廣東工業大學

QG 中期考核详细报告书

题　　目	中期考核
学　　院	计算机学院
专　　业	计算机
年级班别	大一(14)班
学　　号	3123004720
学生姓名	余学辉

2024 年 4 月 3 日

摘要

我们研究了 Krause 提出的意见动态模型：每个代理都有一个用实数表示的意见，并通过与其自身相差小于 1 的所有代理意见的平均值来更新其意见。我们给出了一个新的证明，证明了代理会聚合成簇，同一簇内的所有代理持有相同的观点。然后，我们引入了一种特定的平衡稳定性概念，并在稳定平衡状态下提供了簇间距离的下限。为了更好地理解当代理数量很大时系统的行为，我们还引入并研究了一个涉及连续代理的变体，在某些温和的假设下，得到了部分收敛结果和簇间距离的下限。[P][SEP]索引术语——共识，分散控制，多代理系统，意见动态

-------On Krause’s Multi-Agent Consensus Model With State-Dependent Connectivity

目录

一、 文献阅读

首先是对第一份文献一(李扬)开始阅读，初步理解多智能体的概念，以及对多智能体聚集问题研究的现状有所闻，比如保持拓扑连通性的研究（其中有提到人造势能场方法、约束函数法等等）、智能体资源受限（其中有提到功耗最小化聚集策略、可视角度过大可能会导致智能体发散、对收敛速度优化和其他聚集协议优化等等）。

然后是对第二份文献 1(谢光强)阅读，但是上来就是一致性协议研究，很多没看懂，因此就随便略过。

接着就是对考核文献进行阅读了(要求对第二章进行重点阅读)，对这篇论文文献(On Krause's Multi-Agent Consensus Model With State-Dependent Connectivity)的阅读，通过摘要大致了解这篇论文的主要内容，然后结合语句以及后面的两张图，对公式一进行理解，当时还没想进行实现(以为需要有什么数据集)，接着对收敛性证明进行大致的了解，后面就是实验观察，通过模拟出不同 L 得到簇，然后再用 L/2 进行相减，此时读到这，我想要读完第二章才开始实现，所以就往下读，接着就是本文对图 3 和图 4 进行分析研究，在此之后，就是本文对公式一进行拓展了，对每个数据前的系数赋予不同的权重，然后对聚类稳定性的讨论，最后就是第三章，由于是对连续智能体模型的研究，而且我还不是很理解这个模型，所以就草草略看了。

最后就是第二篇文献了，第二篇文献我是考核的前两天开始阅读的，所以我觉得我理解的应该不是很深，我只知道它的算法，但是

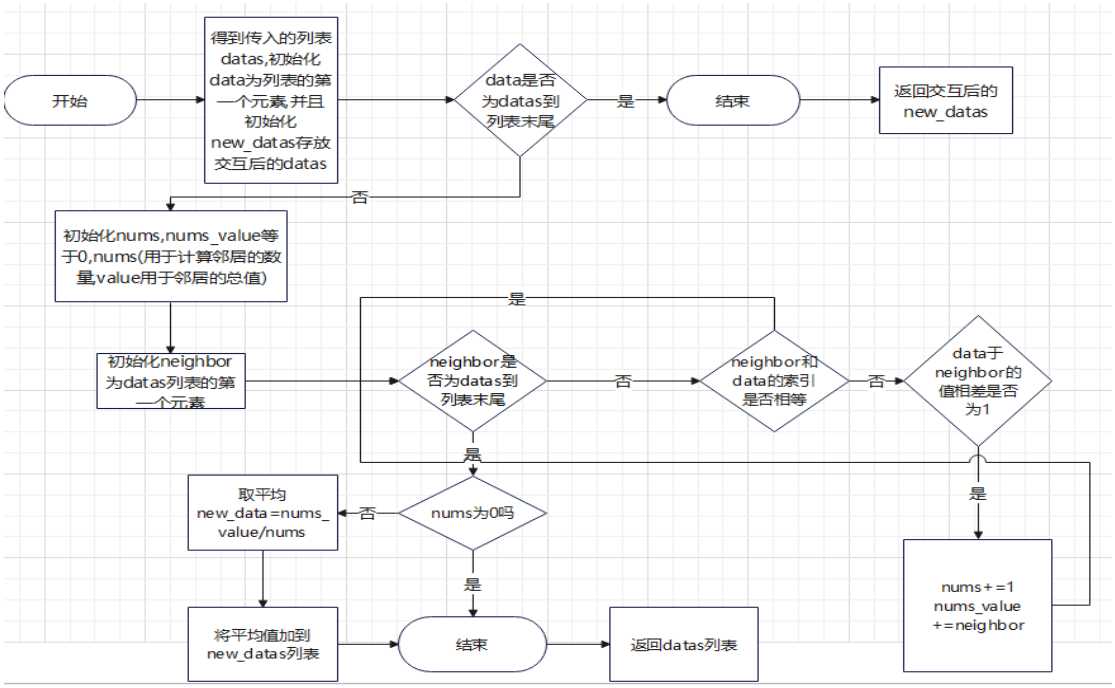
不知道它这个算法的稳定性什么的，所以只是简单模仿模仿...

其他文献我只是看了一下（可能是有点抽象就没看了），比如第四篇文献我看了前两章，还想复现一下的，但是也没实现

二、图像仿真

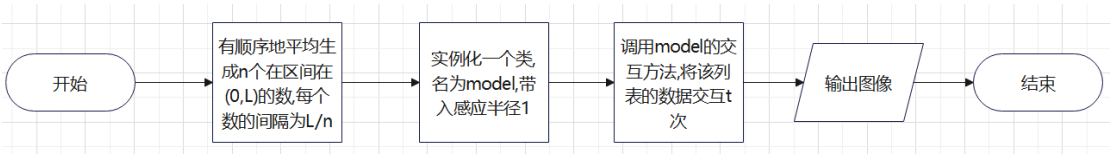
在作图之前需要进行代码实现，所以就简单实现了，没有进行什么算法分析或者优化（纯粹是想到就敲）代码一如下：

```
new_datas = []
# 该循环是计算每个智能体与它相邻的数的平均值
for data in datas:
    nums = 0 # 首先对和它相差1的总值进行归零
    nums_value = 0.0 # 数量也进行归零
    for neighbor in datas:
        if neighbor == data: # 如果遍历到它本身就跳过
            continue
        distance = fabs(data - neighbor)
        # 判断该智能体是否和这个智能体相差1，如果不是就跳过当前循环
        if distance < self.radius:
            nums += 1
            nums_value += neighbor
    if nums != 0: # 此时如过num不为0则说明还没完成形成聚类
        new_data = nums_value / nums
    # 此时如果形成聚类则返回原来的列表
    else:
        return datas
    new_datas.append(new_data) # 更新一次交互后的数据
return new_datas
```



