数值分析实习作业

Dseid

本实训作业满分，但仅供参考

2022 年 5 月 8 日

目录

1 任务一

1.1 等距节点 *Lagrange* 插值

1.1.1 算法描述

拉格朗日插值法根据 n+1 个点构造插值基函数构造 n 次多项式对原函数拟合，本质上就是 n 次多项式插值。倘若选择的多项式次数过高，容易出现龙格现象。如1两侧所示出现了偏离真实函数值较大的情况。此处选择的插值点为：-5，4,...,4，5。多项式次数为 10 次

1.1.2 图像展示

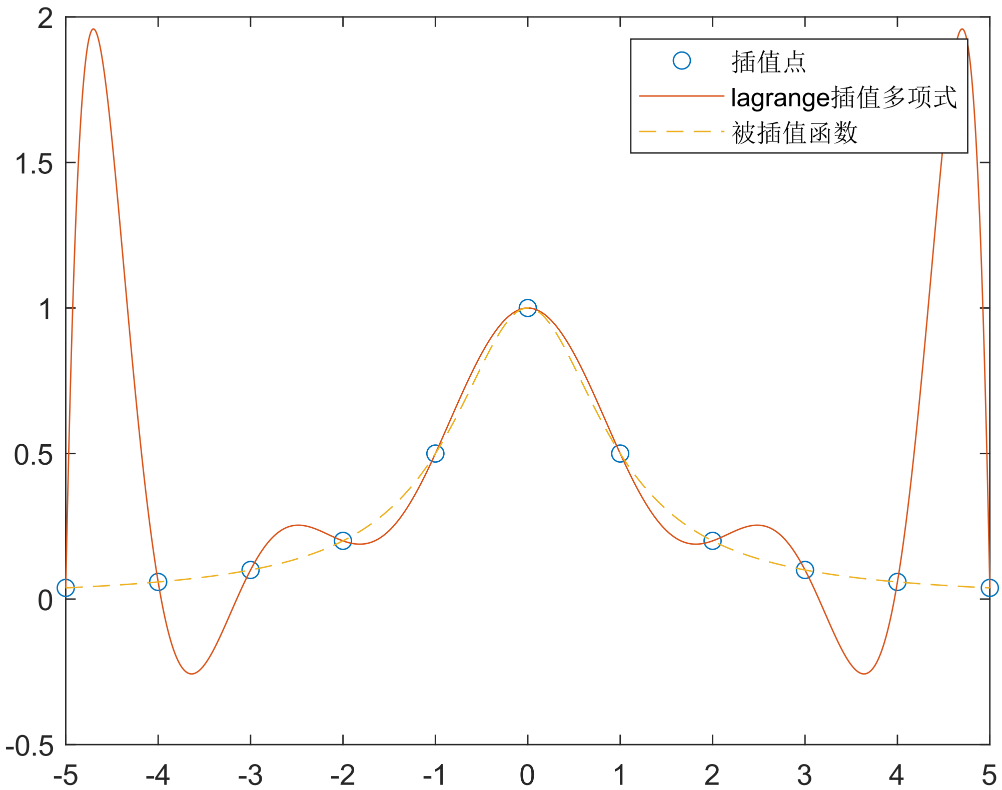

图 1: 等距节点 *Lagrange* 插值

1.1.3 误差计算

采用区间 [-5,5] 上 101 个等距分布 $\mathbf{z_j} = -\mathbf{5} + \frac{\mathbf{j}}{\mathbf{10}}, j = 0, 1, ..., 100$ 处的误差绝对值的平均值来衡量。基于 *Matlab* 计算得绝对误差为 0.2858

1.2 *Chebyshev* 节点上的 *Lagrange* 插值

1.2.1 算法描述

切比雪夫多项式 $T_n(x)$ 在区间 $[-1,1]$ 上有 n 个零点

$$x_k = \cos\frac{2k-1}{2n}\pi, \quad k=1,2,\cdots,n$$

和 $n+1$ 个极值点 (包括端点)

$$x_k = \cos\frac{k\pi}{n}, \quad k=0,1,\cdots,n.$$

这两组点称为切比雪夫点, 它们在插值中有重要作用。
在所有首项系数为 1 的 n 次多项式集合 $\tilde{H}_n$ 中

$$\left\|\tilde{T_n}\right\|_\infty = \min_{P\in\tilde{H}_n}\|P(x)\|_\infty$$

设插值节点 $x_0,x_1,\cdots,x_n$ 为切比雪夫多项式 $T_{n+1}(x)$ 的零点, 被插函数 $f\in C^{n+1}[-1,1], L_n(x)$ 为相应的插值多项式, 则

$$\max_{-1\leqslant x\leqslant 1}|f(x)-L_n(x)| \leqslant \frac{1}{2^n(n+1)!}\left\|f^{(n+1)}(x)\right\|_\infty.$$

对于一般区间 $[a,b]$ 上的插值只要利用变换 (2.14) 式则可得到相应结果, 此时插值节点为

$$x_k = \frac{b-a}{2}\cos\frac{2k+1}{2(n+1)}\pi + \frac{a+b}{2}, \quad k=0,1,\cdots,n.$$

由于高次插值出现龙格现象, 一般 $L_n(x)$ 不收敛于 $f(x)$, 因此等距节点高次插值并不适用. 但若用切比雪夫多项式零点插值却可避免龙格现象, 可保证整个区间上收敛. 如图2两侧拉格朗日插值多项式和原函数拟合的很好。

1.2.2 图像展示

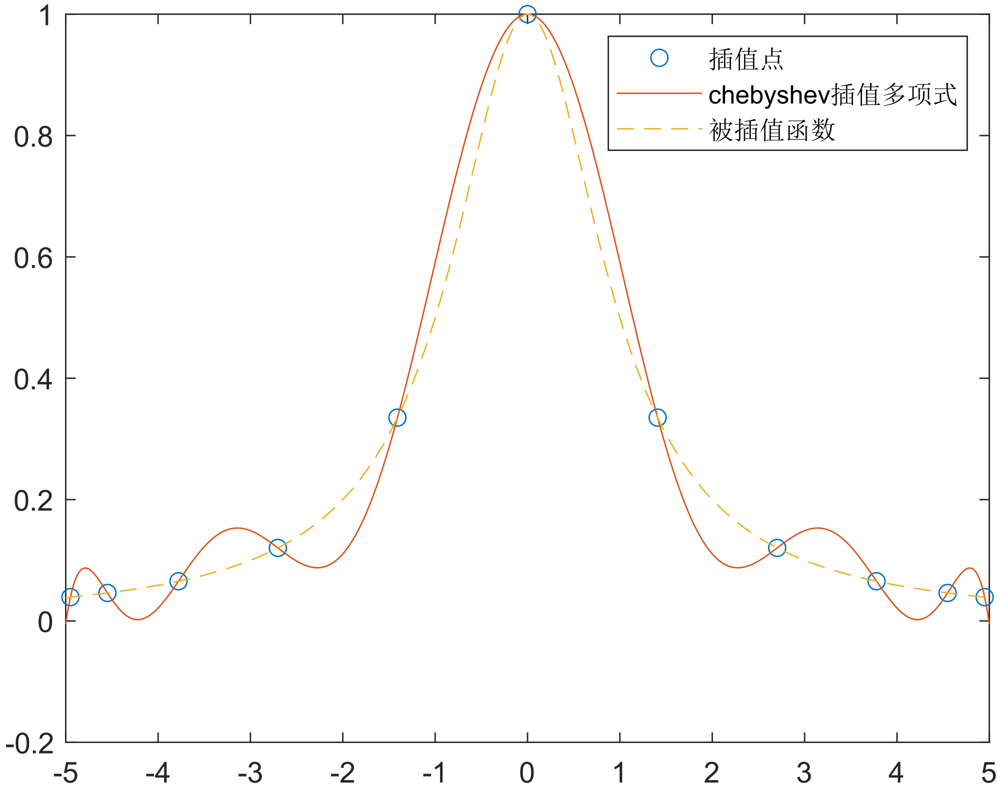

图 2: Chebyshev 节点 *Lagrange* 插值

1.2.3 误差计算

采用区间 [-5,5] 上 101 个等距分布 $\mathbf{z_j} = \mathbf{-5} + \frac{\mathbf{j}}{\mathbf{10}}, j = 0, 1, ..., 100$ 处的误差绝对值的平均值来衡量。基于 *Matlab* 计算得绝对误差为 0.0487, 插值点数量和之前的等距节点数量相同，然而绝对误差却减少了十倍左右。

