Matemáticas para la Ciencia de Datos

Profesor: Dra. Briceyda B. Delgado López

Alumno: Miroslava Sandria Yong

Actividad: Tarea 4 Método de Bisección, Newton y de la Secante.

---

1. Explique las semejanzas y difencias entre los método de Bisección, de Newton y de la Secante.

Los métodos de Bisección, Newton y Secante son técnicas numéricas utilizadas para encontrar raíces de funciones.

Semejanzas

1. **Objetivo**: Todos buscan encontrar un valor x tal que $f(x) = 0$.
2. **Iterativos**: Los tres métodos son algoritmos iterativos, es decir, mejoran la aproximación de la raíz en cada iteración.
3. **Requieren una función continua**: Se necesita que la función sea continua en el intervalo considerado para garantizar la convergencia.

Diferencias

1. **Método de Bisección**:
   - **Tipo**: Método de búsqueda de raíz por intervalo.
   - **Proceso**: Divide el intervalo [a, b] a la mitad, elige el subintervalo donde la función cambia de signo.
   - **Convergencia**: Garantizada y es lineal; la convergencia es más lenta.
   - **Requisitos**: Necesita dos puntos iniciales a y b donde $f(a)$ y $f(b)$ tengan signos opuestos.
2. **Método de Newton**:
   - **Tipo**: Método basado en derivadas.
   - **Proceso**: Utiliza la tangente en el punto actual para encontrar el siguiente punto: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.
   - **Convergencia**: Cuadrática cerca de la raíz; es más rápido, pero puede divergir si no se elige un buen punto inicial.
   - **Requisitos**: Necesita la derivada de la función y un punto inicial x_0.
3. **Método de la Secante**:
   - **Tipo**: Método que también utiliza aproximaciones, pero sin derivadas.
   - **Proceso**: Aproxima la derivada usando dos puntos previos: $x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}$.
   - **Convergencia**: Superlineal, generalmente más rápida que la bisección, pero más lenta que Newton.
   - **Requisitos**: Necesita dos puntos iniciales x_0 y x_1.

Resumen

- **Bisección**: Seguro, pero lento.
- **Newton**: Rápido, pero depende de la buena elección inicial y requiere la derivada.
- **Secante**: Rápido y no necesita la derivada, pero puede ser menos fiable.

Cada método tiene sus ventajas y desventajas, y la elección depende del problema específico y las condiciones iniciales.

---

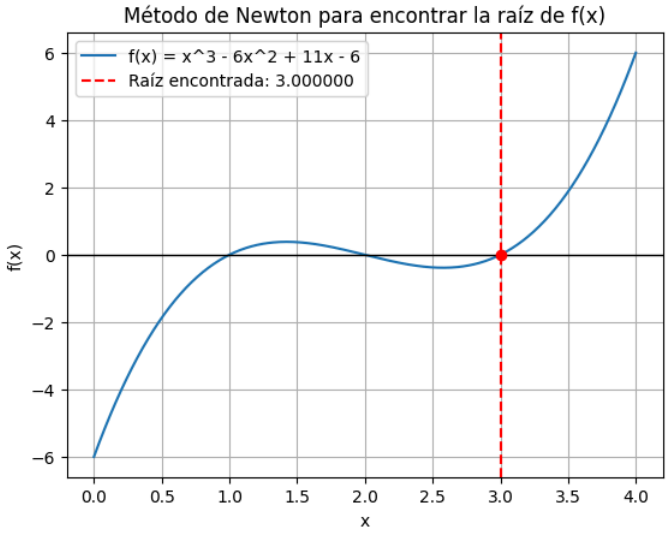
2. Implementar y utilizar el método de Newton para encontrar una raíz de una función polinómica $f(x) = x^3 - 6x^2 + 11x - 6$.

