Segura_Guillermo_Tarea2

February 11, 2024

1 Tarea 2. Optimización

Guillermo Segura Gómez

1.1 Ejercicio 1

Estimar la cantidad de iteraciones que requiere el algoritmo de descenso máximo con paso exacto para alcanzar el minimizador x_* de la función cuadrática

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

donde A es una matriz simétrica y definida positiva que su información y la del arreglo b está almacenada en archivo en formato *npz* contenido en el archivo datosTarea02.zip.

Para hacer esto, calculamos el minimizador x_* de $f(x)$ resolviendo el sistema de ecuaciones $Ax_* = b$ y definimos

$$q(x) = \frac{1}{2}\left(x - x_*\right)^\top A\left(x - x_*\right).$$

Sabemos que $q(x)$ y $f(x)$ sólo difieren en una constante y podemos usar $q(x)$ para estimar la manera en que decrece la función mediante el resultado de la Proposición 6 de la Clase 6 :

$$q\left(x_{k+1}\right) \leq \left(\frac{\lambda_{\max}(A) - \lambda_{\min}(A)}{\lambda_{\max}(A) + \lambda_{\min}(A)}\right)^2 q\left(x_k\right).$$

Si

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

entonces

$$q\left(x_{k+1}\right) \leq c^2 q\left(x_k\right) \leq c^4 q\left(x_{k-1}\right) \leq c^6 q\left(x_{k-2}\right) \leq ... \leq c^{2k} q\left(x_1\right) \leq c^{2(k+1)} q\left(x_0\right).$$

Como

$$2q\left(x_k\right) = \left(x - x_*\right)^\top A\left(x - x_*\right) = \left\|x - x_*\right\|_A^2,$$

$q\left(x_k\right)$ es una medida de la distancia al cuadrado de x_k a x_*, de modo dada una tolerancia $\tau > 0$ podemos buscar el valor k para el cual se cumpla

$$\left\|x_k - x_*\right\|_A = \sqrt{2q\left(x_k\right)} \leq c^k \sqrt{2q\left(x_0\right)} < \tau$$

y ese k es la estimación de la cantidad de iteraciones que requiere el algoritmo.

Nota: Cada archivo npz en el ZIP tiene dos arreglos que corresponden a la matriz A y el vector b. Para leer los datos puede hacer, por ejemplo: npzfile = np.load("datosTarea02/matA_vecb1.npz") A = npzfile['arr_0'] b = npzfile['arr_1'] $b = npzfile\,['\,\mathrm{arr}\,1^{prime}]$

1.1.1 Ejercicio 1.1

Escriba una función que reciba como parámetro el nombre de un archivo npz, lea el archivo y cree la matriz A y el vector b del archivo npz, y calcule el minimizador x_* de $f(x)$ resolviendo el sistema de ecuaciones $Axx_* = b$. 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 A, b y x_*. En caso contrario devolver A, b y None.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
npzfile = np.load("datosTarea02/matA_vecb1.npz")
A = npzfile['arr_0']
b = npzfile['arr_1']

print(A)
```

```
[[6. 4.]
 [4. 6.]]
```

Se implementó el método Cholesky descrito en la página 324 del libro Buden. [1] Además se implementaron las funciones de sustitución hacía atras y sustitución hacia adelante, para poder solcuionar un sistema $Ax = B$.

[3]:
```
def Cholesky(MatA):
    # Dimensiones de la matriz
    dimN = MatA.shape[0]

    # Inicializar la matriz L
    MatL = np.zeros((dimN, dimN))

    # Paso 1. Determinar l_11 = sqrt(a_11)
    MatL[0, 0] = np.sqrt(MatA[0, 0])

    # Paso 2. Para j=2,...,n determine l_{j 1} = a_{j 1} / l_{11}
    for j in range(1, dimN):
        MatL[j, 0] = MatA[j, 0] / MatL[0, 0]

    # Paso 3. Para i=2,...,n-1 haga los pasos 4 y 5
    for i in range(1, dimN):
        # Paso 4. Determinar l_{i i} = (a_{i i} - sum_{k=1}^{i-1} l_{i k}^2)^(1/
↪2)
        sum_k = np.sum(np.square(MatL[i, :i]))
```

```
        MatL[i, i] = np.sqrt(MatA[i, i] - sum_k)

        # Paso 5. Para j=i+1,...,n determine l_{j i} = (a_{j i} -␣
↪sum_{k=1}^{i-1} l_{j k} * l_{i k}) / l_{i i}
        for j in range(i+1, dimN):
            sum_k = np.sum(MatL[j, :i] * MatL[i, :i])
            MatL[j, i] = (MatA[j, i] - sum_k) / MatL[i, i]

    return MatL

def sustitucion_hacia_adelante(L, b):
    y = np.zeros_like(b)
    for i in range(len(b)):
        y[i] = (b[i] - np.dot(L[i, :i], y[:i])) / L[i, i]
    return y

def sustitucion_hacia_atras(LT, y):
    x = np.zeros_like(y)
    for i in range(len(y) - 1, -1, -1):
        x[i] = (y[i] - np.dot(LT[i, i + 1:], x[i + 1:])) / LT[i, i]
    return x
```

[4]:
```
# Ejemplo burden de factorización Cholesky
matrix = np.array([[4, -1, 1], [-1, 4.25, 2.75], [1, 2.75, 3.5]])
L = Cholesky(matrix)
print(L)
```

```
[[ 2.   0.   0. ]
 [-0.5  2.   0. ]
 [ 0.5  1.5  1. ]]
```

Ahora se implementa la función para el ejercicio 1.

[5]:
```
# Ejercicio 1

def Minimizer(fileName):
    npzfile = np.load("datosTarea02/" + fileName + ".npz")
    A = npzfile['arr_0']
    b = npzfile['arr_1']

    try:
            # Calcular L por factorización Cholesky
            L = Cholesky(A)

            # Resolver Ly = b para y usando sustitución hacia adelante
            y = sustitucion_hacia_adelante(L, b)

            # Resolver L^Tx = y para x usando sustitución hacia atrás
```

```
            x = sustitucion_hacia_atras(L.T, y)

    except (ValueError, np.linalg.LinAlgError):
        # En caso de error (matriz no definida positiva), devolver None para x
        x = None

    return A, b, x
```

Para imprimir el sistema se implementa una función.

[6]:
```
def ImprimirSistema(A, b, x):
    # Imprimir la matriz A
    print(f"Matriz A: \n" , A)

    # Imprimir el vector b
    print("\nVector b: \n", b)

    # Comprobar si x es None (lo que significa que la matriz A no es definida␣
 ↪positiva)
    if x is not None:
        # Imprimir la solución x
        print("\nLa solución del sistema es:")
        print("Vector x: \n", x)
    else:
        print("\nNo se encontró solución: La matriz A no es definida positiva.")
```

Se resuelven todos los sistemas utilizando un ciclo.

[7]:
```
import glob
import os

# Especifica la ruta de la carpeta y el patrón de búsqueda
path = "datosTarea02/*.npz"

for filename in glob.glob(path):

    # Eliminar la ruta completa y solo conservar el nombre del archivo
    base_name = os.path.basename(filename)
    name_without_ext = os.path.splitext(base_name)[0]

    # Resolver el sistema
    A, b, x = Minimizer(name_without_ext)

    # Imprimir sistema
    print("Para el sistema " + name_without_ext)
    ImprimirSistema(A, b, x)
```

```
Para el sistema matA_vecb4
Matriz A:
 [[1.79 0.03 0.07 … 0.04 0.06 0.04]
 [0.03 1.75 0.04 … 0.03 0.05 0.04]
 [0.07 0.04 1.76 … 0.06 0.07 0.04]
 …
 [0.04 0.03 0.06 … 1.76 0.03 0.05]
 [0.06 0.05 0.07 … 0.03 1.81 0.05]
 [0.04 0.04 0.04 … 0.05 0.05 1.78]]

