UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

Departamento de Informática

Modelación y Simulación

Laboratorio 1

Gustavo Hurtado

Patricia Melo

Profesor: Gonzalo Acuña

Ayudante: Diego Mellis

Santiago – Chile

2020

TABLA DE CONTENIDO

ÍNDICE DE ILUSTRACIONES

CAPÍTULO 1. INTRODUCCIÓN

Este laboratorio es una introducción al lenguaje de programación MATLAB y a todo su entorno. En él se representarán diferentes tipos de funciones matemáticas con sus respectivos gráficos, tanto en escalas normales, como en escalas logarítmicas. Por otro lado, se realizará la implementación del algoritmo de Newton Rapson para la obtención de raíces de un polinomio, éste se hará de forma recursiva. Finalmente se realizará la implementación de una función que mediante el ingreso de un vector numérico, sumará y obtendrá la raíz cuadrada de los 4 mayores valores del vector y le restará la suma de los 4 menores valores del vector en raíz cuadrada.

El objetivo de este laboratorio es familiarizarse con el lenguaje de programación MATLAB, haciendo uso de sus herramientas y aprendiendo tanto su sintaxis como su lógica.

El informe consta de un marco teórico, donde se verán las bases para entender lo que será utilizado durante el laboratorio, además, cuenta del desarrollo de la primera parte, en la que se muestra la creación de cada gráfico, mientras que en el desarrollo de la segunda parte se observa la implementación de las funciones mencionadas anteriormente. Luego se incluye un manual de uso con ejemplos y finalmente las respectivas conclusiones y referencias del laboratorio.

CAPÍTULO 2. MÁRCO TEÓRICO

A continuación se definirán algunos conceptos para tener un mayor entendimiento en el informe.

1. **MATLAB**

   Como Juan Pablo Requez (2017) menciona, MATLAB es un programa computacional que ejecuta una gran variedad de operaciones y tareas matemáticas. Trabaja con matrices y vectores, puede resolver varios problemas matemáticos como por ejemplo simples ecuaciones o sistemas de ecuaciones diferenciales, entre otros. Permite graficar funciones y así presentar los resultados de una manera más ilustrativa.

2. **Newton Raphson**

   "Sería deseable disponer de una vía sistemática y fiable de construir un modelo $x = g(x)$ para hallar la solución $\overrightarrow{x}$ de la ecuación $f(x) = 0$, comenzando desde cualquier x_0 próximo a la solución y sin preocuparnos de $|g'(x) = \overrightarrow{x}|$ "
   (José Luis de la Fuente, 2017, p. 40).

   Considerando lo mencionado en la cita anterior, Newton Raphson nace bajo la necesidad de obtener raíces de una función $f(x)$. La idea consiste en reemplazar la función $f(x)$ en cada punto del proceso iterativo por el modelo de ella que define su recta tangente en ese punto (Ver figura 2.1), lo que lleva la siguiente fórmula: $x = x_1 - \frac{f(x_1)}{f'(x_1)}$. Como se puede apreciar, esta es una función de carácter recursivo, ya que va utilizando sus mismos valores.

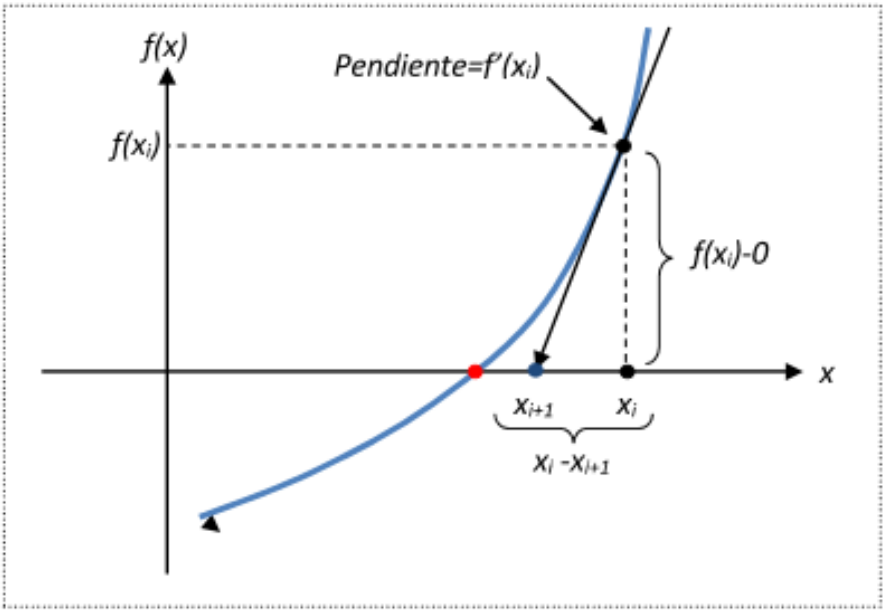

Figura 2.1: Interpretación gráfica de Newton Raphson

CAPÍTULO 3. DESARROLLO PRIMERA PARTE

En esta sección se explicará cómo fue el desarrollo de cada función y su respectivo gráfico, y que es lo que permiten ver y analizar cada uno.

1. **Graficar por separado y en conjunto en el dominio de $[0, 15\pi]$, las siguientes funciones**:

   - $a(x) = 8log_5(4x + 12)$
   - $b(x) = sen(6(log_2(x + 9))) + cos(7(log_6(4x + 32)))$

   Primero se definió el dominio como un vector, y también se procedió a definir las variables log_5 y log_6, esto debido a que logaritmo en base distinto a 10 o 2 no existen en MATLAB.

```
x = [0:0.01:15*pi];

log5 = log(4*x+12)/log(5);
log6 = log(4*x+32)/log(6);
```

   Luego se definieron las funciones $a(x)$ y $b(x)$

```
a = 8*log5;
b = sin(6*(log2(x+9)))+cos(7*(log6));
```

   Para crear varios gráficos y que MATLAB no los cierre luego de ejecutarlos se debe poner $figure$ al inicio de cada uno.

```
figure
plot(x, a, 'r *')
ylabel('Eje y');
xlabel('Eje x [0, 15*pi]');
title('Grafico funcion a(x)= 8log5(4x+12)');
```

Como se aprecia en la línea dos, plot permite crear el gráfico, en el dominio de x, con la función a descrita antes, y que sea graficado con color rojo y el símbolo $*$. Además, en las líneas siguientes se nombran los ejes del gráfico y se le da un título. A continuación se muestra el gráfico generado tras la ejecución.

