# 数值分析上机实验

函数逼近、数据拟合

## 1 问题描述

本次实验的目的利用最佳平方逼近、Tchebychev 截断级数的办法和 Lagrange 插值余项极小化方法来对函数 $f(x) = x^2 \log(2+x), x \in [-1,1]$ 进行逼近.

## 2 实验内容

对于最佳平方逼近，我们可以借助 Legendre 多项式（正交多项式）进行正交投影. 事实上，我们可以考虑由不超过三次的实系数多项式组成的空间：

$$\mathcal{P}_3[-1,1] = \mathrm{Span}(1, x, x^2, x^3), \tag{1}$$

我们选择 Legendre 多项式作为 $\mathcal{P}_3[-1,1]$ 的正交基：

$$\phi_0(x) = 1, \quad \phi_1(x) = x, \quad \phi_2(x) = \frac{3}{2}x^2 - \frac{1}{2}, \quad \phi_3(x) = \frac{5}{2}x^3 - \frac{3}{2}x,$$

此时最佳平方逼近多项式按照下面的公式计算：

$$S_3(x) = \sum_{k=0}^{3} \frac{(f, \phi_k)}{(\phi_k, \phi_k)} \phi_k \tag{2}$$

满足条件：

$$S_3 = \mathrm{argmin}_{S \in \mathcal{P}_3[-1,1]} \|f - S\|, \tag{3}$$

在上面的公式中，$(\cdot,\cdot)$ 指的就是通常 $L^2[-1,1]$ 的内积（权函数为 1），$\|\|$ 指的是由内积诱导的范数.

而对于 Tchebychev 截断级数的方法，则是考察带有权 $\rho(x) = \frac{1}{\sqrt{1-x^2}}$ 的内积，相应地，选择的正交多项式换成 Tchebychev 多项式 $T_k$，我们在本实验中采用一种容易编程的表达式：

$$T_k(x) = \cos(k \arccos(x)), \tag{4}$$

对于以上带权内积下的广义 Fourier 级数，其截断函数：

$$R_3(x) = \frac{c_0}{2} + \sum_{k=1}^{3} c_k T_k(x), \tag{5}$$

这里

$$c_k = \frac{2}{\pi}(f, T_k)_{\rho(x)=\frac{1}{\sqrt{1-x^2}}}, \quad k = 0, 1, \cdots \tag{6}$$

可以证明以上截断函数是 $f(x)$ 的最佳一致逼近的一个好的近似.

我们也可以通过调整插值节点来构造好的插值多项式来得到近似最佳一致逼近. 假设我们在节点 $x_1, \cdots, x_n$ 上插值，则 Lagrange 插值余项公式为：

$$r_{n-1}(x) = f(x) - L_{n-1}(x) = \frac{1}{n!} f^{(n)}(\xi)\omega_n(x), \tag{7}$$

这里 $\omega_n(x) = \prod_{j=1}^{n}(x - x_j)$，记 $M = \left\| f^{(n)} \right\|_\infty$，由此有：

$$\|r_{n-1}\|_\infty \leq \frac{M_n}{n!}\|\omega_n\|_\infty, \tag{8}$$

要极小化插值余项，一个方法是使 $\|\omega_n\|_\infty$ 尽可能小. 我们知道 $\omega_n(x)$ 为首项系数为 1 的多项式. 而首项系数为 1 的多项式中，模最小的多项式应该是 $2^{1-n}T_n(x)$. 即我们应该让 $\omega_n(x) = 2^{1-n}T_n(x)$，即取插值节点 $x_1, \cdots, x_n$ 为 $T_n(x)$ 的零点. 一旦这样取，我们就有：

$$\|r_{n-1}\| \leq \frac{M_n}{n!2^{n-1}}, \tag{9}$$

也就是，这时 $L_{n-1}(x)$ 可以看成是 $f$ 的最佳一直逼近函数的一个很好的近似.

本题中我们选用的插值多项式为 $L_3(x)$，也就是要取 $T_4(x) = 8x^4 - 8x^2 + 1$ 的零点 $x_j = \pm\sqrt{\frac{2\pm\sqrt{2}}{4}}$ 作为插值节点.

