Reto 1

Thomás Rivera
Nicolás Puerto
Juan Mejía

February 2021

1 Algoritmo de Brent

Este algoritmo de Brent, utiliza en cada punto lo más conveniente de las estrategias del de la bisección y del de la secante (o Muller). Este método suele converger muy rápidamente a cero; para las funciones difíciles ocasionales que se encuentran en la práctica.
Problema:Aplicar el algoritmo de Brent para encontrar las raíces del polinomio, con un error menor de 2^{-50}:

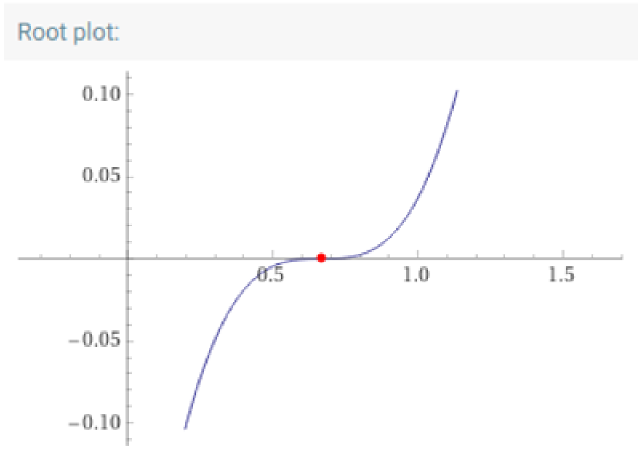
$$f(x) = x^3 - 2x^2 + 4x/3 - 8/27 \tag{1}$$

Figure 1: Codigo Brent

El algoritmo de Brent (Se llama así debido a Richard Brent y se basa en el trabajo de Theodorus Dekker) permite determinar raíces combinando los métodos de interpolación cuadrática inversa, secante y bisección. La idea es utilizar los 2 primeros dado que convergen de manera rápida, pero de ser necesario se usa el último debido a que es más robusto.
Se inserta una prueba adicional que debe ser satisfecha antes de que el resultado del método de la secante se acepte como la iteración siguiente.

Raíz en WolframAlpha

$$x^3 - 2x^2 + \frac{4x}{3} - \frac{8}{27} = 0$$



$x \approx 0.6666666666666666666666666666666666666666666666667$

Figure 2: grafica Brent

codigo en R

```r
#Metodo de Brent

f <- function(x){
  x^3-2*x^2+(4/3)*x-(8/27)
}

a = mpfr(-1,53) #-4.62962
b = mpfr(1,53) #0.03703
Error = 2^-50

#Intercambiar valores
if (abs(f(a)) < abs(f(b))){
  L = a
  a = b
  b = L
}

c = a
MFlag = 1
delta = Error
i = 0

while(abs(b-a) >= Error){
  i = i+1
  if(f(a) != f(c) && f(b) != f(c)){
    #Interpolacion Cuadratica Inversa
    s = ((a*f(b)*f(c))/((f(a)-f(b))*(f(a)-f(c)))) +
        ((b*f(a)*f(c))/((f(b)-f(a))*(f(b)-f(c)))) +
        ((c*f(a)*f(b))/((f(c)-f(a))*(f(c)-f(b))))
    #cat("ICI \n")
  }
  else{
    #Metodo de la Secante
    s = b-f(b)*((b-a)/(f(b)-f(a)))
    #cat("Secante \n")
 }
 if( (s<=(3*a+b)/4 | s>=b)
   | (MFlag == 1 && abs(s-b) >= abs(b-c)/2)
   | (MFlag == 0 && abs(s-b) >= abs(c-d)/2)
   | (MFlag == 1 && abs(b-c) < abs(delta))
   | (MFlag == 0 && abs(c-d) < abs(delta))
   ){
   #Metodo de la Biseccion
   s=(a+b)/2
   MFlag = 1
   #cat("Biseccion \n")
 }
 else{
   MFlag = 0
 }

 #Calcular f(s)
 d = c #(d se asigna por primera vez aca, no se tiene en la 1 iteracion)
 c = b
 if(f(a)*f(s) < 0){
   b = s
 }
 else{
   a = s
 }

 #Intercambiar valores
 if(abs(f(a)) < abs(f(b))){
   L = a
   a = b
   b = L
 }

 cat("iteracion =",i,"Raiz =")
 print(s,50)
}
```

