Escuela Politécnica Nacional

Nombre: Kevin Eduardo Garcia Rodriguez

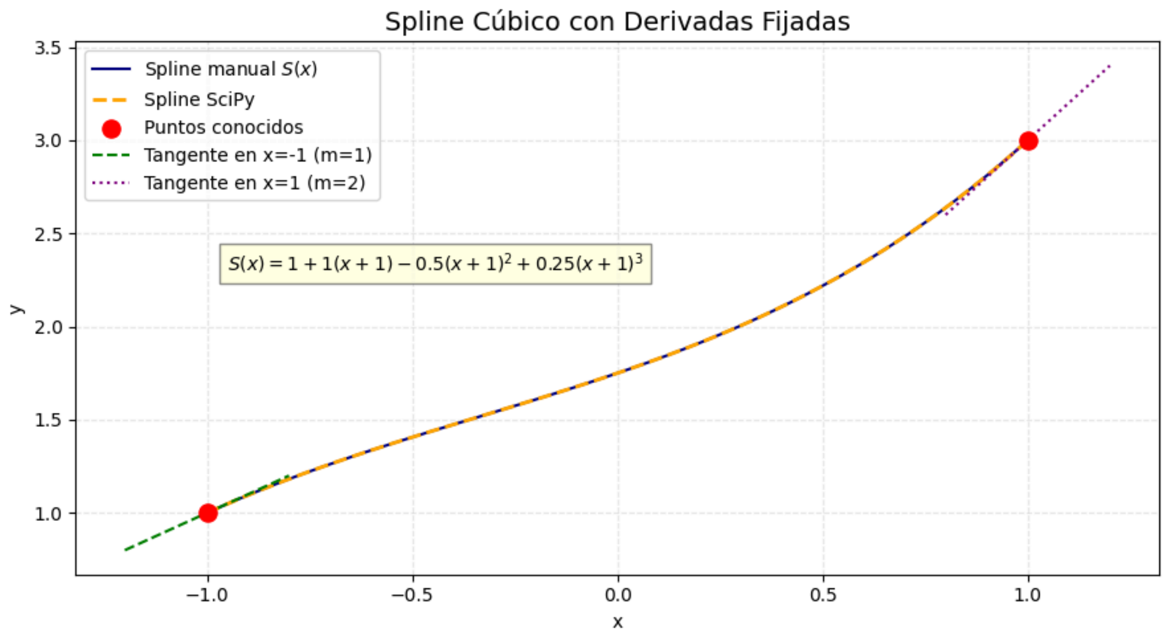
Tema: [Tarea 7] Splines Cubicos

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

1. Dados los puntos (0,1) , (1,5) , (2,3) determine el spline cúbico

In [4]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline

# ▶ Datos base
puntos_x = np.array([-1, 1])
puntos_y = np.array([1, 3])
pendientes = [1, 2]  # Derivadas en los extremos: f'(-1)=1, f'(1)=2

# ▶ Spline cúbico con Scipy (Condiciones de contorno de tipo derivada)
spline_scipy = CubicSpline(puntos_x, puntos_y, bc_type=((1, pendientes[0]), (1,

# ▶ Spline manual: S(x) = a + b(x+1) + c(x+1)^2 + d(x+1)^3
a, b, c, d = 1, 1, -0.5, 0.25

def spline_manual(x):
    dx = x + 1
    return a + b * dx + c * dx**2 + d * dx**3

def spline_manual_derivada(x):
    dx = x + 1
    return b + 2 * c * dx + 3 * d * dx**2

# ▶ Dominio para graficar
x_dom = np.linspace(-1, 1, 300)
y_manual = spline_manual(x_dom)
y_scipy = spline_scipy(x_dom)

# ▶ Gráfico
plt.figure(figsize=(9, 5))
plt.plot(x_dom, y_manual, color='navy', linestyle='-', label='Spline manual $S(x
plt.plot(x_dom, y_scipy, color='orange', linestyle='--', linewidth=2, label='Spl
plt.scatter(puntos_x, puntos_y, s=90, color='red', zorder=5, label='Puntos conoc

# ▶ Tangentes en extremos
x_tg1 = np.linspace(-1.2, -0.8, 20)
x_tg2 = np.linspace(0.8, 1.2, 20)
plt.plot(x_tg1, puntos_y[0] + pendientes[0]*(x_tg1 - puntos_x[0]), 'g--', label=
plt.plot(x_tg2, puntos_y[1] + pendientes[1]*(x_tg2 - puntos_x[1]), 'purple', lin

# ▶ Etiqueta con ecuación
plt.text(-0.95, 2.3, r"$S(x) = 1 + 1(x+1) - 0.5(x+1)^2 + 0.25(x+1)^3$", fontsize
         bbox=dict(facecolor='lightyellow', edgecolor='gray', alpha=0.9))

# ▶ Decoración del gráfico
```

```
plt.title("Spline Cúbico con Derivadas Fijadas", fontsize=14)
plt.xlabel("x")
plt.ylabel("y")
plt.grid(True, linestyle='--', alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()

# ▶ Salida en consola
print("=== Coeficientes del spline manual ===")
print(f"S(x) = {a} + {b}(x+1) + {c}(x+1)^2 + {d}(x+1)^3")
print("Forma expandida: 0.25x^3 + 0.25x^2 + 0.75x + 1.75")

print("\n=== Verificación de condiciones ===")
print(f"S(-1) = {spline_manual(-1)}     (esperado: 1)")
print(f"S(1)  = {spline_manual(1)}      (esperado: 3)")
print(f"S'(-1) = {spline_manual_derivada(-1)}   (esperado: 1)")
print(f"S'(1)  = {spline_manual_derivada(1)}    (esperado: 2)")
```



