# 空间分析算法

# 实习报告

**班级：**

**姓名：**

**学号：**

**提交日期：**

|  |  |  |  |
| --- | --- | --- | --- |
| 实习时间： | 17周一周 | 实习地点： | J14-404等 |
| 实习评语：  指导教师签名：评阅时间： | | | |

# 实验任务和要求：

运用熟悉的编程语言（C#、C++、Python、Java、Matlab等）进行如下算法的设计与开发，要求程序运行稳定、运算结果可靠，并以图形化界面实现结果输出（实验报告中出现的内容这里不作要求，请删除）。

1. 空间实体量测算法设计

（1）请实现两个空间实体间的距离计算：点点距离、点线距离、点面距离、线线距离、线面距离、面面距离；

（2）请实现球面上两点间的距离计算；

（3）请实现面状实体的周长、面积、中心、重心、质心的计算，以及紧凑指数等形态参数的计算；

（4）请实现DEM表面的表面积以及指定高度的体积计算；

（5）请实现现状地物的曲率和弯曲度计算；

1. 线状数据的压缩算法

请运用道格拉斯普克算法进行线目标要素的数据压缩算法设计，以图形化方法实现压缩前、后的线状要素。

1. 不规则三角网（TIN）生成算法

基于离散点数据，运用Delaunay三角网生成准则，使用三角网生长法或逐点插入算法进行 TIN的生成。

1. 累计表面生成及应用算法设计

基于一幅50\*50的网格数据（每个格点值代表通过该网格的花费），随机设定一个网格作为起点，计算该起点到达其它网格的最小花费，进而生成累计表面矩阵。

1. 矢量数据叠加分析算法

（1）点到曲线最短距离计算；

（2）点在多边形内外的判定计算；（实验已做）

（3）曲线与曲线的求交运算；

（4）曲线与面的求交运算；

（5）面面的求交运算。

1. 栅格数据叠加分析算法

（1）进行栅格数据的局部变换、邻域变换、分带变换、全局变换等计算；

（2）基于地图代数，进行两幅栅格数据间的叠加分析计算。

1. 矢量数据的缓冲区分析计算

（1）进行单点、多点的缓冲区生成；（实验已做）

（2）分别基于角平分线法和凸角圆弧法进行线状要素的缓冲区生成；（实验已做）

（3）进行面状要素的缓冲区生成。

1. 基于DEM的地形特征提取算法

以某区域的DEM数据为基础，进行多种地形特征提取的算法设计，主要包括：地形因子计算（坡度、坡向、表面积、体积、坡度坡向变率、曲率、粗糙度、凸凹系数）、地形特征点的提取（山顶点、凹陷点、脊点、谷点、马鞍点、平地点）以及流长、汇水量、流域盆地等的计算网络分析中的最小代价生成树算法。

9. 网络分析中的最小代价生成树算法

基于一幅加权无向网络图，运用破回路算法，进行最小代价生成树的生成，并进行图形表达。

# 开发环境及系统运行环境

## 开发环境

Python 3.6.6

## 运行环境

运行平台：Visual Studio Code



# 算法原理及实现步骤

## 空间实体量测算法设计

1. # 计算面积

def GetShapeArea(self,points\_x=[],points\_y=[]):

area = 0 # 计算面积

def GetShapeArea(self,points\_x=[],points\_y=[]):

area = 0.0

for i in range(len(points\_x)):

if i + 1 <= len(points\_x) - 1:

area += points\_x[i] \* points\_y[i + 1] - points\_x[i + 1] \* points\_y[i]

elif i + 1 > len(points\_x) - 1:

area += points\_x[i] \* points\_y[0] - points\_x[0] \* points\_y[i]

area = area/2.0

return abs(area)

# 显示面积

def ShowShapeArea(self):

for i in range(self.shapeNum):

area = self.GetShapeArea(self.Points\_x\_shape[i],self.Points\_y\_shape[i])

print("图形%d面积: %f" % (i+1,abs(area)))

result = "图形" + str(i+1) + "面积：" + str(abs(area)) + '\n'

self.text\_result.insert('insert',result)

1. # 计算周长

def GetShapeLength(self,points\_x=[],points\_y=[]):

length = 0

for i in range(len(points\_x)):

if i<len(points\_x):

L = math.sqrt((points\_x[i]-points\_x[i-1])\*(points\_x[i]-points\_x[i-1])+(points\_y[i]-points\_y[i-1])\*(points\_y[i]-points\_y[i-1]))

length += L

elif i==len(points\_x):

L = math.sqrt((points\_x[0]-points\_x[i-1])\*(points\_x[0]-points\_x[i-1])+(points\_y[0]-points\_y[i-1])\*(points\_y[0]-points\_y[i-1]))

length += L

return abs(length)

# 显示周长

def ShowShapeLength(self):

for i in range(self.shapeNum):

length = self.GetShapeLength(self.Points\_x\_shape[i],self.Points\_y\_shape[i])

print("图形%d周长: %f" % (i+1,abs(length)))

result = "图形" + str(i+1) + "周长：" + str(abs(length)) + '\n'

self.text\_result.insert('insert',result)

1. # 计算重心

def GetShapeCG(self,points\_x=[],points\_y=[]):

area = self.GetShapeArea(points\_x,points\_y)

x, y = 0.0, 0.0 #重心点坐标

for i in range(len(points\_x)):

lat = points\_x[i] # weidu

lng = points\_y[i] # jingdu

if i == 0:

lat1 = points\_x[-1]

lng1 = points\_y[-1]

else:

lat1 = points\_x[i - 1]

lng1 = points\_y[i - 1]

fg = (lat \* lng1 - lng \* lat1) / 2.0

area += fg

x += fg \* (lat + lat1) / 3.0

y += fg \* (lng + lng1) / 3.0

x = x / area # x坐标

y = y / area # y坐标

return x,y

# 显示重心

def ShowShapeCG(self):

for i in range(self.shapeNum):

x,y = self.GetShapeCG(self.Points\_x\_shape[i],self.Points\_y\_shape[i])

print("图形%d重心：(%f,%f)"%(i+1,x,y))

result = "图形" + str(i+1) + "重心：(" + str(x) + ',' + str(y) + ')\n'

self.text\_result.insert('insert',result).0

for i in range(len(points\_x)):

if i + 1 <= len(points\_x) - 1:

area += points\_x[i] \* points\_y[i + 1] - points\_x[i + 1] \* points\_y[i]

elif i + 1 > len(points\_x) - 1:

area += points\_x[i] \* points\_y[0] - points\_x[0] \* points\_y[i]

area = area/2.0

return abs(area)

1. # 点与点之间的距离

def DistancePointandPoint(self):

distance = math.sqrt((self.Points\_x\_point[0]-self.Points\_x\_point[1])\*\*2 + (self.Points\_y\_point[0]-self.Points\_y\_point[1])\*\*2)

s = "点与点之间的距离：" + str(distance) + '\n'

print(s)

self.text\_result.insert('insert',s)

