# 1 曲线拟合

对于函数类A中给定的函数 $f(x)$，记作 $f(x) \in A$，要求在另一类简单的便于计算的函数类B中求函数 $p(x) \in B$，使 $p(x)$ 与 $f(x)$ 的误差在某种度量意义下最小。

## 1.1 曲线拟合的最小二乘法

### 1.1.1 数学定义

对于在数据 $\{(x_i, y_i), i = 0, 1, \cdots, m\}$ 上的拟合函数 $S^*(x)$，设 $\varphi = \text{span}\{\varphi_0(x), \varphi_1(x), \cdots, \varphi_n(x)\}$ 为 $C[a, b]$ 上线性无关函数族，找一函数 $S^*(x)$，使误差平方和

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

这里

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

当 $\varphi = \text{span}\{\varphi_0(x), \varphi_1(x), \cdots, \varphi_n(x)\}$ 取 $\varphi = \text{span}\{1, x, \cdots, x^n\}$ 时，满足**Haar条件**，即一定能找到唯一一组 $\{a_0, a_1, \cdots, a_n\}$ 使得 $S(x)$ 存在最小值。

要使得获得最小误差平方和，可以等价为取得全部元素加权平方和的最小值，也就是将系数 $\{a_0, a_1, \cdots, a_n\}$ 看作一组未知量。对其求偏导可以得到如下极小值存在的条件和定义：若记

$$(\varphi_j, \varphi_k) = \sum_{i=0}^{m} \omega\left(x_i\right)\varphi_j\left(x_i\right)\varphi_k\left(x_i\right)$$

$$(f, \varphi_k) = \sum_{i=0}^{m} \omega\left(x_i\right) f\left(x_i\right)\varphi_k\left(x_i\right) \equiv d_k, \quad k = 0, 1, \cdots, n$$

满足：

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

即

$$\begin{pmatrix} (\varphi_0, \varphi_0) & (\varphi_0, \varphi_1) & \cdots & (\varphi_0, \varphi_n) \\ (\varphi_1, \varphi_0) & (\varphi_1, \varphi_1) & \cdots & (\varphi_1, \varphi_n) \\ \vdots & \vdots & & \vdots \\ (\varphi_n, \varphi_0) & (\varphi_n, \varphi_1) & \cdots & (\varphi_n, \varphi_n) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} d_0 \\ d_1 \\ \vdots \\ d_n \end{pmatrix}$$

$$Ga = d$$

## 1.1.2 算法实现

最小二乘法的算法实现即通过循环嵌套翻译数学表达式

$$(\varphi_j, \varphi_k) = \sum_{i=0}^{m} \omega(x_i)\varphi_j(x_i)\varphi_k(x_i)$$

$$(f, \varphi_k) = \sum_{i=0}^{m} \omega(x_i) f(x_i)\varphi_k(x_i) \equiv d_k, \quad k = 0, 1, \cdots, n$$

并求解线性方程组。求解线性方程组的方法将在最后一章给出。下面给出上述描述的核心代码

```
    def phiprod(self):
        #确定总长度
        n = self.n
        #初始化G,d向量
        G = np.array([])
        d = np.array([])
        #计算并生成G,d向量
        for i in range(0,n):
            d = np.append(d,np.sum((self.arr1_y)*(self.arr1_x**i)))
            for j in range(0,n):
                #这里的G向量是有n个元素的行向量
                G = np.append(G,np.sum((self.arr1_x**i)*(self.arr1_x**j)))
        #通过.reshape方法将G向量转为n阶方阵
        G = G.reshape(n,n)
        #通过np求逆求解，待更新轮子解法
        #an = np.dot(np.linalg.inv(G), d)
        #通过自制LU求解器
        an = self.MartrixSolver(G,d)
        return an,G,d
```

对于满足最小化的an，代入

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

即可确定最小二乘拟合函数。以函数值方式给出，代码如下所示：

```
    def num(self,x):
        num = 0
        for i in range(0,self.n):
            num = num+(self.an[i])*(x**i)
        return num
```

对应的平方误差可通过定义给出。

```
    def delta(self):
        de = np.zeros(self.lenth)
        for i in range(0,self.lenth):
            de[i] = (self.num(self.arr1_x[i])-self.arr1_y[i])**2
        return np.min(de)
```