```
=== Coeficientes del spline manual ===
S(x) = 1 + 1(x+1) + -0.5(x+1)^2 + 0.25(x+1)^3
Forma expandida: 0.25x^3 + 0.25x^2 + 0.75x + 1.75

=== Verificación de condiciones ===
S(-1) = 1.0     (esperado: 1)
S(1)  = 3.0      (esperado: 3)
S'(-1) = 1.0   (esperado: 1)
S'(1)  = 2.0    (esperado: 2)
```

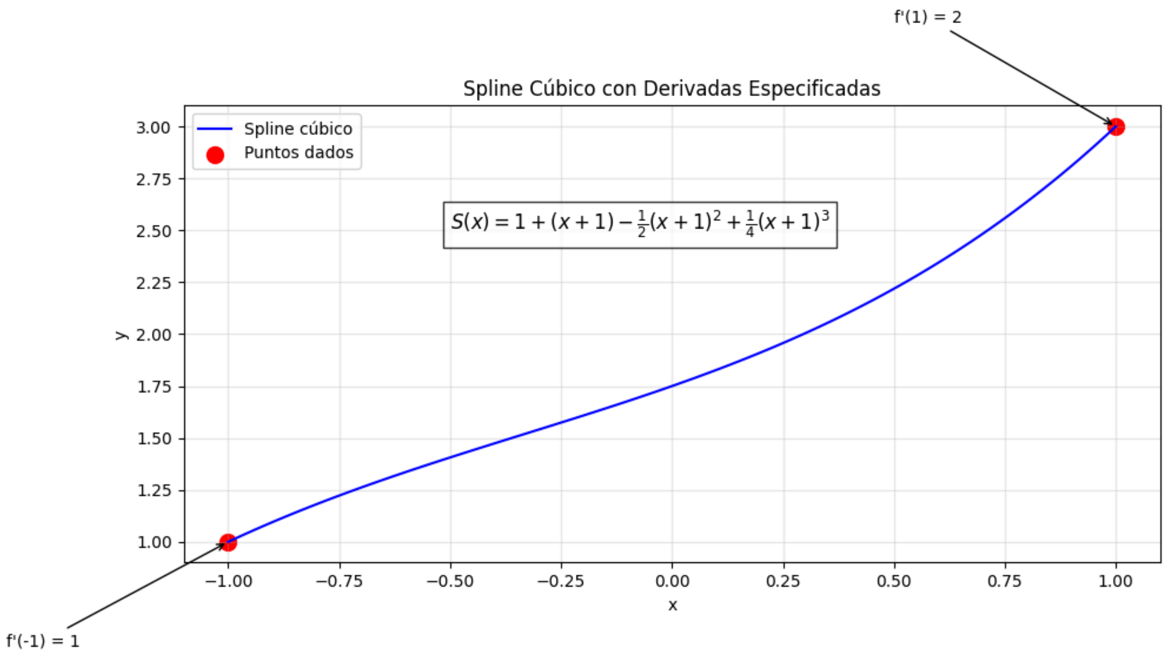
2. Dados los puntos (−1,1), (1,3), determine el spline cúbico sabiendo que

$f'(x0) = 1$

$f'(xn) = 2$

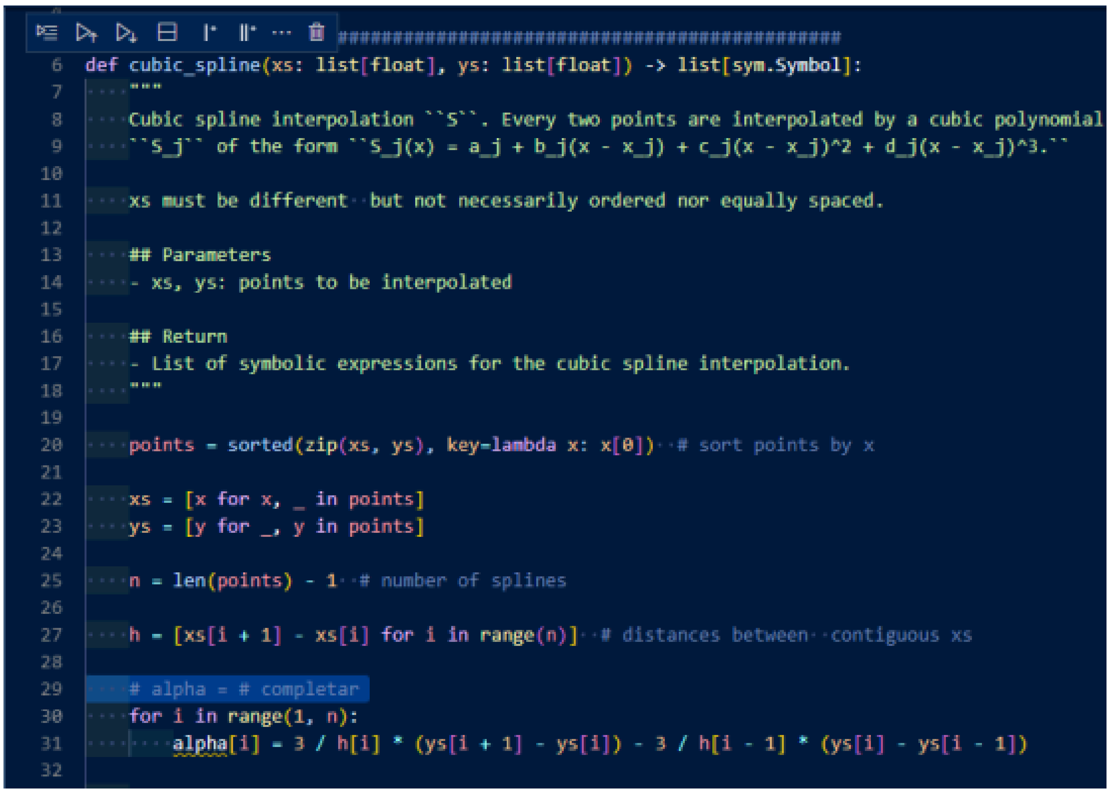
In [5]:
```
import numpy as np
import matplotlib.pyplot as plt

