Pontificia Universidad Javeriana

Asignatura: Análisis Numérico
Docente: Eddy Herrera Daza

Reto 1

Integrantes:

María Camila Aguirre Collante
aguirrec.mcamila@javeriana.edu.co

Estefanía Bermúdez Arroyo
estefania_bermudeza@javeriana.edu.co

Jessica Tatiana Naizaque Guevara
j.naizaque@javeriana.edu.co

Angie Tatiana Peña Peña
penap_at@javeriana.edu.co

Marzo 2021

Índice

1. Introducción

En el presente documento, se encuentran las soluciones a los problemas planteados en el Reto 1 de Análisis Numérico. En este, se trabajaron los temas del Algoritmo de Brent, intersección entre dos curvas, los paquetes Pracma y RootSolve y las funciones base Polyroot y Uniroot. Durante el informe, se presentan los resultados obtenidos de las implementaciones realizadas en el software RStudio. También, se observan tanto condiciones, explicaciones y gráficas, como análisis, comparaciones y conclusiones. Adicionalmente, se evidencia la medición y relación del error en los algoritmos planteados para observar el comportamiento del método.

Conceptos Clave: Algoritmo de Brent, Intersección, Raíces reales, Método de Müller, Ecuación Polinomial, Raíces complejas

Palabras Clave: Algoritmo, Librerías, Iteraciones, Tolerancia, Convergencia, Significancia

2. Algoritmo de Brent

2.1. Condiciones Iniciales

Las condiciones iniciales para poder realizar este algoritmo son las siguentes:

- Las raíces deben encontrarse en el intervalo $[a, b]$, es decir,

$$f(a_k) \cdot f(b_k) < 0 \tag{1}$$

- La función debe ser continua en el intervalo $[a, b]$.
- Se debe cumplir que

$$|f(b)| < |f(a)| \tag{2}$$

de modo que b sea mejor suposición para la raíz, que a. (1)

2.2. Explicación del Algoritmo

El Algoritmo de Brent fue propuesto inicialmente por Theodorus Dekker, sin embargo, este en algunas ocasiones tenía una convergencia bastante lenta. Por lo cual, Richard Brent modificó el algoritmo de Dekker y es conocido como "*Método Brent-Dekker*". Este algoritmo busca raíces combinando los métodos de Bisección, Secante e Interpolación cuadrática inversa. Para lograr realizar cada iteración se calculan dos valores provisionales. El primero se realiza por medio del Método de la Secante y el segundo por el Método de Bisección, de la siguiente forma:

$$s\begin{cases} b_k - \left(\dfrac{b_k - b_{k-1}}{F(X=b_k) - F(X=B_{k-1})}\right) * (F(X=b_k)), & si\ F(b_k) \neq F(b_{k-1}) \\ m & de\ otra\ manera \end{cases}$$

Figura 1: Cálculo del Primer Valor Provicional

Y el cálculo del segundo valor provicional es:

$$m = \frac{a_k + b_k}{2}$$

Figura 2: Cálculo del Segundo Valor Provicional

Donde,

- b_k es la iteración actual, es decir, la estimación actual de la raíz de f.
- a_k es el "contrapunto", es decir, un punto tal que $f(a_k)$ y $f(b_k)$ tienen signos opuestos, por lo que el intervalo $[a_k, b_k]$ contiene la solución. Además, $|f(b_k)|$ debe ser menor o igual que $|f(a_k)|$, de modo que b_k es una menor estimación para la solución desconocida que a_k.
- b_{k-1} es la anterior iteración. (2)

2.3. Diagrama de Flujo

En la Figura 3, se puede observar el diagrama de flujo del programa del Algoritmo de Brent:

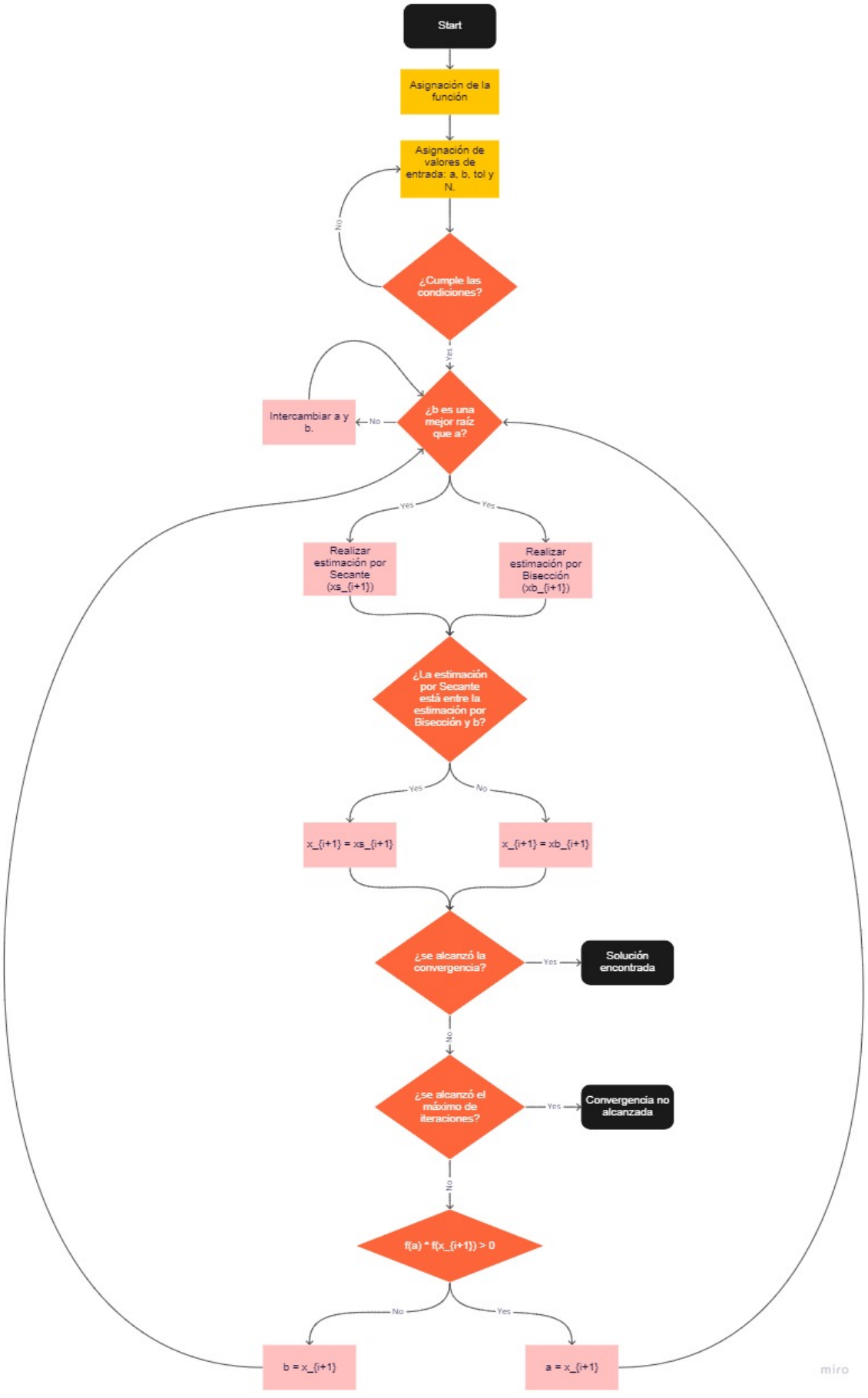

Figura 3: Diagrama de Flujo del Algoritmo de Brent

2.4. Código Fuente

En el Listing 1, se puede ver la implementación del Algoritmo de Brent en R:

```
library(pracma)
library(mi)
library(Rmpfr)
par(mar=c(1,1,1,1)) #Reajustar las configuraciones de Plot
Fx <- function(x) x^3 -2*x^2+(4/3)*x-(8/27) #Funcin

brentFun <- function(f,x0,x1,n,tol) #Declaracin de la funcin Brent
{

  error <- 1 #Valor del erro? permitido
  errores <- c() #Vector para guardar los valores de los errores
  listaErrorAct = c()
  listaErrorAnt = c()
  fx0 <- f(x0) #Guardar la funcin f evaluada en x0
  fx1 <- f(x1) #Guardar la funcin f evaluada en x1
  if(fx0 *fx1 >=0) #Validar que?f(x0) * f(x1) >= 0
  {
    return("La funcin no est  entre corchetes")
  }

  if(abs(fx0) < abs(fx1)) # Intercambiar los valores de x0 y x1 para que
    se cumplan las condiciones
  {
    aux1 = x0
    x0 = x1
    x1 = aux1

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

  while(i < n & abs(x1-x0) > tol) #Ciclo que se repite hasta la
   tolerancia deseada
  {
    errorAnterior = error

    fx0 = f(x0)
    fx1 = f(x1)
    fx2 = f(x2)

    if(f?0 != 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)|
       (band?ra==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(?bs(fx0)<abs(fx1))
    {
      aux3 = x0
      x0 = x1
      x1 = aux3
    }

    error = abs(x1-x0)
    errores = c(errores,as.double(error))

    if(i>1)
    {
      listaErrorAnt = c(listaErrorAnt , as.double(errorAnterior))
      listaErrorAct =?c(listaErrorAct, as.double(error))
    }

    i = i + 1
  }

  iteraciones <- c(1:i)
  #Realizar  G r fico  de  los  errores

  plot(iteraciones, errores, main = "Medicion del error Brent", xlab= "
    Iteraciones", ylab = "Errores", type = 'o', col="blue")?  plot(
    listaErrorAnt,listaErrorAct, main = "Relacin de error Brent",xlab =
    " Error i ",ylab = " Error i+1 ",type= 'o',col ="red")
  retorno = c(x1,i)

  return(retorno)
}
```

```
#Declaracin de la funcin Muller (La funcin se realiz anteriormente)
mullerFun <? function(f){

  #Asignar el Intervalo
  x0 <- 1.0
  x1 <- 0.0

  #Por medio del metodo de biseccion se halla el valor de x2
  x2 <- (x0 + x1)/2
  tol <- 1e-5
  #Obtener el numero maximo de iteraciones
  N <- (1/log(2))*(log(abs(x0-x1)/tol))

  #Meto?o de muler
  h1 <- x1-x0
  h2 <- x2-x1
  d1 <- (f(x1)-f(x0))/h1
  d2 <- (f(x2)-f(x1))/h2
  a <- (d2-d1)/h2+h1
  i <- 0

  #Ciclo que se repite hasta el numero de iteraciones
  while(i< N){b<-d2+h2*a
  D<-(b^2-4*(f(x2)*a))^(1/2)
  if (b > 0){E<-b+D}
  if?(b < 0){E<-b-D}
  h <- 2*f(x2)/E
  p <- x2+h
  if (abs((p-x1)/p)<tol){p
    break}
  x0<-x1
  x1<-x2
  x2<-p
  h1<-x1-x0
  d1<-(f(x1)-f(x0))/h1
  d2<-(f(x2)-f(x1))/h2
  a <-(d2-d1)/h2+h1
  i<-i+1
  }

  mpfr(p,168)

}
#---------------------------------?----#

#Grfico de la funcin
x1 <- seq(-10,10,0.01)
plot (x1,Fx(x1),type="l",col="red", xlab = "x",ylab = "y")
abline(h=0,col="blue")



n = 1000
tol = 2^-50
a = 0.0
b = 5.0

cat (" Mtodo de Brent para la funcin x^3 -2*x^2+(4/3)*x-(8/27)")
```

```
166
167 t <- proc.tim?()
168 res = brentFun(Fx,a,b,n,tol)
169 proc.time()-t
170 cat ("Raiz:")
171 mpfr(res[1],168)
172 cat ("Nmero de iteraciones", res[2])
173
174
175 cat (" Mtodo de Muller para la funcin x^3 -2*x^2+(4/3)*x-(8/27)")
176 t <- proc.time()
177 mullerFun (Fx)
178 proc.time()-t
```

