OPTIMIZACIÓN: TAREA 7

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 Newton truncado descrito en el Algoritmo 1 y 2 de la Clase 20.

1.1.

Programar la función que implementa el Algoritmo 1, que calcula una aproximación de la solución del sistema de Newton.

- Haga que la función devuelva la dirección $\mathbf{p}_k$ y el número de iteraciones realizadas.

In [ ]:
```python
def CG(gk: np.array, Hk: np.array, n: int, tol: float):
    """
    CONJUGATE GRADIENT APPLIED TO NEWTON SYSTEM.

    Args:
    - gk:  gradient of f at xk.
    - Hk:  Hessian of f at xk.
```

```
    - n:   dimention.
    - tol: tolerance.

    Outputs:
    - pk: search direction.
    - j:  number of iterations.
    """
    zj = 0
    rj = gk
    dj = -rj
    for j in range(n-1):
        if dj.T @ Hk @ dj <= 0:
            if j == 0:
                pk = -gk
            else:
                pk = zj
        aj = (rj.T @ rj) / (dj.T @ Hk @ dj)
        zj = zj + aj * dj
        rj_n = rj + aj * Hk @ dj
        Bj = (rj_n.T @ rj_n) / (rj.T @ rj)
        dj = -rj_n + Bj * dj
        if np.linalg.norm(rj_n) < tol:
            pk = zj
            return pk, j
        rj = rj_n
    pk = zj
    return pk, n
```

1.2.

Programar la función que implementa el Algoritmo 2.

- Use el algoritmo de backtracking con la condición de descenso suficiente para calcular el tamaño de paso α_k.
- Defina la variable binaria res de modo que $True$ si se cumple la condición de salida $\|\mathbf{g}_k\| < \tau$ y $False$ si termina por iteraciones.
- Calcule el promedio de las iteraciones realizadas por el Algoritmo 1
- Haga que la función devuelva $\mathbf{x}_k, \mathbf{g}_k, k, res$ y el promedio de la iteraciones realizadas por el Algoritmo 1.

In [ ]:
```
def NEWTON_CG_LINESEARCH_METHOD(f, gradf, Hessf, xk: np.array, tol: float, m
                                alpha_i: float, p: float, c: float, Nb: int)
    """
    NEWTON CONJUGATE GRADIENT METHOD WITH LINE SEARCH.

    Args:
    - f:       function to minimize.
    - gradf:   gradient of f.
    - Hessf:   Hessian of f.
```

```
    - xk:      initial point.
    - tol:     tolerance.
    - maxiter: maximum number of iterations.
    - alpha_i: initial alpha.
    - p:       reduction factor.
    - c:       constant for Armijo condition.
    - Nb:      maximum number of iterations for line search.

    Outputs:
    - xk:   optimal point.
    - gk:   gradient at xk.
    - k:    number of iterations.
    - T/F:  if the method converged.
    - MEAN: average number of iterations for line search.
    """
    n = len(xk)
    iteraciones = []
    for k in range(maxiter):
        gk = gradf(xk)
        if np.linalg.norm(gk) < tol:
            return xk, gk, k, True, np.mean(iteraciones)
        Hk = Hessf(xk)
        ek = min(0.5, np.sqrt(np.linalg.norm(gk)))*np.sqrt(np.linalg.norm(gk
        pk, i = CG(gk, Hk, n, ek)
        ak = BACKTRAKING(alpha_i = alpha_i, p = p, c = c, xk = xk, f = f,
                         fxk=f(xk), gradfxk = gk, pk = pk, Nb = Nb)[0]
        iteraciones.append(i)
        xk = xk + ak * pk
    return xk, gk, maxiter, False, np.mean(iteraciones)
```

1.3.

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 $\mathbf{x}$ 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 $\mathbf{g}_k$,
- el promedio del número de iteraciones realizadas por el Algoritmo 1.
- la variable res para saber si el algoritmo puedo converger.

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

Función 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
Hessf_cuad = lambda x: A1
x0 = np.zeros(n, dtype = float)
tol = np.sqrt(n)*eps**(1/3)
xk, gk, k, RES, MEAN = NEWTON_CG_LINESEARCH_METHOD(f = f_cuad, gradf = gradf
                                                  Hessf = Hessf_cuad, xk = 
                                                  maxiter = N, alpha_i = al
                                                  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("PROMEDIO:    ", MEAN)
print("CONVERGENCIA:", RES)
```

```
DIMENSION:     10
f(x0):         0.0
ITERACIONES:   1
f(xk):         -0.24999999999999994
xk:            [0.05 0.05 0.05 0.05] ... [0.05 0.05 0.05 0.05]
||gk||:        6.280369834735101e-16
PROMEDIO:      0.0
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
Hessf_cuad = lambda x: A1
x0 = np.zeros(n, dtype = float)
tol = np.sqrt(n)*eps**(1/3)
xk, gk, k, RES, MEAN = NEWTON_CG_LINESEARCH_METHOD(f = f_cuad, gradf = gradf
                                                   Hessf = Hessf_cuad, xk = 
                                                   maxiter = N, alpha_i = al
                                                   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("PROMEDIO:     ", MEAN)
print("CONVERGENCIA:", RES)
```

