Reto Parcial Analisis Numérico

Maria Paula Covelli Reyes
David Mateo Henao Prieto
Juan Sebastian Pulecio Romero

Septiembre 13, 2020

1. Evaluación de las raíces de un Polinomio

Evaluar las raíces de un polinomio implica varios desafíos, aún para el software profesional. Como se requiere poder evaluar las posibles raíces del polinomio, es necesario dedicarle un momento a los detalles como, las derivadas evaluadas en valores x_0.

1.1. Sea $P(x) = a_0x^n + a_1x^{n-1}) + ... + an$ un polinomio, entonces, implementar en R y/o Python una modificación del método de Horner que evalue de manera eficiente $f'(x0)$ y $f''(x0)$ la primera y segunda derivada en cualquier punto

Explicacion teorica

Antes de iniciar a solucionar el punto se tiene en cuenta que el metodo de Horner es un algoritmo que permite calcular el resultado de un polinomio para un determinado valor de x. Lo que hace en si el metodo es reducir la cantidad de operaciones que se necesiten para llegar a un resultado, esto fomenta una tecnica mas eficiente, con menos tiempo y memoria gastada al momento de programarlo en un computador.

El metodo de Horner nos dice que, sea $P(x) = a_0x^n + a_1x^{n-1} + ... + a_n$ si dividimos la función Sea $P(x)$ entre el termino Sea $(x - x_0)$, obtenemos:

$$\frac{P(x)}{(x - x_0)} = \mathrm{Q(x)} + \mathrm{b_0}$$

Donde podemos decir que $b_0 = 0$ si x_0 es una raiz de $P(x)$ y $Q(x)$ es un polinomio de grado $n - 1$. Entonces, por definición, podemos obtener que:

$$Q(x) = b_n x^{n-1} + b_{n-1}x^{n-2} + ... + b_1$$
$$(x - x_0)Q(x) + b_0 = (x - x_0)(b_n x^{n-1} + b_{n-1}x^{n-2} + ... + b_1) + b_0$$
$$(x - x_0)Q(x) + b_0 =$$
$$(b_n x^n + b_{n-1}x^{n-1} + ... + b_1 x)(b_n x_0 x^{n-1} + b_{n-1}x_0 x^{n-2} + ... + b_1 x_0) + b_0$$
$$P(x) = b_n x^n + (b_{n-1} - b_n x_0)x^{n-1} + ... + (b_1 - b_2 x_0)x + (b_0 - b_1 x_0)$$

Dado que los coeficienres a son conocidos entonces de manera recursiva podemos inferir que:

$$a_n = b_n$$
$$b_k = a_k + b_{k+1}x_0 \text{ con } k = 0, 1, 2, \ldots, n-1$$
$$P(x) = (x - x_0)Q(x) + b_0$$
$$P(x) = (x - x_0)Q(x_0) + b_0 = b_0$$

Si deseamos derivar la funcion $P(x)$

$$P'(x) = (x - x_0)Q'(x_0) + Q(x)$$
$$P'(x_0) = (x_0 - x_0)Q'(x_0) + Q(x_0)$$
$$Q(x) = (x - x_0)R(x) + b_0^{'}$$

Si deseamos derivar la funcion $P'(x)$

$$P''(x) = (x - x_0)Q''(x_0) + Q'(x)$$
$$P''(x_0) = (x_0 - x_0)Q''(x_0) + Q'(x_0)$$
$$Q'(x) = (x - x_0)R(x) + b_0^{''}$$

1.1.1. Primera y segunda derivada en números reales

Para este caso se realizó una función adicional la cual nos permite derivar el polinomio inicial. Esta función nos da como resultado un vector con los coeficientes del polinomio ya derivados.

- **Entradas**:

  Coeficientes = 1,-5,-9,155,-250

  $x_0 = 2$

- **salidas**:

  Coeficientes = 4,-15,-18,155

  Resultado es 91

  número minimo de operaciones de 6 con 3 sumas y 3 multiplicaciones.

Para este caso de la segunda derivada la función nos da como resultado un vector con los coeficientes del polinomio ya derivados por segunda vez.

