Pontificia Universidad Javeriana

Análisis numérico

Reto 1

Autores:
John Stiven
Jimmy Alejandro
Pablo Veintemilla
Camilo Maldonado

Septiembre 2020

RETO 1

EDDY HERRERA

Resumen. Presentamos aca el metodo de blah blah

Parte 1. Metodo de Newton-Horner y Metodo de Laguerre

1. Metodo de Newton-Horner

El método de Newton-Horner y el método de Laguerre son métodos iterativos usados para dar con la solución de ecuaciones polinómicas, son métodos que aseguran una convergencia rápida hacia las soluciones aplicando teorías de polinomios.
Método de Newton-Horner:

Dado un polinomio de grado n

$$P(x) = \sum_{j=0}^{n} a_j x^j$$

se puede utilizar el método de Horner para factorizar en un valor x0 usando la división sintética así obtener un factor de la forma

$$P(x) = (x - x_0) \sum_{j=0}^{n-1} b_j x^j + h_1$$

Donde h1 es un sumando adicional para completar la factorización. Así pues se puede afirmar que

$$P(x_0) = h_1$$

Y de esta forma igualmente en P'(x)

$$P'(x) = \frac{d}{dx}\left((x - x_0) \sum_{j=0}^{n-1} b_j x^j + h_1 \right)$$

Date: octubre 1 de 2020.

Se sabe que la derivada de h1 es 0, la derivada de (x – x0) es 1 y la derivada de $_(j = 0)^(n - 1) b_j x^j$ $seriaunpolinomiodegradon-2alquesepuedellamarG(x)aplicandoladerivadadelproductoenP(x)setieneque$

$$P'(x) = \sum_{j=0}^{n-1} b_j x^j + (x - x_0)G(x)$$

Se puede realizar la misma división sintética con centro en x0 sobre el polinomio

$$(x - x_0) \sum_{j=0}^{n-2} c_j x^j + h_2$$

Entonces al evaluar P'(x) en x0

$$P'(x_0) = h_2$$

Lo cual sería igual que solo aplicar la división en el polinomio de grado n-1 y extraer el nuevo sumando h2 .Conociendo el valor de P(x) y P'(x) se puede usar el método iterativo de Newton-Raphson teniendo el valor de x0

$$x_{k+1} = x_k - \frac{h_1}{h_2}, k = \{0, 1, 2, \dots\}$$

Se tiene un nuevo valor de x con el que se puede hacer una nueva iteración hasta hallar la solución.

Método de Laguerre:

Dado un polinomio

$$P(x) = \sum_{j=0}^{n} a_j x^j$$

se debe proponer una solución x = r y unas n-1 soluciones en x = q de tal forma que

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

De esto podemos derivar P(x) y obtener

$$P'(x) = (x-q)^{n-1} + (x-r)(n-1)(x-q)^{n-2}$$

Y se puede decir que

$$P'(x) = P(x)\left(\frac{1}{x-r} + \frac{n-1}{x-q}\right)$$

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

Si derivamos esta expresión de nuevo

$$\frac{P''(x)}{P(x)} - \left(\frac{P'(x)}{P(x)}\right)^2 = -\frac{1}{(x-r)^2} - \frac{n-1}{(x-q)^2}$$

Podemos sustituir

$$G(x) = \frac{1}{x-r} + \frac{n-1}{x-q}$$

$$H(x) = \frac{1}{(x-r)^2} + \frac{n-1}{(x-q)^2}$$

Podemos despejar (x-q) de las dos ecuaciones y poderlo igualar, a lo cual se obtiene una ecuación cuadrática para (x-r) la cual se soluciona

$$x - r = \frac{n}{G \pm \sqrt{(n-1)(nH - G^2)}}$$

Para obtener el método iterativo de Laguerre

$$x_{k+1} = x_k - \frac{n}{G \pm \sqrt{(n-1)(nH - G^2)}}, \qquad k = \{0, 1, 2, \dots\}$$

$$G = \frac{P'(x)}{P(x)}, \qquad H = G^2 - \frac{P''(x)}{P(x)}$$

Los métodos están enfocados en asegurar la menor perdida de significancia y poder encontrar las diferentes raíces de un polinomio con convergencia teórica cubica.

Raíces Reales y Complejas:

Tanto el método de Newton-Horner como el Método de Laguerre son capaces de estimar de forma precisa una solución tanto dentro de los números reales como en los números complejos, más aún en la implementación se puede ver que se necesita una aproximación al cero en la parte imaginaria para conseguir estos resultados.

Pruebas:

Se realizaron pruebas tanto con diferentes polinomios como para diferentes valores de tolerancia y cambiando los valores iniciales se puede dar un comportamiento diferente en los resultados en numero de iteraciones como en el comportamiento de los errores graficados.

1. Prueba base: P(x) = $x^4 - 5x^3 - 9x^2 + 155x - 250$

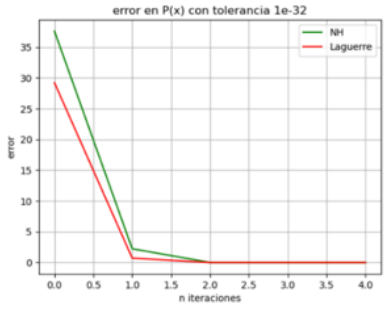
```
polinomios = [[1,-5,-9,155,-250]]
```

Soluciones teoricas

The solutions are

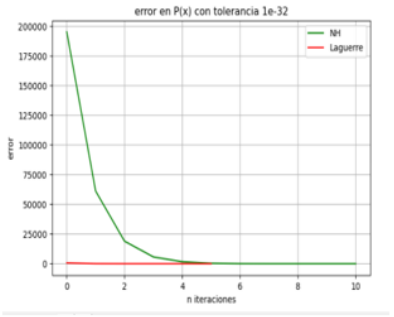
$x = 2, x = -5, x = 4 + 3i, x = 4 - 3i$

X0 = 0, se encuentra la raíz real x = 2



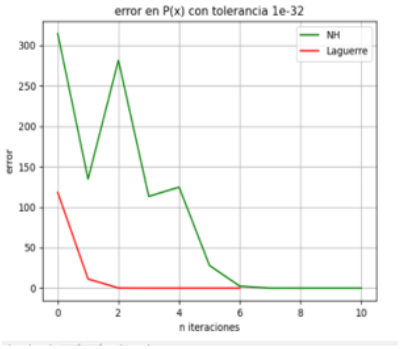

```
Metodo de Newton Horner, tolerancia  1e-32
1  x = 1.612903225806451512625017130631 02+0.00000000000000000000000000000000j
2  x = 1.975625794948043356669131753733 38+0.00000000000000000000000000000000j
3  x = 1.999901921783725722292501814081 33+0.00000000000000000000000000000000j
4  x = 1.999999998414384139167054854624 44+0.00000000000000000000000000000000j
5  x = 2.000000000000000000000000000000 00+0.00000000000000000000000000000000j
P(x*) = 0.000000000000000000000000000000 00+0.00000000000000000000000000000000j

Metodo de Lagurre, tolerancia  1e-32
1  x = 1.694915254237287838279257812246 21+0.00000000000000000000000000000000j
2  x = 1.992192965556836448115518578561 02+0.00000000000000000000000000000000j
3  x = 1.999994974209352349703294748906 05+0.00000000000000000000000000000000j
4  x = 1.999999999997918109784222906455 40+0.00000000000000000000000000000000j
5  x = 2.000000000000000000000000000000 00+0.00000000000000000000000000000000j
P(x*) = 0.000000000000000000000000000000 00+0.00000000000000000000000000000000j
```