以上三种逼近函数的代码实现如下，实现思路见代码注释：

```
import numpy as np
from scipy import integrate
import matplotlib.pyplot as plt
import math

#function $f(x) = x^2\log(2+x)$
def fun(x):
    return x**2 * np.log(2+x)

#zeros of Tchebychev polynomial T_4
t_zeros = [np.sqrt((2+np.sqrt(2))/4), np.sqrt((2-np.sqrt(2))/4), -np.sqrt((2+np.sqrt
    (2))/4), -np.sqrt((2-np.sqrt(2))/4)]
```

```
    def innerproduct(fun1, fun2):
    """innerproduct with rho = 1"""
    result,_ = integrate.quad(lambda x:fun1(x)*fun2(x), -1, 1)
    return result
def square_approx(x):
    """orthogonal projection for square approx."""
    value = innerproduct(lambda x:1, fun)/innerproduct(lambda x:1, lambda x:1) * 1\
            + innerproduct(lambda x:x, fun)/innerproduct(lambda x:x, lambda x:x) * x \
            + innerproduct(lambda x:(1.5*x**2-0.5), fun)/innerproduct(lambda x:(1.5*x**2-0.5), lambda x:(1.5*x**2-0.5)) * (1.5*x**2-0.5)\
            + innerproduct(lambda x:(2.5*x**3-1.5*x), fun)/innerproduct(lambda x:(2.5*x**3-1.5*x),lambda x:(2.5*x**3-1.5*x)) * (2.5*x**3-1.5*x)
    return value
def T_polynomial(x,n):
    """Tchebychev polynomials"""
    x_reg = x
    if x > 1:
        x_reg = 1
    if x < -1:
        x_reg = -1
    return np.cos(n*np.arccos(x_reg))
def weight(x):
    return 1/np.sqrt(1-x**2)
def weight_innerproduct(fun1, fun2):
    """weighted innerproduct with rho = 1/sqrt(1-x^2)"""
    result,_ = integrate.quad(lambda x:fun1(x)*fun2(x)*weight(x), -1, 1)
    return (2/math.pi)*result
def truncate_fourier(x):
    """truncated Tchebychev Fourier series for approximated best uniform approx."""
    clist = [weight_innerproduct(lambda x: T_polynomial(x,k), fun) for k in range(4)]
    return clist[0]/2 + clist[1]*T_polynomial(x,1) + clist[2]*T_polynomial(x,2) + clist[3]*T_polynomial(x,3)
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 = t_zeros
    prod = np.prod([(x-nodes[i])/(nodes[k]-nodes[i]) for i in range(len(nodes)) if i != k])
    return prod
def Ln(x, n=3):
```

```
    """p_n(x) = \sum_{k=0}^n y_k l_k(x)"""
    nodes = t_zeros
    lk_basis = np.array([lk(x, nodes, n, k) for k in range(len(nodes))])
    yk_list = np.array([fun(nodes[k]) for k in range(len(nodes))])
    p_n_x = lk_basis @ yk_list
    return p_n_x
```

# 3 实验结果及讨论

我们运行以上代码，并画出三种逼近函数的图像，如图1所示.

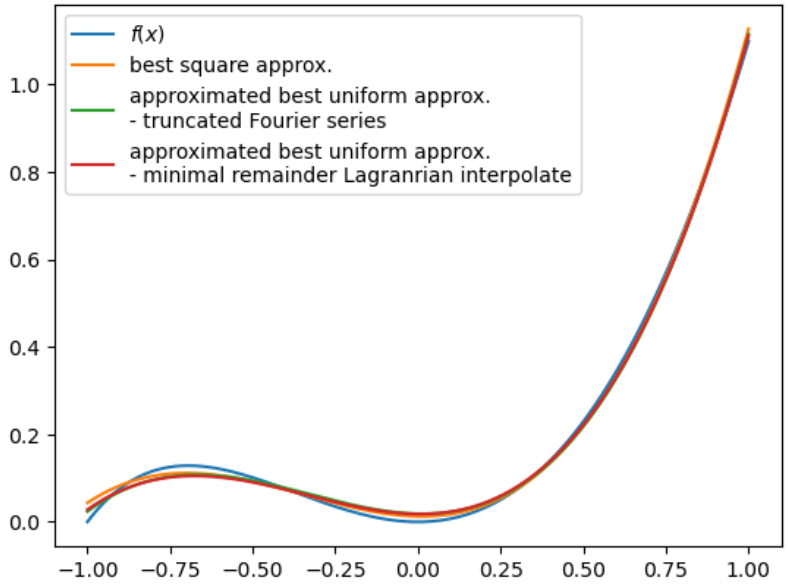


图 1: 三种逼近函数的图像

从图像中可以看出，三种逼近都可以很好地对原函数进行拟合，而且，三种逼近所用到的“自由度”都很小（都是在 $\mathcal{P}_3[-1,1]$ 中做逼近），多项式阶数很低，使用起来很方便.

为了比较不同格式函数逼近的不同表现，我们对误差函数 $S^* - f$ 进行作图，其中 $S^*$ 表示用不同方法得到的逼近函数，函数图像如2所示

可见三种逼近格式的误差函数的无穷范数都能大致控制在 0.04 以内，可认为三种逼近都实现了良好的一致逼近.

另外，三种逼近格式都是在 0 附近误差较小，而靠近 $\pm 1$ 时误差较大，整体来看，最佳平方逼近的表现略好于其他两种逼近格式，而使用截断广义 Fourier 级数和极小化插值余项来近似最佳平方逼近的表现类似.

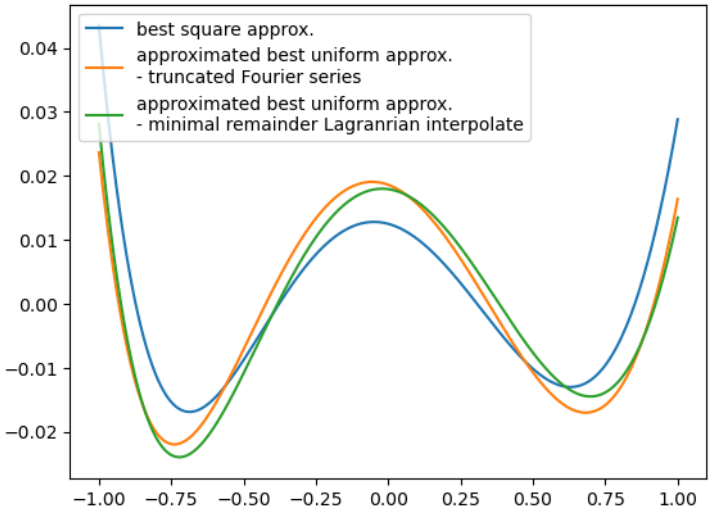
best square approx.
approximated best uniform approx.
- truncated Fourier series
approximated best uniform approx.
- minimal remainder Lagranrian interpolate
0.04
0.03
0.02
0.01
0.00
−0.01
−0.02
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

图 2: 误差函数