

# 2023-2024 学年秋季学期

## 《计算思维实训（1）》(0830A030)

### 课程报告

成绩 (百分制)
-------------

学号	22121799	学院	计算机工程与科学学院
姓名	金义杰	手工签名	
报告题目	无尺度网络的基本研究与验证		
实训报告成绩 50%	实训过程描述：清晰、全面。 (5%)		
	报告主要部分：1、计算思维实例叙述清楚、有意义；2、分析到位；3、思路独特或有创意；4、关键实现技术描述清楚详细；5、内容丰富；6、不直接粘贴所有代码；7、代码有注释或说明。 (25%)		
	实训收获体会：感受真实、深刻，建议有价值。 (10%)		
	书写格式：书写规范、用词正确、无明显的错别字，图表、代码清晰规范，格式协调、效果好，参考文献书写、引用规范合理。 (10%)		
工作实绩 50%	1、平时表现、进步情况 (25%)，2、工作实绩 (25%)		
教师对该生工作实绩简要评述：			
教师签名：			
日期： 年 月 日			

# Part 1. 本学期实训过程概述

## 1. 实训总体概况、实训过程

本次实训我进行了无尺度网络的建模与数据分析，最终成功建立模型并分析其数据，复现了论文中的成果。我的研究过程主要分为以下几个阶段：

1. 初览论文：大致明了所需构建模型的特点，确定建模方式，即为 C++ 建模获得数据，Python 数据分析可视化；
2. 查阅更多资料：明了其中的数学原理，并思考出构建模型所需的各项参数；
3. 思考数据结构的设计：最开始想要使用图来构成一个网络，但是发现超出现阶段自己能力范围，随后选择了线性结构作为节点连接的主结构，用边向量来实现拓扑结构的思路。
4. 着手建模：经过多次修改，从手动构建数据结构，利用指针来创造边，但是发现自己的各项算法开销太大，导致数据计算过慢，之后重新写全新的程序，反复重构三次得到了现阶段自己构建出的最快且最准确的模型，并且同时做好了数据的保存形式，用以后续分析。
5. 数据分析：读取先前的数据，利用 Python 中常用的数据分析包进行分析与可视化，涉及一些数据清洗和处理的问题，可视化的过程是早在建模时就已经考虑周全，数据的保存形式都是为可视化和分析服务。
6. 书写报告与总结：过程中发现部分数据体现特性不充分，因此多进行了若干实验得出更多数据用以验证，并学习了一些书写论文的规范性要求，并且发现了自己大量的不足之处例如代码规范性、选用数据结构的规范性。

实训过程和报告撰写完成之后，针对自己总结的不足的地方，我还对自己已学习的知识进行拓展，并探索新的知识、新的领域，例如上面提到的数据结构，真正的图结构，邻接表实现网络生成进行了探索，开始学习一些共享内存多线程并行计算的实现方式，这些方式方法都会对我的算法运算速度有较大的提升，后续的更深入学习可以让我对已完成的模型进行优化和调整，得到更为长足的进步。

## 2. 对计算思维的认识

从本次实训中，结合我过往的认识，我总结出一下一段话用来解释我心中的计算思维：计算机科学是一种富有高度交叉性的科学，计算思维也是其核心思想之一，是计算机从业者和学习者的必须技能。其中包括了很多方面，例如逻辑能力与思维、建模抽象能力、算法能力、代码能力、数学基础（离散、组合、微积分、线性代数、概率论等）、计算机基础知识的充分认识等……而计算思维，正是这些知识与思维的有机结合。

### 3. 实训体会及建议

#### 体会：

本次计算思维实训，充分拓展了我的思维灵活性，磨练了我的 C++ 面向对象编程和算法实现能力，让我对建立模型有了一个初步的认识，并且初步学会了如何阅读、理解、应用论文中讲述的知识，大大提高了我的信息检索能力，锻炼了我数据分析的技巧。可以说，本次计算思维实训让我受益颇多。

当然，我还有很多不足的地方，例如数据结构的选择上，后续我可以选择更高深的数据结构（例如真正的图结构）和算法（更多高深的查找算法，例如利用图邻接表来进行网络生成的算法）进行更多研究，以及我可视化网络、度分布的方式较为粗糙，还有代码规范性不足且使用语法陈旧、无并行代码，这都是我有待进步的地方。

#### 建议：

首先谈我自己的情况，仅针对我个人来说，我没有任何建议，因为在本次实训中，我学到了很多有意义的知识，并得到了相对于过去的自己而言，很大的提升。

我认为老师对我的指导和实训选题都是十分符合我现阶段实力的，正好比我实训前的实力更高，但又在我已学习知识范围内，通过老师一定的指点还是可以大部分独立完成整项实训任务，并且没有过多的依靠前人已经留下的成品包库（例如深度学习、AI 实训会用到的 PyTorch 和 Tensorflow 深度学习框架等）简化我的各项操作，而是更多靠自己的思维实现整个项目的研究，我认为这样的规划是很好的。

因此，对我来说，本次计算思维实训从老师的选题到我自己实现后对自己带来的提升，我都认为是很好的，我对我的导师及其选择没有过多建议。

然而据我了解，这个情况也许并不普适于所有同学，下面是我诚挚的建议。

所谓计算思维实训，培养的是我的计算思维，而非简单的调包调参能力（在没有 ml、dl 基础知识支持的情况下做 AI 相关工作何尝不是简单调参调包）。然而据我了解，部分同学的任务更多是 AI 相关，抑或是进行 Java 开发的搬砖工作（较为容易的工作，比课设 C++ 还要简单），我认为这样的话，对计算思维的培养是稍显不足的。

在实训选题的选择上，选择更多符合我们现阶段能力，且更多依靠自身能力可以实现且带来提升的项目，也许会是更好的。

以上便是我的建议，如有不得体之处，恳请海涵。

## Part 2. 综合实训报告

# 无尺度网络的基本研究与验证

22121799 金义杰

计算机工程与科学学院

**摘要:** 在现代科技时代，网络已经成为连接人与信息的重要工具。针对无尺度网络在复杂网络中的广泛应用，本文运用数学建模、面向对象程序设计思想、以及数据分析方法，验证了无尺度网络模型在理论上的正确性，并通过针对大规模复杂网络的数据分析，得出了其符合无尺度网络模型分布特点的结论。此外，本文还介绍了无尺度网络的基本原理和概念，以及在现实生活中的典型应用。总之，本文为深入了解无尺度网络提供了具有参考价值的研究结果和实践指南。

