面向对象程序设计作业（4）

姓名：	学号：	成绩：

编写一个用最小二乘法拟合多项式曲线的程序。要求：

1） 可以拟合任意 n 次多项式$\mathrm{P_n}(x)=\sum_{i=0}^{n}a_i x^i$；

2） 使用梯度下降法求解多项式参数；并输出该参数得到的多项式在样本上的平均误差$l(\theta)$（见下文）。

3） 设计至少 3 个测试用例测试你的代码；

4） 合理设计多文件结构，把代码按照不同功能划分到不同文件中。

5） 练习实践课堂上介绍的程序测试与调试技术。

本作业需要了解的算法知识如下：

假设有 m 个数据样本$(x_i,y_i),i=1\ldots m$，该样本满足$\forall i:y_i=P_n(x_i)+\eta_i$,其中$\eta_i$是一个很小的噪声，服从 0 均值高斯分布（使用 std::normal_distrubtion 生成满足高斯分布的随机数来模拟这个噪声，参考：http://www.cplusplus.com/reference/random/normal_distribution/）。

最小二乘拟合即找出合适的多项式参数$\theta=(a_0,a_1,\ldots,a_n)$使得平方误差的均值$l(\theta)=\frac{1}{n}\sum_{i=1}^{n}(P_n(x_i)-y_i)^2$达到极小值。使用梯度下降法寻找最佳θ的过程是：从一个初始值θ_0(通常可以取 0)开始，按照下式迭代得到新的参数：$\theta_{k+1}=\theta_k-\alpha\frac{dl(\theta_k)}{d\theta}$。在本作业中，对每一个分别更新：$a_i\leftarrow a_i-\alpha\frac{\partial l(\theta)}{\partial a_i}$。

你需要编写函数产生随机样本、计算$l(\theta)$、$\frac{\partial l(\theta)}{\partial a_i}$以及上述更新过程。

要求提交以下内容：

1） 你设计的多文件结构中，每个文件的文件名与该文件负责的主要逻辑功能。

2） 把你所设计的多个文件依次插入在本文档后面的表格中，每一个源文件都要指明文件名。

3） 把每一个测试用例的运行结果截图粘贴在本文档后面。

4） 列出你在开发本程序过程中碰到的一个 bug，把包含 bug 的函数代码插入到本文档，并指明 bug 的具体位置，并给出如何修正该 bug。

在下面说明多文档结构：

Main.cpp 主函数

CreateData.h 头文件，用来为数据生成符合 guss 分布的噪声

GradDescentIterration.h 头文件 ，用来做梯度下降并迭代

在下面插入代码，列出每个源文件的文件名，在文件名之后换行

插入代码：

CreateData.h

```
#ifndef COMPLEX_H_INCLUDED
#define COMPLEX_H_INCLUDED

#include<vector>
#include<random>
using namespace std;
void CreateNoiseData(vector<pair<double,double>>&data)

{
    default_random_engine generator;
     normal_distribution<double>dist;
     int Data_Size=data.size();
     for(int i=0;i<Data_Size;i++)
     {
         data[i].second=data[i].second+dist(generator);

     }
     return;
}

#endif // COMPLEX_H_INCLUDED
```