总的来说就是，把时间点 t 的意见进行遍历，遍历期间进行对和它相差 1 的意见总和进行平均，然后记录到下个时间点 t+1 的意见上

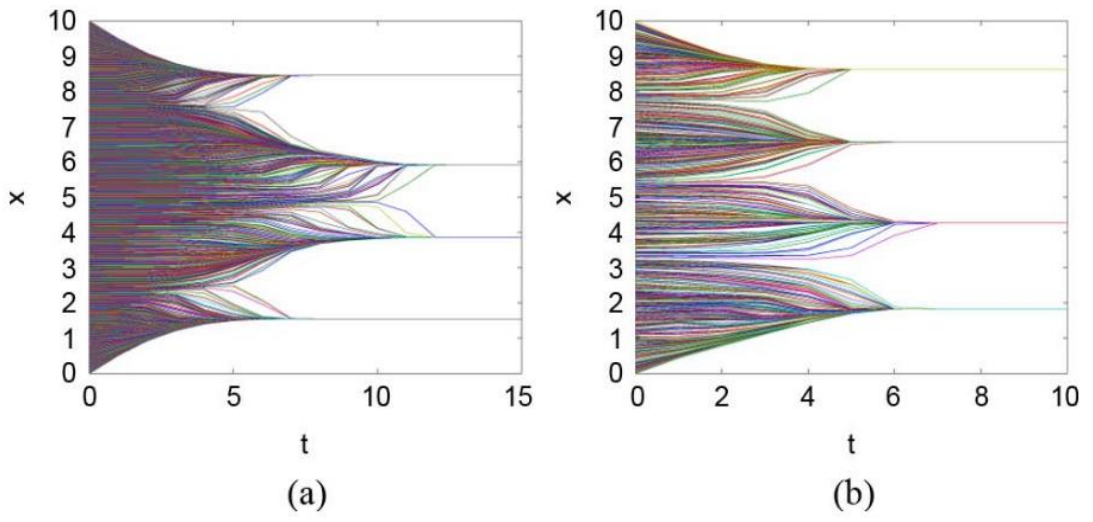
```
for time in range(10):
    new_list.append(custom_list)
    custom_list = model.interact(custom_list)

plt.plot( *args: new_list,linewidth=0.3)
```



这图就是将上述算法循环(15 次)10 次以得到聚类(图一)，容易看出 time 表示的就是横坐标 t

(1)图一(fig1)：

原图：



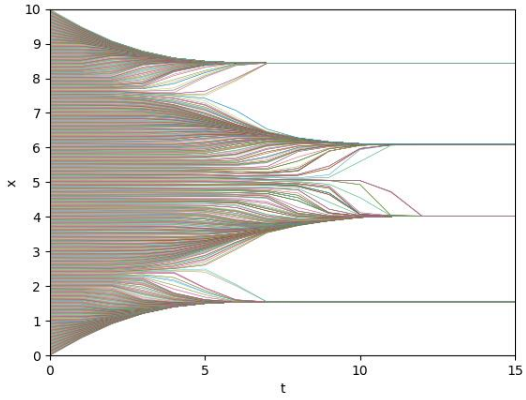
分析：由于第二张图带有随机性，所以和原图是有些出入的，而我对这两个数据是这样选的：

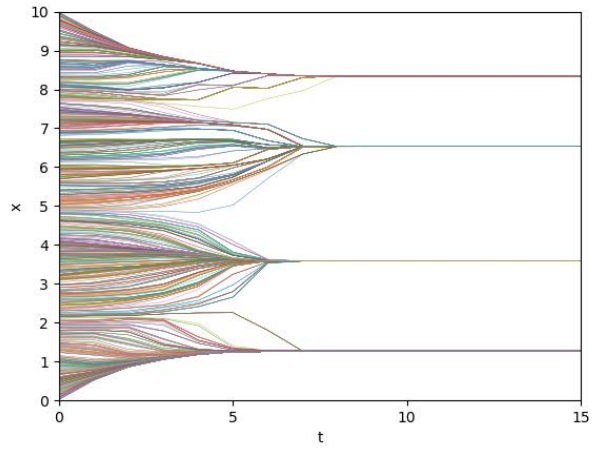
```
L = 5
n = 200
number_of_elements = n
step = L/number_of_elements
start = 0.0
end = L
custom_list = [round(start + i * step, 2) for i in range(number_of_elements)]
# custom_list = np.random.uniform(0, 10, 1000)
```

第一个 custom_list1 就是有顺序地生成 1000 个 0 到 10 的数

第二个 custom_list2 就是随机顺序生成 1000 个 0 到 10 的数(在仿真过程中会得到其他不同的图，而且像图 1 那样的图生成概率也不高，我测试了几次才测试出来)