**关键词:** 无尺度网络；网络建模；度分布；网络可视化；随机图论；

## 1 关于无尺度网络

无尺度网络（BA 无标度网络），是一种经典的复杂拓扑网络模型，由艾伯特-拉斯洛·巴拉巴西（Albert-László Barabási）于 1998 年与雷卡·阿尔伯特（Réka Albert）共同发现 [1]。

在无尺度网络被提出前，最经典的复杂拓扑网络模型是 ER ( Erdős–Rényi ) 随机网络 [1]，在 ER 随机网络模型的理论中网络中，各个节点以相对简单形式互相连接，数学理论是基于随机图论。

尽管人们已经基于 ER 随机网络进行了很深入的研究，但在计算机的快速发展推动下，真实世界的网络拓扑信息开始愈发复杂，单纯的随机网络已经很难准确表达现实世界中的网络特性。巴拉巴西和阿尔伯特（后简称 BA）便是在这个背景下，以几个大型数据库为研究基础，针对多个方面进行了数据统计和网络建模分析。

在他们研究之前不久，人们已经提出了 WS ( Watts-Strogatz ) 小世界网络模型，这个模型中，节点按照边的长度进行选择性连接，其揭示了人与人之间经过很少的几层关系便可以相互关联，体现了小世界的特性 [1][3]。但是，其缺乏了对度的判断，这个度，就是每个节点的边数量。

BA 在对数据库进行研究的过程中，发现了节点之间的连接具有聚集的特点，经过深入研究，他们发现这个聚集的特点来自于每个节点的连接方式，他们并不仅仅只是简单地按照距离远近选择连接，而是更多的按照度的分布选择连接。BA 统计数据之后，

绘制出度分布图，发现了网络节点的度分布具有幂律分布<sup>1</sup>的特性，这样的特性造成了网络中存在大量的普通节点和少量的超级节点（Hubs），这些特点便是网络的无尺度特性 [1][2]。

无尺度网络模型一经发表便广受关注，随着互联网的发展，其各种特性也被验证。鉴于其模型的优秀性，我基于 C++ 进行复杂网络建模 [5][6]，对其特性进行验证并统计数据，利用 Python 进行数据分析和绘图可视化研究 [4]，并成功验证无尺度网络的无尺度特性。

## 2 网络建模（仅进行核心功能讲解，完整代码在附录中）

网络的建模基于 BA 的 Scale-Free Networks 论文 [1][2]。选择使用 C++ 进行建模 [5][6]。

### 2.1 核心类成员设计

建立一个节点类，私有成员 `int k` 记录当前结点的度数，`vector<vector<double>> edges` 记录指向的节点的坐标，`vector<double> position` 记录本节点坐标，验证无尺度特性主要利用 `k`，实现网络可视化主要利用 `edges` 和 `position`。

```
1. class Node {
2.     private:
3.     int k; // 记录度数
4.     vector<vector<double>> edges; // 记录指向节点的坐标
5.     vector<double> position; // 记录本节点坐标
6.     public:
7.     Node(): k(0) { // 默认构造，k 初始化为 0，并初始化坐标
8.         position.resize(2, RandDouble(range));
9.     }
10.    Node(int n): k(0) { // 加参数构造，k 初始化为 0，仅初始化坐标为空，后续生成坐标
11.        position.resize(2);
12.    }
13.    // 为了方便，全部使用友元函数
14.    friend void Link(std::vector<Node> &sfn, Node &p);
15.    friend void GetPosition(Node &p);
16.    friend void LinkEdges(std::vector<Node> &sfn, const int &index, Node &p);
17.    friend bool IsAvai(Node x, Node p);
18.    friend void CalculateDistribution(std::vector<Node> sfn, std::vector<double> &dis);
19.    friend void SaveDistribution(std::vector<Node> sfn, const std::string &filename);
20.    friend void SavePosition(std::vector<Node> sfn, const std::string &filename);
21.};
```

<sup>1</sup> 幂律分布： $P(X = x) \propto x^{-\alpha}$ ，横纵坐标均取对数之后呈负相关即  $\log P(X = x) \propto -\alpha \log x$

主要操作函数包括：

1. 核心操作函数：**Link()** 在拓扑空间和内存线性空间上连接每个节点。拓扑空间连接使用 **edges** 存放地址实现构建，线性空间使用 **vector.push\_back()** 线性添加。拓扑空间用于构建网络，线性空间用于存储网络查找结点。
2. 辅助函数：**CalculateDistribution()** 计算节点的度分布情况，**IsAvai()** 判断某节点是否已被当前正在连入网络的节点连接，**GetPosition()** 随机生成每个节点的地址。
3. 存储函数：**SaveDistribution()** 用于记录每个节点的度分布情况到 csv 文件中，**SavePosition()** 用于记录每个节点的地址和其指向节点的地址。

其中核心操作函数 **Link()** 实现如下：

```
1. void Link(vector<Node> &sfn, Node &p) {  
2.     int cnt = 0; // 记录成功连接的次数  
3.     GetPosition(p);  
4.     while(cnt < edge) {  
5.         _cnt_++;  
6.         int index = RandInt(sfn.size()); // 随机生成 0~sfn.size() - 1 之间整数选取已有节点  
7.         double rand_num = RandDouble(); // 随机生成 0~1 之间浮点数判断是否连接  
8.         double prob_link;  
9.         if(k_sum > 4) {  
10.             prob_link = (double) sfn[index].k / k_sum;  
11.         } else {  
12.             if(k_sum == 0) {  
13.                 prob_link = 1.0;  
14.             } else prob_link = 1 - 1.0 / k_sum;  
15.         }  
16.         // ShowInfo(index, rand_num, prob_link, cnt);  
17.         if(prob_link >= rand_num) {  
18.             if(cnt == 0) {  
19.                 LinkEdges(sfn, index, p);  
20.             } else {  
21.                 bool tmp = IsAvai(sfn[cnt], p);  
22.                 if(tmp) LinkEdges(sfn, index, p);  
23.             }  
24.             cnt++;  
25.         }  
26.     }  
27.     sfn.push_back(p);  
28. }
```

## 2.2 随机数生成

为了实现各项连接，设计了三个随机数生成函数：

```
1. double RandDouble();
2. double RandDouble(unsigned long long range);
3. int RandInt(int num);
```