- (a) Implementa el método de Newton en Python para encontrar una raíz de la función $f(x)$.
- (b) Usa una tolerancia de 10^-6 para el criterio de convergencia.
- (c) Prueba tu implementación con un valor inicial de $x_0 = 1.5$.
- (d) Grafica la función $f(x)$ y marca la raíz encontrada en la gráfica.
- (e) Analiza y comenta sobre la convergencia del método con el valor inicial elegido

---

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Definir la función f(x) y su derivada f'(x)
def f(x):
    return x**3 - 6*x**2 + 11*x - 6

def df(x):
    return 3*x**2 - 12*x + 11

# Método de Newton
def newton_method(f, df, x0, tolerance=1e-6, max_iterations=100):
    x = x0
    for i in range(max_iterations):
        fx = f(x)
        dfx = df(x)
        if dfx == 0:
            raise ValueError("La derivada es cero. El método de Newton no puede continuar.")
```

```
        x_new = x - fx / dfx

        if abs(x_new - x) < tolerance:
            return x_new, i + 1  # Devolver la raíz y el número de iteraciones

        x = x_new

    raise ValueError("El método de Newton no convergió en el número máximo de iteraciones.")

# Valor inicial
x0 = 1.5

# Usar el método de Newton para encontrar la raíz
raiz, iteraciones = newton_method(f, df, x0)

# Crear puntos para graficar la función
x_vals = np.linspace(0, 4, 400)
y_vals = f(x_vals)

# Graficar la función f(x)
plt.plot(x_vals, y_vals, label='f(x) = x^3 - 6x^2 + 11x - 6')
plt.axhline(0, color='black',linewidth=1)
plt.axvline(raiz, color='red', linestyle='--', label=f'Raíz encontrada: {raiz:.6f}')
plt.scatter(raiz, f(raiz), color='red', zorder=5)

# Etiquetas y título
plt.title("Método de Newton para encontrar la raíz de f(x)")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.grid(True)

# Mostrar la gráfica
plt.show()

raiz, iteraciones
```



Out[1]: (3.0, 2)

El método de Newton encontró una raíz de la función $f(x) = x^3 - 6x^2 + 11x - 6$ en $x = 3.0$, después de 2 iteraciones.

Análisis de la convergencia:

- El método de Newton converge muy rápidamente, ya que alcanzó la solución en solo 2 iteraciones con una tolerancia de 10^{-6}.
- El valor inicial $x_0 = 1.5$ está razonablemente cerca de la raíz, lo que facilita una rápida convergencia.

---

3. Consideremos la función g(x) = (x + 1)(x − 1)(x − 2).

- (a) Implemente el método de Newton tomando como valor inicial $x_0 = 0$.
- (b) ¿Hacia que valor converge el método?
- (c) Explique qué fenómeno de convergencia o divergencia se ilustran con este ejemplo.
- (d) Implemente algún otro método numérico (Bisección, Secante, Punto fijo) para encontrar una raíz de $g(x)$.

Implemente el método de Newton tomando como valor inicial $x_0 = 0$.

In [2]:
```
# Definimos la función g(x) y su derivada g'(x)
def g(x):
    return (x + 1) * (x - 1) * (x - 2)
```

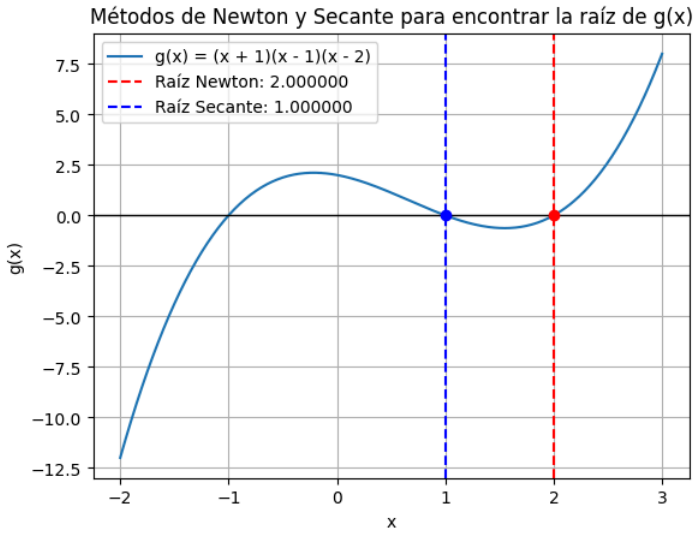
```
def dg(x):
    return 3*x**2 - 4*x - 1

