数值分析上机实验

插值法

1 问题描述

本次实验的目的是通过上机实验来对 Lagrange 基函数插值、线性插值法和三次样条插值法的曲线拟合效果进行实际测试。

Lagrange 基函数插值法属于多项式插值。所谓多项式插值，就是给定 $\{x_i\}_{i=0}^n \subset [a,b]$ 是 $n+1$ 个互不相同节点，对于任何给定的一组函数值 $\{y_i\}_{i=0}^n$ 要找到一个多项式函数 $p_n(x) \in \mathcal{P}_n$，满足插值条件 $p_n(x_i) = y_i$. Lagrange 基函数插值法给出了一种找到插值多项式的简单方法，即考虑以下的多项式函数：

$$\ell_k(x) = \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i}, \quad k = 0, 1, \cdots, n, \tag{1}$$

容易验证 ℓ_k 是线性无关的 n 次多项式，而且满足：

$$\ell_k(x_m) = \delta_{km} = \begin{cases} 0, & k \neq m \\ 1, & k = m \end{cases} \qquad k, m = 0, 1, \cdots, n. \tag{2}$$

因此，$p_n(x) := \sum_{k=0}^n y_k \ell_k(x) \in \mathcal{P}_n$ 就是满足插值条件的多项式函数.

关于多项式插值的误差，有如下结果：

定理 1.1 (插值误差估计). *$f : [a,b] \to \mathbb{R}$ 是 $n+1$ 次连续可微的函数，p_n 是在 $n+1$ 个不同节点上的插值多项式，则插值余项满足：*

$$R_n(f)(x) = \frac{f^{(n+1)}(\eta)}{(n+1)!} \prod_{j=0}^{n} (x - x_j), \quad \forall x \in [a,b], \tag{3}$$

其中 $\eta = \eta(x) \in (a,b)$.

从这个结果可以看出，如果 f 的 $n+1$ 阶导数一致有界，也就是 $\left\|f^{(n+1)}\right\|_\infty \leq M$，那么此时插值多项式 $L_n(f) \rightrightarrows f$（按照无穷范数收敛到 f）. 但是，如果 f 各阶导数不是一致有界，则无法做到一致收敛，甚至可能在一些区域有发散的现象（即插值多项式的次数越高，收敛行为反而越差）. 事实上，任给 $[a,b]$ 上一组节点，总会存在一个连续函数 f 使得其 Lagrange 型插值多项式 $L_n(f)$ 不收敛于 f.

为了克服以上高阶病态问题，分段低次插值是一种解决方案。所谓分段低次插值，就是找到一个函数 $I_h(x)$，满足 $I_h \in C[a,b]$ 以及 $I_h(x_j) = f(x_j)$，同时满足 I_h 限制在两个相邻节点的区间 $[x_k, x_{k+1}]$ 是低次多项式. 对于分段线性函数，我们要求 $I_1^h|_{[x_k,x_{k+1}]}$ 是一次多项式。

分段线性插值函数同样可以用 Lagrange 型的基函数来构造，这里取的基函数是下面一些所谓的 “帽型函数”：

$$\begin{aligned} &\phi_0(x) = \begin{cases} 0, & x_1 \le x; \\ \frac{x-x_1}{x_0-x_1}, & x_0 \le x \le x_1; \end{cases} \qquad \phi_n(x) = \begin{cases} 0, & x \le x_{n-1}; \\ \frac{x-x_{n-1}}{x_n-x_{n-1}}, & x_{n-1} \le x \le x_n; \end{cases} \\ &\phi_j(x) = \begin{cases} 0, & x \le x_{j-1} \quad \text{or} \quad x \ge x_{j+1}; \\ \frac{x-x_{j-1}}{x_j-x_{j-1}}, & x_{j-1} \le x \le x_j; \\ \frac{x-x_{j+1}}{x_j-x_{j+1}}, & x_j \le x \le x_{j+1}; \end{cases} \qquad j = 1, 2, \cdots, n-1. \end{aligned} \tag{4}$$