```
DIMENSION:     100
f(x0):         0.0
ITERACIONES:   1
f(xk):         -0.25000000000000006
xk:            [0.005 0.005 0.005 0.005] ... [0.005 0.005 0.005 0.005]
||gk||:        7.691850745534255e-16
PROMEDIO:      0.0
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
Hessf_cuad = lambda x: A1
x0 = np.zeros(n, dtype = float)
tol = np.sqrt(n)*eps**(1/3)
xk, gk, k, RES, MEAN = NEWTON_CG_LINESEARCH_METHOD(f = f_cuad, gradf = gradf
                                                   Hessf = Hessf_cuad, xk = 
                                                   maxiter = N, alpha_i = al
                                                   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("PROMEDIO:      ", MEAN)
print("CONVERGENCIA:", RES)
```

```
DIMENSION:     1000
f(x0):         0.0
ITERACIONES:   1
f(xk):         -0.2500000000000049
xk:            [0.0005 0.0005 0.0005 0.0005] ... [0.0005 0.0005 0.0005 0.000
5]
||gk||:        1.192009396658756e-13
PROMEDIO:      0.0
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
Hessf_cuad = lambda x: A2
x0 = np.zeros(n, dtype = float)
tol = np.sqrt(n)*eps**(1/3)
xk, gk, k, RES, MEAN = NEWTON_CG_LINESEARCH_METHOD(f = f_cuad, gradf = gradf
                                                   Hessf = Hessf_cuad, xk =
                                                   maxiter = N, alpha_i = al
                                                   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("PROMEDIO:    ", MEAN)
print("CONVERGENCIA:", RES)
```

```
DIMENSION:    10
f(x0):        0.0
ITERACIONES:  22
f(xk):        -1.7934208015015511
xk:           [ 1.36906831 -1.16629192  1.6089342  -0.61322014] ... [-0.6132
2014  1.6089342  -1.16629192  1.36906831]
||gk||:       1.5307316549914036e-05
PROMEDIO:     0.3181818181818182
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
Hessf_cuad = lambda x: A2
x0 = np.zeros(n, dtype = float)
tol = np.sqrt(n)*eps**(1/3)
xk, gk, k, RES, MEAN = NEWTON_CG_LINESEARCH_METHOD(f = f_cuad, gradf = gradf
                                                  Hessf = Hessf_cuad, xk = 
                                                  maxiter = N, alpha_i = al
                                                  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("PROMEDIO:    ", MEAN)
print("CONVERGENCIA:", RES)
```

```
DIMENSION:    100
f(x0):        0.0
ITERACIONES:  97
f(xk):        -14.494331024801777
xk:           [ 1.44646083 -1.41684769  2.11130648 -1.42597653] ... [-1.4259
7732  2.11130208 -1.41684881  1.44646501]
||gk||:       6.0175662526390146e-05
PROMEDIO:     2.3402061855670104
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
Hessf_cuad = lambda x: A2
x0 = np.zeros(n, dtype = float)
tol = np.sqrt(n)*eps**(1/3)
xk, gk, k, RES, MEAN = NEWTON_CG_LINESEARCH_METHOD(f = f_cuad, gradf = gradf
                                                   Hessf = Hessf_cuad, xk =
                                                   maxiter = N, alpha_i = al
                                                   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("PROMEDIO:    ", MEAN)
print("CONVERGENCIA:", RES)
```

```
DIMENSION:    1000
f(x0):        0.0
ITERACIONES:  246
f(xk):        -141.43698480430896
xk:           [ 1.4468278  -1.41800279  2.11341523 -1.42884304] ... [-1.4288
3692  2.11339703 -1.41801106  1.44684836]
||gk||:       0.0001753439970808574
PROMEDIO:     0.4715447154471545
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)
tol = np.sqrt(n)*eps**(1/3)
xk, gk, k, RES, MEAN = NEWTON_CG_LINESEARCH_METHOD(f = f_Beale, gradf = grad
                                                   Hessf = Hess_Beale, xk =
                                                   maxiter = N, alpha_i = al
                                                   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("PROMEDIO:    ", MEAN)
print("CONVERGENCIA:", RES)
```

