Optimización I. Tarea 2

Y. Sarahi García Gozález

Librerías

```
import numpy as np
from numpy.linalg import cholesky,solve,eigvals,norm
import matplotlib.pyplot as plt
from math import sqrt
```

```
print("Tarea realizada en MacOs. \nLas versiones de las librerías y de python utili
from platform import python_version
print("Python version", python_version())
print("Numpy version", np.__version__)
```

```
Tarea realizada en MacOs.
Las versiones de las librerías y de python utilizadas fueron:

Python version 3.11.7
Numpy version 1.26.3
```

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

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

Ejercicio 1

1. Escriba una función que reciba como parámetro el nombre de un archivo `npz`, lea el archivo y cree la matriz $\mathbf{A}$ y el vector $\mathbf{b}$ del archivo `npz`, y calcule el minimizador $\mathbf{x}_*$ de $f(\mathbf{x})$ resolviendo el sistema de ecuaciones $\mathbf{A}\mathbf{x}_* = \mathbf{b}$. Use la factorización de Cholesky para resolver el sistema de ecuaciones y de esta manera saber si la matriz es definida positiva, y en este caso devolver $\mathbf{A}$, $\mathbf{b}$ y $\mathbf{x}_*$. En caso contrario devolver $\mathbf{A}$, $\mathbf{b}$ y $None$.

```
def positive_definite(A):
    '''
    Esta función determina si una matriz es definida positiva

    Parametros: (numpy.ndarray) Matriz A

    Returns: True (es definida positiva) / False (caso contrario)
    '''
    try:
        L=cholesky(A)
    except np.linalg.LinAlgError:
        print("La matriz no es definida positiva")
        return False, None

    return True, L

def read_and_solve_sistem_from_npz(file_npz):

    '''
    Esta función lee A y b de un archivo npz e intenta resolver el sistema Ax=b
    con el método de Cholesky

    Parámetros: Ruta del archivo npz file_npz

    Returns: numpy.ndarray
        A, b , x (si A es cholesky-factorizable)/ A,b none (en caso contrario)
    '''
    npzfile = np.load(file_npz)

    A  = npzfile['arr_0']
    b  = npzfile['arr_1']

    condition,L=positive_definite(A)
    if condition:
        x = solve(L.T, solve(L, b))  # Resolviendo el sistema Ax=b usando la factor
        return A, b, x

    return A,b,None
```

2. Programe la función que evalua la función $q(\mathbf{x}) = \frac{1}{2}(\mathbf{x}-\mathbf{x}_*)^\top \mathbf{A}(\mathbf{x}-\mathbf{x}_*)$. La función recibe como parámetros el punto $\mathbf{x}$, la matriz $\mathbf{A}$ y el punto $\mathbf{x}_*$ y devolver el valor de $q(\mathbf{x})$.

```
def q(x, A, x_star):
    """
    q(x) = 1/2 * (x - x_star)^T * A * (x - x_star)

    Parámetros: matriz A y vectores x, x_star (numpy.ndarray)

    Returns: q(x)
    """
    d = x - x_star
    q = 0.5 * np.dot(np.dot(d.T, A), d)
    return q
```

3. Programe una función estima la cantidad de iteraciones que el algoritmo requiere. Esta función recibe como argumentos la matriz $\mathbf{A}$, el punto $\mathbf{x}_0$, el punto $\mathbf{x}_*$ y una tolerancia $\tau > 0$. La función calcula la cantidad c descrita anteriormente y determina el entero k que cumple con $c^k \sqrt{2q(\mathbf{x}_0)} < \tau$. La función debe devolver k y c.

$$ c = \frac{\lambda_{\max}(\mathbf{A})-\lambda_{\min}(\mathbf{A})}{ \lambda_{\max}(\mathbf{A})+\lambda_{\min}(\mathbf{A})} , $$

```
def num_iterations(A,x0,x,tau):
    #Calculamos eigenlavores
    eigen_valores=eigvals(A)
    eigen_valores=np.sort(eigen_valores)
    #calculamos c
    c=(eigen_valores[-1]-eigen_valores[0])/(eigen_valores[-1]+eigen_valores[0])
    k=1

    q0=q(x0,A,x)
    #usamos escala logarítmica
    while k*np.log(c) + (0.5)*np.log(2*q0) >= np.log(tau):
        k += 1

    return c,k
```

4. Pruebe la función del punto anterior usando los datos de cada archivo `npz` contenidos en el archivo `datosTarea02.zip`. Use la función del Punto 1 y si se pudo calcular $\mathbf{x}_*$, defina n como el tamaño del vector $\mathbf{b}$, el punto inicial $\mathbf{x}_0 = (10, 10, \ldots, 10)^\top$ de dimensión n y ejecute la función del Punto 3 usando como tolerancia $\tau = \sqrt{\epsilon_m}$, donde ϵ_m es el épsilon de la máquina.

   Imprima el valor n, $q(\mathbf{x}_0)$, k, c.

- Matriz A_1 y vector b_1

```
A1,b1,x1=read_and_solve_sistem_from_npz("datosTarea02/matA_vecb1.npz")
if x1 is not None:
    x1_0=np.full(len(x1), 10)
    q1=q(x1_0,A1,x1)
    c1,k1=num_iterations(A1,x1_0,x1,sqrt(epsilon))
    print("Para el sitema A1, b1\n")
    print("El valor de x es:",x1)
    print("El valor de q1(x_0) es:",q1)
    print("El valor de c1 es:",c1)
    print("El valor de k1 es:",k1)
```

```
Para el sitema A1, b1

El valor de x es: [-0.3 -0.8]
El valor de q1(x_0) es: 1113.1499999999999
El valor de c1 es: 0.6666666666666666
El valor de k1 es: 54
```

- Matriz A_2 y vector b_2

```
A2,b2,x2=read_and_solve_sistem_from_npz("datosTarea02/matA_vecb2.npz")
if x2 is not None:
    x2_0=np.full(len(x2), 10)
    q2=q(x2_0,A2,x2)
    c2,k2=num_iterations(A2,x2_0,x2,sqrt(epsilon))
    print("Para el sitema A2, b2\n")
    print("La solución al sistema es x=",x2)
    print("El valor de q2(x_0) es:",q2)
    print("El valor de c2 es:",c2)
    print("El valor de k2 es:",k2)
```

```
Para el sitema A2, b2

La solución al sistema es x= [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
El valor de q2(x_0) es: 2658.825
El valor de c2 es: 0.8610359125293413
El valor de k2 es: 150
```

- Matriz A_3 y vector b_3

```
A3,b3,x3=read_and_solve_sistem_from_npz("datosTarea02/matA_vecb3.npz")
if x3 is not None:
    x3_0=np.full(len(x3), 10)
    q3=q(x3_0,A3,x3)
    c3,k3=num_iterations(A3,x3_0,x3,sqrt(epsilon))
    print("Para el sitema A3, b3\n")
    print("La solución al sistema es x=",x3)
    print("El valor de q3(x_0) es:",q3)
    print("El valor de c3 es:",c3)
    print("El valor de k3 es:",k3)
```

```
Para el sitema A3, b3

La solución al sistema es x= [-1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.]
El valor de q3(x_0) es: 18134.270000000004
El valor de c3 es: 0.04175532669469306
El valor de k3 es: 8
```

- Matriz A_4 y vector b_4

```
A4,b4,x4=read_and_solve_sistem_from_npz("datosTarea02/matA_vecb4.npz")
if x4 is not None:
    x4_0=np.full(len(x4), 10)
    q4=q(x4_0,A4,x4)
    c4,k4=num_iterations(A4,x4_0,x4,sqrt(epsilon))
    print("Para el sitema A4, b4\n")
    print("La solución al sistema es x=",x4)
    print("El valor de q4(x_0) es:",q4)
    print("El valor de c4 es:",c4)
    print("El valor de k4 es:",k4)
```

```
Para el sitema A4, b4

La solución al sistema es x= [ 1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.
  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.  1. -1.]
El valor de q4(x_0) es: 543978.790000002
El valor de c4 es: 0.9132114471426372
El valor de k4 es: 276
```

Ejercicio 2

Programe el Algoritmo 2 de la Clase 5 para optimizar funciones cuadráticas de la forma

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x} - \mathbf{b}^\top \mathbf{x}$$

con el método de descenso máximo con paso exacto.