X0 = -3, se encuentra la raíz real x = -5




```
Metodo de Newton Horner, tolerancia  1e-32
1  x = -20.058823529411764496899195364676420-0.000000000000000000000000000000090j
2  x = -14.893438265227951333713463100139050-0.000000000000000000000000000000067j
3  x = -11.098735131574294143774750409647820-0.000000000000000000000000000000049j
4  x = -8.3794397496049306539589451858773 8-0.000000000000000000000000000000034j
5  x = -6.5440653216099811118056095438078 0-0.000000000000000000000000000000022j
6  x = -5.4884125484364050188374761112034 3-0.000000000000000000000000000000011j
7  x = -5.0649552309088201339952509442810 0-0.000000000000000000000000000000003j
8  x = -5.0014051188602133279914596641901 9-0.000000000000000000000000000000000j
9  x = -5.0000000676491318074567971052601 93-0.000000000000000000000000000000000j
10  x = -5.000000000000157207580286922166 12-0.000000000000000000000000000000000j
11  x = -5.000000000000000000000000000000 00-0.000000000000000000000000000000000j
P(x*) = 0.000000000000000000000000000000+0.000000000000000000000000000000000j

Metodo de Lagurre, tolerancia  1e-32
1  x = -5.788461538461538324895627738442 27-0.000000000000000000000000000000001j
2  x = -5.092914111852901548616046056848 88-0.000000000000000000000000000000000j
3  x = -5.001456629916395613122404029127 21-0.000000000000000000000000000000000j
4  x = -5.000000363642069345360141596756 88-0.000000000000000000000000000000000j
5  x = -5.000000000000023092638912203256 04-0.000000000000000000000000000000000j
6  x = -5.000000000000000000000000000000 00-0.000000000000000000000000000000000j
P(x*) = 0.000000000000000000000000000000+0.000000000000000000000000000000000j
```

X0 = 3+2.3i, se encuentra la raiz compleja 4+3i




```
Metodo de Newton Horner, tolerancia  1e-32
1  x = 5.17491771961498780996180357760750+1.738719450573680269656051677884 53j
2  x = 3.90913500727568141712788893464878+1.862749119377267614794391192845 07j
3  x = 2.64292923636129861719723521673586+3.470934111256188892014051816695 05j
4  x = 3.32490316501352456413087566033937+2.562683203014800348285007203230 63j
5  x = 4.53409454646934088373066813510376 96+2.801270704241438203041525412118 06j
6  x = 4.02355383046182701889392774319276+2.858361870860972153707280085654 93j
7  x = 3.99108296532828488167865543800872+3.006661901290044180967697684536 69j
8  x = 3.99995743989789787065092241391540+2.999955069876811375451097774202 94j
9  x = 4.00000000159490021189867547946051+3.000000001060174170675054483581 33j
10  x = 4.00000000000000000000000000000000+2.999999999999995559107901499373 8j
11  x = 4.00000000000000000000000000000000+3.000000000000000000000000000000 00j
P(x*) = 0.000000000000000000000000000000+0.000000000000000000000000000000000j

Metodo de Lagurre, tolerancia  1e-32
1  x = 4.47203884501401738305048638721928+3.114040620037691553534386912360 79j
2  x = 4.04818139478704619449445090140216+2.975565890167819915745894832070 92j
3  x = 3.99972126388142079633780667791143+2.999326155645198888806671588099 562j
4  x = 4.00000003314203311788332939613611+3.000000128768229412656864951713 94j
5  x = 4.00000000000000000000000000000000+3.000000000000044408920985006266 16j
6  x = 4.00000000000000000000000000000000+2.999999999999995559107901499373 8j
7  x = 4.00000000000000000000000000000000+3.000000000000000000000000000000 00j
P(x*) = 0.000000000000000000000000000000+0.000000000000000000000000000000000j
```

2. Otras pruebas: P1(x) = $x^2 + 4, P2(x) = x^2 - 5x + 6$

```
polinomios = [[1,0,4],[1,-5,6]]
```

P1(x): $x^2 + 4 tiene una raiz compleja y conjugada x = 2i$

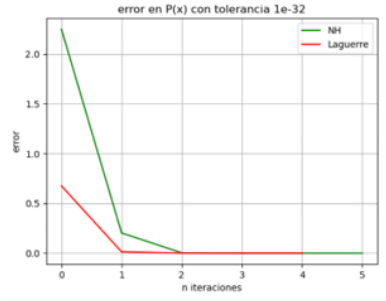
X0 = 0, se obtiene el valor de la raíz, pero se necesitan muchas iteraciones para la tolerancia mínima


```
105  x = 0.000000000000000000000000000000+9.995607995119579314291513583157 21j
106  x = 0.000000000000000000000000000000+5.197891876253764031901027919957 41j
107  x = 0.000000000000000000000000000000+2.983717312294334789157801424153 15j
108  x = 0.000000000000000000000000000000+2.162163443989852495974446355830 88j
109  x = 0.000000000000000000000000000000+2.006081173613342816253179989871 57j
110  x = 0.000000000000000000000000000000+2.000009217142606932071657865890 30j
111  x = 0.000000000000000000000000000000+2.000000000021238566461079244618 30j
P(x*) = -0.000000000884954265844316978473 22+0.000000000000000000000000000000000j

Metodo de Lagurre, tolerancia  1e-08
1  x = 0.000000000000000000000000000000+2.828427124746189846860033867415 04j  e
2  x = 0.000000000000000000000000000000+2.070552368820165915811230661347 51j  e
3  x = 0.000000000000000000000000000000+2.000611374255583552184134532581 08j  e
4  x = 0.000000000000000000000000000000+2.000000046715169510491705295862 63j  e
5  x = 0.000000000000000000000000000000+2.000000000000044408920985006262 j  e
P(x*) = -0.000000000000001776356839400250 46+0.000000000000000000000000000000000j
```

X0 = i se obtiene la solución con un número menor de iteraciones

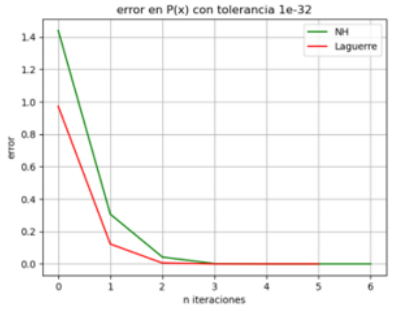
$P2(x) = x^2 - 5x + 6 = (x-2)(x-3), tienedosracesrealesx = 2yx = 3$

X0 = 0, se encuentra la raíz x = 2 o un valor muy aproximado




```
Metodo de Newton Horner, tolerancia  1e-32
1  x = 0.000000000000000000000000000000+2.500000000000000000000000000000j
2  x = 0.000000000000000000000000000000+2.049999999999998223643160599749 5j
3  x = 0.000000000000000000000000000000+2.000609756097560865129025842179 54j
4  x = 0.000000000000000000000000000000+2.000000092922294747666001057950 78j
5  x = 0.000000000000000000000000000000+2.000000000000022204460492503130 8j
6  x = 0.000000000000000000000000000000+2.000000000000000000000000000000j
P(x*) = 0.000000000000000000000000000000+0.000000000000000000000000000000j

Metodo de Lagurre, tolerancia  1e-32
1  x = 0.000000000000000000000000000000+2.162277660168379078697853401536 12j
2  x = 0.000000000000000000000000000000+2.003162215682247282444450320326 72j
3  x = 0.000000000000000000000000000000+2.000001248963434896666058193659 41j
4  x = 0.000000000000000000000000000000+2.000000000001949551631241774885 0j
5  x = 0.000000000000000000000000000000+2.000000000000000000000000000000j
P(x*) = 0.000000000000000000000000000000+0.000000000000000000000000000000j
```

