Escuela Politécnica Nacional

Nombre: Kevin Eduardo Garcia Rodriguez

Tema: [Taller 4] Splines Cubicos

Repositorio GIT: https://github.com/Nattyrd/Metodos-Numericos-2025B

In [24]:
```
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
```

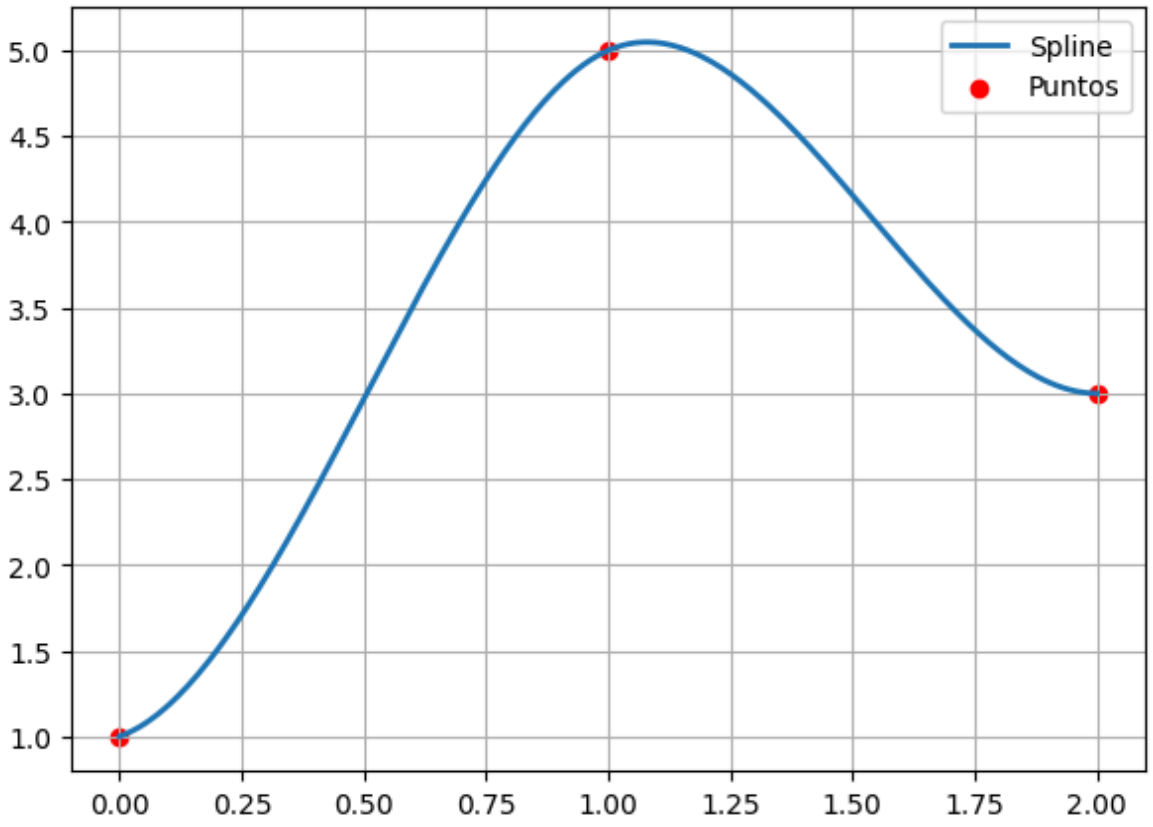
In [30]:
```
class CubicSpline:
    def __init__(self, x, y, spline_type="natural", B0=0, B1=0, Bn=0):
        self.x = np.array(x)
        self.y = np.array(y)
        self.n = len(x) - 1
        self.h = np.diff(self.x)

        self.spline_type = spline_type
        self.B0 = B0
        self.B1 = B1
        self.Bn = Bn

        self._compute_spline()

    def _compute_spline(self):
        n = self.n
        h = self.h
        x = self.x
        y = self.y

        A = np.zeros((n+1, n+1))
        B = np.zeros(n+1)

        # Condición de contorno
        if self.spline_type == "natural":
            A[0, 0] = 1
            A[n, n] = 1
        else:
            A[0, 0] = 2*h[0]
            A[0, 1] = h[0]
            B[0] = 6*((y[1] - y[0])/h[0] - self.B0)

            A[n, n-1] = h[n-1]
            A[n, n] = 2*h[n-1]
            B[n] = 6*(self.Bn - (y[n] - y[n-1])/h[n-1])

        # Sistema interno
        for i in range(1, n):
            A[i, i-1] = h[i-1]
            A[i, i] = 2*(h[i-1] + h[i])
            A[i, i+1] = h[i]
            B[i] = 6*((y[i+1] - y[i])/h[i] - (y[i] - y[i-1])/h[i-1])
```

```
        # Resolver sistema
        self.M = np.linalg.solve(A, B)

        # Coeficientes
        self.a = y[:-1]
        self.b = (y[1:] - y[:-1]) / h - h * (2*self.M[:-1] + self.M[1:]) / 6
        self.c = self.M[:-1] / 2
        self.d = (self.M[1:] - self.M[:-1]) / (6*h)

    def print_coefficients(self):
        print("\n Coeficientes del spline:\n")
        for i in range(self.n):
            print(f"Intervalo [{self.x[i]}, {self.x[i+1]}]:")
            print(f"a = {self.a[i]:.4f}, b = {self.b[i]:.4f}, c = {self.c[i]:.4f

    def print_functions(self):
        print("\n Funciones spline por tramo:\n")
        x = sp.symbols("x")
        for i in range(self.n):
            func = self.a[i] + self.b[i]*(x - self.x[i]) + self.c[i]*(x - self.x
            print(f"S{i}(x) = {sp.expand(func)}\n")

    def evaluate(self, value):
        value = np.array(value)
        result = np.zeros_like(value, dtype=float)

        for i in range(self.n):
            mask = (value >= self.x[i]) & (value <= self.x[i+1])
            dx = value[mask] - self.x[i]
            result[mask] = self.a[i] + self.b[i]*dx + self.c[i]*dx**2 + self.d[i

        return result

    def plot(self):
        x_plot = np.linspace(self.x[0], self.x[-1], 300)
        y_plot = self.evaluate(x_plot)

        plt.figure(figsize=(7, 5))
        plt.plot(x_plot, y_plot, label="Spline", linewidth=2)
        plt.scatter(self.x, self.y, color="red", label="Puntos")
        plt.legend()
        plt.grid()
        plt.show()

    def animate_B1(self, B1_min=-5, B1_max=5, frames=50):
        fig, ax = plt.subplots()
        x_plot = np.linspace(self.x[0], self.x[-1], 300)
        line, = ax.plot([], [], lw=2)
        ax.scatter(self.x, self.y, color='black')

        def update(frame):
            self.B1 = B1_min + (B1_max - B1_min) * frame / frames
            self._compute_spline()
            y_plot = self.evaluate(x_plot)
            line.set_data(x_plot, y_plot)
            ax.set_title(f"Spline condicionado - B1 = {self.B1:.2f}")
            return line,

        anim = FuncAnimation(fig, update, frames=frames, interval=100)
```

