Tarea 04

March 8, 2022

1 Curso de Optimización (DEMAT)

1.1 Tarea 4

Descripción:	Fechas
Fecha de publicación del documento:	**Febrero 24, 2022**
Fecha límite de entrega de la tarea:	**Marzo 6, 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.

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.

1.2 Ejercicio 1 (5 puntos)

Programar el método de descenso máximo con tamaño de paso exacto para minimizar funciones cuadráticas:

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

donde $\mathbf{A} \in \mathbb{R}^{n\times n}$ y $x \in \mathbb{R}^n$.

Dado el vector b, la matriz $\mathbf{A}$, un punto inicial x_0, un número máximo de iteraciones N, la tolerancia $\tau > 0$. Fijar $k = 0$ y repetir los siguientes pasos:

1. Calcular el gradiente en el punto x_k,

$$g_k = \nabla f(x_k) = \mathbf{A}x_k - b.$$

2. Si $\|g_k\| < \tau$, entonces x_k es (casi) un punto estacionario. Hacer $res = 1$ y terminar el ciclo.
3. Elegir la dirección de descenso como $p_k = -g_k$.
4. Calcular el tamaño de paso α_k que minimiza el valor de la función

$$\phi_k(\alpha) = f(x_k + \alpha p_k)$$

es decir, calcular

$$\alpha_k = -\frac{g_k^\top p_k}{p_k^\top \mathbf{A} p_k}$$

5. Calcular el siguiente punto de la secuencia como

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

6. Si $k + 1 \geq N$, hacer $res = 0$ y terminar.
7. Si no, hacer $k = k + 1$ y volver el paso 1.
8. Devolver el punto x_k, $f_k = \frac{1}{2}x_k^\top \mathbf{A}x_k - b^\top x_k$, g_k, k y res.

---

1. Escriba una función que implementa el algoritmo anterior usando arreglos de Numpy.

2. Escriba una función para probar el funcionamiento del método de descenso máximo. Esta función debe recibir como parámetros el nombre de un archivo `.npy` que contiene las entradas de una matriz cuadrada $\mathbf{A}$, el vector b, un punto inicial x_0, el número máximo de iteraciones N y la tolerancia τ.

- Esta función debe crear la matriz $\mathbf{A}$ y el vector b leyendo los archivos de datos.
- Obtener el número de filas r de la matriz e imprimir este valor.
- Compruebe que la matriz es simétrica y definida positiva calculando e imprimiendo el valor $\|\mathbf{A} - \mathbf{A}^\top\|$ y su eigenvalor más pequeño (use la función `numpy.linalg.eig()`).
- Ejecutar la función del Inciso 1.

- Dependiendo del valor de la variable res, imprima un mensaje que diga que el algoritmo convergió ($res = 1$) o no ($res = 0$).
- Imprimir k, f_k, la norma de g_k, y los primeros 3 y últimos 3 elementos del arreglo x_k.
- Calcule directamente el minimizador resolviendo la ecuación $Ax_* = b$ e imprima el valor del error $\|x_k - x_*\|$.

3. Pruebe la función del Inciso 2 usando $N = 1000$, la tolerancia $\tau = \epsilon_m^{1/3}$, donde ϵ_m es el épsilon de la máquina, y los arreglos que se incluyen en el archivo datosTarea04.zip, de la siguiente manera:

Matriz	Vector	Punto para iniciar la secuencia
A1.npy	b1.npy	$x_0 = (0, -5)$
A1.npy	b1.npy	$x_0 = (7045, 7095)$
A2.npy	b2.npy	$x_0 = (0, 0, ..., 0) \in \mathbb{R}^{500}$
A2.npy	b2.npy	$x_0 = (10000, 10000, ..., 10000) \in \mathbb{R}^{500}$

