Aproximaciones Numéricas: Método de Series de Taylor y Newton-Raphson

Juan Sebastán Peñaloza Quintana

10 Marzo 2025

Abstract

Este documento presenta dos métodos numéricos: la aproximación mediante la serie de Taylor y el método de Newton-Raphson. Se muestra cómo se puede utilizar la serie de Taylor para aproximar funciones, específicamente la función $\sin(x)$, y se detalla el proceso iterativo del método de Newton-Raphson para encontrar raíces de funciones. Se incluyen ejemplos de código en Python para ilustrar la implementación de ambos métodos.

1 Parte 1: Método de Series de Taylor

La serie de Taylor es una representación de una función como una suma infinita de términos que se calculan a partir de los valores de las derivadas de la función en un único punto. La serie de Taylor de una función $f(x)$ alrededor del punto a se expresa como:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots$$

En general, la aproximación de orden n es:

$$f(x) \approx \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x-a)^k,$$

donde:

- $f^{(k)}(a)$ es la k-ésima derivada evaluada en a.
- $k!$ es el factorial de k.

Como ejemplo, la serie de Maclaurin (Taylor alrededor de 0) para la función $\sin(x)$ es:

$$\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

Esta aproximación mejora conforme se agregan más términos.

A continuación se muestra el código en Python que implementa la aproximación de $\sin(x)$ mediante la serie de Taylor y la gráfica de las aproximaciones para distintos órdenes.

Listing 1: $\sin(x)$ Taylor

```python
import numpy as np
import matplotlib.pyplot as plt
import math

def taylor_sin(x, order):
    result = 0.0
    max_m = order // 2 # Considera términos impares hasta x^order
    for m in range(max_m + 1):
        term = ((-1)**m) * (x**(2*m + 1)) / math.factorial(2*m + 1)
        result += term
    return result

# Generar valores de x
x = np.linspace(-np.pi, np.pi, 400)
# Función original
y_sin = np.sin(x)

# Configuración de la gráfica
plt.figure(figsize=(12, 6))
plt.plot(x, y_sin, label=‘‘ $\sin(x)$ ’’ , linewidth=2)

# Aproximaciones de Taylor de diferentes órdenes
orders = [1, 3, 5, 7]
for order in orders:
    y_taylor = np.array([taylor_sin(xi, order) for xi in x])
    plt.plot(x, y_taylor, label=f"$O^{{({order})}}$")

plt.title('Aproximaciones de Taylor para $\sin(x)$')
plt.xlabel('x')
plt.ylabel('$\sin(x)$')
plt.legend(loc='upper left')
plt.grid(True)
plt.show()
```

Listing 1: $\sin(x)$ Taylor

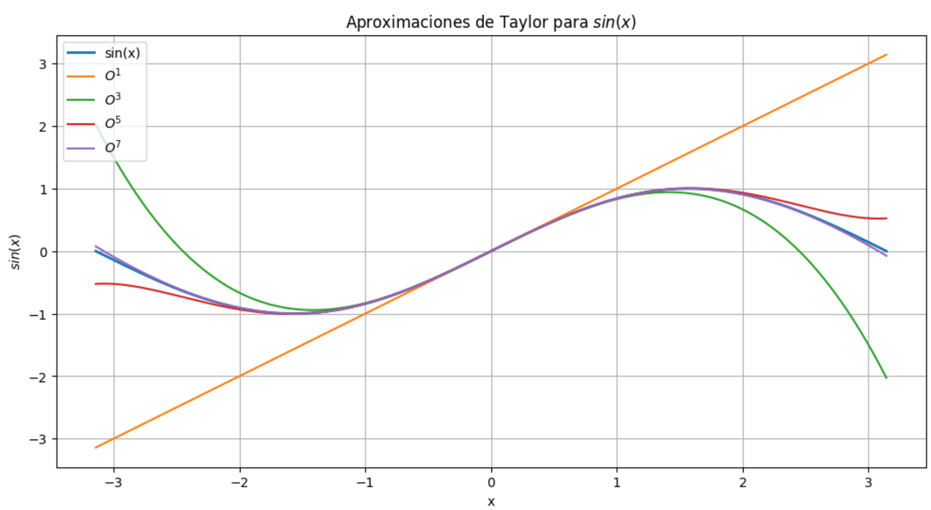

Figure 1: Polinomios para diferentes ordenes de Taylor

Ahora para la funcion exponencial La serie de Taylor para la función exponencial e^x alrededor de $x_0 = 0$ es:

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Listing 2: $\exp(x)$ con Taylor

```python


def taylor_exp(x, order):
    result = 0.0
    for m in range(order + 1):
        term = (x**m) / math.factorial(m)
        result += term
    return result

# Generar valores de x
x = np.linspace(-2, 2, 400)
# Función original
y_exp = np.exp(x)

# Configuración de la gráfica
plt.figure(figsize=(12, 6))
plt.plot(x, y_exp, label='exp(x)', linewidth=2)

# Aproximaciones de Taylor de diferentes órdenes
orders = [1, 3, 5, 7]
for order in orders:
    y_taylor = np.array([taylor_exp(xi, order) for xi in x])
    plt.plot(x, y_taylor, label=f'$O^{({order})}$')

plt.title('Aproximaciones de Taylor para $exp(x)$')
plt.xlabel('x')
plt.ylabel('$exp(x)$')
plt.legend(loc='upper left')
plt.grid(True)
plt.show()
```

Listing 2: $\exp(x)$ con Taylor

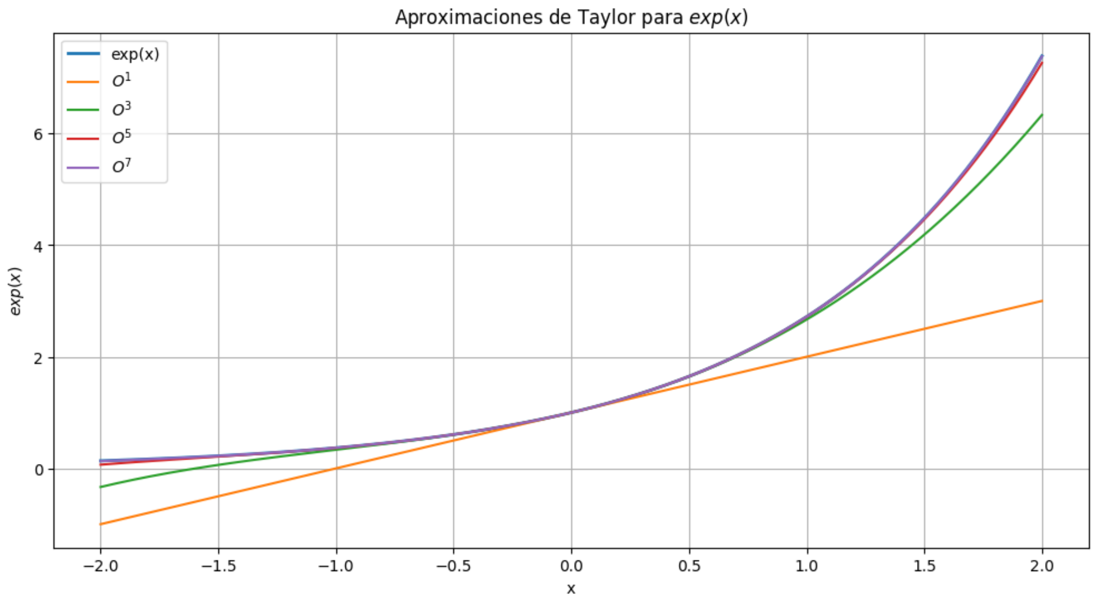

Figure 2: Polinomios para diferentes ordenes de Taylor

2 Parte 2: Método de Newton-Raphson

El método de Newton-Raphson es un procedimiento iterativo para aproximar las raíces de una función $f(x)$. La idea básica es aproximar la función mediante la tangente en un punto cercano a la raíz y usar el punto de intersección de esta tangente con el eje x para obtener una mejor estimación. La fórmula iterativa es:

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

donde:

- x_n es la aproximación actual.
- x_{n+1} es la nueva aproximación.
- $f(x_n)$ es el valor de la función en x_n.
- $f'(x_n)$ es la derivada en x_n.