此时 $\{\phi_i\}$ 仍然满足 $\phi_i(x_j) = \delta_{ij}$，所以分段线性插值函数 $I_h(x)$ 可以写成

$$I_h(x) = \sum_{j=0}^{n} f(x_j)\phi_j(x). \tag{5}$$

但是，用分段低次插值函数往往只能得到光滑性很差的函数. 例如分段线性插值函数只有 C^0 连续性. 而如果采用高次分段插值，则其阶数会增长非常快（例如如果要求插值函数具有 C^2 光滑性，则至少需要五阶多项式，此时可能又会容易出现高阶病态问题）. 一个解决方案是采用样条插值，三次样条插值采用 Hermite 型插值函数，满足该函数在每相邻两个节点上的限制是一个三次多项式，而且函数和导数满足一定的连续性使其整体具有 C^2 光滑性。具体来说：

$$S_3^h(x) = \sum_{j=0}^{n} [f_j \alpha_j(x) + m_j \beta_j(x)], \tag{6}$$

其中 α_j 和 β_j 分别 Hermite 型的两个插值基函数，定义为：

$$\begin{aligned} \alpha_j(x) &= \begin{cases} \left(1 + 2\frac{x-x_j}{x_{j-1}-x_j}\right)\left(\frac{x-x_{j-1}}{x_j-x_{j-1}}\right)^2, & x_{j-1} \le x \le x_j (j = 0 \text{ 时略去}); \\ \left(1 + 2\frac{x-x_j}{x_{j+1}-x_j}\right)\left(\frac{x-x_{j+1}}{x_j-x_{j+1}}\right)^2, & x_j \le x \le x_{j+1} (j = n \text{ 时略去}); \\ 0, & \text{其他情形}; \end{cases} \\ \beta_j(x) &= \begin{cases} (x - x_j)\left(\frac{x-x_{j-1}}{x_j-x_{j-1}}\right)^2, & x_{j-1} \le x \le x_j (j = 0 \text{ 时略去}); \\ (x - x_j)\left(\frac{x-x_{j+1}}{x_j-x_{j+1}}\right)^2, & x_j \le x \le x_{j+1} (j = n \text{ 时略去}); \\ 0, & \text{其他情形}; \end{cases} \end{aligned} \tag{7}$$

值得注意的是，加上连续性条件和插值条件后，三次样条插值仍需两个条件才能确定 f_j 和 m_j 的取值. 这附加的两个条件往往是边界条件. 我们可以根据问题的需要人为地选择使用什么类型的边界条件. 在本次实验中，我们选用自然边界条件，也就是断点处二阶导数为 0.

2 实验内容

本次实验中，我们对函数 $f(x) = \frac{1}{1+25x^2}$ 在 $[-1,1]$ 上取等距节点 $x_j = -1 + \frac{2j}{n}, j = 0, 1, \cdots, n$，取适当的 n（例如 $n = 6, 10, 16$ 等），求出其 n 次 Lagrange 插值多项式 $L_n(x)$，分段线性插值多项式 $I_1^h(x)$ 以及三次样条插值函数 $S_3^h(x)$（自然边界条件），并对结果进行比较。

我们首先根据公式 (1) 定义 Lagrange 型的插值基函数，并利用这些插值基函数来构造 Lagrange 插值函数.

```
def func(x):
    return 1/(25*x**2 + 1)

def lk(x, nodes, n, k):
    """lagrange intercep. basis function l_k = \prod_{i=0,i\ne k}^n (x-x_i)/(x_k-x_i)"""
    nodes = [-1+2*j/n for j in range(n+1)]
    prod = np.prod([(x-nodes[i])/(nodes[k]-nodes[i]) for i in range(len(nodes)) if i != k])
    return prod

def Ln(n, x):
    """p_n(x) = \sum_{k=0}^n y_k l_k(x)"""
    nodes = [-1+2*j/n for j in range(n+1)]
    lk_basis = np.array([lk(x, nodes, n, k) for k in range(len(nodes))])
    yk_list = np.array([func(nodes[k]) for k in range(len(nodes))])
    p_n_x = lk_basis @ yk_list
    return p_n_x
```