```
DIMENSION:     2
f(x0):         3347.203125
ITERACIONES:   303
f(xk):         1.0406054905540409e-10
xk:            [2.99997453 0.49999359]
||gk||:        8.270842722018332e-06
PROMEDIO:      1.0165016501650166
CONVERGENCIA:  True
```

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)
tol = np.sqrt(n)*eps**(1/3)
xk, gk, k, RES, MEAN = NEWTON_CG_LINESEARCH_METHOD(f = f_Himmelblau, gradf =
                                                   Hessf = Hess_Himmelblau,
                                                   maxiter = N, alpha_i = al
                                                   c = c, Nb = Nb)
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("PROMEDIO:     ", MEAN)
print("CONVERGENCIA:", RES)
```

```
DIMENSION:     2
f(x0):         130.0
ITERACIONES:   19
f(xk):         5.260828354624073e-13
xk:            [2.99999987 2.00000009]
||gk||:        7.77271199768923e-06
PROMEDIO:      0.42105263157894735
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)
tol = np.sqrt(n)*eps**(1/3)
xk, gk, k, RES, MEAN = NEWTON_CG_LINESEARCH_METHOD(f = f_Rosenbrock, gradf =
                                                  Hessf = Hess_Rosenbrock,
                                                  maxiter = N, alpha_i = al
                                                  c = c, Nb = Nb)
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("PROMEDIO:    ", MEAN)
print("CONVERGENCIA:", RES)
```

```
DIMENSION:    2
f(x0):        24.199999999999996
ITERACIONES:  5000
f(xk):        2.4516036579601876
xk:           [-0.56382959  0.32567598]
||gk||:       2.075163968829345
PROMEDIO:     1.9996
CONVERGENCIA: False
```

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)
tol = np.sqrt(n)*eps**(1/3)
xk, gk, k, RES, MEAN = NEWTON_CG_LINESEARCH_METHOD(f = f_Rosenbrock, gradf =
                                                  Hessf = Hess_Rosenbrock,
                                                  maxiter = N, alpha_i = al
                                                  c = c, Nb = Nb)
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("PROMEDIO:    ", MEAN)
print("CONVERGENCIA:", RES)
```

```
DIMENSION:    20
f(x0):        4598.000000000001
ITERACIONES:  5000
f(xk):        19.4882350039715
xk:           [-0.61878307  0.39649438  0.16675802  0.03844237] ... [1.03156
251e-02 1.00512735e-02 1.01400139e-02 7.43921058e-06]
||gk||:       0.5140967920786855
PROMEDIO:     3.0002
CONVERGENCIA: False
```

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)
tol = np.sqrt(n)*eps**(1/3)
```

```python
xk, gk, k, RES, MEAN = NEWTON_CG_LINESEARCH_METHOD(f = f_Rosenbrock, gradf =
                                                    Hessf = Hess_Rosenbrock,
                                                    maxiter = N, alpha_i = al
                                                    c = c, Nb = Nb)
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("PROMEDIO:     ", MEAN)
print("CONVERGENCIA:", RES)
```

```
DIMENSION:     40
f(x0):         9680.000000000002
ITERACIONES:   5000
f(xk):         39.28658778440892
xk:            [-0.61806366  0.39558009  0.16620719  0.03840828] ... [0.01029
175 0.00991944 0.01042095 0.00019364]
||gk||:        0.5194618790851864
PROMEDIO:      3.0002
CONVERGENCIA: False
```

2.- EJERCICIO 2:

Programar las funciones que calcule el gradiente y la Hessiana usando el método de diferencias finitas.

2.1.

Programe la función que calcule una aproximación del gradiente de una función $f(\mathbf{x})$ en un punto $\mathbf{x} \in \mathbb{R}^n$ dado usando el esquema de diferencias finitas hacia adelante (Página 20 de la Clase 20).

- La función recibe como parámetros la función f, el punto $\mathbf{x}$ y el incremento h y devuelve el arreglo de tamaño n con las aproximaciones de aproximaciones de las derivadas parciales en el punto $\mathbf{x}$.

In [ ]:
```python
def GRADIENT(f, x: np.array, h: float):
    """
    COMPUTE THE GRADIENT OF A FUNCTION USING FORWARD FINITE DIFFERENCES.

    Args:
    - f: function to compute the gradient.
    - x: point to compute the gradient.
```

```
    - h: step size.

    Outputs:
    - grad: gradient of f at x.
    """
    n = len(x)
    grad = np.zeros(n)
    for i in range(n):
        ei = np.zeros(n)
        ei[i] = 1
        grad[i] = (f(x + h*ei) - f(x)) / h
    return grad
```

2.2.

Programe la función que calcule una aproximación de la Hessiana de una función $f(\mathbf{x})$ en un punto $\mathbf{x} \in \mathbb{R}^n$ dado usando el esquema de diferencias finitas de la Página 22 de la Clase 20.

- La función recibe como parámetros la función f, el punto $\mathbf{x}$ y el incremento h y devuelve una matriz simétrica de tamaño n que tiene las aproximaciones de las segundas derivadas parciales de f en el punto $\mathbf{x}$.