GradDescentIterration.h

```
#include<iostream>
#include<math.h>
#include<vector>
#include<random>
```

```
5. using namespace std;
6. #define pii pair<int,int>
7. #define pll pair<ll,ll>
8. #define pdd pair<double,double>
9.
10. const double LearnRate=1e-5;//学习率
11. const double delta=1e-8;//计算梯度的 delta
12. const double Accuracy=1e-6;
13. //int cnt=200000;//迭代次数
14.
15. double CalcPolyn(vector<double>Coef,double x)
16. {
17.     double sum=Coef[0];
18.     for(int i=1;i<(int)Coef.size();i++)
19.     {
20.         sum+=Coef[i]*pow(x,i);
21.     }
22.     return sum;
23. }
24.
25. double Calc_L_Theta(vector<double>Coef,vector<pdd>data
    )
26. {
27.     double sum=0;
28.     for(int i=0;i<(int)data.size();i++)
29.     {
30.         sum+=pow((CalcPolyn(Coef,data[i].first)-
    data[i].second),2);
31.     }
32.     double ans=sum/(int)data.size();
33.     return ans;
34. }
35.
36. void GradDescent(vector<double>&Coef,vector<pdd>data)

37. {
38.     vector<double>TempCoef1=Coef;
39.     for(int i=0;i<(int)Coef.size();i++)
40.     {
41.         vector<double>TempCoef2=TempCoef1;
42.         TempCoef2[i]+=delta;
43.         double grad=(Calc_L_Theta(TempCoef2,data)-
    Calc_L_Theta(TempCoef1,data))/delta;
44.         // cout<<"grad"<<grad<<endl;
```

```
45.         Coef[i]-=grad*LearnRate;
46.     }
47. }
48.
49. void iteration(vector<double>&Coef,vector<pdd>data)
50. {
51.     int cnt=1;
52.     vector<double>Temp=Coef;
53.     while(cnt++)
54.     {
55.         bool flag=1;
56.         GradDescent(Coef,data);
57.
58.         for(int i=0;i<(int)Coef.size();i++)
59.         {
60.             if(fabs(Coef[i]-Temp[i])>Accuracy)
61.             {
62.                 Temp=Coef;
63.                 flag=0;
64.                 break;
65.             }
66.
67.         }
68.         if(flag==1)
69.         {
70.             cout<<"迭代次数为: "<<cnt<<endl;
71.             break;
72.         }
73.     }
74. }
```

