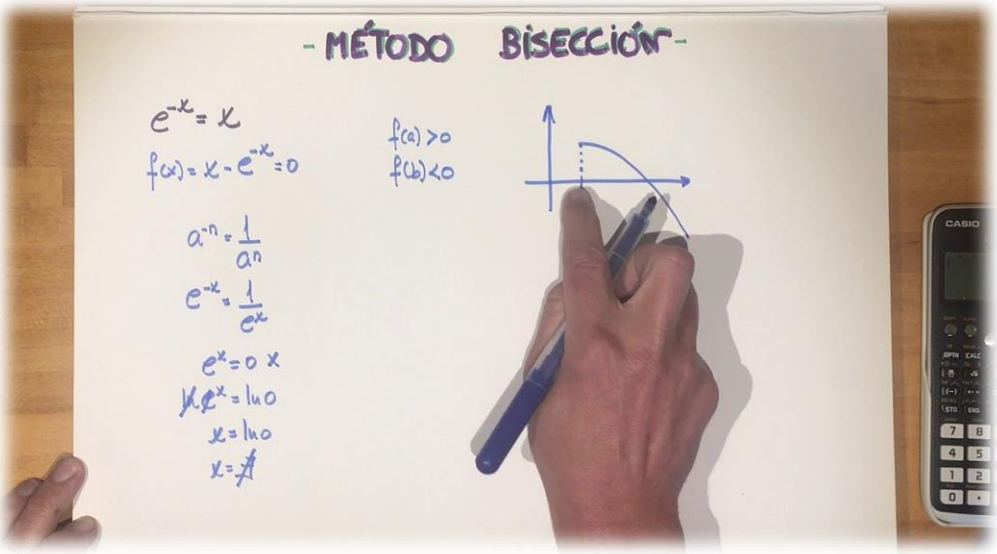
Universidad Autónoma de Nuevo León

Facultad de Ciencias Físico Matemáticas

Análisis Numérico

Evidencia 2:

"Método de Bisección"



Alumno: Jesús Armando Espino Rodríguez

Matricula: 1844607

Profesora: María del Carmen Martínez Cejudo

Grupo: 031

Horario: 09:00 am a 10:00 am

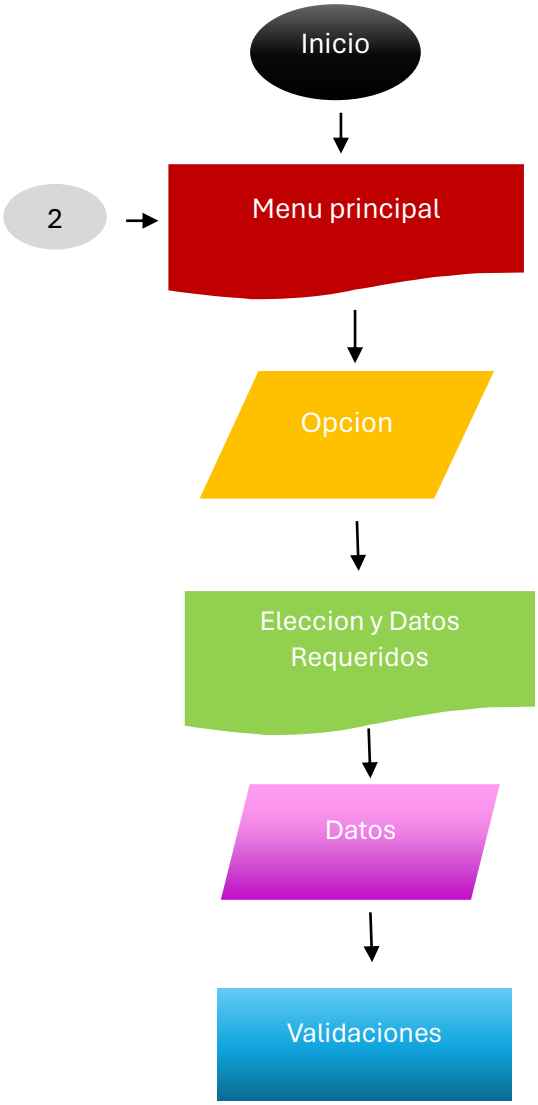
Método de Bisección.

