assn03

April 8, 2025

1 14

求次数小于等于 3 的多项式 $P(x)$ 使其满足条件

$$P(0) = 0, \quad P'(0) = 1, \quad P(1) = 1, \quad P'(1) = 2$$

1.1 Answer

插值多项式为 $H_3(x)$，满足条件

$$\left.\begin{aligned} H_3(0) = 0, \quad H_3(1) = 1 \\ H_3'(0) = 1, \quad H_3'(1) = 2 \end{aligned}\right\}$$

令

$$H_3(x) = \alpha_0(x)f(0) + \alpha_1(x)f(1) + \beta_0(x)f'(0) + \beta_1(x)f'(1)$$

它们分别满足条件

$$\begin{gathered} \alpha_0(0) = 1, \quad \alpha_0(1) = 0, \quad \alpha_0'(0) = \alpha_0'(1) = 0 \\ \alpha_1(0) = 0, \quad \alpha_1(1) = 1, \quad \alpha_1'(0) = \alpha_1'(1) = 0 \\ \beta_0(0) = \beta_0(1) = 0, \quad \beta_0'(0) = 1, \quad \beta_0'(1) = 0 \\ \beta_1(0) = \beta_1(1) = 0, \quad \beta_1'(0) = 0, \quad \beta_1'(1) = 1 \end{gathered}$$

因此，可令

$$a_0(x) = (ax + b)\left(\frac{x-1}{0-1}\right)^2 = (ax+b)(x-1)^2$$

解得

$$\alpha_0(x) = \left(1 + 2\frac{x-0}{1-0}\right)(x-1)^2 = 1 - 3x^2 + 2x^3$$

同理求得

$$
\begin{aligned}
\alpha_1(x) &= (1+2\frac{x-1}{0-1})x^2 = 3x^2 - 2x^3 \\
\beta_0(x) &= (x-0)(x-1)^2 = x - 2x^2 + x^3 \\
\beta_1(x) &= (x-1)x^2 = -x^2 + x^3
\end{aligned}
$$

故

$$
\begin{aligned}
H_3(x) &= \left[x - 2x^2 + x^3\right] + \left[3x^2 - 2x^3\right] + 2\left[-x^2 + x^3\right] \\
&= x - x^2 + x^3.
\end{aligned}
$$

综上所述，满足要求的三次多项式为

$$
\boxed{P(x) = x - x^2 + x^3.}
$$

2 16

求一个次数不高于 4 次的多项式 $P(x)$，使它满足

$$
P(0) = P'(0) = 0, \quad P(1) = P'(1) = 1, \quad P(2) = 1
$$

2.1 Answer

对节点 0 与 1，有条件

$$
P(0) = 0, \quad P'(0) = 0, \quad P(1) = 1, \quad P'(1) = 1.
$$

埃尔米特插值 (两点三次)

$$
H(x) = f(0)H_{00}(x) + f'(0)H_{10}(x) + f(1)H_{01}(x) + f'(1)H_{11}(x),
$$

其标准基函数（与 14 题相同）为

$$
\begin{aligned}
H_{00}(x) &= 1 - 3x^2 + 2x^3, \\
H_{10}(x) &= x - 2x^2 + x^3, \\
H_{01}(x) &= 3x^2 - 2x^3, \\
H_{11}(x) &= -x^2 + x^3.
\end{aligned}
$$

代入基函数表达式，

$$
H(x) = (3x^2 - 2x^3) + (-x^2 + x^3) = 2x^2 - x^3.
$$

设

$$R(x) = kx^2(x-1)^2.$$

显然：

- $R(0) = 0^2(-1)^2 = 0, \quad R(1) = 1^2 \cdot 0^2 = 0,$
- $R'(0) = 0,\ R'(1) = 0.$

最终插值多项式可写为

$$P(x) = H(x) + R(x) = (2x^2 - x^3) + kx^2(x-1)^2.$$

现在利用 $P(2) = 1$ 来确定 k：

- $H(2) = 2 \cdot 4 - 8 = 0.$
- $R(2) = k \cdot 2^2(2-1)^2 = 4k.$

于是有

$$P(2) = 0 + k \cdot 4 = 1 \quad \Longrightarrow \quad k = \frac{1}{4}.$$

代入 $k = \frac{1}{4}$，得

$$\boxed{P(x) = 2x^2 - x^3 + \frac{1}{4}x^2(x-1)^2.}$$

3 17

设 $f(x) = \frac{1}{1+x^2}$，在 $-5 \le x \le 5$ 上取 $n = 10$，按等距节点求分段线性插值函数 $I_h(x)$，计算各节点中点处的 $I_h(x)$ 与 $f(x)$ 的值，并估计误差。