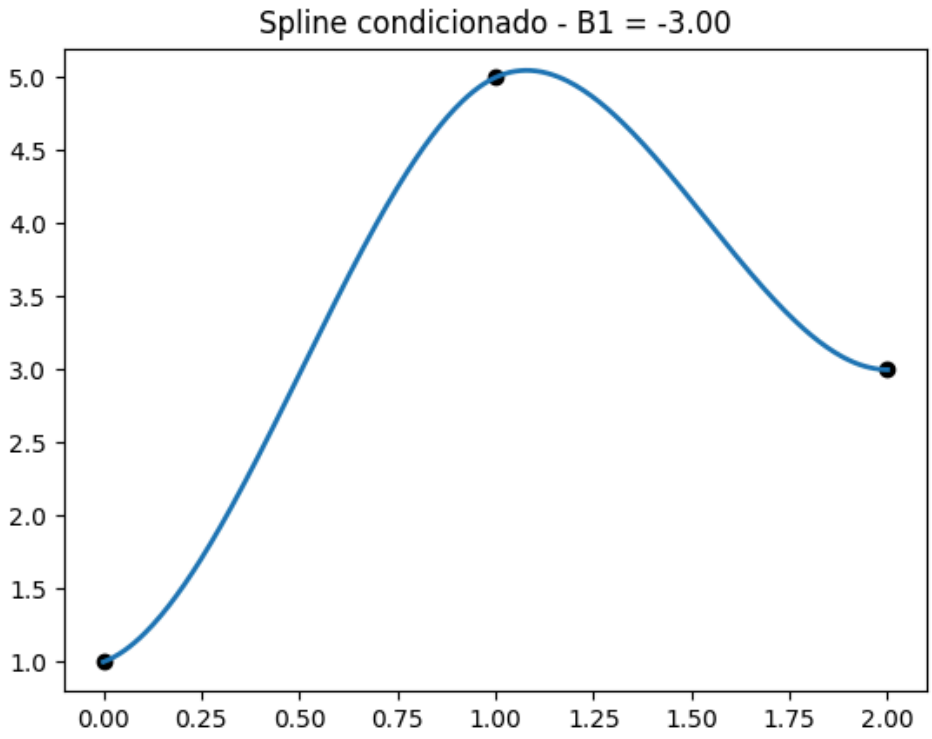
```
        plt.close(fig)
        return anim
```

