MODELADO Y SIMULACIÓN
Métodos de Aceleración

CLASE 2

Ing. Omar Cáceres

Método de aceleración Aitken

Es una técnica numérica que se usa para mejorar la velocidad de convergencia de una secuencia que converge lentamente hacia un límite. básicamente, toma tres elementos consecutivos de una secuencia y calcula un valor "acelerado" que se acerca más rápidamente al valor límite. la fórmula central del método es:

$$x^*_n = x_{(n)} - \frac{(x_{n+1} - x_n)^2}{x_{n+2} - 2x_{n+1} + x_n}$$

$x_{(n)}$ *termino de sucesión original*

x^*_n *termino de sucesión acelerado*

Se requiere tres términos iterativos consecutivos para poder iniciar la primera extrapolación , y este valor puede ser considerado el inicial de un nuevo proceso de aceleración tras calcular los próximos dos términos al mismo

CLASE 2

Ing. Omar Cáceres

Condiciones de convergencia

·**Secuencia Convergente:** La secuencia debe ser **monótona y convergente** hacia un valor límite L. El método no es efectivo si la secuencia diverge o si oscila de manera significativa.

·**Disponibilidad de Tres :** El denominador de la fórmula de Aitken:

·**Términos Consecutivos:** Para aplicar la fórmula de Aitken, necesitas al menos tres términos consecutivos de la secuencia:

·**Diferencia no Nula en el denominador**

·**Convergencia Relativamente Lenta:** El método es especialmente útil cuando la secuencia converge lentamente al límite, ya que Aitken acelera el proceso. Si la secuencia ya converge rápidamente, la mejora puede ser mínima.

·**Estabilidad Numérica:** Si la secuencia presenta un comportamiento numérico inestable o errores significativos en los cálculos, los resultados de la aceleración pueden no ser fiables.

MODELADO Y SIMULACIÓN
Código Aitken

CLASE 2

Ing. Omar Cáceres

```
def punto_fijo_con_aitken_tabla(g, x0, tol=1e-6, max_iter=100):
    """
    Método del punto fijo con aceleración de Aitken que muestra una tabla de iteraciones.
    g: función de iteración (polinomio en este caso).
    x0: aproximación inicial.
    tol: tolerancia para la convergencia.
    max_iter: número máximo de iteraciones.
    """
    x = x0
    print(f"{'Iteración':<10}{'x':<20}{'x1 = g(x)':<20}{'x2 = g(x1)':<20}{'x_acelerado':<20}{'Error':<20}")
    print("-" * 100)

    for i in range(max_iter):
        # Calcular tres puntos consecutivos de la iteración
        x1 = g(x)
        x2 = g(x1)

        # Aplicar la aceleración de Aitken
        denominador = x2 - 2 * x1 + x
        if denominador != 0:
            x_acelerado = x - (x1 - x)**2 / denominador
        else:
            x_acelerado = x2  # Si no se puede aplicar Aitken, continuar con x

        # Calcular error relativo
        error = abs(x_acelerado - x)

        # Mostrar valores en la tabla
        print(f"{i + 1:<10}{x:<20.10f}{x1:<20.10f}{x2:<20.10f}{x_acelerado:<20.10f}{error:<20.10f}")

        # Verificar convergencia
        if error < tol:
            print(f"\nConvergencia alcanzada en la iteración {i + 1}.")
            return x_acelerado

        # Actualizar para la siguiente iteración
        x = x_acelerado

    print("\nEl método no converge después del número máximo de iteraciones.")
    return None

# Ejemplo de uso
def g(x):
    return (2*x-1)**(1/2)  # Polinomio g(x)

x0 = 2  # Aproximación inicial
resultado = punto_fijo_con_aitken_tabla(g, x0)
if resultado is not None:
    print(f"\nLa solución aproximada es: {resultado}")
```