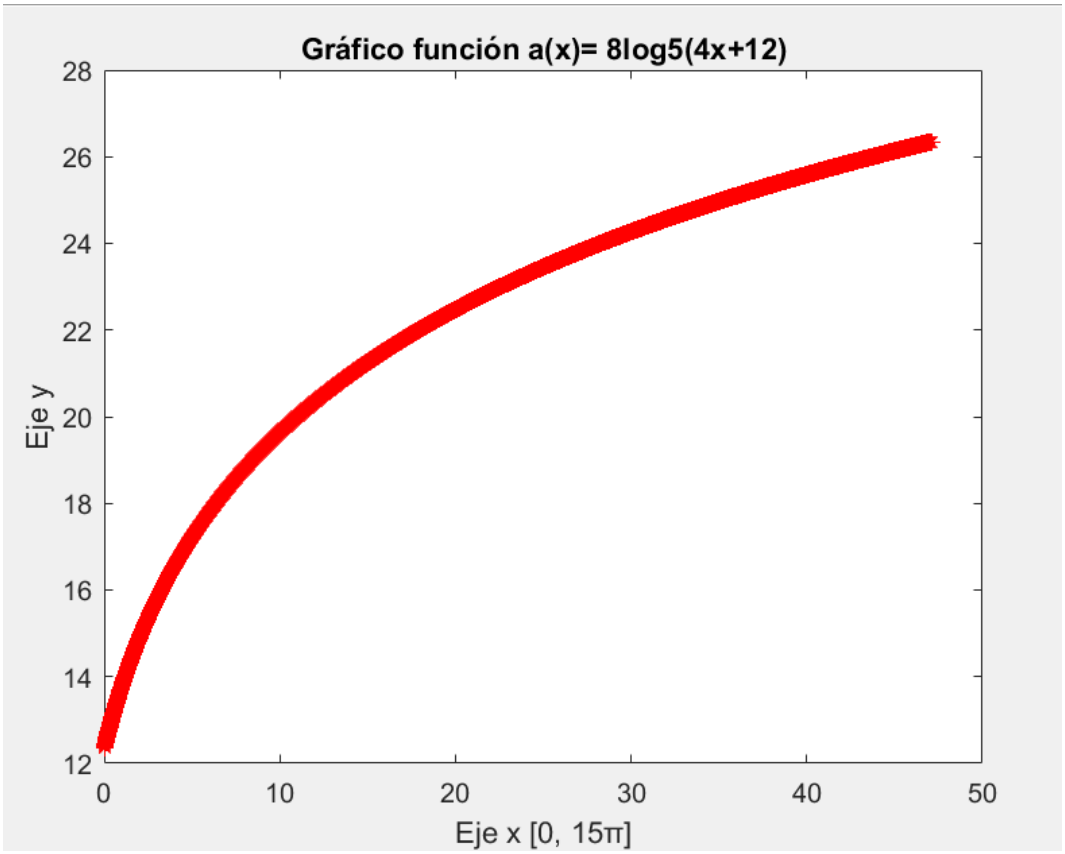

Figura 3.1: Gráfico de la función a(x) generado por MATLAB

El segundo gráfico solicitado es similar al primero, solo que con color verde y símbolo +, utilizando la función $b(x)$ descrita al inicio.

```
1    figure
2    plot(x, b, 'g +')
3    ylabel('Eje y');
4    xlabel('Eje x [0, 15*pi]');
5    title('Grafico funcion b(x)= sin(6(log2(x+9)))+cos(7(log6(4x+32)))');
```