1. # 点与线之间的距离

def DistancePointandLine(self):

k,b = 0.0,0.0

for each in self.Points\_line:

k = math.sqrt((each[0]-each[2])\*\*2 + (each[1]-each[3])\*\*2)#斜率

b = each[1] - k\*each[0]#常数b

A,B,C = k,-1,b

distance = (abs(A\*self.Points\_x\_point[0]+B\*self.Points\_y\_point[0]+C))/math.sqrt(A\*A + B\*B)

s = "点与线之间的距离：" + str(distance) + '\n'

print(s)

self.text\_result.insert('insert',s)

1. # 点与面之间的距离

def DistancePointandShape(self):

x,y = self.GetShapeCG(self.Points\_x\_shape[0],self.Points\_y\_shape[0])#重心x,y坐标值

distance = math.sqrt((self.Points\_x\_point[0]-x)\*\*2 + (self.Points\_y\_point[0]-y)\*\*2)

s = "点与面之间的距离：" + str(distance) + '\n'

print(s)

self.text\_result.insert('insert',s)

1. # 线与面之间的距离

def DistanceLineandShape(self):

k,b = 0.0,0.0

for each in self.Points\_line:

k = math.sqrt((each[0]-each[2])\*\*2 + (each[1]-each[3])\*\*2)#斜率

b = each[1] - k\*each[0]#常数b

A,B,C = k,-1,b

x,y = self.GetShapeCG(self.Points\_x\_shape[0],self.Points\_y\_shape[0])#重心x,y坐标值

distance = (abs(A\*x+B\*y+C))/math.sqrt(A\*A + B\*B)

s = "线与面之间的距离：" + str(distance) + '\n'

print(s)

self.text\_result.insert('insert',s)

1. # 面与面之间的距离

def DistanceShapeandShape(self):

x1,y1 = self.GetShapeCG(self.Points\_x\_shape[0],self.Points\_y\_shape[0])#重心x1,y1坐标值

x2,y2 = self.GetShapeCG(self.Points\_x\_shape[1],self.Points\_y\_shape[1])#重心x2,y2坐标值

distance = math.sqrt((x1-x2)\*\*2 + (y1-y2)\*\*2)

s = "面与面之间的距离：" + str(distance) + '\n'

print(s)

self.text\_result.insert('insert',s)

## 不规则三角网（TIN）生成算法

1. # 获取DEM所有点的数据

def GetDEMAllData():

time1 = time.time()

getDemData()

global X\_min,X\_max,Y\_min,Y\_max,Z\_insert,z\_x,z\_y

X\_min = int(min(x\_konw))

X\_max = int(max(x\_konw)) + 1

Y\_min = int(min(y\_konw))

Y\_max = int(max(y\_konw)) + 1

for i in range(X\_min, X\_max):

X\_insert.append(i)

print('插值点X坐标输入完成！')

for j in range(Y\_min, Y\_max):

Y\_insert.append(j)

print('插入点Y坐标输入完成！')

all\_num = (X\_max - X\_min) \* (Y\_max - Y\_min)

key = 1

for i in range(X\_min, X\_max):

for j in range(Y\_min, Y\_max):

Z\_insert.append(interpolation(i, j, lst\_surface[0:need\_num]))

print("共%d,正计算第%d个" % (all\_num, key))

key += 1

print('插入点Z值计算完成！')

time2 = time.time()

print("用时：%f s" % (time2-time1))

z\_x = X\_max - X\_min

z\_y = Y\_max - Y\_min

print(z\_x)

print(z\_y)

1. # 三维表面图

def IDWdraw3dsurface():

print('正在绘制...')

IDW\_draw\_3d\_surface(X\_insert, Y\_insert, Z\_insert, z\_x, z\_y)

print('绘制完成！')

def IDW\_draw\_3d\_surface(x, y, z, z\_x, z\_y):

fig = plt.figure()

ax = Axes3D(fig)

# Plot the surface.

x = np.array(x)

y = np.array(y)

z = np.array(z)

x,y = np.meshgrid(x,y)

z = z.reshape(z\_y,z\_x)# 顺序是(y的范围,x的范围)，不要反了

surf = ax.plot\_surface(x, y, z,cmap='rainbow')

# Add a color bar which maps values to colors.

fig.colorbar(surf, shrink=0.5, aspect=5)

ax.set\_xlabel('X') # 设置x坐标轴

ax.set\_ylabel('Y') # 设置y坐标轴

ax.set\_zlabel('Z') # 设置z坐标轴

plt.show()

1. #绘制二维DEM

def Drawgrid2dDEM():

npgrid=dem.readfile(z\_y,z\_x,Z\_insert)

pre=npgrid

npgrid=dem.AddRound(npgrid)

dx,dy=dem.Cacdxdy(npgrid,22.5,22.5)

slope,arf=dem.CacSlopAsp(dx,dy)

dem.np.savetxt("slope.csv",slope,delimiter=",")

Drawgrid(judge=0,A=pre,strs="bone")

## 矢量数据的缓冲区分析

1. # 点缓冲区

def BufferPoints(self):

size = int(self.entry\_input3.get())#缓冲区大小

for i in range(len(self.Points\_x\_point)):

x1,y1 = int(self.Points\_x\_point[i])-size,int(self.Points\_y\_point[i])-size

x2,y2 = int(self.Points\_x\_point[i])+size,int(self.Points\_y\_point[i])+size

self.drawpad.create\_oval(x1,y1,x2,y2,fill="",outline="blue")

self.text\_result.insert('insert',"点缓冲区建立成功！\n")

1. # 线缓冲区

def BufferLines(self):

size = int(self.entry\_input3.get())#缓冲区大小

line\_k = []#存放直线斜率

buffer\_line\_up = []#缓冲区上边

buffer\_line\_down = []#缓冲区下边

for i in range(self.lineNum):

line\_k.append(self.GetLineSlope(self.Points\_x\_line[i],self.Points\_y\_line[i]))

print(line\_k)

i = 0

for k in line\_k:

if(k>0):

x1 = self.Points\_x\_line[i][0]+size\*math.sin(math.atan(k))

y1 = self.Points\_y\_line[i][0]-size\*math.cos(math.atan(k))

x2 = self.Points\_x\_line[i][1]+size\*math.sin(math.atan(k))

y2 = self.Points\_y\_line[i][1]-size\*math.cos(math.atan(k))

buffer = [x1,y1,x2,y2]

buffer\_line\_up.append(buffer)

else:

k = -k

x1 = self.Points\_x\_line[i][0]-size\*math.sin(math.atan(k))

y1 = self.Points\_y\_line[i][0]-size\*math.cos(math.atan(k))

x2 = self.Points\_x\_line[i][1]-size\*math.sin(math.atan(k))

y2 = self.Points\_y\_line[i][1]-size\*math.cos(math.atan(k))

buffer = [x1,y1,x2,y2]

buffer\_line\_up.append(buffer)

i += 1

i = 0

for k in line\_k:

if(k>0):

x3 = self.Points\_x\_line[i][0]-size\*math.sin(math.atan(k))