# Datos del problema
```

```
x = np.array([-1, 1])
y = np.array([1, 3])
derivadas = np.array([1, 2])  # f'(-1)=1, f'(1)=2

# Coeficientes obtenidos manualmente
a, b, c, d = 1, 1, -0.5, 0.25

# Definición del spline
def S(x_val):
    return a + b*(x_val + 1) + c*(x_val + 1)**2 + d*(x_val + 1)**3

# Definición de la derivada
def S_prime(x_val):
    return b + 2*c*(x_val + 1) + 3*d*(x_val + 1)**2

# Crear datos para graficar
x_vals = np.linspace(-1, 1, 100)
y_vals = S(x_vals)

# Gráfico
plt.figure(figsize=(10, 6))
plt.plot(x_vals, y_vals, 'b-', label='Spline cúbico')
plt.scatter(x, y, color='red', s=100, label='Puntos dados')

# Marcamos las derivadas
plt.annotate(f"f'(-1) = {derivadas[0]}", xy=(-1, 1), xytext=(-1.5, 0.5),
             arrowprops=dict(arrowstyle='->'))
plt.annotate(f"f'(1) = {derivadas[1]}", xy=(1, 3), xytext=(0.5, 3.5),
             arrowprops=dict(arrowstyle='->'))

# Ecuación en el gráfico
ecuacion = r'$S(x) = 1 + (x+1) - \frac{1}{2}(x+1)^2 + \frac{1}{4}(x+1)^3$'
plt.text(-0.5, 2.5, ecuacion, fontsize=12, bbox=dict(facecolor='white', alpha=0.

plt.title('Spline Cúbico con Derivadas Especificadas')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()

# Verificación
print("Verificación:")
print(f"S(-1) = {S(-1)} (debería ser 1)")
print(f"S(1) = {S(1)} (debería ser 3)")
print(f"S'(-1) = {S_prime(-1)} (debería ser 1)")
print(f"S'(1) = {S_prime(1)} (debería ser 2)")
```



```
Verificación:
S(-1) = 1.0 (debería ser 1)
S(1) = 3.0 (debería ser 3)
S'(-1) = 1.0 (debería ser 1)
S'(1) = 2.0 (debería ser 2)
```

3. Diríjase al pseudocódigo del spline cúbico con frontera natural provisto en clase, en base a ese pseudocódigo complete la siguiente función:



```python
##################################################
def cubic_spline(xs: list[float], ys: list[float]) -> list[sym.Symbol]:
    """
    Cubic spline interpolation ``S``. Every two points are interpolated by a cubic polynomial
    ``S_j`` of the form ``S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3.``

    xs must be different  but not necessarily ordered nor equally spaced.

    ## Parameters
    - xs, ys: points to be interpolated

    ## Return
    - List of symbolic expressions for the cubic spline interpolation.
    """

    points = sorted(zip(xs, ys), key=lambda x: x[0])  # sort points by x

    xs = [x for x, _ in points]
    ys = [y for _, y in points]

    n = len(points) - 1  # number of splines

    h = [xs[i + 1] - xs[i] for i in range(n)]  # distances between  contiguous xs

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

```

In [6]:
```python
import sympy as sym
from IPython.display import display

def cubic_spline(xs: list[float], ys: list[float]) -> list[sym.Symbol]:
    points = sorted(zip(xs, ys), key=lambda x: x[0])
    xs = [x for x, _ in points]
    ys = [y for _, y in points]

    n = len(points) - 1
    h = [xs[i+1] - xs[i] for i in range(n)]

    alpha = [0] * (n+1)
    for i in range(1, n):
        alpha[i] = (3/h[i])*(ys[i+1]-ys[i]) - (3/h[i-1])*(ys[i]-ys[i-1])

    l = [1.0]
    mu = [0.0]
    z = [0.0]

    for i in range(1, n):
        l.append(2*(xs[i+1]-xs[i-1]) - h[i-1]*mu[i-1])
        mu.append(h[i]/l[i])
        z.append((alpha[i] - h[i-1]*z[i-1])/l[i])

    l.append(1.0)
    z.append(0.0)
    c = [0.0]*(n+1)

    for i in range(n-1, -1, -1):
        c[i] = z[i] - mu[i]*c[i+1]

    x = sym.Symbol('x')
    splines = []

    for j in range(n):
        a = ys[j]
        b = (ys[j+1]-ys[j])/h[j] - h[j]*(c[j+1]+2*c[j])/3
        d = (c[j+1]-c[j])/(3*h[j])

        S = a + b*(x-xs[j]) + c[j]*(x-xs[j])**2 + d*(x-xs[j])**3
        splines.append(S)

    return splines
```

In [7]:
```
xs = [1, 2, 3]
ys = [-5, -4, 3]

splines = cubic_spline(xs=xs, ys=ys)
_ = [display(s) for s in splines]
print("______")
_ = [display(s.expand()) for s in splines]
```

$$-0.5x + 1.5(x-1)^3 - 4.5$$

$$4.0x - 1.5(x-2)^3 + 4.5(x-2)^2 - 12.0$$

