一. 本周工作总结

1. 本周的主要工作分配在 Delaunay 三角网格自编程、论文阅读以及探究迭代 voronoi 算法的可行性，同时还复习了数据结构。

2. 首先是 Delaunay 三角网，承接上周，本周已经彻底完成实现了代码的编写，并基本实现了与 python 集成 Delaunay 函数完全相同的功能，下面由我来进行介绍：

   该代码编写的思路基于三角网格生长算法，即依次按照 x 坐标升序遍历所有点然后每个点单独生成三角形的过程。

   首先我们需要构建一个超级三角形，使得所有点都要包括在该三角形中。我的思路是先构造一个圆，其包含了所有点集，然后求这个圆的外接等边三角形。关于圆心的选取我是取得为$((x_max - x_min)/2 + x_min, (y_max - y_min)/2 + y_min)$，本来半径我是想取 x 轴方向的跨度与 y 轴方向跨度中的较大的一个，但是由于我的圆心选择问题，我怕存在偏心现象即如下图右上角点所示：

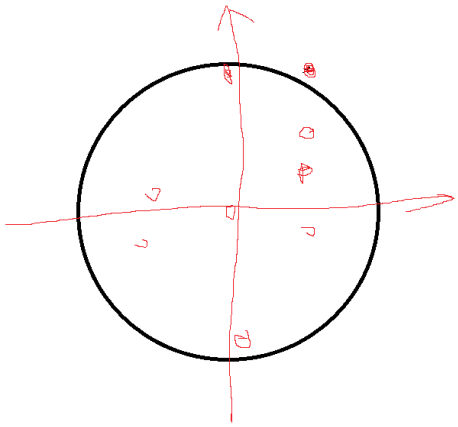


Fig1. 偏心现象

   所以我们还应该将半径取得更大一点，大到足以包含这些极端偏离分布的点，所以我最终取得半径为：

```
R = max(abs((x_max-x_min)/2), abs((y_max-y_min)/2)) + (x_max-x_min)
```

这样足以囊括所有情况。然后三角形的三个顶点坐标，我利用的是等边三角形五心合一的性质很方便的就算出来了。

接下来，我们需要考虑最复杂的一段，就是数据结构表示。我考虑的方面是，我们将所有待生成点按照 x 轴坐标顺序排序，用二维数组表示其坐标值，然后将超级三角形三点插入到数组的最前面，构成完整的数据数组。

对于三角形边的表示是最为复杂，因为涉及到 AB=BA，很容易表示重复。我突然间想到了 python 中的特殊数据类型——集合，因为在集合中即使我们输入{1,0}其最终也会按照{0,1}进行排序，方便了后续的去重。而对于整个边集来说，由于 set 和 turple 两种数据类型不容易对其进行 remove 和 insert。因此我采用了列表(list)的数据类型，最终复合构建了 list(set)模型方便后续进行操作。至于三角形表示，我也使用了 list(set)模型，其中每个 set 用其顶点在数据数组中的位置来表示。以上过程便是我们完成数据结构抽象的过程。