In [ ]:
```
def HESSIAN(f, x: np.array, h: float):
    """
    COMPUTE THE HESSIAN MATRIX OF A FUNCTION USING FINITE DIFFERENCES.

    Args:
    - f: function to compute the Hessian.
    - x: point to compute the Hessian.
    - h: step size.

    Outputs:
    - Hess: Hessian of f at x.
    """
    n = len(x)
    Hess = np.zeros((n,n))
    for i in range(n):
        ei = np.zeros(n)
        ei[i] = 1
        for j in range(n):
            ej = np.zeros(n)
            ej[j] = 1
            Hess[i,j] = (f(x + h*ei + h*ej) - f(x + h*ei) - f(x + h*ej) + f(
    return Hess
```

2.3.

Modifique la función `errorRelativo_grad` para reportar estadísticas del error relativo de la implementación del gradiente analítico `gradf` de una función respecto al gradiente calculado con `autograd`, para que mida el error relativo entre la función `gradf` y la aproximación del gradiente usando diferencias finitas. Hay que agregar como parámetro de `errorRelativo_grad` el incremento h para que se pueda llamar la función del Punto 1.

In [ ]:
```
def errorRelativo_grad(f, gradf, n: int, nt: int, h: float):
    """
    Print statistics of relative errors in the gradient calculation (analyti
    using finite differences.

    Args:
    - f:     function to find the gradient.
    - gradf: analytical gradient of f.
    - n:     dimension.
    - nt:    number of tests.
    - h:     step size.
    """
    ve = np.zeros(nt)
    gf = lambda x: GRADIENT(f = f, x = x, h = h) # Funcion gradiente generac
    for i in range(nt):
        x0  = np.random.randn(n)
        g0  = gradf(x0)
        ga  = gf(x0)
        ve[i] = np.linalg.norm(g0-ga) / np.linalg.norm(ga)
    print('ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h =', h)
    print('Min: %.2e   Media: %.2e    Max: %.2e' %(np.min(ve), np.mean(ve),
```

2.4.

Programar la función `errorRelativo_hess`, similar a la función del punto anterior, para que reporte estadísticas del error relativo entre una función que calcula la Hessiana de f de manera analítica en un punto $\mathbf{x}$ y la aproximación de la Hessiana en $\mathbf{x}$ usando diferencias finitas.

In [ ]:
```
def errorRelativo_hess(f, Hessf, n: int , nt: int, h: float):
    """
    Print statistics of relative errors in the Hessian matrix calculation (a
    using finite differences.

    Args:
    - f:     function to find the Hessian.
    - Hessf: analytical Hessian of f.
    - n:     dimension.
    - nt:    number of tests.
    - h:     step size.
    """
    ve = np.zeros(nt)
    Hf = lambda x: HESSIAN(f = f, x = x, h = h) # Funcion Hessiana generada
```

```
    for i in range(nt):
        x0  = np.random.randn(n)
        H0  = Hessf(x0)
        Ha  = Hf(x0)
        ve[i] = np.linalg.norm(H0-Ha) / np.linalg.norm(Ha)
    print('ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h =', h)
    print('Min: %.2e   Media: %.2e    Max: %.2e' %(np.min(ve), np.mean(ve),
```

2.5.

Pruebe las funciones `errorRelativo_grad` con cada una de las funciones del Ejercicio 1 usando $h = 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}$. ¿Cuál es el valor de h que conviene usar para aproximar el gradiente y cuál para aproximar la Hessiana?

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

$n = 10$

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
Hessf_cuad = lambda x: A1

errorRelativo_grad(f = f_cuad, gradf = gradf_cuad, n = n, nt = 50, h = 1e-5)
errorRelativo_grad(f = f_cuad, gradf = gradf_cuad, n = n, nt = 50, h = 1e-6)
errorRelativo_grad(f = f_cuad, gradf = gradf_cuad, n = n, nt = 50, h = 1e-7)
errorRelativo_grad(f = f_cuad, gradf = gradf_cuad, n = n, nt = 50, h = 1e-8)
```

```
ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-05
Min: 2.90e-06   Media: 5.08e-06    Max: 9.30e-06

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-06
Min: 2.74e-07   Media: 5.28e-07    Max: 9.88e-07

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-07
Min: 1.60e-08   Media: 5.36e-08    Max: 1.20e-07

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-08
Min: 1.39e-08   Media: 6.34e-08    Max: 2.39e-07
```

In [ ]:
```
errorRelativo_hess(f = f_cuad, Hessf = Hessf_cuad, n = n, nt = 50, h = 1e-5)
errorRelativo_hess(f = f_cuad, Hessf = Hessf_cuad, n = n, nt = 50, h = 1e-6)
errorRelativo_hess(f = f_cuad, Hessf = Hessf_cuad, n = n, nt = 50, h = 1e-7)
errorRelativo_hess(f = f_cuad, Hessf = Hessf_cuad, n = n, nt = 50, h = 1e-8)
```