- **Entradas**:

  Coeficientes = 1,-5,-9,155,-250

  $x_0 = 2$

- **salidas**:

  Coeficientes = 12,-30,-18

  Resultado es -30

  número minimo de operaciones de 4 con 2 sumas y 2 multiplicaciones.

1.1.2. Algoritmo Evaluación de la primera y segunda derivada en números reales

Algorithm 1 Evaluación de la primera y segunda derivada de un polinomio con una modificación de Horner

```
#función inicial x^4-5x^3-9x^2+155x-250
     #primera derivada 4x^3-15x^2-18x+155
     #segunda derivada 12x^2-30x-18

horner <- function(coeficientes, x)
{
    #implementación del método de Horner, antes debemos
    conocer los coeficientes de la derivada

    eval <- coeficient
    i <- 0
    for(j in coeficientes[2:length(coeficientes)])
        {
        eval <- x*eval + j
        i <- i + 2
        }
    return(cat("\nEl resultado de la evaluación en ",x,"
    es: ",eval ,"\nEl numero de operaciones realizadas
    fue de: ", i))
}

porderivar <- function(coeficientes)
{
    #Esta función se encarga de encontrar los coeficientes
    de cada una de las variables  derivar, acompañadas por
    su respectivo exponente

    grado <- length(coeficientes)-1
    deriv <- c()
    for(i in coeficientes[1:length(coeficientes)-1])
        {
        aux <- i*grado
        deriv <- c(deriv, aux)
        grado <- grado - 1
        }
    return (deriv)

    #retorna un vector con los nuevos coeficientes
    (derivada)
}
```

```
x0 <- -2 #Valor con el que se van a evaluar las derivadas

coeficientes <- c(1,-5,-9,155,-200). #polinomio original


#primera derivada
derivada<-porderivar(coeficientes)
cat("\nPrimera derivada:")
cat ("\nLos coeficientes de la derivada son",derivada)
horner(derivada,x0)
cat("\n-------------------------------------------------")

#segunda derivada
derivada<-derivar(derivada)
cat("\nSegunda derivada:")
cat ("\nLos coeficientes de la derivada son",derivada)
horner(derivada,x0)
```

1.1.3. Explicación Algoritmo

El código implementado cuenta con los valores de entrada de los coeficientes que acompañan las X del polinomio puestos en un orden determinado con el fin de facilitar la implementación por lo cual si usted ve que los coeficientes son 1,-5,-9,155,-200 esto quiere decir que el primero valor (1) vendrá acompañado por un valor de X elevado a la n-1 (4) y el último valor del vector (-200) vendrá acompañado por una X elevada a la 0; es decir 1.

El otro dato que se tendrá de entrada tiene que ver con el valor por el que se desea reemplazar las X del polinomio después de realizar las respectivas derivadas (primera y segunda). Por ende el proceso a seguir será:

1. Realizar la derivada
2. Encontrar los coeficientes que acompañan dicha derivada
3. Reemplazar las X de la derivada por el valor propuesto
4. Encontrar el total después de realizar un número de operaciones

Por lo mencionado anteriormente a la primera función del código a la que van a entrar los valores iniciales será a la función porderivar la cual retornará un vector de los coeficientes de dicha derivada; Posterior a esos valores retornados se llevarán a la función Horner junto con el valor por el que se desean reemplazar las X. En dicha función se obtendrá el respectivo valor de la solución; es decir el valor que se obtiene al reemplazar el valor inicial en cada una de las X.

