Reto 1 Análisis Numérico

José Calderón Santiago Fernández Mariana Galavís
Germán Velasco

Marzo 14 2021

1. Algoritmo de Brent

Es un algoritmo de búsqueda de raíz que combina el método de bisección, el método de la secante y la interpolación cuadrática inversa. Este método es una alteración del método Dekker, desarrollada por Richard Brent.
Suponiendo que se desea resolver la función $F = 0$, por el método de bisección obtenemos dos valores, a_0 y b_0, de tal forma que $F(a_0)$ y $F(b_0)$ tienen signos distintos.
Si F es continua en los valores intermedio del teorema de valor intermedio se indica que existe una solución en el intervalo $[a_0, b_0]$.
Cada iteración consta de tres elementos, b, es la iteración actual, es decir, la aproximación actual de la raíz de la función, a, es el contra punto, es decir, donde se cumple que $F(a)$ tiene signo opuesto a $F(b)$ y b anterior, es el valor aproximado de la raíz pero en la iteración anterior. Luego de esto se elige un nuevo valor contra punto, si $F(a)$ y $F(banterior)$ tienen signos opuestos se escoge el nuevo contra punto igual a a, de lo contrario el nuevo contra punto es igual a b. Para usar este método en la resolución del ejercicio 1 del reto, hicimos uso de la biblioteca de SciPy. Esta incluye módulos para la optimización, álgebra lineal, integración, interpolación entre otros [3]. En especial, el módulo de optimize nos proporciona funciones que nos ayudan a resolver la búsqueda de raíces, problema que nos concierne en este punto de la actividad. El proceso que se sigue para su uso es:

1. Importar la librería de acuerdo con la sentencia $from\ scipy\ import\ optimize$.
2. Hacer el llamado a la función *brentq* del módulo *optimize* de acuerdo a la sentencia (con valores por defecto) $optimize.brentq(f, a, b, args = (), xtol = 2e^{-12}, rtol = 8{,}881784197001252e{-}16, maxiter = 100, full_output = False, disp = True)$

Teniendo en cuenta que Brent integra el método de bisección y el método de la secante, las entradas f, a y b son de carácter obligatorio. Entonces, de acuerdo con la condición de tolerancia dada, se implementó la función adaptada al ejercicio de la siguiente manera:

$$optimize.brentq(f, a, b, xtol = 2 * *(-50), full_output = 1)$$

Las entradas corresponden a:

- f [función]: esta debe ser continua y $f(a)$ y $f(b)$ deben tener signos opuestos.
- a [limite inferior]: número que representa el inicio del intervalo donde se encuentra la raíz.
- b [limite superior]: número que representa el fin del intervalo donde se encuentra la raíz.
- xtol [tolerancia]: número positivo que representa el valor máximo permitido para la diferencia entre una aproximación a la solución y el valor anterior. Este valor no puede ser menor al valor $4 * (2^{-52})$.
- full output[retorno valores]: retorna solo la raíz si su entrada es igual a 0 o False ó la raíz y un objeto que muestra valores entre los cuales se encuentran las iteraciones si su entrada es igual a 1 o True.

Las salidas que se esperan del método por la manera en la que este se empleó son dos: la aproximación a la raíz resultante del ejecutar el método y el número de iteraciones. Es importante mencionar que el criterio empleado para escoger los parámetros a y b por el grupo, se basó en la observación de la gráfica del polinomio en Wolfram Alpha.
A continuación se presentan los resultados obtenidos:

Cuadro 1: Resultados del método de Brent y Newton en diferentes intervalos

Intervalo	Método de Brent	# de iteraciones	X0	Método de Newton	# de iteraciones
(0,2)	0,666664981	54	0	0,666665114	44
(0,1)	0,666668671	51	0	0,666665114	44
(0.2,1)	0,666664986	50	0.5	0,666662389	38
(0.4,0.8)	0,666670183	46	0.5	0,666662389	38
(0.58,0.68)	0,666670589	40	0.58	0,66666346	36

Cuadro 2: Errores de calculo respecto a los errores y a la herramienta Wolfram Alpha

