Taller 03: Series de Taylor y polinomios de Lagrange

Métodos Numéricos

Christopher Criollo

2025-11-11

Tabla de Contenidos

1 EJERCICIOS **1**
- 1.1 **Grafique las curvas de las series de Taylor de varios órdenes para los siguientes casos:** . . . 1
  - 1.1.1 Problema 1 . . . 2
  - 1.1.2 Problema 2 . . . 6
  - 1.1.3 Problema 3 . . . 9
- 1.2 **Encuentre el polinomio de Lagrange para los siguientes datos y grafique:** . . . 12
  - 1.2.1 Problema 1 . . . 12
  - 1.2.2 Problema 2 . . . 14
  - 1.2.3 Problema 3 . . . 16
- 1.3 **CONCLUSIONES** . . . 18
  - 1.3.1 Conclusiones sobre Polinomios de Taylor . . . 18
  - 1.3.2 Conclusiones sobre Interpolación de Lagrange . . . 19

1 EJERCICIOS

1.1 Grafique las curvas de las series de Taylor de varios órdenes para los siguientes casos:

Polinomio de Tylor

$$P_n(x) = f(x_0)+f'(x_0)(x-x_0)+\frac{f''(x_0)}{2!}(x-x_0)^2+\cdots+\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n = \sum_{k=0}^{n}\frac{f^{(k)}(x_0)}{k!}(x-x_0)^k$$

1.1.1 Problema 1

Se busca encontrar las aproximaciones del Polinomio de Taylor con orden 3 para la función:

$$f(x) = \ln(x)$$

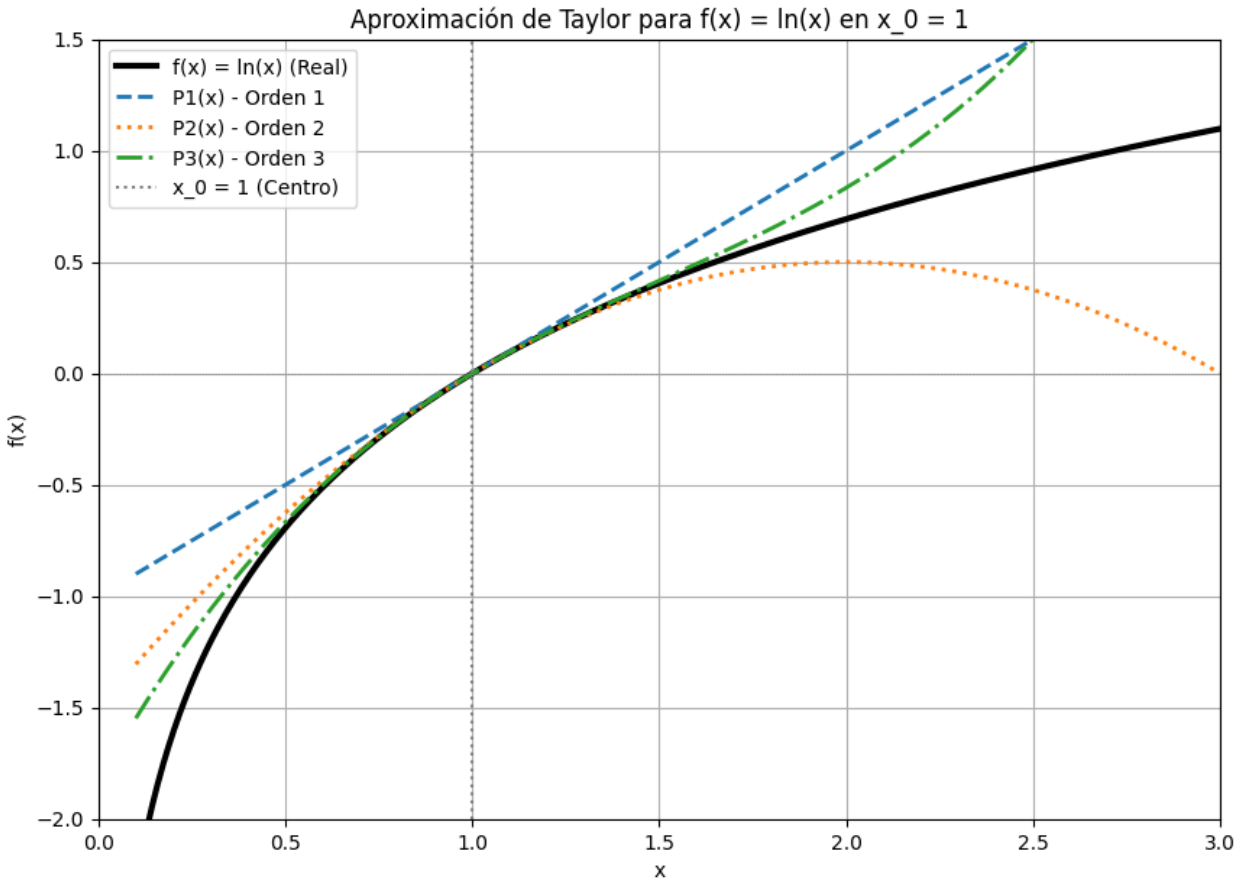
El polinomio estará centrado alrededor del punto:

$$x_0 = 1$$

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

# Funciones
def f_ln(x):
    with np.errstate(divide='ignore'):
        return np.log(x)

def P1_ln(x):
    term_1 = (x - 1)
    return term_1

def P2_ln(x):
    term_1 = (x - 1)
    term_2 = -0.5 * (x - 1)**2
    return term_1 + term_2

def P3_ln(x):
    term_1 = (x - 1)
    term_2 = -0.5 * (x - 1)**2
    term_3 = (1/3) * (x - 1)**3
    return term_1 + term_2 + term_3

x = np.linspace(0.1, 3.0, 400)