三个函数分别的作用：

1. **RandInt (int num)**: 基于现阶段已连入网络节点的总数量 **num** 生成小于 **num** 的随机整型数据，用于随机查找节点 X 进行连接请求
2. **RandDouble ()**: 随机生成一个 0 ~ 1 之间的双精度浮点数，用于判断新添加节点 A 是否连接某节点 X，如果 A 随机浮点数小于等于 X 的度分布占总度数的比例且先前没有与该节点连接过则进行连接，否则跳过。
3. **RandDouble (unsigned long long range)**: 基于外部输入 **range** (总节点数) 的值，随机生成双精度浮点数，用于生成地址，得到随机地址坐标，从而实现网络可视化。

## 2.3 数据处理参数

为了实现各项操作，添加几个全局变量简化操作：

```
1. static unsigned long long range = 0;
2. static int edge = 0;
3. static unsigned long long _cnt_ = 0;
4. static unsigned long long k_sum = 0;
```

几个全局静态变量的作用如下：

1. **range**: 记录外部传入的总节点数，用于各项相关计算
2. **edge**: 记录外部传入的单节点连出有向边数量，用于各项连接功能实现
3. **\_cnt\_**: 记录操作次数，用于在终端可视化网络生成过程的进度情况
4. **k\_sum**: 记录总度数，用于计算度分布情况和判断连接等操作

## 2.3 数据记录

### 2.3.1 度分布数据记录

首先基于现有私有成员中的数据进行度分布计算

```
1. void CalculateDistribution(vector<Node> sfn, vector<double> &dis) {
2.     int k_max = sfn[0].k;
3.     for(int i = 0; i < sfn.size(); i++) {
4.         if(sfn[i].k > k_max) k_max = sfn[i].k;
5.     }
6.     dis.resize(k_max + 1, 0);
```

```

7.     for(int i = 0; i < sfn.size(); i++) {
8.         dis[sfn[i].k]++;
9.     }
10. }
```

先得出最大度值 **k\_max**, 将 **dis** 给 **resize**, 随后以每个节点的 **k** 为下标查找 **dis** 对应下标位置并加 1, 从而记录每个 **k** 值对应节点数量。最后度分布情况保存到向量 **dis** 中。

随后将计算好的数据存储到 **csv** 文件中便于后续分析

```

1. void SaveDistribution(vector<double> sfn, const string &filename) {
2.     ofstream ofile(filename);
3.     if(!ofile.is_open()) {
4.         cout << "FAIL TO OPEN FILE: " << filename << endl;
5.         return;
6.     } else {
7.         for(int i = 0; i < sfn.size(); i++) {
8.             ofile << i << "," << sfn[i] << endl;
9.         } cout << "DISTRIBUTION DATA SAVED IN FILE: " << filename << endl;
10.    }
11. }
```

将 **dis** 传入函数为形参 **sfn**, 迭代写入数据, 向量下标为度数 **k**, 对应数据为 **sfn[i]**, 以一对数据的形式写入, 随后换行。

### 2.3.2 地址坐标数据记录

记录地址的过程相对复杂, 首先存入本节点的地址坐标, 随后存入每个有向边指向的节点的地址坐标。

```

1. void SavePosition(vector<Node> sfn, const string &filename) {
2.     ofstream ofile(filename);
3.     if(!ofile.is_open()) {
4.         cout << "FAIL TO OPEN FILE: " << filename << endl;
5.         return;
6.     } else {
7.         for(int i = 0; i < sfn.size(); i++) {
8.             ofile << sfn[i].position[0] << "," << sfn[i].position[1] << ",";
9.             if(sfn[i].edges.size() < edge) {
10.                 for(int j = 0; j < sfn[i].edges.size(); j++) {
11.                     ofile << RandDouble(range) << "," << RandDouble(range) << ",";
12.                 } ofile << endl;
13.             } else {
14.                 for(int j = 0; j < sfn[i].edges.size(); j++) {
15.                     ofile << sfn[i].edges[j][0] << "," << sfn[i].edges[j][1] << ",";
16.                 } ofile << endl;
17.             }
18.         }
19.     }
20. }
```

```
17.         }
18.     }
19.     cout << "POSITION DATA SAVED IN FILE: " << filename << endl;
20. }
21. }
```

如果边向量长度小于指定边数量则代表是初始节点，初始节点不会主动连出有向边，由于初始节点仅有 1 个位数，为其生成几个随机边指向地址便于数据处理（不会影响度分布，因为此时是存储地址函数）。

等于则是后续连入节点，正常迭代存入，一个节点信息存完则换行。

存储操作的形式是为了后续 Python 数据分析服务。

### 3 数据处理与分析

在 C++ 建模完成之后，进入 Jupyter Notebook 导入已有数据进行数据分析，调用 Python 库 Pandas（数据分析库），NumPy（科学计算库），matplotlib.pyplot（可视化库）

```
1. import pandas as pd
2. import numpy as np
3. import matplotlib.pyplot as plt
```

#### 3.1 度分布分析

##### 3.1.1 读入数据及预处理

读入已保存度分布数据，随后进行预处理

预处理消除度为 0 的数据，对整个 **dataframe** 取 **log** 得到新的 **dataframe**

```
1. df = pd.read_csv("kDistribution_data.csv", names = ['K', 'P(k)'])
2. df
3. df = df[df['P(k)'] != 0]
4. df = df[df['K'] != 0]
5. df = np.log(df)
```

K	P(k)
0	0
1	0
2	65
3	69838
4	278
...	...
977	977
978	978
979	979
980	980
981	981

982 rows × 2 columns

图 1. 度分布数据 (100003 节点)

K	P(k)
2	0.693147
3	1.098612
4	1.386294
5	1.609438
6	1.791759
...	...
479	6.171701
533	6.278521
618	6.426488
735	6.599870
981	6.888572

167 rows × 2 columns

图 2. 度分布数据 (100003 节点)

### 3.1.2 可视化度分布情况

数据处理完毕，使用 **plt** 可视化 **dataframe**，绘制散点图

```

1. # 绘制散点图，并设置颜色为红色
2. plt.scatter(df['K'], df['P(k)'], color='cyan', alpha=0.7)
3. plt.grid()
4. # 设置坐标轴标签和标题
5. plt.xlabel('Degree (K)')
6. plt.ylabel('Number of Nodes')
7. plt.title('Degree Distribution')
9. # 显示图形
10. plt.show()
```