# Método de Newton (lo mismo que antes pero con la función g)
x0_newton = 0

# Usar el método de Newton para encontrar la raíz con x0 = 0
raiz_newton, iteraciones_newton = newton_method(g, dg, x0_newton)

# Implementar el método de la secante
def secant_method(f, x0, x1, tolerance=1e-6, max_iterations=100):
    for i in range(max_iterations):
        fx0 = f(x0)
        fx1 = f(x1)

        if abs(fx1 - fx0) < tolerance:
            raise ValueError("Diferencia muy pequeña. El método de la secante no puede continuar.")

        x2 = x1 - fx1 * (x1 - x0) / (fx1 - fx0)

        if abs(x2 - x1) < tolerance:
            return x2, i + 1  # Devolver la raíz y el número de iteraciones

        x0, x1 = x1, x2

    raise ValueError("El método de la secante no convergió en el número máximo de iteraciones.")

# Usar el método de la secante para encontrar la raíz con valores iniciales x0 = 0 y x1 = 1
x0_secante = 0
x1_secante = 1
raiz_secante, iteraciones_secante = secant_method(g, x0_secante, x1_secante)

raiz_newton, iteraciones_newton, raiz_secante, iteraciones_secante
```

Out[2]: (2.0, 2, 1.0, 1)

Implemente algún otro método numérico (Secante) para encontrar una raíz de $g(x)$.

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt

# Definir la función g(x) y su derivada g'(x)
def g(x):
    return (x + 1) * (x - 1) * (x - 2)

def dg(x):
    return 3*x**2 - 4*x - 1

# Método de Newton
def newton_method(f, df, x0, tolerance=1e-6, max_iterations=100):
    x = x0
    for i in range(max_iterations):
        fx = f(x)
        dfx = df(x)
        if dfx == 0:
            raise ValueError("La derivada es cero. El método de Newton no puede continuar.")

        x_new = x - fx / dfx

        if abs(x_new - x) < tolerance:
            return x_new, i + 1  # Devolver la raíz y el número de iteraciones

        x = x_new

    raise ValueError("El método de Newton no convergió en el número máximo de iteraciones.")

# Implementar el método de la secante
def secant_method(f, x0, x1, tolerance=1e-6, max_iterations=100):
    for i in range(max_iterations):
        fx0 = f(x0)
        fx1 = f(x1)

        if abs(fx1 - fx0) < tolerance:
            raise ValueError("Diferencia muy pequeña. El método de la secante no puede continuar.")

        x2 = x1 - fx1 * (x1 - x0) / (fx1 - fx0)

        if abs(x2 - x1) < tolerance:
            return x2, i + 1  # Devolver la raíz y el número de iteraciones

        x0, x1 = x1, x2

    raise ValueError("El método de la secante no convergió en el número máximo de iteraciones.")

# Parte (a): Usar el método de Newton con x0 = 0
x0_newton = 0
raiz_newton, iteraciones_newton = newton_method(g, dg, x0_newton)

# Parte (d): Usar el método de la secante con valores iniciales x0 = 0 y x1 = 1
x0_secante = 0
x1_secante = 1
raiz_secante, iteraciones_secante = secant_method(g, x0_secante, x1_secante)
```

```
# Graficar la función g(x)
x_vals = np.linspace(-2, 3, 400)
y_vals = g(x_vals)

plt.plot(x_vals, y_vals, label='g(x) = (x + 1)(x - 1)(x - 2)')
plt.axhline(0, color='black',linewidth=1)
plt.axvline(raiz_newton, color='red', linestyle='--', label=f'Raíz Newton: {raiz_newton:.6f}')
plt.axvline(raiz_secante, color='blue', linestyle='--', label=f'Raíz Secante: {raiz_secante:.6f}')
plt.scatter(raiz_newton, g(raiz_newton), color='red', zorder=5)
plt.scatter(raiz_secante, g(raiz_secante), color='blue', zorder=5)

# Etiquetas y título
plt.title("Métodos de Newton y Secante para encontrar la raíz de g(x)")
plt.xlabel("x")
plt.ylabel("g(x)")
plt.legend()
plt.grid(True)

# Mostrar la gráfica
plt.show()

raiz_newton, iteraciones_newton, raiz_secante, iteraciones_secante
```