3.1 Answer

利用 python 进行数值求解，可以得到：

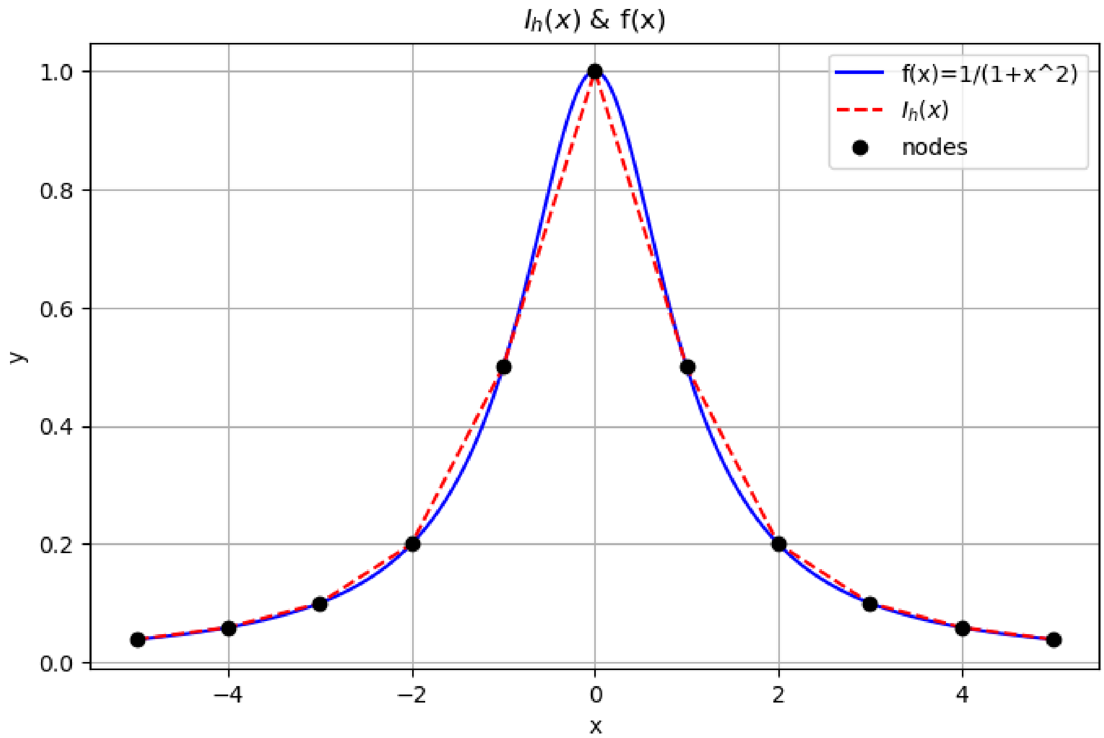
[12]:
```
import numpy as np
import matplotlib.pyplot as plt
# 定义 f(x)
def f(x):
    return 1/(1+x**2)
# 区间 [-5,5] 取 n=10 等距子区间 (共 11 个节点)
a, b = -5, 5
n = 10
x_nodes = np.linspace(a, b, n+1)   # 节点
f_nodes = f(x_nodes)
# 每个子区间的中点
x_mid = (x_nodes[:-1] + x_nodes[1:]) / 2
# 分段线性插值 I_h 在每个子区间上是直线,
# 因此在中点处的插值值等于两个端点函数值的平均:
I_mid = (f_nodes[:-1] + f_nodes[1:]) / 2
```

```python
# 真值: f 在中点处的值
f_mid = f(x_mid)
# 计算各中点处绝对误差
error = f_mid - I_mid
# 计算在更细分的节点上的误差
x2_nodes = np.linspace(a, b, 10001)  # 细分的节点
f2_nodes = f(x2_nodes)
error2 = np.abs(f2_nodes - np.interp(x2_nodes, x_nodes, f_nodes))
# 显示结果
print("子区间编号    中点 x_mid    I_h(mid)      f(mid)        绝对误差")
for i in range(n):
    print(f"{i:4d}      {x_mid[i]:8.3f}    {I_mid[i]:12.6f}  {f_mid[i]:12.6f} ␣
 ↪{error[i]:12.6f}")

print("\nmax error: ", np.max(error2))
# 绘图比较 f(x) 与分段线性插值 I_h(x)
x_plot = np.linspace(a, b, 401)
I_h = np.interp(x_plot, x_nodes, f_nodes)
plt.figure(figsize=(8, 5))
plt.plot(x_plot, f(x_plot), 'b-', label="f(x)=1/(1+x^2)")
plt.plot(x_plot, I_h, 'r--', label="$I_h(x)$")
plt.plot(x_nodes, f_nodes, 'ko', label="nodes")
plt.xlabel("x")
plt.ylabel("y")
plt.title("$I_h(x)$ & f(x)")
plt.legend()
plt.grid(True)
plt.show()
```

```
子区间编号    中点 x_mid     I_h(mid)      f(mid)        绝对误差
   0         -4.500       0.048643      0.047059     -0.001584
   1         -3.500       0.079412      0.075472     -0.003940
   2         -2.500       0.150000      0.137931     -0.012069
   3         -1.500       0.350000      0.307692     -0.042308
   4         -0.500       0.750000      0.800000      0.050000
   5          0.500       0.750000      0.800000      0.050000
   6          1.500       0.350000      0.307692     -0.042308
   7          2.500       0.150000      0.137931     -0.012069
   8          3.500       0.079412      0.075472     -0.003940
   9          4.500       0.048643      0.047059     -0.001584

max error:  0.0674421560550783
```



理论误差限分析:

对于分段线性插值，在区间 [a,b] 上（等距节点，步长 h）如果 $f \in C^2$，那么在任一子区间内的插值误差满足

$$|f(x) - I_h(x)| \le \frac{h^2}{8} \max_{\xi \in [x_i, x_{i+1}]} |f''(\xi)|.$$

已知

$$f(x) = \frac{1}{1+x^2}, \quad x \in [-5, 5],$$

且取 n=10 个子区间，所以步长

$$h = \frac{5-(-5)}{10} = 1.$$

接下来我们需要估计 $f''(x)$ 的最大值。

先计算一阶导数:

$$f(x) = (1+x^2)^{-1} \quad \Longrightarrow \quad f'(x) = -\frac{2x}{(1+x^2)^2}.$$

再求二阶导数:

$$f''(x) = \frac{-2(1+x^2)^2 - (-2x)\cdot 2(1+x^2)\cdot 2x}{(1+x^2)^4}.$$

化简:

$$f''(x) = \frac{-2(1+x^2)^2 + 8x^2(1+x^2)}{(1+x^2)^4} = \frac{-2(1+x^2) + 8x^2}{(1+x^2)^3} = \frac{-2+6x^2}{(1+x^2)^3}.$$