下面是基本算法的原理，首先我们从左到右遍历采样点，对于大的超级三角形，我们先将三点与第一个采样点连线，构成了三个小的三角形

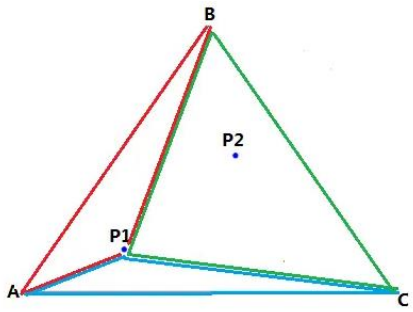


Fig2. 三角网络初生成

然后我们开始引入第二个采样点，我们的基本思路是判断该采样点与现有三

角形的位置关系，即在三角形外接圆外或者是三角形外接圆内。我们先将所有的三角形存储在一个名为：

```
temp_tri_table
```

的临时三角形列表。下面我们考虑第二个点与此时存在的所有三角形的外接圆的位置关系。

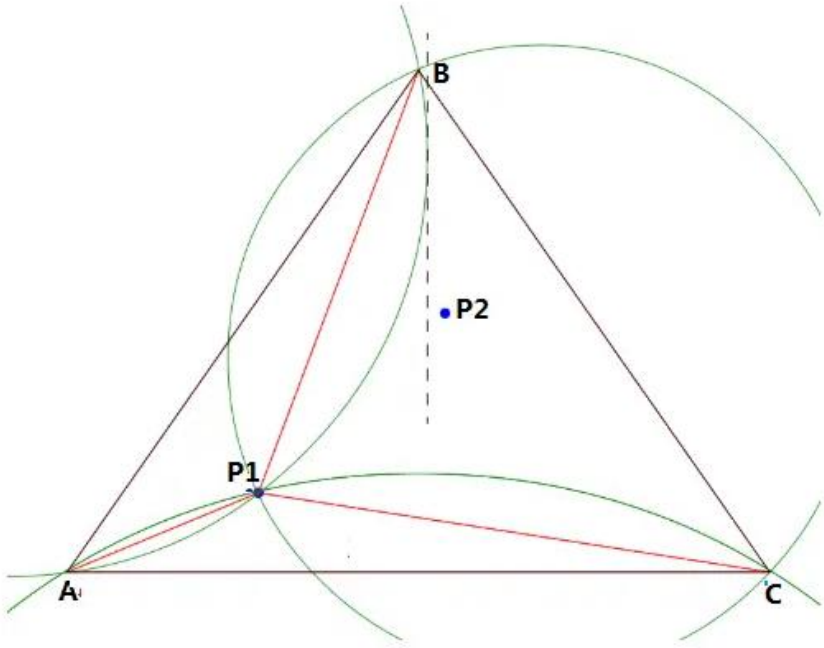


Fig3. 判断过程

如果在三角形外接圆内的话，则由 Delaunay 三角网的空圆特性，该三角形不符合要求得删去，因为我们是 for 循环遍历临时三角形列表，此时我们要将删去的三角形的索引存放在如下所示的索引列表

```
idx_table
```

同时该类型三角形我们需要进行拆分成边的操作，将边存放在下列边集：

```
temp_edge
```

实现代码我调用了 itertools 模块里的 combination 函数，实现过程如下：

```
# 生成所有包含两个元素的组合，并将每个组合转换成元组并加入新集合中
new_set = set()
for combo in combinations(temp_tri_table[i], 2):
    new_set.add(tuple(combo))
```

```
print(new_set)
new_list = [set(t) for t in new_set]
print(new_list)
temp_edge.extend(new_list)
```

由于 set 不太适合做变换，我们现将 set 转换为 turple，然后再转回 set。

分解为边后，我们需要将这些边与现在正在判断的三角形构成新的三角形，然后将这些新的三角形存储在 temp_tri_table 中，等待其与下一个点的关系判断。实现代码如下：

```
"""删除不符合要求的临时三角形"""
temp_tri_table_remove = [temp_tri_table[t] for t in idx_table]
set1 = set(frozenset(i) for i in temp_tri_table)
set2 = set(frozenset(i) for i in temp_tri_table_remove)
result = [set(i) for i in set1.difference(set2)]  #对两个集合求差集
temp_tri_table = result

"""将临时边集与新点重新构成三角形，并更新到临时三角形列表里"""
new_temp_tri = [{idx_point, *d} for d in temp_edge]
temp_tri_table.extend(new_temp_tri)
```

如果在三角形外接圆外部，此时会分为两种情况，如果在外接圆的右侧，则无论后面的点再怎么出现，一定都在该三角形外接圆的右侧，故此三角形一定是 Delaunay 三角形，后续的操作我们无须对其进行判断。我们构建了一个名为：

```
delaunay_tri_table
```

的表，将该类三角形存储在里面，并且记录位置与 idx_table，待后续删除，但要注意的是，此时我们不需要拆分该类三角形为边，不需将边存放在 temp_edge 中。