Diferencia entre Brent y Newton	Error de Brent	Error de Newton
1325097225	16856354083	15531256857
35574041000	2004278414	15531256857
25968047379	1680777801	4277582539
779422259,2	35166400533	4277582539
7129606141	39227069725	3206899168

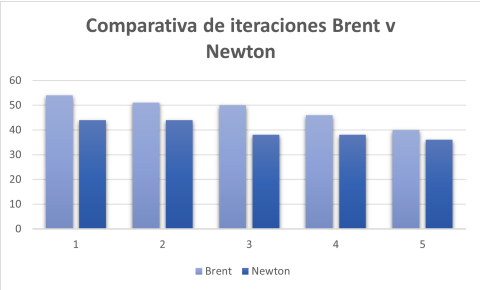

Figura 1: Comparativa de iteraciones entre los dos métodos

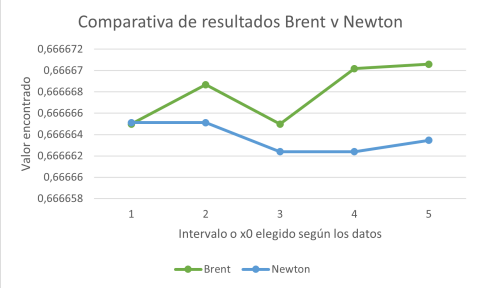

Figura 2: Comparativa de resultados entre los dos métodos

De los resultados presentados anteriormente y su respectiva representación gráfica, se evidencia que el método de Brent converge a unas solución más rápido a medida que se aproximan los valores a y b a x0 y que este último a su vez se aproxime al valor de la raíz. Sin embargo, y a pesar de esta rápida convergencia, al compararlo con Newton vemos que para este caso resulta más eficiente en cuanto a que tiene un número de iteraciones menor que el de Brent a medida que el x_0 se aproxima a la raíz. Respecto al resultado entregado por estos dos métodos y teniendo en cuenta que la solución con la herramienta Wolfram Alpha para el ejercicio con 15 cifras significativas es 0,6666666666666667, podemos apreciar que el método híbrido de Brent presenta menor exactitud que el de Newton.

2. Introducción a las ecuaciones no lineales

Se denomina ecuación a una igualdad matemática que existe entre dos expresiones algebraicas denominadas miembros en las que aparecen números y letras (llamadas incógnitas) relacionadas a través de operaciones matemáticas. Las ecuaciones pueden ser lineales o no lineales. Por un lado, las ecuaciones lineales o de primer grado, son igualdades algebraicas con incógnitas cuyo exponente es 1 y su gráfica corresponde a una línea recta. Para solucionar un sistema de ecuaciones lineales, se tienen métodos directos e indirectos. Entre los métodos directos, cuya principal característica es la eliminación de las incógnitas que aparecen en el sistema, tenemos: método de Gauss, método de Gauss-Jordan, la regla de Cramer, entre otros. Respecto a los métodos indirectos, se presentan los métodos iterativos de Jacobi y el método de Gauss-seidel. Por otro lado, una ecuación no lineal, es una igualdad que tiene al menos una incógnita con exponente mayor a 1. Las ecuaciones no lineales más sencillas son las polinómicas, $P_n(x) = 0$ con $n \geq 2$.
En general, este tipo de ecuaciones no se puede resolver con un número finito de operaciones por lo que se emplean métodos iterativos para su solución. Para su funcionamiento, estos necesitan uno o varios datos iniciales para realizar una sucesión de valores x que se espera converjan a un 0 de la función. Ejemplos de estos métodos iterativos son el método de Newton Raphson y el método del punto fijo [5].
Para solucionar un sistema de ecuaciones no lineales, es importante tener en cuenta que ahora se realizara el cálculo para un espacio de n dimensiones. La forma general de un sistema de ecuaciones no lineales es:

$$\begin{cases} f_1(x_1, x_2, x_3, \ldots, x_n) = 0 \\ f_2(x_1, x_2, x_3, \ldots, x_n) = 0 \\ f_3(x_1, x_2, x_3, \ldots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, x_3, \ldots, x_n) = 0 \end{cases} \tag{1}$$

$$F(x_1, x_2, x_3, \ldots, x_n) = [f_1(x_1, x_2, x_3, \ldots, x_n), f_2(x_1, x_2, x_3, \ldots, x_n), \ldots, f_n(x_1, x_2, x_3, \ldots, x_n)]$$

