Curso de Optimización (DEMAT)

Tarea 3

Leslie Janeth Quincosa Ramírez

Nota: No sé porqué al pasar a pdf el notebook, se ven tan mal las ecuaciones. En el ipynb sí se ve bien todo.

Descripción:	Fechas
Fecha de publicación del documento:	**Febrero 18, 2022**
Fecha límite de entrega de la tarea:	**Febrero 27, 2022**

Indicaciones

Puede escribir el código de los algoritmos que se piden en una celda de este notebook o si lo prefiere, escribir las funciones en un archivo `.py` independiente e importar la funciones para usarlas en este notebook. Lo importante es que en el notebook aparezcan los resultados de la pruebas realizadas y que:

- **Si se requieren otros archivos para poder reproducir los resultados, para mandar la tarea cree un archivo ZIP en el que incluya el notebook y los archivos adicionales.**
- **Si todos los códigos para que se requieren para reproducir los resultados están en el notebook, no hace falta comprimirlo y puede anexar sólo el notebook en la tarea del Classroom.**
- **Exportar el notebook a un archivo PDF y anexarlo en la tarea del Classroom como un archivo independiente. No lo incluya dentro del ZIP , porque la idea que lo pueda accesar directamente para poner anotaciones y la calificación de cada ejercicio.**

En la descripción de los ejercicios se nombran algunas variables para el algoritmo, pero sólo es para facilitar la descripción. En la implementación pueden nombrar sus variables como gusten.

En los algoritmos se describen las entradas de las funciones. La intención es que tomen en cuenta lo que requiere el algoritmo y que tiene que haber parámetros que permitan controlar el comportamiento del algoritmo, evitando que dejen fijo un valor y que no se puede modificar para hacer diferentes pruebas. Si quieren dar esta información usando un tipo de dato que contenga todos los valores o usar variables por separado, etc., lo pueden hacer y no usen variables globales si no es necesario.

Lo mismo para los valores que devuelve una función. Pueden codificar como gusten la manera en que regresa los cálculos. El punto es que podamos tener acceso a los resultados, sin usar variables globales, y que la función no sólo imprima los valores que después no los podamos usar.

Ejercicio 1 (4 puntos)

La función de Rosenbrock se define como

$$\begin{aligned} & f(x_1, x_2) \\ & = 100\left(x_2 - x_1^2\right)^2 \\ & + (1 - x_1)^2. \end{aligned}$$

1. **Calcule las expresiones del gradiente y la Hessiana de la función de Rosenbrock.**
2. **Escriba las funciones en Python que evaluan la función de Rosenbrock, su gradiente y Hessiana.**
3. **Muestre que $x_* = (1, 1)^\top$ es el único punto estacionario de la función.**
4. **Calcule los eigenvalores de la matriz Hessiana de f en el punto x_* para mostrar que es definida positiva, por lo que x_* corresponde a un mínimo.**
5. **Grafique la función de Rosenbrock en el rectángulo $[-1.5, 1.5]$. Use las funciones `surface()` e `imshow()`**

$$\times [-1, 2]$$

para generar la gráfica 3D y la vista 2D.

Solución:

Respuesta 1.1.

En esta celda puede escribir las expresiones del Punto 1 o agregar una imagen que tenga la respuesta

La función de Rosenbroock es

$$f(x_1, x_2) = 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2.$$

El gradiente de la función es

$$\nabla f(x_1, x_2) = \begin{bmatrix} \frac{\partial}{\partial x_1}\left(\\ -400x\left(x_2 - x_1^2\right) \\ -2\left(1 - x_1\right)\right) \\ \frac{\partial}{\partial x_2} \\ \left(100\left(x_1 - x_1^2\right)^2 \\ +\left(1 - x_1\right)^2\right) \end{bmatrix} = \begin{bmatrix} -400x_1\left(x_2 - x_1^2\right) \\ -2\left(1 - x_1\right) \\ 200\left(x_2 - x_1^2\right) \end{bmatrix}$$

La matriz hessiana es

$$H_f = \begin{bmatrix} -400(x_2 - x_1^2) & -400x \\ +800x_1^2 + 2 & \\ -400x_1 & 200 \end{bmatrix}$$