函数 $f(x) = \frac{1}{1+x^2}$ 为偶函数，其二阶导数也是偶函数，因此只需要考虑 x 0。

观察表达式:

$$f''(x) = \frac{-2+6x^2}{(1+x^2)^3}.$$

可以发现，在整个区间内 $|f''(x)|$ 最大值就在 $x = 0$ 处，为 2。

带入标准误差公式，有

$$|f(x) - I_h(x)| \leq \frac{h^2}{8} \max_{x\in[-5,5]} |f''(x)| \leq \frac{1^2}{8} \cdot 2 = \frac{2}{8} = 0.25.$$

综上，对于在 $[-5, 5]$ 上取 10 个等距节点进行分段线性插值，理论上误差满足

$$\boxed{|f(x) - I_h(x)| \leq 0.25}$$

符合数值计算结果

4 19

求 $f(x) = x^4$ 在 $[a, b]$ 上的分段埃尔米特插值，并估计误差。

4.1 Answer

将区间 $[a, b]$ 等分为 n 个子区间，每个子区间的长度为

$$h = \frac{b-a}{n}.$$

令分点为

$$x_i = a + i\,h, \quad i = 0, 1, \dots, n.$$

在任一子区间 $[x_i, x_{i+1}]$ 上，构造三次多项式 $H(x)$，满足

$$\begin{aligned} H(x_i) &= f(x_i) = x_i^4, & H'(x_i) &= f'(x_i) = 4x_i^3, \\ H(x_{i+1}) &= f(x_{i+1}) = x_{i+1}^4, & H'(x_{i+1}) &= f'(x_{i+1}) = 4x_{i+1}^3. \end{aligned}$$

利用标准的两点埃尔米特插值公式，令

$$t = \frac{x - x_i}{h}, \quad 0 \leq t \leq 1,$$

则插值多项式写为

$$H(x) = f(x_i)\,H_{00}(t) + f(x_{i+1})\,H_{01}(t) + h\,f'(x_i)\,H_{10}(t) + h\,f'(x_{i+1})\,H_{11}(t),$$

其中标准基函数（与 14、16 题相同）为

$$\begin{aligned}
H_{00}(t) &= 2t^3 - 3t^2 + 1,\\
H_{10}(t) &= t^3 - 2t^2 + t,\\
H_{01}(t) &= -2t^3 + 3t^2,\\
H_{11}(t) &= t^3 - t^2.
\end{aligned}$$

误差分析:

$f \in C^4$，则在任一子区间 $[x_i, x_{i+1}]$ 内存在 $\xi \in (x_i, x_{i+1})$ 使得插值误差满足:

$$f(x) - H(x) = \frac{f^{(4)}(\xi)}{4!}\,(x - x_i)^2\,(x - x_{i+1})^2.$$

$f(x) = x^4$ 的四阶导数为

$$f^{(4)}(x) = 24,$$

因此余项公式简化为

$$f(x) - H(x) = (x - x_i)^2\,(x - x_{i+1})^2.$$

令 $x = x_i + th$（其中 $t \in [0, 1]$），则有

$$x - x_i = t\,h, \quad x - x_{i+1} = t\,h - h = h(t-1).$$

因此，误差的绝对值为

$$|f(x) - H(x)| = h^4\,t^2(1-t)^2.$$

在 $t \in [0, 1]$ 上，函数 $t^2(1-t)^2$ 的最大值出现在 $t = \frac{1}{2}$，其值为

$$\left(\frac{1}{2}\right)^2\left(1 - \frac{1}{2}\right)^2 = \frac{1}{16}.$$

因此，在每个子区间上最大误差满足

$$|f(x) - H(x)| \le \frac{h^4}{16}.$$

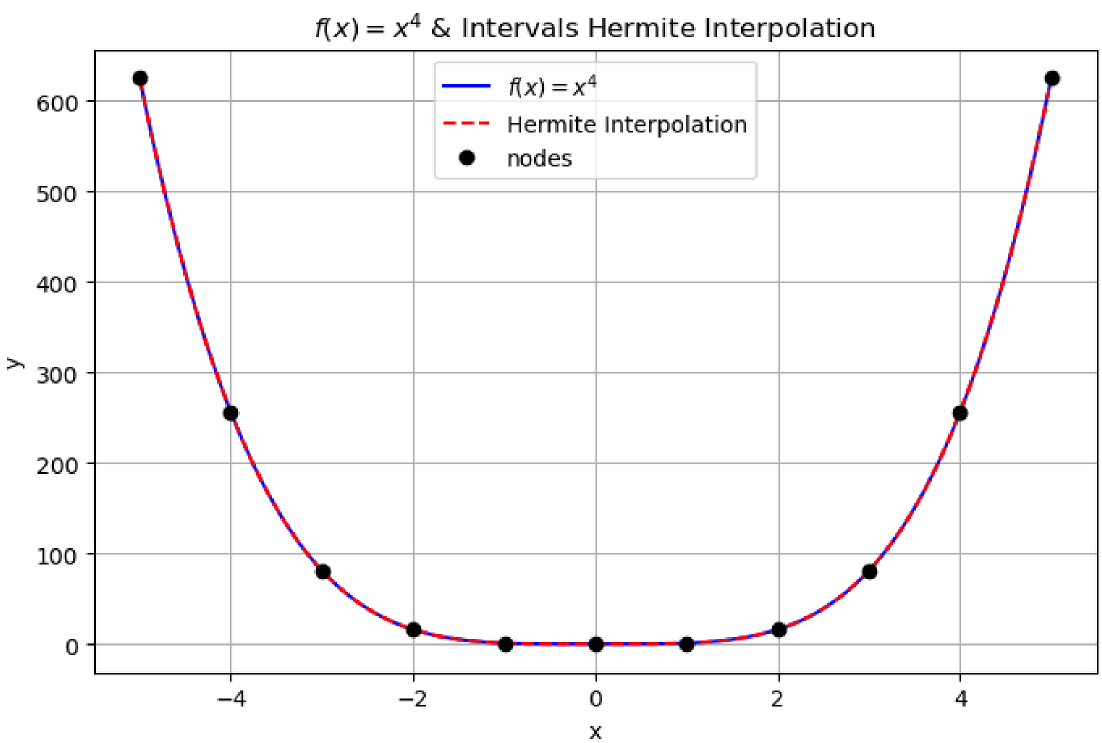
使用 python 验证结果:

[24]:
```
# 设置区间和分割数
a, b = -5, 5
n = 10
h = (b - a) / n
# 定义 f(x)=x^4 及其导数
def f(x):
    return x**4
def fp(x):
    return 4*x**3
# 定义 Hermite 基函数
def H00(t):
```

```
    return 2*t**3 - 3*t**2 + 1
def H10(t):
    return t**3 - 2*t**2 + t
def H01(t):
    return -2*t**3 + 3*t**2
def H11(t):
    return t**3 - t**2
# 修改后的分段 Hermite 插值函数，处理边界点情况
def hermite_piecewise(x_val, nodes):
    # 如果 x_val 恰好等于第一个节点，直接返回 f(x_val)
    if np.isclose(x_val, nodes[0]):
        return f(x_val)
    # 如果 x_val 恰好等于最后一个节点，归为最后一子区间
    if np.isclose(x_val, nodes[-1]):
        i = len(nodes) - 2
    else:
        # np.searchsorted(nodes, x_val) 返回第一个 >= x_val 的索引
        i = np.searchsorted(nodes, x_val) - 1
    x_i = nodes[i]
    x_ip1 = nodes[i+1]
    t = (x_val - x_i) / h
    val = (f(x_i)*H00(t) + f(x_ip1)*H01(t) +
           h*fp(x_i)*H10(t) + h*fp(x_ip1)*H11(t))
    return val
# 构造节点
nodes = np.linspace(a, b, n+1)
# 构造细网格用于评估插值函数
x_plot = np.linspace(a, b, 400)
H_vals = np.array([hermite_piecewise(x, nodes) for x in x_plot])
f_vals = f(x_plot)
# 计算误差
error = np.abs(f_vals - H_vals)
max_error = np.max(error)
# 理论误差上界，每个子区间上最大误差 <= h^4/16
error_bound = h**4 / 16
print(f"步长 h = {h}")
print(f"每个子区间理论误差上界 h^4/16 = {error_bound:.6e}")
print(f"整体最大误差 = {max_error:.6e}")
# 绘图比较真值和插值函数
plt.figure(figsize=(8,5))
plt.plot(x_plot, f_vals, 'b-', label=r'$f(x)=x^4$')
plt.plot(x_plot, H_vals, 'r--', label='Hermite Interpolation')
plt.plot(nodes, f(nodes), 'ko', label='nodes')
plt.xlabel('x')
plt.ylabel('y')
plt.title('$f(x)=x^4$ & Intervals Hermite Interpolation')
plt.legend()
```

```python
plt.grid(True)
plt.show()
```

```
步长 h = 1.0
每个子区间理论误差上界 h^4/16 = 6.250000e-02
整体最大误差 = 6.249921e-02
```



5 20

给定数据表如下

x_j	0.25	0.30	0.39	0.45	0.53
y_j	0.5000	0.5477	0.6245	0.6708	0.7280

试求三次样条插值 $S(x)$ 并满足条件：

1. $S'(0.25) = 1.0000, \quad S'(0.53) = 0.6868.$
2. $S''(0.25) = S''(0.53) = 0.$

设各区间步长

$$h_j = x_{j+1} - x_j, \quad j = 0, 1, 2, 3,$$

以及差商

$$d_j = \frac{y_{j+1} - y_j}{h_j}, \quad j = 0, 1, 2, 3.$$

记节点处的未知值为

$$m_j = S''(x_j), \quad j = 0, 1, 2, 3, 4.$$

然后标准三次样条内节点（$j = 1, 2, 3$）满足方程

$$h_{j-1}\, m_{j-1} + 2(h_{j-1} + h_j)\, m_j + h_j\, m_{j+1} = 6\Big(d_j - d_{j-1}\Big), \quad j = 1, 2, 3.$$

对于边界，分别有两种情况。

5.1 （一）两端一阶导数值已知

边界条件给定：

$$S'(x_0) = 1.0000, \quad S'(x_4) = 0.6868,$$

对应的边界方程为：

$$2h_0\, m_0 + h_0\, m_1 = 6\Big(d_0 - S'(x_0)\Big),$$

$$h_3\, m_3 + 2h_3\, m_4 = 6\Big(S'(x_4) - d_3\Big).$$

因此，构造的 5×5 系统可以写成矩阵形式

$$A_{\text{clamped}}\, \mathbf{m} = \mathbf{b}_{\text{clamped}},$$

其中

$$A_{\text{clamped}} = \begin{pmatrix} 2h_0 & h_0 & 0 & 0 & 0 \\ h_0 & 2(h_0 + h_1) & h_1 & 0 & 0 \\ 0 & h_1 & 2(h_1 + h_2) & h_2 & 0 \\ 0 & 0 & h_2 & 2(h_2 + h_3) & h_3 \\ 0 & 0 & 0 & h_3 & 2h_3 \end{pmatrix}, \quad \mathbf{b}_{\text{clamped}} = \begin{pmatrix} 6\Big(d_0 - S'(x_0)\Big) \\ 6\Big(d_1 - d_0\Big) \\ 6\Big(d_2 - d_1\Big) \\ 6\Big(d_3 - d_2\Big) \\ 6\Big(S'(x_4) - d_3\Big) \end{pmatrix}.$$

