Análisis Método del punto fijo

Juan José Bolaños, David Andres Duarte, David Saavedra

9 de marzo del 2021

1 Algoritmo de brent

El método de Brent es un algoritmo propuesto por Theodorus Dekker, sin embargo, Richard Brent propuso una modificación al algoritmo de Dekker ya que este en ciertas circunstancias convergía lentamente. El método combina la interpolación cuadrática inversa y los métodos de bisección y de la secante; estos dos últimos para generar el siguiente intervalo de interpolación.
Problema: Aplicar el algoritmo de Brent para encontrar las raíces del polinomio, con un error menor de 2^{-50}:

$$f(x) = x^3 - 2x^2 + 4x/3 - 8/27 \tag{1}$$

Solución

Para dar solución Brent propuso una modificación inserta una prueba adicional que debe cumplirse antes de que se acepte el resultado del método de la secante como la siguiente iteración. Dos desigualdades deben ser satisfechas simultáneamente dada una tolerancia numérica δ, si el paso anterior uso bisección la desigualdad seria:

$$|\delta| < |b_k - b_{k-1}| \tag{2}$$

Debe sostenerse para realizar la interpolación, de lo contrario se realizará el método de bisección y se usa el resultado para la siguiente iteración, siendo así la desigualdad es:

$$|\delta| < |b_{k-1} - b_{k-2}| \tag{3}$$

Para realizar la siguiente acción para elegir interpolación cuando la desigualdad es verdadera o bisección cuando la desigualdad es falsa, además si el paso anterior utilizo bisección, la desigualdad da:

$$|\delta - b_k| < \frac{1}{2}|b_k - b_{k-1}| \tag{4}$$

Debe mantenerse, de lo contrario se realiza el método de bisección y su resultado se usa para la siguiente iteración. Si el paso anterior realizo la interpolación, entonces la desigualdad

$$|\delta - b_k| < \frac{1}{2}|b_{k-1} - b_{k-2}| \tag{5}$$

se utiliza en su lugar.

```
brent = function(f, x0, x1, n, tol)
{
  fx0 = f(x0)
  fx1 = f(x1)
  if (abs(fx0) < abs(fx1)){
    aux1 = x0
    x0 = x1
    x1 = aux1

    aux2 = fx0
    fx0 = fx1
    fx1 = aux2
  }
  x2 = x0
  fx2 = fx0
  bandera = TRUE
  i = 0
  d = 0

  while (i < n & abs(x1 - x0) > tol){
    fx0 = f(x0)
    fx1 = f(x1)
    fx2 = f(x2)

    if (fx0 != fx2 & fx1 != fx2){
      op1 = (x0 * fx1 * fx2) / ((fx0 - fx1) *  (fx0 - fx2))
      op2 = (x1 * fx0 * fx2) / ((fx1 - fx0) * (fx1 - fx2))
      op3 = (x2 * fx0 * fx1) / ((fx2 - fx0) * (fx2 - fx1))
      s = op1 + op2 + op3
    } else{
      s = x1 - ((fx1 * (x1 - x0)) / (fx1 - fx0))
    }

    if ((s < ((3 * x0 + x1) / 4) | s > x1) |
        (bandera == TRUE & (abs(s - x1)) >= (abs(x1 - x2) / 2)) |
        (bandera == FALSE & (abs(s - x1)) >= (abs(x2 - d) / 2)) |
        (bandera == TRUE & (abs(x1 - x2)) < tol) |
        (bandera == FALSE & (abs(x2 - d)) < tol)){
      s = (x0 + x1) / 2
      bandera = TRUE
    } else{
      bandera = FALSE
    }
    fs = f(s)
    d = x2
    x2 = x1

    if ((fx0 * fs) < 0){
      x1 = s
    } else{
      x0 = s
    }

    if (abs(fx0) < abs(fx1)){
      aux3 = x0
      x0 = x1
```

```
57        x1 = aux3
58      }
59
60
61      i = i + 1
62    }
63    cat("Raiz: ", cantidadNumeros(x1, 20), "\n", "N mero de
          iteraciones: ", i)
64  }
```

```
Raiz:  0.66666030883789062 50
 Número de iteraciones:  20
```

Figure 1: Resultados Brent

2 Intersección entre curvas

En geometría, una intersección es un punto, curva, superficie, volumen o linea recta, que es común a dos o varios elementos. El caso mas simple que podemos encontrar en la geometría es la intersección de dos rectas distintas, que bine es un punto o no existe si estas son paralelas.
La determinación de una intersección de dos curvas en R^2 que son diferenciales y poseen un punto de intersección, si poseen un punto del plano en común y ese punto tiene:

- Intersección transversal
- Intersección tangente

Para este punto vamos aplicar la técnica de aproximación a la raız NEWTON-RAPHSON, que desarrollamos en el trabajo en grupo, con el fin de encontrar la intersección entre:

$x^2 + xy = 10$
$y + 3xy^2 = 57$

$$x^2 + xy = 10; y + 3xy^2 = 57 \tag{6}$$

Vamos a resolver este problema mediante la ecuacion de Newton-Raphson para así resolver de forma iterativa.para este caso vamos a multiplicar la ecuación de Newton-Raphson por la matriz Jacobiana.

1. Para lograr nuestra implementación en lenguaje de programación Python lo que vamos a necesitar es igualar a 0 cada una de nuestras ecuaciones.

$f1 = x^2 + xy - 10 = 0$

$f2 = y + 3xy^2 - 57 = 0$

2. Luego de esto necesitamos definir la derivada de F en una matriz que contiene las derivadas con respecto a x, y. de cada una de nuestras dos ecuaciones.

3. Ya que tenemos nuestra función y nuestra derivada, definimos el numero de iteraciones máximas que deseamos. Para pasar a realizar la interacción de nuestro Newton-Raphson.

4. Se calcula la Jacobiana inversa, para luego multiplicarla por F(x).

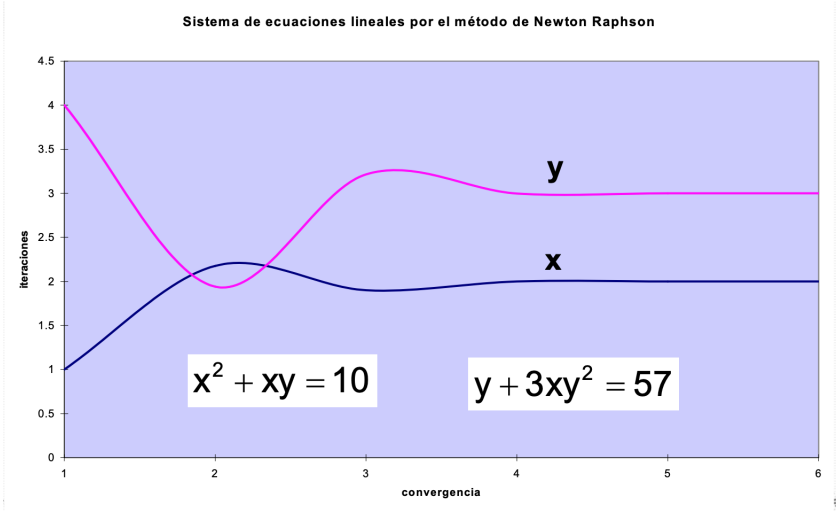

Figure 2: Iteraciones vs Convergencia por Newthon Raphson [2]

```
1           #PUNTO NUMERO 2 RETO1
2       import numpy
3
4       #Para esta implementacion x[0]= x; x[1]= y;
5
6       def F(x):
7
8         #Igualamos ambas ecuaciones a 0
9         #Funcion1: x^2 + xy - 10 = 0
10        funcion1= x[0]**2+(x[0])*(x[1])-10
11        #Funcion2: y + 3xy^2 -57 = 0
12        funcion2= x[1]+3*(x[0])*(x[1]**2)-57
13        #Nos devuelve la matriz
14        return numpy.array([funcion1,funcion2])
15
16      #Jacobiana de F
17      def JacobianaF(x):
```

