Tarea 6

March 18, 2022

1 Curso de Optimización (DEMAT)

1.1 Tarea 6

Descripción:	Fechas
Fecha de publicación del documento:	**Marzo 11, 2022**
Fecha límite de entrega de la tarea:	**Marzo 20, 2022**

1.1.1 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.

1.2 Ejercicio 1 (5 puntos)

Programar el método de Newton con tamaño de paso fijo $\alpha = 1$.

La función recibe como parámetros la función que calcula el gradiente $g(x)$ de la función objetivo $f : \mathbb{R}^n \to \mathbb{R}$, la función que calcula la Hessiana $H(x)$ de f, un punto inicial x_0, un número máximo de iteraciones N, y la tolerancia $\tau > 0$. Fijar $k = 0$ y repetir los siguientes pasos:

1. Calcular el gradiente g_k en el punto x_k, $g_k = g(x_k)$.
2. Si $\|g_k\| < \tau$, hacer $res = 1$ y terminar.
3. Si no se cumple el criterio, calcular la Hessiana $H_k = H(x_k)$.
4. Intentar calcular la factorización de Cholesky de H_k.
5. Si la factorización no se puede realizar, imprimir el mensaje de error, hacer $res = 0$ y terminar el ciclo.

6. Si se obtuvo la factorización, resolver el sistema de ecuaciones $H_k p_k = -g_k$ (esto da la dirección de descenso como p_k).
7. Calcular el siguiente punto de la secuencia como

$$x_{k+1} = x_k + p_k$$

8. Si $k+1 \geq N$, hacer $res = 0$ y terminar.
9. Si no, hacer $k = k+1$ y volver el paso 1.
10. Devolver el punto x_k, g_k, k y res.

Nota: Para calcular la factorización de Cholesky y resolver el sistema de ecuaciones puede usar las funciones `scipy.linalg.cho_factor` y `scipy.linalg.cho_solve`. Si la matriz no es definida positiva, la función `cho_factor` lanza la excepción `scipy.linalg.LinAlgError`. Puede usar esto para terminar el ciclo.

---

1. Programe la función que implementa el algoritmo del método de Newton, almacenando en una lista los puntos $x_0, x_1, ..., x_k$ que genera el algoritmo. Haga que la función devuelva esta lista.
2. Use la función de Rosenbrock, su gradiente y Hessiana para probar el algoritmo.

- Use $N = 1000$, la tolerancia $\tau = \sqrt{\epsilon_m}$, donde ϵ_m es el épsilon de la máquina, y el punto inicial $x_0 = (-1.2, 1)$.
- Si el algoritmo converge, imprima un mensaje que indique esto y genere una gráfica que muestre las curvas de nivel de la función f y la trayectoria de los puntos $x_0, x_1, ..., x_k$. Para generar esta gráfica use una discretización de los intervalos $[-1.5, 1.5]$ en la dirección X y $[-1, 2]$ en la dirección Y.
- Imprima el punto final x_k, $f(x_k)$, la magnitud del gradiente g_k y el número de iteraciones k realizadas.
- Repita la prueba partiendo del punto inicial $x_0 = (-12, 10)$.