- El último punto $\mathbf{x}_k$ generado por el algoritmo,
- el número k de iteraciones realizadas y
- Una variable indicadora que es $True$ si el algoritmo termina por cumplirse la condición de paro ($\|\alpha_k \mathbf{g}_k\| < \tau$) o $False$ si termina porque se alcanzó el número máximo de iteraciones.

```
def descenso_maximo_cuadraticas(A,b,x0,tau,n):
  xk=x0
  indicador=False
  for i in range(n):
    gradiente=np.dot(A,xk)-b
    ak=np.dot(gradiente.T,gradiente)/np.dot(gradiente,np.dot(A,gradiente))

    if norm(gradiente)<(tau/ak):
      indicador=True
      return xk,i,indicador

    xk=xk-np.dot(ak,gradiente)

  return xk,i,indicador
```

2. Programe la función que evalúa la función $f(\mathbf{x})$. La función recibe como argumentos la matriz $\mathbf{A}$ y el vector $\mathbf{b}$, y devuelve el valor $\frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x} - \mathbf{b}^\top \mathbf{x}$.

```
def funcion_cuadratica(A, b, x):
    """
    Evalúa la función f(x) = 0.5 * x^T * A * x - b^T * x.

    Parámetros: Matriz A, vectores b y x (numpy.ndarray)

    Return: Valor de la función evaluada en x.
    """
    return 0.5 * np.dot(x.T, np.dot(A, x)) - np.dot(b.T, x)
```

3. Pruebe el algoritmo con las matrices y vectores que se encuentran en los archivos `npz` que están contenidos en el archivo `datosTarea02.zip`.

- Matriz A_1 y vector b_1

```
#NOTA: A1,b1,x1 se obtuvieron en el ejercicio anterior
#tambien k1: numero de iteraciones aproximado

#definimos el punto inicial
x1_0=np.full(len(b1), 10)
#calculamos la funcion cuadratica en el punto inicial
f_x0_1=funcion_cuadratica(A1,b1,x1_0)
#llamamos a la funcion descenso maximo para funcines cuadráticas
xk_1,k_1,insicador_1=descenso_maximo_cuadraticas(A1,b1,x1_0,sqrt(epsilon),k1)
#calculamos la funcion cuadratica en el minimo encotrado
f_xk_1=funcion_cuadratica(A1,b1,xk_1)
#calculamos la diferencia entre el minimo encontrado y la solucion del ej anterio
diferencia1=norm(xk_1-x1)
if insicador_1:
    print("Para el sitema A1, b1\n")
    print("El valor de f(x_0) es:",f_x0_1)
    print("El numero de iteraciones fue:",k_1)
    print("El valor minimo que alcanza f es: f(x_k)=",f_xk_1)
    print("||xk - x*|| :",diferencia1)
```

```
Para el sitema A1, b1

El valor de f(x_0) es: 1110.0
El numero de iteraciones fue: 5
El valor minimo que alcanza f es: f(x_k)= -3.1500000000000004
||xk - x*|| : 1.4608710462443025e-09
```

- Matriz A_2 y vector b_2

```
#NOTA: A2,b2,x2 se obtuvieron en el ejercicio anterior
#tambien k2: numero de iteraciones aproximado

#definimos el punto inicial
x2_0=np.full(len(b2), 10)
#calculamos la funcion cuadratica en el punto inicial
f_x0_2=funcion_cuadratica(A2,b2,x2_0)
#llamamos a la funcion descenso maximo para funcines cuadráticas
xk_2,k_2,insicador_2=descenso_maximo_cuadraticas(A2,b2,x2_0,sqrt(epsilon),k2)
#calculamos la funcion cuadratica en el minimo encotrado
f_xk_2=funcion_cuadratica(A2,b2,xk_2)
#calculamos la diferencia entre el minimo encontrado y la solucion del ej anterio
diferencia2=norm(xk_2-x2)
if insicador_2:
    print("Para el sitema A2, b2\n")
    print("El valor de f(x_0) es:",f_x0_2)
    print("El numero de iteraciones fue:",k_2)
    print("El valor minimo que alcanza f es: f(x_k)=",f_xk_2)
    print("||xk - x*|| :",diferencia2)
```

```
Para el sitema A2, b2

El valor de f(x_0) es: 2626.0
El numero de iteraciones fue: 105
El valor minimo que alcanza f es: f(x_k)= -32.825
||xk - x*|| : 6.309047497713108e-08
```

- Matriz A_3 y vector b_3

```
#NOTA: A3,b3,x3 se obtuvieron en el ejercicio anterior
#tambien k3: numero de iteraciones aproximado

#definimos el punto inicial
x3_0=np.full(len(b3), 10)
#calculamos la funcion cuadratica en el punto inicial
f_x0_3=funcion_cuadratica(A3,b3,x3_0)
#llamamos a la funcion descenso maximo para funcines cuadráticas
xk_3,k_3,insicador_3=descenso_maximo_cuadraticas(A3,b3,x3_0,sqrt(epsilon),k3)
#calculamos la funcion cuadratica en el minimo encotrado
f_xk_3=funcion_cuadratica(A3,b3,xk_3)
#calculamos la diferencia entre el minimo encontrado y la solucion del ej anterio
diferencia3=norm(xk_3-x3)
if insicador_3:
    print("Para el sitema A3, b3\n")
    print("El valor de f(x_0) es:",f_x0_3)
    print("El numero de iteraciones fue:",k_3)
    print("El valor minimo que alcanza f es: f(x_k)=",f_xk_3)
    print("||xk - x*|| :",diferencia3)
```

```
Para el sitema A3, b3

El valor de f(x_0) es: 17984.4
El numero de iteraciones fue: 7
El valor minimo que alcanza f es: f(x_k)= -149.87
||xk - x*|| : 1.232687539667818e-09
```

- Matriz A_4 y vector b_4

```
#NOTA: A4,b4,x4 se obtuvieron en el ejercicio anterior
#tambien k4: numero de iteraciones aproximado

#definimos el punto inicial
x4_0=np.full(len(b4), 10)
#calculamos la funcion cuadratica en el punto inicial
f_x0_4=funcion_cuadratica(A4,b4,x4_0)
#llamamos a la funcion descenso maximo para funcines cuadráticas
xk_4,k_4,insicador_4=descenso_maximo_cuadraticas(A4,b4,x4_0,sqrt(epsilon),k4)
#calculamos la funcion cuadratica en el minimo encotrado
f_xk_4=funcion_cuadratica(A4,b4,xk_4)
#calculamos la diferencia entre el minimo encontrado y la solucion del ej anterio
diferencia4=norm(xk_4-x4)
if insicador_4:
    print("Para el sitema A4, b4\n")
    print("El valor de f(x_0) es:",f_x0_4)
    print("El numero de iteraciones fue:",k_4)
    print("El valor minimo que alcanza f es: f(x_k)=",f_xk_4)
    print("||xk - x*|| :",diferencia4)
```

```
Para el sitema A4, b4

El valor de f(x_0) es: 543542.6
El numero de iteraciones fue: 117
El valor minimo que alcanza f es: f(x_k)= -436.1899999999994
||xk - x*|| : 1.08798605430435e-07
```

4. Escriba un comentario sobre si el número de iteraciones estimadas fue una buena cota superior.

Ejercicio 3

Programe el Algoritmo 1 de la Clase 5 de descenso máximo, usando el método de la sección dorada para obtener $\alpha_k \in [0, 1]$:

```
def seccion_dorada(f,x_l,x_u,epsilon,N):
    '''
    Esta función busca el mínimo de f en el intervalo [x_l,x_u]
    argumentos:
        f: funcion a optimizar
        x_l,x_u: limites inferior y superior del intervalo de busqueda
        epsilon: tolerancia
        N: número máximo de iteraciones
    returns:
        x_k: el punto donde se minimiza f
        x_l,x_u: intervalo donde se ecunetra el minimo (intervalo de incertidumbr
        k: número de iteraciones realizadas
        b_res: true si el algoritmo terminó porque se cumplió el criterio de paso
    '''

    rho=(sqrt(5)-1)/(2)

    for i in range(N):
        b=rho*(x_u-x_l)
        x_1=x_u-b
        x_3=x_l+b
        if f(x_1)<f(x_3):
            x_u=x_3
            x_k=x_1
        else:
            x_l=x_1
            x_k=x_3

        if np.abs(x_u-x_l) < epsilon:
            return x_k

    return None
```