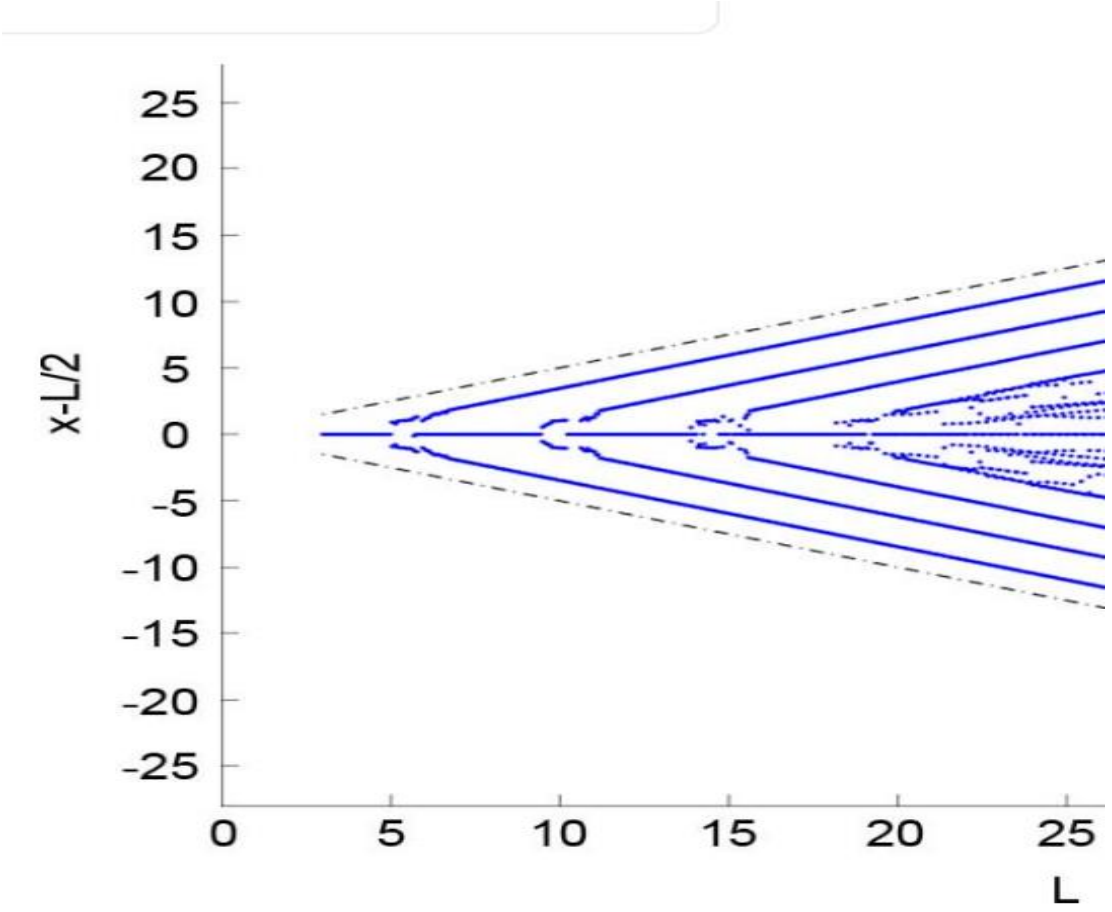
仿真图：



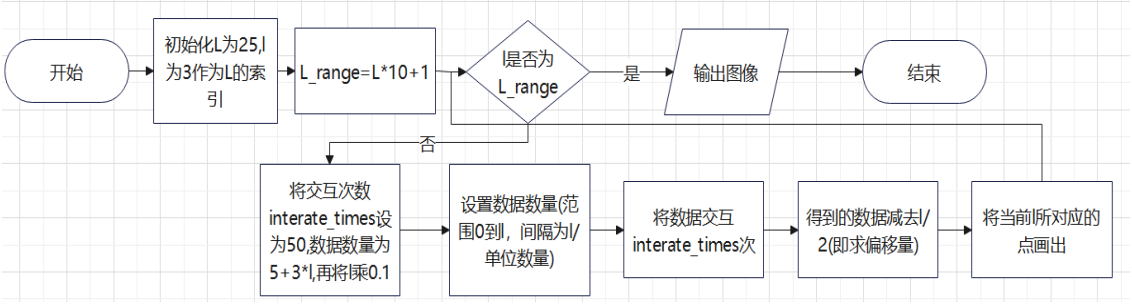


(2)图二(fig2):

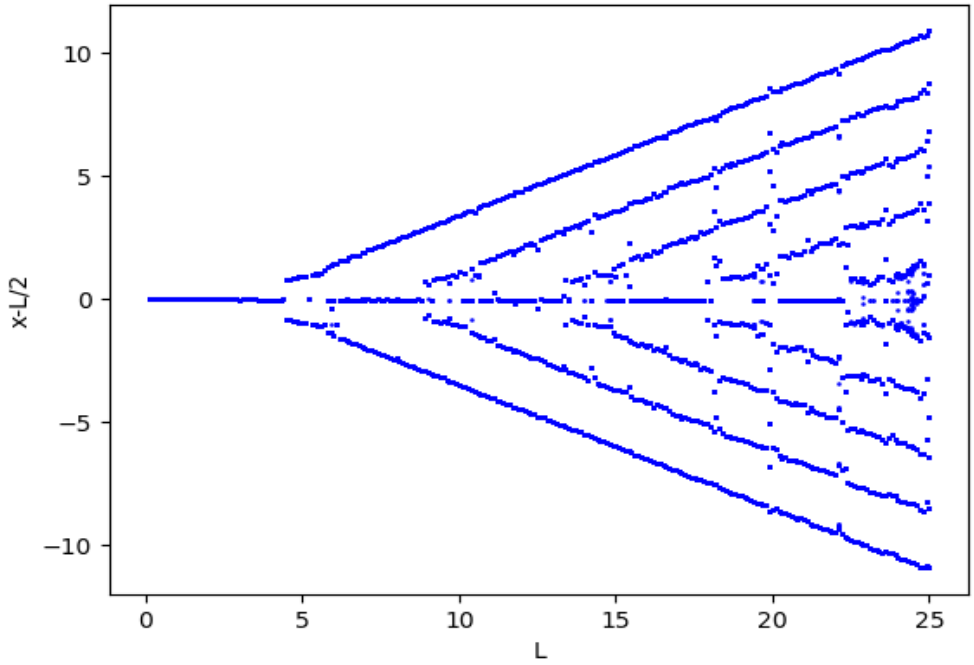
原图：

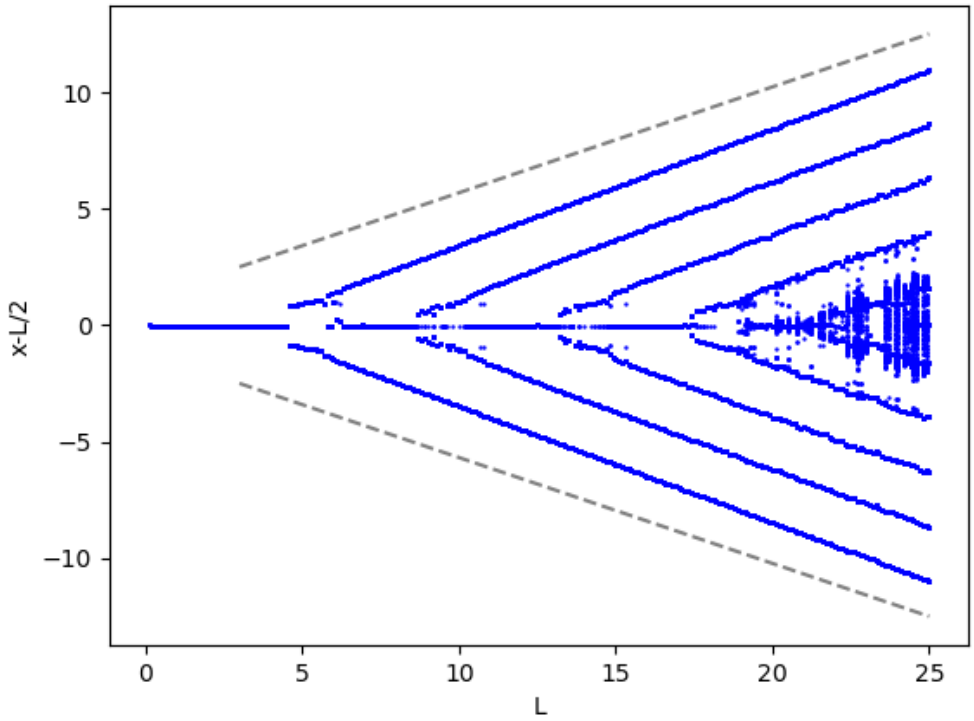


分析：易知，每个 L 所对应的 x-L/2 其实就是最终交互完成的聚集对中心的偏移，所以就有用得到的 custom_list 减去 L/2,然后用 L 从 1 循环到 25 即可，有多次实验得到一般交互次数在 30 左右就能形成全部聚类，这里保险一点就 50 次



