Grado en Ingeniería Matemática

Simulación Numérica

Taller 2: Interpolación

Integrante:
López Rodríguez, Claudia

10 de febrero de 2025

Resumen

El taller consiste en implementar en Python tres métodos de interpolación: Newton, Lagrange y Splines cúbicos, utilizando los datos de los documentos `Datos Taller 2.txt` y `Datos Taller 2_B.txt`. Para cada método, se debe calcular el polinomio de interpolación y graficarlo junto con los datos experimentales en el rango [0,6].

Índice

1. Introducción.

El presente taller tiene como objetivo implementar diferentes métodos de interpolación en Python a partir de un conjunto de datos dados. A continuación, se presentan los enunciados de las tareas a realizar:

1. Cree una función en Python que a partir de un conjunto de $n+1$ datos, calcule el polinomio de interpolación de Newton de grado n. Grafique el polinomio de interpolación junto con los datos experimentales aportados en el archivo `"Datos Taller 2.txt"` en el rango [0, 6].

2. Cree una función en Python que a partir de un conjunto de $n+1$ datos, calcule el polinomio de interpolación de Lagrange de grado n. Grafique el polinomio de interpolación junto con los datos experimentales aportados en el archivo `"Datos Taller 2.txt"` en el rango [0, 6].

3. Cree una función en Python que a partir de un conjunto de $n+1$ datos, calcule el polinomio de interpolación a través de splines cúbicos. Grafique el polinomio de interpolación junto con los datos experimentales aportados en el archivo `"Datos Taller 2.txt"` en el rango [0, 6].

2. Métodos y modelos matemáticos.

En este taller se emplean tres métodos fundamentales de interpolación numérica: el polinomio de interpolación de Newton, el polinomio de interpolación de Lagrange y los splines cúbicos. A continuación, se explica brevemente cada uno de ellos.

2.1. Interpolación de Newton

La interpolación de Newton se basa en la forma dividida de las diferencias finitas y se expresa como:

$$P_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots + f[x_0, \ldots, x_n](x - x_0)\ldots(x - x_{n-1}) \quad (1)$$

donde $f[x_0, x_1, \ldots, x_k]$ son las diferencias divididas de Newton, calculadas recursivamente como:

$$f[x_i, x_{i+1}, \ldots, x_{i+k}] = \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i} \quad (2)$$

Este método permite construir un polinomio de interpolación eficiente y bien condicionado para conjuntos de datos dados.

2.2. Interpolación de Lagrange

El polinomio de interpolación de Lagrange tiene la forma:

$$P_n(x) = \sum_{i=0}^{n} L_i(x) f(x_i) \quad (3)$$

donde las bases de Lagrange $L_i(x)$ se definen como:

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \quad (4)$$

Este método evita el cálculo de diferencias divididas, pero puede ser menos eficiente para conjuntos de datos grandes debido a la necesidad de evaluar productos en cada paso.

2.3. Interpolación por Splines Cúbicos

La interpolación por splines cúbicos consiste en construir una serie de polinomios cúbicos $S_i(x)$ que interpolan los datos en intervalos entre los puntos x_i. Cada spline cúbico tiene la forma:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3, \quad x_i \leq x \leq x_{i+1} \quad (5)$$

Para garantizar suavidad, se imponen condiciones de continuidad en la función, su primera y segunda derivada:

$$S_i(x_i) = f(x_i), \quad S_i(x_{i+1}) = f(x_{i+1}) \quad (6)$$

$$S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}), \quad S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}) \quad (7)$$

Además, se pueden utilizar condiciones adicionales como la condición natural $S''(x_0) = S''(x_n) = 0$ para determinar completamente los coeficientes de los polinomios cúbicos.

Estos tres métodos permiten obtener aproximaciones polinomiales a partir de los datos experimentales proporcionados, asegurando una representación precisa en el intervalo especificado.

3. Resultados

3.1. Interpolación de Newton

3.1.1. Observaciones

- La interpolación de Newton sigue los datos experimentales con precisión, pero presenta oscilaciones más marcadas en ciertos intervalos.
- En conjuntos de datos con cambios bruscos, el polinomio de Newton puede generar fluctuaciones indeseadas debido a su estructura basada en diferencias divididas.
- A medida que aumenta el número de puntos, la función interpolante se vuelve más sensible, lo que puede causar efectos de sobreajuste.

