ANÁLISIS NUMÉRICO PARA INGENIERÍA

JUAN PABLO ALVARADO VILLALOBOS
KEVIN STEVEN CORDERO ZÚNIGA
EMANUEL ESQUIVEL LÓPEZ
LUIS LÓPEZ SALAS

Tarea 1 - II Parte

1. **Notas a tener en cuenta.**

Para el cálculo de λ_k se utilizó la ecuación de Armijo (la 2.6 en el documento de BFGS), esto se debe a que la ecuación 2.5 era restrictiva para este cálculo. La segunda inecuación de la ecuación 2.5, ocasiona que el método funcione, pero no es consistente, muchas veces debido a que los valores iniciales son aleatorios, el programa no es capaz de encontrar un lambda que cumple con las dos inecuaciones, siendo la segunda muy restrictiva, por lo que no se obtenían resultados correctos 2 de cada 5 veces. Debido a esto, se le preguntó al profesor como podíamos solucionarlo y nos mencionó que utilicemos la ecuación 2.6, debido a que no restringe el cálculo del lambda y ocasiona que los resultados sean más consistentes 5 de cada 5 veces. Por esa razón no se menciona la segunda inecuación en el pseudocódigo y en el código se encuentra comentada.

2. **Función de Prueba Elegida.**

Se seleccionó la función 117 de las funciones de prueba. La función Schumer Steiglitz:

$$f(x) = \sum_{i=1}^{D} x_i^4$$

El mínimo global de la función se encuentra en:

$$f(x) = 0$$

Para usar esta función en Octave, con el programa BFGS, se realiza de la siguiente manera:

```
[xk, k, error] = p2_bfgs('x**4 + y**4 + z**4 + w**4 + q**4', ['x' 'y' 'z' 'w' 'q'], 10**-5, 25)
```

Donde xk será un vector con las aproximaciones de convergencia de la función, k es la cantidad de iteraciones realizadas y el error indica la exactitud de las aproximaciones.

El primer argumento de la función es la función a optimizar, el segundo es un vector que contiene todas las variables presentes en la función, el tercero es la tolerancia del error (esto quiere decir que cuando el error sea menor que la tolerancia, el programa se detenga) y el último argumento es la cantidad de iteraciones máximas a realizar por el programa.

3. **Pseudocódigo.**

- Valores Iniciales: Función f a optimizar, una lista con las variables presentes en la función y números de $Tol \in \mathbb{R} > 0$ y $MáxIter \in \mathbb{R} > 0$.

- Pasos:

1. Se obtiene el vector inicial x_k, este vector tendrá un largo equivalente a la cantidad de variables presentes en la función y todos sus valores serán aleatorios mayores que cero.
2. Se obtiene el gradiente de la función g.
3. Se evalúa el vector x_k en el gradiente para obtener $g(x_k)$.
4. Se obtiene la matriz identidad (b_k) de dimensión $n\,x\,n$, donde n es la cantidad de variables presentes en la función.
5. $\alpha = 1$.
6. $\varepsilon = 1$.
7. Se calculan los valores de σ_1 y σ_2 de manera aleatoria, donde $0 < \sigma_1 < 0.5 < \sigma_2 < 1$.
8. $k = 0$.
9. $error = tol + 1$.
10. Mientras $error > tol$ y $k < iterMax$. (Abarca del paso 11 al 40).
11. $p_k = b_k^{-1} * -g(x_k)$.
12. $\lambda_k = 1$.
13. $count = 1$.
14. $check = false$.
15. Mientras $true$. (Abarca del paso 16 al 24).
16. Si $f(x_k + \lambda_k * p_k) <= f(x_k) + \sigma_1 * \lambda_k * g(x_k)^T * p_k$.
17. $count = 1$.
18. Fin del mientras del paso 15.
19. Si $count > 10$.
20. $check = true$.
21. Fin del mientras del paso 15.
22. Sino
23. $\lambda_k = \lambda_k / 5$.
24. $count \mathrel{+}= 1$.
25. Si $check = true$.
26. Fin del mientras del paso 10.
27. $x_{k+1} = x_k + \lambda_k * p_k$.
28. $s_k = x_{k+1} - x_k$.
29. Se evalúa el vector x_{k+1} en el gradiente para obtener $g(x_{k+1})$.
30. $y_k = g(x_{k+1}) - g(x_k)$.
31. Si $(y_k^T * s_k) / \|s_k\|^2 >= \varepsilon * \| g(x_k)\|^\alpha$
32. $b_{k+1} = b_k - (b_k * s_k * s_k^T * b_k) / (s_k^T * b_k * s_k) + (y_k * y_k^T) / (y_k^T * s_k)$.
33. Sino
34. $b_{k+1} = b_k$.
35. $b_k = b_{k+1}$.
36. $x_k = x_{k+1}$.
37. $g(x_k) = g(x_{k+1})$.

