UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

Departamento de Informática

Modelación y Simulación

Laboratorio 1

Nicolás Alarcón

Isaac Espinoza

Profesor: Gonzalo Acuña

Ayudante: Francisco Muñoz

Santiago – Chile

2019

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

CAPÍTULO 1. INTRODUCCIÓN

Los modelos son representaciones abstractas de fenómenos, sistemas o procesos que suceden en el mundo real. Un modelo debe servir para analizar, describir, simular o predecir el fenómeno, sistema o proceso que representa, por lo tanto debe ser capaz de entregar salidas o resultados dependiendo de datos de entrada. Un modelo matemático requiere una formulación matemática, la cual puede ser complicada de manipular, por lo tanto existen muchas herramientas que facilitan esto. En este laboratorio se utilizará Matlab de forma introductoria para realizar gráficos y manipular funciones, ademas de familiarización con la sintaxis de su lenguaje de programación. Por lo tanto, en este informe se describirán los resultados correspondientes a la confección de 5 gráficos aplicando conceptos de escala normal y logarítmica, además de la implementación del algoritmo de Newton Raphson y el uso de la representación de polinomios de Matlab.

- Aprender sintaxis básica de Matlab.
- Practicar manejo de funciones y polinomios.
- Confeccionar 5 gráficos con distintas características.
- Implementación recursiva del método Newton Raphson para encontrar raíces.
- Manual de usuario indicando casos de uso para el programa resultante.

CAPÍTULO 2. MÁRCO TEÓRICO

- Escala logarítmica: Escala que utiliza los logaritmos de las magnitudes que se desean graficar. Se utiliza para poder apreciar claramente magnitudes con variaciones muy grandes entre cada punto a representar. [de Cantabria (2019)]

- Newton Raphson: Método numérico que se utiliza para buscar raíces en funciones mediante aproximaciones sucesivas a partir de un valor inicial x_0. Newton-Raphson utiliza la siguiente fórmula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2.1}$$

  Por lo tanto, se obtiene el valor siguiente de la aproximación utilizando el valor anterior y el resultado de la función divida en la derivada evaluadas en el valor anterior. [Larrosa (n.d.)]

- Matlab: Entorno de escritorio que utiliza un lenguaje de programación propio, que facilita la expresión de vectores, arreglos, matrices y proporciona herramientas matemáticas. [MathWorks (2019a)]

CAPÍTULO 3. DESARROLLO PRIMERA PARTE

La primera parte del laboratorio consiste en la confección de gráficos en MATLAB. Las funciones que deben graficarse son las siguientes:

- $a(x) = 12log_5(3x + 7)$
- $b(x) = sen(4(log_2(x + 8))) + cos(5(log_6(2x + 46)))$
- $c(x) = 7e^{(x+12)}$

Además, a(x) y b(x) deben ser evaluadas en el intervalo [0, 15pi] con espacios de 0.01 entre cada valor. Respecto a c(x), debe ser evaluada en el intervalo [-10,10] con espaciado de 0.05. El desarrollo de la primera parte se hizo completamente en IDE de MATLAB. Primeramente, se subdividieron las tareas en distintas funciones y por ende en varios archivos. Se definen los siguientes archivos:

- main.m: Archivo que contiene el ”main”de la parte 1. Se definen las variables, funciones, etc y además se llaman a las funciones correspondientes para realizar los gráficos.
- graficar.m: Archivo que contiene a la función del mismo nombre. Realiza los gráficos para las funciones a(x) y b(x).
- graficarSemilogy.m: Archivo que contiene a la función del mismo nombre. Realiza los gráficos en escala normal y logarítmica para la función c(x).

A continuación se explicará en mayor profundidad cada uno de los archivos y funciones utilizadas.

