Hoja de Trabajo 5 - Introducción a los Algoritmos de Optimización para Ciencia de Datos

Eduardo Andrés Santizo Olivet (21007361)

1. Determine si las siguientes proposiciones son verdaderas o falsas; en cualquier caso, justifique su respuesta.

   a) En general, los algoritmos de optimización nos proveen una expresión matemática *cerrada* para poder resolver un problema de optimización.

   b) La tasa de convergencia de un algoritmo de optimización describe que tan "rápido" dicho algoritmo converge a la solución del problema.

   c) En la práctica, un algoritmo con una tasa de convergencia cuadrática se aproxima más rápido a la solución que uno con tasa de convergencia lineal.

   d) En la práctica, un algoritmo con tasa de convergencia superlineal es comparable a uno con tasa de convergencia cuadrática.

   e) Si un algoritmo tiene tasa de convergencia r con constante C, esto significa que, en cualquier iteración k, se observa que $e_{k+1} = C\, e_k^r$.

   a. Falso: Incluso para funciones sumamente simples es imposible despejar una ecuación para obtener la solución real del problema. En estos casos es necesaria la utilización de un método iterativo para obtener un aproximado de la solución.
   b. Falso: La tasa de convergencia describe la velocidad a la cual una sucesión convergente se acerca a su valor límite (el cual no necesariamente consiste exactamente de su solución).
   c. Falso: En la práctica, una tasa de convergencia super-lineal (una variante de la tasa lineal) y una cuadrática son muy similares, aunque en la teoría la cuadrática siempre resulta más rápida.
   d. Verdadero: Aunque en la teoría una tasa de convergencia cuadrática alcanza más rápidamente una mayor precisión a comparación de una tasa super-lineal, en la práctica estas dos tasas de convergencia tienden a alcanzar un resultado satisfactorio a velocidades similares.
   e. Falso: Durante las primeras iteraciones no existe una garantía de observar una reducción de la norma del error a una tasa dada. La tasa de convergencia solo es posible de ser observada en el límite.

2. Considere la sucesión definida mediante:

$$x_0 = a > 0 \quad \text{y} \quad x_{k+1} = \frac{1}{2}\left(x_k + \frac{a}{x_k}\right).$$

   Si $x_k \to \sqrt{a}$ cuando $k \to \infty$, determine la tasa de convergencia (*rate of convergence*) y la constante (*rate constant*) respectiva para la sucesión $\{x_k\}_{k=0}^{\infty}$.

$$\lim_{k\to\infty} \left(\frac{1}{2}\left(x_k + \frac{A}{x_k}\right)\right) = \sqrt{A} \qquad x^* = \sqrt{A}$$

$$e_k = |x_k - x^*| = |x_k - \sqrt{A}|$$

$$e_{k+1} = |x_{k+1} - x^*|$$

$$= \left|\frac{1}{2}\left(x_k + \frac{A}{x_k}\right) - \sqrt{A}\right|$$

$$= \left|\frac{1}{2}\left(\frac{x_k^2 + A}{x_k}\right) - \sqrt{A}\right| = \left|\frac{x_k^2 + A - 2x_k\sqrt{A}}{x_k}\right| = \left|\frac{(x_k - \sqrt{A})^2}{2x_k}\right| = \frac{e_k^2}{2x_k}$$

$$\frac{e_{k+1}}{e_k^2} = \frac{e_k^2}{2x_k}\,\frac{1}{e_k^2} = \frac{1}{2x_k}$$

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k^2} = \lim_{k\to\infty} \frac{1}{2x_k} = \frac{1}{2}\lim_{k\to\infty} \frac{1}{x_k} = \frac{1}{2\sqrt{A}} \qquad \text{CUANDO } k\to\infty,\ x_k \to \sqrt{A}$$

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k^2} = \lim_{k\to\infty} \frac{1}{2x_k} = \frac{1}{2}\lim_{k\to\infty} \frac{1}{x_k} = \frac{1}{2\sqrt{A}} \qquad \text{CUANDO } k\to\infty,\ x_k \to \sqrt{A}$$

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k^2} = \frac{1}{2\sqrt{A}} \qquad r = 2 \qquad C = 1/(2\sqrt{A})$$

La sucesión en cuestión cuenta con una tasa de convergencia cuadrática (r = 2) con una constante de tasa dependiente de del valor de "a" (C = $1/(2\sqrt{a})$)

3. Considere el problema de optimización:

$$\min_{x\in\mathbb{R}} f(x)$$

en donde, $f : \mathbb{R} \to \mathbb{R}_\ni \ f(x) = 4x^4 - 4x + 1$.

a) Resuelva este problema de optimización utilizando el *método de Newton-Raphson* que implementó en el Laboratorio 2. Despliegue una tabla que muestre claramente las distintas iteraciones que realizó.

