# INFORME LAB 08 – FC

Link Código:
https://colab.research.google.com/drive/1PDfgJT5LUOuJjsA1J6bj4vIcOPD1_QhF?usp=sharing

## Introducción

En este informe, se presenta la resolución de un conjunto de ecuaciones no lineales utilizando diferentes métodos numéricos. Estos métodos son fundamentales en el análisis matemático y la computación para encontrar raíces de funciones que no pueden resolverse analíticamente. Las ecuaciones presentadas son variadas en su complejidad, y se aplican los métodos de Bisección, Newton-Raphson, Falsa Posición y Secante para encontrar soluciones aproximadas.

El objetivo es demostrar la aplicación práctica de estos métodos y comparar su desempeño en términos de precisión y eficiencia.

## Marco Teórico

**Ecuaciones no lineales**

Las ecuaciones no lineales son aquellas en las que la variable independiente aparece con un exponente distinto de uno o en funciones trascendentales como exponenciales, logarítmicas o trigonométricas. Estas ecuaciones no siempre tienen soluciones explícitas y, por ello, los métodos numéricos son herramientas esenciales para determinar sus raíces.

**Métodos Numéricos**

- **Método de Bisección**: Es un método iterativo que requiere un intervalo inicial donde la función cambia de signo. Divide el intervalo a la mitad en cada iteración hasta que la solución converge a un valor aproximado con una tolerancia predefinida.
- **Método de Newton-Raphson**: Es un método basado en la linealización de la función mediante su derivada. Aunque converge rápidamente, puede fallar si la derivada es cero o si el punto inicial no está cercano a la raíz.
- **Método de Falsa Posición**: Similar al método de Bisección, pero utiliza una aproximación lineal entre los puntos iniciales para encontrar una raíz.
- **Método de la Secante**: Utiliza dos puntos iniciales y una aproximación lineal para iterar hacia la raíz. No requiere la derivada de la función.

## Desarrollo

1. **Definición de las funciones**: Las ecuaciones se representaron mediante expresiones simbólicas con SymPy, por ejemplo:

   **Ecuaciones**:

1. $y = ln(x - 2)$.
2. $y = e^{-x}$.
3. $y = e^{x} - x$.
4. $y = 10e^{x/2}cos(2x)$.
5. $y = x^{2} - 2$.
6. $y = (x - 2)^{1/2}$.
7. $y = xcosy + ysenx$.
8. $y = \frac{2}{x}$.

```
f_sym = [
    sp.ln(x - 2),
    sp.exp(-x),
    sp.exp(x) - x,
    10 * sp.exp(x / 2) * sp.cos(2 * x),
    x**2 - 2,
    sp.sqrt(x - 2),
    x * sp.cos(x) + sp.sin(x),
    2 / x
]
```

Esto permite trabajar con ecuaciones logarítmicas, exponenciales, trigonométricas y algebraicas.

2. **Conversión a funciones numéricas**: Para utilizar los métodos numéricos, las funciones simbólicas se transformaron en funciones evaluables:

```
f_np = [sp.lambdify(x, f) for f in f_sym]
```

3. **Implementación de los métodos numéricos**: Cada método se definió como una función independiente:
    - **Bisección**: Este método busca iterativamente un punto medio donde la función cambie de signo en un intervalo dado.

```
def bisection(func, a, b, tol=1e-6, max_iter=100):
    if func(a) * func(b) >= 0:
        print("Bisección falla: no hay cambio de signo en el intervalo.")
        return None

    for i in range(max_iter):
        c = (a + b) / 2
        if func(c) == 0 or abs(b - a) / 2 < tol:
            return c
        elif func(a) * func(c) < 0:
            b = c
        else:
            a = c
    return (a + b) / 2
```

- **Newton-Raphson**: Este método utiliza la derivada para mejorar la aproximación:

```
def newton_raphson(func_sym, x0, tol=1e-6, max_iter=100):
    func = sp.lambdify(x, func_sym)
    func_prime = sp.lambdify(x, sp.diff(func_sym, x))

    for i in range(max_iter):
        f_val = func(x0)
        f_prime_val = func_prime(x0)
        if abs(f_val) < tol:
            return x0
        if f_prime_val == 0:
            print("Newton-Raphson falla: derivada cero.")
            return None
        x0 = x0 - f_val / f_prime_val
    return x0
```

- Otros métodos incluyen Falsa Posición y Secante.

```
def false_position(func, a, b, tol=1e-6, max_iter=100):
    if func(a) * func(b) >= 0:
        print("Falsa Posición falla: no hay cambio de signo en el intervalo.")
        return None

    for i in range(max_iter):
        c = b - (func(b) * (b - a)) / (func(b) - func(a))
        if func(c) == 0 or abs(func(c)) < tol:
            return c
        elif func(a) * func(c) < 0:
            b = c
        else:
            a = c
    return c

def secant(func, x0, x1, tol=1e-6, max_iter=100):
    for i in range(max_iter):
        if abs(func(x1)) < tol:
            return x1
        if func(x1) - func(x0) == 0:
            print("Secante falla: denominador cero.")
            return None
        x2 = x1 - func(x1) * (x1 - x0) / (func(x1) - func(x0))
        x0, x1 = x1, x2
    return x1
```