MODELADO Y SIMULACIÓN

Método Newton Raphson

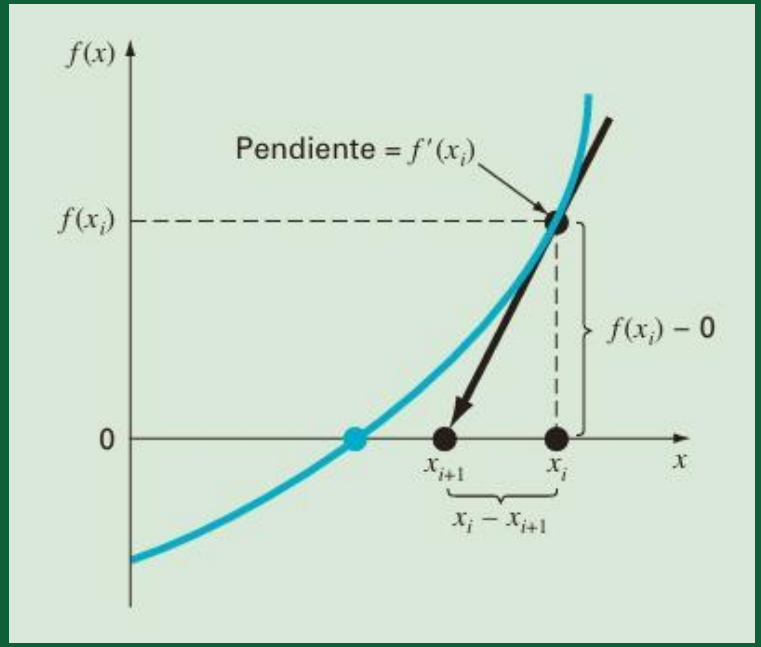
CLASE 2

Ing. Omar Cáceres

Método de Newton Raphson

el método de newton-raphson es un método iterativo muy eficiente para encontrar raíces de funciones no lineales f(x)=0. se basa en la aproximación lineal de la función mediante su derivada, y su fórmula iterativa es:

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



Ventajas:

- Convergencia rápida si el punto inicial está cerca de la raíz.
- Simple de implementar.

Limitaciones:

- Puede fallar o divergir si la derivada es cero o si el punto inicial no está cerca de la raíz.
- Requiere conocer la derivada de la función.