b) ¿Es *global* o *local* el mínimo que encontró? Justifique su respuesta.

c) Estime la *tasa de convergencia* r y la constante C utilizando una *regresión lineal*. Para ello, recuerde que definimos la tasa de convergencia mediante el siguiente límite:

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k^r} = C.$$

Asumiendo un comportamiento "ideal", podemos escribir:

$$e_{k+1} = C\, e_k^r \quad \text{para todo } k.$$

Al aplicar logaritmo natural a ambos miembros de la ecuación anterior, obtenemos:

$$\underbrace{\ln|e_{k+1}|}_{y} = r\ \underbrace{\ln|e_k|}_{x} + \ln C.$$

Por tanto, dada una "sucesión" de errores $\{e_k\}$ (generada por su programa), podemos utilizar una regresión lineal de la forma $y = \alpha x + \beta$ para estimar r y C.

d) Repita *todos* los incisos anteriores pero ahora utilizando el *método de la Bisección*.

e) Finalmente, realice un análisis comparativo entre los dos métodos utilizados para resolver este problema. Auxíliese de gráficos y del análisis empírico de las tasas de convergencia para justificar su respuesta.

A.

Iter	Xk	Error
1.00000000	0.75000000	2.75000000
2.00000000	0.64814815	0.35655134
3.00000000	0.63046611	0.00963857
4.00000000	0.62996093	0.00000772

B. $f(x) = 4x^4 - 4x + 1$

$f'(x) = 16x^3 - 4$

$f''(x) = 48x^2$

Le segunda derivada es positiva semi-definida para todo el dominio del problema de optimización, lo que implica que la función es convexa. Dado que las restricciones también son convexas (indicando que se trata de un problema convexo) y que se encontró un mínimo local a través del uso del método de Newton-Raphson, se puede llegar a concluir que el mínimo encontrado consiste de un minimizador global.

Theorem 3.1

Any local minimum x^* of the convex problem (1) is also a global minimizer.

C.

```
from sklearn.linear_model import LinearRegression

# La columna de "Error" de la tabla se extrae en la forma de un array de numpy
error_newton = df_Newton["Error"].to_numpy()

# Se extraen dos arrays:
# - ek: Valores que van desde el índice 0 hasta el penúltimo índice (Time step k)
# - ek1: Valores que van desde el índice 1 hasta el último índice (Time step k+1)
ek = error_newton[0:-1]
ek1 = error_newton[1:len(error_newton)]

# Se define la "X" y "Y" del problema de regresión
X = np.log(ek)
Y = np.log(ek1)

# Se redimensionan los arrays para ser bidimensionales
X = np.reshape(X, (-1, 1))
Y = np.reshape(Y, (-1, 1))

# Se hace la regresión lineal
reg = LinearRegression().fit(X,Y)

# Se imprimen los resultados
print(f"R2 = {reg.score(X,Y)} | M (r) = {reg.coef_[0,0]} | B (ln(C)) = {reg.intercept_[0]}")

# Se imprime la tasa de convergencia del algoritmo y el coeficiente
print(f"r = {reg.coef_[0,0]} | C = {np.exp(reg.intercept_[0])}")
```

✓ 0.5s

```
R2 = 0.9992056006164993 | M (r) = 1.9083427423351704 | B (ln(C)) = -2.84958750044381
r = 1.9083427423351704 | C = 0.05786818655347996
```

De acuerdo a los cálculos realizados utilizando los datos presentados arriba (4 iteraciones), el método de Newton iniciando con una solución de x0 = 2 y con una precisión de h = 0.0001, tiene una tasa de convergencia casi cuadrática (r = 1.91) y una constante casi nula (C = 0.058)

A pesar de esto, cabe mencionar que si la estimación inicial se aleja de la solución real (por ejemplo, si se coloca x0 = 0), entonces el método comienza a tornarse super-lineal (r = 1 y C = 0), como puede verse en los resultados presentados a continuación (donde al algoritmo le tomó 59 iteraciones converger):

```
R2 = 0.9977729747346459 | M (r) = 1.0186335777977338 | B (ln(C)) = -2.0594857294119677
r = 1.0186335777977338 | C = 0.12751953257990092
```

D.

