函数逼近

实验内容

本实验要求实现最佳平方逼近与最小二乘拟合，并完成两种方法之前的对比。输入区间 $[a,b]$ ，参数 c 作为标准函数 $f(x)=\frac{1}{cx^2+1}$ 的值，参数 k 作为所构造的逼近多项式的次数$(k=1,2,3)$ 。参数 $n+1$ 作为采样点的个数，参数 m 作为试验点的个数。要求选用勒让德多项式作最佳平方逼近；在区间 $[a,b]$ 上均匀采集 n 个采集点，利用这 $n+1$ 个采集点，计算采集点上的函数值，构造最小二乘拟合多项式函数。之后再选取 m 个点作为实验点，计算在这 m 个实验点上所构造的逼近函数与给定的目标函数 $f(x)$ 的平均误差。同时对比两种逼近方法之间的精度差异。

实验原理

基于勒让德多项式的最佳平方逼近

由最佳平方函数逼近的定义并对函数式中每个未知系数求偏导可得以下方程组：

$$\sum_{j=0}^{n}(\varphi_k(x),\varphi_j(x))a_j=(f(x),\varphi_k(x))$$

当使用勒让德多项式作为逼近函数的基时，逼近函数可以表示为：

$$S_n^*(x)=a_0^*P_0(x)+a_1^*P_1(x)+\ldots+a_n^*P_n(x)$$

其中每个系数可以表示为 $a_k^*=\frac{(f(x),P_k(x))}{(P_k(x),P_k(x))}=\frac{2k+1}{2}\int_{-1}^{1}f(x)P_k(x)dx$

