Métodos cerrados y abiertos para encontrar raíces

Neil Otniel Moreno Rivera,
Universidad de Guanajuato, no.morenorivera@ugto.mx

***Resumen*—Este documento sirve para comparar y poder dar una respuesta, sobre cuál de estos métodos numéricos es el más optimo para encontrar raíces. Ya sea en rapidez, complejidad, exactitud, etc. Estos métodos son conformados por los cerrados, el de bisección y falsa posición, para los métodos abiertos, punto fijo, Newton-Raphson y secante.**

***Abstract*-- This document serves to compare and to be able to give an answer, about which of these numerical methods is the most optimal to find roots. Whether in speed, complexity, accuracy, etc. These methods are made up of the closed ones, bisection and false position, for the open methods, fixed point, Newton-Raphson and secant.**

I. Introducción

Para encontrar las raíces de ecuaciones se ocupa de métodos que aprovechan el hecho de que una función cambia de signo en la vecindad de una raíz. A estas técnicas se les llama métodos cerrados, o de intervalos, porque se necesita de dos valores iniciales para la raíz. Como su nombre lo indica, dichos valores iniciales deben "encerrar", o estar a ambos lados de la raíz. Los métodos particulares descritos aquí emplean diferentes estrategias para reducir sistemáticamente el tamaño del intervalo y así converger a la respuesta correcta. Como preámbulo de estas técnicas se analizarán los métodos gráficos para representar tanto las funciones como sus raíces. Además de la utilidad de los métodos gráficos para determinar valores iniciales, también son útiles para visualizar las propiedades de las funciones y el comportamiento de los diversos métodos numéricos.[1]

También se verán los métodos abiertos. En contraste, los métodos abiertos descritos se basan en fórmulas que requieren únicamente de un solo valor de inicio **X** o que empiecen con un par de ellos, pero que no necesariamente encierran la raíz. éstos, algunas veces, divergen o se alejan de la raíz verdadera a medida que se avanza en el cálculo. Sin embargo, cuando los métodos abiertos convergen, en general lo hacen mucho más rápido que los métodos cerrados. Empecemos el análisis de los métodos abiertos y cerrados para comparar su rapidez, complejidad y exactitud. [2]

La función que usaremos es la siguiente:

$F(x) = x^2 + 21x + 110$(Fig. 1)

Puesto que es, relativamente fácil conocer la raíz de esta, la cual es: -10 y -11, posee 2 raíces, nos conformaremos con que cualquier método, encuentre alguna de las dos, solo encontraremos una raíz de las 2, puesto que esto es un reporte de métodos que solo encuentran una raíz, no de métodos multi raíz. Para facilidad y rapidez, utilizaremos como método de comparación, el error aproximado, esto porque posee una fácil implementación en todos los métodos, tanto abiertos como cerrados, junto con el numero de iteraciones a realizadas, dicho vulgarmente "Cuantas veces el programa tuvo que correrse para encontrar una raíz aproximada", no utilizamos el tiempo que tarda puesto que esto dependerá del equipo donde sea ejecutado el Código y nunca dará un mismo tiempo, por lo que seria tomar medidas un tanto ambiguas.

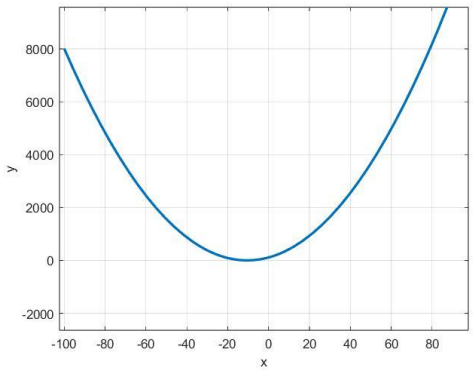


Fig. 1. Grafica de la función a usar para la comparación.

II. Métodos cerrados

A. *Bisección*

El método de bisección, conocido también como de corte binario, de partición de intervalos o de Bolzano, es un tipo de búsqueda incremental en el que el intervalo se divide siempre a la mitad. Si la función cambia de signo sobre un intervalo, se evalúa el valor de la función en el punto medio. La posición de la raíz se determina situándola en el punto medio del subintervalo, dentro del cual ocurre un cambio de signo. El proceso se repite hasta obtener una mejor aproximación. En la figura 2, se muestra como funciona el método de una forma bastante grafica. [3]

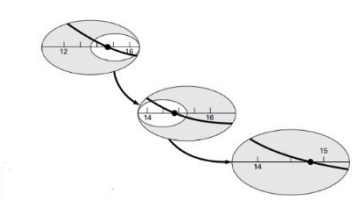


Fig. 2. Demostración grafica de cómo funciona el algoritmo.

