Tarea 06: Serie de Taylor y Polinomios de Lagrange

Pasquel Johann

Tabla de Contenidos

GITHUB **1**

CONJUNTO DE EJERCICIOS **1**

Determine el orden de la mejor aproximación para las siguientes funciones, usando la Serie de Taylor y el Polinomio de Lagrange: 1

GITHUB

https://github.com/Vladimirjon/MetodosNumericos_PasquelJohann

CONJUNTO DE EJERCICIOS

Determine el orden de la mejor aproximación para las siguientes funciones, usando la Serie de Taylor y el Polinomio de Lagrange:

1. $\frac{1}{25 \cdot x^2+1}$, $x_0 = 0$

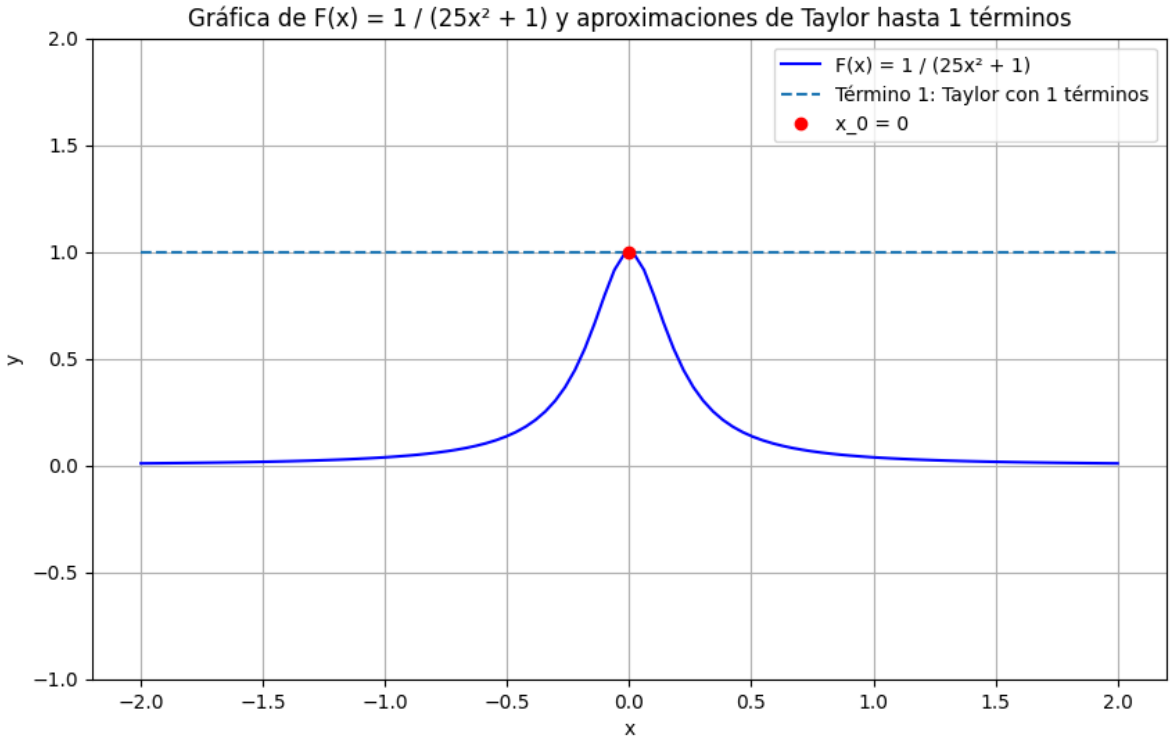
Serie de Taylor

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

```python
# Equation F(x) = 1 / (25 * x^2 + 1)
def equation(x):
    return 1 / (25 * x**2 +1)
```

```python
def taylor_series_f(x, terms):
    taylor_sum = np.zeros_like(x)
    for n in range(terms):
        coefficient = (-1)**n * math.factorial(2 * n) / (math.factorial(n) * (25**n))
        term = coefficient * x**(2 * n)
        taylor_sum += term
    return taylor_sum
```

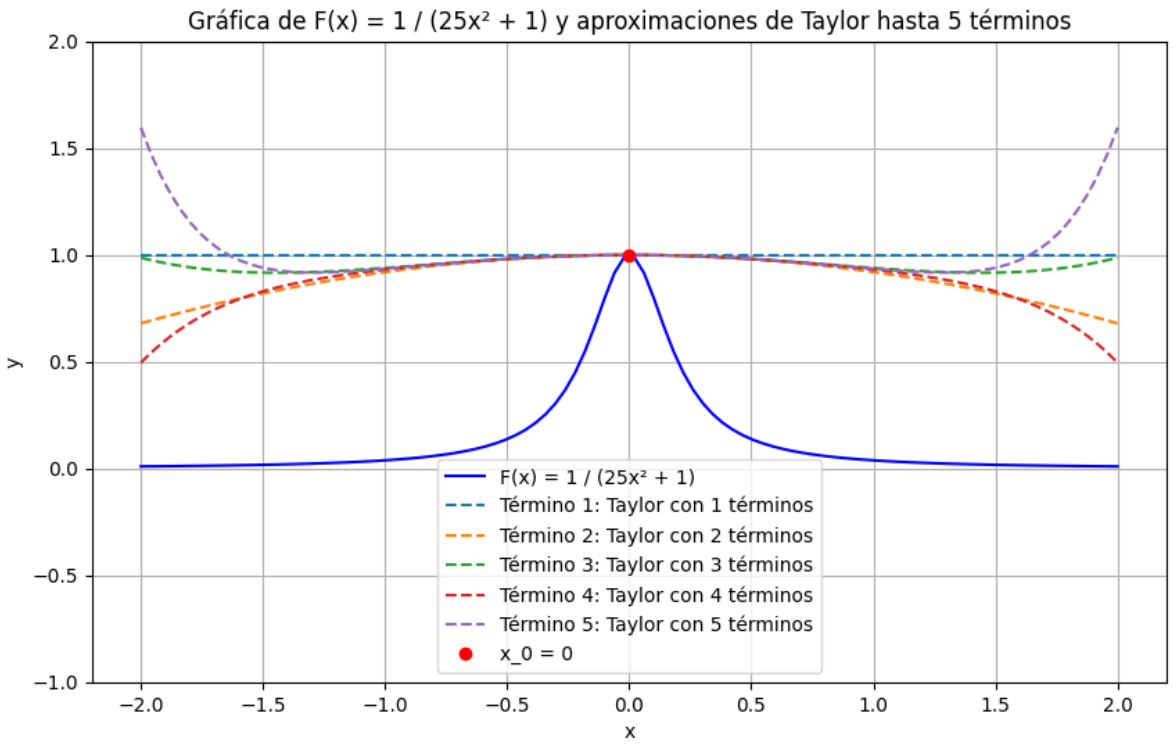
```python
def plot_f_with_taylor(x, max_terms):
    y = equation(x)
    plt.figure(figsize=(10, 6))
    plt.plot(x, y, label='F(x) = 1 / (25x² + 1)', color='blue')
    for terms in range(1, max_terms + 1):
        y_taylor = taylor_series_f(x, terms)
        plt.plot(x, y_taylor, '--', label=f'Término {terms}: Taylor con {terms} términos')

    # x_0 = 0
    plt.plot(0, equation(0), 'ro', label='x_0 = 0')

    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(f'Gráfica de F(x) = 1 / (25x² + 1) y aproximaciones de Taylor hasta {max_terms}
    plt.ylim([-1, 2])
    plt.grid(True)
    plt.legend()
    plt.show()

# Intervalo de x y con 1 término
x = np.linspace(-2, 2, 100)
plot_f_with_taylor(x, 1)
```



$$P_1(x) = 1$$

```python
def plot_f_with_taylor(x, max_terms):
    y = equation(x)
    plt.figure(figsize=(10, 6))
    plt.plot(x, y, label='F(x) = 1 / (25x² + 1)', color='blue')
    for terms in range(1, max_terms + 1):
        y_taylor = taylor_series_f(x, terms)
        plt.plot(x, y_taylor, '--', label=f'Término {terms}: Taylor con {terms} términos')

    # x_0 = 0
    plt.plot(0, equation(0), 'ro', label='x_0 = 0')

    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(f'Gráfica de F(x) = 1 / (25x² + 1) y aproximaciones de Taylor hasta {max_terms}
    plt.ylim([-1, 2])
    plt.grid(True)
    plt.legend()
    plt.show()
```