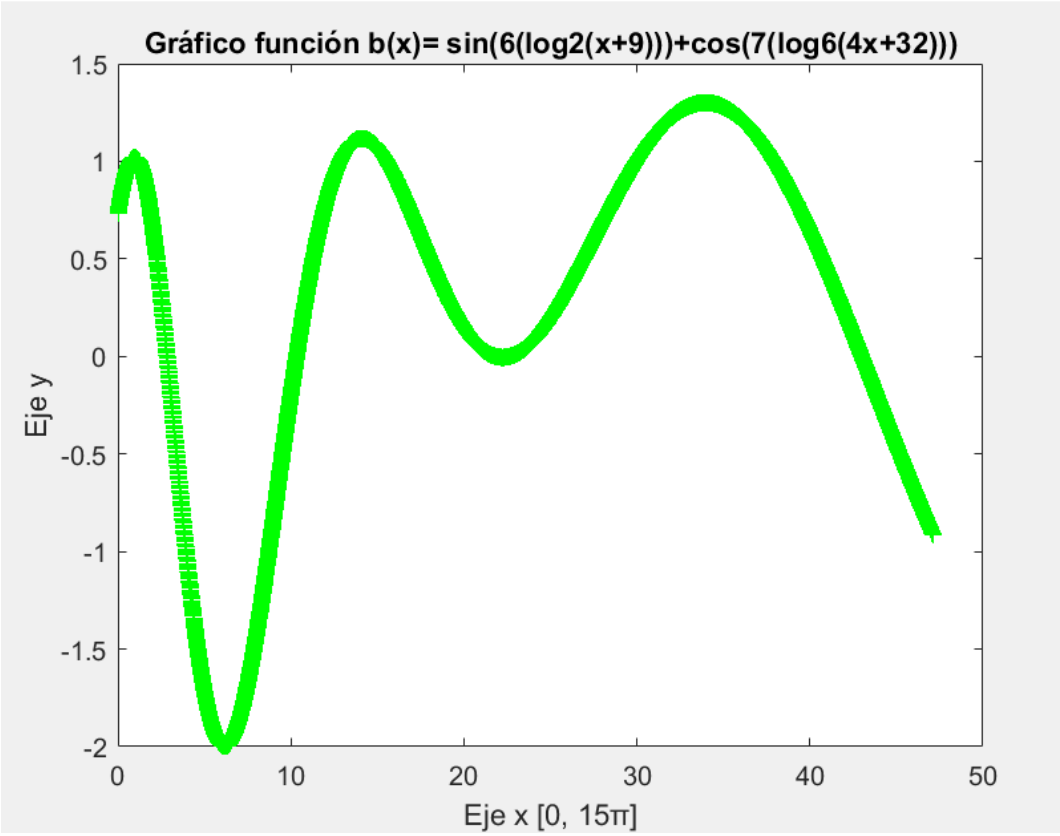

Figura 3.2: Gráfico de la función b(x) generado por MATLAB

El tercer gráfico une las funciones $a(x)$ y $b(x)$, con los mismos colores y símbolos vistos antes. Para identificar a qué función corresponde cada línea del gráfico se utiliza $legend$, además en este se indica que va en el extremo sur oeste del gráfico. Cabe destacar que el cuadro agregado por $legend$ se puede mover a cualquier parte del gráfico, evitando así que tape alguna función.

```
figure
plot(x, a, 'r *', x, b, 'g +')
ylabel('Eje y');
xlabel('Eje x [0, 15*pi]');
title('Grafico funcion a(x) y b(x)');
legend('Funcion a(x)', 'Funcion b(x)', 'Location', 'southwest');
```

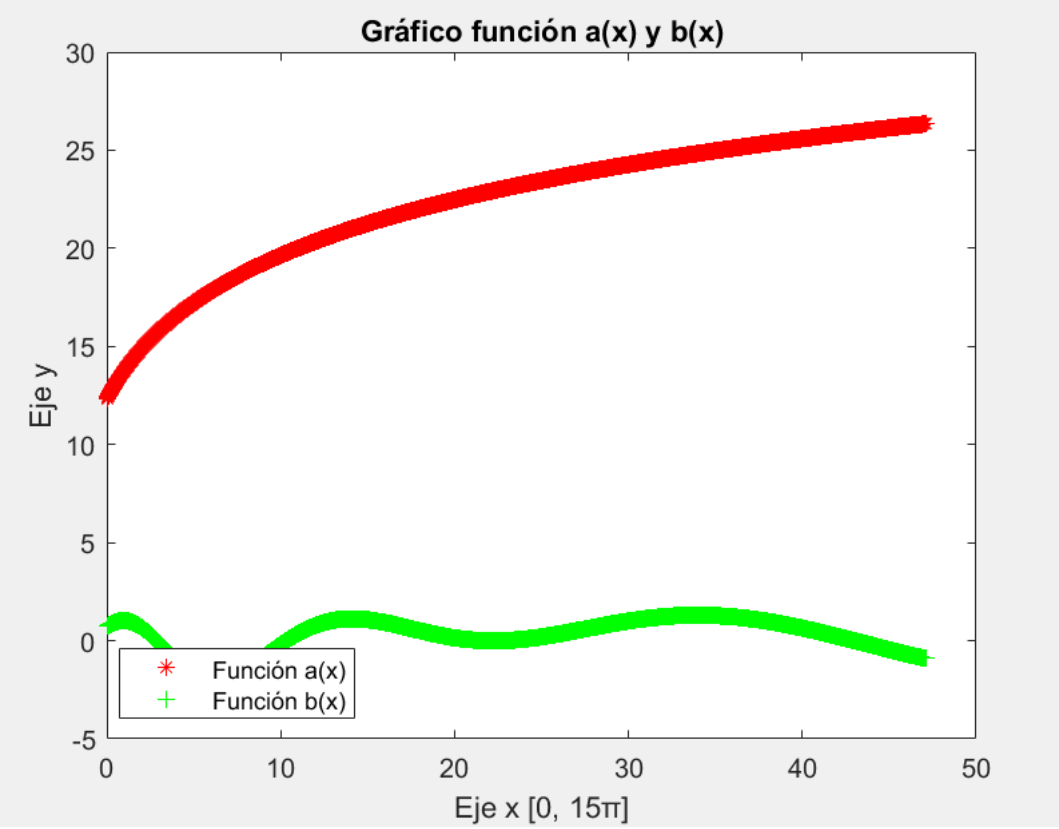

Figura 3.3: Gráfico de las funciones a(x) y b(x) generado por MATLAB

2. **Graficar en escala normal y logarítmica en el dominio de $[-10, 10]$ con espaciado de 0.01 la siguiente función**

   - $c(x) = 6 * e^{(x+18)}$

   En primer lugar, para desarrollar esta parte es necesario definir el dominio de la función, como se ve en la línea 1 del código que se encuentra más abajo. Luego se define también la función $c(x) = 6 * e^{(x+18)}$ en la línea 2. En esto se puede apreciar que en vez de usar e, se hace uso de la función que posee MATLAB llamada exp.

   En las líneas 3 y 4 llama a las funciones para realizar tanto el gráfico con base normal, como el gráfico con base logarítmica, respectivamente.

```
1        x = -10:0.05:10;
2        c = 6*exp(x + 18);
3        graph_normal(x,c)
4        graph_log(x,c)
```

   La función $graph\_normal(x, c)$ se puede apreciar en el código que se verá a continuación. Se puede observar que la función recibe los parámetros X1 e Y1, siendo X1 el dominio de la función, e Y1 la función como tal.

   En este caso, se define una nueva figura para que no se cierre el gráfico al ejecutarse, luego en las las líneas 3 y 4 se crean los ejes, mientras que en la línea 5 se genera el gráfico como tal, teniendo como entrada el dominio de la función y la función. En las líneas 6, 7 y 8 se genera primero la etiqueta del eje y, luego la del eje x y posteriormente el título del gráfico. Finalmente se agregan opciones extras, como la de mostrar el cuadriculado.

```
1        function graph_normal(X1, Y1)
2            figure1 = figure;
3            axes1 = axes('Parent',figure1);
4            hold(axes1,'on');
5            plot(X1,Y1);
6            ylabel('Eje y normal');
7            xlabel('Eje x [-10:10]');
```

```
8           title('Grafico Funcion c(x) = 6 * e^{x+18} base normal');
9
10          box(axes1,'on');
11          grid(axes1,'on');
12          hold(axes1,'off');
```

El resultado entregado por esta función se puede apreciar en la figura 3.4.