El sistema anterior se puede representar como $F(x) = 0$ y la solución correspondiente es el vector $X = [x_1, x_2, x_3, \ldots, x_n]$ que hace que todas las ecuaciones sean iguales a 0 al mismo tiempo.
Algunos conceptos para tener en cuenta para los sistemas de ecuaciones no lineales son:

- Vector gradiente: Es un vector con todas las derivadas parciales posibles de la función.
- Matriz Hessiana: Matriz de nxn que contiene todas las derivadas parciales de segundo orden en una matriz.
- Derivadas parciales: Derivada de una función con respecto a una variable manteniendo las otras como constantes.
- Matriz Jacobiana: Matriz formada por las derivadas parciales de primer oden de la función (tratada ampliamente en la sección 4).
- Norma Vectorial: Longitud o magnitud del vector en consideración.

3. Intersección entre curvas con el método de Newton

Inicialmente hablaremos de que es una intersección entre las curvas, hablando en términos geométricos es el punto, línea recta o curva en el cual dos curvas o las imágenes de superficies se entrelazan de maneja que convergen en un punto y una imagen, esto lo dice el matemático Erich Hartmann en su libro Geometry and Algorithms for COMPUTER AIDED DESIGN: "La determinación de la intersección de planos o rectas definidos en un espacio dimensional superior, es una tarea simple de álgebra lineal, es decir, la solución de un sistema de ecuaciones lineales. Pero en general, la determinación de una intersección conduce a sistemas no lineales. Los problemas de intersección entre una línea y una sección cónica o una conducen a ecuaciones de segundo grado que se pueden resolver fácilmente. Las intersecciones entre cuádricas llevan a ecuaciones cuárticas, que se pueden resolver algebraicamente."[4] En consecuencia, al autor, se recomienda usar métodos de análisis numérico, siendo uno de estos el método de Newton-Raphson ya que este es perfecto para encontrar las mejores aproximaciones a los ceros o raíces en funciones reales y además lograr encontrar los puntos críticos de la función con su primera derivada, como lo menciona en el documento de la universidad de granada "Métodos de resolución de ecuaciones no lineales", esto nos indica que el método trabajado en anteriores entregas es el más óptimo en términos de rendimiento para este punto crucial, ya que está enfocado en conseguir la respuesta en el menor número de iteraciones posibles.

Ahora hablando del problema se nos pide encontrar la interacción entre dos con dicho método, inicialmente se adaptará el algoritmo inicial para encontrar esos puntos de corte significativos con un error menor al 2^{-16} y tomando valores base de WolframAlpha y las gráficas a comparar con la herramienta Symbolab.
A continuación, se presentan las gráficas:

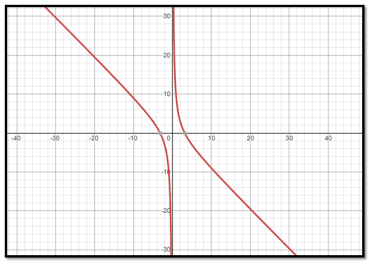

Figura 3: Gráfico de la curva $x^2 + xy = 10$

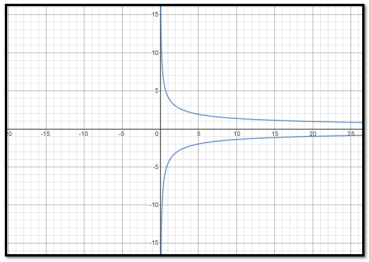

Figura 4: Gráfico de la curva $y + 3xy^2 = 57$

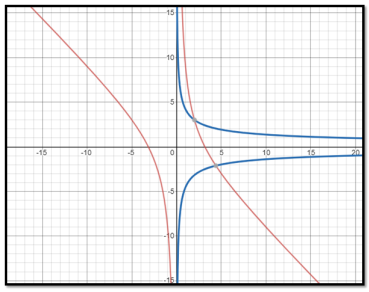

Figura 5: Gráfico que refleja intersección entre curvas

Cuadro 3: Resultados generales de la aplicación del método generalizado de Newton comparado con Wolfram Alpha

