数值分析第 5 次上机作业

学号：221840189，姓名：王晨光

§1 问题

求 $f(x)=e^x$ 在区间 $[0,1]$ 上的 n 次最佳平方逼近多项式（分以下四种情况）

$$\varphi(x)=\sum_{k=0}^{n}a_k\varphi_k(x),$$

- $n=5, \varphi_k(x)=x^k$;
- $n=5, \varphi_k(x)$ 为 k 次 Legendre 多项式;
- $n=10, \varphi_k(x)=x^k$;
- $n=10, \varphi_k(x)$ 为 k 次 Legendre 多项式;

§2 算法思路

取权函数 $W(x)=1$,要求 $f(x)=e^x$ 的 n 次最佳平方逼近多项式,即要求 $\varphi^*\in\Phi_n=\mathrm{span}\{\varphi_0,\varphi_1,\cdots,\varphi_n\}$，使得

$$\begin{aligned}\|f-\varphi^*\|_2^2&=\int_a^b[f(x)-\varphi^*(x)]^2\mathrm{d}x\\&=\inf_{\varphi\in\Phi_n}\int_a^b[f(x)-\varphi(x)]^2\mathrm{d}x\end{aligned}$$

令

$$\varphi(x)=a_0\varphi_0(x)+a_1\varphi_1(x)+\cdots+a_n\varphi_n(x)$$

$$F(a_0,a_1,\cdots,a_n)=\int_a^b[f(x)-\sum_{k=0}^{n}a_k\varphi_k(x)]^2\mathrm{d}x$$

则问题转化为求 F 的极小值

$$\frac{\partial F}{\partial a_j}=2\int_a^b[f(x)-\sum_{k=0}^{n}a_k\varphi_k(x)][-\varphi_j(x)]\mathrm{d}x=0,\ j=0,1,\cdots,n$$

由此可得

$$\sum_{k=0}^{n}(\varphi_j,\varphi_k)a_k=(f,\varphi_j),\quad j=0,1,\cdots,n$$

由此,只需储存矩阵 $G=((\varphi_j,\varphi_k))_{n+1}$ 与列向量 $f_{ph}=((f,\varphi_0),\cdots,(f,\varphi_n))^T$,令 $a=(a_0,a_1,\cdots,a_n)^T$，再解线性方程组 $G\cdot a=f_{ph}$ 即可求出 $a_j,j=0,1,\cdots,n$.

当 $\varphi_k(x)$ 为 Legendre 多项式时，由于 $\{\varphi_j(x)\}_{j=0}^{n}$ 为直交函数系，$G=((\varphi_j,\varphi_k))_{n+1}$ 为对角阵，故可直接用向量储存对角线元素.

Algorithm 1 返回 a 以及逼近函数图像

Require: 逼近次数 n

Ensure: 系数向量 a

```
function CoefficientAcquisition(n)
    初始化 G 矩阵或向量以及 f_ph 向量
    for i from 0 to n do
        for j from 0 to n do
            G_{i,j} ← (φ_i, φ_j)
        end for
        f_ph^i ← (f, φ_i)
    end for
    a ← G^{-1} · f_ph
    φ ← Σ_{i=0}^{n} a_i φ_i(x)
    作出 φ 与 f 的函数图像
    return a
end function
```

§ 3　结果分析

以下为程序测试结果，返回了图像以及对应的系数 a。

1. 当 $n = 5$ 时

i	a_i
0	0.9999975939481748
1	1.000099801486657
2	0.49901917513980787
3	0.1704895392633275
4	0.03480111544339166
5	0.013872004898576736

表 1: $n = 5, \varphi_i(x) = x^i$ 系数

i	a_i
0	1.1732975422355196
1	1.1083386696360753
2	0.3525658021585433
3	0.07436115096015425
4	0.007954540780936126
5	0.0017615244408117519