1.2.1 Solución:

[ ]:
```
# En esta celda puede poner el código de las funciones
# o poner la instrucción para importarlas de un archivo .py
import numpy as np
from scipy.linalg import cho_solve, cho_factor
import itertools
import plotly.graph_objects as go
from IPython.core.display import Image, display

def newthon_method(gf, Hf, x_0, N, T):

    x_k = x_0
    k = 0
    res = 0
    points = [x_0]
    while (k < N):
```

```python
        gf_k = gf(x_k)
        if np.linalg.norm(gf_k) < T:
            res = 1
            break
        Hf_k = Hf(x_k)
        try:
            p_k = cho_solve(cho_factor(Hf_k), -gf_k)
        except:
            break
        x_k = x_k + p_k
        points.append(x_k)
        k = k + 1

    points = np.array(points)
    return points, gf(x_k), k, res

def test(f, gf, Hf, x_0, N, T, xtl, xtr, ytl, ytr, p):
    points, g_k, k, res = newthon_method(gf, Hf, x_0, N, T)
    print(f"El algoritmo " + ("no " if res == 0 else "") + "convergió en:")
    x_k = points[-1]
    x_output = np.concatenate((x_k[0,:3], x_k[0,-min(3, len(x_k)-3):]if len(x_k) > 3 else []))
    print(f"k = {k}, ||g_k|| = {np.linalg.norm(g_k)}, x_k = {x_output}")
    fig = go.Figure()
    xtl = min(xtl, min(points[:,0,0]))
    xtr = max(xtr, max(points[:,0,0]))
    ytl = min(ytl, min(points[:,0,1]))
    ytr = max(ytr, max(points[:,0,1]))
    xtl, xtr = xtl - (xtr - xtl)/10, xtr + (xtr - xtl)/10
    ytl, ytr = ytl - (ytr - ytl)/10, ytr + (ytr - ytl)/10
    x = np.linspace(xtl,xtr,p)
    y = np.linspace(ytl,ytr,p)
    fig.add_contour(
        x=x,
        y=y,
        z=np.array([f(np.array([[x_i,y_i]])) for y_i, x_i in itertools.product(y,x)]).reshape(p,p),
        name='Rosenbrock'
    )
    fig.add_trace(
        go.Scatter(
            x = points[:,0,0],
            y = points[:,0,1],
            name='Newton'
        )
    )
    fig.update_layout(
```

```
        template="simple_white",
        title="Rosenbrock",
        width=500,
        height=500
    )
    #fig.show()
```

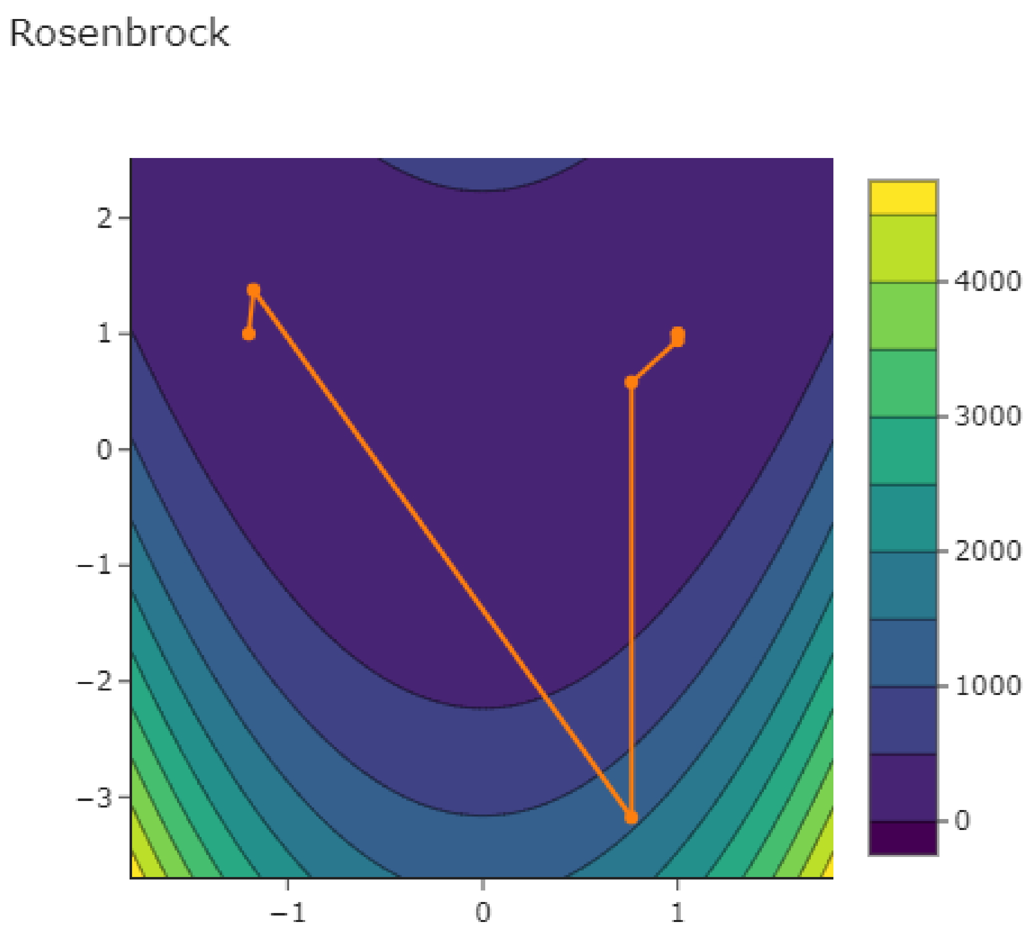
```
C:\Users\batma\AppData\Local\Temp\ipykernel_10500\2626011398.py:7:
DeprecationWarning:

Importing display from IPython.core.display is deprecated since IPython 7.14,
please import from IPython display
```