y3 = self.Points\_y\_line[i][0]+size\*math.cos(math.atan(k))

x4 = self.Points\_x\_line[i][1]-size\*math.sin(math.atan(k))

y4 = self.Points\_y\_line[i][1]+size\*math.cos(math.atan(k))

buffer = [x4,y4,x3,y3]

buffer\_line\_down.append(buffer)

else:

k = -k

x3 = self.Points\_x\_line[i][0]+size\*math.sin(math.atan(k))

y3 = self.Points\_y\_line[i][0]+size\*math.cos(math.atan(k))

x4 = self.Points\_x\_line[i][1]+size\*math.sin(math.atan(k))

y4 = self.Points\_y\_line[i][1]+size\*math.cos(math.atan(k))

buffer = [x4,y4,x3,y3]

buffer\_line\_down.append(buffer)

i += 1

print(buffer\_line\_up)

print(buffer\_line\_down)

for i in range(len(buffer\_line\_up)):

points = buffer\_line\_up[i] + buffer\_line\_down[i]

self.drawpad.create\_polygon(points,fill="",outline="blue")

self.text\_result.insert('insert','线缓冲区建立成功！\n')

1. # 面缓冲区

def BufferShapes(self):

size = int(self.entry\_input3.get())#缓冲区大小

x1,x4 = 200-size,200-size

x2,x3 = 500+size,500+size

y1,y2 = 200-size,200-size

y3,y4 = 500+size,500+size

p = [x1,y1,x2,y2,x3,y3,x4,y4]

self.drawpad.create\_polygon(p,fill="",outline="red")

self.text\_result.insert('insert','面缓冲区建立成功！\n')

## 基于DEM的地形特征提取算法

1. #绘制坡度图

def DrawgridSlope():

npgrid=dem.readfile(z\_y,z\_x,Z\_insert)

npgrid=dem.AddRound(npgrid)

dx,dy=dem.Cacdxdy(npgrid,22.5,22.5)

slope,arf=dem.CacSlopAsp(dx,dy)

dem.np.savetxt("slope.csv",slope,delimiter=",")

Drawgrid(judge=0,A=slope,strs="rainbow")

1. #绘制坡向图

def Drawgrid2dDEMAspectOfSlope():

npgrid=dem.readfile(z\_y,z\_x,Z\_insert)

npgrid=dem.AddRound(npgrid)

dx,dy=dem.Cacdxdy(npgrid,22.5,22.5)

slope,arf=dem.CacSlopAsp(dx,dy)

dem.np.savetxt("slope.csv",slope,delimiter=",")

Drawgrid(judge=0,A=arf)

## 网络分析中的最小代价生成树算法

1. # 得到有权无向图初始数据

def GetMSTdata(self):

self.Clear()

s = ""

data = []

s,data = MSTdata()

result = []#存放相邻点

size = len(data)#树节点个数

print(data)

for i in range(size):

for j in range(len(data[i])):

k = data[i][j]

if(k!=0 and k!=9999):

t = [i,j]

result.append(t)

print(result)

for i in range(len(result)):

line = []

line += self.TreeData[result[i][0]]

line += self.TreeData[result[i][1]]

print(line)

self.drawpad.create\_line(line,fill="blue")

self.text\_result.insert('insert',"有权无向图绘制完成！\n")

self.text\_result.insert('insert',s)

1. # MSTkruskal算法

def ShowMSTkruskal(self):

self.Clear()

kruskal = []

kruskal = MSTkruskal()

result = []#存放相邻点

print(kruskal)

for each in kruskal:

result.append(each[0:2])

print(result)

for i in range(len(result)):

line = []

line += self.TreeData[result[i][0]]

line += self.TreeData[result[i][1]]

print(line)

self.drawpad.create\_line(line,fill="blue")

self.text\_result.insert('insert',"kruskal算法MST绘制完成！\n")

1. # MSTprim算法

def ShowMSTprim(self):

self.Clear()

prim = []

prim = MSTprim()

result = []#存放相邻点

print(prim)

for each in prim:

result.append(each[0:2])

print(result)

for i in range(len(result)):

line = []

line += self.TreeData[result[i][0]]

line += self.TreeData[result[i][1]]

print(line)

self.drawpad.create\_line(line,fill="blue")

self.text\_result.insert('insert',"prims算法MST绘制完成！\n")

## 绘制Koch曲线

1. # 绘制koch曲线的数据

def DrawkochData(self):

self.kochLength,self.kochNum = int(self.entry\_input1.get()),int(self.entry\_input2.get())

print(self.kochLength)

print(self.kochNum)

1. # 开始绘制koch曲线

def StartDrawKoch(self):

DrawKoch(self.kochLength,self.kochNum)

1. def koch(size, n):

if n == 0:

turtle.fd(size)

else:

for angle in [0, 60, -120, 60]:

turtle.left(angle)

koch(size/3, n-1)

def DrawKoch(length, num):

turtle.setup(800, 400)

turtle.speed(0) # 控制绘制速度

turtle.penup()

turtle.goto(-300, -50)

turtle.pendown()

turtle.pensize(2)

koch(length, num) # 0阶科赫曲线长度，阶数

turtle.hideturtle()

## 绘制空间实体

1. # 得到点图形鼠标左键坐标

def PointGetPoints(self,event):

self.p\_point.append(event.x)

self.p\_point.append(event.y)

self.p\_x\_point.append(event.x)

self.p\_y\_point.append(event.y)

self.pointNum += 1

print(event.x,event.y)

# 开始绘制点

def StartDrawPoint(self,event):

self.Points\_point += self.p\_point# 将临时坐标值列表存入总列表

self.Points\_x\_point += self.p\_x\_point# 将临时坐标x值列表存入总列表

self.Points\_y\_point += self.p\_y\_point# 将临时坐标y值列表存入总列表

# x1,y1 = int(self.Points\_x\_point[self.pointNum-1][0])-1,int(self.Points\_y\_point[self.pointNum-1][0])-1

# x2,y2 = int(self.Points\_x\_point[self.pointNum-1][0])+1,int(self.Points\_y\_point[self.pointNum-1][0])+1

# self.drawpad.create\_oval(x1,y1,x2,y2,outline="red")

for i in range(self.pointNum):

x1,y1 = int(self.Points\_x\_point[i])-3,int(self.Points\_y\_point[i])-3

x2,y2 = int(self.Points\_x\_point[i])+3,int(self.Points\_y\_point[i])+3

self.drawpad.create\_oval(x1,y1,x2,y2,fill="red",outline="red")

print(self.Points\_x\_point)

print(self.Points\_y\_point)

self.p\_point = [] # 清空临时列表

self.p\_x\_point = []# 清空临时列表

self.p\_y\_point = []# 清空临时列表

# 准备绘制点

def drawPoint(self):

root.bind("<Button-1>",self.PointGetPoints)

root.bind("<Button-2>",self.PointUnbindLeft)

root.bind("<Button-3>",self.StartDrawPoint)