3.0.1 main.m

Como ya se mencionó, en el archivo main.m se hacen todas las declaraciones de variables correspondientes. Además, se utiliza el concepto de funciones anónimas para el manejo de las funciones matemáticas que deben ser graficadas. Según MATLAB, se definen las funciones anónimas como: ”Una función anónima es una función no almacenada en un archivo de programa,

pero está asociada a una variable cuyo tipo de datos es function_handle. Las funciones anónimas pueden aceptar entradas y salidas de retorno, tal como lo hacen las funciones estándar. Sin embargo, sólo pueden contener una única instrucción ejecutable.” MathWorks (2019b)

El uso del concepto de funciones anónimas no solo se utiliza para representar a las funciones propuestas a(x), b(x) y c(x), sino que también se usa para poder representar los logaritmos en base 5 y en base 6, debido a que estos no se incluyen en las funciones básicas de MATLAB. Una vez se definen las funciones a(x) y b(x), se crea un array que contiene los valores en los cuales las funciones serán evaluadas. En este caso, se utiliza simplemente la notación de vectores de MATLAB de la siguiente forma: **vector = [inicio, espaciado, fin]**

- Inicio: Se indica el valor inicial del vector.
- Espaciado: Se indica el valor de la distancia entre cada elemento del vector.
- Fin: Se indica el valor final del vector.

Luego, se evalúan a(x) y b(x) en el vector correspondiente y se grafican con la función ”graficar”. Finalmente, se realizan las mismas acciones para la función c(x) y se grafica con la función ”graficarSemilogy”.

3.0.2 graficar.m

Este archivo contiene a la función del mismo nombre. No posee salida y recibe cuatro argumentos.

- Función 1: Vector con los valores obtenidos al evaluar la función a(x).
- Función 2: Vector con los valores obtenidos al evaluar la función b(x).
- Valores x: Vector con los valores del eje x para ambas funciones.
- Índice de Figura

Primero, se crea la figura correspondiente a la función a(x), luego se le asignan valores al título, etiquetas de los ejes y leyenda del gráfico. Además se indica que la función a(x) debe graficarse

con puntos de la forma '*' y de color rojo. Se repiten los mismos pasos, pero esta vez para la función b(x), en donde simplemente se indica que debe graficarse como puntos de la forma '+' de color verde. Finalmente, se crea una tercera figura en la cual se grafican al mismo tiempo las funciones a(x) y b(x). De la misma forma, se les asignan sus colores y formas correspondientes, además de etiquetas de los ejes, título y leyenda.

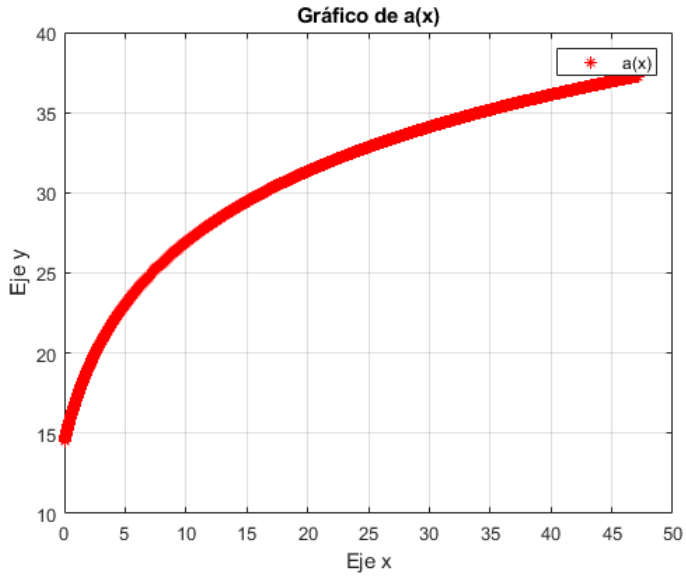

Figura 3.1: Gráfico de función a(x)