X0 = 4, se encuentra la raíz x = 3



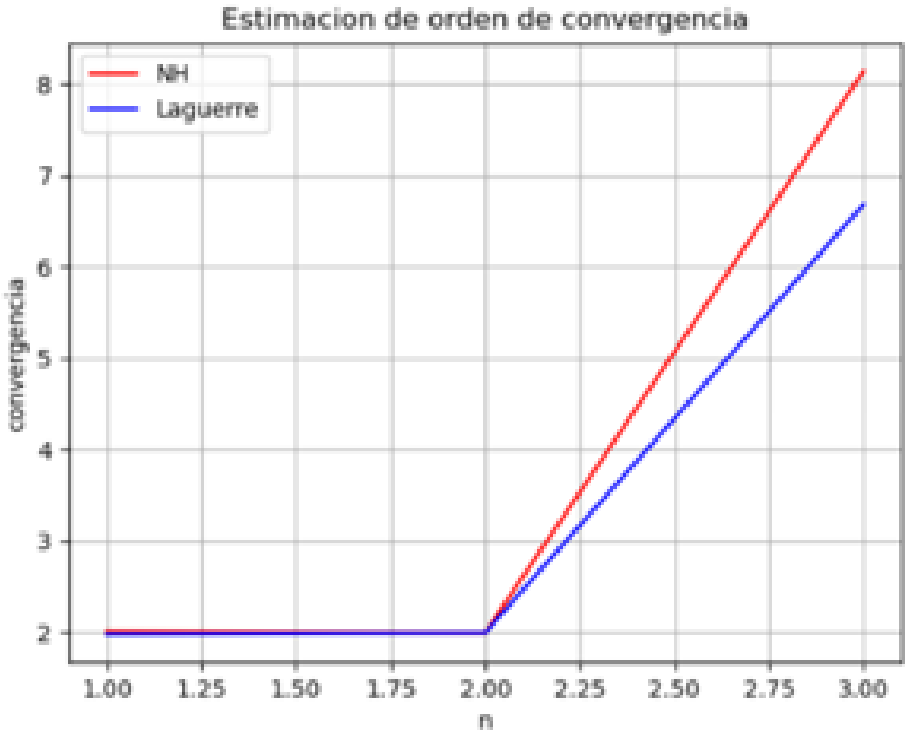

```
Metodo de Newton Horner, tolerancia  1e-32
1  x = 1.199999999999999955591079014993 74+0.000000000000000000000000000000j
2  x = 1.753846153846154054534167688755 4+0.000000000000000000000000000000j
3  x = 1.959397303727200112533068931952 58+0.000000000000000000000000000000j
4  x = 1.998475239805509673374217527452 86+0.000000000000000000000000000000j
5  x = 1.999997682174605273530687554739 42+0.000000000000000000000000000000j
6  x = 1.999999999994628074873048717563 51+0.000000000000000000000000000000j
7  x = 2.000000000000000444088920985006 262+0.000000000000000000000000000000j
P(x*) = 0.000000000000000000000000000000-0.000000000000000000000000000000j

Metodo de Lagurre, tolerancia  1e-32
1  x = 1.394448724536010653096695932617 88+0.000000000000000000000000000000j
2  x = 1.889600447100860280968959159508 81+0.000000000000000000000000000000j
3  x = 1.994525353735569161983676167437 81+0.000000000000000000000000000000j
4  x = 1.999985095830263848171171048306 86+0.000000000000000000000000000000j
5  x = 1.999999999888934176794919039821 25+0.000000000000000000000000000000j
6  x = 2.000000000000000000000000000000+0.000000000000000000000000000000j
P(x*) = 0.000000000000000000000000000000-0.000000000000000000000000000000j
```

Orden de convergencia:

Se trato de realizar una grafica sobre las interacciones del error para poder encontrar el orden de convergencia y lo qeue se obtuvo fue lo siguiente

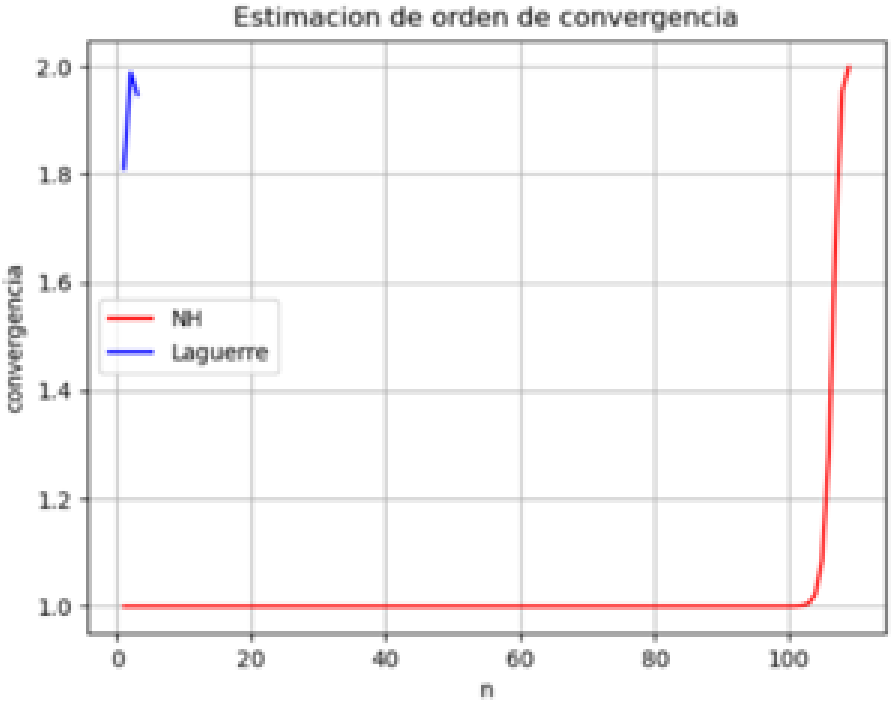
Polinomio de prueba:

En el polinomio de pruebas con la tolerancia minima se puede observar una contante en las primeras 2 iteraciones del valor 'p' del orden de convergencia cercano a 2, pero luego de esto se produce una irregularidad.

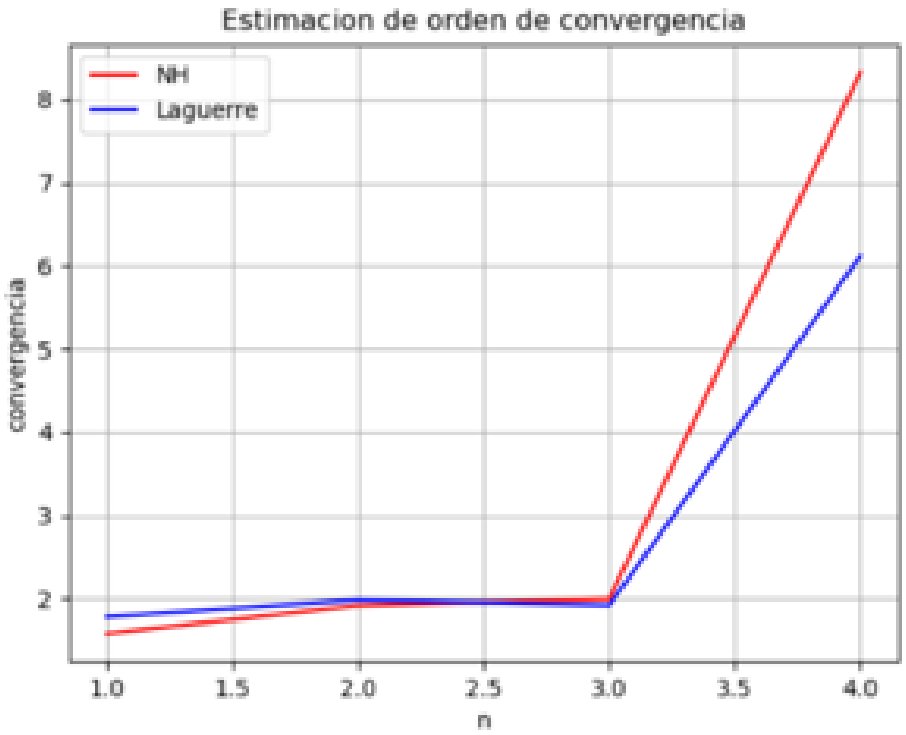


$P(x) = x^2 + 4$

Teniendoencuentaelprcesodondeelalgoritmorealizoelmayornumerodeiteraciones($x0 =$
0)*enelmtododeNewton*$-$*Horner*, *sepuederealizarmsclaramentelaestimacindelordendeconvergencia.Enelmtod*
Hornersepuedeverclaramenteunaconstanteenp $= 1$*hastaelfinaldondeseproducelairregularidad*, *mientrasquee*



$P(x) = x^2 - 5x + 6$
$En el cl culo de las dos races de este polinomio se puede observar el mismo comportamiento de la gr fica, primero semant$



$Comparativaconotrosmtodosnumricosy Conclusiones$:

$La convergencia tericadelosmtodosde Newton-Hornery Laguerrenosaseguraquelosmtodossecomportarandema$
$Raphson, conlarestriccindesolopoderseraplicadosa funcionespolinomiales. Talvezsiseintentarealizarunaestim$

Parte 2. Método de Brent

Parametros Métodos Brent y Taylor:

Los parámetros para estos algoritmos son:

1. Brent
   a) Función: debe ser continua y tener signos diferentes en el rango AB
   b) A: inicio de intervalo
   c) B: fin de intervalo
   d) Tolerancia
   e) Full Output: True, para obtener los datos de la convergencia
   f) Disp: True para mostrar el orden de convergencia
2. Taylor
   a) Función
   b) X: donde se quiere aproximar el polinomio

c) N: el orden del polinomio a aproximar

Significancia

El algoritmo de Brent que nos proporciona Python nos permite hallar resultados con un error mínimo, para el ejemplo, encontramos que nos da una solución, en pocas iteraciones, con un error de hasta 2**-90.

```
brent (0.6666685251964675,       converged: True
           flag: 'converged'
 function_calls: 50
     iterations: 49
           root: 0.6666685251964675)
taylor x/3 - 0.296296296296296
[0.66667125]
>>>
=========== RESTART: C:\Users\pbr98\Desktop\reto1\bren
brent (0.6666678608664627,       converged: True
           flag: 'converged'
 function_calls: 47
     iterations: 46
           root: 0.6666678608664627)
taylor x/3 - 0.296296296296296
[0.66667125]
>>>
```

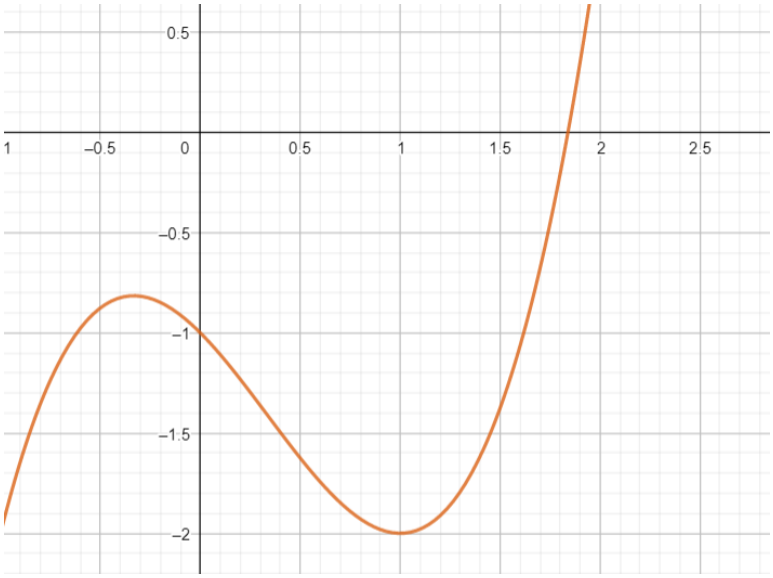
En la imagen anterior vemos dos resultados, donde en el primero se ha utilizado un error de 2**-90 mientras que, en el segundo, la tolerancia fue de 2**-16. Esto no afecta el orden de convergencia a groso modo, pues la cantidad de iteraciones ha disminuido muy poco para lo que aumento el error.
De esto podemos concluir además que el método nos ofrece significancia hasta el 5to o 6to decimal, de ahí en adelante, el error podría aumentar.
En este caso, para hallar la raíz del polinomio de Taylor, se utilizó la función root, la cual no es precisa, por lo que se optó por hallar las raíces con brentq para su comparación.

```
brent (0.6666685251964675,       converged: True
           flag: 'converged'
 function_calls: 50
     iterations: 49
           root: 0.6666685251964675)
taylor 1.33333344365383e-10*x - 8.8888229399231e-11
[ 0.666661665334398 ]
>>>
=========== RESTART: C:\Users\pbr98\Desktop\reto1\bren
brent (0.6666685251964675,       converged: True
           flag: 'converged'
 function_calls: 50
     iterations: 49
           root: 0.6666685251964675)
taylor 1.33333344365383e-10*x - 8.8888229399231e-11
[ 0.666661665334398 ]
```

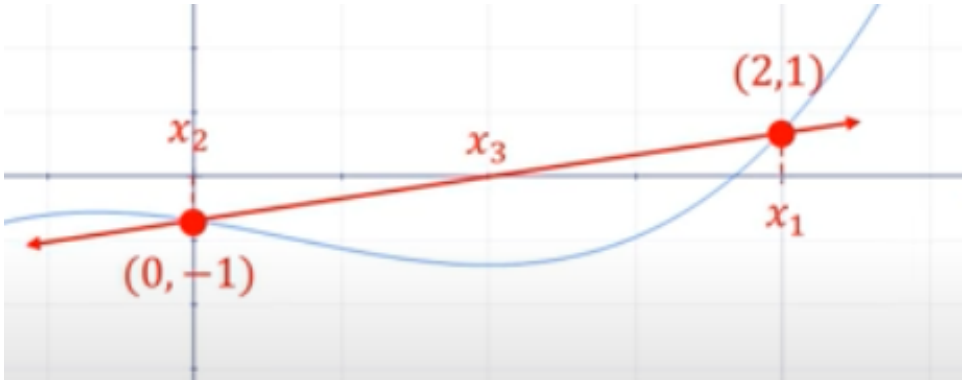
En el caso de Taylor, evaluando el polinomio arrojado con brentq, nos da resultados iguales con error tanto de 2**90 como de 2**-16 y el valor, nos presta las mismas 5 cifras significativas, hasta volverse completamente diferente a las dos respuestas arrojadas por brentq.
El algoritmo de Taylor presenta valores distintos desde el 5 decimal, incluso cambiando el grado del polinomio, el mejor parece ser con el polinomio de grado 4.

Algoritmo de Brent
Créditos a Oscar Veliz, con un video que me explico el método.
Para la siguiente función f(x) la grafica es la que se muestra.

$$f(x) = x^3 - x^2 - x - 1$$



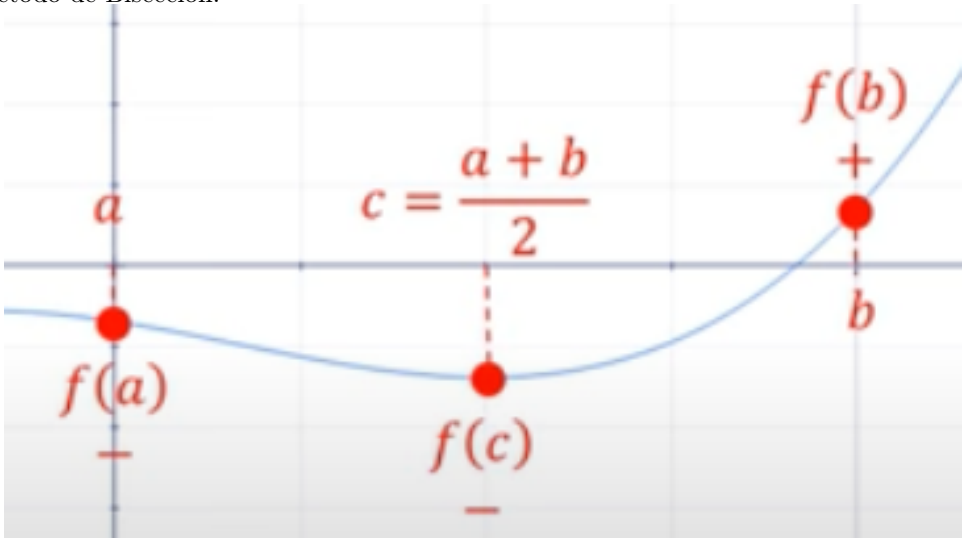
Para poder aplicar el algoritmo, es necesario que la función sea continua y que en el intervalo donde vamos a evaluar la función, f(x) tenga signos contrarios, como vemos en la función, entre 1.5 y 2, hay un cambio de signo, por lo que se puede aplicar este método.
Para explicar este método entonces debemos también explicar los tres métodos que lo componen.
Método de la secante.



