

计算物理 A 第十三题

杨旭鹏 PB17000234

2019 年秋季

目录

1 题目描述	2
2 算法	2
2.1 程序子函数介绍	2
2.1.1 随机粒子初始位置产生器	2
2.1.2 粒子随机行走器	3
2.1.3 盒计数法统计器	3
2.1.4 Sandbox 统计器	3
2.2 16807 产生器	3
3 程序使用方法	4
4 程序结果与讨论	4
4.1 DLA 模拟结果可视化	4
4.2 分形维数计算	6
4.3 有趣的讨论——逃逸区域对于结果的影响	9
5 心得与体会	15
6 附录	16
A DLA 模拟 C 语言源程序	16
B 分形维数统计 C 语言程序源码	24
C 可视化绘图及数据处理 Python 程序源码	27

1 题目描述

进行单中心 DLA 模型的模拟 (可以用圆形边界, 也可以用正方形边界), 并用两种方法计算模拟得到的 DLA 图形的分形维数, 求分形维数时需要作出双对数图。

2 算法

将二维平面划分为一个个正方形格点, 点阵中心放置一个种子粒子作为凝聚中心, 在距离种子某距离的产生边界上随机产生一个粒子, 使其做随机行走。当粒子走到与团簇粒子相接触时, 就被粘住不动, 成为团簇的一部分; 当粒子越过远离中心某距离的逃离边界时, 粒子被判定逃离, 重新由起始位置开始随机行走。当团簇不断扩大时, 产生边界和逃离边界跟随变大。直到团簇的大小为规定值时, 停止。

此程序利用二维数组表示二维平面上的格点, 原点处于二维数组的中心位置。规定二维数组的边界距离中心有 LEN 个单位长度 (即二维数组的形状为 $(2LEN + 1) \times (2LEN + 1)$, 原点的行号和列号均为 LEN)。规定粒子随机行走一步只能向上, 向下, 向左或向右移动一个单位长度 (即一格) 且朝四个方向移动的概率相同。当粒子周围 8 个格点有团簇粒子时, 判定该粒子粘连到团簇粒子, 停止随机行走, 将此位置所对应的二维数组的值改为 1, 表示此处有粒子, 并在产生边界上随机产生一个新的粒子, 重复此过程。

2.1 程序子函数介绍

2.1.1 随机粒子初始位置产生器

传入参数包括一在 $(0, 1)$ 均匀抽取的随机值 (由 16807 产生器产生) 和产生边界距离原点的大小 r 。例如对于圆形边界, 将传入进来的随机值 $\times 2\pi$ 即得到在 $(0, 2\pi)$ 均匀抽取的随机值 θ 作为粒子起始位置在极坐标系下的角度值。则随机粒子初始位置的 (x, y) 坐标分别为 $round(r\cos(\theta))$ 和 $round(r\sin(\theta))$ 。其中 $round()$ 为四舍五入函数, 因为最后初始位置要落实到某个格点上。则随机粒子在二维矩阵上的行号 i 和列号 j 分别为 $x + LEN$ 和 $y + LEN$ 。

2.1.2 粒子随机行走器

传入参数为粒子的初始位置；逃逸边界半径；二维平面对应的二维矩阵；在 $[-2^{31}, 2^{31} - 1]$ 均匀抽取的随机整数，用于作为产生随机数的种子。利用 16807 产生器产生均匀分布在 $[0, 1]$ 的随机数，对于此粒子的第 k 步来说，若第 k 个随机数 $\in (0, \frac{1}{4}]$ ，则向 x 轴正方向走一个单位长度；若 $\in (\frac{1}{4}, \frac{1}{2}]$ ，向 x 轴负向走一个单位；若 $\in (\frac{1}{2}, \frac{3}{4}]$ ，向 y 轴正向走一个单位；若 $\in (\frac{3}{4}, 1]$ ，向 y 轴负向走一个单位。每步结束后，都判断周围 8 个格点是否有团簇粒子，是否走出逃逸边界。

2.1.3 盒计数法统计器

此子程序包含于另一用于数据处理的 C 程序。将生成好的包含团簇粒子位置信息的二维矩阵传入，程序会统计出以原点为中心，长度为 1024 个格点的 $N(\epsilon)$

2.1.4 Sandbox 统计器

此子程序包含于另一用于数据处理的 C 程序。将生成好的包含团簇粒子位置信息的二维矩阵传入，程序会统计出以原点为中心，长度为 $11, 21, 31, \dots, 601$ 个格点的 $N(r)$

2.2 16807 产生器

作用中所产生的随机数使用 16807 产生器产生。16807 产生器属于线性同余法产生器的特例。而线性同余法方法为：

$$\begin{aligned} I_{n+1} &= (aI_n + b) \bmod m \\ x_n &= I_n / m \end{aligned} \tag{1}$$

其中整数 $I_i \in [0, m - 1]$ ， a, b, m 为算法中的可调参数，其选取直接影响产生器的质量。选取参数：

$$\begin{cases} a = 7^5 = 16807 \\ b = 0 \\ m = 2^{31} - 1 = 2147483647 \end{cases} \tag{2}$$

即为所谓的 16807 产生器。由于直接利用 1 编写程序时计算 $(aI_n \bmod m)$ 时很容易造成数据溢出，故采取 Schrage 方法进行具体编程的实现：

$$aI_n \bmod m = \begin{cases} a(I_n \bmod q) - r[I_n/q], & if \geq 0 \\ a(I_n \bmod q) - r[I_n/q] + m, & otherwise \end{cases} \quad (3)$$

