- 基本理论

1. 每个个体看作一个点（暂时用二维的点）【多维情况下，每一个维度都可以是对个人某一属性的描述】
2. 点的走向作为个人的发展情况
3. 社会初期随机发展，一定时间后个人发展受朋友影响
4. 前一次的点与后一次的点构成一个线段
5. 在某一时间内，$人_A$ 对应的线段 与$人_B$对应的线段相交，则认定两人相识（成为朋友）
6. 两点之间的距离作为两个人亲密度的度量，前提是成为朋友
7. 在达到一定年龄之后， 人们会选择与自己亲密度最佳（距离最小）的异性朋友作为伴侣（成为恋人）， 一定时间后产生新的人类
8. **未完待续**

- 算法实现

Code: humans

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 24 10:48:48 2015

# Human

@author: Zhang_Jun
"""
import random
import math


class human(object):
    def __init__(self,ID):
        self.ID = ID  # 编号
        self.sex = random.randint(1,2)  # 随机产生性别
        self.x_before = 0 ; self.y_before = 0  # 个人状态描述(前)
        self.x_after = 0  ; self.y_after = 0   # 个人状态描述(后)
        self.vector = [self.x_after - self.x_before , self.y_after -
self.y_before] # 个人生活方向
        self.ability = 1 # 发展潜力，决定了个人发展方向的模长
        self.age = 0  # 个人的年龄
        self.friend = [(ID,self)]   #  个人的所有好友列表 (编号，包含好友所有信息的
对象)
        self.friend_distance_sex = [(ID,0,0)] # 好友的亲密度与性别属性 (编号、亲
密度、性别同性与否[0:同性，1: 异性])
        self.mate = [ID,self,100000]  #  个人的配偶 ,配偶编号 以及 亲密度
        self.stage = 1     # 个人的生存状况 1 为 生 0 为 死
        self.time = 0  # 记录时间流逝，可以构造一个与年龄转换的函数 （自定义）
    def initiate_grow(self,step):  # 社会刚形成期间，每个人随机发展，不受他人影响
        for i in range(step):  # step 为控制发展次数的参数()
            self.time = self.time + 1   # 每step一次，增加一个时间单位
            self.x_before = self.x_after
            angle = random.randint(0,360)/180.0 * math.pi
            self.x_after = self.x_after + self.ability * math.cos(angle)
            self.y_before = self.y_after
            self.y_after = self.y_after + self.ability * math.sin(angle)
            self.vector = [self.x_after - self.x_before , self.y_after -
self.y_before]

    def grow(self,step,coef_friend,coef_random):
        self.time = self.time = self.time + 1
        self.x_before = self.x_after
        self.y_before = self.y_after
        friend_effect  = [i * coef_friend for i in
norm(sum_direction(self.friend))]
        angle = random.randint(0,360)/180.0 * math.pi
        random_effect = [i*coef_random for i in norm([math.cos(angle) ,
math.sin(angle)])]
        direction =[i * self.ability for i in
norm(sum_list(friend_effect,random_effect))]
        [self.x_after,self.y_after] = sum_list([self.x_before,self.y_before]
, direction)
        self.vector = [self.x_after - self.x_before , self.y_after -
self.y_before]
        self.ability = sigmoid(inverse_sigmoid(self.ability) + coef_random *
random.randint(-5,5)+ coef_friend *
inverse_sigmoid(friend_ability(self.friend,self.ability)))

def sigmoid(x):  # 控制个人能力增长的幅度
    return 2.0/(math.exp(-0.1*x)+1)

def inverse_sigmoid(y):
    return 10*math.log(y / (2.0-y) )

def sum_list(a,b):
    return [x+y for x,y in zip(a,b)]

def sum_direction(friend_List):
    direction =[0,0]
    for friend  in friend_List:
        direction = sum_list(direction,friend[1].vector)
    return direction

def norm(L):
    ss = 0 # 平方和
    for i in L:
        ss = ss + i*i
    sss = math.sqrt(ss)
    return [x/sss for x in L]

def friend_ability(friend_List,self_ability):
    fri_ability = [fri[1].ability for fri in friend_List]
    return (1.0*sum(fri_ability)/len(fri_ability) - self_ability)*0.5 + 1