* Revista Argentina de Trabajos Estudiantiles. Patrocinada por la IEEE.

Como criterio de paro se utiliza una estimación del error aproximado, esto para decidir en que momento es conveniente detener el proceso, siguiente la siguiente ecuación:

$$Ea = \left|\frac{x_{i+1} - x_i}{x_{i+1}}\right|$$

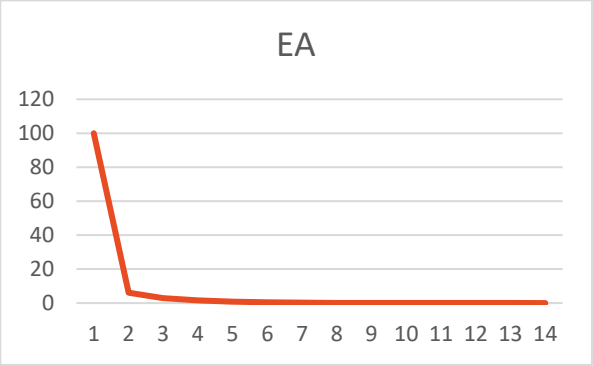
Para utilizar el código escrito en Matlab, debemos tomar en cuenta unas cosas, los limites tanto superior como inferior, pero en estos debemos revisar su posición con respecto al eje y, con esto hago referencia a que deben tener signos opuestos, en nuestro caso podríamos decir que -13 y -10.3, son buenos límites para encerrar la raíz, también el error máximo permitido que en todos los casos será 0.1% y que nuestro algoritmo haga el resto, arrojando esta tabla.

El programa en Matlab se muestra en el apéndice 1.

Inferior	Superior	Raíz	Iteración	EA
-13	-10.3	-10.975	1	100
-11.65	-10.3	-11.313	2	6.1503417
-11.65	-10.975	-11.144	3	2.9834254
-11.313	-10.975	-11.059	4	1.5143017
-11.144	-10.975	-11.017	5	0.7629274
-11.059	-10.975	-10.996	6	0.3829244
-11.017	-10.975	-11.007	7	0.1918295
-11.017	-10.996	-11.001	8	0.0958228
-11.007	-10.996	-10.999	9	0.0479344
-11.001	-10.996	-11	10	0.0239729
-11.001	-10.999	-10.999	11	0.011985
-11	-10.999	-11	12	0.0059929
-11	-10.999	-11	13	0.0029963
-11	-11	-11	14	0.0014982

Tabla 1. Tabla de resultados del primer método.

Encontrando con relativa rapidez y soltura la raíz exacta -11, utilizando 14 iteraciones para este proceso.



Grafica 1. Muestra cómo se reduce el error.

B. Falsa posición

Aun cuando la bisección es una técnica perfectamente válida para determinar raíces, su método de aproximación por "fuerza bruta" es relativamente ineficiente. La falsa posición es una alternativa basada en una visualización gráfica.

Un inconveniente del método de bisección es que al dividir el intervalo de **xl** a **xu** en intervalos iguales, no se toman en consideración las magnitudes de **f(xl)** y **f(xu)**. Por ejemplo, si **f(xl)** está mucho más cercana a cero que **f(xu)**, es lógico que la raíz se encuentre más cerca de xl que de xu. Un método alternativo que aprovecha esta visualización gráfica consiste en unir **f(xl)** y **f(xu)** con una línea recta. La intersección de esta línea con el eje de las x representa una mejor aproximación de la raíz. El hecho de que se reemplace la curva por una línea recta da una **"falsa posición"** de la raíz; de aquí el nombre de método de la falsa posición, o en latín, regula falsi. También se le conoce como método de interpolación lineal. [4].

La explicación puede ser un tanto difícil de entender, por lo que usando algo más gráfico, podemos explicarlo mejor. El método de falsa posición se basa en triángulos semejantes.