- El método de bisección es un algoritmo utilizado para encontrar las raíces de una función continua en un intervalo dado. Es uno de los métodos más simples y básicos para la búsqueda de raíces en análisis numérico. El principio fundamental detrás del método de bisección es el teorema del valor intermedio, que establece que, si una función continua cambia de signo en un intervalo, entonces debe haber al menos una raíz en ese intervalo.

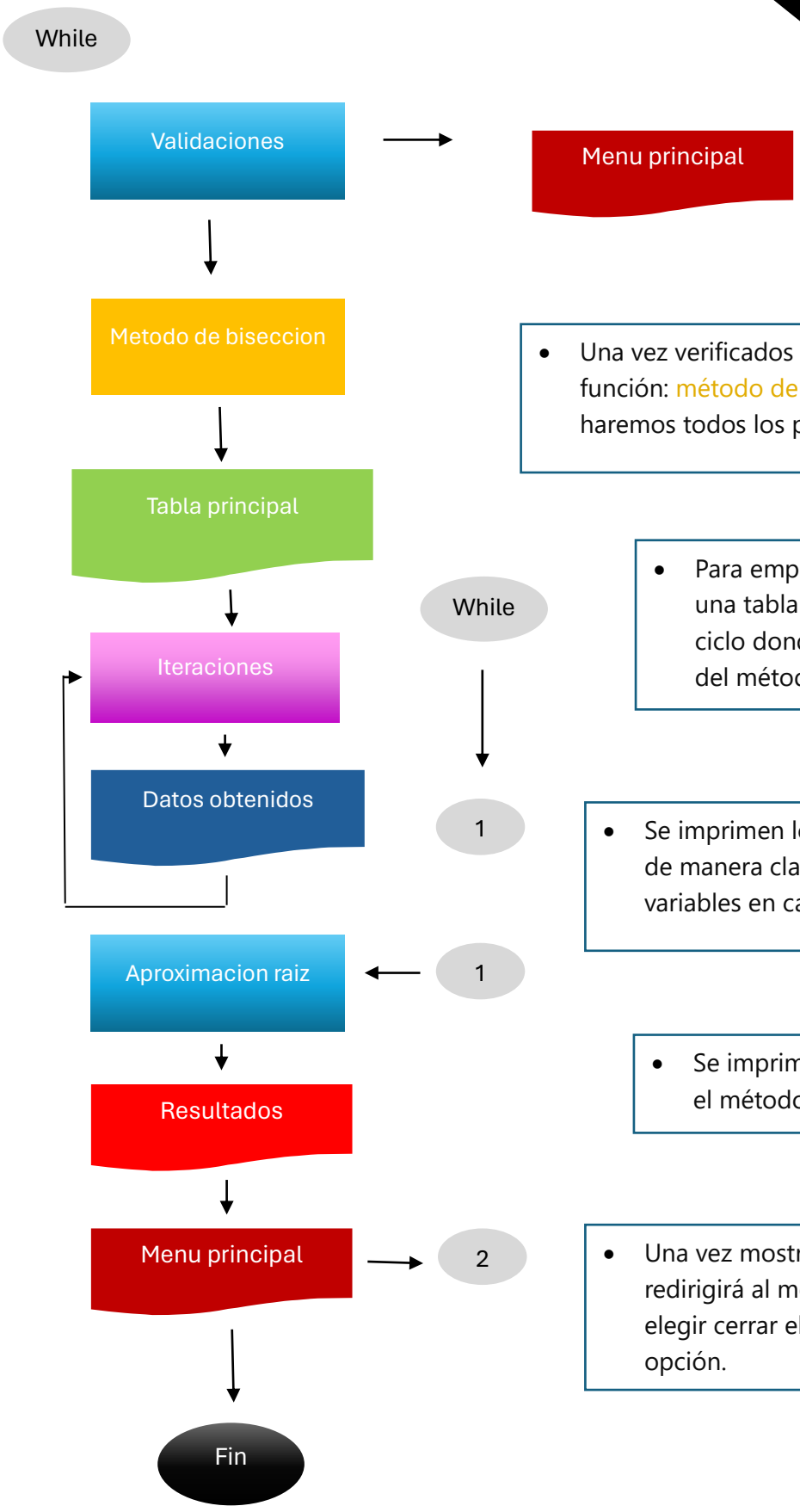
- El proceso de bisección implica dividir repetidamente el intervalo en dos partes iguales y seleccionar la mitad en la cual la función cambia de signo. Esto reduce sucesivamente el tamaño del intervalo que contiene la raíz, hasta que el intervalo se vuelve lo suficientemente pequeño como para satisfacer algún criterio de convergencia predefinido.