Vector b:
 [ 1.73 -1.58  1.35 -1.52  1.93 -1.19  1.82 -1.41  1.76 -1.31  2.37 -2.07
  1.65 -1.53  1.89 -1.6   1.75 -1.26  1.72 -1.42  1.97 -1.3   1.09 -1.28
  1.85 -2.07  1.71 -1.42  1.49 -1.91  1.46 -1.53  1.56 -1.76  1.93 -1.38
  1.97 -1.81  1.53 -1.65  2.12 -1.17  1.68 -3.11  1.73 -2.18  1.75 -1.22
  2.01 -1.94  2.01 -2.1   1.85 -2.3   1.32 -1.04  1.45 -2.23  2.05 -1.93
  1.46 -1.74  1.67 -1.14  2.24 -1.2   1.73 -1.53  2.11 -1.23  2.06 -1.74
  1.55 -1.33  1.69 -1.43  1.59 -1.44  1.83 -1.03  2.05 -2.23  1.52 -2.06
  1.01 -2.06  1.76 -1.84  1.73 -1.64  1.9  -1.4   1.47 -1.19  1.3  -1.44
  2.1  -1.97  1.43 -1.66  1.69 -1.3   1.61 -1.53  1.82 -1.65  2.25 -1.29
  1.55 -1.1   0.72 -2.56  1.7  -2.08  1.53 -1.41  2.21 -0.87  2.34 -1.48
  1.63 -2.25  1.75 -1.71  1.93 -2.07  1.85 -1.32  1.87 -2.6   1.68 -2.11
  1.29 -1.4   2.36 -1.54  1.66 -2.    1.85 -1.68  2.04 -2.01  1.87 -1.82
  1.81 -1.64  1.86 -2.02  1.18 -1.36  1.6  -1.67  2.12 -2.05  1.71 -1.66
  1.45 -1.6   1.77 -1.3   1.42 -1.78  1.93 -0.92  1.71 -1.82  1.83 -1.76
  1.84 -2.2   2.11 -1.43  1.27 -1.47  1.49 -1.4   1.7  -1.78  1.87 -1.38
  2.44 -1.32  1.59 -2.13  1.97 -1.68  2.3  -2.14  2.04 -2.26  2.05 -1.8
  1.14 -1.57  1.77 -1.46  1.89 -2.19  1.37 -2.43  1.88 -1.39  1.28 -2.35
  1.11 -1.97  1.49 -1.67  1.62 -1.51  1.84 -2.08  1.43 -1.54  1.4  -1.97
  1.7  -1.55  1.86 -2.35  1.59 -1.67  2.06 -1.83  1.82 -2.21  1.36 -1.57
  1.34 -1.76  1.64 -1.2   2.61 -1.96  2.05 -1.37  1.2  -1.92  1.67 -1.38
  1.42 -2.11  1.63 -1.88  2.13 -2.45  1.89 -2.32  1.68 -1.84  1.84 -1.73
  1.42 -1.45  1.76 -2.35  2.33 -1.79  2.52 -1.67  1.97 -1.95  1.15 -1.79
  2.12 -1.87  1.72 -1.23  1.37 -1.32  2.21 -1.99  1.91 -1.48  0.88 -2.02
  2.12 -2.36  1.41 -2.35  2.22 -1.08  2.52 -1.65  1.48 -1.68  1.63 -1.92
  2.32 -1.67  1.85 -1.96  1.98 -2.33  1.42 -2.5   1.49 -1.44  1.39 -2.39
  1.72 -1.73  1.99 -2.41  2.43 -1.35  1.92 -1.9   1.52 -1.55  2.41 -1.88
  1.26 -1.68  2.47 -1.41  1.95 -1.74  1.7  -2.11  2.13 -2.53  1.46 -1.8
  1.42 -2.25  2.08 -1.67  1.1  -1.86  1.51 -1.92  1.8  -1.75  1.38 -1.42
  1.38 -2.03  1.75 -1.91  1.43 -1.88  1.64 -1.34  1.49 -2.15  1.84 -1.87
  1.35 -2.46  1.36 -1.07  1.62 -1.47  2.33 -1.76  1.77 -2.74  1.74 -2.07
  2.06 -2.26  2.14 -1.05  1.98 -2.13  2.37 -1.67  1.68 -2.28  1.34 -1.32
  1.94 -1.42  1.66 -1.54  2.08 -1.95  2.21 -1.9   1.89 -1.5   1.11 -1.97
  1.82 -1.67  1.71 -1.64  1.85 -2.01  1.84 -1.64  2.3  -1.75  1.29 -1.46
  1.83 -1.19  1.67 -1.77  1.82 -2.31  1.22 -2.13  1.18 -2.18  1.85 -0.3
  1.94 -2.46  1.99 -1.5   2.25 -1.64  1.92 -1.91  2.08 -2.21  1.9  -1.74
  1.9  -1.33  1.96 -0.92  1.82 -2.49  1.97 -1.97  1.27 -1.69  1.32 -1.82
  1.14 -1.49  1.47 -2.16  2.01 -1.57  1.69 -1.72  1.97 -2.08  1.68 -1.8
```

```
 1.34 -1.35  1.27 -1.71  2.27 -1.8   1.29 -1.44  1.4  -1.7   0.74 -1.8
 1.65 -1.78  0.96 -1.39  1.27 -2.6   2.35 -1.85  1.42 -1.73  1.82 -1.33
 1.76 -1.63  1.87 -2.07  1.86 -1.78  1.94 -1.72  1.61 -1.64  2.3  -2.17
 1.38 -1.95  1.59 -2.37  1.89 -1.64  1.62 -1.08  2.29 -1.55  1.97 -1.01
 1.51 -1.85  1.33 -1.89  2.16 -1.81  1.72 -1.64]

La solución del sistema es:
Vector 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.  1. -1.  1. -1.  1. -1.]
Para el sistema matA_vecb3
Matriz A:
 [[ 2.91 -0.    0.   … -0.01  0.    0.  ]
 [-0.    2.9   0.01 …  0.   -0.    0.01]
 [ 0.    0.01  2.91 … -0.01  0.   -0.  ]
 …
 [-0.01  0.   -0.01 …  2.9   0.01  0.  ]
 [ 0.   -0.    0.   …  0.01  2.9  -0.  ]
 [ 0.    0.01 -0.   …  0.   -0.    2.9 ]]

Vector b:
 [-2.97 -3.03 -2.99 -2.95 -2.97 -3.   -3.   -3.   -3.   -3.01 -3.01 -2.94
```

```
 -2.99 -3.01 -2.99 -2.99 -3.01 -3.03 -2.97 -3.02 -3.01 -3.03 -3.   -3.01
 -3.03 -3.   -2.98 -3.03 -2.97 -3.03 -2.98 -2.95 -2.95 -2.99 -2.99 -2.99
 -2.96 -3.03 -3.01 -3.02 -2.96 -3.03 -3.01 -3.02 -2.97 -2.95 -3.02 -3.04
 -2.99 -2.97 -2.99 -2.99 -3.02 -3.01 -2.97 -3.01 -2.98 -3.02 -3.03 -2.99
 -3.01 -2.97 -3.02 -2.96 -2.98 -3.   -2.99 -3.   -3.   -3.01 -3.02 -2.99
 -2.97 -2.94 -2.99 -3.05 -3.05 -2.99 -3.01 -2.98 -3.04 -3.   -2.96 -3.01
 -3.02 -2.99 -3.02 -3.04 -3.   -2.97 -3.01 -3.02 -2.98 -3.02 -2.98 -3.03
 -2.98 -2.96 -2.94 -3.03]

La solución del sistema es:
Vector 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.]
Para el sistema matA_vecb2
Matriz A:
 [[2.58 0.13 0.89 0.89 0.63 0.57 0.59 0.64 0.8  0.79]
 [0.13 2.09 0.14 0.47 0.54 0.62 0.17 0.23 0.23 0.52]
 [0.89 0.14 2.12 0.79 0.48 0.1  0.74 0.89 0.07 0.43]
 [0.89 0.47 0.79 2.3  0.36 0.52 0.21 0.31 0.74 0.21]
 [0.63 0.54 0.48 0.36 2.25 0.25 0.26 0.42 0.49 0.52]
 [0.57 0.62 0.1  0.52 0.25 1.95 0.56 0.78 0.21 0.38]
 [0.59 0.17 0.74 0.21 0.26 0.56 2.29 0.42 0.71 0.48]
 [0.64 0.23 0.89 0.31 0.42 0.78 0.42 2.03 0.47 0.42]
 [0.8  0.23 0.07 0.74 0.49 0.21 0.71 0.47 2.76 0.4 ]
 [0.79 0.52 0.43 0.21 0.52 0.38 0.48 0.42 0.4  2.34]]

Vector b:
 [8.51 5.14 6.65 6.8  6.2  5.94 6.43 6.61 6.88 6.49]

La solución del sistema es:
Vector x:
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Para el sistema matA_vecb1
Matriz A:
 [[6. 4.]
 [4. 6.]]

Vector b:
 [-5. -6.]