1. # 得到多边形鼠标左键坐标

def ShapeGetPoints(self,event):

self.p\_shape.append(event.x)

self.p\_shape.append(event.y)

self.p\_x\_shape.append(event.x)

self.p\_y\_shape.append(event.y)

print(event.x,event.y)

# 开始绘制多边形

def StartDrawShape(self,event):

self.shapeNum += 1

self.Points\_shape.append(self.p\_shape)# 将临时坐标值列表存入总列表

self.Points\_x\_shape.append(self.p\_x\_shape)# 将临时坐标x值列表存入总列表

self.Points\_y\_shape.append(self.p\_y\_shape)# 将临时坐标y值列表存入总列表

# self.p.clear()# 清空临时列表(IndexError: tuple index out of range,不能放在前面，原因未知。现在清空，上一句函数的Points也会被清空。)

self.drawpad.create\_polygon(self.Points\_shape[self.shapeNum-1],fill="",outline="black")

print(self.shapeNum)

# 不要使用clear()方法清空，会导致将总列表的数值也清空，原因可能是函数进程未结束时append(self.p)会与self.p一直关联。

print(self.Points\_shape)

self.p\_shape = [] # 清空临时列表

self.p\_x\_shape = []# 清空临时列表

self.p\_y\_shape = []# 清空临时列表

# 准备绘制多边形

def drawShape(self):

root.bind("<Button-1>",self.ShapeGetPoints)

root.bind("<Button-2>",self.ShapeUnbindLeft)

root.bind("<Button-3>",self.StartDrawShape)

1. # 得到直线鼠标左键坐标

def LineGetPoints(self,event):

self.p\_line.append(event.x)

self.p\_line.append(event.y)

self.p\_x\_line.append(event.x)

self.p\_y\_line.append(event.y)

print(event.x,event.y)

# 开始绘制直线

def StartDrawLine(self,event):

self.lineNum += 1

self.Points\_line.append(self.p\_line)# 将临时坐标值列表存入总列表

self.Points\_x\_line.append(self.p\_x\_line)# 将临时坐标x值列表存入总列表

self.Points\_y\_line.append(self.p\_y\_line)# 将临时坐标y值列表存入总列表

self.drawpad.create\_line(self.Points\_line[self.lineNum-1],fill="red")

self.p\_line = [] # 清空临时列表

self.p\_x\_line = []# 清空临时列表

self.p\_y\_line = []# 清空临时列表

print(self.Points\_line)

# 准备绘制直线

def drawLine(self):

root.bind("<Button-1>",self.LineGetPoints)

root.bind("<Button-2>",self.LineUnbindLeft)

root.bind("<Button-3>",self.StartDrawLine)

1. # 得到椭圆鼠标坐标坐标

def OvalGetPoints(self,event):

self.p\_oval.append(event.x)

self.p\_oval.append(event.y)

self.p\_x\_oval.append(event.x)

self.p\_y\_oval.append(event.y)

print(event.x,event.y)

# 开始绘制椭圆

def StartDrawOval(self,event):

self.ovalNum += 1

self.Points\_oval.append(self.p\_oval)# 将临时坐标值列表存入总列表

self.Points\_x\_oval.append(self.p\_x\_oval)# 将临时坐标x值列表存入总列表

self.Points\_y\_oval.append(self.p\_y\_oval)# 将临时坐标y值列表存入总列表

self.drawpad.create\_oval(self.Points\_oval[self.ovalNum-1],fill="",outline="blue")

self.p\_oval = [] # 清空临时列表

self.p\_x\_oval = []# 清空临时列表

self.p\_y\_oval = []# 清空临时列表

# 准备绘制椭圆

def drawOval(self):

root.bind("<Button-1>",self.OvalGetPoints)

root.bind("<Button-2>",self.OvalUnbindLeft)

root.bind("<Button-3>",self.StartDrawOval)

## 云模型

1. # 二维云模型

def plot\_cloud(self):

plot\_cloud\_model(0, 1, 0.1, 500, moni=True)

1. # 三维云模型

def plot\_2d\_cloud(self):

plot\_2d\_cloud\_model([0, 1], [0.3, 0.3], [0.01, 0.05], 2000)

1. def Cloud\_compute(xl):

'''计算云滴的数字特征'''

xl = np.array(xl)

# S2 = np.var(xl) #用的方差

S2 = np.std(xl) #用的标准差

Ex = np.mean(xl)

En = np.sqrt(np.pi/2) \* np.mean( np.abs(xl-Ex) )

He = np.sqrt( np.abs(S2\*S2 - En\*En) )

return (Ex, En, He)

def plot\_cloud\_model(Ex, En, He, n, title='', grid=False, moni=False):

Y = np.zeros((1, n))

X = np.random.normal(loc=En, scale=He, size=n)

Y = Y[0]

plt.rcParams['font.sans-serif'] = ['SimHei']

# 用来正常显示中文标签

plt.rcParams['axes.unicode\_minus'] = False

# 用来正常显示负号

fig = plt.figure(0)

ax = fig.add\_subplot(111)

for i in range(n):

Enn = X[i]

X[i] = np.random.normal(loc=Ex, scale=np.abs(Enn), size=1)

Y[i] = np.exp(-(X[i]-Ex)\*(X[i]-Ex)/(2\*Enn\*Enn))

ax.scatter(X[i], Y[i], s=10, alpha=0.5, c='r', marker='o')

if title == '':

title = '期望:%.2f,熵:%.2f,超熵:%.2f,云滴数:%d' % (Ex, En, He, n)

ax.set\_title(title)

ax.set\_xlabel('指标值')

ax.set\_ylabel('确定度')

ax.grid(True)

if moni:

print (Cloud\_compute(X))

plt.show()

def plot\_2d\_cloud\_model(Ex, En, He, n, title='', grid=False):

Y = np.zeros((1, n))

X0 = np.random.normal(loc=En[0], scale=He[0], size=n)

Y = Y[0]

X1 = np.random.normal(loc=En[1], scale=He[1], size=n)

plt.rcParams['font.sans-serif'] = ['SimHei']

# 用来正常显示中文标签

plt.rcParams['axes.unicode\_minus'] = False

# 用来正常显示负号

fig = plt.figure(0)

ax = fig.add\_subplot(111, projection='3d')

for i in range(n):

Enn0 = X0[i]

X0[i] = np.random.normal(loc=Ex[0], scale=np.abs(Enn0), size=1)

# Y0[i] = np.exp(-(X0[i]-Ex[0])\*(X0[i]-Ex[0])/(2\*Enn\*Enn))

Enn1 = X1[i]

X1[i] = np.random.normal(loc=Ex[1], scale=np.abs(Enn1), size=1)