其中 $m = aq + r$, 即 $q = m/a = 127773$, $r = m \bmod a = 2836$ 。即可利用此方法产生伪随机数序列。

3 程序使用方法

此程序设计为参数在程序代码中直接赋值形式, 每次需在程序源码中进行参数调整, 进而编译运行, 以简化每次输入的过程 (重复运行时不用在此输入)。需要调整的参数包括宏定义中的 LEN , `main` 函数中的总粒子数 N , 逃逸边界半径值, 粒子产生边界半径值 (此两个半径值均可设定动态, 静态, 不同的函数形式)。调整完参数后, 编译运行, 程序会自动输出包含团簇信息的二维数组至文件。

对于分形维数统计程序, 在源码中修改读取的数据文件名和路径, 编译运行, 即可输出统计结果

4 程序结果与讨论

4.1 DLA 模拟结果可视化

当我们选取粒子数 $N = 5 \times 10^4$, 逃逸边界, 粒子产生边界均为圆形, 半径 (R_{gen}, R_{esp}) 分别为大于团簇粒子最外围粒子半径 ($r_{cluster}$) 5 个单位长度, 2 倍粒子产生半径时, 得到结果:

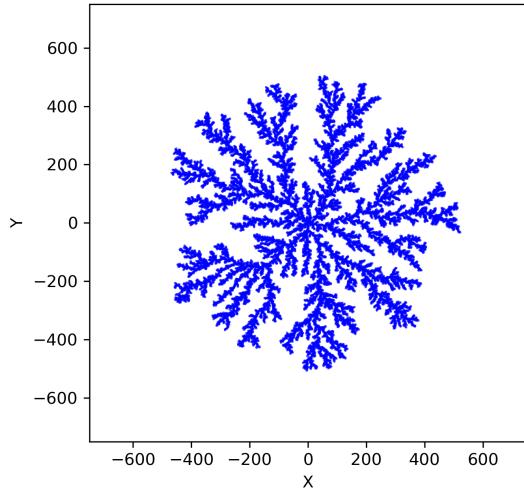


图 1: $R_{gen} = r_{cluster} + 5$, $R_{esp} = 2R_{gen}$ 的可视化模拟结果

修改逃逸边界半径为 $2.5r_{cluster}$ 时有:

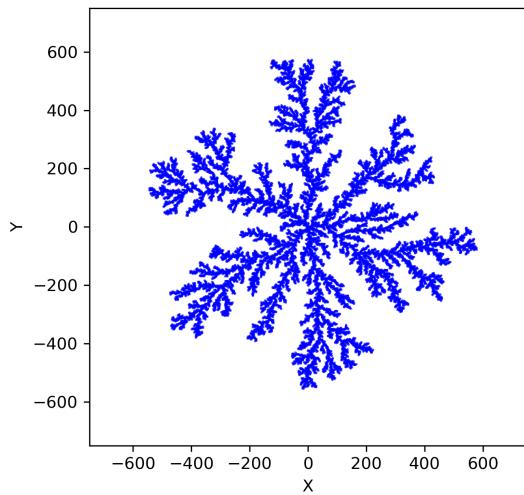


图 2: $R_{gen} = r_{cluster} + 5$, $R_{esp} = 2.5R_{gen}$ 的可视化模拟结果

修改逃逸边界半径为 $3r_{cluster}$ 时有:

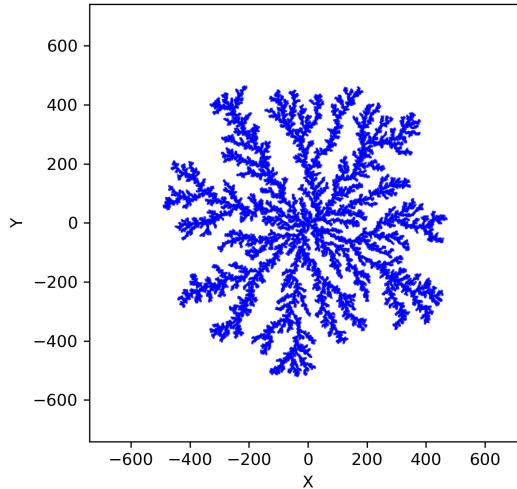


图 3: $R_{gen} = r_{cluster} + 5, R_{esp} = 3R_{gen}$ 的可视化模拟结果

可以看出三种逃逸边界的半径之间基本没有什么不同。

4.2 分形维数计算

对每个半径生成的数据做合计数法和 Sandbox 计数分别得到 $\ln N - \ln(1/\epsilon)$ 和 $\ln N - \ln r$ 的散点图，线性拟合得到的斜率即为数值模拟得到的分形维数。有如下结果：