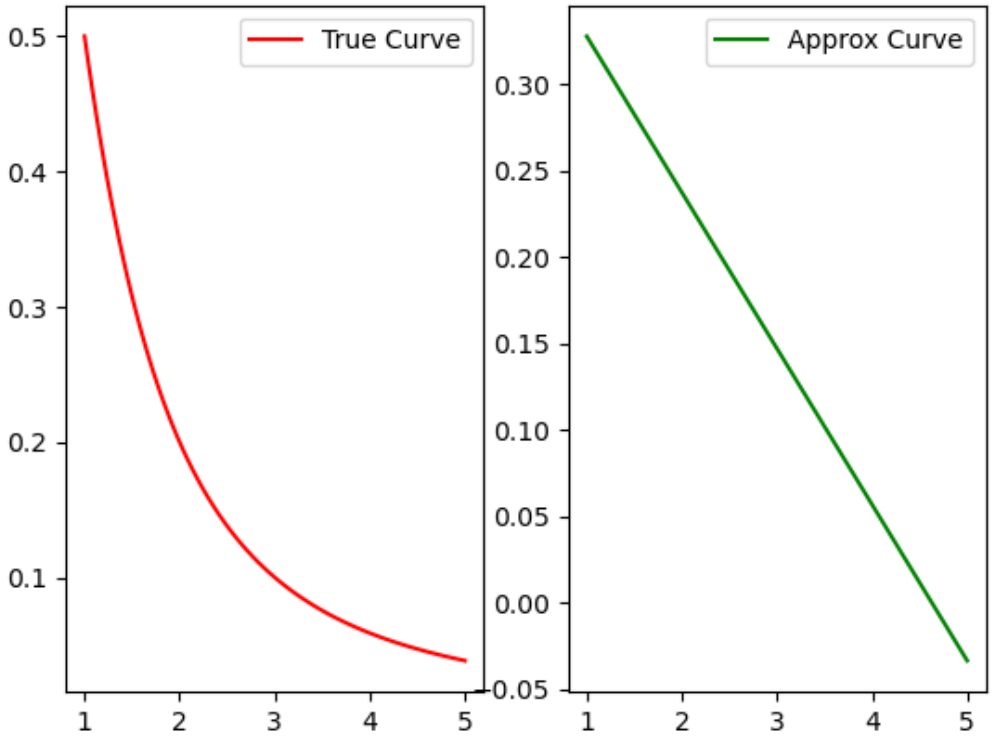
最小二乘法拟合

最小二乘法拟合即不给定目标函数的表达式，只有离散的点进行拟合，因此在这种情况下我们可以将内积由积分换成求和再次解线性方程组即可。

实验结果

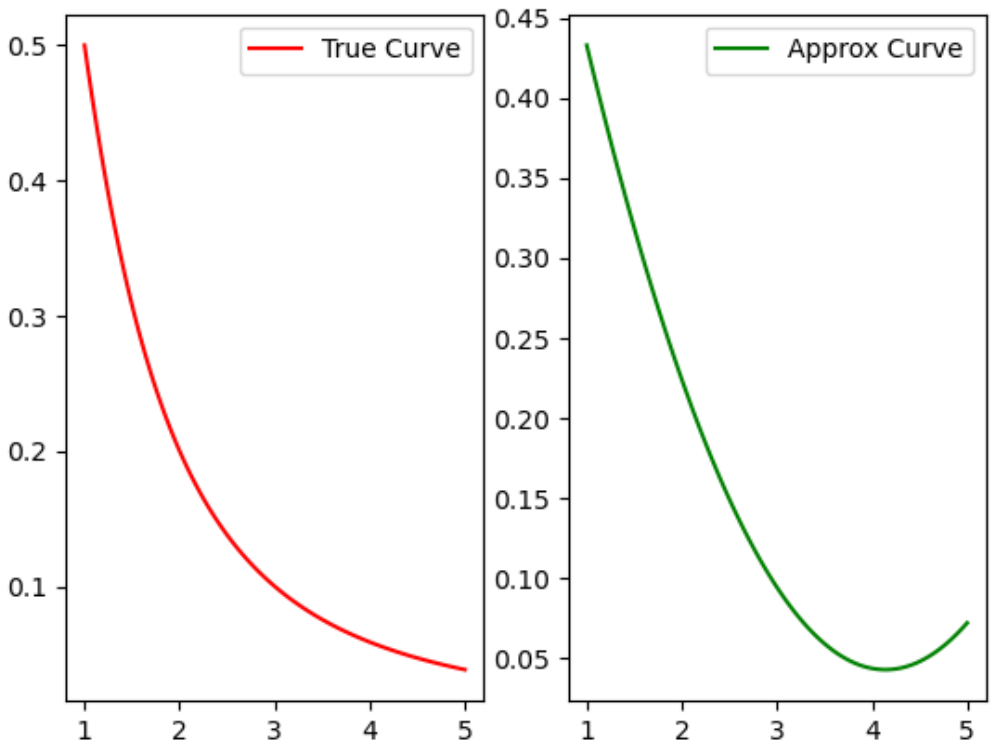
基于勒让德多项式的最佳平方逼近

k = 1



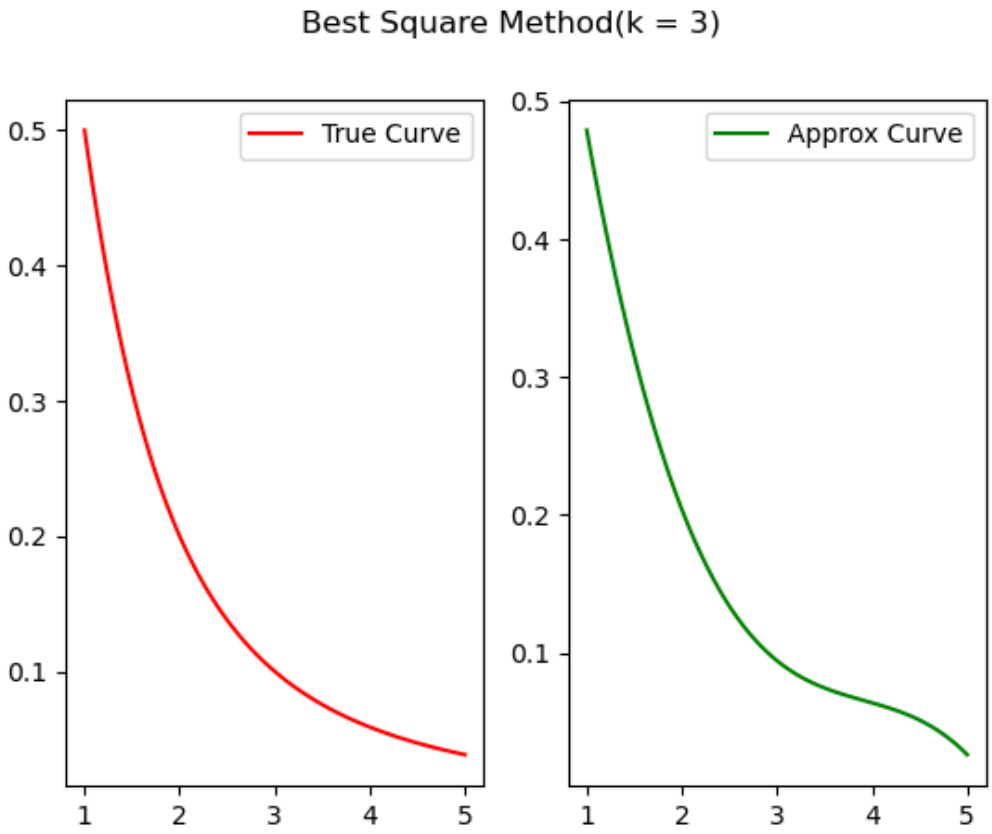

```
PS F:\KuangjuX\作业\数值计算\Numerical-Analysis> python .\example.py
标准函数计算的结果为: 0.22035677887337105, 逼近函数计算的结果为: 0.24803390473434966, 误差为: 0.02767712586097862
标准函数计算的结果为: 0.34979009715716053, 逼近函数计算的结果为: 0.2947651838276111, 误差为: 0.05502491332954945
标准函数计算的结果为: 0.08382400410225675, 逼近函数计算的结果为: 0.11937103513425469, 误差为: 0.03554703103199794
标准函数计算的结果为: 0.2282960986378309, 逼近函数计算的结果为: 0.25186477245979255, 误差为: 0.02356867382196165
标准函数计算的结果为: 0.16079660795672504, 逼近函数计算的结果为: 0.21159942493875372, 误差为: 0.05080281698202868
