

UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA



Laboratorio 1 - Modelación y Simulación

Integrantes: Maximiliano Arévalo Sáez
Benjamín Muñoz Tapia
Curso: Modelación y Simulación
Sección A-1
Profesor: Gonzalo Acuña Leiva

19 de Mayo de 2020

Tabla de contenidos

1. Introducción	1
2. Marco teórico	2
3. Desarrollo	3
3.1. Desarrollo primera parte	3
3.2. Desarrollo segunda parte	5
3.2.1. Algoritmo de Newton-Raphson	5
3.2.2. Algoritmo de las raíces del vector	7
4. Manual de uso	9
4.1. Primer ejemplo de Newton Raphson:	9
4.2. Segundo ejemplo de Newton Raphson	9
4.3. Ejemplo para algoritmo de raíces cuadradas:	11
5. Conclusiones	13
Bibliografía	14

1. Introducción

Hoy en día se sabe que la matemática es una herramienta que permite comprender fenómenos de la realidad, y también ayuda a predecirlos. Entre las cosas que permite también está el analizar comportamientos y modelos, lo cual se ha hecho más fácil cada vez con las herramientas que han surgido en los últimos tiempos como es el caso de MATLAB, el cual permite procesar grandes cantidades de datos de manera más simple. Es por esto que en este trabajo se trabajará con funciones y operatorias básicas que permite MATLAB.

El presente informe entrega en primer lugar el desarrollo de un breve análisis de dos funciones, luego de haberlas implementado y graficado, para luego pasar a una segunda parte en la que se aplica el método de Newton Raphson con el fin de resolver distintos polinomios que serán entregados por el usuario, y así al final pasar a las conclusiones. Como objetivo general se tiene el cumplimiento total del trabajo, mientras que los objetivos específicos serían lograr graficar funciones, aplicar escalas normal y logarítmica, aplicar el algoritmo del método de Newton Raphson, y finalmente hacer los cálculos pedidos con vectores.

2. Marco teórico

1. Método de Newton Raphson: Es un método numérico que permite hallar la raíz de una función mediante su primera derivada. Se caracteriza por ser uno de los más eficientes actualmente según el numero de iteraciones que requiere. La formula de este método es:

$$x_{n+1} = x_n + \frac{\dot{f}_n}{f_n} \quad (1)$$

El método va iterando cuantas veces sea necesario y siempre cumpliendo una tolerancia de error dado, y a medida que itera se acerca a la raíz del polinomio, aunque solo funciona con polinomios que tengan soluciones reales.

2. Escala logarítmica: Como su nombre lo dice usa el logaritmo como medida para evaluar. Se usa principalmente cuando se trabaja con funciones exponenciales o con funciones con muchos valores, para ambos casos permite ilustrar de mejor forma el comportamiento que tienen las gráficas en comparación a la escala normal.
3. MATLAB: Es un programa que permite realizar diversos cálculos matemáticos utilizando vectores y matrices, también permite trabajar con números reales y complejos, caracteres y otras estructuras que sean más complejas.

3. Desarrollo

3.1. Desarrollo primera parte

En esta sección del trabajo se desarrolla la habilidad de gráfico de funciones en MATLAB, tanto en escala normal como logarítmica. Las funciones con las cuales se va a trabajar son:

$$a(x) = 8 \log_5(4x + 12) \quad (2)$$

$$b(x) = \sin(6(\log_2(x + 9))) + \cos(7 \log_6(4x + 32)) \quad (3)$$

Ambas deben ser graficadas en el intervalo $[0, 15\pi]$ con separaciones de 0.01, siendo la primera graficada en rojo con asterisco y la segunda en verde con guiones. Para esto en primer lugar se definen los intervalos en matlab, y el vector se multiplica en la función creada. Para la implementación de los logaritmos, se usa la propiedad de cambio de base ya que MATLAB solo ofrece logaritmos en base 10. Los gráficos de las ecuaciones (2) y (3), quedan de la siguiente forma:

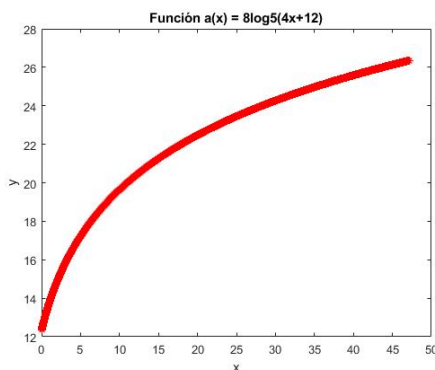


Figura 1: Función $a(x)$

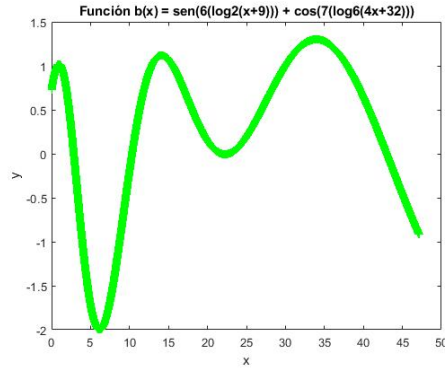


Figura 2: Función $b(x)$

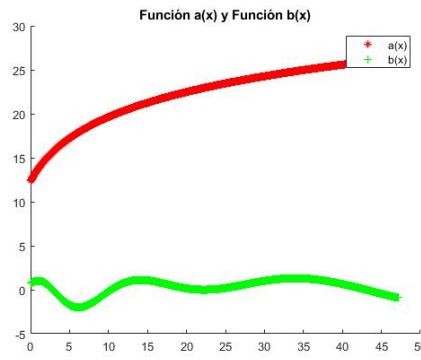


Figura 3: Funciones $a(x)$ y $b(x)$

Despues se hacen las comparaciones de escala normal y logarítmica en el intervalo $[-10,10]$ con particiones de 0.05 para la siguiente función:

$$c(x) = 6e^{x+18} \quad (4)$$

La escala logarítmica tiene el objetivo de ver en una mejor perspectiva el desarrollo de una función exponencial, ya que en una escala lineal esta no se aprecia al ser una curva que crece drásticamente en un punto, mientras que en la escala logarítmica se logra ver de mejor forma los valores que toma la función. A continuación se presentan ambas gráficas, tanto lineal como logarítmica

