Tarea 5

March 13, 2022

1 Curso de Optimización (DEMAT)

1.1 Tarea 5

Descripción:	Fechas
Fecha de publicación del documento:	**Marzo 6, 2022**
Fecha límite de entrega de la tarea:	**Marzo 13, 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 (6 puntos)

Programar el método de descenso máximo con tamaño de paso fijo y probarlo.

El algoritmo recibe como parámetros la función gradiente $g(x)$ de la función objetivo, un punto inicial x_0, el valor del tamaño de paso α, un número máximo de iteraciones N, 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. Elegir la dirección de descenso como $p_k = -g_k$.

4. Calcular el siguiente punto de la secuencia como

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

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

De acuerdo con la proposición vista en la clase 12, para que el método de máximo descenso con paso fijo para funciones cuadráticas converja se requiere que el tamaño de paso α cumpla con

$$0 < \alpha < \frac{2}{\lambda_{\max}(A)} = \alpha_{\max},$$

donde $\lambda_{\max}(A)$ es el eigenvalor más grande de A.

1. Escriba una función que implementa el algoritmo de descenso máximo con paso fijo.

2. Programe las funciones cuadráticas y sus gradientes

$$f_i(x) = \frac{1}{2} x^\top \mathbf{A}_i x - \mathbf{b}_i^\top x, \quad i = 1, 2$$

   donde

$$\mathbf{A}_1 = \begin{bmatrix} 1.18 & 0.69 \\ 0.69 & 3.01 \end{bmatrix}, \qquad \mathbf{b}_1 = \begin{pmatrix} -0.24 \\ 0.99 \end{pmatrix}.$$

$$\mathbf{A}_2 = \begin{bmatrix} 6.36 & -3.07 & -2.8 & -3.42 & -0.68 \\ -3.07 & 10.19 & 0.74 & 0.5 & 0.72 \\ -2.8 & 0.74 & 4.97 & -1.48 & 1.93 \\ -3.42 & 0.5 & -1.48 & 4.9 & -0.97 \\ -0.68 & 0.72 & 1.93 & -0.97 & 3.21 \end{bmatrix}, \qquad \mathbf{b}_2 = \begin{pmatrix} 0.66 \\ 0.37 \\ -2.06 \\ 0.14 \\ 1.36 \end{pmatrix}.$$

3. Fije el número máximo de iteraciones $N = 2000$ y la tolerancia $\tau = \sqrt{\epsilon_m}$, donde ϵ_m es el épsilon de la máquina. Para cada función cuadrática, calcule $\alpha_{\max}$ de la matriz $\mathbf{A}_i$. Pruebe con los tamaños de paso α igual a $1.1\alpha_{\max}$ y $0.9\alpha_{\max}$. Use el punto inicial

$$\mathbf{x}_0 = \begin{pmatrix} -38.12 \\ -55.87 \end{pmatrix} \quad \text{para} \quad f_1$$

$$\mathbf{x}_0 = \begin{pmatrix} 4.60 \\ 6.85 \\ 4.31 \\ 4.79 \\ 8.38 \end{pmatrix} \quad \text{para} \quad f_2$$