Listing 1: Código en R de Algoritmo de Brent Vs Método de Müller

2.5. Solución del Problema

Al aplicar el Algoritmo de Brent en la ecuación

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

se obtienen los siguientes resultados:

Tolerancia	Brent	Iter. Brent	Müller	Iter. Müller
$1e^{-8}$	0.6666660308837890625	20	0.66772071719867598372	27
$1e^{-16}$	0.6666660308837890625	20	0.66706789141243927332	54
$1e^{-32}$	0.6666660308837890625	20	0.66680203816653449422	107
$1e^{-56}$	0.6666660308837890625	20	0.66671689200353145832	187
$2e^{-50}$	0.6666660308837890625	20	0.66672907400389769261	166

Tabla 1: Resultados obtenidos para $f(x) = x^3 - 2x^2 + \frac{4x}{3} - \frac{8}{27}$

2.6. Gráficas

La gráfica de la medición del error del Algoritmo de Brent es:

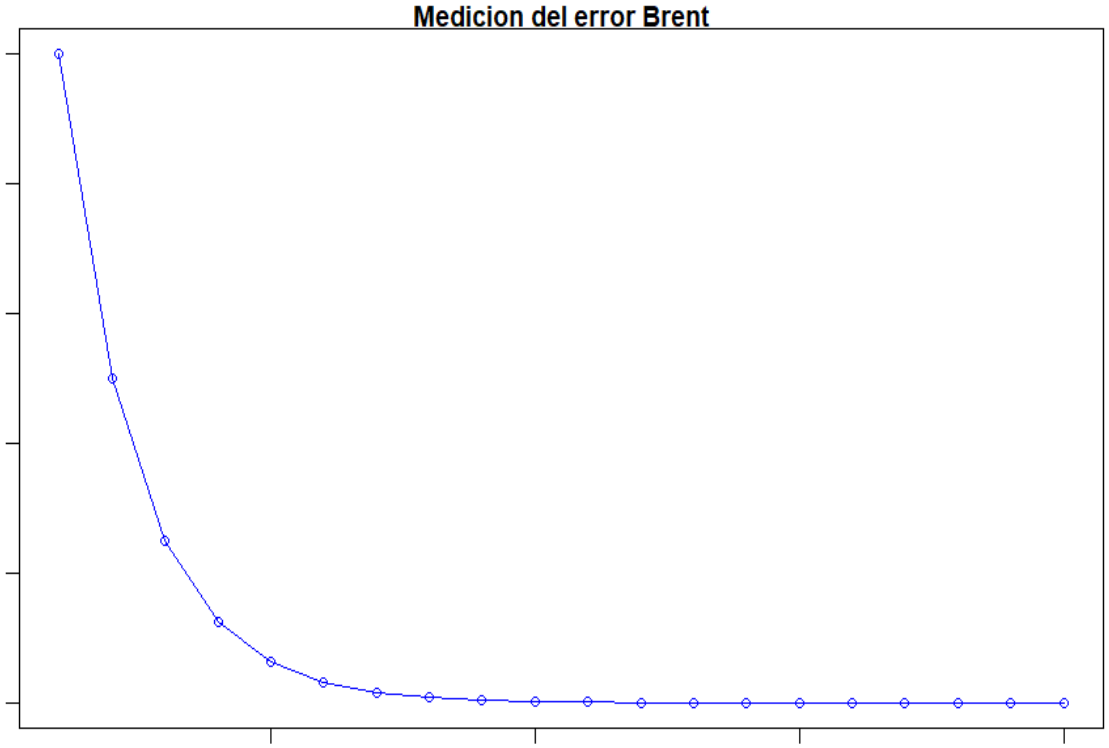

Figura 4: Error de la Medición del Algoritmo de Brent

La gráfica de la relación de error del Algoritmo de Brent es:

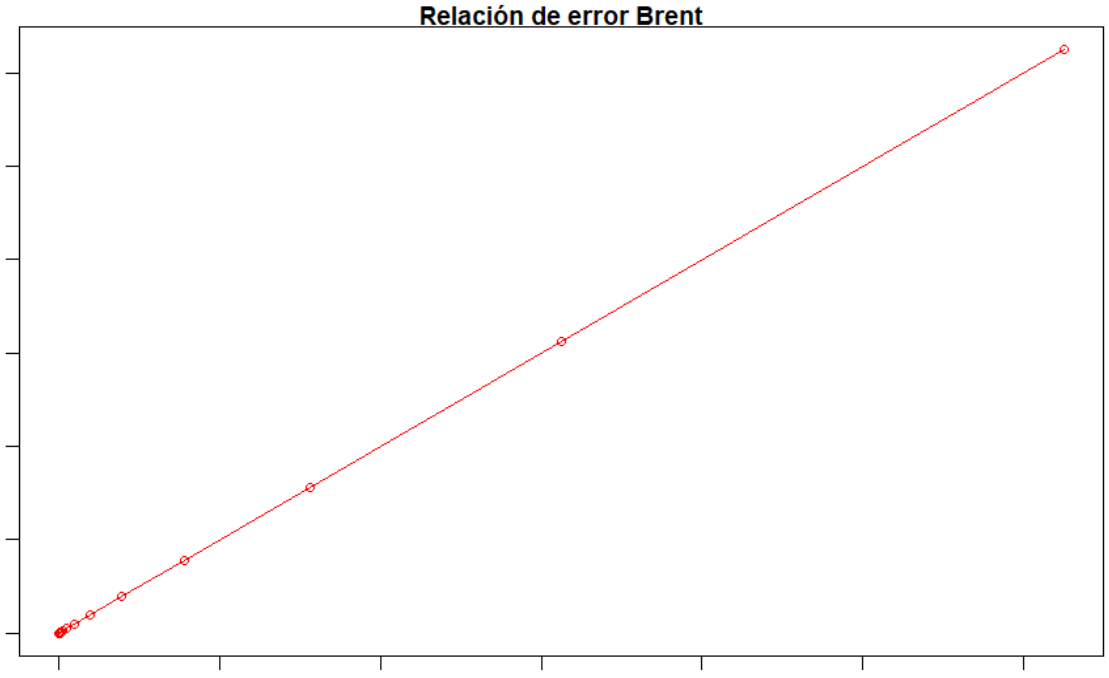

Figura 5: Relación del Error del Algoritmo de Brent

2.7. Algoritmo Brent Vs Método Müller

Algunas diferencias que existen entre el Método de Müller y el Algoritmo de Brent son:

- El Algoritmo de Brent utiliza tres métodos como base: el Método de la Secante, el de la Interpolación Cuadrática Inversa y el de Bisección. En cambio, el Método de Müller solo tiene como referencia el Método de la Secante y Bisección.
- El Método de Müller converge rápidamente, pero requiere más cálculos por iteración (3) y el Algoritmo de Brent converge superlinealmente. (4)
- Al ejecutar el código fuente (Listing 1), se puede observar que el Algoritmo de Brent tiene más tiempo de ejecución que el Método de Müller.
- El Método de Müller realiza mayor cantidad de iteraciones que el Algoritmo de Brent.
- El Algoritmo de Brent no es aplicado si las raíces a encontrar son complejas, mientras que el Método de Müller si logra encontrarlas. (5)
- El Algoritmo de Brent es más preciso que el Método de Müller.

3. Intersección entre Curvas

3.1. Explicación

En este documento, se usó el Método de Müller para encontrar los puntos de intersección entre dos curvas.

Si se tienen dos curvas $h(x)$ y $g(x)$, para encontrar los puntos de intersección se requiere encontrar todos los valores de x tal que $h(x) = g(x)$, que es lo mismo que resolver $h(x) - g(x) = 0$. Se puede considerar $f(x) = h(x) - g(x)$, por ende, por medio del Método de Müller se puede encontrar el x tal que $f(x) = 0$.(6)

3.2. Diagrama de flujo

En la Figura 6, se puede observar el diagrama de flujo del programa del Método de Müller:

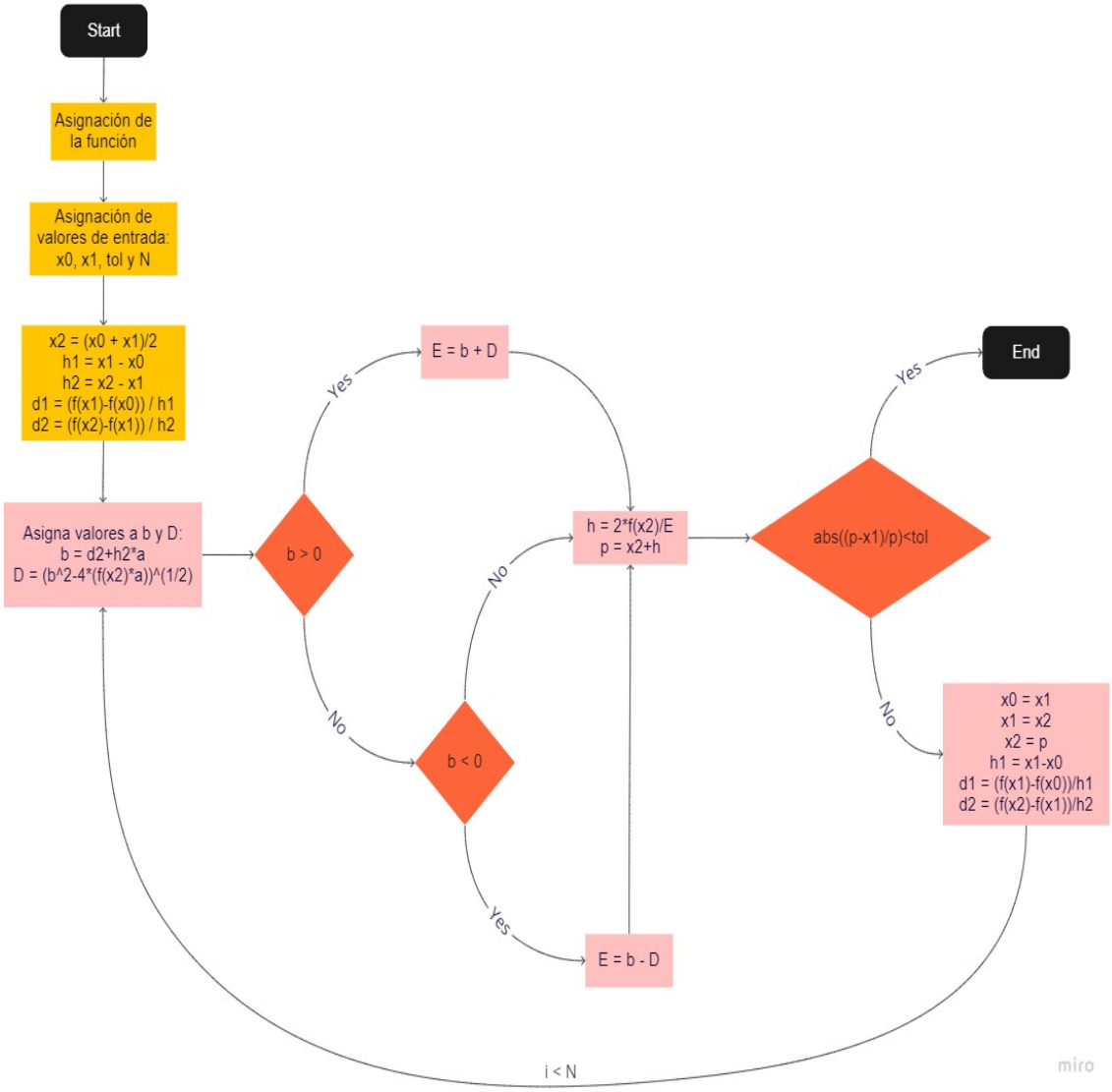

Figura 6: Diagrama de flujo del Método de Müller

3.3. Código Fuente

En el Listing 2, se puede observar la implementación del Método de Müller para encontrar la intersección entre curvas en R:

```
library(Rmpfr)

f <- function(x) (-(sqrt(6840*x^3 + 3249*x^2-1140*x+100)-57*x+10)/60*x)
    #Funcion

#Creacion del vector que guardara la sucesion
iniciales <- c()

#Definir el intervalo
x0 <- 0.56
x1 <- 3.0

# Por medio del metodo de biseccion obtenemos el valor de x2
```

```r
x2 <- (x0 + x1)/2

# Asignamos el valor de la tolerancia
tol <- 1e-16

# Obtenemos el n mero  m ximo  de iteraciones
N <- (1/log(2))*(log(abs(x0-x1)/tol))
h1 <- x1-x0
h2 <- x2-x1
d1 <- (f(x1)-f(x0))/h1
d2 <- (f(x2)-f(x1))/h2
a <- (d2-d1)/h2+h1
i <- 0
cat(tol, "\n")
iniciales <- c()
valida <- TRUE
#Ciclo que se repite hasta el n mero de  m ximo  de iteraciones
while(valida == TRUE){
  b <- d2+h2*a
  D<-(b^2-4*(f(x2)*a))^(1/2)
  #Validacin para la asignacin del E
  if (b > 0){E<-b+D}
  if (b < 0){E<-b-D}
  h <- 2*f(x2)/E
  result <- x2+h
  cat(result, "\n")
  iniciales <- c(iniciales,result)
  #Condicional que para el programa cuando el valor sea menor a la
   toleracia
  if (abs((result-x1)/result)<tol){
    valida == FALSE
    result
    break
  }

  #Asignacin de las variables
  x0 <- x1
  x1 <- x2
  x2 <- result
  h1 <- x1-x0
  d1 <- (f(x1)-f(x0))/h1
  d2 <- (f(x2)-f(x1))/h2
  a <-(d2-d1)/h2+h1
  i <- i+1
}
cat("Iter: ", i)
# Analizar convergencia
lineal <- FALSE
for (t in 1:i){
  if (iniciales[t+1] == 0.5*(iniciales[t])){
    lineal <- TRUE
  } else {
    lineal <- FALSE
  }
}
if (lineal == TRUE){
  cat("La convergencia es lineal")
} else {cat("La convergencia NO es lineal")}
```

```
70
71 cat("Usando precisin de la funcin mpfr obtenemos como resultado: \n")
72 mpfr(result, 500)
```

Listing 2: Código en R de intersección entre dos curvas

3.4. Solución del Problema

Se quiere encontrar la intersección entre las ecuaciones 4 y 5:

$$x^2 + xy = 10 \tag{4}$$

$$y + 3xy^2 = 57 \tag{5}$$

Teniendo en cuenta lo establecido en la sección 3.1, se puede despejar las ecuaciones 4 y 5, obteniendo como resultado las ecuaciones 6 y 7, respectivamente.

$$\frac{x^2 + xy}{10} = 1 \tag{6}$$

$$\frac{y + 3xy^2}{57} = 1 \tag{7}$$

Evidentemente, se pueden igualar las ecuaciones, dando como resultado la siguiente igualdad:

$$\frac{x^2 + xy}{10} = \frac{y + 3xy^2}{57} \tag{8}$$

Con el fin de usar el Método de Müller, se despeja la ecuación 8 para que quede de la forma $f(x) = 0$

$$\frac{x^2 + xy}{10} - \frac{y + 3xy^2}{57} = 0 \tag{9}$$

$$\frac{57x^2 + 57xy - 10y - 30xy^2}{570} = 0 \tag{10}$$

$$57x^2 + 57xy - 10y - 30xy^2 = 0 \tag{11}$$

Para poder despejar la ecuación 11 en función de x, se usó la herramienta Wolfram Alpha. Al realizar este despeje, se obtiene el valor de y.

