In [ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
```

In [ ]:
```python
def make_system(xs: np.ndarray, ys: np.ndarray) :
    # Constructs the matrix A and the vector b for spline interpolation
    # Might not work
    N = len(xs)
    A = np.zeros((N, N))
    b = np.zeros(N)
    # Make A
    for i in range(N):
        if i == 0:
            temp = 1.0 / (xs[i+1] - xs[i])
            A[i][i] = 2.0 * temp
            A[i][i+1] = temp
        elif i == N-1:
            temp = 1.0 / (xs[i] - xs[i-1])
            A[i][i] = 2.0 * temp
            A[i][i-1] = temp
        else:
            A[i][i-1] = 1.0 / (xs[i] - xs[i-1])
            A[i][i+1] = 1.0 / (xs[i+1] - xs[i])
            A[i][i] = 2.0* (A[i][i-1] + A[i][i+1])

    # Make b
    for i in range(N):
        if i == 0:
            temp = (ys[i+1] - ys[i]) / (xs[i+1] - xs[i])**2
            b[i] = 3 * temp
        elif i == N-1:
            temp = (ys[i] - ys[i-1]) / (xs[i] - xs[i-1])**2
            b[i] = 3 * temp
        else:
            templ = (ys[i] - ys[i-1]) / (xs[i] - xs[i-1])**2
            tempr = (ys[i+1] - ys[i]) / (xs[i+1] - xs[i])**2
            b[i] = 3* (templ + tempr)
    return A, b

def make_terms(xs: np.ndarray, ys: np.ndarray, ks: np.ndarray):
    N = len(xs)
    ays = np.zeros(N-1)
    bs = np.zeros(N-1)
    for i in range(1, N):
        ays[i-1] = ks[i-1] * (xs[i] - xs[i-1]) - (ys[i] - ys[i-1])
        bs[i-1] = -ks[i] * (xs[i] - xs[i-1]) + (ys[i] - ys[i-1])
    return ays, bs

def make_spline(x_left: float, x_right: float, y_left: float, y_right: float, k: float, a: float, b: float):
    # Makes the spline bewteen the two endpoints
    # 100 is a placeholder
    t = np.linspace(0, 1, 100)  # t should range from 0 to 1

    # Construct the spline between the two endpoints
    q_i = (1.0 - t) * y_left + t * y_right + t * (1.0 - t) * ((1.0 - t) * a + t * b)

    return q_i

def cubic_interpolation(xs: np.ndarray, ys: np.ndarray):
    x_interp = np.zeros((len(xs)-1, 100))
    y_interp = np.zeros((len(ys)-1, 100))
    # Make system
    A, b = make_system(xs, ys)
    # solve system
    ks = sp.linalg.solve(A, b, assume_a='sym')
    ays, bs = make_terms(xs, ys, ks)
    for i in range(len(xs)-1):
        x_interp[i, :] = np.linspace(xs[i], xs[i+1], 100)
        y_interp[i, :] = make_spline(xs[i], xs[i+1], ys[i], ys[i+1], ks[i], ays[i], bs[i])
    return x_interp, y_interp
```

In [ ]:
```python
colors = ['tab:blue', 'tab:green', 'tab:red', 'k']
ns = [3, 5, 7]

# Lists to store labels and handles for legend
labels = []
handles = []

for i, n in enumerate(ns):
    xs = np.linspace(-1, 1, n)
    ys = xs**3
    x_cubic, y_cubic = cubic_interpolation(xs, ys)

    for j in range(len(x_cubic)):
        plt.plot(x_cubic[j], y_cubic[j], color=colors[i])  # No need for labels here

    # Collect labels and handles for legend
    labels.append(f"{n}")
    handles.append(plt.Line2D([], [], color=colors[i]))  # Creating handles manually

xs = np.linspace(-1, 1)
ys = xs**3

# Plot the original function (outside the loop)
plt.plot(xs, ys, label='Original Function', color = colors[-1])
labels.append(r"$x^3$")
handles.append(plt.Line2D([], [], color=colors[-1]))
```

```python
# Add legend using collected labels and handles
plt.legend(handles, labels)
plt.grid(True)
plt.gca().set_facecolor((0.9, 0.9, 0.9))
plt.xlabel("x")
plt.ylabel("y")
plt.title(r"Cubic Splines of $f(x) = x^3$")

# Show the plot
plt.show()
```

In [ ]:

```python
colors = ['tab:blue', 'tab:green', 'tab:red', 'k']
ns = [3, 5, 7]
indices = {}
# Lists to store labels and handles for legend
labels = []
handles = []


for i, n in enumerate(ns):
    xs = np.linspace(-1, 1, n)
    ys = xs**3

    x_cubic, y_cubic = cubic_interpolation(xs, ys)
    y_true = np.zeros_like(x_cubic)
    y_err = np.zeros_like(x_cubic)
    for k in range(len(x_cubic)):
        y_true[k, :] = x_cubic[k]**3
        y_err[k, :] = np.abs(y_cubic[k] - y_true[k])


    y_err_line = y_err.reshape(-1)
    peak_indices, _ = sp.signal.find_peaks(y_err_line)

    index_i = []
    for j in range(len(x_cubic)):
        index_i.append(x_cubic[j][peak_indices[j]%100])
        plt.plot(x_cubic[j], y_err[j], color=colors[i], zorder=0)
        plt.scatter(x_cubic[j][peak_indices[j] % 100], y_err_line[peak_indices[j]], c='k', zorder=1 , s = 15)
    indices[ns[i]] = index_i

    # Collect labels and handles for legend
    labels.append(f"{n}")
    handles.append(plt.Line2D([], [], color=colors[i]))  # Creating handles manually



# Add legend using collected labels and handles
plt.legend(handles, labels)
plt.grid(True)
plt.gca().set_facecolor((0.9, 0.9, 0.9))
plt.xlabel("x")
plt.ylabel("y")
plt.title(r"Cubic Splines of $f(x) = x^3$")
plt.gca().set_axisbelow(True)
# Show the plot
plt.show()
```