38. $error = \| \nabla f(x_k) \|$.
39. $k = k + 1$.
40. Iterar.
41. Fin.

- Valores Finales: Vector xk, k, error.

4. Código en GNU Octave.

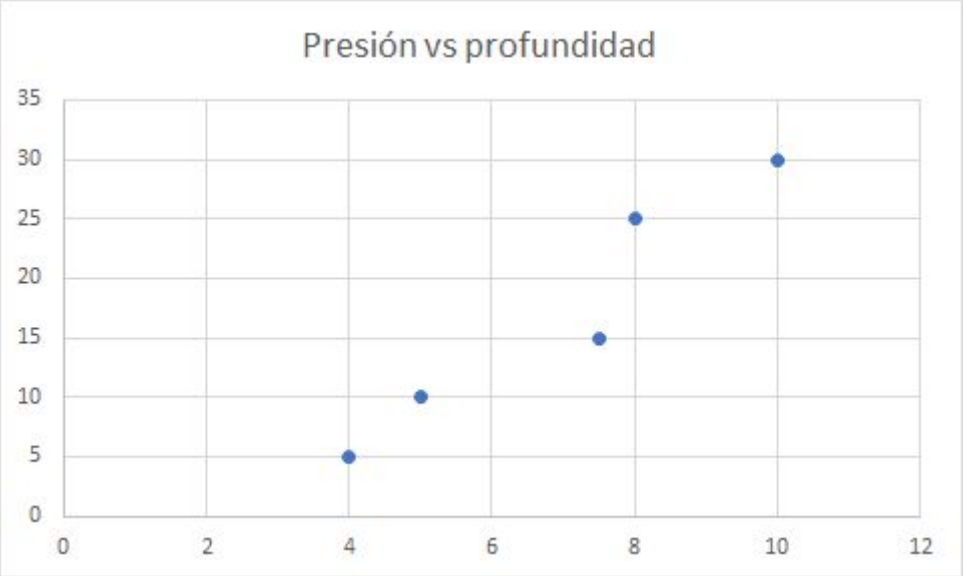
Este código se puede encontrar en el archivo adjunto p2_bfgs.m

5. Problema de Ingeniería.

El problema de la ingeniería seleccionado, pertenece al ámbito de ciencias de la computación pero tiene amplias aplicaciones dentro de diversas ingenierías. El problema en cuestión es acerca la temática del "Machine Learning" un área que estudia métodos o algoritmos para que las máquinas puedan "aprender" a partir de extensas bases de datos. Un aplicación de esto es en la clasificación de correos donde se toman datos como el remitente, cantidad de palabras, … etc. para clasificar un correo como spam.
Existen muchos algoritmos y problemas que se pueden solucionar con "Machine Learning" pero el comportamiento en todos es tomar datos o hechos y entrenar un algoritmo para que realice predicciones aproximadas. Por esto se usa en muchos sectores desde simulaciones aerodinámicas hasta en las finanzas, y por eso se considera un problema de ingeniería.
El problema que queremos resolver en este informe consiste en analizar datos de proporción lineal y generar un algoritmo que pueda predecir nuevos datos.
Para ilustrar mejor este problema se toma el caso de un diseñador de submarinos que sabe que la presión depende de muchos factores, como la forma, de la superficie, las corrientes, la altitud,..etc. Pero quiere predecir la presión en las paredes externas usando el factor principal que es la profundidad. Para esto se cuenta con datos del valor de algunas presiones medidas vs. su profundidad:

Profundidad(km)	Presión(kPa)
5	10
7.5	15
8	25
4	5
10	30

Nota: Los datos son aproximados y no son de ningún caso real. Esta clase de problemas suele contar con miles de datos. Pero para fines ilustrativos y para que la convergencia no tome demasiado tiempo se usan solo 5. Además no se eligen problemas demasiado complejos complejos de ingeniería por ese mismo motivo, pero el procedimiento sería el mismo: tomar los datos y entrenar al algoritmo.
Los datos se pueden visualizar en la siguiente gráfica, tomando la profundidad en el eje X y la presión en el eje Y:



Entonces el objetivo a lograr es **obtener una función que tome una profundidad y me devuelva una presión:**

$Pred_i(x) \approx y$

Como se puede ver los datos siguen una tendencia lineal entonces se establece la predicción de la forma:

$Pred_i(m,b) = x_i * m + b$

Donde el factor x_i es un dato fijo, pero lo tanto los factores m y b se pueden ajustar para lograr predicciones más o menos exactas. Para saber que tan bien están los factores se usa algo conocido como una **"función costo"**, la cual determina que tanto se aleja la predicción de los valores reales en cada punto:

$$costo(m,b) = \sum_{i=0}^{n} (Pred_i - y_i)^2$$

Y el gran objetivo en "Machine Learning" en general es reducir la "función costo" a un valor mínimo, variando algunos factores, en este caso m y b. Este paso es lo que se conoce como "entrenar" el algoritmo ya que se usarán los datos para mejorar las predicciones. Por lo que en este punto precisamente se usa el algoritmo BFGS para optimizar esta función de costo.

