1 正交多项式作为基函数下的多项式投影

1.1 正交多项式投影

对函数 $f \in L^2_\omega(I)$，定义其加权 L^2 范数为：

$$\|f\|_{L^2_\omega} = \left(\int_I f^2(x)\, \omega(x)\, dx \right)^{\frac{1}{2}}$$

对应的内积为：

$$(f, g) = \int_I f(x) g(x)\, \omega(x)\, dx$$

设 $\{\phi_i(x)\}_{i=0}^N \subset \mathcal{P}_N$ 为权函数 $\omega(x)$ 对应的正交多项式族，满足：

$$(\phi_m, \phi_n) = \|\phi_m\|^2_{L^2_\omega}\, \delta_{mn}$$

定义投影算子 $P_N : L^2_\omega(I) \to \mathcal{P}_N$，使得对任意 $f \in L^2_\omega(I)$，有：

$$(f, \phi_i) = (P_N f, \phi_i), \quad i = 0, 1, \ldots, N$$

则投影函数可表示为：

$$P_N f = \sum_{i=0}^{N} \hat{f}_i\, \phi_i(x), \quad \text{其中} \quad \hat{f}_i = \frac{(f, \phi_i)}{\|\phi_i\|^2_{L^2_\omega}}$$

实现正交多项式投影的代码如下，函数 projection 用于计算 $\hat{f}_i$，函数 evaluate 用于计算最终的 $\sum_{i=0}^N \hat{f}_i\, \phi_i(x)$. 其中积分的部分使用了高斯求积进行数值积分。

```
/// @brief 正交多项式投影的系数
/// @param f 待投影的函数
/// @param poly 基函数（正交多项式）
/// @param N 正交多项式的最高阶数
/// @return 投影系数
std::vector<double> projection(
    const std::function<double(double)> &f,
    const Polynomial &poly,
    int N)
{
    std::vector<double> coeffs(N + 1);
    const auto &points = poly.get_points();
    const auto &weights = poly.get_weights();
    std::vector<double> vals;
    for (size_t i = 0; i < points.size(); ++i)
    {
        double x = points[i];
        double w = weights[i];
        double fx = f(x);
```

```
        vals = poly.eval_all(N, x);
        for (int k = 0; k <= N; ++k)
        {
            coeffs[k] += w * fx * vals[k]; // 相当于计算内积时做的积分
        }
    }

    // 归一化
    for (int k = 0; k <= N; ++k)
    {
        coeffs[k] /= poly.norm(k);
    }
    return coeffs;
}

/// @brief 求网格上每个点的值
/// @param coeffs 投影系数
/// @param poly 正交多项式
/// @param x_grid 网格
/// @return 每个网格点上的投影值
std::vector<double> evaluate(
    const std::vector<double> &coeffs,
    const Polynomial &poly,
    const std::vector<double> &x_grid)
{
    std::vector<double> result;
    result.reserve(x_grid.size());

    for (double x : x_grid)
    {
        auto phis = poly.eval_all(coeffs.size() - 1, x);
        double sum = 0.0;
        for (size_t k = 0; k < coeffs.size(); ++k)
        {
            sum += coeffs[k] * phis[k];
        }
        result.push_back(sum);
    }

    return result;
}
```

1.2 误差的收敛性

设 $f(x) \in H_\omega^{2m}[-1,1]$，其中 $m \geq 0$，则存在与 N 无关的常数 $C > 0$，使得：

$$\|f - P_N f\|_{L^2_\omega[-1,1]} \leq CN^{-2m} \|f\|_{H^{2m}_\omega[-1,1]}$$

并且有：

1. 对于固定的 N，函数 f 越光滑（即 $2m$ 越大），收敛性越快，即**谱收敛**。
2. 若 $f(x)$ 是解析函数（无限光滑），则：

$$\|f - P_N f\| \leq Ce^{-\alpha N} \|f\|_{L^2}$$

3. 若 $f(x)$ 有间断，则会出现 Gibbs 现象。

L2 误差由下面的代码计算：

```
double l2_error(
    const std::function<double(double)> &f,
    const std::vector<double> &coeffs,
    const Polynomial &poly)
{
    auto residual = [&](double x)
    {
        double fx = f(x);
        auto phis = poly.eval_all(coeffs.size() - 1, x);
        double approx = 0;
        for (size_t k = 0; k < coeffs.size(); ++k)
            approx += coeffs[k] * phis[k];
        return (fx - approx) * (fx - approx);
    };
    return std::sqrt(poly.integrate(residual));
}
```