为了便于计算，将以上求解过程打包为类，全部代码如下：全部打代码可前往Github仓库https://github.com/DMCXE/Numerical_Computation 中文件`FitSquares.py`下查看。

```
import numpy as np
import matplotlib.pyplot as plt
import time
class FitSquares_polynomial:
    def __init__(self,arr1,n):
        self.arr1 = arr1
        self.arr1_x = arr1[:,0]
        self.arr1_y = arr1[:,1]
        self.lenth = len(arr1)
        self.n = n
        self.an = self.phiprod()[0]

    def phiprod(self):
        #确定总长度
        n = self.n
        #初始化G,d向量
        G = np.array([])
        d = np.array([])
        #计算并生成G,d向量
        for i in range(0,n):
            d = np.append(d,np.sum((self.arr1_y)*(self.arr1_x**i)))
            for j in range(0,n):
                #这里的G向量是有n个元素的行向量
                G = np.append(G,np.sum((self.arr1_x**i)*(self.arr1_x**j)))
        #通过.reshape方法将G向量转为n阶方阵
        G = G.reshape(n,n)
        #通过np求逆求解，待更新轮子解法
        #an = np.dot(np.linalg.inv(G), d)
        #通过自制LU求解器
        an = self.MartrixSolver(G,d)
        return an,G,d

    def num(self,x):
        num = 0
        for i in range(0,self.n):
            num = num+(self.an[i])*(x**i)
        return num

    def visualize(self,start,end,step,text):
```

```
        x = np.linspace(start,end,step)
        y = np.zeros(1)
        for i in x:
            y = np.append(y,self.num(i))
        y = y[1:]
        plt.figure()
        plt.scatter(self.arr1_x, self.arr1_y, c='red')
        if text is True:
            for j in range(0,self.lenth):
                plt.text(self.arr1_x[j],self.arr1_y[j],(self.arr1_x[j],self.arr1_y[j]))
        plt.plot(x,y)
        plt.show()

    def delta(self):
        de = np.zeros(self.lenth)
        for i in range(0,self.lenth):
            de[i] = (self.num(self.arr1_x[i])-self.arr1_y[i])**2
        return np.min(de)

    #LU分解
    def MartrixSolver(self,A, d):
        n = len(A)
        U = np.zeros((n, n))
        L = np.zeros((n, n))

        for i in range(0, n):
            U[0, i] = A[0, i]
            L[i, i] = 1
            if i > 0:
                L[i, 0] = A[i, 0] / U[0, 0]
        # LU分解
        for r in range(1, n):
            for i in range(r, n):
                sum1 = 0
                sum2 = 0
                ii = i + 1
                for k in range(0, r):
                    sum1 = sum1 + L[r, k] * U[k, i]
                    if ii < n and r != n - 1:
                        sum2 = sum2 + L[ii, k] * U[k, r]
                U[r, i] = A[r, i] - sum1
                if ii < n and r != n - 1:
                    L[ii, r] = (A[ii, r] - sum2) / U[r, r]
        # 求解y
        y = np.zeros(n)
        y[0] = d[0]

        for i in range(1, n):
            sumy = 0
```

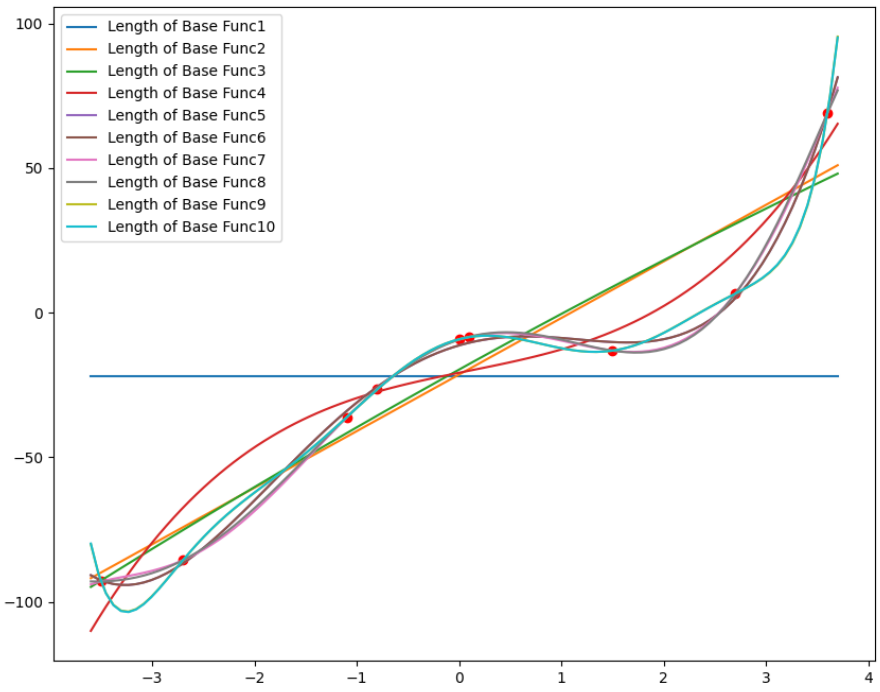
```
            for k in range(0, i):
                sumy = sumy + L[i, k] * y[k]
            y[i] = d[i] - sumy
        # 求解x
        x = np.zeros(n)
        x[n - 1] = y[n - 1] / U[n - 1, n - 1]
        for i in range(n - 2, -1, -1):
            sumx = 0
            for k in range(i + 1, n):
                sumx = sumx + U[i, k] * x[k]
            x[i] = (y[i] - sumx) / U[i, i]

        return x
```