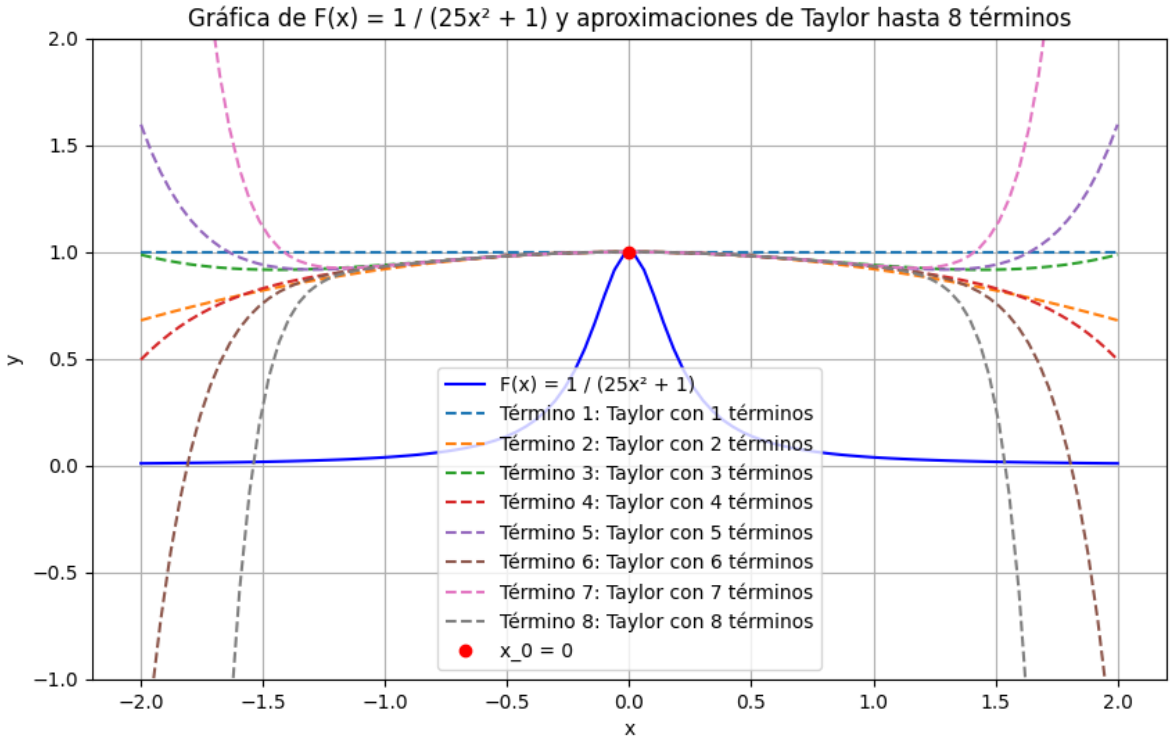
```
# Intervalo de x y con 5 términos
x = np.linspace(-2, 2, 100)
plot_f_with_taylor(x, 5)
```



$$P_5(x) = 1 - 25x^2 + 625x^4$$

```
def plot_f_with_taylor(x, max_terms):
    y = equation(x)
    plt.figure(figsize=(10, 6))
    plt.plot(x, y, label='F(x) = 1 / (25x² + 1)', color='blue')
    for terms in range(1, max_terms + 1):
        y_taylor = taylor_series_f(x, terms)
        plt.plot(x, y_taylor, '--', label=f'Término {terms}: Taylor con {terms} términos')

    # x_0 = 0
    plt.plot(0, equation(0), 'ro', label='x_0 = 0')

    plt.xlabel('x')
    plt.ylabel('y')
```

```
    plt.title(f'Gráfica de F(x) = 1 / (25x² + 1) y aproximaciones de Taylor hasta {max_terms}
    plt.ylim([-1, 2])
    plt.grid(True)
    plt.legend()
    plt.show()

# Intervalo de x y con 8 términos
x = np.linspace(-2, 2, 100)
plot_f_with_taylor(x, 8)
```



$$P_8(x) = 1 - 25x^2 + 625x^4 - 15625x^6 + 390625x^8$$

Para esta función $F(x) = \frac{1}{25x^2+1}$, la Serie de Taylor **no proporciona una aproximación práctica para valores lejanos a** $x = 0$.

La mejor aproximación usando Taylor sería con **pocos términos y en un rango muy cercano a** $x = 0$

La Serie de Taylor **no es ideal** para esta función. Esta función es una **racional**, con una dependencia cuadrática en el denominador $25x^2 + 1$. Cuando se añaden más términos a la Serie de Taylor, los polinomios tienden a oscilar y divergir lejos de $x_0 = 0$.

Polinomio de Lagrange

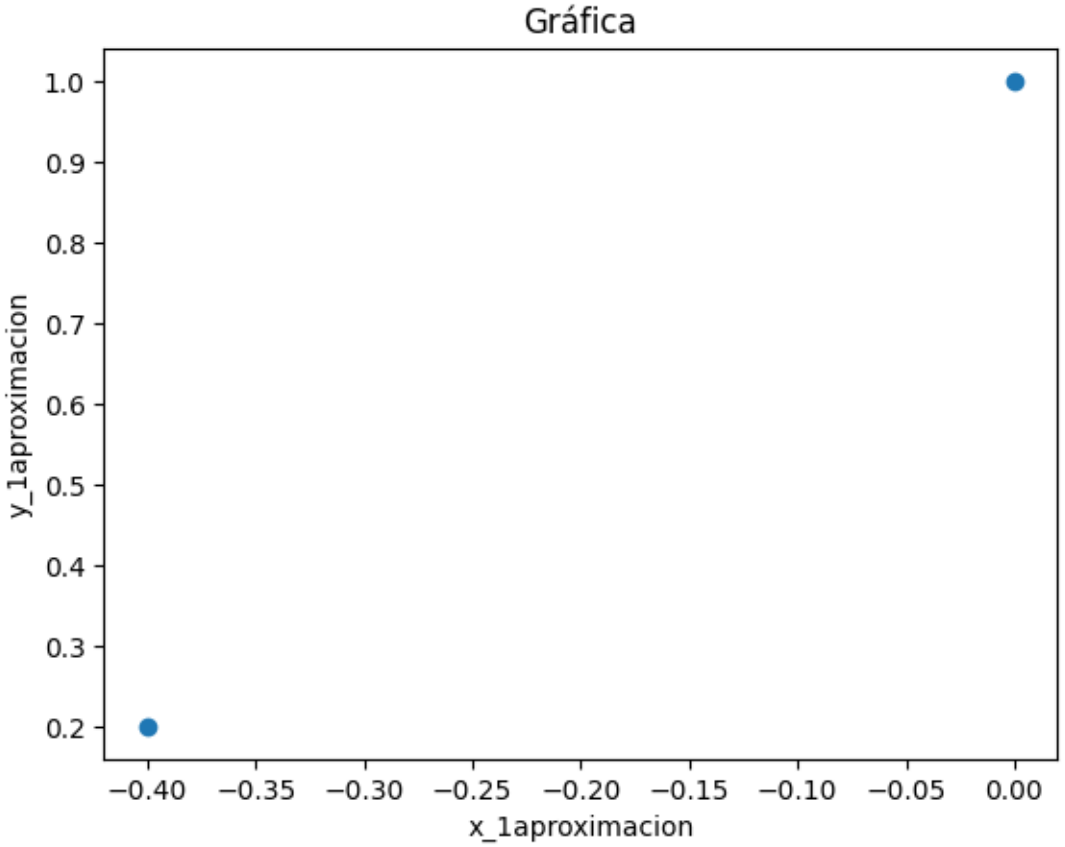
En este caso, para construir el polinomio de Lagrange de $F(x) = \frac{1}{25x^2+1}$, usaremos diferentes cantidades de puntos para las distintas aproximaciones.

Primera aproximación

Puntos: $(0, 1)$, $(-0.4, 0.2)$

```
x_1aproximacion = [0,-0.4]
y_1aproximacion = [1, 0.2]
```