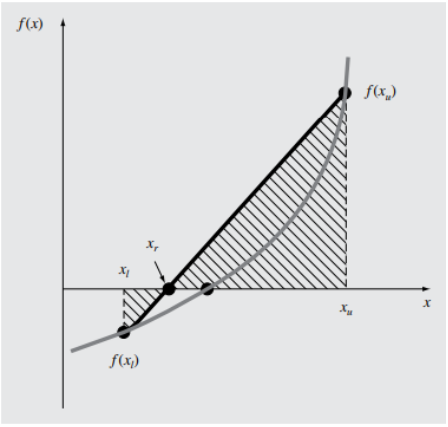


Fig. 3. Demostración grafica de cómo funciona el algoritmo.

Al utilizar triángulos semejantes debemos insertar una línea recta imaginaria, la cual se estima con la siguiente formula:

$$\frac{F(x_l)}{x_r - x_l} = \frac{F(x_u)}{x_r - x_u}$$

Que al despejarla, obtenemos una fórmula más casual:

$$x_r = x_u - \frac{F(x_u)(x_l - x_l)}{F(x_r) - F(x_u)}$$

El valor de x_r calculado con la ecuación (5.7), reemplazará, después, a cualquiera de los dos valores iniciales, xl o xu, y da un valor de la función con el mismo signo de f(xr). De esta manera, los valores xl y xu siempre encierran la verdadera raíz. El proceso se repite hasta que la aproximación a la raíz sea adecuada.[4]

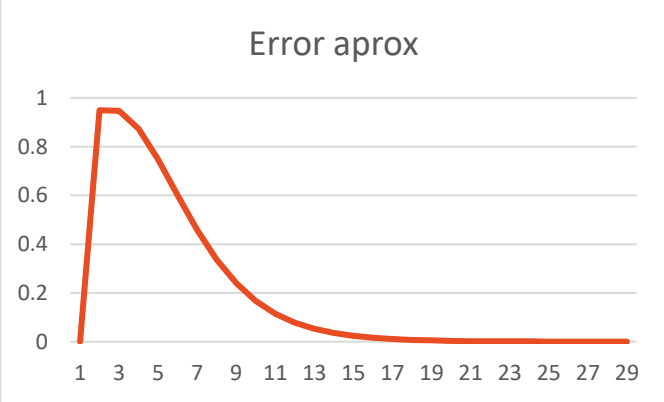
Para el manejo del programa realizado en Matlab, debemos revisar algunas cuestiones, al igual que el método de bisección, es que la raíz este en el rango deseado, que los limites que introducimos posean signos opuestos, para que podamos obtener valores. Al utilizar el código, arrojo esta tabla, como limites utilizamos -13 y -10.3, al igual que el método de bisección, puesto que como habíamos dicho anteriormente, eran buenos límites para cumplir la regla de signos. Junto con el error de 0.1%.

El programa en Matlab se muestra en el apéndice 2.

It.	Xa	Xr	Xb	Error aprox
1	-13	-10.3913043	-10.3	
2	-13	-10.4909091	-10.3913043	0.949
3	-13	-10.5912409	-10.4909091	0.947
4	-13	-10.684507	-10.5912409	0.873
5	-13	-10.7649528	-10.684507	0.747
6	-13	-10.829981	-10.7649528	0.6
7	-13	-10.8798444	-10.829981	0.458
8	-13	-10.9165541	-10.8798444	0.336
9	-13	-10.9427778	-10.9165541	0.24
10	-13	-10.9611101	-10.9427778	0.167
11	-13	-10.9737329	-10.9611101	0.115

12	-13	-10.9823339	-10.9737329	0.078
13	-13	-10.9881528	-10.9823339	0.053
14	-13	-10.9920706	-10.9881528	0.036
15	-13	-10.9946997	-10.9920706	0.024
16	-13	-10.9964602	-10.9946997	0.016
17	-13	-10.9976374	-10.9964602	0.011
18	-13	-10.9984237	-10.9976374	0.007
19	-13	-10.9989486	-10.9984237	0.005
20	-13	-10.9992988	-10.9989486	0.003
21	-13	-10.9995324	-10.9992988	0.002
22	-13	-10.9996882	-10.9995324	0.001
23	-13	-10.9997921	-10.9996882	0.001
24	-13	-10.9998614	-10.9997921	0.001
25	-13	-10.9999076	-10.9998614	0
26	-13	-10.9999384	-10.9999076	0
27	-13	-10.9999589	-10.9999384	0
28	-13	-10.9999726	-10.9999589	0
29	-13	-10.9999817	-10.9999726	0

Tabla 2. Tabla de resultados del segundo método.