# Valores de la función
y_real = f_ln(x)
y_p1 = P1_ln(x)
y_p2 = P2_ln(x)
y_p3 = P3_ln(x)

print("Gráfico Completo")
plt.figure(figsize=(10, 7))
```

```
plt.plot(x, y_real, label='f(x) = ln(x) (Real)', color='black', linewidth=3)
plt.plot(x, y_p1, label='P1(x) - Orden 1', linestyle='--', linewidth=2)
plt.plot(x, y_p2, label='P2(x) - Orden 2', linestyle=':', linewidth=2)
plt.plot(x, y_p3, label='P3(x) - Orden 3', linestyle='-.', linewidth=2)
plt.title('Aproximación de Taylor para f(x) = ln(x) en x_0 = 1')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.axvline(x=1, color='gray', linestyle=':', label='x_0 = 1 (Centro)')
plt.axhline(y=0, color='gray', linestyle=':', linewidth=0.5)
plt.ylim(-2.0, 1.5)
plt.xlim(0, 3)

plt.legend()
plt.grid(True)
plt.show()
```

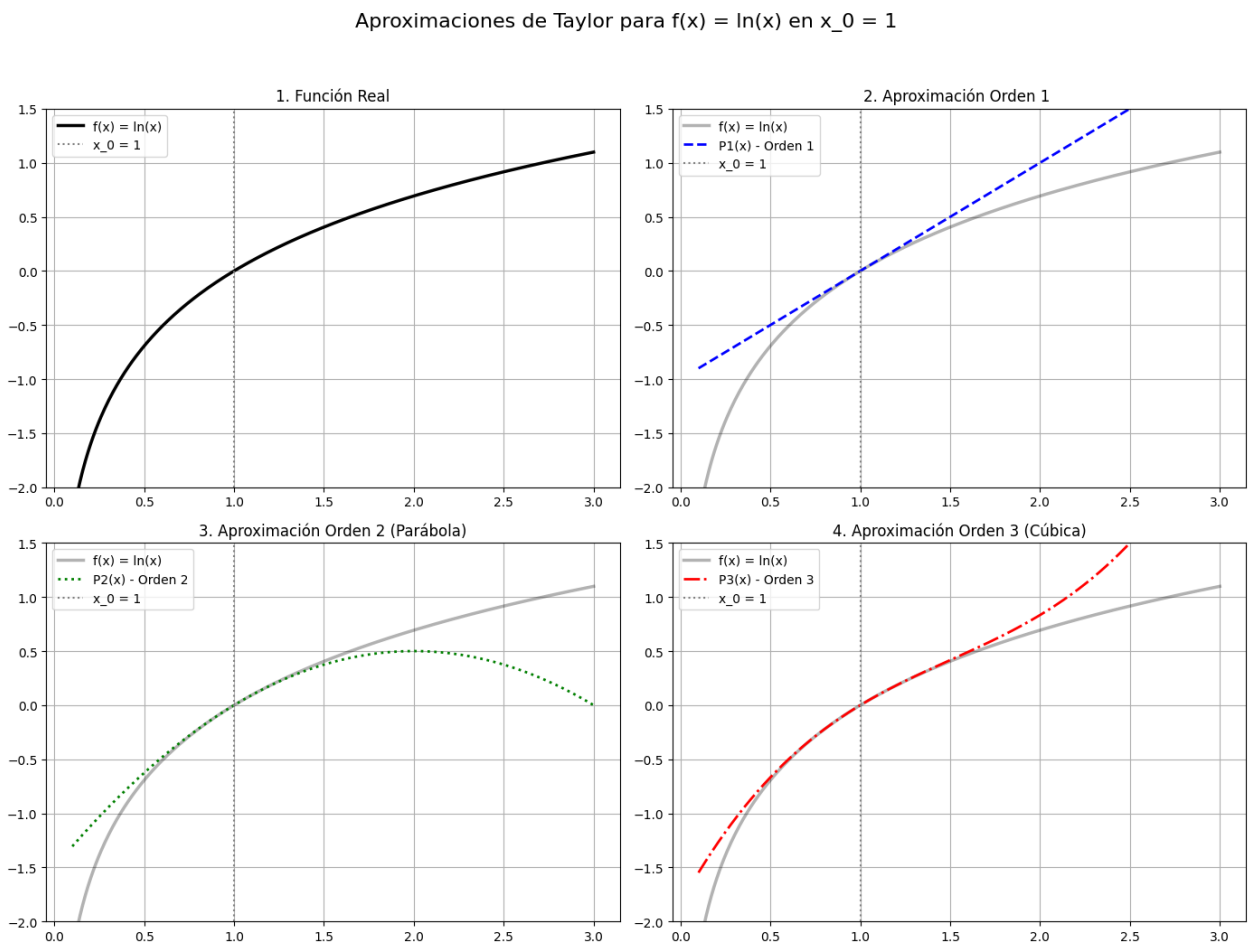
Gráfico Completo



```
print("Gráficas")

fig, axs = plt.subplots(2, 2, figsize=(14, 11))

fig.suptitle('Aproximaciones de Taylor para f(x) = ln(x) en x_0 = 1', fontsize=16)

y_limites = [-2.0, 1.5]

# --- Gráfico 1
axs[0, 0].plot(x, y_real, label='f(x) = ln(x)', color='black', linewidth=2.5)
axs[0, 0].set_title('1. Función Real')
axs[0, 0].axvline(x=1, color='gray', linestyle=':', label='x_0 = 1')
axs[0, 0].set_ylim(y_limites)
axs[0, 0].legend()
axs[0, 0].grid(True)