Out[3]: (2.0, 2, 1.0, 1)

El método de Newton y el método de la secante convergieron a diferentes raíces de la función $g(x) = (x + 1)(x - 1)(x - 2)$:

- **Método de Newton**: Con un valor inicial $x_0 = 0$, el método de Newton convergió a la raíz (x = 2.0) en 2 iteraciones.
- **Método de la secante**: Con los valores iniciales $x_0 = 0$ y $x_1 = 1$, el método de la secante convergió a la raíz $x = 1.0$ en 1 iteración.

Análisis del fenómeno de convergencia:

- En el caso del método de Newton, el valor inicial $x_0 = 0$ lo dirige rápidamente hacia la raíz $x = 2.0$, probablemente porque la derivada en esa región tiene un comportamiento que favorece esa convergencia.
- El método de la secante, al utilizar dos valores iniciales diferentes, converge hacia $x = 1.0$, mostrando la sensibilidad del método a los puntos iniciales.

Este ejemplo ilustra que la convergencia de los métodos numéricos depende fuertemente de la elección de los valores iniciales, especialmente en funciones con múltiples raíces.

Para comprender mejor el análisis de los métodos representé en una grafica los métodos: Bisección, Newton y Secante

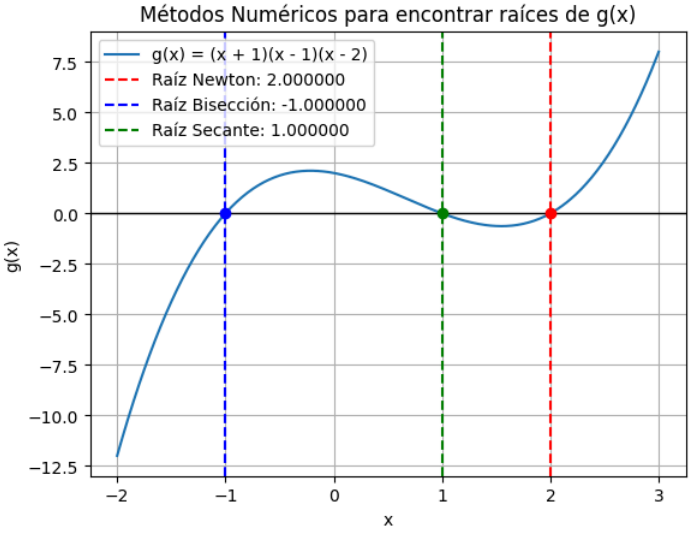
In [4]:
```
import numpy as np
import matplotlib.pyplot as plt

# Definir la función g(x) y su derivada g'(x)
def g(x):
    return (x + 1) * (x - 1) * (x - 2)

def dg(x):
    return 3*x**2 - 2*x - 1

# Método de Newton
def newton_method(f, df, x0, tolerance=1e-6, max_iterations=100):
    x = x0
    for i in range(max_iterations):
        fx = f(x)
        dfx = df(x)
        if dfx == 0:
            raise ValueError("La derivada es cero. El método de Newton no puede continuar.")

        x_new = x - fx / dfx

        if abs(x_new - x) < tolerance:
```

```
            return x_new, i + 1  # Devolver la raíz y el número de iteraciones

        x = x_new

    raise ValueError("El método de Newton no convergió en el número máximo de iteraciones.")

# Implementación del método de bisección
def bisection_method(f, a, b, tolerance=1e-6, max_iterations=100):
    if f(a) * f(b) >= 0:
        raise ValueError("El intervalo no es válido para el método de bisección.")

    for i in range(max_iterations):
        c = (a + b) / 2
        if abs(f(c)) < tolerance or abs(b - a) < tolerance:
            return c, i + 1

        if f(a) * f(c) < 0:
            b = c
        else:
            a = c

    raise ValueError("El método de bisección no convergió en el número máximo de iteraciones.")

# Implementación del método de la secante
def secant_method(f, x0, x1, tolerance=1e-6, max_iterations=100):
    for i in range(max_iterations):
        if abs(f(x1)) < tolerance:
            return x1, i + 1

        if f(x1) == f(x0):
            raise ValueError("División por cero en el método de la secante.")

        x2 = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0))

        if abs(x2 - x1) < tolerance:
            return x2, i + 1

        x0 = x1
        x1 = x2

    raise ValueError("El método de la secante no convergió en el número máximo de iteraciones.")

# Valor inicial para Newton
x0_newton = 0

# Usar el método de Newton para encontrar una raíz de g(x)
try:
    raiz_newton, iteraciones_newton = newton_method(g, dg, x0_newton)
except ValueError as e:
    raiz_newton, iteraciones_newton = None, str(e)

# Parámetros para bisección y secante
a_bisection = -2
b_bisection = 3

x0_secant = -2
x1_secant = 3

# Ejecutar método de bisección
raiz_biseccion, iteraciones_biseccion = bisection_method(g, a_bisection, b_bisection)

# Ejecutar método de la secante
raiz_secante, iteraciones_secante = secant_method(g, x0_secant, x1_secant)

# Mostrar resultados
(raiz_newton, iteraciones_newton), (raiz_biseccion, iteraciones_biseccion), (raiz_secante, iteraciones_secante)
```