$x_0 = 0.25,\ x_4 = 0.53.$

5.2 （二）自然边界条件

自然边界条件要求

$$m_0 = m_4 = 0.$$

则只在内节点 $j = 1, 2, 3$ 有方程：

$$h_{j-1}\, m_{j-1} + 2(h_{j-1} + h_j)\, m_j + h_j\, m_{j+1} = 6\Big(d_j - d_{j-1}\Big), \quad j = 1, 2, 3.$$

记向量为 $(m_1, m_2, m_3)^T$，矩阵形式为

$$A_{\text{natural}} = \begin{pmatrix} 2(h_0+h_1) & h_1 & 0 \\ h_1 & 2(h_1+h_2) & h_2 \\ 0 & h_2 & 2(h_2+h_3) \end{pmatrix}, \quad \mathbf{b}_{\text{natural}} = \begin{pmatrix} 6(d_1-d_0) \\ 6(d_2-d_1) \\ 6(d_3-d_2) \end{pmatrix}.$$

并令 $m_0 = m_4 = 0$，得到完整的 m 向量为 $(0, m_1, m_2, m_3, 0)^T$.

利用 Python 代码分别求解这两种边界条件数值解：

[30]:
```python
# 给定数据
x = np.array([0.25, 0.30, 0.39, 0.45, 0.53])
y = np.array([0.5000, 0.5477, 0.6245, 0.6708, 0.7280])
n_nodes = len(x)

# 计算各区间步长和差商 d_j
h = np.diff(x)   # [0.05, 0.09, 0.06, 0.08]
d = np.diff(y) / h

# ----- (一) 夹紧边界条件 -----
# 已知: S'(x0)=1.0000, S'(x4)=0.6868.
A_clamped = np.zeros((5,5))
b_clamped = np.zeros(5)

# 边界左:  2h0*m0 + h0*m1 = 6(d0 - S'(x0))
A_clamped[0, 0] = 2 * h[0]
A_clamped[0, 1] = h[0]
b_clamped[0] = 6 * (d[0] - 1.0000)

# 内节点 j=1,2,3
for j in range(1, 4):
    A_clamped[j, j-1] = h[j-1]
    A_clamped[j, j]   = 2 * (h[j-1] + h[j])
    A_clamped[j, j+1] = h[j]
    b_clamped[j] = 6 * (d[j] - d[j-1])

# 边界右:  h3*m3 + 2h3*m4 = 6(S'(x4)-d3), S'(x4)=0.6868
A_clamped[4, 3] = h[3]
A_clamped[4, 4] = 2 * h[3]
b_clamped[4] = 6 * (0.6868 - d[3])

print("夹紧边界条件下的系数矩阵 A_clamped:")
print(A_clamped)
print("右侧向量 b_clamped:")
print(b_clamped)

m_clamped = np.linalg.solve(A_clamped, b_clamped)
```

```python
print("夹紧条件下求得 m = [m0, m1, m2, m3, m4]:")
print(m_clamped)

# ----- (二) 自然边界条件 -----
# 自然条件: m0 = m4 = 0, 只需解内节点 m1, m2, m3.
A_natural = np.zeros((3,3))
b_natural = np.zeros(3)

# 内节点 j=1,2,3
# j=1:
A_natural[0, 0] = 2 * (h[0] + h[1])
A_natural[0, 1] = h[1]
b_natural[0] = 6 * (d[1]-d[0])
# j=2:
A_natural[1, 0] = h[1]
A_natural[1, 1] = 2 * (h[1] + h[2])
A_natural[1, 2] = h[2]
b_natural[1] = 6 * (d[2]-d[1])
# j=3:
A_natural[2, 1] = h[2]
A_natural[2, 2] = 2 * (h[2] + h[3])
b_natural[2] = 6 * (d[3]-d[2])

print("\n自然边界条件下的系数矩阵 A_natural:")
print(A_natural)
print("右侧向量 b_natural:")
print(b_natural)

m_natural_inner = np.linalg.solve(A_natural, b_natural)
# m0 = m4 = 0, 构造完整向量
m_natural = np.hstack(([0], m_natural_inner, [0]))
print("自然条件下求得 m = [m0, m1, m2, m3, m4]:")
print(m_natural)

#--- 定义三次样条评价函数 ---
def eval_spline(x_val, x_nodes, y_nodes, m):
    # 找所在区间 [x_j, x_{j+1}]
    if x_val <= x_nodes[0]:
        j = 0
    elif x_val >= x_nodes[-1]:
        j = len(x_nodes)-2
    else:
        j = np.searchsorted(x_nodes, x_val) - 1
    x_j = x_nodes[j]
    x_j1 = x_nodes[j+1]
    h_seg = x_j1 - x_j
    A = (x_j1 - x_val) / h_seg
```

```python
    B = (x_val - x_j) / h_seg
    # 三次样条公式
    Sx = A*y_nodes[j] + B*y_nodes[j+1] + (h_seg**2)/6 * ((A**3 - A)*m[j] +
 ↪(B**3 - B)*m[j+1])
    # 一阶导数
    Sx1 = (y_nodes[j+1]-y_nodes[j]) / h_seg + (h_seg/6)*((3*B**2-1)*m[j+1] -
 ↪(3*A**2-1)*m[j])
    # 二阶导数
    Sx2 = A*m[j] + B*m[j+1]
    return Sx, Sx1, Sx2

def evaluate_spline_on_grid(x_nodes, y_nodes, m, num=400):
    x_plot = np.linspace(x_nodes[0], x_nodes[-1], num)
    S_vals = np.zeros_like(x_plot)
    S1_vals = np.zeros_like(x_plot)
    S2_vals = np.zeros_like(x_plot)
    for i, xv in enumerate(x_plot):
        S_vals[i], S1_vals[i], S2_vals[i] = eval_spline(xv, x_nodes, y_nodes, m)
    return x_plot, S_vals, S1_vals, S2_vals

# 分别评价两种边界条件下的样条
x_plot, S_clamped, S1_clamped, S2_clamped = evaluate_spline_on_grid(x, y,
 ↪m_clamped)
_, S_natural, S1_natural, S2_natural = evaluate_spline_on_grid(x, y, m_natural)

#--- 绘图: 以两列方式并排比较 ---
fig, ax = plt.subplots(3, 2, figsize=(12,10), sharex=True)
# 第一列: 夹紧条件下
ax[0,0].plot(x_plot, S_clamped, 'r-', label=r"$S(x)$")
ax[0,0].plot(x, y, 'ko', label="nodes")
ax[0,0].set_title("Clamped Spline: $S(x)$")
ax[0,0].legend(); ax[0,0].grid(True)

ax[1,0].plot(x_plot, S1_clamped, 'b-', label=r"$S'(x)$")
ax[1,0].set_title("Clamped Spline: $S'(x)$")
ax[1,0].legend(); ax[1,0].grid(True)

ax[2,0].plot(x_plot, S2_clamped, 'g-', label=r"$S''(x)$")
ax[2,0].set_title("Clamped Spline: $S''(x)$")
ax[2,0].set_xlabel("x"); ax[2,0].legend(); ax[2,0].grid(True)

# 第二列: 自然条件下
ax[0,1].plot(x_plot, S_natural, 'r-', label=r"$S(x)$")
ax[0,1].plot(x, y, 'ko', label="nodes")
ax[0,1].set_title("Natural Spline: $S(x)$")
ax[0,1].legend(); ax[0,1].grid(True)
```