Grafica 2. Muestra cómo se reduce el error.

Encontrando una raíz aproximada -10.9999726, utilizando 29 iteraciones para este proceso.

III. MÉTODOS ABIERTOS

A. *Punto fijo*

Como se dijo antes, los métodos abiertos emplean una fórmula para predecir la raíz. Esta fórmula puede desarrollarse como una iteración simple de punto fijo (también llamada iteración de un pun to o sustitución sucesiva o método de punto fijo), al arreglar la ecuación f(x) = 0 de tal modo que x esté del lado izquierdo de la ecuación:

$$x = g(x)$$

Esta transformación se realiza mediante operaciones algebraicas o simplemente sumando x a cada lado de la ecuación original.

$$G(x) = x^2 + 21x + 110$$

$$x = \frac{-x^2 - 110}{21}$$

La utilidad de la ecuación es que proporciona una fórmula para predecir un nuevo valor de x en función del valor anterior de x. De esta manera, dado un valor inicial para la raíz x_i, la ecuación (6.1) se utiliza para obtener una nueva aproximación x_{i+1}, expresada por la fórmula iterativa

$$x_{i+1} = g(xi)$$

Como en otras fórmulas iterativas, el error aproximado de esta ecuación se calcula usando el error normalizado estimado:

$$Ea = \left|\frac{x_{i+1} - x_i}{x_{i+1}}\right|$$

El uso del programa realizado en Matlab, considera las siguientes cuestiones, a manera de función por lo que la entrada es un poco diferente a la de los métodos anteriores, por lo que los datos se introducen de la siguiente manera.

```
PuntoFijo(x0,es,imax,gx)
```

Para este caso, nosotros lo introducimos así, utilizamos como punto el numero 0, como error máximo aceptado de manera porcentual un 0.1%, en un máximo de 60 iteraciones y con la formula despejada de aquella manera.

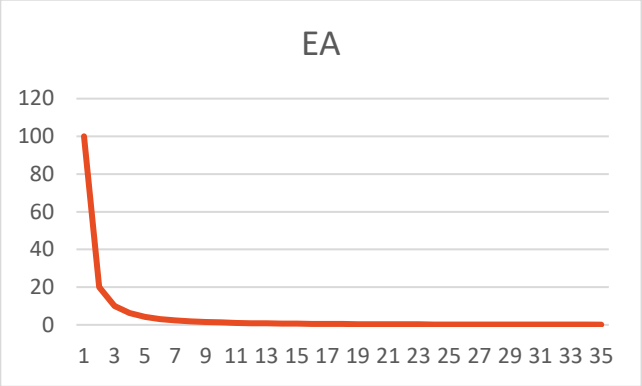
Punto_fijo(0,0.1,60,'-(x.^2+110)/21')

El programa en Matlab se muestra en el apéndice 3.

IT	XR	EA
1	-5.2381	100
2	-6.54465	19.9637
3	-7.27774	10.07299
4	-7.76026	6.21788
5	-8.10579	4.26279
6	-8.36685	3.12016
7	-8.57163	2.38902
8	-8.7368	1.89054
9	-8.87294	1.53429
10	-8.9871	1.27025
11	-9.08419	1.06878
12	-9.16773	0.91134
13	-9.24035	0.78586
14	-9.304	0.68415
15	-9.36021	0.60052
16	-9.41017	0.53089
17	-9.45482	0.47228
18	-9.49494	0.42248
19	-9.53114	0.37978
20	-9.56393	0.3429
21	-9.59375	0.31083
22	-9.62096	0.28276
23	-9.64585	0.25805
24	-9.66868	0.2362
25	-9.68969	0.21677
26	-9.70905	0.19942
27	-9.72694	0.18388
28	-9.74349	0.1699
29	-9.75884	0.15727
30	-9.77309	0.14584
31	-9.78635	0.13546

32	-9.7987	0.12601
33	-9.81021	0.11738
34	-9.82096	0.10949
35	-9.83102	0.10225

Tabla 3. Tabla de resultados del tercer método.



Grafica 3. Muestra cómo se reduce el error.