La solución del sistema es:
Vector x:
 [-0.3 -0.8]
```

1.1.2 Ejercicio 2.1

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

[8]:
```
def q(x, A, x_star):
    diff = x - x_star  # Diferencia entre x y x_star
    return 0.5 * np.dot(diff.T, np.dot(A, diff))  # Calcula q(x)
```

1.1.3 Ejercicio 1.3 y 1.4

Programe una función estima la cantidad de iteraciones que el algoritmo requiere. Esta función recibe como argumentos la matriz A, el punto x_0, el punto x_* y una tolerancia $\tau > 0$. La función calcula la cantidad c descrita anteriormente y determina el entero k que cumple con $c^k\sqrt{2q(x_0)} < \tau$. La función debe devolver k y c.

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 x_*, defina n como el tamaño del vector b, el punto inicial $x_0 = (10, 10, ..., 10)^\top$ 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 $n, q(x_0), k, c$.

Para calcular

Para calcular el número de iteraciones $ k $ necesarias para que el algoritmo de descenso máximo alcance una cierta tolerancia $ $ con respecto a la distancia al minimizador $ x_* $, se puede usar la relación proporcionada:

$$\sqrt{2q(x_k)} \leq c^k\sqrt{2q(x_0)} < \tau$$

Despejando para k

$$c^{2k}q(x_0) < \frac{\tau^2}{2}$$

Tomando logaritmos en ambos lados para resolver para k:

$$2k\log c < \log\left(\frac{\tau^2}{2q(x_0)}\right)$$

$$k > -\frac{1}{2\log c}\log\left(\frac{2q(x_0)}{\tau^2}\right)$$

[9]:
```
def NoIterations(A, x0, x_star, tau):

    # Calcula los valores propios de A
    valores_propios = np.linalg.eigvals(A)
```

```
    # Encuentra el valor propio máximo y mínimo
    lambda_max = np.max(valores_propios)
    lambda_min = np.min(valores_propios)

    # Calcula c
    c = (lambda_max - lambda_min) / (lambda_max + lambda_min)
    # print(f"El valor de c: ", c)

    # Despejar k para calcularlo
    if c < 1:  # Verificar que c sea menor que 1 para evitar el logaritmo de␣
 ↪números no positivos
        k = -(1 / (2 * np.log(c))) * np.log((2 * q(x0, A, x_star)) / (tau ** 2))
        print(f"El valor de q: ", q(x0, A, x_star))
        k = np.ceil(k)  # Redondear k al entero más cercano hacia arriba
    else:
        k = np.inf  # Si c no es menor que 1, el algoritmo no converge

    return c, k
```

Ahora implementamos todos los ejemplo como arriba realizamos la factorización Cholesky

[10]:
```
# Especifica la ruta de la carpeta y el patrón de búsqueda
path = "datosTarea02/*.npz"

# Obtener el épsilon de la máquina para números de punto flotante de doble␣
 ↪precisión
epsilon_m = np.finfo(float).eps

# Calcular la tolerancia como la raíz cúbica del épsilon de la máquina
tolerancia = epsilon_m ** (1/3)

for filename in glob.glob(path):

    # Eliminar la ruta completa y solo conservar el nombre del archivo
    base_name = os.path.basename(filename)
    name_without_ext = os.path.splitext(base_name)[0]

    # Resolver el sistema
    A, b, x = Minimizer(name_without_ext)

    # Encontrar número de iteraciones
    x0 = np.ones(len(b))*10

    print("Solución de sistema " + name_without_ext)
    print(f"Valor de n: ", len(b))

    c, k = NoIterations(A, x0, x, tolerancia)
```

```
    print(f"c calculado: ", c)
    print(f"Número de iteraciones calculado: ", k)
```

```
Solución de sistema matA_vecb4
Valor de n:  500
El valor de q:  543978.79
c calculado:  0.9132114471426372
Número de iteraciones calculado:  209.0
Solución de sistema matA_vecb3
Valor de n:  100
El valor de q:  18134.270000000004
c calculado:  0.04175532669469306
Número de iteraciones calculado:  6.0
Solución de sistema matA_vecb2
Valor de n:  10
El valor de q:  2658.8249999999994
c calculado:  0.8610359125293413
Número de iteraciones calculado:  109.0
Solución de sistema matA_vecb1
Valor de n:  2
El valor de q:  1113.15
c calculado:  0.6666666666666666
Número de iteraciones calculado:  40.0
```

1.2 Ejercicio 2

Programe el algoritmo 2 de la clase 5 para optimizar funciones de la forma

$$f(x) = \frac{1}{2}x^T Ax - b^T x$$

con el método de descenso máximo con paso exacto. ### Ejercicio 2.1

La función que implemente el algoritmo recibe como argumento: - la matriz A (que se supone que es simétrica y definida positiva), - el vector b de la función cuadrática, - un punto inicial x_0 - una tolerancia τ y - el número máximo de iteraciones N.

La función debe devolver: - El último punto 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 ($\|\alpha_k g_k\| < \tau$) o False si termina porque se alcanzó el número máximo de iteraciones.

[11]:
```
def DescensoMax_Func2(MatA, vecb, xk, tau, N):
    for i in range(N):
        gk = MatA @ xk - vecb

        alpha_k = np.dot(gk.T, gk) / np.dot(gk.T, (MatA @ gk))
```

```
        # Verifica la condición de parada usando la norma del paso
        if np.linalg.norm(alpha_k * gk) < tau:
            return xk, i, True

        # Actualiza xk para la siguiente iteración
        xk = xk - alpha_k * gk

    return xk, N, False
```

1.2.1 Ejercicio 2.2

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

[12]:
```
def f(matA, vecb, x):
    term1 = 0.5 * x.T @ matA @ x  # Calcula 1/2 x^T A x
    term2 = vecb.T @ x            # Calcula b^T x
    return term1 - term2
```

1.2.2 Ejercicio 2.3

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

Para cada archivo npy haga lo siguiente: - Use la función del Punto 1 del Ejercicio 1 para obtener A, b y x_*. Si x_* no es None continúe y defina la variable n como el tamaño del vector b. Imprima el valor de n para saber la dimensión de la variable x. - Haga $x_0 = (10, 10, ..., 10)^\top$ de dimensión n. - Defina la tolerancia $\tau = \sqrt{\epsilon_m}$, donde ϵ_m es el épsilon de la máquina. - Calcule el punto x_k con el algoritmo. Elija el número de iteraciones máximas para el algoritmo. Puede tomar como referencia el resultado en el Ejercicio 1. - Imprima los valores

$$f(x_0), k, f(x_k), \|x_k - x_*\|,$$

y x_k si $n \leq 6$, o los primeros tres elementos y los últimos tres elementos del arreglo x_k si $n > 6$.

[13]:
```
# Especifica la ruta de la carpeta y el patrón de búsqueda
path = "datosTarea02/*.npz"

# Obtener el épsilon de la máquina para números de punto flotante de doble␣
 ↪precisión
epsilon_m = np.finfo(float).eps

# Calcular la tolerancia como la raíz cúbica del épsilon de la máquina
tolerancia = epsilon_m ** (1/3)

for filename in glob.glob(path):

    # Eliminar la ruta completa y solo conservar el nombre del archivo
    base_name = os.path.basename(filename)
    name_without_ext = os.path.splitext(base_name)[0]
```

```
    # Resolver el sistema
    A, b, x = Minimizer(name_without_ext)

    # Definición de n
    n = len(b)

    # Definición de x0
    x0 = np.ones(n)*10

    # Define el número máximo de iteraciones N
    N = 1000

    # Calcula el punto x_k con el algoritmo de descenso máximo
    x_k, num_iteraciones, convergio = DescensoMax_Func2(A, b, x0, tolerancia, N)

    # Calcula y muestra los valores solicitados
    valor_inicial = f(A, b, x0)
    valor_final = f(A, b, x_k)
    norma_diferencia = np.linalg.norm(x_k - x)

    print("Solución de sistema " + name_without_ext)

    print(f"Valor de n: {n}")
    print(f"f(x0): {valor_inicial}")
    print(f"Número de iteraciones (k): {num_iteraciones}")
    print(f"f(x_k): {valor_final}")
    print(f"Norma de la diferencia ||x_k - x_*||: {norma_diferencia}")

    # Imprimir x_k
    if n <= 6:
        print(f"x_k: {x_k}")
    else:
        # Imprime los primeros y últimos tres elementos de x_k
        print(f"x_k (primeros 3 elementos): {x_k[:3]}")
        print(f"x_k (últimos 3 elementos): {x_k[-3:]}")
```

```
Solución de sistema matA_vecb4
Valor de n: 500
f(x0): 543542.6
Número de iteraciones (k): 77
f(x_k): -436.1899999992252
Norma de la diferencia ||x_k - x_*||: 3.532614479400813e-05
x_k (primeros 3 elementos): [ 0.99999929 -1.00000012  0.99999889]
x_k (últimos 3 elementos): [-0.999998    0.99999656 -0.99999944]
Solución de sistema matA_vecb3
Valor de n: 100
f(x0): 17984.4
```