- Aunque el método de bisección es simple y garantiza la convergencia hacia la raíz, puede ser relativamente lento en comparación con otros métodos más avanzados, especialmente en funciones donde la raíz se encuentra muy lejos del punto inicial. Sin embargo, su simplicidad lo hace útil como punto de partida para otros métodos más eficientes o como un método de referencia para validar resultados más complicados.

Diagrama de Flujo



- El diagrama comenzaría con un nodo de inicio, indicando el inicio del programa.
- Se mostrará el menú principal del programa, el cual contará con 3 opciones actualmente.
- Se debe ingresar una opción de las mostradas en el menú principal.
- Se imprimirá la elección tomada, para posteriormente pedirnos los datos requeridos por el método.
- Se ingresan los datos requeridos, en este caso se pedirá una función, intervalos y una tolerancia deseada.
- Se realizarán validaciones sobre los datos ingresados en caso de no ser correctos se mostrará el menú principal del programa.



- Una vez verificados los datos entramos en la función: método de bisección en el cual haremos todos los procesos.

- Para empezar, imprimimos la fila principal de una tabla para posteriormente entrar en un ciclo donde se llevará a cabo las iteraciones del método.

- Se imprimen los datos, en donde veremos de manera clara los resultados de las variables en cada iteración

- Se imprimirán los resultados obtenidos por el método.

- Una vez mostrados los resultados se redirigirá al menú principal, donde podrá elegir cerrar el programa o probar otra opción.

Codificación del programa

Menú de bienvenida

```
def menu():
    while True:
        print("\nSeleccione una de las siguientes opciones:")
        print("1. Eliminación gaussiana")
        print("2. Método de Bisección")
        print("3. ...")
        print("4. ...")
        print("5. Cerrar el programa")

        opcion = input("Ingrese el número de la opción deseada: ")

        if opcion == "1":
            resolver_eliminacion_gaussiana()
        elif opcion == "2":
            resolver_metodo_biseccion()
        elif opcion == "5":
            print("Cerrando el programa...")
            break
        else:
            print("Opción inválida. Por favor, seleccione una opción válida.")
```

- Para empezar con este programa, tenemos un menú principal, en el cual, actualmente tenemos 3 opciones:
  - Eliminación gaussiana.
  - Método bisección.
  - Cerrando el programa
- Comenzamos seleccionando una opción en la terminal, para después entrar en un sub-menú donde se pedirán los datos necesarios para resolver el problema planteado
- En caso de ingresar una opción que este fuera de las antes mencionadas, se mostrara un mensaje de error, para después mostrar nuevamente el menú.

Función: resolver_metodo_biseccion

```python
def resolver_metodo_biseccion():
    print("Has seleccionado Método de Bisección.")

    # Solicitar al usuario que ingrese la función
    funcion_str = input("Ingrese la función para la cual desea encontrar la raíz (ej. 'x**2 - 4'): ")
    try:
        funcion = eval("lambda x: " + funcion_str)  # Convertir la cadena en una función
    except Exception as e:
        print("Error al definir la función:", e)
        return

    # Pedir al usuario los extremos del intervalo y la tolerancia
    while True:
        try:
            a = float(input("Ingrese el extremo izquierdo del intervalo: "))
            b = float(input("Ingrese el extremo derecho del intervalo: "))
            if a >= b:
                raise ValueError("El extremo izquierdo debe ser menor que el extremo derecho.")
            tol = float(input("Ingrese la tolerancia deseada: "))
            if tol <= 0:
                raise ValueError("La tolerancia debe ser un número positivo.")
            break
        except ValueError as e:
            print("Error:", e)

    # Verificar si f(a) y f(b) tienen signos opuestos en los extremos del intervalo
    if funcion(a) * funcion(b) >= 0:
        print("La función no cumple con el requisito de tener signos opuestos en los extremos del intervalo.")
        print(f"Valor de la función en el extremo izquierdo ({a}): {funcion(a)}")
        print(f"Valor de la función en el extremo derecho ({b}): {funcion(b)}")
        return
```

- Entrando a esta función, encontraremos un mensaje confirmando la elección que hemos tomado. Después tendremos un mensaje que nos pedirá ingresar una función por pantalla, verificando que esta cumpla con los criterios de sintaxis.