随后，我们根据公式 (4) 构造分段线性插值基函数 $\phi_k(x)$，并利用它构造分段线性插值函数 $I_1^h(x)$.

```
def phik(x, nodes, n, k):
    """linear intercep. basis function phi_k"""
    if k == 0:
        if x > nodes[1]:
            return 0
        if x <= nodes[1]:
            return (x-nodes[1])/(nodes[0]-nodes[1])
    if k == len(nodes)-1:
        if x < nodes[-2]:
            return 0
        if x >= nodes[-2]:
            return (x-nodes[-2])/(nodes[-1]-nodes[-2])
    else:
```

```
        if (x <= nodes[k-1]) or (x >= nodes[k+1]):
            return 0;
        if (nodes[k-1] < x < nodes[k]):
            return (x-nodes[k-1])/(nodes[k]-nodes[k-1])
        if (nodes[k] <= x < nodes[k+1]):
            return (x-nodes[k+1])/(nodes[k]-nodes[k+1])

def In(n, x):
    """I_n(x) = \sum_{k=0}^n y_k \phi_k(x)"""
    nodes = [-1+2*j/n for j in range(n+1)]
    phik_basis = np.array([phik(x, nodes, n, k) for k in range(len(nodes))])
    yk_list = np.array([func(nodes[k]) for k in range(len(nodes))])
    I_n_x = phik_basis @ yk_list
    return I_n_x
```

最后，我们来处理三次样条插值。三次样条插值法需要用到 Hermite 型的基函数 α_k 和 β_k，按照公式 (7) 定义如下：

```
def hermite_alpha(x, nodes, n, k):
    """hermite alpha basis function"""
    if k == 0:
        if x > nodes[1]:
            return 0
        if x <= nodes[1]:
            return (1+2*(x-nodes[0])/(nodes[1]-nodes[0]))*((x-nodes[1])/(nodes[0]-nodes[1]))**2
    if k == len(nodes)-1:
        if x < nodes[-2]:
            return 0
        if x >= nodes[-2]:
            return (1+2*(x-nodes[-1])/(nodes[-2]-nodes[-1]))*((x-nodes[-2])/(nodes[-1]-nodes[-2]))**2
    else:
        if (x <= nodes[k-1]) or (x >= nodes[k+1]):
            return 0;
        if (nodes[k-1] < x < nodes[k]):
            return (1+2*(x-nodes[k])/(nodes[k-1]-nodes[k]))*((x-nodes[k-1])/(nodes[k]-nodes[k-1]))**2
        if (nodes[k] <= x < nodes[k+1]):
            return (1+2*(x-nodes[k])/(nodes[k+1]-nodes[k]))*((x-nodes[k+1])/(nodes[k]-nodes[k+1]))**2

def hermite_beta(x, nodes, n, k):
```

```
    """hermite beta basis function"""
    if k == 0:
        if x > nodes[1]:
            return 0
        if x <= nodes[1]:
            return (x-nodes[0])*((x-nodes[1])/(nodes[0]-nodes[1]))**2
    if k == len(nodes)-1:
        if x < nodes[-2]:
            return 0
        if x >= nodes[-2]:
            return (x-nodes[-1])*((x-nodes[-2])/(nodes[-1]-nodes[-2]))**2
    else:
        if (x <= nodes[k-1]) or (x >= nodes[k+1]):
            return 0;
        if (nodes[k-1] < x < nodes[k]):
            return (x-nodes[k])*((x-nodes[k-1])/(nodes[k]-nodes[k-1]))**2
        if (nodes[k] <= x < nodes[k+1]):
            return (x-nodes[k])*((x-nodes[k+1])/(nodes[k]-nodes[k+1]))**2
```

为了求解系数，我们用上二阶导数的连续条件，也就是需要求解所谓“转角方程”，这是一个三对角矩阵的线性系统：

$$\begin{pmatrix} 2 & \mu_1 & & & \\ \lambda_2 & 2 & \mu_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \mu_{n-2} \\ & & & \lambda_{n-1} & 2 \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ \vdots \\ m_{n-2} \\ m_{n-1} \end{pmatrix} = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_{n-2} \\ g_{n-1} \end{pmatrix} \tag{8}$$

这里我们采用自然边界条件也就是满足 $S''(x_0) = S''(x_n) = 0$，即 $m_0 = m_n = 0$. 其中：

$$\lambda_j = \frac{h_{j+1}}{h_j + h_{j+1}}, \quad \mu_j = 1 - \lambda_j, \quad g_j = 3(\lambda_j f[x_{j-1}, x_j] + \mu_j f[x_j, x_{j+1}]), \tag{9}$$

这里 h_j 是分划的区间长度，也就是 $h_j = x_j - x_{j-1}, j = 1, \cdots, n$.