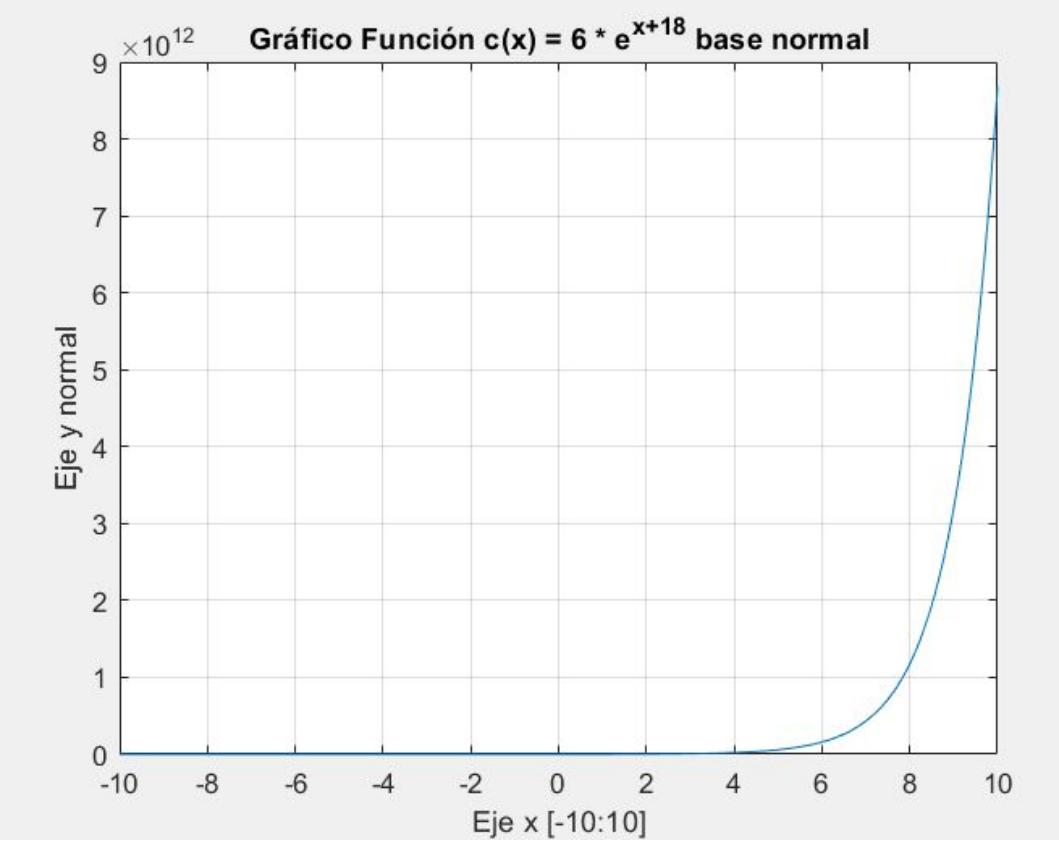

Figura 3.4: Gráfico de la función c(x) generado por MATLAB con base normal

Luego la función $graph\_log(x, c)$ se puede apreciar en el código que se mostrará más abajo. Al igual que $graph\_normal(x, c)$, la función recibe los parámetros X1 e Y1, siendo X1 el dominio de la función e Y1, la función como tal.

La principal diferencia con la función anterior, es que en vez de usar plot, hace uso de semilogy, que básicamente es un plot que aplica logaritmo al eje y, es decir, a la función $c(x)$.

```
function graph_log(X1, Y1)
    figure
    semilogy(X1,Y1)
    ylabel('Eje y logaritmico');
    xlabel('Eje x [-10:10]');
    title('Grafico Funcion c(x) = 6 * e^{x+18} con logaritmo');
    grid on;
```

En la figura 3.5 se puede apreciar el gráfico de la función $c(x)$, pero aplicando logaritmo al eje Y.