Encontrando una raíz aproximada -9.83102, utilizando 35 iteraciones para este proceso.

B. Newton-Raphson

En análisis numérico, el método de Newton (conocido también como el método de Newton-Raphson o el método de Newton-Fourier) es un algoritmo eficiente para encontrar aproximaciones de los ceros o raíces de una función real. También puede ser usado para encontrar el máximo o mínimo de una función, encontrando los ceros de su primera derivada.
El método de Newton-Raphson es un método abierto, en el sentido de que su convergencia global no está
garantizada. La única manera de alcanzar la convergencia es seleccionar un valor inicial lo suficientemente cercano a la raíz buscada. Así, se ha de comenzar la iteración con un valor razonablemente cercano al cero (denominado punto de arranque o valor supuesto). La relativa cercanía del punto inicial a la raíz depende mucho de la naturaleza de la propia función; si ésta presenta múltiples puntos de inflexión o pendientes grandes en el entorno de la raíz, entonces las probabilidades de que el algoritmo diverja aumentan, lo cual exige seleccionar un valor supuesto cercano a la raíz. Una vez se ha hecho esto, el método linealiza la función por la recta tangente en ese valor supuesto. La abscisa en el origen de dicha recta será, según el método, una mejor aproximación de la raíz que el valor anterior. Se realizarán sucesivas iteraciones hasta que el método haya convergido lo suficiente. Sea f : [a, b] -> R función derivable definida en el intervalo real [a, b]. Empezamos con un valor inicial x_0 y definimos para cada número natural n [5]

$$x_{n+1} = x_n - \frac{F(x_n)}{F'(x_n)}$$

Donde F' denota la derivada de F.

Nótese que el método descrito es de aplicación exclusiva para funciones de una sola variable con forma analítica o implícita cognoscible.

En la siguiente tabla, se muestra la salida del programa elaborado en Matlab, el cual muestra el número de iteraciones, la raíz que obtuvo al final y el error aproximado como criterio de paro.

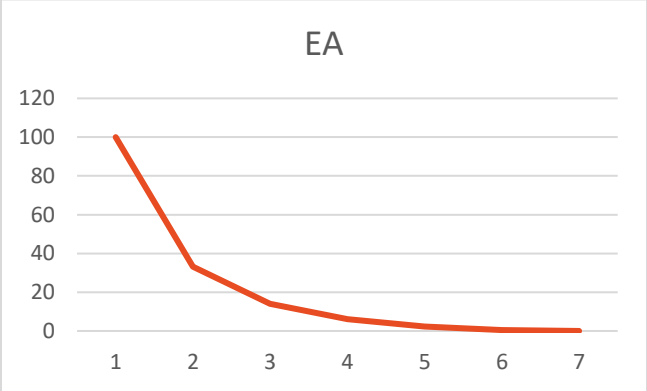
Como valores de entrada, solamente se tuvo que introducir la función, un punto de inicio y el porcentaje de error aceptado.

Los valores introducidos fueron, nuestra función mencionada en un inicio, como numero de inicio fue el 0 y como porcentaje de error, como en todo fue del 0.1%.

El programa en Matlab se muestra en el apéndice 4.

IT	XR	EA
1	-5.2381	100
2	-7.84529	33.23263
3	-9.12556	14.02947
4	-9.72183	6.13335
5	-9.95028	2.2959
6	-9.99775	0.4748
7	-9.99999	0.02243

Tabla 4. Tabla de resultados del cuarto método.



Grafica 4. Muestra cómo se reduce el error.

Al final, esta tabla nos muestra que logro encontrar la raíz de -10, en tan solo 7 iteraciones, lo cual hasta el momento ha sido, el método en el que menos iteraciones fueron necesarias para encontrar una raíz.

C. Secante

Un problema potencial en la implementación del método de Newton-Raphson es la evaluación de la derivada. Aunque esto no es un inconveniente para los polinomios ni para muchas otras funciones, existen algunas funciones cuyas derivadas en ocasiones resultan muy difíciles de calcular. En dichos casos, la derivada se puede aproximar mediante una diferencia finita dividida hacia atrás. [6]

Utilizando la definición de la secante, de una forma que nos sea conveniente, como en la siguiente ecuación:

$$F'(x) \cong \frac{F(x_{i-1}) - F(x_i)}{x_{i-1} - x_i}$$

Entonces, con esta aproximación, sustituimos la ecuación que teníamos para el método de Newton-Raphson, para obtener la ecuación iterativa:

$$x_{i+1} = \frac{F(x_i)(x_{i-1} - x_i)}{F(x_{i-1}) - F(x_i)}$$

Esta ecuación, es la formula para el método de la secante, por lo que se requieren dos valores iniciales de x, como si fuera un método cerrado, pero como no se necesita la diferencia de signos como en uno cerrado no se clasifica como un método cerrado.