1.3 分段线性插值

1.3.1 算法描述

分段线性插值就是通过插值点用折线段连接起来逼近 $f(x)$. 设已知节点 $a = x_0 < x_1 < \cdots < x_n = b$ 上的函数值 $f_0, f_1, \cdots, f_n$, 记 $h_k = x_{k+1} - x_k, h = \max\limits_k h_k$, 求一折线函数 $I_h(x)$ 满足:

(1) $I_h(x) \in C[a, b]$;

(2) $I_h(x_k) = f_k (k = 0, 1, \cdots, n)$;

(3) $I_h(x)$ 在每个小区间 $[x_k, x_{k+1}]$ 上是线性函数. 则称 $I_h(x)$ 为分段线性插值函数. 如图3所示

1.3.2 图像展示

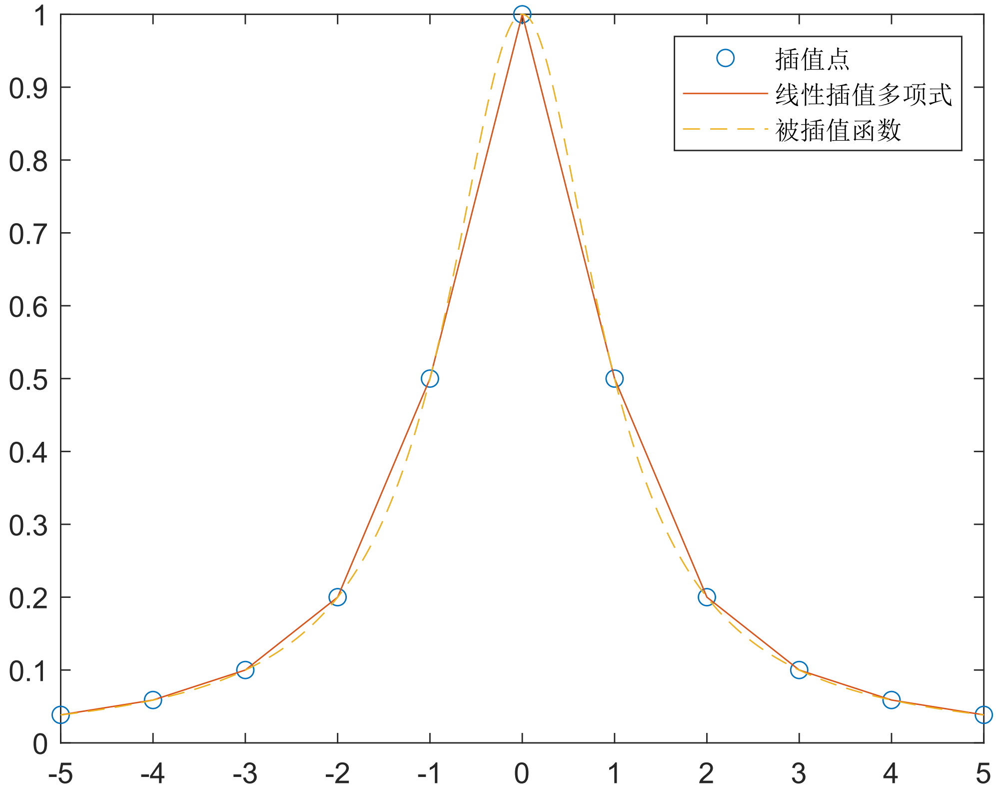

图 3: 分段线性插值插值

1.3.3 误差计算

采用区间 [-5,5] 上 101 个等距分布 $\mathbf{z_j = -5 + \frac{j}{10}}, j = 0, 1, ..., 100$ 处的误差绝对值的平均值来衡量。基于 *Matlab* 计算得绝对误差为 0.0148, 插值点数量和之前的等距节点数量相同，然而绝对误差相比之前两种多项式插值方法都要更少。这说明在节点数量相同的情况下，相比于多项式插值，线性插值更适合用来对原函数进行拟合。

1.4 三次样条插值

1.4.1 算法描述

若函数 $S(x) \in C^2[a, b]$, 且在每个小区间 $[x_j, x_{j+1}]$ 上是三次多项式, 其中 $a = x_0 < x_1 < \cdots < x_n = b$ 是给定节点, 则称 $S(x)$ 是节点 $x_0, x_1, \cdots, x_n$ 上的三次样条函数. 若在节点 x_j 上给定函数值 $y_j = f(x_j)\,(j = 0, 1, \cdots, n)$, 并成立

$$S(x_j) = y_j, \quad j = 0, 1, \cdots, n,$$

则称 $S(x)$ 为三次样条插值函数。从定义知要求出 $S(x)$, 在每个小区间 $[x_j,x_{j+1}]$ 上要确定 4 个待定系数, 而共有 n 个小区间, 故应确定 $4n$ 个参数. 根据 $S(x)$ 在 $[a,b]$ 上二阶导数连续, 在节点 $x_j(j=1,2,\cdots,n-1)$ 处应满足连续性条件

$$S\left(x_j-0\right)=S\left(x_j+0\right)$$

$$S'\left(x_j-0\right)=S'\left(x_j+0\right)$$

$$S''\left(x_j-0\right)=S''\left(x_j+0\right)$$

这里共有 $3n-3$ 个条件, 再加上 $S(x)$ 满足插值条件, 共有 $4n-2$ 个条件, 因此还需要加上 2 个条件才能确定 $S(x)$. 通常可在区间 $[\cdot a,b]$ 的端点 $a=x_0, b=x_n$ 上各加一个条件 (称为边界条件), 可根据实际问题的要求给定. 本次插值使用第一类边界条件: 已知两端的一阶导数值
即

$$S'\left(x_0\right)=f_0', \quad S'\left(x_n\right)=f_n'.$$

相比分段线性插值在区间的端点不连续，三次样条插值可以保证二阶导数值连续，图4中直观的诠释了三次样条插值这一突出特性。

1.4.2 图像展示

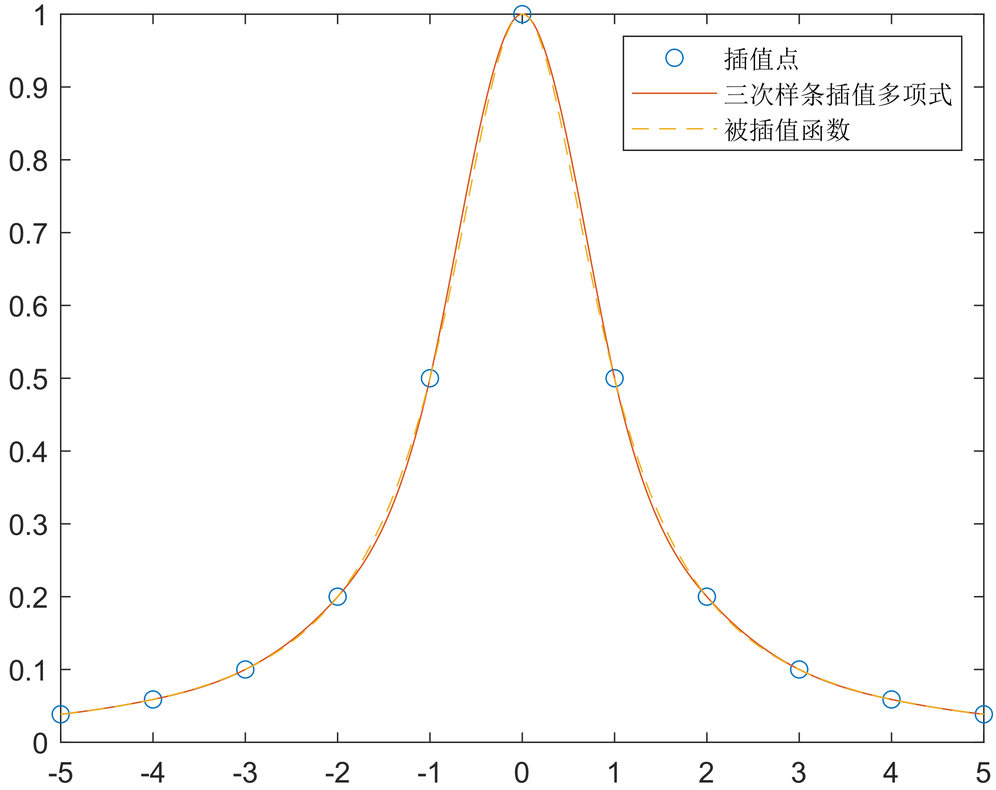

图 4: 三次样条插值

1.4.3 误差计算

设 $f(x) \in C^4[a,b], S(x)$ 为满足第一种或第二种边界条件的三次样条函数, 令 $h = \max_{0\leqslant i\leqslant n-1} h_i, h_i = x_{i+1} - x_i (i = 0, 1, \cdots, n-1)$, 则有估计式:

$$\max_{a\leqslant x\leqslant b} \left|f^{(k)}(x) - S^{(k)}(x)\right| \leqslant C_k \max_{a\leqslant x\leqslant b} \left|f^{(4)}(x)\right| h^{4-k}, \quad k = 0, 1, 2$$