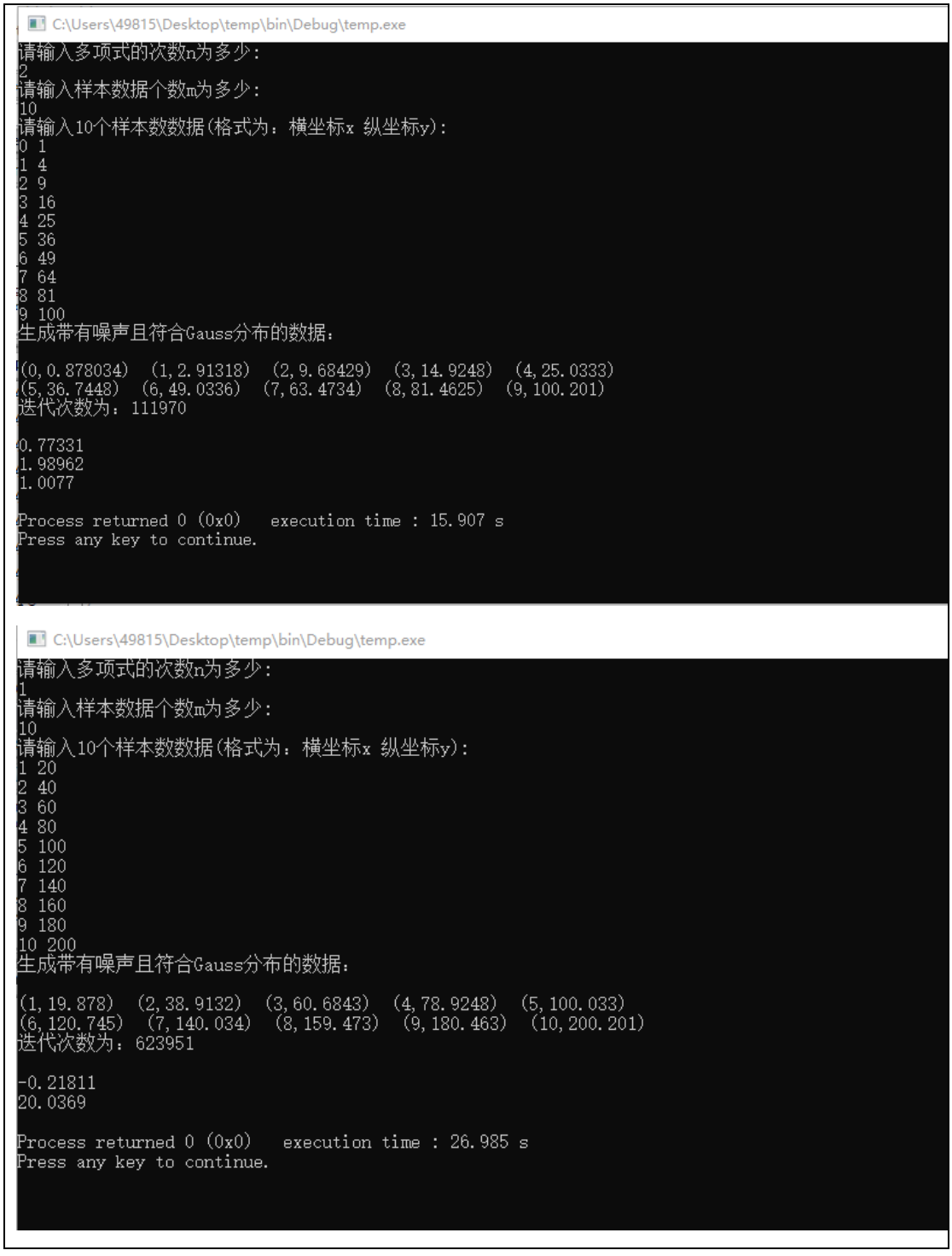
Main.cpp

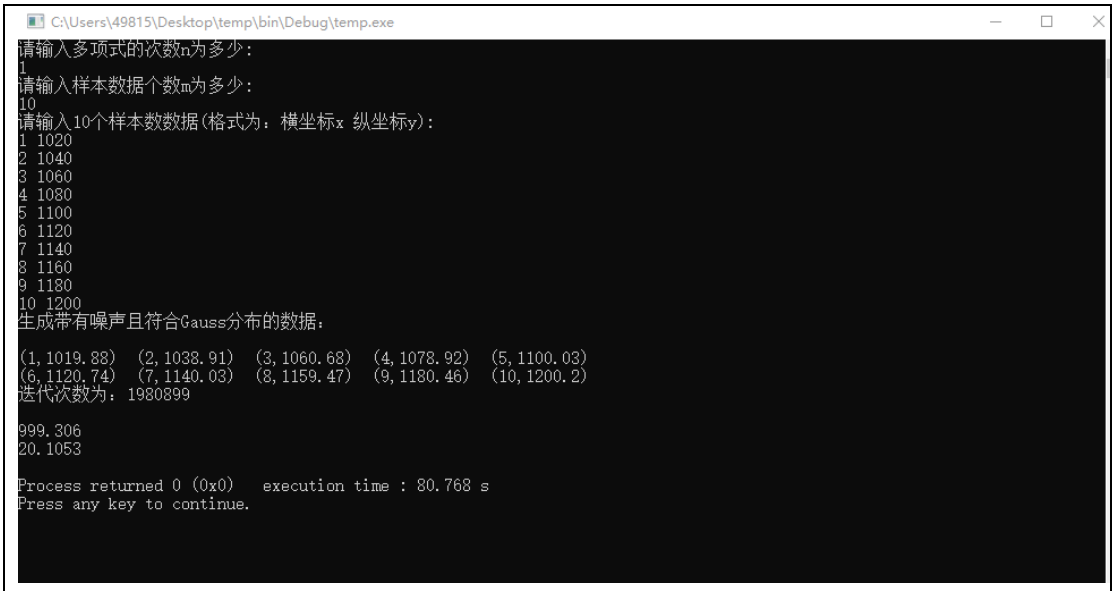
```
1. //Chillstep =v=
2. #include<iostream>
3. #include<math.h>
4. #include<vector>
5. #include<random>
6. #include "CreateData.h"
7. #include "GradDescentIterration.h"
8. using namespace std;
9. #define pii pair<int,int>
10. #define pll pair<ll,ll>
```

```
#define pdd pair<double,double>


int main()
{
    int n,m;
    vector<double>Coef;
    vector<pdd>data;
    cout<<"请输入多项式的次数 n 为多少:"<<endl;
    cin>>n;
    cout<<"请输入样本数据个数 m 为多少:"<<endl;
    cin>>m;
    cout<<"请输入"<<m<<"个样本数数据(格式为：横坐标 x 纵坐标 y):"<<endl;
    for(int i=0;i<m;i++)
    {
        double tempx,tempy;
        cin>>tempx>>tempy;
        data.push_back(make_pair(tempx,tempy));
    }
    cout<<"生成带有噪声且符合 Gauss 分布的数据： "<<endl;
    CreateNoiseData(data);
    for(int i=0;i<(int)data.size();i++)
    {
        if(i%5==0) cout<<endl;
        cout<<"("<<data[i].first<<","<<data[i].second<<")"<<"  ";
        if(i==(int)data.size()-1) cout<<endl;
    }
    for(int i=0;i<=n;i++) Coef.push_back(0);
    iteration(Coef,data);
    cout<<endl;
    for(int i=0;i<(int)Coef.size();i++)
    {
        cout<<Coef[i]<<endl;
    }

    return 0;
}
/*
2
10
0 1
1 4
```

```
2 9
3 16
4 25
5 36
6 49
7 64
8 81
9 100


1
10
1 20
2 40
3 60
4 80
5 100
6 120
7 140
8 160
9 180
10 200

1
10
1 1020
2 1040
3 1060
4 1080
5 1100
6 1120
7 1140
8 1160
9 1180
10 1200
*/
```