Punto (x, y) seleccionados	**Metodo Newton**	**WolframAlpha**	**Diferencia**	**Error porcentual**
(4, -4)	4,39374427	4,39374419	0,00000008	0,00000182
	-2,11778121	-2,11778101	0,00000020	0,00000944
(6,-6)	4,39374128	4,39374419	0,00000291	0,00006623
	-2,11777616	-2,11778101	0,00000485	0,00022901
(8,-8)	4,39374420	4,39374419	0,00000001	0,00000023
	-2,11778103	-2,11778101	0,00000002	0,00000094
(10, -10)	4,39374416	4,39374419	0,00000003	0,00000068
	-2,11778104	-2,11778101	0,00000003	0,00000142
(12, -12)	4,39374422	4,39374419	0,00000003	0,00000068
	-2,11778096	-2,11778101	0,00000005	0,00000236
(14, -14)	4,39374420	4,39374419	0,00000001	0,00000023
	-2,11778102	-2,11778101	0,00000001	0,00000047

A partir de los resultados mostrados en la tabla anterior, podemos concluir que el método de Newton resulta ser una buena opción para resolver un sistema de ecuaciones no lineales ya que al compararlo con los resultados de la herramienta WolframAlpha la diferencia entre el resultado del método y esta es mínima demostrando una alta exactitud. Cabe resaltar qué, estos resultados fueron obtenidos con funciones de librerías de Scipy, de las cuales hablaremos en las siguientes dos secciones.

4. Generalización del método de Newton para sistemas de ecuaciones no lineales

Este método es caracterizado por ser la generalización del método base de Newton-Raphson para encontrar raíces en funciones de una sola variable. Permite así tener la base para construir y solucionar sistemas más complejos en los cuales se involucren una o más variables en diferentes potencias, estos son los sistemas de ecuaciones no lineales.
De igual forma, cabe resaltar que este método funciona gracias al uso del Jacobiano inverso, sin embargo, definiremos primero el Jacobiano para posteriormente continuar el desarrollo del método [1].
El Jacobiano es definido como una matriz formada por las derivadas parciales de primer orden de una función determinada y es definida de la siguiente forma:

$$\mathbf{J}_{(x_1,\ldots,x_n)} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

Entonces, el Jacobiano inverso está definido de esta forma:

$$\mathbf{J}_{(\mathbf{x_1},\ldots,\mathbf{x_n})}{}^{-1} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}^{-1}$$

Con esto en mente entonces se puede definir la generalización del método de la siguiente manera:

Sea X el vector $(x_1, ..., x_n)$, $F(X)$ el sistema de ecuaciones y $J(X)$ el Jacobiano determinado para dicho sistema entonces la formula recursiva para el método de Newton resulta [1],

$$X_{k+1} = X_k - J^{-1}F(X_k)$$

La convergencia del método se asegura mediante una tolerancia determinada y la segunda norma de la matriz resultante de $J^{-1}F(X_k)$ tal que [1]:

$$\left\| J^{-1}F(X_k) \right\|_2 < tol$$

Para la implementación de este algoritmo en Python se utilizó la librería Scipy, específicamente el apartado de optimización y búsqueda de raíces (scipy.optimize), en particular la función scipy.optimize.newton-krylov [3]. Esta función recibe los siguientes parámetros:

- $F(X)$, sistema de ecuaciones a resolver.

- xin, puntos iniciales (de partida) a considerar en el sistema de ecuaciones.
- x_{tol}, máxima tolerancia soportada por el algoritmo.
- Opciones adicionales.