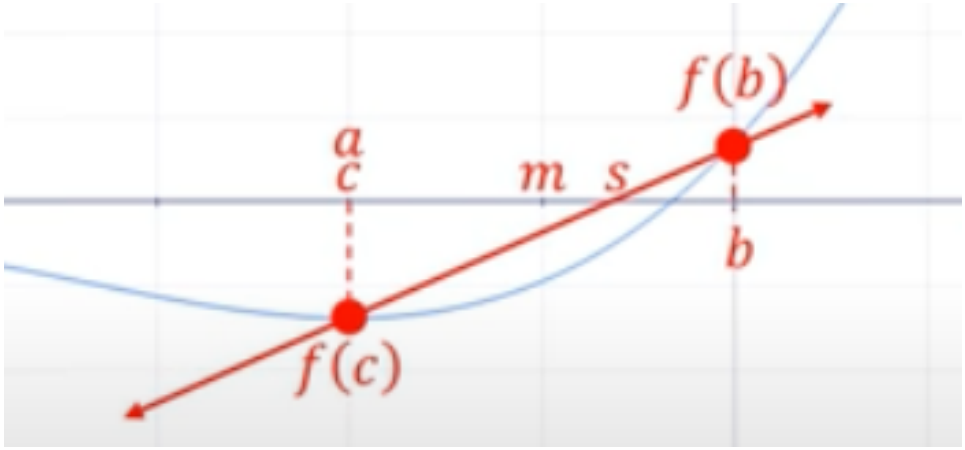
En una función, tomamos dos puntos y dibujamos una secante entre ellos, vemos que toca el eje x en x3 y este se convierte en otro punto, al proyectarlo a la función.

Utilizando x2 y x3, dibujamos una secante también y vemos que se interseca en x4, realizamos el mismo procedimiento y vemos que la secante no se interseca con el eje x por lo que ahí el método tiene un problema y es que en este caso, el método diverge.

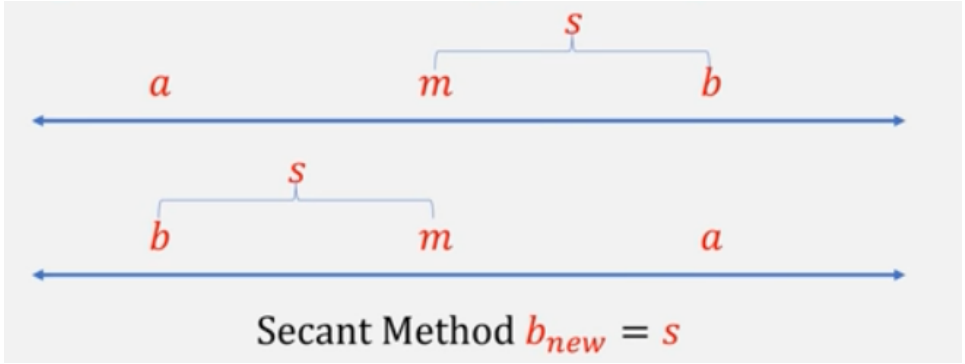
Método de Biseccion.



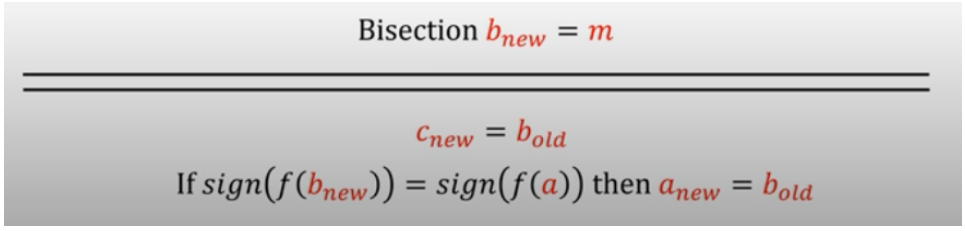
En este método, tomamos de nuevo dos puntos, a y b y sacamos un tercer punto en la mitad. Se repite el procedimiento utilizando c y b ya que el signo eso puesto. Repetimos el proceso hasta llegar al la raiz.
Método de Dekker.
El propone que combinar estos dos últimos métodos, ya que el método de bisección es muy lento pero converge y el de la secante puede diverger. Para esto, dekker dice que entonces, se usara el método de la secante y se cambiara al método de bisección cuando se necesite. Teniendo a y b, con signos distintos, se obtiene m, como a+b/2, se utiliza el método de secante, pero si se divide por 0 entonces s permanecerá siendo m.



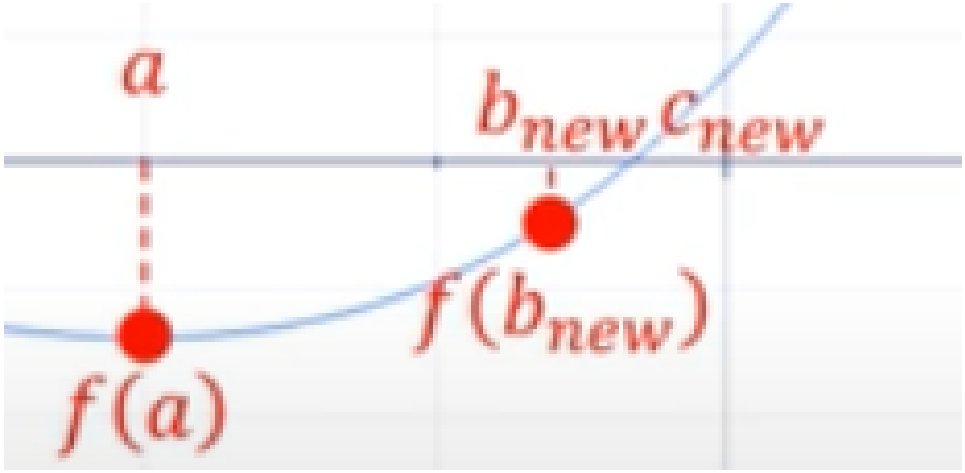
Existen entonces diferentes casos posibles. El primero, es cuando s esta entre m y b, donde el nuevo b será.



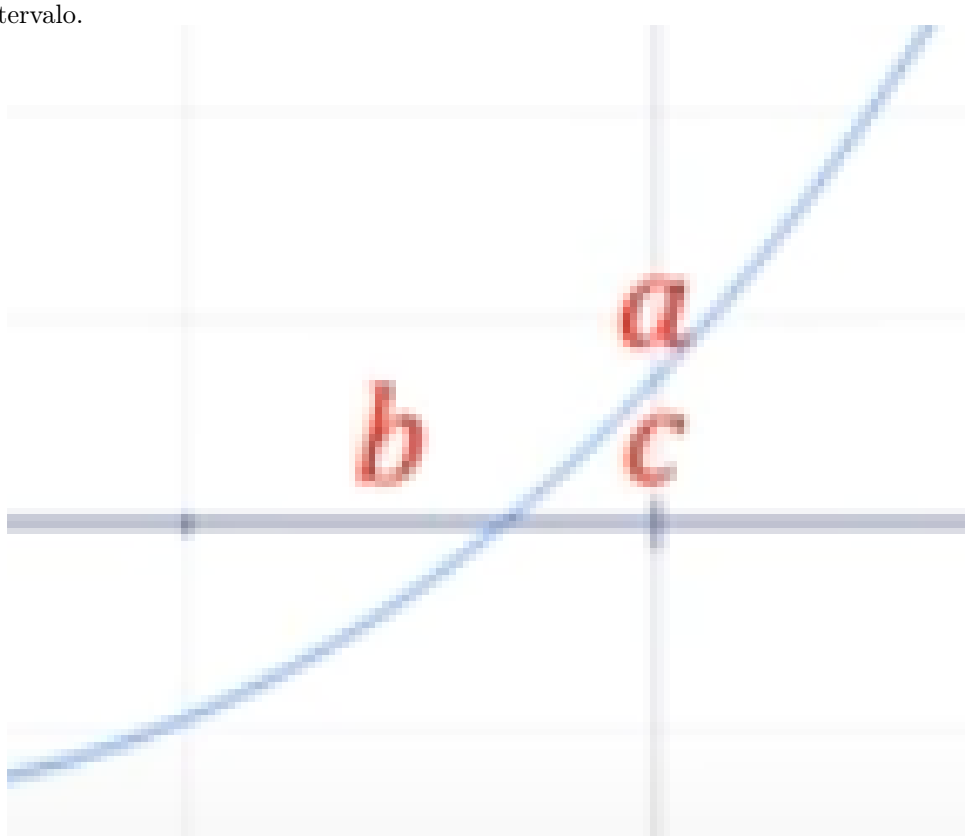
De no ser así, se utilizará bisección y se actualizaran las variables de la siguiente manera.