Use las tolerancias $\tau_1 = \sqrt{n}\epsilon_m^{1/3}, \tau_2 = \epsilon_m^{1/2}$, donde ϵ_m es el épsilon de la máquina, use el número de iteraciones máximas $N = 10000$ para el descenso máximo y $N_{gs} = 200$ para el método de la sección dorada.

```
def descenso_minimo_I(f,df,x0,tau1=sqrt(2)*((epsilon)**(0.33)),tau2=sqrt(epsilon)
    '''
    Esta funcion busca el minimo de la funcion f
```

```
    Parametros:
        f: función a optimizar
        df: gradiente de f
        x_0: valor inicial
        tau_1, N_1: tolerancia y numero maximo de iteraciones (descenso)
        tau_2, N_2: tolerancia y numero maximo de iteraciones (seccion dorada)

        NOTA:los argumentos predeterminados son específicos para este ejercicio y

    returns:
        x_k: ultimo punto de la sucesión que genera el algoritmo
        k: número de términos d ela sucesión
        True/False: Indica si se satisfizo la condición de tolerancia
        x1,x2...xk: sucesión de puntos (np.array)
    '''

    sucesion_x=[] #aqui alamcenamos la sucesion de puntos
    indicador=False #nos indica si se cumplo o no la condicion de tolerancia


    #calculmos p0 y a0 para x0
    p0=df(x0)
    phi= lambda a: f(x0-a*p0)
    a0=seccion_dorada(phi,0,1,tau2,n2)


    #igualamos ak,pk y xk a a0,p0 y x0
    ak,pk,xk=a0,p0,x0
    sucesion_x.append(x0)


    for i in range(n1):

        if norm(ak*pk)<tau1: #si se cumple la condicion de tolerania:
            indicador=True #indicador verdadero
            break #y romepos el ciclo

        #agoritmo de descenso
        pk=df(xk)

        #calculamos el alpha minimo en cada iteracion
        phi= lambda a: f(xk-a*pk)
        ak=seccion_dorada(phi,0,1,tau2,n2)
        if ak is not None:
            xk=xk-ak*pk
```

```
            sucesion_x.append(xk)
        else:
            print("no se encontro el mismi")



    if len(x0)==2:
        return sucesion_x[-1],len(sucesion_x),indicador,sucesion_x

    return sucesion_x[-1],len(sucesion_x),indicador,None


def contornosFnc2D(fncf, puntos, xleft, xright, ybottom, ytop, levels):
    # Crea una discretización uniforme del intervalo [xleft, xright]
    ax = np.linspace(xleft, xright, 250)
    # Crea una discretización uniforme del intervalo [ybottom, ytop]
    ay = np.linspace(ybottom, ytop, 200)
    # La matriz mX que tiene las abscisas
    mX, mY = np.meshgrid(ax, ay)
    # Se crea el arreglo mZ con los valores de la función en cada nodo
    mZ = mX.copy()
    for i, y in enumerate(ay):
        for j, x in enumerate(ax):
            mZ[i,j] = fncf(np.array([x, y]))
    # Grafica de las curvas de nivel
    fig, ax = plt.subplots()
    CS = ax.contourf(mX, mY, mZ, levels, cmap='coolwarm')
    plt.colorbar(CS, ax=ax)
    # Grafica los puntos dados
    puntos_x = [p[0] for p in puntos]
    puntos_y = [p[1] for p in puntos]
    ax.plot(puntos_x, puntos_y, 'r.',label="Sucesión")
    ax.plot(puntos_x[0], puntos_y[0], 'g*',label="punto inicial")
    ax.plot(puntos_x[-1], puntos_y[-1], 'b*',label="mínimo encontrado")
     # Grafica los puntos como puntos rojos
    ax.set_xlabel('x1')
    ax.set_ylabel('x2')
    ax.set_title('Contornos de la función sucesión')
    plt.legend()
    plt.show()
```

- **Función de Himmelblau:**

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

\begin{align*} &\text{Derivada parcial de } f \text{ con respecto a } x_1:
&\frac{\partial f}{\partial x_1} = 4x_1(x_1^2 + x_2 - 11) + 2(x_1 + x_2^2 - 7)

```
\\
&\text{Derivada parcial de } f \text{ con respecto a } x_2: \\
&\frac{\partial f}{\partial x_2} = 2(x_1^2 + x_2 - 11) + 4x_2(x_1 + x_2^2 - 7) \\
```

\end{align*}

Esta función tiene 4 mínimos:

$f(3, 2) = 0$

$f(-2.805, 2.1313) = 0$

$f(-3.779, -3.283) = 0$

$f(3.584, -1.848) = 0$

```
#Funciones auxiliares
def H1(x):
    return ((x[0])**2+x[1]-11)**2

def H2(x):
    return (x[0]+(x[1])**2-7)**2

#Funcion Himmelblau y su derivada
def Himmelblau(x):
    return (H1(x)+H2(x))

def D_Himmelblau(x):
    return np.array([4*x[0]*H1(x)+2*H2(x),2*H1(x)+4*x[1]*H2(x)])
```

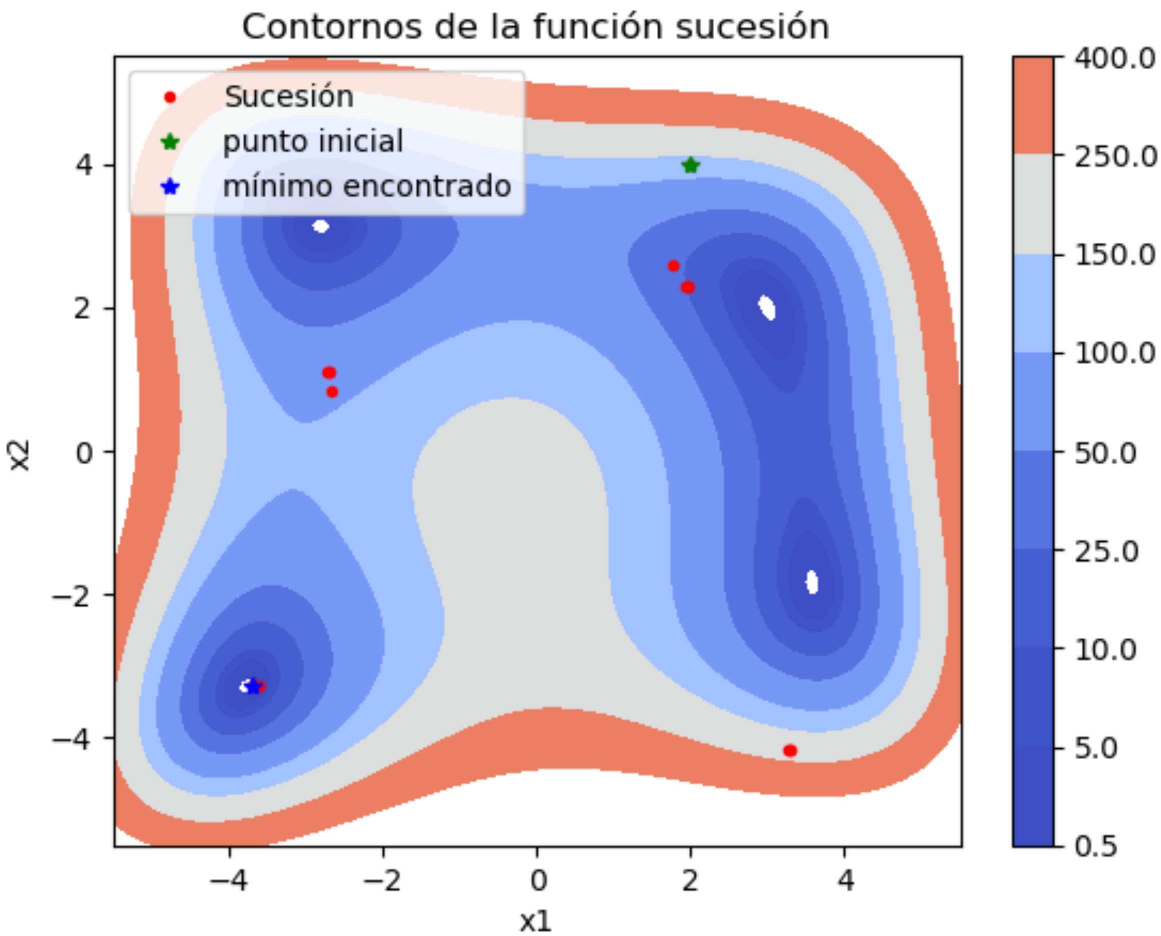
A continuación se aplica a la función de Himmelblau el algoritmo para los siguientes dos puntos inciales:

$$\mathbf{x}_0 = (2., 4.)$$
$$\mathbf{x}_0 = (0., 0.)$$

```
x0=[2,4]

xk,k,indicador,sucesion=descenso_minimo_I(Himmelblau,D_Himmelblau,x0)

if indicador:
    print("Para el punto inicial x0=",x0)
    print("Después de",k,"iteraciónes el mínimo se encontró en xk=",xk)
    print("La sucesión es:")
    for i in range(k):
        print(sucesion[i])
```

```
Para el punto inicial x0= [2, 4]
Después de 17 iteraciónes el mínimo se encontró en xk= [-3.67975964 -3.279292
La sucesión es:
[2, 4]
[1.77509706 2.60044473]
[-2.67526551  0.8302803 ]
[-3.59825263 -3.27535529]
[ 3.27428657 -4.17184922]
[1.94436122 2.29794849]
[-2.70549143  1.09139026]
[-3.66520395 -3.27818624]
[ 3.2824644  -4.16950332]
[1.96520854 2.29229875]
[-2.70755903  1.09283472]
[-3.67529451 -3.27896632]
[ 3.28641522 -4.16640385]
[1.97417756 2.28976325]
[-2.70846727  1.09334609]
[-3.6797597  -3.27929227]
[-3.67975964 -3.27929228]
```