如果不在外接圆的右侧，则该三角形待定，不要对其进行下次的操作。等待下一个采样点的判断，重复上述过程。

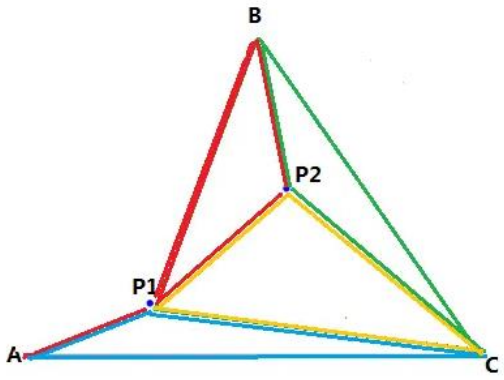
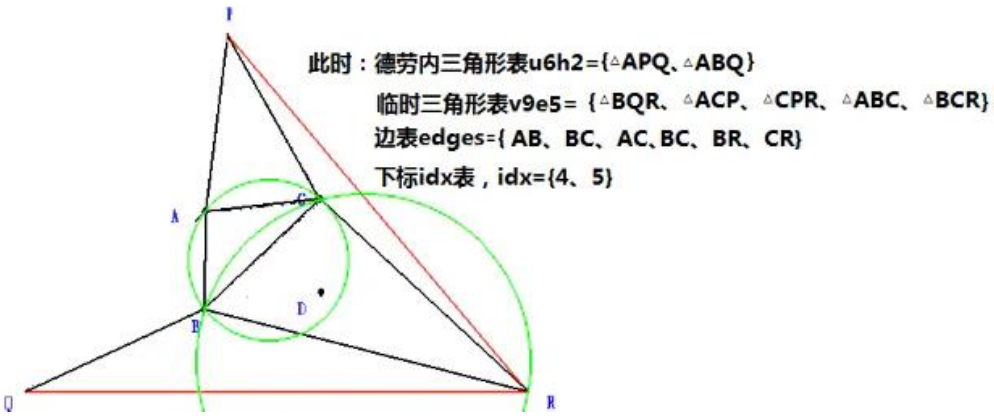
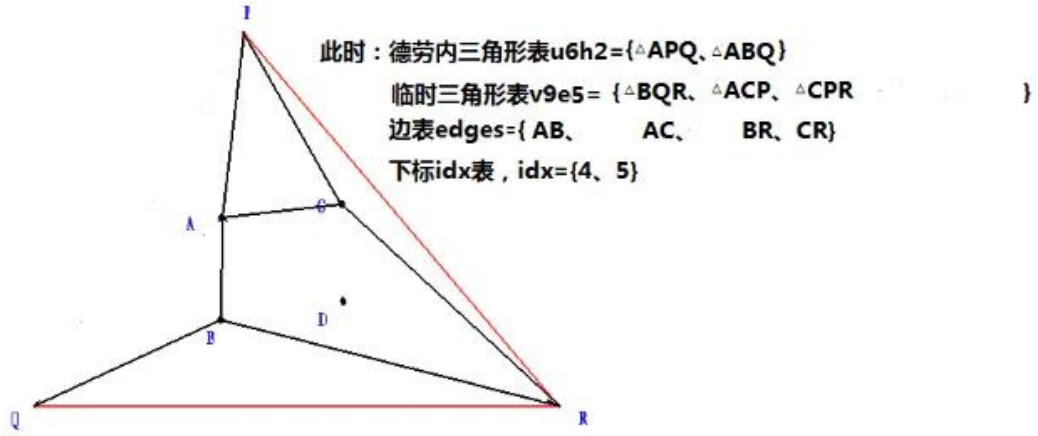


Fig4. 第二个点引入后的最终临时三角形组成

再删除边的时候还存在一种特殊情况，如果非 Delaunay 三角形类的三角形需要删除为边，然后重连为三角形。如果此时有两个三角形或者多个，他们俩共边，此时我们的边集里一定有两个相同的元素。如果我们只是单一地去重，后续生成的三角网络一定会有三家角形交叉，因此对于此类的边，我们需要将两者都删除，这样才能后续继续生成不交叉的三角形。