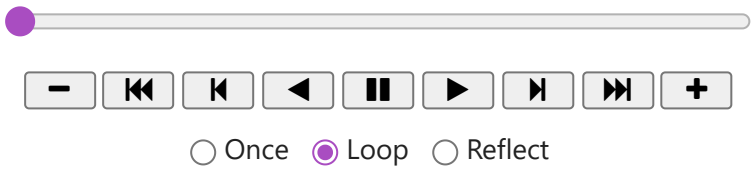
In [32]:
```
x = [0, 1, 2]
y = [1, 5, 3]

spline = CubicSpline(x, y, spline_type="condicionado", B0=1, Bn=0)

spline.print_coefficients()
spline.print_functions()
spline.plot()

anim = spline.animate_B1(-3, 3, 60)
HTML(anim.to_jshtml())
```

```
 Coeficientes del spline:

Intervalo [0, 1]:
a = 1.0000, b = 1.0000, c = 8.7500, d = -5.7500

Intervalo [1, 2]:
a = 5.0000, b = 1.2500, c = -8.5000, d = 5.2500


 Funciones spline por tramo:

S0(x) = -5.75*x**3 + 8.75*x**2 + 1.0*x + 1

S1(x) = 5.25*x**3 - 24.25*x**2 + 34.0*x - 10.0
```



Out[32]:

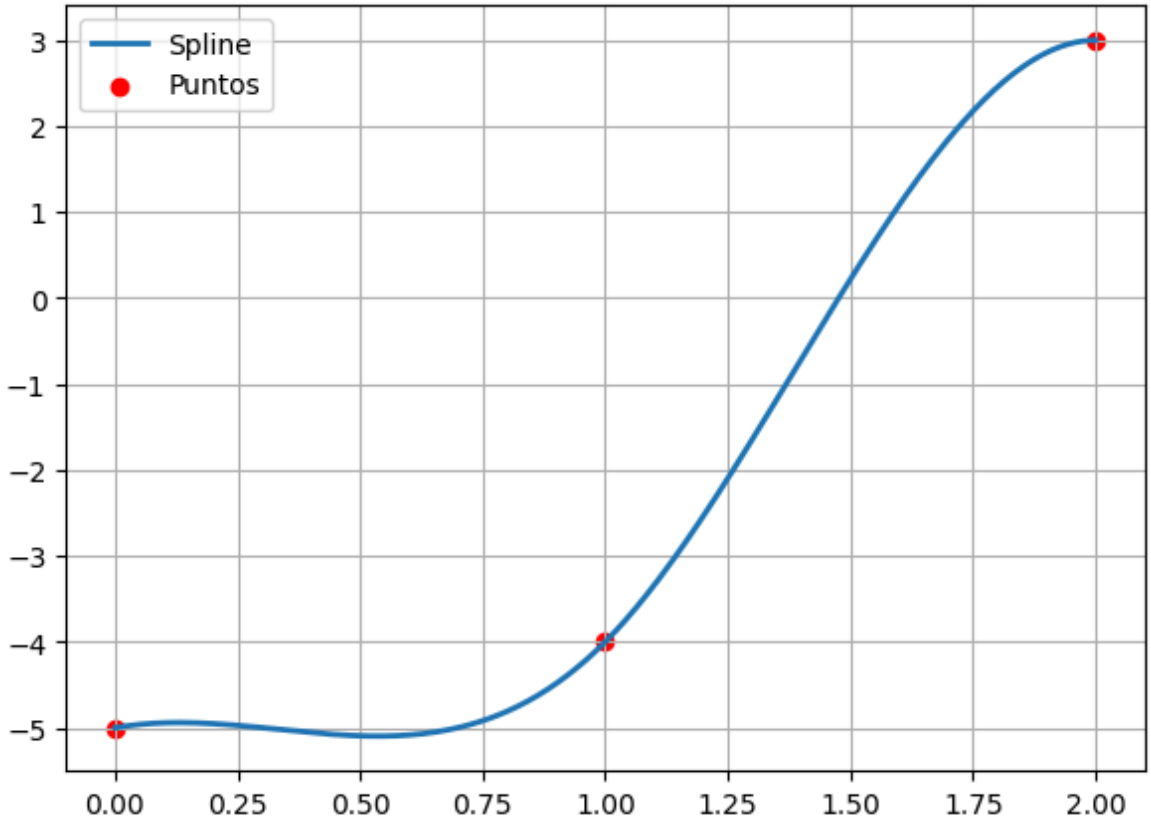




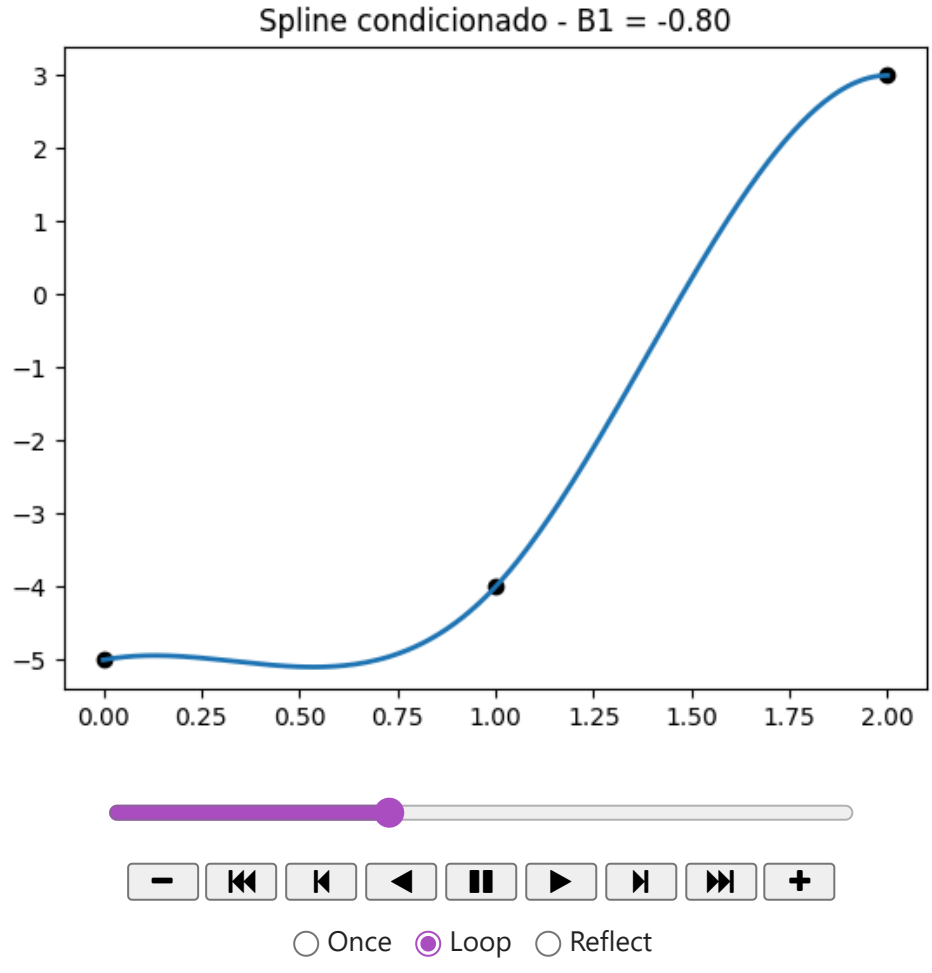
Once Loop Reflect

In [33]:
```
x1 = [0, 1, 2]
y1 = [-5, -4, 3]

spline1 = CubicSpline(x1, y1, spline_type="condicionado", B0=1, Bn=0)

spline1.print_coefficients()
spline1.print_functions()
spline1.plot()

anim = spline1.animate_B1(-3, 3, 60)
HTML(anim.to_jshtml())