5. Método de Broyden

Este método, conocido por ser estar en la clasificación quasi-newton, está tomado en consideración debido a que es una modificación leve al método anteriormente expuesto, esto principalmente para evitar el cálculo iterativo del Jacobiano inverso. En vez de dicha matriz, utiliza una nueva que permite que sea fácilmente actualizada y operada durante cada iteración.Su definición como método es la siguiente:

Sea A la matriz de aproximación del Jacobiano, se define el método tal que [2]:

$$X_{k+1} = X_k - A^{-1}F(X_k)$$

Sea P la solución real del sistema de ecuaciones, la convergencia del método es superlineal y se asegura mediante una tolerancia determinada, tal que [2]:

$$\lim_{x\to\infty} \frac{\left\|x_{i+1} - p\right\|}{\left\|x_i - p\right\|}$$

Para la implementación de este algoritmo en Python se utilizó la librería Scipy, específicamente el apartado de optimización y búsqueda de raíces (scipy.optimize), en particular la función scipy.optimize.broyden1 y scipy.optimize.broyden2, estos últimos números siendo el número de veces en que la matriz operada se aproxima al Jacobiano [3]. Estas funciones reciben los siguientes parámetros:

- $F(X)$, sistema de ecuaciones a resolver.
- xin, puntos iniciales (de partida) a considerar en el sistema de ecuaciones.
- x_{tol}, máxima tolerancia soportada por el algoritmo.
- Opciones adicionales.

6. Conclusiones

Teniendo en cuenta el desarrollo e investigación respecto a las temáticas tratadas en el reto 1 de Análisis Numérico, se llegaron a las siguientes conclusiones:

- El conocimiento y estudio de las librerías de aplicación de métodos numéricos como Scipy permite que la resolución de problemas sea más rápida y eficiente, teniendo herramientas como estas, el trabajo se realiza de forma más confiable.

- La lectura de código para comprender los procesos internos que lleva el algoritmo implementado es importante siempre que se quiera entender el método usado a profundidad.
- Siempre que se proceda a evaluar un método numérico tales como los mencionados en las secciones anteriores debe ser verificado por una herramienta matemática confiable, tal es el ejemplo de WolframAlpha, mecanismo por el cual aseguramos que la solución propuesta es óptima.
- De acuerdo con el desarrollo anterior, se concluye, son métodos muy efectivos a la hora de resolver sistemas de ecuaciones no lineales, dando resultados muy precisos de acuerdo con la solución real como se muestran en las tablas posteriores.

Cuadro 4: Evaluación de puntos en los dos métodos evaluados (Newton y Broyden)

Punto (x,y)	Resultado Newton (x)	Resultado Broyden (x)	Resultado Newton (y)	Resultado Broyden (y)
(4, -4)	4,39374419	4,39374419	-2,11778101	-2,11778102
(6, -6)	4,39374420	4,39374419	-2,11778101	-2,11778101
(2, 2)	2,00000000	1,99999997	3,00000000	3,00000002
(5, 5)	2,00000000	3,00000000	2,00000000	3,00000000

Cuadro 5: Evaluación de diferencias en los dos métodos evaluados (Newton y Broyden) comparados con WolframAlpha

Diferencia Wolfram N (x)	Diferencia Wolfram N (y)	Diferencia Wolfram B (x)	Diferencia Wolfram B (y)
0,000000003288591	0,000000004714180	0,000000003288591	0,000000005285820
0,000000006711410	0,000000004714180	0,000000003288591	0,000000004714180
0,000000000000000	0,000000000000000	0,000000030000000	0,000000020000000
0,000000000000000	1,000000000000000	1,000000000000000	0,000000000000000

Referencias

[1] C. Remani, "Numerical Methods for Solving Systems of Nonlinear Equations," p. 38.

[2] Y. Wang, "METHODS FOR SOLVING NONLINEAR EQUATIONS," p. 4.

[3] "Optimization and root finding (scipy.optimize) — SciPy v1.6.1 Reference Guide." [Online]. Available: https://docs.scipy.org/doc/scipy/reference/optimize.html

[4] E. Hartmann, "Geometry and Algorithms for," p. 160.

[5] R. L. Burden, "Numerical Analysis," *Numerical Analysis*, p. 895, 2010.