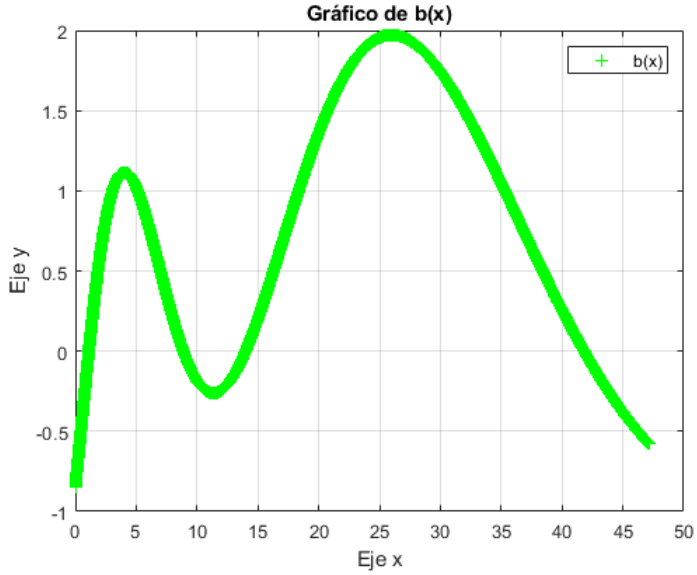

Figura 3.2: Gráfico de función b(x)

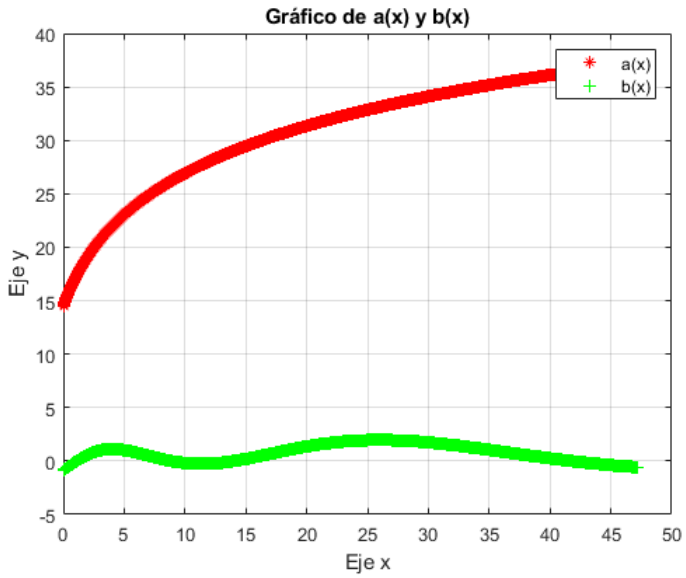

Figura 3.3: Gráficos de funciones a(x) y b(x)

3.0.3 graficarSemilogy.m

De la misma forma que para el caso anterior, este archivo contiene a la función del mismo nombre. No posee salida y recibe tres argumentos.

- Función 1: Vector con los valores obtenidos al evaluar la función c(x).
- Valores x: Vector con los valores del eje x.
- Índice de Figura

Primero se crea una figura y se grafican los valores de c(x) en escala normal con la función 'plot', además se setean los nombres de los ejes y se le asigna un título al gráfico. Luego, se crea una segunda figura, pero esta vez se grafica utilizando la función 'semilogy'. Esta función crea un gráfico aplicando logaritmo en base 10 al eje y, mientras que aplica una escala normal al eje x.

Respecto a los gráficos, en el caso específico de la función c(x), que consiste en una función exponencial, la escala logarítmica permite apreciar de mejor manera el comportamiento de c(x). En el intervalo [-10,4], no se puede observar casi ningún cambio a simple vista en escala normal, mientras que en escala logarítmica se aprecia claramente el crecimiento de la función