# 2 题目解答

## Q1:对已知的一组数据点用最小二乘法求拟合曲线

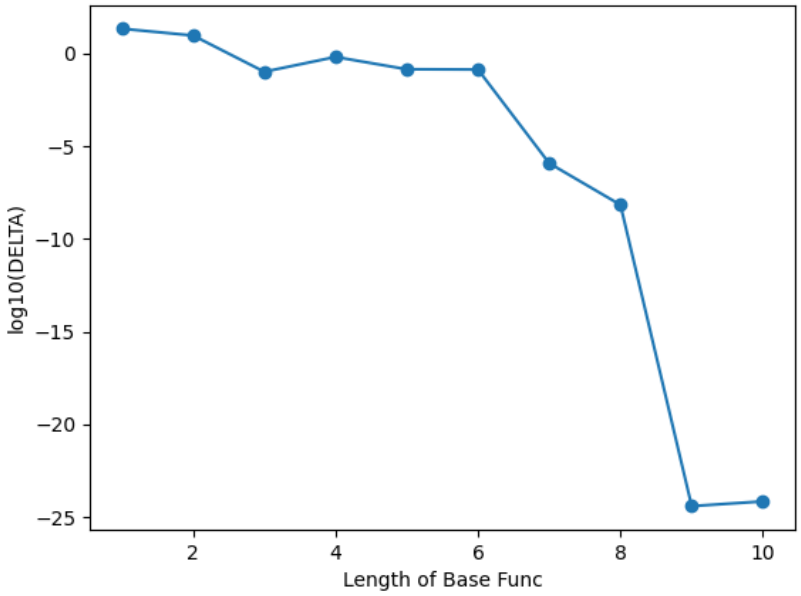
通过上述曲线拟合对于基函数$\varphi = \text{span}\{1, x, \cdots, x^n\}$进行拟合，拟合基函数长度最长不超过数据集的n+1，通过上述方法，可以得到不同基函数长度对应的最小二乘法拟合曲线：不难发现，随着阶次对增加，不稳定性同样增加。




以及不同基函数长度对应的系数：

```
Length of Base Func= 1 an= [-21.81222222]
Length of Base Func= 2 an= [-21.37773908  19.55174151]
Length of Base Func= 3 an= [-19.6623326   19.60209865  -0.35144845]
Length of Base Func= 4 an= [-20.76374472   7.13304311  -0.33601638   1.26861687]
Length of Base Func= 5 an= [-11.30317404  10.21695601  -9.32186911   0.92713941   0.72293613]
Length of Base Func= 6 an= [-1.13001345e+01  1.02349335e+01 -9.32219004e+00  9.21360043e-01
7.22914777e-01  3.55434045e-04]
Length of Base Func= 7 an= [-9.30154339e+00  1.07722607e+01 -1.35170605e+01  7.97083208e-01
  1.62339514e+00  9.01957676e-03 -4.62457522e-02]
Length of Base Func= 8 an= [-9.31163050e+00  1.14654566e+01 -1.32690038e+01  1.74737556e-01
  1.56666349e+00  1.19089744e-01 -4.31072464e-02 -5.17118901e-03]
Length of Base Func= 9 an= [-9.16000000e+00  9.03993242e+00 -1.75740077e+01  1.28085808e+00
  4.66726925e+00 -1.85471114e-02 -5.60082416e-01 -8.43794971e-04 2.36303529e-02]
Length of Base Func= 10 an= [-9.16000000e+00  9.03861680e+00 -1.75628255e+01  1.30126505e+00
  4.66201598e+00 -3.29021353e-02 -5.59414013e-01  1.53576232e-03 2.36086515e-02 -1.08506944e-04]
```