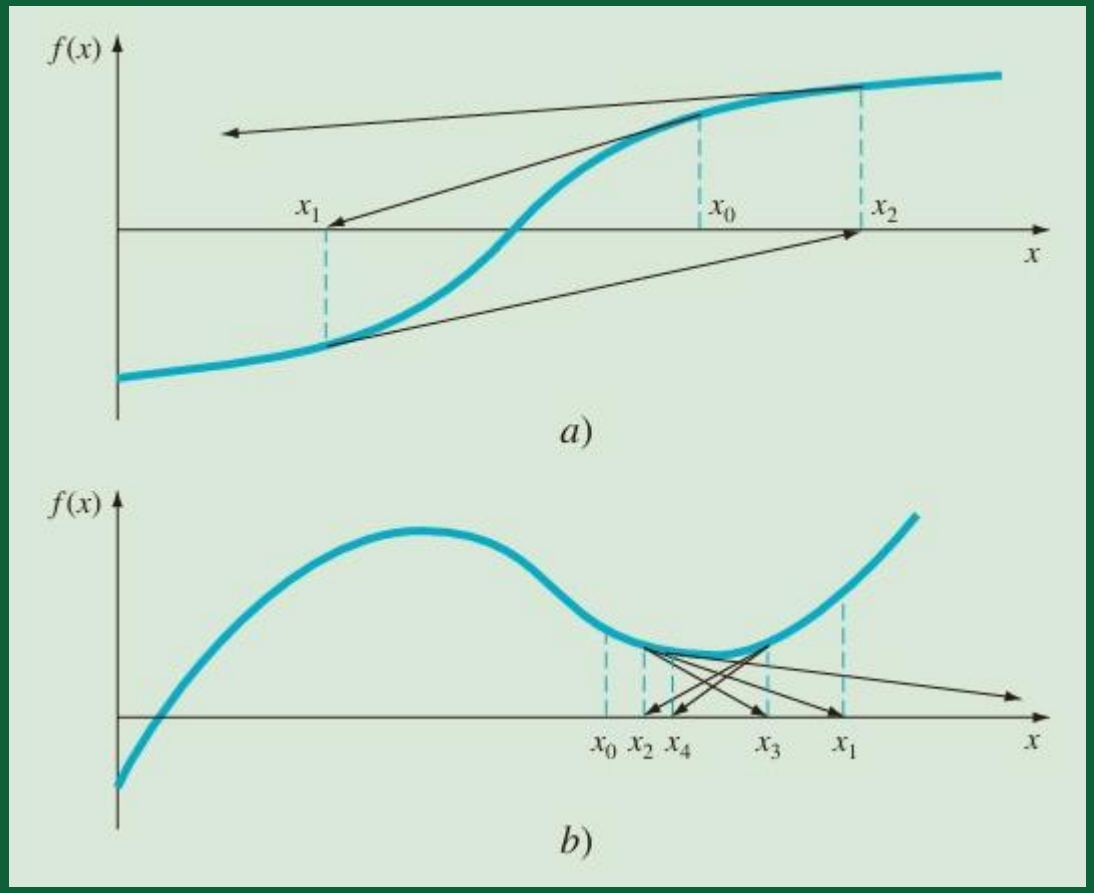
MODELADO Y SIMULACIÓN

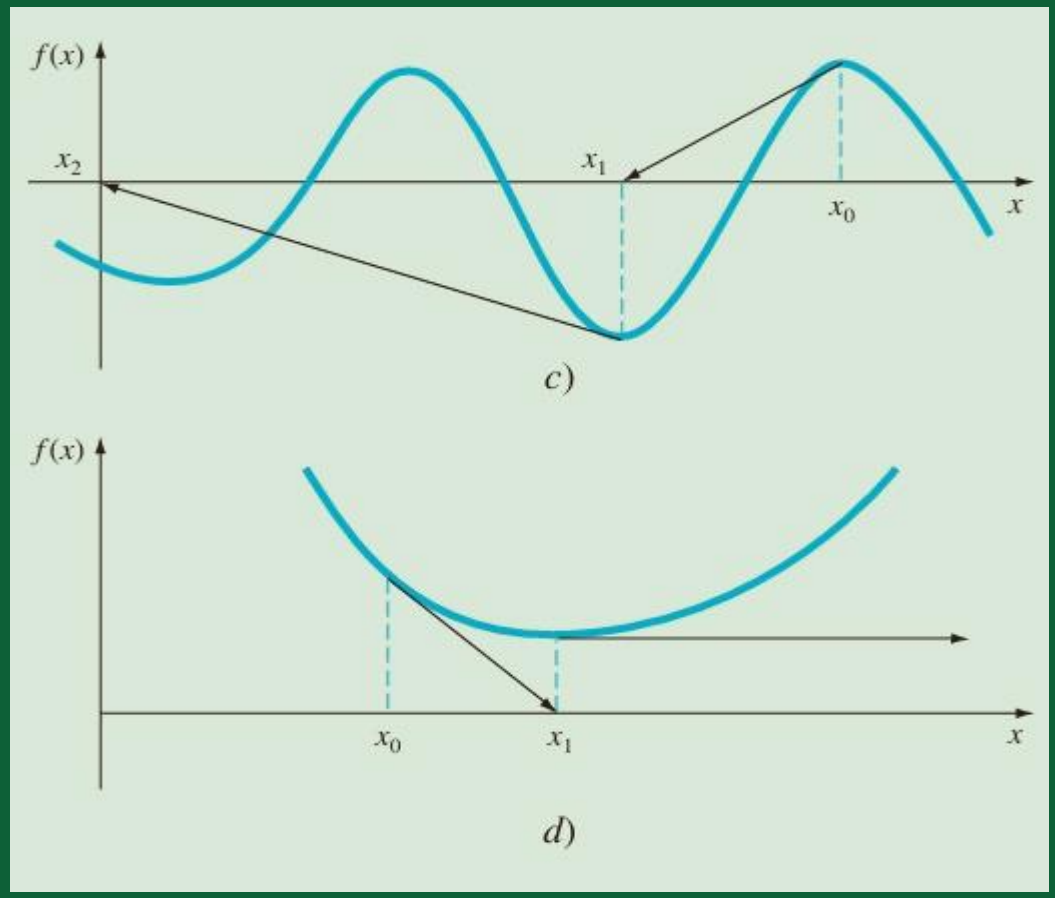
Método de Newton Raphson

CLASE 2

Ing. Omar Cáceres

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





MODELADO Y SIMULACIÓN
Código Newton Raphson

CLASE 2

Ing. Omar Cáceres

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.misc import derivative
from tabulate import tabulate

# Método de Newton-Raphson
def newton_raphson(f, valor_inicial, iteraciones=100, tolerancia=1e-6,
precision=5):
    x = valor_inicial
    results = []
    for i in range(iteraciones):
        fx = round(f(x), precision)
        dfx = round(derivative(f, x, dx=tolerancia), precision)
        if dfx == 0:
            raise ValueError("La derivada es cero. El método no puede
continuar.")
        x_new = round(x - fx / dfx, precision)
        results.append([i+1, x, fx, dfx, x_new])
        print(tabulate(results, headers=["Iteración", "x", "f(x)", "f'(x)",
"Resultado"], tablefmt="grid"))
        if abs(x_new - x) < tolerancia:
            return x_new
        x = x_new
    raise ValueError("El método no convergió o faltan iteraciones.").
```

```
def graficar(f, raiz):
    # Graficar la función
    x = np.linspace(0, 3, 400)
    y = f(x)
    plt.plot(x, y, label='$f(x)$')
    plt.axhline(0, color='black',linewidth=0.5)
    plt.axvline(0, color='black',linewidth=0.5)
    plt.grid(color = 'gray', linestyle = '--', linewidth = 0.5)
     # Marcar la raíz encontrada
    plt.plot(raiz, f(raiz), 'ro', label=f'Raíz: x = {raiz:.5f}')
      # Añadir leyenda y mostrar la gráfica
    plt.legend()
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.title('Gráfica de la función y su raíz')
    plt.show()
# Definir la función para la cual quieres encontrar la raíz
def f(x):
    return x**3 - x-4
# Valor inicial
valor_inicial = 1.0
# Encontrar la raíz utilizando el método de Newton-Raphson
raiz = newton_raphson(f, valor_inicial)
# Imprimir la raíz encontrada
print(f"La raíz encontrada es: {raiz}")
# Graficar la función y la raíz
graficar(f, raiz)
```