Para su uso en el programa realizado en Matlab, como ya mencionamos necesitamos dos valores de x, un inferior y un

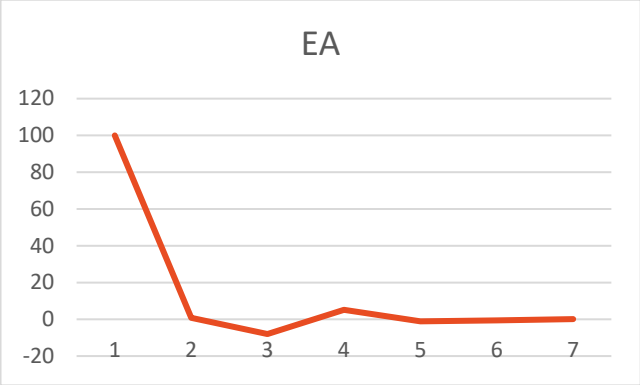
superior, a su vez que un porcentaje de error permitida y claro nuestra función.

En este caso usamos, los límites que usamos para los métodos cerrados, que ya vimos que encierran a la raíz y como porcentaje de error permitido 0.1% el usado generalmente por todos los métodos. Como resultado del programa obtenemos esta tabla, que muestra como es normal, el número de iteración, la raíz aproximada y el error aproximado como criterio de paro.

El programa en Matlab se muestra en el apéndice 5.

IT	XR	EA
1	-10.3	100
2	-10.3913043	0.879
3	-9.6197183	-8.021
4	-10.1504644	5.229
5	-10.0465263	-1.035
6	-9.9912821	-0.553
7	-10.0004215	0.091

Tabla 3. Tabla de resultados del tercer método.



Grafica 5. Muestra cómo se reduce el error.

Como resultado final del proceso, el cual fue bastante rápido, obtuvimos una raíz aproximada de -10.0004215 en solamente 7 iteraciones, igual que a su método padre (Entiéndase padre, como que este método parte como ramificación del método anterior).

IV. Conclusiones

Como conclusión, viendo los gráficos, los resultados y los errores, podemos asegurar que los métodos abiertos son mucho más rápidos, por número de iteraciones pero la desventaja que se tiene es su complejidad, con esto me refiero a que el tiempo de ejecución puede ser bastante prolongado, aunque en este reporte no estamos midiendo eso, aunque también podemos referirnos a complejidad como dificultad de procesamiento o de uso, tomando en cuenta los más rápidos, método de Newton-Raphson y el método de la secante, estos implican obtener la derivada o la definición de la secante de la función, por lo que si el equipo en el que se esta usando este algoritmo puede ser algo pesado o incluso al hacerlo a mano puede haber grandes complicaciones en funciones grandes. A su vez el método de punto fijo, utiliza una forma un tanto antinatural de colocar una función, tiene fundamento el porque se utiliza así, pero eso le agrega indudablemente complejidad. Así que diría que depende de como será usado estos métodos, si el problema dispone de capacidad de computo decente, es indudablemente mejor utilizar un método abierto en particular y más usado el Newton-Raphson, pero si no se tiene un equipo de computo o procesamiento decente, o incluso es algo de segundo plano, para ahorrar costes o cualquier limitante importante, diría que utilizar uno cerrado no estaría tan errado.

Anotaciones

Este reporte fue realizado en tiempo récord, por lo que no se pudo implementar y comparar todos los tipos de errores, por lo que se tomo el más significativo y fácil de aplicar.