仿真图：

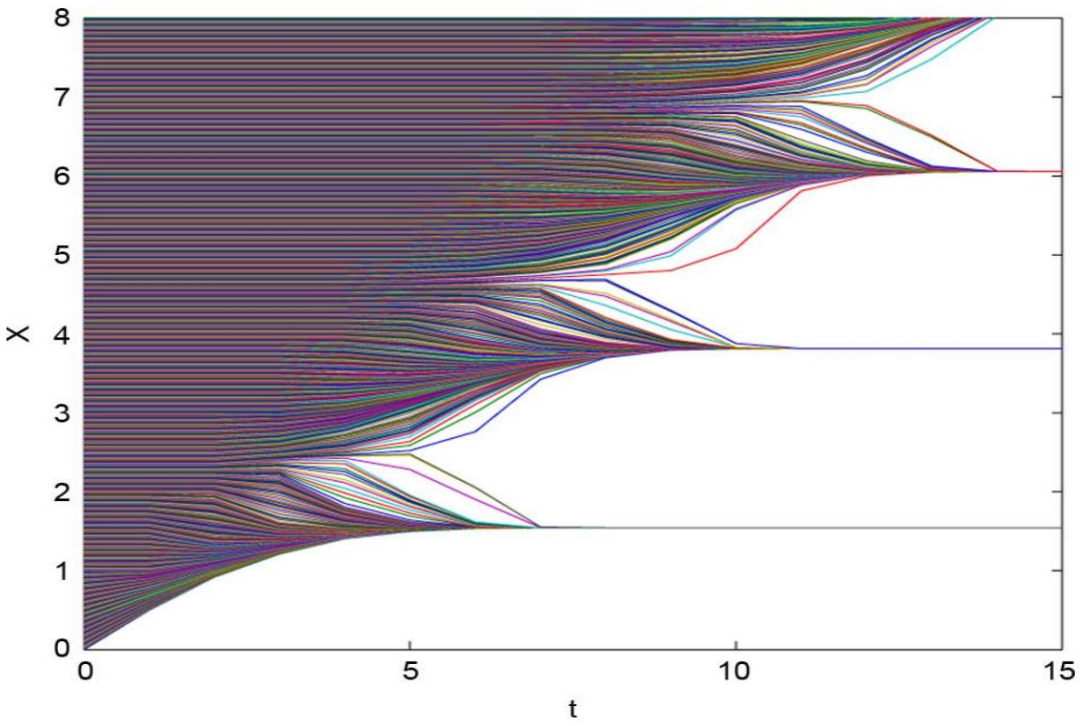




然后数据大小我也实验过多次，如果很大就会跑的很久，我就选了小的小一点，大的大一点(比如 L 为 4 就 20，L 为 25 就 1000)，但是最后的数据仿真的不好：

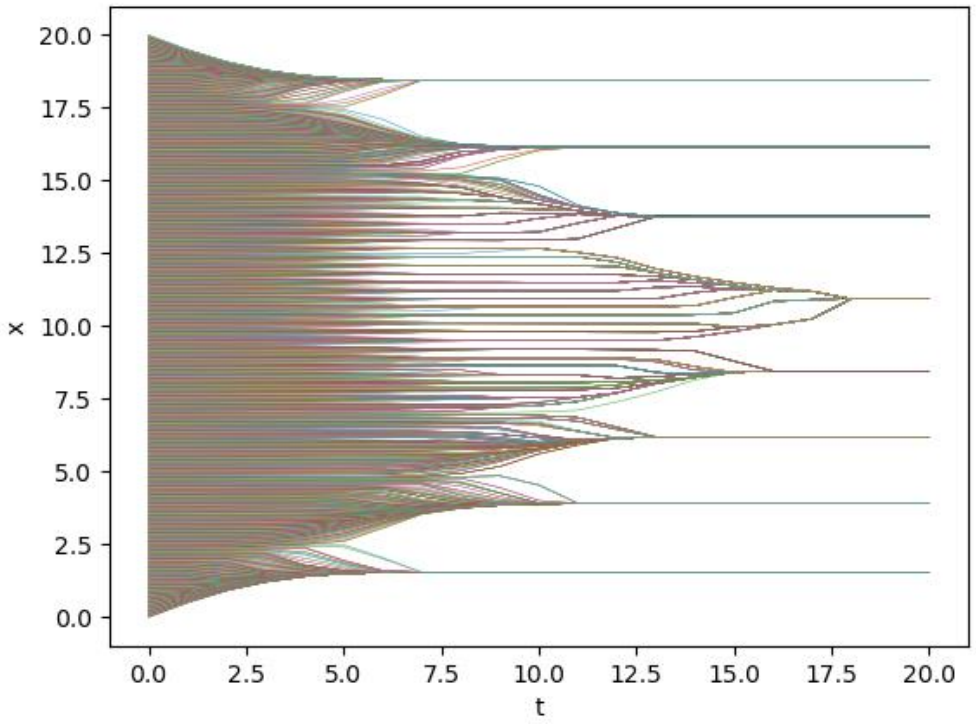
(3)图三(fig3):

原图：

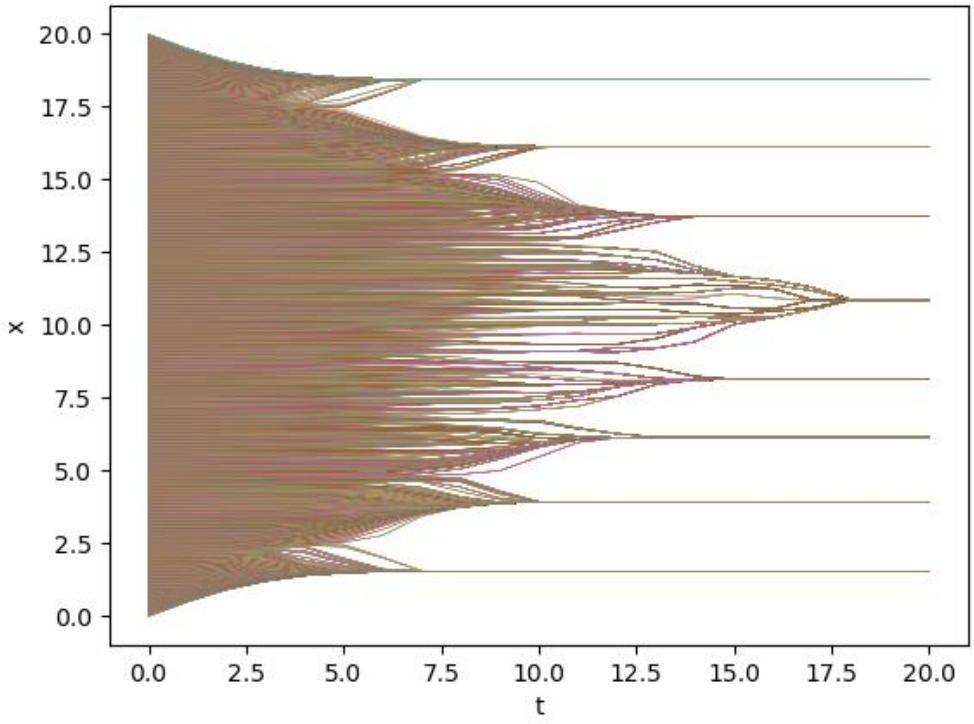


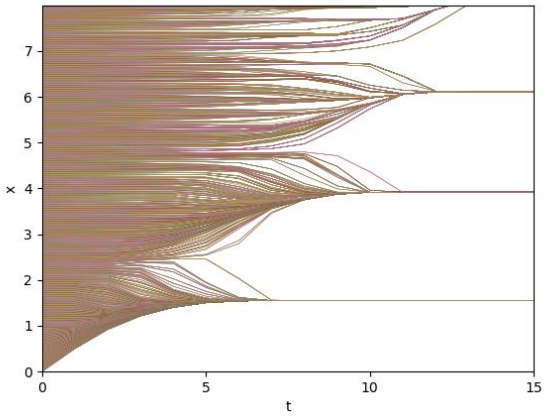
分析：可以看出来这张图应该是一张图的某部分，所以我就估计它的 L 区间长度以该是差不多九的两倍 18，再四舍五入就是 20，所以我选 L=20，然后 t 为 21