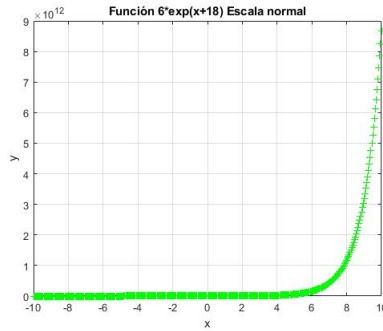


Figura 4: Función $c(x)$, Escala Normal

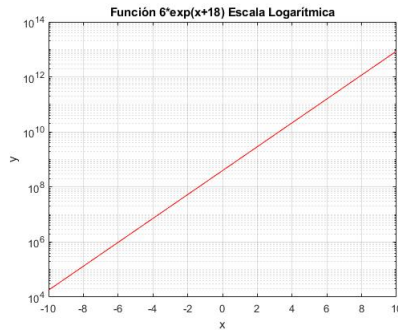


Figura 5: Función $c(x)$, Escala logarítmica

3.2. Desarrollo segunda parte

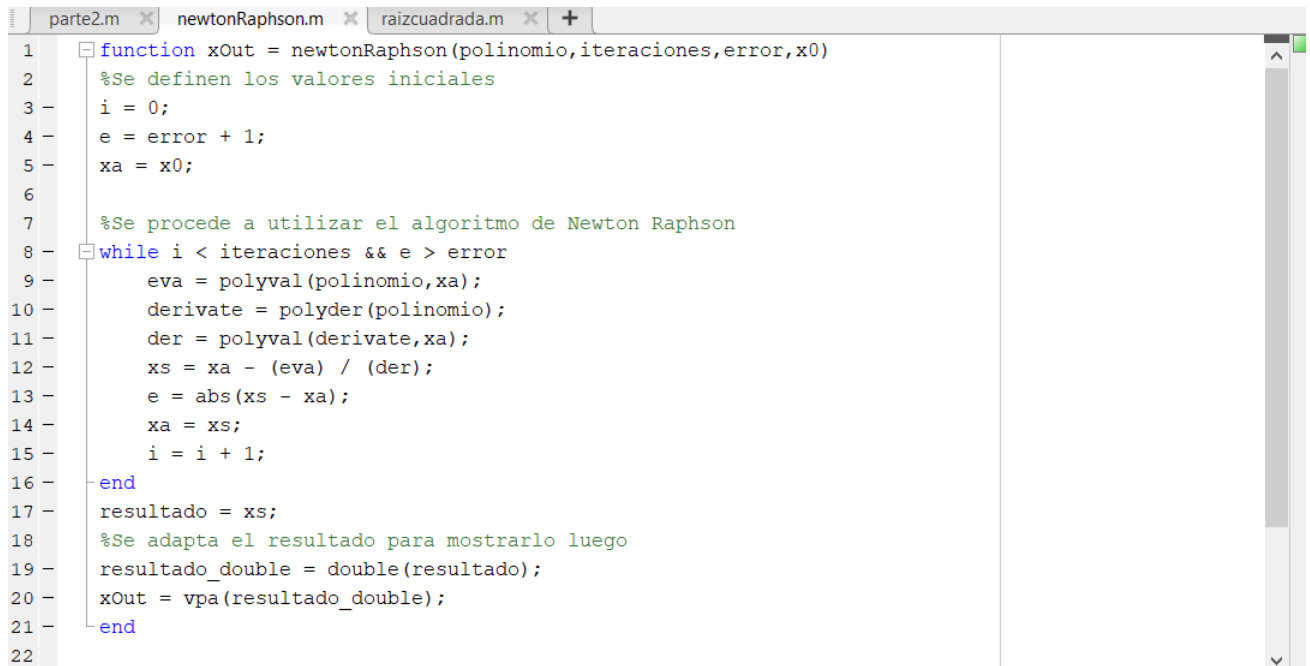
Para la segunda parte se solicita la implementación del algoritmo de Newton-Raphson y un archivo ".m" que permita obtener las raíces de un vector considerando determinados valores de este, es por esto que a continuación se explicará la implementación de cada uno de estos algoritmos.

3.2.1. Algoritmo de Newton-Raphson

Para esta sección se solicita la implementación del algoritmo de Newton-Raphson para obtener la raíz de una función dada o una aproximación de ella. Es importante señalar que la función será de una sola variable, la cual es entregada por consola en formato de polinomio.

La siguiente imagen corresponde al archivo ".m" que contiene la función de Newton-Raphson, luego de esta se explicará el funcionamiento del código para comprender

como trabaja.



```
1 function xOut = newtonRaphson(polinomio,iteraciones,error,x0)
2 %Se definen los valores iniciales
3 i = 0;
4 e = error + 1;
5 xa = x0;
6
7 %Se procede a utilizar el algoritmo de Newton Raphson
8 while i < iteraciones && e > error
9     eva = polyval(polinomio,xa);
10    derivate = polyder(polinomio);
11    der = polyval(derivate,xa);
12    xs = xa - (eva) / (der);
13    e = abs(xs - xa);
14    xa = xs;
15    i = i + 1;
16 end
17 resultado = xs;
18 %Se adapta el resultado para mostrarlo luego
19 resultado_double = double(resultado);
20 xOut = vpa(resultado_double);
21 end
22
```

Figura 6: Código del algoritmo de Newton Raphson

La función "*newtonRaphson*" recibe como parámetros el polinomio o función a la que se le obtendrá la raíz (o aproximación de esta), el número de iteraciones para el algoritmo, el error aceptado y un valor inicial (x_0).