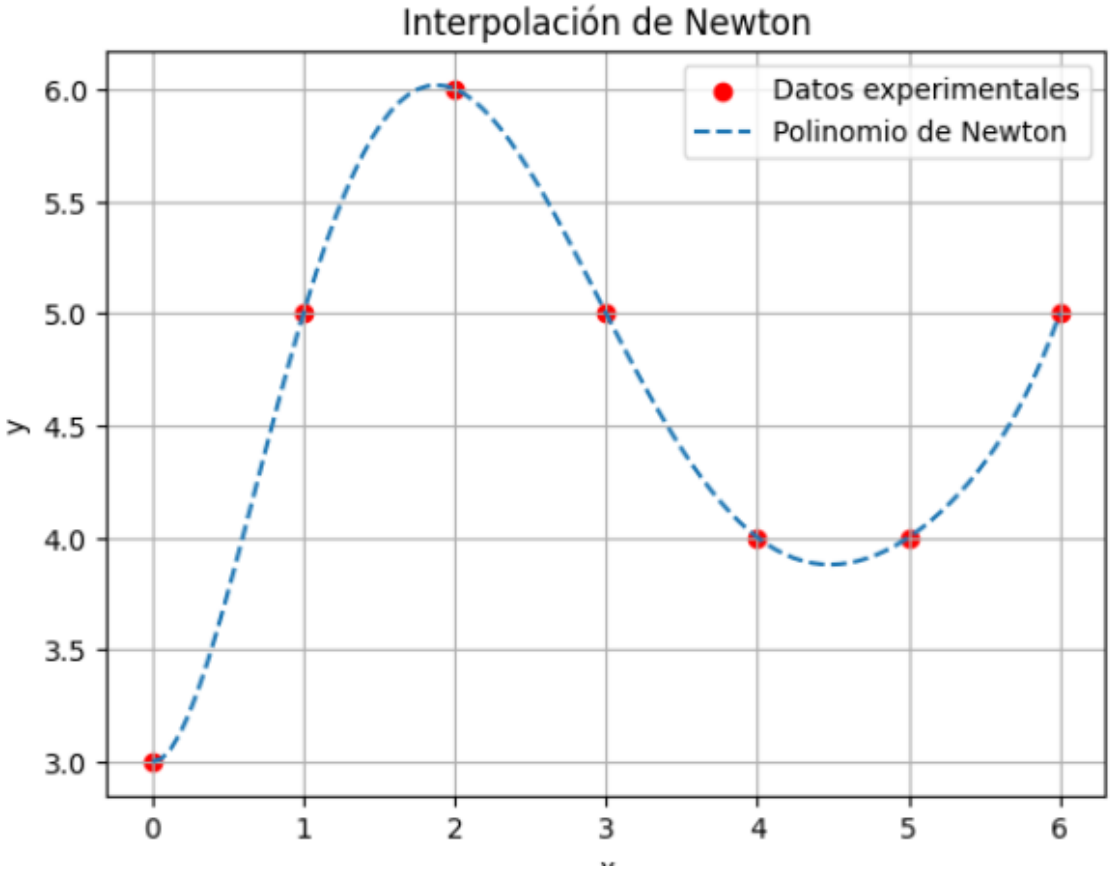

Figura 1: Interpolación de Newton.

3.2. Interpolación de Lagrange

3.2.1. Observaciones

- La interpolación de Lagrange presenta un comportamiento similar al de Newton, pero con una tendencia a oscilar aún más en los extremos (fenómeno de Runge).
- Como en Newton, cuando los datos tienen grandes variaciones, el polinomio de Lagrange responde con oscilaciones que pueden generar una mala representación en ciertas regiones.
- En los gráficos, se observa que la curva interpolante pasa exactamente por los puntos experimentales, pero con fluctuaciones en intervalos amplios.