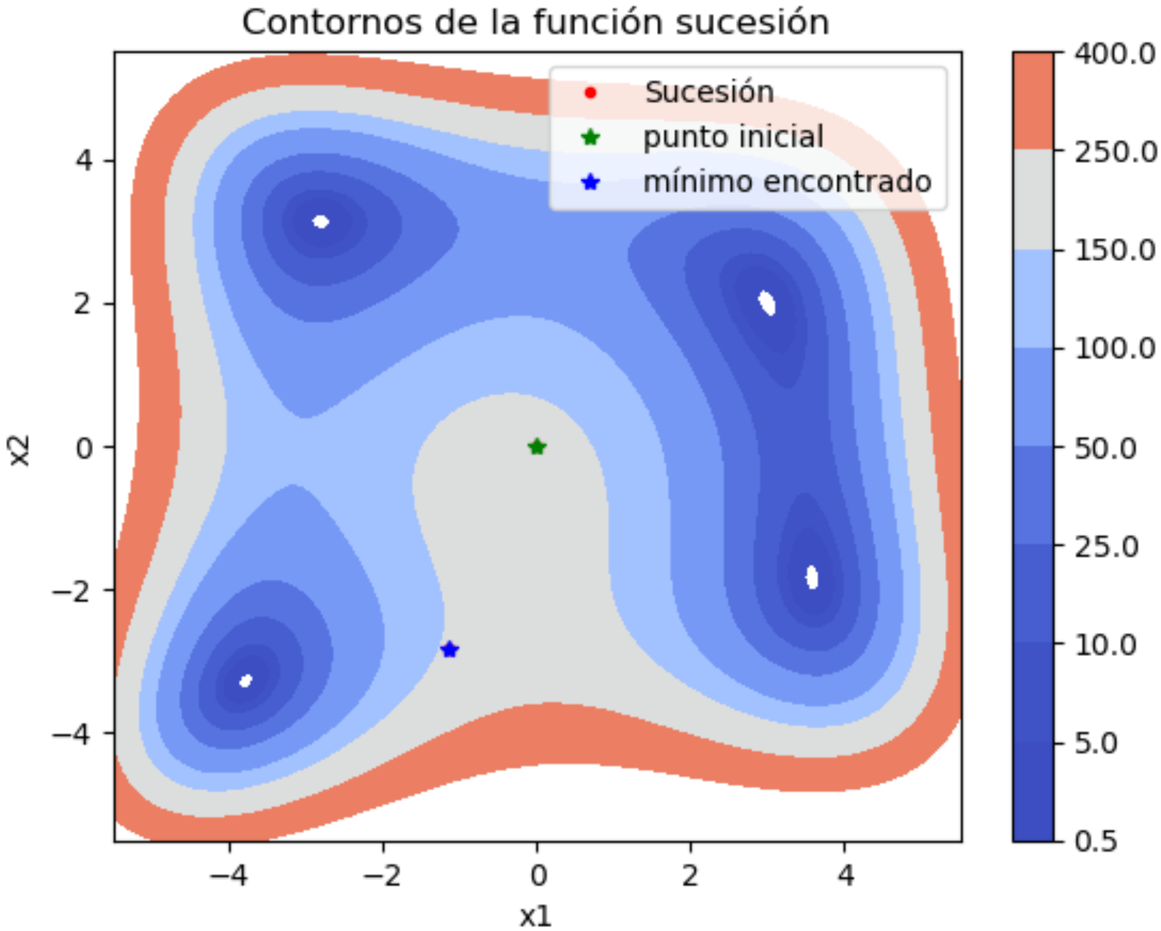
```
contornosFnc2D(Himmelblau, sucesion,xleft=-5.5, xright=5.5, ybottom=-5.5, ytop=5.
               levels=[0.5, 5, 10, 25, 50, 100, 150, 250, 400])
```



```
x0=[0,0]

xk,k,indicador,sucesion=descenso_minimo_I(Himmelblau,D_Himmelblau,x0)

if indicador:
    print("Para el punto inicial x0=",x0)
    print("Después de",k,"iteraciónes el mínimo se encontró en xk=",xk)
    print("La sucesión es:")
    for i in range(k):
        print(sucesion[i])
```

```
Para el punto inicial x0= [0, 0]
Después de 3 iteraciónes el mínimo se encontró en xk= [-1.14290774 -2.8222970
La sucesión es:
[0, 0]
[-1.1429128  -2.82229488]
[-1.14290774 -2.82229709]
```

```
contornosFnc2D(Himmelblau, sucesion,xleft=-5.5, xright=5.5, ybottom=-5.5, ytop=5.
               levels=[0.5, 5, 10, 25, 50, 100, 150, 250, 400])
```



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

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

$$\text{Derivada parcial de } f \text{ con respecto a } x_1 :$$
$$\begin{aligned}\frac{\partial f}{\partial x_1} &= 2(1.5 - x_1 + x_1 x_2)(-1 + x_2)\\ &\quad + 2(2.25 - x_1 + x_1 x_2^2)(-1 + x_2^2)\\ &\quad + 2(2.625 - x_1 + x_1 x_2^3)(-1 + x_2^3)\end{aligned}$$
$$\text{Derivada parcial de } f \text{ con respecto a } x_1 :$$
$$\begin{aligned}\frac{\partial f}{\partial x_2} &= 2(1.5 - x_1 + x_1 x_2)(x_1)\\ &\quad + 2(2.25 - x_1 + x_1 x_2^2)(2 x_1 x_2)\\ &\quad + 2(2.625 - x_1 + x_1 x_2^3)(3 x_1 x_2^2)\end{aligned}$$

Esta función tiene 1 mínimos:

$$f(3, 0.5) = 0$$

```
# Definir la función f
def Beale(x):
    return (1.5 - (x[0]) + (x[0])*(x[1]))**2 + (2.25 - (x[0]) + (x[0])*(x[1])**2)

def D_Beale(x):
    Partial_Beale_dx1 = 2*(1.5 - (x[0]) + (x[0])*(x[1]))*(-1 + (x[1])) + 2*(2.25 
    Partial_Beale_dx2 = 2*(1.5 - (x[0]) + (x[0])*(x[1]))*((x[0])) + 2*(2.25 - (x[
    return np.array([Partial_Beale_dx1, Partial_Beale_dx2])
```

A continuación se aplica a la función de Beale el algoritmo para los siguientes dos puntos inciales:

$$\mathbf{x}_0 = (2., 3.)$$
$$\mathbf{x}_0 = (2., 4.)$$

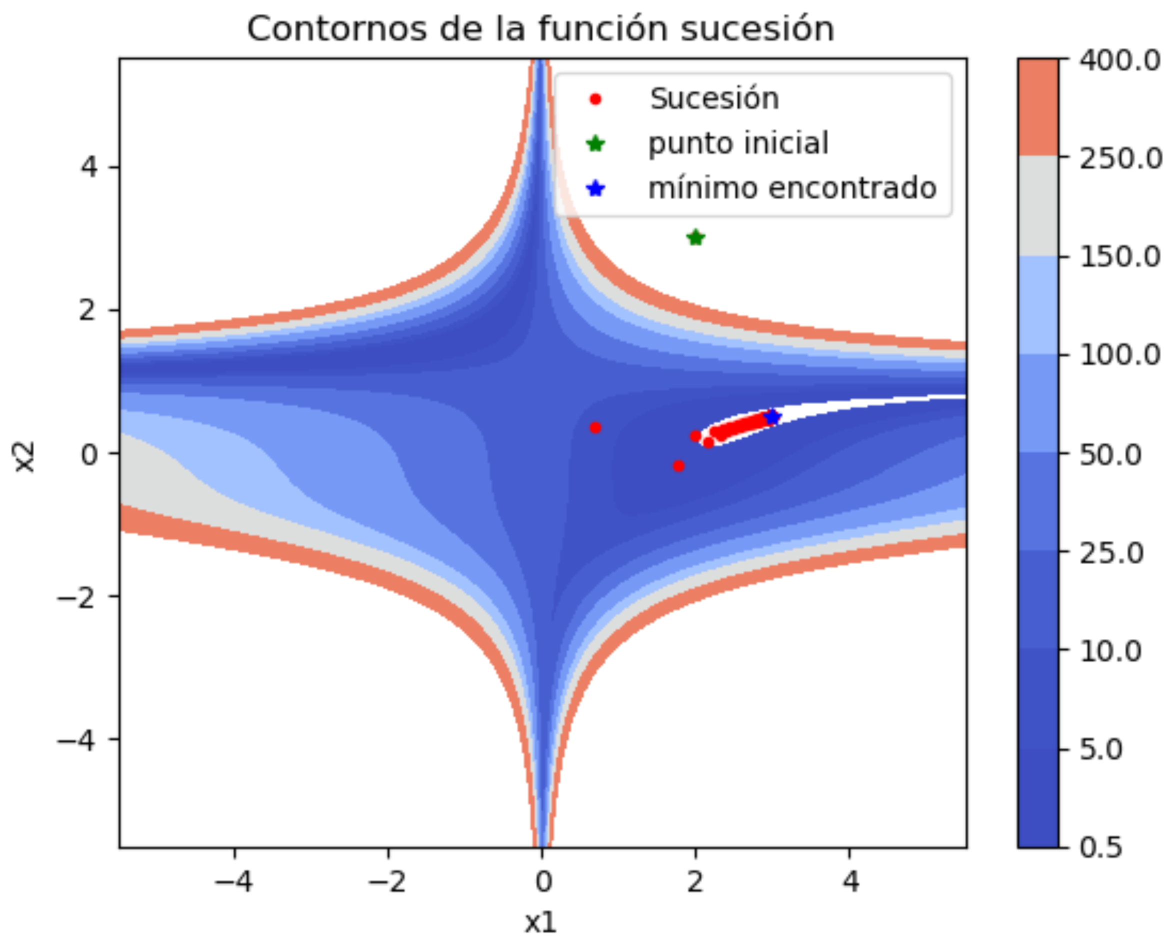
```
x0=[2,3]

xk,k,indicador,sucesion=descenso_minimo_I(Beale,D_Beale,x0)

if indicador:
    print("Para el punto inicial x0=",x0)
    print("Después de",k,"iteraciónes el mínimo se encontró en xk=",xk)
    print("La sucesión es:")
```

```
for i in range(k):
    print(sucesion[i])
```

```
Para el punto inicial x0= [2, 3]
Después de 470 iteraciónes el mínimo se encontró en xk= [2.99944505 0.4998645
La sucesión es:
[2, 3]
[0.68445851 0.34785667]
[ 1.7722452  -0.19175753]
[1.98095386 0.22896981]
[2.15705146 0.14161373]
[2.22697061 0.2825609 ]
[2.31550695 0.238641  ]
[2.35438407 0.3170118 ]
[2.41135691 0.28874949]
[2.43728236 0.34101155]
[2.4785061  0.32056184]
[2.49752665 0.3589046 ]
[2.52945373 0.34306665]
[2.54425471 0.37290334]
[2.57009663 0.36008403]
[2.58208218 0.38424522]
[2.60365273 0.37354479]
[2.61364073 0.39367918]
[2.63205835 0.38454282]
[2.64056331 0.40168761]
[2.65656368 0.39375036]
[2.66392853 0.40859688]
[2.67802011 0.40160652]
[2.68448399 0.41463679]
[2.69703223 0.40841203]
[2.70276799 0.41997451]
[2.71404401 0.41438086]
[2.71918037 0.42473503]
[2.72939064 0.41967006]
[2.73402582 0.42901393]
[2.7433311  0.42439789]
[2.74754166 0.43288579]
[2.75606946 0.42865544]
[2.75991614 0.43640981]
[2.76776923 0.43251415]
[2.77130098 0.43963365]
[2.77856342 0.43603099]
[2.7818202  0.44259621]
[2.78856154 0.43925206]
[2.79157642 0.44532963]
[2.79785486 0.4422151 ]
```

```
[2.80065545 0.44786069]
[2.80652018 0.4449514 ]
[2.80912976 0.45021196]
[2.8146227 0.4474871]
[2.81706111 0.4524026 ]
[2.82221826 0.44984431]
[2.82450249 0.454449  ]
[2.82935502 0.45204183]
[2.83149974 0.4563653 ]
[2.83607478 0.45409577]
[2.83809276 0.45816372]
[2.84241409 0.45602006]
[2.84431645 0.45985497]
[2.84840501 0.45782678]
[2.85020157 0.46144841]
```

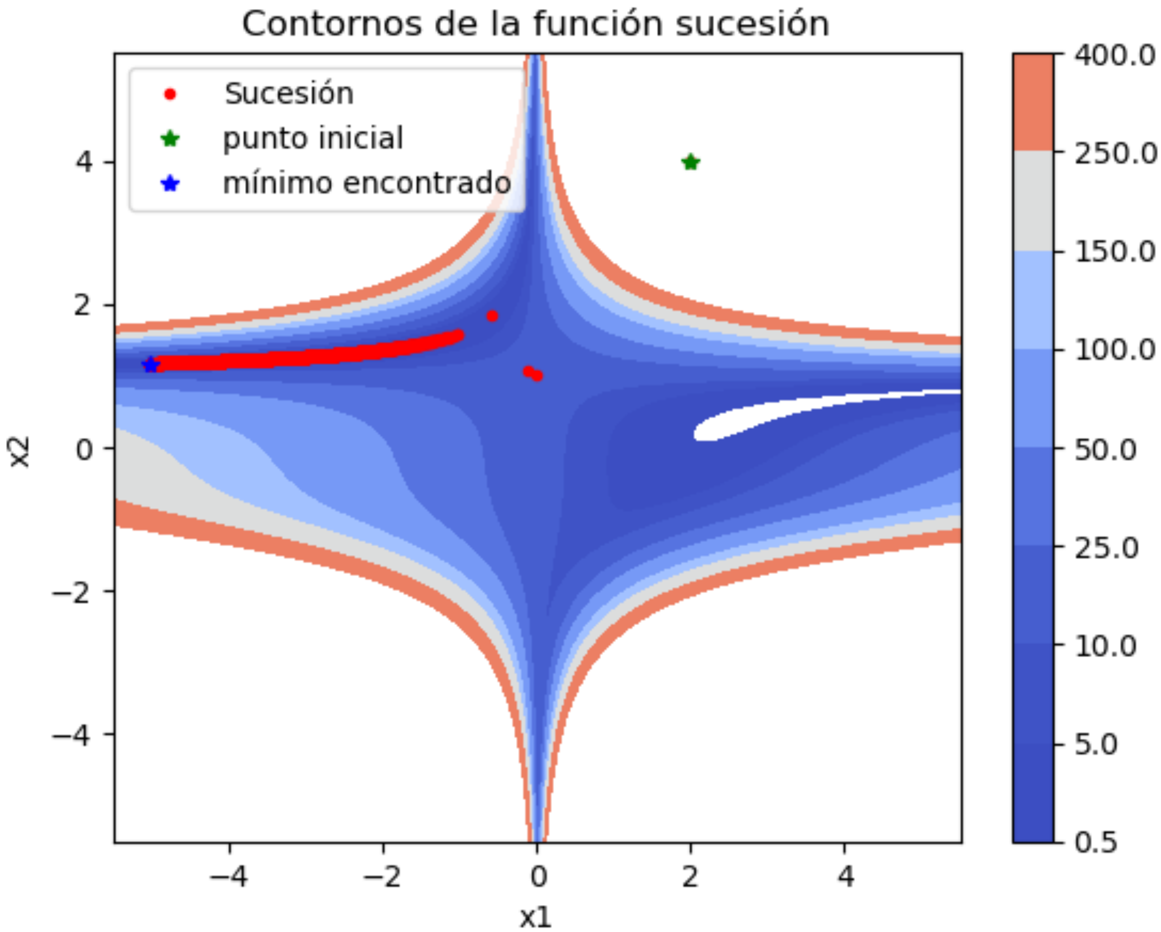
```
contornosFnc2D(Beale, sucesion,xleft=-5.5, xright=5.5, ybottom=-5.5, ytop=5.5,
               levels=[0.5, 5, 10, 25, 50, 100, 150, 250, 400])
```



```
x0=[2,4]

xk,k,indicador,sucesion=descenso_minimo_I(Beale,D_Beale,x0)

if indicador:
    print("Para el punto inicial x0=",x0)
    print("Después de",k,"iteraciónes el mínimo se encontró en xk=",xk)
    print("La sucesión es:")
    for i in range(k):
        print(sucesion[i])
else:
    print("Para el punto inicial x0=",x0)
    print("No se cumplió la condución de tolerancia")
    print("El útimo elemento de la sucesión es xk=",xk, "con k=",k)
```

```
Para el punto inicial x0= [2, 4]
No se cumplió la condución de tolerancia
El útimo elemento de la sucesión es xk= [-5.01259866  1.16977361] con k= 1000
```

```
contornosFnc2D(Beale, sucesion,xleft=-5.5, xright=5.5, ybottom=-5.5, ytop=5.5,
                    levels=[0.5, 5, 10, 25, 50, 100, 150, 250, 400])
```