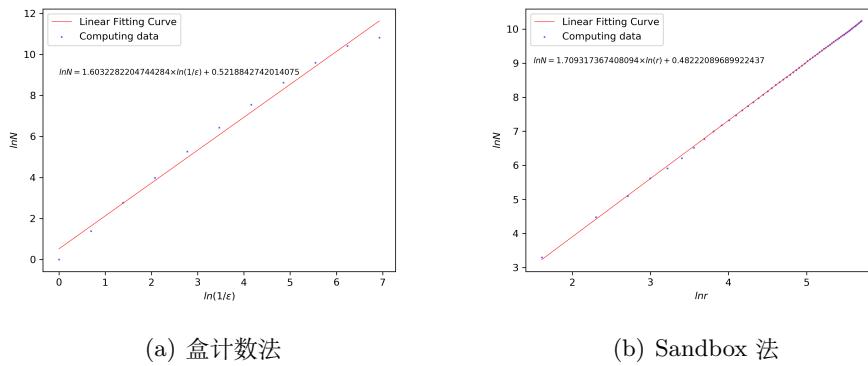


图 4: $R_{gen} = r_{cluster} + 5, R_{esp} = 2R_{gen}$ 的分形维数计算

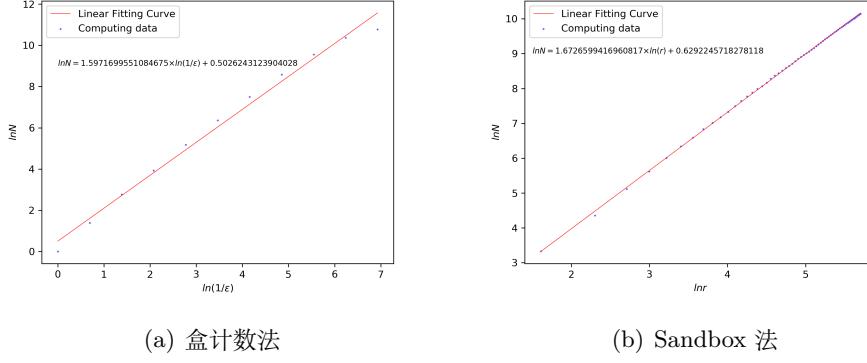


图 5: $R_{gen} = r_{cluster} + 5, R_{esp} = 2.5R_{gen}$ 的分形维数计算

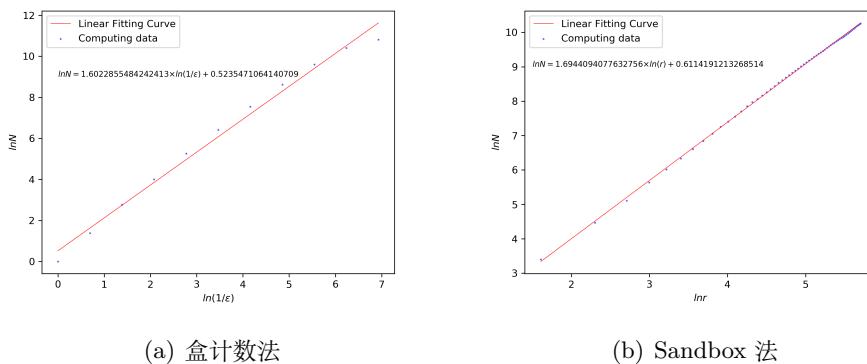


图 6: $R_{gen} = r_{cluster} + 5, R_{esp} = 3R_{gen}$ 的分形维数计算

我们看到两个方法得到的分形维数还是有一些差异的，具体原因不知，可能由于盒计数法点数不够多，导致误差较大。可认为 DLA 的分形维数在 $1.6 - 1.7$ 之间。有趣的是，我们能够看出对于盒计数法来说，最后一个点偏离拟合直线许多，于是我们认为删掉最后一个点的数据得到：

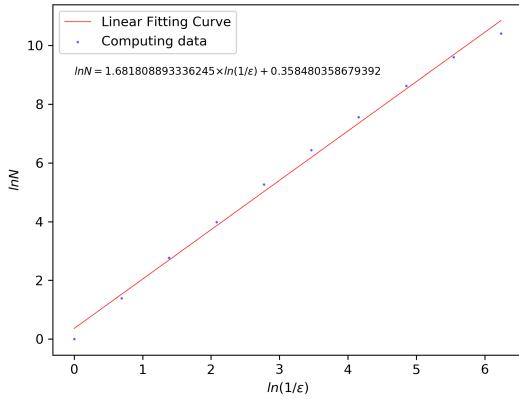


图 7: $R_{gen} = r_{cluster} + 5, R_{esp} = 2R_{gen}$ 删掉最后一个数据点的盒计数法

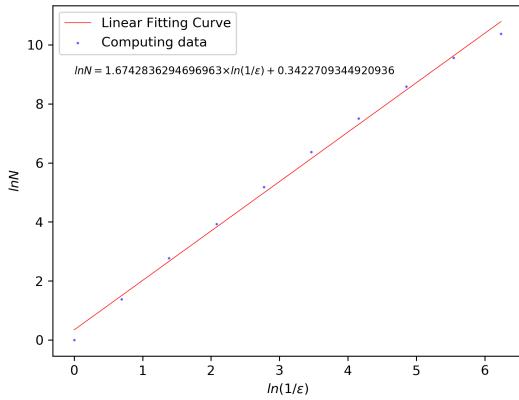


图 8: $R_{gen} = r_{cluster} + 5, R_{esp} = 2.5R_{gen}$ 删掉最后一个数据点的盒计数法

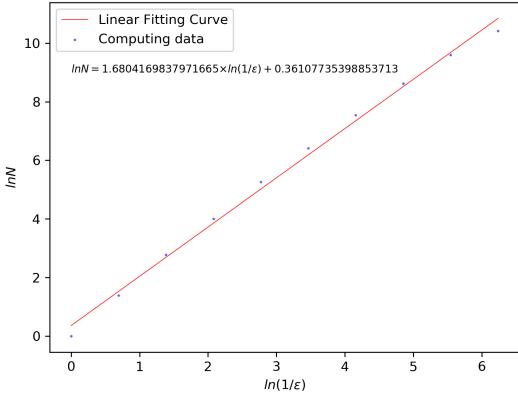


图 9: $R_{gen} = r_{cluster} + 5, R_{esp} = 3R_{gen}$ 删掉最后一个数据点的盒计数法

可以看出删掉最后一个数据点后，盒计数法得到的分形维数与 Sandbox 法得出的结果相差不大。具体原因目前并不清楚。

4.3 有趣的讨论——逃逸区域对于结果的影响

由于在实际情况下，某粒子可以随机行走到很远的地方（例如电解很稀的 $CuSO_4$ 溶液时，容器边界相对于团簇大小来说很远），故逃逸边界只是数值模拟所必须的条件，且设置成为可动的边界是为了提高程序的运行速度。根据计算物理讲义上写到的一般取粒子产生边界大于团簇边界 5 个单位，逃逸边界设置为产生边界的 $2 \sim 3$ 倍。