```
ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-05
Min: 8.30e-06   Media: 3.39e-05    Max: 1.07e-04

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-06
Min: 1.17e-03   Media: 3.54e-03    Max: 1.04e-02

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-07
Min: 6.58e-02   Media: 3.22e-01    Max: 1.06e+00

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-08
Min: 9.75e-01   Media: 1.01e+00    Max: 1.04e+00
```

$n = 100$

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
Hessf_cuad = lambda x: A1
errorRelativo_grad(f = f_cuad, gradf = gradf_cuad, n = n, nt = 50, h = 1e-5)
errorRelativo_grad(f = f_cuad, gradf = gradf_cuad, n = n, nt = 50, h = 1e-6)
errorRelativo_grad(f = f_cuad, gradf = gradf_cuad, n = n, nt = 50, h = 1e-7)
errorRelativo_grad(f = f_cuad, gradf = gradf_cuad, n = n, nt = 50, h = 1e-8)
```

```
ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-05
Min: 4.21e-06   Media: 5.03e-06    Max: 5.80e-06

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-06
Min: 3.89e-07   Media: 4.99e-07    Max: 5.82e-07

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-07
Min: 3.66e-08   Media: 1.03e-07    Max: 2.95e-07

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-08
Min: 2.74e-07   Media: 9.04e-07    Max: 2.93e-06
```

In [ ]:

```
errorRelativo_hess(f = f_cuad, Hessf = Hessf_cuad, n = n, nt = 10, h = 1e-5)
errorRelativo_hess(f = f_cuad, Hessf = Hessf_cuad, n = n, nt = 10, h = 1e-6)
errorRelativo_hess(f = f_cuad, Hessf = Hessf_cuad, n = n, nt = 10, h = 1e-7)
errorRelativo_hess(f = f_cuad, Hessf = Hessf_cuad, n = n, nt = 10, h = 1e-8)
```

```
ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-05
Min: 7.84e-04   Media: 1.10e-03    Max: 1.75e-03

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-06
Min: 5.55e-02   Media: 1.40e-01    Max: 3.02e-01

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-07
Min: 9.75e-01   Media: 9.94e-01    Max: 1.01e+00

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-08
Min: 1.00e+00   Media: 1.00e+00    Max: 1.00e+00
```

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

$n = 10$

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
Hessf_cuad = lambda x: A2
errorRelativo_grad(f = f_cuad, gradf = gradf_cuad, n = n, nt = 50, h = 1e-5)
errorRelativo_grad(f = f_cuad, gradf = gradf_cuad, n = n, nt = 50, h = 1e-6)
errorRelativo_grad(f = f_cuad, gradf = gradf_cuad, n = n, nt = 50, h = 1e-7)
errorRelativo_grad(f = f_cuad, gradf = gradf_cuad, n = n, nt = 50, h = 1e-8)
```

```
ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-05
Min: 1.79e-06   Media: 3.51e-06    Max: 8.29e-06

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-06
Min: 1.46e-07   Media: 4.03e-07    Max: 1.17e-06

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-07
Min: 1.14e-08   Media: 3.56e-08    Max: 7.63e-08

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-08
Min: 6.25e-09   Media: 4.86e-08    Max: 1.32e-07
```

In [ ]:

```
errorRelativo_hess(f = f_cuad, Hessf = Hessf_cuad, n = n, nt = 50, h = 1e-5)
errorRelativo_hess(f = f_cuad, Hessf = Hessf_cuad, n = n, nt = 50, h = 1e-6)
errorRelativo_hess(f = f_cuad, Hessf = Hessf_cuad, n = n, nt = 50, h = 1e-7)
errorRelativo_hess(f = f_cuad, Hessf = Hessf_cuad, n = n, nt = 50, h = 1e-8)
```

```
ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-05
Min: 6.59e-06   Media: 2.57e-05    Max: 9.33e-05

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-06
Min: 1.67e-04   Media: 1.89e-03    Max: 6.14e-03

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-07
Min: 3.18e-02   Media: 2.56e-01    Max: 1.05e+00

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-08
Min: 9.39e-01   Media: 1.00e+00    Max: 1.04e+00
```

$n = 100$

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
Hessf_cuad = lambda x: A2
errorRelativo_grad(f = f_cuad, gradf = gradf_cuad, n = n, nt = 50, h = 1e-5)
errorRelativo_grad(f = f_cuad, gradf = gradf_cuad, n = n, nt = 50, h = 1e-6)
errorRelativo_grad(f = f_cuad, gradf = gradf_cuad, n = n, nt = 50, h = 1e-7)
errorRelativo_grad(f = f_cuad, gradf = gradf_cuad, n = n, nt = 50, h = 1e-8)
```

```
ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-05
Min: 2.00e-06   Media: 2.74e-06    Max: 3.80e-06

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-06
Min: 2.04e-07   Media: 2.75e-07    Max: 4.24e-07

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-07
Min: 2.37e-08   Media: 4.72e-08    Max: 7.84e-08

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-08
Min: 2.28e-07   Media: 3.71e-07    Max: 6.53e-07
```