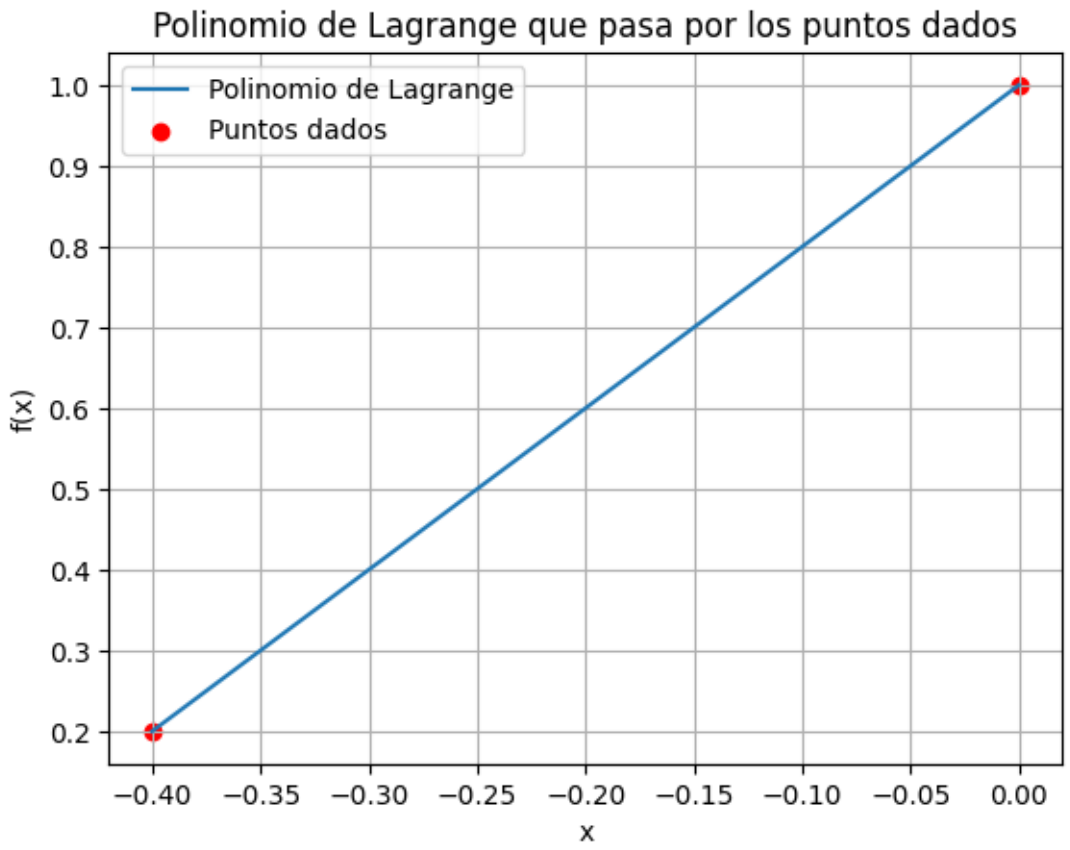
```
import matplotlib.pyplot as plt

plt.scatter(x_1aproximacion,y_1aproximacion)
plt.xlabel("x_1aproximacion")
plt.ylabel("y_1aproximacion")
plt.title("Gráfica")
plt.show()
```



```python
def lagrange_polynomial(x: float, x_points: np.ndarray, y_points: np.ndarray) -> float:
    n = len(x_points)
    result = 0
    for k in range(n):
        L_k = 1
        for i in range(n):
            if i != k:
                L_k *= (x - x_points[i]) / (x_points[k] - x_points[i])
        result += y_points[k] * L_k
    return result
```

```python
x_values_1aproximacion = np.linspace(min(x_1aproximacion), max(x_1aproximacion), 500)
y_values_1aproximacion = [lagrange_polynomial(x, x_1aproximacion, y_1aproximacion) for x in x
```

```python
plt.plot(x_values_1aproximacion, y_values_1aproximacion, label="Polinomio de Lagrange")
plt.scatter(x_1aproximacion, y_1aproximacion, color="red", label="Puntos dados")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.title("Polinomio de Lagrange que pasa por los puntos dados")
plt.legend()
plt.grid()
plt.show()
```



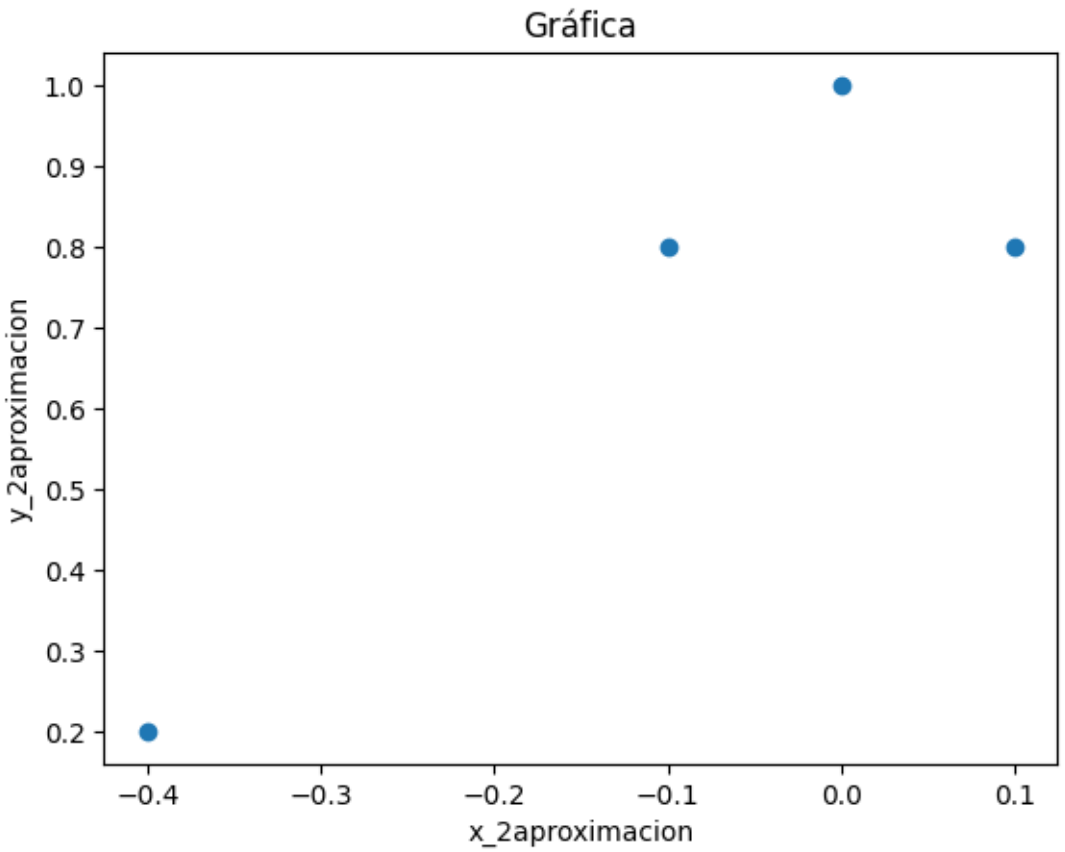
$$P(x) = 2x + 1$$

Segunda aproximación

Puntos: $(0, 1)$, $(-0.4, 0.2)$, $(0.1, 0.8)$, $(-0.1, 0.8)$

```
x_2aproximacion = [0,-0.4, 0.1, -0.1]
y_2aproximacion = [1, 0.2, 0.8, 0.8]
```

```
import matplotlib.pyplot as plt

plt.scatter(x_2aproximacion,y_2aproximacion)
plt.xlabel("x_2aproximacion")
plt.ylabel("y_2aproximacion")
plt.title("Gráfica")
plt.show()
```

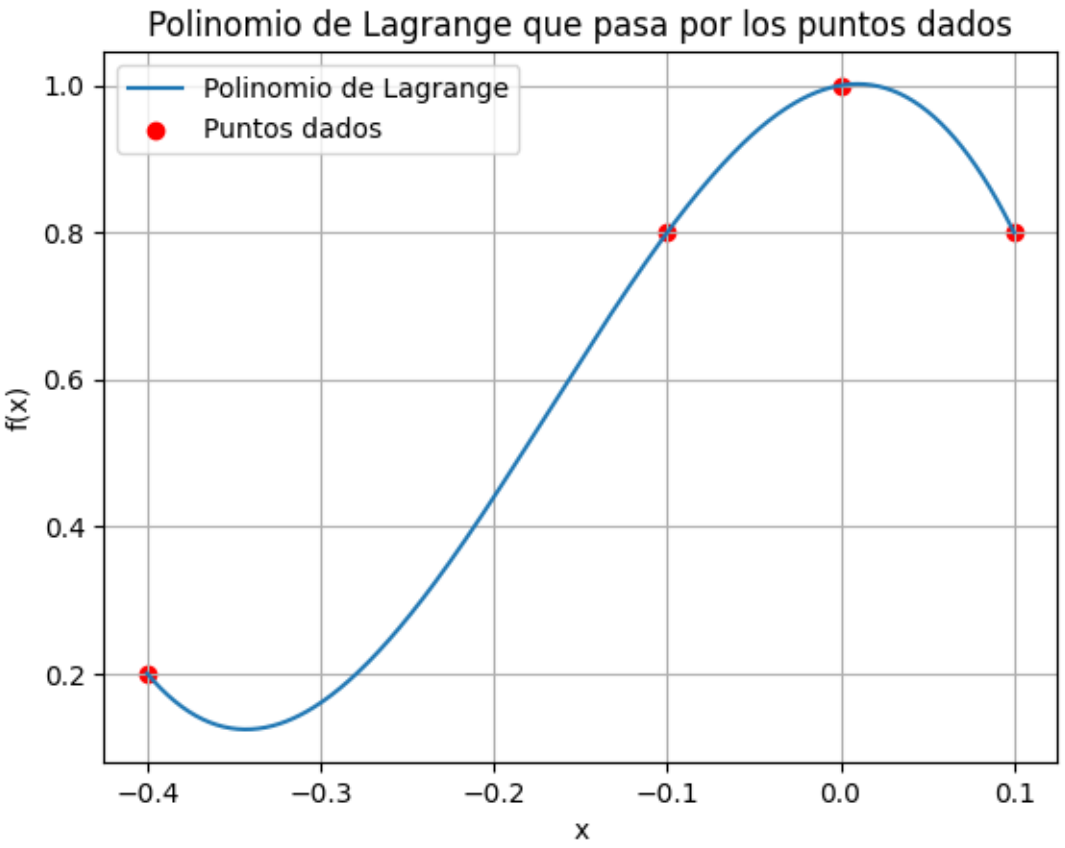



```python
def lagrange_polynomial(x: float, x_points: np.ndarray, y_points: np.ndarray) -> float:
    n = len(x_points)
    result = 0
    for k in range(n):
        L_k = 1
        for i in range(n):
            if i != k:
                L_k *= (x - x_points[i]) / (x_points[k] - x_points[i])
        result += y_points[k] * L_k
    return result
```

```python
x_values_2aproximacion = np.linspace(min(x_2aproximacion), max(x_2aproximacion), 500)
y_values_2aproximacion = [lagrange_polynomial(x, x_2aproximacion, y_2aproximacion) for x in
```

```python
plt.plot(x_values_2aproximacion, y_values_2aproximacion, label="Polinomio de Lagrange")
plt.scatter(x_2aproximacion, y_2aproximacion, color="red", label="Puntos dados")
plt.xlabel("x")
```

```
plt.ylabel("f(x)")
plt.title("Polinomio de Lagrange que pasa por los puntos dados")
plt.legend()
plt.grid()
plt.show()
```