$$y_1 = -\frac{\sqrt{6840x^3 + 3249x^2 - 1140x + 100} - 57x + 10}{60x} \tag{12}$$

Al aplicar el Método de Müller en la ecuación 12 y teniendo en cuenta el intervalo $[0.56, 3]$ se obtienen los siguientes resultados:

Tolerancia	Resultado
$1e^{-8}$	2.00635381057259
$1e^{-16}$	2.00635381057259
$1e^{-32}$	2.00635381057259
$1e^{-56}$	2.00635381057259

Tabla 2: Resultados obtenidos para $y_1 = -\frac{\sqrt{6840x^3+3249x^2-1140x+100}-57x+10}{60x}$

Como se puede observar en la Tabla 2, el resultado de x_1 es $x_1 = 2.00635381057259$. Si se reemplaza este valor en la ecuación 4, se obtiene que el resultado de y_1 es $y_1 = 2.977811966821436$. Por ende, se encuentra que el primer punto de intersección es:

$$(2.00635381057259, 2.977811966821436).$$

Al aplicar el Método de Müller en la ecuación 12 y teniendo en cuenta el intervalo $[2.3, 5]$ se obtienen los siguientes resultados:

Tolerancia	Resultado
$1e^{-8}$	4.35379339083584
$1e^{-16}$	4.35379339083584
$1e^{-32}$	4.35379339083584
$1e^{-56}$	4.35379339083584

Tabla 3: Resultados obtenidos para $y_1 = -\frac{\sqrt{6840x^3+3249x^2-1140x+100}-57x+10}{60x}$

Como se puede observar en la Tabla 3, el resultado de x_2 es $x_2 = 4.35379339083584$. Si se reemplaza este valor en la ecuación 4, se obtiene que el resultado de y_2 es $y_2 = -2.05694576801371$. Por ende, se encuentra que el segundo punto de intersección es:

$$(4.35379339083584, -2.05694576801371)$$

Al graficar las ecuaciones 4 y 5 en la herramienta GeoGrebra, se obtiene que los puntos de intersección son:

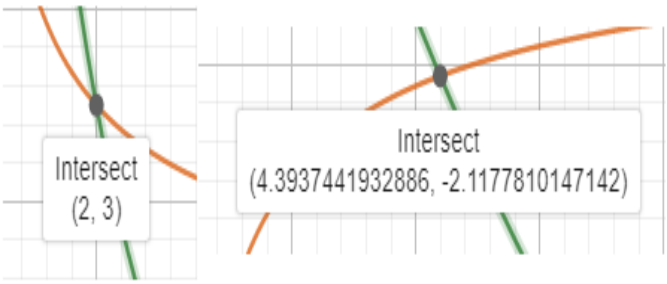

Figura 7: Puntos de Intersección entre la ecuación X y Y

Al comparar los resultados dados usando el Método de Müller con los de la Figura 5, es notorio que el resultado generado por el Método de Müller genera una situación de mal condicionamiento, debido a que la entrada de x en la ecuación 4 tienen un pequeño error, generando que el valor de y difiera al valor real.

3.5. Gráficas

En la Figura 8 se puede observar la gráfica de las ecuaciones 4 y 5 y las intersecciones que existen entre estas. Esta gráfica fue hecha con la herramienta GeoGebra.