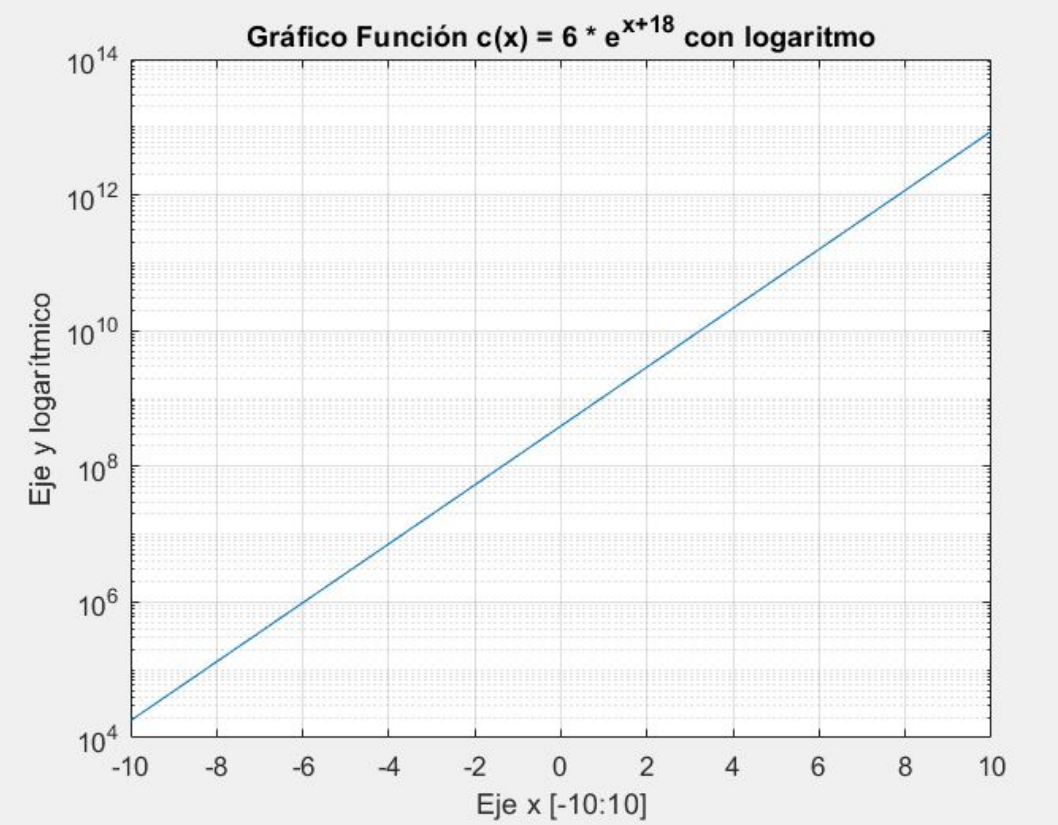

Figura 3.5: Gráfico de la función c(x) generado por MATLAB con base logarítmica

A modo de comparación entre los gráficos de la figura 3.4 y 3.5, se puede apreciar cómo cambia totalmente la forma del gráfico, ya que mientras una es una función exponencial, la otra es simplemente una recta. Claro, esto sucede por la naturaleza de la función $c(x)$. Aún así, se puede ver claramente como ambas funciones son crecientes a lo largo del dominio.

Como ventaja del gráfico normal, se tiene que es más fácil entender el hecho de que algo crezca de manera exponencial, es decir, de manera más lenta en un principio, pero luego de manera muy acelerada, mientras que esto no se ve en el gráfico logarítmico, ya que pareciera ser algo constante. Sin embargo, en la figura 3.4 pareciera que desde -10 a 4, el valor de la función es 0, y que luego se ve un crecimiento de los valores, pero no es más que un efecto causado por la enorme magnitud del eje y, que en este caso, es del orden de 10^{12}. Esto no ocurre en el gráfico logarítmico, ya que ahora no sólo se ve la magnitud 10^{12}, si no, que va desde 10^4 hasta 10^{13} aproximadamente, lo que permite mostrar los valores dentro del gráfico.

CAPÍTULO 4. DESARROLLO SEGUNDA PARTE

A lo largo de este capítulo se verá el desarrollo de las funciones solicitadas, que en este caso son Newton Rapson y la función parte 2.2 .

1. **Implementación Newton Raphson recursivo**

   La función de Newthon Raphson como se mencionó en el marco teórico, de por sí hace uso de una recursividad para hacer el cálculo de los resultados, por lo que su implementación algorítmica de manera recursiva resulta natural.

   Se puede apreciar en el código que se encuentra a continuación, que la función recursiva recibe como parámetros de entrada un poliniomio (fx), la cantidad máxima de iteraciones o llamadas recursivas (max iter), la tolerancia de error (error) y un valor inicial por el cual partir la búsqueda de la raíz (x0), mientras que en resultado, se va almacenando el último valor calculado. Cabe mencionar que x0 sólo vale en un principio lo que se ingresa por consola, porque luego tomará el valor calculado en la recursión.

```
1   function [result] = recursive_newton_raphson(fx,max_iter,error,x0)
2       fx_value = polyval(fx,x0);
3       dfx = polyder(fx);
4       dfx_value = polyval(dfx,x0);
5
6       %Criterio de parada
7       if abs(fx_value) < error || max_iter == 0
8           result = x0;
9       else
10          %Calculo del resultado
11          result = x0 - fx_value/dfx_value;
12          max_iter = max_iter - 1;
13          %Llamada recursiva
14          result = recursive_newton_raphson(fx,max_iter,error,result);
15      end
16  end
```

En la línea 2 se calcula el valor del polinomio (fx) en x0. Luego en la línea 3 se hace el

cálculo de la derivada de fx y en la siguiente se evalua esta derivada en x0.

Como criterio de parada se tiene que el valor absoluto del polinomio evaluado en el último resultado obtenido por la función recursiva, debe ser menor que la tolerancia de error, o que, la cantidad de iteraciones llegue a 0. Se entrega el valor inicial ingresado por consola como el resultado de la función, lo que termina con la recursión permitiendo a MATLAB calcular el valor de las recursiones anteriores.

Si no se entra al criterio de parada, entonces se calcula el valor de la raíz con la ecuación $x = x_1 - \frac{f(x_1)}{f'(x_1)}$ que fue vista en el marco teórico, se descuenta una iteración y luego se hace la llamada recursiva con el nuevo valor calculado en la línea 11.

Cabe mencionar que este proceso se realizará hasta que la tolerancia de error ingresada al comienzo sea menor que el error que se va obteniendo en cada iteración, o que se alcancen la cantidad máxima de iteraciones. Ahora, mientras mayor sea el valor máximo de iteraciones, más preciso será el resultado. Lo mismo ocurre con la tolerancia, mientras más pequeño sea el error aceptado, más preciso será el cálculo de la raíz. Por otro lado, el valor inicial x0 también es importante, ya que dependiendo de qué tan cerca o lejos se encuentre de la raíz a encontrar, puede cambiar drásticamente los resultados.

2. **Función parte 2.2**

La función creada en esta sección recibe un vector con números, luego calcula la raíz cuadrada de la suma de los cuatro elementos mayores, al resultado le resta la raíz cuadrada de la suma de los cuatro elementos menores del vector, como se muestra a continuación, considerando que x_1, x_2, x_3, x_4 son los elementos mayores y que x_5, x_6, x_7, x_8 los elementos menores.

$$función = \sqrt{x_1 + x_2 + x_3 + x_4} - \sqrt{x_5 + x_6 + x_7 + x_8}$$

Cabe mencionar que los elementos mayores y menores pueden ser los mismos, por lo que se necesitan como mínimo cuatro elementos en el vector.