# --- Gráfico 2
axs[0, 1].plot(x, y_real, label='f(x) = ln(x)', color='black', linewidth=2.5, alpha=0.3)
axs[0, 1].plot(x, y_p1, label='P1(x) - Orden 1', color='blue', linestyle='--', linewidth
axs[0, 1].set_title('2. Aproximación Orden 1 ')
axs[0, 1].axvline(x=1, color='gray', linestyle=':', label='x_0 = 1')
axs[0, 1].set_ylim(y_limites)
axs[0, 1].legend()
axs[0, 1].grid(True)

# --- Gráfico 3
axs[1, 0].plot(x, y_real, label='f(x) = ln(x)', color='black', linewidth=2.5, alpha=0.3)
axs[1, 0].plot(x, y_p2, label='P2(x) - Orden 2', color='green', linestyle=':', linewidth
axs[1, 0].set_title('3. Aproximación Orden 2 (Parábola)')
axs[1, 0].axvline(x=1, color='gray', linestyle=':', label='x_0 = 1')
axs[1, 0].set_ylim(y_limites)
axs[1, 0].legend()
axs[1, 0].grid(True)

# --- Gráfico 4
axs[1, 1].plot(x, y_real, label='f(x) = ln(x)', color='black', linewidth=2.5, alpha=0.3)
axs[1, 1].plot(x, y_p3, label='P3(x) - Orden 3', color='red', linestyle='-.', linewidth=
axs[1, 1].set_title('4. Aproximación Orden 3 (Cúbica)')
axs[1, 1].axvline(x=1, color='gray', linestyle=':', label='x_0 = 1')
axs[1, 1].set_ylim(y_limites)
axs[1, 1].legend()
axs[1, 1].grid(True)

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
```

```
plt.show()
```

Gráficas



El polinomio estará centrado alrededor del punto:

$$x_0 = 0$$

```
import math

# Punto de expansión
x_0 = 0

try:
    # 1. Intentar evaluar f(x_0)
    print(f"Paso 1: Evaluando f(x_0) = ln({x_0})")
    f_x0 = math.log(x_0)
```

```
    # 2. Intentar evaluar f'(x_0)
    print(f"Paso 2: Evaluando f'(x_0) = 1 / {x_0}")
    f_prime_x0 = 1 / x_0

    print("¡Cálculo exitoso!") # Esta línea nunca se alcanzará

except ValueError as e:
    print(f"\nERROR CAPTURADO (ValueError): {e}")
    print("-----------------------------------------------------------------")
    print(f"Explicación: La función f(x) = ln(x) no está definida en x=0.")
    print("El logaritmo de 0 tiende a menos infinito, lo que causa un error matemático."

except ZeroDivisionError as e:
    print(f"\nERROR CAPTURADO (ZeroDivisionError): {e}")
    print("-----------------------------------------------------------------")
    print(f"Explicación: La primera derivada, f'(x) = 1/x, no está definida en x=0.")
    print("Se produce un error de división por cero al intentar calcular 1/0.")
```

```
Paso 1: Evaluando f(x_0) = ln(0)