表 2: $n = 5$,Legendre 多项式系数

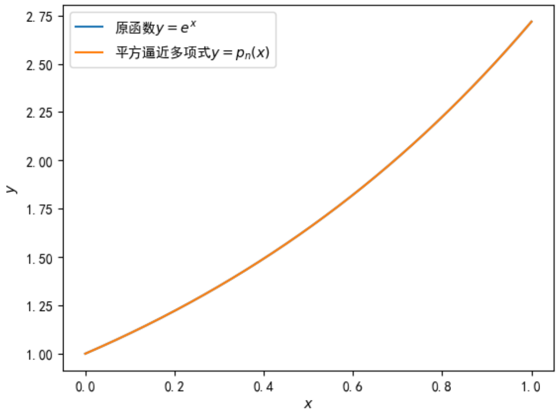

图 1: $n = 5$ 时 x^k 逼近

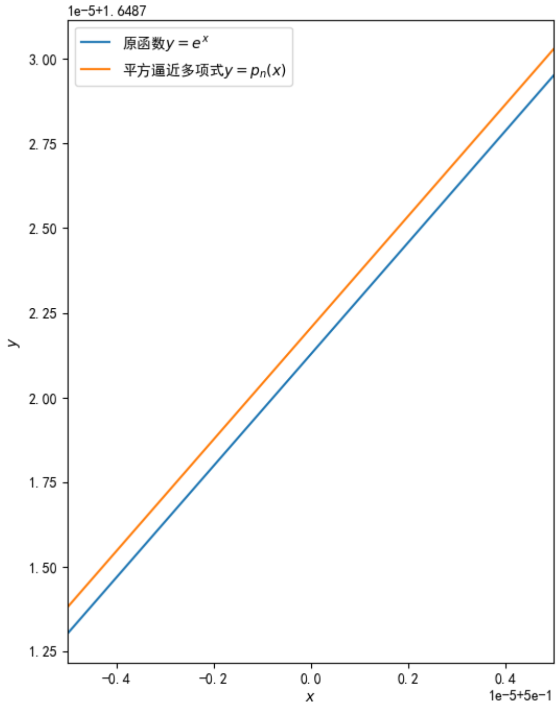

图 2: $n = 5$ 时 x^k 逼近 (局部)

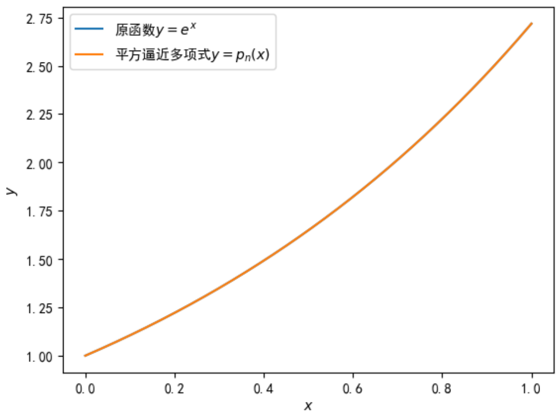

图 3: $n = 5$ 时 Legendre 多项式逼近

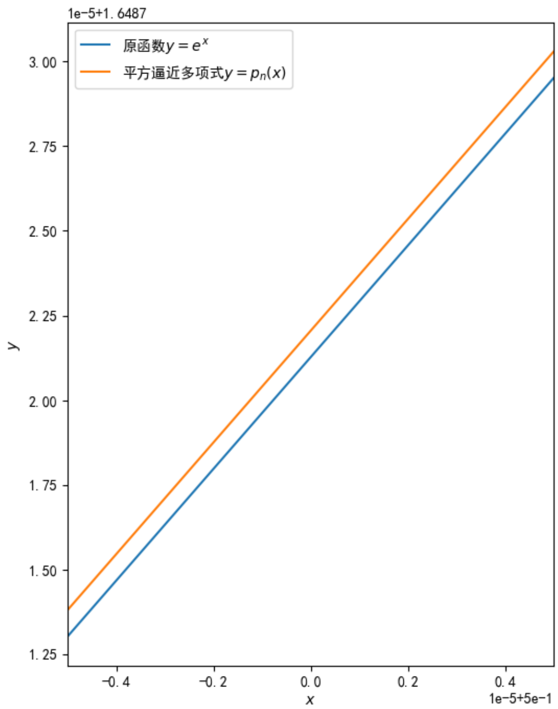

图 4: $n = 5$ 时 Legendre 多项式逼近 (局部)

