**C++ 语 言 程 序 设 计**

实

验

报

告

大作业

姓名： 陈家苇

学号： 180510113

班级： 自动化2班

**一 实验项目**

* 1. 基于抽象优化类optimizer，编写自己的派生优化类，实现多元方程未知数计算
  2. 大作业说明：
* 功能实现：20分，可运行，且运行结果准确
* C++的使用程度，代码书写：20分
* 报告：20分
* 提交内容：
* 电子版大作业报告+程序项目文件（codeblocks）
* 电子版大作业报告命名为学号+姓名
* 程序文件打包为zip或rar，压缩包命名为学号+姓名
* 纸质版大作业报告+实验4报告
* 各班统一收齐后拷贝给我电子版，提交纸质版（11月13日）

**二 实验原理**

1. 介绍自己使用的最优化算法的原理与基本步骤，给出算法的流程图

选择的是PSO算法，即粒子群算法。它是1995年由Eberhart和Kennedy提出的一种全局搜索算法，也是一种模拟自然界生物活动以及群体智能的随机搜索算法。

基本粒子群算法步骤：

（1）初始化，初始化所有粒子的速度和位置，并将个体的历史最优 pBest 设为当前位置，将群体中最优的个体作为当前的 gBest；

（2）根据目标（测试）函数计算各粒子适应度值，并初始化个体、全局最优值

（3）判断是否满足终止条件，是则搜索停止，输出搜索结果；否则继续下步

（4）根据速度、位置更新公式更新各粒子的速度和位置

（5）根据目标函数计算**各粒子适应度值**

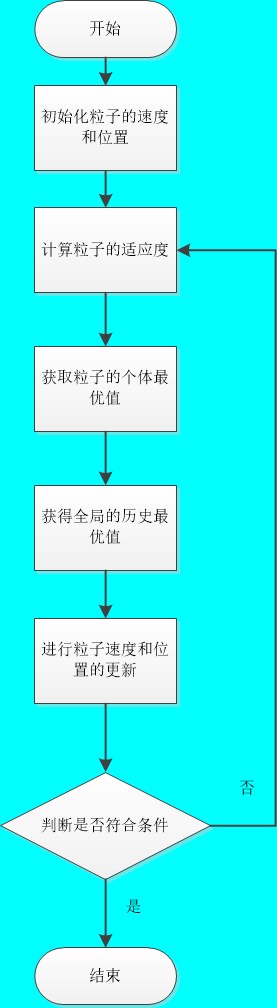
（6）更新各粒子**历史最优值**以及**全局最优值**

（7）判断是否达到结束条件，如果没有，继续步骤上述计算过程，否则输出 gBest并结束

附（速度、位置更新公式：

速度：

位置：）



1. 简单评估自己算法的运行效率

（1）空间复杂度，对于本程序，只需算其数据空间即变量所占大小。

一共有11个变量，sizeof(int)=4, sizeof(double)=8;整体所占字节数43881\*8+2\*4=351056字节；

（2）运行迭代次数为maxgen，2000次；

（3）运算结果精度，即error，本测试函数的结果约为4.83\*10-28左右；

值得一提的是，给出不同的y，error的大小不同，但基本在1e-25到1e-29之间，误差较小；

（4）适用范围：本优化算法需要测试函数的变量维度不超过20，且所有x的取值范围需要在-100到100之间。

1. 程序实现与结果

代码部分：

（工程文件包括一个main.cpp,一个PSO.h，一个OPTIMIZER.h）

/////文件main.cpp://///

#include <iostream>

#include <math.h>

#include "OPTIMIZER.h"

#include "PSO.h"

using namespace std;

double test\_f(double \*x, double \*y)//供测试的函数，物理磁定位算法问题

{

int xd=6;

int yd=8\*3;

double sensor\_position[8][3] =

{

{ -0.10, 0, 0 },

{ -0.05, -0.05, 0 },

{ 0, -0.1, 0 },

{ 0.05, -0.05, 0 },

{ 0.10, 0, 0 },

{ 0.05, 0.05, 0 },

{ 0, 0.10, 0 },

{ -0.05, 0.05, 0 },

};

double K=0.09788\*3.14159\*0.005\*0.005\*0.01\*1090\*10000;

double res=0;

double a,b,c,m,n,p,R,xs,ys,zs;

a=x[0];b=x[1];c=x[2];

m=x[3];n=x[4];p=x[5];

for (int i=0;i<8;i++)