标准函数计算的结果为: 0.04792240674156511, 逼近函数计算的结果为: 0.015429368801307991, 误差为: 0.03249303794025712
标准函数计算的结果为: 0.263021084542846, 逼近函数计算的结果为: 0.2667299010140250S, 误差为: 0.003708816471179044
标准函数计算的结果为: 0.05291854755285122, 逼近函数计算的结果为: 0.035903680091916995, 误差为: 0.017014867460934224
标准函数计算的结果为: 0.28564490665593517, 逼近函数计算的结果为: 0.27508175569904203, 误差为: 0.010563150956893141
标准函数计算的结果为: 0.13008525380398628, 逼近函数计算的结果为: 0.18438192278048643, 误差为: 0.05429666897650015
```

k = 2




```
PS F:\KuangjuX\作业\数值计算\Numerical-Analysis> python .\example.py
标准函数计算的结果为: 0.25444838457403973, 逼近函数计算的结果为: 0.27622781810468133, 误差为: 0.0217794335306416
标准函数计算的结果为: 0.15973066999180463, 逼近函数计算的结果为: 0.17775838235016464, 误差为: 0.01802771235836001
标准函数计算的结果为: 0.3698260741314304, 逼近函数计算的结果为: 0.3608981456535344, 误差为: 0.00892792847789603
标准函数计算的结果为: 0.3334539761609791, 逼近函数计算的结果为: 0.33701998656905074, 误差为: 0.003566010408071640
标准函数计算的结果为: 0.05338562834897648, 逼近函数计算的结果为: 0.04293358613943383, 误差为: 0.010452042209542649
标准函数计算的结果为: 0.061833108889099896, 逼近函数计算的结果为: 0.04512047960593761, 误差为: 0.016712629283162285
标准函数计算的结果为: 0.041896564839059916, 逼近函数计算的结果为: 0.059027439702572806, 误差为: 0.017130874863512890
标准函数计算的结果为: 0.1596156233273114, 逼近函数计算的结果为: 0.17761462588248614, 误差为: 0.017799002555174726
标准函数计算的结果为: 0.11645419998146907, 逼近函数计算的结果为: 0.11878175191406826, 误差为: 0.0023275519325991895
标准函数计算的结果为: 0.06367849489537915, 逼近函数计算的结果为: 0.046443638866101235, 误差为: 0.017234856029277913
```