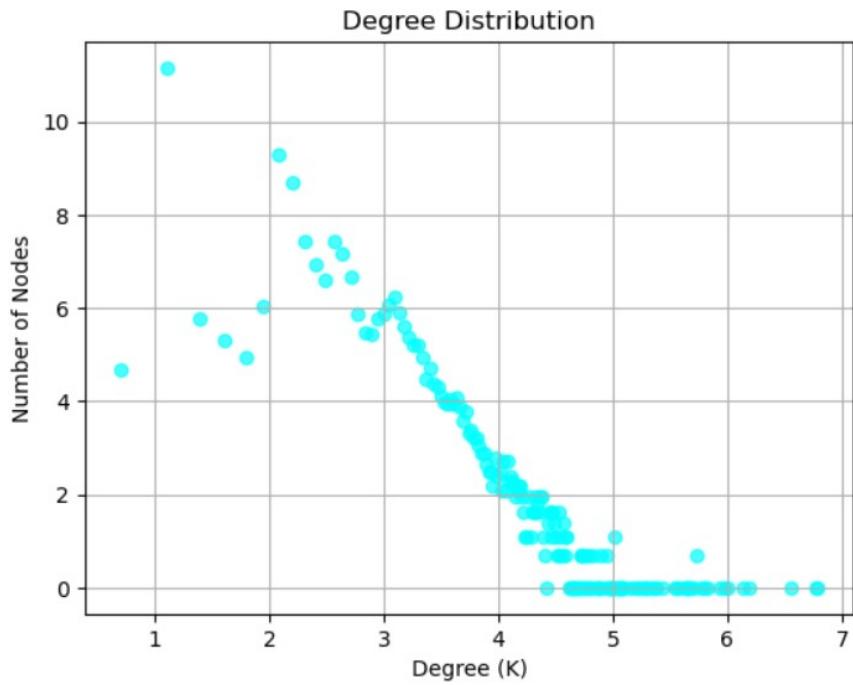


图 3. 度分布图 (1000003 节点)

可见，100003 节点的网络数据呈现出明显的幂律分布特点，从度分布上来看，网络建模成功实现了无尺度网络的构建。

下面是从 100 节点到 100000 节点的幂律分布情况变化（重新运行绘制）

这些图像也可以很好的体现出无尺度网络的生长过程

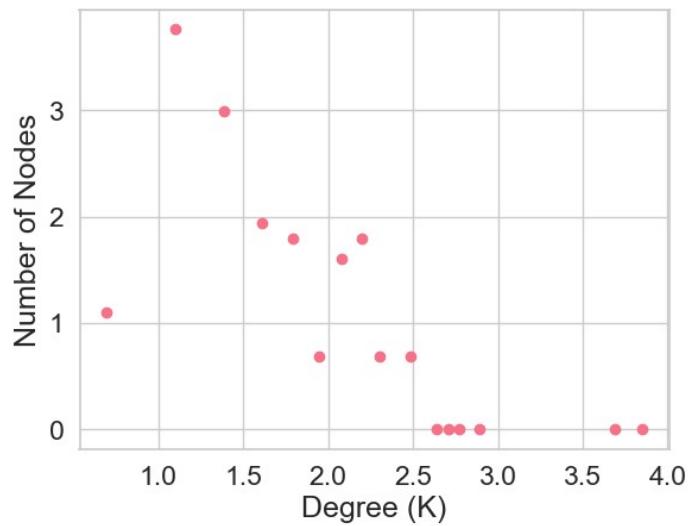


图 4. 度分布图 (103 节点)

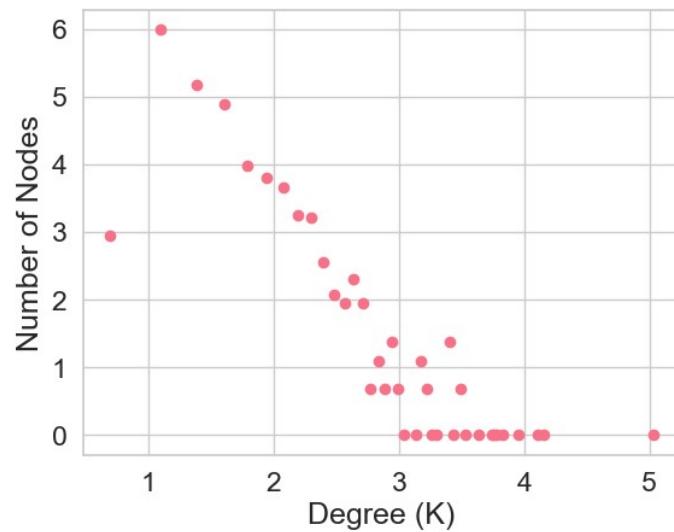


图 5. 度分布图 (1003 节点)

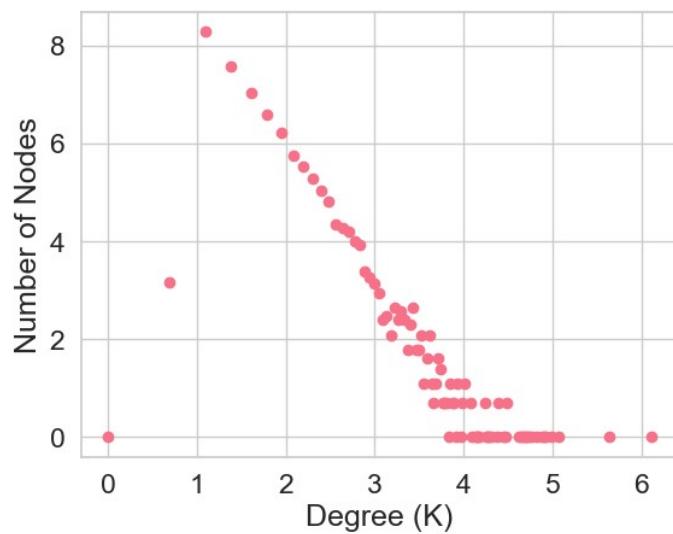


图 6. 度分布图 (10003 节点)

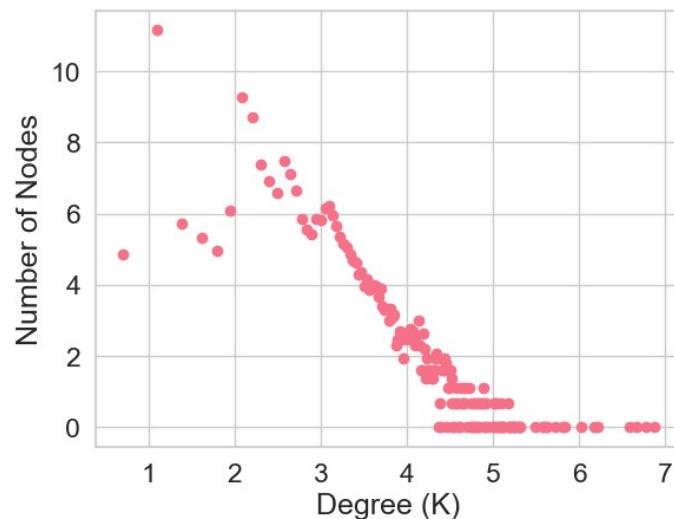


图 7. 度分布图 (100003 节点)