El código creado para esta función se presenta a continuación.

```
function [resultado] = parte2_2(vector)

largo = length(vector);
resultado=0;
suma1=0;
suma2=0;

if isnumeric(vector) == 1
    if largo > 3
        suma1 = sumarCuatroMayores(vector);
        suma2 = sumarCuatroMenores(vector);
        resultado = sqrt(suma1) - sqrt(suma2);
        texto = sprintf('El resultado de la funcion es: %d', resultado
            );
        disp(texto);
    else
        disp('Ingrese un vector de 4 o mas elementos');
    end
else
    disp('Debe ingresar un vector que contenga solo numeros');
end
end
```

La línea 1 declara la función, luego la línea 3 obtiene el largo del vector, las siguientes inicializan variables.

Las validaciones del vector se encuentran en la línea 8 y 9, la primera se asegura que el vector contenga datos numéricos, la segunda verifica que el vector tenga cuatro o más elementos.

Luego de comprobar que el vector esta ingresado correctamente se procede a sumar los cuatro elementos mayores y menores del vector, llamando a las funciones $sumarCuatroMayores$ y $sumarCuatroMenores$, estas funciones serán explicadas más adelante. Continuando en la línea 12 se calcula el resultado final de la función, utilizando $sqrt$ para calcular la raíz cuadrada correspondiente. Para que el resultado se entregue por pantalla se utiliza la función $sprintf$ y $disp$.

En caso de que el vector no sea correcto, entonces se le avisará por pantalla al usuario su error.

La función $sumarCuatroMayores$ recibe un vector, luego con un contador i que inicia en 0 va en busca del elemento mayor del vector, esto lo hace con la función $max()$, lo suma en una variable llamada $suma$, después ubica la posición de dicho elemento con la función $find()$ y luego lo elimina, así vuelve a buscar el siguiente mayor, hasta que repita este ciclo cuatro veces, es decir, con $i = 4$. Finalmente entrega la suma obtenida.

```
1   function suma = sumarCuatroMayores(vector)
2   suma=0;
3   elemento=0;
4   indice=0;
5   i=0;
6   while (i<4)
7       i=i+1;
8       elemento= max(vector);
9       suma = suma + elemento;
10      %Obtengo el indice del elemento mas alto y lo elimino del vector.
11      indice = find(vector==elemento);
12      vector(indice) = [];
13  end
14  end
```

La función $sumarCuatroMenores$ es muy similar a la anterior, cambiando solamente la función $max()$ por la de $min()$ en la línea 8, de esta forma busca los cuatro elementos menores y los suma, entregando dicho resultado.

```
1   function suma = sumarCuatroMenores(vector)
2   suma=0;
3   elemento=0;
4   indice=0;
5   i=0;
6   while (i<4)
7       i=i+1;
8       elemento= min(vector);
9       suma = suma + elemento;
10      %Obtengo el indice del elemento mas bajo y lo elimino del vector.
11      indice = find(vector==elemento);
12      vector(indice) = [];
13  end
14  end
```

CAPÍTULO 5. ANEXOS

5.1 MANUAL DE USUARIO

En este capítulo se explicará como utilizar las funciones y gráficos creados.

1. **Parte 1.1**

   Esta sección corresponde a las funciones $a(x)$ y $b(x)$, para mostrar los gráficos de dichas funciones primero debe abrir el archivo $parte1\_1.m$ en MATLAB, luego existen dos formas de ejecutarlo, la primera es simplemente hacer click en el botón Run, la segunda es escribir en la consola $parte1\_1$ y apretar enter, como se aprecia en la figura 5.1. Cabe mencionar que los gráficos generados quedan uno sobre otro, para poder verlos todos se deben mover.

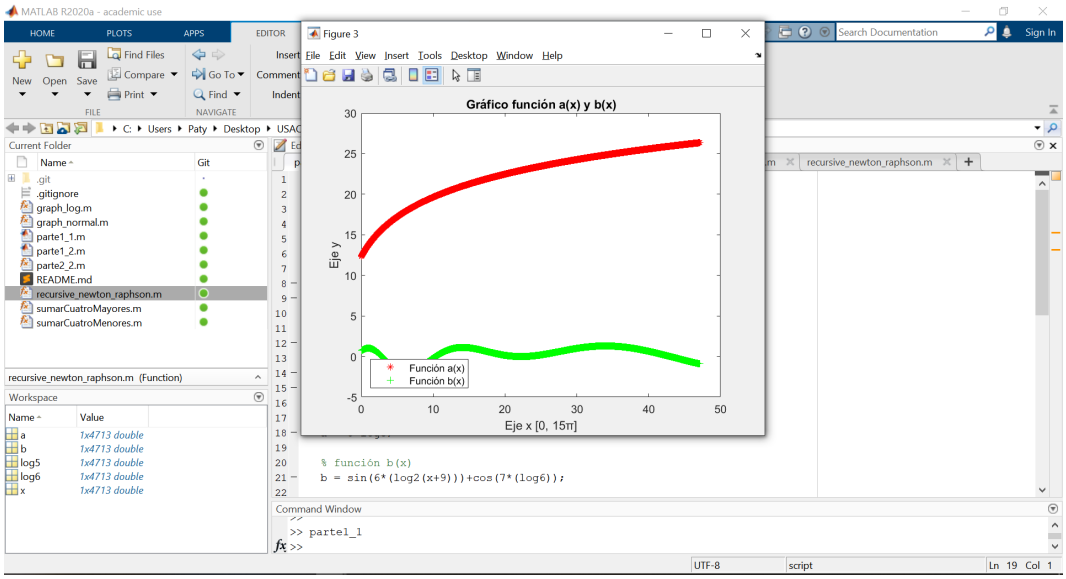

Figura 5.1: Ejecución del programa parte 1.1

2. **Parte 1.2**

   Para mostrar los gráficos correspondientes de la función $c(x)$ tanto en su versión con base normal y con base logarítmica, simplemente se debe abrir el archivo llamado $parte1\_2.m$ y ejecutarlo. Para esto existe 2 formas, una es presionar el botón Run que se encuentra en la parte superior de la interfaz de MATLAB, mientras que la otra es escribir en consola $parte1\_2$

como se puede apreciar en la figura 5.2 .