Para encontrar la evaluación de la segunda derivada lo único que hay que hacer es enviarle a la función porderivar los coeficientes de la derivada para que obtenga los coeficientes de esa función ya derivada y posteriormente enviarle a la función Horner los coeficientes obtenidos en la función anterior junto con el valor a reemplazar

1.1.4. Resultados

Resultados de primera y segunda derivada con un valor de $x_0 = 2$:

```
Primera derivada:
Los coeficientes de la derivada son 4 -15 -18 155
El resultado de la evaluacion en 2 es: 91
El numero de operaciones realizadas fue de: 6
-----------------------------------------------------
Segunda derivada:
Los coeficientes de la derivada son 12 -30 -18
El resultado de la evaluacion en 2 es: -30
El numero de las operaciones realizadas fue de: 4
```

Resultados de primera y segunda derivada con un valor de $x_0 =$:

```
Primera derivada:
Los coeficientes de la derivada son 4 -15 -18 155
El resultado de la evaluacion en 3.141593 es:74.43237
El numero de las operaciones realizadas fue de: 6
-----------------------------------------------------------
Segunda derivada:
Los coeficientes de la derivada son 12 -30 -18
El resultado de la evaluacion en 3.141593 es: 6.187473
El numero de las operaciones realizadas fue de: 4
```

1.2. Primera y segunda derivada en números Imaginarios

Para este caso se realizó una función adicional la cual nos permite derivar el polinomio inicial. Esta función nos da como resultado un vector con los coeficientes del polinomio ya derivados.

- **Entradas**:

  Coeficientes = 1,-5,-9,155,-250

  $x_0 = 1 + i$

- **salidas**:

  Coeficientes = 4,-15,-18,155

  Resultado es 129 -40i

  número minimo de operaciones de 6 con 3 sumas y 3 multiplicaciones.

Para este caso de la segunda derivada la función nos da como resultado un vector con los coeficientes del polinomio ya derivados por segunda vez.

- **Entradas**:

  Coeficientes = 1,-5,-9,155,-250

  $x_0 = 1 + i$

- **salidas**:

  Coeficientes = 12,-30,-18

  Resultado es -48-6i

  número minimo de operaciones de 4 con 2 sumas y 2 multiplicaciones.

1.2.1. Algoritmo Evación de la primera y segunda derivada en números Imaginarios

```
x0 <- complex(real = -1, imaginary = 1)
#Valor con el que se van a evaluar las derivadas
```

1.2.2. Explicacion algoritmo

Si queremos obtener un resultado de la ecuación pero en números complejos se puede usar la misma ecuación con la única diferencia de cambiar el valor inicial a reemplazar y ponerlo en números complejos de la siguiente manera;

Para el caso 1 (Respuesta Real) debemos declarar el valor a reemplazar de la siguiente manera:

```
X0<-cualquier número real
```

Para el caso de los números complejos se debe declarar de la siguiente forma:

```
X0<- complex(real=num, imaginary=num)
```

Siendo num cualquier número real.
Esa es la única diferencia ya que en el resto del proceso se mantiene el mismo código

1.2.3. Resultados

Resultados de primera y segunda derivada con un valor de $x_0 = 1 + i$:

```
Primera derivada:
Los coeficientes de la derivada son 4 -15 -18 155
El resultado de la evaluacion en 1+1i es: 129-40i
El numero de operaciones realizadas fue de: 6
-----------------------------------------------------
Segunda derivada:
Los coeficientes de la derivada son 12 -30 -18
El resultado de la evaluacion en 1+1i es: -48-6i
El numero de las operaciones realizadas fue de: 4
```

Resultados de primera y segunda derivada con un valor de $x_0 = 2 + 2i$:

```
Primera derivada:
Los coeficientes de la derivada son 4 -15 -18 155
El resultado de la evaluacion en 2+2i es: 55-92i
El numero de operaciones realizadas fue de: 6
-----------------------------------------------------
Segunda derivada:
Los coeficientes de la derivada son 12 -30 -18
El resultado de la evaluacion en 2+2i es: -78+36i
El numero de las operaciones realizadas fue de: 4
```

1.3. Utilizar los resultados anteriores y el algoritmo de Laguerre para obtener un metodo de, Newton- Horner, de convergencia cuadratica en el que el algoritmo de Newton reemplaza al de Laguerre.