k = 3




```
PS F:\KuangjuX\作业\数值计算\Numerical-Analysis> python .\example.py
标准函数计算的结果为: 0.4653868197196585, 逼近函数计算的结果为: 0.4519708749732728, 误差为: 0.013415944746385688
标准函数计算的结果为: 0.1231567621214545, 逼近函数计算的结果为: 0.11762376251162603, 误差为: 0.005532999609828476
标准函数计算的结果为: 0.468695462756992, 逼近函数计算的结果为: 0.4545942485763822, 误差为: 0.014101214180609778
标准函数计算的结果为: 0.23800268106713607, 逼近函数计算的结果为: 0.24531061869278675, 误差为: 0.007307937625650684
标准函数计算的结果为: 0.09455175274085441, 逼近函数计算的结果为: 0.08929032709940611, 误差为: 0.005261425641448297
标准函数计算的结果为: 0.0744981695421665, 逼近函数计算的结果为: 0.07375975985101463, 误差为: 0.0007384096911518673
标准函数计算的结果为: 0.33302402775564216, 逼近函数计算的结果为: 0.33926431039361715, 误差为: 0.006240282637974992
标准函数计算的结果为: 0.0816937603021533, 逼近函数计算的结果为: 0.07883003198262982, 误差为: 0.002863728319523487
标准函数计算的结果为: 0.12944659387410404, 逼近函数计算的结果为: 0.12444314943588833, 误差为: 0.005003444438215709
标准函数计算的结果为: 0.12525106005674005, 逼近函数计算的结果为: 0.11987872237811156, 误差为: 0.005372337678628491
```

以上是我们分别使用 k = 1, 2, 3 基于勒让德多项式对于标准函数在 $[1, 5]$ 区间之间的拟合，并随机在区间内选取10个点进行比较并计算误差。可以看到在 k 在 1 到 3 之间的时候随着 k 的增长误差是逐渐减小的。

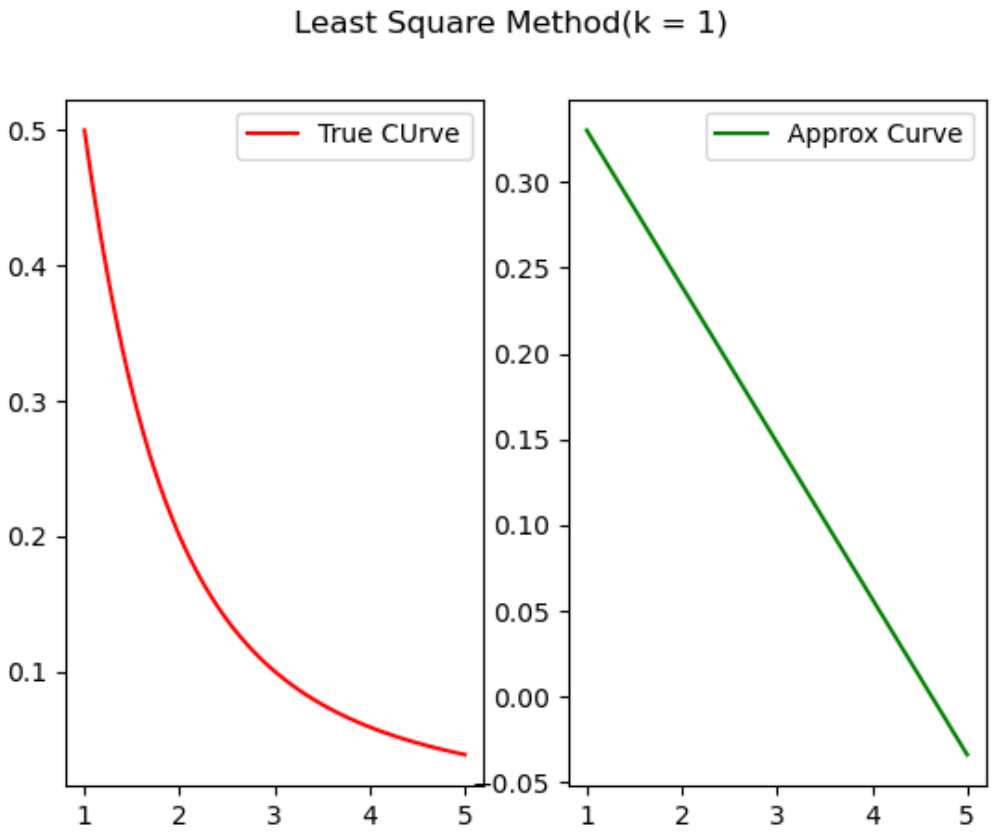
使用 **Python 的实现** 如下所示：

```
    def legrand_fit(self):
        # 使用勒让德多项式作为正交多项式进行拟合
        self.coefficients = []
        for i in range(0, self.k + 1):
            (res, _) = integrate.quad(mul_fn, -1, 1, args=(legendre(i), self.a,
self.b, self.c))
            res *= ((2 * i + 1) / 2)
            self.coefficients.append(res)
        for i in range(0, self.k + 1):
            self.f += (self.coefficients[i] * legendre(i))
```

