Università della Svizzera italiana

Faculty of Informatics

Introduction to Computational Science

Assignment4

Zhixiang Dai, Daniele del Pozzo

Spring Semester, April 9, 2025

1 Linear Spline Interpolation

(a) Construction of the Linear Spline:
For each $i = 1, 2, \ldots, n$, define $s(x)$ in the interval $[x_{i-1}, x_i]$ by

$$s(x) = f(x_{i-1}) \frac{x_i - x}{x_i - x_{i-1}} + f(x_i) \frac{x - x_{i-1}}{x_i - x_{i-1}},$$

that satisfies $s(x_{i-1}) = f(x_{i-1})$ and $s(x_i) = f(x_i)$.

(b) Error Estimate:
In the interval $[x_{i-1}, x_i]$, there exists $\tilde{\xi} \in [x_{i-1}, x_i]$ such that

$$f(x) - s(x) = \frac{f''(\tilde{\xi})}{2} (x - x_{i-1})(x - x_i).$$

Taking absolute values gives

$$|f(x) - s(x)| = \frac{|f''(\tilde{\xi})|}{2} |(x - x_{i-1})(x - x_i)|.$$

The maximum of $|(x - x_{i-1})(x - x_i)|$ in $[x_{i-1}, x_i]$ occurs at the midpoint $x = \frac{x_{i-1}+x_i}{2}$ and is

$$\max_{x \in [x_{i-1}, x_i]} |(x - x_{i-1})(x - x_i)| = \frac{h^2}{4}.$$

Hence, for all $x \in [x_{i-1}, x_i]$,

$$|f(x) - s(x)| \leq \frac{|f''(\tilde{\xi})|}{2} \cdot \frac{h^2}{4} = \frac{h^2}{8} |f''(\tilde{\xi})|.$$

By the Mean Value Theorem, there exists some $\xi_i \in [x_{i-1}, x_i]$ such that

$$|f(x) - s(x)| \leq \frac{1}{8} h^2 |f''(\xi_i)|.$$

QED.

2 Adaptive Interpolation

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

def compress(x, y, tol):
    comp_x = [x[0]]
    comp_y = [y[0]]
    start_idx = 0
    max_err = 0.0

    for i in range(2, len(x)):
        x_start, x_curr = x[start_idx], x[i]
        y_start, y_curr = y[start_idx], y[i]
        local_max_err = 0.0
        for j in range(start_idx + 1, i):
            t = (x[j] - x_start) / (x_curr - x_start)
            y_lin = (1 - t) * y_start + t * y_curr
            err = abs(y[j] - y_lin)
            local_max_err = max(local_max_err, err)
        if local_max_err > tol:
            comp_x.append(x[i - 1])
            comp_y.append(y[i - 1])
            start_idx = i - 1
            max_err = max(max_err, local_max_err)

    comp_x.append(x[-1])
    comp_y.append(y[-1])
    max_err = 0.0
    for k in range(len(comp_x) - 1):
        left_idx = np.where(x == comp_x[k])[0][0]
        right_idx = np.where(x == comp_x[k + 1])[0][0]
        for j in range(left_idx + 1, right_idx):
            t = (x[j] - comp_x[k]) / (comp_x[k + 1] - comp_x[k])
            y_lin = (1 - t) * comp_y[k] + t * comp_y[k + 1]
            err = abs(y[j] - y_lin)
            max_err = max(max_err, err)

    return np.array(comp_x), np.array(comp_y), max_err

# ---------------------- Testing the compress function ---------------------- #

delta = np.linspace(0, 2 * np.pi, 51)
y = np.sin(delta)
tol = 0.01

comp_x, comp_y, max_err = compress(delta, y, tol)

print("Original number of nodes:", len(delta))
print("Compressed number of nodes:", len(comp_x))
print("Maximum interpolation error:", max_err)

plt.figure(figsize=(8, 4))
plt.plot(delta, y, 'b.-', label='Original')
plt.plot(comp_x, comp_y, 'ro-', label='Compressed')
plt.title('Adaptive Interpolation with Tolerance = 0.01')
plt.xlabel('x')
```