仿真图：



看得出来还是很多出入的，而且可能数据量不是很大，所以后面会有些空白，然后我选择再增加点数据，但也是有些出入的





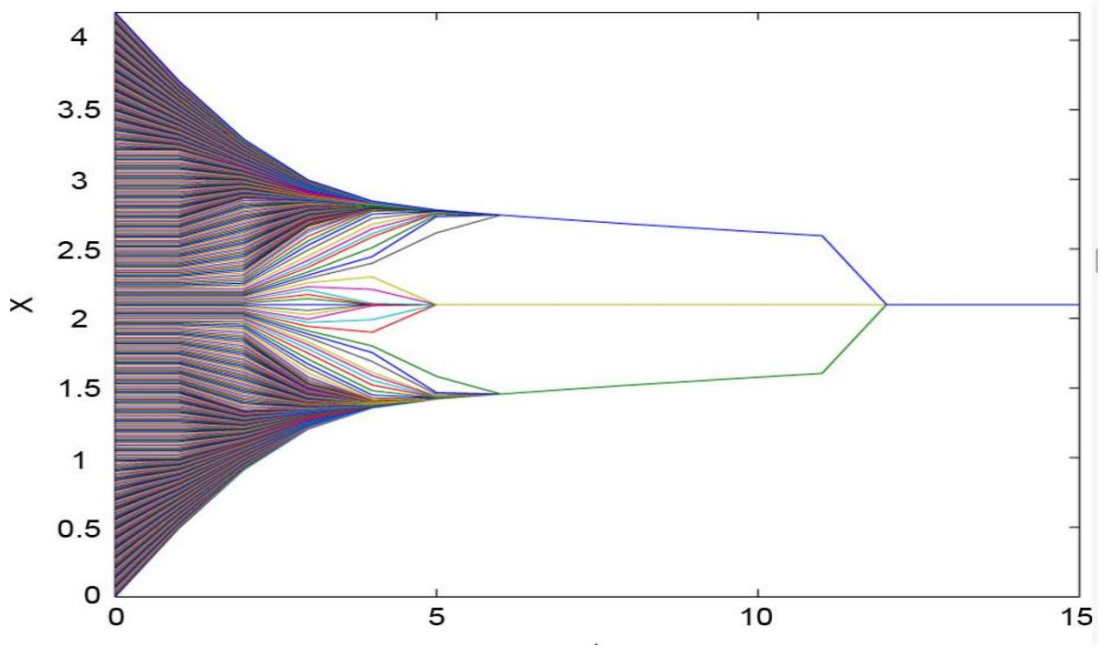
加入权重：对每个意见前的系数赋予权重，使得它们的权重不同，从而改变聚类的位置和收敛速度

代码实现：

```
if distance < self.radius:
    total_weighted_value += neighbor * weights[neighbor_index]  # 使用邻居的值乘以权重
    total_weight += weights[neighbor_index]  # 累加权重
```

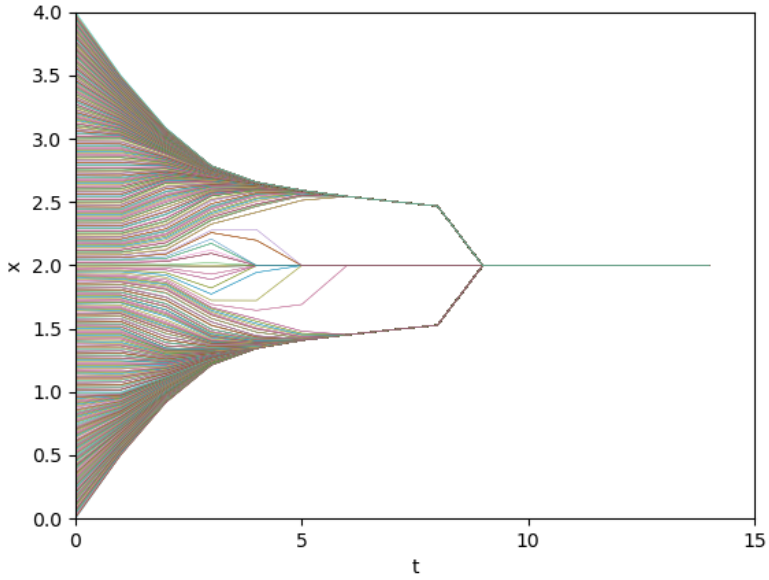
(4)图 4(fig4):

原图：



分析：可以看出 L 区间长度是 4，时间段为 15，数据应该是有顺序均匀的

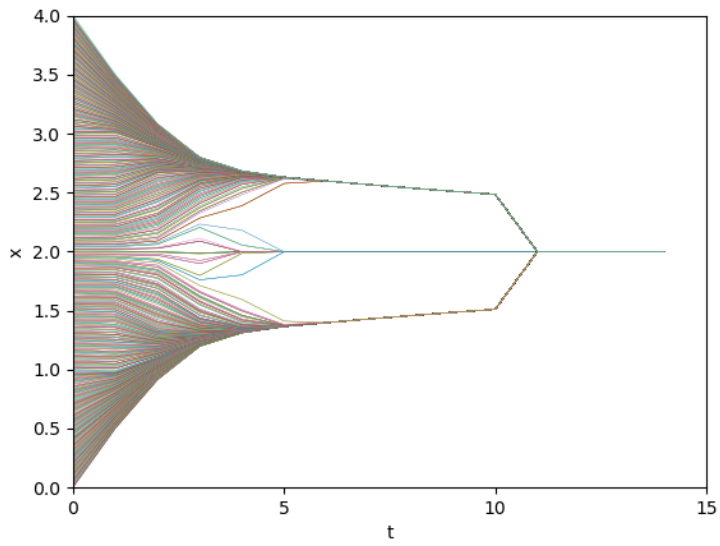
仿真图：



可以看出模拟地看起来差不多，但是仔细看，这里收敛到一个聚类所需时间差不多是九点多一点，但是论文中的图是十二多一点才收敛完成的，所以还需要改一下下，但是怎么实现呢，这个收敛速度它是不和测试的数量有关的(前提是在区间内平均分布，如果将其不平均分布，则会有很大的工作量)，所以我选择增加权重，这是为什么我先实现增加权重后实现图 4：