Figure 3: codigo Brent

Resultados Para desarrollar el ejercicio se utilizaron números con precisión

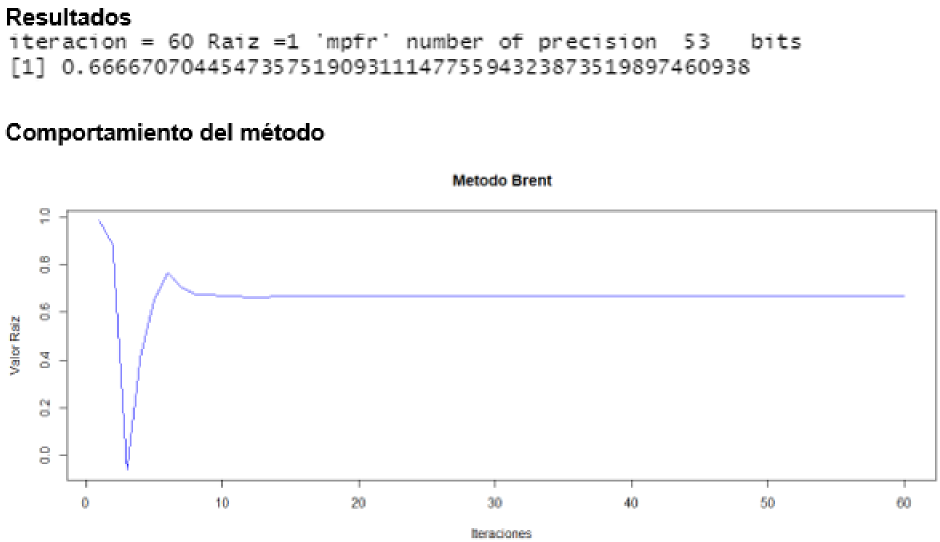


Figure 4: resultado Brent

de 53 bits y el algoritmo realiza 60 iteraciones, sin embargo a pesar de su complejidad no fue capaz de llegar al valor establecido por Wolfram Alpha. En cuanto a su comportamiento, inicia con el método de la interpolación cuadrática inversa y de la secante en las primeras instancias; a medida que avanza comienza a usar bisección para asegurar un mejor resultado. Llega a un punto en donde el cambio es mínimo y permanece casi de forma constante.

2 Intersección entre curvas

2. Intersección entre curvas

La intersección entre dos curvas es un problema comun del cálculo y que enfrenta el desafio de solucionar un sistema de ecuaciones no lineales. No obstante, la solución se puede encontrar reduciendo el problema a determinar una aproximación adecuada de los ceros de una ecuación, con un error menor de 2^{-16} .

Problema: Aplicar la técnica de aproximación a la raíz, que desarrollo en el trabajo en grupo, para encontrar la intersección entre

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

Figure 5: punto 2

El algoritmo que utilizamos para encontrar la interseccion entre las dos funciones es el algoritmo de newthon.Sin embargo, lo primero que utilizamos fue el despeje de las ecuaciones tienendo como referencia la variable x para que la Y quedara fuera de la ecuacion y asi no tener que utilizar el jacobiano dado que teniamos dos variables (x,y). A continuacion, la grafica de la funcion resultante (despues de igualar las dos funciones)