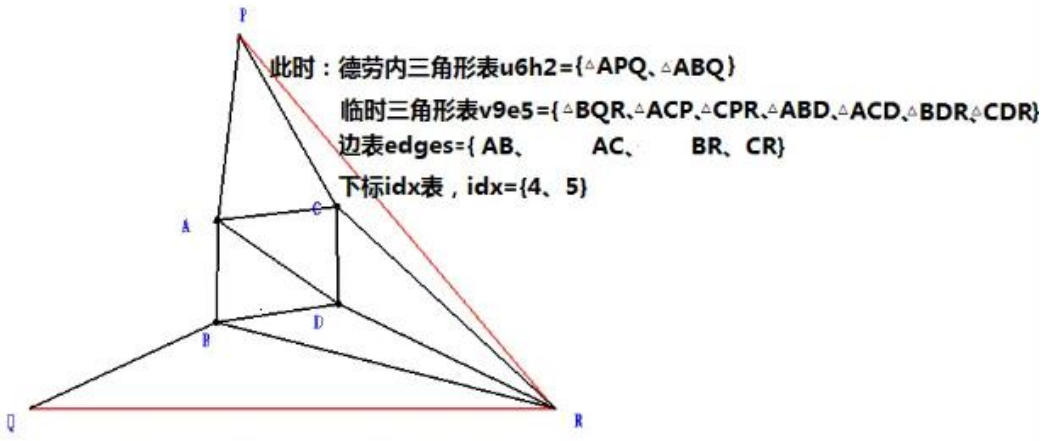


Fig 5,6,7 相邻三角删边重构样例

代码实现如下：

```
"""对边集完全去重，只要元素重复立刻删除！！"""

unique_data = []
for d in temp_edge:
    if d not in unique_data:
        unique_data.append(d)
    else:
        unique_data.remove(d)
temp_edge = unique_data
```

当这个点遍历结束后，我们清空 temp_edge 与 idx_table，然后等待下一轮的插入。我们按照以上过程完成所有采样点的遍历即可。

当完成所有点的遍历后，我们会剩下 delaunay_tri_table 与 temp_tri_table，我们将二者组合起来，即得到了符合 delaunay 三角要求的集合。由于我们的超级三角形的点是我们自己加上的，所以我们还需要三角形集合中，包含顶点的三角形都删除，这样最终留下的就是采样点自己构成的 delaunay 三角剖分三角形合集。然后我们使用 triplot 函数画出 delaunay 三角网络，下面是展示效果：