Y[i] = np.exp(-(X0[i] - Ex[0]) \* (X0[i] - Ex[0]) / (2 \* Enn0 \* Enn0)-(X1[i] - Ex[1]) \* (X1[i] - Ex[1]) / (2 \* Enn1 \* Enn1))

ax.scatter(X0[i], X1[i], Y[i], s=10, alpha=0.5, c='r', marker='o')

if title == '':

title = '期望:[%.2f,%.2f],熵:[%.2f,%.2f],超熵:[%.2f,%.2f],云滴数:%d' % (Ex[0], Ex[1], En[0], En[1], He[0], He[1], n)

ax.set\_title(title)

ax.set\_xlabel('指标值1')

ax.set\_ylabel('指标值2')

ax.set\_zlabel('确定度')

ax.grid(True)

plt.show()

## K均值聚类

1. # 开始k值聚类

def StartKmeans(self):

ShowKmeans()

self.text\_result.insert('insert','K值聚类计算完成！\n')

INF = 9999999.0

def loadDataSet(fileName, splitChar='\t'):

"""

输入：文件名

输出：数据集

描述：从文件读入数据集

"""

dataSet = []

with open(fileName) as fr:

for line in fr.readlines():

curline = line.strip().split(splitChar)

fltline = list(map(float, curline))

dataSet.append(fltline)

return dataSet

def createDataSet():

"""

输出：数据集

描述：生成数据集

"""

dataSet = [[0.0, 2.0],

[0.0, 0.0],

[1.5, 0.0],

[5.0, 0.0],

[5.0, 2.0]]

return dataSet

def distEclud(vecA, vecB):

"""

输入：向量A, 向量B

输出：两个向量的欧式距离

"""

return sqrt(sum(power(vecA - vecB, 2)))

def randCent(dataSet, k):

"""

输入：数据集, 聚类个数

输出：k个随机质心的矩阵

"""

n = shape(dataSet)[1]

centroids = mat(zeros((k, n)))

for j in range(n):

minJ = min(dataSet[:, j])

rangeJ = float(max(dataSet[:, j]) - minJ)

centroids[:, j] = minJ + rangeJ \* random.rand(k, 1)

return centroids

def kMeans(dataSet, k, distMeans=distEclud, createCent=randCent):

"""

输入：数据集, 聚类个数, 距离计算函数, 生成随机质心函数

输出：质心矩阵, 簇分配和距离矩阵

"""

m = shape(dataSet)[0]

clusterAssment = mat(zeros((m, 2)))

centroids = createCent(dataSet, k)

clusterChanged = True

while clusterChanged:

clusterChanged = False

for i in range(m): # 寻找最近的质心

minDist = INF

minIndex = -1

for j in range(k):

distJI = distMeans(centroids[j, :], dataSet[i, :])

if distJI < minDist:

minDist = distJI

minIndex = j

if clusterAssment[i, 0] != minIndex:

clusterChanged = True

clusterAssment[i, :] = minIndex, minDist\*\*2

for cent in range(k): # 更新质心的位置

ptsInClust = dataSet[nonzero(clusterAssment[:, 0].A == cent)[0]]

centroids[cent, :] = mean(ptsInClust, axis=0)

return centroids, clusterAssment

def plotFeature(dataSet, centroids, clusterAssment):

m = shape(centroids)[0]

fig = plt.figure()

scatterMarkers = ['s', 'o', '^', '8', 'p', 'd', 'v', 'h', '>', '<']

scatterColors = ['blue', 'green', 'yellow', 'purple', 'orange', 'black', 'brown']

ax = fig.add\_subplot(111)

for i in range(m):

ptsInCurCluster = dataSet[nonzero(clusterAssment[:, 0].A == i)[0], :]

markerStyle = scatterMarkers[i % len(scatterMarkers)]

colorSytle = scatterColors[i % len(scatterColors)]

ax.scatter(ptsInCurCluster[:, 0].flatten().A[0], ptsInCurCluster[:, 1].flatten().A[0], marker=markerStyle, c=colorSytle, s=90)

ax.scatter(centroids[:, 0].flatten().A[0], centroids[:, 1].flatten().A[0], marker='+', c='red', s=300)

def main():

#dataSet = loadDataSet('testSet2.txt')

dataSet = loadDataSet('data/788points.txt', splitChar=',')

#dataSet = createDataSet()

dataSet = mat(dataSet)

resultCentroids, clustAssing = kMeans(dataSet, 6)

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print(resultCentroids)

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

plotFeature(dataSet, resultCentroids, clustAssing)

def ShowKmeans():

start = time.clock()

main()

end = time.clock()

print('finish all in %s' % str(end - start))

plt.show()

## 最小代价树

# 得到有权无向图初始数据

def GetMSTdata(self):

self.Clear()

s = ""

data = []

s,data = MSTdata()

result = []#存放相邻点

size = len(data)#树节点个数

print(data)

for i in range(size):

for j in range(len(data[i])):

k = data[i][j]

if(k!=0 and k!=9999):

t = [i,j]

result.append(t)

print(result)

for i in range(len(result)):

line = []

line += self.TreeData[result[i][0]]

line += self.TreeData[result[i][1]]

print(line)

self.drawpad.create\_line(line,fill="blue")

self.text\_result.insert('insert',"有权无向图绘制完成！\n")

self.text\_result.insert('insert',s)

# MSTkruskal算法

def ShowMSTkruskal(self):

self.Clear()

kruskal = []

kruskal = MSTkruskal()

result = []#存放相邻点

print(kruskal)

for each in kruskal:

result.append(each[0:2])

print(result)

for i in range(len(result)):

line = []

line += self.TreeData[result[i][0]]

line += self.TreeData[result[i][1]]

print(line)

self.drawpad.create\_line(line,fill="blue")

self.text\_result.insert('insert',"kruskal算法MST绘制完成！\n")

# MSTprim算法

def ShowMSTprim(self):

self.Clear()

prim = []

prim = MSTprim()

result = []#存放相邻点

print(prim)

for each in prim:

result.append(each[0:2])

print(result)

for i in range(len(result)):

line = []

line += self.TreeData[result[i][0]]

line += self.TreeData[result[i][1]]

print(line)

self.drawpad.create\_line(line,fill="blue")

self.text\_result.insert('insert',"prims算法MST绘制完成！\n")

def MSTdata():

s = ""

s = '邻接矩阵为\n'+str(graph.maps)+'\n'+'节点数据为:'+str(graph.nodenum)+'边数为:'+str(graph.edgenum)+'\n'

return s,graph.maps

def MSTkruskal():