En nuestro ejemplo, tenemos que s esta entre m y b por lo que b va a ser ahora s y c va a ser nuestro b anterior.



Pero como vemos, cuando validamos los signos vemos que la raiz no esta en el intervalo.



Por ello, decimos que a va a tener el ultimo valor de b, los cuales se van a invertir (a y b) y se va a repetir el proceso.

Aquí va un ejemplo numérico.
Donde utilizamos la función x**3 – x**2 -x -1, empezando en 1 y 2 con un error

de 10**-7 e iteramos hasta que la resta de a-b sea menor que el error, por lo que acaba el algoritmo y b es nuestra respuesta final.

a	b	c	m	s	$f(a)$	$f(b)$	$f(b_{new})$
1	2	1	1.5	1.666666667	-2	1	-0.814814815
2	1.666666667	2	1.833333333	1.81632653061	1	-0.814814815	-0.123230967
2	1.81632653061	1.666666667	1.90816326531	1.84299391123	1	-0.123230967	0.020341595
1.81632653061	1.84299391123	1.81632653061	1.82966022092	1.83921563321	-0.123230967	0.020341595	-0.00038904
1.84299391123	1.83921563321	1.84299391123	1.84110477222	1.83928653794	0.020341595	-0.00038904	-0.000001189
1.84299391123	1.83928653794	1.83921563321	1.84114022459	1.83928675523	0.020341595	-0.000001189	0.00000000006
1.83928653794	1.83928675523	1.83928653794	1.83928664658	1.83928675521	-0.000001189	0.00000000006	0

Método de interpolación inversa.

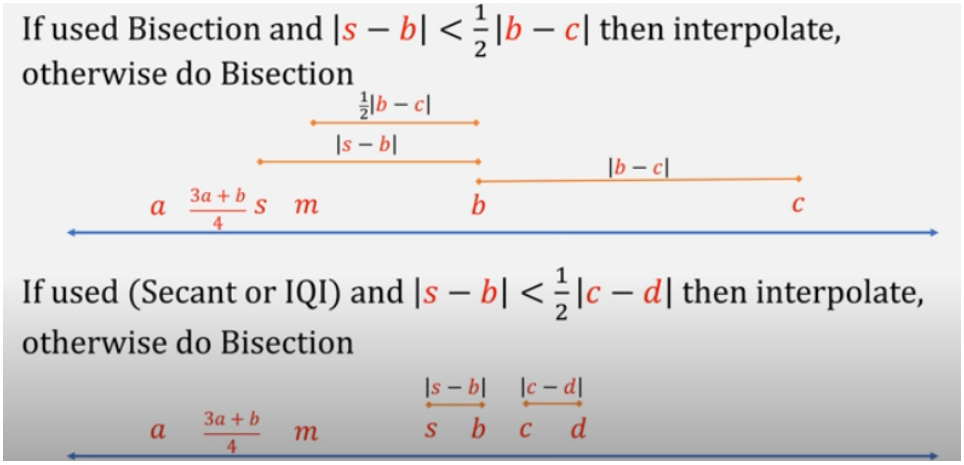
Su método consiste, o añade, una interpolación cuadrática que, de no lograrse, usa secante, y, si aún no así no se consigue, usa bisección. Para no caer con raíces imaginarias, Brent usa cuadrática inversa. Para la interpolación utiliza polinomios de Lagrange, lo que produce una parábola.

- $x_1 = 0, x_2 = 1, x_3 = 2$
- $y_1 = -1, y_2 = -2, y_3 = 1$
- Swap x's for y's
- $L_2(y) = x_1 \frac{y-y_2}{y_1-y_2}\cdot\frac{y-y_3}{y_1-y_3} + x_2 \frac{y-y_1}{y_2-y_1}\cdot\frac{y-y_3}{y_2-y_3} + x_3 \frac{y-y_1}{y_3-y_1}\cdot\frac{y-y_2}{y_3-y_2}$
- $L_2(y) = 0 \frac{y-(-2)}{(-1)-(-2)}\cdot\frac{y-1}{(-1)-1} + 1 \frac{y-(-1)}{(-2)-(-1)}\cdot\frac{y-1}{(-2)-1} + 2 \frac{y-(-1)}{1-(-1)}\cdot\frac{y-(-2)}{1-(-2)}$
- $L_2(y) = \frac{2y^2}{3} + y + \frac{1}{3}$

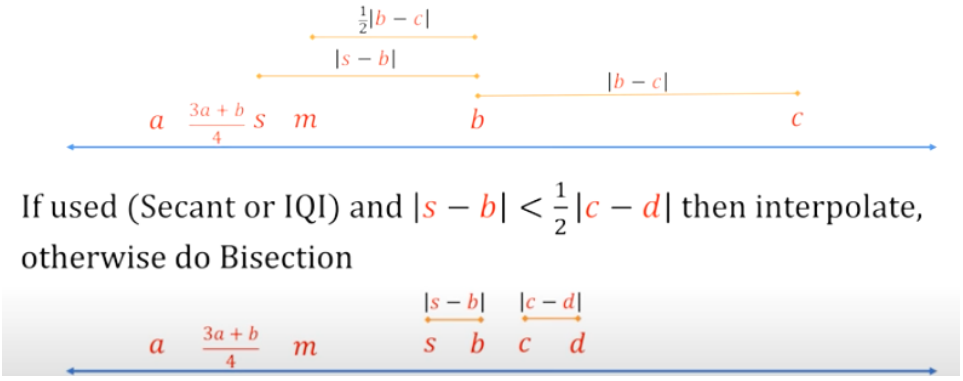
Como interpolación, tiene varios problemas también a la hora de hallar las raíces, se usa de manera prudente en el método de Brent.

Método de Brent.

El método trabaja los mismos valores a,b,c,m y s que venimos hablando. Primero, intenta calcular s con el método de interpolación, si falla, saca la secante, si no, vuelve a bisección. Actualiza A,b,c mirando si s esta entre b y (3*(a+b))/4 y repite validando el error y que F(b o s) sean 0. A demás, el método guarda un nuevo valor d, que guarda el valor de c.
La idea del algoritmo es forzar bisección cuando no nos den resultados los demás métodos. Para esto, Brent usa algunas pruebas.
Si usamos bisección y s-b < (b-c) entonces se va a interpolar, si no, se continua con bisección. Esto es porque bisección nos daría un mejor salto que la interpolación.



Ahora, si usamos secante o interpolación y s-b < 1/2c-d entonces también se sigue con interpolación, si no, se fuerza de nuevo bisección. Esto quiere decir que si la interpolación da pasos muy pequeños, será mejor utilizar bisección.