```

```
 Coeficientes del spline:

Intervalo [0, 1]:
a = -5.0000, b = 1.0000, c = -4.7500, d = 4.7500

Intervalo [1, 2]:
a = -4.0000, b = 5.7500, c = 9.5000, d = -8.2500


 Funciones spline por tramo:

S0(x) = 4.75*x**3 - 4.75*x**2 + 1.0*x - 5

S1(x) = -8.25*x**3 + 34.25*x**2 - 38.0*x + 8.0
```



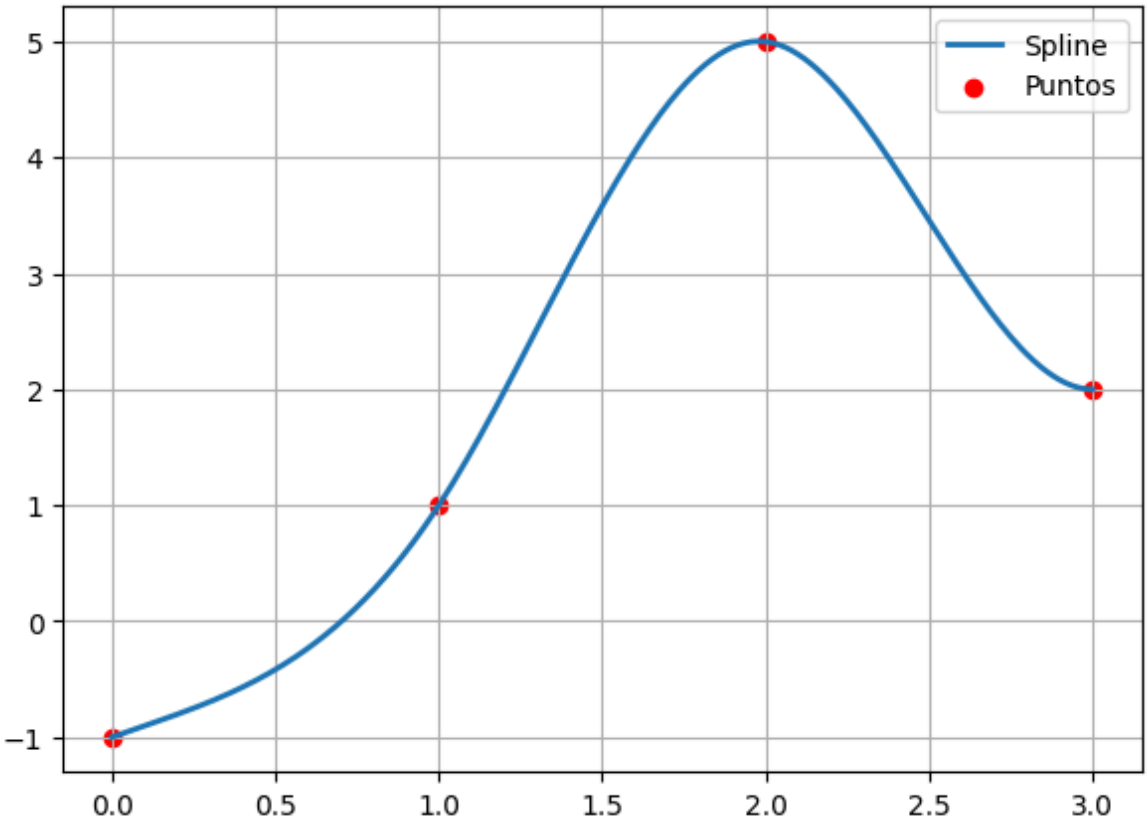
Out[33]:



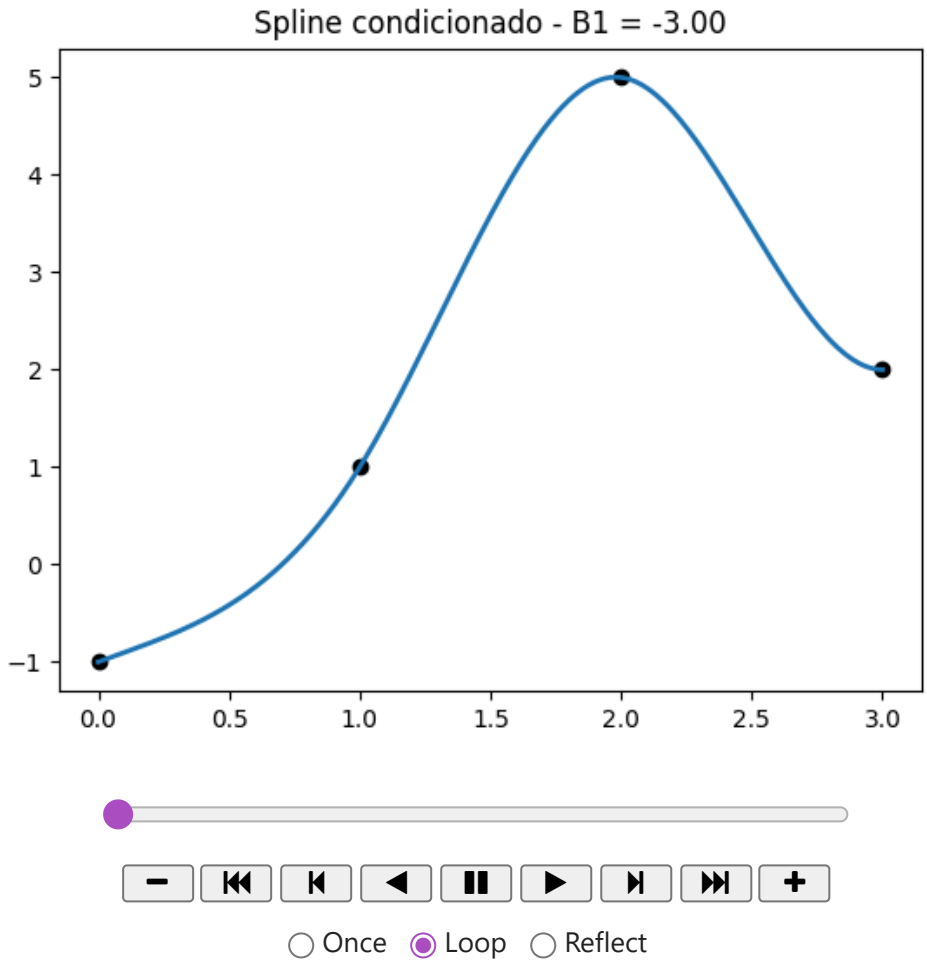
In [36]:
```
x1 = [0, 1, 2, 3]
y1 = [-1, 1, 5, 2]

spline1 = CubicSpline(x1, y1, spline_type="condicionado", B0=1, Bn=0)
```

```
spline1.print_coefficients()
spline1.print_functions()
spline1.plot()

anim = spline1.animate_B1(-3, 3, 60)
HTML(anim.to_jshtml())
```

```
 Coeficientes del spline:

Intervalo [0, 1]:
a = -1.0000, b = 1.0000, c = -0.3333, d = 1.3333

Intervalo [1, 2]:
a = 1.0000, b = 4.3333, c = 3.6667, d = -4.0000

Intervalo [2, 3]:
a = 5.0000, b = -0.3333, c = -8.3333, d = 5.6667


 Funciones spline por tramo:

S0(x) = 1.33333333333333*x**3 - 0.333333333333333*x**2 + 1.0*x - 1

S1(x) = -4.0*x**3 + 15.6666666666667*x**2 - 15.0*x + 4.33333333333333

S2(x) = 5.66666666666667*x**3 - 42.3333333333333*x**2 + 101.0*x - 73.0
```