```
______
```

$$1.5x^3 - 4.5x^2 + 4.0x - 6.0$$

$$-1.5x^3 + 13.5x^2 - 32.0x + 18.0$$

4. Usando la función anterior, encuentre el spline cúbico para:

xs = [1, 2, 3]

ys = [2, 3, 5]

In [8]:
```
import sympy as sym
from IPython.display import display

def cubic_spline(xs: list[float], ys: list[float]) -> list[sym.Symbol]:

    points = sorted(zip(xs, ys), key=lambda x: x[0])
    xs = [x for x, _ in points]
    ys = [y for _, y in points]

    n = len(points) - 1
    h = [xs[i+1] - xs[i] for i in range(n)]

    alpha = [0] * (n+1)
    for i in range(1, n):
        alpha[i] = (3/h[i])*(ys[i+1]-ys[i]) - (3/h[i-1])*(ys[i]-ys[i-1])

    l = [1.0]
    mu = [0.0]
    z = [0.0]

    for i in range(1, n):
        l.append(2*(xs[i+1]-xs[i-1]) - h[i-1]*mu[i-1])
        mu.append(h[i]/l[i])
        z.append((alpha[i] - h[i-1]*z[i-1])/l[i])

    l.append(1.0)
    z.append(0.0)
    c = [0.0]*(n+1)

    for i in range(n-1, -1, -1):
        c[i] = z[i] - mu[i]*c[i+1]

    x = sym.Symbol('x')
    splines = []

    for j in range(n):
        a = ys[j]
        b = (ys[j+1]-ys[j])/h[j] - h[j]*(c[j+1]+2*c[j])/3
        d = (c[j+1]-c[j])/(3*h[j])

        S = a + b*(x-xs[j]) + c[j]*(x-xs[j])**2 + d*(x-xs[j])**3
        splines.append(S)

    return splines
```

In [9]:
```
# Example usage
xs = [0, 1, 2]
ys = [2, 3, 5]

splines = cubic_spline(xs=xs, ys=ys)
print("Spline expressions:")
_ = [display(s) for s in splines]
print("\nExpanded forms:")
_ = [display(s.expand()) for s in splines]
```

```
Spline expressions:
```

$$0.25x^3 + 0.75x + 2$$

$$1.5x - 0.25(x-1)^3 + 0.75(x-1)^2 + 1.5$$

```
Expanded forms:
```

$$0.25x^3 + 0.75x + 2$$

$$-0.25x^3 + 1.5x^2 - 0.75x + 2.5$$

5) Usando la función anterior, encuentre el spline cúbico para:

xs = [0, 1, 2, 3]

ys = [-1 ,1, 5, 2]

In [10]:
```python
import sympy as sym
from IPython.display import display


def cubic_spline_clamped(
    xs: list[float], ys: list[float], B0: float, B1: float
) -> list[sym.Symbol]:


    points = sorted(zip(xs, ys), key=lambda x: x[0])
    xs = [x for x, _ in points]
    ys = [y for _, y in points]
    n = len(points) - 1
    h = [xs[i + 1] - xs[i] for i in range(n)]

    alpha = [0] * (n + 1)
    alpha[0] = 3 / h[0] * (ys[1] - ys[0]) - 3 * B0
    alpha[-1] = 3 * B1 - 3 / h[n - 1] * (ys[n] - ys[n - 1])

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

    l = [2 * h[0]]
    u = [0.5]
    z = [alpha[0] / l[0]]

    for i in range(1, n):
        l += [2 * (xs[i + 1] - xs[i - 1]) - h[i - 1] * u[i - 1]]
        u += [h[i] / l[i]]
        z += [(alpha[i] - h[i - 1] * z[i - 1]) / l[i]]

    l.append(h[n - 1] * (2 - u[n - 1]))
    z.append((alpha[n] - h[n - 1] * z[n - 1]) / l[n])
    c = [0] * (n + 1)
    c[-1] = z[-1]

    x = sym.Symbol("x")
    splines = []
    for j in range(n - 1, -1, -1):
        c[j] = z[j] - u[j] * c[j + 1]
        b = (ys[j + 1] - ys[j]) / h[j] - h[j] * (c[j + 1] + 2 * c[j]) / 3
        d = (c[j + 1] - c[j]) / (3 * h[j])
        a = ys[j]
```

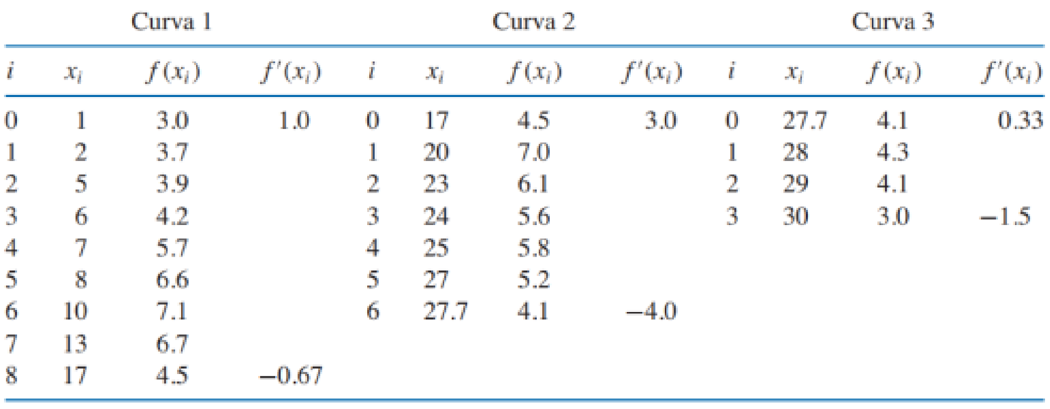
```
            print(j, a, b, c[j], d)
            S = a + b * (x - xs[j]) + c[j] * (x - xs[j]) ** 2 + d * (x - xs[j]) ** 3

            splines.append(S)
        splines.reverse()
        return splines
```

In [11]:
```
cubic_spline_clamped(
    xs=[0, 1, 2, 3],
    ys=[-1, 1, 5, 2],
    B0=1,
    B1=10,
)
```