可见，构建的网络度分布数据基本符合幂律分布情况，其无尺度特性得以验证，后续进一步进行可视化分析用以交叉验证结论。

## 3.2 网络可视化分析

随后进行可视化绘制，首先读入数据，进行数据预处理和数据清洗，随后利用向量位置对应的方式进行绘图操作。

### 3.2.1 数据预处理与数据清洗

读入数据，进行数据预处理和数据清洗。

初始数据

```
1. df2 = pd.read_csv("locations.csv", header = None, names = ['x1', 'y1', 'x2', 'y2', 'x3', 'y3', 'x4',  
'y4', 'i'])
```

	x1	y1	x2	y2	x3	y3	x4	y4	i
0	9.820550e+09	9.820550e+09	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	2.537920e+09	2.537920e+09	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	1.569570e+09	1.569570e+09	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	2.011170e+08	2.838220e+08	9.820550e+09	9.820550e+09	2.537920e+09	2.537920e+09	2.537920e+09	2.537920e+09	NaN
4	8.096260e+09	5.768910e+09	9.426560e+09	2.419510e+09	1.709950e+09	7.773370e+09	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...
99998	3.274030e+09	4.636070e+09	2.358780e+09	1.424300e+09	8.499400e+09	6.787620e+09	3.492230e+09	5.445110e+09	NaN
99999	3.452250e+09	7.297280e+09	2.779020e+09	5.470140e+09	1.993770e+09	9.036230e+09	3.603630e+09	9.406110e+09	NaN
100000	4.779810e+09	3.403120e+09	7.005220e+09	4.406870e+08	8.474990e+08	3.443710e+09	9.124730e+09	3.526410e+09	NaN
100001	9.113130e+09	5.509810e+09	1.911070e+09	9.238870e+09	7.849670e+09	8.127690e+09	5.683160e+09	2.163460e+09	NaN
100002	4.398330e+09	4.649190e+09	5.470440e+09	7.729120e+09	2.971280e+09	9.949030e+08	5.259870e+09	2.405770e+09	NaN

100003 rows × 9 columns

图 8. 地址坐标初始数据（100003 节点）

```
1. df2 = df2.drop(columns=['i'])  
2. df2.fillna(method='bfill', inplace=True)
```

	x1	y1	x2	y2	x3	y3	x4	y4
0	9.820550e+09	9.820550e+09	NaN	NaN	NaN	NaN	NaN	NaN
1	2.537920e+09	2.537920e+09	NaN	NaN	NaN	NaN	NaN	NaN
2	1.569570e+09	1.569570e+09	NaN	NaN	NaN	NaN	NaN	NaN
3	2.011170e+08	2.838220e+08	9.820550e+09	9.820550e+09	2.537920e+09	2.537920e+09	2.537920e+09	2.537920e+09
4	8.096260e+09	5.768910e+09	9.426560e+09	2.419510e+09	1.709950e+09	7.773370e+09	NaN	NaN
...	...	...	...	...	...	...	...	...
99998	3.274030e+09	4.636070e+09	2.358780e+09	1.424300e+09	8.499400e+09	6.787620e+09	3.492230e+09	5.445110e+09
99999	3.452250e+09	7.297280e+09	2.779020e+09	5.470140e+09	1.993770e+09	9.036230e+09	3.603630e+09	9.406110e+09
100000	4.779810e+09	3.403120e+09	7.005220e+09	4.406870e+08	8.474990e+08	3.443710e+09	9.124730e+09	3.526410e+09
100001	9.113130e+09	5.509810e+09	1.911070e+09	9.238870e+09	7.849670e+09	8.127690e+09	5.683160e+09	2.163460e+09
100002	4.398330e+09	4.649190e+09	5.470440e+09	7.729120e+09	2.971280e+09	9.949030e+08	5.259870e+09	2.405770e+09

100003 rows × 8 columns

图 9. 地址坐标数据-去除 i 列 (100003 节点)

	x1	y1	x2	y2	x3	y3	x4	y4
0	9.820550e+09	9.820550e+09	9.820550e+09	9.820550e+09	2.537920e+09	2.537920e+09	2.537920e+09	2.537920e+09
1	2.537920e+09	2.537920e+09	9.820550e+09	9.820550e+09	2.537920e+09	2.537920e+09	2.537920e+09	2.537920e+09
2	1.569570e+09	1.569570e+09	9.820550e+09	9.820550e+09	2.537920e+09	2.537920e+09	2.537920e+09	2.537920e+09
3	2.011170e+08	2.838220e+08	9.820550e+09	9.820550e+09	2.537920e+09	2.537920e+09	2.537920e+09	2.537920e+09
4	8.096260e+09	5.768910e+09	9.426560e+09	2.419510e+09	1.709950e+09	7.773370e+09	2.381660e+09	3.054900e+09
...	...	...	...	...	...	...	...	...
99998	3.274030e+09	4.636070e+09	2.358780e+09	1.424300e+09	8.499400e+09	6.787620e+09	3.492230e+09	5.445110e+09
99999	3.452250e+09	7.297280e+09	2.779020e+09	5.470140e+09	1.993770e+09	9.036230e+09	3.603630e+09	9.406110e+09
100000	4.779810e+09	3.403120e+09	7.005220e+09	4.406870e+08	8.474990e+08	3.443710e+09	9.124730e+09	3.526410e+09
100001	9.113130e+09	5.509810e+09	1.911070e+09	9.238870e+09	7.849670e+09	8.127690e+09	5.683160e+09	2.163460e+09
100002	4.398330e+09	4.649190e+09	5.470440e+09	7.729120e+09	2.971280e+09	9.949030e+08	5.259870e+09	2.405770e+09

100003 rows × 8 columns

图 10. 地址坐标数据-填充 NAN 数据 (100003 节点)