1.3.1. comparacion con algoritmo de Laguerre

Si consideramos un caso especial en el cual existe un polinomio de grado n en el que su raiz es x= r y las n-1 restantes es una raíz multiple de x=q, podriamos expresar el polinomio de la siguiente manera:

$$p_n(x) = (x - r)(x - q)^{n-1}$$

Si calculamos su derivada obtenemos:

$$p'_n(x) = (x-q)^{n-1} + (n-1)(x-r)(x-q)^{n-2} = p_n(x)\left(\frac{1}{x-r} + \frac{n-1}{(x-q)^2}\right)$$

es decir:

$$\frac{p'_n(x)}{p_n(x)} = \frac{1}{x-r} + \frac{n-1}{x-q}$$

Luego calculamos la segunda derivada:

$$\frac{p''_n(x)}{p_n(x)} - \left[\frac{p'_n(x)}{p_n(x)}\right]^2 = -\frac{1}{(x-r)^2} - \frac{n-1}{(x-q)^2}$$

Introducimos la anotacion:

$$g(x) = \frac{p'_n(x)}{p_n(x)} = \frac{1}{x-r} + \frac{n-1}{x-q}$$

$$h(x) = g^2(x) - \frac{p''_n(x)}{p_n(x)} = -\frac{1}{(x-r)^2} - \frac{n-1}{(x-q)^2}$$

Si despejamos de la primera funcion x – q y lo sustituimos en la segunda, obtenemos la función cuadrática de x – r, tambien llamada fórmula de Laguerre:

$$x - r = \frac{n}{g(x) \pm \sqrt{(n-1)[nh(x) - g^2(x)]}}$$

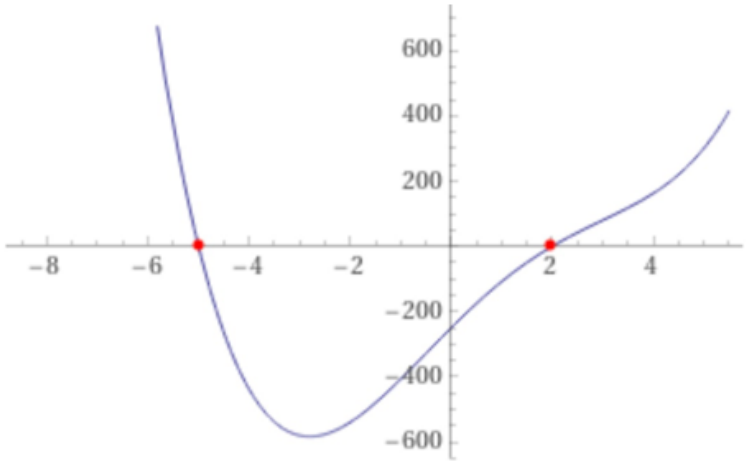
1.3.2. Algoritmo

Algorithm 2 comparacion con el algoritmo de Horner

```
p<-function (x) x^4-5*x^3-9*x^2+155*x-250
x0<-1

function (p, x0, nmax = 25, tol = .Machine$double.eps^(1/2))
{
  n <- length(p) - 1
  p1 <- polyder(p)#ecuación derivada 1
  p2 <- polyder(p1)#ecuación derivada 2
  y0 <- polyval(p, x0)

  if (abs(y0) < tol)
    return(x0)
  for (m in 1:nmax) {
    a <- polyval(p1, x0)/y0
    a2 <- a^2
    b <- a2 - polyval(p2, x0)/y0
    x <- x0 - n/(a + a/abs(a) * sqrt((n - 1) * (n * b - a2)))
    y <- polyval(p, x)
    if (abs(y) < tol)
      break
  }
  return(x)
}
cat("\nx0:",x0)
cat("\np(x0)",p(x0))#valor del polinomio sin derivar
```



2. 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). En su acepción moderna fue formulado en 1973 por Richard Peirce Brent, Australia, 1946.El método de Brent, también conocido como zeroin, ha sido el método más popular para encontrar ceros de funciones desde que se desarrollo en 1972. Este metodo suele converger muy rápidamente a cero; para las funciones difíciles ocasionales que se encuentran en la práctica.

2.1. Aplicar el algoritmo de Brent para encontrar las raíces del polinomio:

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

Explicacion teorica
El método de Brent es un algoritmo basado en los puntos clave de otros métodos como lo son la bisección y la secante. Este fue formulado por Richard Pierce Brent en 1973.