```
2 5 -3.0 -13.0 13.0
1 1 5.0 5.0 -6.0
0 -1 1.0 -1.0 2.0
```

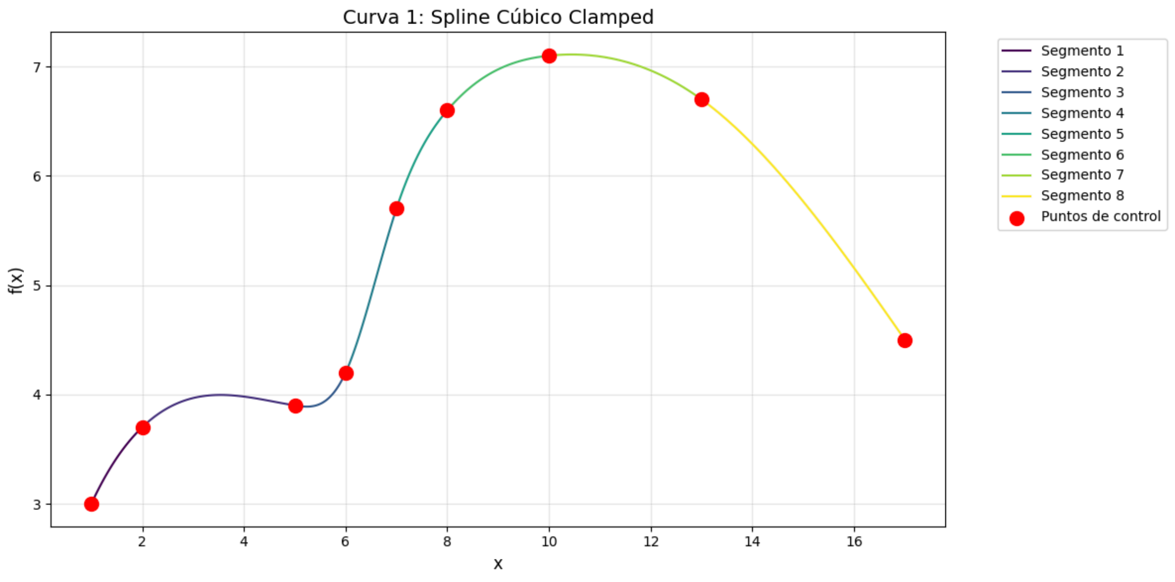
Out[11]:
```
[2.0*x**3 - 1.0*x**2 + 1.0*x - 1,
 5.0*x - 6.0*(x - 1)**3 + 5.0*(x - 1)**2 - 4.0,
 -3.0*x + 13.0*(x - 2)**3 - 13.0*(x - 2)**2 + 11.0]
```

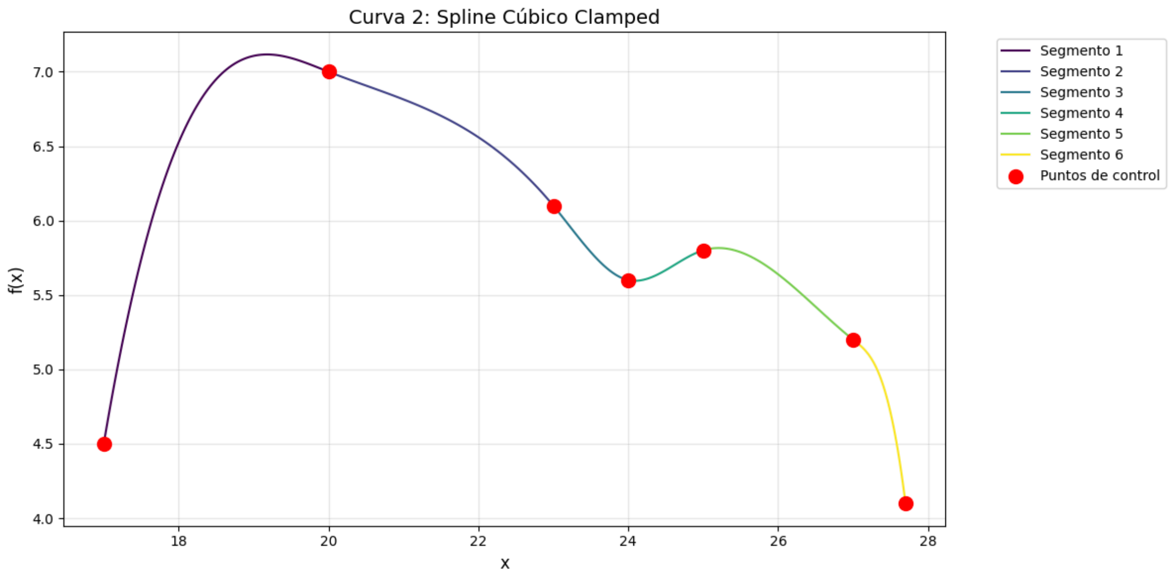
6) Use la función cubic_spline_clamped, provista en el enlace de Github, para graficar los datos de la siguiente tabla.

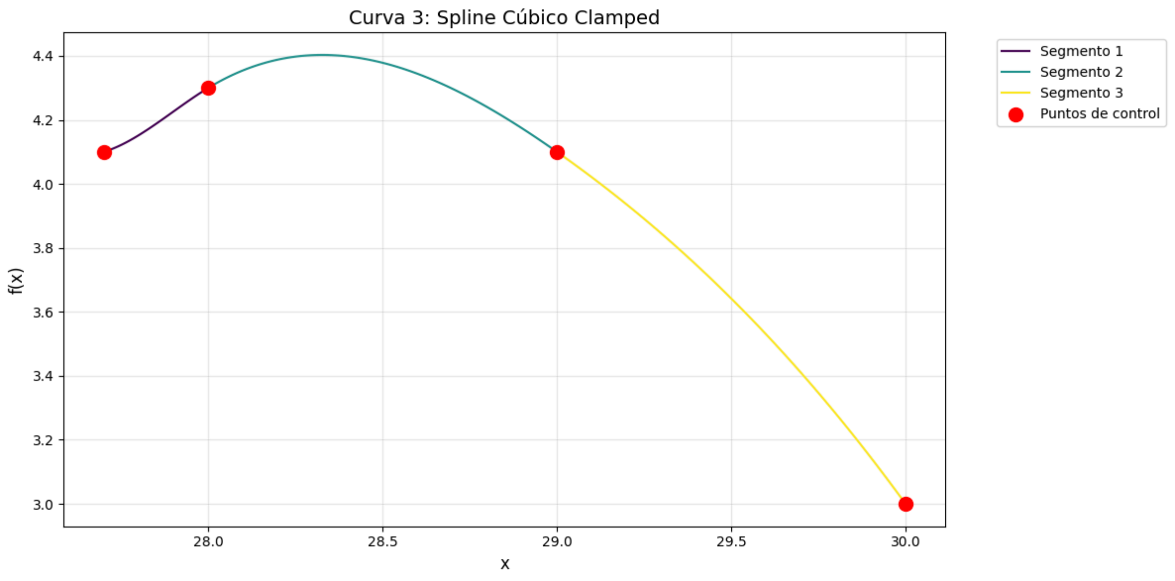


Curva 1				Curva 2				Curva 3			
i	x_i	$f(x_i)$	$f'(x_i)$	i	x_i	$f(x_i)$	$f'(x_i)$	i	x_i	$f(x_i)$	$f'(x_i)$
0	1	3.0	1.0	0	17	4.5	3.0	0	27.7	4.1	0.33
1	2	3.7		1	20	7.0		1	28	4.3	
2	5	3.9		2	23	6.1		2	29	4.1	
3	6	4.2		3	24	5.6		3	30	3.0	−1.5
4	7	5.7		4	25	5.8					
5	8	6.6		5	27	5.2					
6	10	7.1		6	27.7	4.1	−4.0				
7	13	6.7									
8	17	4.5	−0.67								

In [12]:
```
import sympy as sym
from IPython.display import display, Math
import matplotlib.pyplot as plt
import numpy as np