[ ]:
```
# Pruebas realizadas a la función de Rosenbrock
def f(x):
    return 100*(x[0,1] - x[0,0]**2)**2 + (1 - x[0,0])**2

def gf(x):
    return np.array(
        [
            400*(x[0,0]**3-x[0,0]*x[0,1])+2*(x[0,0]-1),
            200*(x[0,1]-x[0,0]**2)
        ]
    )

def hf(x):
    return np.array(
        [
            [
                1200*x[0,0]**2-400*x[0,1]+2,
                -400*x[0,0]
            ],
            [
                -400*x[0,0],
                200
            ]
        ]
    )
```

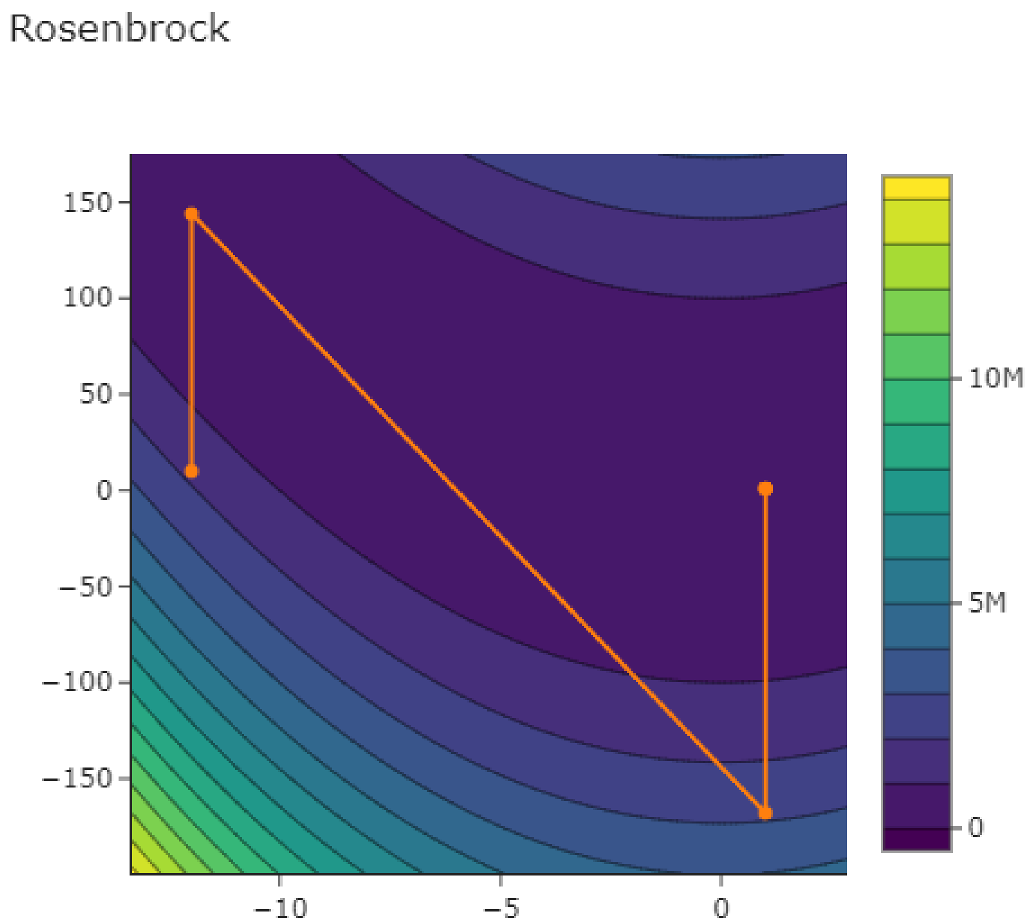
[ ]:
```
# Prueba realizada a la función del Ejercicio 1

x_0 = np.array([[-1.2,1]])
N = 1000
T = np.finfo(float).eps**(1/2)
xtl = -1.5
xtr = 1.5
```

```
ytl = -1
ytr = 2
p = 100
test(f, gf, hf, x_0, N, T, xtl, xtr, ytl, ytr, p)
display(Image("img/e_1_1.png"))
x_0 = np.array([[-12,10]])
test(f, gf, hf, x_0, N, T, xtl, xtr, ytl, ytr, p)
display(Image("img/e_1_2.png"))
```

```
El algoritmo convergió en:
k = 6, ||g_k|| = 8.285755243355701e-09, x_k = [1. 1.]
```



```
El algoritmo convergió en:
k = 5, ||g_k|| = 0.0, x_k = [1. 1.]
```