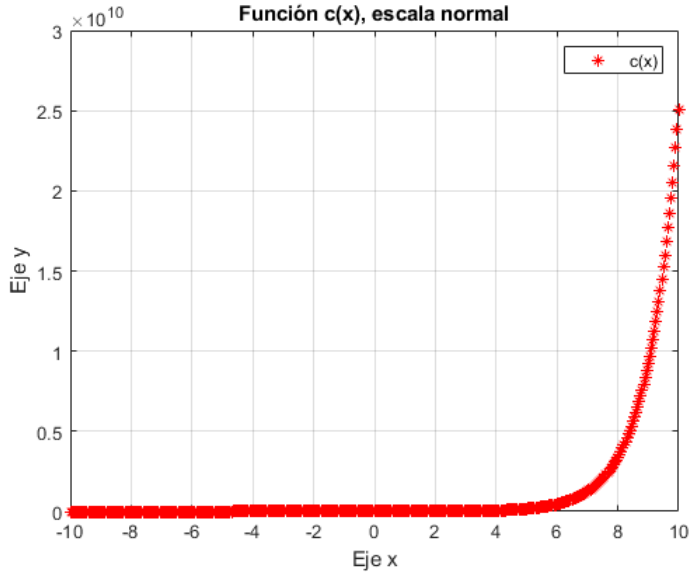

Figura 3.4: Gráfico de función c(x) en escala normal

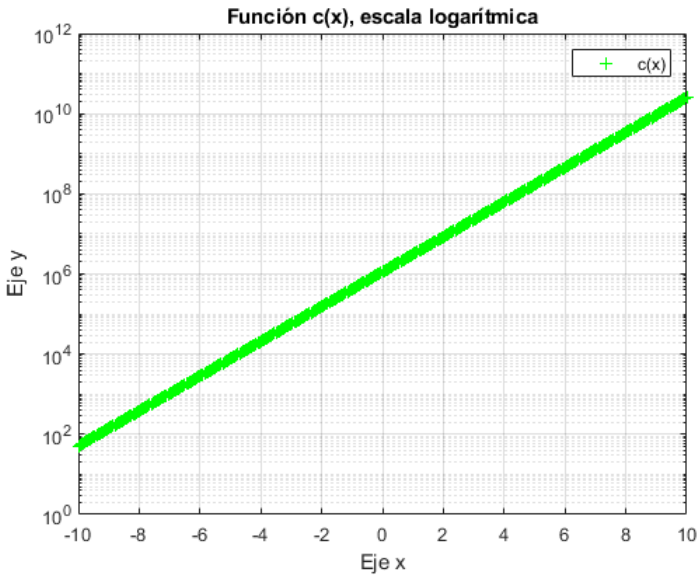

Figura 3.5: Gráfico de función c(x), escala logarítmica

en ese intervalo. Por lo tanto, la escala logarítmica puede servir para analizar funciones con comportamiento asintótico, como la función exponencial (posee una asíntota en el 0 del eje x).

CAPÍTULO 4. DESARROLLO SEGUNDA PARTE

4.0.1 Newton Raphson

La segunda parte del laboratorio consiste en Implementar el algoritmo de Newton-Raphson, que entregue como resultado una raíz de una función dada o una aproximación de ella. Las entradas de la función deben ser 4:

- El polinomio: Expresado en forma de vector con valores numéricos válidos y largo mayor a 0.
- Numero máximo de iteraciones: Número de iteraciones debe ser un entero mayor a 0, escalar.
- Error: Valor mayor a 0, escalar.
- Valor inicial (X0): Cualquier valor numérico escalar.

En el archivo main2parte1.m, se pide el ingreso de los 4 parámetros de Newton Raphson, los cuales son validados con los criterios ya mencionados.

A continuación se presenta el pseudocódigo de la función de Newton Raphson.

```
NewtonRaphson(funcion, iteracionesMaxima, ErrorMaximo, X0);
polinomioDerivado = polyder(funcion) %se derivada la funci n
x1 = x0 - evaluarPolinomio(X0)/polinomioDerivado(x0)
Error = abs ( x1 - x0)
if  Error < ErrorMaximo
  return x1
if else iteracionesMaxima == 0
  return x1
else
      iteracionesMaxima =  iteracionesMaxima -1
      %llamado recursivo
      NewtonRaphson(funcion, iteracionesMaxima, ErrorMaximo, X1);
```

El algoritmo intenta encontrar aproximaciones de los ceros o raíces de una función de entrada, para esto utiliza la fórmula de Newton Raphson (2.1).