2. 当 $n = 10$ 时

i	a_i
0	1.0000000012113701
1	0.9999998681210073
2	0.5000035268463334
3	0.16662625984232918
4	0.04191227935312167
5	0.007455096067967402
6	0.0033287923143171866
7	-0.002479244981759852
8	0.0022732391669669206
9	-0.0010475780120787397
10	0.00020958957362995516

表 3: $n = 10$, $\varphi_i(x) = x^i$ 系数

i	a_i
0	1.17610487769128
1	1.1012448045699013
2	0.3609156609890722
3	0.06749975436161927
4	0.01221017595693133
5	-0.0002873295948090008
6	0.0007933224883379857
7	-0.0002668997568865246
8	8.219614376558308e-05
9	-1.6453268506992322e-05
10	1.7202942628052338e-06

表 4: $n = 10$, Legendre 多项式系数

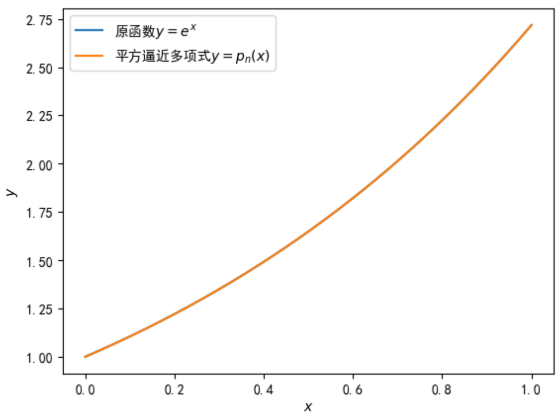

图 5: $n = 10$ 时 x^k 逼近

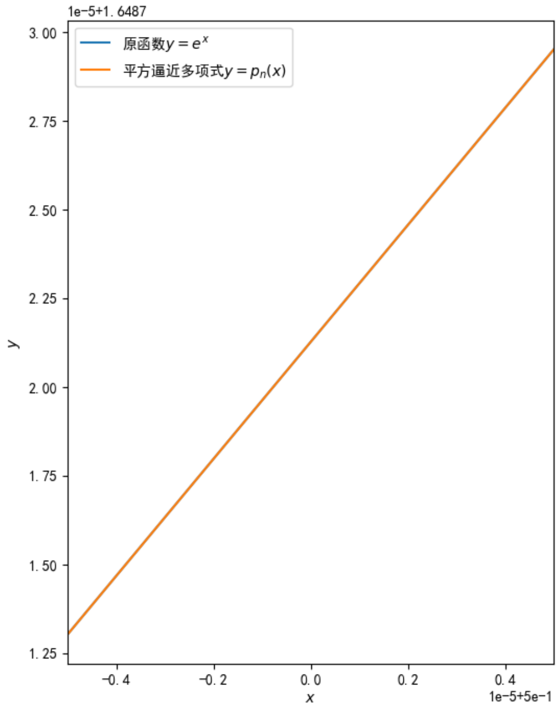

图 6: $n = 10$ 时 x^k 逼近 (局部)