```
plt.ylabel('y')
plt.legend()
plt.show()
```

The results are

```
Original number of nodes: 51
Compressed number of nodes: 22
Maximum interpolation error: 0.00871233077845468
```

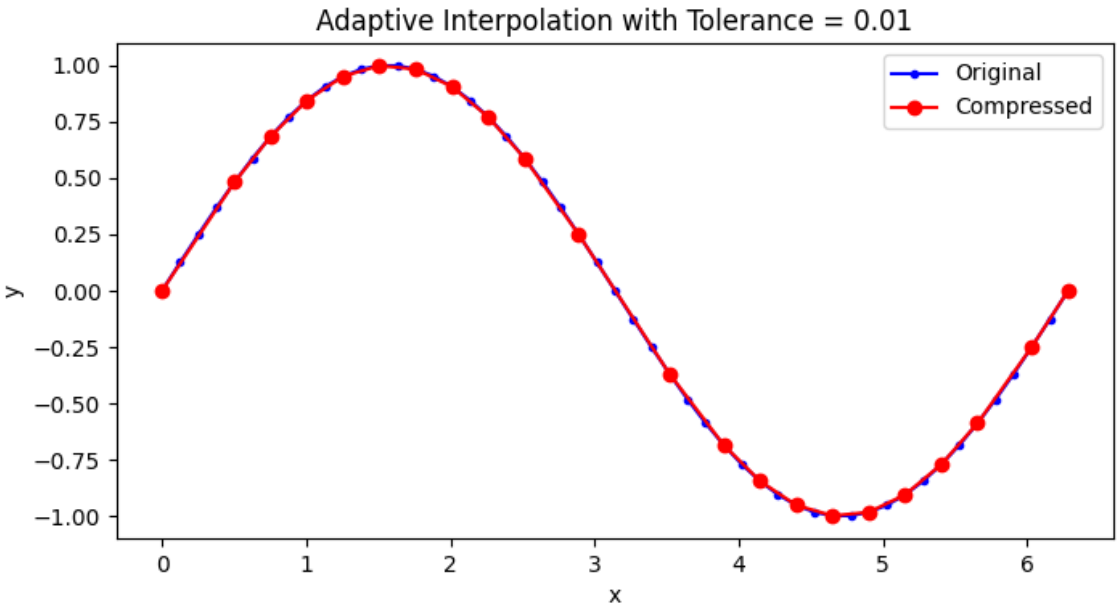

Figure 1: Adaptive Interpolation with Tolerance = 0.01

3 Cubic Spline Interpolation

Given the cubic B-spline:

$$B_3(x) = \frac{1}{6}\begin{cases}(2-|x|)^3 - 4(1-|x|)^3, & |x| \le 1\\ (2-|x|)^3, & 1 < |x| \le 2\\ 0, & |x| > 2\end{cases}$$

We look for the spline of the form:

$$s(x) = \sum_{j=-1}^{3} a_j B_3(x-j)$$

which interpolates the points:

$$(x_0, y_0) = (0,1), \quad (x_1, y_1) = (1,5), \quad (x_2, y_2) = (2,1)$$

with natural boundary conditions:

$$s^{(2)}(0) = 0, \quad s^{(2)}(2) = 0$$

Step 1

To determine the coefficients $a_{-1}, a_0, a_1, a_2, a_3$, we form the linear system using interpolation and boundary conditions.

We know:

$$B_3(0) = \frac{2}{3}, \quad B_3(\pm 1) = \frac{1}{6}$$
$$B_3^{(2)}(0) = -2, \quad B_3^{(2)}(\pm 1) = 1$$

Step 2

The full linear system is:

$$\begin{bmatrix} 1 & -2 & 1 & 0 & 0 \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & 0 & 0 \\ 0 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & 0 \\ 0 & 0 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ 0 & 0 & 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} a_{-1} \\ a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 0 \\ y_0 \\ y_1 \\ y_2 \\ 0 \end{bmatrix}$$

That is:

$$\begin{bmatrix} 1 & -2 & 1 & 0 & 0 \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & 0 & 0 \\ 0 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & 0 \\ 0 & 0 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ 0 & 0 & 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} a_{-1} \\ a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 5 \\ 1 \\ 0 \end{bmatrix}$$