Out[36]:



Metodo para graficar segun a,b,c,d

In [37]:

```
def graficar_splines(a, b, c, d, x_intervals, puntos_originales):
    """
    a,b,c,d deben ser listas donde cada posición corresponde a un intervalo.
    x_intervals debe ser una lista con los puntos nodales (ej: [0,1,2,3])
    puntos_originales: lista de tuplas [(x,y), ...]
    """

    # Imprimir funciones spline
    print("FUNCIONES DE SPLINE OBTENIDAS:\n")
    for i in range(len(a)):
        print(f"S{i}(x) = {a[i]:.6f} + {b[i]:.6f}(x - {x_intervals[i]}) + "
              f"{c[i]:.6f}(x - {x_intervals[i]})^2 + {d[i]:.6f}(x - {x_intervals

    # Graficar spline usando puntos densos
    X = []
    Y = []

    for i in range(len(a)):
        xs = np.linspace(x_intervals[i], x_intervals[i+1], 50)
        ys = a[i] + b[i]*(xs - x_intervals[i]) + c[i]*(xs - x_intervals[i])**2 +
        X.extend(xs)
        Y.extend(ys)

    plt.plot(X, Y, label="Spline Resultante")

    # Graficar puntos originales
```

```
    x_pts = [p[0] for p in puntos_originales]
    y_pts = [p[1] for p in puntos_originales]
    plt.scatter(x_pts, y_pts, marker='o', label="Puntos Originales")

    plt.title("Spline cúbico generado a partir de coeficientes")
    plt.grid(True)
    plt.legend()
    plt.show()
```

In [ ]:

```
# ================ USO DEL MÉTODO =================
# EJEMPLO ( con error en el calculo de a,b,c,d):

a = [1, 2.5]
b = [0.5, -1]
c = [0.2, 0.3]
d = [-0.1, 0.05]

x_intervals = [0, 1, 2]
puntos_originales = [(0, 1), (1, 2), (2, 3)]

graficar_splines(a, b, c, d, x_intervals, puntos_originales)
```

```
FUNCIONES DE SPLINE OBTENIDAS:

S0(x) = 1.000000 + 0.500000(x - 0) + 0.200000(x - 0)^2 + -0.100000(x - 0)^3
S1(x) = 2.500000 + -1.000000(x - 1) + 0.300000(x - 1)^2 + 0.050000(x - 1)^3
```