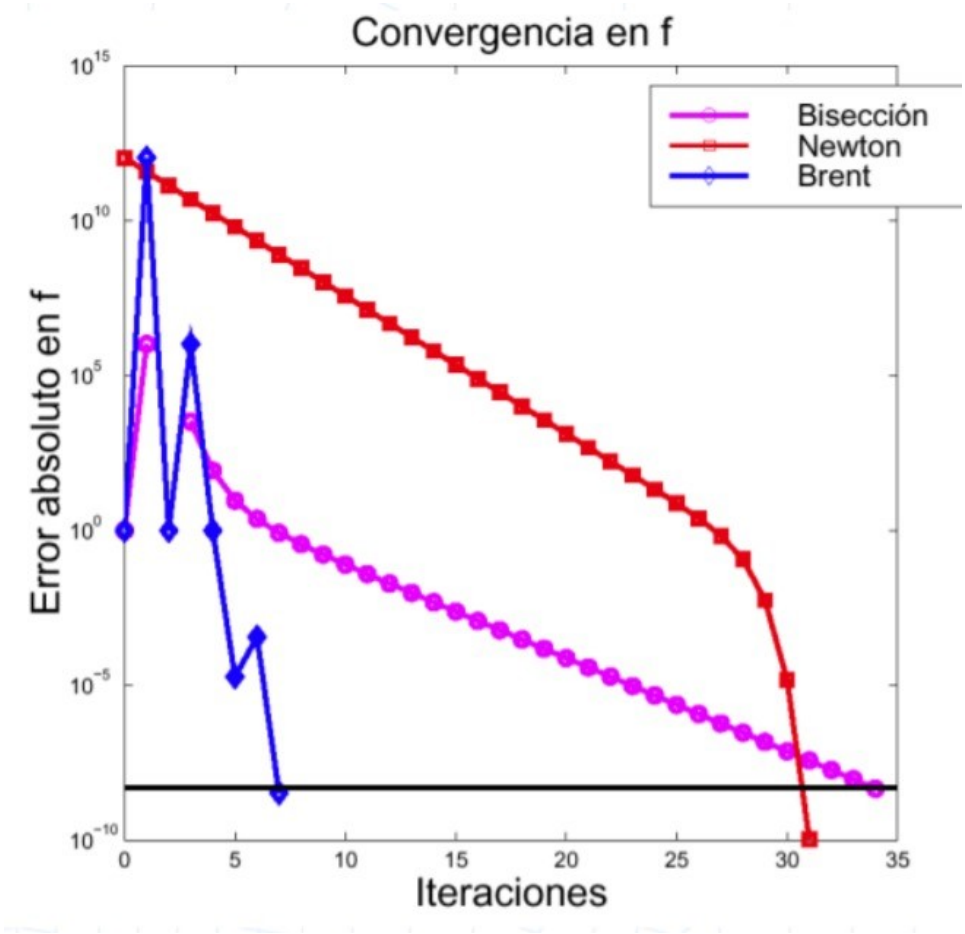


Comparaciones

Method	Steps	Root Found
Bisection	24	1.8392867743968964
Secant	6	1.839286755226924
Dekker	7	1.8392867552141612
IQI - $1, \frac{3}{2}, 2$	5	1.8392867552230825
Brent	6	1.8392867552141612

He encontrado estas comparaciones y he llegado a las siguientes conclusiones:

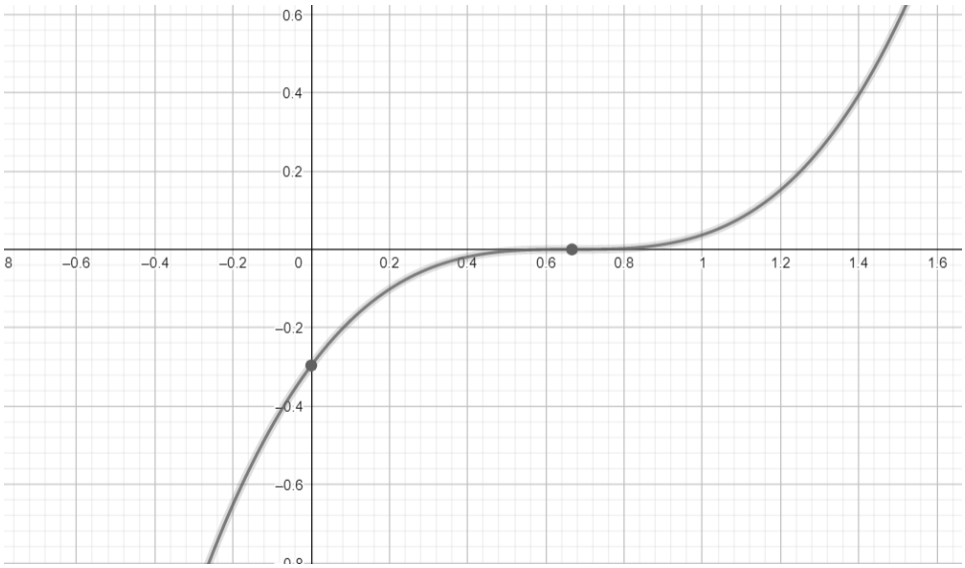
1. El algoritmo de Dekker es tan rápido como el de la secante, pero dependen del comportamiento de la función.
2. Interpolación Es tan rápido como Müller cuando funciona, aunque falla en repetidas ocasiones.
3. Brent es igual de rápido que interpolación cuando esta cerca, Dekker y secante cuando se comportan.
4. Brent dice que su algoritmo parce ser tan rápido como el de Dekker pero que además garantiza la convergencia para cualquier función con una velocidad razonable.



Pruebas

Para estas pruebas, Taylor se mantendrá fijo en 0.6 con polinomio de grado 4.

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



1. En esta prueba, notamos que el método converge en 61 iteraciones y presenta un error de 2.66e-06 lo que es un error absoluto relativamente alto.

```
a: -3  b: 3
Brent
      converged: True
           flag: 'converged'
 function_calls: 61
     iterations: 60
           root: 0.6666680000766564
Taylor  0.6666706695951692
error:  2.669518512798952e-06
```

2. Para tratar de reducir el error, se han cambiado los límites del algoritmo

```
a: -2  b: 2
Brent
      converged: True
           flag: 'converged'
 function_calls: 61
     iterations: 60
           root: 0.6666660657635841
Taylor  0.666670669595169 2
error:  4.603831585070495e-06
```

El error aumenta

3. Se cambian de nuevo los límites, esta vez, acercándolo mas a la solución

```
a: 0  b: 1
Brent
      converged: True
           flag: 'converged'
 function_calls: 50
     iterations: 49
           root: 0.6666685251964675
Taylor  0.6666706695951692
error:  2.1443987017200072e-06
```

No solo disminuye el error si no también el número de iteraciones.

4. Se reduce una ultima vez los límites, esta vez entre 0.5 y 1.

```
a: 0.5  b: 1
Brent
      converged: True
           flag: 'converged'
 function_calls: 48
     iterations: 47
           root: 0.6666628518987887
Taylor  0.6666706695951692
error:  7.81769638047436e-06
```

Logramos reducir aun mas el número de iteraciones, pero el error aumenta levemente. Aquí nos percatamos de que lo que ocurrió es que el valor del algoritmo de Brent se acercó aun más a la solución y

5. En esta prueba cambiamos la tolerancia del error de 2**-90 a 2**-16

```
a: 0.5  b: 1
Brent
      converged: True
           flag: 'converged'
 function_calls: 44
     iterations: 43
           root: 0.6666671560678465
Taylor  0.666670621360187
error:  3.465292340409931e-06
```

Se reducen las iteraciones nuevamente. Concluimos que el error con respecto a taylor siempre estará alrededor de xe-6 ya que esta es la cantidad de cifras significativas que estudiamos en la parte de significancia.

Parte 3. Metodo de Remez

Parametros Métodos Remez y Taylor:

Los parámetros para estos algoritmos son:

1. Remez
   a) Función: debe ser continua
   b) A: inicio de intervalo
   c) B: fin de intervalo

d) Tolerancia
e) x: El valor inicial de para Chebyshev

2. Taylor
   a) Función
   b) X: donde se quiere aproximar el polinomio
   c) N: el orden del polinomio a aproximar

El algoritmo de intercambio de Remez o el algoritmo de Remez es un algoritmo iterativo cuyo objetivo es el de encontrar aproximaciones simples a funciones. Para esto, el algoritmo de Remez comienza con la función a ser aproximada y un conjunto x de n+2. El algoritmo construye un polinomio de grado n-ésimo que lleva a valores de error alternos tomando como referencia n+2. Las ecuaciones del método se resuelven a continuación:

Dados los puntos de referencia se puede resolver el sistema para encontrar el polinomio P y el número e.