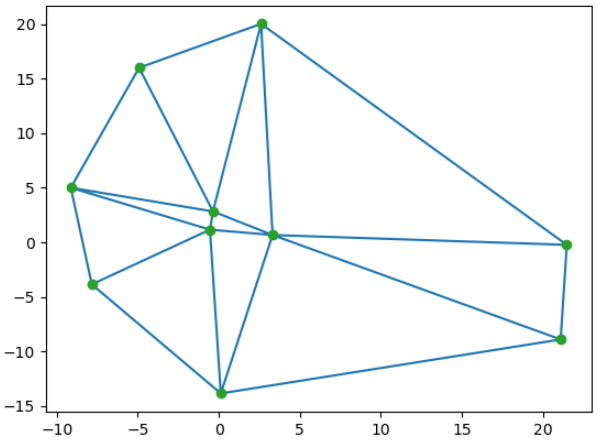


Fig8. 十个随机点的 delaunay 三角剖分图

然后我将我自己生成的随机点的剖分与 python 中集成的 Delaunay 函数的剖分效果做了比较，结果也是完全一样：

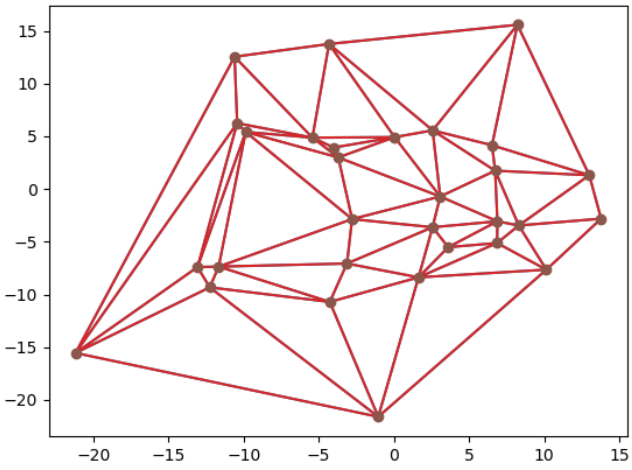


Fig9. 三十个随机点的 delaunay 剖分，二者完全重合

至此我便完成了所有的三角剖分工作。

完整代码实现如下：

```
import numpy as np
import matplotlib.pyplot as plt
import math
from itertools import combinations
from scipy.spatial import Delaunay
```

```
points = np.random.randn(30,2)*10
print("排序前的坐标",'\n',points)
index_sorted = np.argsort(points[:,0])
points_sorted = points[index_sorted]
print("排序后的坐标",'\n',points_sorted)


"""定义超级三角形顶点的坐标"""
x_mean = np.mean(points_sorted[:,0])
y_mean = np.mean(points_sorted[:,1])
x_max = np.max(points_sorted[:,0])
y_max = np.max(points_sorted[:,1])
x_min = np.min(points_sorted[:,0])
y_min = np.min(points_sorted[:,1])
print("各点 x，y 的基础数据信息",'\n',x_max, y_max, x_min, y_min)


#规定超级三角形的内切圆的圆心坐标以及内切圆半径
xc = x_min + (x_max-x_min)/2
yc = y_min + (y_max-y_min)/2
r = max(abs((x_max-x_min)/2), abs((y_max-y_min)/2)) + (x_max-x_min) #
保证囊括所有的点，并且点不在超级三角形的边上

#求超级三角形的三点坐标
def calc_incenter(xc, yc, r):
    R = r / math.tan(math.pi/6)
    x1, y1 = xc - R, yc - r
    x2, y2 = xc, yc + r / math.sin(math.pi/6)
    x3, y3 = xc + R, yc - r
    return np.array([x1, y1]), np.array([x2, y2]), np.array([x3, y3])

super_vertice = np.zeros([3,2])
A,B,C = calc_incenter(xc, yc, r)
print(A,B,C)
super_vertice[0,:] = A
super_vertice[1,:] = B
super_vertice[2,:] = C
print(super_vertice)


"""将 super_vertice 与生成点坐标整合到一个数组里"""
final_array = np.insert(points_sorted,0,super_vertice,axis=0)
```

```
print(final_array)
print(final_array.dtype)


"""求三点围城的三角形的外接圆圆心以及外接圆半径"""

def circumcenter(tri):
    a = tri[0]
    b = tri[1]
    c = tri[2]

    D = 2*(a[0]*(b[1]-c[1])+b[0]*(c[1]-a[1])+c[0]*(a[1]-b[1]))
    Ux = 1/D*((a[0]**2+a[1]**2)*(b[1]-c[1])+(b[0]**2+b[1]**2)*(c[1]-
a[1])+(c[0]**2+c[1]**2)*(a[1]-b[1]))
    Uy = 1/D*((a[0]**2+a[1]**2)*(c[0]-b[0])+(b[0]**2+b[1]**2)*(a[0]-
c[0])+(c[0]**2+c[1]**2)*(b[0]-a[0]))
    U = np.array([Ux, Uy])

    R = np.linalg.norm(U-a)
    return U, R



"""主题程序，生成三角网络"""

delaunay_tri_table = [] #记录完整的符合要求的 delaunay 三角形索引坐标
temp_tri_table = []  #临时三角形列表
temp_edge = [] #需要删除的边表
idx_table = []  #需要删除的三角形列表对应的索引位置
centers = []  #每个三角形的外接圆圆心坐标
radius = []  #每个三角形外接圆对应的半径
idx_point = 3
for dx, dy in points_sorted:
    if dx == x_min:
        temp_tri_table.append(set([0,1,idx_point]))  #插入的三角形使用集合
进行表示，因为集合可以自动去重，并且可以进行排序
        temp_tri_table.append(set([0,2,idx_point]))  #同时这里也符合整体的
数据结构：列表+集合形式
        temp_tri_table.append(set([2,1,idx_point]))
        idx_point += 1
        continue


    """计算每个三角形的圆心坐标以及半径,并且在每个大 for 循环结束时，都要将其重新
```

```
清空"""
    for temp_tri_idx in temp_tri_table:
        vertices_per_tri = final_array[list(temp_tri_idx)]
        U, R = circumcenter(vertices_per_tri)