#-------------相遇判断函数-------------------------
# 线段的交点   ->  是否有交点/交点是否在线段上
# -kx+y=b

def meet(human1,human2) :
    delta =1e-10
    P1 = [human1.x_before,human1.y_before]
    P2 = [human1.x_after,human1.y_after]
    P3 = [human2.x_before,human2.y_before]
    P4 = [human2.x_after,human2.y_after]

    if (P2[0]-P1[0])==0:  #  in case of denominator equals 0 (避免分母为0的情
况)
        if (P3[0]-P4[0])==0:
            return 0
        else:
            k2 = 1.0*(P4[1]-P3[1])/(P4[0]-P3[0])
            b2 = -k2*P3[0]+P3[1]
            P = [P1[0],k2*P1[0]+b2]
            if (P[0]-P1[0])*(P[0]-P2[0])<=0 and (P[0]-P3[0])*(P[0]-P4[0])<=0
\
            and (P[1]-P1[1])*(P[1]-P2[1])<=0 and (P[1]-P3[1])*(P[1]-P4[1])
<=0 :
                return 1
            else:
                return 0
    elif (P4[0]-P3[0])==0:  #  in case of denominator equals 0 (避免分母为0的
情况)
        if (P2[0]-P1[0])==0:
            return 0
        else:
            k1 = 1.0*(P2[1]-P1[1])/(P2[0]-P1[0])
            b1 = -k1*P1[0]+P1[1]
            P = [P3[0],k1*P3[0]+b1]
            if (P[0]-P1[0])*(P[0]-P2[0])<=0 and (P[0]-P3[0])*(P[0]-P4[0])<=0
\
            and (P[1]-P1[1])*(P[1]-P2[1])<=0 and (P[1]-P3[1])*(P[1]-P4[1])
<=0 :
                return 1
            else:
                return 0
    else:
        k1 = 1.0 * (P2[1] - P1[1]) / (P2[0] - P1[0])
        k2 = 1.0 * (P4[1] - P3[1]) / (P4[0] - P3[0])
        b1 = -k1 * P1[0] + P1[1]
        b2 = -k2 * P3[0] + P3[1]
        if abs(k1 - k2) < delta:  #  if use k1=k2  will cause singular
matrix 精确度问题
            return 0
        else: # 求解两直线交点
            P=[0,0]
            P[0] = (b1 - b2) / (k2 - k1)
            P[1] = (k2 * b1 - k1 * b2) / (k2-k1)
            # 判断交点是否在线段上
            if (P[0]-P1[0])*(P[0]-P2[0])<=0 and (P[0]-P3[0])*(P[0]-P4[0])<=0
\
            and (P[1]-P1[1])*(P[1]-P2[1])<=0 and (P[1]-P3[1])*(P[1]-P4[1])
<=0 :
                return 1
            else:
                return 0
```

Code: simulate （待注释）

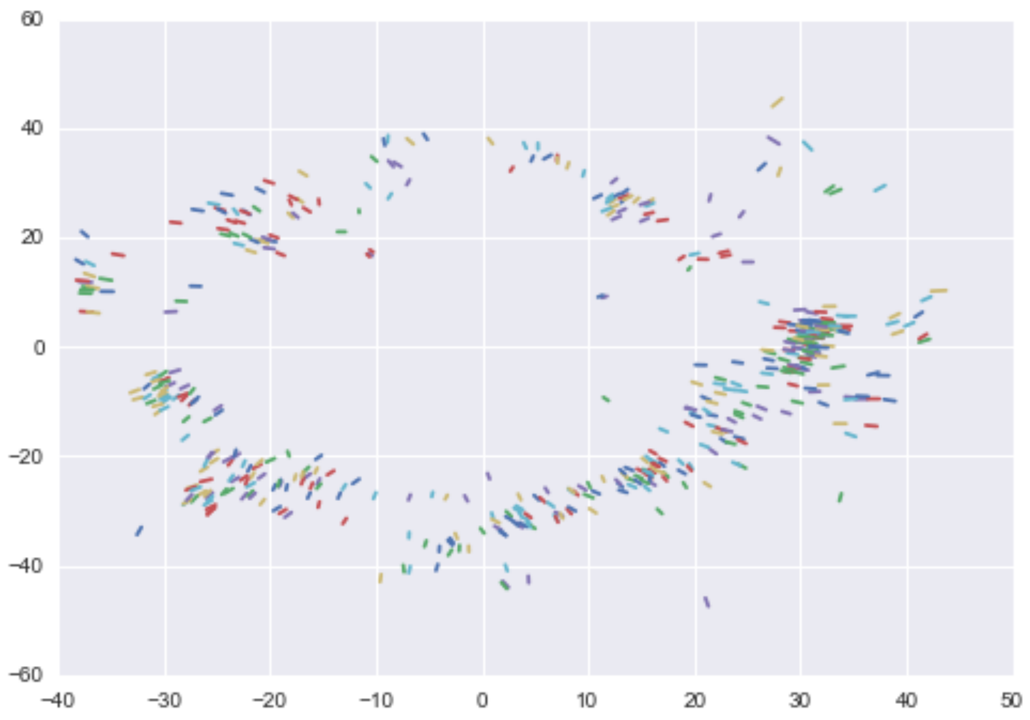
```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 24 11:14:11 2015