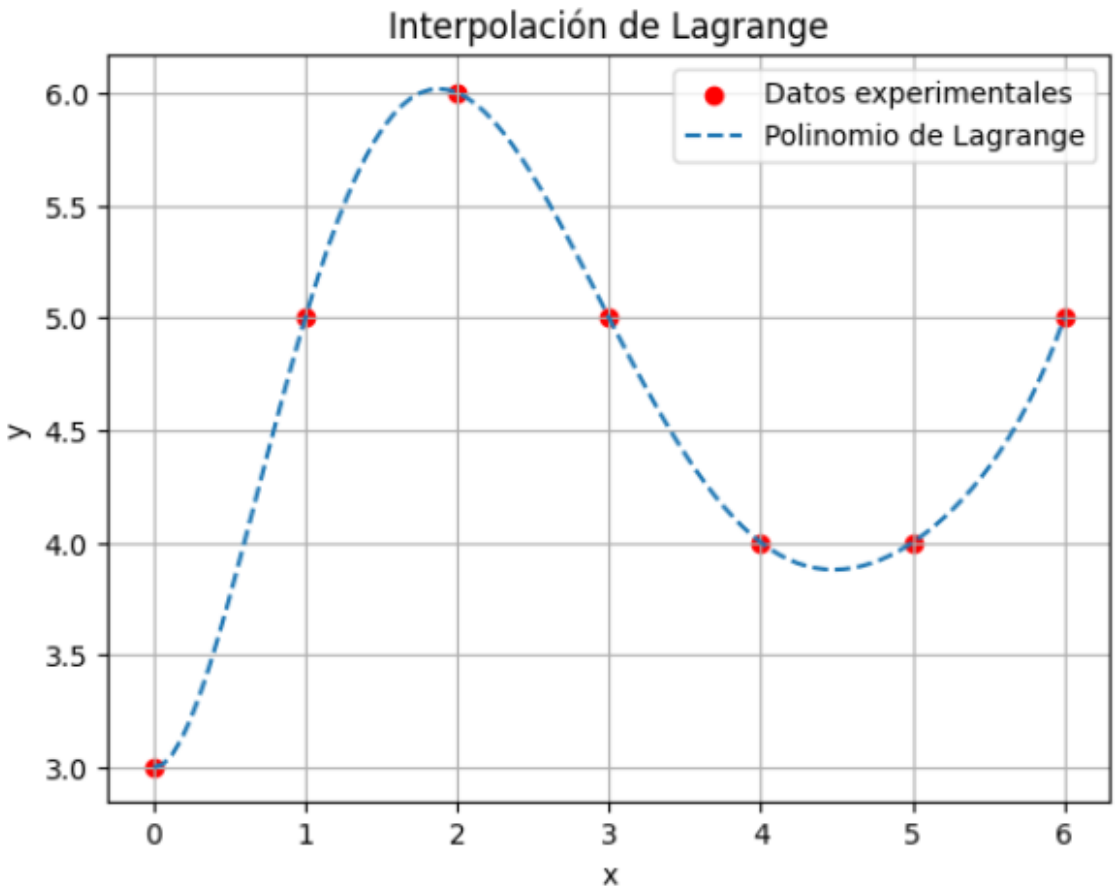

Figura 2: Interpolación de Lagrange.

3.3. Interpolación con Splines Cúbicos

3.3.1. Observaciones

- A diferencia de Newton y Lagrange, los Splines Cúbicos generan una curva suave que sigue bien la tendencia de los datos experimentales sin oscilaciones excesivas.
- En el primer gráfico con datos ruidosos, se observan algunas variaciones en la interpolación, pero la curva es más estable y menos propensa a fluctuaciones indeseadas.
- En el segundo gráfico, con datos más ordenados, la interpolación es casi perfecta, con transiciones suaves entre los puntos.