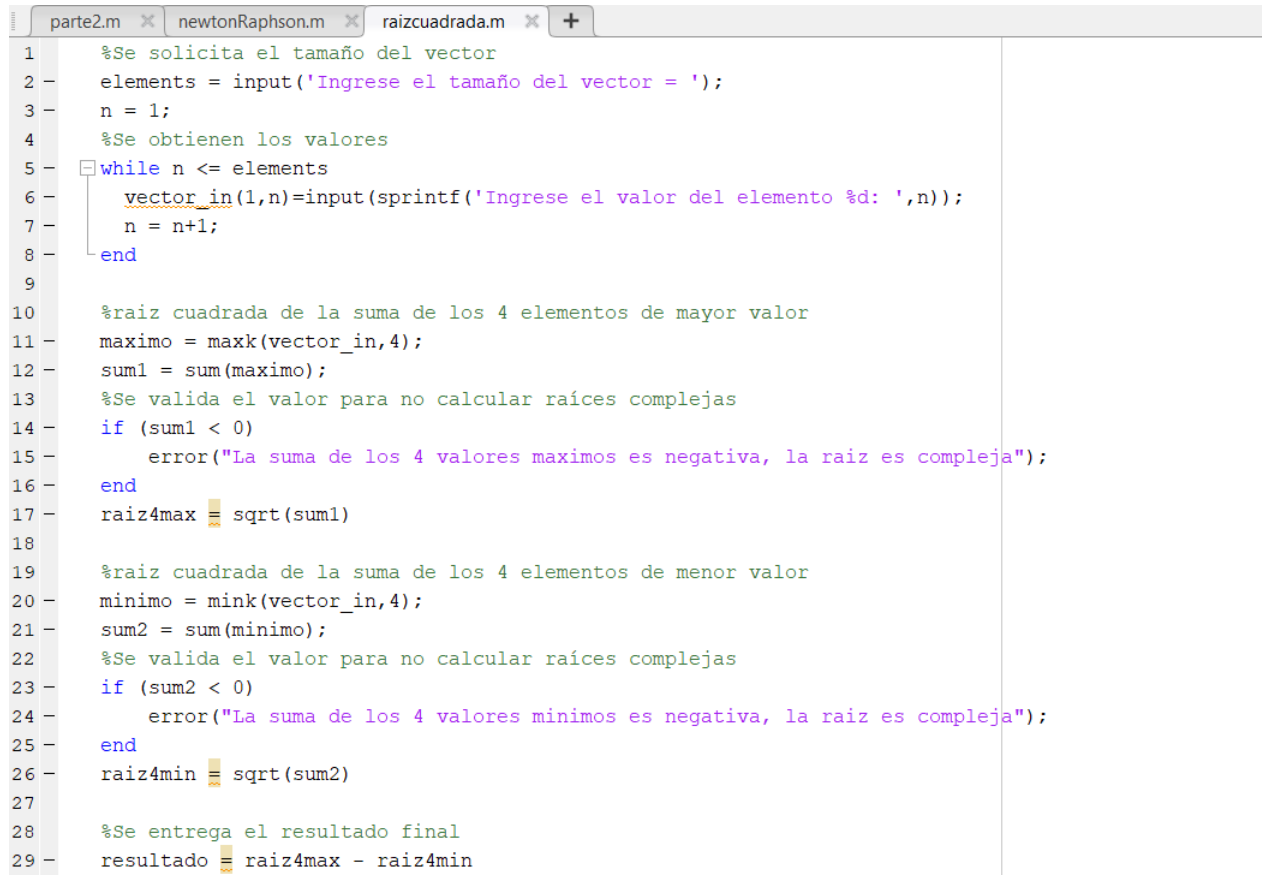
Luego se definen variables con sus valores iniciales, como el número de iteración actual igual a cero, una tolerancia aceptada como el error entregado mas uno y el valor requerido (x_a) para comenzar a aplicar el algoritmo indicado. Mientras se cumpla la condición de que las iteraciones requeridas no se han completado y que la tolerancia sea mayor al error solicitado, el algoritmo funcionará actualizando sus valores hasta que alguna de las condiciones no se cumpla.

Una vez que el algoritmo no pueda seguir, se almacenan los valores obtenidos asignando como el resultado final el xs . Las líneas 19 y 20 tienen la función de modificar el resultado para entregar un número aproximado y fácil de identificar, finalmente se entrega la solución del algoritmo para los parámetros entregados inicialmente.

3.2.2. Algoritmo de las raíces del vector

Para este caso se solicita entregar un vector por consola para retornar como resultado la sustracción entre la raíz cuadrada de la suma de los 4 elementos de mayor valor y la raíz cuadrada de la suma de los 4 elementos de menor valor.