其中我们的 `mul_fn` 函数可以将勒让德多项式与目标多项式相乘，并利用 `scipy` 的 `integrate` 来求积分。

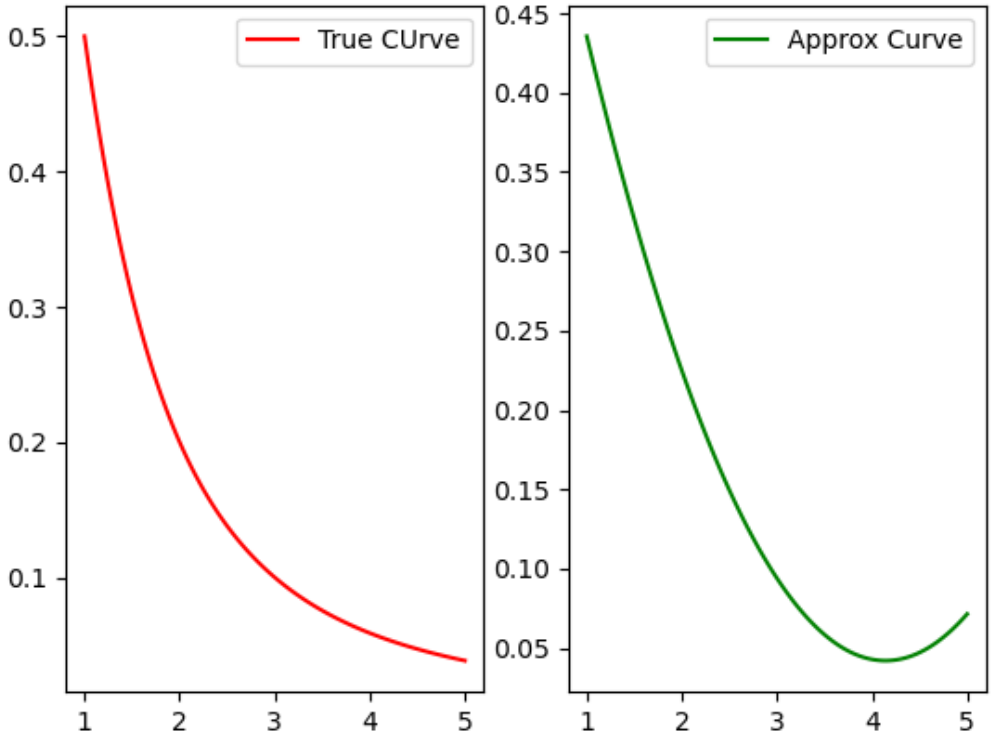
最小二乘法拟合

k = 1



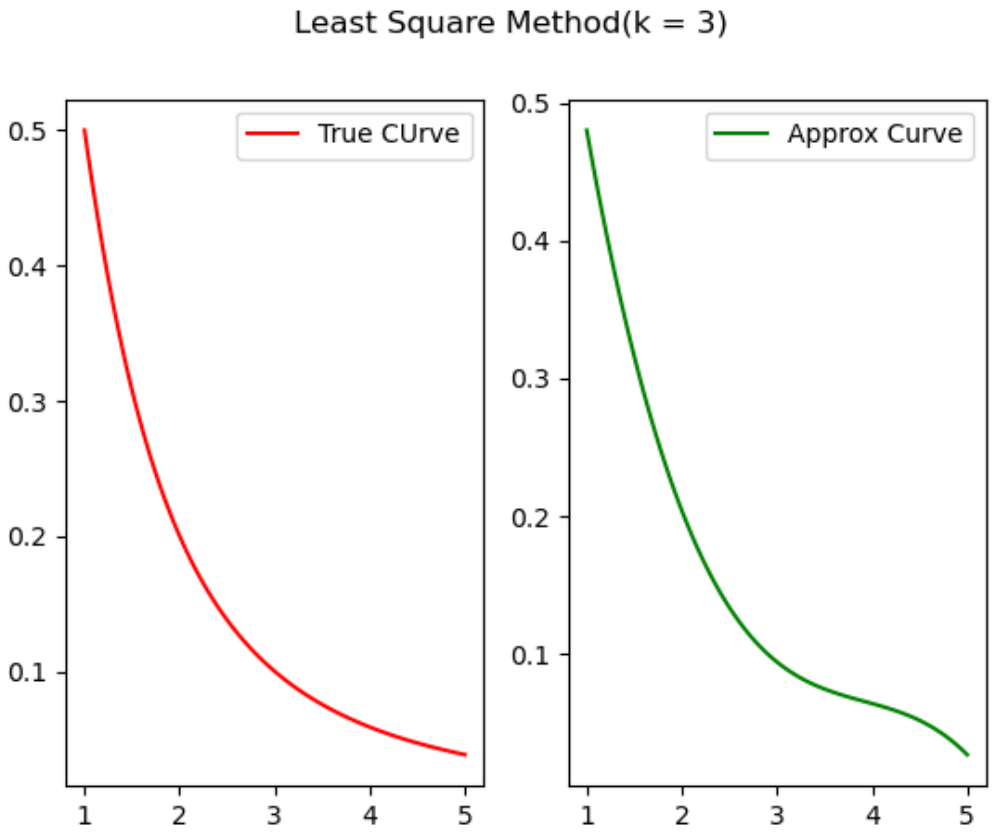

```
PS F:\KuangjuX\作业\数值计算\Numerical-Analysis> python .\example.py
标准函数计算的结果为：0.36305753146399133，逼近函数计算的结果为：0.30075640081066396，误差为：0.062301130653327375
标准函数计算的结果为：0.04008093479675029，逼近函数计算的结果为：-0.02418116153679195，误差为：0.06426209633354224
标准函数计算的结果为：0.038783745996050625，逼近函数计算的结果为：-0.03187637021160605，误差为：0.07066011620765668
标准函数计算的结果为：0.07734579227721675，逼近函数计算的结果为：0.1069114291873906，误差为：0.02956563691017386
标准函数计算的结果为：0.14346301073991335，逼近函数计算的结果为：0.19889367176414985，误差为：0.055430661024236494
标准函数计算的结果为：0.37187212160710953，逼近函数计算的结果为：0.3030213397065342，误差为：0.06885078190057531
标准函数计算的结果为：0.18143435222837864，逼近函数计算的结果为：0.2279696826750449，误差为：0.046535330446666245
标准函数计算的结果为：0.05867706095375111，逼近函数计算的结果为：0.05670732700041664，误差为：0.001969733953334468
标准函数计算的结果为：0.17701241559331402，逼近函数计算的结果为：0.2250412592371785，误差为：0.04802884364386448
标准函数计算的结果为：0.4049967719779343，逼近函数计算的结果为：0.3109930085410856，误差为：0.09400376343684874
```