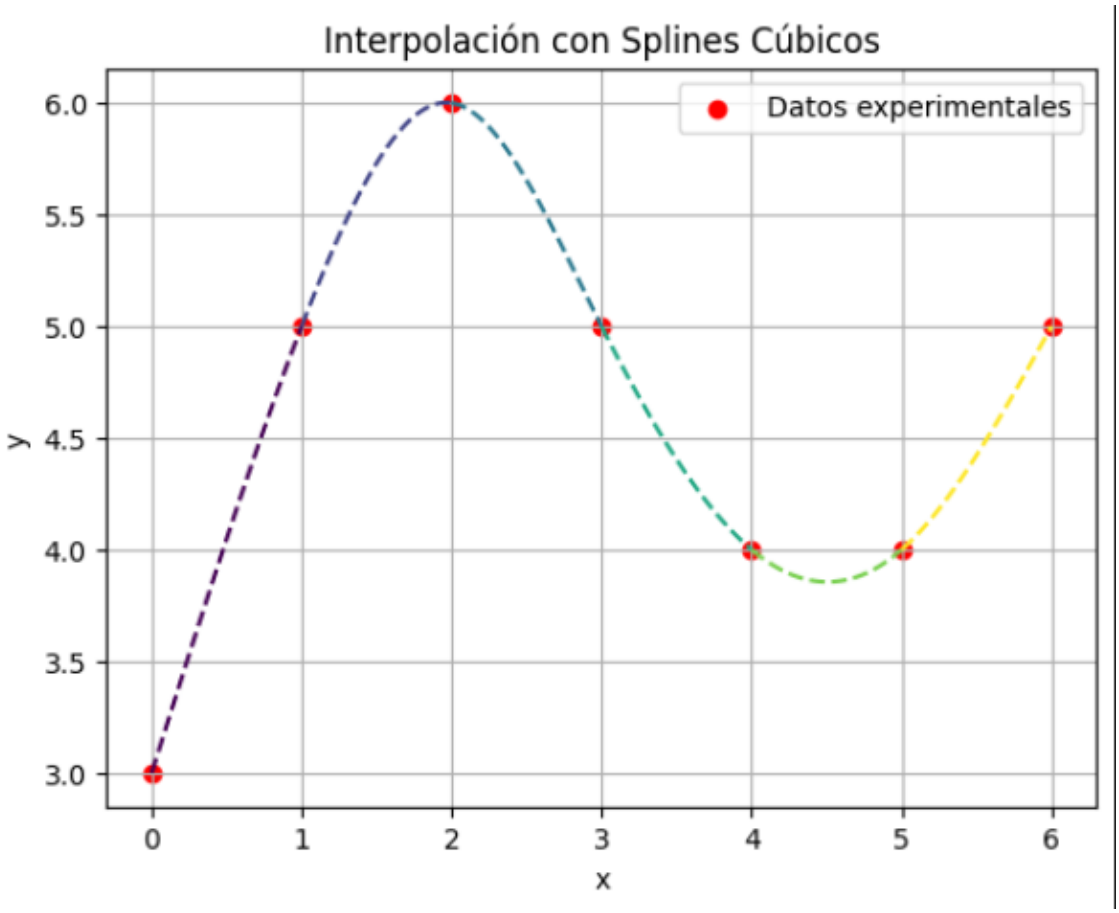

Figura 3: Interpolación con Splines Cúbicos.