ERROR CAPTURADO (ValueError): math domain error
-----------------------------------------------------------------
Explicación: La función f(x) = ln(x) no está definida en x=0.
El logaritmo de 0 tiende a menos infinito, lo que causa un error matemático.
```

1.1.2 Problema 2

Se busca encontrar las aproximaciones del Polinomio de Taylor con orden 3 para la función:

$$f(x) = cos(x)$$

El polinomio estará centrado alrededor del punto:

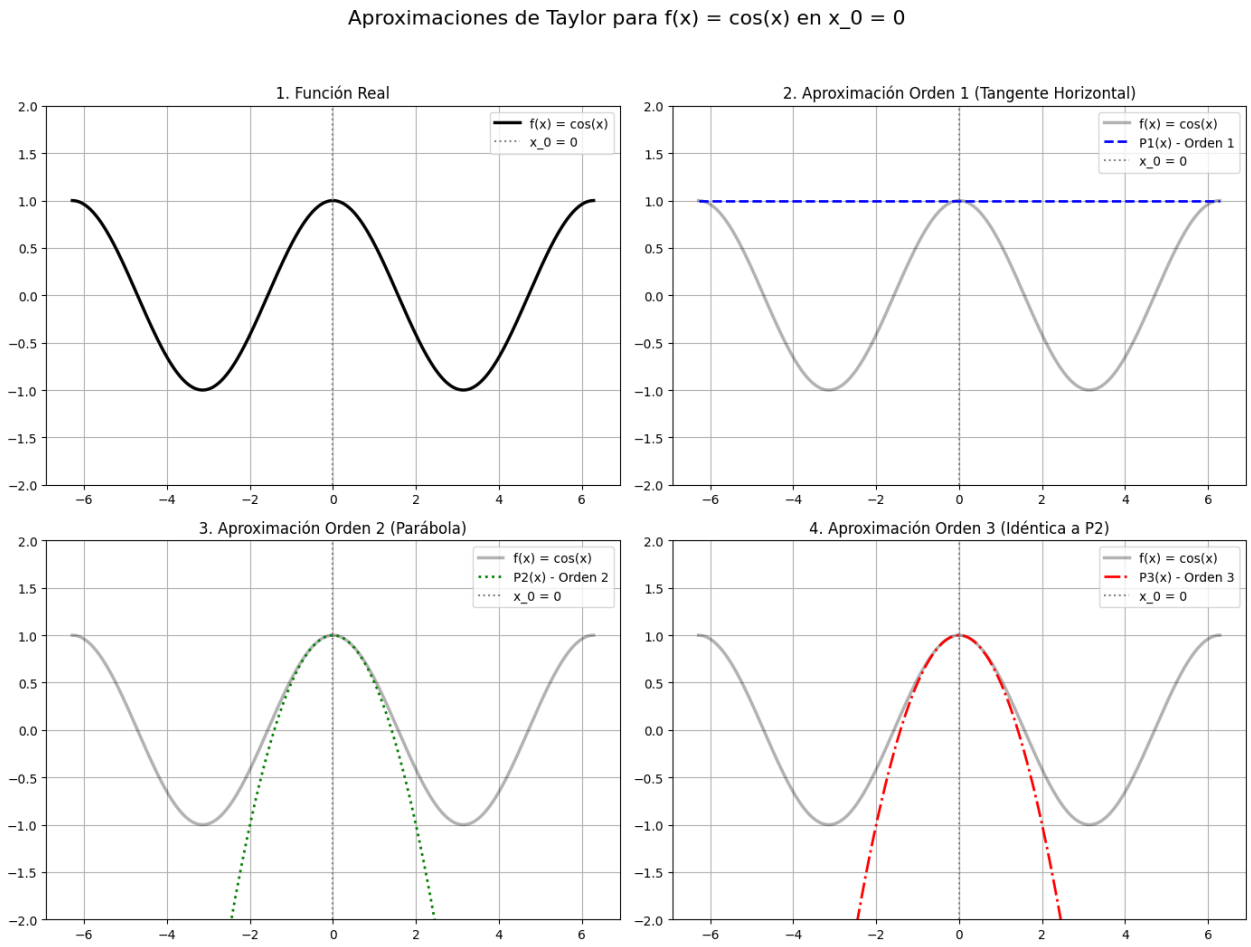
$$x_0 = 0$$

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

def f_cos(x):
    return np.cos(x)
```

```
def P1_cos(x):
    return np.ones_like(x)

def P2_cos(x):
    return 1 - 0.5 * (x**2)

def P3_cos(x):
    return 1 - 0.5 * (x**2)

x = np.linspace(-2 * np.pi, 2 * np.pi, 400)

# Calculamos los valores 'y' para cada función
y_real = f_cos(x)
y_p1 = P1_cos(x)
y_p2 = P2_cos(x)
y_p3 = P3_cos(x)
```

```
fig, axs = plt.subplots(2, 2, figsize=(14, 11))

fig.suptitle('Aproximaciones de Taylor para f(x) = cos(x) en x_0 = 0', fontsize=16)

y_limites = [-2.0, 2.0]
x_0 = 0

# Gráfico 1
axs[0, 0].plot(x, y_real, label='f(x) = cos(x)', color='black', linewidth=2.5)
axs[0, 0].set_title('1. Función Real')
axs[0, 0].axvline(x=x_0, color='gray', linestyle=':', label='x_0 = 0')
axs[0, 0].set_ylim(y_limites)
axs[0, 0].legend()
axs[0, 0].grid(True)

# Gráfico 2
axs[0, 1].plot(x, y_real, label='f(x) = cos(x)', color='black', linewidth=2.5, alpha=0.3
axs[0, 1].plot(x, y_p1, label='P1(x) - Orden 1', color='blue', linestyle='--', linewidth
axs[0, 1].set_title('2. Aproximación Orden 1 (Tangente Horizontal)')
axs[0, 1].axvline(x=x_0, color='gray', linestyle=':', label='x_0 = 0')
axs[0, 1].set_ylim(y_limites)
axs[0, 1].legend()
axs[0, 1].grid(True)

# Gráfico 3
axs[1, 0].plot(x, y_real, label='f(x) = cos(x)', color='black', linewidth=2.5, alpha=0.3
axs[1, 0].plot(x, y_p2, label='P2(x) - Orden 2', color='green', linestyle=':', linewidth
```

```
axs[1, 0].set_title('3. Aproximación Orden 2 (Parábola)')
axs[1, 0].axvline(x=x_0, color='gray', linestyle=':', label='x_0 = 0')
axs[1, 0].set_ylim(y_limites)
axs[1, 0].legend()
axs[1, 0].grid(True)

# Gráfico 4
axs[1, 1].plot(x, y_real, label='f(x) = cos(x)', color='black', linewidth=2.5, alpha=0.3
axs[1, 1].plot(x, y_p3, label='P3(x) - Orden 3', color='red', linestyle='-.', linewidth=
axs[1, 1].set_title('4. Aproximación Orden 3 (Idéntica a P2)')
axs[1, 1].axvline(x=x_0, color='gray', linestyle=':', label='x_0 = 0')
axs[1, 1].set_ylim(y_limites)
axs[1, 1].legend()
axs[1, 1].grid(True)

plt.tight_layout(rect=[0, 0.03, 1, 0.95])

# Mostrar el resultado
plt.show()
```



1.1.3 Problema 3

Se busca encontrar las aproximaciones del Polinomio de Taylor con orden 3 para la función:

$$\frac{1}{1-x}$$

El polinomio estará centrado alrededor del punto:

$$x_0 = \frac{1}{2}$$

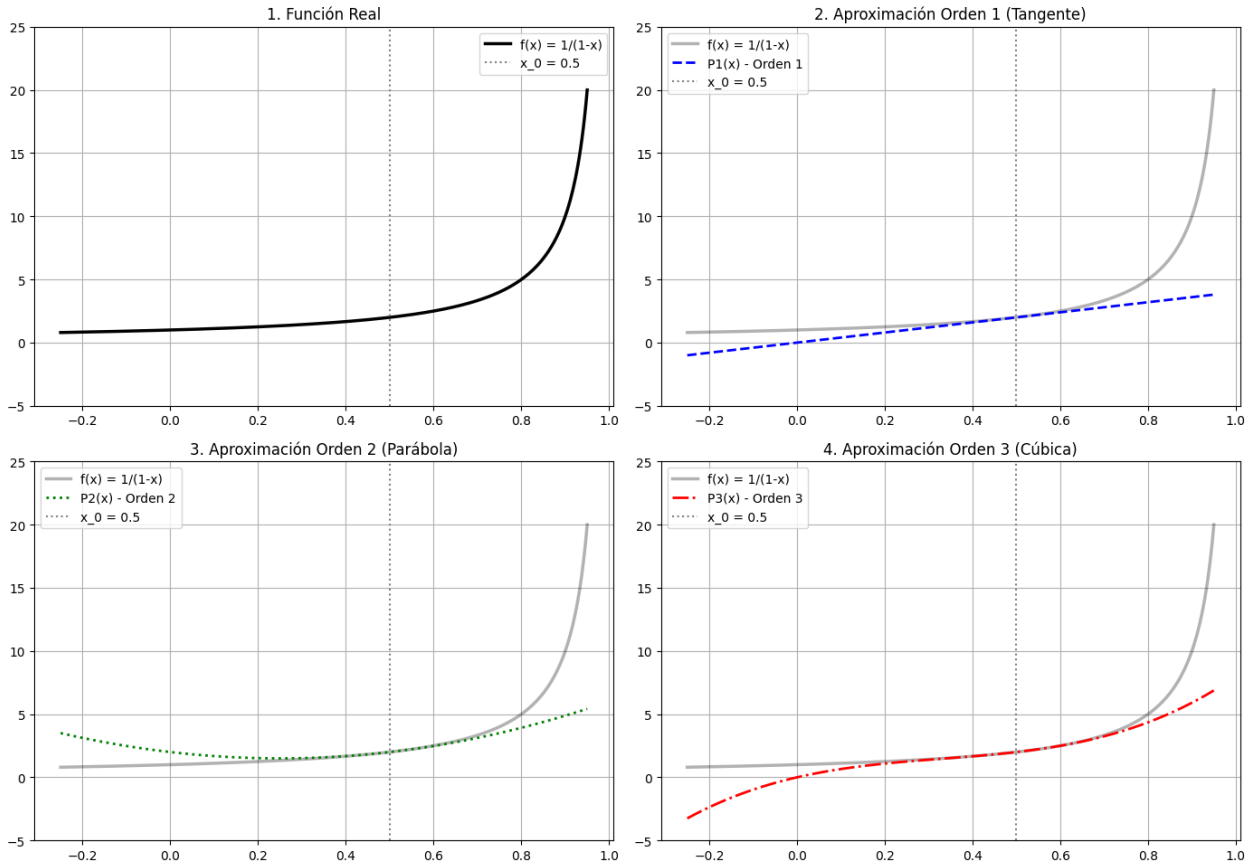
```python
import numpy as np
import matplotlib.pyplot as plt

def f_racional(x):
    return 1 / (1 - x)
```

```
x_0 = 0.5

def P1_racional(x):
    return 2 + 4 * (x - x_0)

def P2_racional(x):
    term_1 = 2 + 4 * (x - x_0)
    term_2 = 8 * (x - x_0)**2
    return term_1 + term_2

def P3_racional(x):
    term_1 = 2 + 4 * (x - x_0)
    term_2 = 8 * (x - x_0)**2
    term_3 = 16 * (x - x_0)**3
    return term_1 + term_2 + term_3

x = np.linspace(-0.25, 0.95, 400)
x_0 = 0.5

# Calculamos los valores 'y' para cada función
y_real = f_racional(x)
y_p1 = P1_racional(x)
y_p2 = P2_racional(x)
y_p3 = P3_racional(x)
```

```
print(f"Generando gráficos para f(x) = 1/(1-x) en x_0 = {x_0}...")

fig, axs = plt.subplots(2, 2, figsize=(14, 11))

fig.suptitle(f'Aproximaciones de Taylor para f(x) = 1/(1-x) en x_0 = {x_0}', fontsize=16

y_limites = [-5, 25]

#  Gráfico 1
axs[0, 0].plot(x, y_real, label='f(x) = 1/(1-x)', color='black', linewidth=2.5)
axs[0, 0].set_title('1. Función Real')
axs[0, 0].axvline(x=x_0, color='gray', linestyle=':', label=f'x_0 = {x_0}')
axs[0, 0].set_ylim(y_limites)
axs[0, 0].legend()
axs[0, 0].grid(True)

#  Gráfico 2
axs[0, 1].plot(x, y_real, label='f(x) = 1/(1-x)', color='black', linewidth=2.5, alpha=0.
axs[0, 1].plot(x, y_p1, label='P1(x) - Orden 1', color='blue', linestyle='--', linewidth
```

```
axs[0, 1].set_title('2. Aproximación Orden 1 (Tangente)')
axs[0, 1].axvline(x=x_0, color='gray', linestyle=':', label=f'x_0 = {x_0}')
axs[0, 1].set_ylim(y_limites)
axs[0, 1].legend()
axs[0, 1].grid(True)

#  Gráfico 3
axs[1, 0].plot(x, y_real, label='f(x) = 1/(1-x)', color='black', linewidth=2.5, alpha=0.
axs[1, 0].plot(x, y_p2, label='P2(x) - Orden 2', color='green', linestyle=':', linewidth
axs[1, 0].set_title('3. Aproximación Orden 2 (Parábola)')
axs[1, 0].axvline(x=x_0, color='gray', linestyle=':', label=f'x_0 = {x_0}')
axs[1, 0].set_ylim(y_limites)
axs[1, 0].legend()
axs[1, 0].grid(True)

#  Gráfico 4
axs[1, 1].plot(x, y_real, label='f(x) = 1/(1-x)', color='black', linewidth=2.5, alpha=0.
axs[1, 1].plot(x, y_p3, label='P3(x) - Orden 3', color='red', linestyle='-.', linewidth=
axs[1, 1].set_title('4. Aproximación Orden 3 (Cúbica)')
axs[1, 1].axvline(x=x_0, color='gray', linestyle=':', label=f'x_0 = {x_0}')
axs[1, 1].set_ylim(y_limites)
axs[1, 1].legend()
axs[1, 1].grid(True)

plt.tight_layout(rect=[0, 0.03, 1, 0.95])

plt.show()
```

```
Generando gráficos para f(x) = 1/(1-x) en x_0 = 0.5...
```



