MetodosNumericosT6

November 27, 2024

1 ESCUELA POLITÉCNICA NACIONAL

1.1 MÉTODOS NUMÉRICOS

1.1.1 TAREA 6

David Alejandro Puga Novoa - GR1CC - 27/11/2024

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

- Escriba las fórmulas de los diferentes polinomios
- Grafique las diferentes aproximaciones

Series de Taylor Empezaremos creando una función para realizar las series de Taylor:

[2]:
```
import matplotlib.pyplot as plt
import numpy as np
from typing import Callable
import sympy as sym

def taylor_approx(fcn: Callable[[float], float], x0: float, n: int) -> sym.
 ↪Symbol:
    x = sym.symbols("x")
    f = sym.sympify(fcn(x))
    taylor: sym.Symbol = 0
    for i in range(n + 1):
        term = f.diff(x, i).subs(x, x0) / sym.factorial(i) * (x - x0) ** i
        taylor += term
    return taylor
```

Utilizaremos las series de Taylor hasta $n = 3$, se mostrará el resultado de la función "taylor_approx" utilizando como parámetros la función original, x_0 y el orden del polinomio.

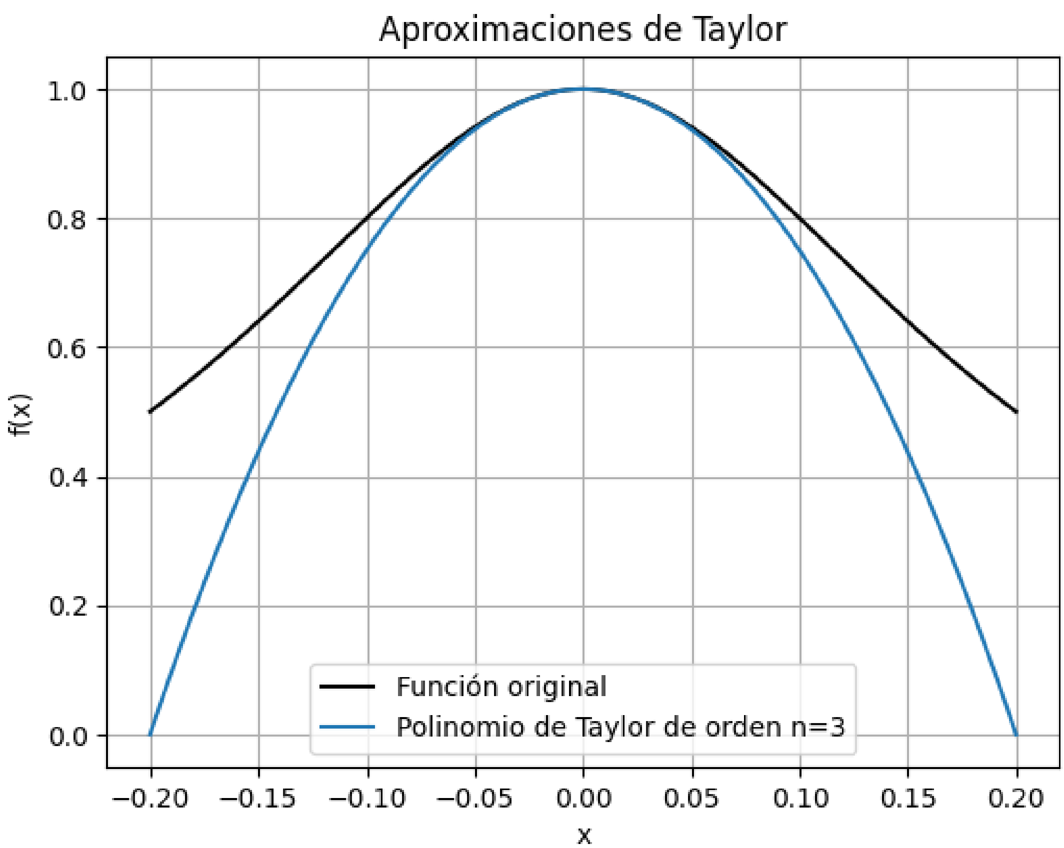
Quedando asi la función (1):

[17]:
```
func = lambda x : 1 / (25*x*x + 1)
taylor_pol = taylor_approx(func, 0, 3)
taylor_pol
```