In [3]:

```
# Respuesta 1.2. En esta celda puede poner el código de las funciones

import numpy as np

def function(x):
    xT = x.T
    return 100 * (xT[1] - xT[0]**2)**2 + (1-xT[0])**2

def gradient(x):
    xT = x.T
    return np.array([-400*xT[0]*(xT[1]- xT[0]**2) -2*(1-xT[0]), 200*(xT[1]-xT[0]**2)])

def hessian(x):
    xT = x.T
```

```
    return np.array([[-400*(xT[1]-xT[0]**2) + 800*xT[0]**2 + 2, -400*xT[0]],[-400*xT[0],
200]])
```

1.3

Primeramente notemos que es un punto estacionario:

Por definición un punto estacionario x es aquel que cumple que $\nabla f(x) = 0$. Veamos cual es aquel que cumple esta condición

$$\nabla f(x_1, x_2) = \begin{bmatrix} -400x_1\left(x_2 - x_1^2\right) \\ -2\left(1 - x_1\right) \\ 200\left(x_2 - x_1^2\right) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Esto implica que

$$200(x_2 - x_1^2) = 0 \implies x_2 = x_1^2$$

Entonces al sustituir en la primer igualdad

$$-400x\left(x_1^2 - x_1^2\right) - 2\left(1 - x_1\right) = 0 \implies -2(1 - x_1) = 0 \implies x_1 = 1, x_2 = (1)^2 = 1$$

De modo que el único punto que cumple con esa condición es

$$x^* = (1,1)^\top$$

En efecto veamos que

$$\nabla f(1,1) = \begin{bmatrix} -400(1)\left(1 - 1^2\right) \\ -2\left(1 - 1\right) \\ 200\left(1 - 1^2\right) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Por lo tanto $x* = (1,1)^\top$ es un punto estacionario y éste es el único.

In [4]:

```
from numpy.linalg import eigvals
# Respuesta 1.4.\
```

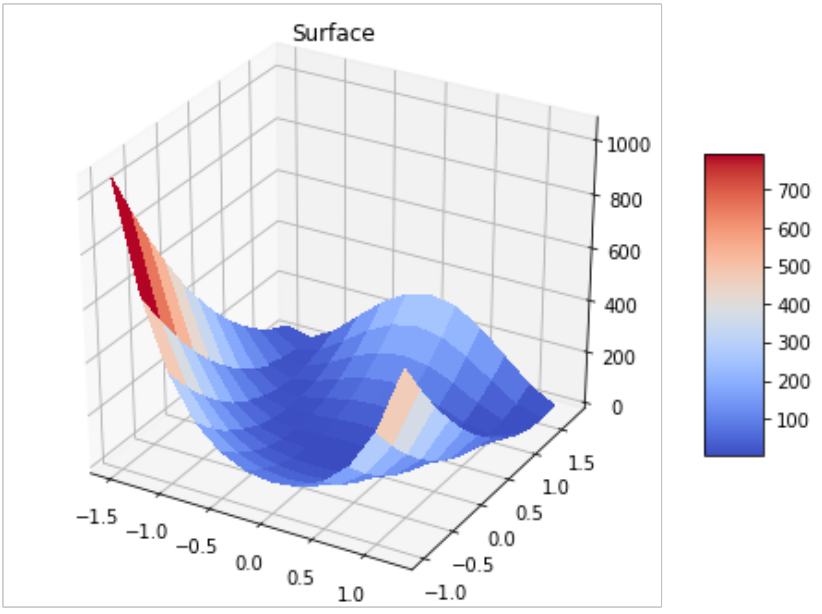
```
a = np.array([1,1])
eigval = np.linalg.eigvals (hessian(a))

print (eigval)
```

```
[1.00160064e+03 3.99360767e-01]
```

Dado que los eigenvalores son todas positivas en x^*, entonces la Hessiana es definida positiva y por lo tanto x_* corresponde a un mínimo.