1.3 Ejercicio 2 (5 puntos)

Programar el método de Newton con tamaño de paso ajustado por el algoritmo de backtracking.

1. Modifique la función del Ejercicio 2 que implementa el algoritmo del método de Newton para incluir como parámetros a la función objetivo $f(x)$ y los parámetros ρ y c_1 del algoritmo de backtraking.

- Después de obtener la dirección de descenso p_k, calcular el tamaño de paso α_k usando como valor inicial $\bar{\alpha}_0 = 1$ en el algoritmo de backtracking.

- Hacer

$$x_{k+1} = x_k + \alpha_k p_k.$$

2. Repita la prueba del algoritmo, como se indicó en el Ejercicio 1, a la función de Rosenbrock, para ver como cambia la trayectoria de los puntos $x_0, x_1, ..., x_k$ en comparación con el resultado anterior, partiendo de $x_0 = (-1.2, 1)$ y de $x_0 = (-12, 10)$.

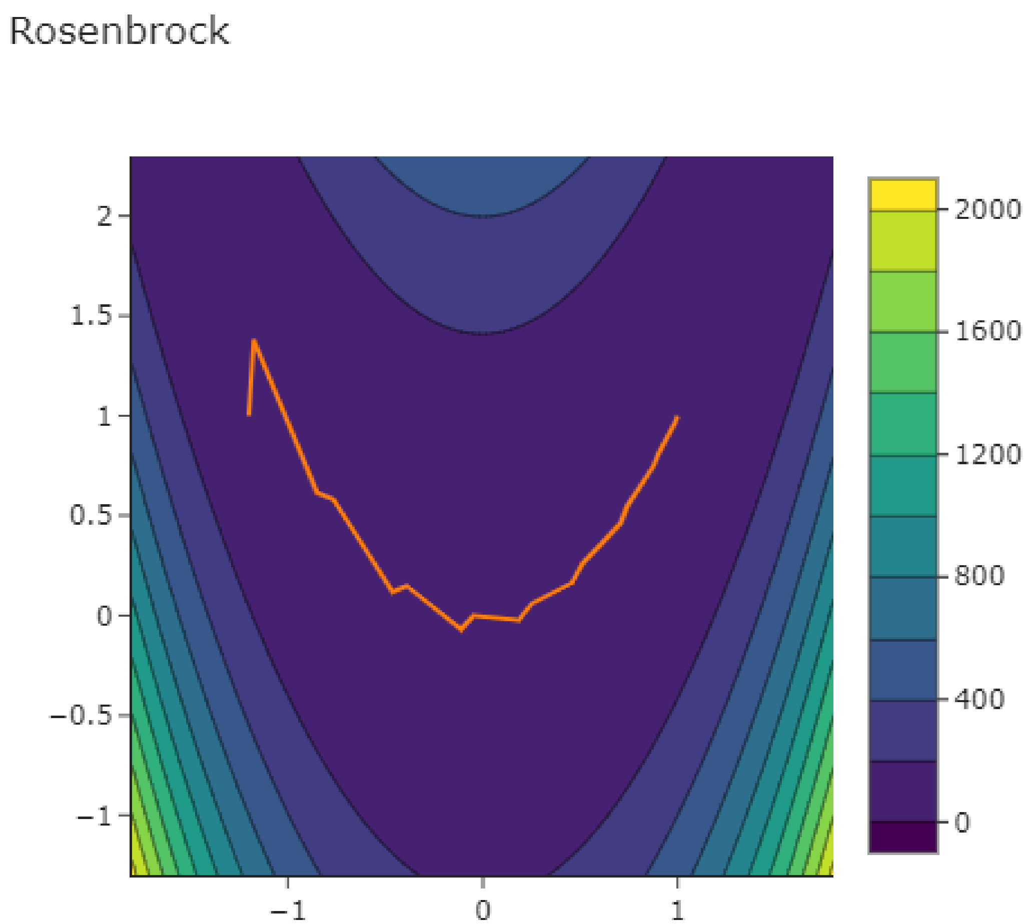
[ ]:
```python
# En esta celda puede poner el código de las funciones
# o poner la instrucción para importarlas de un archivo .py

def backtracking(f, f_k, gf_k, x_k, p_k, alpha, ro, c):
    while (f(x_k + alpha*p_k) > f_k + c*alpha*gf_k@p_k.T):
        alpha = ro*alpha
    return alpha

def newthon_method_backtracking(gf, Hf, x_0, alpha, ro, c, N, T):

    x_k = x_0
    k = 0
    res = 0
    points = [x_0]
    while (k < N):
        gf_k = gf(x_k)
        if np.linalg.norm(gf_k) < T:
            res = 1
            break
        Hf_k = Hf(x_k)
        try:
            p_k = cho_solve(cho_factor(Hf_k), -gf_k)
        except:
            break