```
Número de iteraciones (k): 5
f(x_k): -149.8699999999988
Norma de la diferencia ||x_k - x_*||: 8.944448733815078e-07
x_k (primeros 3 elementos): [-0.99999999 -1.00000018 -1.00000012]
x_k (últimos 3 elementos): [-0.99999997 -0.99999993 -1.00000015]
Solución de sistema matA_vecb2
Valor de n: 10
f(x0): 2626.0
Número de iteraciones (k): 65
f(x_k): -32.824999999800625
Norma de la diferencia ||x_k - x_*||: 2.7041631131850196e-05
x_k (primeros 3 elementos): [0.99999604 0.99999779 1.00001441]
x_k (últimos 3 elementos): [0.99998744 1.00000671 0.99999931]
Solución de sistema matA_vecb1
Valor de n: 2
f(x0): 1110.0
Número de iteraciones (k): 4
f(x_k): -3.1499999999999706
Norma de la diferencia ||x_k - x_*||: 7.708290085177699e-08
x_k: [-0.29999995 -0.79999994]
```

1.2.3 Ejercicio 2.4

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

Los números de iteraciones calculados fueron:

Solución de sistema matA_vecb4: Número de iteraciones calculado: 209.0

Solución de sistema matA_vecb3: Número de iteraciones calculado: 6.0

Solución de sistema matA_vecb2: Número de iteraciones calculado: 109.0

Solución de sistema matA_vecb1: Número de iteraciones calculado: 40.0

La única buen aproximación fue la del sistema 3, ya que se calcularon 6 iteraciones y el valor real fue de 5. Para los demás casos el número de iteraciones calculado dista bastante del real. Sin embargo, si nos referimos únicamente al número calculado como cota superior, éste si fue un buen parámetro de referencia ya que en todos los casos las iteraciones calculadas fueron un número superior a las reales, siendo un buen estimador para este valor.

1.3 Ejercicio 3

Programar el algoritmo 1 de la clase 5 de descenso máximo, usando el método de la sección dorada para obtener $\alpha_k \epsilon [0, 1]$

$$a_k = \arg \min_{\alpha_k \epsilon [0,1]} f(x_k - \alpha \nabla f(x_k))$$

1.3.1 Ejercicio 3.1

La función que implementa el algoritmo recibe como entrada: - La función $f(x)$, - el gradiente $\nabla f(x)$ de la función f, - un punto inicial x_0, - las tolerancia $\tau_1 > 0$ y $\tau_2 > 0$, - el número máximo de iteraciones N para el algoritmo de descenso máximo, y - el número máximo de iteraciones N_{gs} para el método de la sección dorada.

La función devuelve - El último punto 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 ($\|\alpha_k p_k\| < \tau_1$) o False si termina porque se alcanzó el número máximo de iteraciones. Un arreglo que contiene la secuencia de puntos $x_0, x_1, \ldots x_k$ si la dimensión de la variable es $n = 2$, y es vacío en otro caso. Es decir, sólo cuando la dimensión guardamos la secuencia de puntos.

Dentro de esta función se puede definir $phi(\alpha) = f\left(x_k - \alpha \nabla f\left(x_k\right)\right)$ y usar el algoritmo de la sección dorada de la Tarea 1 para calcular su minimizador α_k en el intervalo [0,1]. Para hacer esto, puede usar una función lambda como en el ejemplo en las notas de la Ayudantía 2. Use la tolerancia τ_2 para el algoritmo de la sección dorada.

[14]:
```python
def golden_section(f, xMin, xMax, tol, nMax):
    rho = (np.sqrt(5) - 1) / 2
    xk = None  # Inicializar xk

    for k in range(nMax):
        b = (1 - rho) * (xMax - xMin)
        x1 = xMin + b
        x3 = xMax - b

        if f(x1) < f(x3):
            xMax = x3
            xk = x1
        elif f(x1) > f(x3):
            xMin = x1
            xk = x3

        if (xMax - xMin) < tol:
            return xk, True

    return xk, False
```

[15]:
```python
def DescensoMax(f, gradf, x0, tau1, tau2, NMax, NGolden):
    xk = np.array(x0)
    sequence = []

    for k in range(NMax):
        gk = gradf(xk)
        pk = -gk

        # Definir phi(alpha) para el método de la sección dorada
        phi = lambda alpha: f(xk + alpha * pk)
```

```
        # Calcular tamaño de paso usando el método de la sección dorada
        alpha_k, convergence = golden_section(phi, 0, 1, tau2, NGolden)

        # Verificar la condición de parada
        if np.linalg.norm(alpha_k * pk) < tau1:
            return xk, k, True, sequence

        # Actualizar xk para la siguiente iteración
        xk = xk + alpha_k * pk

        # Guardar la secuencia de puntos si la dimensión de x es 2
        if len(x0) == 2:
            sequence.append(xk.tolist())

    return xk, NMax, False, sequence
```

1.3.2 Ejercicio 3.2

Para probar el algoritmo, programe las siguientes funciones, calcule su gradiente de manera analítica y programe la función correspondiente. Use cada punto x_0 como punto inicial del algoritmo.

Función de Himmelblau: Para $x = (x_1, x_2)$

$$f(x) = \left(x_1^2 + x_2 - 11\right)^2 + \left(x_1 + x_2^2 - 7\right)^2.$$
$$x_0 = (2., 4.)$$
$$x_0 = (0., 0.)$$

Función de Beale : Para $x = (x_1, x_2)$

$$f(x) = \left(1.5 - x_1 + x_1 x_2\right)^2 + \left(2.25 - x_1 + x_1 x_2^2\right)^2 + \left(2.625 - x_1 + x_1 x_2^3\right)^2.$$
$$x_0 = (2., 3.)$$
$$x_0 = (2., 4.)$$

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

$$f(x) = \sum_{i=1}^{n-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(1 - x_i\right)^2\right] \quad n \geq 2$$
$$x_0 = (-2.1, 4.5)$$
$$x_0 = (-1.2, 1.0)$$
$$x_0 = (-2.1, 4.5, -2.1, 4.5, -2.1, 4.5, -2.1, 4.5, -2.1, 4.5)$$
$$x_0 = (-1.2, 1.0, -1.2, 1.0, -1.2, 1.0, -1.2, 1.0, -1.2, 1.0)$$

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.

Para las funciones de dos variables grafique los contornos de nivel. Modifique la función contornosFnc2D, o haga la suya, y pase como argumento la secuencia de puntos que devuelve el algoritmo para visualizar la trayectoria de los puntos x_k.

[16]:
```
def himmelblau(x):
    return (x[0]**2 + x[1] - 11)**2 + (x[0] + x[1]**2 - 7)**2

def grad_himmelblau(x):
    df_dx1 = 4 * x[0] * (x[0]**2 + x[1] - 11) + 2 * (x[0] + x[1]**2 - 7)
    df_dx2 = 2 * (x[0]**2 + x[1] - 11) + 4 * x[1] * (x[0] + x[1]**2 - 7)
    return np.array([df_dx1, df_dx2])
```

[17]:
```
def beale(x):
    return ((1.5 - x[0] + x[0]*x[1])**2 +
            (2.25 - x[0] + x[0]*x[1]**2)**2 +
            (2.625 - x[0] + x[0]*x[1]**3)**2)

def grad_beale(x):
    x1, x2 = x
    df_dx1 = 2*(1.5 - x1 + x1*x2)*(-1 + x2) + 2*(2.25 - x1 + x1*x2**2)*(-1 +␣
 ↪x2**2) + 2*(2.625 - x1 + x1*x2**3)*(-1 + x2**3)
    df_dx2 = 2*(1.5 - x1 + x1*x2)*x1 + 2*(2.25 - x1 + x1*x2**2)*2*x1*x2 + 2*(2.
 ↪625 - x1 + x1*x2**3)*3*x1*x2**2
    return np.array([df_dx1, df_dx2])
```

[18]:
```
def rosenbrock(x):
    return sum(100*(x[1:] - x[:-1]**2)**2 + (1 - x[:-1])**2)

def grad_rosenbrock(x):
    df_dx = np.zeros_like(x)
    n = len(x)
    df_dx[:-1] += -400 * x[:-1] * (x[1:] - x[:-1]**2) + 2 * (x[:-1] - 1)  #␣
 ↪Derivadas parciales para x_i donde i < n
    df_dx[1:] += 200 * (x[1:] - x[:-1]**2)  # Derivadas parciales para x_{i+1}␣
 ↪donde i < n
    return df_dx
```

[19]:
```
# Puntos iniciales para la función de Himmelblau
puntos_iniciales_himmelblau = [np.array([2.0, 4.0]), np.array([0.0, 0.0])]