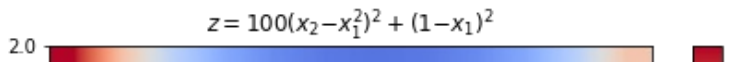
In [5]:

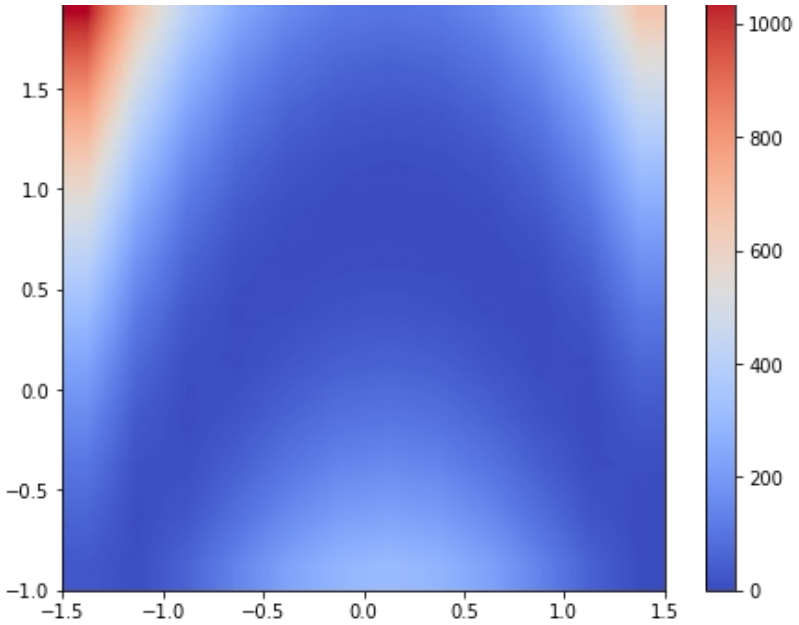
```
# Respuesta 1.5.
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from matplotlib import cm

#3D graph
fig = plt.figure(figsize=(8,6))
ax = plt.axes(projection='3d')
ax.set_title('Surface')
# Make data.
X = np.arange(-1.5, 1.5, 0.25)
Y = np.arange(-1, 2, 0.25)
X, Y = np.meshgrid(X, Y)
Vec = np.array([X,Y]).T
Z = function(Vec)

# Plot the surface.
surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm, linewidth=0, antialiased=False)

# Add a color bar which maps values to colors.
fig.colorbar(surf, shrink=0.5, aspect=5)

plt.show()
```



In [6]:

```
# 2D graph

fig = plt.figure(figsize=(8,6))
im = plt.imshow(Z, cmap='coolwarm', extent=(-1.5, 1.5, -1, 2),
                interpolation='bilinear')
plt.colorbar(im)
plt.title('$z=100(x_2 - x_1^2 )^2 + (1 - x_1)^2$')
plt.show()
```





Notemos que hay toda una franja donde se puede ver que la función se aproxima al mínimo. Vimos también que el valor mínimo de f está en $f(1,1)$.

$$= 0$$

Ejercicio 2 (3 puntos)

Programe la función que devuelve una aproximación del gradiente de una función en un punto particular usando diferencias finitas.

1. **La función que calcula la aproximación debe recibir como parámetros una función escalar f, el punto x y el incremento $h > 0$.**

- **Si n es el tamaño del arreglo x, cree un arreglo de tamaño n para almacenar las componentes de las aproximaciones del vector gradiente. Para aproximar la i-ésima derivada parcial use**

$$\frac{\partial f}{\partial x_i}(x) \approx \frac{f(x + he_i) - f(x)}{h},$$

donde e_i es el i-ésimo vector canónico.

1. **Pruebe la función comparando el gradiente analítico de la función de Rosenbrock en varios puntos y varios valores del parámetro h:**

- **Seleccione $h \in \{0.001, 0.0001, 0.00001\}$.**
- **Tome $x = (-1.5, 2)$ con $\lambda \in \{0, 0.5, +\lambda(2.5, -1) \quad 1.0\}$. Imprima el valor h, el punto x, el gradiente $g_a(x)$ obtenido con la función analítica programada en el Ejercicio 1, el gradiente $g_{df}(x;h)$ obtenido por diferencias finitas y la norma del vector $\|g_a(x) - g_{df}(x;h)\|$ (puede elegir la norma que quiera usar).**

Solución:

In [7]:

```
# Respuesta 2.1.
#Calculo de gradiente por diferencias finitas
def GradDifFin(f, x, h):
```

```
    n = np.size(x)
    Mx = np.tile(x , (n,1) ) #Repite el vector x, n veces.
    eval = Mx + h*np.identity(n)
    partial = (1/h)*(f(eval) - f(x))
    return partial
```

In [8]:

```
# Respuesta 2.2.


for h in [0.001, 0.0001, 0.00001]:
    for lamb in [0, 0.5, 1.0]:
        x = np.array([-1.5,2]) + lamb * np.array([2.5, -1])
        gdf = GradDifFin(function, x, h)
        print ('Valor h:', h)
        print ('Punto x:', x)
        print ('ga(x)=', gradient(x))
        print ('gdf(x; h)=', gdf)
        print ('Norma=', np.linalg.norm(gradient(x)- gdf))
        print ('\n')
```

```
Valor h: 0.001
Punto x: [-1.5  2. ]
ga(x)= [-155.  -50.]
gdf(x; h)= [-154.0495999  -49.9      ]
Norma= 0.9556465612998835


Valor h: 0.001
Punto x: [-0.25  1.5 ]
ga(x)= [141.25 287.5 ]
gdf(x; h)= [140.9884001 287.6      ]
Norma= 0.2800616140542944


Valor h: 0.001
Punto x: [1. 1.]
ga(x)= [-0.  0.]
gdf(x; h)= [0.4014001 0.1      ]
Norma= 0.4136689984515496


Valor h: 0.0001
Punto x: [-1.5  2. ]
ga(x)= [-155.  -50.]
gdf(x; h)= [-154.904906  -49.99    ]
Norma= 0.09561835007987253


Valor h: 0.0001
Punto x: [-0.25  1.5 ]
ga(x)= [141.25 287.5 ]
gdf(x; h)= [141.223849 287.51    ]
Norma= 0.027997764395229302


Valor h: 0.0001
Punto x: [1. 1.]
ga(x)= [-0.  0.]
gdf(x; h)= [0.040104 0.01    ]
Norma= 0.04133195886983943


Valor h: 1e-05
Punto x: [-1.5  2. ]
ga(x)= [-155.  -50.]
gdf(x; h)= [-154.99049006  -49.999     ]
Norma= 0.009562370974456795


Valor h: 1e-05
```

```
Punto x: [-0.25  1.5 ]
ga(x)= [141.25 287.5 ]
gdf(x; h)= [141.24738499 287.501     ]
Norma= 0.002799694922227753


Valor h: 1e-05
Punto x: [1. 1.]
ga(x)= [-0.  0.]
gdf(x; h)= [0.00401004 0.001     ]
Norma= 0.004132846573836988
```

Notemos que los valores de los gradientes aproximados y analíticos, son muy similares, por loq eu podemos afirmar que el método de aproximación es bueno. Sin embargo, al obtener las normas vemos que los valores podrían considerarse relativamente grandes. De modo que esto puede indicar que existan métodos aún más precisos de aproximación.

Ejercicio 3 (3 puntos)

Programe la función que devuelve una aproximación de la Hessiana de una función en un punto particular usando diferencias finitas.

1. **La función que calcula la aproximación debe recibir como parámetros una función escalar f, el punto x y el incremento $h > 0$.**

- **Si n es el tamaño del arreglo x, cree una matriz de tamaño $n \times n$ para almacenar las entradas de las aproximaciones de las segundas derivadas parciales. Puede usar**

$$\begin{aligned}\frac{\partial^2 f}{\partial x_i \partial x_j}(x) \\ f(x + he_i + he_j) \\ - f(x + he_i) \\ - f(x + he_j) \\ + f(x) \\ \approx \frac{}{h^2},\end{aligned}$$

donde e_i es el i-ésimo vector canónico.