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

def f_square(x, A, b):
    return ((x@A.T)@x.T)/2 - b@x.T

def gf_square(x, A, b):
    return x@A.T - b

def gradient_descent_square_function(f, gf, A, b, x_0, N, T):

    x_k = x_0
    k = 0
    res = 0
    while (k < N):
        gf_k = gf(x_k, A, b)
        if np.linalg.norm(gf_k) < T:
            res = 1
            break
        p_k = -gf_k
        a_k = -(gf_k@p_k.T)/(p_k@A@p_k.T)
        x_k = x_k + a_k*p_k
        k = k + 1

    return x_k, f(x_k, A, b), gf(x_k, A, b), k, res

def files_data(file_name, x_0, N, T):
```

```
    A = np.load("A"+file_name)
    print(A.shape)
    print(np.linalg.norm(A-A.T),np.min(np.linalg.eigvals(A)))
    b = np.load("b"+file_name)
    b = np.array([b])
    print(b.shape)
    x_k, f_k, g_k, k, res = gradient_descent_square_function(f_square,␣
 ↪gf_square, A, b, x_0, N, T)
    print(f"El algoritmo " + ("no" if res == 0 else "") + " convergió en:")
    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}, f_k = {f_k}, ||g_k|| = {np.linalg.norm(g_k)}, x_k =␣
 ↪{x_output}")
    print(np.linalg.norm(x_k-np.linalg.solve(A, b.squeeze())))
```

[ ]:
```
# Lectura de datos y pruebas realizadas

x_0 = np.array([[[0,-5]],[[7045,7095]]])
for x_0_i in x_0:
    files_data("1.npy", x_0_i, 1000, np.finfo(float).eps**(1/3))

N = 500
x_0 = np.array([[np.zeros(N)],[np.repeat(10000,N)]])
for x_0_i in x_0:
    files_data("2.npy", x_0_i, 1000, np.finfo(float).eps**(1/3))
```

```
(2, 2)
0.0 0.10000000000000003
(1, 2)
El algoritmo  convergió en:
k = 69, f_k = [[-62.75]], ||g_k|| = 4.767340293857206e-06, x_k = [-24.49997287
25.49997743]
3.5289985397025655e-05
(2, 2)
0.0 0.10000000000000003
(1, 2)
El algoritmo  convergió en:
k = 1, f_k = [[-62.75]], ||g_k|| = 6.421214021448452e-13, x_k = [-24.5  25.5]
9.131096203816022e-13
(500, 500)
1.5063918215255853e-14 0.09999999999999293
(1, 500)
El algoritmo  convergió en:
k = 332, f_k = [[-5239.54141208]], ||g_k|| = 5.996601774308784e-06, x_k =
[-11.61784712   5.44982336   0.96506345]
3.9297511566363496e-05
```