其中 $C_0 = \frac{5}{384}, C_1 = \frac{1}{24}, C_2 = \frac{3}{8}$.

采用区间 [-5,5] 上 101 个等距分布 $\mathbf{z_j = -5 + \frac{j}{10}}, j = 0, 1, ..., 100$ 处的误差绝对值的平均值来衡量。基于 *Matlab* 计算得绝对误差为 0.0040, 插值点数量和之前的等距节点数量相同，然而绝对误差相比之前三种方法都要更少，从图像上也可以直观的看出拟合的优度。这说明在节点数量相同的情况下，相比于多项式插值、线性插值, 三次样条插值很好的保留了原函数的光滑特性，达到了更好的拟合优度。

1.5 *Legendre* 多项式最佳平方逼近

1.5.1 算法描述

如果

$$
\begin{aligned}
\|f(x)-P^*(x)\|_2^2 &= \min_{P\in H_n}\|f(x)-P(x)\|_2^2 \\
&= \min_{P\in H_n}\int_a^b [f(x)-P(x)]^2\,\mathrm{d}x,
\end{aligned}
$$

则称 $P^*(x)$ 为 $f(x)$ 在 $[a,b]$ 上的最佳平方逼近多项式。用 $\{1,x,\cdots,x^n\}$ 作基, 求最佳平方逼近多项式, 当 n 较大时, 系数矩阵是高度病态的, 因此直接求解法方程是相当困难的, 通常是采用正交多项式作基，如 *Legendre* 多项式作为基底求最佳平方逼近多项式。

如果 $f(x)\in C[a,b]$, 求 $[a,b]$ 上的最佳平方逼近多项式, 做变换

$$
x=\frac{b-a}{2}t+\frac{b+a}{2}\quad(-1\leqslant t\leqslant 1),
$$

于是 $F(t)=f\left(\frac{b-a}{2}t+\frac{b+a}{2}\right)$ 在 $[-1,1]$ 上可用勒让德多项式做最佳平方逼近多项式 $S_n^*(t)$, 从而得到区间 $[a,b]$ 上的最佳平方逼近多项式

$$
S_n^*\left(\frac{1}{b-a}(2x-a-b)\right)
$$

由于勒让德多项式 $\{\mathrm{P}_k(x)\}$ 是在区间 $[-1,1]$ 上由 $\left\{1,x,\cdots,x^k,\cdots\right\}$ 正交化得到的, 因此利用函数的勒让德展开部分和得到最佳平方逼近多项式与由

$$
S(x)=a_0+a_1x+\cdots+a_nx^n
$$

直接通过解法方程得到 H_n 中的最佳平方逼近多项式是一致的, 只是当 n 较大时法方程出现病态, 计算误差较大, 不能使用, 而用勒让德展开不用解线性方程组, 不存在病态问题, 计算公式比较方便, 因此通常都用这种方法求最佳平方逼近多项式。如图5所示，采用四阶勒让德多项式进行逼近

1.5.2 图像展示

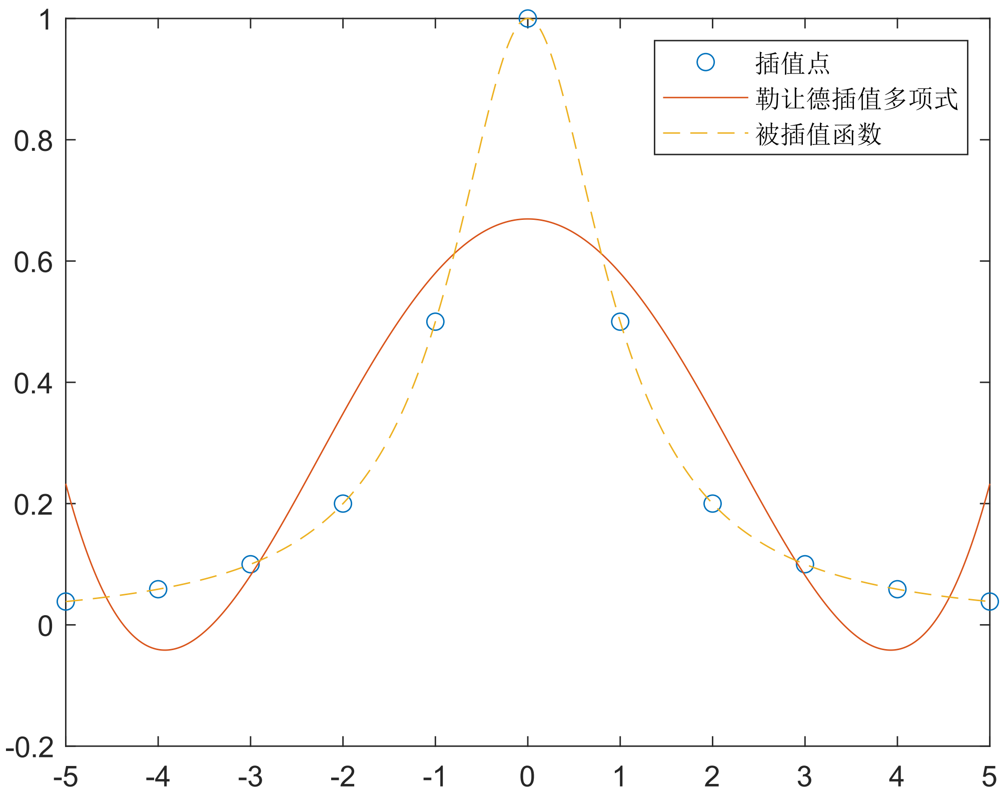

图 5: *Legendre* 多项式拟合

1.5.3 误差计算

定理：设 $f(x) \in C^2[-1,1], S_n^*(x)$ 是用勒让德多项式拟合原函数得到的多项式，则对任意 $x \in [-1,1]$ 和 $\forall \varepsilon > 0$，当 n 充分大时有

$$|f(x) - S_n^*(x)| \leqslant \frac{\varepsilon}{\sqrt{n}}.$$

采用区间 [-5,5] 上 101 个等距分布 $\mathbf{z_j} = -\mathbf{5} + \frac{\mathbf{j}}{\mathbf{10}}, j = 0, 1, ..., 100$ 处的误差绝对值的平均值来衡量。基于 *Matlab* 计算得绝对误差为 0.1082，由于选择的勒让德的多项式阶数不高，拟合绝对误差相比前面几种方法稍有逊色。

1.6 最小二乘法拟合多项式

1.6.1 算法描述

在函数的最佳平方逼近中 $f(x) \in C[a,b]$，如果 $f(x)$ 只在一组离散点集 $\{x_i, i = 0, 1, \cdots, m\}$ 上给出，这就是科学实验中经常见到的实验数据 $\{(x_i, y_i), i = 0, 1, \cdots, m\}$ 的曲线拟合，这里 $y_i =$