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

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

Esta función tiene 1 mínimo (para cada n):

$f(1, 1) = 0$

$f(1, 1, 1...) = 0$

```
def Rosenbrock(x):
    n = len(x)
    suma = 0
    for i in range(n-1):
        suma += 100 * (x[i+1] - x[i]**2)**2 + (1 - x[i])**2
    return suma

def D_Rosenbrock(x):
    n = len(x)
    gradient = np.zeros(n)
    for i in range(n-1):
        gradient[i] += -400 * x[i] * (x[i+1] - x[i]**2) - 2 * (1 - x[i])
        gradient[i+1] += 200 * (x[i+1] - x[i]**2)
    return gradient
```

A continuación se aplica a la función de Himmelblau el algoritmo para los siguientes dos puntos inciales:

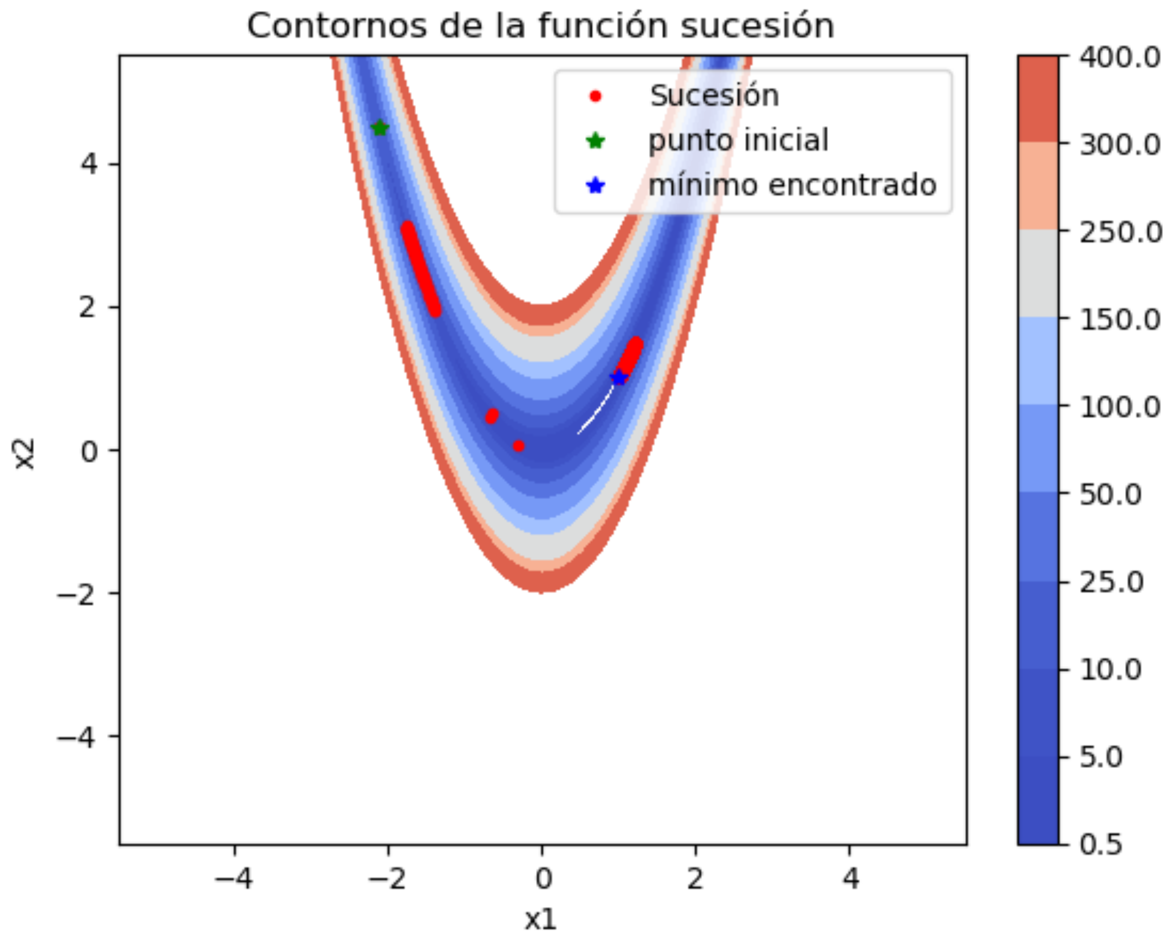
$$\mathbf{x}_0 = (-2.1, 4.5)$$
$$\mathbf{x}_0 = (-1.2, 1.0)$$
$$\mathbf{x}_0 = (-2.1, 4.5, -2.1, 4.5, -2.1, 4.5, -2.1, 4.5, -2.1, 4.5)$$
$$\mathbf{x}_0 = (-1.2, 1.0, -1.2, 1.0, -1.2, 1.0, -1.2, 1.0, -1.2, 1.0)$$

```
x0=[-2.1,4.5]

xk,k,indicador,sucesion=descenso_minimo_I(Rosenbrock,D_Rosenbrock,x0)

if indicador:
    print("Para el punto inicial x0=",x0)
    print("Después de",k,"iteraciónes el mínimo se encontró en xk=",xk)
    print("La sucesión es:")
    for i in range(k):
        print(sucesion[i])
```

```
Para el punto inicial x0= [-2.1, 4.5]
Después de 3115 iteraciónes el mínimo se encontró en xk= [1.00338062 1.006783
La sucesión es:
[-2.1, 4.5]
[-2.11854997  4.49518877]
[-1.75812879  3.10823099]
[-1.76069909  3.1068881 ]
[-1.75417471  3.09440158]
[-1.75676364  3.09304882]
```

```
 [-1.7501704   3.08043073]
 [-1.75277885  3.07906775]
 [-1.74611295  3.06630968]
 [-1.74874196  3.06493607]
 [-1.74200475  3.0520427 ]
 [-1.74465387  3.05065845]
 [-1.73784266  3.03762432]
 [-1.74051257  3.03622911]
 [-1.7336248  3.0230491]
 [-1.73631614  3.02164263]
 [-1.72934982  3.0083132 ]
 [-1.73206305  3.00689519]
 [-1.7250158   2.99341182]
 [-1.72775146  2.99198199]
 [-1.72061712  2.97833081]
 [-1.72337681  2.97688855]
 [-1.71615563  2.96307188]
 [-1.7189392   2.96161707]
 [-1.71162768  2.94762783]
 [-1.71443597  2.94616007]
 [-1.70702962  2.93198886]
 [-1.70986372  2.93050766]
 [-1.70236019  2.91615055]
 [-1.70522064  2.91465558]
 [-1.69761603  2.90010486]
 [-1.70050377  2.89859564]
 [-1.69279713  2.88385106]
 [-1.69571229  2.88232738]
 [-1.68789655  2.86737356]
 [-1.69084083  2.8658347 ]
 [-1.68291042  2.8506602 ]
 [-1.68588515  2.84910557]
 [-1.67783705  2.83370542]
 [-1.68084289  2.83213457]
 [-1.6726739   2.81650346]
 [-1.67571154  2.81491595]
 [-1.66741554  2.79904157]
 [-1.67048639  2.79743673]
 [-1.66205895  2.7813111 ]
 [-1.665164    2.77968837]
 [-1.65659828  2.7632978 ]
 [-1.65973905  2.76165643]
 [-1.65102786  2.74498657]
 [-1.654206    2.74332576]
 [-1.6453443   2.72636783]
 [-1.64856089  2.72468694]
 [-1.63954262  2.70742991]
 [-1.64279894  2.7057282 ]
 [-1.6336134   2.68814997]
 [-1.63691187  2.68642635]
```

```
contornosFnc2D(Rosenbrock, sucesion,xleft=-5.5, xright=5.5, ybottom=-5.5, ytop=5.
               levels=[ 0.5, 5, 10, 25, 50, 100, 150, 250,300, 400])
```