从此数据处理完成，开始进行可视化。

### 3.2.2 网络可视化

此后将数据分为(x1, y1), (x2, y2),(x3, y3),(x4, y4)四个矩阵，其中第一组分别与后三组连线，实现网络可视化。

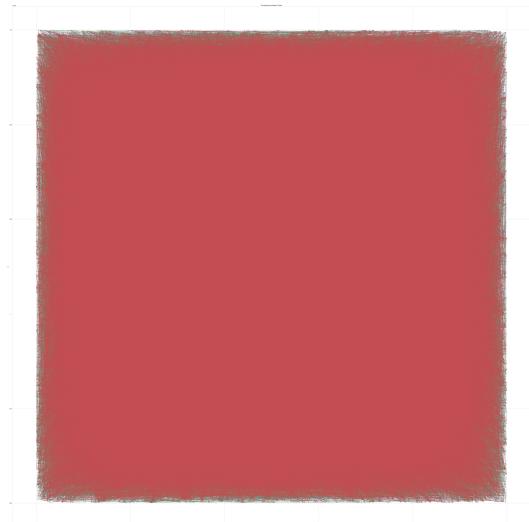


图 11. 网络可视化图（100003 节点）

可视化图由于节点过多，无法观察具体情况，因此选择减少节点数量重新绘制。

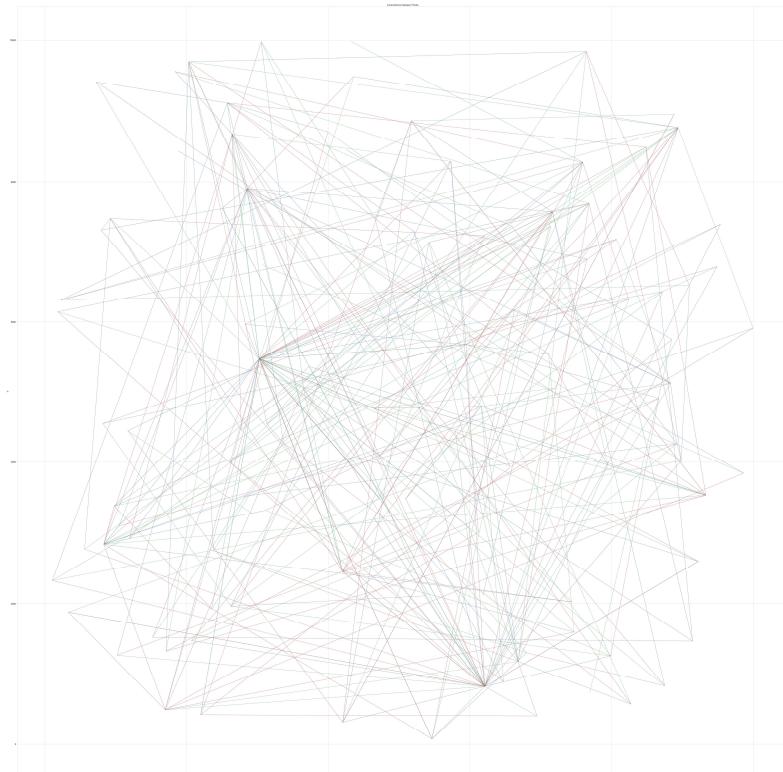


图 12. 网络可视化图（103 节点）

可见，103 节点时已经初步显示出了无尺度网络的特性。

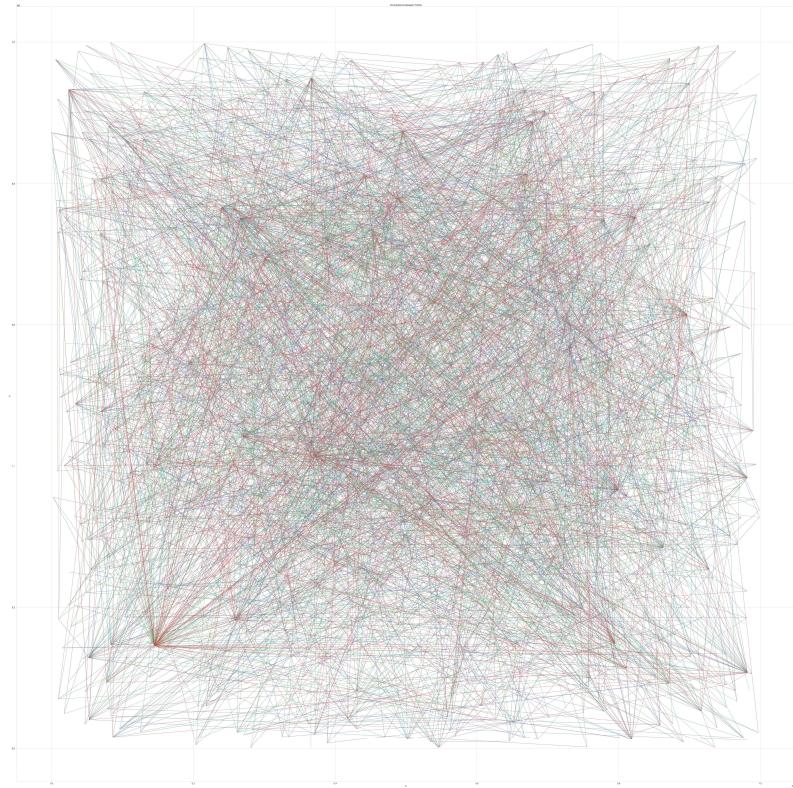


图 13. 网络可视化图（1003 节点）

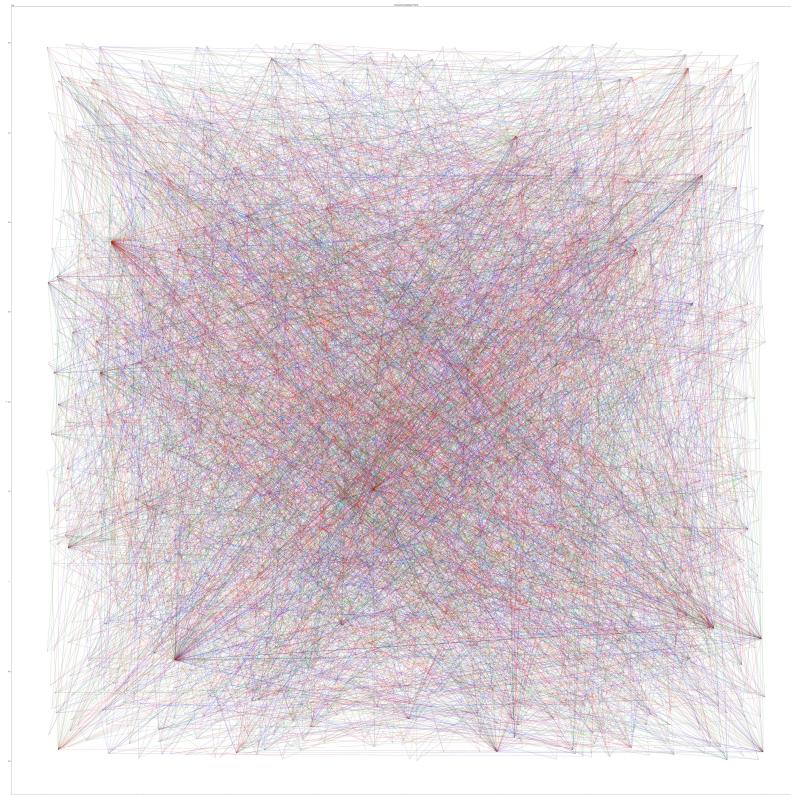


图 14. 网络可视化图（2003 节点）

可见，节点数目达到 1000 及以上的时候无尺度网络中的超级节点已经开始显现，此

时可以判断无尺度网络的建立已经实现。

## 4 总结

本次针对无尺度网络的研究，使用 C++ 构建无尺度网络模型，线性结构为基础结构，用于存储节点，图结构为拓扑结构用于模拟网络生成，成功实现了无尺度网络的构建。通过存储数据进行数据分析，验证得到无尺度网络度分布基本符合幂律分布，其无尺度特性得到验证。可视化网络的形式也成功验证了无尺度网络的特性。总体来说，本次针对无尺度网络的研究复现基本成功。

无尺度网络的特性可以进一步地解释社会关系、自然与人之间的关系，并且可以部分体现出人脑的思考方式，从而为类脑科学和人工智能中神经网络的建立提供参考。实际上，现在很多神经网络的特点都对无尺度网络的特性有所借鉴。本次实验的复现部分证实了无尺度网络的理论真实性与实用性，后续会进一步深入研究用以验证该网络本身更多的特性。

## 附录

C++建模代码（核心功能讲解在正文中）

```
1. #include <iostream>
2. #include <vector>
3. #include <ctime>
4. #include <chrono>
5. #include <fstream>
6. #include <cmath>
7. using namespace std;
8.
9. static unsigned long long range = 0;
10. static int edge = 0;
11. static unsigned long long _cnt_ = 0;
12. static unsigned long long k_sum = 0;
13.
14. double RandDouble();
15. double RandDouble(unsigned long long range);
16. int RandInt(int num);
17. void ShowInfo(int index, double rand_num, double prob_link, int cnt);
```

```

18.
19. class Node {
20.     private:
21.         int k;
22.         vector<vector<double>> edges;
23.         vector<double> position;
24.     public:
25.         Node(): k(0) {
26.             position.resize(2, RandDouble(range));
27.         }
28.         Node(int n): k(0) {
29.             position.resize(2);
30.         }
31.         friend void Link(std::vector<Node> &sfn, Node &p);
32.         friend void GetPosition(Node &p);
33.         friend void LinkEdges(std::vector<Node> &sfn, const int &index, Node &p);
34.         friend bool IsAvai(Node x, Node p);
35.         friend void CalculateDistribution(std::vector<Node> sfn, std::vector<double> &dis);
36.         friend void SaveDistribution(std::vector<Node> sfn, const std::string &filename);
37.         friend void SavePosition(std::vector<Node> sfn, const std::string &filename);