- Entramos en un bucle, en el cual se pedirá ingresar el intervalo extremo izquierdo, después el intervalo extremo derecho y por ultimo la tolerancia deseada, dentro de este bucle se hacen validaciones para verificar que los intervalos sean correctos y la tolerancia sea positiva.

- Al final tenemos una última verificación la cual determina si la función evaluada en dichos intervalos tiene signos opuestos, de ser así, continua con el procedimiento, de lo contrario se imprime un mensaje de error y nos devuelve al menú principal.

Función: metodo_biseccion

```python
def metodo_biseccion(f, a, b, tol=1e-6, max_iter=100):
    # Verificar si f(a) y f(b) tienen signos opuestos
    if f(a) * f(b) >= 0:
        print("La función no cumple con el requisito de tener signos opuestos en los extremos del intervalo.")
        return None

    # Inicializar el contador de iteraciones
    iter_count = 0

    # Imprimir los detalles iniciales
    print(f"Iteración\tExtremo izquierdo\tExtremo derecho\tRaíz aproximada\t\tTolerancia")
    print("-------------------------------------------------------------------------------------------")

    # Bucle principal del método de bisección
    while (b - a) / 2 > tol and iter_count < max_iter:
        c = (a + b) / 2  # Calcular el punto medio del intervalo
        if f(c) == 0:
            break  # La raíz fue encontrada exactamente
        elif f(c) * f(a) < 0:
            b = c  # La raíz está en el intervalo [a, c]
        else:
            a = c  # La raíz está en el intervalo [c, b]

        # Imprimir los detalles de la iteración actual
        print(f"{iter_count+1}\t\t{a:.6f}\t\t{b:.6f}\t\t{c:.6f}\t\t{(b-a)/2:.10f}")
        iter_count += 1

    # Calcular la aproximación de la raíz
    aproximacion_raiz = (a + b) / 2

    return aproximacion_raiz
```

- Aquí tenemos la función método_biseccion la cual es la encargada de llevar a cabo dicho método, en ella veremos que recibirá como parámetros:
  - f: Una función
  - a: Intervalo extremo izquierdo
  - b: Intervalo extremo derecho.
  - tol=1e-6: tolerancia deseada
  - max_iter=100: máximo de iteraciones en dicho proceso

- Tenemos nuevamente una verificación para asegurar que la función evaluada en dichos intervalos tenga signos opuestos. Después dejamos lista una impresión para elaborar una tabla con las iteraciones que saldrán del bucle principal.

- En el bucle se hará el proceso del método de bisección el cual tendrá la función de calcular el punto medio, después tendremos una sentencia de control que verificara si la función evaluada en una variable c es igual a cero, entonces la raíz se habrá encontrado, de lo contrario evaluaremos en que intervalo se encuentra la raíz para posteriormente imprimir los resultados obtenidos. Esto se repetirá hasta encontrar una raíz satisfactoria.

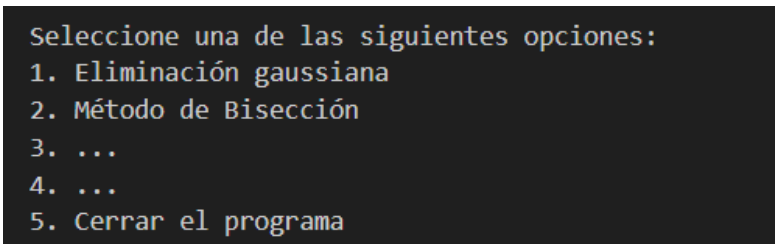
- Una ves terminadas las iteraciones del bucle, procedemos a calcular la aproximación de la raíz, la cual será retornada por la función.

```
# Resolver utilizando el método de bisección
raiz = metodo_biseccion(funcion, a, b, tol)

# Mostrar la aproximación de la raíz
if raiz is not None:
    print("\nAproximación de la raíz:", raiz)
else:
    print("\nNo se pudo encontrar una raíz dentro del intervalo especificado.")
```