In [ ]:

```
errorRelativo_hess(f = f_cuad, Hessf = Hessf_cuad, n = n, nt = 10, h = 1e-5)
errorRelativo_hess(f = f_cuad, Hessf = Hessf_cuad, n = n, nt = 10, h = 1e-6)
errorRelativo_hess(f = f_cuad, Hessf = Hessf_cuad, n = n, nt = 10, h = 1e-7)
errorRelativo_hess(f = f_cuad, Hessf = Hessf_cuad, n = n, nt = 10, h = 1e-8)
```

```
ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-05
Min: 3.87e-04   Media: 7.35e-04    Max: 1.04e-03

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-06
Min: 2.94e-02   Media: 5.33e-02    Max: 8.04e-02

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-07
Min: 9.30e-01   Media: 9.84e-01    Max: 1.02e+00

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-08
Min: 9.99e-01   Media: 1.00e+00    Max: 1.00e+00
```

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

In [ ]:
```
errorRelativo_grad(f = f_Beale, gradf = grad_Beale, n = 2, nt = 50, h = 1e-5
errorRelativo_grad(f = f_Beale, gradf = grad_Beale, n = 2, nt = 50, h = 1e-6
errorRelativo_grad(f = f_Beale, gradf = grad_Beale, n = 2, nt = 50, h = 1e-7
errorRelativo_grad(f = f_Beale, gradf = grad_Beale, n = 2, nt = 50, h = 1e-8
```

```
ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-05
Min: 1.51e-06   Media: 7.84e-06    Max: 4.87e-05

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-06
Min: 5.61e-08   Media: 6.87e-07    Max: 5.20e-06

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-07
Min: 6.28e-09   Media: 1.23e-07    Max: 1.58e-06

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-08
Min: 3.85e-09   Media: 2.13e-08    Max: 4.99e-08
```

In [ ]:
```
errorRelativo_hess(f = f_Beale, Hessf = Hess_Beale, n = 2, nt = 50, h = 1e-5
errorRelativo_hess(f = f_Beale, Hessf = Hess_Beale, n = 2, nt = 50, h = 1e-6
errorRelativo_hess(f = f_Beale, Hessf = Hess_Beale, n = 2, nt = 50, h = 1e-7
errorRelativo_hess(f = f_Beale, Hessf = Hess_Beale, n = 2, nt = 50, h = 1e-8
```

```
ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-05
Min: 2.83e-06   Media: 1.78e-05    Max: 5.69e-05

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-06
Min: 3.03e-06   Media: 2.33e-04    Max: 1.34e-03

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-07
Min: 2.09e-04   Media: 3.66e-02    Max: 1.58e-01

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-08
Min: 6.64e-02   Media: inf    Max: inf
```

```
/var/folders/_h/2wf1t3v96t99m5n9pzmlm9pm0000gn/T/ipykernel_2237/892321556.p
y:19: RuntimeWarning: divide by zero encountered in scalar divide
  ve[i] = np.linalg.norm(H0-Ha) / np.linalg.norm(Ha)
```

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

In [ ]:
```
errorRelativo_grad(f = f_Himmelblau, gradf = grad_Himmelblau, n = 2, nt = 50
errorRelativo_grad(f = f_Himmelblau, gradf = grad_Himmelblau, n = 2, nt = 50
errorRelativo_grad(f = f_Himmelblau, gradf = grad_Himmelblau, n = 2, nt = 50
errorRelativo_grad(f = f_Himmelblau, gradf = grad_Himmelblau, n = 2, nt = 50
```

```
ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-05
Min: 9.73e-07   Media: 7.60e-06    Max: 4.78e-05

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-06
Min: 5.75e-08   Media: 6.74e-07    Max: 3.32e-06

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-07
Min: 1.20e-08   Media: 5.76e-08    Max: 2.97e-07

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-08
Min: 3.63e-09   Media: 1.47e-07    Max: 1.04e-06
```

In [ ]:
```
errorRelativo_hess(f = f_Himmelblau, Hessf = Hess_Himmelblau, n = 2, nt = 50
errorRelativo_hess(f = f_Himmelblau, Hessf = Hess_Himmelblau, n = 2, nt = 50
errorRelativo_hess(f = f_Himmelblau, Hessf = Hess_Himmelblau, n = 2, nt = 50
errorRelativo_hess(f = f_Himmelblau, Hessf = Hess_Himmelblau, n = 2, nt = 50
```

```
ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-05
Min: 5.49e-06   Media: 1.84e-05    Max: 4.12e-05

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-06
Min: 2.32e-04   Media: 1.72e-03    Max: 4.09e-03

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-07
Min: 1.07e-02   Media: 1.48e-01    Max: 3.36e-01

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-08
Min: 9.50e-01   Media: inf    Max: inf
```

```
/var/folders/_h/2wf1t3v96t99m5n9pzmlm9pm0000gn/T/ipykernel_2237/892321556.p
y:19: RuntimeWarning: divide by zero encountered in scalar divide
  ve[i] = np.linalg.norm(H0-Ha) / np.linalg.norm(Ha)
```