{

xs=sensor\_position[i][0];

ys=sensor\_position[i][1];

zs=sensor\_position[i][2];

double bx,by,bz;

bx=y[i\*3+0];

by=y[i\*3+1];

bz=y[i\*3+2];

R=sqrt((xs-a)\*(xs-a)+(ys-b)\*(ys-b)+(zs-c)\*(zs-c));

bx-=K\*((3\*(m\*(xs-a)+n\*(ys-b)+p\*(zs-c))\*(xs-a))/(R\*R\*R\*R\*R)-m/(R\*R\*R));

by-=K\*((3\*(m\*(xs-a)+n\*(ys-b)+p\*(zs-c))\*(ys-b))/(R\*R\*R\*R\*R)-n/(R\*R\*R));

bz-=K\*((3\*(m\*(xs-a)+n\*(ys-b)+p\*(zs-c))\*(zs-c))/(R\*R\*R\*R\*R)-p/(R\*R\*R));

res+=bx\*bx+by\*by+bz\*bz;

}

return res;

}

void generate\_data(double \*x, double \*y)

{

int xd=6;

int yd=8\*3;

double sensor\_position[8][3] =

{

{ -0.10, 0, 0 },

{ -0.05, -0.05, 0 },

{ 0, -0.1, 0 },

{ 0.05, -0.05, 0 },

{ 0.10, 0, 0 },

{ 0.05, 0.05, 0 },

{ 0, 0.10, 0 },

{ -0.05, 0.05, 0 },

};

double K=0.09788\*3.14159\*0.005\*0.005\*0.01\*1090\*10000;

double a,b,c,m,n,p,xs,ys,zs,R;

a=x[0];b=x[1];c=x[2];

m=x[3];n=x[4];p=x[5];

for (int i=0;i<8;i++)

{

xs=sensor\_position[i][0];

ys=sensor\_position[i][1];

zs=sensor\_position[i][2];

R=sqrt((xs-a)\*(xs-a)+(ys-b)\*(ys-b)+(zs-c)\*(zs-c));

y[i\*3+0]=K\*((3\*(m\*(xs-a)+n\*(ys-b)+p\*(zs-c))\*(xs-a))/(R\*R\*R\*R\*R)-m/(R\*R\*R));

y[i\*3+1]=K\*((3\*(m\*(xs-a)+n\*(ys-b)+p\*(zs-c))\*(ys-b))/(R\*R\*R\*R\*R)-n/(R\*R\*R));

y[i\*3+2]=K\*((3\*(m\*(xs-a)+n\*(ys-b)+p\*(zs-c))\*(zs-c))/(R\*R\*R\*R\*R)-p/(R\*R\*R));

}

}

int main()

{

double x[6]={-0.02,-0.08,0.15,0,1,0};

//预计范围-0.1<x[0]<0.1;-0.1<x[1]<0.1;0.1<x[2]<0.2;-1<=x[3]<=1;-1<=x[4]<=1;-1<=x[5]<=1且满足x[3]\*x[3]+x[4]\*x[4]+x[5]\*x[5]=1

double y[24]={0};

generate\_data(x,y);

for (int i=0;i<8;i++)

{

for(int j=0;j<3;j++)

{

cout<<y[i\*3+j]<<" ";

}

cout<<endl;

}

cout<<endl;

opt1 opti;

OPTIMIZER \*opt=&opti;

cout<<"The error is:"<<opt->setOptimizer(test\_f,x,y,6,24)<<endl;

cout<<endl<<"The result is:"<<endl<<x[0]<<endl<<x[1]<<endl<<x[2]<<endl<<x[3]<<endl<<x[4]<<endl<<x[5]<<endl<<endl;

//cout<<sizeof(int)<<endl<<sizeof(double);//为了确定所需数据空间的大小，此处可以选择输出字节数，以确定占据大小，输出结果应该为4和8

return 0;

}

/////下面是PSO.h/////

#ifndef PSO\_H

#define PSO\_H

#include <cmath>

#include<time.h>

#include<stdlib.h>

#include "OPTIMIZER.h"

#define sizepop 30

#define maxgen 2000//迭代次数

#define dim 20

#define c1 2.0//学习因子

#define c2 2.0

#define w 0.6//权重因子

#define popmax 100 // 个体最大取值

#define popmin -100 // 个体最小取值

#define Vmax 10.0 // 速度最大值

#define Vmin -10.0 //速度最小值

class opt1:public OPTIMIZER

{

public:

double pop[sizepop][dim];

double v[sizepop][dim];

double fitness[sizepop];

double gbest[dim];

double pbest[sizepop][dim];

double result[maxgen];

double fitnesspbest[sizepop];

double fitnessgbest;

double genbest[maxgen][dim];

void pop\_init(void);//种群初始化

double\* min(double\* ,int);//max函数定义

void PSO\_func();//迭代

double setOptimizer(double (\*foo)(double \*,double \*),double \*x1,double \*y1,int xd1,int yd1)

