Tarea 8 Optimización

Roberto Vásquez Martínez
Profesor: Joaquín Peña Acevedo

01/Mayo/2022

1 Ejercicio 1 (3 puntos)

Sea $x = (x_1, x_2, ..., x_n)$ la variable independiente.

Programar las siguientes funciones y sus gradientes:

- Función cuadrática

$$f(\mathbf{x}) = 0.5\mathbf{x}^\top \mathbf{A}\mathbf{x} - \mathbf{b}^\top \mathbf{x}.$$

Si $\mathbf{I}$ es la matriz identidad y $\mathbf{1}$ es la matriz llena de 1's, ambas de tamaño n, entonces

$$\mathbf{A} = n\mathbf{I} + \mathbf{1} = \begin{bmatrix} n & 0 & \cdots & 0 \\ 0 & n & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & n \end{bmatrix} + \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

- Función generalizada de Rosenbrock

$$f(x) = \sum_{i=1}^{n-1} 100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2$$

$$x_0 = (-1.2, 1, -1.2, 1, ..., -1.2, 1)$$

En la implementación de cada función y de su gradiente, se recibe como argumento la variable x y definimos n como la longitud del arreglo x, y con esos datos aplicamos la definición correspondiente.

Estas funciones van a ser usadas para probar los algoritmos de optimización. El punto x_0 que aparece en la definición de cada función es el punto inicial que se sugiere para el algoritmo de optimización.

1.1 Solución:

A continuación, importamos el módulo `lib_t8.py` que contiene la definición de las funciones anteriores y sus gradientes. Sea f_Q y f_R las funciones cuadráticas y generalizada de Rosenbrock respectivamente. Vamos a probar estas funciones en el punto $\mathbf{x} = (1, 1)$.

Sabemos que $\mathbf{x}$ es punto crítico de f_R con óptimo $f_R(\mathbf{x}) = 0$, mientras que, haciendo unos cuantos cálculos podemos ver que $f_Q(\mathbf{x}) = 2$ y $\nabla f_Q(\mathbf{x}) = (3,3)^T$. Compararemos estos valores con los resultados numéricos.

En primer lugar, evaluamos en $\mathbf{x}$ la función cuadrática f_Q programada

[2]:
```
import lib_t8
import importlib
importlib.reload(lib_t8)
from lib_t8 import *

# Punto de prueba
x_proof=np.array([1.0,1.0])

# Implementación de la función cuadrática y su gradiente
print('Valor de la funcion cuadrática en (1,1): ',quad_fun(x_proof))
print('Valor del gradiente de la funcion cuadrática en (1,1):␣
 ↪',grad_quad_fun(x_proof))
```

```
Valor de la funcion cuadrática en (1,1):  2.0
Valor del gradiente de la funcion cuadrática en (1,1):  [[3.]
 [3.]]
```

Realizamos la misma prueba para la función generalizada de Rosenbrock, sabiendo que $f_R(\mathbf{x}) = 0$ y $\nabla f_R(\mathbf{x}) = 0$

[3]:
```
#  Implementación de la función generalizada de Rosenbrock y su gradiente
print('Valor de la funcion generalizada de Rosenbrock en (1,1):␣
 ↪',gen_rosenbrock(x_proof))
print('Valor del gradiente de la funcion generalizada de Rosenbrock en (1,1):␣
 ↪',grad_gen_rosenbrock(x_proof))
```

```
Valor de la funcion generalizada de Rosenbrock en (1,1):  0.0
Valor del gradiente de la funcion generalizada de Rosenbrock en (1,1):  [-0.
0.]
```

2 Ejercicio 2 (3.5 puntos)

Programar el método de gradiente conjugado no lineal de Fletcher-Reeves:

La implementación recibe como argumentos a la función objetivo f, su gradiente ∇f, un punto inicial x_0, el máximo número de iteraciones N y una tolerancia $\tau > 0$.

1. Calcular $\nabla f_0 = \nabla f(x_0)$, $p_0 = -\nabla f_0$ y hacer $res = 0$.
2. Para $k = 0, 1, ..., N$:

- Si $\|\nabla f_k\| < \tau$, hacer $res = 1$ y terminar el ciclo
- Usando backtracking calcular el tamaño de paso α_k
- Calcular $x_{k+1} = x_k + \alpha_k p_k$

- Calcular $\nabla f_{k+1} = \nabla f(x_{k+1})$
- Calcular

$$\beta_{k+1} = \frac{\nabla f_{k+1}^\top \nabla f_{k+1}}{\nabla f_k^\top \nabla f_k}$$