Este proceso de búsqueda se realiza a través de un llamado recursivo al algoritmo y en cada iteración se busca obtener un cero más aproximado. Este proceso se repetiría la cantidad máxima de iteraciones ingresadas por el usuario o se verá condicionado por el error máximo admitido.

4.0.2 Resta de Raíces de Vectores

En la segunda parte del laboratorio se solicita realizar un programa que reciba como entrada un vector en el programa main2parte2.m (especificado en el anexo) y despliegue por pantalla el resultado de la raíz cuadrada de la suma de los 4 elementos de mayor valor, menos el resultado de la suma de la raíz cuadrada de los 4 elementos de menor valor. La función se encuentra en el archivo sumarFunciones.m, el cual es llamado en main2Parte2.m. Como requisito se debe manejar el ingreso erróneo de los valores del vector y de la cantidad de elementos del vector, lo cual se maneja en el main al pedir el vector como entrada del usuario.

A continuación se presenta el pseudocódigo del programa

```
sumarFunciones(vector)
 if el vector es menos a 4
    return ("El vector ingresado tiene menos de 4 elementos")
end
 %Ordenar el vector original de mayor a menor
 %Obtener los primeros 4 elementos de la lista ordenada
 %Sumar los primeros 4 elementos

 %Ordenar el vector original de menor a mayor
 %Obtener los primeros 4 elementos de la lista ordenada
 %Sumar los primero 4 elementos

 if Suma de los primero 4 elementos mayores <0
   return valores ingresados no son validos
 if Suma de los primero 4 elementos menores < 0
   return valores ingresados no son validos
end
return resultado = sqrt(suma de los mayores) - sqrt(suma de los menores)
```

CAPÍTULO 5. CONCLUSIONES

El laboratorio anterior permitió alcanzar el objetivo principal, el cual era acercar el manejo del lenguaje de programación MATLAB, resultando ser una herramienta de trabajo cientifico y matemático. Este tipo de problemas suelen ser complejos y necesitan gran precisión, pero al trabajar con el lenguaje y entorno de MATLAB se pueden programar problemas complejos con una sintaxis simple y sin sacrificar eficiencia o eficacia.

Durante el laboratorio se pudo implementar todos los requisitos de manera exitosa, además se pudo apreciar sobre la utilidad de la escala logarítmica para comportamientos asintóticos en algunas funciones y la facilidad de MATLAB para aplicar ese tipo de escalas a los gráficos solo con una línea de código. También, se logró implementar la función de Newton Raphson de manera recursiva la cual nos permitió encontrar las raíces de un polinomio y finalmente, se pudo implementar exitosamente una función que recibiera y manipulara vectores para obtener la suma de los 4 elementos mayores y menores, para posteriormente retornar la resta de ambas raíces cuadradas. Todas estas tareas pudieron realizarse aprovechando la sintaxis simple y las herramientas que MATLAB provee para el fácil tratamiento de funciones, vectores y polinomios.

CAPÍTULO 6. ANEXOS

6.1 CÓDIGOS

6.1.1 main.m

```
log5 = @(x) log(x) / log(5);

log2 = @(x) log(x) / log(2);

log6 = @(x) log(x) / log(6);

a = @(x) 12*log5(3*x+7);

b = @(x) sin(4*(log2(x+8))) + cos(5*(log6(2*x+46)))

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

resultadoA = a(x)

resultadoB = b(x)

graficar(resultadoA,resultadoB,1)

c = @(x) 7*exp(x+12)

x = [-10:.05:10]

resultadoC = c(x)

graficarSemilogy(resultadoC, 4)
```