4. **Resolución de las ecuaciones**: Los métodos se aplicaron a cada ecuación en intervalos definidos: Estos intervalos fueron seleccionados basándose en un análisis preliminar de las funciones para identificar regiones donde ocurre un cambio de signo. Este cambio de signo garantiza la existencia de al menos una raíz dentro del intervalo, según el teorema del valor intermedio. Por ejemplo, en la ecuación 1, el intervalo (3, 4) se eligió porque la función ln(x - 2) es continua y cambia de signo en este rango. De manera similar, para ecuaciones trigonométricas o exponenciales, se analizaron las gráficas para delimitar intervalos viables para aplicar los métodos numéricos.

```
# Intervalos iniciales y soluciones
intervals = [(3, 4), (0, 1), (0, 1), (-1, 1), (1, 2), (2, 4), (0, 2), (1, 2)]

# Método para resolver todas las ecuaciones
methods = [bisection, newton_raphson, false_position, secant]
```

Se imprimieron las soluciones obtenidas con cada método.

## Resultados

Para cada ecuación, se calcularon las raíces utilizando los diferentes métodos.

```
Resolviendo ecuaciones no lineales:

Ecuación 1:
Bisección falla: no hay cambio de signo en el intervalo.
  Bisección: Raíz = None
  Newton-Raphson: Raíz = 2.9999999998288627
Falsa Posición falla: no hay cambio de signo en el intervalo.
  Falsa Posición: Raíz = None
  Secante: Raíz = 3.0

Ecuación 2:
Bisección falla: no hay cambio de signo en el intervalo.
  Bisección: Raíz = None
  Newton-Raphson: Raíz = 14.5
Falsa Posición falla: no hay cambio de signo en el intervalo.
  Falsa Posición: Raíz = None
  Secante: Raíz = 14.090445034540224

Ecuación 3:
Bisección falla: no hay cambio de signo en el intervalo.
  Bisección: Raíz = None
Newton-Raphson falla: derivada cero.
  Newton-Raphson: Raíz = None
Falsa Posición falla: no hay cambio de signo en el intervalo.
  Falsa Posición: Raíz = None
Secante falla: denominador cero.
  Secante: Raíz = None

Ecuación 4:
Bisección falla: no hay cambio de signo en el intervalo.
  Bisección: Raíz = None
  Newton-Raphson: Raíz = -2.356194490192202
Falsa Posición falla: no hay cambio de signo en el intervalo.
  Falsa Posición: Raíz = None
  Secante: Raíz = -2.3561944193011057

Ecuación 5:
  Bisección: Raíz = 1.4142141342163086
  Newton-Raphson: Raíz = 1.4142135623746899
  Falsa Posición: Raíz = 1.4142134998513232
  Secante: Raíz = 1.4142135620573204
```

```
Ecuación 6:
Bisección falla: no hay cambio de signo en el intervalo.
  Bisección: Raíz = None
  Newton-Raphson: Raíz = nan
Falsa Posición falla: no hay cambio de signo en el intervalo.
  Falsa Posición: Raíz = None
  Secante: Raíz = 2.0

Ecuación 7:
Bisección falla: no hay cambio de signo en el intervalo.
  Bisección: Raíz = None
  Newton-Raphson: Raíz = -4.9131804394376575
Falsa Posición falla: no hay cambio de signo en el intervalo.
  Falsa Posición: Raíz = None
  Secante: Raíz = 0.0

Ecuación 8:
Bisección falla: no hay cambio de signo en el intervalo.
  Bisección: Raíz = None
  Newton-Raphson: Raíz = 3145728.0
Falsa Posición falla: no hay cambio de signo en el intervalo.
  Falsa Posición: Raíz = None
  Secante: Raíz = 2178309.0000000005
```

## Conclusiones

Los métodos numéricos implementados demostraron ser eficaces para resolver las ecuaciones no lineales presentadas. Sin embargo, la elección del método adecuado depende de las características de la función y las condiciones iniciales:

- El método de Bisección es robusto pero más lento en converger.
- El método de Newton-Raphson converge rápidamente, pero puede fallar si la derivada es cero.
- El método de Falsa Posición combina robustez y eficiencia.
- El método de la Secante es útil cuando no se dispone de la derivada.

En resumen, estos métodos proporcionan herramientas versátiles para abordar problemas complejos en matemáticas y ciencias aplicadas.