def cubic_spline_clamped(xs: list[float], ys: list[float], B0: float, B1: float)
    points = sorted(zip(xs, ys), key=lambda x: x[0])
    xs = [x for x, _ in points]
    ys = [y for _, y in points]
    n = len(points) - 1
    h = [xs[i+1] - xs[i] for i in range(n)]

    alpha = [0] * (n+1)
    alpha[0] = 3/h[0]*(ys[1]-ys[0]) - 3*B0
    alpha[-1] = 3*B1 - 3/h[n-1]*(ys[n]-ys[n-1])
    for i in range(1, n):
        alpha[i] = 3/h[i]*(ys[i+1]-ys[i]) - 3/h[i-1]*(ys[i]-ys[i-1])

    l = [2*h[0]]
    mu = [0.5]
    z = [alpha[0]/l[0]]
    for i in range(1, n):
        l.append(2*(xs[i+1]-xs[i-1]) - h[i-1]*mu[i-1])
        mu.append(h[i]/l[i])
        z.append((alpha[i] - h[i-1]*z[i-1])/l[i])
    l.append(h[n-1]*(2 - mu[n-1]))
    z.append((alpha[n] - h[n-1]*z[n-1])/l[n])

    c = [0]*(n+1)
    c[-1] = z[-1]

    x = sym.Symbol('x')
    splines = []
    for j in range(n-1, -1, -1):
        c[j] = z[j] - mu[j]*c[j+1]
        b = (ys[j+1]-ys[j])/h[j] - h[j]*(c[j+1]+2*c[j])/3
        d = (c[j+1]-c[j])/(3*h[j])
        a = ys[j]
        S = a + b*(x-xs[j]) + c[j]*(x-xs[j])**2 + d*(x-xs[j])**3
        splines.append(S)
    splines.reverse()
    return splines

# -------------------- FUNCIÓN DE GRÁFICA --------------------
def plot_splines(xs, ys, splines, title, figsize=(12, 6)):
    plt.figure(figsize=figsize)
    colors = plt.cm.viridis(np.linspace(0, 1, len(splines)))
    x_sym = sym.Symbol('x')

    for i in range(len(splines)):
        x_segment = np.linspace(xs[i], xs[i+1], 100)
        y_segment = [splines[i].subs(x_sym, val) for val in x_segment]
        plt.plot(x_segment, y_segment, color=colors[i], label=f'Segmento {i+1}')

    plt.scatter(xs, ys, color='red', s=100, zorder=5, label='Puntos de control')
    plt.title(title, fontsize=14)
    plt.xlabel('x', fontsize=12)
    plt.ylabel('f(x)', fontsize=12)
    plt.grid(True, alpha=0.3)
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.tight_layout()
    plt.show()