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

$n = 2$

In [ ]:
```
errorRelativo_grad(f = f_Rosenbrock, gradf = grad_Rosenbrock, n = 2, nt = 50
errorRelativo_grad(f = f_Rosenbrock, gradf = grad_Rosenbrock, n = 2, nt = 50
errorRelativo_grad(f = f_Rosenbrock, gradf = grad_Rosenbrock, n = 2, nt = 50
errorRelativo_grad(f = f_Rosenbrock, gradf = grad_Rosenbrock, n = 2, nt = 50
```

```
ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-05
Min: 2.92e-06   Media: 3.71e-05    Max: 4.61e-04

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-06
Min: 5.06e-07   Media: 1.83e-06    Max: 2.14e-05

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-07
Min: 3.97e-08   Media: 2.38e-07    Max: 3.67e-06

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-08
Min: 1.28e-09   Media: 2.29e-08    Max: 8.79e-08
```

In [ ]:
```
errorRelativo_hess(f = f_Rosenbrock, Hessf = Hess_Rosenbrock, n = 2, nt = 50
errorRelativo_hess(f = f_Rosenbrock, Hessf = Hess_Rosenbrock, n = 2, nt = 50
errorRelativo_hess(f = f_Rosenbrock, Hessf = Hess_Rosenbrock, n = 2, nt = 50
errorRelativo_hess(f = f_Rosenbrock, Hessf = Hess_Rosenbrock, n = 2, nt = 50
```

```
ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-05
Min: 4.51e-06   Media: 1.78e-05    Max: 3.89e-05

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-06
Min: 1.57e-06   Media: 7.20e-05    Max: 4.78e-04

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-07
Min: 4.91e-05   Media: 8.10e-03    Max: 4.31e-02

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-08
Min: 1.02e-02   Media: 4.83e-01    Max: 2.12e+00
```

$n = 20$

In [ ]:
```
errorRelativo_grad(f = f_Rosenbrock, gradf = grad_Rosenbrock, n = 20, nt = 5
errorRelativo_grad(f = f_Rosenbrock, gradf = grad_Rosenbrock, n = 20, nt = 5
errorRelativo_grad(f = f_Rosenbrock, gradf = grad_Rosenbrock, n = 20, nt = 5
errorRelativo_grad(f = f_Rosenbrock, gradf = grad_Rosenbrock, n = 20, nt = 5
```

```
ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-05
Min: 4.87e-06   Media: 7.20e-06    Max: 1.25e-05

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-06
Min: 4.63e-07   Media: 7.35e-07    Max: 1.19e-06

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-07
Min: 4.32e-08   Media: 7.33e-08    Max: 1.28e-07

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-08
Min: 2.26e-08   Media: 5.07e-08    Max: 9.84e-08
```

In [ ]:
```
errorRelativo_hess(f = f_Rosenbrock, Hessf = Hess_Rosenbrock, n = 20, nt = 5
errorRelativo_hess(f = f_Rosenbrock, Hessf = Hess_Rosenbrock, n = 20, nt = 5
errorRelativo_hess(f = f_Rosenbrock, Hessf = Hess_Rosenbrock, n = 20, nt = 5
errorRelativo_hess(f = f_Rosenbrock, Hessf = Hess_Rosenbrock, n = 20, nt = 5
```

```
ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-05
Min: 1.10e-05   Media: 1.80e-05    Max: 3.70e-05

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-06
Min: 3.35e-04   Media: 1.45e-03    Max: 2.95e-03

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-07
Min: 3.27e-02   Media: 1.17e-01    Max: 3.13e-01

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-08
Min: 9.58e-01   Media: 9.96e-01    Max: 1.04e+00
```

$n = 40$

In [ ]:
```
errorRelativo_grad(f = f_Rosenbrock, gradf = grad_Rosenbrock, n = 40, nt = 5
errorRelativo_grad(f = f_Rosenbrock, gradf = grad_Rosenbrock, n = 40, nt = 5
errorRelativo_grad(f = f_Rosenbrock, gradf = grad_Rosenbrock, n = 40, nt = 5
errorRelativo_grad(f = f_Rosenbrock, gradf = grad_Rosenbrock, n = 40, nt = 5
```

```
ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-05
Min: 4.36e-06   Media: 6.98e-06    Max: 1.04e-05

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-06
Min: 4.99e-07   Media: 6.98e-07    Max: 1.07e-06

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-07
Min: 4.65e-08   Media: 6.85e-08    Max: 1.06e-07

ERRORES RELATIVOS EN EL CÁLCULO DEL GRADIENTE PARA h = 1e-08
Min: 5.35e-08   Media: 9.27e-08    Max: 1.35e-07
```