6.1.2 graficar.m

```
function graficar(valores, valores2,f)

figure1 = figure(f);

axes1 = axes('Parent',figure1);

plot([valores],'r*');

grid on

title("Gráfico de a(x)")
```

```
xlabel('Eje x')

ylabel('Eje y');

legend('a(x)');

figure2 = figure(f+1)

axes2 = axes('Parent',figure2);

plot([valores2],'g+');

grid on

title("Gráfico de b(x)")

xlabel('Eje x')

ylabel('Eje y');

legend('b(x)');

figure3 = figure(f+2)

axes3 = axes('Parent',figure3);

box(axes3,'on');

hold(axes3,'all');

plot([valores],'r*'); hold on;

plot([valores2],'g+'); hold on;

grid on

title("Gráfico de a(x) y b(x)")

xlabel('Eje x')

ylabel('Eje y');

legend('a(x)',"b(x)");

end
```

6.1.3 graficarSemilogy.m

```
function graficarSemilogy(valores,f)

figure1 = figure(f);
```

```
axes1 = axes('Parent',figure1);


plot([valores],'r*');

grid on

title("Función c)")

xlabel('Iteraciones')

ylabel('Errores');

legend('Gauss-Jacobi');

figure2 = figure(f+1)

axes2 = axes('Parent',figure2);

semilogy([valores],'g+');

grid on

title("Función c), semilogy")

xlabel('Iteraciones')

ylabel('Errores');

legend('Gauss-Jacobi');

end
```

6.1.4 main2parte1.m

```
prompt = 'Ingrese el polinomio en su representación array:\n';

prompt1= 'Ingrese el valor inicial x0\n';

prompt2 = 'Ingrese el numero maximo de iteraciones\n';

prompt3 = 'Ingrese la tolerancia\n';

polinomio = [];

x0 = 'a';
```

```
iter = 'a';

err = 'a';

while length(polinomio) == 0 || isnumeric(polinomio) ==0

    try

        polinomio = input(prompt)


    catch

       disp("Error al ingresar polinomio, intentelo nuevamente")

    end

end


while isnumeric(x0) == 0|| isscalar(x0) == 0

    try

        x0 = input(prompt1)


    catch

       disp("Error al ingresar x0, intentelo nuevamente")

    end

end


while isnumeric(iter) == 0 || isscalar(iter) == 0 || iter <= 0 || mod(iter, 1) ~= 0

    try

        iter = input(prompt2)


    catch

       disp("Error al ingresar numero iteraciones, intentelo nuevamente")

    end

end
```

```
while isnumeric(err) == 0|| isscalar(err) == 0 || err < 0
    try
        err = input(prompt3)

    catch
       disp("Error al ingresar tolerancia, intentelo nuevamente")
    end
end

newtonRaphson(polinomio, iter, err, x0)
```

6.1.5 newtonRaphson.m

```
%Función Newton-Raphson.
%ENTRADA: * polinomio
%         * cantidad de iteraciones máxima
%         * Error maximo
%         * Valor inicial del algoritmo.
%SALIDA:la raíz del polinomio ingresado.
function [Respuesta] = newtonRaphson(fun, iteracionMax, Error,X0)
    DevFun = polyder(fun); %funcion que obtiene la derivada
    X1 = X0;
    Respuesta = X1 - (polyval(fun,X1)/polyval(DevFun,X1));
    ErrorR = abs(Respuesta - X1);
    if ErrorR < Error
        disp(Respuesta)
        return
```

```
    elseif iteracionMax == 0
        disp(Respuesta)
        return
    else
        iteracionMax  = iteracionMax-1;

        Respuesta = newtonRaphson(fun, iteracionMax, Error, Respuesta);

    end
end
```

6.1.6 main2parte2.m

```
prompt = 'Ingrese un vector con un número de elementos mayor a 4:\n';
vector = [];
while length(vector) == 0 || length(vector) < 4 || isnumeric(vector)==0
    try
        vector = input(prompt)
    catch
       disp("Error al ingresar vector, intentelo nuevamente")
    end
end
sumarFunciones(vector)
```

6.1.7 sumarFunciones.m

```
function [resultado] = sumarFunciones(vector)
if length(vector) < 4
```

```
    disp("El vector ingresado tiene menos de 4 elementos")
    return
end
mayores = sort(vector, 'descend');
mayores = mayores(1:4)
menores = sort(vector, 'ascend');
menores = menores(1:4)
sumaMayores = sum(mayores);
sumaMenores = sum(menores);
resultado = sqrt(sumaMayores) - sqrt(sumaMenores);
```

6.2 MANUAL DE USUARIO

6.2.1 Inicio

Requerimientos:

- Tener instalado MATLAB, versión mínima R2018a
- Descargar y descomprimir archivo .zip con el código fuente.