# Puntos iniciales para la función de Beale
puntos_iniciales_beale = [np.array([2.0, 3.0]), np.array([2.0, 4.0])]

# Puntos iniciales para la función de Rosenbrock
puntos_iniciales_rosenbrock = [
    np.array([-2.1, 4.5]),
    np.array([-1.2, 1.0]),
```

```
    np.array([-2.1, 4.5, -2.1, 4.5, -2.1, 4.5, -2.1, 4.5, -2.1, 4.5]),
    np.array([-1.2, 1.0, -1.2, 1.0, -1.2, 1.0, -1.2, 1.0, -1.2, 1.0])
]
# Epsilon de la máquina
epsilon_m = np.finfo(float).eps

n = 2  # Dimensión del problema

# Configuración de tolerancias
tau1 = np.sqrt(n) * epsilon_m**(1/3)
tau2 = epsilon_m**(1/2)

# Número máximo de iteraciones para el descenso máximo y la sección dorada
NMax = 10000
NGolden = 200

# Función para probar el algoritmo de descenso máximo con diferentes funciones
def probar_descenso_maximo(func, grad_func, puntos_iniciales):
    for x0 in puntos_iniciales:
        xk, k, convergio, secuencia = DescensoMax(func, grad_func, x0, tau1,␣
 ↪tau2, NMax, NGolden)
        valor_final = func(xk)
        print(f"Resultado para x0 = {x0}:")
        print(f"xk = {xk}, k = {k}, f(xk) = {valor_final}, convergió:␣
 ↪{convergio}")
        if len(x0) == 2 and secuencia:
            print(f"Secuencia de puntos: {secuencia[:50]}")
        print()

# Probar con la función de Himmelblau
print("Función de Himmelblau:")
probar_descenso_maximo(himmelblau, grad_himmelblau, puntos_iniciales_himmelblau)

# Probar con la función de Beale
print("Función de Beale:")
probar_descenso_maximo(beale, grad_beale, puntos_iniciales_beale)

# Probar con la función de Rosenbrock
print("Función de Rosenbrock:")
probar_descenso_maximo(rosenbrock, grad_rosenbrock, puntos_iniciales_rosenbrock)
```

```
Función de Himmelblau:
Resultado para x0 = [2. 4.]:
xk = [ 3.58442714 -1.84812629], k = 9, f(xk) = 7.454787957081565e-11, convergió:
True
Secuencia de puntos: [[2.0185580695549272, 2.422564087831168],
[2.870337456772578, 2.4325844636708696], [2.9191483454738703,
```

```
-1.7166015771488774], [3.574439050164983, -1.708892743369137],
[3.576057814716588, -1.8464910871957199], [3.5843078523023912,
-1.846394028173136], [3.584328001207576, -1.848106925422064],
[3.5844268967813675, -1.848105762159447], [3.584427138307729,
-1.8481262921423067]]

Resultado para x0 = [0. 0.]:
xk = [3.00000017 2.00000628], k = 14, f(xk) = 6.931124616723682e-10, convergió:
True
Secuencia de puntos: [[1.7826996201516425, 2.8013851173811526],
[3.000848202031971, 2.0261996931506445], [2.991698168276046, 2.011821074595092],
[3.000180444353276, 2.006423257085746], [2.997956346572549, 2.0029282479158836],
[3.0000432883874657, 2.001600191349801], [2.999490379665525, 2.000731336556288],
[3.000010722947768, 2.0004002085776107], [2.9998725129405095,
2.0001830219593044], [3.000002677866644, 2.000100189891608], [2.999968082492123,
2.000045825744042], [3.000000670260779, 2.000025087931141], [2.999992007634209,
2.000011475415287], [3.0000001678546857, 2.000006282455451]]

Función de Beale:
Resultado para x0 = [2. 3.]:
xk = [2.99949463 0.49986994], k = 476, f(xk) = 4.1415964238432026e-08,
convergió: True
Secuencia de puntos: [[0.6844585065291056, 0.3478566721419707],
[1.7722451996156143, -0.19175753391447392], [1.9809538648594986,
0.2289698215613501], [2.157051451783589, 0.1416137332870706],
[2.226970610606555, 0.2825609042317759], [2.315506954123908,
0.23864100521381726], [2.35438406585313, 0.3170117952611238],
[2.411356916759044, 0.2887494860106898 4], [2.4372823612526187,
0.34101154490783964], [2.4785061012636604, 0.3205618390533924],
[2.4975266451125147, 0.35890459442217154], [2.5294537379865267,
0.3430666462558246], [2.5442547074879123, 0.372903332960256],
[2.570096631924094, 0.360084030489911], [2.582082176504773,
0.38424521269938117], [2.6036527358509605, 0.3735447896258303],
[2.613640725444031, 0.39367918176100114], [2.6320583570997607,
0.3845428193434496], [2.6405633041653376, 0.4016876071156643],
[2.6565636837621973, 0.3937503628084198], [2.663928532047774,
0.4085968736085817], [2.678020118292692, 0.40160651702976335],
[2.684483991999971, 0.4146367844663322], [2.6970322359543237,
0.40841202693486306], [2.702767992270387, 0.4199745124874592],
[2.7140440145747506, 0.41438086114093425], [2.7191803651801485,
0.4247350280395374], [2.729390644784147, 0.4196700560319988],
[2.7340258221690847, 0.4290139277301565], [2.7433311056796787,
0.42439789357086755], [2.7475416637720285, 0.43288579283319956],
[2.75606946133565, 0.4286554421987429], [2.7599161363388487,
0.4364098059464688], [2.7677692411749524, 0.4325141502946219],
[2.7713009778824547, 0.4396336443112366], [2.7785634243312387,
0.4360309943601702], [2.7818202034423583, 0.4425962110693758],
[2.7885615498625484, 0.43925206110518306], [2.7915764213649346,
```

```
0.4453296273344973], [2.797854870567872, 0.44221510476370823],
[2.8006554512373705, 0.44786068901729287], [2.8065201862164604,
0.44495139630379815], [2.8091297693278405, 0.4502119594536769],
[2.8146227085789373, 0.4474871033384285], [2.817061117152004,
0.45240260188804055], [2.822218270849003, 0.4498443148936565],
[2.8245024995106776, 0.4544490060242888], [2.8293550279537167,
0.4520418325484623], [2.8314997524517946, 0.4563652991008731],
[2.83607479274064, 0.4540957767098917]]

Resultado para x0 = [2. 4.]:
xk = [-5.01457951  1.16971512], k = 10000, f(xk) = 0.7328929924732723,
convergió: False
Secuencia de puntos: [[-0.002701263290723954, 1.0041912894090932],
[-0.11936432252267205, 1.0796097596913516], [-0.5955393508205068,
1.8343070522942764], [-1.0233269703225027, 1.5643957955117058],
[-1.0343912563166113, 1.5819317979483762], [-1.096513739589183,
1.5427359054577021], [-1.1049352977484286, 1.5560834295102224],
[-1.1508579787231812, 1.5271086060600954], [-1.1578983993863696,
1.5382670873644313], [-1.1950111355815978, 1.5148508881179998],
[-1.2011676413238344, 1.5246084372008604], [-1.2326422364298883,
1.5047495424988737], [-1.2381703487513032, 1.5135111114501223],
[-1.2656858780407372, 1.4961501912248971], [-1.2707374489952297,
1.504156485784426], [-1.295299028188795, 1.488659358813437],
[-1.2999730112122894, 1.496067210199968], [-1.3222341692238642,
1.4820215010011988], [-1.3265993232225137, 1.4889398692555924],
[-1.3470113886222563, 1.4760608563320794], [-1.3511178232751122,
1.4825691838513184], [-1.37000635190309, 1.470651452874136],
[-1.373891919327564, 1.4768097263366153], [-1.39150029769794,
1.4656997085198231], [-1.3951944341728322, 1.4715545809160409],
[-1.4117097831775027, 1.4611341980027073], [-1.4152359165482151,
1.4667227964873135], [-1.4308057244073826, 1.4568990372184243],
[-1.4341828766897413, 1.4622515307820325], [-1.4489252480300918,
1.4529498520161863], [-1.4521691046021887, 1.4580910885509837],
[-1.4661803204213213, 1.4492507008539903], [-1.4693039975802908,
1.4542014474293432], [-1.482663952625662, 1.4457719617943265],
[-1.485678561121688, 1.4505498377565331], [-1.498454198438662,
1.4424890371067773], [-1.5013692449221963, 1.4471091197771642],
[-1.5136171683932569, 1.4393812855021788], [-1.5164408594801304,
1.4438565777722812], [-1.5282094664216492, 1.4364311596199],
[-1.5309489452456495, 1.4407729845421686], [-1.5422800955434968,
1.4336236030920981], [-1.5449416300732237, 1.4378419061224876],
[-1.5558715820666167, 1.430945677345753], [-1.5584607033672657,
1.4350492208458931], [-1.5690211494677853, 1.4283861325149758],
[-1.5715427695162865, 1.4323826927307113], [-1.5817615700069565,
1.4259351404117475], [-1.5842200708605525, 1.4298316465400298],
[-1.5941220600447934, 1.423583978930241]]