In [ ]:
```
errorRelativo_hess(f = f_Rosenbrock, Hessf = Hess_Rosenbrock, n = 40, nt = 5
errorRelativo_hess(f = f_Rosenbrock, Hessf = Hess_Rosenbrock, n = 40, nt = 5
errorRelativo_hess(f = f_Rosenbrock, Hessf = Hess_Rosenbrock, n = 40, nt = 5
```

```
errorRelativo_hess(f = f_Rosenbrock, Hessf = Hess_Rosenbrock, n = 40, nt = 5
```

```
ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-05
Min: 1.53e-05   Media: 3.73e-05    Max: 7.27e-05

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-06
Min: 9.82e-04   Media: 3.70e-03    Max: 1.04e-02

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-07
Min: 1.51e-01   Media: 3.44e-01    Max: 5.80e-01

ERRORES RELATIVOS EN EL CÁLCULO DE LA HESSIANA PARA h = 1e-08
Min: 9.88e-01   Media: 9.99e-01    Max: 1.00e+00
```

2.6.

¿Cuál es el valor de h que conviene usar para aproximar el gradiente y cuál para aproximar la Hessiana?

> Comparando los datos de error del cálculo del gradiente y de la matriz Hessiana de las funciones, se puede notar que para mayor exactitud, es conveniente elegir un valor pequeño de h suficientemente pequeño, sin embargo, no demasiado ya que se nota un deterioro en el error promedio en el caso de $h = 10^{-8}$. Además que existe más posibilidad de llegar a una indeterminación. Dado esto, para el cálculo del gradiente, el mejor valor de h es $h = 10^{-7}$.
>
> Por otro lado, para el cáclulo de la matriz Hessiana se puede notar que tiene un mejor desempeño cuando $h = 10^{-5}$.

3.- EJERCICIO 3:

Seleccionar un artículo para el proyecto final.

- El proyecto final se puede presentar de manera individual o en equipo formado por dos estudiantes.
- La entrega del proyecto consiste programar el algoritmo descrito en el artículo seleccionado y realizar pruebas para reproducir algunos resultados presentados en el artículo o diseñar los experimentos de prueba. El objetivo es mostrar las ventajas o limitaciones que tiene el algoritmo propuesto.

- Es válido delimitar el alcance, de manera que si aparecen varios algoritmos en el artículo, se puede seleccionar alguno de ellos para su implementación y validación.
- Hay que elaborar un reporte en el que se dé una introducción, algunos fundamentos teóricos, el planteamiento del problema, la descripción del algoritmo, los resultados obtenidos y las conclusiones.
- Hay que hacer una presentación de unos 15 minutos en el día acordado y entregar el reporte, el código y las pruebas realizadas.
- Se puede entregar un notebook como el reporte y usarlo en la presentación, para que no tener que elaborar un documento con el reporte, otro con el script del código y pruebas y otro para la presentación.
- Habrá dos fechas de entrega. La primera fecha es para los estudiantes de posgrado que será entre el 27 de mayo y el 4 de junio. La segunda fecha es para los estudiantes de licenciatura que será entre el 3 de junio y el 10 de junio.
- Si el equipo está formado por un estudiante de licenciatura y otro de posgrado tendrá que presentar el proyecto en la primera fecha.
- Para la selección se puede tomar uno de los artículos de la lista que se presenta a continuación.
- Estos artículos son una referencia. También pueden proponer algún artículo adicional, pero recomienda que cuiden que para entenderlo no tengan que revisar otras fuentes o que tengan que implementar algoritmos que requieran de temas que no fueron cubiertos en el curso y que les consuma demasiado tiempo hacer esa revisión, por ejemplo, en temas de optimización combinatoria, entera, mixta, multiobjetivo, etc.

1. Escriba el nombre de los miembros del equipo junto con el nombre del programa académico.
2. Escriba el título del artículo seleccionado
3. Si no es un artículo de la lista o que esté en el Classroom, agregue el PDF como parte de la entrega de la Tarea 7.

> **MIEMBRO:** Ezau Faridh Torres Torres.
>
> **El artículo seleccionado es el número 23:**
>
> A family of hybrid conjugate gradient method with restart procedure for unconstrained optimizations and image restorations. Xianzhen Jiang, Xiaomin Ye, Zefeng Huang, Meixing Liu. 2023
> https://drive.google.com/file/d/1h1TQpydDxkYBF0G_jd7O7-9J69X-V-dl/view?usp=sharing