Tarea 9 Optimización

Roberto Vásquez Martínez
Profesor: Joaquín Peña Acevedo

08/Mayo/2022

1 Ejercicio 1 (2 puntos)

Programar las siguientes funciones y sus gradientes, de modo que dependan de la dimensión n de la variable $\mathbf{x}$:

- Función "Tridiagonal 1" generalizada

$$f(x) = \sum_{i=1}^{n-1} (x_i + x_{i+1} - 3)^2 + (x_i - x_{i+1} + 1)^4$$

- Función generalizada de Rosenbrock

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

1.1 Solución

Sea f_T la función tridiagonal generalizada y f_R la función generalizada de Rosenbrock. Vamos a evaluar estas funciones en $\mathbf{x}_* = (1, 1)$.

Sabemos que $\mathbf{x}_*$ es punto crítico de f_R, además es el punto donde se alcanza el valor óptimo de la función para $n = 2$ con $f_R(\mathbf{x}_*) = 0$.

Por otro lado, haciendo algunas cuentas vemos que $f_T(\mathbf{x}_*) = 2$ mientras que $\nabla f_T(\mathbf{x}_*) = [2, -6]^T$.

A continuación, importamos el módulo `lib_t9` que contiene las implementaciones de f_T y f_R, y probamos el desempeño de estas funciones con los valores teóricos en $\mathbf{x}_* = (1, 1)$.

En primer lugar, mostramos este desempeño con la función f_T

[1]:
```
# Implementación de la funciones y sus gradientes
import lib_t9
import importlib
importlib.reload(lib_t9)
from lib_t9 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 tridiagonal generalizada en (1,1):␣
 ↪',tri_diag_gen(x_proof))
print('Valor del gradiente de la funcion tridiagonal generalizada en (1,1):␣
 ↪',grad_tri_diag_gen(x_proof))
```

```
Valor de la funcion tridiagonal generalizada en (1,1):  2.0
Valor del gradiente de la funcion tridiagonal generalizada en (1,1):  [[ 2.]
 [-6.]]
```

Ahora mostramos el resultado de la prueba con la función generalizada de Rosenbrock sabiendo que $f_R(\mathbf{x}_*) = 0$ y $\nabla f_R(\mathbf{x}_*) = 0$.

[2]:
```
#  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 (8 puntos)

Programar y probar el método BFGS modificado.

1. Programar el algoritmo descrito en la diapositiva 16 de la clase 23. Agregue una variable *res* que indique si el algoritmo terminó porque se cumplió que la magnitud del gradiente es menor que la toleracia dada.
2. Probar el algoritmo con las funciones del Ejercicio 1 con la matriz H_0 como la matriz identidad y el punto inicial x_0 como:

- La función generalizada de Rosenbrock:

$$x_0 = (-1.2, 1, -1.2, 1, ..., -1.2, 1) \in \mathbb{R}^n$$

- La función Tridiagonal 1 generalizada:

$$x_0 = (2, 2, ..., 2) \in \mathbb{R}^n$$

Pruebe el algoritmo con la dimensión $n = 2, 10, 100$.

3. Fije el número de iteraciones máximas a $N = 50000$, y la tolerancia $\tau = \epsilon_m^{1/3}$, donde ϵ_m es el épsilon máquina, para terminar las iteraciones si la magnitud del gradiente es menor que τ. En cada caso, imprima los siguiente datos:

- n,

- $f(x_0)$,
- Usando la variable *res*, imprima un mensaje que indique si el algoritmo convergió,
- el número k de iteraciones realizadas,
- $f(x_k)$,
- la norma del vector ∇f_k, y
- las primeras y últimas 4 entradas del punto x_k que devuelve el algoritmo.

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 usado en el algoritmo BFGS modificado

[3]:
```
importlib.reload(lib_t9)
import lib_t9
from lib_t9 import *

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

2.1.1 Función tridiagonal generalizada

En primer lugar, probamos el algoritmo BFGS modificado con la función tridiagonal generalizada considerando $\mathbf{x}_0 = (2, 2, ..., 2) \in \mathbb{R}^n$ y $H_0 = I_n$, la matriz identidad de tamaño n. Probamos el desempeño del algoritmo tomando $n = 2, 10, 20$.

Por otro lado, es fácil ver que el valor óptimo de f_T es 0 cuando $n = 2$ y se obtiene cuando $\mathbf{x}_* = (1, 2)$, por lo que esperaríamos que el algoritmo modificado BFGS devuelva un $\mathbf{x}_k$ tal que $\mathbf{x}_k \approx (1, 2)$ y $f_T(\mathbf{x}_k) \approx 0$ en el caso $n = 2$.

[4]:
```
# Valores de n
n=[2,10,20]
# Paras para cada valor de n
for nn in n:
    x0=np.tile([2.0,2.0],int(nn/2))
    proof_mod_BFGS(tri_diag_gen,grad_tri_diag_gen,x0,np.eye(len(x0)),N,tol,rho)
    print('\n\n')
```

```
El algoritmo BFGS modificado CONVERGE
n =  2
f(x0) =  2.0
xk =  [1.00254814 1.99745384]
k =  29
fk =  6.77419173979345e-10
||gk|| =  5.645790494903032e-06
```

```
El algoritmo BFGS modificado CONVERGE
n =  10
f(x0) =  18.0
Primer y últimas 4 entradas de xk =  [1.0246468  1.34361033 1.43890902
1.47645337] ... [1.5235467  1.56109104 1.65638982 1.9753535 ]
k =  109
fk =  7.211216703292181
||gk|| =  4.86950935670342e-06
```

```
El algoritmo BFGS modificado CONVERGE
n =  20
f(x0) =  38.0
Primer y últimas 4 entradas de xk =  [1.02448178 1.34326073 1.43811809
1.47459085] ... [1.52540918 1.56188185 1.65673918 1.97551787]
k =  127
fk =  17.210307642088182
||gk|| =  5.956629884465766e-06
```

2.1.2 Función Generalizada de Rosenbrock

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

[5]:
```
# 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_mod_BFGS(gen_rosenbrock,grad_gen_rosenbrock,x0,np.
 ↪eye(len(x0)),N,tol,rho)
    print('\n\n')
```

```
El algoritmo BFGS modificado CONVERGE
n =  2
f(x0) =  24.199999999999996
xk =  [1.00000076 1.00000153]
k =  112
fk =  5.803466762986857e-13
||gk|| =  8.008587744718231e-07
```

```
El algoritmo BFGS modificado CONVERGE
n =  10
f(x0) =  2057.0
Primer y últimas 4 entradas de xk =  [1. 1. 1. 1.] ... [1.00000004 1.00000007
1.00000014 1.00000028]
k =  1232
fk =  4.310855071996698e-14
||gk|| =  5.491033316798961e-06
```

```
El algoritmo BFGS modificado CONVERGE
n =  20
f(x0) =  4598.0
Primer y últimas 4 entradas de xk =  [1. 1. 1. 1.] ... [1.00000003 1.00000007
1.00000012 1.00000025]
k =  2659
fk =  4.356059233642792e-14
||gk|| =  5.541646466284757e-06
```