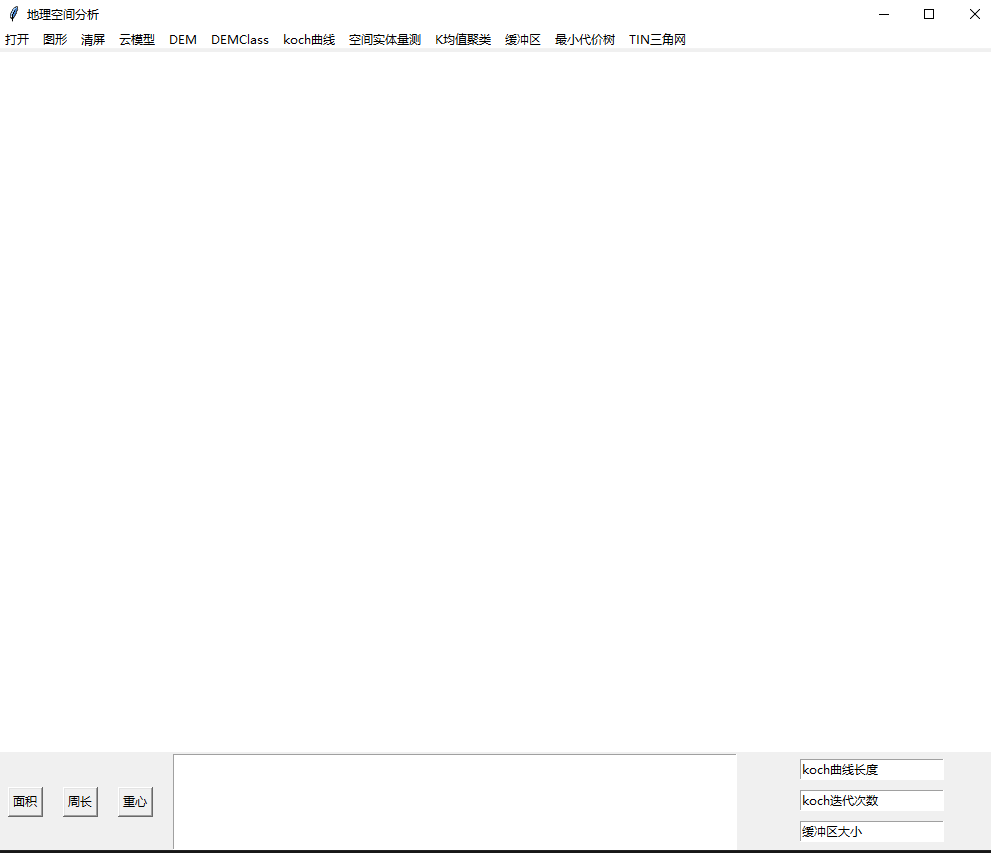
return graph.kruskal()

def MSTprim():

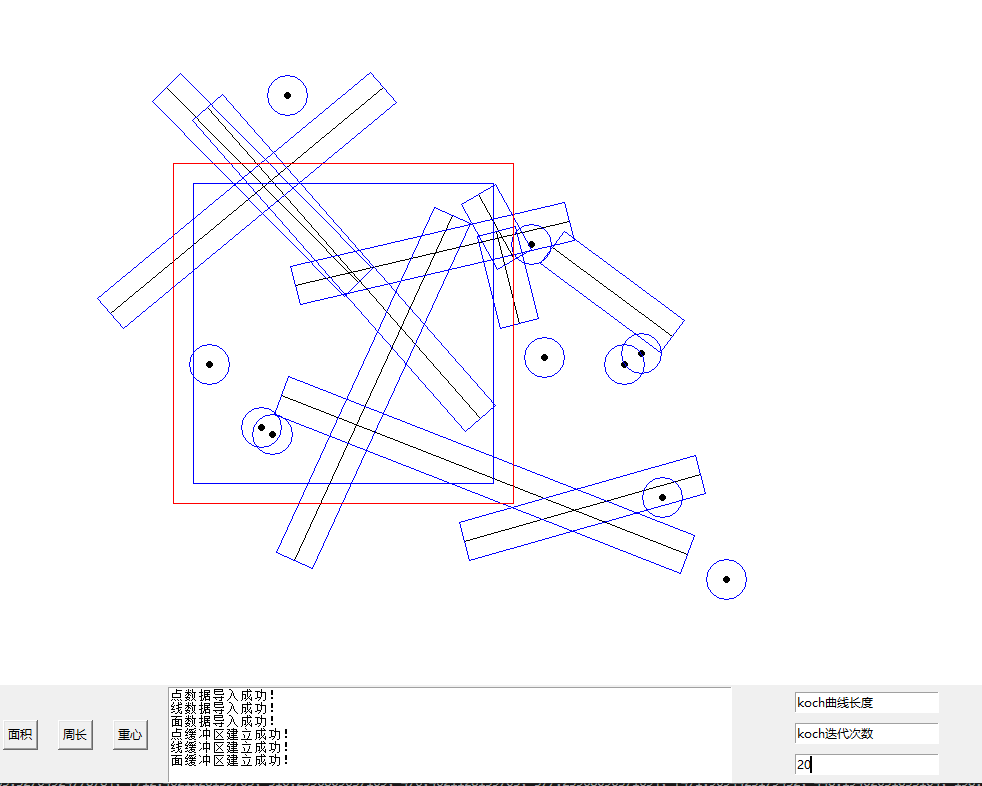
return graph.prim()