1.3 Legendre 多项式

Legendre 多项式 $P_n(x)$ 是区间 $[-1,1]$ 上权函数 $w(x) \equiv 1$ 的正交多项式：

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n}(x^2 - 1)^n$$

满足递推关系（对 $n > 0$）：

$$P_{n+1}(x) = \frac{2n+1}{n+1} x P_n(x) - \frac{n}{n+1} P_{n-1}(x)$$

前几项具体形式为：

$$P_0(x) = 1, \quad P_1(x) = x, \quad P_2(x) = \frac{3}{2}x^2 - \frac{1}{2}, \quad \dots$$

L2 误差如表1所示：

表 1: 不同函数在 Legendre 正交投影下的 L^2 误差

函数	$N=2$	$N=4$	$N=8$	$N=16$	$N=32$
$f_1(x)$	5.0088e-01	3.2354e-01	5.3275e-02	5.0222e-03	3.9416e-04
$f_2(x)$	1.0202e-01	5.0984e-02	2.2264e-02	8.8244e-03	3.2452e-03
$f_3(x)$	2.7579e-01	2.5962e-02	3.5557e-05	8.2104e-13	2.5188e-15
$f_4(x)$	7.0706e-01	5.3022e-01	3.8641e-01	2.7695e-01	1.9576e-01

对 $f_1(x)$,$f_2(x)$,$f_4(x)$ 画 $log(error)$ 与 $logN$ 的坐标图，对 $f_3(x)$ 画 $log(error)$ 与 N 的坐标图，如图1所示：

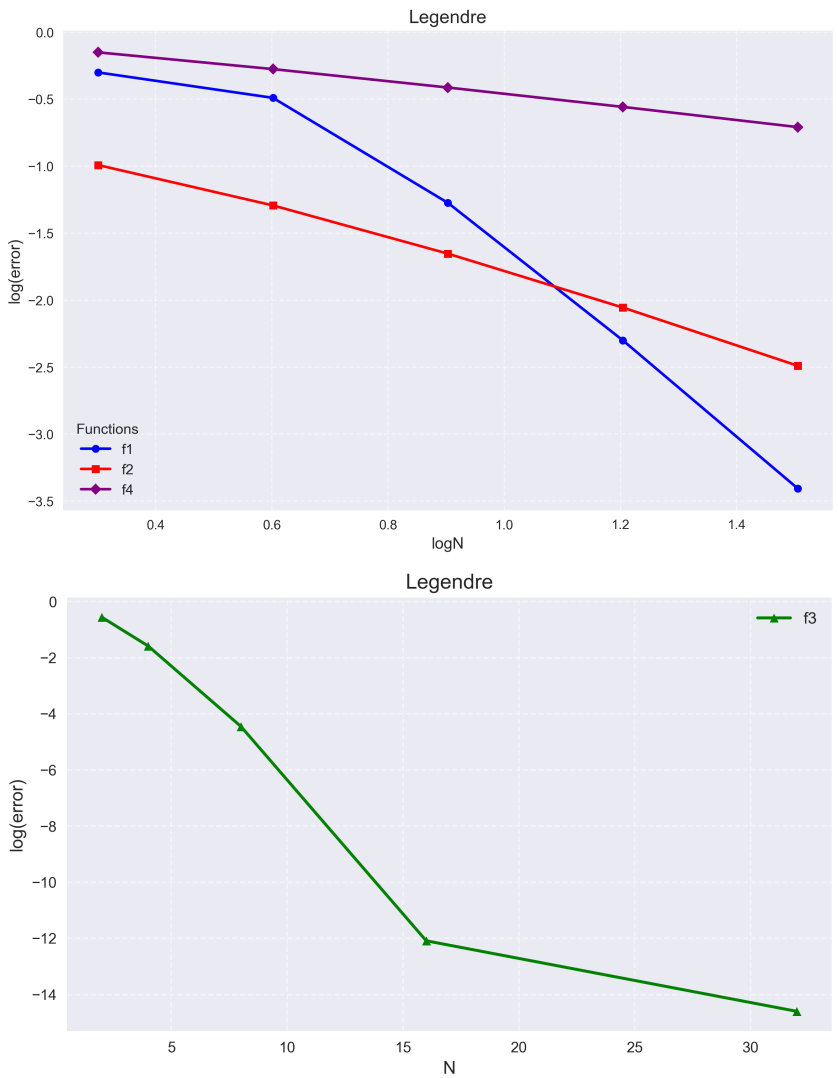

图 1: Legendre 投影的 L^2 误差

从图1可以看出，对于函数 $f_1(x)=|\sin(\pi x)|^3$,$f_2(x)=|x|$,$f_4(x)=sign(x)$，$log(error)=-klog(N)$，接近直线，这与理论分析相符合。并且，$|\sin(\pi x)|^3$ 在 $x=0$ 处是二阶可导的，比 $|x|$ 更加光滑，因此坐标图中 $f_1(x)$ 的斜率更陡，收敛更快。