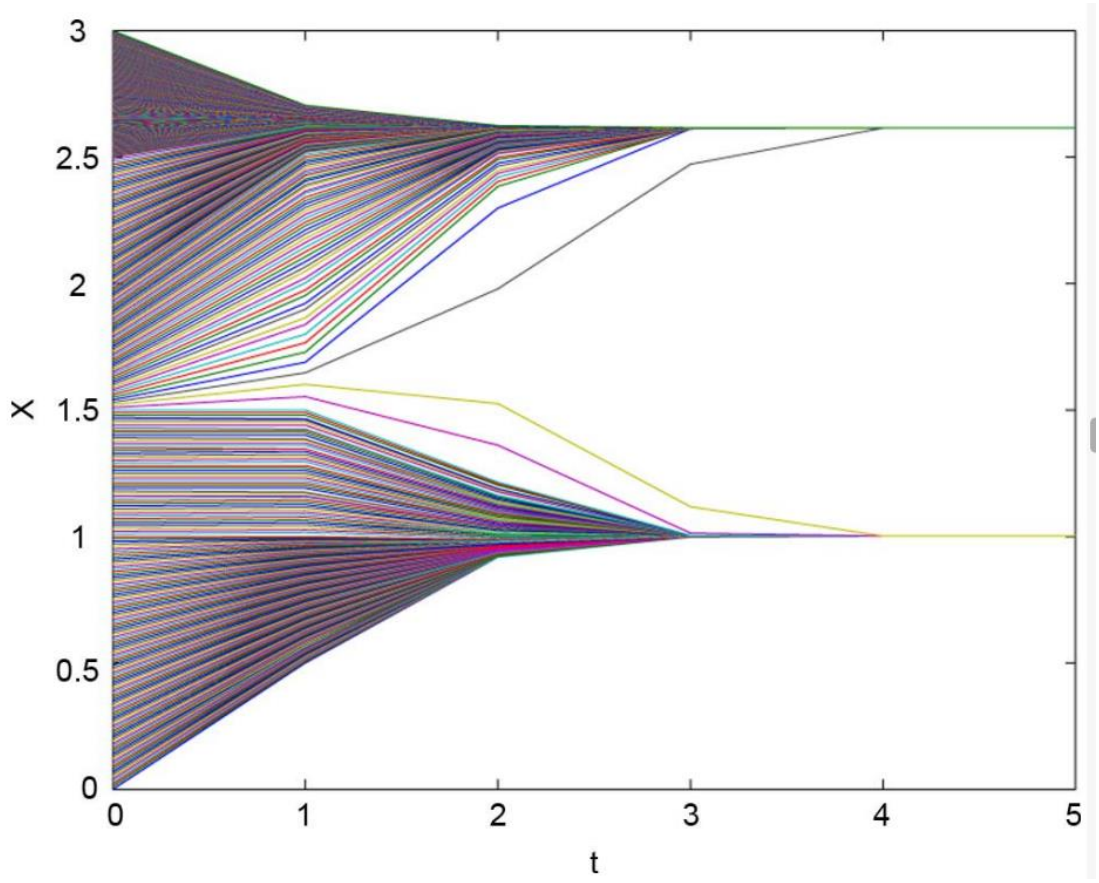
```
for i in range(n):
    if i > 450:
        nums = [50]
    elif i < 350:
        nums = [50]
    else:
        nums = [47]
    # nums = [1]
    weights += nums
```

增加权重后(将上下区间数据的权重改的稍微比中间区间高一点)，就有如下效果：



(5)图 5(fig5)：

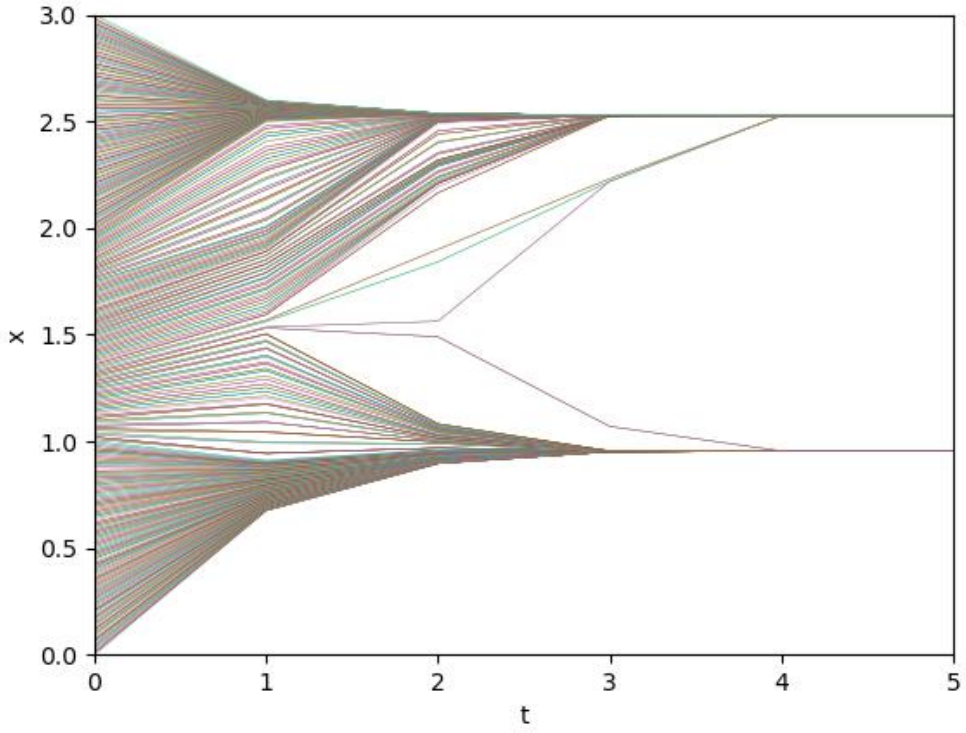
原图



分析：由图可知，最后聚集形成的意见在 2.5 和 1 左右，所以可以将区间(2,3)和区间(1,0.5)的权数加大一点：

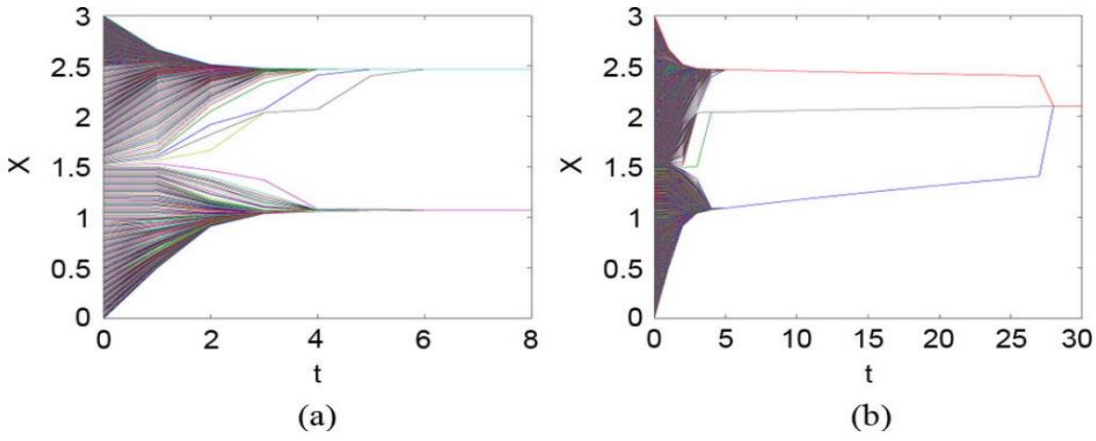
```
for i in range(n):
    if i > 550 :
        nums = [20]
    elif i> 400 and i < 550:
        nums = [8]
    elif i > 300 and i < 400:
        nums = [1]
    elif i>150 and i < 200:
        nums = [6]
    elif i >50 and i < 100:
        nums = [2]
    else:
        nums = [1]
    weights += nums
```