@author: Zhang_Jun
"""



import human
import math
import pylab as plt
import numpy as np
import pandas as pd
import seaborn
seaborn.set()



# -------------------------generate humans-----------------

humans = []
human_number = 1000 # number of  initiate humans

for i in range(human_number):
    humans.append(human.human(i))
```

```
#----------------------------------------------------------



#-------------------- the initiate grows of humans---------

Time = 2

for people in humans:
    people.initiate_grow(Time)

#-----------------show the vector of humans--------------------
V = [people.vector for people in humans ]

human_V = np.array(V)
#plt.plot(human_V[:,0],human_V[:,1],'.')
#plt.figure()

#---------------------------------------------
def draw_grow(number):
    Position =
[[people.x_before,people.x_after,people.y_before,people.y_after] for people
in humans]
    X_Position = np.array(Position)[:,:2]
    Y_Position =  np.array(Position)[:,2:]
    for i in range(number):
        plt.plot(X_Position[i],Y_Position[i])

#---------------------------------------------------------

for i in range(10):
    for people in humans:
        people.initiate_grow(1)
    draw_grow(human_number)

#  after initiate grow  , the growth of people should folow some rule



marrage_age_bottom = 20
marrage_age_top = 40



#-------------------------------------meet  -------------
Meet=np.zeros(human_number*human_number).reshape(human_number,human_number)
Meet_people=[]

for t in range(10):
    for people in humans:
        people.grow(1,0.8,0.2)
    #draw_grow(human_number)
    for i in range(human_number):
        for j in range(human_number):
            if Meet[i,j]==0 and i != j:
                Meet[i,j] = human.meet(humans[i],humans[j])
            if Meet[i,j] == 1 and (i,j) not in Meet_people:
                Meet_people.append((i,j))
    #print Meet_people
    #print Meet

    for fri in Meet_people:   # friend in Meet_people
        if (fri[1],humans[fri[1]]) not in humans[fri[0]].friend:
            humans[fri[0]].friend.append((fri[1],humans[fri[1]]))
            distance = math.sqrt( math.pow(humans[fri[1]].x_after \
            - humans[fri[0]].x_after , 2) + math.pow(humans[fri[1]].y_after
\

            - humans[fri[0]].y_after , 2))
            sex_compare = 0 if humans[fri[0]].sex == humans[fri[1]].sex else
1

humans[fri[0]].friend_distance_sex.append((fri[1],distance,sex_compare))
# --------------------------------------------------------
    #draw_grow(human_number)
    have_friend = set([p[0] for p in Meet_people])

