2, 1.0], dtype = float)
n = len(x0)
H0 = np.eye(n)
tol = np.sqrt(n)*eps**(1/3)
xk, gk, k, RES = BFGS_MOD(f = f_Rosenbrock, gradf = grad_Rosenbrock, xk = x0
                          Hk = H0, N = N, alpha_i = alpha_i, p = p, c = c, N
print("DIMENSION:   ", n)
print("f(x0):       ", f_Rosenbrock(x0))
print("ITERACIONES: ", k)
print("f(xk):       ", f_Rosenbrock(xk))
print("xk:          ", xk)
```

```
print("||gk||:      ", np.linalg.norm(gk))
print("CONVERGENCIA:", RES)
```

```
DIMENSION:    2
f(x0):        24.199999999999996
ITERACIONES:  34
f(xk):        2.745636868826416e-17
xk:           [1.         0.99999999]
||gk||:       8.834628308482201e-08
CONVERGENCIA: True
```

In [ ]:
```
x0 = np.array([-1.2, 1.0, -1.2, 1.0, -1.2, 1.0, -1.2, 1.0, -1.2, 1.0, -1.2,
n = len(x0)
H0 = np.eye(n)
tol = np.sqrt(n)*eps**(1/3)
xk, gk, k, RES = BFGS_MOD(f = f_Rosenbrock, gradf = grad_Rosenbrock, xk = x0
                          Hk = H0, N = N, alpha_i = alpha_i, p = p, c = c, N
print("DIMENSION:   ", n)
print("f(x0):       ", f_Rosenbrock(x0))
print("ITERACIONES: ", k)
print("f(xk):       ", f_Rosenbrock(xk))
print("xk:          ", xk[:4], "...", xk[-4:])
print("||gk||:      ", np.linalg.norm(gk))
print("CONVERGENCIA:", RES)
```

```
DIMENSION:    20
f(x0):        4598.000000000001
ITERACIONES:  128
f(xk):        2.304160307156283e-14
xk:           [1. 1. 1. 1.] ... [0.99999999 0.99999997 0.99999995 0.9999999
1]
||gk||:       5.657759584151205e-06
CONVERGENCIA: True
```

In [ ]:
```
x0 = np.array([-1.2, 1.0, -1.2, 1.0, -1.2, 1.0, -1.2, 1.0, -1.2, 1.0, -1.2,
n = len(x0)
H0 = np.eye(n)
tol = np.sqrt(n)*eps**(1/3)
xk, gk, k, RES = BFGS_MOD(f = f_Rosenbrock, gradf = grad_Rosenbrock, xk = x0
                          Hk = H0, N = N, alpha_i = alpha_i, p = p, c = c, N
print("DIMENSION:   ", n)
print("f(x0):       ", f_Rosenbrock(x0))
print("ITERACIONES: ", k)
print("f(xk):       ", f_Rosenbrock(xk))
print("xk:          ", xk[:4], "...", xk[-4:])
print("||gk||:      ", np.linalg.norm(gk))
print("CONVERGENCIA:", RES)
```

```
DIMENSION:    40
f(x0):        9680.000000000002
ITERACIONES:  223
f(xk):        4.085288272310319e-13
xk:           [1.00000001 1.00000001 1.         1.        ] ... [1.00000001
1.00000004 1.0000001  1.0000002 ]
||gk||:       2.9394848599856312e-05
CONVERGENCIA: True
```