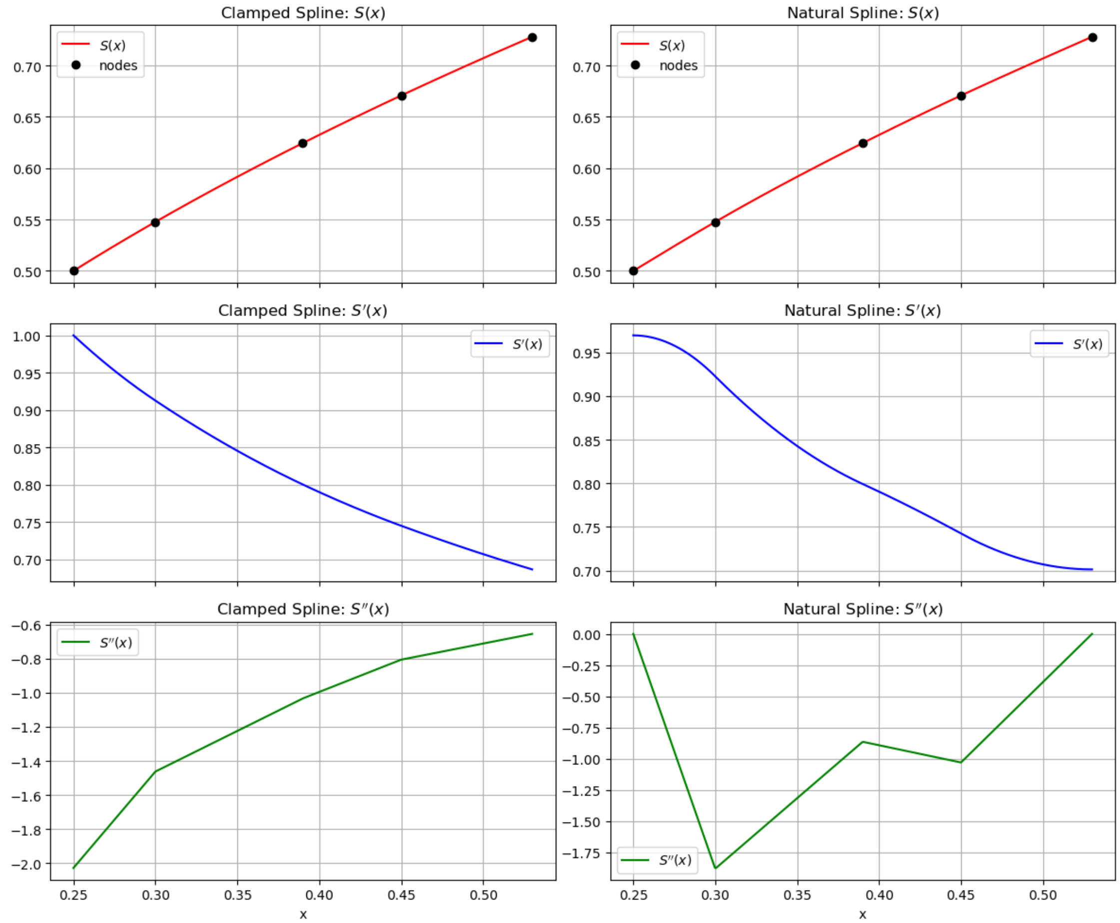
```python
ax[1,1].plot(x_plot, S1_natural, 'b-', label=r"$S'(x)$")
ax[1,1].set_title("Natural Spline: $S'(x)$")
ax[1,1].legend(); ax[1,1].grid(True)

ax[2,1].plot(x_plot, S2_natural, 'g-', label=r"$S''(x)$")
ax[2,1].set_title("Natural Spline: $S''(x)$")
ax[2,1].set_xlabel("x"); ax[2,1].legend(); ax[2,1].grid(True)

plt.tight_layout()
plt.show()
```

```
夹紧边界条件下的系数矩阵 A_clamped:
[[0.1  0.05 0.   0.   0.  ]
 [0.05 0.28 0.09 0.   0.  ]
 [0.   0.09 0.3  0.06 0.  ]
 [0.   0.   0.06 0.28 0.08]
 [0.   0.   0.   0.08 0.16]]
右侧向量 b_clamped:
[-0.276  -0.604  -0.49   -0.34   -0.1692]
夹紧条件下求得 m = [m0, m1, m2, m3, m4]:
[-2.0286295  -1.462741   -1.03334495 -0.80583043 -0.65458479]

自然边界条件下的系数矩阵 A_natural:
[[0.28 0.09 0.  ]
 [0.09 0.3  0.06]
 [0.   0.06 0.28]]
右侧向量 b_natural:
[-0.604 -0.49  -0.34 ]
自然条件下求得 m = [m0, m1, m2, m3, m4]:
[ 0.         -1.8795495  -0.86362379 -1.02922347  0.        ]
```