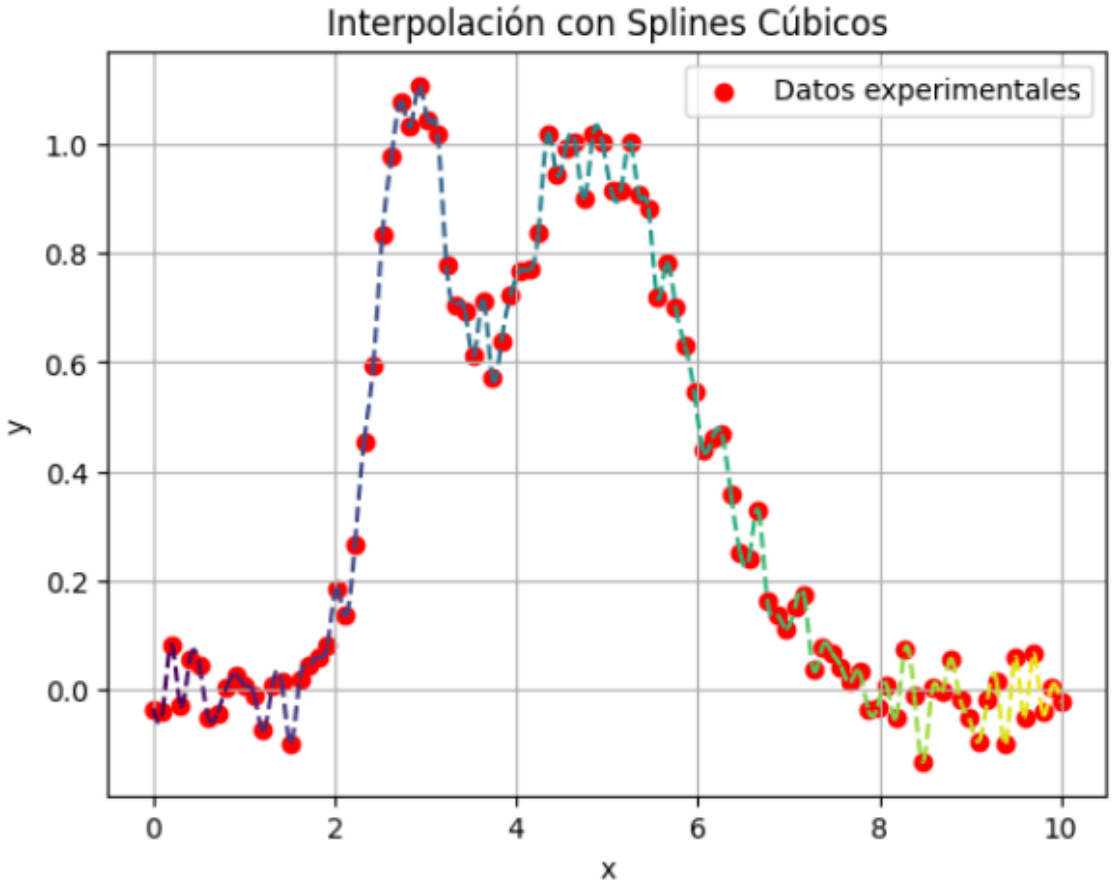

Figura 4: Interpolación con Splines Cúbicos (datos 2B).

4. Discusión.

La interpolación es una herramienta fundamental en el análisis de datos y modelado matemático. En este estudio, se han comparado tres métodos distintos: Newton, Lagrange y Splines Cúbicos, cada uno con características particulares en cuanto a precisión, estabilidad y comportamiento ante conjuntos de datos con variaciones.

Los métodos polinómicos de **Newton y Lagrange** presentan un ajuste exacto a los datos experimentales, pero a costa de posibles oscilaciones no deseadas, especialmente en los extremos del intervalo (fenómeno de Runge). En particular, el método de Lagrange tiende a generar fluctuaciones más marcadas, ya que la interpolación polinómica de grado elevado puede volverse inestable. Aunque Newton comparte esta tendencia, su construcción mediante diferencias divididas le otorga una ventaja computacional al permitir la adición de nuevos puntos sin recalcular toda la interpolación. Sin embargo, ambos métodos sufren de falta de suavidad en conjuntos de datos grandes o con variaciones abruptas.

Por otro lado, los **Splines Cúbicos** muestran una clara ventaja en términos de estabilidad y suavidad. En los gráficos analizados, este método logra una interpolación más precisa sin generar oscilaciones excesivas, incluso en presencia de datos con cambios bruscos. A diferencia de los polinomios de alto grado, los Splines

dividen el intervalo en secciones más manejables, lo que reduce la sensibilidad a los extremos y mejora la representación de los datos experimentales. Además, su capacidad para garantizar continuidad en la primera y segunda derivada proporciona una curva más realista y suave.

En conclusión, la elección del método de interpolación depende de la naturaleza de los datos y el objetivo del análisis. Si se requiere una interpolación exacta sin considerar estabilidad, Newton o Lagrange pueden ser opciones viables para conjuntos pequeños. Sin embargo, si la prioridad es suavidad y estabilidad en la interpolación, los **Splines Cúbicos son la mejor alternativa**, especialmente cuando los datos presentan variaciones significativas.

5. Conclusiones.

A partir del análisis realizado sobre los métodos de interpolación de Newton, Lagrange y Splines Cúbicos, se puede concluir que la elección del método adecuado depende de la naturaleza de los datos y del propósito del análisis.