2.1 Ejemplo: Aproximación de la raíz cuadrada de 612

En este ejemplo se define la función $f(x) = x^2 - 612$ y su derivada, y se utiliza el método de Newton para aproximar $\sqrt{612}$.

Listing 3: Método de Newton para la raíz de $x^2 - 612$

```python
import numpy as np
import math

def f(x):
    return x**2 - 612

def df(x):
    return 2*x

def newton_method(initial_guess, tolerance=1e-7, max_iterations=100):
    x_n = initial_guess
    for n in range(max_iterations):
        f_xn = f(x_n)
        df_xn = df(x_n)
        if df_xn == 0:
            print("Derivada cero. No se puede continuar.")
            return None
        x_n1 = x_n - f_xn / df_xn
        if abs(x_n1 - x_n) < tolerance:
            return x_n1
        x_n = x_n1
    print(No se alcanzó la convergencia.)
    return None

# Valor inicial
initial_guess = 20
approximation = newton_method(initial_guess)
print(f"La aproximación de la raíz cuadrada de 612 es: {approximation}")
```

Listing 3: Método de Newton para la raíz de $x^2 - 612$

2.2 Método de Newton-Raphson General

Se usa a continuación el método de Newton-Raphson para encontrar las raíces de una función $f(x)$, dada su derivada $f'(x)$, un valor inicial y criterios de tolerancia y número máximo de iteraciones. Dada una función $f(x)$ continua y derivable, el método de Newton-Raphson aproxima la raíz de f usando la fórmula:

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

Los pasos son:

1. **Elección de una aproximación inicial**: Seleccionar un valor x_0 cercano a la raíz esperada.
2. **Cálculo de la nueva aproximación**: Aplicar la fórmula iterativamente.
3. **Criterio de convergencia**: Repetir hasta que la diferencia entre dos iteraciones consecutivas sea menor que una tolerancia predefinida.

Listing 4: Método general de Newton-Raphson

```python
import numpy as np

def newton_raphson(f, df, x0, tol=1e-6, max_iter=20):
    """
    Método de Newton-Raphson para encontrar las raíces de la función f(x).
    Parámetros:
        - f: Función lambda en una variable x, f(x).
        - df: Derivada de la función lambda f(x).
        - x0: Valor inicial para comenzar la iteración.
        - tol: Tolerancia para la convergencia.
        - max_iter: Número máximo de iteraciones.
    Retorna:
        - x: Raíz aproximada de la función.
        - iteraciones: Número de iteraciones realizadas.
    """
    x = x0
    for i in range(max_iter):
        fx = f(x)
        dfx = df(x)
        if abs(fx) < tol:
            print(f"Convergencia alcanzada en {i+1} iteraciones.")
            return x, i+1
        if dfx == 0:
            raise ValueError("La derivada se anuló. El método no puede continuar.\n")
        x = x - fx / dfx
    raise ValueError("No se alcanzó la convergencia después del número máximo de iteraciones.\n")

# Ejemplo de uso
f = lambda x: np.cos(x) - x**3
df = lambda x: -np.sin(x) - 3*x**2
x0 = [-1.5, 0, 0.5, 1, 2, 3, 4]

for x in x0:
    try:
        root, iterations = newton_raphson(f, df, x)
        print(f"Raíz encontrada en x = {root} con {iterations} iteraciones, iniciando en x0 = {x}.")
    except ValueError as e:
        print(e)
```

Listing 4: Método general de Newton-Raphson

3 Conclusión

En este documento se han presentado dos técnicas fundamentales en el análisis numérico: la aproximación mediante series de Taylor y el método de Newton-Raphson. Ambas metodologías permiten aproximar funciones y raíces de ecuaciones con distintos niveles de precisión, dependiendo del número de términos o iteraciones utilizadas. Las implementaciones en Python ilustran de forma práctica cómo se pueden aplicar estos métodos en problemas reales.