6 3

下列数据点的插值

x	0	1	4	9	16	25	36	49	64
y	0	1	2	3	4	5	6	7	8

可以得到平方根的近似，在区间 $[0, 64]$ 上作图。

(1) 用这 9 个点作 8 次多项式 $L_8(x)$
(2) 用三次样条（第一边界条件）程序求 $S(x)$

从得到的结果看在 $[0, 64]$ 上，哪个插值更精确？在 $[0, 1]$ 上，哪个更精确？

6.1 Answer

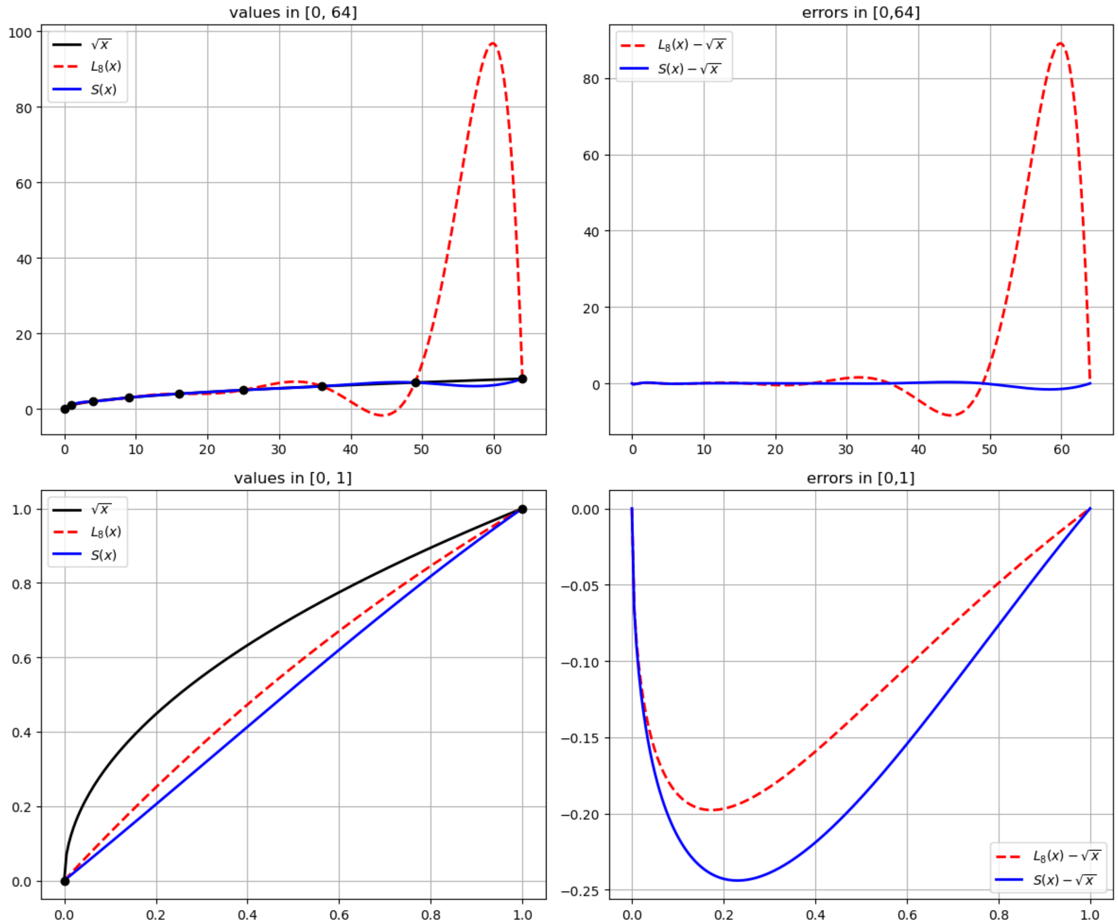
使用 python 分别用两种方法进行插值：

[33]:

```python
# 给定数据
x_nodes = np.array([0, 1, 4, 9, 16, 25, 36, 49, 64], dtype=float)
y_nodes = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8], dtype=float)
# 真值函数，即平方根（注意：由于数据正好满足 y = sqrt(x))
f = lambda x: np.sqrt(x)
# ---- (1) 全局 8 次多项式插值 L8(x) ----
# 使用 np.polyfit 求得系数（最高次系数在前）
coeffs = np.polyfit(x_nodes, y_nodes, 8)
# 构造多项式函数
L8 = np.poly1d(coeffs)
# ---- (2) 三次样条插值 S(x)（夹紧边界条件）----
# 构造步长及差商
h = np.diff(x_nodes)  # h_j, j=0,...,7 (注意：数据有 9 点，共 8 个子区间)
d = np.diff(y_nodes)/h
# 构造 5 个节点：由于数据点个数为 9，我们构 ue 构成三次样条时 m = S''(x) at all␣
↪nodes
# 注意：x_nodes 已用作节点，这里我们有 n=9，子区间数 = 8.
n = len(x_nodes)
# 构造系数矩阵 A (n x n) 和 b (n x 1)
A = np.zeros((n, n))
b_vec = np.zeros(n)
# 对内节点 j=1,...,n-2 (即 j=1,...,7)
for j in range(1, n-1):
    A[j, j-1] = h[j-1]
    A[j, j]   = 2*(h[j-1]+h[j])
    A[j, j+1] = h[j]
    b_vec[j] = 6*(d[j]-d[j-1])
# 边界：夹紧条件 S'(x0)=1, S'(x63)=S'(64)=0.6868
# 对左端点
A[0, 0] = 2*h[0]
A[0, 1] = h[0]
b_vec[0] = 6*(d[0]-1.0)
# 对右端点
A[-1, -2] = h[-1]
A[-1, -1] = 2*h[-1]
b_vec[-1] = 6*(0.6868-d[-1])
# 求解 m 向量
m = np.linalg.solve(A, b_vec)
# 定义样条评价函数（前面已经给出公式）
def eval_spline(x_val, x_nodes, y_nodes, m):
    if x_val <= x_nodes[0]:
        j = 0
    elif x_val >= x_nodes[-1]:
        j = len(x_nodes)-2
    else:
        j = np.searchsorted(x_nodes, x_val) - 1
    xj = x_nodes[j]
```

```python
    xj1 = x_nodes[j+1]
    hj = xj1 - xj
    A_coef = (xj1 - x_val)/hj
    B_coef = (x_val - xj)/hj
    Sx = A_coef*y_nodes[j] + B_coef*y_nodes[j+1] + (hj**2)/6*(
 ↪(A_coef**3-A_coef)*m[j] + (B_coef**3-B_coef)*m[j+1] )
    return Sx