# -------------------- MOSTRAR ECUACIONES --------------------
def display_equations(splines, xs, curve_name):
    print(f"\n{'-'*50}")
    print(f"Ecuaciones para {curve_name}:")
    print(f"{'-'*50}")
    for i, s in enumerate(splines):
        display(Math(f"S_{i}(x) = {sym.latex(s.simplify())} \\quad \\text{{para
        print(f"Forma expandida: {sym.latex(s.expand())}\n")

# -------------------- VERIFICAR FRONTERAS --------------------
def verify_boundary_conditions(splines, xs, ys, B0, B1):
    x = sym.Symbol('x')
    print("\nVerificación de condiciones de frontera:")
    S0_prime = sym.diff(splines[0], x)
    Sn_prime = sym.diff(splines[-1], x)
    calculated_B0 = S0_prime.subs(x, xs[0])
    calculated_B1 = Sn_prime.subs(x, xs[-1])
    print(f"f'({xs[0]}) calculado: {calculated_B0.evalf()}, esperado: {B0}")
    print(f"f'({xs[-1]}) calculado: {calculated_B1.evalf()}, esperado: {B1}")

# -------------------- VERIFICAR CONTINUIDAD --------------------
def check_continuity(splines, xs):
    x = sym.Symbol('x')
    print("\nVerificación de continuidad en puntos intermedios:")
    for i in range(1, len(splines)):
        left = splines[i-1]
        right = splines[i]
        point = xs[i]
        f_cont = sym.simplify(left.subs(x, point) - right.subs(x, point))
        f_prime_cont = sym.simplify(sym.diff(left, x).subs(x, point) - sym.diff(
        f_double_prime_cont = sym.simplify(sym.diff(left, x, 2).subs(x, point) -
        print(f"x = {point}: f = {f_cont}, f' = {f_prime_cont}, f'' = {f_double_

# -------------------- DATOS DE LAS CURVAS --------------------
# Curva 1
xs1 = [1, 2, 5, 6, 7, 8, 10, 13, 17]
ys1 = [3.0, 3.7, 3.9, 4.2, 5.7, 6.6, 7.1, 6.7, 4.5]
B0_1 = 1.0
B1_1 = -0.67

# Curva 2
xs2 = [17, 20, 23, 24, 25, 27, 27.7]
ys2 = [4.5, 7.0, 6.1, 5.6, 5.8, 5.2, 4.1]
B0_2 = 3.0
B1_2 = -4.0

# Curva 3
xs3 = [27.7, 28, 29, 30]
ys3 = [4.1, 4.3, 4.1, 3.0]
B0_3 = 0.33
B1_3 = -1.5

# -------------------- CALCULAR SPLINES --------------------
splines1 = cubic_spline_clamped(xs1, ys1, B0_1, B1_1)
splines2 = cubic_spline_clamped(xs2, ys2, B0_2, B1_2)
splines3 = cubic_spline_clamped(xs3, ys3, B0_3, B1_3)

# -------------------- GRAFICAR CURVAS --------------------
plot_splines(xs1, ys1, splines1, 'Curva 1: Spline Cúbico Clamped')
plot_splines(xs2, ys2, splines2, 'Curva 2: Spline Cúbico Clamped')
plot_splines(xs3, ys3, splines3, 'Curva 3: Spline Cúbico Clamped')

# -------------------- MOSTRAR ECUACIONES --------------------
display_equations(splines1, xs1, "Curva 1")
display_equations(splines2, xs2, "Curva 2")
display_equations(splines3, xs3, "Curva 3")

# -------------------- VERIFICAR FRONTERAS Y CONTINUIDAD --------------------
verify_boundary_conditions(splines1, xs1, ys1, B0_1, B1_1)
verify_boundary_conditions(splines2, xs2, ys2, B0_2, B1_2)
verify_boundary_conditions(splines3, xs3, ys3, B0_3, B1_3)

check_continuity(splines1, xs1)
check_continuity(splines2, xs2)
check_continuity(splines3, xs3)
```







```
--------------------------------------------------
Ecuaciones para Curva 1:
--------------------------------------------------
```

$$S_0(x) = 0.0468099653460708x^3 - 0.487239861384283x^2 + 1.83404982673035x + 1.606380069307$$

```
Forma expandida: 0.0468099653460708 x^{3} - 0.487239861384283 x^{2} + 1.834049826
73035 x + 1.60638006930786
```

$S_1(x) = 0.0265552121382411x^3 - 0.365711342137305x^2 + 1.5909927882364x + 1.768418094970$

```
Forma expandida: 0.0265552121382411 x^{3} - 0.365711342137305 x^{2} + 1.590992788
2364 x + 1.7684180949705
```

$S_2(x) = 0.341862882832256x^3 - 5.09532640254753x^2 + 25.2390680902875x - 37.64504074178$

```
Forma expandida: 0.341862882832256 x^{3} - 5.09532640254753 x^{2} + 25.2390680902
875 x - 37.6450407417814
```

$S_3(x) = -0.574548094033905x^3 + 11.4000711810434x^2 - 73.7333174112578x + 160.2997302613$

```
Forma expandida: - 0.574548094033905 x^{3} + 11.4000711810434 x^{2} - 73.73331741
12578 x + 160.299730261309
```

$S_4(x) = 0.156329493303363x^3 - 3.94835815303925x^2 + 33.7056879273205x - 90.39128219537$

```
Forma expandida: 0.156329493303363 x^{3} - 3.94835815303925 x^{2} + 33.7056879273
205 x - 90.3912821953733
```

$S_5(x) = 0.0239201086447503x^3 - 0.770532921232554x^2 + 8.28308607286689x - 22.5976772501$