Función de Rosenbrock:
```

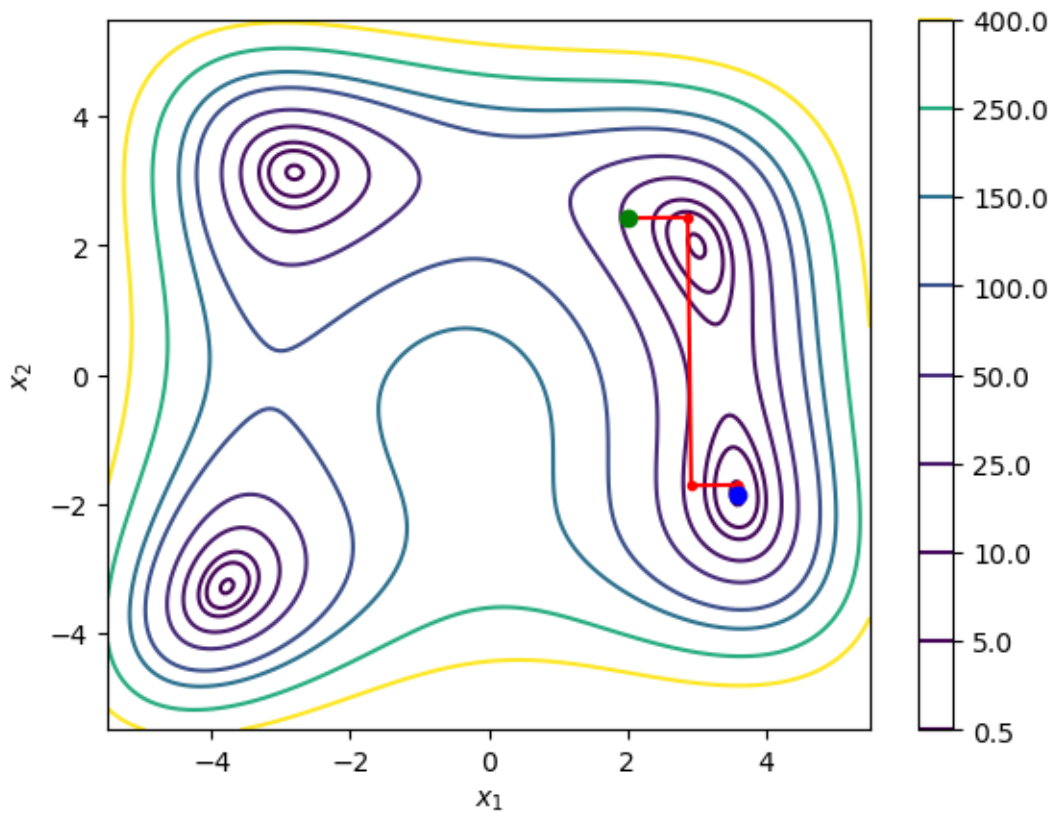
```
Resultado para x0 = [-2.1  4.5]:
xk = [1.00299222 1.00602105], k = 3179, f(xk) = 9.029857281261538e-06,
convergió: True
Secuencia de puntos: [[-2.1185499709497018, 4.495188768341576],
[-1.7581287865797026, 3.1082309920299895], [-1.760699094342843,
3.106888103020637], [-1.7541747059602244, 3.0944015779678975],
[-1.756763636214544, 3.093048824557439], [-1.7501703971212763,
3.080430727217293], [-1.752778854746042, 3.079067749472969],
[-1.746112954857701, 3.066309681340378], [-1.7487419562679452,
3.0649360673274226], [-1.7420047462342174, 3.052042700860551],
[-1.744653868733393, 3.0506584469435083], [-1.7378426645781617,
3.0376243162031407], [-1.7405125737251277, 3.036229110171517],
[-1.733624801782661, 3.0230491015367056], [-1.7363161356266406,
3.021642630847443], [-1.7293498182286415, 3.0083131988914165],
[-1.732063047713187, 3.0068951929192687], [-1.7250157956185093,
2.993411818041702], [-1.7277514590792775, 2.991981989657714],
[-1.7206171157905343, 2.978330810755397], [-1.7233768054349499,
2.976888549035711], [-1.7161556317917321, 2.963071878657105],
[-1.7189392022491738, 2.961617067518707], [-1.7116276845111156,
2.947627833907217], [-1.7144359721348543, 2.9461600733819906],
[-1.7070296157077456, 2.9319888630632884], [-1.709863724671792,
2.9305076613295147], [-1.7023601887073216, 2.9161505492612223],
[-1.70522064091529, 2.9146555753667953], [-1.6976160279916042,
2.900104859605525], [-1.7005037700317998, 2.898595644537179],
[-1.6927971298448388, 2.8838510634082715], [-1.6957122888795,
2.8823273793228004], [-1.6878965490135793, 2.867373563156664],
[-1.6908408291013652, 2.8658347099331434], [-1.6829104185918526,
2.850660203103491], [-1.6858851459311013, 2.8491055687102156],
[-1.6778370479411964, 2.8337054118676166], [-1.6808428893723228,
2.8321345639137103], [-1.6726739017563974, 2.816503468931107],
[-1.6757115403587204, 2.814915964461272], [-1.6674155350595437,
2.799041571373638], [-1.670486393684527, 2.7974367315272146],
[-1.6620589562065087, 2.781311110187854], [-1.6651639969317495,
2.7796883797907554], [-1.6565982814406037, 2.7632978044205116],
[-1.6597390538755776, 2.7616564364610974], [-1.6510278624417638,
2.74498656804925], [-1.654206002386274, 2.7433257640292306],
[-1.645344301304522, 2.7263678368666064]]

Resultado para x0 = [-1.2  1. ]:
xk = [1.00358886 1.007213  ], k = 8323, f(xk) = 1.2930107888125386e-05,
convergió: True
Secuencia de puntos: [[1.4408777658438983, 2.077909292181183],
[1.4410565052965465, 2.077468708302403], [1.4406707440005162,
2.077312209944801], [1.4408494567883599, 2.0768716913217675],
[1.4404637185073346, 2.076715202007373], [1.4406424044763737,
2.0762747479433887], [1.4402566880466419, 2.076118267661157],
[1.4404353478329481, 2.0756778801739095], [1.440049653600086,
2.075521408074461], [1.4402282876721195, 2.075081087076356], [1.43984261528024,
```

```
2.074924623379759], [1.4400212233357033, 2.074484370385742],
[1.4396355744145264, 2.074327914623088], [1.4398141564467988,
2.0738877262150037], [1.4394285308034473, 2.0737312807091732],
[1.4396070854115692, 2.0732911561154164], [1.4392214815712165,
2.0731347201396266], [1.4394000099115931, 2.072694662528574],
[1.4390144289151527, 2.0725382349934707], [1.4391929307585773,
2.0720982445196907], [1.43880737294772, 2.0719418254033184],
[1.4389858483146263, 2.0715019013109157], [1.4386003130310165,
2.0713454909385813], [1.4387787621636072, 2.070905634042673],
[1.4383932500280074, 2.070749232295548], [1.4385716729080213,
2.0703094413453935], [1.4381861839304375, 2.0701530491399556],
[1.4383645800746958, 2.0697133235563863], [1.4379791158682327,
2.0695569417600694], [1.438157483934101, 2.069117278600942],
[1.4377720416831477, 2.0689609074649407], [1.4379503830204792,
2.0685213111555654], [1.4375649652030147, 2.0683649494066962],
[1.4377432790812812, 2.067925417615932], [1.437357883021948,
2.0677690654582777], [1.4375361712781034, 2.067329601107307],
[1.4371507984290286, 2.0671732581331193], [1.4373290595853894,
2.066733861810654], [1.4369437115365162, 2.0665775275640836],
[1.437121946009127, 2.066138194899796], [1.4367366213864605, 2.065981872105646],
[1.4369148279600137, 2.06554260442158], [1.4365295270765612,
2.0653862917383456], [1.4367077061301443, 2.0649470903087535],
[1.4363224288424488, 2.064790787062104], [1.4365005813240093,
2.0643516516341953], [1.4361153267970912, 2.0641953582116312],
[1.4362934525012327, 2.0637562910813028], [1.4359082230144078,
2.063600006786931], [1.4360863212781627, 2.063161004786324]]