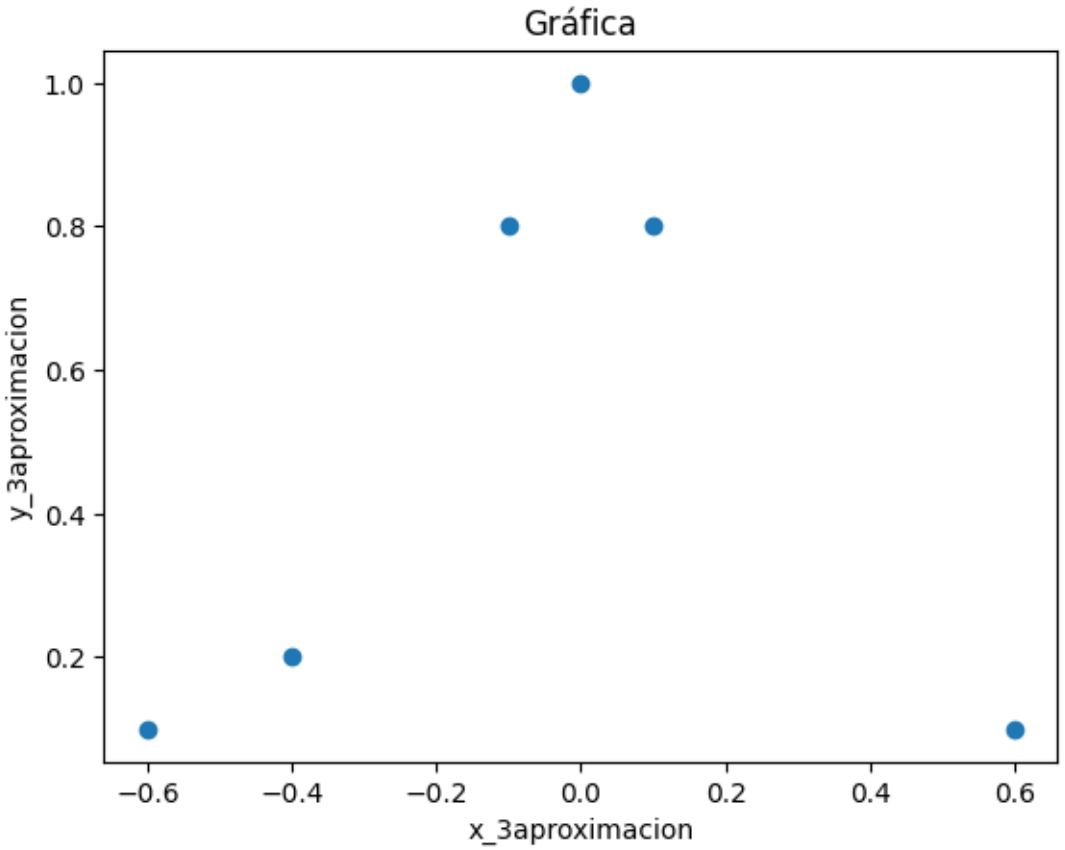
$$P(x) = -40,13x^3 - 20.04x^2 + 0.4053x + 1$$

Tercera aproximación

Puntos: $(0,1)$, $(-0.4,0.2)$, $(0.1,0.8)$, $(-0.1,0.8)$, $(-0.6,0.1)$, $(0.6,0.1)$

```
x_3aproximacion = [0,-0.4, 0.1, -0.1, -0.6, 0.6]
y_3aproximacion = [1, 0.2, 0.8, 0.8, 0.1, 0.1]
```

```
import matplotlib.pyplot as plt

plt.scatter(x_3aproximacion,y_3aproximacion)
plt.xlabel("x_3aproximacion")
plt.ylabel("y_3aproximacion")
plt.title("Gráfica")
plt.show()
```

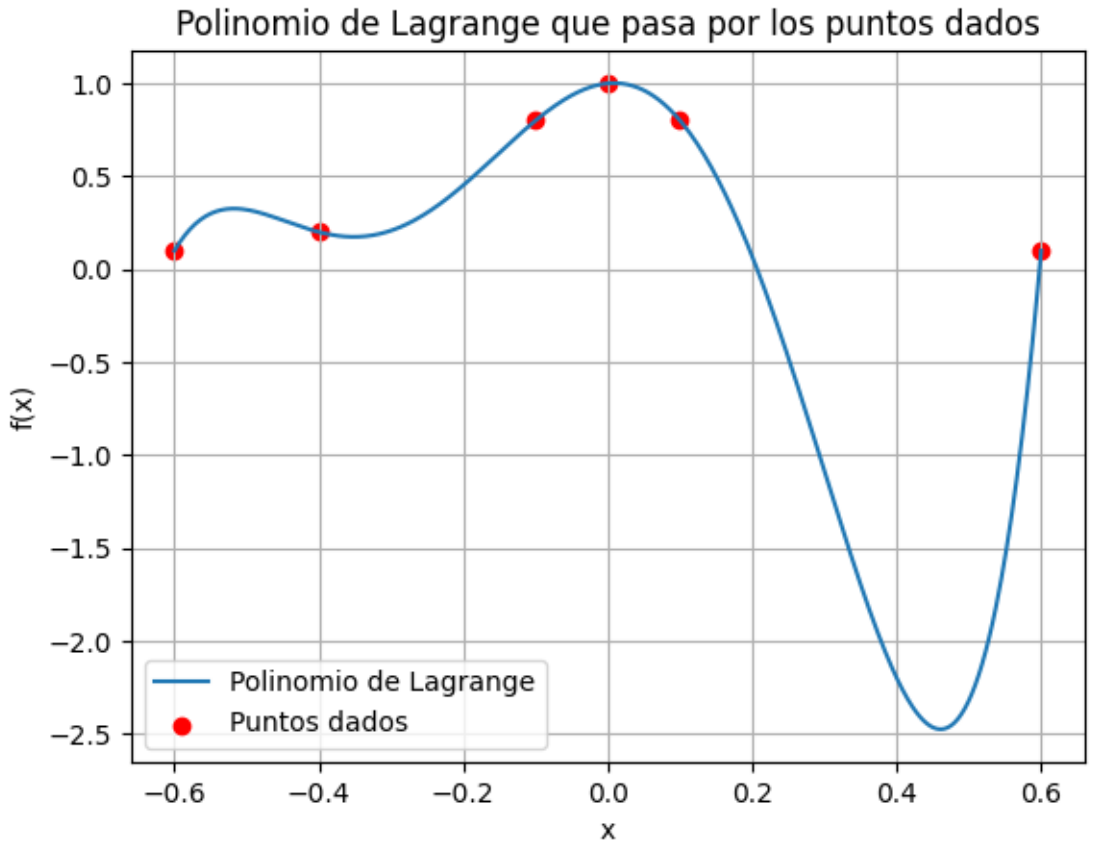



```
def lagrange_polynomial(x: float, x_points: np.ndarray, y_points: np.ndarray) -> float:
    n = len(x_points)
    result = 0
    for k in range(n):
        L_k = 1
        for i in range(n):
            if i != k:
                L_k *= (x - x_points[i]) / (x_points[k] - x_points[i])
```

```
        result += y_points[k] * L_k
    return result
```

```
x_values_3aproximacion = np.linspace(min(x_3aproximacion), max(x_3aproximacion), 500)
y_values_3aproximacion = [lagrange_polynomial(x, x_3aproximacion, y_3aproximacion) for x in x
```

```
plt.plot(x_values_3aproximacion, y_values_3aproximacion, label="Polinomio de Lagrange")
plt.scatter(x_3aproximacion, y_3aproximacion, color="red", label="Puntos dados")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.title("Polinomio de Lagrange que pasa por los puntos dados")
plt.legend()
plt.grid()
plt.show()
```



Con menos puntos **(como en el caso de 2 puntos), el polinomio es lineal** y ofrece una aproximación más estable, pero menos precisa.