Este consiste o se aplica a un determinado intervalo [a,b] en donde la función cuenta con signos distintos en dichos extremos. En este intervalo se sigue la pista a un punto Xi, que es el mejor en sentido del error.

Posteriormente aplica una interpolación cuadrática inversa, es decir, no $y = f(x)$ sino $x = f(y)$ a los puntos del intervalo y a $x_i = (f(x_i), x_i), (f(a_i), a_i) y (f(b_i), b_i)$. Todo esto con el fin del reemplazar uno de estos tres puntos con aquel donde $x = p(y = 0)$.

2.1.1. Algoritmo

Algorithm 3 Algoritmo de brent

```
f <- function(x) x^3-2*x^2+4*x/3-8/27 #Ecuación
 a<-0 #Límite inferior
 b<-1 #Límite superior
 maxiter=50000 #cantidad máxima de iteraciones
 tol=1*10^-15  #toleracia a trabajar

  stopifnot(is.numeric(a), is.numeric(b), length(a)
  == 1, length(b) == 1)
  #nos permite detectar si hay algo incorrecto en los datos
  de entrada

  x1 <- a # límite inferior
  f1 <- f(x1) #Valor del polinomio evaluado en el límite inferior

  x2 <- b# límite inferior
  f2 <- f(x2) #Valor del polinomio evaluado en el límite superior

  if (f1 * f2 > 0)
  #condición para saber si hay raíz dentro de dicho intervalo
    stop("No hay ninguna raíz en los intervalos planteados [a, b].")
  #Si no hay raíz no se va a seguir con el codigo

  x3 <- 0.5 * (a + b) #Punto medio del intervalo
  niter <- 1

  while (niter<=maxiter) {
    f3 <- f(x3)#evaluación de la función en el punto medio

    if (abs(f3) < tol) {
        x0 <- x3
        break
      }
    if (f1 * f3 < 0)
        b <- x3
    else a <- x3

    if ((b - a) < tol * max(abs(b), 1)) {
      x0 <- 0.5 * (a + b)
      break
    }

    denominador <- (f2 - f1) * (f3 - f1) * (f2 - f3)
    numerador <- x3 * (f1 - f2) * (f2 - f3 + f1) + f2 * x1 *
    (f2 - f3) + f1 * x2 * (f3 - f1)
                                12
    if (denominador == 0) {
      dx <- b - a
    }
```

```
    else {
      dx <- f3 * numerador/denominador
    }

    x <- x3 + dx
    if ((b - x) * (x - a) < 0) {
      dx <- 0.5 * (b - a)
      x <- a + dx
    }

    if (x1 < x3) {
      x2 <- x3
      f2 <- f3
    }

    else {
      x1 <- x3
      f1 <- f3
    }

    niter <- niter + 1
    if (abs(x - x3) < tol) {
      x0 <- x
      break
    }
    x3 <- x
  }

  if (niter > maxiter)
    cat("Se ha alcanzado el númeor máximo de iteraciones")

   prec <- min(abs(x1 - x3), abs(x2 - x3))


  cat("\nLos resultados obtenidos por el método de Brent
  son los siguientes:")

  cat("\nValor de la raíz",x0)
  cat("\nFuncion evaluada en la raíz",f(x0))
  cat("\nPrecisión obtenida",prec)
  cat("\nNúmero de iteraciones",niter)
```

2.1.2. Explicación Algoritmo

El algoritmo de Brent fue utilizado para obtener el valor de una raíz, además de su respectivo valor evaluado en la función y la precisión del proceso en un número de iteraciones.

Los datos de entrada de este son los siguientes:

1. La función (polinomio)
2. El límite inferior del intervalo
3. El límite superior del intervalo
4. El número máximo de iteraciones
5. La tolerancia permitida

Inicialmente se realiza una comprobación de los datos para ver si no hay ningún tipo de irregularidad como por ejemplo que los límites sean datos numéricos.

Posteriormente se realiza la evaluación a cada uno de los límites en la función ya que estas por determinadas condiciones van a ir cambiando.