def evaluate_spline(x_nodes, y_nodes, m, x_plot):
    S_vals = np.array([eval_spline(xv, x_nodes, y_nodes, m) for xv in x_plot])
    return S_vals
# 构造评价网格
x_full = np.linspace(x_nodes[0], x_nodes[-1], 400)
x_small = np.linspace(0, 1, 200)  # [0,1] 上的子区间
L8_full = L8(x_full)
S_full  = evaluate_spline(x_nodes, y_nodes, m, x_full)
f_full  = f(x_full)
L8_small = L8(x_small)
S_small  = evaluate_spline(x_nodes, y_nodes, m, x_small)
f_small  = f(x_small)
# ---- 绘图 ----
fig, axs = plt.subplots(2, 2, figsize=(12,10))
# 整个区间 [0,64]
axs[0,0].plot(x_full, f_full, 'k-', lw=2, label=r"$\sqrt{x}$")
axs[0,0].plot(x_full, L8_full, 'r--', lw=2, label=r"$L_8(x)$")
axs[0,0].plot(x_full, S_full, 'b-', lw=2, label=r"$S(x)$")
axs[0,0].plot(x_nodes, y_nodes, 'ko', ms=6)
axs[0,0].set_title("values in [0, 64]")
axs[0,0].legend(); axs[0,0].grid(True)
# 放大观察 L8 和 S 在 [0,64] 上的误差
axs[0,1].plot(x_full, L8_full - f_full, 'r--', lw=2, label=r"$L_8(x)-\sqrt{x}$")
axs[0,1].plot(x_full, S_full - f_full, 'b-', lw=2, label=r"$S(x)-\sqrt{x}$")
axs[0,1].set_title("errors in [0,64]")
axs[0,1].legend(); axs[0,1].grid(True)
# 局部区间 [0,1]
axs[1,0].plot(x_small, f_small, 'k-', lw=2, label=r"$\sqrt{x}$")
axs[1,0].plot(x_small, L8_small, 'r--', lw=2, label=r"$L_8(x)$")
axs[1,0].plot(x_small, S_small, 'b-', lw=2, label=r"$S(x)$")
axs[1,0].plot([0,1], [f(0), f(1)], 'ko', ms=6)
axs[1,0].set_title("values in [0, 1]")
axs[1,0].legend(); axs[1,0].grid(True)
# 误差放大 [0,1]
axs[1,1].plot(x_small, L8_small - f_small, 'r--', lw=2,
 ↪label=r"$L_8(x)-\sqrt{x}$")
axs[1,1].plot(x_small, S_small - f_small, 'b-', lw=2, label=r"$S(x)-\sqrt{x}$")
axs[1,1].set_title("errors in [0,1]")
axs[1,1].legend(); axs[1,1].grid(True)
plt.tight_layout()
```

```
plt.show()
```



从 python 的计算结果可以看出:

1. 在区间 $[0, 64]$ 上，$S(x)$ 的误差明显更小。
2. 在区间 $[0, 1]$ 上，$L_8(x)$ 的误差略小于 $S(x)$。