{

fun=foo;

x=x1;

y=y1;

xd=xd1;

yd=yd1;

PSO\_func();

int bestgennumber=\*(min(result,maxgen));

for(int i=0;i<xd;i++)

{

x[i] = genbest[bestgennumber][i];

}

return fun(x,y);

}

};

void opt1::pop\_init(void)

{

for(int i=0;i<sizepop;i++)

{

for(int j=0;j<xd;j++)

{

pop[i][j]=(((double)rand())/RAND\_MAX-0.5)\*2;

v[i][j]=(((double)rand())/RAND\_MAX-0.5)/10;

}

fitness[i]=fun(pop[i],y);

}

}

double\* opt1::min(double\* fit,int size)

{

int index=0;//初始化序号

double min=\*fit;//初始化最大值为数组第一个元素

static double bestfitindex[2];

for(int i=0;i<size;i++)

{

if(\*(fit+i)<min)

{

min=\*(fit+i);

index=i;

}

}

bestfitindex[0]=index;

bestfitindex[1]=min;

return bestfitindex;

}

void opt1::PSO\_func(void)

{

pop\_init();

double \* best\_fit\_index; // 用于存放群体极值和其位置

best\_fit\_index = min(fitness,sizepop); //求群体极值

int index = (int)(\*best\_fit\_index);

for(int i=0;i<xd;i++) // 群体极值位置

{

gbest[i] = pop[index][i];

}

for(int i=0;i<sizepop;i++) // 个体极值位置

{

for(int j=0;j<xd;j++)

{

pbest[i][j] = pop[i][j];

}

}

for(int i=0;i<sizepop;i++) // 个体极值适应度值

{

fitnesspbest[i] = fitness[i];

}

double bestfitness = \*(best\_fit\_index+1); //群体极值适应度值

fitnessgbest = bestfitness;

//迭代寻优

for(int i=0;i<maxgen;i++)

{

for(int j=0;j<sizepop;j++)

{

for(int k=0;k<xd;k++)

{

double rand1 = (double)rand()/RAND\_MAX; //0到1之间的随机数

double rand2 = (double)rand()/RAND\_MAX;

v[j][k] = w\*v[j][k] + c1\*rand1\*(pbest[j][k]-pop[j][k]) + c2\*rand2\*(gbest[k]-pop[j][k]);

if(v[j][k] > Vmax)

{

v[j][k] = Vmax;

}

if(v[j][k] < Vmin)

{

v[j][k] = Vmin;

}

pop[j][k] = pop[j][k] + v[j][k];

if(pop[j][k] > popmax)

{

pop[j][k] = popmax;

}

if(pop[j][k] < popmin)

{

pop[j][k] = popmin;

}

}

fitness[j] = fun(pop[j],y); //新粒子适应度值

}

for(int j=0;j<sizepop;j++)

{

if(fitness[j] < fitnesspbest[j])// 个体极值更新

{

for(int k=0;k<xd;k++)

{

pbest[j][k] = pop[j][k];

}

fitnesspbest[j] = fitness[j];

}

if(fitness[j] < fitnessgbest) // 群体极值的更新

{

for(int k=0;k<xd;k++)

{

gbest[k] = pop[j][k];

}

fitnessgbest = fitness[j];

}

}

for(int k=0;k<xd;k++)

{

genbest[i][k] = gbest[k]; // 每代最优值取值粒子位置记录

}

result[i] = fitnessgbest; // 每代的最优值记录到数组

}

}

#endif // PSO\_H

/////下面是OPTIMIZER.h/////

#ifndef OPTIMIZER\_H

#define OPTIMIZER\_H

class OPTIMIZER

{

public:

virtual double setOptimizer(double (\*foo)(double \*,double \*),double \*,double \*,int ,int)=0;

double\* PSO\_func();

protected:

double \*x;

double \*y;

int xd;