# 软件操作步骤及效果

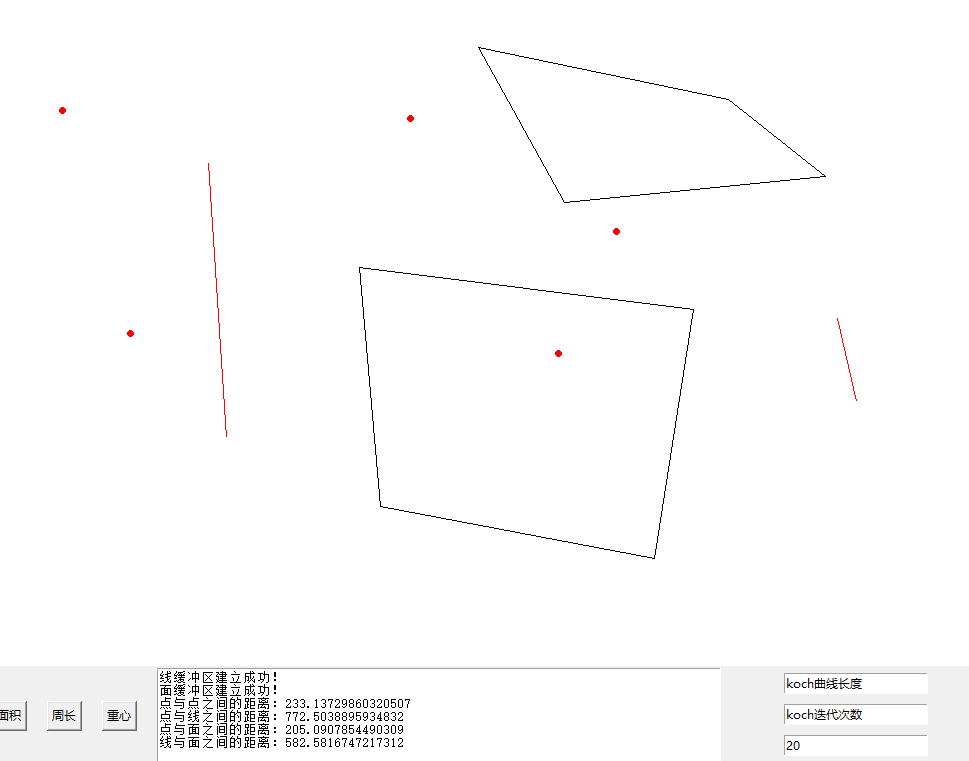
## 主界面



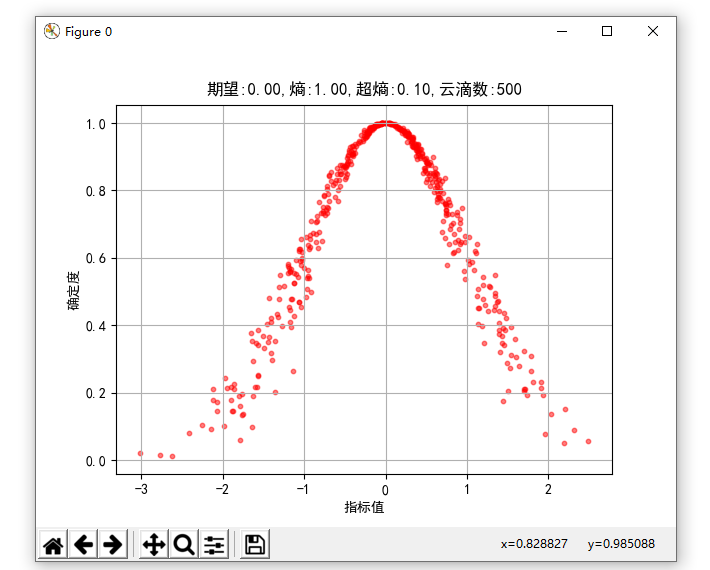
## 加载空间实体数据并绘制缓冲区

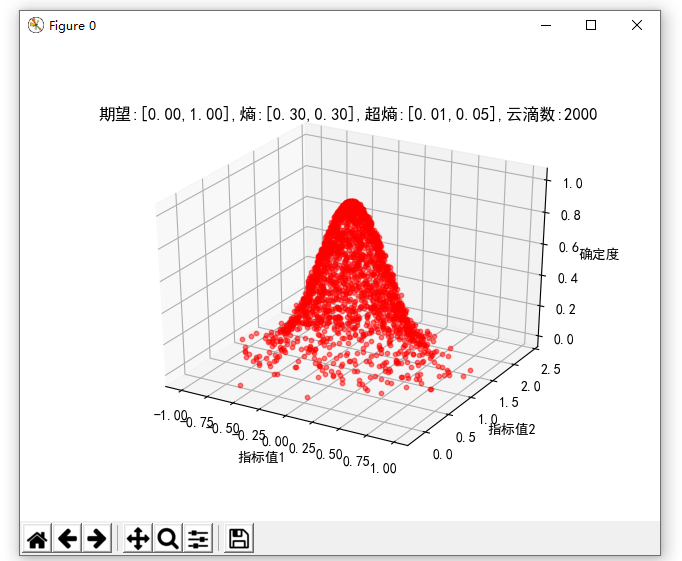


## 空间实体自由绘制

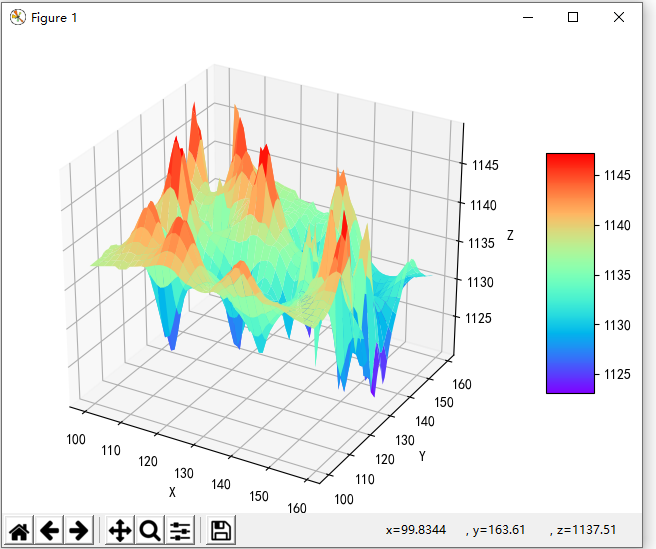


## 建立二维三维云模型

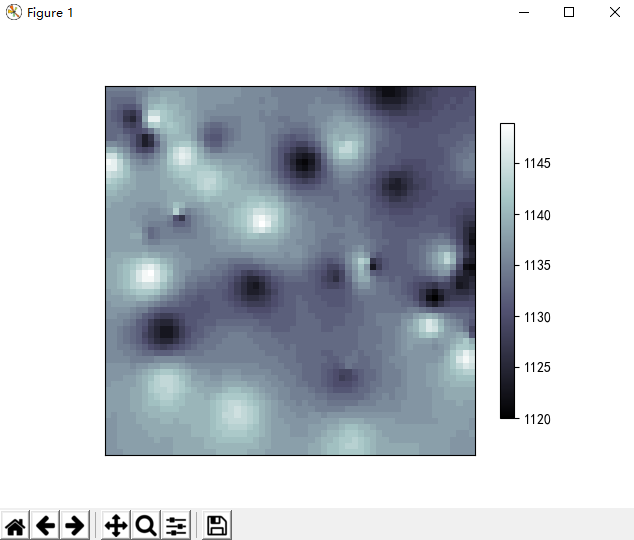


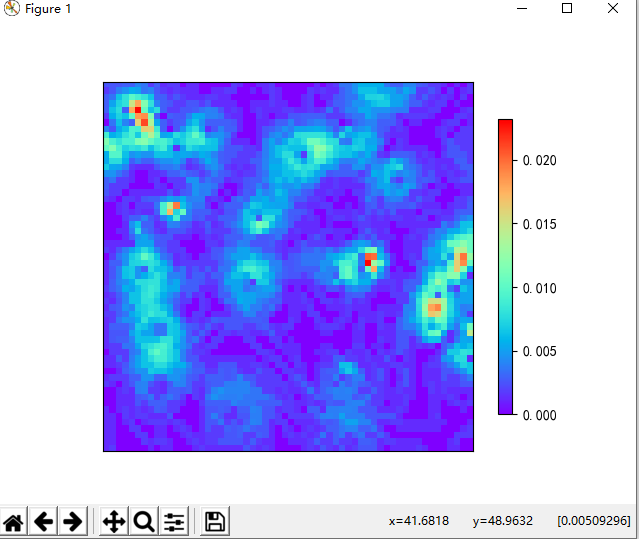