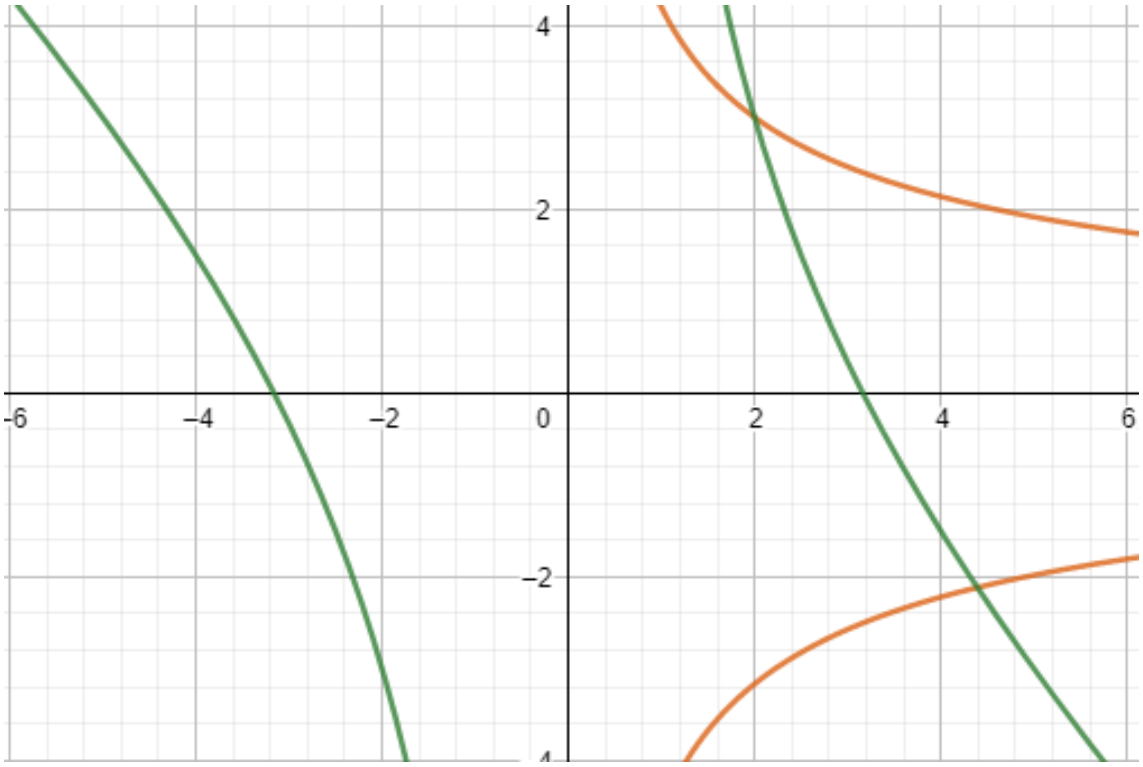

Figura 8: Gráfica de la Intersección entre la Ecuación 4 y 5

4. Librerías en R

4.1. Función Base Polyroot

Esta es una función de R que se usa para encontrar raíces de una ecuación polinomial. Es decir, tiene la forma:

$$p(x) = z_1 + (z_2 \cdot x) + (z_3 \cdot x_2) + ... + (z_n \cdot x_{n-1}) \tag{13}$$

La función se utiliza así: ***polyroot (z)*** , donde z es un vector de coeficientes polinomiales ubicados en orden ascendente. (7)

Esta función se encarga de hacer uso del algoritmo "Jenkins-Traub", el cual tiene como objetivo calcular raíces polinomiales, no obstante, este es más eficiente en comparación con otros métodos porque converge a raíces polinomiales a una mejor tasa que la fórmula cuadrática. (8)

Con ayuda de este algoritmo se encuentran los $n - 1$ ceros complejos de la ecuación polinomial. El vector z es evaluado y si las mayores potencias tienen cero, se descarta su uso. Polyroot permite cualquier grado del polinomio, sin importar cuán grande sea. Sin embargo, la estabilidad numérica presentada funcionará de forma óptima únicamente para los polinomios de bajo grado. (9)

Por ejemplo, se tiene la ecuación polinomial

$$p(x) = 12 - 2x + 3x^2 \tag{14}$$

en donde se tienen los coeficientes 12, -2 y 3, respectivamente. Por lo cual, aplicando la función, se tiene el siguiente código aplicado en R (7):

```
ecuacion <- c(12,-2,3)
polyroot(ecuacion)
```

Listing 3: Código en R Polyroot de la ecuación

Y el resultado fue:

$$x_1 = 0.333333 + 1.972027i$$
$$x_2 = 0.333333 - 1.972027i$$

Resolviendo esta ecuación en el software Wolfram Alpha, se obtiene el siguiente resultado:

$$x = \frac{1}{3}\left(1 - i\sqrt{35}\right) \qquad x \approx 0.33333333333333 - 1.97202659436654\,i$$
$$x = \frac{1}{3}\left(1 + i\sqrt{35}\right) \qquad x \approx 0.33333333333333 + 1.97202659436654\,i$$

Figura 9: Raíces de $p(x) = 12 - 2x + 3x^2$ en Wolfram Alpha

Como otro ejemplo, se tiene la ecuación polinomial

$$p(x) = \frac{1}{2} - 5x + 8x^2 - 4x^3 + x^4 \tag{15}$$

con los coeficientes 1/2, -5, 8, -4 y 1, respectivamente.

Y el código implementado en R es:

```
ecuacion <- c(1/2, -5, 8, -4, 1)
polyroot(ecuacion)
```

Listing 4: Código en R Polyroot de la ecuación

Y el resultado fue:

$$x_1 = 0.1226314 + 0.000000i$$
$$x_2 = 0.8084142 - 0.000000i$$
$$x_3 = 1.5344772 - 1.639788i$$
$$x_4 = 1.5344772 + 1.639788i$$

Resolviendo esta ecuación en el software Wolfram Alpha, se obtiene el siguiente resultado:

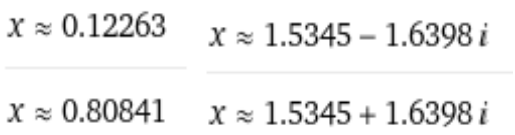

Figura 10: Raíces de $p(x) = \frac{1}{2} - 5x + 8x^2 - 4x^3 + x^4$ en Wolfram Alpha

Con lo cual, se concluye que la función da los resultados correctos, sin embargo, con una menor cantidad de cifras significativas, lo cual hace que el resultado pierda precisión.

4.2. Función Base Uniroot

La función, que pertenece a R, se encarga de buscar una raíz dentro del intervalo que se envía como parámetro por medio de dos variables (interval, lower y upper). La función se implementa de la siguiente forma:

```
uniroot(f, interval, ...,lower = min(interval), upper = max(interval),
        f.lower = f(lower, ...), f.upper = f(upper, ...),
        tol = .Machine$double.eps^0.25, maxiter = 1000)
```

Listing 5: Implementación de Uniroot

Donde,

- `f` → Es la función para la que se busca la raíz
- `interval` → Es un vector de los puntos finales del intervalo
- `...` → Argumentos que se le envían a la función f
- `lower, upper` → Son los puntos finales extremos del intervalo
- `f.lower, f.upper` → Son las imágenes de los puntos extremos del intervalo evaluadas en la función
- `extendInt` → Como f(upper) y f(lower) deben tener signos diferentes, si este caracter ingresado es "no", se mostrará un error si ambas imágenes tienen el mismo signo. Si el caracter es "yes", el intervalo se extiende en ambos lados hasta que un signo cambie satisfaciendo la condición
- `check.conv` → Indica si una advertencia de convergencia de la función "uniroot" debe detectarse como error y si la no convergencia en la máxima iteración debe ser un error y no una advertencia
- `tol` → Tolerancia
- `maxiter` → Número máximo de iteraciones
- `trace` → Se ingresa un número positivo para producir información de rastreo (10)