Con más puntos **(como 6 puntos)**, el polinomio es de grado alto $(n-1)$, lo que introduce oscilaciones significativas.

La mejor aproximación ocurre cuando se utilizan pocos puntos en un intervalo reducido, evitando grados altos del polinomio que generen oscilaciones.

Para esta función $F(x) = \frac{1}{25x^2+1}$, el Polinomio de Lagrange funciona mejor con un número moderado de puntos. Usar más puntos tiende a introducir oscilaciones que degradan la precisión, especialmente en los extremos.

2. $arctanx$**,** $x_0 = 1$

Serie de Taylor

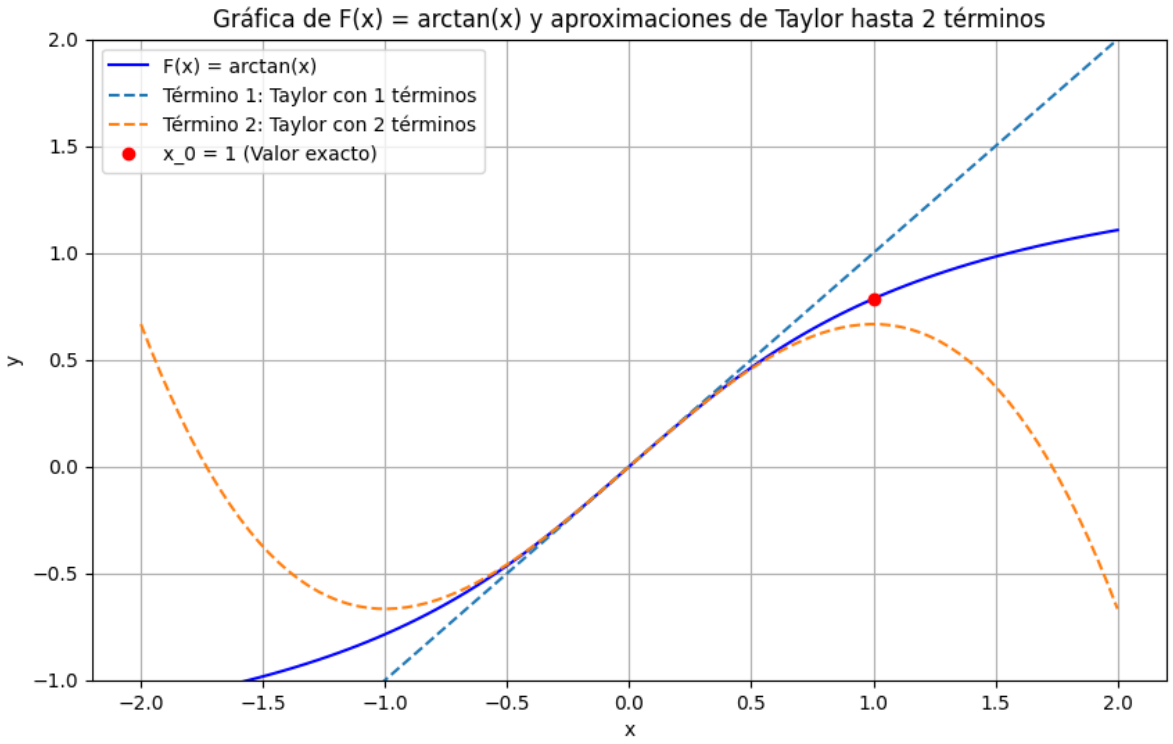
```
import numpy as np
import matplotlib.pyplot as plt
import math
```

```
# Equation F(x) = arctan(x)
def equation(x):
    return np.arctan(x)
```

```
def taylor_series_arctan(x, terms):
    taylor_sum = np.zeros_like(x)
    for n in range(terms):
        coefficient = (-1)**n / (2 * n + 1)
        term = coefficient * x**(2 * n + 1)
        taylor_sum += term
    return taylor_sum
```

```
def plot_arctan_with_taylor(x, max_terms):
    y = np.arctan(x)

    plt.figure(figsize=(10, 6))
    plt.plot(x, y, label='F(x) = arctan(x)', color='blue')

    for terms in range(1, max_terms + 1):
        y_taylor = taylor_series_arctan(x, terms)
        plt.plot(x, y_taylor, '--', label=f'Término {terms}: Taylor con {terms} términos')

    # Punto x_0 = 1
```

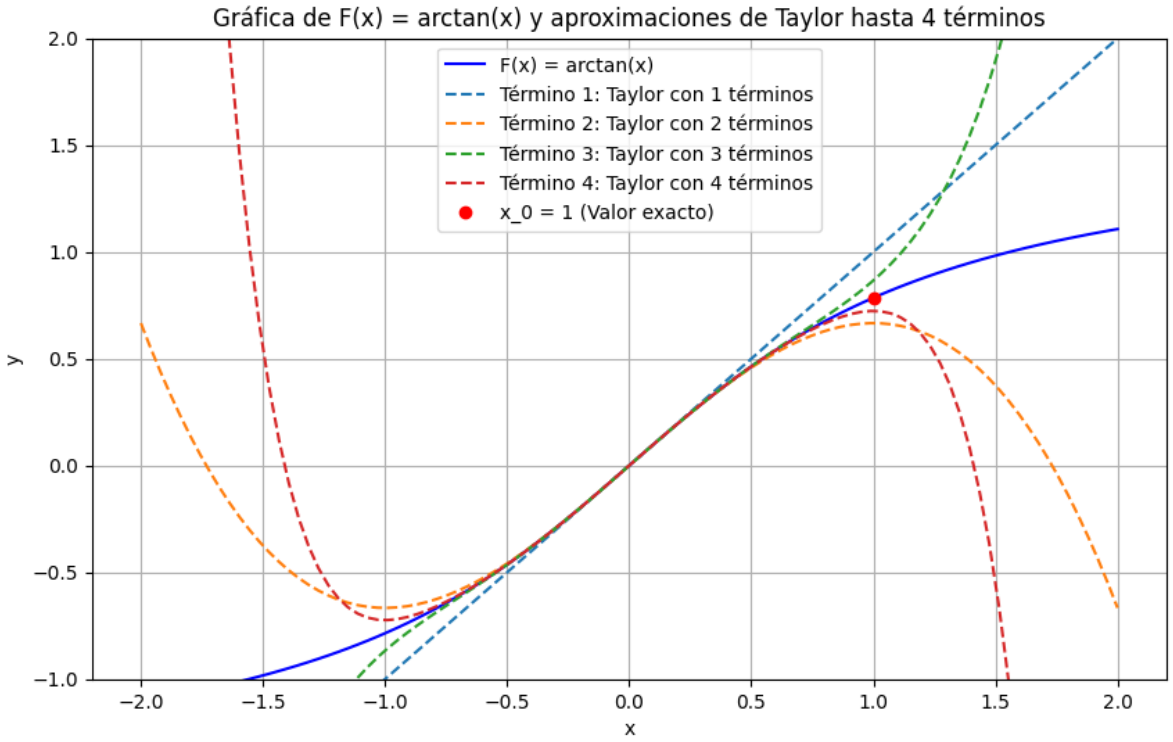
```
    plt.plot(1, np.arctan(1), 'ro', label='x_0 = 1 (Valor exacto)')

    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(f'Gráfica de F(x) = arctan(x) y aproximaciones de Taylor hasta {max_terms} tér
    plt.ylim([-1, 2])
    plt.grid(True)
    plt.legend()
    plt.show()


x = np.linspace(-2, 2, 100)
plot_arctan_with_taylor(x, 2)  # Ejemplo con 2 términos
```



$$P_2(x) = \frac{\pi}{4} + \frac{1}{2}(x-1) - \frac{1}{4}(x-1)^2$$

```
def plot_arctan_with_taylor(x, max_terms):
    y = np.arctan(x)
```

