Trabajo Práctico de Optimización no Lineal

Investigación Operativa (M) - Introducción a la Investigación Operativa y Optimización (LCD)

Segundo cuatrimestre 2024

El trabajo debe ser resuelto en grupos de hasta tres personas, manteniendo los grupos armados para el primer parcial. Se deberá entregar un único archivo `.ipynb`*, enviado a bersetche@gmail.com, que debe contener tanto las soluciones de los ejercicios como comentarios de los resultados obtenidos, con las justificaciones pertinentes basadas en lo visto en clase. La fecha límite de entrega es el 15/11 a las 23:55. En el caso de haber errores graves, podrán contar con una fecha de reentrega.*

No se pueden utilizar librerías de Python, salvo `numpy`, `matplotlib`, `plotly` y librerías nativas.

Ejercicio 1.

1. Para funciones cuadráticas $f(x) = \frac{1}{2}x^T Ax$ con matrices $A \in \mathbb{R}^{10\times 10}$ definidas positivas y diagonales, elaborar una serie de tests tales que muestren las dificultades del Método del Gradiente con longitud de paso óptima a medida que aumenta el orden de diferencia entre el menor y el mayor autovalor. Sea t_k^* la longitud del paso óptimo en la k-ésima iteración, comparar en cada caso con los resultados obtenidos tomando:
   - $\frac{3}{4}t_k^*$
   - $\frac{1}{2}t_k^*$
   - $\frac{1}{4}t_k^*$
   - $\gamma_k t_k^*$ donde γ_k es un número aleatorio en $(0, 1]$
2. Para los tests elaborados anteriormente, comprar el desempeño del método entre tomar:
   - t_k^*
   - $0{,}9t_k^*$
   - $0{,}99t_k^*$
3. Las direcciones del Método del Gradiente con longitud de paso óptima son ortogonales entre sí en iteraciones consecutivas: $\langle -\nabla f(x_k), -\nabla f(x_{k+1})\rangle = 0$. Utilizando la longitud de paso óptima, intentar quebrar esta ortogonalidad rotando la dirección con un ángulo de:
   - $\theta = -\frac{1}{3}\pi$
   - $\theta = -\frac{1}{6}\pi$
   - $\theta = \frac{1}{6}\pi$
   - $\theta = \frac{1}{3}\pi$

   La dirección vendría dada por la siguiente fórmula:
   $$d_k = \begin{bmatrix}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{bmatrix}(-\nabla f(x_k))$$
   Al realizar la rotación, debe chequearse que d_k sea de descenso. En caso contrario, en esa iteración tomar como dirección $d_k = -\nabla f(x_k)$. Para este ítem, generar tests como los del ítem 1. pero con $A \in \mathbb{R}^{2\times 2}$. Para cada valor de θ y para cada test, comparar con el Método de Gradiente con longitud de paso óptima.

Ejercicio 2.

1. Para funciones cuadráticas $f(x) = \frac{1}{2}x^T Ax$ con matrices $A \in \mathbb{R}^{n\times n}$ definidas positivas y diagonales, elaborar una serie de tests para corroborar la terminación en tantos pasos como autovalores distintos de A del Gradiente Conjugado.
2. Sea t_k^* la longitud del paso óptimo en la k-ésima iteración y sea β_k el coeficiente de corrección, experimentar qué sucede si en vez de utilizar (t_k^*, β_k) se utiliza:

- $(0{,}9t_k^*, \beta_k)$
- $(0{,}99t_k^*, \beta_k)$
- $(t_k^*, 0{,}9\beta_k)$
- $(t_k^*, 0{,}99\beta_k)$
- $(0{,}9t_k^*, 0{,}9\beta_k)$
- $(0{,}99t_k^*, 0{,}99\beta_k)$

Comentar cómo el desempeño de Gradientes Conjugados depende de la exactitud de la búsqueda lineal.

3. Comparar el desempeño de Gradiente Conjugado con el Método del Gradiente para funciones cuadráticas con matrices muy mal condicionadas (es decir, que $\frac{\lambda_{\text{máx}}}{\lambda_{\min}}$ sea muy grande).