    for i in have_friend:
        table1 = humans[i].friend  # 所有的朋友 （编号和对象）
        table2 = humans[i].friend_distance_sex    # 朋友的属性（编号、亲密度、性
别同性与否）
        ID_OBJ = pd.DataFrame(table1,columns=('ID','OBJ'))   # 转化为pandas
格式
        ID_DIS_Sex = pd.DataFrame(table2,columns=('ID','DIS','SEX'))    # 转
化为pandas 格式
        S_ID_DIS_SEX = ID_DIS_Sex.sort(columns='DIS')     # 按照亲密度排列
        S_ID_DIS_HSEX = S_ID_DIS_SEX[S_ID_DIS_SEX.SEX == 1]   #  获取异性列表
        if len(S_ID_DIS_HSEX)>0:
            mate_DIS = S_ID_DIS_HSEX.DIS.values[0]  # 记录与配偶的亲密度
            mate_ID = S_ID_DIS_HSEX.ID.values[0]   # 记录配偶ID
            if mate_DIS < humans[mate_ID].mate[2]:   # A的理想配偶已经有配偶的情
况判定及处理
                humans[humans[mate_ID].mate[0]].mate=
[humans[mate_ID].mate[0],humans[mate_ID].mate[1],100000]
                humans[i].mate = [mate_ID,ID_OBJ[ID_OBJ.ID==mate_ID]