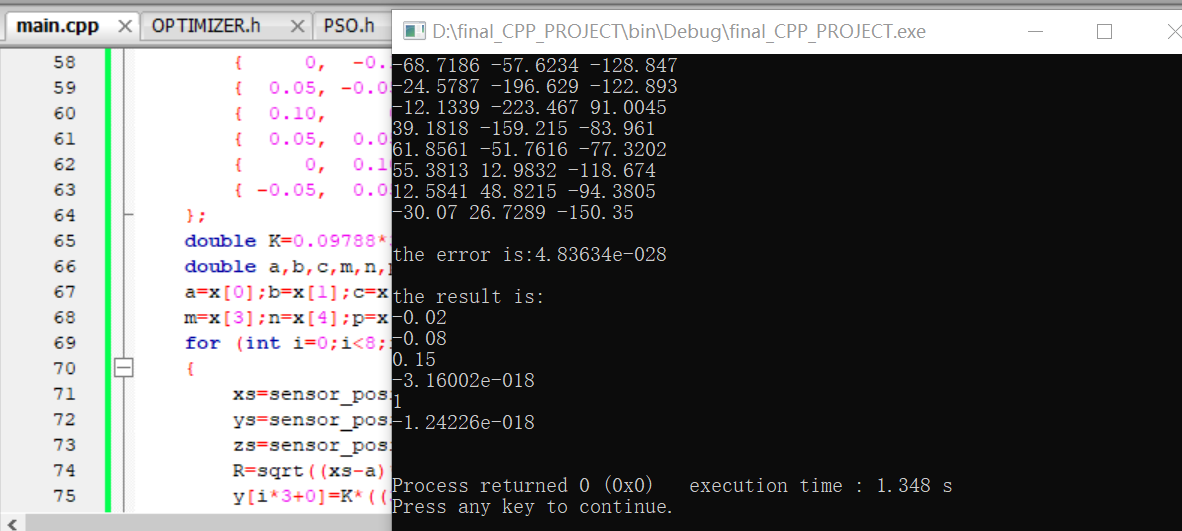
int yd;

double (\*fun)(double \*,double \*);

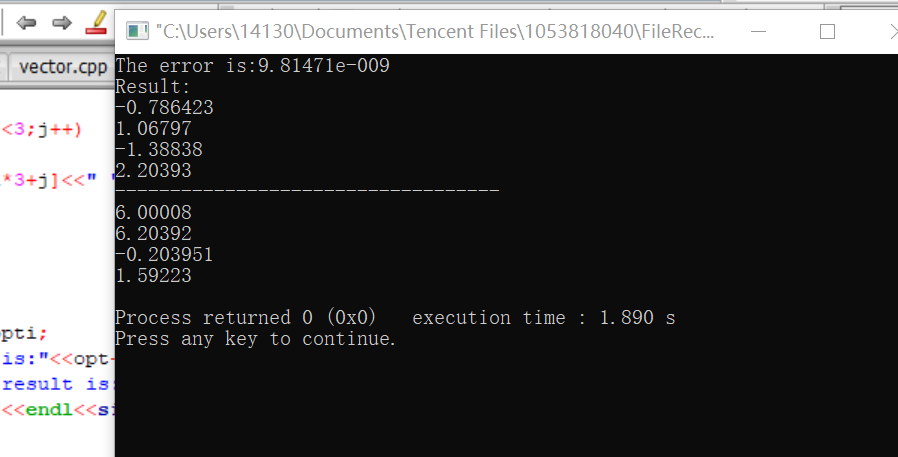
};

#endif // OPTIMIZER\_H

运行结果截图：



另附群里老师曾经所发的Matrix\_4x4的测试结果（书写代码过程中曾使用过，后将测试函数改成了磁场问题）：



**三 实验总结与建议**

粒子群算法PSO简单易懂，在选择这个算法之前，先对老师所提到的其他算法，如牛顿迭代法、LM算法、遗传算法做了一定的查询研究，发现粒子群算法最容易理解和使用（也可能因为个人能力有限）。在使用粒子群算法需要注意的问题就是其权重因子、学习因子，最大最小位置、最大最小速度的设置，对误差都有一定的影响，所以此次实验的难点之一就是如何调参，从而使得error尽量小。

本次实验收获很大，更加深入理解和使用c++类以及继承和派生的知识，而且对于粒子群算法这个有意思的算法有了一定理解，应用方面可能还不够清晰，只是针对测试函数及其近似类型的函数做了思考，没有设想过一些奇怪或者复杂的函数也没有考虑对y的干扰因素。同时在编写程序的过程中，对面向对象编程也有了更深的理解。每次实验都会存在的概念和知识点的使用问题依旧存在，以后对于代码的熟练程度还需要加强。