Ejercicio 3. Para la matriz A que se encuentra en el archivo `matriz.npy`, hallar el minimizador de $f(x) = \frac{1}{2}x^T Ax$ con el Método de Newton y con cada uno de los Métodos Cuasi-Newton (Broyden Mala, DFP, BFGS, BBR). Para que este experimento sea relevante, el Método de Newton debe estar modificado de manera tal que la dirección de descenso se calcule utilizando la función `eq_lineal_chol` en vez de `np.linalg.solve`. La implementación de `eq_lineal_chol` está en el Notebook con código complementario.

Comparar la cantidad de iteraciones y los tiempos de ejecución. ¿A qué se debe la diferencia en la cantidad de tiempo requerido?

Detalles sobre implementación: Utilizar `A = np.load('matriz.npy')` para cargar la matriz desde el archivo (el cual debe encontrarse en la misma carpeta que el Notebook). Para medir el tiempo de ejecución de una parte del código:

```
from time import time

s = time()

############################################################
# CODIGO AL CUAL SE LE DESEA MEDIR EL TIEMPO DE EJECUCION
############################################################

print('Tiempo de ejecucion: ', time() - s)
```

Ejercicio 4.

1. Comparar el desempeño (en cantidad de iteraciones) del Método del Gradiente y sus variantes del Ejercicio 1 Item 1 con el Método de Barzilai-Borwein-Raydan. Se pueden utilizar los tests del Ejercicio 1 Item 1.
2. Sea $\{x_k\}_k$ la secuencia generada por BBR, realizar experimentos que permitan mostrar que la secuencia $\{f(x_k)\}_k$ no es necesariamente monótona decreciente. Para mostrar esto, realizar un gráfico de k contra $f(x_k)$.

   Para realizar un gráfico básico utilizando `matplotlib`, pueden referirse a esta plantilla:

   ```
   import numpy as np
   import matplotlib.pyplot as plt

   ktop = # Cantidad de iteraciones de BBR
   ks = np.arange(ktop)
   fs = # Lista con los valores de cada f(x_k)
   plt.plot(ks, np.array(fs))
   plt.show()
   ```

Ejercicio 5. Los métodos estudiados en la materia, en general, convergen a un mínimo local. Dada una función $f \in C^2$, la heurística más sencilla para intentar hallar un mínimo global (o al menos un *buen* mínimo local) consiste en correr repetidas veces el método comenzando desde distintos puntos aleatorios.

La función Bird está definida como:

$$f(x, y) = \sin(x)e^{\left[(1-\cos y)^2\right]} + \cos(y)e^{\left[(1-\sin x)^2\right]} + (x - y)^2$$

y tiene dos minimizadores globales:

$$(4{,}701055751981055;\ 3{,}152946019601391)$$
$$(-1{,}582142172055011;\ -3{,}130246799635430)$$

1. Graficar la función Bird con `plot_fun` en $[-10, 10] \times [-10, 10]$

2. Implementar la función `random_restart` que lleve a cabo la heurística para funciones $\mathbb{R}^2 \to \mathbb{R}$, comenzando con puntos aleatorios en el cuadrado $[-c, c] \times [-c, c]$. La función debe tomar como input la función `f` a minimizar, `N` la cantidad de puntos aleatorios a generar y `c` que describe el dominio donde se desean inicializar aleatoriamente los puntos. La función debe generar N puntos iniciales aleatorios, y a partir de cada uno de ellos, correr un Método de Cuasi-Newton (el mismo para todos los puntos iniciales). Debe devolver la mejor solución obtenida.

   Un punto aleatorio en $[-c, c] \times [-c, c]$ se puede obtener con:

   ```
   x = 2 * c * np.random.random(2) - c
   ```

3. Testear la función del ítem anterior a la función Bird en $[-15, 15] \times [-15, 15]$ con distintos valores de N. ¿Qué tan buena es la solución obtenida?