['OBJ'].values[0],mate_DIS]
                humans[mate_ID].mate =[i,humans[i],mate_DIS]



have_mate = [i for i in have_friend if humans[i].mate[0]!=i]
```

```
print len(have_mate)  #  拥有配偶的人数

#------------------show the vector of humans--------------------
plt.figure()
V = [people.vector for people in humans ]

human_V = np.array(V)
plt.plot(human_V[:,0],human_V[:,1],'.')


plt.figure()
#  draw friends
for me in have_friend:
    for number in range(len(humans[me].friend)):
        Position =[humans[me].x_after,humans[me].friend[number]
[1].x_after,humans[me].y_after,humans[me].friend[number][1].y_after]
        plt.plot(Position[:2],Position[2:],'g-',linewidth=0.3)
```

运行以上代码

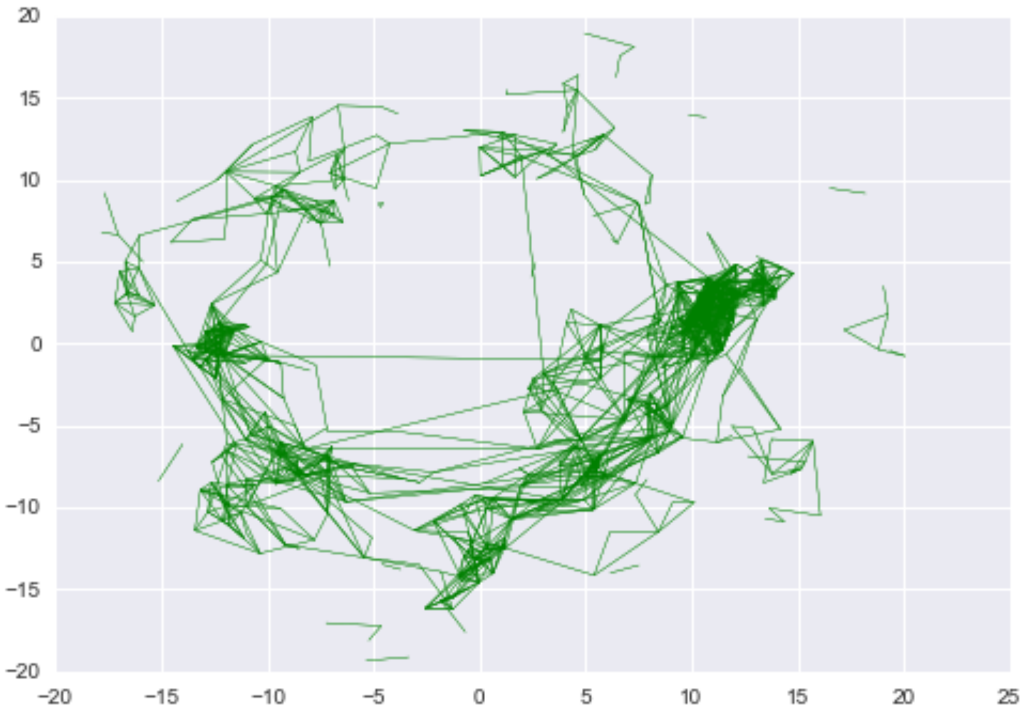
```
500人发展一段时间后的情况图（某一时刻）
```



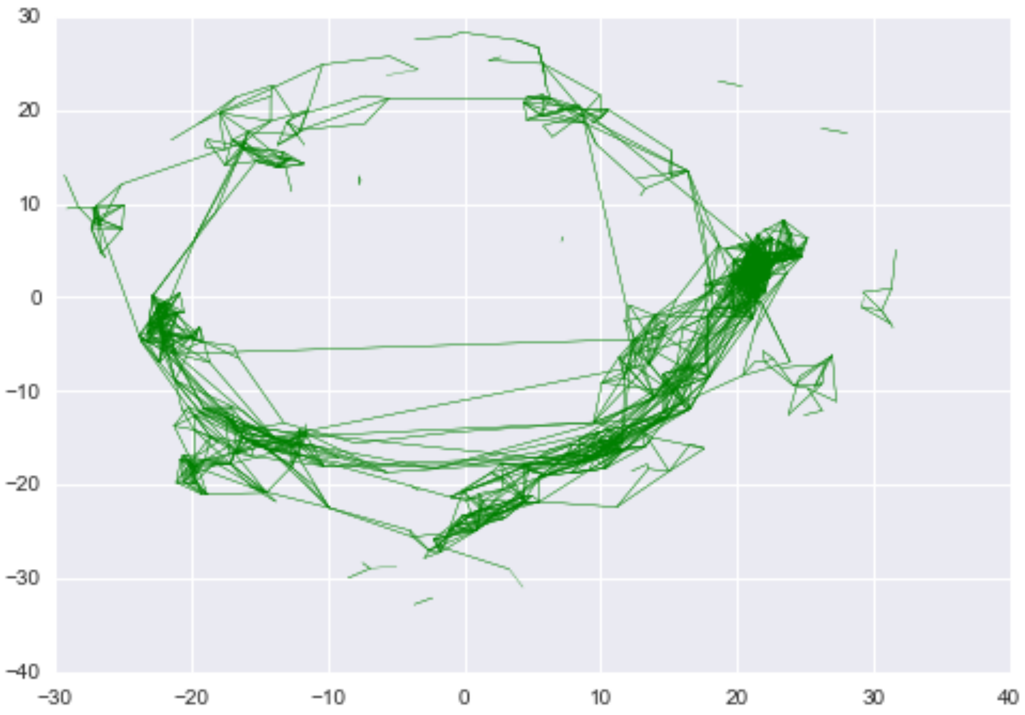
运行后查看变量（人）的情况