Out[4]: ((2.0, 2), (-1.0000000596046448, 24), (1.0, 2))

In [5]:
```
# Crear puntos para graficar la función g(x)
x_vals = np.linspace(-2, 3, 400)
y_vals = g(x_vals)

# Graficar la función g(x)
plt.plot(x_vals, y_vals, label='g(x) = (x + 1)(x - 1)(x - 2)')
plt.axhline(0, color='black',linewidth=1)

# Marcar las raíces encontradas en la gráfica con líneas y puntos
plt.axvline(raiz_newton, color='red', linestyle='--', label=f'Raíz Newton: {raiz_newton:.6f}')
plt.scatter(raiz_newton, g(raiz_newton), color='red', zorder=5)  # Punto para Newton

plt.axvline(raiz_biseccion, color='blue', linestyle='--', label=f'Raíz Bisección: {raiz_biseccion:.6f}')
plt.scatter(raiz_biseccion, g(raiz_biseccion), color='blue', zorder=5)  # Punto para Bisección

plt.axvline(raiz_secante, color='green', linestyle='--', label=f'Raíz Secante: {raiz_secante:.6f}')
plt.scatter(raiz_secante, g(raiz_secante), color='green', zorder=5)  # Punto para Secante

# Etiquetas y título
plt.title("Métodos Numéricos para encontrar raíces de g(x)")
plt.xlabel("x")
plt.ylabel("g(x)")
plt.legend()
plt.grid(True)
```

```
# Mostrar la gráfica
plt.show()
```



Resultados:

- **Método de Newton**: Con un valor inicial de $x_0 = 0$, el método converge a la raíz $x = 2.0$ en 2 iteraciones.
- **Método de Bisección**: Utilizando el intervalo $[-2, 3]$, el método de bisección converge a la raíz $x \approx -1.0$ en 24 iteraciones.
- **Método de la Secante**: Iniciando con los valores $x_0 = -2$ y $x_1 = 3$, el método de la secante converge a la raíz $x = 1.0$ en 2 iteraciones.