Los métodos polinómicos, como Newton y Lagrange, garantizan una interpolación exacta pasando por todos los puntos experimentales, pero pueden generar oscilaciones significativas, especialmente cuando se trabaja con conjuntos de datos grandes o con variaciones bruscas. En particular, el método de Lagrange es más susceptible al fenómeno de Runge, mientras que Newton ofrece una ventaja computacional al permitir la adición de nuevos puntos sin recalcular toda la interpolación.

Por otro lado, los Splines Cúbicos han demostrado ser una alternativa más estable y eficiente, proporcionando curvas suaves y precisas sin oscilaciones excesivas. Su capacidad para dividir el dominio en segmentos cúbicos garantiza una mejor representación de los datos experimentales y una mayor estabilidad en la interpolación.

En general, si el objetivo es obtener una representación suave y precisa de los datos sin grandes fluctuaciones, los Splines Cúbicos son la opción más recomendable. Sin embargo, para conjuntos de datos pequeños donde la exactitud en los puntos experimentales es primordial, los métodos polinómicos pueden ser adecuados. La elección final dependerá del equilibrio entre precisión, estabilidad y eficiencia computacional en cada caso específico.

6. Referencias

Referencias

[1] Facultad de Ingeniería de la UNMdP. *Diferencias Divididas de Newton*. Disponible en: https://www3.fi.mdp.edu.ar/metodos/apuntes/diferencias%20divididas.pdf

[2] UNAM, Facultad de Ingeniería. *Interpolación de Lagrange*. Disponible en: https://www.ingenieria.unam.mx/pinilla/PE105117/pdfs/tema4/4-1_lagrange.pdf

[3] Universidad de Sevilla. *Interpolación con Splines Cúbicos*. Disponible en: https://biblus.us.es/bibing/proyectos/abreproy/3989/fichero/MEMORIA%252F13_ANEXO7.pdf

7. Anexos.

```python
def diferencias_divididas(x, y):
    """ Calcula la tabla de diferencias divididas de Newton """
    n = len(x)
    coef = np.array(y, dtype=float)

    for j in range(1, n):
        for i in range(n - 1, j - 1, -1):
            coef[i] = (coef[i] - coef[i - 1]) / (x[i] - x[i - j])

    return coef

def polinomio_newton(x_eval, x, coef):
    """ Evalúa el polinomio de Newton en los puntos dados """
    n = len(coef)
    resultado = coef[-1]

    for i in range(n - 2, -1, -1):
        resultado = resultado * (x_eval - x[i]) + coef[i]

    return resultado

# Cargar los datos del archivo
datos = np.loadtxt("Datos_Taller_2.txt")
x_datos, y_datos = datos[:, 0], datos[:, 1]

# Calcular coeficientes del polinomio de Newton
coeficientes = diferencias_divididas(x_datos, y_datos)

# Evaluar el polinomio en un rango de puntos para graficar
x_graf = np.linspace(0, 6, 100)
y_graf = [polinomio_newton(x, x_datos, coeficientes) for x in x_graf]

# Graficar
plt.scatter(x_datos, y_datos, color='red', label='Datos experimentales')
plt.plot(x_graf, y_graf, label='Polinomio de Newton', linestyle='--')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid()
plt.title('Interpolación de Newton')
plt.show()
```

Figura 5: Interpolación de Newton.

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

def lagrange_basis(x, x_vals, k):
    """ Calcula el k-ésimo polinomio base de Lagrange """
    L_k = 1
    for i in range(len(x_vals)):
        if i != k:
            L_k *= (x - x_vals[i]) / (x_vals[k] - x_vals[i])
    return L_k

def polinomio_lagrange(x_eval, x_vals, y_vals):
    """ Evalúa el polinomio de interpolación de Lagrange """
    resultado = 0
    for k in range(len(x_vals)):
        resultado += y_vals[k] * lagrange_basis(x_eval, x_vals, k)
    return resultado

# Cargar los datos del archivo
datos = np.loadtxt("Datos_Taller_2.txt")
x_datos, y_datos = datos[:, 0], datos[:, 1]

# Evaluar el polinomio en un rango de puntos para graficar
x_graf = np.linspace(0, 6, 100)
y_graf = [polinomio_lagrange(x, x_datos, y_datos) for x in x_graf]

# Graficar
plt.scatter(x_datos, y_datos, color='red', label='Datos experimentales')
plt.plot(x_graf, y_graf, label='Polinomio de Lagrange', linestyle='--')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid()
plt.title('Interpolación de Lagrange')
plt.show()
```

Figura 6: Intepolación de Lagrange.