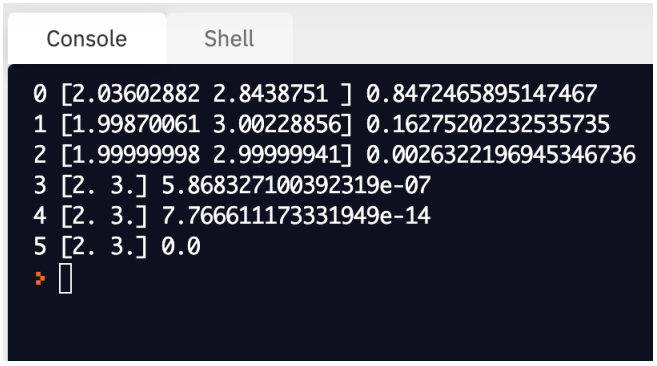

Figure 3: Resultado Implementación Punto 2 en Python

```
18
19          #Nos devuelve la matriz donde sus componentes estan
                dadas por derivadas con respecto x, y
20          return numpy.array([[2*x[0]+x[1],x[0]],[3*x[1]**2,1+2*x
                [1]*3*x[0]]])
21
22
23        numeroIteraciones = 100
24        x= numpy.array([1.5,3.5])
25
26        for j in range(numeroIteraciones):
27          cajon=x
28          #Sacar la Jacobiana Inversa
29          Jacobianainversa=numpy.linalg.inv(JacobianaF(x))
30          # x= JacobianaInversa x F
31          x = x -numpy.dot(Jacobianainversa,F(x))
32          #Sacar Error
33          e=numpy.linalg.norm(x-cajon)
34          #Imprimir numero Iteraciones - result - error
35          print(j,x,e)
36          #cumpla con el error (sea menor)
37          if e<1e-16:
38            break
```

3 Librerías en R y/o Python

3.1 Algoritmo de Brent

```
1   library(pracma)
2   f = function(x) x^3-2*x^2+(4*x/3)-(8/27)
3   #listaTolerancias(1e-10,1e-32,1e-50)
4   brent(f, 0.5, 1,maxiter=100, tol=2^-50)
```

root: Valor de la raíz
f.root: Valor de la función evaluada en la raíz

f.calls: Numero de iteraciones
estim.prec: Precisión estimada

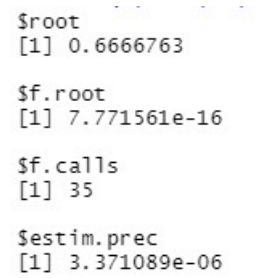

Figure 4: Resultados Brent con librerias

Librerias y funciones en R

Polyroot:
Es una función de R que se encarga de encontrar ceros en un polinomio real o complejo; se implementa de la forma polyroot(z) donde z es el vector de coeficientes polinomiales en orden creciente.
Retorna los n-1 ceros complejos del polinomio utilizando el algoritmo Jenkins-Traub. [3]

Uniroot:
Es una función que se encarga de encontrar la raíz de una ecuación dentro de un intervalo el cual se pasa por parámetro, muy similar a lo que hace el método de bisección y de Brent.
Se implementa de la forma uniroot(f, lower, upper) siendo f la función, y lower y upper los intervalos. [4]

Pracma:
Es una librería que proporciona una gran cantidad de funciones de análisis numérico, álgebra lineal, optimización numérica, ecuaciones diferenciales, series de tiempo, además de algunas funciones matemáticas especiales que son conocidas.
Es muy parecida a MATLAB, incluso manejan funciones que tienen el mismo nombre. [5]

Rootsolve:
Es una librería que proporciona funciones que sirven para encontrar raíces de funciones no lineales y para realizar diferentes análisis de equilibrio de ecuaciones diferenciales ordinarias y estados estacionarios.
Contiene funciones que:
- Generan gradientes y matrices Jacobianas.

- Encuentra raíces de ecuaciones no lineales principalmente mediante el método Newton Raphson.
- Estiman condiciones de un sistema.
- Entre muchas otras. [6]

Librerias y funciones en Python

Numpy:
Para el caso del problema de intersección de curvas utilizamos la libreria Numpy en Python.
Esta librería proporciona diferentes funciones que dan soporte a la creación de matrices y vectores multidimensionales y de gran tamaño, además de brindar diferentes funciones matemáticas para operar con ellas. Su funcionalidad es comparable con MATLAB. [7]

4 Referencias

[1] Zhang Z. An improvement of the Brent's Method. International Journal of Experimental Algorithms. 2, 21-26, 2011.
[2] Castro C. Metodos numericos.Departamento académico de ingeniería de minas y civil. 20,2, 2014
[3] R: Find Zeros of a Real or Complex Polynomial. (s. f.). R Documentation. Avaible: https://stat.ethz.ch/R-manual/R-patched/library/base/html/polyroot.html
[4] R: One Dimensional Root (Zero) Finding. (s. f.). R Documentation.Avaible: https://stat.ethz.ch/R-manual/R-patched/library/stats/html/uniroot.html
[5] pracma package — R Documentation. (s. f.). R Documentation. Avaible: https://www.rdocumentation.org/packages/pracma/versions/1.9.9
[6] rootSolve package — R Documentation. (s. f.). R Documentation. Avaible: https://www.rdocumentation.org/packages/rootSolve/versions/1.8.2.1
[7] NumPy. (s. f.). Wikipedia, la enciclopedia libre. Avaible: https://es.wikipedia.org/wiki/NumPyCaracter