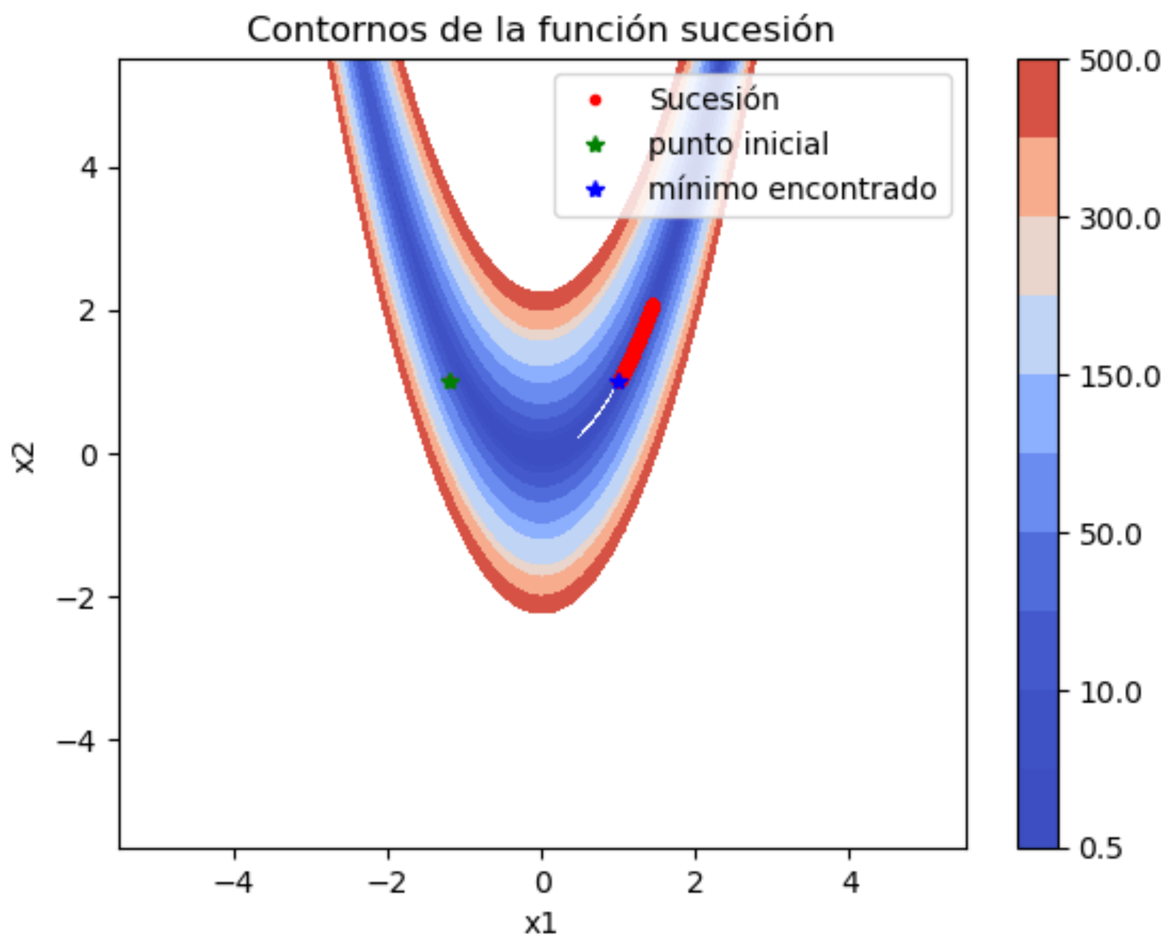

```
x0=[-1.2,1]

xk,k,indicador,sucesion=descenso_minimo_I(Rosenbrock,D_Rosenbrock,x0)

if indicador:
    print("Para el punto inicial x0=",x0)
    print("Después de",k,"iteraciónes el mínimo se encontró en xk=",xk)
    print("La sucesión es:")
    for i in range(k):
        print(sucesion[i])
```

```
Para el punto inicial x0= [-1.2, 1]
Después de 8173 iteraciónes el mínimo se encontró en xk= [1.0040565  1.008138
La sucesión es:
[-1.2, 1]
[1.44087777 2.07790929]
```

```
 [1.44105651 2.07746871]
 [1.44067074 2.07731221]
 [1.44084946 2.07687169]
 [1.44046372 2.0767152 ]
 [1.4406424  2.07627475]
 [1.44025669 2.07611827]
 [1.44043535 2.07567788]
 [1.44004965 2.07552141]
 [1.44022829 2.07508109]
 [1.43984262 2.07492462]
 [1.44002122 2.07448437]
 [1.43963557 2.07432791]
 [1.43981416 2.07388773]
 [1.43942853 2.07373128]
 [1.43960709 2.07329116]
 [1.43922148 2.07313472]
 [1.43940001 2.07269466]
 [1.43901443 2.07253823]
 [1.43919293 2.07209824]
 [1.43880737 2.07194183]
 [1.43898585 2.0715019 ]
 [1.43860031 2.07134549]
 [1.43877876 2.07090563]
 [1.43839325 2.07074923]
 [1.43857167 2.07030944]
 [1.43818618 2.07015305]
 [1.43836458 2.06971332]
 [1.43797912 2.06955694]
 [1.43815748 2.06911728]
 [1.43777204 2.06896091]
 [1.43795038 2.06852131]
 [1.43756497 2.06836495]
 [1.43774328 2.06792542]
 [1.43735788 2.06776907]
 [1.43753617 2.0673296 ]
 [1.4371508  2.06717326]
 [1.43732906 2.06673386]
 [1.43694371 2.06657753]
 [1.43712195 2.06613819]
 [1.43673662 2.06598187]
 [1.43691483 2.0655426 ]
 [1.43652953 2.06538629]
 [1.43670771 2.06494709]
 [1.43632243 2.06479079]
 [1.43650058 2.06435165]
 [1.43611533 2.06419536]
 [1.43629345 2.06375629]
 [1.43590822 2.06360001]
 [1.43608632 2.063161  ]
 [1.43570112 2.06300473]
```

```
[1.43587919 2.06256579]
[1.43549401 2.06240953]
[1.43567205 2.06197066]
[1.43528689 2.06181441]
```

```
contornosFnc2D(Rosenbrock, sucesion,xleft=-5.5, xright=5.5, ybottom=-5.5, ytop=5.
               levels=[ 0.5, 5, 10, 25, 50, 100, 150, 250,300, 400,500])
```



```
x0=[-2.1, 4.5, -2.1, 4.5, -2.1, 4.5, -2.1, 4.5, -2.1, 4.5]

xk,k,indicador,sucesion=descenso_minimo_I(Rosenbrock,D_Rosenbrock,x0)

if indicador:
    print("Para el punto inicial x0=",x0)
    print("Después de",k,"iteraciónes el mínimo se encontró en \n xk=",xk)
```

```
Para el punto inicial x0= [-2.1, 4.5, -2.1, 4.5, -2.1, 4.5, -2.1, 4.5, -2.1, 4
Después de 7781 iteraciónes el mínimo se encontró en
 xk= [0.99997994 0.99995841 0.99991928 0.99983454 0.99967253 0.99933911
 0.99867953 0.99735071 0.9946977  0.98939561]
```

```
x0=[-1.2, 1.0, -1.2, 1.0, -1.2, 1.0, -1.2, 1.0, -1.2, 1.0]

xk,k,indicador,sucesion=descenso_minimo_I(Rosenbrock,D_Rosenbrock,x0)

if indicador:
    print("Para el punto inicial x0=",x0)
    print("Después de",k,"iteraciónes el mínimo se encontró en \n xk=",xk)
```

```
Para el punto inicial x0= [-1.2, 1.0, -1.2, 1.0, -1.2, 1.0, -1.2, 1.0, -1.2, 1
Después de 7642 iteraciónes el mínimo se encontró en
 xk= [0.99997944 0.99995745 0.99991726 0.99983062 0.99966452 0.99932322
 0.99864753 0.99728678 0.99456977 0.98914054]
```