1.2 Encuentre el polinomio de Lagrange para los siguientes datos y grafique:

1.2.1 Problema 1

Se busca encontrar el Polinomio de Lagrange para:

$$(0,0),(30,0.5),(60,\frac{\sqrt{3}}{2}),(90,1)$$

Este es un polinomio de grado 3 porque se tienen 4 puntos

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import lagrange

x_puntos = np.array([0, 30, 60, 90])
```

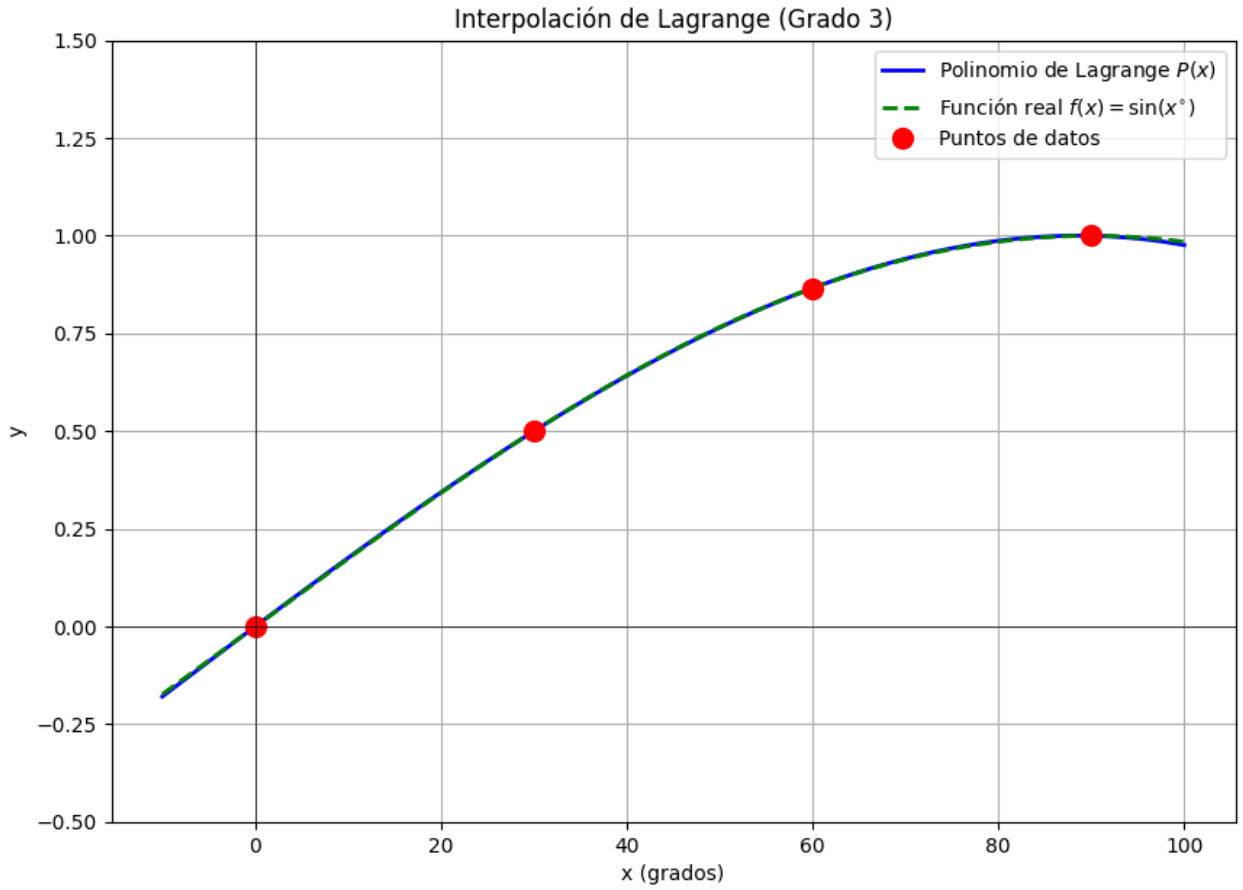
```
y_puntos = np.array([0, 0.5, np.sqrt(3)/2, 1])

print("Puntos x:", x_puntos)
print("Puntos y:", y_puntos)

# Calcular el polinomio de Lagrange
P = lagrange(x_puntos, y_puntos)

print("\nCoeficientes del polinomio P(x) (de mayor a menor grado):")
print(P.coef)

print("\nPolinomio P(x):")
print(P)
```

```
Puntos x: [ 0 30 60 90]
Puntos y: [0.        0.5       0.8660254 1.       ]

Coeficientes del polinomio P(x) (de mayor a menor grado):
[-6.05408712e-07 -1.99435471e-05  1.78098409e-02  0.00000000e+00]

Polinomio P(x):
            3             2
-6.054e-07 x - 1.994e-05 x + 0.01781 x
```

```
print("Gráfica")

x_curva = np.linspace(-10, 100, 400)
y_curva_P = P(x_curva)
y_curva_real_sin = np.sin(np.deg2rad(x_curva))


plt.figure(figsize=(10, 7))
plt.plot(x_curva, y_curva_P, label='Polinomio de Lagrange $P(x)$', color='blue', linewid
plt.plot(x_curva, y_curva_real_sin, label='Función real $f(x) = \sin(x^{\circ})$', color
plt.plot(x_puntos, y_puntos, 'o', color='red', markersize=10, label='Puntos de datos')

plt.title('Interpolación de Lagrange (Grado 3)')
plt.xlabel('x (grados)')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
```