Referencias

[1] Chapra S. y CanaleR. (1989).Métodos numéricos para Ingenieros. 7ma. Edición. México: McGraw Hill. Pp 96.
[2] Chapra S. y CanaleR. (1989).Métodos numéricos para Ingenieros. 7ma. Edición. México: McGraw Hill. Pp 113.
[3] Chapra S. y CanaleR. (1989).Métodos numéricos para Ingenieros. 7ma. Edición. México: McGraw Hill. Pp 100.
[4] Chapra S. y CanaleR. (1989).Métodos numéricos para Ingenieros. 7ma. Edición. México: McGraw Hill. Pp 105.
[5] J. C. Virgil, “Application of numerical methods to solve nonlinear equations for sea ...,” *URP*, Feb-2011. [Online]. Available: https://www.urp.edu.pe/pdf/id/2555/n/app#:~:text=En%20an%C3%A1lisis%20num%C3%A9rico%2C%20el%20m%C3%A9todo,ra%C3%ADces%20de%20una%20funci%C3%B3n%20real. [Accessed: 09-Mar-2023].
[6] Chapra S. y CanaleR. (1989).Métodos numéricos para Ingenieros. 7ma. Edición. México: McGraw Hill. Pp 121.
[7]

Bibliografia

- Chapra S. y CanaleR. (1989).Métodos numéricos para Ingenieros. 7ma. Edición. México: McGraw Hill.
- B.R. Hunt, R.L. Lipsman, and J.M. Rosenberg. A Guide to MATLAB, for beginners and experienced users. Cambridge University Press, 2001.
- Stormy Attaway. MATLAB: A Practical Introduction to Programming and Problem Solving. Elsevier Inc, 2012
- J. C. Virgil, “Application of numerical methods to solve nonlinear equations for sea ...,” URP, Feb-2011. [Online]. Available: https://www.urp.edu.pe/pdf/id/2555/n/app#:~:text=En%20an%C3%A1lisis%20num%C3%A9rico%2C%20el%20m%C3%A9todo,ra%C3%ADces%20de%20una%20funci%C3%B3n%20real. [Accessed: 09-Mar-2023].

APÉNDICES

Apéndice 1. (Bisección)

```
clear; close all; clc
syms x
fx=(x.*x+21*x+110);
a=input('Intervalo inferior: ');
b=input('Intervalo superior: ');
fi=subs(fx,a);
fs=subs(fx,b);
while sign(fi)==sign(fs)
    a=a-1;
    b=b+1;
    fi=subs(fx,a);
    fs=subs(fx,b);
end
%variables auxiliares
E=1; e=0.0001; A=zeros(2,4);
i=1;
V=0;
E_Ap=0;
Error=0;
r=-11;
iter=1;
cont=1;
while E>e
    A(i,1:2)=[a b];
    xm=0.5*(a+b);
    fp=subs(fx,xm);
    if fi*fp<0
        b=xm;
    elseif(fi*fp>0)
        a=xm;
    end
    E=abs(xm-0.5*(a+b));
    A(i,3)=0.5*(a+b);    %Nuevo valor de xm
    A(i,4)=cont;
    cont=cont+1;
    i=i+1;
    V(i)=xm;
    e1=100*abs((r-xm)/r);
    Error1(i)=e1;
    Error(i)=E*100;
    if(i>1)
        E_Ap(i)=100*abs((V(i)-V(i-1))/V(i));
%         A(i,5)=E_Ap(i);
    end
end
x=['La raiz es xm= ' num2str(A(end,3))];
disp(x)
fplot(fx,[-100,100],'LineWidth',2)
xlabel('x'); ylabel('y')
title('Metodo de Biseccion');
grid on
figure
plot(Error1)
hold on;
plot(Error1,'*')
plot(Error)
hold on;
plot(Error,'*')
plot(E_Ap)
hold on
plot(E_Ap,'*')
grid on
while iter<=i
    fprintf('%11.7f \n',E_Ap(iter));
    iter=iter+1;
end
T=array2table(A,'VariableNames',{'a','b','xm'
,'i'})
```

