# 智能算法作业三

### 18级计算机科学与技术2班 叶劲亨

## 实验目的

* + 使用PSO（粒子群优化算法）解决最值求解问题
  + 掌握PSO算法的代码编写
  + 研究惯性因子w和加速因子c对算法收敛结果的影响

### 实验内容

实现粒子群程序，求解连续函数问题，并对算法参数进行调查，写成报告。

求解问题：



## 实验要求

1、参数设定：维数n=30

N=20, w=0.9~0.4, c1=c2=2.0, Gmax=2000

2、对参数w和c1/c2进行调查，分别设w=0.1,0.2,0.3….,0.9；c1=c2=0.5,1.0，1.5，…3.5，对每一组参数组合在函数上运行30次，基于30次的平均结果进行报告的撰写。

## 4、解题思路：

本次实验采用c++实现，下面介绍c++的pso代码实现。

### 4.1 参数设定：

种群规模N = 20

粒子的长度D 由待求解函数的维度决定，本次问题D = 30

粒子的范围R 由题目给定的xi取值范围决定

最大速度Vmax 设置为相应维搜索空间的20%

惯性权重w 

加速系数c1和c2

终止条件 达到最大迭代次数时停止

全局和局部PSO 本次实验采用全局PSO

同步和异步更新 本次实验采用异步更新

粒子位置越界处理 如果位置越界，随机给该位置分配一个在对应维度取值范围的值

### 4.2 待求解函数定义（以函数二为例）

// 定义待求解函数

class F {

public:

double operator()(const vector<double>& x) {

double res = 0.0;

for (double d : x) {

res += d \* d - 10 \* cos(2 \* PI \* d) + 10;

}

return res;

}

};

### 4.3 PSO类实现（详细代码见附件）

class PSO {

public:

//待求解函数

F func;

//种群规模

int pop\_size;

//惯性权重

double w;

//迭代次数

int generations;

//加速因子

double c1;

double c2;

//粒子长度

int particle\_len;

//取值范围

vector<vector<double>>\* particles\_bound; // 30 \* 2

//粒子

vector<vector<double>> particles; // 20 \* 30

//每个粒子的速度

vector<vector<double>> velocities;

//每个粒子的最优位置

vector<vector<double>> p\_best;

vector<double> p\_best\_val;

//全局最优

vector<double> g\_best;

double g\_best\_val = INT\_MAX;

public:

//构造函数，传入算法必要参数

PSO(F func, int pop\_size, double w, int generations, double c1, double c2, int particle\_len, vector<vector<double>>\* particles\_bound) :

func(func), pop\_size(pop\_size), w(w), generations(generations), c1(c1), c2(c2),

particle\_len(particle\_len), particles\_bound(particles\_bound) {}

//初始化粒子群

void init\_particles() {

this->particles = vector<vector<double>>(this->pop\_size);

this->p\_best = vector<vector<double>>(this->pop\_size);

this->p\_best\_val = vector<double>(this->particle\_len, INT\_MAX);

//初始化速度

this->velocities = vector<vector<double>>(this->pop\_size);

for (int i = 0; i < pop\_size; ++i) {

for (int j = 0; j < particle\_len; ++j) {

double span = (\*particles\_bound)[i][1] - (\*particles\_bound)[i][0];

particles[i].push\_back(rand() / (double)(RAND\_MAX)\* span + (\*particles\_bound)[i][0]);

velocities[i].push\_back(rand() / (double)(RAND\_MAX) \* 20 - 10);

}

}

}

//评估最优个体，记录最优个体所在位置

void evaluate() {

g\_best.clear();

double temp;

for (int i = 0; i < pop\_size; ++i) {

temp = func(particles[i]);

//cout << temp << endl;

if (temp < p\_best\_val[i]) {

p\_best\_val[i] = temp;

p\_best[i] = vector<double>(particles[i]);

}

if (temp < g\_best\_val) {

g\_best\_val = temp;

this->g\_best = vector<double>(particles[i]);

}

}

}

//更新个体的速度和位置，同时记录个体历史最优和全局最优

void update() {

double span = 10.24;

double v\_span = span / 5;

double temp;

for (int i = 0; i < pop\_size; ++i) {

for (int j = 0; j < particle\_len; ++j) {

double& p = particles[i][j];

double& v = velocities[i][j];

v = w \* v + c1 \* ((double)rand() / RAND\_MAX) \* (p\_best[i][j] - p) + c2 \* ((double)rand() / RAND\_MAX) \* (g\_best[j] - p);

if (v > v\_span) {

v = v > 0 ? v\_span : -v\_span;