Adicionalmente se plantea una condición que nos permite saber si hay o no una raíz en ese intervalo definido la cual es realizando un producto entre ambas funciones; es decir, $f(a) * f(b)$.

Ya conociendo que dentro de ese intervalo hay un raíz procedemos a declarar un punto medio el cual se hace básicamente sumando los dos límites y dividiéndolos entre 2; con ese valor también evaluaremos la función ($f(c)$).

Ya teniendo los tres puntos creamos un while que se va a desarrollar hasta que el contador de iteraciones alcance o sea igual al número máximo establecido al inicio lo que nos dará a conocer que en ciclo ha terminado. Del mismo modo este ciclo se puede cortar si el la resta entre x3 y x es menor que la tolerancia lo que nos dará a entender que ya cuenta con la precisión que deseamos.

Dentro de este ciclo se plantean varios condicionales como los siguientes:

```
1) if(abs(f3)<tol)
2) if(f1*f3<0)
3) if ((b - a) < tol * max(abs(b), 1))
4) if ((b - x) * (x - a) < 0)
5) if (x1 < x3)
6) if (abs(x - x3) < tol)
```

Estos condicionales dentro del ciclo irán cambiando los valores de x1,x2 y x3 y sus respectivas funciones lo que finalizará con un resultado de la raíz denotado

en el código como x0. Este valor se evaluará en el polinomio lo cual nos debe dar un resultado igual o muy próximo a cero. Si esto ocurre nos dará a entender que la aplicación fue hecha correctamente.

Del mismo modo se imprimirán los valores de precisión y número de iteraciones requeridas.

2.1.3. Resultados

El resultado obtenido con el metodo de Brent

```
Los resultados obtenidos por el metodo de Brent son
los siguientes:
valor de la raiz 0.6666581
Funcion evaluada en la raiz -6.661338e-16
Precision obtenida 2.314416e-06
Numero de iteraciones 36
```

3. Optima aproximación Polinómica

La implementación de punto flotante de una función en un intervalo a menudo se reduce a una aproximación polinómica, siendo el polinomio típicamente proporcionado por el algoritmo Remez. Sin embargo, la evaluación de coma flotante de un polinomio de Remez a veces conduce a cancelaciones catastróficas. El algoritmo Remez es una metodología para localizar la aproximación racional minimax a una función. La cancelaciones que también hay que considerar en el caso de del método de Horner, suceden cuando algunos de los coeficientes polinómicos son de magnitud muy pequeña con respecto a otros. En este caso, es mejor forzar estos coeficientes a cero, lo que también reduce el recuento de operaciones. Esta técnica, usada clásicamente 1 de 2 para funciones pares o impares, puede generalizarse a una clase de funciones mucho más grande.

3.1. Aplicar esta técnica de aproximación polinómica, para $f(x) = e^{senx-cosx^2}$ en el intervalo$[2^{-8}; 2^{-8}]$ con una precisión deseada doble- doble y un error menor de 2^{-90} y comparar con la aproximación de Taylor.

Explicacion teorica
Podemos decir que el algoritmo de Remez se usa para producir un polinomio$P(x)$ que se aproxima a una función $f(x)$ en un intervalo dado. De hecho, es un algoritmo iterativo que converge a un polinomio que tiene una función de error con n+2 niveles extrenos. Es importante considerar que, el algoritmo de Remez utiliza la posibilidad de que se puede construir un polinomio de grado n-énesimo que conduce a niveles y valores de errores alternos, dados los n+2 puntos de

referencia.

Teniendo en cuenta los puntos de referencia dados n+2 $x_1, x_2, \ldots, x_{n+2}$, donde $x_1 y x_{n+2}$ pueden ser considerados como los puntos finales del intervalo que se esta aproximando, podemos resolver las siguientes ecuaciones:

$$P(x_1) - f(X_1) = +\varepsilon$$
$$P(x_2) - f(X_2) = -\varepsilon$$
$$P(x_3) - f(X_3) = +\varepsilon$$
$$P(x_{n+2}) + F(X_{n+2}) = \pm\varepsilon$$

Dado que $x_1, x_2, \ldots, x_{n+2}$ son datos dados, se conocen sus potencias, por lo que $f(X_1), f(X_2), ..., f(X_{n+2}$ también se conocen.