3. Repita la prueba para función de Rosenbrock usando el punto inicial $\mathbf{x}_0 = (-2.1, 4.5)$ usando $\tau_2 = \epsilon_m^{1/4}$ y $N_{gs} = 50$ para relajar las condiciones de paro del método de la sección dorada y ver si podemos terminar más rápido. Escriba un comentario sobre si conviene hacer esto o cuando no conviene hacerlo.

```
x0=[-2.1,4.5]

xk,k,indicador,sucesion=descenso_minimo_I(Rosenbrock,D_Rosenbrock,x0,tau2=epsilon

if indicador:
    print("Para el punto inicial x0=",x0)
    print("Después de",k,"iteraciónes el mínimo se encontró en xk=",xk)
```

```
    print("La sucesión es:")
    for i in range(k):
        print(sucesion[i])
```

```
Para el punto inicial x0= [-2.1, 4.5]
Después de 6761 iteraciónes el mínimo se encontró en xk= [1.00138956 1.002786
La sucesión es:
[-2.1, 4.5]
[-2.11943433  4.4949594 ]
[-2.11838937  4.49479375]
[-2.11698175  4.47766385]
[2.1981218  4.83342237]
[2.19780455 4.83330586]
[2.19797886 4.83280835]
[2.19766604 4.83269085]
[2.19784923 4.83215531]
[2.19751116 4.83204357]
[2.19769634 4.8315659 ]
[2.19738348 4.83144844]
[2.19756646 4.830913  ]
[2.19722869 4.83080122]
[2.19741336 4.8303237 ]
[2.19710109 4.83020614]
[2.1972832  4.82967094]
[2.19694643 4.82955897]
[2.19712992 4.82908176]
[2.19681889 4.82896394]
[2.19699943 4.82842915]
[2.19666436 4.82831682]
[2.19684602 4.82784007]
[2.19653687 4.82772186]
[2.19671517 4.82718762]
[2.1963825  4.82707478]
[2.19656168 4.82659863]
[2.19625503 4.82647988]
[2.19643834 4.8259223 ]
[2.19609084 4.82581285]
[2.19626771 4.82537328]
[2.19594745 4.8252354 ]
[2.19613911 4.82482849]
[2.19582574 4.82468905]
[2.19601053 4.82428372]
[2.19570402 4.82414274]
[2.19588872 4.82372369]
[2.19557375 4.82358465]
[2.19575996 4.82317902]
[2.19545212 4.82303836]
[2.19563802 4.82261907]
[2.19532193 4.82248031]
[2.19550908 4.8220745 ]
```

```
 [2.1952004  4.82193407]
 [2.195387   4.82151465]
 [2.19507029 4.82137606]
 [2.19525788 4.82097017]
 [2.19494886 4.82082985]
 [2.19513564 4.82041042]
 [2.19481884 4.82027188]
 [2.19500637 4.81986604]
 [2.1946975  4.81972571]
 [2.19488396 4.81930638]
 [2.19456757 4.81916777]
 [2.19475454 4.81876209]
 [2.19444631 4.81862164]
```

Ejercicio 4

Sea $f(x) = (x-1)^2$ con $x \in \mathbb{R}$ y generamos la secuencia

$$x_{k+1} = x_k - \frac{\alpha}{2^k} f'(x_k)$$

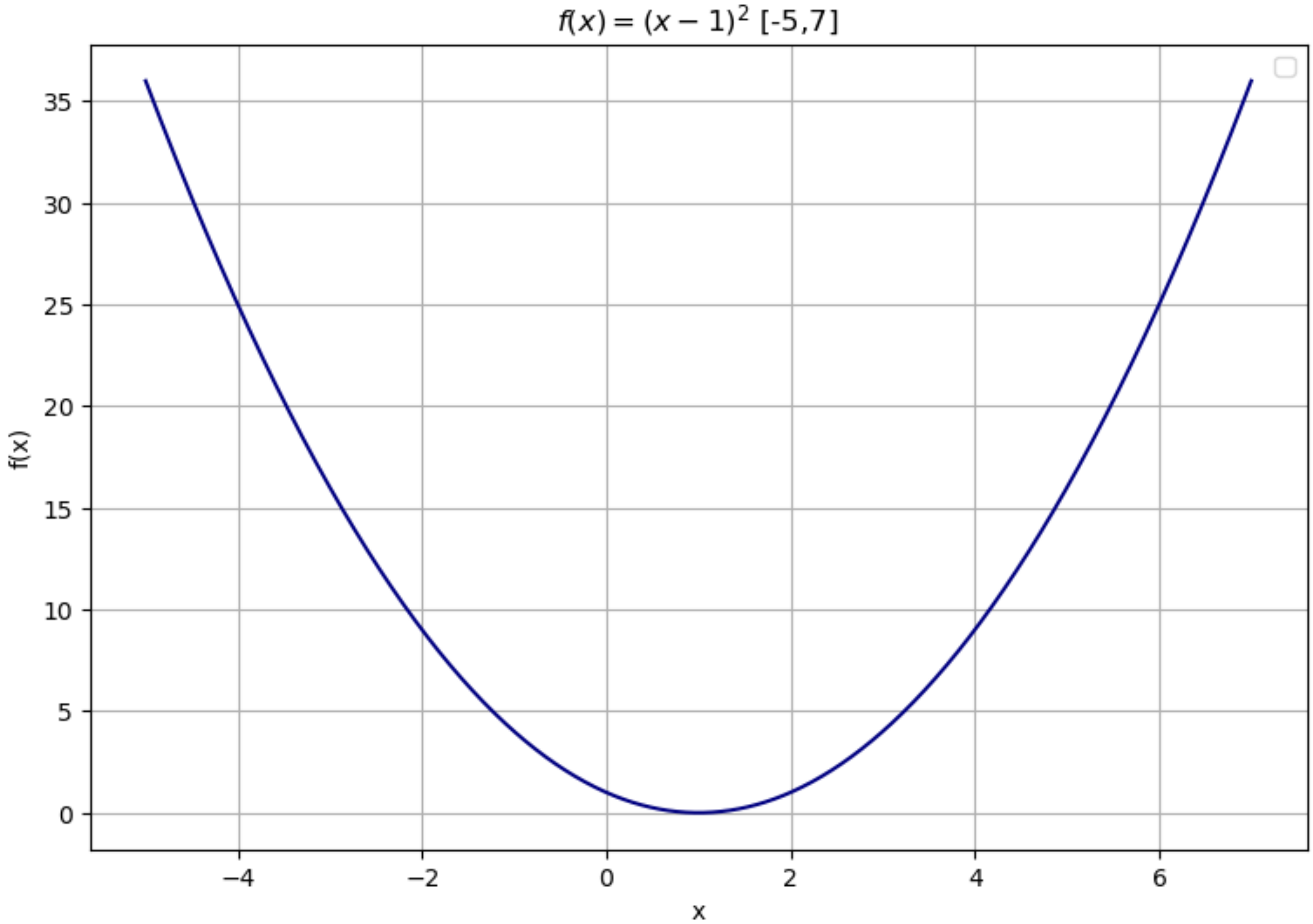
con $0 < \alpha < 1$, para obtener el minimizador de la función $f(x)$. ¿Tiene este algoritmo la propiedad de descenso, es decir, $f(x_{k+1}) < f(x_k)$ a partir de un cierto k? ¿Es el algoritmo globalmente convergente?

```
def f(x):
    return (x-1)**2

def df(x):
    return 2*(x-1)
```

Observemos la gráfica de f:

```
x = np.linspace(-5, 7, 1000)

plt.figure(figsize=(9,6))
plt.plot(x,f(x),'navy')
#plt.plot(x_m3,f3(x_m3),'ro',label="Mínimo $(0.67958,f2(0.67958))$")
plt.ylabel('f(x)')
plt.xlabel('x')
plt.legend()
plt.title('$f(x)=(x-1)^2$ [-5,7]')
plt.grid()
```

```
No artists with labels found to put in legend.  Note that artists whose label
```



Este algoritmo NO cumple la propiedad del descenso pues al realizar $x_{k+1} = x_k - \frac{\alpha}{2^k} f'(x_k)$ siempre puede suceder que $x_{k+1} < 1$ y $x_k > 1$ (o al revés) de manera que que $f(x_{k+1}) >= f(x_k)$ pues la función f es simétrica respecto a 1. Esto puede suceder con otras funciones también.

El algoritmo para esta función es globalmente convergente pues el gradiente siempre apunta en la misma dirección ya que hay un único mínimo global y la función es convexa. Con otras funciones esto no necesariamente sucede, puede haber más de un mínimo cerca y el gradiente puede cambiar de dirección y estancarse.