        a_k = backtracking(f, f(x_k), gf_k, x_k, p_k, alpha, ro, c)
        x_k = x_k + a_k * p_k
        points.append(x_k)
        k = k + 1

    points = np.array(points)
    return points, gf(x_k), k, res

def test(f, gf, Hf, x_0, alpha, ro, c, N, T, xtl, xtr, ytl, ytr, p):
    points, g_k, k, res = newthon_method_backtracking(gf, Hf, x_0, alpha, ro,␣
↪c, N, T)
    print(f"El algoritmo " + ("no " if res == 0 else "") + "convergió en:")
    x_k = points[-1]
    x_output = np.concatenate((x_k[0,:3], x_k[0,-min(3, len(x_k)-3):]if␣
↪len(x_k) > 3 else []))
    print(f"k = {k}, ||g_k|| = {np.linalg.norm(g_k)}, x_k = {x_output}")
    fig = go.Figure()
```

```python
    xtl = min(xtl, min(points[:,0,0]))
    xtr = max(xtr, max(points[:,0,0]))
    ytl = min(ytl, min(points[:,0,1]))
    ytr = max(ytr, max(points[:,0,1]))
    xtl, xtr = xtl - (xtr - xtl)/10, xtr + (xtr - xtl)/10
    ytl, ytr = ytl - (ytr - ytl)/10, ytr + (ytr - ytl)/10
    x = np.linspace(xtl,xtr,p)
    y = np.linspace(ytl,ytr,p)
    fig.add_contour(
        x=x,
        y=y,
        z=np.array([f(np.array([[x_i,y_i]])) for y_i, x_i in itertools.
 ↪product(y,x)]).reshape(p,p),
        name='Rosenbrock'
    )
    fig.add_trace(
        go.Scatter(
            x = points[:,0,0],
            y = points[:,0,1],
            name='Newton'
        )
    )
    fig.update_layout(
        template="simple_white",
        title="Rosenbrock",
        width=500,
        height=500
    )
    #fig.show()
```