38.     };
39.
40. // 友元函数区
41. void GetPosition(Node &p) {
42.     p.position[0] = RandDouble(range);
43.     p.position[1] = RandDouble(range);
44. }
45. void LinkEdges(vector<Node> &sfn, const int &index, Node &p) {
46.     p.edges.push_back(sfn[index].position);
47.     p.k++;
48.     sfn[index].k++;
49.     int sum = 0;
50.     for(int i = 0; i < sfn.size(); i++) {
51.         sum += sfn[i].k;
52.     } k_sum = sum;
53. }
54. bool IsAvai(Node x, Node p) {
55.     for(int i = 0; i < p.edges.size(); i++) {
56.         if(x.position[0] == p.edges[i][0] && x.position[1] == p.edges[i][1]) {
57.             return false;
58.         } else continue;
59.     } return true;
60. }

```

```

61. void Link(vector<Node> &sfn, Node &p) {
62.     int cnt = 0; // 记录成功连接的次数
63.    GetPosition(p);
64.     while(cnt < edge) {
65.         _cnt_++;
66.         int index = RandInt(sfn.size()); // 随机生成 0~sfn.size() - 1 之间整数选取已有节点
67.         double rand_num = RandDouble(); // 随机生成 0~1 之间浮点数判断是否连接
68.         double prob_link;
69.         if(k_sum > 4) {
70.             prob_link = (double) sfn[index].k / k_sum;
71.         } else {
72.             if(k_sum == 0) {
73.                 prob_link = 1.0;
74.             } else prob_link = 1 - 1.0 / k_sum;
75.         }
76.         // ShowInfo(index, rand_num, prob_link, cnt);
77.         if(prob_link >= rand_num) {
78.             if(cnt == 0) {
79.                 LinkEdges(sfn, index, p);
80.             } else {
81.                 bool tmp = IsAvai(sfn[cnt], p);
82.                 if(tmp) LinkEdges(sfn, index, p);
83.             }
84.             cnt++;
85.         }
86.     }
87.     sfn.push_back(p);
88. }
89.
90. // 普通函数区
91. void Init(vector<Node> &sfn, int m0) {
92.     for(int i = 0; i < m0; i++) {
93.         Node p;
94.         sfn.push_back(p);
95.     }
96. }
97. void ShowInfo(int index, double rand_num, double prob_link, int cnt) {
98.     cout << "index: " << index << endl
99.     << "rand_num: " << rand_num << endl
100.    << "prob_link: " << prob_link << endl
101.    << "cnt: " << cnt << endl
102.    << "k_sum: " << k_sum << endl << endl;
103. }