我们定义函数 `AngleEquation` 用来生成转角线性系统中的矩阵和向量，并定义 `spline_ms` 来求解线性系统从而得到系数 m_j：

```
def AngleEquation(nodes):
    """get the tridiagonal angle equation matrix and the gvector"""
    hlist = [nodes[i] - nodes[i-1] for i in range(1,len(nodes))]
    lambda_list = [hlist[i+1]/(hlist[i]+hlist[i+1]) for i in range(len(nodes)-2)]
    mu_list = [1-lambda_ for lambda_ in lambda_list]
    mat = np.zeros([len(nodes)-2, len(nodes)-2])
    mat += np.diag([2]*(len(nodes)-2))
```

```
    for i in range(len(nodes)-3):
        mat[i,i+1] = mu_list[i]
        mat[i+1,i] = lambda_list[i+1]
    gvector = [lambda_list[i]*(func(nodes[i+1])-func(nodes[i]))/(nodes[i+1]-nodes[i
    ]) + \
               mu_list[i]*(func(nodes[i+2])-func(nodes[i+1]))/(nodes[i+2]-nodes[i
    +1]) for i in range(len(nodes)-2)]
    return mat, gvector

def spline_ms(nodes):
    """get the coefficgients of hermite beta functions"""
    mat, gvector = AngleEquation(nodes)
    ms = np.linalg.solve(mat, gvector)
    ms = np.insert(ms, 0, 0)
    ms = np.insert(ms, len(ms), 0)
    return ms
```

有了以上准备，我们就可以构建三次样条插值函数了：

```
def Sn3(n, x):
    """S_n(x) = \sum_{k=0}^n y_k \alpha_k(x) + m_k \beta_k(x)"""
    nodes = [-1+2*j/n for j in range(n+1)]
    alphak_basis = np.array([hermite_alpha(x, nodes, n, k) for k in range(len(nodes)
    )])
    betak_basis = np.array([hermite_beta(x, nodes, n, k) for k in range(len(nodes))
    ])
    yk_list = np.array([func(nodes[k]) for k in range(len(nodes))])
    mk_list = spline_ms(nodes)
    S_n_3_x = alphak_basis @ yk_list + betak_basis @ mk_list
    return S_n_3_x
```

3 实验结果及讨论

我们对 Lagrange 型插值取 $n = 6, 10, 15$，对分段线性和三次样条插值分别取 $n = 6, 10, 16, 20$，得到的结果如图1所示.

从图像中可以看出，Lagrange 插值函数在大约 $0.726 \le |x| \le 1$ 时是发散的，当 n 越来越大时，事实上中间部分收敛得越来越好，但是靠近 $x = \pm 1$ 附近的振荡程度却随着阶数的增大急剧增大. 这是因为，虽然 f 的光滑性很好（$f \in C^\infty[-1, 1]$），但是 $f^{(n)}(x)$ 在 $[-1, 1]$ 上却不是一致有界的，事实上 $\left\|f^{(n)}\right\|_\infty \to +\infty$. 这是多项式插值中高阶病态的一个典型例子，称为 Runge 现象，是由 Runge 于 1901 年首先发现的.