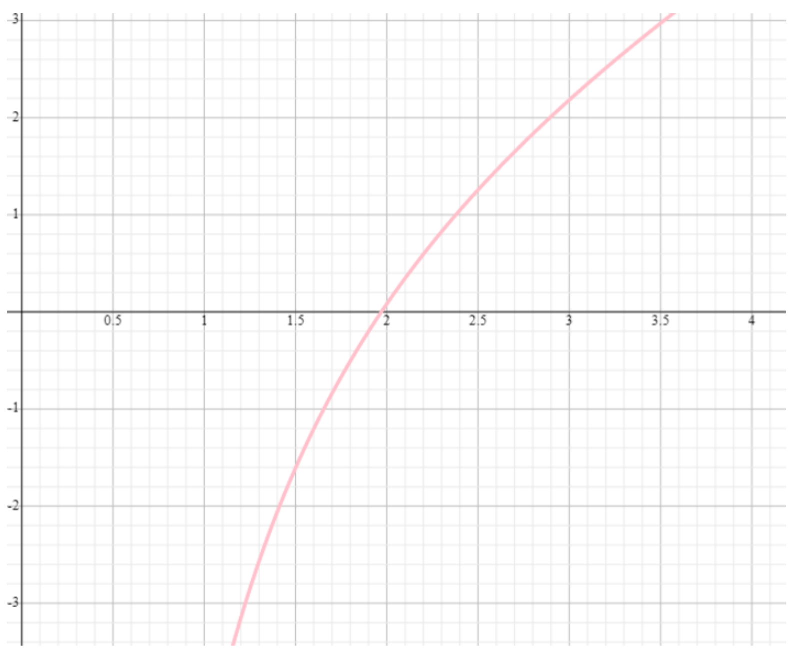


Figure 6: punto 2

codigo en R

```r
newton_rapshon = function( valor,tol){

  aux=valor
  num=0
  derivada=0

  plot(x = 1,
       xlab = "X",
       ylab = "Resultado",
       xlim = c(-3, 4),
       ylim = c(-3, 4),
       main = "Newton Rapshon",
       type = "n")


  repeat{
    resultado=aux-((sqrt(19)/sqrt(aux)-(10-aux^2)/aux)/(1+10/aux^2-sqrt(19)/((2*aux)^(3/2))))
    num=num+1
    funcion=sqrt((19)/sqrt(x))-((10-x^2)/x)
    if(funcion==0){
      break;
    }else{
      derivada=1+10/aux^2-sqrt(19)/((2*aux)^(3/2))
      if(derivada==0)
        break;
    }
    if(resultado < 2^(tol)){
      break;
    }else{
      cat ("Iteracion: ",num,"Resultado:\n\n")
      print(resultado,50)
            ## "valor de x",aux,"valor de f(x)",funcion,"valor f'(x)",derivada,"\n\n")
      points(x = aux, y = resultado)
      if(num>10)
         break;
    }

    aux=resultado
  }
}
x0 = mpfr(3,50)
tol=-16
newton_rapshon (x0,tol)
```

Figure 7: codigo 2

Resultados

```
1 'mpfr' number of precision  50   bits
[1] 1.7967774386072488113086365046910941600799560546875
Iteracion:  2 Resultado:

1 'mpfr' number of precision  50   bits
[1] 1.9462723383967546197936826501972973346710205078125
Iteracion:  3 Resultado:

1 'mpfr' number of precision  50   bits
[1] 1.9681756027771353245725549641065299510955810546875
Iteracion:  4 Resultado:

1 'mpfr' number of precision  50   bits
[1] 1.970043027333009177937128697521984577178955078125
Iteracion:  5 Resultado:

1 'mpfr' number of precision  50   bits
[1] 1.97018718222816602292368770577013492584228515625
Iteracion:  6 Resultado:

1 'mpfr' number of precision  50   bits
[1] 1.9701982125546972923757493845187127590179443359375
Iteracion:  7 Resultado:

1 'mpfr' number of precision  50   bits
[1] 1.9701990559882229803179143345914781093597412109375
Iteracion:  8 Resultado:

1 'mpfr' number of precision  50   bits
[1] 1.970199120477968079967467929236590862274169921875
Iteracion:  9 Resultado:

1 'mpfr' number of precision  50   bits
[1] 1.97019912540889663432608358561992645263671875
Iteracion:  10 Resultado:

1 'mpfr' number of precision  50   bits
[1] 1.9701991257859194917045897454954683780670166015625
Iteracion:  11 Resultado:

1 'mpfr' number of precision  50   bits
[1] 1.9701991258147462104943770100362598896026611328125
There were 11 warnings (use warnings() to see them)
> |
```

Figure 8: resultados 2

Si lo comparamos con la gráfica de la función, podemos suponer que hay una gran similitud. Ahora bien, necesitamos ver especficamente el valor que nos arroja el Wolphram.