Para el ejemplo, se toma la ecuación 1, utilizando el siguiente código implementado en R:

```r
fun <- function(x) {x^3 - 2*x^2 +4*x/3 - 8/27}
uniroot(fun, lower=0, upper=1)
```

Listing 6: Código en R UniRoot para la ecuación 1

El resultado fue:

```
$root
[1] 0.666647

$f.root
[1] -7.660539e-15

$iter
[1] 25

$init.it
[1] NA

$estim.prec
[1] 6.103516e-05
```

Figura 11: Raíces de la Ecuación 1

Resolviendo esta ecuación en el software Wolfram Alpha, se obtiene el siguiente resultado:

$$x \approx 0.666666666666667$$

Con lo cual, se concluye que la funcion da los resultados correctos, sin embargo, con una menor cantidad de cifras significativas, lo cual hace que el resultado pierda precision. Además, la función tiene 5 salidas las cuales presentan información adicional a la raíz de la ecuación, como por ejemplo, el número de iteraciones realizadas.

4.3. Paquete Pracma

Utilizando la optimización y las rutinas de series de tiempo, el paquete pracma en R proporciona funciones avanzadas que puedan ser ejecutadas para el análisis numérico de diferentes problemas matemáticos. Se usan nombres de funciones de otros softwares tales como: Octave y Matlab, esto con el fin de minimizar la portabilidad.

El paquete contiene funciones para encontrar raíces, resolver matrices especiales, ordenar rutinas, ecuaciones diferenciales y sistemas de ecuaciones ordinarios y para ser utilizadas en interpolación, polinomios y diferenciación e integración numérica. (11)

4.4. Paquete RootSolve

El paquete rootSolve se creó con el objetivo de aplicar funciones que:

- Puedan generar matrices jacobianas y gradientes.
- Encuentren raíces de ecuaciones no lineales utilizando el método Newton-Raphson.

- Por medio del método de Newton-Raphson o de una ejecución dinámica, logren estimar condiciones (de un estado estable) de un sistema de ecuaciones completas, en bandas o dispersa.
- Resuelvan condiciones de estado estacionario para ecuaciones diferenciales parciales de uno o varios componentes, que se han convertido a EDO mediante diferenciación numérica. (12)

5. Conclusiones

- Gracias al análisis realizado en este documento se puede concluir que el Algoritmo de Brent es mucho más preciso que el Método de Müller y, así mismo, realiza una cantidad menor de iteraciones para encontrar las raíces de las ecuaciones.
- El Algoritmo de Brent tiene bastantes ventajas en cuando al Método de Müller, una de ellas es que tiene la facilidad de utilizar tres métodos en uno solo. Como se mencionó anteriormente, estos son: Método de la Secante, Método de Bisección e Interpolación Cuadrática Inversa.
- Un gran contra del Método de Müller es que cada iteración hace que se descarte una posible raíz del resultado esperado.
- Dependiendo del método utilizado, la convergencia será diferente, por lo cual, se establece que no siempre se llegará con la misma velocidad al resultado esperado.
- Se debe tener cuidado al momento de definir los intervalos a analizar cuando se realice el Método de Müller.
- En este documento, se puede ver un ejemplo de un problema mal condicionado, donde un pequeño error en la entrada, genera un gran cambio en la solución.
- Al tener concimiento de las librerías, es más fácil la implementación en los códigos.

Referencias

Trinh, K. Brent's method, 2021. Recuperado de: https://kevinttrinh.com/brents-method/ [1].

Vásquez, A. Método de Brent, s.f. Recuperado de: https://es.scribd.com/document/372893988/Me-todo-de-Brent [2].

O'Daniel Ray, C. Comparative analysis of polynomial root finding techniques, 1967. Recuperado de: https://scholarsmine.mst.edu/masters_theses/2941 [3].

Anónimo. Método de Brent - Brent's method Método de Brent, 2021. Recuperado de: https://es.qaz.wiki/wiki/Brent%27s_method [4].

De Simone, P. Haarth, R. *Estudio de casos prácticos*. 2009. [5].

Jhevon. Use Newton's Method for Intersection of 2 Curves, 2010. Recuperado de: https://mathhelpforum.com/threads/solved-use-newtons-method-for-intersection-of-2-curves.136788/ [6].

Nidhi. Find roots or zeros of a Polynomial in R Programming { polyroot() Function, 2020. Recuperado de: https://www.geeksforgeeks.org/find-roots-or-zeros-of-a-polynomial-in-r-programming-polyroot-function/ [7].
Binner, D. Polynomial Root-finding with the Jenkins-Traub Algorithm, 2008. Recuperado de: https://mathblog.com/polynomial-root-finding-with-the-jenkins-traub-algorithm/ [8].
Base de datos R Documentation. Polyroot, s.f. Recuperado de: https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/polyroot [9].
Base de datos R Documentation. Uniroot, s.f. Recuperado de: https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/uniroot [10].
Borchers, H. Package 'pracma', 2021. Recuperado de: https://cran.r-project.org/web/packages/pracma/pracma.pdf [11].
Base de datos R Documentation. RootSolve, s.f. Recuperado de: https://www.rdocumentation.org/packages/rootSolve/versions/1.8.2.1/topics/rootSolve-package [12].