```

```

104. template <class T>
105. void ShowCPUTimes(T &start) {
106.     auto end = std::chrono::system_clock::now(); // 记录结束时间
107.     std::chrono::duration<double> elapsed_seconds = end - start; // 计算花费的时间
108.     std::cout << "Elapsed time: " << elapsed_seconds.count() << "s\t ";
109.     std::cout << "Calculation times: " << _cnt_ << " times" << endl << endl;
110. }
111.
112. // 随机数生成函数
113. double RandDouble() { // 生成 0~1 之间随机浮点数，判断是否与查找到的结点连接
114.     double random_double = static_cast<double>(rand()) / static_cast<double>(RAND_MAX);
115.     return random_double;
116. }
117. double RandDouble(unsigned long long range) {
118.     double random_double = static_cast<double>(rand()) / RAND_MAX * range;
119.     return random_double;
120. }
121. int RandInt(int num) { // 生成所有结点数(num)之内的随机整型数据，用以随机查找结点判断是否连接
122.     int random_int = static_cast<int>(rand()) % num;
123.     return random_int;
124. }
125.
126. // 数据保存函数（属于友元函数）
127. void CalculateDistribution(vector<Node> sfn, vector<double> &dis) {
128.     int k_max = sfn[0].k;
129.     for(int i = 0; i < sfn.size(); i++) {
130.         if(sfn[i].k > k_max) k_max = sfn[i].k;
131.     }
132.     dis.resize(k_max + 1, 0);
133.     for(int i = 0; i < sfn.size(); i++) {
134.         dis[sfn[i].k]++;
135.     }
136. }
137. void SaveDistribution(vector<double> sfn, const string &filename) {
138.     ofstream ofile(filename);
139.     if(!ofile.is_open()) {
140.         cout << "FAIL TO OPEN FILE: " << filename << endl;
141.         return;
142.     } else {
143.         for(int i = 0; i < sfn.size(); i++) {
144.             ofile << i << "," << sfn[i] << endl;
145.         } cout << "DISTRIBUTION DATA SAVED IN FILE: " << filename << endl;
146.     }

```

```

147. }
148. void SavePosition(vector<Node> sfn, const string &filename) {
149.     ofstream ofile(filename);
150.     if(!ofile.is_open()) {
151.         cout << "FAIL TO OPEN FILE: " << filename << endl;
152.         return;
153.     } else {
154.         for(int i = 0; i < sfn.size(); i++) {
155.             ofile << sfn[i].position[0] << "," << sfn[i].position[1] << ",";
156.             if(sfn[i].edges.size() < edge) {
157.                 for(int j = 0; j < sfn[i].edges.size(); j++) {
158.                     ofile << RandDouble(range) << "," << RandDouble(range) << ",";
159.                 } ofile << endl;
160.             } else {
161.                 for(int j = 0; j < sfn[i].edges.size(); j++) {
162.                     ofile << sfn[i].edges[j][0] << "," << sfn[i].edges[j][1] << ",";
163.                 } ofile << endl;
164.             }
165.         }
166.         cout << "POSITION DATA SAVED IN FILE: " << filename << endl;
167.     }
168. }
169.
170. int main() {
171.     // ===== 网络生成 ===== //
172.     auto start = std::chrono::system_clock::now(); // 记录开始时间
173.     srand(static_cast<unsigned int>(time(nullptr))); // 使用当前时间作为随机数生成器的种子
174.     int m, m0, t;
175.     cout << "Input m, m0, t: " << endl;
176.     cin >> m >> m0 >> t;
177.     range = pow(t, 2); edge = m;
178.     vector<Node> sfn;
179.     // 初始化无尺度网络, 输入的 m0 代表初始节点数
180.     Init(sfn, m0);
181.     int count = 0;
182.     for(int i = 0, j = 0; i < t; i++, j++) {
183.         Node p(m);
184.         Link(sfn, p);
185.         if(j == t / 100) {
186.             count++;
187.             cout << "Already finished: " << count << " / 100\t";
188.             j = 0;
189.             ShowCPUTimes(start);

```

```

190.         }
191.     }
192.     cout << "Already finished: " << count + 1 << " / 100" << endl;
193.     ShowCPUTimes(start);
194.     cout << "<<<<<----- FINISHED ----->>>>>" <<
endl;
195.
196. // ===== 度分布计算及其存储 ===== //
197. vector<double> dis;
198. CalculateDistribution(sfn, dis);
199. SaveDistribution(dis, "kDistribution_data.csv");
200.
201. // ===== 节点地址指向存储 ===== //
202. SavePosition(sfn, "locations.csv");
203. }

```

### Python 数据分析代码 (Jupyter 中分单元格)

```

1. import pandas as pd
2. import numpy as np
3. import matplotlib.pyplot as plt

1. df = pd.read_csv("kDistribution_data.csv", names = ['K', 'P(k)'])
2. df

1. df = df[df['P(k)'] != 0]
2. df = df[df['K'] != 0]
3. df = np.log(df)

1. df

1. # 绘制度分布图
2. plt.scatter(df['K'], df['P(k)'])
3.
4. # 设置坐标轴标签和标题
5. plt.xlabel('Degree (K)')
6. plt.ylabel('Number of Nodes')

1. df2 = pd.read_csv("locations.csv", header = None, names = ['x1', 'y1', 'x2', 'y2', 'x3', 'y3', 'x4',
'y4', 'i'])
2. df2

1. df2 = df2.drop(columns=['i'])
2. df2

1. df2.bfill(inplace=True)
2. df2

```

```
1. # 提取坐标数据
2. x1 = df2['x1']
3. y1 = df2['y1']
4.
5. x2 = df2['x2']
6. y2 = df2['y2']
7.
8. x3 = df2['x3']
9. y3 = df2['y3']
10.
11. x4 = df2['x4']
12. y4 = df2['y4']
13. # 绘制图形
14. plt.figure(figsize=(100, 100), dpi = 70) # 设置图形的尺寸和分辨率
15. plt.plot([x1, x2], [y1, y2], 'b-', alpha = 0.7)
16. plt.plot([x1, x3], [y1, y3], 'g-', alpha = 0.7)
17. plt.plot([x1, x4], [y1, y4], 'r-', alpha = 0.7)
18. # 设置图形标题和标签
19. plt.title('Connections between Points')
20. plt.xlabel('X')
21. plt.ylabel('Y')
22. # 保存图形到文件
23. plt.savefig('plot.png', dpi=300) # 指定保存图像的分辨率为 300dpi
24. # 显示图形
25. plt.show()
```

## 参考文献

- [1] Barabási A L, Albert R, Jeong H. Mean-field theory for scale-free random networks[J]. *Physica A: Statistical Mechanics and its Applications*, 1999, 272(1-2): 173-187.
- [2] Barabási A L, Bonabeau E. Scale-free networks[J]. *Scientific american*, 2003, 288(5): 60-69.
- [3] Watts D J, Strogatz S H. Collective dynamics of ‘small-world’networks[J]. *nature*, 1998, 393(6684): 440-442.
- [4] McKinney W. Python for data analysis: Data wrangling with Pandas, NumPy, and IPython[M]. " O'Reilly Media, Inc.", 2012.
- [5] Meyers S. Effective C++: 55 specific ways to improve your programs and designs[M]. Pearson Education, 2005.
- [6] Stroustrup B. The C++ programming language[M]. Pearson Education, 2013.