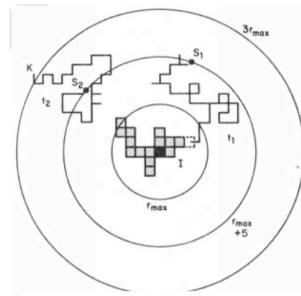


图 10: 计算物理讲义上的 DLA 算法边界半径设置示意图

故假设我们将逃逸半径的值设置的比较小时，看看结果会如何，有：
可以看到逃逸半径的减少，使 DLA 团簇更加密集。

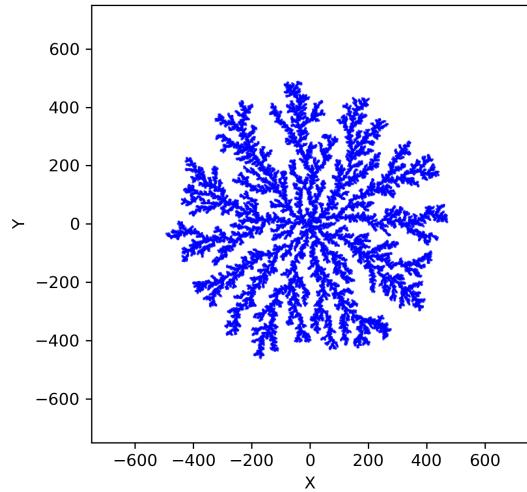


图 11: $R_{gen} = r_{cluster} + 5, R_{esp} = 1.1R_{gen}$ 的可视化模拟结果

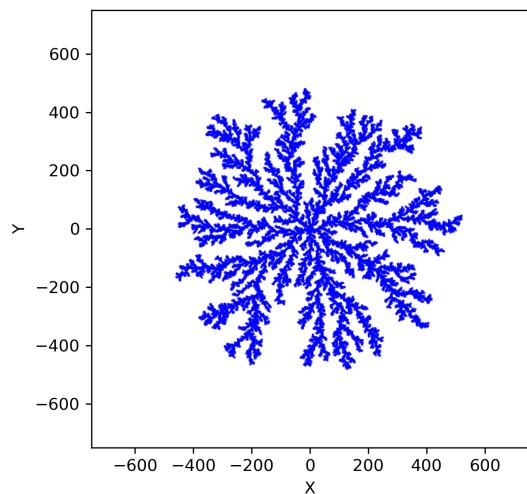


图 12: $R_{gen} = r_{cluster} + 5, R_{esp} = 1.5R_{gen}$ 的可视化模拟结果

若这样变化的趋势不够明显，下面有更加夸张的结果，将逃逸半径的值改为产生半径的值加上某一定值¹，得到：

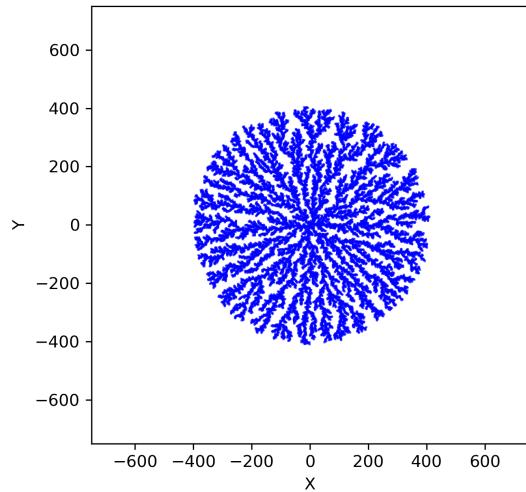


图 13: $R_{gen} = r_{cluster} + 5, R_{esp} = R_{gen} + 2$ 的可视化模拟结果

¹注意此处逃逸半径的设置并不是很科学，因为半径的设置应与团簇的相对大小有关，而加上某一定值会使得边界与团簇的相对大小不断变化。但此处这么设置是为了夸张的展示此现象