- Calcular

$$p_{k+1} = -\nabla f_{k+1} + \beta_{k+1} p_k$$

3. Devolver $x_k, \nabla f_k, k, res$

---

1. Escriba la función que implemente el algoritmo anterior.
2. Pruebe el algoritmo usando para cada una de las funciones del Ejercicio 1, tomando el punto x_0 que se indica.
3. Fije $N = 50000$, $\tau = \epsilon_m^{1/3}$.
4. Para cada función del Ejercicio 1 cree el punto x_0 correspondiente usado $n = 2, 10, 20$ y ejecute el algoritmo. Imprima

- n,
- f(x0),
- las primeras y últimas 4 entradas del punto x_k que devuelve el algoritmo,
- f(xk),
- la norma del vector ∇f_k,
- el número k de iteraciones realizadas,
- la variable res para saber si el algoritmo puedo converger.

2.1 Solución:

Actualizamos el módulo con el que estamos trabajando, definimos la tolerancia, el número máximo de iteraciones y el parámetro ρ del algoritmo de backtracking

[71]:
```
importlib.reload(lib_t8)
import lib_t8
from lib_t8 import *

# Tolerancia y numero maximo de iteraciones
tol=np.finfo(float).eps**(1/3)
N=50000
rho=0.8
```

Función cuadrática En primer lugar, probamos el algoritmo de Fletcher-Reeves como en clase, en este caso como es una función cuadrática es equivalente al método de gradiente conjugado. A continuación, probamos el desempeño en esta función para $n = 2, 10, 20$

Por la proposición 1 de la Clase 17, el método de gradiente conjugado en este caso es globalmente convergente y el punto óptimo es $\mathbf{x}_* = \frac{1}{2n}(1, 1, \ldots, 1)^T$. Daremos cualquier punto inicial y compararemos con el óptimo correspondiente, en este caso usaremos la misma condición que se sugiere en la función generalizada de Rosenbrock

[72]:
```
# Valores de n
n=[2,10,20]
# Paras para cada valor de n
for nn in n:
    x0=np.tile([-1.2,1.0],int(nn/2))
    proof_fletcher_reeves(quad_fun,grad_quad_fun,x0,N,tol,rho)
    print('\n\n')
```

```
El algoritmo de Fletcher-Reeves CONVERGE
n =  2
f(x0) =  2.66
xk =  [0.25000229 0.2499987 ]
k =  91
fk =  -0.24999999999256428
||gk|| =  5.803891132856387e-06



El algoritmo de Fletcher-Reeves CONVERGE
n =  10
f(x0) =  62.49999999999999
Primer y últimas 4 entradas de xk =  [0.04999983 0.05000018 0.04999983
0.05000018] … [0.04999983 0.05000018 0.04999983 0.05000018]
k =  134
fk =  -0.24999999999843664
||gk|| =  5.603306622825684e-06



El algoritmo de Fletcher-Reeves CONVERGE
n =  20
f(x0) =  248.0
Primer y últimas 4 entradas de xk =  [0.02499995 0.02500007 0.02499995
0.02500007] … [0.02499995 0.02500007 0.02499995 0.02500007]
k =  138
fk =  -0.2499999999992326
||gk|| =  5.707639527877831e-06
```

Como usamos backtracking para estimar el óptimo del tamaño de paso no se necesariamente se obtiene la solución en n pasos y en efecto hemos obtenido los vectores solución de la forma

$\frac{1}{2n}(1,1,\ldots,1)$ para $n = 2, 10, 20$, que corresponden a los valores donde la función alcanza su óptimo en cada caso.

Función Generalizada de Rosenbrock Repetimos la prueba ahora con la función generalizada de Rosenbrock, sabiendo que el óptimo es $\mathbf{x}_* = (1,1,\ldots,1)$

[73]:
```
# Paras para cada valor de n
for nn in n:
    x0=np.tile([-1.2,1.0],int(nn/2))
    proof_fletcher_reeves(gen_rosenbrock,grad_gen_rosenbrock,x0,N,tol,rho)
    print('\n\n')
```

```
El algoritmo de Fletcher-Reeves CONVERGE
n =  2
f(x0) =  24.199999999999996
xk =  [0.99999539 0.99999077]
k =  251
fk =  2.1277093635685042e-11
||gk|| =  4.780235391540514e-06



El algoritmo de Fletcher-Reeves CONVERGE
n =  10
f(x0) =  2057.0
Primer y últimas 4 entradas de xk =  [0.99999999 1.         1.         1.
] … [0.99999999 0.99999998 0.99999995 0.9999999 ]
k =  2019
fk =  2.193108694121752e-14
||gk|| =  5.749420469861276e-06



El algoritmo de Fletcher-Reeves CONVERGE
n =  20
f(x0) =  4598.0
Primer y últimas 4 entradas de xk =  [1. 1. 1. 1.] … [0.99999999 0.99999998
0.99999997 0.99999993]
k =  38376
fk =  1.5946132549678812e-14
||gk|| =  6.039770533007446e-06
```