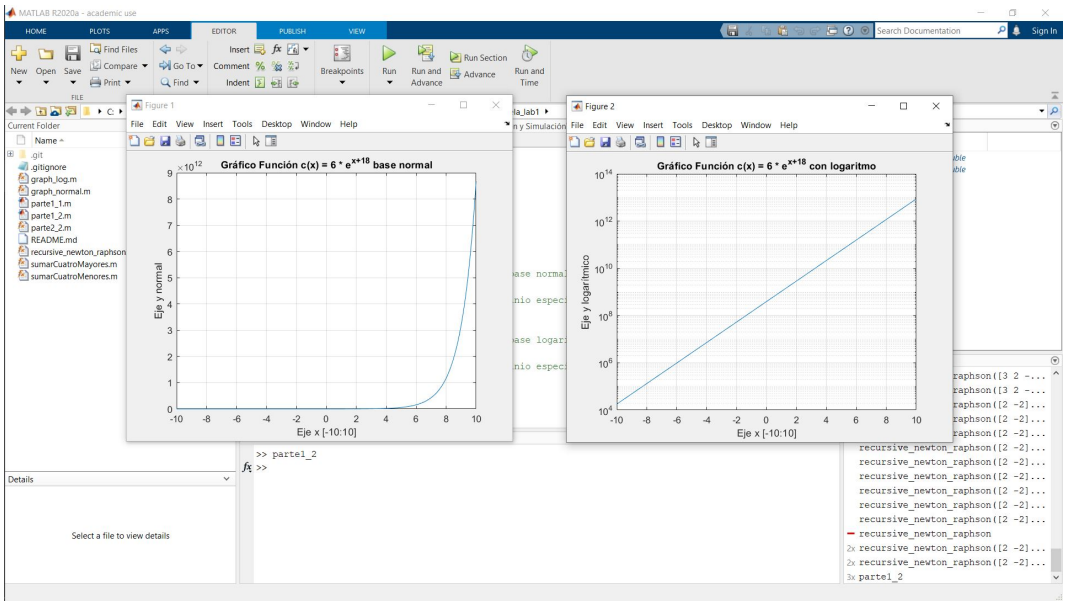

Figura 5.2: Ejecución del programa parte 1.2

Lo que se obtiene al ejecutar el programa son ambos gráficos de $c(x)$. Es posible que los gráficos aparezcan superpuestos uno sobre otro, por lo que simplemente se deben mover las ventanas y separarlos.

3. **Parte 2.1 Newton Raphson**

Para hacer uso de la función de Newton Raphson se debe abrir el archivo $recursive\_newton\_raphson.m$ y en la consola que entrega la interfaz de Matlab ingresar el nombre de la función y sus respectivos parámetros con el siguiente formato:

```
>> recursive_newton_raphson(Polinomio, Iteraciones , Tolerancia, Punto inicial)
```

Siendo un polinomio $x^3 + 2x - 1$ escrito de la forma [1 0 2 -1] por ejemplo, mientras que la cantidad de iteraciones debe ser un número mayor o igual a 0, al igual que la tolerancia. El punto inicial es un número real.

De esta manera se tienen 3 ejemplos diferentes para obtener una raíz de los polinomios $3x^2 + 2x^2 - 2$, $x^3 + 2x - 2$ y $2x - 2$ respectivamente.

- **Ejemplo 1**

```
>> recursive_newton_raphson([3 2 -2],100,10^(-7),0.5)
```

En este caso se tiene un polinomio escrito de la forma [3 2 -2], con 100 iteraciones máximas y una tolerancia al error de 10^{-7}, mientras que su punto inicial es el 0.5 .

Se obtiene como resultado $0{,}548583770355594$, tal como se puede apreciar en la figura 5.3 .

```
>> recursive_newton_raphson([3 2 -2],100,10^(-7),0.5)
ans =
   0.548583770355594
```

Figura 5.3: Resultado de Newton Raphson ejemplo 1 en consola.

- **Ejemplo 2**

```
>> recursive_newton_raphson([1 0 2 -2],100,10^(-4),1)
```

En la figura 5.4 se ve el resultado para este ejemplo. Considerando los mismos parámetros que para el ejemplo 1.

```
>> recursive_newton_raphson([1 0 2 -2],100,10^(-7),1)
ans =
   0.770916997059264
```

Figura 5.4: Resultado de Newton Raphson ejemplo 2 en consola.

- **Ejemplo 3**

```
>> recursive_newton_raphson([2 -2],10,10^(-7),-10)
```

En la figura 5.5 se ve el resultado para este ejemplo. Los parámetros son los mismos que en los casos anteriores, pero con diferentes valores.

```
>> recursive_newton_raphson([2 -2],10,10^(-7),-10)

ans =

     1
```

Figura 5.5: Resultado de Newton Raphson ejemplo 3 en consola.

4. **Parte 2.2 función**

Para ejecutar la función $parte2\_2$ primero debe abrir el archivo $parte2\_2.m$ en MATLAB. Como se necesita de un vector para trabajar esta función, entonces debe crearlo con antelación por consola y al momento de llamar la función ingresarlo.

```
>> vector = [2, 4, 6, 1, 3, 5, 7, 8, 9]
>> parte2_2(vector)
```

Otra forma es simplemente llamar la función y escribir el vector en el mismo instante.

```
>> parte2_2([2, 4, 6, 1, 3, 5, 7, 8, 9])
```

El resultado de lo escrito anteriormente se muestra por pantalla, como se aprecia en la siguiente figura.