4. En cada caso imprima x_k, $\|g_k\|$, el número de iteraciones k y el valor de res.

1.2.1 Solución:

```
import numpy as np

def gradient_descent_fixed_step(gf, x_0, alpha, 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 = alpha
        x_k = x_k + a_k*p_k
        k = k + 1

    return x_k, gf(x_k), k, res

def test(g, x_0, alpha, N, T):
    x_k, g_k, k, res = gradient_descent_fixed_step(g, x_0, alpha, 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}, ||g_k|| = {np.linalg.norm(g_k)}, x_k = {x_output}")

def square_function_test(A, b, x_0, alpha, N, T):
    test((lambda x : x@A.T - b), x_0, alpha, N, T)
```

```
# Pruebas del algoritmo
import itertools

A = [
    np.array(
        [
            [1.18, 0.69],
            [0.69 , 3.01]
        ]
    ),
    np.array(
        [
            [6.36  , -3.07 , -2.8  , -3.42 , -0.68 ],
            [-3.07 , 10.19 ,  0.74 ,  0.5  ,  0.72],
            [-2.8  , 0.74  ,  4.97 , -1.48 ,  1.93],
            [-3.42 , 0.5   , -1.48 ,  4.9  , -0.97],
```

```
            [-0.68 , 0.72  ,  1.93 , -0.97 ,  3.21 ]
        ]
    )
]


b = [
        np.array(
            [[-0.24, 0.99]]
        ),
        np.array(
            [[0.66, 0.37, -2.06, 0.14, 1.36]]
        ),
]


x = [
        np.array(
            [[-38.12, -55.87]]
        ),
        np.array(
            [[4.60,  6.85,  4.31,  4.79,  8.38]]
        )
]

for (A_i, b_i, x_i), fact in itertools.product(zip(A,b,x),[1.1,.9]):
    eig_vals = np.linalg.eigvals(A_i)
    eig_vals = eig_vals[np.isreal(eig_vals)].real
    square_function_test(A_i, b_i, x_i, fact*(2/max(eig_vals)), 2000, np.
 ↪finfo(float).eps**(1/2))
```

```
El algoritmo no convergió en:
k = 2000, ||g_k|| = inf, x_k = [-4.77996787e+159 -1.42775834e+160]
El algoritmo convergió en:
k = 105, ||g_k|| = 1.413707145775169e-08, x_k = [-0.45696914  0.43365738]
El algoritmo no convergió en:
k = 2000, ||g_k|| = inf, x_k = [-7.41437598e+158  9.63088421e+158
3.28353818e+158]
El algoritmo convergió en:
k = 1064, ||g_k|| = 1.471192113738863e-08, x_k = [-2.77194407 -0.52190805
-3.05959477]
```

Comentarios sobre las trayectorias:

No convergio el algoritmo, cuando tomamos un paso fijo mayor a $\frac{2}{\lambda_{\max}(A)}$ como habiamos demostrado, pero si converge cuando es menor.

1.3 Ejercicio 2 (4 puntos)

Pruebe el método de descenso máximo con paso fijo aplicado a la función de Rosenbrock.

Encuentre un valor adecuado para α para que el algoritmo converja. Use como punto inicial el punto $(-12, 10)$.

Imprima x_k, $\|g_k\|$, el número de iteraciones k y el valor de res.

1.3.1 Solución:

[ ]:
```python
# 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)]] )
```

[ ]:
```python
# Pruebas realizadas a la función de Rosenbrock
tl=0
tr=1
convergence=True
ans=0
x_0 = np.array([[-12,10]])
while(convergence):
  trd1 = tl+(tr-tl)/3
  trd2 = tl+2*((tr-tl)/3)
  ans1 = gradient_descent_fixed_step(gf_rosenbrock, x_0, trd1, 20000, np.
 ↪finfo(float).eps**(1/2))
  ans2 = gradient_descent_fixed_step(gf_rosenbrock, x_0, trd2, 20000, np.
 ↪finfo(float).eps**(1/2))
  if(ans1[3]==1):
    convergence = False
    ans = trd1
  if(ans2[3]==1):
    convergence = False
    ans = trd2
  if(np.linalg.norm(ans1[1]) > np.linalg.norm(ans2[1])):
    tl = trd1
  else:
    tr = trd2
test(gf_rosenbrock, x_0, trd1, 20000, np.finfo(float).eps**(1/2))
```

```
C:\Users\batma\AppData\Local\Temp\ipykernel_10164\2467917378.py:7:
RuntimeWarning: overflow encountered in double_scalars
  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)]] )
C:\Users\batma\AppData\Local\Temp\ipykernel_10164\2467917378.py:7:
RuntimeWarning: invalid value encountered in double_scalars
  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)]] )
```

```
El algoritmo convergió en:
k = 1904, ||g_k|| = 1.4808304112176072e-08, x_k = [0.99999999 0.99999998]
```

Se uso una busqueda ternaria para encontrar el paso adecuado para que el descenso por gradiente con paso fijo converja con la función Rosenbrock.