$x \approx 1.970199125817132285706406$

Figure 9: resultados 2

Ya con la imagen de wolphran, podemos aceptar que es un valor en extremo cercano, lo cual nos demuestra que el metodo se ejecutó de manera correcta

3 Librerías en R

para el caso de Brent, los parámetros para estos algoritmos son:

a) Función: debe ser continua y tener signos diferentes en el rango AB

b) A: inicio de intervalo

c) B: fin de intervalo

d) maxiter: número máximo de iteraciones

e) tol: Tolerancia

También encontramos 4 valores en la salida.

a)Root : que viene siendo la raíz encontrada por el método

b)f.root : Nos da la ubicación de la raíz y los valores de la función cuando es evaluada en ese punto.

c)f.calls : numero aproximado de iteraciones usados

d)f.estim.prec : número aproximado de la precisión para la raíz

Ahora bien, hablando especificamente de la funcion "Brent" ubicada en el paquete "Pracama" encontramos el codigo correspondiente

```
library(pracma)

f <- function(x){
  x^3-2*x^2+(4/3)*x-(8/27)
}
polyroot(c((-8/27),(4/3),-2,1))

uniroot(f,c(0,1),tol=0.0000000000000000000000001)

brentDekker(f,-1,1,maxiter=39,tol=2^-50)

print(uniroot.all(f,c(0,1),tol=0.000000001),22) #rootsolve
```

Figure 10: Codigo Brent

A continuación, la salida. podemos observar como el valor es practicamente

```
$estim.prec
[1] 7.629395e-06

> library(pracma)
>
> f <- function(x){
+   x^3-2*x^2+(4/3)*x-(8/27)
+ }
> brentDekker(f,-1,1,maxiter=39,tol=2^-50)
$root
[1] 0.6666762

$f.root
[1] 7.771561e-16

$f.calls
[1] 40

$estim.prec
[1] 3.126852e-06
```

Figure 11: Codigo Brent

el mismo al realizado manualmente en el punto 1. igualmente encontramos que las llamadas aproximadas son 40 (aunque en realidad son menos porque pusimos 39 y pudimos ejecutar el codigo) y tambien podemos observar el error de estimacion que viene siendo la salida "estim.prec".

Ahora tenemos que analizar y comparar el punto 2, el cual nos pide hallar la intersección entre dos funciones (explicacion dada ya anteriormente)
Lo que nosotros hicimos fue utilizar la funcion Uniroot. ella nos haya cuando se intersecta(es decir, cuando es 0) una funcion en un intervalo, entonces utilizamos el paquete pracma.

```
library(pracma)

f <- function(x){
  (sqrt(19)/sqrt(x))-((10-x^2)/x)
}
polyroot(c(1,-1))

uniroot(f,lower = -3, upper = 4, f.lower = -3,  tol=2^-16)
```

Figure 12: Codigo Uniroot

Si comparamos la nueva salida, con la salida anterior. Nos encontramos con que la raiz en este caso esta en 0.66666701, inclusive encontramos que el error de aproximacion que nos da la función, es bastante bajo y dice adem´,as que esta ubicado en el punto 0. por estas mismas razones consideramos que está mal implementado, o que el codigo cuenta con algun error. En conclusión, es demasiado diferente al caso anterior donde nos da en 1.9701, aproximadamente

```
> uniroot(f,lower = -3, upper = 4, f.lower = -3,  tol=2^-16)
$root
[1] 0.6666701

$f.root
[1] 0

$iter
[1] 36

$init.it
[1] NA

$estim.prec
[1] 4.987804e-05

>
```

Figure 13: Codigo Uniroot