Una vez descomprimido el archivo con código fuente, se generará una carpeta de nombre 'src' con los siguientes archivos: Por lo tanto, se debe abrir los archivos main.m, main2Parte1.m y

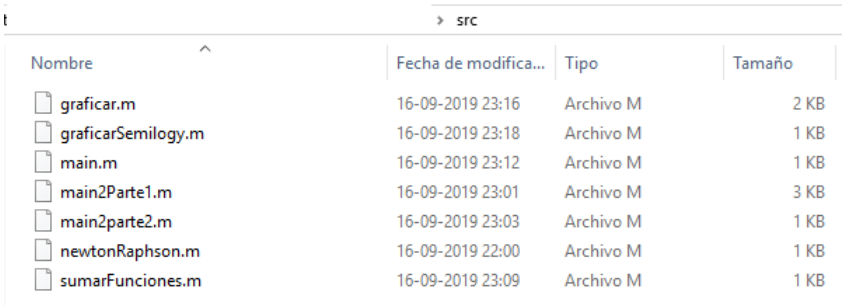

Figura 6.1: Carpeta src

main2Parte2.m con MATLAB y agregar el 'path' carpeta 'src' en caso de ser necesario.

6.2.2 Parte 1

Para ejecutar la parte 1, simplemente se debe ejecutar el archivo main.m con el IDE de MATLAB. Automáticamente se generarán los 5 gráficos correspondientes a la parte 1 del

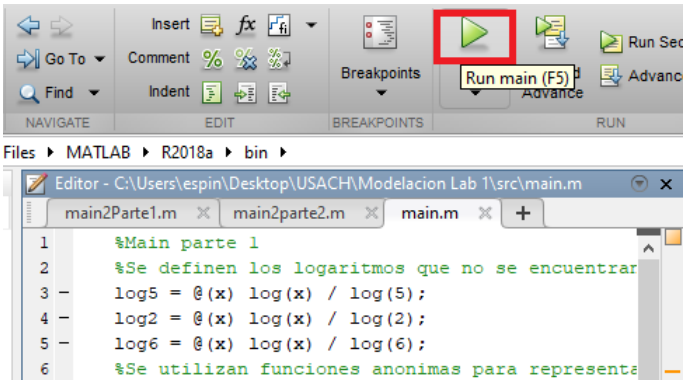

Figura 6.2: Ejecutar primera parte del laboratorio

laboratorio.

6.2.3 Parte 2: Newton Raphson

De forma similar a la parte 1, se debe abrir el archivo main2Parte1.m en MATLAB y ejecutarlo. Se pedirá el ingreso de parámetros al usuario, los cuales son los siguientes y se piden en el siguiente orden:

1. Polinomio: Se recibe en forma de vector, sólo se aceptan valores numéricos.
2. x0: Se recibe en forma de escalar, sólo puede ser un valor numérico.
3. Iteraciones: Se recibe en forma de escalar, sólo puede ser un entero mayor a 0 y valor numérico.
4. Tolerancia: Se recibe en forma de escalar, debe ser un número mayor o igual a 0.

En caso de fallar al ingresar algún valor, se pedirá nuevamente hasta que se ingrese correctamente.