```
plt.ylim(-0.5, 1.5) # Ajustar el eje y para mejor visibilidad

# Mostrar el gráfico
plt.show()
```

Gráfica



1.2.2 Problema 2

Se busca encontrar el Polinomio de Lagrange para:

$$(1,1),(2,2),(3,2)$$

Este es un polinomio de grado 2 porque se tienen 4 puntos

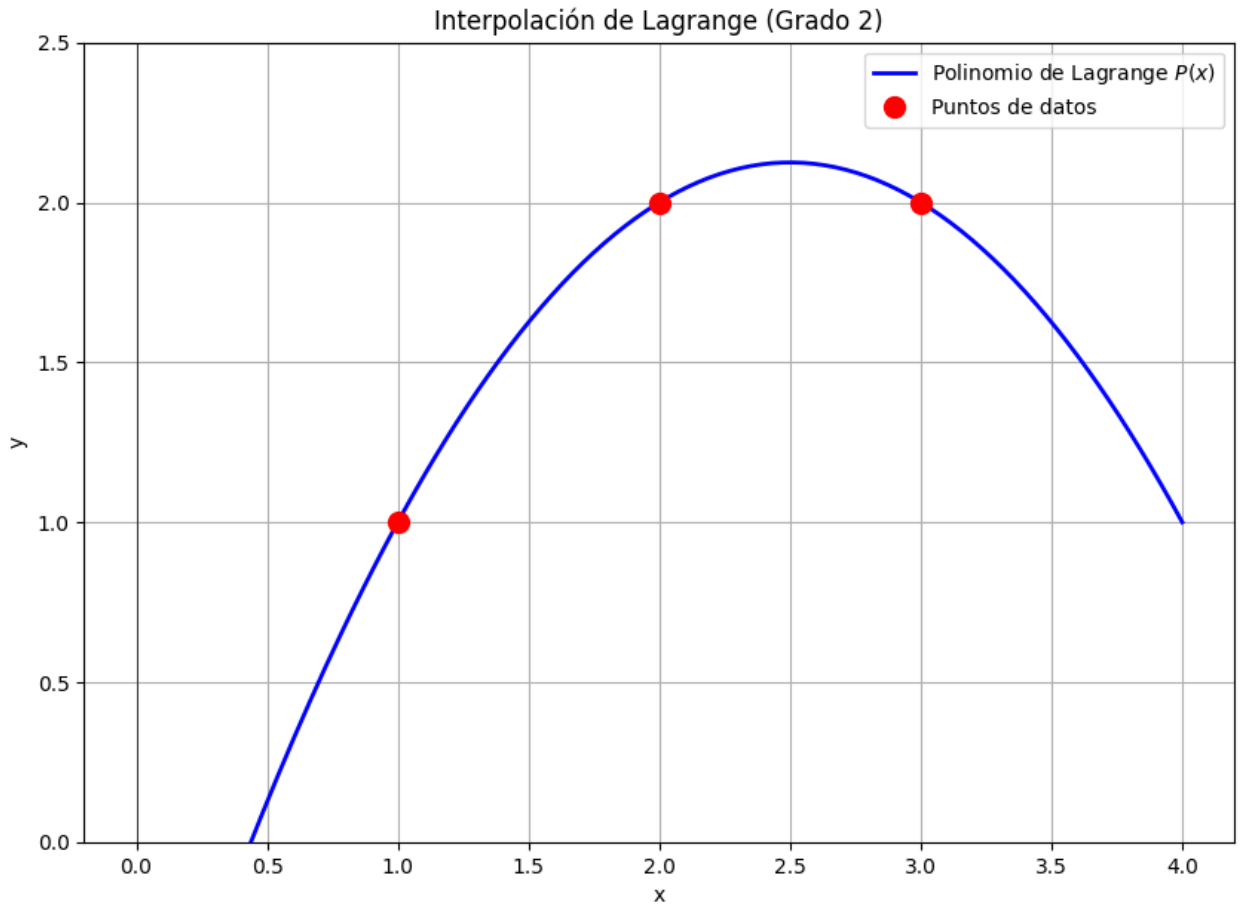
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import lagrange

x_puntos = np.array([1, 2, 3])

y_puntos = np.array([1, 2, 2])

print("Puntos x:", x_puntos)
print("Puntos y:", y_puntos)

# Calcular el polinomio de Lagrange
P = lagrange(x_puntos, y_puntos)

print("\nCoeficientes del polinomio P(x) (de mayor a menor grado):")
print(P.coef)

print("\nPolinomio P(x):")
print(P)
```

```
Puntos x: [1 2 3]
Puntos y: [1 2 2]

Coeficientes del polinomio P(x) (de mayor a menor grado):
[-0.5  2.5 -1. ]

Polinomio P(x):
      2
-0.5 x + 2.5 x - 1
```

```python
print("Gráfico")

x_curva = np.linspace(0, 4, 200)
y_curva_P = P(x_curva)
plt.figure(figsize=(10, 7))
plt.plot(x_curva, y_curva_P, label='Polinomio de Lagrange $P(x)$', color='blue', linewid
plt.plot(x_puntos, y_puntos, 'o', color='red', markersize=10, label='Puntos de datos')

plt.title('Interpolación de Lagrange (Grado 2)')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.axhline(0, color='black', linewidth=0.5)
```

```
plt.axvline(0, color='black', linewidth=0.5)
plt.ylim(0, 2.5) # Ajustar el eje y

# Mostrar el gráfico
plt.show()
```