```
(500, 500)
1.5063918215255853e-14 0.09999999999999293
(1, 500)
El algoritmo  convergió en:
k = 453, f_k = [[-5239.54141208]], ||g_k|| = 5.7003604586793e-06, x_k =
[-11.61784726   5.4498234    0.96506346]
3.859553623165663e-05
```

1.3 Ejercicio 2 (5 puntos)

Programar el método de descenso máximo con tamaño de paso seleccionado por la estrategia de backtracking:

Algoritmo de descenso máximo con backtracking:

Dada una función $f : \mathbb{R}^n \to \mathbb{R}$, su gradiente $g : \mathbb{R}^n \to \mathbb{R}^n$, un punto inicial x_0, un número máximo de iteraciones N, una tolerancia $\tau > 0$. Fijar $k = 0$ y repetir los siguientes pasos:

1. Calcular el gradiente en el punto x_k:

$$g_k = \nabla f(x_k) = g(x_k)$$

2. Si $\|g_k\| < \tau$, x_k es un aproximadamente un punto estacionario, por lo que hay que hacer $res = 1$ y terminar el ciclo.
3. Eligir la dirección de descenso como $p_k = -g_k$.
4. Calcular el tamaño de paso α_k mediante la estrategia de backtraking, usando el algoritmo que describe más adelante.
5. Calcular el siguiente punto de la secuencia como

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

6. Si $k + 1 > N$, hacer $res = 0$ y terminar.
7. Si no, hacer $k = k + 1$ y volver el paso 1.
8. Devolver el punto x_k, $f_k = f(x_k)$, g_k, k y res.

** Algoritmo de backtracking **

Backtracking(f, f_k, g_k, x_k, p_k, α_{ini}, ρ, c)

El algoritmo recibe la función f, el punto x_k, $f_k = f(x_k)$, la dirección de descenso p_k, un valor inicial α_{ini}, $\rho \in (0, 1)$, $c \in (0, 1)$.

Fijar $\alpha = \alpha_{ini}$ y repetir los siguientes pasos:

1. Si se cumple la condición

$$f(x_k + \alpha p_k) \leq f_k + c\alpha g_k^\top p_k,$$

terminar el ciclo devolviendo

2. Hacer $\alpha = \rho\alpha$ y regresar al paso anterior.

---

1. Escriba una función que implementa el algoritmo de backtracking.
2. Escriba la función que implementa el algoritmo de máximo descenso con búsqueda inexacta, usando backtraking. Tiene que recibir como paramétros todos los elementos que se listaron para ambos algoritmos.
3. Escriba una función para probar el funcionamiento del método de descenso máximo. Esta función debe recibir la función f, la función g que devuelve su gradiente, el punto inicial x_0, el número
máximo de iteraciones N, la tolerancia $\tau > 0$ y el factor ρ del algoritmo de backtracking.

- Fijar los parámetros $\alpha_{ini} = 2$ y $c = 0.0001$ del algoritmo de backtracking.
- Ejecutar la función del Inciso 2.
- Dependiendo del valor de la variable res, imprima un mensaje que diga que el algoritmo convergió ($res = 1$) o no ($res = 0$)
- Imprimir k, x_k, f_k y la norma de g_k.

4. Pruebe la función del Inciso 3 usando $N = 10000$, $\rho = 0.8$, la tolerancia $\tau = \epsilon_m^{1/3}$, donde ϵ_m es el épsilon de la máquina. Aplique esta función a:

- La función de Rosenbrock, descrita en la Tarea 3, usando como punto inicial $x_0 = (-1.2, 1)$ y $x_0 = (-12, 10)$. Como referencia, el minimizador de la función es $x_* = (1, 1)$.

5. Repita el inciso anterior con $\rho = 0.5$.

1.3.1 Solución:

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

def f_rosenbrock(x):
    return 100*(x[0,1] - x[0,0]**2)**2 + (1 - x[0,0])**2

def gf_rosenbrock(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 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 gradient_descent_backtracking(f, gf, x_0, alpha, ro, c, N, T):

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

```
        gf_k = gf(x_k)
        if np.linalg.norm(gf_k) < T:
            res = 1
            break
        p_k = -gf_k
        a_k = backtracking(f, f(x_k), gf_k, x_k, p_k, alpha, ro, c)
        x_k = x_k + a_k*p_k
        k = k + 1

    return x_k, f(x_k), gf(x_k), k, res

def exercise(f, gf, x_0, N, T, ro):

    x_k, f_k, g_k, k, res = gradient_descent_backtracking(f, gf, x_0, 2, ro, 0.
 ↪0001, N, T)
    print(f"El algoritmo " + ("no" if res == 0 else "") + " convergió en:")
    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}, f_k = {f_k}, ||g_k|| = {np.linalg.norm(g_k)}, x_k =␣
 ↪{x_output}")
```

[ ]:
```
# Lectura de datos y pruebas realizadas

x_0 = np.array([[[-1.2,1]],[[-12,10]]])
for x_0_i in x_0:
    exercise(f_rosenbrock, gf_rosenbrock, x_0_i, 10000, np.finfo(float).eps**(1/
 ↪3), 0.8)
```

```
El algoritmo no convergió en:
k = 10000, f_k = 6.274640640397673e-07, ||g_k|| = 0.001941845100290949, x_k =
[1.00079208 1.00158391]
El algoritmo no convergió en:
k = 10000, f_k = 8.531110165045911, ||g_k|| = 3.1564912743986095, x_k = [
3.92075554 15.37404809]
```