仿真图：



(6)图 6(fig6):

原图：

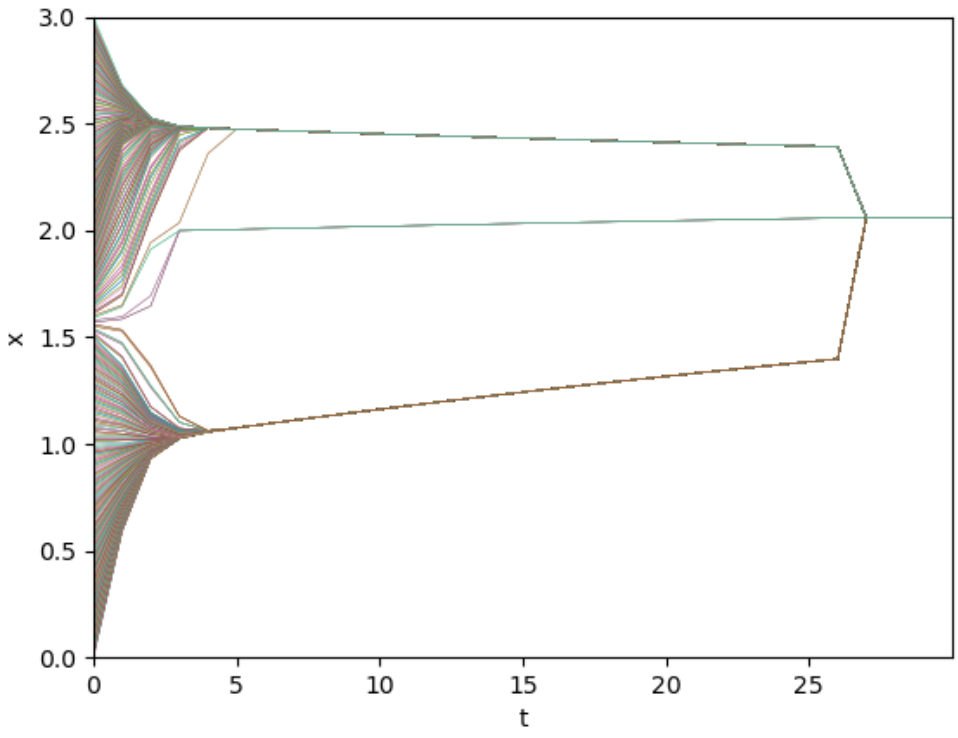


分析：图中的 a 和图 5 是差不多的，所以就不把它放上来了，图 b 则是有一个干扰项的意见，将原本不应该聚集在一起的聚类集合在了一

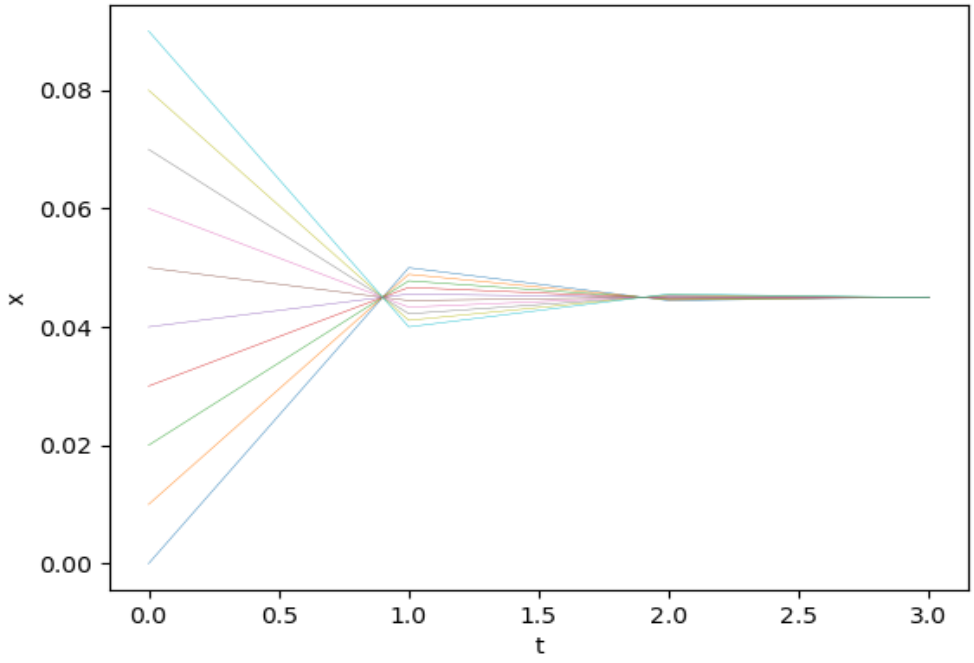
起，但是这干扰项需要有一定的权重才能将两个聚类聚集在一起，以下是我摸索出来的权重：

```
for i in range(n):
    if i > 500:
        nums = [925]
    elif i < 200 and i > 100:
        nums = [300]
    elif i > 300 and i < 400:
        nums = [200]
    else:
        nums = [142]
    weights += nums
```