El código usado para calcular el m y b óptimos con un error de 10^{-5} se muestra en la siguiente ilustración (Es algo extenso porque se debe añadir toda la tabla de valores a la string):

```
[xk, k, error] = p2_bfgs('(10-m*5-b)**2+(15-m*7.5-b)**2+(25-m*8-b)**2+(5-m*4-b)**2+(30-m*10-b)**2',['m','b'], 10**-5,5)
```

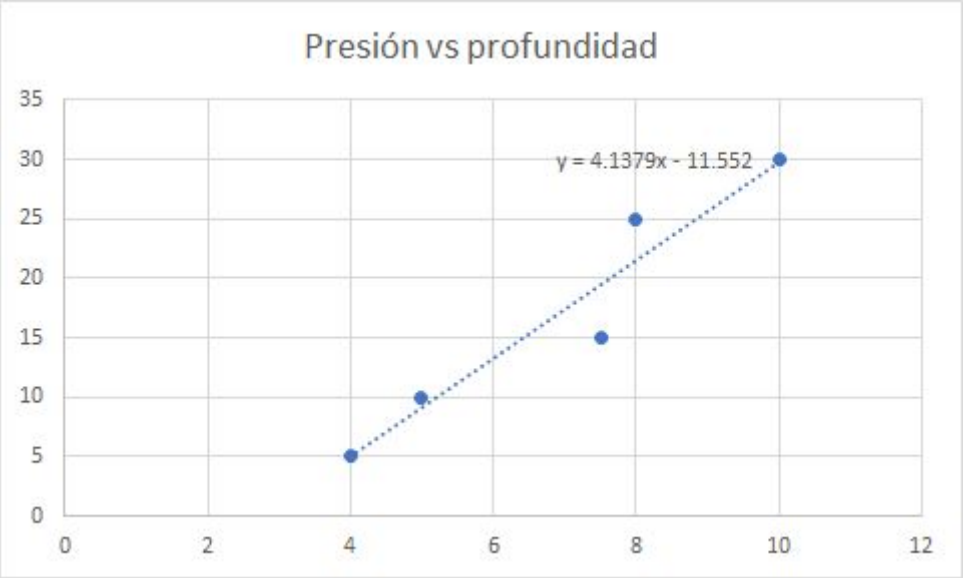
<u>Nota</u>: Puede tomar bastante tiempo en converger ya que tiene muchos datos a operar en cada iteración. Y esto mismo en algunos casos provoca que el lambda nunca converga, por la que se puede retirar el segundo criterio para que converja con más seguridad.

Después de ejecutar ese código se obtienen los valores óptimos :

$m = 4.1379$

$b = -11.552$

La curva resultante sería la siguiente:



Como se puede ver con los factores m y b finales se logra una línea muy cercana a los valores reales. Y con la ecuación de predicción se pueden realizar miles de predicciones para una gran cantidad de valores de profundidad. Claro entre más casos reales se usen más fiables serán las predicciones. Para concluir se puede afirmar que esta clase de algoritmos son muy versátiles ya que se ajustan a datos reales, y le dan un uso a grandes cantidades de datos.

<u>Nota:</u> este mismo método se podría utilizar también para distribuciones parabólicas si se ajusta la predicción de acuerdo al caso. y también se podrían agregar más factores como por ejemplo la temperatura, en ese caso bastaría con agregarlo a la predicción y añadirle un factor:

$$Pred_i(A, B, C) = prof * A + temp * B + C$$

*Y junto a la **función costo** se podrían encontrar A, B y C óptimos.*

En el archivo p2_solucion_aplicacion.m, se puede encontrar la implementación del problema seleccionado. Los datos insertados son los que se pueden encontrar en la gráfica de Presión vs Profundidad. Al ejecutar la función, se puede corroborar que los valores de m y b son correctos y consistentes. Cabe indicar que el valor inicial xk en este problema también se calcula de manera aleatoria, solamente se le debe suministrar la tolerancia y la cantidad de iteraciones máximas.

Bibliografía consultada:

Bonaccorso, G., 2017. *Machine Learning Algorithms*. 1st ed. Packt Publishing Ltd, pp.72-93.