[17]: $1 - 25x^2$

Y como necesitamos ver graficamente la función, creamos una función para este cometido:

[18]:
```
def plot_taylor_approx(fcn: Callable[[float], float], taylor_poly, n: int,␣
↪x_range: tuple):
    x = sym.symbols("x")
    x_vals = np.linspace(x_range[0], x_range[1], 1000)

    original_fcn = sym.lambdify(x, fcn(x), "numpy")
    y_vals = original_fcn(x_vals)
    plt.plot(x_vals, y_vals, label=f"Función original", color='black')

    taylor_fcn = sym.lambdify(x, taylor_poly, "numpy")

    taylor_y_vals = taylor_fcn(x_vals)
    if np.isscalar(taylor_y_vals):
        taylor_y_vals = np.full_like(x_vals, taylor_y_vals)

    plt.plot(x_vals, taylor_y_vals, label=f"Polinomio de Taylor de orden n={n}")

    plt.xlabel("x")
    plt.ylabel("f(x)")
    plt.title("Aproximaciones de Taylor")
    plt.legend()
    plt.grid(True)
    plt.show()

plot_taylor_approx(func, taylor_pol, 3, (-0.2, 0.2))
```



Ahora para la función (2) con lambda:

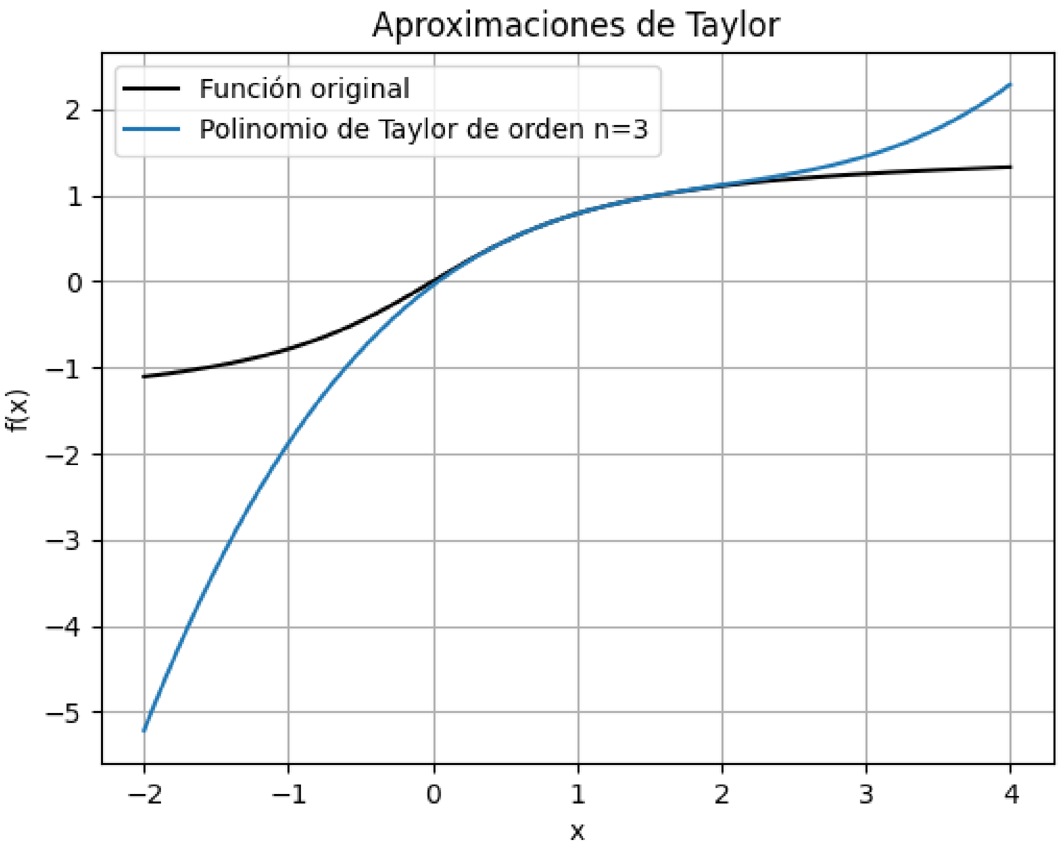
[20]:
```
func2 = lambda x : sym.atan(x)
taylor_pol = taylor_approx(func2, 1, 3)
taylor_pol
```

[20]:

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

Y su función graficamente:

[22]:
```
plot_taylor_approx(func2, taylor_pol, 3, (-2, 4))
```



Polinomio de Lagrange Empezaremos creando una función para realizar los polinomios de lagrange:

[24]:
```
from scipy.interpolate import lagrange
```

Es necesario más de un punto si queremos que funcione por ello tomaremos 3 puntos incluyendo el ya asignado como dato.