```
In [274]: have_mate
Out[274]:
[1,
 2,
 3,
 6,
 8,
 9,
 10,
 11,
 13,
 16,
 17,
 18,
 19,
 ...}

In [277]: have_friend
Out[277]:
{0,
 1,
 2,
 3,
 4,
 5,
 6,
 8,
 9,
 10,
 ...}

In [281]: humans[11].mate
Out[281]: [216, <human.human at 0x3c5a79e8>, 0.12324587994054474]

In [282]: humans[216].mate
Out[282]: [11, <human.human at 0x5fa5ee48>, 0.12324587994054474
```

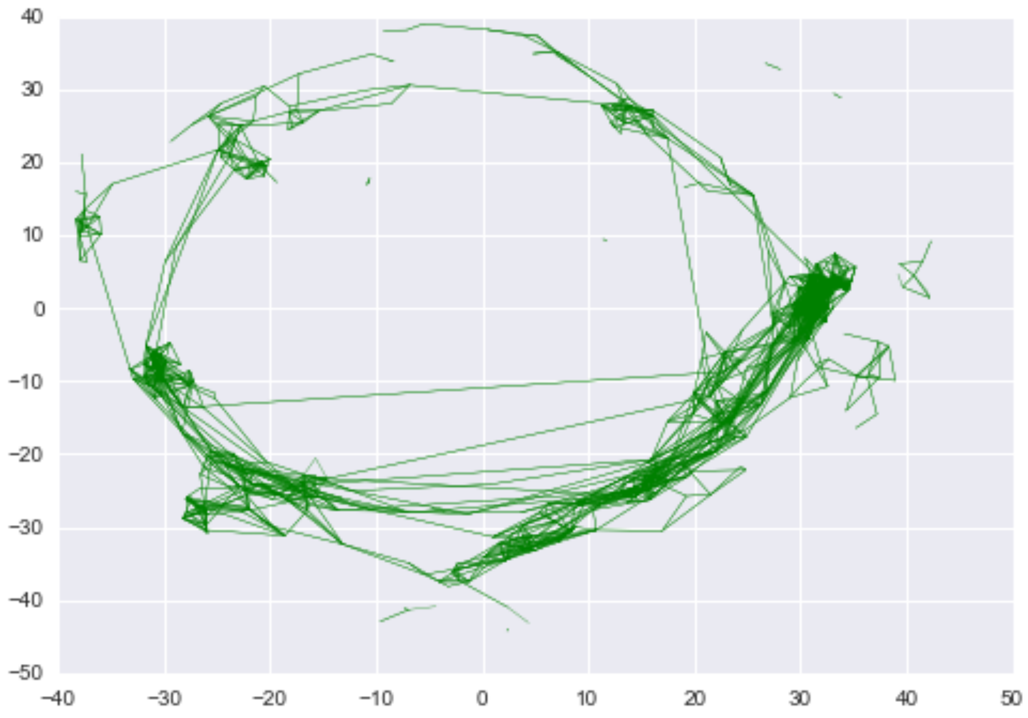
朋友图（多阶段发展）



20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20
25

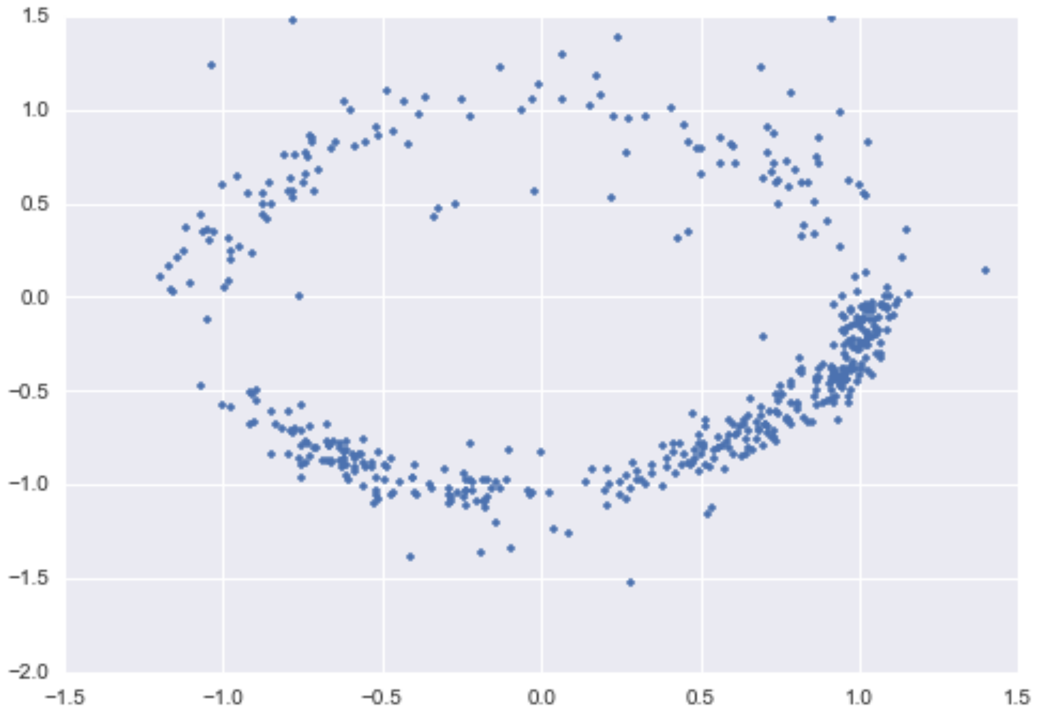


30
20
10
0
−10
−20
−30
−40
−30
−20
−10
0
10
20
30
40



一定时期后形成一定的格局

人们的发展方向分布


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5