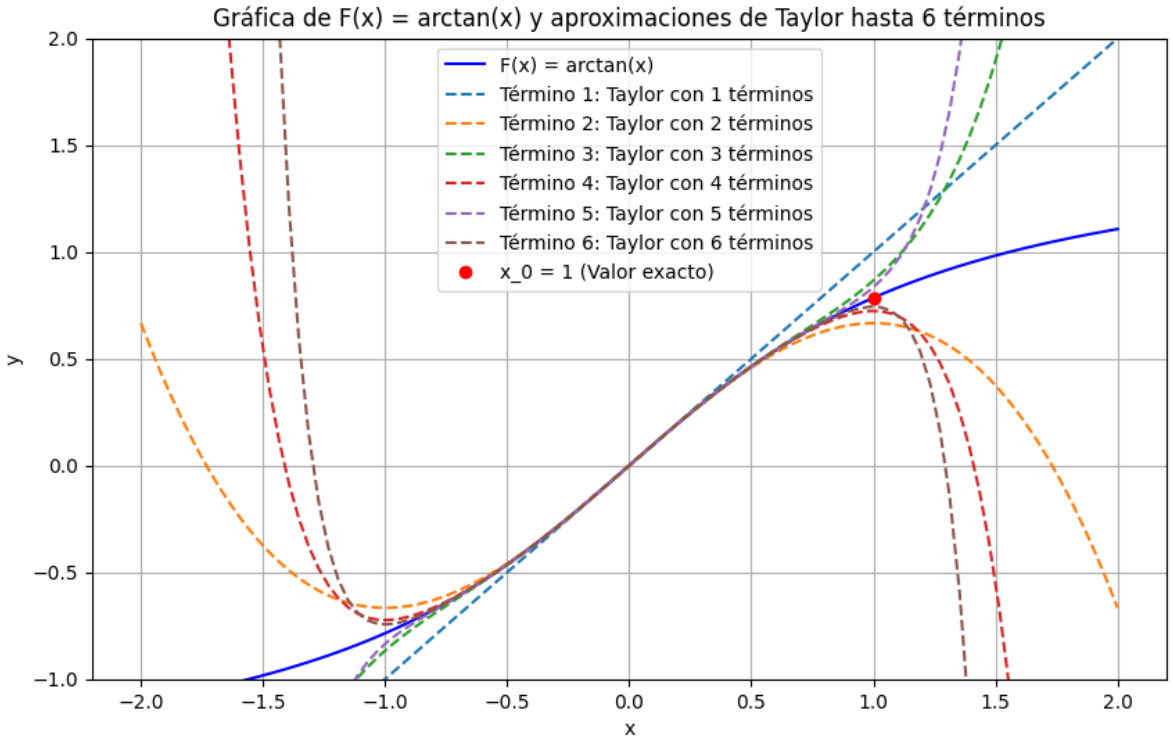
```
    plt.figure(figsize=(10, 6))
    plt.plot(x, y, label='F(x) = arctan(x)', color='blue')

    for terms in range(1, max_terms + 1):
        y_taylor = taylor_series_arctan(x, terms)
        plt.plot(x, y_taylor, '--', label=f'Término {terms}: Taylor con {terms} términos')

    # Punto x_0 = 1
    plt.plot(1, np.arctan(1), 'ro', label='x_0 = 1 (Valor exacto)')

    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(f'Gráfica de F(x) = arctan(x) y aproximaciones de Taylor hasta {max_terms} tér
    plt.ylim([-1, 2])
    plt.grid(True)
    plt.legend()
    plt.show()


x = np.linspace(-2, 2, 100)
plot_arctan_with_taylor(x, 4)  # Ejemplo con 4 términos
```



$$P_4(x) = \frac{\pi}{4} + \frac{1}{2}(x-1) - \frac{1}{4}(x-1)^2 + \frac{1}{12}(x-1)^3$$

```python
def plot_arctan_with_taylor(x, max_terms):
    y = np.arctan(x)

    plt.figure(figsize=(10, 6))
    plt.plot(x, y, label='F(x) = arctan(x)', color='blue')

    for terms in range(1, max_terms + 1):
        y_taylor = taylor_series_arctan(x, terms)
        plt.plot(x, y_taylor, '--', label=f'Término {terms}: Taylor con {terms} términos')

    # Punto x_0 = 1
    plt.plot(1, np.arctan(1), 'ro', label='x_0 = 1 (Valor exacto)')

    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(f'Gráfica de F(x) = arctan(x) y aproximaciones de Taylor hasta {max_terms} tér
    plt.ylim([-1, 2])
    plt.grid(True)
```

```
    plt.legend()
    plt.show()


x = np.linspace(-2, 2, 100)
plot_arctan_with_taylor(x, 6)  # Ejemplo con 6 términos
```



$$P_6(x) = \frac{\pi}{4} + \frac{1}{2}(x-1) - \frac{1}{4}(x-1)^2 + \frac{1}{12}(x-1)^3 - \frac{1}{40}(x-1)^5 + \frac{1}{64}(x-1)^6$$

La serie de Taylor tiene un rango de convergencia limitado por su punto de expansión $x_0 = 1$

El orden óptimo depende del rango en el que se desee precisión y el nivel de error aceptable. Para aplicaciones prácticas, evaluar el error puede ayudar a determinar cuántos términos son suficientes.

A medida que incluimos términos de mayor orden, la acumulación de errores numéricos o la complejidad de los coeficientes puede impactar ligeramente la precisión de la gráfica.

La función $arctan(x)$ **tiene un crecimiento lento y la serie de Taylor no capta este comportamiento de forma eficiente con pocos términos.**

Polinomio de Lagrange

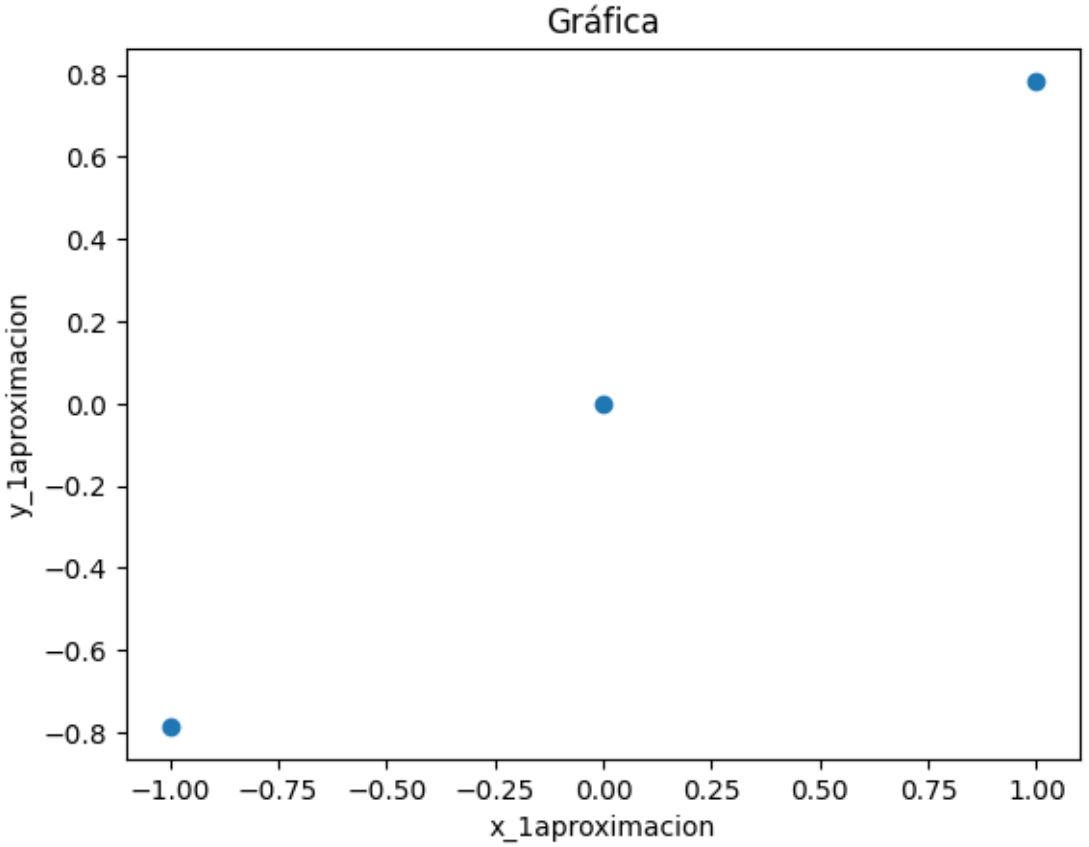
En este caso, para construir el polinomio de Lagrange de $arctanx$, usaremos diferentes cantidades de puntos para las distintas aproximaciones.

Primera aproximación

Puntos: $(0, 0)$, $(1, 0.785)$, $(-1, -0.785)$

```
x_1aproximacion = [0, 1, -1]
y_1aproximacion = [0, 0.785, -0.785]
```