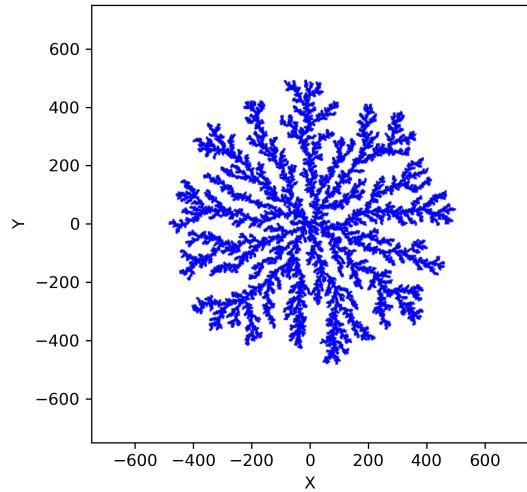


图 14: $R_{gen} = r_{cluster} + 5, R_{esp} = R_{gen} + 50$ 的可视化模拟结果

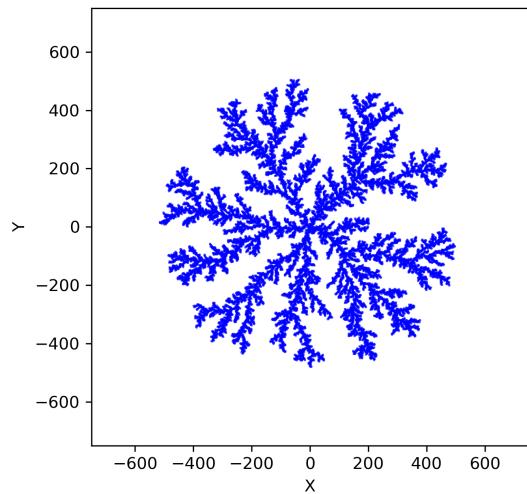


图 15: $R_{gen} = r_{cluster} + 5, R_{esp} = R_{gen} + 200$ 的可视化模拟结果

可以明显的看出此规律。为了定量描述此规律，我们对上面三种情况进行了分形维数的统计，得到：

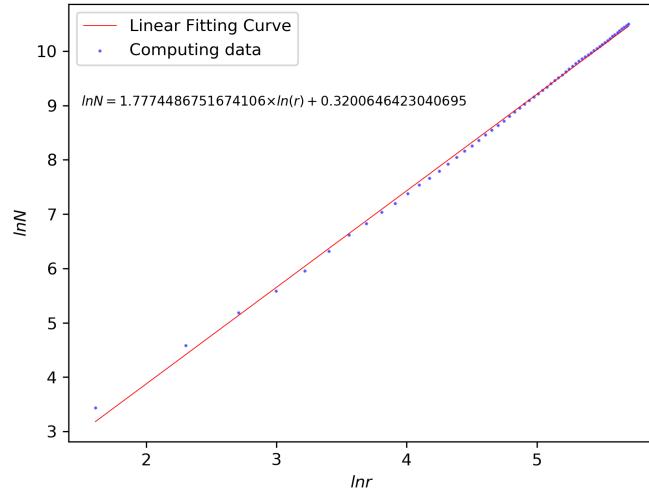


图 16: $R_{gen} = r_{cluster} + 5, R_{esp} = R_{gen} + 2$ 的 Sandbox 法计算分形维数

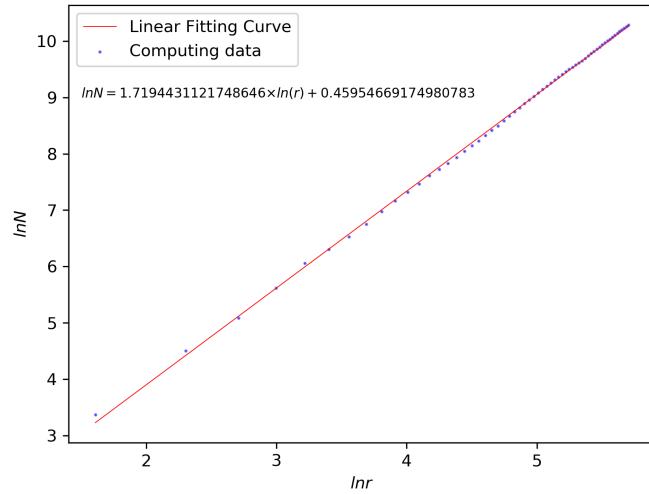


图 17: $R_{gen} = r_{cluster} + 5, R_{esp} = R_{gen} + 50$ 的 Sandbox 法计算分形维数

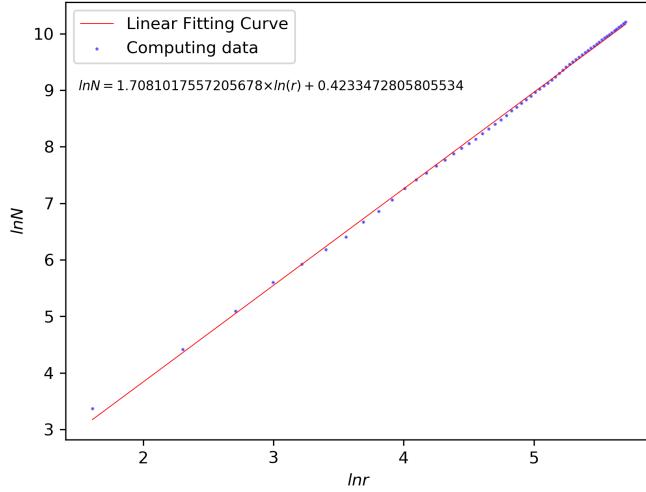


图 18: $R_{gen} = r_{cluster} + 5, R_{esp} = R_{gen} + 200$ 的 Sandbox 法计算分形维数

可以明显的看出随着逃逸半径的增大，分形维数越来越小，即图形越来越稀疏。

而产生这种现象的原因，个人认为实际团簇粒子生长过程或逃逸半径很大的时候，某随机粒子可游走到相对团簇粒子比较远的距离后并折返回来，到达与其实位置相差角度很大的地方（以团簇中心为原点，极坐标下的角度），从而有一部分随机粒子可以绕过在其实位置周围的团簇分支，到达别的地方。而若逃逸半径很小的时候，会导致这部分粒子几乎消失，从而粒子几乎只能到达在其实位置附近的团簇分支上。由于团簇的各项不均匀性会不断放大，故逃逸半径较小的模拟结果会使得团簇更加密集，此种现象应该与实际不符。故我们设置 DLA 模拟的逃逸半径时，应兼顾比较符合实际且计算量不会很大的值。