El algoritmo implementado se muestra en la siguiente imagen, y se explicará a continuación.



```
1 %Se solicita el tamaño del vector
2 elements = input('Ingrese el tamaño del vector = ');
3 n = 1;
4 %Se obtienen los valores
5 while n <= elements
6     vector_in(1,n)=input(sprintf('Ingrese el valor del elemento %d: ',n));
7     n = n+1;
8 end
9
10 %raiz cuadrada de la suma de los 4 elementos de mayor valor
11 maximo = maxk(vector_in,4);
12 sum1 = sum(maximo);
13 %Se valida el valor para no calcular raíces complejas
14 if (sum1 < 0)
15     error("La suma de los 4 valores maximos es negativa, la raiz es compleja");
16 end
17 raiz4max = sqrt(sum1)
18
19 %raiz cuadrada de la suma de los 4 elementos de menor valor
20 minimo = mink(vector_in,4);
21 sum2 = sum(minimo);
22 %Se valida el valor para no calcular raíces complejas
23 if (sum2 < 0)
24     error("La suma de los 4 valores minimos es negativa, la raiz es compleja");
25 end
26 raiz4min = sqrt(sum2)
27
28 %Se entrega el resultado final
29 resultado = raiz4max - raiz4min
```

Figura 7: Código del algoritmo de las raíces del vector

Se solicita el número de elementos que tendrá el vector, luego se solicitan los valores de este de manera independiente y se almacenan en una variable. Luego, con funciones propias de MATLAB se obtienen los cuatro valores máximos dentro del vector, los cuales se suman y luego se obtiene su raíz. Por otro lado, se hace lo mismo para los cuatro valores mínimos. Finalmente, se realiza la sustracción de estas dos raíces para así entregar el resultado por pantalla.

Es importante señalar que se verifica que tanto la suma de los cuatro valores mínimos como la de los cuatro valores máximos sea mayor o igual a cero, ya que para este algoritmo solo se consideran las raíces positivas para evitar entrar al dominio de los números complejos.

Nota: al momento de ingresar los valores del vector, estos deben ser numéricos. Ya que si se entrega una letra el programa indicará que hay un error, aunque no se pudo implementar un mensaje de error específico ya que MATLAB reconocía la variable como indefinida antes de entrar al if de la validación.

4. Manual de uso

Esta sección da a explicar como usar el programa para utilizar el método de Newton Raphson y el archivo `raizcuadrada.m` que permite hacer la raíz cuadrada de los 4 numeros de mayor valor menos la raíz cuadrada de los 4 numeros de menor valor.

4.1. Primer ejemplo de Newton Raphson:

1. Abra el archivo `parte2.m` en MATLAB
2. Oprima Runz observe la terminal, que se encuentra en la parte inferior de la pantalla
3. Se le pedira ingresar el polinomio como vector, esto quiere decir, que debe ingresar los coeficientes separados (por espacios o comas) en un paréntesis de corchetes, por ejemplo, si se ingresa `[1,2,3,4]` se obtendría el siguiente polinomio:

$$1 + 2x + 3x^2 + 4x^3 \quad (5)$$

4. Luego se le pide el numero de iteraciones, por favor ingrese solo numeros y no letras, o no funcionará. Tenga en cuenta que entre más iteraciones entregue más preciso será el método.
5. Se le pedirá la tolerancia de error, esto es el margen de error que desea que el algoritmo tenga. Ingrese los porcentajes en forma de decimal o potencia de 10 (10 elevado al valor).
6. Finalmente, se le solicitará el valor inicial de la función, luego de escribirlo y presionar Enter se mostrará el resultado obtenido por la consola inferior.

4.2. Segundo ejemplo de Newton Raphson

Para este ejemplo se utilizará un ejercicio real y se representará mediante imágenes la manera de usar el algoritmo.

1. Al abrir el archivo *parte2.m* y seleccionar *Run* en la parte superior, se puede apreciar en la consola inferior que el algoritmo comenzó a funcionar y se empiezan a solicitar los datos para obtener el resultado.