```
Forma expandida: 0.0239201086447503 x^{3} - 0.770532921232554 x^{2} + 8.283086072
86689 x - 22.5976772501638
```

$S_6(x) = -0.00255606547823463x^3 + 0.023752302456995x^2 + 0.340233835971401x + 3.87849687$

```
Forma expandida: - 0.00255606547823463 x^{3} + 0.023752302456995 x^{2} + 0.340233
835971401 x + 3.87849687282113
```

$S_7(x) = 0.00574178139926946x^3 - 0.299863725765665x^2 + 4.54724220286598x - 14.35187271$

```
Forma expandida: 0.00574178139926946 x^{3} - 0.299863725765665 x^{2} + 4.54724220
286598 x - 14.3518727170554
```

```
--------------------------------------------------
Ecuaciones para Curva 2:
--------------------------------------------------
```

$S_0(x) = 0.12616207628025x^3 - 7.53497434135573x^2 + 149.806607471118x - 984.439023122068$

```
Forma expandida: 0.12616207628025 x^{3} - 7.53497434135573 x^{2} + 149.8066074711
18 x - 984.439023122068
```

$S_1(x) = -0.022930673285195x^3 + 1.41059063257098x^2 - 29.1046920074162x + 208.3029734014$

```
Forma expandida: - 0.022930673285195 x^{3} + 1.41059063257098 x^{2} - 29.10469200
74162 x + 208.302973401493
```

$S_2(x) = 0.280127236863149x^3 - 19.5004051676648x^2 + 451.848211398006x - 3479.0026193734$

```
Forma expandida: 0.280127236863149 x^{3} - 19.5004051676648 x^{2} + 451.848211398
006 x - 3479.00261937341
```

$S_3(x) = -0.357384536100794x^3 + 26.4004424857391x^2 - 649.772132283688x + 5333.960130080$

```
Forma expandida: - 0.357384536100794 x^{3} + 26.4004424857391 x^{2} - 649.7721322
83688 x + 5333.96013008014
```

$S_4(x) = 0.0882021573401092x^3 - 7.0185595223286x^2 + 185.702917918006x - 1628.3319549339$

```
Forma expandida: 0.0882021573401092 x^{3} - 7.0185595223286 x^{2} + 185.702917918
006 x - 1628.33195493397
```

$S_5(x) = -2.56800212665878x^3 + 208.133987481581x^2 - 5623.41585118756x + 50653.736967016$

```
Forma expandida: - 2.56800212665878 x^{3} + 208.133987481581 x^{2} - 5623.4158511
8756 x + 50653.7369670161


--------------------------------------------------
Ecuaciones para Curva 3:
--------------------------------------------------
```

$S_0(x) = -3.79941327466078x^3 + 317.993289328931x^2 - 8870.74279427938x + 82483.079611294$

```
Forma expandida: - 3.79941327466078 x^{3} + 317.993289328931 x^{2} - 8870.7427942
7938 x + 82483.079611294
```

$S_1(x) = 0.296039603960395x^3 - 26.0247524752475x^2 + 761.762376237622x - 7420.3019801980$

```
Forma expandida: 0.296039603960395 x^{3} - 26.0247524752475 x^{2} + 761.762376237
622 x - 7420.30198019801
```

$S_2(x) = -0.0653465346534656x^3 + 5.41584158415843x^2 - 150.014851485149x + 1393.54455445$

```
Forma expandida: - 0.0653465346534656 x^{3} + 5.41584158415843 x^{2} - 150.014851
485149 x + 1393.54455445545


Verificación de condiciones de frontera:
f'(1) calculado: 1.00000000000000, esperado: 1.0
f'(17) calculado: -0.670000000000000, esperado: -0.67

Verificación de condiciones de frontera:
f'(17) calculado: 3.00000000000000, esperado: 3.0
f'(27.7) calculado: -4.00000000000000, esperado: -4.0

Verificación de condiciones de frontera:
f'(27.7) calculado: 0.329999999999998, esperado: 0.33
f'(30) calculado: -1.50000000000000, esperado: -1.5

Verificación de continuidad en puntos intermedios:
x = 2: f = 4.44089209850063E-16, f' = 1.11022302462516E-16, f'' = -5.551115123125
78E-17
x = 5: f = 8.88178419700125E-16, f' = 0, f'' = 7.77156117237610E-16
x = 6: f = -8.88178419700125E-16, f' = 0, f'' = 0
x = 7: f = 8.88178419700125E-16, f' = 4.44089209850063E-16, f'' = -1.776356839400
25E-15
x = 8: f = 8.88178419700125E-16, f' = 8.88178419700125E-16, f'' = 4.4408920985006
3E-16
x = 10: f = 0, f' = 2.22044604925031E-16, f'' = 2.77555756156289E-17
x = 13: f = 0, f' = -2.22044604925031E-16, f'' = -2.77555756156289E-17

Verificación de continuidad en puntos intermedios:
x = 20: f = -8.88178419700125E-16, f' = -5.32907051820075E-15, f'' = -4.440892098
50063E-16
x = 23: f = 0, f' = 4.44089209850063E-16, f'' = 4.44089209850063E-16
x = 24: f = -1.77635683940025E-15, f' = -1.77635683940025E-15, f'' = 0
x = 25: f = 0, f' = 8.88178419700125E-16, f'' = -1.77635683940025E-15
x = 27: f = 8.88178419700125E-16, f' = 8.88178419700125E-16, f'' = 1.065814103640
15E-14

Verificación de continuidad en puntos intermedios:
x = 28: f = -8.88178419700125E-16, f' = -8.88178419700125E-16, f'' = 7.1054273576
0100E-14
x = 29: f = -8.88178419700125E-16, f' = -8.88178419700125E-15, f'' = 7.1054273576
0100E-15
```