在下面粘贴程序运行截图：


C:\Users\49815\Desktop\temp\bin\Debug\temp.exe
请输入多项式的次数n为多少：
2
请输入样本数据个数m为多少：
10
请输入10个样本数数据(格式为：横坐标x 纵坐标y)：
0 1
1 4
2 9
3 16
4 25
5 36
6 49
7 64
8 81
9 100
生成带有噪声且符合Gauss分布的数据：
(0,0.878034) (1,2.91318) (2,9.68429) (3,14.9248) (4,25.0333)
(5,36.7448) (6,49.0336) (7,63.4734) (8,81.4625) (9,100.201)
迭代次数为：111970
0.77331
1.98962
1.0077
Process returned 0 (0x0) execution time : 15.907 s
Press any key to continue.
C:\Users\49815\Desktop\temp\bin\Debug\temp.exe
请输入多项式的次数n为多少：
1
请输入样本数据个数m为多少：
10
请输入10个样本数数据(格式为：横坐标x 纵坐标y)：
1 20
2 40
3 60
4 80
5 100
6 120
7 140
8 160
9 180
10 200
生成带有噪声且符合Gauss分布的数据：
(1,19.878) (2,38.9132) (3,60.6843) (4,78.9248) (5,100.033)
(6,120.745) (7,140.034) (8,159.473) (9,180.463) (10,200.201)
迭代次数为：623951
-0.21811
20.0369
Process returned 0 (0x0) execution time : 26.985 s
Press any key to continue.



在下面列出你遇到的 bug 以及对 bug 的修正：

1.遇到了不是 Bug 但让我以为是 bug 的问题

开始用函数 x^2+2x+1 生成了 10 个数据，而对于 x 选择的比较小，保持在 1-10 左右。最后得出的结果显示为 0.574094 2.10207 0.996549，我一直以为算法有问题，常数项和差的比较多，就一直找 bug，其实是因为噪声原因，噪声的干扰就在 1 左右，所以对常数项 1 的干扰比较大。故不接近 1 是正常的。

2.关于学习率和迭代次数的选取

开始时学习率选的 1e-3 有些大了，会导致在第一个测试样例无法收敛，所以适当的调整到了 1e-4 符合的还是比较好的。同时要注意学习率过低会导致收敛过于慢，迭代次数要增加，会过于耗费时间。

3.关于答案精度的问题，如果每个系数与上一次迭代的差值在 1e-3 到 1e-5，我发现对于某些数据拟合的不是很好，因此调小到了

1e-6，虽然迭代时间会比较长，但是对于各种各样的函数来说，这种精度的鲁棒性会比较好。