Iter	Xk	Error
1.00000000	0.75000000	2.75000000
2.00000000	0.62500000	0.09375000
3.00000000	0.68750000	1.19921875
4.00000000	0.65625000	0.52197266
5.00000000	0.64062500	0.20660400
6.00000000	0.63281250	0.05457306
7.00000000	0.62890625	0.02004910
8.00000000	0.63085938	0.01714647
9.00000000	0.62988281	0.00148015
10.00000000	0.63037109	0.00782595
11.00000000	0.63012695	0.00317110
12.00000000	0.63000488	0.00084502

```
# La columna de "Error" de la tabla se extrae en la forma de un array de numpy
error_biseccion = df_Biseccion["Error"].to_numpy()

# Se extraen dos arrays:
# - ek: Valores que van desde el índice 0 hasta el penúltimo índice (Time step k)
# - ek1: Valores que van desde el índice 1 hasta el último índice (Time step k+1)
ek = error_biseccion[0:-1]
ek1 = error_biseccion[1:len(error_biseccion)]

# Se define la "X" y "Y" del problema de regresión
X = np.log(ek)
Y = np.log(ek1)

# Se redimensionan los arrays para ser bidimensionales
X = np.reshape(X, (-1, 1))
Y = np.reshape(Y, (-1, 1))

# Se hace la regresión lineal
reg = LinearRegression().fit(X,Y)

# Se imprimen los resultados
print(f"R2 = {reg.score(X,Y)} | M (r) = {reg.coef_[0,0]} | B (ln(C)) = {reg.intercept_[0]}")

# Se imprime la tasa de convergencia del algoritmo y el coeficiente
```

11.00000000	0.63012695	0.00317110
12.00000000	0.63000488	0.00084502
13.00000000	0.62994385	0.00031767
14.00000000	0.62997437	0.00026365
15.00000000	0.62995911	0.00002702

```
# Se imprimen los resultados
print(f"R2 = {reg.score(X,Y)} | M (r) = {reg.coef_[0,0]} | B (ln(C)) = {reg.intercept_[0]}")

# Se imprime la tasa de convergencia del algoritmo y el coeficiente
print(f"r = {reg.coef_[0,0]} | C = {np.exp(reg.intercept_[0])}")
```

✓ 0.5s

```
R2 = 0.8021573516651372 | M (r) = 0.9615084460280845 | B (ln(C)) = -0.8862520037203092
r = 0.9615084460280845 | C = 0.4121977769312091
```

Al repetir la resolución de la ecuación pero ahora utilizando el método de Bisección (utilizando un rango de -1 a 1 y la misma precisión de h = 0.0001 que en el método de Newton) se llegó a determinar que este algoritmo contaba con una tasa de convergencia casi lineal (r = 0.96) y una constante que indica una tasa de convergencia intermedia entre una tasa sublineal y superlineal (C = 0.41).

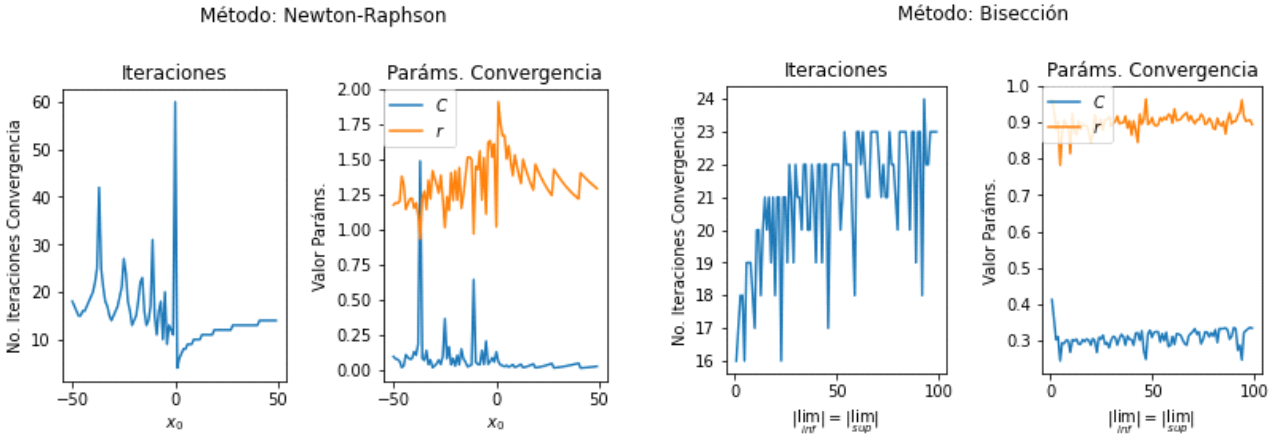
Curiosamente, este algoritmo parece ser mucho más consistente que el método de Newton-Raphson. Al expandir la región de búsqueda entre -100 y 100 (manteniendo todas las demás cantidades intactas), el número de iteraciones creció únicamente a 22 y el algoritmo mantuvo una tasa casi lineal (r = 0.92) y una constante un poco menor a la que tenía en el ejemplo anterior (C = 0.31).

```
R2 = 0.9686790791504611 | M (r) = 0.9185658367641278 | B (ln(C)) = -1.1592840884620477
r = 0.9185658367641278 | C = 0.3137106896110934
```

E. Al observar los resultados obtenidos, parece que el método de Newton-Raphson, bajo condiciones óptimas, consiste del método más rápido a comparación del método de Bisección. En un escenario ideal, el método de Newton cuenta con una tasa de convergencia casi cuadrática, mientras que el método de Bisección cuenta con una tasa lineal (no super o sub lineal), lo que indica que el método de Newton es mucho más rápido.