$f\left(x_i\right)(i=0,1,\cdots,m)$, 要求一个函数 $y=S^*(x)$ 与所给数据 $\{\left(x_i,y_i\right), i=0,1,\cdots,m\}$ 拟合, 若记误差 $\delta_i=S^*\left(x_i\right)-y_i(i=0,1,\cdots,m), \delta=\left(\delta_0,\delta_1,\cdots,\delta_m\right)^{\mathrm{T}}$, 设 $\varphi_0(x), \varphi_1(x), \cdots, \varphi_n(x)$ 是 $C[a,b]$ 上线性无关函数族, 在

$$\varphi=\operatorname{span}\left\{\varphi_0(x), \varphi_1(x), \cdots, \varphi_n(x)\right\}$$

中找一函数 $S^*(x)$, 使误差平方和

$$\|\delta\|_2^2=\sum_{i=0}^m \delta_i^2=\sum_{i=0}^m\left[S^*\left(x_i\right)-y_i\right]^2=\min_{S(x)\in\varphi}\sum_{i=0}^m\left[S\left(x_i\right)-y_i\right]^2,$$

这里

$$S(x)=a_0\varphi_0(x)+a_1\varphi_1(x)+\cdots+a_n\varphi_n(x) \quad (n<m).$$

这就是一般的最小二乘逼近, 用几何语言说, 就称为曲线拟合的最小二乘法。采用四次多项式最小二乘法，在-5 到 5 的间距为 1 的节点上拟合原函数，如图6所示

1.6.2 图像展示

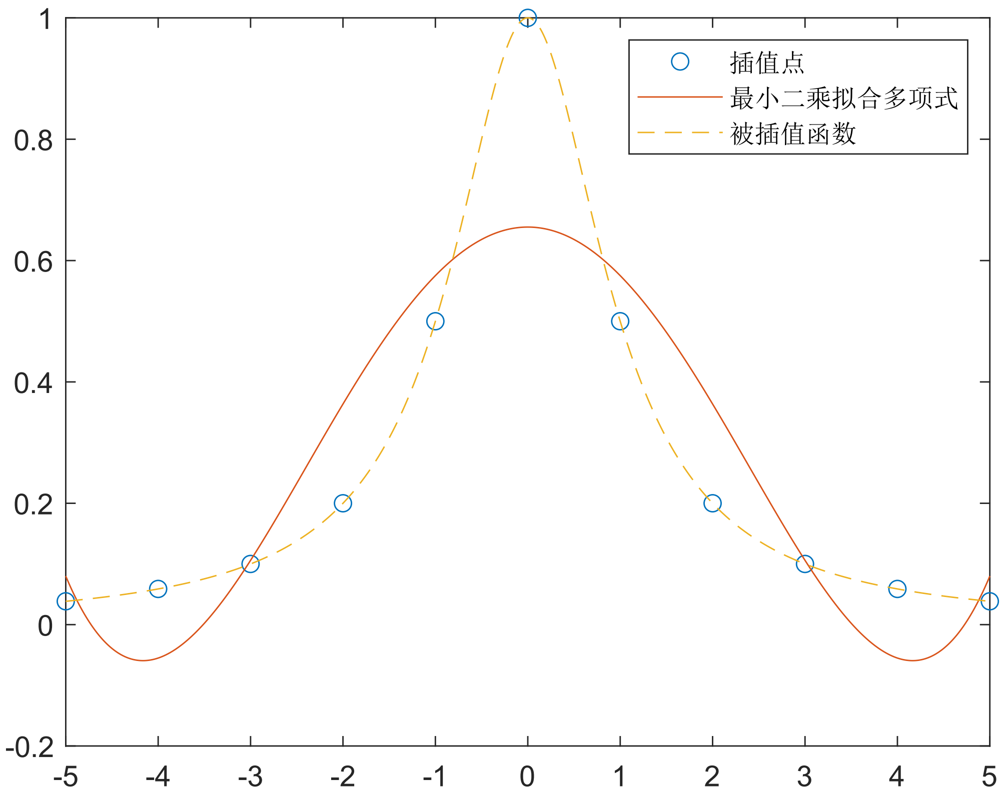

图 6: 最小二乘法拟合多项式

1.6.3 误差计算

定理：设 $f(x) \in C^2[-1,1], S_n^*(x)$ 是用勒让德多项式拟合原函数得到的多项式, 则对任意 $x \in [-1,1]$ 和 $\forall \varepsilon > 0$, 当 n 充分大时有

$$|f(x) - S_n^*(x)| \leqslant \frac{\varepsilon}{\sqrt{n}}.$$

这意味着 如果$f(x)$ 满足光滑性条件还可得到$S_n^*(x)$ 一致收敛于$f(x)$ 的结论.

采用区间 [-5,5] 上 101 个等距分布 $\mathbf{z_j} = -\mathbf{5} + \frac{\mathbf{j}}{\mathbf{10}}, j = 0, 1, ..., 100$ 处的误差绝对值的平均值来衡量。基于 $Matlab$ 计算得绝对误差为 0.1115, 与四次勒让德的多项式逼近类似，由于选择的阶数不高，拟合绝对误差大体上与四次勒让德的多项式逼近相同。

2 任务二

2.1 算法描述及误差阶分析

2.1.1 复合梯形公式

将区间 $[a,b]$ 划分为 n 等份, 分点 $x_k = a + kh, h = \frac{b-a}{n}, k = 0,1,\cdots,n$, 在每个子区间 $[x_k, x_{k+1}]\,(k = 0,1,\cdots,n-1)$ 上采用梯形公式, 则得

$$I = \int_a^b f(x)\mathrm{d}x = \sum_{k=0}^{n-1}\int_{x_k}^{x_{k+1}} f(x)\mathrm{d}x = \frac{h}{2}\sum_{k=0}^{n-1}\left[f\left(x_k\right) + f\left(x_{k+1}\right)\right] + R_n(f).$$

记

$$T_n = \frac{h}{2}\sum_{k=0}^{n-1}\left[f\left(x_k\right) + f\left(x_{k+1}\right)\right] = \frac{h}{2}\left[f(a) + 2\sum_{k=1}^{n-1} f\left(x_k\right) + f(b)\right],$$

称为复合梯形公式, 其余项

$$R_n(f) = I - T_n = \sum_{k=0}^{n-1}\left[-\frac{h^3}{12} f''\left(\eta_k\right)\right], \quad \eta_k \in \left(x_k, x_{k+1}\right).$$

由于 $f(x) \in C^2[a,b]$, 且

$$\min_{0\leqslant k\leqslant n-1} f''\left(\eta_k\right) \leqslant \frac{1}{n}\sum_{k=0}^{n-1} f''\left(\eta_k\right) \leqslant \max_{0\leqslant k\leqslant n-1} f''\left(\eta_k\right),$$

所以 $\exists \eta \in (a,b)$ 使

$$f''(\eta) = \frac{1}{n}\sum_{k=0}^{n-1} f''\left(\eta_k\right)$$

于是复合梯形公式的余项为

$$R_n(f) = -\frac{b-a}{12} h^2 f''(\eta).$$

可以看出误差是 h^2 阶。

2.1.2 复合辛普森公式

将区间 $[a,b]$ 分为 n 等份, 在每个子区间 $[x_k, x_{k+1}]$ 上采用辛普森公式 (2.3), 若记 $x_{k+1/2} = x_k + \frac{1}{2}h$, 则得

$$\begin{aligned} I &= \int_a^b f(x)\mathrm{d}x = \sum_{k=0}^{n-1}\int_{x_k}^{x_{k+1}} f(x)\mathrm{d}x \\ &= \frac{h}{6}\sum_{k=0}^{n-1}\left[f\left(x_k\right) + 4f\left(x_{k+1/2}\right) + f\left(x_{k+1}\right)\right] + R_n(f). \end{aligned}$$

记

$$\begin{aligned} S_n &= \frac{h}{6}\sum_{k=0}^{n-1}\left[f\left(x_k\right) + 4f\left(x_{k+1/2}\right) + f\left(x_{k+1}\right)\right] \\ &= \frac{h}{6}\left[f(a) + 4\sum_{k=0}^{n-1} f\left(x_{k+1/2}\right) + 2\sum_{k=1}^{n-1} f\left(x_k\right) + f(b)\right], \end{aligned}$$

称为复合辛普森求积公式, 其余项由 (2.5) 式得

$$R_n(f)=I-S_n=-\frac{h}{180}\left(\frac{h}{2}\right)^4\sum_{k=0}^{n-1}f^{(4)}\left(\eta_k\right),\quad \eta_k\in\left(x_k,x_{k+1}\right).$$

于是当 $f(x)\in C^4[a,b]$ 时, 与复合梯形公式相似有

$$R_n(f)=I-S_n=-\frac{b-a}{180}\left(\frac{h}{2}\right)^4 f^{(4)}(\eta),\quad \eta\in(a,b).$$

由 (3.6) 式看出, 误差阶为 h^4。

2.1.3 龙贝格积分算法

从梯形公式出发, 将区间 $[a,b]$ 逐次二分可提高求积公式精度, 当 $[a,b]$ 分为 n 等份时, 有

$$I-T_n=-\frac{b-a}{12}h^2f''(\eta),\quad \eta\in[a,b],\quad h=\frac{b-a}{n}.$$

若记 $T_n=T(h)$, 当区间 $[a,b]$ 分为 $2n$ 等份时, 则有 $T_{2n}=T\left(\frac{h}{2}\right)$, 并且有

$$T(h)=I+\frac{b-a}{12}h^2f''(\eta),\quad \lim_{h\to 0}T(h)=T(0)=I,$$

可以证明梯形公式的余项可展成级数形式, 即有下面定理.