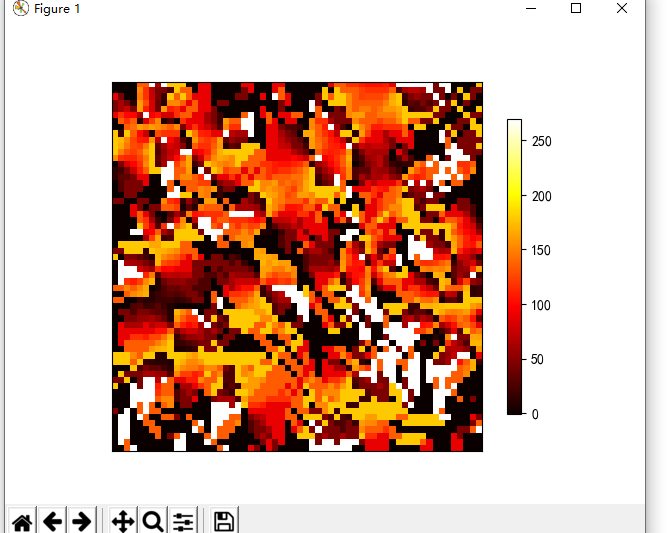
## 绘制DEM曲面图



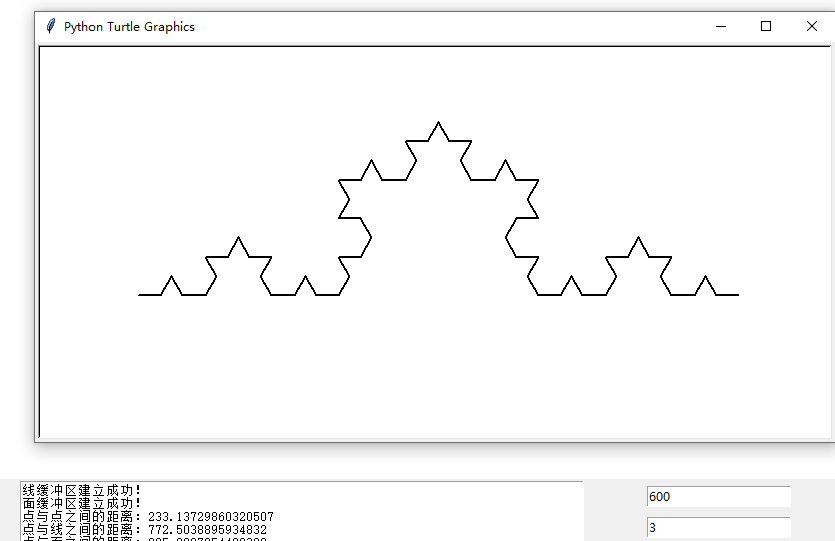
## 地形因子（坡度、坡向和二维DEM）的绘制



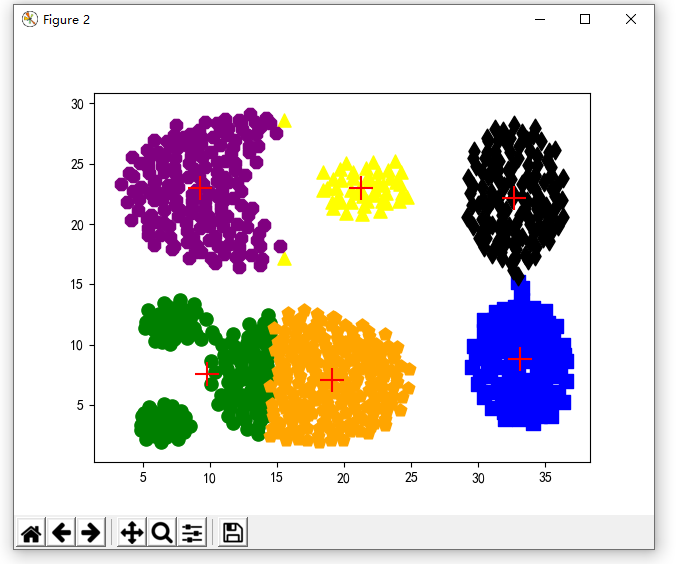




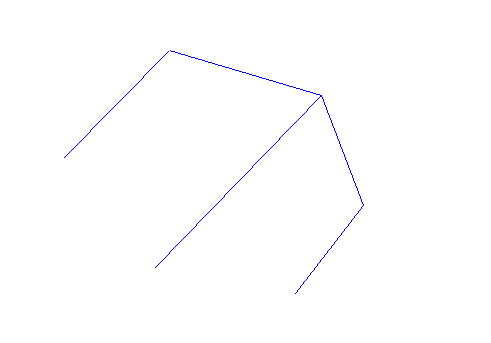
## 绘制koch曲线



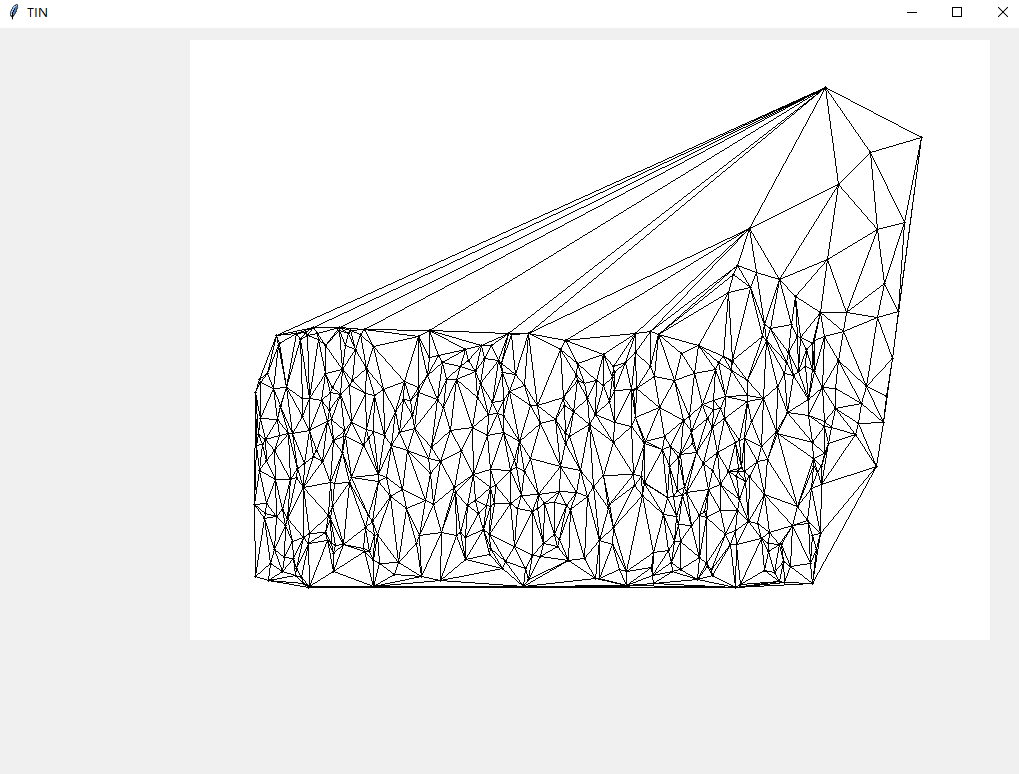
## K均值聚类



## 最小生成素算法



## 三角网生成



# 实习总结

总结可从几个方面考虑，包括：技术提升与收获、解决问题能力提升、合作意识等