Métodos de análisis numérico

Fabián Pallares
fpallares@javeriana.edu.co
GitHub: FabianPallaresJ

Santiago Jaramillo
l_jaramillo@javeriana.edu.co
GitHub: l_jara20

Agosto de 2019

1 Método de bisección

Este método consiste en obtener una mejor aproximación de la raíz a partir de un intervalo inicial (a,b) en el cual hay un cambio de signo en la función, es decir:

$$f(a) * f(b) < 0$$

. [3]

Se obtiene el punto medio:

$$c = \frac{a+b}{2}$$

Así, se determina un nuevo un intervalo teniendo en cuenta la funcion evaluada en c. Para esto, verificamos dónde se encuentra el resultado cero multiplicando uno de los extremos evaluado con el c evaluado. Si el valor es negativo, significa que ahí se encuentra la raíz, por lo cual se vuelve a ejecutar la división para un nuevo valor de c.

Este proceso es realizado hasta que el valor de b - a sea menor al error (tolerancia) ingresado por el usuario.

Algoritmo:

Listing 1: Algoritmo de bisección

```
def biseccion(f, a, b, e):
    count = 0
    c = 0
    while b - a >= e:
        count += 1
        c = (a + b) / 2
        if f(c) == 0:
            return c
        else:
            if f(a)  * f(c) > 0:
                a = c
            else:
```

```
            b = c
    print("Iteraciones: " + str(count))
    return c
```

2 Tri-sección

Para este ejercicio, se quiere implementar una modificación al algoritmo de bisección que permita reducir el número d eiteraciones realizando una busqueda dividiendo en tres partes el intervalo definido.

Para esto, hacemos las verificaciones correspondientes, pero ahora con los nuevos intervalos (c y d) determinando en dónde se encuentra la raíz de la función.

Al ser encontrada la ubicación de la raíz, se procede a hacer nuevamente las dos divisiones hasta que de nuevo se cumpla el criterio que nos da el error ingresado por el usuario

$$b - a < e$$

Nota: Para ambos métodos se hacen también las comprobaciones de la función evaluada en c para determinar si es cero, en caso de serlo, se retorna el valor encontrado para c.

Algoritmo:

Listing 2: Algoritmo de tri-sección

```
def triseccion(f, a, b, e):
    count = 0
    d = 0
    c = 0
    while b - a >= e:
        count += 1
        c = a + ((b - a) / 3)
        d = c + ((b - a) / 3)
        if f(c) == 0:
            return c

        elif f(d) == 0:
            return d

        else:
            if f(a) * f(c) > 0:

                if f(c) * f(d) > 0:
                    a = d
                else:
                    a = c
                    b = d
            else:
                b = c
    print("El numero de iteraciones es: " + str(count))
    return d
```

3 Método de Newton

Este método es uno de los mas utilizados para localizar raíces ya que en general es muy eficiente y siempre converge para una función polinomial.

Se requiere que las funciones sean diferenciables, y por tanto, continuas, para poder aplicar este método.

Se debe partir de un valor inicial para la raíz: xi , este puede ser cualquier valor, el método convergirá a la raíz mas cercana.

Si se extiende una tangente desde el punto b , el punto donde esta tangente cruza al eje x representa una aproximación mejorada de la raíz.

Para encontrar el valor de x (el valor aproximado para la raíz), se toma una ecuación que obtenemos desde la fórmula de la pendiente de una recta. Resultando finalmente así:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

El método de Newton-Raphson es convergente cuadráticamente, es decir, el error es aproximadamente al cuadrado del error anterior.

Esto significa que el numero de cifras decimales correctas se duplica aproximadamente en cada interacción.

4 Método del punto fijo

Un punto fijo de una función g, es un número p tal que $g(p) = p$. El problema de encontrar las soluciones de una ecuación $f(x) = 0$ y el de encontrar los puntos fijos de una función $h(x)$ son equivalentes en el siguiente sentido: dado el problema de encontar las soluciones de una ecuación $f(x) = 0$, podemos definir una función g con un punto fijo p de muchas formas; por ejemplo, $f(x) = x - g(x)$. En forma inversa, si la función g tiene un punto fijo en p, entonces la función definida por $f(x) = x - g(x)$ posee un cero en p. [2]

El método de punto fijo inicia con una aproximación inicial x_0 y $x_{i+1} = g(x_i)$ genera una sucesión de aproximaciones la cual converge a la solución de la ecuación $f(x) = 0$. A la función g se le conoce como función iteradora. Se puede demostrar que dicha sucesión $\langle x_n \rangle$ converge siempre y cuando $|g'(x)| < 1$.