1. Primer Paso:
   El primer paso para desarrollar el metodo es seleccionar el valor de un x_0 que debe estar dentro del intervalo de la funcion, con esto se genera un grupo de equis $x_0, x_1, ..., x_{n+1} en\, [a, b]$, esta cantidad de numeros va desde 0 hasta el valor de n+1. En el cual n es el grado que fue enviado a la funcion. A continuacion la formula desarrollada para generar el valor de las equis:

$$x_i = \frac{1}{2}(a+b) + \frac{1}{2}(b-a) \cdot cos\left(\frac{2i-1}{2n}\pi\right) \tag{0.1}$$

2. Segundo Paso:
   Vamos a considerar el siguiente sistema de ecuaciones:

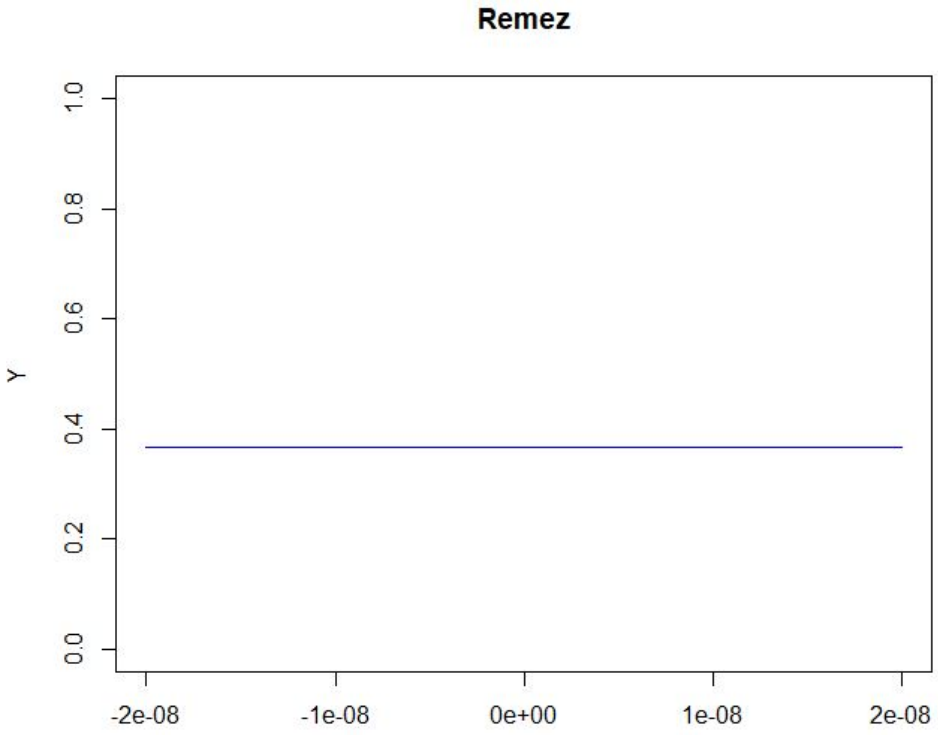
$$\begin{cases} p_0 + p_1 x_0 + p_2 x_o^2 + ... + p_n x_0^n - f(x_0) & = +\epsilon \\ p_0 + p_1 x_1 + p_2 x_1^2 + ... + p_n x_1^n - f(x_1) & = -\epsilon \\ p_0 + p_1 x_2 + p_2 x_2^2 + ... + p_n x_2^n - f(x_2) & = +\epsilon \\ ... & ... \\ p_0 + p_1 x_{n+1} + p_2 x_{n+1}^2 + ... + p_n x_{n+1}^n - f(x_{n+1}) & = (-1)^{n+1}\epsilon \end{cases}$$

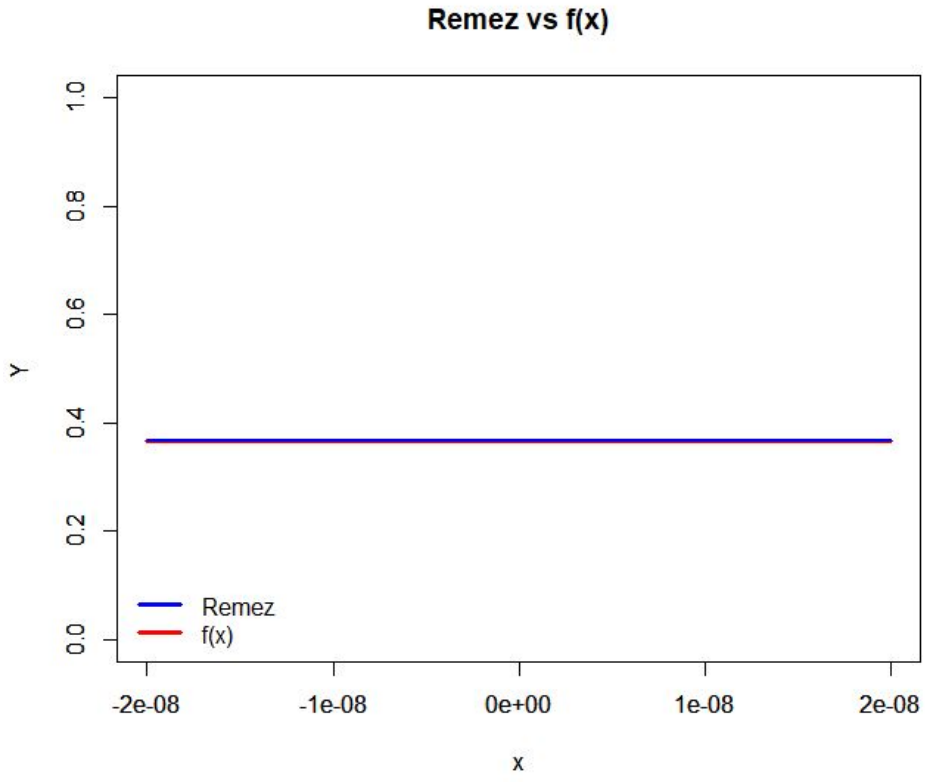
   Entonces cada una de las columnas del sistema de ecuaciones excepto la ultima, correspondes a cada uno de los grados que fueron enviados a la funcion. Y para cada una de las filas se considerara el valor de las equis generadas en el primer paso. La ultima columan del sistema corresponde al error que se recibe en la funcion

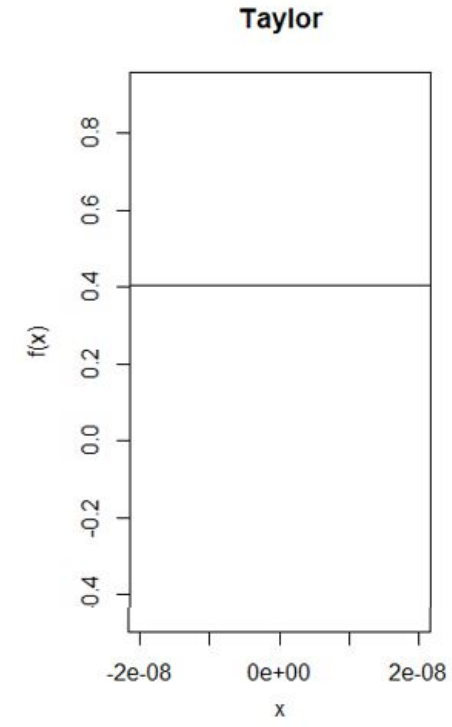
3. Tecer Paso:

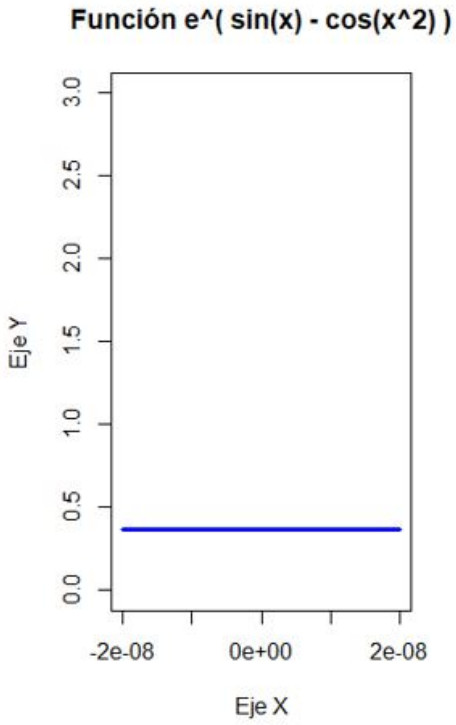
   Solucionando el sistema de ecuaciones encontraremos un polinomio que corresponde a la aproximacion que esperamos que el metodo de remez nos proporciona y asi compararlo con la funcion f(x)

El metodo de remez


Remez
Y
1.0
0.8
0.6
0.4
0.2
0.0
-2e-08
-1e-08
0e+00
1e-08
2e-08


Remez vs f(x)
Y
1.0
0.8
0.6
0.4
0.2
0.0
Remez
f(x)
-2e-08
-1e-08
0e+00
1e-08
2e-08
x


Taylor
f(x)
0.8
0.6
0.4
0.2
0.0
-0.2
0.4
-2e-08
0e+00
2e-08
x


Función e^(sin(x) - cos(x^2))
Eje Y
3.0
2.5
2.0
1.5
1.0
0.5
0.0
-2e-08
0e+00
2e-08
Eje X