k = 2




```
PS F:\KuangjuX\作业\数值计算\Numerical-Analysis> python .\example.py
标准函数计算的结果为：0.4585046235382813，逼近函数计算的结果为：0.41416749238799067，误差为：0.044332969965837465
标准函数计算的结果为：0.12344864171685886，逼近函数计算的结果为：0.12899007941959528，误差为：0.005541437702736421
标准函数计算的结果为：0.08042596572092252，逼近函数计算的结果为：0.06506040232742571，误差为：0.015365563393496803
标准函数计算的结果为：0.04224873550780399，逼近函数计算的结果为：0.05737254980138007，误差为：0.01512381429357608
标准函数计算的结果为：0.049505044566261085，逼近函数计算的结果为：0.04434496978286595，误差为：0.005160074783395134
标准函数计算的结果为：0.08038225603827577，逼近函数计算的结果为：0.06499985459378388，误差为：0.015382401444491892
标准函数计算的结果为：0.4433330968983807，逼近函数计算的结果为：0.4059758821191686，误差为：0.0373572147792121l
标准函数计算的结果为：0.05579328418961708，逼近函数计算的结果为：0.04206527810097105，误差为：0.013728006088646029
标准函数计算的结果为：0.2518539570220063，逼近函数计算的结果为：0.2751777629182285，误差为：0.023323805896222183
标准函数计算的结果为：0.12404212349727609，逼近函数计算的结果为：0.12984996098269086，误差为：0.005807837485414774
```

k = 3




```
PS F:\KuangjuX\作业\数值计算\Numerical-Analysis> python .\example.py
标准函数计算的结果为: 0.38370211449599323, 逼近函数计算的结果为: 0.38533982247479115, 误差为: 0.0016377079787979198
标准函数计算的结果为: 0.10626725911793064, 逼近函数计算的结果为: 0.09995652087226503, 误差为: 0.006310738245665609
标准函数计算的结果为: 0.2395426083700024, 逼近函数计算的结果为: 0.24712324759664217, 误差为: 0.007580639226639763
标准函数计算的结果为: 0.13441698756176668, 逼近函数计算的结果为: 0.1296750652360391, 误差为: 0.004741922325727565
标准函数计算的结果为: 0.07497590686653667, 逼近函数计算的结果为: 0.073980441017239, 误差为: 0.0009954658492976748
标准函数计算的结果为: 0.03889015673865025, 逼近函数计算的结果为: 0.02855645714757482, 误差为: 0.010333699591075426
标准函数计算的结果为: 0.047371743193664946, 逼近函数计算的结果为: 0.051974436557212345, 误差为: 0.0046026933635473985
标准函数计算的结果为: 0.04289117217919937, 逼近函数计算的结果为: 0.042526970999094194, 误差为: 0.0003642011801051753
标准函数计算的结果为: 0.10295152421811588, 逼近函数计算的结果为: 0.096754548012156, 误差为: 0.006196976205959881
标准函数计算的结果为: 0.4598535186827161, 逼近函数计算的结果为: 0.4487835683042226, 误差为: 0.01106995037849351
```

在最小二乘法拟合中，我们假设不知道原函数的表达式，分别假设 k = 1, 2, 3 随机选取了 $[1, 5]$ 区间上 100 个点进行拟合，拟合结果和误差如上图所示，可见随着 k 的增长误差是逐渐较少的，但是误差仍然比使用最佳平方逼近要大。

使用 **Python 的实现** 如下所示：

```python
    def fit(self):
        A = []
        B = []
        for i in range(0, self.k + 1):
            row = []
            for j in range(0, self.k + 1):
                res = 0
                for item in self.samples:
                    res += pow(item.x, i + j)
                row.append(res)
            y = 0
            for item in self.samples:
                y += pow(item.x, i) * item.y
            A.append(row)
            B.append(y)
        res = list(reversed(np.linalg.solve(A, B)))
        f = np.poly1d(res)
        self.f = f
```

在我们的实现中，我们根据公式，构建矩阵形式 $Ax = B$ ，我们遍历矩阵每一项并求对应的内积并放入到矩阵的位置中，同时求 B 矩阵内每一项的内积，最后解矩阵即可得到对应的多项式的系数。