3 Ejercicio 3 (3.5 puntos)

Programar el método de gradiente conjugado no lineal de usando la fórmula de Hestenes-Stiefel:

En este caso el algoritmo es igual al del Ejercicio 2, con excepción del cálculo de β_{k+1}. Primero se calcula el vector $\mathbf{y}_k$ y luego β_{k+1}:

$$\mathbf{y}_k = \nabla f_{k+1} - \nabla f_k$$
$$\beta_{k+1} = \frac{\nabla f_{k+1}^\top \mathbf{y}_k}{p_k^\top \mathbf{y}_k}$$

1. Repita el Ejercicio 2 usando la fórmula de Hestenes-Stiefel.
2. ¿Cuál de los métodos es mejor para encontrar los óptimos de las funciones de prueba?

3.1 Solución:

Haremos las mismas pruebas tanto para la función cuadrática como la función generalizada de Rosenbrock ahora utilizando las actualización de Hestenes-Stiefel.

Función cuadrática En la siguiente celda presentamos el resultado de Hestenes-Stiefel para la función f_Q

[74]:
```
importlib.reload(lib_t8)
import lib_t8
from lib_t8 import *

# Paras para cada valor de n
for nn in n:
    x0=np.tile([-1.2,1.0],int(nn/2))
    proof_hestenes_stiefel(quad_fun,grad_quad_fun,x0,N,tol,rho)
    print('\n\n')
```

```
El algoritmo de Hestenes-Stiefel CONVERGE
n =  2
f(x0) =  2.66
xk =  [0.25000217 0.24999881]
k =  66
fk =  -0.24999999999340244
||gk|| =  5.497058549749721e-06



El algoritmo de Hestenes-Stiefel CONVERGE
n =  10
f(x0) =  62.49999999999999
Primer y últimas 4 entradas de xk =  [0.05000019 0.04999985 0.05000019
0.04999985] … [0.05000019 0.04999985 0.05000019 0.04999985]
k =  162
```

```
fk =  -0.24999999999844713
||gk|| =  5.650924470696626e-06
```

```
El algoritmo de Hestenes-Stiefel CONVERGE
n =  20
f(x0) =  248.0
Primer y últimas 4 entradas de xk =  [0.02499993 0.02500004 0.02499993
0.02500004] … [0.02499993 0.02500004 0.02499993 0.02500004]
k =  227
fk =  -0.24999999999929282
||gk|| =  5.793279751916127e-06
```

Función Generalizada de Rosenbrock Se puede ver que en el caso cuadrático el desempeño es el mismo porque ambas actualizaciones son equivalente al método de gradiente conjugado para el caso cuadrática.

El caso de mayor interés, y en el que esperamos descrepancias es utilizando como función de prueba a la función f_R.

A continuación, presentamos el resumen de las pruebas

[75]:
```
# Paras para cada valor de n
for nn in n:
    x0=np.tile([-1.2,1.0],int(nn/2))
    proof_hestenes_stiefel(gen_rosenbrock,grad_gen_rosenbrock,x0,N,tol,rho)
    print('\n\n')
```

```
El algoritmo de Hestenes-Stiefel CONVERGE
n =  2
f(x0) =  24.199999999999996
xk =  [1.0000003 1.0000006]
k =  110
fk =  1.0800792500369892e-13
||gk|| =  5.942872244700956e-06
```

```
El algoritmo de Hestenes-Stiefel CONVERGE
n =  10
f(x0) =  2057.0
Primer y últimas 4 entradas de xk =  [1. 1. 1. 1.] … [1.00000001 1.00000002
1.00000004 1.00000009]
k =  783
fk =  1.4272406448603126e-14
```

```
||gk|| =  5.906937683335181e-06
```

```
El algoritmo de Hestenes-Stiefel CONVERGE
n =  20
f(x0) =  4598.0
Primer y últimas 4 entradas de xk =  [1. 1. 1. 1.] … [0.99999994 0.99999988
0.99999976 0.99999953]
k =  503
fk =  8.817321104442124e-14
||gk|| =  5.955100340841743e-06
```

Comparando Frechet-Reeves con Hestenes-Stiefel, Frechet-Reeves tiene un mejor comportamiento en el caso de la función f_Q en general, sin embargo, en el caso de la función de Rosenbrock, el comportamiento de Hestenes-Stiefel es mucho mejor que Frechet-Reeves respecto al número de iteraciones.