仿真图：



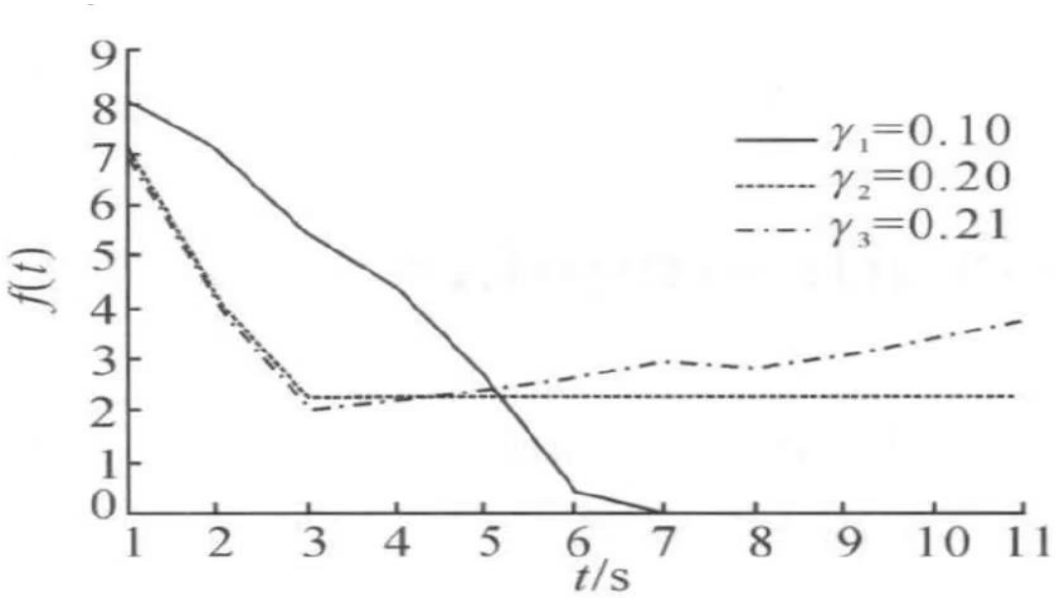
有趣的图：



这图是我想着 L 区间为 0.1 的话图会怎么样才搞来的

关于论文 2 中 fig2 的仿真：

原图：

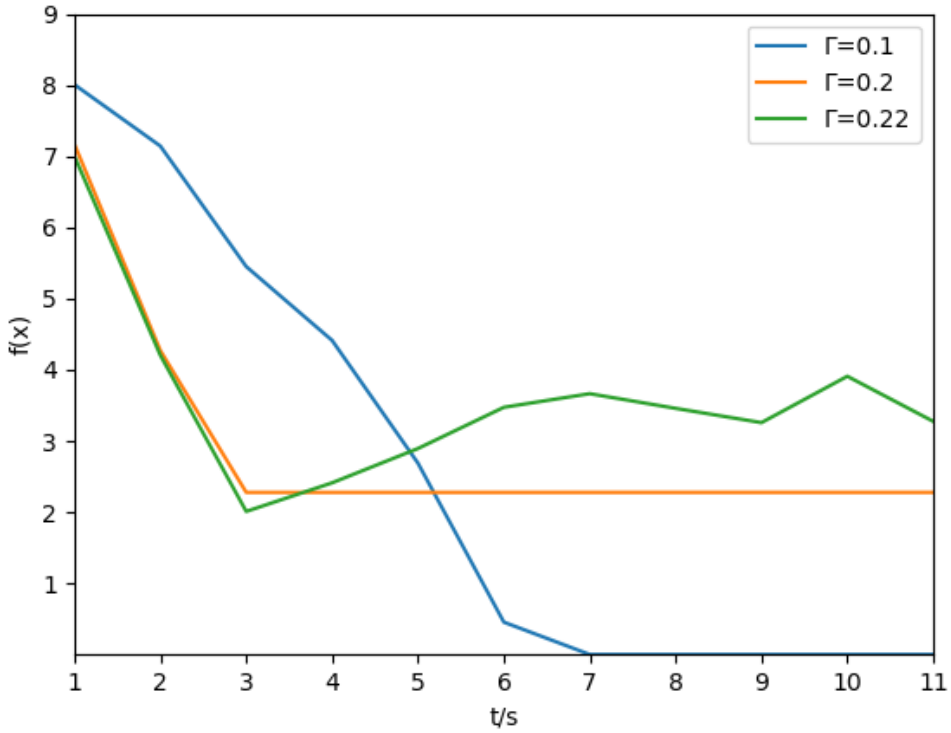


代码：

```
for gama in gamas:
    model = one.Interaction( radius: 1, gama=gama)
    f_list = []
    temp = coordinates
    for i in range(12):
        print(f"Γ={gama}")
        print(coordinates)
        sum_x = sum(coord[0] for coord in coordinates)
        sum_y = sum(coord[1] for coord in coordinates)

        # 计算平均值得到中心坐标
        center_x = sum_x / len(coordinates)
        center_y = sum_y / len(coordinates)
        c = np.array((center_x, center_y))
        print(c)
        fx = 0.0
        for coord in coordinates:
            fx += sqrt((coord[0] - c[0])**2 + (coord[1] - c[1])**2)
        coordinates = model.interact(coordinates)
        f_list.append(fx)
    coordinates = temp
    t = [i+1 for i in range(len(f_list)-1)]
    plt.plot( *args: t, f_list[1:], label=f'Γ={gama}')
```

```
def interact(self, datas) -> np.ndarray:
    new_datas = np.array((0.0, 0.0))
    for data in datas:
        nums_value = np.array((0.0, 0.0))
        for neighbor in datas:
            if np.array_equal(neighbor, data):  # 如果遍历到它本身就跳过
                continue
            distance = sqrt((data[0] - neighbor[0])**2 + (data[1] - neighbor[1])**2)
            # 判断该智能体是否和这个智能体相差1，如果不是就跳过当前循环
            if distance <= self.radius:
                nums_value += (neighbor.astype(float) - data.astype(float))
        new_data = nums_value * self.gama
        new_datas = np.vstack((new_datas, data + new_data))
        # new_datas = np.concatenate((new_datas, new_data + data), axis=0) # 更新一次交互后的数据
    new_datas = new_datas[1:, :]
    return new_datas
```

仿真图：



三、 总结与体会

工作总结：总的来说，我这次的工作就是阅读文献，其中把文献中的前两章重点阅读了，但是后面的章节草草略过，还有把前面两部分的文献也泛读了，然后剩下的时间都用在对文献中的图仿真（其中对于图 2 的复现所需要的数据量有些大，一时半会儿弄不出来，所以也是拖了一两天）然后撰写文档，整理代码，ppt

体会：第一次参加这样的考核，所以多少有些慌张，所以是想早点做，开始时我看到那么多参考文献，而且还有两篇是英文的，还有一篇有一百多页那么长，我就有点站不住脚，但是后来看了看考核任务是对图片的部分复现，开始时有些不适应，但我还是硬着头皮上了，

结果用了半天多点的时间看了看第五篇文献的前两张（主要时翻译麻烦），星期天晚上才把第一张图给仿真了（虽然不用仿真，但是我觉得是必要的），然后第二张图我是星期一早上把它差不多弄出来了，但是有点潦草，应该是数据量不够，到后面的 L 都是歪七扭八的，我觉得仿真的不够好，所以我就多次带入不同的数据量，交互次数，如果带入过大搞不好弄我一整天.....，但我数据小的话，仿真得又不好，所以也是一直让它跑，这几天我电脑的 cpu 都要干没了，然后在这期间我就仿真其他图了，除了图二我仿真也只用了半天时间，归根结底，实现的算法还是很好敲的，所以从星期二中午开始剩下的时间就是开始撰写文档，准备 ppt，但是有点担心的是，会考到其他文献的内容，因为其他的文献基本只是了解了以下，不过这次我也吸取了很多教训，每次做任务是都要看内容，不然会搞得自己手忙脚乱的，还有就是阅读英文文档的时候应该对整体有个了解，知道哪些重要，重要的话就要仔细体会阅读，而不太重要的就不要太仔细（我就是因为这个搞了半天的时间）。但是后来我把图仿真之后我就不知道我要干什么了，感觉仅仅只会仿真的话，应该很多人比我做的更多吧....

到了考核前两天，我有点闲，所以就有点坐不住脚，所以像看看第二篇文献，然后就试着仿真一下 fig2 的图，不过只知道算法怎么算，但是不知道后面得到稳定性什么的，所以就感觉有点像是为了仿真而仿真....

四、 参考文献

1. Vincent D. Blondel，On Krause’s Multi-Agent Consensus Model With State-Dependent Connectivity，Julien M. Hendrickx, and John N. Tsitsiklis, Fellow, IEEE

2.李扬，多智能体聚集问题研究综述，田家赫，谢光强 ，郭小全

3. 谢光强，基于切换拓扑的多智能体协作控制研究综述，阳 开，李 杨 ，徐 峰

4、[2]多主体汇聚问题离散算法的稳定性_郑军