定理 1 设 $f(x)\in C^\infty[a,b]$, 则有

$$T(h)=I+\alpha_1h^2+\alpha_2h^4+\cdots+\alpha_lh^{2l}+\cdots,$$

其中系数 $\alpha_l(l=1,2,\cdots)$ 与 h 无关. 此定理可利用 $f(x)$ 的泰勒展开推导得到, 此处从略。

定理1表明 $T(h)\approx I$ 是 $O\left(h^2\right)$ 阶, 在 (4.2) 式中, 若用 $h/2$ 代替 h, 有

$$T\left(\frac{h}{2}\right)=I+\alpha_1\frac{h^2}{4}+\alpha_2\frac{h^4}{16}+\cdots+\alpha_l\left(\frac{h}{2}\right)^{2l}+\cdots.$$

则有

$$S(h)=\frac{4T(h/2)-T(h)}{3}=I+\beta_1h^4+\beta_2h^6+\cdots,$$

这里 $\beta_1,\beta_2,\cdots$ 是与 h 无关的系数. 用 $S(h)$ 近似积分值 I, 其误差阶为 $O\left(h^4\right)$, 这比复合梯形公式的误差阶 $O\left(h^2\right)$ 提高了, 容易看到 $S(h)=S_n$, 即将 $[a,b]$ 分为 n 等份得到的复合辛普森公式[1]. 这种将计算 I 的近似值的误差阶由 $O\left(h^2\right)$ 提高到 $O\left(h^4\right)$ 的方法称为外推算法。

设以 $T_0^{(k)}$ 表示二分 k 次后求得的梯形值, 且以 $T_m^{(k)}$ 表示序列 $\left\{T_0^{(k)}\right\}$ 的 m 次加速值, 则依递推公式 (4.9) 可得

$$T_m^{(k)}=\frac{4^m}{4^m-1}T_{m-1}^{(k+1)}-\frac{1}{4^m-1}T_{m-1}^{(k)},\quad k=1,2,\cdots.$$

公式 (4.11) 也称为龙贝格求积算法。

[1]稍后会验证区间等分 10 份的复合辛普森公式与经过一次龙贝格加速的区间等分 10 份的复合梯形公式的结果完全一致

2.1.4 高斯求积公式

形如下式的机械求积公式

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{k=0}^n A_k f(x_k)$$

含有 $2n+2$ 个待定参数 $x_k, A_k(k=0,1,\cdots,n)$. 当 x_k 为等距节点时得到的插值求积公式其代数精度至少为 n 次, 如果适当选取 $x_k(k=0,1,\cdots,n)$, 有可能使求积公式具有 $2n+1$ 次代数精度。如果求积公式具有 $2n+1$ 次代数精度, 则称其节点 $x_k(k=0,1,\cdots,n)$ 为高斯点, 相应公式称为高斯型求积公式. 在高斯求积公式中, 若取权函数 $\rho(x)=1$, 区间为 $[-1,1]$, 则得公式

$$\int_{-1}^1 f(x)\mathrm{d}x \approx \sum_{k=0}^n A_k f(x_k).$$

我们知道勒让德多项式是区间 $[-1,1]$ 上的正交多项式, 因此, 勒让德多项式 $\mathrm{P}_{n+1}(x)$ 的零点就是求积公式的高斯点. 此时称为高斯-勒让德求积公式。

n	x_k	A_k
0	0.0000000	2.0000000
1	±0.5773503	1.0000000
2	±0.7745967	0.5555556
	0.0000000	0.8888889
3	±0.8611363	0.3478548
	±0.3399810	0.6521452
4	±0.9061798	0.2369269
	±0.5384693	0.4786287
	0.0000000	0.5688889
5	±0.9324695	0.1713245
	±0.6612094	0.3607616
	±0.2386192	0.4679139

表 1: 高斯 - 勒让德求积公式的节点和系数

复合高斯公式将区间 $[a,b]$ 分为 n 等份, 在每个子区间 $[x_k, x_{k+1}]$ 上采用高斯公式。

下面讨论 Gauss 求积公式的误差

定理 2 *定理设 $f \in C^{2n+2}[a,b]$, 那么 Gauss 求积公式 (4.8) 的误差表达式为*

$$E_n(f) = \frac{1}{(2n+2)!} f^{(2n+2)}(\eta) \int_a^b \rho(x)\left[\omega_{n+1}(x)\right]^2 \mathrm{d}x,$$

其中 $\eta \in [a,b], \omega_{n+1}(x) = (x-x_0)(x-x_1)\cdots(x-x_n)$

设 $(b-a)$=h, 则

$$\omega_{n+1}(x) = (x-x_0)(x-x_1)\cdots(x-x_n)$$

可以近似的视作为 h^m 阶，则易得复合高斯公式的误差为 h^{2m} 阶，其中 m 为高斯点的个数。

2.2 误差计算

采用区间 [-5,5] 上 101 个等距分布 $\mathbf{z_j = -5 + \frac{j}{10}}, j = 0, 1, ..., 100$ 处的误差绝对值的平均值来衡量。基于 *Matlab* 计算得各种方法计算积分结果与误差如下表所示：

method	*value*	*bias*
trapz	2.2132	1.1×10^{-3}
simpsons	2.2143	4.3816×10^{-6}
Romberg1	2.2143	4.3816×10^{-6}
Romberg2	2.2142	5.4711×10^{-5}
gauss_2points	2.2143	2.9372×10^{-6}
gauss_4points	2.2143	2.7243×10^{-6}

表 2: 不同方法计算积分结果

2.3 误差比较与评价

由表2可知，复合四点高斯公式的误差最小，其次是复合两点高斯公式。而在复合梯形公式基础上改进的复合辛普森公式、龙贝格积分法的误差均相比于复合梯形公式减少了 100 倍以上。值得注意的是复合梯形公式经过一次龙贝格加速后所得到的结果，和辛普森公式所得到的完全一致。这与理论推导相照应。

此外，本次实验中不难发现，龙贝格二次加速得到的结果竟然比龙贝格一次加速的结果误差更大，经过和室友、同学认真的讨论，结论是：这仅仅是偶然现象，虽然龙贝格二次加速的误差阶数绝对会更小，却不意味着绝对误差一定会小于一次龙贝格加速的误差。此外，经过尝试三次，四次龙贝格加速之后，误差总体上仍然显示为下降趋势，这意味着算法本身问题不大。本次实验中遇到的情况的属于正常现象。

3 选做题

3.1 积分公式误差阶理论分析

根据 2.1 节 (**见此处**) 算法描述中的详细分析，各种积分公式的误差阶如表3所示：

method	$bias_order$
trapz	h^2
simpsons	h^4
gauss_3points	h^6

表 3: 不同积分方法误差阶数

3.2 数值验证误差阶

根据 $Matlab$ 计算，在 k 取不同的值时，通过

$$\frac{\log(e_k) - \log(e_{k-1})}{\log(h_k) - \log(h_{k-1})}$$

验证收敛阶 $O(h^\alpha)$ 中的阶数 α ，（其中 h_k, e_k 分别为 $k = 2,3,4,5,6$ 时对应的区间长度和积分误差）所得到的阶数如表4所示：

k	3	4	5	6
$trapz\_order$	1.1141	1.9888	1.9984	1.9996
$simpsons$	4.3245	7.8773	4.7853	3.9970
$gauss\_3points$	9.01449	12.52915	5.99745	5.9939