而 $f_3(x)=\cos(\pi x)$ 是解析的，因此 $f_3(x)$ 的坐标图中 $log(error)=-kN$，同样接近直线。

我们还可以画出 Legendre 多项式的投影与原函数的对比，如图2所示，可以看出随着正交多项式最高阶数 N 的增大，投影结果越来越接近原函数。

同时，由于 $f(x) = sign(x)$ 是间断的，因此图像中出现了 Gibbs 现象。

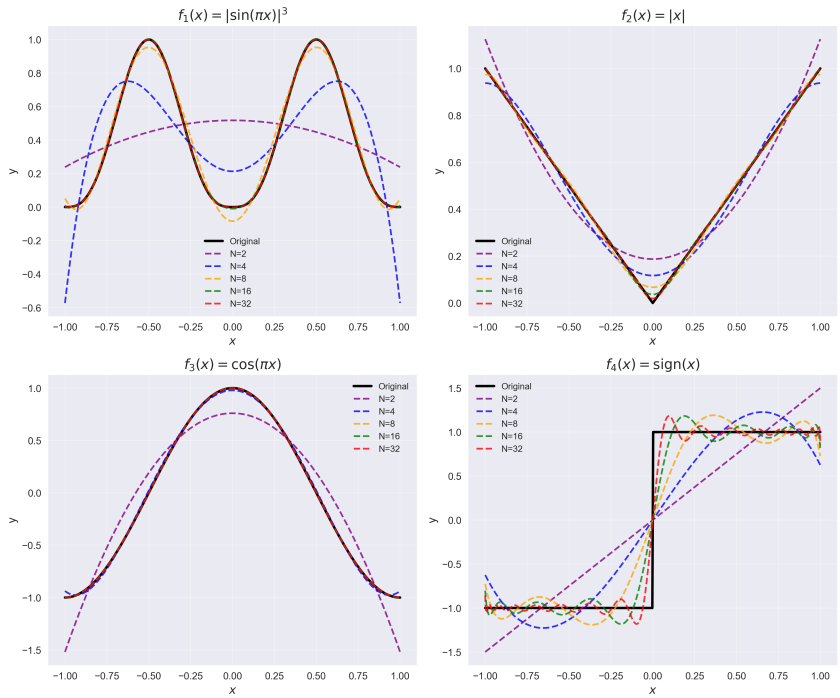

图 2: Legendre 投影与原函数

1.4 Hermite 多项式

Hermite 多项式是在区间 $(-\infty, \infty)$ 上关于权函数 $\omega(x) = \frac{1}{\sqrt{\pi}} e^{-x^2}$ 的正交多项式。

其递推关系为:

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x)$$

L2 误差如表2所示:

表 2: Hermite 正交投影下的 L^2 误差

函数	$N = 2$	$N = 4$	$N = 8$	$N = 16$	$N = 32$
$f_1(x)$	4.8453e-01	4.8452e-01	4.8368e-01	4.1403e-01	7.3055e-02
$f_2(x)$	1.9694e-01	1.2479e-01	7.4930e-02	4.2935e-02	2.2931e-02
$f_3(x)$	8.4753e-01	6.3508e-01	1.9094e-01	2.5303e-03	4.0499e-09
$f_4(x)$	7.9964e-01	6.7009e-01	5.5748e-01	4.5973e-01	3.7172e-01

画出 $log(error)$ 与 $logN$ 和 $log(error)$ 与 N 的坐标图如图3所示:

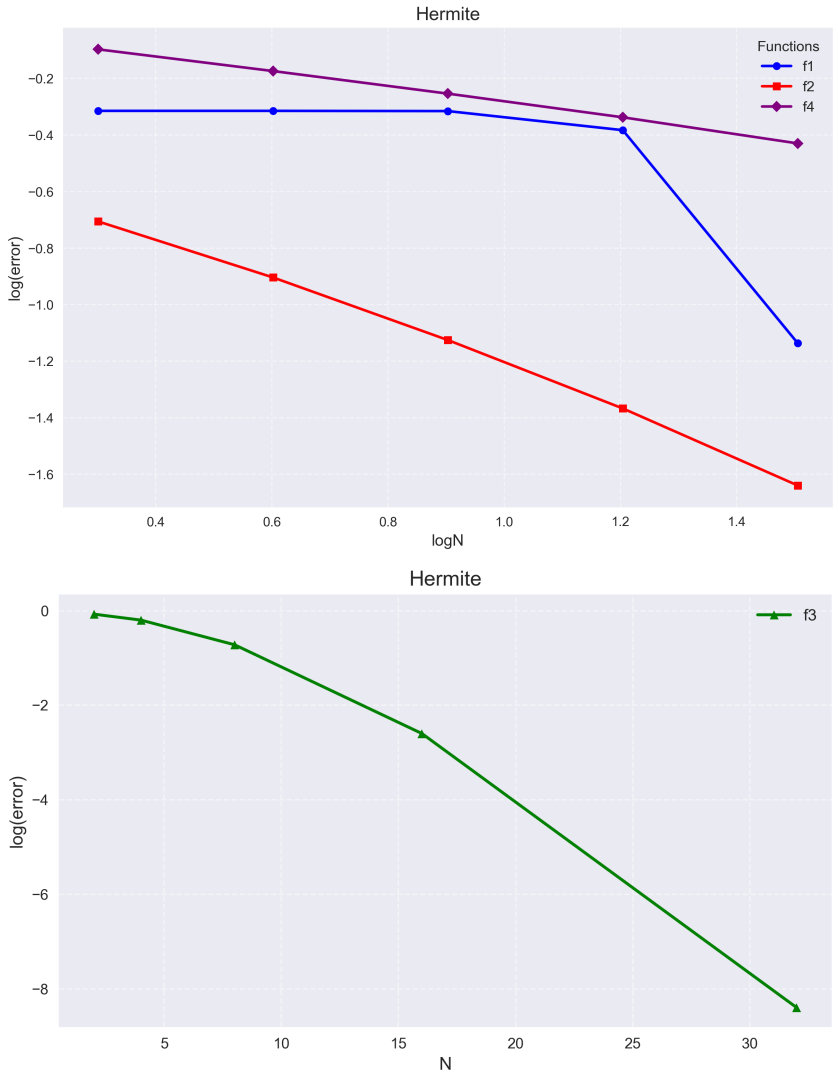

图 3: Hermite 投影的 L^2 误差

与 Legendre 多项式的结果一样，对于函数 $f(x) = |\sin(\pi x)|^3$,$f(x) = |x|$,$f(x) = sign(x)$,$log(error) = -klog(N)$，接近直线，并且 $|\sin(\pi x)|^3$ 比 $|x|$ 更加光滑，因此坐标图中 $f_1(x)$ 的斜率更陡，收敛更快。

而 $f(x) = \cos(\pi x)$ 是解析的，因此 $f_3(x)$ 的坐标图中 $log(error) = -kN$，同样接近直线。

我们同样画出 Hermite 多项式的投影与原函数的对比，如图4所示，与 Legendre 多项式的结果一致，由于 $f(x) = sign(x)$ 是间断的，图像中出现了 Gibbs 现象。

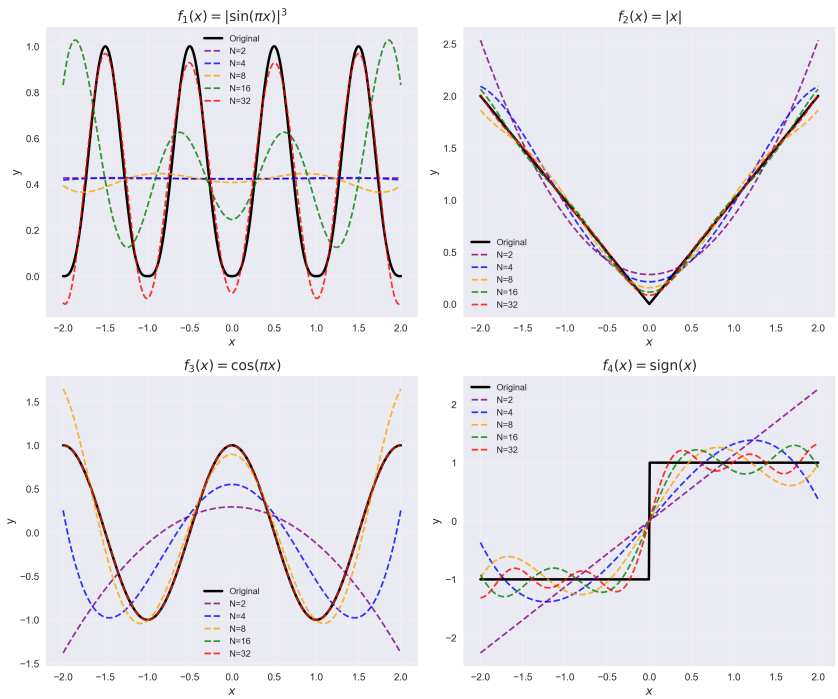

图 4: Hermite 投影与原函数

对比 Legendre 多项式与 Hermite 多项式的结果，对于本次测试的函数，在 $[-1, 1]$ 区间上，以 Legendre 多项式作为基函数的多项式投影与原函数更为接近，误差更小。

2 使用的第三方库

本次作业使用了 Boost 库（version 1.89.0）和 GSL 库（version 2.8）。参考网页链接：

- **Boost 库**：`https://www.boost.org/library/latest/math/`
- **GSL 库**：`https://www.gnu.org/software/gsl/doc/html/index.html`