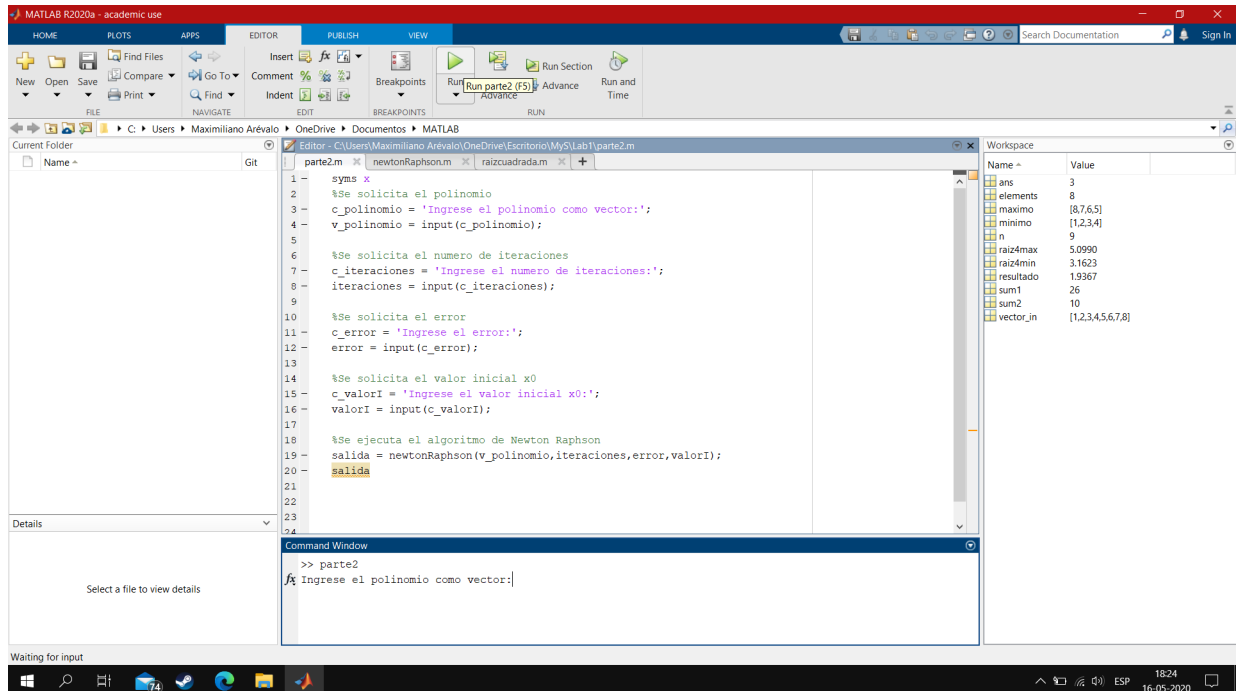


Figura 8: Ejemplo de ejecución Newton Raphson

2. En las siguientes imágenes se puede identificar que se solicitan los valores correspondientes para cada parámetro, ingresando *Enter* luego de ingresar los valores.



Figura 9: Código del algoritmo de las raíces del vector

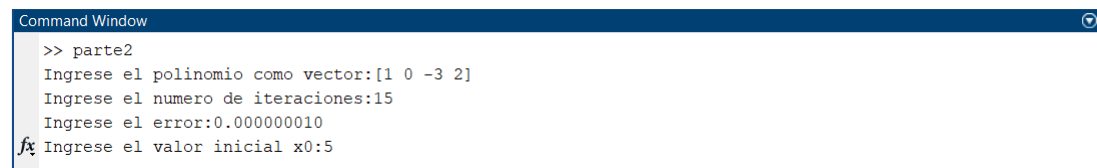
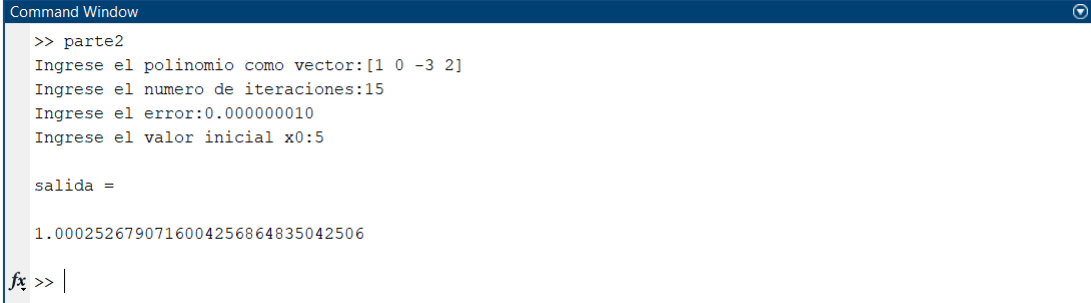