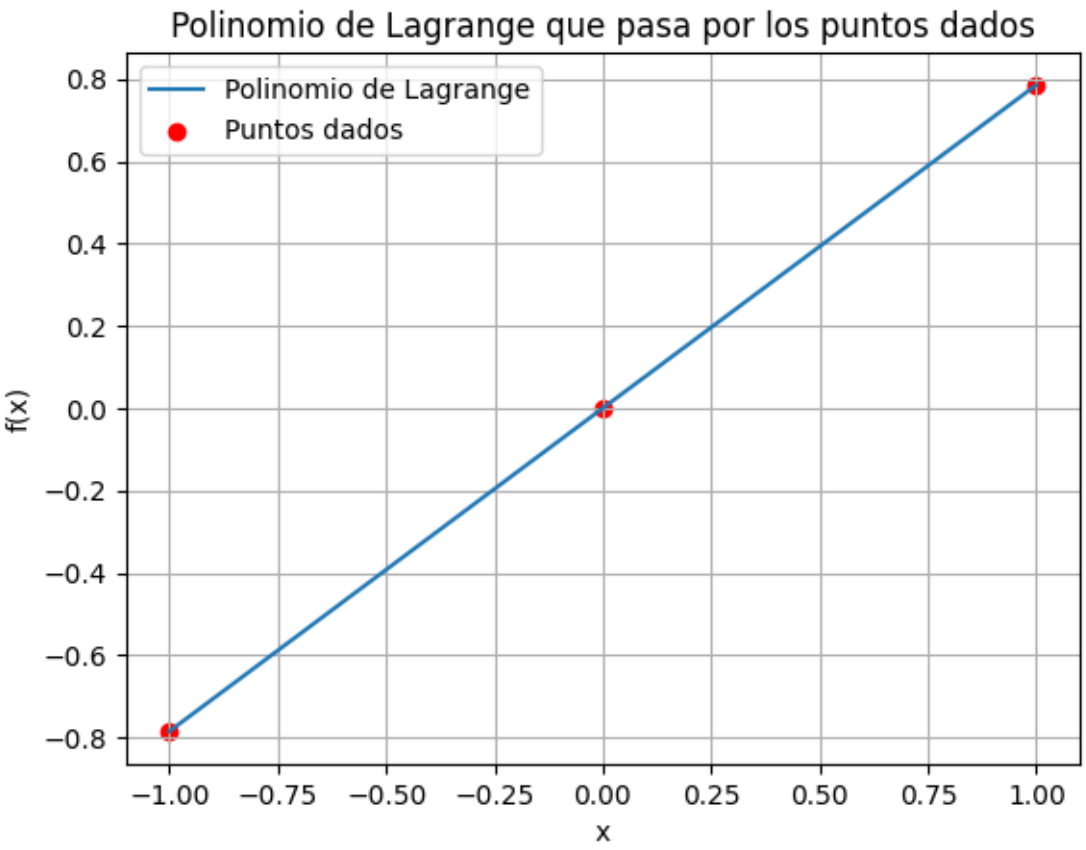
```
import matplotlib.pyplot as plt

plt.scatter(x_1aproximacion,y_1aproximacion)
plt.xlabel("x_1aproximacion")
plt.ylabel("y_1aproximacion")
plt.title("Gráfica")
plt.show()
```



```
def lagrange_polynomial(x: float, x_points: np.ndarray, y_points: np.ndarray) -> float:
    n = len(x_points)
    result = 0
    for k in range(n):
        L_k = 1
        for i in range(n):
            if i != k:
                L_k *= (x - x_points[i]) / (x_points[k] - x_points[i])
        result += y_points[k] * L_k
    return result
```

```
x_values_1aproximacion = np.linspace(min(x_1aproximacion), max(x_1aproximacion), 500)
y_values_1aproximacion = [lagrange_polynomial(x, x_1aproximacion, y_1aproximacion) for x in x
```

```
plt.plot(x_values_1aproximacion, y_values_1aproximacion, label="Polinomio de Lagrange")
plt.scatter(x_1aproximacion, y_1aproximacion, color="red", label="Puntos dados")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.title("Polinomio de Lagrange que pasa por los puntos dados")
plt.legend()
plt.grid()
plt.show()
```



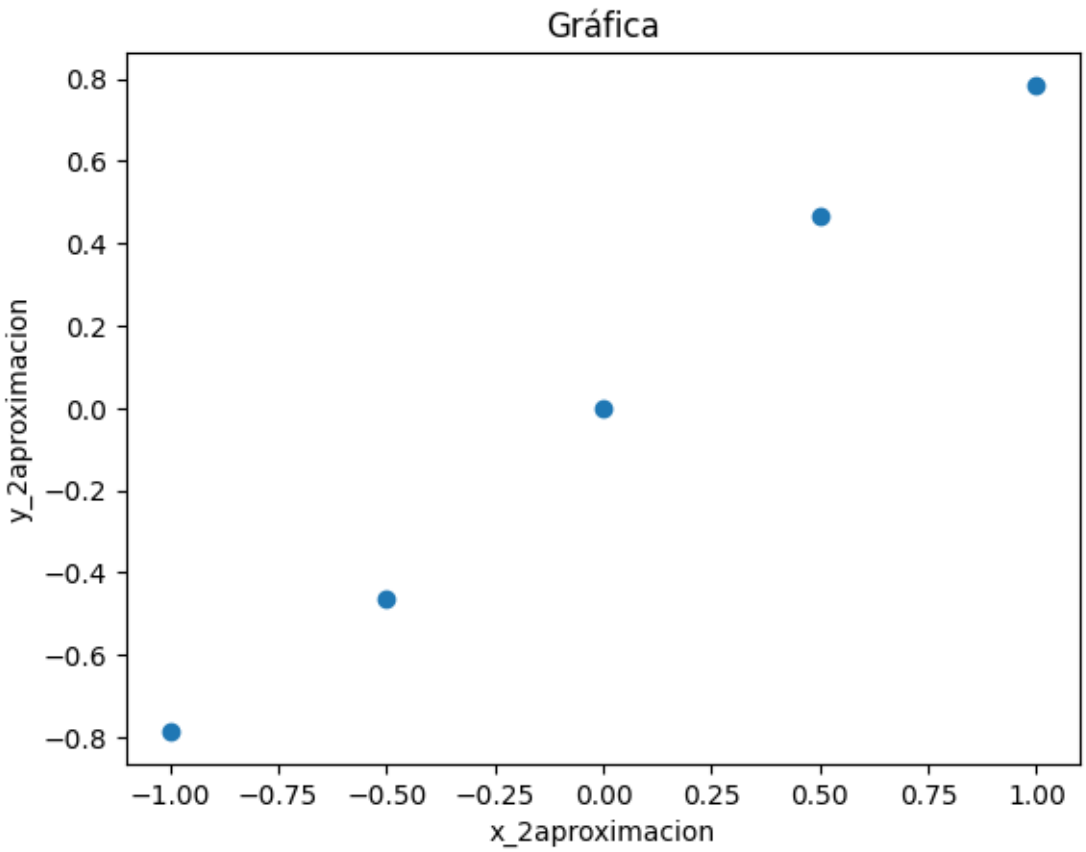
$$P(x) = 0.785x$$

Segunda aproximación

Puntos: $(0, 0)$, $(1, 0.785)$, $(-1, -0.785)$, $(0.5, 0.464)$, $(-0.5, -0.464)$

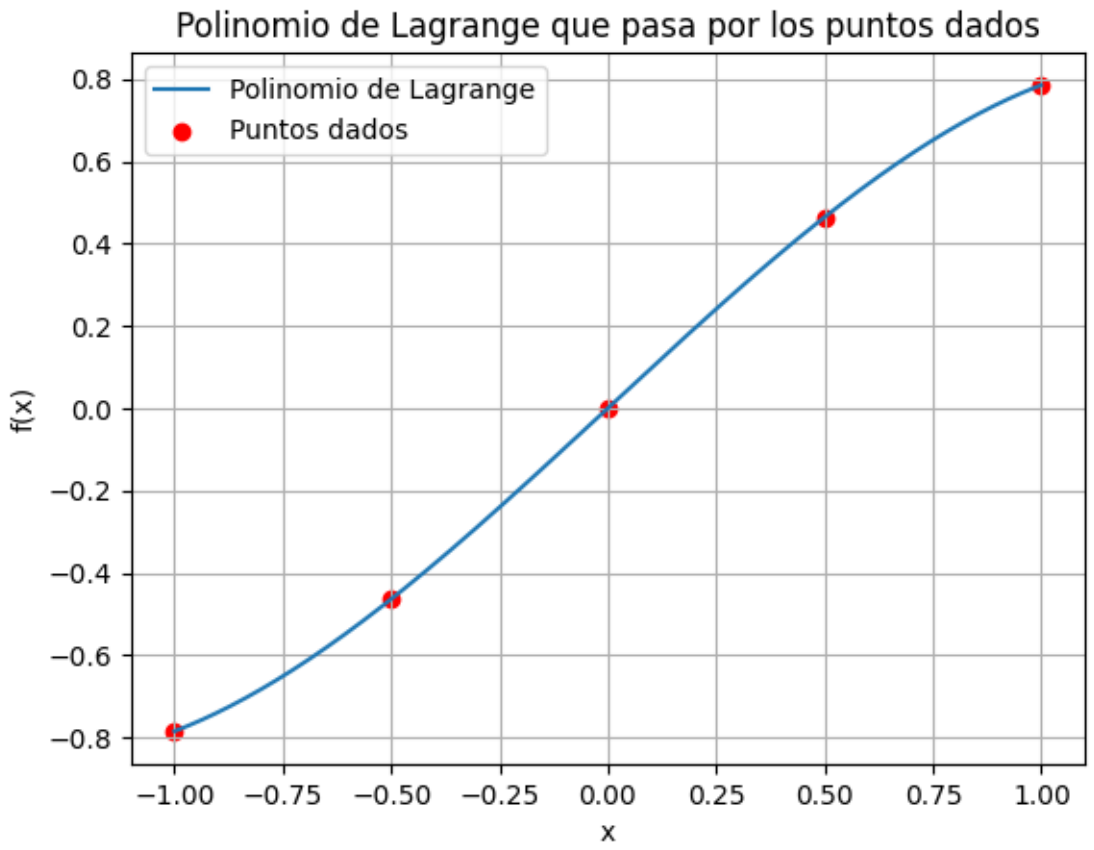
```
x_2aproximacion = [0, 1, -1, 0.5, -0.5]
y_2aproximacion = [0, 0.785, -0.785, 0.464, -0.464]
```

```
import matplotlib.pyplot as plt

plt.scatter(x_2aproximacion,y_2aproximacion)
plt.xlabel("x_2aproximacion")
plt.ylabel("y_2aproximacion")
plt.title("Gráfica")
plt.show()
```




```
x_values_2aproximacion = np.linspace(min(x_2aproximacion), max(x_2aproximacion), 500)
y_values_2aproximacion = [lagrange_polynomial(x, x_2aproximacion, y_2aproximacion) for x in x
```

```
plt.plot(x_values_2aproximacion, y_values_2aproximacion, label="Polinomio de Lagrange")
plt.scatter(x_2aproximacion, y_2aproximacion, color="red", label="Puntos dados")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.title("Polinomio de Lagrange que pasa por los puntos dados")
plt.legend()
plt.grid()
plt.show()
```