表 4: 数值验证误差阶数

通过表中数据可以明显看出，随着 k 值不断增大，不同积分公式的 k 值趋向于预估的值，说明理论推导正确。

4 代码

4.1 任务一

由于题目要求发生变化（不允许调用库函数），原始代码调用了库函数，现已经重新对所有调用库函数的地方做出了补充，补充了调用自己编的函数的方法，原调用库函数的方法予以保留。

—2022/05/18

4.1.1 主函数部分

```
% 下面这里是lagrange插值插值
clc
%任务1.1
% 需要插值的点
x=linspace(-5,5,11);
y=1./(1+x.^2);
%根据插值点构造插值多项式
xx=-5:0.01:5;
yy = lagrange(xx,x,y);
% 与真实函数图像做对比，f是真实函数图像。
f=1./(1+xx.^2);
plot(x,y,'o',xx,yy,xx,f,'--');
legend('插值点','lagrange插值多项式','被插值函数');
saveas(gcf,'lagrange.svg');
% 1.画出每个插值点，形式为圆圈
% 2.画出插值函数生成的图
% 3.画出真实函数图像以做对比

%误差计算
xxx=linspace(-5,5,101);
yyy = lagrange(xxx,x,y);
real_values=1./(1+xxx.^2);
abs(yyy-real_values);
bias=mean(abs(yyy-real_values))

clc
%任务1.2
% 需要插值的点
% find 11 Chebyshev nodes and mark them on the plot
n = 11;
k = 1:11; % iterator
xc = cos((2*k-1)/2/n*pi); % Chebyshev nodes
x=5.*xc;
y=1./(1+x.^2);
%根据插值点构造插值多项式
xx=-5:0.01:5;
yy = lagrange(xx,x,y);
% 与真实函数图像做对比，f是真实函数图像。
f=1./(1+xx.^2);
plot(x,y,'o',xx,yy,xx,f,'--');
legend('插值点','chebyshev插值多项式','被插值函数');
```

```
saveas(gcf,'chebyshev.svg');
% 1.画出每个插值点，形式为圆圈
% 2.画出插值函数生成的图
% 3.画出真实函数图像以做对比

%误差计算
xxx=linspace(-5,5,101);
yyy = lagrange(xxx,x,y);
real_values=1./(1+xxx.^2);
abs(yyy-real_values);
bias=mean(abs(yyy-real_values))



clc
%任务1.3 （1）分段线性插值不调库法
syms x
y = 1/(1+x^2)
x0 = -5:5
y0 = 1./(1+x0.^2)
error = 0
for i = 1:10
    a=y0(i)*(x-x0(i+1))/(x0(i)-x0(i+1))+y0(i+1)*(x-x0(i))/(x0(i+1)-x0(i));
    for j = -6+i:0.1:-5+i
        error = error + abs(subs(a,x,j)-subs(y,x,j));
    end
    fplot(a,[-6+i,-5+i])
    hold on
end
a= eval(vpa(error,5))
fplot(y,[-5,5])
saveas(gcf,'another_linear.svg');
hold off



clc
%任务1.3 （2）分段线性插值调库法
% 需要插值的点
x=linspace(-5,5,11);
y=1./(1+x.^2);
%根据插值点构造插值多项式
xx=-5:0.01:5;
yy = interp1(x,y,xx);
% 与真实函数图像做对比，f是真实函数图像。
f=1./(1+xx.^2);
plot(x,y,'o',xx,yy,xx,f,'--')
legend('插值点','线性插值多项式','被插值函数')
saveas(gcf,'linear.svg');
% 1.画出每个插值点，形式为圆圈
% 2.画出插值函数生成的图
% 3.画出真实函数图像以做对比

%误差计算
```

```
xxx=linspace(-5,5,101);
yyy = interp1(x,y,xxx);
real_values=1./(1+xxx.^2);
abs(yyy-real_values);
bias=mean(abs(yyy-real_values))




clc
%任务1.4 (1)三次样条插值调库法
% 需要插值的点
x=linspace(-5,5,11);
y=1./(1+x.^2);

syms m
f = 1/(1+m^2);
diff_f = diff(f);
y1 = subs(diff_f,m,-5);
y2 = subs(diff_f,m,5);
y1=eval(y1);
y2=eval(y2);
y_spline=[y1 y y2];
%根据两端点的一阶导数值(clamped方法)
%即第一类边界条件
%根据插值点构造插值多项式
cs = spline(x,y_spline);

%计算三次样条插值各个密集点的值
xx=-5:0.01:5;
yy = ppval(cs,xx);
% 与真实函数图像做对比，f是真实函数图像。
f=1./(1+xx.^2);
plot(x,y,'o',xx,yy,xx,f,'--')
legend('插值点','三次样条插值多项式','被插值函数')
saveas(gcf,'spline.svg');
% 1.画出每个插值点，形式为圆圈
% 2.画出插值函数生成的图
% 3.画出真实函数图像以做对比

%误差计算
xxx=linspace(-5,5,101);
yyy = ppval(cs,xxx);
real_values=1./(1+xxx.^2);
abs(yyy-real_values);
bias=mean(abs(yyy-real_values))



clc
%任务1.4 (2)三次样条插值自编法
% 需要插值的点
```

```
x=linspace(-5,5,11);
y=1./(1+x.^2);

syms m
f = 1/(1+m^2);
diff_f = diff(f);
y1 = subs(diff_f,m,-5);
y2 = subs(diff_f,m,5);
y1=eval(y1);
y2=eval(y2);
%根据两端点的一阶导数值(clamped方法)
%即第一类边界条件
%根据插值点构造插值多项式

%计算三次样条插值各个密集点的值
xx=-5:0.01:5;
yy = splineA(x,y,xx,[1,2],[y1,y2]);
% 与真实函数图像做对比, f是真实函数图像。
f=1./(1+xx.^2);
plot(x,y,'o',xx,yy,xx,f,'--')
legend('插值点','三次样条插值多项式','被插值函数')
saveas(gcf,'another_spline.svg');
% 1.画出每个插值点, 形式为圆圈
% 2.画出插值函数生成的图
% 3.画出真实函数图像以做对比

%误差计算
xxx=linspace(-5,5,101);
yyy = ppval(cs,xxx);
real_values=1./(1+xxx.^2);
abs(yyy-real_values);
bias=mean(abs(yyy-real_values))

clc
%任务1.5 最佳平方逼近多项式
% 需要插值的点
x=linspace(-5,5,11);
y=1./(1+x.^2);

syms m
legend_f=0;
f=1/(1+25*m^2);
for i = 0:4
    s=legendreP(i,m)*f;
    legend_f = legend_f+legendreP(i,m)*eval(int(s,m,-1,1))*(2*i+1)/2;
end
%计算勒让德多项式插值各个密集点的值
legend_f=subs(legend_f,m,m/5);
xx=-5:0.01:5;
yy = eval(subs(legend_f,m,xx));
% 与真实函数图像做对比, f是真实函数图像。
f=1./(1+xx.^2);
plot(x,y,'o',xx,yy,xx,f,'--')
legend('插值点','勒让德插值多项式','被插值函数')
```

```
saveas(gcf,'legendre.svg');
% 1.画出每个插值点，形式为圆圈
% 2.画出插值函数生成的图
% 3.画出真实函数图像以做对比

%误差计算
xxx=linspace(-5,5,101);
yyy = eval(subs(legend_f,m,xxx));
real_values=1./(1+xxx.^2);
abs(yyy-real_values);
bias=mean(abs(yyy-real_values))



clc
%%%任务1.6 最小二乘拟合多项式
%% 需要插值的点
x_points=linspace(-5,5,11);
y=1./(1+x_points.^2);
%根据插值点构造插值多项
syms x;
f = 1./(1+x.^2);
p = least_squares(f,x_points);
%此处由于不允许调库，所以改成手写
%如果调库，可以使用polyfit
xx=-5:0.01:5;
p4=subs(p,x,xx);
xxx=linspace(-5,5,101);
poly_vals=eval(subs(p,x,xxx));
%根据插值多项式生成一系列密集点用来画图

%与真实函数图像做对比，f是真实函数图像。
f=1./(1+xx.^2);
plot(x_points,y,'o',xx,p4,xx,f,'--');
legend('插值点','最小二乘拟合多项式','被插值函数');
saveas(gcf,'least_squares.svg');
%1.画出每个插值点，形式为圆圈
%2.画出插值函数生成的图
%3.画出真实函数图像以做对比

% 误差计算

real_values=1./(1+xxx.^2);
abs(poly_vals-real_values);
bias=mean(abs(poly_vals-real_values));
```