对于分段线性插值以及样条插值，从图像上看，的确是随着阶数的增加，插值函数有一致逼

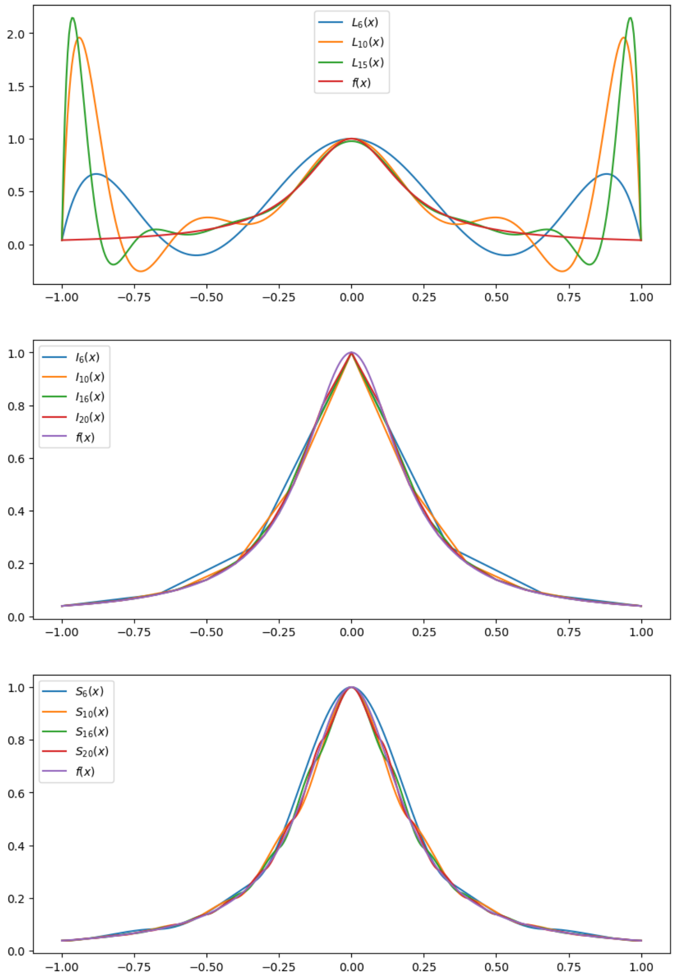

图 1: $L_n(x), I_1^h(x), S_3^h(x)$ 以及 $f(x)$ 的图像

近于 $f(x)$ 的趋势，且的确是阶数越高，逼近效果越好。事实上，我们可以做出 $n = 6, 10, 16, 20, 100$ 的误差函数 $f(x) - I_1^n(x)$ 以及 $f(x) - S_3^n(x)$ 的图像，如图2所示：从图中可以看出，当 $n = 20$ 时，

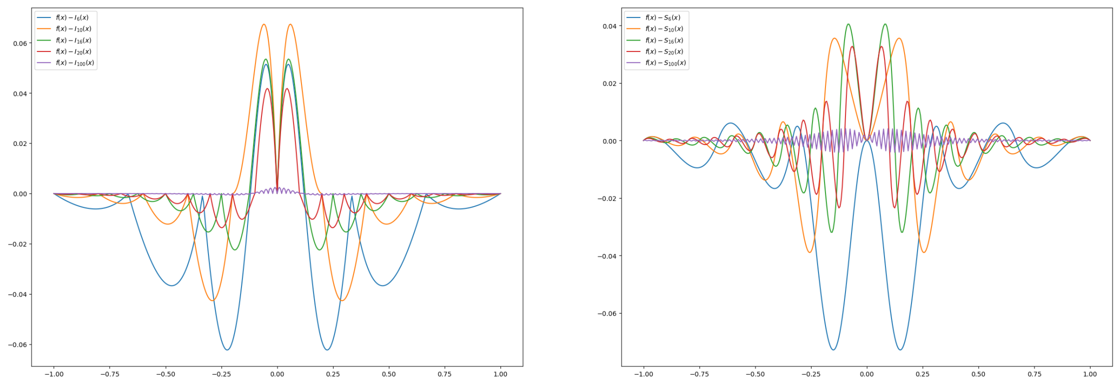

图 2: 误差函数

两种方法的无穷范数下的误差 $\|f - I_n\|_\infty$ 或 $\|f - S_n\|_\infty$ 就都可以控制在 0.04 之内. 取 $n = 100$ 时，两种方法误差都已经非常小. 而且，在图2中可看出，分段线性插值的一致收敛速度快于三次

样条插值。但是，从图1可以看出，三次样条函数的光滑性的确非常好（C^2 光滑，而分段样条插值只有 C^0 连续性）. 在实际工作中，我们可以针对具体问题的要求，选择适合进行插值的格式.