另外，本人一开始选取的边界为正方形边界，故导致模拟出来的团簇并不各项均匀：

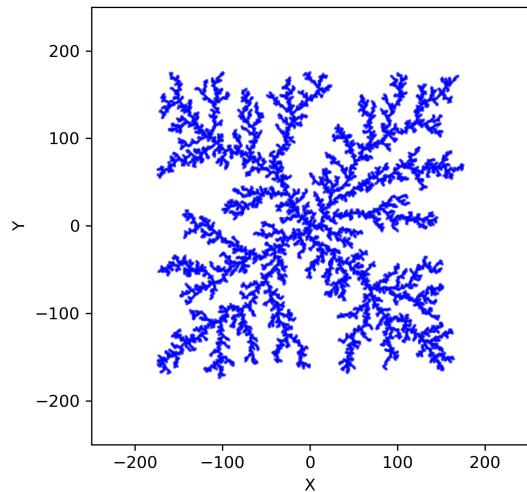


图 19: 正方形边界 10^4 个粒子的可视化模拟结果

不均匀性的出现也十分好理，因为在正方形边缘上均匀分布的随机点在角度上的分布显然不均匀，4 个角处的分布更加密集。

5 心得与体会

通过此次作业，对 DLA 模型有了更加深刻的认识，且对计算分形维数的方法更加熟练。更加深刻的体会到计算模拟程序中模拟更加准确与耗费资源时间之间的矛盾。

6 附录

A DLA 模拟 C 语言源程序

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <string.h>
5 #define a 16807
6 #define b 0
7 #define m 2147483647
8 #define r (m%a)
9 #define q (m/a)
10 #define Pi 3.1415926
11 #define ROUND(x) ((x)>=0?(int)((x)+0.5):(int)((x)-0.5))
12 #define GETMATELEM(base,i,j,imax) ((*base + (i) * (imax) + (j)))
    //取二维数组元素
13 #define LEN 1500 //DLA位置数组的 X or Y 轴长度的一半
14
15
16 //利用/dev/random 产生"真"随机数
17 int my_realrandom(int ran[],int n){
18     FILE *fp1 = fopen("/dev/random","r");
19     for(int i=0;i<n;i++){
20         fread(&ran[i], 1, 4, fp1);
21     }
22     fclose(fp1);
23     return 0;
24 }
25
26
27
28 int my_filewriter_int(char str[],int num[],int n){
29     FILE * fp;
30     fp = fopen(str,"w+");
31
32     for(int i=0;i<n;i++)
33     {
```

```

34     if (i == (n-1)){
35         fprintf(fp,"%d",num[i]);
36         break;
37     }
38     fprintf(fp,"%d,",num[i]);
39
40 }
41 fclose(fp);
42 return 0;
43 }
44
45
46 int my_filereader_int(char str[],int num[],int n){
47     FILE * fp;
48     fp = fopen(str,"r");
49
50     for(int i=0;i<n-1;i++)
51     {
52         fscanf(fp,"%d,",&num[i]);
53
54     }
55     fscanf(fp,"%d",&num[n-1]); //最后一个数据后不加 ","
56     fclose(fp);
57     return 0;
58 }
59
60
61
62 // Schrage 方法产生随机数
63 int my_schrage(double ran[],int seed,int n){
64     for (int i = 0; i < n - 1; i++) {
65         if (seed >= 0) {
66             ran[i] = seed / (double) m;
67         } else {
68             ran[i] = (seed + m) / (double) m;
69         }
70         if(seed == m-1){
71             if(a >= b){ //由于Schrage方法只对z in

```

```

(0,m-1)成立, 故这里要讨论z == m-1的情况
72     seed = m + (b-a) % m;
73 }
74 else seed = (b-a) % m;
75
76 }
77 else seed = ((a * (seed % q) - r * (seed / q)) + b % m) %
    m; //递推式
78 }
79 if (seed >= 0) {
80     ran[n-1] = seed / (double) m;
81 } else {
82     ran[n-1] = (seed + m) / (double) m;
83 }
84 return 0;
85 }

86
87
88
89
90
91
92 // Schrage 方法迭代器
93 int my_schrage_iter(int seed){
94     if(seed == m-1){
95         if(a >= b){ //由于Schrage方法只对z in
96             (0,m-1)成立, 故这里要讨论z == m-1的情况
97             seed = m + (b-a) % m;
98         }
99     }
100 else seed = ((a * (seed % q) - r * (seed / q)) + b % m) % m;
    //递推式
102
103 return seed;
104 }
105

```

```

106
107 int randomparticle(double ran,int min,int *i,int *j){
108     //产生随机粒子的初始位置
109     double theta = 2*Pi*ran;
110     //为了使初始位置更加均匀，我们取圆形边界，产生均匀分布的随机角度值
111     double x = min*cos(theta);
112     //以min为半径，计算此初始位置相对于原点的x,y值
113     double xf = x-floor(x); //x的小数部分
114     double y = min*sin(theta);
115     double yf = y-floor(y); //y的小数部分
116     *i = (int)( LEN + floor(x) + floor(2*xf) );
117     //结束后，i,j变为初始位置的矩阵坐标
118     *j = (int)( LEN + floor(y) + floor(2*yf) );
119
120
121     /* //以下为正方形边界
122     int n = (int)floor(8*min*ran); //粒子位于边缘处的哪个位置
123     //printf("n= %d\n",n);
124
125     if (n < 2*min+1){ //一下过程均为找到此位置的行号和列好
126         *i = LEN-min;
127         *j = LEN-min+n;
128         return 0;
129     }
130     else n -= 2*min;
131     if (n < 2*min+1){
132         *i = LEN-min+n;
133         *j = LEN+min;
134         return 0;
135     }
136     else n -= 2*min;
137     if (n < 2*min+1){
138         *i = LEN+min;
139         *j = LEN+min-n;

```

```

140     *j = LEN-min;
141     */
142     return 0;
143 }
144
145
146 int particleRW(int *o,int *p,int max,int *position,int seed){
    //产生的随机粒子随机行走器
    //o,p为初始粒子的位置，max为判定粒子逃逸的范围，position为总位置坐标数组
147     int i = *o;
148     int j = *p;
149     double ran;
150     int flag = 0;
151     for(int k = -1;k<2;k++){ //查看粒子周围是否有团簇粒子，有的话flag
        != 0
152         for(int l = -1;l<2;l++){
            flag += GETMATELEM(position, (i-k), (j-l), (2*LEN+1));
        }
155     }
156     while (flag == 0 && sqrt( pow(i-LEN,2)+pow(j-LEN,2) ) < max ){
        //粒子周围没有团簇粒子且在逃逸区域以内，则继续随机行走
        if (seed >= 0) {
            ran = seed / (double) m;
        }
160         else {
            ran = (seed + m) / (double) m;
        }
163         if(ran<0.25) j +=1 ; //4个方向随机行走
        else if(ran<0.5 && ran >= 0.25) j -= 1;
        else if(ran<0.75 && ran>=0.5 ) i += 1;
        else i -= 1;
167         seed = my_schrage_iter(seed);
168         for(int k = -1;k<2;k++){
            //查看粒子周围是否有团簇粒子，有的话flag != 0
            for(int l = -1;l<2;l++){
                flag += GETMATELEM(position, (i-k), (j-l), (2*LEN+1));
            }
        }
172     }

```