3.1.1. Algoritmo

Algorithm 4 Algoritmo tecnica de aproximacion polinomica

```
library(pracma)#para métodos de soluciones

f <- function(x){ #para crear la ecuación a utilizar
  return (exp(sin(x)-cos(x^2)))
}

evaluar <- function(x, pol){

#esta función será la que me permita evaluar en taylor,
obtener el resultador del polinomio y el respectivo error

  grado <- length(pol)-1
  aux <- 0
  cont <- 0

  for(i in pol){
    aux <-  aux + i*(x^grado)
    grado <- grado - 1
    cont <- cont + 2

  }
  cat("\n-------------------------------------")
  return (cat("\nEvaluado en Taylor:", aux,"\n",
    #valor obtenido por taylor
    "Evaluado en la ecuación:",f(x) ,"\n",
    #valor obtenido de la ecuación
    "Número de operaciones:", cont, "\n",
    #número de operaciones
    "Error Relativo:",abs(aux-f(x)),"\n"))
    #Error obtenido
}

pol <- taylor(f,2,3) #con ayuda de la biblioteca pracma
limites <- (2^-8-(-2^-8))  #los límites superior e inferior
x <- 2  #valor que se reemplazará en la función
para encontrar su valor

puntos <- c()#número de valores a
for(i in 1:10){
  puntos <- c(puntos, x)
  x <- x + limites
}

for(i in puntos){
  cat(evaluar(i, pol))
}

plot(f, xlim = c(-10,10),ylim=c(0,10), col="red", ylab = "f(x)")
plot(f, xlim = c(-2^-8,2^-8),ylim=c(-4,4), col="red", ylab = "f(x)")
```

3.1.2. Explicación del algoritmo

En este punto lo que se busca como objetivo principal es realizar una comparación entre dos opciones de encontrar el valor de una ecuación al momento en el que las variables son reemplazadas por un número real; Este código inicia declarando un límite máximo y mínimo el cual ya fue propuesto en el enunciado del ejercicio.

Posteriormente se creará un vector el cual su tamaño tiene que ver con la cantidad de particiones a realizar en ese intervalo; del mismo modo cada una de esas particiones entrará a ser evaluada en el mismo ciclo en el que se obtendrá el polinomio de Taylor.

Durante el ciclo se va a obtener el valor de Taylor de la siguiente forma:

```
for(i in p){
    aux <-  aux + i*(x^grado)
    grado <- grado - 1
    cont <- cont + 2
  }
```

En donde p¡-taylor (f, 1, 3)

Siendo f la ecuación a evaluar $(exp(sin(x) - cos(x^2)))$.

Al finalizar este ciclo ya tendremos el valor de Tyler el cual lo compararemos con el respectivo valor de la ecuación f en un punto la cual es únicamente reemplazando cada valor de X con dicho número.

Finalmente para la obtención del error se realiza la respectiva resta entre el valor de Taylor y el obtenido en la ecuación.

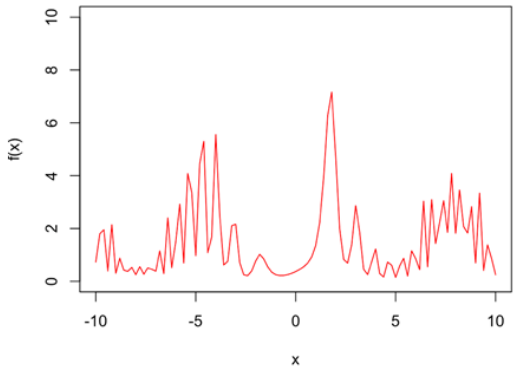
3.1.3. Resultados

De la tabla podemos explicar que los valores obtenidos con la evaluación de Taylor y por la evaluación directa de la ecuación son bastante similares, obteniendo errores por debajo de 1 porciento. Sin embargo, somos consientes de que estos valores pueden ser mejorados utilizando precisión doble y un mayor uso de cifras significativas.