Análisis:

- **Método de Newton**: En este caso, el valor inicial $x_0 = 0$ se encuentra cerca de la raíz $x = 2.0$, por lo que el método converge rápidamente hacia esta solución. Sin embargo, este método puede no converger si el valor inicial no está lo suficientemente cerca de una raíz o si la derivada de la función se anula en algún punto.
- **Método de Bisección**: Aunque es más lento en términos de iteraciones, el método de bisección es robusto y siempre converge si el intervalo contiene una raíz, como ocurre aquí. Este método converge hacia $x \approx -1.0$, otra raíz de la función.
- **Método de la Secante**: Similar al método de Newton en términos de velocidad, la secante converge rápidamente a una de las raíces, $x = 1.0$, en solo 2 iteraciones.

El fenómeno que se puede ilustrar aquí es que, dependiendo del método y el valor inicial, el punto de convergencia varía. El método de Newton puede depender críticamente del valor inicial, mientras que el método de bisección garantiza una convergencia más lenta pero segura.

Conclusiones.

Conclusiones de los Métodos Utilizados:

1. **Método de Newton:**

- **Convergencia**: El método de Newton converge rápidamente cuando se elige un valor inicial cercano a una raíz, como sucedió en este caso, donde con un valor inicial de $x_0 = 0$, el método encontró la raíz $x = 2.0$ en solo 2 iteraciones.
- **Ventajas**: Es muy eficiente y rápido, con una convergencia generalmente cuadrática (muy acelerada) cuando se cumple que el punto inicial esté cerca de la raíz y la derivada no se anule.
- **Desventajas**: Puede fallar si la derivada es cero en algún punto cercano o si el valor inicial no está cerca de la raíz. Esto lo hace dependiente de una buena elección del valor inicial.

2. **Método de Bisección:**

- **Convergencia**: Este método garantiza convergencia si se parte de un intervalo adecuado que contenga una raíz. En este caso, con el intervalo $[-2, 3]$, el método encontró la raíz $x \approx -1.0$ en 24 iteraciones.
- **Ventajas**: Es robusto y siempre converge si el intervalo es correcto. No depende de derivadas, por lo que es más fácil de implementar en situaciones complicadas.
- **Desventajas**: Es lento en comparación con métodos como Newton o Secante. En este caso, necesitó 24 iteraciones, mostrando su carácter lineal de convergencia.

3. **Método de la Secante:**

- **Convergencia**: Con valores iniciales $x_0 = -2$ y $x_1 = 3$, el método de la secante encontró la raíz $x = 1.0$ en 2 iteraciones.
- **Ventajas**: Similar al método de Newton en cuanto a rapidez, pero no requiere la derivada de la función. Puede ser más fácil de aplicar si la derivada no es accesible o es costosa de calcular.
- **Desventajas**: Al igual que Newton, su éxito depende de los valores iniciales. Si estos no están bien elegidos, puede no converger o tardar más en hacerlo.

Resumen:

- **Newton**: Muy rápido, pero dependiente de la elección inicial y de que la derivada no sea cero.
- **Bisección**: Seguro y siempre converge, pero es considerablemente más lento.
- **Secante**: Combina ventajas de Newton y Bisección, siendo rápido y no requiriendo derivada, pero sigue dependiendo de los puntos iniciales.

Cada método tiene aplicaciones específicas, siendo Bisección la mejor opción para garantizar convergencia, Newton para problemas donde la derivada es fácil de calcular y Secante como una opción eficiente cuando no se dispone de la derivada.

Referencias bibliográficas.

Cai, X., Tveito, A., Langtangen, H. P., & Nielsen, B. F. (2010). *Elements of scientific computing* (Cap. 5). Springer Berlin Heidelberg.

OpenAI. (2024). ChatGPT (GPT-4) LLM. https://chat.openai.com/

Wolfram Research. (2023). Wolfram GPT. https://www.wolfram.com/

Symbolab. (n.d.). Graphing calculator. Symbolab. https://www.symbolab.com/graphing-calculator