4.1.2 自定义函数部分

```
function y = splineA(xd,yd,x,Ends,Ders)
n=length(xd);

```

```
h=diff(xd); h1=h(1); hn=h(n-1);
h=[h1 h1 h1 h hn hn hn];
Xd=[xd(1)-3*h1 xd(1)-2*h1 xd(1)-h1 xd xd(n)+hn xd(n)+2*hn xd(n)+3*hn];
[B,G]=coeffs(h);
if Ends(1)==2
    alpha0=G(4,1)/G(2,1);
    beta0=G(6,1)/G(2,1);
    gamma0=Ders(1)/G(2,1);
elseif Ends(1)==1
    alpha0=G(3,1)/G(1,1);
    beta0=G(5,1)/G(1,1);
    gamma0=Ders(1)/G(1,1);
end
B0=Bi(xd(1),0,B,Xd);
Aa(1)=2/3-B0*alpha0;
Ac(1)=Bi(xd(1),2,B,Xd)-B0*beta0;
Ab(1)=0;
yd(1)=yd(1)-B0*gamma0;
for ii=2:n-1
    Aa(ii)=2/3;
    Ac(ii)=Bi(xd(ii),ii+1,B,Xd);
    Ab(ii)=Bi(xd(ii),ii-1,B,Xd);
end
if Ends(2)==2
    alphan=G(2,2)/G(6,2);
    betan=G(4,2)/G(6,2);
    gamman=Ders(1)/G(6,2);
elseif Ends(2)==1
    alphan=G(1,2)/G(5,2);
    betan=G(3,2)/G(5,2);
    gamman=Ders(2)/G(5,2);
end
Bn1=Bi(xd(n),n+1,B,Xd);
Aa(n)=2/3-Bn1*betan;
Ac(n)=0;
Ab(n)=Bi(xd(n),n-1,B,Xd)-Bn1*alphan;
yd(n)=yd(n)-gamman*Bn1;
w=tri(Aa,Ab,Ac,yd);
a0=-alpha0*w(1)-beta0*w(2)+gamma0;
anp1=-alphan*w(n-1)-betan*w(n)+gamman;
w=[a0 w anp1];
nx=length(x);
y=zeros(1,nx);
for ix=1:nx
    sum=0;
    for k=1:n+2
        sum=sum+w(k)*Bi(x(ix),k-1,B,Xd);
    end
    y(ix)=sum;
end

% local functions used by algorithm
function y = tri( a, b, c, f )
N = length(f);
```

```
v = zeros(1,N);
y = v;
w = a(1);
y(1) = f(1)/w;
for i=2:N
    v(i-1) = c(i-1)/w;
    w = a(i) - b(i)*v(i-1);
    y(i) = ( f(i) - b(i)*y(i-1) )/w;
end
for j=N-1:-1:1
    y(j) = y(j) - v(j)*y(j+1);
end

function [B,G] =coeffs(h)
N=length(h);
B=zeros(N-3,16);
G=zeros(6,2);
for j=1:N-3
    hm2=h(j); hm1=h(j+1); h1=h(j+2); h2=h(j+3);
    d2=solver(hm2,hm1,h1,h2);
    B(j,4)=d2(1);
    B(j,5)=d2(1)*hm2^3;
    B(j,6)=3*d2(1)*hm2^2;
    B(j,7)=3*d2(1)*hm2;
    Tm3=(hm2+hm1)^3-hm1^3;
    B(j,8)=(2/3-Tm3*d2(1))/hm1^3;
    B(j,9)=-d2(2)*h2^3;
    B(j,10)=3*d2(2)*h2^2;
    B(j,11)=-3*d2(2)*h2;
    T3=(h2+h1)^3-h1^3;
    B(j,12)=(-2/3-T3*d2(2))/h1^3;
    B(j,16)=d2(2);
    if j==1
        G(1,1)=3*d2(2)*h2^2;
        G(2,1)=-6*d2(2)*h2;
    end
    if j==2
        G(3,1)=3*d2(2)*h2*(h2+2*h1)+3*B(j,12)*h1^2;
        G(4,1)=-6*(d2(2)*h2+B(j,12)*h1);
    end
    if j==3
        G(5,1)=3*d2(1)*hm2^2;
        G(6,1)=6*d2(1)*hm2;
    end
    if j==N-5
        G(1,2)=3*d2(2)*h2^2;
        G(2,2)=-6*d2(2)*h2;
    end
    if j==N-4
        G(3,2)=3*d2(2)*h2*(h2+2*h1)+3*B(j,12)*h1^2;
        G(4,2)=-6*(d2(2)*h2+B(j,12)*h1);
    end
    if j==N-3
        G(5,2)=3*d2(1)*hm2^2;
```

```
            G(6,2)=6*d2(1)*hm2;
        end

end

function d2=solver(hm2,hm1,h1,h2)
a=h1*hm2*(hm2+hm1)^2;
b=-hm1*h2*(h2+h1)^2;
c=h1^2*hm2*(hm2+hm1)*(hm2+2*hm1);
d=-hm1^2*h2*(h2+h1)*(h2+2*h1);
A=[[a b];[c d]];
bb=[2*(hm1+h1)/3; -2*(hm1^2-h1^2)/3];
d2=A\bb;

function g=Bi(x,i,B,Xd)
j=i+1;
if x<Xd(j) || x>Xd(j+4)
    g=0;
elseif x<Xd(j+1)
    g=B(j,4)*(x-Xd(j))^3;
elseif x<Xd(j+2)
    g=B(j,5)+B(j,6)*(x-Xd(j+1))+B(j,7)*(x-Xd(j+1))^2+B(j,8)*(x-Xd(j+1))^3;
elseif x<Xd(j+3)
    g=B(j,9)+B(j,10)*(x-Xd(j+3))+B(j,11)*(x-Xd(j+3))^2+B(j,12)*(x-Xd(j+3))^3;
else
    g=B(j,16)*(x-Xd(j+4))^3;
end


function y=lagrange(x,pointx,pointy)
n=size(pointx,2);
L=ones(n,size(x,2));
if (size(pointx,2)~=size(pointy,2))
   fprintf(1,'\nERROR!\nPOINTX and POINTY must have the same number of elements\n');
   y=NaN;
else
   for i=1:n
      for j=1:n
         if (i~=j)
            L(i,:)=L(i,:).*(x-pointx(j))/(pointx(i)-pointx(j));
         end
      end
   end
   y=0;
   for i=1:n
      y=y+pointy(i)*L(i,:);
   end
end

function final_function=least_squares(f,pointx)
clc;
syms x;

%初始化正交多项式组
```

```
p=cell(5,1);

%0次正交多项式
p{1}=1;
%1次正交多项式
p{2}= x;

%2,3,4次正交多项式

for k=2:4
    alpha(k+1) = dot(subs(p{k},x,pointx),subs(p{k}*x,x,pointx))/dot(subs(p{k},x,pointx),subs(p{k},x
        ,pointx));
    beta(k)=dot(subs(p{k},x,pointx),subs(p{k},x,pointx))/dot(subs(p{k-1},x,pointx),subs(p{k-1},x,
        pointx));
    p{k+1} = (x-alpha(k+1))*p{k}-beta(k)*p{k-1};
end

%得到最终的正交多项式
final_function=0;
for k=1:5
    a(k)=dot(subs(f,x,pointx),subs(p{k},x,pointx))/dot(subs(p{k},x,pointx),subs(p{k},x,pointx));
    final_function=final_function+a(k)*p{k};
end


% end
```