```python
def tridiagonal_matrix_algorithm(a, b, c, d):
    """ Resuelve el sistema tridiagonal Ax = d """
    n = len(d)
    c_prime = np.zeros(n-1)
    d_prime = np.zeros(n)
    x = np.zeros(n)

    c_prime[0] = c[0] / b[0]
    d_prime[0] = d[0] / b[0]

    for i in range(1, n-1):
        temp = b[i] - a[i-1] * c_prime[i-1]
        c_prime[i] = c[i] / temp
        d_prime[i] = (d[i] - a[i-1] * d_prime[i-1]) / temp

    d_prime[-1] = (d[-1] - a[-1] * d_prime[-2]) / (b[-1] - a[-1] * c_prime[-2])
    x[-1] = d_prime[-1]

    for i in range(n-2, -1, -1):
        x[i] = d_prime[i] - c_prime[i] * x[i+1]

    return x
```

Figura 7: Interpolación con Splines Cúbicos.

```python
def splines_cubicos(x_vals, y_vals):
    """ Calcula los coeficientes de los splines cúbicos """
    n = len(x_vals) - 1
    h = np.diff(x_vals)
    alpha = np.zeros(n)

    for i in range(1, n):
        alpha[i] = (3 / h[i]) * (y_vals[i+1] - y_vals[i]) - (3 / h[i-1]) * (y_vals[i] - y_vals[i-1])

    a = h[:-1]
    b = 2 * (h[:-1] + h[1:])
    c = h[1:]
    d = alpha[1:]

    c_vals = np.zeros(n+1)
    c_vals[1:-1] = tridiagonal_matrix_algorithm(a, b, c, d)

    b_vals = np.zeros(n)
    d_vals = np.zeros(n)

    for i in range(n):
        b_vals[i] = (y_vals[i+1] - y_vals[i]) / h[i] - h[i] * (c_vals[i+1] + 2 * c_vals[i]) / 3
        d_vals[i] = (c_vals[i+1] - c_vals[i]) / (3 * h[i])

    return x_vals, y_vals, b_vals, c_vals, d_vals
```

Figura 8: Interpolación con Splines Cúbicos.

```python
def evaluar_spline(x, x_vals, y_vals, b_vals, c_vals, d_vals):
    """ Evalúa el spline cúbico en un punto x """
    for i in range(len(x_vals) - 1):
        if x_vals[i] <= x <= x_vals[i+1]:
            dx = x - x_vals[i]
            return y_vals[i] + b_vals[i] * dx + c_vals[i] * dx**2 + d_vals[i] * dx**3
    return None

# Cargar los datos del archivo
datos = np.loadtxt("Datos_Taller_2.txt")
x_datos, y_datos = datos[:, 0], datos[:, 1]

# Calcular coeficientes de los splines cúbicos
x_vals, y_vals, b_vals, c_vals, d_vals = splines_cubicos(x_datos, y_datos)

# Graficar cada tramo de spline con un color distinto
colors = plt.cm.viridis(np.linspace(0, 1, len(x_vals) - 1))
plt.scatter(x_datos, y_datos, color='red', label='Datos experimentales')

for i in range(len(x_vals) - 1):
    x_graf = np.linspace(x_vals[i], x_vals[i+1], 50)
    y_graf = [evaluar_spline(x, x_vals, y_vals, b_vals, c_vals, d_vals) for x in x_graf]
    plt.plot(x_graf, y_graf, color=colors[i], linestyle='--')

plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid()
plt.title('Interpolación con Splines Cúbicos')
plt.show()
```

Figura 9: Interpolación con Splines Cúbicos.