}

p += velocities[i][j];

if (p < (\*particles\_bound)[j][0] + 1e-5 || p >(\*particles\_bound)[j][1] - 1e-5) {

p = rand() / (double)(RAND\_MAX)\* span + (\*particles\_bound)[i][0];

}

}

//异步更新

temp = func(particles[i]);

if (temp < p\_best\_val[i]) {

p\_best\_val[i] = temp;

p\_best[i] = vector<double>(particles[i]);

}

if (temp < g\_best\_val) {

g\_best\_val = temp;

this->g\_best = vector<double>(particles[i]);

}

}

}

//运行

void run() {

this->init\_particles();

evaluate();

for (int i = 0; i < generations; ++i) {

//cout << "generation: " << i << ": " << g\_best\_val << endl;

update();

}

//cout << "finished: " << g\_best\_val << endl;

}

void test() {

for (int i = 0; i < pop\_size; ++i) {

for (int j = 0; j < particle\_len; ++j) {

cout << particles[i][j] << " ";

}

cout << endl;

}

}

};

## 4.2 实现步骤

### 4.2.1 将参数依次传入

### PSO pso(F(), pop\_size, w1, generations, c, c, particle\_len, bound);

### 4.2.2 定义必要参数

F func();

int pop\_size = 20;

double w = 0.729;

int generations = 2000;

double c1 = 1.49445;

double c2 = 1.49445;

int particle\_len = 30;

vector<vector<double>>\* bound = new vector<vector<double>>(30);

for (int i = 0; i < 30; ++i) {

(\*bound)[i].push\_back(-5.12);

(\*bound)[i].push\_back(5.12);

}

vector<double> w\_list({ 0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9 });

vector<double> c\_list({ 0.5,1.0,1.5,2.0,2.5,3.0,3.5 });

int epochs = 30;

### 4.2.3 运行并记录结果

ofstream file;

file.open("D://data.csv");

writeHead(file, particle\_len);

for (double w1 : w\_list) {

for (double c : c\_list) {

for (int e = 0; e < epochs; ++e) {

PSO pso(F(), pop\_size, w1, generations, c, c, particle\_len, bound);

pso.run();

cout << "w: " << w1 << ", c1=c2: " << c << ", epoch: " << e << ", g\_best: " << pso.g\_best\_val << endl;

file << w1 << "," << c << "," << e << "," << pso.g\_best\_val;

file << "\n";

}

}

### }

### 4.2.4 得到csv格式的输出文件（见附件）

示例：

w,c1=c2,epoch,g\_best

0.1,0.5,0,3708.75

0.1,0.5,1,5608.51

......

### 4.2.5 处理结果，绘制图像

绘制图像使用的是python的seaborn库

4.2.3.3 绘制热力学图

**import** numpy **as** np  
**import** pandas **as** pd  
**import** matplotlib.pyplot **as** plt  
**import** seaborn **as** sns  
  
data = pd.read\_csv(**'data2.csv'**, delimiter=**","**)  
  
w\_list = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]  
c\_list = [0.5,1.0,1.5,2.0,2.5,3.0,3.5]  
  
average = pd.DataFrame(columns=[**'w'**,**'c'**,**'g\_best'**])  
  
i = 0  
**for** w **in** w\_list:  
 **for** c **in** c\_list:  
 temp = data[(data[**'w'**]==w) & (data[**'c1=c2'**]==c)]  
 average.loc[i] = [w, c, np.average(temp[**'g\_best'**])]  
 i += 1  
  
*# 绘图*res = np.array(average[**'g\_best'**]).reshape((len(w\_list), len(c\_list))).T  
plt.figure(figsize=(20,10))  
  
sns.heatmap(res, annot=**True**, fmt=**'.3f'**, xticklabels=w\_list, yticklabels=c\_list)



heatmap1.png



Heatmap2.png

颜色越深，代表用这一组参数得到的30次平均结果中越小，因为这是最小值优化问题，所以效果越好。

对本次实验结果分析得知：

1、惯性权重w取2.0附近一般能加速算法的收敛，w太小结果难以收敛，w太大导致结果发散

2、加速因子c不宜取过大或或小，取c=0.5附近能得到较好结果

## 5、源代码：

见附件