4.2 任务二

4.2.1 主函数部分

```



% Compound trapezoid formula
clc
clear
syms x;
f=@(x) 1./(1+x.^2);
a=-2;
b=2;
n=20;
h=(b-a)/n;
sum=0;
for k=1:1:n-1
  x(k)=a+k*h;
  y(k)=f(x(k));
  sum=sum+y(k);
end
answer=h/2*(f(a)+f(b)+2*sum);
```

```
trapz_answer=eval(answer)
ground_truth=2*atan(2)
trapz_bias= abs(trapz_answer-ground_truth)

simpsons_answer = simpsons(f,a,b,n)
simpsons_bias = abs(simpsons_answer-ground_truth)

syms h;
% h=(b-a)/10;
sum=0;
for k=1:1:10-1
  x(k)=a+k*h;
  y(k)=f(x(k));
  sum=sum+y(k);
end
trapz_10_formula=h/2*(f(a)+f(b)+2*sum);
%在trapz_10_formula基础上再次二分之后得到的公式如下


sum=0;
for k=1:1:20-1
  x(k)=a+k*h/2;
  y(k)=f(x(k));
  sum=sum+y(k);
end
trapz_20_formula=h/4*(f(a)+f(b)+2*sum);
%里面的h仍为(b-a)/10


Romberg1_formula = trapz_20_formula*4/3-1/3*trapz_10_formula
Romberg1=eval(subs(Romberg1_formula,h,(b-a)/10))
Romberg1_bias = abs(Romberg1-ground_truth)
%上面的先分10份然后用龙贝格一次加速的任务已完成



syms h;
% h=(b-a)/5;
sum=0;
for k=1:1:5-1
  x(k)=a+k*h;
  y(k)=f(x(k));
  sum=sum+y(k);
end
trapz_5_formula=h/2*(f(a)+f(b)+2*sum);
%上面是分成五份的复合梯形公式。

syms h;
% h=(b-a)/5;
sum=0;
for k=1:1:10-1
  x(k)=a+k*h/2;
  y(k)=f(x(k));
  sum=sum+y(k);
```

```
end
trapz_10_formula=h/4*(f(a)+f(b)+2*sum);
%上面是分成十份的复合梯形公式。
%里面的h仍为(b-a)/5

syms h;
% h=(b-a)/5;
sum=0;
 for k=1:1:20-1
   x(k)=a+k*h/4;
   y(k)=f(x(k));
   sum=sum+y(k);
end
trapz_20_formula=h/8*(f(a)+f(b)+2*sum);
%上面是分成二十份的复合梯形公式。
%里面的h仍为(b-a)/5

romberg21_formula=trapz_10_formula*4/3-1/3*trapz_5_formula;
romberg22_formula=trapz_20_formula*4/3-1/3*trapz_10_formula;
%上面分别进行了第一次龙贝格加速

romberg2_formula=romberg22_formula*16/15-1/15*romberg21_formula
%这里进行第二次龙贝格加速

romberg2=eval(subs(romberg2_formula,h,(b-a)/5))
romberg2_bias = abs(romberg2-ground_truth)



%下面使用高斯勒让德积分公式
a = linspace(-2,2,11);
%matlab的数组下标从1开始
sum=0;
 for i = 1:10
      sum = sum+Gauss_Legendre_quadrature(f,a(i),a(i+1),2);
end
% 复合的 2 点高斯公式
gauss_2points = sum
gauss_2points_bias= abs(gauss_2points-ground_truth)
a = linspace(-2,2,6);
%matlab的数组下标从1开始
sum=0;
 for i = 1:5
      sum = sum+Gauss_Legendre_quadrature(f,a(i),a(i+1),4);
end
% 复合的 4 点高斯公式
gauss_4points = sum
gauss_4points_bias= abs(gauss_4points-ground_truth)
```

4.2.2 自定义函数部分

```
function I = Gauss_Legendre_quadrature(f,a,b,point)
%在-1到1基础上
a0=(b+a)/2;%中心点向右平移的长度
a1=(b-a)/2;%在长度上伸展的倍数

if point==2
    digits(100)
    x0=a0+(a1*(-1/(sqrt(3))));
    x1=a0+(a1*(1/(sqrt(3))));
    fx0=a1*f(x0);
    fx1=a1*f(x1);
    I=fx0+fx1;
end

if point==3
    digits(100)
    x0=a0+(a1*(-vpa(sqrt(3/5))));
    x1=a0;
    x2=a0+(a1*vpa(sqrt(3/5)));
    fx0=a1*vpa(f(x0));
    fx1=a1*vpa(f(x1));
    fx2=a1*vpa(f(x2));
    I=(vpa(5/9)*fx0)+(vpa(8/9)*fx1)+(vpa(5/9)*fx2);
end

if point==4
    digits(100)
    x0=a0+(a1*-vpa((sqrt(3/7-2/7*sqrt(6/5))),100));
    x1=a0+(a1*-vpa((sqrt(3/7+2/7*sqrt(6/5))),100));
    x2=a0+(a1*vpa((sqrt(3/7+2/7*sqrt(6/5))),100));
    x3=a0+(a1*vpa((sqrt(3/7-2/7*sqrt(6/5))),100));
    fx0=a1*f(x0);
    fx1=a1*f(x1);
    fx2=a1*f(x2);
    fx3=a1*f(x3);
    I=((18+vpa(sqrt(30),100))/36*fx0)+((18-vpa(sqrt(30),100))/36*fx1)+((18-vpa(sqrt(30),100))/36*
        fx2)+((18+vpa(sqrt(30),100))/36*fx3);
end

%Interpretation of results

function I = simpsons(f,a,b,n)
% This function computes the integral "I" via Simpson's rule in the interval [a,b] with n+1 equally
    spaced points

if numel(f)>1 % If the input provided is a vector
    n=numel(f)-1; h=(b-a)/n;
    I= h/3*(f(1)+2*sum(f(3:2:end-2))+4*sum(f(2:2:end))+f(end));
else % If the input provided is an anonymous function
    h=(b-a)/n; xi=a:h:b;
    I= h/3*(f(xi(1))+2*sum(f(xi(3:2:end-2)))+4*sum(f(xi(2:2:end)))+f(xi(end)));
end
```

4.3 选做题

```
digits(100)
%下面是选做题:
%先验证复合三点高斯公式收敛阶数
for k=2:8
    sum=0;
    a = linspace(-2,2,2^k+1);
    for i = 1:2^k
        sum = sum+vpa(Gauss_Legendre_quadrature(f,a(i),a(i+1),3),100);
    end
    gauss_3points_bias(k)=abs(sum-ground_truth)
end

for k=2:7
    a = log(gauss_3points_bias(k+1))-log(gauss_3points_bias(k));
    b = log(4/(2^(k+1)))-log(4/(2^(k)));
    order_gauss = a/b
end
%此后的结果趋向于大概是6, 说明复合梯形公式余项阶数是6


%验证复合梯形公式收敛阶数
for m=2:6
    % Compound trapezoid formula
    b=2;
    a=-2;
    n=2^m;
    h=(b-a)/n;
    sum=0;
    for k=1:1:n-1
      x(k)=a+k*h;
      y(k)=f(x(k));
      sum=sum+y(k);
    end
    answer=h/2*(f(a)+f(b)+2*sum);
    trapz_answer=eval(answer);
    ground_truth=2*atan(2);
    trapz_bias(m)= abs(trapz_answer-ground_truth);
end
trapz_bias
for k=2:5
    a = log(trapz_bias(k+1))-log(trapz_bias(k));
    b = log(4/(2^(k+1)))-log(4/(2^(k)));
    order_trapz = a/b
end
%此后的结果趋向于大概是2, 说明复合梯形公式余项阶数是2

```

```
%验证复合辛普森公式收敛阶数
for m=2:6
    a=-2;
    b=2;
    simpsons_answer = simpsons(f,a,b,2^m);
    simpsons_bias(m) = abs(simpsons_answer-ground_truth);
end
simpsons_bias
for k=2:5
    a = log(simpsons_bias(k+1))-log(simpsons_bias(k));
    b = log(4/(2^(k+1)))-log(4/(2^(k)));
    order_simpson = a/b
end
%此后的结果趋向于大概是4，说明复合辛普森公式余项阶数是4
```