Apéndice 2. (Falsa posición)

```
clear; clc; close all;
clear; syms x;
xai=input('Ingrese limite inferior: ');
xbi=input('Ingrese limite superior: ');
tol=input('Ingrese el porcentaje de Error: ');
f=(x.*x+21*x+110);
f1=eval(subs(f,xai));
f2=eval(subs(f,xbi));
i=1;
ea(1)=100;
fplot(f,[-100,100],'LineWidth',2)
xlabel('x'); ylabel('y')
if f1*f2 < 0
    xa(1)=xai;f1=subs(f,xa(1));
    xb(1)=xbi;f2=subs(f,xb(1));
    xr(1)=xa(1)-f1*(xb(1)-xa(1))/(f2-f1);
f3=subs(f,xr(1));
    fprintf('It.       Xa           Xr        Xb
Error aprox \n');
    fprintf('%2d \t %11.7f \t %11.7f \t %11.7f
\n',i,xa(i),xr(i),xb(i));
    while abs(ea(i))>=tol,
      if f1*f3 < 0
         xa(i+1)=xa(i);f1=subs(f,xa(i+1));
         xb(i+1)=xr(i);f2=subs(f,xb(i+1));
      end
      if f1*f3> 0
         xa(1)=xr(i);
         xb(1)=xb(i);
      end
      xr(i+1)=xa(i+1)-f1*(xb(i+1)-xa(i+1))/(f2-f1);
      ea(i+1)=abs((xr(i+1)-xr(i))/(xr(i+1)))*100;
      fprintf('%2d \t %11.7f \t %11.7f \t %11.7f \t
%7.3f \n',...
               i+1,xa(i+1),xr(i+1),xb(i+1),ea(i+1));
      i=i+1;

   end
else
   fprintf('No existe una raíz en ese intervalo');
end
plot(ea)
hold on
plot(ea,'*')
```

Apéndice 3. (Punto fijo)

```
function PuntoFijo(x0,es,imax,gx)
xr=x0;
iter=0;
g=inline(gx);
do=0;
i=0;
E_a=0;
xr_i=0;
fprintf('It.       Xr        Error aprox \n');
while(do==0)
    xrold=xr;
    xr=g(xrold);
    iter=iter+1;
    i=i+1;
    if(xr~=0)
        ea=abs((xr-xrold)/xr)*100;
        E_a(i)=ea;
    end
    if((ea<es) || (iter>=imax))
        break;
    end
    xr_i(i)=xr;
    fprintf('%2d \t %3.5f \t %3.5f \t
\n',i,xr_i(i),E_a(i));
end
plot(E_a)
hold on
plot(E_a,'*')
grid on
```

Apéndice 4. (Newton-Raphson)

```
%Limpiamos la pantalla y mostramos el nombre del
método
clear
clc
disp('Método de Newton Raphson')
%Damos de alta la variable simbólica X
syms x
%Introducimos la función,el punto de inicio,así como
%porcentaje de error
f=input('Introduzca la función f(x):');
pi=input('Introduzca el punto de inicio:');
err=input('Porcentaje de error:');
%Graficamos la función
ezplot(f)
grid on
%Calculamos la derivada de la función
d=diff(f);
d=inline(d);
f=inline(f);
ea=100;
j=0;
E_a=0;
xr=0;
fprintf('It.       Xr        Error aprox \n');
while ea>err
    %Aproximamos la raiz con la fórmula correpondiente
    xi=pi-(f(pi)/d(pi));
    %Calculamos el porcentaje de error
    ea=abs(((xi-pi)/xi)*100);
    pi=xi;
    E_a(j+1)=ea;
    xr(j+1)=xi;
    fprintf('%2d \t %3.5f \t %3.5f \t
\n',j+1,xr(j+1),E_a(j+1));
    j=j+1;
end
%Mostramos los resultados en pantalla (con 3
decimales)
fprintf('\nRaiz= %10.3f en %d Iteraciones',pi,j)
```

Apéndice 5. (Secante)

```
xf(1)=input('Ingrese el intervalo inferior: ');
xf(2)=input('Ingrese el intervalo superior: ');
tol=input('Ingrese el porcentaje de error: ');
syms x;
f=input('Ingrese la funciòn: ');
f1=subs(f,x,fx(1));
f2=subs(f,x,fx(2));
ea(1)=100;
i=1;
j=2;
while abs(ea(i))>=tol
   xf(j+1)=(xf(j-1)*f2-xf(j)*f1)/(f2-f1);  f1=f2;
f2=subs(f,x,xf(j+1));
   ea(i+1)=(xf(j+1)-xf(j))/xf(j+1)*100;
   j=j+1;
   i=i+1;
end

fprintf(' i     xf(i)          Error aprox (i) \n');
%fprintf('%2d\t%11.7f\t\n',0,x(1));
for k=2:j;
    fprintf('%2d\t%11.7f\t%7.3f\n',k-1,xf(k),ea(k-1));
end
```