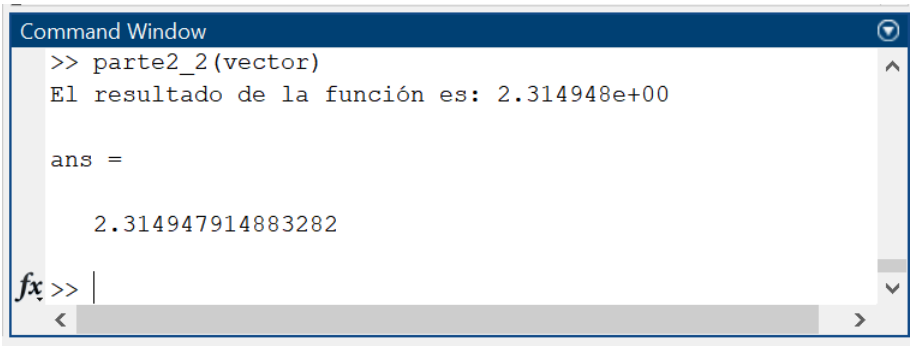

Figura 5.6: Ejemplo 1 de ejecución parte2_2

Algunos de los casos en donde el vector no es válido son los siguientes:

- Ingresar un vector que no contenga números.

```
>> vector = ['h', 'o', 'l', 'a']
>> parte2_2(vector)
```

Entregando como respuesta lo siguiente:

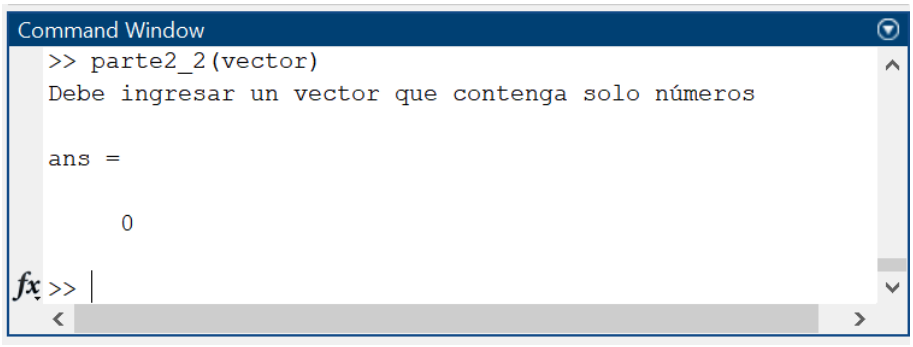

Figura 5.7: Ejemplo 2 de ejecución parte2_2

- Ingresar un vector que contenga menos de 4 elementos.

```
>> parte2_2([1, 2, 3])
```

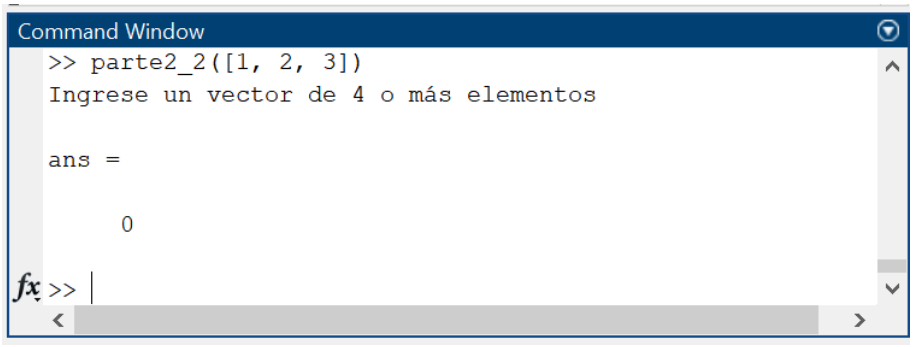

Figura 5.8: Ejemplo 3 de ejecución parte2_2

De esta forma se le indica al usuario cómo debe ir el vector para poder ocupar esta función.

CAPÍTULO 6. CONCLUSIONES

Gracias a esta experiencia se logró conocer más la herramienta MATLAB, entendiendo como utilizar vectores y funciones matemáticas. También se aprendió a gráficar diversas funciones, utilizando la escala normal y logarítmica en ellas y ver las diferencias que surgen una con respecto a la otra, además, se logró implementar nuestras propias funciones, es por este motivo que se puede afirmar que se cumplió el objetivo planteado en la introducción.

Con respecto a los resultados obtenidos, primero en la parte 1, la sección de gráficos, se puede decir que se generaron de manera exitosa, ya que estos gráficos siguen el comportamiento de las funciones, por ejemplo la función $a(x)$ es una función logarítmica, y su gráfico representa dicho comportamiento, lo mismo con las otras funciones, ya que esto también ocurre con $c(x)$, que es una función exponencial y su gráfico lo representa correctamente. En la parte 2, las funciones creadas (Newton Raphson y parte 2.2), se comprobaron que eran correctas, dado que los resultados de la primera se comparó con las raíces de los polinomios ingresados en Wolfram Alpha, obteniendo exactamente los mismos valores, teniendo en cuenta una cantidad de iteraciones y tolerancia de error buena, y la segunda función se comprobó con una calculadora a parte, realizando los cálculos uno a uno y llegando al mismo resultado que entrega MATLAB.

BIBLIOGRAFÍA

ACAPMI (2017). ¿qué es matlab? [Online] http://acapmi.com/blog/2017/09/18/que-es-matlab/.

de la Fuente O'Connor, J. L. (2017). Ingeniería de los algoritmos y métodos numéricos. [Online] http://www.jldelafuenteoconnor.es/Libro2017_NV_10-8_SP.pdf.

Mathworks (2012). Borrar elemento vector. [Online] https://es.mathworks.com/matlabcentral/answers/48938-delete-element-from-vector.

Mathworks (2015). How do i define the recursive function? [Online] https://la.mathworks.com/matlabcentral/answers/216376-how-do-i-define-the-recursive-function.

Mathworks (2019). Añadir leyenda en el gráfico. [Online] https://es.mathworks.com/help/matlab/creating_plots/add-title-axis-labels-and-legend-to-graph.html.

Mathworks (2020). Ver si un vector contiene datos numéricos. [Online] https://www.mathworks.com/help/matlab/ref/isnumeric.html.

Mathworks (sf). Buscar índice en vector matlab. [Online] https://la.mathworks.com/help/matlab/ref/find.html.