Resultado para x0 = [-2.1  4.5 -2.1  4.5 -2.1  4.5 -2.1  4.5 -2.1  4.5]:
xk = [0.99998159 0.99996448 0.99992621 0.99985562 0.99970639 0.99941575
 0.99882483 0.99764861 0.99528853 0.9905769 ], k = 7987, f(xk) =
2.9644332551148124e-05, convergió: True

Resultado para x0 = [-1.2  1.  -1.2  1.  -1.2  1.  -1.2  1.  -1.2  1. ]:
xk = [0.99998114 0.99996369 0.99992439 0.99985228 0.99969932 0.99940196
 0.99879684 0.99759288 0.99517684 0.99035414], k = 7846, f(xk) =
3.106780923694802e-05, convergió: True
```

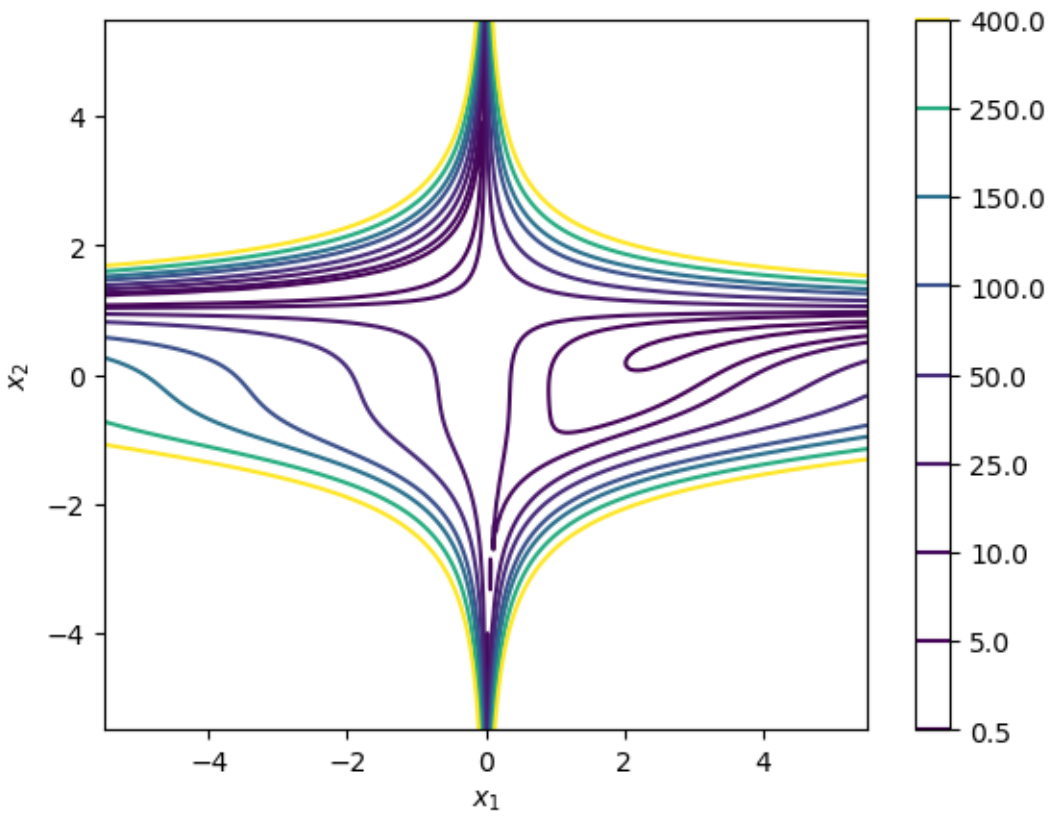
[20]:
```python
def contornosFnc2D(fncf, xleft, xright, ybottom, ytop, levels, secuencia=None):
    ax = np.linspace(xleft, xright, 250)
    ay = np.linspace(ybottom, ytop, 200)
    mX, mY = np.meshgrid(ax, ay)
    mZ = np.array([[fncf(np.array([x, y])) for x in ax] for y in ay])

    fig, ax = plt.subplots()
    CS = ax.contour(mX, mY, mZ, levels, cmap='viridis')
    plt.colorbar(CS, ax=ax)
```

```
    ax.set_xlabel('$x_1$')
    ax.set_ylabel('$x_2$')

    # Graficar la secuencia de puntos
    if secuencia is not None:
        secuencia = np.array(secuencia)
        ax.plot(secuencia[:, 0], secuencia[:, 1], 'r.-')  # 'r.-' para puntos␣
 ↪rojos conectados por líneas
        ax.plot(secuencia[0, 0], secuencia[0, 1], 'go')  # Punto de inicio en␣
 ↪verde
        ax.plot(secuencia[-1, 0], secuencia[-1, 1], 'bo')  # Punto final en azul

    plt.show()
```

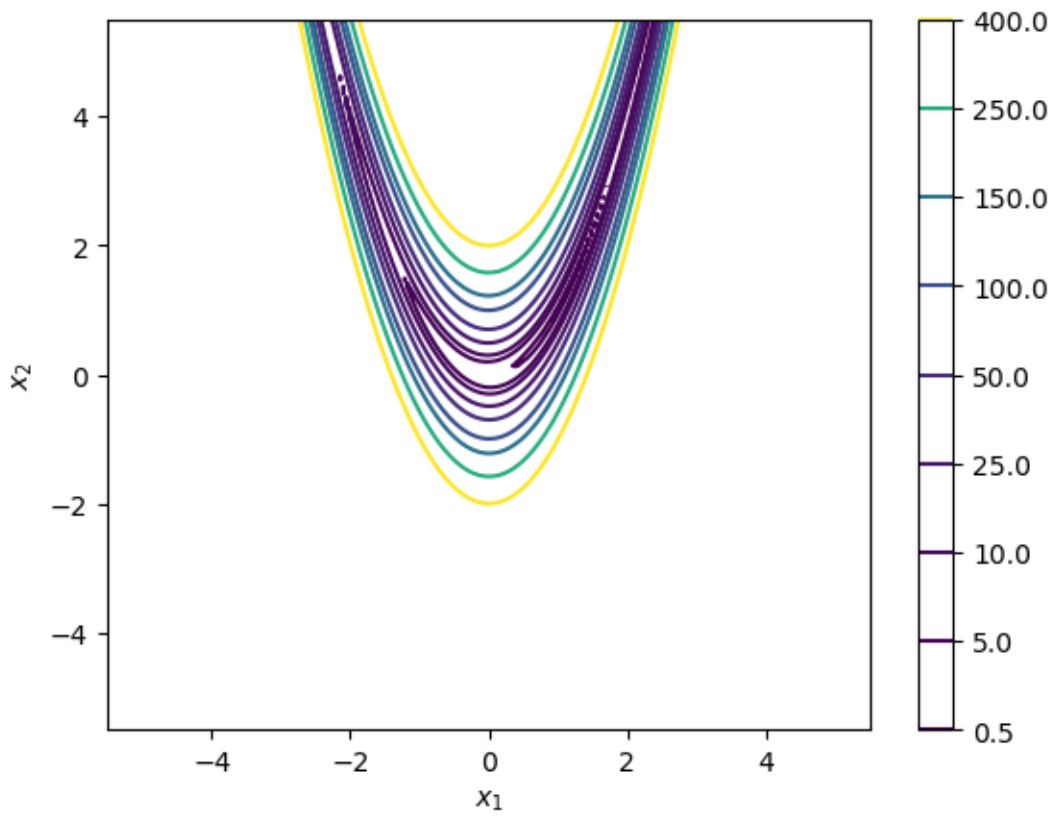
[21]:
```
# Función de Himmelblau
secuencia = np.array([
    [2.01855807, 2.42256409],
    [2.87033746, 2.43258446],
    [2.91914835, -1.71660158],
    [3.57443905, -1.70889274],
    [3.57605781, -1.84649109],
    [3.58430785, -1.84639403],
    [3.584328, -1.84810693],
    [3.5844269, -1.84810576],
    [3.58442714, -1.84812629]
])
contornosFnc2D(himmelblau, xleft=-5.5, xright=5.5, ybottom=-5.5, ytop=5.5,␣
 ↪levels=[0.5, 5, 10, 25, 50, 100, 150, 250, 400], secuencia=secuencia)
```



[22]:
```
# Función de Beale
secuencia = []
contornosFnc2D(beale, xleft=-5.5, xright=5.5, ybottom=-5.5, ytop=5.5, levels=[0.
↪5, 5, 10, 25, 50, 100, 150, 250, 400], secuencia=None)
```



[23]:
```
# Función de Rosenbrock
secuencia = []
contornosFnc2D(rosenbrock, xleft=-5.5, xright=5.5, ybottom=-5.5, ytop=5.5,␣
↪levels=[0.5, 5, 10, 25, 50, 100, 150, 250, 400], secuencia=None)
```



1.3.3 Ejercicio 3.3

Repita la prueba para función de Rosenbrock usando el punto inicial $x_0 = (-2.1, 4.5)$ 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.

[24]:
```
# Puntos iniciales para la función de Rosenbrock
puntos_iniciales_rosenbrock = [
    np.array([-2.1, 4.5])]