1. **Pruebe la función comparando el gradiente analítico de la función de Rosenbrock en varios puntos y varios valores del parámetro h:**

- **Seleccione $h \in \{0.001,$**
  $0.0001,$
  $0.00001\}$
- **Tome $x = (-1.5, 2)$ con $\lambda \in \{0, 0.5,$ Imprima el valor h, el punto x, la Hessiana $H_a(x)$ obtenido con la**
  $+\lambda(2.5, -1)$ $1.0\}$
  función analítica programada en el Ejercicio 1, la Hessiana $H_{df}(x; h)$ obtenido por diferencias finitas y la norma de la matriz $\|H_a(x)$ (puede elegir la norma que quiera usar).
  $- H_{df}(x; h)\|$

In [9]:

```
# Respuesta 3.1.

#Calculo de la matriz Hessiana por diferencias finitas
def HessDifFin(f, x, h):
    n = np.size(x)
    H = np.zeros((n,n))
    for i in range(n):
```

```
        ei = np.eye(1, n, i)
        for j in range(n):
            ej = np.eye(1, n, j)
            H[i,j] = (1/(h**2))*(f(x + h*ei + h*ej) - f(x + h*ei) - f(x + h*ej) + f(x))
    return H
```

In [10]:

```
# Respuesta 3.2.

for h in [0.001, 0.0001, 0.00001]:
    for lamb in [0, 0.5, 1.0]:
        x = np.array([-1.5,2]) + lamb * np.array([2.5, -1])
        Hdf = HessDifFin(function, x, h)
        print ('Valor h:', h)
        print ('Punto x:', x)
        print ('Ha(x)=', hessian(x))
        print ('Hdf(x; h)=', Hdf)
        print ('Norma=', np.linalg.norm(hessian(x)- Hdf))
        print ('\n')
```

```
Valor h: 0.001
Punto x: [-1.5  2. ]
Ha(x)= [[1902.  600.]
 [ 600.  200.]]
Hdf(x; h)= [[1898.40139999  599.8       ]
 [ 599.8         200.        ]]
Norma= 3.6096983250551644


Valor h: 0.001
Punto x: [-0.25  1.5 ]
Ha(x)= [[-523.  100.]
 [ 100.  200.]]
Hdf(x; h)= [[-523.59860001   99.79999999]
 [  99.79999999  199.99999998]]
Norma= 0.662058898046224


Valor h: 0.001
Punto x: [1. 1.]
Ha(x)= [[ 802. -400.]
 [-400.  200.]]
Hdf(x; h)= [[ 804.4014 -400.2   ]
 [-400.2     200.    ]]
Norma= 2.4179995779507264


Valor h: 0.0001
Punto x: [-1.5  2. ]
Ha(x)= [[1902.  600.]
 [ 600.  200.]]
Hdf(x; h)= [[1901.64001346  599.98000008]
 [ 599.98000008  199.99999985]]
Norma= 0.36109597644905883


Valor h: 0.0001
Punto x: [-0.25  1.5 ]
Ha(x)= [[-523.  100.]
 [ 100.  200.]]
Hdf(x; h)= [[-523.05998395   99.97999655]
 [  99.97999655  199.99999097]]
Norma= 0.06632005904773697


Valor h: 0.0001
Punto x: [1. 1.]
Ha(x)= [[ 802. -400.]
 [-400.  200.]]
Hdf(x; h)= [[ 802.240014 -400.02    ]
 [-400.02      200.      ]]
```

```
Norma= 0.24167482336199914


Valor h: 1e-05
Punto x: [-1.5  2. ]
Ha(x)= [[1902.  600.]
 [ 600.  200.]]
Hdf(x; h)= [[1901.96402627  599.99800683]
 [ 599.99800683  199.99999878]]
Norma= 0.03608399064933921


Valor h: 1e-05
Punto x: [-0.25  1.5 ]
Ha(x)= [[-523.  100.]
 [ 100.  200.]]
Hdf(x; h)= [[-523.00549669   99.99837403]
 [  99.99837403  200.00015866]]
Norma= 0.005960397911606387


Valor h: 1e-05
Punto x: [1. 1.]
Ha(x)= [[ 802. -400.]
 [-400.  200.]]
Hdf(x; h)= [[ 802.02400015 -400.00200001]
 [-400.00200001  200.        ]]
Norma= 0.024166244287936563
```

Aquí podemos ver lo mismo que ocurrió con la aproximación del gradiente. Vemos que es una buena aproximación, sin embargo al tomar la norma de la diferencia no es tan pequeña.