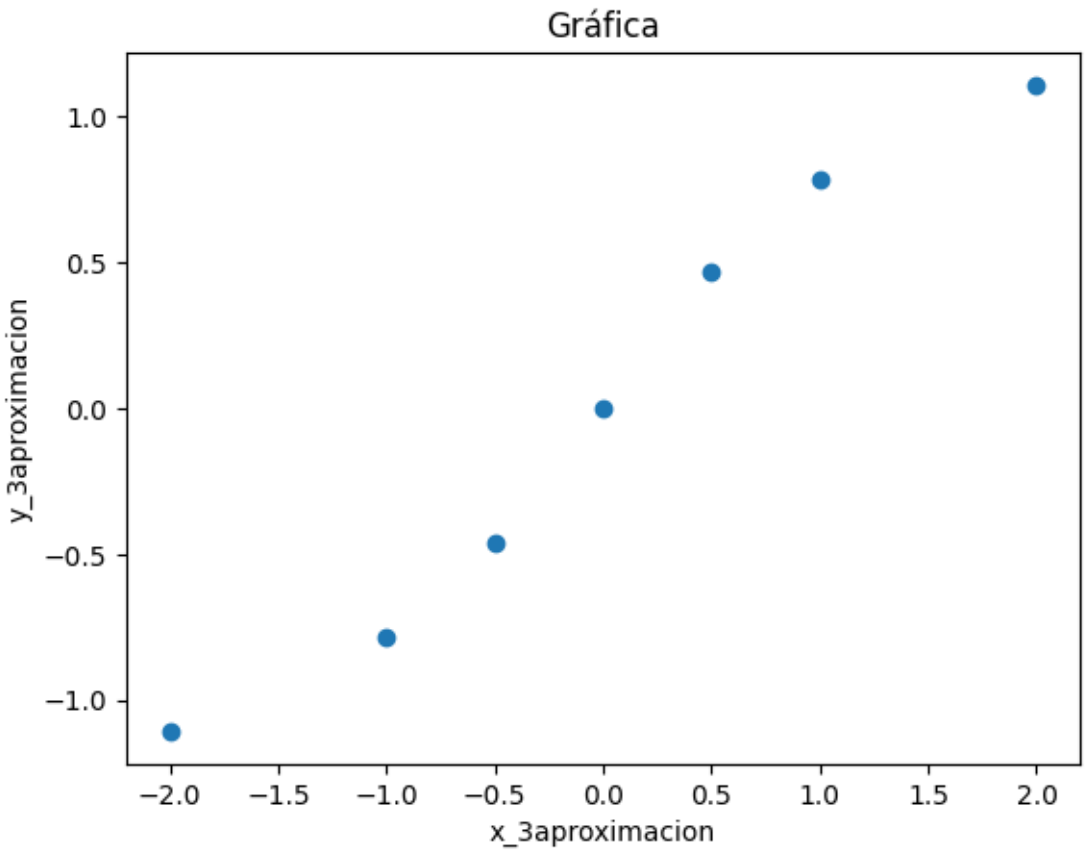


Tercera aproximación

Puntos: $(0,0)$, $(1,0.785)$, $(-1,-0.785)$, $(0.5,0.464)$, $(-0.5,-0.464)$, $(2,1.107)$, $(-2,-1.107)$

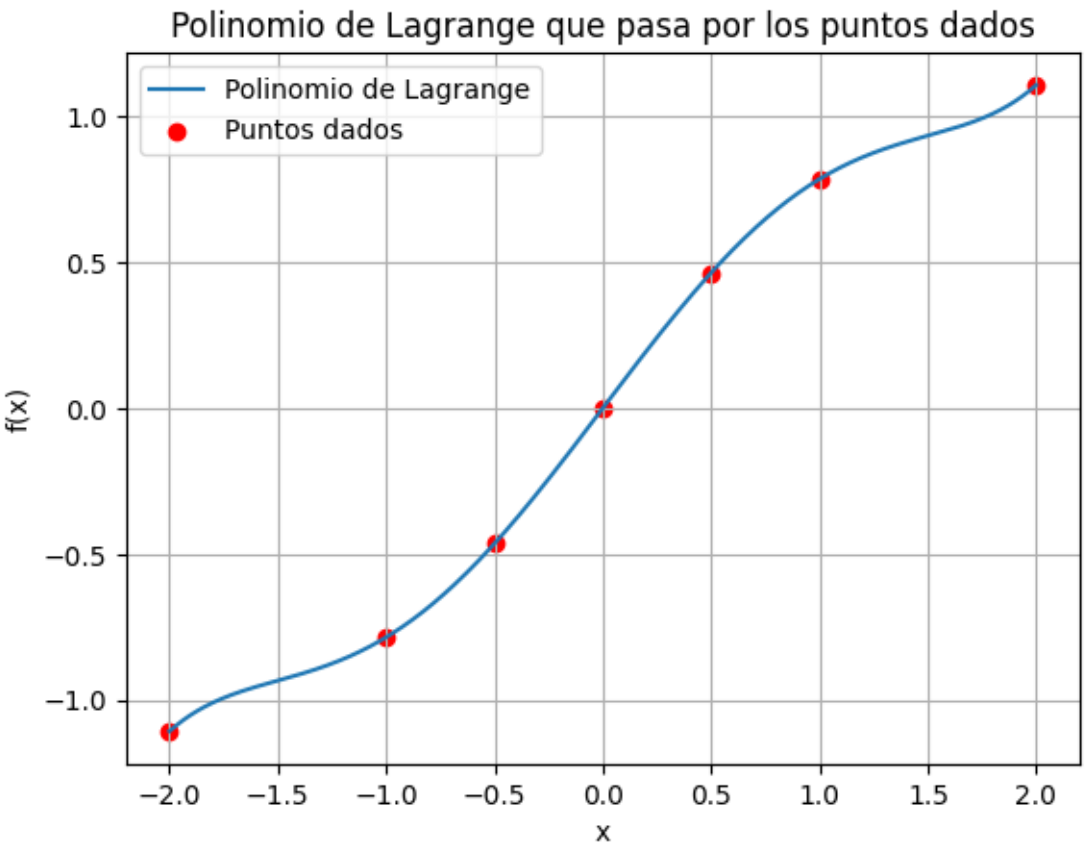
```
x_3aproximacion = [0, 1, -1, 0.5, -0.5, 2, -2]
y_3aproximacion = [0, 0.785, -0.785, 0.464, -0.464, 1.107, -1.107]
```

```
import matplotlib.pyplot as plt

plt.scatter(x_3aproximacion,y_3aproximacion)
plt.xlabel("x_3aproximacion")
plt.ylabel("y_3aproximacion")
plt.title("Gráfica")
plt.show()
```




```
x_values_3aproximacion = np.linspace(min(x_3aproximacion), max(x_3aproximacion), 500)
y_values_3aproximacion = [lagrange_polynomial(x, x_3aproximacion, y_3aproximacion) for x in x
```

```
plt.plot(x_values_3aproximacion, y_values_3aproximacion, label="Polinomio de Lagrange")
plt.scatter(x_3aproximacion, y_3aproximacion, color="red", label="Puntos dados")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.title("Polinomio de Lagrange que pasa por los puntos dados")
plt.legend()
plt.grid()
plt.show()
```



El polinomio de **grado 4 (segunda gráfica) es suficiente**, ya que tiene un ajuste excelente en ese intervalo y captura la curva de la función con precisión.

El polinomio de grado 6 (tercera gráfica) es necesario para obtener una buena aproximación. Este polinomio minimiza los errores en los extremos, pero aún puede mostrar oscilaciones en intervalos más grandes.

Los resultados en estas gráficas muestran una **mejora significativa en comparación con la serie de Taylor**, porque el polinomio de Lagrange pasa exactamente por los puntos seleccionados y ajusta mejor la curva general.