# Epsilon de la máquina
epsilon_m = np.finfo(float).eps

n = 2  # Dimensión del problema

# Configuración de tolerancias
tau1 = np.sqrt(n) * epsilon_m**(1/3)
tau2 = epsilon_m**(1/4)

# Número máximo de iteraciones para el descenso máximo y la sección dorada
```

```python
NMax = 10000
NGolden = 50

# Función para probar el algoritmo de descenso máximo con diferentes funciones
def probar_descenso_maximo(func, grad_func, puntos_iniciales):
    for x0 in puntos_iniciales:
        xk, k, convergio, secuencia = DescensoMax(func, grad_func, x0, tau1,␣
 ↪tau2, NMax, NGolden)
        valor_final = func(xk)
        print(f"Resultado para x0 = {x0}:")
        print(f"xk = {xk}, k = {k}, f(xk) = {valor_final}, convergió:␣
 ↪{convergio}")
        if len(x0) == 2 and secuencia:
            print(f"Secuencia de puntos: {secuencia[:50]}")
        print()

# Probar con la función de Rosenbrock
print("Función de Rosenbrock:")
probar_descenso_maximo(rosenbrock, grad_rosenbrock, puntos_iniciales_rosenbrock)
```

```
Función de Rosenbrock:
Resultado para x0 = [-2.1  4.5]:
xk = [1.01786922 1.03607662], k = 10000, f(xk) = 0.0003193447538303153,
convergió: False
Secuencia de puntos: [[-2.1194343305958374, 4.494959395522693],
[-2.1183893667638776, 4.494793754885195], [-2.116981748234764,
4.477663847822244], [2.198121798059142, 4.8334223652524635],
[2.1978045488511486, 4.833305859458063], [2.197978863165252, 4.832808347678439],
[2.197666043182985, 4.832690849249358], [2.197849227466463, 4.832155314587567],
[2.197511155560805, 4.832043574334124], [2.1976963427138134, 4.831565898643821],
[2.1973834751674133, 4.831448440428732], [2.1975664639837866,
4.8309129979101195], [2.197228690880421, 4.830801219832426], [2.197413360257517,
4.830323703306146], [2.1971010917712457, 4.830206138230426], [2.197283196731332,
4.829670944506045], [2.1969464275835295, 4.829558968187829],
[2.1971299179128145, 4.829081761334951], [2.1968188916916045,
4.828963943044999], [2.1969994297907944, 4.828429153600929], [2.196664363525053,
4.828316819980406], [2.1968460210556215, 4.827840071656647], [2.196536872303455,
4.827721855562505], [2.1967151707803176, 4.827187623613431], [2.196382495128169,
4.8270747761142365], [2.196561678267119, 4.82659863246533], [2.196255029686957,
4.82647987676543], [2.196438338727599, 4.825922298230658], [2.196090842264761,
4.825812853963053], [2.1962677139938336, 4.825373284333643],
[2.1959474540734316, 4.825235404089944], [2.196139106445254, 4.824828491082063],
[2.1958257449527805, 4.824689054845999], [2.196010529535088, 4.824283720084222],
[2.195704016230975, 4.824142739108937], [2.1958887236140963, 4.82372368583005],
[2.195573746754131, 4.823584645638046], [2.1957599611143164, 4.823179015307927],
[2.195452121155307, 4.823038364501367], [2.1956380215855766, 4.822619070862498],
[2.1953219283792706, 4.822480313201772], [2.1955090802358335, 4.82207449933464],
```

```
[2.1952004018911633, 4.821934067610004], [2.195386995920431, 4.821514647293269],
[2.1950702920604575, 4.821376057093339], [2.195257883459843, 4.820970173050857],
[2.194948860107232, 4.82082984812021], [2.195135643975993, 4.820410415828772],
[2.194818839000171, 4.820271877104217], [2.195006369107195, 4.819866036943375]]
```

Cuando se relajan las condiciones de paro del método de la sección dorada, como en el caso de usar $ _2 = m^{1/4} $ y $ N{gs} = 50 $, es posible que el algoritmo termine más rápidamente porque se requiere una menor precisión para detener la búsqueda del tamaño de paso óptimo $ _k $. Sin embargo, este enfoque puede tener varias implicaciones ya que al relajar las condiciones de paro, existe el riesgo de que el tamaño de paso $ _k $ no sea lo suficientemente preciso, lo que podría llevar a que el algoritmo de descenso máximo no converja al mínimo real de la función objetivo, lo que parece que ocurre en este caso ya que el algoritmo no convergió.

En este caso específico, el hecho de que el algoritmo no convergiera al relajar las condiciones de paro del método de la sección dorada indica que, para la función de Rosenbrock y el punto inicial dado, es crucial una mayor precisión en la determinación del tamaño de paso $ _k $. Esto subraya la importancia de elegir cuidadosamente los parámetros del algoritmo, en particular las condiciones de paro, en función de las propiedades de la función objetivo y los requisitos del problema específico. En problemas donde es esencial encontrar un mínimo preciso o cuando se trabaja con funciones complicadas como la de Rosenbrock, relajar demasiado las condiciones de paro puede llevar a resultados insatisfactorios o a la falta de convergencia.

1.4 Ejercicio 4

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

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

con $0 < \alpha < 0.5$, para obtener el minimizador de la función $f(x)$. Muestre que la secuencia $\{x_k\}$ converge a 1. ¿Tiene este algoritmo la propiedad de descenso, es decir, $f(x_{k+1}) < f(x_k)$ a partir de un cierto k ?

Para abordar este problema, primero definimos la función y su derivada, y luego analizamos la secuencia y la convergencia.

Dada la función $ f(x) = (x - 1)^2 $, su derivada es:

$$f'(x) = 2(x-1)$$

La secuencia dada es:

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

Sustituyendo la derivada de $ f(x) $ en la secuencia, obtenemos:

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

Para mostrar que x_k converge a 1, podemos analizar el comportamiento de la secuencia a medida que $ k $ se hace grande.

A medida que $ k $ aumenta, el término $ /2^{k-1} $ se hace muy pequeño debido a la condición $ 0 < < 0.5 $ y al factor $ 2^{k-1} $ en el denominador. Esto significa que el ajuste que se hace a $ x_k $ en cada paso se reduce exponencialmente. Por lo tanto, los cambios en $ x_k $ se vuelven insignificantes, y $ x_k $ se estabiliza.

Además, el signo de $ (x_k - 1) $ asegura que el ajuste siempre empuje $ x_k $ hacia 1: si $ x_k > 1 $, entonces $ x_{k+1} < x_k $; si $ x_k < 1 $, entonces $ x_{k+1} > x_k $. Por lo tanto, independientemente del valor inicial $ x_0 $, la secuencia x_k se mueve hacia 1 y, debido a la reducción exponencial del término de ajuste, eventualmente converge a 1.

Para determinar si el algoritmo tiene la propiedad de descenso, necesitamos verificar si $ f(x_{k+1}) < f(x_k) $ a partir de un cierto $ k $.

Dado que la función $ f(x) $ es estrictamente convexa y su mínimo se alcanza en $ x = 1 $, cualquier movimiento hacia 1 reduce el valor de $ f(x) $. Sin embargo, debido a la reducción exponencial del término de ajuste $ /2^{k-1} (x_k - 1) $, puede haber un punto a partir del cual los ajustes sean tan pequeños que, debido a la precisión numérica o a la elección de $ $, el valor de $ f(x_{k+1}) $ no sea estrictamente menor que $ f(x_k) $ para algunos $ k $. Esto dependerá de la precisión numérica de la implementación y del valor específico de $ $.

Sin embargo, para valores de $ k $ lo suficientemente pequeños y ajustes significativos, el algoritmo debería exhibir la propiedad de descenso debido a la naturaleza convexa de $ f(x) $ y al hecho de que cada paso mueve $ x_k $ más cerca del mínimo en $ x = 1 $, reduciendo $ f(x_k) $. A medida que $ x_k $ se acerca a 1, la propiedad de descenso puede no mantenerse estrictamente en cada paso debido a la disminución del tamaño del paso.

1.5 Referencias

[1] Burden, Richard L. Numerical analysis. Brooks/Cole Cengage Learning, 2011.