```

173     }
174     if(flag == 0) return 1; //此时粒子逃逸，需要重新模拟
175     *o = i;
176     *p = j;
177     return 0; //结束后i,j变为此粒子最终的位置
178 }
179
180 int boxcount(int* position,int *count,int Len,int maxr){
//盒计数法统计N, Len为位置二维数组的半径, maxr, 为统计范围的半径
181     int flag = 0;
182     int n = 0; //区域内是否有粒子的累加器
183     int len;
184     if( (2*Len) < 2*maxr ){
185         printf("LEN is too short to count");
186         return 1;
187     }
188     for(int i = 1; i < 2*maxr + 1; i = i*2){
189         len = 2*maxr/i;
190         for(int k = 0 ;k < i ; k++){ //方格计数
191             for(int l = 0 ;l < i ; l++){
192                 n = 0;
193                 for(int o = Len - maxr + len*k; o < Len - maxr
194                     +len*(k+1); o++){
195                     for(int p = Len - maxr + len*l; p < Len - maxr
196                         +len*(l+1); p++){
197                         n += GETMATELEM(position, o, p, (2*Len+1));
198                     }
199                 }
200                 if (n != 0) count[flag] += 1;
201             }
202             flag++;
203         }
204         return 0;
205     }
206
207

```

```

208 int sandboxcount(int*position, int*count,int rmax,int Len){
209     //sandbox法统计N, rmax为统计区域的半径最大值, Len为位置二维矩阵的半径
210     int radius = 5; //起始统计范围半径为5
211     int flag = 0;
212     int n = 0;
213     for(;radius < rmax + 1;radius += 5){ //统计范围半径依次增加5
214         n = 0;
215         for(int i = Len - radius; i < Len + radius; i++){
216             for(int j = Len - radius; j < Len + radius; j++){
217                 n += GETMATELEM(position, i, j, (2*Len+1));
218             }
219         }
220         count[flag] = n;
221         flag++;
222     }
223     return 0;
224 }
225
226
227 int main(int argc, const char * argv[]) {
228
229
230     int N = 50000; //粒子总数
231     int min,max; //分别对应产生粒子的区域和判断粒子逃逸的范围
232     min = 5;
233     max = 205 ;
234     int i,j;
235     int *position = malloc(sizeof(int)*(2*LEN+1)*(2*LEN+1));
236     //存放位置坐标的数组
237     for(int i =0;i<(2*LEN+1)*(2*LEN+1);i++){ //位置坐标初始化
238         position[i] = 0;
239     }
240
241     GETMATELEM(position, (LEN), (LEN), (2*LEN+1)) = 1;
242     //设定原点(0,0)处初始有一粒子

```

```

243     int *seed = malloc(sizeof(int)*N); //随机数种子
244     my_realrandom(seed, N);
245     double *ran = malloc(sizeof(double)*N); //每次粒子的初始位置
246     my_schrage(ran, seed[0], N);
247     int flag = 0;
248     for(int k =0;k<N;k++){
249         flag = 0;
250         randomparticle(ran[k], min,&i, &j);
251         flag = particleRW(&i, &j, max, position, seed[k]);
252         if(flag == 1){
253             while (flag == 1) {
254                 flag = 0;
255                 my_realrandom(&seed[k], 1);
256                 flag = particleRW(&i, &j, max, position, seed[k]);
257             }
258         }
259         GETMATELEM(position, (i), (j), (2*LEN+1)) = 1;
260         if(abs(i-LEN) > min-5) min = 5 + abs(i-LEN);
261         if(abs(j-LEN) > min-5) min = 5 + abs(j-LEN);
262         max = min + 200 ;
263     }
264     my_filewriter_int("p.dat", position, (2*LEN+1)*(2*LEN+1));
265
266     return 0;
267 }
```

B 分形维数统计 C 语言程序源码

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 int my_filereader_int(char str[],int num[],int n){
6     FILE * fp;
7     fp = fopen(str,"r");
8
9     for(int i=0;i<n-1;i++)
10    {
11        fscanf(fp,"%d,",&num[i]);
12    }
13
14    fscanf(fp,"%d",&num[n-1]); //最后一个数据后不加 ","
15    fclose(fp);
16    return 0;
17 }
18
19 int boxcount(int* position,int *count,int Len,int maxr){
//盒计数法统计N, Len为位置二维数组的半径, maxr, 为统计范围的半径
20     int flag = 0;
21     int n = 0; //区域内是否有粒子的累加器
22     int len;
23     if( (2*Len) < 2*maxr ){
24         printf("LEN is too short to count");
25         return 1;
26     }
27     for(int i = 1; i < 2*maxr + 1; i = i*2){
28         len = 2*maxr/i;
29         for(int k = 0 ;k < i ; k++){ //方格计数
30             for(int l = 0 ;l < i ; l++){
31                 n = 0;
32                 for(int o = Len - maxr + len*k; o < Len - maxr
+len*(k+1); o++){
33                     for(int p = Len - maxr + len*l; p < Len - maxr
+len*(l+1); p++ ){

```

```

34             n += GETMATELEM(position, o, p, (2*Len+1));
35         }
36     }
37
38     if (n != 0) count[flag] += 1;
39 }
40 }
41 flag++;
42 }
43 return 0;
44 }
45
46
47 int sandboxcount(int*position, int*count,int rmax,int Len){
    //sandbox法统计N, rmax为统计区域的半径最大值, Len为位置二维矩阵的半径
48 int radius = 5; //起始统计范围半径为5
49 int flag = 0;
50 int n = 0;
51 for(;radius < rmax + 1;radius += 5){ //统计范围半径依次增加5
52     n = 0;
53     for(int i = Len - radius; i < Len + radius; i++){
54         for(int j = Len - radius; j < Len + radius; j++){
55             n += GETMATELEM(position, i, j, (2*Len+1));
56         }
57     }
58     count[flag] = n;
59     flag++;
60 }
61 return 0;
62 }
63
64
65
66 int main(int argc, const char * argv[]) {
67     int len = 512;
68     int Len = 1500;
69     int *position = malloc(sizeof(int)*(2*Len+1)*(2*Len+1));
70     int *count1 = malloc(sizeof(int)*11); //盒计数法统计N存放数组