下图给出了不同基函数长度对应的误差函数变化规律，横坐标为基函数长度，纵坐标为Log10化误差，越小越好。




实现的代码为：

```python
import numpy as np
import matplotlib.pyplot as plt
from FitSquares import FitSquares_polynomial as Fp

Arr_x = np.array([-3.5,-2.7,-1.1,-0.8,0,0.1,1.5,2.7,3.6])
Arr_y = np.array([-92.9,-85.56,-36.15,-26.52,-9.16,-8.43,-13.12,6.59,68.94])
Arr = np.c_[Arr_x,Arr_y]

'假设正交基为1,x,xn'
FP = Fp(Arr,6)
print(FP.an)
print(FP.delta())
```

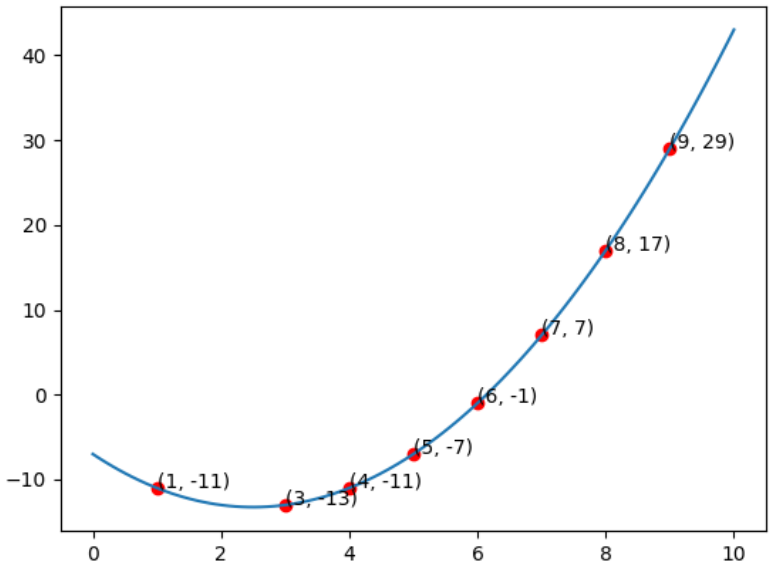
```
FP.visualize(-3.6,3.7,100,False)

delta = np.array([])
plt.figure(figsize=(10, 8))
plt.scatter(Arr_x,Arr_y,c="red",zorder=0)
for i in range(1,11):
    FPi = Fp(Arr,i)
    delta = np.append(delta,FPi.delta())
    print("Length of Base Func=",i,"an=",FPi.an)
    x = np.linspace(-3.6, 3.7, 100)
    y = np.array([])
    for x0 in x:
        y = np.append(y,FPi.num(x0))
    plt.plot(x,y,label="Length of Base Func"+str(i))
    plt.legend()
plt.show()

plt.figure()
plt.plot(range(1,11),np.log10(delta),marker = "o")
plt.xlabel("Length of Base Func")
plt.ylabel("log10(DELTA)")
plt.show()
```

# Q2: 对指定数据进行{1,x,x2}逼近

根据要求，通过上述方法进行拟合，得到的结果为：




基函数对应的系数为：[-7. -5.  1.]

误差Delta为：5.048709793414476e-29

在x=6.5处的函数值为：2.7500000000000213；

实现方法为：

```
import numpy as np
import matplotlib.pyplot as plt
from FitSquares import FitSquares_polynomial as Fp
X = np.array([1,3,4,5,6,7,8,9])
Y = np.array([-11,-13,-11,-7,-1,7,17,29])
Arr = np.c_[X,Y]

'使用拟合基函数1,x,x2进行拟合'
FP = Fp(Arr,3)
print(FP.delta(),FP.an)
print("x=6.5处的值为=",FP.num(6.5))
FP.visualize(0,10,100,True)
```