        centers.append(U)
        radius.append(R)
        #vertices_per_tri.clear()

    """引入新的顶点进行各列表迭代"""
    for i, r in enumerate(radius):
        distance = np.sqrt((centers[i][0] - dx)**2 + (centers[i][1] -
dy)**2)
        print(distance)
        if distance > r:
            if centers[i][0] + r < dx:
                delaunay_tri_table.append(temp_tri_table[i]) #添加
delaunay 三角
                idx_table.append(i) #后面将 delaunay 三角形从临时三角形表里删
去
        else:
            idx_table.append(i)
            new_set = set()
            # 生成所有包含两个元素的组合，并将每个组合转换成元组并加入新集合中
            for combo in combinations(temp_tri_table[i], 2):
                new_set.add(tuple(combo))
            print(new_set)
            new_list = [set(t) for t in new_set]
            print(new_list)
            temp_edge.extend(new_list)

    """对边集完全去重，只要元素重复立刻删除！！"""

    print(temp_edge)

    unique_data = []
    for d in temp_edge:
        if d not in unique_data:
            unique_data.append(d)
        else:
            unique_data.remove(d)
    temp_edge = unique_data


    #这里我一开始写错了，我以为是去重而不是完全去重
```

```
    #temp_edge_new0= set(tuple(sorted(s)) for s in temp_edge)
    #temp_edge_new1 = [set(t) for t in temp_edge_new0]
    #temp_edge = temp_edge_new1



    #temp_edge = remove_duplicates(temp_edge)

    """删除不符合要求的临时三角形"""
    temp_tri_table_remove = [temp_tri_table[t] for t in idx_table]
    set1 = set(frozenset(i) for i in temp_tri_table)
    set2 = set(frozenset(i) for i in temp_tri_table_remove)
    result = [set(i) for i in set1.difference(set2)]  #对两个集合求差集
    temp_tri_table = result

    """将临时边集与新点重新构成三角形，并更新到临时三角形列表里"""
    new_temp_tri = [{idx_point, *d} for d in temp_edge]
    temp_tri_table.extend(new_temp_tri)

    """清空临时边表与临时索引表"""
    temp_edge = []
    idx_table = []
    if dx == x_max:
        break
    centers = []
    radius = []
    idx_point += 1




#合并临时三角形列表与 delaunay 表
for s in delaunay_tri_table:
    temp_tri_table.append(s)
temp_tri_table = np.array(temp_tri_table)

#将里面所有基本元素转换为列表
temp_tri_list = []
for s in temp_tri_table:
    temp_tri_list.append(list(s))

#将列表转换为 ndarray 矩阵方便进行判断
temp_tri_array = np.array(temp_tri_list)
print(temp_tri_array)
```

```
temp_tri_array[temp_tri_array < 3] = 0

#构造布尔数组，记录每行是否都大于零
positive_rows = np.all(temp_tri_array > 0, axis=1)

#使用布尔索引提取符合条件的行
result = temp_tri_array[positive_rows]

"""绘图"""
#使用 sicpy 中集成好的 delaunay 函数进行绘图并相互比较
tri = Delaunay(points_sorted)
plt.triplot(points_sorted[:,0], points_sorted[:,1], tri.simplices)
plt.plot(points_sorted[:,0], points_sorted[:,1], '*')

#plt.show()
#以下是使用自己的结果进行绘制比较
plt.triplot(final_array[:,0], final_array[:,1], result)
plt.plot(points_sorted[:,0], points_sorted[:,1], 'o')
plt.show()
print("hello world")
print(temp_tri_table)
```