```

```
71     int *count2 = malloc(sizeof(int)*60); //Sandbox法统计N存放数组
72
73     for (int i = 0;i<11;i++){ //数组初始化
74         count1[i] = 0;
75     }
76
77     for (int i = 0;i<60;i++){ //数组初始化
78         count2[i] = 0;
79     }
80
81     my_filereader_int("p-1500-5*104-5-2*min.dat", position,
82                         (2*Len+1)*(2*Len+1)); //读取之前生成的位置矩阵
83
84     boxcount(position, count1, Len, len);
85     sandboxcount(position, count2, 300,Len);
86
87     my_filewriter_int("box.txt", count1, 11);
88     my_filewriter_int("sandbox.txt", count2, 60);
89
90     return 0;
91 }
```

C 可视化绘图及数据处理 Python 程序源码

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 import math
4
5 plt.rcParams['savefig.dpi'] = 300 #图片像素
6 plt.rcParams['figure.dpi'] = 300 #分辨率
7 # 默认的像素: [6.0,4.0], 分辨率为100, 图片尺寸为 600&400
8 fig1 = plt.figure()
9 fig2 = plt.figure()
10 fig3 = plt.figure()
11 max = 500 #团簇位置信息二维矩阵半径
12 ax1 = fig1.add_subplot(111)
13 ax2 = fig2.add_subplot(111)
14 ax3 = fig3.add_subplot(111)
15
16
17
18 X = []
19 Y = []
20 y = []
21 box = []
22 sandbox = []
23 xbox = []
24 xsandbox = []
25
26 xlabel = []
27 xtsandbox = []
28
29
30 with open('problem 13/p-500-104-1.dat', 'r') as f:
31     while True:
32         lines = f.readline() # 整行读取数据
33         if not lines:
34             break
35         y = [float(i) for i in lines.split(',')]] # 将整行数据分割处理
36         y = np.array(y) # 将数据从list类型转换为array类型。
```

```

37
38
39 with open('problem 13/box-5-3*min.txt', 'r') as f:
40     while True:
41         lines = f.readline() # 整行读取数据
42         if not lines:
43             break
44         box = [float(i) for i in lines.split(',')]] #
45             将整行数据分割处理
46         box = np.array(box) # 将数据从list类型转换为array类型。
47
48 with open('problem 13/sandbox-5-3*min.txt', 'r') as f:
49     while True:
50         lines = f.readline() # 整行读取数据
51         if not lines:
52             break
53         sandbox = [float(i) for i in lines.split(',')]] #
54             将整行数据分割处理
55         sandbox = np.array(sandbox) # 将数据从list类型转换为array类型。
56
56 for i in range(pow(2*max+1, 2)):
57     if y[i] != 0:
58         X.append(i % (2*max+1)-max)
59         Y.append(max - i/(2*max+1))
60
61 box = np.delete(box, 10) #是否排除盒计数法最后一个数据点
62 print(box)
63
64 box = np.log(box) # 盒计数法横轴数据点生成
65 xbox = np.ones(len(box))
66 for i in range(len(box)):
67     xbox[i] = math.log(pow(2, i))
68
69 sandbox = np.log(sandbox) # Sandbox法横轴数据点生成
70 xsandbox = np.ones(60)
71 for i in range(60):
72     xsandbox[i] = math.log(5 + i*5)

```

```

73
74
75 [ab, bb] = np.polyfit(xbox, box, 1) # 两计算分形维数方法的线性拟合
76 [asb, bsb] = np.polyfit(xsandbox, sandbox, 1)
77 print([ab, bb])
78 print([asb, bsb])
79
80
81 ax1.scatter(X, Y, c='b', label='', s= 1, alpha=0.5, marker='o') #
82 DLA模拟结果散点图
83 ax1.set_xlabel('X')
84 ax1.set_ylabel('Y')
85 ax1.set_xlim(-max/2, max/2)
86 ax1.set_ylim(-max/2, max/2)
87 ax1.set_aspect('equal')
88 fig1.savefig("1.png")
89
90
91 ax2.scatter(xbox, box, c='b', label='Computing data', s= 1,
92 alpha=0.5, marker='o') # 盒计数法对数图
93 ax2.plot(xbox, ab*xbox+bb, lw=0.5, color='r', label='Linear Fitting
94 Curve')
95 ax2.set_xlabel(r'$\ln(1/\epsilon)$')
96 ax2.set_ylabel(r'$\ln N$')
97 ax2.legend(loc=2)
98 ax2.text(0, 9, r"$\ln N = $" + str(ab) + r'$ \times \ln(1/$
99 r'$\epsilon)$' + str(bb), fontsize=8, color="black")
100 fig2.savefig("2.png")
101
102 ax3.scatter(xsandbox, sandbox, c='b', label='Computing data', s= 1,
103 alpha=0.5, marker='o') # sandbox对数图
104 ax3.plot(xsandbox, asb*xsandbox+bsb, lw=0.5, color='r',
105 label='Linear Fitting Curve')
106 ax3.set_xlabel(r'$\ln r$')
107 ax3.set_ylabel(r'$\ln N$')
108 ax3.text(1.5, 9, r"$\ln N = $" + str(asb) + r'$ \times \ln(r) + $" + str(bsb),
109 fontsize=8, color="black")
110 ax3.legend(loc=2)

```

```
104 fig3.savefig("3.png")
```