6.2.3.1 Casos de prueba

1. Caso 1: Se utilizaron los siguientes valores.

   - Polinomio: [1 0 1 1]
   - x0 = 0
   - Iteraciones = 10
   - Tolerancia = 10^{-18}

   Se obtuvo el resultado: **-0.6823**

```
>> main2Parte1
Ingrese el polinomio en su representación array:
[1 0 1 1]
polinomio =
     1     0     1     1
Ingrese el valor inicial x0
0
x0 =
     0
Ingrese el numero maximo de iteraciones
10
iter =
    10
Ingrese la tolerancia
10^-18
```

```
err =

   1.0000e-18

   -0.6823

ans =

   -0.6823

>>
```

2. Caso 2:

   - Polinomio: [2 3]
   - x0 = 0
   - Iteraciones = 10
   - Tolerancia = 10^{-18}

   Se obtuvo el resultado: **-1.5**

```
>> main2Parte1

Ingrese el polinomio en su representación array:

[2 3]

polinomio =

     2     3

Ingrese el valor inicial x0

0

x0 =

     0

Ingrese el numero maximo de iteraciones

10

iter =

    10

Ingrese la tolerancia
```

```
10^-18

err =

   1.0000e-18

   -1.5000

ans =

   -1.5000

>>
```

3. Caso 3:

   - Polinomio: [5 0 2 1]
   - x0 = 0
   - Iteraciones = 10
   - Tolerancia = 10^{-18}

   Se obtuvo el resultado: **-0.3717**

```
>> main2Parte1

Ingrese el polinomio en su representación array:

[5 0 2 1]

polinomio =

     5     0     2     1

Ingrese el valor inicial x0

0

x0 =

     0

Ingrese el numero maximo de iteraciones

10

iter =

    10

Ingrese la tolerancia
```

```
10^-18

err =

   1.0000e-18

   -0.3717

ans =

   -0.3717

>>
```

6.2.4 Parte 2: Suma de Vectores

Se debe abrir el archivo main2Parte2.m en MATLAB y ejecutarlo desde el IDE. De la misma forma que para Newton Raphson, se pide el ingreso de un vector al usuario. Se verifica que el vector debe ser de largo mayor a 4 elementos.

6.2.4.1 Casos de Prueba

1. Caso 1: Se ingresa el vector [1 1 1 1] Resultado: **0**

```
>> main2parte2

Ingrese un vector con un número de elementos mayor a 4:

[1 1 1 1]

vector =

     1     1     1     1

mayores =

     1     1     1     1

menores =

     1     1     1     1

ans =
```

```
     0

>>
```

2. Caso 2: Se ingresa el vector [-1 -2 -3 -4 5 6 7 8] Resultado: **5.0990 - 3.1623i**

```
>> main2parte2

Ingrese un vector con un número de elementos mayor a 4:

[-1 -2 -3 -4 5 6 7 8]

vector =

    -1    -2    -3    -4     5     6     7     8

mayores =

     8     7     6     5

menores =

    -4    -3    -2    -1

ans =

   5.0990 - 3.1623i

>>
```

3. Caso 3: En este caso se prueba con un vector de menor largo [1 1 1], pero inmediatamente se pide un vector válido. Se ingresa el vector [7 8 5 3 5 6] y se obtiene el resultado **0.7401**

```
>> main2parte2

Ingrese un vector con un número de elementos mayor a 4:

[1 1 1]

vector =

     1     1     1

Ingrese un vector con un número de elementos mayor a 4:

[7 8 5 3 5 6]

vector =

     7     8     5     3     5     6
```

```
mayores =

     8     7     6     5

menores =

     3     5     5     6

ans =

    0.7401

>>
```

BIBLIOGRAFÍA

de Cantabria, I. G. (2019). Escalas logarítmicas.

Larrosa, I. (n.d.). Método de newton-raphson. Recuperado desde https://www.geogebra.org/m/XCrwWHzy ".

MathWorks (2019a). Recuperado desde https://la.mathworks.com/products/matlab.html?s_tid=hp_ff_p_matlab".

MathWorks (2019b). Funciones anónimas. Recuperado desde https://la.mathworks.com/help/matlab/matlab_prog/anonymous-functions.html".