Solving this system gives the correct values for a_j, and the final spline is:

$$s(x) = \sum_{j=-1}^{3} a_j B_3(x - j)$$

4 Programming Task I

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

# Cubic B-spline basis function
def B3(x):
    x = np.atleast_1d(x)  # ensure array input
    y = np.zeros_like(x, dtype=np.float64)

    # Piecewise cases
    i1 = (-2 < x) & (x < -1)
    i2 = (-1 <= x) & (x < 1)
    i3 = (1 <= x) & (x < 2)

    y[i1] = 0.5 * (x[i1] + 2)**3
    y[i2] = 0.5 * (3 * np.abs(x[i2])**3 - 6 * x[i2]**2 + 4)
    y[i3] = 0.5 * (2 - x[i3])**3

    return y / 3

# Function to compute spline coefficients (alpha)
def b3interpolate(y):
```

```
    n = len(y)
    A = np.zeros((n, n))

    for i in range(n):
        for j in range(-1, 3):
            idx = i + j
            if 0 <= idx < n:
                A[i, idx] += B3(j)

    # Natural boundary conditions
    A[0] = 0
    A[0, 0:3] = [1, -2, 1]
    y[0] = 0

    A[-1] = 0
    A[-1, -3:] = [1, -2, 1]
    y[-1] = 0

    alpha = np.linalg.solve(A, y)
    return alpha

# Function to evaluate the spline at given x values
def spline_curve(alpha, x):
    x = np.asarray(x)
    s = np.zeros_like(x, dtype=np.float64)
    n = len(alpha)

    for k in range(n):
        s += alpha[k] * B3(x - k)

    return s

# Given data points
y_data = np.array([
    -0.0044, -0.0213, -0.0771, -0.2001, -0.3521, -0.3520,
     0, 1, 2, 3, 4, 5, 0.5741, 0.8673, 0.5741,
     0, -0.3520
])

# Compute spline coefficients
alpha = b3interpolate(y_data.copy())

# Evaluation points
x = np.arange(0, 16.01, 0.01)
v = spline_curve(alpha, x)

# Plot
plt.plot(x, v)
plt.xlabel('x')
plt.ylabel('Spline value')
plt.title('Cubic Spline Interpolation')
plt.show()
```

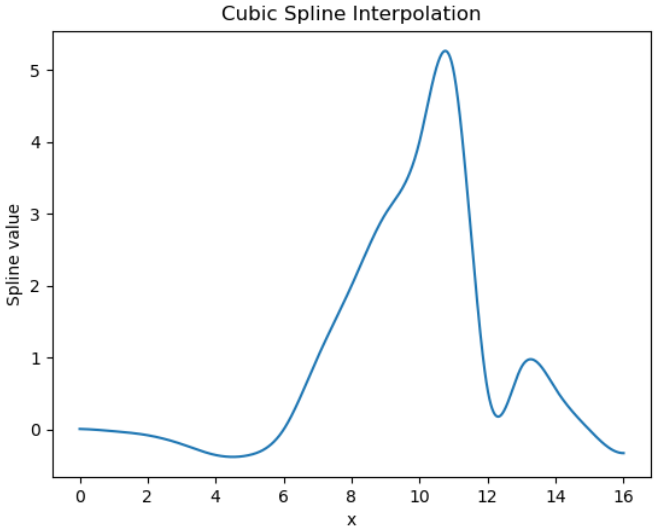

Figure 2: Cubic spline interpolation