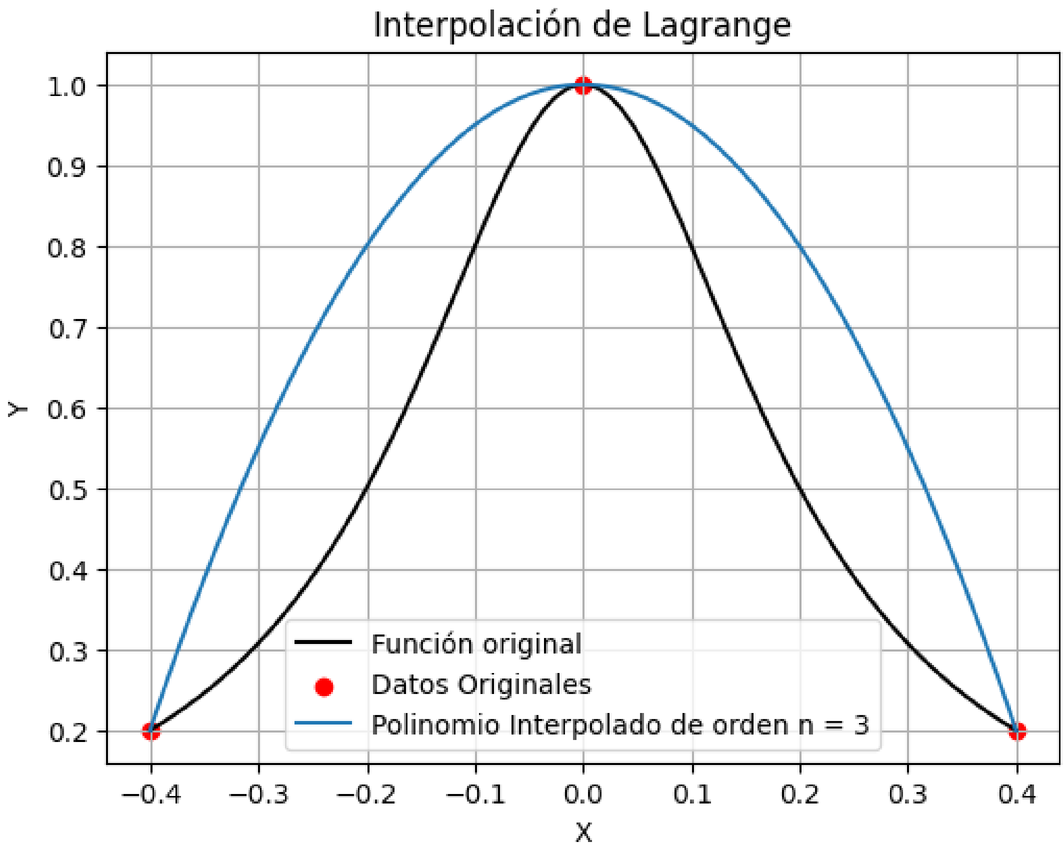
Calcula el polinomio de lagrange para la primera función (Dos puntos extra agregados para que se asemeje a la función original)

[39]:
```
X = [-0.4, 0, 0.4]
Y = [0.2, 1, 0.2]

polynomial = lagrange(X, Y)
print(polynomial)
```

```
    2
-5 x + 1
```

Ahora seria crear la función para graficar la función original y la función resultante del polinomio de lagangre:

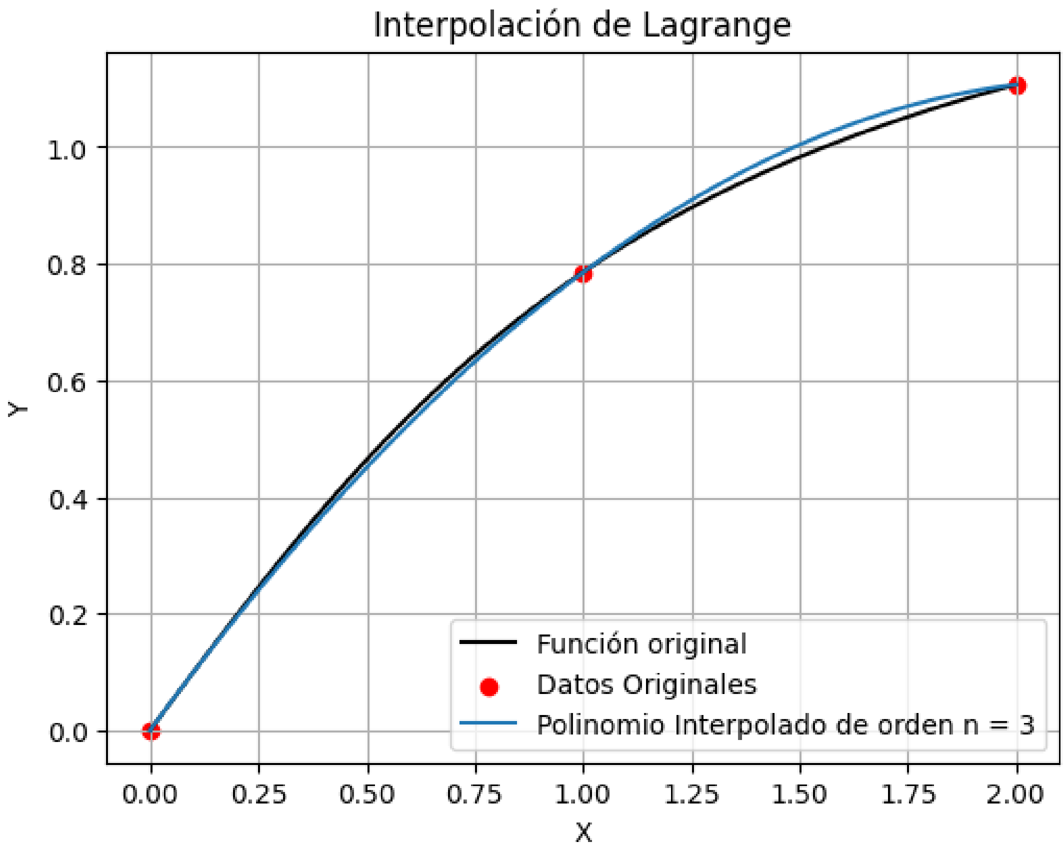
[42]:
```python
def plot_lagrange(fcn : Callable[[float], float], pol, n: int, x_range: tuple,␣
↪X, Y):
    x = sym.symbols("x")
    x_vals = np.linspace(x_range[0], x_range[1], 1000)

    original_fcn = sym.lambdify(x, fcn(x), "numpy")
    y_vals = original_fcn(x_vals)
    plt.plot(x_vals, y_vals, label=f"Función original", color='black')

    x_values = np.linspace(x_range[0], x_range[1], 1000)
    y_values = polynomial(x_values)

    plt.scatter(X, Y, color='red', label='Datos Originales')
    plt.plot(x_values, y_values, label=f'Polinomio Interpolado de orden n =␣
↪{n}')

    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title('Interpolación de Lagrange')
    plt.legend()
    plt.grid(True)
    plt.show

plot_lagrange(func, polynomial, 3, (-0.4, 0.4), X, Y)
```



Y ahora para la función (2) representamos algunos valores o pares ordenados de la propia función original para usar el polinomio de lagrange:

[48]:
```
X = [0, 1, 2]
Y = [0, 0.785, 1.107]

polynomial = lagrange(X, Y)
print(polynomial)

plot_lagrange(func2, polynomial, 3, (0, 2), X, Y)
```

```
         2
-0.2315 x + 1.017 x
```


Interpolación de Lagrange
1.0
0.8
0.6
0.4
0.2
0.0
Y
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
X
Función original
Datos Originales
Polinomio Interpolado de orden n = 3