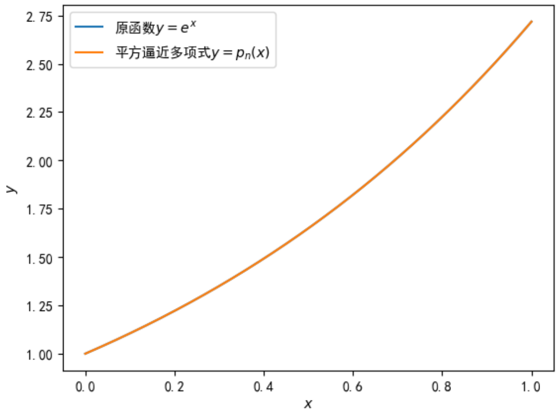

图 7: $n = 10$ 时 Legendre 多项式逼近

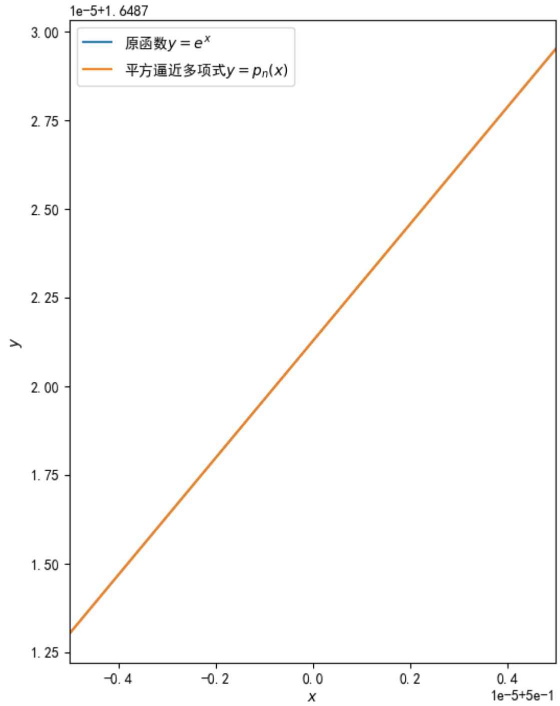

图 8: $n = 10$ 时 Legendre 多项式逼近 (局部)

§ 4 结论

可以看出，最佳平方逼近可以很好地逼近原函数，且在函数系选取相同的情况下，次数越高，逼近效果越好。

§5 附录：程序代码

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import legendre
from scipy.integrate import quad
# display Chinese in gragh
plt.rcParams['font.sans-serif']=['SimHei']
# display "-" in gragh
plt.rcParams['axes.unicode_minus']=False

# 定义原函数
def f(x):
    return np.exp(x)

# 定义正交多项式
def phi(n, k, x, flag):
    if k == 0:
        return np.ones_like(x)
    elif k == 1:
        return x
    elif k > 1:
        if flag == 0:
            return x**k
        elif flag == 1:
            # 使用Legendre多项式
            p = legendre(k)
            return p(x)

# 计算最佳平方逼近多项式系数
def compute_coefficients(n, flag):
    A = np.zeros((n+1, n+1))
    b = np.zeros(n+1)
    for i in range(n+1):
        for j in range(n+1):
            integrand = lambda x: phi(n, i, x, flag) * phi(n, j, x, flag)
            A[i, j], _ = quad(integrand, 0, 1)
        integrand_f = lambda x: f(x) * phi(n, i, x, flag)
        b[i], _ = quad(integrand_f, 0, 1)
    return np.linalg.solve(A, b)

# 生成逼近多项式函数
def approximation_function(n, x, flag):
    coefficients = compute_coefficients(n, flag)
    result = np.zeros_like(x)
```

```
    for i in range(n+1):
        result += coefficients[i] * phi(n, i, x, flag)
        print(coefficients[i])
    return result

# 绘制图像
def plot_comparison(n, flag):
    x = np.linspace(0, 1, 1000)
    plt.plot(x, f(x), label='原函数$y=e^x$')
    plt.plot(x, approximation_function(n, x, flag), label='平方逼近多项式$y=p_{n}(
    plt.xlabel('$x$')
    plt.ylabel('$y$')
    # plt.title('Comparison between original and approximation function (n={})'.fo
    plt.legend()
    plt.show()

# 绘制局部放大图像
def plot_local_enlargement(n, flag):
    x = np.linspace(0.499995, 0.500005, 1000)
    plt.figure(figsize=(6, 8))
    plt.plot(x, f(x), label='原函数$y=e^x$')
    plt.plot(x, approximation_function(n, x, flag), label='平方逼近多项式$y=p_{n}(
    plt.xlabel('$x$')
    plt.ylabel('$y$')
    # plt.title('Comparison between original and approximation function (n={})'.fo
    plt.xlim(0.499995, 0.500005)
    plt.legend()
    plt.show()

# plot the gragh and print the coefficients
plot_comparison(5,0)  # n=5, x^k
plot_comparison(5,1)  # n=5, Legendre polynomial
plot_comparison(10,0) # n=10, x^k
plot_comparison(10,1) # n=10, Legendre polynomial
plot_local_enlargement(10, 0)
plot_local_enlargement(5, 0)
```