Evaluación en Taylor	Evaluación en la ecuación	Error obtenido
0,9099253	0,9179485	0,008
0,9112329	0,9191339	0,007901012
0,912542	0,9203222	0,007780234
0,9138527	0,9215135	0,007660826
0,915165	0,9227078	0,007542777
0,916479	0,9239051	0,007426077
0,9177946	0,9251053	0,007310718
0,9191119	0,9263086	0,007196687
0,9204309	0,9275149	0,007083977
0,9217516	0,9287241	0,006972577

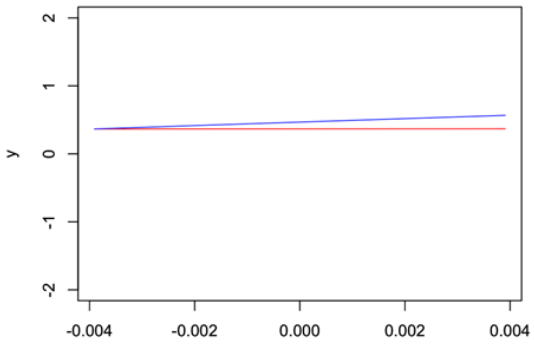
Tabla 1

Grafica de la función dada:



grafica 1

Remez vs ecuacion:



grafica2

4. Conclusiones

4.1. Conclusión Punto 1:

Del desarrollo del primer punto se puede concluir que el algoritmo implementado es eficiente y adecuado para la solución del problema sea para la parte real o la imaginaria debido a que en ambos casos calculaba las derivadas correspondientes para arrojar el valor correcto; sin embargo, a la hora de compararlo con el método de Laguerre se encontró un problema en el momento de imprimir el valor de las derivadas finales ya que no nos era posible y solo se pudo imprimir el valor del polinomio inicial.

```
x0:1
p(x0)-108
```

4.2. Conclusión Punto 2:

Con respecto al punto 2 y la implementación del método de Brent se pudo ejercutar de una forma correcta en donde el valor resultante nos daba los que en un principio se esperaba; del mismo modo se sabía que la raíz era correcta ya que al momento de evaluar ese valor en la función nos daba 6.661338e-16 lo cual es un valor muy cercano a cero. Por otro lado podemos percibir que este método requiere de un número mayor de iteraciones en comparación al de newton; sin embargo como grupo podemos decir que esté método es mucho mas efectivo que el de posición falsa el cual nostocó presentar en una de las clases de análisis numérico. Esto se puede decir debido a que para que la implementación del método de posición falsa geerara un resultado aproximado a 2/3 requirió de más de 100 iteraciones mientras que el de Brent en 36 repeticiones nos generó un resultado mucho más aproximado y con un error más pequeño.

4.3. Conclusión Punto 3:

Con respecto a este pnto se presentaron varios problemas debido a la complejidad de la función. En ocasiones nos generó que los errores relativos no fieran todos iguales debido a que no nos fue posible aumentar la presición a el valor de 2^{-90}. Sin embargo, con los valores obtenidos pudimos hacer un análisis adecuado con respecto a los valores que nos arrojaba la evaluación con taylor y la evaluación directa en la ecuación. De esto podemos conlcuir que se hubieran obtenido unos mejores valores que nos hubieran permitido generar un análisis óptimo.

4.4. Conclusión final:

El trabajo realizado en este reto de análisis numérico a pesar de que se presentaron inconvenientes con respecto a la presición fue posible realizar un análisis correcto que nos permitió desarrollar los diferentes algoritmos y generar respuestas coherentes a lo que se esperaba desde un principio. A pesar de que se podían mejorar estos resultados con respecto a la exactitud nos dieron muy cercanos.

Bibliografia

Faires, B. y. (2011). Análisis Numérico. Ciudad de México: Cengage Learning.

Schnabel, D. y. (1996). Numerical Methods for uncontrained optimization and non lineal equations. No especifica: SIA.

Saleri, S. y. (2000). Numerical Mathematics . New York: Springer.

OConnor, J. L. (Septiembre de 2017). Ingeniería de los algoritmos y métodos numéricos España.