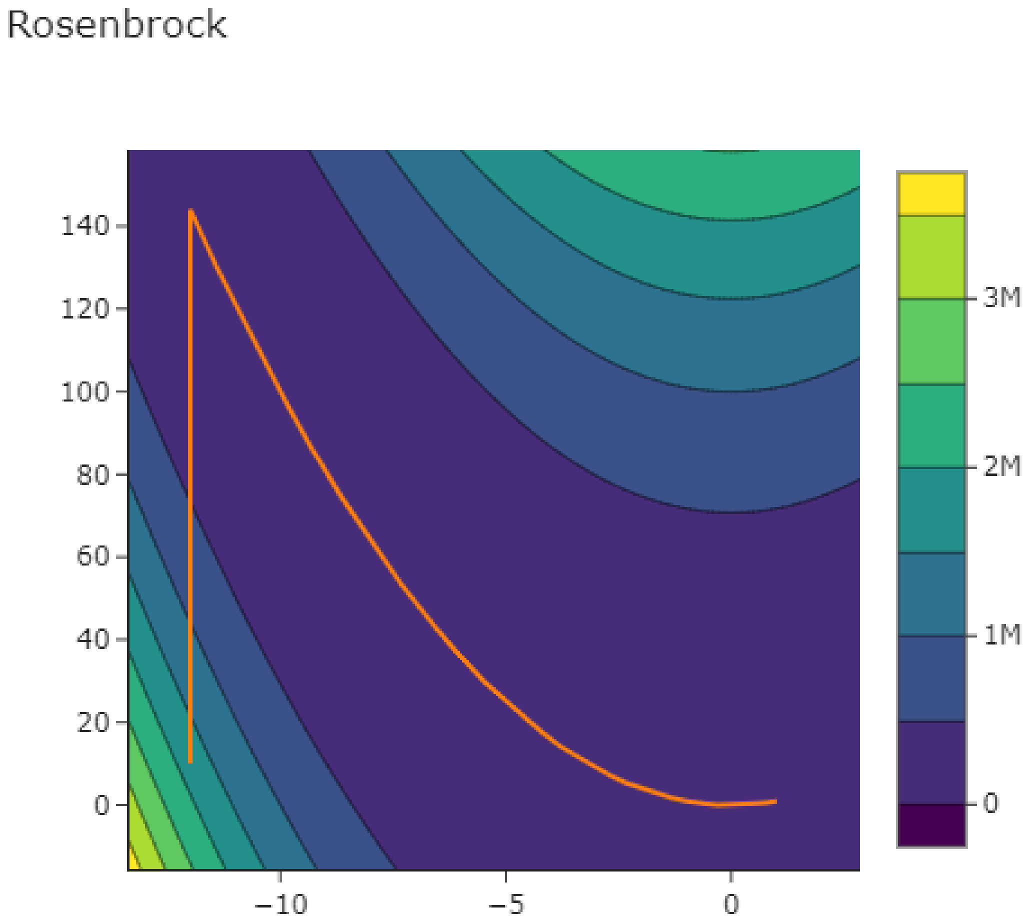
[ ]:
```python
# Prueba realizada a la función del Ejercicio 1

x_0 = np.array([[-1.2,1]])
N = 1000
T = np.finfo(float).eps**(1/2)
xtl = -1.5
xtr = 1.5
ytl = -1
ytr = 2
p = 100
test(f, gf, hf, x_0, 1, 0.8, 0.0001, N, T, xtl, xtr, ytl, ytr, p)
display(Image("img/e_2_1.png"))
x_0 = np.array([[-12,10]])
test(f, gf, hf, x_0, 1, 0.8, 0.0001, N, T, xtl, xtr, ytl, ytr, p)
display(Image("img/e_2_2.png"))
```

El algoritmo convergió en:

```
k = 20, ||g_k|| = 3.21430828407909e-11, x_k = [1. 1.]
```




```
El algoritmo convergió en:
k = 57, ||g_k|| = 3.4893405656074994e-10, x_k = [1. 1.]
```


Rosenbrock
140
120
100
80
60
40
20
0
−10
−5
0
3M
2M
1M
0