Figura 10: Ingreso de valores correspondientes

3. Luego de ingresar el ultimo valor el algoritmo muestra el resultado obtenido al aplicar el método de Newton Raphson para los valores y función entregados.



```
Command Window
>> parte2
Ingrese el polinomio como vector:[1 0 -3 2]
Ingrese el numero de iteraciones:15
Ingrese el error:0.000000010
Ingrese el valor inicial x0:5

salida =

1.0002526790716004256864835042506

fx >> |
```

Figura 11: Resultado obtenido

4.3. Ejemplo para algoritmo de raíces cuadradas:

1. Abra el archivo *raizcuadrada.m* en MATLAB
2. Oprima *Run* y vea la terminal, le aparecerá un mensaje como el de la siguiente imagen:



```
Command Window
>> raizcuadrada
fx Ingrese el tamaño del vector = |
```

Figura 12: Inicio del algoritmo de las raíces

3. En donde deberá ingresar el tamaño del vector, es decir, el número de elementos que posee. Luego deberá ingresar los valores del vector uno a uno, se deben ingresar valores numéricos, ya que si se ingresa una letra el algoritmo solicitará nuevamente un valor numérico para esa posición del vector.



```
Command Window
Ingrese el tamaño del vector = 8
Ingrese el valor del elemento 1: 10
Ingrese el valor del elemento 2: 20
Ingrese el valor del elemento 3: 30
Ingrese el valor del elemento 4: 40
Ingrese el valor del elemento 5: 50
Ingrese el valor del elemento 6: 60
Ingrese el valor del elemento 7: 70
fx Ingrese el valor del elemento 8: 80
```

Figura 13: Ingreso de los valores del vector

4. Luego de ingresar todos los valores del vector, el algoritmo le mostrará el resultado de la suma de la raíz cuadrada de los cuatro elementos de mayor valor, los de menor valor y el resultado de la resta entre ambos.



```
Command Window
raiz4max =
    16.1245

raiz4min =
    10

resultado =
    6.1245

fx >>
```

The image shows a MATLAB Command Window with a dark blue header. The text inside is white. It displays the results of three assignments: 'raiz4max' is 16.1245, 'raiz4min' is 10, and 'resultado' is 6.1245. At the bottom, there is a prompt 'fx >>'.

Figura 14: Resultados obtenido

5. Conclusiones

A partir de los gráficos se puede ver que con las herramientas de MATLAB se pueden realizar distintos tipos de gráficos, en más de un tipo de escala y marcar una región deseada a evaluar. Al mismo tiempo se pueden diferenciar las funciones graficadas si se quieren mostrar múltiples curvas en un mismo plano. Por otro lado el aplicar escala normal y logarítmica a una misma función exponencial ayuda a diferenciar su comportamiento como curva, pero también se puede ver mejor como van cambiando los valores y crece con la escala logarítmica.

Con respecto a la segunda parte se puede mencionar que el método de Newton-Raphson es una buena alternativa para realizar el cálculo de las raíces de una función, pero es importante señalar que este método posee ventajas y desventajas. Dentro de las ventajas se puede mencionar que puede ser rápido en comparación a otros frente a determinadas condiciones y es eficiente al utilizar ecuaciones no lineales. Sin embargo es necesario conocer la primera derivada de la función, lo que en ciertos casos puede ser algo complejo o demoroso de obtener e incluso provoca que el desarrollo del algoritmo sea muy lento dependiendo de la complejidad de las derivadas de la función. Además, pueden existir casos en los que no se pueda obtener una raíz o aproximación producto de la no convergencia en la aplicación del método, junto con señalar que es ineficiente en ecuaciones lineales.

En general se pueden ver las herramientas que ofrece MATLAB para el trabajo de funciones y ver como estas se comportan en distintos intervalos y en distintas escalas, lo cual complementa las observaciones posibles. Por otro lado sus funciones ayudan a simplificar la aplicación de algoritmos como lo es el Método de Newton Raphson que requiere múltiples recursiones, y también la facilidad de manipular vectores y matrices para hacer más rápidos los cálculos que se necesiten.

Finalmente se puede decir que los objetivos del laboratorio fueron cumplidos con éxito, ya que se logró representar de manera gráfica funciones en escala normal y logarítmica, comprender las diferentes formas de representar gráficos en MATLAB, aprender a implementar un algoritmo para obtener raíces como el de Newton Raphson e implementar un algoritmo para obtener una solución en base a determinados valores de un vector.

Bibliografía