A pesar de esto, al variar las condiciones iniciales de cada algoritmo, se llegó a determinar que el método de Newton es mucho más sensible a cambios en dichas condiciones que el método de Bisección. Como se observó previamente, el método de Bisección presenta consistentemente una tasa de convergencia lineal, mientras que el método de Newton, a medida que x0 tiende a 0, comienza a adoptar una tasa super-lineal.

Al realizar un análisis un poco más extenso de lo previamente establecido (analizando muchas más condiciones iniciales), se pudo llegar a concluir que efectivamente el método de Newton presenta mucha más "volatilidad" en las tasas de convergencia y constantes que adopta, mientras que el método de bisección permanece mucho más estable. Otra observación importante es que el número de iteraciones para converger del método de Bisección parecen tender a un valor constante, mientras que las del método de Newton varían en gran medida en la sección "negativa" de las condiciones iniciales.



Sin embargo, aunque esto parezca una ventaja clara del método de Bisección por sobre el método de Newton, es importante mencionar que el método de Bisección cuenta con una desventaja muy importante: Si el intervalo de búsqueda no contiene la solución del problema, el algoritmo está destinado a divergir. El método de Newton por otro lado, aunque se torne super-lineal, llega eventualmente a la solución (siempre y cuando el problema sea convexo).

4. (*Problema Extra* – 25 puntos) Sea $f : \mathbb{R} \to \mathbb{R}$ una función con primera y segunda derivadas continuas. Sea $x^\star$ una solución de la ecuación no lineal $f(x) = 0$ con $f'(x^\star) \neq 0$. Demuestre que, si $|x_0 - x^\star|$ es suficientemente pequeña, entonces la sucesión $\{x_k\}_{k=0}^{\infty}$ producida por el *método de Newton* tiene una *tasa de convergencia cuadrática* con constante $C = \left|\dfrac{f''(x^\star)}{2f'(x^\star)}\right|$. Asuma la convergencia de algoritmo, es decir que $\{x_k\}_{k=0}^{\infty}$ converge a $x^\star$.

$|x_0 - x^*| = \beta$ DONDE $\beta > 0$ y $\beta \ll 1$

ENTONCES

$$f(x^*) = f(x_k + \beta) = f(x_k) + f'(x_k)(x^* - x_k) + \frac{1}{2}(x^* - x_k)^2 f''(x_k) = 0 \quad (1)$$

$$\beta = x_k - x^*$$

METODO DE NEWTON

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \longrightarrow f(x_k) = f'(x_k)(x_k - x_{k+1}) \quad (2)$$

SUSTITUYENDO (2) EN (1)

$$f'(x_k)(x_k - x_{k+1}) + f'(x_k)(x^* - x_k) + \frac{1}{2}(x^* - x_k)^2 f''(x_k) = 0$$

$$f'(x_k)(x_k - x_{k+1} + x^* - x_k) + \frac{1}{2}(x^* - x_k)^2 f''(x_k) = 0$$

$$f'(x_k)(x^* - x_{k+1}) + \frac{1}{2}(x^* - x_k)^2 f''(x_k) = 0$$

$A_{k+1} = x^* - x_{k+1}$

$A_k = x^* - x_k$

$$f'(x_k) A_{k+1} + \frac{1}{2} A_k^2 f''(x_k) = 0$$

$$f'(x_k) A_{k+1} = -\frac{1}{2} A_k^2 f''(x_k)$$

$$\frac{A_{k+1}}{A_k^2} = -\frac{f''(x_k)}{2f'(x_k)} \longrightarrow \left|\frac{A_{k+1}}{A_k^2}\right| = \left|-\frac{f''(x_k)}{2f'(x_k)}\right|$$

$e_{k+1} = |A_{k+1}| = |x_{k+1} - x^*|$

$e_k = |A_k| = |x_k - x^*|$

EL ORDEN DE LA RESTA ES IRRELEVANTE

$$\frac{e_{k+1}}{e_k^2} =$$

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k^2} = \left|\frac{f''(x^*)}{2f'(x^*)}\right| \qquad \lim_{k\to\infty} x_k = x^*$$

TASA CONVERGENCIA: $r = 2$

CONSTANTE: $C = \left|\dfrac{f''(x^*)}{2f'(x^*)}\right|$