5 Método de la secante

El principal inconveniente del método de Newton estriba en que requiere conocer el valor de la primera derivada de la función en el punto. Sin embargo, la forma funcional de f(x) dificulta en ocasiones el cálculo de la derivada. En estos casos es más útil emplear el método de la secante. El método de la secante parte de dos puntos (y no sólo uno como el método de Newton) y estima la tangente (es decir, la pendiente de la recta) por una aproximación que, al reemplazar en la ecuación de Newton, nos resulta:

$$x_2 = x_0 - \frac{f(x_i) - x_0}{f(x_1) - f(x_0)}$$

Lo cual podemos represntar con el siguiente gráfico[3]:

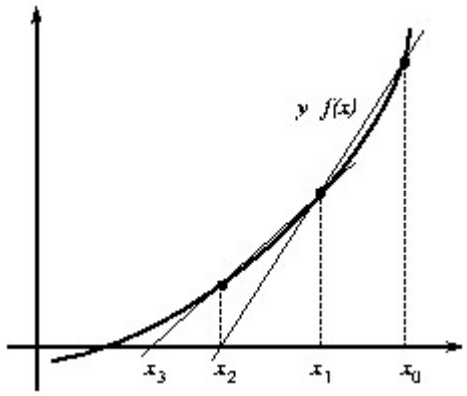

Figure 1: Representación del método de la secante

El algoritmo utilizado para este método fue el siguiente:

Listing 3: Algoritmo de secante

```
def secante(f, a, b, e):
    count = 0
    while abs(b - a) >= e:
        count += 1
        aux = b
        aux1 = (b - (f(b) * (a - b)) / (f(a) - f(b)))
        a = aux
        b = aux1
    print("Iteraciones: " + str(count))
    return b
```

6 Método de posición falsa

El método de la falsa posición pretende conjugar la seguridad del método de la bisección con la rapidez del método de la secante.[1] Este método, como en el método de la bisección, parte de dos puntos que rodean a la raíz f(x) = 0, es decir, dos puntos x0 y x1 tales que

$$f(x0) * f(x1) < 0$$

. La siguiente aproximación, x2, se calcula como la intersección con el eje X de la recta que une ambos puntos (empleando la ecuación (35) del método de

la secante). La asignación del nuevo intervalo de búsqueda se realiza como en el método de la bisección: entre ambos intervalos, [x0, x2] y [x2, x1], se toma aquel que cumpla

$$f(x) * f(x2) < 0$$

. En la figura se representa geométricamente este método.

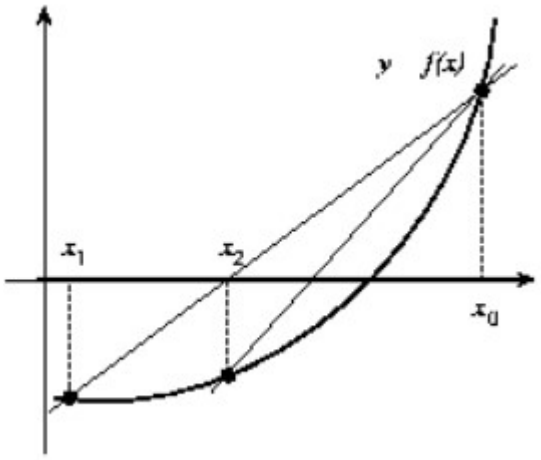

Figure 2: Método iterativo de falsa posición

Algoritmo:

Listing 4: Algoritmo de Posición falsa

```
def p_falsa(f, a, b, e):
    count = 0
    c = 0
    while b - a >= e:
        count += 1
        c = b - ((f(b) * (a - b))/(f(a) - f(b)))
        if f(c) == 0:
            return c
        else:
            if f(a)  * f(c) > 0:
                a = c
            else:
                b = c
    print("Iteraciones: " + str(count))
    return c
```

References

[1] Universidad de San Buenaventura. *Métodos numéricos*. Universidad de San Buenaventura, 2016.

[2] Francisco Palacios. *Resolución aprocimada de ecuaciones*. Universidad Politécnica de Cataluña, 2009.

[3] Wladimiro Diaz Villanueva. *Cálculo de raíces de ecuaciones*. Universitat de València, 1998.