Gráfico



1.2.3 Problema 3

Se busca encontrar el Polinomio de Lagrange para:

$$(-2, 5), (1, 7), (3, 11), (7, 34)$$

Este es un polinomio de grado 3 porque se tienen 4 puntos

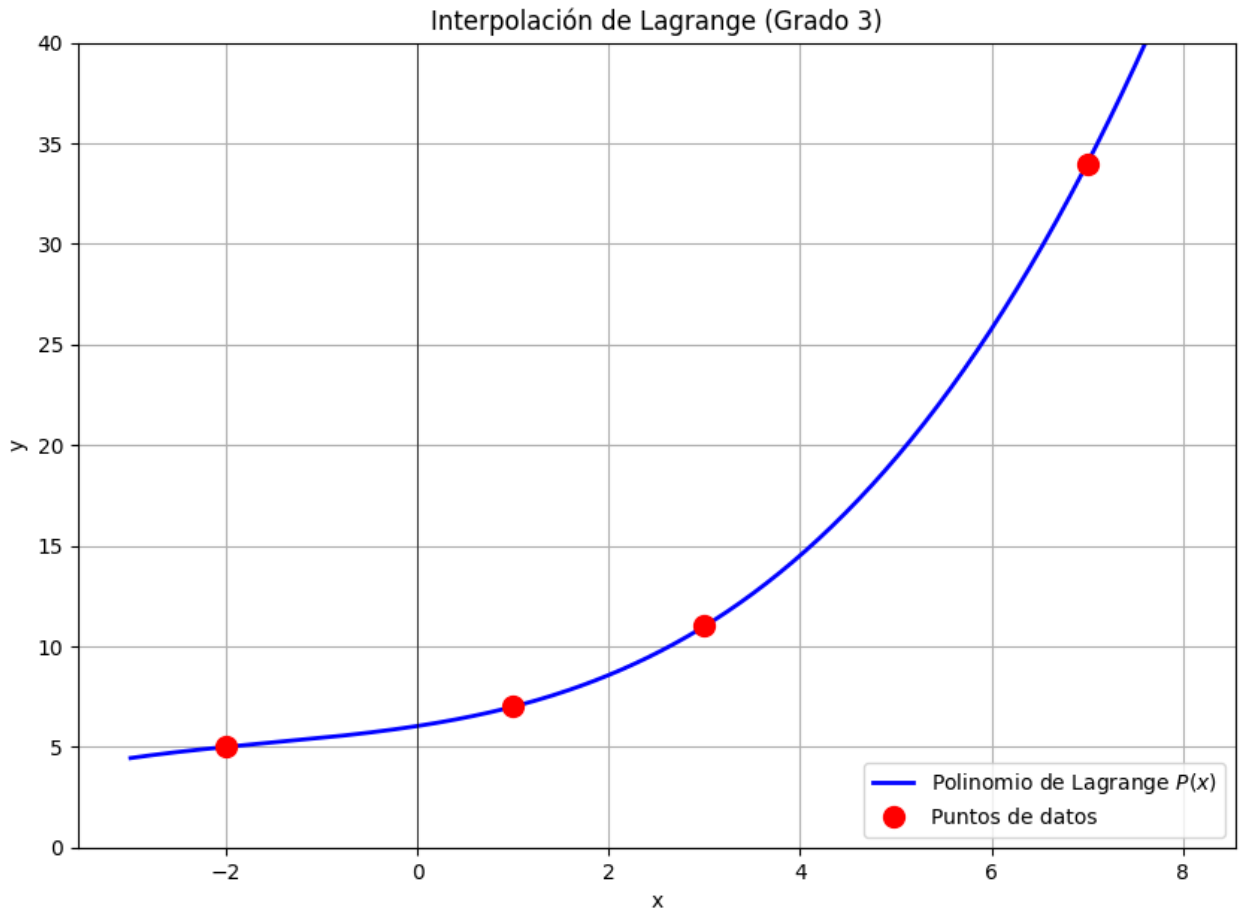
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import lagrange

x_puntos = np.array([-2, 1, 3, 7])

y_puntos = np.array([5, 7, 11, 34])

print("Puntos x:", x_puntos)
print("Puntos y:", y_puntos)

P = lagrange(x_puntos, y_puntos)

print("\nCoeficientes del polinomio P(x) (de mayor a menor grado):")
print(P.coef)

print("\nPolinomio P(x):")
print(P)
```

```
Puntos x: [-2  1  3  7]
Puntos y: [ 5  7 11 34]

Coeficientes del polinomio P(x) (de mayor a menor grado):
[0.03981481 0.18703704 0.73425926 6.03888889]

Polinomio P(x):
         3         2
0.03981 x + 0.187 x + 0.7343 x + 6.039
```

```python
print("Gráfica")

x_curva = np.linspace(-3, 8, 400)
y_curva_P = P(x_curva)


plt.figure(figsize=(10, 7))
plt.plot(x_curva, y_curva_P, label='Polinomio de Lagrange $P(x)$', color='blue', linewid
plt.plot(x_puntos, y_puntos, 'o', color='red', markersize=10, label='Puntos de datos')

plt.title('Interpolación de Lagrange (Grado 3)')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid(True)
```

```
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.ylim(0, 40) # Ajustar el eje y

# Mostrar el gráfico
plt.show()
```

Gráfica



1.3 CONCLUSIONES

1.3.1 Conclusiones sobre Polinomios de Taylor

- **Aproximación Local:** El Polinomio de Taylor aproxima una función conocida (como $\ln(x)$ o $\cos(x)$) usando la información de sus derivadas en un solo punto (x_0).
- **Mejora con el Grado:** La aproximación se vuelve más precisa cerca del punto x_0 a medida que aumentamos el orden del polinomio.

- **Requisito de Definición:** El método falla si la función o sus derivadas no están definidas en el punto x_0 (como vimos con $\ln(x)$ en $x_0 = 0$).

1.3.2 Conclusiones sobre Interpolación de Lagrange

- **Ajuste Exacto a Puntos:** El Polinomio de Lagrange crea una nueva función que pasa exactamente por un conjunto de puntos de datos discretos.
- **Útil para Datos Tabulados:** Es ideal cuando no se conoce la función original, sino que solo se tiene una tabla de valores como lo visto en los problemas 1, 2 y 3.
- **Grado Definido por Puntos:** El grado del polinomio resultante es siempre uno menos que el número de puntos utilizados.