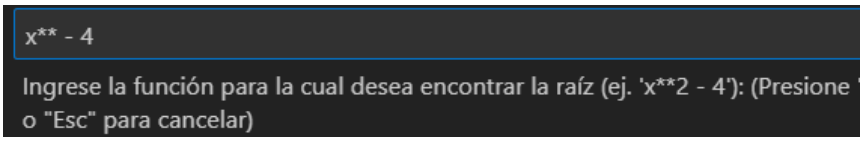
- La aproximación de la raíz retornada ira a la variable raíz

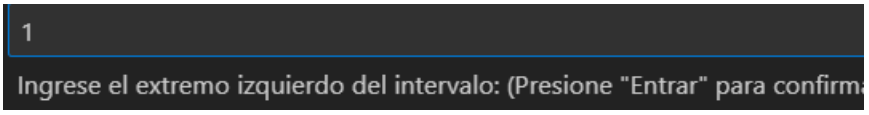
- Por último, tendremos una sentencia de control que nos imprimirá el resultado de la variable raíz, en caso de obtener algún otro resultado se imprimirá un mensaje de error.

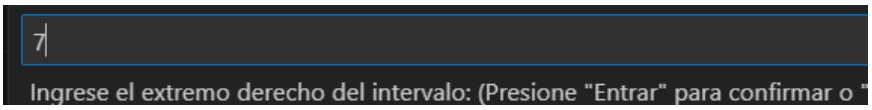


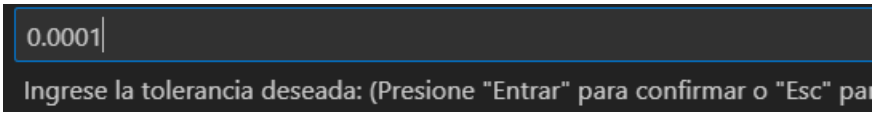
- Aquí vemos el menú principal del programa, donde tendremos que seleccionar una opción.

- Se imprime la opción seleccionada









Aquí vemos como se lleva a cabo el proceso de ingresar los datos requeridos por el método.

Primero vemos la función ingresada, después el intervalo izquierdo y el intervalo derecho.

Por último, la tolerancia deseada

```
Iteración   Extremo izquierdo   Extremo derecho Raíz aproximada     Tolerancia
-------------------------------------------------------------------------------
1       1.000000        4.000000        4.000000        1.5000000000
2       1.000000        2.500000        2.500000        0.7500000000
3       1.750000        2.500000        1.750000        0.3750000000
4       1.750000        2.125000        2.125000        0.1875000000
5       1.937500        2.125000        1.937500        0.0937500000
6       1.937500        2.031250        2.031250        0.0468750000
7       1.984375        2.031250        1.984375        0.0234375000
8       1.984375        2.007812        2.007812        0.0117187500
9       1.996094        2.007812        1.996094        0.0058593750
10      1.996094        2.001953        2.001953        0.0029296875
11      1.999023        2.001953        1.999023        0.0014648438
12      1.999023        2.000488        2.000488        0.0007324219
```

- Aquí vemos los resultados obtenidos de cada iteración del método de bisección, empezando por el numero de iteración, el valor del extremo izquierdo, el valor del extremo derecho, la raíz hasta ahora aproximada y la tolerancia
- Una ves se haya encontrado la raíz, o se cumpla la tolerancia deseada el ciclo se detiene, dándonos los resultados finales

```
Aproximación de la raíz: 1.999755859375
```

- Aquí vemos el resultado final.

```
Seleccione una de las siguientes opciones:
1. Eliminación gaussiana
2. Método de Bisección
3. ...
4. ...
5. Cerrar el programa
Cerrando el programa...
```

- Al final vemos impresión del menú, al cual se le ingreso la opción 5, la cual finaliza el programa

Conclusiones

- Durante la implementación del programa se vieron posibles alternativas para llevar a cabo este trabajo, se reitero la idea anterior relacionada al código de eliminación gaussiana para elaborar un código mas extenso que pudiera abarcar el método de bisección, así como algunos otros métodos futuros.
- Fue entretenido elaborar este documento, se hizo con la intención de ser llamativo y de fácil comprensión, siendo un guía en cada paso de la elaboración e introducción del método.
- Espero que sea de su agrado y éxito a todos.

Referencias Bibliográficas

- Kolman, Bernard, y Hill, David R. (2019). "Elementary Linear Algebra with Applications". Pearson.
- Python Documentation: https://docs.python.org/3/library/array.html
- "Análisis Numérico" de Richard L. Burden y J. Douglas Faires.